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

Documentation Changes

April 2015

Document Number: 252046-046

Contents

Revision History

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

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

§

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 7, Volume 1
5	Updates to Chapter 13, Volume 1
6	Updates to Chapter 16, Volume 1
7	Updates to Chapter 17, Volume 1
8	Updates to Appendix A, Volume 1
9	Updates to Chapter 1, Volume 2A
10	Updates to Chapter 2, Volume 2A
11	Updates to Chapter 3, Volume 2A
12	Updates to Chapter 4, Volume 2B
13	Updates to Chapter 1, Volume 3A
14	Updates to Chapter 2, Volume 3A
15	Updates to Chapter 4, Volume 3A
16	Updates to Chapter 6, Volume 3A
17	Updates to Chapter 10, Volume 3A
18	Updates to Chapter 13, Volume 3A
19	Updates to Chapter 14, Volume 3B
20	Updates to Chapter 17, Volume 3B
21	Updates to Chapter 18, Volume 3B
22	Updates to Chapter 19, Volume 3B
23	Updates to Chapter 22, Volume 3B
24	Updates to Chapter 25, Volume 3C
25	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.

...

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

See also:

- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Fortran Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Software Development Tools:
  http://www.intel.com/cd/software/products/asmo-na/eng/index.htm
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in three or seven volumes):
  http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
  http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html
- Intel 64 Architecture x2APIC Specification:

  http://www.intel.com/content/www/us/en/architecture-and-technology/64-architecture-x2apic-specification.html
- Intel® Trusted Execution Technology Measured Launched Environment Programming Guide:

  http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html
- Developing Multi-threaded Applications: A Platform Consistent Approach:
  https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
  http://software.intel.com/en-us/articles/ap949-using-spin-loops-on-intel-pentiumr-4-processor-and-intel-xeonr-processor/
- Performance Monitoring Unit Sharing Guide
  http://software.intel.com/file/30388

Literature related to selected features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference
  https://software.intel.com/en-us/isa-extensions

- Intel® Software Guard Extensions (Intel® SGX) Programming Reference
  https://software.intel.com/en-us/isa-extensions/intel-sgx

More relevant links are:

- Intel® Developer Zone:

  https://software.intel.com/en-us
- Developer centers:

  http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html
- Processor support general link:

  http://www.intel.com/support/processors/
- Software products and packages:

  http://www.intel.com/cd/software/products/asmo-na/eng/index.htm
- Intel® Hyper-Threading Technology (Intel® HT Technology):

  http://www.intel.com/technology/platform-technology/hyper-threading/index.htm

...

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.

...

4.8.3.5 Operating on SNaNs and QNaNs

When a floating-point operation is performed on an SNaN and/or a QNaN, the result of the operation is either a QNaN delivered to the destination operand or the generation of a floating-point invalid operation exception, depending on the following rules:

- If one of the source operands is an SNaN and the floating-point invalid-operation exception is not masked (see Section 4.9.1.1, "Invalid Operation Exception (#I)"), then a floating-point invalid-operation exception is signaled and no result is stored in the destination operand.
- If either or both of the source operands are NaNs and floating-point invalid-operation exception is masked, the result is as shown in Table 4-7. When an SNaN is converted to a QNaN, the conversion is handled by setting the most-significant fraction bit of the SNaN to 1. Also, when one of the source operands is an SNaN, the floating-point invalid-operation exception flag is set. Note that for some combinations of source operands, the result is different for x87 FPU operations and for SSE/SSE2/SSE3/SSE4.1 operations. Intel AVX follows the same behavior as SSE/SSE2/SSE3/SSE4.1 in this respect.
- When neither of the source operands is a NaN, but the operation generates a floating-point invalid-operation exception (see Tables 8-10 and 11-1), the result is commonly a QNaN FP Indefinite (Section 4.8.3.7).

Any exceptions to the behavior described in Table 4-7 are described in Section 8.5.1.2, "Invalid Arithmetic Operand Exception (#IA)," and Section 11.5.2.1, "Invalid Operation Exception (#I)."

...

4.8.3.6 Using SNaNs and QNaNs in Applications

Except for the rules given at the beginning of Section 4.8.3.4, "NaNs," for encoding SNaNs and QNaNs, software is free to use the bits in the significand of a NaN for any purpose. Both SNaNs and QNaNs can be encoded to carry and store data, such as diagnostic information.

By unmasking the invalid operation exception, the programmer can use signaling NaNs to trap to the exception handler. The generality of this approach and the large number of NaN values that are available provide the sophisticated programmer with a tool that can be applied to a variety of special situations.

For example, a compiler can use signaling NaNs as references to uninitialized (real) array elements. The compiler can preinitialize each array element with a signaling NaN whose significand contains the index (relative position) of the element. Then, if an application program attempts to access an element that it has not initialized, it can use the NaN placed there by the compiler. If the invalid operation exception is unmasked, an interrupt will occur, and the exception handler will be invoked. The exception handler can determine which element has been accessed, since the operand address field of the exception pointer will point to the NaN, and the NaN will contain the index number of the array element.

Quiet NaNs are often used to speed up debugging. In its early testing phase, a program often contains multiple errors. An exception handler can be written to save diagnostic information in memory whenever it is invoked. After storing the diagnostic data, it can supply a quiet NaN as the result of the erroneous instruction, and that NaN can point to its associated diagnostic area in memory. The program will then continue, creating a different NaN for each error. When the program ends, the NaN results can be used to access the diagnostic data saved at the time the errors occurred. Many errors can thus be diagnosed and corrected in one test run.

In embedded applications that use computed results in further computations, an undetected QNaN can invalidate all subsequent results. Such applications should therefore periodically check for QNaNs and provide a recovery mechanism to be used if a QNaN result is detected.

...

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.

...

5.12 AESNI AND PCLMULQDQ

Six AESNI instructions operate on XMM registers to provide accelerated primitives for block encryption/decryption using Advanced Encryption Standard (FIPS-197). The PCLMULQDQ instruction performs carry-less multiplication for two binary numbers up to 64-bit wide.

AESDEC	Perform an AES decryption round using an 128-bit state and a round key
AESDECLAST	Perform the last AES decryption round using an 128-bit state and a round key
AESENC	Perform an AES encryption round using an 128-bit state and a round key
AESENCLAST	Perform the last AES encryption round using an 128-bit state and a round key
AESIMC	Perform an inverse mix column transformation primitive
AESKEYGENASSIST	Assist the creation of round keys with a key expansion schedule
PCLMULQDQ	Perform carryless multiplication of two 64-bit numbers

...

5.19 64-BIT MODE INSTRUCTIONS

The following instructions are introduced in 64-bit mode. This mode is a sub-mode of IA-32e mode.

CDQE	Convert doubleword to quadword
CMPSQ	Compare string operands
CMPXCHG16B	Compare RDX:RAX with m128
LODSQ	Load qword at address (R)SI into RAX
MOVSQ	Move qword from address (R)SI to (R)DI
MOVZX (64-bits)	Move bytes/words to doublewords/quadwords, zero-extension
STOSQ	Store RAX at address RDI
SWAPGS	Exchanges current GS base register value with value in MSR address C0000102H
SYSCALL	Fast call to privilege level 0 system procedures
SYSRET	Return from fast system call

...

4. Updates to Chapter 7, Volume 1

Change bars show changes to Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

...

7.3.1.2 Exchange Instructions

The exchange instructions swap the contents of one or more operands and, in some cases, perform additional operations such as asserting the LOCK signal or modifying flags in the EFLAGS register.

The XCHG (exchange) instruction swaps the contents of two operands. This instruction takes the place of three MOV instructions and does not require a temporary location to save the contents of one operand location while the other is being loaded. When a memory operand is used with the XCHG instruction, the processor's LOCK signal is automatically asserted. This instruction is thus useful for implementing semaphores or similar data structures for process synchronization. See "Bus Locking" in Chapter 8, "Multiple-Processor Management,"of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for more information on bus locking.

The BSWAP (byte swap) instruction reverses the byte order in a 32-bit register operand. Bit positions 0 through 7 are exchanged with 24 through 31, and bit positions 8 through 15 are exchanged with 16 through 23. Executing this instruction twice in a row leaves the register with the same value as before. The BSWAP instruction is useful for converting between "big-endian" and "little-endian" data formats. This instruction also speeds execution of decimal arithmetic. (The XCHG instruction can be used to swap the bytes in a word.)

Table 7-2 Conditional Move Instructions

Instruction Mnemonic	Status Flag States	Condition Description
Unsigned Conditional Moves		
CMOVA/CMOVNBE	(CF or ZF) = 0	Above/not below or equal
CMOVAE/CMOVNB	CF = 0	Above or equal/not below
CMOVNC	CF = 0	Not carry
CMOVB/CMOVNAE	CF = 1	Below/not above or equal

Table 7-2 Conditional Move Instructions (Contd.)

CMOVC	CF = 1	Carry
CMOVBE/CMOVNA	(CF or ZF) = 1	Below or equal/not above
CMOVE/CMOVZ	ZF = 1	Equal/zero
CMOVNE/CMOVNZ	ZF = 0	Not equal/not zero
CMOVP/CMOVPE	PF = 1	Parity/parity even
CMOVNP/CMOVPO	PF = 0	Not parity/parity odd
Signed Conditional Moves		
CMOVGE/CMOVNL	(SF xor OF) = 0	Greater or equal/not less
CMOVL/CMOVNGE	(SF xor OF) = 1	Less/not greater or equal
CMOVLE/CMOVNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
CMOVO	OF = 1	Overflow
CMOVNO	OF = 0	Not overflow
CMOVS	SF = 1	Sign (negative)
CMOVNS	SF = 0	Not sign (non-negative)

The XADD (exchange and add) instruction swaps two operands and then stores the sum of the two operands in the destination operand. The status flags in the EFLAGS register indicate the result of the addition. This instruction can be combined with the LOCK prefix (see “LOCK—Assert LOCK# Signal Prefix” in Chapter 3, “Instruction Set Reference, A-M,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A)* in a multiprocessing system to allow multiple processors to execute one DO loop.

The CMPXCHG (compare and exchange) and CMPXCHG8B (compare and exchange 8 bytes) instructions are used to synchronize operations in systems that use multiple processors. The CMPXCHG instruction requires three operands: a source operand in a register, another source operand in the EAX register, and a destination operand. If the values contained in the destination operand and the EAX register are equal, the destination operand is replaced with the value of the other source operand (the value not in the EAX register). Otherwise, the original value of the destination operand is loaded in the EAX register. The status flags in the EFLAGS register reflect the result that would have been obtained by subtracting the destination operand from the value in the EAX register.

The CMPXCHG instruction is commonly used for testing and modifying semaphores. It checks to see if a semaphore is free. If the semaphore is free, it is marked allocated; otherwise it gets the ID of the current owner. This is all done in one uninterruptible operation. In a single-processor system, the CMPXCHG instruction eliminates the need to switch to protection level 0 (to disable interrupts) before executing multiple instructions to test and modify a semaphore.

For multiple processor systems, CMPXCHG can be combined with the LOCK prefix to perform the compare and exchange operation atomically. (See “Locked Atomic Operations” in Chapter 8, “Multiple-Processor Management,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*, for more information on atomic operations.)

The CMPXCHG8B instruction also requires three operands: a 64-bit value in EDX:EAX, a 64-bit value in ECX:EBX, and a destination operand in memory. The instruction compares the 64-bit value in the EDX:EAX registers with the destination operand. If they are equal, the 64-bit value in the ECX:EBX registers is stored in the destination operand. If the EDX:EAX registers and the destination are not equal, the destination is loaded in the EDX:EAX registers. The CMPXCHG8B instruction can be combined with the LOCK prefix to perform the operation atomically.

...

7.3.2 Binary Arithmetic Instructions

Binary arithmetic instructions operate on 8-, 16-, and 32-bit numeric data encoded as signed or unsigned binary integers. The binary arithmetic instructions may also be used in algorithms that operate on decimal (BCD) values.

For the purpose of this discussion, these instructions are divided into subordinate subgroups of instructions that:

- Add and subtract
- Increment and decrement
- Compare and change signs
- Multiply and divide

...

7.3.2.4 Comparison and Sign Change Instructions

The CMP (compare) instruction computes the difference between two integer operands and updates the OF, SF, ZF, AF, PF, and CF flags according to the result. The source operands are not modified, nor is the result saved. The CMP instruction is commonly used in conjunction with a J*cc* (jump) or SET*cc* (byte set on condition) instruction, with the latter instructions performing an action based on the result of a CMP instruction.

The NEG (negate) instruction subtracts a signed integer operand from zero. The effect of the NEG instruction is to change the sign of a two's complement operand while keeping its magnitude.

7.3.2.5 Multiplication and Division Instructions

The processor provides two multiply instructions, MUL (unsigned multiply) and IMUL (signed multiply), and two divide instructions, DIV (unsigned divide) and IDIV (signed divide).

The MUL instruction multiplies two unsigned integer operands. The result is computed to twice the size of the source operands (for example, if word operands are being multiplied, the result is a doubleword).

The IMUL instruction multiplies two signed integer operands. The result is computed to twice the size of the source operands; however, in some cases the result is truncated to the size of the source operands (see “IMUL—Signed Multiply” in Chapter 3, “Instruction Set Reference, A-M,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A)*.

The DIV instruction divides one unsigned operand by another unsigned operand and returns a quotient and a remainder.

The IDIV instruction is identical to the DIV instruction, except that IDIV performs a signed division.

...

7.3.3 Decimal Arithmetic Instructions

Decimal arithmetic can be performed by combining the binary arithmetic instructions ADD, SUB, MUL, and DIV (discussed in Section 7.3.2, “Binary Arithmetic Instructions”) with the decimal arithmetic instructions. The decimal arithmetic instructions are provided to carry out the following operations:

- To adjust the results of a previous binary arithmetic operation to produce a valid BCD result.
- To adjust the operands of a subsequent binary arithmetic operation so that the operation will produce a valid BCD result.

These instructions operate on both packed and unpacked BCD values. For the purpose of this discussion, the decimal arithmetic instructions are divided into subordinate subgroups of instructions that provide:

- Packed BCD adjustments

- Unpacked BCD adjustments

...

7.3.3.2 Unpacked BCD Adjustment Instructions

The AAA (ASCII adjust after addition), AAS (ASCII adjust after subtraction), AAM (ASCII adjust after multiplication), and AAD (ASCII adjust before division) instructions adjust the results of arithmetic operations performed on unpacked BCD values (see Section 4.7, "BCD and Packed BCD Integers"). All these instructions assume that the value to be adjusted is stored in the AL register or, in one instance, the AL and AH registers.

The AAA instruction adjusts the contents of the AL register following the addition of two unpacked BCD values. It converts the binary value in the AL register into a decimal value and stores the result in the AL register in unpacked BCD format (the decimal number is stored in the lower 4 bits of the register and the upper 4 bits are cleared). If a decimal carry occurred as a result of the addition, the CF flag is set and the contents of the AH register are incremented by 1.

The AAS instruction adjusts the contents of the AL register following the subtraction of two unpacked BCD values. Here again, a binary value is converted into an unpacked BCD value. If a borrow was required to complete the decimal subtract, the CF flag is set and the contents of the AH register are decremented by 1.

The AAM instruction adjusts the contents of the AL register following a multiplication of two unpacked BCD values. It converts the binary value in the AL register into a decimal value and stores the least significant digit of the result in the AL register (in unpacked BCD format) and the most significant digit, if there is one, in the AH register (also in unpacked BCD format).

The AAD instruction adjusts a two-digit BCD value so that when the value is divided with the DIV instruction, a valid unpacked BCD result is obtained. The instruction converts the BCD value in registers AH (most significant digit) and AL (least significant digit) into a binary value and stores the result in register AL. When the value in AL is divided by an unpacked BCD value, the quotient and remainder will be automatically encoded in unpacked BCD format.

7.3.4 Decimal Arithmetic Instructions in 64-Bit Mode

Decimal arithmetic instructions are not supported in 64-bit mode, they are either invalid or not encodable.

...

7.3.6 Shift and Rotate Instructions

The shift and rotate instructions rearrange the bits within an operand. For the purpose of this discussion, these instructions are further divided into subordinate subgroups of instructions that:

- Shift bits
- Double-shift bits (move them between operands)
- Rotate bits

...

7.3.7 Bit and Byte Instructions

These instructions operate on bit or byte strings. For the purpose of this discussion, they are further divided into subordinate subgroups that:

- Test and modify a single bit

- Scan a bit string
- Set a byte given conditions
- Test operands and report results

...

7.3.8 Control Transfer Instructions

The processor provides both conditional and unconditional control transfer instructions to direct the flow of program execution. Conditional transfers are taken only for specified states of the status flags in the EFLAGS register. Unconditional control transfers are always executed.

For the purpose of this discussion, these instructions are further divided into subordinate subgroups that process:

- Unconditional transfers
- Conditional transfers
- Software interrupts

7.3.8.1 Unconditional Transfer Instructions

The JMP, CALL, RET, INT, and IRET instructions transfer program control to another location (destination address) in the instruction stream. The destination can be within the same code segment (near transfer) or in a different code segment (far transfer).

Jump instruction — The JMP (jump) instruction unconditionally transfers program control to a destination instruction. The transfer is one-way; that is, a return address is not saved. A destination operand specifies the address (the instruction pointer) of the destination instruction. The address can be a **relative address** or an **absolute address**.

A **relative address** is a displacement (offset) with respect to the address in the EIP register. The destination address (a near pointer) is formed by adding the displacement to the address in the EIP register. The displacement is specified with a signed integer, allowing jumps either forward or backward in the instruction stream.

An **absolute address** is a offset from address 0 of a segment. It can be specified in either of the following ways:

- **An address in a general-purpose register —** This address is treated as a near pointer, which is copied into the EIP register. Program execution then continues at the new address within the current code segment.
- **An address specified using the standard addressing modes of the processor —** Here, the address can be a near pointer or a far pointer. If the address is for a near pointer, the address is translated into an offset and copied into the EIP register. If the address is for a far pointer, the address is translated into a segment selector (which is copied into the CS register) and an offset (which is copied into the EIP register).

In protected mode, the JMP instruction also allows jumps to a call gate, a task gate, and a task-state segment.

Call and return instructions — The CALL (call procedure) and RET (return from procedure) instructions allow a jump from one procedure (or subroutine) to another and a subsequent jump back (return) to the calling procedure.

The CALL instruction transfers program control from the current (or calling) procedure to another procedure (the called procedure). To allow a subsequent return to the calling procedure, the CALL instruction saves the current contents of the EIP register on the stack before jumping to the called procedure. The EIP register (prior to transferring program control) contains the address of the instruction following the CALL instruction. When this address is pushed on the stack, it is referred to as the **return instruction pointer** or **return address**.

The address of the called procedure (the address of the first instruction in the procedure being jumped to) is specified in a CALL instruction the same way as it is in a JMP instruction (see “Jump instruction” on page 2-16). The address can be specified as a relative address or an absolute address. If an absolute address is specified, it can be either a near or a far pointer.

The RET instruction transfers program control from the procedure currently being executed (the called procedure) back to the procedure that called it (the calling procedure). Transfer of control is accomplished by copying the return instruction pointer from the stack into the EIP register. Program execution then continues with the instruction pointed to by the EIP register.

The RET instruction has an optional operand, the value of which is added to the contents of the ESP register as part of the return operation. This operand allows the stack pointer to be incremented to remove parameters from the stack that were pushed on the stack by the calling procedure.

See Section 6.3, “Calling Procedures Using CALL and RET,” for more information on the mechanics of making procedure calls with the CALL and RET instructions.

Return from interrupt instruction — When the processor services an interrupt, it performs an implicit call to an interrupt-handling procedure. The IRET (return from interrupt) instruction returns program control from an interrupt handler to the interrupted procedure (that is, the procedure that was executing when the interrupt occurred). The IRET instruction performs a similar operation to the RET instruction (see “Call and return instructions” on page 2-16) except that it also restores the EFLAGS register from the stack. The contents of the EFLAGS register are automatically stored on the stack along with the return instruction pointer when the processor services an interrupt.

7.3.8.2 Conditional Transfer Instructions

The conditional transfer instructions execute jumps or loops that transfer program control to another instruction in the instruction stream if specified conditions are met. The conditions for control transfer are specified with a set of condition codes that define various states of the status flags (CF, ZF, OF, PF, and SF) in the EFLAGS register.

Conditional jump instructions — The J*cc* (conditional) jump instructions transfer program control to a destination instruction if the conditions specified with the condition code (*cc*) associated with the instruction are satisfied (see Table 7-4). If the condition is not satisfied, execution continues with the instruction following the J*cc* instruction. As with the JMP instruction, the transfer is one-way; that is, a return address is not saved.

Table 7-4 Conditional Jump Instructions

Instruction Mnemonic	Condition (Flag States)	Description
Unsigned Conditional Jumps		
JA/JNBE	(CF or ZF) = 0	Above/not below or equal
JAE/JNB	CF = 0	Above or equal/not below
JB/JNAE	CF = 1	Below/not above or equal
JBE/JNA	(CF or ZF) = 1	Below or equal/not above
JC	CF = 1	Carry
JE/JZ	ZF = 1	Equal/zero
JNC	CF = 0	Not carry
JNE/JNZ	ZF = 0	Not equal/not zero
JNP/JPO	PF = 0	Not parity/parity odd
JP/JPE	PF = 1	Parity/parity even
JCXZ	CX = 0	Register CX is zero
JECXZ	ECX = 0	Register ECX is zero

Table 7-4 Conditional Jump Instructions (Contd.)

Instruction Mnemonic	Condition (Flag States)	Description
Signed Conditional Jumps		
JG/JNLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
JGE/JNL	(SF xor OF) = 0	Greater or equal/not less
JL/JNGE	(SF xor OF) = 1	Less/not greater or equal
JLE/JNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
JNO	OF = 0	Not overflow
JNS	SF = 0	Not sign (non-negative)
JO	OF = 1	Overflow
JS	SF = 1	Sign (negative)

The destination operand specifies a relative address (a signed offset with respect to the address in the EIP register) that points to an instruction in the current code segment. The J*cc* instructions do not support far transfers; however, far transfers can be accomplished with a combination of a J*cc* and a JMP instruction (see "J*cc*—Jump if Condition Is Met" in Chapter 3, "Instruction Set Reference, A-M," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A)*.

Table 7-4 shows the mnemonics for the J*cc* instructions and the conditions being tested for each instruction. The condition code mnemonics are appended to the letter "J" to form the mnemonic for a J*cc* instruction. The instructions are divided into two groups: unsigned and signed conditional jumps. These groups correspond to the results of operations performed on unsigned and signed integers respectively. Those instructions listed as pairs (for example, JA/JNBE) are alternate names for the same instruction. Assemblers provide alternate names to make it easier to read program listings.

The JCXZ and JECXZ instructions test the CX and ECX registers, respectively, instead of one or more status flags. See "Jump if zero instructions" on page 7-17 for more information about these instructions.

Loop instructions — The LOOP, LOOPE (loop while equal), LOOPZ (loop while zero), LOOPNE (loop while not equal), and LOOPNZ (loop while not zero) instructions are conditional jump instructions that use the value of the ECX register as a count for the number of times to execute a loop. All the loop instructions decrement the count in the ECX register each time they are executed and terminate a loop when zero is reached. The LOOPE, LOOPZ, LOOPNE, and LOOPNZ instructions also accept the ZF flag as a condition for terminating the loop before the count reaches zero.

The LOOP instruction decrements the contents of the ECX register (or the CX register, if the address-size attribute is 16), then tests the register for the loop-termination condition. If the count in the ECX register is non-zero, program control is transferred to the instruction address specified by the destination operand. The destination operand is a relative address (that is, an offset relative to the contents of the EIP register), and it generally points to the first instruction in the block of code that is to be executed in the loop. When the count in the ECX register reaches zero, program control is transferred to the instruction immediately following the LOOP instruction, which terminates the loop. If the count in the ECX register is zero when the LOOP instruction is first executed, the register is pre-decremented to FFFFFFFFH, causing the loop to be executed 2^{32} times.

The LOOPE and LOOPZ instructions perform the same operation (they are mnemonics for the same instruction). These instructions operate the same as the LOOP instruction, except that they also test the ZF flag.

If the count in the ECX register is not zero and the ZF flag is set, program control is transferred to the destination operand. When the count reaches zero or the ZF flag is clear, the loop is terminated by transferring program control to the instruction immediately following the LOOPE/LOOPZ instruction.

The LOOPNE and LOOPNZ instructions (mnemonics for the same instruction) operate the same as the LOOPE/LOOPZ instructions, except that they terminate the loop if the ZF flag is set.

Jump if zero instructions — The JECXZ (jump if ECX zero) instruction jumps to the location specified in the destination operand if the ECX register contains the value zero. This instruction can be used in combination with a loop instruction (LOOP, LOOPE, LOOPZ, LOOPNE, or LOOPNZ) to test the ECX register prior to beginning a loop. As described in "Loop instructions" on page 2-18, the loop instructions decrement the contents of the ECX register before testing for zero. If the value in the ECX register is zero initially, it will be decremented to FFFFFFFFH on the first loop instruction, causing the loop to be executed 2^{32} times. To prevent this problem, a JECXZ instruction can be inserted at the beginning of the code block for the loop, causing a jump out of the loop if the ECX register count is initially zero. When used with repeated string scan and compare instructions, the JECXZ instruction can determine whether the loop terminated because the count reached zero or because the scan or compare conditions were satisfied.

The JCXZ (jump if CX is zero) instruction operates the same as the JECXZ instruction when the 16-bit address-size attribute is used. Here, the CX register is tested for zero.

...

7.3.9.2 Repeated String Operations

Each of the string instructions described in Section 7.3.9.1 perform one iteration of a string operation. To operate on strings longer than a doubleword, the string instructions can be combined with a repeat prefix (REP) to create a repeating instruction or be placed in a loop.

When used in string instructions, the ESI and EDI registers are automatically incremented or decremented after each iteration of an instruction to point to the next element (byte, word, or doubleword) in the string. String operations can thus begin at higher addresses and work toward lower ones, or they can begin at lower addresses and work toward higher ones. The DF flag in the EFLAGS register controls whether the registers are incremented (DF = 0) or decremented (DF = 1). The STD and CLD instructions set and clear this flag, respectively.

The following repeat prefixes can be used in conjunction with a count in the ECX register to cause a string instruction to repeat:

- **REP** — Repeat while the ECX register not zero.
- **REPE / REPZ** — Repeat while the ECX register not zero and the ZF flag is set.
- **REPNE / REPNZ** — Repeat while the ECX register not zero and the ZF flag is clear.

When a string instruction has a repeat prefix, the operation executes until one of the termination conditions specified by the prefix is satisfied. The REPE/REPZ and REPNE/REPNZ prefixes are used only with the CMPS and SCAS instructions. Also, note that a REP STOS instruction is the fastest way to initialize a large block of memory.

7.3.9.3 Fast-String Operation

To improve performance, more recent processors support modifications to the processor's operation during the string store operations initiated with the MOVS, MOVSB, STOS, and STOSB instructions. This optimized operation, called **fast-string operation**, is used when the execution of one of those instructions meets certain initial conditions (see below). Instructions using fast-string operation effectively operate on the string in groups that may include multiple elements of the native data size (byte, word, doubleword, or quadword). With fast-string operation, the processor recognizes interrupts and data breakpoints only on boundaries between these groups. Fast-string operation is used only if the source and destination addresses both use either the WB or WC memory types.

The initial conditions for fast-string operation are implementation-specific and may vary with the native string size. Examples of parameters that may impact the use of fast-string operation include the following:

- the alignment indicated in the EDI and ESI alignment registers;
- the address order of the string operation;
- the value of the initial operation counter (ECX); and
- the difference between the source and destination addresses.

NOTE

Initial conditions for fast-string operation in future Intel 64 or IA-32 processor families may differ from above. The *Intel® 64 and IA-32 Architectures Optimization Reference Manual* may contain model-specific information.

Software can disable fast-string operation by clearing the fast-string-enable bit (bit 0) of IA32_MISC_ENABLE MSR. However, Intel recommends that system software always enable fast-string operation.

When fast-string operation is enabled (because IA32_MISC_ENABLE[0] = 1), some processors may further enhance the operation of the REP MOVSB and REP STOSB instructions. A processor supports these enhancements if CPUID.(EAX=07H, ECX=0H):EBX[bit 9] is 1. The *Intel® 64 and IA-32 Architectures Optimization Reference Manual* may include model-specific recommendations for use of these enhancements.

The stores produced by fast-string operation may appear to execute out of order. Software dependent upon sequential store ordering should not use string operations for the entire data structure to be stored. Data and semaphores should be separated. Order-dependent code should write to a discrete semaphore variable after any string operations to allow correctly ordered data to be seen by all processors. Atomicity of load and store operations is guaranteed only for native data elements of the string with native data size, and only if they are included in a single cache line. See Section 8.2.4, "Fast-String Operation and Out-of-Order Stores" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.*

...

7.3.10 I/O Instructions

The IN (input from port to register), INS (input from port to string), OUT (output from register to port), and OUTS (output string to port) instructions move data between the processor's I/O ports and either a register or memory.

The register I/O instructions (IN and OUT) move data between an I/O port and the EAX register (32-bit I/O), the AX register (16-bit I/O), or the AL (8-bit I/O) register. The I/O port being read or written to is specified with an immediate operand or an address in the DX register.

The block I/O instructions (INS and OUTS) instructions move blocks of data (strings) between an I/O port and memory. These instructions operate similar to the string instructions (see Section 7.3.9, "String Operations"). The ESI and EDI registers are used to specify string elements in memory and the repeat prefix (REP) is used to repeat the instructions to implement block moves. The assembler recognizes the following alternate mnemonics for these instructions: INSB (input byte), INSW (input word), and INSD (input doubleword), and OUTSB (output byte), OUTSW (output word), and OUTSD (output doubleword).

The INS and OUTS instructions use an address in the DX register to specify the I/O port to be read or written to.

...

7.3.13 Flag Control (EFLAG) Instructions

The Flag Control (EFLAG) instructions allow the state of selected flags in the EFLAGS register to be read or modified. For the purpose of this discussion, these instructions are further divided into subordinate subgroups of instructions that manipulate:

- Carry and direction flags
- The EFLAGS register
- Interrupt flags

7.3.13.1 Carry and Direction Flag Instructions

The STC (set carry flag), CLC (clear carry flag), and CMC (complement carry flag) instructions allow the CF flag in the EFLAGS register to be modified directly. They are typically used to initialize the CF flag to a known state before an instruction that uses the flag in an operation is executed. They are also used in conjunction with the rotate-with-carry instructions (RCL and RCR).

The STD (set direction flag) and CLD (clear direction flag) instructions allow the DF flag in the EFLAGS register to be modified directly. The DF flag determines the direction in which index registers ESI and EDI are stepped when executing string processing instructions. If the DF flag is clear, the index registers are incremented after each iteration of a string instruction; if the DF flag is set, the registers are decremented.

...

7.3.13.3 Interrupt Flag Instructions

The STI (set interrupt flag) and CLI (clear interrupt flag) instructions allow the interrupt IF flag in the EFLAGS register to be modified directly. The IF flag controls the servicing of hardware-generated interrupts (those received at the processor's INTR pin). If the IF flag is set, the processor services hardware interrupts; if the IF flag is clear, hardware interrupts are masked.

The ability to execute these instructions depends on the operating mode of the processor and the current privilege level (CPL) of the program or task attempting to execute these instructions.

...

7.3.15 Segment Register Instructions

The processor provides a variety of instructions that address the segment registers of the processor directly. These instructions are only used when an operating system or executive is using the segmented or the real-address mode memory model.

For the purpose of this discussion, these instructions are divided into subordinate subgroups of instructions that allow:

- Segment-register load and store
- Far control transfers
- Software interrupt calls
- Handling of far pointers

...

7.3.15.2 Far Control Transfer Instructions

The JMP and CALL instructions (see Section 7.3.8, "Control Transfer Instructions") both accept a far pointer as a destination to transfer program control to a segment other than the segment currently being pointed to by the CS register. When a far call is made with the CALL instruction, the current values of the EIP and CS registers are both pushed on the stack.

The RET instruction (see "Call and return instructions" on page 2-16) can be used to execute a far return. Here, program control is transferred from a code segment that contains a called procedure back to the code segment that contained the calling procedure. The RET instruction restores the values of the CS and EIP registers for the calling procedure from the stack.

7.3.15.3 Software Interrupt Instructions

The software interrupt instructions INT, INTO, and IRET (see Section 7.3.8.4, “Software Interrupt Instructions”) can also call and return from interrupt and exception handler procedures that are located in a code segment other than the current code segment. With these instructions, however, the switching of code segments is handled transparently from the application program.

...

7.3.17.1 RDRAND

The RDRAND instruction returns a random number. All Intel processors that support the RDRAND instruction indicate the availability of the RDRAND instruction via reporting CPUID.01H:ECX.RDRAND[bit 30] = 1.

RDRAND returns random numbers that are supplied by a cryptographically secure, deterministic random bit generator DRBG. The DRBG is designed to meet the NIST SP 800-90A standard. The DRBG is re-seeded frequently from an on-chip non-deterministic entropy source to guarantee data returned by RDRAND is statistically uniform, non-periodic and non-deterministic.

In order for the hardware design to meet its security goals, the random number generator continuously tests itself and the random data it is generating. Runtime failures in the random number generator circuitry or statistically anomalous data occurring by chance will be detected by the self test hardware and flag the resulting data as being bad. In such extremely rare cases, the RDRAND instruction will return no data instead of bad data.

Under heavy load, with multiple cores executing RDRAND in parallel, it is possible, though unlikely, for the demand of random numbers by software processes/threads to exceed the rate at which the random number generator hardware can supply them. This will lead to the RDRAND instruction returning no data transitorily. The RDRAND instruction indicates the occurrence of this rare situation by clearing the CF flag.

The RDRAND instruction returns with the carry flag set (CF = 1) to indicate valid data is returned. It is recommended that software using the RDRAND instruction to get random numbers retry for a limited number of iterations while RDRAND returns CF=0 and complete when valid data is returned, indicated with CF=1. This will deal with transitory underflows. A retry limit should be employed to prevent a hard failure in the RNG (expected to be extremely rare) leading to a busy loop in software.

The intrinsic primitive for RDRAND is defined to address software’s need for the common cases (CF = 1) and the rare situations (CF = 0). The intrinsic primitive returns a value that reflects the value of the carry flag returned by the underlying RDRAND instruction. The example below illustrates the recommended usage of an RDRAND intrinsic in a utility function, a loop to fetch a 64 bit random value with a retry count limit of 10. A C implementation might be written as follows:

```
-------------------------------------------------------------------------------------------
#define SUCCESS 1
#define RETRY_LIMIT_EXCEEDED 0
#define RETRY_LIMIT 10

int get_random_64( unsigned __int 64 * arand)
{int i ;
    for ( i = 0; i < RETRY_LIMIT; i ++) {
        if(_rdrand64_step(arand) ) return SUCCESS;
    }
    return RETRY_LIMIT_EXCEEDED;
}
-------------------------------------------------------------------------------
```

...

5. Updates to Chapter 13, Volume 1

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

...

13.1 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.
- Bits 7:5 correspond to the three state components used for the additional register state used by Intel® Advanced Vector Extensions 512 (**AVX-512 state**):
  - — State component 5 is used for the 8 64-bit opmask registers k0–k7 (**opmask state**).
  - — State component 6 is used for the upper 256 bits of the registers ZMM0–ZMM15. These 16 256-bit values are denoted ZMM0_H–ZMM15_H (**ZMM_Hi256 state**).
  - — State component 7 is used for the 16 512-bit registers ZMM16–ZMM31 (**Hi16_ZMM state**).
- Bit 9 corresponds to the state component used for the protection-key feature's register PKRU (**PKRU state**). See Section 13.5.5.

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; AVX state is state component 2; AVX-512 state comprises state components 5–7; and PKRU state is state component 9.

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 13.7 through Section 13.10.

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 13.11 and Section 13.12.

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, and for PKRU 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 and use of protection keys, 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[2] = 1. Otherwise, any execution of an AVX instruction causes an invalid-opcode exception (#UD).

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

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

- XCR0[7:5] are associated with AVX-512 state (see Section 13.5.4). Software can use the XSAVE feature set to manage AVX-512 state only if XCR0[7:5] = 111b. In addition, software can execute AVX-512 instructions only if CR4.OSXSAVE = 1 and XCR0[7:5] = 111b. Otherwise, any execution of an AVX-512 instruction causes an invalid-opcode exception (#UD).

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

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

- XCR0[9] is associated with PKRU state (see Section 13.5.5). Software can use the XSAVE feature set to manage PKRU state only if XCR0[9] = 1. The value of XCR0[9] in no way determines whether software can use protection keys or execute other instructions that access PKRU state (these instructions can be executed even if XCR0[9] = 0).

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

- XCR0[63:10], XCR0[8], and XCR0[4:3] are reserved. Executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and any corresponding bit in EDX:EAX is not 0. These bits in XCR0 are all 0 coming out of RESET.

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

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

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

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

AVX-512 state and software can execute AVX-512 instructions. If XCR0[9] = 1, the XSAVE feature set can be used to manage PKRU state.

The IA32_XSS MSR (with MSR index DA0H) is zero coming out of RESET. If CR4.OSXSAVE = 1, CPUID.(EAX=0DH,ECX=1):EAX[3] = 1, and CPL = 0, executing the WRMSR instruction with ECX = DA0H writes the 64-bit value in EDX:EAX to the IA32_XSS MSR (EAX is written to IA32_XSS[31:0] and EDX to IA32_XSS[63:32]). 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* ≥ 2. The following state components are currently supported in the extended area: state component 2 contains AVX state; state components 5–7 contain AVX-512 state; and state component 9 contains PKRU state.

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* ≥ 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* ≥ 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, state component *i* is located at a byte offset $location_I$ from the base address of the XSAVE area, where $location_I$ is determined by the following items:
    - If XCOMP_BV[*j*] = 0 for every *j*, 2 ≤ *j* < *i*, $location_I$ is 576. (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 ≤ *j* < *i*, be the greatest value such that XCOMP_BV[*j*] = 1. Then $location_I$ is determined by the following values: $location_J$; $size_J$, as enumerated in CPUID.(EAX=0DH,ECX=*j*):EAX; and the value of $align_I$, as enumerated in CPUID.(EAX=0DH,ECX=*i*):ECX[1]:
      - — If $align_I = 0$, $location_I = location_J + size_J$. (This item implies that state component *i* is located immediately following the preceding state component whose bit is set in XCOMP_BV.)
      - — If $align_I = 1$, $location_I = \text{ceiling}(location_J + size_J, 64)$. (This item implies that state component *i* is located on the next 64-byte boundary following the preceding state component whose bit is set in XCOMP_BV.)

13.5 XSAVE-MANAGED STATE

The section provides details regarding how the XSAVE feature set interacts with the various XSAVE-managed state components.

Unless otherwise state, the state pertaining to a particular state component is saved beginning at byte 0 of the section of the XSAVE are corresponding to that state component.

...

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.[1]
- 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 modify these bytes; executions of XRSTOR and XRSTORS outside 64-bit mode do not update XMM8–XMM15. See Section 13.13.

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.

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.(EAX=0DH,ECX=2):EAX enumerates the size (in bytes) required for AVX state.

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 modify bytes 255:128; executions of XRSTOR and XRSTORS outside 64-bit mode do not update YMM8_H–YMM15_H. See Section 13.13.

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

1. 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 13.7 through Section 13.11 for details.

(XCR0[2] = 1). AVX instructions cannot be used unless the XSAVE feature set is enabled and has been configured to manage AVX state.

13.5.4 AVX-512 State

The register state used by the Intel® Advanced Vector Extensions 512 (AVX-512) comprises the MXCSR register, the 8 64-bit opmask registers k0–k7, and 32 512-bit vector registers called ZMM0–ZMM31. For each *i*, 0 <= *i* <= 15, the low 256 bits of register ZMM*i* is identical to the AVX register YMM*i*. Thus, the new state register state added by AVX comprises the following state components:

- The opmask registers, collective called **opmask state**.
- The upper 256 bits of the registers ZMM0–ZMM15. These 16 256-bit values are denoted ZMM0_H–ZMM15_H and are collectively called **ZMM_Hi256 state**.
- The 16 512-bit registers ZMM16–ZMM31, collectively called **Hi16_ZMM state**.

Together, these three state components compose **AVX-512 state**.

As noted in Section 13.1, the XSAVE feature set manages AVX-512 state as state components 5–7. Thus, AVX-512 state is located in the extended region of the XSAVE area (see Section 13.4.3). The following items detail how these state components are organized in this region:

- **Opmask state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=5):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 opmask state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=5):EAX enumerates the size (in bytes) required for opmask state.
- **ZMM_Hi256 state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=6):EBX enumerates the offset of the section of the extended region of the XSAVE area used for ZMM_Hi256 state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=6):EAX enumerates the size (in bytes) required for ZMM_Hi256 state.

  The XSAVE feature set partitions ZMM0_H–ZMM15_H in a manner similar to that used for the XMM registers (see Section 13.5.2). Bytes 255:0 of the ZMM_Hi256-state section are used for ZMM0_H–ZMM7_H. Bytes 511:256 are used for ZMM8_H–ZMM15_H, but they are used only in 64-bit mode. Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not modify bytes 511:256; executions of XRSTOR and XRSTORS outside 64-bit mode do not update ZMM8_H–ZMM15_H. See Section 13.13.
- **Hi16_ZMM state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=7):EBX enumerates the offset of the section of the extended region of the XSAVE area used for Hi16_ZMM state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=7):EAX enumerates the size (in bytes) required for Hi16_ZMM state.

  The XSAVE feature set accesses Hi16_ZMM state only in 64-bit mode. Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not modify the Hi16_ZMM section; executions of XRSTOR and XRSTORS outside 64-bit mode do not update ZMM16–ZMM31. See Section 13.13.

All three components of AVX-512 state are XSAVE-managed and the AVX-512 feature is XSAVE-enabled. The XSAVE feature set can operate on AVX-512 state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage AVX-512 state (XCR0[7:5] = 111b). AVX-512 instructions cannot be used unless the XSAVE feature set is enabled and has been configured to manage AVX-512 state.

13.5.5 PKRU State

The register state used by the protection-key feature (**PKRU state**) is the 32-bit PKRU register. As noted in Section 13.1, the XSAVE feature set manages PKRU state as state component 9. Thus, PKRU 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=9):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 PKRU state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=9):EAX enumerates the size (in bytes) required for PKRU state. The XSAVE feature set uses bytes 3:0 of the PK-state section for the PKRU register.

PKRU state is XSAVE-managed but the protection-key feature is not XSAVE-enabled. The XSAVE feature set can operate on PKRU state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage PKRU state (XCR0[9] = 1). Software can otherwise use protection keys and access PKRU state even if the XSAVE feature set is not enabled or has not been configured to manage PKRU state.

The value of the PKRU register determines the access right for user-mode linear addresses. (See Section 4.6, "Access Rights," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.) The access rights that pertain to an execution of the XRSTOR and XRSTORS instructions are determined by the value of the register before the execution and not by any value that the execution might load into the PKRU register.

13.6 PROCESSOR TRACKING OF XSAVE-MANAGED STATE

The XSAVEOPT, XSAVEC, and XSAVES instructions use two optimization to reduce the amount of data that they write to memory. They avoid writing data for any state component known to be in its initial configuration (the **init optimization**). In addition, if either XSAVEOPT or XSAVES is using the same XSAVE area as that used by the most recent execution of XRSTOR or XRSTORS, it may avoid writing data for any state component whose configuration is known not to have been modified since then (the **modified optimization**). (XSAVE does not use these optimizations, and XSAVEC does not use the modified optimization.) The operation of XSAVEOPT, XSAVEC, and XSAVES are described in more detail in Section 13.9 through Section 13.11.

A processor can support the init and modified optimizations with special hardware that tracks the state components that might benefit from those optimizations. Other implementations might not include such hardware; such a processor would always consider each such state component as not in its initial configuration and as modified since the last execution of XRSTOR or XRSTORS.

The following notation describes the state of the init and modified optimizations:

- XINUSE denotes the state-component bitmap corresponding to the init optimization. If XINUSE[*i*] = 0, state component *i* is known to be in its initial configuration; otherwise XINUSE[*i*] = 1. It is possible for XINUSE[*i*] to be 1 even when state component *i* is in its initial configuration. On a processor that does not support the init optimization, XINUSE[*i*] is always 1 for every value of *i*.

  Executing XGETBV with ECX = 1 returns in EDX:EAX the logical-AND of XCR0 and the current value of the XINUSE state-component bitmap. Such an execution of XGETBV always sets EAX[1] to 1 if XCR0[1] = 1 and MXCSR does not have its RESET value of 1F80H. Section 13.2 explains how software can determine whether a processor supports this use of XGETBV.
- XMODIFIED denotes the state-component bitmap corresponding to the modified optimization. If XMODIFIED[*i*] = 0, state component *i* is known not to have been modified since the most recent execution of XRSTOR or XRSTORS; otherwise XMODIFIED[*i*] = 1. It is possible for XMODIFIED[*i*] to be 1 even when state component *i* has not been modified since the most recent execution of XRSTOR or XRSTORS. On a processor that does not support the modified optimization, XMODIFIED[*i*] is always 1 for every value of *i*.

A processor that implements the modified optimization saves information about the most recent execution of XRSTOR or XRSTORS in a quantity called **XRSTOR_INFO**, a 4-tuple containing the following: (1) the CPL; (2) whether the logical processor was in VMX non-root operation; (3) the linear address of the XSAVE area; and (4) the XCOMP_BV field in the XSAVE area. An execution of XSAVEOPT or XSAVES uses the modified optimization only if that execution corresponds to XRSTOR_INFO on these four parameters.

This mechanism implies that, depending on details of the operating system, the processor might determine that an execution of XSAVEOPT by one user application corresponds to an earlier execution of XRSTOR by a different application. For this reason, Intel recommends the application software not use the XSAVEOPT instruction.

The following items specify the initial configuration each state component (for the purposes of defining the XINUSE bitmap):

- **x87 state.** x87 state is in its initial configuration if the following all hold: FCW is 037FH; FSW is 0000H; FTW is FFFFH; FPU CS and FPU DS are each 0000H; FPU IP and FPU DP are each 00000000_00000000H; each of ST0–ST7 is 0000_00000000_00000000H.
- **SSE state.** In 64-bit mode, SSE state is in its initial configuration if each of XMM0–XMM15 is 0. Outside 64-bit mode, SSE state is in its initial configuration if each of XMM0–XMM7 is 0. XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update XMM8–XMM15. (See Section 13.13)
- **AVX state.** In 64-bit mode, AVX state is in its initial configuration if each of YMM0_H–YMM15_H is 0. Outside 64-bit mode, AVX state is in its initial configuration if each of YMM0_H–YMM7_H is 0. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update YMM8_H–YMM15_H. (See Section 13.13)
- **Opmask state.** Opmask state is in its initial configuration if each of the opmask registers k0–k7 is 0.
- **ZMM_Hi256 state.** In 64-bit mode, ZMM_Hi256 state is in its initial configuration if each of ZMM0_H–ZMM15_H is 0. Outside 64-bit mode, ZMM_Hi256 state is in its initial configuration if each of ZMM0_H–ZMM7_H is 0. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update ZMM8_H–ZMM15_H. (See Section 13.13)
- **Hi16_ZMM state.** In 64-bit mode, Hi16_ZMM state is in its initial configuration if each of ZMM16–ZMM31 is 0. Outside 64-bit mode, Hi16_ZMM state is always in its initial configuration. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update ZMM31–ZMM31. (See Section 13.13)
- **PKRU state.** PKRU state is in its initial configuration if the value of the PKRU is 0.

13.7 OPERATION OF XSAVE

The XSAVE 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**. The logical-AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the state components to be saved.

The following conditions cause execution of the XSAVE 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 the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

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

- If RFBM[*i*] = 0, XSTATE_BV[*i*] is not changed.
- If RFBM[*i*] = 1, XSTATE_BV[*i*] is set to the value of XINUSE[*i*]. Section 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. The nature of that optimization implies the following:
  - If state component *i* is in its initial configuration, XINUSE[*i*] may be either 0 or 1, and XSTATE_BV[*i*] may be written with either 0 or 1.

    XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. Thus, XSTATE_BV[1] may be written with 0 even if MXCSR does not have its RESET value of 1F80H.
  - If state component *i* is not in its initial configuration, XINUSE[*i*] = 1 and XSTATE_BV[*i*] is written with 1.

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.

(As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

The XSAVE instruction does not write any part of the XSAVE header other than the XSTATE_BV field; in particular, it does **not** write to the XCOMP_BV field.

Execution of XSAVE 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 XSAVE instruction always uses the standard format for the extended region (see Section 13.4.3).

The MXCSR register and MXCSR_MASK are part of SSE state (see Section 13.5.2) and are thus associated with RFBM[1]. However, the XSAVE instruction also saves these values when RFBM[2] = 1 (even if RFBM[1] = 0).

See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

13.8 OPERATION OF XRSTOR

The XRSTOR 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**. The logical-AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the state components to be restored.

The following conditions cause execution of the XRSTOR 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 the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

After checking for these faults, the XRSTOR instruction reads the XCOMP_BV field in the XSAVE area's XSAVE header (see Section 13.4.2). If XCOMP_BV[63] = 0, the **standard form of XRSTOR** is executed (see Section 13.8.1); otherwise, the **compacted form of XRSTOR** is executed (see Section 13.8.2).[2]

See Section 13.2 for details of how to determine whether the compacted form of XRSTOR is supported.

13.8.1 Standard Form of XRSTOR

The standard from of XRSTOR performs additional fault checking. Either of the following conditions causes a general-protection exception (#GP):

- The XSTATE_BV field of the XSAVE header sets a bit that is not set in XCR0.
- Bytes 23:8 of the XSAVE header are not all 0 (this implies that all bits in XCOMP_BV are 0).[3]

If none of these conditions cause a fault, the processor updates each state component *i* for which RFBM[*i*] = 1. XRSTOR updates state component *i* based on the value of bit *i* in the XSTATE_BV field of the XSAVE header:

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.
2. If the processor does not support the compacted form of XRSTOR, it may execute the standard form of XRSTOR without first reading the XCOMP_BV field. A processor supports the compacted form of XRSTOR only if it enumerates CPUID.(EAX=0DH,ECX=1):EAX[1] as 1.
3. Bytes 63:24 of the XSAVE header are also reserved. Software should ensure that bytes 63:16 of the XSAVE header are all 0 in any XSAVE area. (Bytes 15:8 should also be 0 if the XSAVE area is to be used on a processor that does not support the compaction extensions to the XSAVE feature set.)

- If XSTATE_BV[*i*] = 0, the state component is set to its initial configuration. Section 13.6 specifies the initial configuration of each state component.

  The initial configuration of state component 1 pertains only to the XMM registers and not to MXCSR. See below for the treatment of MXCSR
- If XSTATE_BV[*i*] = 1, the state component is loaded with data from the XSAVE area. See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

  State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \leq i \leq 62$, is located in the extended region; the standard form of XRSTOR uses the standard format for the extended region (see Section 13.4.3).

The MXCSR register is part of state component 1, SSE state (see Section 13.5.2). However, the standard form of XRSTOR loads the MXCSR register from memory whenever the RFBM[1] (SSE) or RFBM[2] (AVX) is set, regardless of the values of XSTATE_BV[1] and XSTATE_BV[2]. The standard form of XRSTOR causes a general-protection exception (#GP) if it would load MXCSR with an illegal value.

13.8.2 Compacted Form of XRSTOR

The compacted from of XRSTOR performs additional fault checking. Any of the following conditions causes a #GP:

- The XCOMP_BV field of the XSAVE header sets a bit in the range 62:0 that is not set in XCR0.
- The XSTATE_BV field of the XSAVE header sets a bit (including bit 63) that is not set in XCOMP_BV.
- Bytes 63:16 of the XSAVE header are not all 0.

If none of these conditions cause a fault, the processor updates each state component *i* for which RFBM[*i*] = 1. XRSTOR updates state component *i* based on the value of bit *i* in the XSTATE_BV field of the XSAVE header:

- If XSTATE_BV[*i*] = 0, the state component is set to its initial configuration. Section 13.6 specifies the initial configuration of each state component.

  If XSTATE_BV[1] = 0, the compacted form XRSTOR initializes MXCSR to 1F80H. (This differs from the standard from of XRSTOR, which loads MXCSR from the XSAVE area whenever either RFBM[1] or RFBM[2] is set.)

  State component *i* is set to its initial configuration as indicated above if RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0 — **even if XCOMP_BV[*i*] = 0**. This is true for all values of *i*, including 0 (x87 state) and 1 (SSE state).
- If XSTATE_BV[*i*] = 1, the state component is loaded with data from the XSAVE area.[1] See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

  State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \leq i \leq 62$, is located in the extended region; the compacted form of the XRSTOR instruction uses the compacted format for the extended region (see Section 13.4.3).

The MXCSR register is part of SSE state (see Section 13.5.2) and is thus loaded from memory if RFBM[1] = XSTATE_BV[*i*] = 1. The compacted form of XRSTOR does not consider RFBM[2] (AVX) when determining whether to update MXCSR. (This is a difference from the standard form of XRSTOR.) The compacted form of XRSTOR causes a general-protection exception (#GP) if it would load MXCSR with an illegal value.

1. Earlier fault checking ensured that, if the instruction has reached this point in execution and XSTATE_BV[*i*] is 1, then XCOMP_BV[*i*] is also 1.

13.8.3 XRSTOR and the Init and Modified Optimizations

Execution of the XRSTOR instruction causes the processor to update is tracking for the init and modified optimizations (see Section 13.6). The following items provide details:

- The processor updates its tracking for the init optimization as follows:
  - If RFBM[*i*] = 0, XINUSE[*i*] is not changed.
  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0, state component *i* may be tracked as init; XINUSE[*i*] may be set to 0 or 1. (As noted in Section 13.6, a processor need not implement the init optimization for state component *i*; a processor that does not do so implicitly maintains XINUSE[*i*] = 1 at all times.)
  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 1, state component *i* is tracked as not init; XINUSE[*i*] is set to 1.
- The processor updates its tracking for the modified optimization and records information about the XRSTOR execution for future interaction with the XSAVEOPT and XSAVES instructions (see Section 13.9 and Section 13.11) as follows:
  - If RFBM[*i*] = 0, state component *i* is tracked as modified; XMODIFIED[*i*] is set to 1.
  - If RFBM[*i*] = 1, state component *i* may be tracked as unmodified; XMODIFIED[*i*] may be set to 0 or 1. (As noted in Section 13.6, a processor need not implement the modified optimization for state component *i*; a processor that does not do so implicitly maintains XMODIFIED[*i*] = 1 at all times.)
  - XRSTOR_INFO is set to the 4-tuple $\langle w,x,y,z \rangle$, where w is the CPL (0); x is 1 if the logical processor is in VMX non-root operation and 0 otherwise; *y* is the linear address of the XSAVE area; and *z* is XCOMP_BV. In particular, the standard form of XRSTOR always sets *z* to all zeroes, while the compacted form of XRSTORS never does so (because it sets at least bit 63 to 1).

13.9 OPERATION OF XSAVEOPT

The operation of XSAVEOPT is similar to that of XSAVE. Unlike XSAVE, XSAVEOPT uses the init optimization (by which it may omit saving state components that are in their initial configuration) and the modified optimization (by which it may omit saving state components that have not been modified since the last execution of XRSTOR); see Section 13.6. See Section 13.2 for details of how to determine whether XSAVEOPT is supported.

The XSAVEOPT 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**. The logical (bitwise) AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the state components to be saved.

The following conditions cause execution of the XSAVEOPT 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 the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

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

- If RFBM[*i*] = 0, XSTATE_BV[*i*] is not changed.
- If RFBM[*i*] = 1, XSTATE_BV[*i*] is set to the value of XINUSE[*i*]. Section 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. The nature of that optimization implies the following:
  - If the state component is in its initial configuration, XINUSE[*i*] may be either 0 or 1, and XSTATE_BV[*i*] may be written with either 0 or 1.

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.

XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. Thus, XSTATE_BV[1] may be written with 0 even if MXCSR does not have its RESET value of 1F80H.

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

(As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

The XSAVEOPT instruction does not write any part of the XSAVE header other than the XSTATE_BV field; in particular, it does not write to the XCOMP_BV field.

Execution of XSAVEOPT saves into the XSAVE area those state components corresponding to bits that are set in RFBM (subject to the optimizations described below). 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 XSAVEOPT instruction always uses the standard 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. See Section 13.13 for details regarding faults caused by memory accesses.

Execution of XSAVEOPT performs two optimizations that reduce the amount of data written to memory:

- **Init optimization.**
  If XINUSE[*i*] = 0, state component *i* is not saved to the XSAVE area (even if RFBM[*i*] = 1). (See below for exceptions made for MXCSR.)
- **Modified optimization.**
  Each execution of XRSTOR and XRSTORS establishes XRSTOR_INFO as a 4-tuple $\langle w,x,y,z \rangle$ (see Section 13.8.3 and Section 13.12). Execution of XSAVEOPT uses the modified optimization only if the following all hold for the current value of XRSTOR_INFO:

  — *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* is 00000000_00000000H. (This last item implies that XSAVEOPT does not use the modified optimization if the last execution of XRSTOR used the compacted form, or if an execution of XRSTORS followed the last execution of XRSTOR.)

  If XSAVEOPT uses the modified optimization and XMODIFIED[*i*] = 0 (see Section 13.6), state component *i* is not saved to the XSAVE area.

  (In practice, the benefit of the modified optimization for state component *i* depends on how the processor is tracking state component *i*; see Section 13.6. Limitations on the tracking ability may result in state component *i* being saved even though is in the same configuration that was loaded by the previous execution of XRSTOR.)

  Depending on details of the operating system, an execution of XSAVEOPT by a user application might use the modified optimization when the most recent execution of XRSTOR was by a different application. Because of this, Intel recommends the application software not use the XSAVEOPT instruction.

The MXCSR register and MXCSR_MASK are part of SSE state (see Section 13.5.2) and are thus associated with bit 1 of RFBM. However, the XSAVEOPT instruction also saves these values when RFBM[2] = 1 (even if RFBM[1] = 0). The init and modified optimizations do not apply to the MXCSR register and MXCSR_MASK.

13.10 OPERATION OF XSAVEC

The operation of XSAVEC is similar to that of XSAVE. Two main differences are (1) XSAVEC uses the compacted format for the extended region of the XSAVE area; and (2) XSAVEC uses the init optimization (see Section 13.6).

Unlike XSAVEOPT, XSAVEC does not use the modified optimization. See Section 13.2 for details of how to determine whether XSAVEC is supported.

The XSAVEC 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**. The logical (bitwise) AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the state components to be saved.

The following conditions cause execution of the XSAVEC 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 the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

If none of these conditions cause a fault, execution of XSAVEC writes the XSTATE_BV field of the XSAVE header (see Section 13.4.2), setting XSTATE_BV[*i*] ($0 \leq i \leq 63$) as follows:[2]

- If RFBM[*i*] = 0, XSTATE_BV[*i*] is written as 0.
- 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 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. 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.

  XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. However, if RFBM[1] = 1 and MXCSR does not have the value 1F80H, XSAVEC writes XSTATE_BV[1] as 1 even if XINUSE[1] = 0.

  (As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

The XSAVEC 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 XSAVEC saves into the XSAVE area those state components corresponding to bits that are set in RFBM (subject to the init optimization described below). State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \leq i \leq 62$, is located in the extended region; the XSAVEC 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. See Section 13.13 for details regarding faults caused by memory accesses.

Execution of XSAVEC 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, XSAVEC writes saves all of state component 1 (SSE — including the XMM registers) even if XINUSE[1] = 0. Unlike the XSAVE instruction, RFBM[2] does not determine whether XSAVEC saves MXCSR and MXCSR_MASK.

13.11 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

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.
2. Unlike the XSAVE and XSAVEOPT instructions, the XSAVEC instruction does **not** read the XSTATE_BV field of the XSAVE header.

(3) XSAVES uses the modified optimization (see Section 13.6). 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.[1]

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.
- 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 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. 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.

  XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. 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.

  (As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

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 XSAVES 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 (subject to the optimizations described below). 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. See Section 13.13 for details regarding faults caused by memory accesses.

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 13.8.3 and Section 13.12). 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

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.

- *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 13.6), state component *i* is not saved to the XSAVE area.

13.12 OPERATION OF XRSTORS

The operation of XRSTORS is similar to that of XRSTOR. Three main differences are (1) XRSTORS can be executed only if CPL = 0; (2) XRSTORS can operate on the state components whose bits are set in XCR0 | IA32_XSS; and (3) XRSTORS has only a compacted form (no standard form; see Section 13.8). See Section 13.2 for details of how to determine whether XRSTORS is supported.

The XRSTORS 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 restored.

The following conditions cause execution of the XRSTOR 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.[1]

After checking for these faults, the XRSTORS instruction reads the first 64 bytes of the XSAVE header, including the XSTATE_BV and XCOMP_BV fields (see Section 13.4.2). A #GP occurs if any of the following conditions hold for the values read:

- XCOMP_BV[63] = 0.
- XCOMP_BV sets a bit in the range 62:0 that is not set in XCR0 | IA32_XSS.
- XSTATE_BV sets a bit (including bit 63) that is not set in XCOMP_BV.
- Bytes 63:16 of the XSAVE header are not all 0.

If none of these conditions cause a fault, the processor updates each state component *i* for which RFBM[*i*] = 1. XRSTORS updates state component *i* based on the value of bit *i* in the XSTATE_BV field of the XSAVE header:

- If XSTATE_BV[*i*] = 0, the state component is set to its initial configuration. Section 13.6 specifies the initial configuration of each state component. If XSTATE_BV[1] = 0, XRSTORS initializes MXCSR to 1F80H.

  State component *i* is set to its initial configuration as indicated above if RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0 — **even if XCOMP_BV[*i*] = 0**. This is true for all values of *i*, including 0 (x87 state) and 1 (SSE state).
- If XSTATE_BV[*i*] = 1, the state component is loaded with data from the XSAVE area.[2] See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

  State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \leq i \leq 62$, is located in the extended region; XRSTORS uses the compacted format for the extended region (see Section 13.4.3).

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.
2. Earlier fault checking ensured that, if the instruction has reached this point in execution and XSTATE_BV[*i*] is 1, then XCOMP_BV[*i*] is also 1.

The MXCSR register is part of SSE state (see Section 13.5.2) and is thus loaded from memory if RFBM[1] = XSTATE_BV[*i*] = 1. XRSTORS causes a general-protection exception (#GP) if it would load MXCSR with an illegal value.

Like XRSTOR, execution of XRSTORS causes the processor to update is tracking for the init and modified optimizations (see Section 13.6 and Section 13.8.3). The following items provide details:

- The processor updates its tracking for the init optimization as follows:
  - If RFBM[*i*] = 0, XINUSE[*i*] is not changed.
  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0, state component *i* may be tracked as init; XINUSE[*i*] may be set to 0 or 1.
  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 1, state component *i* is tracked as not init; XINUSE[*i*] is set to 1.
- The processor updates its tracking for the modified optimization and records information about the XRSTORS execution for future interaction with the XSAVEOPT and XSAVES instructions as follows:
  - If RFBM[*i*] = 0, state component *i* is tracked as modified; XMODIFIED[*i*] is set to 1.
  - If RFBM[*i*] = 1, state component *i* may be tracked as unmodified; XMODIFIED[*i*] may be set to 0 or 1.
  - XRSTOR_INFO is set to the 4-tuple ⟨*w*,*x*,*y*,*z*⟩, where *w* is the CPL; *x* is 1 if the logical processor is in VMX non-root operation and 0 otherwise; *y* is the linear address of the XSAVE area; and *z* is XCOMP_BV (this implies that *z*[63] = 1).

13.13 MEMORY ACCESSES BY THE XSAVE FEATURE SET

Each instruction in the XSAVE feature set operates on a set of XSAVE-managed state components. The specific set of components on which an instruction operates is determined by the values of XCR0, the IA32_XSS MSR, EDX:EAX, and (for XRSTOR and XRSTORS) the XSAVE header.

Section 13.4 provides the details necessary to determine the location of each state component for any execution of an instruction in the XSAVE feature set. An execution of an instruction in the XSAVE feature set may access any byte of any state component on which that execution operates.

Section 13.5 provides details of the different XSAVE-managed state components. Some portions of some of these components are accessible only in 64-bit mode. Executions of XRSTOR and XRSTORS outside 64-bit mode will not update those portions; executions of XSAVE, XSAVEC, XSAVEOPT, and XSAVES will not modify the corresponding locations in memory.

Despite this fact, any execution of these instructions outside 64-bit mode may access any byte in any state component on which that execution operates — even those at addresses corresponding to registers that are accessible only in 64-bit mode. As result, such an execution may incur a fault due to an attempt to access such an address.

For example, an execution of XSAVE outside 64-bit mode may incur a page fault if paging does not map as read/write the section of the XSAVE area containing state component 7 (Hi16_ZMM state) — despite the fact that state component 7 can be accessed only in 64-bit mode.

...

6. Updates to Chapter 16, Volume 1

Change bars show changes to Chapter 16 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

...

16.1 I/O PORT ADDRESSING

The processor permits applications to access I/O ports in either of two ways:

- Through a separate I/O address space
- Through memory-mapped I/O

Accessing I/O ports through the I/O address space is handled through a set of I/O instructions and a special I/O protection mechanism. Accessing I/O ports through memory-mapped I/O is handled with the processor's general-purpose move and string instructions, with protection provided through segmentation or paging. I/O ports can be mapped so that they appear in the I/O address space or the physical-memory address space (memory mapped I/O) or both.

One benefit of using the I/O address space is that writes to I/O ports are guaranteed to be completed before the next instruction in the instruction stream is executed. Thus, I/O writes to control system hardware cause the hardware to be set to its new state before any other instructions are executed. See Section 16.6, "Ordering I/O," for more information on serializing of I/O operations.

...

16.3.1 Memory-Mapped I/O

I/O devices that respond like memory components can be accessed through the processor's physical-memory address space (see Figure 16-1). When using memory-mapped I/O, any of the processor's instructions that reference memory can be used to access an I/O port located at a physical-memory address. For example, the MOV instruction can transfer data between any register and a memory-mapped I/O port. The AND, OR, and TEST instructions may be used to manipulate bits in the control and status registers of a memory-mapped peripheral device.

When using memory-mapped I/O, caching of the address space mapped for I/O operations must be prevented. With the Pentium 4, Intel Xeon, and P6 family processors, caching of I/O accesses can be prevented by using memory type range registers (MTRRs) to map the address space used for the memory-mapped I/O as uncacheable (UC). See Chapter 11, "Memory Cache Control" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual*, *Volume 3A*, for a complete discussion of the MTRRs.

The Pentium and Intel486 processors do not support MTRRs. Instead, they provide the KEN# pin, which when held inactive (high) prevents caching of all addresses sent out on the system bus. To use this pin, external address decoding logic is required to block caching in specific address spaces.

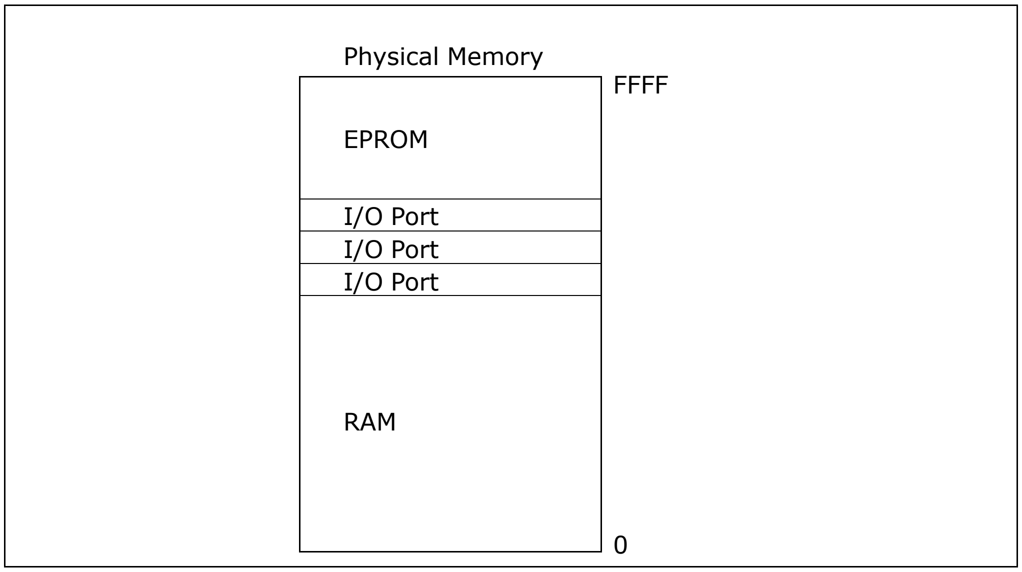


Figure 16-1 Memory-Mapped I/O

All the IA-32 processors that have on-chip caches also provide the PCD (page-level cache disable) flag in page table and page directory entries. This flag allows caching to be disabled on a page-by-page basis. See “Page-Directory and Page-Table Entries” in Chapter 4 of in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.*

...

16.4 I/O INSTRUCTIONS

The processor’s I/O instructions provide access to I/O ports through the I/O address space. (These instructions cannot be used to access memory-mapped I/O ports.) There are two groups of I/O instructions:

- Those that transfer a single item (byte, word, or doubleword) between an I/O port and a general-purpose register
- Those that transfer strings of items (strings of bytes, words, or doublewords) between an I/O port and memory

The register I/O instructions IN (input from I/O port) and OUT (output to I/O port) move data between I/O ports and the EAX register (32-bit I/O), the AX register (16-bit I/O), or the AL (8-bit I/O) register. The address of the I/O port can be given with an immediate value or a value in the DX register.

The string I/O instructions INS (input string from I/O port) and OUTS (output string to I/O port) move data between an I/O port and a memory location. The address of the I/O port being accessed is given in the DX register; the source or destination memory address is given in the DS:ESI or ES:EDI register, respectively.

When used with the repeat prefix REP, the INS and OUTS instructions perform string (or block) input or output operations. The repeat prefix REP modifies the INS and OUTS instructions to transfer blocks of data between an I/O port and memory. Here, the ESI or EDI register is incremented or decremented (according to the setting of the DF flag in the EFLAGS register) after each byte, word, or doubleword is transferred between the selected I/O port and memory.

See the references for IN, INS, OUT, and OUTS in Chapter 3 and Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A & 2B*, for more information on these instructions.

...

16.5.1 I/O Privilege Level

In systems where I/O protection is used, the IOPL field in the EFLAGS register controls access to the I/O address space by restricting use of selected instructions. This protection mechanism permits the operating system or executive to set the privilege level needed to perform I/O. In a typical protection ring model, access to the I/O address space is restricted to privilege levels 0 and 1. Here, the kernel and the device drivers are allowed to perform I/O, while less privileged device drivers and application programs are denied access to the I/O address space. Application programs must then make calls to the operating system to perform I/O.

The following instructions can be executed only if the current privilege level (CPL) of the program or task currently executing is less than or equal to the IOPL: IN, INS, OUT, OUTS, CLI (clear interrupt-enable flag), and STI (set interrupt-enable flag). These instructions are called **I/O sensitive** instructions, because they are sensitive to the IOPL field. Any attempt by a less privileged program or task to use an I/O sensitive instruction results in a general-protection exception (#GP) being signaled. Because each task has its own copy of the EFLAGS register, each task can have a different IOPL.

The I/O permission bit map in the TSS can be used to modify the effect of the IOPL on I/O sensitive instructions, allowing access to some I/O ports by less privileged programs or tasks (see Section 16.5.2, “I/O Permission Bit Map”).

A program or task can change its IOPL only with the POPF and IRET instructions; however, such changes are privileged. No procedure may change the current IOPL unless it is running at privilege level 0. An attempt by a less privileged procedure to change the IOPL does not result in an exception; the IOPL simply remains unchanged.

The POPF instruction also may be used to change the state of the IF flag (as can the CLI and STI instructions); however, the POPF instruction in this case is also I/O sensitive. A procedure may use the POPF instruction to change the setting of the IF flag only if the CPL is less than or equal to the current IOPL. An attempt by a less privileged procedure to change the IF flag does not result in an exception; the IF flag simply remains unchanged.

16.5.2 I/O Permission Bit Map

The I/O permission bit map is a device for permitting limited access to I/O ports by less privileged programs or tasks and for tasks operating in virtual-8086 mode. The I/O permission bit map is located in the TSS (see Figure 16-2) for the currently running task or program. The address of the first byte of the I/O permission bit map is given in the I/O map base address field of the TSS. The size of the I/O permission bit map and its location in the TSS are variable.

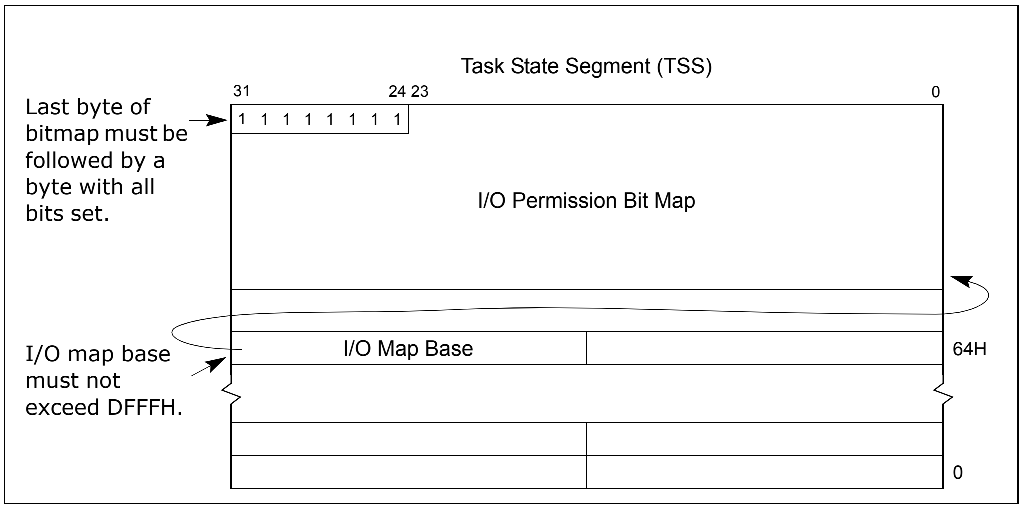

Task State Segment (TSS)
31
24 23
0
Last byte of bitmap must be followed by a byte with all bits set.
1 1 1 1 1 1 1 1
I/O Permission Bit Map
I/O map base must not exceed DFFFH.
I/O Map Base
64H
0

Figure 16-2 I/O Permission Bit Map

Because each task has its own TSS, each task has its own I/O permission bit map. Access to individual I/O ports can thus be granted to individual tasks.

If in protected mode and the CPL is less than or equal to the current IOPL, the processor allows all I/O operations to proceed. If the CPL is greater than the IOPL or if the processor is operating in virtual-8086 mode, the processor checks the I/O permission bit map to determine if access to a particular I/O port is allowed. Each bit in the map corresponds to an I/O port byte address. For example, the control bit for I/O port address 29H in the I/O address space is found at bit position 1 of the sixth byte in the bit map. Before granting I/O access, the processor tests all the bits corresponding to the I/O port being addressed. For a doubleword access, for example, the processors tests the four bits corresponding to the four adjacent 8-bit port addresses. If any tested bit is set, a general-protection exception (#GP) is signaled. If all tested bits are clear, the I/O operation is allowed to proceed.

Because I/O port addresses are not necessarily aligned to word and doubleword boundaries, the processor reads two bytes from the I/O permission bit map for every access to an I/O port. To prevent exceptions from being generated when the ports with the highest addresses are accessed, an extra byte needs to be included in the TSS immediately after the table. This byte must have all of its bits set, and it must be within the segment limit.

It is not necessary for the I/O permission bit map to represent all the I/O addresses. I/O addresses not spanned by the map are treated as if they had set bits in the map. For example, if the TSS segment limit is 10 bytes past the bit-map base address, the map has 11 bytes and the first 80 I/O ports are mapped. Higher addresses in the I/O address space generate exceptions.

If the I/O bit map base address is greater than or equal to the TSS segment limit, there is no I/O permission map, and all I/O instructions generate exceptions when the CPL is greater than the current IOPL.

16.6 ORDERING I/O

When controlling I/O devices it is often important that memory and I/O operations be carried out in precisely the order programmed. For example, a program may write a command to an I/O port, then read the status of the I/O device from another I/O port. It is important that the status returned be the status of the device **after** it receives the command, not **before**.

When using memory-mapped I/O, caution should be taken to avoid situations in which the programmed order is not preserved by the processor. To optimize performance, the processor allows cacheable memory reads to be reordered ahead of buffered writes in most situations. Internally, processor reads (cache hits) can be reordered around buffered writes. When using memory-mapped I/O, therefore, it is possible that an I/O read might be performed before the memory write of a previous instruction. The recommended method of enforcing program ordering of memory-mapped I/O accesses with the Pentium 4, Intel Xeon, and P6 family processors is to use the MTRRs to make the memory mapped I/O address space uncacheable; for the Pentium and Intel486 processors, either the KEN# pin or the PCD flags can be used for this purpose (see Section 16.3.1, "Memory-Mapped I/O").

When the target of a read or write is in an uncacheable region of memory, memory reordering does not occur externally at the processor's pins (that is, reads and writes appear in-order). Designating a memory mapped I/O region of the address space as uncacheable insures that reads and writes of I/O devices are carried out in program order. See Chapter 11, "Memory Cache Control" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for more information on using MTRRs.

Another method of enforcing program order is to insert one of the serializing instructions, such as the CPUID instruction, between operations. See Chapter 8, "Multiple-Processor Management" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for more information on serialization of instructions.

It should be noted that the chip set being used to support the processor (bus controller, memory controller, and/or I/O controller) may post writes to uncacheable memory which can lead to out-of-order execution of memory accesses. In situations where out-of-order processing of memory accesses by the chip set can potentially cause faulty memory-mapped I/O processing, code must be written to force synchronization and ordering of I/O operations. Serializing instructions can often be used for this purpose.

When the I/O address space is used instead of memory-mapped I/O, the situation is different in two respects:

- The processor never buffers I/O writes. Therefore, strict ordering of I/O operations is enforced by the processor. (As with memory-mapped I/O, it is possible for a chip set to post writes in certain I/O ranges.)
- The processor synchronizes I/O instruction execution with external bus activity (see Table 16-1).

Table 16-1 I/O Instruction Serialization

Instruction Being Executed	Processor Delays Execution of …		Until Completion of …	
	Current Instruction?	Next Instruction?	Pending Stores?	Current Store?
IN	Yes		Yes	
INS	Yes		Yes	
REP INS	Yes		Yes	
OUT		Yes	Yes	Yes
OUTS		Yes	Yes	Yes
REP OUTS		Yes	Yes	Yes

...

7. Updates to Chapter 17, Volume 1

Change bars show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

...

17.1 USING THE CPUID INSTRUCTION

Use the CPUID instruction for processor identification in the Pentium M processor family, Pentium 4 processor family, Intel Xeon processor family, P6 family, Pentium processor, and later Intel486 processors. This instruction returns the family, model and (for some processors) a brand string for the processor that executes the instruction. It also indicates the features that are present in the processor and gives information about the processor's caches and TLB.

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. The CPUID instruction will cause the invalid opcode exception (#UD) if executed on a processor that does not support it.

To obtain processor identification information, a source operand value is placed in the EAX register to select the type of information to be returned. When the CPUID instruction is executed, selected information is returned in the EAX, EBX, ECX, and EDX registers. For a complete description of the CPUID instruction, tables indicating values returned, and example code, see *CPUID—CPU Identification* in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.*

17.1.1 Notes on Where to Start

The following guidelines are among the most important, and should always be followed when using the CPUID instruction to determine available features:

- Always begin by testing for the "GenuineIntel," message in the EBX, EDX, and ECX registers when the CPUID instruction is executed with EAX equal to 0. If the processor is not genuine Intel, the feature identification flags may have different meanings than are described in Intel documentation.
- Test feature identification flags individually and do not make assumptions about undefined bits.

17.1.2 Identification of Earlier IA-32 Processors

The CPUID instruction is not available in earlier IA-32 processors up through the earlier Intel486 processors. For these processors, several other architectural features can be exploited to identify the processor.

The settings of bits 12 and 13 (IOPL), 14 (NT), and 15 (reserved) in the EFLAGS register are different for Intel's 32-bit processors than for the Intel 8086 and Intel 286 processors. By examining the settings of these bits (with the PUSHF/PUSHFD and POPF/POPFD instructions), an application program can determine whether the processor is an 8086, Intel 286, or one of the Intel 32-bit processors:

- 8086 processor — Bits 12 through 15 of the EFLAGS register are always set.
- Intel 286 processor — Bits 12 through 15 are always clear in real-address mode.
- 32-bit processors — In real-address mode, bit 15 is always clear and bits 12 through 14 have the last value loaded into them. In protected mode, bit 15 is always clear, bit 14 has the last value loaded into it, and the IOPL bits depend on the current privilege level (CPL). The IOPL field can be changed only if the CPL is 0.

Other EFLAGS register bits that can be used to differentiate between the 32-bit processors:

- Bit 18 (AC) — Implemented only on the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The inability to set or clear this bit distinguishes an Intel386 processor from the later IA-32 processors.
- Bit 21 (ID) — Determines if the processor is able to execute the CPUID instruction. The ability to set and clear this bit indicates that it is a Pentium 4, Intel Xeon, P6 family, Pentium, or later-version Intel486 processor.

To determine whether an x87 FPU or NPX is present in a system, applications can write to the x87 FPU status and control registers using the FNINIT instruction and then verify that the correct values are read back using the FNSTENV instruction.

After determining that an x87 FPU or NPX is present, its type can then be determined. In most cases, the processor type will determine the type of FPU or NPX; however, an Intel386 processor is compatible with either an Intel 287 or Intel 387 math coprocessor.

The method the coprocessor uses to represent ∞ (after the execution of the FINIT, FNINIT, or RESET instruction) indicates which coprocessor is present. The Intel 287 math coprocessor uses the same bit representation for +∞ and –∞; whereas, the Intel 387 math coprocessor uses different representations for +∞ and –∞.

...

8. Updates to Appendix A, Volume 1

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

...

A.1 EFLAGS AND INSTRUCTIONS

Table A-2 summarizes how the instructions affect the flags in the EFLAGS register. The following codes describe how the flags are affected.

Table A-1 Codes Describing Flags

T	Instruction tests flag.
M	Instruction modifies flag (either sets or resets depending on operands).
0	Instruction resets flag.
1	Instruction sets flag.
—	Instruction's effect on flag is undefined.
R	Instruction restores prior value of flag.
Blank	Instruction does not affect flag.

Table A-2 EFLAGS Cross-Reference

Instruction	OF	SF	ZF	AF	PF	CF	TF	IF	DF	NT	RF
AAA	—	—	—	TM	—	M					
AAD	—	M	M	—	M	—					
AAM	—	M	M	—	M	—					
AAS	—	—	—	TM	—	M					
ADC	M	M	M	M	M	TM					
ADD	M	M	M	M	M	M					
AND	0	M	M	—	M	0					
ARPL			M								
BOUND											
BSF/BSR	—	—	M	—	—	—					
BSWAP											
BT/BTS/BTR/BTC	—	—		—	—	M					

Table A-2 EFLAGS Cross-Reference (Contd.)

Instruction	OF	SF	ZF	AF	PF	CF	TF	IF	DF	NT	RF
CALL											
CBW											
CLC						0					
CLD									0		
CLI								0			
CLTS											
CMC						M					
CMOV*cc*	T	T	T		T	T					
CMP	M	M	M	M	M	M					
CMPS	M	M	M	M	M	M			T		
CMPXCHG	M	M	M	M	M	M					
CMPXCHG8B			M								
COMISD	0	0	M	0	M	M					
COMISS	0	0	M	0	M	M					
CPUID											
CWD											
DAA	—	M	M	TM	M	TM					
DAS	—	M	M	TM	M	TM					
DEC	M	M	M	M	M						
DIV	—	—	—	—	—	—					
ENTER											
ESC											
FCMOV*cc*			T		T	T					
FCOMI, FCOMIP, FUCOMI, FUCOMIP	0	0	M	0	M	M					
HLT											
IDIV	—	—	—	—	—	—					
IMUL	M	—	—	—	—	M					
IN											
INC	M	M	M	M	M						
INS									T		
INT							0			0	
INTO	T						0			0	
INVD											
INVLPG											
UCOMISD	0	0	M	0	M	M					
UCOMISS	0	0	M	0	M	M					

Table A-2 EFLAGS Cross-Reference (Contd.)

Instruction	OF	SF	ZF	AF	PF	CF	TF	IF	DF	NT	RF
IRET	R	R	R	R	R	R	R	R	R	T	
J*cc*	T	T	T		T	T					
JCXZ											
JMP											
LAHF											
LAR			M								
LDS/LES/LSS/LFS/LGS											
LEA											
LEAVE											
LGDT/LIDT/LLDT/LMSW											
LOCK											
LODS									T		
LOOP											
LOOPE/LOOPNE			T								
LSL			M								
LTR											
MONITOR											
MWAIT											
MOV											
MOV control, debug, test	—	—	—	—	—	—					
MOVS									T		
MOVSX/MOVZX											
MUL	M	—	—	—	—	M					
NEG	M	M	M	M	M	M					
NOP											
NOT											
OR	0	M	M	—	M	0					
OUT											
OUTS									T		
POP/POPA											
POPF	R	R	R	R	R	R	R	R	R	R	
PUSH/PUSHA/PUSHF											
RCL/RCR 1	M					TM					
RCL/RCR count	—					TM					
RDMSR											
RDPMC											

Table A-2 EFLAGS Cross-Reference (Contd.)

Instruction	OF	SF	ZF	AF	PF	CF	TF	IF	DF	NT	RF
RDTSC											
REP/REPE/REPNE											
RET											
ROL/ROR 1	M					M					
ROL/ROR count	—					M					
RSM	M	M	M	M	M	M	M	M	M	M	M
SAHF		R	R	R	R	R					
SAL/SAR/SHL/SHR 1	M	M	M	—	M	M					
SAL/SAR/SHL/SHR count	—	M	M	—	M	M					
SBB	M	M	M	M	M	TM					
SCAS	M	M	M	M	M	M			T		
SET*cc*	T	T	T		T	T					
SGDT/SIDT/SLDT/SMSW											
SHLD/SHRD	—	M	M	—	M	M					
STC						1					
STD									1		
STI								1			
STOS									T		
STR											
SUB	M	M	M	M	M	M					
TEST	0	M	M	—	M	0					
UD2											
VERR/VERRW			M								
WAIT											
WBINVD											
WRMSR											
XADD	M	M	M	M	M	M					
XCHG											
XLAT											
XOR	0	M	M	—	M	0					

...

9. 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.

...

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

See also:

- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Fortran Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Software Development Tools:
  http://www.intel.com/cd/software/products/asmo-na/eng/index.htm
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in three or seven volumes):
  http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
  http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html
- Intel 64 Architecture x2APIC Specification:

  http://www.intel.com/content/www/us/en/architecture-and-technology/64-architecture-x2apic-specification.html
- Intel® Trusted Execution Technology Measured Launched Environment Programming Guide:

  http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html
- Developing Multi-threaded Applications: A Platform Consistent Approach:
  https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
  http://software.intel.com/en-us/articles/ap949-using-spin-loops-on-intel-pentiumr-4-processor-and-intel-xeonr-processor/
- Performance Monitoring Unit Sharing Guide
  http://software.intel.com/file/30388

Literature related to selected features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference
  https://software.intel.com/en-us/isa-extensions
- Intel® Software Guard Extensions (Intel® SGX) Programming Reference
  https://software.intel.com/en-us/isa-extensions/intel-sgx

More relevant links are:

- Intel® Developer Zone:

  https://software.intel.com/en-us
- Developer centers:

  http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html
- Processor support general link:

http://www.intel.com/support/processors/

- Software products and packages:

  http://www.intel.com/cd/software/products/asmo-na/eng/index.htm
- Intel® Hyper-Threading Technology (Intel® HT Technology):

  http://www.intel.com/technology/platform-technology/hyper-threading/index.htm

...

10. Updates to Chapter 2, Volume 2A

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

...

2.4 INSTRUCTION EXCEPTION SPECIFICATION

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

"See Exceptions Type 2"

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

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

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

NOTE

Instructions that operate only with MMX, X87, or general-purpose registers are not covered by the exception classes defined in this section. For instructions that operate on MMX registers, see Section 22.25.3, "Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

Table 2-14 Exception class description

Exception Class	Instruction set	Mem arg	Floating-Point Exceptions (#XM)
Type 1	AVX, Legacy SSE	16/32 byte explicitly aligned	none
Type 2	AVX, Legacy SSE	16/32 byte not explicitly aligned	yes
Type 3	AVX, Legacy SSE	< 16 byte	yes
Type 4	AVX, Legacy SSE	16/32 byte not explicitly aligned	no
Type 5	AVX, Legacy SSE	< 16 byte	no

Exception Class	Instruction set	Mem arg	Floating-Point Exceptions (#XM)
Type 6	AVX (no Legacy SSE)	Varies	(At present, none do)
Type 7	AVX, Legacy SSE	none	none
Type 8	AVX	none	none
Type 11	F16C	8 or 16 byte, Not explicitly aligned, no AC#	yes
Type 12	AVX2	Not explicitly aligned, no AC#	no

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

Table 2-15 Instructions in each Exception Class

Exception Class	Instruction
Type 1	(V)MOVAPD, (V)MOVAPS, (V)MOVDQA, (V)MOVNTDQ, (V)MOVNTDQA, (V)MOVNTPD, (V)MOVNTPS
Type 2	(V)ADDPD, (V)ADDPS, (V)ADDSUBPD, (V)ADDSUBPS, (V)CMPPD, (V)CMPPS, (V)CVTDQ2PS, (V)CVTPD2DQ, (V)CVTPD2PS, (V)CVTPS2DQ, (V)CVTTPD2DQ, (V)CVTTPS2DQ, (V)DIVPD, (V)DIVPS, (V)DPPD*, (V)DPPS*, VFMADD132PD, VFMADD213PD, VFMADD231PD, VFMADD132PS, VFMADD213PS, VFMADD231PS, VFMADDSUB132PD, VFMADDSUB213PD, VFMADDSUB231PD, VFMADDSUB132PS, VFMADDSUB213PS, VFMADDSUB231PS, VFMSUBADD132PD, VFMSUBADD213PD, VFMSUBADD231PD, VFMSUBADD132PS, VFMSUBADD213PS, VFMSUBADD231PS, VFMSUB132PD, VFMSUB213PD, VFMSUB231PD, VFMSUB132PS, VFMSUB213PS, VFMSUB231PS, VFNMADD132PD, VFNMADD213PD, VFNMADD231PD, VFNMADD132PS, VFNMADD213PS, VFNMADD231PS, VFNMSUB132PD, VFNMSUB213PD, VFNMSUB231PD, VFNMSUB132PS, VFNMSUB213PS, VFNMSUB231PS, (V)HADDPD, (V)HADDPS, (V)HSUBPD, (V)HSUBPS, (V)MAXPD, (V)MAXPS, (V)MINPD, (V)MINPS, (V)MULPD, (V)MULPS, (V)ROUNDPS, (V)SQRTPD, (V)SQRTPS, (V)SUBPD, (V)SUBPS
Type 3	(V)ADDSD, (V)ADDSS, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)COMISS, (V)CVTPS2PD, (V)CVTSD2SI, (V)CVTSD2SS, (V)CVTSI2SD, (V)CVTSI2SS, (V)CVTSS2SD, (V)CVTSS2SI, (V)CVTTSD2SI, (V)CVTTSS2SI, (V)DIVSD, (V)DIVSS, VFMADD132SD, VFMADD213SD, VFMADD231SD, VFMADD132SS, VFMADD213SS, VFMADD231SS, VFMSUB132SD, VFMSUB213SD, VFMSUB231SD, VFMSUB132SS, VFMSUB213SS, VFMSUB231SS, VFNMADD132SD, VFNMADD213SD, VFNMADD231SD, VFNMADD132SS, VFNMADD213SS, VFNMADD231SS, VFNMSUB132SD, VFNMSUB213SD, VFNMSUB231SD, VFNMSUB132SS, VFNMSUB213SS, VFNMSUB231SS, (V)MAXSD, (V)MAXSS, (V)MINSD, (V)MINSS, (V)MULSD, (V)MULSS, (V)ROUNDSD, (V)ROUNDSS, (V)SQRTSD, (V)SQRTSS, (V)SUBSD, (V)SUBSS, (V)UCOMISD, (V)UCOMISS

Exception Class	Instruction
Type 4	(V)AESDEC, (V)AESDECLAST, (V)AESENC, (V)AESENCLAST, (V)AESIMC, (V)AESKEYGENASSIST, (V)ANDPD, (V)ANDPS, (V)ANDNPD, (V)ANDNPS, (V)BLENDPD, (V)BLENDPS, VBLENDVPD, VBLENDVPS, (V)LDDQU***, (V)MASKMOVDQU, (V)PTEST, VTESTPS, VTESTPD, (V)MOVDQU*, (V)MOVSHDUP, (V)MOVSLDUP, (V)MOVUPD*, (V)MOVUPS*, (V)MPSADBW, (V)ORPD, (V)ORPS, (V)PABSB, (V)PABSW, (V)PABSD, (V)PACKSSWB, (V)PACKSSDW, (V)PACKUSWB, (V)PACKUSDW, (V)PADDB, (V)PADDW, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PALIGNR, (V)PAND, (V)PANDN, (V)PAVGB, (V)PAVGW, (V)PBLENDVB, (V)PBLENDW, (V)PCMP(E/I)STRI/M***, (V)PCMPEQB, (V)PCMPEQW, (V)PCMPEQD, (V)PCMPEQQ, (V)PCMPGTB, (V)PCMPGTW, (V)PCMPGTD, (V)PCMPGTQ, (V)PCLMULQDQ, (V)PHADDW, (V)PHADDD, (V)PHADDSW, (V)PHMINPOSUW, (V)PHSUBD, (V)PHSUBW, (V)PHSUBSW, (V)PMADDWD, (V)PMADDUBSW, (V)PMAXSB, (V)PMAXSW, (V)PMAXSD, (V)PMAXUB, (V)PMAXUW, (V)PMAXUD, (V)PMINSB, (V)PMINSW, (V)PMINSD, (V)PMINUB, (V)PMINUW, (V)PMINUD, (V)PMULHUW, (V)PMULHRSW, (V)PMULHW, (V)PMULLW, (V)PMULLD, (V)PMULUDQ, (V)PMULDQ, (V)POR, (V)PSADBW, (V)PSHUFB, (V)PSHUFD, (V)PSHUFHW, (V)PSHUFLW, (V)PSIGNB, (V)PSIGNW, (V)PSIGND, (V)PSLLW, (V)PSLLD, (V)PSLLQ, (V)PSRAW, (V)PSRAD, (V)PSRLW, (V)PSRLD, (V)PSRLQ, (V)PSUBB, (V)PSUBW, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PUNPCKHBW, (V)PUNPCKHWD, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKLBW, (V)PUNPCKLWD, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PXOR, (V)RCPPS, (V)RSQRTPS, (V)SHUFPD, (V)SHUFPS, (V)UNPCKHPD, (V)UNPCKHPS, (V)UNPCKLPD, (V)UNPCKLPS, (V)XORPD, (V)XORPS, VPBLENDD, VPERMD, VPERMPS, VPERMPD, VPERMQ, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ, VPERMILPD, VPERMILPS, VPERM2F128
Type 5	(V)CVTDQ2PD, (V)EXTRACTPS, (V)INSERTPS, (V)MOVD, (V)MOVQ, (V)MOVDDUP, (V)MOVLPD, (V)MOVLPS, (V)MOVHPD, (V)MOVHPS, (V)MOVSD, (V)MOVSS, (V)PEXTRB, (V)PEXTRD, (V)PEXTRW, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ, (V)RCPSS, (V)RSQRTSS, (V)PMOVSX/ZX, VLDMXCSR*, VSTMXCSR
Type 6	VEXTRACTF128, VBROADCASTSS, VBROADCASTSD, VBROADCASTF128, VINSERTF128, VMASKMOVPS**, VMASKMOVPD**, VPMASKMOVD, VPMASKMOVQ, VBROADCASTI128, VPBROADCASTB, VPBROADCASTD, VPBROADCASTW, VPBROADCASTQ, VEXTRACTI128, VINSERTI128, VPERM2I128
Type 7	(V)MOVLHPS, (V)MOVHLPS, (V)MOVMSKPD, (V)MOVMSKPS, (V)PMOVMSKB, (V)PSLLDQ, (V)PSRLDQ, (V)PSLLW, (V)PSLLD, (V)PSLLQ, (V)PSRAW, (V)PSRAD, (V)PSRLW, (V)PSRLD, (V)PSRLQ
Type 8	VZEROALL, VZEROUPPER
Type 11	VCVTPH2PS, VCVTPS2PH
Type 12	VGATHERDPS, VGATHERDPD, VGATHERQPS, VGATHERQPD, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

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

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

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

...

2.4.1 Exceptions Type 1 (Aligned memory reference)

Table 2-18 Type 1 Class Exception Conditions

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

...

2.4.3 Exceptions Type 3 (<16 Byte memory argument)

Table 2-20 Type 3 Class Exception Conditions

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

2.4.4 Exceptions Type 4 (>=16 Byte mem arg no alignment, no floating-point exceptions)

Table 2-21 Type 4 Class Exception Conditions

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

NOTES:

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

2.4.5 Exceptions Type 5 (<16 Byte mem arg and no FP exceptions)

Table 2-22 Type 5 Class Exception Conditions

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

2.4.6 Exceptions Type 6 (VEX-Encoded Instructions Without Legacy SSE Analogues)

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

Table 2-23 Type 6 Class Exception Conditions

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

2.4.7 Exceptions Type 7 (No FP exceptions, no memory arg)

Table 2-24 Type 7 Class Exception Conditions

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

2.4.8 Exceptions Type 8 (AVX and no memory argument)

Table 2-25 Type 8 Class Exception Conditions

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

2.4.9 Exception Type 11 (VEX-only, mem arg no AC, floating-point exceptions)

Table 2-26 Type 11 Class Exception Conditions

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

2.4.10 Exception Type 12 (VEX-only, VSIB mem arg, no AC, no floating-point exceptions)

Table 2-27 Type 12 Class Exception Conditions

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

2.5.1 Exception Conditions for VEX-Encoded GPR Instructions

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

Table 2-28 VEX-Encoded GPR Instructions

Exception Class	Instruction
See Table 2-29	ANDN, BLSI, BLSMSK, BLSR, BZHI, MULX, PDEP, PEXT, RORX, SARX, SHLX, SHRX

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

Table 2-29 Exception Definition (VEX-Encoded GPR Instructions)

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

...

11. Updates to Chapter 3, Volume 2A

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

...

3.1.1.9 Operation Section

The "Operation" section contains an algorithm description (frequently written in pseudo-code) for the instruction. Algorithms are composed of the following elements:

- Comments are enclosed within the symbol pairs "(*" and "*)".
- Compound statements are enclosed in keywords, such as: IF, THEN, ELSE and FI for an if statement; DO and OD for a do statement; or CASE... OF for a case statement.
- A register name implies the contents of the register. A register name enclosed in brackets implies the contents of the location whose address is contained in that register. For example, ES:[DI] indicates the contents of the location whose ES segment relative address is in register DI. [SI] indicates the contents of the address contained in register SI relative to the SI register's default segment (DS) or the overridden segment.

- Parentheses around the "E" in a general-purpose register name, such as (E)SI, indicates that the offset is read from the SI register if the address-size attribute is 16, from the ESI register if the address-size attribute is 32. Parentheses around the "R" in a general-purpose register name, (R)SI, in the presence of a 64-bit register definition such as (R)SI, indicates that the offset is read from the 64-bit RSI register if the address-size attribute is 64.
- Brackets are used for memory operands where they mean that the contents of the memory location is a segment-relative offset. For example, [SRC] indicates that the content of the source operand is a segment-relative offset.
- $A \leftarrow B$ indicates that the value of B is assigned to A.
- The symbols $=$, $\neq$, $>$, $<$, $\geq$, and $\leq$ are relational operators used to compare two values: meaning equal, not equal, greater or equal, less or equal, respectively. A relational expression such as $A = B$ is TRUE if the value of A is equal to B; otherwise it is FALSE.
- The expression "« COUNT" and "» COUNT" indicates that the destination operand should be shifted left or right by the number of bits indicated by the count operand.

The following identifiers are used in the algorithmic descriptions:

- **OperandSize and AddressSize** — The OperandSize identifier represents the operand-size attribute of the instruction, which is 16, 32 or 64-bits. The AddressSize identifier represents the address-size attribute, which is 16, 32 or 64-bits. For example, the following pseudo-code indicates that the operand-size attribute depends on the form of the MOV instruction used.

```
IF Instruction = MOVW
    THEN OperandSize ← 16;
ELSE
    IF Instruction = MOVD
        THEN OperandSize ← 32;
    ELSE
        IF Instruction = MOVQ
            THEN OperandSize ← 64;
        FI;
    FI;
FI;
```

  See "Operand-Size and Address-Size Attributes" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for guidelines on how these attributes are determined.

- **StackAddrSize** — Represents the stack address-size attribute associated with the instruction, which has a value of 16, 32 or 64-bits. See "Address-Size Attribute for Stack" in Chapter 6, "Procedure Calls, Interrupts, and Exceptions," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.
- **SRC** — Represents the source operand.
- **DEST** — Represents the destination operand.
- **VLMAX** — The maximum vector register width pertaining to the instruction. This is not the vector-length encoding in the instruction's prefix but is instead determined by the current value of XCR0. For existing processors, VLMAX is 256 whenever XCR0.YMM[bit 2] is 1. Future processors may defined new bits in XCR0 whose setting may imply other values for VLMAX.

VLMAX Definition

XCR0 Component	VLMAX
XCR0.YMM	256

The following functions are used in the algorithmic descriptions:

- **ZeroExtend(value)** — Returns a value zero-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, zero extending a byte value of –10 converts the byte from F6H to a doubleword value of 000000F6H. If the value passed to the ZeroExtend function and the operand-size attribute are the same size, ZeroExtend returns the value unaltered.
- **SignExtend(value)** — Returns a value sign-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, sign extending a byte containing the value –10 converts the byte from F6H to a doubleword value of FFFFFFF6H. If the value passed to the SignExtend function and the operand-size attribute are the same size, SignExtend returns the value unaltered.
- **SaturateSignedWordToSignedByte** — Converts a signed 16-bit value to a signed 8-bit value. If the signed 16-bit value is less than –128, it is represented by the saturated value -128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateSignedDwordToSignedWord** — Converts a signed 32-bit value to a signed 16-bit value. If the signed 32-bit value is less than –32768, it is represented by the saturated value –32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateSignedWordToUnsignedByte** — Converts a signed 16-bit value to an unsigned 8-bit value. If the signed 16-bit value is less than zero, it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToSignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than –128, it is represented by the saturated value –128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateToSignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than –32768, it is represented by the saturated value –32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateToUnsignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToUnsignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 65535, it is represented by the saturated value 65535 (FFFFH).
- **LowOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the least significant word of the doubleword result in the destination operand.
- **HighOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the most significant word of the doubleword result in the destination operand.
- **Push(value)** — Pushes a value onto the stack. The number of bytes pushed is determined by the operand-size attribute of the instruction. See the "Operation" subsection of the "PUSH—Push Word, Doubleword or Quadword Onto the Stack" section in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.
- **Pop()** removes the value from the top of the stack and returns it. The statement EAX ← Pop(); assigns to EAX the 32-bit value from the top of the stack. Pop will return either a word, a doubleword or a quadword depending on the operand-size attribute. See the "Operation" subsection in the "POP—Pop a Value from the Stack" section of Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.
- **PopRegisterStack** — Marks the FPU ST(0) register as empty and increments the FPU register stack pointer (TOP) by 1.
- **Switch-Tasks** — Performs a task switch.
- **Bit(BitBase, BitOffset)** — Returns the value of a bit within a bit string. The bit string is a sequence of bits in memory or a register. Bits are numbered from low-order to high-order within registers and within memory bytes. If the BitBase is a register, the BitOffset can be in the range 0 to [15, 31, 63] depending on the mode

and register size. See Figure 3-1: the function Bit[RAX, 21] is illustrated.

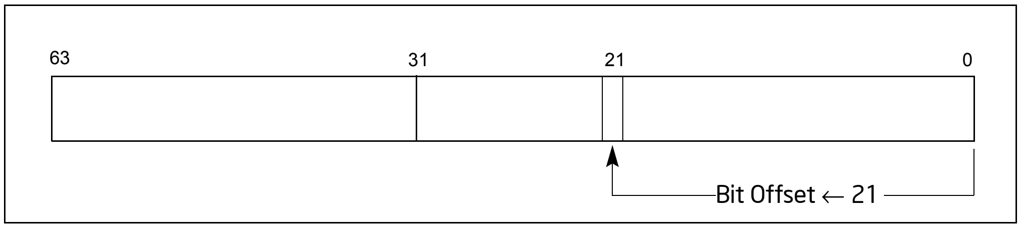


Figure 3-1 Bit Offset for BIT[RAX, 21]

If BitBase is a memory address, the BitOffset can range has different ranges depending on the operand size (see Table 3-2).

Table 3-2 Range of Bit Positions Specified by Bit Offset Operands

Operand Size	Immediate BitOffset	Register BitOffset
16	0 to 15	-2^{15} to $2^{15}-1$
32	0 to 31	-2^{31} to $2^{31}-1$
64	0 to 63	-2^{63} to $2^{63}-1$

The addressed bit is numbered (Offset MOD 8) within the byte at address (BitBase + (BitOffset DIV 8)) where DIV is signed division with rounding towards negative infinity and MOD returns a positive number (see Figure 3-2).

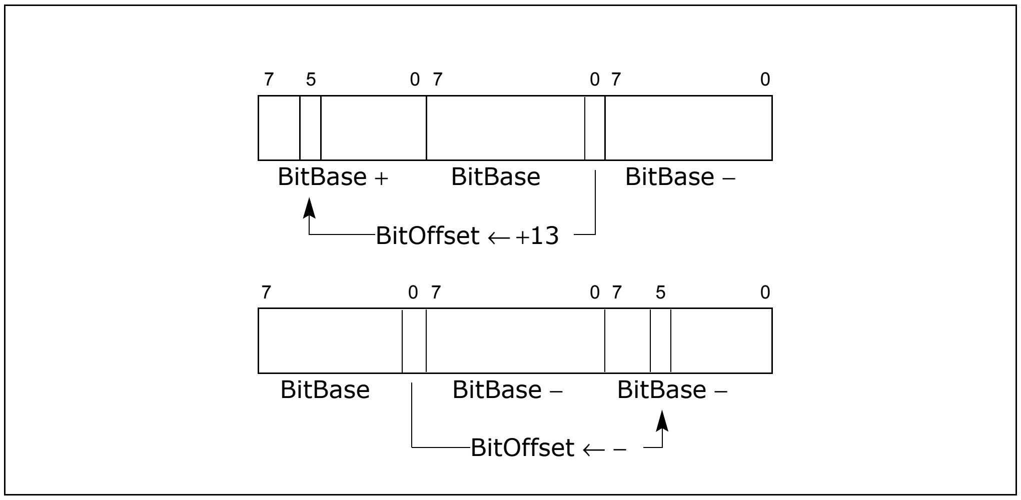


Figure 3-2 Memory Bit Indexing

...

CMP—Compare Two Operands

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
3C *ib*	CMP AL, *imm8*	I	Valid	Valid	Compare *imm8* with AL.
3D *iw*	CMP AX, *imm16*	I	Valid	Valid	Compare *imm16* with AX.
3D *id*	CMP EAX, *imm32*	I	Valid	Valid	Compare *imm32* with EAX.
REX.W + 3D *id*	CMP RAX, *imm32*	I	Valid	N.E.	Compare *imm32 sign-extended to 64-bits* with RAX.
80 /7 *ib*	CMP *r/m8, imm8*	MI	Valid	Valid	Compare *imm8* with *r/m8.*
REX + 80 /7 *ib*	CMP *r/m8**, *imm8*	MI	Valid	N.E.	Compare *imm8* with *r/m8.*
81 /7 *iw*	CMP *r/m16, imm16*	MI	Valid	Valid	Compare *imm16* with *r/m16.*
81 /7 *id*	CMP *r/m32, imm32*	MI	Valid	Valid	Compare *imm32* with *r/m32.*
REX.W + 81 /7 *id*	CMP *r/m64, imm32*	MI	Valid	N.E.	Compare *imm32 sign-extended to 64-bits* with *r/m64.*
83 /7 *ib*	CMP *r/m16, imm8*	MI	Valid	Valid	Compare *imm8* with *r/m16.*
83 /7 *ib*	CMP *r/m32, imm8*	MI	Valid	Valid	Compare *imm8* with *r/m32.*
REX.W + 83 /7 *ib*	CMP *r/m64, imm8*	MI	Valid	N.E.	Compare *imm8* with *r/m64.*
38 /*r*	CMP *r/m8, r8*	MR	Valid	Valid	Compare *r8* with *r/m8.*
REX + 38 /*r*	CMP *r/m8**, *r8**	MR	Valid	N.E.	Compare *r8* with *r/m8.*
39 /*r*	CMP *r/m16, r16*	MR	Valid	Valid	Compare *r16* with *r/m16.*
39 /*r*	CMP *r/m32, r32*	MR	Valid	Valid	Compare *r32* with *r/m32.*
REX.W + 39 /*r*	CMP *r/m64,r64*	MR	Valid	N.E.	Compare *r64* with *r/m64.*
3A /*r*	CMP *r8, r/m8*	RM	Valid	Valid	Compare *r/m8* with *r8.*
REX + 3A /*r*	CMP *r8**, *r/m8**	RM	Valid	N.E.	Compare *r/m8 with r8.*
3B /*r*	CMP *r16, r/m16*	RM	Valid	Valid	Compare *r/m16* with *r16.*
3B /*r*	CMP *r32, r/m32*	RM	Valid	Valid	Compare *r/m32* with *r32.*
REX.W + 3B /*r*	CMP *r64, r/m64*	RM	Valid	N.E.	Compare *r/m64* with *r64.*

NOTES:

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

Instruction Operand Encoding

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

Description

Compares the first source operand with the second source operand and sets the status flags in the EFLAGS register according to the results. The comparison is performed by subtracting the second operand from the first operand and then setting the status flags in the same manner as the SUB instruction. When an immediate value is used as an operand, it is sign-extended to the length of the first operand.

The condition codes used by the J*cc*, CMOV*cc*, and SET*cc* instructions are based on the results of a CMP instruction. Appendix B, "EFLAGS Condition Codes," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, shows the relationship of the status flags and the condition codes.

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

Operation

```
temp ← SRC1 − SignExtend(SRC2);
ModifyStatusFlags; (* Modify status flags in the same manner as the SUB instruction*)
```

Flags Affected

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

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

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

#UD	If the LOCK prefix is used.

...

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. *)
```

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.

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

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	Maximum Input Value for Basic CPUID Information (see Table 3-18)
	EBX	"Genu"
	ECX	"ntel"
	EDX	"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	Feature Information (see Figure 3-6 and Table 3-19)
	EDX	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	Cache and TLB Information (see Table 3-21)
	EBX	Cache and TLB Information
	ECX	Cache and TLB Information
	EDX	Cache and TLB Information
03H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature.
	CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLE.BOOT_NT4[bit 22] = 0 (default).	
	Deterministic Cache Parameters Leaf	

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

Initial EAX Value		Information Provided about the Processor
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 2-89.
	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_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS) are supported if set. Bit 08: HWP_Notification. IA32_HWP_INTERRUPT MSR is supported if set. Bit 09: HWP_Activity_Window. IA32_HWP_REQUEST[bits 41:32] is supported if set. Bit 10: HWP_Energy_Performance_Preference. IA32_HWP_REQUEST[bits 31:24] is supported if set. Bit 11: HWP_Package_Level_Request. IA32_HWP_REQUEST_PKG MSR is supported if set. Bit 12: Reserved. Bit 13: HDC. HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, IA32_THREAD_STALL MSRs are supported if set. Bits 31 - 15: Reserved
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor Bits 31 - 04: Reserved
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency. Bits 02 - 01: Reserved = 0 Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H). Bits 31 - 04: Reserved = 0
	EDX	Reserved = 0

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

Initial EAX Value	Information Provided about the Processor	
	Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)	
07H		Sub-leaf 0 (Input ECX = 0). *
	EAX	Bits 31-00: Reports the maximum input value for supported leaf 7 sub-leaves.
	EBX	Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: 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. Supports Supervisor-Mode Access Prevention (and the CLAC/STAC instructions) if 1. Bits 24:21: Reserved Bit 25: Intel Processor Trace Bits 31:26: Reserved
	ECX	Bit 00: PREFETCHWT1 Bits 02:01: Reserved Bit 03: PKU. Supports protection keys for user-mode pages if 1. Bit 04: OSPKE. If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions) Bits 31:05: Reserved
	EDX	Reserved **NOTE:** * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.
	Direct Cache Access Information Leaf	
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H)
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
	Architectural Performance Monitoring Leaf	

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 supported bits of the lower 32 bits of XCR0. XCR0[n] can be set to 1 only if EAX[n] is 1. Bit 00: x87 state Bit 01: SSE state Bit 02: AVX state Bits 04 - 03: Reserved Bit 07 - 05: AVX-512 state Bits 31- 08: Reserved
	EBX	Bits 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.
	ECX	Bit 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e all the valid bit fields in XCR0.
	EDX	Bit 31-00: Reports the supported bits of the upper 32 bits of XCR0. XCR0[n+32] can be set to 1 only if EDX[n] is 1. Bits 31- 00: Reserved
	Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)	
0DH	EAX	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 XCRO | IA32_XSS.
	ECX	Bits 31-00: Reports the supported bits of the lower 32 bits of the IA32_XSS MSR. IA32_XSS[n] can be set to 1 only if ECX[n] is 1. Bits 02-00: used for XCR0 Bits 04 - 03: Reserved Bit 07 - 05: used for XCR0 Bits 31-08: Reserved

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

Initial EAX Value		Information Provided about the Processor
	EDX	Bits 31-00: Reports the supported bits of the upper 32 bits of the IA32_XSS MSR. IA32_XSS[n+32] can be set to 1 only if EDX[n] is 1. Bits 31- 00: Reserved
	Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)	
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. Each sub-leaf index (starting at position 2) is supported if it corresponds to a supported bit in either the XCR0 register or the IA32_XSS MSR. 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 bit n (corresponding to the sub-leaf index) is supported in the IA32_XSS MSR; it is clear if bit n is instead supported in XCR0. 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

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

Initial EAX Value	Information Provided about the Processor	
	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.
	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	

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

Initial EAX Value	Information Provided about the Processor	
15H		**NOTES:** If EBX[31:0] is 0, the TSC/"core crystal clock" ratio is not enumerated. EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency. "TSC frequency" = "core crystal clock frequency" * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	EAX	Bits 31:0: An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31-0: An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.
	ECX	Bits 31:0: Reserved = 0.
	EDX	Bits 31:0: Reserved = 0.
	Processor Frequency Information Leaf	
16H	EAX	Bits 15:0: Processor Base Frequency (in MHz). Bits 31:16: Reserved =0
	EBX	Bits 15:0: Maximum Frequency (in MHz). Bits 31:16: Reserved = 0
	ECX	Bits 15:0: Bus (Reference) Frequency (in MHz). Bits 31:16: Reserved = 0
	EDX	Reserved
		NOTES: * Data is returned from this interface in accordance with the processor's specification and does not reflect actual values. Suitable use of this data includes the display of processor information in like manner to the processor brand string and for determining the appropriate range to use when displaying processor information e.g. frequency history graphs. The returned information should not be used for any other purpose as the returned information does not accurately correlate to information / counters returned by other processor interfaces. While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.
	Unimplemented CPUID Leaf Functions	
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 3-18).
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved

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

Initial EAX Value	Information Provided about the Processor	
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode Bits 04-01 Reserved Bit 05: LZCNT Bits 07-06 Reserved Bit 08: PREFETCHW Bits 31-09 Reserved
	EDX	Bits 10-00: Reserved Bit 11: SYSCALL/SYSRET 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 EBX ECX EDX	Processor Brand String Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000003H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000004H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000005H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0
80000006H	EAX EBX	Reserved = 0 Reserved = 0
	ECX	Bits 07-00: Cache Line size in bytes Bits 11-08: Reserved Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units
	EDX	Reserved = 0

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

Initial EAX Value	Information Provided about the Processor	
		NOTES: * L2 associativity field encodings: 00H - Disabled 01H - Direct mapped 02H - 2-way 04H - 4-way 06H - 8-way 08H - 16-way 0FH - Fully associative
80000007H	EAX	Reserved = 0
	EBX	Reserved = 0
	ECX	Reserved = 0
	EDX	Bits 07-00: Reserved = 0 Bit 08: Invariant TSC available if 1 Bits 31-09: Reserved = 0
80000008H	EAX	Linear/Physical Address size Bits 07-00: #Physical Address Bits* Bits 15-8: #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 (* "ineI", with i in the low eight bits of DL *)
ECX ← 6c65746eh (* "ntel", with n in the low eight bits of CL *)

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

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

IA32_BIOS_SIGN_ID Returns Microcode Update Signature

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

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

When CPUID executes with EAX set to 01H, version information is returned in EAX (see Figure 3-5). 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.

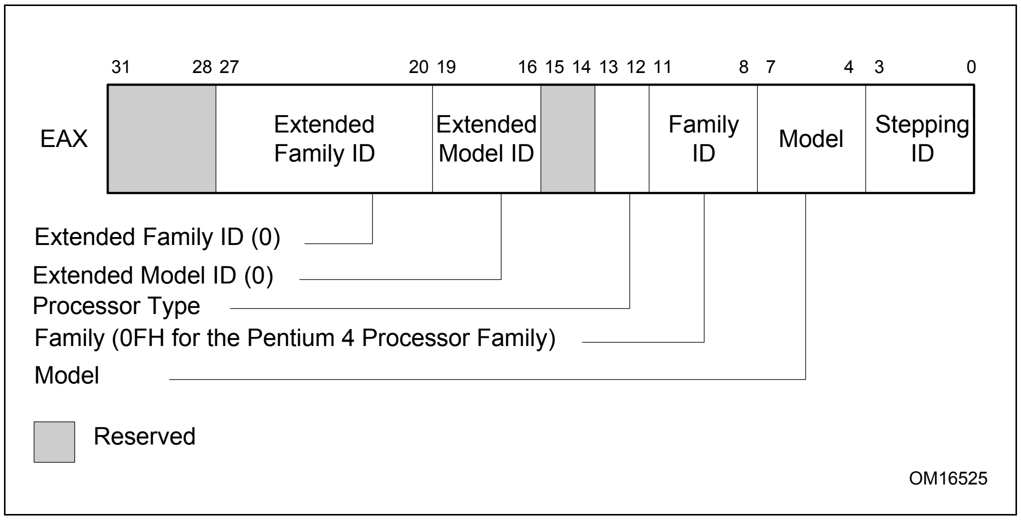


Figure 3-5 Version Information Returned by CPUID in EAX

Table 3-18 Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B
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 = 01H: Returns Additional Information in EBX

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

- Brand index (low byte of EBX) — this number provides an entry into a brand string table that contains brand strings for IA-32 processors. More information about this field is provided later in this section.
- CLFLUSH instruction cache line size (second byte of EBX) — this number indicates the size of the cache line flushed 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 = 01H: Returns Feature Information in ECX and EDX

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

- Figure 3-6 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.

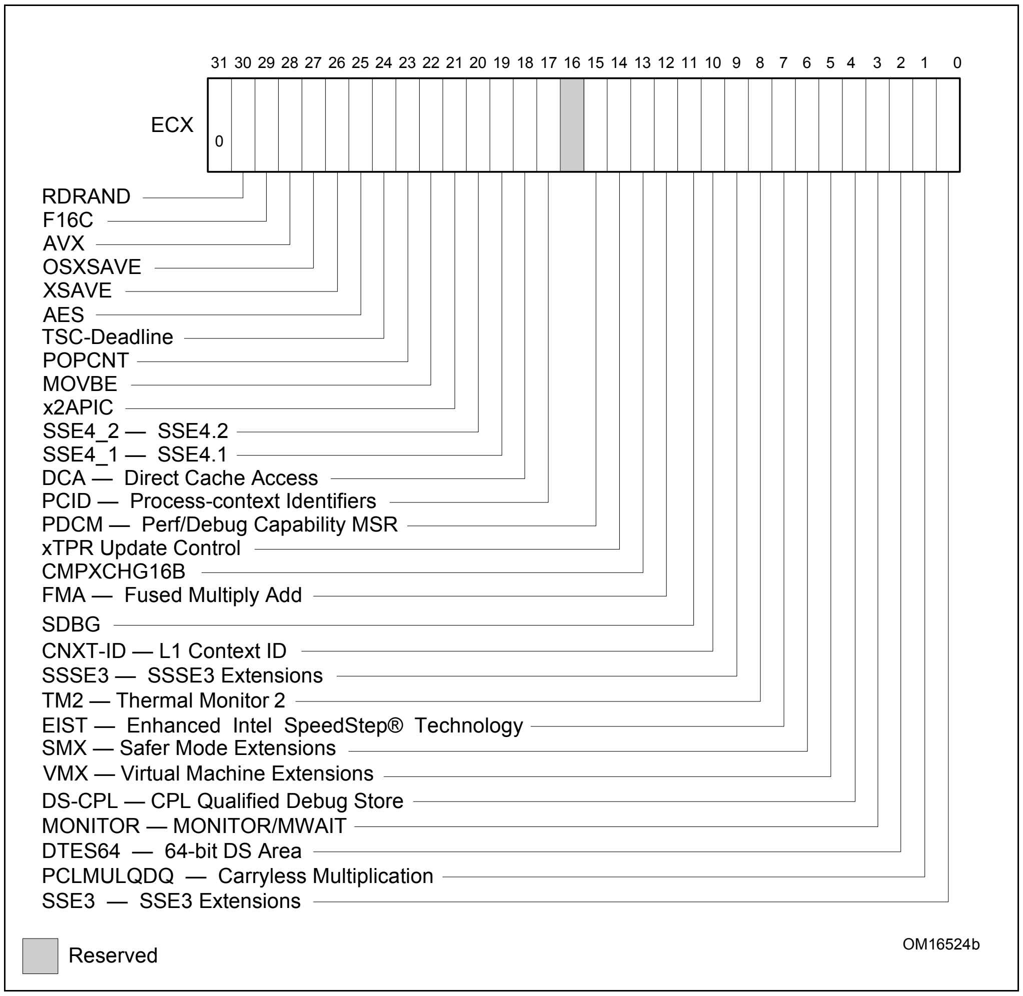


Figure 3-6 Feature Information Returned in the ECX Register

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".

Table 3-19 Feature Information Returned in the ECX Register (Contd.)

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

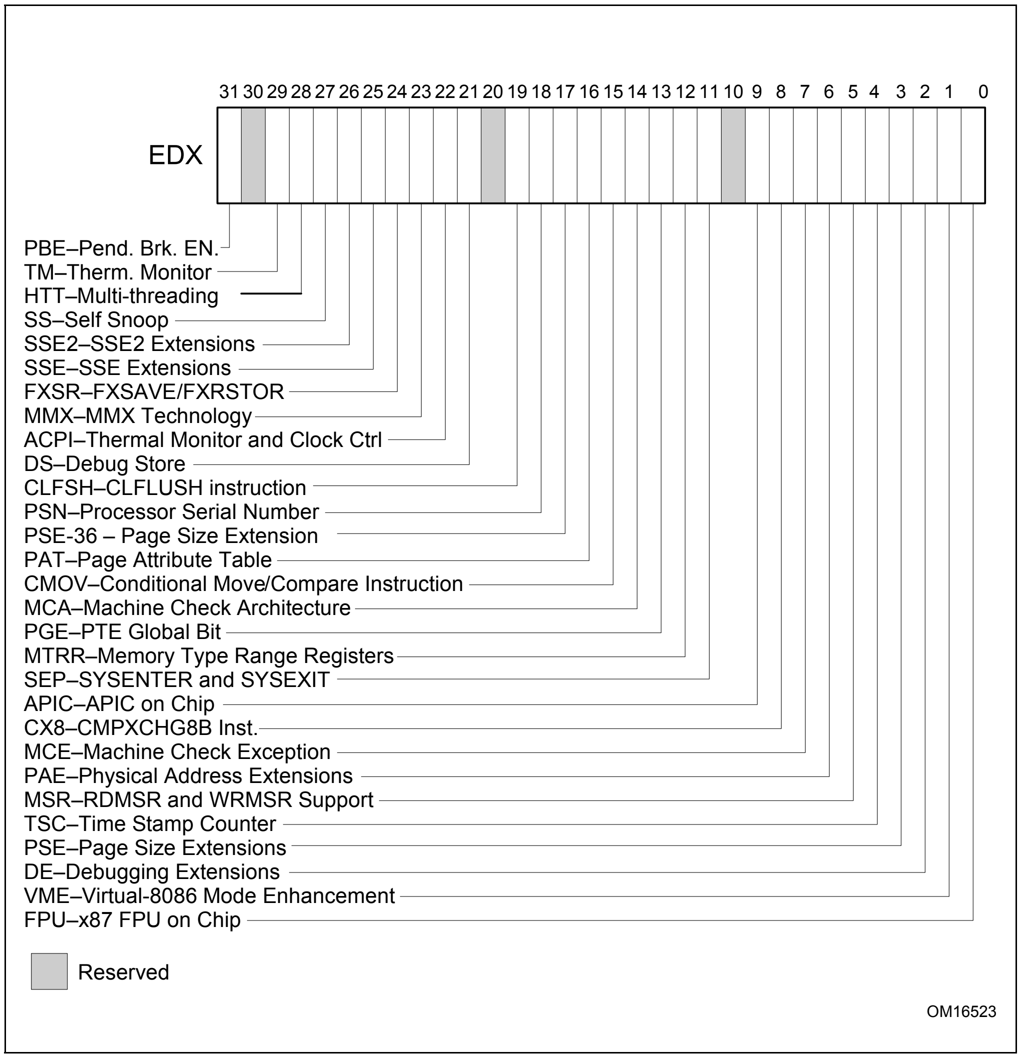


Figure 3-7 Feature Information Returned in the EDX Register

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.

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

Bit #	Mnemonic	Description
3	PSE	**Page Size Extension.** Large pages of size 4 MByte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1.
7	MCE	**Machine Check Exception.** Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	**APIC On-Chip.** The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).
10	Reserved	Reserved
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.
13	PGE	**Page Global Bit.** The global bit is supported in paging-structure entries that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.
14	MCA	**Machine Check Architecture.** 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

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

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

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

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

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors. There are four types of encoding values for the byte descriptor, the encoding type is noted in the second column of Table 3-21. 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
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

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

Value	Type	Description
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries
50H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries
51H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries
52H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries
55H	TLB	Instruction TLB: 2-MByte or 4-MByte pages, fully associative, 7 entries
56H	TLB	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries
57H	TLB	Data TLB0: 4 KByte pages, 4-way associative, 16 entries
59H	TLB	Data TLB0: 4 KByte pages, fully associative, 16 entries
5AH	TLB	Data TLB0: 2-MByte or 4 MByte pages, 4-way set associative, 32 entries
5BH	TLB	Data TLB: 4 KByte and 4 MByte pages, 64 entries
5CH	TLB	Data TLB: 4 KByte and 4 MByte pages,128 entries
5DH	TLB	Data TLB: 4 KByte and 4 MByte pages,256 entries
60H	Cache	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size
61H	TLB	Instruction TLB: 4 KByte pages, fully associative, 48 entries
63H	TLB	Data TLB: 1 GByte pages, 4-way set associative, 4 entries
66H	Cache	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size
67H	Cache	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size
68H	Cache	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size
70H	Cache	Trace cache: 12 K-μop, 8-way set associative
71H	Cache	Trace cache: 16 K-μop, 8-way set associative
72H	Cache	Trace cache: 32 K-μop, 8-way set associative
76H	TLB	Instruction TLB: 2M/4M pages, fully associative, 8 entries
78H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 64byte line size
79H	Cache	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7AH	Cache	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7BH	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7CH	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector
7DH	Cache	2nd-level cache: 2 MByte, 8-way set associative, 64byte line size
7FH	Cache	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size
80H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64-byte line size
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size

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

Value	Type	Description
A0H	DTLB	DTLB: 4k pages, fully associative, 32 entries
B0H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries
B1H	TLB	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries
B2H	TLB	Instruction TLB: 4KByte pages, 4-way set associative, 64 entries
B3H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 128 entries
B4H	TLB	Data TLB1: 4 KByte pages, 4-way associative, 256 entries
B5H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 64 entries
B6H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 128 entries
BAH	TLB	Data TLB1: 4 KByte pages, 4-way associative, 64 entries
C0H	TLB	Data TLB: 4 KByte and 4 MByte pages, 4-way associative, 8 entries
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries
C2H	DTLB	DTLB: 4 KByte/2 MByte pages, 4-way associative, 16 entries
C3H	STLB	Shared 2nd-Level TLB: 4 KByte /2 MByte pages, 6-way associative, 1536 entries. Also 1GBbyte pages, 4-way, 16 entries.
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.

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

When CPUID executes with EAX set to 14H and ECX = 0H, the processor returns information about Intel Processor Trace extensions. See Table 3-17.

When CPUID executes with EAX set to 14H and ECX = n (n > 1and less than the number of non-zero bits in CPUID.(EAX=14H, ECX= 0H).EAX and CPUID.(EAX=0DH, ECX= 0H).EDX), the processor returns information about packet generation in Intel Processor Trace. See Table 3-17.

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

When CPUID executes with EAX set to 15H and ECX = 0H, the processor returns information about Time Stamp Counter/Core Crystal Clock. See Table 3-17.

INPUT EAX = 16H: Returns Processor Frequency Information

When CPUID executes with EAX set to 16H, the processor returns information about Processor Frequency Information. See Table 3-17.

METHODS FOR RETURNING BRANDING INFORMATION

Use the following techniques to access branding information:

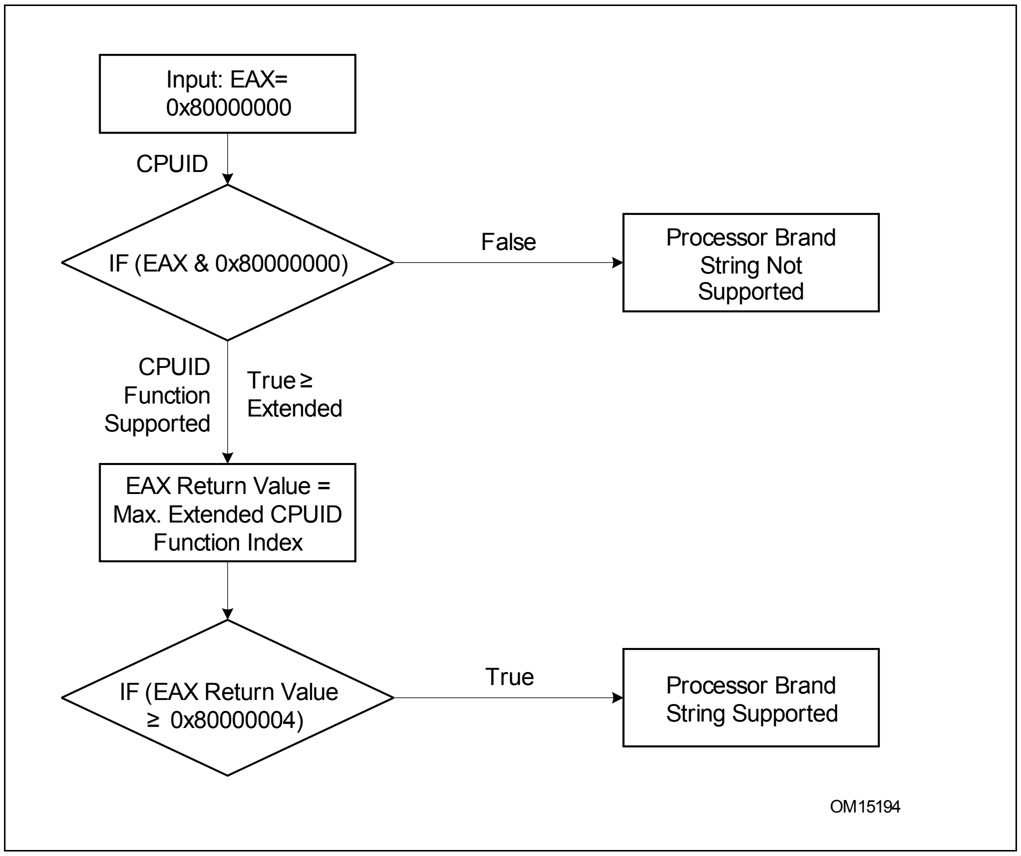
1. Processor brand string method.
2. Processor brand index; this method uses a software supplied brand string table.

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.



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

Figure 3-8 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 "

Table 3-22 Processor Brand String Returned with Pentium 4 Processor (Contd.)

80000003H	EAX = 286C6574H	"(let"
	EBX = 50202952H	"P)R"
	ECX = 69746E65H	"itne"
	EDX = 52286D75H	"R(mu"
80000004H	EAX = 20342029H	" 4)"
	EBX = 20555043H	" UPC"
	ECX = 30303531H	"0051"
	EDX = 007A484DH	"\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.

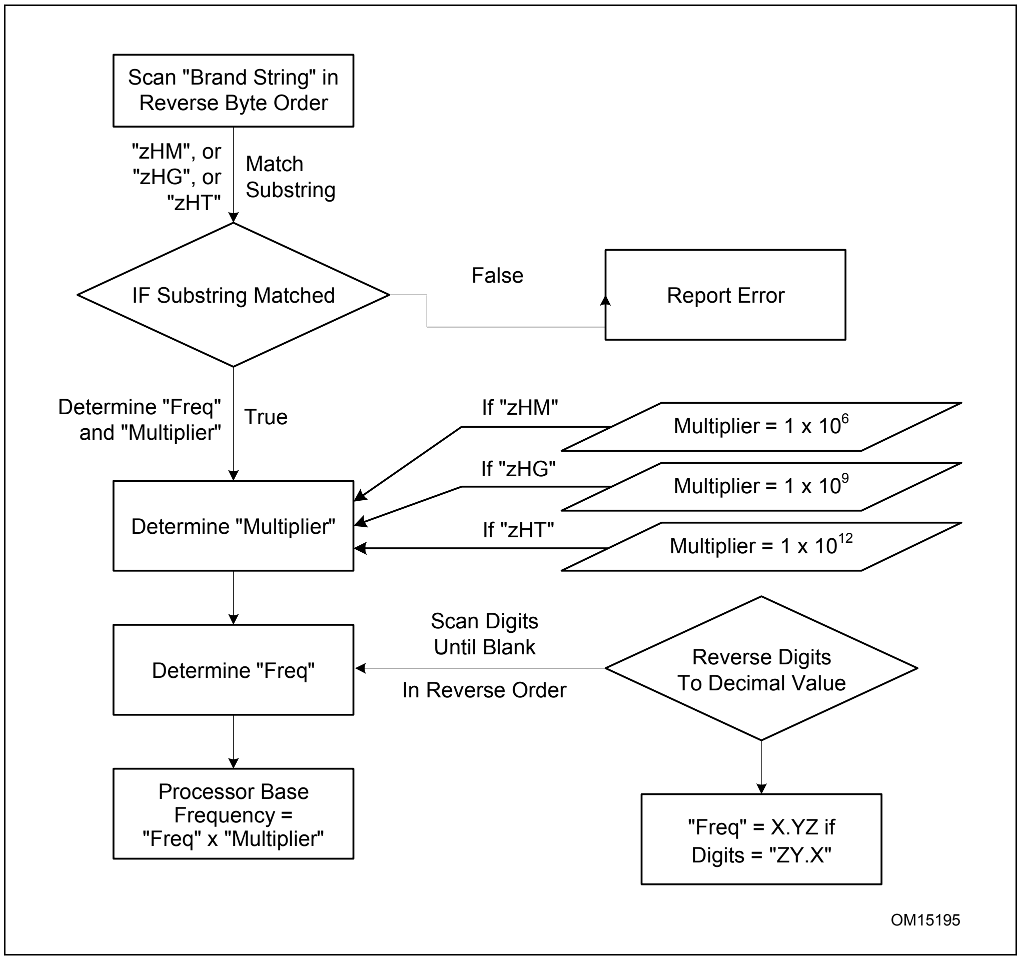

Scan "Brand String" in Reverse Byte Order
"zHM", or "zHG", or "zHT"
Match Substring
IF Substring Matched
False
Report Error
Determine "Freq" and "Multiplier"
True
If "zHM"
Multiplier = 1 x 10^6
If "zHG"
Multiplier = 1 x 10^9
If "zHT"
Multiplier = 1 x 10^{12}
Determine "Multiplier"
Scan Digits Until Blank
In Reverse Order
Determine "Freq"
Reverse Digits To Decimal Value
Processor Base Frequency = "Freq" x "Multiplier"
"Freq" = X.YZ if Digits = "ZY.X"
OM15195

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 = 14H:
            EAX ← Intel Processor Trace Enumeration Leaf; (* See Table 3-17. *)
            EBX ← Intel Processor Trace Enumeration Leaf;
            ECX ← Intel Processor Trace Enumeration Leaf;
            EDX ← Intel Processor Trace Enumeration Leaf;
        BREAK;
    EAX = 15H:
            EAX ← Time Stamp Counter/Core Crystal Clock Information Leaf; (* See Table 3-17. *)
            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;
        EAX = 16H:
            EAX ← Processor Frequency Information Enumeration Leaf; (* See Table 3-17. *)
            EBX ← Processor Frequency Information Enumeration Leaf;
            ECX ← Processor Frequency Information Enumeration Leaf;
            EDX ← Processor Frequency Information Enumeration 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.

...

IRET/IRETD—Interrupt Return

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

Instruction Operand Encoding

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

Description

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

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

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

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

- Return from virtual-8086 mode.
- Return to virtual-8086 mode.

- Intra-privilege level return.
- Inter-privilege level return.
- Return from nested task (task switch).

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

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

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

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

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

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

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

Operation

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

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

```
                CS ← Pop(); (* 16-bit pop *)
                EFLAGS[15:0] ← Pop();
    FI;
    END;

RETURN-FROM-VIRTUAL-8086-MODE:
(* Processor is in virtual-8086 mode when IRET is executed and stays in virtual-8086 mode *)
    IF IOPL = 3 (* Virtual mode: PE = 1, VM = 1, IOPL = 3 *)
        THEN IF OperandSize = 32
            THEN
                EIP ← Pop();
                CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
                EFLAGS ← Pop();
                (* VM, IOPL,VIP and VIF EFLAG bits not modified by pop *)
                IF EIP not within CS limit
                    THEN #GP(0); FI;
            ELSE (* OperandSize = 16 *)
                EIP ← Pop(); (* 16-bit pop; clear upper 16 bits *)
                CS ← Pop(); (* 16-bit pop *)
                EFLAGS[15:0] ← Pop(); (* IOPL in EFLAGS not modified by pop *)
                IF EIP not within CS limit
                    THEN #GP(0); FI;
            FI;
        ELSE
            #GP(0); (* Trap to virtual-8086 monitor: PE = 1, VM = 1, IOPL < 3 *)
    FI;
END;

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

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

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

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

RETURN-TO-OUTER-PRIVILEGE-LEVEL:
    IF new mode ≠ 64-Bit Mode
        THEN
            IF EIP is not within CS limit
                THEN #GP(0); FI;
        ELSE (* new mode = 64-bit mode *)
            IF RIP is non-canonical
                    THEN #GP(0); FI;
    FI;
    EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
    IF OperandSize = 32
        THEN EFLAGS(RF, AC, ID) ← tempEFLAGS; FI;
    IF CPL ≤ IOPL
        THEN EFLAGS(IF) ← tempEFLAGS; FI;
    IF CPL = 0
        THEN
            EFLAGS(IOPL) ← tempEFLAGS;
            IF OperandSize = 32
                THEN EFLAGS(VM, VIF, VIP) ← tempEFLAGS; FI;
            IF OperandSize = 64
                THEN EFLAGS(VIF, VIP) ← tempEFLAGS; FI;
    FI;
    CPL ← CS(RPL);
    FOR each SegReg in (ES, FS, GS, and DS)
        DO
```

```
                tempDesc ← descriptor cache for SegReg (* hidden part of segment register *)
                IF tempDesc(DPL) < CPL AND tempDesc(Type) is data or non-conforming code
                    THEN (* Segment register invalid *)
                        SegReg ← NULL;
                FI;
            OD;
END;

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

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

Flags Affected

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

Protected Mode Exceptions

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

#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

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

Other exceptions same as in Protected Mode.

64-Bit Mode Exceptions

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

#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.

...

12. 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.

...

RDPKRU—Read Protection Key Rights for User Pages

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

Instruction Operand Encoding

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

Description

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

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

In 64-bit mode, bits 63:32 of RCX are ignored, and RDPKRU clears bits 63:32 of each of RDX and RAX.

Operation

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

Flags Affected

None.

Protected Mode Exceptions

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

If CR4.PKE = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

RDTSC—Read Time-Stamp Counter

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

Instruction Operand Encoding

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

Description

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

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

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

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

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

This instruction was introduced by the Pentium processor.

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

Operation

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

Flags Affected

None.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

RDTSCP—Read Time-Stamp Counter and Processor ID

Opcode*	**Instruction**	**Op/ En**	**64-Bit Mode**	**Compat/ Leg Mode**	**Description**
0F 01 F9	RDTSCP	NP	Valid	Valid	Read 64-bit time-stamp counter and 32-bit IA32_TSC_AUX value into EDX:EAX and ECX.

Instruction Operand Encoding

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

Description

Loads the current value of the processor's time-stamp counter (a 64-bit MSR) into the EDX:EAX registers and also loads the IA32_TSC_AUX MSR (address C000_0103H) into the ECX register. The EDX register is loaded with the high-order 32 bits of the IA32_TSC MSR; the EAX register is loaded with the low-order 32 bits of the IA32_TSC MSR; and the ECX register is loaded with the low-order 32-bits of IA32_TSC_AUX MSR. On processors that support the Intel 64 architecture, the high-order 32 bits of each of RAX, RDX, and RCX are cleared.

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

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

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

The presence of the RDTSCP instruction is indicated by CPUID leaf 80000001H, EDX bit 27. If the bit is set to 1 then RDTSCP is present on the processor.

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

Operation

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

Flags Affected

None.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

VPGATHERDD/VPGATHERQD — Gather Packed Dword Values Using Signed Dword/Qword Indices

Opcode/ Instruction	Op/ En	64/ 32-bit Mode	CPUID Feature Flag	Description
VEX.DDS.128.66.0F38.W0 90 /r VPGATHERDD *xmm1, vm32x, xmm2*	RMV	V/V	AVX2	Using dword indices specified in *vm32x*, gather dword values from memory conditioned on mask specified by *xmm2*. Conditionally gathered elements are merged into *xmm1*.
VEX.DDS.128.66.0F38.W0 91 /r VPGATHERQD *xmm1, vm64x, xmm2*	RMV	V/V	AVX2	Using qword indices specified in *vm64x*, gather dword values from memory conditioned on mask specified by *xmm2*. Conditionally gathered elements are merged into *xmm1*.
VEX.DDS.256.66.0F38.W0 90 /r VPGATHERDD *ymm1, vm32y, ymm2*	RMV	V/V	AVX2	Using dword indices specified in *vm32y*, gather dword from memory conditioned on mask specified by *ymm2*. Conditionally gathered elements are merged into *ymm1*.
VEX.DDS.256.66.0F38.W0 91 /r VPGATHERQD *xmm1, vm64y, xmm2*	RMV	V/V	AVX2	Using qword indices specified in *vm64y*, gather dword values from memory conditioned on mask specified by *xmm2*. Conditionally gathered elements are merged into *xmm1*.

Instruction Operand Encoding

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

Description

The instruction conditionally loads up to 4 or 8 dword values from memory addresses specified by the memory operand (the second operand) and using dword indices. The memory operand uses the VSIB form of the SIB byte to specify a general purpose register operand as the common base, a vector register for an array of indices relative to the base and a constant scale factor.

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

Using qword indices, the instruction conditionally loads up to 2 or 4 qword values from the VSIB addressing memory operand, and updates the lower half of the destination register. The upper 128 or 256 bits of the destination register are zero'ed with qword indices.

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

If the data size and index size are different, part of the destination register and part of the mask register do not correspond to any elements being gathered. This instruction sets those parts to zero. It may do this to one or both of those registers even if the instruction triggers an exception, and even if the instruction triggers the exception before gathering any elements.

VEX.128 version: For dword indices, the instruction will gather four dword values. For qword indices, the instruction will gather two values and zeroes the upper 64 bits of the destination.

VEX.256 version: For dword indices, the instruction will gather eight dword values. For qword indices, the instruction will gather four values and zeroes the upper 128 bits of the destination.

Note that:

- If any pair of the index, mask, or destination registers are the same, this instruction results a UD fault.
- The values may be read from memory in any order. Memory ordering with other instructions follows the Intel-64 memory-ordering model.
- Faults are delivered in a right-to-left manner. That is, if a fault is triggered by an element and delivered, all elements closer to the LSB of the destination will be completed (and non-faulting). Individual elements closer to the MSB may or may not be completed. If a given element triggers multiple faults, they are delivered in the conventional order.
- Elements may be gathered in any order, but faults must be delivered in a right-to-left order; thus, elements to the left of a faulting one may be gathered before the fault is delivered. A given implementation of this instruction is repeatable - given the same input values and architectural state, the same set of elements to the left of the faulting one will be gathered.
- This instruction does not perform AC checks, and so will never deliver an AC fault.
- This instruction will cause a #UD if the address size attribute is 16-bit.
- This instruction will cause a #UD if the memory operand is encoded without the SIB byte.
- This instruction should not be used to access memory mapped I/O as the ordering of the individual loads it does is implementation specific, and some implementations may use loads larger than the data element size or load elements an indeterminate number of times.
- The scaled index may require more bits to represent than the address bits used by the processor (e.g., in 32-bit mode, if the scale is greater than one). In this case, the most significant bits beyond the number of address bits are ignored.

Operation

```
DEST ← SRC1;
BASE_ADDR: base register encoded in VSIB addressing;
VINDEX: the vector index register encoded by VSIB addressing;
SCALE: scale factor encoded by SIB:[7:6];
DISP: optional 1, 4 byte displacement;
MASK ← SRC3;
```

VPGATHERDD (VEX.128 version)

```
FOR j← 0 to 3
    i ← j * 32;
    IF MASK[31+i] THEN
        MASK[i +31:i] ← FFFFFFFFH; // extend from most significant bit
    ELSE
        MASK[i +31:i] ← 0;
    FI;
ENDFOR
MASK[VLMAX-1:128] ← 0;
FOR j← 0 to 3
    i ← j * 32;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX[i+31:i])*SCALE + DISP;
    IF MASK[31+i] THEN
        DEST[i +31:i] ← FETCH_32BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +31:i] ← 0;
ENDFOR
DEST[VLMAX-1:128] ← 0;
(non-masked elements of the mask register have the content of respective element cleared)
```

VPGATHERQD (VEX.128 version)

```
FOR j← 0 to 3
    i ← j * 32;
    IF MASK[31+i] THEN
        MASK[i +31:i] ← FFFFFFFFH; // extend from most significant bit
    ELSE
        MASK[i +31:i] ← 0;
    FI;
ENDFOR
MASK[VLMAX-1:128] ← 0;
FOR j← 0 to 1
    k ← j * 64;
    i ← j * 32;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX1[k+63:k])*SCALE + DISP;
    IF MASK[31+i] THEN
        DEST[i +31:i] ← FETCH_32BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +31:i] ← 0;
ENDFOR
MASK[127:64] ← 0;
DEST[VLMAX-1:64] ← 0;
(non-masked elements of the mask register have the content of respective element  cleared)
```

VPGATHERDD (VEX.256 version)

```
FOR j← 0 to 7
    i ← j * 32;
    IF MASK[31+i] THEN
        MASK[i +31:i] ← FFFFFFFFH; // extend from most significant bit
    ELSE
        MASK[i +31:i] ← 0;
    FI;
ENDFOR
FOR j← 0 to 7
    i ← j * 32;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX1[i+31:i])*SCALE + DISP;
    IF MASK[31+i] THEN
        DEST[i +31:i] ← FETCH_32BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +31:i] ← 0;
ENDFOR
(non-masked elements of the mask register have the content of respective element cleared)
```

VPGATHERQD (VEX.256 version)

```
FOR j← 0 to 7
    i ← j * 32;
    IF MASK[31+i] THEN
         MASK[i +31:i] ← FFFFFFFFH; // extend from most significant bit
    ELSE
         MASK[i +31:i] ← 0;
    FI;
ENDFOR
FOR j← 0 to 3
    k ← j * 64;
    i ← j * 32;
    DATA_ADDR ← BASE_ADDR + (SignExtend(VINDEX1[k+63:k])*SCALE + DISP;
    IF MASK[31+i] THEN
         DEST[i +31:i] ← FETCH_32BITS(DATA_ADDR); // a fault exits the instruction
    FI;
    MASK[i +31:i] ← 0;
ENDFOR
MASK[VLMAX-1:128] ← 0;
DEST[VLMAX-1:128] ← 0;
(non-masked elements of the mask register have the content of respective element  cleared)
```

Intel C/C++ Compiler Intrinsic Equivalent

VPGATHERDD: __m128i _mm_i32gather_epi32 (int const * base, __m128i index, const int scale);

VPGATHERDD: __m128i _mm_mask_i32gather_epi32 (__m128i src, int const * base, __m128i index, __m128i mask, const int scale);

VPGATHERDD: __m256i _mm256_i32gather_epi32 (int const * base, __m256i index, const int scale);

VPGATHERDD: __m256i _mm256_mask_i32gather_epi32 (__m256i src, int const * base, __m256i index, __m256i mask, const int scale);

VPGATHERQD: __m128i _mm_i64gather_epi32 (int const * base, __m128i index, const int scale);

VPGATHERQD: __m128i _mm_mask_i64gather_epi32 (__m128i src, int const * base, __m128i index, __m128i mask, const int scale);

VPGATHERQD: __m128i _mm256_i64gather_epi32 (int const * base, __m256i index, const int scale);

VPGATHERQD: __m128i _mm256_mask_i64gather_epi32 (__m128i src, int const * base, __m256i index, __m128i mask, const int scale);

SIMD Floating-Point Exceptions

None

Other Exceptions

See Exceptions Type 12

...

WRPKRU—Write Data to User Page Key Register

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

Instruction Operand Encoding

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

Description

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

WRPKRU can be executed only if CR4.PKE = 1; otherwise, a general-protection exception (#GP) occurs. Software can discover the value of CR4.PKE by examining CPUID.(EAX=07H,ECX=0H):ECX.OSPKE [bit 4].

In 64-bit mode, WRPKRU ignores bits 63:32 of each of RAX, RCX, and RDX.

Operation

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

Flags Affected

None.

Protected Mode Exceptions

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

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

13. 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.

...

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

See also:

- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Fortran Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Software Development Tools:
  http://www.intel.com/cd/software/products/asmo-na/eng/index.htm
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in three or seven volumes):
  http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
  http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html
- Intel 64 Architecture x2APIC Specification:

  http://www.intel.com/content/www/us/en/architecture-and-technology/64-architecture-x2apic-specification.html
- Intel® Trusted Execution Technology Measured Launched Environment Programming Guide:

  http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html
- Developing Multi-threaded Applications: A Platform Consistent Approach:
  https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
  http://software.intel.com/en-us/articles/ap949-using-spin-loops-on-intel-pentiumr-4-processor-and-intel-xeonr-processor/
- Performance Monitoring Unit Sharing Guide
  http://software.intel.com/file/30388

Literature related to selected features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference
  https://software.intel.com/en-us/isa-extensions
- Intel® Software Guard Extensions (Intel® SGX) Programming Reference
  https://software.intel.com/en-us/isa-extensions/intel-sgx

More relevant links are:

- Intel® Developer Zone:
  https://software.intel.com/en-us
- Developer centers:
  http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html
- Processor support general link:
  http://www.intel.com/support/processors/
- Software products and packages:
  http://www.intel.com/cd/software/products/asmo-na/eng/index.htm
- Intel® Hyper-Threading Technology (Intel® HT Technology):
  http://www.intel.com/technology/platform-technology/hyper-threading/index.htm

...

14. 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.

...

2.1 OVERVIEW OF THE SYSTEM-LEVEL ARCHITECTURE

System-level architecture consists of a set of registers, data structures, and instructions designed to support basic system-level operations such as memory management, interrupt and exception handling, task management, and control of multiple processors.

Figure 2-1 provides a summary of system registers and data structures that applies to 32-bit modes. System registers and data structures that apply to IA-32e mode are shown in Figure 2-2.

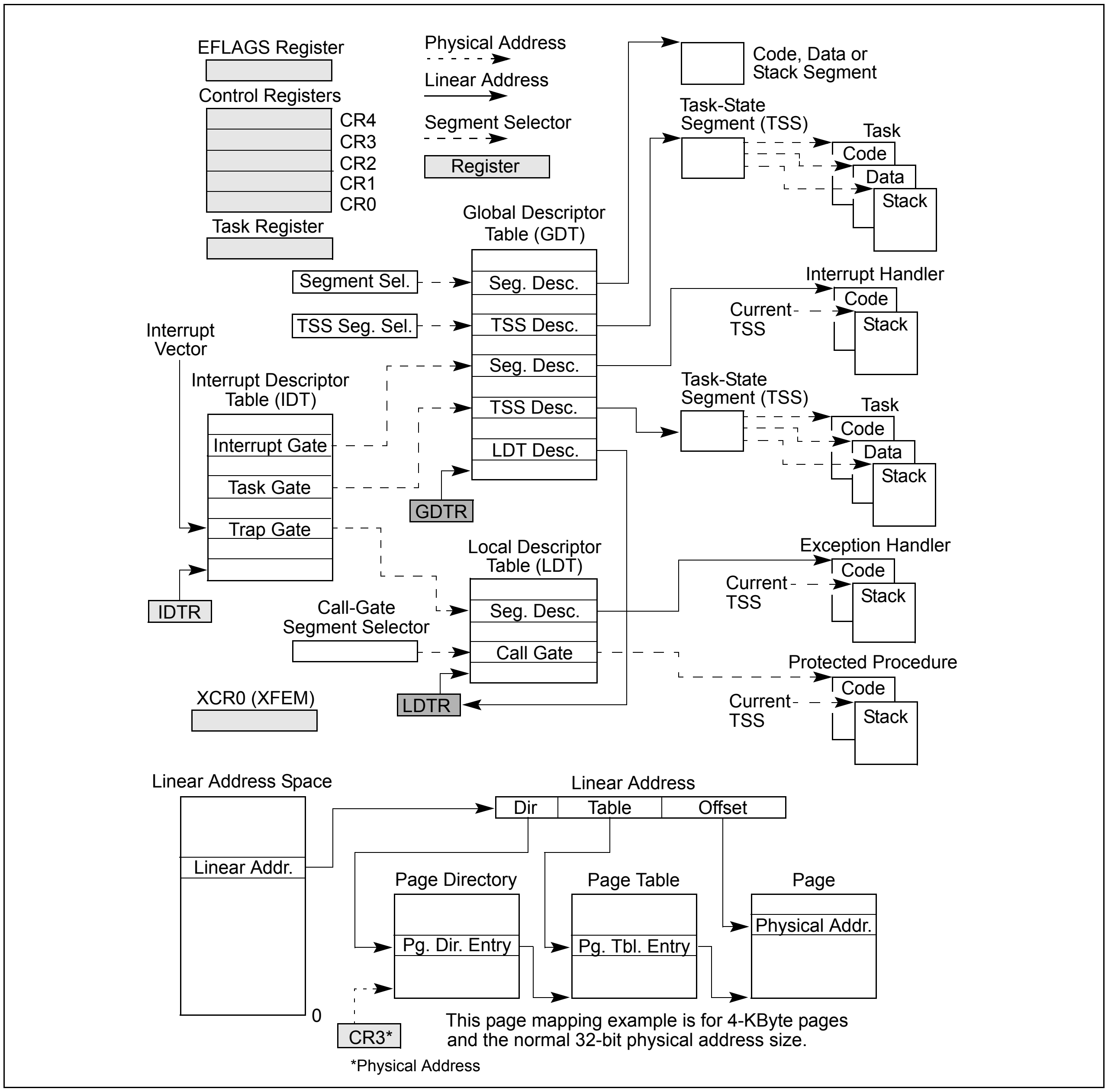


Figure 2-1 IA-32 System-Level Registers and Data Structures

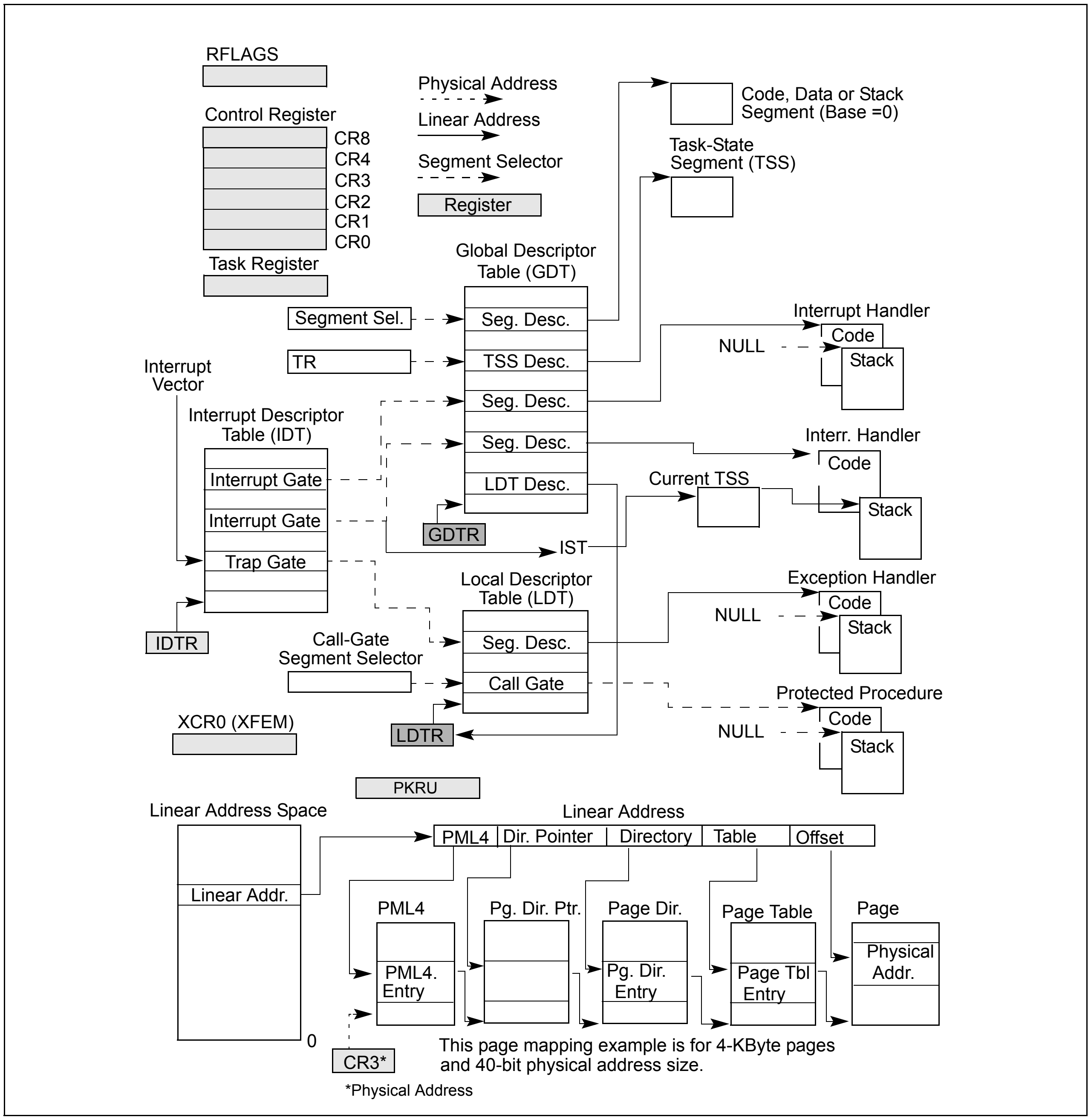


Figure 2-2 System-Level Registers and Data Structures in IA-32e Mode

...

2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-7) determine operating mode of the processor and the characteristics of the currently executing task. These registers are 32 bits in all 32-bit modes and compatibility mode.

In 64-bit mode, control registers are expanded to 64 bits. The MOV CRn instructions are used to manipulate the register bits. Operand-size prefixes for these instructions are ignored. The following is also true:

- Bits 63:32 of CR0 and CR4 are reserved and must be written with zeros. Writing a nonzero value to any of the upper 32 bits results in a general-protection exception, #GP(0).
- All 64 bits of CR2 are writable by software.
- Bits 51:40 of CR3 are reserved and must be 0.
- The MOV CRn instructions do not check that addresses written to CR2 and CR3 are within the linear-address or physical-address limitations of the implementation.
- Register CR8 is available in 64-bit mode only.

The control registers are summarized below, and each architecturally defined control field in these control registers are described individually. In Figure 2-7, the width of the register in 64-bit mode is indicated in parenthesis (except for CR0).

- **CR0** — Contains system control flags that control operating mode and states of the processor.
- **CR1** — Reserved.
- **CR2** — Contains the page-fault linear address (the linear address that caused a page fault).
- **CR3** — Contains the physical address of the base of the paging-structure hierarchy and two flags (PCD and PWT). Only the most-significant bits (less the lower 12 bits) of the base address are specified; the lower 12 bits of the address are assumed to be 0. The first paging structure must thus be aligned to a page (4-KByte) boundary. The PCD and PWT flags control caching of that paging structure in the processor's internal data caches (they do not control TLB caching of page-directory information).

  When using the physical address extension, the CR3 register contains the base address of the page-directory-pointer table In IA-32e mode, the CR3 register contains the base address of the PML4 table.

  See also: Chapter 4, "Paging."
- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities. The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.

- **CR8** — Provides read and write access to the Task Priority Register (TPR). It specifies the priority threshold value that operating systems use to control the priority class of external interrupts allowed to interrupt the processor. This register is available only in 64-bit mode. However, interrupt filtering continues to apply in compatibility mode.

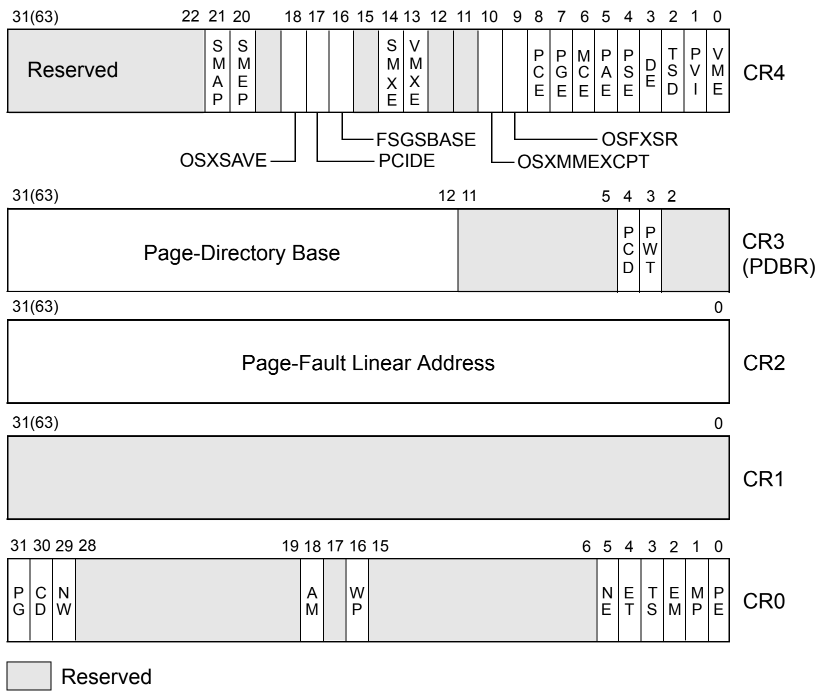


Figure 2-7 Control Registers

When loading a control register, reserved bits should always be set to the values previously read. The flags in control registers are:

PG **Paging (bit 31 of CR0)** — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 4, “Paging.”

On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.

CD **Cache Disable (bit 30 of CR0)** — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor’s internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 11-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 11.5.3, “Preventing Caching,” and Section 11.5, “Cache Control.”

NW **Not Write-through (bit 29 of CR0)** — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 11-5 for detailed information about the affect of the NW flag on caching for other settings of the CD and NW flags.

AM **Alignment Mask (bit 18 of CR0)** — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

WP **Write Protect (bit 16 of CR0)** — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting; see Section 4.1.3 and Section 4.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX.

NE **Numeric Error (bit 5 of CR0)** — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.

See also: "Software Exception Handling" in Chapter 8, "Programming with the x87 FPU," and Appendix A, "EFLAGS Cross-Reference," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

ET **Extension Type (bit 4 of CR0)** — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

TS **Task Switched (bit 3 of CR0)** — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

- If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/ SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.
- If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.
- If the EM flag is set, the setting of the TS flag has no affect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

Table 2-2 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 12-1 and 13-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.

The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE /SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).

The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3//SSSE3/SSE4 instruction; the x87 FPU/MMX/SSE/SSE2/ SSE3/SSSE3/SSE4 context is never saved.

Table 2-2 Action Taken By x87 FPU Instructions for Different Combinations of EM, MP, and TS

CR0 Flags			x87 FPU Instruction Type	
EM	MP	TS	Floating-Point	WAIT/FWAIT
0	0	0	Execute	Execute.
0	0	1	#NM Exception	Execute.
0	1	0	Execute	Execute.
0	1	1	#NM Exception	#NM exception.
1	0	0	#NM Exception	Execute.
1	0	1	#NM Exception	Execute.
1	1	0	#NM Exception	Execute.
1	1	1	#NM Exception	#NM exception.

EM **Emulation (bit 2 of CR0)** — Indicates that the processor does not have an internal or external x87 FPU when set; indicates an x87 FPU is present when clear. This flag also affects the execution of MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

When the EM flag is set, execution of an x87 FPU instruction generates a device-not-available exception (#NM). This flag must be set when the processor does not have an internal x87 FPU or is not connected to an external math coprocessor. Setting this flag forces all floating-point instructions to be handled by software emulation. Table 9-2 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the EM, MP, and TS flags.

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 12-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see Table 13-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

MP **Monitor Coprocessor (bit 1 of CR0).** — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CR0). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 9-2 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the MP, EM, and TS flags.

PE **Protection Enable (bit 0 of CR0)** — Enables protected mode when set; enables real-address mode when clear. This flag does not enable paging directly. It only enables segment-level protection. To enable paging, both the PE and PG flags must be set.

See also: Section 9.9, "Mode Switching."

PCD **Page-level Cache Disable (bit 4 of CR3)** — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, "Paging and Memory Typing". This bit is not used if paging is disabled, with PAE paging, or with IA-32e paging if CR4.PCIDE=1.

PWT **Page-level Write-Through (bit 3 of CR3)** — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, "Paging and Memory Typing". This bit is not used if paging is disabled, with PAE paging, or with IA-32e paging if CR4.PCIDE=1.

VME **Virtual-8086 Mode Extensions (bit 0 of CR4)** — Enables interrupt- and exception-handling extensions in virtual-8086 mode when set; disables the extensions when clear. Use of the virtual mode extensions can improve the performance of virtual-8086 applications by eliminating the overhead of calling the virtual-8086 monitor to handle interrupts and exceptions that occur while executing an 8086 program and, instead, redirecting the interrupts and exceptions back to the 8086 program's handlers. It also provides hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multitasking and multiple-processor environments.

See also: Section 20.3, "Interrupt and Exception Handling in Virtual-8086 Mode."

PVI **Protected-Mode Virtual Interrupts (bit 1 of CR4)** — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.

See also: Section 20.4, "Protected-Mode Virtual Interrupts."

TSD **Time Stamp Disable (bit 2 of CR4)** — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported (if CPUID.80000001H:EDX[27] = 1).

DE **Debugging Extensions (bit 3 of CR4)** — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.

See also: Section 17.2.2, "Debug Registers DR4 and DR5."

PSE **Page Size Extensions (bit 4 of CR4)** — Enables 4-MByte pages with 32-bit paging when set; restricts 32-bit paging to pages to 4 KBytes when clear.

See also: Section 4.3, "32-Bit Paging."

PAE **Physical Address Extension (bit 5 of CR4)** — When set, enables paging to produce physical addresses with more than 32 bits. When clear, restricts physical addresses to 32 bits. PAE must be set before entering IA-32e mode.

See also: Chapter 4, "Paging."

MCE **Machine-Check Enable (bit 6 of CR4)** — Enables the machine-check exception when set; disables the machine-check exception when clear.

See also: Chapter 15, "Machine-Check Architecture."

PGE **Page Global Enable (bit 7 of CR4)** — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory or page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.

When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.

See also: Section 4.10, "Caching Translation Information."

PCE **Performance-Monitoring Counter Enable (bit 8 of CR4)** — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.

OSFXSR

Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4) — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX instructions, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID feature flags FXSR indicates availability of the FXSAVE/FXRSTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRSTOR to save/restore the contents of the X87 FPU, XMM and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

OSXMMEXCPT

Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4) — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (#XM) is generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

VMXE

VMX-Enable Bit (bit 13 of CR4) — Enables VMX operation when set. See Chapter 23, “Introduction to Virtual-Machine Extensions.”

SMXE

SMX-Enable Bit (bit 14 of CR4) — Enables SMX operation when set. See Chapter 5, “Safer Mode Extensions Reference” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C*.

FSGSBASE

FSGSBASE-Enable Bit (bit 16 of CR4) — Enables the instructions RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE.

PCIDE

PCID-Enable Bit (bit 17 of CR4) — Enables process-context identifiers (PCIDs) when set. See Section 4.10.1, “Process-Context Identifiers (PCIDs)”. Can be set only in IA-32e mode (if IA32_EFER.LMA = 1).

OSXSAVE

XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4) — When set, this flag: (1) indicates (via CPUID.01H:ECX.OSXSAVE[bit 27]) that the operating system supports the use of the XGETBV, XSAVE and XRSTOR instructions by general software; (2) enables the XSAVE and XRSTOR instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in XCR0; (3) enables the processor to execute XGETBV and XSETBV instructions in order to read and write XCR0. See Section 2.6 and Chapter 13, “System Programming for Instruction Set Extensions and Processor Extended States”.

SMEP

SMEP-Enable Bit (bit 20 of CR4) — Enables supervisor-mode execution prevention (SMEP) when set. See Section 4.6, “Access Rights”.

SMAP

SMAP-Enable Bit (bit 21 of CR4) — Enables supervisor-mode access prevention (SMAP) when set. See Section 4.6, "Access Rights."

PKE

Protection-Key-Enable Bit (bit 22 of CR4) — Enables IA-32e paging to associate each linear address with a protection key. The PKRU register specifies, for each protection key, whether user-mode linear addresses with that protection key can be read or written. This bit also enables access to the PKRU register using the RDPKRU and WRPKRU instructions.

TPL

Task Priority Level (bit 3:0 of CR8) — This sets the threshold value corresponding to the highest-priority interrupt to be blocked. A value of 0 means all interrupts are enabled. This field is available in 64-bit mode. A value of 15 means all interrupts will be disabled.

...

2.6 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.

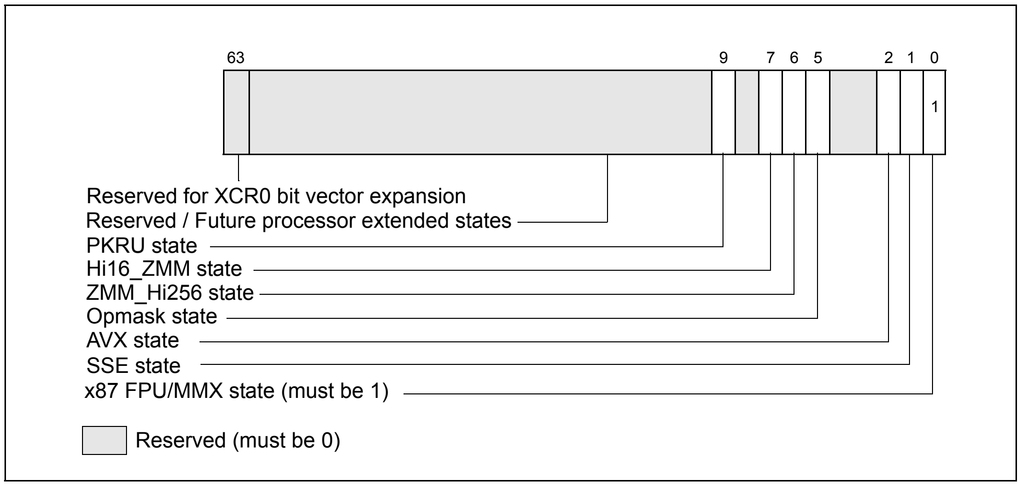

63
9
7
6
5
2
1
0
1
Reserved for XCR0 bit vector expansion
Reserved / Future processor extended states
PKRU state
Hi16_ZMM state
ZMM_Hi256 state
Opmask state
AVX state
SSE state
x87 FPU/MMX state (must be 1)
Reserved (must be 0)

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).
- XCR0.opmask (bit 5): If 1, AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the opmask registers k0–k7.
- XCR0.ZMM_Hi256 (bit 6): If 1, AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the lower ZMM registers (ZMM0-ZMM15 in 64-bit mode; otherwise ZMM0-ZMM7).
- XCR0.Hi16_ZMM (bit 7): If 1, AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper ZMM registers (ZMM16-ZMM31, only in 64-bit mode).
- XCR0.PKRU (bit 9): If 1, the XSAVE feature set can be used to manage the PKRU register (see Section 2.7).

An attempt to use XSETBV to write to XCR0 results in general-protection exceptions (#GP) if it would do any of the following:

- 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);
- clear XCR0.x87;
- clear XCR0.SSE and set XCR0.AVX;
- clear XCR0.AVX and set any of XCR0.opmask, XCR0.ZMM_Hi256, and XCR0.Hi16_ZMM; or
- set any of XCR0.opmask, XCR0.ZMM_Hi256, and XCR0.Hi16_ZMM while not setting all of them.

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

2.7 PROTECTION KEY RIGHTS REGISTER (PKRU)

If CPUID.(EAX=07H,ECX=0H):ECX.PK [bit 3] = 1, the processor supports the protection-key feature for IA-32e paging. The feature allows selective protection of user-mode pages depending on the 4-bit protection key assigned to each page. The **protection key rights register for user pages** (PKRU) allows software to specify the access rights for each protection key.

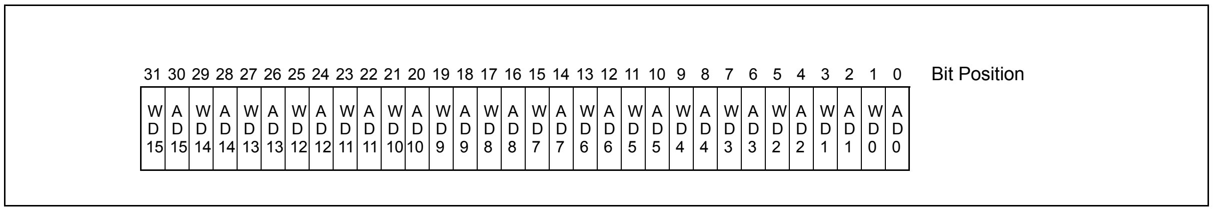

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit Position
WD15 AD15 WD14 AD14 WD13 AD13 WD12 AD12 WD11 AD11 WD10 AD10 WD9 AD9 WD8 AD8 WD7 AD7 WD6 AD6 WD5 AD5 WD4 AD4 WD3 AD3 WD2 AD2 WD1 AD1 WD0 AD0

Figure 2-9 Protection Key Rights Register for User Pages (PKRU)

The layout of the PKRU register is shown in Figure 2-9. It contains 16 pairs of disable controls to prevent data accesses to user-mode linear addresses based on their protection keys. Each protection key *i* is associated with two bits in the PKRU register:

- Bit 2*i*, shown as “AD*i*” (access disable): if set, the processor prevents any data accesses to user-mode linear addresses with protection key *i*.
- Bit 2*i*+1, shown as “WD*i*” (write disable): if set, the processor prevents write accesses to user-mode linear addresses with protection key *i*.

See Section 4.6.2, “Protection Keys,” for details of how the processor uses the PKRU register to control accesses to user-mode linear addresses.

...

15. 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, SMAP, and PKE flags in control register CR4 (bit 4, bit 5, bit 7, bit 17, bit 20, bit 21, and bit 22, 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, CR4.PKE, 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, CR4.PKE, 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. With IA-32e paging, software can enable a facility by which a logical processor caches information for multiple linear-address spaces. The processor may retain cached information when software switches between different linear-address spaces.
- Support for protection keys. With IA-32e paging, software can enable a facility by which each linear address is associated with a **protection key**. Software can use a new control register to determine, for each protection keys, how software can access linear addresses associated with that protection key.

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 and protection keys?
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]

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. Protection keys are used only if certain conditions hold; see Section 4.6.2.

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:

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.

- 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, SMAP, and PKE flags in CR4 (bit 4, bit 7, bit 17, bit 20, bit 21, and bit 22 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.

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

CR4.PKE allows each linear address to be associated with a **protection key**. The PKRU register specifies, for each protection key, whether linear addresses with that protection key can be read or written by software. See Section 4.6 for more information.

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.)

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.

4.1.4 Enumeration of Paging Features by CPUID

Software can discover support for different paging features using the CPUID instruction:

- PSE: page-size extensions for 32-bit paging.
  If CPUID.01H:EDX.PSE [bit 3] = 1, CR4.PSE may be set to 1, enabling support for 4-MByte pages with 32-bit paging (see Section 4.3).
- PAE: physical-address extension.
  If CPUID.01H:EDX.PAE [bit 6] = 1, CR4.PAE may be set to 1, enabling PAE paging (this setting is also required for IA-32e paging).
- PGE: global-page support.
  If CPUID.01H:EDX.PGE [bit 13] = 1, CR4.PGE may be set to 1, enabling the global-page feature (see Section 4.10.2.4).
- PAT: page-attribute table.
  If CPUID.01H:EDX.PAT [bit 16] = 1, the 8-entry page-attribute table (PAT) is supported. When the PAT is supported, three bits in certain paging-structure entries select a memory type (used to determine type of caching used) from the PAT (see Section 4.9.2).
- PSE-36: page-size extensions with 40-bit physical-address extension.
  If CPUID.01H:EDX.PSE-36 [bit 17] = 1, the PSE-36 mechanism is supported, indicating that translations using 4-MByte pages with 32-bit paging may produce physical addresses with up to 40 bits (see Section 4.3).
- PCID: process-context identifiers.
  If CPUID.01H:ECX.PCID [bit 17] = 1, CR4.PCIDE may be set to 1, enabling process-context identifiers (see Section 4.10.1).
- SMEP: supervisor-mode execution prevention.
  If CPUID.(EAX=07H,ECX=0H):EBX.SMEP [bit 7] = 1, CR4.SMEP may be set to 1, enabling supervisor-mode execution prevention (see Section 4.6).
- SMAP: supervisor-mode access prevention.
  If CPUID.(EAX=07H,ECX=0H):EBX.SMAP [bit 20] = 1, CR4.SMAP may be set to 1, enabling supervisor-mode access prevention (see Section 4.6).
- PK: protection keys.
  If CPUID.(EAX=07H,ECX=0H):ECX.PK [bit 3] = 1, CR4.PKE may be set to 1, enabling protection keys (see Section 4.6).
- NX: execute disable.
  If CPUID.80000001H:EDX.NX [bit 20] = 1, IA32_EFER.NXE may be set to 1, allowing PAE paging and IA-32e paging to disable execute access to selected pages (see Section 4.6). (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.NXE to be set to 1.)
- Page1GB: 1-GByte pages.
  If CPUID.80000001H:EDX.Page1GB [bit 26] = 1, 1-GByte pages are supported with IA-32e paging (see Section 4.5).
- LM: IA-32e mode support.
  If CPUID.80000001H:EDX.LM [bit 29] = 1, IA32_EFER.LME may be set to 1, enabling IA-32e paging. (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.LME to be set to 1.)
- CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. (For processors that do not support CPUID function 80000008H, the width is generally 36 if CPUID.01H:EDX.PAE [bit 6] = 1 and 32 otherwise.) This width is referred to as MAXPHYADDR. MAXPHYADDR is at most 52.
- CPUID.80000008H:EAX[15:8] reports the linear-address width supported by the processor. Generally, this value is 48 if CPUID.80000001H:EDX.LM [bit 29] = 1 and 32 otherwise. (Processors that do not support CPUID function 80000008H, support a linear-address width of 32.)

...

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.

- 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.

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.

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.

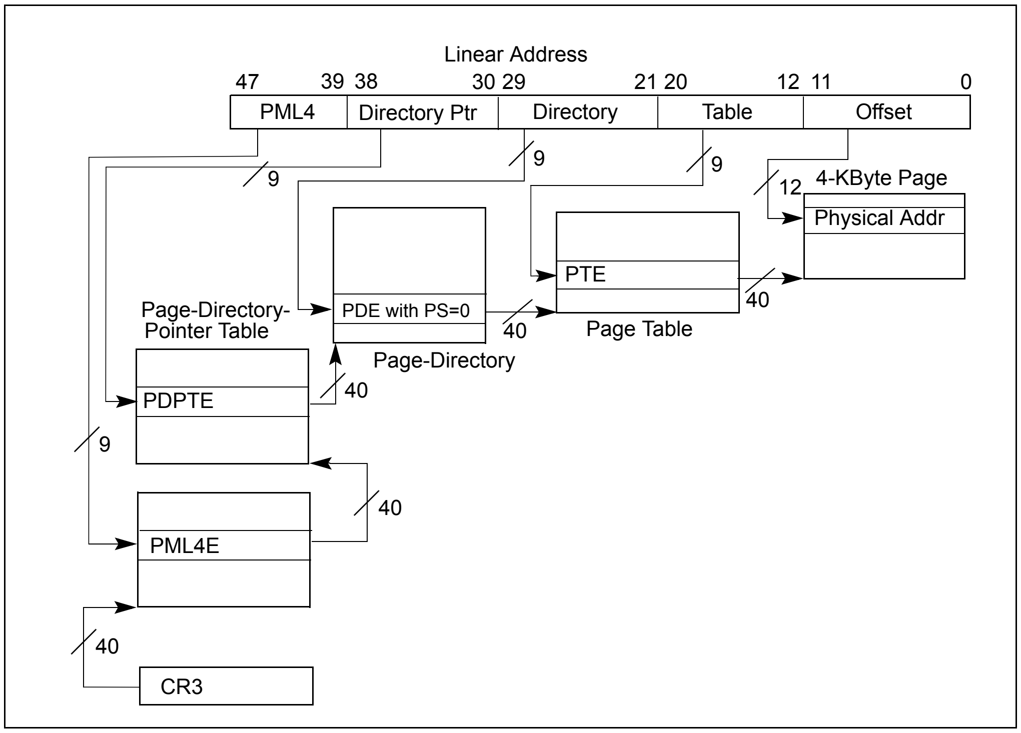


Figure 4-8 Linear-Address Translation to a 4-KByte Page using IA-32e Paging

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

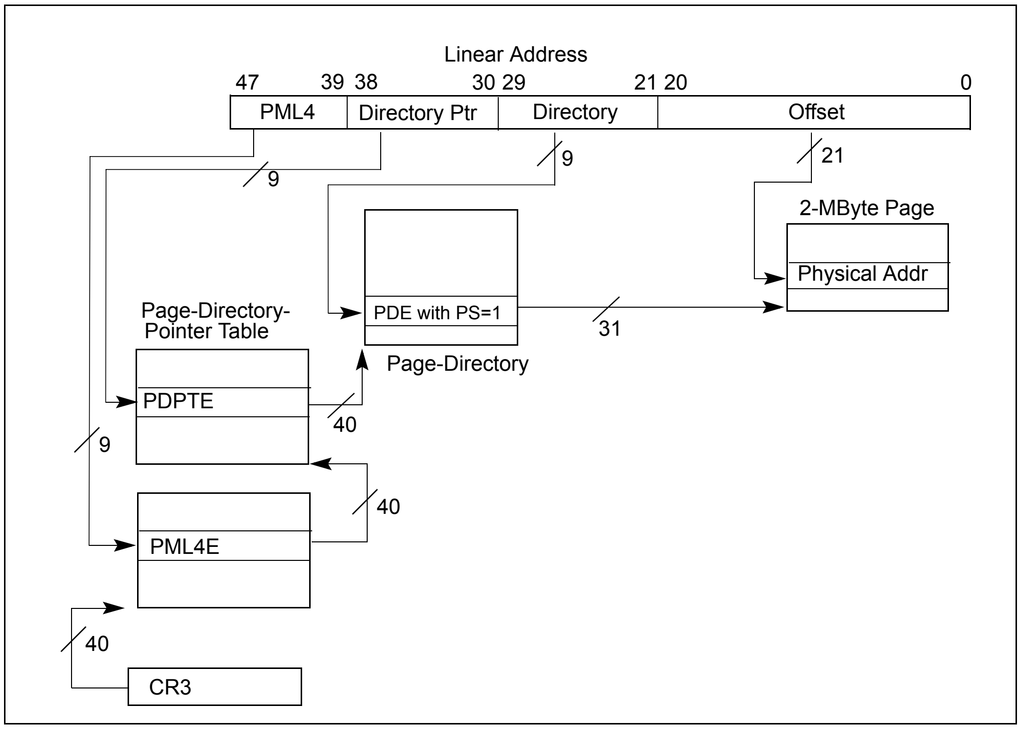


Figure 4-9 Linear-Address Translation to a 2-MByte Page using IA-32e Paging

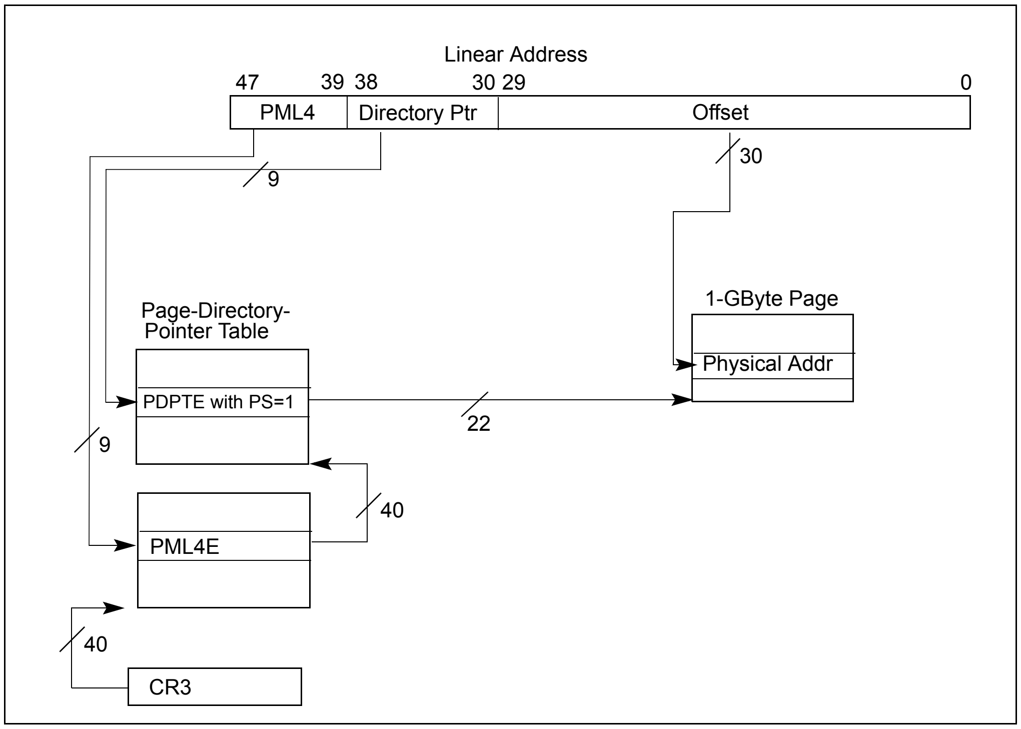


Figure 4-10 Linear-Address Translation to a 1-GByte Page using IA-32e Paging

If CR4.PKE = 1, IA-32e associates with each linear address a **protection key**. Section 4.6 explains how the processor uses the protection key in its determination of the access rights of each linear address.

The following items describe the IA-32e paging process in more detail as well has how the page size and protection key are 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 CR4.PKE = 1, the linear address's protection key is the value of bits 62:59 of the PDPTE.
- 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 CR4.PKE = 1, the linear address's protection key is the value of bits 62:59 of the PDE.
- 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:

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.

    — Bits 51:12 are from the PDE.
    — Bits 11:3 are bits 20:12 of the linear address.
    — Bits 2:0 are all 0.

- Because a PTE is identified using bits 47:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-19). 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 CR4.PKE = 1, the linear address's protection key is the value of bits 62:59 of the PTE.

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

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
7 (PS)	Reserved (must be 0)
11:8	Ignored
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

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

Bit Position(s)	Contents
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
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)
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)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
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)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
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)

.

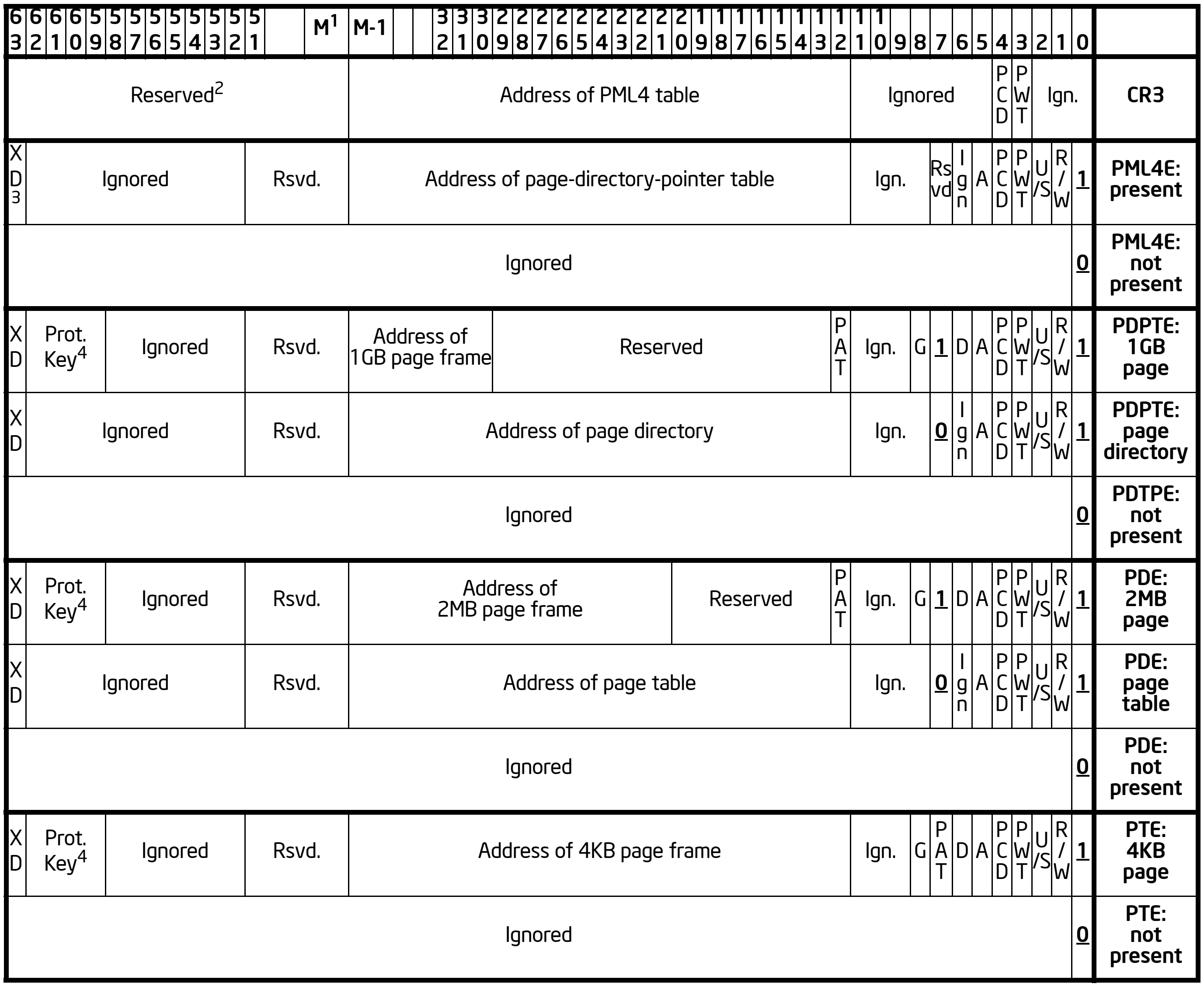

6 6 6 6 5 5 5 5 5 5 5 5 5
3 2 1 0 9 8 7 6 5 4 3 2 1
M1
M-1
3 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
Reserved2 | Address of PML4 table | Ignored | PCD | PWT | Ign. | CR3
XD3 | Ignored | Rsvd. | Address of page-directory-pointer table | Ign. | Rsvd | Ign | A | PCD | PWT | U/S | R/W | 1 | PML4E: present
Ignored | 0 | PML4E: not present
XD | Prot. Key4 | Ignored | Rsvd. | Address of 1GB page frame | Reserved | PAT | Ign. | G | 1 | D | A | PCD | PWT | U/S | R/W | 1 | PDPTE: 1GB page
XD | Ignored | Rsvd. | Address of page directory | Ign. | 0 | Ign | A | PCD | PWT | U/S | R/W | 1 | PDPTE: page directory
Ignored | 0 | PDTPE: not present
XD | Prot. Key4 | Ignored | Rsvd. | Address of 2MB page frame | Reserved | PAT | Ign. | G | 1 | D | A | PCD | PWT | U/S | R/W | 1 | PDE: 2MB page
XD | Ignored | Rsvd. | Address of page table | Ign. | 0 | Ign | A | PCD | PWT | U/S | R/W | 1 | PDE: page table
Ignored | 0 | PDE: not present
XD | Prot. Key4 | Ignored | Rsvd. | Address of 4KB page frame | Ign. | G | PAT | D | A | PCD | PWT | U/S | R/W | 1 | PTE: 4KB page
Ignored | 0 | PTE: not present

Figure 4-11 Formats of CR3 and Paging-Structure Entries with IA-32e Paging

NOTES:

1. M is an abbreviation for MAXPHYADDR.
2. Reserved fields must be 0.
3. If IA32_EFER.NXE = 0 and the P flag of a paging-structure entry is 1, the XD flag (bit 63) is reserved.
4. If CR4.PKE = 0, the protection key is ignored.

4.6 ACCESS RIGHTS

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

Section 4.6.1 describes how the processor determines the access rights for each linear address. Section 4.6.2 provides additional information about how protection keys contribute to access-rights determination. (They do so only with IA-32e paging and only if CR4.PKE = 1.)

4.6.1 Determination of Access Rights

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

Some operations implicitly access system data structures with linear addresses; the resulting accesses to those data structures are supervisor-mode accesses regardless of CPL. Examples of such accesses include the following: accesses to the global descriptor table (GDT) or local descriptor table (LDT) to load a segment descriptor; accesses to the interrupt descriptor table (IDT) when delivering an interrupt or exception; and accesses to the task-state segment (TSS) as part of a task switch or change of CPL. All these accesses are called **implicit supervisor-mode accesses** regardless of CPL. Other accesses made while CPL < 3 are called **explicit supervisor-mode accesses**.

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

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 with a protection key for which read access is permitted.
    - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
      - — If EFLAGS.AC = 1 and the access is explicit, data may be read from any user-mode address with a protection key for which read access is permitted.
      - — If EFLAGS.AC = 0 or the access is implicit, data may not be read from any user-mode address.

    Section 4.6.2 explains how protection keys are associated with user-mode addresses and the accesses that are permitted for each protection key.
  - — 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:

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

        - If CR0.WP = 0, access rights depend on the value of CR4.SMAP:
            - If CR4.SMAP = 0, data may be written to any user-mode address with a protection key for which write access is permitted.
            - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
                - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a protection key for which write access is permitted.
                - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
        - If CR0.WP = 1, access rights depend on the value of CR4.SMAP:
            - If CR4.SMAP = 0, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
            - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
                - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
                - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.

        Section 4.6.2 explains how protection keys are associated with user-mode addresses and the accesses that are permitted for each protection key.
    - 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 with a protection key for which read access is permitted. Section 4.6.2 explains how protection keys are associated with user-mode addresses and the accesses that are permitted for each protection key.
        - Data may not be read from any supervisor-mode address.
    - Data writes.
      Access rights depend on the mode of the linear address:
        - Data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted. Section 4.6.2 explains how protection keys are associated with user-mode addresses and the accesses that are permitted for each protection key.
        - 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.

4.6.2 Protection Keys

The protection-key feature provides an additional mechanism by which IA-32e paging controls access to user-mode addresses. When CR4.PKE = 1, every linear address is associated with the 4-bit **protection key** located in bits 62:59 of the paging-structure entry that mapped the page containing the linear address (see Section 4.5). The PKRU register determines, for each protection key, whether user-mode addresses with that protection key may be read or written.

If CR4.PKE = 0, or if IA-32e paging is not active, the processor does not associate linear addresses with protection keys and does not use the access-control mechanism described in this section. In either of these cases, a reference in Section 4.6.1 to a user-mode address with a protection key should be considered a reference to any user-mode address.

The PKRU register (protection key rights for user pages) is a 32-bit register with the following format: for each *i* ($0 \leq i \leq 15$), PKRU[2*i*] is the **access-disable bit** for protection key *i* (AD*i*); PKRU[2*i*+1] is the **write-disable bit** for protection key *i* (WD*i*).

Software can use the RDPKRU and WRPKRU instructions with ECX = 0 to read and write PKRU. In addition, the PKRU register is XSAVE-managed state and can thus be read and written by instructions in the XSAVE feature set. See Chapter 13, "Managing State Using the XSAVE Feature Set," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for more information about the XSAVE feature set.

How a linear address's protection key controls access to the address depends on the mode of a linear address:

- A linear address's protection controls only data accesses to the address. It does not in any way affect instructions fetches from the address.

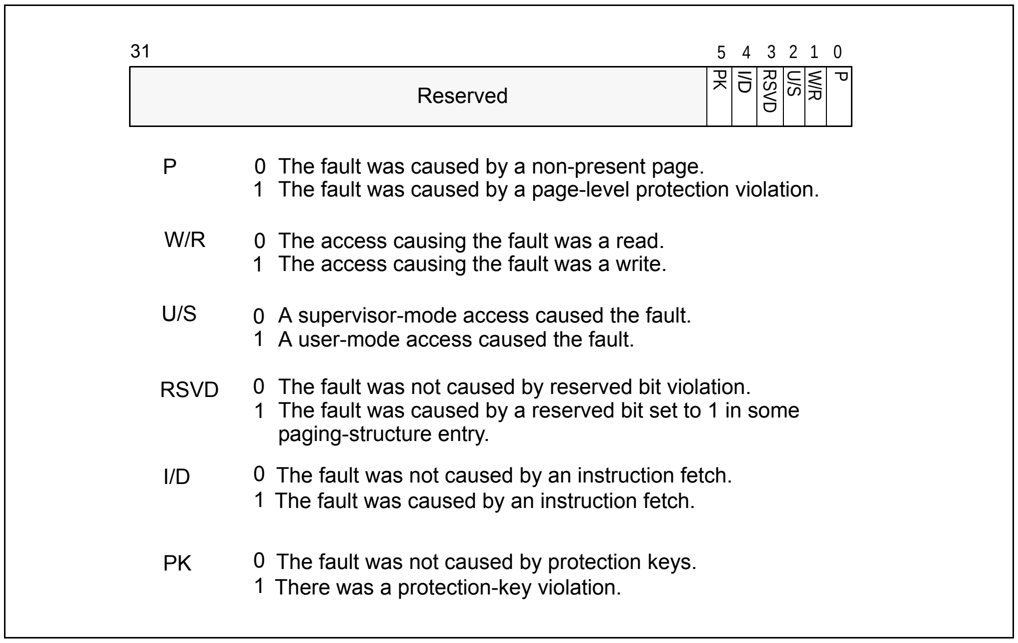
- The protection key of a supervisor-mode address is ignored and does not control data accesses to the address. Because of this, Section 4.6.1 does not refer to protection keys when specifying the access rights for supervisor-mode addresses.
- Use of the protection key *i* of a user-mode address depends on the value of the PKRU register:
  — If AD*i* = 1, no data accesses are permitted.
  — If WD*i* = 1, permission may be denied to certain data write accesses:
    - User-mode write accesses are not permitted.
    - Supervisor-mode write accesses are not permitted if CR0.WP = 1. (If CR0.WP = 0, WD*i* does not affect supervisor-mode write accesses to user-mode addresses with protection key *i*.)

4.7 PAGE-FAULT EXCEPTIONS

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

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

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



P	0	The fault was caused by a non-present page.
	1	The fault was caused by a page-level protection violation.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
U/S	0	A supervisor-mode access caused the fault.
	1	A user-mode access caused the fault.
RSVD	0	The fault was not caused by reserved bit violation.
	1	The fault was caused by a reserved bit set to 1 in some paging-structure entry.
I/D	0	The fault was not caused by an instruction fetch.
	1	The fault was caused by an instruction fetch.
PK	0	The fault was not caused by protection keys.
	1	There was a protection-key violation.

Figure 4-12 Page-Fault Error Code

- P flag (bit 0).
  This flag is 0 if there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.

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

  Bits reserved in the paging-structure entries are reserved for future functionality. Software developers should be aware that such bits may be used in the future and that a paging-structure entry that causes a page-fault exception on one processor might not do so in the future.
- I/D flag (bit 4).
  This flag is 1 if (1) the access causing the page-fault exception was an instruction fetch; and (2) either (a) CR4.SMEP = 1; or (b) both (i) CR4.PAE = 1 (either PAE paging or IA-32e paging is in use); and (ii) IA32_EFER.NXE = 1. Otherwise, the flag is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- PK flag (bit 5).
  This flag is 1 if (1) IA32_EFER.LMA = CR4.PKE = 1; (2) the access causing the page-fault exception was a data access; (3) the linear address was a user-mode address with protection key *i*; and (5) the PKRU register (see Section 4.6.2) is such that either (a) AD*i* = 1; or (b) the following all hold: (i) WD*i* = 1; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode access.

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

...

4.10.2.2 Caching Translations in TLBs

The processor may accelerate the paging process by caching individual translations in **translation lookaside buffers** (**TLBs**). Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number (see Section 4.6):
  - The logical-AND of the R/W flags.
  - The logical-AND of the U/S flags.
  - The logical-OR of the XD flags (necessary only if IA32_EFER.NXE = 1).
  - The protection key (necessary only with IA-32e paging and CR4.PKE = 1).
- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry in which the PS flag is 1):

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

— The dirty flag (see Section 4.8).

— The memory type (see Section 4.9).

(TLB entries may contain other information as well. A processor may implement multiple TLBs, and some of these may be for special purposes, e.g., only for instruction fetches. Such special-purpose TLBs may not contain some of this information if it is not necessary. For example, a TLB used only for instruction fetches need not contain information about the R/W and dirty flags.)

As noted in Section 4.10.1, any TLB entries created by a logical processor are associated with the current PCID.

Processors need not implement any TLBs. Processors that do implement TLBs may invalidate any TLB entry at any time. Software should not rely on the existence of TLBs or on the retention of TLB entries.

...

4.10.4.2 Recommended Invalidation

The following items provide some recommendations regarding when software should perform invalidations:

- If software modifies a paging-structure entry that maps a page (rather than referencing another paging structure), it should execute INVLPG for any linear address with a page number whose translation uses that paging-structure entry.[1]

  (If the paging-structure entry may be used in the translation of different page numbers — see Section 4.10.3.3 — software should execute INVLPG for linear addresses with each of those page numbers; alternatively, it could use MOV to CR3 or MOV to CR4.)
- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:
  - — Execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry. However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute INVLPG at least once.
  - — Execute MOV to CR3 if the modified entry controls no global pages.
  - — Execute MOV to CR4 to modify CR4.PGE.
- If CR4.PCIDE = 1 and software modifies a paging-structure entry that does not map a page or in which the G flag (bit 8) is 0, additional steps are required if the entry may be used for PCIDs other than the current one. Any one of the following suffices:
  - — Execute MOV to CR4 to modify CR4.PGE, either immediately or before again using any of the affected PCIDs. For example, software could use different (previously unused) PCIDs for the processes that used the affected PCIDs.
  - — For each affected PCID, execute MOV to CR3 to make that PCID current (and to load the address of the appropriate PML4 table). If the modified entry controls no global pages and bit 63 of the source operand to MOV to CR3 was 0, no further steps are required. Otherwise, execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry; if no page numbers that would use the entry have translations, execute INVLPG at least once.
- If software using PAE paging modifies a PDPTE, it should reload CR3 with the register's current value to ensure that the modified PDPTE is loaded into the corresponding PDPTE register (see Section 4.4.1).
- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 4.10.3.3), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute INVLPG with two (or more) linear

1. One execution of INVLPG is sufficient even for a page with size greater than 4 KBytes.

addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use MOV to CR3 or MOV to CR4.)

- As noted in Section 4.10.2, the TLBs may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes. A reference to a linear address in the address range may use any of these translations.

  Software wishing to prevent this uncertainty should not write to a paging-structure entry in a way that would change, for any linear address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g., PDE); then invalidate any translations for the affected linear addresses (see above); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.
- Software should clear bit 63 of the source operand to a MOV to CR3 instruction that establishes a PCID that had been used earlier for a different linear-address space (e.g., with a different value in bits 51:12 of CR3). This ensures invalidation of any information that may have been cached for the previous linear-address space.

  This assumes that both linear-address spaces use the same global pages and that it is thus not necessary to invalidate any global TLB entries. If that is not the case, software should invalidate those entries by executing MOV to CR4 to modify CR4.PGE.

...

4.10.4.4 Delayed Invalidation

Required invalidations may be delayed under some circumstances. Software developers should understand that, between the modification of a paging-structure entry and execution of the invalidation instruction recommended in Section 4.10.4.2, the processor may use translations based on either the old value or the new value of the paging-structure entry. The following items describe some of the potential consequences of delayed invalidation:

- If a paging-structure entry is modified to change the P flag from 1 to 0, an access to a linear address whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, write accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the U/S flag from 0 to 1, user-mode accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the XD flag from 1 to 0, instruction fetches from linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.

As noted in Section 8.1.1, an x87 instruction or an SSE instruction that accesses data larger than a quadword may be implemented using multiple memory accesses. If such an instruction stores to memory and invalidation has been delayed, some of the accesses may complete (writing to memory) while another causes a page-fault exception.[1] In this case, the effects of the completed accesses may be visible to software even though the overall instruction caused a fault.

In some cases, the consequences of delayed invalidation may not affect software adversely. For example, when freeing a portion of the linear-address space (by marking paging-structure entries "not present"), invalidation using INVLPG may be delayed if software does not re-allocate that portion of the linear-address space or the memory that had been associated with it. However, because of speculative execution (or errant software), there may be accesses to the freed portion of the linear-address space before the invalidations occur. In this case, the following can happen:

- Reads can occur to the freed portion of the linear-address space. Therefore, invalidation should not be delayed for an address range that has read side effects.

1. If the accesses are to different pages, this may occur even if invalidation has not been delayed.

- The processor may retain entries in the TLBs and paging-structure caches for an extended period of time. Software should not assume that the processor will not use entries associated with a linear address simply because time has passed.
- As noted in Section 4.10.3.1, the processor may create an entry in a paging-structure cache even if there are no translations for any linear address that might use that entry. Thus, if software has marked "not present" all entries in page table, the processor may subsequently create a PDE-cache entry for the PDE that references that page table (assuming that the PDE itself is marked "present").
- If software attempts to write to the freed portion of the linear-address space, the processor might not generate a page fault. (Such an attempt would likely be the result of a software error.) For that reason, the page frames previously associated with the freed portion of the linear-address space should not be reallocated for another purpose until the appropriate invalidations have been performed.

...

16. Updates to Chapter 6, Volume 3A

Change bars show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

Interrupt 14—Page-Fault Exception (#PF)

Exception Class Fault.

Description

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

- The P (present) flag in a page-directory or page-table entry needed for the address translation is clear, indicating that a page table or the page containing the operand is not present in physical memory.
- The procedure does not have sufficient privilege to access the indicated page (that is, a procedure running in user mode attempts to access a supervisor-mode page). If the SMAP flag is set in CR4, a page fault may also be triggered by code running in supervisor mode that tries to access data at a user-mode address. If the PKE flag is set in CR4, the PKRU register may cause page faults on data accesses to user-mode addresses with certain protection keys.
- Code running in user mode attempts to write to a read-only page. If the WP flag is set in CR0, the page fault will also be triggered by code running in supervisor mode that tries to write to a read-only page.
- An instruction fetch to a linear address that translates to a physical address in a memory page with the execute-disable bit set (for information about the execute-disable bit, see Chapter 4, "Paging"). If the SMEP flag is set in CR4, a page fault will also be triggered by code running in supervisor mode that tries to fetch an instruction from a user-mode address.
- One or more reserved bits in 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.
  - PK flag (bit 5).
    This flag is 1 if the access causing the page-fault exception was a data access to a user-mode address with protection key disallowed by the value of the PKRU register.

  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.

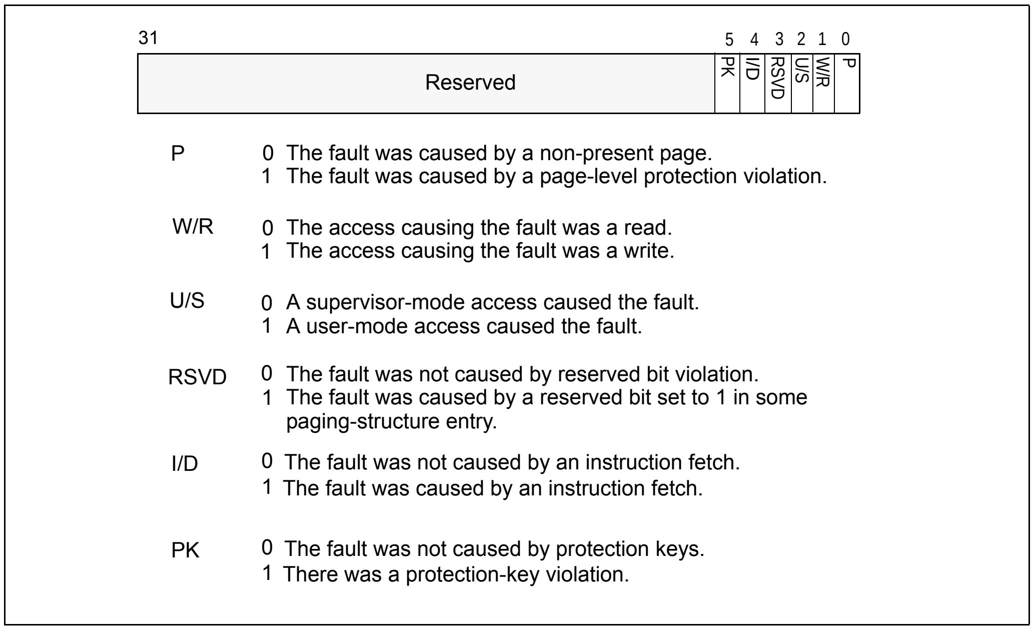

31
5 4 3 2 1 0
Reserved
PK
I/D
RSVD
U/S
W/R
P
P
0 The fault was caused by a non-present page.
1 The fault was caused by a page-level protection violation.
W/R
0 The access causing the fault was a read.
1 The access causing the fault was a write.
U/S
0 A supervisor-mode access caused the fault.
1 A user-mode access caused the fault.
RSVD
0 The fault was not caused by reserved bit violation.
1 The fault was caused by a reserved bit set to 1 in some paging-structure entry.
I/D
0 The fault was not caused by an instruction fetch.
1 The fault was caused by an instruction fetch.
PK
0 The fault was not caused by protection keys.
1 There was a protection-key violation.

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.
- 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.

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.

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.

...

17. Updates to Chapter 10, Volume 3A

Change bars show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

10.5.4.1 TSC-Deadline Mode

The mode of operation of the local-APIC timer is determined by the LVT Timer Register. Specifically, if CPUID.01H:ECX.TSC_Deadline[bit 24] = 0, the mode is determined by bit 17 of the register; if CPUID.01H:ECX.TSC_Deadline[bit 24] = 1, the mode is determined by bits 18:17. See Figure 10-8. (If CPUID.01H:ECX.TSC_Deadline[bit 24] = 0, bit 18 of the register is reserved.) A write to the LVT Timer Register that changes the timer mode disarms the local APIC timer. The supported timer modes are given in Table 10-2. The three modes of the local APIC timer are mutually exclusive.

Table 10-2 Local APIC Timer Modes

LVT Bits [18:17]	Timer Mode
00b	One-shot mode, program count-down value in an initial-count register. See Section 10.5.4
01b	Periodic mode, program interval value in an initial-count register. See Section 10.5.4
10b	TSC-Deadline mode, program target value in IA32_TSC_DEADLINE MSR.
11b	Reserved

TSC-deadline mode allows software to use the local APIC timer to signal an interrupt at an absolute time. In TSC-deadline mode, writes to the initial-count register are ignored; and current-count register always reads 0. Instead, timer behavior is controlled using the IA32_TSC_DEADLINE MSR.

The IA32_TSC_DEADLINE MSR (MSR address 6E0H) is a per-logical processor MSR that specifies the time at which a timer interrupt should occur. Writing a non-zero 64-bit value into IA32_TSC_DEADLINE arms the timer. An interrupt is generated when the logical processor's time-stamp counter equals or exceeds the target value in the IA32_TSC_DEADLINE MSR.[1] When the timer generates an interrupt, it disarms itself and clears the IA32_TSC_DEADLINE MSR. Thus, each write to the IA32_TSC_DEADLINE MSR generates at most one timer interrupt.

1. If the logical processor is in VMX non-root operation, a read of the time-stamp counter (using either RDMSR, RDTSC, or RDTSCP) may not return the actual value of the time-stamp counter; see Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. It is the responsibility of software operating in VMX root operation to coordinate the virtualization of the time-stamp counter and the IA32_TSC_DEADLINE MSR.

In TSC-deadline mode, writing 0 to the IA32_TSC_DEADLINE MSR disarms the local-APIC timer. Transitioning between TSC-deadline mode and other timer modes also disarms the timer.

The hardware reset value of the IA32_TSC_DEADLINE MSR is 0. In other timer modes (LVT bit 18 = 0), the IA32_TSC_DEADLINE MSR reads zero and writes are ignored.

Software can configure the TSC-deadline timer to deliver a single interrupt using the following algorithm:

1. Detect support for TSC-deadline mode by verifying CPUID.1:ECX.24 = 1.
2. Select the TSC-deadline mode by programming bits 18:17 of the LVT Timer register with 10b.
3. Program the IA32_TSC_DEADLINE MSR with the target TSC value at which the timer interrupt is desired. This causes the processor to arm the timer.
4. The processor generates a timer interrupt when the value of time-stamp counter is greater than or equal to that of IA32_TSC_DEADLINE. It then disarms the timer and clear the IA32_TSC_DEADLINE MSR. (Both the time-stamp counter and the IA32_TSC_DEADLINE MSR are 64-bit unsigned integers.)
5. Software can re-arm the timer by repeating step 3.

The following are usage guidelines for TSC-deadline mode:

- Writes to the IA32_TSC_DEADLINE MSR are not serialized. Therefore, system software should not use WRMSR to the IA32_TSC_DEADLINE MSR as a serializing instruction. Read and write accesses to the IA32_TSC_DEADLINE and other MSR registers will occur in program order.
- Software can disarm the timer at any time by writing 0 to the IA32_TSC_DEADLINE MSR.
- If timer is armed, software can change the deadline (forward or backward) by writing a new value to the IA32_TSC_DEADLINE MSR.
- If software disarms the timer or postpones the deadline, race conditions may result in the delivery of a spurious timer interrupt. Software is expected to detect such spurious interrupts by checking the current value of the time-stamp counter to confirm that the interrupt was desired.[1]
- In xAPIC mode (in which the local-APIC registers are memory-mapped), software must serialize between the memory-mapped write to the LVT entry and the WRMSR to IA32_TSC_DEADLINE. In x2APIC mode, no serialization is required between the two writes (by WRMSR) to the LVT and IA32_TSC_DEADLINE MSRs.

The following is a sample algorithm for serializing writes in xAPIC mode:

1. Memory-mapped write to LVT Timer Register, setting bits 18:17 to 10b.
2. WRMSR to the IA32_TSC_DEADLINE MSR a value much larger than current time-stamp counter.
3. If RDMSR of the IA32_TSC_DEADLINE MSR returns zero, go to step 2.
4. WRMSR to the IA32_TSC_DEADLINE MSR the desired deadline.

...

18. Updates to Chapter 13, Volume 3A

Change bars show changes to Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

1. If the logical processor is in VMX non-root operation, a read of the time-stamp counter (using either RDMSR, RDTSC, or RDTSCP) may not return the actual value of the time-stamp counter; see Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. It is the responsibility of software operating in VMX root operation to coordinate the virtualization of the time-stamp counter and the IA32_TSC_DEADLINE MSR.

13.1 PROVIDING OPERATING SYSTEM SUPPORT FOR SSE EXTENSIONS

To use SSE extensions, the operating system or executive must provide support for initializing the processor to use these extensions, for handling SIMD floating-point exceptions, and for using FXSAVE and FXRSTOR (Section 10.5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*) to manage context. XSAVE feature set can also be used to manage SSE state along with other processor extended states as described in 13.5. This section primarily focuses on using FXSAVE/FXRSTOR to manage SSE state. Because SSE extensions share the same state, experience the same sets of non-numerical and numerical exception behavior, these guidelines that apply to SSE also apply to other sets of SIMD extensions that operate on the same processor state and subject to the same sets of non-numerical and numerical exception behavior.

Chapter 11, "Programming with Streaming SIMD Extensions 2 (SSE2)" and Chapter 12, "Programming with SSE3, SSSE3 and SSE4," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, provide details on SSE instruction set.

...

13.1.3 Initialization of the SSE Extensions

The operating system or executive should carry out the following steps to set up SSE extensions for use by application programs:

1. Set CR4.OSFXSR[bit 9] = 1. Setting this flag implies that the operating system provides facilities for saving and restoring SSE state using FXSAVE and FXRSTOR instructions. These instructions may be used to save the SSE state during task switches and when invoking the SIMD floating-point exception (#XM) handler (see Section 13.1.5, "Providing a Handler for the SIMD Floating-Point Exception (#XM)").

   If the processor does not support the FXSAVE and FXRSTOR instructions, attempting to set the OSFXSR flag causes a general-protection exception (#GP) to be generated.
2. Set CR4.OSXMMEXCPT[bit 10] = 1. Setting this flag implies that the operating system provides a SIMD floating-point exception (#XM) handler (see Section 13.1.5, "Providing a Handler for the SIMD Floating-Point Exception (#XM)").

NOTE

The OSFXSR and OSXMMEXCPT bits in control register CR4 must be set by the operating system. The processor has no other way of detecting operating-system support for the FXSAVE and FXRSTOR instructions or for handling SIMD floating-point exceptions.

3. Clear CR0.EM[bit 2] = 0. This action disables emulation of the x87 FPU, which is required when executing SSE instructions (see Section 2.5, "Control Registers").
4. Set CR0.MP[bit 1] = 1. This setting is required for Intel 64 and IA-32 processors that support the SSE extensions (see Section 9.2.1, "Configuring the x87 FPU Environment").

Table 13-1 and Table 13-2 show the actions of the processor when an SSE instruction is executed, depending on the following:

- OSFXSR and OSXMMEXCPT flags in control register CR4
- SSE/SSE2/SSE3/SSSE3/SSE4 feature flags returned by CPUID
- EM, MP, and TS flags in control register CR0

Table 13-1 Action Taken for Combinations of OSFXSR, OSXMMEXCPT, SSE, SSE2, SSE3, EM, MP, and TS[1]

CR4		CPUID	CR0 Flags			Action
OSFXSR	OSXMMEXCPT	SSE, SSE2, SSE3[2], SSE4_1[3]	EM	MP[4]	TS	
0	X[5]	X	X	1	X	#UD exception.
1	X	0	X	1	X	#UD exception.
1	X	1	1	1	X	#UD exception.
1	0	1	0	1	0	Execute instruction; #UD exception if unmasked SIMD floating-point exception is detected.
1	1	1	0	1	0	Execute instruction; #XM exception if unmasked SIMD floating-point exception is detected.
1	X	1	0	1	1	#NM exception.

NOTES:
1. For execution of any SSE instruction except the PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, and CLFLUSH instructions.
2. Exception conditions due to CR4.OSFXSR or CR4.OSXMMEXCPT do not apply to FISTTP.
3. Only applies to DPPS, DPPD, ROUNDPS, ROUNDPD, ROUNDSS, ROUNDSD.
4. For processors that support the MMX instructions, the MP flag should be set.
5. X = Don’t care.

Table 13-2 Action Taken for Combinations of OSFXSR, SSSE3, SSE4, EM, and TS

CR4	CPUID	CR0 Flags		Action
OSFXSR	SSSE3 SSE4_1[1] SSE4_2[2]	EM	TS	
0	X[3]	X	X	#UD exception.
1	0	X	X	#UD exception.
1	1	1	X	#UD exception.
1	1	0	1	#NM exception.

NOTES:
1. Applies to SSE4_1 instructions except DPPS, DPPD, ROUNDPS, ROUNDPD, ROUNDSS, ROUNDSD.
2. Applies to SSE4_2 instructions except CRC32 and POPCNT.
3. X = Don’t care.

The SIMD floating-point exception mask bits (bits 7 through 12), the flush-to-zero flag (bit 15), the denormals-are-zero flag (bit 6), and the rounding control field (bits 13 and 14) in the MXCSR register should be left in their default values of 0. This permits the application to determine how these features are to be used.

13.1.4 Providing Non-Numeric Exception Handlers for Exceptions Generated by the SSE Instructions

SSE instructions can generate the same type of memory-access exceptions (such as page faults and limit violations) and other non-numeric exceptions as other Intel 64 and IA-32 architecture instructions generate.

Ordinarily, existing exception handlers can handle these and other non-numeric exceptions without code modification. However, depending on the mechanisms used in existing exception handlers, some modifications might need to be made.

The SSE extensions can generate the non-numeric exceptions listed below:

- Memory Access Exceptions:
  - — Stack-segment fault (#SS).
  - — General protection exception (#GP). Executing most SSE instructions with an unaligned 128-bit memory reference generates a general-protection exception. (The MOVUPS and MOVUPD instructions allow unaligned a loads or stores of 128-bit memory locations, without generating a general-protection exception.) A 128-bit reference within the stack segment that is not aligned to a 16-byte boundary will also generate a general-protection exception, instead a stack-segment fault exception (#SS).
  - — Page fault (#PF).
  - — Alignment check (#AC). When enabled, this type of alignment check operates on operands that are less than 128-bits in size: 16-bit, 32-bit, and 64-bit. To enable the generation of alignment check exceptions, do the following:
    - Set the AM flag (bit 18 of control register CR0)
    - Set the AC flag (bit 18 of the EFLAGS register)
    - CPL must be 3

    If alignment check exceptions are enabled, 16-bit, 32-bit, and 64-bit misalignment will be detected for the MOVUPD and MOVUPS instructions; detection of 128-bit misalignment is not guaranteed and may vary with implementation.
- System Exceptions:
  - — Invalid-opcode exception (#UD). This exception is generated when executing SSE instructions under the following conditions:
    - SSE/SSE2/SSE3/SSSE3/SSE4_1/SSE4_2 feature flags returned by CPUID are set to 0. This condition does not affect the CLFLUSH instruction, nor POPCNT.
    - The CLFSH feature flag returned by the CPUID instruction is set to 0. This exception condition only pertains to the execution of the CLFLUSH instruction.
    - The POPCNT feature flag returned by the CPUID instruction is set to 0. This exception condition only pertains to the execution of the POPCNT instruction.
    - The EM flag (bit 2) in control register CR0 is set to 1, regardless of the value of TS flag (bit 3) of CR0. This condition does not affect the PAUSE, PREFETCH*h*, MOVNTI, SFENCE, LFENCE, MFENCE, CLFLUSH, CRC32 and POPCNT instructions.
    - The OSFXSR flag (bit 9) in control register CR4 is set to 0. This condition does not affect the PSHUFW, MOVNTQ, MOVNTI, PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, CLFLUSH, CRC32 and POPCNT instructions.
    - Executing an instruction that causes a SIMD floating-point exception when the OSXMMEXCPT flag (bit 10) in control register CR4 is set to 0. See Section 13.4.1, "Using the TS Flag to Control the Saving of the x87 FPU and SSE State."
  - — Device not available (#NM). This exception is generated by executing a SSE instruction when the TS flag (bit 3) of CR0 is set to 1.

Other exceptions can occur during delivery of the above exceptions.

13.1.5 Providing a Handler for the SIMD Floating-Point Exception (#XM)

SSE instructions do not generate numeric exceptions on packed integer operations. They can generate the following numeric (SIMD floating-point) exceptions on packed and scalar single-precision and double-precision floating-point operations.

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

These SIMD floating-point exceptions (with the exception of the denormal operand exception) are defined in the IEEE Standard 754 for Binary Floating-Point Arithmetic and represent the same conditions that cause x87 FPU floating-point error exceptions (#MF) to be generated for x87 FPU instructions.

Each of these exceptions can be masked, in which case the processor returns a reasonable result to the destination operand without invoking an exception handler. However, if any of these exceptions are left unmasked, detection of the exception condition results in a SIMD floating-point exception (#XM) being generated. See Chapter 6, "Interrupt 19—SIMD Floating-Point Exception (#XM)."

To handle unmasked SIMD floating-point exceptions, the operating system or executive must provide an exception handler. The section titled "SSE and SSE2 SIMD Floating-Point Exceptions" in Chapter 11, "Programming with Streaming SIMD Extensions 2 (SSE2)," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, describes the SIMD floating-point exception classes and gives suggestions for writing an exception handler to handle them.

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

...

13.3 SAVING AND RESTORING SSE STATE

The SSE state consists of the state of the XMM and MXCSR registers. Intel recommends the following method for saving and restoring this state:

- Execute the FXSAVE instruction to save the state of the XMM and MXCSR registers to memory.
- Execute the FXRSTOR instruction to restore the state of the XMM and MXCSR registers from the image saved in memory earlier.

This save and restore method is required for all operating systems. XSAVE feature set can also be used to save/restore SSE state. See Section 13.5, "The XSAVE Feature Set and Processor Extended State Management," for using the XSAVE feature set to save/restore SSE state.

In some cases, applications may choose to save only the XMM and MXCSR registers in the following manner:

- Execute MOVDQ instructions to save the contents of the XMM registers to memory.
- Execute a STMXCSR instruction to save the state of the MXCSR register to memory.

Such applications must restore the XMM and MXCSR registers as follows:

- Execute MOVDQ instructions to load the saved contents of the XMM registers from memory into the XMM registers.
- Execute a LDMXCSR instruction to restore the state of the MXCSR register from memory.

13.4 DESIGNING OS FACILITIES FOR SAVING X87 FPU, SSE AND EXTENDED STATES ON TASK OR CONTEXT SWITCHES

The x87 FPU and SSE state consist of the state of the x87 FPU, XMM, and MXCSR registers. The FXSAVE and FXRSTOR instructions provide a fast method for saving and restoring this state. The XSAVE feature set can also be used to save FP and SSE state along with other extended states (see Section 13.5).

Older operating systems may use FSAVE/FNSAVE and FRSTOR to save the x87 FPU state. These facilities can be extended to save and restore SSE state by substituting FXSAVE and FXRSTOR or the XSAVE feature set in place of FSAVE/FNSAVE and FRSTOR.

If task or context switching facilities are written from scratch, any of several approaches may be taken for using the FXSAVE and FXRSTOR instructions or the XSAVE feature set to save and restore x87 FPU and SSE state:

- The operating system can require applications that are intended to be run as tasks take responsibility for saving the states prior to a task suspension during a task switch and for restoring the states when the task is resumed. This approach is appropriate for cooperative multitasking operating systems, where the application has control over (or is able to determine) when a task switch is about to occur and can save state prior to the task switch.
- The operating system can take the responsibility for saving the states as part of the task switch process and restoring the state of the registers when a suspended task is resumed. This approach is appropriate for preemptive multitasking operating systems, where the application cannot know when it is going to be preempted and cannot prepare in advance for task switching.
- The operating system can take the responsibility for saving the states as part of the task switch process, but delay the restoring of the states until an instruction operating on the states is actually executed by the new task. See Section 13.4.1, “Using the TS Flag to Control the Saving of the x87 FPU and SSE State,” for more information. This approach is called lazy restore.

  The use of lazy restore mechanism in context switches is not recommended when XSAVE feature set is used to save/restore states for the following reasons.

  — With XSAVE feature set, Intel processors have optimizations in place to avoid saving the state components that are in their initial configurations or when they have not been modified since they were restored last. These optimizations eliminate the need for lazy restore. See section 13.5.4 in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

  — Intel processors have power optimizations when state components are in their initial configurations. Use of lazy restore retains the non-initial configuration of the last thread and is not power efficient.

  — Not all extended states support lazy restore mechanisms. As such, when one or more such states are enabled it becomes very inefficient to use lazy restore as it results in two separate state restore, one in context switch for the states that does not support lazy restore and one in the #NM handler for states that support lazy restore.

...

13.5 THE XSAVE FEATURE SET AND PROCESSOR EXTENDED STATE MANAGEMENT

The architecture of XSAVE feature set is described in CHAPTER 13 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*. The XSAVE feature set includes the following:

- An extensible data layout for existing and future processor state extensions. The layout of the XSAVE area extends from the 512-byte FXSAVE/FXRSTOR layout to provide compatibility and migration path from managing the legacy FXSAVE/FXRSTOR area. The XSAVE area is described in more detail in Section 13.4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.
- CPUID enhancements for feature enumeration. See Section 13.2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.
- Control register enhancement and dedicated register for enabling each processor extended state. See Section 13.3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.
- Instructions to save state to and restore state from the XSAVE area. See Section 13.7 through Section 13.9 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Operating systems can utilize XSAVE feature set to manage both FP/SSE state and processor extended states. CPUID leaf 0DH enumerates XSAVE feature set related information. The following guidelines provide the steps an operating system needs to take to support legacy FP/SSE states and processor extended states.

1. Check that the processor supports the XSAVE feature set
2. Determine the set of XSAVE managed features that the operating system intends to enable and calculate the size of the buffer needed to save/restore the states during context switch and other flows
3. Enable use of XSAVE feature set and XSAVE managed features
4. Provide an initialization for the XSAVE managed feature state components
5. Provide (if necessary) required exception handlers for exceptions generated each of the XSAVE managed features.

...

13.5.2 Determining the XSAVE Managed Feature States And The Required Buffer Size

Each XSAVE managed feature has one or more state components associated with it. An operating system policy needs to determine the XSAVE managed features to support and determine the corresponding state components to enable. When determining the XSAVE managed features to support, operating system needs to take into account the dependencies between them (e.g. AVX feature depends on SSE feature). Similarly, when a XSAVE managed feature has more than one state component, all of them need to be enabled. Each logical processor enumerates supported XSAVE state components in CPUID.(EAX=0DH, ECX=0).EDX:EAX. An operating system may enable all or a subset of the state components enumerated by the processor based on the OS policy.

The size of the memory buffer needed to save enabled XSAVE state components depends on whether the OS opts-in to use compacted format or not. Section 13.4.3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* describes the layout of the extended region of the XSAVE area.

13.5.3 Enable the Use Of XSAVE Feature Set And XSAVE State Components

Operating systems need to enable the use of XSAVE feature set by writing to CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCR0 and to support processor extended state management using XSAVE/XRSTOR. When XSAVE feature set is enabled, all enumerated XSAVE sub features such as optimized save, compaction and supervisor state support are also enabled. Operating systems also need to enable the XSAVE state components in XCR0 using XSETBV instruction.

XSAVE state components can subsequently be disabled in XCR0. However, disabling state components of AVX or AVX-512 that are not in initial configuration may incur power and performance penalty on SSE and AVX instructions respectively. If AVX state is disabled when it is not in its initial configuration, subsequent SSE instructions may incur a penalty. If AVX-512 state is disabled when it is not in its initial configuration, subsequent SSE and AVX instructions may incur a penalty. It is recommended that the operating systems and VMM set AVX or AVX-512 state components to their initial configuration before disabling them. This can be achieved by one of the two methods below.

- Using XRSTOR: Operating system or VMM can set the state of AVX or AVX-512 state components using XRSTOR instruction before disabling them in XCR0.
- Using VZEROUPPER: Operating system or VMM can set AVX and AVX-512 state components to their initial configuration using VZEROUPPER instruction before disabling them in XCR0. Note that this will set both AVX and AVX-512 state components to their initial configuration. If the intent is to only disable AVX-512 state, Operating system or VMM will need to save AVX state before executing VZEROUPPER and restore it afterwards.

13.5.4 Provide an Initialization for the XSAVE State Components

The XSAVE header of a newly allocated XSAVE area should be initialized to all zeroes before saving context. An operating system may choose to establish beginning state-component values for a task by executing XRSTOR from an XSAVE area that the OS has configured. If it is desired to begin state component i in its initial configuration, the OS should clear bit i in the XSTATE_BV field in the XSAVE header; otherwise, it should set that bit and place the desired beginning value in the appropriate location in the XSAVE area.

When a buffer is allocated for compacted size, software must ensure that the XCOMP_BV field is setup correctly before restoring from the buffer. Bit 63 of the XCOMP_BV field indicates that the save area is in the compacted format and the remaining bits indicate the states that have space allocated in the save area. If the buffer is first used to save the state in compacted format, then the save instructions will setup the XCOMP_BV field appropriately. If the buffer is first used to restore the state, then software must set up the XCOMP_BV field.

...

13.6 INTEROPERABILITY OF THE XSAVE FEATURE SET AND FXSAVE/FXRSTOR

The FXSAVE instruction writes x87 FPU and SSE state information to a 512-byte FXSAVE save area. FXRSTOR restores the processor’s x87 FPU and SSE states from an FXSAVE area. The XSAVE features set supports x87 FPU and SSE states using the same layout as the FXSAVE area to provide interoperability of FXSAVE versus XSAVE, and FXRSTOR versus XRSTOR. The XSAVE feature set allows system software to manage SSE state independent of x87 FPU states. Thus system software that had been using FXSAVE and FXRSTOR to manage x87 FPU and SSE states can transition to using the XSAVE feature set to manage x87 FPU, SSE and other processor extended states in a systematic and forward-looking manner. See Section 10.5 and Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1* for more details.

System software can implement forward-looking processor extended state management using the XSAVE feature set. In this case, system software must specify the bit vector mask in EDX:EAX appropriately when executing XSAVE/XRSTOR instructions.

For instance, the OS can supply instructions in the XSAVE feature set with a bit vector in EDX:EAX with the two least significant bits (corresponding to x87 FPU and SSE state) equal to 0. Then, the XSAVE instruction will not write the processor’s x87 FPU and SSE state into memory. Similarly, the XRSTOR instruction executed with a value in EDX:EAX with the least two significant bit equal to 0 will not restore nor initialize the processor’s x87 FPU and SSE state.

The processor’s action as a result of executing XRSTOR is given in Section 13.8 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*. The instruction may be used to initialize x87 FPU or XMM registers. When the MXCSR register is updated from memory, reserved bit checking is enforced. The saving/restoring of MXCSR is bound to the SSE state, independent of the x87 FPU state. The action of XSAVE is given in Section 13.7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

...

13.8.1 Intel® Advanced Vector Extensions (Intel® AVX)

Intel AVX instructions comprises of 256-bit and 128-bit instructions that operates on 256-bit YMM registers. The XSAVE feature set allows software to save and restore the state of these registers. See Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

For processors that support YMM states, the YMM state exists in all operating modes. However, the available instruction interfaces to access YMM states may vary in different modes.

Operating systems must use the XSAVE feature set for YMM state management. The XSAVE feature set also provides flexible and efficient interface to manage XMM/MXCSR states and x87 FPU states in conjunction with newer processor extended states like YMM states. Operating systems may need to be aware of the following when supporting AVX.

- Saving/Restoring AVX state in non-compacted format without SSE state will also save/restore MXCSR even though MXCSR is not part of AVX state. This does not happen when compacted format is used.
- Few AVX instructions such as VZEROUPPER/VZEROALL may operate on future expansion of YMM registers.

An operating system must enable its YMM state management to support AVX and any 256-bit extensions that operate on YMM registers. Otherwise, an attempt to execute an instruction in AVX extensions (including an enhanced 128-bit SIMD instructions using VEX encoding) will cause a #UD exception.

AVX instructions may generate SIMD floating-point exceptions. An OS must enable SIMD floating-point exception support by setting CR4.OSXMMEXCPT[bit 10]=1.

13.8.2 Intel® Advanced Vector Extensions 512 (Intel® AVX-512)

Intel AVX-512 instructions are encoded using EVEX prefix. The EVEX encoding scheme can support 512-bit, 256-bit and 128-bit instructions that operate on opmask, ZMM, YMM and XMM registers.

For processors that support the Intel AVX-512 family of instructions, the extended processor states (ZMM and opmask registers) exist in all operating modes. However, the access to these states may vary in different modes. The processor's support for instruction extensions that employ EVEX prefix encoding is independent of the processor's support for using XSAVE feature set on those states.

Instructions requiring EVEX prefix encoding are generally supported in 64-bit, 32-bit modes, and 16-bit protected mode. They are not supported in Real mode, Virtual-8086 mode or entering into SMM mode. Note that bits MAX_VL-1:256 (511:256) of ZMM register state are maintained across transitions into and out of these modes. Because the XSAVE feature set instruction can operate in all operating modes, it is possible that the processor's ZMM register state can be modified by software in any operating mode by executing XRSTOR.

Operating systems must use the XSAVE/XRSTOR/XSAVEOPT instructions for ZMM and opmask state management. An OS must enable its ZMM and opmask state management to support Intel AVX-512 Foundation instructions. Otherwise, an attempt to execute an instruction in Intel AVX-512 Foundation instructions (including a scalar 128-bit SIMD instructions using EVEX encoding) will cause a #UD exception. An operating system, which enables the AVX-512 state to support Intel AVX-512 Foundation instructions, is also sufficient to support the rest of the Intel AVX-512 family of instructions. Note that even though ZMM8-ZMM31 are not accessible in 32 bit mode, a 32 bit OS is still required to allocate memory for the entire ZMM state.

Intel AVX-512 Foundation instructions may generate SIMD floating-point exceptions. An OS must enable SIMD floating point exception support by setting CR4.OSXMMEXCPT[bit 10]=1.

...

19. Updates to Chapter 14, Volume 3B

Change bars show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

14.4.3 HWP Performance Range and Dynamic Capabilities

The OS reads the IA32_HWP_CAPABILITIES MSR to comprehend the limits of the HWP-managed performance range as well as the dynamic capability, which may change during processor operation. The enumerated performance range values reported by IA32_HWP_CAPABILITIES directly map to initial frequency targets (prior to workload-specific frequency optimizations of HWP). However the mapping is processor family specific.

The layout of the IA32_HWP_CAPABILITIES MSR is shown in Figure 14-6. The bit fields are described below:

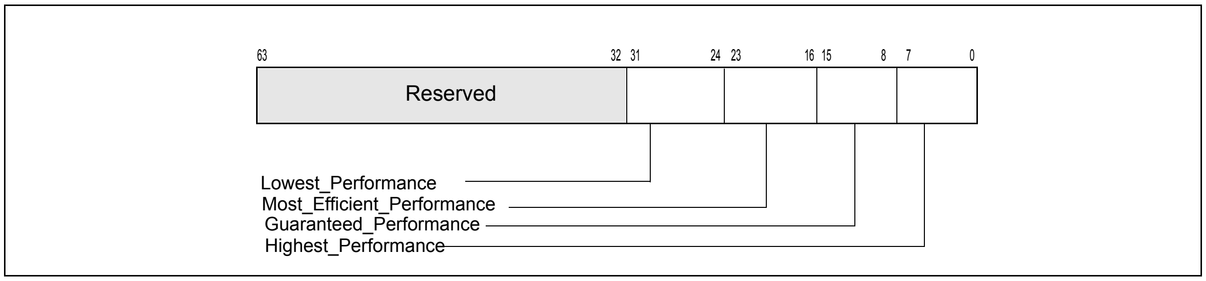


Figure 14-6 IA32_HWP_CAPABILITIES Register

...

20. 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.

...

17.2 DEBUG REGISTERS

Eight debug registers (see Figure 17-1 for 32-bit operation and Figure 17-2 for 64-bit operation) control the debug operation of the processor. These registers can be written to and read using the move to/from debug register form of the MOV instruction. A debug register may be the source or destination operand for one of these instructions.

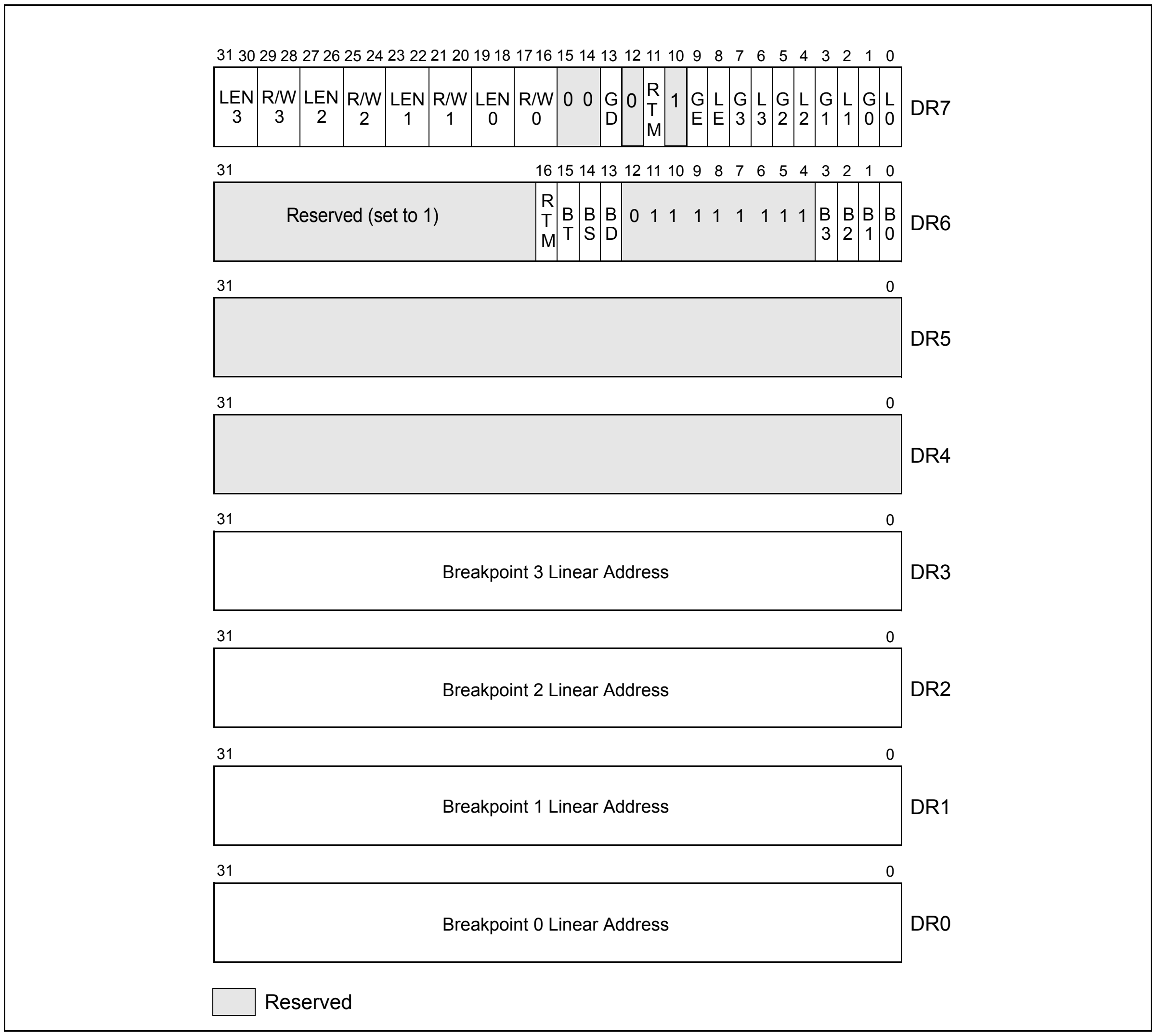


Figure 17-1 Debug Registers

...

17.4.1 IA32_DEBUGCTL MSR

The **IA32_DEBUGCTL** MSR provides bit field controls to enable debug trace interrupts, debug trace stores, trace messages enable, single stepping on branches, last branch record recording, and to control freezing of LBR stack or performance counters on a PMI request. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 17-3 for the MSR layout and the bullets below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0) —** When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the Section 17.5.1, "LBR Stack" (Intel® Core™2 Duo and Intel® Atom™ Processor Family) and Section 17.6.1, "LBR Stack" (processors based on Intel® Microarchitecture code name Nehalem).

- **BTF (single-step on branches) flag (bit 1) —** When set, the processor treats the TF flag in the EFLAGS register as a "single-step on branches" flag rather than a "single-step on instructions" flag. This mechanism allows single-stepping the processor on taken branches. See Section 17.4.3, "Single-Stepping on Branches," for more information about the BTF flag.
- **TR (trace message enable) flag (bit 6) —** When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 17.4.4, "Branch Trace Messages," for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7) —** When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 17.4.9, "BTS and DS Save Area."
- **BTINT (branch trace interrupt) flag (bit 8) —** When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 17.4.5, "Branch Trace Store (BTS)," for a description of this mechanism.

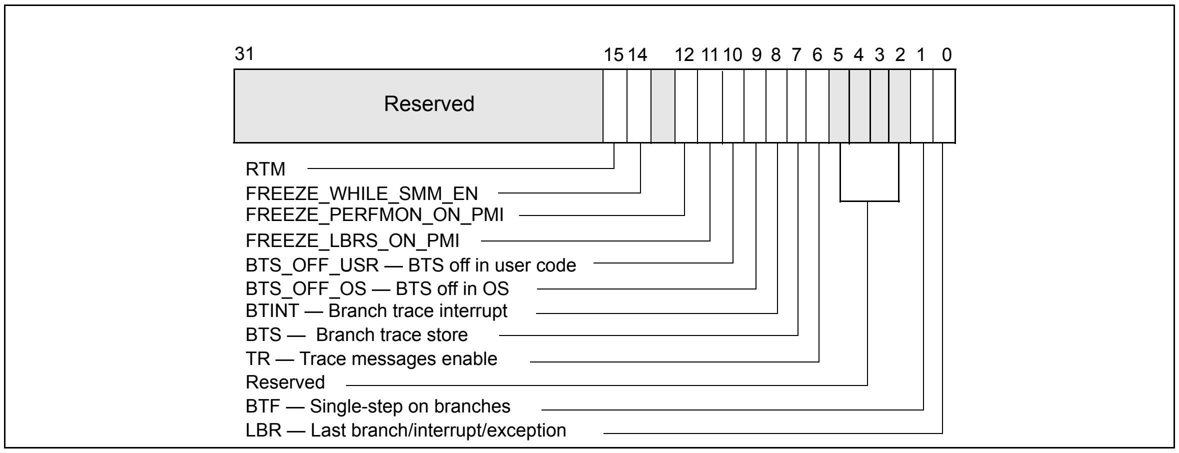

31
15 14
12 11 10 9 8 7 6 5 4 3 2 1 0
Reserved
RTM
FREEZE_WHILE_SMM_EN
FREEZE_PERFMON_ON_PMI
FREEZE_LBRS_ON_PMI
BTS_OFF_USR — BTS off in user code
BTS_OFF_OS — BTS off in OS
BTINT — Branch trace interrupt
BTS — Branch trace store
TR — Trace messages enable
Reserved
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

Figure 17-3 IA32_DEBUGCTL MSR for Processors based on Intel Core microarchitecture

- **BTS_OFF_OS (branch trace off in privileged code) flag (bit 9) —** When set, BTS or BTM is skipped if CPL is 0. See Section 17.10.2.
- **BTS_OFF_USR (branch trace off in user code) flag (bit 10) —** When set, BTS or BTM is skipped if CPL is greater than 0. See Section 17.10.2.
- **FREEZE_LBRS_ON_PMI flag (bit 11) —** When set, the LBR stack is frozen on a hardware PMI request (e.g. when a counter overflows and is configured to trigger PMI). See Section 17.4.7 for details.
- **FREEZE_PERFMON_ON_PMI flag (bit 12) —** When set, the performance counters (IA32_PMCx and IA32_FIXED_CTRx) are frozen on a PMI request. See Section 17.4.7 for details.
- **FREEZE_WHILE_SMM_EN (bit 14) —** If this bit is set, upon the delivery of an SMI, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler. Subsequently, the enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its service. Note that system software must check if the processor supports the IA32_DEBUGCTL.FREEZE_WHILE_SMM_EN control bit. IA32_DEBUGCTL.FREEZE_WHILE_SMM_EN is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 18.16 for details of detecting the presence of IA32_PERF_CAPABILITIES MSR.

- **RTM (bit 15)** — If this bit is set, advanced debugging of RTM transactional regions is enabled if DR7.RTM is also set. See Section 17.3.3.

...

17.4.7 Freezing LBR and Performance Counters on PMI

Many issues may generate a performance monitoring interrupt (PMI); a PMI service handler will need to determine cause to handle the situation. Two capabilities that allow a PMI service routine to improve branch tracing and performance monitoring are available for processors supporting architectural performance monitoring version 2 or greater (i.e. CPUID.0AH:EAX[7:0] > 1). These capabilities provides the following interface in IA32_DEBUGCTL to reduce runtime overhead of PMI servicing, profiler-contributed skew effects on analysis or counter metrics:

- **Freezing LBRs on PMI (bit 11)**— Allows the PMI service routine to ensure the content in the LBR stack are associated with the target workload and not polluted by the branch flows of handling the PMI. Depending on the version ID enumerated by CPUID.0AH:EAX.ArchPerfMonVerID[bits 7:0], two flavors are supported:
  - — Legacy Freeze_LBR_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID >1. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the LBR is frozen on the overflowed condition of the buffer area, the processor clears the LBR bit (bit 0) in IA32_DEBUGCTL. Software must then re-enable IA32_DEBUGCTL.LBR to resume recording branches. When using this feature, software should be careful about writes to IA32_DEBUGCTL to avoid re-enabling LBRs by accident if they were just disabled.
  - — Streamlined Freeze_LBR_on_PMI is supported for ArchPerfMonVerID >= 4. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the processor behaves as follows:
    - sets IA32_PERF_GLOBAL_STATUS.LBR_Frz =1 to disable recording, but does not change the LBR bit (bit 0) in IA32_DEBUGCTL. The LBRs are frozen on the overflowed condition of the buffer area.
- **Freezing PMCs on PMI** (bit 12) — Allows the PMI service routine to ensure the content in the performance counters are associated with the target workload and not polluted by the PMI and activities within the PMI service routine. Depending on the version ID enumerated by CPUID.0AH:EAX.ArchPerfMonVerID[bits 7:0], two flavors are supported:
  - — Legacy Freeze_Perfmon_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID >1. If IA32_DEBUGCTL.Freeze_Perfmon_On_PMI = 1, the performance counters are frozen on the counter overflowed condition when the processor clears the IA32_PERF_GLOBAL_CTRL MSR (see Figure 18-3). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 18.4.1, "Fixed-function Performance Counters"). Software must re-enable counts by writing 1s to the corresponding enable bits in IA32_PERF_GLOBAL_CTRL before leaving a PMI service routine to continue counter operation.
  - — Streamlined Freeze_Perfmon_on_PMI is supported for ArchPerfMonVerID >= 4. The processor behaves as follows:
    - sets IA32_PERF_GLOBAL_STATUS.CTR_Frz =1 to disable counting on a counter overflow condition, but does not change the IA32_PERF_GLOBAL_CTRL MSR.

Freezing LBRs and PMCs on PMIs (both legacy and streamlined operation) occur when one of the following applies:

- A performance counter had an overflow and was programmed to signal a PMI in case of an overflow.
  - — For the general-purpose counters; enabling PMI is done by setting bit 20 of the IA32_PERFEVTSELx register.
  - — For the fixed-function counters; enabling PMI is done by setting the 3rd bit in the corresponding 4-bit control field of the MSR_PERF_FIXED_CTR_CTRL register (see Figure 18-1) or IA32_FIXED_CTR_CTRL MSR (see Figure 18-2).
- The PEBS buffer is almost full and reaches the interrupt threshold.

- The BTS buffer is almost full and reaches the interrupt threshold.

Table 17-3 compares the interaction of the processor with the PMI handler using the legacy versus streamlined Freeza_Perfmon_On_PMI interface.

Table 17-3 Legacy and Streamlined Operation with Freeze_Perfmon_On_PMI = 1, Counter Overflowed

Legacy Freeze_Perfmon_On_PMI	Streamlined Freeze_Perfmon_On_PMI	Comment
Processor freezes the counters on overflow	Processor freezes the counters on overflow	Unchanged
Processor clears IA32_PERF_GLOBAL_CTRL	Processor set IA32_PERF_GLOBAL_STATUS.CTR_FTZ	
Handler reads IA32_PERF_GLOBAL_STATUS (0x38E) to examine which counter(s) overflowed	**mask** = RDMSR(0x38E)	Similar
Handler services the PMI	Handler services the PMI	Unchanged
Handler writes 1s to IA32_PERF_GLOBAL_OVF_CTL (0x390)	Handler writes **mask** into IA32_PERF_GLOBAL_OVF_RESET (0x390)	
Processor clears IA32_PERF_GLOBAL_STATUS	Processor clears IA32_PERF_GLOBAL_STATUS	Unchanged
Handler re-enables IA32_PERF_GLOBAL_CTRL	None	Reduced software overhead

17.4.8 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel 64 and IA-32 processor families. However, the number of MSRs in the LBR stack and the valid range of TOS pointer value can vary between different processor families. Table 17-4 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily_DisplayModel encoding (see CPUID instruction in Chapter 3 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*).

Table 17-4 LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Component of an LBR Entry	Range of TOS Pointer
06_4EH, 06_5EH	32	FROM_IP, TO_IP, LBR_INFO[1]	0 to 31
06_3DH, 06_47H, 06_4FH, 06_56H	16	FROM_IP, TO_IP	0 to 15
06_3CH, 06_45H, 06_46H, 06_3FH	16	FROM_IP, TO_IP	0 to 15
06_2AH, 06_2DH, 06_3AH, 06_3EH	16	FROM_IP, TO_IP	0 to 15
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	16	FROM_IP, TO_IP	0 to 15
06_17H, 06_1DH	4	FROM_IP, TO_IP	0 to 3
06_0FH	4	FROM_IP, TO_IP	0 to 3
06_37H, 06_4AH, 06_4CH, 06_4DH, 06_5AH, 06_5DH	8	FROM_IP, TO_IP	0 to 7
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	8	FROM_IP, TO_IP	0 to 7

NOTES:
1. See Section 17.9.

The last branch recording mechanism tracks not only branch instructions (like JMP, Jcc, LOOP and CALL instructions), but also other operations that cause a change in the instruction pointer (like external interrupts, traps and faults). The branch recording mechanisms generally employs a set of MSRs, referred to as last branch record (LBR) stack. The size and exact locations of the LBR stack are generally model-specific (see Chapter 35, "Model-Specific Registers (MSRs)" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* for model-specific MSR addresses).

- **Last Branch Record (LBR) Stack —** The LBR consists of N pairs of MSRs (N is listed in the LBR stack size column of Table 17-4) that store source and destination address of recent branches (see Figure 17-3):
  - MSR_LASTBRANCH_0_FROM_IP (address is model specific) through the next consecutive (N-1) MSR address store source addresses
  - MSR_LASTBRANCH_0_TO_IP (address is model specific) through the next consecutive (N-1) MSR address store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant M bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address is model specific) contains an M-bit pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. The valid range of the M-bit POS pointer is given in Table 17-4.

17.4.8.1 LBR Stack and Intel® 64 Processors

LBR MSRs are 64-bits. If IA-32e mode is disabled, only the lower 32-bits of the address is recorded. If IA-32e mode is enabled, the processor writes 64-bit values into the MSR.

In 64-bit mode, last branch records store 64-bit addresses; in compatibility mode, the upper 32-bits of last branch records are cleared.

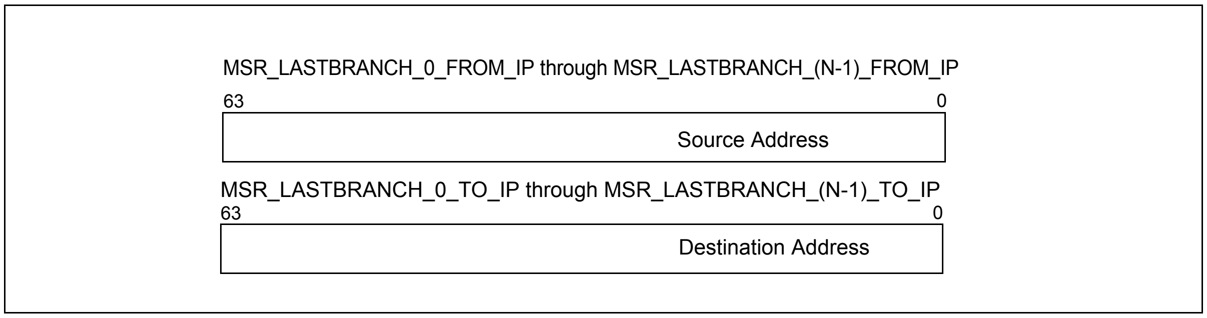

MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_(N-1)_FROM_IP
63
0
Source Address
MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_(N-1)_TO_IP
63
0
Destination Address

Figure 17-4 64-bit Address Layout of LBR MSR

Software should query an architectural MSR IA32_PERF_CAPABILITIES[5:0] about the format of the address that is stored in the LBR stack. Four formats are defined by the following encoding:

- **000000B (32-bit record format) —** Stores 32-bit offset in current CS of respective source/destination,
- **000001B (64-bit LIP record format) —** Stores 64-bit linear address of respective source/destination,
- **000010B (64-bit EIP record format) —** Stores 64-bit offset (effective address) of respective source/destination.
- **000011B (64-bit EIP record format) and Flags** — Stores 64-bit offset (effective address) of respective source/destination. LBR flags are supported in the upper bits of 'FROM' register in the LBR stack. See LBR stack details below for flag support and definition.
- **000100B (64-bit EIP record format), Flags and TSX** — Stores 64-bit offset (effective address) of respective source/destination. LBR flags and TSX info are supported in the upper bits of 'FROM' register in the LBR stack.

— **000101B (64-bit EIP record format), Flags, TSX, LBR_INFO** — Stores 64-bit offset (effective address) of respective source/destination. LBR flags, TSX, and elapsed cycle from last LBR update are supported in the LBR_INFO MSR stack.

Processor's support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX[PERF_CAPAB_MSR] (bit 15).

...

17.5 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ 2 DUO AND INTEL® ATOM™ PROCESSOR FAMILY)

The Intel Core 2 Duo processor family and Intel Xeon processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture provide last branch interrupt and exception recording. The facilities described in this section also apply to Intel Atom processor family. These capabilities are similar to those found in Pentium 4 processors, including support for the following facilities:

- **Debug Trace and Branch Recording Control —** The IA32_DEBUGCTL MSR provide bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 17.4.1 for a description of the flags. See Figure 17-3 for the MSR layout.
- **Last branch record (LBR) stack —** There are a collection of MSR pairs that store the source and destination addresses related to recently executed branches. See Section 17.5.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts**
  - — See Section 17.4.2 and Section 17.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.
  - — The Intel Atom processor family clears the TR flag when the FREEZE_LBRS_ON_PMI flag is set.
- **Branch trace messages —** See Section 17.4.4.
- **Last exception records —** See Section 17.10.3.
- **Branch trace store and CPL-qualified BTS —** See Section 17.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11) —** see Section 17.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12) —** see Section 17.4.7 for legacy Freeze_Perfmon_On_PMI operation.
- **FREEZE_WHILE_SMM_EN (bit 14) —** FREEZE_WHILE_SMM_EN is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 17.4.1.

...

17.6 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME NEHALEM

The processors based on Intel® microarchitecture code name Nehalem and Intel® microarchitecture code name Westmere support last branch interrupt and exception recording. These capabilities are similar to those found in Intel Core 2 processors and adds additional capabilities:

- **Debug Trace and Branch Recording Control —** The IA32_DEBUGCTL MSR provides bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 17.4.1 for a description of the flags. See Figure 17-11 for the MSR layout.

- **Last branch record (LBR) stack —** There are 16 MSR pairs that store the source and destination addresses related to recently executed branches. See Section 17.6.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts —** See Section 17.4.2 and Section 17.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.
- **Branch trace messages —** The IA32_DEBUGCTL MSR provides bit fields for software to enable each logical processor to generate branch trace messages. See Section 17.4.4. However, not all BTM messages are observable using the Intel® QPI link.
- **Last exception records —** See Section 17.10.3.
- **Branch trace store and CPL-qualified BTS —** See Section 17.4.6 and Section 17.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11) —** see Section 17.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12) —** see Section 17.4.7 for legacy Freeze_Perfmon_On_PMI operation.
- **UNCORE_PMI_EN (bit 13) —** When set. this logical processor is enabled to receive an counter overflow interrupt form the uncore.
- **FREEZE_WHILE_SMM_EN (bit 14) —** FREEZE_WHILE_SMM_EN is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 17.4.1.

Processors based on Intel microarchitecture code name Nehalem provide additional capabilities:

- **Independent control of uncore PMI —** The IA32_DEBUGCTL MSR provides a bit field (see Figure 17-11) for software to enable each logical processor to receive an uncore counter overflow interrupt.
- **LBR filtering —** Processors based on Intel microarchitecture code name Nehalem support filtering of LBR based on combination of CPL and branch type conditions. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT.

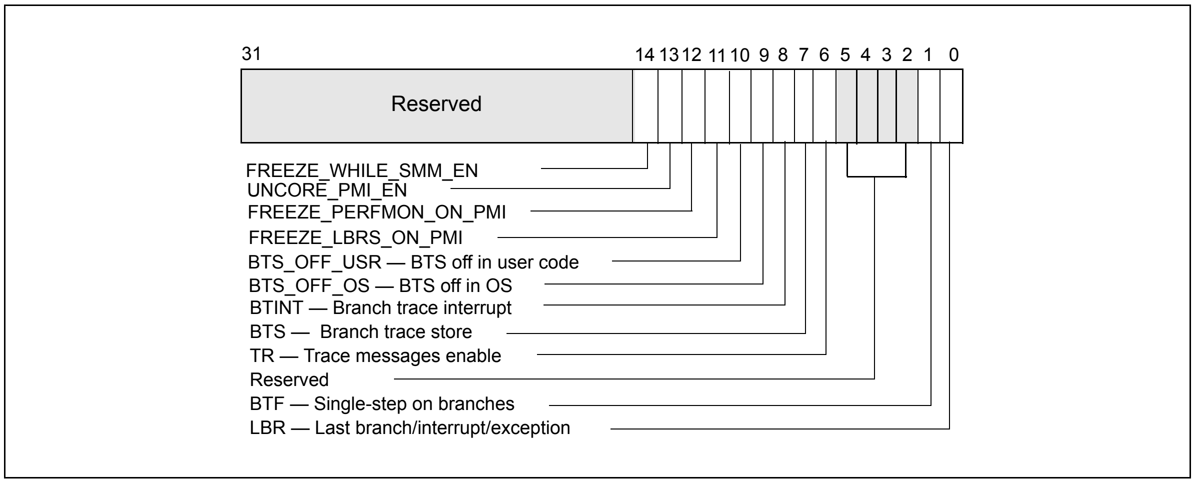

31
14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Reserved
FREEZE_WHILE_SMM_EN
UNCORE_PMI_EN
FREEZE_PERFMON_ON_PMI
FREEZE_LBRS_ON_PMI
BTS_OFF_USR — BTS off in user code
BTS_OFF_OS — BTS off in OS
BTINT — Branch trace interrupt
BTS — Branch trace store
TR — Trace messages enable
Reserved
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

Figure 17-11 IA32_DEBUGCTL MSR for Processors based on Intel microarchitecture code name Nehalem

17.6.1 LBR Stack

Processors based on Intel microarchitecture code name Nehalem provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is shown in Table 17-7 and Table 17-8.

Table 17-7 MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/O	The linear address of the branch instruction itself, this is the “branch from“ address.
SIGN_EXt	62:48	R/0	Signed extension of bit 47 of this register.
MISPRED	63	R/O	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

Table 17-8 MSR_LASTBRANCH_x_TO_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/O	The linear address of the target of the branch instruction itself, this is the “branch to“ address.
SIGN_EXt	63:48	R/0	Signed extension of bit 47 of this register.

Processors based on Intel microarchitecture code name Nehalem have an LBR MSR Stack as shown in Table 17-9.

...

17.8 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON HASWELL MICROARCHITECTURE

Generally, all of the last branch record, interrupt and exception recording facility described in Section 17.7, “Last Branch, Interrupt, and Exception Recording for Processors based on Intel® Microarchitecture code name Sandy Bridge”, apply to next generation processors based on Intel microarchitecture code name Haswell.

The LBR facility also supports an alternate capability to profile call stack profiles. Configuring the LBR facility to conduct call stack profiling is by writing 1 to the MSR_LBR_SELECT.EN_CALLSTACK[bit 9]; see Table 17-12. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 17.7.

Table 17-12 MSR_LBR_SELECT for Intel® microarchitecture code name Haswell

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches occurring in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches occurring in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
EN_CALLSTACK[1]	9		Enable LBR stack to use LIFO filtering to capture Call stack profile
Reserved	63:10		Must be zero

NOTES:

1. Must set valid combination of bits 0-8 in conjunction with bit 9 (as described below), otherwise the contents of the LBR MSRs are undefined.

The call stack profiling capability is an enhancement of the LBR facility. The LBR stack is a ring buffer typically used to profile control flow transitions resulting from branches. However, the finite depth of the LBR stack often become less effective when profiling certain high-level languages (e.g. C++), where a transition of the execution flow is accompanied by a large number of leaf function calls, each of which returns an individual parameter to form the list of parameters for the main execution function call. A long list of such parameters returned by the leaf functions would serve to flush the data captured in the LBR stack, often losing the main execution context.

When the call stack feature is enabled, the LBR stack will capture unfiltered call data normally, but as return instructions are executed the last captured branch record is flushed from the on-chip registers in a last-in first-out (LIFO) manner. Thus, branch information relative to leaf functions will not be captured, while preserving the call stack information of the main line execution path.

The configuration of the call stack facility is summarized below:

- Set IA32_DEBUGCTL.LBR (bit 0) to enable the LBR stack to capture branch records. The source and target addresses of the call branches will be captured in the 16 pairs of From/To LBR MSRs that form the LBR stack.
- Program the Top of Stack (TOS) MSR that points to the last valid from/to pair. This register is incremented by 1, modulo 16, before recording the next pair of addresses.
- Program the branch filtering bits of MSR_LBR_SELECT (bits 0:8) as desired.
- Program the MSR_LBR_SELECT to enable LIFO filtering of return instructions with:
  - — The following bits in MSR_LBR_SELECT must be set to ‘1’: JCC, NEAR_IND_JMP, NEAR_REL_JMP, FAR_BRANCH, EN_CALLSTACK;
  - — The following bits in MSR_LBR_SELECT must be cleared: NEAR_REL_CALL, NEAR-IND_CALL, NEAR_RET;
  - — At most one of CPL_EQ_0, CPL_NEQ_0 is set.

Note that when call stack profiling is enabled, “zero length calls” are excluded from writing into the LBRs. (A “zero length call” uses the attribute of the call instruction to push the immediate instruction pointer on to the stack and then pops off that address into a register. This is accomplished without any matching return on the call.)

17.8.1 LBR Stack Enhancement

Processors based on Intel microarchitecture code name Haswell provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is enumerated by IA32_PERF_CAPABILITIES[5:0] = 04H, and is shown in Table 17-13 and Table 17-8.

Table 17-13 MSR_LASTBRANCH_x_FROM_IP with TSX Information

Bit Field	Bit Offset	Access	Description
Data	47:0	R/O	The linear address of the branch instruction itself, this is the “branch from“ address.
SIGN_EXT	60:48	R/O	Signed extension of bit 47 of this register.
TSX_ABORT	61	R/O	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region, or EIP of the RTM Abort Handler
IN_TSX	62	R/O	When set, indicates the entry occurred in a TSX region
MISPRED	63	R/O	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

17.9 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SKY LAKE MICROARCHITECTURE

Processors based on the Sky Lake microarchitecture provide a number of enhancement with storing last branch records:

- enumeration of new LBR format: encoding 001001b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 17.4.8.1.
- Each LBR stack entry consists of a triplets of MSRs:
  - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 17-7.
  - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 17-8.
  - MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data.
- Size of LBR stack increased to 32.

Processors based on the Sky Lake microarchitecture supports the same LBR filtering capabilities as described in Table 17-12.

Table 17-14 LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_4EH, 06_5EH	32	0 to 31

17.9.1 MSR_LBR_INFO_x MSR

The layout of each MSR_LBR_INFO_x MSR is shown in Table 17-15.

Table 17-15 MSR_LBR_INFO_x

Bit Field	Bit Offset	Access	Description
Cycle Count (saturating)	15:0	R/O	Elapsed core clocks since last update to the LBR stack
Reserved	60:16	R/O	Reserved
TSX_ABORT	61	R/O	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region OR EIP of the RTM Abort Handler
IN_TSX	62	R/O	When set, indicates the entry occurred in a TSX region.
MISPRED	63	R/O	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

17.9.2 Streamlined Freeze_LBRs_On_PMI Operation

The capability to freeze the content of LBR to maintain recorded data quality continues to use the same interface IA32_DEBUGCTL.Freeze_LBRs_ON_PMI. Architectural performance monitoring version 4 and above supports a streamlined Freeze_LBRs_ON_PMI operation for PMI service routine that replaces the legacy Freeze_LBRs_ON_PMI operation (see Section 17.4.7).

Table 17-16 compares the interaction of the processor with the PMI handler.

Table 17-16 Legacy and Streamlined Operation with Freeze_LBRs_On_PMI = 1, Buffer Full

Legacy Freeze_LBRs_On_PMI	**Streamlined Freeze_LBRs_On_PMI**	**Comment**
Processor freezes the LBR stack on PEBS buffer full	Processor freezes the LBR stack on PEBS buffer full	Unchanged
Processor clears IA32_DEBUGCTRL.LBR (0x1D9)	Processor set IA32_PERF_GLOBAL_STATUS.LBR_Frz	
dbgmask = RDMSR(0x1D9)	**mask** = RDMSR(0x38E)	
Handler services the PMI	Handler services the PMI	Unchanged
Updates **dbgmask** to include LBR for subsequent write to IA32_DEBUG_CTL	Handler writes **mask** into IA32_PERF_GLOBAL_OVF_RESET (0x390)	
NA	Processor clears IA32_PERF_GLOBAL_STATUS	
Handler writes **dbgmask** to re-enables IA32_DEBUGCTL.LBR	NA	Prevents race condition of MSR 0x1D9 being updated by the processor and handler

17.9.3 LBR behavior on software C6

The LBR contents are cleared when a software request successfully enters a C6 or deeper sleep-state. This includes the FROM, TO, INFO, LAST_BRANCH, LER and LBR_TOS registers. The LBR enable bit and LBR_FROZEN bit will not be changed. The LBR-time of the first LBR record inserted after an exit from such a C6 request will be zero.

...

21. Updates to Chapter 18, Volume 3B

Change bars show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

18.1 PERFORMANCE MONITORING OVERVIEW

Performance monitoring was introduced in the Pentium processor with a set of model-specific performance-monitoring counter MSRs. These counters permit selection of processor performance parameters to be monitored and measured. The information obtained from these counters can be used for tuning system and compiler performance.

In Intel P6 family of processors, the performance monitoring mechanism was enhanced to permit a wider selection of events to be monitored and to allow greater control events to be monitored. Next, Pentium 4 and Intel Xeon processors introduced a new performance monitoring mechanism and new set of performance events.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, Pentium 4, and Intel Xeon processors are not architectural. They are all model specific (not compatible among processor families). Intel Core Solo and Intel Core Duo processors support a set of architectural performance events and a set of non-architectural performance events. Processors based on Intel Core microarchitecture and Intel® Atom™ microarchitecture support enhanced architectural performance events and non-architectural performance events.

Starting with Intel Core Solo and Intel Core Duo processors, there are two classes of performance monitoring capabilities. The first class supports events for monitoring performance using counting or sampling usage. These events are non-architectural and vary from one processor model to another. They are similar to those available in Pentium M processors. These non-architectural performance monitoring events are specific to the microarchitecture and may change with enhancements. They are discussed in Section 18.3, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)." Non-architectural events for a given microarchitecture can not be enumerated using CPUID; and they are listed in Chapter 19, "Performance-Monitoring Events."

The second class of performance monitoring capabilities is referred to as architectural performance monitoring. This class supports the same counting and sampling usages, with a smaller set of available events. The visible behavior of architectural performance events is consistent across processor implementations. Availability of architectural performance monitoring capabilities is enumerated using the CPUID.0AH. These events are discussed in Section 18.2.

See also:

— Section 18.2, "Architectural Performance Monitoring"

— Section 18.3, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)"

— Section 18.4, "Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)"

— Section 18.5, "Performance Monitoring (Processors Based on Intel® Atom™ Microarchitecture)"

— Section 18.6, "Performance Monitoring (Processors Based on the Silvermont Microarchitecture)"

— Section 18.7, "Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Nehalem"

— Section 18.7.4, "Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Westmere"

— Section 18.8, "Performance Monitoring for Processors Based on Intel® Microarchitecture Code Name Sandy Bridge"

— Section 18.8.8, "Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility"

— Section 18.9, "3rd Generation Intel® Core™ Processor Performance Monitoring Facility"

— Section 18.10, "4th Generation Intel® Core™ Processor Performance Monitoring Facility"

— Section 18.11, "Intel® Core™ M Processor Performance Monitoring Facility"

— Section 18.12, "Next Generation Intel® Core™ Processor Performance Monitoring Facility"

— Section 18.13, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)"

— Section 18.14, "Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture"

— Section 18.17, "Performance Monitoring and Dual-Core Technology"

— Section 18.18, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache"

— Section 18.20, "Performance Monitoring (P6 Family Processor)"

— Section 18.21, "Performance Monitoring (Pentium Processors)"

...

18.2.2 Architectural Performance Monitoring Version 2

The enhanced features provided by architectural performance monitoring version 2 include the following:

- **Fixed-function performance counter register and associated control register** — Three of the architectural performance events are counted using three fixed-function MSRs (IA32_FIXED_CTR0 through IA32_FIXED_CTR2). Each of the fixed-function PMC can count only one architectural performance event.

  Configuring the fixed-function PMCs is done by writing to bit fields in the MSR (IA32_FIXED_CTR_CTRL) located at address 38DH. Unlike configuring performance events for general-purpose PMCs (IA32_PMCx) via UMASK field in (IA32_PERFEVTSELx), configuring, programming IA32_FIXED_CTR_CTRL for fixed-function PMCs do not require any UMASK.
- **Simplified event programming** — Most frequent operation in programming performance events are enabling/disabling event counting and checking the status of counter overflows. Architectural performance event version 2 provides three architectural MSRs:
  - — IA32_PERF_GLOBAL_CTRL allows software to enable/disable event counting of all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or any general-purpose PMCs via a single WRMSR.
  - — IA32_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single RDMSR.
  - — IA32_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single WRMSR.
- **PMI Overhead Mitigation** — Architectural performance monitoring version 2 introduces two bit field interface in IA32_DEBUGCTL for PMI service routine to accumulate performance monitoring data and LBR records with reduced perturbation from servicing the PMI. The two bit fields are:
  - — IA32_DEBUGCTL.Freeze_LBR_On_PMI(bit 11). In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 17.4.7 for details of the legacy Freeze LBRs on PMI control.
  - — IA32_DEBUGCTL.Freeze_PerfMon_On_PMI(bit 12). In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 17.4.7 for details of the legacy Freeze LBRs on PMI control.

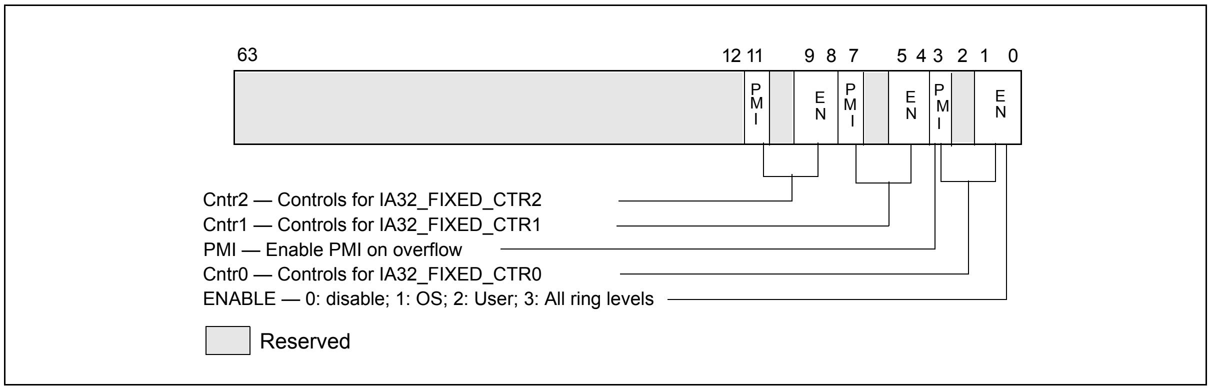
The facilities provided by architectural performance monitoring version 2 can be queried from CPUID leaf 0AH by examining the content of register EDX:

- Bits 0 through 4 of CPUID.0AH.EDX indicates the number of fixed-function performance counters available per core,
- Bits 5 through 12 of CPUID.0AH.EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.

NOTE

Early generation of processors based on Intel Core microarchitecture may report in CPUID.0AH:EDX of support for version 2 but indicating incorrect information of version 2 facilities.

The IA32_FIXED_CTR_CTRL MSR include multiple sets of 4-bit field, each 4 bit field controls the operation of a fixed-function performance counter. Figure 18-2 shows the layout of 4-bit controls for each fixed-function PMC. Two sub-fields are currently defined within each control. The definitions of the bit fields are:



63
12 11
9 8 7
5 4 3 2 1 0
PMI
EN
PMI
EN
PMI
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow
Cntr0 — Controls for IA32_FIXED_CTR0
ENABLE — 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

Figure 18-2 Layout of IA32_FIXED_CTR_CTRL MSR

- **Enable field (lowest 2 bits within each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring 0. When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring greater than 0. Writing 0 to both bits stops the performance counter. Writing a value of 11B enables the counter to increment irrespective of privilege levels.
- **PMI field (the fourth bit within each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

IA32_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting of each performance counter. Figure 18-3 shows the layout of IA32_PERF_GLOBAL_CTRL. Each enable bit in IA32_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

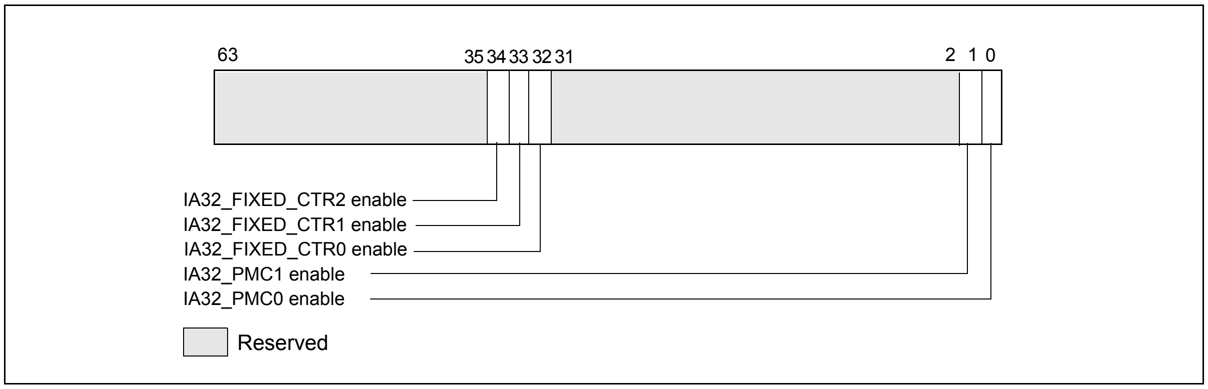


Figure 18-3 Layout of IA32_PERF_GLOBAL_CTRL MSR

The fixed-function performance counters supported by architectural performance version 2 is listed in Table 18-8, the pairing between each fixed-function performance counter to an architectural performance event is also shown.

IA32_PERF_GLOBAL_STATUS MSR provides single-bit status for software to query the overflow condition of each performance counter. The MSR also provides additional status bit to indicate overflow conditions when counters are programmed for precise-event-based sampling (PEBS). IA32_PERF_GLOBAL_STATUS MSR also provides a CondChgd bit to indicate changes to the state of performance monitoring hardware. Figure 18-4 shows the layout of IA32_PERF_GLOBAL_STATUS. A value of 1 in bits 0, 1, 32 through 34 indicates a counter overflow condition has occurred in the associated counter.

When a performance counter is configured for PEBS, overflow condition in the counter generates a performance-monitoring interrupt signaling a PEBS event. On a PEBS event, the processor stores data records into the buffer area (see Section 18.15.5), clears the counter overflow status., and sets the "OvfBuffer" bit in IA32_PERF_GLOBAL_STATUS.

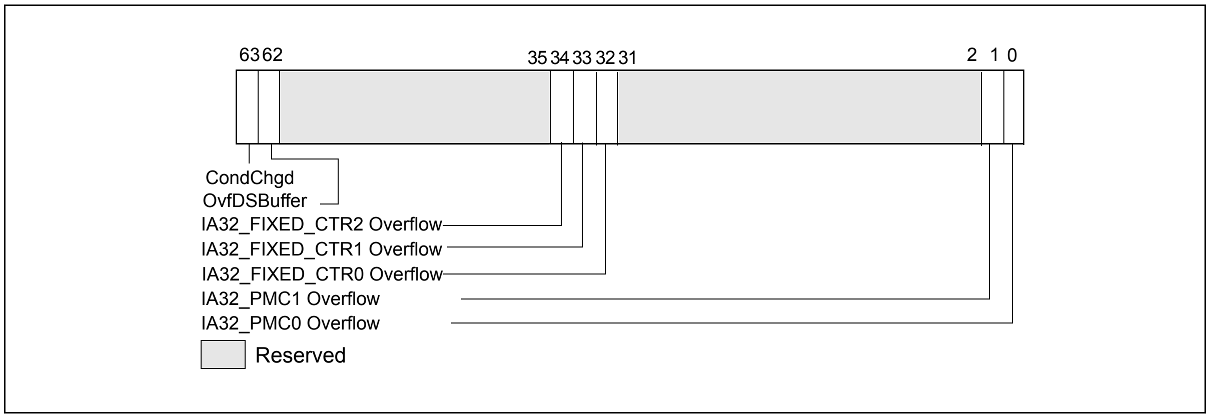


Figure 18-4 Layout of IA32_PERF_GLOBAL_STATUS MSR

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow indicator(s) of any general-purpose or fixed-function counters via a single WRMSR. Software should clear overflow indications when

- Setting up new values in the event select and/or UMASK field for counting or sampling
- Reloading counter values to continue sampling
- Disabling event counting or sampling.

The layout of IA32_PERF_GLOBAL_OVF_CTL is shown in Figure 18-5.

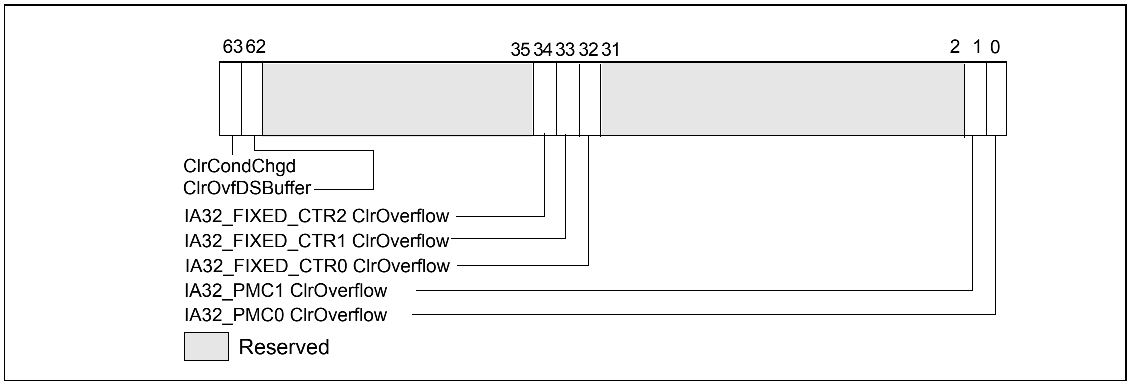


Figure 18-5 Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

18.2.3 Architectural Performance Monitoring Version 3

Processors supporting architectural performance monitoring version 3 also supports version 1 and 2, as well as capability enumerated by CPUID leaf 0AH. Specifically, version 3 provides the following enhancement in performance monitoring facilities if a processor core comprising of more than one logical processor, i.e. a processor core supporting Intel Hyper-Threading Technology or simultaneous multi-threading capability:

- Anythread counting for processor core supporting two or more logical processors. The interface that supports AnyThread counting include:
  — Each IA32_PERFEVTSELx MSR (starting at MSR address 186H) support the bit field layout defined in Figure 18-6.

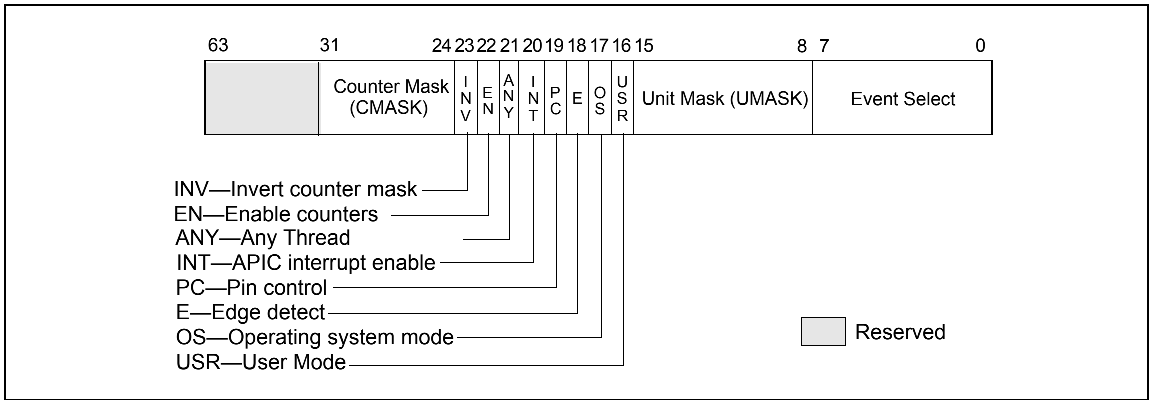


Figure 18-6 Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3 for processor core comprising of two or more logical processors. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core. When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.

— Each fixed-function performance counter IA32_FIXED_CTRx (starting at MSR address 309H) is configured by a 4-bit control block in the IA32_PERF_FIXED_CTR_CTRL MSR. The control block also allow thread-specificity configuration using an AnyThread bit. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown.

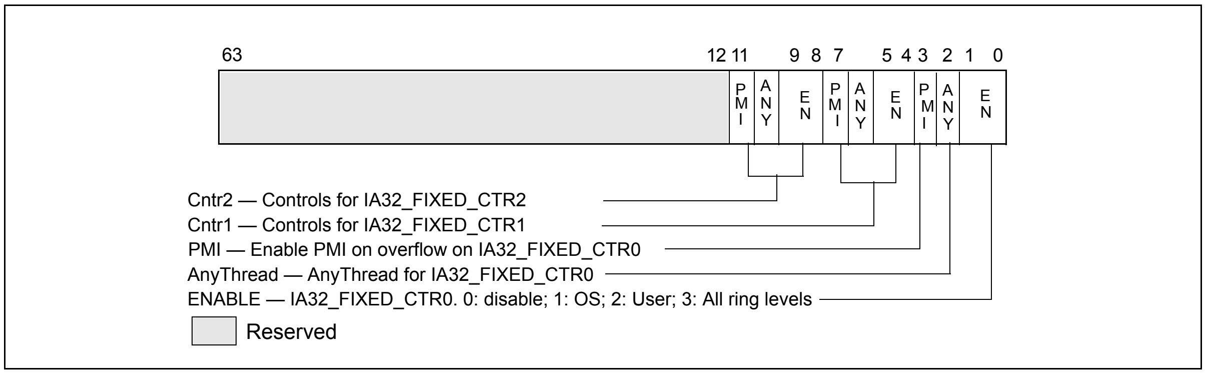

63
12 11
9 8 7
5 4 3 2 1 0
PMI
ANY
EN
PMI
ANY
EN
PMI
ANY
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow on IA32_FIXED_CTR0
AnyThread — AnyThread for IA32_FIXED_CTR0
ENABLE — IA32_FIXED_CTR0. 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

Figure 18-7 IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

Each control block for a fixed-function performance counter provides a **AnyThread** (bit position 2 + 4*N, N= 0, 1, etc.) bit. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the ENABLE setting of the corresponding control block of IA32_PERF_FIXED_CTR_CTRL) occurring across all logical processors sharing a processor core. When an **AnyThread** bit is 0 in IA32_PERF_FIXED_CTR_CTRL, the corresponding fixed counter only increments the associated event conditions occurring in the logical processor which programmed the IA32_PERF_FIXED_CTR_CTRL MSR.

- The IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL MSRs provide single-bit controls/status for each general-purpose and fixed-function performance counter. Figure 18-8 and Figure 18-9 show the layout of these MSRs for N general-purpose performance counters (where N is reported by CPUID.0AH:EAX[15:8]) and three fixed-function counters.

  Note: The Intel Atom processor family supports two general-purpose performance monitoring counters (i.e. N =2 in Figure 18-9), other processor families in Intel 64 architecture may support a different value of N in Figure 18-9. The number N is reported by CPUID.0AH:EAX[15:8]. The Intel Core i7 processor supports four general-purpose performance monitoring counters (i.e. N =4 in Figure 18-9).

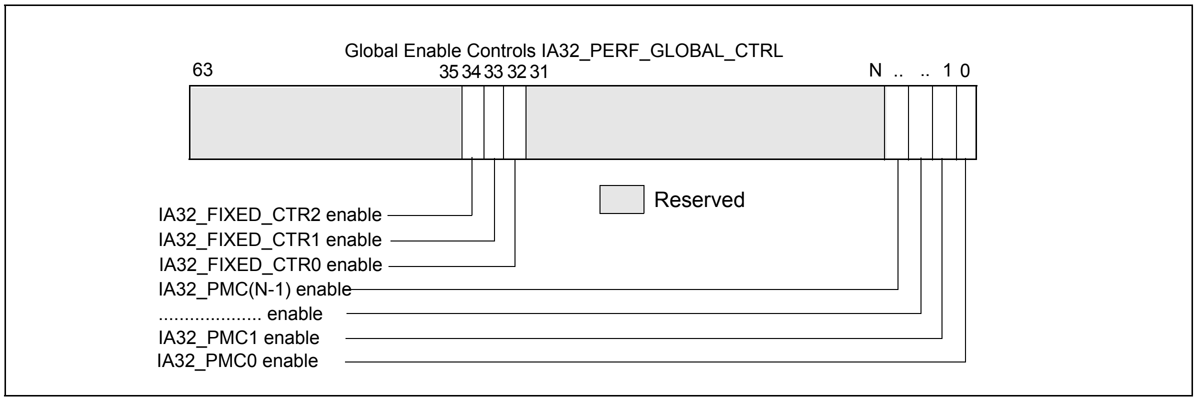


Figure 18-8 Layout of Global Performance Monitoring Control MSR

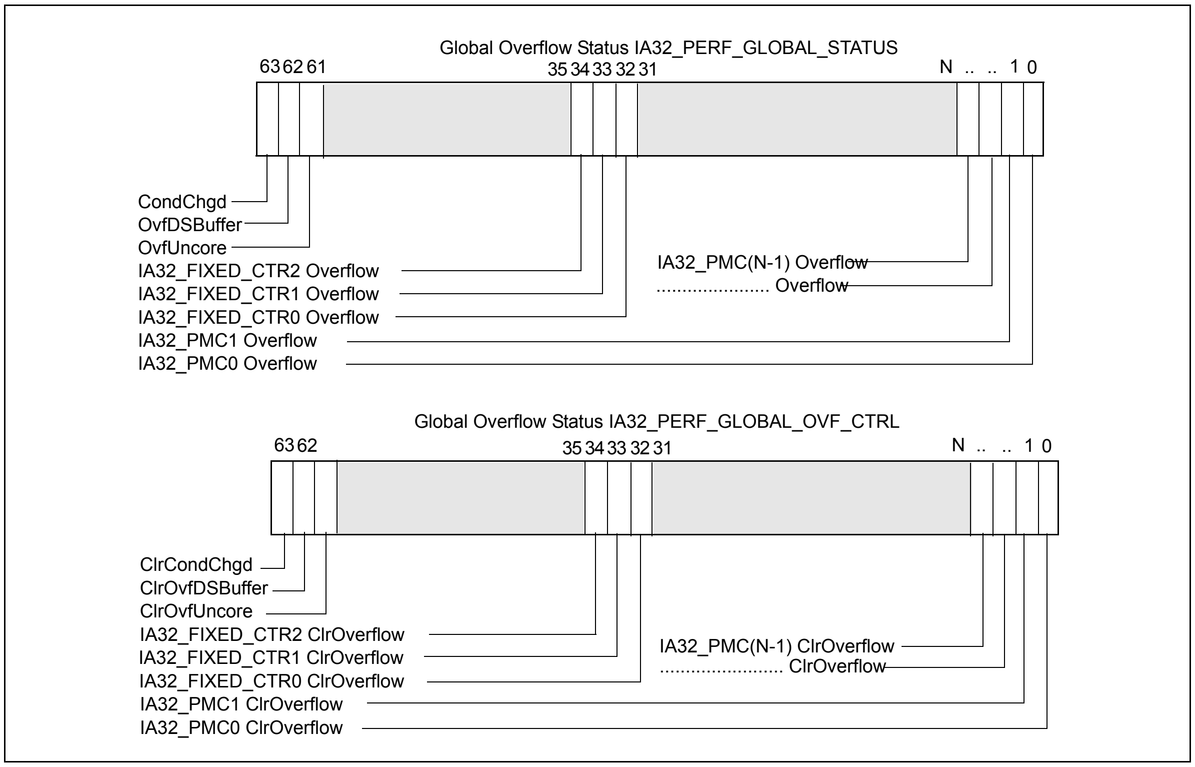


Figure 18-9 Global Performance Monitoring Overflow Status and Control MSRs

18.2.3.1 AnyThread Counting and Software Evolution

The motivation for characterizing software workload over multiple software threads running on multiple logical processors of the same processor core originates from a time earlier than the introduction of the AnyThread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL. While Anythread counting provides some benefits in simple software environments of an earlier era, the evolution contemporary software environments introduce certain concepts and pre-requisites that AnyThread counting does not comply with.

One example is the proliferation of software environments that support multiple virtual machines (VM) under VMX (see Chapter 23, “Introduction to Virtual-Machine Extensions”) where each VM represents a domain separated from one another.

A Virtual Machine Monitor (VMM) that manages the VMs may allow individual VM to employ performance monitoring facilities to profiles the performance characteristics of a workload. The use of the Anythread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL is discouraged with software environments supporting virtualization or requiring domain separation.

Specifically, Intel recommends VMM:

- configure the MSR bitmap to cause VM-exits for WRMSR to IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in VMX non-Root operation (see CHAPTER 24 for additional information),
- clear the AnyThread bit of IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in the MSR-load lists for VM exits and VM entries (see CHAPTER 24, CHAPTER 26, and CHAPTER 27).

Even when operating in simpler legacy software environments which might not emphasize the pre-requisites of a virtualized software environment, the use of the AnyThread interface should be moderated and follow any event-specific guidance where explicitly noted (see relevant sections of Chapter 19, “Performance Monitoring Events”).

18.2.4 Architectural Performance Monitoring Version 4

Processors supporting architectural performance monitoring version 4 also supports version 1, 2, and 3, as well as capability enumerated by CPUID leaf 0AH. Version 4 introduced a streamlined PMI overhead mitigation interface that replaces the legacy semantic behavior but retains the same control interface in IA32_DEBUGCTL.Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI. Specifically version 4 provides the following enhancement:

- New indicators (LBR_FRZ, CTR_FRZ) in IA32_PERF_GLOBAL_STATUS, see Section 18.2.4.1.
- Streamlined Freeze/PMI Overhead management interfaces to use IA32_DEBUGCTL.Freeze_LBRs_On_PMI and IA32_DEBUGCTL.Freeze_PerfMon_On_PMI: see Section 18.2.4.1. Legacy semantics of Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI (applicable to version 2 and 3) are not supported with version 4 or higher.
- Fine-grain separation of control interface to manage overflow/status of IA32_PERF_GLOBAL_STATUS and read-only performance counter enabling interface in IA32_PERF_GLOBAL_STATUS: see Section 18.2.4.2.
- Performance monitoring resource in-use MSR to facilitate cooperative sharing protocol between perfmon-managing privilege agents.

18.2.4.1 Enhancement in IA32_PERF_GLOBAL_STATUS

The IA32_PERF_GLOBAL_STATUS MSR provides the following indicators with architectural performance monitoring version 4:

- IA32_PERF_GLOBAL_STATUS.LBR_FRZ[bit 58]: This bit is set due to the following conditions:
  - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI has been set by the profiling agent, and
  - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently the LBR stack is frozen.

  Effectively, the IA32_PERF_GLOBAL_STATUS.LBR_FRZ bit also serve as an read-only control to enable capturing data in the LBR stack. To enable capturing LBR records, the following expression must hold with architectural perfmon version 4 or higher:

  - (IA32_DEBUGCTL.LBR & (!IA32_PERF_GLOBAL_STATUS.LBR_FRZ)) =1
- IA32_PERF_GLOBAL_STATUS.CTR_FRZ[bit 59]: This bit is set due to the following conditions:
  - IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI has been set by the profiling agent, and
  - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently, all the performance counters are frozen.

  Effectively, the IA32_PERF_GLOBAL_STATUS.CTR_FRZ bit also serve as an read-only control to enable programmable performance counters and fixed counters in the core PMU. To enable counting with the performance counters, the following expression must hold with architectural perfmon version 4 or higher:

  - (IA32_PERFEVTSELn.EN & IA32_PERF_GLOBAL_CTRL.PMCn & (!IA32_PERF_GLOBAL_STATUS.CTR_FRZ)) = 1 for programmable counter 'n', or
  - (IA32_PERF_FIXED_CRTL.ENi & IA32_PERF_GLOBAL_CTRL.FCi & (!IA32_PERF_GLOBAL_STATUS.CTR_FRZ)) = 1 for fixed counter 'i'

The read-only enable interface IA32_PERF_GLOBAL_STATUS.CTR_FRZ provides a more efficient flow for a PMI handler to use IA32_DEBUGCTL.Freeza_Perfmon_On_PMI to filter out data that may distort target workload analysis, see Table 17-3. It should be noted the IA32_PERF_GLOBAL_CTRL register continue to serve as the primary interface to control all performance counters of the logical processor.

For example, when the Freeze-On-PMI mode is not being used, a PMI handler would be setting IA32_PERF_GLOBAL_CTRL as the very last step to commence the overall operation after configuring the individual counter registers, controls and PEBS facility. This does not only assure atomic monitoring but also avoids unnecessary complications (e.g. race conditions) when software attempts to change the core PMU configuration while some counters are kept enabled.

Additionally, IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55]: On processors that support Intel Processor Trace and configured to store trace output packets to physical memory using the ToPA scheme, bit 55 is set when a PMI occurred due to a ToPA entry memory buffer was completely filled.

IA32_PERF_GLOBAL_STATUS also provides an indicator to distinguish interaction of performance monitoring operations with other side-band activities, which apply Intel SGX on processors that support SGX (For additional information about Intel SGX, see “Intel® Software Guard Extensions Programming Reference”.):

- IA32_PERF_GLOBAL_STATUS.ASCI[bit 60]: This bit is set when data accumulated in any of the configured performance counters (i.e. IA32_PMCx or IA32_FIXED_CTRx) may include contributions from direct or indirect operation of Intel SGX to protect an enclave (since the last time IA32_PERF_GLOBAL_STATUS.ASCI was cleared).

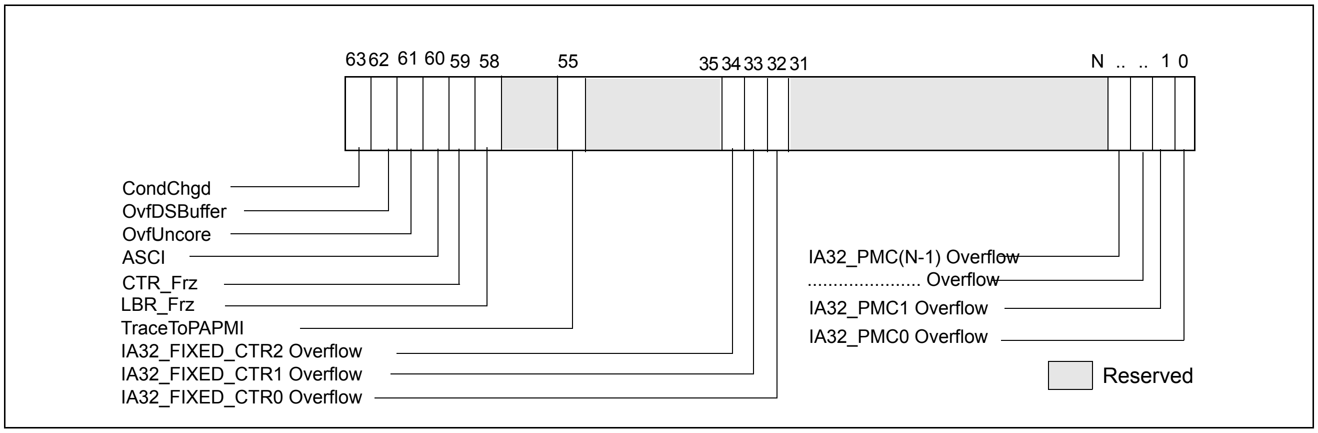


Figure 18-10 IA32_PERF_GLOBAL_STATUS MSR and Architectural Perfmon Version 4

Note, a processor’s support for IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55] is enumerated as a result of CPUID enumerated capability of Intel Processor Trace and the use of the ToPA buffer scheme. Support of IA32_PERF_GLOBAL_STATUS.ASCI[bit 60] is enumerated by the CPUID enumeration of Intel SGX.

18.2.4.2 IA32_PERF_GLOBAL_STATUS_RESET and IA32_PERF_GLOBAL_STATUS_SET MSRS

With architectural performance monitoring version 3 and lower, clearing of the set bits in IA32_PERF_GLOBAL_STATUS MSR by software is done via IA32_PERF_GLOBAL_OVF_CTRL MSR. Starting with architectural performance monitoring version 4, software can manage the overflow and other indicators in IA32_PERF_GLOBAL_STATUS using separate interfaces to set or clear individual bits.

The address and the architecturally-defined bits of IA32_PERF_GLOBAL_OVF_CTRL is inherited by IA32_PERF_GLOBAL_STATUS_RESET (see Figure 18-11). Further, IA32_PERF_GLOBAL_STATUS_RESET provides additional bit fields to clear the new indicators in IA32_PERF_GLOBAL_STATUS described in Section 18.2.4.1.

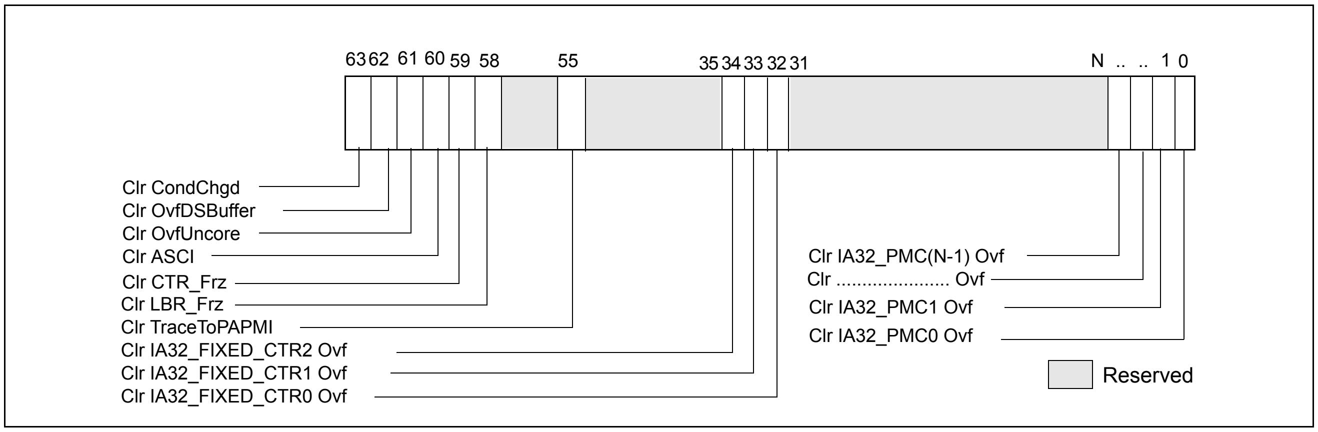


Figure 18-11 IA32_PERF_GLOBAL_STATUS_RESET MSR and Architectural Perfmon Version 4

The IA32_PERF_GLOBAL_STATUS_SET MSR is introduced with architectural performance monitoring version 4. It allows software to set individual bits in IA32_PERF_GLOBAL_STATUS. The IA32_PERF_GLOBAL_STATUS_SET interface can be used by a VMM to virtualize the state of IA32_PERF_GLOBAL_STATUS across VMs.

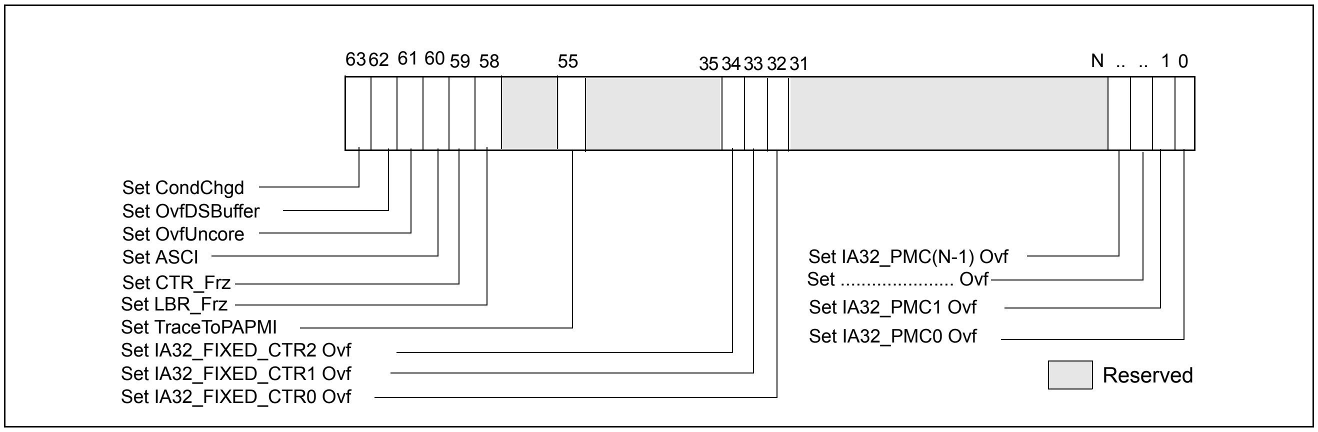


Figure 18-12 IA32_PERF_GLOBAL_STATUS_SET MSR and Architectural Perfmon Version 4

18.2.4.3 IA32_PERF_GLOBAL_INUSE MSR

In a contemporary software environment, multiple privileged service agents may wish to employ the processor's performance monitoring facilities. The IA32_MISC_ENABLES.PERFMON_AVAILABLE[bit 7] interface could not serve the need of multiple agent adequately. A white paper, "Performance Monitoring Unit Sharing Guideline"[1], proposed a cooperative sharing protocol that is voluntary for participating software agents.

Architectural performance monitoring version 4 introduces a new MSR, IA32_PERF_GLOBAL_INUSE, that simplifies the task of multiple cooperating agents to implement the sharing protocol.

The layout of IA32_PERF_GLOBAL_INUSE is shown in Figure 18-13.

1. Available at http://www.intel.com/sdm

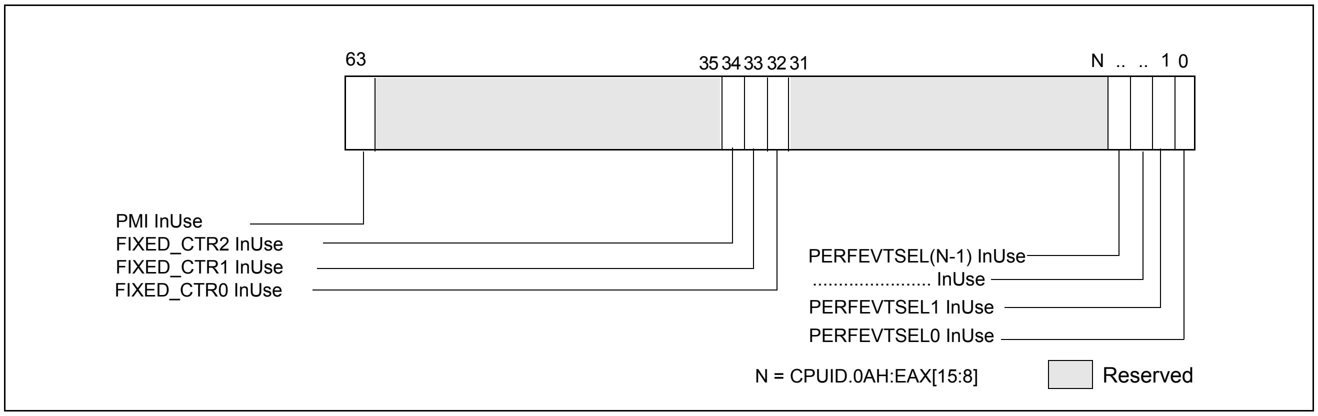


Figure 18-13 IA32_PERF_GLOBAL_INUSE MSR and Architectural Perfmon Version 4

The IA32_PERF_GLOBAL_INUSE MSR provides an "InUse" bit for each programmable performance counter and fixed counter in the processor. Additionally, it includes an indicator if the PMI mechanism has been configured by a profiling agent.

- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL0_InUse[bit 0]: This bit reflects the logical state of (IA32_PERFEVTSEL0[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL1_InUse[bit 1]: This bit reflects the logical state of (IA32_PERFEVTSEL1[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL2_InUse[bit 2]: This bit reflects the logical state of (IA32_PERFEVTSEL2[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSELn_InUse[bit n]: This bit reflects the logical state of (IA32_PERFEVTSELn[7:0] != 0), n < CPUID.0AH:EAX[15:8].
- IA32_PERF_GLOBAL_INUSE.FC0_InUse[bit 32]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[1:0] != 0).
- IA32_PERF_GLOBAL_INUSE.FC1_InUse[bit 33]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[5:4] != 0).
- IA32_PERF_GLOBAL_INUSE.FC2_InUse[bit 34]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[9:8] != 0).
- IA32_PERF_GLOBAL_INUSE.PMI_InUse[bit 32]: This bit is set if any one of the following bit is set:
  - — IA32_PERFEVTSELn.INT[bit 20], n < CPUID.0AH:EAX[15:8];
  - — IA32_FIXED_CTR_CTRL.ENi_PMI, i = 0, 1, 2;
  - — IA32_PEBS_ENABLES.EN_PMCi, i = 0, 1, 2, 3

...

18.4.4.2 PEBS Record Format

The PEBS record format may be extended across different processor implementations. The IA32_PERF_CAPABILITES MSR defines a mechanism for software to handle the evolution of PEBS record format in processors that support architectural performance monitoring with version id equals 2 or higher. The bit fields of IA32_PERF_CAPABILITES are defined in Table 35-2 of Chapter 35, "Model-Specific Registers (MSRs)". The relevant bit fields that governs PEBS are:

- PEBSTrap [bit 6]: When set, PEBS recording is trap-like. After the PEBS-enabled counter has overflowed, PEBS record is recorded for the next PEBS-able event at the completion of the sampled instruction causing the PEBS event. When clear, PEBS recording is fault-like. The PEBS record is recorded before the sampled instruction causing the PEBS event.
- PEBSSaveArchRegs [bit 7]: When set, PEBS will save architectural register and state information according to the encoded value of the PEBSRecordFormat field. When clear, only the return instruction pointer and flags are recorded. On processors based on Intel Core microarchitecture, this bit is always 1
- PEBSRecordFormat [bits 11:8]: Valid encodings are:
  - 0000B: Only general-purpose registers, instruction pointer and RFLAGS registers are saved in each PEBS record (seeSection 18.13.7).
  - 0001B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS and load latency data. (seeSection 18.7.1.1).
  - 0010B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS, load latency data, and TSX tuning information. (seeSection 18.10.2).
  - 0011B: PEBS record includes additional information of load latency data, TSX tuning information, TSC data, and the applicable counter field replaces IA32_PERF_GLOBAL_STATUS at offset 90H. (see Section 18.12.1.1).

...

18.8 PERFORMANCE MONITORING FOR PROCESSORS BASED ON INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel® Xeon® processor E3-1200 family are based on Intel microarchitecture code name Sandy Bridge; this section describes the performance monitoring facilities provided in the processor core. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 18.2.3) and a host of non-architectural monitoring capabilities.

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

The core PMU's capability is similar to those described in Section 18.7.1 and Section 18.7.4, with some differences and enhancements relative to Intel microarchitecture code name Westmere summarized in Table 18-25.

Table 18-25 Core PMU Comparison

Box	Intel® microarchitecture code name Sandy Bridge	Intel® microarchitecture code name Westmere	Comment
# of Fixed counters per thread	3	3	Use CPUID to enumerate # of counters.
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48 , W: 32/48	R:48, W:32	See Section 18.2.4.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to enumerate # of counters.

Table 18-25 Core PMU Comparison

Box	Intel® microarchitecture code name Sandy Bridge	Intel® microarchitecture code name Westmere	Comment
PMI Overhead Mitigation	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling ▪ Freeze_while_SMM	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling ▪ Freeze_while_SMM	See Section 17.4.7
Precise Event Based Sampling (PEBS) Events	See Table 18-27	See Table 18-10	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 18.8.4.2; ▪ Data source encoding, ▪ STLB miss encoding, ▪ Lock transaction encoding	Data source encoding	
PEBS-Precise Store	Section 18.8.4.3	No	
PEBS-PDIR	yes (using precise INST_RETIRED.ALL)	No	
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types	MSR 1A6H and 1A7H, limited response types	Nehalem supports 1A6H only.

...

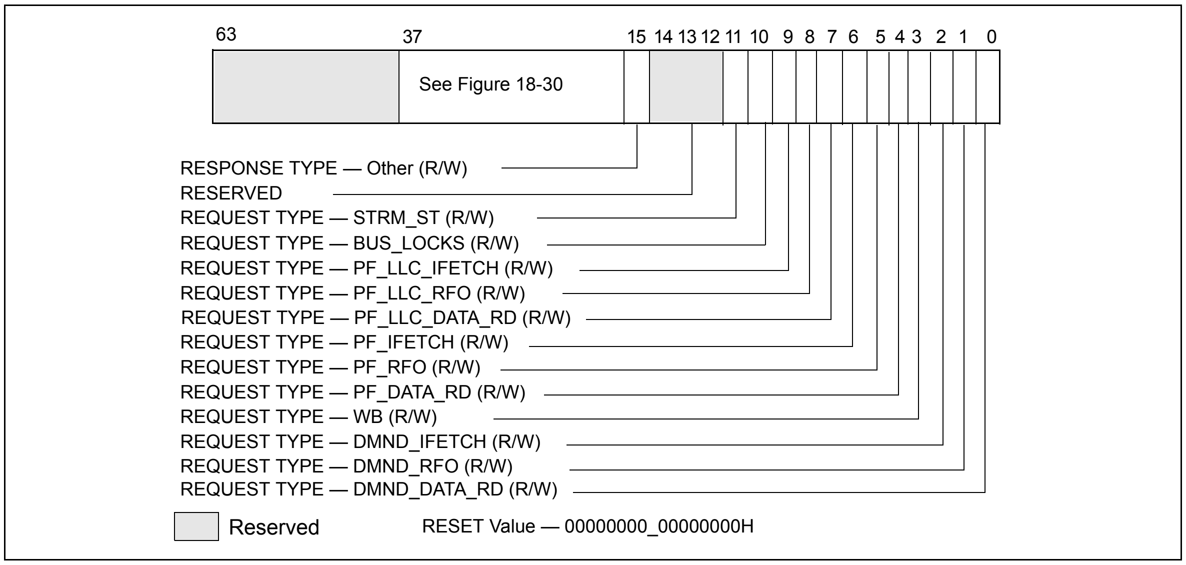
18.8.5 Off-core Response Performance Monitoring

The core PMU in processors based on Intel microarchitecture code name Sandy Bridge provides off-core response facility similar to prior generation. Off-core response can be programmed only with a specific pair of event select and counter MSR, and with specific event codes and predefine mask bit value in a dedicated MSR to specify attributes of the off-core transaction. Two event codes are dedicated for off-core response event programming. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Table 18-30 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 18-30 Off-Core Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-3	B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)
PMC0-3	BBH	01H	MSR_OFFCORE_RSP_1 (address 1A7H)

The layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 are shown in Figure 18-36 and Figure 18-37. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.



63
37
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
See Figure 18-30
RESPONSE TYPE — Other (R/W)
RESERVED
REQUEST TYPE — STRM_ST (R/W)
REQUEST TYPE — BUS_LOCKS (R/W)
REQUEST TYPE — PF_LLC_IFETCH (R/W)
REQUEST TYPE — PF_LLC_RFO (R/W)
REQUEST TYPE — PF_LLC_DATA_RD (R/W)
REQUEST TYPE — PF_IFETCH (R/W)
REQUEST TYPE — PF_RFO (R/W)
REQUEST TYPE — PF_DATA_RD (R/W)
REQUEST TYPE — WB (R/W)
REQUEST TYPE — DMND_IFETCH (R/W)
REQUEST TYPE — DMND_RFO (R/W)
REQUEST TYPE — DMND_DATA_RD (R/W)
Reserved
RESET Value — 00000000_00000000H

Figure 18-36 Request_Type Fields for MSR_OFFCORE_RSP_x

Table 18-31 MSR_OFFCORE_RSP_x Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	(R/W). Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	(R/W). Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	(R/W). Counts the number of demand and DCU prefetch instruction cacheline reads. Does not count L2 code read prefetches.
WB	3	(R/W). Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	(R/W). Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	(R/W). Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	(R/W). Counts the number of code reads generated by L2 prefetchers.
PF_LLC_DATA_RD	7	(R/W). L2 prefetcher to L3 for loads.
PF_LLC_RFO	8	(R/W). RFO requests generated by L2 prefetcher
PF_LLC_IFETCH	9	(R/W). L2 prefetcher to L3 for instruction fetches.
BUS_LOCKS	10	(R/W). Bus lock and split lock requests
STRM_ST	11	(R/W). Streaming store requests
OTHER	15	(R/W). Any other request that crosses IDI, including I/O.

...

18.8.8 Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5-2600 product family has some similarities with those of the Intel Xeon processor E7 family. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope. For example, each Cbox caching agent has a set of local performance counters, and the power controller unit (PCU) has its own local performance counters. Up to 8 C-Box units are supported in the uncore sub-system.

Table 18-36 summarizes the uncore PMU facilities providing MSR interfaces.

Table 18-36 Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	4	44	Yes	per-box	None
PCU	1	4	48	Yes	per-box	Match/Mask
U-Box	1	2	44	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 family is available in "Intel® Xeon® Processor E5 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 35-19.

...

18.9.1 Intel® Xeon® Processor E5 v2 and E7 v2 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5 v2 and Intel Xeon Processor E7 v2 product families are based on the Ivy Bridge-E microarchitecture. There are some similarities with those of the Intel Xeon processor E5 family based on the Sandy Bridge microarchitecture. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope.

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v2 and Intel Xeon Processor E7 v2 families are available in "Intel® Xeon® Processor E5 v2 and E7 v2 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 35-23.

18.10 4TH GENERATION INTEL® CORE™ PROCESSOR PERFORMANCE MONITORING FACILITY

The 4th generation Intel® Core™ processor and Intel® Xeon® processor E3-1200 v3 product family are based on the Haswell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 18.2.3) and a host of non-architectural monitoring capabilities.

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

The core PMU's capability is similar to those described in Section 18.8 through Section 18.8.5, with some differences and enhancements summarized in Table 18-37. Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 18.10.5. For details of Intel TSX, see Chapter 15, "Programming with Intel® Transactional Synchronization Extensions" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Table 18-37 Core PMU Comparison

Box	Intel® microarchitecture code name Haswell	Intel® microarchitecture code name Sandy Bridge	Comment
# of Fixed counters per thread	3	3	
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48 , W: 32/48	R:48 , W: 32/48	See Section 18.2.4.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to enumerate # of counters.
PMI Overhead Mitigation	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling ▪ Freeze_while_SMM	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling ▪ Freeze_while_SMM	See Section 17.4.7
Precise Event Based Sampling (PEBS) Events	See Table 18-27 and Section 18.10.5.1	See Table 18-27	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 18.8.4.2;	See Section 18.8.4.2;	
PEBS-Precise Store	No, replaced by Data Address profiling	Section 18.8.4.3	
PEBS-PDIR	yes (using precise INST_RETIRED.ALL)	yes (using precise INST_RETIRED.ALL)	
PEBS-EventingIP	yes	no	
Data Address Profiling	yes	no	
LBR Profiling	yes	yes	
Call Stack Profiling	yes, see Section 17.8	no	Use LBR facility
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types	MSR 1A6H and 1A7H; Extended request and response types	
Intel TSX support for Perfmon	See Section 18.10.5;	no	

18.10.1 Precise Event Based Sampling (PEBS) Facility

The PEBS facility in the 4th Generation Intel Core processor is similar to those in processors based on Intel micro-architecture code name Sandy Bridge, with several enhanced features. The key components and differences of PEBS facility relative to Intel microarchitecture code name Sandy Bridge is summarized in Table 18-38.

Table 18-38 PEBS Facility Comparison

Box	Intel® microarchitecture code name Haswell	Intel® microarchitecture code name Sandy Bridge	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 18.7.1.1	Section 18.7.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 18-21	Figure 18-35	

Table 18-38 PEBS Facility Comparison

Box	Intel® microarchitecture code name Haswell	Intel® microarchitecture code name Sandy Bridge	Comment
PEBS record layout	Table 18-39, Enhanced fields at offsets 98H, A0H, A8H, B0H	Table 18-18, Enhanced fields at offsets 98H, A0H, A8H	
PEBS Events	See Table 18-27	See Table 18-27	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 18-28	Table 18-28	
PEBS-Precise Store	no, replaced by data address profiling	yes; see Section 18.8.4.3	
PEBS-PDIR	yes	yes	IA32_PMC1 only
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

18.10.2 PEBS Data Format

The PEBS record format for the 4th Generation Intel Core processor is shown in Table 18-39. The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 18-39 PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	Data Linear Address

Table 18-39 PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
40H	R/EBP	A0H	Data Source Encoding
48H	R/ESP	A8H	Latency value (core cycles)
50H	R8	B0H	EventingIP
58H	R9	B8H	TX Abort Information (Section 18.10.5.1)

The layout of PEBS records are almost identical to those shown in Table 18-18. Offset B0H is a new field that records the eventing IP address of the retired instruction that triggered the PEBS assist.

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

In the core PMU of the 4th generation Intel Core processor, load latency facility and PDIR capabilities are unchanged. However, precise store is replaced by an enhanced capability, data address profiling, that is not restricted to store address. Data address profiling also records information in PEBS records at offsets 98H, A0H, and ABH.

18.10.3 PEBS Data Address Profiling

The Data Linear Address facility is also abbreviated as DataLA. The facility is a replacement or extension of the precise store facility in previous processor generations. The DataLA facility complements the load latency facility by providing a means to profile load and store memory references in the system, leverages the PEBS facility, and provides additional information about sampled loads and stores. Having precise memory reference events with linear address information for both loads and stores provides information to improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

The DataLA facility in the 4th generation processor supports the following events configured to use PEBS:

Table 18-40 Precise Events That Supports Data Linear Address Profiling

Event Name	Event Name
MEM_UOPS_RETIRED.STLB_MISS_LOADS	MEM_UOPS_RETIRED.STLB_MISS_STORES
MEM_UOPS_RETIRED.LOCK_LOADS	MEM_UOPS_RETIRED.SPLIT_STORES
MEM_UOPS_RETIRED.SPLIT_LOADS	MEM_UOPS_RETIRED.ALL_STORES
MEM_UOPS_RETIRED.ALL_LOADS	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM
MEM_LOAD_UOPS_RETIRED.L1_HIT	MEM_LOAD_UOPS_RETIRED.L2_HIT
MEM_LOAD_UOPS_RETIRED.L3_HIT	MEM_LOAD_UOPS_RETIRED.L1_MISS
MEM_LOAD_UOPS_RETIRED.L2_MISS	MEM_LOAD_UOPS_RETIRED.L3_MISS
MEM_LOAD_UOPS_RETIRED.HIT_LFB	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM
UOPS_RETIRED.ALL (if load or store is tagged)	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE

DataLA can use any one of the IA32_PMC0-IA32_PMC3 counters. Counter overflows will initiate the generation of PEBS records. Upon counter overflow, hardware captures the linear address and possible other status information of the retiring memory uop. This information is then written to the PEBS record that is subsequently generated.

To enable the DataLA facility, software must complete the following steps. Please note that the DataLA facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture DataLA information.

- Complete the PEBS configuration steps.
- Program the an event listed in Table 18-40 using any one of IA32_PERFEVTSEL0-IA32_PERFEVTSEL3.
- Set the corresponding IA32_PEBS_ENABLE.PEBS_EN_CTRx bit. This enables the corresponding IA32_PMCx as a PEBS counter and enables the DataLA facility.

When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H and A8H, as shown in Table 18-41.

Table 18-41 Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	▪ **DCU Hit** (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_UOPS_RETIRED.STLB_MISS_STORES, MEM_UOPS_RETIRED.LOCK_STORES, MEM_UOPS_RETIRED.SPLIT_STORES, MEM_UOPS_RETIRED.ALL_STORES ▪ Other bits are zero, The STLB_MISS, LOCK bit information can be obtained by programming the corresponding store event in Table 18-40.
Reserved	A8H	Always zero

...

18.10.4 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 18.8.5. The event codes are listed in Table 18-30. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 18-42.
- Supplier information (bits 30:16): see Table 18-43 and Table 18-43.
- Snoop response information (bits 37:31): see Table 18-33.

Table 18-42 MSR_OFFCORE_RSP_x Request_Type Definition (Haswell microarchitecture)

Bit Name	Offset	Description
DMND_DATA_RD	0	(R/W). Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	(R/W). Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	(R/W). Counts the number of demand and DCU prefetch instruction cacheline reads. Does not count L2 code read prefetches.
Reserved	3	Reserved
PF_DATA_RD	4	(R/W). Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	(R/W). Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	(R/W). Counts the number of code reads generated by L2 prefetchers.

Table 18-42 MSR_OFFCORE_RSP_x Request_Type Definition (Contd.)(Haswell microarchitecture)

Bit Name	Offset	Description
Reserved	7-14	Reserved
OTHER	15	(R/W). Any other request that crosses IDI, including I/O.

The supplier information field listed in Table 18-43 and Table 18-43. The fields vary across products (according to CPUID signatures) and is noted in the description.

Table 18-43 MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_3CH, 06_46H)

Subtype	Bit Name	Offset	Description
Common	Any	16	(R/W). Catch all value for any response types.
Supplier Info	NO_SUPP	17	(R/W). No Supplier Information available
	L3_HITM	18	(R/W). M-state initial lookup stat in L3.
	L3_HITE	19	(R/W). E-state
	L3_HITS	20	(R/W). S-state
	Reserved	21	Reserved
	LOCAL	22	(R/W). Local DRAM Controller
	Reserved	30:23	Reserved

Table 18-44 MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_45H)

Subtype	Bit Name	Offset	Description
Common	Any	16	(R/W). Catch all value for any response types.
Supplier Info	NO_SUPP	17	(R/W). No Supplier Information available
	L3_HITM	18	(R/W). M-state initial lookup stat in L3.
	L3_HITE	19	(R/W). E-state
	L3_HITS	20	(R/W). S-state
	Reserved	21	Reserved
	L4_HIT_LOCAL_L4	22	(R/W). L4 Cache
	L4_HIT_REMOTE_HOP0_L4	23	(R/W). L4 Cache
	L4_HIT_REMOTE_HOP1_L4	24	(R/W). L4 Cache
	L4_HIT_REMOTE_HOP2P_L4	25	(R/W). L4 Cache
	Reserved	30:26	Reserved

18.10.4.1 Off-core Response Performance Monitoring in Intel Xeon Processors E5 v3 Series

Table 18-43 lists the supplier information field that apply to Intel Xeon processor E5 v3 series (CPUID signature 06_3FH).

Table 18-45 MSR_OFFCORE_RSP_x Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	(R/W). Catch all value for any response types.
Supplier Info	NO_SUPP	17	(R/W). No Supplier Information available
	L3_HITM	18	(R/W). M-state initial lookup stat in L3.
	L3_HITE	19	(R/W). E-state
	L3_HITS	20	(R/W). S-state
	L3_HITF	21	(R/W). F-state
	LOCAL	22	(R/W). Local DRAM Controller
	Reserved	26:23	Reserved
	L3_MISS_REMOTE_HOP0	27	(R/W). Hop 0 Remote supplier
	L3_MISS_REMOTE_HOP1	28	(R/W). Hop 1 Remote supplier
	L3_MISS_REMOTE_HOP2P	29	(R/W). Hop 2 or more Remote supplier
	Reserved	30	Reserved

...

18.10.7 Intel® Xeon® Processor E5 v3 Family Uncore Performance Monitoring Facility

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v3 families are available in "Intel® Xeon® Processor E5 v3 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 35-28.

18.11 INTEL® CORE™ M PROCESSOR PERFORMANCE MONITORING FACILITY

The Intel® Core™ M processor and the 5th Generation Intel Core processor families are based on the Broadwell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 18.2.3) and a host of non-architectural monitoring capabilities.

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

The core PMU has the same capability as those described in Section 18.10. IA32_PERF_GLOBAL_STATUS provide a bit indicator (bit 55) for PMI handler to distinguish PMI due to output buffer overflow condition due to accumulating packet data from Intel Processor Trace.

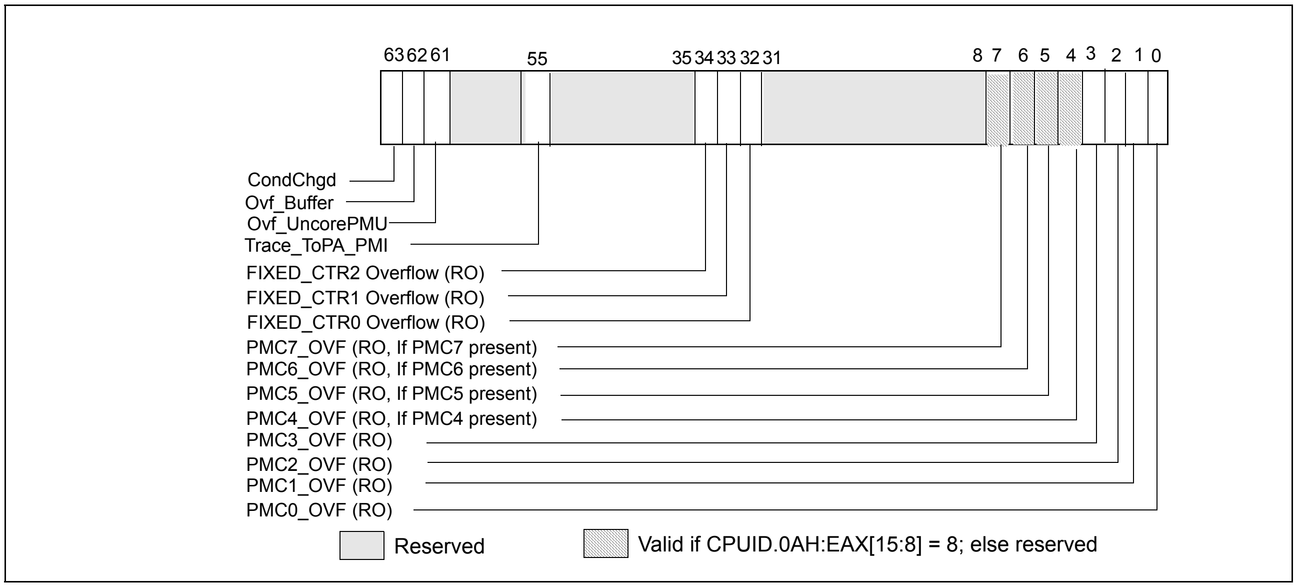


Figure 18-41 IA32_PERF_GLOBAL_STATUS MSR in Broadwell Microarchitecture

Details of Intel Processor Trace is described in Chapter 36, "Intel® Processor Trace". IA32_PERF_GLOBAL_OVF_CTRL MSR provide a corresponding reset control bit.

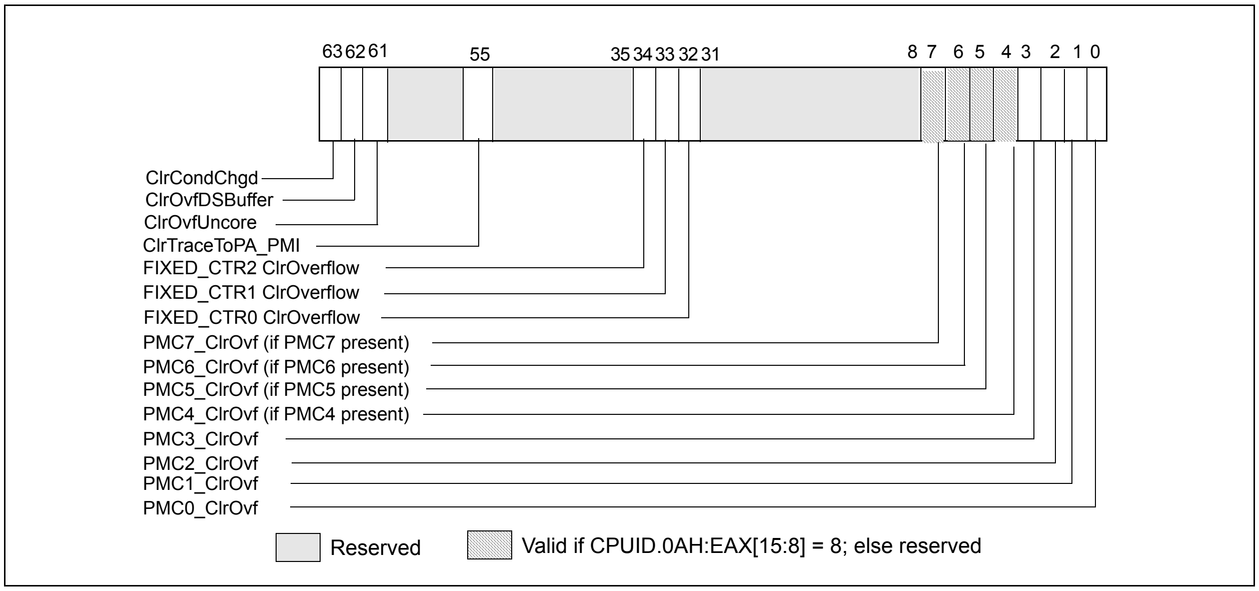


Figure 18-42 IA32_PERF_GLOBAL_OVF_CTRL MSR in Broadwell microarchitecture

The specifics of non-architectural performance events are listed in Chapter 19, "Performance Monitoring Events".

18.12 NEXT GENERATION INTEL® CORE™ PROCESSOR PERFORMANCE MONITORING FACILITY

The next generation Intel® Core™ processor is based on the Sky Lake microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 4 (see Section 18.2.4) and a host of non-architectural monitoring capabilities.

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

The core PMU's capability is similar to those described in Section 18.8 through Section 18.8.5, with some differences and enhancements summarized in Table 18-37. Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 18.10.5. For details of Intel TSX, see Chapter 15, "Programming with Intel® Transactional Synchronization Extensions" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Performance monitoring result may be affected by side-band activity on processors that support Intel SGX, details are described in Chapter 7 of the "Intel® Software Guard Extensions Programming Reference".

Table 18-48 Core PMU Comparison

Box	Intel® microarchitecture code name Sky Lake	Intel® microarchitecture code name Haswell and Broadwell	Comment
# of Fixed counters per thread	3	3	
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48 , W: 32/48	R:48 , W: 32/48	See Section 18.2.4.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	CPUID enumerates # of counters.
Architectural Perfmon version	4	3	See Section 18.2.4
PMI Overhead Mitigation	▪ Freeze_Perfmon_on_PMI with streamlined semantics. ▪ Freeze_on_LBR with streamlined semantics ▪ Freeze_while_SMM	▪ Freeze_Perfmon_on_PMI with legacy semantics. ▪ Freeze_on_LBR with legacy semantics for branch profiling ▪ Freeze_while_SMM	See Section 17.4.7. Legacy semantics not supported with version 4 or higher
Counter and Buffer Overflow Status Management	▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_STATUS_RESET ▪ Set via IA32_PERF_GLOBAL_STATUS_SET	▪ Query via IA32_PERF_GLOBAL_STATUS ▪ Reset via IA32_PERF_GLOBAL_OVF_CTRL	See Section 18.2.4.
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/ Overhead/Interference	▪ Individual counter overflow ▪ PEBS buffer overflow ▪ ToPA buffer overflow ▪ CTR_Frz, LBR_Frz, ASCI	▪ Individual counter overflow ▪ PEBS buffer overflow ▪ ToPA buffer overflow (applicable to Broadwell microarchitecture)	See Section 18.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	▪ CTR_Frz, ▪ LBR_Frz	NA	See Section 18.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	NA	See Section 18.2.4.3.

Table 18-48 Core PMU Comparison (Contd.)

Box	Intel® microarchitecture code name Sky Lake	Intel® microarchitecture code name Haswell and Broadwell	Comment
Precise Event Based Sampling (PEBS) Events	See Table 18-51	See Table 18-27	IA32_PMC4-PMC7 do not support PEBS.
LBR Record Format Encoding	000101b	000100b	Section 17.4.8.1
LBR Size	32 entries	16 entries	
LBR Entry	From_IP/To_IP/LBR_Info triplet	From_IP/To_IP pair	Section 17.9
LBR Timing	yes	no	Section 17.9.1
Call Stack Profiling	yes, see Section 17.8	yes, see Section 17.8	Use LBR facility
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types	MSR 1A6H and 1A7H; Extended request and response types	
Intel TSX support for Perfmon	See Section 18.10.5;	See Section 18.10.5;	

18.12.1 Precise Event Based Sampling (PEBS) Facility

The PEBS facility in the Next Generation Intel Core processor provides a number enhancement relative to PEBS in processors based on Haswell/Broadwell microarchitectures. The key components and differences of PEBS facility relative to Haswell/Broadwell microarchitecture is summarized in Table 18-49.

Table 18-49 PEBS Facility Comparison

Box	Intel® microarchitecture code name Sky Lake	Intel® microarchitecture code name Haswell and Broadwell	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 18.7.1.1	Section 18.7.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 18-21	Figure 18-21	
PEBS-EventingIP	yes	yes	
PEBS record format encoding	0011b	0010b	
PEBS record layout	Table 18-50, Enhanced fields at offsets 98H- B8H; and new fields at C0H	Table 18-39, Enhanced fields at offsets 98H, A0H, A8H, B0H	
Multi-counter PEBS resolution	PEBS record 90H resolves the eventing counter overflow	PEBS record 90H reflects IA32_PERF_GLOBAL_STATUS	
PEBS Events	See Table 18-51	See Table 18-27	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-PDIR	yes	yes	IA32_PMC1 only
PEBS-Load Latency	See Section 18.8.4.2;	See Section 18.8.4.2;	
PEBS-Precise Store	No, replaced by Data Address profiling	No, replaced by Data Address profiling	see Section 18.8.4.3
Data Address Profiling	yes	yes	

Table 18-49 PEBS Facility Comparison

Box	Intel® microarchitecture code name Sky Lake	Intel® microarchitecture code name Haswell and Broadwell	Comment
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.	Small SAV(CountDown) value incur higher overhead than prior generation.	

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

18.12.1.1 PEBS Data Format

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

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

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

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

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

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

With format 0010b, a snapshot of the IA32_PERF_GLOBAL_STATUS may be useful to resolve the situations when more than one of IA32_PMICx have been configured to collect PEBS data and two consecutive overflows of the PEBS-enabled counters are sufficiently far apart in time. It is also possible for the image at 90H to indicate multiple PEBS-enabled counters have overflowed. In the latter scenario, software cannot to correlate the PEBS record entry to the multiple overflowed bits.

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

18.12.1.2 PEBS Events

The list of PEBS events supported for processors based on the Sky Lake microarchitecture is shown in Table 18-51.

Table 18-51 PEBS Performance Events for the Sky Lake Microarchitecture

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST	01H
		ALL_CYCLES[1]	01H
OTHER_ASSISTS	C1H	PAGE_A_D	01H
BR_INST_RETIRED	C4H	CONDITIONAL	01H
		NEAR_CALL	02H
		ALL_BRANCHES	04H
		NEAR_RETURN	08H
		NEAR_TAKEN	20H
		FAR_BRACHES	40H
BR_MISP_RETIRED	C5H	CONDITIONAL	01H
		ALL_BRANCHES	04H
		NEAR_TAKEN	20H
HLE_RETIRED	C8H	ABORTED	04H
RTM_RETIRED	C9H	ABORTED	04H
MEM_INST_RETIRED[3]	D0H	STLB_MISS_LOADS	11H
		STLB_MISS_STORE	12H
		LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H

Table 18-51 PEBS Performance Events for the Sky Lake Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_LOAD_RETIRED[2]	D1H	L1_Hit	01H
		L2_Hit	02H
		L3_Hit	04H
		L1_Miss	08H
		L2_Miss	10H
		L3_Miss	20H
		Hit_LFB	40H
MEM_LOAD_UOPS_LLC_HIT_RETIRED[3]	D2H	XSNP_Miss	01H
		XSNP_Hit	02H
		XSNP_Hitm	04H
		XSNP_None	08H

NOTES:

1. INST_RETIRED.ALL_CYCLES is configured with additional parameters of cmask = 10 and INV = 1
2. Instruction with at least one load uop experiencing the condition specified in the UMask.

18.12.1.3 Data Address Profiling

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

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

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

18.12.2 Off-core Response Performance Monitoring

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

- Transaction request type encoding (bits 15:0): see Table 18-53.
- Supplier information (bits 30:16): see Table 18-54.
- Snoop response information (bits 37:31): see Table 18-33.

Table 18-53 MSR_OFFCORE_RSP_x Request_Type Definition (Sky Lake microarchitecture)

Bit Name	Offset	Description
DMND_DATA_RD	0	(R/W). Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count hw or sw prefetches.
DMND_RFO	1	(R/W). Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
Reserved	2	Reserved
CORE_MWB	3	Counts the number of writebacks of core modified cachelines
Reserved	6:4	Reserved
PF_L3_DATA_RD	7	(R/W). Counts the number of MLC prefetches into L3
PF_L3_RFO	8	(R/W). Counts the number of RFO requests generated by MLC prefetches to L3.
Reserved	9	Reserved
PF_L1	10	(R/W). Counts the number of software prefetches and L1 hw prefetcher
STRM_ST	11	(R/W). Counts the number of streaming store requests
CORE_NMWB	12	Counts the number of writebacks of core non-modified cachelines
Reserved	13-14	Reserved
OTHER	15	(R/W). Any other request that crosses IDI, including I/O.

Table 18-54 lists the supplier information field that apply to next generation Intel Core processors. (CPUID signature 06_4EH, 06_5EH).

Table 18-54 MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 06_4EH, 06_5EH)

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

18.13 PERFORMANCE MONITORING (PROCESSORS BASED ON INTEL NETBURST® MICROARCHITECTURE)

The performance monitoring mechanism provided in Pentium 4 and Intel Xeon processors is different from that provided in the P6 family and Pentium processors. While the general concept of selecting, filtering, counting, and reading performance events through the WRMSR, RDMSR, and RDPMC instructions is unchanged, the setup mechanism and MSR layouts are incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMC instruction has been enhanced to read the additional performance counters provided in the Pentium 4 and Intel Xeon processors and to allow faster reading of counters.

The event monitoring mechanism provided with the Pentium 4 and Intel Xeon processors (based on Intel NetBurst microarchitecture) consists of the following facilities:

- The IA32_MISC_ENABLE MSR, which indicates the availability in an Intel 64 or IA-32 processor of the performance monitoring and precise event-based sampling (PEBS) facilities.
- Event selection control (ESCR) MSRs for selecting events to be monitored with specific performance counters. The number available differs by family and model (43 to 45).
- 18 performance counter MSRs for counting events.
- 18 counter configuration control (CCCR) MSRs, with one CCCR associated with each performance counter. CCCRs sets up an associated performance counter for a specific method of counting.
- A debug store (DS) save area in memory for storing PEBS records.
- The IA32_DS_AREA MSR, which establishes the location of the DS save area.
- The debug store (DS) feature flag (bit 21) returned by the CPUID instruction, which indicates the availability of the DS mechanism.
- The MSR_PEBS_ENABLE MSR, which enables the PEBS facilities and replay tagging used in at-retirement event counting.
- A set of predefined events and event metrics that simplify the setting up of the performance counters to count specific events.

Table 18-57 lists the performance counters and their associated CCCRs, along with the ESCRs that select events to be counted for each performance counter. Predefined event metrics and events are listed in Chapter 19, “Performance-Monitoring Events.”

...

18.15.5 Cycle Counting and Opportunistic Processor Operation

As a result of the state transitions due to opportunistic processor performance operation (see Chapter 14, “Power and Thermal Management”), a logical processor or a processor core can operate at frequency different from the Processor Base frequency.

The following items are expected to hold true irrespective of when opportunistic processor operation causes state transitions:

- The time stamp counter operates at a fixed-rate frequency of the processor.
- The IA32_MPERF counter increments at a fixed frequency irrespective of any transitions caused by opportunistic processor operation.
- The IA32_FIXED_CTR2 counter increments at the same TSC frequency irrespective of any transitions caused by opportunistic processor operation.
- The Local APIC timer operation is unaffected by opportunistic processor operation.
- The TSC, IA32_MPERF, and IA32_FIXED_CTR2 operate at close to the maximum non-turbo frequency, which is equal to the product of scalable bus frequency and maximum non-turbo ratio.

For processors based on Intel Core microarchitecture, the scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] at (0CDH), see Chapter 35, "Model-Specific Registers (MSRs)". The maximum resolved bus ratio can be read from the following bit field:

- If XE operation is disabled, the maximum resolved bus ratio can be read in MSR_PLATFORM_ID[12:8]. It corresponds to the Processor Base frequency.
- IF XE operation is enabled, the maximum resolved bus ratio is given in MSR_PERF_STAT[44:40], it corresponds to the maximum XE operation frequency configured by BIOS.

XE operation of an Intel 64 processor is implementation specific. XE operation can be enabled only by BIOS. If MSR_PERF_STAT[31] is set, XE operation is enabled. The MSR_PERF_STAT[31] field is read-only.

18.16 IA32_PERF_CAPABILITIES MSR ENUMERATION

The layout of IA32_PERF_CAPABILITIES MSR is shown in Figure 18-49, it provides enumeration of a variety of interfaces:

- IA32_PERF_CAPABILITIES.LBR_FMT[bits 5:0]: encodes the LBR format, details are described in Section 17.4.8.1.
- IA32_PERF_CAPABILITIES.PEBSTrap[6]: Trap/Fault-like indicator of PEBS recording assist, see Section 18.4.4.2.
- IA32_PERF_CAPABILITIES.PEBSArchRegs[7]: Indicator of PEBS assist save architectural registers, see Section 18.4.4.2.
- IA32_PERF_CAPABILITIES.PEBS_FMT[bits 11:8]: Specifies the encoding of the layout of PEBS records, see Section 18.4.4.2.
- IA32_PERF_CAPABILITIES.SMM_FRZ[12]: Indicates IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if 1, see Section 18.16.1.
- IA32_PERF_CAPABILITIES.FULL_WRITE[13]: Indicates the processor supports IA32_A_PMCx interface for updating bits 32 and above of IA32_PMCx, see Section 18.2.5.

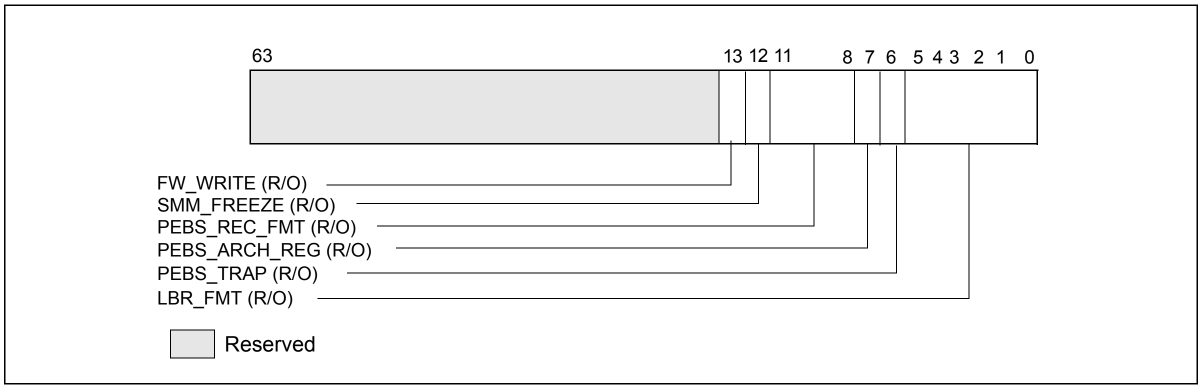


Figure 18-49 Layout of IA32_PERF_CAPABILITIES MSR

18.16.1 Filtering of SMM Handler Overhead

When performance monitoring facilities and/or branch profiling facilities (see Section 17.5, "Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel® Atom™ Processor Family)") are enabled, these facilities capture event counts, branch records and branch trace messages occurring in a logical processor. The occurrence of interrupts, instruction streams due to various interrupt handlers all contribute to the results recorded by these facilities.

If CPUID.01H:ECX.PDCM[bit 15] is 1, the processor supports the IA32_PERF_CAPABILITIES MSR. If IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is 1, the processor supports the ability for system software using performance monitoring and/or branch profiling facilities to filter out the effects of servicing system management interrupts.

If the FREEZE_WHILE_SMM capability is enabled on a logical processor and after an SMI is delivered, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler.

The enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its servicing.

It is the responsibility of the SMM code to ensure the state of the performance monitoring and branch profiling facilities are preserved upon entry or until prior to exiting the SMM. If any of this state is modified due to actions by the SMM code, the SMM code is required to restore such state to the values present at entry to the SMM handler.

System software is allowed to set IA32_DEBUGCTL.FREEZE_WHILE_SMM_EN[bit 14] to 1 only supported as indicated by IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] reporting 1.

...

22. Updates to Chapter 19, Volume 3B

Change bars show changes to Chapter 19 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

This chapter lists the performance-monitoring events that can be monitored with the Intel 64 or IA-32 processors. The ability to monitor performance events and the events that can be monitored in these processors are mostly model-specific, except for architectural performance events, described in Section 19.1.

Non-architectural performance events (i.e. model-specific events) are listed for each generation of microarchitecture:

- Section 19.2 - Processors based on Broadwell microarchitecture
- Section 19.3 - Processors based on Haswell microarchitecture
- Section 19.3.1 - Processors based on Haswell-E microarchitecture
- Section 19.4 - Processors based on Ivy Bridge microarchitecture
- Section 19.4.1 - Processors based on Ivy Bridge-E microarchitecture
- Section 19.5 - Processors based on Sandy Bridge microarchitecture
- Section 19.6 - Processors based on Intel® microarchitecture code name Nehalem
- Section 19.7 - Processors based on Intel® microarchitecture code name Westmere
- Section 19.8 - Processors based on Enhanced Intel® Core™ microarchitecture
- Section 19.9 - Processors based on Intel® Core™ microarchitecture
- Section 19.10 - Processors based on the Silvermont microarchitecture
- Section 19.11 - Processors based on Intel® Atom™ microarchitecture
- Section 19.12 - Intel® Core™ Solo and Intel® Core™ Duo processors
- Section 19.13 - Processors based on Intel NetBurst® microarchitecture
- Section 19.14 - Pentium® M family processors

- Section 19.15 - P6 family processors
- Section 19.16 - Pentium® processors

NOTE

These performance-monitoring events are intended to be used as guides for performance tuning. The counter values reported by the performance-monitoring events are approximate and believed to be useful as relative guides for tuning software. Known discrepancies are documented where applicable.

All performance event encodings not documented in the appropriate tables for the given processor are considered reserved, and their use will result in undefined counter updates with associated overflow actions.

The event tables listed this chapter provide information for tool developers to support architectural and non-architectural performance monitoring events. The tables are up to date at processor launch, but are subject to changes. The most up to date event tables and additional details of performance event implementation for end-user (including additional details beyond event code/umask) can found at the "perfmon" repository provided by The Intel Open Source Technology Center (https://download.01.org/perfmon/).

...

19.2 PERFORMANCE MONITORING EVENTS FOR THE INTEL® CORE™ M PROCESSORS

The Intel® Core™ M processors and the 5th generation Intel Core processors are based on the Broadwell microarchitecture. They support the architectural performance-monitoring events listed in Table 19-1. Non-architectural performance-monitoring events in the processor core are listed in Table 19-3. The events in Table 19-3 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_3DH and 06_47H. Table 19-4 lists performance events supporting Intel TSX (see Section 18.10.5) and are available on processor based Broadwell microarchitecture.

Non-architectural performance monitoring events that are located in the uncore sub-system are implementation specific between different platforms using processors based on Broadwell microarchitecture and with different DisplayFamily_DisplayModel signatures. Processors with CPUID signature of DisplayFamily_DisplayModel 06_3DH and 06_47H support uncore performance events listed in Table 19-5.

...

19.3 PERFORMANCE MONITORING EVENTS FOR THE 4TH GENERATION INTEL® CORE™ PROCESSORS

4th generation Intel® Core™ processors and Intel Xeon processor E3-1200 v3 product family are based on the Haswell microarchitecture. They support the architectural performance-monitoring events listed in Table 19-1. Non-architectural performance-monitoring events in the processor core are listed in Table 19-3. The events in Table 19-3 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_3CH, 06_45H and 06_46H. Table 19-4 lists performance events focused on supporting Intel TSX (see Section 18.10.5).

Additional information 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 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LD_BLOCKS.STORE_FORWARD	loads blocked by overlapping with store buffer that cannot be forwarded .	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	
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.	
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_4K	Completed page walks due to demand load misses that caused 4K page walks in any TLB levels.	
08H	04H	DTLB_LOAD_MISSES.WALK_COMPLETED_2M_4M	Completed page walks due to demand load misses that caused 2M/4M page walks in any TLB levels.	
08H	0EH	DTLB_LOAD_MISSES.WALK_COMPLETED	Completed page walks in any TLB of any page size due to demand load misses	
08H	10H	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
08H	20H	DTLB_LOAD_MISSES.STLB_HIT_4K	Load misses that missed DTLB but hit STLB (4K).	
08H	40H	DTLB_LOAD_MISSES.STLB_HIT_2M	Load misses that missed DTLB but hit STLB (2M).	
08H	60H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
08H	80H	DTLB_LOAD_MISSES.PDE_CACHE_MISS	DTLB demand load misses with low part of linear-to-physical address translation missed	
0DH	03H	INT_MISC.RECOVERY_CYCLES	Cycles waiting to recover after Machine Clears except JEClear. Set Cmask= 1.	Set Edge to count occurrences
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
0EH	10H	UOPS_ISSUED.FLAGS_MERGE	Number of flags-merge uops allocated. Such uops adds delay.	
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA or similar uops allocated. Such uop has 3 sources (e.g. 2 sources + immediate) regardless if as a result of LEA instruction or not.	
0EH	40H	UOPS_ISSUED.SiNGLE_MUL	Number of multiply packed/scalar single precision uops allocated.	

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
24H	21H	L2_RQSTS.DEMAND_DATA_RD_MISS	Demand Data Read requests that missed L2, no rejects.	
24H	41H	L2_RQSTS.DEMAND_DATA_RD_HIT	Demand Data Read requests that hit L2 cache.	
24H	E1H	L2_RQSTS.ALL_DEMAND_DATA_RD	Counts any demand and L1 HW prefetch data load requests to L2.	
24H	42H	L2_RQSTS.RFO_HIT	Counts the number of store RFO requests that hit the L2 cache.	
24H	22H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache.	
24H	E2H	L2_RQSTS.ALL_RFO	Counts all L2 store RFO requests.	
24H	44H	L2_RQSTS.CODE_RD_HIT	Number of instruction fetches that hit the L2 cache.	
24H	24H	L2_RQSTS.CODE_RD_MISS	Number of instruction fetches that missed the L2 cache.	
24H	27H	L2_RQSTS.ALL_DEMAND_MISS	Demand requests that miss L2 cache.	
24H	E7H	L2_RQSTS.ALL_DEMAND_REFERENCES	Demand requests to L2 cache.	
24H	E4H	L2_RQSTS.ALL_CODE_RD	Counts all L2 code requests.	
24H	50H	L2_RQSTS.L2_PF_HIT	Counts all L2 HW prefetcher requests that hit L2.	
24H	30H	L2_RQSTS.L2_PF_MISS	Counts all L2 HW prefetcher requests that missed L2.	
24H	F8H	L2_RQSTS.ALL_PF	Counts all L2 HW prefetcher requests.	
24H	3FH	L2_RQSTS.MISS	All requests that missed L2.	
24H	FFH	L2_RQSTS.REFERENCES	All requests to L2 cache.	
27H	50H	L2_DEMAND_RQSTS.WB_HIT	Not rejected writebacks that hit 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
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.	Counter 2 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).	

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
49H	02H	DTLB_STORE_MISSES.WALK_COMPLETED_4K	Completed page walks due to store misses in one or more TLB levels of 4K page structure.	
49H	04H	DTLB_STORE_MISSES.WALK_COMPLETED_2M_4M	Completed page walks due to store misses in one or more TLB levels of 2M/4M page structure.	
49H	0EH	DTLB_STORE_MISSES.WALK_COMPLETED	Completed page walks due to store miss in any TLB levels of any page size (4K/2M/4M/1G).	
49H	10H	DTLB_STORE_MISSES.WALK_DURATION	Cycles PMH is busy with this walk.	
49H	20H	DTLB_STORE_MISSES.STLB_HIT_4K	Store misses that missed DTLB but hit STLB (4K).	
49H	40H	DTLB_STORE_MISSES.STLB_HIT_2M	Store misses that missed DTLB but hit STLB (2M).	
49H	60H	DTLB_STORE_MISSES.STLB_HIT	Store operations that miss the first TLB level but hit the second and do not cause page walks.	
49H	80H	DTLB_STORE_MISSES.PDE_CACHE_MISS	DTLB store misses with low part of linear-to-physical address translation missed.	
4CH	01H	LOAD_HIT_PRE.SW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.	
4CH	02H	LOAD_HIT_PRE.HW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
58H	04H	MOVE_ELIMINATION.INT_NOT_ELIMINATED	Number of integer Move Elimination candidate uops that were not eliminated.	
58H	08H	MOVE_ELIMINATION.SIMD_NOT_ELIMINATED	Number of SIMD Move Elimination candidate uops that were not eliminated.	
58H	01H	MOVE_ELIMINATION.INT_ELIMINATED	Number of integer Move Elimination candidate uops that were eliminated.	
58H	02H	MOVE_ELIMINATION.SIMD_ELIMINATED	Number of SIMD Move Elimination candidate uops that were eliminated.	
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.	Use only when HTT is off
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Offcore outstanding Demand code Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	Use only when HTT is off
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.	Use only when HTT is off

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Offcore outstanding cacheable data read transactions in SQ to uncore. Set Cmask=1 to count cycles.	Use only when HTT is off
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. Add Edge=1 to count # of delivery.	Can combine Umask 04H, 08H
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS_busy by MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H
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
79H	18H	IDQ.ALL_DSB_CYCLES_ANY_UOPS	Counts cycles DSB is delivered at least one uops. Set Cmask = 1.	
79H	18H	IDQ.ALL_DSB_CYCLES_4_UOPS	Counts cycles DSB is delivered four uops. Set Cmask = 4.	
79H	24H	IDQ.ALL_MITE_CYCLES_ANY_UOPS	Counts cycles MITE is delivered at least one uops. Set Cmask = 1.	
79H	24H	IDQ.ALL_MITE_CYCLES_4_UOPS	Counts cycles MITE is delivered four uops. Set Cmask = 4.	
79H	3CH	IDQ.MITE_ALL_UOPS	# of uops delivered to IDQ from any path.	
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 ITLB that causes a page walk of any page size.	
85H	02H	ITLB_MISSES.WALK_COMPLETED_4K	Completed page walks due to misses in ITLB 4K page entries.	
85H	04H	ITLB_MISSES.WALK_COMPLETED_2M_4M	Completed page walks due to misses in ITLB 2M/4M page entries.	
85H	0EH	ITLB_MISSES.WALK_COMPLETED	Completed page walks in ITLB of any page size.	

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
85H	10H	ITLB_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
85H	20H	ITLB_MISSES.STLB_HIT_4K	ITLB misses that hit STLB (4K).	
85H	40H	ITLB_MISSES.STLB_HIT_2M	ITLB misses that hit STLB (2M).	
85H	60H	ITLB_MISSES.STLB_HIT	ITLB misses that hit STLB. 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	01H	BR_INST_EXEC.COND	Qualify conditional near branch instructions executed, but not necessarily retired.	Must combine with umask 40H, 80H
88H	02H	BR_INST_EXEC.DIRECT_JMP	Qualify all unconditional near branch instructions excluding calls and indirect branches.	Must combine with umask 80H
88H	04H	BR_INST_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify executed indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
88H	08H	BR_INST_EXEC.RETURN_NEAR	Qualify indirect near branches that have a return mnemonic.	Must combine with umask 80H
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Qualify unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Qualify indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
88H	40H	BR_INST_EXEC.NONTAKEN	Qualify non-taken near branches executed.	Applicable to umask 01H only
88H	80H	BR_INST_EXEC.TAKEN	Qualify taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	
89H	01H	BR_MISP_EXEC.COND	Qualify conditional near branch instructions mispredicted.	Must combine with umask 40H, 80H
89H	04H	BR_MISP_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify mispredicted indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Qualify mispredicted indirect near branches that have a return mnemonic.	Must combine with umask 80H
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Qualify mispredicted unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Qualify mispredicted indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
89H	40H	BR_MISP_EXEC.NONTAKEN	Qualify mispredicted non-taken near branches executed.	Applicable to umask 01H only
89H	80H	BR_MISP_EXEC.TAKEN	Qualify mispredicted taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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_EXECUTED_PORT.PORT_0	Cycles which a Uop is dispatched on port 0 in this thread.	Set AnyThread to count per core
A1H	02H	UOPS_EXECUTED_PORT.PORT_1	Cycles which a Uop is dispatched on port 1 in this thread.	Set AnyThread to count per core
A1H	04H	UOPS_EXECUTED_PORT.PORT_2	Cycles which a uop is dispatched on port 2 in this thread.	Set AnyThread to count per core
A1H	08H	UOPS_EXECUTED_PORT.PORT_3	Cycles which a uop is dispatched on port 3 in this thread.	Set AnyThread to count per core
A1H	10H	UOPS_EXECUTED_PORT.PORT_4	Cycles which a uop is dispatched on port 4 in this thread.	Set AnyThread to count per core
A1H	20H	UOPS_EXECUTED_PORT.PORT_5	Cycles which a uop is dispatched on port 5 in this thread.	Set AnyThread to count per core
A1H	40H	UOPS_EXECUTED_PORT.PORT_6	Cycles which a Uop is dispatched on port 6 in this thread.	Set AnyThread to count per core
A1H	80H	UOPS_EXECUTED_PORT.PORT_7	Cycles which a Uop is dispatched on port 7 in this thread	Set AnyThread to count per core
A2H	01H	RESOURCE_STALLS.ANY	Cycles Allocation is stalled due to Resource Related reason.	
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.	
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_PENDING	Cycles with pending L2 miss loads. Set Cmask=2 to count cycle.	Use only when HTT is off
A3H	02H	CYCLE_ACTIVITY.CYCLES_LDM_PENDING	Cycles with pending memory loads. Set Cmask=2 to count cycle.	
A3H	05H	CYCLE_ACTIVITY.STALLS_L2_PENDING	Number of loads missed L2.	Use only when HTT is off
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 data cache miss loads. Set Cmask=8 to count cycle.	PMC2 only
A3H	0CH	CYCLE_ACTIVITY.STALLS_L1D_PENDING	Execution stalls due to L1 data cache miss loads. Set Cmask=0CH.	PMC2 only
A8H	01H	LSD.UOPS	Number of Uops delivered by the LSD.	
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.	Use only when HTT is off
B0H	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Demand code read requests sent to uncore.	Use only when HTT is off

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B0H	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM.	Use only when HTT is off
B0H	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	Use only when HTT is off
B1H	02H	UOPS_EXECUTED.CORE	Counts total number of uops to be executed per-core each cycle.	Do not need to set ANY
B7H	01H	OFF_CORE_RESPONSE_0	see Table 18-43 or Table 18-44.	Requires MSR 01A6H
BBH	01H	OFF_CORE_RESPONSE_1	See Table 18-43 or Table 18-44.	Requires MSR 01A7H
BCH	11H	PAGE_WALKER_LOADS.DTLB_L1	Number of DTLB page walker loads that hit in the L1+FB.	
BCH	21H	PAGE_WALKER_LOADS.ITLB_L1	Number of ITLB page walker loads that hit in the L1+FB.	
BCH	12H	PAGE_WALKER_LOADS.DTLB_L2	Number of DTLB page walker loads that hit in the L2.	
BCH	22H	PAGE_WALKER_LOADS.ITLB_L2	Number of ITLB page walker loads that hit in the L2.	
BCH	14H	PAGE_WALKER_LOADS.DTLB_L3	Number of DTLB page walker loads that hit in the L3.	
BCH	24H	PAGE_WALKER_LOADS.ITLB_L3	Number of ITLB page walker loads that hit in the L3.	
BCH	18H	PAGE_WALKER_LOADS.DTLB_MEMORY	Number of DTLB page walker loads from memory.	
BCH	28H	PAGE_WALKER_LOADS.ITLB_MEMORY	Number of ITLB page walker loads from memory.	
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.	
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;
C1H	08H	OTHER_ASSISTS.AVX_TO_SSE	Number of transitions from AVX-256 to legacy SSE when penalty applicable.	
C1H	10H	OTHER_ASSISTS.SSE_TO_AVX	Number of transitions from SSE to AVX-256 when penalty applicable.	
C1H	40H	OTHER_ASSISTS.ANY_WB_ASSIST	Number of microcode assists invoked by HW upon uop writeback.	
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 and DataLA, use Any=1 for core granular.
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	Number of self-modifying-code machine clears detected.	

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	Supports PEBS
C5H	20H	BR_MISP_RETIRED.NEAR_TAKEN	Number of near branch instructions retired that were taken but mispredicted.	
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 FP assists due to Output values.	
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 FP assists due to input values.	
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.	Specify threshold in MSR 3F6H
D0H	01H	MEM_UOPS_RETIRED.LOADS	Qualify retired memory uops that are loads. Combine with umask 10H, 20H, 40H, 80H.	Supports PEBS and DataLA
D0H	10H	MEM_UOPS_RETIRED.STLB_MISS	Qualify retired memory uops with STLB miss. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS and DataLA
D0H	40H	MEM_UOPS_RETIRED.SPLIT	Qualify retired memory uops with line split. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS and DataLA

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D0H	80H	MEM_UOPS_RETIRED.ALL	Qualify any retired memory uops. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS and DataLA
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS and DataLA
D1H	02H	MEM_LOAD_UOPS_RETIRED.L2_HIT	Retired load uops with L2 cache hits as data sources.	Supports PEBS and DataLA
D1H	04H	MEM_LOAD_UOPS_RETIRED.L3_HIT	Retired load uops with L3 cache hits as data sources.	Supports PEBS and DataLA
D1H	08H	MEM_LOAD_UOPS_RETIRED.L1_MISS	Retired load uops missed L1 cache as data sources.	Supports PEBS and DataLA
D1H	10H	MEM_LOAD_UOPS_RETIRED.L2_MISS	Retired load uops missed L2. Unknown data source excluded.	Supports PEBS and DataLA
D1H	20H	MEM_LOAD_UOPS_RETIRED.L3_MISS	Retired load uops missed L3. Excludes unknown data source .	Supports PEBS and DataLA
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 and DataLA
D2H	01H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS	Retired load uops which data sources were L3 hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS and DataLA
D2H	02H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	Retired load uops which data sources were L3 and cross-core snoop hits in on-pkg core cache.	Supports PEBS and DataLA
D2H	04H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM	Retired load uops which data sources were HitM responses from shared L3.	Supports PEBS and DataLA
D2H	08H	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_NONE	Retired load uops which data sources were hits in L3 without snoops required.	Supports PEBS and DataLA
D3H	01H	MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCAL_DRAM	Retired load uops which data sources missed L3 but serviced from local dram.	Supports PEBS and DataLA.
E6H	1FH	BACLEARS.ANY	Number of front end re-steers due to BPU misprediction.	
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	Any MLC or L3 HW prefetch accessing L2, 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.

Table 19-3 Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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	05H	L2_LINES_OUT.DEMAND_CLEAN	Clean L2 cache lines evicted by demand.	
F2H	06H	L2_LINES_OUT.DEMAND_DIRTY	Dirty L2 cache lines evicted by demand.	

19.3.1 Performance Monitoring Events in the Processor Core of Intel Xeon Processor E5 v3 Family

Non-architectural performance monitoring events in the processor core that are applicable only to Intel Xeon processor E5 v3 family based on the Haswell-E microarchitecture, with CPUID signature of DisplayFamily_DisplayModel 06_3FH, are listed in Table 19-8. The performance events listed in Table 19-3 and Table 19-4 also apply Intel Xeon processor E5 v3 family, except that the OFF_CORE_RESPONSE_x event listed in Table 19-3 should reference Table 18-45.

Uncore performance monitoring events for Intel Xeon Processor E5 v3 families are described in “Intel® Xeon® Processor E5 v3 Uncore Performance Monitoring Programming Reference Manual”.

Table 19-6 Non-Architectural Performance Events Applicable only to the Processor Core of Intel® Xeon® Processor E5 v3 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D3H	04H	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_DRAM	Retired load uops whose data sources was remote DRAM (snoop not needed, Snoop Miss).	Supports PEBS
D3H	10H	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_HITM	Retired load uops whose data sources was remote cache HITM.	Supports PEBS
D3H	20H	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_FWD	Retired load uops whose data sources was forwards from a remote cache.	Supports PEBS

...

19.4 PERFORMANCE MONITORING EVENTS FOR 3RD GENERATION INTEL® CORE™ PROCESSORS

3rd generation Intel® Core™ processors and Intel Xeon processor E3-1200 v2 product family are based on Intel microarchitecture code name Ivy 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 19-7. The events in Table 19-7 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_3AH.

Additional information 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 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LD_BLOCKS.STORE_FORWARD	loads blocked by overlapping with store buffer that cannot be forwarded .	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are temporarily blocked because all resources for handling the split accesses are in use.	
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.	
08H	81H	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Misses in all TLB levels that cause a page walk of any page size from demand loads.	
08H	82H	DTLB_LOAD_MISSES.WALK_COMPLETED	Misses in all TLB levels that caused page walk completed of any size by demand loads.	
08H	84H	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk due to demand loads.	
08H	88H	DTLB_LOAD_MISSES.LARGE_PAGE_WALK_DURATION	Page walk for a large page completed for Demand load	
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
0EH	10H	UOPS_ISSUED.FLAGS_MERGE	Number of flags-merge uops allocated. Such uops adds delay.	
0EH	20H	UOPS_ISSUED.SLOW_LEA	Number of slow LEA or similar uops allocated. Such uop has 3 sources (e.g. 2 sources + immediate) regardless if as a result of LEA instruction or not.	
0EH	40H	UOPS_ISSUED.SiNGLE_MUL	Number of multiply packed/scalar single precision uops allocated.	
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* or AVX-128 double precision FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE	Counts number of SSE* or AVX-128 single precision FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_PACKED SINGLE	Counts number of SSE* or AVX-128 single precision FP packed uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE	Counts number of SSE* or AVX-128 double precision FP scalar uops executed.	

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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.	
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.	
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	Counts all L2 HW prefetcher requests that hit L2.	
24H	80H	L2_RQSTS.PF_MISS	Counts all L2 HW prefetcher requests that missed L2.	
24H	C0H	L2_RQSTS.ALL_PF	Counts all L2 HW prefetcher requests.	
27H	01H	L2_STORE_LOCK_RQSTS.MISS	RFOs that miss cache lines	
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 that missed LLC.	
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 lines in any state.	
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
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache.	see Table 19-1

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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	Non-SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.	
4CH	02H	LOAD_HIT_PRE.HW_PF	Non-SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
58H	04H	MOVE_ELIMINATION.INT_NOT_ELIMINATED	Number of integer Move Elimination candidate uops that were not eliminated.	
58H	08H	MOVE_ELIMINATION.SIMD_NOT_ELIMINATED	Number of SIMD Move Elimination candidate uops that were not eliminated.	
58H	01H	MOVE_ELIMINATION.INT_ELIMINATED	Number of integer Move Elimination candidate uops that were eliminated.	
58H	02H	MOVE_ELIMINATION.SIMD_ELIMINATED	Number of SIMD Move Elimination candidate uops that were eliminated.	
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.	
5FH	04H	DTLB_LOAD_MISSES.STLB_HIT	Counts load operations that missed 1st level DTLB but hit the 2nd level.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Offcore outstanding Demand Data Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
60H	02H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD	Offcore outstanding Demand Code 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. Add Edge=1 to count # of delivery.	Can combine Umask 04H, 08H
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS_busy by MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H
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
79H	18H	IDQ.ALL_DSB_CYCLES_ANY_UOPS	Counts cycles DSB is delivered at least one uops. Set Cmask = 1.	
79H	18H	IDQ.ALL_DSB_CYCLES_4_UOPS	Counts cycles DSB is delivered four uops. Set Cmask = 4.	
79H	24H	IDQ.ALL_MITE_CYCLES_ANY_UOPS	Counts cycles MITE is delivered at least one uops. Set Cmask = 1.	
79H	24H	IDQ.ALL_MITE_CYCLES_4_UOPS	Counts cycles MITE is delivered four uops. Set Cmask = 4.	
79H	3CH	IDQ.MITE_ALL_UOPS	# of uops delivered to IDQ from any path.	
80H	04H	ICACHE.IFETCH_STALL	Cycles where a code-fetch stalled due to L1 instruction-cache miss or an iTLB miss	
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	

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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	01H	BR_INST_EXEC.COND	Qualify conditional near branch instructions executed, but not necessarily retired.	Must combine with umask 40H, 80H
88H	02H	BR_INST_EXEC.DIRECT_JMP	Qualify all unconditional near branch instructions excluding calls and indirect branches.	Must combine with umask 80H
88H	04H	BR_INST_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify executed indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
88H	08H	BR_INST_EXEC.RETURN_NEAR	Qualify indirect near branches that have a return mnemonic.	Must combine with umask 80H
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Qualify unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Qualify indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
88H	40H	BR_INST_EXEC.NONTAKEN	Qualify non-taken near branches executed.	Applicable to umask 01H only
88H	80H	BR_INST_EXEC.TAKEN	Qualify taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	
89H	01H	BR_MISP_EXEC.COND	Qualify conditional near branch instructions mispredicted.	Must combine with umask 40H, 80H
89H	04H	BR_MISP_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify mispredicted indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Qualify mispredicted indirect near branches that have a return mnemonic.	Must combine with umask 80H
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Qualify mispredicted unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Qualify mispredicted indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
89H	40H	BR_MISP_EXEC.NONTAKEN	Qualify mispredicted non-taken near branches executed.	Applicable to umask 01H only
89H	80H	BR_MISP_EXEC.TAKEN	Qualify mispredicted taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H.	
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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	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.	
A3H	01H	CYCLE_ACTIVITY.CYCLES_L2_PENDING	Cycles with pending L2 miss loads. Set AnyThread to count per core.	
A3H	02H	CYCLE_ACTIVITY.CYCLES_LDM_PENDING	Cycles with pending memory loads. Set AnyThread to count per core.	Restricted to counters 0-3 when HTT is disabled.
A3H	04H	CYCLE_ACTIVITY.CYCLES_NO_EXECUTE	Cycles of dispatch stalls. Set AnyThread to count per core.	Restricted to counters 0-3 when HTT is disabled.
A3H	05H	CYCLE_ACTIVITY.STALLS_L2_PENDING	Number of loads missed L2.	Restricted to counters 0-3 when HTT is disabled.
A3H	06H	CYCLE_ACTIVITY.STALLS_LDM_PENDING		Restricted to counters 0-3 when HTT is disabled.
A3H	08H	CYCLE_ACTIVITY.CYCLES_L1D_PENDING	Cycles with pending L1 cache miss loads. Set AnyThread to count per core.	PMC2 only
A3H	0CH	CYCLE_ACTIVITY.STALLS_L1D_PENDING	Execution stalls due to L1 data cache miss loads. Set Cmask=0CH.	PMC2 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	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.	

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B0H	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.	
B0H	02H	OFFCORE_REQUESTS.DEMAND_CODE_RD	Demand code read requests sent to uncore.	
B0H	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore, including regular RFOs, locks, ItoM	
B0H	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	
B1H	01H	UOPS_EXECUTED.THREAD	Counts total number of uops to be executed per-thread each cycle. Set Cmask = 1, INV =1 to count stall cycles.	
B1H	02H	UOPS_EXECUTED.CORE	Counts total number of uops to be executed per-core each cycle.	Do not need to set ANY
B7H	01H	OFFCORE_RESPONSE_0	See Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A6H
BBH	01H	OFFCORE_RESPONSE_1	See Section 18.8.5, "Off-core Response Performance Monitoring".	Requires MSR 01A7H
BDH	01H	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries.	
BDH	20H	TLB_FLUSH.STLB_ANY	Count number of STLB flush attempts.	
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
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.	
C1H	80H	OTHER_ASSISTS.WB	Number of times microcode assist is invoked by hardware upon uop writeback	
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, use Any=1 for core granular.
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	Number of self-modifying-code machine clears detected.	
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.	

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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.	Supports PEBS
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.	Supports PEBS
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	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	Supports PEBS
C5H	20H	BR_MISP_RETIRED.NEAR_TAKEN	Mispredicted taken branch instructions retired.	Supports PEBS
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 FP assists due to Output values.	Supports PEBS
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 FP assists due to input values.	Supports PEBS
CAH	08H	FP_ASSIST.SIMD_OUTPUT	Number of SIMD FP assists due to Output values.	Supports PEBS
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.	Specify threshold in MSR 3F6H. PMC 3 only.
CDH	02H	MEM_TRANS_RETIRED.PRECISE_STORE	Sample stores and collect precise store operation via PEBS record. PMC3 only.	See Section 18.8.4.3
D0H	01H	MEM_UOPS_RETIRED.LOADS	Qualify retired memory uops that are loads. Combine with umask 10H, 20H, 40H, 80H.	Supports PEBS
D0H	10H	MEM_UOPS_RETIRED.STLB_MISS	Qualify retired memory uops with STLB miss. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS
D0H	40H	MEM_UOPS_RETIRED.SPLIT	Qualify retired memory uops with line split. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS
D0H	80H	MEM_UOPS_RETIRED.ALL	Qualify any retired memory uops. Must combine with umask 01H, 02H, to produce counts.	Supports PEBS
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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 whose data source was LLC hit with no snoop required.	Supports PEBS
D1H	08H	MEM_LOAD_UOPS_RETIRED.L1_MISS	Retired load uops whose data source followed an L1 miss	Supports PEBS
D1H	10H	MEM_LOAD_UOPS_RETIRED.L2_MISS	Retired load uops that missed L2, excluding unknown sources	Supports PEBS
D1H	20H	MEM_LOAD_UOPS_RETIRED.LLC_MISS	Retired load uops whose data source is LLC miss	Supports PEBS. Restricted to counters 0-3 when HTT is disabled.
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
D3H	01H	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM	Retired load uops whose data source was local memory (cross-socket snoop not needed or missed).	Supports PEBS.
E6H	1FH	BACLEARS.ANY	Number of front end re-steers due to BPU misprediction.	
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	Any MLC or LLC HW prefetch accessing L2, 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.

Table 19-7 Non-Architectural Performance Events In the Processor Core of 3rd Generation Intel® Core™ i7, i5, i3 Processors (Contd.)

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
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 the MLC prefetcher.	
F2H	08H	L2_LINES_OUT.PF_DIRTY	Dirty L2 cache lines evicted by the MLC prefetcher.	
F2H	0AH	L2_LINES_OUT.DIRTY_ALL	Dirty L2 cache lines filling the L2.	Counting does not cover rejects.

...

23. 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.

...

22.25.3 Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers

MMX instructions and a subset of SSE, SSE2, SSSE3 instructions operate on MMX registers. The exception conditions of these instructions are described in the following tables.

Table 22-4 Exception Conditions for Legacy SIMD/MMX Instructions with FP Exception and 16-Byte Alignment

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form

Table 22-4 Exception Conditions for Legacy SIMD/MMX Instructions with FP Exception and 16-Byte Alignment

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
General Protection, #GP(0)	X	X	X	X	Legacy SSE: Memory operand is not 16-byte aligned
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
#XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1
Applicable Instructions	CVTPD2PI, CVTTPD2PI				

Table 22-5 Exception Conditions for Legacy SIMD/MMX Instructions with XMM and FP Exception

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1
Applicable Instructions	CVTPI2PS, CVTPS2PI, CVTTPS2PI				

Table 22-6 Exception Conditions for Legacy SIMD/MMX Instructions with XMM and without FP Exception

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is ‘0’
#MF[1]	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	CVTPI2PD				

NOTES:

1. Applies to “CVTPI2PD xmm, mm” but not “CVTPI2PD xmm, m64”.

Table 22-7 Exception Conditions for SIMD/MMX Instructions with Memory Reference

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	PABSB, PABSD, PABSW, PACKSSWB, PACKSSDW, PACKUSWB, PADDB, PADDD, PADDQ, PADDW, PADDSB, PADDSW, PADDUSB, PADDUSW, PALIGNR, PAND, PANDN, PAVGB, PAVGW, PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW, PHADDD, PHADDW, PHADDSW, PHSUBD, PHSUBW, PHSUBSW, PINSRW, PMADDUBSW, PMADDWD, PMAXSW, PMAXUB, PMINSW, PMINUB, PMULHRSW, PMULHUW, PMULHW, PMULLW, PMULUDQ, PSADBW, PSHUFB, PSHUFW, PSIGNB PSIGND PSIGNW, PSLLW, PSLLD, PSLLQ, PSRAD, PSRAW, PSRLW, PSRLD, PSRLQ, PSUBB, PSUBD, PSUBQ, PSUBW, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ, PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ, PXOR				

Table 22-8 Exception Conditions for Legacy SIMD/MMX Instructions without FP Exception

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1. If ModR/M.mod ≠ 11b[1]
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
#GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the destination operand is in a non-writable segment.[2] If the DS, ES, FS, or GS register contains a NULL segment selector.[3]
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
#AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	MASKMOVQ, MOVNTQ, "MOVQ (mmreg)"				

NOTES:

1. Applies to MASKMOVQ only.
2. Applies to MASKMOVQ and MOVQ (mmreg) only.
3. Applies to MASKMOVQ only.

Table 22-9 Exception Conditions for Legacy SIMD/MMX Instructions without Memory Reference

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM			X	X	If CR0.TS[bit 3]=1
Applicable Instructions	PEXTRW, PMOVMSKB				

...

24. Updates to Chapter 25, Volume 3C

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

...

25.2 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 24.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally through the guest IDT. This use of the exception bitmap applies also to exceptions generated by the instructions INT3, INTO, BOUND, and UD2.

  Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

  Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.
- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the

case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.

- **External interrupts.** An external interrupt causes a VM exit if the "external-interrupt exiting" VM-execution control is 1. (See Section 25.6 for an exception.) Otherwise, the interrupt is delivered normally through the IDT. (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The interrupt is not delivered through the IDT and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the "NMI exiting" VM-execution control is 1. Otherwise, it is delivered using descriptor 2 of the IDT. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered through the IDT and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)
- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)
- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 25.4.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 34.15.2.[1]
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 25.5.1 for details of operation of the VMX-preemption timer.

  Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the "NMI-window exiting" VM-execution control and lower priority events.

  These VM exits wake a logical processor from the same inactive states as would a non-maskable interrupt. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

In addition, there are controls that cause VM exits based on the readiness of guest software to receive interrupts:

- If the "interrupt-window exiting" VM-execution control is 1, a VM exit occurs before execution of any instruction if RFLAGS.IF = 1 and there is no blocking of events by STI or by MOV SS (see Table 24-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.6.5).

  Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.

  These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.
- If the "NMI-window exiting" VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no blocking of events by MOV SS (see Table 24-3). (A logical processor may also prevent such a VM exit if there is blocking of events by STI.) Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.6.6).

  VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

1. Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 34.14.1.

These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

...

25. Updates to Chapter 29, Volume 3C

Change bars show changes to Chapter 29 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

...

29.4.3.2 APIC-Write Emulation

If the processor virtualizes a write access to the APIC-access page, it performs additional actions after completion of an operation of which the access was a part. These actions are called **APIC-write emulation**.

The details of APIC-write emulation depend upon the page offset of the virtualized write access:[1]

- 080H (task priority). The processor clears bytes 3:1 of VTPR and then causes TPR virtualization (Section 29.1.2).
- 0B0H (end of interrupt). If the "virtual-interrupt delivery" VM-execution control is 1, the processor clears VEOI and then causes EOI virtualization (Section 29.1.4); otherwise, the processor causes an APIC-write VM exit (Section 29.4.3.3).
- 300H (interrupt command — low). If the "virtual-interrupt delivery" VM-execution control is 1, the processor checks the value of VICR_LO to determine whether the following are all true:
  - Reserved bits (31:20, 17:16, 13) and bit 12 (delivery status) are all 0.
  - Bits 19:18 (destination shorthand) are 01B (self).
  - Bit 15 (trigger mode) is 0 (edge).
  - Bits 10:8 (delivery mode) are 000B (fixed).
  - Bits 7:4 (the upper half of the vector) are **not** 0000B.

  If all of the items above are true, the processor performs self-IPI virtualization using the 8-bit vector in byte 0 of VICR_LO (Section 29.1.5).

  If the "virtual-interrupt delivery" VM-execution control is 0, or if any of the items above are false, the processor causes an APIC-write VM exit (Section 29.4.3.3).
- 310H–313H (interrupt command — high). The processor clears bytes 2:0 of VICR_HI. No other virtualization or VM exit occurs.
- Any other page offset. The processor causes an APIC-write VM exit (Section 29.4.3.3).

APIC-write emulation takes priority over system-management interrupts (SMIs), INIT signals, and lower priority events. APIC-write emulation is not blocked if RFLAGS.IF = 0 or by the MOV SS, POP SS, or STI instructions.

If an operation causes a fault after a write access to the APIC-access page and before APIC-write emulation, and that fault is delivered without a VM exit, APIC-write emulation occurs after the fault is delivered and before the fault handler can execute. If an operation causes a VM exit (perhaps due to a fault) after a write access to the APIC-access page and before APIC-write emulation, the APIC-write emulation does not occur.

1. For any operation, there can be only one page offset for which a write access was virtualized. This is because a write access is not virtualized if the processor has already virtualized a write access for the same operation with a different page offset.

...

29.4.4 Instruction-Specific Considerations

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

- **CLFLUSH.** With regard to faulting, the processor operates as if CLFLUSH reads from the linear address in its source operand. If that address translates to one on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will flush the corresponding cache line on the virtual-APIC page instead of the APIC-access page.
- **ENTER.** With regard to faulting, the processor operates if ENTER writes to the byte referenced by the final value of the stack pointer (even though it does not if its size operand is non-zero). If that value translates to an address on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will cause the APIC-write emulation appropriate to the address's page offset.
- **MASKMOVQ and MAKSMOVDQU.** Even if the instruction's mask is zero, the processor may operate with regard to faulting as if MASKMOVQ or MASKMOVDQU writes to memory (the behavior is implementation-specific). In such a situation, an APIC-access VM exit may occur.
- **MONITOR.** With regard to faulting, the processor operates as if MONITOR reads from the effective address in RAX. If the resulting linear address translates to one on the APIC-access page, the instruction may cause an APIC-access VM exit.[1] If it does not, it will monitor the corresponding address on the virtual-APIC page instead of the APIC-access page.
- **PREFETCH.** An execution of the PREFETCH instruction that would result in an access to the APIC-access page does not cause an APIC-access VM exit. Such an access may prefetch data; if so, it is from the corresponding address on the virtual-APIC page.

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

...

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

26. 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, 06_5EH	Next Generation Intel Core Processor based on Sky Lake microarchitecture
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_47H	5th generation Intel Core processors based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-2600/1600 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

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

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_1AH	Intel Core i7 Processor, Intel Xeon processor 3400, 3500, 5500 series
06_1DH	Intel Xeon processor MP 7400 series
06_17H	Intel Xeon processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_4CH	Intel® Atom™ processor X7-Z8000 and X5-Z8000 series based on Airmont Microarchitecture
06_5DH	Intel® Atom™ processor X3-C3000 based on Silvermont Microarchitecture
06_5AH	Intel Atom processor Z3500 series
06_4AH	Intel Atom processor Z3400 series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 series
06_36H	Intel Atom processor S1000 Series
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	Intel Atom processor family, Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon processor, Intel Xeon processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon processor, Intel Pentium III processor
06_03H, 06_05H	Intel Pentium II Xeon processor, Intel Pentium II processor
06_01H	Intel Pentium Pro processor
05_01H, 05_02H, 05_04H	Intel Pentium processor, Intel Pentium processor with MMX Technology

35.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table 35-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 35-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a machine specified MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 35-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of "DF_DM" (see Table 35-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as "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 17.14, "Time-Stamp Counter."	05_01H
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	**Platform ID (RO)** The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	06_01H
		49:0	Reserved.	
		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

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	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
		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

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		19:16	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	
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	

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	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])
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.	

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			
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	06_01H
		7:0	VCNT: The number of variable memory type ranges in the processor.	
		8	Fixed range MTRRs are supported when set.	
		9	Reserved.	
		10	WC Supported when set.	
		11	SMRR Supported when set.	
		63:12	Reserved.	
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

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			
		27	MCG_LMCE_P: Indicates that the processor support extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH
		63:28	Reserved.	
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (R/W0)	06_01H
		0	RIPV. Restart IP valid	06_01H
		1	EIPV. Error IP valid	06_01H
		2	MCIP. Machine check in progress	06_01H
		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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.	
		63:32	Reserved.	
187H	391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	Performance Event Select Register 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
188H	392	IA32_PERFEVTSEL2	Performance Event Select Register 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
189H	393	IA32_PERFEVTSEL3	Performance Event Select Register 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
18AH-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

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	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	
		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

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			
		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.	
		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.	

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			
		16	**Enhanced Intel SpeedStep Technology Enable (R/W)** 0= Enhanced Intel SpeedStep Technology disabled 1 = Enhanced Intel SpeedStep Technology enabled	If CPUID.01H: ECX[7] =1
		17	Reserved.	
		18	**ENABLE MONITOR FSM (R/W)** When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/ MWAIT are not supported. Software attempts to execute MONITOR/ MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
		21:19	Reserved.	
		22	**Limit CPUID Maxval (R/W)** When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 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

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			
		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.	
		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)	

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	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)	
		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.	

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH
		7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH
		8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
		9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
		10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH
		11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H: ECX[15] = 1 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 38FH) 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.	

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	**Valid** Enable range mask.	
		31:12	**PhysMask** SMRR address range mask.	
		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

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			
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	06_01H
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	06_01H
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	06_01H
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	

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			
		55:51	Reserved.	
		58:56	PA7	
		63:59	Reserved.	
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

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			
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
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	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 1to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1to count while CPL > 0.	
		6	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
		8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	
		9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0.	
		10	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		11	EN2_PMI: Enable PMI when fixed counter 2 overflows.	
		63:12	Reserved.	
38EH	910	IA32_PERF_GLOBAL_STATUS	Global Performance Counter Status (RO)	If CPUID.0AH: EAX[7:0] > 0
		0	Ovf_PMC0: Overflow status of IA32_PMC0.	If CPUID.0AH: EAX[15:8] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH: EAX[15:8] > 1

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.	If CPUID.0AH: EAX[15:8] > 2
		3	Ovf_PMC3: Overflow status of IA32_PMC3.	If CPUID.0AH: EAX[15:8] > 3
		31:4	Reserved.	
		32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0.	If CPUID.0AH: EAX[7:0] > 1
		33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1.	If CPUID.0AH: EAX[7:0] > 1
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH: EAX[7:0] > 1
		54:35	Reserved.	
		55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer was completely filled.	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1) and IA32_RTIT_CTL.ToPA = 1
		57:56	Reserved.	
		58	LBR_Frz: LBRs are frozen due to ▪ IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1, ▪ The LBR stack overflowed	If CPUID.0AH: EAX[7:0] > 3
		59	CTR_Frz: Performance counters in the core PMU are frozen due to ▪ IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1, ▪ one or more core PMU counters overflowed.	If CPUID.0AH: EAX[7:0] > 3
		60	ASCI: Data in the performance counters in the core PMU may include contributions from the direct or indirect operation intel SGX to protect an enclave.	If CPUID.(EAX=07H, ECX=0):EBX[bit 2] = 1
		61	Ovf_Uncore: Uncore counter overflow status.	If CPUID.0AH: EAX[7:0] > 2
		62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.0AH: EAX[7:0] > 0
		63	CondChgd: status bits of this register has changed.	If CPUID.0AH: EAX[7:0] > 0
38FH	911	IA32_PERF_GLOBAL_CTRL	Global Performance Counter Control (R/W) Counter increments while the result of ANDing respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.	If CPUID.0AH: EAX[7:0] > 0

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_PMC0	If CPUID.0AH: EAX[15:8] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[15:8] > 1
		2	EN_PMC2	If CPUID.0AH: EAX[15:8] > 2
		n	EN_PMCn	If CPUID.0AH: EAX[15:8] > n
		31:n+1	Reserved.	
		32	EN_FIXED_CTR0	If CPUID.0AH: EDX[4:0] > 0
		33	EN_FIXED_CTR1	If CPUID.0AH: EDX[4:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EDX[4:0] > 2
		63:35	Reserved.	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0 && CPUID.0AH: EAX[7:0] <= 3
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved.	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EDX[4:0] > 2
		54:35	Reserved.	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1) and IA32_RTIT_CTL.ToPA = 1
		60:56	Reserved.	
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf: bit.	If CPUID.0AH: EAX[7:0] > 0
		63	Set to 1to clear CondChgd: bit.	If CPUID.0AH: EAX[7:0] > 0

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			
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Global Performance Counter Overflow Reset Control (R/W)	If CPUID.0AH: EAX[7:0] > 3
		0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH: EAX[15:8] > 0
		1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved.	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH: EDX[4:0] > 0
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH: EDX[4:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH: EDX[4:0] > 2
		54:35	Reserved.	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If (CPUID.(EAX=07H, ECX=0):EBX[bit 25] = 1) and IA32_RTIT_CTL.ToPA = 1
		57:56	Reserved.	
		58	Set 1 to Clear LBR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to Clear CTR_Frz bit.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to Clear ASCI bit.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
		62	Set 1 to Clear OvfBuf: bit.	If CPUID.0AH: EAX[7:0] > 0
		63	Set to 1to clear CondChgd: bit.	If CPUID.0AH: EAX[7:0] > 0
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Global Performance Counter Overflow Set Control (R/W)	If CPUID.0AH: EAX[7:0] > 3
		0	Set 1 to cause Ovf_PMC0 = 1.	If CPUID.0AH: EAX[7:0] > 3
		1	Set 1 to cause Ovf_PMC1 = 1	If CPUID.0AH: EAX[15:8] > 1
		2	Set 1 to cause Ovf_PMC2 = 1	If CPUID.0AH: EAX[15:8] > 2
		n	Set 1 to cause Ovf_PMCn = 1	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved.	
		32	Set 1 to cause Ovf_FIXED_CTR0 = 1.	If CPUID.0AH: EAX[7:0] > 3
		33	Set 1 to cause Ovf_FIXED_CTR1 = 1.	If CPUID.0AH: EAX[7:0] > 3

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	Set 1 to cause Ovf_FIXED_CTR2 = 1.	If CPUID.0AH: EAX[7:0] > 3
		54:35	Reserved.	
		55	Set 1 to cause Trace_ToPA_PMI = 1.	If CPUID.0AH: EAX[7:0] > 3
		57:56	Reserved.	
		58	Set 1 to cause LBR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to cause CTR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to cause ASCI = 1.	If CPUID.0AH: EAX[7:0] > 3
		61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.0AH: EAX[7:0] > 3
		62	Set 1 to cause OvfBuf = 1.	If CPUID.0AH: EAX[7:0] > 3
		63	Reserved	
392H	914	IA32_PERF_GLOBAL_INUSE	Indicator of core perfmon interface is in use (RO)	If CPUID.0AH: EAX[7:0] > 3
		0	IA32_PERFEVTSEL0 in use	
		1	IA32_PERFEVTSEL1 in use	If CPUID.0AH: EAX[15:8] > 1
		2	IA32_PERFEVTSEL2 in use	If CPUID.0AH: EAX[15:8] > 2
		n	IA32_PERFEVTSELn in use	If CPUID.0AH: EAX[15:8] > n
		31:n	Reserved.	
		32	IA32_FIXED_CTR0 in use	
		33	IA32_FIXED_CTR1 in use	
		34	IA32_FIXED_CTR2 in use	
		35-35	Reserved or Model specific.	
		63	PMI in use.	
3F1H	1009	IA32_PEBS_ENABLE	PEBS Control (R/W)	
		0	Enable PEBS on IA32_PMC0.	06_0FH
		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

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			
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
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

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			
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
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

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			
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
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

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			
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])
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])

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			
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
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)

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6:0	Reserved	
		MAXPHYADDR[3]-1:7	Base physical address 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,	
		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)

Table 35-2 IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		4:0	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
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

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			
		7:0	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
		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.	

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			
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)
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)

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			
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)
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)

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			
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)
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))

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: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**	
		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 17.16.2.1, "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.	

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			
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
		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.	

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	**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
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].

...

35.4.1 MSRs In Future Intel Atom Processors Based on Airmont Microarchitecture

Intel Atom processor X7-Z8000 and X5-Z8000 series are based on the Airmont microarchitecture. They support 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.
		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.

Table 35-10 MSRs in Intel Atom Processors Based on the Airmont Microarchitecture (Contd.)

Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - 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."
		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."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 35-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.14, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Package	**Platform ID (R)** See Table 35-2.
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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDC-TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDC/TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 133.33MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_ CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-sate support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		23:16		Reserved.
		24		**Interrupt filtering enable (R/W)** When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.
		25		**C3 state auto demotion enable (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		63:27		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_ BASE	Core	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
186H	390	IA32_PERFEVTSEL0	Thread	See Table 35-2.
		7:0		**Event Select**
		15:8		**UMask**
		16		**USR**
		17		**OS**
		18		**Edge**
		19		**PC**
		20		**INT**
		21		**AnyThread**
		22		**EN**
		23		**INV**
		31:24		**CMASK**
		63:32		Reserved.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 35-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 35-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 35-2.
198H	408	IA32_PERF_STATUS	Core	See Table 35-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
199H	409	IA32_PERF_CTL	Thread	See Table 35-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	**Clock Modulation (R/W)** See Table 35-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved.
		3:1		**On demand Clock Modulation Duty Cycle (R/W)**
		4		**On demand Clock Modulation Enable (R/W)**
		63:5		Reserved.
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 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:39		Reserved.
1A2H	418	MSR_TEMPERATURE_TARGET	Thread	
		15:0		Reserved.
		23:16		**Temperature Target (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A4H	420	MSR_MISC_FEATURE_CONTROL		**Miscellaneous Feature Control (R/W)**
		0	Core	**L2 Hardware Prefetcher Disable (R/W)** If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.
		1	Core	**L2 Adjacent Cache Line Prefetcher Disable (R/W)** If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).
		2	Core	**DCU Hardware Prefetcher Disable (R/W)** If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	**DCU IP Prefetcher Disable (R/W)** If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction Pointer of previous loads) to determine whether to prefetch additional lines.
		63:4		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	**Offcore Response Event Select Register (R/W)**
1AAH	426	MSR_MISC_PWR_MGMT		See http://biosbits.org.
		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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		14:0	Package	**TDP Limit (R/W)** TDP limit in 1/8 Watt granularity.
		15	Package	**TDP Limit Override Enable (R/W)** A value = 0 indicates override is not active, and a value = 1 indicates active.
		30:16	Package	**TDC Limit (R/W)** TDC limit in 1/8 Amp granularity.
		31	Package	**TDC Limit Override Enable (R/W)** A value = 0 indicates override is not active, and a value = 1 indicates active.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** **RO** if MSR_PLATFORM_INFO.[28] = 0, **RW** if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
1C8H	456	MSR_LBR_SELECT	Core	**Last Branch Record Filtering Select Register (R/W)** See Section 17.6.2, "Filtering of Last Branch Records."
		0		**CPL_EQ_0**
		1		**CPL_NEQ_0**
		2		**JCC**
		3		**NEAR_REL_CALL**
		4		**NEAR_IND_CALL**
		5		**NEAR_RET**
		6		**NEAR_IND_JMP**
		7		**NEAR_REL_JMP**
		8		**FAR_BRANCH**
		63:9		Reserved.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 35-2.
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.
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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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.
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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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)**
		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)

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
400H	1024	IA32_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."
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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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."
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 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (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.”
600H	1536	IA32_DS_AREA	Thread	**DS Save Area (R/W)** See Table 35-2. See Section 18.13.4, “Debug Store (DS) Mechanism.”

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_ LASTBRANCH_0_FROM_IP	Thread	**Last Branch Record 0 From IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** 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.
68DH	1677	MSR_ LASTBRANCH_13_FROM_IP	Thread	**Last Branch Record 13 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_ LASTBRANCH_14_FROM_IP	Thread	**Last Branch Record 14 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
68FH	1679	MSR_ LASTBRANCH_15_FROM_IP	Thread	**Last Branch Record 15 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_ LASTBRANCH_0_TO_IP	Thread	**Last Branch Record 0 To IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction 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.
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.

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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)
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)

Table 35-11 MSRs in Processors Based on Intel® Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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 17.14.2, "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 17.14, "Time-Stamp Counter," and see Table 35-2.
17H	23	IA32_PLATFORM_ID	Package	**Platform ID (R)** See Table 35-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 35-2.
34H	52	MSR_SMI_COUNT	Thread	**SMI Counter (R/O)**
		31:0		**SMI Count (R/O)** Count SMIs.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64 Processor (R/W)** See Table 35-2.
		0		**Lock (R/WL)**
		1		**Enable VMX inside SMX operation (R/WL)**
		2		**Enable VMX outside SMX operation (R/WL)**
		14:8		**SENTER local functions enables (R/WL)**
		15		**SENTER global functions enable (R/WL)**
79H	121	IA32_BIOS_UPDT_TRIG	Core	**BIOS Update Trigger Register (W)** See Table 35-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	**BIOS Update Signature ID (RO)** See Table 35-2.
C1H	193	IA32_PMC0	Thread	**Performance Counter Register** See Table 35-2.

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			
C2H	194	IA32_PMC1	Thread	**Performance Counter Register** See Table 35-2.
C3H	195	IA32_PMC2	Thread	**Performance Counter Register** See Table 35-2.
C4H	196	IA32_PMC3	Thread	**Performance Counter Register** See Table 35-2.
C5H	197	IA32_PMC4	Core	**Performance Counter Register (if core not shared by threads)**
C6H	198	IA32_PMC5	Core	**Performance Counter Register (if core not shared by threads)**
C7H	199	IA32_PMC6	Core	**Performance Counter Register (if core not shared by threads)**
C8H	200	IA32_PMC7	Core	**Performance Counter Register (if core not shared by threads)**
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		Reserved.
		10		**I/O MWAIT Redirection Enable (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		Reserved.
		15		**CFG Lock (R/WO)** When set, lock bits 15:0 of this register until next reset.
		24:16		Reserved.
		25		**C3 state auto demotion enable (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		**Enable C3 undemotion (R/W)** When set, enables undemotion from demoted C3.
		28		**Enable C1 undemotion (R/W)** When set, enables undemotion from demoted C1.
		63:29		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_ BASE	Core	**Power Management IO Redirection in C-state (R/W)** See http://biosbits.org.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:0		**LVL_2 Base Address (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count (RW)** See Table 35-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count (RW)** See Table 35-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 35-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 35-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 35-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 35-2.
179H	377	IA32_MCG_CAP	Thread	See Table 35-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.

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			
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.
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.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0		**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.
		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.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		0	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.
		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.

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			
		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.
1B1H	433	IA32_PACKAGE_THERM_ STATUS	Package	See Table 35-2.
1B2H	434	IA32_PACKAGE_THERM_ INTERRUPT	Package	See Table 35-2.
1C8H	456	MSR_LBR_SELECT	Thread	**Last Branch Record Filtering Select Register (R/W)** See Section 17.6.2, “Filtering of Last Branch Records.”
		0		**CPL_EQ_0**
		1		**CPL_NEQ_0**
		2		**JCC**
		3		**NEAR_REL_CALL**
		4		**NEAR_IND_CALL**
		5		**NEAR_RET**
		6		**NEAR_IND_JMP**

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			
		7		**NEAR_REL_JMP**
		8		**FAR_BRANCH**
		63:9		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control (R/W)** See Table 35-2.
		0		**LBR: Last Branch Record**
		1		**BTF**
		5:2		Reserved.
		6		**TR: Branch Trace**
		7		**BTS: Log Branch Trace Message to BTS buffer**
		8		**BTINT**
		9		**BTS_OFF_OS**
		10		**BTS_OFF_USER**
		11		**FREEZE_LBR_ON_PMI**
		12		**FREEZE_PERFMON_ON_PMI**
		13		**ENABLE_UNCORE_PMI**
		14		**FREEZE_WHILE_SMM**
		63:15		Reserved.
1DDH	477	MSR_LER_FROM_LIP	Thread	**Last Exception Record From Linear IP (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	**Last Exception Record To Linear IP (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 35-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 35-2.
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 35-2.

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			
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 35-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 35-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 35-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 35-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 35-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 35-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 35-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 35-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 35-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 35-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 35-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 35-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 35-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 35-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 35-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 35-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 35-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 35-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 35-2.
250H	592	IA32_MTRR_FIX64K_ 00000	Thread	See Table 35-2.
258H	600	IA32_MTRR_FIX16K_ 80000	Thread	See Table 35-2.
259H	601	IA32_MTRR_FIX16K_ A0000	Thread	See Table 35-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 35-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 35-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 35-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 35-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 35-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 35-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 35-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 35-2.
277H	631	IA32_PAT	Thread	See Table 35-2.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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		See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0	Thread	**Ovf_PMC0**
		1	Thread	**Ovf_PMC1**
		2	Thread	**Ovf_PMC2**
		3	Thread	**Ovf_PMC3**
		4	Core	**Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Core	**Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)**
		6	Core	**Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Core	**Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.

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	Thread	**Ovf_FixedCtr0**
		33	Thread	**Ovf_FixedCtr1**
		34	Thread	**Ovf_FixedCtr2**
		60:35		Reserved.
		61	Thread	**Ovf_Uncore**
		62	Thread	**Ovf_BufDSSAVE**
		63	Thread	**CondChgd**
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0	Thread	**Set 1 to enable PMC0 to count**
		1	Thread	**Set 1 to enable PMC1 to count**
		2	Thread	**Set 1 to enable PMC2 to count**
		3	Thread	**Set 1 to enable PMC3 to count**
		4	Core	**Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4)**
		5	Core	**Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5)**
		6	Core	**Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6)**
		7	Core	**Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Set 1 to enable FixedCtr0 to count**
		33	Thread	**Set 1 to enable FixedCtr1 to count**
		34	Thread	**Set 1 to enable FixedCtr2 to count**
		63:35		Reserved.
390H	912	IA32_PERF_GLOBAL_OVF_CTRL		See Table 35-2. See Section 18.4.2, "Global Counter Control Facilities."
		0	Thread	**Set 1 to clear Ovf_PMC0**
		1	Thread	**Set 1 to clear Ovf_PMC1**
		2	Thread	**Set 1 to clear Ovf_PMC2**
		3	Thread	**Set 1 to clear Ovf_PMC3**
		4	Core	**Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Core	**Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)**
		6	Core	**Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Core	**Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Set 1 to clear Ovf_FixedCtr0**

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			
		33	Thread	**Set 1 to clear Ovf_FixedCtr1**
		34	Thread	**Set 1 to clear Ovf_FixedCtr2**
		60:35		Reserved.
		61	Thread	**Set 1 to clear Ovf_Uncore**
		62	Thread	**Set 1 to clear Ovf_BufDSSAVE**
		63	Thread	**Set 1 to clear CondChgd**
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.7.1.1, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		62:36		Reserved.
		63		Enable Precise Store. (R/W)
3F6H	1014	MSR_PEBS_LD_LAT	Thread	see See Section 18.7.1.2, "Load Latency Performance Monitoring Facility."
		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.

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			
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
3FEH	1022	MSR_CORE_C7_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C7 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
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."

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			
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
		0		**PCU Hardware Error (R/W)** When set, enables signaling of PCU hardware detected errors.
		1		**PCU Controller Error (R/W)** When set, enables signaling of PCU controller detected errors
		2		**PCU Firmware Error (R/W)** When set, enables signaling of PCU firmware detected errors
		63:2		Reserved.
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
480H	1152	IA32_VMX_BASIC	Thread	**Reporting Register of Basic VMX Capabilities (R/O)** See Table 35-2. See Appendix A.1, “Basic VMX Information.”
481H	1153	IA32_VMX_PINBASED_ CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Controls (R/O)** See Table 35-2. See Appendix A.3, “VM-Execution Controls.”
482H	1154	IA32_VMX_PROCBASED_ CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)** See Appendix A.3, “VM-Execution Controls.”
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.”

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			
487H	1159	IA32_VMX_CR0_FIXED1	Thread	**Capability Reporting Register of CR0 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	**Capability Reporting Register of CR4 Bits Fixed to 0 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	**Capability Reporting Register of CR4 Bits Fixed to 1 (R/O)** See Table 35-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	**Capability Reporting Register of VMCS Field Enumeration (R/O)** See Table 35-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Thread	**Capability Reporting Register of Secondary Processor-based VM-execution Controls (R/O)** See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Thread	**Capability Reporting Register of EPT and VPID (R/O)** See Table 35-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Flex Controls (R/O)** See Table 35-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls (R/O)** See Table 35-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Flex Controls (R/O)** See Table 35-2
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.

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			
600H	1536	IA32_DS_AREA	Thread	**DS Save Area (R/W)** See Table 35-2. See Section 18.13.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	**Unit Multipliers used in RAPL Interfaces (R/O)** See Section 14.9.1, "RAPL Interfaces."
60AH	1546	MSR_PKGC3_IRTL	Package	**Package C3 Interrupt Response Limit (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C3 state.
		12:10		**Time Unit (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved.
		15		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60BH	1547	MSR_PKGC6_IRTL	Package	**Package C6 Interrupt Response Limit (R/W)** This MSR defines the budget allocated for the package to exit from C6 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C6 state.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		12:10		**Time Unit (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		Reserved.
		15		**Valid (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		**Package C2 Residency Counter. (R/O)** Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_POWER_LIMIT	Package	**PKG RAPL Power Limit Control (R/W)** See Section 14.9.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	**PKG Energy Status (R/O)** See Section 14.9.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameters (R/W)** See Section 14.9.3, "Package RAPL Domain."
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control (R/W)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	**PP0 Energy Status (R/O)** See Section 14.9.4, "PP0/PP1 RAPL Domains."
680H	1664	MSR_ LASTBRANCH_0_FROM_IP	Thread	**Last Branch Record 0 From IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction**. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.6.1, "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.

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			
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	**Last Branch Record 2 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	**Last Branch Record 3 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	**Last Branch Record 4 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	**Last Branch Record 5 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	**Last Branch Record 6 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	**Last Branch Record 7 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	**Last Branch Record 8 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	**Last Branch Record 9 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	**Last Branch Record 10 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	**Last Branch Record 11 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	**Last Branch Record 12 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	**Last Branch Record 13 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	**Last Branch Record 14 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	**Last Branch Record 15 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	**Last Branch Record 0 To IP (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	**Last Branch Record 1 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	**Last Branch Record 2 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6C3H	1731	MSR_ LASTBRANCH_3_TO_IP	Thread	**Last Branch Record 3 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_ LASTBRANCH_4_TO_IP	Thread	**Last Branch Record 4 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_ LASTBRANCH_5_TO_IP	Thread	**Last Branch Record 5 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_ LASTBRANCH_6_TO_IP	Thread	**Last Branch Record 6 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_ LASTBRANCH_7_TO_IP	Thread	**Last Branch Record 7 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_ LASTBRANCH_8_TO_IP	Thread	**Last Branch Record 8 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_ LASTBRANCH_9_TO_IP	Thread	**Last Branch Record 9 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_ LASTBRANCH_10_TO_IP	Thread	**Last Branch Record 10 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_ LASTBRANCH_11_TO_IP	Thread	**Last Branch Record 11 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_ LASTBRANCH_12_TO_IP	Thread	**Last Branch Record 12 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_ LASTBRANCH_13_TO_IP	Thread	**Last Branch Record 13 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_ LASTBRANCH_14_TO_IP	Thread	**Last Branch Record 14 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_ LASTBRANCH_15_TO_IP	Thread	**Last Branch Record 15 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table 35-2.
802H-83FH		X2APIC MSRs	Thread	See Table 35-2.
C000_ 0080H		IA32_EFER	Thread	**Extended Feature Enables** See Table 35-2.
C000_ 0081H		IA32_STAR	Thread	**System Call Target Address (R/W)** See Table 35-2.
C000_ 0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address (R/W)** See Table 35-2.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C000_ 0084H		IA32_FMASK	Thread	**System Call Flag Mask (R/W)** See Table 35-2.
C000_ 0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS (R/W)** See Table 35-2.
C000_ 0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0102H		IA32_KERNEL_GSBASE	Thread	**Swap Target of BASE Address of GS (R/W)** See Table 35-2.
C000_ 0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature (R/W)** See Table 35-2 and Section 17.14.2, "IA32_TSC_AUX Register and RDTSCP Support."

...

35.8.2 MSRs In Intel® Xeon® Processor E5 Family (Based on Intel® Microarchitecture Code Name Sandy Bridge)

Table 35-18 lists selected model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family (based on Intel® microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH, see Table 35-1.

Table 35-18 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17FH	383	MSR_ERROR_CONTROL	Package	**MC Bank Error Configuration (R/W)**
		0		Reserved
		1		**MemError Log Enable (R/W)** When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.

Table 35-18 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge 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.
		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.
285H	645	IA32_MC5_CTL2	Package	See Table 35-2.
286H	646	IA32_MC6_CTL2	Package	See Table 35-2.
287H	647	IA32_MC7_CTL2	Package	See Table 35-2.
288H	648	IA32_MC8_CTL2	Package	See Table 35-2.
289H	649	IA32_MC9_CTL2	Package	See Table 35-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 35-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 35-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 35-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 35-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 35-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 35-2.
290H	656	IA32_MC16_CTL2	Package	See Table 35-2.
291H	657	IA32_MC17_CTL2	Package	See Table 35-2.
292H	658	IA32_MC18_CTL2	Package	See Table 35-2.
293H	659	IA32_MC19_CTL2	Package	See Table 35-2.
39CH	924	MSR_PEBS_NUM_ALT	Package	
		0		**ENABLE_PEBS_NUM_ALT (RW)** Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see Table 19-11
		63:1		Reserved (must be zero).

Table 35-18 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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,” and Chapter 16.
416H	1046	MSR_MC5_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
417H	1047	MSR_MC5_MISC	Package	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.”
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
425H	1061	MSR_MC9_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
426H	1062	MSR_MC9_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
427H	1063	MSR_MC9_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
429H	1065	MSR_MC10_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
42AH	1066	MSR_MC10_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
42BH	1067	MSR_MC10_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
42DH	1069	MSR_MC11_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
42EH	1070	MSR_MC11_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
42FH	1071	MSR_MC11_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
431H	1073	MSR_MC12_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
432H	1074	MSR_MC12_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
433H	1075	MSR_MC12_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”

Table 35-18 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
435H	1077	MSR_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
436H	1078	MSR_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	MSR_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	MSR_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	MSR_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	MSR_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	MSR_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	MSR_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	MSR_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	MSR_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
442H	1090	MSR_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	MSR_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	MSR_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
446H	1094	MSR_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	MSR_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	MSR_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44AH	1098	MSR_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	MSR_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	MSR_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44EH	1102	MSR_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	MSR_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
613H	1555	MSR_PKG_PERF_STATUS	Package	**Package RAPL Perf Status (R/O)**
618H	1560	MSR_DRAM_POWER_LIMIT	Package	**DRAM RAPL Power Limit Control (R/W)** See Section 14.9.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_ STATUS	Package	**DRAM Energy Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	**DRAM Performance Throttling Status (R/O)** See Section 14.9.5, "DRAM RAPL Domain."

Table 35-18 Selected MSRs Supported by Intel® Xeon® Processors E5 Family (based on Sandy Bridge microarchitecture) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
61CH	1564	MSR_DRAM_POWER_INFO	Package	**DRAM RAPL Parameters (R/W)** See Section 14.9.5, "DRAM RAPL Domain."

35.8.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 Family

Intel Xeon Processor E5 family is based on the Sandy Bridge microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 35-19. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH

Table 35-19 Uncore PMU MSRs in Intel® Xeon® Processor E5 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C08H		MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-box UCLK fixed counter control
C09H		MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-box UCLK fixed counter
C10H		MSR_U_PMON_EVNTSEL0	Package	Uncore U-box perfmon event select for U-box counter 0.
C11H		MSR_U_PMON_EVNTSEL1	Package	Uncore U-box perfmon event select for U-box counter 1.
C16H		MSR_U_PMON_CTR0	Package	Uncore U-box perfmon counter 0
C17H		MSR_U_PMON_CTR1	Package	Uncore U-box perfmon counter 1
C24H		MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU perfmon for PCU-box-wide control
C30H		MSR_PCU_PMON_EVNTSEL0	Package	Uncore PCU perfmon event select for PCU counter 0.
C31H		MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU perfmon event select for PCU counter 1.
C32H		MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU perfmon event select for PCU counter 2.
C33H		MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU perfmon event select for PCU counter 3.
C34H		MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU perfmon box-wide filter.
C36H		MSR_PCU_PMON_CTR0	Package	Uncore PCU perfmon counter 0.
C37H		MSR_PCU_PMON_CTR1	Package	Uncore PCU perfmon counter 1.
C38H		MSR_PCU_PMON_CTR2	Package	Uncore PCU perfmon counter 2.
C39H		MSR_PCU_PMON_CTR3	Package	Uncore PCU perfmon counter 3.
D04H		MSR_C0_PMON_BOX_CTL	Package	Uncore C-box 0 perfmon local box wide control.
D10H		MSR_C0_PMON_EVNTSEL0	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 0.
D11H		MSR_C0_PMON_EVNTSEL1	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 1.
D12H		MSR_C0_PMON_EVNTSEL2	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 2.
D13H		MSR_C0_PMON_EVNTSEL3	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 3.

Table 35-19 Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D14H		MSR_C0_PMON_BOX_FILTER	Package	Uncore C-box 0 perfmon box wide filter.
D16H		MSR_C0_PMON_CTR0	Package	Uncore C-box 0 perfmon counter 0.
D17H		MSR_C0_PMON_CTR1	Package	Uncore C-box 0 perfmon counter 1.
D18H		MSR_C0_PMON_CTR2	Package	Uncore C-box 0 perfmon counter 2.
D19H		MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter 3.
D24H		MSR_C1_PMON_BOX_CTL	Package	Uncore C-box 1 perfmon local box wide control.
D30H		MSR_C1_PMON_EVNTSEL0	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 0.
D31H		MSR_C1_PMON_EVNTSEL1	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 1.
D32H		MSR_C1_PMON_EVNTSEL2	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 2.
D33H		MSR_C1_PMON_EVNTSEL3	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 3.
D34H		MSR_C1_PMON_BOX_FILTER	Package	Uncore C-box 1 perfmon box wide filter.
D36H		MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter 0.
D37H		MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter 1.
D38H		MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter 2.
D39H		MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter 3.
D44H		MSR_C2_PMON_BOX_CTL	Package	Uncore C-box 2 perfmon local box wide control.
D50H		MSR_C2_PMON_EVNTSEL0	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 0.
D51H		MSR_C2_PMON_EVNTSEL1	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 1.
D52H		MSR_C2_PMON_EVNTSEL2	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 2.
D53H		MSR_C2_PMON_EVNTSEL3	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 3.
D54H		MSR_C2_PMON_BOX_FILTER	Package	Uncore C-box 2 perfmon box wide filter.
D56H		MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter 0.
D57H		MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter 1.
D58H		MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter 2.
D59H		MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter 3.
D64H		MSR_C3_PMON_BOX_CTL	Package	Uncore C-box 3 perfmon local box wide control.
D70H		MSR_C3_PMON_EVNTSEL0	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 0.
D71H		MSR_C3_PMON_EVNTSEL1	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 1.
D72H		MSR_C3_PMON_EVNTSEL2	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 2.
D73H		MSR_C3_PMON_EVNTSEL3	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 3.
D74H		MSR_C3_PMON_BOX_FILTER	Package	Uncore C-box 3 perfmon box wide filter.
D76H		MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter 0.
D77H		MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter 1.
D78H		MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter 2.

Table 35-19 Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D79H		MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter 3.
D84H		MSR_C4_PMON_BOX_CTL	Package	Uncore C-box 4 perfmon local box wide control.
D90H		MSR_C4_PMON_EVNTSEL0	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 0.
D91H		MSR_C4_PMON_EVNTSEL1	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 1.
D92H		MSR_C4_PMON_EVNTSEL2	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 2.
D93H		MSR_C4_PMON_EVNTSEL3	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 3.
D94H		MSR_C4_PMON_BOX_FILTER	Package	Uncore C-box 4 perfmon box wide filter.
D96H		MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter 0.
D97H		MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter 1.
D98H		MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter 2.
D99H		MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter 3.
DA4H		MSR_C5_PMON_BOX_CTL	Package	Uncore C-box 5 perfmon local box wide control.
DB0H		MSR_C5_PMON_EVNTSEL0	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 0.
DB1H		MSR_C5_PMON_EVNTSEL1	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 1.
DB2H		MSR_C5_PMON_EVNTSEL2	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 2.
DB3H		MSR_C5_PMON_EVNTSEL3	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 3.
DB4H		MSR_C5_PMON_BOX_FILTER	Package	Uncore C-box 5 perfmon box wide filter.
DB6H		MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter 0.
DB7H		MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter 1.
DB8H		MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter 2.
DB9H		MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter 3.
DC4H		MSR_C6_PMON_BOX_CTL	Package	Uncore C-box 6 perfmon local box wide control.
DD0H		MSR_C6_PMON_EVNTSEL0	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 0.
DD1H		MSR_C6_PMON_EVNTSEL1	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 1.
DD2H		MSR_C6_PMON_EVNTSEL2	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 2.
DD3H		MSR_C6_PMON_EVNTSEL3	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 3.
DD4H		MSR_C6_PMON_BOX_FILTER	Package	Uncore C-box 6 perfmon box wide filter.
DD6H		MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter 0.
DD7H		MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter 1.
DD8H		MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter 2.
DD9H		MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter 3.
DE4H		MSR_C7_PMON_BOX_CTL	Package	Uncore C-box 7 perfmon local box wide control.
DF0H		MSR_C7_PMON_EVNTSEL0	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 0.
DF1H		MSR_C7_PMON_EVNTSEL1	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 1.

Table 35-19 Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DF2H		MSR_C7_PMON_EVNTSEL2	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 2.
DF3H		MSR_C7_PMON_EVNTSEL3	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 3.
DF4H		MSR_C7_PMON_BOX_FILTER	Package	Uncore C-box 7 perfmon box wide filter.
DF6H		MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter 0.
DF7H		MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter 1.
DF8H		MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter 2.
DF9H		MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter 3.

...

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 35-21, and Table 35-22.

Table 35-22 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.

Table 35-22 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			
		23:16		**MCG_EXT_CNT**
		24		**MCG_SER_P**
		25		Reserved.
		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.
		62:56		Reserved
		63	Package	**Semaphore for Turbo Ratio Limit Configuration** If 1, the processor uses override configuration[1] specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).
29DH	669	IA32_MC29_CTL2	Package	See Table 35-2.

Table 35-22 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			
29EH	670	IA32_MC30_CTL2	Package	See Table 35-2.
29FH	671	IA32_MC31_CTL2	Package	See Table 35-2.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See Section 18.7.1.1, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
41BH	1051	IA32_MC6_MISC	Package	Misc MAC information of Integrated I/O. (R/O) see Section 15.3.2.4
		5:0		Recoverable Address LSB
		8:6		Address Mode
		15:9		Reserved
		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	

Table 35-22 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			
See Table 35-16, Table 35-21 for other MSR definitions applicable to Intel Xeon processor E7 v2 with CPUID signature 06_3AH				

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

35.9.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Intel Xeon Processor E5 v2 and E7 v2 families are based on the Ivy Bridge-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 35-19 and Table 35-23. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 v2 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3EH.

Table 35-23 Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C00H		MSR_PMON_GLOBAL_CTL	Package	Uncore perfmon per-socket global control.
C01H		MSR_PMON_GLOBAL_STATUS	Package	Uncore perfmon per-socket global status.
C06H		MSR_PMON_GLOBAL_CONFIG	Package	Uncore perfmon per-socket global configuration.
C15H		MSR_U_PMON_BOX_STATUS	Package	Uncore U-box perfmon U-box wide status.
C35H		MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU perfmon box wide status.
D1AH		MSR_C0_PMON_BOX_FILTER1	Package	Uncore C-box 0 perfmon box wide filter1.
D3AH		MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-box 1 perfmon box wide filter1.
D5AH		MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-box 2 perfmon box wide filter1.
D7AH		MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-box 3 perfmon box wide filter1.
D9AH		MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-box 4 perfmon box wide filter1.
DBAH		MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-box 5 perfmon box wide filter1.
DDAH		MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-box 6 perfmon box wide filter1.
DFAH		MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-box 7 perfmon box wide filter1.
E04H		MSR_C8_PMON_BOX_CTL	Package	Uncore C-box 8 perfmon local box wide control.
E10H		MSR_C8_PMON_EVNTSEL0	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 0.
E11H		MSR_C8_PMON_EVNTSEL1	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 1.
E12H		MSR_C8_PMON_EVNTSEL2	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 2.
E13H		MSR_C8_PMON_EVNTSEL3	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 3.

Table 35-23 Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E14H		MSR_C8_PMON_BOX_FILTER	Package	Uncore C-box 8 perfmon box wide filter.
E16H		MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter 0.
E17H		MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter 1.
E18H		MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter 2.
E19H		MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter 3.
E1AH		MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-box 8 perfmon box wide filter1.
E24H		MSR_C9_PMON_BOX_CTL	Package	Uncore C-box 9 perfmon local box wide control.
E30H		MSR_C9_PMON_EVNTSEL0	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 0.
E31H		MSR_C9_PMON_EVNTSEL1	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 1.
E32H		MSR_C9_PMON_EVNTSEL2	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 2.
E33H		MSR_C9_PMON_EVNTSEL3	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 3.
E34H		MSR_C9_PMON_BOX_FILTER	Package	Uncore C-box 9 perfmon box wide filter.
E36H		MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter 0.
E37H		MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter 1.
E38H		MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter 2.
E39H		MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter 3.
E3AH		MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-box 9 perfmon box wide filter1.
E44H		MSR_C10_PMON_BOX_CTL	Package	Uncore C-box 10 perfmon local box wide control.
E50H		MSR_C10_PMON_EVNTSEL0	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 0.
E51H		MSR_C10_PMON_EVNTSEL1	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 1.
E52H		MSR_C10_PMON_EVNTSEL2	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 2.
E53H		MSR_C10_PMON_EVNTSEL3	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 3.
E54H		MSR_C10_PMON_BOX_FILTER	Package	Uncore C-box 10 perfmon box wide filter.
E56H		MSR_C10_PMON_CTR0	Package	Uncore C-box 10 perfmon counter 0.
E57H		MSR_C10_PMON_CTR1	Package	Uncore C-box 10 perfmon counter 1.
E58H		MSR_C10_PMON_CTR2	Package	Uncore C-box 10 perfmon counter 2.
E59H		MSR_C10_PMON_CTR3	Package	Uncore C-box 10 perfmon counter 3.
E5AH		MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-box 10 perfmon box wide filter1.
E64H		MSR_C11_PMON_BOX_CTL	Package	Uncore C-box 11 perfmon local box wide control.
E70H		MSR_C11_PMON_EVNTSEL0	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 0.
E71H		MSR_C11_PMON_EVNTSEL1	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 1.
E72H		MSR_C11_PMON_EVNTSEL2	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 2.
E73H		MSR_C11_PMON_EVNTSEL3	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 3.
E74H		MSR_C11_PMON_BOX_FILTER	Package	Uncore C-box 11 perfmon box wide filter.

Table 35-23 Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E76H		MSR_C11_PMON_CTR0	Package	Uncore C-box 11 perfmon counter 0.
E77H		MSR_C11_PMON_CTR1	Package	Uncore C-box 11 perfmon counter 1.
E78H		MSR_C11_PMON_CTR2	Package	Uncore C-box 11 perfmon counter 2.
E79H		MSR_C11_PMON_CTR3	Package	Uncore C-box 11 perfmon counter 3.
E7AH		MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-box 11 perfmon box wide filter1.
E84H		MSR_C12_PMON_BOX_CTL	Package	Uncore C-box 12 perfmon local box wide control.
E90H		MSR_C12_PMON_EVNTSEL0	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 0.
E91H		MSR_C12_PMON_EVNTSEL1	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 1.
E92H		MSR_C12_PMON_EVNTSEL2	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 2.
E93H		MSR_C12_PMON_EVNTSEL3	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 3.
E94H		MSR_C12_PMON_BOX_FILTER	Package	Uncore C-box 12 perfmon box wide filter.
E96H		MSR_C12_PMON_CTR0	Package	Uncore C-box 12 perfmon counter 0.
E97H		MSR_C12_PMON_CTR1	Package	Uncore C-box 12 perfmon counter 1.
E98H		MSR_C12_PMON_CTR2	Package	Uncore C-box 12 perfmon counter 2.
E99H		MSR_C12_PMON_CTR3	Package	Uncore C-box 12 perfmon counter 3.
E9AH		MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-box 12 perfmon box wide filter1.
EA4H		MSR_C13_PMON_BOX_CTL	Package	Uncore C-box 13 perfmon local box wide control.
EB0H		MSR_C13_PMON_EVNTSEL0	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 0.
EB1H		MSR_C13_PMON_EVNTSEL1	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 1.
EB2H		MSR_C13_PMON_EVNTSEL2	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 2.
EB3H		MSR_C13_PMON_EVNTSEL3	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 3.
EB4H		MSR_C13_PMON_BOX_FILTER	Package	Uncore C-box 13 perfmon box wide filter.
EB6H		MSR_C13_PMON_CTR0	Package	Uncore C-box 13 perfmon counter 0.
EB7H		MSR_C13_PMON_CTR1	Package	Uncore C-box 13 perfmon counter 1.
EB8H		MSR_C13_PMON_CTR2	Package	Uncore C-box 13 perfmon counter 2.
EB9H		MSR_C13_PMON_CTR3	Package	Uncore C-box 13 perfmon counter 3.
EBAH		MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-box 13 perfmon box wide filter1.
EC4H		MSR_C14_PMON_BOX_CTL	Package	Uncore C-box 14 perfmon local box wide control.
ED0H		MSR_C14_PMON_EVNTSEL0	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 0.
ED1H		MSR_C14_PMON_EVNTSEL1	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 1.
ED2H		MSR_C14_PMON_EVNTSEL2	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 2.
ED3H		MSR_C14_PMON_EVNTSEL3	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 3.
ED4H		MSR_C14_PMON_BOX_FILTER	Package	Uncore C-box 14 perfmon box wide filter.
ED6H		MSR_C14_PMON_CTR0	Package	Uncore C-box 14 perfmon counter 0.

Table 35-23 Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
ED7H		MSR_C14_PMON_CTR1	Package	Uncore C-box 14 perfmon counter 1.
ED8H		MSR_C14_PMON_CTR2	Package	Uncore C-box 14 perfmon counter 2.
ED9H		MSR_C14_PMON_CTR3	Package	Uncore C-box 14 perfmon counter 3.
EDAH		MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-box 14 perfmon box wide filter1.

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 35-20, and Table 35-24. For an MSR listed in Table 35-16 that also appears in Table 35-24, Table 35-24 supercede Table 35-16.

The MSRs listed in Table 35-24 also apply to processors based on Haswell-E microarchitecture (see Section 35.11).

Table 35-24 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 35-20
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 18.10.5.1 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 18.10.5.1 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 18.10.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results

Table 35-24 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		33		IN_TXCP: see Section 18.10.5.1 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 18.10.5.1 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**
		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 35-20
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O). See Table 35-20
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O). See Table 35-20
64BH	1611	MSR_CONFIG_TDP_ CONTROL	Package	**ConfigTDP Control (R/W)** See Table 35-20

Table 35-24 Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	**ConfigTDP Control (R/W)** See Table 35-20
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 35-25 lists model-specific registers (MSRs) that are specific to 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3CH/06_45H/06_46H, see Table Table 35-1.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	**C-State Configuration Control (R/W)** Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org.
		3:0		**Package C-State Limit (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s
		9:4		Reserved
		10		**I/O MWAIT Redirection Enable (R/W)**
		14:11		Reserved
		15		**CFG Lock (R/WO)**
		24:16		Reserved
		25		**C3 State Auto Demotion Enable (R/W)**

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

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
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
1C8H	456	MSR_LBR_SELECT	Thread	**Last Branch Record Filtering Select Register (R/W)**
		0		**CPL_EQ_0**
		1		**CPL_NEQ_0**
		2		**JCC**
		3		**NEAR_REL_CALL**
		4		**NEAR_IND_CALL**
		5		**NEAR_RET**
		6		**NEAR_IND_JMP**
		7		**NEAR_REL_JMP**
		8		**FAR_BRANCH**
		9		**EN_CALL_STACK**
		63:9		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
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.
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.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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
		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.

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

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

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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.

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		15:12		**Reserved**

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

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-25 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 35-25 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 35-25 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 35-20, Table 35-24 for other MSR definitions applicable to processors with CPUID signatures 063CH, 06_46H				

...

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 35-21, Table 35-24, and Table 35-27.

Table 35-27 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

Table 35-27 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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)**
		0		Reserved
		1		**MemError Log Enable (R/W)** When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C** Maximum turbo ratio limit of 4 core active.
		39:32	Package	**Maximum Ratio Limit for 5C** Maximum turbo ratio limit of 5 core active.
		47:40	Package	**Maximum Ratio Limit for 6C** Maximum turbo ratio limit of 6 core active.

Table 35-27 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		55:48	Package	**Maximum Ratio Limit for 7C** Maximum turbo ratio limit of 7 core active.
		63:56	Package	**Maximum Ratio Limit for 8C** Maximum turbo ratio limit of 8 core active.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 9C** Maximum turbo ratio limit of 9 core active.
		15:8	Package	**Maximum Ratio Limit for 10C** Maximum turbo ratio limit of 10 core active.
		23:16	Package	**Maximum Ratio Limit for 11C** Maximum turbo ratio limit of 11 core active.
		31:24	Package	**Maximum Ratio Limit for 12C** Maximum turbo ratio limit of 12 core active.
		39:32	Package	**Maximum Ratio Limit for 13C** Maximum turbo ratio limit of 13 core active.
		47:40	Package	**Maximum Ratio Limit for 14C** Maximum turbo ratio limit of 14 core active.
		55:48	Package	**Maximum Ratio Limit for 15C** Maximum turbo ratio limit of 15 core active.
		63:56	Package	**Maximum Ratio Limit for16C** Maximum turbo ratio limit of 16 core active.
1AFH	431	MSR_TURBO_RATIO_LIMIT2	Package	**Maximum Ratio Limit of Turbo Mode** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 17C** Maximum turbo ratio limit of 17 core active.
		15:8	Package	**Maximum Ratio Limit for 18C** Maximum turbo ratio limit of 18 core active.
		62:16	Package	Reserved
		63	Package	**Semaphore for Turbo Ratio Limit Configuration** If 1, the processor uses override configuration[1] specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1 and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).

Table 35-27 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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	
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	

Table 35-27 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
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	
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)**

Table 35-27 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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.
		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)**

Table 35-27 Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		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 35-21, Table 35-24 for other MSR definitions applicable to processors with CPUID signature 06_3FH				

NOTES:
1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

35.11.1 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

Intel Xeon Processor E5 v3 family are based on the Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 35-28. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3FH.

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
700H		MSR_PMON_GLOBAL_CTL	Package	Uncore perfmon per-socket global control.
701H		MSR_PMON_GLOBAL_STATUS	Package	Uncore perfmon per-socket global status.
702H		MSR_PMON_GLOBAL_CONFIG	Package	Uncore perfmon per-socket global configuration.
703H		MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-box UCLK fixed counter control
704H		MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-box UCLK fixed counter
705H		MSR_U_PMON_EVNTSEL0	Package	Uncore U-box perfmon event select for U-box counter 0.
706H		MSR_U_PMON_EVNTSEL1	Package	Uncore U-box perfmon event select for U-box counter 1.
708H		MSR_U_PMON_BOX_STATUS	Package	Uncore U-box perfmon U-box wide status.
709H		MSR_U_PMON_CTR0	Package	Uncore U-box perfmon counter 0
70AH		MSR_U_PMON_CTR1	Package	Uncore U-box perfmon counter 1

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
710H		MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU perfmon for PCU-box-wide control
711H		MSR_PCU_PMON_EVNTSEL0	Package	Uncore PCU perfmon event select for PCU counter 0.
712H		MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU perfmon event select for PCU counter 1.
713H		MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU perfmon event select for PCU counter 2.
714H		MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU perfmon event select for PCU counter 3.
715H		MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU perfmon box-wide filter.
716H		MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU perfmon box wide status.
717H		MSR_PCU_PMON_CTR0	Package	Uncore PCU perfmon counter 0.
718H		MSR_PCU_PMON_CTR1	Package	Uncore PCU perfmon counter 1.
719H		MSR_PCU_PMON_CTR2	Package	Uncore PCU perfmon counter 2.
71AH		MSR_PCU_PMON_CTR3	Package	Uncore PCU perfmon counter 3.
720H		MSR_S0_PMON_BOX_CTL	Package	Uncore SBo 0 perfmon for SBo 0 box-wide control
721H		MSR_S0_PMON_EVNTSEL0	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 0.
722H		MSR_S0_PMON_EVNTSEL1	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 1.
723H		MSR_S0_PMON_EVNTSEL2	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 2.
724H		MSR_S0_PMON_EVNTSEL3	Package	Uncore SBo 0 perfmon event select for SBo 0 counter 3.
725H		MSR_S0_PMON_BOX_FILTER	Package	Uncore SBo 0 perfmon box-wide filter.
726H		MSR_S0_PMON_CTR0	Package	Uncore SBo 0 perfmon counter 0.
727H		MSR_S0_PMON_CTR1	Package	Uncore SBo 0 perfmon counter 1.
728H		MSR_S0_PMON_CTR2	Package	Uncore SBo 0 perfmon counter 2.
729H		MSR_S0_PMON_CTR3	Package	Uncore SBo 0 perfmon counter 3.
72AH		MSR_S1_PMON_BOX_CTL	Package	Uncore SBo 1 perfmon for SBo 1 box-wide control
72BH		MSR_S1_PMON_EVNTSEL0	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 0.
72CH		MSR_S1_PMON_EVNTSEL1	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 1.
72DH		MSR_S1_PMON_EVNTSEL2	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 2.
72EH		MSR_S1_PMON_EVNTSEL3	Package	Uncore SBo 1 perfmon event select for SBo 1 counter 3.
72FH		MSR_S1_PMON_BOX_FILTER	Package	Uncore SBo 1 perfmon box-wide filter.
730H		MSR_S1_PMON_CTR0	Package	Uncore SBo 1 perfmon counter 0.
731H		MSR_S1_PMON_CTR1	Package	Uncore SBo 1 perfmon counter 1.
732H		MSR_S1_PMON_CTR2	Package	Uncore SBo 1 perfmon counter 2.
733H		MSR_S1_PMON_CTR3	Package	Uncore SBo 1 perfmon counter 3.
734H		MSR_S2_PMON_BOX_CTL	Package	Uncore SBo 2 perfmon for SBo 2 box-wide control
735H		MSR_S2_PMON_EVNTSEL0	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 0.
736H		MSR_S2_PMON_EVNTSEL1	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 1.

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
737H		MSR_S2_PMON_EVNTSEL2	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 2.
738H		MSR_S2_PMON_EVNTSEL3	Package	Uncore SBo 2 perfmon event select for SBo 2 counter 3.
739H		MSR_S2_PMON_BOX_FILTER	Package	Uncore SBo 2 perfmon box-wide filter.
73AH		MSR_S2_PMON_CTR0	Package	Uncore SBo 2 perfmon counter 0.
73BH		MSR_S2_PMON_CTR1	Package	Uncore SBo 2 perfmon counter 1.
73CH		MSR_S2_PMON_CTR2	Package	Uncore SBo 2 perfmon counter 2.
73DH		MSR_S2_PMON_CTR3	Package	Uncore SBo 2 perfmon counter 3.
73EH		MSR_S3_PMON_BOX_CTL	Package	Uncore SBo 3 perfmon for SBo 3 box-wide control
73FH		MSR_S3_PMON_EVNTSEL0	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 0.
740H		MSR_S3_PMON_EVNTSEL1	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 1.
741H		MSR_S3_PMON_EVNTSEL2	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 2.
742H		MSR_S3_PMON_EVNTSEL3	Package	Uncore SBo 3 perfmon event select for SBo 3 counter 3.
743H		MSR_S3_PMON_BOX_FILTER	Package	Uncore SBo 3 perfmon box-wide filter.
744H		MSR_S3_PMON_CTR0	Package	Uncore SBo 3 perfmon counter 0.
745H		MSR_S3_PMON_CTR1	Package	Uncore SBo 3 perfmon counter 1.
746H		MSR_S3_PMON_CTR2	Package	Uncore SBo 3 perfmon counter 2.
747H		MSR_S3_PMON_CTR3	Package	Uncore SBo 3 perfmon counter 3.
E00H		MSR_C0_PMON_BOX_CTL	Package	Uncore C-box 0 perfmon for box-wide control
E01H		MSR_C0_PMON_EVNTSEL0	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 0.
E02H		MSR_C0_PMON_EVNTSEL1	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 1.
E03H		MSR_C0_PMON_EVNTSEL2	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 2.
E04H		MSR_C0_PMON_EVNTSEL3	Package	Uncore C-box 0 perfmon event select for C-box 0 counter 3.
E05H		MSR_C0_PMON_BOX_FILTER0	Package	Uncore C-box 0 perfmon box wide filter 0.
E06H		MSR_C0_PMON_BOX_FILTER1	Package	Uncore C-box 0 perfmon box wide filter 1.
E07H		MSR_C0_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon box wide status.
E08H		MSR_C0_PMON_CTR0	Package	Uncore C-box 0 perfmon counter 0.
E09H		MSR_C0_PMON_CTR1	Package	Uncore C-box 0 perfmon counter 1.
E0AH		MSR_C0_PMON_CTR2	Package	Uncore C-box 0 perfmon counter 2.
E0BH		MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter 3.
E10H		MSR_C1_PMON_BOX_CTL	Package	Uncore C-box 1 perfmon for box-wide control
E11H		MSR_C1_PMON_EVNTSEL0	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 0.
E12H		MSR_C1_PMON_EVNTSEL1	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 1.
E13H		MSR_C1_PMON_EVNTSEL2	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 2.
E14H		MSR_C1_PMON_EVNTSEL3	Package	Uncore C-box 1 perfmon event select for C-box 1 counter 3.

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E15H		MSR_C1_PMON_BOX_FILTER0	Package	Uncore C-box 1 perfmon box wide filter 0.
E16H		MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-box 1 perfmon box wide filter1.
E17H		MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon box wide status.
E18H		MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter 0.
E19H		MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter 1.
E1AH		MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter 2.
E1BH		MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter 3.
E20H		MSR_C2_PMON_BOX_CTL	Package	Uncore C-box 2 perfmon for box-wide control
E21H		MSR_C2_PMON_EVNTSEL0	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 0.
E22H		MSR_C2_PMON_EVNTSEL1	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 1.
E23H		MSR_C2_PMON_EVNTSEL2	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 2.
E24H		MSR_C2_PMON_EVNTSEL3	Package	Uncore C-box 2 perfmon event select for C-box 2 counter 3.
E25H		MSR_C2_PMON_BOX_FILTER0	Package	Uncore C-box 2 perfmon box wide filter 0.
E26H		MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-box 2 perfmon box wide filter1.
E27H		MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon box wide status.
E28H		MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter 0.
E29H		MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter 1.
E2AH		MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter 2.
E2BH		MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter 3.
E30H		MSR_C3_PMON_BOX_CTL	Package	Uncore C-box 3 perfmon for box-wide control
E31H		MSR_C3_PMON_EVNTSEL0	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 0.
E32H		MSR_C3_PMON_EVNTSEL1	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 1.
E33H		MSR_C3_PMON_EVNTSEL2	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 2.
E34H		MSR_C3_PMON_EVNTSEL3	Package	Uncore C-box 3 perfmon event select for C-box 3 counter 3.
E35H		MSR_C3_PMON_BOX_FILTER0	Package	Uncore C-box 3 perfmon box wide filter 0.
E36H		MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-box 3 perfmon box wide filter1.
E37H		MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon box wide status.
E38H		MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter 0.
E39H		MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter 1.
E3AH		MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter 2.
E3BH		MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter 3.
E40H		MSR_C4_PMON_BOX_CTL	Package	Uncore C-box 4 perfmon for box-wide control
E41H		MSR_C4_PMON_EVNTSEL0	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 0.
E42H		MSR_C4_PMON_EVNTSEL1	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 1.

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E43H		MSR_C4_PMON_EVNTSEL2	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 2.
E44H		MSR_C4_PMON_EVNTSEL3	Package	Uncore C-box 4 perfmon event select for C-box 4 counter 3.
E45H		MSR_C4_PMON_BOX_FILTER0	Package	Uncore C-box 4 perfmon box wide filter 0.
E46H		MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-box 4 perfmon box wide filter1.
E47H		MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon box wide status.
E48H		MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter 0.
E49H		MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter 1.
E4AH		MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter 2.
E4BH		MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter 3.
E50H		MSR_C5_PMON_BOX_CTL	Package	Uncore C-box 5 perfmon for box-wide control
E51H		MSR_C5_PMON_EVNTSEL0	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 0.
E52H		MSR_C5_PMON_EVNTSEL1	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 1.
E53H		MSR_C5_PMON_EVNTSEL2	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 2.
E54H		MSR_C5_PMON_EVNTSEL3	Package	Uncore C-box 5 perfmon event select for C-box 5 counter 3.
E55H		MSR_C5_PMON_BOX_FILTER0	Package	Uncore C-box 5 perfmon box wide filter 0.
E56H		MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-box 5 perfmon box wide filter1.
E57H		MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon box wide status.
E58H		MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter 0.
E59H		MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter 1.
E5AH		MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter 2.
E5BH		MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter 3.
E60H		MSR_C6_PMON_BOX_CTL	Package	Uncore C-box 6 perfmon for box-wide control
E61H		MSR_C6_PMON_EVNTSEL0	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 0.
E62H		MSR_C6_PMON_EVNTSEL1	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 1.
E63H		MSR_C6_PMON_EVNTSEL2	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 2.
E64H		MSR_C6_PMON_EVNTSEL3	Package	Uncore C-box 6 perfmon event select for C-box 6 counter 3.
E65H		MSR_C6_PMON_BOX_FILTER0	Package	Uncore C-box 6 perfmon box wide filter 0.
E66H		MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-box 6 perfmon box wide filter1.
E67H		MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon box wide status.
E68H		MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter 0.
E69H		MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter 1.
E6AH		MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter 2.
E6BH		MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter 3.
E70H		MSR_C7_PMON_BOX_CTL	Package	Uncore C-box 7 perfmon for box-wide control

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E71H		MSR_C7_PMON_EVNTSEL0	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 0.
E72H		MSR_C7_PMON_EVNTSEL1	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 1.
E73H		MSR_C7_PMON_EVNTSEL2	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 2.
E74H		MSR_C7_PMON_EVNTSEL3	Package	Uncore C-box 7 perfmon event select for C-box 7 counter 3.
E75H		MSR_C7_PMON_BOX_FILTER0	Package	Uncore C-box 7 perfmon box wide filter 0.
E76H		MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-box 7 perfmon box wide filter1.
E77H		MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon box wide status.
E78H		MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter 0.
E79H		MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter 1.
E7AH		MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter 2.
E7BH		MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter 3.
E80H		MSR_C8_PMON_BOX_CTL	Package	Uncore C-box 8 perfmon local box wide control.
E81H		MSR_C8_PMON_EVNTSEL0	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 0.
E82H		MSR_C8_PMON_EVNTSEL1	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 1.
E83H		MSR_C8_PMON_EVNTSEL2	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 2.
E84H		MSR_C8_PMON_EVNTSEL3	Package	Uncore C-box 8 perfmon event select for C-box 8 counter 3.
E85H		MSR_C8_PMON_BOX_FILTER0	Package	Uncore C-box 8 perfmon box wide filter0.
E86H		MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-box 8 perfmon box wide filter1.
E87H		MSR_C8_PMON_BOX_STATUS	Package	Uncore C-box 8 perfmon box wide status.
E88H		MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter 0.
E89H		MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter 1.
E8AH		MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter 2.
E8BH		MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter 3.
E90H		MSR_C9_PMON_BOX_CTL	Package	Uncore C-box 9 perfmon local box wide control.
E91H		MSR_C9_PMON_EVNTSEL0	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 0.
E92H		MSR_C9_PMON_EVNTSEL1	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 1.
E93H		MSR_C9_PMON_EVNTSEL2	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 2.
E94H		MSR_C9_PMON_EVNTSEL3	Package	Uncore C-box 9 perfmon event select for C-box 9 counter 3.
E95H		MSR_C9_PMON_BOX_FILTER0	Package	Uncore C-box 9 perfmon box wide filter0.
E96H		MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-box 9 perfmon box wide filter1.
E97H		MSR_C9_PMON_BOX_STATUS	Package	Uncore C-box 9 perfmon box wide status.
E98H		MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter 0.
E99H		MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter 1.
E9AH		MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter 2.

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E9BH		MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter 3.
EA0H		MSR_C10_PMON_BOX_CTL	Package	Uncore C-box 10 perfmon local box wide control.
EA1H		MSR_C10_PMON_EVNTSEL0	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 0.
EA2H		MSR_C10_PMON_EVNTSEL1	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 1.
EA3H		MSR_C10_PMON_EVNTSEL2	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 2.
EA4H		MSR_C10_PMON_EVNTSEL3	Package	Uncore C-box 10 perfmon event select for C-box 10 counter 3.
EA5H		MSR_C10_PMON_BOX_FILTER0	Package	Uncore C-box 10 perfmon box wide filter0.
EA6H		MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-box 10 perfmon box wide filter1.
EA7H		MSR_C10_PMON_BOX_STATUS	Package	Uncore C-box 10 perfmon box wide status.
EA8H		MSR_C10_PMON_CTR0	Package	Uncore C-box 10 perfmon counter 0.
EA9H		MSR_C10_PMON_CTR1	Package	Uncore C-box 10 perfmon counter 1.
EAAH		MSR_C10_PMON_CTR2	Package	Uncore C-box 10 perfmon counter 2.
EABH		MSR_C10_PMON_CTR3	Package	Uncore C-box 10 perfmon counter 3.
EB0H		MSR_C11_PMON_BOX_CTL	Package	Uncore C-box 11 perfmon local box wide control.
EB1H		MSR_C11_PMON_EVNTSEL0	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 0.
EB2H		MSR_C11_PMON_EVNTSEL1	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 1.
EB3H		MSR_C11_PMON_EVNTSEL2	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 2.
EB4H		MSR_C11_PMON_EVNTSEL3	Package	Uncore C-box 11 perfmon event select for C-box 11 counter 3.
EB5H		MSR_C11_PMON_BOX_FILTER0	Package	Uncore C-box 11 perfmon box wide filter0.
EB6H		MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-box 11 perfmon box wide filter1.
EB7H		MSR_C11_PMON_BOX_STATUS	Package	Uncore C-box 11 perfmon box wide status.
EB8H		MSR_C11_PMON_CTR0	Package	Uncore C-box 11 perfmon counter 0.
EB9H		MSR_C11_PMON_CTR1	Package	Uncore C-box 11 perfmon counter 1.
EBAH		MSR_C11_PMON_CTR2	Package	Uncore C-box 11 perfmon counter 2.
EBBH		MSR_C11_PMON_CTR3	Package	Uncore C-box 11 perfmon counter 3.
EC0H		MSR_C12_PMON_BOX_CTL	Package	Uncore C-box 12 perfmon local box wide control.
EC1H		MSR_C12_PMON_EVNTSEL0	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 0.
EC2H		MSR_C12_PMON_EVNTSEL1	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 1.
EC3H		MSR_C12_PMON_EVNTSEL2	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 2.
EC4H		MSR_C12_PMON_EVNTSEL3	Package	Uncore C-box 12 perfmon event select for C-box 12 counter 3.
EC5H		MSR_C12_PMON_BOX_FILTER0	Package	Uncore C-box 12 perfmon box wide filter0.
EC6H		MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-box 12 perfmon box wide filter1.
EC7H		MSR_C12_PMON_BOX_STATUS	Package	Uncore C-box 12 perfmon box wide status.
EC8H		MSR_C12_PMON_CTR0	Package	Uncore C-box 12 perfmon counter 0.

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
EC9H		MSR_C12_PMON_CTR1	Package	Uncore C-box 12 perfmon counter 1.
ECAH		MSR_C12_PMON_CTR2	Package	Uncore C-box 12 perfmon counter 2.
ECBH		MSR_C12_PMON_CTR3	Package	Uncore C-box 12 perfmon counter 3.
ED0H		MSR_C13_PMON_BOX_CTL	Package	Uncore C-box 13 perfmon local box wide control.
ED1H		MSR_C13_PMON_EVNTSEL0	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 0.
ED2H		MSR_C13_PMON_EVNTSEL1	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 1.
ED3H		MSR_C13_PMON_EVNTSEL2	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 2.
ED4H		MSR_C13_PMON_EVNTSEL3	Package	Uncore C-box 13 perfmon event select for C-box 13 counter 3.
ED5H		MSR_C13_PMON_BOX_FILTER0	Package	Uncore C-box 13 perfmon box wide filter0.
ED6H		MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-box 13 perfmon box wide filter1.
ED7H		MSR_C13_PMON_BOX_STATUS	Package	Uncore C-box 13 perfmon box wide status.
ED8H		MSR_C13_PMON_CTR0	Package	Uncore C-box 13 perfmon counter 0.
ED9H		MSR_C13_PMON_CTR1	Package	Uncore C-box 13 perfmon counter 1.
EDAH		MSR_C13_PMON_CTR2	Package	Uncore C-box 13 perfmon counter 2.
EDBH		MSR_C13_PMON_CTR3	Package	Uncore C-box 13 perfmon counter 3.
EE0H		MSR_C14_PMON_BOX_CTL	Package	Uncore C-box 14 perfmon local box wide control.
EE1H		MSR_C14_PMON_EVNTSEL0	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 0.
EE2H		MSR_C14_PMON_EVNTSEL1	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 1.
EE3H		MSR_C14_PMON_EVNTSEL2	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 2.
EE4H		MSR_C14_PMON_EVNTSEL3	Package	Uncore C-box 14 perfmon event select for C-box 14 counter 3.
EE5H		MSR_C14_PMON_BOX_FILTER	Package	Uncore C-box 14 perfmon box wide filter0.
EE6H		MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-box 14 perfmon box wide filter1.
EE7H		MSR_C14_PMON_BOX_STATUS	Package	Uncore C-box 14 perfmon box wide status.
EE8H		MSR_C14_PMON_CTR0	Package	Uncore C-box 14 perfmon counter 0.
EE9H		MSR_C14_PMON_CTR1	Package	Uncore C-box 14 perfmon counter 1.
EEAH		MSR_C14_PMON_CTR2	Package	Uncore C-box 14 perfmon counter 2.
EEBH		MSR_C14_PMON_CTR3	Package	Uncore C-box 14 perfmon counter 3.
EF0H		MSR_C15_PMON_BOX_CTL	Package	Uncore C-box 15 perfmon local box wide control.
EF1H		MSR_C15_PMON_EVNTSEL0	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 0.
EF2H		MSR_C15_PMON_EVNTSEL1	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 1.
EF3H		MSR_C15_PMON_EVNTSEL2	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 2.
EF4H		MSR_C15_PMON_EVNTSEL3	Package	Uncore C-box 15 perfmon event select for C-box 15 counter 3.
EF5H		MSR_C15_PMON_BOX_FILTER0	Package	Uncore C-box 15 perfmon box wide filter0.
EF6H		MSR_C15_PMON_BOX_FILTER1	Package	Uncore C-box 15 perfmon box wide filter1.

Table 35-28 Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
EF7H		MSR_C15_PMON_BOX_STATUS	Package	Uncore C-box 15 perfmon box wide status.
EF8H		MSR_C15_PMON_CTR0	Package	Uncore C-box 15 perfmon counter 0.
EF9H		MSR_C15_PMON_CTR1	Package	Uncore C-box 15 perfmon counter 1.
EFAH		MSR_C15_PMON_CTR2	Package	Uncore C-box 15 perfmon counter 2.
EFBH		MSR_C15_PMON_CTR3	Package	Uncore C-box 15 perfmon counter 3.
F00H		MSR_C16_PMON_BOX_CTL	Package	Uncore C-box 16 perfmon for box-wide control
F01H		MSR_C16_PMON_EVNTSEL0	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 0.
F02H		MSR_C16_PMON_EVNTSEL1	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 1.
F03H		MSR_C16_PMON_EVNTSEL2	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 2.
F04H		MSR_C16_PMON_EVNTSEL3	Package	Uncore C-box 16 perfmon event select for C-box 16 counter 3.
F05H		MSR_C16_PMON_BOX_FILTER0	Package	Uncore C-box 16 perfmon box wide filter 0.
F06H		MSR_C16_PMON_BOX_FILTER1	Package	Uncore C-box 16 perfmon box wide filter 1.
F07H		MSR_C16_PMON_BOX_STATUS	Package	Uncore C-box 16 perfmon box wide status.
F08H		MSR_C16_PMON_CTR0	Package	Uncore C-box 16 perfmon counter 0.
F09H		MSR_C16_PMON_CTR1	Package	Uncore C-box 16 perfmon counter 1.
F0AH		MSR_C16_PMON_CTR2	Package	Uncore C-box 16 perfmon counter 2.
E0BH		MSR_C16_PMON_CTR3	Package	Uncore C-box 16 perfmon counter 3.
F10H		MSR_C17_PMON_BOX_CTL	Package	Uncore C-box 17 perfmon for box-wide control
F11H		MSR_C17_PMON_EVNTSEL0	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 0.
F12H		MSR_C17_PMON_EVNTSEL1	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 1.
F13H		MSR_C17_PMON_EVNTSEL2	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 2.
F14H		MSR_C17_PMON_EVNTSEL3	Package	Uncore C-box 17 perfmon event select for C-box 17 counter 3.
F15H		MSR_C17_PMON_BOX_FILTER0	Package	Uncore C-box 17 perfmon box wide filter 0.
F16H		MSR_C17_PMON_BOX_FILTER1	Package	Uncore C-box 17 perfmon box wide filter1.
F17H		MSR_C17_PMON_BOX_STATUS	Package	Uncore C-box 17 perfmon box wide status.
F18H		MSR_C17_PMON_CTR0	Package	Uncore C-box 17 perfmon counter 0.
F19H		MSR_C17_PMON_CTR1	Package	Uncore C-box 17 perfmon counter 1.
F1AH		MSR_C17_PMON_CTR2	Package	Uncore C-box 17 perfmon counter 2.
F1BH		MSR_C17_PMON_CTR3	Package	Uncore C-box 17 perfmon counter 3.

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 35-20, Table 35-24, Table 35-25, Table 35-29, and Table 35-30. For an MSR listed in Table 35-30 that also appears in the model-specific tables of prior generations, Table 35-30 supercede prior generation tables.

Table 35-29 lists MSRs that are common to processors based on the Broadwell microarchitectures (including CPUID signatures 06_3DH, 06_47H, 06_4FH, and 06_56H).

Table 35-29 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 35-29 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 35-29 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].

...

35.14 MSRS IN NEXT GENERATION INTEL® CORE™ PROCESSORS

The next generation Intel® Core™ processor family is based on the Sky Lake microarchitecture. They have CPUID DisplayFamily_DisplayModel signatures of 06_4EH and 06_5EH, supports the MSR interfaces listed in Table 35-16, Table 35-17, Table 35-20, Table 35-24, Table 35-30, and Table 35-32. For an MSR listed in Table 35-32 that also appears in the model-specific tables of prior generations, Table 35-32 supercede prior generation tables.

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake 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.
		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.

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		14		**Cross Domain Limit status (RO)** See Table 35-2.
		15		**Cross Domain Limit log (R/WC0)** See Table 35-2.
		22:16		**Digital Readout (RO)** See Table 35-2.
		26:23		Reserved.
		30:27		**Resolution in degrees Celsius (RO)** See Table 35-2.
		31		**Reading Valid (RO)** See Table 35-2.
		63:32		Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS (R/W)** Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.
38EH	910	IA32_PERF_GLOBAL_STAUS		See Table 35-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	**Ovf_PMC0**
		1	Thread	**Ovf_PMC1**
		2	Thread	**Ovf_PMC2**
		3	Thread	**Ovf_PMC3**
		4	Thread	**Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Thread	**Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)**
		6	Thread	**Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Thread	**Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Ovf_FixedCtr0**
		33	Thread	**Ovf_FixedCtr1**
		34	Thread	**Ovf_FixedCtr2**
		54:35		Reserved.
		55	Thread	**Trace_ToPA_PMI.**
		57:56		Reserved.
		58	Thread	**LBR_Frz.**
		59	Thread	**CTR_Frz.**
		60	Thread	**ASCI.**

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		61	Thread	**Ovf_Uncore**
		62	Thread	**Ovf_BufDSSAVE**
		63	Thread	**CondChgd**
390H	912	IA32_PERF_GLOBAL_STATUS_RESET		See Table 35-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	**Set 1 to clear Ovf_PMC0**
		1	Thread	**Set 1 to clear Ovf_PMC1**
		2	Thread	**Set 1 to clear Ovf_PMC2**
		3	Thread	**Set 1 to clear Ovf_PMC3**
		4	Thread	**Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Thread	**Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)**
		6	Thread	**Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Thread	**Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Set 1 to clear Ovf_FixedCtr0**
		33	Thread	**Set 1 to clear Ovf_FixedCtr1**
		34	Thread	**Set 1 to clear Ovf_FixedCtr2**
		54:35		Reserved.
		55	Thread	**Set 1 to clear Trace_ToPA_PMI.**
		57:56		Reserved.
		58	Thread	**Set 1 to clear LBR_Frz.**
		59	Thread	**Set 1 to clear CTR_Frz.**
		60	Thread	**Set 1 to clear ASCI.**
		61	Thread	**Set 1 to clear Ovf_Uncore**
		62	Thread	**Set 1 to clear Ovf_BufDSSAVE**
		63	Thread	**Set 1 to clear CondChgd**
391H	913	IA32_PERF_GLOBAL_STATUS_SET		See Table 35-2. See Section 18.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	**Set 1 to cause Ovf_PMC0 = 1**
		1	Thread	**Set 1 to cause Ovf_PMC1 = 1**
		2	Thread	**Set 1 to cause Ovf_PMC2 = 1**
		3	Thread	**Set 1 to cause Ovf_PMC3 = 1**
		4	Thread	**Set 1 to cause Ovf_PMC4=1 (if CPUID.0AH:EAX[15:8] > 4)**
		5	Thread	**Set 1 to cause Ovf_PMC5=1 (if CPUID.0AH:EAX[15:8] > 5)**

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		6	Thread	**Set 1 to cause Ovf_PMC6=1 (if CPUID.0AH:EAX[15:8] > 6)**
		7	Thread	**Set 1 to cause Ovf_PMC7=1 (if CPUID.0AH:EAX[15:8] > 7)**
		31:8		Reserved.
		32	Thread	**Set 1 to cause Ovf_FixedCtr0 = 1**
		33	Thread	**Set 1 to cause Ovf_FixedCtr1 = 1**
		34	Thread	**Set 1 to cause Ovf_FixedCtr2 = 1**
		54:35		Reserved.
		55	Thread	**Set 1 to cause Trace_ToPA_PMI = 1**
		57:56		Reserved.
		58	Thread	**Set 1 to cause LBR_Frz = 1**
		59	Thread	**Set 1 to cause CTR_Frz = 1**
		60	Thread	**Set 1 to cause ASCI = 1**
		61	Thread	**Set 1 to cause Ovf_Uncore**
		62	Thread	**Set 1 to cause Ovf_BufDSSAVE**
		63		**Reserved**
392H	913	IA32_PERF_GLOBAL_INUSE		See Table 35-2.
64EH	1615	MSR_PPERF	THREAD	Productive Performance Count. (R/O).
		63:0		Hardware's view of workload scalability. See Section 14.4.5.1
652H	1614	MSR_PKG_HDC_CONFIG	Package	**HDC Configuration (R/W).**
		2:0		**PKG_Cx_Monitor.** **Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY**
		63: 3		**Reserved**
653H	1615	MSR_CORE_HDC_ RESIDENCY	Core	Core HDC Idle Residency. (R/O).
		63:0		Core_Cx_Duty_Cycle_Cnt.
655H	1617	MSR_PKG_HDC_SHALLOW_ RESIDENCY	Package	Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle. (R/O).
		63:0		Pkg_C2_Duty_Cycle_Cnt.
656H	1618	MSR_PKG_HDC_DEEP_ RESIDENCY	Package	Package Cx HDC Idle Residency. (R/O).
		63:0		Pkg_Cx_Duty_Cycle_Cnt.
658H	1620	MSR_WEIGHTED_CORE_C0	Package	Core-count Weighted C0 Residency. (R/O).

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.
659H	1621	MSR_ANY_CORE_C0	Package	Any Core C0 Residency. (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor core in the package is in C0.
65AH	1622	MSR_ANY_GFXE_C0	Package	Any Graphics Engine C0 Residency. (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.
65BH	1623	MSR_CORE_GFXE_OVERLAP_C0	Package	Core and Graphics Engine Overlapped C0 Residency. (R/O)
		63:0		Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.
690H	1680	MSR_ LASTBRANCH_16_FROM_IP	Thread	**Last Branch Record 16 From IP (R/W)** One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction**. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.9
691H	1681	MSR_ LASTBRANCH_17_FROM_IP	Thread	**Last Branch Record 17 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_ LASTBRANCH_18_FROM_IP	Thread	**Last Branch Record 18 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_ LASTBRANCH_19_FROM_IP	Thread	**Last Branch Record 19From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_ LASTBRANCH_20_FROM_IP	Thread	**Last Branch Record 20 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_ LASTBRANCH_21_FROM_IP	Thread	**Last Branch Record 21 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_ LASTBRANCH_22_FROM_IP	Thread	**Last Branch Record 22 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
697H	1687	MSR_ LASTBRANCH_23_FROM_IP	Thread	**Last Branch Record 23 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_ LASTBRANCH_24_FROM_IP	Thread	**Last Branch Record 24 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
699H	1689	MSR_ LASTBRANCH_25_FROM_IP	Thread	**Last Branch Record 25 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_ LASTBRANCH_26_FROM_IP	Thread	**Last Branch Record 26 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_ LASTBRANCH_27_FROM_IP	Thread	**Last Branch Record 27 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_ LASTBRANCH_28_FROM_IP	Thread	**Last Branch Record 28 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_ LASTBRANCH_29_FROM_IP	Thread	**Last Branch Record 29 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_ LASTBRANCH_30_FROM_IP	Thread	**Last Branch Record 30 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_ LASTBRANCH_31_FROM_IP	Thread	**Last Branch Record 31 From IP (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
6D0H	1744	MSR_ LASTBRANCH_16_TO_IP	Thread	**Last Branch Record 16 To IP (R/W)** One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **destination instruction**. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.9
6D1H	1745	MSR_ LASTBRANCH_17_TO_IP	Thread	**Last Branch Record 17 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_ LASTBRANCH_18_TO_IP	Thread	**Last Branch Record 18 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_ LASTBRANCH_19_TO_IP	Thread	**Last Branch Record 19To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_ LASTBRANCH_20_TO_IP	Thread	**Last Branch Record 20 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_ LASTBRANCH_21_TO_IP	Thread	**Last Branch Record 21 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_ LASTBRANCH_22_TO_IP	Thread	**Last Branch Record 22 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_ LASTBRANCH_23_TO_IP	Thread	**Last Branch Record 23 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_ LASTBRANCH_24_TO_IP	Thread	**Last Branch Record 24 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6D9H	1753	MSR_ LASTBRANCH_25_TO_IP	Thread	**Last Branch Record 25 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_ LASTBRANCH_26_TO_IP	Thread	**Last Branch Record 26 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DBH	1755	MSR_ LASTBRANCH_27_TO_IP	Thread	**Last Branch Record 27 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_ LASTBRANCH_28_TO_IP	Thread	**Last Branch Record 28 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_ LASTBRANCH_29_TO_IP	Thread	**Last Branch Record 29 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DEH	1758	MSR_ LASTBRANCH_30_TO_IP	Thread	**Last Branch Record 30 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_ LASTBRANCH_31_TO_IP	Thread	**Last Branch Record 31 To IP (R/W)** See description of MSR_LASTBRANCH_0_TO_IP.
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, "Enabling HWP"
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities"
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Section 14.4.4, "Managing HWP"
773H	1907	IA32_HWP_INTERRUPT	Thread	See Section 14.4.6, "HWP Notifications"
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, "Managing HWP"
		7:0		**Minimum Performance (R/W).**
		15:8		**Maximum Performance (R/W).**
		23:16		**Desired Performance (R/W).**
		31:24		**Energy/Performance Preference (R/W).**
		41:32		**Activity Window (R/W).**
		42		**Package Control (R/W).**
		63:43		Reserved.
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, "HWP Feedback"
DB0H	3504	IA32_PKG_HDC_CTL	Package	See Section 14.5.2, "Package level Enabling HDC"
DB1H	3505	IA32_PM_CTL1	Thread	See Section 14.5.3, "Logical-Processor Level HDC Control"
DB2H	3506	IA32_THREAD_STALL	Thread	See Section 14.5.4.1, "IA32_THREAD_STALL"

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DC0H	3520	MSR_LBR_INFO_0	Thread	**Last Branch Record 0 Additional Information (R/W)** One of 32 triplet of last branch record registers on the last branch record stack. This part of the stack contains flag, TSX-related and elapsed cycle information. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.6.1, "LBR Stack."
DC1H	3521	MSR_LBR_INFO_1	Thread	**Last Branch Record 1 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC2H	3522	MSR_LBR_INFO_2	Thread	**Last Branch Record 2 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC3H	3523	MSR_LBR_INFO_3	Thread	**Last Branch Record 3 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC4H	3524	MSR_LBR_INFO_4	Thread	**Last Branch Record 4 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC5H	3525	MSR_LBR_INFO_5	Thread	**Last Branch Record 5 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC6H	3526	MSR_LBR_INFO_6	Thread	**Last Branch Record 6 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC7H	3527	MSR_LBR_INFO_7	Thread	**Last Branch Record 7 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC8H	3528	MSR_LBR_INFO_8	Thread	**Last Branch Record 8 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DC9H	3529	MSR_LBR_INFO_9	Thread	**Last Branch Record 9 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DCAH	3530	MSR_LBR_INFO_10	Thread	**Last Branch Record 10 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DCBH	3531	MSR_LBR_INFO_11	Thread	**Last Branch Record 11 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DCCH	3532	MSR_LBR_INFO_12	Thread	**Last Branch Record 12 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DCDH	3533	MSR_LBR_INFO_13	Thread	**Last Branch Record 13 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DCEH	3534	MSR_LBR_INFO_14	Thread	**Last Branch Record 14 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DCFH	3535	MSR_LBR_INFO_15	Thread	**Last Branch Record 15 Additional Information (R/W)** See description of MSR_LBR_INFO_0.

Table 35-32 Additional MSRs Supported by Future Generation Intel® Core™ Processors Based on Sky Lake Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DD0H	3536	MSR_LBR_INFO_16	Thread	**Last Branch Record 16 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD!H	3537	MSR_LBR_INFO_17	Thread	**Last Branch Record 17 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD2H	3538	MSR_LBR_INFO_18	Thread	**Last Branch Record 18 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD3H	3539	MSR_LBR_INFO_19	Thread	**Last Branch Record 19 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD4H	3520	MSR_LBR_INFO_20	Thread	**Last Branch Record 20 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD5H	3521	MSR_LBR_INFO_21	Thread	**Last Branch Record 21 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD6H	3522	MSR_LBR_INFO_22	Thread	**Last Branch Record 22 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD7H	3523	MSR_LBR_INFO_23	Thread	**Last Branch Record 23 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD8H	3524	MSR_LBR_INFO_24	Thread	**Last Branch Record 24 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DD9H	3525	MSR_LBR_INFO_25	Thread	**Last Branch Record 25 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DDAH	3526	MSR_LBR_INFO_26	Thread	**Last Branch Record 26 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DDBH	3527	MSR_LBR_INFO_27	Thread	**Last Branch Record 27 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DDCH	3528	MSR_LBR_INFO_28	Thread	**Last Branch Record 28 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DDDH	3529	MSR_LBR_INFO_29	Thread	**Last Branch Record 29 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DDEH	3530	MSR_LBR_INFO_30	Thread	**Last Branch Record 30 Additional Information (R/W)** See description of MSR_LBR_INFO_0.
DDFH	3531	MSR_LBR_INFO_31	Thread	**Last Branch Record 31 Additional Information (R/W)** See description of MSR_LBR_INFO_0.

...