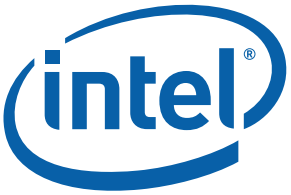

Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

January 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: 252046-045

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

§

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

Nomenclature

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

Summary Tables of Changes

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

Codes Used in Summary Tables

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

Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 1, Volume 1
2	Updates to Chapter 4, Volume 1
3	Updates to Chapter 5, Volume 1
4	Updates to Chapter 8, Volume 1
5	Updates to Chapter 11, Volume 1
6	Updates to Chapter 13, Volume 1
7	Updates to Appendix E, Volume 1
8	Updates to Chapter 1, Volume 2A
9	Updates to Chapter 3, Volume 2A
10	Updates to Chapter 4, Volume 2B
11	Updates to Chapter 1, Volume 3A
12	Updates to Chapter 2, Volume 3A
13	Updates to Chapter 4, Volume 3A
14	Updates to Chapter 6, Volume 3A
15	Updates to Chapter 8, Volume 3A
16	Updates to Chapter 11, Volume 3A
17	Updates to Chapter 17, Volume 3B
18	Updates to Chapter 19, Volume 3B
19	Updates to Chapter 22, Volume 3B
20	Updates to Chapter 29, Volume 3B
21	Updates to Chapter 33, Volume 3C
22	Updates to Chapter 35, Volume 3C

Documentation Changes

1. Updates to Chapter 1, Volume 1

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

...

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

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

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

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

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

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

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

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

The Intel® Atom™ 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 the 3rd generation Intel® Core™ processors are based on the Intel® microarchitecture code name Ivy Bridge and support Intel 64 architecture.

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

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

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

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

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

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

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

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

...

2. Updates to Chapter 4, Volume 1

Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1:* Basic Architecture.

...

4.8.3.2 Normalized and Denormalized Finite Numbers

Non-zero, finite numbers are divided into two classes: normalized and denormalized. The normalized finite numbers comprise all the non-zero finite values that can be encoded in a normalized real number format between zero and ∞. In the single-precision floating-point format shown in Figure 4-12, this group of numbers includes all the numbers with biased exponents ranging from 1 to 254_{10} (unbiased, the exponent range is from -126_{10} to $+127_{10}$).

When floating-point numbers become very close to zero, the normalized-number format can no longer be used to represent the numbers. This is because the range of the exponent is not large enough to compensate for shifting the binary point to the right to eliminate leading zeros.

When the biased exponent is zero, smaller numbers can only be represented by making the integer bit (and perhaps other leading bits) of the significand zero. The numbers in this range are called **denormalized** numbers. The use of leading zeros with denormalized numbers allows smaller numbers to be represented. However, this denormalization may cause a loss of precision (the number of significant bits is reduced by the leading zeros).

When performing normalized floating-point computations, an IA-32 processor normally operates on normalized numbers and produces normalized numbers as results. Denormalized numbers represent an **underflow** condition. The exact conditions are specified in Section , "4.9.1.5 Numeric Underflow Exception (#U)."

A denormalized number is computed through a technique called gradual underflow. Table 4-6 gives an example of gradual underflow in the denormalization process. Here the single-precision format is being used, so the minimum exponent (unbiased) is -126_{10}. The true result in this example requires an exponent of -129_{10} in order to have a normalized number. Since -129_{10} is beyond the allowable exponent range, the result is denormalized by inserting leading zeros until the minimum exponent of -126_{10} is reached.

Table 4-6 Denormalization Process

Operation	Sign	Exponent*	Significand
True Result	0	–129	1.01011100000...00
Denormalize	0	–128	0.10101110000...00
Denormalize	0	–127	0.01010111000...00
Denormalize	0	–126	0.00101011100...00
Denormal Result	0	–126	0.00101011100...00

* Expressed as an unbiased, decimal number.

In the extreme case, all the significant bits are shifted out to the right by leading zeros, creating a zero result.

The Intel 64 and IA-32 architectures deal with denormal values in the following ways:

- It avoids creating denormals by normalizing numbers whenever possible.
- It provides the floating-point underflow exception to permit programmers to detect cases when denormals are created.
- It provides the floating-point denormal-operand exception to permit procedures or programs to detect when denormals are being used as source operands for computations.

...

4.9.1.5 Numeric Underflow Exception (#U)

The processor detects a potential floating-point numeric underflow condition whenever the result of rounding with unbounded exponent (taking into account precision control for x87) is non-zero and tiny; that is, non-zero and less than the smallest possible normalized, finite value that will fit into the destination operand. Table 4-11 shows the threshold range for numeric underflow for each of the floating-point formats (assuming normalized results); underflow occurs when a rounded result falls strictly within the threshold range. The ability to detect and handle underflow is provided to prevent a very small result from propagating through a computation and causing another exception (such as overflow during division) to be generated at a later time. Results which trigger underflow are also potentially less accurate.

Table 4-11 Numeric Underflow (Normalized) Thresholds

Floating-Point Format	Underflow Thresholds*
Single Precision	$\|x\| < 1.0 * 2^{-126}$
Double Precision	$\|x\| < 1.0 * 2^{-1022}$
Double Extended Precision	$\|x\| < 1.0 * 2^{-16382}$

* Where 'x' is the result rounded to destination precision with an unbounded exponent range.

How the processor handles an underflow condition, depends on two related conditions:

- creation of a tiny, non-zero result
- creation of an inexact result; that is, a result that cannot be represented exactly in the destination format

Which of these events causes an underflow exception to be reported and how the processor responds to the exception condition depends on whether the underflow exception is masked:

- **Underflow exception masked —** The underflow exception is reported (the UE flag is set) only when the result is both tiny and inexact. The processor returns a correctly signed result whose magnitude is less than or equal to the smallest positive normal floating-point number to the destination operand, regardless of inexactness.
- **Underflow exception not masked —** The underflow exception is reported when the result is non-zero tiny, regardless of inexactness. The processor leaves the source and destination operands unaltered or stores a biased result in the destination operand (depending whether the underflow exception was generated during an SSE/SSE2/SSE3 floating-point operation or an x87 FPU operation) and invokes a software exception handler.

See the following sections for information regarding the numeric underflow exception when detected while executing x87 FPU instructions or while executing SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.5, "Numeric Underflow Exception (#U)"
- SIMD floating-point exceptions; Section 11.5.2.5, "Numeric Underflow Exception (#U)"

...

3. 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

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
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	Intel 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, TSX, INVPCID	Intel Xeon processor E3-1200 v3 product family; 4th Generation Intel Core processor family.
ADX, RDSEED, CLAC, STAC	Intel Core M processor family; 5th 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 3A & 3B.*

...

5.1.15 BMI1, BMI2

ANDN	Bitwise AND of first source with inverted 2nd source operands.
BEXTR	Contiguous bitwise extract
BLSI	Extract lowest set bit
BLSMSK	Set all lower bits below first set bit to 1
BLSR	Reset lowest set bit
BZHI	Zero high bits starting from specified bit position
LZCNT	Count the number leading zero bits
MULX	Unsigned multiply without affecting arithmetic flags
PDEP	Parallel deposit of bits using a mask
PEXT	Parallel extraction of bits using a mask
RORX	Rotate right without affecting arithmetic flags
SARX	Shift arithmetic right
SHLX	Shift logic left
SHRX	Shift logic right
TZCNT	Count the number trailing zero bits

5.1.15.1 Detection of VEX-encoded GPR Instructions, LZCNT and TZCNT, PREFETCHW

VEX-encoded general-purpose instructions do not operate on any vector registers.

There are separate feature flags for the following subsets of instructions that operate on general purpose registers, and the detection requirements for hardware support are:

CPUID.(EAX=07H, ECX=0H):EBX.BMI1[bit 3]: if 1 indicates the processor supports the first group of advanced bit manipulation extensions (ANDN, BEXTR, BLSI, BLSMSK, BLSR, TZCNT);

CPUID.(EAX=07H, ECX=0H):EBX.BMI2[bit 8]: if 1 indicates the processor supports the second group of advanced bit manipulation extensions (BZHI, MULX, PDEP, PEXT, RORX, SARX, SHLX, SHRX);

CPUID.EAX=80000001H:ECX.LZCNT[bit 5]: if 1 indicates the processor supports the LZCNT instruction.

CPUID.EAX=80000001H:ECX.PREFTEHCHW[bit 8]: if 1 indicates the processor supports the PREFTEHCHW instruction. CPUID.(EAX=07H, ECX=0H):ECX.PREFTEHCHWT1[bit 0]: if 1 indicates the processor supports the PREFTEHCHWT1 instruction.

...

4. 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.

...

8.3.7 Trigonometric Instructions

The following instructions perform four common trigonometric functions:

FSIN	Sine
FCOS	Cosine
FSINCOS	Sine and cosine
FPTAN	Tangent
FPATAN	Arctangent

These instructions operate on the top one or two registers of the x87 FPU register stack and they return their results to the stack. The source operands for the FSIN, FCOS, FSINCOS, and FPTAN instructions must be given in radians; the source operand for the FPATAN instruction is given in rectangular coordinate units.

The FSINCOS instruction returns both the sine and the cosine of a source operand value. It operates faster than executing the FSIN and FCOS instructions in succession.

The FPATAN instruction computes the arctangent of ST(1) divided by ST(0), returning a result in radians. It is useful for converting rectangular coordinates to polar coordinates.

See Section 8.3.8, "Approximation of Pi" and Section 8.3.10, "Transcendental Instruction Accuracy" for information regarding the accuracy of these instructions.

8.3.8 Approximation of Pi

When the argument (source operand) of a trigonometric function is within the domain of the function, the argument is automatically reduced by the appropriate multiple of 2π through the same reduction mechanism used by the FPREM and FPREM1 instructions. The internal value of π (3.1415926…) that the x87 FPU uses for argument

reduction and other computations, denoted as Pi in the expression below. The numerical value of Pi can be written as:

$Pi = 0.f * 2^2$

where the fraction f is expressed in binary form as:

f = C90FDAA2 2168C234 C

(The spaces in the fraction above indicate 32-bit boundaries.)

The internal approximation Pi of the value π has a 66 significant bits. Since the exact value of π represented in binary has the next 3 bits equal to 0, it means that Pi is the value of π rounded to nearest-even to 68 bits, and also the value of π rounded toward zero (truncated) to 69 bits.

However, accuracy problems may arise because this relatively short finite approximation Pi of the number π is used for calculating the reduced argument of the trigonometric function approximations in the implementations of FSIN, FCOS, FSINCOS, and FPTAN. Alternately, this means that FSIN (x), FCOS (x), and FPTAN (x) are really approximating the mathematical functions sin (x * π /Pi), cos (x * π / Pi), and tan (x * π / Pi), and not exactly sin (x), cos (x), and tan (x). (Note that FSINCOS is the equivalent of FSIN and FCOS combined together). The period of sin (x * π /Pi) for example is 2* Pi, and not 2π.

See also Section 8.3.10, "Transcendental Instruction Accuracy" for more information on the accuracy of these functions.

...

8.3.10 Transcendental Instruction Accuracy

New transcendental instruction algorithms were incorporated into the IA-32 architecture beginning with the Pentium processors. These new algorithms (used in transcendental instructions FSIN, FCOS, FSINCOS, FPTAN, FPATAN, F2XM1, FYL2X, and FYL2XP1) allow a higher level of accuracy than was possible in earlier IA-32 processors and x87 math coprocessors. The accuracy of these instructions is measured in terms of **units in the last place** (**ulp**). For a given argument *x*, let *f*(*x*) and *F*(*x*) be the correct and computed (approximate) function values, respectively. The error in ulps is defined to be:

$$error = \left| \frac{f(x) - F(x)}{2^{k-63}} \right|$$

where *k* is an integer such that:

$1 \le 2^{-k} f(x) < 2.$

With the Pentium processor and later IA-32 processors, the worst case error on transcendental functions is less than 1 ulp when rounding to the nearest (even) and less than 1.5 ulps when rounding in other modes. The functions are guaranteed to be monotonic, with respect to the input operands, throughout the domain supported by the instruction.

However, for FSIN, FCOS, FSINCOS, and FPTAN which approximate periodic trigonometric functions, the previous statement about maximum ulp errors is true only when these instructions are applied to reduced argument (see Section 8.3.8, "Approximation of Pi"). This is due to the fact that only 66 significant bits are retained in the finite approximation Pi of the number π (3.1415926…), used internally for calculating the reduced argument in FSIN, FCOS, FSINCOS, and FPTAN. This approximation of π is not always sufficiently accurate for good argument reduction.

For single precision, the argument of FSIN, FCOS, FSINCOS, and FPTAN must exceed 200,000 radians in order for the error of the result to exceed 1 ulp when rounding to the nearest (even), or 1.5 ulps when rounding in other (directed) rounding modes.

For double and double-extended precision, the ulp errors will grow above these thresholds for arguments much smaller in magnitude. The ulp errors increase significantly when the argument approaches the value of π (or Pi) for FSIN, and when it approaches π/2(or Pi/2) for FCOS, FSINCOS, and FPTAN.

For all three IEEE precisions supported (32-bit single precision, 64-bit double precision, and 80-bit double-extended precision), applying FSIN, FCOS, FSINCOS, or FPTAN to arguments larger than a certain value can lead to reduced arguments (calculated internally) that are inaccurate or even very inaccurate in some cases. This leads to equally inaccurate approximations of the corresponding mathematical functions. In particular, arguments that are close to certain values will lose significance when reduced, leading to increased relative (and ulp) errors in the results of FSIN, FCOS, FSINCOS, and FPTAN. These values are:

- any non-zero multiple of π for FSIN,
- any multiple of π, plus π/2 for FCOS, and
- any non-zero multiple of π/2 for FSINCOS and FPTAN.

If the arguments passed to FSIN, FCOS, FSINCOS, and FPTAN are not close to these values then even the finite approximation Pi of π used internally for argument reduction will allow for results that have good accuracy.

Therefore, in order to avoid such errors it is recommended to perform accurate argument reduction in software, and to apply FSIN, FCOS, FSINCOS, and FPTAN to reduced arguments only. Regardless of the target precision (single, double, or double-extended), it is safe to reduce the argument to a value smaller in absolute value than about 3π/4 for FSIN, and smaller than about 3π/8 for FCOS, FSINCOS, and FPTAN.

The thresholds shown above are not exact. For example, accuracy measurements show that the double-extended precision result of FSIN will not have errors larger than 0.72 ulp for $|x| < 2.82$ (so $|x| < 3\pi/4$ will ensure good accuracy, as $3\pi/4 < 2.82$). On the same interval, double precision results from FSIN will have errors at most slightly larger than 0.5 ulp, and single precision results will be correctly rounded in the vast majority of cases.

Likewise, the double-extended precision result of FCOS will not have errors larger than 0.82 ulp for $|x| < 1.31$ (so $|x| < 3\pi/8$ will ensure good accuracy, as $3\pi/8 < 1.31$). On the same interval, double precision results from FCOS will have errors at most slightly larger than 0.5 ulp, and single precision results will be correctly rounded in the vast majority of cases.

FSINCOS behaves similarly to FSIN and FCOS, combined as a pair.

Finally, the double-extended precision result of FPTAN will not have errors larger than 0.78 ulp for $|x| < 1.25$ (so $|x| < 3\pi/8$ will ensure good accuracy, as $3\pi/8 < 1.25$). On the same interval, double precision results from FPTAN will have errors at most slightly larger than 0.5 ulp, and single precision results will be correctly rounded in the vast majority of cases.

A recommended alternative in order to avoid the accuracy issues that might be caused by FSIN, FCOS, FSINCOS, and FPTAN, is to use good quality mathematical library implementations of the sin, cos, sincos, and tan functions, for example those from the Intel® Math Library available in the Intel® Compiler.

The instructions FYL2X and FYL2XP1 are two operand instructions and are guaranteed to be within 1 ulp only when y equals 1. When y is not equal to 1, the maximum ulp error is always within 1.35 ulps in round to nearest mode. (For the two operand functions, monotonicity was proved by holding one of the operands constant.)

...

8.5.5 Numeric Underflow Exception (#U)

The x87 FPU detects a potential floating-point numeric underflow condition whenever the result of an arithmetic instruction is non-zero and tiny; that is, the magnitude of the rounded result with unbounded exponent is non-zero and less than the smallest possible normalized, finite value that will fit into the floating-point format of the destination operand. (See Section 4.9.1.5, "Numeric Underflow Exception (#U)," for additional information about the numeric underflow exception.)

Like numeric overflow, numeric underflow can occur on arithmetic operations where the result is stored in an x87 FPU data register. It can also occur on store floating-point operations (with the FST and FSTP instructions), where a within-range value in a data register is stored in memory in the smaller single-precision or double-precision floating-point formats. A numeric underflow exception cannot occur when storing values in an integer or BCD integer format, because a value with magnitude less than 1 is always rounded to an integral value of 0 or 1, depending on the rounding mode in effect.

The flag (UE) for the numeric-underflow exception is bit 4 of the x87 FPU status word, and the mask bit (UM) is bit 4 of the x87 FPU control word.

When a numeric-underflow condition occurs and the exception is masked, the x87 FPU performs the operation described in Section 4.9.1.5, "Numeric Underflow Exception (#U)."

When the exception is not masked, the action of the x87 FPU depends on whether the instruction is supposed to store the result in a memory location or on the x87 FPU resister stack.

- **Destination is a memory location —** (Can occur only with a store instruction.) The UE flag is set and a software exception handler is invoked (see Section 8.7, "Handling x87 FPU Exceptions in Software"). The top-of-stack pointer (TOP) and source and destination operands remain unchanged, and no result is stored in memory.
  Because the data in the stack is in double extended-precision format, the exception handler has the option either of re-exchanges the store instruction after proper adjustment of the operand or of rounding the significand on the stack to the destination's precision as the standard requires. The exception handler should ultimately store a value into the destination location in memory if the program is to continue.
- **Destination is the register stack —** The significand of the result is rounded according to current settings of the precision and rounding control bits in the x87 FPU control word and the exponent of the result is adjusted by multiplying it by 2^{24576}. (For instructions not affected by the precision field, the significand is rounded to double extended precision.) The resulting value is stored in the destination operand. Condition code bit C1 in the x87 FPU status register (acting here as a "round-up bit") is set if the significand was rounded upward and cleared if the result was rounded toward 0. After the result is stored, the UE flag is set and a software exception handler is invoked. The scaling bias value 24,576 is the same as is used for the overflow exception and has the same effect, which is to translate the result as nearly as possible to the middle of the double extended-precision floating-point exponent range.

  When using the FSCALE instruction, massive underflow can occur, where the magnitude of the result is too small to be represented, even with a bias-adjusted exponent. Here, if underflow occurs again after the result has been biased, a properly signed 0 is stored in the destination operand.

...

5. Updates to Chapter 11, Volume 1

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

...

11.5.2.5 Numeric Underflow Exception (#U)

The processor reports a numeric underflow exception whenever the magnitude of the rounded result of an arithmetic instruction, with unbounded exponent, is less than the smallest possible normalized, finite value that will fit in the destination operand and the numeric-underflow exception is not masked. If the numeric underflow exception is masked, both underflow and the inexact-result condition must be detected before numeric underflow is reported. This exception can be generated with the ADDPS, ADDSS, ADDPD, ADDSD, SUBPS, SUBSS, SUBPD, SUBSD, MULPS, MULSS, MULPD, MULSD, DIVPS, DIVSS, DIVPD, DIVSD, CVTPD2PS, CVTSD2SS, ADDSUBPD,

ADDSUBPS, HADDPD, HADDPS, HSUBPD, and HSUBPS instructions. The flag (UE) and mask (UM) bits for the numeric underflow exception are bits 4 and 11, respectively, in the MXCSR register.

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

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

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

...

6. 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.

The XSAVE feature set extends the functionality of the FXSAVE and FXRSTOR instructions (see Section 10.5, "FXSAVE and FXRSTOR Instructions") by supporting the saving and restoring of processor state in addition to the x87 execution environment (**x87 state**) and the registers used by the streaming SIMD extensions (**SSE state**).

The **XSAVE feature set** comprises eight instructions. XGETBV and XSETBV allow software to read and write the extended control register XCR0, which controls the operation of the XSAVE feature set. XSAVE, XSAVEOPT, XSAVEC, and XSAVES are four instructions that save processor state to memory; XRSTOR and XRSTORS are corresponding instructions that load processor state from memory. XGETBV, XSAVE, XSAVEOPT, XSAVEC, and XRSTOR can be executed at any privilege level; XSETBV, XSAVES, and XRSTORS can be executed only if CPL = 0.

The XSAVE feature set organizes the state that manages into **state components**. Operation of the instructions is based on **state-component bitmaps** that have the same format as XCR0: each bit corresponds to a state component. Section 13.1 discusses these state components and bitmaps in more detail.

Section 13.2 describes how the processor enumerates support for the XSAVE feature set and for **XSAVE-enabled features** (those features that require use of the XSAVE feature set for their enabling). Section 13.3 explains how software can enable the XSAVE feature set and XSAVE-enabled features.

The XSAVE feature set allows saving and loading processor state from a region of memory called an **XSAVE area**. Section 13.4 presents details of the XSAVE area and its organization. Each XSAVE-managed state component is associated with a section of the XSAVE area. Section 13.5 describes in detail each of the XSAVE-managed state components.

Section 13.6 through Section 13.11 describe the operation of XSAVE, XRSTOR, XSAVEOPT, XSAVEC, XSAVES, and XRSTORS, respectively.

13.1 XSAVE-SUPPORTED FEATURES AND STATE-COMPONENT BITMAPS

The XSAVE feature set supports the saving and restoring of **state components**, each of which is a discrete set of processor registers (or parts of registers). In general, each such state component corresponds to a particular CPU feature. Such a feature is **XSAVE-supported**. Some XSAVE-supported features use registers in multiple XSAVE-managed state components.

The XSAVE feature set organizes the state components of the XSAVE-supported features using **state-component bitmaps**. A state-component bitmap comprises 64 bits; each bit in such a bitmap corresponds to a single state component. The following bits are defined in state-component bitmaps:

- Bit 0 corresponds to the state component used for the x87 FPU execution environment (**x87 state**). See Section 13.5.1.
- Bit 1 corresponds to the state component used for registers used by the streaming SIMD extensions (**SSE state**). See Section 13.5.2.
- Bit 2 corresponds to the state component used for the additional register state used by the Intel® Advanced Vector Extensions (**AVX state**). See Section 13.5.3.

Other bits in the range 62:3 are not currently defined in state-component bitmaps and are reserved for future expansion. As individual state component is defined within bits 62:3, additional sub-sections are updated within Section 13.5 over time. Bit 63 is used for special functionality in some bitmaps and does not correspond to any state component.

The state component corresponding to bit *i* of state-component bitmaps is called **state component *i***. Thus, x87 state is state component 0; SSE state is state component 1; and AVX state is state component 2.

The XSAVE feature set uses state-component bitmaps in multiple ways. Most of the instructions use an implicit operand (in EDX:EAX), called the **instruction mask**, which is the state-component bitmap that specifies the state components on which the instruction operates.

Extended control register XCR0 contains a state-component bitmap that specifies the state components that software has enabled the full XSAVE feature set to manage. If the bit corresponding to a state component is clear in XCR0, the following instructions in the XSAVE feature set will not operate on that state component, regardless of the value of the instruction mask: XSAVE, XRSTOR, XSAVEOPT, and XSAVEC. Details of the operation of these instructions are given in Section 13.6 through Section 13.9.

The IA32_XSS MSR (index DA0H) contains a state-component bitmap that specifies the state components that software has enabled XSAVES and XRSTORS to manage. If the bit corresponding to a state component is clear in the logical-OR of XCR0 and IA32_XSS (XCR0 | IA32_XSS), XSAVES and XRSTORS will not operate on that state component, regardless of the value of the instruction mask. Details of the operation of these instructions are given in Section 13.10 and Section 13.11.

Some XSAVE-supported features can be used only if XCR0 has been configured so that the features' state components can be managed by the XSAVE feature set. Such state components and features are **XSAVE-enabled**. In general, the processor will not modify (or allow modification of) the registers of a state component of an XSAVE-enabled feature if the bit corresponding to that state component is clear in XCR0. (If software clears such a bit in XCR0, the processor preserves the corresponding state component.) If an XSAVE-enabled feature has not been fully enabled in XCR0, execution of any instruction defined for that feature causes an invalid-opcode exception (#UD).

As will be explained in Section 13.3, the XSAVE feature set is enabled only if CR4.OSXSAVE[bit 18] = 1. If CR4.OSXSAVE = 0, the processor treats XSAVE-enabled state features and their state components as if all bits in XCR0 were clear; the state components cannot be modified and the features' instructions cannot be executed.

The state components for x87 state and for SSE state are XSAVE-managed but the corresponding features are not XSAVE-enabled. Processors allow modification of this state, as well as execution of x87 FPU instructions and SSE instructions, regardless of the value of CR4.OSXSAVE and XCR0.

...

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[1] = 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.
- XCR0[63:3] is reserved. Executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and any bit in EDX or EAX[31:3] is not 0. Bits 63:3 of 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:

- 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.

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

...

13.4.3 Extended Region of an XSAVE Area

The extended region of an XSAVE area starts at byte offset 576 from the area's base address. The size of the extended region is determined by which state components the processor supports and which bits have been set in XCR0 | IA32_XSS (see Section 13.3).

The XSAVE feature set uses the extended area for each state component *i*, where $i \geq 2$. (Currently, the extended region is used only for AVX state, which is state component 2.)

The extended region of the an XSAVE area may have one of two formats. The **standard format** is supported by all processors that support the XSAVE feature set; the **compacted format** is supported by those processors that support the compaction extensions to the XSAVE feature set (see Section 13.2). Bit 63 of the XCOMP_BV field in the XSAVE header (see Section 13.4.2) indicates which format is used.

The following items describe the two possible formats of the extended region:

- **Standard format.** Each state component *i* ($i \geq 2$) is located at the byte offset from the base address of the XSAVE area enumerated in CPUID.(EAX=0DH,ECX=*i*):EBX. (CPUID.(EAX=0DH,ECX=*i*):EAX enumerates the number of bytes required for state component *i*.
- **Compacted format.** Each state component *i* ($i \geq 2$) is located at a byte offset from the base address of the XSAVE area based on the XCOMP_BV field in the XSAVE header:
  - If XCOMP_BV[*i*] = 0, state component *i* is not in the XSAVE area.
  - If XCOMP_BV[*i*] = 1, the following items apply:
    - If XCOMP_BV[*j*] = 0 for every *j*, $2 \leq j < i$, state component *i* is located at a byte offset 576 from the base address of the XSAVE area. (This item applies if *i* is the first bit set in bits 62:2 of the XCOMP_BV; it implies that state component *i* is located at the beginning of the extended region.)
    - Otherwise, let *j*, $2 \leq j < i$, be the greatest value such that XCOMP_BV[*j*] = 1. Then state component *i* is located at a byte offset X from the location of state component *j*, where X is the number of bytes required for state component *j* as enumerated in CPUID.(EAX=0DH,ECX=*j*):EAX. (This item implies that state component *i* immediately follows the preceding state component whose bit is set in XCOMP_BV.)

...

13.5.1 x87 State

Instructions in the XSAVE feature set can manage the same state of the x87 FPU execution environment (**x87 state**) that can be managed using the FXSAVE and FXRSTOR instructions. They organize all x87 state in the legacy region of the XSAVE area (see Section 13.4.1). This region is illustrated in Table 13-1; the x87 state is listed below, along with details of its interactions with the XSAVE feature set:

- Bytes 1:0, 3:2, 7:6. These are used for the x87 FPU Control Word (FCW), the x87 FPU Status Word (FSW), and the x87 FPU Opcode (FOP), respectively.
- Byte 4 is used for an abridged version of the x87 FPU Tag Word (FTW). The following items describe its usage:
  - For each *j*, $0 \leq j \leq 7$, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save a 0 into bit *j* of byte 4 if x87 FPU data register ST*j* has a empty tag; otherwise, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save a 1 into bit *j* of byte 4.
  - For each *j*, $0 \leq j \leq 7$, XRSTOR and XRSTORS establish the tag value for x87 FPU data register ST*j* as follows. If bit *j* of byte 4 is 0, the tag for ST*j* in the tag register for that data register is marked empty (11B); otherwise, the x87 FPU sets the tag for ST*j* based on the value being loaded into that register (see below).
- Bytes 15:8 are used as follows:
  - If the instruction has no REX prefix, or if REX.W = 0:

    - Bytes 11:8 are used for bits 31:0 of the x87 FPU Instruction Pointer Offset (FIP).
    - If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 0, bytes 13:12 are used for x87 FPU Instruction Pointer Selector (FPU CS). Otherwise, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save these bytes as 0000H, and XRSTOR and XRSTORS ignore them.
    - Bytes 15:14 are not used.
  - If the instruction has a REX prefix with REX.W = 1, bytes 15:8 are used for the full 64 bits of FIP.
- Bytes 23:16 are used as follows:
  - If the instruction has no REX prefix, or if REX.W = 0:
    - Bytes 19:16 are used for bits 31:0 of the x87 FPU Data Pointer Offset (FDP).
    - If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 0, bytes 21:20 are used for x87 FPU Data Pointer Selector (FPU DS). Otherwise, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save these bytes as 0000H; and XRSTOR and XRSTORS ignore them.
    - Bytes 23:22 are not used.
  - If the instruction has a REX prefix with REX.W = 1, bytes 23:16 are used for the full 64 bits of FDP.
- Bytes 31:24 are used for SSE state (see Section 13.5.2).
- Bytes 159:32 are used for the registers ST0–ST7 (MM0–MM7). Each of the 8 register is allocated a 128-bit region, with the low 80 bits used for the register and the upper 48 bits unused.

x87 state is XSAVE-managed but the x87 FPU feature is not XSAVE-enabled. The XSAVE feature set can operate on x87 state only if the feature set is enabled (CR4.OSXSAVE = 1).[1] Software can otherwise use x87 state even if the XSAVE feature set is not enabled.

13.5.2 SSE State

Instructions in the XSAVE feature set can manage the registers used by the streaming SIMD extensions (**SSE state**) just as the FXSAVE and FXRSTOR instructions do. They organize all SSE state in the legacy region of the XSAVE area (see Section 13.4.1). This region is illustrated in Table 13-1; the SSE state is listed below, along with details of its interactions with the XSAVE feature set:

- Bytes 23:0 are used for x87 state (see Section 13.5.1).
- Bytes 27:24 are used for the MXCSR register. XRSTOR and XRSTORS generate general-protection faults (#GP) in response to attempts to set any of the reserved bits of the MXCSR register.[2]
- Bytes 31:28 are used for the MXCSR_MASK value. XRSTOR and XRSTORS ignore this field.
- Bytes 159:32 are used for x87 state.
- Bytes 287:160 are used for the registers XMM0–XMM7.
- Bytes 415:288 are used for the registers XMM8–XMM15. These fields are used only in 64-bit mode. Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not write to these bytes; executions of XRSTOR and XRSTORS outside 64-bit mode do not read these bytes and do not update XMM8–XMM15.

SSE state is XSAVE-managed but the SSE feature is not XSAVE-enabled. The XSAVE feature set can operate on SSE state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage SSE state (XCR0[1] = 1). Software can otherwise use SSE state even if the XSAVE feature set is not enabled or has not been configured to manage SSE state.

1. The processor ensures that XCR0[0] is always 1.
2. While MXCSR and MXCSR_MASK are part of SSE state, their treatment by the XSAVE feature set is not the same as that of the XMM registers. See Section 13.6 through Section 13.10 for details.

13.5.3 AVX State

The register state used by the Intel® Advanced Vector Extensions (AVX) comprises the MXCSR register and 16 256-bit vector registers called YMM0–YMM15. The low 128 bits of each register YMM*i* is identical to the SSE register XMM*i*. Thus, the new state register state added by AVX comprises the upper 128 bits of the registers YMM0–YMM15. These 16 128-bit values are denoted YMM0_H–YMM15_H and are collectively called **AVX state**.

As noted in Section 13.1, the XSAVE feature set manages AVX state as state component 2. Thus, AVX state is located in the extended region of the XSAVE area (see Section 13.4.3).

As noted in Section 13.2, CPUID.(EAX=0DH,ECX=2):EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section of the extended region of the XSAVE area used for AVX state (when the standard format of the extended region is used). CPUID returns this value as 576. CPUID.(EAX=0DH,ECX=2):EAX enumerates the size (in bytes) required for AVX state. CPUID returns this value as 256.

The XSAVE feature set partitions YMM0_H–YMM15_H in a manner similar to that used for the XMM registers (see Section 13.5.2). Bytes 127:0 of the AVX-state section are used for YMM0_H–YMM7_H. Bytes 255:128 are used for YMM8_H–YMM15_H, but they are used only in 64-bit mode. (Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not write to bytes 255:128; executions of XRSTOR and XRSTORS outside 64-bit mode do not read these bytes and do not update YMM8_H–YMM15_H.)

AVX state is XSAVE-managed and the AVX feature is XSAVE-enabled. The XSAVE feature set can operate on AVX state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage AVX state (XCR0[1] = XCR0[2] = 1).[1] AVX instructions cannot be used unless the XSAVE feature set is enabled and has been configured to manage AVX state.

...

13.10 OPERATION OF XSAVES

The operation of XSAVES is similar to that of XSAVEC. The main differences are (1) XSAVES can be executed only if CPL = 0; (2) XSAVES can operate on the state components whose bits are set in XCR0 | IA32_XSS; and (3) XSAVES uses the modified optimization (see Section 13.5.4). See Section 13.2 for details of how to determine whether XSAVES is supported.

The XSAVES instruction takes a single memory operand, which is an XSAVE area. In addition, the register pair EDX:EAX is an implicit operand used as a state-component bitmap (see Section 13.1) called the **instruction mask**. EDX:EAX & (XCR0 | IA32_XSS) (the logical AND the instruction mask with the logical OR of XCR0 and IA32_XSS) is the **requested-feature bitmap** (**RFBM**) of the state components to be saved.

The following conditions cause execution of the XSAVES instruction to generate a fault:

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If CPL > 0 or if the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[2]

If none of these conditions cause a fault, execution of XSAVES writes the XSTATE_BV field of the XSAVE header (see Section 13.4.2), setting XSTATE_BV[*i*] ($0 \le i \le 63$) as follows:

- If RFBM[*i*] = 0, XSTATE_BV[*i*] is written as 0.

1. The XSETBV instruction can set XCR0[2] to 1 only if it is also setting XCR0[1] to 1. XSETBV generates a general-protection exception (#GP) in response to attempts to set XCR0[2] while clearing XCR0[1].
2. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.

- If RFBM[*i*] = 1, XSTATE_BV[*i*] is set to the value of XINUSE[*i*] (see below for an exception made for XSTATE_BV[1]). Section 13.5.4 defines XINUSE to describe the processor init optimization. The nature of that optimization implies the following:

  — If state component *i* is in its initial configuration, XSTATE_BV[*i*] may be written with either 0 or 1.

  — If state component *i* is not in its initial configuration, XSTATE_BV[*i*] is written with 1.

  Section 13.6 specifies the initial configuration of each state component. However, if RFBM[1] = 1 and MXCSR does not have the value 1F80H, XSAVES writes XSTATE_BV[1] as 1 even if XINUSE[1] = 0.

The XSAVES instructions sets bit 63 of the XCOMP_BV field of the XSAVE header while writing RFBM[62:0] to XCOMP_BV[62:0]. The XSAVEC instruction does not write any part of the XSAVE header other than the XSTATE_BV and XCOMP_BV fields.

Execution of XSAVES saves into the XSAVE area those state components corresponding to bits that are set in RFBM. State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \le i \le 62$, is located in the extended region; the XSAVES instruction always uses the compacted format for the extended region (see Section 13.4.3).

See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes.

Execution of XSAVES performs the init optimization to reduce the amount of data written to memory. If XINUSE[*i*] = 0, state component *i* is not saved to the XSAVE area (even if RFBM[*i*] = 1). However, if RFBM[1] = 1 and MXCSR does not have the value 1F80H, XSAVES writes saves all of state component 1 (SSE — including the XMM registers) even if XINUSE[1] = 0.

Like XSAVEOPT, XSAVES may perform the modified optimization. Each execution of XRSTOR and XRSTORS establishes XRSTOR_INFO as a 4-tuple $\langle w,x,y,z \rangle$ (see Section 13.7.3 and Section 13.11). Execution of XSAVES uses the modified optimization only if the following all hold:

- *w* = CPL;
- *x* = 1 if and only if the logical processor is in VMX non-root operation;
- *y* is the linear address of the XSAVE area being used by XSAVEOPT; and
- *z*[63] is 1 and *z*[62:0] = RFBM[62:0]. (This last item implies that XSAVES does not use the modified optimization if the last execution of XRSTOR used the standard form and followed the last execution of XRSTORS.)

If XSAVES uses the modified optimization and XMODIFIED[*i*] = 0 (see Section 13.5.4), state component *i* is not saved to the XSAVE area.

...

7. Updates to Appendix E, Volume 1

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

...

E.4.3 Example SIMD Floating-Point Emulation Implementation

The sample code listed below may be considered as being part of a user-level floating-point exception filter for the SSE/SSE2/SSE3 numeric instructions. It is assumed that the filter function is invoked by a low-level exception handler (invoked for exception 19 when an unmasked floating-point exception occurs), and that it operates as explained in Section E.4.1, "Floating-Point Emulation." The sample code does the emulation only for the SSE instructions for addition, subtraction, multiplication, and division. For this, it uses C code and x87 FPU operations.

Operations corresponding to other SSE/SSE2/SSE3 numeric instructions can be emulated similarly. The example assumes that the emulation function receives a pointer to a data structure specifying a number of input parameters: the operation that caused the exception, a set of sub-operands (unpacked, of type float), the rounding mode (the precision is always single), exception masks (having the same relative bit positions as in the MXCSR but starting from bit 0 in an unsigned integer), and flush-to-zero and denormals-are-zeros indicators.

The output parameters are a floating-point result (of type float), the cause of the exception (identified by constants not explicitly defined below), and the exception status flags. The corresponding C definition is:

```
typedef struct {
  unsigned int operation;                 //SSE or SSE2 operation: ADDPS, ADDSS, ...
  unsigned int operand1_uint32; //first operand value
  unsigned int operand2_uint32; //second operand value (if any)
  float result_fval; // result value (if any)
  unsigned int rounding_mode; //rounding mode
  unsigned int exc_masks; //exception masks, in the order P,U,O,Z,D,I
  unsigned int exception_cause; //exception cause
  unsigned int status_flag_inexact; //inexact status flag
  unsigned int status_flag_underflow; //underflow status flag
  unsigned int status_flag_overflow; //overflow status flag
  unsigned int status_flag_divide_by_zero;
                                                  //divide by zero status flag
  unsigned int status_flag_denormal_operand;
                                                  //denormal operand status flag
  unsigned int status_flag_invalid_operation;
                                                  //invalid operation status flag
  unsigned int ftz; // flush-to-zero flag
  unsigned int daz; // denormals-are-zeros flag
} EXC_ENV;
```

The arithmetic operations exemplified are emulated as follows:

1. If the denormals-are-zeros mode is enabled (the DAZ bit in MXCSR is set to 1), replace all the denormal inputs with zeroes of the same sign (the denormal flag is not affected by this change).
2. Perform the operation using x87 FPU instructions, with exceptions disabled, the original user rounding mode, and single precision. This reveals invalid, denormal, or divide-by-zero exceptions (if there are any) and stores the result in memory as a double precision value (whose exponent range is large enough to look like "unbounded" to the result of the single precision computation).
3. If no unmasked exceptions were detected, determine if the magnitude of the result is less than the smallest normal number that can be represented in single precision format, or greater than the largest normal number that can be represented in single precision format (huge). If an unmasked overflow or underflow occurs, calculate the scaled result that will be handed to the user exception handler, as specified by IEEE Standard 754.
4. If no exception was raised, calculate the result with a "bounded" exponent. If the result is tiny, it requires denormalization (shifting the significand right while incrementing the exponent to bring it into the admissible range of [-126,+127] for single precision floating-point numbers).

   The result obtained in step 2 cannot be used because it might incur a double rounding error (it was rounded to 24 bits in step 2, and might have to be rounded again in the denormalization process). To overcome this is, calculate the result as a double precision value, and store it to memory in single precision format.

   Rounding first to 53 bits in the significand, and then to 24 never causes a double rounding error (exact

properties exist that state when double-rounding error occurs, but for the elementary arithmetic operations, the rule of thumb is that if an infinitely precise result is rounded to 2p+1 bits and then again to p bits, the result is the same as when rounding directly to p bits, which means that no double-rounding error occurs).

5. If the result is inexact and the inexact exceptions are unmasked, the calculated result will be delivered to the user floating-point exception handler.
6. The flush-to-zero case is dealt with if the result is tiny.
7. The emulation function returns RAISE_EXCEPTION to the filter function if an exception has to be raised (the exception_cause field indicates the cause). Otherwise, the emulation function returns DO_NOT_ RAISE_EXCEPTION. In the first case, the result is provided by the user exception handler called by the filter function. In the second case, it is provided by the emulation function. The filter function has to collect all the partial results, and to assemble the scalar or packed result that is used if execution is to continue.

...

8. 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 processor QX6000 series
- Intel® Xeon® processor 7100 series

- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processor family
- Intel® 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

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

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

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

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

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

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

...

9. 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.

...

AAA—ASCII Adjust After Addition

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
37	AAA	NP	Invalid	Valid	ASCII adjust AL after addition.

Instruction Operand Encoding

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

Description

Adjusts the sum of two unpacked BCD values to create an unpacked BCD result. The AL register is the implied source and destination operand for this instruction. The AAA instruction is only useful when it follows an ADD instruction that adds (binary addition) two unpacked BCD values and stores a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

If the addition produces a decimal carry, the AH register increments by 1, and the CF and AF flags are set. If there was no decimal carry, the CF and AF flags are cleared and the AH register is unchanged. In either case, bits 4 through 7 of the AL register are set to 0.

This instruction executes as described in compatibility mode and legacy mode. It is not valid in 64-bit mode.

Operation

```
IF 64-Bit Mode
    THEN
        #UD;
    ELSE
        IF ((AL AND 0FH) > 9) or (AF = 1)
            THEN
                AX ← AX + 106H;
                AF ← 1;
                CF ← 1;
            ELSE
                AF ← 0;
                CF ← 0;
        FI;
        AL ← AL AND 0FH;
FI;
```

Flags Affected

The AF and CF flags are set to 1 if the adjustment results in a decimal carry; otherwise they are set to 0. The OF, SF, ZF, and PF flags are undefined.

Protected Mode Exceptions

#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

Same exceptions as protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as protected mode.

Compatibility Mode Exceptions

Same exceptions as protected mode.

64-Bit Mode Exceptions

#UD	If in 64-bit mode.

...

AESDEC—Perform One Round of an AES Decryption Flow

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 38 DE /r AESDEC xmm1, xmm2/m128	RM	V/V	AES	Perform one round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
VEX.NDS.128.66.0F38.WIG DE /r VAESDEC xmm1, xmm2, xmm3/m128	RVM	V/V	Both AES and AVX flags	Perform one round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm2 with a 128-bit round key from xmm3/m128; store the result in xmm1.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
RVM	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	NA

Description

This instruction performs a single round of the AES decryption flow using the Equivalent Inverse Cipher, with the round key from the second source operand, operating on a 128-bit data (state) from the first source operand, and store the result in the destination operand.

Use the AESDEC instruction for all but the last decryption round. For the last decryption round, use the AESDE-CLAST instruction.

128-bit Legacy SSE version: The first source operand and the destination operand are the same and must be an XMM register. The second source operand can be an XMM register or a 128-bit memory location. Bits (VLMAX-1:128) of the corresponding YMM destination register remain unchanged.

VEX.128 encoded version: The first source operand and the destination operand are XMM registers. The second source operand can be an XMM register or a 128-bit memory location. Bits (VLMAX-1:128) of the destination YMM register are zeroed.

Operation

AESDEC

```
STATE ← SRC1;
RoundKey ← SRC2;
STATE ← InvShiftRows( STATE );
STATE ← InvSubBytes( STATE );
STATE ← InvMixColumns( STATE );
DEST[127:0] ← STATE XOR RoundKey;
DEST[VLMAX-1:128] (Unmodified)
```

VAESDEC

```
STATE ← SRC1;
RoundKey ← SRC2;
STATE ← InvShiftRows( STATE );
STATE ← InvSubBytes( STATE );
STATE ← InvMixColumns( STATE );
DEST[127:0] ← STATE XOR RoundKey;
DEST[VLMAX-1:128] ← 0
```

Intel C/C++ Compiler Intrinsic Equivalent

(V)AESDEC: __m128i _mm_aesdec (__m128i, __m128i)

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4.

...

AESKEYGENASSIST—AES Round Key Generation Assist

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A DF /r ib AESKEYGENASSIST xmm1, xmm2/m128, imm8	RMI	V/V	AES	Assist in AES round key generation using an 8 bits Round Constant (RCON) specified in the immediate byte, operating on 128 bits of data specified in xmm2/m128 and stores the result in xmm1.
VEX.128.66.0F3A.WIG DF /r ib VAESKEYGENASSIST xmm1, xmm2/m128, imm8	RMI	V/V	Both AES and AVX flags	Assist in AES round key generation using 8 bits Round Constant (RCON) specified in the immediate byte, operating on 128 bits of data specified in xmm2/m128 and stores the result in xmm1.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
RMI	ModRM:reg (w)	ModRM:r/m (r)	imm8	NA

Description

Assist in expanding the AES cipher key, by computing steps towards generating a round key for encryption, using 128-bit data specified in the source operand and an 8-bit round constant specified as an immediate, store the result in the destination operand.

The destination operand is an XMM register. The source operand can be an XMM register or a 128-bit memory location.

128-bit Legacy SSE version: Bits (VLMAX-1:128) of the corresponding YMM destination register remain unchanged.

VEX.128 encoded version: Bits (VLMAX-1:128) of the destination YMM register are zeroed.

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

Operation

AESKEYGENASSIST

```
X3[31:0] ← SRC [127: 96];
X2[31:0] ← SRC [95: 64];
X1[31:0] ← SRC [63: 32];
X0[31:0] ← SRC [31: 0];
RCON[31:0] ← ZeroExtend(Imm8[7:0]);
DEST[31:0] ← SubWord(X1);
DEST[63:32 ] ← RotWord( SubWord(X1) ) XOR RCON;
DEST[95:64] ← SubWord(X3);
DEST[127:96] ← RotWord( SubWord(X3) ) XOR RCON;
DEST[VLMAX-1:128] (Unmodified)
```

VAESKEYGENASSIST

```
X3[31:0] ← SRC [127: 96];
X2[31:0] ← SRC [95: 64];
X1[31:0] ← SRC [63: 32];
X0[31:0] ← SRC [31: 0];
RCON[31:0] ← ZeroExtend(Imm8[7:0]);
DEST[31:0] ← SubWord(X1);
DEST[63:32 ] ← RotWord( SubWord(X1) ) XOR RCON;
DEST[95:64] ← SubWord(X3);
DEST[127:96] ← RotWord( SubWord(X3) ) XOR RCON;
DEST[VLMAX-1:128] ← 0;
```

Intel C/C++ Compiler Intrinsic Equivalent

(V)AESKEYGENASSIST: __m128i _mm_aeskeygenassist (__m128i, const int)

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 4; additionally

#UD	If VEX.vvvv ≠ 1111B.

...

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. *)
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. *)
```

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

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

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

See also:

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

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

Table 3-17 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" "ineI"
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-5)
	EBX	Bits 07-00: Brand Index Bits 15-08: CLFLUSH line size (Value ∗ 8 = cache line size in bytes) Bits 23-16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31-24: Initial APIC ID
	ECX EDX	Feature Information (see Figure 3-6 and Table 3-19) Feature Information (see Figure 3-7 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	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.)
	EDX	Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature. See AP-485, *Intel Processor Identification and the CPUID Instruction* (Order Number 241618) for more information on PSN.
	CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLE.BOOT_NT4[bit 22] = 0 (default).	

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

Initial EAX Value	Information Provided about the Processor
	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 3-182. 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 **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

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

Initial EAX Value	Information Provided about the Processor	
05H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled Bits 31 - 02: Reserved
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWAIT Bits 07 - 04: Number of C1* sub C-states supported using MWAIT Bits 11 - 08: Number of C2* sub C-states supported using MWAIT Bits 15 - 12: Number of C3* sub C-states supported using MWAIT Bits 19 - 16: Number of C4* sub C-states supported using MWAIT Bits 23 - 20: Number of C5* sub C-states supported using MWAIT Bits 27 - 24: Number of C6* sub C-states supported using MWAIT Bits 31 - 28: Number of C7* sub C-states supported using MWAIT **NOTE:** * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
	Thermal and Power Management Leaf	
06H	EAX	Bit 00: Digital temperature sensor is supported if set Bit 01: Intel Turbo Boost Technology Available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bit 07: HWP. HWP base registers (IA32_PM_ENALBE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS) are supported if set. Bit 08: HWP_Notification. IA32_HWP_INTERRUPT MSR is supported if set. Bit 09: HWP_Activity_Window. IA32_HWP_REQUEST[bits 41:32] is supported if set. Bit 10: HWP_Energy_Performance_Preference. IA32_HWP_REQUEST[bits 31:24] is supported if set. Bit 11: HWP_Package_Level_Request. IA32_HWP_REQUEST_PKG MSR is supported if set. Bit 12: Reserved. Bit 13: HDC. HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, IA32_THREAD_STALL MSRs are supported if set. Bits 31 - 15: Reserved
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor Bits 31 - 04: Reserved
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency. Bits 02 - 01: Reserved = 0 Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H). Bits 31 - 04: Reserved = 0

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

Initial EAX Value	Information Provided about the Processor	
	EDX	Reserved = 0
	Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)	
07H		Sub-leaf 0 (Input ECX = 0). * Bits 31-00: Reports the maximum input value for supported leaf 7 sub-leaves. 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: Reserved. 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 Bits 24:21: Reserved Bit 25: Intel Processor Trace Bits 31:26: Reserved Bit 00: PREFETCHWT1 Bit 31-01: Reserved 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.
		EAX EBX ECX EDX
	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	

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

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

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

Initial EAX Value	Information Provided about the Processor	
		** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0 : invalid 1 : SMT 2 : Core 3-255 : Reserved
	Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)	
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: 128-bit SSE Bit 02: 256-bit AVX Bits 31- 03: 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 valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved.
	Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)	
0DH	EAX	Bits 31-04: Reserved 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
	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 valid bit fields of the lower 32 bits of IA32_XSS. If a bit is 0, the corresponding bit field in IA32_XSS is reserved. Bits 07-00: Reserved Bit 08: IA32_XSS[bit 8] is supported if 1 Bits 31-09: Reserved
	EDX	Bits 31-00: Reports the valid bit fields of the upper 32 bits of IA32_XSS. If a bit is 0, the corresponding bit field in IA32_XSS is 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 valid sub-leaf index maps to a valid bit in either the XCR0 register or the IA32_XSS MSR starting at bit position 2. * 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 sub-leaf index, n, maps to a valid bit in the IA32_XSS MSR and bit 0 is clear if n maps to a valid bit in XCR0. Bits 31-1 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)	
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 EDX
	EAX	Reserved.

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

Initial EAX Value	Information Provided about the Processor	
	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. Bits 31:02: 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 31- 01: Reserved
	ECX	Bit 00: If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. Bit 01: If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. Bit 30:02: 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
Time Stamp Counter/Core Crystal Clock Information-leaf		
15H		**NOTES:** If EBX[31:0] is 0, the TSC/"core crystal clock" ration 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.

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

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 31:0: Reserved = 0.
	Unimplemented CPUID Leaf Functions	
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (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

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

Initial EAX Value	Information Provided about the Processor
80000005H	EAX Reserved = 0 EBX Reserved = 0 ECX Reserved = 0 EDX Reserved = 0
80000006H	EAX Reserved = 0 EBX Reserved = 0 ECX Bits 07-00: Cache Line size in bytes Bits 11-08: Reserved Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units EDX Reserved = 0 **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 = 0: Returns CPUID's Highest Value for Basic Processor Information and the Vendor Identification String

When CPUID executes with EAX set to 0, the processor returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register (see Table 3-18) and is processor specific.

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

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

EDX ← 49656e69h (* "inel", with i in the low eight bits of DL *)
ECX ← 6c65746eh (* "ntel", with n in the low eight bits of CL *)

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

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

IA32_BIOS_SIGN_ID Returns Microcode Update Signature

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

INPUT EAX = 1: Returns Model, Family, Stepping Information

When CPUID executes with EAX set to 1, version information is returned in EAX (see Figure 3-5). For example: model, family, and processor type for the Intel Xeon processor 5100 series is as follows:

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

See Table 3-18 for available processor type values. Stepping IDs are provided as needed.

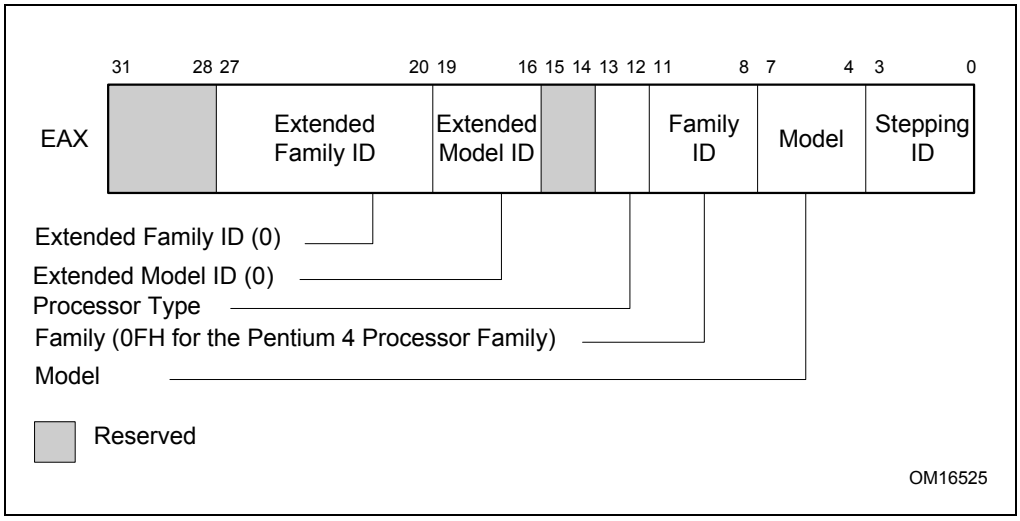

Figure 3-5 Version Information Returned by CPUID in EAX

Table 3-18 Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B

Table 3-18 Processor Type Field

Type	Encoding
Dual processor (not applicable to Intel486 processors)	10B
Intel reserved	11B

NOTE

See Chapter 17 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for information on identifying earlier IA-32 processors.

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

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

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

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

INPUT EAX = 1: Returns Additional Information in EBX

When CPUID executes with EAX set to 1, 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 with CLFLUSH instruction in 8-byte increments. This field was introduced in the Pentium 4 processor.
- Local APIC ID (high byte of EBX) — this number is the 8-bit ID that is assigned to the local APIC on the processor during power up. This field was introduced in the Pentium 4 processor.

INPUT EAX = 1: Returns Feature Information in ECX and EDX

When CPUID executes with EAX set to 1, feature information is returned in ECX and EDX.

- Figure 3-6 and Table 3-19 show encodings for ECX.
- Figure 3-7 and Table 3-20 show encodings for EDX.

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

NOTE

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

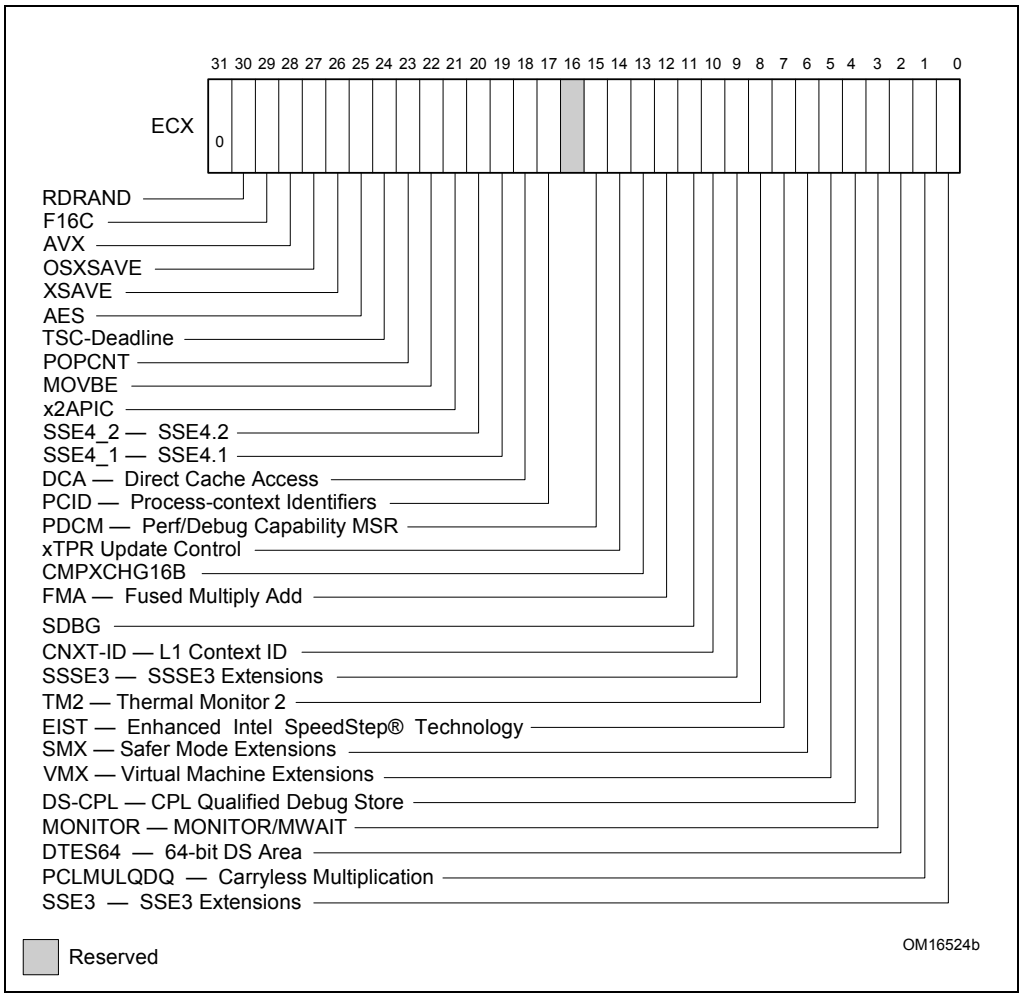

Figure 3-6 Feature Information Returned in the ECX Register

Table 3-19 Feature Information Returned in the ECX Register

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

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

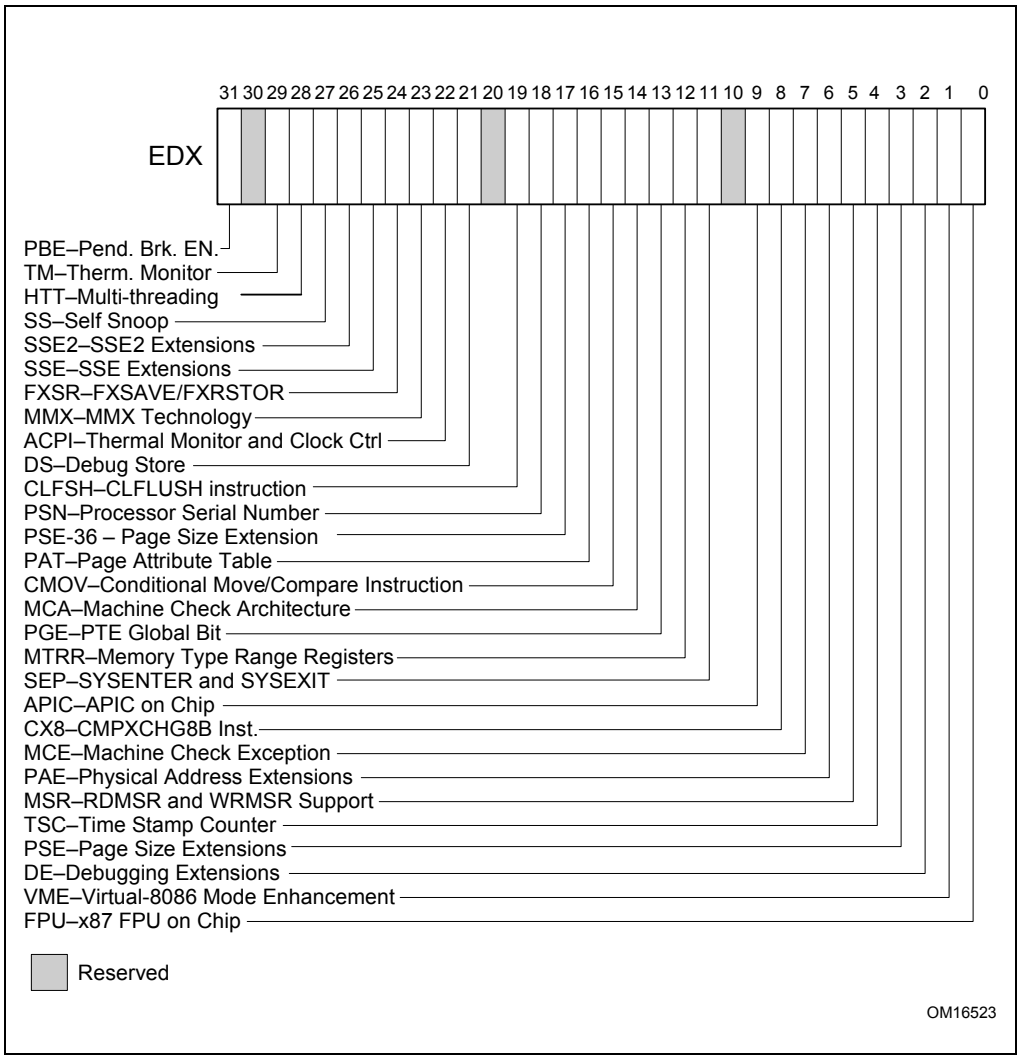

Figure 3-7 Feature Information Returned in the EDX Register

Table 3-20 More on Feature Information Returned in the EDX Register

Bit #	Mnemonic	Description
0	FPU	**Floating Point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	**Page Size Extension.** Large pages of size 4 MByte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.

Table 3-20 More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1.
7	MCE	**Machine Check Exception.** Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	**APIC On-Chip.** The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).
10	Reserved	Reserved
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.
13	PGE	**Page Global Bit.** The global bit is supported in paging-structure entries that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.
14	MCA	**Machine Check Architecture.** The Machine Check Architecture, which provides a compatible mechanism for error reporting in P6 family, Pentium 4, Intel Xeon processors, and future processors, is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	**Conditional Move Instructions.** The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported
16	PAT	**Page Attribute Table.** Page Attribute Table is supported. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory accessed through a linear address on a 4KB granularity.
17	PSE-36	**36-Bit Page Size Extension.** 4-MByte pages addressing physical memory beyond 4 GBytes are supported with 32-bit paging. This feature indicates that upper bits of the physical address of a 4-MByte page are encoded in bits 20:13 of the page-directory entry. Such physical addresses are limited by MAXPHYADDR and may be up to 40 bits in size.
18	PSN	**Processor Serial Number.** The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	**CLFLUSH Instruction.** CLFLUSH Instruction is supported.
20	Reserved	Reserved
21	DS	**Debug Store.** The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities (see Chapter 23, "Introduction to Virtual-Machine Extensions," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).

Table 3-20 More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Max APIC IDs reserved field is Valid.** A value of 0 for HTT indicates there is only a single logical processor in the package and software should assume only a single APIC ID is reserved. A value of 1 for HTT indicates the value in CPUID.1.EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package) is valid for the package.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved
31	PBE	**Pending Break Enable.** The processor supports the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

INPUT EAX = 2: TLB/Cache/Prefetch Information Returned in EAX, EBX, ECX, EDX

When CPUID executes with EAX set to 2, the processor returns information about the processor's internal TLBs, cache and prefetch hardware in the EAX, EBX, ECX, and EDX registers. The information is reported in encoded form and fall into the following categories:

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors. There are four types of encoding values for the byte descriptor, the encoding type is noted in the second column of Table 3-21. Table 3-21 lists the encoding of these descriptors. Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache, prefetch, or TLB types. The descriptors may appear in any order. Note also a processor may report a general descriptor type (FFH) and not report any byte descriptor of "cache type" via CPUID leaf 2.

Table 3-21 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

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

Value	Type	Description
04H	TLB	Data TLB: 4 MByte pages, 4-way set associative, 8 entries
05H	TLB	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries
06H	Cache	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size
08H	Cache	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size
09H	Cache	1st-level instruction cache: 32KBytes, 4-way set associative, 64 byte line size
0AH	Cache	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size
0BH	TLB	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries
0CH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size
0DH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 64 byte line size
0EH	Cache	1st-level data cache: 24 KBytes, 6-way set associative, 64 byte line size
1DH	Cache	2nd-level cache: 128 KBytes, 2-way set associative, 64 byte line size
21H	Cache	2nd-level cache: 256 KBytes, 8-way set associative, 64 byte line size
22H	Cache	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector
23H	Cache	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
24H	Cache	2nd-level cache: 1 MBytes, 16-way set associative, 64 byte line size
25H	Cache	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
29H	Cache	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
2CH	Cache	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size
30H	Cache	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size
40H	Cache	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache
41H	Cache	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size
42H	Cache	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size
43H	Cache	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size
44H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size
45H	Cache	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size
46H	Cache	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size
47H	Cache	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size
48H	Cache	2nd-level cache: 3MByte, 12-way set associative, 64 byte line size
49H	Cache	3rd-level cache: 4MB, 16-way set associative, 64-byte line size (Intel Xeon processor MP, Family 0FH, Model 06H); 2nd-level cache: 4 MByte, 16-way set associative, 64 byte line size
4AH	Cache	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size
4BH	Cache	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size
4CH	Cache	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries
50H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries

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

Value	Type	Description
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
70H	Cache	Trace cache: 12 K-µop, 8-way set associative
71H	Cache	Trace cache: 16 K-µop, 8-way set associative
72H	Cache	Trace cache: 32 K-µop, 8-way set associative
76H	TLB	Instruction TLB: 2M/4M pages, fully associative, 8 entries
78H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 64byte line size
79H	Cache	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7AH	Cache	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7BH	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7CH	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector
7DH	Cache	2nd-level cache: 2 MByte, 8-way set associative, 64byte line size
7FH	Cache	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size
80H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64-byte line size
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size
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

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

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

Example 3-1 Example of Cache and TLB Interpretation

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

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

Which means:

- The least-significant byte (byte 0) of register EAX is set to 01H. This value should be ignored.

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

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

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

Software can enumerate the deterministic cache parameters for each level of the cache hierarchy starting with an index value of 0, until the parameters report the value associated with the cache type field is 0. The architecturally defined fields reported by deterministic cache parameters are documented in Table 3-17.

This Cache Size in Bytes

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

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

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

INPUT EAX = 05H: Returns MONITOR and MWAIT Features

When CPUID executes with EAX set to 05H, the processor returns information about features available to MONITOR/MWAIT instructions. The MONITOR instruction is used for address-range monitoring in conjunction with MWAIT instruction. The MWAIT instruction optionally provides additional extensions for advanced power management. See Table 3-17.

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

When CPUID executes with EAX set to 06H, the processor returns information about thermal and power management features. See Table 3-17.

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

When CPUID executes with EAX set to 07H and ECX = 0, the processor returns information about the maximum input value for sub-leaves that contain extended feature flags. See Table 3-17.

When CPUID executes with EAX set to 07H and the input value of ECX is invalid (see leaf 07H entry in Table 3-17), the processor returns 0 in EAX/EBX/ECX/EDX. In subleaf 0, EAX returns the maximum input value of the highest leaf 7 sub-leaf, and EBX, ECX & EDX contain information of extended feature flags.

INPUT EAX = 09H: Returns Direct Cache Access Information

When CPUID executes with EAX set to 09H, the processor returns information about Direct Cache Access capabilities. See Table 3-17.

INPUT EAX = 0AH: Returns Architectural Performance Monitoring Features

When CPUID executes with EAX set to 0AH, the processor returns information about support for architectural performance monitoring capabilities. Architectural performance monitoring is supported if the version ID (see Table 3-17) is greater than Pn 0. See Table 3-17.

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

INPUT EAX = 0BH: Returns Extended Topology Information

When CPUID executes with EAX set to 0BH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 0BH by verifying (a) the highest leaf index supported by CPUID is >= 0BH, and (b) CPUID.0BH:EBX[15:0] reports a non-zero value. See Table 3-17.

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

When CPUID executes with EAX set to 0DH and ECX = 0, the processor returns information about the bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area. See Table 3-17.

When CPUID executes with EAX set to 0DH and ECX = n (n > 1, and is a valid sub-leaf index), the processor returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. See Table 3-17. Software can use the forward-extendable technique depicted below to query the valid sub-leaves and obtain size and offset information for each processor extended state save area:

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

INPUT EAX = 0FH: Returns Platform Quality of Service (PQoS) Monitoring Enumeration Information

When CPUID executes with EAX set to 0FH and ECX = 0, the processor returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type. See Table 3-17.

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

INPUT EAX = 10H: Returns Platform Quality of Service (PQoS) Enforcement Enumeration Information

When CPUID executes with EAX set to 10H and ECX = 0, the processor returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit

1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type. See Table 3-17.

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

METHODS FOR RETURNING BRANDING INFORMATION

Use the following techniques to access branding information:

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

These two methods are discussed in the following sections. For methods that are available in early processors, see Section: "Identification of Earlier IA-32 Processors" in Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

The Processor Brand String Method

Figure 3-8 describes the algorithm used for detection of the brand string. Processor brand identification software should execute this algorithm on all Intel 64 and IA-32 processors.

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

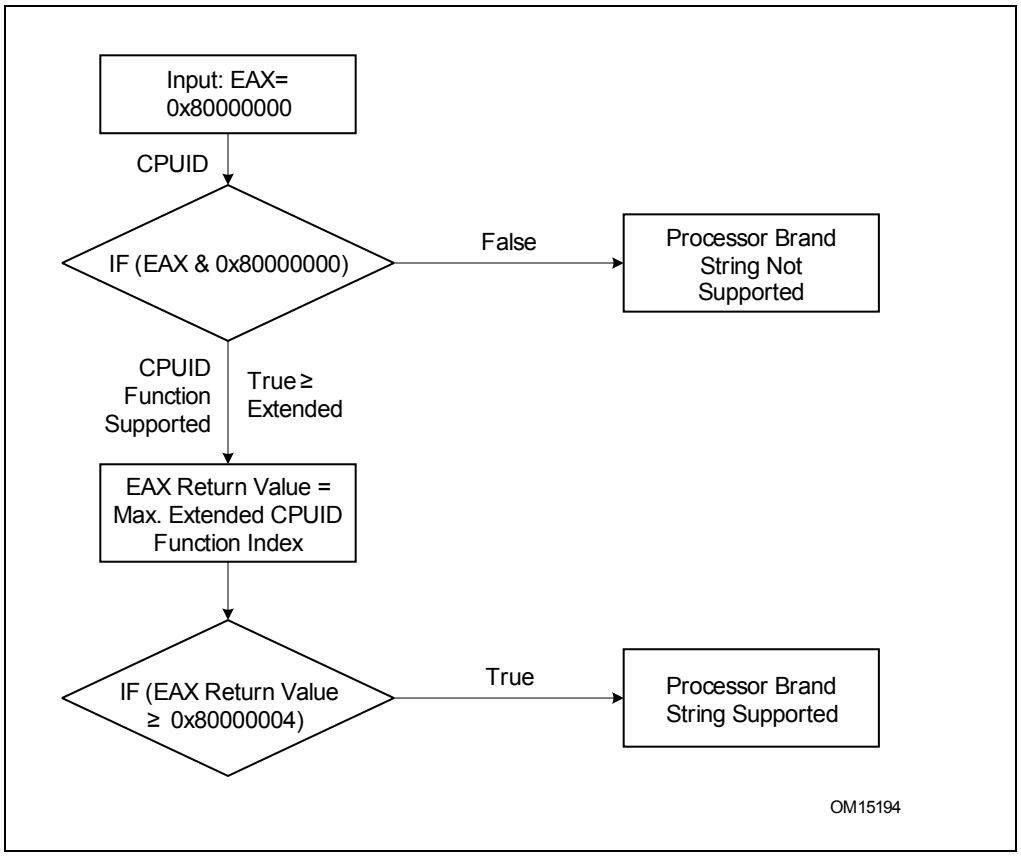

Figure 3-8 Determination of Support for the Processor Brand String

How Brand Strings Work

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

Table 3-22 shows the brand string that is returned by the first processor in the Pentium 4 processor family.

Table 3-22 Processor Brand String Returned with Pentium 4 Processor

EAX Input Value	Return Values	ASCII Equivalent
80000002H	EAX = 20202020H EBX = 20202020H ECX = 20202020H EDX = 6E492020H	“ ” “ ” “ ” “nI ”
80000003H	EAX = 286C6574H EBX = 50202952H ECX = 69746E65H EDX = 52286D75H	“(let” “P)R” “itne” “R(mu”
80000004H	EAX = 20342029H EBX = 20555043H ECX = 30303531H EDX = 007A484DH	“ 4)” “ UPC” “0051” “\0zHM”

Extracting the Processor Frequency from Brand Strings

Figure 3-9 provides an algorithm which software can use to extract the Processor Base frequency from the processor brand string.

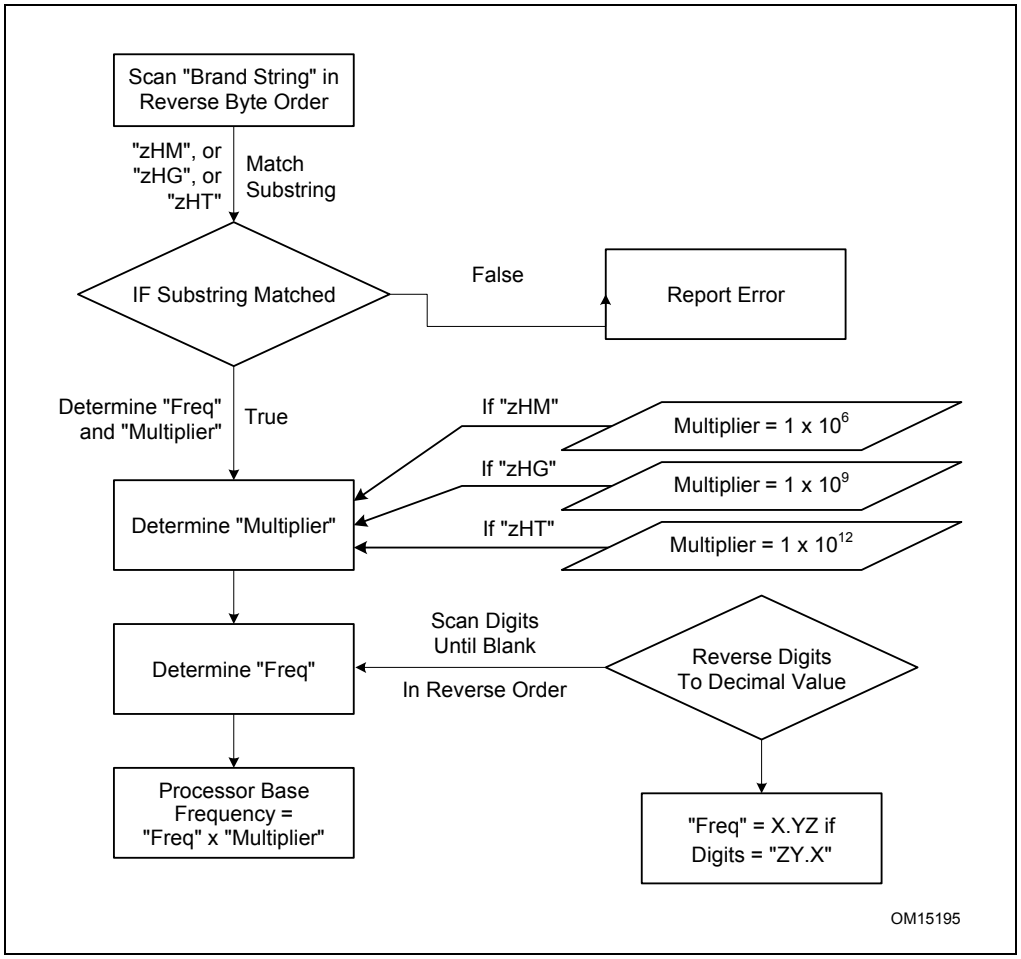

Figure 3-9 Algorithm for Extracting Processor Frequency

The Processor Brand Index Method

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

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

Table 3-23 shows brand indices that have identification strings associated with them.

Table 3-23 Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

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

NOTES:

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

IA-32 Architecture Compatibility

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

Operation

IA32_BIOS_SIGN_ID MSR ← Update with installed microcode revision number;

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

```
BREAK;
EAX = 1H:
    EAX[3:0] ← Stepping ID;
    EAX[7:4] ← Model;
    EAX[11:8] ← Family;
    EAX[13:12] ← Processor type;
    EAX[15:14] ← Reserved;
    EAX[19:16] ← Extended Model;
    EAX[27:20] ← Extended Family;
    EAX[31:28] ← Reserved;
    EBX[7:0] ← Brand Index; (* Reserved if the value is zero. *)
    EBX[15:8] ← CLFLUSH Line Size;
    EBX[16:23] ← Reserved; (* Number of threads enabled = 2 if MT enable fuse set. *)
    EBX[24:31] ← Initial APIC ID;
    ECX ← Feature flags; (* See Figure 3-6. *)
    EDX ← Feature flags; (* See Figure 3-7. *)
BREAK;
EAX = 2H:
    EAX ← Cache and TLB information;
    EBX ← Cache and TLB information;
    ECX ← Cache and TLB information;
    EDX ← Cache and TLB information;
BREAK;
EAX = 3H:
    EAX ← Reserved;
    EBX ← Reserved;
    ECX ← ProcessorSerialNumber[31:0];
    (* Pentium III processors only, otherwise reserved. *)
    EDX ← ProcessorSerialNumber[63:32];
    (* Pentium III processors only, otherwise reserved. *
BREAK
EAX = 4H:
    EAX ← Deterministic Cache Parameters Leaf; (* See Table 3-17. *)
    EBX ← Deterministic Cache Parameters Leaf;
    ECX ← Deterministic Cache Parameters Leaf;
    EDX ← Deterministic Cache Parameters Leaf;
BREAK;
EAX = 5H:
    EAX ← MONITOR/MWAIT Leaf; (* See Table 3-17. *)
    EBX ← MONITOR/MWAIT Leaf;
    ECX ← MONITOR/MWAIT Leaf;
    EDX ← MONITOR/MWAIT Leaf;
BREAK;
EAX = 6H:
    EAX ← Thermal and Power Management Leaf; (* See Table 3-17. *)
    EBX ← Thermal and Power Management Leaf;
    ECX ← Thermal and Power Management Leaf;
    EDX ← Thermal and Power Management Leaf;
BREAK;
EAX = 7H:
```

```
        EAX ← Structured Extended Feature Flags Enumeration Leaf; (* See Table 3-17. *)
        EBX ← Structured Extended Feature Flags Enumeration Leaf;
        ECX ← Structured Extended Feature Flags Enumeration Leaf;
        EDX ← Structured Extended Feature Flags Enumeration Leaf;
    BREAK;
    EAX = 8H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    EAX = 9H:
        EAX ← Direct Cache Access Information Leaf; (* See Table 3-17. *)
        EBX ← Direct Cache Access Information Leaf;
        ECX ← Direct Cache Access Information Leaf;
        EDX ← Direct Cache Access Information Leaf;
    BREAK;
    EAX = AH:
        EAX ← Architectural Performance Monitoring Leaf; (* See Table 3-17. *)
        EBX ← Architectural Performance Monitoring Leaf;
        ECX ← Architectural Performance Monitoring Leaf;
        EDX ← Architectural Performance Monitoring Leaf;
        BREAK
    EAX = BH:
        EAX ← Extended Topology Enumeration Leaf; (* See Table 3-17. *)
        EBX ← Extended Topology Enumeration Leaf;
        ECX ← Extended Topology Enumeration Leaf;
        EDX ← Extended Topology Enumeration Leaf;
    BREAK;
    EAX = CH:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    EAX = DH:
        EAX ← Processor Extended State Enumeration Leaf; (* See Table 3-17. *)
        EBX ← Processor Extended State Enumeration Leaf;
        ECX ← Processor Extended State Enumeration Leaf;
        EDX ← Processor Extended State Enumeration Leaf;
    BREAK;
    EAX = EH:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    EAX = FH:
        EAX ← Platform Quality of Service Monitoring Enumeration Leaf; (* See Table 3-17. *)
        EBX ← Platform Quality of Service Monitoring Enumeration Leaf;
```

```
            ECX ← Platform Quality of Service Monitoring Enumeration Leaf;
            EDX ← Platform Quality of Service Monitoring Enumeration Leaf;
        BREAK;
        EAX = 10H:
            EAX ← Platform Quality of Service Enforcement Enumeration Leaf; (* See Table 3-17. *)
            EBX ← Platform Quality of Service Enforcement Enumeration Leaf;
            ECX ← Platform Quality of Service Enforcement Enumeration Leaf;
            EDX ← Platform Quality of Service Enforcement Enumeration Leaf;
        BREAK;
        EAX = 15H:
            EAX ← Time Stamp Counter/Core Crystal Clock Information Leaf; (* See Table 3-17. *)
            EBX ← Time Stamp Counter/Core Crystal Clock Information Leaf;
            ECX ← Time Stamp Counter/Core Crystal Clock Information Leaf;
            EDX ← Time Stamp Counter/Core Crystal Clock Information Leaf;
        BREAK;
    BREAK;
    EAX = 80000000H:
        EAX ← Highest extended function input value understood by CPUID;
        EBX ← Reserved;
        ECX ← Reserved;
        EDX ← Reserved;
    BREAK;
    EAX = 80000001H:
        EAX ← Reserved;
        EBX ← Reserved;
        ECX ← Extended Feature Bits (* See Table 3-17.*);
        EDX ← Extended Feature Bits (* See Table 3-17. *);
    BREAK;
    EAX = 80000002H:
        EAX ← Processor Brand String;
        EBX ← Processor Brand String, continued;
        ECX ← Processor Brand String, continued;
        EDX ← Processor Brand String, continued;
    BREAK;
    EAX = 80000003H:
        EAX ← Processor Brand String, continued;
        EBX ← Processor Brand String, continued;
        ECX ← Processor Brand String, continued;
        EDX ← Processor Brand String, continued;
    BREAK;
    EAX = 80000004H:
        EAX ← Processor Brand String, continued;
        EBX ← Processor Brand String, continued;
        ECX ← Processor Brand String, continued;
        EDX ← Processor Brand String, continued;
    BREAK;
    EAX = 80000005H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
```

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

Flags Affected

None.

Exceptions (All Operating Modes)

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

...

CVTSD2SI—Convert Scalar Double-Precision FP Value to Integer

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
F2 0F 2D /r CVTSD2SI *r32, xmm/m64*	RM	V/V	SSE2	Convert one double-precision floating-point value from *xmm/m64* to one signed doubleword integer *r32*.
F2 REX.W 0F 2D /r CVTSD2SI *r64, xmm/m64*	RM	V/N.E.	SSE2	Convert one double-precision floating-point value from *xmm/m64* to one signed quadword integer sign-extended into *r64*.
VEX.LIG.F2.0F.W0 2D /r VCVTSD2SI r32, xmm1/m64	RM	V/V	AVX	Convert one double precision floating-point value from xmm1/m64 to one signed doubleword integer r32.
VEX.LIG.F2.0F.W1 2D /r VCVTSD2SI r64, xmm1/m64	RM	V/N.E.[1]	AVX	Convert one double precision floating-point value from xmm1/m64 to one signed quadword integer sign-extended into r64.

NOTES:
1. Encoding the VEX prefix with VEX.W=1 in non-64-bit mode is ignored.

Instruction Operand Encoding

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

Description

Converts a double-precision floating-point value in the source operand (second operand) to a signed doubleword integer in the destination operand (first operand). The source operand can be an XMM register or a 64-bit memory location. The destination operand is a general-purpose register. When the source operand is an XMM register, the double-precision floating-point value is contained in the low quadword of the register.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register.

If a converted result exceeds the range limits of signed doubleword integer (in non-64-bit modes or 64-bit mode with REX.W/VEX.W=0), the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value (80000000H) is returned.

If a converted result exceeds the range limits of signed quadword integer (in 64-bit mode and REX.W/VEX.W = 1), the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value (80000000_00000000H) is returned.

Legacy SSE instructions: Use of the REX.W prefix promotes the instruction to 64-bit operation. See the summary chart at the beginning of this section for encoding data and limits.

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

Operation

```
IF 64-Bit Mode and OperandSize = 64
    THEN
        DEST[63:0] ← Convert_Double_Precision_Floating_Point_To_Integer64(SRC[63:0]);
    ELSE
        DEST[31:0] ← Convert_Double_Precision_Floating_Point_To_Integer32(SRC[63:0]);
FI;
```

Intel C/C++ Compiler Intrinsic Equivalent

int _mm_cvtsd_si32(__m128d a)

__int64 _mm_cvtsd_si64(__m128d a)

SIMD Floating-Point Exceptions

Invalid, Precision.

Other Exceptions

See Exceptions Type 3; additionally

#UD	If VEX.vvvv ≠ 1111B.

...

CVTTSD2SI—Convert with Truncation Scalar Double-Precision FP Value to Signed Integer

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
F2 0F 2C /r CVTTSD2SI *r32, xmm/m64*	RM	V/V	SSE2	Convert one double-precision floating-point value from *xmm/m64* to one signed doubleword integer in *r32* using truncation.
F2 REX.W 0F 2C /r CVTTSD2SI *r64, xmm/m64*	RM	V/N.E.	SSE2	Convert one double precision floating-point value from *xmm/m64* to one signedquadword integer in *r64* using truncation.
VEX.LIG.F2.0F.W0 2C /r VCVTTSD2SI r32, xmm1/m64	RM	V/V	AVX	Convert one double-precision floating-point value from xmm1/m64 to one signed doubleword integer in r32 using truncation.
VEX.LIG.F2.0F.W1 2C /r VCVTTSD2SI r64, xmm1/m64	RM	V/N.E.[1]	AVX	Convert one double precision floating-point value from xmm1/m64 to one signed quadword integer in r64 using truncation.

NOTES:
1. Encoding the VEX prefix with VEX.W=1 in non-64-bit mode is ignored.

Instruction Operand Encoding

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

Description

Converts a double-precision floating-point value in the source operand (second operand) to a signed doubleword integer (or signed quadword integer if operand size is 64 bits) in the destination operand (first operand). The source operand can be an XMM register or a 64-bit memory location. The destination operand is a general purpose register. When the source operand is an XMM register, the double-precision floating-point value is contained in the low quadword of the register.

When a conversion is inexact, a truncated (round toward zero) result is returned.

If a converted result exceeds the range limits of signed doubleword integer (in non-64-bit modes or 64-bit mode with REX.W/VEX.W=0), the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value (80000000H) is returned.

If a converted result exceeds the range limits of signed quadword integer (in 64-bit mode and REX.W/VEX.W = 1), the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value (80000000_00000000H) is returned.

Legacy SSE instructions: In 64-bit mode, Use of the REX.W prefix promotes the instruction to 64-bit operation. See the summary chart at the beginning of this section for encoding data and limits.

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

Operation

```
IF 64-Bit Mode and OperandSize = 64
    THEN
        DEST[63:0] ← Convert_Double_Precision_Floating_Point_To_
                        Integer64_Truncate(SRC[63:0]);
    ELSE
        DEST[31:0] ← Convert_Double_Precision_Floating_Point_To_
                        Integer32_Truncate(SRC[63:0]);
FI;
```

Intel C/C++ Compiler Intrinsic Equivalent

int _mm_cvttsd_si32(__m128d a)

__int64 _mm_cvttsd_si64(__m128d a)

SIMD Floating-Point Exceptions

Invalid, Precision.

Other Exceptions

See Exceptions Type 3; additionally

#UD	If VEX.vvvv ≠ 1111B.

...

FCOS— Cosine

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 FF	FCOS	Valid	Valid	Replace ST(0) with its approximate cosine.

Description

Computes the approximate cosine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$. The following table shows the results obtained when taking the cosine of various classes of numbers.

Table 3-32 FCOS Results

ST(0) SRC	ST(0) DEST
– ∞	*
– F	–1 to +1
– 0	+ 1
+ 0	+ 1
+ F	– 1 to + 1
+ ∞	*
NaN	NaN

NOTES:

F Means finite floating-point value.

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

If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range. It is up to the program to check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be reduced to the range of the instruction by subtracting an appropriate integer multiple of 2π. However, even within the range -2^{63} to $+2^{63}$, inaccurate results can occur because the finite approximation of π used internally for argument reduction is not sufficient in all cases. Therefore, for accurate results it is safe to apply FCOS only to arguments reduced accurately in software, to a value smaller in absolute value than $3\pi/8$. See the sections titled "Pi" and "Transcendental Instruction Accuracy" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a discussion of the proper value to use for π in performing such reductions.

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

Operation

```
IF |ST(0)| < 2^63
THEN
    C2 ← 0;
    ST(0) ← FCOS(ST(0)); // approximation of cosine
ELSE (* Source operand is out-of-range *)
    C2 ← 1;
FI;
```

FPU Flags Affected

C1	Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. Undefined if C2 is 1.
C2	Set to 1 if outside range (-2^{63} < source operand < $+2^{63}$); otherwise, set to 0.
C0, C3	Undefined.

Floating-Point Exceptions

#IS	Stack underflow occurred.
#IA	Source operand is an SNaN value, ∞, or unsupported format.
#D	Source is a denormal value.
#P	Value cannot be represented exactly in destination format.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

FNOP—No Operation

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 D0	FNOP	Valid	Valid	No operation is performed.

Description

Performs no FPU operation. This instruction takes up space in the instruction stream but does not affect the FPU or machine context, except the EIP register and the FPU Instruction Pointer.

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

FPU Flags Affected

C0, C1, C2, C3 undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

FPTAN—Partial Tangent

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 F2	FPTAN	Valid	Valid	Replace ST(0) with its approximate tangent and push 1 onto the FPU stack.

Description

Computes the approximate tangent of the source operand in register ST(0), stores the result in ST(0), and pushes a 1.0 onto the FPU register stack. The source operand must be given in radians and must be less than $\pm 2^{63}$. The following table shows the unmasked results obtained when computing the partial tangent of various classes of numbers, assuming that underflow does not occur.

Table 3-42 FPTAN Results

ST(0) SRC	ST(0) DEST
$-\infty$	*
−F	−F to +F
−0	-0
+0	+0
+F	−F to +F
$+\infty$	*
NaN	NaN

NOTES:

F Means finite floating-point value.

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

If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range. It is up to the program to check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be reduced to the range of the instruction by subtracting an appropriate integer multiple of 2π. However, even within the range -2^{63} to $+2^{63}$, inaccurate results can occur because the finite approximation of π used internally for argument reduction is not sufficient in all cases. Therefore, for accurate results it is safe to apply FPTAN only to arguments reduced accurately in software, to a value smaller in absolute value than $3\pi/8$. See the sections titled "Pi" and "Transcendental Instruction Accuracy" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a discussion of the proper value to use for π in performing such reductions.

The value 1.0 is pushed onto the register stack after the tangent has been computed to maintain compatibility with the Intel 8087 and Intel287 math coprocessors. This operation also simplifies the calculation of other trigonometric functions. For instance, the cotangent (which is the reciprocal of the tangent) can be computed by executing a FDIVR instruction after the FPTAN instruction.

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

Operation

```
IF ST(0) < 2^63
    THEN
        C2 ← 0;
        ST(0) ← fptan(ST(0)); // approximation of tan
        TOP ← TOP – 1;
        ST(0) ← 1.0;
    ELSE (* Source operand is out-of-range *)
        C2 ← 1;
FI;
```

FPU Flags Affected

C1	Set to 0 if stack underflow occurred; set to 1 if stack overflow occurred. Set if result was rounded up; cleared otherwise.
C2	Set to 1 if outside range (-2^{63} < source operand < $+2^{63}$); otherwise, set to 0.
C0, C3	Undefined.

Floating-Point Exceptions

#IS	Stack underflow or overflow occurred.
#IA	Source operand is an SNaN value, ∞, or unsupported format.
#D	Source operand is a denormal value.
#U	Result is too small for destination format.
#P	Value cannot be represented exactly in destination format.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

FSIN—Sine

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 FE	FSIN	Valid	Valid	Replace ST(0) with the approximate of its sine.

Description

Computes an approximation of the sine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$. The following table shows the results obtained when taking the sine of various classes of numbers, assuming that underflow does not occur.

Table 3-44 FSIN Results

SRC (ST(0))	DEST (ST(0))
– ∞	*
– F	– 1 to + 1
– 0	–0
+ 0	+ 0
+ F	– 1 to +1
+ ∞	*
NaN	NaN

NOTES:

F Means finite floating-point value.

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

If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range. It is up to the program to check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be reduced to the range of the instruction by subtracting an appropriate integer multiple of 2π. However, even within the range -2^{63} to $+2^{63}$, inaccurate results can occur because the finite approximation of π used internally for argument reduction is not sufficient in all cases. Therefore, for accurate results it is safe to apply FSIN only to arguments reduced accurately in software, to a value smaller in absolute value than $3\pi/4$. See the sections titled "Pi" and "Transcendental Instruction Accuracy" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a discussion of the proper value to use for π in performing such reductions.

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

Operation

```
IF -2^63 < ST(0) < 2^63
    THEN
        C2 ← 0;
        ST(0) ← fsin(ST(0)); // approximation of the mathematical sin function
    ELSE (* Source operand out of range *)
        C2 ← 1;
FI;
```

FPU Flags Affected

C1	Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C2	Set to 1 if outside range (-2^{63} < source operand < $+2^{63}$); otherwise, set to 0.
C0, C3	Undefined.

Floating-Point Exceptions

#IS	Stack underflow occurred.
#IA	Source operand is an SNaN value, ∞, or unsupported format.
#D	Source operand is a denormal value.
#P	Value cannot be represented exactly in destination format.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

FSINCOS—Sine and Cosine

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 FB	FSINCOS	Valid	Valid	Compute the sine and cosine of ST(0); replace ST(0) with the approximate sine, and push the approximate cosine onto the register stack.

Description

Computes both the approximate sine and the cosine of the source operand in register ST(0), stores the sine in ST(0), and pushes the cosine onto the top of the FPU register stack. (This instruction is faster than executing the FSIN and FCOS instructions in succession.)

The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$. The following table shows the results obtained when taking the sine and cosine of various classes of numbers, assuming that underflow does not occur.

Table 3-45 FSINCOS Results

SRC	DEST	
ST(0)	ST(1) Cosine	ST(0) Sine
$-\infty$	*	*
−F	−1 to +1	−1 to +1
−0	+1	−0
+0	+1	+0
+F	−1 to +1	−1 to +1
$+\infty$	*	*
NaN	NaN	NaN

NOTES:

F Means finite floating-point value.

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

If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range. It is up to the program to check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be reduced to the range of the instruction by subtracting an appropriate integer multiple of 2π. However, even within the range -2^{63} to $+2^{63}$, inaccurate results can occur because the finite approximation of π used internally for argument reduction is not sufficient in all cases. Therefore, for accurate results it is safe to apply FSINCOS only to arguments reduced accurately in software, to a value smaller in absolute value than $3\pi/8$. See the sections titled "Pi" and "Transcendental Instruction Accuracy" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a discussion of the proper value to use for π in performing such reductions.

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

Operation

```
IF ST(0) < 2^63
    THEN
        C2 ← 0;
        TEMP ← fcos(ST(0)); // approximation of cosine
        ST(0) ← fsin(ST(0)); // approximation of sine
        TOP ← TOP – 1;
        ST(0) ← TEMP;
    ELSE (* Source operand out of range *)
        C2 ← 1;
FI;
```

FPU Flags Affected

C1	Set to 0 if stack underflow occurred; set to 1 of stack overflow occurs. Set if result was rounded up; cleared otherwise.
C2	Set to 1 if outside range (-2^{63} < source operand < $+2^{63}$); otherwise, set to 0.
C0, C3	Undefined.

Floating-Point Exceptions

#IS	Stack underflow or overflow occurred.
#IA	Source operand is an SNaN value, ∞, or unsupported format.
#D	Source operand is a denormal value.
#U	Result is too small for destination format.
#P	Value cannot be represented exactly in destination format.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

FXRSTOR—Restore x87 FPU, MMX, XMM, and MXCSR State

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F AE /1 FXRSTOR *m512byte*	M	Valid	Valid	Restore the x87 FPU, MMX, XMM, and MXCSR register state from *m512byte*.
REX.W+ 0F AE /1 FXRSTOR64 *m512byte*	M	Valid	N.E.	Restore the x87 FPU, MMX, XMM, and MXCSR register state from *m512byte*.

Instruction Operand Encoding

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

Description

Reloads the x87 FPU, MMX technology, XMM, and MXCSR registers from the 512-byte memory image specified in the source operand. This data should have been written to memory previously using the FXSAVE instruction, and in the same format as required by the operating modes. The first byte of the data should be located on a 16-byte boundary. There are three distinct layouts of the FXSAVE state map: one for legacy and compatibility mode, a second format for 64-bit mode FXSAVE/FXRSTOR with REX.W=0, and the third format is for 64-bit mode with FXSAVE64/FXRSTOR64. Table 3-52 shows the layout of the legacy/compatibility mode state information in memory and describes the fields in the memory image for the FXRSTOR and FXSAVE instructions. Table 3-55 shows the layout of the 64-bit mode state information when REX.W is set (FXSAVE64/FXRSTOR64). Table 3-56 shows the layout of the 64-bit mode state information when REX.W is clear (FXSAVE/FXRSTOR).

The state image referenced with an FXRSTOR instruction must have been saved using an FXSAVE instruction or be in the same format as required by Table 3-52, Table 3-55, or Table 3-56. Referencing a state image saved with an FSAVE, FNSAVE instruction or incompatible field layout will result in an incorrect state restoration.

The FXRSTOR instruction does not flush pending x87 FPU exceptions. To check and raise exceptions when loading x87 FPU state information with the FXRSTOR instruction, use an FWAIT instruction after the FXRSTOR instruction.

If the OSFXSR bit in control register CR4 is not set, the FXRSTOR instruction may not restore the states of the XMM and MXCSR registers. This behavior is implementation dependent.

If the MXCSR state contains an unmasked exception with a corresponding status flag also set, loading the register with the FXRSTOR instruction will not result in a SIMD floating-point error condition being generated. Only the next occurrence of this unmasked exception will result in the exception being generated.

Bits 16 through 32 of the MXCSR register are defined as reserved and should be set to 0. Attempting to write a 1 in any of these bits from the saved state image will result in a general protection exception (#GP) being generated.

Bytes 464:511 of an FXSAVE image are available for software use. FXRSTOR ignores the content of bytes 464:511 in an FXSAVE state image.

Operation

```
IF 64-Bit Mode
    THEN
        (x87 FPU, MMX, XMM15-XMM0, MXCSR)    Load(SRC);
    ELSE
        (x87 FPU, MMX, XMM7-XMM0, MXCSR) ← Load(SRC);
FI;
```

x87 FPU and SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment. (See alignment check exception [#AC] below.)
	For an attempt to set reserved bits in MXCSR.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
	If CR0.EM[bit 2] = 1.
#UD	If CPUID.01H:EDX.FXSR[bit 24] = 0.
	If instruction is preceded by a LOCK prefix.
#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).
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	For an attempt to set reserved bits in MXCSR.
#NM	If CR0.TS[bit 3] = 1.
	If CR0.EM[bit 2] = 1.
#UD	If CPUID.01H:EDX.FXSR[bit 24] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.
#AC	For unaligned memory reference.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
	For an attempt to set reserved bits in MXCSR.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
	If CR0.EM[bit 2] = 1.
#UD	If CPUID.01H:EDX.FXSR[bit 24] = 0.
	If instruction is preceded by a LOCK prefix.
#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).

...

FXSAVE—Save x87 FPU, MMX Technology, and SSE State

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F AE /0 FXSAVE *m512byte*	M	Valid	Valid	Save the x87 FPU, MMX, XMM, and MXCSR register state to *m512byte*.
REX.W+ 0F AE /0 FXSAVE64 *m512byte*	M	Valid	N.E.	Save the x87 FPU, MMX, XMM, and MXCSR register state to *m512byte*.

Instruction Operand Encoding

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

Description

Saves the current state of the x87 FPU, MMX technology, XMM, and MXCSR registers to a 512-byte memory location specified in the destination operand. The content layout of the 512 byte region depends on whether the processor is operating in non-64-bit operating modes or 64-bit sub-mode of IA-32e mode.

Bytes 464:511 are available to software use. The processor does not write to bytes 464:511 of an FXSAVE area.

The operation of FXSAVE in non-64-bit modes is described first.

Non-64-Bit Mode Operation

Table 3-52 shows the layout of the state information in memory when the processor is operating in legacy modes.

Table 3-52 Non-64-bit-Mode Layout of FXSAVE and FXRSTOR Memory Region

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
Rsvd	FPU CS	FPU IP		FOP	Rsvd	FTW	FSW	FCW	0
MXCSR_MASK		MXCSR		Rsrvd	FPU DS		FPU DP		16
Reserved			ST0/MM0						32
Reserved			ST1/MM1						48
Reserved			ST2/MM2						64
Reserved			ST3/MM3						80
Reserved			ST4/MM4						96
Reserved			ST5/MM5						112
Reserved			ST6/MM6						128
Reserved			ST7/MM7						144
XMM0									160
XMM1									176
XMM2									192
XMM3									208
XMM4									224
XMM5									240
XMM6									256
XMM7									272
Reserved									288
Reserved									304
Reserved									320
Reserved									336
Reserved									352
Reserved									368
Reserved									384
Reserved									400
Reserved									416
Reserved									432
Reserved									448
Available									464
Available									480
Available									496

The destination operand contains the first byte of the memory image, and it must be aligned on a 16-byte boundary. A misaligned destination operand will result in a general-protection (#GP) exception being generated (or in some cases, an alignment check exception [#AC]).

The FXSAVE instruction is used when an operating system needs to perform a context switch or when an exception handler needs to save and examine the current state of the x87 FPU, MMX technology, and/or XMM and MXCSR registers.

The fields in Table 3-52 are defined in Table 3-53.

Table 3-53 Field Definitions

Field	Definition
FCW	x87 FPU Control Word (16 bits). See Figure 8-6 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the x87 FPU control word.
FSW	x87 FPU Status Word (16 bits). See Figure 8-4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the x87 FPU status word.
Abridged FTW	x87 FPU Tag Word (8 bits). The tag information saved here is abridged, as described in the following paragraphs.
FOP	x87 FPU Opcode (16 bits). The lower 11 bits of this field contain the opcode, upper 5 bits are reserved. See Figure 8-8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the x87 FPU opcode field.
FPU IP	x87 FPU Instruction Pointer Offset (32 bits). The contents of this field differ depending on the current addressing mode (32-bit or 16-bit) of the processor when the FXSAVE instruction was executed: 32-bit mode — 32-bit IP offset. 16-bit mode — low 16 bits are IP offset; high 16 bits are reserved. See "x87 FPU Instruction and Operand (Data) Pointers" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a description of the x87 FPU instruction pointer.
FPU CS	x87 FPU Instruction Pointer Selector (16 bits). If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 1, the processor deprecates the FPU CS and FPU DS values, and this field is saved as 0000H.
FPU DP	x87 FPU Instruction Operand (Data) Pointer Offset (32 bits). The contents of this field differ depending on the current addressing mode (32-bit or 16-bit) of the processor when the FXSAVE instruction was executed: 32-bit mode — 32-bit DP offset. 16-bit mode — low 16 bits are DP offset; high 16 bits are reserved. See "x87 FPU Instruction and Operand (Data) Pointers" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a description of the x87 FPU operand pointer.
FPU DS	x87 FPU Instruction Operand (Data) Pointer Selector (16 bits). If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 1, the processor deprecates the FPU CS and FPU DS values, and this field is saved as 0000H.
MXCSR	MXCSR Register State (32 bits). See Figure 10-3 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the MXCSR register. If the OSFXSR bit in control register CR4 is not set, the FXSAVE instruction may not save this register. This behavior is implementation dependent.
MXCSR_ MASK	MXCSR_MASK (32 bits). This mask can be used to adjust values written to the MXCSR register, ensuring that reserved bits are set to 0. Set the mask bits and flags in MXCSR to the mode of operation desired for SSE and SSE2 SIMD floating-point instructions. See "Guidelines for Writing to the MXCSR Register" in Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for instructions for how to determine and use the MXCSR_MASK value.

Table 3-53 Field Definitions (Contd.)

Field	Definition
ST0/MM0 through ST7/ MM7	x87 FPU or MMX technology registers. These 80-bit fields contain the x87 FPU data registers or the MMX technology registers, depending on the state of the processor prior to the execution of the FXSAVE instruction. If the processor had been executing x87 FPU instruction prior to the FXSAVE instruction, the x87 FPU data registers are saved; if it had been executing MMX instructions (or SSE or SSE2 instructions that operated on the MMX technology registers), the MMX technology registers are saved. When the MMX technology registers are saved, the high 16 bits of the field are reserved.
XMM0 through XMM7	XMM registers (128 bits per field). If the OSFXSR bit in control register CR4 is not set, the FXSAVE instruction may not save these registers. This behavior is implementation dependent.

The FXSAVE instruction saves an abridged version of the x87 FPU tag word in the FTW field (unlike the FSAVE instruction, which saves the complete tag word). The tag information is saved in physical register order (R0 through R7), rather than in top-of-stack (TOS) order. With the FXSAVE instruction, however, only a single bit (1 for valid or 0 for empty) is saved for each tag. For example, assume that the tag word is currently set as follows:

R7	R6	R5	R4	R3	R2	R1	R0
11	xx	xx	xx	11	11	11	11

Here, 11B indicates empty stack elements and "xx" indicates valid (00B), zero (01B), or special (10B).

For this example, the FXSAVE instruction saves only the following 8 bits of information:

R7	R6	R5	R4	R3	R2	R1	R0
0	1	1	1	0	0	0	0

Here, a 1 is saved for any valid, zero, or special tag, and a 0 is saved for any empty tag.

The operation of the FXSAVE instruction differs from that of the FSAVE instruction, the as follows:

- FXSAVE instruction does not check for pending unmasked floating-point exceptions. (The FXSAVE operation in this regard is similar to the operation of the FNSAVE instruction).
- After the FXSAVE instruction has saved the state of the x87 FPU, MMX technology, XMM, and MXCSR registers, the processor retains the contents of the registers. Because of this behavior, the FXSAVE instruction cannot be used by an application program to pass a "clean" x87 FPU state to a procedure, since it retains the current state. To clean the x87 FPU state, an application must explicitly execute an FINIT instruction after an FXSAVE instruction to reinitialize the x87 FPU state.
- The format of the memory image saved with the FXSAVE instruction is the same regardless of the current addressing mode (32-bit or 16-bit) and operating mode (protected, real address, or system management). This behavior differs from the FSAVE instructions, where the memory image format is different depending on the addressing mode and operating mode. Because of the different image formats, the memory image saved with the FXSAVE instruction cannot be restored correctly with the FRSTOR instruction, and likewise the state saved with the FSAVE instruction cannot be restored correctly with the FXRSTOR instruction.

The FSAVE format for FTW can be recreated from the FTW valid bits and the stored 80-bit FP data (assuming the stored data was not the contents of MMX technology registers) using Table 3-54.

Table 3-54 Recreating FSAVE Format

Exponent all 1's	Exponent all 0's	Fraction all 0's	J and M bits	FTW valid bit	x87 FTW	
0	0	0	0x	1	Special	10
0	0	0	1x	1	Valid	00
0	0	1	00	1	Special	10
0	0	1	10	1	Valid	00

Table 3-54 Recreating FSAVE Format (Contd.)

Exponent all 1's	Exponent all 0's	Fraction all 0's	J and M bits	FTW valid bit	x87 FTW	
0	1	0	0x	1	Special	10
0	1	0	1x	1	Special	10
0	1	1	00	1	Zero	01
0	1	1	10	1	Special	10
1	0	0	1x	1	Special	10
1	0	0	1x	1	Special	10
1	0	1	00	1	Special	10
1	0	1	10	1	Special	10
For all legal combinations above.				0	Empty	11

The J-bit is defined to be the 1-bit binary integer to the left of the decimal place in the significand. The M-bit is defined to be the most significant bit of the fractional portion of the significand (i.e., the bit immediately to the right of the decimal place).

When the M-bit is the most significant bit of the fractional portion of the significand, it must be 0 if the fraction is all 0's.

IA-32e Mode Operation

In compatibility sub-mode of IA-32e mode, legacy SSE registers, XMM0 through XMM7, are saved according to the legacy FXSAVE map. In 64-bit mode, all of the SSE registers, XMM0 through XMM15, are saved. Additionally, there are two different layouts of the FXSAVE map in 64-bit mode, corresponding to FXSAVE64 (which requires REX.W=1) and FXSAVE (REX.W=0). In the FXSAVE64 map (Table 3-55), the FPU IP and FPU DP pointers are 64-bit wide. In the FXSAVE map for 64-bit mode (Table 3-56), the FPU IP and FPU DP pointers are 32-bits.

Table 3-55 Layout of the 64-bit-mode FXSAVE64 Map (requires REX.W = 1)

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
FPU IP				FOP	Reserved	FTW	FSW	FCW	**0**
MXCSR_MASK		MXCSR		FPU DP					**16**
Reserved		ST0/MM0							**32**
Reserved		ST1/MM1							**48**
Reserved		ST2/MM2							**64**
Reserved		ST3/MM3							**80**
Reserved		ST4/MM4							**96**
Reserved		ST5/MM5							**112**
Reserved		ST6/MM6							**128**
Reserved		ST7/MM7							**144**
XMM0									**160**
XMM1									**176**
XMM2									**192**
XMM3									**208**

Table 3-55 Layout of the 64-bit-mode FXSAVE64 Map (requires REX.W = 1) (Contd.)

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
XMM4									224
XMM5									240
XMM6									256
XMM7									272
XMM8									288
XMM9									304
XMM10									320
XMM11									336
XMM12									352
XMM13									368
XMM14									384
XMM15									400
Reserved									416
Reserved									432
Reserved									448
Available									464
Available									480
Available									496

Table 3-56 Layout of the 64-bit-mode FXSAVE Map (REX.W = 0)

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
Reserved	FPU CS	FPU IP		FOP	Reserved	FTW	FSW	FCW	0
MXCSR_MASK		MXCSR		Reserved	FPU DS		FPU DP		16
Reserved		ST0/MM0							32
Reserved		ST1/MM1							48
Reserved		ST2/MM2							64
Reserved		ST3/MM3							80
Reserved		ST4/MM4							96
Reserved		ST5/MM5							112
Reserved		ST6/MM6							128
Reserved		ST7/MM7							144
XMM0									160
XMM1									176
XMM2									192
XMM3									208

Table 3-56 Layout of the 64-bit-mode FXSAVE Map (REX.W = 0) (Contd.) (Contd.)

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
XMM4									**224**
XMM5									**240**
XMM6									**256**
XMM7									**272**
XMM8									**288**
XMM9									**304**
XMM10									**320**
XMM11									**336**
XMM12									**352**
XMM13									**368**
XMM14									**384**
XMM15									**400**
Reserved									**416**
Reserved									**432**
Reserved									**448**
Available									**464**
Available									**480**
Available									**496**

Operation

```
IF 64-Bit Mode
    THEN
        IF REX.W = 1
            THEN
                DEST ← Save64BitPromotedFxsave(x87 FPU, MMX, XMM15-XMM0,
                MXCSR);
            ELSE
                DEST ← Save64BitDefaultFxsave(x87 FPU, MMX, XMM15-XMM0, MXCSR);
        FI;
    ELSE
        DEST ← SaveLegacyFxsave(x87 FPU, MMX, XMM7-XMM0, MXCSR);
FI;
```

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment. (See the description of the alignment check exception [#AC] below.)
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
	If CR0.EM[bit 2] = 1.
#UD	If CPUID.01H:EDX.FXSR[bit 24] = 0.
#UD	If the LOCK prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 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 16-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.
	If CR0.EM[bit 2] = 1.
#UD	If CPUID.01H:EDX.FXSR[bit 24] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.
#AC	For unaligned memory reference.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
	If CR0.EM[bit 2] = 1.
#UD	If CPUID.01H:EDX.FXSR[bit 24] = 0.
	If the LOCK prefix is used.
#AC	If this exception is disabled a general protection exception (#GP) is signaled if the memory operand is not aligned on a 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).

Implementation Note

The order in which the processor signals general-protection (#GP) and page-fault (#PF) exceptions when they both occur on an instruction boundary is given in Table 5-2 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*. This order vary for FXSAVE for different processor implementations.

...

Jcc—Jump if Condition Is Met

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
77 *cb*	JA *rel8*	D	Valid	Valid	Jump short if above (CF=0 and ZF=0).
73 *cb*	JAE *rel8*	D	Valid	Valid	Jump short if above or equal (CF=0).
72 *cb*	JB *rel8*	D	Valid	Valid	Jump short if below (CF=1).
76 *cb*	JBE *rel8*	D	Valid	Valid	Jump short if below or equal (CF=1 or ZF=1).
72 *cb*	JC *rel8*	D	Valid	Valid	Jump short if carry (CF=1).
E3 *cb*	JCXZ *rel8*	D	N.E.	Valid	Jump short if CX register is 0.
E3 *cb*	JECXZ *rel8*	D	Valid	Valid	Jump short if ECX register is 0.
E3 *cb*	JRCXZ *rel8*	D	Valid	N.E.	Jump short if RCX register is 0.
74 *cb*	JE *rel8*	D	Valid	Valid	Jump short if equal (ZF=1).
7F *cb*	JG *rel8*	D	Valid	Valid	Jump short if greater (ZF=0 and SF=OF).
7D *cb*	JGE *rel8*	D	Valid	Valid	Jump short if greater or equal (SF=OF).
7C *cb*	JL *rel8*	D	Valid	Valid	Jump short if less (SF≠ OF).
7E *cb*	JLE *rel8*	D	Valid	Valid	Jump short if less or equal (ZF=1 or SF≠ OF).
76 *cb*	JNA *rel8*	D	Valid	Valid	Jump short if not above (CF=1 or ZF=1).
72 *cb*	JNAE *rel8*	D	Valid	Valid	Jump short if not above or equal (CF=1).
73 *cb*	JNB *rel8*	D	Valid	Valid	Jump short if not below (CF=0).
77 *cb*	JNBE *rel8*	D	Valid	Valid	Jump short if not below or equal (CF=0 and ZF=0).
73 *cb*	JNC *rel8*	D	Valid	Valid	Jump short if not carry (CF=0).
75 *cb*	JNE *rel8*	D	Valid	Valid	Jump short if not equal (ZF=0).
7E *cb*	JNG *rel8*	D	Valid	Valid	Jump short if not greater (ZF=1 or SF≠ OF).
7C *cb*	JNGE *rel8*	D	Valid	Valid	Jump short if not greater or equal (SF≠ OF).
7D *cb*	JNL *rel8*	D	Valid	Valid	Jump short if not less (SF=OF).
7F *cb*	JNLE *rel8*	D	Valid	Valid	Jump short if not less or equal (ZF=0 and SF=OF).
71 *cb*	JNO *rel8*	D	Valid	Valid	Jump short if not overflow (OF=0).
7B *cb*	JNP *rel8*	D	Valid	Valid	Jump short if not parity (PF=0).
79 *cb*	JNS *rel8*	D	Valid	Valid	Jump short if not sign (SF=0).
75 *cb*	JNZ *rel8*	D	Valid	Valid	Jump short if not zero (ZF=0).
70 *cb*	JO *rel8*	D	Valid	Valid	Jump short if overflow (OF=1).
7A *cb*	JP *rel8*	D	Valid	Valid	Jump short if parity (PF=1).
7A *cb*	JPE *rel8*	D	Valid	Valid	Jump short if parity even (PF=1).
7B *cb*	JPO *rel8*	D	Valid	Valid	Jump short if parity odd (PF=0).
78 *cb*	JS *rel8*	D	Valid	Valid	Jump short if sign (SF=1).
74 *cb*	JZ *rel8*	D	Valid	Valid	Jump short if zero (ZF = 1).
0F 87 *cw*	JA *rel16*	D	N.S.	Valid	Jump near if above (CF=0 and ZF=0). Not supported in 64-bit mode.

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 87 *cd*	JA *rel32*	D	Valid	Valid	Jump near if above (CF=0 and ZF=0).
0F 83 *cw*	JAE *rel16*	D	N.S.	Valid	Jump near if above or equal (CF=0). Not supported in 64-bit mode.
0F 83 *cd*	JAE *rel32*	D	Valid	Valid	Jump near if above or equal (CF=0).
0F 82 *cw*	JB *rel16*	D	N.S.	Valid	Jump near if below (CF=1). Not supported in 64-bit mode.
0F 82 *cd*	JB *rel32*	D	Valid	Valid	Jump near if below (CF=1).
0F 86 *cw*	JBE *rel16*	D	N.S.	Valid	Jump near if below or equal (CF=1 or ZF=1). Not supported in 64-bit mode.
0F 86 *cd*	JBE *rel32*	D	Valid	Valid	Jump near if below or equal (CF=1 or ZF=1).
0F 82 *cw*	JC *rel16*	D	N.S.	Valid	Jump near if carry (CF=1). Not supported in 64-bit mode.
0F 82 *cd*	JC *rel32*	D	Valid	Valid	Jump near if carry (CF=1).
0F 84 *cw*	JE *rel16*	D	N.S.	Valid	Jump near if equal (ZF=1). Not supported in 64-bit mode.
0F 84 *cd*	JE *rel32*	D	Valid	Valid	Jump near if equal (ZF=1).
0F 84 *cw*	JZ *rel16*	D	N.S.	Valid	Jump near if 0 (ZF=1). Not supported in 64-bit mode.
0F 84 *cd*	JZ *rel32*	D	Valid	Valid	Jump near if 0 (ZF=1).
0F 8F *cw*	JG *rel16*	D	N.S.	Valid	Jump near if greater (ZF=0 and SF=OF). Not supported in 64-bit mode.
0F 8F *cd*	JG *rel32*	D	Valid	Valid	Jump near if greater (ZF=0 and SF=OF).
0F 8D *cw*	JGE *rel16*	D	N.S.	Valid	Jump near if greater or equal (SF=OF). Not supported in 64-bit mode.
0F 8D *cd*	JGE *rel32*	D	Valid	Valid	Jump near if greater or equal (SF=OF).
0F 8C *cw*	JL *rel16*	D	N.S.	Valid	Jump near if less (SF≠ OF). Not supported in 64-bit mode.
0F 8C *cd*	JL *rel32*	D	Valid	Valid	Jump near if less (SF≠ OF).
0F 8E *cw*	JLE *rel16*	D	N.S.	Valid	Jump near if less or equal (ZF=1 or SF≠ OF). Not supported in 64-bit mode.
0F 8E *cd*	JLE *rel32*	D	Valid	Valid	Jump near if less or equal (ZF=1 or SF≠ OF).
0F 86 *cw*	JNA *rel16*	D	N.S.	Valid	Jump near if not above (CF=1 or ZF=1). Not supported in 64-bit mode.
0F 86 *cd*	JNA *rel32*	D	Valid	Valid	Jump near if not above (CF=1 or ZF=1).
0F 82 *cw*	JNAE *rel16*	D	N.S.	Valid	Jump near if not above or equal (CF=1). Not supported in 64-bit mode.
0F 82 *cd*	JNAE *rel32*	D	Valid	Valid	Jump near if not above or equal (CF=1).
0F 83 *cw*	JNB *rel16*	D	N.S.	Valid	Jump near if not below (CF=0). Not supported in 64-bit mode.
0F 83 *cd*	JNB *rel32*	D	Valid	Valid	Jump near if not below (CF=0).
0F 87 *cw*	JNBE *rel16*	D	N.S.	Valid	Jump near if not below or equal (CF=0 and ZF=0). Not supported in 64-bit mode.

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 87 *cd*	JNBE *rel32*	D	Valid	Valid	Jump near if not below or equal (CF=0 and ZF=0).
0F 83 *cw*	JNC *rel16*	D	N.S.	Valid	Jump near if not carry (CF=0). Not supported in 64-bit mode.
0F 83 *cd*	JNC *rel32*	D	Valid	Valid	Jump near if not carry (CF=0).
0F 85 *cw*	JNE *rel16*	D	N.S.	Valid	Jump near if not equal (ZF=0). Not supported in 64-bit mode.
0F 85 *cd*	JNE *rel32*	D	Valid	Valid	Jump near if not equal (ZF=0).
0F 8E *cw*	JNG *rel16*	D	N.S.	Valid	Jump near if not greater (ZF=1 or SF≠ OF). Not supported in 64-bit mode.
0F 8E *cd*	JNG *rel32*	D	Valid	Valid	Jump near if not greater (ZF=1 or SF≠ OF).
0F 8C *cw*	JNGE *rel16*	D	N.S.	Valid	Jump near if not greater or equal (SF≠ OF). Not supported in 64-bit mode.
0F 8C *cd*	JNGE *rel32*	D	Valid	Valid	Jump near if not greater or equal (SF≠ OF).
0F 8D *cw*	JNL *rel16*	D	N.S.	Valid	Jump near if not less (SF=OF). Not supported in 64-bit mode.
0F 8D *cd*	JNL *rel32*	D	Valid	Valid	Jump near if not less (SF=OF).
0F 8F *cw*	JNLE *rel16*	D	N.S.	Valid	Jump near if not less or equal (ZF=0 and SF=OF). Not supported in 64-bit mode.
0F 8F *cd*	JNLE *rel32*	D	Valid	Valid	Jump near if not less or equal (ZF=0 and SF=OF).
0F 81 *cw*	JNO *rel16*	D	N.S.	Valid	Jump near if not overflow (OF=0). Not supported in 64-bit mode.
0F 81 *cd*	JNO *rel32*	D	Valid	Valid	Jump near if not overflow (OF=0).
0F 8B *cw*	JNP *rel16*	D	N.S.	Valid	Jump near if not parity (PF=0). Not supported in 64-bit mode.
0F 8B *cd*	JNP *rel32*	D	Valid	Valid	Jump near if not parity (PF=0).
0F 89 *cw*	JNS *rel16*	D	N.S.	Valid	Jump near if not sign (SF=0). Not supported in 64-bit mode.
0F 89 *cd*	JNS *rel32*	D	Valid	Valid	Jump near if not sign (SF=0).
0F 85 *cw*	JNZ *rel16*	D	N.S.	Valid	Jump near if not zero (ZF=0). Not supported in 64-bit mode.
0F 85 *cd*	JNZ *rel32*	D	Valid	Valid	Jump near if not zero (ZF=0).
0F 80 *cw*	JO *rel16*	D	N.S.	Valid	Jump near if overflow (OF=1). Not supported in 64-bit mode.
0F 80 *cd*	JO *rel32*	D	Valid	Valid	Jump near if overflow (OF=1).
0F 8A *cw*	JP *rel16*	D	N.S.	Valid	Jump near if parity (PF=1). Not supported in 64-bit mode.
0F 8A *cd*	JP *rel32*	D	Valid	Valid	Jump near if parity (PF=1).
0F 8A *cw*	JPE *rel16*	D	N.S.	Valid	Jump near if parity even (PF=1). Not supported in 64-bit mode.

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 8A *cd*	JPE *rel32*	D	Valid	Valid	Jump near if parity even (PF=1).
0F 8B *cw*	JPO *rel16*	D	N.S.	Valid	Jump near if parity odd (PF=0). Not supported in 64-bit mode.
0F 8B *cd*	JPO *rel32*	D	Valid	Valid	Jump near if parity odd (PF=0).
0F 88 *cw*	JS *rel16*	D	N.S.	Valid	Jump near if sign (SF=1). Not supported in 64-bit mode.
0F 88 *cd*	JS *rel32*	D	Valid	Valid	Jump near if sign (SF=1).
0F 84 *cw*	JZ *rel16*	D	N.S.	Valid	Jump near if 0 (ZF=1). Not supported in 64-bit mode.
0F 84 *cd*	JZ *rel32*	D	Valid	Valid	Jump near if 0 (ZF=1).

Instruction Operand Encoding

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

Description

Checks the state of one or more of the status flags in the EFLAGS register (CF, OF, PF, SF, and ZF) and, if the flags are in the specified state (condition), performs a jump to the target instruction specified by the destination operand. A condition code (*cc*) is associated with each instruction to indicate the condition being tested for. If the condition is not satisfied, the jump is not performed and execution continues with the instruction following the J*cc* instruction.

The target instruction is specified with a relative offset (a signed offset relative to the current value of the instruction pointer in the EIP register). A relative offset (*rel8*, *rel16*, or *rel32*) is generally specified as a label in assembly code, but at the machine code level, it is encoded as a signed, 8-bit or 32-bit immediate value, which is added to the instruction pointer. Instruction coding is most efficient for offsets of –128 to +127. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared, resulting in a maximum instruction pointer size of 16 bits.

The conditions for each J*cc* mnemonic are given in the "Description" column of the table on the preceding page. The terms "less" and "greater" are used for comparisons of signed integers and the terms "above" and "below" are used for unsigned integers.

Because a particular state of the status flags can sometimes be interpreted in two ways, two mnemonics are defined for some opcodes. For example, the JA (jump if above) instruction and the JNBE (jump if not below or equal) instruction are alternate mnemonics for the opcode 77H.

The J*cc* instruction does not support far jumps (jumps to other code segments). When the target for the conditional jump is in a different segment, use the opposite condition from the condition being tested for the J*cc* instruction, and then access the target with an unconditional far jump (JMP instruction) to the other segment. For example, the following conditional far jump is illegal:

```
JZ FARLABEL;
```

To accomplish this far jump, use the following two instructions:

```
JNZ BEYOND;
JMP FARLABEL;
BEYOND:
```

The JRCXZ, JECXZ and JCXZ instructions differ from other J*cc* instructions because they do not check status flags. Instead, they check RCX, ECX or CX for 0. The register checked is determined by the address-size attribute. These instructions are useful when used at the beginning of a loop that terminates with a conditional loop instruction (such as LOOPNE). They can be used to prevent an instruction sequence from entering a loop when RCX, ECX or CX is 0. This would cause the loop to execute 2^{64}, 2^{32} or 64K times (not zero times).

All conditional jumps are converted to code fetches of one or two cache lines, regardless of jump address or cacheability.

In 64-bit mode, operand size is fixed at 64 bits. JMP Short is RIP = RIP + 8-bit offset sign extended to 64 bits. JMP Near is RIP = RIP + 32-bit offset sign extended to 64-bits.

Operation

```
IF condition
    THEN
        tempEIP ← EIP + SignExtend(DEST);
        IF OperandSize = 16
            THEN tempEIP ← tempEIP AND 0000FFFFH;
        FI;
    IF tempEIP is not within code segment limit
        THEN #GP(0);
        ELSE EIP ← tempEIP
    FI;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

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

...

LDDQU—Load Unaligned Integer 128 Bits

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
F2 0F F0 /r LDDQU *xmm1*, *mem*	RM	V/V	SSE3	Load unaligned data from *mem* and return double quadword in *xmm1*.
VEX.128.F2.0F.WIG F0 /r VLDDQU xmm1, m128	RM	V/V	AVX	Load unaligned packed integer values from mem to xmm1.
VEX.256.F2.0F.WIG F0 /r VLDDQU ymm1, m256	RM	V/V	AVX	Load unaligned packed integer values from mem to ymm1.

Instruction Operand Encoding

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

Description

The instruction is *functionally similar* to (V)MOVDQU ymm/xmm, m256/m128 for loading from memory. That is: 32/16 bytes of data starting at an address specified by the source memory operand (second operand) are fetched from memory and placed in a destination register (first operand). The source operand need not be aligned on a 32/16-byte boundary. Up to 64/32 bytes may be loaded from memory; this is implementation dependent.

This instruction may improve performance relative to (V)MOVDQU if the source operand crosses a cache line boundary. In situations that require the data loaded by (V)LDDQU be modified and stored to the same location, use (V)MOVDQU or (V)MOVDQA instead of (V)LDDQU. To move a double quadword to or from memory locations that are known to be aligned on 16-byte boundaries, use the (V)MOVDQA instruction.

Implementation Notes

- If the source is aligned to a 32/16-byte boundary, based on the implementation, the 32/16 bytes may be loaded more than once. For that reason, the usage of (V)LDDQU should be avoided when using uncached or write-combining (WC) memory regions. For uncached or WC memory regions, keep using (V)MOVDQU.
- This instruction is a replacement for (V)MOVDQU (load) in situations where cache line splits significantly affect performance. It should not be used in situations where store-load forwarding is performance critical. If performance of store-load forwarding is critical to the application, use (V)MOVDQA store-load pairs when data is 256/128-bit aligned or (V)MOVDQU store-load pairs when data is 256/128-bit unaligned.
- If the memory address is not aligned on 32/16-byte boundary, some implementations may load up to 64/32 bytes and return 32/16 bytes in the destination. Some processor implementations may issue multiple loads to access the appropriate 32/16 bytes. Developers of multi-threaded or multi-processor software should be aware that on these processors the loads will be performed in a non-atomic way.
- If alignment checking is enabled (CR0.AM = 1, RFLAGS.AC = 1, and CPL = 3), an alignment-check exception (#AC) may or may not be generated (depending on processor implementation) when the memory address is not aligned on an 8-byte boundary.

In 64-bit mode, use of the REX.R prefix permits this instruction to access additional registers (XMM8-XMM15).

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

Operation

LDDQU (128-bit Legacy SSE version)
DEST[127:0] ← SRC[127:0]
DEST[VLMAX-1:128] (Unmodified)

VLDDQU (VEX.128 encoded version)
DEST[127:0] ← SRC[127:0]
DEST[VLMAX-1:128] ← 0

VLDDQU (VEX.256 encoded version)
DEST[255:0] ← SRC[255:0]

Intel C/C++ Compiler Intrinsic Equivalent

LDDQU: __m128i _mm_lddqu_si128 (__m128i * p);

VLDDQU: __m256i _mm256_lddqu_si256 (__m256i * p);

Numeric Exceptions

None.

Other Exceptions

See Exceptions Type 4;
Note treatment of #AC varies.

...

MOVMSKPD—Extract Packed Double-Precision Floating-Point Sign Mask

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 50 /r MOVMSKPD *reg, xmm*	RM	V/V	SSE2	Extract 2-bit sign mask from *xmm* and store in *reg*. The upper bits of *r32* or *r64* are filled with zeros.
VEX.128.66.0F.WIG 50 /r VMOVMSKPD *reg, xmm2*	RM	V/V	AVX	Extract 2-bit sign mask from *xmm2* and store in reg. The upper bits of *r32* or *r64* are zeroed.
VEX.256.66.0F.WIG 50 /r VMOVMSKPD *reg, ymm2*	RM	V/V	AVX	Extract 4-bit sign mask from *ymm2* and store in reg. The upper bits of *r32* or *r64* are zeroed.

Instruction Operand Encoding

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

Description

Extracts the sign bits from the packed double-precision floating-point values in the source operand (second operand), formats them into a 2-bit mask, and stores the mask in the destination operand (first operand). The

source operand is an XMM register, and the destination operand is a general-purpose register. The mask is stored in the 2 low-order bits of the destination operand. Zero-extend the upper bits of the destination.

In 64-bit mode, the instruction can access additional registers (XMM8-XMM15, R8-R15) when used with a REX.R prefix. The default operand size is 64-bit in 64-bit mode.

128-bit versions: The source operand is a YMM register. The destination operand is a general purpose register.

VEX.256 encoded version: The source operand is a YMM register. The destination operand is a general purpose register.

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

Operation

(V)MOVMSKPD (128-bit versions)

```
DEST[0] ← SRC[63]
DEST[1] ← SRC[127]
IF DEST = r32
    THEN DEST[31:2] ← 0;
    ELSE DEST[63:2] ← 0;
FI
```

VMOVMSKPD (VEX.256 encoded version)

```
DEST[0] ← SRC[63]
DEST[1] ← SRC[127]
DEST[2] ← SRC[191]
DEST[3] ← SRC[255]
IF DEST = r32
    THEN DEST[31:4] ← 0;
    ELSE DEST[63:4] ← 0;
FI
```

Intel C/C++ Compiler Intrinsic Equivalent

MOVMSKPD: int _mm_movemask_pd (__m128d a)

VMOVMSKPD: _mm256_movemask_pd(__m256d a)

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 7; additionally

#UD	If VEX.vvvv ≠ 1111B.

...

10. 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.

...

PMOVMSKB—Move Byte Mask

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F D7 /r[1] PMOVMSKB *reg, mm*	RM	V/V	SSE	Move a byte mask of *mm* to *reg*. The upper bits of r32 or r64 are zeroed
66 0F D7 /r PMOVMSKB *reg, xmm*	RM	V/V	SSE2	Move a byte mask of *xmm* to *reg*. The upper bits of r32 or r64 are zeroed
VEX.128.66.0F.WIG D7 /r VPMOVMSKB *reg, xmm1*	RM	V/V	AVX	Move a byte mask of *xmm1* to *reg*. The upper bits of r32 or r64 are filled with zeros.
VEX.256.66.0F.WIG D7 /r VPMOVMSKB *reg, ymm1*	RM	V/V	AVX2	Move a 32-bit mask of *ymm1* to *reg*. The upper bits of r64 are filled with zeros.

NOTES:

1. See note in Section 2.4, "Instruction Exception Specification" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A* and Section 22.25.3, "Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Instruction Operand Encoding

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

Description

Creates a mask made up of the most significant bit of each byte of the source operand (second operand) and stores the result in the low byte or word of the destination operand (first operand).

The byte mask is 8 bits for 64-bit source operand, 16 bits for 128-bit source operand and 32 bits for 256-bit source operand. The destination operand is a general-purpose register.

In 64-bit mode, the instruction can access additional registers (XMM8-XMM15, R8-R15) when used with a REX.R prefix. The default operand size is 64-bit in 64-bit mode.

Legacy SSE version: The source operand is an MMX technology register.

128-bit Legacy SSE version: The source operand is an XMM register.

VEX.128 encoded version: The source operand is an XMM register.

VEX.256 encoded version: The source operand is a YMM register.

Note: VEX.vvvv is reserved and must be 1111b.

Operation

PMOVMSKB (with 64-bit source operand and r32)

```
r32[0] ← SRC[7];
r32[1] ← SRC[15];
(* Repeat operation for bytes 2 through 6 *)
r32[7] ← SRC[63];
r32[31:8] ← ZERO_FILL;
```

(V)PMOVMSKB (with 128-bit source operand and r32)

```
r32[0] ← SRC[7];
r32[1] ← SRC[15];
(* Repeat operation for bytes 2 through 14 *)
r32[15] ← SRC[127];
r32[31:16] ← ZERO_FILL;
```

VPMOVMSKB (with 256-bit source operand and r32)

```
r32[0] ← SRC[7];
r32[1] ← SRC[15];
(* Repeat operation for bytes 3rd through 31*)
r32[31] ← SRC[255];
```

PMOVMSKB (with 64-bit source operand and r64)

```
r64[0] ← SRC[7];
r64[1] ← SRC[15];
(* Repeat operation for bytes 2 through 6 *)
r64[7] ← SRC[63];
r64[63:8] ← ZERO_FILL;
```

(V)PMOVMSKB (with 128-bit source operand and r64)

```
r64[0] ← SRC[7];
r64[1] ← SRC[15];
(* Repeat operation for bytes 2 through 14 *)
r64[15] ← SRC[127];
r64[63:16] ← ZERO_FILL;
```

VPMOVMSKB (with 256-bit source operand and r64)

```
r64[0] ← SRC[7];
r64[1] ← SRC[15];
(* Repeat operation for bytes 2 through 31*)
r64[31] ← SRC[255];
r64[63:32] ← ZERO_FILL;
```

Intel C/C++ Compiler Intrinsic Equivalent

PMOVMSKB:	int _mm_movemask_pi8(__m64 a)
(V)PMOVMSKB:	int _mm_movemask_epi8 (__m128i a)
VPMOVMSKB:	int _mm256_movemask_epi8 (__m256i a)

Flags Affected

None.

Numeric Exceptions

None.

Other Exceptions

See Exceptions Type 7; additionally

#UD	If VEX.vvvv ≠ 1111B.

...

POPCNT — Return the Count of Number of Bits Set to 1

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F3 0F B8 /r	POPCNT *r16, r/m16*	RM	Valid	Valid	POPCNT on *r/m16*
F3 0F B8 /r	POPCNT *r32, r/m32*	RM	Valid	Valid	POPCNT on *r/m32*
F3 REX.W 0F B8 */r*	POPCNT *r64, r/m64*	RM	Valid	N.E.	POPCNT on *r/m64*

Instruction Operand Encoding

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

Description

This instruction calculates the number of bits set to 1 in the second operand (source) and returns the count in the first operand (a destination register).

Operation

```
Count = 0;
For (i=0; i < OperandSize; i++)
{       IF (SRC[ i] = 1) // i'th bit
        THEN Count++; FI;
}
DEST ← Count;
```

Flags Affected

OF, SF, ZF, AF, CF, PF are all cleared. ZF is set if SRC = 0, otherwise ZF is cleared.

Intel C/C++ Compiler Intrinsic Equivalent

POPCNT: int _mm_popcnt_u32(unsigned int a);

POPCNT: int64_t _mm_popcnt_u64(unsigned __int64 a);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
	If LOCK prefix is used.

Real-Address Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
	If LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
	If LOCK prefix is used.

...

PSHUFB — Packed Shuffle Bytes

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

NOTES:

1. See note in Section 2.4, "Instruction Exception Specification" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A* and Section 22.25.3, "Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Instruction Operand Encoding

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

Description

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

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

Legacy SSE version: Both operands can be MMX registers.

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

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

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

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

Operation

PSHUFB (with 64 bit operands)

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

PSHUFB (with 128 bit operands)

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

VPSHUFB (VEX.128 encoded version)

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

VPSHUFB (VEX.256 encoded version)

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

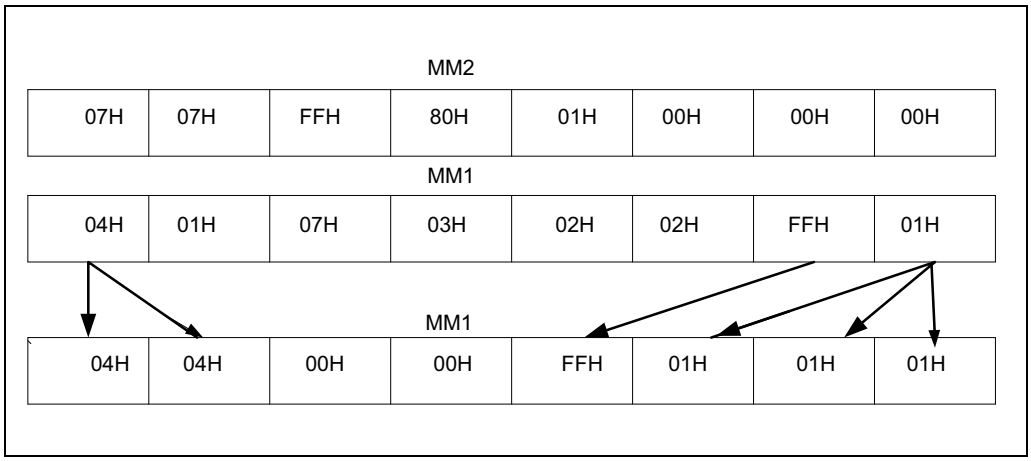

Figure 4-11 PSHUFB with 64-Bit Operands

Intel C/C++ Compiler Intrinsic Equivalent

PSHUFB: __m64 _mm_shuffle_pi8 (__m64 a, __m64 b)

(V)PSHUFB: __m128i _mm_shuffle_epi8 (__m128i a, __m128i b)

VPSHUFB: __m256i _mm256_shuffle_epi8(__m256i a, __m256i b)

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally

#UD If VEX.L = 1.

...

RCPPS—Compute Reciprocals of Packed Single-Precision Floating-Point Values

Opcode*/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 53 /r RCPPS *xmm1, xmm2/m128*	RM	V/V	SSE	Computes the approximate reciprocals of the packed single-precision floating-point values in *xmm2/m128* and stores the results in *xmm1*.
VEX.128.0F.WIG 53 /r VRCPPS *xmm1, xmm2/m128*	RM	V/V	AVX	Computes the approximate reciprocals of packed single-precision values in *xmm2/mem* and stores the results in *xmm1*.
VEX.256.0F.WIG 53 /r VRCPPS *ymm1, ymm2/m256*	RM	V/V	AVX	Computes the approximate reciprocals of packed single-precision values in *ymm2/mem* and stores the results in *ymm1*.

Instruction Operand Encoding

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

Description

Performs a SIMD computation of the approximate reciprocals of the four packed single-precision floating-point values in the source operand (second operand) stores the packed single-precision floating-point results in the destination operand. The source operand can be an XMM register or a 128-bit memory location. The destination operand is an XMM register. See Figure 10-5 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for an illustration of a SIMD single-precision floating-point operation.

The relative error for this approximation is:

$$|\text{Relative Error}| \leq 1.5 * 2^{-12}$$

The RCPPS instruction is not affected by the rounding control bits in the MXCSR register. When a source value is a 0.0, an ∞ of the sign of the source value is returned. A denormal source value is treated as a 0.0 (of the same sign). Tiny results (see Section 4.9.1.5, "Numeric Underflow Exception (#U)" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*) are always flushed to 0.0, with the sign of the operand. (Input values greater than or equal to $|1.11111111110100000000000B*2^{125}|$ are guaranteed to not produce tiny results; input values less than or equal to $|1.00000000000110000000001B*2^{126}|$ are guaranteed to produce tiny results, which are in turn flushed to 0.0; and input values in between this range may or may not produce tiny results, depending on the implementation.) When a source value is an SNaN or QNaN, the SNaN is converted to a QNaN or the source QNaN is returned.

In 64-bit mode, using a REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15).

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

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

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

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

Operation

RCPPS (128-bit Legacy SSE version)

```
DEST[31:0] ← APPROXIMATE(1/SRC[31:0])
DEST[63:32] ← APPROXIMATE(1/SRC[63:32])
DEST[95:64] ← APPROXIMATE(1/SRC[95:64])
DEST[127:96] ← APPROXIMATE(1/SRC[127:96])
DEST[VLMAX-1:128] (Unmodified)
```

VRCPPS (VEX.128 encoded version)

```
DEST[31:0] ← APPROXIMATE(1/SRC[31:0])
DEST[63:32] ← APPROXIMATE(1/SRC[63:32])
DEST[95:64] ← APPROXIMATE(1/SRC[95:64])
DEST[127:96] ← APPROXIMATE(1/SRC[127:96])
DEST[VLMAX-1:128] ← 0
```

VRCPPS (VEX.256 encoded version)
DEST[31:0] ← APPROXIMATE(1/SRC[31:0])
DEST[63:32] ← APPROXIMATE(1/SRC[63:32])
DEST[95:64] ← APPROXIMATE(1/SRC[95:64])
DEST[127:96] ← APPROXIMATE(1/SRC[127:96])
DEST[159:128] ← APPROXIMATE(1/SRC[159:128])
DEST[191:160] ← APPROXIMATE(1/SRC[191:160])
DEST[223:192] ← APPROXIMATE(1/SRC[223:192])
DEST[255:224] ← APPROXIMATE(1/SRC[255:224])

Intel C/C++ Compiler Intrinsic Equivalent

RCCPS: __m128 _mm_rcp_ps(__m128 a)

RCPPS: __m256 _mm256_rcp_ps (__m256 a);

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4; additionally

#UD If VEX.vvvv ≠ 1111B.

...

RCPSS—Compute Reciprocal of Scalar Single-Precision Floating-Point Values

Opcode*/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 53 /r RCPSS *xmm1*, xmm2/m32	RM	V/V	SSE	Computes the approximate reciprocal of the scalar single-precision floating-point value in *xmm2/m32* and stores the result in *xmm1*.
VEX.NDS.LIG.F3.0F.WIG 53 /r VRCPSS *xmm1, xmm2, xmm3/m32*	RVM	V/V	AVX	Computes the approximate reciprocal of the scalar single-precision floating-point value in *xmm3/m32* and stores the result in *xmm1*. Also, upper single precision floating-point values (bits[127:32]) from *xmm2* are copied to *xmm1*[127:32].

Instruction Operand Encoding

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

Description

Computes of an approximate reciprocal of the low single-precision floating-point value in the source operand (second operand) and stores the single-precision floating-point result in the destination operand. The source operand can be an XMM register or a 32-bit memory location. The destination operand is an XMM register. The three high-order doublewords of the destination operand remain unchanged. See Figure 10-6 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for an illustration of a scalar single-precision floating-point operation.

The relative error for this approximation is:

$$|\text{Relative Error}| \leq 1.5 * 2^{-12}$$

The RCPSS instruction is not affected by the rounding control bits in the MXCSR register. When a source value is a 0.0, an ∞ of the sign of the source value is returned. A denormal source value is treated as a 0.0 (of the same sign). Tiny results (see Section 4.9.1.5, "Numeric Underflow Exception (#U)" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*) are always flushed to 0.0, with the sign of the operand. (Input values greater than or equal to $|1.11111111110100000000000B*2^{125}|$ are guaranteed to not produce tiny results; input values less than or equal to $|1.00000000000110000000001B*2^{126}|$ are guaranteed to produce tiny results, which are in turn flushed to 0.0; and input values in between this range may or may not produce tiny results, depending on the implementation.) When a source value is an SNaN or QNaN, the SNaN is converted to a QNaN or the source QNaN is returned.

In 64-bit mode, using a REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15).

128-bit Legacy SSE version: The first source operand and the destination operand are the same. Bits (VLMAX-1:32) of the corresponding YMM destination register remain unchanged.

VEX.128 encoded version: Bits (VLMAX-1:128) of the destination YMM register are zeroed.

Operation

RCPSS (128-bit Legacy SSE version)

```
DEST[31:0] ← APPROXIMATE(1/SRC[31:0])
DEST[VLMAX-1:32] (Unmodified)
```

VRCPSS (VEX.128 encoded version)

```
DEST[31:0] ← APPROXIMATE(1/SRC2[31:0])
DEST[127:32] ← SRC1[127:32]
DEST[VLMAX-1:128] ← 0
```

Intel C/C++ Compiler Intrinsic Equivalent

RCPSS: __m128 _mm_rcp_ss(__m128 a)

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 5.

...

SYSENTER—Fast System Call

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 34	SYSENTER	NP	Valid	Valid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

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

Description

Executes a fast call to a level 0 system procedure or routine. SYSENTER is a companion instruction to SYSEXIT. The instruction is optimized to provide the maximum performance for system calls from user code running at privilege level 3 to operating system or executive procedures running at privilege level 0.

When executed in IA-32e mode, the SYSENTER instruction transitions the logical processor to 64-bit mode; otherwise, the logical processor remains in protected mode.

Prior to executing the SYSENTER instruction, software must specify the privilege level 0 code segment and code entry point, and the privilege level 0 stack segment and stack pointer by writing values to the following MSRs:

- **IA32_SYSENTER_CS** (MSR address 174H) — The lower 16 bits of this MSR are the segment selector for the privilege level 0 code segment. This value is also used to determine the segment selector of the privilege level 0 stack segment (see the Operation section). This value cannot indicate a null selector.
- **IA32_SYSENTER_EIP** (MSR address 176H) — The value of this MSR is loaded into RIP (thus, this value references the first instruction of the selected operating procedure or routine). In protected mode, only bits 31:0 are loaded.
- **IA32_SYSENTER_ESP** (MSR address 175H) — The value of this MSR is loaded into RSP (thus, this value contains the stack pointer for the privilege level 0 stack). This value cannot represent a non-canonical address. In protected mode, only bits 31:0 are loaded.

These MSRs can be read from and written to using RDMSR/WRMSR. The WRMSR instruction ensures that the IA32_SYSENTER_EIP and IA32_SYSENTER_ESP MSRs always contain canonical addresses.

While SYSENTER loads the CS and SS selectors with values derived from the IA32_SYSENTER_CS MSR, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSENTER instruction does not ensure this correspondence.

The SYSENTER instruction can be invoked from all operating modes except real-address mode.

The SYSENTER and SYSEXIT instructions are companion instructions, but they do not constitute a call/return pair. When executing a SYSENTER instruction, the processor does not save state information for the user code (e.g., the instruction pointer), and neither the SYSENTER nor the SYSEXIT instruction supports passing parameters on the stack.

To use the SYSENTER and SYSEXIT instructions as companion instructions for transitions between privilege level 3 code and privilege level 0 operating system procedures, the following conventions must be followed:

- The segment descriptors for the privilege level 0 code and stack segments and for the privilege level 3 code and stack segments must be contiguous in a descriptor table. This convention allows the processor to compute the segment selectors from the value entered in the SYSENTER_CS_MSR MSR.

- The fast system call "stub" routines executed by user code (typically in shared libraries or DLLs) must save the required return IP and processor state information if a return to the calling procedure is required. Likewise, the operating system or executive procedures called with SYSENTER instructions must have access to and use this saved return and state information when returning to the user code.

The SYSENTER and SYSEXIT instructions were introduced into the IA-32 architecture in the Pentium II processor. The availability of these instructions on a processor is indicated with the SYSENTER/SYSEXIT present (SEP) feature flag returned to the EDX register by the CPUID instruction. An operating system that qualifies the SEP flag must also qualify the processor family and model to ensure that the SYSENTER/SYSEXIT instructions are actually present. For example:

```
IF CPUID SEP bit is set
    THEN IF (Family = 6) and (Model < 3) and (Stepping < 3)
        THEN
            SYSENTER/SYSEXIT_Not_Supported; FI;
        ELSE
            SYSENTER/SYSEXIT_Supported; FI;
FI;
```

When the CPUID instruction is executed on the Pentium Pro processor (model 1), the processor returns a the SEP flag as set, but does not support the SYSENTER/SYSEXIT instructions.

Operation

```
IF CR0.PE = 0 OR IA32_SYSENTER_CS[15:2] = 0 THEN #GP(0); FI;

RFLAGS.VM ← 0;                          (* Ensures protected mode execution *)
RFLAGS.IF ← 0;                          (* Mask interrupts *)
IF in IA-32e mode
    THEN
        RSP ← IA32_SYSENTER_ESP;
        RIP ← IA32_SYSENTER_EIP;
ELSE
        ESP ← IA32_SYSENTER_ESP[31:0];
        EIP ← IA32_SYSENTER_EIP[31:0];
FI;

CS.Selector ← IA32_SYSENTER_CS[15:0] AND FFFCH;
                                        (* Operating system provides CS; RPL forced to 0 *)
(* Set rest of CS to a fixed value *)
CS.Base ← 0;                            (* Flat segment *)
CS.Limit ← FFFFFH;                      (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type ← 11;                           (* Execute/read code, accessed *)
CS.S ← 1;
CS.DPL ← 0;
CS.P ← 1;
IF in IA-32e mode
    THEN
        CS.L ← 1;                       (* Entry is to 64-bit mode *)
        CS.D ← 0;                       (* Required if CS.L = 1 *)
    ELSE
        CS.L ← 0;
        CS.D ← 1;                       (* 32-bit code segment*)
FI;
```

```
CS.G ← 1;                              (* 4-KByte granularity *)
CPL ← 0;

SS.Selector ← CS.Selector + 8;         (* SS just above CS *)
(* Set rest of SS to a fixed value *)
SS.Base ← 0;                           (* Flat segment *)
SS.Limit ← FFFFFH;                     (* With 4-KByte granularity, implies a 4-GByte limit *)
SS.Type ← 3;                           (* Read/write data, accessed *)
SS.S ← 1;
SS.DPL ← 0;
SS.P ← 1;
SS.B ← 1;                              (* 32-bit stack segment*)
SS.G ← 1;                              (* 4-KByte granularity *)
```

Flags Affected

VM, IF (see Operation above)

Protected Mode Exceptions

#GP(0)	If IA32_SYSENTER_CS[15:2] = 0.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	The SYSENTER instruction is not recognized in real-address mode.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

UNPCKLPS—Unpack and Interleave Low Packed Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 14 /r UNPCKLPS *xmm1, xmm2/m128*	RM	V/V	SSE	Unpacks and Interleaves single-precision floating-point values from low quadwords of *xmm1* and *xmm2/mem* into *xmm1*.
VEX.NDS.128.0F.WIG 14 /r VUNPCKLPS *xmm1,xmm2, xmm3/m128*	RVM	V/V	AVX	Unpacks and Interleaves single-precision floating-point values from low quadwords of *xmm2* and *xmm3/m128*.
VEX.NDS.256.0F.WIG 14 /r VUNPCKLPS *ymm1,ymm2,ymm3/m256*	RVM	V/V	AVX	Unpacks and Interleaves single-precision floating-point values from low quadwords of *ymm2* and *ymm3/m256*.

Instruction Operand Encoding

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

Description

Performs an interleaved unpack of the low-order single-precision floating-point values from the source operand (second operand) and the destination operand (first operand). See Figure 4-26. The source operand can be an XMM register or a 128-bit memory location; the destination operand is an XMM register.

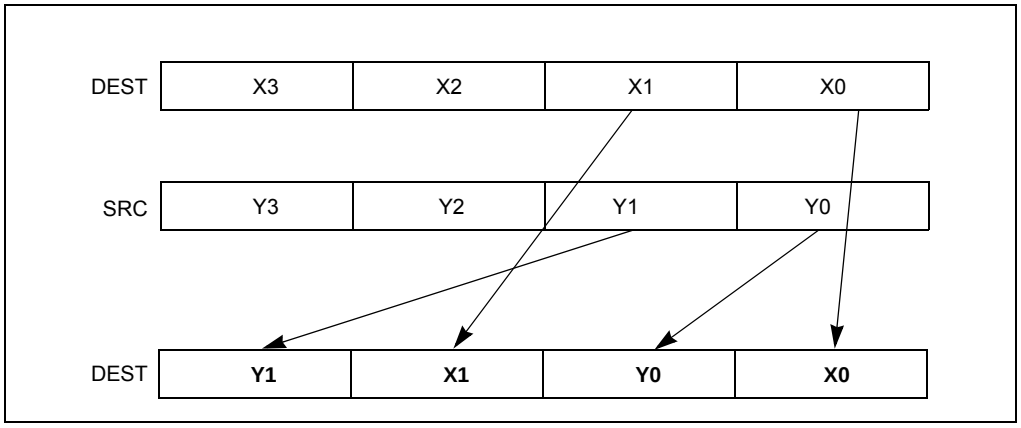

Figure 4-26 UNPCKLPS Instruction Low Unpack and Interleave Operation

When unpacking from a memory operand, an implementation may fetch only the appropriate 64 bits; however, alignment to 16-byte boundary and normal segment checking will still be enforced.

In 64-bit mode, using a REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15).

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

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

Operation

UNPCKLPS (128-bit Legacy SSE version)
DEST[31:0] ← SRC1[31:0]
DEST[63:32] ← SRC2[31:0]
DEST[95:64] ← SRC1[63:32]
DEST[127:96] ← SRC2[63:32]
DEST[VLMAX-1:128] (Unmodified)

VUNPCKLPS (VEX.128 encoded version)
DEST[31:0] ← SRC1[31:0]
DEST[63:32] ← SRC2[31:0]
DEST[95:64] ← SRC1[63:32]
DEST[127:96] ← SRC2[63:32]
DEST[VLMAX-1:128] ← 0

VUNPCKLPS (VEX.256 encoded version)
DEST[31:0] ← SRC1[31:0]
DEST[63:32] ← SRC2[31:0]
DEST[95:64] ← SRC1[63:32]
DEST[127:96] ← SRC2[63:32]
DEST[159:128] ← SRC1[159:128]
DEST[191:160] ← SRC2[159:128]
DEST[223:192] ← SRC1[191:160]
DEST[255:224] ← SRC2[191:160]

Intel C/C++ Compiler Intrinsic Equivalent

UNPCKLPS: __m128 _mm_unpacklo_ps(__m128 a, __m128 b)

UNPCKLPS: __m256 _mm256_unpacklo_ps (__m256 a, __m256 b);

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4.

...

VCVTPS2PH—Convert Single-Precision FP value to 16-bit FP value

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
VEX.256.66.0F3A.W0 1D /r ib VCVTPS2PH *xmm1/m128, ymm2, imm8*	MR	V/V	F16C	Convert eight packed single-precision floating-point value in *ymm2* to packed half-precision (16-bit) floating-point value in *xmm1/mem*. *Imm8* provides rounding controls.
VEX.128.66.0F3A.W0.1D /r ib VCVTPS2PH *xmm1/m64, xmm2, imm8*	MR	V/V	F16C	Convert four packed single-precision floating-point value in *xmm2* to packed half-precision (16-bit) floating-point value in *xmm1/mem*. *Imm8* provides rounding controls.

Instruction Operand Encoding

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

Description

Convert four or eight packed single-precision floating values in first source operand to four or eight packed half-precision (16-bit) floating-point values. The rounding mode is specified using the immediate field (imm8).

Non-zero tiny results are converted to zero, denormals, or the smallest normalized half-precision floating-point value. MXCSR.FTZ is ignored. If a source element is denormal relative to input format with MXCSR.DAZ not set, DM masked and at least one of PM or UM unmasked; a SIMD exception will be raised with DE, UE and PE set.

128-bit version: The source operand is a XMM register. The destination operand is a XMM register or 64-bit memory location. The upper-bits vector register zeroing behavior of VEX prefix encoding still applies if the destination operand is a xmm register. So the upper bits (255:64) of corresponding YMM register are zeroed.

256-bit version: The source operand is a YMM register. The destination operand is a XMM register or 128-bit memory location. The upper-bits vector register zeroing behavior of VEX prefix encoding still applies if the destination operand is a xmm register. So the upper bits (255:128) of the corresponding YMM register are zeroed.

Note: VEX.vvvv is reserved (must be 1111b).

The diagram below illustrates how data is converted from four packed single precision (in 128 bits) to four half precision (in 64 bits) FP values.

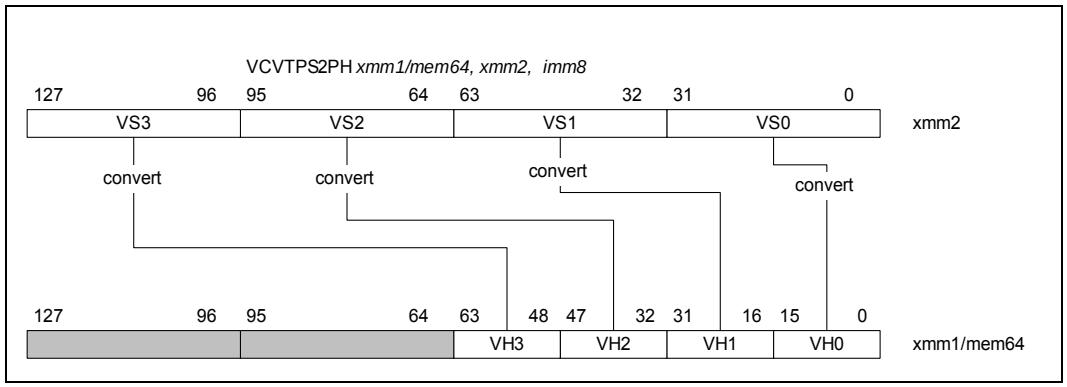

Figure 4-32 VCVTPS2PH (128-bit Version)

The immediate byte defines several bit fields that controls rounding operation. The effect and encoding of RC field are listed in Table 4-17.

Table 4-17 Immediate Byte Encoding for 16-bit Floating-Point Conversion Instructions

Bits	Field Name/value	Description	Comment
Imm[1:0]	RC=00B	Round to nearest even	If Imm[2] = 0
	RC=01B	Round down	
	RC=10B	Round up	
	RC=11B	Truncate	
Imm[2]	MS1=0	Use imm[1:0] for rounding	Ignore MXCSR.RC
	MS1=1	Use MXCSR.RC for rounding	
Imm[7:3]	Ignored	Ignored by processor	

Operation

```
vCvt_s2h(SRC1[31:0])
{
IF Imm[2] = 0
THEN // using Imm[1:0] for rounding control, see Table 4-17
        RETURN Cvt_Single_Precision_To_Half_Precision_FP_Imm(SRC1[31:0]);
ELSE // using MXCSR.RC for rounding control
        RETURN Cvt_Single_Precision_To_Half_Precision_FP_Mxcsr(SRC1[31:0]);
FI;
}
```

VCVTPS2PH (VEX.256 encoded version)

DEST[15:0] ← vCvt_s2h(SRC1[31:0]);
DEST[31:16] ← vCvt_s2h(SRC1[63:32]);
DEST[47:32] ← vCvt_s2h(SRC1[95:64]);
DEST[63:48] ← vCvt_s2h(SRC1[127:96]);
DEST[79:64] ← vCvt_s2h(SRC1[159:128]);
DEST[95:80] ← vCvt_s2h(SRC1[191:160]);
DEST[111:96] ← vCvt_s2h(SRC1[223:192]);
DEST[127:112] ← vCvt_s2h(SRC1[255:224]);
DEST[255:128] ← 0; // if DEST is a register

VCVTPS2PH (VEX.128 encoded version)

DEST[15:0] ← vCvt_s2h(SRC1[31:0]);
DEST[31:16] ← vCvt_s2h(SRC1[63:32]);
DEST[47:32] ← vCvt_s2h(SRC1[95:64]);
DEST[63:48] ← vCvt_s2h(SRC1[127:96]);
DEST[VLMAX-1:64] ←0; // if DEST is a register

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

__m128i _mm_cvtps_ph (__m128 m1, const int imm);

__m128i _mm256_cvtps_ph(__m256 m1, const int imm);

SIMD Floating-Point Exceptions

Invalid, Underflow, Overflow, Precision, Denormal (if MXCSR.DAZ=0);

Other Exceptions

Exceptions Type 11 (do not report #AC); additionally

#UD	If VEX.W=1.

...

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 a single precision floating-point value selected by *imm8* from *xmm3/m128* into ymm2 at the specified destination element specified by *imm8* and zero out destination elements in *ymm1* as indicated in *imm8*.

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.

...

VMASKMOV—Conditional SIMD Packed Loads and Stores

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
VEX.NDS.128.66.0F38.W0 2C /r VMASKMOVPS *xmm1, xmm2, m128*	RVM	V/V	AVX	Conditionally load packed single-precision values from *m128* using mask in *xmm2* and store in *xmm1*.
VEX.NDS.256.66.0F38.W0 2C /r VMASKMOVPS *ymm1, ymm2, m256*	RVM	V/V	AVX	Conditionally load packed single-precision values from *m256* using mask in *ymm2* and store in *ymm1*.
VEX.NDS.128.66.0F38.W0 2D /r VMASKMOVPD *xmm1, xmm2, m128*	RVM	V/V	AVX	Conditionally load packed double-precision values from *m128* using mask in *xmm2* and store in *xmm1*.
VEX.NDS.256.66.0F38.W0 2D /r VMASKMOVPD *ymm1, ymm2, m256*	RVM	V/V	AVX	Conditionally load packed double-precision values from *m256* using mask in *ymm2* and store in *ymm1*.
VEX.NDS.128.66.0F38.W0 2E /r VMASKMOVPS *m128, xmm1, xmm2*	MVR	V/V	AVX	Conditionally store packed single-precision values from *xmm2* using mask in *xmm1*.
VEX.NDS.256.66.0F38.W0 2E /r VMASKMOVPS *m256, ymm1, ymm2*	MVR	V/V	AVX	Conditionally store packed single-precision values from *ymm2* using mask in *ymm1*.
VEX.NDS.128.66.0F38.W0 2F /r VMASKMOVPD *m128, xmm1, xmm2*	MVR	V/V	AVX	Conditionally store packed double-precision values from *xmm2* using mask in *xmm1*.
VEX.NDS.256.66.0F38.W0 2F /r VMASKMOVPD *m256, ymm1, ymm2*	MVR	V/V	AVX	Conditionally store packed double-precision values from *ymm2* using mask in *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
MVR	ModRM:r/m (w)	VEX.vvvv (r)	ModRM:reg (r)	NA

Description

Conditionally moves packed data elements from the second source operand into the corresponding data element of the destination operand, depending on the mask bits associated with each data element. The mask bits are specified in the first source operand.

The mask bit for each data element is the most significant bit of that element in the first source operand. If a mask is 1, the corresponding data element is copied from the second source operand to the destination operand. If the mask is 0, the corresponding data element is set to zero in the load form of these instructions, and unmodified in the store form.

The second source operand is a memory address for the load form of these instruction. The destination operand is a memory address for the store form of these instructions. The other operands are both XMM registers (for VEX.128 version) or YMM registers (for VEX.256 version).

Faults occur only due to mask-bit required memory accesses that caused the faults. Faults will not occur due to referencing any memory location if the corresponding mask bit for that memory location is 0. For example, no faults will be detected if the mask bits are all zero.

Unlike previous MASKMOV instructions (MASKMOVQ and MASKMOVDQU), a nontemporal hint is not applied to these instructions.

Instruction behavior on alignment check reporting with mask bits of less than all 1s are the same as with mask bits of all 1s.

VMASKMOV should not be used to access memory mapped I/O and un-cached memory as the access and the ordering of the individual loads or stores it does is implementation specific.

In cases where mask bits indicate data should not be loaded or stored paging A and D bits will be set in an implementation dependent way. However, A and D bits are always set for pages where data is actually loaded/stored.

Note: for load forms, the first source (the mask) is encoded in VEX.vvvv; the second source is encoded in rm_field, and the destination register is encoded in reg_field.

Note: for store forms, the first source (the mask) is encoded in VEX.vvvv; the second source register is encoded in reg_field, and the destination memory location is encoded in rm_field.

Operation

VMASKMOVPS -128-bit load

```
DEST[31:0] ← IF (SRC1[31]) Load_32(mem) ELSE 0
DEST[63:32] ← IF (SRC1[63]) Load_32(mem + 4) ELSE 0
DEST[95:64] ← IF (SRC1[95]) Load_32(mem + 8) ELSE 0
DEST[127:97] ← IF (SRC1[127]) Load_32(mem + 12) ELSE 0
DEST[VLMAX-1:128] ← 0
```

VMASKMOVPS - 256-bit load

```
DEST[31:0] ← IF (SRC1[31]) Load_32(mem) ELSE 0
DEST[63:32] ← IF (SRC1[63]) Load_32(mem + 4) ELSE 0
DEST[95:64] ← IF (SRC1[95]) Load_32(mem + 8) ELSE 0
DEST[127:96] ← IF (SRC1[127]) Load_32(mem + 12) ELSE 0
DEST[159:128] ← IF (SRC1[159]) Load_32(mem + 16) ELSE 0
DEST[191:160] ← IF (SRC1[191]) Load_32(mem + 20) ELSE 0
DEST[223:192] ← IF (SRC1[223]) Load_32(mem + 24) ELSE 0
DEST[255:224] ← IF (SRC1[255]) Load_32(mem + 28) ELSE 0
```

VMASKMOVPD - 128-bit load

```
DEST[63:0] ← IF (SRC1[63]) Load_64(mem) ELSE 0
DEST[127:64] ← IF (SRC1[127]) Load_64(mem + 16) ELSE 0
DEST[VLMAX-1:128] ← 0
```

VMASKMOVPD - 256-bit load

```
DEST[63:0] ← IF (SRC1[63]) Load_64(mem) ELSE 0
DEST[127:64] ← IF (SRC1[127]) Load_64(mem + 8) ELSE 0
DEST[195:128] ← IF (SRC1[191]) Load_64(mem + 16) ELSE 0
DEST[255:196] ← IF (SRC1[255]) Load_64(mem + 24) ELSE 0
```

VMASKMOVPS - 128-bit store

```
IF (SRC1[31]) DEST[31:0] ← SRC2[31:0]
IF (SRC1[63]) DEST[63:32] ← SRC2[63:32]
IF (SRC1[95]) DEST[95:64] ← SRC2[95:64]
IF (SRC1[127]) DEST[127:96] ← SRC2[127:96]
```

VMASKMOVPS - 256-bit store
IF (SRC1[31]) DEST[31:0] ← SRC2[31:0]
IF (SRC1[63]) DEST[63:32] ← SRC2[63:32]
IF (SRC1[95]) DEST[95:64] ← SRC2[95:64]
IF (SRC1[127]) DEST[127:96] ← SRC2[127:96]
IF (SRC1[159]) DEST[159:128] ←SRC2[159:128]
IF (SRC1[191]) DEST[191:160] ← SRC2[191:160]
IF (SRC1[223]) DEST[223:192] ← SRC2[223:192]
IF (SRC1[255]) DEST[255:224] ← SRC2[255:224]

VMASKMOVPD - 128-bit store
IF (SRC1[63]) DEST[63:0] ← SRC2[63:0]
IF (SRC1[127]) DEST[127:64] ←SRC2[127:64]

VMASKMOVPD - 256-bit store
IF (SRC1[63]) DEST[63:0] ← SRC2[63:0]
IF (SRC1[127]) DEST[127:64] ←SRC2[127:64]
IF (SRC1[191]) DEST[191:128] ← SRC2[191:128]
IF (SRC1[255]) DEST[255:192] ← SRC2[255:192]

Intel C/C++ Compiler Intrinsic Equivalent

__m256 _mm256_maskload_ps(float const *a, __m256i mask)

void _mm256_maskstore_ps(float *a, __m256i mask, __m256 b)

__m256d _mm256_maskload_pd(double *a, __m256i mask);

void _mm256_maskstore_pd(double *a, __m256i mask, __m256d b);

__m128 _mm128_maskload_ps(float const *a, __m128i mask)

void _mm128_maskstore_ps(float *a, __m128i mask, __m128 b)

__m128d _mm128_maskload_pd(double *a, __m128i mask);

void _mm128_maskstore_pd(double *a, __m128i mask, __m128d b);

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 6 (No AC# reported for any mask bit combinations);
additionally

#UD	If VEX.W = 1.

...

VPERMILPS — Permute Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.NDS.128.66.0F38.W0 0C /r VPERMILPS *xmm1, xmm2, xmm3/m128*	RVM	V/V	AVX	Permute single-precision floating-point values in *xmm2* using controls from *xmm3/mem* and store result in *xmm1*.
VEX.128.66.0F3A.W0 04 /r ib VPERMILPS *xmm1, xmm2/m128, imm8*	RMI	V/V	AVX	Permute single-precision floating-point values in *xmm2/mem* using controls from *imm8* and store result in *xmm1*.
VEX.NDS.256.66.0F38.W0 0C /r VPERMILPS *ymm1, ymm2, ymm3/m256*	RVM	V/V	AVX	Permute single-precision floating-point values in *ymm2* using controls from *ymm3/mem* and store result in *ymm1*.
VEX.256.66.0F3A.W0 04 /r ib VPERMILPS *ymm1, ymm2/m256, imm8*	RMI	V/V	AVX	Permute single-precision floating-point values in *ymm2/mem* using controls from *imm8* and store result in *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
RMI	ModRM:reg (w)	ModRM:r/m (r)	imm8	NA

Description

(variable control version)

Permute single-precision floating-point values in the first source operand (second operand) using 8-bit control fields in the low bytes of corresponding elements the shuffle control (third operand) and store results in the destination operand (first operand). The first source operand is a YMM register, the second source operand is a YMM register or a 256-bit memory location, and the destination operand is a YMM register.

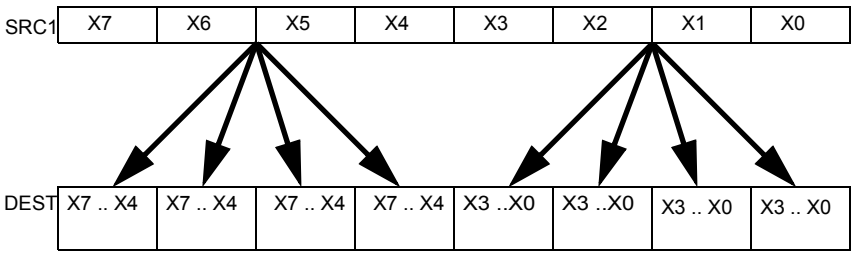

Figure 4-40 VPERMILPS Operation

There is one control byte per destination single-precision element. Each control byte is aligned with the low 8 bits of the corresponding single-precision destination element. Each control byte contains a 2-bit select field (see

Figure 4-41) that determines which of the source elements are selected. Source elements are restricted to lie in the same source 128-bit region as the destination.

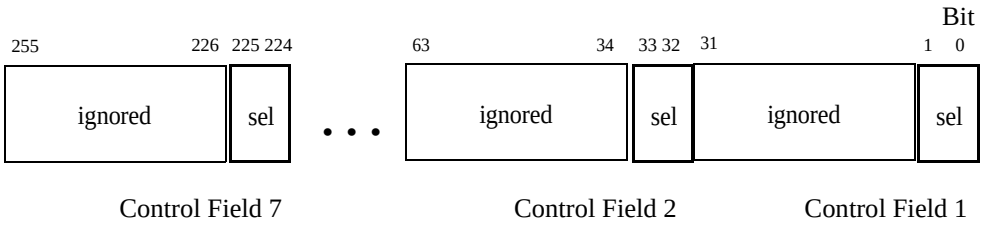

Figure 4-41 VPERMILPS Shuffle Control

(immediate control version)

Permute single-precision floating-point values in the first source operand (second operand) using four 2-bit control fields in the 8-bit immediate and store results in the destination operand (first operand). The source operand is a YMM register or 256-bit memory location and the destination operand is a YMM register. This is similar to a wider version of PSHUFD, just operating on single-precision floating-point values.

Note: For the VEX.128.66.0F3A 04 instruction version, VEX.vvvv is reserved and must be 1111b otherwise instruction will #UD.

Note: For the VEX.256.66.0F3A 04 instruction version, VEX.vvvv is reserved and must be 1111b otherwise instruction will #UD.

Operation

```
Select4(SRC, control) {
CASE (control[1:0]) OF
    0:    TMP ← SRC[31:0];
    1:    TMP ← SRC[63:32];
    2:    TMP ← SRC[95:64];
    3:    TMP ← SRC[127:96];
ESAC;
RETURN TMP
}
```

VPERMILPS (256-bit immediate version)

```
DEST[31:0] ← Select4(SRC1[127:0], imm8[1:0]);
DEST[63:32] ← Select4(SRC1[127:0], imm8[3:2]);
DEST[95:64] ← Select4(SRC1[127:0], imm8[5:4]);
DEST[127:96] ← Select4(SRC1[127:0], imm8[7:6]);
DEST[159:128] ← Select4(SRC1[255:128], imm8[1:0]);
DEST[191:160] ← Select4(SRC1[255:128], imm8[3:2]);
DEST[223:192] ← Select4(SRC1[255:128], imm8[5:4]);
DEST[255:224] ← Select4(SRC1[255:128], imm8[7:6]);
```

VPERMILPS (128-bit immediate version)
DEST[31:0] ← Select4(SRC1[127:0], imm8[1:0]);
DEST[63:32] ← Select4(SRC1[127:0], imm8[3:2]);
DEST[95:64] ← Select4(SRC1[127:0], imm8[5:4]);
DEST[127:96] ← Select4(SRC1[127:0], imm8[7:6]);
DEST[VLMAX-1:128] ← 0

VPERMILPS (256-bit variable version)
DEST[31:0] ← Select4(SRC1[127:0], SRC2[1:0]);
DEST[63:32] ← Select4(SRC1[127:0], SRC2[33:32]);
DEST[95:64] ← Select4(SRC1[127:0], SRC2[65:64]);
DEST[127:96] ← Select4(SRC1[127:0], SRC2[97:96]);
DEST[159:128] ← Select4(SRC1[255:128], SRC2[129:128]);
DEST[191:160] ← Select4(SRC1[255:128], SRC2[161:160]);
DEST[223:192] ← Select4(SRC1[255:128], SRC2[193:192]);
DEST[255:224] ← Select4(SRC1[255:128], SRC2[225:224]);

VPERMILPS (128-bit variable version)
DEST[31:0] ← Select4(SRC1[127:0], SRC2[1:0]);
DEST[63:32] ← Select4(SRC1[127:0], SRC2[33:32]);
DEST[95:64] ← Select4(SRC1[127:0], SRC2[65:64]);
DEST[127:96] ← Select4(SRC1[127:0], SRC2[97:96]);
DEST[VLMAX-1:128] ← 0

Intel C/C++ Compiler Intrinsic Equivalent

VPERMILPS: __m128 _mm_permute_ps (__m128 a, int control);

VPERMILPS: __m256 _mm256_permute_ps (__m256 a, int control);

VPERMILPS: __m128 _mm_permutevar_ps (__m128 a, __m128i control);

VPERMILPS: __m256 _mm256_permutevar_ps (__m256 a, __m256i control);

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 6; additionally

#UD	If VEX.W = 1.

...

XSAVEC—Save Processor Extended States with Compaction

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F C7 /4	XSAVEC *mem*	M	Valid	Valid	Save state components specified by EDX:EAX to *mem* with compaction.
REX.W+ 0F C7 /4	XSAVEC64 *mem*	M	Valid	N.E.	Save state components specified by EDX:EAX to *mem* with compaction.

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), which is the logical-AND of EDX:EAX and XCR0.

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

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

- Execution of XSAVEC is similar to that of XSAVE. XSAVEC differs from XSAVE in that it uses compaction and that it may use the init optimization.
- XSAVEC saves state component *i* if and 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 13.5.4, "Processor Tracking of XSAVE-Managed State.")
- XSAVEC 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").
- XSAVEC writes the logical AND of RFBM and XINUSE to the XSTATE_BV field of the XSAVE header.[2,3] (See Section 13.4.2, "XSAVE Header.") XSAVEC sets bit 63 of the XCOMP_BV field and sets bits 62:0 of that field to RFBM[62:0]. XSAVEC does not write to any parts of the XSAVE header other than the XSTATE_BV and XCOMP_BV fields.
- XSAVEC always uses the compacted format of the extended region of the XSAVE area (see Section 13.4.3, "Extended Region of an XSAVE Area").

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

1. 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, XSAVEC saves SSE state as long as RFBM[1] = 1.
2. Unlike XSAVE and XSAVEOPT, XSAVEC clears bits in the XSTATE_BV field that correspond to bits that are clear in RFBM.
3. 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, XSAVEC sets XSTATE_BV[1] to 1 as long as RFBM[1] = 1.

Operation

```
RFBM ← XCR0 AND EDX:EAX;   /* bitwise logical AND */
COMPMASK ← RFBM OR 80000000_00000000H;

IF RFBM[0] = 1 and XINUSE[0] = 1
    THEN store x87 state into legacy region of XSAVE area;
FI;
IF RFBM[1] = 1 and (XINUSE[1] = 1 or MXCSR ≠ 1F80H)
    THEN store SSE state into legacy region of XSAVE area;
FI;
IF RFBM[2] = 1 AND XINUSE[2] = 1
    THEN store AVX state into extended region of XSAVE area;
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

XSAVEC: void _xsavec(void * , unsigned __int64);

XSAVEC64: void _xsavec64(void * , unsigned __int64);

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If a memory operand is not aligned on a 64-byte boundary, regardless of segment.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CPUID.01H:ECX.XSAVE[bit 26] = 0 or CPUID.(EAX=0DH,ECX=1):EAX.XSAVEC[bit 1] = 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).

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

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.XSAVEC[bit 1] = 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.XSAVEC[bit 1] = 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).

...

11. 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

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 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.

...

12. 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.

...

2.6 EXTENDED CONTROL REGISTERS (INCLUDING XCR0)

If CPUID.01H:ECX.XSAVE[bit 26] is 1, the processor supports one or more **extended control registers** (XCRs). Currently, the only such register defined is XCR0. This register specifies the set of processor state components for which the operating system provides context management, e.g. x87 FPU state, SSE state, AVX state. The OS programs XCR0 to reflect the features for which it provides context management.

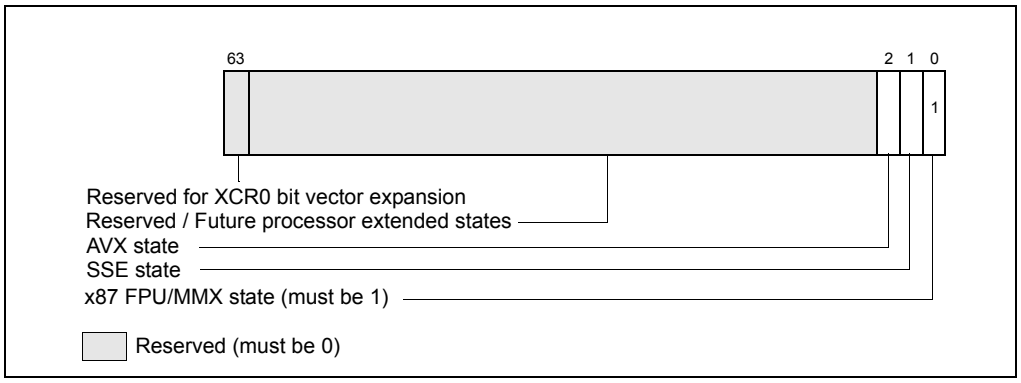

Figure 2-8 XCR0

Software can access XCR0 only if CR4.OSXSAVE[bit 18] = 1. (This bit is also readable as CPUID.01H:ECX.OSXSAVE[bit 27].) Software can use CPUID leaf function 0DH to enumerate the bits in XCR0 that

the processor supports (see CPUID instruction in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Each supported state component is represented by a bit in XCR0. System software enables state components by loading an appropriate bit mask value into XCR0 using the XSETBV instruction.

As each bit in XCR0 (except bit 63) corresponds to a processor state component, XCR0 thus provides support for up to 63 sets of processor state components. Bit 63 of XCR0 is reserved for future expansion and will not represent a processor state component.

Currently, XCR0 defines support for the following state components:

- XCR0.X87 (bit 0): This bit 0 must be 1. An attempt to write 0 to this bit causes a #GP exception.
- XCR0.SSE (bit 1): If 1, the XSAVE feature set can be used to manage MXCSR and the XMM registers (XMM0-XMM15 in 64-bit mode; otherwise XMM0-XMM7).
- XCR0.AVX (bit 2): If 1, AVX instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the YMM registers (YMM0-YMM15 in 64-bit mode; otherwise YMM0-YMM7).

Any attempt to use XSETBV to set a bit reserved in XCR0 for a given processor (as determined by the contents of EAX and EDX after executing CPUID with EAX=0DH, ECX= 0H) results in a #GP exception. An attempt to write 0 to XCR0.x87 (bit 0) also results in a #GP exception, as does any attempt to write 0 to XCR0.SSE (bit 1) and 1 to XCR0.AVX (bit 2).

After reset, all bits (except bit 0) in XCR0 are cleared to zero, XCR0[0] is set to 1.

...

13. 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.

...

4.1 PAGING MODES AND CONTROL BITS

Paging behavior is controlled by the following control bits:

- The WP and PG flags in control register CR0 (bit 16 and bit 31, respectively).
- The PSE, PAE, PGE, PCIDE, SMEP, and SMAP flags in control register CR4 (bit 4, bit 5, bit 7, bit 17, bit 20, and bit 21, respectively).
- The LME and NXE flags in the IA32_EFER MSR (bit 8 and bit 11, respectively).
- The AC flag in the EFLAGS register (bit 18).

Software enables paging by using the MOV to CR0 instruction to set CR0.PG. Before doing so, software should ensure that control register CR3 contains the physical address of the first paging structure that the processor will use for linear-address translation (see Section 4.2) and that structure is initialized as desired. See Table 4-3, Table 4-7, and Table 4-12 for the use of CR3 in the different paging modes.

Section 4.1.1 describes how the values of CR0.PG, CR4.PAE, and IA32_EFER.LME determine whether paging is in use and, if so, which of three paging modes is in use. Section 4.1.2 explains how to manage these bits to establish or make changes in paging modes. Section 4.1.3 discusses how CR0.WP, CR4.PSE, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, and IA32_EFER.NXE modify the operation of the different paging modes.

4.1.1 Three Paging Modes

If CR0.PG = 0, paging is not used. The logical processor treats all linear addresses as if they were physical addresses. CR4.PAE and IA32_EFER.LME are ignored by the processor, as are CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, CR4.SMAP, and IA32_EFER.NXE.

Paging is enabled if CR0.PG = 1. Paging can be enabled only if protection is enabled (CR0.PE = 1). If paging is enabled, one of three paging modes is used. The values of CR4.PAE and IA32_EFER.LME determine which paging mode is used:

- If CR0.PG = 1 and CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 4.3. 32-bit paging uses CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, and CR4.SMAP as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 4.4. PAE paging uses CR0.WP, CR4.PGE, CR4.SMEP, CR4.SMAP, and IA32_EFER.NXE as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1, **IA-32e paging** is used.[1] IA-32e paging is detailed in Section 4.5. IA-32e paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, and IA32_EFER.NXE as described in Section 4.1.3. IA-32e paging is available only on processors that support the Intel 64 architecture.

The three paging modes differ with regard to the following details:

- Linear-address width. The size of the linear addresses that can be translated.
- Physical-address width. The size of the physical addresses produced by paging.
- Page size. The granularity at which linear addresses are translated. Linear addresses on the same page are translated to corresponding physical addresses on the same page.
- Support for execute-disable access rights. In some paging modes, software can be prevented from fetching instructions from pages that are otherwise readable.
- Support for PCIDs. In some paging modes, software can enable a facility by which a logical processor caches information for multiple linear-address spaces. The processor may retain cached information when software switches between different linear-address spaces.

Table 4-1 illustrates the principal differences between the three paging modes.

Table 4-1 Properties of Different Paging Modes

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	Lin.-Addr. Width	Phys.-Addr. Width[1]	Page Sizes	Supports Execute-Disable?	Supports PCIDs?
None	0	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0[2]	32	Up to 40[3]	4 KB 4 MB[4]	No	No
PAE	1	1	0	32	Up to 52	4 KB 2 MB	Yes[5]	No
IA-32e	1	1	1	48	Up to 52	4 KB 2 MB 1 GB[6]	Yes[5]	Yes[7]

1. The LMA flag in the IA32_EFER MSR (bit 10) is a status bit that indicates whether the logical processor is in IA-32e mode (and thus using IA-32e paging). The processor always sets IA32_EFER.LMA to CR0.PG & IA32_EFER.LME. Software cannot directly modify IA32_EFER.LMA; an execution of WRMSR to the IA32_EFER MSR ignores bit 10 of its source operand.

NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.4.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.4 and Section 4.3.
4. 4-MByte pages are used with 32-bit paging only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 4.6.
6. Not all processors that support IA-32e paging support 1-GByte pages; see Section 4.1.4.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 4.10.1.

Because they are used only if IA32_EFER.LME = 0, 32-bit paging and PAE paging is used only in legacy protected mode. Because legacy protected mode cannot produce linear addresses larger than 32 bits, 32-bit paging and PAE paging translate 32-bit linear addresses.

Because it is used only if IA32_EFER.LME = 1, IA-32e paging is used only in IA-32e mode. (In fact, it is the use of IA-32e paging that defines IA-32e mode.) IA-32e mode has two sub-modes:

- Compatibility mode. This mode uses only 32-bit linear addresses. IA-32e paging treats bits 47:32 of such an address as all 0.
- 64-bit mode. While this mode produces 64-bit linear addresses, the processor ensures that bits 63:47 of such an address are identical.[1] IA-32e paging does not use bits 63:48 of such addresses.

...

4.1.3 Paging-Mode Modifiers

Details of how each paging mode operates are determined by the following control bits:

- The WP flag in CR0 (bit 16).
- The PSE, PGE, PCIDE, SMEP, and SMAP flags in CR4 (bit 4, bit 7, bit 17, bit 20, and bit 21 respectively).
- The NXE flag in the IA32_EFER MSR (bit 11).

CR0.WP allows pages to be protected from supervisor-mode writes. If CR0.WP = 0, supervisor-mode write accesses are allowed to linear addresses with read-only access rights; if CR0.WP = 1, they are not. (User-mode write accesses are never allowed to linear addresses with read-only access rights, regardless of the value of CR0.WP.) Section 4.6 explains how access rights are determined, including the definition of supervisor-mode and user-mode accesses.

CR4.PSE enables 4-MByte pages for 32-bit paging. If CR4.PSE = 0, 32-bit paging can use only 4-KByte pages; if CR4.PSE = 1, 32-bit paging can use both 4-KByte pages and 4-MByte pages. See Section 4.3 for more information. (PAE paging and IA-32e paging can use multiple page sizes regardless of the value of CR4.PSE.)

CR4.PGE enables global pages. If CR4.PGE = 0, no translations are shared across address spaces; if CR4.PGE = 1, specified translations may be shared across address spaces. See Section 4.10.2.4 for more information.

CR4.PCIDE enables process-context identifiers (PCIDs) for IA-32e paging (CR4.PCIDE can be 1 only when IA-32e paging is in use). PCIDs allow a logical processor to cache information for multiple linear-address spaces. See Section 4.10.1 for more information.

CR4.SMEP allows pages to be protected from supervisor-mode instruction fetches. If CR4.SMEP = 1, software operating in supervisor mode cannot fetch instructions from linear addresses that are accessible in user mode. Section 4.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

1. Such an address is called **canonical**. Use of a non-canonical linear address in 64-bit mode produces a general-protection exception (#GP(0)); the processor does not attempt to translate non-canonical linear addresses using IA-32e paging.

CR4.SMAP allows pages to be protected from supervisor-mode data accesses. If CR4.SMAP = 1, software operating in supervisor mode cannot access data at linear addresses that are accessible in user mode. Software can override this protection by setting EFLAGS.AC. Section 4.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

IA32_EFER.NXE enables execute-disable access rights for PAE paging and IA-32e paging. If IA32_EFER.NXE = 1, instructions fetches can be prevented from specified linear addresses (even if data reads from the addresses are allowed). Section 4.6 explains how access rights are determined. (IA32_EFER.NXE has no effect with 32-bit paging. Software that wants to use this feature to limit instruction fetches from readable pages must use either PAE paging or IA-32e paging.)

...

4.5 IA-32E PAGING

A logical processor uses IA-32e paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1. With IA-32e paging, linear address are translated using a hierarchy of in-memory paging structures located using the contents of CR3. IA-32e paging translates 48-bit linear addresses to 52-bit physical addresses.[1] Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time.

IA-32e paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the PML4 table. Use of CR3 with IA-32e paging depends on whether process-context identifiers (PCIDs) have been enabled by setting CR4.PCIDE:

- Table 4-12 illustrates how CR3 is used with IA-32e paging if CR4.PCIDE = 0.

Table 4-12 Use of CR3 with IA-32e Paging and CR4.PCIDE = 0

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table during linear-address translation (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table during linear-address translation (see Section 4.9.2)
11:5	Ignored
M–1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation[1]
63:M	Reserved (must be 0)

NOTES:
1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by IA-32e paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.

- Table 4-13 illustrates how CR3 is used with IA-32e paging if CR4.PCIDE = 1.

Table 4-13 Use of CR3 with IA-32e Paging and CR4.PCIDE = 1

Bit Position(s)	Contents
11:0	PCID (see Section 4.10.1)[1]
M–1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation[2]
63:M	Reserved (must be 0)[3]

NOTES:

1. Section 4.9.2 explains how the processor determines the memory type used to access the PML4 table during linear-address translation with CR4.PCIDE = 1.
2. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.
3. See Section 4.10.4.1 for use of bit 63 of the source operand of the MOV to CR3 instruction.

After software modifies the value of CR4.PCIDE, the logical processor immediately begins using CR3 as specified for the new value. For example, if software changes CR4.PCIDE from 1 to 0, the current PCID immediately changes from CR3[11:0] to 000H (see also Section 4.10.4.1). In addition, the logical processor subsequently determines the memory type used to access the PML4 table using CR3.PWT and CR3.PCD, which had been bits 4:3 of the PCID.

IA-32e paging may map linear addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages.[1] Figure 4-8 illustrates the translation process when it produces a 4-KByte page; Figure 4-9 covers the case of a 2-MByte page, and Figure 4-10 the case of a 1-GByte page.

1. Not all processors support 1-GByte pages; see Section 4.1.4.

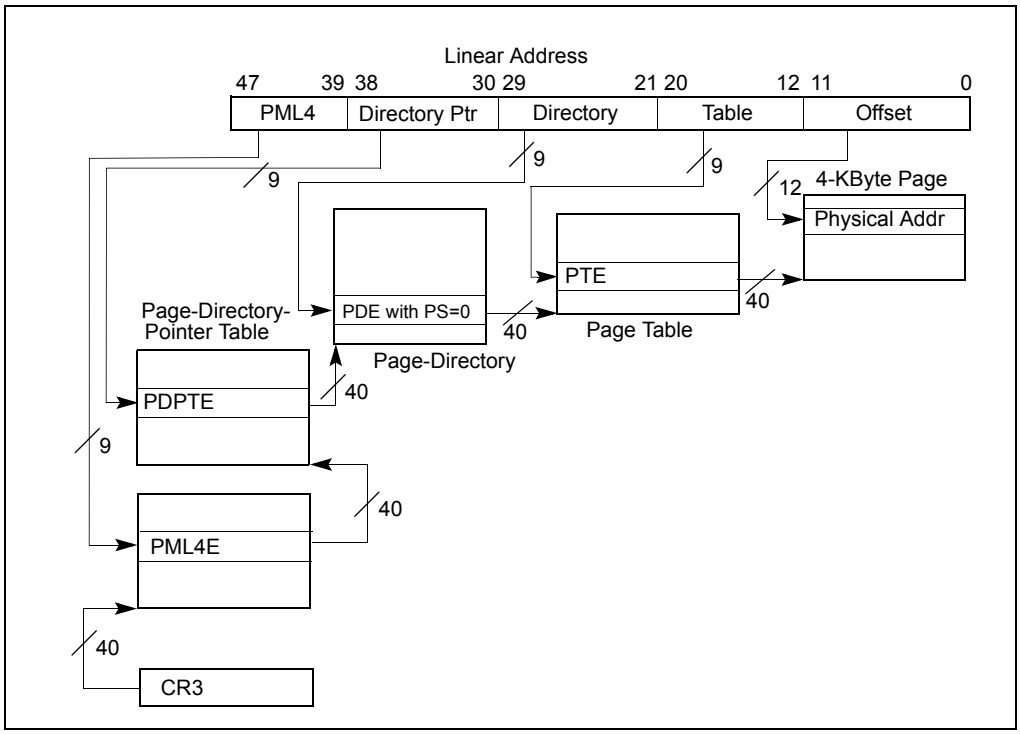

Figure 4-8 Linear-Address Translation to a 4-KByte Page using IA-32e Paging

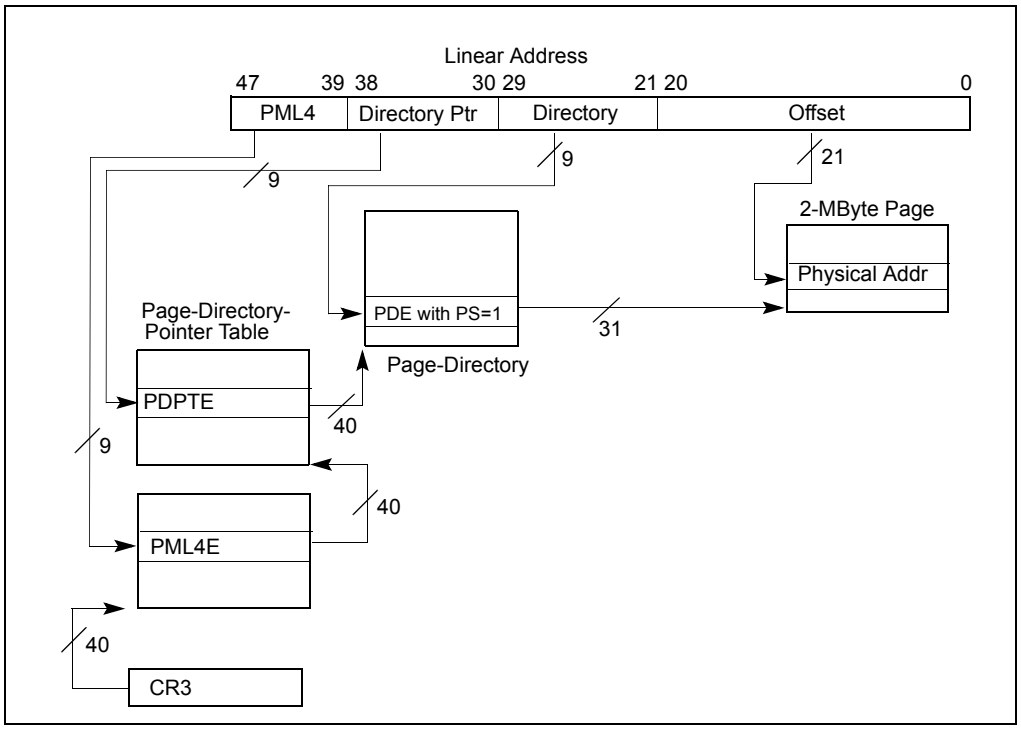

Figure 4-9 Linear-Address Translation to a 2-MByte Page using IA-32e Paging

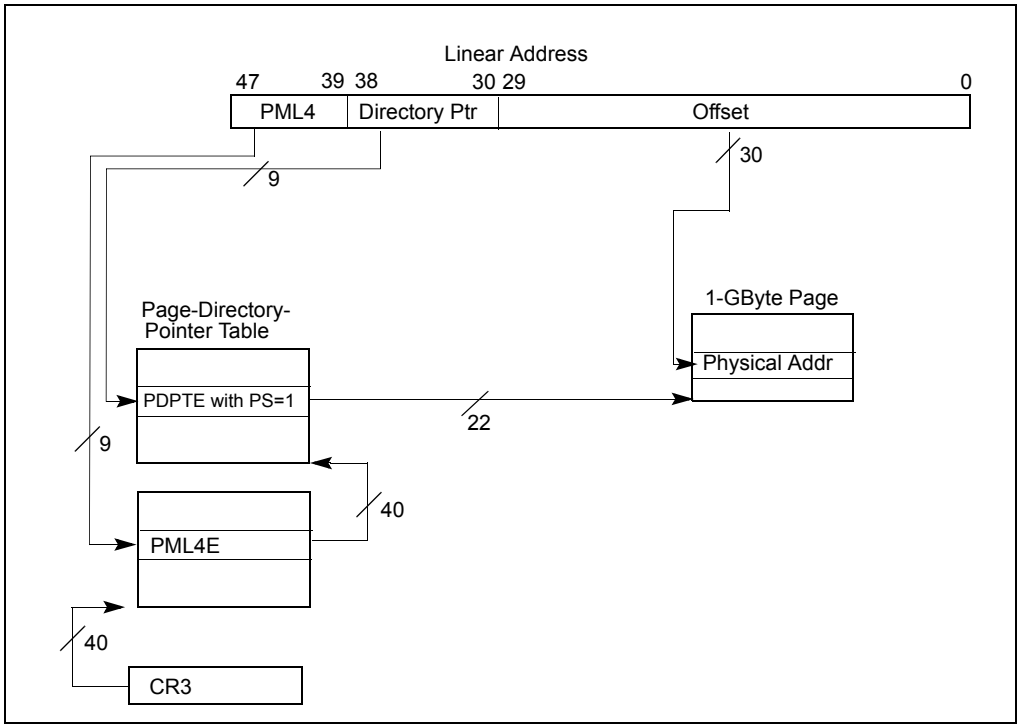

Figure 4-10 Linear-Address Translation to a 1-GByte Page using IA-32e Paging

The following items describe the IA-32e paging process in more detail as well has how the page size is determined.

- A 4-KByte naturally aligned PML4 table is located at the physical address specified in bits 51:12 of CR3 (see Table 4-12). A PML4 table comprises 512 64-bit entries (PML4Es). A PML4E is selected using the physical address defined as follows:
  - Bits 51:12 are from CR3.
  - Bits 11:3 are bits 47:39 of the linear address.
  - Bits 2:0 are all 0.

  Because a PML4E is identified using bits 47:39 of the linear address, it controls access to a 512-GByte region of the linear-address space.
- A 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in bits 51:12 of the PML4E (see Table 4-14). A page-directory-pointer table comprises 512 64-bit entries (PDPTEs). A PDPTE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PML4E.
  - Bits 11:3 are bits 38:30 of the linear address.
  - Bits 2:0 are all 0.

Because a PDPTE is identified using bits 47:30 of the linear address, it controls access to a 1-GByte region of the linear-address space. Use of the PDPTE depends on its PS flag (bit 7):[1]

- If the PDPTE's PS flag is 1, the PDPTE maps a 1-GByte page (see Table 4-15). The final physical address is computed as follows:
  - Bits 51:30 are from the PDPTE.
  - Bits 29:0 are from the original linear address.
- If the PDE's PS flag is 0, a 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of the PDPTE (see Table 4-16). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PDPTE.
  - Bits 11:3 are bits 29:21 of the linear address.
  - Bits 2:0 are all 0.

Because a PDE is identified using bits 47:21 of the linear address, it controls access to a 2-MByte region of the linear-address space. Use of the PDE depends on its PS flag:

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page. The final physical address is computed as shown in Table 4-17.
  - Bits 51:21 are from the PDE.
  - Bits 20:0 are from the original linear address.
- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table 4-18). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PDE.
  - Bits 11:3 are bits 20:12 of the linear address.
  - Bits 2:0 are all 0.

1. The PS flag of a PDPTE is reserved and must be 0 (if the P flag is 1) if 1-GByte pages are not supported. See Section 4.1.4 for how to determine whether 1-GByte pages are supported.

- Because a PTE is identified using bits 47:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-19). The final physical address is computed as follows:
  - Bits 51:12 are from the PTE.
  - Bits 11:0 are from the original linear address.

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 4.7).

The following bits are reserved with IA-32e paging:

- If the P flag of a paging-structure entry is 1, bits 51:MAXPHYADDR are reserved.
- If the P flag of a PML4E is 1, the PS flag is reserved.
- If 1-GByte pages are not supported and the P flag of a PDPTE is 1, the PS flag is reserved.[1]
- If the P flag and the PS flag of a PDPTE are both 1, bits 29:13 are reserved.
- If the P flag and the PS flag of a PDE are both 1, bits 20:13 are reserved.
- If IA32_EFER.NXE = 0 and the P flag of a paging-structure entry is 1, the XD flag (bit 63) is reserved.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

Figure 4-11 gives a summary of the formats of CR3 and the IA-32e paging-structure entries. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are "not present"; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

Table 4-14 Format of an IA-32e PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 512-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Reserved (must be 0)
11:8	Ignored

1. See Section 4.1.4 for how to determine whether 1-GByte pages are supported.

Table 4-14 Format of an IA-32e PML4 Entry (PML4E) that References a Page-Directory-Pointer Table (Contd.)

Bit Position(s)	Contents
M–1:12	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 512-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-15 Format of an IA-32e Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 1-GByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 1-GByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 1-GByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page directory; see Table 4-16)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	Indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)[1]
29:13	Reserved (must be 0)
(M–1):30	Physical address of the 1-GByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:
1. The PAT is supported on all processors that support IA-32e paging.

Table 4-16 Format of an IA-32e Page-Directory-Pointer-Table Entry (PDPTE) that References a Page Directory

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 1-GByte page; see Table 4-15)
11:8	Ignored
(M–1):12	Physical address of 4-KByte aligned page directory referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-17 Format of an IA-32e Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)

Table 4-17 Format of an IA-32e Page-Directory Entry that Maps a 2-MByte Page (Contd.)

Bit Position(s)	Contents
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-18)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
20:13	Reserved (must be 0)
(M–1):21	Physical address of the 2-MByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-18 Format of an IA-32e Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 4-17)
11:8	Ignored
(M–1):12	Physical address of 4-KByte aligned page table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-19 Format of an IA-32e Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

…

4.6 ACCESS RIGHTS

There is a translation for a linear address if the processes described in Section 4.3, Section 4.4.2, and Section 4.5 (depending upon the paging mode) completes and produces a physical address. Whether an access is permitted by a translation is determined by the access rights specified by the paging-structure entries controlling the translation;[1] paging-mode modifiers in CR0, CR4, and the IA32_EFER MSR; EFLAGS.AC; and the mode of the access.

Every access to a linear address is either a **supervisor-mode access** or a **user-mode access**. For all instruction fetches and most data accesses, this distinction is determined by the current privilege level (CPL): accesses made while CPL < 3 are supervisor-mode accesses, while accesses made while CPL = 3 are user-mode accesses.

Some operations implicitly access system data structures with linear addresses; the resulting accesses to those data structures are supervisor-mode accesses regardless of CPL. Examples of such accesses include the following: accesses to the global descriptor table (GDT) or local descriptor table (LDT) to load a segment

1. With PAE paging, the PDPTEs do not determine access rights.

descriptor; accesses to the interrupt descriptor table (IDT) when delivering an interrupt or exception; and accesses to the task-state segment (TSS) as part of a task switch or change of CPL. All these accesses are called **implicit supervisor-mode accesses** regardless of CPL. Other accesses made while CPL < 3 are called **explicit supervisor-mode accesses**.

Access rights are also controlled by the **mode** of a linear address as specified by the paging-structure entries controlling the translation of the linear address. If the U/S flag (bit 2) is 0 in at least one of the paging-structure entries, the address is a **supervisor-mode address**. Otherwise, the address is a **user-mode address**.

The following items detail how paging determines access rights:

- For supervisor-mode accesses:
  - Data may be read (implicitly or explicitly) from any supervisor-mode address.
  - Data reads from user-mode pages.
    Access rights depend on the value of CR4.SMAP:
    - If CR4.SMAP = 0, data may be read from any user-mode address.
    - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
      - If EFLAGS.AC = 1 and the access is explicit, data may be read from any user-mode address.
      - If EFLAGS.AC = 0 or the access is implicit, data may not be read from any user-mode address.
  - Data writes to supervisor-mode addresses.
    Access rights depend on the value of CR0.WP:
    - If CR0.WP = 0, data may be written to any supervisor-mode address.
    - If CR0.WP = 1, data may be written to any supervisor-mode address with a translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation; data may not be written to any supervisor-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
  - Data writes to user-mode addresses.
    Access rights depend on the value of CR0.WP:
    - If CR0.WP = 0, access rights depend on the value of CR4.SMAP:
      - If CR4.SMAP = 0, data may be written to any user-mode address.
      - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
        - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address.
        - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
    - If CR0.WP = 1, access rights depend on the value of CR4.SMAP:
      - If CR4.SMAP = 0, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
      - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
        - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.

        - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
    - Instruction fetches from supervisor-mode addresses.
        - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any supervisor-mode address.
        - For PAE paging or IA-32e paging with IA32_EFER.NXE = 1, instructions may be fetched from any supervisor-mode address with a translation for which the XD flag (bit 63) is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any supervisor-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.
    - Instruction fetches from user-mode addresses.
      Access rights depend on the values of CR4.SMEP:
        - If CR4.SMEP = 0, access writes depend on the paging mode and the value of IA32_EFER.NXE:
            - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
            - For PAE paging or IA-32e paging with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any user-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.
        - If CR4.SMEP = 1, instructions may not be fetched from any user-mode address.
- For user-mode accesses:
    - Data reads.
      Access rights depend on the mode of the linear address:
        - Data may be read from any user-mode address.
        - Data may not be read from any supervisor-mode address.
    - Data writes.
      Access rights depend on the mode of the linear address:
        - Data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation.
        - Data may not be written to any supervisor-mode address.
    - Instruction fetches.
      Access rights depend on the mode of the linear address, the paging mode, and the value of IA32_EFER.NXE:
        - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
        - For PAE paging or IA-32e paging with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation.
        - Instructions may not be fetched from any supervisor-mode address.

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about access rights. The processor may enforce access rights based on the TLBs and paging-structure caches instead of on the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change access rights, the processor might not use that change for a subsequent access to an affected linear address (see Section 4.10.4.3). See Section 4.10.4.2 for how software can ensure that the processor uses the modified access rights.

...

14. 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.

...

6.3.1 External Interrupts

External interrupts are received through pins on the processor or through the local APIC. The primary interrupt pins on Pentium 4, Intel Xeon, P6 family, and Pentium processors are the LINT[1:0] pins, which are connected to the local APIC (see Chapter 10, "Advanced Programmable Interrupt Controller (APIC)"). When the local APIC is enabled, the LINT[1:0] pins can be programmed through the APIC's local vector table (LVT) to be associated with any of the processor's exception or interrupt vectors.

When the local APIC is global/hardware disabled, these pins are configured as INTR and NMI pins, respectively. Asserting the INTR pin signals the processor that an external interrupt has occurred. The processor reads from the system bus the interrupt vector number provided by an external interrupt controller, such as an 8259A (see Section 6.2, "Exception and Interrupt Vectors"). Asserting the NMI pin signals a non-maskable interrupt (NMI), which is assigned to interrupt vector 2.

Table 6-1 Protected-Mode Exceptions and Interrupts

Vector	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[2]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.

Table 6-1 Protected-Mode Exceptions and Interrupts (Contd.)

13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[3]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[4]
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions[5]
20	#VE	Virtualization Exception	Fault	No	EPT violations[6]
21-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT *n* instruction.

NOTES:

1. The UD2 instruction was introduced in the Pentium Pro processor.
2. Processors after the Intel386 processor do not generate this exception.
3. This exception was introduced in the Intel486 processor.
4. This exception was introduced in the Pentium processor and enhanced in the P6 family processors.
5. This exception was introduced in the Pentium III processor.
6. This exception can occur only on processors that support the 1-setting of the "EPT-violation #VE" VM-execution control.

The processor's local APIC is normally connected to a system-based I/O APIC. Here, external interrupts received at the I/O APIC's pins can be directed to the local APIC through the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel® Atom™, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors). The I/O APIC determines the vector number of the interrupt and sends this number to the local APIC. When a system contains multiple processors, processors can also send interrupts to one another by means of the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel Atom, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors).

The LINT[1:0] pins are not available on the Intel486 processor and earlier Pentium processors that do not contain an on-chip local APIC. These processors have dedicated NMI and INTR pins. With these processors, external interrupts are typically generated by a system-based interrupt controller (8259A), with the interrupts being signaled through the INTR pin.

Note that several other pins on the processor can cause a processor interrupt to occur. However, these interrupts are not handled by the interrupt and exception mechanism described in this chapter. These pins include the RESET#, FLUSH#, STPCLK#, SMI#, R/S#, and INIT# pins. Whether they are included on a particular processor is implementation dependent. Pin functions are described in the data books for the individual processors. The SMI# pin is described in Chapter 34, "System Management Mode."

...

Interrupt 8—Double Fault Exception (#DF)

Exception Class Abort.

Description

Indicates that the processor detected a second exception while calling an exception handler for a prior exception. Normally, when the processor detects another exception while trying to call an exception handler, the two exceptions can be handled serially. If, however, the processor cannot handle them serially, it signals the double-fault exception. To determine when two faults need to be signalled as a double fault, the processor divides the exceptions into three classes: benign exceptions, contributory exceptions, and page faults (see Table 6-4).

Table 6-4 Interrupt and Exception Classes

Class	Vector Number	Description
Benign Exceptions and Interrupts	1 2 3 4 5 6 7 9 16 17 18 19 All All	Debug NMI Interrupt Breakpoint Overflow BOUND Range Exceeded Invalid Opcode Device Not Available Coprocessor Segment Overrun Floating-Point Error Alignment Check Machine Check SIMD floating-point INT *n* INTR
Contributory Exceptions	0 10 11 12 13	Divide Error Invalid TSS Segment Not Present Stack Fault General Protection
Page Faults	14 20	Page Fault Virtualization Exception

Table 6-5 shows the various combinations of exception classes that cause a double fault to be generated. A double-fault exception falls in the abort class of exceptions. The program or task cannot be restarted or resumed. The double-fault handler can be used to collect diagnostic information about the state of the machine and/or, when possible, to shut the application and/or system down gracefully or restart the system.

A segment or page fault may be encountered while prefetching instructions; however, this behavior is outside the domain of Table 6-5. Any further faults generated while the processor is attempting to transfer control to the appropriate fault handler could still lead to a double-fault sequence.

Table 6-5 Conditions for Generating a Double Fault

	Second Exception		
First Exception	**Benign**	**Contributory**	**Page Fault**
Benign	Handle Exceptions Serially	Handle Exceptions Serially	Handle Exceptions Serially
Contributory	Handle Exceptions Serially	Generate a Double Fault	Handle Exceptions Serially
Page Fault	Handle Exceptions Serially	Generate a Double Fault	Generate a Double Fault
Double Fault	Handle Exceptions Serially	Enter Shutdown Mode	Enter Shutdown Mode

If another contributory or page fault exception occurs while attempting to call the double-fault handler, the processor enters shutdown mode. This mode is similar to the state following execution of an HLT instruction. In this mode, the processor stops executing instructions until an NMI interrupt, SMI interrupt, hardware reset, or INIT# is received. The processor generates a special bus cycle to indicate that it has entered shutdown mode. Software designers may need to be aware of the response of hardware when it goes into shutdown mode. For example, hardware may turn on an indicator light on the front panel, generate an NMI interrupt to record diagnostic information, invoke reset initialization, generate an INIT initialization, or generate an SMI. If any events are pending during shutdown, they will be handled after an wake event from shutdown is processed (for example, A20M# interrupts).

If a shutdown occurs while the processor is executing an NMI interrupt handler, then only a hardware reset can restart the processor. Likewise, if the shutdown occurs while executing in SMM, a hardware reset must be used to restart the processor.

Exception Error Code

Zero. The processor always pushes an error code of 0 onto the stack of the double-fault handler.

Saved Instruction Pointer

The saved contents of CS and EIP registers are undefined.

Program State Change

A program-state following a double-fault exception is undefined. The program or task cannot be resumed or restarted. The only available action of the double-fault exception handler is to collect all possible context information for use in diagnostics and then close the application and/or shut down or reset the processor.

If the double fault occurs when any portion of the exception handling machine state is corrupted, the handler cannot be invoked and the processor must be reset.

...

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.
- 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 page directory entry are set to 1. See description below of RSVD error code flag.

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.

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.

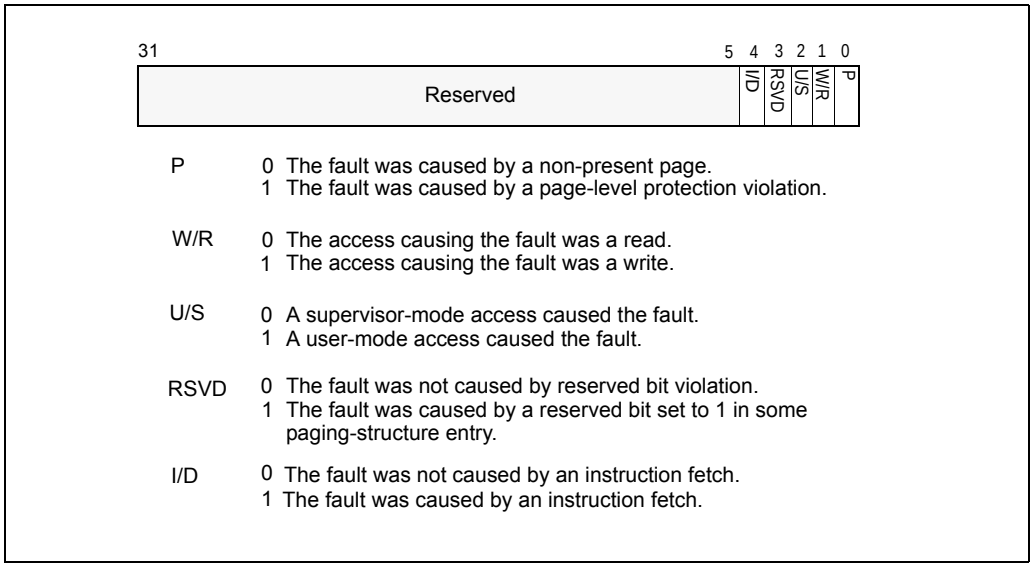

Figure 6-9 Page-Fault Error Code

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:

- 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.

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 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.

...

15. 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.

...

8.4 MULTIPLE-PROCESSOR (MP) INITIALIZATION

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

- It supports controlled booting of multiple processors without requiring dedicated system hardware.

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

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

- **For P6 family or older processors supporting MP operations**— The selection of the BSP and APs (see Section 8.4.1, "BSP and AP Processors") is handled through arbitration on the APIC bus, using BIPI and FIPI messages. These processor generations have CPUID signatures of (family=06H, extended_model=0, model<=0DH), or family <06H. See Section 8.11.1, "Overview of the MP Initialization Process For P6 Family Processors" for a complete discussion of MP initialization for P6 family processors.
- **Early generations of IA processors with family 0FH** — The selection of the BSP and APs (see Section 8.4.1, "BSP and AP Processors") is handled through arbitration on the system bus, using BIPI and FIPI messages (see Section 8.4.3, "MP Initialization Protocol Algorithm for MP Systems"). These processor generations have CPUID signatures of family=0FH, model=0H, stepping<=09H.
- **Later generations of IA processors with family 0FH, and IA processors with system bus** — The selection of the BSP and APs is handled through a special system bus cycle, without using BIPI and FIPI message arbitration (see Section 8.4.3, "MP Initialization Protocol Algorithm for MP Systems"). These processor generations have CPUID signatures of family=0FH with (model=0H, stepping>=0AH) or (model >0, all steppings); or family=06H, extended_model=0, model>=0EH.
- **All other modern IA processor generations supporting MP operations**— The selection of the BSP and APs in the system is handled by platform-specific arrangement of the combination of hardware, BIOS, and/or configuration input options. The basis of the selection mechanism is similar to those of the Later generations of family 0FH and other Intel processor using system bus (see Section 8.4.3, "MP Initialization Protocol Algorithm for MP Systems"). These processor generations have CPUID signatures of family=06H, extended_model>0.

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

...

8.4.3 MP Initialization Protocol Algorithm for MP Systems

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

1. Each logical processor is assigned a unique APIC ID, based on system topology. The unique ID is a 32-bit value if the processor supports CPUID leaf 0BH, otherwise the unique ID is an 8-bit value. (see Section 8.4.5, "Identifying Logical Processors in an MP System").
2. Each logical processor is assigned a unique arbitration priority based on its APIC ID.
3. Each logical processor executes its internal BIST simultaneously with the other logical processors in the system.
4. Upon completion of the BIST, the logical processors use a hardware-defined selection mechanism to select the BSP and the APs from the available logical processors on the system bus. The BSP selection mechanism differs depending on the family, model, and stepping IDs of the processors, as follows:
   — Later generations of IA processors within family 0FH (see Section 8.4), IA processors with system bus (family=06H, extended_model=0, model>=0EH), or all other modern Intel processors (family=06H, extended_model>0):

- The logical processors begin monitoring the BNR# signal, which is toggling. When the BNR# pin stops toggling, each processor attempts to issue a NOP special cycle on the system bus.
- The logical processor with the highest arbitration priority succeeds in issuing a NOP special cycle and is nominated the BSP. This processor sets the BSP flag in its IA32_APIC_BASE MSR, then fetches and begins executing BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).
- The remaining logical processors (that failed in issuing a NOP special cycle) are designated as APs. They leave their BSP flags in the clear state and enter a "wait-for-SIPI state."

— Early generations of IA processors within family 0FH (family=0FH, model=0H, stepping<=09H), P6 family or older processors supporting MP operations (family=06H, extended_model=0, model<=0DH; or family <06H):

- Each processor broadcasts a BIPI to "all including self." The first processor that broadcasts a BIPI (and thus receives its own BIPI vector), selects itself as the BSP and sets the BSP flag in its IA32_APIC_BASE MSR. (See Section 8.11.1, "Overview of the MP Initialization Process For P6 Family Processors" for a description of the BIPI, FIPI, and SIPI messages.)
- The remainder of the processors (which were not selected as the BSP) are designated as APs. They leave their BSP flags in the clear state and enter a "wait-for-SIPI state."
- The newly established BSP broadcasts an FIPI message to "all including self," which the BSP and APs treat as an end of MP initialization signal. Only the processor with its BSP flag set responds to the FIPI message. It responds by fetching and executing the BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).

5. As part of the boot-strap code, the BSP creates an ACPI table and/or an MP table and adds its initial APIC ID to these tables as appropriate.
6. At the end of the boot-strap procedure, the BSP sets a processor counter to 1, then broadcasts a SIPI message to all the APs in the system. Here, the SIPI message contains a vector to the BIOS AP initialization code (at 000VV000H, where VV is the vector contained in the SIPI message).
7. The first action of the AP initialization code is to set up a race (among the APs) to a BIOS initialization semaphore. The first AP to the semaphore begins executing the initialization code. (See Section 8.4.4, "MP Initialization Example," for semaphore implementation details.) As part of the AP initialization procedure, the AP adds its APIC ID number to the ACPI and/or MP tables as appropriate and increments the processor counter by 1. At the completion of the initialization procedure, the AP executes a CLI instruction and halts itself.
8. When each of the APs has gained access to the semaphore and executed the AP initialization code, the BSP establishes a count for the number of processors connected to the system bus, completes executing the BIOS boot-strap code, and then begins executing operating-system boot-strap and start-up code.
9. While the BSP is executing operating-system boot-strap and start-up code, the APs remain in the halted state. In this state they will respond only to INITs, NMIs, and SMIs. They will also respond to snoops and to assertions of the STPCLK# pin.

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

Chapter 35, "Model-Specific Registers (MSRs)," describes how to program the LINT[0:1] pins of the processor's local APICs after an MP configuration has been completed.

...

8.4.4.1 Typical BSP Initialization Sequence

After the BSP and APs have been selected (by means of a hardware protocol, see Section 8.4.3, "MP Initialization Protocol Algorithm for MP Systems"), the BSP begins executing BIOS boot-strap code (POST) at the normal IA-32 architecture starting address (FFFF FFF0H). The boot-strap code typically performs the following operations:

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

```
MOV ESI, APIC_ID; Address of local APIC ID register
MOV EAX, [ESI];
AND EAX, 0FF000000H; Zero out all other bits except APIC ID
MOV BOOT_ID, EAX; Save in memory
```

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

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

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

```
MOV ESI, LVT3;
MOV EAX, [ESI];
AND EAX, FFFFFF00H; Clear out previous vector.
OR EAX, 000000xxH; xx is the 8-bit vector the APIC error handler.
MOV [ESI], EAX;
```

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

processors show up before the 100 millisecond timer expires, the timer can be canceled and skip to step 16. The left-hand-side of the procedure illustrated in Table 8-1 provides an algorithm when the expected processor count is unknown. The right-hand-side of Table 8-1 can be used when the expected processor count is known.

Table 8-1 Broadcast INIT-SIPI-SIPI Sequence and Choice of Timeouts

INIT-SIPI-SIPI when the expected processor count is unknown	INIT-SIPI-SIPI when the expected processor count is known

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

16. Reads and evaluates the COUNT variable and establishes a processor count.
17. If necessary, reconfigures the APIC and continues with the remaining system diagnostics as appropriate.

8.4.4.2 Typical AP Initialization Sequence

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

1. Waits on the BIOS initialization Lock Semaphore. When control of the semaphore is attained, initialization continues.
2. Loads the microcode update into the processor.
3. Initializes the MTRRs (using the same mapping that was used for the BSP).
4. Enables the cache.
5. Executes the CPUID instruction with a value of 0H in the EAX register, then reads the EBX, ECX, and EDX registers to determine if the AP is "GenuineIntel."
6. Executes the CPUID instruction with a value of 1H in the EAX register, then saves the values in the EAX, ECX, and EDX registers in a system configuration space in RAM for use later.
7. Switches to protected mode and ensures that the APIC address space is mapped to the strong uncacheable (UC) memory type.
8. Determines the AP's APIC ID from the local APIC ID register, and adds it to the MP and ACPI tables and optionally to the system configuration space in RAM.

9. Initializes and configures the local APIC by setting bit 8 in the SVR register and setting up the LVT3 (error LVT) for error handling (as described in steps 9 and 10 in Section 8.4.4.1, "Typical BSP Initialization Sequence").
10. Configures the APs SMI execution environment. (Each AP and the BSP must have a different SMBASE address.)
11. Increments the COUNT variable by 1.
12. Releases the semaphore.
13. Executes one of the following:
    — the CLI and HLT instructions (if MONITOR/MWAIT is not supported), or
    — the CLI, MONITOR and MWAIT sequence to enter a deep C-state.
14. Waits for an INIT IPI.

...

16. 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.

...

11.11.2.3 Variable Range MTRRs

The Pentium 4, Intel Xeon, and P6 family processors permit software to specify the memory type for m variable-size address ranges, using a pair of MTRRs for each range. The number m of ranges supported is given in bits 7:0 of the IA32_MTRRCAP MSR (see Figure 11-5 in Section 11.11.1).

The first entry in each pair (IA32_MTRR_PHYSBASEn) defines the base address and memory type for the range; the second entry (IA32_MTRR_PHYSMASKn) contains a mask used to determine the address range. The "n" suffix is in the range 0 through m–1 and identifies a specific register pair.

For P6 family processors, the prefixes for these variable range MTRRs are MTRRphysBase and MTRRphysMask.

Table 11-9 Address Mapping for Fixed-Range MTRRs

Address Range (hexadecimal)								MTRR
63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
70000-7FFFF	60000-6FFFF	50000-5FFFF	40000-4FFFF	30000-3FFFF	20000-2FFFF	10000-1FFFF	00000-0FFFF	IA32_MTRR_FIX64K_00000
9C000 9FFFF	98000-9BFFF	94000-97FFF	90000-93FFF	8C000-8FFFF	88000-8BFFF	84000-87FFF	80000-83FFF	IA32_MTRR_FIX16K_80000
BC000 BFFFF	B8000-BBFFF	B4000-B7FFF	B0000-B3FFF	AC000-AFFFF	A8000-ABFFF	A4000-A7FFF	A0000-A3FFF	IA32_MTRR_FIX16K_A0000
C7000 C7FFF	C6000-C6FFF	C5000-C5FFF	C4000-C4FFF	C3000-C3FFF	C2000-C2FFF	C1000-C1FFF	C0000-C0FFF	IA32_MTRR_FIX4K_C0000
CF000 CFFFF	CE000-CEFFF	CD000-CDFFF	CC000-CCFFF	CB000-CBFFF	CA000-CAFFF	C9000-C9FFF	C8000-C8FFF	IA32_MTRR_FIX4K_C8000
D7000 D7FFF	D6000-D6FFF	D5000-D5FFF	D4000-D4FFF	D3000-D3FFF	D2000-D2FFF	D1000-D1FFF	D0000-D0FFF	IA32_MTRR_FIX4K_D0000

Table 11-9 Address Mapping for Fixed-Range MTRRs (Contd.)

Address Range (hexadecimal)								MTRR
63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
DF000 DFFFF	DE000- DEFFF	DD000- DDFFF	DC000- DCFFF	DB000- DBFFF	DA000- DAFFF	D9000- D9FFF	D8000- D8FFF	IA32_MTRR_ FIX4K_D8000
E7000 E7FFF	E6000- E6FFF	E5000- E5FFF	E4000- E4FFF	E3000- E3FFF	E2000- E2FFF	E1000- E1FFF	E0000- E0FFF	IA32_MTRR_ FIX4K_E0000
EF000 EFFFF	EE000- EEFFF	ED000- EDFFF	EC000- ECFFF	EB000- EBFFF	EA000- EAFFF	E9000- E9FFF	E8000- E8FFF	IA32_MTRR_ FIX4K_E8000
F7000 F7FFF	F6000- F6FFF	F5000- F5FFF	F4000- F4FFF	F3000- F3FFF	F2000- F2FFF	F1000- F1FFF	F0000- F0FFF	IA32_MTRR_ FIX4K_F0000
FF000 FFFFF	FE000- FEFFF	FD000- FDFFF	FC000- FCFFF	FB000- FBFFF	FA000- FAFFF	F9000- F9FFF	F8000- F8FFF	IA32_MTRR_ FIX4K_F8000

...

17. Updates to Chapter 17, Volume 3B

Change bars show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

17.4.9.1 64 Bit Format of the DS Save Area

When DTES64 = 1 (CPUID.1.ECX[2] = 1), the structure of the DS save area is shown in Figure 17-8.

When DTES64 = 0 (CPUID.1.ECX[2] = 0) and IA-32e mode is active, the structure of the DS save area is shown in Figure 17-8. If IA-32e mode is not active the structure of the DS save area is as shown in Figure 17-6.

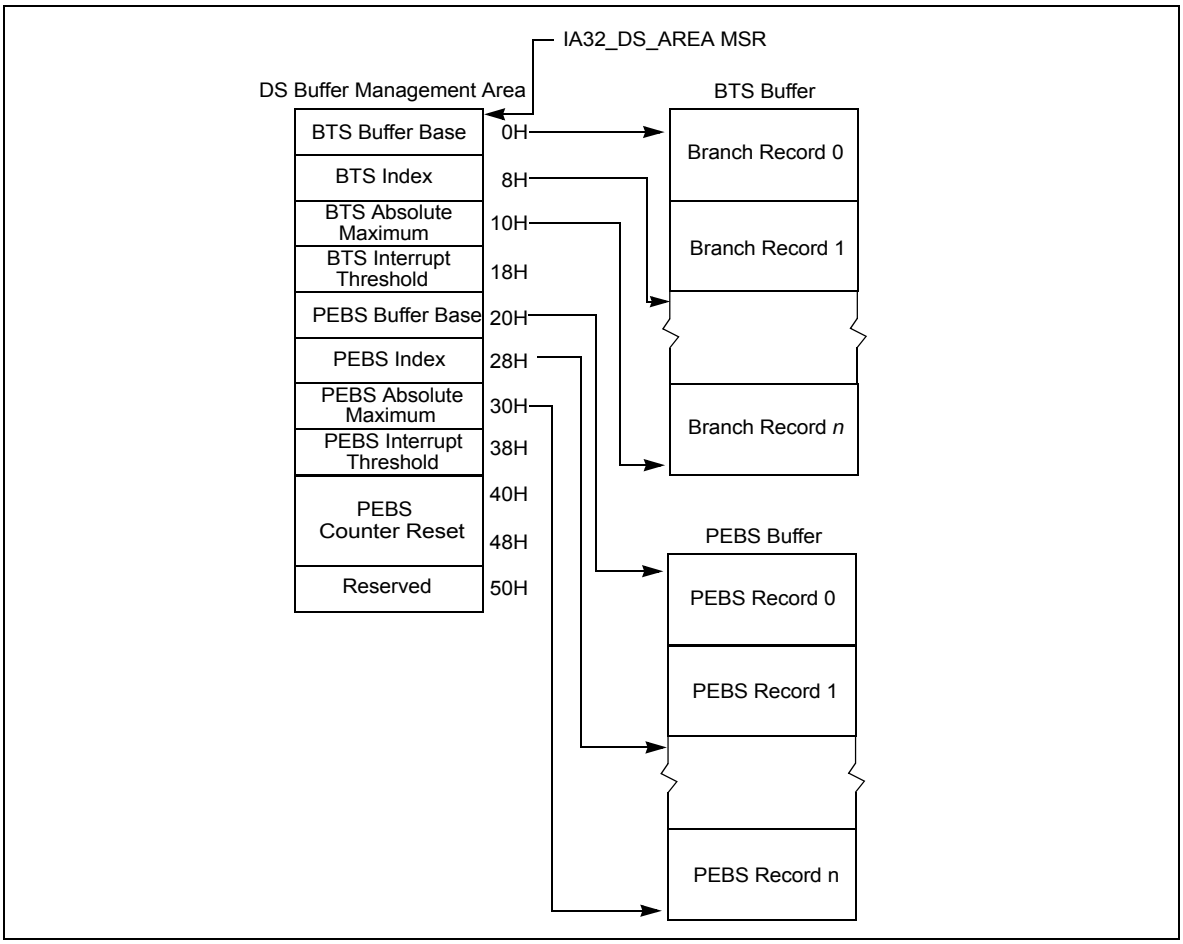

Figure 17-8 IA-32e Mode DS Save Area

The IA32_DS_AREA MSR holds the 64-bit linear address of the first byte of the DS buffer management area. The structure of a branch trace record is similar to that shown in Figure 17-6, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 17-9). The structure of a PEBS record is similar to that shown in Figure 17-7, but each field is 8 bytes in length and architectural states include register R8 through R15. This makes the size of a PEBS record in 64-bit mode 144 bytes (see Figure 17-10).

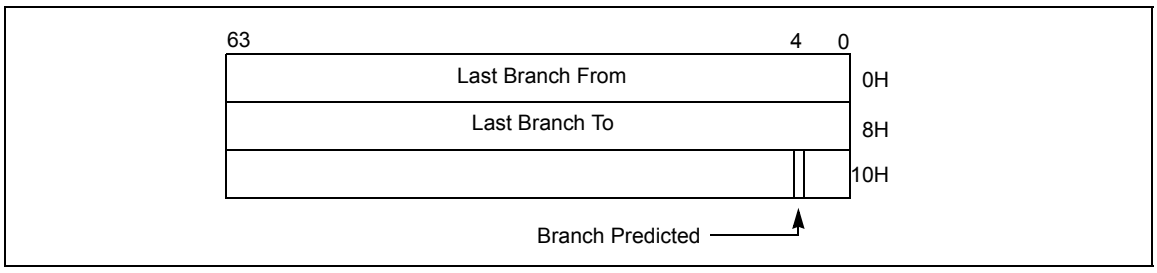

Figure 17-9 64-bit Branch Trace Record Format

...

17.13.4 Invariant Time-Keeping

The invariant TSC is based on the invariant timekeeping hardware (called Always Running Timer or ART), that runs at the core crystal clock frequency. The ratio defined by CPUID leaf 15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX[InvariantTSC] = 1, the following linearity relationship holds between TSC and the ART hardware:

TSC_Value = (ART_Value * CPUID.15H:EBX[31:0])/ CPUID.15H:EAX[31:0] + K

Where 'K' is an offset that can be adjusted by a privileged agent[1].

When ART hardware is reset, both invariant TSC and K are also reset.

...

18. 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.

...

19.5 PERFORMANCE MONITORING EVENTS FOR 2ND GENERATION INTEL® CORE™ I7-2XXX, INTEL® CORE™ I5-2XXX, INTEL® CORE™ I3-2XXX PROCESSOR SERIES

2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel Xeon processor E3-1200 product family are based on the Intel microarchitecture code name Sandy Bridge. They support architectural performance-monitoring events listed in Table 19-1. Non-architectural performance-monitoring events in the processor core are listed in Table 19-9, Table 19-10, and Table 19-11. The events in Table 19-9 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_2AH and 06_2DH. The events in Table 19-10 apply to processors with CPUID signature 06_2AH. The events in Table 19-11 apply to processors with CPUID signature 06_2DH.

Additional informations on event specifics (e.g. derivative events using specific IA32_PERFEVTSELx modifiers, limitations, special notes and recommendations) can be found at http://software.intel.com/en-us/forums/software-tuning-performance-optimization-platform-monitoring.

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	01H	LD_BLOCKS.DATA_UNKNOWN	blocked loads due to store buffer blocks with unknown data.	
03H	02H	LD_BLOCKS.STORE_FORWARD	loads blocked by overlapping with store buffer that cannot be forwarded.	

1. IA32_TSC_ADJUST MSR and the TSC-offset field in the VM execution controls of VMCS are some of the common interfaces that privileged software can use to manage the time stamp counter for keeping time

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	08H	LD_BLOCKS.NO_SR	# of Split loads blocked due to resource not available.	
03H	10H	LD_BLOCKS.ALL_BLOCK	Number of cases where any load is blocked but has no DCU miss.	
05H	01H	MISALIGN_MEM_REF.LOADS	Speculative cache-line split load uops dispatched to L1D.	
05H	02H	MISALIGN_MEM_REF.STORES	Speculative cache-line split Store-address uops dispatched to L1D.	
07H	01H	LD_BLOCKS_PARTIAL.ADDRESS_ALIAS	False dependencies in MOB due to partial compare on address.	
07H	08H	LD_BLOCKS_PARTIAL.ALL_STA_BLOCK	The number of times that load operations are temporarily blocked because of older stores, with addresses that are not yet known. A load operation may incur more than one block of this type.	
08H	01H	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Misses in all TLB levels that cause a page walk of any page size.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED	Misses in all TLB levels that caused page walk completed of any size.	
08H	04H	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
08H	10H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
0DH	03H	INT_MISC.RECOVERY_CYCLES	Cycles waiting to recover after Machine Clears or JEClear. Set Cmask= 1.	Set Edge to count occurrences
0DH	40H	INT_MISC.RAT_STALL_CYCLES	Cycles RAT external stall is sent to IDQ for this thread.	
0EH	01H	UOPS_ISSUED.ANY	Increments each cycle the # of Uops issued by the RAT to RS. Set Cmask = 1, Inv = 1, Any= 1to count stalled cycles of this core.	Set Cmask = 1, Inv = 1to count stalled cycles
10H	01H	FP_COMP_OPS_EXE.X87	Counts number of X87 uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE	Counts number of SSE* double precision FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE	Counts number of SSE* single precision FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_PACKED SINGLE	Counts number of SSE* single precision FP packed uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE	Counts number of SSE* double precision FP scalar uops executed.	
11H	01H	SIMD_FP_256.PACKED_SINGLE	Counts 256-bit packed single-precision floating-point instructions.	
11H	02H	SIMD_FP_256.PACKED_DOUBLE	Counts 256-bit packed double-precision floating-point instructions.	

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
14H	01H	ARITH.FPU_DIV_ACTIVE	Cycles that the divider is active, includes INT and FP. Set 'edge =1, cmask=1' to count the number of divides.	
17H	01H	INSTS_WRITTEN_TO_IQ.INSTS	Counts the number of instructions written into the IQ every cycle.	
24H	01H	L2_RQSTS.DEMAND_DATA_RD_HIT	Demand Data Read requests that hit L2 cache.	
24H	03H	L2_RQSTS.ALL_DEMAND_DATA_RD	Counts any demand and L1 HW prefetch data load requests to L2.	
24H	04H	L2_RQSTS.RFO_HITS	Counts the number of store RFO requests that hit the L2 cache.	
24H	08H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache.	
24H	0CH	L2_RQSTS.ALL_RFO	Counts all L2 store RFO requests.	
24H	10H	L2_RQSTS.CODE_RD_HIT	Number of instruction fetches that hit the L2 cache.	
24H	20H	L2_RQSTS.CODE_RD_MISS	Number of instruction fetches that missed the L2 cache.	
24H	30H	L2_RQSTS.ALL_CODE_RD	Counts all L2 code requests.	
24H	40H	L2_RQSTS.PF_HIT	Requests from L2 Hardware prefetcher that hit L2.	
24H	80H	L2_RQSTS.PF_MISS	Requests from L2 Hardware prefetcher that missed L2.	
24H	C0H	L2_RQSTS.ALL_PF	Any requests from L2 Hardware prefetchers.	
27H	01H	L2_STORE_LOCK_RQSTS.MISS	RFOs that miss cache lines.	
27H	04H	L2_STORE_LOCK_RQSTS.HIT_E	RFOs that hit cache lines in E state.	
27H	08H	L2_STORE_LOCK_RQSTS.HIT_M	RFOs that hit cache lines in M state.	
27H	0FH	L2_STORE_LOCK_RQSTS.ALL	RFOs that access cache lines in any state.	
28H	01H	L2_L1D_WB_RQSTS.MISS	Not rejected writebacks from L1D to L2 cache lines that missed L2.	
28H	02H	L2_L1D_WB_RQSTS.HIT_S	Not rejected writebacks from L1D to L2 cache lines in S state.	
28H	04H	L2_L1D_WB_RQSTS.HIT_E	Not rejected writebacks from L1D to L2 cache lines in E state.	
28H	08H	L2_L1D_WB_RQSTS.HIT_M	Not rejected writebacks from L1D to L2 cache lines in M state.	
28H	0FH	L2_L1D_WB_RQSTS.ALL	Not rejected writebacks from L1D to L2 cache.	
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache.	see Table 19-1

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache.	see Table 19-1
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.	see Table 19-1
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Increments at the frequency of XCLK (100 MHz) when not halted.	see Table 19-1
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmask = 1 and Edge =1 to count occurrences.	PMC2 only; Set Cmask = 1 to count cycles.
49H	01H	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Miss in all TLB levels causes an page walk of any page size (4K/2M/4M/1G).	
49H	02H	DTLB_STORE_MISSES.WALK_COMPLETED	Miss in all TLB levels causes a page walk that completes of any page size (4K/2M/4M/1G).	
49H	04H	DTLB_STORE_MISSES.WALK_DURATION	Cycles PMH is busy with this walk.	
49H	10H	DTLB_STORE_MISSES.STLB_HIT	Store operations that miss the first TLB level but hit the second and do not cause page walks.	
4CH	01H	LOAD_HIT_PRE.SW_PF	Not SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.	
4CH	02H	LOAD_HIT_PRE.HW_PF	Not SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
4EH	02H	HW_PRE_REQ.DL1_MISS	Hardware Prefetch requests that miss the L1D cache. A request is being counted each time it access the cache & miss it, including if a block is applicable or if hit the Fill Buffer for example.	This accounts for both L1 streamer and IP-based (IPP) HW prefetchers.
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
51H	02H	L1D.ALLOCATED_IN_M	Counts the number of allocations of modified L1D cache lines.	
51H	04H	L1D.EVICTION	Counts the number of modified lines evicted from the L1 data cache due to replacement.	
51H	08H	L1D.ALL_M_REPLACEMENT	Cache lines in M state evicted out of L1D due to Snoop HitM or dirty line replacement.	
59H	20H	PARTIAL_RAT_STALLS.FLAGS_MERGE_UOP	Increments the number of flags-merge uops in flight each cycle. Set Cmask = 1 to count cycles.	
59H	40H	PARTIAL_RAT_STALLS.SLOW_LEA_WINDOW	Cycles with at least one slow LEA uop allocated.	
59H	80H	PARTIAL_RAT_STALLS.MUL_SINGLE_UOP	Number of Multiply packed/scalar single precision uops allocated.	

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
5BH	0CH	RESOURCE_STALLS2.ALL_FL_EMPTY	Cycles stalled due to free list empty.	PMC0-3 only regardless HTT
5BH	0FH	RESOURCE_STALLS2.ALL_PRF_CONTROL	Cycles stalled due to control structures full for physical registers.	
5BH	40H	RESOURCE_STALLS2.BOB_FULL	Cycles Allocator is stalled due Branch Order Buffer.	
5BH	4FH	RESOURCE_STALLS2.OOO_RSRC	Cycles stalled due to out of order resources full.	
5CH	01H	CPL_CYCLES.RING0	Unhalted core cycles when the thread is in ring 0.	Use Edge to count transition
5CH	02H	CPL_CYCLES.RING123	Unhalted core cycles when the thread is not in ring 0.	
5EH	01H	RS_EVENTS.EMPTY_CYCLES	Cycles the RS is empty for the thread.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Offcore outstanding Demand Data Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Offcore outstanding cacheable data read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
63H	01H	LOCK_CYCLES.SPLIT_LOCK_UC_LOCK_DURATION	Cycles in which the L1D and L2 are locked, due to a UC lock or split lock.	
63H	02H	LOCK_CYCLES.CACHE_LOCK_DURATION	Cycles in which the L1D is locked.	
79H	02H	IDQ.EMPTY	Counts cycles the IDQ is empty.	
79H	04H	IDQ.MITE_UOPS	Increment each cycle # of uops delivered to IDQ from MITE path. Set Cmask = 1 to count cycles.	Can combine Umask 04H and 20H
79H	08H	IDQ.DSB_UOPS	Increment each cycle. # of uops delivered to IDQ from DSB path. Set Cmask = 1 to count cycles.	Can combine Umask 08H and 10H
79H	10H	IDQ.MS_DSB_UOPS	Increment each cycle # of uops delivered to IDQ when MS busy by DSB. Set Cmask = 1 to count cycles MS is busy. Set Cmask=1 and Edge =1 to count MS activations.	Can combine Umask 08H and 10H
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS is busy by MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H and 20H
79H	30H	IDQ.MS_UOPS	Increment each cycle # of uops delivered to IDQ from MS by either DSB or MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H and 30H

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
80H	02H	ICACHE.MISSES	Number of Instruction Cache, Streaming Buffer and Victim Cache Misses. Includes UC accesses.	
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses in all ITLB levels that cause page walks.	
85H	02H	ITLB_MISSES.WALK_COMPLETED	Misses in all ITLB levels that cause completed page walks.	
85H	04H	ITLB_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
85H	10H	ITLB_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to IQ is full.	
88H	41H	BR_INST_EXEC.NONTAKEN_CONDITIONAL	Not-taken macro conditional branches	
88H	81H	BR_INST_EXEC.TAKEN_CONDITIONAL	Taken speculative and retired conditional branches	
88H	82H	BR_INST_EXEC.TAKEN_DIRECT_JUMP	Taken speculative and retired conditional branches excluding calls and indirects	
88H	84H	BR_INST_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET	Taken speculative and retired indirect branches excluding calls and returns	
88H	88H	BR_INST_EXEC.TAKEN_INDIRECT_NEAR_RETURN	Taken speculative and retired indirect branches that are returns	
88H	90H	BR_INST_EXEC.TAKEN_DIRECT_NEAR_CALL	Taken speculative and retired direct near calls	
88H	A0H	BR_INST_EXEC.TAKEN_INDIRECT_NEAR_CALL	Taken speculative and retired indirect near calls	
88H	C1H	BR_INST_EXEC.ALL_CONDITIONAL	Speculative and retired conditional branches	
88H	C2H	BR_INST_EXEC.ALL_DIRECT_JUMP	Speculative and retired conditional branches excluding calls and indirects	
88H	C4H	BR_INST_EXEC.ALL_INDIRECT_JUMP_NON_CALL_RET	Speculative and retired indirect branches excluding calls and returns	
88H	C8H	BR_INST_EXEC.ALL_INDIRECT_NEAR_RETURN	Speculative and retired indirect branches that are returns	
88H	D0H	BR_INST_EXEC.ALL_NEAR_CALL	Speculative and retired direct near calls	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Speculative and retired branches	
89H	41H	BR_MISP_EXEC.NONTAKEN_CONDITIONAL	Not-taken mispredicted macro conditional branches	
89H	81H	BR_MISP_EXEC.TAKEN_CONDITIONAL	Taken speculative and retired mispredicted conditional branches	

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
89H	84H	BR_MISP_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET	Taken speculative and retired mispredicted indirect branches excluding calls and returns	
89H	88H	BR_MISP_EXEC.TAKEN_RETURN_NEAR	Taken speculative and retired mispredicted indirect branches that are returns	
89H	90H	BR_MISP_EXEC.TAKEN_DIRECT_NEAR_CALL	Taken speculative and retired mispredicted direct near calls	
89H	A0H	BR_MISP_EXEC.TAKEN_INDIRECT_NEAR_CALL	Taken speculative and retired mispredicted indirect near calls	
89H	C1H	BR_MISP_EXEC.ALL_CONDITIONAL	Speculative and retired mispredicted conditional branches	
89H	C4H	BR_MISP_EXEC.ALL_INDIRECT_JUMP_NON_CALL_RET	Speculative and retired mispredicted indirect branches excluding calls and returns	
89H	D0H	BR_MISP_EXEC.ALL_NEAR_CALL	Speculative and retired mispredicted direct near calls	
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Speculative and retired mispredicted branches	
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Count number of non-delivered uops to RAT per thread.	Use Cmask to qualify uop b/w
A1H	01H	UOPS_DISPATCHED_PORT.PORT_0	Cycles which a Uop is dispatched on port 0.	
A1H	02H	UOPS_DISPATCHED_PORT.PORT_1	Cycles which a Uop is dispatched on port 1.	
A1H	0CH	UOPS_DISPATCHED_PORT.PORT_2	Cycles which a Uop is dispatched on port 2.	
A1H	30H	UOPS_DISPATCHED_PORT.PORT_3	Cycles which a Uop is dispatched on port 3.	
A1H	40H	UOPS_DISPATCHED_PORT.PORT_4	Cycles which a Uop is dispatched on port 4.	
A1H	80H	UOPS_DISPATCHED_PORT.PORT_5	Cycles which a Uop is dispatched on port 5.	
A2H	01H	RESOURCE_STALLS.ANY	Cycles Allocation is stalled due to Resource Related reason.	
A2H	02H	RESOURCE_STALLS.LB	Counts the cycles of stall due to lack of load buffers.	
A2H	04H	RESOURCE_STALLS.RS	Cycles stalled due to no eligible RS entry available.	
A2H	08H	RESOURCE_STALLS.SB	Cycles stalled due to no store buffers available. (not including draining form sync).	
A2H	10H	RESOURCE_STALLS.ROB	Cycles stalled due to re-order buffer full.	
A2H	20H	RESOURCE_STALLS.FCSW	Cycles stalled due to writing the FPU control word.	
A3H	02H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 cache miss loads.Set AnyThread to count per core.	PMC2 only
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_PENDING	Cycles with pending L2 miss loads. Set AnyThread to count per core.	

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A3H	04H	CYCLE_ACTIVITY.CYCLES_NO_DISPATCH	Cycles of dispatch stalls. Set AnyThread to count per core.	PMC0-3 only
A8H	01H	LSD.UOPS	Number of Uops delivered by the LSD.	
ABH	01H	DSB2MITE_SWITCHES.COUNT	Number of DSB to MITE switches.	
ABH	02H	DSB2MITE_SWITCHES.PENALTY_CYCLES	Cycles DSB to MITE switches caused delay.	
ACH	02H	DSB_FILL.OTHER_CANCEL	Cases of cancelling valid DSB fill not because of exceeding way limit.	
ACH	08H	DSB_FILL.EXCEED_DSB_LINES	DSB Fill encountered > 3 DSB lines.	
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes, includes 4k/2M/4M pages.	
B0H	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data 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).	
B1H	01H	UOPS_DISPATCHED.THREAD	Counts total number of uops to be dispatched per-thread each cycle. Set Cmask = 1, INV =1 to count stall cycles.	PMC0-3 only regardless HTT
B1H	02H	UOPS_DISPATCHED.CORE	Counts total number of uops to be dispatched per-core each cycle.	Do not need to set ANY
B2H	01H	OFFCORE_REQUESTS_BUFFER.SQ_FULL	Offcore requests buffer cannot take more entries for this thread core.	
B6H	01H	AGU_BYPASS_CANCEL.COUNT	Counts executed load operations with all the following traits: 1. addressing of the format [base + offset], 2. the offset is between 1 and 2047, 3. the address specified in the base register is in one page and the address [base+offset] is in another page.	
B7H	01H	OFF_CORE_RESPONSE_0	see Section 18.9.5, “Off-core Response Performance Monitoring”.	Requires MSR 01A6H
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.9.5, “Off-core Response Performance Monitoring”.	Requires MSR 01A7H
BDH	01H	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries.	
BDH	20H	TLB_FLUSH.STLB_ANY	Count number of STLB flush attempts.	
BFH	05H	L1D_BLOCKS.BANK_CONFLICT_CYCLES	Cycles when dispatched loads are cancelled due to L1D bank conflicts with other load ports.	cmask=1
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; Must quiesce other PMCs.
C1H	02H	OTHER_ASSISTS.ITLB_MISS_RETIRED	Instructions that experienced an ITLB miss.	

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C1H	08H	OTHER_ASSISTS.AVX_STORE	Number of assists associated with 256-bit AVX store operations.	
C1H	10H	OTHER_ASSISTS.AVX_TO_SSE	Number of transitions from AVX-256 to legacy SSE when penalty applicable.	
C1H	20H	OTHER_ASSISTS.SSE_TO_AVX	Number of transitions from SSE to AVX-256 when penalty applicable.	
C2H	01H	UOPS_RETIRED.ALL	Counts the number of micro-ops retired, Use cmask=1 and invert to count active cycles or stalled cycles.	Supports PEBS
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	Supports PEBS
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEARS.SMC	Counts the number of times that a program writes to a code section.	
C3H	20H	MACHINE_CLEARS.MASKMOV	Counts the number of executed AVX masked load operations that refer to an illegal address range with the mask bits set to 0.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement.	See Table 19-1
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	Supports PEBS
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.	Supports PEBS
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	Supports PEBS
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	Supports PEBS
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.	Supports PEBS
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.	
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.	Supports PEBS
C5H	02H	BR_MISP_RETIRED.NEAR_CALL	Direct and indirect mispredicted near call instructions retired.	Supports PEBS
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	Supports PEBS

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C5H	10H	BR_MISP_RETIRED.NOT_TAKEN	Mispredicted not taken branch instructions retired.	Supports PEBS
C5H	20H	BR_MISP_RETIRED.TAKEN	Mispredicted taken branch instructions retired.	Supports PEBS
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 assists due to output value.	
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 assists due to input value.	
CAH	08H	FP_ASSIST.SIMD_OUTPUT	Number of SIMD FP assists due to output values.	
CAH	10H	FP_ASSIST.SIMD_INPUT	Number of SIMD FP assists due to input values.	
CAH	1EH	FP_ASSIST.ANY	Cycles with any input/output SSE* or FP assists.	
CCH	20H	ROB_MISC_EVENTS.LBR_INSERTS	Count cases of saving new LBR records by hardware.	
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. PMC3 only.	Specify threshold in MSR 3F6H
CDH	02H	MEM_TRANS_RETIRED.PRECISE_STORE	Sample stores and collect precise store operation via PEBS record. PMC3 only.	See Section 18.9.4.3
D0H	11H	MEM_UOP_RETIRED.STLB_MISS_LOADS	Load uops with true STLB miss retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	12H	MEM_UOP_RETIRED.STLB_MISS_STORES	Store uops with true STLB miss retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	21H	MEM_UOP_RETIRED.LOCK_LOADS	Load uops with lock access retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	22H	MEM_UOP_RETIRED.LOCK_STORES	Store uops with lock access retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	41H	MEM_UOP_RETIRED.SPLIT_LOADS	Load uops with cacheline split retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	42H	MEM_UOP_RETIRED.SPLIT_STORES	Store uops with cacheline split retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	81H	MEM_UOP_RETIRED.ALL_LOADS	ALL Load uops retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	82H	MEM_UOP_RETIRED.ALL_STORES	ALL Store uops retired to architectural path.	Supports PEBS. PMC0-3 only regardless HTT.
D0H	80H	MEM_UOP_RETIRED.ALL	Qualify any retired memory uops. Must combine with umask 01H, 02H, to produce counts.	
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS. PMC0-3 only regardless HTT
D1H	02H	MEM_LOAD_UOPS_RETIRED.L2_HIT	Retired load uops with L2 cache hits as data sources.	Supports PEBS
D1H	04H	MEM_LOAD_UOPS_RETIRED.LLC_HIT	Retired load uops which data sources were data hits in LLC without snoops required.	Supports PEBS
D1H	20H	MEM_LOAD_UOPS_RETIRED.LLC_MISS	Retired load uops which data sources were data missed LLC (excluding unknown data source).	Supports PEBS

Table 19-9 Non-Architectural Performance Events In the Processor Core Common to 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processors E3 and E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	40H	MEM_LOAD_UOPS_RETIRED.HIT_LFB	Retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.	Supports PEBS
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Retired load uops whose data source was an on-package core cache LLC hit and cross-core snoop missed.	Supports PEBS
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Retired load uops whose data source was an on-package LLC hit and cross-core snoop hits.	Supports PEBS
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops whose data source was an on-package core cache with HitM responses.	Supports PEBS
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Retired load uops whose data source was LLC hit with no snoop required.	Supports PEBS
E6H	01H	BACLEARS.ANY	Counts the number of times the front end is re-steered, mainly when the BPU cannot provide a correct prediction and this is corrected by other branch handling mechanisms at the front end.	
F0H	01H	L2_TRANS.DEMAND_DATA_RD	Demand Data Read requests that access L2 cache.	
F0H	02H	L2_TRANS.RFO	RFO requests that access L2 cache.	
F0H	04H	L2_TRANS.CODE_RD	L2 cache accesses when fetching instructions.	
F0H	08H	L2_TRANS.ALL_PF	L2 or LLC HW prefetches that access L2 cache.	Including rejects
F0H	10H	L2_TRANS.L1D_WB	L1D writebacks that access L2 cache.	
F0H	20H	L2_TRANS.L2_FILL	L2 fill requests that access L2 cache.	
F0H	40H	L2_TRANS.L2_WB	L2 writebacks that access L2 cache.	
F0H	80H	L2_TRANS.ALL_REQUESTS	Transactions accessing L2 pipe.	
F1H	01H	L2_LINES_IN.I	L2 cache lines in I state filling L2.	Counting does not cover rejects.
F1H	02H	L2_LINES_IN.S	L2 cache lines in S state filling L2.	Counting does not cover rejects.
F1H	04H	L2_LINES_IN.E	L2 cache lines in E state filling L2.	Counting does not cover rejects.
F1H	07H	L2_LINES_IN.ALL	L2 cache lines filling L2.	Counting does not cover rejects.
F2H	01H	L2_LINES_OUT.DEMAND_CLEAN	Clean L2 cache lines evicted by demand.	
F2H	02H	L2_LINES_OUT.DEMAND_DIRTY	Dirty L2 cache lines evicted by demand.	
F2H	04H	L2_LINES_OUT.PF_CLEAN	Clean L2 cache lines evicted by L2 prefetch.	
F2H	08H	L2_LINES_OUT.PF_DIRTY	Dirty L2 cache lines evicted by L2 prefetch.	
F2H	0AH	L2_LINES_OUT.DIRTY_ALL	Dirty L2 cache lines filling the L2.	Counting does not cover rejects.
F4H	10H	SQ_MISC.SPLIT_LOCK	Split locks in SQ.	

Non-architecture performance monitoring events in the processor core that are applicable only to Intel processors with CPUID signature of DisplayFamily_DisplayModel 06_2AH are listed in Table 19-10.

Table 19-10 Non-Architectural Performance Events applicable only to the Processor core for 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Retired load uops which data sources were LLC hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS. PMC0-3 only regardless HTT
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Retired load uops which data sources were LLC and cross-core snoop hits in on-pkg core cache.	Supports PEBS.
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops which data sources were HitM responses from shared LLC.	Supports PEBS.
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Retired load uops which data sources were hits in LLC without snoops required.	Supports PEBS.
D4H	02H	MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS	Retired load uops with unknown information as data source in cache serviced the load.	Supports PEBS. PMC0-3 only regardless HTT
B7H/BBH	01H	OFF_CORE_RESPONSE_N	Sub-events of OFF_CORE_RESPONSE_N (suffix N = 0, 1) programmed using MSR 01A6H/01A7H with values shown in the comment column.	
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT_N		10003C0244H
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0244H
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0244H
		OFFCORE_RESPONSE.ALL_CODE_RD.LLC_HIT.MISS_DRAM_N		300400244H
		OFFCORE_RESPONSE.ALL_DATA_RD.LLC_HIT.ANY_RESPONSE_N		3F803C0091H
		OFFCORE_RESPONSE.ALL_DATA_RD.LLC_MISS.DRAM_N		300400091H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.ANY_RESPONSE_N		3F803C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0240H
		OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_MISS.DRAM_N		300400240H
		OFFCORE_RESPONSE.ALL_PF_DATA_RD.LLC_MISS.DRAM_N		300400090H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0120H
		OFFCORE_RESPONSE.ALL_PF_RfO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_HIT.SNOOP_MISS_N		2003C0120H
		OFFCORE_RESPONSE.ALL_PF_RFO.LLC_MISS.DRAM_N		300400120H
		OFFCORE_RESPONSE.ALL_READS.LLC_MISS.DRAM_N		3004003F7H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0122H

Table 19-10 Non-Architectural Performance Events applicable only to the Processor core for 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_HIT.SNOOP_MISS_N		2003C0122H
		OFFCORE_RESPONSE.ALL_RFO.LLC_MISS.DRAM_N		300400122H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0004H
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.DRAM_N		300400004H
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.DRAM_N		300400001H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_HIT.SNOOP_MISS_N		2003C0002H
		OFFCORE_RESPONSE.DEMAND_RFO.LLC_MISS.DRAM_N		300400002H
		OFFCORE_RESPONSE.OTHER.ANY_RESPONSE_N		18000H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0040H
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_MISS.DRAM_N		300400040H
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.DRAM_N		300400010H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_HIT.SNOOP_MISS_N		2003C0020H
		OFFCORE_RESPONSE.PF_L2_RFO.LLC_MISS.DRAM_N		300400020H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.HITM_OTHER_CORE_N		10003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_HIT.SNOOP_MISS_N		2003C0200H
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_MISS.DRAM_N		300400200H
		OFFCORE_RESPONSE.PF_LLC_DATA_RD.LLC_MISS.DRAM_N		300400080H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.ANY_RESPONSE_N		3F803C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.HIT_OTHER_CORE_NO_FWD_N		4003C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.HITM_OTHER_CORE_N		10003C0100H

Table 19-10 Non-Architectural Performance Events applicable only to the Processor core for 2nd Generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.NO_SNOOP_NEEDED_N		1003C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_HIT.SNOOP_MISS_N		2003C0100H
		OFFCORE_RESPONSE.PF_LLC_RFO.LLC_MISS.DRAM_N		300400100H

…

19.6 PERFORMANCE MONITORING EVENTS FOR INTEL® CORE™ I7 PROCESSOR FAMILY AND INTEL® XEON® PROCESSOR FAMILY

Processors based on the Intel microarchitecture code name Nehalem support the architectural and non-architectural performance-monitoring events listed in Table 19-1 and Table 19-13. The events in Table 19-13 generally applies to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_1AH, 06_1EH, 06_1FH, and 06_2EH. However, Intel Xeon processors with CPUID signature of DisplayFamily_DisplayModel 06_2EH have a small number of events that are not supported in processors with CPUID signature 06_1AH, 06_1EH, and 06_1FH. These events are noted in the comment column.

In addition, these processors (CPUID signature of DisplayFamily_DisplayModel 06_1AH, 06_1EH, 06_1FH) also support the following non-architectural, product-specific uncore performance-monitoring events listed in Table 19-14.

Fixed counters in the core PMU support the architecture events defined in Table 19-18.

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
04H	07H	SB_DRAIN.ANY	Counts the number of store buffer drains.	
06H	04H	STORE_BLOCKS.AT_RET	Counts number of loads delayed with at-Retirement block code. The following loads need to be executed at retirement and wait for all senior stores on the same thread to be drained: load splitting across 4K boundary (page split), load accessing uncacheable (UC or WC) memory, load lock, and load with page table in UC or WC memory region.	
06H	08H	STORE_BLOCKS.L1D_BLOCK	Cacheable loads delayed with L1D block code.	
07H	01H	PARTIAL_ADDRESS_ALIAS	Counts false dependency due to partial address aliasing.	
08H	01H	DTLB_LOAD_MISSES.ANY	Counts all load misses that cause a page walk.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED	Counts number of completed page walks due to load miss in the STLB.	
08H	10H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits.	
08H	20H	DTLB_LOAD_MISSES.PDE_MISS	Number of DTLB cache load misses where the low part of the linear to physical address translation was missed.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
08H	80H	DTLB_LOAD_MISSES.LARGE_WALK_COMPLETED	Counts number of completed large page walks due to load miss in the STLB.	
0BH	01H	MEM_INST_RETIRED.LOADS	Counts the number of instructions with an architecturally-visible load retired on the architected path.	
0BH	02H	MEM_INST_RETIRED.STORES	Counts the number of instructions with an architecturally-visible store retired on the architected path.	
0BH	10H	MEM_INST_RETIRED.LATENCY_ABOVE_THRESHOLD	Counts the number of instructions exceeding the latency specified with ld_lat facility.	In conjunction with ld_lat facility
0CH	01H	MEM_STORE_RETIRED.DTLB_MISS	The event counts the number of retired stores that missed the DTLB. The DTLB miss is not counted if the store operation causes a fault. Does not counter prefetches. Counts both primary and secondary misses to the TLB.	
0EH	01H	UOPS_ISSUED.ANY	Counts the number of Uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	
0EH	01H	UOPS_ISSUED.STALLED_CYCLES	Counts the number of cycles no Uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	set "invert=1, cmask = 1"
0EH	02H	UOPS_ISSUED.FUSED	Counts the number of fused Uops that were issued from the Register Allocation Table to the Reservation Station.	
0FH	01H	MEM_UNCORE_RETIRED.L3_DATA_MISS_UNKNOWN	Counts number of memory load instructions retired where the memory reference missed L3 and data source is unknown.	Available only for CPUID signature 06_2EH
0FH	02H	MEM_UNCORE_RETIRED.OTHER_CORE_L2_HITM	Counts number of memory load instructions retired where the memory reference hit modified data in a sibling core residing on the same socket.	
0FH	08H	MEM_UNCORE_RETIRED.REMOTE_CACHE_LOCAL_HOME_HIT	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and HIT in a remote socket's cache. Only counts locally homed lines.	
0FH	10H	MEM_UNCORE_RETIRED.REMOTE_DRAM	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and was remotely homed. This includes both DRAM access and HITM in a remote socket's cache for remotely homed lines.	
0FH	20H	MEM_UNCORE_RETIRED.LOCAL_DRAM	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and required a local socket memory reference. This includes locally homed cachelines that were in a modified state in another socket.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0FH	80H	MEM_UNCORE_RETIRED.UNCACHEABLE	Counts number of memory load instructions retired where the memory reference missed the L1, L2 and L3 caches and to perform I/O.	Available only for CPUID signature 06_2EH
10H	01H	FP_COMP_OPS_EXE.X87	Counts the number of FP Computational Uops Executed. The number of FADD, FSUB, FCOM, FMULs, integer MULsand IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs, and IDIVs. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	
10H	02H	FP_COMP_OPS_EXE.MMX	Counts number of MMX Uops executed.	
10H	04H	FP_COMP_OPS_EXE.SSE_FP	Counts number of SSE and SSE2 FP uops executed.	
10H	08H	FP_COMP_OPS_EXE.SSE2_INTEGER	Counts number of SSE2 integer uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED	Counts number of SSE FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR	Counts number of SSE FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_SINGLE_PRECISION	Counts number of SSE* FP single precision uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_DOUBLE_PRECISION	Counts number of SSE* FP double precision uops executed.	
12H	01H	SIMD_INT_128.PACKED_MPY	Counts number of 128 bit SIMD integer multiply operations.	
12H	02H	SIMD_INT_128.PACKED_SHIFT	Counts number of 128 bit SIMD integer shift operations.	
12H	04H	SIMD_INT_128.PACK	Counts number of 128 bit SIMD integer pack operations.	
12H	08H	SIMD_INT_128.UNPACK	Counts number of 128 bit SIMD integer unpack operations.	
12H	10H	SIMD_INT_128.PACKED_LOGICAL	Counts number of 128 bit SIMD integer logical operations.	
12H	20H	SIMD_INT_128.PACKED_ARITH	Counts number of 128 bit SIMD integer arithmetic operations.	
12H	40H	SIMD_INT_128.SHUFFLE_MOVE	Counts number of 128 bit SIMD integer shuffle and move operations.	
13H	01H	LOAD_DISPATCH.RS	Counts number of loads dispatched from the Reservation Station that bypass the Memory Order Buffer.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
13H	02H	LOAD_DISPATCH.RS_DELAYED	Counts the number of delayed RS dispatches at the stage latch. If an RS dispatch can not bypass to LB, it has another chance to dispatch from the one-cycle delayed staging latch before it is written into the LB.	
13H	04H	LOAD_DISPATCH.MOB	Counts the number of loads dispatched from the Reservation Station to the Memory Order Buffer.	
13H	07H	LOAD_DISPATCH.ANY	Counts all loads dispatched from the Reservation Station.	
14H	01H	ARITH.CYCLES_DIV_BUSY	Counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE. Set 'edge =1, invert=1, cmask=1' to count the number of divides.	Count may be incorrect When SMT is on.
14H	02H	ARITH.MUL	Counts the number of multiply operations executed. This includes integer as well as floating point multiply operations but excludes DPPS mul and MPSAD.	Count may be incorrect When SMT is on
17H	01H	INST_QUEUE_WRITES	Counts the number of instructions written into the instruction queue every cycle.	
18H	01H	INST_DECODED.DEC0	Counts number of instructions that require decoder 0 to be decoded. Usually, this means that the instruction maps to more than 1 uop.	
19H	01H	TWO_UOP_INSTS_DECODED	An instruction that generates two uops was decoded.	
1EH	01H	INST_QUEUE_WRITE_CYCLES	This event counts the number of cycles during which instructions are written to the instruction queue. Dividing this counter by the number of instructions written to the instruction queue (INST_QUEUE_WRITES) yields the average number of instructions decoded each cycle. If this number is less than four and the pipe stalls, this indicates that the decoder is failing to decode enough instructions per cycle to sustain the 4-wide pipeline.	If SSE* instructions that are 6 bytes or longer arrive one after another, then front end throughput may limit execution speed.
20H	01H	LSD_OVERFLOW	Counts number of loops that can't stream from the instruction queue.	
24H	01H	L2_RQSTS.LD_HIT	Counts number of loads that hit the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches. L2 loads can be rejected for various reasons. Only non rejected loads are counted.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	02H	L2_RQSTS.LD_MISS	Counts the number of loads that miss the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches.	
24H	03H	L2_RQSTS.LOADS	Counts all L2 load requests. L2 loads include both L1D demand misses as well as L1D prefetches.	
24H	04H	L2_RQSTS.RFO_HIT	Counts the number of store RFO requests that hit the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches. Count includes WC memory requests, where the data is not fetched but the permission to write the line is required.	
24H	08H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	
24H	0CH	L2_RQSTS.RFOS	Counts all L2 store RFO requests. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	
24H	10H	L2_RQSTS.IFETCH_HIT	Counts number of instruction fetches that hit the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	20H	L2_RQSTS.IFETCH_MISS	Counts number of instruction fetches that miss the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	30H	L2_RQSTS.IFETCHES	Counts all instruction fetches. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	40H	L2_RQSTS.PREFETCH_HIT	Counts L2 prefetch hits for both code and data.	
24H	80H	L2_RQSTS.PREFETCH_MISS	Counts L2 prefetch misses for both code and data.	
24H	C0H	L2_RQSTS.PREFETCHES	Counts all L2 prefetches for both code and data.	
24H	AAH	L2_RQSTS.MISS	Counts all L2 misses for both code and data.	
24H	FFH	L2_RQSTS.REFERENCES	Counts all L2 requests for both code and data.	
26H	01H	L2_DATA_RQSTS.DEMAND.I_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the I (invalid) state, i.e. a cache miss. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	02H	L2_DATA_RQSTS.DEMAND.S_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the S (shared) state. L2 demand loads are both L1D demand misses and L1D prefetches.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
26H	04H	L2_DATA_RQSTS.DEMAND.E_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the E (exclusive) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	08H	L2_DATA_RQSTS.DEMAND.M_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the M (modified) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	0FH	L2_DATA_RQSTS.DEMAND.MESI	Counts all L2 data demand requests. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	10H	L2_DATA_RQSTS.PREFETCH.I_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the I (invalid) state, i.e. a cache miss.	
26H	20H	L2_DATA_RQSTS.PREFETCH.S_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the S (shared) state. A prefetch RFO will miss on an S state line, while a prefetch read will hit on an S state line.	
26H	40H	L2_DATA_RQSTS.PREFETCH.E_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the E (exclusive) state.	
26H	80H	L2_DATA_RQSTS.PREFETCH.M_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the M (modified) state.	
26H	F0H	L2_DATA_RQSTS.PREFETCH.MESI	Counts all L2 prefetch requests.	
26H	FFH	L2_DATA_RQSTS.ANY	Counts all L2 data requests.	
27H	01H	L2_WRITE.RFO.I_STATE	Counts number of L2 demand store RFO requests where the cache line to be loaded is in the I (invalid) state, i.e, a cache miss. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	02H	L2_WRITE.RFO.S_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the S (shared) state. The L1D prefetcher does not issue a RFO prefetch,.	This is a demand RFO request
27H	08H	L2_WRITE.RFO.M_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the M (modified) state. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	0EH	L2_WRITE.RFO.HIT	Counts number of L2 store RFO requests where the cache line to be loaded is in either the S, E or M states. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	0FH	L2_WRITE.RFO.MESI	Counts all L2 store RFO requests.The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	10H	L2_WRITE.LOCK.I_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the I (invalid) state, i.e. a cache miss.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
27H	20H	L2_WRITE.LOCK.S_STATE	Counts number of L2 lock RFO requests where the cache line to be loaded is in the S (shared) state.	
27H	40H	L2_WRITE.LOCK.E_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the E (exclusive) state.	
27H	80H	L2_WRITE.LOCK.M_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the M (modified) state.	
27H	E0H	L2_WRITE.LOCK.HIT	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in either the S, E, or M state.	
27H	F0H	L2_WRITE.LOCK.MESI	Counts all L2 demand lock RFO requests.	
28H	01H	L1D_WB_L2.I_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the I (invalid) state, i.e. a cache miss.	
28H	02H	L1D_WB_L2.S_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the S state.	
28H	04H	L1D_WB_L2.E_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the E (exclusive) state.	
28H	08H	L1D_WB_L2.M_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the M (modified) state.	
28H	0FH	L1D_WB_L2.MESI	Counts all L1 writebacks to the L2 .	
2EH	4FH	L3_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache. The event count includes speculative traffic but excludes cache line fills due to a L2 hardware-prefetch. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	see Table 19-1
2EH	41H	L3_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache. The event count may include speculative traffic but excludes cache line fills due to L2 hardware-prefetches. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	see Table 19-1
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.	see Table 19-1

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
3CH	01H	CPU_CLK_UNHALTED.REF_P	Increments at the frequency of TSC when not halted.	see Table 19-1
40H	01H	L1D_CACHE_LD.I_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the I (invalid) state, i.e. the read request missed the cache.	Counter 0, 1 only
40H	02H	L1D_CACHE_LD.S_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the S (shared) state.	Counter 0, 1 only
40H	04H	L1D_CACHE_LD.E_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the E (exclusive) state.	Counter 0, 1 only
40H	08H	L1D_CACHE_LD.M_STATE	Counts L1 data cache read requests where the cache line to be loaded is in the M (modified) state.	Counter 0, 1 only
40H	0FH	L1D_CACHE_LD.MESI	Counts L1 data cache read requests.	Counter 0, 1 only
41H	02H	L1D_CACHE_ST.S_STATE	Counts L1 data cache store RFO requests where the cache line to be loaded is in the S (shared) state.	Counter 0, 1 only
41H	04H	L1D_CACHE_ST.E_STATE	Counts L1 data cache store RFO requests where the cache line to be loaded is in the E (exclusive) state.	Counter 0, 1 only
41H	08H	L1D_CACHE_ST.M_STATE	Counts L1 data cache store RFO requests where cache line to be loaded is in the M (modified) state.	Counter 0, 1 only
42H	01H	L1D_CACHE_LOCK.HIT	Counts retired load locks that hit in the L1 data cache or hit in an already allocated fill buffer. The lock portion of the load lock transaction must hit in the L1D.	The initial load will pull the lock into the L1 data cache. Counter 0, 1 only
42H	02H	L1D_CACHE_LOCK.S_STATE	Counts L1 data cache retired load locks that hit the target cache line in the shared state.	Counter 0, 1 only
42H	04H	L1D_CACHE_LOCK.E_STATE	Counts L1 data cache retired load locks that hit the target cache line in the exclusive state.	Counter 0, 1 only
42H	08H	L1D_CACHE_LOCK.M_STATE	Counts L1 data cache retired load locks that hit the target cache line in the modified state.	Counter 0, 1 only
43H	01H	L1D_ALL_REF.ANY	Counts all references (uncached, speculated and retired) to the L1 data cache, including all loads and stores with any memory types. The event counts memory accesses only when they are actually performed. For example, a load blocked by unknown store address and later performed is only counted once.	The event does not include non-memory accesses, such as I/O accesses. Counter 0, 1 only
43H	02H	L1D_ALL_REF.CACHEABLE	Counts all data reads and writes (speculated and retired) from cacheable memory, including locked operations.	Counter 0, 1 only
49H	01H	DTLB_MISSES.ANY	Counts the number of misses in the STLB which causes a page walk.	
49H	02H	DTLB_MISSES.WALK_COMPLETED	Counts number of misses in the STLB which resulted in a completed page walk.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
49H	10H	DTLB_MISSES.STLB_HIT	Counts the number of DTLB first level misses that hit in the second level TLB. This event is only relevant if the core contains multiple DTLB levels.	
49H	20H	DTLB_MISSES.PDE_MISS	Number of DTLB misses caused by low part of address, includes references to 2M pages because 2M pages do not use the PDE.	
49H	80H	DTLB_MISSES.LARGE_WALK_COMPLETED	Counts number of misses in the STLB which resulted in a completed page walk for large pages.	
4CH	01H	LOAD_HIT_PRE	Counts load operations sent to the L1 data cache while a previous SSE prefetch instruction to the same cache line has started prefetching but has not yet finished.	
4EH	01H	L1D_PREFETCH.REQUESTS	Counts number of hardware prefetch requests dispatched out of the prefetch FIFO.	
4EH	02H	L1D_PREFETCH.MISS	Counts number of hardware prefetch requests that miss the L1D. There are two prefetchers in the L1D. A streamer, which predicts lines sequentially after this one should be fetched, and the IP prefetcher that remembers access patterns for the current instruction. The streamer prefetcher stops on an L1D hit, while the IP prefetcher does not.	
4EH	04H	L1D_PREFETCH.TRIGGERS	Counts number of prefetch requests triggered by the Finite State Machine and pushed into the prefetch FIFO. Some of the prefetch requests are dropped due to overwrites or competition between the IP index prefetcher and streamer prefetcher. The prefetch FIFO contains 4 entries.	
51H	01H	L1D.REPL	Counts the number of lines brought into the L1 data cache.	Counter 0, 1 only
51H	02H	L1D.M_REPL	Counts the number of modified lines brought into the L1 data cache.	Counter 0, 1 only
51H	04H	L1D.M_EVICT	Counts the number of modified lines evicted from the L1 data cache due to replacement.	Counter 0, 1 only
51H	08H	L1D.M_SNOOP_EVICT	Counts the number of modified lines evicted from the L1 data cache due to snoop HITM intervention.	Counter 0, 1 only
52H	01H	L1D_CACHE_PREFETCH_LOCK_FB_HIT	Counts the number of cacheable load lock speculated instructions accepted into the fill buffer.	
53H	01H	L1D_CACHE_LOCK_FB_HIT	Counts the number of cacheable load lock speculated or retired instructions accepted into the fill buffer.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
63H	01H	CACHE_LOCK_CYCLES.L1D_L2	Cycle count during which the L1D and L2 are locked. A lock is asserted when there is a locked memory access, due to uncacheable memory, a locked operation that spans two cache lines, or a page walk from an uncacheable page table.	Counter 0, 1 only. L1D and L2 locks have a very high performance penalty and it is highly recommended to avoid such accesses.
63H	02H	CACHE_LOCK_CYCLES.L1D	Counts the number of cycles that cacheline in the L1 data cache unit is locked.	Counter 0, 1 only.
6CH	01H	IO_TRANSACTIONS	Counts the number of completed I/O transactions.	
80H	01H	L1I.HITS	Counts all instruction fetches that hit the L1 instruction cache.	
80H	02H	L1I.MISSES	Counts all instruction fetches that miss the L1I cache. This includes instruction cache misses, streaming buffer misses, victim cache misses and uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.	
80H	03H	L1I.READS	Counts all instruction fetches, including uncacheable fetches that bypass the L1I.	
80H	04H	L1I.CYCLES_STALLED	Cycle counts for which an instruction fetch stalls due to a L1I cache miss, ITLB miss or ITLB fault.	
82H	01H	LARGE_ITLB.HIT	Counts number of large ITLB hits.	
85H	01H	ITLB_MISSES.ANY	Counts the number of misses in all levels of the ITLB which causes a page walk.	
85H	02H	ITLB_MISSES.WALK_COMPLETED	Counts number of misses in all levels of the ITLB which resulted in a completed page walk.	
87H	01H	ILD_STALL.LCP	Cycles Instruction Length Decoder stalls due to length changing prefixes: 66, 67 or REX.W (for EM64T) instructions which change the length of the decoded instruction.	
87H	02H	ILD_STALL.MRU	Instruction Length Decoder stall cycles due to Brand Prediction Unit (PBU) Most Recently Used (MRU) bypass.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to a full instruction queue.	
87H	08H	ILD_STALL.REGEN	Counts the number of regen stalls.	
87H	0FH	ILD_STALL.ANY	Counts any cycles the Instruction Length Decoder is stalled.	
88H	01H	BR_INST_EXEC.COND	Counts the number of conditional near branch instructions executed, but not necessarily retired.	
88H	02H	BR_INST_EXEC.DIRECT	Counts all unconditional near branch instructions excluding calls and indirect branches.	
88H	04H	BR_INST_EXEC.INDIRECT_NON_CALL	Counts the number of executed indirect near branch instructions that are not calls.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
88H	07H	BR_INST_EXEC.NON_CALLS	Counts all non call near branch instructions executed, but not necessarily retired.	
88H	08H	BR_INST_EXEC.RETURN_NEAR	Counts indirect near branches that have a return mnemonic.	
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Counts unconditional near call branch instructions, excluding non call branch, executed.	
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Counts indirect near calls, including both register and memory indirect, executed.	
88H	30H	BR_INST_EXEC.NEAR_CALLS	Counts all near call branches executed, but not necessarily retired.	
88H	40H	BR_INST_EXEC.TAKEN	Counts taken near branches executed, but not necessarily retired.	
88H	7FH	BR_INST_EXEC.ANY	Counts all near executed branches (not necessarily retired). This includes only instructions and not micro-op branches. Frequent branching is not necessarily a major performance issue. However frequent branch mispredictions may be a problem.	
89H	01H	BR_MISP_EXEC.COND	Counts the number of mispredicted conditional near branch instructions executed, but not necessarily retired.	
89H	02H	BR_MISP_EXEC.DIRECT	Counts mispredicted macro unconditional near branch instructions, excluding calls and indirect branches (should always be 0).	
89H	04H	BR_MISP_EXEC.INDIRECT_NON_CALL	Counts the number of executed mispredicted indirect near branch instructions that are not calls.	
89H	07H	BR_MISP_EXEC.NON_CALLS	Counts mispredicted non call near branches executed, but not necessarily retired.	
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Counts mispredicted indirect branches that have a rear return mnemonic.	
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Counts mispredicted non-indirect near calls executed, (should always be 0).	
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Counts mispredicted indirect near calls exeucted, including both register and memory indirect.	
89H	30H	BR_MISP_EXEC.NEAR_CALLS	Counts all mispredicted near call branches executed, but not necessarily retired.	
89H	40H	BR_MISP_EXEC.TAKEN	Counts executed mispredicted near branches that are taken, but not necessarily retired.	
89H	7FH	BR_MISP_EXEC.ANY	Counts the number of mispredicted near branch instructions that were executed, but not necessarily retired.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A2H	01H	RESOURCE_STALLS.ANY	Counts the number of Allocator resource related stalls. Includes register renaming buffer entries, memory buffer entries. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, such as if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	Does not include stalls due to SuperQ (off core) queue full, too many cache misses, etc.
A2H	02H	RESOURCE_STALLS.LOAD	Counts the cycles of stall due to lack of load buffer for load operation.	
A2H	04H	RESOURCE_STALLS.RS_FULL	This event counts the number of cycles when the number of instructions in the pipeline waiting for execution reaches the limit the processor can handle. A high count of this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, or instructions dependent upon instructions further down the pipeline that have yet to retire.	When RS is full, new instructions can not enter the reservation station and start execution.
A2H	08H	RESOURCE_STALLS.STORE	This event counts the number of cycles that a resource related stall will occur due to the number of store instructions reaching the limit of the pipeline, (i.e. all store buffers are used). The stall ends when a store instruction commits its data to the cache or memory.	
A2H	10H	RESOURCE_STALLS.ROB_FULL	Counts the cycles of stall due to re-order buffer full.	
A2H	20H	RESOURCE_STALLS.FPCW	Counts the number of cycles while execution was stalled due to writing the floating-point unit (FPU) control word.	
A2H	40H	RESOURCE_STALLS.MXCSR	Stalls due to the MXCSR register rename occurring to close to a previous MXCSR rename. The MXCSR provides control and status for the MMX registers.	
A2H	80H	RESOURCE_STALLS.OTHER	Counts the number of cycles while execution was stalled due to other resource issues.	
A6H	01H	MACRO_INSTS.FUSIONS_DECODED	Counts the number of instructions decoded that are macro-fused but not necessarily executed or retired.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A7H	01H	BACLEAR_FORCE_IQ	Counts number of times a BACLEAR was forced by the Instruction Queue. The IQ is also responsible for providing conditional branch prediciton direction based on a static scheme and dynamic data provided by the L2 Branch Prediction Unit. If the conditional branch target is not found in the Target Array and the IQ predicts that the branch is taken, then the IQ will force the Branch Address Calculator to issue a BACLEAR. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
A8H	01H	LSD.UOPS	Counts the number of micro-ops delivered by loop stream detector.	Use cmask=1 and invert to count cycles
AEH	01H	ITLB_FLUSH	Counts the number of ITLB flushes.	
B0H	40H	OFFCORE_REQUESTS.L1D_WRITEBACK	Counts number of L1D writebacks to the uncore.	
B1H	01H	UOPS_EXECUTED.PORT0	Counts number of Uops executed that were issued on port 0. Port 0 handles integer arithmetic, SIMD and FP add Uops.	
B1H	02H	UOPS_EXECUTED.PORT1	Counts number of Uops executed that were issued on port 1. Port 1 handles integer arithmetic, SIMD, integer shift, FP multiply and FP divide Uops.	
B1H	04H	UOPS_EXECUTED.PORT2_CORE	Counts number of Uops executed that were issued on port 2. Port 2 handles the load Uops. This is a core count only and can not be collected per thread.	
B1H	08H	UOPS_EXECUTED.PORT3_CORE	Counts number of Uops executed that were issued on port 3. Port 3 handles store Uops. This is a core count only and can not be collected per thread.	
B1H	10H	UOPS_EXECUTED.PORT4_CORE	Counts number of Uops executed that where issued on port 4. Port 4 handles the value to be stored for the store Uops issued on port 3. This is a core count only and can not be collected per thread.	
B1H	1FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES_NO_PORT5	Counts cycles when the Uops executed were issued from any ports except port 5. Use Cmask=1 for active cycles; Cmask=0 for weighted cycles; Use CMask=1, Invert=1 to count P0-4 stalled cycles Use Cmask=1, Edge=1, Invert=1 to count P0-4 stalls.	
B1H	20H	UOPS_EXECUTED.PORT5	Counts number of Uops executed that where issued on port 5.	
B1H	3FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES	Counts cycles when the Uops are executing . Use Cmask=1 for active cycles; Cmask=0 for weighted cycles; Use CMask=1, Invert=1 to count P0-4 stalled cycles Use Cmask=1, Edge=1, Invert=1 to count P0-4 stalls.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B1H	40H	UOPS_EXECUTED.PORT015	Counts number of Uops executed that where issued on port 0, 1, or 5.	use cmask=1, invert=1 to count stall cycles
B1H	80H	UOPS_EXECUTED.PORT234	Counts number of Uops executed that where issued on port 2, 3, or 4.	
B2H	01H	OFFCORE_REQUESTS_SQ_FULL	Counts number of cycles the SQ is full to handle off-core requests.	
B7H	01H	OFF_CORE_RESPONSE_0	see Section 18.7.1.3, "Off-core Response Performance Monitoring in the Processor Core".	Requires programming MSR 01A6H
B8H	01H	SNOOP_RESPONSE.HIT	Counts HIT snoop response sent by this thread in response to a snoop request.	
B8H	02H	SNOOP_RESPONSE.HITE	Counts HIT E snoop response sent by this thread in response to a snoop request.	
B8H	04H	SNOOP_RESPONSE.HITM	Counts HIT M snoop response sent by this thread in response to a snoop request.	
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.8, "Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Westmere".	Requires programming MSR 01A7H
C0H	00H	INST_RETIRED.ANY_P	See Table 19-1 Notes: INST_RETIRED.ANY is counted by a designated fixed counter. INST_RETIRED.ANY_P is counted by a programmable counter and is an architectural performance event. Event is supported if CPUID.A.EBX[1] = 0.	Counting: Faulting executions of GETSEC/ VM entry/VM Exit/MWait will not count as retired instructions.
C0H	02H	INST_RETIRED.X87	Counts the number of MMX instructions retired.	
C0H	04H	INST_RETIRED.MMX	Counts the number of floating point computational operations retired: floating point computational operations executed by the assist handler and sub-operations of complex floating point instructions like transcendental instructions.	
C2H	01H	UOPS_RETIRED.ANY	Counts the number of micro-ops retired, (macro-fused=1, micro-fused=2, others=1; maximum count of 8 per cycle). Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists.	Use cmask=1 and invert to count active cycles or stalled cycles
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	
C2H	04H	UOPS_RETIRED.MACRO_FUSED	Counts number of macro-fused uops retired.	
C3H	01H	MACHINE_CLEARS.CYCLES	Counts the cycles machine clear is asserted.	
C3H	02H	MACHINE_CLEARS.MEM_ORDER	Counts the number of machine clears due to memory order conflicts.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C3H	04H	MACHINE_CLEARS.SMC	Counts the number of times that a program writes to a code section. Self-modifying code causes a sever penalty in all Intel 64 and IA-32 processors. The modified cache line is written back to the L2 and L3caches.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement	See Table 19-1
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Counts the number of direct & indirect near unconditional calls retired.	
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement	See Table 19-1
C5H	02H	BR_MISP_RETIRED.NEAR_CALL	Counts mispredicted direct & indirect near unconditional retired calls.	
C7H	01H	SSEX_UOPS_RETIRED.PACKED_SINGLE	Counts SIMD packed single-precision floating point Uops retired.	
C7H	02H	SSEX_UOPS_RETIRED.SCALAR_SINGLE	Counts SIMD calar single-precision floating point Uops retired.	
C7H	04H	SSEX_UOPS_RETIRED.PACKED_DOUBLE	Counts SIMD packed double-precision floating point Uops retired.	
C7H	08H	SSEX_UOPS_RETIRED.SCALAR_DOUBLE	Counts SIMD scalar double-precision floating point Uops retired.	
C7H	10H	SSEX_UOPS_RETIRED.VECTOR_INTEGER	Counts 128-bit SIMD vector integer Uops retired.	
C8H	20H	ITLB_MISS_RETIRED	Counts the number of retired instructions that missed the ITLB when the instruction was fetched.	
CBH	01H	MEM_LOAD_RETIRED.L1D_HIT	Counts number of retired loads that hit the L1 data cache.	
CBH	02H	MEM_LOAD_RETIRED.L2_HIT	Counts number of retired loads that hit the L2 data cache.	
CBH	04H	MEM_LOAD_RETIRED.L3_UNSHARED_HIT	Counts number of retired loads that hit their own, unshared lines in the L3 cache.	
CBH	08H	MEM_LOAD_RETIRED.OTHER_CORE_L2_HIT_HITM	Counts number of retired loads that hit in a sibling core's L2 (on die core). Since the L3 is inclusive of all cores on the package, this is an L3 hit. This counts both clean or modified hits.	
CBH	10H	MEM_LOAD_RETIRED.L3_MISS	Counts number of retired loads that miss the L3 cache. The load was satisfied by a remote socket, local memory or an IOH.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
CBH	40H	MEM_LOAD_RETIRED.HIT_LFB	Counts number of retired loads that miss the L1D and the address is located in an allocated line fill buffer and will soon be committed to cache. This is counting secondary L1D misses.	
CBH	80H	MEM_LOAD_RETIRED.DTLB_MISS	Counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. Counts both primary and secondary misses to the TLB.	
CCH	01H	FP_MMX_TRANS.TO_FP	Counts the first floating-point instruction following any MMX instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
CCH	02H	FP_MMX_TRANS.TO_MMX	Counts the first MMX instruction following a floating-point instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
CCH	03H	FP_MMX_TRANS.ANY	Counts all transitions from floating point to MMX instructions and from MMX instructions to floating point instructions. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
D0H	01H	MACRO_INSTS.DECODED	Counts the number of instructions decoded, (but not necessarily executed or retired).	
D1H	02H	UOPS_DECODED.MS	Counts the number of Uops decoded by the Microcode Sequencer, MS. The MS delivers uops when the instruction is more than 4 uops long or a microcode assist is occurring.	
D1H	04H	UOPS_DECODED.ESP_FOLDING	Counts number of stack pointer (ESP) instructions decoded: push , pop , call , ret, etc. ESP instructions do not generate a Uop to increment or decrement ESP. Instead, they update an ESP_Offset register that keeps track of the delta to the current value of the ESP register.	
D1H	08H	UOPS_DECODED.ESP_SYNC	Counts number of stack pointer (ESP) sync operations where an ESP instruction is corrected by adding the ESP offset register to the current value of the ESP register.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D2H	01H	RAT_STALLS.FLAGS	Counts the number of cycles during which execution stalled due to several reasons, one of which is a partial flag register stall. A partial register stall may occur when two conditions are met: 1) an instruction modifies some, but not all, of the flags in the flag register and 2) the next instruction, which depends on flags, depends on flags that were not modified by this instruction.	
D2H	02H	RAT_STALLS.REGISTERS	This event counts the number of cycles instruction execution latency became longer than the defined latency because the instruction used a register that was partially written by previous instruction.	
D2H	04H	RAT_STALLS.ROB_READ_PORT	Counts the number of cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the out-of-order pipeline. Note that, at this stage in the pipeline, additional stalls may occur at the same cycle and prevent the stalled micro-ops from entering the pipe. In such a case, micro-ops retry entering the execution pipe in the next cycle and the ROB-read port stall is counted again.	
D2H	08H	RAT_STALLS.SCOREBOARD	Counts the cycles where we stall due to microarchitecturally required serialization. Microcode scoreboarding stalls.	
D2H	0FH	RAT_STALLS.ANY	Counts all Register Allocation Table stall cycles due to: Cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the execution pipe. Cycles when partial register stalls occurred Cycles when flag stalls occurred Cycles floating-point unit (FPU) status word stalls occurred. To count each of these conditions separately use the events: RAT_STALLS.ROB_READ_PORT, RAT_STALLS.PARTIAL, RAT_STALLS.FLAGS, and RAT_STALLS.FPSW.	
D4H	01H	SEG_RENAME_STALLS	Counts the number of stall cycles due to the lack of renaming resources for the ES, DS, FS, and GS segment registers. If a segment is renamed but not retired and a second update to the same segment occurs, a stall occurs in the front-end of the pipeline until the renamed segment retires.	
D5H	01H	ES_REG_RENAMES	Counts the number of times the ES segment register is renamed.	
DBH	01H	UOP_UNFUSION	Counts unfusion events due to floating point exception to a fused uop.	
E0H	01H	BR_INST_DECODED	Counts the number of branch instructions decoded.	
E5H	01H	BPU_MISSED_CALL_RET	Counts number of times the Branch Prediction Unit missed predicting a call or return branch.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
E6H	01H	BACLEAR.CLEAR	Counts the number of times the front end is resteered, mainly when the Branch Prediction Unit cannot provide a correct prediction and this is corrected by the Branch Address Calculator at the front end. This can occur if the code has many branches such that they cannot be consumed by the BPU. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline. The effect on total execution time depends on the surrounding code.	
E6H	02H	BACLEAR.BAD_TARGET	Counts number of Branch Address Calculator clears (BACLEAR) asserted due to conditional branch instructions in which there was a target hit but the direction was wrong. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
E8H	01H	BPU_CLEARS.EARLY	Counts early (normal) Branch Prediction Unit clears: BPU predicted a taken branch after incorrectly assuming that it was not taken.	The BPU clear leads to 2 cycle bubble in the Front End.
E8H	02H	BPU_CLEARS.LATE	Counts late Branch Prediction Unit clears due to Most Recently Used conflicts. The PBU clear leads to a 3 cycle bubble in the Front End.	
F0H	01H	L2_TRANSACTIONS.LOAD	Counts L2 load operations due to HW prefetch or demand loads.	
F0H	02H	L2_TRANSACTIONS.RFO	Counts L2 RFO operations due to HW prefetch or demand RFOs.	
F0H	04H	L2_TRANSACTIONS.IFETCH	Counts L2 instruction fetch operations due to HW prefetch or demand ifetch.	
F0H	08H	L2_TRANSACTIONS.PREFETCH	Counts L2 prefetch operations.	
F0H	10H	L2_TRANSACTIONS.L1D_WB	Counts L1D writeback operations to the L2.	
F0H	20H	L2_TRANSACTIONS.FILL	Counts L2 cache line fill operations due to load, RFO, L1D writeback or prefetch.	
F0H	40H	L2_TRANSACTIONS.WB	Counts L2 writeback operations to the L3.	
F0H	80H	L2_TRANSACTIONS.ANY	Counts all L2 cache operations.	
F1H	02H	L2_LINES_IN.S_STATE	Counts the number of cache lines allocated in the L2 cache in the S (shared) state.	
F1H	04H	L2_LINES_IN.E_STATE	Counts the number of cache lines allocated in the L2 cache in the E (exclusive) state.	
F1H	07H	L2_LINES_IN.ANY	Counts the number of cache lines allocated in the L2 cache.	
F2H	01H	L2_LINES_OUT.DEMAND_CLEAN	Counts L2 clean cache lines evicted by a demand request.	
F2H	02H	L2_LINES_OUT.DEMAND_DIRTY	Counts L2 dirty (modified) cache lines evicted by a demand request.	

Table 19-13 Non-Architectural Performance Events In the Processor Core for Intel® Core™ i7 Processor and Intel® Xeon® Processor 5500 Series (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
F2H	04H	L2_LINES_OUT.PREFETCH_CLEAN	Counts L2 clean cache line evicted by a prefetch request.	
F2H	08H	L2_LINES_OUT.PREFETCH_DIRTY	Counts L2 modified cache line evicted by a prefetch request.	
F2H	0FH	L2_LINES_OUT.ANY	Counts all L2 cache lines evicted for any reason.	
F4H	10H	SQ_MISC.SPLIT_LOCK	Counts the number of SQ lock splits across a cache line.	
F6H	01H	SQ_FULL_STALL_CYCLES	Counts cycles the Super Queue is full. Neither of the threads on this core will be able to access the uncore.	
F7H	01H	FP_ASSIST.ALL	Counts the number of floating point operations executed that required micro-code assist intervention. Assists are required in the following cases: SSE instructions, (Denormal input when the DAZ flag is off or Underflow result when the FTZ flag is off): x87 instructions, (NaN or denormal are loaded to a register or used as input from memory, Division by 0 or Underflow output).	
F7H	02H	FP_ASSIST.OUTPUT	Counts number of floating point micro-code assist when the output value (destination register) is invalid.	
F7H	04H	FP_ASSIST.INPUT	Counts number of floating point micro-code assist when the input value (one of the source operands to an FP instruction) is invalid.	
FDH	01H	SIMD_INT_64.PACKED_MPY	Counts number of SID integer 64 bit packed multiply operations.	
FDH	02H	SIMD_INT_64.PACKED_SHIFT	Counts number of SID integer 64 bit packed shift operations.	
FDH	04H	SIMD_INT_64.PACK	Counts number of SID integer 64 bit pack operations.	
FDH	08H	SIMD_INT_64.UNPACK	Counts number of SID integer 64 bit unpack operations.	
FDH	10H	SIMD_INT_64.PACKED_LOGICAL	Counts number of SID integer 64 bit logical operations.	
FDH	20H	SIMD_INT_64.PACKED_ARITH	Counts number of SID integer 64 bit arithmetic operations.	
FDH	40H	SIMD_INT_64.SHUFFLE_MOVE	Counts number of SID integer 64 bit shift or move operations.	

…

19.7 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME WESTMERE

Intel 64 processors based on Intel® microarchitecture code name Westmere support the architectural and non-architectural performance-monitoring events listed in Table 19-1 and Table 19-15. Table 19-15 applies to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_25H, 06_2CH. In addition, these processors (CPUID signature of DisplayFamily_DisplayModel 06_25H, 06_2CH) also support the following non-architectural, product-specific uncore performance-monitoring events listed in Table 19-16. Fixed counters support the architecture events defined in Table 19-18.

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LOAD_BLOCK.OVERLAP_STORE	Loads that partially overlap an earlier store.	
04H	07H	SB_DRAIN.ANY	All Store buffer stall cycles.	
05H	02H	MISALIGN_MEMORY.STORE	All store referenced with misaligned address.	
06H	04H	STORE_BLOCKS.AT_RET	Counts number of loads delayed with at-Retirement block code. The following loads need to be executed at retirement and wait for all senior stores on the same thread to be drained: load splitting across 4K boundary (page split), load accessing uncacheable (UC or WC) memory, load lock, and load with page table in UC or WC memory region.	
06H	08H	STORE_BLOCKS.L1D_BLOCK	Cacheable loads delayed with L1D block code.	
07H	01H	PARTIAL_ADDRESS_ALIAS	Counts false dependency due to partial address aliasing.	
08H	01H	DTLB_LOAD_MISSES.ANY	Counts all load misses that cause a page walk.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED	Counts number of completed page walks due to load miss in the STLB.	
08H	04H	DTLB_LOAD_MISSES.WALK_CYCLES	Cycles PMH is busy with a page walk due to a load miss in the STLB.	
08H	10H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits.	
08H	20H	DTLB_LOAD_MISSES.PDE_MISS	Number of DTLB cache load misses where the low part of the linear to physical address translation was missed.	
0BH	01H	MEM_INST_RETIRED.LOADS	Counts the number of instructions with an architecturally-visible load retired on the architected path.	
0BH	02H	MEM_INST_RETIRED.STORES	Counts the number of instructions with an architecturally-visible store retired on the architected path.	
0BH	10H	MEM_INST_RETIRED.LATENCY_ABOVE_THRESHOLD	Counts the number of instructions exceeding the latency specified with ld_lat facility.	In conjunction with ld_lat facility

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
0CH	01H	MEM_STORE_RETIRED.DTLB_MISS	The event counts the number of retired stores that missed the DTLB. The DTLB miss is not counted if the store operation causes a fault. Does not counter prefetches. Counts both primary and secondary misses to the TLB.	
0EH	01H	UOPS_ISSUED.ANY	Counts the number of Uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	
0EH	01H	UOPS_ISSUED.STALLED_CYCLES	Counts the number of cycles no Uops issued by the Register Allocation Table to the Reservation Station, i.e. the UOPs issued from the front end to the back end.	set "invert=1, cmask = 1"
0EH	02H	UOPS_ISSUED.FUSED	Counts the number of fused Uops that were issued from the Register Allocation Table to the Reservation Station.	
0FH	01H	MEM_UNCORE_RETIRED.UNKNOWN_SOURCE	Load instructions retired with unknown LLC miss (Precise Event).	Applicable to one and two sockets
0FH	02H	MEM_UNCORE_RETIRED.OHTER_CORE_L2_HIT	Load instructions retired that HIT modified data in sibling core (Precise Event).	Applicable to one and two sockets
0FH	04H	MEM_UNCORE_RETIRED.REMOTE_HITM	Load instructions retired that HIT modified data in remote socket (Precise Event).	Applicable to two sockets only
0FH	08H	MEM_UNCORE_RETIRED.LOCAL_DRAM_AND_REMOTE_CACHE_HIT	Load instructions retired local dram and remote cache HIT data sources (Precise Event).	Applicable to one and two sockets
0FH	10H	MEM_UNCORE_RETIRED.REMOTE_DRAM	Load instructions retired remote DRAM and remote home-remote cache HITM (Precise Event).	Applicable to two sockets only
0FH	20H	MEM_UNCORE_RETIRED.OTHER_LLC_MISS	Load instructions retired other LLC miss (Precise Event).	Applicable to two sockets only
0FH	80H	MEM_UNCORE_RETIRED.UNCACHEABLE	Load instructions retired I/O (Precise Event).	Applicable to one and two sockets
10H	01H	FP_COMP_OPS_EXE.X87	Counts the number of FP Computational Uops Executed. The number of FADD, FSUB, FCOM, FMULs, integer MULsand IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs, and IDIVs. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	
10H	02H	FP_COMP_OPS_EXE.MMX	Counts number of MMX Uops executed.	
10H	04H	FP_COMP_OPS_EXE.SSE_FP	Counts number of SSE and SSE2 FP uops executed.	
10H	08H	FP_COMP_OPS_EXE.SSE2_INTEGER	Counts number of SSE2 integer uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED	Counts number of SSE FP packed uops executed.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR	Counts number of SSE FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_SINGLE_PRECISION	Counts number of SSE* FP single precision uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_DOUBLE_PRECISION	Counts number of SSE* FP double precision uops executed.	
12H	01H	SIMD_INT_128.PACKED_MPY	Counts number of 128 bit SIMD integer multiply operations.	
12H	02H	SIMD_INT_128.PACKED_SHIFT	Counts number of 128 bit SIMD integer shift operations.	
12H	04H	SIMD_INT_128.PACK	Counts number of 128 bit SIMD integer pack operations.	
12H	08H	SIMD_INT_128.UNPACK	Counts number of 128 bit SIMD integer unpack operations.	
12H	10H	SIMD_INT_128.PACKED_LOGICAL	Counts number of 128 bit SIMD integer logical operations.	
12H	20H	SIMD_INT_128.PACKED_ARITH	Counts number of 128 bit SIMD integer arithmetic operations.	
12H	40H	SIMD_INT_128.SHUFFLE_MOVE	Counts number of 128 bit SIMD integer shuffle and move operations.	
13H	01H	LOAD_DISPATCH.RS	Counts number of loads dispatched from the Reservation Station that bypass the Memory Order Buffer.	
13H	02H	LOAD_DISPATCH.RS_DELAYED	Counts the number of delayed RS dispatches at the stage latch. If an RS dispatch can not bypass to LB, it has another chance to dispatch from the one-cycle delayed staging latch before it is written into the LB.	
13H	04H	LOAD_DISPATCH.MOB	Counts the number of loads dispatched from the Reservation Station to the Memory Order Buffer.	
13H	07H	LOAD_DISPATCH.ANY	Counts all loads dispatched from the Reservation Station.	
14H	01H	ARITH.CYCLES_DIV_BUSY	Counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE. Set 'edge =1, invert=1, cmask=1' to count the number of divides.	Count may be incorrect When SMT is on
14H	02H	ARITH.MUL	Counts the number of multiply operations executed. This includes integer as well as floating point multiply operations but excludes DPPS mul and MPSAD.	Count may be incorrect When SMT is on

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
17H	01H	INST_QUEUE_WRITES	Counts the number of instructions written into the instruction queue every cycle.	
18H	01H	INST_DECODED.DEC0	Counts number of instructions that require decoder 0 to be decoded. Usually, this means that the instruction maps to more than 1 uop.	
19H	01H	TWO_UOP_INSTS_DECODED	An instruction that generates two uops was decoded.	
1EH	01H	INST_QUEUE_WRITE_CYCLES	This event counts the number of cycles during which instructions are written to the instruction queue. Dividing this counter by the number of instructions written to the instruction queue (INST_QUEUE_WRITES) yields the average number of instructions decoded each cycle. If this number is less than four and the pipe stalls, this indicates that the decoder is failing to decode enough instructions per cycle to sustain the 4-wide pipeline.	If SSE* instructions that are 6 bytes or longer arrive one after another, then front end throughput may limit execution speed.
20H	01H	LSD_OVERFLOW	Number of loops that can not stream from the instruction queue.	
24H	01H	L2_RQSTS.LD_HIT	Counts number of loads that hit the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches. L2 loads can be rejected for various reasons. Only non rejected loads are counted.	
24H	02H	L2_RQSTS.LD_MISS	Counts the number of loads that miss the L2 cache. L2 loads include both L1D demand misses as well as L1D prefetches.	
24H	03H	L2_RQSTS.LOADS	Counts all L2 load requests. L2 loads include both L1D demand misses as well as L1D prefetches.	
24H	04H	L2_RQSTS.RFO_HIT	Counts the number of store RFO requests that hit the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches. Count includes WC memory requests, where the data is not fetched but the permission to write the line is required.	
24H	08H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches.	
24H	0CH	L2_RQSTS.RFOS	Counts all L2 store RFO requests. L2 RFO requests include both L1D demand RFO misses as well as L1D RFO prefetches..	
24H	10H	L2_RQSTS.IFETCH_HIT	Counts number of instruction fetches that hit the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	20H	L2_RQSTS.IFETCH_MISS	Counts number of instruction fetches that miss the L2 cache. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	30H	L2_RQSTS.IFETCHES	Counts all instruction fetches. L2 instruction fetches include both L1I demand misses as well as L1I instruction prefetches.	
24H	40H	L2_RQSTS.PREFETCH_HIT	Counts L2 prefetch hits for both code and data.	
24H	80H	L2_RQSTS.PREFETCH_MISS	Counts L2 prefetch misses for both code and data.	
24H	C0H	L2_RQSTS.PREFETCHES	Counts all L2 prefetches for both code and data.	
24H	AAH	L2_RQSTS.MISS	Counts all L2 misses for both code and data.	
24H	FFH	L2_RQSTS.REFERENCES	Counts all L2 requests for both code and data.	
26H	01H	L2_DATA_RQSTS.DEMAND.I_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the I (invalid) state, i.e. a cache miss. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	02H	L2_DATA_RQSTS.DEMAND.S_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the S (shared) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	04H	L2_DATA_RQSTS.DEMAND.E_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the E (exclusive) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	08H	L2_DATA_RQSTS.DEMAND.M_STATE	Counts number of L2 data demand loads where the cache line to be loaded is in the M (modified) state. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	0FH	L2_DATA_RQSTS.DEMAND.MESI	Counts all L2 data demand requests. L2 demand loads are both L1D demand misses and L1D prefetches.	
26H	10H	L2_DATA_RQSTS.PREFETCH.I_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the I (invalid) state, i.e. a cache miss.	
26H	20H	L2_DATA_RQSTS.PREFETCH.S_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the S (shared) state. A prefetch RFO will miss on an S state line, while a prefetch read will hit on an S state line.	
26H	40H	L2_DATA_RQSTS.PREFETCH.E_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the E (exclusive) state.	
26H	80H	L2_DATA_RQSTS.PREFETCH.M_STATE	Counts number of L2 prefetch data loads where the cache line to be loaded is in the M (modified) state.	
26H	F0H	L2_DATA_RQSTS.PREFETCH.MESI	Counts all L2 prefetch requests.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
26H	FFH	L2_DATA_RQSTS.ANY	Counts all L2 data requests.	
27H	01H	L2_WRITE.RFO.I_STATE	Counts number of L2 demand store RFO requests where the cache line to be loaded is in the I (invalid) state, i.e, a cache miss. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	02H	L2_WRITE.RFO.S_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the S (shared) state. The L1D prefetcher does not issue a RFO prefetch,.	This is a demand RFO request
27H	08H	L2_WRITE.RFO.M_STATE	Counts number of L2 store RFO requests where the cache line to be loaded is in the M (modified) state. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	0EH	L2_WRITE.RFO.HIT	Counts number of L2 store RFO requests where the cache line to be loaded is in either the S, E or M states. The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	0FH	L2_WRITE.RFO.MESI	Counts all L2 store RFO requests.The L1D prefetcher does not issue a RFO prefetch.	This is a demand RFO request
27H	10H	L2_WRITE.LOCK.I_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the I (invalid) state, i.e. a cache miss.	
27H	20H	L2_WRITE.LOCK.S_STATE	Counts number of L2 lock RFO requests where the cache line to be loaded is in the S (shared) state.	
27H	40H	L2_WRITE.LOCK.E_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the E (exclusive) state.	
27H	80H	L2_WRITE.LOCK.M_STATE	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in the M (modified) state.	
27H	E0H	L2_WRITE.LOCK.HIT	Counts number of L2 demand lock RFO requests where the cache line to be loaded is in either the S, E, or M state.	
27H	F0H	L2_WRITE.LOCK.MESI	Counts all L2 demand lock RFO requests.	
28H	01H	L1D_WB_L2.I_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the I (invalid) state, i.e. a cache miss.	
28H	02H	L1D_WB_L2.S_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the S state.	
28H	04H	L1D_WB_L2.E_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the E (exclusive) state.	
28H	08H	L1D_WB_L2.M_STATE	Counts number of L1 writebacks to the L2 where the cache line to be written is in the M (modified) state.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
28H	0FH	L1D_WB_L2.MESI	Counts all L1 writebacks to the L2 .	
2EH	41H	L3_LAT_CACHE.MISS	Counts uncore Last Level Cache misses. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	see Table 19-1
2EH	4FH	L3_LAT_CACHE.REFERENCE	Counts uncore Last Level Cache references. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	see Table 19-1
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.	see Table 19-1
3CH	01H	CPU_CLK_UNHALTED.REF_P	Increments at the frequency of TSC when not halted.	see Table 19-1
49H	01H	DTLB_MISSES.ANY	Counts the number of misses in the STLB which causes a page walk.	
49H	02H	DTLB_MISSES.WALK_COMPLETED	Counts number of misses in the STLB which resulted in a completed page walk.	
49H	04H	DTLB_MISSES.WALK_CYCLES	Counts cycles of page walk due to misses in the STLB.	
49H	10H	DTLB_MISSES.STLB_HIT	Counts the number of DTLB first level misses that hit in the second level TLB. This event is only relevant if the core contains multiple DTLB levels.	
49H	20H	DTLB_MISSES.PDE_MISS	Number of DTLB misses caused by low part of address, includes references to 2M pages because 2M pages do not use the PDE.	
49H	80H	DTLB_MISSES.LARGE_WALK_COMPLETED	Counts number of completed large page walks due to misses in the STLB.	
4CH	01H	LOAD_HIT_PRE	Counts load operations sent to the L1 data cache while a previous SSE prefetch instruction to the same cache line has started prefetching but has not yet finished.	Counter 0, 1 only
4EH	01H	L1D_PREFETCH.REQUESTS	Counts number of hardware prefetch requests dispatched out of the prefetch FIFO.	Counter 0, 1 only
4EH	02H	L1D_PREFETCH.MISS	Counts number of hardware prefetch requests that miss the L1D. There are two prefetchers in the L1D. A streamer, which predicts lines sequentially after this one should be fetched, and the IP prefetcher that remembers access patterns for the current instruction. The streamer prefetcher stops on an L1D hit, while the IP prefetcher does not.	Counter 0, 1 only

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
4EH	04H	L1D_PREFETCH.TRIGGERS	Counts number of prefetch requests triggered by the Finite State Machine and pushed into the prefetch FIFO. Some of the prefetch requests are dropped due to overwrites or competition between the IP index prefetcher and streamer prefetcher. The prefetch FIFO contains 4 entries.	Counter 0, 1 only
4FH	10H	EPT.WALK_CYCLES	Counts Extended Page walk cycles.	
51H	01H	L1D.REPL	Counts the number of lines brought into the L1 data cache.	Counter 0, 1 only
51H	02H	L1D.M_REPL	Counts the number of modified lines brought into the L1 data cache.	Counter 0, 1 only
51H	04H	L1D.M_EVICT	Counts the number of modified lines evicted from the L1 data cache due to replacement.	Counter 0, 1 only
51H	08H	L1D.M_SNOOP_EVICT	Counts the number of modified lines evicted from the L1 data cache due to snoop HITM intervention.	Counter 0, 1 only
52H	01H	L1D_CACHE_PREFETCH_LOCK_FB_HIT	Counts the number of cacheable load lock speculated instructions accepted into the fill buffer.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_DATA	Counts weighted cycles of offcore demand data read requests. Does not include L2 prefetch requests.	counter 0
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_CODE	Counts weighted cycles of offcore demand code read requests. Does not include L2 prefetch requests.	counter 0
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND.RFO	Counts weighted cycles of offcore demand RFO requests. Does not include L2 prefetch requests.	counter 0
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ANY.READ	Counts weighted cycles of offcore read requests of any kind. Include L2 prefetch requests.	counter 0
63H	01H	CACHE_LOCK_CYCLES.L1D_L2	Cycle count during which the L1D and L2 are locked. A lock is asserted when there is a locked memory access, due to uncacheable memory, a locked operation that spans two cache lines, or a page walk from an uncacheable page table. This event does not cause locks, it merely detects them.	Counter 0, 1 only. L1D and L2 locks have a very high performance penalty and it is highly recommended to avoid such accesses.
63H	02H	CACHE_LOCK_CYCLES.L1D	Counts the number of cycles that cacheline in the L1 data cache unit is locked.	Counter 0, 1 only.
6CH	01H	IO_TRANSACTIONS	Counts the number of completed I/O transactions.	
80H	01H	L1I.HITS	Counts all instruction fetches that hit the L1 instruction cache.	
80H	02H	L1I.MISSES	Counts all instruction fetches that miss the L1I cache. This includes instruction cache misses, streaming buffer misses, victim cache misses and uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
80H	03H	L1I.READS	Counts all instruction fetches, including uncacheable fetches that bypass the L1I.	
80H	04H	L1I.CYCLES_STALLED	Cycle counts for which an instruction fetch stalls due to a L1I cache miss, ITLB miss or ITLB fault.	
82H	01H	LARGE_ITLB.HIT	Counts number of large ITLB hits.	
85H	01H	ITLB_MISSES.ANY	Counts the number of misses in all levels of the ITLB which causes a page walk.	
85H	02H	ITLB_MISSES.WALK_COMPLETED	Counts number of misses in all levels of the ITLB which resulted in a completed page walk.	
85H	04H	ITLB_MISSES.WALK_CYCLES	Counts ITLB miss page walk cycles.	
85H	10H	ITLB_MISSES.STLB_HIT	Counts number of ITLB first level miss but second level hits	
85H	80H	ITLB_MISSES.LARGE_WALK_COMPLETED	Counts number of completed large page walks due to misses in the STLB.	
87H	01H	ILD_STALL.LCP	Cycles Instruction Length Decoder stalls due to length changing prefixes: 66, 67 or REX.W (for EM64T) instructions which change the length of the decoded instruction.	
87H	02H	ILD_STALL.MRU	Instruction Length Decoder stall cycles due to Brand Prediction Unit (PBU) Most Recently Used (MRU) bypass.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to a full instruction queue.	
87H	08H	ILD_STALL.REGEN	Counts the number of regen stalls.	
87H	0FH	ILD_STALL.ANY	Counts any cycles the Instruction Length Decoder is stalled.	
88H	01H	BR_INST_EXEC.COND	Counts the number of conditional near branch instructions executed, but not necessarily retired.	
88H	02H	BR_INST_EXEC.DIRECT	Counts all unconditional near branch instructions excluding calls and indirect branches.	
88H	04H	BR_INST_EXEC.INDIRECT_NON_CALL	Counts the number of executed indirect near branch instructions that are not calls.	
88H	07H	BR_INST_EXEC.NON_CALLS	Counts all non call near branch instructions executed, but not necessarily retired.	
88H	08H	BR_INST_EXEC.RETURN_NEAR	Counts indirect near branches that have a return mnemonic.	
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Counts unconditional near call branch instructions, excluding non call branch, executed.	
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Counts indirect near calls, including both register and memory indirect, executed.	
88H	30H	BR_INST_EXEC.NEAR_CALLS	Counts all near call branches executed, but not necessarily retired.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
88H	40H	BR_INST_EXEC.TAKEN	Counts taken near branches executed, but not necessarily retired.	
88H	7FH	BR_INST_EXEC.ANY	Counts all near executed branches (not necessarily retired). This includes only instructions and not micro-op branches. Frequent branching is not necessarily a major performance issue. However frequent branch mispredictions may be a problem.	
89H	01H	BR_MISP_EXEC.COND	Counts the number of mispredicted conditional near branch instructions executed, but not necessarily retired.	
89H	02H	BR_MISP_EXEC.DIRECT	Counts mispredicted macro unconditional near branch instructions, excluding calls and indirect branches (should always be 0).	
89H	04H	BR_MISP_EXEC.INDIRECT_NON_CALL	Counts the number of executed mispredicted indirect near branch instructions that are not calls.	
89H	07H	BR_MISP_EXEC.NON_CALLS	Counts mispredicted non call near branches executed, but not necessarily retired.	
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Counts mispredicted indirect branches that have a rear return mnemonic.	
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Counts mispredicted non-indirect near calls executed, (should always be 0).	
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Counts mispredicted indirect near calls exeucted, including both register and memory indirect.	
89H	30H	BR_MISP_EXEC.NEAR_CALLS	Counts all mispredicted near call branches executed, but not necessarily retired.	
89H	40H	BR_MISP_EXEC.TAKEN	Counts executed mispredicted near branches that are taken, but not necessarily retired.	
89H	7FH	BR_MISP_EXEC.ANY	Counts the number of mispredicted near branch instructions that were executed, but not necessarily retired.	
A2H	01H	RESOURCE_STALLS.ANY	Counts the number of Allocator resource related stalls. Includes register renaming buffer entries, memory buffer entries. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, such as if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	Does not include stalls due to SuperQ (off core) queue full, too many cache misses, etc.
A2H	02H	RESOURCE_STALLS.LOAD	Counts the cycles of stall due to lack of load buffer for load operation.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
A2H	04H	RESOURCE_STALLS.RS_FULL	This event counts the number of cycles when the number of instructions in the pipeline waiting for execution reaches the limit the processor can handle. A high count of this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, or instructions dependent upon instructions further down the pipeline that have yet to retire.	When RS is full, new instructions can not enter the reservation station and start execution.
A2H	08H	RESOURCE_STALLS.STORE	This event counts the number of cycles that a resource related stall will occur due to the number of store instructions reaching the limit of the pipeline, (i.e. all store buffers are used). The stall ends when a store instruction commits its data to the cache or memory.	
A2H	10H	RESOURCE_STALLS.ROB_FULL	Counts the cycles of stall due to re-order buffer full.	
A2H	20H	RESOURCE_STALLS.FPCW	Counts the number of cycles while execution was stalled due to writing the floating-point unit (FPU) control word.	
A2H	40H	RESOURCE_STALLS.MXCSR	Stalls due to the MXCSR register rename occurring to close to a previous MXCSR rename. The MXCSR provides control and status for the MMX registers.	
A2H	80H	RESOURCE_STALLS.OTHER	Counts the number of cycles while execution was stalled due to other resource issues.	
A6H	01H	MACRO_INSTS.FUSIONS_DECODED	Counts the number of instructions decoded that are macro-fused but not necessarily executed or retired.	
A7H	01H	BACLEAR_FORCE_IQ	Counts number of times a BACLEAR was forced by the Instruction Queue. The IQ is also responsible for providing conditional branch prediciton direction based on a static scheme and dynamic data provided by the L2 Branch Prediction Unit. If the conditional branch target is not found in the Target Array and the IQ predicts that the branch is taken, then the IQ will force the Branch Address Calculator to issue a BACLEAR. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
A8H	01H	LSD.UOPS	Counts the number of micro-ops delivered by loop stream detector.	Use cmask=1 and invert to count cycles
AEH	01H	ITLB_FLUSH	Counts the number of ITLB flushes.	
B0H	01H	OFFCORE_REQUESTS.DEMAND.READ_DATA	Counts number of offcore demand data read requests. Does not count L2 prefetch requests.	
B0H	02H	OFFCORE_REQUESTS.DEMAND.READ_CODE	Counts number of offcore demand code read requests. Does not count L2 prefetch requests.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B0H	04H	OFFCORE_REQUESTS.DEMAND.RFO	Counts number of offcore demand RFO requests. Does not count L2 prefetch requests.	
B0H	08H	OFFCORE_REQUESTS.ANY.READ	Counts number of offcore read requests. Includes L2 prefetch requests.	
B0H	10H	OFFCORE_REQUESTS.ANY.RFO	Counts number of offcore RFO requests. Includes L2 prefetch requests.	
B0H	40H	OFFCORE_REQUESTS.L1D_WRITEBACK	Counts number of L1D writebacks to the uncore.	
B0H	80H	OFFCORE_REQUESTS.ANY	Counts all offcore requests.	
B1H	01H	UOPS_EXECUTED.PORT0	Counts number of Uops executed that were issued on port 0. Port 0 handles integer arithmetic, SIMD and FP add Uops.	
B1H	02H	UOPS_EXECUTED.PORT1	Counts number of Uops executed that were issued on port 1. Port 1 handles integer arithmetic, SIMD, integer shift, FP multiply and FP divide Uops.	
B1H	04H	UOPS_EXECUTED.PORT2_CORE	Counts number of Uops executed that were issued on port 2. Port 2 handles the load Uops. This is a core count only and can not be collected per thread.	
B1H	08H	UOPS_EXECUTED.PORT3_CORE	Counts number of Uops executed that were issued on port 3. Port 3 handles store Uops. This is a core count only and can not be collected per thread.	
B1H	10H	UOPS_EXECUTED.PORT4_CORE	Counts number of Uops executed that where issued on port 4. Port 4 handles the value to be stored for the store Uops issued on port 3. This is a core count only and can not be collected per thread.	
B1H	1FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES_NO_PORT5	Counts number of cycles there are one or more uops being executed and were issued on ports 0-4. This is a core count only and can not be collected per thread.	
B1H	20H	UOPS_EXECUTED.PORT5	Counts number of Uops executed that where issued on port 5.	
B1H	3FH	UOPS_EXECUTED.CORE_ACTIVE_CYCLES	Counts number of cycles there are one or more uops being executed on any ports. This is a core count only and can not be collected per thread.	
B1H	40H	UOPS_EXECUTED.PORT015	Counts number of Uops executed that where issued on port 0, 1, or 5.	use cmask=1, invert=1 to count stall cycles
B1H	80H	UOPS_EXECUTED.PORT234	Counts number of Uops executed that where issued on port 2, 3, or 4.	
B2H	01H	OFFCORE_REQUESTS_SQ_FULL	Counts number of cycles the SQ is full to handle off-core requests.	
B3H	01H	SNOOPQ_REQUESTS_OUTSTANDING.DATA	Counts weighted cycles of snoopq requests for data. Counter 0 only.	Use cmask=1 to count cycles not empty.

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B3H	02H	SNOOPQ_REQUESTS_OUTSTANDING.INVALIDATE	Counts weighted cycles of snoopq invalidate requests. Counter 0 only.	Use cmask=1 to count cycles not empty.
B3H	04H	SNOOPQ_REQUESTS_OUTSTANDING.CODE	Counts weighted cycles of snoopq requests for code. Counter 0 only.	Use cmask=1 to count cycles not empty.
B4H	01H	SNOOPQ_REQUESTS.CODE	Counts the number of snoop code requests.	
B4H	02H	SNOOPQ_REQUESTS.DATA	Counts the number of snoop data requests.	
B4H	04H	SNOOPQ_REQUESTS.INVALIDATE	Counts the number of snoop invalidate requests.	
B7H	01H	OFF_CORE_RESPONSE_0	see Section 18.7.1.3, "Off-core Response Performance Monitoring in the Processor Core"	Requires programming MSR 01A6H
B8H	01H	SNOOP_RESPONSE.HIT	Counts HIT snoop response sent by this thread in response to a snoop request.	
B8H	02H	SNOOP_RESPONSE.HITE	Counts HIT E snoop response sent by this thread in response to a snoop request.	
B8H	04H	SNOOP_RESPONSE.HITM	Counts HIT M snoop response sent by this thread in response to a snoop request.	
BBH	01H	OFF_CORE_RESPONSE_1	see Section 18.7.1.3, "Off-core Response Performance Monitoring in the Processor Core"	Use MSR 01A7H
C0H	00H	INST_RETIRED.ANY_P	See Table 19-1 Notes: INST_RETIRED.ANY is counted by a designated fixed counter. INST_RETIRED.ANY_P is counted by a programmable counter and is an architectural performance event. Event is supported if CPUID.A.EBX[1] = 0.	Counting: Faulting executions of GETSEC/ VM entry/VM Exit/MWait will not count as retired instructions.
C0H	02H	INST_RETIRED.X87	Counts the number of floating point computational operations retired: floating point computational operations executed by the assist handler and sub-operations of complex floating point instructions like transcendental instructions.	
C0H	04H	INST_RETIRED.MMX	Counts the number of retired: MMX instructions.	
C2H	01H	UOPS_RETIRED.ANY	Counts the number of micro-ops retired, (macro-fused=1, micro-fused=2, others=1; maximum count of 8 per cycle). Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists.	Use cmask=1 and invert to count active cycles or stalled cycles
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle	
C2H	04H	UOPS_RETIRED.MACRO_FUSED	Counts number of macro-fused uops retired.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C3H	01H	MACHINE_CLEARS.CYCLES	Counts the cycles machine clear is asserted.	
C3H	02H	MACHINE_CLEARS.MEM_ORDER	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEARS.SMC	Counts the number of times that a program writes to a code section. Self-modifying code causes a sever penalty in all Intel 64 and IA-32 processors. The modified cache line is written back to the L2 and L3caches.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement	See Table 19-1
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Counts the number of direct & indirect near unconditional calls retired.	
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement	See Table 19-1
C5H	01H	BR_MISP_RETIRED.CONDITIONAL	Counts mispredicted conditional retired calls.	
C5H	02H	BR_MISP_RETIRED.NEAR_CALL	Counts mispredicted direct & indirect near unconditional retired calls.	
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Counts all mispredicted retired calls.	
C7H	01H	SSEX_UOPS_RETIRED.PACKED_SINGLE	Counts SIMD packed single-precision floating point Uops retired.	
C7H	02H	SSEX_UOPS_RETIRED.SCALAR_SINGLE	Counts SIMD calar single-precision floating point Uops retired.	
C7H	04H	SSEX_UOPS_RETIRED.PACKED_DOUBLE	Counts SIMD packed double-precision floating point Uops retired.	
C7H	08H	SSEX_UOPS_RETIRED.SCALAR_DOUBLE	Counts SIMD scalar double-precision floating point Uops retired.	
C7H	10H	SSEX_UOPS_RETIRED.VECTOR_INTEGER	Counts 128-bit SIMD vector integer Uops retired.	
C8H	20H	ITLB_MISS_RETIRED	Counts the number of retired instructions that missed the ITLB when the instruction was fetched.	
CBH	01H	MEM_LOAD_RETIRED.L1D_HIT	Counts number of retired loads that hit the L1 data cache.	
CBH	02H	MEM_LOAD_RETIRED.L2_HIT	Counts number of retired loads that hit the L2 data cache.	
CBH	04H	MEM_LOAD_RETIRED.L3_UNSHARED_HIT	Counts number of retired loads that hit their own, unshared lines in the L3 cache.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
CBH	08H	MEM_LOAD_RETIRED.OTHER_CORE_L2_HIT_HITM	Counts number of retired loads that hit in a sibling core's L2 (on die core). Since the L3 is inclusive of all cores on the package, this is an L3 hit. This counts both clean or modified hits.	
CBH	10H	MEM_LOAD_RETIRED.L3_MISS	Counts number of retired loads that miss the L3 cache. The load was satisfied by a remote socket, local memory or an IOH.	
CBH	40H	MEM_LOAD_RETIRED.HIT_LFB	Counts number of retired loads that miss the L1D and the address is located in an allocated line fill buffer and will soon be committed to cache. This is counting secondary L1D misses.	
CBH	80H	MEM_LOAD_RETIRED.DTLB_MISS	Counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. Counts both primary and secondary misses to the TLB.	
CCH	01H	FP_MMX_TRANS.TO_FP	Counts the first floating-point instruction following any MMX instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
CCH	02H	FP_MMX_TRANS.TO_MMX	Counts the first MMX instruction following a floating-point instruction. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
CCH	03H	FP_MMX_TRANS.ANY	Counts all transitions from floating point to MMX instructions and from MMX instructions to floating point instructions. You can use this event to estimate the penalties for the transitions between floating-point and MMX technology states.	
D0H	01H	MACRO_INSTS.DECODED	Counts the number of instructions decoded, (but not necessarily executed or retired).	
D1H	01H	UOPS_DECODED.STALL_CYCLES	Counts the cycles of decoder stalls. INV=1, Cmask=1	
D1H	02H	UOPS_DECODED.MS	Counts the number of Uops decoded by the Microcode Sequencer, MS. The MS delivers uops when the instruction is more than 4 uops long or a microcode assist is occurring.	
D1H	04H	UOPS_DECODED.ESP_FOLDING	Counts number of stack pointer (ESP) instructions decoded: push , pop , call , ret, etc. ESP instructions do not generate a Uop to increment or decrement ESP. Instead, they update an ESP_Offset register that keeps track of the delta to the current value of the ESP register.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	08H	UOPS_DECODED.ESP_SYNC	Counts number of stack pointer (ESP) sync operations where an ESP instruction is corrected by adding the ESP offset register to the current value of the ESP register.	
D2H	01H	RAT_STALLS.FLAGS	Counts the number of cycles during which execution stalled due to several reasons, one of which is a partial flag register stall. A partial register stall may occur when two conditions are met: 1) an instruction modifies some, but not all, of the flags in the flag register and 2) the next instruction, which depends on flags, depends on flags that were not modified by this instruction.	
D2H	02H	RAT_STALLS.REGISTERS	This event counts the number of cycles instruction execution latency became longer than the defined latency because the instruction used a register that was partially written by previous instruction.	
D2H	04H	RAT_STALLS.ROB_READ_PORT	Counts the number of cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the out-of-order pipeline. Note that, at this stage in the pipeline, additional stalls may occur at the same cycle and prevent the stalled micro-ops from entering the pipe. In such a case, micro-ops retry entering the execution pipe in the next cycle and the ROB-read port stall is counted again.	
D2H	08H	RAT_STALLS.SCOREBOARD	Counts the cycles where we stall due to microarchitecturally required serialization. Microcode scoreboarding stalls.	
D2H	0FH	RAT_STALLS.ANY	Counts all Register Allocation Table stall cycles due to: Cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the execution pipe. Cycles when partial register stalls occurred Cycles when flag stalls occurred Cycles floating-point unit (FPU) status word stalls occurred. To count each of these conditions separately use the events: RAT_STALLS.ROB_READ_PORT, RAT_STALLS.PARTIAL, RAT_STALLS.FLAGS, and RAT_STALLS.FPSW.	
D4H	01H	SEG_RENAME_STALLS	Counts the number of stall cycles due to the lack of renaming resources for the ES, DS, FS, and GS segment registers. If a segment is renamed but not retired and a second update to the same segment occurs, a stall occurs in the front-end of the pipeline until the renamed segment retires.	
D5H	01H	ES_REG_RENAMES	Counts the number of times the ES segment register is renamed.	
DBH	01H	UOP_UNFUSION	Counts unfusion events due to floating point exception to a fused uop.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
E0H	01H	BR_INST_DECODED	Counts the number of branch instructions decoded.	
E5H	01H	BPU_MISSED_CALL_RET	Counts number of times the Branch Prediciton Unit missed predicting a call or return branch.	
E6H	01H	BACLEAR.CLEAR	Counts the number of times the front end is resteered, mainly when the Branch Prediction Unit cannot provide a correct prediction and this is corrected by the Branch Address Calculator at the front end. This can occur if the code has many branches such that they cannot be consumed by the BPU. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline. The effect on total execution time depends on the surrounding code.	
E6H	02H	BACLEAR.BAD_TARGET	Counts number of Branch Address Calculator clears (BACLEAR) asserted due to conditional branch instructions in which there was a target hit but the direction was wrong. Each BACLEAR asserted by the BAC generates approximately an 8 cycle bubble in the instruction fetch pipeline.	
E8H	01H	BPU_CLEARS.EARLY	Counts early (normal) Branch Prediction Unit clears: BPU predicted a taken branch after incorrectly assuming that it was not taken.	The BPU clear leads to 2 cycle bubble in the Front End.
E8H	02H	BPU_CLEARS.LATE	Counts late Branch Prediction Unit clears due to Most Recently Used conflicts. The PBU clear leads to a 3 cycle bubble in the Front End.	
ECH	01H	THREAD_ACTIVE	Counts cycles threads are active.	
F0H	01H	L2_TRANSACTIONS.LOAD	Counts L2 load operations due to HW prefetch or demand loads.	
F0H	02H	L2_TRANSACTIONS.RFO	Counts L2 RFO operations due to HW prefetch or demand RFOs.	
F0H	04H	L2_TRANSACTIONS.IFETCH	Counts L2 instruction fetch operations due to HW prefetch or demand ifetch.	
F0H	08H	L2_TRANSACTIONS.PREFETCH	Counts L2 prefetch operations.	
F0H	10H	L2_TRANSACTIONS.L1D_WB	Counts L1D writeback operations to the L2.	
F0H	20H	L2_TRANSACTIONS.FILL	Counts L2 cache line fill operations due to load, RFO, L1D writeback or prefetch.	
F0H	40H	L2_TRANSACTIONS.WB	Counts L2 writeback operations to the L3.	
F0H	80H	L2_TRANSACTIONS.ANY	Counts all L2 cache operations.	
F1H	02H	L2_LINES_IN.S_STATE	Counts the number of cache lines allocated in the L2 cache in the S (shared) state.	
F1H	04H	L2_LINES_IN.E_STATE	Counts the number of cache lines allocated in the L2 cache in the E (exclusive) state.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
F1H	07H	L2_LINES_IN.ANY	Counts the number of cache lines allocated in the L2 cache.	
F2H	01H	L2_LINES_OUT.DEMAND_CLEAN	Counts L2 clean cache lines evicted by a demand request.	
F2H	02H	L2_LINES_OUT.DEMAND_DIRTY	Counts L2 dirty (modified) cache lines evicted by a demand request.	
F2H	04H	L2_LINES_OUT.PREFETCH_CLEAN	Counts L2 clean cache line evicted by a prefetch request.	
F2H	08H	L2_LINES_OUT.PREFETCH_DIRTY	Counts L2 modified cache line evicted by a prefetch request.	
F2H	0FH	L2_LINES_OUT.ANY	Counts all L2 cache lines evicted for any reason.	
F4H	04H	SQ_MISC.LRU_HINTS	Counts number of Super Queue LRU hints sent to L3.	
F4H	10H	SQ_MISC.SPLIT_LOCK	Counts the number of SQ lock splits across a cache line.	
F6H	01H	SQ_FULL_STALL_CYCLES	Counts cycles the Super Queue is full. Neither of the threads on this core will be able to access the uncore.	
F7H	01H	FP_ASSIST.ALL	Counts the number of floating point operations executed that required micro-code assist intervention. Assists are required in the following cases: SSE instructions, (Denormal input when the DAZ flag is off or Underflow result when the FTZ flag is off): x87 instructions, (NaN or denormal are loaded to a register or used as input from memory, Division by 0 or Underflow output).	
F7H	02H	FP_ASSIST.OUTPUT	Counts number of floating point micro-code assist when the output value (destination register) is invalid.	
F7H	04H	FP_ASSIST.INPUT	Counts number of floating point micro-code assist when the input value (one of the source operands to an FP instruction) is invalid.	
FDH	01H	SIMD_INT_64.PACKED_MPY	Counts number of SID integer 64 bit packed multiply operations.	
FDH	02H	SIMD_INT_64.PACKED_SHIFT	Counts number of SID integer 64 bit packed shift operations.	
FDH	04H	SIMD_INT_64.PACK	Counts number of SID integer 64 bit pack operations.	
FDH	08H	SIMD_INT_64.UNPACK	Counts number of SID integer 64 bit unpack operations.	
FDH	10H	SIMD_INT_64.PACKED_LOGICAL	Counts number of SID integer 64 bit logical operations.	

Table 19-15 Non-Architectural Performance Events In the Processor Core for Processors Based on Intel® Microarchitecture Code Name Westmere (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
FDH	20H	SIMD_INT_64.PACKED_ARITH	Counts number of SID integer 64 bit arithmetic operations.	
FDH	40H	SIMD_INT_64.SHUFFLE_MOVE	Counts number of SID integer 64 bit shift or move operations.	

…

19.10 PERFORMANCE MONITORING EVENTS FOR PROCESSORS BASED ON THE SILVERMONT MICROARCHITECTURE

Processors based on the Silvermont microarchitecture support the architectural performance-monitoring events listed in Table 19-1 and fixed-function performance events using fixed counter. In addition, they also support the following non-architectural performance-monitoring events listed in Table 19-20. These processors have the CPUID signatures of 06_37H, 06_4AH, 06_4DH, 06_5AH, and 06_5DH.

Table 19-20 Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
03H	01H	REHABQ.LD_BLOCK_ST_FORWARD	Loads blocked due to store forward restriction	This event counts the number of retired loads that were prohibited from receiving forwarded data from the store because of address mismatch.
03H	02H	REHABQ.LD_BLOCK_STD_NOTREADY	Loads blocked due to store data not ready	This event counts the cases where a forward was technically possible, but did not occur because the store data was not available at the right time
03H	04H	REHABQ.ST_SPLITS	Store uops that split cache line boundary	This event counts the number of retire stores that experienced cache line boundary splits
03H	08H	REHABQ.LD_SPLITS	Load uops that split cache line boundary	This event counts the number of retire loads that experienced cache line boundary splits
03H	10H	REHABQ.LOCK	Uops with lock semantics	This event counts the number of retired memory operations with lock semantics. These are either implicit locked instructions such as the XCHG instruction or instructions with an explicit LOCK prefix (F0H).
03H	20H	REHABQ.STA_FULL	Store address buffer full	This event counts the number of retired stores that are delayed because there is not a store address buffer available.
03H	40H	REHABQ.ANY_LD	Any reissued load uops	This event counts the number of load uops reissued from Rehabq
03H	80H	REHABQ.ANY_ST	Any reissued store uops	This event counts the number of store uops reissued from Rehabq

Table 19-20 Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
REAHBQ is an internal queue in the Silvermont microarchitecture that holds memory reference micro-ops which cannot complete for one reason or another. The micro-ops remain in the REHABQ until they can be re-issued and successfully completed. Examples of bottlenecks that cause micro-ops to go into REHABQ include, but are not limited to: cache line splits, blocked store forward and data not ready. There are many other conditions that might cause a load or store to be sent to the REHABQ-- for instance, if an older store has an unknown address, all subsequent stores must be sent to the REHABQ until that older stores address becomes known				
04H	01H	MEM_UOPS_RETIRED.L1_MISS_LOADS	Loads retired that missed L1 data cache	This event counts the number of load ops retired that miss in L1 Data cache. Note that prefetch misses will not be counted.
04H	02H	MEM_UOPS_RETIRED.L2_HIT_LOADS	Loads retired that hit L2	This event counts the number of load micro-ops retired that hit L2.
04H	04H	MEM_UOPS_RETIRED.L2_MISS_LOADS	Loads retired that missed L2	This event counts the number of load micro-ops retired that missed L2.
04H	08H	MEM_UOPS_RETIRED.DTLB_MISS_LOADS	Loads missed DTLB	This event counts the number of load ops retired that had DTLB miss.
04H	10H	MEM_UOPS_RETIRED.UTLB_MISS	Loads missed UTLB	This event counts the number of load ops retired that had UTLB miss.
04H	20H	MEM_UOPS_RETIRED.HITM	Cross core or cross module hitm	This event counts the number of load ops retired that got data from the other core or from the other module.
04H	40H	MEM_UOPS_RETIRED.ALL_LOADS	All Loads	This event counts the number of load ops retired
04H	80H	MEM_UOP_RETIRED.ALL_STORES	All Stores	This event counts the number of store ops retired
05H	01H	PAGE_WALKS.D_SIDE_CYCLES	Duration of D-side page-walks in core cycles	This event counts every cycle when a D-side (walks due to a load) page walk is in progress. Page walk duration divided by number of page walks is the average duration of page-walks. Edge trigger bit must be cleared. Set Edge to count the number of page walks.
05H	02H	PAGE_WALKS.I_SIDE_CYCLES	Duration of I-side page-walks in core cycles	This event counts every cycle when a I-side (walks due to an instruction fetch) page walk is in progress. Page walk duration divided by number of page walks is the average duration of page-walks. Edge trigger bit must be cleared. Set Edge to count the number of page walks.
05H	03H	PAGE_WALKS.WALKS	Total number of page-walks that are completed (I-side and D-side)	This event counts when a data (D) page walk or an instruction (I) page walk is completed or started. Since a page walk implies a TLB miss, the number of TLB misses can be counted by counting the number of pagewalks. Edge trigger bit must be set. Clear Edge to count the number of cycles.
2EH	41H	LONGEST_LAT_CACHE.MISS	L2 cache request misses	This event counts the total number of L2 cache references and the number of L2 cache misses respectively. L3 is not supported in Silvermont microarchitecture.

Table 19-20 Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
2EH	4FH	LONGEST_LAT_CACHE. REFERENCE	L2 cache requests from this core	This event counts requests originating from the core that references a cache line in the L2 cache. L3 is not supported in Silvermont microarchitecture.
30H	00H	L2_REJECT_XQ.ALL	Counts the number of request from the L2 that were not accepted into the XQ	This event counts the number of demand and prefetch transactions that the L2 XQ rejects due to a full or near full condition which likely indicates back pressure from the IDI link. The XQ may reject transactions from the L2Q (non-cacheable requests), BBS (L2 misses) and WOB (L2 write-back victims)
When a memory reference misses the 1st level cache, the request goes to the L2 Queue (L2Q). If the request also misses the 2nd level cache, the request is sent to the XQ, where it waits for an opportunity to be issued to memory across the IDI link. Note that since the L2 is shared between a pair of processor cores, a single L2Q is shared between those two cores. Similarly, there is a single XQ for a pair of processors, situated between the L2Q and the IDI link. The XQ will fill up when the response rate from the IDI link is smaller than the rate at which new requests arrive at the XQ. The event L2_reject_XQ indicates that a request is unable to move from the L2 Queue to the XQ because the XQ is full, and thus indicates that the memory system is oversubscribed				
31H	00H	CORE_REJECT_L2Q.ALL	Counts the number of request that were not accepted into the L2Q because the L2Q is FULL.	This event counts the number of demand and L1 prefetcher requests rejected by the L2Q due to a full or nearly full condition which likely indicates back pressure from L2Q. It also counts requests that would have gone directly to the XQ, but are rejected due to a full or nearly full condition, indicating back pressure from the IDI link. The L2Q may also reject transactions from a core to insure fairness between cores, or to delay a core's dirty eviction when the address conflicts incoming external snoops. (Note that L2 prefetcher requests that are dropped are not counted by this event.).
The core_reject event indicates that a request from the core cannot be accepted at the L2Q. However, there are several additional reasons why a request might be rejected from the L2Q. Beyond rejecting a request because the L2Q is full, a request from one core can be rejected to maintain fairness to the other core. That is, one core is not permitted to monopolize the shared connection to the L2Q/cache/XQ/IDI links, and might have its requests rejected even when there is room available in the L2Q. In addition, if the request from the core is a dirty L1 cache eviction, the hardware must insure that this eviction does not conflict with any pending request in the L2Q. (pending requests can include an external snoop). In the event of a conflict, the dirty eviction request might be rejected even when there is room in the L2Q. Thus, while the L2_reject_XQ event indicates that the request rate to memory from both cores exceeds the response rate of the memory, the Core_reject event is more subtle. It can indicate that the request rate to the L2Q exceeds the response rate from the XQ, or it can indicate the request rate to the L2Q exceeds the response rate from the L2, or it can indicate that one core is attempting to request more than its fair share of response from the L2Q. Or, it can be an indicator of conflict between dirty evictions and other pending requests. In short, the L2_reject_XQ event indicates memory oversubscription. The Core_reject event can indicate either (1) memory oversubscription, (2) L2 oversubscription, (3) rejecting one cores requests to insure fairness to the other core, or (4) a conflict between dirty evictions and other pending requests.				
3CH	00H	CPU_CLK_UNHALTED.C ORE_P	Core cycles when core is not halted	This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. In mobile systems the core frequency may change from time to time. For this reason this event may have a changing ratio with regards to time.
N/A	01H	CPU_CLK_UNHALTED.C ORE	Instructions retired	This uses the fixed counter 1 to count the same condition as CPU_CLK_UNHALTED.CORE_P does.

Table 19-20 Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
3CH	01H	CPU_CLK_UNHALTED.REF_P	Reference cycles when core is not halted	This event counts the number of reference cycles that the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. In mobile systems the core frequency may change from time. This event is not affected by core frequency changes but counts as if the core is running at the maximum frequency all the time.
N/A	02H	CPU_CLK_UNHALTED.REF_TSC	Instructions retired	This uses the fixed counter 2 to count the same condition as CPU_CLK_UNHALTED.REF_P does.
80H	01H	ICACHE.HIT	Instruction fetches from Icache	This event counts all instruction fetches from the instruction cache.
80H	02H	ICACHE.MISSES	Icache miss	This event counts all instruction fetches that miss the Instruction cache or produce memory requests. This includes uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.
80H	03H	ICACHE.ACCESSES	Instruction fetches	This event counts all instruction fetches, including uncacheable fetches.
B7H	01H	OFFCORE_RESPONSE_0	see Section 18.6.2	Requires MSR_OFFCORE_RESP0 to specify request type and response.
B7H	02H	OFFCORE_RESPONSE_1	see Section 18.6.2	Requires MSR_OFFCORE_RESP1 to specify request type and response.
C0H	00H	INST_RETIRED.ANY_P	Instructions retired (PEBS supported with IA32_PMC0).	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.
N/A	00H	INST_RETIRED.ANY	Instructions retired	This uses the fixed counter 0 to count the same condition as INST_RETIRED.ANY_P does.
C2H	01H	UOPS_RETIRED.MS	MSROM micro-ops retired	This event counts the number of micro-ops retired that were supplied from MSROM.
C2H	10H	UOPS_RETIRED.ALL	Micro-ops retired	This event counts the number of micro-ops retired.
The processor decodes complex macro instructions into a sequence of simpler micro-ops. Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists. In some cases micro-op sequences are fused or whole instructions are fused into one micro-op. See other UOPS_RETIRED events for differentiating retired fused and non-fused micro-ops.				
C3H	01H	MACHINE_CLEARS.SMC	Self-Modifying Code detected	This event counts the number of times that a program writes to a code section. Self-modifying code causes a severe penalty in all Intel® architecture processors.
C3H	02H	MACHINE_CLEARS.MEMORY_ORDERING	Stalls due to Memory ordering	This event counts the number of times that pipeline was cleared due to memory ordering issues.
C3H	04H	MACHINE_CLEARS.FP_ASSIST	Stalls due to FP assists	This event counts the number of times that pipeline stalled due to FP operations needing assists.
C3H	08H	MACHINE_CLEARS.ALL	Stalls due to any causes	This event counts the number of times that pipeline stalled due to due to any causes (including SMC, MO, FP assist, etc).

Table 19-20 Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
There are many conditions that might cause a machine clear (including the receipt of an interrupt, or a trap or a fault). All those conditions (including but not limited to MO, SMC and FP) are captured in the ANY event. In addition, some conditions can be specifically counted (i.e. SMC, MO, FP). However, the sum of SMC, MO and FP machine clears will not necessarily equal the number of ANY. FP Assist: Most of the time, the floating point execute unit can properly produce the correct output bits. On rare occasions, it needs a little help. When help is needed, a machine clear is asserted against the instruction. After this machine clear (as described above), the front end of the machine begins to deliver instructions that will figure out exactly what FP operation was asked for, and they will do the extra work to produce the correct FP result (for instance, if the result was a floating point denormal, sometimes the hardware asks the help to produce the correctly rounded IEEE compliant result). SMC: (Self modifying code) The SMC happens when the machine fears that an instruction "in flight" is being changed. For instance, if you wrote a piece of code that wrote to the instruction stream ahead of where you were executing. In the Silvermont microarchitecture, the detection works in a 1K aligned region. If you write to memory within 1K of where you are executing, the hardware may get concerned that an instruction is being modified and a machine clear might be signaled. Since the machine clear allows the store pipeline to drain, when front end restart occurs the correct instructions (after the write) will be executed. MO: (Memory order) The MO machine clear happens when a snoop request occurs and the machine is uncertain if memory ordering will be preserved. For instance, suppose you have two loads, one to address X followed by another to address Y in the program order. Both loads have been issued; however, load to Y completes first and all the dependent ops on this load continue with the data loaded by this load. Load to X is still waiting for the data. Suppose that at the same time another processor writes to the same address Y and causes a snoop to address Y. This presents a problem: the load to Y got the old value, but we have not yet finished loading X. So the other processor saw the loads in a different order by not consuming the latest value from the store to address Y. So we need to un-do everything from the load to address Y so that we will see the post-write data. Note we do not have to un-do load Y if there were no other pending reads-- the fact that the load to X is not yet finished causes this ordering problem.				
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Retired branch instructions	This event counts the number of branch instructions retired.
C4H	7EH	BR_INST_RETIRED.JCC	Retired branch instructions that were conditional jumps	This event counts the number of branch instructions retired that were conditional jumps.
C4H	BFH	BR_INST_RETIRED.FAR_BRANCH	Retired far branch instructions	This event counts the number of far branch instructions retired.
C4H	EBH	BR_INST_RETIRED.NON_RETURN_IND	Retired instructions of near indirect Jmp or call	This event counts the number of branch instructions retired that were near indirect call or near indirect jmp.
C4H	F7H	BR_INST_RETIRED.RETURN	Retired near return instructions	This event counts the number of near RET branch instructions retired
C4H	F9H	BR_INST_RETIRED.CALL	Retired near call instructions	This event counts the number of near CALL branch instructions retired
C4H	FBH	BR_INST_RETIRED.IND_CALL	Retired near indirect call instructions	This event counts the number of near indirect CALL branch instructions retired
C4H	FDH	BR_INST_RETIRED.REL_CALL	Retired near relative call instructions	This event counts the number of near relative CALL branch instructions retired
C4H	FEH	BR_INST_RETIRED.TAKEN_JCC	Retired conditional jumps that were predicted taken	This event counts the number of branch instructions retired that were conditional jumps and predicted taken.
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Retired mispredicted branch instructions	This event counts the number of mispredicted branch instructions retired.

Table 19-20 Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
C5H	7EH	BR_MISP_RETIRED.JCC	Retired mispredicted conditional jumps	This event counts the number of mispredicted branch instructions retired that were conditional jumps.
C5H	BFH	BR_MISP_RETIRED.FAR	Retired mispredicted far branch instructions	This event counts the number of mispredicted far branch instructions retired.
C5H	EBH	BR_MISP_RETIRED.NON_RETURN_IND	Retired mispredicted instructions of near indirect Jmp or call	This event counts the number of mispredicted branch instructions retired that were near indirect call or near indirect jmp.
C5H	F7H	BR_MISP_RETIRED.RETURN	Retired mispredicted near return instructions	This event counts the number of mispredicted near RET branch instructions retired
C5H	F9H	BR_MISP_RETIRED.CALL	Retired mispredicted near call instructions	This event counts the number of mispredicted near CALL branch instructions retired
C5H	FBH	BR_MISP_RETIRED.IND_CALL	Retired mispredicted near indirect call instructions	This event counts the number of mispredicted near indirect CALL branch instructions retired
C5H	FDH	BR_MISP_RETIRED.REL_CALL	Retired mispredicted near relative call instructions	This event counts the number of mispredicted near relative CALL branch instructions retired
C5H	FEH	BR_MISP_RETIRED.TAKEN_JCC	Retired mispredicted conditional jumps that were predicted taken	This event counts the number of mispredicted branch instructions retired that were conditional jumps and predicted taken.
CAH	01H	NO_ALLOC_CYCLES.ROB_FULL	Counts the number of cycles when no uops are allocated and the ROB is full (less than 2 entries available)	Counts the number of cycles when no uops are allocated and the ROB is full (less than 2 entries available)
CAH	20H	NO_ALLOC_CYCLES.RAT_STALL	Counts the number of cycles when no uops are allocated and a RATstall is asserted.	Counts the number of cycles when no uops are allocated and a RATstall is asserted.
CAH	3FH	NO_ALLOC_CYCLES.ALL	Front end not delivering	This event counts the number of cycles when the front-end does not provide any instructions to be allocated for any reason
CAH	50H	NO_ALLOC_CYCLES.NOT_DELIVERED	Front end not delivering backend not stalled	This event counts the number of cycles when the front-end does not provide any instructions to be allocated but the back end is not stalled
The front-end is responsible for fetching the instruction, decoding into micro-ops (uops) and putting them into a micro-op queue to be consumed by back end. The back-end then takes these micro-ops, allocates the required resources. When all resources are ready, micro-ops are executed. If the back-end is not ready to accept micro-ops from the front-end, then we do not want to count these as front-end bottlenecks. However, whenever we have bottlenecks in the back-end, we will have allocation unit stalls and eventually forcing the front-end to wait until the back-end is ready to receive more UOPS. This event counts the cycles only when back-end is requesting more micro-uops and front-end is not able to provide them.				
CBH	01H	RS_FULL_STALL.MEC	MEC RS full	This event counts the number of cycles the allocation pipe line stalled due to the RS for the MEC cluster is full
CBH	1FH	RS_FULL_STALL.ALL	Any RS full	This event counts the number of cycles that the allocation pipe line stalled due to any one of the RS is full

Table 19-20 Performance Events for Silvermont Microarchitecture

Event Num.	Umask Value	Event Name	Definition	Description and Comment
The Silvermont microarchitecture has an allocation pipeline (AKA the RAT) that moves UOPS from the front end to the backend. At the end of the allocate pipe a UOP needs to be written into one of 6 reservation stations (the RS). Each RS holds UOPS that are to be sent to a specific execution (or memory) cluster. Each RS has a finite capacity, and it may accumulate UOPS when it is unable to send a UOP to its execution cluster. Typical reasons why an RS may fill include, but are not limited to, execution of long latency UOPS like divide, or inability to schedule UOPS due to dependencies, or too many outstanding memory references. When the RS becomes full, it is unable to accept more UOPS, and it will stall the allocation pipeline. The RS_FULL_STALL.ANY event will be asserted on any cycle when the allocation is stalled for any one of the RSs being full and not for other reasons. (i.e. the allocate pipeline might be stalled for some other reason, but if RS is not full, the RS_FULL_STALL.ANY will not count) The subevents allow discovery of exactly which RS (or RSs) that are full that prevent further allocation.				
CDH	01H	CYCLES_DIV_BUSY.ANY	Divider Busy	This event counts the number of cycles the divider is busy.
This event counts the cycles when the divide unit is unable to accept a new divide UOP because it is busy processing a previously dispatched UOP. The cycles will be counted irrespective of whether or not another divide UOP is waiting to enter the divide unit (from the RS). This event will count cycles while a divide is in progress even if the RS is empty.				
E6H	01H	BACLEARS.ALL	BACLEARS asserted for any branch	This event counts the number of baclears for any type of branch.
E6H	08H	BACLEARS.RETURN	BACLEARS asserted for return branch	This event counts the number of baclears for return branches.
E6H	10H	BACLEARS.COND	BACLEARS asserted for conditional branch	This event counts the number of baclears for conditional branches.
E7H	01H	MS_DECODED.MS_ENTRY	MS Decode starts	This event counts the number of times the MSROM starts a flow of UOPS.

...

19.10.1 Performance Monitoring Events for Processors Based on the Airmont Microarchitecture

Intel processors based on the Airmont microarchitecture support the same architectural and the non-architectural performance monitoring events as processors based on the Silvermont microarchitecture. All of the events listed in Table 19-20 apply. These processors have the CPUID signatures that include 06_4CH.

...

19. Updates to Chapter 22, Volume 3B

Change bars show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

22.18.6.3 Numeric Underflow Exception (#U)

When the underflow exception is masked on the 32-bit x87 FPUs, the underflow exception is signaled when the result is tiny and inexact (see Section 4.9.1.5, "Numeric Underflow Exception (#U)" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*). When the underflow exception is unmasked and the instruction

is supposed to store the result on the stack, the significand is rounded to the appropriate precision (according to the PC flag in the FPU control word, for those instructions controlled by PC, otherwise to extended precision), after adjusting the exponent.

...

22.18.7.6 FPTAN Instruction

On the 32-bit x87 FPUs, the range of the operand for the FPTAN instruction is much less restricted (| ST(0) | < 2^{63}) than on earlier math coprocessors. The instruction reduces the operand internally using an internal $\pi/4$ constant that is more accurate. The range of the operand is restricted to (| ST(0) | < $\pi/4$) on the 16-bit IA-32 math coprocessors; the operand must be reduced to this range using FPREM. This change has no impact on existing software. See also sections 8.3.8 and section 8.3.10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for more information on the accuracy of the FPTAN instruction.

...

22.18.7.8 FSIN, FCOS, and FSINCOS Instructions

On the 32-bit x87 FPUs, these instructions perform three common trigonometric functions. These instructions do not exist on the 16-bit IA-32 math coprocessors. The availability of these instructions has no impact on existing software, but using them provides a performance upgrade. See also sections 8.3.8 and section 8.3.10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for more information on the accuracy of the FSIN, FCOS, and FSINCOS instructions.

...

20. Updates to Chapter 29, Volume 3B

Change bars show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

29.5 VIRTUALIZING MSR-BASED APIC ACCESSES

When the local APIC is in x2APIC mode, software accesses the local APIC's control registers using the MSR interface. Specifically, software uses the RDMSR and WRMSR instructions, setting ECX (identifying the MSR being accessed) to values in the range 800H–8FFH (see Section 10.12, "Extended XAPIC (x2APIC)"). This section describes how these accesses can be virtualized.

A virtual-machine monitor can virtualize these MSR-based APIC accesses by configuring the MSR bitmaps (see Section 24.6.9) to ensure that the accesses cause VM exits (see Section 25.1.3). Alternatively, there are methods for virtualizing some MSR-based APIC accesses without VM exits.

Normally, an execution of RDMSR or WRMSR that does not fault or cause a VM exit accesses the MSR indicated in ECX. However, such an execution treats some values of ECX in the range 800H–8FFH specially if the "virtualize x2APIC mode" VM-execution control is 1. The following items provide details:

- **RDMSR.** The instruction's behavior depends on the setting of the "APIC-register virtualization" VM-execution control.
  - — If the "APIC-register virtualization" VM-execution control is 0, behavior depends upon the value of ECX.

- If ECX contains 808H (indicating the TPR MSR), the instruction reads the 8 bytes from offset 080H on the virtual-APIC page (VTPR and the 4 bytes above it) into EDX:EAX. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode).
- If ECX contains any other value in the range 800H–8FFH, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a readable APIC register, EDX and EAX are loaded with the value of that register. If the local APIC is not in x2APIC mode or ECX does not indicate a readable APIC register, a general-protection fault occurs.

— If "APIC-register virtualization" is 1 and ECX contains a value in the range 800H–8FFH, the instruction reads the 8 bytes from offset X on the virtual-APIC page into EDX:EAX, where X = (ECX & FFH) « 4. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not in x2APIC mode).

- **WRMSR.** The instruction's behavior depends on the value of ECX and the setting of the "virtual-interrupt delivery" VM-execution control.

  Special processing applies in the following cases: (1) ECX contains 808H (indicating the TPR MSR); (2) ECX contains 80BH (indicating the EOI MSR) and the "virtual-interrupt delivery" VM-execution control is 1; and (3) ECX contains 83FH (indicating the self-IPI MSR) and the "virtual-interrupt delivery" VM-execution control is 1.

  If special processing applies, no general-protection exception is produced due to the fact that the local APIC is in xAPIC mode. However, WRMSR does perform the normal reserved-bit checking:

  — If ECX contains 808H or 83FH, a general-protection fault occurs if either EDX or EAX[31:8] is non-zero.

  — If ECX contains 80BH, a general-protection fault occurs if either EDX or EAX is non-zero.

  If there is no fault, WRMSR stores EDX:EAX at offset X on the virtual-APIC page, where X = (ECX & FFH) « 4. Following this, the processor performs an operation depending on the value of ECX:

  — If ECX contains 808H, the processor performs TPR virtualization (see Section 29.1.2).

  — If ECX contains 80BH, the processor performs EOI virtualization (see Section 29.1.4).

  — If ECX contains 83FH, the processor then checks the value of EAX[7:4] and proceeds as follows:

    - If the value is non-zero, the logical processor performs self-IPI virtualization with the 8-bit vector in EAX[7:0] (see Section 29.1.5).
    - If the value is zero, the logical processor causes an APIC-write VM exit as if there had been a write access to page offset 3F0H on the APIC-access page (see Section 29.4.3.3).

  If special processing does not apply, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a writeable APIC register, the value in EDX:EAX is written to that register. If the local APIC is not in x2APIC mode or ECX does not indicate a writeable APIC register, a general-protection fault occurs.

...

21. Updates to Chapter 33, Volume 3C

Change bars show changes to Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

33.3.3.4 Generation of Virtual Interrupt Events by VMM

The following provides some of the general steps that need to be taken by VMM designs when generating virtual interrupts:

1. Check virtual processor interruptibility state. The virtual processor interruptibility state is reflected in the guest RFLAGS.IF flag and the processor interruptibility-state saved in the guest state area of the controlling-VMCS. If RFLAGS.IF is set and the interruptibility state indicates readiness to take external interrupts (STI-masking and MOV-SS/POP-SS-masking bits are clear), the guest virtual processor is ready to take external interrupts. If the VMM design supports non-active guest sleep states, the VMM needs to make sure the current guest sleep state allows injection of external interrupt events.
2. If the guest virtual processor state is currently not interruptible, a VMM may utilize the "interrupt-window exiting" VM-execution to notify the VMM (through a VM exit) when the virtual processor state changes to interruptible state.
3. Check the virtual interrupt controller state. If the guest VM exposes a virtual local APIC, the current value of its processor priority register specifies if guest software allows dispensing an external virtual interrupt with a specific priority to the virtual processor. If the virtual interrupt is routed through the local vector table (LVT) entry of the local APIC, the mask bits in the corresponding LVT entry specifies if the interrupt is currently masked. Similarly, the virtual interrupt controller's current mask (IO-APIC or PIC) and priority settings reflect guest state to accept specific external interrupts. The VMM needs to check both the virtual processor and interrupt controller states to verify its guest interruptibility state. If the guest is currently interruptible, the VMM can inject the virtual interrupt. If the current guest state does not allow injecting a virtual interrupt, the interrupt needs to be queued by the VMM until it can be delivered.
4. Prioritize the use of VM-entry event injection. A VMM may use VM-entry event injection to deliver various virtual events (such as external interrupts, exceptions, traps, and so forth). VMM designs may prioritize use of virtual-interrupt injection between these event types. Since each VM entry allows injection of one event, depending on the VMM event priority policies, the VMM may need to queue the external virtual interrupt if a higher priority event is to be delivered on the next VM entry. Since the VMM has masked this particular interrupt source (if it was level triggered) and done EOI to the platform interrupt controller, other platform interrupts can be serviced while this virtual interrupt event is queued for later delivery to the VM.
5. Update the virtual interrupt controller state. When the above checks have passed, before generating the virtual interrupt to the guest, the VMM updates the virtual interrupt controller state (Local-APIC, IO-APIC and/or PIC) to reflect assertion of the virtual interrupt. This involves updating the various interrupt capture registers, and priority registers as done by the respective hardware interrupt controllers. Updating the virtual interrupt controller state is required for proper interrupt event processing by guest software.
6. Inject the virtual interrupt on VM entry. To inject an external virtual interrupt to a guest VM, the VMM sets up the VM-entry interruption-information field in the guest controlling-VMCS before entry to guest using VMRESUME. Upon VM entry, the processor will use this vector to access the gate in guest's IDT and the value of RFLAGS and EIP in guest-state area of controlling-VMCS is pushed on the guest stack. If the guest RFLAGS.IF is clear, the STI-masking bit is set, or the MOV- SS/POP-SS-masking bit is set, the VM entry will fail and the processor will load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 26.7).

...

22. 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 list 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	Future Generation Intel Core Processor
06_56H	Next Generation Intel Xeon Processor D Product Family based on Broadwell microarchitecture
06_4FH	Future Generation Intel Xeon processor based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, Future 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-2600/1600 v3 product families based on Haswell-E microarchitecture, Intel Core i7-59xx Processor Extreme Edition
06_3CH, 06_45H, 06_46H	4th Generation Intel Core processor and Intel Xeon processor E3-1200 v3 product family based on Haswell microarchitecture
06_3EH	Intel Xeon processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture
06_3EH	Intel Xeon processor E5-2600/1600 v2 product families and Intel Xeon processor E5-2400 v2 product family based on Ivy Bridge-E microarchitecture, Intel Core i7-49xx Processor Extreme Edition
06_3AH	3rd Generation Intel Core Processor and Intel Xeon processor E3-1200 v2 product family based on Ivy Bridge microarchitecture
06_2DH	Intel Xeon processor E5 Family based on 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 Z8000 series
06_5DH	Future Intel Atom Processor Based on Silvermont Microarchitecture

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

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
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 "MAXPHYWID" in Table 35-2. "MAXPHYWID" 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 17.13, "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.	
		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)	
		(MAXPHYWID - 1):12	APIC Base (R/W)	
		63: MAXPHYWID	Reserved	
3AH	58	IA32_FEATURE_CONTROL	**Control Features in Intel 64 Processor (R/W)**	If CPUID.01H: ECX[bit 5 or bit 6] = 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	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	If CPUID.01H:ECX[bit 5 or bit 6] = 1
			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.	
		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[bit 5 and bit 6] are set to 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[bit 5 or bit 6] = 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[bit 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[bit 6] = 1
		19:16	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.	
		63:21	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			
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H): EBX[1] = 1
		63:0	**THREAD_ADJUST:** Local offset value of the IA32_TSC for a logical processor. Reset value is Zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
79H	121	IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)	BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 9.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
8BH	139	IA32_BIOS_SIGN_ID (BIOS_SIGN/ BBL_CR_D3)	BIOS Update Signature (RO) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
		31:0	Reserved	
		63:32	It is recommended that this field be pre-loaded with 0 prior to executing CPUID. If the field remains 0 following the execution of CPUID; this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature.	
9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	If CPUID.01H: ECX[bit 5 or bit 6] = 1
		0	Valid (R/W)	
		1	Reserved	
		2	Controls SMI unblocking by VMXOFF (see Section 34.14.4)	If IA32_VMX_MISC[bit 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[bit 15])

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			
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
C3H	195	IA32_PMC2	General Performance Counter 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
C4H	196	IA32_PMC3	General Performance Counter 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
C5H	197	IA32_PMC4	General Performance Counter 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
C6H	198	IA32_PMC5	General Performance Counter 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5
C7H	199	IA32_PMC6	General Performance Counter 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
C8H	200	IA32_PMC7	General Performance Counter 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
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.	

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	SMRR Supported when set.	
		63:12	Reserved.	
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

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			
		2	MCIP. Machine check in progress	06_01H
		3	LMCE_S.	If IA32_MCG_CAP.LMCE_P =1
		63:4	Reserved.	
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	06_01H
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 and 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 3BFH) on a PMI request	If CPUID.01H: ECX[15] = 1 and 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[bit 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] = 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)	06_0FH
1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	
1FAH	506	IA32_DCA_0_CAP	DCA type 0 Status and Control register.	06_2EH
		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."	06_01H
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	06_01H
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	06_01H
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	06_01H
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	06_01H
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	06_01H
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	06_01H
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	06_01H
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	06_01H
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	06_01H
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	06_01H
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	06_01H
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	06_01H
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	06_01H
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	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			
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	06_01H
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	06_01H
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	06_01H
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	06_01H
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	See Section 11.11.2.2, "Fixed Range MTRRs."	06_01H
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	06_01H
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	06_01H
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	06_01H
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	06_01H
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	06_01H
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	06_01H
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	06_01H
277H	631	IA32_PAT	IA32_PAT (R/W)	06_05H
		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.	

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			
280H	640	IA32_MC0_CTL2	(R/W)	06_1AH
		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.	06_1AH
282H	642	IA32_MC2_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
283H	643	IA32_MC3_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
284H	644	IA32_MC4_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
285H	645	IA32_MC5_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
286H	646	IA32_MC6_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
287H	647	IA32_MC7_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
288H	648	IA32_MC8_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_1AH
289H	649	IA32_MC9_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28AH	650	IA32_MC10_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28BH	651	IA32_MC11_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28CH	652	IA32_MC12_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28DH	653	IA32_MC13_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28EH	654	IA32_MC14_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
28FH	655	IA32_MC15_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
290H	656	IA32_MC16_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
291H	657	IA32_MC17_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
292H	658	IA32_MC18_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
293H	659	IA32_MC19_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
294H	660	IA32_MC20_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
295H	661	IA32_MC21_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_2EH
296H	662	IA32_MC22_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
297H	663	IA32_MC23_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
298H	664	IA32_MC24_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
299H	665	IA32_MC25_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
29AH	666	IA32_MC26_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
29BH	667	IA32_MC27_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
29CH	668	IA32_MC28_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
29DH	669	IA32_MC29_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH

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			
29EH	670	IA32_MC30_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
29FH	671	IA32_MC31_CTL2	(R/W) same fields as IA32_MC0_CTL2.	06_3EH
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType (R/W)	06_01H
		2:0	Default Memory Type	
		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 (MSR_PERF_FIXED_CTR_CTRL)	Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.	If CPUID.0AH: EAX[7:0] > 1
		0	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0.	
		1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	
		4	EN1_OS: Enable Fixed Counter 1 to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1 to count while CPL > 0.	
		6	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 (MSR_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

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			
		2	Ovf_PMC2: Overflow status of IA32_PMC2.	06_2EH
		3	Ovf_PMC3: Overflow status of IA32_PMC3.	06_2EH
		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
		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 IA32_RTIT_CTL.ToPA = 1
		60:56	Reserved.	
		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 (MSR_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[7:0] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved.	
		32	EN_FIXED_CTR0	If CPUID.0AH: EAX[7:0] > 1
		33	EN_FIXED_CTR1	If CPUID.0AH: EAX[7:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EAX[7:0] > 1
		63:35	Reserved.	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL (MSR_PERF_GLOBAL_OVF_CTRL)	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[7:0] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[7:0] > 0
		31:2	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: EAX[7:0] > 1
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EAX[7:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EAX[7:0] > 1
		60:35	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
3F1H	1009	IA32_PEBS_ENABLE	PEBS Control (R/W)	
		0	Enable PEBS on IA32_PMC0.	06_0FH
		1-3	Reserved or Model specific.	
		31:4	Reserved.	
		35-32	Reserved or Model specific.	
		63:36	Reserved.	
400H	1024	IA32_MC0_CTL	MC0_CTL	06_01H
401H	1025	IA32_MC0_STATUS	MC0_STATUS	06_01H
402H	1026	IA32_MC0_ADDR[1]	MC0_ADDR	06_01H
403H	1027	IA32_MC0_MISC	MC0_MISC	06_01H
404H	1028	IA32_MC1_CTL	MC1_CTL	06_01H
405H	1029	IA32_MC1_STATUS	MC1_STATUS	06_01H
406H	1030	IA32_MC1_ADDR[2]	MC1_ADDR	06_01H
407H	1031	IA32_MC1_MISC	MC1_MISC	06_01H
408H	1032	IA32_MC2_CTL	MC2_CTL	06_01H
409H	1033	IA32_MC2_STATUS	MC2_STATUS	06_01H
40AH	1034	IA32_MC2_ADDR[1]	MC2_ADDR	06_01H
40BH	1035	IA32_MC2_MISC	MC2_MISC	06_01H
40CH	1036	IA32_MC3_CTL	MC3_CTL	06_01H
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	06_01H
40EH	1038	IA32_MC3_ADDR[1]	MC3_ADDR	06_01H
40FH	1039	IA32_MC3_MISC	MC3_MISC	06_01H
410H	1040	IA32_MC4_CTL	MC4_CTL	06_01H
411H	1041	IA32_MC4_STATUS	MC4_STATUS	06_01H
412H	1042	IA32_MC4_ADDR[1]	MC4_ADDR	06_01H
413H	1043	IA32_MC4_MISC	MC4_MISC	06_01H
414H	1044	IA32_MC5_CTL	MC5_CTL	06_0FH

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			
415H	1045	IA32_MC5_STATUS	MC5_STATUS	06_0FH
416H	1046	IA32_MC5_ADDR[1]	MC5_ADDR	06_0FH
417H	1047	IA32_MC5_MISC	MC5_MISC	06_0FH
418H	1048	IA32_MC6_CTL	MC6_CTL	06_1DH
419H	1049	IA32_MC6_STATUS	MC6_STATUS	06_1DH
41AH	1050	IA32_MC6_ADDR[1]	MC6_ADDR	06_1DH
41BH	1051	IA32_MC6_MISC	MC6_MISC	06_1DH
41CH	1052	IA32_MC7_CTL	MC7_CTL	06_1AH
41DH	1053	IA32_MC7_STATUS	MC7_STATUS	06_1AH
41EH	1054	IA32_MC7_ADDR[1]	MC7_ADDR	06_1AH
41FH	1055	IA32_MC7_MISC	MC7_MISC	06_1AH
420H	1056	IA32_MC8_CTL	MC8_CTL	06_1AH
421H	1057	IA32_MC8_STATUS	MC8_STATUS	06_1AH
422H	1058	IA32_MC8_ADDR[1]	MC8_ADDR	06_1AH
423H	1059	IA32_MC8_MISC	MC8_MISC	06_1AH
424H	1060	IA32_MC9_CTL	MC9_CTL	06_2EH
425H	1061	IA32_MC9_STATUS	MC9_STATUS	06_2EH
426H	1062	IA32_MC9_ADDR[1]	MC9_ADDR	06_2EH
427H	1063	IA32_MC9_MISC	MC9_MISC	06_2EH
428H	1064	IA32_MC10_CTL	MC10_CTL	06_2EH
429H	1065	IA32_MC10_STATUS	MC10_STATUS	06_2EH
42AH	1066	IA32_MC10_ADDR[1]	MC10_ADDR	06_2EH
42BH	1067	IA32_MC10_MISC	MC10_MISC	06_2EH
42CH	1068	IA32_MC11_CTL	MC11_CTL	06_2EH
42DH	1069	IA32_MC11_STATUS	MC11_STATUS	06_2EH
42EH	1070	IA32_MC11_ADDR[1]	MC11_ADDR	06_2EH
42FH	1071	IA32_MC11_MISC	MC11_MISC	06_2EH
430H	1072	IA32_MC12_CTL	MC12_CTL	06_2EH
431H	1073	IA32_MC12_STATUS	MC12_STATUS	06_2EH
432H	1074	IA32_MC12_ADDR[1]	MC12_ADDR	06_2EH
433H	1075	IA32_MC12_MISC	MC12_MISC	06_2EH
434H	1076	IA32_MC13_CTL	MC13_CTL	06_2EH
435H	1077	IA32_MC13_STATUS	MC13_STATUS	06_2EH
436H	1078	IA32_MC13_ADDR[1]	MC13_ADDR	06_2EH

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			
437H	1079	IA32_MC13_MISC	MC13_MISC	06_2EH
438H	1080	IA32_MC14_CTL	MC14_CTL	06_2EH
439H	1081	IA32_MC14_STATUS	MC14_STATUS	06_2EH
43AH	1082	IA32_MC14_ADDR[1]	MC14_ADDR	06_2EH
43BH	1083	IA32_MC14_MISC	MC14_MISC	06_2EH
43CH	1084	IA32_MC15_CTL	MC15_CTL	06_2EH
43DH	1085	IA32_MC15_STATUS	MC15_STATUS	06_2EH
43EH	1086	IA32_MC15_ADDR[1]	MC15_ADDR	06_2EH
43FH	1087	IA32_MC15_MISC	MC15_MISC	06_2EH
440H	1088	IA32_MC16_CTL	MC16_CTL	06_2EH
441H	1089	IA32_MC16_STATUS	MC16_STATUS	06_2EH
442H	1090	IA32_MC16_ADDR[1]	MC16_ADDR	06_2EH
443H	1091	IA32_MC16_MISC	MC16_MISC	06_2EH
444H	1092	IA32_MC17_CTL	MC17_CTL	06_2EH
445H	1093	IA32_MC17_STATUS	MC17_STATUS	06_2EH
446H	1094	IA32_MC17_ADDR[1]	MC17_ADDR	06_2EH
447H	1095	IA32_MC17_MISC	MC17_MISC	06_2EH
448H	1096	IA32_MC18_CTL	MC18_CTL	06_2EH
449H	1097	IA32_MC18_STATUS	MC18_STATUS	06_2EH
44AH	1098	IA32_MC18_ADDR[1]	MC18_ADDR	06_2EH
44BH	1099	IA32_MC18_MISC	MC18_MISC	06_2EH
44CH	1100	IA32_MC19_CTL	MC19_CTL	06_2EH
44DH	1101	IA32_MC19_STATUS	MC19_STATUS	06_2EH
44EH	1102	IA32_MC19_ADDR[1]	MC19_ADDR	06_2EH
44FH	1103	IA32_MC19_MISC	MC19_MISC	06_2EH
450H	1104	IA32_MC20_CTL	MC20_CTL	06_2EH
451H	1105	IA32_MC20_STATUS	MC20_STATUS	06_2EH
452H	1106	IA32_MC20_ADDR[1]	MC20_ADDR	06_2EH
453H	1107	IA32_MC20_MISC	MC20_MISC	06_2EH
454H	1108	IA32_MC21_CTL	MC21_CTL	06_2EH
455H	1109	IA32_MC21_STATUS	MC21_STATUS	06_2EH
456H	1110	IA32_MC21_ADDR[1]	MC21_ADDR	06_2EH
457H	1111	IA32_MC21_MISC	MC21_MISC	06_2EH

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			
480H	1152	IA32_VMX_BASIC	**Reporting Register of Basic VMX Capabilities (R/O)** See Appendix A.1, "Basic VMX Information."	If CPUID.01H:ECX.[bit 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.[bit 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.[bit 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.[bit 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.[bit 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.[bit 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.[bit 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.[bit 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.[bit 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.[bit 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.[bit 5] = 1
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.[bit 5] and IA32_VMX_PROCBASED_CTLS[bit 63])

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			
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.[bit 5], IA32_VMX_PROCBASED_CTLS[bit 63], and either IA32_VMX_PROCBASED_CTLS2[bit 33] or IA32_VMX_PROCBASED_CTLS2[bit 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.[bit 5] = 1 and IA32_VMX_BASIC[bit 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.[bit 5] = 1 and IA32_VMX_BASIC[bit 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.[bit 5] = 1 and IA32_VMX_BASIC[bit 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.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])
491H	1169	IA32_VMX_VMFUNC	**Capability Reporting Register of VM-function Controls (R/O)**	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 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
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

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			
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.	
560H	1376	IA32_RTIT_OUTPUT_BASE	**Trace Output Base Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1)
		6:0	Reserved	
		MAXPHYADDR[3]-1:7	Base physical address of the current ToPA table.	
		63:MAXPHYADDR	**Reserved.**	
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	**Trace Output Mask Pointers Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1)
		6:0	Reserved	
		31:7	MaskOrTableOffset	
		63:32	**Output Offset.**	
570H	1392	IA32_RTIT_CTL	**Trace Packet Control Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1)
		0	**TraceEn**	
		1	Reserved,	
		2	**OS**	
		3	**User**	
		6:4	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			
		7	**CR3 filter**	
		8	**ToPA**	
		9	Reserved,	
		10	**TSCEn**	
		11	**DisRETC**	
		12	Reserved,	
		13	**BranchEn**	
		63:14	Reserved, MBZ.	
571H	1393	IA32_RTIT_STATUS	**Tracing Status Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1)
		0	Reserved,	
		1	**ContexEn**, (writes ignored)	
		2	**TriggerEn**, (writes ignored)	
		3	Reserved	
		4	**Error**	
		5	**Stopped**	
		63:6	**Reserved.**	
572H	1394	IA32_RTIT_CR3_MATCH	**Trace Filter CR3 Match Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1)
		4:0	Reserved	
		63:5	CR3[63:5] value to match	
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."	0F_0H
		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.[bit 24] = 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			
770H	1904	IA32_PM_ENABLE	**Enable/disable HWP (R/W)**	If(CPUID.06H:EAX.[bit 7] = 1
		0	**HWP_ENABLE (R/W1-Once).** See Section 14.4.2, "Enabling HWP"	If(CPUID.06H:EAX.[bit 7] = 1
		63:1	Reserved.	
771H	1905	IA32_HWP_CAPABILITIES	**HWP Performance Range Enumeration (RO)**	If(CPUID.06H:EAX.[bit 7] = 1
		7:0	**Highest_Performance** See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If(CPUID.06H:EAX.[bit 7] = 1
		15:8	**Guaranteed_Performance** See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If(CPUID.06H:EAX.[bit 7] = 1
		23:16	**Most_Efficient_Performance** See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If(CPUID.06H:EAX.[bit 7] = 1
		31:24	**Lowest_Performance** See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"	If(CPUID.06H:EAX.[bit 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.[bit 11] = 1
		7:0	**Minimum_Performance** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 11] = 1
		15:8	**Maximum_Performance** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 11] = 1
		23:16	**Desired_Performance** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 11] = 1
		31:24	**Energy_Performance_Preference** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 11] = 1 and CPUID.06HEAX.[bit 10] = 1
		41:32	**Activity_Window** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 11] = 1 and CPUID.06HEAX.[bit 9] = 1
		63:42	Reserved.	
773H	1907	IA32_HWP_INTERRUPT	**Control HWP Native Interrupts (R/W)**	If(CPUID.06H:EAX.[bit 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			
		0	**EN_Guaranteed_Performance_Change.** See Section 14.4.6, "HWP Notifications"	If(CPUID.06H:EAX.[bit 8] = 1
		1	**EN_Excursion_Minimum.** See Section 14.4.6, "HWP Notifications"	If(CPUID.06H:EAX.[bit 8] = 1
		63:2	Reserved.	
774H	1908	IA32_HWP_REQUEST	**Power Management Control Hints to a Logical Processor (R/W)**	If(CPUID.06H:EAX.[bit 7] = 1
		7:0	**Minimum_Performance** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 7] = 1
		15:8	**Maximum_Performance** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 7] = 1
		23:16	**Desired_Performance** See Section 14.4.4, "Managing HWP"	If(CPUID.06H:EAX.[bit 7] = 1
		31:24	**Energy_Performance_Preference** See Section 14.4.4, "Managing HWP"	If CPUID.06HEAX.[bit 7] = 1 and (CPUID.06H:EAX.[bit 10] = 1
		41:32	**Activity_Window** See Section 14.4.4, "Managing HWP"	If CPUID.06HEAX.[bit 7] = 1 and (CPUID.06H:EAX.[bit 9] = 1
		42	**Package_Control** See Section 14.4.4, "Managing HWP"	IfCPUID.06HEAX.[bit 7] = 1 and (CPUID.06H:EAX.[bit 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.[bit 7] = 1
		0	**Guaranteed_Performance_Change (R/WC0).** See Section 14.4.5, "HWP Feedback"	If(CPUID.06H:EAX.[bit 7] = 1
		1	Reserved.	
		2	**Excursion_To_Minimum (R/WC0).** See Section 14.4.5, "HWP Feedback"	If(CPUID.06H:EAX.[bit 7] = 1
		63:3	Reserved.	
802H	2050	IA32_X2APIC_APICID	**x2APIC ID Register (R/O)** See x2APIC Specification	If (CPUID.01H:ECX.[bit 21] = 1)
803H	2051	IA32_X2APIC_VERSION	**x2APIC Version Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
808H	2056	IA32_X2APIC_TPR	**x2APIC Task Priority Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 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			
80AH	2058	IA32_X2APIC_PPR	**x2APIC Processor Priority Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80BH	2059	IA32_X2APIC_EOI	**x2APIC EOI Register (W/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80DH	2061	IA32_X2APIC_LDR	**x2APIC Logical Destination Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80FH	2063	IA32_X2APIC_SIVR	**x2APIC Spurious Interrupt Vector Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
810H	2064	IA32_X2APIC_ISR0	**x2APIC In-Service Register Bits 31:0 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
811H	2065	IA32_X2APIC_ISR1	**x2APIC In-Service Register Bits 63:32 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
812H	2066	IA32_X2APIC_ISR2	**x2APIC In-Service Register Bits 95:64 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
813H	2067	IA32_X2APIC_ISR3	**x2APIC In-Service Register Bits 127:96 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
814H	2068	IA32_X2APIC_ISR4	**x2APIC In-Service Register Bits 159:128 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
815H	2069	IA32_X2APIC_ISR5	**x2APIC In-Service Register Bits 191:160 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
816H	2070	IA32_X2APIC_ISR6	**x2APIC In-Service Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
817H	2071	IA32_X2APIC_ISR7	**x2APIC In-Service Register Bits 255:224 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
818H	2072	IA32_X2APIC_TMR0	**x2APIC Trigger Mode Register Bits 31:0 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
819H	2073	IA32_X2APIC_TMR1	**x2APIC Trigger Mode Register Bits 63:32 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81AH	2074	IA32_X2APIC_TMR2	**x2APIC Trigger Mode Register Bits 95:64 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81BH	2075	IA32_X2APIC_TMR3	**x2APIC Trigger Mode Register Bits 127:96 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81CH	2076	IA32_X2APIC_TMR4	**x2APIC Trigger Mode Register Bits 159:128 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81DH	2077	IA32_X2APIC_TMR5	**x2APIC Trigger Mode Register Bits 191:160 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81EH	2078	IA32_X2APIC_TMR6	**x2APIC Trigger Mode Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81FH	2079	IA32_X2APIC_TMR7	**x2APIC Trigger Mode Register Bits 255:224 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 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			
820H	2080	IA32_X2APIC_IRR0	**x2APIC Interrupt Request Register Bits 31:0 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
821H	2081	IA32_X2APIC_IRR1	**x2APIC Interrupt Request Register Bits 63:32 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
822H	2082	IA32_X2APIC_IRR2	**x2APIC Interrupt Request Register Bits 95:64 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
823H	2083	IA32_X2APIC_IRR3	**x2APIC Interrupt Request Register Bits 127:96 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
824H	2084	IA32_X2APIC_IRR4	**x2APIC Interrupt Request Register Bits 159:128 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
825H	2085	IA32_X2APIC_IRR5	**x2APIC Interrupt Request Register Bits 191:160 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
826H	2086	IA32_X2APIC_IRR6	**x2APIC Interrupt Request Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
827H	2087	IA32_X2APIC_IRR7	**x2APIC Interrupt Request Register Bits 255:224 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
828H	2088	IA32_X2APIC_ESR	**x2APIC Error Status Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
82FH	2095	IA32_X2APIC_LVT_CMCI	**x2APIC LVT Corrected Machine Check Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
830H	2096	IA32_X2APIC_ICR	**x2APIC Interrupt Command Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
832H	2098	IA32_X2APIC_LVT_TIMER	**x2APIC LVT Timer Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
833H	2099	IA32_X2APIC_LVT_THERMAL	**x2APIC LVT Thermal Sensor Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
834H	2100	IA32_X2APIC_LVT_PMI	**x2APIC LVT Performance Monitor Interrupt Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
835H	2101	IA32_X2APIC_LVT_LINT0	**x2APIC LVT LINT0 Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
836H	2102	IA32_X2APIC_LVT_LINT1	**x2APIC LVT LINT1 Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
837H	2103	IA32_X2APIC_LVT_ERROR	**x2APIC LVT Error Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
838H	2104	IA32_X2APIC_INIT_COUNT	**x2APIC Initial Count Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
839H	2105	IA32_X2APIC_CUR_COUNT	**x2APIC Current Count Register (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
83EH	2110	IA32_X2APIC_DIV_CONF	**x2APIC Divide Configuration Register (R/W)**	If (CPUID.01H:ECX.[bit 21] = 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			
83FH	2111	IA32_X2APIC_SELF_IPI	**x2APIC Self IPI Register (W/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
C80H	3200	IA32_DEBUG_INTERFACE	**Silicon Debug Feature Control (R/W)**	If(CPUID.01H:ECX.[bit 11] = 1
		0	**Enable (R/W).** BIOS set 1 to enable Silicon debug features. Default is 0	If(CPUID.01H:ECX.[bit 11] = 1
		29:1	Reserved.	
		30	**Lock (R/W)**: If 1, locks any further change to the MSR. The lock bit is set automatically on the first SMI assertion even if not explicitly set by BIOS. Default is 0.	If(CPUID.01H:ECX.[bit 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.[bit 11] = 1
		63:32	Reserved.	
C8DH	3213	IA32_QM_EVTSEL	**Monitoring Event Select Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX.[bit 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.[bit 12] = 1)
		61:0	**Resource Monitored Data**	
		62	**Unavailable**: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
		63	**Error:** If 1, indicates and unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
C8FH	3215	IA32_PQR_ASSOC	**Resource Association Register (R/W)**	If (CPUID.(EAX=07H, ECX=0):EBX.[bit 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.[bit 15] = 1)
C90H - D8FH		Reserved MSR Address Space for Platform Enforcement Mask Registers	**See Section 17.15.2.1, "Enumeration and Detection Support of Cache Allocation Technology"**	
C90H	3216	IA32_L3_MASK_0	**L3 CQE Mask for COS0 (R/W)**	If (CPUID.(10H, 0):EBX[bit 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.(10H, 1):EDX[15:0]
		31:0	**Capacity Bit Mask (R/W).**	
		63:32	Reserved.	
DA0H	3488	IA32_XSS	**Extended Supervisor State Mask (R/W)**	If(CPUID.(0DH, 1):EAX.[bit 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.[bit 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.[bit 13] = 1
		63:1	Reserved.	
DB1H	3505	IA32_PM_CTL1	**Enable/disable HWP (R/W)**	If(CPUID.06H:EAX.[bit 13] = 1
		0	**HDC_Allow_Block (R/W)** Allow/Block this logical processor for package level HDC control. See Section 14.5.3	If(CPUID.06H:EAX.[bit 13] = 1
		63:1	Reserved.	
DB2H	3506	IA32_THREAD_STALL	**Per-Logical_Processor HDC Idle Residency (R/0)**	If(CPUID.06H:EAX.[bit 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			
		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.[bit 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.80000001.EDX.[bit 20] or CPUID.80000001.EDX.[bit 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.[bit 29] = 1
C000_0082H		IA32_LSTAR	**IA-32e Mode System Call Target Address (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1
C000_0084H		IA32_FMASK	**System Call Flag Mask (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1
C000_0100H		IA32_FS_BASE	**Map of BASE Address of FS (R/W)**	If CPUID.80000001.EDX.[bit 29] = 1
C000_0101H		IA32_GS_BASE	**Map of BASE Address of GS (R/W)**	If CPUID.80000001.EDX.[bit 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_0102H		IA32_KERNEL_GS_BASE	**Swap Target of BASE Address of GS (R/W)**	If CPUID.80000001.EDX.[bit 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."
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 17.13, "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)**

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		9		**Execute BIST (R/O)** 1 = Enabled; 0 = Disabled
		10		**AERR# Observation Enabled (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved.

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 17.11, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
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.

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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:

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.

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

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 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.
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

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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**
		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.
1A0	416	IA32_MISC_ENABLE		**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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.

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		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.

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
1DEH	478	MSR_LER_TO_LIP	Core	**Last Exception Record To Linear IP (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_ 00000	Core	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Core	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Core	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 35-2.
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.

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		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."

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
402H	1026	IA32_MC0_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	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."
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.

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

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

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
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
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.

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

Address		Register Name	Scope	Bit Description
Hex	Dec			
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 future Intel Atom processors (CPUID signatures with DisplayFamily_DisplayModel of 06_4AH, 06_5AH, 06_5DH).

Table 35-7 Specific MSRs Supported by Intel® Atom™ Processors with CPUID Signature 06_37H, 06_4AH, 06_5AH, 06_5DH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)** See Section 14.9.1, "RAPL Interfaces."
		3:0		Power Units. Power related information (in milliWatts) is based on the multiplier, 2^PU; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units. Energy related information (in microJoules) is based on the multiplier, 2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.
		15:13		Reserved
		19:16		Time Unit. The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	**PKG RAPL Power Limit Control (R/W)**
		14:0		Package Power Limit #1. (R/W) See Section 14.9.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 35-7.
		15		Enable Power Limit #1. (R/W) See Section 14.9.3, "Package RAPL Domain."

Table 35-7 Specific MSRs Supported by Intel® Atom™ Processors with CPUID Signature 06_37H, 06_4AH, 06_5AH, 06_5DH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		16		Package Clamping Limitation #1. (R/W) See Section 14.9.3, "Package RAPL Domain."
		23:17		Time Window for Power Limit #1. (R/W) in unit of second. If 0 is specified in bits [23:17], defaults to 1 second window.
		63:24		Reserved
611H	1553	MSR_PKG_ENERGY_STATUS	Package	**PKG Energy Status (R/O)** See Section 14.9.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 35-7
639H	1593	MSR_PP0_ENERGY_STATUS	Package	**PP0 Energy Status (R/O)** See Section 14.9.4, "PP0/PP1 RAPL Domains." and MSR_RAPL_POWER_UNIT in Table 35-7

Table 35-8 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor E3000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_37H).

Table 35-8 Specific MSRs Supported by Intel® Atom™ Processor E3000 Series with CPUID Signature 06_37H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
668H	1640	MSR_CC6_DEMOTION_POLICY_CONFIG	Package	**Core C6 demotion policy config MSR**
		63:0		Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.
669H	1641	MSR_MC6_DEMOTION_POLICY_CONFIG	Package	**Module C6 demotion policy config MSR**
		63:0		Controls module (i.e. two cores sharing the second-level cache) C6 demotion policy. Writing a value of 0 disables module level HW demotion policy.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	**Module C6 Residency Counter (R/O)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.

Table 35-9 lists model-specific registers (MSRs) that are specific to Intel® Atom™ processor C2000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_4DH).

Table 35-9 Specific MSRs Supported by Intel® Atom™ Processor C2000 Series with CPUID Signature 06_4DH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode (RW)**
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		39:32	Package	**Maximum Ratio Limit for 5C** Maximum turbo ratio limit of 5 core active.
		47:40	Package	**Maximum Ratio Limit for 6C** Maximum turbo ratio limit of 6 core active.
		55:48	Package	**Maximum Ratio Limit for 7C** Maximum turbo ratio limit of 7 core active.
		63:56	Package	**Maximum Ratio Limit for 8C** Maximum turbo ratio limit of 8 core active.
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)** See Section 14.9.1, "RAPL Interfaces."
		3:0		Power Units. Power related information (in milliWatts) is based on the multiplier, 2^PU; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.
		7:4		Reserved
		12:8		Energy Status Units. Energy related information (in microJoules) is based on the multiplier, 2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.
		15:13		Reserved
		19:16		Time Unit. The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	**PKG RAPL Power Limit Control (R/W)** See Section 14.9.3, "Package RAPL Domain."
66EH	1646	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameter (R/O)**

Table 35-9 Specific MSRs Supported by Intel® Atom™ Processor C2000 Series (Contd.)with CPUID Signature

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		14:0		Thermal Spec Power. (R/0) The unsigned integer value is the equivalent of thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT
		63:15		Reserved

35.4.1 MSRs In Future Intel Atom Processors Based on Airmont Microarchitecture

Intel Atom processors based on the Airmont microarchitecture supports MSRs listed in Table 35-6, Table 35-7, and Table 35-10. These processors have a CPUID signature with DisplayFamily_DisplayModel including 06_4CH, see Table 35-1.

Table 35-10 MSRs in Intel Atom Processors Based on the Airmont Microarchitecture

Address		Register Name	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO)** This field indicates the intended scaleable bus clock speed for processors based on Airmont microarchitecture:
		4:0		▪ 00000B: 083.3 MHz ▪ 00001B: 100.0 MHz ▪ 00010B: 133.3 MHz ▪ 00011B: 116.5 MHz ▪ 00100B: 083.3 MHz ▪ 00101B: 100.0 MHz ▪ 00110B: 133.3 MHz ▪ 00111B: 116.7 MHz ▪ 01100B: 080.0 MHz ▪ 01101B: 093.3 MHz ▪ 01110B: 090.0 MHz ▪ 01111B: 088.9 MHz ▪ 10100B: 087.5 MHz
		63:5		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.

Table 35-10 MSRs in Intel Atom Processors Based on the Airmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: No limit 001b: C1 010b: C2 110b: C6 111b: C7
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Shared	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - Deep Power Down Technology is the max C-State 010b - C7 is the max C-State to include
		63:19		Reserved.
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)**
		14:0		PP0 Power Limit #1. (R/W) See Section 14.9.4, "PP0/PP1 RAPL Domains." and MSR_RAPL_POWER_UNIT in Table 35-7.
		15		Enable Power Limit #1. (R/W) See Section 14.9.4, "PP0/PP1 RAPL Domains."

Table 35-10 MSRs in Intel Atom Processors Based on the Airmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		16		Reserved
		23:17		Time Window for Power Limit #1. (R/W) Specifies the time duration over which the average power must remain below PP0_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.
		63:24		Reserved

35.5 MSRS IN THE INTEL® MICROARCHITECTURE CODE NAME NEHALEM

Table 35-11 lists model-specific registers (MSRs) that are common for Intel® microarchitecture code name Nehalem. These include Intel Core i7 and i5 processor family. Architectural MSR addresses are also included in Table 35-11. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH, 06_1FH, 06_2EH, see Table 35-1. Additional MSRs specific to 06_1AH, 06_1EH, 06_1FH are listed in Table 35-12. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at http://biosbits.org.

The column "Scope" represents the package/core/thread scope of individual bit field of an MSR. "Thread" means this bit field must be programmed on each logical processor independently. "Core" means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. "Package" means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem

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

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 17.13, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Package	**Platform ID (R)** See Table 35-2.
17H	23	MSR_PLATFORM_ID	Package	**Model Specific Platform ID (R)**
		49:0		Reserved.
		52:50		See Table 35-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
34H	52	MSR_SMI_COUNT	Thread	**SMI Counter (R/O)**
		31:0		**SMI Count (R/O)** Running count of SMI events since last RESET.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64Processor (R/W)** See Table 35-2.
79H	121	IA32_BIOS_ UPDT_TRIG	Core	**BIOS Update Trigger Register (W)** See Table 35-2.
8BH	139	IA32_BIOS_ SIGN_ID	Thread	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Thread	**Performance Counter Register** See Table 35-2.
C2H	194	IA32_PMC1	Thread	**Performance Counter Register** See Table 35-2.
C3H	195	IA32_PMC2	Thread	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Thread	**Performance Counter Register** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	see http://biosbits.org.
		7:0		Reserved.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDC-TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDC/TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 133.33MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-sate support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		23:16		Reserved.
		24		**Interrupt filtering enable (R/W)** When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.
		25		**C3 state auto demotion enable (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		63:27		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSEL0	Thread	See Table 35-2.
		7:0		**Event Select**
		15:8		**UMask**
		16		**USR**
		17		**OS**
		18		**Edge**
		19		**PC**
		20		**INT**
		21		**AnyThread**
		22		**EN**
		23		**INV**
		31:24		**CMASK**
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 35-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 35-2.
198H	408	IA32_PERF_STATUS	Core	See Table 35-2.
		15:0		Current Performance State Value.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:16		Reserved.
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	**Clock Modulation (R/W)** See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved.
		3:1		**On demand Clock Modulation Duty Cycle (R/W)**
		4		**On demand Clock Modulation Enable (R/W)**
		63:5		Reserved.
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.
1A0	416	IA32_MISC_ENABLE		**Enable Misc. Processor Features (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0	Thread	**Fast-Strings Enable** See Table 35-2.
		2:1		Reserved.
		3	Thread	**Automatic Thermal Control Circuit Enable (R/W)** See Table 35-2.
		6:4		Reserved.
		7	Thread	**Performance Monitoring Available (R)** See Table 35-2.
		10:8		Reserved.
		11	Thread	**Branch Trace Storage Unavailable (RO)** See Table 35-2.
		12	Thread	**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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		22	Thread	**Limit CPUID Maxval (R/W)** See Table 35-2.
		23	Thread	**xTPR Message Disable (R/W)** See Table 35-2.
		33:24		Reserved.
		34	Thread	**XD Bit Disable (R/W)** See Table 35-2.
		37:35		Reserved.
		38	Package	**Turbo Mode Disable (R/W)** When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of turbo mode. If power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1A2H	418	MSR_ TEMPERATURE_TARGET	Thread	
		15:0		Reserved.
		23:16		**Temperature Target (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A4H	420	MSR_MISC_FEATURE_ CONTROL		**Miscellaneous Feature Control (R/W)**
		0	Core	**L2 Hardware Prefetcher Disable (R/W)** If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1	Core	**L2 Adjacent Cache Line Prefetcher Disable (R/W)** If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2	Core	**DCU Hardware Prefetcher Disable (R/W)** If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		3	Core	**DCU IP Prefetcher Disable (R/W)** If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction Pointer of previous loads) to determine whether to prefetch additional lines.
		63:4		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	**Offcore Response Event Select Register (R/W)**
1AAH	426	MSR_MISC_PWR_MGMT		See http://biosbits.org.
		0	Package	**EIST Hardware Coordination Disable (R/W)** When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores; When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.
		1	Thread	**Energy/Performance Bias Enable (R/W)** This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.(EAX=06h):ECX[3].
		63:2		Reserved.
1ACH	428	MSR_TURBO_POWER_ CURRENT_LIMIT		See http://biosbits.org.
		14:0	Package	**TDP Limit (R/W)** TDP limit in 1/8 Watt granularity.
		15	Package	**TDP Limit Override Enable (R/W)** A value = 0 indicates override is not active, and a value = 1 indicates active.
		30:16	Package	**TDC Limit (R/W)** TDC limit in 1/8 Amp granularity.
		31	Package	**TDC Limit Override Enable (R/W)** A value = 0 indicates override is not active, and a value = 1 indicates active.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** **RO** if MSR_PLATFORM_INFO.[28] = 0, **RW** if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
1C8H	456	MSR_LBR_SELECT	Core	**Last Branch Record Filtering Select Register (R/W)** See Section 17.6.2, "Filtering of Last Branch Records."
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 35-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	**Last Exception Record From Linear IP (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	**Last Exception Record To Linear IP (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See http://biosbits.org.
		0		Reserved.
		1	Package	**C1E Enable (R/W)** When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 35-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 35-2.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 35-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_ 00000	Thread	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Thread	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Thread	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 35-2.
277H	631	IA32_PAT	Thread	See Table 35-2.
280H	640	IA32_MC0_CTL2	Package	See Table 35-2.
281H	641	IA32_MC1_CTL2	Package	See Table 35-2.
282H	642	IA32_MC2_CTL2	Core	See Table 35-2.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
283H	643	IA32_MC3_CTL2	Core	See Table 35-2.
284H	644	IA32_MC4_CTL2	Core	See Table 35-2.
285H	645	IA32_MC5_CTL2	Core	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	**Default Memory Types (R/W)** See Table 35-2.
309H	777	IA32_FIXED_CTR0	Thread	**Fixed-Function Performance Counter Register 0 (R/W)** See Table 35-2.
30AH	778	IA32_FIXED_CTR1	Thread	**Fixed-Function Performance Counter Register 1 (R/W)** See Table 35-2.
30BH	779	IA32_FIXED_CTR2	Thread	**Fixed-Function Performance Counter Register 2 (R/W)** See Table 35-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 35-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		5:0		LBR Format. See Table 35-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 35-2.
		11:8		PEBS_REC_FORMAT. See Table 35-2.
		12		SMM_FREEZE. See Table 35-2.
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register (R/W)** See Table 35-2.
38EH	910	IA32_PERF_GLOBAL_ STAUS	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STAUS	Thread	**(RO)**
		61		**UNC_Ovf** Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_ CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_GLOBAL_OVF_ CTRL	Thread	**(R/W)**

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		61		**CLR_UNC_Ovf** Set 1 to clear UNC_Ovf.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.7.1.1, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 18.7.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
400H	1024	IA32_MC0_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	MSR_MC0_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	MSR_MC1_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40BH	1035	MSR_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40FH	1039	MSR_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."
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_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	MSR_MC4_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
414H	1044	MSR_MC5_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	MSR_MC5_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	MSR_MC5_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	MSR_MC5_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	MSR_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41AH	1050	MSR_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	MSR_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	MSR_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41EH	1054	MSR_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	MSR_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	MSR_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
422H	1058	MSR_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	MSR_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
480H	1152	IA32_VMX_BASIC	Thread	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_ CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_ CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
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."

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Thread	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Thread	**DS Save Area (R/W)** See Table 35-2. See Section 18.13.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	**Last Branch Record 0 From IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** for one of the last sixteen branches, exceptions, or interrupts taken by the processor. 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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
68DH	1677	MSR_ LASTBRANCH_13_FROM_IP	Thread	**Last Branch Record 13 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_ LASTBRANCH_14_FROM_IP	Thread	**Last Branch Record 14 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_ LASTBRANCH_15_FROM_IP	Thread	**Last Branch Record 15 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_ LASTBRANCH_0_TO_IP	Thread	**Last Branch Record 0 To IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor.
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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6CDH	1741	MSR_ LASTBRANCH_13_TO_IP	Thread	**Last Branch Record 13 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_ LASTBRANCH_14_TO_IP	Thread	**Last Branch Record 14 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_ LASTBRANCH_15_TO_IP	Thread	**Last Branch Record 15 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID register (R/O) See x2APIC Specification.
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request register bits [63:32] (R/O)
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)

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI register (W/O)
C000_0080H		IA32_EFER	Thread	**Extended Feature Enables** See Table 35-2.
C000_0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_0084H		IA32_FMASK	Thread	**System Call Flag Mask (R/W)** See Table 35-2.
C000_0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Thread	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.
C000_0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature. (R/W)** See Table 35-2 and Section 17.13.2, "IA32_TSC_AUX Register and RDTSCP Support."

...

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. All architectural MSRs listed in Table 35-2 are supported. 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 17.13, "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.

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			
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** See Table 35-2.
C6H	198	IA32_PMC5	Core	**Performance Counter Register** See Table 35-2.
C7H	199	IA32_PMC6	Core	**Performance Counter Register** See Table 35-2.
C8H	200	IA32_PMC7	Core	**Performance Counter Register** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		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.

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			
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		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.

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			
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		**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.
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.

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			
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.
		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.

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			
		31		**Reading Valid (RO)** See Table 35-2.
		63:32		Reserved.
1A0	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."
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 35-2.
		0		**LBR: Last Branch Record**
		1		**BTF**
		5:2		Reserved.
		6		**TR: Branch Trace**
		7		**BTS: Log Branch Trace Message to BTS buffer**
		8		**BTINT**
		9		**BTS_OFF_OS**
		10		**BTS_OFF_USER**
		11		**FREEZE_LBR_ON_PMI**
		12		**FREEZE_PERFMON_ON_PMI**
		13		**ENABLE_UNCORE_PMI**
		14		**FREEZE_WHILE_SMM**
		63:15		Reserved.
1DDH	477	MSR_LER_FROM_LIP	Thread	**Last Exception Record From Linear IP (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	**Last Exception Record To Linear IP (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.

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			
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.
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.

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			
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.
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. See Section 18.4.2, "Global Counter Control Facilities."
		0		**Ovf_PMC0**
		1		**Ovf_PMC1**
		2		**Ovf_PMC2**
		3		**Ovf_PMC3**
		31:4		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			
		32		**Ovf_FixedCtr0**
		33		**Ovf_FixedCtr1**
		34		**Ovf_FixedCtr2**
		60:35		Reserved.
		61		**Ovf_Uncore**
		62		**Ovf_BufDSSAVE**
		63		**CondChgd**
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0		**Set 1 to clear Ovf_PMC0**
		1		**Set 1 to clear Ovf_PMC1**
		2		**Set 1 to clear Ovf_PMC2**
		3		**Set 1 to clear Ovf_PMC3**
		31:4		Reserved.
		32		**Set 1 to clear Ovf_FixedCtr0**
		33		**Set 1 to clear Ovf_FixedCtr1**
		34		**Set 1 to clear Ovf_FixedCtr2**
		60:35		Reserved.
		61		**Set 1 to clear Ovf_Uncore**
		62		**Set 1 to clear Ovf_BufDSSAVE**
		63		**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)
		63:36		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			
3F6H	1014	MSR_PEBS_LD_LAT	Thread	see See Section 18.7.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.

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			
		63:0		CORE C7 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
403H	1027	IA32_MC0_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
406H	1030	IA32_MC1_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
407H	1031	IA32_MC1_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40BH	1035	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
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."

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			
482H	1154	IA32_VMX_PROCBASED_CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Controls (R/O)** See Table 35-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Controls (R/O)** See Table 35-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	**Reporting Register of Miscellaneous VMX Capabilities (R/O)** See Table 35-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	**Capability Reporting Register of CR0 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
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

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			
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Table 35-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Table 35-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Flex Controls (R/O)** See Table 35-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 35-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 35-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 35-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 35-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 35-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 35-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 35-2.
4C8H	200	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.

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		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60BH	1547	MSR_PKGC6_IRTL	Package	**Package C6 Interrupt Response Limit (R/W)** This MSR defines the budget allocated for the package to exit from C6 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C6 state.
		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."

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			
614H	1556	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameters (R/W)** See Section 14.9.3, "Package RAPL Domain."
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	**PP0 Energy Status (R/O)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
680H	1664	MSR_ LASTBRANCH_0_FROM_IP	Thread	**Last Branch Record 0 From IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** for one of the last sixteen branches, exceptions, or interrupts taken by the processor. 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.

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			
68CH	1676	MSR_ LASTBRANCH_12_FROM_ IP	Thread	**Last Branch Record 12 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_ LASTBRANCH_13_FROM_ IP	Thread	**Last Branch Record 13 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_ LASTBRANCH_14_FROM_ IP	Thread	**Last Branch Record 14 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_ LASTBRANCH_15_FROM_ IP	Thread	**Last Branch Record 15 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_ LASTBRANCH_0_TO_IP	Thread	**Last Branch Record 0 To IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor.
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.

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			
6CBH	1739	MSR_ LASTBRANCH_11_TO_IP	Thread	**Last Branch Record 11 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_ LASTBRANCH_12_TO_IP	Thread	**Last Branch Record 12 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_ LASTBRANCH_13_TO_IP	Thread	**Last Branch Record 13 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_ LASTBRANCH_14_TO_IP	Thread	**Last Branch Record 14 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_ LASTBRANCH_15_TO_IP	Thread	**Last Branch Record 15 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table 35-2.
802H-83FH		X2APIC MSRs	Thread	See Table 35-2.
C000_0080H		IA32_EFER	Thread	**Extended Feature Enables** See Table 35-2.
C000_0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
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 17.13.2, "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
		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

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			
		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_PER_CTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PER_CTR1	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

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			
		14:13		Reserved.
		15		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
63AH	1594	MSR_PP0_POLICY	Package	**PP0 Balance Policy (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
63BH	1595	MSR_PP0_PERF_STATUS	Package	**PP0 Performance Throttling Status (R/O)** 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_PER_CTR0	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_PER_CTR1	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_PER_CTR0	Package	Uncore C-Box 1, performance counter 0
717H	1815	MSR_UNC_CBO_1_PER_CTR1	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_PER_CTR0	Package	Uncore C-Box 2, performance counter 0

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			
727H	1831	MSR_UNC_CBO_2_PER_CTR1	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_PER_CTR0	Package	Uncore C-Box 3, performance counter 0.
737H	1847	MSR_UNC_CBO_3_PER_CTR1	Package	Uncore C-Box 3, performance counter 1.

…

35.9 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY (BASED ON INTEL® MICROARCHITECTURE CODE NAME IVY BRIDGE)

The 3rd generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v2 product family (based on Intel microarchitecture code name Ivy Bridge) supports the MSR interfaces listed in Table 35-16, Table 35-17 and Table 35-19.

Table 35-19 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.

Table 35-19 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		31:30		Reserved.
		32	Package	**Low Power Mode Support (LPM) (R/O)** When set to 1, indicates that LPM is supported, and when set to 0, indicates LPM is not supported.
		34:33	Package	**Number of ConfigTDP Levels (R/O)** 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 11: Reserved
		39:35		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		55:48	Package	**Minimum Operating Ratio (R/O)** Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.

Table 35-19 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		24:16		Reserved.
		25		**C3 state auto demotion enable (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		**Enable C3 undemotion (R/W)** When set, enables undemotion from demoted C3.
		28		**Enable C1 undemotion (R/W)** When set, enables undemotion from demoted C1.
		63:29		Reserved.
648H	1608	MSR_CONFIG_TDP_ NOMINAL	Package	**Base TDP Ratio (R/O)**
		7:0		**Config_TDP_Base** Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved.
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O)
		14:0		PKG_TDP_LVL1. Power setting for ConfigTDP Level 1.
		15		Reserved
		23:16		Config_TDP_LVL1_Ratio. ConfigTDP level 1 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL1. Max Power setting allowed for ConfigTDP Level 1.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL1. MIN Power setting allowed for ConfigTDP Level 1.

Table 35-19 Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (based on Intel® microarchitecture code name Ivy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63		Reserved.
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O)
		14:0		PKG_TDP_LVL2. Power setting for ConfigTDP Level 2.
		15		Reserved
		23:16		Config_TDP_LVL2_Ratio. ConfigTDP level 2 ratio to be used for this specific processor.
		31:24		Reserved
		46:32		PKG_MAX_PWR_LVL2. Max Power setting allowed for ConfigTDP Level 2.
		47		Reserved
		62:48		PKG_MIN_PWR_LVL2. MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved.
64BH	1611	MSR_CONFIG_TDP_ CONTROL	Package	**ConfigTDP Control (R/W)**
		1:0		**TDP_LEVEL (RW/L)** System BIOS can program this field.
		30:2		Reserved.
		31		**Config_TDP_Lock (RW/L)** When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
64CH	1612	MSR_TURBO_ACTIVATION_ RATIO	Package	**ConfigTDP Control (R/W)**
		7:0		**MAX_NON_TURBO_RATIO (RW/L)** System BIOS can program this field.
		30:8		Reserved.
		31		**TURBO_ACTIVATION_RATIO_Lock (RW/L)** When this bit is set, the content of this register is locked until a reset.
		63:32		Reserved.
See Table 35-16, Table 35-17 for other MSR definitions applicable to processors with CPUID signature 06_3AH				

35.9.1 MSRs In Intel® Xeon® Processor E5 v2 Product Family (Based on Ivy Bridge-E Microarchitecture)

Table 35-20 lists model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 v2 Product Family (based on Ivy Bridge-E microarchitecture). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3EH, see Table 35-1. These processors supports the MSR interfaces listed in Table 35-16, and Table 35-20.

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
4EH	78	MSR_PPIN_CTL	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) Set 1to prevent further writes to MSR_PPIN_CTL. Writing 1 to MSR_PPINCTL[bit 0] is permitted only if MSR_PPIN_CTL[bit 1] is clear, Default is 0. BIOS should provide an opt-in menu to enable the user to turn on MSR_PPIN_CTL[bit 1] for privileged inventory initialization agent to access MSR_PPIN. After reading MSR_PPIN, the privileged inventory initialization agent should write '01b' to MSR_PPIN_CTL to disable further access to MSR_PPIN and prevent unauthorized modification to MSR_PPIN_CTL.
		1		Enable_PPIN (R/W) If 1, enables MSR_PPIN to be accessible using RDMSR. Once set, attempt to write 1 to MSR_PPIN_CTL[bit 0] will cause #GP. If 0, an attempt to read MSR_PPIN will cause #GP. Default is 0.
		63:2		Reserved.
4FH	79	MSR_PPIN	Package	Protected Processor Inventory Number (R/O)
		63:0		**Protected Processor Inventory Number (R/O)** A unique value within a given CPUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to MSR_PPIN is enabled. Access to MSR_PPIN is permitted only if MSR_PPIN_CTL[bits 1:0] = '10b'
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		22:16		Reserved.

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23	Package	**PPIN_CAP (R/O)** When set to 1, indicates that Protected Processor Inventory Number (PPIN) capability can be enabled for privileged system inventory agent to read PPIN from MSR_PPIN. When set to 0, PPIN capability is not supported. An attempt to access MSR_PPIN_CTL or MSR_PPIN will cause #GP.
		27:24		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		63:16		Reserved.
179H	377	IA32_MCG_CAP	Thread	**Global Machine Check Capability (R/O)**
		7:0		**Count**
		8		**MCG_CTL_P**
		9		**MCG_EXT_P**
		10		**MCP_CMCI_P**
		11		**MCG_TES_P**
		15:12		Reserved.
		23:16		**MCG_EXT_CNT**
		24		**MCG_SER_P**
		25		Reserved.
		26		**MCG_ELOG_P**
		63:27		Reserved.
17FH	383	MSR_ERROR_CONTROL	Package	**MC Bank Error Configuration (R/W)**
		0		Reserved
		1		**MemError Log Enable (R/W)** When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
1AEH	430	MSR_TURBO_RATIO_LIMIT 1	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 9C** Maximum turbo ratio limit of 9 core active.
		15:8	Package	**Maximum Ratio Limit for 10C** Maximum turbo ratio limit of 10core active.
		23:16	Package	**Maximum Ratio Limit for 11C** Maximum turbo ratio limit of 11 core active.

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:24	Package	**Maximum Ratio Limit for 12C** Maximum turbo ratio limit of 12 core active.
		63:32		Reserved
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
294H	660	IA32_MC20_CTL2	Package	See Table 35-2.
295H	661	IA32_MC21_CTL2	Package	See Table 35-2.
296H	662	IA32_MC22_CTL2	Package	See Table 35-2.
297H	663	IA32_MC23_CTL2	Package	See Table 35-2.
298H	664	IA32_MC24_CTL2	Package	See Table 35-2.
299H	665	IA32_MC25_CTL2	Package	See Table 35-2.
29AH	666	IA32_MC26_CTL2	Package	See Table 35-2.
29BH	667	IA32_MC27_CTL2	Package	See Table 35-2.
29CH	668	IA32_MC28_CTL2	Package	See Table 35-2.
414H	1044	MSR_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC5 reports MC error from the Intel QPI module.
415H	1045	MSR_MC5_STATUS	Package	
416H	1046	MSR_MC5_ADDR	Package	
417H	1047	MSR_MC5_MISC	Package	

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC6 reports MC error from the integrated I/O module.
419H	1049	MSR_MC6_STATUS	Package	
41AH	1050	MSR_MC6_ADDR	Package	
41BH	1051	MSR_MC6_MISC	Package	
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC7 and MC 8 report MC error from the two home agents.
41DH	1053	MSR_MC7_STATUS	Package	
41EH	1054	MSR_MC7_ADDR	Package	
41FH	1055	MSR_MC7_MISC	Package	
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC7 and MC 8 report MC error from the two home agents.
421H	1057	MSR_MC8_STATUS	Package	
422H	1058	MSR_MC8_ADDR	Package	
423H	1059	MSR_MC8_MISC	Package	
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
425H	1061	MSR_MC9_STATUS	Package	
426H	1062	MSR_MC9_ADDR	Package	
427H	1063	MSR_MC9_MISC	Package	
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
429H	1065	MSR_MC10_STATUS	Package	
42AH	1066	MSR_MC10_ADDR	Package	
42BH	1067	MSR_MC10_MISC	Package	
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	MSR_MC11_STATUS	Package	Bank MC11 reports MC error from a specific channel of the integrated memory controller.
42EH	1070	MSR_MC11_ADDR	Package	
42FH	1071	MSR_MC11_MISC	Package	
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
431H	1073	MSR_MC12_STATUS	Package	
432H	1074	MSR_MC12_ADDR	Package	
433H	1075	MSR_MC12_MISC	Package	
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
435H	1077	MSR_MC13_STATUS	Package	
436H	1078	MSR_MC13_ADDR	Package	
437H	1079	MSR_MC13_MISC	Package	

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
439H	1081	MSR_MC14_STATUS	Package	
43AH	1082	MSR_MC14_ADDR	Package	
43BH	1083	MSR_MC14_MISC	Package	
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
43DH	1085	MSR_MC15_STATUS	Package	
43EH	1086	MSR_MC15_ADDR	Package	
43FH	1087	MSR_MC15_MISC	Package	
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
441H	1089	MSR_MC16_STATUS	Package	
442H	1090	MSR_MC16_ADDR	Package	
443H	1091	MSR_MC16_MISC	Package	
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC17 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
445H	1093	MSR_MC17_STATUS	Package	
446H	1094	MSR_MC17_ADDR	Package	
447H	1095	MSR_MC17_MISC	Package	
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC18 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
449H	1097	MSR_MC18_STATUS	Package	
44AH	1098	MSR_MC18_ADDR	Package	
44BH	1099	MSR_MC18_MISC	Package	
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC19 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
44DH	1101	MSR_MC19_STATUS	Package	
44EH	1102	MSR_MC19_ADDR	Package	
44FH	1103	MSR_MC19_MISC	Package	
450H	1104	MSR_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	MSR_MC20_STATUS	Package	Bank MC20 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
452H	1106	MSR_MC20_ADDR	Package	
453H	1107	MSR_MC20_MISC	Package	
454H	1108	MSR_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC21 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
455H	1109	MSR_MC21_STATUS	Package	
456H	1110	MSR_MC21_ADDR	Package	
457H	1111	MSR_MC21_MISC	Package	

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
458H	1112	MSR_MC22_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC22 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
459H	1113	MSR_MC22_STATUS	Package	
45AH	1114	MSR_MC22_ADDR	Package	
45BH	1115	MSR_MC22_MISC	Package	
45CH	1116	MSR_MC23_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC23 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
45DH	1117	MSR_MC23_STATUS	Package	
45EH	1118	MSR_MC23_ADDR	Package	
45FH	1119	MSR_MC23_MISC	Package	
460H	1120	MSR_MC24_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC24 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
461H	1121	MSR_MC24_STATUS	Package	
462H	1122	MSR_MC24_ADDR	Package	
463H	1123	MSR_MC24_MISC	Package	
464H	1124	MSR_MC25_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC25 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
465H	1125	MSR_MC25_STATUS	Package	
466H	1126	MSR_MC25_ADDR	Package	
467H	1127	MSR_MC25_MISC	Package	
468H	1128	MSR_MC26_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC26 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
469H	1129	MSR_MC26_STATUS	Package	
46AH	1130	MSR_MC26_ADDR	Package	
46BH	1131	MSR_MC26_MISC	Package	
46CH	1132	MSR_MC27_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC27 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
46DH	1133	MSR_MC27_STATUS	Package	
46EH	1134	MSR_MC27_ADDR	Package	
46FH	1135	MSR_MC27_MISC	Package	
470H	1136	MSR_MC28_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC28 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
471H	1137	MSR_MC28_STATUS	Package	
472H	1138	MSR_MC28_ADDR	Package	
473H	1139	MSR_MC28_MISC	Package	
613H	1555	MSR_PKG_PERF_STATUS	Package	**Package RAPL Perf Status (R/O)**
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."

Table 35-20 MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
See Table 35-16, for other MSR definitions applicable to Intel Xeon processor E5 v2 with CPUID signature 06_3EH				

35.9.2 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family

Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with CPUID DisplayFamily_DisplayModel signature 06_3EH supports the MSR interfaces listed in Table 35-16, Table 35-20, and Table 35-21.

Table 35-21 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64 Processor (R/W)** See Table 35-2.
		0		**Lock (R/WL)**
		1		**Enable VMX inside SMX operation (R/WL)**
		2		**Enable VMX outside SMX operation (R/WL)**
		14:8		**SENTER local functions enables (R/WL)**
		15		**SENTER global functions enable (R/WL)**
		20		**LMCE_ON (R/WL)**
		63:21		Reserved.
179H	377	IA32_MCG_CAP	Thread	**Global Machine Check Capability (R/O)**
		7:0		**Count**
		8		**MCG_CTL_P**
		9		**MCG_EXT_P**
		10		**MCP_CMCI_P**
		11		**MCG_TES_P**
		15:12		Reserved.
		23:16		**MCG_EXT_CNT**
		24		**MCG_SER_P**
		25		Reserved.

Table 35-21 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		26		**MCG_ELOG_P**
		27		**MCG_LMCE_P**
		63:28		Reserved.
17AH	378	IA32_MCG_STATUS	Thread	(R/W0)
		0		**RIPV**
		1		**EIPV**
		2		**MCIP**
		3		**LMCE signaled**
		63:4		Reserved.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 9C** Maximum turbo ratio limit of 9 core active.
		15:8	Package	**Maximum Ratio Limit for 10C** Maximum turbo ratio limit of 10core active.
		23:16	Package	**Maximum Ratio Limit for 11C** Maximum turbo ratio limit of 11 core active.
		31:24	Package	**Maximum Ratio Limit for 12C** Maximum turbo ratio limit of 12 core active.
		39:32	Package	**Maximum Ratio Limit for 13C** Maximum turbo ratio limit of 13 core active.
		47:40	Package	**Maximum Ratio Limit for 14C** Maximum turbo ratio limit of 14 core active.
		55:48	Package	**Maximum Ratio Limit for 15C** Maximum turbo ratio limit of 15 core active.
		63:56		Reserved
29DH	669	IA32_MC29_CTL2	Package	See Table 35-2.
29EH	670	IA32_MC30_CTL2	Package	See Table 35-2.
29FH	671	IA32_MC31_CTL2	Package	See Table 35-2.
41BH	1051	IA32_MC6_MISC	Package	Misc MAC information of Integrated I/O. (R/O) see Section 15.3.2.4
		5:0		Recoverable Address LSB
		8:6		Address Mode
		15:9		Reserved

Table 35-21 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:16		PCI Express Requestor ID
		39:32		PCI Express Segment Number
		63:32		Reserved
474H	1140	MSR_MC29_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC29 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
475H	1141	MSR_MC29_STATUS	Package	
476H	1142	MSR_MC29_ADDR	Package	
477H	1143	MSR_MC29_MISC	Package	
478H	1144	MSR_MC30_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC30 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
479H	1145	MSR_MC30_STATUS	Package	
47AH	1146	MSR_MC30_ADDR	Package	
47BH	1147	MSR_MC30_MISC	Package	
47CH	1148	MSR_MC31_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC31 reports MC error from a specific CBo (core broadcast) and its corresponding slice of L3.
47DH	1149	MSR_MC31_STATUS	Package	
47EH	1150	MSR_MC31_ADDR	Package	
47FH	1147	MSR_MC31_MISC	Package	
See Table 35-16, Table 35-20 for other MSR definitions applicable to Intel Xeon processor E7 v2 with CPUID signature 06_3AH				

35.10 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS (BASED ON HASWELL MICROARCHITECTURE)

The 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with CPUID DisplayFamily_DisplayModel signature 06_3CH/06_45H/06_46H, support the MSR interfaces listed in Table 35-16, Table 35-17, Table 35-19, and Table 35-22.

The MSRs listed in Table 35-22 also apply to processors based on Haswell-E microarchitecture (see Section 35.11).

Table 35-22 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3BH	59	IA32_TSC_ADJUST	THREAD	**Per-Logical-Processor TSC ADJUST (R/W)** See Table 35-2.
CEH	206	MSR_PLATFORM_INFO	Package	See Table 35-19

Table 35-22 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
186H	390	IA32_PERFEVTSEL0	THREAD	**Performance Event Select for Counter 0 (R/W)** Supports all fields described inTable 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
187H	391	IA32_PERFEVTSEL1	THREAD	**Performance Event Select for Counter 1 (R/W)** Supports all fields described inTable 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
188H	392	IA32_PERFEVTSEL2	THREAD	**Performance Event Select for Counter 2 (R/W)** Supports all fields described inTable 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
		33		IN_TXCP: see Section 18.11.5.1 When IN_TXCP=1 & IN_TX=1 and in sampling, spurious PMI may occur and transactions may continuously abort near overflow conditions. Software should favor using IN_TXCP for counting over sampling. If sampling, software should use large “sample-after” value after clearing the counter configured to use IN_TXCP and also always reset the counter even when no overflow condition was reported.
189H	393	IA32_PERFEVTSEL3	THREAD	**Performance Event Select for Counter 3 (R/W)** Supports all fields described inTable 35-2 and the fields below.
		32		IN_TX: see Section 18.11.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results
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**

Table 35-22 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		11		**FREEZE_LBR_ON_PMI**
		12		**FREEZE_PERFMON_ON_PMI**
		13		**ENABLE_UNCORE_PMI**
		14		**FREEZE_WHILE_SMM**
		15		**RTM_DEBUG**
		63:15		Reserved.
491H	1169	IA32_VMX_FMFUNC	THREAD	**Capability Reporting Register of VM-function Controls (R/O)** See Table 35-2
648H	1608	MSR_CONFIG_TDP_ NOMINAL	Package	**Base TDP Ratio (R/O)** See Table 35-19
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O). See Table 35-19
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O). See Table 35-19
64BH	1611	MSR_CONFIG_TDP_ CONTROL	Package	**ConfigTDP Control (R/W)** See Table 35-19
64CH	1612	MSR_TURBO_ACTIVATION_ RATIO	Package	**ConfigTDP Control (R/W)** See Table 35-19
C80H	3200	IA32_DEBUG_FEATURE	Package	**Silicon Debug Feature Control (R/W)** See Table 35-2.

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

Table 35-23 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 Table 35-1.

Table 35-23 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.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		3:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: 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.
		57:0		**Reserved**
		58		**SMM_Code_Access_Chk (SMM-RO)** If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.
		59		**Long_Flow_Indication (SMM-RO)** If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.
		63:60		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** 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.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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_PER_CTR0	Package	Uncore Arb unit, performance counter 0
3B1H	947	MSR_UNC_ARB_PER_CTR1	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
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_CONT ROL	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

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2		**SMM_Code_Chk_En (SMM-RW)** This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.
		63:3		Reserved
4E2H	1250	MSR_SMM_DELAYED	Package	**SMM Delayed (SMM-RO)** Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.
		N-1:0		**LOG_PROC_STATE (SMM-RO)** Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	**SMM Blocked (SMM-RO)** Reports the blocked state of all logical processors in the package. Available only while in SMM.
		N-1:0		**LOG_PROC_STATE (SMM-RO)** Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
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."

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		13		**Turbo Transition Attenuation Status (R0)** When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:14		**Reserved**
		16		**PROCHOT Log** When set, indicates that the corresponding PROCHOT Status bit is set. Software can write 0 to this bit to clear PROCHOT Status.
		17		**Thermal Log** When set, indicates that the corresponding Thermal status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Thermal Status.
		19:18		Reserved.
		20		**Graphics Driver Log** When set, indicates that the corresponding Graphics Driver status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Graphics Driver Status.
		21		**Autonomous Utilization-Based Frequency Control Log** When set, indicates that the corresponding Autonomous Utilization-Based Frequency Control status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Autonomous Utilization-Based Frequency Control Status.
		22		**VR Therm Alert Log** When set, indicates that the corresponding VR Therm Alert Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear VR Therm Alert Status.
		23		Reserved.
		24		**Electrical Design Point Log** When set, indicates that the corresponding EDP Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear EDP Status.
		25		**Core Power Limiting Log** When set, indicates that the corresponding Core Power Limiting Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Core Power Limiting Status.
		26		**Package-Level PL1 Power Limiting Log** When set, indicates that the corresponding Package-level Power Limiting PL1 Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Package-level Power Limiting PL1 Status.
		27		**Package-Level PL2 Power Limiting Log** When set, indicates that the corresponding Package-level Power Limiting PL2 Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Package-level Power Limiting PL2 Status.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		28		**Max Turbo Limit Log** When set, indicates that the corresponding Max Turbo Limit Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Max Turbo Limit Status.
		29		**Turbo Transition Attenuation Log** When set, indicates that the corresponding Turbo Transition Attenuation Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Turbo Transition Attenuation Status.
		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		Reserved.
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		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.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:12		**Reserved**
		16		**PROCHOT Log** When set, indicates that the corresponding PROCHOT Status bit is set. Software can write 0 to this bit to clear PROCHOT Status.
		17		**Thermal Log** When set, indicates that the corresponding Thermal status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Thermal Status.
		19:18		Reserved.
		20		**Graphics Driver Log** When set, indicates that the corresponding Graphics Driver status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Graphics Driver Status.
		21		Reserved.
		22		**VR Therm Alert Log** When set, indicates that the corresponding VR Therm Alert Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear VR Therm Alert Status.
		23		Reserved.
		24		**Electrical Design Point Log** When set, indicates that the corresponding EDP Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear EDP Status.
		25		**Graphics Power Limiting Log** When set, indicates that the corresponding Graphics Power Limiting Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Graphics Power Limiting Status.
		26		**Package-Level PL1 Power Limiting Log** When set, indicates that the corresponding Package-level Power Limiting PL1 Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Package-level Power Limiting PL1 Status.
		27		**Package-Level PL2 Power Limiting Log** When set, indicates that the corresponding Package-level Power Limiting PL2 Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Package-level Power Limiting PL2 Status.
		63:28		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)**

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0		**PROCHOT Status (R0)** When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		**Thermal Status (R0)** When set, frequency is reduced below the operating system request due to a thermal event.
		5:2		Reserved.
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		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 corresponding PROCHOT Status bit is set. Software can write 0 to this bit to clear PROCHOT Status.
		17		**Thermal Log** When set, indicates that the corresponding Thermal status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Thermal Status.
		21:18		Reserved.
		22		**VR Therm Alert Log** When set, indicates that the corresponding VR Therm Alert Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear VR Therm Alert Status.
		23		Reserved.
		24		**Electrical Design Point Log** When set, indicates that the corresponding EDP Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear EDP Status.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		25		Reserved.
		26		**Package-Level PL1 Power Limiting Log** When set, indicates that the corresponding Package-level Power Limiting PL1 Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Package-level Power Limiting PL1 Status.
		27		**Package-Level PL2 Power Limiting Log** When set, indicates that the corresponding Package-level Power Limiting PL2 Status bit was set since it was last cleared by software. Software can write 0 to this bit to clear Package-level Power Limiting PL2 Status.
		63:28		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_PER_ CTR0	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_PER_ CTR1	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_PER_ CTR0	Package	Uncore C-Box 1, performance counter 0
717H	1815	MSR_UNC_CBO_1_PER_ CTR1	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_PER_ CTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PER_ CTR1	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.

Table 35-23 MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
736H	1846	MSR_UNC_CBO_3_PER_CTR0	Package	Uncore C-Box 3, performance counter 0.
737H	1847	MSR_UNC_CBO_3_PER_CTR1	Package	Uncore C-Box 3, performance counter 1.
See Table 35-16, Table 35-17, Table 35-19, Table 35-22 for other MSR definitions applicable to processors with CPUID signatures 063CH, 06_46H				

35.10.2 Additional Residency MSRs Supported in 4th Generation Intel® Core™ Processors

The 4th generation Intel® Core™ processor family (based on Haswell microarchitecture) with CPUID DisplayFamily_DisplayModel signature 06_45H supports the MSR interfaces listed in Table 35-16, Table 35-17, Table 35-19, Table 35-22, Table 35-23, and Table 35-24.

Table 35-24 Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily_DisplayModel Signature 06_45H

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)**

Table 35-24 Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily_DisplayModel Signature 06_45H

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		26		**C1 State Auto Demotion Enable (R/W)**
		27		**Enable C3 Undemotion (R/W)**
		28		**Enable C1 Undemotion (R/W)**
		63:29		Reserved
630H	1584	MSR_PKG_C8_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C8 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.
		63:60		Reserved
631H	1585	MSR_PKG_C9_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C9 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.
		63:60		Reserved
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		59:0		Package C10 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.
		63:60		Reserved
See Table 35-16, Table 35-17, Table 35-19, Table 35-22, Table 35-23 for other MSR definitions applicable to processors with CPUID signature 06_45H				

35.11 MSRS IN INTEL® XEON® PROCESSOR E5 26XX V3 PRODUCT FAMILY

Intel® Xeon® processor E5 v3 family and Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID DisplayFamily_DisplayModel = 06_3F). These processors supports the MSR interfaces listed in Table 35-16, Table 35-20, Table 35-22, and Table 35-25.

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org.
		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.
		57:0		**Reserved**
		58		**SMM_Code_Access_Chk (SMM-RO)** If set to 1 indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		**Long_Flow_Indication (SMM-RO)** If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.
		63:60		Reserved
17FH	383	MSR_ERROR_CONTROL	Package	**MC Bank Error Configuration (R/W)**

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0		Reserved
		1		**MemError Log Enable (R/W)** When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		39:32	Package	**Maximum Ratio Limit for 5C** Maximum turbo ratio limit of 5 core active.
		47:40	Package	**Maximum Ratio Limit for 6C** Maximum turbo ratio limit of 6 core active.
		55:48	Package	**Maximum Ratio Limit for 7C** Maximum turbo ratio limit of 7 core active.
		63:56	Package	**Maximum Ratio Limit for 8C** Maximum turbo ratio limit of 8 core active.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 9C** Maximum turbo ratio limit of 9 core active.
		15:8	Package	**Maximum Ratio Limit for 10C** Maximum turbo ratio limit of 10 core active.
		23:16	Package	**Maximum Ratio Limit for 11C** Maximum turbo ratio limit of 11 core active.
		31:24	Package	**Maximum Ratio Limit for 12C** Maximum turbo ratio limit of 12 core active.

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		39:32	Package	**Maximum Ratio Limit for 13C** Maximum turbo ratio limit of 13 core active.
		47:40	Package	**Maximum Ratio Limit for 14C** Maximum turbo ratio limit of 14 core active.
		55:48	Package	**Maximum Ratio Limit for 15C** Maximum turbo ratio limit of 15 core active.
		63:56	Package	**Maximum Ratio Limit for16C** Maximum turbo ratio limit of 16 core active.
1AFH	431	MSR_TURBO_RATIO_LIMIT2	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 17C** Maximum turbo ratio limit of 17 core active.
		15:8	Package	**Maximum Ratio Limit for 18C** Maximum turbo ratio limit of 18 core active.
		63:16	Package	Reserved
414H	1044	MSR_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC5 reports MC error from the Intel QPI 0 module.
415H	1045	MSR_MC5_STATUS	Package	
416H	1046	MSR_MC5_ADDR	Package	
417H	1047	MSR_MC5_MISC	Package	
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC6 reports MC error from the integrated I/O module.
419H	1049	MSR_MC6_STATUS	Package	
41AH	1050	MSR_MC6_ADDR	Package	
41BH	1051	MSR_MC6_MISC	Package	
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC7 reports MC error from the home agent HA 0.
41DH	1053	MSR_MC7_STATUS	Package	
41EH	1054	MSR_MC7_ADDR	Package	
41FH	1055	MSR_MC7_MISC	Package	
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC8 reports MC error from the home agent HA 1.
421H	1057	MSR_MC8_STATUS	Package	
422H	1058	MSR_MC8_ADDR	Package	
423H	1059	MSR_MC8_MISC	Package	

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
425H	1061	MSR_MC9_STATUS	Package	
426H	1062	MSR_MC9_ADDR	Package	
427H	1063	MSR_MC9_MISC	Package	
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
429H	1065	MSR_MC10_STATUS	Package	
42AH	1066	MSR_MC10_ADDR	Package	
42BH	1067	MSR_MC10_MISC	Package	
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
42DH	1069	MSR_MC11_STATUS	Package	
42EH	1070	MSR_MC11_ADDR	Package	
42FH	1071	MSR_MC11_MISC	Package	
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
431H	1073	MSR_MC12_STATUS	Package	
432H	1074	MSR_MC12_ADDR	Package	
433H	1075	MSR_MC12_MISC	Package	
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
435H	1077	MSR_MC13_STATUS	Package	
436H	1078	MSR_MC13_ADDR	Package	
437H	1079	MSR_MC13_MISC	Package	
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
439H	1081	MSR_MC14_STATUS	Package	
43AH	1082	MSR_MC14_ADDR	Package	
43BH	1083	MSR_MC14_MISC	Package	
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
43DH	1085	MSR_MC15_STATUS	Package	
43EH	1086	MSR_MC15_ADDR	Package	
43FH	1087	MSR_MC15_MISC	Package	
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC error from each channel of the integrated memory controllers.
441H	1089	MSR_MC16_STATUS	Package	
442H	1090	MSR_MC16_ADDR	Package	
443H	1091	MSR_MC16_MISC	Package	

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC17 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.
445H	1093	MSR_MC17_STATUS	Package	
446H	1094	MSR_MC17_ADDR	Package	
447H	1095	MSR_MC17_MISC	Package	
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC18 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.
449H	1097	MSR_MC18_STATUS	Package	
44AH	1098	MSR_MC18_ADDR	Package	
44BH	1099	MSR_MC18_MISC	Package	
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC19 reports MC error from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.
44DH	1101	MSR_MC19_STATUS	Package	
44EH	1102	MSR_MC19_ADDR	Package	
44FH	1103	MSR_MC19_MISC	Package	
450H	1104	MSR_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs." through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC20 reports MC error from the Intel QPI 1 module.
451H	1105	MSR_MC20_STATUS	Package	
452H	1106	MSR_MC20_ADDR	Package	
453H	1107	MSR_MC20_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)**
		3:0	Package	**Power Units** See Section 14.9.1, "RAPL Interfaces."
		7:4	Package	**Reserved**
		12:8	Package	**Energy Status Units** Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules)
		15:13	Package	**Reserved**
		19:16	Package	**Time Units** See Section 14.9.1, "RAPL Interfaces."
		63:20		Reserved
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	**Indicator of Frequency Clipping in Processor Cores (R/W) (frequency refers to processor core frequency)**
		0		**PROCHOT Status (R0)** When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.
		1		**Thermal Status (R0)** When set, frequency is reduced below the operating system request due to a thermal event.

Table 35-25 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		5:2		Reserved.
		6		**VR Therm Alert Status (R0)** When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.
		7		Reserved.
		8		**Electrical Design Point Status (R0)** When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g. maximum electrical current consumption).
		63:9		Reserved.
C8DH	3213	IA32_QM_EVTSEL	THREAD	**Monitoring Event Select Register (R/W).** if CPUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] = 1
		7:0		**EventID (RW)** **Event encoding:** **0x0: no monitoring** **0x1: L3 occupancy monitoring** **all other encoding reserved.**
		31:8		Reserved.
		41:32		**RMID (RW)**
		63:42		Reserved.
C8EH	3214	IA32_QM_CTR	THREAD	**Monitoring Counter Register (R/O).** if CPUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] = 1
		61:0		**Resource Monitored Data**
		62		**Unavailable**: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.
		63		**Error:** If 1, indicates and unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.
C8FH	3215	IA32_PQR_ASSOC	THREAD	**Resource Association Register (R/W).**
		9:0		**RMID**
		63: 10		**Reserved**
See Table 35-16, Table 35-20, Table 35-22 for other MSR definitions applicable to processors with CPUID signature 06_3FH				

35.12 MSRS IN INTEL® CORE™ M PROCESSORS AND 5TH GENERATION INTEL CORE PROCESSORS

The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors are based on the Broadwell microarchitecture, with CPUID DisplayFamily_DisplayModel signature 06_3DH, supports the MSR interfaces listed in Table 35-16, Table 35-17, Table 35-19, Table 35-22, Table 35-23, Table 35-26, and Table 35-27.

Table 35-26 lists MSRs that are common to processors based on the Broadwell microarchitectures (including Intel Core M processors, 5th Generation Intel Core processors, future generation of Intel Xeon processor D family and Intel Xeon processors).

Table 35-26 Additional MSRs Supported by Processors Based the Broadwell Microarchitectures

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
38EH	910	IA32_PERF_GLOBAL_ STAUS	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0		**Ovf_PMC0**

Table 35-26 Additional MSRs Supported by Processors Based the Broadwell Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		1		**Ovf_PMC1**
		2		**Ovf_PMC2**
		3		**Ovf_PMC3**
		31:4		Reserved.
		32		**Ovf_FixedCtr0**
		33		**Ovf_FixedCtr1**
		34		**Ovf_FixedCtr2**
		54:35		Reserved.
		55		**Trace_ToPA_PMI. See** Section 36.2.4.1, "Table of Physical Addresses (ToPA)."
		60:56		Reserved.
		61		**Ovf_Uncore**
		62		**Ovf_BufDSSAVE**
		63		**CondChgd**
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0		**Set 1 to clear Ovf_PMC0**
		1		**Set 1 to clear Ovf_PMC1**
		2		**Set 1 to clear Ovf_PMC2**
		3		**Set 1 to clear Ovf_PMC3**
		31:4		Reserved.
		32		**Set 1 to clear Ovf_FixedCtr0**
		33		**Set 1 to clear Ovf_FixedCtr1**
		34		**Set 1 to clear Ovf_FixedCtr2**
		54:35		Reserved.
		55		**Set 1 to clear Trace_ToPA_PMI. See** Section 36.2.4.1, "Table of Physical Addresses (ToPA)."
		60:56		Reserved.
		61		**Set 1 to clear Ovf_Uncore**
		62		**Set 1 to clear Ovf_BufDSSAVE**
		63		**Set 1 to clear CondChgd**
560H	1376	IA32_RTIT_OUTPUT_BASE	THREAD	**Trace Output Base Register (R/W)**
		6:0		Reserved.
		MAXPHYADDR[1]-1:7		**Base physical address of 1st ToPA table.**
		63:MAXPHYADDR		Reserved.

Table 35-26 Additional MSRs Supported by Processors Based the Broadwell Microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	THREAD	**Trace Output Mask Pointers Register (R/W)**
		6:0		Reserved.
		31:7		**MaskOrTableOffset**
		63:32		**Output Offset.**
570H	1392	IA32_RTIT_CTL	Thread	**Trace Packet Control Register (R/W)**
		0		**TraceEn**
		1		Reserved, MBZ.
		2		**OS**
		3		**User**
		6:4		Reserved, MBZ
		7		**CR3 filter**
		8		**ToPA; writing 0 will #GP if also setting TraceEn**
		9		Reserved, MBZ
		10		**TSCEn**
		11		**DisRETC**
		12		Reserved, MBZ
		13		**Reserved; writing 0 will #GP if also setting TraceEn**
		63:14		Reserved, MBZ.
571H	1393	IA32_RTIT_STATUS	Thread	**Tracing Status Register (R/W)**
		0		Reserved, writes ignored.
		1		**ContexEn**, writes ignored.
		2		**TriggerEn**, writes ignored.
		3		Reserved
		4		**Error (R/W)**
		5		**Stopped**
		63:6		Reserved, MBZ.
572H	1394	IA32_RTIT_CR3_MATCH	THREAD	**Trace Filter CR3 Match Register (R/W)**
		4:0		Reserved
		63:5		CR3[63:5] value to match

NOTES:
1. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

Table 35-27 lists MSRs that are specific to Intel Core M processors and 5th Generation Intel Core Processors.

Table 35-27 Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		39:32	Package	**Maximum Ratio Limit for 5C** Maximum turbo ratio limit of 5core active.
		47:40	Package	**Maximum Ratio Limit for 6C** Maximum turbo ratio limit of 6core active.
		63:48		Reserved.
See Table 35-16, Table 35-17, Table 35-19, Table 35-22, Table 35-23, Table 35-26 for other MSR definitions applicable to processors with CPUID signature 06_3DH				

35.13 MSRS IN FUTURE GENERATION INTEL® XEON® PROCESSORS

The MSRs listed in Table 35-28 are available in future generation of Intel® Xeon® Processor D Product Family (CPUID DisplayFamily_DisplayModel = 06_56H). It is based on the Broadwell microarchitecture.

Table 35-28 also applies to future Intel Xeon processors based on the Broadwell microarchitecture (CPUID DisplayFamily_DisplayModel = 06_4FH).

Table 35-28 Additional MSRs Supported by Intel® Xeon® Processors D Family and Future Intel Xeon Processors Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status (R/W)** See Table 35-2.
		0		**Thermal status (RO)** See Table 35-2.

Table 35-28 Additional MSRs Supported by Intel® Xeon® Processors D Family and Future Intel Xeon Processors Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		1		**Thermal status log (R/WC0)** See Table 35-2.
		2		**PROTCHOT # or FORCEPR# status (RO)** See Table 35-2.
		3		**PROTCHOT # or FORCEPR# log (R/WC0)** See Table 35-2.
		4		**Critical Temperature status (RO)** See Table 35-2.
		5		**Critical Temperature status log (R/WC0)** See Table 35-2.
		6		**Thermal threshold #1 status (RO)** See Table 35-2.
		7		**Thermal threshold #1 log (R/WC0)** See Table 35-2.
		8		**Thermal threshold #2 status (RO)** See Table 35-2.
		9		**Thermal threshold #2 log (R/WC0)** See Table 35-2.
		10		**Power Limitation status (RO)** See Table 35-2.
		11		**Power Limitation log (R/WC0)** See Table 35-2.
		12		**Current Limit status (RO)** See Table 35-2.
		13		**Current Limit log (R/WC0)** See Table 35-2.
		14		**Cross Domain Limit status (RO)** See Table 35-2.
		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.

Table 35-28 Additional MSRs Supported by Intel® Xeon® Processors D Family and Future Intel Xeon Processors Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31		**Reading Valid (RO)** See Table 35-2.
		63:32		Reserved.
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, "Enabling HWP"
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, "Managing HWP"
		7:0		**Minimum Performance (R/W).**
		15:8		**Maximum Performance (R/W).**
		23:16		**Desired Performance (R/W).**
		63:24		Reserved.
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, "HWP Feedback"
C8DH	3213	IA32_QM_EVTSEL	THREAD	**Monitoring Event Select Register (R/W).** if CPUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] = 1
		7:0		**EventID (RW)** **Event encoding:** **0x00: no monitoring** **0x01: L3 occupancy monitoring** **0x02: Total memory bandwidth monitoring** **0x03: Local memory bandwidth monitoring** **all other encoding reserved.**
		31:8		Reserved.
		41:32		**RMID (RW)**
		63:42		Reserved.
C8FH	3215	IA32_PQR_ASSOC	THREAD	**Resource Association Register (R/W).**
		9:0		**RMID**
		31:10		**Reserved**
		51:32		**COS (R/W).**
		63: 52		**Reserved**
C90H	3216	IA32_L3_QOS_MASK_0	Package	**L3 Class Of Service Mask - COS 0 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0
		0:19		**CBM: Bit vector of available L3 ways for COS 0 enforcement**
		63:20		**Reserved**
C91H	3217	IA32_L3_QOS_MASK_1	Package	**L3 Class Of Service Mask - COS 1 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1

Table 35-28 Additional MSRs Supported by Intel® Xeon® Processors D Family and Future Intel Xeon Processors Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0:19		**CBM: Bit vector of available L3 ways for COS 1 enforcement**
		63:20		**Reserved**
C92H	3218	IA32_L3_QOS_MASK_2	Package	**L3 Class Of Service Mask - COS 2 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2
		0:19		**CBM: Bit vector of available L3 ways for COS 2 enforcement**
		63:20		**Reserved**
C93H	3219	IA32_L3_QOS_MASK_3	Package	**L3 Class Of Service Mask - COS 3 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3
		0:19		**CBM: Bit vector of available L3 ways for COS 3 enforcement**
		63:20		**Reserved**
C94H	3220	IA32_L3_QOS_MASK_4	Package	**L3 Class Of Service Mask - COS 4 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4
		0:19		**CBM: Bit vector of available L3 ways for COS 4 enforcement**
		63:20		**Reserved**
C95H	3221	IA32_L3_QOS_MASK_5	Package	**L3 Class Of Service Mask - COS 5 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5
		0:19		**CBM: Bit vector of available L3 ways for COS 5 enforcement**
		63:20		**Reserved**
C96H	3222	IA32_L3_QOS_MASK_6	Package	**L3 Class Of Service Mask - COS 6 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6
		0:19		**CBM: Bit vector of available L3 ways for COS 6 enforcement**
		63:20		**Reserved**
C97H	3223	IA32_L3_QOS_MASK_7	Package	**L3 Class Of Service Mask - COS 7 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7
		0:19		**CBM: Bit vector of available L3 ways for COS 7 enforcement**
		63:20		**Reserved**
C98H	3224	IA32_L3_QOS_MASK_8	Package	**L3 Class Of Service Mask - COS 8 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8
		0:19		**CBM: Bit vector of available L3 ways for COS 8 enforcement**
		63:20		**Reserved**
C99H	3225	IA32_L3_QOS_MASK_9	Package	**L3 Class Of Service Mask - COS 9 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9
		0:19		**CBM: Bit vector of available L3 ways for COS 9 enforcement**
		63:20		**Reserved**

Table 35-28 Additional MSRs Supported by Intel® Xeon® Processors D Family and Future Intel Xeon Processors Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C9AH	3226	IA32_L3_QOS_MASK_10	Package	**L3 Class Of Service Mask - COS 10 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX >=10
		0:19		**CBM: Bit vector of available L3 ways for COS 10 enforcement**
		63:20		**Reserved**
C9BH	3227	IA32_L3_QOS_MASK_11	Package	**L3 Class Of Service Mask - COS 11 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX >=11
		0:19		**CBM: Bit vector of available L3 ways for COS 11 enforcement**
		63:20		**Reserved**
C9CH	3228	IA32_L3_QOS_MASK_12	Package	**L3 Class Of Service Mask - COS 12 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX >=12
		0:19		**CBM: Bit vector of available L3 ways for COS 12 enforcement**
		63:20		**Reserved**
C9DH	3229	IA32_L3_QOS_MASK_13	Package	**L3 Class Of Service Mask - COS 13 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX >=13
		0:19		**CBM: Bit vector of available L3 ways for COS 13 enforcement**
		63:20		**Reserved**
C9EH	3230	IA32_L3_QOS_MASK_14	Package	**L3 Class Of Service Mask - COS 14 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX >=14
		0:19		**CBM: Bit vector of available L3 ways for COS 14 enforcement**
		63:20		**Reserved**
C9FH	3231	IA32_L3_QOS_MASK_15	Package	**L3 Class Of Service Mask - COS 15 (R/W).** if CPUID.(EAX=10H, ECX=1):EDX.COS_MAX >=15
		0:19		**CBM: Bit vector of available L3 ways for COS 15 enforcement**
		63:20		**Reserved**

…

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 35-30. 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 35-30 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." andTable 35-2
10H	16	IA32_TIME_STAMP_ COUNTER	Thread	See Section 17.13, "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.
		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.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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)**
		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.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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**
		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)**

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
1A0	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.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		36:32	Package	**Number of Incremental Cores Added to Group 3** Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".
		39:37	Package	**Group Ratio Delta for Group 3** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.
		44:40	Package	**Number of Incremental Cores Added to Group 4** Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".
		47:45	Package	**Group Ratio Delta for Group 4** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.
		52:48	Package	**Number of Incremental Cores Added to Group 5** Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".
		55:53	Package	**Group Ratio Delta for Group 5** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.
		60:56	Package	**Number of Incremental Cores Added to Group 6** Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".
		63:61	Package	**Group Ratio Delta for Group 6** An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.
1B0H	432	IA32_ENERGY_PERF_BIAS	Thread	See Table 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.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 35-2.
277H	631	IA32_PAT	Core	See Table 35-2.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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)
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."

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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
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

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
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."
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 35-19
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O). See Table 35-19
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O). See Table 35-19
64BH	1611	MSR_CONFIG_TDP_ CONTROL	Package	**ConfigTDP Control (R/W)** See Table 35-19
64CH	1612	MSR_TURBO_ACTIVATION_ RATIO	Package	**ConfigTDP Control (R/W)** See Table 35-19
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.

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
		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)
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)

Table 35-30 Selected MSRs Supported by Next Generation Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signature 06_57H

Address		Register Name	Scope	Bit Description
Hex	Dec			
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI register (W/O)
C000_0080H		IA32_EFER	Thread	**Extended Feature Enables** See Table 35-2.
C000_0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.
C000_0084H		IA32_FMASK	Thread	**System Call Flag Mask (R/W)** See Table 35-2.
C000_0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_0102H		IA32_KERNEL_GSBASE	Thread	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.
C000_0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature. (R/W)** See Table 35-2

…