Intel® Architecture Instruction Set Extensions and Future Features Programming Reference

319433-042
DECEMBER 2020

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

No product or component can be absolutely secure.

Your costs and results may vary.

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

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

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

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

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

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

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting http://www.intel.com/design/literature.htm.

Copyright © 2020, Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries.

*Other names and brands may be claimed as the property of others.

Revision History

Revision	Description	Date
-025	• Removed instructions that now reside in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Minor updates to chapter 1. • Updates to Table 2-1, Table 2-2 and Table 2-8 (leaf 07H) to indicate support for AVX512_4VNNIW and AVX512_4FMAPS. • Minor update to Table 2-8 (leaf 15H) regarding ECX definition. • Minor updates to Section 4.6.2 and Section 4.6.3 to clarify the effects of "suppress all exceptions". • Footnote addition to CLWB instruction indicating operand encoding requirement. • Removed PCOMMIT.	September 2016
-026	• Removed CLWB instruction; it now resides in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Added additional 512-bit instruction extensions in chapter 6.	October 2016
-027	• Added TLB CPUID leaf in chapter 2. • Added VPOPCNTD/Q instruction in chapter 6,and CPUID details in chapter 2.	December 2016
-028	• Updated intrinsics for VPOPCNTD/Q instruction in chapter 6.	December 2016
-029	• Corrected typo in CPUID leaf 18H. • Updated operand encoding table format; extracted tuple information from operand encoding. • Added VPERMB back into chapter 5; inadvertently removed. • Moved all instructions from chapter 6 to chapter 5. • Updated operation section of VPMULTISHIFTQB.	April 2017
-030	• Removed unnecessary information from document (chapters 2, 3 and 4). • Added table listing recent instruction set extensions introduction in Intel 64 and IA-32 Processors. • Updated CPUID instruction with additional details. • Added the following instructions: GF2P8AFFINEINVQB, GF2P8AFFINEQB, GF2P8MULB, VAESDEC, VAESDECLAST, VAESENC, VAESENCLAST, VPCLMULQDQ, VPCOMPRESS, VPDPBUSD, VPDPBUSDS, VPDPWSSD, VPDPWSSDS, VPEXPAND, VPOPCNT, VPSHLD, VPSHLDV, VPSHRD, VPSHRDV, VPSHUFBITQMB. • Removed the following instructions: VPMADD52HUQ, VPMADD52LUQ, VPERMB, VPERMI2B, VPERMT2B, and VPMULTISHIFTQB. They can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D. • Moved instructions unique to processors based on the Knights Mill microarchitecture to chapter 3. • Added chapter 4: EPT-Based Sub-Page Permissions. • Added chapter 5: Intel® Processor Trace: VMX Improvements.	October 2017

Revision	Description	Date
-031	• Updated change log to correct typo in changes from previous release. • Updated instructions with imm8 operand missing in operand encoding table. • Replaced "VLMAX" with "MAXVL" to align terminology used across documentation. • Added back information on detection of Intel AVX-512 instructions. • Added Intel® Memory Encryption Technologies instructions PCONFIG and WBNOINVD. These instructions are also added to Table 1-1 "Recent Instruction Set Extensions Introduction in Intel 64 and IA-32 Processors". Added Section 1.5 "Detection of Intel® Memory Encryption Technologies (Intel® MKTME) Instructions". • CPUID instruction updated with PCONFIG and WBNOINVD details. • CPUID instruction updated with additional details on leaf 07H: Intel® Xeon Phi™ only features identified and listed. • CPUID instruction updated with new Intel® SGX features in leaf 12H. • CPUID instruction updated with new PCONFIG information sub-leaf 1BH. • Updated short descriptions in the following instructions: VPDPBUSD, VPDPBUSDS, VPDPWSSD and VPDPWSSDS. • Corrections and clarifications in Chapter 4 "EPT-Based Sub-Page Permissions". • Corrections and clarifications in Chapter 5 "Intel® Processor Trace: VMX Improvements".	January 2018
-032	• Corrected PCONFIG CPUID feature flag on instruction page. • Minor updates to PCONFIG instruction pages: Changed Table 2-2 to use Hex notation; changed "RSVD, MBZ" to "Reserved, must be zero" in two places in Table 2-3. • Minor typo correction in WBNOINVD instruction description.	January 2018
-033	• Updated Table 1-2 "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors" . • Added Section 1.4, "Detection of Future Instructions and Features". • Added CLDEMOTE, MOVDIRI, MOVDIR64B, TPAUSE, UMONITOR and UMWAIT instructions. • Updated the CPUID instruction with details on new instructions/features added, as well as new power management details and information on hardware feedback interface ISA extensions. • Corrections to PCONFIG instruction. • Moved instructions unique to processors based on the Knights Mill microarchitecture to the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Added Chapter 5 "Hardware Feedback Interface ISA Extensions". • Added Chapter 6 "AC Split Lock Detection".	March 2018
-034	• Added clarification to leaf 07H in the CPUID instruction. • Added MSR index for IA32_UMWAIT_CONTROL MSR. • Updated registers in TPAUSE and UMWAIT instructions. • Updated TPAUSE and UMWAIT intrinsics.	May 2018

Revision	Description	Date
-035	• Updated Table 1-2 "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors" to list the AVX512_VNNI instruction set architecture on a separate line due to presence on future processors available sooner than previously listed. • Updated CPUID instruction in various places. • Removal of NDD/DDS/NDS terms from instructions. Note: Previously, the terms NDS, NDD and DDS were used in instructions with an EVEX (or VEX) prefix. These terms indicated that the vvvv field was valid for encoding, and specified register usage. These terms are no longer necessary and are redundant with the instruction operand encoding tables provided with each instruction. The instruction operand encoding tables give explicit details on all operands, indicating where every operand is stored and if they are read or written. If vvvv is not listed as an operand in the instruction operand encoding table, then EVEX (or VEX) vvvv must be 0b1111. • Added additional #GP exception condition to TPAUSE and UMWAIT. • Updated Chapter 5 "Hardware Feedback Interface ISA Extensions" as follows: changed scheduler/software to operating system or OS, changed LP0 Scheduler Feedback to LP0 Capability Values, various description updates, clarified that capability updates are independent, and added an update to clarify that bits 0 and 1 will always be set together in Section 5.1.4. • Added IA32_CORE_CAPABILITY MSR to Chapter 6 "AC Split Lock Detection".	October 2018
-036	• Added AVX512_BF16 instructions in chapter 2; related CPUID information updated in chapter 1. • Added new section to chapter 1 describing bfloat16 format. • CPUID leaf updates to align with the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Removed CLDEMOTE, TPAUSE, UMONITOR, and UMWAIT instructions; they now reside in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Changes now marked by green change bars and green font in order to view changes at a text level.	April 2019
-037	• Removed chapter 3, "EPT-Based Sub-Page Permissions", chapter 4, "Intel® Processor Trace: VMX Improvements", and chapter 6, "Split Lock Detection"; this information is in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Removed MOVDIRI and MOVDIR64B instructions; they now reside in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Updated Table 1-2 with new features in future processors. • Updated Table 1-3 with support for AVX512_VP2INTERSECT. • Updated Table 1-5 with support for ENQCMD: Enqueue Stores. • Added ENQCMD/ENQCMDS and VP2INTERSECTD/VP2INTERSECTQ instructions, and updated CPUID accordingly. • Added new chapter: Chapter 4, Enqueue Stores and Process Address Space Identifiers (PASIDs).	May 2019

Revision	Description	Date
-038	• Removed instruction extensions/features from Table 1-2 "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors" that are available in processors covered in the Intel® 64 and IA-32 Architectures Software Developer's Manual. This information can be found in Chapter 5 "Instruction Set Summary", of Volume 1. • In Section 1.7, "Detection of Future Instructions", removed instructions from Table 1-5 "Future Instructions" that are available in processors covered in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Removed instructions with the following CPUID feature flags: AVX512_VNNI, VAES, GFNI (AVX/AVX512), AVX512_VBMI2, VPCLMULQDQ, AVX512_BITALG; they now reside in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • CPUID instruction updated with Hybrid information sub-leaf 1AH, SERIALIZE and TSXLDTRK support, updates to the L3 Cache Intel RDT Monitoring Capability Enumeration Sub-leaf, and updates to the Memory Bandwidth Allocation Enumeration Sub-leaf. • Replaced ← with := notation in operation sections of instructions. These changes are not marked with change bars. • Added the following instructions: SERIALIZE, XRESLDTRK, XSUSLDTRK. • Update to the VDPBF16PS instruction. • Updates to Chapter 4, "Hardware Feedback Interface ISA Extensions". • Added Chapter 5, "TSX Suspend Load Address Tracking". • Added Chapter 6, "Hypervisor-managed Linear Address Translation". • Added Chapter 7, "Architectural Last Branch Records (LBRs)". • Added Chapter 8, "Non-Write-Back Lock Disable Architecture". • Added Chapter 9, "Intel® Resource Director Technology Feature Updates".	March 2020
-039	• Updated Section 1.1 "About this Document" to reflect chapter changes in this release. • Added Section 1.2 "DisplayFamily and DisplayModel for Future Processors". • Updated Table 1-2 "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors". • CPUID instruction updated. • Removed Chapter 4 "Hardware Feedback Interface". This information is now in the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Updated Figure 5-1 "Example HLAT Software Usage". • Added Table 6-5 "Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)" to Chapter 6. • Added Chapter 8 "Bus Lock and VM Notify".	June 2020

Revision	Description	Date
-040	• Updated Section 1.1 “About this Document” to reflect chapter changes in this release. • Updated Table 1-2 “Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors”. • CPUID instruction updated. • Added notation updates to the beginning of Chapter 2. Updated ENQCMD and ENQCMDS instructions to use this notation. • Added Chapter 3, “Intel® AMX Instruction Set Reference, A-Z”. • Minor updates to Chapter 6, “Hypervisor-managed Linear Address Translation”.	June 2020
-041	• Updated Section 1.1 “About this Document” to reflect chapter changes in this release. • Updated Table 1-2 “Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors”. • CPUID instruction updated for enumeration of several new features. • PCONFIG instruction updated. • Added CLUI, HRESET, SENDUIPI, STUI, TESTUI, UIRET, VPDPBUSD, VPDPBUSDS, VPDPWSSD, and VPDPWSSDS instructions to Chapter 2. • Updated Figure 3-2, “The TMUL Unit”. • Update to pseudocode of TILELOADD/TILELOADDT1 instruction. • Addition to Section 6.2, “VMCS Changes”. • Update to Section 7.1.2.4, “Call-Stack Mode”. • Update to Section 9.1 “Bus Lock Debug Exception”. • Added Chapter 11, “User Interrupts”. • Added Chapter 12, “Performance Monitoring Updates”. • Added Chapter 13, “Enhanced Hardware Feedback Interface”.	October 2020
-042	• CPUID instruction updated. • Removed the following instructions: VCVTNE2PS2BF16, VCVTNEPS2BF16, VDPBF16PS, VP2INTERSECTD/VP2INTERSECTQ, and WBNOINVD. They can be found in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C. • Updated bit positions in Section 6.12, “Changes to VMX Capability Reporting”. • Typo correction in Chapter 8, “Non-Write-Back Lock Disable Architecture”. • Several updates to Chapter 13, “Enhanced Hardware Feedback Interface (EHFI)”. • Added Chapter 14, “Linear Address Masking (LAM)”. • Added Chapter 15, “Error Codes for Processors Based on Sapphire Rapids Microarchitecture”.	December 2020

REVISION HISTORY

CHAPTER 1
FUTURE INTEL® ARCHITECTURE INSTRUCTION EXTENSIONS AND FEATURES

1.1 About This Document 1-1
1.2 DisplayFamily and DisplayModel for Future Processors 1-1
1.3 Instruction Set Extensions and Feature Introduction in Intel® 64 and IA-32 Processors 1-2
1.4 Detection of Future Instructions and Features 1-2
1.5 CPUID Instruction 1-3
CPUID—CPU Identification 1-3
1.6 Compressed Displacement (disp8*N) Support in EVEX 1-44
1.7 bfloat16 Floating-Point Format 1-45

CHAPTER 2
INSTRUCTION SET REFERENCE, A-Z

2.1 Instruction Set Reference 2-1
CLUI — Clear User Interrupt Flag 2-2
ENQCMD — Enqueue Command 2-3
ENQCMDS — Enqueue Command Supervisor 2-6
HRESET — History Reset 2-8
PCONFIG — Platform Configuration 2-10
SENDUIPI — Send User Interprocessor Interrupts 2-17
SERIALIZE — Serialize Instruction Execution 2-19
STUI — Set User Interrupt Flag 2-20
TESTUI — Determine User Interrupt Flag 2-21
UIRET — User-Interrupt Return 2-22
VPDPBUSD — Multiply and Add Unsigned and Signed Bytes 2-24
VPDPBUSDS — Multiply and Add Unsigned and Signed Bytes with Saturation 2-25
VPDPWSSD — Multiply and Add Signed Word Integers 2-26
VPDPWSSDS — Multiply and Add Signed Word Integers with Saturation 2-27
XRESLDTRK — Resume Tracking Load Addresses 2-28
XSUSLDTRK— Suspend Tracking Load Addresses 2-29

CHAPTER 3
INTEL® AMX INSTRUCTION SET REFERENCE, A-Z

3.1 Introduction 3-1
3.1.1 Tile Architecture Details 3-3
3.1.2 TMUL Architecture Details 3-4
3.1.3 Handling of Tile Row and Column Limits 3-4
3.1.4 Exceptions and Interrupts 3-5
3.2 Intel® AMX and the XSAVE Feature Set 3-5
3.2.1 State Components for Intel® AMX 3-5
3.2.2 XSAVE-Related Enumeration for Intel® AMX 3-6
3.2.3 Enabling Intel® AMX As an XSAVE-Enabled Feature 3-6
3.2.4 Loading of XTILECFG and XTILEDATA by XRSTOR and XRSTORS 3-7
3.2.5 Saving of XTILEDATA by XSAVE, XSAVEC, XSAVEOPT, and XSAVES 3-7
3.2.6 Extended Feature Disable (XFD) 3-7
3.3 Recommendations for System Software 3-8
3.4 Implementation Parameters 3-8
3.5 Helper Functions 3-8
3.6 Notation 3-9
3.7 Exception Classes 3-10
3.8 Instruction Set Reference 3-11
LDTILECFG — Load Tile Configuration 3-12
STTILECFG — Store Tile Configuration 3-15
TDPBF16PS — Dot Product of BF16 Tiles Accumulated into Packed Single Precision Tile 3-17
TDPBSSD/TDPBSUD/TDPBUSD/TDPBUUD — Dot Product of Signed/Unsigned Bytes with Dword Accumulation 3-19
TILELOADD/TILELOADDT1 — Load Tile 3-21

TILERELEASE — Release Tile . . . 3-23
TILESTORED — Store Tile . . . 3-24
TILEZERO — Zero Tile . . . 3-25

CHAPTER 4
ENQUEUE STORES AND PROCESS ADDRESS SPACE IDENTIFIERS (PASIDS)

4.1 The IA32_PASID MSR . . . 4-1
4.2 The PASID State Component for the XSAVE Feature Set . . . 4-1
4.3 PASID Translation . . . 4-2
4.3.1 PASID Translation Structures . . . 4-2
4.3.2 The PASID Translation Process . . . 4-3
4.3.3 VMX Support . . . 4-4

CHAPTER 5
INTEL® TSX SUSPEND LOAD ADDRESS TRACKING

CHAPTER 6
HYPERVISOR-MANAGED LINEAR ADDRESS TRANSLATION

6.1 Usage . . . 6-1
6.2 VMCS Changes . . . 6-2
6.3 Changes to EPT Paging-Structure Entries . . . 6-2
6.3.1 Reservation of a Guest Page Type in EPT Paging Structure Entry for Future Use . . . 6-3
6.4 Changes to VMX Support for Address Translation . . . 6-3
6.5 Protected Linear Range . . . 6-5
6.6 Hypervisor-Managed Linear Address Translation . . . 6-5
6.6.1 HLAT Overview . . . 6-6
6.6.2 Operation of HLAT . . . 6-6
6.6.3 Format of the HLAT L5E . . . 6-7
6.6.4 Format of the HLAT L4E . . . 6-7
6.6.5 Format of the HLAT L3E . . . 6-8
6.6.6 Format of the HLAT L2E . . . 6-8
6.6.7 Format of the HLAT L1E . . . 6-9
6.6.8 HLAT Faults . . . 6-10
6.6.9 HLAT Operation . . . 6-11
6.6.10 HLAT Interaction with IA and EPT A/D . . . 6-12
6.6.11 Cached HLAT Derived Information . . . 6-12
6.7 Changes to Guest Physical Accesses . . . 6-13
6.7.1 Paging-Write Interaction with EPT A/D . . . 6-14
6.7.2 IOMMU Interaction . . . 6-15
6.8 Addition to EPT violation Exit Qualification . . . 6-15
6.9 HLAT Interaction with Intel® SGX . . . 6-15
6.10 HLAT Interaction with Nested VT-x . . . 6-16
6.11 Changes to VM Entries . . . 6-16
6.12 Changes to VMX Capability Reporting . . . 6-16

CHAPTER 7
ARCHITECTURAL LAST BRANCH RECORDS (LBRS)

7.1 Behavior . . . 7-1
7.1.1 Logged Operations . . . 7-1
7.1.2 Configuration . . . 7-2
7.1.2.1 Enabling and Disabling . . . 7-2
7.1.2.2 LBR Depth . . . 7-2
7.1.2.3 Branch Type Enabling and Filtering . . . 7-2
7.1.2.4 Call-Stack Mode . . . 7-3
Call-Stack Mode and LBR Freeze . . . 7-3
7.1.2.5 CPL Filtering . . . 7-3
7.1.3 Record Data . . . 7-4
7.1.3.1 IP Fields . . . 7-4

7.1.3.2 Branch Types . . . 7-4
7.1.3.3 Cycle Time . . . 7-4
7.1.3.4 Mispredict Information . . . 7-5
7.1.3.5 Intel® TSX Information . . . 7-5
7.1.4 Interaction with Other Processor Features . . . 7-5
7.1.4.1 SMM . . . 7-5
SMM Transfer Monitor (STM) . . . 7-5
7.1.4.2 VMX . . . 7-5
7.1.4.3 Intel® SGX . . . 7-6
7.1.4.4 Debug Breakpoints . . . 7-6
7.1.4.5 SMX . . . 7-6
7.1.4.6 MWAIT . . . 7-6
7.1.4.7 Precise Event-Based Sampling (PEBS) . . . 7-6
7.2 MSRs . . . 7-6
7.3 Context Switch . . . 7-9
7.3.1 XSAVE and LBR Depth . . . 7-10
7.3.2 INIT and MOD Tracking . . . 7-10
7.3.3 Fast LBR Read Access . . . 7-11
7.3.4 XRSTORS Faulting . . . 7-11
7.4 Enumeration . . . 7-11
7.4.1 CPUID for Architectural LBRs . . . 7-11
7.4.2 CPUID for XSAVE LBR Support . . . 7-12
7.4.3 Enumeration for New VMCS Fields . . . 7-12
7.5 other Impacts . . . 7-13
7.5.1 Branch Trace Store on Intel Atom Processors . . . 7-13
7.5.2 IA32_DEBUGCTL . . . 7-13
7.5.3 IA32_PERF_CAPABILITIES . . . 7-13

CHAPTER 8
NON-WRITE-BACK LOCK DISABLE ARCHITECTURE

8.1 Enumeration . . . 8-1
8.2 Enabling . . . 8-1
8.3 Interaction with Intel® Software Guard Extensions (Intel® SGX) . . . 8-2
8.4 Interaction with VMX Architecture . . . 8-2
8.5 Expected Software Behavior . . . 8-2
8.6 Bus Locks . . . 8-3

CHAPTER 9
BUS LOCK AND VM NOTIFY

9.1 Bus Lock Debug Exception . . . 9-1
9.1.1 Bus Lock VM Exit . . . 9-1
9.2 Notify VM Exit . . . 9-1

CHAPTER 10
INTEL® RESOURCE DIRECTOR TECHNOLOGY FEATURE UPDATES

10.1 Intel® RDT Feature Changes . . . 10-1
10.1.1 Intel® RDT on Processors Based on Ice Lake Server Microarchitecture . . . 10-1
10.1.2 Intel® RDT on Intel Atom® Processors Based on Tremont Microarchitecture . . . 10-1
10.1.3 Intel® RDT in Processors Based on Sapphire Rapids Server Microarchitecture . . . 10-1
10.2 Enumerable Memory Bandwidth Monitoring Counter Width . . . 10-2
10.2.1 Memory Bandwidth Monitoring (MBM) Enabling . . . 10-2
10.2.2 Augmented MBM Enumeration and MSR Interfaces for Extensible Counter Width . . . 10-2
10.3 Second Generation Memory Bandwidth Allocation . . . 10-2
10.3.1 MBA 2.0 Advantages . . . 10-3
10.3.2 MBA 2.0 Software-Visible Changes . . . 10-4
10.4 Third Generation Memory Bandwidth Allocation . . . 10-4
10.4.1 MBA 3.0 Hardware Changes . . . 10-4
10.4.2 MBA 3.0 Software-Visible Changes . . . 10-5

CHAPTER 11
USER INTERRUPTS

11.1 Introduction 11-1
11.2 Enumeration and Enabling 11-1
11.3 User-Interrupt State and User-Interrupt MSRs 11-2
11.3.1 User-Interrupt State 11-2
11.3.2 User-Interrupt MSRs 11-2
11.4 Evaluation and Delivery of User Interrupts 11-3
11.4.1 User-Interrupt Recognition 11-3
11.4.2 User-Interrupt Delivery 11-3
11.5 User-Interrupt Notification Identification and Processing 11-5
11.5.1 User-Interrupt Notification Identification 11-5
11.5.2 User-Interrupt Notification Processing 11-6
11.6 New Instructions 11-7
11.7 User IPIs 11-7
11.8 Legacy Instruction Support 11-7
11.8.1 Support by RDMSR and WRMSR 11-7
11.8.2 Support by the XSAVE Feature Set 11-8
11.8.2.1 User-Interrupt State Component 11-8
11.8.2.2 XSAVE-Related Enumeration 11-9
11.8.2.3 XSAVES 11-9
11.8.2.4 XRSTORS 11-10
11.9 VMX Support 11-10
11.9.1 VMCS Changes 11-10
11.9.2 Changes to VMX Non-Root Operation 11-10
11.9.2.1 Treatment of Ordinary Interrupts 11-11
11.9.2.2 Treatment of Virtual Interrupts 11-11
11.9.2.3 VM Exits Incident to New Operations 11-12
11.9.2.4 Access to the User-Interrupt MSRs 11-12
11.9.2.5 Operation of SENDUIPI 11-12
11.9.3 Changes to VM Entries 11-13
11.9.3.1 Checks on the Guest-State Area 11-13
11.9.3.2 Loading MSRs 11-13
11.9.3.3 Event Injection 11-13
11.9.3.4 User-Interrupt Recognition After VM Entry 11-14
11.9.4 Changes to VM Exits 11-14
11.9.4.1 Recording VM-Exit Information 11-14
11.9.4.2 Saving Guest State 11-14
11.9.4.3 Saving MSRs 11-14
11.9.4.4 Loading Host State 11-14
11.9.4.5 Loading MSRs 11-14
11.9.4.6 User-Interrupt Recognition After VM Exit 11-14
11.9.5 Changes to VMX Capability Reporting 11-15

CHAPTER 12
PERFORMANCE MONITORING UPDATES

12.1 Performance Metrics 12-1
12.2 Processor Event Based Sampling (PEBS) Facility 12-2
12.2.1 Instruction-Accurate PDIR (PDIR++) 12-2
12.2.2 Precise Distribution (PDist) 12-2
12.2.3 Load Latency 12-2
12.2.4 Store Latency 12-3
12.3 Adaptive PEBS 12-4
12.3.1 Memory Access Info 12-4
12.4 Performance Monitoring Event List 12-5
12.4.1 Counter Restrictions Simplification 12-5

CHAPTER 13
ENHANCED HARDWARE FEEDBACK INTERFACE (EHFI)

13.1 Enhanced Hardware Feedback Interface Intended Usage Model 13-2

13.2 Hardware Feedback Interface Pointer . . . 13-3
13.3 Hardware Feedback Interface Configuration . . . 13-3
13.4 Hardware Feedback Interface Notifications . . . 13-4
13.5 Hardware Feedback interface Structure Dynamic update . . . 13-5
13.6 Logical Processor Scope Enhanced Hardware Feedback Interface Configuration . . . 13-5
13.7 Implicit Reset of Package and Logical Processor Scope Configuration MSRs . . . 13-5
13.8 Logical Processor Scope Enhanced Hardware Feedback Interface Run Time Characteristics . . . 13-6
13.9 Enhanced Hardware Feedback Interface Enumeration . . . 13-6
13.10 Logical Processor Scope History . . . 13-6
13.10.1 Enabling Enhanced Hardware Feedback Interface History Reset . . . 13-7
13.10.2 Implicit Enhanced Hardware Feedback Interface History Reset . . . 13-7
13.11 IA32_DEBUGCTL MSR Update . . . 13-7

CHAPTER 14
LINEAR ADDRESS MASKING (LAM)

14.1 Enumeration, Enabling, and Configuration . . . 14-1
14.2 Treatment of Data Accesses with LAM Active for User Pointers . . . 14-1
14.3 Treatment of Data Accesses with LAM Active for Supervisor Pointers . . . 14-3
14.4 Canonicality Checking for Data Addresses Written to Control Registers and MSRs . . . 14-4
14.5 Paging Interactions . . . 14-4
14.6 VMX Interactions . . . 14-4
14.6.1 Guest Linear Address . . . 14-4
14.6.2 VM-Entry Checking of Values of CR3 and CR4 . . . 14-5
14.6.3 CR3-Target Values . . . 14-5
14.6.4 Hypervisor-Managed Linear Address Translation (HLAT) . . . 14-5
14.7 Debug and Tracing Interactions . . . 14-5
14.7.1 Debug Registers . . . 14-5
14.7.2 Intel® Processor Trace . . . 14-5
14.8 Intel® SGX Interactions . . . 14-5
14.9 System Management Mode (SMM) Interactions . . . 14-6

CHAPTER 15
ERROR CODES FOR PROCESSORS BASED ON SAPPHIRE RAPIDS MICROARCHITECTURE

15.1 Integrated Memory Controller Machine Check Errors . . . 15-1

TABLES

PAGE

1-1 CPUID Signature Values of DisplayFamily_DisplayModel. 1-1
1-2 Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors. 1-2
1-3 Information Returned by CPUID Instruction 1-4
1-4 Processor Type Field. 1-25
1-5 Feature Information Returned in the ECX Register 1-27
1-6 More on Feature Information Returned in the EDX Register. 1-28
1-7 Encoding of Cache and TLB Descriptors 1-30
1-8 Processor Brand String Returned with Pentium 4 Processor 1-36
1-9 Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings 1-38
1-10 Compressed Displacement (DISP8*N) Affected by Embedded Broadcast 1-44
1-11 EVEX DISP8*N for Instructions Not Affected by Embedded Broadcast 1-44
2-1 PCONFIG Leaf Encodings 2-10
2-2 MKTME_KEY_PROGRAM_STRUCT Format. 2-10
2-3 Supported Key Programming Commands. 2-11
2-4 Supported Key Error Codes. 2-11
2-5 PCONFIG Operation Variables. 2-12
3-1 Intel® AMX Exception Classes. 3-10
3-1 Memory Area Layout 3-12
4-1 IA32_PASID MSR 4-1
6-1 Kernel Shadow Stack and Paging-Write Access Details. 6-3
6-2 Format of HLATP. 6-6
6-3 Format of HLAT L5E. 6-7
6-4 Format of HLAT L4E. 6-7
6-5 Format of HLAT L3E. 6-8
6-6 Format of HLAT L2E. 6-8
6-7 Format of HLAT L1E. 6-9
6-8 HLAT Page Size Mapping. 6-12
6-9 EPT Violation Behavior. 6-14
7-1 LBR IP Values for Various Operations. 7-2
7-2 Branch Type Filtering Details. 7-3
7-3 IA32_LBR_x_INFO and IA32_LER_INFO Branch Type Encodings. 7-4
7-4 LBR VMCS Fields. 7-5
7-5 Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb). 7-6
7-6 Introduction of New MSRs 7-7
7-7 LBR VMCS Fields. 7-10
7-8 CPUID Leaf 01CH Enumeration of Architectural LBR Capabilities. 7-11
7-9 CPUID Leaf 0DH.0FH Enumeration of XSAVE Support for Architectural LBRs. 7-12
8-1 IA32_CORE_CAPABILITIES MSR. 8-1
8-2 TEST_CTRL MSR. 8-2
8-3 Bus Locks from Non-WB Operation 8-3
10-1 MBA_CFG MSR Definition. 10-4
11-1 Format of User Posted-Interrupt Descriptor — UPID 11-5
12-1 Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures) 12-2
12-2 Memory Access Info Group 12-4
12-3 Special Performance Monitoring Events with Counter Restrictions 12-5
13-1 EHFI with Thread-Specific Hardware Feedback Structure 13-1
13-2 IA32_HW_FEEDBACK_CONFIG Control Options. 13-4
15-1 Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13-20). 15-1

FIGURES

PAGE

Figure 1-1. Version Information Returned by CPUID in EAX. 1-25
Figure 1-2. Feature Information Returned in the ECX Register 1-26
Figure 1-3. Feature Information Returned in the EDX Register 1-28
Figure 1-4. Determination of Support for the Processor Brand String 1-36
Figure 1-5. Algorithm for Extracting Maximum Processor Frequency 1-37
Figure 1-6. Comparison of BF16 to FP16 and FP32 1-45
Figure 2-1. 64-Byte Data Written to Enqueue Registers 2-3
Figure 3-1. Intel® AMX Architecture 3-1
Figure 3-2. The TMUL Unit 3-3
Figure 3-3. Matrix Multiply C+= A*B. 3-4
Figure 4-1. PASID Translation Process 4-3
Figure 6-1. Example HLAT Software Usage 6-1
Figure 6-2. HLAT High Level Representation 6-5
Figure 6-3. New Bit in the IA-32e Paging Structures Recognized During HLAT Walks 6-10
Figure 6-4. Example of Paging-Write and Verify-Paging-Write EPT Control Bits Used for Guest Paging Structures (HLAT or Ordinary Paging). 6-14
Figure 10-1. MBA 2.0 Concept Based Around a Hardware Controller 10-3
Figure 12-1. PERF_METRICS MSR Definition for Alder Lake Client and Sapphire Rapids Server Microarchitectures 12-1
Figure 12-2. Deducing Implied Level 2 Metrics in the Core PMU for Alder Lake Client and Sapphire Rapids Server Microarchitectures. 12-1
Figure 14-1. Canonicality Check When LAM48 is Enabled for User Pointers 14-2
Figure 14-2. Canonicality Check When LAM57 is Enabled for User Pointers with 5-Level Paging 14-2
Figure 14-3. Canonicality Check When LAM57 is Enabled for User Pointers with 4-Level Paging 14-3
Figure 14-4. Canonicality Check When LAM57 is Enabled for Supervisor Pointers with 5-Level Paging 14-3
Figure 14-5. Canonicality Check When LAM48 is Enabled for Supervisor Pointers with 4-Level Paging 14-4

CHAPTER 1 FUTURE INTEL® ARCHITECTURE INSTRUCTION EXTENSIONS AND FEATURES

1.1 ABOUT THIS DOCUMENT

This document describes the software programming interfaces of Intel® architecture instruction extensions and features which may be included in future Intel processor generations. Intel does not guarantee the availability of these interfaces and features in any future product.

The instruction set extensions cover a diverse range of application domains and programming usages. The 512-bit SIMD vector SIMD extensions, referred to as Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instructions, deliver comprehensive set of functionality and higher performance than Intel® Advanced Vector Extensions (Intel® AVX) and Intel® Advanced Vector Extensions 2 (Intel® AVX2) instructions. Intel AVX, Intel AVX2 and many Intel AVX-512 instructions are covered in *Intel® 64 and IA-32 Architectures Software Developer's Manual sets.* The reader can refer to them for basic and more background information related to various features referenced in this document.

The base of the 512-bit SIMD instruction extensions are referred to as Intel AVX-512 Foundation instructions. They include extensions of the Intel AVX and Intel AVX2 family of SIMD instructions but are encoded using a new encoding scheme with support for 512-bit vector registers, up to 32 vector registers in 64-bit mode, and conditional processing using opmask registers.

Chapter 2 is an instruction set reference, providing details on new instructions.

Chapter 3 describes the Intel® Advanced Matrix Extensions (Intel® AMX).

Chapter 4 describes ENQCMD/ENQCMDS instructions and virtualization support.

Chapter 5 describes Intel® TSX Suspend Load Address Tracking.

Chapter 6 describes Hypervisor-managed Linear Address Translation.

Chapter 7 describes architectural Last Branch Records (LBRs).

Chapter 8 describes non-write-back lock disable architecture.

Chapter 9 describes bus lock and VM notify features.

Chapter 10 describes Intel® Resource Director Technology feature updates.

Chapter 11 describes user interrupts.

Chapter 12 describes performance monitoring updates.

Chapter 13 describes the enhanced hardware feedback interface (EHFI).

Chapter 14 describes Linear Address Masking (LAM).

Chapter 15 describes the machine error codes for processors based on Sapphire Rapids microarchitecture.

1.2 DISPLAYFAMILY AND DISPLAYMODEL FOR FUTURE PROCESSORS

Table 1-1 lists the signature values of DisplayFamily and DisplayModel for future processor families discussed in this document.

Table 1-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_6AH, 06_6CH	Future processors based on Ice Lake Server microarchitecture
06_8FH	Future processors based on Sapphire Rapids Server microarchitecture

1.3 INSTRUCTION SET EXTENSIONS AND FEATURE INTRODUCTION IN INTEL® 64 AND IA-32 PROCESSORS

Recent instruction set extensions and features are listed in Table 1-2. Within these groups, most instructions and features are collected into functional subgroups.

Table 1-2. Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors[1]

Instruction Set Architecture / Feature	Introduction
PCONFIG, WBNOINVD	Ice Lake Server
Intel® MKTME	Ice Lake Server
ENCLV	Tremont, Ice Lake Server
Direct stores: MOVDIRI, MOVDIR64B	Tremont, Tiger Lake, Sapphire Rapids
AVX512_BF16	Cooper Lake, Sapphire Rapids
CET: Control-flow Enforcement Technology	Tiger Lake, Sapphire Rapids
AVX512_VP2INTERSECT	Tiger Lake, Sapphire Rapids
Enqueue Stores: ENQCMD and ENQCMDS	Sapphire Rapids
CLDEMOTE	Tremont, Alder Lake, Sapphire Rapids
PTWRITE	Goldmont Plus, Alder Lake, Sapphire Rapids
User Wait: TPAUSE, UMONITOR, UMWAIT	Tremont, Alder Lake, Sapphire Rapids
Architectural LBRs	Alder Lake, Sapphire Rapids
HLAT	Alder Lake, Sapphire Rapids
SERIALIZE	Alder Lake, Sapphire Rapids
Intel® TSX Suspend Load Address Tracking (TSXLDTRK)	Sapphire Rapids
Intel® Advanced Matrix Extensions (Intel® AMX) Includes CPUID Leaf 1EH, "TMUL Information Main Leaf", and CPUID bits AMX-BF16, AMX-TILE, and AMX-INT8.	Sapphire Rapids
Key Locker[2]	Tiger Lake, Alder Lake
AVX-VNNI	Alder Lake[3], Sapphire Rapids
Enhanced Hardware Feedback Interface (EHFI) and HRESET	Alder Lake
User Interrupts (UINTR)	Sapphire Rapids
Intel® Trust Domain Extensions (Intel® TDX)[4]	Future Processors

NOTES:

1. Visit Intel Ark for Intel® product specifications, features and compatibility quick reference guide, and code name decoder.
2. Details on Key Locker can be found here: https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html.
3. Alder Lake Intel Hybrid Technology will not support Intel® AVX-512. ISA features such as Intel® AVX, AVX-VNNI, Intel® AVX2, and UMONITOR/UMWAIT/TPAUSE are supported.
4. Details on Intel® Trust Domain Extensions can be found here: https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html.

1.4 DETECTION OF FUTURE INSTRUCTIONS AND FEATURES

Future instructions and features are enumerated by a CPUID feature flag; details can be found in Table 1-3.

1.5 CPUID INSTRUCTION

CPUID—CPU Identification

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
0F A2	CPUID	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).

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 1-3 shows information returned, depending on the initial value loaded into the EAX register.

Two types of information are returned: basic and extended function information. If a value is entered for CPUID.EAX is invalid for a particular processor, the data for the highest basic information leaf is returned. For example, using the Intel Core 2 Duo E6850 processor, the following is true:

```
CPUID.EAX = 05H (* Returns MONITOR/MWAIT leaf. *)
CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)
CPUID.EAX = 0BH (* INVALID: Returns the same information as CPUID.EAX = 0AH. *)2
CPUID.EAX =1FH (* Returns V2 Extended Topology Enumeration leaf. *)2
CPUID.EAX = 80000008H (* Returns virtual/physical address size data. *)
CPUID.EAX = 8000000AH (* INVALID: Returns same information as CPUID.EAX = 0AH. *)
```

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

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

See also:

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

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

1. On Intel 64 processors, CPUID clears the high 32 bits of the RAX/RBX/RCX/RDX registers in all modes.
2. CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of CPUID leaf 1FH before using leaf 0BH.

Table 1-3. Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX EBX ECX EDX	Maximum Input Value for Basic CPUID Information "Genu" "ntel" "inel"
01H	EAX EBX ECX EDX	Version Information: Type, Family, Model, and Stepping ID (see Figure 1-1) Bits 7-0: Brand Index Bits 15-8: 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** Feature Information (see Figure 1-2 and Table 1-5) Feature Information (see Figure 1-3 and Table 1-6) **NOTES:** * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the maximum number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. *** The 8-bit initial APIC ID in EBX[31:24] is replaced by the 32-bit x2APIC ID, available in Leaf 0BH and Leaf 1FH.*
02H	EAX EBX ECX EDX	Cache and TLB Information (see Table 1-7) Cache and TLB Information Cache and TLB Information Cache and TLB Information
03H	EAX EBX ECX EDX	Reserved Reserved Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.) Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.) **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_ENABLES.BOOT_NT4[bit 22] = 0 (default).	
	Deterministic Cache Parameters Leaf	
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX. See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level" on page 1-33.
	EAX	Bits 4-0: Cache Type Field 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved Bits 7-5: Cache Level (starts at 1) Bits 8: Self Initializing cache level (does not need SW initialization) Bits 9: Fully Associative cache

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
		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: WBINVD/INVD behavior on lower level caches Bit 10: 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:** * Add one to the return value to get the result. ** The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache. *** The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. ****The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.
	MONITOR/MWAIT Leaf	
05H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled Bits 31-02: Reserved
	EDX	Bits 03-00: Number of C0* sub C-states supported using MWAIT Bits 07-04: Number of C1* sub C-states supported using MWAIT Bits 11-08: Number of C2* sub C-states supported using MWAIT Bits 15-12: Number of C3* sub C-states supported using MWAIT Bits 19-16: Number of C4* sub C-states supported using MWAIT Bits 23-20: Number of C5* sub C-states supported using MWAIT Bits 27-24: Number of C6* sub C-states supported using MWAIT Bits 31-28: Number of C7* sub C-states supported using MWAIT **NOTE:** * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	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. Bit 14: Intel® Turbo Boost Max Technology 3.0 available. Bit 15: HWP Capabilities. Highest Performance change is supported if set. Bit 16: HWP PECI override is supported if set. Bit 17: Flexible HWP is supported if set. Bit 18: Fast access mode for the IA32_HWP_REQUEST MSR is supported if set. Bit 19: HW_FEEDBACK. IA32_HW_FEEDBACK_PTR, IA32_HW_FEEDBACK_CONFIG, IA32_PACKAGE_THERM_STATUS bit 26 and IA32_PACKAGE_THERM_INTERRUPT bit 25 are supported if set. Bit 20: Ignoring Idle Logical Processor HWP request is supported if set. Bits 22-21: Reserved. Bit 23: Enhanced Hardware Feedback Interface supported if set. IA32_HW_FEEDBACK_CHAR and IA32_HW_FEEDBACK_THREAD_CONFIG MSRs are supported if set. Bits 31-24: 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 07 -04: Reserved = 0 Bits 15-08: Enumerates the number of Enhanced Hardware Feedback interface classes supported by the processor. Information for that many classes is written into the EHFI structure by the hardware. Bits 31-16: Reserved = 0

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 7-0: Bitmap of supported hardware feedback interface capabilities. 0 = When set to 1, indicates support for performance capability reporting. 1 = When set to 1, indicates support for energy efficiency capability reporting. 2-7 = Reserved Bits 11-8: Enumerates the size of the hardware feedback interface structure in number of 4 KB pages; add one to the return value to get the result. Bits 31-16: Index (starting at 0) of this logical processor's row in the hardware feedback interface structure. Note that on some parts the index may be same for multiple logical processors. On some parts the indices may not be contiguous, i.e., there may be unused rows in the hardware feedback interface structure. **NOTE:** Bits 0 and 1 will always be set together.
Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)		
07H		**NOTES:** Leaf 07H main leaf (ECX = 0). If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0.
	EAX	Bits 31-00: Reports the maximum number sub-leaves that are supported in leaf 07H.
	EBX	Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: SGX Bit 03: BMI1 Bit 04: HLE Bit 05: Intel® AVX2 Bit 06: FDP_EXCPTN_ONLY. x87 FPU Data Pointer updated only on x87 exceptions if 1. Bit 07: SMEP. Supports Supervisor Mode Execution Protection if 1. Bit 08: BMI2 Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID Bit 11: RTM Bit 12: RDT-M. Supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability if 1. Bit 13: Deprecates FPU CS and FPU DS values if 1. Bit 14: Intel® Memory Protection Extensions Bit 15: RDT-A. Supports Intel® Resource Director Technology (Intel® RDT) Allocation capability if 1. Bit 16: AVX512F Bit 17: AVX512DQ Bit 18: RDSEED Bit 19: ADX Bit 20: SMAP Bit 21: AVX512_IFMA Bit 22: Reserved Bit 23: CLFLUSHOPT Bit 24: CLWB Bit 25: Intel Processor Trace Bit 26: AVX512PF (Intel® Xeon Phi™ only.) Bit 27: AVX512ER (Intel® Xeon Phi™ only.) Bit 28: AVX512CD Bit 29: SHA Bit 30: AVX512BW Bit 31: AVX512VL

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: PREFETCHWT1 (Intel® Xeon Phi™ only.) Bit 01: AVX512_VBMI Bit 02: UMIP. Supports user-mode instruction prevention if 1. Bit 03: PKU. Supports protection keys for user-mode pages if 1. Bit 04: OSPKE. If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions). Bit 05: WAITPKG Bit 06: AVX512_VBMI2 Bit 07: CET_SS. Supports CET shadow stack features if 1. Processors that set this bit define bits 1:0 of the IA32_U_CET and IA32_S_CET MSRs. Enumerates support for the following MSRs: IA32_INTERRUPT_SPP_TABLE_ADDR, IA32_PL3_SSP, IA32_PL2_SSP, IA32_PL1_SSP, and IA32_PL0_SSP. Bit 08: GFNI Bit 09: VAES Bit 10: VPCLMULQDQ Bit 11: AVX512_VNNI Bit 12: AVX512_BITALG Bit 13: Reserved Bit 14: AVX512_VPOPCNTDQ Bit 15: Reserved Bit 16: LA57. Supports 57-bit linear addresses and five-level paging if 1. Bits 21-17: The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode. Bit 22: RDPID and IA32_TSC_AUX are available if 1. Bit 23: KL. Supports Key Locker if 1. Bit 24: Reserved Bit 25: CLDEMOTE. Supports cache line demote if 1. Bit 26: Reserved Bit 27: MOVDIRI. Supports MOVDIRI if 1. Bit 28: MOVDIR64B. Supports MOVDIR64B if 1. Bit 29: ENQCMD: Supports Enqueue Stores if 1. Bit 30: SGX_LC. Supports SGX Launch Configuration if 1. Bit 31: PKS. Supports protection keys for supervisor-mode pages if 1.
	EDX	Bits 01-00: Reserved Bit 02: AVX512_4VNNIW (Intel® Xeon Phi™ only.) Bit 03: AVX512_4FMAPS (Intel® Xeon Phi™ only.) Bit 04: Fast Short REP MOV Bit 05: UINTR. If 1, the processor supports user interrupts. Bits 07-06: Reserved Bit 08: AVX512_VP2INTERSECT Bit 09: Reserved Bit 10: MD_CLEAR supported. Bits 13-11: Reserved Bit 14: SERIALIZE Bit 15: Hybrid. If 1, the processor is identified as a hybrid part. Bit 16: TSXLDTRK. If 1, the processor supports Intel TSX suspend load address tracking. Bit 17: Reserved Bit 18: PCONFIG Bit 19: Reserved Bit 20: CET_IBT. Supports CET indirect branch tracking features if 1. Processors that set this bit define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs. Bit 21: Reserved Bit 22: AMX-BF16. If 1, the processor supports tile computational operations on bfloat16 numbers. Bit 23: AVX512_FP16 Bit 24: AMX-TILE. If 1, the processor supports tile architecture. Bit 25: AMX-INT8. If 1, the processor supports tile computational operations on 8-bit integers.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
		Bit 26: Enumerates support for indirect branch restricted speculation (IBRS) and the indirect branch predictor barrier (IBPB). Processors that set this bit support the IA32_SPEC_CTRL MSR and the IA32_PRED_CMD MSR. They allow software to set IA32_SPEC_CTRL[0] (IBRS) and IA32_PRED_CMD[0] (IBPB). Bit 27: Enumerates support for single thread indirect branch predictors (STIBP). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[1] (STIBP). Bit 28: Enumerates support for L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_FLUSH). Bit 29: Enumerates support for the IA32_ARCH_CAPABILITIES MSR. Bit 30: Enumerates support for the IA32_CORE_CAPABILITIES MSR. IA32_CORE_CAPABILITIES is an architectural MSR that enumerates model-specific features. A bit being set in this MSR indicates that a model specific feature is supported; software must still consult CPUID family/model/stepping to determine the behavior of the enumerated feature as features enumerated in IA32_CORE_CAPABILITIES may have different behavior on different processor models. Additionally, on hybrid parts (CPUID.07H.0H:EDX[15]=1), software must consult the native model ID and core type from the Hybrid Information Enumeration Leaf. Bit 31: Enumerates support for Speculative Store Bypass Disable (SSBD). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[2] (SSBD). **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.
Structured Extended Feature Enumeration Sub-leaf (EAX = 07H, ECX = 1)		
07H		**NOTES:** Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.
	EAX	This field reports 0 if the sub-leaf index, *1*, is invalid. Bits 03-00: Reserved. Bit 04: AVX-VNNI. AVX (VEX-encoded) versions of the Vector Neural Network Instructions. Bit 05: AVX512_BF16. Vector Neural Network Instructions supporting BFLOAT16 inputs and conversion instructions from IEEE single precision. Bits 09-06: Reserved. Bit 10: If 1, supports fast zero-length MOVSB. Bit 11: If 1, supports fast short STOSB. Bit 12: If 1, supports fast short CMPSB, SCASB. Bits 21-13: Reserved. Bit 22: HRESET. If 1, supports history reset and the IA32_HRESET_ENABLE MSR. When set, indicates that the Processor History Reset Leaf (EAX = 20H) is valid. Bits 25-23: Reserved. Bit 26: LAM. If 1, supports Linear Address Masking. Bits 31-27: Reserved.
	EBX	This field reports 0 if the sub-leaf index, *1*, is invalid; otherwise it is reserved.
	ECX	This field reports 0 if the sub-leaf index, *1*, is invalid; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, *1*, is invalid; otherwise it is reserved.
Structured Extended Feature Enumeration Sub-leaves (EAX = 07H, ECX = n, n ≥ 2)		
07H		**NOTES:** Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.
	EAX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	EBX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	ECX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	Direct Cache Access Information Leaf	
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H)
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
	Architectural Performance Monitoring Leaf	
0AH	EAX	Bits 07-00: Version ID of architectural performance monitoring Bits 15- 08: Number of general-purpose performance monitoring counter per logical processor Bits 23-16: Bit width of general-purpose, performance monitoring counter Bits 31-24: Length of EBX bit vector to enumerate architectural performance monitoring events
	EBX	Bit 00: Core cycle event not available if 1 or if EAX[31:24]<1. Bit 01: Instruction retired event not available if 1 or if EAX[31:24]<2. Bit 02: Reference cycles event not available if 1 or if EAX[31:24]<3. Bit 03: Last-level cache reference event not available if 1 or if EAX[31:24]<4. Bit 04: Last-level cache misses event not available if 1 or if EAX[31:24]<5. Bit 05: Branch instruction retired event not available if 1 or if EAX[31:24]<6. Bit 06: Branch mispredict retired event not available if 1 or if EAX[31:24]<7. Bit 07: Top-down slots event not available if 1 or if EAX[31:24]<8. Bits 31-08: Reserved = 0
	ECX	Bits 31-00: Supported fixed counters. If bit 'i' is set, it implies that Fixed Counter 'i' is supported. Software is recommended to use the following logic to check if a Fixed Counter is supported on a given processor: FxCtr[i]_is_supported := ECX[i] || (EDX[4:0] > i);
	EDX	Bits 04-00: Number of contiguous fixed-function performance counters starting from 0 (if Version ID > 1). Bits 12-05: Bit width of fixed-function performance counters (if Version ID > 1). Bits 14-13: Reserved = 0. Bit 15: AnyThread deprecation. Bits 31-16: Reserved = 0.
	Extended Topology Enumeration Leaf	
0BH		**NOTES:** *CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH.* 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 returns an invalid level-type of 0 in ECX[15:8]; EAX and EBX will return 0. If an input value N in ECX returns the invalid level-type of 0 in ECX[15:8], other input values with ECX > N also return 0 in ECX[15:8]
	EAX	Bits 04-00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31-05: Reserved.
	EBX	Bits 15-00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31-16: Reserved.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 07-00: Level number. Same value in ECX input. Bits 15-08: Level type***. Bits 31-16: Reserved.
	EDX	Bits 31-00: x2APIC ID the current logical processor. **NOTES:** * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0: invalid 1: SMT 2: Core 3-255: Reserved
Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)		
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	Bits 31-00: Reports the valid bit fields of the lower 32 bits of the XFEATURE_ENABLED_MASK register. If a bit is 0, the corresponding bit field in XCR0 is reserved. Bit 00: Legacy x87. Bit 01: 128-bit SSE. Bit 02: 256-bit AVX Bits 04-03: MPX state Bit 07-05: AVX-512 state. Bit 08: Used for IA32_XSS. Bit 09: PKRU state. Bits 12-10: Reserved. Bits 14-13: Used for IA32_XSS. Bits 15: Reserved. Bit 16: Used for IA32_XSS. Bit 17: XTILECFG. Bit 18: XTILEDATA. Bits 31-19: Reserved.
	EBX	Bits 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.
	ECX	Bit 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e all the valid bit fields in XCR0.
	EDX	Bit 31-00: Reports the valid bit fields of the upper 32 bits of the XCR0 register. If a bit is 0, the corresponding bit field in XCR0 is reserved

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)	
0DH	EAX	Bit 00: XSAVEOPT is available. Bit 01: Supports XSAVEC and the compacted form of XRSTOR if set. Bit 02: Supports XGETBV with ECX = 1 if set. Bit 03: Supports XSAVES/XRSTORS and IA32_XSS if set. Bit 04: Supports Extended Feature Disable (XFD) if set. Bits 31-05: Reserved.
	EBX	Bits 31-00: The size in bytes of the XSAVE area containing all states enabled by XCR0 | IA32_XSS.
	ECX	Bits 31-00: Reports the supported bits of the lower 32 bits of the IA32_XSS MSR. IA32_XSS[n] can be set to 1 only if ECX[n] is 1. Bits 07-00: Used for XCR0. Bit 08: PT state. Bit 09: Used for XCR0. Bit 10: Reserved. Bit 11: CET user state. Bit 12: CET supervisor state. Bit 13: HDC state. Bit 14: UINTR state. Bits 15: Reserved. Bit 16: HWP state. Bits 31-17: Reserved.
	EDX	Bits 31-00: Reports the supported bits of the upper 32 bits of the IA32_XSS MSR. IA32_XSS[n+32] can be set to 1 only if EDX[n] is 1. Bits 31-00: Reserved
	Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)	
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. Each sub-leaf index (starting at position 2) is supported if it corresponds to a supported bit in either the XCR0 register or the IA32_XSS MSR. * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n (0 ≤ n ≤ 31) is invalid if sub-leaf 0 returns 0 in EAX[n] and sub-leaf 1 returns 0 in ECX[n]. Sub-leaf n (32 ≤ n ≤ 63) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32].
	EAX	Bits 31-00: 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*. This field reports 0 if the sub-leaf index, *n*, is invalid*.
	EBX	Bits 31-00: The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, n, does not map to a valid bit in the XCR0 register*.
	ECX	Bit 0 is set if the bit n (corresponding to the sub-leaf index) is supported in the IA32_XSS MSR; it is clear if bit n is instead supported in XCR0. Bit 1 is set if, when the compacted format of an XSAVE area is used, this extended state component located on the next 64-byte boundary following the preceding state component (otherwise, it is located immediately following the preceding state component). Bit 2 is set to indicate support for XFD faulting. Bits 31-03 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.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
Intel Resource Director Technology (Intel RDT) Monitoring Enumeration Sub-leaf (EAX = 0FH, ECX = 0)		
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.
	EAX	Reserved.
	EBX	Bits 31-0: Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.
	EDX	Bit 00: Reserved. Bit 01: Supports L3 Cache Intel RDT Monitoring if 1. Bits 31-02: Reserved
L3 Cache Intel RDT Monitoring Capability Enumeration Sub-leaf (EAX = 0FH, ECX = 1)		
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX.
	EAX	No bits set: 24-bit counters. Bits 07 - 00: Encode counter width offset from 24b: 0x0 = 24-bit counters. 0x1 = 25-bit counters. 0x25 = 61-bit counters. Bit 08: Indicates that bit 61 in IA32_QM_CTR MSR is an overflow bit. Bits 31 - 09: Reserved.
	EBX	Bits 31-00: Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.
	ECX	Maximum range (zero-based) of RMID of this resource type.
	EDX	Bit 00: Supports L3 occupancy monitoring if 1. Bit 01: Supports L3 Total Bandwidth monitoring if 1. Bit 02: Supports L3 Local Bandwidth monitoring if 1. Bits 31-03: Reserved
Intel Resource Director Technology (Intel RDT) Allocation Enumeration Sub-leaf (EAX = 10H, ECX = 0)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX.
	EAX	Reserved.
	EBX	Bit 00: Reserved. Bit 01: Supports L3 Cache Allocation Technology if 1. Bit 02: Supports L2 Cache Allocation Technology if 1. Bit 03: Supports Memory Bandwidth Allocation if 1. Bits 31-04: Reserved.
	ECX	Reserved.
	EDX	Reserved.
L3 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID =1)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 04-00: Length of the capacity bit mask for the corresponding ResID using minus-one notation. Bits 31-05: Reserved

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	EBX	Bits 31-00: Bit-granular map of isolation/contention of allocation units.
	ECX	Bit 00: Reserved. Bit 01: Updates of COS should be infrequent if 1. Bit 02: Code and Data Prioritization Technology supported if 1. Bits 31-03: Reserved
	EDX	Bits 15-00: Highest COS number supported for this ResID. Bits 31-16: Reserved
L2 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID =2)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 04 - 00: Length of the capacity bit mask for the corresponding ResID using minus-one notation. Bits 31 - 05: Reserved.
	EBX	Bits 31 - 00: Bit-granular map of isolation/contention of allocation units.
	ECX	Bits 31 - 00: Reserved.
	EDX	Bits 15 - 00: Highest COS number supported for this ResID. Bits 31 - 16: Reserved.
Memory Bandwidth Allocation Enumeration Sub-leaf (EAX = 10H, ECX = ResID =3)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 11 - 00: Reports the maximum MBA throttling value supported for the corresponding ResID using minus-one notation. Bits 31 - 12: Reserved.
	EBX	Bits 31 - 00: Reserved.
	ECX	Bit 00: Per-thread MBA controls are supported. Bit 01: Reserved. Bit 02: Reports whether the response of the delay values is linear. Bits 31 - 03: Reserved.
	EDX	Bits 15 - 00: Highest COS number supported for this ResID. Bits 31 - 16: Reserved.
Intel® Software Guard Extensions (Intel® SGX) Capability Enumeration Leaf, sub-leaf 0 (EAX = 12H, ECX = 0)		
12H		**NOTES:** Leaf 12H sub-leaf 0 (ECX = 0) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 00: SGX1. If 1, Indicates Intel SGX supports the collection of SGX1 leaf functions. Bit 01: SGX2. If 1, Indicates Intel SGX supports the collection of SGX2 leaf functions. Bits 04-02: Reserved. Bit 05: If 1, indicates Intel SGX supports ENCLV instruction leaves EINCVIRTCHILD, EDECVIRTCHILD, and ESETCONTEXT. Bit 06: If 1, indicates Intel SGX supports ENCLS instruction leaves ETRACKC, ERDINFO, ELDBC, and ELDUC. Bits 31-07: Reserved.
	EBX	Bits 31-00: MISCSELECT. Bit vector of supported extended Intel SGX features.
	ECX	Bits 31-00: Reserved.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 07-00: MaxEnclaveSize_Not64. The maximum supported enclave size in non-64-bit mode is 2^(EDX[7:0]). Bits 15-08: MaxEnclaveSize_64. The maximum supported enclave size in 64-bit mode is 2^(EDX[15:8]). Bits 31-16: Reserved.
	Intel SGX Attributes Enumeration Leaf, sub-leaf 1 (EAX = 12H, ECX = 1)	
12H		**NOTES:** Leaf 12H sub-leaf 1 (ECX = 1) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE.
	EBX	Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE.
	ECX	Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE.
	EDX	Bit 31-00: Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.
	Intel SGX EPC Enumeration Leaf, sub-leaves (EAX = 12H, ECX = 2 or higher)	
12H		**NOTES:** Leaf 12H sub-leaf 2 or higher (ECX >= 2) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1. For sub-leaves (ECX = 2 or higher), definition of EDX,ECX,EBX,EAX[31:4] depends on the sub-leaf type listed below.
	EAX	Bit 03-00: Sub-leaf Type 0000b: Indicates this sub-leaf is invalid. 0001b: This sub-leaf enumerates an EPC section. EBX:EAX and EDX:ECX provide information on the Enclave Page Cache (EPC) section. All other type encodings are reserved.
	Type	0000b. This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.
	Type	0001b. This sub-leaf enumerates an EPC sections with EDX:ECX, EBX:EAX defined as follows. EAX[11:04]: Reserved (enumerate 0). EAX[31:12]: Bits 31:12 of the physical address of the base of the EPC section. EBX[19:00]: Bits 51:32 of the physical address of the base of the EPC section. EBX[31:20]: Reserved. ECX[03:00]: EPC section property encoding defined as follows: If EAX[3:0] 0000b, then all bits of the EDX:ECX pair are enumerated as 0. If EAX[3:0] 0001b, then this section has confidentiality and integrity protection. If EAX[3:0] 0010b, then this section has confidentiality protection only. All other encodings are reserved. ECX[11:04]: Reserved (enumerate 0). ECX[31:12]: Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[19:00]: Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[31:20]: Reserved.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	Intel Processor Trace Enumeration Main Leaf (EAX = 14H, ECX = 0)	
14H		**NOTES:** Leaf 14H main leaf (ECX = 0).
	EAX	Bits 31-00: Reports the maximum sub-leaf supported in leaf 14H.
	EBX	Bit 00: If 1, Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_-MATCH MSR can be accessed. Bits 01: If 1, Indicates support of Configurable PSB and Cycle-Accurate Mode. Bits 02: If 1, Indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset. Bits 03: If 1, Indicates support of MTC timing packet and suppression of COFI-based packets. Bit 04: If 1, indicates support of PTWRITE. Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_R-TIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets. Bit 05: If 1, indicates support of Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation. Bit 06: If 1, indicates support for PSB and PMI preservation. Writes can set IA32_RTIT_CTL[56] (InjectPsbPmiOnEnable), enabling the processor to set IA32_RTIT_STATUS[7] (PendTopaPMI) and/or IA32_RTIT_STATUS[6] (PendPSB) in order to preserve ToPA PMIs and/or PSBs otherwise lost due to Intel PT disable. Writes can also set PendToPAPMI and PendPSB. Bits 31-07: Reserved
	ECX	Bit 00: If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. Bit 01: If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. Bits 02: If 1, Indicates support of Single-Range Output scheme. Bits 03: If 1, Indicates support of output to Trace Transport subsystem. Bit 30-04: Reserved Bit 31: If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.
	EDX	Bits 31-00: Reserved
	Intel Processor Trace Enumeration Sub-leaf (EAX = 14H, ECX = 1)	
14H	EAX	Bits 02-00: Number of configurable Address Ranges for filtering. Bits 15-03: Reserved Bit 31-16: Bitmap of supported MTC period encodings
	EBX	Bits 15-00: Bitmap of supported Cycle Threshold value encodings Bit 31-16: Bitmap of supported Configurable PSB frequency encodings
	ECX	Bits 31-00: Reserved
	EDX	Bits 31-00: Reserved
	Time Stamp Counter and Core Crystal Clock Information Leaf	
15H		**NOTES:** If EBX[31:0] is 0, the TSC and "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 fre-quency. If ECX is 0, the core crystal clock frequency is not enumerated. "TSC frequency" = "core crystal clock frequency" * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	EAX	Bits 31-00: An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31-00: An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 31-00: An unsigned integer which is the nominal frequency of the core crystal clock in Hz.
	EDX	Bits 31-00: Reserved = 0.
	Processor Frequency Information Leaf	
16H	EAX	Bits 15-00: Processor Base Frequency (in MHz). Bits 31-16: Reserved =0
	EBX	Bits 15-00: Maximum Frequency (in MHz). Bits 31-16: Reserved = 0
	ECX	Bits 15-00: Bus (Reference) Frequency (in MHz). Bits 31-16: Reserved = 0
	EDX	Reserved
		NOTES: * Data is returned from this interface in accordance with the processor's specification and does not reflect actual values. Suitable use of this data includes the display of processor information in like manner to the processor brand string and for determining the appropriate range to use when displaying processor information e.g. frequency history graphs. The returned information should not be used for any other purpose as the returned information does not accurately correlate to information / counters returned by other processor interfaces. While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.
	System-On-Chip Vendor Attribute Enumeration Main Leaf (EAX = 17H, ECX = 0)	
17H		**NOTES:** Leaf 17H main leaf (ECX = 0). Leaf 17H output depends on the initial value in ECX. Leaf 17H sub-leaves 1 through 3 reports SOC Vendor Brand String. Leaf 17H is valid if MaxSOCID_Index >= 3. Leaf 17H sub-leaves 4 and above are reserved.
	EAX	Bits 31-00: MaxSOCID_Index. Reports the maximum input value of supported sub-leaf in leaf 17H.
	EBX	Bits 15-00: SOC Vendor ID. Bit 16: IsVendorScheme. If 1, the SOC Vendor ID field is assigned via an industry standard enumeration scheme. Otherwise, the SOC Vendor ID field is assigned by Intel. Bits 31-17: Reserved = 0.
	ECX	Bits 31-00: Project ID. A unique number an SOC vendor assigns to its SOC projects.
	EDX	Bits 31-00: Stepping ID. A unique number within an SOC project that an SOC vendor assigns.
	System-On-Chip Vendor Attribute Enumeration Sub-leaf (EAX = 17H, ECX = 1..3)	
17H	EAX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	EBX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	ECX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	EDX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
		NOTES: Leaf 17H output depends on the initial value in ECX. SOC Vendor Brand String is a UTF-8 encoded string padded with trailing bytes of 00H. The complete SOC Vendor Brand String is constructed by concatenating in ascending order of EAX:EBX:ECX:EDX and from the sub-leaf 1 fragment towards sub-leaf 3.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
System-On-Chip Vendor Attribute Enumeration Sub-leaves (EAX = 17H, ECX > MaxSOCID_Index)		
17H		**NOTES:** Leaf 17H output depends on the initial value in ECX.
	EAX	Bits 31-00: Reserved = 0.
	EBX	Bits 31-00: Reserved = 0.
	ECX	Bits 31-00: Reserved = 0.
	EDX	Bits 31-00: Reserved = 0.
Deterministic Address Translation Parameters Main Leaf (EAX = 18H, ECX = 0)		
18H		**NOTES:** Each sub-leaf enumerates a different address translations structure. Valid sub-leaves do not need to be contiguous or in any particular order. A valid sub-leaf may be in a higher input ECX value than an invalid sub-leaf or than a valid sub-leaf of a higher or lower-level structure. 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. * Add one to the return value to get the result.
	EAX	Bits 31-00: Reports the maximum input value of supported sub-leaf in leaf 18H.
	EBX	Bit 00: 4K page size entries supported by this structure. Bit 01: 2MB page size entries supported by this structure. Bit 02: 4MB page size entries supported by this structure. Bit 03: 1 GB page size entries supported by this structure. Bits 07-04: Reserved. Bits 10-08: Partitioning (0: Soft partitioning between the logical processors sharing this structure). Bits 15-11: Reserved. Bits 31-16: W = Ways of associativity.
	ECX	Bits 31-00: S = Number of Sets.
	EDX	Bits 04-00: Translation cache type field. 00000b: Null (indicates this sub-leaf is not valid). 00001b: Data TLB. 00010b: Instruction TLB. 00011b: Unified TLB. 00100b: Load Only TLB. Hit on loads; fills on both loads and stores. 00101b: Store Only TLB. Hit on stores; fill on stores. All other encodings are reserved. Bits 07-05: Translation cache level (starts at 1). Bit 08: Fully associative structure. Bits 13-09: Reserved. Bits 25-14: Maximum number of addressable IDs for logical processors sharing this translation cache* Bits 31-26: Reserved.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	Deterministic Address Translation Parameters Sub-leaf (EAX = 18H, ECX ≥ 1)	
18H		**NOTES:** 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. * Add one to the return value to get the result.
	EAX	Bits 31-00: Reserved.
	EBX	Bit 00: 4K page size entries supported by this structure. Bit 01: 2MB page size entries supported by this structure. Bit 02: 4MB page size entries supported by this structure. Bit 03: 1 GB page size entries supported by this structure. Bits 07-04: Reserved. Bits 10-08: Partitioning (0: Soft partitioning between the logical processors sharing this structure). Bits 15-11: Reserved. Bits 31-16: W = Ways of associativity.
	ECX	Bits 31-00: S = Number of Sets.
	EDX	Bits 04-00: Translation cache type field. 0000b: Null (indicates this sub-leaf is not valid). 0001b: Data TLB. 0010b: Instruction TLB. 0011b: Unified TLB. All other encodings are reserved. Bits 07-05: Translation cache level (starts at 1). Bit 08: Fully associative structure. Bits 13-09: Reserved. Bits 25-14: Maximum number of addressable IDs for logical processors sharing this translation cache* Bits 31-26: Reserved.
	Key Locker Leaf (EAX = 19H)	
19H	EAX	Bit 00: Key Locker restriction of CPL0-only supported. Bit 01: Key Locker restriction of no-encrypt supported. Bit 02: Key Locker restriction of no-decrypt supported. Bits 31-03: Reserved.
	EBX	Bit 00: AESKLE. If 1, the AES Key Locker instructions are fully enabled. Bit 01: Reserved. Bit 02: If 1, the AES wide Key Locker instructions are supported. Bit 03: Reserved. Bit 04: If 1, the platform supports the Key Locker MSRs and backing up the internal wrapping key. Bits 31-05: Reserved.
	ECX	Bit 00: If 1, the NoBackup parameter to LOADIWKEY is supported. Bit 01: If 1, KeySource encoding of 1 (randomization of the internal wrapping key) is supported. Bits 31- 02: Reserved.
	EDX	Reserved.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	Hybrid Information Sub-leaf (EAX = 1AH, ECX = 0)	
1AH	EAX	Enumerates the native model ID and core type. Bits 31-24: Core type 10H: Reserved 20H: Intel Atom® 30H: Reserved 40H: Intel® Core™ Bits 23-0: Reserved
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
	PCONFIG Information Sub-leaf (EAX = 1BH, ECX ≥ 0)	
1BH		**NOTES:** Leaf 1BH is supported if CPUID.(EAX=07H, ECX=0H):EDX[18] = 1. For sub-leaves of 1BH, the definition of EDX, ECX, EBX, EAX depends on the sub-leaf type listed below. * Currently MKTME is the only defined target and is indicated by identifier 1. An identifier of 0 indicates an invalid target. If MKTME is a supported target, the MKTME_KEY_PROGRAM leaf of PCONFIG is available.
	EAX	Bits 11-00: Sub-leaf type 0: Invalid sub-leaf. On an invalid sub-leaf type returned, subsequent sub-leaves are also invalid. EBX, ECX and EDX all return 0 for this case. 1: Target Identifier. This sub-leaf enumerates PCONFIG targets supported on the platform. Software must scan until an invalid sub-leaf type is returned. EBX, ECX and EDX are defined below for this case. Bits 31-12: 0
	EBX	* Identifier of target 3n+1 (where n is the sub-leaf number, the initial value of ECX).
	ECX	* Identifier of target 3n+2.
	EDX	* Identifier of target 3n+3.
	Tile Information Main Leaf (EAX = 1DH, ECX = 0)	
1DH		**NOTE:** For sub-leaves of 1DH, they are indexed by the palette id. Leaf 1DH sub-leaves 2 and above are reserved.
	EAX	Bits 31-00: max_palette. Highest numbered palette sub-leaf. Value = 1.
	EBX	Bits 31-00: Reserved = 0.
	ECX	Bits 31-00: Reserved = 0.
	EDX	Bits 31-00: Reserved = 0.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	Tile Palette 1 Sub-leaf (EAX = 1DH, ECX = 1)	
1DH	EAX	Bits 15-00: Palette 1 total_tile_bytes. Value = 8192. Bits 31-16: Palette 1 bytes_per_tile. Value = 1024.
	EBX	Bits 15-00: Palette 1 bytes_per_row. Value = 64. Bits 31-16: Palette 1 max_names (number of tile registers). Value = 8.
	ECX	Bits 15-00: Palette 1 max_rows. Value = 16. Bits 31-16: Reserved = 0.
	EDX	Bits 31-00: Reserved = 0.
	TMUL Information Main Leaf (EAX = 1EH, ECX = 0)	
1EH		**NOTE:** Leaf 1EH sub-leaf 1 and above are reserved.
	EAX	Bits 31-00: Reserved = 0.
	EBX	Bits 07-00: tmul_maxk (rows or columns). Value = 16. Bits 23-08: tmul_maxn (column bytes). Value = 64. Bits 31-24: Reserved = 0.
	ECX	Bits 31-00: Reserved = 0.
	EDX	Bits 31-00: Reserved = 0.
	V2 Extended Topology Enumeration Leaf	
1FH		**NOTES:** *CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH and using this if available.* Most of Leaf 1FH output depends on the initial value in ECX. The EDX output of leaf 1FH 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]. Sub-leaf index 0 enumerates SMT level. Each subsequent higher sub-leaf index enumerates a higher-level topological entity in hierarchical order. 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.

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
		NOTES: * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0: Invalid. 1: SMT. 2: Core. 3: Module. 4: Tile. 5: Die. 6-255: Reserved.
Processor History Reset Sub-leaf (EAX = 20H, ECX = 0)		
20H	EAX	Reports the maximum number of sub-leaves that are supported in leaf 20H.
	EBX	Indicates which bits may be set in the IA32_HRESET_ENABLE MSR to enable enhanced hardware feedback interface history. Bit 00: Indicates support for both HRESET's EAX[0] parameter, and IA32_HRESET_ENABLE[0] set by the OS to enable reset of EHFI history. Bits 31-01: Reserved for other history reset capabilities.
	ECX	Reserved.
	EDX	Reserved.
Unimplemented CPUID Leaf Functions		
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
Extended Function CPUID Information		
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information.
	EBX ECX EDX	Reserved Reserved Reserved
80000001H	EAX EBX ECX	Extended Processor Signature and Feature Bits. Reserved Bit 00: LAHF/SAHF available in 64-bit mode Bits 04-01: Reserved Bit 05: LZCNT available Bits 07-06: Reserved Bit 08: PREFETCHW Bits 31-09: Reserved

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 10-00: Reserved Bit 11: SYSCALL/SYSRET available (when 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 ECX EDX	Reserved = 0 Reserved = 0 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 Reserved = 0 **NOTES:** * L2 associativity field encodings: 00H - Disabled 01H - 1 way (direct mapped) 02H - 2 ways 03H - Reserved 04H - 4 ways 05H - Reserved 06H - 8 ways 07H - See CPUID leaf 04H, sub-leaf 2** 08H - 16 ways 09H - Reserved 0AH - 32 ways 0BH - 48 ways 0CH - 64 ways 0DH - 96 ways 0EH - 128 ways 0FH - Fully associative ** CPUID leaf 04H provides details of deterministic cache parameters, including the L2 cache in sub-leaf 2

Table 1-3. Information Returned by CPUID Instruction(Continued)

Initial EAX Value	Information Provided about the Processor	
80000007H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Bits 07-00: Reserved = 0 Bit 08: Invariant TSC available if 1 Bits 31-09: Reserved = 0
80000008H	EAX EBX ECX EDX	Virtual/Physical Address size Bits 07-00: #Physical Address Bits* Bits 15-08: #Virtual Address Bits Bits 31-16: Reserved = 0 Bits 08-00: Reserved = 0 Bit 09: WBNOINVD is available if 1 Bits 31-10: Reserved = 0 Reserved = 0 Reserved = 0 **NOTES:** * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

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

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

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

EBX := 756e6547h (* "Genu", with G in the low 4 bits of BL *)
EDX := 49656e69h (* "inel", with i in the low 4 bits of DL *)
ECX := 6c65746eh (* "ntel", with n in the low 4 bits of CL *)

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

When CPUID executes with EAX set to 0H, 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 10 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 1-1). 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 1-4 for available processor type values. Stepping IDs are provided as needed.

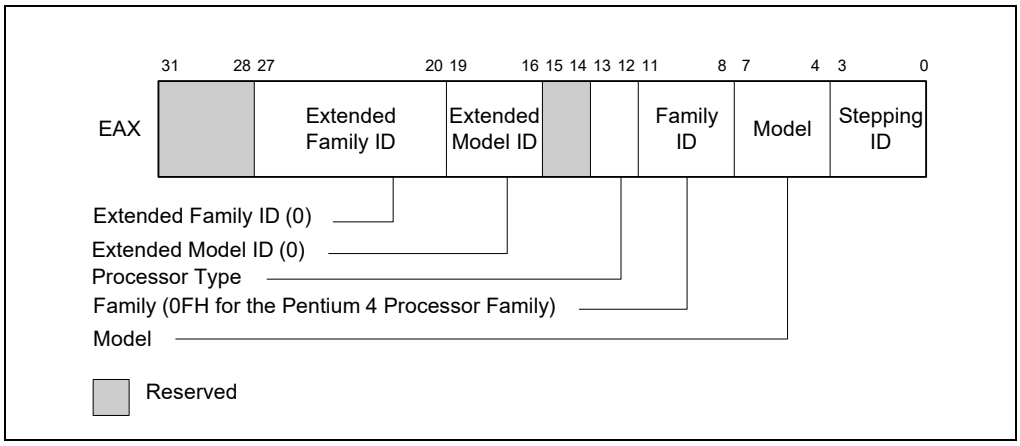


Figure 1-1. Version Information Returned by CPUID in EAX

Table 1-4. 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 "Caching Translation Information" in Chapter 4, "Paging," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, and Chapter 16 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 Displayed_Family = Family_ID;
    ELSE Displayed_Family = Extended_Family_ID + Family_ID;
    (* Right justify and zero-extend 4-bit field. *)
FI;
(* Show Display_Family 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 Displayed_Model = (Extended_Model_ID << 4) + Model_ID;
    (* Right justify and zero-extend 4-bit field; display Model_ID as HEX field.*)
    ELSE Displayed_Model = Model_ID;
FI;
(* Show Display_Model 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 1-2 and Table 1-5 show encodings for ECX.
- Figure 1-3 and Table 1-6 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.

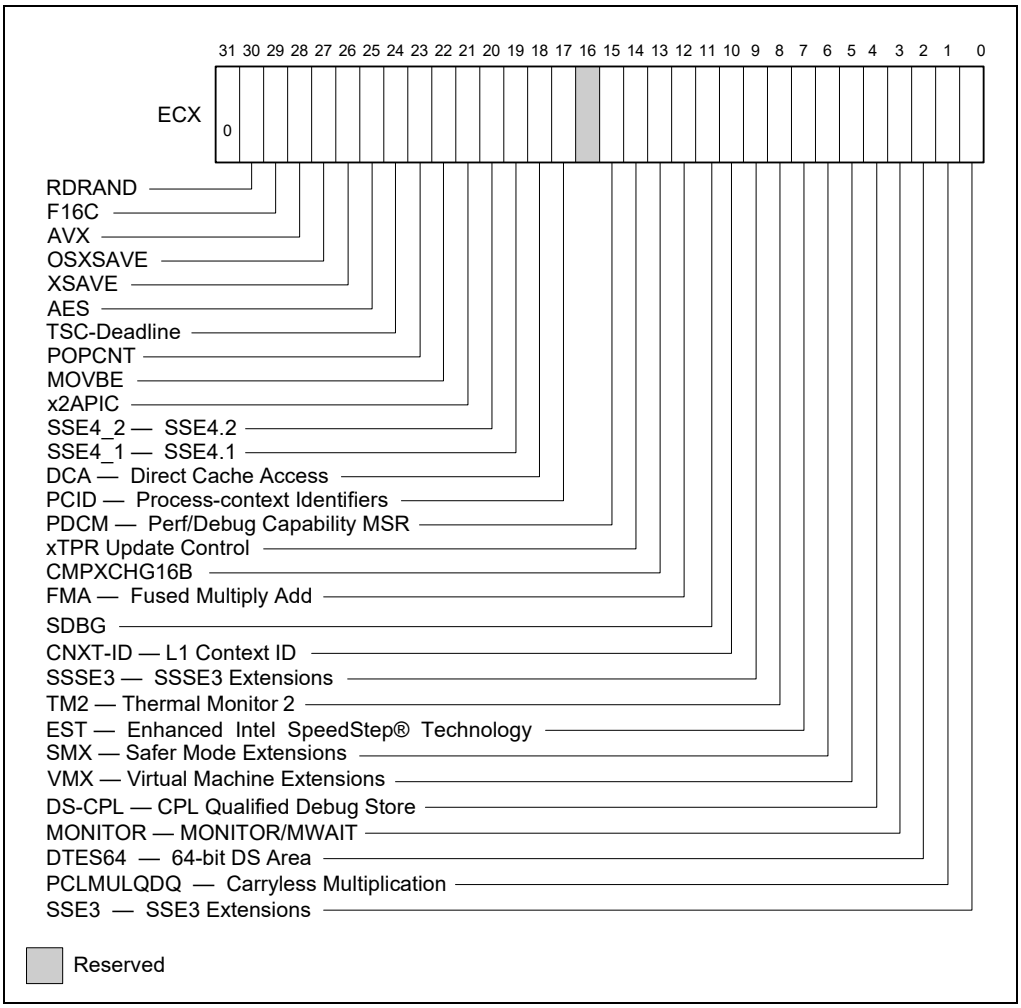


Figure 1-2. Feature Information Returned in the ECX Register

Table 1-5. Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Intel® Streaming SIMD Extensions 3 (Intel® SSE3).** A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	A value of 1 indicates the processor supports 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 6, "Safer Mode Extensions Reference".
7	EST	**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.
14	xTPR Update Control	**xTPR Update Control.** A value of 1 indicates that the processor supports changing IA32_MISC_ENABLES[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	AES	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 that processor supports AVX instructions operating on 256-bit YMM state, and three-operand encoding of 256-bit and 128-bit SIMD instructions.

Table 1-5. Feature Information Returned in the ECX Register (Continued)

Bit #	Mnemonic	Description
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 return 0.

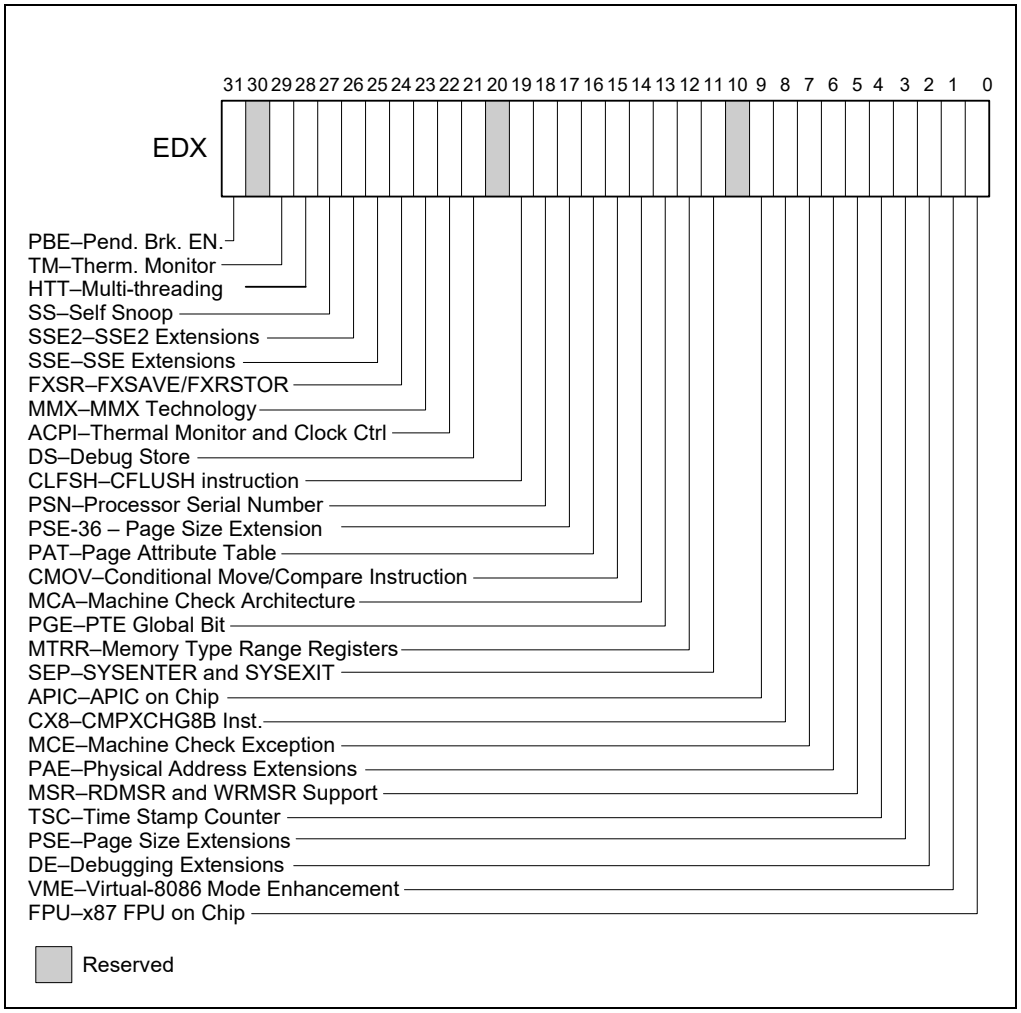


Figure 1-3. Feature Information Returned in the EDX Register

Table 1-6. 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 1-6. More on Feature Information Returned in the EDX Register(Continued)

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

Table 1-6. More on Feature Information Returned in the EDX Register(Continued)

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

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

When CPUID executes with EAX set to 02H, the processor returns information about the processor's internal caches and TLBs in the EAX, EBX, ECX, and EDX registers.

The encoding is as follows:

- The least-significant byte in register EAX (register AL) indicates the number of times the CPUID instruction must be executed with an input value of 02H to get a complete description of the processor's caches and TLBs. The first member of the family of Pentium 4 processors will return a 01H.
- 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. Table 1-7 shows 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 or TLB types. The descriptors may appear in any order.

Table 1-7. Encoding of Cache and TLB Descriptors

Descriptor Value	Cache or TLB Description
00H	Null descriptor
01H	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries
02H	Instruction TLB: 4 MByte pages, 4-way set associative, 2 entries
03H	Data TLB: 4 KByte pages, 4-way set associative, 64 entries
04H	Data TLB: 4 MByte pages, 4-way set associative, 8 entries
05H	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries
06H	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size
08H	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size

Table 1-7. Encoding of Cache and TLB Descriptors (Continued)

Descriptor Value	Cache or TLB Description
0AH	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size
0BH	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries
0CH	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size
22H	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector
23H	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
25H	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
29H	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
2CH	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size
30H	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size
40H	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache
41H	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size
42H	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size
43H	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size
44H	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size
45H	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size
46H	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size
47H	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size
49H	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	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size
4BH	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size
4CH	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size
4DH	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size
4EH	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size
50H	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries
51H	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries
52H	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries
56H	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries
57H	Data TLB0: 4 KByte pages, 4-way associative, 16 entries
5BH	Data TLB: 4 KByte and 4 MByte pages, 64 entries
5CH	Data TLB: 4 KByte and 4 MByte pages,128 entries
5DH	Data TLB: 4 KByte and 4 MByte pages,256 entries
60H	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size
66H	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size
67H	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size
68H	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size
70H	Trace cache: 12 K-µop, 8-way set associative
71H	Trace cache: 16 K-µop, 8-way set associative
72H	Trace cache: 32 K-µop, 8-way set associative
78H	2nd-level cache: 1 MByte, 4-way set associative, 64byte line size
79H	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector

Table 1-7. Encoding of Cache and TLB Descriptors (Continued)

Descriptor Value	Cache or TLB Description
7AH	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7BH	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7CH	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector
7DH	2nd-level cache: 2 MByte, 8-way set associative, 64byte line size
7FH	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size
82H	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size
83H	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size
84H	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size
B0H	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries
B1H	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries
B3H	Data TLB: 4 KByte pages, 4-way set associative, 128 entries
B4H	Data TLB1: 4 KByte pages, 4-way associative, 256 entries
F0H	64-Byte prefetching
F1H	128-Byte prefetching

Example 1-1. 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 indicates that CPUID needs to be executed once with an input value of 2 to retrieve complete information about caches and TLBs.
- 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 1-3.

The CPUID leaf 4 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=0H 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 1-3.

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 1-3.

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

When CPUID executes with EAX set to 07H and ECX = 0H, the processor returns information about the maximum number of sub-leaves that contain extended feature flags. See Table 1-3.

When CPUID executes with EAX set to 07H and ECX = n (n ≥ 1 and less than the number of non-zero bits in CPUID.(EAX=07H, ECX= 0H).EAX), the processor returns information about extended feature flags. See Table 1-3. In sub-leaf 0, only EAX has the number of sub-leaves. In sub-leaf 0, EBX, ECX & EDX all contain 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 1-3.

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 1-3) is greater than Pn 0. See Table 1-3.

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 17, "Debug, Branch Profile, TSC, and Quality of Service," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

INPUT EAX = 0BH: Returns Extended Topology Information

CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH.

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 1-3.

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

When CPUID executes with EAX set to 0DH and ECX = 0H, 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 1-3.

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 1-3. Software can use the forward-extendable technique depicted below to query the valid sub-leaves and obtain size and offset information for each processor extended state save area:

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

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

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

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

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

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

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

INPUT EAX = 12H: Returns Intel SGX Enumeration Information

When CPUID executes with EAX set to 12H and ECX = 0H, the processor returns information about Intel SGX capabilities. See Table 1-3.

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

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

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 1-3.

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

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

When CPUID executes with EAX set to 15H and ECX = 0H, the processor returns information about Time Stamp

Counter and Core Crystal Clock. See Table 1-3.

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 1-3.

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

When CPUID executes with EAX set to 17H, the processor returns information about the System-On-Chip Vendor Attribute Enumeration. See Table 1-3.

INPUT EAX = 18H: Returns Deterministic Address Translation Parameters Information

When CPUID executes with EAX set to 18H, the processor returns information about the Deterministic Address Translation Parameters. See Table 1-3.

INPUT EAX = 19H: Returns Key Locker Information

When CPUID executes with EAX set to 19H, the processor returns information about Key Locker. See Table 1-3.

INPUT EAX = 1AH: Returns Hybrid Information

When CPUID executes with EAX set to 1AH, the processor returns information about hybrid capabilities. See Table 1-3.

INPUT EAX = 1BH: Returns PCONFIG Information

When CPUID executes with EAX set to 1BH, the processor returns information about PCONFIG capabilities. See Table 1-3.

INPUT EAX = 1DH: Returns Tile Information

When CPUID executes with EAX set to 1DH and ECX = 0H, the processor returns information about tile architecture. See Table 1-3.

When CPUID executes with EAX set to 1DH and ECX = 1H, the processor returns information about tile palette 1. See Table 1-3.

INPUT EAX = 1EH: Returns TMUL Information

When CPUID executes with EAX set to 1EH and ECX = 0H, the processor returns information about TMUL capabilities. See Table 1-3.

INPUT EAX = 1FH: Returns V2 Extended Topology Information

When CPUID executes with EAX set to 1FH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 1FH by verifying (a) the highest leaf index supported by CPUID is >= 1FH, and (b) CPUID.1FH:EBX[15:0] reports a non-zero value. See Table 1-3.

INPUT EAX = 20H: Returns Processor History Reset Information

When CPUID executes with EAX set to 20H, the processor returns information about processor history reset. See Table 1-3.

METHODS FOR RETURNING BRANDING INFORMATION

Use the following techniques to access branding information:

1. Processor brand string method; this method also returns the processor’s maximum operating frequency
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 16 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

The Processor Brand String Method

Figure 1-4 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 maximum operating frequency of the processor to the EAX, EBX, ECX, and EDX registers.

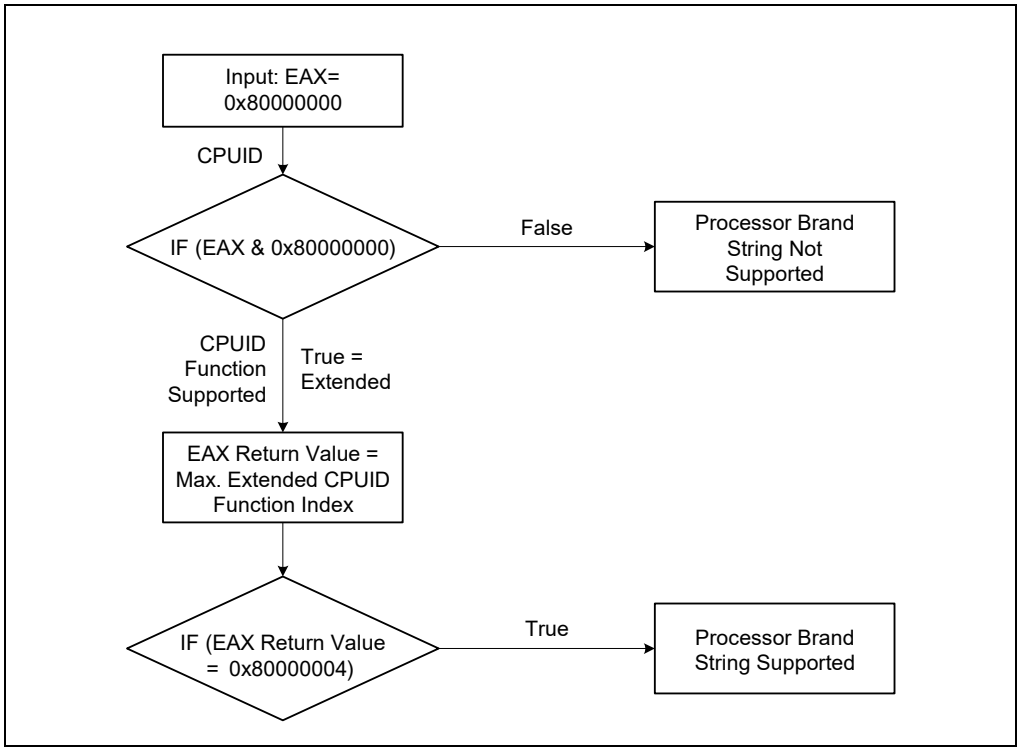


Figure 1-4. Determination of Support for the Processor Brand String

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 1-8 shows the brand string that is returned by the first processor in the Pentium 4 processor family.

Table 1-8. 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 1-8. Processor Brand String Returned with Pentium 4 Processor (Continued)

80000003H	EAX = 286C6574H EBX = 50202952H ECX = 69746E65H EDX = 52286D75H	"(let" "P)R" "itne" "R(mu"
80000004H	EAX = 20342029H EBX = 20555043H ECX = 30303531H EDX = 007A484DH	" 4)" " UPC" "0051" "\0zHM"

Extracting the Maximum Processor Frequency from Brand Strings

Figure 1-5 provides an algorithm which software can use to extract the maximum processor operating frequency from the processor brand string.

NOTE

When a frequency is given in a brand string, it is the maximum qualified frequency of the processor, not the frequency at which the processor is currently running.

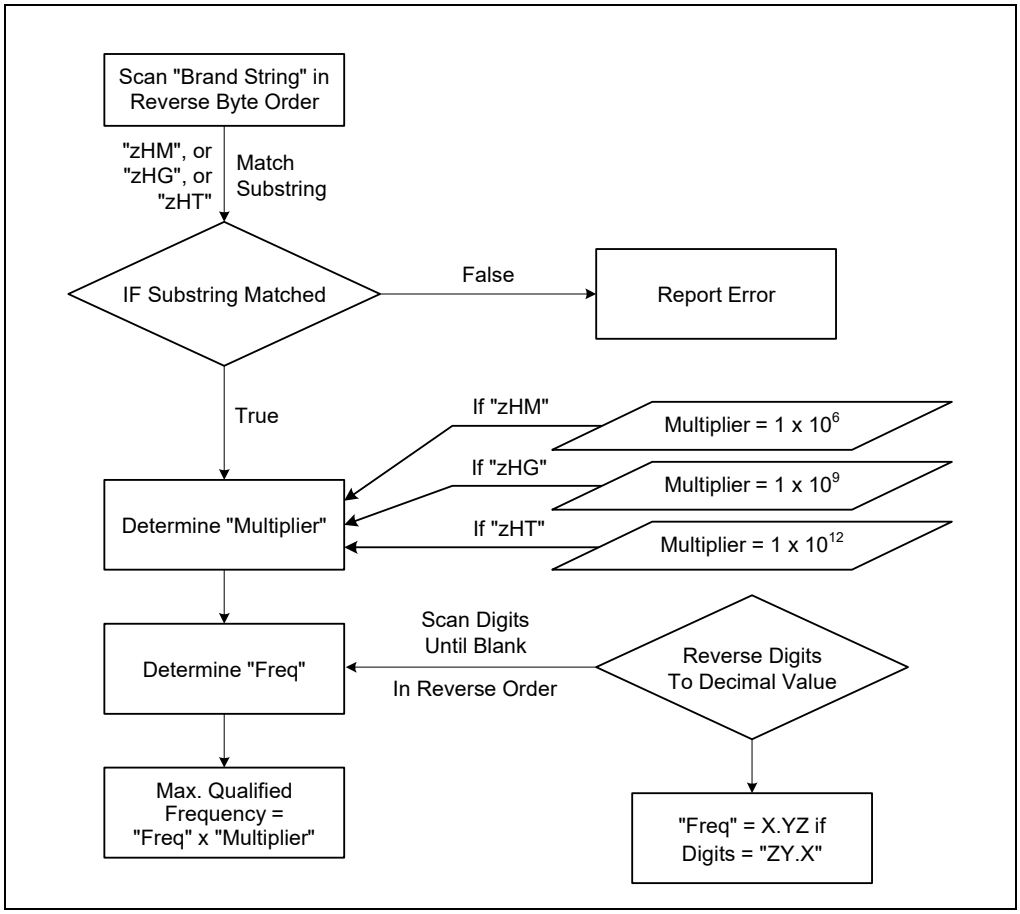


Figure 1-5. Algorithm for Extracting Maximum Processor Frequency

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 01H, 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 1-9 shows brand indices that have identification strings associated with them.

Table 1-9. 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 1-2. *)
        EDX := Feature flags; (* See Figure 1-3. *)
    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 1-3. *)
        EBX := Deterministic Cache Parameters Leaf;
        ECX := Deterministic Cache Parameters Leaf;
        EDX := Deterministic Cache Parameters Leaf;
    BREAK;
    EAX = 5H:
        EAX := MONITOR/MWAIT Leaf; (* See Table 1-3. *)
        EBX := MONITOR/MWAIT Leaf;
        ECX := MONITOR/MWAIT Leaf;
        EDX := MONITOR/MWAIT Leaf;
    BREAK;
    EAX = 6H:
        EAX := Thermal and Power Management Leaf; (* See Table 1-3. *)
```

```
        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 Leaf; (* See Table 1-3. *);
        EBX := Structured Extended Feature Leaf;
        ECX := Structured Extended Feature Leaf;
        EDX := Structured Extended Feature 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 1-3. *)
        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 1-3. *)
        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 1-3. *)
        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 1-3. *)
        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 1-3. *)
```

```
        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 1-3. *)
        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 = 12H:
        EAX := Intel SGX Enumeration Leaf; (* See Table 1-3. *)
        EBX := Intel SGX Enumeration Leaf;
        ECX := Intel SGX Enumeration Leaf;
        EDX := Intel SGX Enumeration Leaf;
    BREAK;
    EAX = 14H:
        EAX := Intel Processor Trace Enumeration Leaf; (* See Table 1-3. *)
        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 and Core Crystal Clock Information Leaf; (* See Table 1-3. *)
        EBX := Time Stamp Counter and Core Crystal Clock Information Leaf;
        ECX := Time Stamp Counter and Core Crystal Clock Information Leaf;
        EDX := Time Stamp Counter and Core Crystal Clock Information Leaf;
    BREAK;
    EAX = 16H:
        EAX := Processor Frequency Information Enumeration Leaf; (* See Table 1-3. *)
        EBX := Processor Frequency Information Enumeration Leaf;
        ECX := Processor Frequency Information Enumeration Leaf;
        EDX := Processor Frequency Information Enumeration Leaf;
    BREAK;
    EAX = 17H:
        EAX := System-On-Chip Vendor Attribute Enumeration Leaf; (* See Table 1-3. *)
        EBX := System-On-Chip Vendor Attribute Enumeration Leaf;
        ECX := System-On-Chip Vendor Attribute Enumeration Leaf;
        EDX := System-On-Chip Vendor Attribute Enumeration Leaf;
    BREAK;
    EAX = 18H:
        EAX := Deterministic Address Translation Parameters Enumeration Leaf; (* See Table 1-3. *)
        EBX := Deterministic Address Translation Parameters Enumeration Leaf;
        ECX :=Deterministic Address Translation Parameters Enumeration Leaf;
        EDX := Deterministic Address Translation Parameters Enumeration Leaf;
    BREAK;
    EAX = 19H:
        EAX := Key Locker Enumeration Leaf; (* See Table 1-3. *)
        EBX := Key Locker Enumeration Leaf;
        ECX := Key Locker Enumeration Leaf;
        EDX := Key Locker Enumeration Leaf;
    BREAK;
    EAX = 1AH:
        EAX := Hybrid Information Enumeration Leaf; (* See Table 1-3. *)
```

```
        EBX := Hybrid Information Enumeration Leaf;
        ECX := Hybrid Information Enumeration Leaf;
        EDX := Hybrid Information Enumeration Leaf;
    BREAK;
    EAX = 1BH:
        EAX := PCONFIG Information Enumeration Leaf; (* See Table 1-3. *)
        EBX := PCONFIG Information Enumeration Leaf;
        ECX := PCONFIG Information Enumeration Leaf;
        EDX := PCONFIG Information Enumeration Leaf;
    BREAK;
    EAX = 1DH:
        EAX := Tile Information Enumeration Leaf; (* See Table 1-3. *)
        EBX := Tile Information Enumeration Leaf;
        ECX := Tile Information Enumeration Leaf;
        EDX := Tile Information Enumeration Leaf;
    BREAK;
    EAX = 1EH:
        EAX := TMUL Information Enumeration Leaf; (* See Table 1-3. *)
        EBX := TMUL Information Enumeration Leaf;
        ECX := TMUL Information Enumeration Leaf;
        EDX := TMUL Information Enumeration Leaf;
    BREAK;
    EAX = 1FH:
        EAX := V2 Extended Topology Enumeration Leaf; (* See Table 1-3. *)
        EBX := V2 Extended Topology Enumeration Leaf;
        ECX := V2 Extended Topology Enumeration Leaf;
        EDX := V2 Extended Topology Enumeration Leaf;
    BREAK;
    EAX = 20H:
        EAX := Processor History Reset Enumeration Leaf; (* See Table 1-3. *)
        EBX := Processor History Reset Enumeration Leaf;
        ECX := Processor History Reset Enumeration Leaf;
        EDX := Processor History Reset Enumeration Leaf;
    BREAK;
    EAX = 80000000H:
        EAX := Highest extended function input value understood by CPUID;
        EBX := Reserved;
        ECX := Reserved;
        EDX := Reserved;
    BREAK;
    EAX = 80000001H:
        EAX := Reserved;
        EBX := Reserved;
        ECX := Extended Feature Bits (* See Table 1-3.*);
        EDX := Extended Feature Bits (* See Table 1-3. *);
    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 = 0;
    BREAK;
    EAX = 80000008H:
        EAX := Reserved = 0;
        EBX := Reserved = 0;
        ECX := Reserved = 0;
        EDX := Reserved = 0;
    BREAK;
    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.§

1.6 COMPRESSED DISPLACEMENT (DISP8*N) SUPPORT IN EVEX

For memory addressing using disp8 form, EVEX-encoded instructions always use a compressed displacement scheme by multiplying disp8 in conjunction with a scaling factor N that is determined based on the vector length, the value of EVEX.b bit (embedded broadcast) and the input element size of the instruction. In general, the factor N corresponds to the number of bytes characterizing the internal memory operation of the input operand (e.g., 64 when the accessing a full 512-bit memory vector). The scale factor N is listed in Table 1-10 and Table 1-11 below, where EVEX encoded instructions are classified using the **tupletype** attribute. The scale factor N of each tupletype is listed based on the vector length (VL) and other factors affecting it.

Table 1-10 covers EVEX-encoded instructions which has a load semantic in conjunction with additional computational or data element movement operation, operating either on the full vector or half vector (due to conversion of numerical precision from a wider format to narrower format). EVEX.b is supported for such instructions for data element sizes which are either dword or qword.

EVEX-encoded instruction that are pure load/store, and "Load+op" instruction semantic that operate on data element size less then dword do not support broadcasting using EVEX.b. These are listed in Table 1-11. Table 1-11 also includes many broadcast instructions which perform broadcast using a subset of data elements without using EVEX.b. These instructions and a few data element size conversion instruction are covered in Table 1-11. Instruction classified in Table 1-11 do not use EVEX.b and EVEX.b must be 0, otherwise #UD will occur.

The tupletype will be referenced in the instruction operand encoding table in the reference page of each instruction, providing the cross reference for the scaling factor N to encoding memory addressing operand.

Note that the disp8*N rules still apply when using 16b addressing.

Table 1-10. Compressed Displacement (DISP8*N) Affected by Embedded Broadcast

TupleType	EVEX.b	InputSize	EVEX.W	Broadcast	N (VL=128)	N (VL=256)	N (VL= 512)	Comment
Full	0	32bit	0	none	16	32	64	Load+Op (Full Vector Dword/Qword)
	1	32bit	0	{1tox}	4	4	4	
	0	64bit	1	none	16	32	64	
	1	64bit	1	{1tox}	8	8	8	
Half	0	32bit	0	none	8	16	32	Load+Op (Half Vector)
	1	32bit	0	{1tox}	4	4	4	

Table 1-11. EVEX DISP8*N for Instructions Not Affected by Embedded Broadcast

TupleType	InputSize	EVEX.W	N (VL= 128)	N (VL= 256)	N (VL= 512)	Comment
Full Mem	N/A	N/A	16	32	64	Load/store or subDword full vector
Tuple1 Scalar	8bit	N/A	1	1	1	1Tuple
	16bit	N/A	2	2	2	
	32bit	0	4	4	4	
	64bit	1	8	8	8	
Tuple1 Fixed	32bit	N/A	4	4	4	1 Tuple, memsize not affected by EVEX.W
	64bit	N/A	8	8	8	
Tuple1_4X	32bit	0	16[1]	N/A	16	4FMA(PS)
Tuple2	32bit	0	8	8	8	Broadcast (2 elements)
	64bit	1	NA	16	16	
Tuple4	32bit	0	NA	16	16	Broadcast (4 elements)
	64bit	1	NA	NA	32	
Tuple8	32bit	0	NA	NA	32	Broadcast (8 elements)

Table 1-11. EVEX DISP8*N for Instructions Not Affected by Embedded Broadcast(Continued)

TupleType	InputSize	EVEX.W	N (VL= 128)	N (VL= 256)	N (VL= 512)	Comment
Half Mem	N/A	N/A	8	16	32	SubQword Conversion
Quarter Mem	N/A	N/A	4	8	16	SubDword Conversion
Eighth Mem	N/A	N/A	2	4	8	SubWord Conversion
Mem128	N/A	N/A	16	16	16	Shift count from memory
MOVDDUP	N/A	N/A	8	32	64	VMOVDDUP

NOTES:

1. Scalar

1.7 BFLOAT16 FLOATING-POINT FORMAT

Intel® Deep Learning Boost (Intel® DL Boost) uses bfloat16 format (BF16). Figure 1-6 illustrates BF16 versus FP16 and FP32.

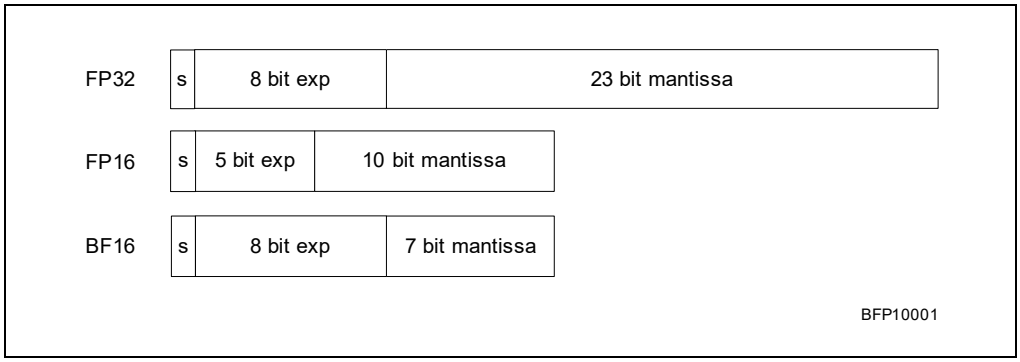


Figure 1-6. Comparison of BF16 to FP16 and FP32

BF16 has several advantages over FP16:

- It can be seen as a short version of FP32, skipping the least significant 16 bits of mantissa.
- There is no need to support denormals; FP32, and therefore also BF16, offer more than enough range for deep learning training tasks.
- FP32 accumulation after the multiply is essential to achieve sufficient numerical behavior on an application level.
- Hardware exception handling is not needed as this is a performance optimization; industry is designing algorithms around checking inf/NaN.

CHAPTER 2
INSTRUCTION SET REFERENCE, A-Z

Instructions described in this document follow the general documentation convention established in *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A*. Additionally, some instructions use notation conventions as described below.

In the instruction encoding, the MODRM byte is represented several ways depending on the role it plays. The MODRM byte has 3 fields: 2-bit MODRM.MOD field, a 3-bit MODRM.REG field and a 3-bit MODRM.RM field. When all bits of the MODRM byte have fixed values for an instruction, the 2-hex nibble value of that byte is presented after the opcode in the encoding boxes on the instruction description pages. When only some fields of the MODRM byte must contain fixed values, those values are specified as follows:

- If only the MODRM.MOD must be 0b11, and MODRM.REG and MODRM.RM fields are unrestricted, this is denoted as **11:rrr:bbb**. The **rrr** correspond to the 3-bits of the MODRM.REG field and the **bbb** correspond to the 3-bits of the MODMR.RM field.
- If the MODRM.MOD field is constrained to be a value other than 0b11, i.e., it must be one of 0b00, 0b01, or 0b10, then we use the notation !(11).
- If for example only the MODRM.REG field had a specific required value, e.g., 0b101, that would be denoted as mm:101:bbb.

NOTE

Historically the *Intel® 64 and IA-32 Architectures Software Developer's Manual* only specified the MODRM.REG field restrictions with the notation /0 ... /7 and did not specify restrictions on the MODRM.MOD and MODRM.RM fields in the encoding boxes.

2.1 INSTRUCTION SET REFERENCE

CLUI — Clear User Interrupt Flag

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 01 EE CLUI	ZO	V/I	UINTR	Clear user interrupt flag; user interrupts blocked when user interrupt flag cleared.

Instruction Operand Encoding

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

Description

CLUI clears the user interrupt flag (UIF). Its effect takes place immediately: a user interrupt cannot be delivered on the instruction boundary following CLUI.

An execution of CLUI inside a transactional region causes a transactional abort; the abort loads EAX as it would have had it been caused due to an execution of CLI.

Operation

UIF := 0;

Flags Affected

None.

Protected Mode Exceptions

#UD The CLUI instruction is not recognized in protected mode.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

#UD The CLUI instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If the LOCK prefix is used.
If executed inside an enclave.
If CR4.UINTR = 0.
If CPUID.07H.0H:EDX.UINTR[bit 5] = 0.

ENQCMD — Enqueue Command

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 0F 38 F8 !(11):rrr:bbb ENQCMD r32/r64, m512	A	V/V	ENQCMD	Atomically enqueue 64-byte user command with PASID from source memory operand to destination offset in ES segment specified in register operand as offset in ES segment.

Instruction Operand Encoding

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

Description

The ENQCMD instruction allows software to write commands to **enqueue registers**, which are special device registers accessed using memory-mapped I/O (MMIO).

Enqueue registers expect writes to have the following format:

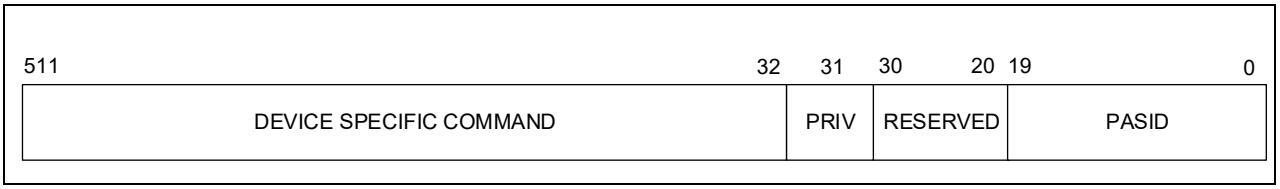


Figure 2-1. 64-Byte Data Written to Enqueue Registers

Bits 19:0 convey the process address space identifier (PASID), a value which system software may assign to individual software threads. Bit 31 contains privilege identification (0 = user; 1 = supervisor). Devices implementing enqueue registers may use these two values along with a device-specific command in the upper 60 bytes. Chapter 4 provides more details regarding how ENQCMD uses PASIDs.

The ENQCMD instruction begins by reading 64 bytes of command data from its source memory operand. This is an ordinary load with cacheability and memory ordering implied normally by the memory type. The source operand need not be aligned, and there is no guarantee that all 64 bytes are loaded atomically.

The instruction then formats those 64 bytes into **command data** with a format consistent with that given in Figure 2-1:

- Command[19:0] get IA32_PASID[19:0].[1]
- Command[30:20] are zero.
- Command[31] is 0 (indicating user).
- Command[511:32] get bits 511:32 of the source operand that was read from memory.

(The instruction ignores bits 31:0 of the source operand.)

The ENQCMD instruction uses an **enqueue store** (defined below) to write this command data to the destination operand. The address of the destination operand is specified in a general-purpose register as an offset into the ES segment (the segment cannot be overridden).[2] The destination linear address must be 64-byte aligned. The operation of an enqueue store disregards the memory type of the destination memory address.

1. It is expected that system software will load the IA32_PASID MSR so that bits 19:0 contain the PASID of the current software thread. The MSR's valid bit, IA32_PASID[31], must be 1. The PASID MSR is discussed in more detail in Section 4.1.
2. In 64-bit mode, the width of the register operand is 64 bits (32 bits with a 67H prefix). Outside 64-bit mode when CS.D = 1, the width is 32 bits (16 bits with a 67H prefix). Outside 64-bit mode when CS.D=0, the width is 16 bits (32 bits with a 67H prefix).

An enqueue store is not ordered relative to older stores to WB or WC memory (including non-temporal stores) or to executions of the CLFLUSHOPT or CLWB (when applied to addresses other than that of the enqueue store). Software can enforce such ordering by executing a fencing instruction such as SFENCE or MFENCE before the enqueue store.

An enqueue store does not write the data into the cache hierarchy, nor does it fetch any data into the cache hierarchy. An enqueue store's command data is never combined with that of any other store to the same address.

Unlike other stores, an enqueue store returns a status, which the ENQCMD instruction loads into the ZF flag in the RFLAGS register:

- ZF = 0 (success) reports that the 64-byte command data was written atomically to a device's enqueue register and has been accepted by the device. (It does not guarantee that the device has acted on the command; it may have queued it for later execution.)
- ZF = 1 (retry) reports that the command data was not accepted. This status is returned if the destination address is an enqueue register but the command was not accepted due to capacity or other temporal reasons. This status is also returned if the destination address was not an enqueue register (including the case of a memory address); in these cases, the store is dropped and is written neither to MMIO nor to memory.

Availability of the ENQCMD instruction is indicated by the presence of the CPUID feature flag ENQCMD (CPUID.(EAX=07H, ECX=0H):ECX[bit 29]).

Operation

```
IF IA32_PASID[31] = 0
    THEN #GP;
ELSE
    COMMAND := (SRC & ~FFFFFFFFH) | (IA32_PASID & FFFFFH);
    DEST := COMMAND;
FI;
```

Intel C/C++ Compiler Intrinsic Equivalent

```
ENQCMD int_enqcmd(void *dst, const void *src)
```

Flags Affected

The ZF flag is set if the enqueue-store completion returns the retry status; otherwise it is cleared. All other flags are cleared.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If destination linear address is not aligned to a 64-byte boundary.
	If the PASID Valid field (bit 31) is 0 in IA32_PASID MSR.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#UD	If CPUID.07H.0H:ECX.ENQCMD[bit 29] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	If destination linear address is not aligned to a 64-byte boundary.
	If the PASID Valid field (bit 31) is 0 in IA32_PASID MSR.
#UD	If CPUID.07H.0H:ECX.ENQCMD[bit 29] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real-address mode. Additionally:

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in non-canonical form.
#GP(0)	If the memory address is in non-canonical form.
	If destination linear address is not aligned to a 64-byte boundary.
	If the PASID Valid field (bit 31) is 0 in IA32_PASID MSR.
#PF(fault-code)	For a page fault.
#UD	If CPUID.07H.0H:ECX.ENQCMD[bit 29].
	If the LOCK prefix is used.

ENQCMDS — Enqueue Command Supervisor

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 38 F8 !(11):rrr:bbb ENQCMDS r32/r64, m512	A	V/V	ENQCMD	Atomically enqueue 64-byte command from source memory operand to destination offset in ES segment specified in register operand as offset in ES segment.

Instruction Operand Encoding

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

Description

The ENQCMDS instruction allows system software to write commands to **enqueue registers**, which are special device registers accessed using memory-mapped I/O (MMIO).

Enqueue registers expect writes to have the format given in Figure 2-1 and explained in the section on "ENQCMD — Enqueue Command."

The ENQCMDS instruction begins by reading 64 bytes of command data from its source memory operand. This is an ordinary load with cacheability and memory ordering implied normally by the memory type. The source operand need not be aligned, and there is no guarantee that all 64 bytes are loaded atomically.

ENQCMDS formats its source data differently from ENQCMD. Specifically, it formats them into **command data** as follows:

- Command[19:0] get bits 19:0 of the source operand that was read from memory. These 20 bits communicate a process address-space identifier (PASID). Chapter 4 provides more details regarding how ENQCMDS uses PASIDs.
- Command[30:20] are zero.
- Command[511:31] get bits 511:31 of the source operand that was read from memory. Bit 31 communicates a privilege identification (0 = user; 1 = supervisor).

(The instruction ignores bits 30:20 of the source operand.)

The ENQCMDS instruction then uses an **enqueue store** (defined below) to write this command data to the destination operand. The address of the destination operand is specified in a general-purpose register as an offset into the ES segment (the segment cannot be overridden).[1] The destination linear address must be 64-byte aligned. The operation of an enqueue store disregards the memory type of the destination memory address.

An enqueue store is not ordered relative to older stores to WB or WC memory (including non-temporal stores) or to executions of the CLFLUSHOPT or CLWB (when applied to addresses other than that of the enqueue store). Software can enforce such ordering by executing a fencing instruction such as SFENCE or MFENCE before the enqueue store.

An enqueue store does not write the data into the cache hierarchy, nor does it fetch any data into the cache hierarchy. An enqueue store's command data is never combined with that of any other store to the same address.

Unlike other stores, an enqueue store returns a status, which the ENQCMDS instruction loads into the ZF flag in the RFLAGS register:

- ZF = 0 (success) reports that the 64-byte command data was written atomically to a device's enqueue register and has been accepted by the device. (It does not guarantee that the device has acted on the command; it may have queued it for later execution.)
- ZF = 1 (retry) reports that the command data was not accepted. This status is returned if the destination address is an enqueue register but the command was not accepted due to capacity or other temporal reasons.

1. In 64-bit mode, the width of the register operand is 64 bits (32 bits with a 67H prefix). Outside 64-bit mode when CS.D = 1, the width is 32 bits (16 bits with a 67H prefix). Outside 64-bit mode when CS.D=0, the width is 16 bits (32 bits with a 67H prefix).

This status is also returned if the destination address was not an enqueue register (including the case of a memory address); in these cases, the store is dropped and is written neither to MMIO nor to memory.

The ENQCMDS instruction may be executed only if CPL = 0. Availability of the ENQCMDS instruction is indicated by the presence of the CPUID feature flag ENQCMD (CPUID.(EAX=07H, ECX=0H):ECX[bit 29]).

Operation

```
DEST := SRC & ~7FF00000H;          // clear bits 30:20
```

Intel C/C++ Compiler Intrinsic Equivalent

```
ENQCMDS int_enqcmds(void *dst, const void *src)
```

Flags Affected

The ZF flag is set if the enqueue-store completion returns the retry status; otherwise it is cleared. All other flags are cleared.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If destination linear address is not aligned to a 64-byte boundary.
	If the current privilege level is not 0.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#UD	If CPUID.07H.0H:ECX.ENQCMD[bit 29] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	If destination linear address is not aligned to a 64-byte boundary.
#UD	If CPUID.07H.0H:ECX.ENQCMD[bit 29] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	The ENQCMDS instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in non-canonical form.
#GP(0)	If the memory address is in non-canonical form.
	If destination linear address is not aligned to a 64-byte boundary.
	If the current privilege level is not 0.
#PF(fault-code)	For a page fault.
#UD	If CPUID.07H.0H:ECX.ENQCMD[bit 29].
	If the LOCK prefix is used.

HRESET — History Reset

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 3A F0 C0 /ib HRESET imm8, <EAX>	A	V/V	HRESET	Processor history reset request. Controlled by the EAX implicit operand.

Instruction Operand Encoding

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

Description

Provides a hint to the processor to selectively reset the prediction history of the current logical processor. HRESET operation is controlled by the implicit EAX operand. The value of the explicit imm8 operand is ignored.

CPUID.07H.01H:EAX.HRESET[bit 22] indicates support of the HRESET instruction. This instruction can only be executed at CPL 0.

The HRESET instruction is capable of providing a reset hint for multiple predictions. Prior to the execution of HRESET, the system software must take the following steps:

1. Enumerate the HRESET capabilities via CPUID.20H.0H:EBX, which indicates what predictions can be reset.
2. Opt-in to reset a subset of the available capabilities by setting the respective bits in the IA32_HRESET_ENABLE MSR. The opt-in bits in the IA32_HRESET_ENABLE MSR are aligned with the HRESET capabilities CPUID bits.

The implicit EAX operand must contain set bits that are a subset of those set in the IA32_HRESET_ENABLE MSR. Otherwise, HRESET generates #GP(0). When EAX=0 this instruction is interpreted as NOP.

Any attempt to execute the HRESET instruction inside a transactional region will result in a transaction abort.

Operation

```
IF EAX = 0
   THEN NOP
   ELSE
      FOREACH i such that EAX[i] = 1
         Reset prediction history for feature i
FI
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If CPL > 0 or (EAX AND NOT IA32_HRESET_ENABLE) ≠0.
#UD	If CPUID.07H.01H:EAX.HRESET[bit 22] = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

#GP(0)	HRESET instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

PCONFIG — Platform Configuration

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

Instruction Operand Encoding

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

Description

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

Each PCONFIG leaf function applies to a specific hardware block called a PCONFIG target, and each PCONFIG target is associated with a numerical identifier. The identifiers of the PCONFIG targets supported by the CPU (which imply the supported leaf functions) are enumerated in the sub-leaves of the PCONFIG-information leaf of CPUID (EAX = 1BH). An attempt to execute an undefined leaf function results in a general-protection exception (#GP).

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

Table 2-1 shows the leaf encodings for PCONFIG.

Table 2-1. PCONFIG Leaf Encodings

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

The MKTME_KEY_PROGRAM leaf of PCONFIG pertains to the MKTME target, which has target identifier 1. It is used by software to manage the key associated with a KeyID. The leaf function is invoked by setting the leaf value of 0 in EAX and the address of MKTME_KEY_PROGRAM_STRUCT in RBX. Successful execution of the leaf clears RAX (set to zero) and ZF, CF, PF, AF, OF, and SF are cleared. In case of failure, the failure reason is indicated in RAX with ZF set to 1 and CF, PF, AF, OF, and SF are cleared. The MKTME_KEY_PROGRAM leaf uses the MKTME_KEY_PROGRAM_-STRUCT in memory shown in Table 2-2.

Table 2-2. MKTME_KEY_PROGRAM_STRUCT Format

Field	Offset (bytes)	Size (bytes)	Comments
KEYID	0	2	Key Identifier.
KEYID_CTRL	2	4	KeyID control: ▪ Bits [7:0]: COMMAND. ▪ Bits [23:8]: ENC_ALG. ▪ Bits [31:24]: Reserved, must be zero.
RESERVED	6	58	Reserved, must be zero.
KEY_FIELD_1	64	64	Software supplied KeyID data key or entropy for KeyID data key.
KEY_FIELD_2	128	64	Software supplied KeyID tweak key or entropy for KeyID tweak key.

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

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

  The command used controls the encryption mode for a KeyID. Table 2-3 provides a summary of the commands supported.

Table 2-3. Supported Key Programming Commands

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

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

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

  All KeyIDs use the TME key on MKTME activation. Software can at any point decide to change the key for a KeyID using the PCONFIG instruction. Change of keys for a KeyID does NOT change the state of the TLB caches or memory pipeline. It is software's responsibility to take appropriate actions to ensure correct behavior.

  Table 2-4 shows the return values associated with the MKTME_KEY_PROGRAM leaf of PCONFIG. On instruction execution, RAX is populated with the return value.

Table 2-4. Supported Key Error Codes

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

PCONFIG Virtualization

Software in VMX root operation can control the execution of PCONFIG in VMX non-root operation using the following VM-execution controls introduced for PCONFIG:

- PCONFIG_ENABLE: This control is a single bit control and enables the PCONFIG instruction in VMX non-root operation. If 0, the execution of PCONFIG in VMX non-root operation causes #UD. Otherwise, execution of PCONFIG works according to PCONFIG_EXITING.
- PCONFIG_EXITING: This is a 64b control and allows VMX root operation to cause a VM-exit for various leaf functions of PCONFIG. This control does not have any effect if the PCONFIG_ENABLE control is clear. It is recommended that VMMs intercept execution of any PCONFIG leaves with which they are not familiar and convert such executions into #GP(0).

PCONFIG Concurrency

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

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

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

Operation

Table 2-5. PCONFIG Operation Variables

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

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

IF (in VMX non-root mode)
{
    IF (VMCS.PCONFIG_ENABLE == 1)
    {
        IF ((EAX > 62 AND VMCS.PCONFIG_EXITING[63] ==1) OR
            (EAX < 63 AND VMCS.PCONFIG_EXITING[EAX] == 1))
        {
            Set VMCS.EXIT_REASON = PCONFIG; //No Exit qualification
            Deliver VMEXIT;
        }
    }
    ELSE
    {
        #UD
    }
}

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

(* KEY_PROGRAM leaf flow *)
IF (EAX == 0)
{
    (* #GP(0) if TME_ACTIVATE MSR is not locked or does not enable TME or multiple keys are not enabled *)
    IF (IA32_TME_ACTIVATE.LOCK != 1 OR IA32_TME_ACTIVATE.ENABLE != 1 OR IA32_TME_ACTIVATE.MK_TME_KEYID_BITS == 0)
#GP(0)

    (* Check MKTME_KEY_PROGRAM_STRUCT is 256B aligned *)
    IF (DS:RBX is not 256B aligned) #GP(0);

    (* Check that MKTME_KEY_PROGRAM_STRUCT is read accessible *)
    <<DS: RBX should be read accessible>>

    (* Copy MKTME_KEY_PROGRAM_STRUCT to a temporary variable *)
    TMP_KEY_PROGRAM_STRUCT = DS:RBX.*;

    (* RSVD field check *)
    IF (TMP_KEY_PROGRAM_STRUCT.RSVD != 0) #GP(0);

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

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

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

    (* Check for a valid command *)
    IF (TMP_KEY_PROGRAM_STRUCT. KEYID_CTRL.COMMAND is not a valid command)
    {
        RFLAGS.ZF = 1;
        RAX = INVALID_PROG_CMD;
        goto EXIT;
```

```
}
(* Check that the KEYID being operated upon is a valid KEYID *)
IF (TMP_KEY_PROGRAM_STRUCT.KEYID >
                2^IA32_TME_ACTIVATE.MK_TME_KEYID_BITS - 1
    OR TMP_KEY_PROGRAM_STRUCT.KEYID >
                IA32_TME_CAPABILITY.MK_TME_MAX_KEYS
    OR TMP_KEY_PROGRAM_STRUCT.KEYID == 0)
{
    RFLAGS.ZF = 1;
    RAX = INVALID_KEYID;
    goto EXIT;
}

(* Check that only one algorithm is requested for the KeyID and it is one of the activated algorithms *)
IF (NUM_BITS(TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG) != 1 ||
    (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG &
        IA32_TME_ACTIVATE. MK_TME_CRYPTO_ALGS == 0))
{
    RFLAGS.ZF = 1;
    RAX = INVALID_ENC_ALG;
    goto EXIT;
}
(* Try to acquire exclusive lock *)
IF (NOT KEY_TABLE_LOCK.ACQUIRE(WRITE))
{
    //PCONFIG failure
    RFLAGS.ZF = 1;
    RAX = DEVICE_BUSY;
    goto EXIT;
}

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

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

case KEYID_SET_KEY_RANDOM:
    TMP_RND_DATA_KEY = <<Generate a random key using hardware RNG>>
    IF (NOT ENOUGH ENTROPY)
    {
        RFLAGS.ZF = 1;
        RAX = ENTROPY_ERROR;
        goto EXIT;
    }
    TMP_RND_TWEAK_KEY = <<Generate a random key using hardware RNG>>
```

```
        IF (NOT ENOUGH ENTROPY)
        {
            RFLAGS.ZF = 1;
            RAX = ENTROPY_ERROR;
            goto EXIT;
        }
        (* Mix user supplied entropy to the data key and tweak key *)
        TMP_RND_DATA_KEY = TMP_RND_KEY XOR
            TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1.BYTES[15:0];
        TMP_RND_TWEAK_KEY = TMP_RND_TWEAK_KEY XOR
            TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2.BYTES[15:0];

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

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

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

    RAX = 0;
    RFLAGS.ZF = 0;

    //Release Lock
    KEY_TABLE_LOCK(RELEASE);

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

end_of_flow
```

Protected Mode Exceptions

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

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

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

SENDUIPI — Send User Interprocessor Interrupts

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F C7 /6 SENDUIPI reg	A	V/I	UINTR	Send interprocessor user interrupt.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (r, w)	NA	NA	NA

Description

The SENDUIPI instruction takes a single register operand. The operand always has 64 bits; operand-size overrides (e.g., the prefix 66) are ignored.

Although SENDUIPI may be executed at any privilege level, all of the instruction's memory accesses are performed with supervisor privilege.

Virtualization of the SENDUIPI instruction (in particular, that of the sending of the notification interrupt) is discussed in Section 11.9.2.5.

The Operation section refers to the values UITTADDR and UITTSZ. The values are defined in Section 11.3.1. It also includes operations on a user posted-interrupt descriptor (UPID). The format of a UPID is defined in Section 11.5.

Operation

```
IF reg > UITTSZ;
    THEN #GP(0);
FI;
read tempUITTE from 16 bytes at UITTADDR+ (reg « 4);
IF tempUITTE.V = 0 or tempUITTE sets any reserved bit (see Section 11.7.1)
    THEN #GP(0);
FI;
read tempUPID from 16 bytes at tempUITTE.UPIDADDR;// under lock
IF tempUPID sets any reserved bits or bits that must be zero (see Table 11-1)
    THEN #GP(0); // release lock
FI;
tempUPID.PIR[tempUITTE.UV] := 1;
IF tempUPID.SN = tempUPID.ON = 0
    THEN
        tempUPID.ON := 1;
        sendNotify := 1;
    ELSE sendNotify := 0;
FI;
write tempUPID to 16 bytes at tempUITTE.UPIDADDR;// release lock
IF sendNotify = 1
    THEN
        IF local APIC is in x2APIC mode
            THEN send ordinary IPI with vector tempUPID.NV
                to 32-bit physical APIC ID tempUPID.NDST;
            ELSE send ordinary IPI with vector tempUPID.NV
                to 8-bit physical APIC ID tempUPID.NDST[15:8];
        FI;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#UD	The SENDUIPI instruction is not recognized in protected mode.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

#UD	The SENDUIPI instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD	If the LOCK prefix is used.
	If executed inside an enclave.
	If CR4.UINTR = 0.
	If CPUID.07H.0H:EDX.UINTR[bit 5] = 0.
#PF	If a page fault occurs.
#GP	If the value of the register operand exceeds UITTSZ.
	If the selected UITTE is not valid or sets any reserved bits.
	If the selected UPID sets any reserved bits.
	If there is an attempt to access memory using a linear address that is not canonical relative to the current paging mode.

SERIALIZE — Serialize Instruction Execution

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 E8 SERIALIZE	ZO	V/V	SERIALIZE	Serialize instruction fetch and execution.

Instruction Operand Encoding

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

Description

Serializes instruction execution. Before the next instruction is fetched and executed, the SERIALIZE instruction ensures that all modifications to flags, registers, and memory by previous instructions are completed, draining all buffered writes to memory. This instruction is also a serializing instruction as defined in the section "Serializing Instructions" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

SERIALIZE does not modify registers, arithmetic flags or memory.

The availability of the SERIALIZE instruction is indicated by the presence of the CPUID feature flag SERIALIZE, bit 14 of the EDX register in sub-leaf CPUID:7H.0H.

Operation

```
Wait_On_Fetch_And_Execution_Of_Next_Instruction_Until(preceding_instructions_complete_and_preceding_stores_globally_visible);
```

Intel C/C++ Compiler Intrinsic Equivalent

```
SERIALIZE void _serialize(void);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

#UD If the LOCK prefix is used.

STUI — Set User Interrupt Flag

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 01 EF STUI	ZO	V/I	UINTR	Set user interrupt flag.

Instruction Operand Encoding

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

Description

STUI sets the user interrupt flag (UIF). Its effect takes place immediately; a user interrupt may be delivered on the instruction boundary following STUI. (This is in contrast with STI, whose effect is delayed by one instruction).

An execution of STUI inside a transactional region causes a transactional abort; the abort loads EAX as it would have had it been due to an execution of STI.

Operation

UIF := 1;

Flags Affected

None.

Protected Mode Exceptions

#UD The STUI instruction is not recognized in protected mode.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

#UD The STUI instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If the LOCK prefix is used.
If executed inside an enclave.
If CR4.UINTR = 0.
If CPUID.07H.0H:EDX.UINTR[bit 5] = 0.

TESTUI — Determine User Interrupt Flag

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 01 ED TESTUI	ZO	V/I	UINTR	Copies the current value of UIF into EFLAGS.CF.

Instruction Operand Encoding

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

TESTUI copies the current value of the user interrupt flag (UIF) into EFLAGS.CF. This instruction can be executed regardless of CPL.

TESTUI may be executed normally inside a transactional region.

Operation

CF := UIF;
ZF := AF := OF := PF := SF := 0;

Flags Affected

The ZF, OF, AF, PF, SF flags are cleared and the CF flags to the value of the user interrupt flag.

Protected Mode Exceptions

#UD The TESTUI instruction is not recognized in protected mode.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

#UD The TESTUI instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If the LOCK prefix is used.
If executed inside an enclave.
If CR4.UINTR = 0.
If CPUID.07H.0H:EDX.UINTR[bit 5] = 0.

UIRET — User-Interrupt Return

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 01 EC UIRET	ZO	V/I	UINTR	Return from handling a user interrupt.

Instruction Operand Encoding

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

Description

UIRET returns from the handling of a user interrupt. It can be executed regardless of CPL.

Execution of UIRET inside a transactional region causes a transactional abort; the abort loads EAX as it would have had it been due to an execution of IRET.

UIRET can be tracked by Architectural Last Branch Records (LBRs), Intel Processor Trace (Intel PT), and Performance Monitoring. For both Intel PT and LBRs, UIRET is recorded in precisely the same manner as IRET. Hence for LBRs, UIRETs fall into the OTHER_BRANCH category, which implies that IA32_LBR_CTL.OTHER_BRANCH[bit 22] must be set to record user-interrupt delivery, and that the IA32_LBR_x_INFO.BR_TYPE field will indicate OTHER_BRANCH for any recorded user interrupt. For Intel PT, control flow tracing must be enabled by setting IA32_RTIT_CTL.BranchEn[bit 13].

UIRET will also increment performance counters for which counting BR_INST_RETIRED.FAR_BRANCH is enabled.

Operation

```
Pop tempRIP;
Pop tempRFLAGS; // see below for how this is used to load RFLAGS
Pop tempRSP;
IF tempRIP is not canonical in current paging mode
    THEN #GP(0);
FI;
IF shadow stack is enabled for CPL = 3
    THEN
        PopShadowStack SSRIP;
        IF SSRIP ≠ tempRIP
            THEN #CP (FAR-RET/IRET);
        FI;
FI;
RIP := tempRIP;
// update in RFLAGS only CF, PF, AF, ZF, SF, TF, DF, OF, NT, RF, AC, and ID
RFLAGS := (RFLAGS & ~254DD5H) | (tempRFLAGS & 254DD5H);
RSP := tempRSP;
UIF := 1;
Clear any cache-line monitoring established by MONITOR or UMONITOR;
```

Flags Affected

See Operation section.

Protected Mode Exceptions

#UD	The UIRET instruction is not recognized in protected mode.

Real-Address Mode Exceptions

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

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

#UD	The UIRET instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the return instruction pointer is non-canonical.
#SS(0)	If an attempt to pop a value off the stack causes a non-canonical address to be referenced.
#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.
#CP	If return instruction pointer from stack and shadow stack do not match.
#UD	If the LOCK prefix is used.
	If executed inside an enclave.
	If CR4.UINTR = 0.
	If CPUID.07H.0H:EDX.UINTR[bit 5] = 0.

VPDPBUSD — Multiply and Add Unsigned and Signed Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 50 /r VPDPBUSD xmm1, xmm2, xmm3/m128	A	V/V	AVX_VNNI	Multiply groups of 4 pairs of signed bytes in xmm3/m128 with corresponding unsigned bytes of xmm2, summing those products and adding them to doubleword result in xmm1.
VEX.256.66.0F38.W0 50 /r VPDPBUSD ymm1, ymm2, ymm3/m256	A	V/V	AVX_VNNI	Multiply groups of 4 pairs of signed bytes in ymm3/m256 with corresponding unsigned bytes of ymm2, summing those products and adding them to doubleword result in ymm1.

Instruction Operand Encoding

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

Description

Multiplies the individual unsigned bytes of the first source operand by the corresponding signed bytes of the second source operand, producing intermediate signed word results. The word results are then summed and accumulated in the destination dword element size operand.

This instruction supports memory fault suppression.

Operation

VPDPBUSD dest, src1, src2

```
VL=(128, 256)
KL=VL/32

ORIGDEST := DEST
FOR i := 0 TO KL-1:

    // Extending to 16b
    // src1extend := ZERO_EXTEND
    // src2extend := SIGN_EXTEND

    p1word := src1extend(SRC1.byte[4*i+0]) * src2extend(SRC2.byte[4*i+0])
    p2word := src1extend(SRC1.byte[4*i+1]) * src2extend(SRC2.byte[4*i+1])
    p3word := src1extend(SRC1.byte[4*i+2]) * src2extend(SRC2.byte[4*i+2])
    p4word := src1extend(SRC1.byte[4*i+3]) * src2extend(SRC2.byte[4*i+3])

    DEST.dword[i] := ORIGDEST.dword[i] + p1word + p2word + p3word + p4word

DEST[MAX_VL-1:VL] := 0
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4.

VPDPBUSDS — Multiply and Add Unsigned and Signed Bytes with Saturation

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 51 /r VPDPBUSDS xmm1, xmm2, xmm3/m128	A	V/V	AVX_VNNI	Multiply groups of 4 pairs signed bytes in xmm3/m128 with corresponding unsigned bytes of xmm2, summing those products and adding them to doubleword result, with signed saturation in xmm1.
VEX.256.66.0F38.W0 51 /r VPDPBUSDS ymm1, ymm2, ymm3/m256	A	V/V	AVX_VNNI	Multiply groups of 4 pairs signed bytes in ymm3/m256 with corresponding unsigned bytes of ymm2, summing those products and adding them to doubleword result, with signed saturation in ymm1.

Instruction Operand Encoding

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

Description

Multiplies the individual unsigned bytes of the first source operand by the corresponding signed bytes of the second source operand, producing intermediate signed word results. The word results are then summed and accumulated in the destination dword element size operand. If the intermediate sum overflows a 32b signed number the result is saturated to either 0x7FFF_FFFF for positive numbers of 0x8000_0000 for negative numbers.

This instruction supports memory fault suppression.

Operation

VPDPBUSDS dest, src1, src2

```
VL=(128, 256)
KL=VL/32

ORIGDEST := DEST
FOR i := 0 TO KL-1:

    // Extending to 16b
    // src1extend := ZERO_EXTEND
    // src2extend := SIGN_EXTEND

    p1word := src1extend(SRC1.byte[4*i+0]) * src2extend(SRC2.byte[4*i+0])
    p2word := src1extend(SRC1.byte[4*i+1]) * src2extend(SRC2.byte[4*i+1])
    p3word := src1extend(SRC1.byte[4*i+2]) * src2extend(SRC2.byte[4*i+2])
    p4word := src1extend(SRC1.byte[4*i+3]) * src2extend(SRC2.byte[4*i+3])

    DEST.dword[i] := SIGNED_DWORD_SATURATE(ORIGDEST.dword[i] + p1word + p2word + p3word + p4word)

DEST[MAX_VL-1:VL] := 0
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4.

VPDPWSSD — Multiply and Add Signed Word Integers

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 52 /r VPDPWSSD xmm1, xmm2, xmm3/m128	A	V/V	AVX_VNNI	Multiply groups of 2 pairs signed words in xmm3/m128 with corresponding signed words of xmm2, summing those products and adding them to doubleword result in xmm1.
VEX.256.66.0F38.W0 52 /r VPDPWSSD ymm1, ymm2, ymm3/m256	A	V/V	AVX_VNNI	Multiply groups of 2 pairs signed words in ymm3/m256 with corresponding signed words of ymm2, summing those products and adding them to doubleword result in ymm1.

Instruction Operand Encoding

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

Description

Multiplies the individual signed words of the first source operand by the corresponding signed words of the second source operand, producing intermediate signed, doubleword results. The adjacent doubleword results are then summed and accumulated in the destination operand.

This instruction supports memory fault suppression.

Operation

VPDPWSSD dest, src1, src2

```
VL=(128, 256)
KL=VL/32
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    p1dword := SRC1.word[2*i+0] * t.word[2*i+0]
    p2dword := SRC1.word[2*i+1] * t.word[2*i+1]
    DEST.dword[i] := ORIGDEST.dword[i] + p1dword + p2dword
DEST[MAX_VL-1:VL] := 0
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4.

VPDPWSSDS — Multiply and Add Signed Word Integers with Saturation

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 53 /r VPDPWSSDS xmm1, xmm2, xmm3/m128	A	V/V	AVX_VNNI	Multiply groups of 2 pairs of signed words in xmm3/m128 with corresponding signed words of xmm2, summing those products and adding them to doubleword result in xmm1, with signed saturation,.
VEX.256.66.0F38.W0 53 /r VPDPWSSDS ymm1, ymm2, ymm3/m256	A	V/V	AVX_VNNI	Multiply groups of 2 pairs of signed words in ymm3/m256 with corresponding signed words of ymm2, summing those products and adding them to doubleword result in ymm1, with signed saturation.

Instruction Operand Encoding

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

Description

Multiplies the individual signed words of the first source operand by the corresponding signed words of the second source operand, producing intermediate signed, doubleword results. The adjacent doubleword results are then summed and accumulated in the destination operand. If the intermediate sum overflows a 32b signed number, the result is saturated to either 0x7FFF_FFFF for positive numbers of 0x8000_0000 for negative numbers.

This instruction supports memory fault suppression.

Operation

VPDPWSSDS dest, src1, src2

```
VL=(128, 256)
KL=VL/32
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    p1dword := SRC1.word[2*i+0] * t.word[2*i+0]
    p2dword := SRC1.word[2*i+1] * t.word[2*i+1]
    DEST.dword[i] := SIGNED_DWORD_SATURATE(ORIGDEST.dword[i] + p1dword + p2dword)
DEST[MAX_VL-1:VL] := 0
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

See Exceptions Type 4.

XRESLDTRK — Resume Tracking Load Addresses

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 0F 01 E9 XRESLDTRK	ZO	V/V	TSXLDTRK	Specifies the end of an Intel TSX suspend read address tracking region.

Instruction Operand Encoding

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

Description

The instruction marks the end of an Intel TSX (RTM) suspend load address tracking region. If the instruction is used inside a suspend load address tracking region it will end the suspend region and all following load addresses will be added to the transaction read set. If this instruction is used inside an active transaction but not in a suspend region it will cause transaction abort.

If the instruction is used outside of a transactional region it behaves like a NOP.

Chapter 5 provides additional information on Intel® TSX Suspend Load Address Tracking.

Operation

XRESLDTRK

```
IF RTM_ACTIVE = 1:
    IF SUSLDTRK_ACTIVE = 1:
        SUSLDTRK_ACTIVE := 0
    ELSE:
        RTM_ABORT
ELSE:
    NOP
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XRESLDTRK void _xresldtrk(void);

SIMD Floating-Point Exceptions

None.

Other Exceptions

#UD If CPUID.(EAX=7, ECX=0):EDX.TSXLDTRK[bit 16] = 0.
If the LOCK prefix is used.

XSUSLDTRK— Suspend Tracking Load Addresses

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 0F 01 E8 XSUSLDTRK	ZO	V/V	TSXLDTRK	Specifies the start of an Intel TSX suspend read address tracking region.

Instruction Operand Encoding

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

Description

The instruction marks the start of an Intel TSX (RTM) suspend load address tracking region. If the instruction is used inside a transactional region, subsequent loads are not added to the read set of the transaction. If the instruction is used inside a suspend load address tracking region it will cause transaction abort.

If the instruction is used outside of a transactional region it behaves like a NOP.

Chapter 5 provides additional information on Intel® TSX Suspend Load Address Tracking.

Operation

XSUSLDTRK

```
IF RTM_ACTIVE = 1:
    IF SUSLDTRK_ACTIVE = 0:
        SUSLDTRK_ACTIVE := 1
    ELSE:
        RTM_ABORT
ELSE:
    NOP
```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

XSUSLDTRK void _xsusldtrk(void);

SIMD Floating-Point Exceptions

None.

Other Exceptions

#UD If CPUID.(EAX=7, ECX=0):EDX.TSXLDTRK[bit 16] = 0.
If the LOCK prefix is used.

CHAPTER 3
INTEL® AMX INSTRUCTION SET REFERENCE, A-Z

3.1 INTRODUCTION

Intel® Advanced Matrix Extensions (Intel® AMX) is a new 64-bit programming paradigm consisting of two components: a set of 2-dimensional registers (tiles) representing sub-arrays from a larger 2-dimensional memory image, and an accelerator able to operate on tiles, the first implementation is called TMUL (tile matrix multiply unit).

An Intel AMX implementation enumerates to the programmer how the tiles can be programmed by providing a palette of options. Two palettes are supported; palette 0 represents the initialized state, and palette 1 consists of 8 KB of storage spread across 8 tile registers named TMM0..TMM7. Each tile has a maximum size of 16 rows x 64 bytes, (1 KB), however the programmer can configure each tile to smaller dimensions appropriate to their algorithm. The tile dimensions supplied by the programmer (rows and bytes_per_row, i.e., **colsb**) are metadata that drives the execution of tile and accelerator instructions. In this way, a single instruction can launch autonomous multi-cycle execution in the tile and accelerator hardware. The palette value (**palette_id**) and metadata are held internally in a tile related control register (TILECFG). The TILECFG contents will be commensurate with that reported in the palette_table (see "CPUID—CPU Identification" in Chapter 1 for a description of the available parameters).

Intel AMX is an extensible architecture. New accelerators can be added, or the TMUL accelerator may be enhanced to provide higher performance. In these cases, the state (TILEDATA) provided by tiles may need to be made larger, either in one of the metadata dimensions (more rows or colsb) and/or by supporting more names. The extensibility is carried out by adding new palette entries describing the additional state. Since execution is driven through metadata, an existing Intel AMX binary could take advantage of larger storage sizes and higher performance TMUL units by selecting the most powerful palette indicated by CPUID and adjusting loop and pointer updates accordingly.

Figure 3-1 shows a conceptual diagram of the Intel AMX architecture. An Intel architecture host drives the algorithm, the memory blocking, loop indices and pointer arithmetic. Tile loads and stores and accelerator commands are sent to multi-cycle execution units. Status, if required, is reported back. Intel AMX instructions are synchronous in the Intel architecture instruction stream and the memory loaded and stored by the tile instructions is coherent with respect to the host's memory accesses. There are no restrictions on interleaving of Intel architecture and Intel AMX code or restrictions on the resources the host can use in parallel with Intel AMX (e.g., Intel AVX-512). There is also no architectural requirement on the Intel architecture compute capability of the Intel architecture host other than it supports 64-bit mode.

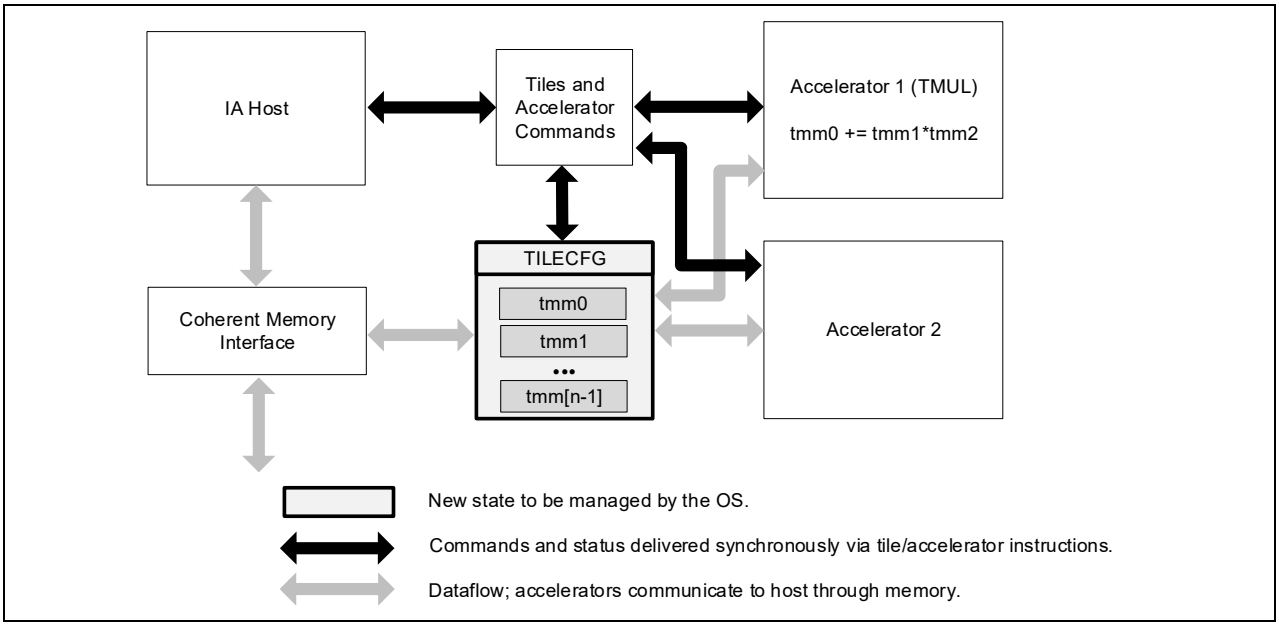


Figure 3-1. Intel® AMX Architecture

Intel AMX instructions use new registers and inherit basic behavior from Intel architecture in the same manner that Intel SSE and Intel AVX did. Tile instructions include loads and stores using the traditional Intel architecture register set as pointers. The TMUL instruction set (defined to be CPUID bits AMX-BF16 and AMX-INT8) only supports reg-reg operations.

TILECFG is programmed using the LDTILECFG instruction. The selected palette defines the available storage and general configuration while the rest of the memory data specifies the number of rows and column bytes for each tile. Consistency checks are performed to ensure the TILECFG matches the restrictions of the palette. A General Protection fault (#GP) is reported if the LDTILECFG fails consistency checks. A successful load of TILECFG with a palette_id other than 0 is represented in this document with TILES_CONFIGURED = 1. When the TILECFG is initialized (palette_id = 0), it is represented in the document as TILES_CONFIGURED = 0. Nearly all Intel AMX instructions will generate a #UD exception if TILES_CONFIGURED is not equal to 1; the exceptions are those that do TILECFG maintenance: LDTILECFG, STTILECFG and TILERELEASE.

If two tiles are configured to contain M rows by N columns of 4-byte data, and two tiles to contain M rows by N columns of 8-byte data, LDTILECFG will ensure that the metadata values are appropriate to the palette (e.g., that rows ≤ 16 and N ≤ 64 for palette 1). The four M and N values can all be different as long as they adhere to the restrictions of the palette. Further dynamic checks are done in the tile and the TMUL instruction set to deal with cases where a legally configured tile may be inappropriate for the instruction operation. Tile registers can be set to ‘invalid’ by configuring the rows and colsb to ‘0’.

Tile loads and stores are strided accesses from the application memory to packed rows of data. Algorithms are expressed assuming row major data layout. Column major users should translate the terms according to their orientation.

TILELOAD* and TILESTORE* instructions are restartable and can handle (up to) 2*rows page faults per instruction. Restartability is provided by a **start_row** parameter in the TILECFG register.

The TMUL unit is conceptually a grid of fused multiply-add units able to read and write tiles. The dimensions of the TMUL unit (tmul_maxk and tmul_maxn) are enumerated similar to the maximum dimensions of the tiles (see “CPUID—CPU Identification” in Chapter 1 for details).

The matrix multiplications in the TMUL instruction set compute C[M][N] += A[M][K] * B[K][N]. The M, N and K values will cause the TMUL instruction set to generate a #UD exception if the following constraints are not met:

- M : ≤ palette.max_rows
- K : ≤ colsb / element_size (A), ≤ palette.max_rows (B) and ≤ tmul_maxk
- N : ≤ colsb / element_size (C and B), ≤ tmul_maxn

In Figure 3-2, the number of rows in tile B matches the K dimension in the matrix multiplication pseudocode. K dimensions smaller than that enumerated in the TMUL grid are also possible and any additional computation the TMUL unit can support will not affect the result.

The number of elements specified by colsb of the B matrix is also less than or equal to tmul_maxn. Any remaining values beyond that specified by the metadata will be set to zero.

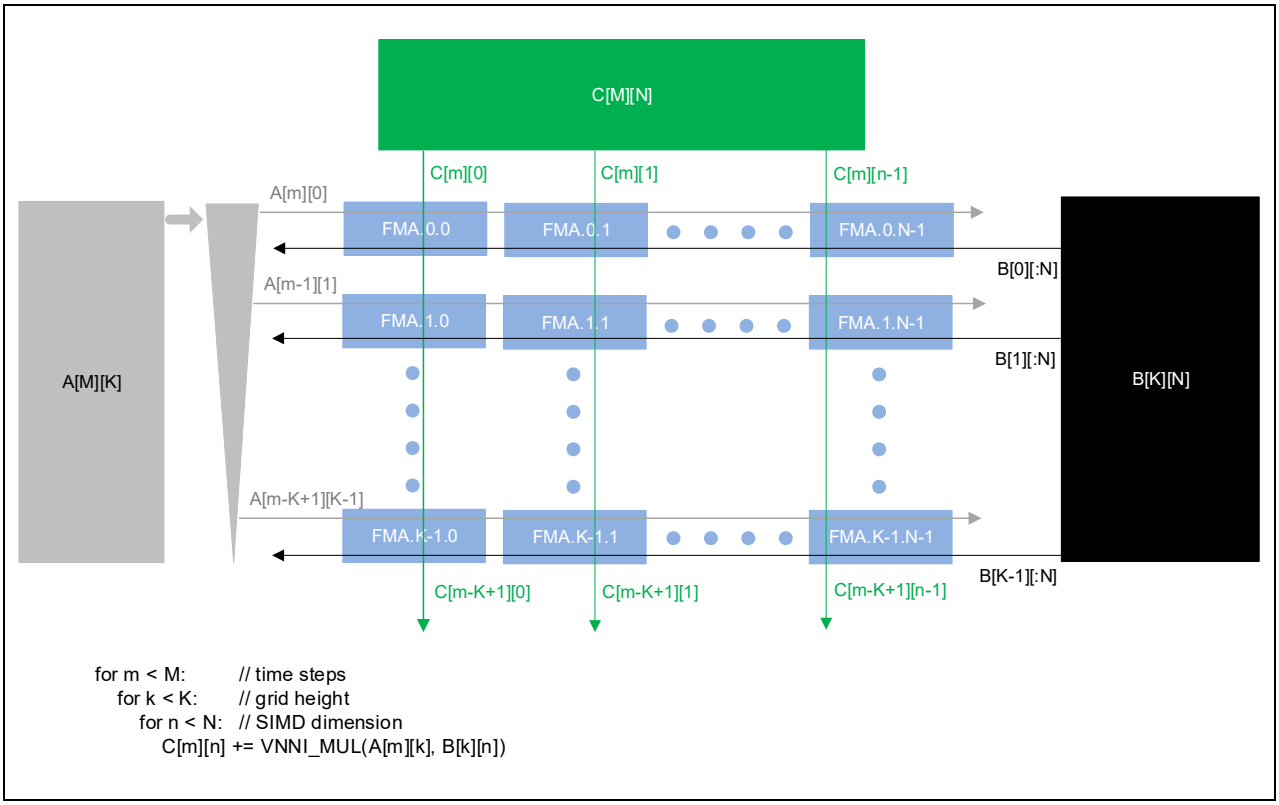


Figure 3-2. The TMUL Unit

The XSAVE feature sets supports context management of the new state defined for Intel AMX. This support is described in Section 3.2.

3.1.1 Tile Architecture Details

The supported parameters for the tile architecture are reported via CPUID; this includes information about how the number of tile registers (max_names) can be configured (the palette). Configuring the tile architecture is intended to be done once when entering a region of tile code using the LDTILECFG instruction specifying the selected palette and describing in detail the configuration for each tile. Incorrect assignments will result in a General Protection fault (#GP). Successful LDTILECFG initializes (zeroes) TILEDATA.

Exiting a tile region is done with the TILERELEASE instruction. It takes no parameters and invalidates all tiles (indicating that the data no longer needs any saving or restoring). Essentially, it is an optimization of LDTILECFG with an implicit palette of 0.

For applications that execute consecutive Intel AMX regions with differing configurations, TILERELEASE is not required between them since the second LDTILECFG will clear all the data while loading the new configuration. There is no instruction set support for automatic nesting of tile regions, though with sufficient effort software can accomplish this by saving and restoring TILEDATA and TILECFG either through the XSAVE architecture or the Intel AMX instructions.

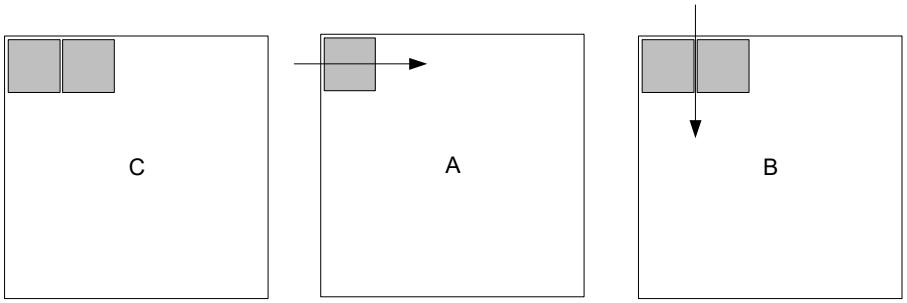
The tile architecture boots in its INIT state, with TILECFG and TILEDATA set to zero. A successfully executing LDTILECFG instruction to a non-zero palette sets the TILES_CONFIGURED=1, indicating the TILECFG is not in the INIT state. The TILERELEASE instruction sets TILES_CONFIGURED = 0 and initializes (zeroes) TILEDATA.

To facilitate handling of tile configuration data, there is a STTILECFG instruction. If the tile configuration is in the INIT state (TILES_CONFIGURED == 0), then STTILECFG will write 64 bytes of zeros. Otherwise STTILECFG will store the TILECFG to memory in the format used by LDTILECFG.

3.1.2 TMUL Architecture Details

The supported parameters for the TMUL architecture are reported via CPUID; see "CPUID—CPU Identification" in Chapter 1, page 1-21, for details. These parameters include a maximum height (**tmul_maxk**) and a maximum SIMD dimension (**tmul_maxn**). The metadata that accompanies the srcdest, src1 and src2 tiles to the TMUL unit will be dynamically checked to see that they match the TMUL unit support for the data type and match the requirements of a meaningful matrix multiplication.

Figure 3-3 shows an example of the inner loop of an algorithm of using the TMUL architecture to compute a matrix multiplication. In this example, we use two result tiles, tmm0 and tmm1, from matrix C to accumulate the intermediate results. One tile from the A matrix (tmm2) is re-used twice as we multiply it by two tiles from the B matrix. The algorithm then advances pointers to load a new A tile and two new B tiles from the directions indicated by the red arrows. An outer loop, not shown, adjusts the pointers for the C tiles.




```
  LDTILECFG [rax]
// assume some outer loops driving the cache tiling (not shown)
{
  TILELOADD tmm0, [rsi+rdi]     // srcdst, RSI points to C, RDI is strided value
  TILELOADD tmm1, [rsi+rdi+N]   // second tile of C, unrolling in SIMD dimension N
  MOV r14, 0
LOOP:
  TILELOADD tmm2, [r8+r9]       // src2 is strided load of A, reused for 2 TMUL instr.
  TILELOADD tmm3, [r10+r11]     // src1 is strided load of B
  TDPBUSD   tmm0, tmm2, tmm3  // update left tile of C
  TILELOADD tmm3, [r10+r11+N]   // src1 loaded with B from next rightmost tile
  TDPBUSD   tmm1, tmm2, tmm3  // update right tile of C
  ADD r8, K                   // update pointers by constants known outside of loop
  ADD r10, K*r11
  ADD r14, K
  CMP r14, LIMIT
  JNE LOOP

  TILESTORED [rsi+rdi],   tmm0  // update the C matrix in memory
  TILESTORED [rsi+rdi+M], tmm1
} // end of outer loop

  TILERELEASE        // return tiles to INIT state
```

Figure 3-3. Matrix Multiply C+= A*B

3.1.3 Handling of Tile Row and Column Limits

Intel AMX operations will zero any rows and any columns beyond the dimensions specified by TILECFG. Tile operations will zero the data beyond the configured number of columns (factoring in the size of the elements) as each row is written. For example, with 64-byte rows and a tile configured with 10 rows and 12 columns, an operation writing dword elements would write each of the first 10 rows with 12*4 bytes of output/result data and zero the remaining 4*4 bytes in each row. Tile operations also fully zero any rows after the first 10 configured rows. When

using a 1 KByte tile with 64-byte rows, there would be 16 rows, so in this example, the last 6 rows would also be zeroed.

Intel AMX instructions will always obey the metadata on reads and the zeroing rules on writes, and so a subsequent XSAVE would see zeros in the appropriate locations. Tiles that are not written by Intel AMX instructions between XRSTOR and XSAVE will write back with the same image they were loaded with regardless of the value of TILECFG.

3.1.4 Exceptions and Interrupts

Tile instructions are restartable so that operations that access strided memory can restart after page faults. To support restarting instructions after these events, the instructions store information in the **TILECFG.start_row** register. TILECFG.start_row indicates the row that should be used for restart; i.e., it indicates **next row after** the rows that have already been successfully loaded (on a TILELOAD) or written to memory (on a TILESTORE) and prevents repeating work that was successfully done.

The TMUL instruction set is not sensitive to the TILECFG.start_row value; this is due to there not being TMUL instructions with memory operands or any restartable faults.

3.2 INTEL® AMX AND THE XSAVE FEATURE SET

Intel AMX is **XSAVE supported**, meaning that it defines processor registers that can be saved and restored using instructions of the XSAVE feature set. Intel AMX is also **XSAVE enabled**, meaning that it must be enabled by the XSAVE feature set before it can be used.

The XSAVE feature set operates on **state components**, each of which is a discrete set of processor registers (or parts of registers). Intel AMX is associated with two state components, XTILECFG and XTILEDATA. Section 3.2.1 describes these state components.

Section 3.2.2 describes how existing enumeration for the XSAVE feature set applies to Intel AMX. Section 3.2.3 explains how software can enable Intel AMX as an XSAVE-enabled feature.

The XTILEDATA state component is very large, and an operating system may prefer not to allocate memory for the XTILEDATA state of every user thread. Such an operating system that enables Intel AMX might prefer to prevent specific user threads from using the feature. An extension called **extended feature disable** (**XFD**) is added to the XSAVE feature set to support such a usage. XFD is described in Section 3.2.6.

3.2.1 State Components for Intel® AMX

As noted earlier, 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). Each state component corresponds to a particular CPU feature. (Some XSAVE-supported features use registers in multiple XSAVE-managed state components.)

The XSAVE feature set organizes state components using **state-component bitmaps**. A state-component bitmap comprises 64 bits; each bit in such a bitmap corresponds to a single state component. Intel AMX defines bits 18:17 for its state components (collectively, these are called **AMX state**):

- State component 17 is used for the 64-byte TILECFG register (**XTILECFG state**).
- State component 18 is used for the 8192 bytes of tile data (**XTILEDATA state**).

These are both **user state components**, meaning that they can be managed by the entire XSAVE feature set. In addition, it implies that setting bits 18:17 of extended control register XCR0 enables Intel AMX. If those bits are zero, execution of an Intel AMX instruction results in an invalid-opcode exception (#UD).

With regard to the XSAVE feature set's init optimization, AMX state is in its initial configuration if the TILECFG register is zero and all tile data are zero.

3.2.2 XSAVE-Related Enumeration for Intel® AMX

A processor enumerates support for the XSAVE feature set and for XSAVE-supported features using the CPUID instruction. Specifically, this is done through sub-functions of CPUID function 0DH. (Software selects a specific sub-function by the value placed in the ECX register.) The following items provide specific details related to Intel AMX:

- CPUID function 0DH, sub-function 0.

  EDX:EAX is a bitmap of all the user state components that can be managed using the XSAVE feature set. (A bit can be set in XCR0 if and only if the corresponding bit is set in this bitmap.) A processor thus enumerates support for Intel AMX by setting both EAX[17] and EAX[18].
- CPUID function 0DH, sub-function 1.

  EAX[4] enumerates general support for extended feature disable (XFD). See Section 3.2.6 for details.
- CPUID function 0DH, sub-function *i* (*i* > 1).

  This sub-function enumerates details for state component *i*. ECX[2] enumerates support for XFD support for this state component.
- CPUID function 0DH, sub-function 17. This sub-function enumerates details for XTILECFG (state component 17). The following items provide specific details:
  - EAX enumerates the size (in bytes) required for XTILECFG, which is 64.
  - EBX enumerates the offset (in bytes, from the base of a standard-format XSAVE area) of the section used for XTILECFG, which is 2752.
  - ECX[0] returns 0, indicating that XTILECFG is a user state component.
  - ECX[1] returns 1, indicating that XTILECFG is located on the next 64-byte boundary following the preceding state component (in a compacted-format XSAVE area).
  - ECX[2] returns 0, indicating no XFD support for XTILECFG.
- CPUID function 0DH, sub-function 18. This sub-function enumerates details for XTILEDATA (state component 18). The following items provide specific details:
  - EAX enumerates the size required for XTILEDATA, which is 8192.
  - EBX enumerates the offset of the section used for XTILEDATA, which is 2816.
  - ECX[0] returns 0, indicating that XTILEDATA is a user state component.
  - ECX[1] returns 1, indicating that XTILEDATA is located on the next 64-byte boundary following the preceding state component.
  - ECX[2] returns 1, indicating XFD support for XTILEDATA.

3.2.3 Enabling Intel® AMX As an XSAVE-Enabled Feature

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]). The following paragraphs provide details relevant to Intel AMX.

XCR0[18:17] are associated with AMX state (see Section 3.2.6). Software can use the XSAVE feature set to manage AMX state only if XCR0[18:17] = 11b. In addition, software can execute Intel AMX instructions only if XCR0[18:17] = 11b. Otherwise, any execution of an Intel AMX instruction causes an invalid-opcode exception (#UD).

XCR0[18:17] have value 00b coming out of RESET. As noted in Section 3.2.2, a processor allows software to set XCR0[18:17] to 11b if and only if CPUID.(EAX=0DH,ECX=0):EAX[17:18] = 11b. In addition, executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and EAX[17] ≠ EAX[18] (XTILECFG and XTILEDATA must be enabled together). This implies that the value of XCR0[17:18] is always either 00b or 11b.

While Intel AMX instructions can be executed only in 64-bit mode, instructions of the XSAVE feature set can operate on XTILECFG and XTILEDATA in any mode. It is recommended that only 64-bit operating systems enable Intel AMX by setting XCR0[18:17].

3.2.4 Loading of XTILECFG and XTILEDATA by XRSTOR and XRSTORS

The LDTILECFG instruction generates a general-protection fault (#GP) if it would load the TILECFG register with an unsupported value. An execution of XRSTOR or XRSTORS does not fault in response to an attempt to load the TILECFG register with such a value. Instead, such executions initialize the register (resulting in TILES_CONFIGURED = 0).

While executions of LDTILECFG initialize TILEDATA, that is not necessarily the case for executions of XRSTOR and XRSTORS that load XTILECFG. An execution of XRSTOR or XRSTORS that is not directed to load XTILEDATA leaves it unmodified, even if the execution is loading XTILECFG.

The current value of the TILECFG register may limit how TMUL instructions access certain parts of XTILEDATA. Such limitations do not apply to XRSTOR and XRSTORS. An execution of either of those instructions loads all 8 KBytes of XTILEDATA regardless of the value in the TILECFG register (or the value that the instruction may be loading into that register).

3.2.5 Saving of XTILEDATA by XSAVE, XSAVEC, XSAVEOPT, and XSAVES

The current value of the TILECFG register may limit how TMUL instructions access certain parts of XTILEDATA. Such limitations do not apply to XSAVE, XSAVEC, XSAVEOPT, and XSAVES. An execution of any of those instructions saves all 8 KBytes of XTILEDATA regardless of the value in the TILECFG register.

3.2.6 Extended Feature Disable (XFD)

An extension called **extended feature disable** (**XFD**) is an extension to the XSAVE feature set that allows an operating system to enable a feature while preventing specific user threads from using the feature. This section describes XFD.

As noted in Section 3.2.2, a processor that supports XFD enumerates CPUID.(EAX=0DH,ECX=1):EAX[4] as 1. Such a processor supports two new MSRs: IA32_XFD (MSR address 1C4H) and IA32_XFD_ERR (MSR address 1C5H). Each of these MSRs contains a state-component bitmap. Bit *i* of either MSR can be set to 1 only if CPUID.(EAX=0DH,ECX=*i*):ECX[2] is enumerated as 1 (see Section 3.2.2). An execution of WRMSR that attempts to set an unsupported bit in either MSR causes a general-protection fault (#GP). The reset values of both of these MSRs is zero.

The first processors to implement Intel AMX will support setting only XTILEDATA (bit 18) in these MSRs.

XFD is enabled for state component *i* if XCR0[*i*] = IA32_XFD[*i*] = 1. (IA32_XFD[*i*] does not affect processor operations if XCR0[*i*] = 0.) When XFD is enabled for a state component, any instruction that would access that state component does not execute and instead generates an device-not-available exception (#NM).

Exceptions are made for certain instructions (including those that initialize the state component). The following items provide details:

- LDTILECFG and TILERELEASE initialize the XTILEDATA state component. An execution of either of these instructions does not generate #NM when XCR0[18] = IA32_XFD[18] = 1; instead, it initializes XTILEDATA normally.
- STTILECFG does not use the XTILEDATA state component. An execution of this instruction does not generate #NM when XCR0[18] = IA32_XFD[18] = 1.
- If XRSTOR or XRSTORS is loading state component *i* and bit *i* of XSTATE_BV field of the XSAVE header is 0, the instruction does not generate #NM when XCR0[*i*] = IA32_XFD[*i*] = 1; instead, it initializes the state component normally. (If bit *i* of XSTATE_BV field of the XSAVE header is 1, the instruction does generate #NM.)
- If XSAVE, XSAVEC, XSAVEOPT, or XSAVES is saving the state component *i*, the instruction does not generate #NM when XCR0[*i*] = IA32_XFD[*i*] = 1; instead, it saves bit *i* of XSTATE_BV field of the XSAVE header as 0 (indicating that the state component is in its initialized state). With the exception of XSAVE, no data is saved for the state component (XSAVE saves the initial value of the state component; for XTILEDATA, this is all zeroes).
- Enclave entry instructions (ENCLU[EENTER] and ENCLU[ERESUME]) generate #NM if XCR0[*i*] = IA32_XFD[*i*] = 1 and bit *i* is set in XFRM field in the attributes of the enclave being entered.

When XFD causes an instruction to generate #NM, the processor loads the IA32_XFD_ERR MSR to identify the disabled state component(s). Specifically, the MSR is loaded with the logical AND of the IA32_XFD MSR and the bitmap corresponding to the state components required by the faulting instruction. (Intel AMX instructions require XTILECFG state and XTILEDATA state to be enabled.)

Device-not-available exceptions that are not due to XFD — those resulting from setting CR0.TS to 1 — do not modify the IA32_XFD_ERR MSR.

3.3 RECOMMENDATIONS FOR SYSTEM SOFTWARE

System software may disable use of Intel AMX by clearing XCR0[18:17], by clearing CR4.OSXSAVE, or by setting IA32_XFD[18]. It is recommended that system software initialize AMX state (e.g., by executing TILERELEASE) before doing so. This is because maintaining AMX state in a non-initialized state may have negative power and performance implications.

System software should not use XFD to implement a "lazy restore" approach to management of the XTILEDATA state component. This approach will **not** operate correctly for a variety of reasons. One is that the LDTILECFG and TILERELEASE instructions initialize XTILEDATA and do not cause an #NM exception. Another is that an execution of XSAVE by a user thread will save XTILEDATA as initialized instead of the data expected by the user thread.

3.4 IMPLEMENTATION PARAMETERS

The parameters are reported via CPUID leaf 1DH. Index 0 reports all zeros for all fields.

```
define palette_table[id]:
    uint16_t total_tile_bytes
    uint16_t bytes_per_tile
    uint16_t bytes_per_row
    uint16_t max_names
    uint16_t max_rows
```

The tile parameters are set by LDTILECFG or XRSTOR* of XTILECFG:

```
define tile[tid]:
    byte rows
    word colsb // bytes_per_row
    bool valid
```

3.5 HELPER FUNCTIONS

Th helper functions used in Intel AMX instructions are defined below.

```
define write_row_and_zero(treg, r, data, nbytes):
   for j in 0 ...nbytes-1:
       treg.row[r].byte[j] := data.byte[j]

   // zero the rest of the row
   for j in nbytes ... palette_table[tilecfg.palette_id].bytes_per_row-1:
      treg.row[r].byte[j] := 0
```

```
define zero_upper_rows(treg, r):
   for i in r ... palette_table[tilecfg.palette_id].max_rows-1:
      for j in 0 ... palette_table[tilecfg.palette_id].bytes_per_row-1:
            treg.row[i].byte[j] := 0
```

```
define zero_tilecfg_start():
   tilecfg.start_row :=0
```

```
define zero_all_tile_data():
  if XCR0[XTILEDATA]:
      b := CPUID(0xD,XTILEDATA).EAX // size of feature
      for j in 0 ...b:
            TILEDATA.byte[j] := 0
```

```
define xcr0_supports_palette(palette_id):
  if palette_id == 0:
     return 1
  elif palette_id == 1:
     if XCR0[XTILECFG] and XCR0[XTILEDATA]:
        return 1
  return 0
```

3.6 NOTATION

Instructions described in this chapter follow the general documentation convention established in *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A*. Additionally, Intel® Advanced Matrix Extensions use notation conventions as described below.

In the instruction encoding boxes, **sibmem** is used to denote an encoding where a MODRM byte and SIB byte are used to indicate a memory operation where the base and displacement are used to point to memory, and the index

register (if present) is used to denote a stride between memory rows. The index register is scaled by the sib.scale field as usual. The base register is added to the displacement, if present.

In the instruction encoding, the MODRM byte is represented several ways depending on the role it plays. The MODRM byte has 3 fields: 2-bit MODRM.MOD field, a 3-bit MODRM.REG field and a 3-bit MODRM.RM field. When all bits of the MODRM byte have fixed values for an instruction, the 2-hex nibble value of that byte is presented after the opcode in the encoding boxes on the instruction description pages. When only some fields of the MODRM byte must contain fixed values, those values are specified as follows:

- If only the MODRM.MOD must be 0b11, and MODRM.REG and MODRM.RM fields are unrestricted, this is denoted as **11:rrr:bbb**. The **rrr** correspond to the 3-bits of the MODRM.REG field and the **bbb** correspond to the 3-bits of the MODMR.RM field.
- If the MODRM.MOD field is constrained to be a value other than 0b11, i.e., it must be one of 0b00, 0b01, or 0b10, then we use the notation !(11).
- If the MODRM.REG field had a specific required value, e.g., 0b101, that would be denoted as mm:101:bbb.

NOTE

Historically the *Intel® 64 and IA-32 Architectures Software Developer's Manual* only specified the MODRM.REG field restrictions with the notation /0 ... /7 and did not specify restrictions on the MODRM.MOD and MODRM.RM fields in the encoding boxes.

3.7 EXCEPTION CLASSES

Alignment exceptions: The Intel AMX instructions that access memory will never generate #AC exceptions.

Table 3-1. Intel® AMX Exception Classes

Class	Instructions	Description
AMX-E1	LDTILECFG	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #GP based on palette and configuration checks (see pseudocode). • #SS(0) if the memory address referencing the SS segment is in a non-canonical form. • #GP if the memory address is in a non-canonical form. • #PF if a page fault occurs.
AMX-E2	STTILECFG	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #SS(0) if the memory address referencing the SS segment is in a non-canonical form. • #GP if the memory address is in a non-canonical form. • #PF if a page fault occurs.

Table 3-1. Intel® AMX Exception Classes(Continued)

Class	Instructions	Description
AMX-E3	TILELOAD, TILELOADDT1, TILESTORE	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #UD if not using SIB addressing. • #UD if TILES_CONFIGURED == 0. • #UD if tsrc or tdest are not valid tiles. • #UD if tsrc.colbytes mod 4 ≠ 0 OR tdest.colbytes mod 4 ≠ 0. • #UD if tilecfg.start_row ≥ tsrc.rows OR tilecfg.start_row ≥ tdest.rows. • #NM if XFD[18] == 1. • #SS(0) if the memory address referencing the SS segment is in a non-canonical form. • #GP if the memory address is in a non-canonical form. • #PF if any memory operand causes a page fault.
AMX-E4	TDPBSSD, TDPBSUD, TDPBUSD,TDPBUUD, TDPBF16PS	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if srcdest == src1 OR src1 == src2 OR srcdest == src2. • #UD if TILES_CONFIGURED == 0. • #UD if srcdest.colbytes mod 4 ≠ 0. • #UD if src1.colbytes mod 4 ≠ 0. • #UD if src2.colbytes mod 4 ≠ 0. • #UD if srcdest/src1/src2 are not valid tiles. • #UD if srcdest.colbytes ≠ src2.colbytes. • #UD if srcdest.rows ≠ src1.rows. • #UD if src1.colbytes / 4 ≠ src2.rows. • #UD if srcdest.colbytes > tmul_maxn. • #UD if src2.colbytes > tmul_maxn. • #UD if src1.colbytes/4 > tmul_maxk. • #UD if src2.rows > tmul_maxk. • #NM if XFD[18] == 1.
AMX-E5	TILZERO	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #UD if TILES_CONFIGURED == 0. • #UD if tdest is not a valid tile. • #NM if XFD[18] == 1.
AMX-E6	TILERELEASE	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111.

3.8 INSTRUCTION SET REFERENCE

LDTILECFG — Load Tile Configuration

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.NP.0F38.W0 49 !(11):000:bbb LDTILECFG m512	A	V/N.E.	AMX-TILE	Load tile configuration as specified in m512.

Instruction Operand Encoding

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

Description

The LDTILECFG instruction takes a operand containing a pointer to a 64-byte memory location containing the description of the tiles to be supported. In order to configure the tiles, the AMX-TILE bit in CPUID must be set and the operating system has to have enabled the tiles architecture.

The memory area first describes the number of tiles selected and then selects from the palette of tile types. Requests must be compatible with the restrictions provided by CPUID.

The memory area describes how many tiles are being used and defines each tile in terms of rows and columns; see Table 3-1 below.

Table 3-1. Memory Area Layout

Byte(s)	Field Name	Description
0	palette	Palette selects the supported configuration of the tiles that will be used.
1	start_row	start_row is used for storing the restart values for interrupted operations.
2-15	reserved, must be zero	
16-17	tile0.colsb	Tile 0 bytes per row.
18-19	tile1.colsb	Tile 1 bytes per row.
20-21	tile2.colsb	Tile 2 bytes per row.
...	(sequence continues)	
30-31	tile7.colsb	Tile 7 bytes per row.
32-47	reserved, must be zero	
48	tile0.rows	Tile 0 rows.
49	tile1.rows	Tile 1 rows.
50	tile2.rows	Tile 2 rows.
...	(sequence continues)	
55	tile7.rows	Tile 7 rows.
56-63	reserved, must be zero	

If a tile row and column pair is not used to specify tile parameters, they must have the value zero. All enabled tiles (based on the palette) must be configured. Specifying tile parameters for more tiles than the implementation limit or the palette limit results in a #GP fault.

If the palette_id is zero, that signifies the INIT state for the both XTILECFG and XTILEDATA. Tiles are zeroed in the INIT state. The only legal non-INIT value for palette_id is 1.

Any attempt to execute the LDTILECFG instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

LDTILECFG mem

```
error := False
buf := read_memory(mem, 64)
temp_tilecfg.palette_id := buf.byte[0]
if temp_tilecfg.palette_id > max_palette:
    error := True
if not xcr0_supports_palette(temp_tilecfg.palette_id):
    error := True
if temp_tilecfg.palette_id !=0:
    temp_tilecfg.start_row := buf.byte[1]
    if buf.byte[2..15] is nonzero:
        error := True
    p := 16
    # configure columns
    for n in 0 ... palette_table[temp_tilecfg.palette_id].max_names-1:
        temp_tilecfg.t[n].colsb:= buf.word[p/2]
        p := p + 2
        if temp_tilecfg.t[n].colsb > palette_table[temp_tilecfg.palette_id].bytes_per_row:
            error := True
    if nonzero(buf[p...47]):
        error := True

    # configure rows
    p := 48
    for n in 0 ... palette_table[temp_tilecfg.palette_id].max_names-1:
        temp_tilecfg.t[n].rows:= buf.byte[p]
        if temp_tilecfg.t[n].rows > palette_table[temp_tilecfg.palette_id].max_rows:
            error := True
        p := p + 1

    if nonzero(buf[p...63]):
        error := True

    # validate each tile's row & col configs are reasonable
    for n in 0 ... palette_table[temp_tilecfg.palette_id].max_names-1:
        if temp_tilecfg.t[n].rows !=0 and temp_tilecfg.t[n].colsb != 0:
            temp_tilecfg.t[n].valid := 1
        elif temp_tilecfg.t[n].rows == 0 and temp_tilecfg.t[n].colsb == 0:
            temp_tilecfg.t[n].valid := 0
        else:
            error := True// one of rows or colsbwas 0 but not both.

if error:
    #GP
elif temp_tilecfg.palette_id == 0:
    TILES_CONFIGURED := 0// init state
    tilecfg := 0// equivalent to 64B of zeros
    zero_all_tile_data()
else:
    tilecfg := temp_tilecfg
    zero_all_tile_data()
    TILES_CONFIGURED := 1
```

Intel C/C++ Compiler Intrinsic Equivalent

LDTILECFG void _tile_loadconfig(const void *);

Flags Affected

None.

Exceptions

AMX-E1; see Section 3.7, "Exception Classes" for details.

STTILECFG — Store Tile Configuration

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 49 !(11):000:bbb STTILECFG m512	A	V/N.E.	AMX-TILE	Store tile configuration in m512.

Instruction Operand Encoding

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

Description

The STTILECFG instruction takes a pointer to a 64-byte memory location (described in Table 3-1) that will, after successful execution of this instruction, contain the description of the tiles that were configured. In order to configure tiles, the AMX-TILE bit in CPUID must be set and the operating system has to have enabled the tiles architecture.

If the tiles are not configured, then STTILECFG stores 64B of zeros to the indicated memory location.

Any attempt to execute the STTILECFG instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

STTILECFG mem

```
if TILES_CONFIGURED == 0:
    //write 64 bytes of zeros at mem pointer
    buf[0..63] := 0
    write_memory(mem, 64, buf)
else:
    buf.byte[0] := tilecfg.palette_id
    buf.byte[1] := tilecfg.start_row
    buf.byte[2..15] := 0

    p := 16
    for n in 0 ... palette_table[tilecfg.palette_id].max_names-1:
        buf.word[p/2] := tilecfg.t[n].colsb
        p := p + 2
    if p < 47:
        buf.byte[p..47] := 0
    p := 48
    for n in 0 ... palette_table[tilecfg.palette_id].max_names-1:
        buf.byte[p++] := tilecfg.t[n].rows
    if p < 63:
        buf.byte[p..63] := 0

    write_memory(mem, 64, buf)
```

Intel C/C++ Compiler Intrinsic Equivalent

STTILECFG void _tile_storeconfig(void *);

Flags Affected

None.

Exceptions

AMX-E2; see Section 3.7, “Exception Classes” for details.

TDPBF16PS — Dot Product of BF16 Tiles Accumulated into Packed Single Precision Tile

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F3.0F38.W0 5C 11:rrr:bbb TDPBF16PS tmm1, tmm2, tmm3	A	V/N.E.	AMX-BF16	Matrix multiply BF16 elements from tmm2 and tmm3, and accumulate the packed single precision elements in tmm1.

Instruction Operand Encoding

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

Description

This instruction performs a set of SIMD dot-products of two BF16 elements and accumulates the results into a packed single precision tile. Each dword element in input tiles tmm2 and tmm3 is interpreted as a BF16 pair. For each possible combination of (row of tmm2, column of tmm3), the instruction performs a set of SIMD dot-products on all corresponding BF16 pairs (one pair from tmm2 and one pair from tmm3), adds the results of those dot-products, and then accumulates the result into the corresponding row and column of tmm1.

"Round to nearest even" rounding mode is used when doing each accumulation of the FMA. Output denormals are always flushed to zero and input denormals are always treated as zero. MXCSR is not consulted nor updated.

Any attempt to execute the TDPBF16PS instruction inside a TSX transaction will result in a transaction abort.

Operation

```
define make_fp32(x):
    // The x parameter is bfloat16. Pack it in to upper 16b of a dword.
    // The bit pattern is a legal fp32 value. Return that bit pattern.
    dword: = 0
    dword[31:16] := x
    return dword
```

TDPBF16PS tsrcdest, tsrc1, tsrc2

```
// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)

# src1 and src2 elements are pairs of bfloat16
elements_src1 := tsrc1.colsb / 4
elements_src2 := tsrc2.colsb / 4
elements_dest := tsrcdest.colsb / 4
elements_temp := tsrcdest.colsb / 2        // Count is in bfloat16 prior to horizontal

for m in 0 ... tsrcdest.rows-1:
    temp1[ 0 ... elements_temp-1 ] := 0
    for k in 0 ... elements_src1-1:
        for n in 0 ... elements_dest-1:

            // FP32 FMA with DAZ=FTZ=1, RNE rounding.
            // MXCSR is neither consulted nor updated.
            // No exceptions raised or denoted.

            temp1.fp32[2*n+0] += make_fp32(tsrc1.row[m].bfloat16[2*k+0]) * make_fp32(tsrc2.row[k].bfloat16[2*n+0])
            temp1.fp32[2*n+1] += make_fp32(tsrc1.row[m].bfloat16[2*k+1]) * make_fp32(tsrc2.row[k].bfloat16[2*n+1])

    for n in 0 ... elements_dest-1:
        // DAZ=FTZ=1, RNE rounding.
        // MXCSR is neither consulted nor updated.
        // No exceptions raised or denoted.
        tmpf32 := temp1.fp32[2*n] + temp1.fp32[2*n+1]
        tsrcdest.row[m].fp32[n] := tsrcdest.row[m].fp32[n] + tmpf32
    write_row_and_zero(tsrcdest, m, tmp, tsrcdest.colsb)

zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tilecfg_start()
```

Intel C/C++ Compiler Intrinsic Equivalent

TDPBF16PS void _tile_dpbf16ps(__tile dst, __tile src1, __tile src2);

Flags Affected

None.

Exceptions

AMX-E4; see Section 3.7, “Exception Classes” for details.

TDPBSSD/TDPBSUD/TDPBUSD/TDPBUUD — Dot Product of Signed/Unsigned Bytes with Dword Accumulation

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F2.0F38.W0 5E 11:rrr:bbb TDPBSSD tmm1, tmm2, tmm3	A	V/N.E.	AMX-INT8	Matrix multiply signed byte elements from tmm2 by signed byte elements from tmm3 and accumulate the dword elements in tmm1.
VEX.128.F3.0F38.W0 5E 11:rrr:bbb TDPBSUD tmm1, tmm2, tmm3	A	V/N.E.	AMX-INT8	Matrix multiply signed byte elements from tmm2 by unsigned byte elements from tmm3 and accumulate the dword elements in tmm1.
VEX.128.66.0F38.W0 5E 11:rrr:bbb TDPBUSD tmm1, tmm2, tmm3	A	V/N.E.	AMX-INT8	Matrix multiply unsigned byte elements from tmm2 by signed byte elements from tmm3 and accumulate the dword elements in tmm1.
VEX.128.NP.0F38.W0 5E 11:rrr:bbb TDPBUUD tmm1, tmm2, tmm3	A	V/N.E.	AMX-INT8	Matrix multiply unsigned byte elements from tmm2 by unsigned byte elements from tmm3 and accumulate the dword elements in tmm1.

Instruction Operand Encoding

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

Description

For each possible combination of (row of tmm2, column of tmm3), the instruction performs a set of SIMD dot-products on all corresponding four byte elements, one from tmm2 and one from tmm3, adds the results of those dot-products, and then accumulates the result into the corresponding row and column of tmm1. Each dword in input tiles tmm2 and tmm3 is interpreted as four byte elements. These may be signed or unsigned. Each letter in the two-letter pattern SU, US, SS, UU indicates the signed/unsigned nature of the values in tmm2 and tmm3, respectively.

Any attempt to execute the TDPBSSD/TDPBSUD/TDPBUSD/TDPBUUD instructions inside an Intel TSX transaction will result in a transaction abort.

Operation

```
define DPBD(c,x,y):// arguments are dwords
    if *x operand is signed*:
        extend_src1 := SIGN_EXTEND
    else:
        extend_src1 := ZERO_EXTEND

    if *y operand is signed*:
        extend_src2 := SIGN_EXTEND
    else:
        extend_src2 := ZERO_EXTEND

    p0dword := extend_src1(x.byte[0]) * extend_src2(y.byte[0])
    p1dword := extend_src1(x.byte[1]) * extend_src2(y.byte[1])
    p2dword := extend_src1(x.byte[2]) * extend_src2(y.byte[2])
    p3dword := extend_src1(x.byte[3]) * extend_src2(y.byte[3])

    c := c + p0dword + p1dword + p2dword + p3dword
```

TDPBSSD, TDPBSUD, TDPBUSD, TDPBUUD tsrcdest, tsrc1, tsrc2 (Register Only Version)

```
// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)

tsrc1_elements_per_row := tsrc1.colsb / 4
tsrc2_elements_per_row := tsrc2.colsb / 4
tsrcdest_elements_per_row := tsrcdest.colsb / 4

for m in 0 ... tsrcdest.rows-1:
    tmp := tsrcdest.row[m]
    for k in 0 ... tsrc1_elements_per_row-1:
        for n in 0 ... tsrcdest_elements_per_row-1:
            DPBD( tmp.dword[n], tsrc1.row[m].dword[k], tsrc2.row[k].dword[n] )
    write_row_and_zero(tsrcdest, m, tmp, tsrcdest.colsb)

zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tilecfg_start()
```

Intel C/C++ Compiler Intrinsic Equivalent

```
TDPBSSD    void _tile_dpbssd(__tile dst, __tile src1, __tile src2);
TDPBSUD    void _tile_dpbsud(__tile dst, __tile src1, __tile src2);
TDPBUSD    void _tile_dpbusd(__tile dst, __tile src1, __tile src2);
TDPBUUD    void _tile_dpbuud(__tile dst, __tile src1, __tile src2);
```

Flags Affected

None.

Exceptions

AMX-E4; see Section 3.7, “Exception Classes” for details.

TILELOADD/TILELOADDT1 — Load Tile

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F2.0F38.W0 4B !(11):rrr:100 TILELOADD tmm1, sibmem	A	V/N.E.	AMX-TILE	Load data into tmm1 as specified by information in sibmem.
VEX.128.66.0F38.W0 4B !(11):rrr:100 TILELOADDT1 tmm1, sibmem	A	V/N.E.	AMX-TILE	Load data into tmm1 as specified by information in sibmem with hint to optimize data caching.

Instruction Operand Encoding

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

Description

This instruction is required to use SIB addressing. The index register serves as a stride indicator. If the SIB encoding omits an index register, the value zero is assumed for the content of the index register.

This instruction loads a tile destination with rows and columns as specified by the tile configuration. The "T1" version provides a hint to the implementation that the data will likely not be reused in the near future and the data caching can be optimized accordingly.

The TILECFG.start_row in the XTILECFG data should be initialized to '0' in order to load the entire tile and is set to zero on successful completion of the TILELOADD instruction. TILELOADD is a restartable instruction and the TILECFG.start_row will be non-zero when restartable events occur during the instruction execution.

Only memory operands are supported and they can only be accessed using a SIB addressing mode, similar to the V[P]GATHER*/V[P]SCATTER* instructions.

Any attempt to execute the TILELOADD/TILELOADDT1 instructions inside an Intel TSX transaction will result in a transaction abort.

Operation

```
TILELOADD[,T1] tdest, tsib

start := tilecfg.start_row

zero_upper_rows(tdest,start)

membegin := tsib.base + displacement
// if no index register in the SIB encoding, the value zero is used.
stride := tsib.index << tsib.scale
nbytes := tdest.colsb
while start < tdest.rows:
    memptr := membegin + start * stride
    write_row_and_zero(tdest, start, read_memory(memptr, nbytes), nbytes)
    start := start + 1
zero_tilecfg_start()
// In the case of a memory fault in the middle of an instruction, the tilecfg.start_row := start
```

Intel C/C++ Compiler Intrinsic Equivalent

```
TILELOADD      void _tile_loadd(__tile dst, const void *base, int stride);
TILELOADDT1    void _tile_stream_loadd(__tile dst, const void *base, int stride);
```

Flags Affected

None.

Exceptions

AMX-E3; see Section 3.7, “Exception Classes” for details.

TILERELEASE — Release Tile

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.NP.0F38.W0 49 C0 TILERELEASE	A	V/N.E.	AMX-TILE	Initialize TILECFG and TILEDATA.

Instruction Operand Encoding

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

Description

This instruction returns TILECFG and TILEDATA to the INIT state.

Any attempt to execute the TILERELEASE instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

```
zero_all_tile_data()
tilecfg := 0// equivalent to 64B of zeros
TILES_CONFIGURED := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
TILERELEASE        void _tile_release(void);
```

Flags Affected

None.

Exceptions

AMX-E6; see Section 3.7, “Exception Classes” for details.

TILESTORED — Store Tile

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F3.0F38.W0 4B !(11):rrr:100 TILESTORED sibmem, tmm1	A	V/N.E.	AMX-TILE	Store a tile in sibmem as specified in tmm1.

Instruction Operand Encoding

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

Description

This instruction is required to use SIB addressing. The index register serves as a stride indicator. If the SIB encoding omits an index register, the value zero is assumed for the content of the index register.

This instruction stores a tile source of rows and columns as specified by the tile configuration.

The TILECFG.start_row in the XTILECFG data should be initialized to '0' in order to store the entire tile and are set to zero on successful completion of the TILESTORED instruction. TILESTORED is a restartable instruction and the TILECFG.start_row will be non-zero when restartable events occur during the instruction execution.

Only memory operands are supported and they can only be accessed using a SIB addressing mode, similar to the V[P]GATHER*/V[P]SCATTER* instructions.

Any attempt to execute the TILESTORED instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

```
TILESTORED tsib, tsrc

start := tilecfg.start_row

membegin := tsib.base + displacement
// if no index register in the SIB encoding, the value zero is used.
stride := tsib.index << tsib.scale

while start < tdest.rows:
    memptr := membegin + start * stride
    write_memory(memptr, tsrc.colsb, tsrc.row[start])
    start := start + 1
zero_tilecfg_start()
// In the case of a memory fault in the middle of an instruction, the tilecfg.start_row := start
```

Intel C/C++ Compiler Intrinsic Equivalent

```
TILESTORED  void _tile_stored(__tile src, void *base, int stride);
```

Flags Affected

None.

Exceptions

AMX-E3; see Section 3.7, “Exception Classes” for details.

TILEZERO — Zero Tile

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F2.0F38.W0 49 11:rrr:000 TILEZERO tmm1	A	V/N.E.	AMX-TILE	Zero the destination tile.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (w)	NA	NA	NA

Description

This instruction zeroes the destination tile.

Any attempt to execute the TILEZERO instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

```
TILEZERO tdest

nbytes := palette_table[palette_id].bytes_per_row

for i in 0 ... palette_table[palette_id].max_rows-1:
    for j in 0 ... nbytes-1:
        tdest.row[i].byte[j] := 0
zero_tilecfg_start()
```

Intel C/C++ Compiler Intrinsic Equivalent

```
TILEZERO      void _tile_zero(__tile dst);
```

Flags Affected

None.

Exceptions

AMX-E5; see Section 3.7, “Exception Classes” for details.

CHAPTER 4
ENQUEUE STORES AND PROCESS ADDRESS SPACE IDENTIFIERS (PASIDS)

Chapter 2 described the ENQCMD and ENQCMDS instructions. These instructions perform **enqueue stores**, which write command data to special device registers called **enqueue registers**.

Bits 19:0 of the 64-byte command data written by an enqueue store conveys the process address space identifier (PASID) associated with the command. Software can use PASIDs to identify individual software threads. Devices supporting enqueue registers may use these PASIDs in responding to commands submitted through those registers.

As explained in Chapter 2, an execution of ENQCMD formats the command data with the PASID specified in bits 19:0 of the IA32_PASID MSR. It is expected that system software will configure that MSR to contain the PASID associated with the software thread that is executing.

ENQCMDS can be executed only by system software operating with CPL = 0. It is the responsibility of system software executing ENQCMDS to configure the command data with the appropriate PASID.

Section 4.1 provides details of the IA32_PASID MSR. Section 4.2 describes how the XSAVE feature set supports that MSR. Section 4.3 presents PASID virtualization, a virtualization feature that allows a virtual-machine monitor to control the PASID values produced by enqueue stores executed by software in a virtual machine.

4.1 THE IA32_PASID MSR

This section describes the IA32_PASID MSR used by the ENQCMD instruction. The MSR can be read and written with the RDMSR and WRMSR instructions, using MSR index D93H. The MSR has format given in Table 4-1.

Table 4-1. IA32_PASID MSR

Bit Offset	Description
19:0	Process address space identifier (PASID). Specifies the PASID of the currently running software thread.
30:20	Reserved
31	Valid. Execution of ENQCMD causes a #GP if this bit is clear.
63:32	Reserved

An execution of WRMSR causes a general-protection exception (#GP) in response to an attempt to set any bit in the ranges 30:20 or 63:32. Executions of RDMSR always return zero for those bits.

Because system software may associate a PASID with a software thread, it may choose to update the IA32_PASID MSR on context switches. To facilitate such a usage, the XSAVE feature set is extended to manage the IA32_PASID MSR. These extensions are detailed in Section 4.2.

4.2 THE PASID STATE COMPONENT FOR THE XSAVE FEATURE SET

As noted in Section 4.1, system software may choose to update the IA32_PASID MSR on context switches. This usage is supported by extensions to the XSAVE feature set.

The XSAVE feature set supports the saving and restoring of state components. These state components are organized using state-component bitmaps (each bit in such a bitmap corresponds to a state component).

A new state component is introduced called **PASID state**. PASID state comprises the IA32_PASID MSR. It is defined to be state component 10, so PASID state is associated with bit 10 in state component bitmaps. It is a

supervisor state component, meaning that it can be managed only by the XSAVES and XRSTORS instructions. System software can enable those instructions to manage PASID state by setting bit 10 in the IA32_XSS MSR.

Processor support for this management of PASID state is enumerated by the CPUID instruction as follows:

- CPUID function 0DH, sub-function 1, enumerates in EDX:ECX a bitmap of the supervisor state components. ECX[10] will be enumerated as 1 to indicate that PASID state is supported.
- If PASID state is supported, CPUID function 0DH, sub-function 10 enumerates details for state component as follows:
  - EAX enumerates 8 as the size (in bytes) required for PASID state. (The state component comprises only the one MSR.)
  - EBX enumerates value 0, as is the case for supervisor state components.
  - ECX[0] enumerates 1, indicating that PASID state is a supervisor state component.
  - ECX[1] enumerates 0, indicating that state component 10 is located immediately following the preceding state component when the compacted format of the extended region of an XSAVE area is used.
  - ECX[31:2] and EDX enumerate 0, as is the case for all state components.

Like WRMSR, XRSTORS causes a general-protection exception (#GP) in response to an attempt to set any bit in the IA32_PASID MSR in the ranges 30:20 or 63:32. Like RDMSR, XSAVES always saves zero for those bits.

The XSAVES instruction optimizes the amount of data that it writes to memory by not writing data for a state component known to be in its initial configuration. PASID state is in its initial configuration if the IA32_PASID MSR is 0.

4.3 PASID TRANSLATION

As noted earlier, an operating system (OS) may use PASIDs to identify individual software threads that are allowed to access devices supporting enqueue registers.

Intel® Scalable I/O Virtualization (Intel® Scalable IOV) defines an approach to hardware-assisted I/O virtualization, extending it to support seamless addition of resources and dynamic provisioning of containers.[1] With Intel Scalable IOV, a virtual-machine monitor (VMM) needs to control the PASIDs that are used by different virtual machines just as the guest OS controls the PASIDs used by software threads.

To allow a VMM to control the PASIDs used by enqueue stores while still allowing efficient use by a guest OS, a new virtualization feature is introduced, called **PASID translation**. PASID translation, if enabled, applies to any enqueue store performed by software in a virtual machine: the 20-bit PASID value specified by the guest operating system (**guest PASID**) for ENQCMD or ENQCMDS is translated into a 20-bit value (**host PASID**) that is used in the resulting enqueue store.

4.3.1 PASID Translation Structures

PASID translation is implemented by two hierarchies of data structures (**PASID-translation hierarchies**) configured by a VMM. Guest PASIDs 00000H to 7FFFFH are translated through the low PASID-translation hierarchy, while guest PASIDs 80000 to FFFFFH are translated through the high PASID-translation hierarchy.

Each PASID-translation hierarchy includes a 4-KByte **PASID directory**. A PASID directory comprises 512 8-byte entries, each of which has the following format:

- Bit 0 is the entry's present bit. The entry is used only if this bit is 1.
- Bits 11:1 are reserved and must be 0.
- Bits M–1:12 specify the 4-KByte aligned address of a PASID table (see below), where M is the physical-address width supported by the processor.
- Bits 63:M are reserved and must be 0.

1. See the *Intel® Scalable I/O Virtualization Technical Specification* for more details.

A PASID-translation hierarchy also includes up to 512 4-KByte **PASID tables**; these are referenced by PASID directory entries (see above). A PASID table comprises 1024 4-byte entries, each of which has the following format:

- Bits 19:0 are the host PASID specified by the entry.
- Bits 30:20 are reserved and must be 0.
- Bits 31 is the entry's valid bit. The entry is used only if this bit is 1.

Section 4.3.2 explains how the PASID-translation hierarchies are used to translate the PASIDs used for enqueue stores.

4.3.2 The PASID Translation Process

Each execution of ENQCMD or ENQCMDS results in an enqueue store with a PASID value. (ENQCMD obtains the PASID from the IA32_PASID MSR; ENQCMDS obtains it from the instruction's source operand.) When PASID translation is enabled, this PASID value is interpreted as a guest PASID. The guest PASID is converted to a host PASID; the enqueue store uses the host PASID for bits 19:0 of the command data that it writes.

The PASID translation process is illustrated in Figure 4-1.

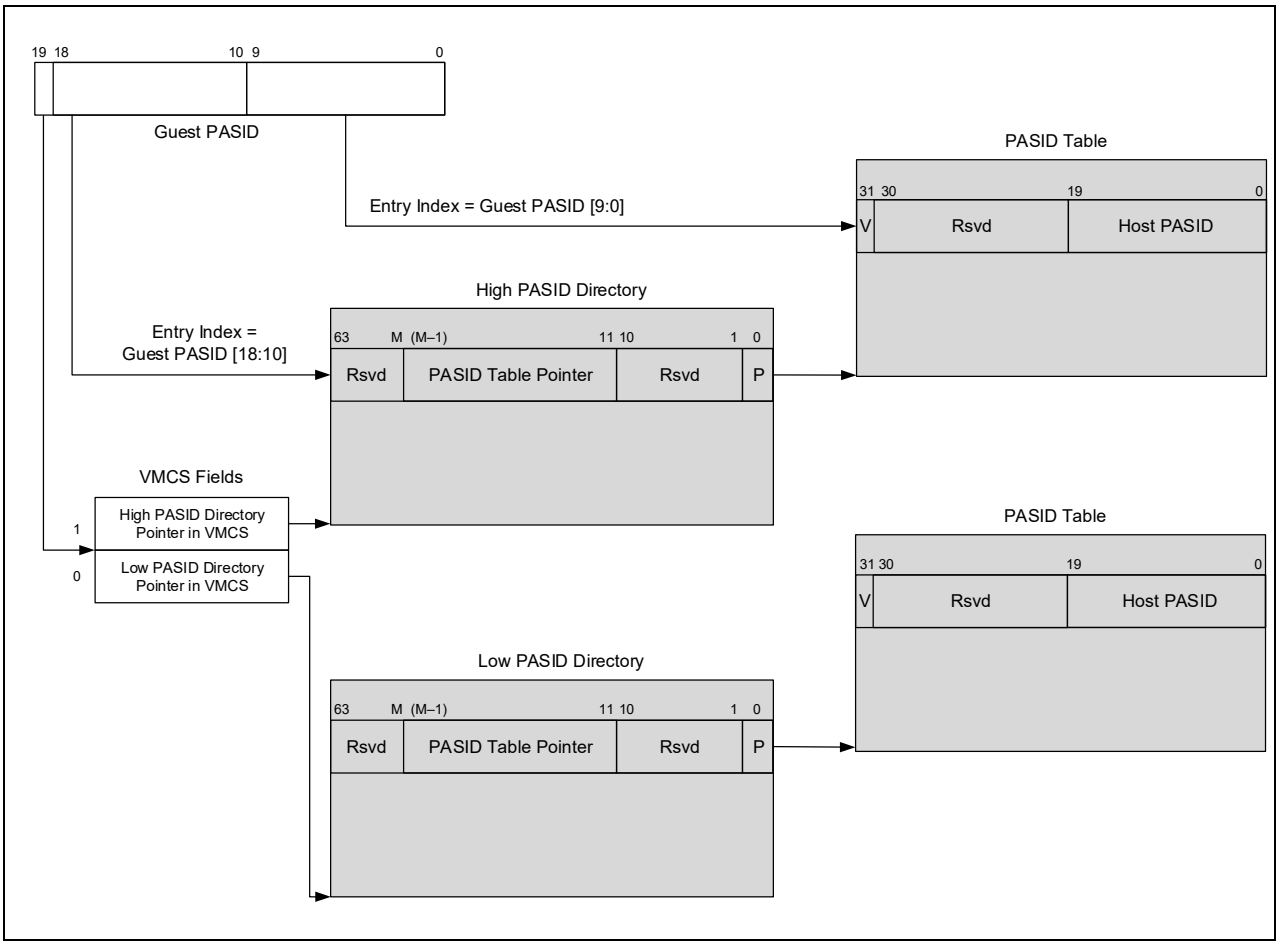


Figure 4-1. PASID Translation Process

The process operates as follows:

- If bit 19 of guest PASID is clear, the low PASID directory is used; otherwise, the high PASID directory is used.

- Bits 18:10 of the guest PASID select an entry from the PASID directory. A VM exit occurs if the entry's valid bit is clear or if any reserved bit is set. Otherwise, bits M:0 of the entry (with bit 0 cleared) contain the physical address of a PASID table, where M is the physical-address width supported by the processor.
- Bits 9:0 of the guest PASID select an entry from the PASID table. A VM exit occurs if the entry's present bit is clear or if any reserved bit is set. Otherwise, bits 19:0 of the entry are the host PASID.

An execution of ENQCMD or ENQCMDS performs PASID translation only after checking for conditions that may result in general-protection exception (the check of IA32_PASID.Valid for ENQCMD; the check of CPL for ENQCMDS) and after loading the instruction's source operand from memory. PASID translation occurs before the actual enqueue store and thus before any faults or VM exits that it may cause (e.g., page faults or EPT violations).

4.3.3 VMX Support

A VMM enables PASID translation by setting secondary processor-based VM-execution control 21. A processor enumerates support for the 1-setting of this control in the normal way (by setting bit 53 of the IA32_VMX_PROC-BASED_CTLS2 MSR). It is expected that any processor that supports the ENQCMD and ENQCMDS instructions will also support PASID virtualization and vice versa.

PASID translation uses two new 64-bit VM-execution control fields in the VMCS: the **low PASID directory address** and the **high PASID directory address**. These are the physical addresses of the low PASID directory and the high PASID directory, respectively. Software can access these new VMCS fields using the encoding pairs 00002038H/00002039H and 0000203AH/0000203BH, respectively.

If the "PASID translation" VM-execution control is 1, VM entry fails if either PASID directory address sets any bit in the ranges 11:0 or 63:M, where M is the physical-address width supported by the processor.

Section 4.3.2 identified situations that may cause a VM exit during PASID translation. Such a VM exit uses basic exit reason 72 (for ENQCMD PASID translation failure) or 73 (ENQCMDS PASID translation failure). The exit qualification is determined as follows:

- For ENQCMD, it is IA32_PASID & 7FFFFH (bits 63:20 are cleared).
- For ENQCMDS, it is SRC & FFFFFFFFH, where SRC is the instruction's source operand (only bits 31:0 may be set).

CHAPTER 5
INTEL® TSX SUSPEND LOAD ADDRESS TRACKING

Chapter 2 described the XSUSLDTRK and XRESLDTRK instructions.

A processor supports Intel® TSX suspend load address tracking if CPUID.07H.EDX.TSXLDTRK [bit 16] = 1. An application must check if the processor supports Intel TSX suspend load address tracking before it uses the Intel TSX suspend load address tracking instructions (XSUSLDTRK, XRESLDTRK). These instructions will generate a #UD exception when used on a processor that does not support TSX suspend load tracking.

Programmers can choose which memory accesses do not need to be tracked in the TSX read set. A programmer who uses the suspend load address tracking feature must ensure that there are no atomicity requirements related to the addresses they choose to exclude from the read set as **hardware will not detect read-write conflicts for those addresses.**

To prevent load addresses from being entered into the read set, the programmer should use the XSUSLDTRK and XRESLDTRK instructions. The XSUSLDTRK instruction specifies the start of a suspend region (addresses of subsequent loads will not be added to the transaction read set), and the XRESLDTRK instruction specifies the end of a suspend region (addresses of subsequent loads will be added to the transaction read set).

The execution of a suspend load address tracking region is very similar to transaction execution with the following exceptions:

- The addresses of loads between suspend/resume are not tracked for read-write conflicts if the addresses are accessed inside the suspend region only (i.e., they are not added to the transaction read set). The addresses are still tracked if they are accessed outside of the suspend region inside the transaction.
- Transaction start/end inside the suspend region is not supported; any execution of XACQUIRE/XBEGIN or XRELEASE/XEND will cause the transaction to abort.
- There is no support for suspend region nesting; XSUSLDTRK will cause a transaction abort.

CHAPTER 6
HYPERVISOR-MANAGED LINEAR ADDRESS TRANSLATION

This chapter provides information about a new VT-x capability called Hypervisor-managed Linear Address Translation (HLAT). This capability is intended to be used by a Hypervisor/Virtual Machine Monitor (VMM) to enforce guest linear translation (to guest physical mappings). When combined with the existing Extended Page Table (EPT) capability, HLAT enables the VMM to ensure the integrity of combined guest linear translation (mappings and permissions) cached by the processor TLB, via a reduced software TCB managed by the VMM. The VMM-enforced guest translations are therefore not subject to tamper by untrusted system software adversaries.

6.1 USAGE

This feature is intended to augment the security functionality for a type of Virtual Machine Monitor (VMM) that may use legacy EPT read/write/execute (XWR) permission bits (bits 2:0 of the EPTE) as well as the new User-execute (XU) access bit (bit 10 of the EPTE) to ensure the integrity of code/data resident in guest physical memory assigned to the guest operating system. EPT permissions are also used in these VMMs to isolate memory; for example, to host a Secure Kernel (SK) that can manage security properties for the General Purpose Kernel (GPK). For such usages, it is important that the VMM ensure that the guest linear address mappings which are used by the General Purpose Kernel to refer to the EPT monitored guest physical pages are access-controlled as well.
Figure 6-1 below shows an example software setup.

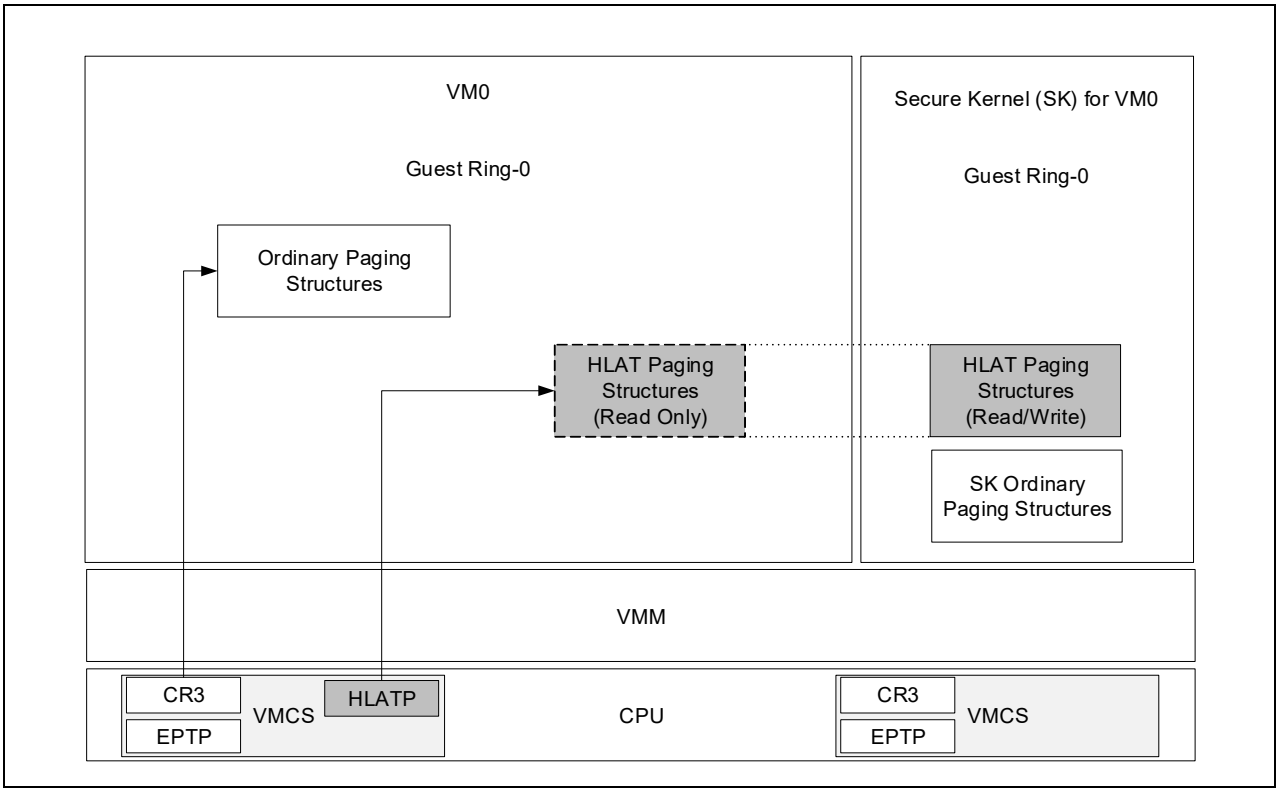


Figure 6-1. Example HLAT Software Usage

VMMs could enforce the integrity of these specific guest linear to guest physical mappings (paging structures) by using legacy EPT permissions to mark the guest physical memory containing the relevant guest paging structures as read-only. The intent of marking these guest paging structures as read-only is to ensure an invalid mapping is not created by guest software. However, such page-table edit control techniques are known to cause very high

overheads due to the requirement that the VMM must monitor all paging contexts created by the (Guest) operating system. HLAT enables a VMM to enforce the integrity of guest linear mappings without this high overhead.

This chapter describes a processor mechanism for the type of VMM described above consisting of:

- A **Hypervisor-managed Linear Address Translation (HLAT)** mechanism which uses an alternate IA paging structure managed in guest physical memory (for example, by a Secure Kernel) that contains guest linear to guest physical translations that the VMM/Secure Kernel wants to enforce.
- A new EPT control bit called **"Paging-Write"** specified in EPT leaf entries. The new bit specifies which guest physical pages hold HLAT or legacy IA paging structures so that the processor can use the Paging-Write as permission to perform A/D bit writes (instead of the software W permission in the EPTE). Typical usage for the Paging-Write bit is with the legacy EPT Write bit cleared. In PAE paging, the PDPTE does not have A/D bits, instead the 4K page-directory-pointer table page contains 4 PDPTR entries. Hence, in PAE paging, the processor ignores the PW bit of leaf entries of CR3 EPT walks. Software note: in this case, the VMM will need to monitor the page-directory-pointer table page for writes using EPT write permissions (or alternately the VMM can emulate the PDPTR load into the VMCS for the guest on a MOV CR3 by configuring VM Exit on load to CR3 in PAE paging).
- A new EPT control bit called **"Verify Paging-Write"** specified in EPT leaf entries (that refer to the final host physical page in the translation). The new bit specifies which guest physical pages should only be referenced via translation (guest) paging structures that are marked as Paging-writable under EPT.

6.2 VMCS CHANGES

A new 64-bit control field, "tertiary processor-based VM-execution controls", is defined. The encoding pair for this field is 00002034H/00002035H.

A new tertiary processor-based VM-execution control, "Enable HLAT", is defined. The bit position of this control is 1.

A new tertiary processor-based VM-execution control, "Enable Paging-Write", is defined. The bit position of this control is 2.

A new tertiary processor-based VM-execution control, "Enable Guest Paging Verification", is defined. The bit position of this control is 3.

If bit 17, "Activate tertiary controls", of the primary processor-based VM-execution controls is 0, the logical processor operates as if the "Enable HLAT", "Enable Paging-Write" and "Enable Guest Paging Verification" VM-execution controls are 0.

Note that the enable controls for "Enable Paging-Write" and "Enable Guest Paging Verification" are independent of the "Enable HLAT" control. If the processor based VM-execution control for "Enable Paging-Write" is clear, the processor operates as if the "Enable Guest Paging Verification" control is 0.

A new 64-bit control field, "Hypervisor-managed Linear Address Translation Pointer", is defined. The encoding pair for this field is 00002040H/00002041H. The structure of this field and the in-memory data structure referenced by the guest physical address embedded in this field are described in Section 6.6. This field is ignored if the processor-based VM-execution control "Enable HLAT" is clear.

6.3 CHANGES TO EPT PAGING-STRUCTURE ENTRIES

A control bit, "Paging Write", in EPT leaf paging-structure entries is defined; the bit position is 58.

A control bit, "Verify Paging-Write", in EPT leaf paging-structure entries is defined; the bit position is 57.

If the "Enable Paging-Write" VM-execution control (see Section 6.2) is 0, the "Paging Write" bit in the EPT leaf paging-structure entries is ignored and remains available to software. If the control is 1, the bit will be defined for all leaf EPT paging structures and used as described in Section 6.4.

If the "Enable Guest Paging Verification" VM-execution control (see Section 6.2) is 0, the "Verify Paging-Write" bit in the EPT leaf paging-structure entries is ignored and remains available to software. If the control is 1, the bit will be defined for all leaf EPT paging-structures that refer to the final host physical page in the translation and used as described in Section 6.4.

The new EPT control bits "Paging-Write" and "Verify Paging-Write" are enabled via tertiary processor-based VM-execution controls. Note that Verify Paging-Write (VPW) relies on Paging-Write (PW) for its interpretation, hence, if PW is disabled, VPW is ignored (and SW available as legacy) and the processor operates as if VPW is disabled, i.e., no EPT fault will occur due to VPW violations.

6.3.1 Reservation of a Guest Page Type in EPT Paging Structure Entry for Future Use

If either of the CET "Enable EPT Kernel Shadow Stack Control" EPT control or the HLAT "Enable Paging-Write" is disabled, then the control bits in the EPTE for the disabled control remain available to software. **(This is same as legacy behavior.)**

If both the CET "Enable EPT Kernel Shadow Stack Control" EPT control and the "Enable Paging-Write" are enabled together, then the encoding in the EPTE for both bits set (11b) may be used in the future for an additional guest page type if needed. Defining this encoding (11b) will need an explicit opt-in control in the future. Note that there is no special treatment for this encoding in the HLAT architecture.

Four Guest Page Types can be expressed via the CET "Enable EPT Kernel Shadow Stack Control" EPT control and the "Enable Paging-Write" EPT control; ordinary guest page (SSS=0b, PW=0b), guest kernel supervisor shadow stack (SSS=1b, PW=0b), guest paging structure (SSS=0b,PW=1b), and undefined (SSS=1b, and PW=1b).

It is the responsibility of the VMM (software) to avoid using the undefined (11b) settings of these two control bits, noting that, Kernel Shadow Stack is to be used by the VMM for the final EPTE in the EPT translation for Kernel shadow stack GPAs, whereas, Paging-Write is used for the final EPTE in the EPT translation of GPAs containing guest paging structures (not for the final page referenced through the guest paging structures). For such a configuration (11b) for SSS and PW specified by the VMM, the processor will enforce both Kernel Shadow Stack access semantics and Paging-write access semantics for those GPAs. See the table below for details.

Table 6-1. Kernel Shadow Stack and Paging-Write Access Details

Kernel Shadow Stack VMCS Control	Paging-Write VMCS Control	Kernel Shadow Stack EPTE Bit	Paging-Write EPTE Bit	EPT Misconfiguration
0	0	Ignored	Ignored	NA
0	1	Ignored	Paging-Write behavior	NA
1	0	Kernel Shadow Stack behavior	Ignored	NA
1	1	Kernel Shadow Stack behavior	Paging-Write behavior	NA: This setting may change in the future when this encoding is utilized for a new guest page type, via a new opt-in VMX control.

6.4 CHANGES TO VMX SUPPORT FOR ADDRESS TRANSLATION

If the logical processor is in VMX non-root operation with EPT enabled, and if the "Enable HLAT" VM-execution control (see Section 6.1) is 0, the translation from guest linear to host physical address is determined by the guest IA paging structures and the EPT paging structures. **(This is same as legacy behavior.)**

When the HLAT mechanism is enabled, a guest linear address is translated either through the HLAT paging structure or the "ordinary" paging structure (guest CR3-rooted paging structure). The processor makes this decision based on whether the guest linear address matches a Protected Linear Range (PLR); see Section 6.5. A PLR match causes the processor to translate the guest linear address through the HLAT paging structure; a mismatch causes the processor to translate the guest linear address based on legacy through the ordinary guest CR3-rooted paging structure. The PLR match is performed using a prefix-mask that is applied to the canonical form of the guest linear address and comparing the masked address against a prefix-match value on the masked canonical guest linear address. The Protected Linear Range is described in Section 6.5. Both paging structures are also walked via EPTs (EPT usage for address translation is the same as legacy). Based on the PLR match, a mapping for a guest linear address to a host physical address is only translated by the processor either through the HLAT IA paging structure

walk, or the ordinary guest CR3-rooted IA paging structure walk; and never both. It is not necessary for a guest linear address translation to be found in the HLAT paging structures (after a PLR match); this allows the VMM to enforce sparse guest supervisor linear address translations via HLAT. The VMM may handle the missing translations by the use of a control bit called a **"restart walk"** bit in the HLAT paging structures. When the processor translation of a guest linear address through the HLAT paging encounters a "restart walk", the processor aborts the walk and performs a legacy walk starting at the ordinary guest CR3-rooted paging structures.

For a PLR match, if the guest linear address translation via the HLAT walk succeeds and a mapping to a guest physical address is found in the HLAT, then the walk is completed successfully and the TLB is filled appropriately. However, an HLAT walk may not complete due to the following reasons:

- A guest linear address translation may be specified as not present in the HLAT paging structures. In this case, the walk is terminated and a page fault is reported to software with a Page-Fault Error Code indicator to indicate HLAT fault due to page not present.
- A guest linear address translation may encounter reserved bits set in the HLAT paging structures. In this case, the walk is terminated and a page fault is reported to software with a Page-Fault Error Code indicator to indicate HLAT fault due to reserved bit set.
- A guest linear address translation may be aborted by the processor encountering a "restart walk" control bit during the walk. In this case, the walk is restarted from the ordinary guest CR3-rooted paging structures. A translation may be found in the ordinary guest CR3-rooted paging structures and the processor response to those conditions is the same as legacy address translation through IA and EPT paging structures, i.e., either the TLB is filled or a page-fault or EPT violation is generated. If the TLB is filled after a restart, the processor ensures that the TLB page size used matches the page size at which the HLAT walk was aborted due to restart. Note that a guest linear address translation that starts at the HLAT paging structures and encounters a "restart walk" switches to ordinary CR3 address translation and cannot architecturally revert back to translation through the HLAT structures.

HLAT relieves the VMM from making guest physical pages that hold the ordinary guest CR3-rooted guest paging structures read-only under EPT. However, to meet the security objective for HLAT, the VMM must make the guest physical pages that hold the HLAT guest paging structures read-only under EPT; this restriction has no legacy compatibility restrictions. Processor page walk A/D bit updates occur as defined, or ordinary paging structures during guest linear address translation through HLAT paging structures. Per legacy behavior, these A/D bit writes will cause EPT violations if the guest paging structure guest physical pages are read only under EPTs. To avoid this performance overhead, a new EPT control bit "Paging-Write" is defined which can be enabled via the new tertiary VM execution control called "Enable Paging-Write". Guest physical pages that have the Paging-Write bit set under EPTs allow the processor page walker to perform A/D bit writes without EPT violations (even if the EPT entry Write permission is clear, effectively EPTE.W||EPTE.PW is used as the leaf EPTE Write permission by the processor). Software writes to guest physical pages are still subject to the EPT Write permission; Paging-Write is ignored for writes from software. Note that Paging-Write can be used for HLAT paging structures or ordinary paging structures irrespective of whether HLAT is enabled or not. Details of EPT violation behavior for Paging-Write is described in Section 6.8, Table 6-9.

Note that by using HLAT and Paging-Write the VMM can enforce guest linear address translation for specific guest linear addresses, however, it cannot enforce restricting guest linear address alias translations to guest physical addresses. The VMM can restrict the effect of aliases by making guest physical pages non-writable under EPTs. However, there may be scenarios where the VMM may wish to restrict aliases to writable guest-physical pages. To enable the VMM to restrict aliases, a new EPT control bit "Verify Paging-Write", is defined which can be enabled via the VM execution control called "Enable Guest Paging Verification". Guest physical pages that have the Verify-Paging-Write bit set under EPT cause the processor page walker to check that only guest physical pages that have the leaf EPT entry attribute Paging-Write were used to translate a guest linear address to that guest physical address. Note that Verify-Paging-Write can be used for HLAT paging structures or ordinary paging structures irrespective of if HLAT is enabled or not.

Specific conditions may cause an EPT violation with a new Exit Qualification bit described in Section 6.8 when "Verify Paging-Write" is enabled.

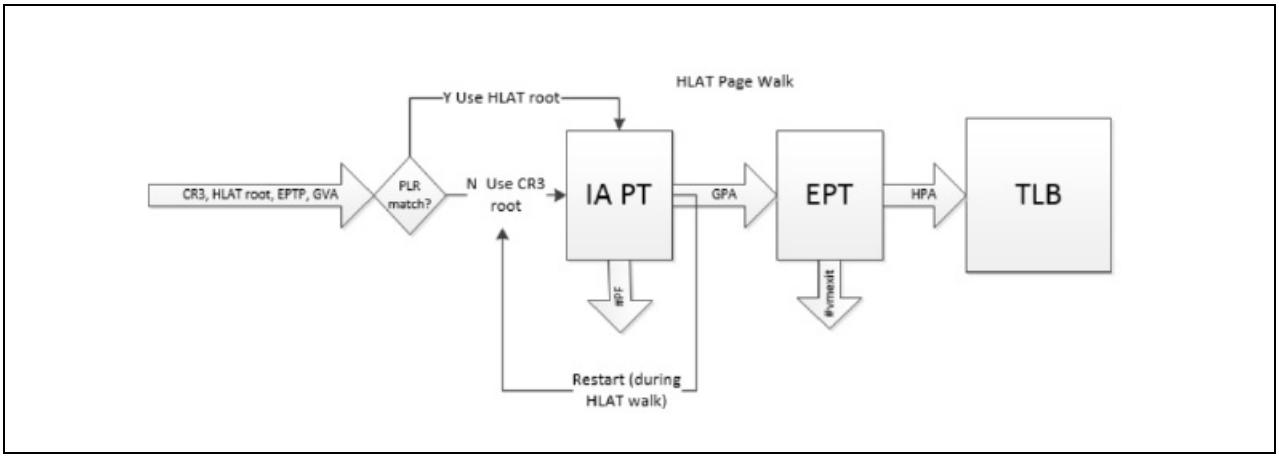


Figure 6-2. HLAT High Level Representation

6.5 PROTECTED LINEAR RANGE

The Protected Linear Range (PLR) is a range of guest linear address space for which the processor page walker performs address translation through the HLAT paging structures when the HLAT mechanism is enabled by the VMM. The PLR is specified using the following two control fields:

1. Bits 53:48 (currently reserved) of the IA32_VMX_EPT_VPID_CAP MSR (address 48CH) will enumerate the maximum allowed value for the HLAT prefix size. The HLAT prefix size holds a value between 0 and 52 and specifies the size of the all 1's MSB prefix (in number of bits) that the processor applies for testing the GLA for a PLR match. For example:
   - A value of 0 reported by the processor specifies that the processor will not apply a PLR prefix match (so if HLAT is enabled, all GLAs will be subject to HLAT lookup).
   - A value of 1 reported by the processor specifies that the processor supports a 1-bit MSB prefix, so GLA values of the canonical form with bit 63 set (and higher) will match the PLR (i.e., upper half of the guest linear address space), and hence will be subject to HLAT lookup.
   - A value of 52 reported by the processor specifies that the processor supports a 52-bit MSB prefix of all 1s (so if HLAT is enabled, the processor enforces HLAT lookup only for the last 4KB page in the guest linear address space).

   NOTE

   Initial implementations may report a 1-bit prefix width in this capability MSR and will not support 32-bit mode paging.

2. A new 16-bit VMCS control field is defined called "HLAT PLR Prefix Size". The VMCS index and encoding for this field is 00000006H. VMM software should program this field to specify the GLA MSB prefix to apply to test the GLA for a PLR match (to condition HLAT walks).

   This 16-bit control field holds a value between 0 and 52 specifying the size of the all 1's prefix (in number of bits) that the processor should apply for matching the GLA to the PLR. A value specified by software higher than what the processor enumerates in the HLAT prefix size value will be truncated by the processor, resulting in the enforcement of an address prefix of size specified by the capability MSR.

6.6 HYPERVISOR-MANAGED LINEAR ADDRESS TRANSLATION

The HLAT is referenced via a 64-bit control field called "Hypervisor-managed Linear Address Translation Pointer" (HLATP) which contains a 4K-aligned guest physical address. The HLAT is populated with translations for guest

linear addresses to guest physical addresses. The HLAT structure is identical for IA32-e (compatibility and 64-bit) modes of VMX-non-root (guest) software. Additionally, VA48 and LA57 are supported via the same HLAT structure format. For IA32 PAE and non-PAE modes (CR4.PAE=0) HLAT lookup is not performed even if the "Enable HLAT" VM-execution control is 1. Disabling paging when HLAT is enabled disables HLAT; the VMM should restrict such guest paging mode changes via CR exiting.

The format of the HLATP is shown in Table 6-2.

Table 6-2. Format of HLATP

Bit Position[1]	Contents
2:0	Reserved (0)
3	Page-level write-through (PWT); indirectly determines the memory type used to access the HLAT PML4 table during linear address translation (see Section "Paging and Memory Typing When the PAT is Supported" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).
4	Page-level cache disable (PCD); indirectly determines the memory type used to access the HLAT PML4 table during linear address translation (see Section "Paging and Memory Typing When the PAT is Supported" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).
11:5	Reserved (0)
N-1:12	Guest physical address of 4KB-aligned HLAT PLM4 table used for linear address translation (if LA57 enabled this is the guest physical Address of the 4KB-aligned HLAT PML5 table used for linear address translation).
63:N	Reserved (0)

NOTES:

1. N is the physical-address width supported by the processor.

Processor access to the HLAT data structures (in guest physical memory) will use the memory type that the MTRRs (memory-type range registers) and EPTs specify for the guest physical address of the access.

Software should ensure that the VMCS and referenced data structures are located at physical addresses that are mapped to WB memory type by the MTRRs.

6.6.1 HLAT Overview

HLAT is active when the "enable HLAT" VM-execution control is 1. The processor looks up the HLAT if, during a guest linear address translation, the guest linear address matches the Protected Linear Range (see Section 6.5). The lookup from guest linear addresses to the guest physical address and attributes is determined by a set of HLAT paging structures. Section 7.2 gives the details of the HLAT paging structures.

The guest paging structure managed by the guest OS specifies the ordinary translation of a guest linear address to the guest physical address and attributes that the guest ring-0 software has programmed, whereas HLAT specifies the alternate translation of the guest linear address to guest physical address and attributes that the Secure Kernel and VMM seek to enforce. A logical processor uses HLAT to translate guest linear addresses only when those guest linear addresses are used to access memory (both for code fetch and data load/store) and the guest linear addresses match the PLR programmed by the VMM/Secure Kernel.

6.6.2 Operation of HLAT

The HLAT translation mechanism uses bits 47:0 in PAE and IA32-e paging modes (or 56:0 in LA57 paging mode) of the guest linear address based on the paging mode of operation of the guest enforced by the VMM. Correspondingly, the HLAT is a 4-level (or 5-level) hierarchical structure.

HLAT structures are accessed to translate a given guest linear address. 48 (or 57) bits of the guest linear address are always used by the logical processor to traverse the HLAT structures as follows:

A 4KB naturally aligned HLAT L5 table is located at the guest physical address specified in bits 51:12 of the "Hypervisor-managed Linear Address Translation Pointer", a 64-bit VM-execution control field. An HLAT L5 table comprises 512 64-bit entries (HLAT L5Es). An HLAT L5E is selected at the guest physical address defined as follows:

- Bits 63:52 are all 0.
- Bits 51:12 are from the HLATP.
- Bits 11:3 are bits 56:48 of the guest linear address.
- Bits 2:0 are all 0.

6.6.3 Format of the HLAT L5E

The format of the HLAT L5E is similar to the format of the ordinary PML5E for the IA paging structure managed by the guest OS. The changes/additions are noted below.

Table 6-3. Format of HLAT L5E

Bit Position	Usage in Ordinary Paging	Usage in HLAT Paging Structures
11	Ignored	Restart If entry is present and this bit is 1, specifies that a page-walk hitting this non-leaf entry must stop and restart the walk from the guest CR3-rooted ordinary paging structure. The Accessed (A) bit is also defined and remaining bits are ignored. If 0, specifies that the walk should continue (and all bits are treated like ordinary paging).

A 4KB naturally aligned HLAT L4 table is located at the guest physical address specified in bits N:12 of the HLAT L5E. An HLAT L4 table comprises 512 64-bit entries (HLAT L4Es). An HLAT L4E is selected at the guest physical address defined as follows:

- Bits 63:52 are all 0.
- Bits 51:12 are from the HLAT L4E.
- Bits 11: 1 3 are bits 47:39 of the guest linear address.
- Bits 2:0 are all 0.

6.6.4 Format of the HLAT L4E

The format of the HLAT L4E is similar to the format of the ordinary PML4E for the IA paging structure managed by the guest OS. The changes/additions are noted below.

Table 6-4. Format of HLAT L4E

Bit Position	Usage in Ordinary Paging	Usage in HLAT Paging Structures
11	Ignored	Restart If entry is present and this bit is 1, specifies that a page-walk hitting this non-leaf entry must stop and restart the walk from the guest CR3-rooted ordinary paging structure. The Accessed (A) bit is also defined and remaining bits are ignored. If 0, specifies that the walk should continue and all bits are treated like ordinary paging.

A 4KB naturally aligned HLAT L3 table is located at the guest physical address specified in bits N:12 of the HLAT L4E. An HLAT L3 table comprises 512 64-bit entries (HLAT L3Es). An HLAT L3E is selected at the guest physical address defined as follows:

- Bits 63:52 are all 0.
- Bits 51:12 are from the HLAT L4E.
- Bits 11:3 are bits 38:30 of the guest linear address.
- Bits 2:0 are all 0.

6.6.5 Format of the HLAT L3E

The HLAT L3E may contain a mapping to a guest physical address of a 1GB page or may contain a reference to an HLAT L2 Table (via a guest physical address of a 4KB naturally aligned address). In either case, the format of the HLAT L3E is similar to the format of the ordinary PDPTE for the IA paging structure managed by the guest OS. The changes/additions are noted below.

Table 6-5. Format of HLAT L3E

Bit Position	Usage in Ordinary Paging	Usage in HLAT Paging Structures
11	Ignored	Restart If entry is present and this bit is 1, specifies that a page-walk hitting this non-leaf entry must stop and restart the walk from the guest CR3-rooted ordinary paging structure. The Accessed (A) bit is also defined and remaining bits are ignored. If 0, specifies that the walk should continue and all bits are treated like ordinary paging.

A 4KB naturally aligned HLAT L2 table is located at the guest physical address specified in bits N:12 of the HLAT L3E. An HLAT L2 table comprises 512 64-bit entries (HLAT L2Es). An HLAT L2E is selected at the guest physical address defined as follows:

- Bits 63:52 are all 0.
- Bits 51:12 are from the HLAT L3E.
- Bits 11:3 are bits 29:21 of the guest linear address.
- Bits 2:0 are all 0.

6.6.6 Format of the HLAT L2E

The HLAT L2E may contain a mapping to a guest physical address of a 2MB page or may contain a reference to an HLAT L1 Table (via a guest physical address of a 4KB naturally aligned address). In either case, the format of the HLAT L2E is similar to the format of the ordinary PDE for the IA paging structure managed by the guest OS. The changes/additions are noted below.

Table 6-6. Format of HLAT L2E

Bit Position	Usage in Ordinary Paging	Usage in HLAT Paging Structures
11	Ignored	Restart If entry is present and this bit is 1, specifies that a page-walk hitting this non-leaf entry must stop and restart the walk from the guest CR3-rooted ordinary paging structure. The Accessed (A) bit is also defined and remaining bits are ignored. If 0, specifies that the walk should continue and all bits are treated like ordinary paging.

A 4KB naturally aligned HLAT L1 table is located at the guest physical address specified in bits N:12 of the HLAT L2E. An HLAT L1 table comprises 512 64-bit entries (HLAT L1Es). An HLAT L1E is selected at the guest physical address defined as follows:

- Bits 63:52 are all 0.
- Bits 51:12 are from the HLAT L2E.
- Bits 11:3 are bits 20:12 of the guest linear address.
- Bits 2:0 are all 0.

6.6.7 Format of the HLAT L1E

The HLAT L1E contains a mapping to a guest physical address of 4KB pages. The format of the HLAT L1E is similar to the format of the ordinary PTE for the IA paging structure managed by the guest OS. The changes/additions are noted below.

Table 6-7. Format of HLAT L1E

Bit Position	Usage in Ordinary Paging	Usage in HLAT Paging Structures
11	Ignored	Restart If entry is present and this bit is 1, specifies that a page-walk hitting this non-leaf entry must stop and restart the walk from the guest CR3-rooted ordinary paging structure. The Accessed (A) bit is also defined and remaining bits are ignored. If 0, specifies that the walk should continue and all bits are treated like ordinary paging.

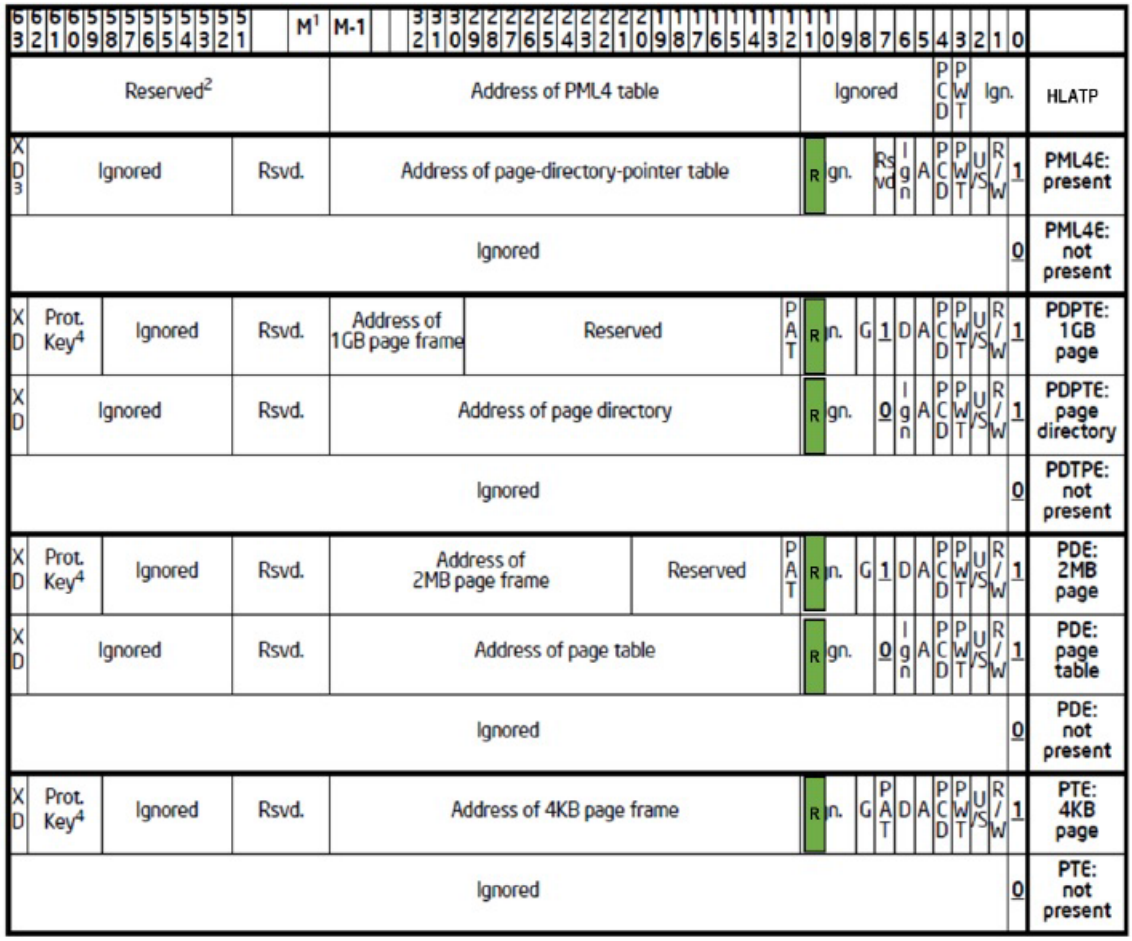


Figure 6-3. New Bit in the IA-32e Paging Structures Recognized During HLAT Walks[1]

NOTES:

1. This bit remains ignored in ordinary page walks to translate a guest linear address.
2. M is an abbreviation for MAXPHYADDR.
3. Reserved fields must be 0.
4. If IA32_EFER.NXE = 0 and the P flag of a paging-structure entry is 1, the XD flag (bit 63) is reserved.
5. If CR4.PKE = 0, the protection key is ignored.

6.6.8 HLAT Faults

When the "Enable HLAT" VM-execution control is 1, guest linear addresses that match the PLR criteria are translated through the HLAT to enforce Secure Kernel/VMM-specified translation. HLAT lookup by the processor can generate page-faults due to not-present or when the translation does not permit the access. If an HLAT page fault does not occur, then the TLB caches the guest linear to host physical combined mapping and permissions derived from the page walk to allow the memory access.

If a page fault occurs due to an HLAT mapping not present or a reserved bit violation during an HLAT walk, the processor sets the "HLAT Fault" (bit 7) in the PFEC reported for the page fault exception.

For HLAT page walks which encounter an HLAT entry with P=R=1, and other reserved bits set, the reserved bit check faults are a higher priority than the restart operation and will be reported to software through a page fault with PFEC bits set as described below.

The PFEC bit 7 flag is set (1) if, the exception resulted during translation of a guest linear address, and:

1. VM-execution control: "enable HLAT" = 1, and
2. The access caused a page-fault exception (for code or data access) during HLAT lookup implying that the linear address matched the PLR criteria.

When this PFEC bit 7 flag is set (1):

- The P flag (bit 0) is cleared (0) implying there is no translation in the HLAT for the linear address because the P flag was 0 in one of the HLAT paging structure entries used to translate that linear address.
- OR The RSVD flag (bit 3) is 1

AND

- PK flag (bit 5) and SGX flag (bit 15) are 0 for both cases.
- I/D (bit 4), R/W (bit 1), U/S (bit 2) and CET SSS (bit 6) flags should be set appropriately.

Note that no PFEC bit is set for faults generated when the HLAT translation does not permit the access (during lookup or from cached HLAT mappings). To differentiate faults due to insufficient permissions in HLAT, the VMM can leverage EPT permissions thus causing an EPT violation or using Virtualization Exceptions, the VMM can generate a #VE exception for page accesses violating the EPT permissions, thus differentiating permission violations reported to the OS from legacy page faults due to a permission violation from ordinary paging (via CR3-rooted paging structures).

6.6.9 HLAT Operation

This section describes the operation of the HLAT lookup and the fault conditions that may occur.

If at the beginning of a guest linear address translation, the PLR matches:

1. An HLAT entry is read (initially, an HLAT L5 entry). A nested walk of the EPT structures is performed to complete this read (memory type is derived from HLAT page walk, and EPT memory type). The EPT walk may lead to an EPT violation which aborts the walk. If the HLAT entry is read successfully:
   a. If the entry is not present, then the HLAT walk is deemed complete and a page fault is reported with the HLAT Fault (bit 7) PFEC bit set.
   b. Instead, if the entry is present but its contents are not configured properly (a reserved bit is set), the HLAT walk is deemed complete and a page fault is reported with the HLAT Fault (bit 7) PFEC bit set.
   c. If the entry is present and the restart (bit 11) is set, the HLAT walk is aborted; the Accessed (A) bit is updated and the remaining HLAT PxE bits are ignored; the page walk is restarted from the guest CR3 rooted page table structure, with the same (or smaller page) fragmentation page size as where the restart occurred; from this point the walk is a legacy nested page table walk. Note that the page size restriction is important to enforce that there is no larger page size mappings in ordinary page tables that supersedes a smaller page size mapping in the HLAT page table. (See Table 6-8.)
   d. If the entry is present and its contents are configured properly, operation depends on whether the entry references another HLAT structure:
      - If the entry does reference another HLAT structure, an entry from that structure is accessed; step 1 is executed for that other entry.
      - Otherwise, the entry is used to produce the guest physical address and permissions; step 2 is executed.
2. The guest physical address and attributes are determined from the HLAT entry and the last stage of the nested EPT walk is attempted to determine the final host physical page address and effective permissions. If no EPT violation occurs then the TLB fill is completed.

Table 6-8. HLAT Page Size Mapping

For an address that matches PLR, Page Size/Level at which "restart" occurred in HLAT paging structure	Page Size at which IA ordinary walk completed successfully (after restart at size from column 1)	Page size cached in TLB (Note: nested EPT page walk may further fragment page mapping - EPT cannot coalesce)
512GB	1GB	1GB
	2MB	2MB
	4KB	4KB
1GB	1GB	1GB
	2MB	2MB
	4KB	4KB
2MB	1GB	2MB
	2MB	2MB
	4KB	4KB
4KB	1GB	4KB
	2MB	4KB
	4KB	4KB

6.6.10 HLAT Interaction with IA and EPT A/D

In ordinary paging, processor implementations cache information from PTE into PxE caches only after updating the Accessed (A) bit in the PTE. **(This is legacy behavior.)**

If an HLAT entry is present (P=1) and is specified with restart (R=1), then in addition to those two bits, the Accessed (A) bit is also defined and the remaining HLAT PxE bits are ignored. Similar to ordinary paging, Accessed (A) bits in HLAT will be updated for any present PxE (including R=1) before the PxE entry is cached.

In ordinary paging, on a page walk due to a memory write, if the ordinary PTE is present (P=1) and write=0, the processor reports a page fault; or, if present (P=1) and write=1 and PTE is a leaf, the processor sets the D bit on the leaf entry. **(This is legacy behavior.)**

In HLAT paging, if an HLAT entry is present (P=1) and not restart (R=0), then the behavior for fault generation and D bit setting is the same as in ordinary paging (for both write=0 and write=1). However, in HLAT paging, if the entry is present (P=1) and restart (R=1) then the rest of the bits (other than A) are ignored.

- For an HLAT page walk due to a write, where an HLAT PTE higher in the hierarchy specifies cumulative write=0, processor fault will be suppressed (since walk is restarted).
- If the HLAT entry is a leaf entry (D bit occurs only at PTE level which is not cached in any PxE structure), the processor suppresses the dirty bit assist (A bit will still be set on all PxE structures).

IA A/D updates may occur in the HLAT paging structures and corresponding EPT A/D updates may occur even if HLAT paging structures indicate a restart.

6.6.11 Cached HLAT Derived Information

Information derived from page walks may be cached by the processor. Paging structure intermediate information may be cached in PxE caches and final combined mappings derived from the guest linear address translation (page walks) may be cached in the processor TLBs. Any modification to the information specified in IA paging entries may be invalidated using legacy operations such as MOV CR3, INVLPG (from VMX non-root mode), and INVVPID (from VMX root mode). **(This is legacy behavior.)**

From the TLB perspective, legacy behavior is not modified for invalidating information cached that is derived from HLAT paging structures. There is no change to information cached in the processor TLBs since HLAT enforces that a single mapping is found for a specific guest linear address; either from the HLAT paging structure or from the guest CR3 rooted paging structure and not both.

Similarly, ASID management is not modified due to HLAT. If PCID is enabled by the guest, PCID always derives from guest CR3 (not HLAT root) and follows the legacy approach. If HLAT mappings are global they will be treated with global ASID; if HLAT mappings are not global, they will map to ASID determined by VPID/EPTP/PCID per legacy behavior.

The paging structure caches (PxE caches) on the other hand, hold additional information that modifies the page walk, hence the following changes are required to the PxE caches:

1. The restart bit is cached as a new output bit from the PxE cache lookup during page walks. A page walk that hits the PxE cache and results in an entry for which the restart bit is cached aborts the walk, and causes the walk to restart from the guest CR3 rooted guest physical address (which may hit other entries in the PxE cache).
2. An HLAT tag bit is provided as input to the PxE lookup to ensure that the walk is performed in HLAT mode until a restart is encountered.

6.7 CHANGES TO GUEST PHYSICAL ACCESSES

EPT permission violations due to accumulated memory read/write/execute permission violations are reported as EPT violation VM Exits. If a logical processor is in VMX non-root operation with EPT enabled, and if the "Enable Paging-Write" VM-execution control (see Section 6.1) is 0, an EPT violation occurs if a write access using a guest-physical address and the write-access bit (bit 1) was clear in any of the EPT paging structure entries used to translate the guest-physical address. This is true for software accesses as well as processor page walker accesses when performing A/D bit updates during page walks through the IA paging structures. **(This is legacy behavior.)**

When HLAT is enabled, the guest physical addressed HLAT paging structures must be readable and writable by the processor to perform translations when the guest kernel is executing; this requires the guest paging structures to be read-write under EPTs. The write accesses include Paging-Write accesses, which are the following:

- Writes by a logical processor to update accessed and dirty flags in a guest paging-structure entry.
- If bit 6 of the EPT pointer (EPTP) is 1 (enabling accessed and dirty flags for EPT), reads by the logical processor of a guest paging-structure entry use it to translate a linear address. (This does not apply to loads of the PDPTE registers by the MOV to CR instruction for PAE paging; such loads of guest PDPTEs are never treated as writes.)

However from a security perspective, HLAT paging structures cannot be made writable under EPTs to protect it from software writes from an untrusted guest. Thus, per legacy behavior, A/D bit updates on the HLAT paging structures would invoke EPT violations that the VMM would have to emulate. To avoid this performance burden, a new access permission bit is proposed in the EPT paging structure leaf entries called "Paging-Write" (PW) access. The bit position is bit 58, but would be chosen from among those that are currently ignored by the processor and available to software. If the "Enable Paging-Write" VM-execution control is 0, the paging-write bits in the EPT leaf paging-structure entries are ignored and remain available to software.

Pages that have EPTE "Paging-Write" set also modify previous architecture with EPT Accessed and Dirty bits, since with legacy architecture, when EPT A/D bits are enabled, processor paging accesses are treated as writes, and the hypervisor would not be able to make guest paging structure pages non-writable for guest software without affecting processor page walks. When "Enable Paging-Write" is set, the hypervisor can make GPAs containing guest HLAT paging structures non-writable for software, and allow write accesses by the processor's paging architecture to update A/D bits.

If the "Enable Paging-Write" VM-execution control is 1, a paging-write access using a guest physical address will not cause an EPT violation if the write-access bit is 1 in all of the non-leaf entries and the paging-write-access bit is 1 in the leaf EPT paging-structure entry used to translate the guest-physical address (in this specific case, the write-access bit in the leaf EPT entry may be 0 or 1 since write EPT permission for leaf entry when Paging-Write is enabled is W | PW). When EPT leaf entry paging-write-access is 0, other cases that cause EPT violations remain unchanged (see table below).

To further illustrate, this table shows the new behavior in **bold** for writes by a logical processor to update accessed/dirty flags in a guest paging-structure entry.

Table 6-9. EPT Violation Behavior

L4 EPT W	L3 EPT W	L2 EPT W	L1 EPT W	L1 EPT PW	(Write-access bit is 1 in all of the non-leaf EPT entries) and (the paging-write-access bit is 1 in the leaf EPT paging-structure entry)	EPT Violation Behavior
0	X	X	X	X	0	EPT violation (same as legacy behavior)
X	0	X	X	X	0	EPT violation (same as legacy behavior)
X	X	0	X	X	0	EPT violation (same as legacy behavior)
1	1	1	0	0	0	EPT violation (same as legacy behavior)
1	1	1	0	1	1	**No EPT violation (new behavior)**

An EPT misconfiguration will occur if, for the translation of a guest-physical address, the paging-write-access bit is 1 and the read access bit (EPTE bit 0) is clear (0) in the leaf EPT paging-structure entry used to translate the guest-physical address.

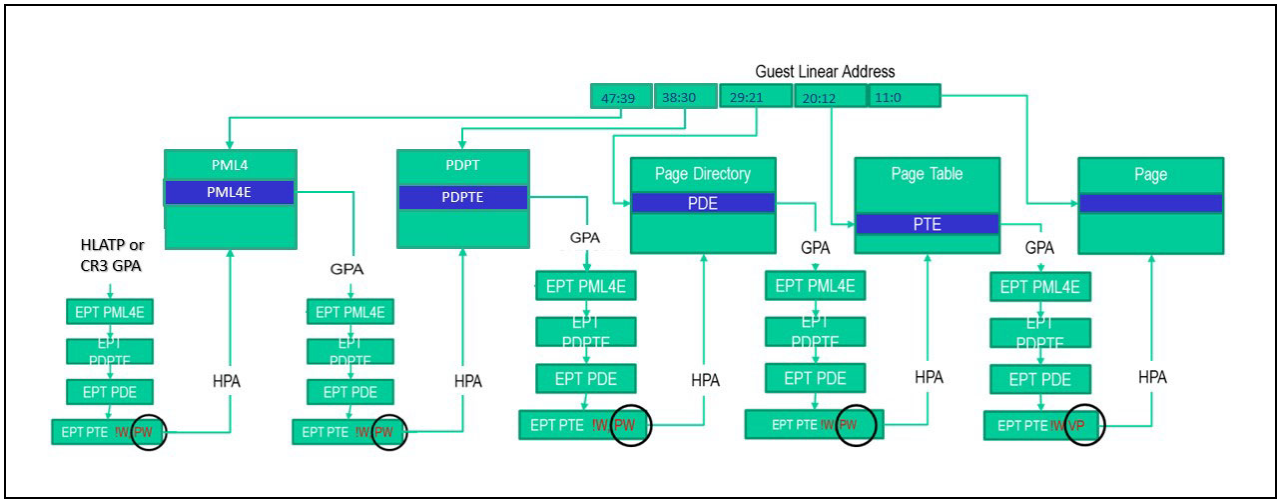


Figure 6-4. Example of Paging-Write and Verify-Paging-Write EPT Control Bits Used for Guest Paging Structures (HLAT or Ordinary Paging)

6.7.1 Paging-Write Interaction with EPT A/D

Software can enable accessed and dirty flags for EPT using bit 6 of the extended page-table pointer (EPTP). If this bit is 1, the processor will set the accessed and dirty flags for EPT. Whenever there is a write to a guest-physical address, the processor sets the dirty flag (if it is not already set) in the EPT paging-structure entry that identifies the final physical address for the guest-physical address (either an EPT PTE or an EPT paging-structure entry in which bit 7 is 1). In addition, when accessed and dirty flags for EPT are enabled, processor accesses to guest paging structure entries are treated as writes. Thus, such an access will cause the processor to set the dirty flag in the leaf EPT paging-structure entry that identifies the final physical address of the guest paging-structure entry. **(This is legacy behavior.)**

When the "Enable Paging-Write" VM-execution control is enabled, and the hypervisor makes guest paging structures non-writable for OS software writes via EPT permissions, then, for inter-operation with EPT Accessed/Dirty bits, the hypervisor must set the Paging-Write access EPT entry control bit (see Section 6.3) to allow the processor paging architecture to safely consider guest paging structure accesses as writable for the page walker (while continuing to enforce the writability for software based on the legacy EPTE.W bit).

6.7.2 IOMMU Interaction

During IOMMU SVM page walks, A/D bits are updated. During these writes, if VT-d paging structures disallow writes on the guest physical addresses that contain the SVM paging structures then an IOMMU page fault is generated. **(This is legacy behavior.)**

Paging-Write does not introduce any change to this legacy behavior. The "Paging-Write" and "Verify Paging-Write" bits are ignored by IOMMU (during nested VT-d page walk). An implication of this is that aliases can be created to guest physical address via SVM page tables. The VMM should protect such guest physical pages via EPT permissions (read-only or read-execute or execute-only). An attacker cannot create a spurious SVM page table structure using HLAT paging structures to cause A/D writes to HLAT paging structures via the IOMMU (since those write attempts will fail due to the VMM setting HLAT paging structures as EPTE.W=0, PW=1). For pages that are writeable under EPTs, an alias via SVM is possible which can be addressed in the future by restricting IOMMU writes to guest physical pages marked "Verify Paging-Write" under VT-d.

6.8 ADDITION TO EPT VIOLATION EXIT QUALIFICATION

Guest physical pages that have the "Verify Paging-Write" bit set under EPT cause the processor page walker to check that only guest physical pages (containing IA paging structures) that have the leaf EPT entry attribute "Paging-Write" were used to translate a guest linear address to that guest physical address.

Note that "Verify Paging-Write" can be used for HLAT paging structures or ordinary paging structures irrespective of whether HLAT is enabled or not.

When all the following conditions are met, an EPT violation is reported to the VMM with a new Exit Qualification bit (position 15) set to indicate an EPT violation due to "Verify Paging-Write":

- "Paging-Write" and "Verify Paging-Write" are both enabled in VM execution controls, and the final (leaf) guest physical page was specified to be "Verify Paging-write" under EPT.
- At least one of the guest physical pages accessed by the processor page walker for the translation of a guest linear address was specified with the "Paging-Write" bit set to 0 under EPT.
- The page walk completed without a page fault. If the page walk was an HLAT page walk, none of the paging entries accessed during the HLAT walk had the restart bit set to 1.

Note that other EPT violation qualifications bits (for example due to permission violation) may be set along with bit position 15.

6.9 HLAT INTERACTION WITH INTEL® SGX

There are no special requirements for HLAT interaction with Intel® SGX.

HLAT applies to guest linear address translations which match the PLR as specified in this document. Enclave linear addresses are asserted to map to the EPCM region by the processor after legacy page walk is completed. Enclave linear range covers user mode linear addresses, and any such accesses that fall within the PLR will be subject to HLAT lookup as defined in this document. For HLAT page faults occurring during in enclave mode, no additional fault information is saved in SSA (same as other fault conditions). For EPT violation VM exits that occur during HLAT page walks, the in-enclave interruptibility bit will be set (same as other VM exit conditions).

6.10 HLAT INTERACTION WITH NESTED VT-X

There are no special requirements for nested VT-x operation for HLAT.

Since the HLAT structures are guest physical addressed when EPTs are enabled, a root hypervisor that exposes HLAT to a guest hypervisor must shadow EPT structures and, therefore, encompasses any HLAT structures created by the nested guest VMs (of the guest hypervisor). If the root or guest VMM enables interception of #PF with PFEC_MASK PFEC.HLAT fault bit 7 set, then the root VMM may have to inject the #PF VMExit for the Guest VMM (after handling it for itself if needed).

6.11 CHANGES TO VM ENTRIES

If the "activate tertiary controls" and "Enable HLAT" VM-execution controls are both 1, VM entries ensure that the "Enable EPT" VM-execution control is 1. The "HLAT Pointer" control field is checked for consistency per Section 6.4.

If the "activate tertiary controls" and "Enable Paging-Write" VM-execution controls are both 1, VM entries ensure that the "Enable EPT" VM-execution control is 1.

If the "activate tertiary controls" and "Enable Guest Paging Verification" VM execution controls are both 1, VM entries ensure that the "Enable EPT" VM execution control is 1.

VM entry fails if the above checks fail. When such a failure occurs, control is passed to the next instruction, RFLAGS.ZF is set to 1 to indicate the failure, and the VM-instruction error field is loaded with value 7, indicating "VM entry with invalid control field(s)." This check may be performed in any order with respect to other checks on VMX controls and the host-state area. Thus, different processors may give different error numbers for the same VMCS.

6.12 CHANGES TO VMX CAPABILITY REPORTING

Section 6.1 specified that a new bit of the tertiary processor-based VM-execution controls would be defined as "Enable HLAT" VM-execution control. A processor that supports the 1-setting of the control sets bit 1 of the IA32_VMX_PROCBASED_CTLS3 MSR (address 492H). RDMSR of that MSR returns 1 in bit 1 of EDX.

Section 6.1 specified that a new bit of the tertiary processor-based VM-execution controls would be defined as "Enable Paging-Write" VM-execution control. A processor that supports the 1-setting of the control sets bit 2 of the IA32_VMX_PROCBASED_CTLS3 MSR (address 492H). RDMSR of that MSR returns 1 in bit 2 of EDX.

Section 6.1 specified that a new bit of the tertiary processor-based VM-execution controls would be defined as "Enable Guest Paging Verification" VM-execution control. A processor that supports the 1-setting of the control sets bit 3 of the IA32_VMX_PROCBASED_CTLS3 MSR (address 492H). RDMSR of that MSR returns 1 in bit 3 of EDX.

CHAPTER 7
ARCHITECTURAL LAST BRANCH RECORDS (LBRS)

Architectural Last Branch Records (LBRs) enable recording of software path history by logging taken branches and other control flow transfers within processor registers. Each LBR record or entry is comprised of three MSRs:

- IA32_LBR_x_FROM_IP — Holds the source IP of the operation.
- IA32_LBR_x_TO_IP — Holds the destination IP of the operation.
- IA32_LBR_x_INFO — Holds metadata for the operation, including mispredict, TSX, and elapsed cycle time information.

The number of LBR records available varies across processor generations, and is specified in CPUID (see Section 7.4).

LBR records are stored in age order. The most recent LBR entry is stored in IA32_LBR_0_*, the next youngest in IA32_LBR_1_*, and so on. When an operation to be recorded completes (retires) with LBRs enabled (IA32_LBR_CTL.LBREn=1), older LBR entries are shifted in the LBR array by one entry, then a record of the new operation is written into entry 0. See Section 7.1.1 for the list of recorded operations.

The number of LBR entries available for recording operations is dictated by the value in IA32_LBR_DEPTH.DEPTH. By default, the DEPTH value matches the maximum number of LBRs supported by the processor, but software may opt to use fewer in order to achieve reduced context switch latency. See Section 7.3.1 for more details.

In addition to the LBRs, there is a single Last Event Record (LER). It records the last taken branch preceding the last exception, hardware interrupt, or software interrupt. Like LBRs, the LER is comprised of three MSRs (IA32_LER_FROM_IP, IA32_LER_TO_IP, IA32_LER_INFO), and is subject to the same dependencies on enabling and filtering.

Which operations are recorded in LBRs depends upon a series of factors:

- Branch Type Filtering — Software must opt in to the types of branches to be logged; see Section 7.1.2.3.
- Current Privilege Level (CPL) — LBRs can be filtered based on CPL; see Section 7.1.2.5.
- LBR Freeze — LBR and LER recording can be suspended by setting IA32_PERF_GLOBAL_STATUS.LBR_FRZ to 1. See Section 17.4.7 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B* for details on LBR_FRZ.

On some implementations, recording LBRs may require constraining the number of operations that can complete in a cycle. As a result, on these implementations, enabling LBRs may have some performance overhead.

7.1 BEHAVIOR

7.1.1 Logged Operations

LBRs can log most control flow transfer operations.

The source IP recorded for a branch instruction is the IP of that instruction. For events which take place between instructions, the source IP recorded is the IP of the next sequential instruction.

The destination IP recorded is always the target of the branch or event, the next instruction that will execute.

The full list of operations and the respective IPs recorded is shown in Table 7-1.

Table 7-1. LBR IP Values for Various Operations

Operation	FROM_IP	TO_IP
Taken Branch[1], Exception, INT3, INTn, INTO, TSX Abort	Current IP	Target IP
Interrupt	Next IP	Target IP
INIT (BSP)	Next IP	Reset Vector
INIT (AP) + SIPI	Next IP	Sipi Vector
EENTER/ERESUME + EEXIT/AEX	Current IP	Target or Trampoline IP
RSM[2]	Target IP	Target IP
#DB, #SMI, VM exit, VM entry	None	None

NOTES:

1. Direct CALLs to the next sequential IP, known as zero-length CALLs, are not treated as taken branches by LBRs.
2. RSM is only recorded in LBRs when IA32_DEBUGCTL.FREEZE_WHILE_SMM is set to 0.

7.1.2 Configuration

7.1.2.1 Enabling and Disabling

LBRs are enabled by setting IA32_LBR_CTL.LBREn to 1.

Some operations, such as entry to a secure mode like SMM or SGX, can cause LBRs to be temporarily disabled. Other operations, such as debug exceptions or some SMX operations, disable LBRs and require software to re-enable them. Details on these interactions can be found in Section 7.1.4.

7.1.2.2 LBR Depth

The number of LBRs used by the processor can be constrained by modifying the IA32_LBR_DEPTH.DEPTH value. DEPTH defaults to the maximum number of LBRs supported by the processor. Allowed DEPTH values can be found in CPUID.(EAX=01CH, ECX=0):EAX[7:0].

Reducing the LBR depth can result in improved performance, by reducing the number of LBRs that need to be read and/or context switched.

On a software write to IA32_LBR_DEPTH, all LBR entries are reset to 0. LERs are not impacted.

A RDMSR or WRMSR to any IA32_LBR_x_* MSRs, such that x ≥ DEPTH, will #GP fault. Note that the XSAVES and XRSTORS instructions access only the LBRs associated with entries 0 to DEPTH-1, see Section 7.3.1 for details.

By clearing the LBR entries on writes to IA32_LBR_DEPTH, and forbidding any software writes to LBRs ≥ DEPTH, it is thereby guaranteed that any LBR entries equal to or above DEPTH will have value 0.

7.1.2.3 Branch Type Enabling and Filtering

Software must opt in to the types of branches that are desired to be recorded. These elections are made in IA32_LBR_CTL; see Section 7.2. Branch type options are listed in Table 7-2; only those enabled will be recorded.

Table 7-2. Branch Type Filtering Details

Branch Type	Operations Recorded
JCC	Jcc, J*CXZ, and LOOP*
NEAR_IND_JMP	JMP r/m*
NEAR_REL_JMP	JMP rel*
NEAR_IND_CALL	CALL r/m*
NEAR_REL_CALL	CALL rel* (excluding CALLs to the next sequential IP)
NEAR_RET	RET (0C3H)
OTHER_BRANCH	JMP/CALL ptr*, JMP/CALL m*, RET (0C8H), SYS*, interrupts, exceptions, IRET, INT3, INTn, INTO, TSX Abort, EENTER, ERESUME, EEXIT, AEX, INIT, SIPI, RSM, breakpoints

These encodings match those in IA32_LBR_x_INFO.BR_TYPE.

Control flow transfers that are not recorded include #DB, VM exit, VM entry, and #SMI.

7.1.2.4 Call-Stack Mode

The LBR array is, by default, treated as a ring buffer that captures control flow transitions. However, the finite depth of the LBR array can be limiting 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. These calls to leaf functions, and their returns, are likely to displace the main execution context from the LBRs.

When Call-Stack mode is enabled, the LBR array can capture unfiltered call data normally, but as return instructions are executed the last captured branch (call) record is flushed from the LBRs in a last-in first-out (LIFO) manner. Thus, branch information pertaining to completed leaf functions will not be retained, while preserving the call stack information of the main line execution path.

Call-Stack mode is enabled by setting IA32_LBR_CTL.CALL_STACK to 1. When enabled, near RET instructions receive special treatment. Rather than adding a new record in LBR_0, a near RET will instead "pop" the CALL entry at LBR_0 by shifting entries LBR_1..LBR_[DEPTH-1] up to LBR_0..LBR_[DEPTH-2], and clearing LBR_[DEPTH-1] to 0. Thus, LBR processing software can consume only valid call-stack entries by reading until finding an entry that is all zeros.

Call-stack mode should be used with branch type enabling configured to capture only CALLs (NEAR_REL_CALL and NEAR_IND_CALL) and RETs (NEAR_RET). When configured in this manner, the LBR array emulates a call stack, where CALLs are "pushed" and RETs "pop" them off the stack. If other branch types (JCC, NEAR_*_JMP, or OTHER_BRANCH) are enabled for recording with call-stack mode, LBR behavior may be undefined.

It is recommended that call-stack mode be used along with CPL filtering, by setting at most one of the OS and USR bits in the IA32_LBR_CTL MSR. LBR call-stack mode does not emulate the stack switch that can occur on CPL transitions, and hence monitoring all CPLs may result in a corrupted LBR call stack.

Call-Stack Mode and LBR Freeze

When IA32_DEBUGCTL.FREEZE_LBRS_ON_PMI=1, IA32_PERF_GLOBAL_STATUS.LBR_FRZ will be set to 1 when a PMI is pended. That will cause LBRs and LERs to cease recording branches until LBR_FRZ is cleared. Because there may be some "skid", or instructions retiring, in between the PMI being pended and the PMI being taken, it is possible that some branches may be missing from the LBRs. In the case of call-stack mode, if a CALL or RET is missed, that can lead to confusing results where CALL entries fail to get "popped" off the stack, and RETs "pop" the wrong CALLs.

An alternative is to clear FREEZE_LBRS_ON_PMI, and instead utilize CPL filtering to limit LBR recording to ring3. This will record branches in the "skid", but avoid recording any branches in the ring0 handler.

7.1.2.5 CPL Filtering

Software must opt in to which CPL(s) will have branches recorded. If IA32_LBR_CTL.OS=1, then branches in CPL=0 can be recorded. If IA32_LBR_CTL.USR=1, then branches in CPL>0 can be recorded. For operations which

change the CPL, the operation is recorded in LBRs only if the CPL at the end of the operation is enabled for LBR recording. In cases where the CPL transitions from a value that is filtered out to a value that is enabled for LBR recording, the FROM_IP address for the recorded CPL transition branch or event will be 0FFFFFFFFFFFFFFFH.

7.1.3 Record Data

7.1.3.1 IP Fields

The source and destination IP values in IA32_LBR_x_[FROM|TO]_IP and IA32_LER_x_[FROM|TO]_IP may hold effective IPs (EIPs) or linear IPs (LIPs), depending on the processor generation. EIP is the offset from the CSbase address, while LIP includes the CSbase address. Which IP type is used is indicated in CPUID.(EAX=01CH, ECX=0):EAX[bit 31].

The value read from this field will always be canonical. Note that this includes the case where a canonical violation (#GP) results from executing sequential code that runs precisely to the end of the lower canonical address space (where IP[63:MAXLINADDR-1] is 0, but IP[MAXLINADDR-2:0] is all ones). In this case, the FROM_IP will hold the lowest canonical address in the upper canonical space, such that IP[63:MAXLINADDR-1] is all ones, and IP[MAXLINADDR-2:0] is 0.

In some cases, due to CPL filtering, the FROM_IP of the recorded operation may be filtered out. In this case 0FFFFFFFFFFFFFFFH will be recorded. See Section 7.1.2.5 for details.

Writes of these fields will be forced canonical, such that the processor ignores the value written to the upper bits (IP[63:MAXLINADDR-1]).

7.1.3.2 Branch Types

The IA32_LBR_x_INFO.BR_TYPE and IA32_LER_INFO.BR_TYPE fields encode the branch types as shown in Table 7-3.

Table 7-3. IA32_LBR_x_INFO and IA32_LER_INFO Branch Type Encodings

Encoding	Branch Type
0000B	JCC
0001B	NEAR_IND_JMP
0010B	NEAR_REL_JMP
0011B	NEAR_IND_CALL
0100B	NEAR_REL_CALL
0101B	NEAR_RET
011xB	Reserved
1xxxB	OTHER_BRANCH

For a list of branch operations that fall into the categories above, see Table 7-2. In future generations, BR_TYPE bits 2:0 may be used to distinguish between differing types of OTHER_BRANCH.

7.1.3.3 Cycle Time

Each time an operation is recorded in an LBR, the value of the LBR cycle timer is recorded in IA32_LBR_x_INFO.CYC_CNT. The LBR cycle timer is a saturating counter that counts at the processor clock rate. Each time an operation is recorded in an LBR, the counter is reset but continues counting.

There is a LBR cycle time valid bit, IA32_LBR_x_INFO.CYC_CNT_VALID. When set, the CYC_CNT field holds a valid value, the number of elapsed cycles since the last operation recorded in an LBR (up to 0FFFFH).

Some implementations may opt to reduce the granularity of the CYC_CNT field for larger values. The implication of this is that the least significant bits may be forced to 1 in cases where the count has reached some minimum threshold. It is guaranteed that this reduced granularity will never result in an inaccuracy of more than 10%.

7.1.3.4 Mispredict Information

IA32_LBR_x_INFO.MISPRED provides an indication of whether the recorded branch was predicted incorrectly by the processor. The bit is set if either the taken/not-taken direction of a conditional branch (Jcc) was mispredicted, or if the target of an indirect branch was mispredicted.

7.1.3.5 Intel® TSX Information

IA32_LBR_x_INFO.IN_TSX indicates whether the operation recorded retired during a TSX transaction. IA32_LBR_x_INFO.TSX_ABORT indicates that the operation is a TSX Abort.

7.1.4 Interaction with Other Processor Features

7.1.4.1 SMM

IA32_LBR_CTL.LBREn is saved and cleared on #SMI, and restored on RSM. As a result of disabling LBRs, the #SMI is not recorded. RSM is recorded only if IA32_DEBUGCTL.FREEZE_WHILE_SMM is set to 0, and the FROM_IP will be set to the same value as the TO_IP.

SMM Transfer Monitor (STM)

LBREn is not cleared on #SMI when it causes SMM VM exit. Instead, the STM should use the VMCS controls described in Section 7.1.4.2 to disable LBRs while in SMM, and to restore them on VM entries that exit SMM.

On VMCALL to configure STM, IA32_LBR_CTL is cleared.

7.1.4.2 VMX

By default, LBR operation persists across VMX transitions. However, VMCS fields have been added to enable constraining LBR usage to within non-root operation only. See details in Table 7-4.

Table 7-4. LBR VMCS Fields

Name	Type	Bit Position	Behavior
Guest IA32_LBR_CTL	Guest State Field	NA	The guest value of IA32_LBR_CTL is written to this field on all VM exits.
Load Guest IA32_LBR_CTL	Entry Control	21	When set, VM entry will write the value from the "Guest IA32_LBR_CTL" guest state field to IA32_LBR_CTL.
Clear IA32_LBR_CTL	Exit Control	26	When set, VM exit will clear IA32_LBR_CTL after the value has been saved to the "Guest IA32_LBR_CTL" guest state field.

To enable "guest-only" LBR use, a VMM should set both the "Load Guest IA32_LBR_CTL" entry control and the "Clear IA32_LBR_CTL" exit control. For "system-wide" LBR use, where LBRs remain enabled across host and guest(s), a VMM should keep both new VMCS controls clear.

VM-entry checks that, if the "Load Guest IA32_LBR_CTL" entry control is 1, bits reserved in the IA32_LBR_CTL MSR must be 0 in the field for that register.

Table 7-5 enumerates the 64-bit guest-state fields.

Table 7-5. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest IA32_LBR_CTL (full)[1]	000001011B	00002816H
Guest IA32_LBR_CTL (high)[1]		00002817H

NOTES:

1. This field exists only on processors that support either the 1-setting of the "load IA32_LBR_CTL" VM-entry control or that of the "clear IA32_LBR_CTL" VM-exit control.

7.1.4.3 Intel® SGX

On entry to an enclave, via EENTER or ERESUME, logging of LBR entries is suspended. On enclave exit, via EEXIT or AEX, logging resumes. The cycle counter will continue to run during enclave execution.

An exception to the above is made for opt-in debug enclaves. For such enclaves, LBR logging is not impacted.

7.1.4.4 Debug Breakpoints

On a debug breakpoint event (#DB), IA32_LBR_CTL.LBREn is cleared. As a result, the operation is not recorded.

7.1.4.5 SMX

On GETSEC leaves SENTER or ENTERACCS, IA32_LBR_CTL is cleared. As a result, the operation is not recorded.

7.1.4.6 MWAIT

On an MWAIT that requests a C-state deeper than C1, IA32_LBR_x_* MSRs may be cleared to 0. IA32_LBR_CTL, IA32_LBR_DEPTH, and IA32_LER_* MSRs will be preserved.

For an MWAIT that enters a C-state equal to or less deep than C1, and all C-states that enter as a result of Hardware Duty Cycling (HDC), all LBR MSRs are preserved.

7.1.4.7 Precise Event-Based Sampling (PEBS)

PEBS records can be configured to include LBRs, by setting PEBS_DATA_CFG.LBREn[3]=1. The number of LBRs to include in the record is also configurable, via PEBS_DATA_CFG.NUM_LBRS[28:24]. For details on PEBS, see Section 18.9 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

If NUM_LBRS is set to a value greater than LBR_DEPTH, then only LBR_DEPTH entries will be written into the PEBS record. Further, the Record Size field will be decreased to match the actual size of the record to be written, and the Record Format field will replace the value of NUM_LBRS with the value of LBR_DEPTH. These adjustments ensure that software is able to properly interpret the PEBS record.

7.2 MSRS

The MSRs that represent the LBR entries (IA32_LBR_x_[TO|FROM|INFO]) and the LER entry (IA32_LER_[TO|FROM|INFO]) do not fault on writes. Any address field written will force sign-extension based on the maximum linear address width supported by the processor, and any non-zero value written to undefined bits may be ignored such that subsequent reads return 0.

On a warm reset, all LBR MSRs, including IA32_LBR_DEPTH, have their values preserved. However, IA32_LBR_CTL.LBREn is cleared to 0, disabling LBRs. If a warm reset is triggered while the processor is in C6, also known as warm init, all LBR MSRs will be reset to their initial values.

Table 7-6. Introduction of New MSRs

Register Address		Register Name / Bit Fields	Bit Description	Reset Value
Hex	Dec			
1DD H	477	IA32_LER_FROM_IP	Last Event Record Source IP Register (R/W)	
		63:0	FROM_IP The source IP of the recorded branch or event, in canonical form.	0
1DEH	478	IA32_LER_TO_IP	Last Event Record Destination IP Register (R/W)	
		63:0	TO_IP The destination IP of the recorded branch or event, in canonical form.	0
1E0H	480	IA32_LER_INFO	Last Event Record Info Register (R/W)	
		55:0	Undefined, may be zero or non-zero. Writes of non- zero values do not fault, but reads may return a different value.	0
		59:56	BR_TYPE The branch type recorded by this LBR. Encodings match those of IA32_LBR_x_INFO.	0
		60	Undefined, may be zero or non-zero. Writes of non- zero values do not fault, but reads may return a different value.	0
		61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel® TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	0
		62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel® TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	0
		63	MISPRED The recorded branch taken/not-taken resolution (for JCCs) or target (for any indirect branch, including RETs) was mispredicted.	0
1200H - 121FH	4608 - 4639	IA32_LBR_x_INFO	Last Branch Record Entry X Info Register (R/W) An attempt to read or write IA32_LBR_x_INFO such that x ≥ IA32_LBR_DEPTH.DEPTH will #GP.	
		15:0	CYC_CNT The elapsed CPU cycles (saturating) since the last LBR was recorded. See Section 7.1.3.3.	0
		55:16	Undefined, may be zero or non-zero. Writes of non- zero values do not fault, but reads may return a different value.	0

Table 7-6. Introduction of New MSRs (Contd.)

Register Address		Register Name / Bit Fields	Bit Description	Reset Value
Hex	Dec			
		59:56	BR_TYPE The branch type recorded by this LBR. Encodings: 0000B: JCC 0001B: JMP Indirect 0010B: JMP Direct 0011B: CALL Indirect 0100B: CALL Direct 0101B: RET 011xB: Reserved 1xxxB: Other Branch	0
		60	CYC_CNT_VALID CYC_CNT value is valid. See Section 7.1.3.3.	0
		61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	0
		62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 11]=0), this bit is undefined.	0
		63	MISPRED The recorded branch direction (Jcc) or target (indirect branch) was mispredicted.	0
14CEH	5326	IA32_LBR_CTL	Last Branch Record Enabling and Configuration Register (R/W)	0
		0	LBREn When set, enables LBR recording.	0
		1	OS When set, allows LBR recording when CPL == 0.	0
		2	USR When set, allows LBR recording when CPL != 0.	0
		3	CALL_STACK When set, records branches in call-stack mode. See Section 7.1.2.4.	0
		15:4	Reserved.	0
		16	JCC When set, records taken conditional branches. See Section 7.1.2.3.	
		17	NEAR_REL_JMP When set, records near relative JMPs. See Section 7.1.2.3.	
		18	NEAR_IND_JMP When set, records near indirect JMPs. See Section 7.1.2.3.	

Table 7-6. Introduction of New MSRs (Contd.)

Register Address		Register Name / Bit Fields	Bit Description	Reset Value
Hex	Dec			
		19	NEAR_REL_CALL When set, records near relative CALLs. See Section 7.1.2.3.	
		20	NEAR_IND_CALL When set, records near indirect CALLs. See Section 7.1.2.3.	
		21	NEAR_RET When set, records near RETs. See Section 7.1.2.3.	
		22	OTHER_BRANCH When set, records other branches. See Section 7.1.2.3.	
		63:23	Reserved.	
14CFH	5327	IA32_LBR_DEPTH	Last Branch Record Maximum Stack Depth Register (R/W)	
		N:0	DEPTH The number of LBRs to be used for recording. Supported values are indicated by the bitmap in CPUID.(EAX=01CH,ECX=0):EAX[7:0]. The reset value will match the maximum supported by the CPU. Writes of unsupported values will #GP fault.	Varies
		63:N+1	Reserved.	0
1500H - 151FH	5376 - 5407	IA32_LBR_x_FROM_IP	Last Branch Record entry X source IP register (R/W). An attempt to read or write IA32_LBR_x_FROM_IP such that x >= IA32_LBR_DEPTH.DEPTH will #GP.	
		63:0	FROM_IP The source IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	0
1600H - 161F H	5632 - 5663	IA32_LBR_x_TO_IP	Last Branch Record Entry X Destination IP Register (R/W) An attempt to read or write IA32_LBR_x_TO_IP such that x >= IA32_LBR_DEPTH.DEPTH will #GP.	
		63:0	TO_IP The destination IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	0

7.3 CONTEXT SWITCH

On processors whose XSAVE feature set supports XSAVES and XRSTORS, the availability of support for Architectural LBR configuration state save and restore can be determined from CPUID.(EAX=0DH, ECX=1):EDX:ECX[bit 15]. When available, the Architectural LBR state can be saved using XSAVES and restored by XRSTORS, in conjunction with the bit field associated with the supervisory state component in IA32_XSS. See Chapter 13, "Managing State Using the XSAVE Feature Set" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

The IA32_XSS MSR is zero coming out of RESET. Once IA32_XSS[bit 15] is set, system software operating at CPL=0 can use XSAVES/XRSTORS with the appropriate requested- feature bitmap (RFBM) to manage supervisor state components in the XSAVE map. The layout of the Architectural LBR component state in the XSAVE area is shown in Table 7-7.

Table 7-7. LBR VMCS Fields

Offset Within Component Area	Field
0H	IA32_LBR_CTL
8H	IA32_LBR_DEPTH
10H	IA32_LER_FROM_IP
18H	IA32_LER_TO_IP
20H	IA32_LER_INFO
28H	IA32_LBR_0_FROM_IP
30H	IA32_LBR_0_TO_IP
38H	IA32_LBR_0_INFO
40H	IA32_LBR_1_FROM_IP
…	…
320H	IA32_LBR_31_INFO

Regardless of the number of LBRs supported by the processor, the size of the LBR state save region is constant.

7.3.1 XSAVE and LBR Depth

The behavior of XSAVES and XRSTORS when IA32_XSS[bit 15] is set depends on the value of IA32_LBR_DEPTH.DEPTH. When restoring IA32_LBR_x_* MSRs, only those MSRs associated with entries 0 through DEPTH-1 are saved or restored. Others are not accessed. For this reason, use of XSAVES and XRSTORS with a reduced DEPTH value can result in reduced context switch latency.

IA32_LBR_DEPTH is saved by XSAVES, but it is not written by XRSTORS in any circumstance. Instead, XRSTORS reads the saved IA32_LBR_DEPTH value and does the following.

- #GP if the saved value has any reserved bit or reserved value violations.
- Compare the saved value with the IA32_LBR_DEPTH MSR.
  - On a mismatch, the IA32_LBR_x_* MSRs are cleared to 0, while the IA32_LBR_CTL and LER MSRs are restored.

If there is no reserved bit or reserved value fault, and no depth mismatch, LBR state restore proceeds as normal. If any reserved bits in IA32_LBR_CTL are set a #GP will result.

Note that the end of the Architectural LBR State Component memory region may be accessed by XSAVES and XRSTORS, even when the DEPTH is reduced. Thus the pages for this memory region should be present and writable.

7.3.2 INIT and MOD Tracking

XSTATE_BV[bit 15] indicates the INIT state for Architectural LBRs. INIT tracking for Architectural LBRs includes all LBR MSRs with the exception of IA32_LBR_DEPTH. Any software write to a tracked LBR MSR renders MSRs non-INIT, with the exception of writes of value 0 to IA32_LBR_CTL.

An XRSTORS with XSTATE_BV[bit 15] set to 0 will clear all LBR MSRs, with the exception of IA32_LBR_DEPTH, to 0. Such an XRSTORS will not compare the current value of IA32_LBR_DEPTH with the value to be restored, since LBRs will not be in use upon completion of the restore. IA32_LBR_DEPTH is not modified by XRSTORS in any scenario.

An XSAVES with XINUSE[bit 15] set to 0 will clear XSTATE_BV[bit 15], but otherwise does not write to the Architectural LBR component state save area.

Note that this XSAVES behavior implies that the saved value of IA32_LBR_DEPTH could become stale while the rest of the LBRs are INIT, since modifications to IA32_LBR_DEPTH do not effect INIT tracking. This will have no impact on LBR behavior, as a subsequent XRSTORS that detects a depth mismatch will either ignore the IA32_LBR_DEPTH value (if XSTATE_BV[bit15]=0) or will re-initialize the IA32_LBR_x_* MSRs (if XSTATE_BV[bit 15]=1).

On XRSTORS with IA32_LBR_DEPTH mismatch, INIT tracking is not modified.

There is no MOD tracking for Architectural LBRs; they should be treated as modified anytime they are not in INIT state.

It is recommended that software initialize the Architectural LBR State Component memory to all zeros, and to clear XSTATE_BV[bit 15].

7.3.3 Fast LBR Read Access

XSAVES provides a faster means than RDMSR for software to read all LBRs. When using XSAVES for reading LBRs rather than for context switch, software should take care to ensure that XSAVES does not write LBR state to an area of memory that has been or will be used by XRSTORS. This could corrupt INIT tracking.

7.3.4 XRSTORS Faulting

If an XRSTORS with IA32_XSS[bit 15] set to 1 faults for any reason, LBR MSRs may retain the pre-XRSTORS values, or may hold the newly restored values, or may be left in a state of partial restoration.

7.4 ENUMERATION

7.4.1 CPUID for Architectural LBRs

If CPUID.(EAX=07H, ECX=0):EDX[19] is set to 1, the processor supports Architectural LBRs. In this case, CPUID leaf 01CH indicates details of the Architectural LBRs capabilities. The leaf 01CH format is shown in Table 7-8.

Table 7-8. CPUID Leaf 01CH Enumeration of Architectural LBR Capabilities

CPUID.(EAX=01CH, ECX=0)		Name	Description
Register	Bits		
EAX	7:0	Supported LBR Depth Values	For each bit *n* set in this field, the IA32_LBR_DEPTH.DEPTH value 8*(*n*+1) is supported.
	29:8	Reserved	Reserved.
	30	Deep C-state Reset	If set, indicates that LBRs may be cleared on an MWAIT that requests a C-state numerically greater than C1.
	31	IP Values Contain LIP	If set, LBR IP values contain LIP. If clear, IP values contain EIP.
EBX	0	CPL Filtering Supported	If set, the processor supports setting IA32_LBR_CTL[2:1] to non-zero value.
	1	Branch Filtering Supported	If set, the processor supports setting IA32_L-BR_CTL[22:16] to non-zero value.
	2	Call-stack Mode Supported	If set, the processor supports setting IA32_LBR_CTL[3] to 1.
	31:3	Reserved	Reserved.

Table 7-8. CPUID Leaf 01CH Enumeration of Architectural LBR Capabilities (Contd.)

CPUID.(EAX=01CH, ECX=0)		Name	Description
Register	Bits		
ECX	0	Mispredict Bit Supported	IA32_LBR_x_INFO[63] holds indication of branch misprediction (MISPRED)
	1	Timed LBRs Supported	IA32_LBR_x_INFO[15:0] holds CPU cycles since last LBR entry (CYC_CNT), and IA32_LBR_x_INFO[60] holds an indication of whether the value held there is valid (CYC_CNT_VALID).
	2	Branch Type Field Supported	IA32_LBR_INFO_x[59:56] holds indication of the recorded operation's branch type (BR_TYPE).
	31:3	Reserved	Reserved.
EDX	31:0	Reserved	Reserved.

For leaf 01CH, CPUID will ignore the ECX value.

7.4.2 CPUID for XSAVE LBR Support

XSAVE support for Architectural LBRs is enumerated in CPUID.(EAX=0DH, ECX=0FH); see details in Table 7-9.

Table 7-9. CPUID Leaf 0DH.0FH Enumeration of XSAVE Support for Architectural LBRs

CPUID.(EAX=0DH, ECX=0FH)		Name	Description
Register	Bits		
EAX	31:0	Size	Size, in bytes, of the Arch LBR save area.
EBX	31:0	Offset	Offset, in bytes, of the start of the Arch LBR save area from the beginning of the XSAVE/XRSTOR area.
ECX	0	Supervisor State	Set if bit 15 is supported in the IA32_XSS MSR; it is clear if bit 15 is instead supported in XCR0.
	1	Aligned	Set if, when the compacted format of an XSAVE area is used, this extended state component located on the next 64-byte boundary following the preceding state component. Otherwise, it is located immediately following the preceding state component.
	31:2	Reserved	Reserved.
EDX	31:0	Reserved	Reserved.

7.4.3 Enumeration for New VMCS Fields

New VMX controls and fields also include new enumeration bits.

- IA32_VMX_TRUE_EXIT_CTLS MSR (address 048FH) and IA32_VMX_EXIT_CTLS MSR (address 0483H) will have bit 26 set to 0, indicating allowed 0-setting of "Clear IA32_LBR_CTL" exit control, and bit 58 set to 1, indicating allowed 1-setting of "Clear IA32_LBR_CTL" exit control.
- IA32_VMX_TRUE_ENTRY_CTLS MSR (address 0490H) and IA32_VMX_ENTRY_CTLS MSR (address 0484H) will have bit 21 set to 0, indicating allowed 0-setting of "Load Guest IA32_LBR_CTL" exit control, and bit 53 set to 1, indicating allowed 1-setting of "Load Guest IA32_LBR_CTL" exit control.

  Bit 53 also indicates support for the "Guest IA32_LBR_CTL" guest state field.

7.5 OTHER IMPACTS

7.5.1 Branch Trace Store on Intel Atom Processors

Branch Trace Store (BTS) on Intel Atom processors that support Architectural LBRs has dependencies on the LBR configuration. BTS will store out the LBR_0 (TOS) record each time a taken branch or event retires. If any filtering of LBRs is employed, or if LBRs are disabled, some duplicate entries may be stored by BTS. Like LBRs and LERs, BTS is suspended when IA32_PERF_GLOBAL_STATUS.LBR_FRZ is set to 1.

BTS will change to cease issuing branch records for zero-length CALLs (direct near CALLs to the next sequential IP) to align with Architectural LBR behavior.

7.5.2 IA32_DEBUGCTL

On processors that do not support model-specific LBRs, IA32_DEBUGCTL[bit 0] has no meaning. It can be written to 0 or 1, but reads will always return 0.

7.5.3 IA32_PERF_CAPABILITIES

On processors that do not support model-specific LBRs, IA32_PERF_CAPABILITIES.LBR_FMT will have the value 03FH.

CHAPTER 8
NON-WRITE-BACK LOCK DISABLE ARCHITECTURE

Locked read-modify-write (RMW) to a memory operation is used explicitly by several Intel architecture set instructions, such as ADD with a lock prefix, and implicitly by other instructions and flows, such as updating a segment access bit or page tables access/dirty bits.

Locked RMW access is usually handled through processor cache in the lower hierarchies, and it only impacts software running on same logical processors that share this cache.

If the memory type of this locked RMW is not write-back, the processor can't handle it within the internal cache and will issue a bus lock operation. This operation will block all logical processors and devices from accessing memory until the operation has completed.

Having a burst of bus locks by one of the logical processors may cause starvation to the rest of the logical processors and devices.

The new architecture will allow software to disable non-WB lock operation. Once the feature is enabled, performing a non-WB lock operation by software will generate a general protection fault (#GP).

8.1 ENUMERATION

The non-write-back lock disable capability will be enumerated through a model specific bit (bit 4) in the IA32_CORE_CAPABILITIES MSR.

Table 8-1. IA32_CORE_CAPABILITIES MSR

Register Address		Architectural MSR Name / Bit Fields	Description
Hex	Decimal		
CFH	207	IA32_CORE_CAPABILITIES	IA32 Core Capability Register
		3:0	Reserved
		4	Non-WB Lock disable #GP(0) exception for non-WB locked accesses supported.
		5	Split Lock disable #AC(0) exception for split locked accesses supported.
		63:6	Reserved

8.2 ENABLING

This model specific feature will add a new MSR control bit (bit 28) in the TEST_CTRL MSR in order to generate a general protection fault (#GP) each time a non-WB load lock is detected.

Table 8-2. TEST_CTRL MSR

Register Address		Architectural MSR Name / Bit Fields	Description
Hex	Decimal		
33H	55	TEST_CTRL	Test Control Register
		27:0	Reserved
		28	Enable #GP(0) exception for non-write-back locked accesses.
		29	Enable #AC(0) exception for split locked accesses.
		31:30	Reserved

8.3 INTERACTION WITH INTEL® SOFTWARE GUARD EXTENSIONS (INTEL® SGX)

Processor Reserved Memory (PRM) used for Intel® SGX used can run with non-WB memory accesses by following the steps below.

1. Configure the Memory Type field (bits 2:0) of MSR_PRMRR_BASE_0 (address 2A0H) to be non-WB.
2. Set the cache disable (bit 30) of CR0.

When the processor is configured in this manner, the processor will not generate #GP(0) as a result of locked accesses to non-WB memory when EPT is enabled, even if the non-WB lock disable (bit 28) of TEST_CTRL MSR (address 33H) is set to 1.

8.4 INTERACTION WITH VMX ARCHITECTURE

There are two cases where a locked cycle can be issued on a VMM configuration with non-WB memory type.

1. VMM enabled EPT and EPT A/D and configured EPT memory type to non-WB. In this case, EPT A/D assist will issue a locked load to non-WB memory.
2. VMM set "process posted interrupts" VM-execution control, posted-interrupt descriptor mapped to non-WB memory type. Posted interrupt processing will update the descriptor with locked load to non-WB memory.

When the processor is configured in this manner, the processor will not generate #GP(0) as a result of a locked access to non-WB memory when EPT is enabled even if the non-WB lock disable (bit 28) of TEST_CTRL MSR (address 33H) is set to 1.

8.5 EXPECTED SOFTWARE BEHAVIOR

Software can ensure that bus locks as a result of non-WB locked access are never taken, or at least a general protection fault is signaled, by performing the following operations:

- Set Non-WB Lock Disable (bit 28) of the TEST_CTRL MSR (address 33H).
- Do not set Cache Disable (bit 30) of CR0.
- Configure MSR_PRMRR_BASE_0 (address 2A0H) Memory Type field (bits 2:0) to WB memory type only.
- For a VMM that enabled EPT and EPT A/D, bits must configure EPT paging structures to WB memory type.
- For a VMM that enabled posted-interrupt via the "process posted interrupts" VM-execution control, ensure the posted-interrupt descriptor is mapped to WB memory type.

8.6 BUS LOCKS

Cases for bus locks than can come from non-WB Lock operation are shown in Table 8-3.

Table 8-3. Bus Locks from Non-WB Operation

Category	Instructions/Flows	Conditions
Arithmetic	LOCK + {ADD, SUB, AND, OR, XOR, ADC, SBB, INC, DEC, NOT, NEG}	
Compare/Test	LOCK + {BTC, BTR, BTS}	
Exchange	XCHG, LOCK XADD/CMPXCHG/XCHG	
Segmentation	LSL, LAR, VERR, VERW LDS, LES, LFS, LGS, LSS MOV DS, MOV ES, MOV FS, MOV GS, MOV SS POP DS, POP ES, POP FS, POP GS, POP SS	Setting segment accessed bit in descriptor in non-WB memory.
Call / Interrupt / Exception	Far call, Far JMP Far RET, IRET INTn, INT3, INTO, INT1 Call through interrupt/trap gate	Setting segment accessed bit in descriptor in non-WB memory.
Tasking	LTR, Task Switch	Setting/Clearing TSS busy when TSS in non-WB memory. Setting segment accessed bit in descriptor in non-WB memory.
Paging	Code fetch (A bit update), All instructions that have memory operands (A/D bits update)	Page tables in non-WB memory.
Enclave	ENCLU, ENCLS, AEX	
Posted Interrupts	Updating the posted interrupt descriptor uses locked RMW for atomic operations	Posted interrupt descriptor in non-WB memory.

CHAPTER 9
BUS LOCK AND VM NOTIFY

9.1 BUS LOCK DEBUG EXCEPTION

A logical processor can be configured to generate a debug exception (#DB) as a trap delivered in the instruction boundary following acquisition of a bus lock if the processor is at privilege level > 0 on this instruction boundary. Software enables these debug exceptions by setting bit 2 of the IA32_DEBUGCTL MSR. The CPU enumerates support for the 1-setting of this bit using CPUID.(EAX=7, ECX=0).ECX[24].

A debug exception due to acquisition of a bus lock is reported as a trap following execution of the instruction acquiring the bus lock if the privilege level is > 0. The processor identifies such debug exceptions using bit 11 of DR6. Because DR6[11] has formerly always been 1, delivery of a bus-lock #DB **clears** DR6[11]. All other debug exceptions leave bit 11 unmodified. To avoid confusion in identifying debug exceptions, software debug-exception handlers should set bit 11 to 1 before returning to the interrupted task.

A VM exit sets bit 11 of the pending debug exception field in the guest-state area of the VMCS to indicate that a bus lock debug exception was pending but not delivered. A VM exit that sets this bit also sets bit 12 of that field. (VM exits also sets bit 12 to indicate that at least one data or I/O breakpoint was met and was enabled in DR7, or that a debug exception related to advanced debugging of RTM transactional regions occurred.)

9.1.1 Bus Lock VM Exit

A VMM can enable VM exit on a bus lock that was acquired in VMX non-root operation by setting secondary processor-based execution control bit 30. A processor enumerates support for the 1-setting of this control by setting bit 62 of the IA32_VMX_PROCBASED_CTLS2 MSR. When enabled, the processor generates a VM exit with exit reason 74 if the processor detected one or more bus locks were caused during execution in VMX non-root operation. Such a VM exit is trap-like and is delivered following execution of the instruction acquiring the bus lock. If delivery of this VM exit was pre-empted by a higher priority VM exit, then bit 26 of the exit-reason field in the VMCS is set to 1.

9.2 NOTIFY VM EXIT

A VMM can enable **notification VM exits to occur** if no interrupt windows occur in VMX non-root operation for a specified amount of time (**notify window**). These VM exits are enabled by setting bit 31 of the secondary processor-based execution control. A processor enumerates support for the 1-setting of this control by setting bit 63 of the IA32_VMX_PROCBASED_CTLS2 MSR. The VMM configures the notify window in units of crystal clock cycles in a new 32-bit VM-execution control field in the VMCS (notify window) that can be accessed with the VMREAD and VMWRITE instructions using encoding 00004024H.

A notification VM exit reports basic exit reason 75 and exit qualification determined as follows:

- Bit 0 - VM context invalid.
- Bits 11:1 are reserved.
- Bit 12 - if set the VM exit was incident to an execution of IRET that unblocked NMIs.
- All other bits are reserved.

If the VMM-notify VM exit occurred incident to delivery of a vectored event, then IDT vectoring information and applicable error code are recorded in the VMCS.

CHAPTER 10
INTEL® RESOURCE DIRECTOR TECHNOLOGY FEATURE UPDATES

Intel® Resource Director Technology (Intel® RDT) provides a number of monitoring and control capabilities for shared resources in multiprocessor systems. This chapter covers updates to the feature that will be available in future Intel processors.

10.1 INTEL® RDT FEATURE CHANGES

10.1.1 Intel® RDT on Processors Based on Ice Lake Server Microarchitecture

Processors based on Ice Lake Server microarchitecture add the following Intel RDT enhancements:

- 32-bit MBM counters (vs. 24-bit in prior generations), and new CPUID enumeration capabilities for counter width.
- Second Generation Memory Bandwidth Allocation (MBA 2.0): Introduces an advanced hardware feedback controller which operates at microsecond timescales, and software-selectable min/max delay value resolution capabilities. Note that delay values may be thought of as "throttling values" applied to the threads running on a core, as described in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*. MBA 2.0 also adds a work-conserving feature in which applications that frequently access the L3 cache may be throttled by a lesser amount until they exceed the user-specified memory bandwidth usage threshold, enhancing system throughput and efficiency, in addition to adding more precise calibration and controls.
- 15 MBA / L3 CAT CLOS: Improved feature consistency and interface flexibility. The previous generation of processors supported 16 L3 CAT CLOS, but only 8 MBA 1.0 CLOS. The changes in enumerated CLOS counts per-feature are already enumerated in the architecture via CPUID.

10.1.2 Intel® RDT on Intel Atom® Processors Based on Tremont Microarchitecture

Intel Atom® processors based on Tremont microarchitecture add the following Intel RDT enhancements:

- L2 CAT/CDP: L2 CAT/CDP and L3 CAT/CDP enabled simultaneously. As these are legacy features already defined, no new software enabling should be required.
- Matches Ice Lake Server microarchitecture support for traditional Intel RDT uncore features: L3 CAT/CDP, CMT, MBM, MBA 2.0. As these features are architectural, no new software enabling is required aside from MBA 2.0.
- New features added in Ice Lake Server microarchitecture also carry forward to Tremont microarchitecture, with the same software enabling required. These features include 32-bit MBM counters, MBA 2.0, and 15 MBA/L3 CAT CLOS.

10.1.3 Intel® RDT in Processors Based on Sapphire Rapids Server Microarchitecture

Processors based on Sapphire Rapids Server microarchitecture add the following Intel RDT enhancements:

- L2 CAT and CDP: Includes control over the L2 cache and the ability to partition the L2 cache into separate code and data virtual caches. No new software enabling is required; this is the same behavior as described in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.
- Third Generation Memory Bandwidth Allocation (MBA 3.0): New per-thread capability for bandwidth control, enabling precise bandwidth shaping and noisy neighbor control. Some portions of the control infrastructure now operate at core frequencies for controls which are responsive at the nanosecond level.
- STLB QoS: Capability to manage the second-level translation lookaside buffer structure within the core (STLB) in a manner quite similar to CAT (CLOS-based, with capacity masks). This may enable software which is sensitive to TLB performance to achieve better determinism. This is a model-specific feature due to the microarchitectural nature of the STLB structure. The code regions of interest must be manually accessed.

10.2 ENUMERABLE MEMORY BANDWIDTH MONITORING COUNTER WIDTH

Memory Bandwidth Monitoring (MBM) is an Intel RDT feature which tracks total and local bandwidth generated which misses the L3 cache.

The original Memory Bandwidth Monitoring (MBM) architectural definition defines counters of up to 62 bits in the IA32_QM_CTR MSR, and the first-generation MBM implementation used 24-bit counters. Software is required to poll at ≥ 1Hz to ensure that data is retrieved before a counter rollover occurs more than once. This ≥ 1Hz sampling ensures that under worst-case conditions rollover between samples occurs at most once, but under more typical conditions rollover typically requires multiple seconds to occur.

As bandwidths scale, extensions to more elegantly handle high-bandwidth future systems are desirable. One of these extensions, detailed in this chapter, includes an enumerable MBM counter width. Ice Lake Server microarchitecture utilizes this definition to implement 32-bit MBM counters, but future growth should be anticipated.

10.2.1 Memory Bandwidth Monitoring (MBM) Enabling

Memory Bandwidth Monitoring, like other Intel RDT features, uses CPUID for enumeration, and MSRs for assigning RMIDs and retrieving counter data. For CPUID enumeration details, see the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*. For additional MBM details, see Chapter 17 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

10.2.2 Augmented MBM Enumeration and MSR Interfaces for Extensible Counter Width

A field is added to CPUID to enumerate the MBM counter width in platforms which support the extensible MBM counter width feature.

Prior to this point, CPUID.0F.[ECX=1]:EAX was reserved. This CPUID output register (EAX) is redefined to provide two new fields:

- Encode counter width as offset from 24b in bits[7:0].
- An overflow bit in bit[8].

See "CPUID—CPU Identification" in Chapter 1 for details.

In EAX bits 7:0, the counter width is encoded as an offset from 24b. A value of zero in this field means 24-bit counters are supported. A value of 8 indicates that 32-bit counters are supported, as in processors based on Ice Lake Server microarchitecture.

With the addition of this enumerable counter width, the requirement that software poll at ≥ 1Hz is removed. Software may poll at a varying rate with reduced risk of rollover, and under typical conditions rollover is likely to require hundreds of seconds (though this value is not explicitly specified, and may vary and decrease over time). If software seeks to ensure that rollover does not occur more than once between samples, then sampling at ≥ 1Hz while consuming the enumerated counter widths' worth of data will provide this guarantee, for a specific platform and counter width, under all conditions.

Software which uses the MBM event retrieval MSR interface should be updated to comprehend this new format, which enables up to 62-bit MBM counters to be provided by future platforms. Higher-level software which consumes the resulting bandwidth values is not expected to be affected.

10.3 SECOND GENERATION MEMORY BANDWIDTH ALLOCATION

The second generation of Memory Bandwidth Allocation (MBA 2.0) is implemented in processors based on Ice Lake Server microarchitecture, and improves the behavior and accuracy of MBA, along with providing increased throughput while using the feature and greater efficiency. Rather than a strict bandwidth control mechanism, a more dynamic hardware controller is used internally which can react to changing bandwidth conditions at the microsecond level.

Prior to using the MBA 2.0 feature, the MBA 2.0 hardware controller requires a BIOS-assisted calibration process which may include inputs such as the number of memory channels populated and other system parameters; this is

a change from MBA 1.0 which did not require this. Intel BIOS reference code includes a default configuration which is recommended for general usage.

MBA 2.0 in Ice Lake Server and Tremont microarchitectures moves from static throttling at the core/uncore interface to a more dynamic control scheme based on a hardware controller that tracks actual DRAM bandwidth. This allows software which uses primarily the L3 cache to observe increased throughput for a given throttling level, or benefits for software which exhibits L3-bound phases. Due to the closer consideration of memory bandwidth loading, this enhancement may lead to an increase in system efficiency when using MBA 2.0, relative to MBA 1.0.

MBA 1.0 is established as a legacy feature. Backward compatibility of the software interfaces is preserved, and MBA 2.0 changes manifest as enhancements atop the MBA 1.0 baseline.

As with the prior generation feature, MBA 2.0 uses CPUID for enumeration, and throttling is performed using a mapping created from software thread-to-CLOS (in the IA32_PQR_ASSOC MSR) which is then mapped per-CLOS to delay values via the IA32_L2_QoS_Ext_BW_Thrtl_n MSRs. User software specifies a per-CLOS delay value, 0-90% bandwidth throttling for instance, though the max and granularity are platform dependent and enumerated in CPUID.

10.3.1 MBA 2.0 Advantages

MBA 2.0 adds some additional features over MBA 1.0 as described below.

1. Previously, only the maximum delay value across two CLOS on a physical core could be selected in MBA. MBA 2.0 allows a minimum delay value to also be selected.
2. Only a single calibration table was possible in MBA 1.0, meaning different memory configurations had different linearity / percent delay value error values depending on the configuration. This is addressed by the BIOS support in MBA 2.0, and certain BIOS implementations may program a different calibration table per memory configuration, for instance.
3. The MBA 2.0 controller provides the ability to more closely monitor the memory bandwidth loading and deliver more optimal results.
4. MBA 2.0 includes a hardware controller, reducing the need for a fine-grained software controller to manage application phases. (A software controller is still valuable to translate MBA throttling values to bandwidths in GB/s or application SLAs.)

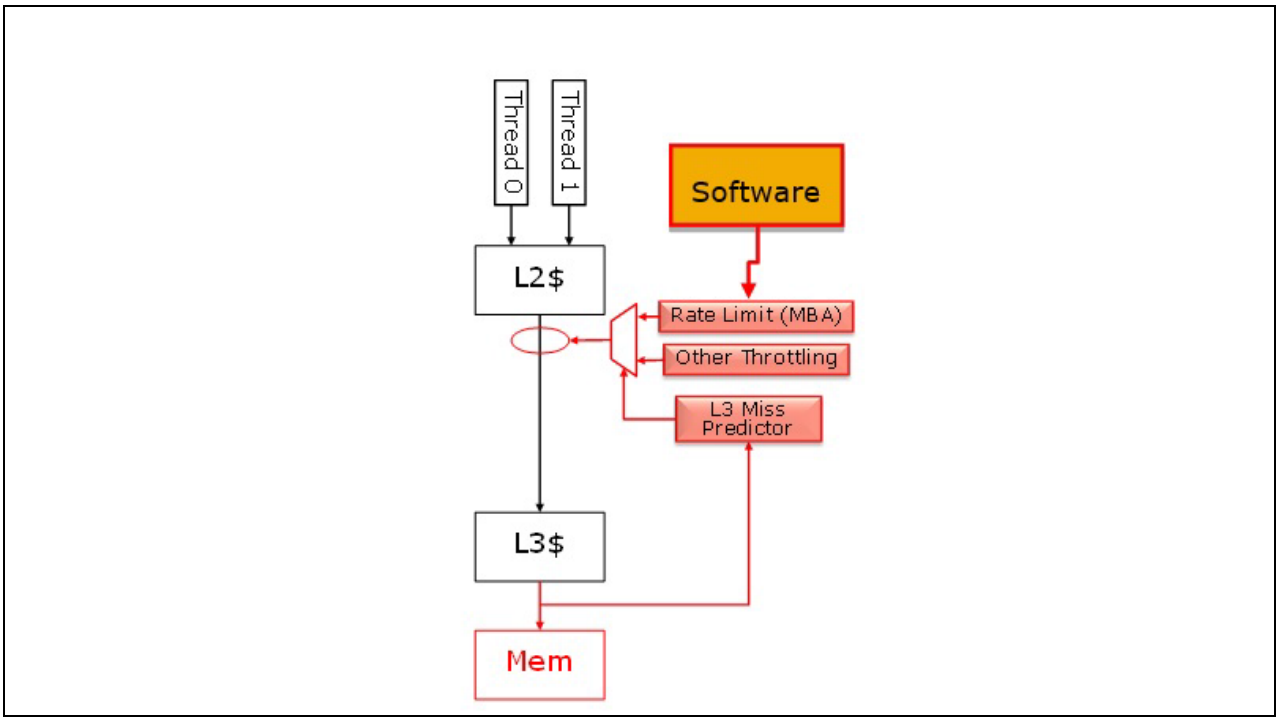


Figure 10-1. MBA 2.0 Concept Based Around a Hardware Controller

MBA 2.0 implementation is shown in Figure 10-1. MBA 2.0 operates through the use of an advanced new hardware controller and feedback mechanism which allows automated hardware monitoring and control around the user-provided delay value set point. This set point and associated delay value infrastructure remains unchanged from MBA 1.0, preserving software compatibility.

MBA 2.0 enhancements over MBA 1.0, in addition to the new hardware controller, include:

1. Configurable delay selection across threads.
    - MBA 1.0 implementation statically picks the max MBADelay across the threads running on a core (by calculating value = max(delayValue(CLOS[thread0]),delayValue(CLOS[thread1]))).
    - Software may have the option to pick either maximum or minimum delay to be resolved and applied across the threads; maximum value remains the default.
2. Increasing CLOSIDs from 8 to 15.
    - Skylake Server microarchitecture has 8 CLOSIDs for MBA 1.0.
    - Ice Lake Server microarchitecture increases this value to 15 (also consistent with L3 CAT).

10.3.2 MBA 2.0 Software-Visible Changes

A new MSR is introduced with MBA 2.0 to allow software to select from the maximum (default) or minimum of resolved delay values (see formula above). This capability is controlled via a bit in the new MBA_CFG MSR, shown below.

Table 10-1. MBA_CFG MSR Definition

Register Address		Architectural MSR Name / Bit Fields	Description
Hex	Decimal		
C84H	3204	MBA_CFG	MBA Configuration Register
		0	Min (1) or max (0) of per-thread MBA delays.
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).

Note that bit[0] for min/max configuration is supported in MBA 2.0, but is removed when MBA 3.0 moves the controller logic to per-thread capable. This transient feature existence is why the min/max control remains model-specific.

To enumerate and manage support for the model-specific min/max feature, software may use processor family/model/stepping to match supported products, then CPUID to later detect MBA 3.0 support.

10.4 THIRD GENERATION MEMORY BANDWIDTH ALLOCATION

The third generation MBA (MBA 3.0) feature on Sapphire Rapids microarchitecture further enhances the feature with per-thread control and a further improved controller design.

MBA 3.0 follows the past MBA precedent of delivering significant enhancements without a major software overhaul, and while preserving backward compatibility.

10.4.1 MBA 3.0 Hardware Changes

MBA 3.0 builds upon the hardware controller introduced with MBA 2.0, which enabled significant system-level benefits, and removes the per-core throttling limitation. Throttling values are no longer selected as the “min” or “max” of the two delay values for the threads running on the core, instead throttling values are directly applied to the threads running on the core.

While this means that more direct throttling of threads is possible, future usage guidance may be necessary to help explain the effects of Intel® Hyper-Threading Technology contention vs. cache and memory contention, and how these effects may be understood by software.

10.4.2 MBA 3.0 Software-Visible Changes

In order to allow software to change its tuning behavior and detect that per-thread throttling is supported on a particular product generation, a new CPUID bit is added to the MBA CPUID leaf to indicate this. See "CPUID—CPU Identification" in Chapter 1 for details.

Despite another significant improvement of the hardware controller infrastructure architecture and improved capabilities, controller responsiveness, new internal microarchitecture, and transient-arresting capabilities, no new software interface changes are required to make use of MBA 3.0 relative to prior generations. Software previously using the MBA 2.0 min/max selection capability should discontinue use of the MBA_CFG MSR.

CHAPTER 11
USER INTERRUPTS

11.1 INTRODUCTION

This chapter details an architectural feature called user interrupts.

This feature defines user interrupts as new events in the architecture. They are delivered to software operating in 64-bit mode with CPL = 3 without any change to segmentation state. Different user interrupts are distinguished by a 6-bit user-interrupt vector, which is pushed on the stack as part of user-interrupt delivery. A new instruction, UIRET (user-interrupt return) reverses user-interrupt delivery.

The user-interrupt architecture is configured by new supervisor-managed state. This state includes new MSRs. In expected usages, an operating system (OS) will update the content of these MSRs when switch between OS-managed threads.

One of the MSRs references a data structure called the **user posted-interrupt descriptor (UPID)**. User interrupts for an OS-managed thread can be posted in the UPID associated with that thread. Such user interrupts will be delivered after receipt of an ordinary interrupt (also identified in the UPID) called a **user-interrupt notification**.[1]

System software can define operations to post user interrupts and to send user-interrupt notifications. In addition, the user-interrupt architecture defines a new instruction, SENDUIPI, by which application software can send inter-processor user interrupts (user IPIs). An execution of SENDUIPI posts a user interrupt in a UPID and sends a user-interrupt notification.

(Platforms may include mechanisms to process external interrupts as either ordinary interrupts or user interrupts. Those processed as user interrupts would be posted in UPIDs may result in user-interrupt notifications. Specifics of such mechanisms are outside of the scope of this document.)

Section 11.2 explains how a processor enumerates support for user interrupts and how they are enabled by system software. Section 11.3 identifies the new processor state defined for user interrupts. Section 11.4 explains how a processor identifies and delivers user interrupts. Section 11.5 describes how a processor identifies and processes user-interrupt notifications. Section 11.7 defines new support for user inter-processor interrupts (user IPIs). Section 11.8 details how existing instructions support the new processor state and presents instructions to be introduced for user interrupts. Section 11.8.2 and Section 11.9 describe how user interrupts are supported by the XSAVE feature set and the VMX extensions, respectively.

11.2 ENUMERATION AND ENABLING

Software enables user interrupts by setting bit 25 (UINTR) in control register CR4. Setting CR4.UINTR enables user-interrupt delivery (Section 11.4.2), user-interrupt notification identification (Section 11.5.1), and the user-interrupt instructions (Section 11.6). It does not affect the accessibility of the user-interrupt MSRs (Section 11.3) by RDMSR, WRMSR or the XSAVE feature set.

Processor support for user interrupts is enumerated by CPUID.(EAX=7,ECX=0):EDX[5]. If this bit is set, software can set CR4.UINTR to 1 and can access the user-interrupt MSRs using RDMSR and WRMSR (see Section 11.3 and Section 11.8.1).

The user-interrupt feature is XSAVE-managed (see Section 11.8.2). This implies that aspects of the feature are enumerated as part of enumeration of the XSAVE feature set. See Section 11.8.2.2 for details.

1. For clarity, this chapter uses the term **ordinary interrupts** to refer to those events in the existing interrupt architecture, which are typically delivered to system software operating with CPL = 0.

11.3 USER-INTERRUPT STATE AND USER-INTERRUPT MSRS

The user-interrupt architecture defines the following new state. Some of this state can be accessed via the RDMSR and WRMSR instructions (through new user-interrupt MSRs detailed in Section 11.3.2) and some can be accessed using new instructions described in Section 11.6.

11.3.1 User-Interrupt State

The following are the elements of the new state (enumerated here independent of how they are accessed):

- **UIRR: user-interrupt request register.**

  This value includes one bit for each of the 64 user-interrupt vectors. If UIRR[i] = 1, a user interrupt with vector i is requesting service. The notation **UIRRV** is used to refer to the position of the most significant bit set in UIRR; if UIRR = 0, UIRRV = 0.

- **UIF: user-interrupt flag.**

  If UIF = 0, user-interrupt delivery is blocked; if UIF = 1, user interrupts may be delivered. User-interrupt delivery clears UIF, and the new UIRET instruction sets it. Section 11.6 defines other new instructions for accessing UIF.

- **UIHANDLER: user-interrupt handler.**

  This is the linear address of the user-interrupt handler. User-interrupt delivery loads this address into RIP.

- **UISTACKADJUST: user-interrupt stack adjustment.**

  This value controls adjustment to the stack pointer (RSP) prior to user-interrupt delivery. It can account for an OS ABI's "red zone" or be configured to load RSP with an alternate stack pointer.

  The value UISTACKADJUST must be canonical. If bit 0 is 1, user-interrupt delivery loads RSP with UISTACK-ADJUST; otherwise, it subtracts UISTACKADJUST from RSP. Either way, user-interrupt delivery then aligns RSP to a 16-byte boundary. See Section 11.4.2 for details.

- **UINV: user-interrupt notification vector.**

  This is the vector of those ordinary interrupts that are treated as user-interrupt notifications (Section 11.5.1). When the logical processor receives user-interrupt notification, it processes the user interrupts in the **user posted-interrupt descriptor (UPID)** referenced by UPIDADDR (see below and Section 11.5.2).

- **UPIDADDR: user posted-interrupt descriptor address.**

  This is the linear address of the UPID that the logical processor consults upon receiving an ordinary interrupt with vector UINV.

- **UITTADDR: user-interrupt target table address.**

  This is the linear address of user-interrupt target table (UITT), which the logical processor consults when software invokes the SENDUIPI instruction (see Section 11.7).

- **UITTSZ: user-interrupt target table size.**

  This value is the highest index of a valid entry in the UITT (see Section 11.7).

11.3.2 User-Interrupt MSRs

Some of the state elements identified in Section 11.3.1 can be accessed as user-interrupt MSRs using the RDMSR and WRMSR instructions:

- IA32_UINTR_RR MSR (MSR address 985H). This MSR is an interface to UIRR (64 bits).
- IA32_UINTR_HANDLER MSR (MSR address 986H). This MSR is an interface to the UIHANDLER address (see Section 11.8.1 for canonicality checking).
- IA32_UINTR_STACKADJUST MSR (MSR address 987H). This MSR is an interface to the UISTACKADJUST value (see Section 11.8.1 for canonicality checking).
- IA32_UINTR_MISC MSR (MSR address 988H). This MSR is an interface to the UITTSZ and UINV values. The MSR has the following format:

— bits 31:0 are UITTSZ;

— bits 39:32 are UINV; and

— bits 63:40 are reserved (see Section 11.8.1 for reserved-bit checking).

Because this MSR will share an 8-byte portion of the XSAVE area with UIF (see Section 11.8.2), bit 63 of the MSR will never be used and will always be reserved.

- IA32_UINTR_PD MSR (MSR address 989H). This MSR is an interface to the UPIDADDR address (see Section 11.8.1 for canonicality and reserved-bit checking).
- IA32_UINTR_TT MSR (MSR address 98AH). This MSR is an interface to the UITTADDR address (in addition, bit 0 enables SENDUIPI; see Section 11.8.1 for canonicality and reserved-bit checking).

11.4 EVALUATION AND DELIVERY OF USER INTERRUPTS

A processor determines whether there is a user interrupt to deliver based on UIRR. Section 11.4.1 describes this recognition of pending user interrupts. Once a logical processor has recognized a pending user interrupt, it will deliver it on subsequent instruction boundary by causing a control-flow change asynchronous to software execution. Section 11.4.2 details this process of user-interrupt delivery.

11.4.1 User-Interrupt Recognition

There is a user interrupt pending whenever UIRR ≠ 0.

Any instruction or operation that modifies UIRR updates the logical processor's recognition of a pending user interrupt. The following instructions and operations may need to do this:

- WRMSR to the IA32_UINTR_RR MSR (Section 11.8.1).
- XRSTORS of the user-interrupt state component (Section 11.8.2.4).
- User-interrupt delivery (Section 11.4.2).
- User-interrupt notification processing (Section 11.5.2).
- VMX transitions that load the IA32_UINTR_RR MSR (Section 11.9.3.2 and Section 11.9.4.6).

Each of these instructions or operations results in recognition of a pending user interrupt if it completes with UIRR ≠ 0; if it completes with UIRR = 0, no pending user interrupt is recognized.

Once recognized, a pending user interrupt may be delivered to software; see Section 11.4.2.

11.4.2 User-Interrupt Delivery

If CR4.UINTR = 1 and a user interrupt has been recognized (see Section 11.4.1), it will be delivered at an instruction boundary when the following conditions all hold: (1) UIF = 1; (2) there is no blocking by MOV SS or by POP SS[1]; (3) CPL = 3; (4) IA32_EFER.LMA = CS.L = 1 (the logical processor is in 64-bit mode); and (5) software is not executing inside an enclave.

User-interrupt delivery has priority just below that of ordinary interrupts. It wakes a logical processor from the states entered using the TPAUSE and UMWAIT instructions[2]; it does not wake a logical processor in the shutdown state or in the wait-for-SIPI state.

User-interrupt delivery does not change CPL (it occurs entirely with CPL = 3). The following pseudocode details the behavior of user-interrupt delivery:

1. Execution of the STI instruction does not block delivery of user interrupts for one instruction as it does ordinary interrupts. If a user interrupt is delivered immediately following execution of a STI instruction, ordinary interrupts are not blocked after delivery of the user interrupt.
2. User-interrupt delivery occurs only if CPL = 3. Since the HLT and MWAIT instructions can be executed only if CPL = 0, a user interrupt can never be delivered when a logical processor is an activity state that was entered using one of those instructions.

```
IF UIHANDLER is not canonical in current paging mode
        THEN #GP(0);
FI;
tempRSP := RSP;
IF UISTACKADJUST[0] = 1
        THEN RSP := UISTACKADJUST;
        ELSE RSP := RSP – UISTACKADJUST;
FI;
RSP := RSP & ~FH;                   // force the stack to be 16-byte aligned
Push tempRSP;
Push RFLAGS;
Push RIP;
Push UIRRV;                         // 64-bit push; upper 58 bits pushed as 0
IF shadow stack is enabled for CPL = 3
        THEN ShadowStackPush RIP;
FI;
IF end-branch is enabled for CPL = 3
        THEN IA32_U_CET.TRACKER := WAIT_FOR_ENDBRANCH;
FI;
UIRR[Vector] := 0;
IF UIRR = 0
        THEN cease recognition of any pending user interrupt;
FI;
UIF := 0;
RFLAGS.TF := 0;
RFLAGS.RF := 0;
RIP := UIHANDLER;
```

If UISTACKADJUST[0] = 0, user-interrupt delivery decrements RSP by UISTACKADJUST; otherwise, it loads RSP with UISTACKADJUST. In either case, user-interrupt delivery aligns RSP to a 16-byte boundary by clearing RSP[3:0].

User-interrupt delivery that occurs during transactional execution causes transactional execution to abort and a transition to a non-transactional execution. The transactional abort loads EAX as it would had it been due to an ordinary interrupt. User-interrupt delivery occurs after the transactional abort is processed.

The stack accesses performed by user-interrupt delivery may incur faults (page faults, or stack faults due to canonicality violations). RSP is restored to its original value before such a fault is delivered (memory locations above the top of the stack may have been written). If such a fault produces an error code that uses the EXT bit, that bit will be cleared to 0.

If such a fault occurs, UIRR is not updated and UIF is not cleared and, as a result, the logical processor continues to recognize that a user interrupt is pending and user-interrupt delivery will normally recur after the fault is handled.

If shadow-stack feature of control-flow enforcement technology (CET) is enabled for CPL = 3, user-interrupt delivery pushes the return instruction pointer the shadow stack. If indirect-branch-tracking feature of CET is enabled, user-interrupt delivery transitions the indirect branch tracker to the WAIT_FOR_ENDBRANCH state; an ENDBR64 instruction is expected as first instruction of the user-interrupt handler.

Section 11.9.2.3 discusses VM exits that may occur during user-interrupt delivery.

User-interrupt delivery can be tracked by Architectural Last Branch Records (LBRs), Intel® Processor Trace (Intel® PT), and Performance Monitoring. For both Intel PT and LBRs, user-interrupt delivery is recorded in precisely the same manner as ordinary interrupt delivery. Hence for LBRs, user interrupts fall into the OTHER_BRANCH category, which implies that IA32_LBR_CTL.OTHER_BRANCH[bit 22] must be set to record user-interrupt delivery, and that the IA32_LBR_x_INFO.BR_TYPE field will indicate OTHER_BRANCH for any recorded user interrupt. For Intel PT, control flow tracing must be enabled by setting IA32_RTIT_CTL.BranchEn[bit 13].

User-interrupt delivery will also increment performance counters for which counting BR_INST_RETIRED.FAR_BRANCH is enabled. Some implementations may have dedicated events for counting user-interrupt delivery; see processor-specific event lists at https://download.01.org/perfmon/index/.

11.5 USER-INTERRUPT NOTIFICATION IDENTIFICATION AND PROCESSING

User-interrupt posting is the process by which a platform agent (or software operating on a CPU) records user interrupts in a **user posted-interrupt descriptor** (UPID) in memory. The platform agent (or software) may send an ordinary interrupt (called a **user-interrupt notification**) to the logical processor on which the target of the user interrupt is operating.

A UPID has the format given in Table 11-1.

Table 11-1. Format of User Posted-Interrupt Descriptor — UPID

Bit Position(s)	Name	Description
0	Outstanding notification	If this bit is set, there is a notification outstanding for one or more user interrupts in PIR.
1	Suppress notification	If this bit is set, agents (including SENDUIPI) should not send notifications when posting user interrupts in this descriptor.
15:2	Reserved	User-interrupt notification processing ignores these bits; must be zero for SENDUIPI.
23:16	Notification vector	Used by SENDUIPI.
31:24	Reserved	User-interrupt notification processing ignores these bits; must be zero for SENDUIPI.
63:32	Notification destination	Target physical APIC ID – used by SENDUIPI. In xAPIC mode, bits 47:40 are the 8-bit APIC ID. In x2APIC mode, the entire field forms the 32-bit APIC ID.
127:64	Posted-interrupt requests (PIR)	One bit for each user-interrupt vector. There is a user-interrupt request for a vector if the corresponding bit is 1.

The notation **PIR** (posted-interrupt requests) refers to the 64 posted-interrupt requests in a UPID.

If an ordinary interrupt arrives while CR4.UINTR = IA32_EFER.LMA = 1, the logical processor determines whether the interrupt is a user-interrupt notification. This process is called **user-interrupt notification identification** and is described in Section 11.5.1.

Once a logical processor has identified a user-interrupt notification, it copies user interrupts in the UPID's PIR into UIRR. This process is called **user-interrupt notification processing** and is described in Section 11.5.2.

A logical processor is not interruptible during either user-interrupt notification identification or user-interrupt notification processing or between those operations (when they occur in succession).

11.5.1 User-Interrupt Notification Identification

If CR4.UINTR = IA32_EFER.LMA = 1, a logical processor performs user-interrupt notification identification when it receives an ordinary interrupt. The following algorithm describes the response by the processor to an ordinary interrupt when CR4.UINTR = IA32_EFER.LMA = 1[1]:

1. The local APIC is acknowledged; this provides the processor core with an interrupt vector, V.
2. If V = UINV, the logical processor continues to the next step. Otherwise, an interrupt with vector V is delivered normally through the IDT; the remainder of this algorithm does not apply and user-interrupt notification processing does not occur.

1. If the interrupt arrives between iterations of a REP-prefixed string instruction, the processor first updates state as follows: RIP is loaded to reference the string instruction; RCX, RSI, and RDI are updated as appropriate to reflect the iterations completed; and RFLAGS.RF is set to 1.

3. The processor writes zero to the EOI register in the local APIC; this dismisses the interrupt with vector V = UINV from the local APIC.

User-interrupt notification identification involves acknowledgment of the local APIC and thus occurs only when ordinary interrupts are not masked.

(The behavior described above may be modified in VMX non-root operation; see Section 11.9.2.2 and Section 11.9.3.3.)

If user-interrupt notification identification completes step #3, the logical processor then performs user-interrupt notification processing as described in Section 11.5.2.

An ordinary interrupt that occurs during transactional execution causes the transactional execution to abort and transition to a non-transactional execution. This occurs before user-interrupt notification identification.

An ordinary interrupt that occurs while software is executing inside an enclave causes an asynchronous enclave exit (AEX). This AEX occurs before user-interrupt notification identification.

11.5.2 User-Interrupt Notification Processing

Once a logical processor has identified a user-interrupt notification, it performs **user-interrupt notification processing** using the UPID at the linear address in the IA32_UINTR_PD MSR.

The following algorithm describes user-interrupt notification processing:

1. The logical processor clears the outstanding-notification bit (bit 0) in the UPID. This is done atomically so as to leave the remainder of the descriptor unmodified (e.g., with a locked AND operation).
2. The logical processor reads PIR (bits 127:64 of the UPID) into a temporary register and writes all zeros to PIR. This is done atomically so as to ensure that each bit cleared is set in the temporary register (e.g., with a locked XCHG operation).
3. If any bit is set in the temporary register, the logical processor sets in UIRR each bit corresponding to a bit set in the temporary register (e.g., with an OR operation) and recognizes a pending user interrupt (if it has not already done so).

The logical processor performs the steps above in an uninterruptible manner. Steps #1 and #2 may be combined into a single atomic step. If step #3 leads to recognition of a user interrupt, the processor may deliver that interrupt on the following instruction boundary (see Section 11.4.2).

Although user-interrupt notification processing may occur at any privilege level, all of the memory accesses in steps #1 and #2 are performed with supervisor privilege.

Step #1 and step #2 each access the UPID using a linear address and may therefore incur faults (page faults, or general-protection faults due to canonicality violations). If such a fault produces an error code that uses the EXT bit, that bit will be set to 1.

If such a fault occurs, updates to architectural state performed by the earlier user-interrupt notification identification (Section 11.5.1) remain committed and are not undone; if such a fault occurs at step #2 (if it is not performed atomically with step #1), any update to architectural state performed by step #1 also remains committed. System software is advised to prevent such faults (e.g., by ensuring that no page fault occurs and that the linear address in the IA32_UINTR_PD MSR is canonical with respect to the paging mode in use).

(System software executing in VMX non-root operation is not normally expected to prevent VM exits due to event such as EPT violations. Section 11.9.2.3 discusses the treatment of user-interrupt notification processing when such events occur.)

The user-interrupt notification identification that precedes user-interrupt notification processing may occur due to an ordinary interrupt (Section 11.5.1), a virtual interrupt (Section 11.9.2.2), or an interrupt injected by VM entry (Section 11.9.3.3). The following items specify the activity state of the logical processor for each of these cases of user-interrupt notification processing:

- If user-interrupt notification identification was due to an ordinary interrupt or a virtual interrupt and the logical processor had been in the HLT state before that interrupt, it returns to the HLT state following user-interrupt notification processing.
- If user-interrupt notification identification was due to an interrupt injected by VM entry and the activity-state field in the guest-state area of the VMCS indicated the HLT state, the logical processor enters the HLT state following user-interrupt notification processing.

- In all other cases, the logical processor is in the active state following user-interrupt notification processing.

Section 11.9.2.3 discusses VM exits that may occur during user-interrupt notification processing.

11.6 NEW INSTRUCTIONS

The user-interrupt architecture defines new instructions for control-flow transfer and access to new state. UIRET is a new instruction to effect a return from a user-interrupt handler. Other new instructions allow access by user code to UIF. User IPIs also use a new instruction, SENDUIPI. See Section 2.1, "Instruction Set Reference" for details on instructions.

11.7 USER IPIS

Processors support the sending of interprocessor user interrupts (user IPIs) through a user-interrupt target table (configured by system software) and the SENDUIPI instruction (executed by application software). Operation of SENDUIPI is presented in Section 2.1, "Instruction Set Reference".

The **user-interrupt target table (UITT)** is a data structure composed of 16-byte entries. Each UITT entry (UITTE) has the following format:

- Bit 0: **V**, a valid bit.
- Bits 7:1 are reserved and must be 0.
- Bits 15:8: **UV**, the user-interrupt vector (in the range 0–63, so bits 15:14 must be 0).
- Bits 63:16 are reserved.
- Bits 127:64: **UPIDADDR**, the linear address of a UPID (64-byte aligned, so bits 69:64 must be 0).

The UITT is located at the linear address UITTADDR; UITTSZ is the highest index of a valid entry in the UITT (thus, the number of entries in the UITT is UITTSZ + 1).

11.8 LEGACY INSTRUCTION SUPPORT

Certain instructions support the user-interrupt architecture. The RDMSR and WRMSR instructions support access to the user-interrupt MSRs (Section 11.8.1). The architecture is also supported by the XSAVE feature set (Section 11.8.2).

11.8.1 Support by RDMSR and WRMSR

The RDMSR and WRMSR instructions support normal read and write operations for the user-interrupt MSRs defined in Section 11.3. These operations are supported even if CR4.UINTR = 0. The following items identify points that are specific to these MSRs:

- IA32_UINTR_RR MSR (MSR address 985H).
  - This MSR holds the current value of UIRR.
  - Following a WRMSR to this MSR, the logical processor recognizes a pending user interrupt if and only if some bit is set in the MSR.
- IA32_UINTR_HANDLER MSR (MSR address 986H).
  - This MSR holds the current value of UIHANDLER. This is a linear address that must be canonical relative to the maximum linear-address width supported by the processor.[1]
  - WRMSR to this MSR causes a general-protection fault (#GP) if its source operand does not meet this requirement.

1. CPUID.80000008H:EAX[15:8] enumerates the maximum linear-address width supported by the processor.

- IA32_UINTR_STACKADJUST MSR (MSR address 987H).
  - This MSR holds the current value of UISTACKADJUST. This value includes a linear address that must be canonical relative to the maximum linear-address width supported by the processor.
  - WRMSR to this MSR causes a general-protection fault (#GP) if its source operand does not meet this requirement.
  - Bit 0 of this MSR corresponds to UISTACKADJUST[0], which controls how user-interrupt delivery updates the stack pointer. WRMSR may set it to either 0 or 1.
- IA32_UINTR_MISC MSR (MSR address 988H).
  - Bits 31:0 of this MSR hold the current value of UITTSZ, while bits 39:32 hold the current value of UINV.
  - Bits 63:40 of this MSR are reserved. WRMSR causes a #GP if it would set any of those bits (if EDX[31:8] ≠ 000000H).
  - Because this MSR shares an 8-byte portion of the XSAVE area with UIF (see Section 11.8.2), bit 63 of the MSR will never be used and will always be reserved.
- IA32_UINTR_PD MSR (MSR address 989H).
  - This MSR holds the current value of UPIDADDR. This is a linear address that must be canonical relative to the maximum linear-address width supported by the processor.
  - WRMSR to this MSR causes a general-protection fault (#GP) if its source operand does not meet this requirement.
  - Bits 5:0 of this MSR are reserved. WRMSR causes a #GP if it would set any of those bits (if EAX[5:0] ≠ 000000b).
- IA32_UINTR_TT MSR (MSR address 98AH).
  - Bit 63:4 of this MSR holds the current value of UITTADDR. This a linear address that must be canonical relative to the maximum linear-address width supported by the processor.
  - WRMSR to this MSR causes a general-protection fault (#GP) if its source operand does not meet this requirement.
  - Bits 3:1 of this MSR are reserved. WRMSR causes a #GP if it would set any of those bits (if EAX[3:1] ≠ 000b).
  - Bit 0 of this MSR determines whether the SENDUIPI instruction is enabled. WRMSR may set it to either 0 or 1.

11.8.2 Support by the XSAVE Feature Set

The state identified in Section 11.3 may be specific to an OS-managed user thread, and system software would then need to change the values of this state when changing user threads. This context management is facilitated by adding support for this state to the XSAVE feature set. This section describes that support.

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). Each such state component corresponds to an XSAVE-supported feature. 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. Some state components are supervisor state components. The XSAVE feature supports **supervisor state components** with only the XSAVES and XRSTORS instructions.

Section 11.8.2.1 defines a new supervisor state component for user interrupts. Section 11.8.2.2 explains XSAVE-specific enumeration of the user-interrupt state component. Section 11.8.2.3 specifies how XSAVES will manage this state component, and Section 11.8.2.4 does the same for XRSTORS.

11.8.2.1 User-Interrupt State Component

The XSAVE feature set will manage the user-interrupt registers with a supervisor **user-interrupt state component**. Bit 14 in the state-component bitmaps is assigned for the user-interrupt state component; this specification will refer to that position with the notation "UINTR." System software enables the processor to manage the user-

interrupt state component by setting IA32_XSS.UINTR. (This implies that XSETBV will not allow XCR0.UINTR to be set.)

The user-interrupt state component comprises 48 bytes in memory with the following layout:

- Bytes 7:0 are for UIHANDLER (the IA32_UINTR_HANDLER MSR).
- Bytes 15:8 are for UISTACKADJUST (the IA32_UINTR_STACKADJUST MSR).
- Bytes 23:16 are for UITTSZ and UINV (from the IA32_UINTR_MISC MSR) and for UIF, organized as follows:
  - Byte 19:16 is for UITTSZ (bits 31:0 of the IA32_UINTR_MISC MSR).
  - Byte 20 is for UINV (bits 39:32 of the IA32_UINTR_MISC MSR).
  - Bytes 22:21 (2 bytes) and bits 6:0 of byte 23 are reserved. (They may be used for bits 62:40 if the IA32_UINTR_MISC MSR, if they are defined in the future.)
  - Bit 7 of byte 23 is for UIF.

  Because bit 7 of byte 23 is for UIF (which is not part of the IA32_UINTR_MISC MSR), software that reads a value from bytes 23:16 should clear bit 63 of that 64-bit value before attempting to write it to the IA32_UINTR_MISC MSR.
- Bytes 31:24 are for UPIDADDR (the IA32_UINTR_PD MSR).
- Bytes 39:32 are for UIRR (the IA32_UINTR_RR MSR).
- Bytes 47:40 are for UITTADDR (the IA32_UINTR_TT MSR, including the bit 0, the valid bit).

The user-interrupt state component is in its initial state if all user-interrupt registers are zero.

Certain portions of a supervisor state component may be identified as **master-enable state**. XSAVES and XRSTORS treat this state specially. UINV is the master-enable state for the user-interrupt state component. See Section 11.8.2.3 and Section 11.8.2.4 for the treatment of this state by XSAVES and XRSTORS, respectively.

11.8.2.2 XSAVE-Related Enumeration

The XSAVE feature set includes an architecture to enumerate details about each XSAVE-supported state component. The following items provide details of the XSAVE-specific enumeration of the user-interrupt state component:

- CPUID.(EAX=0DH,ECX=1):EBX enumerates the size in bytes of an XSAVE area containing all states currently enabled by XCR0 | IA32_XSS. When IA32_XSS.UINTR[bit 14] = 1, this value will includes the 48 bytes required for the user-interrupt state component (see Section 11.8.2.1).
- CPUID.(EAX=0DH,ECX=1):ECX.UINTR[bit 14] is enumerated as 1, indicating that the user-interrupt state component is a supervisor state component and that IA32_XSS.UINTR can be set to 1.
- CPUID.(EAX=0DH,ECX=14):EAX is enumerated as 48 (30H), the size in bytes of the user-interrupt state component.
- CPUID.(EAX=0DH,ECX=14):EBX is enumerated as 0 (this is the case for any supervisor state component).
- CPUID.(EAX=0DH,ECX=14):ECX[0] is enumerated as 1, indicating that the user-interrupt state component is a supervisor state component.
- CPUID.(EAX=0DH,ECX=14):ECX[1] is enumerated as 0, indicating that the user-interrupt state component need not be aligned on a 64-byte boundary.
- CPUID.(EAX=0DH,ECX=14):ECX[31:2] are reserved and enumerated as 0.
- CPUID.(EAX=0DH,ECX=14):EDX is reserved and enumerated as 0.

11.8.2.3 XSAVES

The management of the user-interrupt state component by XSAVES follows the architecture of the XSAVE feature set. The following items identify points that are specific to saving the user-interrupt state component:

- XSAVES writes the user-interrupt registers to the user-interrupt state component using the format specified in Section 11.8.2.1.
- XSAVES stores zeros to bits and bytes identified in Section 11.8.2.1 as reserved.

- The values saved for UIHANDLER, UPIDADDR, and UITTADDR are always canonical relative to the maximum linear-address width enumerated by CPUID[1].
- After saving the user-interrupt state component, XSAVES clears UINV. (UINV is IA32_UINTR_MISC[39:32]; XSAVES does not modify the remainder of that MSR.)

11.8.2.4 XRSTORS

The management of the user-interrupt state component by XRSTORS follows the architecture of the XSAVE feature set. The following items identify points that are specific to restoring the user-interrupt state component:

- Before restoring the user-interrupt state component, XRSTORS verifies that UINV is 0. If it is not, XRSTORS causes a general-protection fault (#GP) before loading any part of the user-interrupt state component. (UINV is IA32_UINTR_MISC[39:32]; XRSTORS does not check the contents of the remainder of that MSR.)
- If the instruction mask and XSAVE area used by XRSTORS indicates that the user-interrupt state component should be loaded from the XSAVE area, XRSTORS reads the user-interrupt registers from the XSAVE area using the format identified in Section 11.8.2.1. The values read cause a general-protection fault (#GP) in any of the following cases:
  - If any of the bits and bytes identified as reserved is not zero;
  - If the value to be loaded into any one of UIHANDLER, UISTACKADJUST, UPIDADDR, or UITTADDR is not canonical relative to the maximum linear-address width enumerated by CPUID; or
  - If the value to be loaded into either UPIDADDR or UITTADDR sets any of the bits reserved in that register (the reserved bits are bits 5:0 of UPIDADDR and bits 3:1 of UITTADDR; bit 0 of UITTADDR is the valid bit for SENDUIPI).
- If XRSTORS causes a fault or a VM exit after loading any part of the user-interrupt state component, XRSTORS clears UINV before delivering the fault or VM exit. (Other elements of user-interrupt state, including other parts of the IA32_UINTR_MISC MSR, may retain the values that were loaded by XRSTORS.)
- After a non-faulting execution of XRSTORS that loads the user-interrupt state component, the logical processor recognizes a pending user interrupt if and only if some bit is set in the new value of UIRR (see Section 11.4.1).

11.9 VMX SUPPORT

The VMX architecture supports virtualization of the instruction set and its system architecture. Certain extensions are needed to support virtualization of user interrupts. This section describes these extensions.

11.9.1 VMCS Changes

A new VM-exit control is defined called **clear UINV**. The control has been assigned position 27.

A new VM-entry control is defined called **load UINV**. The control has been assigned position 19.

Guest UINV is a new 16-bit field in the guest-state area (encoding to be determined), corresponding to UINV. The VMCS-field encoding for the guest UINV is 00000814H.

The guest UINV field exists only on processors that support the 1-setting of either the "clear UINV" VM-exit control or the "load UINV" VM-entry control.

11.9.2 Changes to VMX Non-Root Operation

This section describes changes to VMX non-root operation to support user interrupts.

1. They need might not be canonical relative to the current paging mode if it supports only smaller linear addresses.

11.9.2.1 Treatment of Ordinary Interrupts

Outside of VMX non-root operation, a logical processor with CR4.UINTR = IA32_EFER.LMA = 1 responds to an ordinary interrupt by performing user-interrupt notification identification (Section 11.5.1) and, if it succeeds, user-interrupt notification processing (see Section 11.5.2).

In VMX non-root operation, the treatment of ordinary interrupts depends on the setting of the "external-interrupt exiting" VM-execution control:

- If the control is 0, user-interrupt notification identification and, if it succeeds, user-interrupt notification processing occur normally.
- If the control is 1, the logical processor does not perform user-interrupt notification identification (or user-interrupt notification processing). Instead, legacy behavior applies: a VM exit occurs (unless the interrupt causes posted-interrupt processing for interrupt virtualization).

11.9.2.2 Treatment of Virtual Interrupts

If the "virtual-interrupt delivery" VM-execution control is 1, a logical processor in VMX non-root operation may deliver virtual interrupts to guest software. This is done by using a virtual interrupt's vector to select a descriptor from the IDT and using that descriptor to deliver the interrupt.

If CR4.UINTR = IA32_EFER.LMA = 1, the delivery of virtual interrupts is modified. Specifically, the logical processor first performs a form of user-interrupt notification identification (modified as indicated from the definition in Section 11.5.1)[1]:

1. Instead of acknowledging the local APIC (as specified in Section 11.5.1), the logical processor performs the initial steps of virtual-interrupt delivery:
   V := RVI;
   VISR[V] := 1;
   SVI := V;
   VPPR := V & F0H;
   VIRR[V] := 0;
   IF any bit is set in VIRR
   THEN RVI := highest index of bit set in VIRR
   ELSE RVI := 0;
   FI;
   cease recognition of any pending virtual interrupt;
   (RVI, SVI, VIRR, VISR, and VPPR are defined by the architecture for virtual interrupts.)
2. If V = UINV, the logical processor continues to the next step. Otherwise, a virtual interrupt with vector V is delivered normally through the IDT; the remainder of this algorithm does not apply and user-interrupt notification processing does not occur.
3. Instead of writing zero to the EOI register in the local APIC (as specified in Section 11.5.1), the logical processor performs the initial steps of EOI virtualization:
   VISR[V] := 0;
   IF any bit is set in VISR
   THEN SVI := highest index of bit set in VISR
   ELSE SVI := 0;
   FI;
   perform PPR virtualization;

   Unlike EOI virtualization resulting from a guest write to the EOI register (as defined for virtual-interrupt delivery), the logical processor does not check the EOI-exit bitmap as part of this modified form of user-interrupt notification identification, and the corresponding VM exits cannot occur.

This modified form of user-interrupt notification identification occurs only when virtual interrupts are not masked (e.g., only if RFLAGS.IF = 1).

1. If virtual-interrupt delivery occurs between iterations of a REP-prefixed string instruction, the processor will first update state as follows: RIP is loaded to reference the string instruction; RCX, RSI, and RDI are updated as appropriate to reflect the iterations completed; and RFLAGS.RF is set to 1.

If this modified form of user-interrupt notification identification completes step #3, the logical processor then performs user-interrupt notification processing as specified in Section 11.5.2.

A logical processor is not interruptible during this modified form of user-interrupt notification identification or between it and any subsequent user-interrupt notification processing.

A virtual interrupt that occurs during transactional execution causes the transactional execution to abort and transition to a non-transactional execution. This occurs before this modified form of user-interrupt notification identification.

A virtual interrupt that occurs while software is executing inside an enclave normally causes an asynchronous enclave exit (AEX). Such an AEX would occur before this modified form of user-interrupt notification identification.

11.9.2.3 VM Exits Incident to New Operations

The user-interrupt architecture introduces user-interrupt delivery (Section 11.4.2) and user-interrupt notification processing (Section 11.5.2).

These operations access memory using linear addresses: user-interrupt delivery writes to the stack; user-interrupt notification processing read and writes a UPID at the linear address in the IA32_UINTR_PD MSR[1]. Such memory accesses may incur faults (#GP, #PF, etc.) that may cause VM exits (depending on the configuration of the exception bitmap in the VMCS). In addition, memory accesses in VMX non-root operation may incur APIC-access VM exits, EPT violations, EPT misconfigurations, page-modification log-full VM exits, and SPP-induced VM exits[2].

In general, such VM exits are treated normally. The following items present special cases:

- An APIC-access VM exit, an EPT violation, a page-modification log-full VM exit, or SPP-induced VM exit that occurs during user-interrupt delivery will set bit 16 of the exit qualification to 1, indicating that the VM exit was "asynchronous to instruction execution."
- Any VM exit that occurs during user-interrupt notification processing (including those due to faults) will set the IDT-vectoring information field to indicate that the VM exit was incident to an interrupt with the vector UINV (to the value 8000000xyH, where xy = UINV). If the logical processor would have entered the HLT state following user-interrupt notification processing (see Section 11.5.2), the VM exit saves "HLT" into the activity-state field of the guest-state area of the VMCS.

11.9.2.4 Access to the User-Interrupt MSRs

The MSR bitmaps do not affect a logical processor's ability to read or write the user-interrupt MSRs as part of user-interrupt recognition, user-interrupt delivery, user-interrupt notification identification, or user-interrupt notification processing. The MSR bitmaps control only operation of the RDMSR and WRMSR instructions.

11.9.2.5 Operation of SENDUIPI

As noted in Section 2.1, the operation of SENDUIPI concludes with the following step (executed under certain conditions):

```
IF local APIC is in x2APIC mode
        THEN send ordinary IPI with vector tempUPID.NV
                to 32-bit physical APIC ID tempUPID.NDST;
        ELSE send ordinary IPI with vector tempUPID.NV
                to 8-bit physical APIC ID tempUPID.NDST[15:8];
FI;
```

Outside of VMX non-root operation, the logical processor will send this IPI by writing to the local APIC's interrupt-command register (ICR). In VMX non-root operation, behavior depends on the settings of the "use TPR shadow" and "virtualize APIC accesses" VM-execution controls:

1. The new UIRET and SENDUIPI instructions also access memory with linear addresses. Because they are instructions, the existing VMX architecture fully defines the operation of any resulting VM exits.
2. SPP-induced VM exits include both SPP misses and SPP misconfigurations.

1. If the "use TPR shadow" VM-execution control is 0, the behavior is not modified: the logical processor sends the specified IPI by writing to the local APIC's ICR as specified above.
2. If the "use TPR shadow" VM-execution control is 1 and the "virtualize APIC accesses" VM-execution control is 0, the logical processor virtualizes the sending of an x2APIC-mode IPI by performing the following steps:
   a. Writing the 64-bit value Z to offset 300H on the virtual-APIC page (VICR), where Z[7:0] = tempUPID.NV (the 8-bit virtual vector), Z[63:32] = tempUPID.NDST (the 32-bit virtual APIC ID) and Z[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
   b. Causing an APIC-write VM exit with exit qualification 300H.

   APIC-write VM exits are trap-like: the value of CS:RIP saved in the guest-state area of the VMCS references the instruction after SENDUIPI. The basic exit reason for an APIC-write VM exit is "APIC write" (56). The exit qualification is the page offset of the write access that led to the VM exit — 300H in this case.
3. If the "use TPR shadow" and "virtualize APIC accesses" VM-execution controls are both 1, the logical processor virtualizes the sending of an xAPIC-mode IPI by performing the following steps:
   a. Writing the 32-bit value X to offset 310H on the virtual-APIC page (VICR_HI), where X[31:24] = tempUPID.NDST[15:8] (the 8-bit virtual APIC ID) and X[23:0] = 000000H[1].
   b. Writing the 32-bit value Y to offset 300H on the virtual-APIC page (VICR_LO), where Y[7:0] = tempUPID.NV (the 8-bit virtual vector) and Y[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
   c. Causing an APIC-write VM exit with exit qualification 300H (see above).

11.9.3 Changes to VM Entries

This section describes how the user-interrupt architecture affects the operation of VM entries.

11.9.3.1 Checks on the Guest-State Area

If the "load UINV" VM-entry control is 1, VM entries ensure that bits 15:8 of the guest UINV field are 0. VM entry fails if this check fails. Such failures are treated as all VM-entry failures that occur during or after loading guest state.

11.9.3.2 Loading MSRs

VM entries may load MSRs from the VM-entry MSR-load area. If a VM entry loads any of the user-interrupt MSRs, it does so in a manner consistent with that of WRMSR (see Section 11.8.1).

11.9.3.3 Event Injection

The legacy behavior of VM entry is such that, if the VM-entry interruption-information field has a value of the form 8000000xyH, VM entry injects an interrupt with vector V = xyH. This is done by using V to select a descriptor from the IDT and using that descriptor to deliver the interrupt.

If bit 25 (UINTR) is set to 1 in the CR4 field in the guest-state area of the VMCS and the "IA-32e mode guest" VM-entry control is 1, VM entry is modified if it is injecting an interrupt. Specifically, the logical processor first performs a form of user-interrupt notification identification (modified as indicated from the definition in Section 11.5.1):

1. This step, acknowledging the local APIC, is omitted.
2. If UINV = V (where V is the vector of the interrupt being injected), the logical processor continues to the next step[2]. Otherwise, an interrupt with vector V is delivered normally through the IDT; the remainder of this algorithm does not apply and user-interrupt notification processing does not occur.

---

1. For xAPIC mode (which is virtualized if the "virtualize APIC accesses" VM-execution control is 1), the destination APIC ID is in byte 1 (not byte 0) of the UPID's 4-byte NDST field.
2. If VM entry loaded UINV from the VMCS, the checking of UINV is based on the value loaded.

3. This step, writing zero to the EOI register in the local APIC, is omitted.

Because VM entry allows interrupt injection only when interrupts are not masked in a guest (e.g., when RFLAGS is being loaded with a value that sets bit 9, IF), this modified form of user-interrupt notification identification occurs only when virtual interrupts are not masked.

If user-interrupt notification identification completes step #2, the logical processor then performs user-interrupt notification processing as detailed Section 11.5.2.

A logical processor is not interruptible during this modified form of user-interrupt notification identification or between it and any subsequent user-interrupt notification processing.

This change in VM-entry event injection occurs as long as UINTR is set to 1 in the CR4 field in the guest-state area of the VMCS and the "IA-32e mode guest" VM-entry control is 1; the settings of the "external-interrupt exiting" and "virtual-interrupt delivery" VM-execution controls do not affect this change.

11.9.3.4 User-Interrupt Recognition After VM Entry

A VM entry results in recognition of a pending user interrupt if it completes with UIRR ≠ 0; if it completes with UIRR = 0, no pending user interrupt is recognized.

11.9.4 Changes to VM Exits

This section describes how the user-interrupt architecture affects the operation of VM exits.

11.9.4.1 Recording VM-Exit Information

As noted in Section 11.9.2.3, an APIC-access VM exit, an EPT violation, or a page-modification log-full VM exit that occurs during user-interrupt delivery sets bit 16 of the exit qualification to 1, indicating that the VM exit was "asynchronous to instruction execution."

A VM exit that occurs during user-interrupt notification processing sets the IDT-vectoring information field to indicate that the VM exit was incident to an interrupt with the vector UINV (to the value 8000000xyH, where xy = UINV).

11.9.4.2 Saving Guest State

If a processor supports user interrupts, every VM exit saves UINV into the guest UINV field in the VMCS (bits 15:8 of the field are cleared).

11.9.4.3 Saving MSRs

VM exits may save MSRs into the VM-exit MSR-store area. If a VM exit saves any of the user-interrupt MSRs, it does so in a manner consistent with that of RDMSR (see Section 11.8.1).

11.9.4.4 Loading Host State

If the "clear UINV" VM-exit control is 1, VM exit clears UINV.

11.9.4.5 Loading MSRs

VM exits may load MSRs from the VM-exit MSR-load area. If a VM exit loads any of the user-interrupt MSRs, it does so in a manner consistent with that of WRMSR (see Section 11.8.1).

11.9.4.6 User-Interrupt Recognition After VM Exit

A VM exit results in recognition of a pending user interrupt if it completes with UIRR ≠ 0; if it completes with UIRR = 0, no pending user interrupt is recognized.

11.9.5 Changes to VMX Capability Reporting

Section 11.9.1 identified a new VM-exit control "clear UINV" at bit position 27. Processors supporting the 1-settings of this control enumerate that support by setting bit 59 in each of the IA32_VMX_EXIT_CTLS MSR (index 483H) and the IA32_VMX_TRUE_EXIT_CTLS MSR (index 48FH).

Section 11.9.1 identified a new VM-entry control "load UINV" at bit position 19. Processors supporting the 1-settings of this control enumerate that support by setting bit 51 in each of the IA32_VMX_ENTRY_CTLS MSR (index 484H) and the IA32_VMX_TRUE_ENTRY_CTLS MSR (index 490H).

Section 11.2 defined CR4[25] as CR4.UINTR, a new bit that can be set in CR4. Processors supporting the 1-settings of that bit in VMX operation enumerate that support by setting bit 25 in the IA32_VMX_CR4_FIXED1 MSR (index 489H).

CHAPTER 12
PERFORMANCE MONITORING UPDATES

This chapter covers performance monitoring updates for processors based on Alder Lake Client microarchitecture and processors based on Sapphire Rapids Server microarchitecture.

12.1 PERFORMANCE METRICS

For Alder Lake Client and Sapphire Rapids Server microarchitectures, the core PMU supports the built-in metrics that were introduced in the Ice Lake microarchitecture PMU. It also complies with the PERF_METRICS MSR and its software interface as described in section 18.3.9, "Next Generation Intel® Core™ Processor Performance Monitoring Facility", in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

This core PMU extends the PERF_METRICS MSR to feature TMA method level 2 metrics, as shown in Figure 12-1.

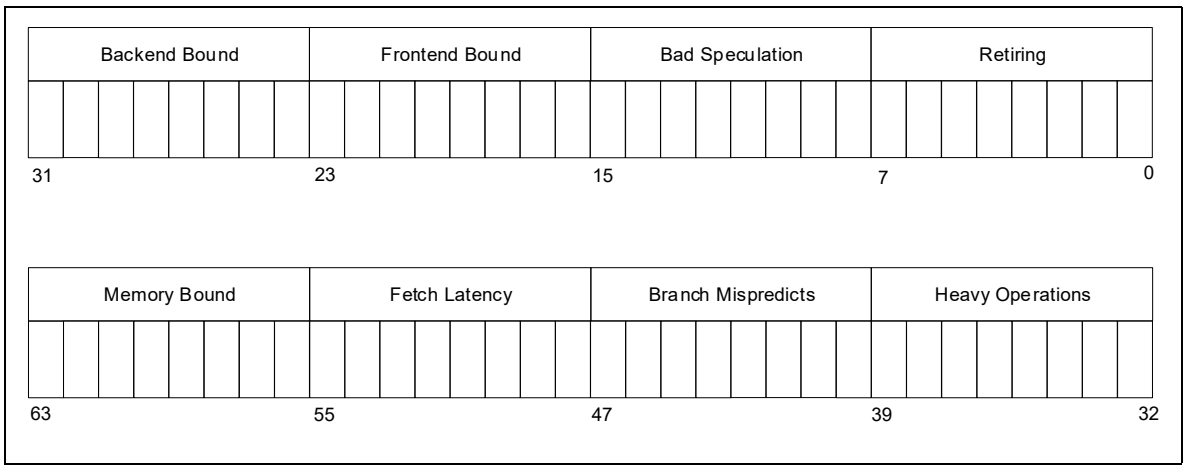


Figure 12-1. PERF_METRICS MSR Definition for Alder Lake Client and Sapphire Rapids Server Microarchitectures

The lower half of the register is the TMA level 1 metrics (legacy). The upper half is also divided into four 8-bit fields, each of which is an integer fraction of 255. Additionally, each of the new level 2 metrics in the upper half is a subset of the corresponding level 1 metric in the lower half (that is, its parent node per the TMA hierarchy). This enables software to deduce the other four level 2 metrics by subtracting corresponding metrics as shown in Figure 12-2.

```
Light_Operations = Retiring - Heavy_Operations
Machine_Clears = Bad_Speculation - Branch_Mispredicts
Fetch_Bandwidth = Frontend_Bound - Fetch_Latency
Core_Bound = Backend_Bound - Memory_Bound
```

Figure 12-2. Deducing Implied Level 2 Metrics in the Core PMU for Alder Lake Client and Sapphire Rapids Server Microarchitectures

The PERF_METRICS MSR and fixed counter 3 of the core PMU for Alder Lake Client and Sapphire Rapids Server microarchitectures feature 12 metrics in total that cover all level 1 and level 2 nodes of the TMA hierarchy.

12.2 PROCESSOR EVENT BASED SAMPLING (PEBS) FACILITY

12.2.1 Instruction-Accurate PDIR (PDIR++)

Instruction-accurate PDIR (PDIR++) is an enhancement to the Precise Distribution of Instructions Retired (PDIR) facility that was introduced in Sandy Bridge microarchitecture as described in Section 18.3.4.4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

With PDIR++, PEBS is taken on the next instruction after the one that caused the overflow. If the instruction is macro-fused, PEBS is taken once the macro-fusion pair of instructions retire. This facility is available only on Fixed Counter 0.

For this facility to work, the overflow counter must be initialized with a reload-value < -127, which implies a Sample After Value > 127.

NOTE

Macro-fusion may forbid a particular instruction from obtaining PEBS samples when using a fixed reload-value on a tight endless loop. Therefore, it is recommended to "normalize" samples for each basic-block of instructions. This implies distributing the total sample counts evenly over all instructions within a basic block.

12.2.2 Precise Distribution (PDist)

On previous microarchitecture, a multi-cycle skid may be noticed on precise events using PEBS. Alder Lake Client and Sapphire Rapids Server microarchitectures introduce a new Precise Distribution (PDist) facility that eliminates the skid when a precise event is programmed on general (programmable) counter 0. It follows a mechanism similar to PDIR++, described in Section 12.2.1, where PEBS is pended on the exact instruction that causes the overflow.

There is no "+1 legacy PEBS behavior" after overflow as the case is with traditional PEBS (which is still supported by general-purpose performance monitoring counters 1-7).

The following events do not support the PDist behavior:

- INST_RETIRED.*
- MEM_TRANS_RETIRED.LOAD_LATENCY_*

For this facility to work, the overflow counter must be initialized with a reload-value < -127, which implies a Sample After Value of > 127.

NOTE

Precise events on PC0 may tune the reload value differently than general-purpose performance monitoring counters 1-7 when attempting to improve accuracy via prime reload values.

12.2.3 Load Latency

Data Source: The encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 12-1. In the descriptions, local memory refers to system memory physically attached to a processor package, and remote memory refers to system memory or cache physically attached to another processor package (in a server product).

Table 12-1. Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)

Encoding	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request).
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. This request was merged into an outstanding cache miss to same cache-line address.
03H	L2 HIT. This request was satisfied by the L2 cache.

Table 12-1. Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures) (Contd.)

Encoding	Description
04H	L3 HIT. This request was satisfied by the L3 cache with no coherency actions performed (snooping).
05H	XCORE MISS. This request was satisfied by the L3 cache but involved a coherency check in some sibling core(s).
06H	XCORE HIT. This request was satisfied by the L3 cache but involved a coherency check that hit a non-modified copy in a sibling core.
07H	XCORE FWD. This request was satisfied by a sibling core where either a modified (cross-core HITM) or a non-modified (cross-core FWD) cache-line copy was found.
08H	Local Far Memory. This request has missed the L3 cache and was serviced by local far memory.
09H	Remote Far Memory. This request has missed the L3 cache and was serviced by remote far memory.
0AH	Local Near Memory. This request has missed the L3 cache and was serviced by local near memory.
0BH	Remote Near Memory. This request has missed the L3 cache and was serviced by remote near memory.
0CH	Remote FWD. This request has missed the L3 cache and a non-modified cache-line copy was forwarded from a remote cache.
0DH	Remote HITM. This request has missed the L3 cache and a modified cache-line was forwarded from a remote cache.
0EH	I/O. Request of input/output operation.
0FH	UC. The request was to uncacheable memory.

12.2.4 Store Latency

Store latency support is available on processors based on Alder Lake Client and Sapphire Rapids Server microarchitectures. Store latency is a PEBS extension that provides a means to profile store memory accesses in the system. It complements the load latency extension.

Store latency leverages the PEBS facility where it can provide additional information about sampled stores. The additional information includes the data address, memory auxiliary info (e.g., Data Source, STLB miss) and the latency of the store access. Normal stores (those preceded with a read-for-ownership) as well as streaming stores are supported by store latency.

Memory store operations typically do not limit performance since they update the memory with no operation that directly depends on them. Thus, data out of this facility should be carefully used once stores are suspected as a performance limiter; for example, once the TMA node of Backend_Bound.Memory_Bound.Store_Bound is flagged.[1]

To enable the store latency facility, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- Program the MEM_TRANS_RETIRED.STORE_SAMPLE event on programmable counter 0 (IA32_PerfEvtSel0[15:0] = 2CDH).
- Set IA32_PEBS_ENABLE[0].

The store latency information is written into a PEBS record as shown in Table 12-2.

The store latency relies on the PEBS facility, so the PEBS configuration must be completed first. Unlike load latency, there is no option to filter on subset of stores that exceed a certain threshold.

1. For more details about the method, refer to Section B.1, “Top-Down Analysis Method” of the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

12.3 ADAPTIVE PEBS

12.3.1 Memory Access Info

Table 12-2 describes the updated Memory Access Info group. Each field is 64-bits with the offset specified in bytes relative to the start of the Memory Access Info group within the Adaptive PEBS record. See Section 18.9.2.2.2, "Memory Access Info", in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B* as a reference. New fields in Alder Lake Client and Sapphire Rapids Server microarchitectures are shaded gray.

Table 12-2. Memory Access Info Group

Field Name	Sub-field Name	Bits	Description
Access Address (offset 0x0)	DLA	[63:0]	This field reports the data linear address (DLA) of the memory access in canonical form. A zero value indicates the processor could not retrieve the address of the particular access.
Access Info (offset 0x8)	Data Src	[3:0]	An encoded value indicating the memory hierarchy source which satisfied the access. These encodings are detailed in Table 12-1. A zero value indicates the processor could not retrieve the data source of the particular access.
	STLB-miss	[4]	A value of 1 indicates the access has missed the Second-level TLB (STLB).
	IsLock	[5]	A value of 1 indicates the access was part of a locked (atomic) memory transaction.
	Data Blk	[6]	A value of 1 indicates the load was blocked since its data could not be forwarded from a preceding store.
	Address Blk	[7]	A value of 1 indicates the load was blocked due to potential address conflict with a preceding store.
Access Latency (offset 0x10)	Instruction Latency	[15:0]	Measured instruction latency in core cycles. For loads, the latency starts by the dispatch of the load operation for execution and lasts until completion of the instruction it belongs to. This field includes the entire latency including time for data-dependency resolution or TLB lookups.
	Cache Latency	[47:32]	Measured cache access latency in core cycles. For loads, the latency starts by the actual cache access until the data is returned by the memory subsystem. For stores, the latency starts when the demand write accesses the L1 data-cache and lasts until the cacheline write is completed in the memory subsystem. This field does not include non-data-cache latency such as memory ordering checks or TLB lookups.
TSX (offset 0x18)	HLE Info		

To determine which fields are supported for certain performance monitoring events, consult the Memory Info attribute in the event list at 01.org.

NOTE

There may be additional block reasons, even if DataBlk and AddressBlk are both clear, e.g., non-optimal instruction latency.

12.4 PERFORMANCE MONITORING EVENT LIST

12.4.1 Counter Restrictions Simplification

Alder Lake Client and Sapphire Rapids Server microarchitectures allow identification of performance monitoring events with counter restrictions based on event encodings. The general rule is: Event Codes < 0x90 are restricted to counters 0-3. Event Codes ≥ 0x90 are likely to have no restrictions. Table 12-3 lists the exceptions to this rule.

Table 12-3. Special Performance Monitoring Events with Counter Restrictions

Event Encoding[1]	Event Name	Counter Restriction
xx3C	CPU_CLK_UNHALTED.*	0-7 (No restriction for all architectural events.)
xx2E	LONGEST_LAT_CACHE.*	
xxDx	MEM_*_RETIRED.*	0-3
01A3, 02A3, 08A3	Some CYCLE_ACTIVITY sub-events	0-3
02CD	MEM_TRANS_RETIRED.STORE_SAMPLE	0
04A4	TOPDOWN.BAD_SPEC_SLOTS	0
08A4	TOPDOWN.BR_MISPREDICT_SLOTS	
xxCE	AMX_OPS_RETIRED	0

NOTES:

1. Linux perf rUUEE syntax, where UU is the Unit Mask field and EE is the Event Select (also known as Event Code) field in the IA32_PERFEVTSELx MSRs.

CHAPTER 13
ENHANCED HARDWARE FEEDBACK INTERFACE (EHFI)

Intel processors that enumerate CPUID.06H.0H:EAX.HW_FEEDBACK[bit 19] as 1 support the hardware feedback interface (HFI). The hardware feedback interface is described in Section 14.6 "Hardware Feedback Interface" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B.*

Intel processors that enumerate CPUID.06H.0H:EAX[bit 23] as 1 support the enhanced hardware feedback interface (EHFI). Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a memory resident table and software thread specific index (Class ID) that points into that table and selects which data to use for that software thread. The table structure is shown below. Its size and various pointers into it are computed immediately following Table 13-1.

The table contains two types of guiding information: performance related guidance and energy efficiency related guidance. It is up to the OS to choose which of the two to refer to when scheduling software threads on logical processors. The table is dynamically updated by the hardware. A notification interrupt is delivered to the OS immediately after the table update.

Table 13-1. EHFI with Thread-Specific Hardware Feedback Structure

Byte Offset	Size (Bytes)	Description
0	8	Time-stamp of when the table was last updated by hardware. This is a time-stamp in crystal clock units. Initialized by OS to 0.
8	1	Class 0 Performance Capability Changed If set to a non-zero value, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + 1	1	Class 0 Energy Efficiency Capability Changed If set to a non-zero value, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + 2	CP - 2	Class 0 Change Indication for Future Capabilities Unavailable if additional capabilities are not enumerated.
8 + CP	1	Class 1 Performance Capability Changed If set to a non-zero value, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + CP + 1	1	Class 1 Energy Efficiency Capability Changed If set to a non-zero value, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + CP + 2	CP - 2	Class 1 Change Indication for Future Capabilities Unavailable if additional capabilities are not enumerated.
8 + 2*CP	1	Class 2 Performance Capability Changed If set to a non-zero value, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + 2*CP + 1	1	Class 2 Energy Efficiency Capability Changed If set to a non-zero value, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + 2*CP + 2	CP - 2	Class 2 Change Indication for Future Capabilities Unavailable if additional capabilities are not enumerated.
8 + 3*CP	1	Class 3 Performance Capability Changed If set to a non-zero value, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.

Table 13-1. EHFI with Thread-Specific Hardware Feedback Structure

Byte Offset	Size (Bytes)	Description
8 + 3*CP + 1	1	Class 3 Energy Efficiency Capability Changed If set to a non-zero value, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + 3*CP + 2	CP - 2	Class 3 Change Indication for Future Capabilities Unavailable if additional capabilities are not enumerated.
8 + 4*CP	CP*(CL-4) + R8	Change Indication for Future Capabilities Size is rounded up by R8 to the nearest whole multiple of 8 bytes.
8 + CP*CL + R8	CL*CP +R8	LP_0 capability values (field size is rounded up by R8 to the nearest whole multiple of 8 bytes).
8 + 2*(CP*CL + R8)	CL*CP +R8	LP_1 capability values (field size is rounded up by R8 to the nearest whole multiple of 8 bytes).
…	…	…
8 + (i+1)*(CP*CL + R8)	CL*CP +R8	LP_i capability values (field size is rounded up by R8 to the nearest whole multiple of 8 bytes).
8 + N*(CP*CL + R8)	CL*CP +R8	LP_{N-1} capability values (field size is rounded up by R8 to the nearest whole multiple of 8 bytes).

N is the number of Logical Processors on the socket.

See "CPUID—CPU Identification" in Chapter 1 for the number of classes (CL) and the number of supported capabilities (CP). Both upper case CL and CP denote total number of classes and capabilities defined for the processor. Lower case cl and cp denote one instance of a class or capability. cl and cp are counted starting zero.

R8 is the number of bytes necessary to round up the Capability Change Indication array to whole multiple of 8 bytes.

Table size: 8 + (N+1)* (CP * CL + R8)

Byte offset of $Capability_{cp}$ of $Class_{cl}$ change indication: 8 + CP * cl + cp

Byte offset of LP_i entry: 8 + (i+1) * (CP * CL + R8)

Byte offset of $capability_{cp}$ of $class_{cl}$ of LP_i: 8 + (i+1) * (CP * CL + R8) + CP * cl + cp

13.1 ENHANCED HARDWARE FEEDBACK INTERFACE INTENDED USAGE MODEL

When the OS Scheduler needs to decide which one of multiple free logical processors to assign to a software thread that is ready to execute, it can choose one of the following options:

1. The free logical processor with the highest performance value of that software thread class, if the system is scheduling for performance.
2. The free logical processor with the highest energy efficiency value of that software thread class, if the system is scheduling for energy efficiency.

When the OS Scheduler needs to decide which of two logical processors (i,j) to assign to which of two software threads whose Class IDs are k1 and k2, it can compute the two performance ratios: $Perf_{ijkx} = Perf_{ikx} / Perf_{jkx}$, or two energy efficiency ratios: $Energy_{ijkx} = Energy_{ikx} / Energy_{jkx}$ between the two logical processors for each of the two classes, depending on whether the OS is scheduling for performance or for energy efficiency.

For example, assume that the system is scheduling for performance and that $Perf_{ijk1} > Perf_{ijk2}$. The OS Scheduler will assign the software thread whose Class ID is k1 to logical processor i, and the one whose Class ID is k2 to logical processor j.

When the two software threads in question belong to the same Class ID, the OS Scheduler can schedule to higher performance logical processors within that class when scheduling for performance and to higher energy efficiency logical processors within that class when scheduling for energy efficiency.

For the HFI, where all software threads effectively belong to the same class (class 0 in the EHFI), the OS Scheduler can use similar logic and schedule to higher performance logical processors when scheduling for performance, and to higher energy efficiency logical processors when scheduling for energy efficiency.

The core ordering of the performance and energy columns may be different between HFI with thread-specific hardware feedback supported classes.

13.2 HARDWARE FEEDBACK INTERFACE POINTER

The physical address of the HFI/EHFI structure is programmed by the OS into a package scope MSR named IA32_HW_FEEDBACK_PTR. The MSR is structured as follows:

- Bit 0 – Valid. When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.
- Bits 11:1 – Reserved.
- Bits MAXPHYADDR-1:12 – ADDR. This is the physical address of the page frame of the first page of this structure.
- Bits 63:MAXPHYADDR[1] – Reserved.

The address of this MSR is 17D0H. This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

CPUID.06H.0H:EDX[11:8] enumerates the size of memory that must be allocated by the OS for this structure.

13.3 HARDWARE FEEDBACK INTERFACE CONFIGURATION

The operating system enables HFI/EHFI using a package scope MSR named IA32_HW_FEEDBACK_CONFIG (address 17D1H). This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

The MSR is structured as follows:

- Bit 0 – Enable. When set to 1, enables HFI.
- Bit 1 – Enable thread-specific hardware feedback (or multi-class support). Both bits 0 and 1 must be set for EHFI with thread-specific hardware feedback to be enabled. The extra class columns in the EHFI table are updated by hardware immediately following setting those two bits, as well as during run time as necessary.
- Bits 63:2 – Reserved.

Before enabling HFI, the OS must set a valid HFI structure using the IA32_HW_FEEDBACK_PTR MSR.

When the OS sets bit 0 only, the hardware populates class 0 capabilities only in the HFI structure. When bit 1 is set after or together with bit 0, the EHFI multi-class structure is populated.

When either the HFI structure or the EHFI structure are ready to use by the OS, the hardware sets IA32_PACKAGE_THERM_STATUS[bit 26]. An interrupt is generated by the hardware if IA32_PACKAGE_THERM_INTERRUPT[bit 25] is set.

When the OS clears bit 1 but leaves bit 0 set, EHFI is disabled, but HFI is kept operational. IA32_PACKAGE_THERM_STATUS[bit 26] is NOT set in this case.

Clearing bit 0 disables both HFI and EHFI, independent of the bit 1 state. Setting bit 1 to '1' while keeping bit 0 at '0' is an invalid combination which is quietly ignored.

When the OS clears bit 0, hardware sets the IA32_PACKAGE_THERM_STATUS[bit 26] to 1 to acknowledge disabling of the interface. The OS should wait for this bit to be set to 1 to reclaim the memory of the EHFI structure, as by setting IA32_PACKAGE_THERM_STATUS[bit 26] hardware guarantees not to write into the EHFI structure anymore.

The OS may clear bit 0 only after receiving an indication from the hardware that the structure initialization is complete via the same IA32_PACKAGE_THERM_STATUS[bit 26], following enabling of HFI/EHFI, thus avoiding a race condition between OS and hardware.

1. MAXPHYADDR is reported in CPUID.80000008H:EAX[7:0].

Bit 1 is valid only if CPUID[6].EAX[bit 23] is set. When setting this bit while support is not enumerated, the hardware generates #GP.

Table 13-2 summarizes the control options described above.

See Section 13.7 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

Table 13-2. IA32_HW_FEEDBACK_CONFIG Control Options

Pre-Bit 1	Pre-Bit 0	Post-Bit 1	Post-Bit 0	Action	IA32_PACKAGE_THERM_STATUS [bit 26] and Interrupt
0	0	0	0	Reset value.	Both Hardware Feedback Interface and Enhanced Hardware Feedback Interface are disabled, no status bit set, no interrupt is generated.
0	0	0	1	Enable HFI structure.	Set the status bit and generate interrupt if enabled.
0	0	1	0	Invalid option; quietly ignored by the hardware.	No action (no update in the table).
0	0	1	1	Enable HFI and EHFI.	Set the status bit and generate interrupt if enabled.
0	1	0	0	Disable HFI support.	Set the status bit and generate interrupt if enabled.
0	1	1	0	Disable HFI and EHFI.	Set the status bit and generate interrupt if enabled.
0	1	1	1	Enable EHFI.	Set the status bit and generate interrupt if enabled.
1	0	0	0	No action; keeps HFI and EHFI disabled.	No action (no update in the table).
1	0	0	1	Enable HFI.	Set the status bit and generate interrupt if enabled.
1	0	1	1	Enable HFI and EHFI.	Set the status bit and generate interrupt if enabled.
1	1	0	0	Disable HFI and EHFI.	Set the status bit and generate interrupt if enabled.
1	1	0	1	Disable EHFI; keep HFI enabled.	No action (no update in the table).
1	1	1	0	Disable HFI and EHFI.	Set the status bit and generate interrupt if enabled.

13.4 HARDWARE FEEDBACK INTERFACE NOTIFICATIONS

The IA32_PACKAGE_THERM_STATUS MSR is extended with a new bit, hardware feedback interface structure change status (bit 26, R/WC0), to indicate that the hardware has updated the HFI/EHFI structure. This is a sticky bit and once set, indicates that the OS should read the structure to determine the change and adjust its scheduling decisions. Once set, the hardware will not generate any further updates to this structure until the OS clears this bit by writing 0.

The OS can enable interrupt-based notifications when the structure is updated by hardware through a new enable bit, hardware feedback interrupt enable (bit 25, R/W), in the IA32_PACKAGE_THERM_INTERRUPT MSR. When this bit is set to 1, it enables the generation of an interrupt when the HFI/EHFI structure is updated by hardware. When

the enable bit transitions from 0 to 1, hardware will generate an initial notification, with the IA32_PACKAGE_THERM_STATUS bit 26 set to 1, to indicate that the OS should read the current HFI/EHFI structure.

13.5 HARDWARE FEEDBACK INTERFACE STRUCTURE DYNAMIC UPDATE

The HFI/EHFI structure can be updated dynamically during run time. Changes to the structure may occur to one or more of its cells. Such changes may occur for one or more logical processors. The hardware sets a non zero value in the "capability change" field of the HFI/EHFI structure as an indication for the OS to read that capability for all logical processors. A thermal interrupt is delivered to indicate to the OS that the structure has just changed. Section 13.4 contains more details on this notification mechanism. The hardware clears all "capability change" fields after the OS resets IA32_PACKAGE_THERM_STATUS[bit 26].

Zeroing a performance or energy efficiency cell hints to the OS that it is beneficial not to schedule software threads of that class on the associated logical processor for performance or energy efficiency reasons, respectively. If SMT is supported, it may be the case that the hardware zeroes one of the core's logical processors only. Zeroing the performance and energy efficiency cells of all classes for a logical processor implies that the hardware provides a hint to the OS to completely avoid scheduling work on that logical processor.

A few example reasons for runtime changes in the HGS/HGS+ Table:

- Over clocking run time update that changes the capability values.
- Change in run time physical constraints.
- Run time performance or energy efficiency optimization.
- Change in core frequency, voltage, or power budget.

13.6 LOGICAL PROCESSOR SCOPE ENHANCED HARDWARE FEEDBACK INTERFACE CONFIGURATION

The operating system enables EHFI at the logical processor scope using a logical processor scope MSR named IA32_HW_FEEDBACK_THREAD_CONFIG (address 17D4H).

The MSR is read/write and is structured as follows:

- Bit 0 – Enables EHFI. When set to 1, logical processor scope enhanced hardware feedback is enabled. Default is 0 (disabled).
- Bits 63:1 – Reserved.

Bit 0 of the logical processor scope configuration MSR can be cleared or set regardless of the state of the HFI/EHFI package configuration MSR state. Even when bit 0 of all logical processor configuration MSRs is clear, the processor can still update the EHFI structure if it is still enabled in the IA32_HW_FEEDBACK_CONFIG package scope MSR. When the operating system clears IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0], hardware clears the history accumulated on that logical processor which otherwise drives assigning the Class ID to the software thread that executes on that logical processor. As long as IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set, the Class ID is available for the operating system to read, independent of the state of the package scope IA32_HW_FEEDBACK_CONFIG[1:0] bits.

See Section 13.7 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

13.7 IMPLICIT RESET OF PACKAGE AND LOGICAL PROCESSOR SCOPE CONFIGURATION MSRS

HFI/EHFI enable bits are reset by hardware in the following scenarios:

1. When GetSec[SENTER] is issued:

   a. The processor implicitly resets the HFI/EHFI enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on all sockets (packages) in the system.
   b. The processor implicitly resets the EHFI enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors in the system across all sockets.
   c. The processor implicitly clears the HFI/EHFI table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR across all sockets.
2. When GetSec[ENTERACCS] is issued:
   a. The processor implicitly resets the HFI/EHFI enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on the socket where the GetSec[ENTERACCS] instruction was issued.
   b. The processor implicitly resets the EHFI enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors on the socket where the GetSec[ENTERACCS] instruction was issued.
   c. The processor implicitly clears the HFI/EHFI table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR on the socket where the GetSec[ENTERACCS] instruction was issued.
3. When INIT or Wait for SIPI signals are processed by a logical processor:
   a. The processor implicitly resets the EHFI enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on that logical processor, whether the signal was in the context of GetSec[ENTERACCS] or not.

If the OS requires HFI/EHFI to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/EHFI.

13.8 LOGICAL PROCESSOR SCOPE ENHANCED HARDWARE FEEDBACK INTERFACE RUN TIME CHARACTERISTICS

The processor provides the operating system with run time feedback about the execution characteristics of the software thread executing on logical processors whose IA32_HW_FEEDBACK_CONFIG[bit 0] is set.

The run time feedback is communicated via a read-only MSR named IA32_THREAD_FEEDBACK_CHAR. This is a logical processor scope MSR whose address is 17D2H. This MSR is structured as follows:

- Bits 7:0 – Application Class ID, pointing into the EHFI structure described in Table 13-1.
- Bits 62:8 – Reserved.
- Bit 63 – Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.

This MSR is valid only if CPUID.06H:EAX[bit 23] is set.

The valid bit is cleared by the hardware in the following cases:

- The hardware does not have enough information to provide the operating system with a reliable Class ID.
- The operating system cleared the logical processor's IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] bit.
- The HRESET instruction is executed while configured to reset the EHFI history.

13.9 ENHANCED HARDWARE FEEDBACK INTERFACE ENUMERATION

See "CPUID—CPU Identification" in Chapter 1 for enumeration of EHFI.

13.10 LOGICAL PROCESSOR SCOPE HISTORY

The operating system can reset the EHFI related history accumulated on the current logical processor it is executing on by issuing the HRESET instruction. See "CPUID—CPU Identification" in Chapter 1 for enumeration of the HRESET instruction and Chapter 2, "HRESET — History Reset" for additional instruction details.

13.10.1 Enabling Enhanced Hardware Feedback Interface History Reset

The IA32_HRESET_ENABLE MSR is a read/write MSR and is structured as follows:

- Bit 0 – Enable reset of the EHFI history.
- Bits 31:1 – Reserved for other capabilities that can be reset by the HRESET instruction.
- Bits 63:32 – Reserved.

The operating system should set IA32_HRESET_ENABLE[bit 0] to enable EHFI history reset via the HRESET instruction.

13.10.2 Implicit Enhanced Hardware Feedback Interface History Reset

The EHFI history is implicitly reset in the following scenarios:

1. When the processor enters or exits SMM mode and IA32_DEBUGCTL MSR.FREEZE_WHILE_SMM (bit 14) is set, the EHFI history is implicitly reset by the processor.
2. When GetSec[SENTER] is issued, the processor resets the EHFI history on all logical processors in the system, including logical processors on other sockets (other than the one GetSec(SENTER) is executed).
3. When GetSec[ENTERACCS] is issued, the processor resets the EHFI history on the logical processor it is executed on.
4. When INIT or Wait for SIPI signals are processed by a logical processor, the EHFI history is reset whether the signal was a result of GetSec[ENTERACCS] or not.

If the operating system requires HFI/EHFI to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/EHFI.

13.11 IA32_DEBUGCTL MSR UPDATE

FREEZE_WHILE_SMM (bit 14): If this bit is set, upon the delivery of an SMI, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler. If the thread-specific EHFI support was enabled before transferring control to the SMI handler, then the processor will also reset EHFI history (see Section 13.10 for more details about EHFI enable, reset and history reset operations).

Subsequently, the enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1 and the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its service. If the thread-specific EHFI support is enabled when RSM is executed, then the processor resets EHFI history.

Note that system software must check if the processor supports the IA32_DEBUGCTL.FREEZE_WHILE_SMM control bit. IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1.

CHAPTER 14
LINEAR ADDRESS MASKING (LAM)

This chapter describes a new feature called **linear-address masking** (**LAM**). LAM modifies the checking that is applied to 64-bit linear addresses, allowing software to use of the untranslated address bits for metadata.

In 64-bit mode, linear address have 64 bits and are translated either with 4-level paging, which translates the low 48 bits of each linear address, or with 5-level paging, which translates 57 bits. The upper linear-address bits are reserved through the concept of **canonicality**. A linear address is 48-bit canonical if bits 63:47 of the address are identical; it is 57-bit canonical if bits 63:56 are identical. (Clearly, any linear address that is 48-bit canonical is also 57-bit canonical.) When 4-level paging is active, the processor requires all linear addresses used to access memory to be 48-bit canonical; similarly, 5-level paging ensures that all linear addresses are 57-bit canonical.

Software usages that associate metadata with a pointer might benefit from being able to place metadata in the upper (untranslated) bits of the pointer itself. However, the canonicality enforcement mentioned earlier implies that software would have to mask the metadata bits in a pointer (making it canonical) before using it as a linear address to access memory. LAM allows software to use pointers with metadata without having to mask the metadata bits. With LAM enabled, the processor masks the metadata bits in a pointer before using it as a linear address to access memory.

LAM is supported only in 64-bit mode and applies only addresses used for data accesses. LAM doe not apply to addresses used for instruction fetches or to those that specify the targets of jump and call instructions.

14.1 ENUMERATION, ENABLING, AND CONFIGURATION

LAM support by the processor is enumerated by the CPUID feature flag CPUID.(EAX=07H, ECX=01H):EAX.LAM[bit 26]. Enabling and configuration of LAM is controlled by the following new bits in control registers: CR3[61] (**LAM_U48**), CR3[62] (**LAM_U57**), and CR4[28] (**LAM_SUP**). The use of these control bit is explained below.

LAM supports configurations that differ regarding which pointer bits are masked and can be used for metadata. With **LAM48**, pointer bits in positions 62:48 are masked (resulting in a **LAM width** of 15); with **LAM57**, pointer bits in positions 62:57 are masked (a LAM width of 6). The LAM width may be configured differently for user and supervisor pointers. LAM identifies pointer as a user pointer if bit 63 of the pointer is 0 and as a supervisor pointer if bit 63 of the pointer is 1.

CR3.LAM_U48 and CR3.LAM_U57 enable and configure LAM for user pointers:

- If CR3.LAM_U48 = CR3.LAM_U57 = 0, LAM is not enabled for user pointers.
- If CR3.LAM_U48 = 1 and CR3.LAM_U57 = 0, LAM48 is enabled for user pointers (a LAM width of 15).
- If CR3.LAM_U57 = 1, LAM57 applies to user pointers (a LAM width of 6; CR3.LAM_U48 is ignored).

CR4.LAM_SUP enables and configures LAM for supervisor pointers:

- If CR3.LAM_SUP = 0, LAM is not enabled for supervisor pointers.
- If CR3.LAM_SUP = 1, LAM is enabled for supervisor pointers with a width determined by the paging mode:
  - If 4-level paging is enabled, LAM48 is enabled for supervisor pointers (a LAM width of 15).
  - If 5-level paging is enabled, LAM47 is enabled for supervisor pointers (a LAM width of 6).

Note that the LAM identification of a pointer as user or supervisor is based solely on the value of pointer bit 63 and does not, for the purposes of LAM, depend on the CPL.

14.2 TREATMENT OF DATA ACCESSES WITH LAM ACTIVE FOR USER POINTERS

Recall that, without LAM, canonicality checks are defined so that 4-level paging requires bits 63:47 of each pointer to be identical, while 5-level paging requires bits 63:56 to be identical. LAM allows some of these bits to be used as metadata by modifying canonicality checking.

When LAM48 is enabled for user pointers (see Section 14.1), the processor allows bits 62:48 of a user pointer to be used as metadata. Regardless of the paging mode, the processor performs a modified canonicality check that enforces that bit 47 of the pointer matches bit 63. As illustrated in Figure 14-1, bits 62:48 are not checked and are thus available for software metadata. After this modified canonicality check is performed, bits 62:48 are masked by sign-extending the value of bit 47 (0), and the resulting (48-bit canonical) address is then passed on for translation by paging.

(Note also that, without LAM, canonicality checking with 5-level paging does not apply to bit 47 of a user pointer; when LAM48 is enabled for user pointers, bit 47 of a user pointer must be 0. Note also that linear-address bits 56:47 are translated by 5-level paging. With LAM_U48, these bits are always 0 in any linear address derived from a user pointer: bits 56:48 of the pointer contained metadata, while bit 47 is required to be 0.)

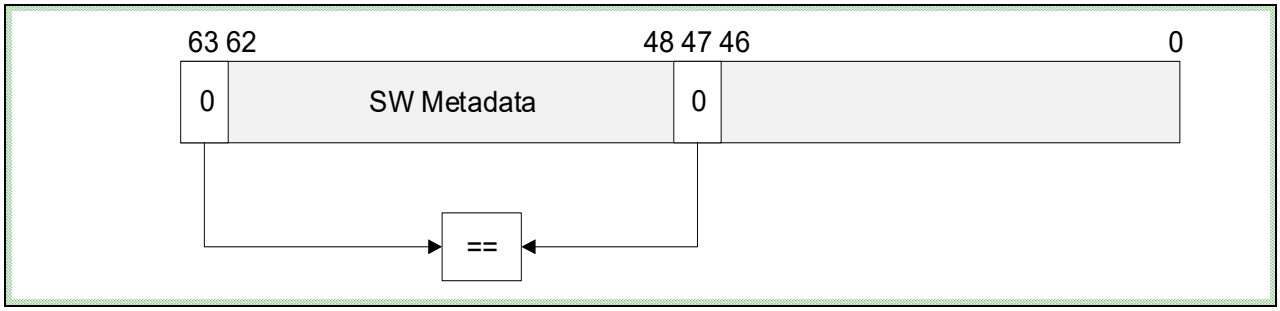


Figure 14-1. Canonicality Check When LAM48 is Enabled for User Pointers

When LAM57 is enabled for user pointers, the processor allows bits 62:57 of a user pointer to be used as metadata. With 5-level paging, the processor performs a modified canonicality check that enforces only that bit 56 of the pointer matches bit 63. As illustrated in Figure 14-2, bits 62:57 are not checked and are thus available for software metadata. After this modified canonicality check is performed, bits 62:57 are masked by sign-extending the value of bit 56 (0), and the resulting (57-bit canonical) address is then passed on for translation by 5-level paging.

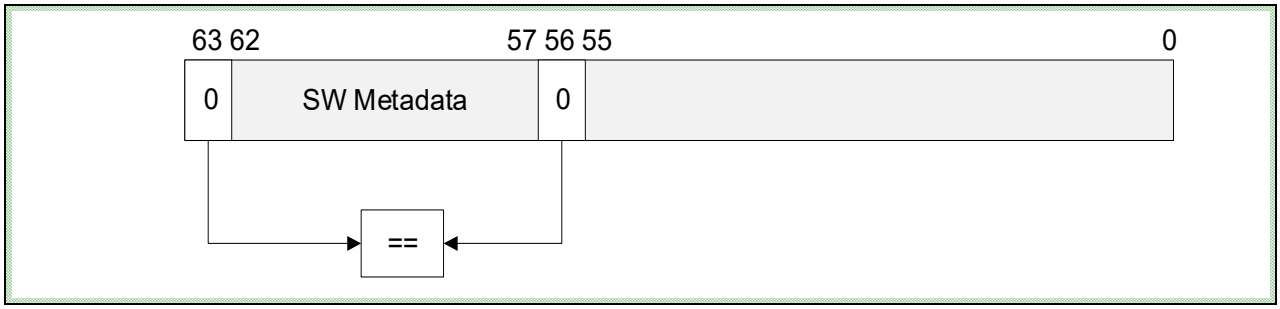


Figure 14-2. Canonicality Check When LAM57 is Enabled for User Pointers with 5-Level Paging

When LAM57 is enabled for user pointers with 4-level paging, the processor performs a modified canonicality check that enforces only that bits 56:47 of a user pointer match bit 63. As illustrated in Figure 14-3, bits 62:57 are not checked and are thus available for software metadata. After this modified canonicality check is performed, bits 62:57 are masked by sign-extending the value of bit 56 (0), and the resulting (48-bit canonical) address is then passed on for translation by 4-level paging.

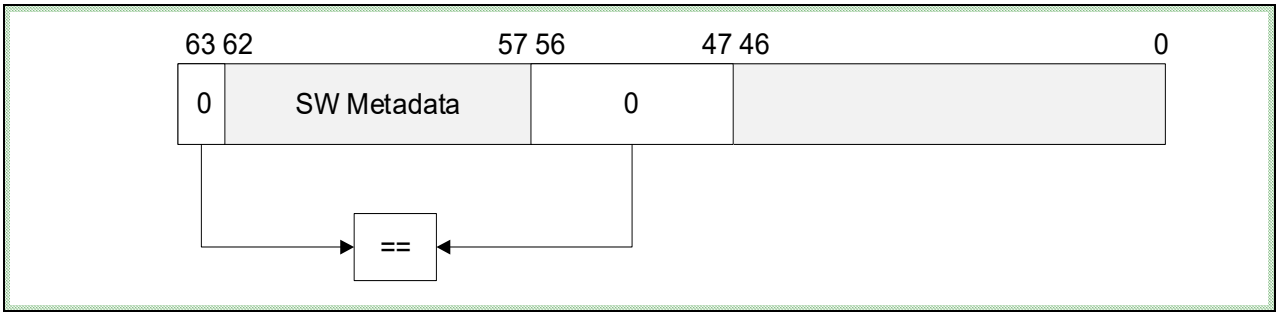


Figure 14-3. Canonicality Check When LAM57 is Enabled for User Pointers with 4-Level Paging

14.3 TREATMENT OF DATA ACCESSES WITH LAM ACTIVE FOR SUPERVISOR POINTERS

As with user pointers (Section 14.2), LAM can be configured to modify canonicality checking to allow use of metadata in supervisor pointers. For supervisor pointers, the number of metadata bits (the LAM width) available depends on the paging mode active: with 5-level paging, enabling LAM for supervisor pointers results in LAM57; with 4-level paging, it results in LAM48 (see Section 14.1).

When LAM57 is enabled for supervisor pointers (5-level paging), the processor performs a modified canonicality check that enforces only that bit 56 of a supervisor pointer matches bit 63. As illustrated in Figure 14-4, bits 62:57 are not checked and are thus available for software metadata. After this modified canonicality check is performed, bits 62:57 are masked by sign-extending the value of bit 56 (1), and the resulting (57-bit canonical) address is then passed on for translation by 5-level paging.

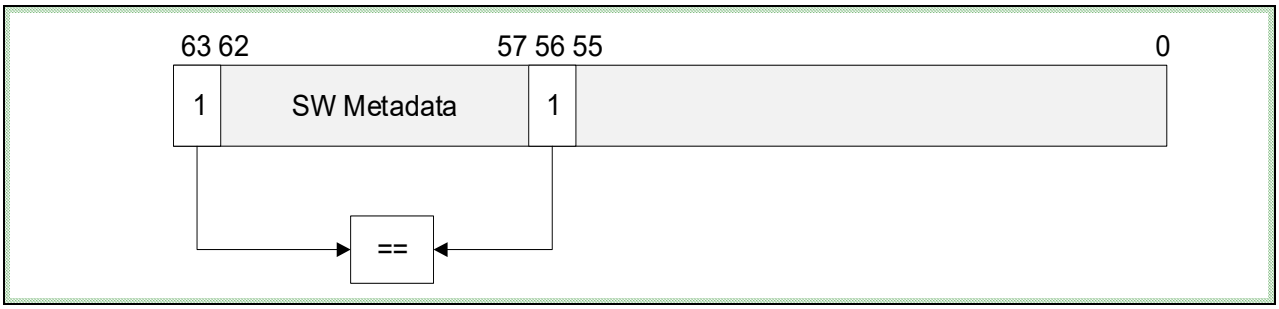


Figure 14-4. Canonicality Check When LAM57 is Enabled for Supervisor Pointers with 5-Level Paging

When LAM48 is enabled for supervisor pointers (4-level paging), the processor performs a modified canonicality check that enforces only that bit 47 of a supervisor pointer matches bit 63. As illustrated in Figure 14-5, bits 62:48 are not checked and are thus available for software metadata. After this modified canonicality check is performed, bits 62:48 are masked by sign-extending the value of bit 47 (1), and the resulting (48-bit canonical) address is then passed on for translation by 4-level paging.

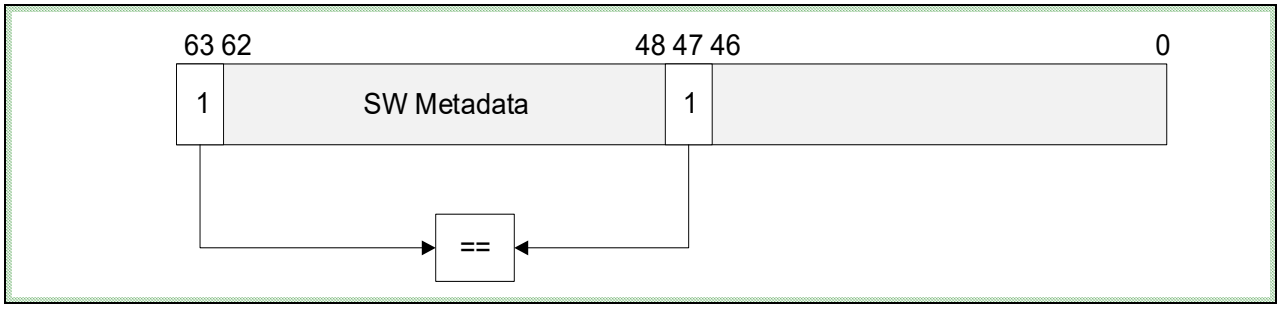


Figure 14-5. Canonicality Check When LAM48 is Enabled for Supervisor Pointers with 4-Level Paging

14.4 CANONICALITY CHECKING FOR DATA ADDRESSES WRITTEN TO CONTROL REGISTERS AND MSRS

Processors that support LAM continue to require the addresses written to control registers or MSRs be 57-bit canonical if the processor supports 5-level paging or 48-bit canonical if it supports only 4-level paging; LAM masking is not performed on these writes. When the contents of such registers are used as pointers to access memory, the processor performs canonicality checking and masking based on paging mode and LAM mode configuration active at the time of access.

14.5 PAGING INTERACTIONS

As explained in Section 14.2 and Section 14.3, LAM masks certain bits in a pointer by sign-extension, resulting in a linear address to be translated by paging.

In most cases, the address bits in the masked positions are not used by address translation. However, if 5-level paging and LAM_U48 are both active, bit 47 of a user pointer must be zero and is extended over bits 62:48 to form a linear address — even though bits 56:48 are used by 5-level paging. This implies that, when LAM_U48 is active, bits 56:47 are 0 in any linear address translated for a user pointer.

Page faults report the faulting linear address in CR2. Because LAM masking (by sign-extension) applies before paging, the faulting linear address recorded in CR2 does not contain the masked metadata.

The INVLPG instruction is used to invalidate any translation lookaside buffer (TLB) entries for a memory address specified with the source operand. LAM does not apply to the specified memory address. Thus, in 64-bit mode, if the memory address specified is in non-canonical form then the INVLPG is the same as a NOP.

The INVPCID instruction invalidates mappings in the TLB and paging structure caches based on the processor context identifier (PCID). The INVPCID descriptor provides the memory address to invalidate when the descriptor is of type 0 (individual-address invalidation). LAM does not apply to the specified memory address, and in 64-bit mode if this memory address is in non-canonical form then the processor generates a #GP(0) exception.

14.6 VMX INTERACTIONS

14.6.1 Guest Linear Address

Certain VM exits save in a VMCS field the guest linear address pertaining to the VM exit. Because such a linear address results from masking the original pointer, the processor does not report the masked metadata in the VMCS. The guest linear address saved is always the result of the sign-extension described in Section 14.2 and Section 14.3.

14.6.2 VM-Entry Checking of Values of CR3 and CR4

VM entry checks the values of the CR3 and CR4 fields in the guest-area and host-state area of the VMCS. In particular, the bits in these fields that correspond to bits reserved in the corresponding register are checked and must be 0.

On processors that enumerate support for LAM (Section 14.1), VM entry allows bits 62:61 to be set in either CR3 field and allows bit 28 to be set in either CR4 field.

14.6.3 CR3-Target Values

If the "CR3-load exiting" VM-execution control is 1, execution of MOV to CR3 in VMX non-root operation causes a VM exit unless the value of the instruction's source operand is equal to one of the CR3-target values specified in the VMCS.

Processor support for LAM does not change this behavior. The comparison of the instruction source operand to each of the CR3-target values considers all 64 bits, including the two new bits that determine LAM enabling for user pointers (see Section 14.1).

14.6.4 Hypervisor-Managed Linear Address Translation (HLAT)

Hypervisor-managed linear-address translation (HLAT) is enabled when the "enable HLAT" tertiary processor-based VM-execution control is 1.

When HLAT is enabled for a guest, the processor translates a linear address using HLAT paging structures (instead of guest paging structures) if the address matches the Protected Linear Range (PLR). When LAM is active, it is the linear address (derived from a pointer by masking) that is checked for a PLR match.

The hierarchy of HLAT paging structures is located using a guest-physical address in the VMCS (instead of the guest-physical address in CR3). Nevertheless, LAM enabling and configuration for user pointers is based on the value of CR3[62:61] (see Section 14.1) even when the guest-physical address in CR3 is not used for translating the linear addresses derived from user pointers.

14.7 DEBUG AND TRACING INTERACTIONS

14.7.1 Debug Registers

Debug registers DR0-DR3 can be programmed with linear addresses that are matched against memory accesses for data breakpoints or instruction breakpoints. When LAM is active, it is the linear address (derived from a pointer by masking) that is checked for matching the contents of the debug registers.

14.7.2 Intel® Processor Trace

Intel Processor Trace supports a CR3-filtering mechanism by which generation of packets containing architectural states can be enabled or disabled based on the value of CR3 matching the contents of the IA32_RTIT_CR3_MATCH MSR. On processors that support LAM, bits 62:61 of the CR3 (see Section 14.1) must also match bits 62:61 of this MSR to enable tracing.

14.8 INTEL® SGX INTERACTIONS

Memory operands of ENCLS, ENCLU, and ENCLV that are data pointers follow the LAM architecture and mask suitably. Code pointers continue to not mask metadata bits. ECREATE does not mask BASEADDR specified in SECS, and the unmasked BASEADDR must be canonical.

Two new SECS attribute bits are defined for LAM support in enclave mode:

- ATTRIBUTE.LAM_U48 (bit 9) - Activate LAM for user data pointers and use of bits 62:48 as masked metadata in enclave mode. This bit can be set if CPUID.(EAX=12H, ECX=01H):EAX[9] is 1.
- ATTRIBUTE.LAM_U57 (bit 8) - Activate LAM for user data pointers and use of bits 62:57 as masked metadata in enclave mode. This bit can be set if CPUID.(EAX=12H, ECX=01H):EAX[8] is 1.

ECREATE causes #GP(0) if ATTRIBUTE.LAM_U48 bit is 1 and CPUID.(EAX=12H, ECX=01H):EAX[9] is 0, or if ATTRIBUTE.LAM_U57 bit is 1 and CPUID.(EAX=12H, ECX=01H):EAX[8] is 0.

If SECS.ATTRIBUTES.LAM_U57 is 1, then LAM57 is enabled for user pointers during execution of an enclave controlled by the SECS (regardless of the value of CR3). If SECS.ATTRIBUTES.LAM_U57 is 0 and SECS.ATTRIBUTES.LAM_U48 is 1, then LAM48 is enabled for user pointers during execution of an enclave controlled by the SECS (regardless of the value of CR3).

When in enclave mode, supervisor data pointers are not subject to any masking.

The following ENCLU leaf functions check for linear addresses to be within the ELRANGE. When LAM is active, this check is performed on the linear addresses that result from masking metadata bits in user pointers used by the leaf functions.

- EACCEPT
- EACCEPTCOPY
- EGETKEY
- EMODPE
- EREPORT

The following linear address fields in the Intel SGX data structures hold linear addresses that are either loaded into the EPCM or are written out from the EPCM and do not contain any metadata.

- SECS.BASEADDR
- PAGEINFO.LINADDR

14.9 SYSTEM MANAGEMENT MODE (SMM) INTERACTIONS

On processors that enumerate support for LAM (Section 14.1), RSM allows restoring CR3 with a value that sets either or both bit 62 and bit 61 and restoring a value of CR4 with a value that sets bit 28.

CHAPTER 15
ERROR CODES FOR PROCESSORS BASED ON SAPPHIRE RAPIDS MICROARCHITECTURE

15.1 INTEGRATED MEMORY CONTROLLER MACHINE CHECK ERRORS

MC error codes associated with integrated memory controllers for future processors based on Sapphire Rapids microarchitecture are reported in the MSRs IA32_MC13_STATUS — IA32_MC20_STATUS.

The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*).

Table 15-1. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13-20)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	0001H - Address parity error. 0002H - Data parity error. 0003H - Data ECC error. 0004H - Data byte enable parity error. 0007H - Transaction ID parity error. 0008H - Corrected patrol scrub error. 0010H - Uncorrected patrol scrub error. 0020H - Corrected spare error. 0040H - Uncorrected spare error. 0080H - Corrected read error. 00A0H - Uncorrected read error. 00C0H - Uncorrected metadata. 0100H - WDB read parity error. 0106H - DDR_T_DPPP data BE error. 0107H - DDR_T_DPPP data error. 0108H - DDR link failure. 0111H - PCLS CAM error. 0112H - PCLS data error. 0200H - DDR4 command / address parity error. 0220H - HBM command / address parity error. 0221H - HBM data parity error. 0400H - RPQ0 parity (primary) error. 0800H - DDR-T bad request. 0801H - DDR Data response to an invalid entry. 0802H - DDR data response to an entry not expecting data. 0803H - DDR4 completion to an invalid entry. 0804H - DDR-T completion to an invalid entry. 0805H - DDR data/completion FIFO overflow. 0806H - DDR-T ERID correctable parity error.

Type	Bit No.	Bit Function	Bit Description
			0807H - DDR-T ERID uncorrectable error. 0808H - DDR-T interrupt received while outstanding interrupt was not ACKed. 0809H - ERID FIFO overflow. 080AH - DDR-T error on FNV write credits. 080BH - DDR-T error on FNV read credits. 080CH -DDR-T scheduler error. 080DH - DDR-T FNV error event. 080EH - DDR-T FNV thermal event. 080FH - CMI packet while idle. 0810H - DDR_T_RPQ_REQ_PARITY_ERR. 0811H - DDR_T_WPQ_REQ_PARITY_ERR. 0812H - 2LM_NMFILLWR_CAM_ERR. 0813H - CMI_CREDIT_OVERSUB_ERR. 0814H - CMI_CREDIT_TOTAL_ERR. 0815H - CMI_CREDIT_RSVD_POOL_ERR. 0816H - DDR_T_RD_ERROR. 0817H - WDB_FIFO_ERR. 0818H - CMI_REQ_FIFO_OVERFLOW. 0819H - CMI_REQ_FIFO_UNDERFLOW. 081AH - CMI_RSP_FIFO_OVERFLOW. 081BH - CMI_RSP_FIFO_UNDERFLOW. 081CH - CMI_MISC_MC_CRDT_ERRORS. 081DH - CMI_MISC_MC_ARB_ERRORS. 081EH - DDR_T_WR_CMPL_FIFO_OVERFLOW. 081FH - DDR_T_WR_CMPL_FIFO_UNDERFLOW. 0820H - CMI_RD_CPL_FIFO_OVERFLOW. 0821H - CMI_RD_CPL_FIFO_UNDERFLOW. 0822H - TME_KEY_PAR_ERR. 0823H - TME_CMI_MISC_ERR. 0824H - TME_CMI_OVFL_ERR. 0825H - TME_CMI_UFL_ERR. 0826H - TME_TEM_SECURE_ERR. 0827H - TME_UFILL_PAR_ERR. 0829H - INTERNAL_ERR. 082AH - TME_INTEGRITY_ERR. 082BH - TME_TDX_ERR 082CH - TME_UFILL_TEM_SECURE_ERR. 082DH - TME_KEY_POISON_ERR. 082EH - TME_SECURITY_ENGINE_ERR. 1008H - CORR_PATSCRUB_MIRR2ND_ERR. 1010H - UC_PATSCRUB_MIRR2ND_ERR. 1020H - COR_SPARE_MIRR2ND_ERR. 1040H - UC_SPARE_MIRR2ND_ERR. 1080H - HA_RD_MIRR2ND_ERR. 10A0H - HA_UNCORR_RD_MIRR2ND_ERR.

Type	Bit No.	Bit Function	Bit Description
	37:32	Other info	Other Info.
	56:38		See Chapter 15, "Machine-Check Architecture" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to *Chapter 15, "Machine-Check Architecture,"* in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B* for more information.

INDEX

B

Brand information 1-35
    processor brand index 1-38
    processor brand string 1-36

C

Cache and TLB information 1-30
Cache Inclusiveness 1-5
CLFLUSH instruction
    CPUID flag 1-29
CMOVcc flag 1-29
CMOVcc instructions
    CPUID flag 1-29
CMPXCHG16B instruction
    CPUID bit 1-27
CMPXCHG8B instruction
    CPUID flag 1-29
CPUID instruction 1-3, 1-29
    36-bit page size extension 1-29
    APIC on-chip 1-29
    basic CPUID information 1-4
    cache and TLB characteristics 1-4, 1-30
    CLFLUSH flag 1-29
    CLFLUSH instruction cache line size 1-26
    CMPXCHG16B flag 1-27
    CMPXCHG8B flag 1-29
    CPL qualified debug store 1-27
    debug extensions, CR4.DE 1-28
    debug store supported 1-29
    deterministic cache parameters leaf 1-4, 1-7, 1-9, 1-10, 1-11, 1-12, 1-13, 1-14, 1-15, 1-16, 1-21
    extended function information 1-22
    feature information 1-28
    FPU on-chip 1-28
    FSAVE flag 1-30
    FXRSTOR flag 1-30
    IA-32e mode available 1-23
    input limits for EAX 1-24
    L1 Context ID 1-27
    local APIC physical ID 1-26
    machine check architecture 1-29
    machine check exception 1-29
    memory type range registers 1-29
    MONITOR feature information 1-33
    MONITOR/MWAIT flag 1-27
    MONITOR/MWAIT leaf 1-5, 1-6, 1-7, 1-10, 1-17, 1-21
    MWAIT feature information 1-33
    page attribute table 1-29
    page size extension 1-29
    performance monitoring features 1-33
    physical address bits 1-24
    physical address extension 1-29
    power management 1-33, 1-34, 1-35
    processor brand index 1-26, 1-35
    processor brand string 1-23, 1-35
    processor serial number 1-29
    processor type field 1-25
    RDMSR flag 1-29
    returned in EBX 1-25
    returned in ECX & EDX 1-26
    self snoop 1-30
    SpeedStep technology 1-27
    SS2 extensions flag 1-30

SSE extensions flag 1-30
SSE3 extensions flag 1-27
SSSE3 extensions flag 1-27
SYSENTER flag 1-29
SYSEXIT flag 1-29
thermal management 1-33, 1-34, 1-35
thermal monitor 1-27, 1-30
time stamp counter 1-29
using CPUID 1-3
vendor ID string 1-24
version information 1-4, 1-33
virtual 8086 Mode flag 1-28
virtual address bits 1-24
WRMSR flag 1-29

F

Feature information, processor 1-3
FXRSTOR instruction
CPUID flag 1-30
FXSAVE instruction
CPUID flag 1-30

I

IA-32e mode
CPUID flag 1-23
Instruction set
grouped by processor 1-2

L

L1 Context ID 1-27

M

Machine check architecture
CPUID flag 1-29
description 1-29
MMX instructions
CPUID flag for technology 1-30
Model & family information 1-33
MONITOR instruction
CPUID flag 1-27
feature data 1-33
MWAIT instruction
CPUID flag 1-27
feature data 1-33

P

Pending break enable 1-30
Performance-monitoring counters
CPUID inquiry for 1-33

R

RDMSR instruction
CPUID flag 1-29

S

Self Snoop 1-30
SpeedStep technology 1-27
SSE extensions
CPUID flag 1-30
SSE2 extensions
CPUID flag 1-30
SSE3

CPUID flag 1-27
SSE3 extensions
CPUID flag 1-27
SSSE3 extensions
CPUID flag 1-27
Stepping information 1-33
SYSENTER instruction
CPUID flag 1-29
SYSEXIT instruction
CPUID flag 1-29

T

Thermal Monitor
CPUID flag 1-30
Thermal Monitor 2 1-27
CPUID flag 1-27
Time Stamp Counter 1-29

V

Version information, processor 1-3

W

WBINVD/INVD bit 1-5
WRMSR instruction
CPUID flag 1-29

X

XRSTOR 1-34
XSAVE 1-27, 1-34