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

Volume 3D: System Programming Guide, Part 4

NOTE: The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of ten volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference, A-L*, Order Number 253666; *Instruction Set Reference, M-U*, Order Number 253667; *Instruction Set Reference, V*, Order Number 326018; *Instruction Set Reference*, W-Z, Order Number 334569; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669; *System Programming Guide, Part 3*, Order Number 326019; *System Programming Guide, Part 4*, Order Number 332831; *Model-Specific Registers,* Order Number 335592. Refer to all ten volumes when evaluating your design needs.

Order Number: 332831-080US
June 2023

Notices & Disclaimers

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.

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.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/0BSD. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

CHAPTER 34
INTRODUCTION TO INTEL® SOFTWARE GUARD EXTENSIONS

34.1 OVERVIEW

Intel® Software Guard Extensions (Intel® SGX) is a set of instructions and mechanisms for memory accesses added to Intel® Architecture processors. Intel SGX can encompass two collections of instruction extensions, referred to as SGX1 and SGX2, see Table 34-1 and Table 34-2. The SGX1 extensions allow an application to instantiate a protected container, referred to as an enclave. The enclave is a trusted area of memory, where critical aspects of the application functionality have hardware-enhanced confidentiality and integrity protections. New access controls to restrict access to software not resident in the enclave are also introduced. The SGX2 extensions allow additional flexibility in runtime management of enclave resources and thread execution within an enclave.

Chapter 35 covers main concepts, objects and data structure formats that interact within the Intel SGX architecture. Chapter 36 covers operational aspects ranging from preparing an enclave, transferring control to enclave code, and programming considerations for the enclave code and system software providing support for enclave execution. Chapter 37 describes the behavior of Asynchronous Enclave Exit (AEX) caused by events while executing enclave code. Chapter 38 covers the syntax and operational details of the instruction and associated leaf functions available in Intel SGX. Chapter 39 describes interaction of various aspects of IA32 and Intel® 64 architectures with Intel SGX. Chapter 40 covers Intel SGX support for application debug, profiling, and performance monitoring.

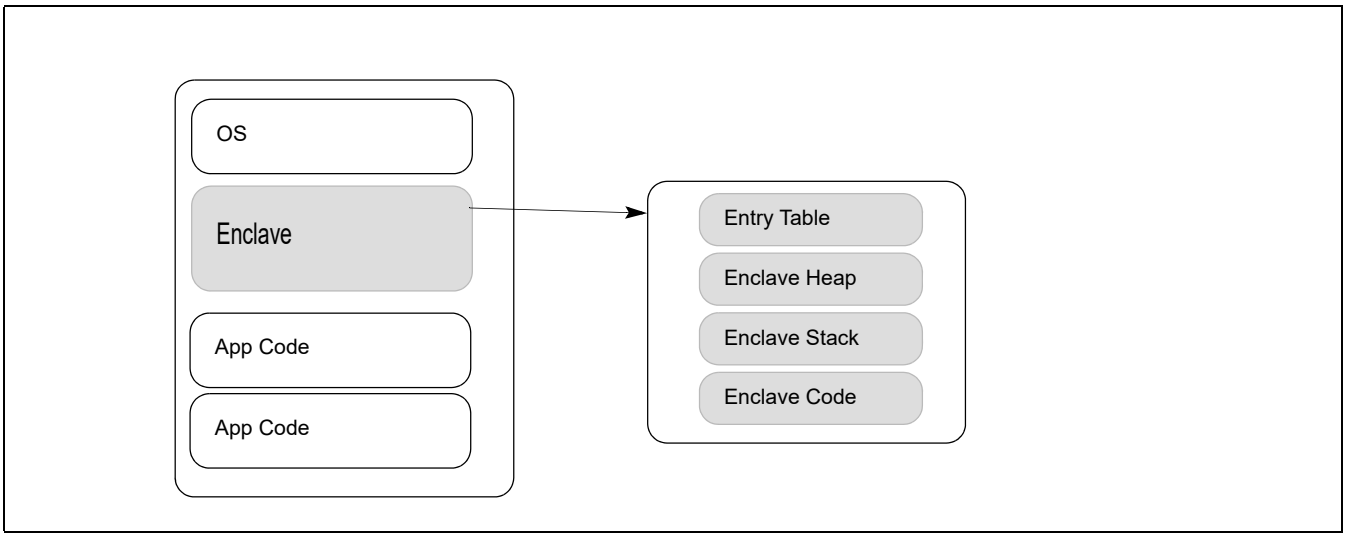

Figure 34-1. An Enclave Within the Application's Virtual Address Space

34.2 ENCLAVE INTERACTION AND PROTECTION

Intel SGX allows the protected portion of an application to be distributed in the clear. Before the enclave is built, the enclave code and data are free for inspection and analysis. The protected portion is loaded into an enclave where its code and data is measured. Once the application's protected portion of the code and data are loaded into an enclave, memory access controls are in place to restrict access by external software. An enclave can prove its identity to a remote party and provide the necessary building-blocks for secure provisioning of keys and credentials. The application can also request an enclave-specific and platform-specific key that it can use to protect keys and data that it wishes to store outside the enclave.[1]

1. For additional information, see white papers on Intel SGX at https://www.intel.com/content/www/us/en/developer/tools/isa-extensions/overview.html.

Intel SGX introduces two significant capabilities to the Intel Architecture. First is the change in enclave memory access semantics. The second is protection of the address mappings of the application.

34.3 ENCLAVE LIFE CYCLE

Enclave memory management is divided into two parts: address space allocation and memory commitment. Address space allocation is the specification of the range of linear addresses that the enclave may use. This range is called the ELRANGE. No actual resources are committed to this region. Memory commitment is the assignment of actual memory resources (as pages) within the allocated address space. This two-phase technique allows flexibility for enclaves to control their memory usage and to adjust dynamically without overusing memory resources when enclave needs are low. Commitment adds physical pages to the enclave. An operating system may support separate allocate and commit operations.

During enclave creation, code and data for an enclave are loaded from a clear-text source, i.e., from non-enclave memory.

Untrusted application code starts using an initialized enclave typically by using the EENTER leaf function provided by Intel SGX to transfer control to the enclave code residing in the protected Enclave Page Cache (EPC). The enclave code returns to the caller via the EEXIT leaf function. Upon enclave entry, control is transferred by hardware to software inside the enclave. The software inside the enclave switches the stack pointer to one inside the enclave. When returning back from the enclave, the software swaps back the stack pointer then executes the EEXIT leaf function.

On processors that support the SGX2 extensions, an enclave writer may add memory to an enclave using the SGX2 instruction set, after the enclave is built and running. These instructions allow adding additional memory resources to the enclave for use in such areas as the heap. In addition, SGX2 instructions allow the enclave to add new threads to the enclave. The SGX2 features provide additional capabilities to the software model without changing the security properties of the Intel SGX architecture.

Calling an external procedure from an enclave could be done using the EEXIT leaf function. Software would use EEXIT and a software convention between the trusted section and the untrusted section.

An active enclave consumes resources from the Enclave Page Cache (EPC, see Section 34.5). Intel SGX provides the EREMOVE instruction that an EPC manager can use to reclaim EPC pages committed to an enclave. The EPC manager uses EREMOVE on every enclave page when the enclave is torn down. After successful execution of EREMOVE the EPC page is available for allocation to another enclave.

34.4 DATA STRUCTURES AND ENCLAVE OPERATION

There are 2 main data structures associated with operating an enclave, the SGX Enclave Control Structure (SECS, see Section 35.7) and the Thread Control Structure (TCS, see Section 35.8).

There is one SECS for each enclave. The SECS contains meta-data about the enclave which is used by the hardware and cannot be directly accessed by software. Included in the SECS is a field that stores the enclave build measurement value. This field, MRENCLAVE, is initialized by the ECREATE instruction and updated by every EADD and EEXTEND. It is locked by EINIT.

Every enclave contains one or more TCS structures. The TCS contains meta-data used by the hardware to save and restore thread specific information when entering/exiting the enclave. There is one field, FLAGS, that may be accessed by software. This field can only be accessed by debug enclaves. The flag bit, DBGOPTIN, allows to single step into the thread associated with the TCS. (see Section 35.8.1)

The SECS is created when ECREATE (see Table 34-1) is executed. The TCS can be created using the EADD instruction or the SGX2 instructions (see Table 34-2).

34.5 ENCLAVE PAGE CACHE

The Enclave Page Cache (EPC) is the secure storage used to store enclave pages when they are a part of an executing enclave. For an EPC page, hardware performs additional access control checks to restrict access to the page. After the current page access checks and translations are performed, the hardware checks that the EPC page

is accessible to the program currently executing. Generally an EPC page is only accessed by the owner of the executing enclave or an instruction which is setting up an EPC page

The EPC is divided into EPC pages. An EPC page is 4KB in size and always aligned on a 4KB boundary.

Pages in the EPC can either be valid or invalid. Every valid page in the EPC belongs to one enclave instance. Each enclave instance has an EPC page that holds its SECS. The security metadata for each EPC page is held in an internal micro-architectural structure called Enclave Page Cache Map (EPCM, see Section 34.5.1).

The EPC is managed by privileged software. Intel SGX provides a set of instructions for adding and removing content to and from the EPC. The EPC may be configured by BIOS at boot time. On implementations in which EPC memory is part of system DRAM, the contents of the EPC are protected by an encryption engine.

34.5.1 Enclave Page Cache Map (EPCM)

The EPCM is a secure structure used by the processor to track the contents of the EPC. The EPCM holds one entry for each page in the EPC. The format of the EPCM is micro-architectural, and consequently is implementation dependent. However, the EPCM contains the following architectural information:

- The status of EPC page with respect to validity and accessibility.
- An SECS identifier (see Section 35.20) of the enclave to which the page belongs.
- The type of page: regular, SECS, TCS or VA.
- The linear address through which the enclave is allowed to access the page.
- The specified read/write/execute permissions on that page.

The EPCM structure is used by the CPU in the address-translation flow to enforce access-control on the EPC pages. The EPCM structure is described in Table 35-29, and the conceptual access-control flow is described in Section 35.5.

The EPCM entries are managed by the processor as part of various instruction flows.

34.6 ENCLAVE INSTRUCTIONS AND INTEL® SGX

The enclave instructions available with Intel SGX are organized as leaf functions under three instruction mnemonics: ENCLS (ring 0), ENCLU (ring 3), and ENCLV (VT root mode). Each leaf function uses EAX to specify the leaf function index, and may require additional implicit input registers as parameters. The use of EAX is implied implicitly by the ENCLS, ENCLU, and ENCLV instructions; ModR/M byte encoding is not used with ENCLS, ENCLU, and ENCLV. The use of additional registers does not use ModR/M encoding and is implied implicitly by the respective leaf function index.

Each leaf function index is also associated with a unique, leaf-specific mnemonic. A long-form expression of Intel SGX instruction takes the form of ENCLx[LEAF_MNEMONIC], where 'x' is either 'S', 'U', or 'V'. The long-form expression provides clear association of the privilege-level requirement of a given "leaf mnemonic". For simplicity, the unique "Leaf_Mnemonic" name is used (omitting the ENCLx for convenience) throughout in this document.

Details of individual SGX leaf functions are described in Chapter 38. Table 34-1 provides a summary of the instruction leaves that are available in the initial implementation of Intel SGX, which is introduced in the 6th generation Intel Core processors. Table 34-2 summarizes enhancement of Intel SGX for future Intel processors.

Table 34-1. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EADD]	Add an EPC page to an enclave.	ENCLU[EENTER]	Enter an enclave.
ENCLS[EBLOCK]	Block an EPC page.	ENCLU[EEXIT]	Exit an enclave.
ENCLS[ECREATE]	Create an enclave.	ENCLU[EGETKEY]	Create a cryptographic key.
ENCLS[EDBGRD]	Read data from a debug enclave by debugger.	ENCLU[EREPORT]	Create a cryptographic report.

Table 34-1. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EDBGWR]	Write data into a debug enclave by debugger.	ENCLU[ERESUME]	Re-enter an enclave.
ENCLS[EEXTEND]	Extend EPC page measurement.		
ENCLS[EINIT]	Initialize an enclave.		
ENCLS[ELDB]	Load an EPC page in blocked state.		
ENCLS[ELDU]	Load an EPC page in unblocked state.		
ENCLS[EPA]	Add an EPC page to create a version array.		
ENCLS[EREMOVE]	Remove an EPC page from an enclave.		
ENCLS[ETRACK]	Activate EBLOCK checks.		
ENCLS[EWB]	Write back/invalidate an EPC page.		

Table 34-2. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX2

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EAUG]	Allocate EPC page to an existing enclave.	ENCLU[EACCEPT]	Accept EPC page into the enclave.
ENCLS[EMODPR]	Restrict page permissions.	ENCLU[EMODPE]	Enhance page permissions.
ENCLS[EMODT]	Modify EPC page type.	ENCLU[EACCEPTCOPY]	Copy contents to an augmented EPC page and accept the EPC page into the enclave.

Table 34-3. VMX Operation and Supervisor Mode Enclave Instruction Leaf Functions in Long-Form of OVERSUB

VMX Operation	Description	Supervisor Instruction	Description
ENCLV[EDECVIRTCHILD]	Decrement the virtual child page count.	ENCLS[ERDINFO]	Read information about EPC page.
ENCLV[EINCVIRTCHILD]	Increment the virtual child page count.	ENCLS[ETRACKC]	Activate EBLOCK checks with conflict reporting.
ENCLV[ESETCONTEXT]	Set virtualization context.	ENCLS[ELDBC/UC]	Load an EPC page with conflict reporting.

34.7 DISCOVERING SUPPORT FOR INTEL® SGX AND ENABLING ENCLAVE INSTRUCTIONS

Detection of support of Intel SGX and enumeration of available and enabled Intel SGX resources are queried using the CPUID instruction. The enumeration interface comprises the following:

- Processor support of Intel SGX is enumerated by a feature flag in CPUID leaf 07H: CPUID.(EAX=07H, ECX=0H):EBX.SGX[bit 2]. If CPUID.(EAX=07H, ECX=0H):EBX.SGX = 1, the processor has support for Intel SGX, and requires opt-in enabling by BIOS via IA32_FEATURE_CONTROL MSR.

  If CPUID.(EAX=07H, ECX=0H):EBX.SGX = 1, CPUID will report via the available sub-leaves of CPUID.(EAX=12H) on available and/or configured Intel SGX resources.
- The available and configured Intel SGX resources enumerated by the sub-leaves of CPUID.(EAX=12H) depend on the state of BIOS configuration.

34.7.1 Intel® SGX Opt-In Configuration

On processors that support Intel SGX, IA32_FEATURE_CONTROL provides the SGX_ENABLE field (bit 18). Before system software can configure and enable Intel SGX resources, BIOS is required to set IA32_FEATURE_CONTROL.SGX_ENABLE = 1 to opt-in the use of Intel SGX by system software.

The semantics of setting SGX_ENABLE follows the rules of IA32_FEATURE_CONTROL.LOCK (bit 0). Software is considered to have opted into Intel SGX if and only if IA32_FEATURE_CONTROL.SGX_ENABLE and IA32_FEATURE_CONTROL.LOCK are set to 1. The setting of IA32_FEATURE_CONTROL.SGX_ENABLE (bit 18) is not reflected by CPUID.

Table 34-4. Intel® SGX Opt-in and Enabling Behavior

CPUID.(07H,0H):EBX.SGX	CPUID.(12H)	FEATURE_CONTROL.LOCK	FEATURE_CONTROL.SGX_ENABLE	Enclave Instruction
0	Invalid	X	X	#UD
1	Valid*	X	X	#UD**
1	Valid*	0	X	#GP
1	Valid*	1	0	#GP
1	Valid*	1	1	Available (see Table 34-5 for details of SGX1 and SGX2).
* Leaf 12H enumeration results are dependent on enablement. ** See list of conditions in the #UD section of the reference pages of ENCLS and ENCLU				

34.7.2 Intel® SGX Resource Enumeration Leaves

If CPUID.(EAX=07H, ECX=0H):EBX.SGX = 1, the processor also supports querying CPUID with EAX=12H on Intel SGX resource capability and configuration. The number of available sub-leaves in leaf 12H depends on the Opt-in and system software configuration. Information returned by CPUID.12H is thread specific; software should not assume that if Intel SGX instructions are supported on one hardware thread, they are also supported elsewhere.

A properly configured processor exposes Intel SGX functionality with CPUID.EAX=12H reporting valid information (non-zero content) in three or more sub-leaves, see Table 34-5.

- CPUID.(EAX=12H, ECX=0H) enumerates Intel SGX capability, including enclave instruction opcode support.
- CPUID.(EAX=12H, ECX=1H) enumerates Intel SGX capability of processor state configuration and enclave configuration in the SECS structure (see Table 35-3).
- CPUID.(EAX=12H, ECX >1) enumerates available EPC resources.

Table 34-5. CPUID Leaf 12H, Sub-Leaf 0 Enumeration of Intel® SGX Capabilities

CPUID.(EAX=12H,ECX=0)		Description Behavior
Register	Bits	
EAX	0	SGX1: If 1, indicates leaf functions of SGX1 instruction listed in Table 34-1 are supported.
	1	SGX2: If 1, indicates leaf functions of SGX2 instruction listed in Table 34-2 are supported.
	4:2	Reserved (0)
	5	OVERSUB: If 1, indicates Intel SGX supports instructions: EINCVIRTCHILD, EDECVIRTCHILD, and ESETCONTEXT.
	6	OVERSUB: If 1, indicates Intel SGX supports instructions: ETRACKC, ERDINFO, ELDBC, and ELDUC.
	31:7	Reserved (0)
EBX	31:0	MISCSELECT: Reports the bit vector of supported extended features that can be written to the MISC region of the SSA.
ECX	31:0	Reserved (0).

Table 34-5. CPUID Leaf 12H, Sub-Leaf 0 Enumeration of Intel® SGX Capabilities

CPUID.(EAX=12H,ECX=0)		Description Behavior
Register	Bits	
EDX	7:0	MaxEnclaveSize_Not64: the maximum supported enclave size is 2^(EDX[7:0]) bytes when not in 64-bit mode.
	15:8	MaxEnclaveSize_64: the maximum supported enclave size is 2^(EDX[15:8]) bytes when operating in 64-bit mode.
	31:16	Reserved (0).

Table 34-6. CPUID Leaf 12H, Sub-Leaf 1 Enumeration of Intel® SGX Capabilities

CPUID.(EAX=12H,ECX=1)		Description Behavior
Register	Bits	
EAX	31:0	Report the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE. SECS.ATTRIBUTES[n] can be set to 1 using ECREATE only if EAX[n] is 1, where n < 32.
EBX	31:0	Report the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE. SECS.ATTRIBUTES[n+32] can be set to 1 using ECREATE only if EBX[n] is 1, where n < 32.
ECX	31:0	Report the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE. SECS.ATTRIBUTES[n+64] can be set to 1 using ECREATE only if ECX[n] is 1, where n < 32.
EDX	31:0	Report the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE. SECS.ATTRIBUTES[n+96] can be set to 1 using ECREATE only if EDX[n] is 1, where n < 32.

On processors that support Intel SGX1 and SGX2, CPUID leaf 12H sub-leaf 2 report physical memory resources available for use with Intel SGX. These physical memory sections are typically allocated by BIOS as **Processor Reserved Memory**, and available to the OS to manage as EPC.

To enumerate how many EPC sections are available to the EPC manager, software can enumerate CPUID leaf 12H with sub-leaf index starting from 2, and decode the sub-leaf-type encoding (returned in EAX[3:0]) until the sub-leaf type is invalid. All invalid sub-leaves of CPUID leaf 12H return EAX/EBX/ECX/EDX with 0.

Table 34-7. CPUID Leaf 12H, Sub-Leaf Index 2 or Higher Enumeration of Intel® SGX Resources

CPUID.(EAX=12H,ECX > 1)		Description Behavior
Register	Bits	
EAX	3:0	0000b: This sub-leaf is invalid; EDX:ECX:EBX:EAX return 0. 0001b: This sub-leaf enumerates an EPC section. EBX:EAX and EDX:ECX provide information on the Enclave Page Cache (EPC) section. All other encodings are reserved.
	11:4	Reserved (enumerate 0).
	31:12	If EAX[3:0] = 0001b, these are bits 31:12 of the physical address of the base of the EPC section.
EBX	19:0	If EAX[3:0] = 0001b, these are bits 51:32 of the physical address of the base of the EPC section.
	31:20	Reserved.
ECX	3: 0	If ECX[3:0] = 0000b, then all bits of the EDX:ECX pair are enumerated as 0. If ECX[3:0] = 0001b, then this section has confidentiality and integrity protection. If ECX[3:0] = 0010b, then this section has confidentiality protection only. All other encodings are reserved.
	11:4	Reserved (enumerate 0).
	31:12	If EAX[3:0] = 0001b, these are bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory.

Table 34-7. CPUID Leaf 12H, Sub-Leaf Index 2 or Higher Enumeration of Intel® SGX Resources

CPUID.(EAX=12H,ECX > 1)		Description Behavior
Register	Bits	
EDX	19: 0	If EAX[3:0] = 0001b, these are bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory.
	31:20	Reserved.

34.8 INTEL® SGX INTERACTIONS WITH CONTROL-FLOW ENFORCEMENT TECHNOLOGY

This section discusses extensions to the Intel SGX architecture to support CET.

34.8.1 CET in Enclaves Model

Each enclave has its private configuration for CET that is not shared with the CET configurations of the enclosing application. On entry into the enclave, the CET state of the enclosing application is saved into scratchpad registers inside the processor and the CET state of the enclave is established. On an asynchronous exit, the enclave CET state is saved into the enclave state save area frame. On exit from the enclave, the CET state of the enclosing application is re-established from the scratchpad registers.

A new page type, PT_SS_FIRST, is used to denote pages in an enclave that can be used as a first page of a shadow stack.

A new page type, PT_SS_REST, is used to denote pages in an enclave that can be used as a non-first page of a shadow stack.

A page denoted as PT_SS_FIRST and PT_SS_REST will be a legal target for shadow_stack_load, shadow_stack_store, and regular load operations. Regular stores will be disallowed to such pages. A PT_SS_FIRST/PT_SS_REST page must be writeable in the IA page tables and in EPT.

When in enclave mode, shadow_stack_load and shadow_stack_store operations must be to addresses in the enclave ELRANGE.

The EAUG instruction is extended to allocate pages of type PT_SS_FIRST/PT_SS_REST; this page type requires specifying a SECINFO structure with page parameters. Shadow page permission must be R/W. Regular R/W pages may continue to be allocated by providing a SECINFO pointer value of 0. Regular R/W pages may also be allocated by providing a SECINFO structure that specifies the page parameters. The EAUG instruction creates a shadow-stack-restore token at offset 0xFF8 on a PT_SS_FIRST page. This allows a dynamically created shadow stack to be restored using the RSTORSSP instruction. The EADD and EAUG instructions disallow creation of a PT_SS_FIRST or PT_SS_REST page as the first or last page in ELRANGE.

The EADD instruction requires that the PT_SS_REST page be all zeroes. The EADD instruction requires that a PT_SS_FIRST page be all zeroes except the 8 bytes at offset 0xFF8 on that page that must have a shadow-stack-restore token. This shadow-stack-restore token must have a linear address which is the linear address of the PT_SS_FIRST page + 4096. As an enclave could be loaded at varying linear addresses, the enclave builder should not extend the measurement of the PT_SS_FIRST pages into the measurement registers. On first entry on to the enclave using a TCS, the enclave software can use the RSTORSSP instruction to restore its SSP. Subsequent to performing a RSTORSSP, the enclave software can use the INCSSP instruction to pop the previous-ssp token that is created by the RSTORSSP instruction at the top of the restored shadow stack.

On an enclave entry, the SSP will be initialized to the value in a new TCS field called PREVSSP. The PREVSSP field is written with the value of SSP on enclave exit and is loaded into SSP at enclave entry. When a TCS page is added using EADD or accepted using EACCEPT, the processor requires the PREVSSP field to be initialized to 0.

34.8.2 Operations Not Supported on Shadow Stack Pages

The following operations are not allowed on pages of type PT_SS_FIRST and PT_SS_REST:

- EACCEPTCOPY
- EMODPR
- EMODPE

34.8.3 Indirect Branch Tracking – Legacy Compatibility Treatment

The legacy code page bitmap is tested using the page offset within the ELRANGE instead of the absolute linear address of the address where ENDBRANCH was missed; see the detailed algorithm in Section 17.3.6, "Legacy Compatibility Treatment," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for additional details.

CHAPTER 35
ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

35.1 OVERVIEW OF ENCLAVE EXECUTION ENVIRONMENT

When an enclave is created, it has a range of linear addresses to which the processor applies enhanced access control. This range is called the ELRANGE (see Section 34.3). When an enclave generates a memory access, the existing IA32 segmentation and paging architecture are applied. Additionally, linear addresses inside the ELRANGE must map to an EPC page otherwise when an enclave attempts to access that linear address a fault is generated.

The EPC pages need not be physically contiguous. System software allocates EPC pages to various enclaves. Enclaves must abide by OS/VMM imposed segmentation and paging policies. OS/VMM-managed page tables and extended page tables provide address translation for the enclave pages. Hardware requires that these pages are properly mapped to EPC (any failure generates an exception).

Enclave entry must happen through specific enclave instructions:

- ENCLU[EENTER], ENCLU[ERESUME].

Enclave exit must happen through specific enclave instructions or events:

- ENCLU[EEXIT], Asynchronous Enclave Exit (AEX).

Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations. As an example a read of an enclave page may result in the return of all one's or return of cyphertext of the cache line. Writing to an enclave page may result in a dropped write or a machine check at a later time. The processor will provide the protections as described in Section 35.4 and Section 35.5 on such accesses.

35.2 TERMINOLOGY

A memory access to the ELRANGE and initiated by an instruction executed by an enclave is called a Direct Enclave Access (Direct EA).

Memory accesses initiated by certain Intel® SGX instruction leaf functions such as ECREATE, EADD, EDBGRD, EDBGWR, ELDU/ELDB, EWB, EREMOVE, EENTER, and ERESUME to EPC pages are called Indirect Enclave Accesses (Indirect EA). Table 35-1 lists additional details of the indirect EA of SGX1 and SGX2 extensions.

Direct EAs and Indirect EAs together are called Enclave Accesses (EAs).

Any memory access that is not an Enclave Access is called a non-enclave access.

35.3 ACCESS-CONTROL REQUIREMENTS

Enclave accesses have the following access-control attributes:

- All memory accesses must conform to segmentation and paging protection mechanisms.
- Code fetches from inside an enclave to a linear address outside that enclave result in a #GP(0) exception.
- Shadow-stack-load or shadow-stack-store from inside an enclave to a linear address outside that enclave results in a #GP(0) exception.
- Non-enclave accesses to EPC memory result in undefined behavior. EPC memory is protected as described in Section 35.4 and Section 35.5 on such accesses.
- EPC pages of page types PT_REG, PT_TCS, and PT_TRIM must be mapped to ELRANGE at the linear address specified when the EPC page was allocated to the enclave using ENCLS[EADD] or ENCLS[EAUG] leaf functions. Enclave accesses through other linear address result in a #PF with the PFEC.SGX bit set.

- Direct EAs to any EPC pages must conform to the currently defined security attributes for that EPC page in the EPCM. These attributes may be defined at enclave creation time (EADD) or when the enclave sets them using SGX2 instructions. The failure of these checks results in a #PF with the PFEC.SGX bit set.
  - Target page must belong to the currently executing enclave.
  - Data may be written to an EPC page if the EPCM allow write access.
  - Data may be read from an EPC page if the EPCM allow read access.
  - Instruction fetches from an EPC page are allowed if the EPCM allows execute access.
  - Shadow-stack-load from an EPC page and shadow-stack-store to an EPC page are allowed only if the page type is PT_SS_FIRST or PT_SS_REST.
  - Data writes that are not shadow-stack-store are not allowed if the EPCM page type is PT_SS_FIRST or PT_SS_REST.
  - Target page must not have a restricted page type[1] (PT_SECS, PT_TCS, PT_VA, or PT_TRIM).
  - The EPC page must not be BLOCKED.
  - The EPC page must not be PENDING.
  - The EPC page must not be MODIFIED.

35.4 SEGMENT-BASED ACCESS CONTROL

Intel SGX architecture does not modify the segment checks performed by a logical processor. All memory accesses arising from a logical processor in protected mode (including enclave access) are subject to segmentation checks with the applicable segment register.

To ensure that outside entities do not modify the enclave's logical-to-linear address translation in an unexpected fashion, ENCLU[EENTER] and ENCLU[ERESUME] check that CS, DS, ES, and SS, if usable (i.e., not null), have segment base value of zero. A non-zero segment base value for these registers results in a #GP(0).

On enclave entry either via EENTER or ERESUME, the processor saves the contents of the external FS and GS registers, and loads these registers with values stored in the TCS at build time to enable the enclave’s use of these registers for accessing the thread-local storage inside the enclave. On EEXIT and AEX, the contents at time of entry are restored. On AEX, the values of FS and GS are saved in the SSA frame. On ERESUME, FS and GS are restored from the SSA frame. The details of these operations can be found in the descriptions of EENTER, ERESUME, EEXIT, and AEX flows.

35.5 PAGE-BASED ACCESS CONTROL

35.5.1 Access-control for Accesses that Originate from Non-SGX Instructions

Intel SGX builds on the processor's paging mechanism to provide page-granular access-control for enclave pages. Enclave pages are designed to be accessible only from inside the currently executing enclave if they belong to that enclave. In addition, enclave accesses must conform to the access control requirements described in Section 35.3. or through certain Intel SGX instructions. Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations.

35.5.2 Memory Accesses that Split Across ELRANGE

Memory data accesses are allowed to split across ELRANGE (i.e., a part of the access is inside ELRANGE and a part of the access is outside ELRANGE) while the processor is inside an enclave. If an access splits across ELRANGE, the

1. EPCM may allow write, read or execute access only for pages with page type PT_REG.

processor splits the access into two sub-accesses (one inside ELRANGE and the other outside ELRANGE), and each access is evaluated. A code-fetch access that splits across ELRANGE results in a #GP due to the portion that lies outside of the ELRANGE.

35.5.3 Implicit vs. Explicit Accesses

Memory accesses originating from Intel SGX instruction leaf functions are categorized as either explicit accesses or implicit accesses. Table 35-1 lists the implicit and explicit memory accesses made by Intel SGX leaf functions.

35.5.3.1 Explicit Accesses

Accesses to memory locations provided as explicit operands to Intel SGX instruction leaf functions, or their linked data structures are called explicit accesses.

Explicit accesses are always made using logical addresses. These accesses are subject to segmentation, paging, extended paging, and APIC-virtualization checks, and trigger any faults/exit associated with these checks when the access is made.

The interaction of explicit memory accesses with data breakpoints is leaf-function-specific, and is documented in Section 40.3.4.

35.5.3.2 Implicit Accesses

Accesses to data structures whose physical addresses are cached by the processor are called implicit accesses. These addresses are not passed as operands of the instruction but are implied by use of the instruction.

These accesses do not trigger any access-control faults/exits or data breakpoints. Table 35-1 lists memory objects that Intel SGX instruction leaf functions access either by explicit access or implicit access. The addresses of explicit access objects are passed via register operands with the second through fourth column of Table 35-1 matching implicitly encoded registers RBX, RCX, RDX.

Physical addresses used in different implicit accesses are cached via different instructions and for different durations. The physical address of SECS associated with each EPC page is cached at the time the page is added to the enclave via ENCLS[EADD] or ENCLS[EAUG], or when the page is loaded to EPC via ENCLS[ELDB] or ENCLS[ELDU]. This binding is severed when the corresponding page is removed from the EPC via ENCLS[EREMOVE] or ENCLS[EWB]. Physical addresses of TCS and SSA pages are cached at the time of most-recent enclave entry. Exit from an enclave (ENCLU[EEXIT] or AEX) flushes this caching. Details of Asynchronous Enclave Exit is described in Chapter 37.

The physical addresses that are cached for use by implicit accesses are derived from logical (or linear) addresses after checks such as segmentation, paging, EPT, and APIC virtualization checks. These checks may trigger exceptions or VM exits. Note, however, that such exception or VM exits may not occur after a physical address is cached and used for an implicit access.

Table 35-1. List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions

Instr. Leaf	Enum.	Explicit 1	Explicit 2	Explicit 3	Implicit
EACCEPT	SGX2	SECINFO	EPCPAGE		SECS
EACCEPTCOPY	SGX2	SECINFO	EPCPAGE (Src)	EPCPAGE (Dst)	
EADD	SGX1	PAGEINFO and linked structures	EPCPAGE		
EAUG	SGX2	PAGEINFO and linked structures	EPCPAGE		SECS
EBLOCK	SGX1	EPCPAGE			SECS
ECREATE	SGX1	PAGEINFO and linked structures	EPCPAGE		
EDBGRD	SGX1	EPCADDR	Destination		SECS
EDBGWR	SGX1	EPCADDR	Source		SECS
EDECVIRTCHILD	OVERSUB	EPCPAGE	SECS		
EENTER	SGX1	TCS and linked SSA			SECS

Table 35-1. List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions (Contd.)

Instr. Leaf	Enum.	Explicit 1	Explicit 2	Explicit 3	Implicit
EEXIT	SGX1				SECS, TCS
EEXTEND	SGX1	SECS	EPCPAGE		
EGETKEY	SGX1	KEYREQUEST	KEY		SECS
EINCVIRTCHILD	OVERSUB	EPCPAGE	SECS		
EINIT	SGX1	SIGSTRUCT	SECS	EINITTOKEN	
ELDB/ELDU	SGX1	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	
ELDBC/ELDUC	OVERSUB	PAGEINFO and linked structures	EPCPAGE	VAPAGE	
EMODPE	SGX2	SECINFO	EPCPAGE		
EMODPR	SGX2	SECINFO	EPCPAGE		SECS
EMODT	SGX2	SECINFO	EPCPAGE		SECS
EPA	SGX1	EPCADDR			
ERDINFO	OVERSUB	RDINFO	EPCPAGE		
EREMOVE	SGX1	EPCPAGE			SECS
EREPORT	SGX1	TARGETINFO	REPORTDATA	OUTPUTDATA	SECS
ERESUME	SGX1	TCS and linked SSA			SECS
ESETCONTEXT	OVERSUB		SECS	ContextValue	
ETRACK	SGX1	EPCPAGE			
ETRACKC	OVERSUB		EPCPAGE		
EWB	SGX1	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	SECS
Asynchronous Enclave Exit*					SECS, TCS, SSA
*Details of Asynchronous Enclave Exit (AEX) is described in Section 37.4					

35.6 INTEL® SGX DATA STRUCTURES OVERVIEW

Enclave operation is managed via a collection of data structures. Many of the top-level data structures contain sub-structures. The top-level data structures relate to parameters that may be used in enclave setup/maintenance, by Intel SGX instructions, or AEX event. The top-level data structures are:

- SGX Enclave Control Structure (SECS)
- Thread Control Structure (TCS)
- State Save Area (SSA)
- Page Information (PAGEINFO)
- Security Information (SECINFO)
- Paging Crypto MetaData (PCMD)
- Enclave Signature Structure (SIGSTRUCT)
- EINIT Token Structure (EINITTOKEN)
- Report Structure (REPORT)
- Report Target Info (TARGETINFO)
- Key Request (KEYREQUEST)
- Version Array (VA)
- Enclave Page Cache Map (EPCM)
- Read Info (RDINFO)

Details of the top-level data structures and associated sub-structures are listed in Section 35.7 through Section 35.20.

35.7 SGX ENCLAVE CONTROL STRUCTURE (SECS)

The SECS data structure requires 4K-Bytes alignment.

Table 35-2. Layout of SGX Enclave Control Structure (SECS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
SIZE	0	8	Size of enclave in bytes; must be power of 2.
BASEADDR	8	8	Enclave Base Linear Address must be naturally aligned to size.
SSAFRAMESIZE	16	4	Size of one SSA frame in pages, including XSAVE, pad, GPR, and MISC (if CPUID.(EAX=12H, ECX=0):.EBX != 0).
MISCSELECT	20	4	Bit vector specifying which extended features are saved to the MISC region (see Section 35.7.2) of the SSA frame when an AEX occurs.
CET_LEG_BITMAP_OFFSET	24	8	Page aligned offset of legacy code page bitmap from enclave base. Software is expected to program this offset such that the entire bitmap resides in the ELRANGE when legacy compatibility mode for indirect branch tracking is enabled. However this is not enforced by the hardware. This field exists when CPUID.(EAX=7, ECX=0):EDX.CET_IBT[bit 20] is enumerated as 1, else it is reserved.
CET_ATTRIBUTES	32	1	CET feature attributes of the enclave; see Table 35-5. This field exists when CPUID.(EAX=12,ECX=1):EAX[6] is enumerated as 1, else it is reserved.
RESERVED	33	15	
ATTRIBUTES	48	16	Attributes of the Enclave, see Table 35-3.
MRENCLAVE	64	32	Measurement Register of enclave build process. See SIGSTRUCT for format.
RESERVED	96	32	
MRSIGNER	128	32	Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for format.
RESERVED	160	32	
CONFIGID	192	64	Post EINIT configuration identity.
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Post EINIT configuration security version number (SVN).
RESERVED	262	3834	The RESERVED field consists of the following: ▪ EID: An 8 byte Enclave Identifier. Its location is implementation specific. ▪ PAD: A 352 bytes padding pattern from the Signature (used for key derivation strings). It's location is implementation specific. ▪ VIRTCHILDCNT: An 8 byte Count of virtual children that have been paged out by a VMM. Its location is implementation specific. ▪ ENCLAVECONTEXT: An 8 byte Enclave context pointer. Its location is implementation specific. ▪ ISVFAMILYID: A 16 byte value assigned to identify the family of products the enclave belongs to. ▪ ISVEXTPRODID: A 16 byte value assigned to identify the product identity of the enclave. ▪ The remaining 3226 bytes are reserved area. The entire 3834 byte field must be cleared prior to executing ECREATE.

35.7.1 ATTRIBUTES

The ATTRIBUTES data structure is comprised of bit-granular fields that are used in the SECS, the REPORT and the KEYREQUEST structures. CPUID.(EAX=12H, ECX=1) enumerates a bitmap of permitted 1-setting of bits in ATTRIBUTES.

Table 35-3. Layout of ATTRIBUTES Structure

Field	Bit Position	Description
INIT	0	This bit indicates if the enclave has been initialized by EINIT. It must be cleared when loaded as part of ECREATE. For EREPORT instruction, TARGET_INFO.ATTRIBUTES[ENIT] must always be 1 to match the state after EINIT has initialized the enclave.
DEBUG	1	If 1, the enclave permit debugger to read and write enclave data using EDBGRD and EDBGWR.
MODE64BIT	2	Enclave runs in 64-bit mode.
RESERVED	3	Must be Zero.
PROVISIONKEY	4	Provisioning Key is available from EGETKEY.
EINITTOKEN_KEY	5	EINIT token key is available from EGETKEY.
CET	6	Enable CET attributes. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0 this bit is reserved and must be 0.
KSS	7	Key Separation and Sharing Enabled.
RESERVED	9:8	Must be zero.
AEXNOTIFY	10	The bit indicates that threads within the enclave may receive AEX notifications.
RESERVED	63:11	Must be zero.
XFRM	127:64	XSAVE Feature Request Mask. See Section 39.7.

35.7.2 SECS.MISCSELECT Field

CPUID.(EAX=12H, ECX=0):EBX[31:0] enumerates which extended information that the processor can save into the MISC region of SSA when an AEX occurs. An enclave writer can specify via SIGSTRUCT how to set the SECS.MISCSELECT field. The bit vector of MISCSELECT selects which extended information is to be saved in the MISC region of the SSA frame when an AEX is generated. The bit vector definition of extended information is listed in Table 35-4.

If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, SECS.MISCSELECT field must be all zeros.

The SECS.MISCSELECT field determines the size of MISC region of the SSA frame, see Section 35.9.2.

Table 35-4. Bit Vector Layout of MISCSELECT Field of Extended Information

Field	Bit Position	Description
EXINFO	0	Report information about page fault and general protection exception that occurred inside an enclave.
CPINFO	1	Report information about control protection exception that occurred inside an enclave. When CPUID.(EAX=12H, ECX=0):EBX[1] is 0, this bit is reserved.
Reserved	31:2	Reserved (0).

35.7.3 SECS.CET_ATTRIBUTES Field

The SECS.CET_ATTRIBUTES field can be used by the enclave writer to enable various CET attributes in an enclave. This field exists when CPUID.(EAX=12,ECX=1):EAX[6] is enumerated as 1. Bits 1:0 are defined when CPUID.(EAX=7, ECX=0):ECX.CET_SS is 1, and bits 5:2 are defined when CPUID.(EAX=7, ECX=0):EDX.CET_IBT is 1.

Table 35-5. Bit Vector Layout of CET_ATTRIBUTES Field of Extended Information

Field	Bit Position	Description
SH_STK_EN	0	When set to 1, enable shadow stacks.
WR_SHSTK_EN	1	When set to 1, enables the WRSS{D,Q}W instructions.
ENDBR_EN	2	When set to 1, enables indirect branch tracking.
LEG_IW_EN	3	Enable legacy compatibility treatment for indirect branch tracking.
NO_TRACK_EN	4	When set to 1, enables use of no-track prefix for indirect branch tracking.
SUPPRESS_DIS	5	When set to 1, disables suppression of CET indirect branch tracking on legacy compatibility.
Reserved	7:6	Reserved (0).

35.8 THREAD CONTROL STRUCTURE (TCS)

Each executing thread in the enclave is associated with a Thread Control Structure. It requires 4K-Bytes alignment.

Table 35-6. Layout of Thread Control Structure (TCS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
STAGE	0	8	Enclave execution state of the thread controlled by this TCS. A value of 0 indicates that this TCS is available for enclave entry. A value of 1 indicates that a logical processor is currently executing an enclave in the context of this TCS.
FLAGS	8	8	The thread's execution flags (see Section 35.8.1).
OSSA	16	8	Offset of the base of the State Save Area stack, relative to the enclave base. Must be page aligned.
CSSA	24	4	Current slot index of an SSA frame, cleared by EADD and EACCEPT.
NSSA	28	4	Number of available slots for SSA frames.
OENTRY	32	8	Offset in enclave to which control is transferred on EENTER relative to the base of the enclave.
AEP	40	8	The value of the Asynchronous Exit Pointer that was saved at EENTER time.
OFSBASE	48	8	Offset to add to the base address of the enclave for producing the base address of FS segment inside the enclave. Must be page aligned.
OGSBASE	56	8	Offset to add to the base address of the enclave for producing the base address of GS segment inside the enclave. Must be page aligned.
FSLIMIT	64	4	Size to become the new FS limit in 32-bit mode.
GSLIMIT	68	4	Size to become the new GS limit in 32-bit mode.
OCETSSA	72	8	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the offset of the CET state save area from enclave base. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.
PREVSSP	80	8	When CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] is 1, this field records the SSP at the time of AEX or EEXIT; used to setup SSP on entry. When CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] is 0, this field is reserved and must be 0.
RESERVED	72	4024	Must be zero.

35.8.1 TCS.FLAGS

Table 35-7. Layout of TCS.FLAGS Field

Field	Bit Position	Description
DBGOPTIN	0	If set, allows debugging features (single-stepping, breakpoints, etc.) to be enabled and active while executing in the enclave on this TCS. Hardware clears this bit on EADD. A debugger may later modify it if the enclave's ATTRIBUTES.DEBUG is set.
AEXNOTIFY	1	A thread that enters the enclave cannot receive AEX notifications unless this flag is set to 1.
RESERVED	63:2	Must be zero.

35.8.2 State Save Area Offset (OSSA)

The OSSA points to a stack of State Save Area (SSA) frames (see Section 35.9) used to save the processor state when an interrupt or exception occurs while executing in the enclave.

35.8.3 Current State Save Area Frame (CSSA)

CSSA is the index of the current SSA frame that will be used by the processor to determine where to save the processor state on an interrupt or exception that occurs while executing in the enclave. It is an index into the array of frames addressed by OSSA. CSSA is incremented on an AEX and decremented on an ERESUME.

35.8.4 Number of State Save Area Frames (NSSA)

NSSA specifies the number of SSA frames available for this TCS. There must be at least one available SSA frame when EENTER-ing the enclave or the EENTER will fail.

35.9 STATE SAVE AREA (SSA) FRAME

When an AEX occurs while running in an enclave, the architectural state is saved in the thread's current SSA frame, which is pointed to by TCS.CSSA. An SSA frame must be page aligned, and contains the following regions:

- The XSAVE region starts at the base of the SSA frame, this region contains extended feature register state in an XSAVE/FXSAVE-compatible non-compacted format.
- A Pad region: software may choose to maintain a pad region separating the XSAVE region and the MISC region. Software choose the size of the pad region according to the sizes of the MISC and GPRSGX regions.
- The GPRSGX region. The GPRSGX region is the last region of an SSA frame (see Table 35-8). This is used to hold the processor general purpose registers (RAX … R15), the RIP, the outside RSP and RBP, RFLAGS, and the AEX information.
- The MISC region (If CPUIDEAX=12H, ECX=0):EBX[31:0] != 0). The MISC region is adjacent to the GRPSGX region, and may contain zero or more components of extended information that would be saved when an AEX occurs. If the MISC region is absent, the region between the GPRSGX and XSAVE regions is the pad region that software can use. If the MISC region is present, the region between the MISC and XSAVE regions is the pad region that software can use. See additional details in Section 35.9.2.

Table 35-8. Top-to-Bottom Layout of an SSA Frame

Region	Offset (Byte)	Size (Bytes)	Description
XSAVE	0	Calculate using CPUID leaf 0DH information	The size of XSAVE region in SSA is derived from the enclave's support of the collection of processor extended states that would be managed by XSAVE. The enablement of those processor extended state components in conjunction with CPUID leaf 0DH information determines the XSAVE region size in SSA.
Pad	End of XSAVE region	Chosen by enclave writer	Ensure the end of GPRSGX region is aligned to the end of a 4KB page.
MISC	base of GPRSGX – sizeof(MISC)	Calculate from highest set bit of SECS.MISCSELECT	See Section 35.9.2.
GPRSGX	SSAFRAMESIZE – 176	176	See Table 35-9 for layout of the GPRSGX region.

35.9.1 GPRSGX Region

The layout of the GPRSGX region is shown in Table 35-9.

Table 35-9. Layout of GPRSGX Portion of the State Save Area

Field	OFFSET (Bytes)	Size (Bytes)	Description
RAX	0	8	
RCX	8	8	
RDX	16	8	
RBX	24	8	
RSP	32	8	
RBP	40	8	
RSI	48	8	
RDI	56	8	
R8	64	8	
R9	72	8	
R10	80	8	
R11	88	8	
R12	96	8	
R13	104	8	
R14	112	8	
R15	120	8	
RFLAGS	128	8	Flag register.
RIP	136	8	Instruction pointer.
URSP	144	8	Non-Enclave (outside) stack pointer. Saved by EENTER, restored on AEX.
URBP	152	8	Non-Enclave (outside) RBP pointer. Saved by EENTER, restored on AEX.
EXITINFO	160	4	Contains information about exceptions that cause AEXs, which might be needed by enclave software (see Section 35.9.1.1).
RESERVED	164	3	

Table 35-9. Layout of GPRSGX Portion of the State Save Area (Contd.)

Field	OFFSET (Bytes)	Size (Bytes)	Description
AEXNOTIFY	167	1	Bit 0: This bit allows enclave software to dynamically enable/disable AEX notifications. An enclave thread cannot receive AEX notifications unless this bit is set to 1 in the thread's current SSA frame. All other bits are reserved.
FSBASE	168	8	FS BASE.
GSBASE	176	8	GS BASE.

35.9.1.1 EXITINFO

EXITINFO contains the information used to report exit reasons to software inside the enclave. It is a 4 byte field laid out as in Table 35-10. The VALID bit is set only for the exceptions conditions which are reported inside an enclave. See Table 35-11 for which exceptions are reported inside the enclave. If the exception condition is not one reported inside the enclave then VECTOR and EXIT_TYPE are cleared.

When a higher priority event, such as SMI, and a pending debug exception occur at the same time when executing inside an enclave, the higher priority event has precedence. As an example for an SMI, the SSA exit info is zero. The debug exception will be delivered upon return from the SMI. In such cases, the EXITINFO field will not contain the information of a debug exception.

Table 35-10. Layout of EXITINFO Field

Field	Bit Position	Description
VECTOR	7:0	Exception number of exceptions reported inside enclave.
EXIT_TYPE	10:8	011b: Hardware exceptions. 110b: Software exceptions. Other values: Reserved.
RESERVED	30:11	Reserved as zero.
VALID	31	0: unsupported exceptions. 1: Supported exceptions. Includes two categories: • Unconditionally supported exceptions: #DE, #DB, #BP, #BR, #UD, #MF, #AC, #XM. • Conditionally supported exception: — #PF, #GP if SECS.MISCSELECT.EXINFO = 1. — #CP if SECS.MISCSELECT.CPINFO=1.

35.9.1.2 VECTOR Field Definition

Table 35-11 contains the VECTOR field. This field contains information about some exceptions which occur inside the enclave. These vector values are the same as the values that would be used when vectoring into regular exception handlers. All values not shown are not reported inside an enclave.

Table 35-11. Exception Vectors

Name	Vector #	Description
#DE	0	Divider exception.
#DB	1	Debug exception.
#BP	3	Breakpoint exception.
#BR	5	Bound range exceeded exception.
#UD	6	Invalid opcode exception.
#GP	13	General protection exception. Only reported if SECS.MISCSELECT.EXINFO = 1.
#PF	14	Page fault exception. Only reported if SECS.MISCSELECT.EXINFO = 1.

Table 35-11. Exception Vectors (Contd.)

Name	Vector #	Description
#MF	16	x87 FPU floating-point error.
#AC	17	Alignment check exceptions.
#XM	19	SIMD floating-point exceptions.
#CP	21	Control protection exception. Only reported if SECS.MISCSELECT.CPINFO=1.

35.9.2 MISC Region

The layout of the MISC region is shown in Table 35-12. The number of components that the processor supports in the MISC region corresponds to the bits of CPUID.(EAX=12H, ECX=0):EBX[31:0] set to 1. Each set bit in CPUID.(EAX=12H, ECX=0):EBX[31:0] has a defined size for the corresponding component, as shown in Table 35-12. Enclave writers needs to do the following:

- Decide which MISC region components will be supported for the enclave.
- Allocate an SSA frame large enough to hold the components chosen above.
- Instruct each enclave builder software to set the appropriate bits in SECS.MISCSELECT.

The first component, EXINFO, starts next to the GPRSGX region. Additional components in the MISC region grow in ascending order within the MISC region towards the XSAVE region.

The size of the MISC region is calculated as follows:

- If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISC region is not supported.
- If CPUID.(EAX=12H, ECX=0):EBX[31:0] != 0, the size of MISC region is derived from sum of the highest bit set in SECS.MISCSELECT and the size of the MISC component corresponding to that bit. Offset and size information of currently defined MISC components are listed in Table 35-12. For example, if the highest bit set in SECS.MISCSELECT is bit 0, the MISC region offset is OFFSET(GPRSGX)-16 and size is 16 bytes.
- The processor saves a MISC component i in the MISC region if and only if SECS.MISCSELECT[i] is 1.

Table 35-12. Layout of MISC region of the State Save Area

MISC Components	OFFSET (Bytes)	Size (Bytes)	Description
EXINFO	Offset(GPRSGX) -16	16	If CPUID.(EAX=12H, ECX=0):EBX[0] = 1, exception information on #GP or #PF that occurred inside an enclave can be written to the EXINFO structure if specified by SECS.MISCSELECT[0] = 1. If CPUID.(EAX=12H, ECX=0):EBX[1] = 1, exception information on #CP that occurred inside an enclave can be written to the EXINFO structure if specified by SECS.MISCSELECT[1] = 1.
Future Extension	Below EXINFO	TBD	Reserved. (Zero size if CPUID.(EAX=12H, ECX=0):EBX[31:1] =0).

35.9.2.1 EXINFO Structure

Table 35-13 contains the layout of the EXINFO structure that provides additional information.

Table 35-13. Layout of EXINFO Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MADDR	0	8	If #PF: contains the page fault linear address that caused a page fault. If #GP: the field is cleared. If #CP: the field is cleared.
ERRCD	8	4	Exception error code for either #GP or #PF.
RESERVED	12	4	

35.9.2.2 Page Fault Error Code

Table 35-14 contains page fault error code that may be reported in EXINFO.ERRCD.

Table 35-14. Page Fault Error Code

Name	Bit Position	Description
P	0	Same as non-SGX page fault exception P flag.
W/R	1	Same as non-SGX page fault exception W/R flag.
U/S[1]	2	Always set to 1 (user mode reference).
RSVD	3	Same as non-SGX page fault exception RSVD flag.
I/D	4	Same as non-SGX page fault exception I/D flag.
PK	5	Protection Key induced fault.
RSVD	14:6	Reserved.
SGX	15	EPCM induced fault.
RSVD	31:5	Reserved.

NOTES:

1. Page faults incident to enclave mode that report U/S=0 are not reported in EXINFO.

35.10 CET STATE SAVE AREA FRAME

The CET state save area consists of an array of CET state save frames. The number of CET state save frames is equal to the TCS.NSSA. The current CET SSA frame is indicated by TCS.CSSA. The offset of the CET state save area is specified by TCS.OCETSSA.

Table 35-15. Layout of CET State Save Area Frame

Field	Offset (Bytes)	Size (Bytes)	Description
SSP	0	8	Shadow Stack Pointer. This field is reserved when CPUID.(EAX=7, ECX=0):ECX[CET_SS] is 0.
IB_TRACK_STATE	8	8	Indirect branch tracker state: Bit 0: SUPPRESS – suppressed(1), tracking(0) Bit 1: TRACKER - IDLE (0), WAIT_FOR_ENDBRANCH (1) Bits 63:2 – Reserved This field is reserved when CPUID.(EAX=7, ECX=0):EDX[CET_IBT] is 0.

35.11 PAGE INFORMATION (PAGEINFO)

PAGEINFO is an architectural data structure that is used as a parameter to the EPC-management instructions. It requires 32-Byte alignment.

Table 35-16. Layout of PAGEINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
LINADDR	0	8	Enclave linear address.
SRCPGE	8	8	Effective address of the page where contents are located.
SECINFO/PCMD	16	8	Effective address of the SECINFO or PCMD (for ELDU, ELDB, EWB) structure for the page.
SECS	24	8	Effective address of EPC slot that currently contains the SECS.

35.12 SECURITY INFORMATION (SECINFO)

The SECINFO data structure holds meta-data about an enclave page.

Table 35-17. Layout of SECINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
FLAGS	0	8	Flags describing the state of the enclave page.
RESERVED	8	56	Must be zero.

35.12.1 SECINFO.FLAGS

The SECINFO.FLAGS are a set of fields describing the properties of an enclave page.

Table 35-18. Layout of SECINFO.FLAGS Field

Field	Bit Position	Description
R	0	If 1 indicates that the page can be read from inside the enclave; otherwise the page cannot be read from inside the enclave.
W	1	If 1 indicates that the page can be written from inside the enclave; otherwise the page cannot be written from inside the enclave.
X	2	If 1 indicates that the page can be executed from inside the enclave; otherwise the page cannot be executed from inside the enclave.
PENDING	3	If 1 indicates that the page is in the PENDING state; otherwise the page is not in the PENDING state.
MODIFIED	4	If 1 indicates that the page is in the MODIFIED state; otherwise the page is not in the MODIFIED state.
PR	5	If 1 indicates that a permission restriction operation on the page is in progress, otherwise a permission restriction operation is not in progress.
RESERVED	7:6	Must be zero.
PAGE_TYPE	15:8	The type of page that the SECINFO is associated with.
RESERVED	63:16	Must be zero.

35.12.2 PAGE_TYPE Field Definition

The SECINFO flags and EPC flags contain bits indicating the type of page.

Table 35-19. Supported PAGE_TYPE

TYPE	Value	Description
PT_SECS	0	Page is an SECS.
PT_TCS	1	Page is a TCS.
PT_REG	2	Page is a regular page.
PT_VA	3	Page is a Version Array.
PT_TRIM	4	Page is in trimmed state.
PT_SS_FIRST	5	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, Page is first page of a shadow stack. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this value is reserved.
PT_SS_REST	6	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, Page is not first page of a shadow stack. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this value is reserved.
	All others	Reserved.

35.13 PAGING CRYPTO METADATA (PCMD)

The PCMD structure is used to keep track of crypto meta-data associated with a paged-out page. Combined with PAGEINFO, it provides enough information for the processor to verify, decrypt, and reload a paged-out EPC page. The size of the PCMD structure (128 bytes) is architectural.

EWB calculates the Message Authentication Code (MAC) value and writes out the PCMD. ELDB/U reads the fields and checks the MAC.

The format of PCMD is as follows:

Table 35-20. Layout of PCMD Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
SECINFO	0	64	Flags describing the state of the enclave page; R/W by software.
ENCLAVEID	64	8	Enclave Identifier used to establish a cryptographic binding between paged-out page and the enclave.
RESERVED	72	40	Must be zero.
MAC	112	16	Message Authentication Code for the page, page meta-data and reserved field.

35.14 ENCLAVE SIGNATURE STRUCTURE (SIGSTRUCT)

SIGSTRUCT is a structure created and signed by the enclave developer that contains information about the enclave. SIGSTRUCT is processed by the EINIT leaf function to verify that the enclave was properly built.

SIGSTRUCT includes ENCLAVEHASH as SHA256 digest, as defined in FIPS PUB 180-4. The digests are byte strings of length 32. Each of the 8 HASH dwords is stored in little-endian order.

SIGSTRUCT includes four 3072-bit integers (MODULUS, SIGNATURE, Q1, Q2). Each such integer is represented as a byte strings of length 384, with the most significant byte at the position "offset + 383", and the least significant byte at position "offset".

The (3072-bit integer) SIGNATURE should be an RSA signature, where: a) the RSA modulus (MODULUS) is a 3072-bit integer; b) the public exponent is set to 3; c) the signing procedure uses the EMSA-PKCS1-v1.5 format with DER encoding of the "DigestInfo" value as specified in of PKCS#1 v2.1/RFC 3447.

The 3072-bit integers Q1 and Q2 are defined by:

q1 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

SIGSTRUCT must be page aligned

In column 5 of Table 35-21, 'Y' indicates that this field should be included in the signature generated by the developer.

Table 35-21. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
HEADER	0	16	Must be byte stream 06000000E10000000000010000000000H	Y
VENDOR	16	4	Intel Enclave: 00008086H Non-Intel Enclave: 00000000H	Y
DATE	20	4	Build date is yyyymmdd in hex: yyyy=4 digit year, mm=1-12, dd=1-31	Y
HEADER2	24	16	Must be byte stream 01010000600000006000000001000000H	Y
SWDEFINED	40	4	Available for software use.	Y

Table 35-21. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
RESERVED	44	84	Must be zero.	Y
MODULUS	128	384	Module Public Key (keylength=3072 bits).	N
EXPONENT	512	4	RSA Exponent = 3.	N
SIGNATURE	516	384	Signature over Header and Body.	N
MISCSELECT[1]	900	4	Bit vector specifying Extended SSA frame feature set to be used.	Y
MISCMASK	904	4	Bit vector mask of MISCSELECT to enforce.	Y
CET_ATTRIBUTES	908	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Enclave CET attributes that must be set. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.	Y
CET_ATTRIBUTES_MASK	909	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Mask of CET attributes to enforce. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.	Y
RESERVED	910	2	Must be zero.	Y
ISVFAMILYID	912	16	ISV assigned Product Family ID.	Y
ATTRIBUTES	928	16	Enclave Attributes that must be set.	Y
ATTRIBUTEMASK	944	16	Mask of Attributes to enforce.	Y
ENCLAVEHASH	960	32	MRENCLAVE of enclave this structure applies to.	Y
RESERVED	992	16	Must be zero.	Y
ISVEXTPRODID	1008	16	ISV assigned extended Product ID.	Y
ISVPRODID	1024	2	ISV assigned Product ID.	Y
ISVSVN	1026	2	ISV assigned SVN (security version number).	Y
RESERVED	1028	12	Must be zero.	N
Q1	1040	384	Q1 value for RSA Signature Verification.	N
Q2	1424	384	Q2 value for RSA Signature Verification.	N

NOTES:

1. If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISCSELECT must be 0.

35.15 EINIT TOKEN STRUCTURE (EINITTOKEN)

The EINIT token is used by EINIT to verify that the enclave is permitted to launch. EINIT token is generated by an enclave in possession of the EINITTOKEN key (the Launch Enclave).

EINIT token must be 512-Byte aligned.

Table 35-22. Layout of EINIT Token (EINITTOKEN)

Field	OFFSET (Bytes)	Size (Bytes)	MACed	Description
Valid	0	4	Y	Bit 0: 1: Valid; 0: Invalid. All other bits reserved.
RESERVED	4	44	Y	Must be zero.
ATTRIBUTES	48	16	Y	ATTRIBUTES of the Enclave.
MRENCLAVE	64	32	Y	MRENCLAVE of the Enclave.
RESERVED	96	32	Y	Reserved.
MRSIGNER	128	32	Y	MRSIGNER of the Enclave.
RESERVED	160	32	Y	Reserved.
CPUSVNLE	192	16	N	Launch Enclave's CPUSVN.
ISVPRODIDLE	208	02	N	Launch Enclave's ISVPRODID.
ISVSVNLE	210	02	N	Launch Enclave's ISVSVN.
CET_MASKED_ATTRIBUTES_LE	212	1	N	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Launch enclaves masked CET attributes. This should be set to LE's CET_ATTRIBUTES masked with CET_ATTTRIBUTES_MASK of the LE's KEYREQUEST. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved.
RESERVED	213	23	N	Reserved.
MASKEDMISCSELECTLE	236	4		Launch Enclave's MASKEDMISCSELECT: set by the LE to the resolved MISCSELECT value, used by EGETKEY (after applying KEYREQUEST's masking).
MASKEDATTRIBUTESLE	240	16	N	Launch Enclave's MASKEDATTRIBUTES: This should be set to the LE's ATTRIBUTES masked with ATTRIBUTEMASK of the LE's KEYREQUEST.
KEYID	256	32	N	Value for key wear-out protection.
MAC	288	16	N	Message Authentication Code on EINITTOKEN using EINITTOKEN_KEY.

35.16 REPORT (REPORT)

The REPORT structure is the output of the EREPORT instruction, and must be 512-Byte aligned.

Table 35-23. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CPUSVN	0	16	The security version number of the processor.
MISCSELECT	16	4	Bit vector specifying which extended features are saved to the MISC region of the SSA frame when an AEX occurs.
CET_ATTRIBUTES	20	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field reports the CET_ATTRIBUTES of the Enclave. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.
RESERVED	21	11	Zero.
ISVEXTNPRODID	32	16	The value of SECS.ISVEXTPRODID.
ATTRIBUTES	48	16	ATTRIBUTES of the Enclave. See Section 35.7.1.
MRENCLAVE	64	32	The value of SECS.MRENCLAVE.
RESERVED	96	32	Zero.
MRSIGNER	128	32	The value of SECS.MRSIGNER.
RESERVED	160	32	Zero.

Table 35-23. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CONFIGID	192	64	Value provided by SW to identify enclave's post EINIT configuration.
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Value provided by SW to indicate expected SVN of enclave's post EINIT configuration.
RESERVED	262	42	Zero.
ISVFAMILYID	304	16	The value of SECS.ISVFAMILYID.
REPORTDATA	320	64	Data provided by the user and protected by the REPORT's MAC, see Section 35.16.1.
KEYID	384	32	Value for key wear-out protection.
MAC	416	16	Message Authentication Code on the report using report key.

35.16.1 REPORTDATA

REPORTDATA is a 64-Byte data structure that is provided by the enclave and included in the REPORT. It can be used to securely pass information from the enclave to the target enclave.

35.17 REPORT TARGET INFO (TARGETINFO)

This structure is an input parameter to the EREPORT leaf function. The address of TARGETINFO is specified as an effective address in RBX. It is used to identify the target enclave which will be able to cryptographically verify the REPORT structure returned by EREPORT. TARGETINFO must be 512-Byte aligned.

Table 35-24. Layout of TARGETINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MEASUREMENT	0	32	The MRENCLAVE of the target enclave.
ATTRIBUTES	32	16	The ATTRIBUTES field of the target enclave.
CET_ATTRIBUTES	48	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the CET_ATTRIBUTES field of the target enclave. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved.
RESERVED	49	1	Must be zero.
CONFIGSVN	50	2	CONFIGSVN of the target enclave.
MISCSELECT	52	4	The MISCSELECT of the target enclave.
RESERVED	56	8	Must be zero.
CONFIGID	64	64	CONFIGID of target enclave.
RESERVED	128	384	Must be zero.

35.18 KEY REQUEST (KEYREQUEST)

This structure is an input parameter to the EGETKEY leaf function. It is passed in as an effective address in RBX and must be 512-Byte aligned. It is used for selecting the appropriate key and any additional parameters required in the derivation of that key.

Table 35-25. Layout of KEYREQUEST Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	2	Identifies the Key Required.
KEYPOLICY	2	2	Identifies which inputs are required to be used in the key derivation.
ISVSVN	4	2	The ISV security version number that will be used in the key derivation.
CET_ATTRIBUTES_MASK	6	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides a mask that defines which CET_ATTRIBUTES bits will be included in key derivation. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, then this field is reserved and must be 0.
RESERVED	7	1	Must be zero.
CPUSVN	8	16	The security version number of the processor used in the key derivation.
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in key derivation.
KEYID	40	32	Value for key wear-out protection.
MISCMASK	72	4	A mask defining which MISCSELECT bits will be included in key derivation.
CONFIGSVN	76	2	Identifies which enclave Configuration's Security Version should be used in key derivation.
RESERVED	78	434	

35.18.1 KEY REQUEST KeyNames

Table 35-26. Supported KEYName Values

Key Name	Value	Description
EINITTOKEN_KEY	0	EINIT_TOKEN key
PROVISION_KEY	1	Provisioning Key
PROVISION_SEAL_KEY	2	Provisioning Seal Key
REPORT_KEY	3	Report Key
SEAL_KEY	4	Seal Key
	All others	Reserved

35.18.2 Key Request Policy Structure

Table 35-27. Layout of KEYPOLICY Field

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register.
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register.
NOISVPRODID	2	If 1, derive key WITHOUT using the enclave' ISVPRODID value.
CONFIGID	3	If 1, derive key using the enclave's CONFIGID value.
ISVFAMILYID	4	If 1, derive key using the enclave ISVFAMILYID value.
ISVEXTPRODID	5	If 1, derive key using enclave's ISVEXTPRODID value.
RESERVED	15:6	Must be zero.

35.19 VERSION ARRAY (VA)

In order to securely store the versions of evicted EPC pages, Intel SGX defines a special EPC page type called a Version Array (VA). Each VA page contains 512 slots, each of which can contain an 8-byte version number for a page evicted from the EPC. When an EPC page is evicted, software chooses an empty slot in a VA page; this slot receives the unique version number of the page being evicted. When the EPC page is reloaded, there must be a VA slot that must hold the version of the page. If the page is successfully reloaded, the version in the VA slot is cleared.

VA pages can be evicted, just like any other EPC page. When evicting a VA page, a version slot in some other VA page must be used to hold the version for the VA being evicted. A Version Array Page must be 4K-Bytes aligned.

Table 35-28. Layout of Version Array Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
Slot 0	0	8	Version Slot 0
Slot 1	8	8	Version Slot 1
…			
Slot 511	4088	8	Version Slot 511

35.20 ENCLAVE PAGE CACHE MAP (EPCM)

EPCM is a secure structure used by the processor to track the contents of the EPC. The EPCM holds exactly one entry for each page that is currently loaded into the EPC. EPCM is not accessible by software, and the layout of EPCM fields is implementation specific.

Table 35-29. Content of an Enclave Page Cache Map Entry

Field	Description
VALID	Indicates whether the EPCM entry is valid.
R	Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.
W	Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.
X	Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.
PT	EPCM page type (PT_SECS, PT_TCS, PT_REG, PT_VA, PT_TRIM, PT_SS_FIRST, PT_SS_REST).
ENCLAVESECS	SECS identifier of the enclave to which the EPC page belongs.
ENCLAVEADDRESS	Linear enclave address of the EPC page.
BLOCKED	Indicates whether the EPC page is in the blocked state.
PENDING	Indicates whether the EPC page is in the pending state.
MODIFIED	Indicates whether the EPC page is in the modified state.
PR	Indicates whether the EPC page is in a permission restriction state.

35.21 READ INFO (RDINFO)

The RDINFO structure contains status information about an EPC page. It must be aligned to 32-Bytes.

Table 35-30. Layout of RDINFO Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
STATUS	0	8	Page status information.
FLAGS	8	8	EPCM state of the page.
ENCLAVECONTEXT	16	8	Context pointer describing the page's parent location.

35.21.1 RDINFO Status Structure

Table 35-31. Layout of RDINFO STATUS Structure

Field	Bit Position	Description
CHILDPRESENT	0	Indicates that the page has one or more child pages present (always zero for non-SECS pages). In VMX non-root operation includes the presence of virtual children.
VIRTCHLDPRESENT	1	Indicates that the page has one or more virtual child pages present (always zero for non-SECS pages). In VMX non-root operation this value is always zero.
RESERVED	63:2	

35.21.2 RDINFO Flags Structure

Table 35-32. Layout of RDINFO FLAGS Structure

Field	Bit Position	Description
R	0	Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.
W	1	Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.
X	2	Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.
PENDING	3	Indicates whether the EPC page is in the pending state.
MODIFIED	4	Indicates whether the EPC page is in the modified state.
PR	5	Indicates whether the EPC page is in a permission restriction state.
RESERVED	7:6	
PAGE_TYPE	15:8	Indicates the page type of the EPC page.
RESERVED	62:16	
BLOCKED	63	Indicates whether the EPC page is in the blocked state.

CHAPTER 36
ENCLAVE OPERATION

The following aspects of enclave operation are described in this chapter:

- Enclave creation: Includes loading code and data from outside of enclave into the EPC and establishing the enclave entity.
- Adding pages and measuring the enclave.
- Initialization of an enclave: Finalizes the cryptographic log and establishes the enclave identity and sealing identity.
- Enclave entry and exiting including:
  - — Controlled entry and exit.
  - — Asynchronous Enclave Exit (AEX) and resuming execution after an AEX.

36.1 CONSTRUCTING AN ENCLAVE

Figure 36-1 illustrates a typical Enclave memory layout.

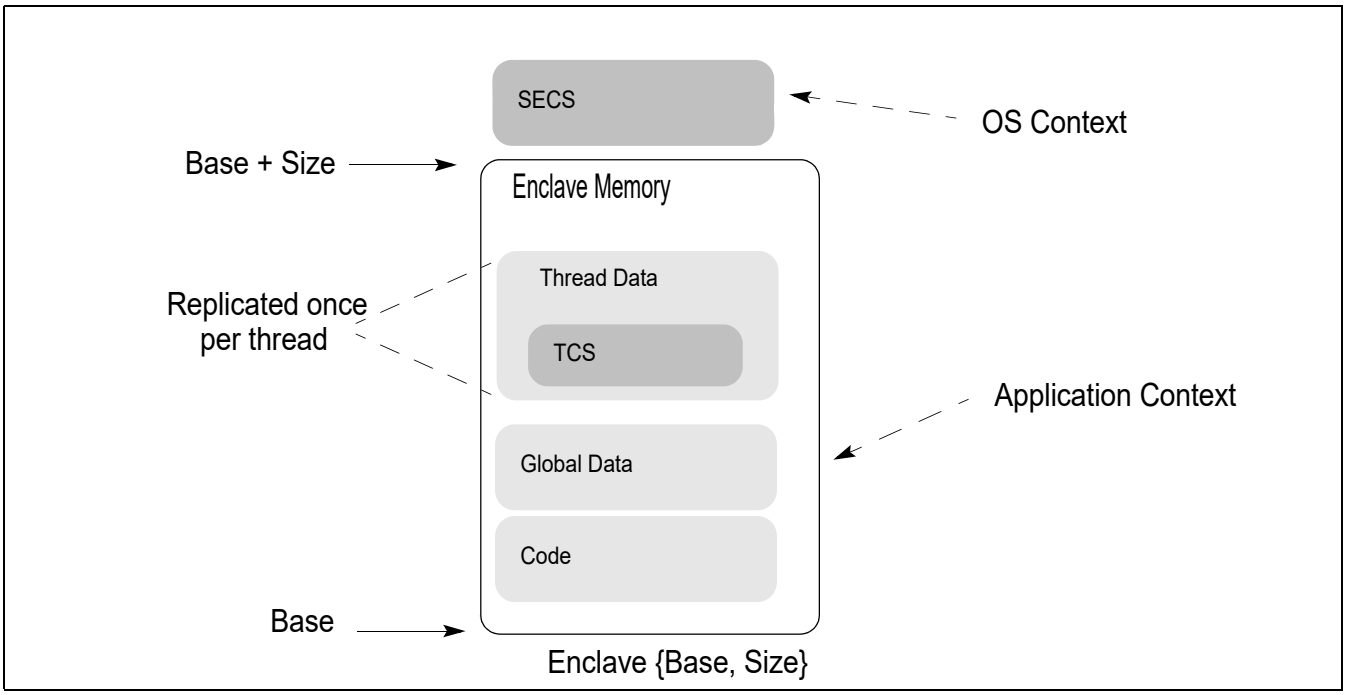

Figure 36-1. Enclave Memory Layout

The enclave creation, commitment of memory resources, and finalizing the enclave's identity with measurement comprises multiple phases. This process can be illustrated by the following exemplary steps:

1. The application hands over the enclave content along with additional information required by the enclave creation API to the enclave creation service running at privilege level 0.
2. The enclave creation service running at privilege level 0 uses the ECREATE leaf function to set up the initial environment, specifying base address and size of the enclave. This address range, the ELRANGE, is part of the application's address space. This reserves the memory range. The enclave will now reside in this address

region. ECREATE also allocates an Enclave Page Cache (EPC) page for the SGX Enclave Control Structure (SECS). Note that this page is not required to be a part of the enclave linear address space and is not required to be mapped into the process.

3. The enclave creation service uses the EADD leaf function to commit EPC pages to the enclave, and use EEXTEND to measure the committed memory content of the enclave. For each page to be added to the enclave:
   — Use EADD to add the new page to the enclave.
   — If the enclave developer requires measurement of the page as a proof for the content, use EEXTEND to add a measurement for 256 bytes of the page. Repeat this operation until the entire page is measured.
4. The enclave creation service uses the EINIT leaf function to complete the enclave creation process and finalize the enclave measurement to establish the enclave identity. Until an EINIT is executed, the enclave is not permitted to execute any enclave code (i.e., entering the enclave by executing EENTER would result in a fault).

36.1.1 ECREATE

The ECREATE leaf function sets up the initial environment for the enclave by reading an SGX Enclave Control Structure (SECS) that contains the enclave's address range (ELRANGE) as defined by BASEADDR and SIZE, the ATTRIBUTES and MISCSELECT bitmaps, and the SSAFRAMESIZE. It then securely stores this information in an Enclave Page Cache (EPC) page. ELRANGE is part of the application's address space. ECREATE also initializes a cryptographic log of the enclave's build process.

36.1.2 EADD and EEXTEND Interaction

Once the SECS has been created, enclave pages can be added to the enclave via EADD. This involves converting a free EPC page into either a PT_REG or a PT_TCS page.

When EADD is invoked, the processor will update the EPCM entry with the type of page (PT_REG or PT_TCS), the linear address used by the enclave to access the page, and the enclave access permissions for the page. It associates the page to the SECS provided as input. The EPCM entry information is used by hardware to manage access control to the page. EADD records EPCM information in the cryptographic log stored in the SECS and copies 4 KBytes of data from unprotected memory outside the EPC to the allocated EPC page.

System software is responsible for selecting a free EPC page. System software is also responsible for providing the type of page to be added, the attributes of the page, the contents of the page, and the SECS (enclave) to which the page is to be added as requested by the application. Incorrect data would lead to a failure of EADD or to an incorrect cryptographic log and a failure at EINIT time.

After a page has been added to an enclave, software can measure a 256 byte region as determined by the developer by invoking EEXTEND. Thus to measure an entire 4KB page, system software must execute EEXTEND 16 times. Each invocation of EEXTEND adds to the cryptographic log information about which region is being measured and the measurement of the section.

Entries in the cryptographic log define the measurement of the enclave and are critical in gaining assurance that the enclave was correctly constructed by the untrusted system software.

36.1.3 EINIT Interaction

Once system software has completed the process of adding and measuring pages, the enclave needs to be initialized by the EINIT leaf function. After an enclave is initialized, EADD and EEXTEND are disabled for that enclave (An attempt to execute EADD/EEXTEND to enclave after enclave initialization will result in a fault). The initialization process finalizes the cryptographic log and establishes the **enclave identity** and **sealing identity** used by EGETKEY and EREPORT.

A cryptographic hash of the log is stored as the **enclave identity**. Correct construction of the enclave results in the cryptographic hash matching the one built by the enclave owner and included as the ENCLAVEHASH field of SIGSTRUCT. The **enclave identity** provided by the EREPORT leaf function can be verified by a remote party.

The EINIT leaf function checks the EINIT token to validate that the enclave has been enabled on this platform. If the enclave is not correctly constructed, or the EINIT token is not valid for the platform, or SIGSTRUCT isn't properly signed, then EINIT will fail. See the EINIT leaf function for details on the error reporting.

The **enclave identity** is a cryptographic hash that reflects the enclave attributes and MISCSELECT value, content of the enclave, the order in which it was built, the addresses it occupies in memory, the security attributes, and access right permissions of each page. The **enclave identity** is established by the EINIT leaf function.

The **sealing identity** is managed by a sealing authority represented by the hash of the public key used to sign the SIGSTRUCT structure processed by EINIT. The sealing authority assigns a product ID (ISVPRODID) and security version number (ISVSVN) to a particular enclave identity.

EINIT establishes the sealing identity using the following steps:

1. Verifies that SIGSTRUCT is properly signed using the public key enclosed in the SIGSTRUCT.
2. Checks that the measurement of the enclave matches the measurement of the enclave specified in SIGSTRUCT.
3. Checks that the enclave's attributes and MISCSELECT values are compatible with those specified in SIGSTRUCT.
4. Finalizes the measurement of the enclave and records the **sealing identity** (the sealing authority, product id and security version number) and **enclave identity** in the SECS.
5. Sets the ATTRIBUTES.INIT bit for the enclave.

36.1.4 Intel® SGX Launch Control Configuration

Intel® SGX Launch Control is a set of controls that govern the creation of enclaves. Before the EINIT leaf function will successfully initialize an enclave, a designated Launch Enclave must create an EINITTOKEN for that enclave. Launch Enclaves have SECS.ATTRIBUTES.EINITTOKEN_KEY = 1, granting them access to the EINITTOKEN_KEY from the EGETKEY leaf function. EINITTOKEN_KEY must be used by the Launch Enclave when computing EINITTOKEN.MAC, the Message Authentication Code of the EINITTOKEN.

The hash of the public key used to sign the SIGSTRUCT of the Launch Enclave must equal the value in the IA32_SGXLEPUBKEYHASH MSRs. Only Launch Enclaves are allowed to launch without a valid token.

The IA32_SGXLEPUBKEYHASH MSRs are provided to designate the platform's Launch Enclave. IA32_SGXLEPUBKEYHASH defaults to digest of Intel's launch enclave signing key after reset.

IA32_FEATURE_CONTROL bit 17 controls the permissions on the IA32_SGXLEPUBKEYHASH MSRs when CPUID.(EAX=12H, ECX=00H):EAX[0] = 1. If IA32_FEATURE_CONTROL is locked with bit 17 set, IA32_SGXLEPUBKEYHASH MSRs are reconfigurable (writeable). If either IA32_FEATURE_CONTROL is not locked or bit 17 is clear, the MSRs are read only. By leaving these MSRs writable, system SW or a VMM can support a plurality of Launch Enclaves for hosting multiple execution environments. See Table 40.2.2 for more details.

36.2 ENCLAVE ENTRY AND EXITING

36.2.1 Controlled Entry and Exit

The EENTER leaf function is the method to enter the enclave under program control. To execute EENTER, software must supply an address of a TCS that is part of the enclave to be entered. The TCS holds the location inside the enclave to transfer control to and a pointer to the SSA frame inside the enclave that an AEX should store the register state to.

When a logical processor enters an enclave, the TCS is considered busy until the logical processors exits the enclave. An attempt to enter an enclave through a busy TCS results in a fault. Intel® SGX allows an enclave builder to define multiple TCSs, thereby providing support for multithreaded enclaves.

Software must also supply to EENTER the Asynchronous Exit Pointer (AEP) parameter. AEP is an address external to the enclave which an exception handler will return to using IRET. Typically the location would contain the ERESUME instruction. ERESUME transfers control back to the enclave, to the address retrieved from the enclave thread's saved state.

EENTER performs the following operations:

1. Check that TCS is not busy and flush all cached linear-to-physical mappings.
2. Change the mode of operation to be in enclave mode.
3. Save the old RSP, RBP for later restore on AEX (Software is responsible for setting up the new RSP, RBP to be used inside enclave).
4. Save XCR0 and replace it with the XFRM value for the enclave.
5. Check if software wishes to debug (applicable to a debuggable enclave):
   — If not debugging, then configure hardware so the enclave appears as a single instruction.
   — If debugging, then configure hardware to allow traps, breakpoints, and single steps inside the enclave.
6. Set the TCS as busy.
7. Transfer control from outside enclave to predetermined location inside the enclave specified by the TCS.

The EEXIT leaf function is the method of leaving the enclave under program control. EEXIT receives the target address outside of the enclave that the enclave wishes to transfer control to. It is the responsibility of enclave software to erase any secret from the registers prior to invoking EEXIT. To allow enclave software to easily perform an external function call and re-enter the enclave (using EEXIT and EENTER leaf functions), EEXIT returns the value of the AEP that was used when the enclave was entered.

EEXIT performs the following operations:

1. Clear enclave mode and flush all cached linear-to-physical mappings.
2. Mark TCS as not busy.
3. Transfer control from inside the enclave to a location on the outside specified as parameter to the EEXIT leaf function.

36.2.2 Asynchronous Enclave Exit (AEX)

Asynchronous and synchronous events, such as exceptions, interrupts, traps, SMIs, and VM exits may occur while executing inside an enclave. These events are referred to as Enclave Exiting Events (EEE). Upon an EEE, the processor state is securely saved inside the enclave (in the thread's current SSA frame) and then replaced by a synthetic state to prevent leakage of secrets. The process of securely saving state and establishing the synthetic state is called an Asynchronous Enclave Exit (AEX). Details of AEX is described in Chapter 37, "Enclave Exiting Events."

As part of most EEEs, the AEP is pushed onto the stack as the location of the eventing address. This is the location where control will return to after executing the IRET. The ERESUME leaf function can be executed from that point to reenter the enclave and resume execution from the interrupted point.

After AEX has completed, the logical processor is no longer in enclave mode and the exiting event is processed normally. Any new events that occur after the AEX has completed are treated as having occurred outside the enclave (e.g., a #PF in dispatching to an interrupt handler).

36.2.3 Resuming Execution After AEX

After system software has serviced the event that caused the logical processor to exit an enclave, the logical processor can continue enclave execution using ERESUME. ERESUME restores processor state and returns control to where execution was interrupted.

If the cause of the exit was an exception or a fault and was not resolved, the event will be triggered again if the enclave is re-entered using ERESUME. For example, if an enclave performs a divide by 0 operation, executing ERESUME will cause the enclave to attempt to re-execute the faulting instruction and result in another divide by 0 exception. Intel® SGX provides the means for an enclave developer to handle enclave exceptions from within the enclave. Software can enter the enclave at a different location and invoke the exception handler within the enclave by executing the EENTER leaf function. The exception handler within the enclave can read the fault information from the SSA frame and attempt to resolve the faulting condition or simply return and indicate to software that the enclave should be terminated (e.g., using EEXIT).

36.2.3.1 ERESUME Interaction

ERESUME restores registers depending on the mode of the enclave (32 or 64 bit).

- In 32-bit mode (IA32_EFER.LMA = 0 || CS.L = 0), the low 32-bits of the legacy registers (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP, and EFLAGS) are restored from the thread's GPR area of the current SSA frame. Neither the upper 32 bits of the legacy registers nor the 64-bit registers (R8 … R15) are loaded.
- In 64-bit mode (IA32_EFER.LMA = 1 && CS.L = 1), all 64 bits of the general processor registers (RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8 … R15, RIP, and RFLAGS) are loaded.

Extended features specified by SECS.ATTRIBUTES.XFRM are restored from the XSAVE area of the current SSA frame. The layout of the x87 area depends on the current values of IA32_EFER.LMA and CS.L:

- IA32_EFER.LMA = 0 || CS.L = 0
  - — 32-bit load in the same format that XSAVE/FXSAVE uses with these values.
- IA32_EFER.LMA = 1 && CS.L = 1
  - — 64-bit load in the same format that XSAVE/FXSAVE uses with these values as if REX.W = 1.

36.2.3.2 Asynchronous Enclave Exit Notify and EDECCSSA

Asynchronous Enclave Exit Notify (AEX-Notify) is an extension to Intel SGX that allows Intel SGX enclaves to be notified after an asynchronous enclave exit (AEX) has occurred. EDECCSSA is a new Intel SGX user leaf function (ENCLU[EDECCSSA]) that can facilitate AEX notification handling, as well as software exception handling. This section provides information about changes to the Intel SGX architecture that support AEX-Notify and ENCLU[EDECCSSA].

NOTE

On some platforms, AEX-Notify and the EDECCSSA user leaf function may be enumerated by CPUID following a microcode update.

The following list summarizes the additions to existing Intel SGX data structures to support AEX-Notify:

- SECS.ATTRIBUTES.AEXNOTIFY: This enclave supports AEX-Notify.
- TCS.FLAGS.AEXNOTIFY: This enclave thread may receive AEX notifications.
- SSA.GPRSGX.AEXNOTIFY: Enclave-writable byte that allows enclave software to dynamically enable/disable AEX notifications.

An AEX notification is delivered by ENCLU[ERESUME] when the following conditions are met:

1. TCS.FLAGS.AEXNOTIFY is set.
2. TCS.CSSA (the current slot index of an SSA frame) is greater than zero.
3. TCS.SSA[TCS.CSSA-1].GPRSGX.AEXNOTIFY[0] is set.

Note that AEX increments TCS.CSSA, and ENCLU[ERESUME] decrements TCS.CSSA, except when an AEX notification is delivered. Instead of decrementing TCS.CSSA and restoring state from the SSA, ENCLU[ERESUME] delivers an AEX notification by behaving as ENCLU[EENTER]. Implications of this behavior include:

- The enclave thread is resumed at EnclaveBase + TCS.OENTRY.
- EAX contains the (non-decremented) value of TCS.CSSA.
- RCX contains the address of the IP following ENCLU[ERESUME].
- The architectural state saved by the most recent AEX is preserved in TCS.SSA[TCS.CSSA-1].

The enclave thread can return to the previous SSA context by invoking ENCLU[EDECCSSA], which decrements TCS.CSSA.

36.3 CALLING ENCLAVE PROCEDURES

36.3.1 Calling Convention

In standard call conventions subroutine parameters are generally pushed onto the stack. The called routine, being aware of its own stack layout, knows how to find parameters based on compile-time-computable offsets from the SP or BP register (depending on runtime conventions used by the compiler).

Because of the stack switch when calling an enclave, stack-located parameters cannot be found in this manner. Entering the enclave requires a modified parameter passing convention.

For example, the caller might push parameters onto the untrusted stack and then pass a pointer to those parameters in RAX to the enclave software. The exact choice of calling conventions is up to the writer of the edge routines; be those routines hand-coded or compiler generated.

36.3.2 Register Preservation

As with most systems, it is the responsibility of the callee to preserve all registers except that used for returning a value. This is consistent with conventional usage and tends to optimize the number of register save/restore operations that need be performed. It has the additional security result that it ensures that data is scrubbed from any registers that were used by enclave to temporarily contain secrets.

36.3.3 Returning to Caller

No registers are modified during EEXIT. It is the responsibility of software to remove secrets in registers before executing EEXIT.

36.4 INTEL® SGX KEY AND ATTESTATION

36.4.1 Enclave Measurement and Identification

During the enclave build process, two “measurements” are taken of each enclave and are stored in two 256-bit Measurement Registers (MR): MRENCLAVE and MRSIGNER. MRENCLAVE represents the enclave's contents and build process. MRSIGNER represents the entity that signed the enclave's SIGSTRUCT.

The values of the Measurement Registers are included in attestations to identify the enclave to remote parties. The MRs are also included in most keys, binding keys to enclaves with specific MRs.

36.4.1.1 MRENCLAVE

MRENCLAVE is a unique 256 bit value that identifies the code and data that was loaded into the enclave during the initial launch. It is computed as a SHA256 hash that is initialized by the ECREATE leaf function. EADD and EEXTEND leaf functions record information about each page and the content of those pages. The EINIT leaf function finalizes the hash, which is stored in SECS.MRENCLAVE. Any tampering with the build process, contents of a page, page permissions, etc will result in a different MRENCLAVE value.

Figure 36-2 illustrates a simplified flow of changes to the MRENCLAVE register when building an enclave:

- Enclave creation with ECREATE.
- Copying a non-enclave source page into the EPC of an un-initialized enclave with EADD.
- Updating twice of the MRENCLAVE after modifying the enclave’s page content, i.e., EEXTEND twice.
- Finalizing the enclave build with EINIT.

Details on specific values inserted in the hash are available in the individual instruction definitions.

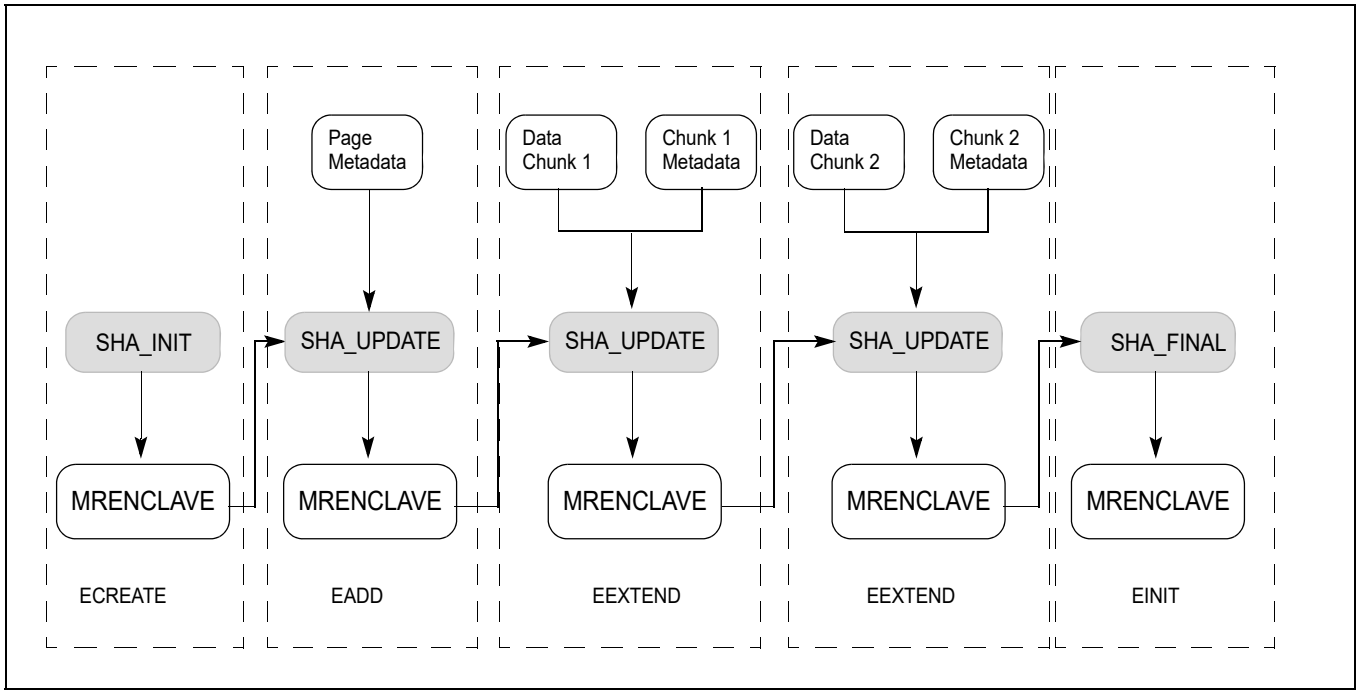

Figure 36-2. Measurement Flow of Enclave Build Process

36.4.1.2 MRSIGNER

Each enclave is signed using a 3072 bit RSA key. The signature is stored in the SIGSTRUCT. In the SIGSTRUCT, the enclave's signer also assigns a product ID (ISVPRODID) and a security version (ISVSVN) to the enclave. MRSIGNER is the SHA-256 hash of the signer's public key. For platforms that support Key Separation and Sharing (CPUID.(EAX=12H, ECX=1).EAX.KSS[7]) the SIGSTRUCT can additionally specify an 16 byte extended product ID (ISVEXTPRODID), and a 16 byte family ID (ISVFAMILYID).

In attestation, MRSIGNER can be used to allow software to approve of an enclave based on the author rather than maintaining a list of MRENCLAVEs. It is used in key derivation to allow software to create a lineage of an application. By signing multiple enclaves with the same key, the enclaves will share the same keys and data. Combined with security version numbering, the author can release multiple versions of an application which can access keys for previous versions, but not future versions of that application.

36.4.1.3 CONFIGID

For platforms that support enhancements for key separation and sharing (CPUID.(EAX=12H, ECX=1).EAX.KSS[7]) when the enclave is created the platform can additionally provide 32-byte configuration identifier (CONFIGID). How this value is used is dependent on the enclave but it is intended to allow enclave creators to indicate what additional content may be accepted by the enclave post-initialization.

36.4.2 Security Version Numbers (SVN)

Intel® SGX supports a versioning system that allows the signer to identify different versions of the same software released by an author. The security version is independent of the functional version an author uses and is intended to specify security equivalence. Multiple releases with functional enhancements may all share the same SVN if they all have the same security properties or posture. Each enclave has an SVN and the underlying hardware has an SVN.

The SVNs are attested to in EREPORT and are included in the derivation of most keys, thus providing separation between data for older/newer versions.

36.4.2.1 Enclave Security Version

In the SIGSTRUCT, the MRSIGNER is associated with a 16-bit Product ID (ISVPRODID) and a 16 bit integer SVN (ISVSVN). Together they define a specific group of versions of a specific product. Most keys, including the Seal Key, can be bound to this pair.

To support upgrading from one release to another, EGETKEY will return keys corresponding to any value less than or equal to the software's ISVSVN.

36.4.2.2 Hardware Security Version

CPUSVN is a 128 bit value that reflects the microcode update version and authenticated code modules supported by the processor. Unlike ISVSVN, CPUSVN is not an integer and cannot be compared mathematically. Not all values are valid CPUSVNs.

Software must ensure that the CPUSVN provided to EGETKEY is valid. EREPORT will return the CPUSVN of the current environment. Software can execute EREPORT with TARGETINFO set to zeros to retrieve a CPUSVN from REPORTDATA. Software can access keys for a CPUSVN recorded previously, provided that each of the elements reflected in CPUSVN are the same or have been upgraded.

36.4.2.3 CONFIGID Security Version

The CONFIGID field can be used to contain the hash of a signing key for verifying the additional content. In this case, similar to the relationship between MRSIGNER and ISVSVN, CONFIGID needs a CONFIGID Security Version Number. CONFIGIDSVN can be specified at the same time as CONFIGID.

36.4.3 Keys

Intel® SGX provides software with access to keys unique to each processor and rooted in HW keys inserted into the processor during manufacturing.

Each enclave requests keys using the EGETKEY leaf function. The key is based on enclave parameters such as measurement, the enclave signing key, security attributes of the enclave, and the Hardware Security version of the processor itself. A full list of parameter options is specified in the KEYREQUEST structure, see details in Section 35.18.

By deriving keys using enclave properties, SGX guarantees that if two enclaves call EGETKEY, they will receive a unique key only accessible by the respective enclave. It also guarantees that the enclave will receive the same key on every future execution of EGETKEY. Some parameters are optional or configurable by software. For example, a Seal key can be based on the signer of the enclave, resulting in a key available to multiple enclaves signed by the same party.

The EGETKEY leaf function provides several key types. Each key is specific to the processor, CPUSVN, and the enclave that executed EGETKEY. The EGETKEY instruction definition details how each of these keys is derived, see Table 38-66. Additionally,

- SEAL Key: The Seal key is a general purpose key for the enclave to use to protect secrets. Typical uses of the Seal key are encrypting and calculating MAC of secrets on disk. There are 2 types of Seal Key described in Section 36.4.3.1.
- REPORT Key: This key is used to compute the MAC on the REPORT structure. The EREPORT leaf function is used to compute this MAC, and destination enclave uses the Report key to verify the MAC. The software usage flow is detailed in Section 36.4.3.2.
- EINITTOKEN_KEY: This key is used by Launch Enclaves to compute the MAC on EINITTOKENs. These tokens are then verified in the EINIT leaf function. The key is only available to enclaves with ATTRIBUTE.EINITTOKEN_KEY set to 1.
- PROVISIONING Key and PROVISIONING SEAL Key: These keys are used by attestation key provisioning software to prove to remote parties that the processor is genuine and identify the currently executing TCB. These keys are only available to enclaves with ATTRIBUTE.PROVISIONKEY set to 1.

36.4.3.1 Sealing Enclave Data

Enclaves can protect persistent data using Seal keys to provide encryption and/or integrity protection. EGETKEY provides two types of Seal keys specified in KEYREQUEST.KEYPOLICY field: MRENCLAVE-based key and MRSIGNER-based key.

The MRENCLAVE-based keys are available only to enclave instances sharing the same MRENCLAVE. If a new version of the enclave is released, the Seal keys will be different. Retrieving previous data requires additional software support.

The MRSIGNER-based keys are bound to the 3 tuple (MRSIGNER, ISVPRODID, ISVSVN). These keys are available to any enclave with the same MRSIGNER and ISVPRODID and an ISVSVN equal to or greater than the key in questions. This is valuable for allowing new versions of the same software to retrieve keys created before an upgrade.

For platforms that support enhancements for key separation and sharing (CPUID.(EAX=12H, ECX=1).EAX.KSS[7]) four additional key policies for seal key derivation are provided. These add the ISVEXTPRODID, ISVFAMILYID, and CONFIGID/CONFIGSVN to the key derivation. Additionally, there is a policy to remove ISVPRODID from a key derivation to create a shared between different products that share the same MRSIGNER.

36.4.3.2 Using REPORTs for Local Attestation

Intel SGX provides a means for enclaves to securely identify one another, this is referred to as "Local Attestation". SGX provides a hardware assertion, REPORT that contains calling enclaves Attributes, Measurements and User supplied data (described in detail in Section 35.16). Figure 36-3 shows the basic flow of information.

1. The source enclave determines the identity of the target enclave to populate TARGETINFO.
2. The source enclave calls EREPORT instruction to generate a REPORT structure. The EREPORT instruction conducts the following:
   — Populates the REPORT with identify information about the calling enclave.
   — Derives the Report Key that is returned when the target enclave executes the EGETKEY. TARGETINFO provides information about the target.
   — Computes a MAC over the REPORT using derived target enclave Report Key.
3. Non-enclave software copies the REPORT from source to destination.
4. The target enclave executes the EGETKEY instruction to request its REPORT key, which is the same key used by EREPORT at the source.
5. The target enclave verifies the MAC and can then inspect the REPORT to identify the source.

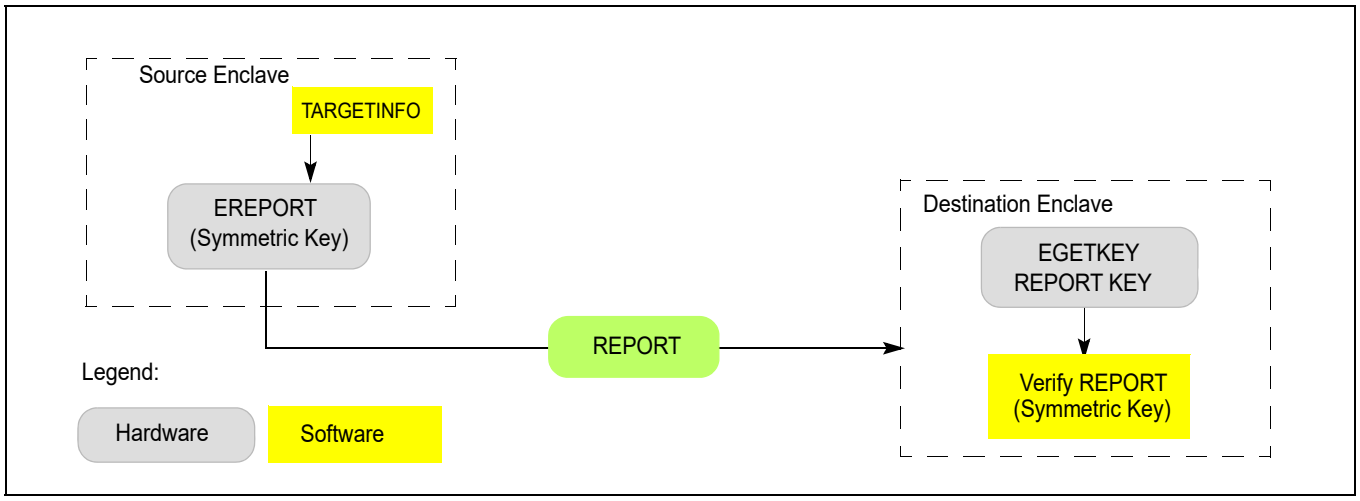

Figure 36-3. SGX Local Attestation

36.5 EPC AND MANAGEMENT OF EPC PAGES

EPC layout is implementation specific, and is enumerated through CPUID (see Table 34-7 for EPC layout). EPC is typically configured by BIOS at system boot time.

36.5.1 EPC Implementation

EPC must be properly protected against attacks. One example of EPC implementation could use a Memory Encryption Engine (MEE). An MEE provides a cost-effective mechanism of creating cryptographically protected volatile storage using platform DRAM. These units provide integrity, replay, and confidentiality protection. Details are implementation specific.

36.5.2 OS Management of EPC Pages

The EPC is a finite resource. SGX1 (i.e., CPUID.(EAX=12H, ECX=0):EAX.SGX1 = 1 but CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 0) provides the EPC manager with leaf functions to manage this resource and properly swap pages out of and into the EPC. For that, the EPC manager would need to keep track of all EPC entries, type and state, context affiliation, and SECS affiliation.

Enclave pages that are candidates for eviction should be moved to BLOCKED state using EBLOCK instruction that ensures no new cached virtual to physical address mappings can be created by attempts to reference a BLOCKED page.

Before evicting blocked pages, EPC manager should execute ETRACK leaf function on that enclave and ensure that there are no stale cached virtual to physical address mappings for the blocked pages remain on any thread on the platform.

After removing all stale translations from blocked pages, system software should use the EWB leaf function for securely evicting pages out of the EPC. EWB encrypts a page in the EPC, writes it to unprotected memory, and invalidates the copy in EPC. In addition, EWB also creates a cryptographic MAC (PCMD.MAC) of the page and stores it in unprotected memory. A page can be reloaded back to the processor only if the data and MAC match. To ensure that only the latest version of the evicted page can be loaded back, the version of the evicted page is stored securely in a Version Array (VA) in EPC.

SGX1 includes two instructions for reloading pages that have been evicted by system software: ELDU and ELDB. The difference between the two instructions is the value of the paging state at the end of the instruction. ELDU results in a page being reloaded and set to an UNBLOCKED state, while ELDB results in a page loaded to a BLOCKED state.

ELDB is intended for use by a Virtual Machine Monitor (VMM). When a VMM reloads an evicted page, it needs to restore it to the correct state of the page (BLOCKED vs. UNBLOCKED) as it existed at the time the page was evicted. Based on the state of the page at eviction, the VMM chooses either ELDB or ELDU.

36.5.2.1 Enhancement to Managing EPC Pages

On processors supporting SGX2 (i.e., CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 1), the EPC manager can manage EPC resources (while enclave is running) with more flexibility provided by the SGX2 leaf functions. The additional flexibility is described in Section 36.5.7 through Section 36.5.11.

36.5.3 Eviction of Enclave Pages

Intel SGX paging is optimized to allow the Operating System (OS) to evict multiple pages out of the EPC under a single synchronization.

The suggested flow for evicting a list of pages from the EPC is:

1. For each page to be evicted from the EPC:
   a. Select an empty slot in a Version Array (VA) page.
      - If no empty VA page slots exist, create a new VA page using the EPA leaf function.

b. Remove linear-address to physical-address mapping from the enclave context's mapping tables (page table and EPT tables).

c. Execute the EBLOCK leaf function for the target page. This sets the target page state to BLOCKED. At this point no new mappings of the page will be created. So any access which does not have the mapping cached in the TLB will generate a #PF.

2. For each enclave containing pages selected in step 1:

   — Execute an ETRACK leaf function pointing to that enclave's SECS. This initiates the tracking process that ensures that all caching of linear-address to physical-address translations for the blocked pages is cleared.

3. For all logical processors executing in processes (OS) or guests (VMM) that contain the enclaves selected in step 1:

   — Issue an IPI (inter-processor interrupt) to those threads. This causes those logical processors to asynchronously exit any enclaves they might be in, and as a result flush cached linear-address to physical-address translations that might hold stale translations to blocked pages. There is no need for additional measures such as performing a "TLB shootdown".

4. After enclaves exit, allow logical processors to resume normal operation, including enclave re-entry as the tracking logic keeps track of the activity.

5. For each page to be evicted:

   — Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents, and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

At this point, system software has the only copy of each page data encrypted with its page metadata in main memory.

36.5.4 Loading an Enclave Page

To reload a previously evicted page, system software needs four elements: the VA slot used when the page was evicted, a buffer containing the encrypted page contents, a buffer containing the page metadata, and the parent SECS to associate this page with. If the VA page or the parent SECS are not already in the EPC, they must be reloaded first.

1. Execute ELDB/ELDU (depending on the desired BLOCKED state for the page), passing as parameters: the EPC page linear address, the VA slot, the encrypted page, and the page metadata.
2. Create a mapping in the enclave context's mapping tables (page tables and EPT tables) to allow the application to access that page (OS: system page table; VMM: EPT).

The ELDB/ELDU instruction marks the VA slot empty so that the page cannot be replayed at a later date.

36.5.5 Eviction of an SECS Page

The eviction of an SECS page is similar to the eviction of an enclave page. The only difference is that an SECS page cannot be evicted until all other pages belonging to the enclave have been evicted. Since all other pages have been evicted, there will be no threads executing inside the enclave and tracking with ETRACK isn't necessary. When reloading an enclave, the SECS page must be reloaded before all other constituent pages.

1. Ensure all pages are evicted from enclave.
2. Select an empty slot in a Version Array page.

   — If no VA page exists with an empty slot, create a new one using the EPA function leaf.

3. Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

36.5.6 Eviction of a Version Array Page

VA pages do not belong to any enclave and tracking with ETRACK isn't necessary. When evicting the VA page, a slot in a different VA page must be specified in order to provide versioning of the evicted VA page.

1. Select a slot in a Version Array page other than the page being evicted.
   — If no VA page exists with an empty slot, create a new one using the EPA leaf function.
2. Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents, and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

36.5.7 Allocating a Regular Page

On processors that support SGX2, allocating a new page to an already initialized enclave is accomplished by invoking the EAUG leaf function. Typically, the enclave requests that the OS allocates a new page at a particular location within the enclave's address space. Once allocated, the page remains in a pending state until the enclave executes the corresponding EACCEPT leaf function to accept the new page into the enclave. Page allocation operations may be batched to improve efficiency.

The typical process for allocating a regular page is as follows:

1. Enclave requests additional memory from OS when the current allocation becomes insufficient.
2. The OS invokes the EAUG leaf function to add a new memory page to the enclave.
   a. EAUG may only be called on a free EPC page.
   b. Successful completion of the EAUG instruction places the target page in the VALID and PENDING state.
   c. All dynamically created pages have the type PT_REG and content of all zeros.
3. The OS maps the page in the enclave context's mapping tables.
4. The enclave issues an EACCEPT instruction, which verifies the page's attributes and clears the PENDING state. At that point the page becomes accessible for normal enclave use.

36.5.8 Allocating a TCS Page

On processors that support SGX2, allocating a new TCS page to an already initialized enclave is a two-step process. First the OS allocates a regular page with a call to EAUG. This page must then be accepted and initialized by the enclave to which it belongs. Once the page has been initialized with appropriate values for a TCS page, the enclave requests the OS to change the page's type to PT_TCS. This change must also be accepted. As with allocating a regular page, TCS allocation operations may be batched.

A typical process for allocating a TCS page is as follows:

1. Enclave requests an additional page from the OS.
2. The OS invokes EAUG to add a new regular memory page to the enclave.
   a. EAUG may only be called on a free EPC page.
   b. Successful completion of the EAUG instruction places the target page in the VALID and PENDING state.
3. The OS maps the page in the enclave context's mapping tables.
4. The enclave issues an EACCEPT instruction, at which point the page becomes accessible for normal enclave use.
5. The enclave initializes the contents of the new page.
6. The enclave requests that the OS convert the page from type PT_REG to PT_TCS.
7. OS issues an EMODT instruction on the page.
   a. The parameters to EMODT indicate that the regular page should be converted into a TCS.

   b. EMODT forces all access rights to a page to be removed because TCS pages may not be accessed by enclave code.

8. The enclave issues an EACCEPT instruction to confirm the requested modification.

36.5.9 Trimming a Page

On processors that support SGX2, Intel SGX supports the trimming of an enclave page as a special case of EMODT. Trimming allows an enclave to actively participate in the process of removing a page from the enclave (deallocation) by splitting the process into first removing it from the enclave's access and then removing it from the EPC using the EREMOVE leaf function. The page type PT_TRIM indicates that a page has been trimmed from the enclave's address space and that the page is no longer accessible to enclave software. Modifications to a page in the PT_TRIM state are not permitted; the page must be removed and then reallocated by the OS before the enclave may use the page again. Page deallocation operations may be batched to improve efficiency.

The typical process for trimming a page from an enclave is as follows:

1. Enclave signals OS that a particular page is no longer in use.
2. OS invokes the EMODT leaf function on the page, requesting that the page's type be changed to PT_TRIM.
   a. SECS and VA pages cannot be trimmed in this way, so the initial type of the page must be PT_REG or PT_TCS.
   b. EMODT may only be called on valid enclave pages.
3. OS invokes the ETRACK leaf function on the enclave containing the page to track removal the TLB addresses from all the processors.
4. Issue an IPI (inter-processor interrupt) to flush the stale linear-address to physical-address translations for all logical processors executing in processes that contain the enclave.
5. Enclave issues an EACCEPT leaf function.
6. The OS may now permanently remove the page from the EPC (by issuing EREMOVE).

36.5.10 Restricting the EPCM Permissions of a Page

On processors that support SGX2, restricting the EPCM permissions associated with an enclave page is accomplished using the EMODPR leaf function. This operation requires the cooperation of the OS to flush stale entries to the page and to update the page-table permissions of the page to match. Permissions restriction operations may be batched.

The typical process for restricting the permissions of an enclave page is as follows:

1. Enclave requests that the OS to restrict the permissions of an EPC page.
2. OS performs permission restriction, flushing cached linear-address to physical-address translations, and page-table modifications.
   a. Invokes the EMODPR leaf function to restrict permissions (EMODPR may only be called on VALID pages).
   b. Invokes the ETRACK leaf function on the enclave containing the page to track removal of the TLB addresses from all the processor.
   c. Issue an IPI (inter-processor interrupt) to flush the stale linear-address to physical-address translations for all logical processors executing in processes that contain the enclave.
   d. Sends IPIs to trigger enclave thread exit and TLB shootdown.
   e. OS informs the Enclave that all logical processors should now see the new restricted permissions.
3. Enclave invokes the EACCEPT leaf function.
   a. Enclave may access the page throughout the entire process.
   b. Successful call to EACCEPT guarantees that no stale cached linear-address to physical-address translations are present.

36.5.11 Extending the EPCM Permissions of a Page

On processors that support SGX2, extending the EPCM permissions associated with an enclave page is accomplished directly by the enclave using the EMODPE leaf function. After performing the EPCM permission extension, the enclave requests the OS to update the page table permissions to match the extended permission. Security wise, permission extension does not require enclave threads to leave the enclave as TLBs with stale references to the more restrictive permissions will be flushed on demand, but to allow forward progress, an OS needs to be aware that an application might signal a page fault.

The typical process for extending the permissions of an enclave page is as follows:

1. Enclave invokes EMODPE to extend the EPCM permissions associated with an EPC page (EMODPE may only be called on VALID pages).
2. Enclave requests that OS update the page tables to match the new EPCM permissions.
3. Enclave code resumes.
    a. If cached linear-address to physical-address translations are present to the more restrictive permissions, the enclave thread will page fault. The SGX2-aware OS will see that the page tables permit the access and resume the thread, which can now successfully access the page because exiting cleared the TLB.
    b. If cached linear-address to physical-address translations are not present, access to the page with the new permissions will succeed without an enclave exit.

36.5.12 VMM Oversubscription of EPC

On processors supporting oversubscription enhancements (i.e., CPUID.(EAX=12H, ECX=0):EAX[5]=1 & EAX[6] = 1) a Virtual Machine Monitor or other executive can more efficiently manage the EPC space available on the platform between virtualized entities. A typical process for using these instructions to support oversubscribing the physical EPC space on the platform is as follows:

1. VMM creates data structures for SECS tracking including a count of child pages.
2. VMM selects possible EPC victim pages.
3. VMM ages the victim pages. Some of the selected pages will be accessed by the guest. In this case the VMM will remove these pages from the victim pool and return them to the guest.
4. VMM makes remaining pages not present in EPT. It then issues IPI on each page to remove TLB mappings.
5. For every EPC victim page the VMM obtains the victim's SECS page info using ERDINFO.
    a. ENCLAVECONTEXT field in RDINFO structure will indicate the location of SECS, and the PAGE_TYPE field will indicate the page type.
    b. Child pages of SECS can be evicted.
    c. SECS pages may be evicted if the child count is zero.
    d. Some pages may be returned to active state depending on such things as page type or child count.
6. VMM increments its evicted page count for the SECS of each page (stored in the data structure created in 1).
7. If this is the first evicted page of that SECS, set Marker on SECS of the victim page (EINCVIRTCHILD). This locks the SECS in the guest. The guest cannot page out the SECS.
8. EBLOCK, ETRACK, EWB eviction sequence is executed for page.
9. After loading an SECS page back in, the VMM will set the correct ENCLAVECONTEXT for the guest using ESETCONTEXT instruction.

36.6 CHANGES TO INSTRUCTION BEHAVIOR INSIDE AN ENCLAVE

This section covers instructions whose behavior changes when executed in enclave mode.

36.6.1 Illegal Instructions

The instructions listed in Table 36-1 are ring 3 instructions which become illegal when executed inside an enclave. Executing these instructions inside an enclave will generate an exception.

The first row of Table 36-1 enumerates instructions that may cause a VM exit for VMM emulation. Since a VMM cannot emulate enclave execution, execution of any of these instructions inside an enclave results in an invalid-opcode exception (#UD) and no VM exit.

The second row of Table 36-1 enumerates I/O instructions that may cause a fault or a VM exit for emulation. Again, enclave execution cannot be emulated, so execution of any of these instructions inside an enclave results in #UD.

The third row of Table 36-1 enumerates instructions that load descriptors from the GDT or the LDT or that change privilege level. The former class is disallowed because enclave software should not depend on the contents of the descriptor tables and the latter because enclave execution must be entirely with CPL = 3. Again, execution of any of these instructions inside an enclave results in #UD.

The fourth row of Table 36-1 enumerates instructions that provide access to kernel information from user mode and can be used to aid kernel exploits from within enclave. Execution of any of these instructions inside an enclave results in #UD.

Table 36-1. Illegal Instructions Inside an Enclave

Instructions	Result	Comment
CPUID, GETSEC, RDPMC, SGDT, SIDT, SLDT, STR, VMCALL, VMFUNC	#UD	Might cause VM exit.
IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD	#UD	I/O fault may not safely recover. May require emulation.
Far call, Far jump, Far Ret, INT *n*/INTO, IRET, LDS/LES/LFS/LGS/LSS, MOV to DS/ES/SS/FS/GS, POP DS/ES/SS/FS/GS, SYSCALL, SYSENTER	#UD	Access segment register could change privilege level.
SMSW	#UD	Might provide access to kernel information.
ENCLU[EENTER], ENCLU[ERESUME]	#GP	Cannot enter an enclave from within an enclave.

RDTSC and RDTSCP are legal inside an enclave for processors that support SGX2 (subject to the value of CR4.TSD). For processors which support SGX1 but not SGX2, RDTSC and RDTSCP will cause #UD.

RDTSC and RDTSCP instructions may cause a VM exit when inside an enclave.

Software developers must take into account that the RDTSC/RDTSCP results are not immune to influences by other software, e.g., the TSC can be manipulated by software outside the enclave.

36.6.2 RDRAND and RDSEED Instructions

These instructions may cause a VM exit if the "RDRAND exiting" VM-execution control is 1. Unlike other instructions that can cause VM exits, these instructions are legal inside an enclave. As noted in Section 28.1 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C, any VM exit originating on an instruction boundary inside an enclave sets bit 27 of the exit-reason field of the VMCS. If a VMM receives a VM exit due to an attempt to execute either of these instructions determines (by that bit) that the execution was inside an enclave, it can do either of two things. It can clear the "RDRAND exiting" VM-execution control and execute VMRESUME; this will result in the enclave executing RDRAND or RDSEED again, and this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

NOTE

It is expected that VMMs that virtualize Intel SGX will not set "RDRAND exiting" to 1.

36.6.3 PAUSE Instruction

The PAUSE instruction may cause a VM exit from an enclave if the "PAUSE exiting" VM-execution control is 1. Unlike other instructions that can cause VM exits, the PAUSE instruction is legal inside an enclave. If a VMM receives a VM exit due to the 1-setting of "PAUSE exiting", it can do either of two things. It can clear the "PAUSE exiting" VM-

execution control and execute VMRESUME; this will result in the enclave executing PAUSE again, but this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

The PAUSE instruction may also cause a VM exit outside of an enclave if the "PAUSE-loop exiting" VM-execution control is 1, but as the "PAUSE-loop exiting" control is ignored at CPL > 0 (see Section 26.1.3), VM exit from an enclave due to the 1-setting of "PAUSE-LOOP exiting" will never occur.

NOTE

It is expected that VMMs that virtualize Intel SGX will not set "PAUSE exiting" to 1.

36.6.4 Executions of INT1 and INT3 Inside an Enclave

The INT1 and INT3 instructions are legal inside an enclave, however, their behavior inside an enclave differs from that outside an enclave. See Section 40.4.1 for details.

36.6.5 INVD Handling when Enclaves Are Enabled

Once processor reserved memory protections are activated (see Section 36.5), any execution of INVD will result in a #GP(0).

CHAPTER 37
ENCLAVE EXITING EVENTS

Certain events, such as exceptions and interrupts, incident to (but asynchronous with) enclave execution may cause control to transition outside of enclave mode. (Most of these also cause a change of privilege level.) To protect the integrity and security of the enclave, the processor will exit the enclave (and enclave mode) before invoking the handler for such an event. For that reason, such events are called **enclave-exiting events** (EEE); EEEs include external interrupts, non-maskable interrupts, system-management interrupts, exceptions, and VM exits.

The process of leaving an enclave in response to an EEE is called an **asynchronous enclave exit** (AEX). To protect the secrecy of the enclave, an AEX saves the state of certain registers within enclave memory and then loads those registers with fixed values called **synthetic state**.

37.1 COMPATIBLE SWITCH TO THE EXITING STACK OF AEX

AEXs load registers with a pre-determined synthetic state. These registers may be later pushed onto the appropriate stack in a form as defined by the enclave-exiting event. To allow enclave execution to resume after the invoking handler has processed the enclave exiting event, the asynchronous enclave exit loads the address of trampoline code outside of the enclave into RIP. This trampoline code eventually returns to the enclave by means of an ENCLU(ERESUME) leaf function. Prior to exiting the enclave the RSP and RBP registers are restored to their values prior to enclave entry.

The stack to be used is chosen using the same rules as for non-SGX mode:

- If there is a privilege level change, the stack will be the one associated with the new ring.
- If there is no privilege level change, the current application stack is used.
- If the IA-32e IST mechanism is used, the exit stack is chosen using that method.

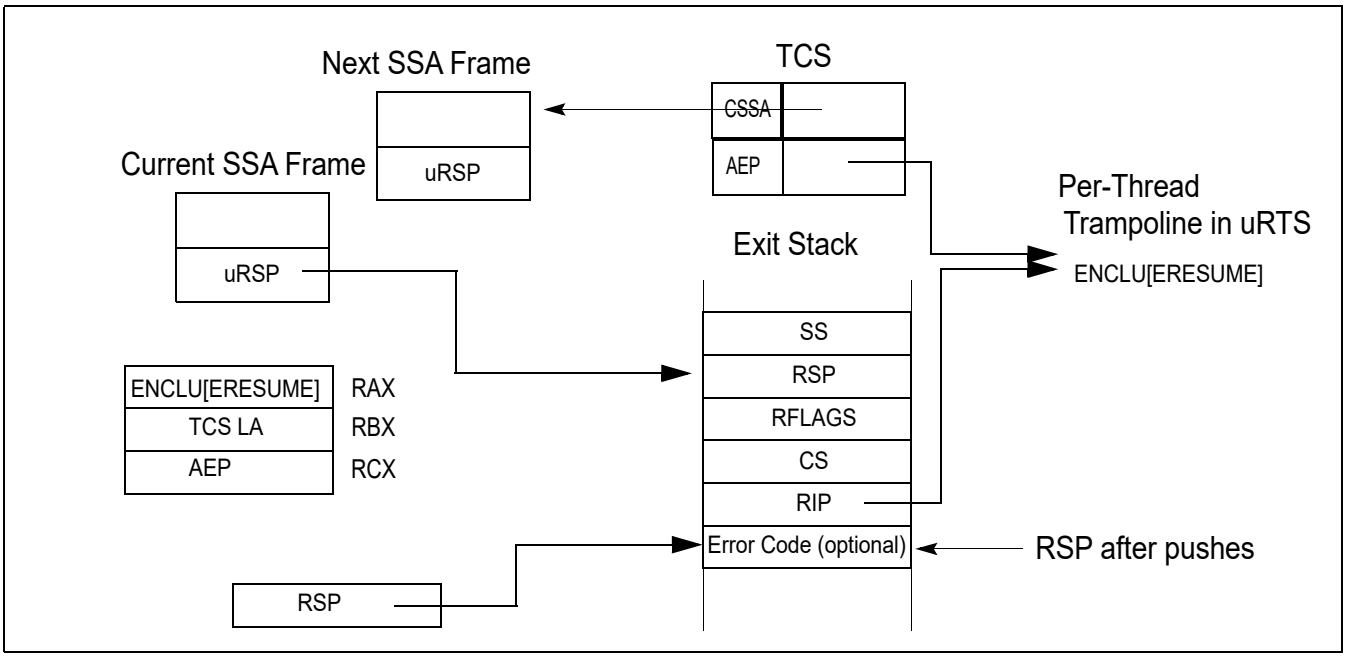

Figure 37-1. Exit Stack Just After Interrupt with Stack Switch

In all cases, the choice of exit stack and the information pushed onto it is consistent with non-SGX operation. Figure 37-1 shows the Application and Exiting Stacks after an exit with a stack switch. An exit without a stack switch uses the Application Stack. The ERESUME leaf index value is placed into RAX, the TCS pointer is placed in RBX and the AEP (see below) is placed into RCX to facilitate resuming the enclave after the exit.

Upon an AEX, the AEP (Asynchronous Exit Pointer) is loaded into the RIP. The AEP points to a trampoline code sequence which includes the ERESUME instruction that is later used to reenter the enclave.

The following bits of RFLAGS are cleared before RFLAGS is pushed onto the exit stack: CF, PF, AF, ZF, SF, OF, RF. The remaining bits are left unchanged.

37.2 STATE SAVING BY AEX

The State Save Area holds the processor state at the time of an AEX. To allow handling events within the enclave and re-entering it after an AEX, the SSA can be a stack of multiple SSA frames as illustrated in Figure 37-2.

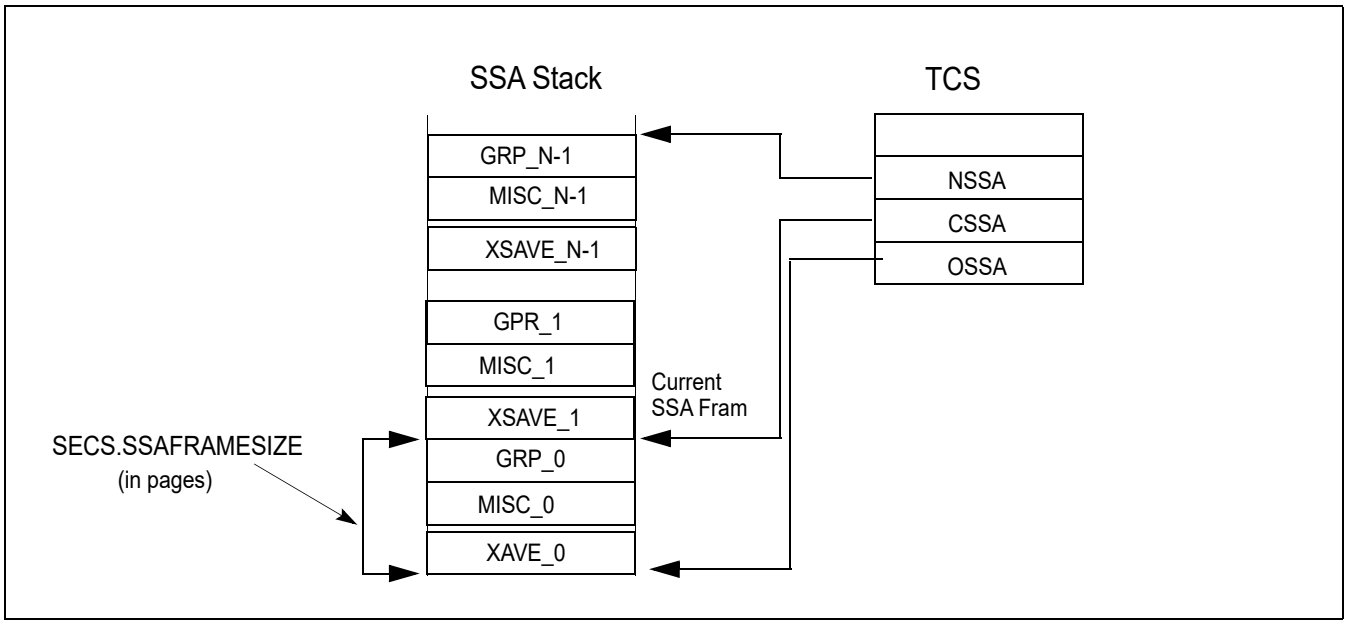

Figure 37-2. The SSA Stack

The location of the SSA frames to be used is controlled by the following variables in the TCS and the SECS:

- Size of a frame in the State Save Area (SECS.SSAFRAMESIZE): This defines the number of 4-KByte pages in a single frame in the State Save Area. The SSA frame size must be large enough to hold the GPR state, the XSAVE state, and the MISC state.
- Base address of the enclave (SECS.BASEADDR): This defines the enclave's base linear address from which the offset to the base of the SSA stack is calculated.
- Number of State Save Area Slots (TCS.NSSA): This defines the total number of slots (frames) in the State Save Area stack.
- Current State Save Area Slot (TCS.CSSA): This defines the slot to use on the next exit.
- State Save Area Offset (TCS.OSSA): This defines the offset of the base address of a set of State Save Area slots from the enclave’s base address.

When an AEX occurs, hardware selects the SSA frame to use by examining TCS.CSSA. Processor state is saved into the SSA frame (see Section 37.4) and loaded with a synthetic state (as described in Section 37.3.1) to avoid leaking secrets, RSP and RBP are restored to their values prior to enclave entry, and TCS.CSSA is incremented. As will be described later, if an exception takes the last slot, it will not be possible to reenter the enclave to handle the

exception from within the enclave. A subsequent ERESUME restores the processor state from the current SSA frame and frees the SSA frame.

The format of the XSAVE section of SSA is identical to the format used by the XSAVE/XRSTOR instructions. On EENTER, CSSA must be less than NSSA, ensuring that there is at least one State Save Area slot available for exits. If there is no free SSA frame when executing EENTER, the entry will fail.

37.3 SYNTHETIC STATE ON ASYNCHRONOUS ENCLAVE EXIT

37.3.1 Processor Synthetic State on Asynchronous Enclave Exit

Table 37-1 shows the synthetic state loaded on AEX. The values shown are the lower 32 bits when the processor is in 32 bit mode and 64 bits when the processor is in 64 bit mode.

Table 37-1. GPR, x87 Synthetic States on Asynchronous Enclave Exit

Register	Value
RAX	3 (ENCLU[3] is ERESUME).
RBX	Pointer to TCS of interrupted enclave thread.
RCX	AEP of interrupted enclave thread.
RDX, RSI, RDI	0.
RSP	Restored from SSA.uRSP.
RBP	Restored from SSA.uRBP.
R8-R15	0 in 64-bit mode; unchanged in 32-bit mode.
RIP	AEP of interrupted enclave thread.
RFLAGS	CF, PF, AF, ZF, SF, OF, RF bits are cleared. All other bits are left unchanged.
x87/SSE State	Unless otherwise listed here, all x87 and SSE state are set to the INIT state. The INIT state is the state that would be loaded by the XRSTOR instruction with bits 1:0 both set in the requested feature bitmask (RFBM), and both clear in XSTATE_BV the XSAVE header.
FCW	On #MF exception: set to 037EH. On all other exits: set to 037FH.
FSW	On #MF exception: set to 8081H. On all other exits: set to 0H.
MXCSR	On #XM exception: set to 1F01H. On all other exits: set to 1FB0H.
CR2	If the event that caused the AEX is a #PF, and the #PF does not directly cause a VM exit, then the low 12 bits are cleared. If the #PF leads directly to a VM exit, CR2 is not updated (usual IA behavior). Note: The low 12 bits are not cleared if a #PF is encountered during the delivery of the EEE that caused the AEX. This is because the #PF was not the EEE.
FS, GS	Restored to values as of most recent EENTER/ERESUME.

37.3.2 Synthetic State for Extended Features

When CR4.OSXSAVE = 1, extended features (those controlled by XCR0[63:2]) are set to their respective INIT states when this corresponding bit of SECS.XFRM is set. The INIT state is the state that would be loaded by the XRSTOR instruction had the instruction mask and the XSTATE_BV field of the XSAVE header each contained the value XFRM. (When the AEX occurs in 32-bit mode, those features that do not exist in 32-bit mode are unchanged.)

37.3.3 Synthetic State for MISC Features

State represented by SECS.MISCSELECT might also be overridden by synthetic state after it has been saved into the SSA. State represented by MISCSELECT[0] is not overridden but if the exiting event is a page fault then lower 12 bits of CR2 are cleared.

37.4 AEX FLOW

On Enclave Exiting Events (interrupts, exceptions, VM exits or SMIs), the processor state is securely saved inside the enclave, a synthetic state is loaded and the enclave is exited. The EEE then proceeds in the usual exit-defined fashion. The following sections describes the details of an AEX:

1. The exact processor state saved into the current SSA frame depends on whether the enclave is a 32-bit or a 64-bit enclave. In 32-bit mode (IA32_EFER.LMA = 0 || CS.L = 0), the low 32 bits of the legacy registers (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EIP, and EFLAGS) are stored. The upper 32 bits of the legacy registers and the 64-bit registers (R8 … R15) are not stored.

   In 64-bit mode (IA32_EFER.LMA = 1 && CS.L = 1), all 64 bits of the general processor registers (RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8 … R15, RIP, and RFLAGS) are stored.

   The state of those extended features specified by SECS.ATTRIBUTES.XFRM are stored into the XSAVE area of the current SSA frame. The layout of the x87 and XMM portions (the 1st 512 bytes) depends on the current values of IA32_EFER.LMA and CS.L:

   If IA32_EFER.LMA = 0 || CS.L = 0, the same format (32-bit) that XSAVE/FXSAVE uses with these values.

   If IA32_EFER.LMA = 1 && CS.L = 1, the same format (64-bit) that XSAVE/FXSAVE uses with these values when REX.W = 1.

   The cause of the AEX is saved in the EXITINFO field. See Table 35-10 for details and values of the various fields.

   The state of those miscellaneous features (see Section 35.7.2) specified by SECS.MISCSELECT are stored into the MISC area of the current SSA frame.

   If CET was enabled in the enclave, then the CET state of the enclave is saved in the CET state save area. If shadow stacks were enabled in the enclave, then the SSP is also saved into the TCS.PREVSSP field.
2. Synthetic state is created for a number of processor registers to present an opaque view of the enclave state. Table 37-1 shows the values for GPRs, x87, SSE, FS, GS, Debug, and performance monitoring on AEX. The synthetic state for other extended features (those controlled by XCR0[62:2]) is set to their respective INIT states when their corresponding bit of SECS.ATTRIBUTES.XFRM is set. The INIT state is that state as defined by the behavior of the XRSTOR instruction when HEADER.XSTATE_BV[n] is 0. Synthetic state of those miscellaneous features specified by SECS.MISCSELECT depends on the miscellaneous feature. There is no synthetic state required for the miscellaneous state controlled by SECS.MISCSELECT[0].
3. Any code and data breakpoints that were suppressed at the time of enclave entry are unsuppressed when exiting the enclave.
4. RFLAGS.TF is set to the value that it had at the time of the most recent enclave entry (except for the situation that the entry was opt-in for debug; see Section 40.2). In the SSA, RFLAGS.TF is set to 0.
5. RFLAGS.RF is set to 0 in the synthetic state. In the SSA, the value saved is the same as what would have been saved on stack in the non-SGX case (architectural value of RF). Thus, AEXs due to interrupts, traps, and code breakpoints save RF unmodified into SSA, while AEXs due to other faults save RF as 1 in the SSA.

   If the event causing AEX happened on intermediate iteration of a REP-prefixed instruction, then RF=1 is saved on SSA, irrespective of its priority.
6. Any performance monitoring activity (including PEBS) or profiling activity (LBR, Tracing using Intel PT) on the exiting thread that was suppressed due to the enclave entry on that thread is unsuppressed. Any counting that had been demoted from AnyThread counting to MyThread counting (on one logical processor) is promoted back to AnyThread counting.
7. The CET state of the enclosing application is restored to the state at the time of the most recent enclave entry, and if CET indirect branch tracking was enabled then the indirect branch tracker is unsuppressed and moved to the WAIT_FOR_ENDBRANCH state.

37.4.1 AEX Operational Detail

Temp Variables in AEX Operational Flow

Name	Type	Size (bits)	Description
TMP_RIP	Effective Address	32/64	Address of instruction at which to resume execution on ERESUME.
TMP_MODE64	binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1)).
TMP_BRANCH_RECORD	LBR Record	2x64	From/To address to be pushed onto LBR stack.

The pseudo code in this section describes the internal operations that are executed when an AEX occurs in enclave mode. These operations occur just before the normal interrupt or exception processing occurs.

```
(* Save RIP for later use *)
TMP_RIP = Linear Address of Resume RIP
(* Is the processor in 64-bit mode? *)
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Save all registers, When saving EFLAGS, the TF bit is set to 0 and
   the RF bit is set to what would have been saved on stack in the non-SGX case *)

 IF (TMP_MODE64 = 0)
   THEN
       Save EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EFLAGS, EIP into the current SSA frame using
CR_GPR_PA; (* see Table 38-5 for list of CREGs used to describe internal operation within Intel SGX *)
       SSA.RFLAGS.TF := 0;
   ELSE    (* TMP_MODE64 = 1 *)
       Save RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8-R15, RFLAGS, RIP into the current SSA frame using
CR_GPR_PA;
       SSA.RFLAGS.TF := 0;
FI;
Save FS and GS BASE into SSA using CR_GPR_PA;

(* store XSAVE state into the current SSA frame's XSAVE area using the physical addresses
   that were determined and cached at enclave entry time with CR_XSAVE_PAGE_i. *)
For each XSAVE state i defined by (SECS.ATTRIBUTES.XFRM[i] = 1, destination address cached in
CR_XSAVE_PAGE_i)
  SSA.XSAVE.i := XSAVE_STATE_i;

(* Clear bytes 8 to 23 of XSAVE_HEADER, i.e., the next 16 bytes after XHEADER_BV *)

CR_XSAVE_PAGE_0.XHEADER_BV[191:64] := 0;

(* Clear bits in XHEADER_BV[63:0] that are not enabled in ATTRIBUTES.XFRM *)

CR_XSAVE_PAGE_0.XHEADER_BV[63:0] :=
   CR_XSAVE_PAGE_0.XHEADER_BV[63:0] & SECS(CR_ACTIVE_SECS).ATTRIBUTES.XFRM;
   Apply synthetic state to GPRs, RFLAGS, extended features, etc.

(* Restore the RSP and RBP from the current SSA frame's GPR area using the physical address
   that was determined and cached at enclave entry time with CR_GPR_PA. *)
RSP := CR_GPR_PA.URSP;
RBP := CR_GPR_PA.URBP;
```

```
(* Restore the FS and GS *)
FS.selector := CR_SAVE_FS.selector;
FS.base := CR_SAVE_FS.base;
FS.limit := CR_SAVE_FS.limit;
FS.access_rights := CR_SAVE_FS.access_rights;
GS.selector := CR_SAVE_GS.selector;
GS.base := CR_SAVE_GS.base;
GS.limit := CR_SAVE_GS.limit;
GS.access_rights := CR_SAVE_GS.access_rights;

(* Examine exception code and update enclave internal states*)
exception_code := Exception or interrupt vector;

(* Indicate the exit reason in SSA *)
IF (exception_code = (#DE OR #DB OR #BP OR #BR OR #UD OR #MF OR #AC OR #XM ))
   THEN
       CR_GPR_PA.EXITINFO.VECTOR := exception_code;
       IF (exception code = #BP)
           THEN CR_GPR_PA.EXITINFO.EXIT_TYPE := 6;
           ELSE CR_GPR_PA.EXITINFO.EXIT_TYPE := 3;
       FI;
       CR_GPR_PA.EXITINFO.VALID := 1;
   ELSE IF (exception_code is #PF or #GP )
       THEN
       (* Check SECS.MISCSELECT using CR_ACTIVE_SECS *)
       IF (SECS.MISCSELECT[0] is set)
           THEN
           CR_GPR_PA.EXITINFO.VECTOR := exception_code;
           CR_GPR_PA.EXITINFO.EXIT_TYPE := 3;
           IF (exception_code is #PF)
               THEN
                   SSA.MISC.EXINFO. MADDR := CR2;
                   SSA.MISC.EXINFO.ERRCD := PFEC;
                   SSA.MISC.EXINFO.RESERVED := 0;
           ELSE
               SSA.MISC.EXINFO. MADDR := 0;
               SSA.MISC.EXINFO.ERRCD := GPEC;
               SSA.MISC.EXINFO.RESERVED := 0;
           FI;
           CR_GPR_PA.EXITINFO.VALID := 1;
   ELSE IF (exception code is #CP)
       THEN
           IF (SECS.MISCSELECT[1] is set)
               THEN
                   CR_GPR_PA.EXITINFO.VECTOR := exception_code;
                   CR_GPR_PA.EXITINFO.EXIT_TYPE := 3;
                   CR_GPR_PA.EXITINFO.VALID := 1;
                   SSA.MISC.EXINFO. MADDR := 0;
                   SSA.MISC.EXINFO.ERRCD := CPEC;
                   SSA.MISC.EXINFO.RESERVED := 0;
           FI;
       FI;
   ELSE
```

```
        CR_GPR_PA.EXITINFO.VECTOR := 0;
        CR_GPR_PA.EXITINFO.EXIT_TYPE := 0
        CR_GPR_PA.REASON.VALID := 0;
FI;

(* Execution will resume at the AEP *)
RIP := CR_TCS_PA.AEP;

(* Set EAX to the ERESUME leaf index *)
EAX := 3;

(* Put the TCS LA into RBX for later use by ERESUME *)
RBX := CR_TCS_LA;

(* Put the AEP into RCX for later use by ERESUME *)
RCX := CR_TCS_PA.AEP;

(* Increment the SSA frame # *)
CR_TCS_PA.CSSA := CR_TCS_PA.CSSA + 1;

(* Restore XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
   THEN XCR0 := CR_SAVE_XCR0; FI;

Un-suppress all code breakpoints that are outside ELRANGE

IF (CPUID.(EAX=12H, ECX=1):EAX[6]= 1)
   THEN
        IF (CR4.CET == 1 AND IA32_U_CET.SH_STK_EN == 1)
            THEN
                CR_CET_SAVE_AREA_PA.SSP := SSP;
                CR_TCS_PA.PREVSSP := SSP;
        FI;
        IF (CR4.CET == 1 AND IA32_U_CET.ENDBR_EN == 1)
            THEN
                CR_CET_SAVE_AREA_PA.TRACKER := IA32_U_CET.TRACKER;
                CR_CET_SAVE_AREA_PA.SUPPRESS := IA32_U_CET.SUPPRESS;
        FI;
FI;
IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7H, ECX=0):ECX[CET_SS] = 1)
   THEN
        (* restore enclosing applications CET state *)
        IA32_U_CET := CR_SAVE_IA32_U_CET;

        IF (CPUID.(EAX=7, ECX=0):ECX[CET_SS])
            SSP := CR_SAVE_SSP; FI;

        (* If indirect branch tracking enabled for enclosing application *)
        (* then move the tracker to wait_for_endbranch *)
        IF (CR4.CET == 1 AND IA32_U_CET.ENDBR_EN == 1)
            THEN
                IA32_U_CET.TRACKER := WAIT_FOR_ENDBRANCH;
                IA32_U_CET.SUPPRESS := 0;
        FI;
```

```
FI;

(* Update the thread context to show not in enclave mode *)
CR_ENCLAVE_MODE := 0;

(* Assure consistent translations. *)
Flush linear context including TLBs and paging-structure caches

IF (CR_DBGOPTIN = 0)
    THEN
        Un-suppress all breakpoints that overlap ELRANGE
        (* Clear suppressed breakpoint matches *)
        Restore suppressed breakpoint matches
        (* Restore TF *)
        RFLAGS.TF := CR_SAVE_TF;
        Un-suppress monitor trap flag;
        Un-suppress branch recording facilities;
        Un-suppress all suppressed performance monitoring activity;
        Promote any sibling-thread counters that were demoted from AnyThread to MyThread during enclave
entry back to AnyThread;
FI;

IF the "monitor trap flag" VM-execution control is 1
    THEN Pend MTF VM Exit at the end of exit; FI;

(* Clear low 12 bits of CR2 on #PF *)
IF (Exception code is #PF)
    THEN CR2 := CR2 & ~0xFFF; FI;

(* end_of_flow *)
(* Execution continues with normal event processing. *)
```

CHAPTER 38
INTEL® SGX INSTRUCTION REFERENCES

This chapter describes the supervisor and user level instructions provided by Intel® Software Guard Extensions (Intel® SGX). In general, various functionality is encoded as leaf functions within the ENCLS (supervisor), ENCLU (user), and the ENCLV (virtualization operation) instruction mnemonics. Different leaf functions are encoded by specifying an input value in the EAX register of the respective instruction mnemonic.

38.1 INTEL® SGX INSTRUCTION SYNTAX AND OPERATION

ENCLS, ENCLU, and ENCLV instruction mnemonics for all leaf functions are covered in this section.

For all instructions, the value of CS.D is ignored; addresses and operands are 64 bits in 64-bit mode and are otherwise 32 bits. Aside from EAX specifying the leaf number as input, each instruction leaf may require all or some subset of the RBX/RCX/RDX as input parameters. Some leaf functions may return data or status information in one or more of the general purpose registers.

38.1.1 ENCLS Register Usage Summary

Table 38-1 summarizes the implicit register usage of supervisor mode enclave instructions.

Table 38-1. Register Usage of Privileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
ECREATE	00H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EADD	01H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EINIT	02H (In)	SIGSTRUCT (In, EA)	SECS (In, EA)	EINITTOKEN (In, EA)
EREMOVE	03H (In)		EPCPAGE (In, EA)	
EDBGRD	04H (In)	Result Data (Out)	EPCPAGE (In, EA)	
EDBGWR	05H (In)	Source Data (In)	EPCPAGE (In, EA)	
EEXTEND	06H (In)	SECS (In, EA)	EPCPAGE (In, EA)	
ELDB	07H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDU	08H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
EBLOCK	09H (In)		EPCPAGE (In, EA)	
EPA	0AH (In)	PT_VA (In)	EPCPAGE (In, EA)	
EWB	0BH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ETRACK	0CH (In)		EPCPAGE (In, EA)	
EAUG	0DH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EMODPR	0EH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODT	0FH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
ERDINFO	010H (In)	RDINFO (In, EA*)	EPCPAGE (In, EA)	
ETRACKC	011H (In)		EPCPAGE (In, EA)	
ELDBC	012H (In)	PAGEINFO (In, EA*)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDUC	013H (In)	PAGEINFO (In, EA*)	EPCPAGE (In, EA)	VERSION (In, EA)
EA: Effective Address				

38.1.2 ENCLU Register Usage Summary

Table 38-2 summarizes the implicit register usage of user mode enclave instructions.

Table 38-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
EREPORT	00H (In)	TARGETINFO (In, EA)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)
EGETKEY	01H (In)	KEYREQUEST (In, EA)	KEY (In, EA)	
EENTER	02H (In)	TCS (In, EA)	AEP (In, EA)	
	RBX.CSSA (Out)		Return (Out, EA)	
ERESUME	03H (In)	TCS (In, EA)	AEP (In, EA)	
EEXIT	04H (In)	Target (In, EA)	Current AEP (Out)	
EACCEPT	05H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODPE	06H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EACCEPTCOPY	07H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	EPCPAGE (In, EA)
EDECCSSA	09H (In)			
EA: Effective Address				

38.1.3 ENCLV Register Usage Summary

Table 38-3 summarizes the implicit register usage of virtualization operation enclave instructions.

Table 38-3. Register Usage of Virtualization Operation Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
EDECVIRTCHILD	00H (In)	EPCPAGE (In, EA)	SECS (In, EA)	
EINCVIRTCHILD	01H (In)	EPCPAGE (In, EA)	SECS (In, EA)	
ESETCONTEXT	02H (In)		EPCPAGE (In, EA)	Context Value (In, EA)
EA: Effective Address				

38.1.4 Information and Error Codes

Information and error codes are reported by various instruction leaf functions to show an abnormal termination of the instruction or provide information which may be useful to the developer. Table 38-4 shows the various codes and the instruction which generated the code. Details of the meaning of the code is provided in the individual instruction.

Table 38-4. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
No Error	0	
SGX_INVALID_SIG_STRUCT	1	EINIT
SGX_INVALID_ATTRIBUTE	2	EINIT, EGETKEY
SGX_BLKSTATE	3	EBLOCK
SGX_INVALID_MEASUREMENT	4	EINIT
SGX_NOTBLOCKABLE	5	EBLOCK
SGX_PG_INVLD	6	EBLOCK, ERDINFO, ETRACKC

Table 38-4. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
SGX_EPC_PAGE_CONFLICT	7	EBLOCK, EMODPR, EMODT, ERDINFO , EDECVIRTCHILD, EINCVIRTCHILD, ELDBC, ELDUC, ESETCONTEXT, ETRACKC
SGX_INVALID_SIGNATURE	8	EINIT
SGX_MAC_COMPARE_FAIL	9	ELDB, ELDU, ELDBC, ELDUC
SGX_PAGE_NOT_BLOCKED	10	EWB
SGX_NOT_TRACKED	11	EWB, EACCEPT
SGX_VA_SLOT_OCCUPIED	12	EWB
SGX_CHILD_PRESENT	13	EWB, EREMOVE
SGX_ENCLAVE_ACT	14	EREMOVE
SGX_ENTRYEPOCH_LOCKED	15	EBLOCK
SGX_INVALID_EINITTOKEN	16	EINIT
SGX_PREV_TRK_INCMPL	17	ETRACK, ETRACKC
SGX_PG_IS_SECS	18	EBLOCK
SGX_PAGE_ATTRIBUTES_MISMATCH	19	EACCEPT, EACCEPTCOPY
SGX_PAGE_NOT_MODIFIABLE	20	EMODPR, EMODT
SGX_PAGE_NOT_DEBUGGABLE	21	EDBGRD, EDBGWR
SGX_INVALID_COUNTER	25	EDECVIRTCHILD
SGX_PG_NONEPC	26	ERDINFO
SGX_TRACK_NOT_REQUIRED	27	ETRACKC
SGX_INVALID_CPUSVN	32	EINIT, EGETKEY
SGX_INVALID_ISVSVN	64	EGETKEY
SGX_UNMASKED_EVENT	128	EINIT
SGX_INVALID_KEYNAME	256	EGETKEY

38.1.5 Internal CREGs

The CREGs as shown in Table 5-4 are hardware specific registers used in this document to indicate values kept by the processor. These values are used while executing in enclave mode or while executing an Intel SGX instruction. These registers are not software visible and are implementation specific. The values in Table 38-5 appear at various places in the pseudo-code of this document. They are used to enhance understanding of the operations.

Table 38-5. List of Internal CREG

Name	Size (Bits)	Scope
CR_ENCLAVE_MODE	1	LP
CR_DBGOPTIN	1	LP
CR_TCS_LA	64	LP
CR_TCS_PA	64	LP
CR_ACTIVE_SECS	64	LP
CR_ELRANGE	128	LP
CR_SAVE_TF	1	LP
CR_SAVE_FS	64	LP
CR_GPR_PA	64	LP
CR_XSAVE_PAGE_n	64	LP

Table 38-5. List of Internal CREG

Name	Size (Bits)	Scope
CR_SAVE_DR7	64	LP
CR_SAVE_PERF_GLOBAL_CTRL	64	LP
CR_SAVE_DEBUGCTL	64	LP
CR_SAVE_PEBS_ENABLE	64	LP
CR_CPUSVN	128	PACKAGE
CR_SGXOWNEREPOCH	128	PACKAGE
CR_SAVE_XCR0	64	LP
CR_SGX_ATTRIBUTES_MASK	128	LP
CR_PAGING_VERSION	64	PACKAGE
CR_VERSION_THRESHOLD	64	PACKAGE
CR_NEXT_EID	64	PACKAGE
CR_BASE_PK	128	PACKAGE
CR_SEAL_FUSES	128	PACKAGE
CR_CET_SAVE_AREA_PA	64	LP
CR_ENCLAVE_SS_TOKEN_PA	64	LP
CR_SAVE_IA32_U_CET	64	LP
CR_SAVE_SSP	64	LP

38.1.6 Concurrent Operation Restrictions

Under certain conditions, Intel SGX disallows certain leaf functions from operating concurrently. Listed below are some examples of concurrency that are not allowed.

- For example, Intel SGX disallows the following leafs to concurrently operate on the same EPC page.
  - ECREATE, EADD, and EREMOVE are not allowed to operate on the same EPC page concurrently with themselves.
  - EADD, EEXTEND, and EINIT leaves are not allowed to operate on the same SECS concurrently.
- Intel SGX disallows the EREMOVE leaf from removing pages from an enclave that is in use.
- Intel SGX disallows entry (EENTER and ERESUME) to an enclave while a page from that enclave is being removed.

When disallowed operation is detected, a leaf function may do one of the following:

- Return an SGX_EPC_PAGE_CONFLICT error code in RAX.
- Cause a #GP(0) exception.

To prevent such exceptions, software must serialize leaf functions or prevent these leaf functions from accessing the same EPC page.

38.1.6.1 Concurrency Tables of Intel® SGX Instructions

The tables below detail the concurrent operation restrictions of all SGX leaf functions. For each leaf function, the table has a separate line for each of the EPC pages the leaf function accesses.

For each such EPC page, the base concurrency requirements are detailed as follows:

- **Exclusive Access** means that no other leaf function that requires either shared or exclusive access to the same EPC page may be executed concurrently. For example, EADD requires an exclusive access to the target page it accesses.

- **Shared Access** means that no other leaf function that requires an exclusive access to the same EPC page may be executed concurrently. Other leaf functions that require shared access may run concurrently. For example, EADD requires a shared access to the SECS page it accesses.
- **Concurrent Access** means that any other leaf function that requires any access to the same EPC page may be executed concurrently. For example, EGETKEY has no concurrency requirements for the KEYREQUEST page.

In addition to the base concurrency requirements, additional concurrency requirements are listed, which apply only to specific sets of leaf functions. For example, there are additional requirements that apply for EADD, EXTEND, and EINIT. EADD and EEXTEND can't execute concurrently on the same SECS page.

The tables also detail the leaf function's behavior when a conflict happens, i.e., a concurrency requirement is not met. In this case, the leaf function may return an SGX_EPC_PAGE_CONFLICT error code in RAX, or it may cause an exception. In addition, the tables detail those conflicts where a VM Exit may be triggered, and list the Exit Qualification code that is provided in such cases.

Table 38-6. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions		
			Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPT	Target	[DS:RCX]	Shared	#GP	
	SECINFO	[DS:RBX]	Concurrent		
EACCEPTCOPY	Target	[DS:RCX]	Concurrent		
	Source	[DS:RDX]	Concurrent		
	SECINFO	[DS:RBX]	Concurrent		
EADD	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP	
EAUG	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP	
EBLOCK	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
ECREATE	SECS	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
EDBGRD	Target	[DS:RCX]	Shared	#GP	
EDBGWR	Target	[DS:RCX]	Shared	#GP	
EDECVIRTCHILD	Target	[DS:RBX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	[DS:RCX]	Concurrent		
EENTERTCS	SECS	[DS:RBX]	Shared	#GP	
EEXIT			Concurrent		
EEXTEND	Target	[DS:RCX]	Shared	#GP	
	SECS	[DS:RBX]	Concurrent		
EGETKEY	KEYREQUEST	[DS:RBX]	Concurrent		
	OUTPUTDATA	[DS:RCX]	Concurrent		
EINCVIRTCHILD	Target	[DS:RBX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	[DS:RCX]	Concurrent		
EINIT	SECS	[DS:RCX]	Shared	#GP	

Table 38-6. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions		
			Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ELDB/ELDU	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA	[DS:RDX]	Shared	#GP	
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP	
EDLBC/ELDUC	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
	VA	[DS:RDX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	SGX_EPC_PAGE_CONFLICT	
EMODPE	Target	[DS:RCX]	Concurrent		
	SECINFO	[DS:RBX]	Concurrent		
EMODPR	Target	[DS:RCX]	Shared	#GP	
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
EPA	VA	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
ERDINFO	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
EREMOVE	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
EREPORT	TARGETINFO	[DS:RBX]	Concurrent		
	REPORTDATA	[DS:RCX]	Concurrent		
	OUTPUTDATA	[DS:RDX]	Concurrent		
ERESUME	TCS	[DS:RBX]	Shared	#GP	
ESETCONTEXT	SECS	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
ETRACK	SECS	[DS:RCX]	Shared	#GP	
ETRACKC	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	Implicit	Concurrent		
EWB	Source	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA	[DS:RDX]	Shared	#GP	

Table 38-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPT	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	

Table 38-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPTCOPY	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	Source	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EADD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Exclusive	#GP	Concurrent	
EAUG	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EBLOCK	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ECREATE	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDBGRD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDBGWR	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDECVIRTCHILD	Target	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EENTERTCS	SECS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EEXIT			Concurrent		Concurrent		Concurrent	
EEXTEND	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	
EGETKEY	KEYREQUEST	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EINCVIRTCHILD	Target	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EINIT	SECS	[DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	
ELDB/ELDU	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EDLBC/ELDUC	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EMODPE	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EMODPR	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	

Table 38-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	
EPA	VA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ERDINFO	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREMOVE	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREPORT	TARGETINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
ERESUME	TCS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
ESETCONTEXT	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ETRACK	SECS	[DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT[1]
ETRACKC	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	Implicit	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT[1]
EWB	Source	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	

NOTES:

1. SGX_CONFLICT VM Exit Qualification =TRACKING_RESOURCE_CONFLICT.

38.2 INTEL® SGX INSTRUCTION REFERENCE

ENCLS—Execute an Enclave System Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 CF ENCLS	ZO	V/V	NA	This instruction is used to execute privileged Intel SGX leaf functions that are used for managing and debugging the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
ZO	NA	NA	NA	See Section 38.3

Description

The ENCLS instruction invokes the specified privileged Intel SGX leaf function for managing and debugging enclaves. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLS instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute the instruction when CPL > 0 results in #UD. The instruction produces a general-protection exception (#GP) if CR0.PG = 0 or if an attempt is made to invoke an undefined leaf function.

In VMX non-root operation, execution of ENCLS may cause a VM exit if the "enable ENCLS exiting" VM-execution control is 1. In this case, execution of individual leaf functions of ENCLS is governed by the ENCLS-exiting bitmap field in the VMCS. Each bit in that field corresponds to the index of an ENCLS leaf function (as provided in EAX).

Software in VMX root operation can thus intercept the invocation of various ENCLS leaf functions in VMX non-root operation by setting the "enable ENCLS exiting" VM-execution control and setting the corresponding bits in the ENCLS-exiting bitmap.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 || CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 || CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, and is the REX prefix in 64-bit mode.

Operation

```
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE = 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0
    THEN #UD; FI;

IF (CPL > 0)
    THEN #UD; FI;

IF in VMX non-root operation and the "enable ENCLS exiting" VM-execution control is 1
    THEN
        IF EAX < 63 and ENCLS_exiting_bitmap[EAX] = 1 or EAX> 62 and ENCLS_exiting_bitmap[63] = 1
            THEN VM exit;
        FI;
FI;
IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;

IF (EAX is an invalid leaf number)
    THEN #GP(0); FI;
```

```
IF CR0.PG = 0
    THEN #GP(0); FI;

(* DS must not be an expanded down segment *)
IF not in 64-bit mode and DS.Type is expand-down data
    THEN #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If data segment expand down.
	If CR0.PG=0.

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD	ENCLS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.

ENCLU—Execute an Enclave User Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 D7 ENCLU	ZO	V/V	NA	This instruction is used to execute non-privileged Intel SGX leaf functions.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
ZO	NA	NA	NA	See Section 38.4

Description

The ENCLU instruction invokes the specified non-privileged Intel SGX leaf functions. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLU instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute this instruction when CPL < 3 results in #UD. The instruction produces a general-protection exception (#GP) if either CR0.PG or CR0.NE is 0, or if an attempt is made to invoke an undefined leaf function. The ENCLU instruction produces a device not available exception (#NM) if CR0.TS = 1.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 or CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 and CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, as is the REX prefix in 64-bit mode.

Operation

```
IN_64BIT_MODE := 0;
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

(* If enclosing app has CET indirect branch tracking enabled then if it is not ERESUME leaf cause a #CP fault *)
(* If the ERESUME is not successful it will leave tracker in WAIT_FOR_ENDBRANCH *)
TRACKER = (CPL == 3) ? IA32_U_CET.TRACKER : IA32_S_CET.TRACKER
IF EndbranchEnabledAndNotSuppressed(CPL) and TRACKER = WAIT_FOR_ENDBRANCH and
  (EAX != ERESUME or CR0.TS or (in SMM) or (CPUID.SGX_LEAF.0:EAX.SE1 = 0) or (CPL < 3))
    THEN
        Handle CET State machine violation      (* see Section 17.3.6, "Legacy Compatibility Treatment," in the
                                                   Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1. *)
FI;

IF CR0.PE= 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0
    THEN #UD; FI;

IF CR0.TS = 1
    THEN #NM; FI;

IF CPL < 3
    THEN #UD; FI;

IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;
```

```
IF EAX is invalid leaf number
    THEN #GP(0); FI;

IF CR0.PG = 0 or CR0.NE = 0
    THEN #GP(0); FI;

IN_64BIT_MODE := IA32_EFER.LMA AND CS.L ? 1 : 0;
(* Check not in 16-bit mode and DS is not a 16-bit segment *)
IF not in 64-bit mode and CS.D = 0
    THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 1 and (EAX = 2 or EAX = 3) (* EENTER or ERESUME *)
    THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 0 and (EAX = 0 or EAX = 1 or EAX = 4 or EAX = 5 or EAX = 6 or EAX = 7 or EAX = 9)
(* EREPORT, EGETKEY, EEXIT, EACCEPT, EMODPE, EACCEPTCOPY, or EDECCSSA *)
    THEN #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 3.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1.
	If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0.
	If operating in 16-bit mode.
	If data segment is in 16-bit mode.
	If CR0.PG = 0 or CR0.NE= 0.
#NM	If CR0.TS = 1.

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD	ENCLS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 3.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1.
	If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0.
	If CR0.NE= 0.
#NM	If CR0.TS = 1.

ENCLV—Execute an Enclave VMM Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 C0 ENCLV	ZO	V/V	NA	This instruction is used to execute privileged SGX leaf functions that are reserved for VMM use. They are used for managing the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
ZO	NA	NA	NA	See Section 38.3

Description

The ENCLV instruction invokes the virtualization SGX leaf functions for managing enclaves in a virtualized environment. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In non 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLV instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, if it is executed in system-management mode (SMM), or not in VMX operation. Additionally, any attempt to execute the instruction when CPL > 0 results in #UD. The instruction produces a general-protection exception (#GP) if CR0.PG = 0 or if an attempt is made to invoke an undefined leaf function.

Software in VMX root mode of operation can enable execution of the ENCLV instruction in VMX non-root mode by setting enable ENCLV execution control in the VMCS. If enable ENCLV execution control in the VMCS is clear, execution of the ENCLV instruction in VMX non-root mode results in #UD.

When execution of ENCLV instruction in VMX non-root mode is enabled, software in VMX root operation can intercept the invocation of various ENCLV leaf functions in VMX non-root operation by setting the corresponding bits in the ENCLV-exiting bitmap.

Addresses and operands are 32 bits in 32-bit mode (IA32_EFER.LMA == 0 || CS.L == 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA == 1 && CS.L == 1). CS.D value has no impact on address calculation.

Segment override prefixes and address-size override prefixes are ignored, as is the REX prefix in 64-bit mode.

Operation

```
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE = 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.OSS = 0
    THEN #UD; FI;

IF not in VMX Operation or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD; FI;

IF (CPL > 0)
    THEN #UD; FI;

IF in VMX non-root operation
  IF "enable ENCLV exiting" VM-execution control is 1
    THEN
      IF EAX < 63 and ENCLV_exiting_bitmap[EAX] = 1 or EAX> 62 and ENCLV_exiting_bitmap[63] = 1
          THEN VM exit;
      FI;
  ELSE
    #UD; FI;
```

```
FI;

IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;

IF (EAX is an invalid leaf number)
    THEN #GP(0); FI;

IF CR0.PG = 0
    THEN #GP(0); FI;

(* DS must not be an expanded down segment *)
IF not in 64-bit mode and DS.Type is expand-down data
    THEN #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions.

Protected Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.OSS [bit 5] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If data segment expand down.
	If CR0.PG=0.

Real-Address Mode Exceptions

#UD	ENCLV is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD	ENCLV is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.OSS [bit 5] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.

38.3 INTEL® SGX SYSTEM LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLS instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EADD—Add a Page to an Uninitialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLS[EADD]	IR	V/V	SGX1	This leaf function adds a page to an uninitialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EADD (In)	Address of a PAGEINFO (In)	Address of the destination EPC page (In)

Description

This leaf function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. As part of the association, the enclave offset and the security attributes are measured and extended into the SECS.MRENCLAVE. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of EADD leaf function.

EADD Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EADD Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	If security attributes specifies a TCS and the source page specifies unsupported TCS values or fields.
The SECS has been initialized.	The specified enclave offset is outside of the enclave address space.

Concurrency Restrictions

Table 38-8. Base Concurrency Restrictions of EADD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EADD	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	

Table 38-9. Additional Concurrency Restrictions of EADD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EADD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGE-INFO.SECS	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EADD Operational Flow

Name	Type	Size (bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
TMP_LINADDR := DS:RBX.LINADDR;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECS is not 4KByte aligned or
    DS:TMP_SECINFO is not 64Byte aligned or TMP_LINADDR is not 4KByte aligned)
    THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
    THEN #PF(DS:TMP_SECS); FI;

SCRATCH_SECINFO := DS:TMP_SECINFO;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero or
```

```
    ! (SCRATCH_SECINFO.FLAGS.PT is PT_REG or SCRATCH_SECINFO.FLAGS.PT is PT_TCS or
    (SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
    (SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1)) )
    THEN #GP(0); FI;

(* If PT_SS_FIRST/PT_SS_REST page types are requested then CR4.CET must be 1 *)
IF ( (SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST OR
    SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST) AND CR4.CET == 0)
    THEN #GP(0); FI;

(* Check the EPC page for concurrency *)
IF (EPC page is not available for EADD)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                VMCS.Guest-linear_address := DS:RCX;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EADD)
    THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPGE[32767:0];

CASE (SCRATCH_SECINFO.FLAGS.PT)

    PT_TCS:
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
        IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and
            ((DS:TCS.FSLIMIT & 0FFFH ≠ 0FFFH) or (DS:TCS.GSLIMIT & 0FFFH ≠ 0FFFH) )) #GP(0); FI;
        (* Ensure TCS.PREVSSP is zero *)
        IF (CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1) and (DS:RCX.PREVSSP != 0) #GP(0); FI;
        BREAK;
    PT_REG:
        IF (SCRATCH_SECINFO.FLAGS.W = 1 and SCRATCH_SECINFO.FLAGS.R = 0) #GP(0); FI;
        BREAK;
    PT_SS_FIRST:
    PT_SS_REST:
    (* SS pages cannot created on first or last page of ELRANGE *)
```

```
    IF ( TMP_LINADDR = DS:TMP_SECS.BASEADDR or TMP_LINADDR = (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE - 0x1000) )
        THEN #GP(0); FI;
    IF ( DS:RCX[4087:0] != 0 ) #GP(0); FI;
    IF (SCRATCH_SECINFO.FLAGS.PT == PT_SS_FIRST)
        THEN
            (* Check that valid RSTORSSP token exists *)
            IF ( DS:RCX[4095:4088] != ((TMP_LINADDR + 0x1000) | DS:TMP_SECS.ATTRIBUTES.MODE64BIT) ) #GP(0); FI;
            (* Check the 8 bytes are zero *)
            IF ( DS:RCX[4095:4088] != 0 ) #GP(0); FI;
    FI;
    IF (SCRATCH_SECINFO.FLAGS.W = 0 OR SCRATCH_SECINFO.FLAGS.R = 0 OR
     SCRATCH_SECINFO.FLAGS.X = 1) #GP(0); FI;
        BREAK;
ESAC;

(* Check the enclave offset is within the enclave linear address space *)
IF (TMP_LINADDR < DS:TMP_SECS.BASEADDR or TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE)
    THEN #GP(0); FI;

(* Check concurrency of measurement resource*)
IF (Measurement being updated)
    THEN #GP(0); FI;

(* Check if the enclave to which the page will be added is already in Initialized state *)
IF (DS:TMP_SECS already initialized)
    THEN #GP(0); FI;

(* For TCS pages, force EPCM.rwx bits to 0 and no debug access *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        SCRATCH_SECINFO.FLAGS.R := 0;
        SCRATCH_SECINFO.FLAGS.W := 0;
        SCRATCH_SECINFO.FLAGS.X := 0;
        (DS:RCX).FLAGS.DBGOPTIN := 0; // force TCS.FLAGS.DBGOPTIN off
        DS:RCX.CSSA := 0;
        DS:RCX.AEP := 0;
        DS:RCX.STATE := 0;
FI;

(* Add enclave offset and security attributes to MRENCLAVE *)
TMP_ENCLAVEOFFSET := TMP_LINADDR - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] := 0000000044444145H; // "EADD"
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] := SCRATCH_SECINFO[375:0]; // 48 bytes
DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(* Add enclave offset and security attributes to MRENCLAVE *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_LINADDR;
```

```
(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
	If the TCS page PREVSSP field is not zero.
	If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
	If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
	If the TCS page PREVSSP field is not zero.
	If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
	If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

EAUG—Add a Page to an Initialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0DH ENCLS[EAUG]	IR	V/V	SGX2	This leaf function adds a page to an initialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EAUG (In)	Address of a PAGEINFO (In)	Address of the destination EPC page (In)

Description

This leaf function zeroes a page of EPC memory, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in the EPCM. As part of the association, the security attributes are configured to prevent access to the EPC page until a corresponding invocation of the EACCEPT leaf or EACCEPT-COPY leaf confirms the addition of the new page into the enclave. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EAUG leaf function.

EAUG Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Must be zero	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EAUG Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	The specified enclave offset is outside of the enclave address space.
The SECS has been initialized.	

Concurrency Restrictions

Table 38-10. Base Concurrency Restrictions of EAUG

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EAUG	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	

Table 38-11. Additional Concurrency Restrictions of EAUG

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EAUG	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGE-INFO.SECS	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EAUG Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
IF (DS:RBX.SECINFO is not 0)
    THEN
        IF (DS:TMP_SECINFO is not 64B aligned)
            THEN #GP(0); FI;
FI;

TMP_LINADDR := DS:RBX.LINADDR;

IF ( DS:TMP_SECS is not 4KByte aligned or TMP_LINADDR is not 4KByte aligned )
    THEN #GP(0); FI;

IF DS:RBX.SRCPAGE is not 0
    THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
    THEN #PF(DS:TMP_SECS); FI;

(* Check the EPC page for concurrency *)
```

```
IF (EPC page in use)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                VMCS.Guest-linear_address := DS:RCX;
                Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI:

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* copy SECINFO contents into a scratch SECINFO *)
IF (DS:RBX.SECINFO is 0)
    THEN
        (* allocate and initialize a new scratch SECINFO structure *)
        SCRATCH_SECINFO.PT := PT_REG;
        SCRATCH_SECINFO.R := 1;
        SCRATCH_SECINFO.W := 1;
        SCRATCH_SECINFO.X := 0;
        << zero out remaining fields of SCRATCH_SECINFO >>
    ELSE
        (* copy SECINFO contents into scratch SECINFO *)
        SCRATCH_SECINFO := DS:TMP_SECINFO;
        (* check SECINFO flags for misconfiguration *)
        (* reserved flags must be zero *)
        (* SECINFO.FLAGS.PT must either be PT_SS_FIRST, or PT_SS_REST *)
        IF ( (SCRATCH_SECINFO reserved fields are not 0) or
        CPUID.(EAX=12H, ECX=1):EAX[6] is 0) OR
         (SCRATCH_SECINFO.PT is not PT_SS_FIRST, or PT_SS_REST) OR
         ( (SCRATCH_SECINFO.FLAGS.R is 0) OR (SCRATCH_SECINFO.FLAGS.W is 0) OR (SCRATCH_SECINFO.FLAGS.X is 1) ) )
            THEN #GP(0); FI;
FI;
(* Check if PT_SS_FIRST/PT_SS_REST page types are requested then CR4.CET must be 1 *)
IF ( (SCRATCH_SECINFO.PT is PT_SS_FIRST OR SCRATCH_SECINFO.PT is PT_SS_REST) AND CR4.CET == 0 )
    THEN #GP(0); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EAUG)
    THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

(* Check if the enclave to which the page will be added is in the Initialized state *)
IF (DS:TMP_SECS is not initialized)
    THEN #GP(0); FI;
```

```
(* Check the enclave offset is within the enclave linear address space *)
IF ( (TMP_LINADDR < DS:TMP_SECS.BASEADDR) or (TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE) )
    THEN #GP(0); FI;

IF ( (SCRATCH_SECINFO.PT is PT_SS_FIRST OR SCRATCH_SECINFO.PT is PT_SS_REST) )
    THEN
        (* SS pages cannot created on first or last page of ELRANGE *)
        IF ( TMP_LINADDR == DS:TMP_SECS.BASEADDR OR
         TMP_LINADDR == (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE - 0x1000) )
            THEN
                #GP(0); FI;
FI;

(* Clear the content of EPC page*)
DS:RCX[32767:0] := 0;

IF (CPUID.(EAX=07H, ECX=0H):ECX[CET_SS] = 1)
    THEN
        (* set up shadow stack RSTORSSP token *)
        IF (SCRATCH_SECINFO.PT is PT_SS_FIRST)
        THEN
            DS:RCX[0xFF8] := (TMP_LINADDR + 0x1000) | TMP_SECS.ATTRIBUTES.MODE64BIT; FI;
FI;

(* Set EPCM security attributes *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_LINADDR;
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 1;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;

(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;

(* Set EPCM valid fields *)
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
	If the enclave is not initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form. If a memory operand is not properly aligned. If a memory operand is locked. If the enclave is not initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.

EBLOCK—Mark a page in EPC as Blocked

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLS[EBLOCK]	IR	V/V	SGX1	This leaf function marks a page in the EPC as blocked.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	EBLOCK (In)	Return error code (Out)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be marked as BLOCKED. This instruction can only be executed when current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

An error code is returned in RAX.

The table below provides additional information on the memory parameter of EBLOCK leaf function.

EBLOCK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 38-12. EBLOCK Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EBLOCK successful.
SGX_BLKSTATE	Page already blocked. This value is used to indicate to a VMM that the page was already in BLOCKED state as a result of EBLOCK and thus will need to be restored to this state when it is eventually reloaded (using ELDB).
SGX_ENTRYEPOCH_LOCKED	SECS locked for Entry Epoch update. This value indicates that an ETRACK is currently executing on the SECS. The EBLOCK should be reattempted.
SGX_NOTBLOCKABLE	Page type is not one which can be blocked.
SGX_PG_INVLD	Page is not valid and cannot be blocked.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB.

Concurrency Restrictions

Table 38-13. Base Concurrency Restrictions of EBLOCK

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EBLOCK	Target [DS:RCX]	Shared	SGX_EPC_PAGE_ CONFLICT	

Table 38-14. Additional Concurrency Restrictions of EBLOCK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EBLOCK	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EBLOCK Operational Flow

Name	Type	Size (Bits)	Description
TMP_BLKSTATE	Integer	64	Page is already blocked.

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

(* Check the EPC page for concurrency*)
IF (EPC page in use)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PG_INVLD;
        GOTO DONE;
FI;

IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_TRIM)
 and EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN
        RFLAGS.CF := 1;
        IF (EPCM(DS:RCX).PT = PT_SECS)
            THEN RAX := SGX_PG_IS_SECS;
            ELSE RAX := SGX_NOTBLOCKABLE;
        FI;
        GOTO DONE;
FI;

(* Check if the page is already blocked and report blocked state *)
TMP_BLKSTATE := EPCM(DS:RCX).BLOCKED;
```

```
(* at this point, the page must be valid and PT_TCS or PT_REG or PT_TRIM*)
IF (TMP_BLKSTATE = 1)
    THEN
        RFLAGS.CF := 1;
        RAX := SGX_BLKSTATE;
    ELSE
        EPCM(DS:RCX).BLOCKED := 1
FI;
DONE:
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Sets CF if page is BLOCKED or not blockable, otherwise cleared. Clears PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

ECREATE—Create an SECS page in the Enclave Page Cache

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLS[ECREATE]	IR	V/V	SGX1	This leaf function begins an enclave build by creating an SECS page in EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	ECREATE (In)	Address of a PAGEINFO (In)	Address of the destination SECS page (In)

Description

ENCLS[ECREATE] is the first instruction executed in the enclave build process. ECREATE copies an SECS structure outside the EPC into an SECS page inside the EPC. The internal structure of SECS is not accessible to software.

ECREATE will set up fields in the protected SECS and mark the page as valid inside the EPC. ECREATE initializes or checks unused fields.

Software sets the following fields in the source structure: SECS:BASEADDR, SECS:SIZE in bytes, ATTRIBUTES, CONFIGID, and CONFIGSVN. SECS:BASEADDR must be naturally aligned on an SECS.SIZE boundary. SECS.SIZE must be at least 2 pages (8192).

The source operand RBX contains an effective address of a PAGEINFO structure. PAGEINFO contains an effective address of a source SECS and an effective address of an SECINFO. The SECS field in PAGEINFO is not used.

The RCX register is the effective address of the destination SECS. It is an address of an empty slot in the EPC. The SECS structure must be page aligned. SECINFO flags must specify the page as an SECS page.

ECREATE Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

ECREATE will fault if the SECS target page is in use; already valid; outside the EPC. It will also fault if addresses are not aligned; unused PAGEINFO fields are not zero.

If the amount of space needed to store the SSA frame is greater than the amount specified in SECS.SSAFRAME-SIZE, a #GP(0) results. The amount of space needed for an SSA frame is computed based on DS:TMP_SECS.ATTRIBUTES.XFRM size. Details of computing the size can be found Section 39.7.

Concurrency Restrictions

Table 38-15. Base Concurrency Restrictions of ECREATE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ECREATE	SECS [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 38-16. Additional Concurrency Restrictions of ECREATE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ECREATE	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ECREATE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the SECS source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the SECS page to be added.
TMP_XSIZE	SSA Size	64	The size calculation of SSA frame.
TMP_MISC_SIZE	MISC Field Size	64	Size of the selected MISC field components.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECINFO := DS:RBX.SECINFO;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECINFO is not 64Byte aligned)
    THEN #GP(0); FI;

IF (DS:RBX.LINADDR ! = 0 or DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

(* Check for misconfigured SECINFO flags*)
IF (DS:TMP_SECINFO reserved fields are not zero or DS:TMP_SECINFO.FLAGS.PT ≠ PT_SECS)
    THEN #GP(0); FI;

TMP_SECS := RCX;

IF (EPC entry in use)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
```

```
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                    VMCS.Exit_qualification.error := 0;
                    VMCS.Guest-physical_address :=
                        << translation of DS:TMP_SECS produced by paging >>;
                    VMCS.Guest-linear_address := DS:TMP_SECS;
                Deliver VMEXIT;
                ELSE
                    #GP(0);
            FI;
FI;

IF (EPC entry in use)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPGE[32767:0];

(* Check lower 2 bits of XFRM are set *)
IF ( ( DS:TMP_SECS.ATTRIBUTES.XFRM BitwiseAND 03H) ≠ 03H)
    THEN #GP(0); FI;

IF (XFRM is illegal)
    THEN #GP(0); FI;

(* Check legality of CET_ATTRIBUTES *)
IF ((DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_ATTRIBUTES ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[5:2] ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[1:0] ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1 and
     (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) not canonical) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0 and
     (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) & 0xFFFFFFFF00000000) ||
    (DS:TMP_SECS.CET_ATTRIBUTES.reserved fields not 0) or
     (DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) is not page aligned))
    THEN
        #GP(0);
FI;

(* Make sure that the SECS does not have any unsupported MISCSELECT options*)
IF ( !(CPUID.(EAX=12H, ECX=0):EBX[31:0] & DS:TMP_SECS.MISCSELECT[31:0]) )
    THEN
        EPCM(DS:TMP_SECS).EntryLock.Release();
        #GP(0);
FI;

( * Compute size of MISC area *)
TMP_MISC_SIZE := compute_misc_region_size();

(* Compute the size required to save state of the enclave on async exit, see Section 39.7.2.2*)
```

```
TMP_XSIZE := compute_xsave_size(DS:TMP_SECS.ATTRIBUTES.XFRM) + GPR_SIZE + TMP_MISC_SIZE;

(* Ensure that the declared area is large enough to hold XSAVE and GPR stat *)
IF ( DS:TMP_SECS.SSAFRAMESIZE*4096 < TMP_XSIZE)
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.BASEADDR is not canonical) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.BASEADDR and 0FFFFFFFF00000000H) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[7:0]) ) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[15:8]) ) )
    THEN #GP(0); FI;

(* Enclave size must be at least 8192 bytes and must be power of 2 in bytes*)
IF (DS:TMP_SECS.SIZE < 8192 or popcnt(DS:TMP_SECS.SIZE) > 1)
    THEN #GP(0); FI;

(* Ensure base address of an enclave is aligned on size*)
IF ( ( DS:TMP_SECS.BASEADDR and (DS:TMP_SECS.SIZE-1) ) )
    THEN #GP(0); FI;

(* Ensure the SECS does not have any unsupported attributes*)
IF ( DS:TMP_SECS.ATTRIBUTES and (~CR_SGX_ATTRIBUTES_MASK) )
    THEN #GP(0); FI;

IF ( DS:TMP_SECS reserved fields are not zero)
    THEN #GP(0); FI;

(* Verify that CONFIGID/CONFIGSVN are not set with attribute *)
IF ( ((DS:TMP_SECS.CONFIGID ≠ 0) or (DS:TMP_SECS.CONFIGSVN ≠0)) AND (DS:TMP_SECS.ATTRIBUTES.KSS == 0 ))
    THEN #GP(0); FI;

Clear DS:TMP_SECS to Uninitialized;
DS:TMP_SECS.MRENCLAVE := SHA256INITIALIZE(DS:TMP_SECS.MRENCLAVE);
DS:TMP_SECS.ISVSVN := 0;
DS:TMP_SECS.ISVPRODID := 0;

(* Initialize hash updates etc*)
Initialize enclave's MRENCLAVE update counter;

(* Add "ECREATE" string and SECS fields to MRENCLAVE *)
TMPUPDATEFIELD[63:0] := 0045544145524345H; // "ECREATE"
TMPUPDATEFIELD[95:64] := DS:TMP_SECS.SSAFRAMESIZE;
TMPUPDATEFIELD[159:96] := DS:TMP_SECS.SIZE;
IF (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 1)
    THEN
        TMPUPDATEFIELD[223:160] := DS:TMP_SECS.CET_LEG_BITMAP_OFFSET;
    ELSE
        TMPUPDATEFIELD[223:160] := 0;
```

```
FI;
TMPUPDATEFIELD[511:160] := 0;
DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(* Set EID *)
DS:TMP_SECS.EID := LockedXAdd(CR_NEXT_EID, 1);

(* Initialize the virtual child count to zero *)
DS:TMP_SECS.VIRTCHILDCNT := 0;

(* Load ENCLAVECONTEXT with Address out of paging of SECS *)
<< store translation of DS:RCX produced by paging in SECS(DS:RCX).ENCLAVECONTEXT >>

(* Set the EPCM entry, first create SECS identifier and store the identifier in EPCM *)
EPCM(DS:TMP_SECS).PT := PT_SECS;
EPCM(DS:TMP_SECS).ENCLAVEADDRESS := 0;
EPCM(DS:TMP_SECS).R := 0;
EPCM(DS:TMP_SECS).W := 0;
EPCM(DS:TMP_SECS).X := 0;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the reserved fields are not zero.
	If PAGEINFO.SECS is not zero.
	If PAGEINFO.LINADDR is not zero.
	If the SECS destination is locked.
	If SECS.SSAFRAMESIZE is insufficient.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the SECS destination is outside the EPC.

64-Bit Mode Exceptions

#GP(0)	If a memory address is non-canonical form.
	If a memory operand is not properly aligned.
	If the reserved fields are not zero.
	If PAGEINFO.SECS is not zero.
	If PAGEINFO.LINADDR is not zero.
	If the SECS destination is locked.
	If SECS.SSAFRAMESIZE is insufficient.

#PF(error code)	If a page fault occurs in accessing memory operands. If the SECS destination is outside the EPC.

EDBGRD—Read From a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLS[EDBGRD]	IR	V/V	SGX1	This leaf function reads a dword/quadword from a debug enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDBGRD (In)	Return error code (Out)	Data read from a debug enclave (Out)	Address of source memory in the EPC (In)

Description

This leaf function copies a quadword/doubleword from an EPC page belonging to a debug enclave into the RBX register. Eight bytes are read in 64-bit mode, four bytes are read in non-64-bit modes. The size of data read cannot be overridden.

The effective address of the source location inside the EPC is provided in the register RCX.

EDBGRD Memory Parameter Semantics

EPCQW
Read access permitted by Enclave

The error codes are:

Table 38-17. EDBGRD Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EDBGRD successful.
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state.

The instruction faults if any of the following:

EDBGRD Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is beyond the architectural size of the TCS (SGX_TCS_LIMIT).
An operand causing any segment violation.	May page fault.
CPL > 0.	

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDBGRD does not result in a #GP.

Concurrency Restrictions

Table 38-18. Base Concurrency Restrictions of EDBGRD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDBGRD	Target [DS:RCX]	Shared	#GP	

Table 38-19. Additional Concurrency Restrictions of EDBGRD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGRD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGRD Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1))
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ( (TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned) )
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing the same EPCM entry *)
IF (Another instruction modifying the same EPCM entry is executing)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (SOURCE) is pointing to a PT_REG or PT_TCS or PT_VA or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_VA)
and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX points to an accessible EPC page *)
IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
```

```
        RAX := SGX_PAGE_NOT_DEBUGGABLE;
        GOTO DONE;
FI;

(* If source is a TCS, then make sure that the offset into the page is not beyond the TCS size*)
IF ( ( EPCM(DS:RCX). PT = PT_TCS) and ((DS:RCX) & FFFH ≥ SGX_TCS_LIMIT) )
    THEN #GP(0); FI;

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF ( (EPCM(DS:RCX).PT = PT_REG) or (EPCM(DS:RCX).PT = PT_TCS) )
    THEN
        TMP_SECS := GET_SECS_ADDRESS;
        IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
            THEN #GP(0); FI;
        IF ( (TMP_MODE64 = 1) )
            THEN RBX[63:0] := (DS:RCX)[63:0];
            ELSE EBX[31:0] := (DS:RCX)[31:0];
        FI;
    ELSE
        TMP_64BIT_VAL[63:0] := (DS:RCX)[63:0] & (~07H); // Read contents from VA slot
        IF (TMP_MODE64 = 1)
            THEN
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN RBX[63:0] := 0FFFFFFFFFFFFFFFFH;
                    ELSE RBX[63:0] := 0H;
                FI;
            ELSE
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN EBX[31:0] := 0FFFFFFFFH;
                    ELSE EBX[31:0] := 0H;
                FI;
FI;

(* clear EAX and ZF to indicate successful completion *)
RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is set if the page is MODIFIED or PENDING; RAX contains the error code. Otherwise ZF is cleared and RAX is set to 0. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights.
	If DS segment is unusable.
	If RCX points to a memory location not 4Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA.
	If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.

#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form. If RCX points to a memory location not 8Byte-aligned. If the address in RCX points to a page belonging to a non-debug enclave. If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA. If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.
#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

EDBGWR—Write to a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLS[EDBGWR]	IR	V/V	SGX1	This leaf function writes a dword/quadword to a debug enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDBGWR (In)	Return error code (Out)	Data to be written to a debug enclave (In)	Address of Target memory in the EPC (In)

Description

This leaf function copies the content in EBX/RBX to an EPC page belonging to a debug enclave. Eight bytes are written in 64-bit mode, four bytes are written in non-64-bit modes. The size of data cannot be overridden.

The effective address of the target location inside the EPC is provided in the register RCX.

EDBGWR Memory Parameter Semantics

EPCQW
Write access permitted by Enclave

The instruction faults if any of the following:

EDBGWR Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is not the FLAGS word.
An operand causing any segment violation.	May page fault.
CPL > 0.	

The error codes are:

Table 38-20. EDBGWR Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EDBGWR successful.
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state.

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDBGRD does not result in a #GP.

Concurrency Restrictions

Table 38-21. Base Concurrency Restrictions of EDBGWR

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDBGWR	Target [DS:RCX]	Shared	#GP	

Table 38-22. Additional Concurrency Restrictions of EDBGWR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGWR	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGWR Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1)).
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ( (TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned) )
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing the same EPCM entry *)
IF (Another instruction modifying the same EPCM entry is executing)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS)
  and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX points to an accessible EPC page *)
IF ( (EPCM(DS:RCX).PENDING is not 0) or (EPCM(DS:RCS).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
```

```
        RAX := SGX_PAGE_NOT_DEBUGGABLE;
        GOTO DONE;
FI;

(* If destination is a TCS, then make sure that the offset into the page can only point to the FLAGS field*)
IF ( ( EPCM(DS:RCX). PT = PT_TCS) and ((DS:RCX) & FF8H ≠ offset_of_FLAGS & 0FF8H) )
    THEN #GP(0); FI;

(* Locate the SECS for the enclave to which the DS:RCX page belongs *)
TMP_SECS := GET_SECS_PHYS_ADDRESS(EPCM(DS:RCX).ENCLAVESECS);

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 1) )
    THEN (DS:RCX)[63:0] := RBX[63:0];
    ELSE (DS:RCX)[31:0] := EBX[31:0];
FI;

(* clear EAX and ZF to indicate successful completion *)
RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0
```

Flags Affected

ZF is set if the page is MODIFIED or PENDING; RAX contains the error code. Otherwise ZF is cleared and RAX is set to 0. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights.
	If DS segment is unusable.
	If RCX points to a memory location not 4Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a location inside TCS that is not the FLAGS word.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.
	If RCX points to a memory location not 8Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a location inside TCS that is not the FLAGS word.

#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

EEXTEND—Extend Uninitialized Enclave Measurement by 256 Bytes

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLS[EEXTEND]	IR	V/V	SGX1	This leaf function measures 256 bytes of an uninitialized enclave page.

Instruction Operand Encoding

Op/En	EAX	EBX	RCX
IR	EEXTEND (In)	Effective address of the SECS of the data chunk (In)	Effective address of a 256-byte chunk in the EPC (In)

Description

This leaf function updates the MRENCLAVE measurement register of an SECS with the measurement of an EXTEND string compromising of "EEXTEND" || ENCLAVEOFFSET || PADDING || 256 bytes of the enclave page. This instruction can only be executed when current privilege level is 0 and the enclave is uninitialized.

RBX contains the effective address of the SECS of the region to be measured. The address must be the same as the one used to add the page into the enclave.

RCX contains the effective address of the 256 byte region of an EPC page to be measured. The DS segment is used to create linear addresses. Segment override is not supported.

EEXTEND Memory Parameter Semantics

EPC[RCX]
Read access by Enclave

The instruction faults if any of the following:

EEXTEND Faulting Conditions

RBX points to an address not 4KBytes aligned.	RBX does not resolve to an SECS.
RBX does not point to an SECS page.	RBX does not point to the SECS page of the data chunk.
RCX points to an address not 256B aligned.	RCX points to an unused page or a SECS.
RCX does not resolve in an EPC page.	If SECS is locked.
If the SECS is already initialized.	May page fault.
CPL > 0.	

Concurrency Restrictions

Table 38-23. Base Concurrency Restrictions of EEXTEND

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EEXTEND	Target [DS:RCX]	Shared	#GP	
	SECS [DS:RBX]	Concurrent		

Table 38-24. Additional Concurrency Restrictions of EEXTEND

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EEXTEND	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EEXTEND Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF (DS:RBX is not 4096 Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve to an EPC page)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX is not 256Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing EPCM *)
IF (Other instructions accessing EPCM)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS)
  and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN #PF(DS:RCX); FI;

TMP_SECS := Get_SECS_ADDRESS();

IF (DS:RBX does not resolve to TMP_SECS)
    THEN #GP(0); FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)
IF ( (Other instruction accessing MRENCLAVE) or (Other instructions checking or updating the initialized state of the SECS))
```

```
    THEN #GP(0); FI;

(* Calculate enclave offset *)
TMP_ENCLAVEOFFSET := EPCM(DS:RCX).ENCLAVEADDRESS - TMP_SECS.BASEADDR;
TMP_ENCLAVEOFFSET := TMP_ENCLAVEOFFSET + (DS:RCX & 0FFFH)

(* Add EEXTEND message and offset to MRENCLAVE *)
TMPUPDATEFIELD[63:0] := 00444E4554584545H; // "EEXTEND"
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] := 0; // 48 bytes
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(*Add 256 bytes to MRENCLAVE, 64 byte at a time *)
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[511:0] );
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1023: 512] );
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1535: 1024] );
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[2047: 1536] );
INC enclave's MRENCLAVE update counter by 4;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If the address in RBX is outside the DS segment limit.
	If RBX points to an SECS page which is not the SECS of the data chunk.
	If the address in RCX is outside the DS segment limit.
	If RCX points to a memory location not 256Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RBX is non-canonical form.
	If RBX points to an SECS page which is not the SECS of the data chunk.
	If RCX is non-canonical form.
	If RCX points to a memory location not 256 Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

EINIT—Initialize an Enclave for Execution

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLS[EINIT]	IR	V/V	SGX1	This leaf function initializes the enclave and makes it ready to execute enclave code.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EINIT (In)	Error code (Out)	Address of SIGSTRUCT (In)	Address of SECS (In)	Address of EINITTOKEN (In)

Description

This leaf function is the final instruction executed in the enclave build process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to start user code execution using the EENTER instruction.

EINIT takes the effective address of a SIGSTRUCT and EINITTOKEN. The SIGSTRUCT describes the enclave including MRENCLAVE, ATTRIBUTES, ISVSVN, a 3072 bit RSA key, and a signature using the included key. SIGSTRUCT must be populated with two values, q1 and q2. These are calculated using the formulas shown below:

q1 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

The EINITTOKEN contains the MRENCLAVE, MRSIGNER, and ATTRIBUTES. These values must match the corresponding values in the SECS. If the EINITTOKEN was created with a debug launch key, the enclave must be in debug mode as well.

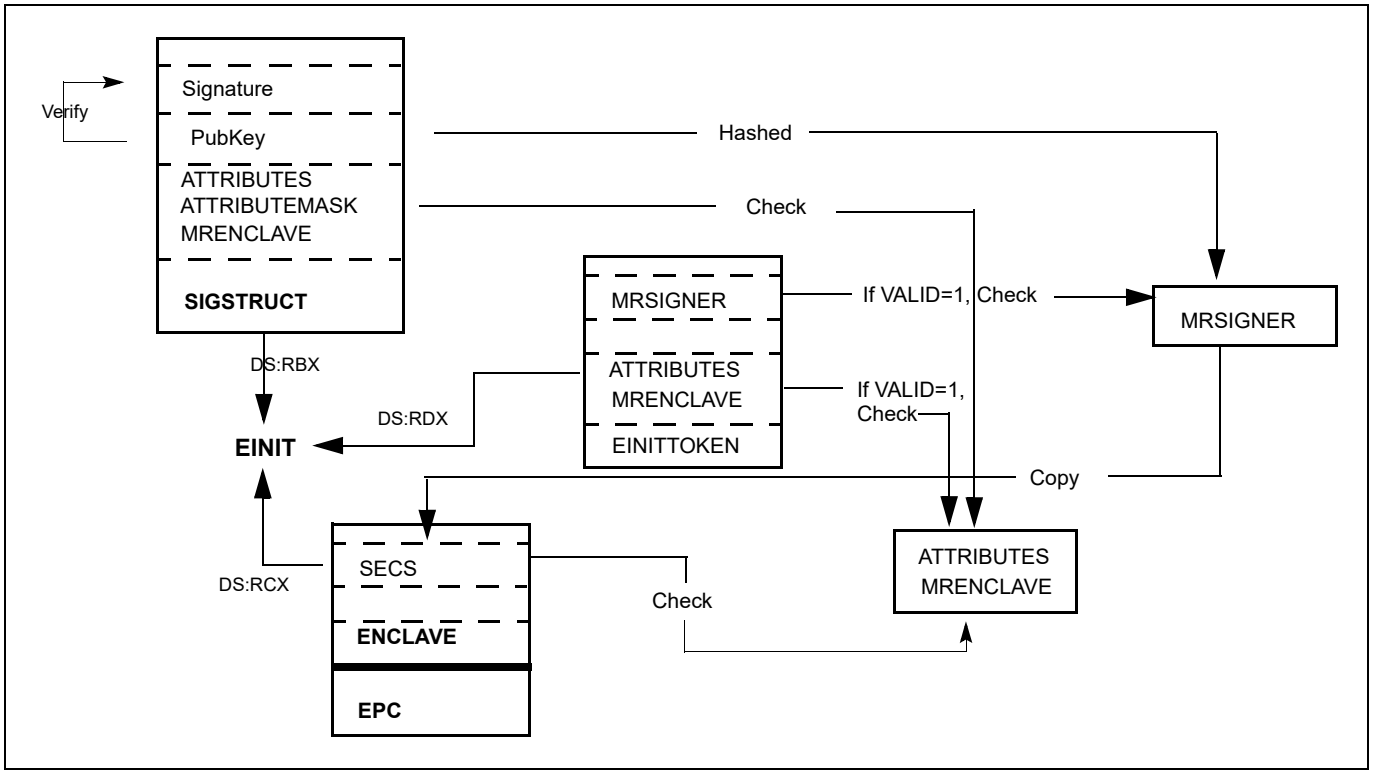

Figure 38-1. Relationships Between SECS, SIGSTRUCT, and EINITTOKEN

EINIT Memory Parameter Semantics

SIGSTRUCT	SECS	EINITTOKEN
Access by non-Enclave	Read/Write access by Enclave	Access by non-Enclave

EINIT performs the following steps, which can be seen in Figure 38-1:

1. Validates that SIGSTRUCT is signed using the enclosed public key.
2. Checks that the completed computation of SECS.MRENCLAVE equals SIGSTRUCT.HASHENCLAVE.
3. Checks that no controlled ATTRIBUTES bits are set in SIGSTRUCT.ATTRIBUTES unless the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
4. Checks that the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SIGSTRUCT.ATTRIBUTES equals the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.
5. If EINITTOKEN.VALID is 0, checks that the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
6. If EINITTOKEN.VALID is 1, checks the validity of EINITTOKEN.
7. If EINITTOKEN.VALID is 1, checks that EINITTOKEN.MRENCLAVE equals SECS.MRENCLAVE.
8. If EINITTOKEN.VALID is 1 and EINITTOKEN.ATTRIBUTES.DEBUG is 1, SECS.ATTRIBUTES.DEBUG must be 1.
9. Commits SECS.MRENCLAVE, and sets SECS.MRSIGNER, SECS.ISVSVN, and SECS.ISVPRODID based on SIGSTRUCT.
10. Update the SECS as Initialized.

Periodically, EINIT polls for certain asynchronous events. If such an event is detected, it completes with failure code (ZF=1 and RAX = SGX_UNMASKED_EVENT), and RIP is incremented to point to the next instruction. These events includes external interrupts, non-maskable interrupts, system-management interrupts, machine checks, INIT signals, and the VMX-preemption timer. EINIT does not fail if the pending event is inhibited (e.g., external interrupts could be inhibited due to blocking by MOV SS blocking or by STI).

The following bits in RFLAGS are cleared: CF, PF, AF, OF, and SF. When the instruction completes with an error, RFLAGS.ZF is set to 1, and the corresponding error bit is set in RAX. If no error occurs, RFLAGS.ZF is cleared and RAX is set to 0.

The error codes are:

Table 38-25. EINIT Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EINIT successful.
SGX_INVALID_SIG_STRUCT	If SIGSTRUCT contained an invalid value.
SGX_INVALID_ATTRIBUTE	If SIGSTRUCT contains an unauthorized attributes mask.
SGX_INVALID_MEASUREMENT	If SIGSTRUCT contains an incorrect measurement. If EINITTOKEN contains an incorrect measurement.
SGX_INVALID_SIGNATURE	If signature does not validate with enclosed public key.
SGX_INVALID_LICENSE	If license is invalid.
SGX_INVALID_CPUSVN	If license SVN is unsupported.
SGX_UNMASKED_EVENT	If an unmasked event is received before the instruction completes its operation.

Concurrency Restrictions

Table 38-26. Base Concurrency Restrictions of EINIT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EINIT	SECS [DS:RCX]	Shared	#GP	

Table 38-27. Additional Concurrency Restrictions of ENIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EINIT	SECS [DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EINIT Operational Flow

Name	Type	Size	Description
TMP_SIG	SIGSTRUCT	1808Bytes	Temp space for SIGSTRUCT.
TMP_TOKEN	EINITTOKEN	304Bytes	Temp space for EINITTOKEN.
TMP_MRENCLAVE		32Bytes	Temp space for calculating MRENCLAVE.
TMP_MRSIGNER		32Bytes	Temp space for calculating MRSIGNER.
CONTROLLED_ATTRIBUTES	ATTRIBUTES	16Bytes	Constant mask of all ATTRIBUTE bits that can only be set for authorized enclaves.
TMP_KEYDEPENDENCIES	Buffer	224Bytes	Temp space for key derivation.
TMP_EINITTOKENKEY		16Bytes	Temp space for the derived EINITTOKEN Key.
TMP_SIG_PADDING	PKCS Padding Buffer	352Bytes	The value of the top 352 bytes from the computation of Signature^3 modulo MRSIGNER.

```
(* make sure SIGSTRUCT and SECS are aligned *)
IF ( (DS:RBX is not 4KByte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* make sure the EINITTOKEN is aligned *)
IF (DS:RDX is not 512Byte Aligned)
    THEN #GP(0); FI;

(* make sure the SECS is inside the EPC *)
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SIG[14463:0] := DS:RBX[14463:0]; // 1808 bytes
TMP_TOKEN[2423:0] := DS:RDX[2423:0]; // 304 bytes
```

```
(* Verify SIGSTRUCT Header. *)
IF ( (TMP_SIG.HEADER ≠ 06000000E10000000000010000000000h) or
     ((TMP_SIG.VENDOR ≠ 0) and (TMP_SIG.VENDOR ≠ 00008086h) ) or
     (TMP_SIG HEADER2 ≠ 01010000600000006000000001000000h) or
     (TMP_SIG.EXPONENT   ≠ 00000003h) or (Reserved space is not 0's) )
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_INVALID_SIG_STRUCT;
          GOTO EXIT;
FI;

(* Open "Event Window" Check for Interrupts. Verify signature using embedded public key, q1, and q2. Save upper 352 bytes of the
PKCS1.5 encoded message into the TMP_SIG_PADDING*)
IF (interrupt was pending) THEN
     RFLAGS.ZF := 1;
     RAX := SGX_UNMASKED_EVENT;
     GOTO EXIT;
FI
IF (signature failed to verify) THEN
     RFLAGS.ZF := 1;
     RAX := SGX_INVALID_SIGNATURE;
     GOTO EXIT;
FI;
(*Close "Event Window" *)

(* make sure no other Intel SGX instruction is modifying SECS*)
IF (Other instructions modifying SECS)
     THEN #GP(0); FI;

IF ( (EPCM(DS:RCX). VALID = 0) or (EPCM(DS:RCX).PT ≠ PT_SECS) )
     THEN #PF(DS:RCX); FI;

(* Verify ISVFAMILYID is not used on an enclave with KSS disabled *)
IF ((TMP_SIG.ISVFAMILYID != 0) AND (DS:RCX.ATTRIBUTES.KSS == 0))
     THEN
       RFLAGS.ZF := 1;
       RAX := SGX_INVALID_SIG_STRUCT;
       GOTO EXIT;
FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)
IF ( (Other instruction modifying MRENCLAVE) or (Other instructions modifying the SECS's Initialized state))
     THEN #GP(0); FI;

(* Calculate finalized version of MRENCLAVE *)
(* SHA256 algorithm requires one last update that compresses the length of the hashed message into the output SHA256 digest *)
TMP_ENCLAVE := SHA256FINAL( (DS:RCX).MRENCLAVE, enclave's MRENCLAVE update count *512);

(* Verify MRENCLAVE from SIGSTRUCT *)
IF (TMP_SIG.ENCLAVEHASH ≠ TMP_MRENCLAVE)
     RFLAGS.ZF := 1;
     RAX := SGX_INVALID_MEASUREMENT;
     GOTO EXIT;
FI;
```

```
TMP_MRSIGNER := SHA256(TMP_SIG.MODULUS)

(* if controlled ATTRIBUTES are set, SIGSTRUCT must be signed using an authorized key *)
CONTROLLED_ATTRIBUTES := 0000000000000020H;
IF ( ( (DS:RCX.ATTRIBUTES & CONTROLLED_ATTRIBUTES) ≠ 0) and (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;
FI;

(* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
IF ( (DS:RCX.ATTRIBUTES & TMP_SIG.ATTRIBUTEMASK) ≠ (TMP_SIG.ATTRIBUTE & TMP_SIG.ATTRIBUTEMASK) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;
FI;

( *Verify SIGSTRUCT.MISCSELECT requirements are met *)
IF ( (DS:RCX.MISCSELECT & TMP_SIG.MISCMASK) ≠ (TMP_SIG.MISCSELECT & TMP_SIG.MISCMASK) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    IF ( DS:RCX.CET_ATTRIBUTES & TMP_SIG.CET_ATTRIBUTES_MASK ≠ TMP_SIG.CET_ATTRIBUTES &
     TMP_SIG.CET_ATTRIB-UTES_MASK )
        THEN
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_ATTRIBUTE;
            GOTO EXIT
    FI;
FI;

(* If EINITTOKEN.VALID[0] is 0, verify the enclave is signed by an authorized key *)
IF (TMP_TOKEN.VALID[0] = 0)
    IF (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH)
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_EINITTOKEN;
        GOTO EXIT;
    FI;
    GOTO COMMIT;
FI;

(* Debug Launch Enclave cannot launch Production Enclaves *)
IF ( (DS:RDX.MASKEDATTRIBUTESLE.DEBUG = 1) and (DS:RCX.ATTRIBUTES.DEBUG = 0) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;
```

```
(* Check reserve space in EINIT token includes reserved regions and upper bits in valid field *)
IF (TMP_TOKEN reserved space is not clear)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;

(* EINIT token must not have been created by a configuration beyond the current CPU configuration *)
IF (TMP_TOKEN.CPUSVN must not be a configuration beyond CR_CPUSVN)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_CPUSVN;
    GOTO EXIT;
FI;

(* Derive Launch key used to calculate EINITTOKEN.MAC *)
HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;
HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;

TMP_KEYDEPENDENCIES.KEYNAME := EINITTOKEN_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := TMP_TOKEN.ISVPRODIDLE;
TMP_KEYDEPENDENCIES.ISVSVN := TMP_TOKEN.ISVSVNLE;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_TOKEN.MASKEDATTRIBUTESLE;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
TMP_KEYDEPENDENCIES.MRSIGNER := IA32_SGXLEPUBKEYHASH;
TMP_KEYDEPENDENCIES.KEYID := TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := TMP_TOKEN.CPUSVNLE;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_TOKEN.MASKEDMISCSELECTLE;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := 0;
TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_TOKEN.CET_MASKED_ATTRIBUTES_ LE;
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
FI;

(* Calculate the derived key*)
TMP_EINITTOKENKEY := derivekey(TMP_KEYDEPENDENCIES);

(* Verify EINITTOKEN was generated using this CPU's Launch key and that it has not been modified since issuing by the Launch
Enclave. Only 192 bytes of EINITTOKEN are CMACed *)
IF (TMP_TOKEN.MAC ≠ CMAC(TMP_EINITTOKENKEY, TMP_TOKEN[1535:0] ) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;
```

```
(* Verify EINITTOKEN (RDX) is for this enclave *)
IF ( (TMP_TOKEN.MRENCLAVE ≠ TMP_MRENCLAVE) or (TMP_TOKEN.MRSIGNER ≠ TMP_MRSIGNER) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_MEASUREMENT;
    GOTO EXIT;
FI;

(* Verify ATTRIBUTES in EINITTOKEN are the same as the enclave's *)
IF (TMP_TOKEN.ATTRIBUTES ≠ DS:RCX.ATTRIBUTES)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINIT_ATTRIBUTE;
    GOTO EXIT;
FI;

COMMIT:
(* Commit changes to the SECS; Set ISVPRODID, ISVSVN, MRSIGNER, INIT ATTRIBUTE fields in SECS (RCX) *)
DS:RCX.MRENCLAVE := TMP_MRENCLAVE;
(* MRSIGNER stores a SHA256 in little endian implemented natively on x86 *)
DS:RCX.MRSIGNER := TMP_MRSIGNER;
DS:RCX.ISVEXTPRODID := TMP_SIG.ISVEXTPRODID;
DS:RCX.ISVPRODID := TMP_SIG.ISVPRODID;
DS:RCX.ISVSVN := TMP_SIG.ISVSVN;
DS:RCX.ISVFAMILYID := TMP_SIG.ISVFAMILYID;
DS:RCX.PADDING := TMP_SIG_PADDING;

(* Mark the SECS as initialized *)
Update DS:RCX to initialized;

(* Set RAX and ZF for success*)
    RFLAGS.ZF := 0;
    RAX := 0;
EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set and RAX contains the error code. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand is not properly aligned.
	If another instruction is modifying the SECS.
	If the enclave is already initialized.
	If the SECS.MRENCLAVE is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RCX does not resolve in an EPC page.
	If the memory address is not a valid, uninitialized SECS.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is not properly aligned.
	If another instruction is modifying the SECS.
	If the enclave is already initialized.
	If the SECS.MRENCLAVE is in use.

#PF(error code)	If a page fault occurs in accessing memory operands. If RCX does not resolve in an EPC page. If the memory address is not a valid, uninitialized SECS.

ELDB/ELDU/ELDBC/ELDUC—Load an EPC Page and Mark its State

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLS[ELDB]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as blocked.
EAX = 08H ENCLS[ELDU]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as unblocked.
EAX = 12H ENCLS[ELDBC]	IR	V/V	EAX[6]	This leaf function behaves lie ELDB but with improved conflict handling for oversubscription.
EAX = 13H ENCLS[ELDUC]	IR	V/V	EAX[6]	This leaf function behaves like ELDU but with improved conflict handling for oversubscription.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	ELDB/ELDU (In)	Return error code (Out)	Address of the PAGEINFO (In)	Address of the EPC page (In)	Address of the version-array slot (In)

Description

This leaf function copies a page from regular main memory to the EPC. As part of the copying process, the page is cryptographically authenticated and decrypted. This instruction can only be executed when current privilege level is 0.

The ELDB leaf function sets the BLOCK bit in the EPCM entry for the destination page in the EPC after copying. The ELDU leaf function clears the BLOCK bit in the EPCM entry for the destination page in the EPC after copying.

RBX contains the effective address of a PAGEINFO structure; RCX contains the effective address of the destination EPC page; RDX holds the effective address of the version array slot that holds the version of the page.

The ELDBC/ELDUC leafs are very similar to ELDB and ELDU. They provide an error code on the concurrency conflict for any of the pages which need to acquire a lock. These include the destination, SECS, and VA slot.

The table below provides additional information on the memory parameter of ELDB/ELDU leaf functions.

ELDB/ELDU/ELDBC/ELBUC Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	PAGEINFO.SECS	EPCPAGE	Version-Array Slot
Non-enclave read access	Non-enclave read access	Non-enclave read access	Enclave read/write access	Read/Write access permitted by Enclave	Read/Write access permitted by Enclave

The error codes are:

Table 38-28. ELDB/ELDU/ELDBC/ELBUC Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	ELDB/ELDU successful.
SGX_MAC_COMPARE_FAIL	If the MAC check fails.

Concurrency Restrictions

Table 38-29. Base Concurrency Restrictions of ELDB/ELDU/ELDBC/ELBUC

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ELDB/ELDU	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA [DS:RDX]	Shared	#GP	
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	
ELDBC/ELBUC	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
	VA [DS:RDX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS [DS:RBX]PAGEINFO.SECS	Shared	SGX_EPC_PAGE_CONFLICT	

Table 38-30. Additional Concurrency Restrictions of ELDB/ELDU/ELDBC/ELBUC

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ELDB/ELDU	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
ELDBC/ELBUC	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ELDB/ELDU/ELDBC/ELBUC Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Memory page	4KBytes	
TMP_SECS	Memory page	4KBytes	
TMP_PCMD	PCMD	128 Bytes	
TMP_HEADER	MACHEADER	128 Bytes	
TMP_VER	UINT64	64	
TMP_MAC	UINT128	128	
TMP_PK	UINT128	128	Page encryption/MAC key.
SCRATCH_PCMD	PCMD	128 Bytes	

```
(* Check PAGEINFO and EPCPAGE alignment *)
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;
```

```
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check VASLOT alignment *)
IF (DS:RDX is not 8Byte aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_PCMD := DS:RBX.PCMD;

(* Check alignment of PAGEINFO (RBX) linked parameters. Note: PCMD pointer is overlaid on top of PAGEINFO.SECINFO field *)
IF ( (DS:TMP_PCMD is not 128Byte aligned) or (DS:TMP_SRCPGE is not 4KByte aligned) )
    THEN #GP(0); FI;

(* Check concurrency of EPC by other Intel SGX instructions *)
IF (other instructions accessing EPC)
    THEN
    IF ((EAX==07h) OR (EAX==08h))   (* ELDB/ELDU *)
        THEN
            IF (<<VMX non-root operation>> AND
                    <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
                THEN
                    VMCS.Exit_reason := SGX_CONFLICT;
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                    VMCS.Exit_qualification.error := 0;
                    VMCS.Guest-physical_address :=
                << translation of DS:RCX produced by paging >>;
                    VMCS.Guest-linear_address := DS:RCX;
                    Deliver VMEXIT;
                ELSE
                    #GP(0);
            FI;
        ELSE (* ELDBC/ELDUC *)
            IF (<<VMX non-root operation>> AND
                    <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
                THEN
                    VMCS.Exit_reason := SGX_CONFLICT;
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_ERROR;
                    VMCS.Exit_qualification.error := SGX_EPC_PAGE_CONFLICT;
                    VMCS.Guest-physical_address :=
                << translation of DS:RCX produced by paging >>;
                    VMCS.Guest-linear_address := DS:RCX;
                    Deliver VMEXIT;
                ELSE
            RFLAGS.ZF := 1;
              RFLAGS.CF := 0;
            RAX := SGX_EPC_PAGE_CONFLICT;
            GOTO ERROR_EXIT;
            FI;
```

```
            FI;
FI;

(* Check concurrency of EPC and VASLOT by other Intel SGX instructions *)
IF (Other instructions modifying VA slot) THEN
    IF ((EAX==07h) OR (EAX==08h)) (* ELDB/ELDU *)
            THEN #GP(0);
    ELSE (* ELDBC/ELDUC *)
            RFLAGS.ZF := 1;
            RFLAGS.CF := 0;
            RAX := SGX_EPC_PAGE_CONFLICT;
            GOTO ERROR_EXIT;
    FI;
FI;

(* Verify EPCM attributes of EPC page, VA, and SECS *)
IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~0FFFH).PT ≠ PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Copy PCMD into scratch buffer *)
SCRATCH_PCMD[1023: 0] := DS:TMP_PCMD[1023:0];

(* Zero out TMP_HEADER*)
TMP_HEADER[sizeof(TMP_HEADER)-1: 0] := 0;

TMP_HEADER.SECINFO := SCRATCH_PCMD.SECINFO;
TMP_HEADER.RSVD := SCRATCH_PCMD.RSVD;
TMP_HEADER.LINADDR := DS:RBX.LINADDR;

(* Verify various attributes of SECS parameter *)
IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
    THEN
        IF ( DS:TMP_SECS is not 4KByte aligned)
            THEN #GP(0) FI;
        IF (DS:TMP_SECS does not resolve within an EPC)
            THEN #PF(DS:TMP_SECS) FI;
        IF ( Another instruction is currently modifying the SECS) THEN
            IF ((EAX==07h) OR (EAX==08h)) (* ELDB/ELDU *)
                THEN #GP(0);
            ELSE (* ELDBC/ELDUC *)
                RFLAGS.ZF := 1;
                RFLAGS.CF := 0;
                RAX := SGX_EPC_PAGE_CONFLICT;
                GOTO ERROR_EXIT;
            FI;
        FI;
        TMP_HEADER.EID := DS:TMP_SECS.EID;
    ELSE
```

```
        (* TMP_HEADER.SECINFO.FLAGS.PT is PT_SECS or PT_VA which do not have a parent SECS, and hence no EID binding *)
        TMP_HEADER.EID := 0;
        IF (DS:TMP_SECS ≠ 0)
            THEN #GP(0) FI;
FI;

(* Copy 4KBytes SRCPGE to secure location *)
DS:RCX[32767: 0] := DS:TMP_SRCPGE[32767: 0];
TMP_VER := DS:RDX[63:0];

(* Decrypt and MAC page. AES_GCM_DEC has 2 outputs, {plain text, MAC} *)
(* Parameters for AES_GCM_DEC {Key, Counter, ..} *)
{DS:RCX, TMP_MAC} := AES_GCM_DEC(CR_BASE_PK, TMP_VER << 32, TMP_HEADER, 128, DS:RCX, 4096);

IF ( (TMP_MAC ≠ DS:TMP_PCMD.MAC) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_MAC_COMPARE_FAIL;
        GOTO ERROR_EXIT;
FI;

(* Clear VA Slot *)
DS:RDX := 0

(* Commit EPCM changes *)
EPCM(DS:RCX).PT := TMP_HEADER.SECINFO.FLAGS.PT;
EPCM(DS:RCX).RWX := TMP_HEADER.SECINFO.FLAGS.RWX;
EPCM(DS:RCX).PENDING := TMP_HEADER.SECINFO.FLAGS.PENDING;
EPCM(DS:RCX).MODIFIED := TMP_HEADER.SECINFO.FLAGS.MODIFIED;
EPCM(DS:RCX).PR := TMP_HEADER.SECINFO.FLAGS.PR;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_HEADER.LINADDR;

IF ( ((EAX = 07H) or (EAX = 12H)) and (TMP_HEADER.SECINFO.FLAGS.PT is NOT PT_SECS or PT_VA))
    THEN
        EPCM(DS:RCX).BLOCKED := 1;
    ELSE
        EPCM(DS:RCX).BLOCKED := 0;
FI;

IF (TMP_HEADER.SECINFO.FLAGS.PT is PT_SECS)
    << store translation of DS:RCX produced by paging in SECS(DS:RCX).ENCLAVECONTEXT >>
FI;

EPCM(DS:RCX). VALID := 1;

RAX := 0;
RFLAGS.ZF := 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the instruction's EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand expected to be in EPC does not resolve to an EPC page.
	If one of the EPC memory operands has incorrect page type.
	If the destination EPC page is already valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the instruction's EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand expected to be in EPC does not resolve to an EPC page.
	If one of the EPC memory operands has incorrect page type.
	If the destination EPC page is already valid.

EMODPR—Restrict the Permissions of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0EH ENCLS[EMODPR]	IR	V/V	SGX2	This leaf function restricts the access rights associated with a EPC page in an initialized enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODPR (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function restricts the access rights associated with an EPC page in an initialized enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not restrict the page permissions will have no effect. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPR leaf function.

EMODPR Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODPR Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 38-31. EMODPR Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EMODPR successful.
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB.

Concurrency Restrictions

Table 38-32. Base Concurrency Restrictions of EMODPR

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODPR	Target [DS:RCX]	Shared	#GP	

Table 38-33. Additional Concurrency Restrictions of EMODPR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODPR	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	

Operation

Temp Variables in EMODPR Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or
    (SCRATCH_SECINFO.FLAGS.R is 0 and SCRATCH_SECINFO.FLAGS.W is not 0) )
    THEN #GP(0); FI;

(* Check concurrency with SGX1 or SGX2 instructions on the EPC page *)
IF (SGX1 or other SGX2 instructions accessing EPC page)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID is 0 )
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PAGE_NOT_MODIFIABLE;
```

```
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT is not PT_REG)
    THEN #PF(DS:RCX); FI;

TMP_SECS := GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
  THEN #GP(0); FI;

(* Set the PR bit to indicate that permission restriction is in progress *)
EPCM(DS:RCX).PR := 1;

(* Update EPCM permissions *)
EPCM(DS:RCX).R := EPCM(DS:RCX).R & SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := EPCM(DS:RCX).W & SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := EPCM(DS:RCX).X & SCRATCH_SECINFO.FLAGS.X;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

EMODT—Change the Type of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0FH ENCLS[EMODT]	IR	V/V	SGX2	This leaf function changes the type of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function modifies the type of an EPC page. The security attributes are configured to prevent access to the EPC page at its new type until a corresponding invocation of the EACCEPT leaf confirms the modification. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODT leaf function.

EMODT Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODT Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 38-34. EMODT Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EMODT successful.
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODPR, or EWB.

Concurrency Restrictions

Table 38-35. Base Concurrency Restrictions of EMODT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODT	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR

Table 38-36. Additional Concurrency Restrictions of EMODT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODT	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	

Operation

Temp Variables in EMODT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or
    !(SCRATCH_SECINFO.FLAGS.PT is PT_TCS or SCRATCH_SECINFO.FLAGS.PT is PT_TRIM) )
    THEN #GP(0); FI;

(* Check concurrency with SGX1 instructions on the EPC page *)
IF (other SGX1 instructions accessing EPC page)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).VALID is 0)
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
```

```
FI;

IF (!(EPCM(DS:RCX).PT is PT_REG or
    ((EPCM(DS:RCX).PT is PT_TCS or PT_SS_FIRST or PT_SS_REST) and SCRATCH_SECINFO.FLAGS.PT is PT_TRIM)))
        THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PAGE_NOT_MODIFIABLE;
        GOTO DONE;
FI;

TMP_SECS := GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
    THEN #GP(0); FI;

(* Update EPCM fields *)
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).MODIFIED := 1;
EPCM(DS:RCX).R := 0;
EPCM(DS:RCX).W := 0;
EPCM(DS:RCX).X := 0;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

EPA—Add Version Array

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0AH ENCLS[EPA]	IR	V/V	SGX1	This leaf function adds a Version Array to the EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EPA (In)	PT_VA (In, Constant)	Effective address of the EPC page (In)

Description

This leaf function creates an empty version array in the EPC page whose logical address is given by DS:RCX, and sets up EPCM attributes for that page. At the time of execution of this instruction, the register RBX must be set to PT_VA.

The table below provides additional information on the memory parameter of EPA leaf function.

EPA Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

Concurrency Restrictions

Table 38-37. Base Concurrency Restrictions of EPA

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EPA	VA [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 38-38. Additional Concurrency Restrictions of EPA

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EPA	VA [DS:RCX]	Concurrent	L	Concurrent		Concurrent	

Operation

```
IF (RBX ≠ PT_VA or DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions accessing the page)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
```

```
                THEN
                    VMCS.Exit_reason := SGX_CONFLICT;
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                    VMCS.Exit_qualification.error := 0;
                    VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                    VMCS.Guest-linear_address := DS:RCX;
                Deliver VMEXIT;
                ELSE
                    #GP(0);
            FI;
FI;

(* Check EPC page must be empty *)
IF (EPCM(DS:RCX). VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Clears EPC page *)
DS:RCX[32767:0] := 0;

EPCM(DS:RCX).PT := PT_VA;
EPCM(DS:RCX).ENCLAVEADDRESS := 0;
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).RWX := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the EPC page.
	If RBX is not set to PT_VA.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the EPC page.
	If RBX is not set to PT_VA.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If the EPC page is valid.

ERDINFO—Read Type and Status Information About an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 10H ENCLS[ERDINFO]	IR	V/V	EAX[6]	This leaf function returns type and status information about an EPC page.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	ERDINFO (In)	Return error code (Out)	Address of a RDINFO structure (In)	Address of the destination EPC page (In)

Description

This instruction reads type and status information about an EPC page and returns it in a RDINFO structure. The STATUS field of the structure describes the status of the page and determines the validity of the remaining fields. The FLAGS field returns the EPCM permissions of the page; the page type; and the BLOCKED, PENDING, MODIFIED, and PR status of the page. For enclave pages, the ENCLAVECONTEXT field of the structure returns the value of SECS.ENCLAVECONTEXT. For non-enclave pages (e.g., VA) ENCLAVECONTEXT returns 0.

For invalid or non-EPC pages, the instruction returns an information code indicating the page's status, in addition to populating the STATUS field.

ERDINFO returns an error code if the destination EPC page is being modified by a concurrent SGX instruction.

RBX contains the effective address of a RDINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of ERDINFO leaf function.

ERDINFO Memory Parameter Semantics

RDINFO	EPCPAGE
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

ERDINFO Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A memory operand is not properly aligned.
DS segment is unusable (32b mode).	A page fault occurs in accessing memory operands.
A memory address is in a non-canonical form (64b mode).	

The error codes are:

Table 38-39. ERDINFO Return Value in RAX

Error Code	Value	Description
No Error	0	ERDINFO successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.
SGX_PG_INVLD		Target page is not a valid EPC page.
SGX_PG_NONEPC		Page is not an EPC page.

Concurrency Restrictions

Table 38-40. Base Concurrency Restrictions of ERDINFO

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ERDINFO	Target [DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	

Table 38-41. Additional Concurrency Restrictions of ERDINFO

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ERDINFO	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ERDINFO Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_RDINFO	Linear Address	64	Address of the RDINFO structure.

```
(* check alignment of RDINFO structure (RBX) *)
IF (DS:RBX is not 32Byte Aligned) THEN
   #GP(0); FI;

(* check alignment of the EPCPAGE (RCX) *)
IF (DS:RCX is not 4KByte Aligned) THEN
   #GP(0); FI;

(* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
IF (DS:RCX does not resolve within EPC) THEN
   RFLAGS.CF := 1;
   RFLAGS.ZF := 0;
   RAX := SGX_PG_NONEPC;
   goto DONE;
FI;

(* Check the EPC page for concurrency *)
IF (EPC page is being modified) THEN
   RFLAGS.ZF = 1;
   RFLAGS.CF = 0;
   RAX = SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check page validity *)
IF (EPCM(DS:RCX).VALID = 0) THEN
   RFLAGS.CF = 1;
```

```
    RFLAGS.ZF = 0;
    RAX = SGX_PG_INVLD;
    goto DONE;
FI;

(* clear the fields of the RDINFO structure *)
TMP_RDINFO := DS:RBX;
TMP_RDINFO.STATUS := 0;
TMP_RDINFO.FLAGS := 0;
TMP_RDINFO.ENCLAVECONTEXT := 0;

(* store page info in RDINFO structure *)
TMP_RDINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
TMP_RDINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
TMP_RDINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
TMP_RDINFO.FLAGS.PR := EPCM(DS:RCX).PR;
TMP_RDINFO.FLAGS.PAGE_TYPE := EPCM(DS:RCX).PAGE_TYPE;
TMP_RDINFO.FLAGS.BLOCKED := EPCM(DS:RCX).BLOCKED;

(* read SECS.ENCLAVECONTEXT for enclave child pages *)
IF ((EPCM(DS:RCX).PAGE_TYPE = PT_REG) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_TCS) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_TRIM) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_SS_FIRST) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_SS_REST)
   ) THEN
    TMP_SECS := Address of SECS for (DS:RCX);
    TMP_RDINFO.ENCLAVECONTEXT := SECS(TMP_SECS).ENCLAVECONTEXT;
FI;

(* populate enclave information for SECS pages *)
IF (EPCM(DS:RCX).PAGE_TYPE = PT_SECS) THEN
    IF ((VMX non-root mode) and
        (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)
       ) THEN
        TMP_RDINFO.STATUS.CHILDPRESENT :=
                        ((SECS(DS:RCX).CHLDCNT ≠ 0) or
                          SECS(DS:RCX).VIRTCHILDCNT ≠ 0);
    ELSE
        TMP_RDINFO.STATUS.CHILDPRESENT := (SECS(DS:RCX).CHLDCNT ≠ 0);
        TMP_RDINFO.STATUS.VIRTCHILDPRESENT :=
                        (SECS(DS:RCX).VIRTCHILDCNT ≠ 0);
        TMP_RDINFO.ENCLAVECONTEXT := SECS(DS_RCX).ENCLAVECONTEXT;
    FI;
FI;

RAX := 0;
RFLAGS.ZF := 0;
RFLAGS.CF := 0;

DONE:
(* clear flags *)
RFLAGS.PF := 0;
RFLAGS.AF := 0;
```

```
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF is set if ERDINFO fails due to concurrent operation with another SGX instruction; otherwise cleared.

CF is set if page is not a valid EPC page or not an EPC page; otherwise cleared.

PF, AF, OF, and SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

EREMOVE—Remove a page from the EPC

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLS[EREMOVE]	IR	V/V	SGX1	This leaf function removes a page from the EPC.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	EREMOVE (In)	Return error code (Out)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be un-associated with its SECS and be marked as unused. This instruction leaf can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

The instruction fails if the operand is not properly aligned or does not refer to an EPC page or the page is in use by another thread, or other threads are running in the enclave to which the page belongs. In addition the instruction fails if the operand refers to an SECS with associations.

EREMOVE Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

The instruction faults if any of the following:

EREMOVE Faulting Conditions

The memory operand is not properly aligned.	The memory operand does not resolve in an EPC page.
Refers to an invalid SECS.	Refers to an EPC page that is locked by another thread.
Another Intel SGX instruction is accessing the EPC page.	RCX does not contain an effective address of an EPC page.
the EPC page refers to an SECS with associations.	

The error codes are:

Table 38-42. EREMOVE Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EREMOVE successful.
SGX_CHILD_PRESENT	If the SECS still have enclave pages loaded into EPC.
SGX_ENCLAVE_ACT	If there are still logical processors executing inside the enclave.

Concurrency Restrictions

Table 38-43. Base Concurrency Restrictions of EREMOVE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EREMOVE	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 38-44. Additional Concurrency Restrictions of EREMOVE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREMOVE	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREMOVE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve to an EPC page)
    THEN #PF(DS:RCX); FI;

TMP_SECS := Get_SECS_ADDRESS();

(* Check the EPC page for concurrency *)
IF (EPC page being referenced by another Intel SGX instruction)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                VMCS.Guest-linear_address := DS:RCX;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

(* if DS:RCX is already unused, nothing to do*)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PT = PT_TRIM AND EPCM(DS:RCX).MODIFIED = 0))
    THEN GOTO DONE;
FI;
```

```
IF ( (EPCM(DS:RCX).PT = PT_VA) OR
    ((EPCM(DS:RCX).PT = PT_TRIM) AND (EPCM(DS:RCX).MODIFIED = 0)) )
    THEN
        EPCM(DS:RCX).VALID := 0;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT = PT_SECS)
    THEN
        IF (DS:RCX has an EPC page associated with it)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_CHILD_PRESENT;
                GOTO ERROR_EXIT;
        FI;
        (* treat SECS as having a child page when VIRTCHILDCNT is non-zero *)
        IF (<<in VMX non-root operation>> AND
       <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>> AND
       (SECS(DS:RCX).VIRTCHILDCNT ≠ 0))
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_CHILD_PRESENT
                GOTO ERROR_EXIT
        FI;
        EPCM(DS:RCX).VALID := 0;
        GOTO DONE;
FI;

IF (Other threads active using SECS)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_ENCLAVE_ACT;
        GOTO ERROR_EXIT;
FI;

IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM) or
 (EPCM(DS:RCX).PT is PT_SS_FIRST) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        EPCM(DS:RCX).VALID := 0;
        GOTO DONE;
FI;

DONE:
RAX := 0;
RFLAGS.ZF := 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the page.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If the memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the page.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the memory operand is not an EPC page.

ETRACK—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0CH ENCLS[ETRACK]	IR	V/V	SGX1	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	ETRACK (In)	Return error code (Out)	Pointer to the SECS of the EPC page (In)

Description

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

The table below provides additional information on the memory parameter of ETRACK leaf function.

ETRACK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 38-45. ETRACK Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	ETRACK successful.
SGX_PREV_TRK_INCMPL	All processors did not complete the previous shoot-down sequence.

Concurrency Restrictions

Table 38-46. Base Concurrency Restrictions of ETRACK

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ETRACK	SECS [DS:RCX]	Shared	#GP	

Table 38-47. Additional Concurrency Restrictions of ETRACK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ETRACK	SECS [DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT

Operation

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions using tracking facility on this SECS)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := TRACKING_RESOURCE_CONFLICT;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := SECS(TMP_SECS).ENCLAVECONTEXT;
                VMCS.Guest-linear_address := 0;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).PT ≠ PT_SECS)
    THEN #PF(DS:RCX); FI;

(* All processors must have completed the previous tracking cycle*)
IF ( (DS:RCX).TRACKING ≠ 0) )
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := TRACKING_REFERENCE_CONFLICT;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := SECS(TMP_SECS).ENCLAVECONTEXT;
                VMCS.Guest-linear_address := 0;
            Deliver VMEXIT;
        FI;
    RFLAGS.ZF := 1;
        RAX := SGX_PREV_TRK_INCMPL;
        GOTO DONE;
    ELSE
        RAX := 0;
        RFLAGS.ZF := 0;
FI;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If another thread is concurrently using the tracking facility on this SECS.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

ETRACKC—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 11H ENCLS[ETRACKC]	IR	V/V	EAX[6]	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX	
IR	ETRACK (In)	Return error code (Out)	Address of the destination EPC page (In, EA)	Address of the SECS page (In, EA)

Description

The ETRACKC instruction is thread safe variant of ETRACK leaf and can be executed concurrently with other CPU threads operating on the same SECS.

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

The table below provides additional information on the memory parameter of ETRACK leaf function.

ETRACKC Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 38-48. ETRACKC Return Value in RAX

Error Code	Value	Description
No Error	0	ETRACKC successful.
SGX_EPC_PAGE_CONFLICT	7	Failure due to concurrent operation of another SGX instruction.
SGX_PG_INVLD	6	Target page is not a VALID EPC page.
SGX_PREV_TRK_INCMPL	17	All processors did not complete the previous tracking sequence.
SGX_TRACK_NOT_REQUIRED	27	Target page type does not require tracking.

Concurrency Restrictions

Table 38-49. Base Concurrency Restrictions of ETRACKC

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ETRACKC	Target [DS:RCX]	Shared	SGX_EPC_PAGE_ CONFLICT	
	SECS implicit	Concurrent		

Table 38-50. Additional Concurrency Restrictions of ETRACKC

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ETRACKC	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS implicit	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT

Operation

Temp Variables in ETRACKC Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.

```
(* check alignment of EPCPAGE (RCX) *)
IF (DS:RCX is not 4KByte Aligned) THEN
#GP(0); FI;

(* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
#PF(DS:RCX, PFEC.SGX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page is being modified) THEN
   RFLAGS.ZF := 1;
   RFLAGS.CF := 0;
   RAX := SGX_EPC_PAGE_CONFLICT;
   goto DONE_POST_LOCK_RELEASE;
FI;

(* check to make sure the page is valid *)
IF (EPCM(DS:RCX).VALID = 0) THEN
   RFLAGS.ZF := 1;
   RFLAGS.CF := 0;
   RAX := SGX_PG_INVLD;
   GOTO DONE;
FI;

(* find out the target SECS page *)
IF (EPCM(DS:RCX).PT is PT_REG or PT_TCS or PT_TRIM or PT_SS_FIRST or PT_SS_REST) THEN
   TMP_SECS := Obtain SECS through EPCM(DS:RCX).ENCLAVESECS;
ELSE IF (EPCM(DS:RCX).PT is PT_SECS) THEN
   TMP_SECS := Obtain SECS through (DS:RCX);
ELSE
   RFLAGS.ZF := 0;
   RFLAGS.CF := 1;
   RAX := SGX_TRACK_NOT_REQUIRED;
   GOTO DONE;
FI;
```

```
(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions using tracking facility on this SECS) THEN
    IF ((VMX non-root mode) and
    (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)) THEN
        VMCS.Exit_reason := SGX_CONFLICT;
        VMCS.Exit_qualification.code := TRACKING_RESOURCE_CONFLICT;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address :=
            SECS(TMP_SECS).ENCLAVECONTEXT;
        VMCS.Guest-linear_address := 0;
        Deliver VMEXIT;
    FI;

  RFLAGS.ZF := 1;
  RFLAGS.CF := 0;
  RAX := SGX_EPC_PAGE_CONFLICT;
  GOTO DONE;
FI;
(* All processors must have completed the previous tracking cycle*)
IF ( (TMP_SECS).TRACKING ≠ 0) )
THEN
    IF ((VMX non-root mode) and
    (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)) THEN
        VMCS.Exit_reason := SGX_CONFLICT;
        VMCS.Exit_qualification.code := TRACKING_REFERENCE_CONFLICT;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address :=
            SECS(TMP_SECS).ENCLAVECONTEXT;
        VMCS.Guest-linear_address := 0;
        Deliver VMEXIT;
    FI;

  RFLAGS.ZF := 1;
  RFLAGS.CF := 0;
  RAX := SGX_PREV_TRK_INCMPL;
  GOTO DONE;
FI;

RFLAGS.ZF := 0;
RFLAGS.CF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.PF,AF,OF,SF := 0;
```

Flags Affected

ZF is set if ETRACKC fails due to concurrent operations with another SGX instructions or target page is an invalid EPC page or tracking is not completed on SECS page; otherwise cleared.

CF is set if target page is not of a type that requires tracking; otherwise cleared.

PF, AF, OF, and SF are cleared.

Protected Mode Exceptions

#GP(0)	If the memory operand violates access-control policies of DS segment. If DS segment is unusable. If the memory operand is not properly aligned.
#PF(error code)	If the memory operand expected to be in EPC does not resolve to an EPC page. If a page fault occurs in access memory operand.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form. If a memory operand is not properly aligned.
#PF(error code)	If the memory operand expected to be in EPC does not resolve to an EPC page. If a page fault occurs in access memory operand.

EWB—Invalidate an EPC Page and Write out to Main Memory

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0BH ENCLS[EWB]	IR	V/V	SGX1	This leaf function invalidates an EPC page and writes it out to main memory.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EWB (In)	Error code (Out)	Address of an PAGEINFO (In)	Address of the EPC page (In)	Address of a VA slot (In)

Description

This leaf function copies a page from the EPC to regular main memory. As part of the copying process, the page is cryptographically protected. This instruction can only be executed when current privilege level is 0.

The table below provides additional information on the memory parameter of EPA leaf function.

EWB Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	EPCPAGE	VASLOT
Non-EPC R/W access	Non-EPC R/W access	Non-EPC R/W access	EPC R/W access	EPC R/W access

The error codes are:

Table 38-51. EWB Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EWB successful.
SGX_PAGE_NOT_BLOCKED	If page is not marked as blocked.
SGX_NOT_TRACKED	If EWB is racing with ETRACK instruction.
SGX_VA_SLOT_OCCUPIED	Version array slot contained valid entry.
SGX_CHILD_PRESENT	Child page present while attempting to page out enclave.

Concurrency Restrictions

Table 38-52. Base Concurrency Restrictions of EWB

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EWB	Source [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA [DS:RDX]	Shared	#GP	

Table 38-53. Additional Concurrency Restrictions of EWB

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EWB	Source [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Exclusive	

Operation

Temp Variables in EWB Operational Flow

Name	Type	Size (Bytes)	Description
TMP_SRCPGE	Memory page	4096	
TMP_PCMD	PCMD	128	
TMP_SECS	SECS	4096	
TMP_BPEPOCH	UINT64	8	
TMP_BPREFCOUNT	UINT64	8	
TMP_HEADER	MAC Header	128	
TMP_PCMD_ENCLAVEID	UINT64	8	
TMP_VER	UINT64	8	
TMP_PK	UINT128	16	

```
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (DS:RDX is not 8Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

(* EPCPAGE and VASLOT should not resolve to the same EPC page*)
IF (DS:RCX and DS:RDX resolve to the same EPC page)
    THEN #GP(0); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
(* Note PAGEINFO.PCMD is overlaid on top of PAGEINFO.SECINFO *)
TMP_PCMD := DS:RBX.PCMD;

If (DS:RBX.LINADDR ≠ 0) OR (DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

IF ( (DS:TMP_PCMD is not 128Byte Aligned) or (DS:TMP_SRCPGE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Check for concurrent Intel SGX instruction access to the page *)
IF (Other Intel SGX instruction is accessing page)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
```

```
                VMCS.Guest-linear_address := DS:RCX;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

(*Check if the VA Page is being removed or changed*)
IF (VA Page is being modified)
    THEN #GP(0); FI;

(* Verify that EPCPAGE and VASLOT page are valid EPC pages and DS:RDX is VA *)
IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~FFFH).PT is not PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Perform page-type-specific exception checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM ) or
 (EPCM(DS:RCX).PT is PT_SS_FIRST ) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        TMP_SECS = Obtain SECS through EPCM(DS:RCX)
    (* Check that EBLOCK has occurred correctly *)
    IF (EBLOCK is not correct)
        THEN #GP(0); FI;
FI;

RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

(* Zero out TMP_HEADER*)
TMP_HEADER[ sizeof(TMP_HEADER) - 1 : 0] := 0;

(* Perform page-type-specific checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM )or
 (EPCM(DS:RCX).PT is PT_SS_FIRST ) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        (* check to see if the page is evictable *)
        IF (EPCM(DS:RCX).BLOCKED = 0)
            THEN
                RAX := SGX_PAGE NOT_BLOCKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
        FI;
        (* Check if tracking done correctly *)
        IF (Tracking not correct)
            THEN
                RAX := SGX_NOT_TRACKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
        FI;

        (* Obtain EID to establish cryptographic binding between the paged-out page and the enclave *)
```

```
            TMP_HEADER.EID := TMP_SECS.EID;

            (* Obtain EID as an enclave handle for software *)
            TMP_PCMD_ENCLAVEID := TMP_SECS.EID;
        ELSE IF (EPCM(DS:RCX).PT is PT_SECS)
            (*check that there are no child pages inside the enclave *)
            IF (DS:RCX has an EPC page associated with it)
                THEN
                    RAX := SGX_CHILD_PRESENT;
                    RFLAGS.ZF := 1;
                    GOTO ERROR_EXIT;
            FI:
            (* treat SECS as having a child page when VIRTCHILDCNT is non-zero *)
            IF (<<in VMX non-root operation>> AND
          <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>> AND
          (SECS(DS:RCX).VIRTCHILDCNT ≠ 0))
                THEN
                    RFLAGS.ZF := 1;
                    RAX := SGX_CHILD_PRESENT;
                    GOTO ERROR_EXIT;
            FI;
            TMP_HEADER.EID := 0;
            (* Obtain EID as an enclave handle for software *)
            TMP_PCMD_ENCLAVEID := (DS:RCX).EID;
        ELSE IF (EPCM(DS:RCX).PT is PT_VA)
            TMP_HEADER.EID := 0; // Zero is not a special value
            (* No enclave handle for VA pages*)
            TMP_PCMD_ENCLAVEID := 0;
FI;

TMP_HEADER.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;
TMP_HEADER.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
TMP_HEADER.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
TMP_HEADER.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
TMP_HEADER.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
TMP_HEADER.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;

(* Encrypt the page, DS:RCX could be encrypted in place. AES-GCM produces 2 values, {ciphertext, MAC}. *)
(* AES-GCM input parameters: key, GCM Counter, MAC_HDR, MAC_HDR_SIZE, SRC, SRC_SIZE)*)
{DS:TMP_SRCPGE, DS:TMP_PCMD.MAC} := AES_GCM_ENC(CR_BASE_PK), (TMP_VER << 32),
    TMP_HEADER, 128, DS:RCX, 4096);

(* Write the output *)
Zero out DS:TMP_PCMD.SECINFO
DS:TMP_PCMD.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
DS:TMP_PCMD.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
DS:TMP_PCMD.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
DS:TMP_PCMD.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
DS:TMP_PCMD.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;
DS:TMP_PCMD.RESERVED := 0;
DS:TMP_PCMD.ENCLAVEID := TMP_PCMD_ENCLAVEID;
DS:RBX.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;

(*Check if version array slot was empty *)
```

```
IF ([DS.RDX])
    THEN
        RAX := SGX_VA_SLOT_OCCUPIED
        RFLAGS.CF := 1;
FI;

(* Write version to Version Array slot *)
[DS.RDX] := TMP_VER;

(* Free up EPCM Entry *)
EPCM.(DS:RCX).VALID := 0;
ERROR_EXIT:
```

Flags Affected

ZF is set if page is not blocked, not tracked, or a child is present. Otherwise cleared.

CF is set if VA slot is previously occupied, Otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If the tracking resource is in use.
	If the EPC page or the version array page is invalid.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If one of the EPC memory operands has incorrect page type.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If the tracking resource is in use.
	If the EPC page or the version array page in invalid.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If one of the EPC memory operands has incorrect page type.

38.4 INTEL® SGX USER LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLU instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of the implicitly-encoded register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EACCEPT—Accept Changes to an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLU[EACCEPT]	IR	V/V	SGX2	This leaf function accepts changes made by system software to an EPC page in the running enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EACCEPT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function accepts changes to a page in the running enclave by verifying that the security attributes specified in the SECINFO match the security attributes of the page in the EPCM. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPT leaf function.

EACCEPT Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPT Faulting Conditions

The operands are not properly aligned.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	Page type is PT_REG and MODIFIED bit is 0.
SECINFO contains an invalid request.	Page type is PT_TCS or PT_TRIM and PENDING bit is 0 and MODIFIED bit is 1.
If security attributes of the SECINFO page make the page inaccessible.	

The error codes are:

Table 38-54. EACCEPT Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EACCEPT successful.
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values.
SGX_NOT_TRACKED	The OS did not complete an ETRACK on the target page.

Concurrency Restrictions

Table 38-55. Base Concurrency Restrictions of EACCEPT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPT	Target [DS:RCX]	Shared	#GP	
	SECINFO [DS:RBX]	Concurrent		

Table 38-56. Additional Concurrency Restrictions of EACCEPT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPT	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EACCEPT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operands belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX is not within CR_ELRANGE)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
    (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or
    (EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ (DS:RBX & FFFH)) )
    THEN #PF(DS:RBX); FI;

(* Copy 64 bytes of contents *)
SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero )
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;
```

```
IF (DS:RCX is not within CR_ELRANGE)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check that the combination of requested PT, PENDING, and MODIFIED is legal *)
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 0 )
    THEN
        IF (NOT (((SCRATCH_SECINFO.FLAGS.PT is PT_REG) and
         ((SCRATCH_SECINFO.FLAGS.PR is 1) or
         (SCRATCH_SECINFO.FLAGS.PENDING is 1)) and
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) or
         ((SCRATCH_SECINFO.FLAGS.PT is PT_TCS or PT_TRIM) and
         (SCRATCH_SECINFO.FLAGS.PR is 0) and
         (SCRATCH_SECINFO.FLAGS.PENDING is 0) and
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 1) )))
            THEN #GP(0); FI
    ELSE
        IF (NOT (((SCRATCH_SECINFO.FLAGS.PT is PT_REG) AND
         ((SCRATCH_SECINFO.FLAGS.PR is 1) OR
         (SCRATCH_SECINFO.FLAGS.PENDING is 1)) AND
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) OR
         ((SCRATCH_SECINFO.FLAGS.PT is PT_TCS OR PT_TRIM) AND
         (SCRATCH_SECINFO.FLAGS.PENDING is 0) AND
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 1) AND
         (SCRATCH_SECINFO.FLAGS.PR is 0)) OR
         ((SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST or PT_SS_REST) AND
         (SCRATCH_SECINFO.FLAGS.PENDING is 1) AND
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 0) AND
         (SCRATCH_SECINFO.FLAGS.PR is 0))))
            THEN #GP(0); FI;
    FI;

(* Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).BLOCKED is not 0) or
  ((EPCM(DS:RCX).PT is not PT_REG) and (EPCM(DS:RCX).PT is not PT_TCS) and (EPCM(DS:RCX).PT is not PT_TRIM)
  and (EPCM(DS:RCX).PT is not PT_SS_FIRST) and (EPCM(DS:RCX).PT is not PT_SS_REST)) or
  (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS))
    THEN #PF((DS:RCX); FI;

(* Check the destination EPC page for concurrency *)
IF ( EPC page in use )
    THEN #GP(0); FI;

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN #PF(DS:RCX); FI;

(* Verify that accept request matches current EPC page settings *)
IF ( (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX) or (EPCM(DS:RCX).PENDING ≠ SCRATCH_SECINFO.FLAGS.PENDING) or
    (EPCM(DS:RCX).MODIFIED ≠ SCRATCH_SECINFO.FLAGS.MODIFIED) or (EPCM(DS:RCX).R ≠ SCRATCH_SECINFO.FLAGS.R) or
    (EPCM(DS:RCX).W ≠ SCRATCH_SECINFO.FLAGS.W) or (EPCM(DS:RCX).X ≠ SCRATCH_SECINFO.FLAGS.X) or
    (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) )
```

```
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
        GOTO DONE;
FI;
(* Check that all required threads have left enclave *)
IF (Tracking not correct)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_NOT_TRACKED;
        GOTO DONE;
FI;

(* Get pointer to the SECS to which the EPC page belongs *)
TMP_SECS = << Obtain physical address of SECS through EPCM(DS:RCX)>>
(* For TCS pages, perform additional checks *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;

    (* Check that TCS.FLAGS.DBGOPTIN, TCS stack, and TCS status are correctly initialized *)
    (* check that TCS.PREVSSP is 0 *)
    IF ( ((DS:RCX).FLAGS.DBGOPTIN is not 0) or ((DS:RCX).CSSA ≥ (DS:RCX).NSSA) or ((DS:RCX).AEP is not 0) or ((DS:RCX).STATE is not 0)
or ((CPUID.(EAX=07H, ECX=0H):ECX[CET_SS] = 1) AND ((DS:RCX).PREVSSP != 0)))
        THEN #GP(0); FI;

    (* Check consistency of FS & GS Limit *)
    IF ( (TMP_SECS.ATTRIBUTES.MODE64BIT is 0) and ((DS:RCX.FSLIMIT & FFFH ≠ FFFH) or (DS:RCX.GSLIMIT & FFFH ≠ FFFH)) )
        THEN #GP(0); FI;
FI;

(* Clear PENDING/MODIFIED flags to mark accept operation complete *)
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;

(* Clear EAX and ZF to indicate successful completion *)
RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page cannot be accepted, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

EACCEPTCOPY—Initialize a Pending Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLU[EACCEPTCOPY]	IR	V/V	SGX2	This leaf function initializes a dynamically allocated EPC page from another page in the EPC.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EACCEPTCOPY (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)	Address of the source EPC page (In)

Description

This leaf function copies the contents of an existing EPC page into an uninitialized EPC page (created by EAUG). After initialization, the instruction may also modify the access rights associated with the destination EPC page. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX and RDX each contain the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPTCOPY leaf function.

EACCEPTCOPY Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)	EPCPAGE (Source)
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPTCOPY Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	If security attributes of the source EPC page make the page inaccessible.
The EPC page is not valid.	RBX does not contain an effective address in an EPC page in the running enclave.
SECINFO contains an invalid request.	RCX/RDX does not contain an effective address of an EPC page in the running enclave.

The error codes are:

Table 38-57. EACCEPTCOPY Return Value in RAX

Error Code (see Table 38-4)	Description
No Error	EACCEPTCOPY successful.
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values.

Concurrency Restrictions

Table 38-58. Base Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPTCOPY	Target [DS:RCX]	Concurrent		
	Source [DS:RDX]	Concurrent		
	SECINFO [DS:RBX]	Concurrent		

Table 38-59. Additional Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPTCOPY	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	Source [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EACCEPTCOPY Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF ( (DS:RCX is not 4KByte Aligned) or (DS:RDX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) or (DS:RDX is not within CR_ELRANGE))
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

IF ( (EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
    (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or (EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or
    (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ DS:RBX) )
    THEN #PF(DS:RBX); FI;
```

```
(* Copy 64 bytes of contents *)
SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or (SCRATCH_SECINFO.FLAGS.R=0) AND(SCRATCH_SECINFO.FLAGS.W≠0 ) or
     (SCRATCH_SECINFO.FLAGS.PT is not PT_REG) )
     THEN #GP(0); FI;

(* Check security attributes of the source EPC page *)
IF ( (EPCM(DS:RDX).VALID = 0) or (EPCM(DS:RCX).R = 0) or (EPCM(DS:RDX).PENDING ≠ 0) or (EPCM(DS:RDX).MODIFIED ≠ 0) or
     (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
     (EPCM(DS:RDX).ENCLAVEADDRESS ≠ DS:RDX))
     THEN #PF(DS:RDX); FI;

(* Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
     (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
          GOTO DONE;
FI;

(* Check the destination EPC page for concurrency *)
IF (destination EPC page in use )
     THEN #GP(0); FI;

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
     (EPCM(DS:RCX).R ≠ 1) or (EPCM(DS:RCX).W ≠ 1) or (EPCM(DS:RCX).X ≠ 0) or
     (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
     (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
          GOTO DONE;
FI;

(* Copy 4KBbytes form the source to destination EPC page*)
DS:RCX[32767:0] := DS:RDX[32767:0];

(* Update EPCM permissions *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PENDING := 0;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page is not modifiable, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

EDECCSSA—Decrements TCS.CSSA

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLU[EDECCSSA]	IR	V/V	EDECCSSA	This leaf function decrements TCS.CSSA.

Instruction Operand Encoding

Op/En	EAX
IR	EDECCSSA (In)

Description

This leaf function changes the current SSA frame by decrementing TCS.CSSA for the current enclave thread. If the enclave has enabled CET shadow stacks or indirect branch tracking, then EDECCSSA also changes the current CET state save frame. This instruction leaf can only be executed inside an enclave.

EDECCSSA Memory Parameter Semantics

TCS
Read/Write access by Enclave

The instruction faults if any of the following:

EDECCSSA Faulting Conditions

TCS.CSSA is 0.	TCS is not valid or available or locked.
The SSA frame is not valid or in use.	

Concurrency Restrictions

Table 38-60. Base Concurrency Restrictions of EDECCSSA

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDECCSSA	TCS [CR_TCS_PA]	Shared	#GP	

Table 38-61. Additional Concurrency Restrictions of EDECCSSA

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDECCSSA	TCS [CR_TCS_PA]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDECCSSA Operational Flow

Name	Type	Size (bits)	Description
TMP_SSA	Effective Address	32/64	Address of current SSA frame.
TMP_XSIZE	Integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the target frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the target SSA frame.
TMP_XSAVE_PAGE_PA_n	Physical Address	32/64	Physical address of the nth page within the target SSA frame.
TMP_CET_SAVE_AREA	Effective Address	32/64	Address of the current CET save area.
TMP_CET_SAVE_PAGE	Effective Address	32/64	Address of the current CET save area page.

```
IF (CR_TCS_PA.CSSA = 0)
    THEN #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA := CR_TCS_PA.OSSA + CR_ACTIVE_SECS.BASEADDR + 4096 * CR_ACTIVE_SECS.SSAFRAMESIZE * (CR_TCS_PA.CSSA - 1);
TMP_XSIZE := compute_XSAVE_frame_size(CR_ACTIVE_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;
    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE_.MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMPSSA_PAGE) or
    (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
    (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(CR_TCS_PA).ENCLAVESECS) or
    (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    TMP_XSAVE_PAGE_PA_n := Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR := TMP_SSA + 4096 * CR_ACTIVE_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
Check that DS:TMP_SSA_PAGE is read/write accessible;
If a fault occurs, release locks, abort and deliver that fault;
IF (DS:TMP_GPR does not resolve to EPC page)
    THEN #PF(DS:TMP_GPR); FI;
```

```
IF (EPCM(DS:TMP_GPR).VALID = 0)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or
    (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS ≠ EPCM(CR_TCS_PA).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
        THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)
    THEN
        IF (TMP_GPR + (sizeof(GPRSGX_AREA) -1) is not in DS segment)
            THEN #GP(0); FI;
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        IF ((CR_ACTIVE_SECS.CET_ATTRIBUTES.SH_STK_EN == 1) OR (CR_ACTIVE_SECS.CET_ATTRIBUTES.ENDBR_EN == 1))
            THEN
                (* Compute linear address of what will become new CET state save area and cache its PA *)
                TMP_CET_SAVE_AREA := CR_TCS_PA.OCETSSA + CR_ACTIVE_SECS.BASEADDR + (CR_TCS_PA.CSSA - 1) * 16;
                TMP_CET_SAVE_PAGE := TMP_CET_SAVE_AREA & ~0xFFF;
                Check the TMP_CET_SAVE_PAGE page is read/write accessible
                If fault occurs release locks, abort and deliver fault

                (* read the EPCM VALID, PENDING, MODIFIED, BLOCKED and PT fields atomically *)
                IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR
                (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(CR_TCS_PA).ENCLAVESECS))
            THEN #PF(DS:TMP_CET_SAVE_PAGE); FI;
        FI;
FI;

(* At this point, the instruction is guaranteed to complete *)
CR_TCS_PA.CSSA := CR_TCS_PA.CSSA - 1;

CR_GPR_PA := Physical_Address(DS:TMP_GPR);

FOR EACH TMP_XSAVE_PAGE_n
    CR_XSAVE_PAGE_n := TMP_XSAVE_PAGE_PA_n;
ENDFOR

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
```

```
        IF ((TMP_SECS.CET_ATTRIBUTES.SH_STK_EN == 1) OR
        (TMP_SECS.CET_ATTRIBUTES.ENDBR_EN == 1))
            THEN
                CR_CET_SAVE_AREA_PA := Physical_Address(DS:TMP_CET_SAVE_AREA);
        FI;
FI;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave. If CR_TCS_PA.CSSA = 0.
#PF(error code)	If a page fault occurs in accessing memory. If one or more pages of the target SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page. If CET is enabled for the enclave and the target CET SSA frame is not readable/writable, or does not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave. If CR_TCS_PA.CSSA = 0.
#PF(error code)	If a page fault occurs in accessing memory. If one or more pages of the target SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page. If CET is enabled for the enclave and the target CET SSA frame is not readable/writable, or does not resolve to a valid PT_REG EPC page.

EENTER—Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLU[EENTER]	IR	V/V	SGX1	This leaf function is used to enter an enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	
IR	EENTER (In)	Content of RBX.CSSA (Out)	Address of a TCS (In)	Address of AEP (In)	Address of IP following EENTER (Out)

Description

The ENCLU[EENTER] instruction transfers execution to an enclave. At the end of the instruction, the logical processor is executing in enclave mode at the RIP computed as EnclaveBase + TCS.OENTRY. If the target address is not within the CS segment (32-bit) or is not canonical (64-bit), a #GP(0) results.

EENTER Memory Parameter Semantics

TCS
Enclave access

EENTER is a serializing instruction. The instruction faults if any of the following occurs:

Address in RBX is not properly aligned.	Any TCS.FLAGS's must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use.	Either of TCS-specified FS and GS segment is not a subsets of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.	

The following operations are performed by EENTER:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or interrupt.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 40.1.2):
  - On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 40.2.5).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 40.2.2).

- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.
- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 40.2.3):
  - — All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
  - — PEBS is suppressed.
  - — AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set
  - — If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 38-62. Base Concurrency Restrictions of EENTER

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EENTER	TCS [DS:RBX]	Shared	#GP	

Table 38-63. Additional Concurrency Restrictions of EENTER

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EENTER	TCS [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EENTER Operational Flow

Name	Type	Size (Bits)	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)
IF (TMP_MODE64 = 0 and (DS not usable or ( ( DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1) ) )
    THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)
IF (TMP_MODE64 = 0)
    THEN
```

```
            IF(CS.base ≠ 0 or DS.base ≠ 0) #GP(0); FI;
            IF(ES usable and ES.base ≠ 0) #GP(0); FI;
            IF(SS usable and SS.base ≠ 0) #GP(0); FI;
            IF(SS usable and SS.B = 0) #GP(0); FI;
FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (CS:RCX is not canonical) )
    THEN #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions are operating on TCS)
    THEN #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) or (EPCM(DS:RBX).PT ≠ PT_TCS) )
    THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS := Address of SECS for TCS;

(* Ensure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & FFFFFFFFFFFFFFFCH) ≠ 0)
    THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    THEN #GP(0); FI;

(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
    THEN #GP(0); FI;
```

```
IF (CR4.OSFXSR = 0)
    THEN #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    THEN
        IF (TMP_SECS.ATTRIBUTES.XFRM ≠ 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUTES.XFRM & XCR0) ≠ TMP_SECS.ATTRIBUES.XFRM) THEN #GP(0); FI;
FI;

IF ((DS:RBX).CSSA.FLAGS.DBGOPTIN = 0) and (DS:RBX).CSSA.FLAGS.AEXNOTIFY ≠ TMP_SECS.ATTRIBUTES.AEXNOTIFY))
    THEN #GP(0); FI;

(* Make sure the SSA contains at least one more frame *)
IF ( (DS:RBX).CSSA ≥ (DS:RBX).NSSA)
    THEN #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA := (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * (DS:RBX).CSSA;
TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort, and deliver that fault;

    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMP_SSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
        (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0) )
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
If a fault occurs; release locks, abort, and deliver that fault;

IF (DS:TMP_GPR does not resolve to EPC page)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
```

```
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS EPCM(DS:RBX).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
    THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)
    THEN
        IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
FI;

CR_GPR_PA := Physical_Address (DS: TMP_GPR);

(* Validate TCS.OENTRY *)
TMP_TARGET := (DS:RBX).OENTRY + TMP_SECS.BASEADDR;
IF (TMP_MODE64 = 1)
    THEN
        IF (TMP_TARGET is not canonical) THEN #GP(0); FI;
    ELSE
        IF (TMP_TARGET > CS limit) THEN #GP(0); FI;
FI;

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    THEN
        TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_FSLIMIT := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
        TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        TMP_GSLIMIT := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
        (* if FS wrap-around, make sure DS has no holes*)
        IF (TMP_FSLIMIT < TMP_FSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
        (* if GS wrap-around, make sure DS has no holes*)
        IF (TMP_GSLIMIT < TMP_GSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
    ELSE
        TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
            THEN #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE)
    THEN #GP(0); FI;

TMP_IA32_U_CET := 0
```

```
TMP_SSP : = 0

IF CPUID.(EAX=12H, ECX=1):EAX[6] = 1
    THEN
        IF ( CR4.CET = 0 )
            THEN
                (* If part does not support CET or CET has not been enabled and enclave requires CET then fail *)
                IF ( TMP_SECS.CET_ATTRIBUTES ≠ 0 OR TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0 ) #GP(0); FI;
        FI;
        (* If indirect branch tracking or shadow stacks enabled but CET state save area is not 16B aligned then fail EENTER *)
        IF ( TMP_SECS.CET_ATTRIBUTES.SH_STK_EN = 1 OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN = 1 )
            THEN
                IF (DS:RBX.OCETSSA is not 16B aligned) #GP(0); FI;
        FI;

    IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN)
        THEN
            (* Setup CET state from SECS, note tracker goes to IDLE *)
            TMP_IA32_U_CET = TMP_SECS.CET_ATTRIBUTES;
            IF (TMP_IA32_U_CET.LEG_IW_EN = 1 AND TMP_IA32_U_CET.ENDBR_EN = 1 )
                THEN
                    TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.BASEADDR;
                    TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.CET_LEG_BITMAP_BASE;
            FI;

            (* Compute linear address of what will become new CET state save area and cache its PA *)
            TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA) * 16
            TMP_CET_SAVE_PAGE = TMP_CET_SAVE_AREA & ~0xFFF;

            Check the TMP_CET_SAVE_PAGE page is read/write accessible
            If fault occurs release locks, abort, and deliver fault

            (* Read the EPCM VALID, PENDING, MODIFIED, BLOCKED, and PT fields atomically *)
            IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS))
                THEN
                    #PF(DS:TMP_CET_SAVE_PAGE);
            FI;

            CR_CET_SAVE_AREA_PA := Physical address(DS:TMP_CET_SAVE_AREA)

            IF TMP_IA32_U_CET.SH_STK_EN = 1
                THEN
                    TMP_SSP = TCS.PREVSSP;
            FI;
    FI;
```

```
FI;

CR_ENCLAVE_MODE := 1;
CR_ACTIVE_SECS := TMP_SECS;
CR_ELRANGE := (TMPSECS.BASEADDR, TMP_SECS.SIZE);

(* Save state for possible AEXs *)
CR_TCS_PA := Physical_Address (DS:RBX);
CR_TCS_LA := RBX;
CR_TCS_LA.AEP := RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector := FS.selector;
CR_SAVE_FS_base := FS.base;
CR_SAVE_FS_limit := FS.limit;
CR_SAVE_FS_access_rights := FS.access_rights;
CR_SAVE_GS_selector := GS.selector;
CR_SAVE_GS_base := GS.base;
CR_SAVE_GS_limit := GS.limit;
CR_SAVE_GS_access_rights := GS.access_rights;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 := XCR0;
    XCR0 := TMP_SECS.ATTRIBUTES.XFRM;
FI;

RCX := RIP;
RIP := TMP_TARGET;
RAX := (DS:RBX).CSSA;
(* Save the outside RSP and RBP so they can be restored on interrupt or EEXIT *)
DS:TMP_SSA.U_RSP := RSP;
DS:TMP_SSA.U_RBP := RBP;

(* Do the FS/GS swap *)
FS.base := TMP_FSBASE;
FS.limit := DS:RBX.FSLIMIT;
FS.type := 0001b;
FS.W := DS.W;
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS.L;
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS.W;
GS.S := 1;
```

```
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS.L;
GS.unusable := 0;
GS.selector := 0BH;

CR_DBGOPTIN := TCS.FLAGS.DBGOPTIN;
Suppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        Suppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        CR_SAVE_TF := RFLAGS.TF;
        RFLAGS.TF := 0;
        Suppress_monitor_trap_flag for the source of the execution of the enclave;
        Suppress any pending debug exceptions;
        Suppress any pending MTF VM exit;
    ELSE
        IF RFLAGS.TF = 1
            THEN pend a single-step #DB at the end of EENTER; FI;
        IF the "monitor trap flag" VM-execution control is set
            THEN pend an MTF VM exit at the end of EENTER; FI;
FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7H, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* Save enclosing application CET state into save registers *)
        CR_SAVE_IA32_U_CET := IA32_U_CET
        (* Setup enclave CET state *)
        IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
            THEN
                CR_SAVE_SSP := SSP
                SSP := TMP_SSP
        FI;

        IA32_U_CET := TMP_IA32_U_CET;

FI;

Flush_linear_context;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry.

Protected Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If any reserved field in the TCS FLAG is set.
	If the target address is not within the CS segment.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	If a page fault occurs in accessing memory.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
	If the target address is not canonical.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

EEXIT—Exits an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLU[EEXIT]	IR	V/V	SGX1	This leaf function is used to exit an enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EEXIT (In)	Target address outside the enclave (In)	Address of the current AEP (Out)

Description

The ENCLU[EEXIT] instruction exits the currently executing enclave and branches to the location specified in RBX. RCX receives the current AEP. If RBX is not within the CS (32-bit mode) or is not canonical (64-bit mode) a #GP(0) results.

EEXIT Memory Parameter Semantics

Target Address
Non-Enclave read and execute access

If RBX specifies an address that is inside the enclave, the instruction will complete normally. The fetch of the next instruction will occur in non-enclave mode, but will attempt to fetch from inside the enclave. This fetch returns a fixed data pattern.

If secrets are contained in any registers, it is responsibility of enclave software to clear those registers.

If XCR0 was modified on enclave entry, it is restored to the value it had at the time of the most recent EENTER or ERESUME.

If the enclave is opt-out, RFLAGS.TF is loaded from the value previously saved on EENTER.

Code and data breakpoints are unsuppressed.

Performance monitoring counters are unsuppressed.

Concurrency Restrictions

Table 38-64. Base Concurrency Restrictions of EEXIT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EEXIT		Concurrent		

Table 38-65. Additional Concurrency Restrictions of EEXIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EEXIT		Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EEXIT Operational Flow

Name	Type	Size (Bits)	Description
TMP_RIP	Effective Address	32/64	Saved copy of CRIP for use when creating LBR.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF (TMP_MODE64 = 1)
    THEN
        IF (RBX is not canonical) THEN #GP(0); FI;
    ELSE
        IF (RBX > CS limit) THEN #GP(0); FI;
FI;

TMP_RIP := CRIP;
RIP := RBX;

(* Return current AEP in RCX *)
RCX := CR_TCS_PA.AEP;

(* Do the FS/GS swap *)
FS.selector := CR_SAVE_FS.selector;
FS.base := CR_SAVE_FS.base;
FS.limit := CR_SAVE_FS.limit;
FS.access_rights := CR_SAVE_FS.access_rights;
GS.selector := CR_SAVE_GS.selector;
GS.base := CR_SAVE_GS.base;
GS.limit := CR_SAVE_GS.limit;
GS.access_rights := CR_SAVE_GS.access_rights;

(* Restore XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
    XCR0 := CR_SAVE__XCR0;
FI;

Unsuppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        UnSuppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        Restore suppressed breakpoint matches;
        RFLAGS.TF := CR_SAVE_TF;
        UnSuppress_montior_trap_flag;
        UnSuppress_LBR_Generation;
        UnSuppress_performance monitoring_activity;
        Restore performance monitoring counter AnyThread demotion to MyThread in enclave back to AnyThread
FI;

IF RFLAGS.TF = 1
    THEN Pend Single-Step #DB at the end of EEXIT;
FI;
```

```
IF the "monitor trap flag" VM-execution control is set
    THEN pend a MTF VM exit at the end of EEXIT;
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        (* Record PREVSSP *)
        IF (IA32_U_CET.SH_STK_EN == 1)
            THEN CR_TCS_PA.PREVSSP = SSP; FI;
FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* Restore enclosing app's CET state from the save registers *)
        IA32_U_CET := CR_SAVE_IA32_U_CET;
        IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
            THEN SSP := CR_SAVE_SSP; FI;

        (* Update enclosing app's TRACKER if enclosing app has indirect branch tracking enabled *)
        IF (CR4.CET = 1 AND IA32_U_CET.ENDBR_EN = 1)
            THEN
                IA32_U_CET.TRACKER := WAIT_FOR_ENDBRANCH;
                IA32_U_CET.SUPPRESS := 0
        FI;
FI;

CR_ENCLAVE_MODE := 0;
CR_TCS_PA.STATE := INACTIVE;

(* Assure consistent translations *)
Flush_linear_context;
```

Flags Affected

RFLAGS.TF is restored from the value previously saved in EENTER or ERESUME.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RBX is outside the CS segment.
#PF(error code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RBX is not canonical.
#PF(error code)	If a page fault occurs in accessing memory operands.

EGETKEY—Retrieves a Cryptographic Key

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLU[EGETKEY]	IR	V/V	SGX1	This leaf function retrieves a cryptographic key.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EGETKEY (In)	Return error code (Out)	Address to a KEYREQUEST (In)	Address of the OUTPUTDATA (In)

Description

The ENCLU[EGETKEY] instruction returns a 128-bit secret key from the processor specific key hierarchy. The register RBX contains the effective address of a KEYREQUEST structure, which the instruction interprets to determine the key being requested. The Requesting Keys section below provides a description of the keys that can be requested. The RCX register contains the effective address where the key will be returned. Both the addresses in RBX & RCX should be locations inside the enclave.

EGETKEY derives keys using a processor unique value to create a specific key based on a number of possible inputs. This instruction leaf can only be executed inside an enclave.

EEGETKEY Memory Parameter Semantics

KEYREQUEST	OUTPUTDATA
Enclave read access	Enclave write access

After validating the operands, the instruction determines which key is to be produced and performs the following actions:

- The instruction assembles the derivation data for the key based on the Table 38-66.
- Computes derived key using the derivation data and package specific value.
- Outputs the calculated key to the address in RCX.

The instruction fails with #GP(0) if the operands are not properly aligned. Successful completion of the instruction will clear RFLAGS.{ZF, CF, AF, OF, SF, PF}. The instruction returns an error code if the user tries to request a key based on an invalid CPUSVN or ISVSVN (when the user request is accepted, see the table below), requests a key for which it has not been granted the attribute to request, or requests a key that is not supported by the hardware. These checks may be performed in any order. Thus, an indication by error number of one cause (for example, invalid attribute) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same Enclave. The correctness of software should not rely on the order resulting from the checks documented in this section. In such cases the ZF flag is set and the corresponding error bit (SGX_INVALID_SVN, SGX_INVALID_ATTRIBUTE, SGX_INVALID_KEYNAME) is set in RAX and the data at the address specified by RCX is unmodified.

Requesting Keys

The KEYREQUEST structure (see Section 35.18.1) identifies the key to be provided. The Keyrequest.KeyName field identifies which type of key is requested.

Deriving Keys

Key derivation is based on a combination of the enclave specific values (see Table 38-66) and a processor key. Depending on the key being requested a field may either be included by definition or the value may be included from the KeyRequest. A "yes" in Table 38-66 indicates the value for the field is included from its default location, identified in the source row, and a "request" indicates the values for the field is included from its corresponding KeyRequest field.

Table 38-66. Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODID	ISVEXT PRODID	ISVFAM ILYID	MRENCLAVE	MRSIGNER	CONFIG ID	CONFIGS VN	RAND
Source	Key Dependent Constant	Y := SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES; R := AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;	CR_SGX OWNER EPOCH	Y := CPUSVN Register; R := Req.CPU SVN;	R := Req.ISV SVN;	SECS. ISVID	SECS.IS VEXTPR ODID	SECS.IS VFAMIL YID	SECS. MRENCLAVE	SECS. MRSIGNER	SECS.CO NFIGID	SECS.CO NFIGSVN	Req. KEYID
EINITTOKEN	Yes	Request	Yes	Request	Request	Yes	No	No	No	Yes	No	No	Request
Report	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	Yes	Yes	Request
Seal	Yes	Request	Yes	Request	Request	Request	Request	Request	Request	Request	Request	Request	Request
Provisioning	Yes	Request	No	Request	Request	Yes	No	No	No	Yes	No	No	Yes
Provisioning Seal	Yes	Request	No	Request	Request	Request	Request	Request	No	Yes	Request	Request	Yes

Keys that permit the specification of a CPU or ISV's code's, or enclave configuration's SVNs have additional requirements. The caller may not request a key for an SVN beyond the current CPU, ISV or enclave configuration's SVN, respectively.

Several keys are access controlled. Access to the Provisioning Key and Provisioning Seal key requires the enclave's ATTRIBUTES.PROVISIONKEY be set. The EINITTOKEN Key requires ATTRIBUTES.EINITTOKEN_KEY be set and SECS.MRSIGNER equal IA32_SGXLEPUBKEYHASH.

Some keys are derived based on a hardcode PKCS padding constant (352 byte string):

HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;

HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH

HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;

The error codes are:

Table 38-67. EGETKEY Return Value in RAX

Error Code (see Table 38-4)	Value	Description
No Error	0	EGETKEY successful.
SGX_INVALID_ATTRIBUTE		The KEYREQUEST contains a KEYNAME for which the enclave is not authorized.
SGX_INVALID_CPUSVN		If KEYREQUEST.CPUSVN is an unsupported platforms CPUSVN value.
SGX_INVALID_ISVSVN		If KEYREQUEST software SVN (ISVSVN or CONFIGSVN) is greater than the enclave's corresponding SVN.
SGX_INVALID_KEYNAME		If KEYREQUEST.KEYNAME is an unsupported value.

Concurrency Restrictions

Table 38-68. Base Concurrency Restrictions of EGETKEY

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent		
	OUTPUTDATA [DS:RCX]	Concurrent		

Table 38-69. Additional Concurrency Restrictions of EGETKEY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EGETKEY Operational Flow

Name	Type	Size (Bits)	Description
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_ATTRIBUTES		128	Temp Space for the calculation of the sealable Attributes.
TMP_ISVEXTPRODID		16 bytes	Temp Space for ISVEXTPRODID.
TMP_ISVPRODID		2 bytes	Temp Space for ISVPRODID.
TMP_ISVFAMILYID		16 bytes	Temp Space for ISVFAMILYID.
TMP_CONFIGID		64 bytes	Temp Space for CONFIGID.
TMP_CONFIGSVN		2 bytes	Temp Space for CONFIGSVN.
TMP_OUTPUTKEY		128	Temp Space for the calculation of the key.

```
(* Make sure KEYREQUEST is properly aligned and inside the current enclave *)
IF ( (DS:RBX is not 512Byte aligned) or (DS:RBX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RBX is an EPC address and the EPC page is valid *)
IF ( (DS:RBX does not resolve to an EPC address) or (EPCM(DS:RBX).VALID = 0) )
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #PF(DS:RBX);
FI;

(* Make sure OUTPUTDATA is properly aligned and inside the current enclave *)
IF ( (DS:RCX is not 16Byte aligned) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RCX is an EPC address and the EPC page is valid *)
IF ( (DS:RCX does not resolve to an EPC address) or (EPCM(DS:RCX).VALID = 0) )
```

```
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).W = 0) )
    THEN #PF(DS:RCX);
FI;

(* Verify RESERVED spaces in KEYREQUEST are valid *)
IF ( (DS:RBX).RESERVED ≠ 0) or (DS:RBX.KEYPOLICY.RESERVED ≠ 0) )
    THEN #GP(0); FI;

TMP_CURRENTSECS := CR_ACTIVE_SECS;

(* Verify that CONFIGSVN & New Policy bits are not used if KSS is not enabled *)
IF ((TMP_CURRENTSECS.ATTRIBUTES.KSS == 0) AND ((DS:RBX.KEYPOLICY & 0x003C ≠ 0) OR (DS:RBX.CONFIGSVN > 0)))
    THEN #GP(0); FI;
(* Determine which enclave attributes that must be included in the key. Attributes that must always be include INIT & DEBUG *)
REQUIRED_SEALING_MASK[127:0] := 00000000 00000000 00000000 00000003H;
TMP_ATTRIBUTES := (DS:RBX.ATTRIBUTEMASK | REQUIRED_SEALING_MASK) & TMP_CURRENTSECS.ATTRIBUTES;

(* Compute MISCSELECT fields to be included *)
TMP_MISCSELECT := DS:RBX.MISCMASK & TMP_CURRENTSECS.MISCSELECT

(* Compute CET_ATTRIBUTES fields to be included *)
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN TMP_CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES_ MASK & TMP_CURRENTSECS.CET_ATTRIBUTES; FI;
TMP_KEYDEPENDENCIES := 0;

CASE (DS:RBX.KEYNAME)
    SEAL_KEY:
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CONFIGSVN > TMP_CURRENTSECS.CONFIGSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;

        (*Include enclave identity?*)
```

```
            TMP_MRENCLAVE := 0;
            IF (DS:RBX.KEYPOLICY.MRENCLAVE = 1)
                THEN TMP_MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
            FI;
            (*Include enclave author?*)
            TMP_MRSIGNER := 0;
            IF (DS:RBX.KEYPOLICY.MRSIGNER = 1)
                THEN TMP_MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
            FI;
(* Include enclave product family ID? *)
    TMP_ISVFAMILYID := 0;
    IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
        THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
            FI;

    (* Include enclave product ID? *)
    TMP_ISVPRODID := 0;
    IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
        TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
            FI;

    (* Include enclave Config ID? *)
    TMP_CONFIGID := 0;
    TMP_CONFIGSVN := 0;
    IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
        TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;
        TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
            FI;

    (* Include enclave extended product ID? *)
    TMP_ISVEXTPRODID := 0;
    IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1 )
        TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
    FI;

            //Determine values key is based on
            TMP_KEYDEPENDENCIES.KEYNAME := SEAL_KEY;
            TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
            TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
            TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
            TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
            TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
            TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
            TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
            TMP_KEYDEPENDENCIES.MRENCLAVE := TMP_MRENCLAVE;
            TMP_KEYDEPENDENCIES.MRSIGNER := TMP_MRSIGNER;
            TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
            TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
            TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
            TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
            TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
            TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
            TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
            TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;
```

```
        TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
        IF CPUID.(EAX=12H, ECX=1):EAX[6] = 1
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := DS:RBX.CET_ATTRIBUTES _MASK;
        FI;
        BREAK;
    REPORT_KEY:
        //Determine values key is based on
        TMP_KEYDEPENDENCIES.KEYNAME := REPORT_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVSVN := 0;
        TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRSIGNER := 0;
        TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN := CR_CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := 0;
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := TMP_CURRENTSECS.CONFIGID;
        TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
        FI;
        BREAK;
    EINITTOKEN_KEY:
        (* Check ENCLAVE has EINITTOKEN Key capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.EINITTOKEN_KEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
```

```
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME := EINITTOKEN_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
        TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := 0;
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := 0;
        TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
        FI;
        BREAK;
PROVISION_KEY:
(* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME := PROVISION_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := 0;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
```

```
            TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
            TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
            TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
            TMP_KEYDEPENDENCIES.KEYID := 0;
            TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := 0;
            TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
            TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
            TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
            TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
            TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
            TMP_KEYDEPENDENCIES.CONFIGID := 0;
            IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
                  THEN
                        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
            FI;
            BREAK;
      PROVISION_SEAL_KEY:
            (* Check ENCLAVE has PROVISIONING capability *)
            IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
                  THEN
                        RFLAGS.ZF := 1;
                        RAX := SGX_INVALID_ATTRIBUTE;
                        GOTO EXIT;
            FI;
            IF (DS:RBX.CPUSVN is beyond current CPU configuration)
                  THEN
                        RFLAGS.ZF := 1;
                        RAX := SGX_INVALID_CPUSVN;
                        GOTO EXIT;
            FI;
            IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
                  THEN
                        RFLAGS.ZF := 1;
                        RAX := SGX_INVALID_ISVSVN;
                        GOTO EXIT;
            FI;
(* Include enclave product family ID? *)
      TMP_ISVFAMILYID := 0;
      IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
         THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
            FI;

      (* Include enclave product ID? *)
      TMP_ISVPRODID := 0;
      IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
         TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
            FI;

      (* Include enclave Config ID? *)
      TMP_CONFIGID := 0;
      TMP_CONFIGSVN := 0;
      IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
         TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;
```

```
        TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
            FI;

    (* Include enclave extended product ID? *)
    TMP_ISVEXTPRODID := 0;
    IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1)
        TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
    FI;

            (* Determine values key is based on *)
            TMP_KEYDEPENDENCIES.KEYNAME := PROVISION_SEAL_KEY;
            TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
            TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
            TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
            TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
            TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := 0;
            TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
            TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
            TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
            TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
            TMP_KEYDEPENDENCIES.KEYID := 0;
            TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
            TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
            TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
            TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
            TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
            TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
            TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;
            TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
            IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
                THEN
                    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
            FI;
            BREAK;
    DEFAULT:
            (* The value of KEYNAME is invalid *)
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_KEYNAME;
            GOTO EXIT:
ESAC;

(* Calculate the final derived key and output to the address in RCX *)
TMP_OUTPUTKEY := derivekey(TMP_KEYDEPENDENCIES);
DS:RCX[15:0] := TMP_OUTPUTKEY;
RAX := 0;
RFLAGS.ZF := 0;

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

Flags Affected

ZF is cleared if successful, otherwise ZF is set. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is outside the DS segment limit.
	If KEYREQUEST format is invalid.
#PF(error code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is not canonical.
	If KEYREQUEST format is invalid.
#PF(error code)	If a page fault occurs in accessing memory operands.

EMODPE—Extend an EPC Page Permissions

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLU[EMODPE]	IR	V/V	SGX2	This leaf function extends the access rights of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EMODPE (In)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function extends the access rights associated with an existing EPC page in the running enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not extend the page permissions will have no effect. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPE leaf function.

EMODPE Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EMODPE Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is not valid.	RCX does not contain an effective address of an EPC page in the running enclave.
SECINFO contains an invalid request.	

Concurrency Restrictions

Table 38-70. Base Concurrency Restrictions of EMODPE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODPE	Target [DS:RCX]	Concurrent		
	SECINFO [DS:RBX]	Concurrent		

Table 38-71. Additional Concurrency Restrictions of EMODPE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODPE	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EMODPE Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RBX).VALID = 0) or (EPCM(DS:RBX).R = 0) or (EPCM(DS:RBX).PENDING ≠ 0) or (EPCM(DS:RBX).MODIFIED ≠ 0) or
    (EPCM(DS:RBX).BLOCKED ≠ 0) or (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0xFFF)) )
    THEN #PF(DS:RBX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero )
    THEN #GP(0); FI;

(* Check security attributes of the EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN #GP(0); FI;

(* Re-Check security attributes of the EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
    THEN #PF(DS:RCX); FI;

(* Check for misconfigured SECINFO flags*)
IF ( (EPCM(DS:RCX).R = 0) and (SCRATCH_SECINFO.FLAGS.R = 0) and (SCRATCH_SECINFO.FLAGS.W ≠ 0) )
    THEN #GP(0); FI;
```

```
(* Update EPCM permissions *)
EPCM(DS:RCX).R := EPCM(DS:RCX).R | SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := EPCM(DS:RCX).W | SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := EPCM(DS:RCX).X | SCRATCH_SECINFO.FLAGS.X;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.

EREPORT—Create a Cryptographic Report of the Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLU[EREPORT]	IR	V/V	SGX1	This leaf function creates a cryptographic report of the enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX	RDX
IR	EREPORT (In)	Address of TARGETINFO (In)	Address of REPORTDATA (In)	Address where the REPORT is written to in an OUTPUTDATA (In)

Description

This leaf function creates a cryptographic REPORT that describes the contents of the enclave. This instruction leaf can only be executed when inside the enclave. The cryptographic report can be used by other enclaves to determine that the enclave is running on the same platform.

RBX contains the effective address of the MRENCLAVE value of the enclave that will authenticate the REPORT output, using the REPORT key delivered by EGETKEY command for that enclave. RCX contains the effective address of a 64-byte REPORTDATA structure, which allows the caller of the instruction to associate data with the enclave from which the instruction is called. RDX contains the address where the REPORT will be output by the instruction.

EREPORT Memory Parameter Semantics

TARGETINFO	REPORTDATA	OUTPUTDATA
Read access by Enclave	Read access by Enclave	Read/Write access by Enclave

This instruction leaf perform the following:

1. Validate the 3 operands (RBX, RCX, RDX) are inside the enclave.
2. Compute a report key for the target enclave, as indicated by the value located in RBX(TARGETINFO).
3. Assemble the enclave SECS data to complete the REPORT structure (including the data provided using the RCX (REPORTDATA) operand).
4. Computes a cryptographic hash over REPORT structure.
5. Add the computed hash to the REPORT structure.
6. Output the completed REPORT structure to the address in RDX (OUTPUTDATA).

The instruction fails if the operands are not properly aligned.

CR_REPORT_KEYID, used to provide key wearout protection, is populated with a statistically unique value on boot of the platform by a trusted entity within the SGX TCB.

The instruction faults if any of the following:

EREPORT Faulting Conditions

An effective address not properly aligned.	An memory address does not resolve in an EPC page.
If accessing an invalid EPC page.	If the EPC page is blocked.
May page fault.	

Concurrency Restrictions

Table 38-72. Base Concurrency Restrictions of EREPORT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EREPORT	TARGETINFO [DS:RBX]	Concurrent		
	REPORTDATA [DS:RCX]	Concurrent		
	OUTPUTDATA [DS:RDX]	Concurrent		

Table 38-73. Additional Concurrency Restrictions of EREPORT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREPORT	TARGETINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RDX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREPORT Operational Flow

Name	Type	Size (bits)	Description
TMP_ATTRIBUTES		32	Physical address of SECS of the enclave to which source operand belongs.
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_REPORTKEY		128	REPORTKEY generated by the instruction.
TMP_REPORT		3712	

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Address verification for TARGETINFO (RBX) *)
IF ( (DS:RBX is not 512Byte Aligned) or (DS:RBX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
```

```
    THEN #PF(DS:RBX);
FI;

(* Verify RESERVED spaces in TARGETINFO are valid *)
IF (DS:RBX.RESERVED != 0)
    THEN #GP(0); FI;

(* Address verification for REPORTDATA (RCX) *)
IF ( (DS:RCX is not 128Byte Aligned) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).R = 0) )
    THEN #PF(DS:RCX);
FI;

(* Address verification for OUTPUTDATA (RDX) *)
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RDX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX).VALID = 0)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX).BLOCKED = 1)
    THEN #PF(DS:RDX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RDX).ENCLAVEADDRESS ≠ (DS:RDX & ~0FFFH) ) or (EPCM(DS:RDX).W = 0) )
    THEN #PF(DS:RDX);
FI;

(* REPORT MAC needs to be computed over data which cannot be modified *)
TMP_REPORT.CPUSVN := CR_CPUSVN;
TMP_REPORT.ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
TMP_REPORT.ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
TMP_REPORT.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
TMP_REPORT.ISVSVN := TMP_CURRENTSECS.ISVSVN;
TMP_REPORT.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
TMP_REPORT.REPORTDATA := DS:RCX[511:0];
TMP_REPORT.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
```

```
TMP_REPORT.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
TMP_REPORT.MRRESERVED := 0;
TMP_REPORT.KEYID[255:0] := CR_REPORT_KEYID;
TMP_REPORT.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
TMP_REPORT.CONFIGID := TMP_CURRENTSECS.CONFIGID;
TMP_REPORT.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN TMP_REPORT.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES; FI;

(* Derive the report key *)
TMP_KEYDEPENDENCIES.KEYNAME := REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := 0;
TMP_KEYDEPENDENCIES.ISVSVN := 0;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := DS:RBX.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := DS:RBX.MEASUREMENT;
TMP_KEYDEPENDENCIES.MRSIGNER := 0;
TMP_KEYDEPENDENCIES.KEYID := TMP_REPORT.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := DS:RBX.MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := DS:RBX.CONFIGID;
TMP_KEYDEPENDENCIES.CONFIGSVN := DS:RBX.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
FI;

(* Calculate the derived key*)
TMP_REPORTKEY := derivekey(TMP_KEYDEPENDENCIES);

(* call cryptographic CMAC function, CMAC data are not including MAC&KEYID *)
TMP_REPORT.MAC := cmac(TMP_REPORTKEY, TMP_REPORT[3071:0] );
DS:RDX[3455: 0] := TMP_REPORT;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If the address in RCS is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RCX is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

ERESUME—Re-Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLU[ERESUME]	IR	V/V	SGX1	This leaf function is used to re-enter an enclave after an interrupt.

Instruction Operand Encoding

Op/En	RAX	RBX	RCX
IR	ERESUME (In)	Address of a TCS (In)	Address of AEP (In)

Description

The ENCLU[ERESUME] instruction resumes execution of an enclave that was interrupted due to an exception or interrupt, using the machine state previously stored in the SSA.

ERESUME Memory Parameter Semantics

TCS
Enclave read/write access

The instruction faults if any of the following occurs:

Address in RBX is not properly aligned.	Any TCS.FLAGS's must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use by another enclave.	Either of TCS-specified FS and GS segment is not a subset of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
Offsets 520-535 of the XSAVE area not 0.	The bit vector stored at offset 512 of the XSAVE area must be a subset of SECS.ATTRIBUTES.XFRM.
The SSA frame is not valid or in use.	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.

The following operations are performed by ERESUME:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or an asynchronous exit due to any Interrupt event.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 40.1.2):
  - On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 40.2.5).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 40.2.3).
- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.

- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 40.2.3):
  - All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
  - PEBS is suppressed.
  - AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set.
  - If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 38-74. Base Concurrency Restrictions of ERESUME

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ERESUME	TCS [DS:RBX]	Shared	#GP	

Table 38-75. Additional Concurrency Restrictions of ERESUME

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ERESUME	TCS [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ERESUME Operational Flow

Name	Type	Size	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_TARGET	Effective Address	32/64	Address of first instruction inside enclave at which execution is to resume.
TMP_SECS	Effective Address	32/64	Physical address of SECS for this enclave.
TMP_SSA	Effective Address	32/64	Address of current SSA frame.
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.
TMP_BRANCH_RECORD	LBR Record		From/to addresses to be pushed onto the LBR stack.
TMP_NOTIFY	Boolean	1	When set to 1, deliver an AEX notification.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)
IF (TMP_MODE64 = 0 and (DS not usable or ( ( DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1))))
```

```
    THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)
IF (TMP_MODE64 = 0)
    THEN
        IF(CS.base ≠ 0 or DS.base ≠ 0) #GP(0); FI;
        IF(ES usable and ES.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.B = 0) #GP(0); FI;
FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (CS:RCX is not canonical))
    THEN #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions are operating on TCS)
    THEN #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) or (EPCM(DS:RBX).PT ≠ PT_TCS))
    THEN #PF(DS:RBX); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned))
    THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS := Address of SECS for TCS;

(* Make sure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & FFFFFFFFFFFFFFFCH) ≠ 0)
    THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
```

```
    THEN #GP(0); FI;

(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT))
    THEN #GP(0); FI;

IF (CR4.OSFXSR = 0)
    THEN #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    THEN
        IF (TMP_SECS.ATTRIBUTES.XFRM ≠ 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUTES.XFRM & XCR0) ≠ TMP_SECS.ATTRIBUTES.XFRM) THEN #GP(0); FI;
FI;

IF ( (DS:RBX).CSSA.FLAGS.DBGOPTIN = 0) and (DS:RBX).CSSA.FLAGS.AEXNOTIFY ≠ TMP_SECS.ATTRIBUTES.AEXNOTIFY))
    THEN #GP(0); FI;

(* Make sure the SSA contains at least one active frame *)
IF ( (DS:RBX).CSSA = 0)
    THEN #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA := (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * ( (DS:RBX).CSSA - 1);
TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;
    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE_.MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
        (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0) )
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
Check that DS:TMP_SSA_PAGE is read/write accessible;
If a fault occurs, release locks, abort and deliver that fault;
IF (DS:TMP_GPR does not resolve to EPC page)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
```

```
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0))
    THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)
    THEN
        IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
FI;

CR_GPR_PA := Physical_Address (DS: TMP_GPR);

IF ((DS:RBX).FLAGS.AEXNOTIFY = 1) and (DS:TMP_GPR.AEXNOTIFY[0] = 1))
    THEN
        TMP_NOTIFY := 1;
    ELSE
        TMP_NOTIFY := 0;
FI;

IF (TMP_NOTIFY = 1)
    THEN
        (* Make sure the SSA contains at least one more frame *)
        IF ((DS:RBX).CSSA ≥ (DS:RBX).NSSA)
            THEN #GP(0); FI;

        TMP_SSA := TMP_SSA + 4096 * TMP_SECS.SSAFRAMESIZE;
        TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

        FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
            (* Check page is read/write accessible *)
            Check that DS:TMP_SSA_PAGE is read/write accessible;
            If a fault occurs, release locks, abort and deliver that fault;

            IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
                THEN #PF(DS:TMP_SSA_PAGE); FI;
            IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
                THEN #PF(DS:TMP_SSA_PAGE); FI;
            IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
                THEN #PF(DS:TMP_SSA_PAGE); FI;
            IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or
            (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
                THEN #PF(DS:TMP_SSA_PAGE); FI;
            IF ((EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMP_SSA_PAGE) or
            (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
            (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
            (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0))
                THEN #PF(DS:TMP_SSA_PAGE); FI;
            CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);
        ENDFOR
```

```
        (* Compute address of GPR area*)
        TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
        If a fault occurs; release locks, abort and deliver that fault;

        IF (DS:TMP_GPR does not resolve to EPC page)
            THEN #PF(DS:TMP_GPR); FI;
        IF (EPCM(DS:TMP_GPR).VALID = 0)
            THEN #PF(DS:TMP_GPR); FI;
        IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
            THEN #PF(DS:TMP_GPR); FI;
        IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
            THEN #PF(DS:TMP_GPR); FI;
        IF ((EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or
        (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
        (EPCM(DS:TMP_GPR).ENCLAVESECS EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0))
            THEN #PF(DS:TMP_GPR); FI;

        IF (TMP_MODE64 = 0)
            THEN
                IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
        FI;

        CR_GPR_PA := Physical_Address (DS: TMP_GPR);

        TMP_TARGET := (DS:RBX).OENTRY + TMP_SECS.BASEADDR;
    ELSE
        TMP_TARGET := (DS:TMP_GPR).RIP;
FI;

IF (TMP_MODE64 = 1)
    THEN
        IF (TMP_TARGET is not canonical) THEN #GP(0); FI;
    ELSE
        IF (TMP_TARGET > CS limit) THEN #GP(0); FI;
FI;

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    THEN
        TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_FSLIMIT := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
        TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        TMP_GSLIMIT := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
        (* if FS wrap-around, make sure DS has no holes*)
        IF (TMP_FSLIMIT < TMP_FSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
        (* if GS wrap-around, make sure DS has no holes*)
        IF (TMP_GSLIMIT < TMP_GSBASE)
            THEN
```

```
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
    ELSE
        IF (TMP_NOTIFY = 1)
            THEN
                TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
                TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
            ELSE
                TMP_FSBASE := DS:TMP_GPR.FSBASE;
                TMP_GSBASE := DS:TMP_GPR.GSBASE;
        FI;
        IF ((TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
            THEN #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE))
    THEN #GP(0); FI;

TMP_IA32_U_CET := 0
TMP_SSP := 0

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        IF ( CR4.CET = 0 )
            THEN
                (* If part does not support CET or CET has not been enabled and enclave requires CET then fail *)
                IF (TMP_SECS.CET_ATTRIBUTES ≠ 0 OR TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) #GP(0); FI;
        FI;
        (* If indirect branch tracking or shadow stacks enabled but CET state save area is not 16B aligned then fail ERESUME *)
        IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN = 1 OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN = 1)
            THEN
                IF (DS:RBX.OCETSSA is not 16B aligned) #GP(0); FI;
        FI;

IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN)
    THEN
        (* Setup CET state from SECS, note tracker goes to IDLE *)
        TMP_IA32_U_CET = TMP_SECS.CET_ATTRIBUTES;
        IF (TMP_IA32_U_CET.LEG_IW_EN = 1 AND TMP_IA32_U_CET.ENDBR_EN = 1)
            THEN
                TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.BASEADDR;
                TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.CET_LEG_BITMAP_BASE;
        FI;

        (* Compute linear address of what will become new CET state save area and cache its PA *)
        IF (TMP_NOTIFY = 1)
            THEN
                TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA) * 16;
            ELSE
                TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA - 1) * 16;
        FI;
```

```
        TMP_CET_SAVE_PAGE = TMP_CET_SAVE_AREA & ~0xFFF;

        Check the TMP_CET_SAVE_PAGE page is read/write accessible
        If fault occurs release locks, abort and deliver fault

        (* read the EPCM VALID, PENDING, MODIFIED, BLOCKED and PT fields atomically *)
        IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS))
              THEN
                   #PF(DS:TMP_CET_SAVE_PAGE);
        FI;

        CR_CET_SAVE_AREA_PA := Physical address(DS:TMP_CET_SAVE_AREA)
        IF (TMP_NOTIFY = 1)
              THEN
                   IF TMP_IA32_U_CET.SH_STK_EN = 1
                        THEN TMP_SSP = TCS.PREVSSP; FI;
              ELSE
                   TMP_SSP = CR_CET_SAVE_AREA_PA.SSP
                   TMP_IA32_U_CET.TRACKER = CR_CET_SAVE_AREA_PA.TRACKER;
                   TMP_IA32_U_CET.SUPPRESS = CR_CET_SAVE_AREA_PA.SUPPRESS;
                   IF ( (TMP_MODE64 = 1 AND TMP_SSP is not canonical) OR
                         (TMP_MODE64 = 0 AND (TMP_SSP & 0xFFFFFFFF00000000) ≠ 0) OR
                        (TMP_SSP is not 4 byte aligned) OR
                        (TMP_IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH AND TMP_IA32_U_CET.SUPPRESS = 1) OR
                        (CR_CET_SAVE_AREA_PA.Reserved ≠ 0) ) #GP(0); FI;
                   FI;
        FI;
FI;

IF (TMP_NOTIFY = 0)
    THEN
        (* SECS.ATTRIBUTES.XFRM selects the features to be saved. *)
        (* CR_XSAVE_PAGE_n: A list of 1 or more physical address of pages that contain the XSAVE area. *)
        XRSTOR(TMP_MODE64, SECS.ATTRIBUTES.XFRM, CR_XSAVE_PAGE_n);

        IF (XRSTOR failed with #GP)
              THEN
                   DS:RBX.STATE := INACTIVE;
                   #GP(0);
        FI;
FI;

CR_ENCLAVE_MODE := 1;
CR_ACTIVE_SECS := TMP_SECS;
CR_ELRANGE := (TMP_SECS.BASEADDR, TMP_SECS.SIZE);
```

```
(* Save sate for possible AEXs *)
CR_TCS_PA := Physical_Address (DS:RBX);
CR_TCS_LA := RBX;
CR_TCS_LA.AEP := RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector := FS.selector;
CR_SAVE_FS_base := FS.base;
CR_SAVE_FS_limit := FS.limit;
CR_SAVE_FS_access_rights := FS.access_rights;
CR_SAVE_GS_selector := GS.selector;
CR_SAVE_GS_base := GS.base;
CR_SAVE_GS_limit := GS.limit;
CR_SAVE_GS_access_rights := GS.access_rights;

IF (TMP_NOTIFY = 1)
    THEN
        (* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
        IF (CR4.OSXSAVE = 1)
            THEN
                CR_SAVE_XCR0 := XCR0;
                XCR0 := TMP_SECS.ATTRIBUTES.XFRM;
        FI;
FI;

RIP := TMP_TARGET;

IF (TMP_NOTIFY = 1)
    THEN
        RCX := RIP;
        RAX := (DS:RBX).CSSA;
        (* Save the outside RSP and RBP so they can be restored on interrupt or EEXIT *)
        DS:TMP_SSA.U_RSP := RSP;
        DS:TMP_SSA.U_RBP := RBP;
    ELSE
        Restore_GPRs from DS:TMP_GPR;

        (*Restore the RFLAGS values from SSA*)
        RFLAGS.CF := DS:TMP_GPR.RFLAGS.CF;
        RFLAGS.PF := DS:TMP_GPR.RFLAGS.PF;
        RFLAGS.AF := DS:TMP_GPR.RFLAGS.AF;
        RFLAGS.ZF := DS:TMP_GPR.RFLAGS.ZF;
        RFLAGS.SF := DS:TMP_GPR.RFLAGS.SF;
        RFLAGS.DF := DS:TMP_GPR.RFLAGS.DF;
        RFLAGS.OF := DS:TMP_GPR.RFLAGS.OF;
        RFLAGS.NT := DS:TMP_GPR.RFLAGS.NT;
        RFLAGS.AC := DS:TMP_GPR.RFLAGS.AC;
        RFLAGS.ID := DS:TMP_GPR.RFLAGS.ID;
        RFLAGS.RF := DS:TMP_GPR.RFLAGS.RF;
        RFLAGS.VM := 0;
        IF (RFLAGS.IOPL = 3)
            THEN RFLAGS.IF := DS:TMP_GPR.RFLAGS.IF; FI;

        IF (TCS.FLAGS.OPTIN = 0)
```

```
            THEN RFLAGS.TF := 0; FI;

        (* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
        IF (CR4.OSXSAVE = 1)
            THEN
                CR_SAVE_XCR0 := XCR0;
                XCR0 := TMP_SECS.ATTRIBUTES.XFRM;
        FI;

        (* Pop the SSA stack*)
        (DS:RBX).CSSA := (DS:RBX).CSSA -1;
FI;

(* Do the FS/GS swap *)
FS.base := TMP_FSBASE;
FS.limit := DS:RBX.FSLIMIT;
FS.type := 0001b;
FS.W := DS.W;
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS.L;
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS.W;
GS.S := 1;
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS.L;
GS.unusable := 0;
GS.selector := 0BH;

CR_DBGOPTIN := TCS.FLAGS.DBGOPTIN;
Suppress all code breakpoints that are outside ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        Suppress all code breakpoints that overlap with ELRANGE;
        CR_SAVE_TF := RFLAGS.TF;
        RFLAGS.TF := 0;
        Suppress any MTF VM exits during execution of the enclave;
        Clear all pending debug exceptions;
        Clear any pending MTF VM exit;
    ELSE
```

```
        IF (TMP_NOTIFY = 1)
            THEN
                IF RFLAGS.TF = 1
                    THEN pend a single-step #DB at the end of ERESUME; FI;
                IF the "monitor trap flag" VM-execution control is set
                    THEN pend an MTF VM exit at the end of ERESUME; FI;
        ELSE
            Clear all pending debug exceptions;
            Clear pending MTF VM exits;
        FI;
FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* Save enclosing application CET state into save registers *)
        CR_SAVE_IA32_U_CET := IA32_U_CET
        (* Setup enclave CET state *)
        IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
            THEN
                CR_SAVE_SSP := SSP
                SSP := TMP_SSP;
        FI;
        IA32_U_CET := TMP_IA32_U_CET;
FI;

(* Assure consistent translations *)
Flush_linear_context;
Clear_Monitor_FSM;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment.
	If any reserved field in the TCS FLAG is set.
	If the target address is not within the CS segment.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	If a page fault occurs in accessing memory.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment.
	If any reserved field in the TCS FLAG is set.
	If the target address is not canonical.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
	If SECS.ATTRIBUTES.AEXNOTIFY ≠ TCS.FLAGS.AEXNOTIFY and TCS.FLAGS.DBGOPTIN = 0.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

38.5 INTEL® SGX VIRTUALIZATION LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLV instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EDECVIRTCHILD—Decrement VIRTCHILDCNT in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLV[EDECVIRTCHILD]	IR	V/V	EAX[5]	This leaf function decrements the SECS VIRTCHILDCNT field.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDECVIRTCHILD (In)	Return error code (Out)	Address of an enclave page (In)	Address of an SECS page (In)

Description

This instruction decrements the SECS VIRTCHILDCNT field. This instruction can only be executed when current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

EDECVIRTCHILD Memory Parameter Semantics

EPCPAGE	SECS
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EDECVIRTCHILD Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A page fault occurs in accessing memory operands.
DS segment is unusable (32b mode).	RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
A memory address is in a non-canonical form (64b mode).	RCX does not refer to an SECS page.
A memory operand is not properly aligned.	RBX does not refer to an enclave page associated with SECS referenced in RCX.

Concurrency Restrictions

Table 38-76. Base Concurrency Restrictions of EDECVIRTCHILD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDECVIRTCHILD	Target [DS:RBX]	Shared	SGX_EPC_PAGE_ CONFLICT	
	SECS [DS:RCX]	Concurrent		

Table 38-77. Additional Concurrency Restrictions of EDECVIRTCHILD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDECVIRTCHILD	Target [DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDECVIRTCHILD Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_VIRTCHILDCNT	Integer	64	Number of virtual child pages.

EDECVIRTCHILD Return Value in RAX

Error	Value	Description
No Error	0	EDECVIRTCHILD Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.
SGX_INVALID_COUNTER		Attempt to decrement counter that is already zero.

```
(* check alignment of DS:RBX *)
IF (DS:RBX is not 4K aligned) THEN
   #GP(0); FI;

(* check DS:RBX is an linear address of an EPC page *)
IF (DS:RBX does not resolve within an EPC) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check DS:RCX is an linear address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
   #PF(DS:RCX, PFEC.SGX); FI;

(* Check the EPCPAGE for concurrency *)
IF (EPCPAGE is being modified) THEN
   RFLAGS.ZF = 1;
   RAX = SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check that the EPC page is valid *)
IF (EPCM(DS:RBX).VALID = 0) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check that the EPC page has the correct type and that the back pointer matches the pointer passed as the pointer to parent *)
IF ((EPCM(DS:RBX).PAGE_TYPE = PT_REG) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_TCS) or
```

```
    (EPCM(DS:RBX).PAGE_TYPE = PT_TRIM) or
    (EPCM(DS:RBX).PAGE_TYPE = PT_SS_FIRST) or
    (EPCM(DS:RBX).PAGE_TYPE = PT_SS_REST))
      THEN
    (* get the SECS of DS:RBX *)
    TMP_SECS := Address of SECS for (DS:RBX);
ELSE IF (EPCM(DS:RBX).PAGE_TYPE = PT_SECS) THEN
    (* get the physical address of DS:RBX *)
    TMP_SECS := Physical_Address(DS:RBX);
ELSE
    (* EDECVIRTCHILD called on page of incorrect type *)
    #PF(DS:RBX, PFEC.SGX); FI;

IF (TMP_SECS ≠ Physical_Address(DS:RCX)) THEN
    #GP(0); FI;

(* Atomically decrement virtchild counter and check for underflow *)
Locked_Decrement(SECS(TMP_SECS).VIRTCHILDCNT);
IF (There was an underflow) THEN
    Locked_Increment(SECS(TMP_SECS).VIRTCHILDCNT);
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_COUNTER;
    goto DONE;
FI;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF is set if EDECVIRTCHILD fails due to concurrent operation with another SGX instruction, or if there is a VIRTCHILDCNT underflow. Otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
	RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
	If RCX does not refer to an SECS page.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form. If a memory operand is not properly aligned. RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands. If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS). If RCX does not refer to an SECS page.

EINCVIRTCHILD—Increment VIRTCHILDCNT in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLV[EINCVIRTCHILD]	IR	V/V	EAX[5]	This leaf function increments the SECS VIRTCHILDCNT field.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EINCVIRTCHILD (In)	Return error code (Out)	Address of an enclave page (In)	Address of an SECS page (In)

Description

This instruction increments the SECS VIRTCHILDCNT field. This instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create a linear address. Segment override is not supported.

EINCVIRTCHILD Memory Parameter Semantics

EPCPAGE	SECS
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EINCVIRTCHILD Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A page fault occurs in accessing memory operands.
DS segment is unusable (32b mode).	RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
A memory address is in a non-canonical form (64b mode).	RCX does not refer to an SECS page.
A memory operand is not properly aligned.	RBX does not refer to an enclave page associated with SECS referenced in RCX.

Concurrency Restrictions

Table 38-78. Base Concurrency Restrictions of EINCVIRTCHILD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EINCVIRTCHILD	Target [DS:RBX]	Shared	SGX_EPC_PAGE_ CONFLICT	
	SECS [DS:RCX]	Concurrent		

Table 38-79. Additional Concurrency Restrictions of EINCVIRTCHILD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EINCVIRTCHILD	Target [DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EINCVIRTCHILD Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.

EINCVIRTCHILD Return Value in RAX

Error	Value	Description
No Error	0	EINCVIRTCHILD Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.

```
(* check alignment of DS:RBX *)
IF (DS:RBX is not 4K aligned) THEN
   #GP(0); FI;

(* check DS:RBX is an linear address of an EPC page *)
IF (DS:RBX does not resolve within an EPC) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check DS:RCX is an linear address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
   #PF(DS:RCX, PFEC.SGX); FI;

(* Check the EPCPAGE for concurrency *)
IF (EPCPAGE is being modified) THEN
   RFLAGS.ZF = 1;
   RAX = SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check that the EPC page is valid *)
IF (EPCM(DS:RBX).VALID = 0) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check that the EPC page has the correct type and that the back pointer matches the pointer passed as the pointer to parent *)
IF ((EPCM(DS:RBX).PAGE_TYPE = PT_REG) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_TCS) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_TRIM) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_SS_FIRST) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_SS_REST))
```

```
    THEN
  (* get the SECS of DS:RBX *)
  TMP_SECS := Address of SECS for (DS:RBX);
ELSE IF (EPCM(DS:RBX).PAGE_TYPE = PT_SECS) THEN
  (* get the physical address of DS:RBX *)
  TMP_SECS := Physical_Address(DS:RBX);
ELSE
  (* EINCVIRTCHILD called on page of incorrect type *)
  #PF(DS:RBX, PFEC.SGX); FI;

IF (TMP_SECS ≠ Physical_Address(DS:RCX)) THEN
  #GP(0); FI;

(* Atomically increment virtchild counter *)
Locked_Increment(SECS(TMP_SECS).VIRTCHILDCNT);

RFLAGS.ZF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF is set if EINCVIRTCHILD fails due to concurrent operation with another SGX instruction; otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
	RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
	If RCX does not refer to an SECS page.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
	RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
	If RCX does not refer to an SECS page.

ESETCONTEXT—Set the ENCLAVECONTEXT Field in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLV[ESETCONTEXT]	IR	V/V	EAX[5]	This leaf function sets the ENCLAVECONTEXT field in SECS.

Instruction Operand Encoding

Op/En	EAX		RCX	RDX
IR	ESETCONTEXT (In)	Return error code (Out)	Address of the destination EPC page (In, EA)	Context Value (In, EA)

Description

The ESETCONTEXT leaf overwrites the ENCLAVECONTEXT field in the SECS. ECREATE and ELD of an SECS set the ENCLAVECONTEXT field in the SECS to the address of the SECS (for access later in ERDINFO). The ESETCONTEXT instruction allows a VMM to overwrite the default context value if necessary, for example, if the VMM is emulating ECREATE or ELD on behalf of the guest.

The content of RCX is an effective address of the SECS page to be updated, RDX contains the address pointing to the value to be stored in the SECS. The DS segment is used to create linear address. Segment override is not supported.

The instruction fails if:

- The operand is not properly aligned.
- RCX does not refer to an SECS page.

ESETCONTEXT Memory Parameter Semantics

EPCPAGE	CONTEXT
Read access permitted by Enclave	Read/Write access permitted by Non Enclave

The instruction faults if any of the following:

ESETCONTEXT Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A memory operand is not properly aligned.
DS segment is unusable (32b mode).	A page fault occurs in accessing memory operands.
A memory address is in a non-canonical form (64b mode).	

Concurrency Restrictions

Table 38-80. Base Concurrency Restrictions of ESETCONTEXT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ESETCONTEXT	SECS [DS:RCX]	Shared	SGX_EPC_PAGE_ CONFLICT	

Table 38-81. Additional Concurrency Restrictions of ESETCONTEXT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ESETCONTEXT	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ESETCONTEXT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_CONTEXT	CONTEXT	64	Data Value of CONTEXT.

ESETCONTEXT Return Value in RAX

Error	Value	Description
No Error	0	ESETCONTEXT Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.

```
(* check alignment of the EPCPAGE (RCX) *)
IF (DS:RCX is not 4KByte Aligned) THEN
   #GP(0); FI;

 (* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
IF (DS:RCX does not resolve within an EPC)THEN
   #PF(DS:RCX, PFEC.SGX); FI;

(* check alignment of the CONTEXT field (RDX) *)
IF (DS:RDX is not 8Byte Aligned) THEN
   #GP(0); FI;

 (* Load CONTEXT into local variable *)
TMP_CONTEXT := DS:RDX

(* Check the EPC page for concurrency *)
IF (EPC page is being modified) THEN
   RFLAGS.ZF := 1;
   RFLAGS.CF := 0;
   RAX := SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check page validity *)
IF (EPCM(DS:RCX).VALID = 0) THEN
   #PF(DS:RCX, PFEC.SGX);
FI;

(* check EPC page is an SECS page *)
```

```
IF (EPCM(DS:RCX).PT is not PT_SECS) THEN
   #PF(DS:RCX, PFEC.SGX);
FI;

(* load the context value into SECS(DS:RCX).ENCLAVECONTEXT *)
SECS(DS:RCX).ENCLAVECONTEXT := TMP_CONTEXT;

RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is set if ESETCONTEXT fails due to concurrent operation with another SGX instruction; otherwise cleared. CF, PF, AF, OF, and SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

CHAPTER 39
INTEL® SGX INTERACTIONS WITH IA32 AND INTEL® 64 ARCHITECTURE

Intel® SGX provides Intel® Architecture with a collection of enclave instructions for creating protected execution environments on processors supporting IA32 and Intel® 64 architectures. These Intel SGX instructions are designed to work with legacy software and the various IA32 and Intel 64 modes of operation.

39.1 INTEL® SGX AVAILABILITY IN VARIOUS PROCESSOR MODES

The Intel SGX extensions (see Table 34-1) are available only when the processor is executing in protected mode of operation. Additionally, the extensions are not available in System Management Mode (SMM) of operation or in Virtual 8086 (VM86) mode of operation. Finally, all leaf functions of ENCLU and ENCLS require CR0.PG enabled.

The exact details of exceptions resulting from illegal modes and their priority are listed in the reference pages of ENCLS and ENCLU.

39.2 IA32_FEATURE_CONTROL

IA32_FEATURE_CONTROL MSR provides two new bits related to two aspects of Intel SGX: using the instruction extensions and launch control configuration.

39.2.1 Availability of Intel SGX

IA32_FEATURE_CONTROL[bit 18] allows BIOS to control the availability of Intel SGX extensions. For Intel SGX extensions to be available on a logical processor, bit 18 in the IA32_FEATURE_CONTROL MSR on that logical processor must be set, and IA32_FEATURE_CONTROL MSR on that logical processor must be locked (bit 0 must be set). See Section 34.7.1 for additional details. OS is expected to examine the value of bit 18 prior to enabling Intel SGX on the thread, as the settings of bit 18 is not reflected by CPUID.

39.2.2 Intel SGX Launch Control Configuration

The IA32_SGXLEPUBKEYHASHn MSRs used to configure authorized launch enclaves' MRSIGNER digest value. They are present on logical processors that support the collection of SGX1 leaf functions (i.e., CPUID.(EAX=12H, ECX=00H):EAX[0] = 1) and that CPUID.(EAX=07H, ECX=00H):ECX[30] = 1. IA32_FEATURE_CONTROL[bit 17] allows to BIOS to enable write access to these MSRs. If IA32_FEATURE_CONTROL.LE_WR (bit 17) is set to 1 and IA32_FEATURE_CONTROL is locked on that logical processor, IA32_SGXLEPUBKEYHASH MSRs on that logical processor are writeable. If this bit 17 is not set or IA32_FEATURE_CONTROL is not locked, IA32_SGXLEPUBKEY-HASH MSRs are read only. See Section 36.1.4 for additional details.

39.3 INTERACTIONS WITH SEGMENTATION

39.3.1 Scope of Interaction

Intel SGX extensions are available only when the processor is executing in a protected mode operation (see Section 39.1 for Intel SGX availability in various processor modes). Enclaves abide by all the segmentation policies set up by the OS, but they can be more restrictive than the OS.

Intel SGX interacts with segmentation at two levels:

- The Intel SGX instruction (see the enclave instruction in Table 34-1).

- While executing inside an enclave (legacy instructions and enclave instructions permitted inside an enclave).

39.3.2 Interactions of Intel® SGX Instructions with Segment, Operand, and Addressing Prefixes

All the memory operands used by the Intel SGX instructions are interpreted as offsets within the data segment (DS). The segment-override prefix on Intel SGX instructions is ignored.

Operand size is fixed for each enclave instruction. The operand-size prefix is reserved, and results in a #UD exception if used.

All address sizes are determined by the operating mode of the processor. The address-size prefix is ignored. This implies that while operating in 64-bit mode of operation, the address size is always 64 bits, and while operating in 32-bit mode of operation, the address size is always 32 bits. Additionally, when operating in 16-bit addressing, memory operands used by enclave instructions use 32 bit addressing; the value of CS.D is ignored.

39.3.3 Interaction of Intel® SGX Instructions with Segmentation

All leaf functions of ENCLU and ENCLS instructions require that the DS segment be usable, and be an expand-up segment. Failing this check results in generation of a #GP(0) exception.

The Intel SGX leaf functions used for entering the enclave (ENCLU[EENTER] and ENCLU[ERESUME]) operate as follows:

- All usable segment registers except for FS and GS have a zero base.
- The contents of the FS/GS segment registers (including the hidden portion) is saved in the processor.
- New FS and GS values compatible with enclave security are loaded from the TCS
- The linear ranges and access rights available under the newly-loaded FS and GS must abide to OS policies by ensuring they are subsets of the linear-address range and access rights available for the DS segment.
- The CS segment mode (64-bit, compatible, or 32 bit modes) must be consistent with the segment mode for which the enclave was created, as indicated by the SECS.ATTRIBUTES.MODE64 bit, and that the CPL of the logical processor is 3

An exit from the enclave either via ENCLU[EEXIT] or via an AEX restores the saved values of FS/GS segment registers.

39.3.4 Interactions of Enclave Execution with Segmentation

During the course of execution, enclave code abides by all segmentation policies as dictated by IA32 and Intel 64 Architectures, and generates appropriate exceptions on violations.

Additionally, any attempt by software executing inside an enclave to modify the processor's segmentation state (e.g., via MOV seg register, POP seg register, LDS, far jump, etc; excluding WRFSBASE/WRGSBASE) results in the generation of a #UD. See Section 36.6.1 for more information.

Upon enclave entry via the EENTER leaf function, FS is loaded from the (TCS.OFSBASE + SECS.BASEADDR) and TCS.FSLIMIT fields and GS is loaded from the (TCS.OGSBASE + SECS.BASEADDR) and TCS.GSLIMIT fields.

Execution of WRFSBASE and WRGSBASE from inside a 64-bit enclave is allowed. The processor will save the new values into the current SSA frame on an asynchronous exit (AEX) and restore them back on enclave entry via ENCLU[ERESUME] instruction.

39.4 INTERACTIONS WITH PAGING

Intel SGX instructions are available only when the processor is executing in a protected mode of operation. Additionally, all Intel SGX leaf functions except for EDBGRD and EDBGWR are available only if paging is enabled. Any attempt to execute these leaf functions with paging disabled results in an invalid-opcode exception (#UD). As with

segmentation, enclaves abide by all the paging policies set up by the OS, but they can be more restrictive than the OS.

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the DS segment, and the linear addresses generated by combining these offsets with DS segment register are subject to paging-based access control if paging is enabled at the time of the execution of the leaf function.

Since the ENCLU[EENTER] and ENCLU[ERESUME] can only be executed when paging is enabled, and since paging cannot be disabled by software running inside an enclave (recall that enclaves always run with CPL = 3), enclave execution is always subject to paging-based access control. The Intel SGX access control itself is implemented as an extension to the existing paging modes. See Section 35.5 for details.

Execution of Intel SGX instructions may set accessed and dirty flags on accesses to EPC pages that do not fault even if the instruction later causes a fault for some other reason.

39.5 INTERACTIONS WITH VMX

Intel SGX functionality (including SGX1 and SGX2) can be made available to software running in either VMX root operation or VMX non-root operation, as long as the processor is using a legal mode of operation (see Section 39.1).

A VMM has the flexibility to configure a VMCS to permit a guest to use any subset of the ENCLS leaf functions. Availability of the ENCLU leaf functions in VMX non-root operation has the same requirement as ENCLU leaf functions outside of a virtualized environment.

Details of the VMCS control to allow VMM to configure support of Intel SGX in VMX non-root operation is described in Section 39.5.1

39.5.1 VMM Controls to Configure Guest Support of Intel® SGX

Intel SGX capabilities are primarily exposed to the software via the CPUID instruction. VMMs can virtualize CPUID instruction to expose/hide this capability to/from guests.

Some of Intel SGX resources are exposed/controlled via model-specific registers (see Section 34.7). VMMs can virtualize these MSRs for the guests using the MSR bitmaps referenced by pointers in the VMCS.

The VMM can partition the Enclave Page Cache, and assign various partitions to (a subset of) its guests via the usual memory-virtualization techniques such as paging or the extended page table mechanism (EPT).

The VMM can set the "enable ENCLS exiting" VM-execution controls to cause a VM exit when the ENCLS instruction is executed in VMX non-root operation. If the "enable ENCLS exiting" control is 0, all of the ENCLS leaf functions are permitted in VMX non-root operation. If the "enable ENCLS exiting" control is 1, execution of ENCLS leaf functions in VMX non-root operation is governed by consulting the bits in a new 64-bit VM-execution control field called the ENCLS-exiting bitmap (Each bit in the bitmap corresponds to an ENCLS leaf function with an EAX value that is identical to the bit's position). When bits in the "ENCLS-exiting bitmap" are set, attempts to execute the corresponding ENCLS leaf functions in VMX non-root operation causes VM exits. The checking for these VM exits occurs immediately after checking that CPL = 0.

39.5.2 Interactions with the Extended Page Table Mechanism (EPT)

Intel SGX instructions are fully compatible with the extended page-table mechanism (EPT; see Section 29.3).

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the DS segment, and the linear addresses generated by combining these offsets with DS segment register are subject to paging and EPT. As with paging, enclaves abide by all the policies set up by the VMM.

The Intel SGX access control itself is implemented as an extension to paging and EPT, and may be more restrictive. See Section 39.4 for details of this extension.

An execution of an Intel SGX instruction may set accessed and dirty flags for EPT (when enabled; see Section 29.3.5) on accesses to EPC pages that do not fault or cause VM exits even if the instruction later causes a fault or VM exit for some other reason.

39.5.3 Interactions with APIC Virtualization

This section applies to Intel SGX in VMX non-root operation when the "virtualize APIC accesses" VM-execution control is 1.

A memory access by an enclave instruction that implicitly uses a cached physical address is never checked for overlap with the APIC-access page. Such accesses never cause APIC-access VM exits and are never redirected to the virtual-APIC page. Implicit memory accesses can only be made to the SECS, the TCS, or the SSA of an enclave (see Section 35.5.3.2).

An explicit Enclave Access (a linear memory access which is either from within an enclave into its ELRANGE, or an access by an Intel SGX instruction that is expected to be in the EPC) that overlaps with the APIC-access page causes a #PF exception (APIC page is expected to be outside of EPC).

Non-Enclave accesses made either by an Intel SGX instruction or by a logical processor inside an enclave to an address that without SGX would have caused redirection to the virtual-APIC page instead cause an APIC-access VM exit.

Other than implicit accesses made by Intel SGX instructions, guest-physical and physical accesses are not considered "enclave accesses"; consequently, such accesses result in undefined behavior if these accesses eventually reach EPC. This applies to any non-enclave physical accesses.

While a logical processor is executing inside an enclave, an attempt to execute an instruction outside of ELRANGE results in a #GP(0), even if the linear address would translate to a physical address that overlaps the APIC-access page.

39.5.4 Interactions with VT and SGX concurrency

In some cases, a VMM is required to handle conflicts between its own operation and a guest operation on EPC pages that are present in both guest and VMM address space. These conflict would otherwise cause the guest to experience an unexpected behavior (vs. running directly on the h/w). These conflict cases are:

- ETRACK/ETRACKC failure due to Entry Epoch Object Lock conflict or reference tracking check failure.
- EPC Page Resource conflict.

A new exit reason is defined for all those cases: SGX_CONFLICT (value 71). The VMCS exit qualification field details the specific case as follows:

Table 39-1. SGX Conflict Exit Qualification

Bits	Size (bits)	Name	Description
15:0	16	Code	Exit qualification code. The following values are defined: 0: TRACKING_RESOURCE_CONFLICT 1: TRACKING_REFERENCE_CONFLICT 2: EPC_PAGE_CONFLICT_EXCEPTION 3: EPC_PAGE_CONFLICT_ERROR Other: Reserved
31:16	16	Error	Error code. Applicable only if the exit qualification code is EPC_PAGE_CONFLICT_ERROR; contains the error code that would be returned in RAX if the instruction was executed on bare metal platform or if the ENABLE_EPC_VIRTUALIZATION_EXTENSIONS bit in the secondary processor control field is not set. In other cases this field is reserved as 0.
63:32	32	Reserved	Always 0.

This SGX_CONFLICT exiting behavior is controlled by a VM execution control called ENABLE_EPC_VIRTUALIZATION_EXTENSIONS (bit 29 of the secondary processor control field).

Details for various SGX_CONFLICT VMEXIT cases are provided in the following sections.

39.5.5 Virtual Child Tracking

SGX oversubscription support adds the ability to associate virtual children with each enclave using the ENCLV[EINCVIRTCHILD] and ENCLV[EDECVIRTCHILD] instructions. The VMM enables checking of the virtual child count by EREMOVE and EWB in guests with a new VM execution control called ENABLE_EPC_VIRTUALIZATION_EXTENSIONS.

When in VMX non-root operation and the ENABLE_EPC_VIRTUALIZATION_EXTENSIONS control enabled, the following instructions change their behavior:

- EWB and EREMOVE return the SGX_CHILD_PRESENT error code if any virtual or physical children are associated with the enclave.
- ERDINFO set STATUS.CHILDPRESENT if any virtual or physical children are associated with the enclave.

39.5.6 Handling EPCM Entry Lock Conflicts

When performing paging within a VMM, it is possible for a contention on the EPC page to happen in the following case:

- The VMM performs an ELDB/ELDU/ELDBC/ELDUC of an enclave page, and the guest attempts to perform some SGX instruction (e.g., EREMOVE) where the same SECS parent page is required.

A similar conflict may occur if the VMM uses EINCVIRTCHILD or EDECVIRTCHILD pointing to an SECS page. In all other cases where a SGX instruction executed by the VMM the applicable EPC page should not be mapped to the guest, thus no resource conflict occurs.

This conflicting situation can cause the guest's instruction to fail and cause guest instability. To help the VMM manage such conflicts, the SGX VMM paging extensions introduce a new VM-Exit that will be triggered whenever the guest encounters a resource conflict.

The exit reason is SGX_CONFLICT. The exit qualification field is used to distinguish the two kinds of resource conflicts:

- A value of EPCM_RESOURCE_CONFLICT_EXCEPTION (2) in the exit qualification code field indicates that a resource conflict occurred that would result in a #GP. In that case, the exit qualification error field is set to zero.
- A value of EPC_PAGE_CONFLICT_ERROR (3) in the exit qualification code field indicates that a resource conflict occurred that would result in an error code being return in RAX. In that case, the exit qualification error field is set to SGX_EPC_PAGE_CONFLICT.

The Guest Linear Address and Guest Physical Address fields are set to the guest linear and guest physical addresses respectively of the EPC page on which the conflict occurred. The VMM may determine which instruction induced the exit by reading RAX. The exit also populates the VM-exit instruction length field.

The VMM can determine whether the conflict may be due to its own operation, e.g., by setting a per-enclave busy indicator before executing ELD*, and clearing it afterwards. In that case, the VMM can handle an SGX Conflict (EPCM_PAGE_CONFLICT_*) exit by resuming guest execution at the same instruction, allowing the guest to re-execute the instruction. The VMM may also take steps to throttle its own paging thread to reduce contention with the guest.

If the VMM determines that the conflict is not due to its own operation, it may inject a #GP (in case of EPC_PAGE_CONFLICT_EXCEPTION), or emulate an error code as the guest instruction would return (in case of EPC_PAGE_CONFLICT_ERROR) by setting ZF and copying the error value provided in the exit qualification to guest RAX.

To gracefully handle resource contention on the VMM side, the VMM should use the new ELDBC and ELDUC instructions. These are similar to ELDB and ELDU respectively, except that on EPC resource contention they return an SGX_EPC_PAGE_CONFLICT error instead of issuing a #GP. In case of an error, the VMM can retry the instruction, possibly throttling the guest to assure progress.

When using EDECVIRTCHILD and EINCVIRTCHILD, the VMM should preferably point to the enclave child page, not to the SECS page, avoiding resource conflict on the SECS. If the VMM chooses to point to the SECS page, it should handle conflicts in the same way as handling the ELD* case.

39.5.7 Context Tracking

The ENCLAVECONTEXT field in the SECS is available for use by the VMM to track context information associated with that enclave, such as the GPA of the SECS in the context of the appropriate guest. This field is initialized by the successful execution of ECREATE and ELD of an SECS page. The value stored in the ENCLAVECONTEXT field will be the translation of the target page address produced by paging (GPA in VMMs that have EPTs turned on). VMMs may override this default value by calling the ENCLV[ESETCONTEXT] instruction, which allows the VMM to store an arbitrary 64-bit value in the ENCLAVECONTEXT field. The VMM may later access the ENCLAVECONTEXT field by calling ENCLS[ERDINFO] on any member page of the enclave, including the SECS.

For nested virtualization cases, the lowest level VMM can make SGX oversubscription instructions higher level guest VMMs. In that case the lower level VMM can simply inject #GP to higher level VMMs when attempting to execute these instructions.

However, if VMMs expose SGX oversubscription instructions to higher level VMMs, then VMMs have to use ENCLV[ESETCONTEXT] instruction to properly manage the ENCLAVECONTEXT field of SECS during paging operations. That may involve emulating ECREATE, ELD, ESETCONTEXT, and ERDINFO instructions apart from managing ENCLAVECONTEXT values.

39.6 INTEL® SGX INTERACTIONS WITH ARCHITECTURALLY-VISIBLE EVENTS

All architecturally visible events (exceptions, interrupts, SMI, NMI, INIT, VM exit) can be detected while inside an enclave and will cause an asynchronous enclave exit if they are not blocked. Additionally, INT3, and the SignalTXTMsg[SENTER] (i.e., GETSEC[SENTER]'s rendezvous event message) events also cause asynchronous enclave exits. Note that SignalTXTMsg[SEXIT] (i.e., GETSEC[SEXIT]'s teardown message) does not cause an AEX.

On an AEX, information about the event causing the AEX is stored in the SSA (see Section 37.4 for details of AEX). The information stored in the SSA only describes the first event that triggered the AEX. If parsing/delivery of the first event results in detection of further events (e.g., VM exit, double fault, etc.), then the event information in the SSA is not updated to reflect these subsequently detected events.

39.7 INTERACTIONS WITH THE PROCESSOR EXTENDED STATE AND MISCELLANEOUS STATE

39.7.1 Requirements and Architecture Overview

Processor extended states are the ISA features that are enabled by the settings of CR4.OSXSAVE and the XCR0 register. Processor extended states are normally saved/restored by software via XSAVE/XRSTOR instructions. Details of discovery of processor extended states and management of these states are described in Chapter 13 of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

Additionally, the following requirements apply to Intel SGX:

- On an AEX, the Intel SGX architecture must protect the processor extended state and miscellaneous state by saving them in the enclave's state-save area (SSA), and clear the secrets from the processor extended state that is used by an enclave.
- Intel SGX architecture must verify that the SSA frame size is large enough to contain all the processor extended states and miscellaneous state used by the enclave.
- Intel SGX architecture must ensure that enclaves can only use processor extended state that is enabled by system software in XCR0.
- Enclave software should be able to discover only those processor extended state and miscellaneous state for which such protection is enabled.
- The processor extended states that are enabled inside the enclave must be approved by the enclave developer:
  - — Certain processor extended state (e.g., Memory Protection Extensions, see Appendix E, "Intel® Memory Protection Extensions," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1)

modify the behavior of the legacy ISA software. If such features are enabled for enclaves that do not understand those features, then such a configuration could lead to a compromise of the enclave's security.

- The processor extended states that are enabled inside the enclave must form an integral part of the enclave's identity. This requirement has two implications:
  - Service providers may decide to assign different trust level to the same enclave depending on the ISA features the enclave is using.

To meet these requirements, the Intel SGX architecture defines a sub-field called X-Feature Request Mask (XFRM) in the ATTRIBUTES field of the SECS. On enclave creation (ENCLS[ECREATE] leaf function), the required SSA frame size is calculated by the processor from the list of enabled extended and miscellaneous states and verified against the actual SSA frame size defined by SECS.SSAFRAMESIZE.

On enclave entry, after verifying that XFRM is only enabling features that are already enabled in XCR0, the value in the XCR0 is saved internally by the processor, and is replaced by the XFRM. On enclave exit, the original value of XCR0 is restored. Consequently, while inside the enclave, the processor extended states enabled in XFRM are in enabled state, and those that are disabled in XFRM are in disabled state.

The entire ATTRIBUTES field, including the XFRM subfield is integral part of enclave's identity (i.e., its value is included in reports generated by ENCLU[EREPORT], and select bits from this field can be included in key-derivation for keys obtained via the ENCLU[EGETKEY] leaf function).

Enclave developers can create their enclave to work with certain features and fallback to another code path in case those features aren't available (e.g., optimize for AVX and fallback to SSE). For this purpose Intel SGX provides the following fields in SIGSTRUCT: ATTRIBUTES, ATTRIBUTESMASK, MISCSELECT, and MISCMASK. EINIT ensures that the final SECS.ATTRIBUTES and SECS.MISCSELECT comply with the enclave developer's requirements as follows:

SIGSTRUCT.ATTRIBUTES & SIGSTRUCT.ATTRIBUTEMASK = SECS.ATTRIBUTES & SIGSTRUCT.ATTRIBUTEMASK

SIGSTRUCT.MISCSELECT & SIGSTRUCT.MISCMASK = SECS.MISCSELECT & SIGSTRUCT.MISCMASK.

On an asynchronous enclave exit, the processor extended states enabled by XFRM are saved in the current SSA frame, and overwritten by synthetic state (see Section 37.3 for the definition of the synthetic state). When the interrupted enclave is resumed via the ENCLU[ERESUME] leaf function, the saved state for processor extended states enabled by XFRM is restored.

39.7.2 Relevant Fields in Various Data Structures

39.7.2.1 SECS.ATTRIBUTES.XFRM

The ATTRIBUTES field of the SECS data structure (see Section 35.7) contains a sub-field called XSAVE-Feature Request Mask (XFRM). Software populates this field at the time of enclave creation according to the features that are enabled by the operating system and approved by the enclave developer.

Intel SGX architecture guarantees that during enclave execution, the processor extended state configuration of the processor is identical to what is required by the XFRM sub-field. All the processor extended states enabled in XFRM are saved on AEX from the enclave and restored on ERESUME.

The XFRM sub-field has the same layout as XCR0, and has consistency requirements that are similar to those for XCR0. Specifically, the consistency requirements on XFRM values depend on the processor implementation and the set of features enabled in CR4.

Legal values for SECS.ATTRIBUTES.XFRM conform to these requirements:

- XFRM[1:0] must be set to 0x3.
- If the processor does not support XSAVE, or if the system software has not enabled XSAVE, then XFRM[63:2] must be zero.
- If the processor does support XSAVE, XFRM must contain a value that would be legal if loaded into XCR0.

The various consistency requirements are enforced at different times in the enclave's life cycle, and the exact enforcement mechanisms are elaborated in Section 39.7.3 through Section 39.7.6.

On processors not supporting XSAVE, software should initialize XFRM to 0x3. On processors supporting XSAVE, software should initialize XFRM to be a subset of XCR0 that would be present at the time of enclave execution.

Because bits 0 and 1 of XFRM must always be set, the use of Intel SGX requires that SSE be enabled (CR4.OSFXSR = 1).

39.7.2.2 SECS.SSAFRAMESIZE

The SSAFRAMESIZE field in the SECS data structure specifies the number of pages which software allocated[1] for each SSA frame, including both the GPRSGX area, MISC area, the XSAVE area (x87 and XMM states are stored in the latter area), and optionally padding between the MISC and XSAVE area. The GPRSGX area must hold all the general-purpose registers and additional Intel SGX specific information. The MISC area must hold the Miscellaneous state as specified by SECS.MISCSELECT, the XSAVE area holds the set of processor extended states specified by SECS.ATTRIBUTES.XFRM (see Section 35.9 for the layout of SSA and Section 39.7.3 for ECREATE's consistency checks). The SSA is always in non-compacted format.

If the processor does not support XSAVE, the XSAVE area will always be 576 bytes; a copy of XFRM (which will be set to 0x3) is saved at offset 512 on an AEX.

If the processor does support XSAVE, the length of the XSAVE area depends on SECS.ATTRIBUTES.XFRM. The length would be equal to what CPUID.(EAX=0DH, ECX= 0):EBX would return if XCR0 were set to XFRM. The following pseudo code illustrates how software can calculate this length using XFRM as the input parameter without modifying XCR0:

```
offset = 576;
size_last_x = 0;
For x=2 to 63
IF (XFRM[x] != 0) Then
   tmp_offset = CPUID.(EAX=0DH, ECX= x):EBX[31:0];
    IF (tmp_offset >= offset + size_last_x) Then
        offset = tmp_offset;
        size_last_x = CPUID.(EAX=0DH, ECX= x):EAX[31:0];
     FI;
FI;
EndFor
return (offset + size_last_x); (* compute_xsave_size(XFRM), see "ECREATE—Create an SECS page in the Enclave Page Cache"*)
```

Where the non-zero bits in XFRM are a subset of non-zero bit fields in XCR0.

The size of the MISC region depends on the setting of SECS.MISCSELECT and can be calculated using the layout information described in Section 35.9.2

39.7.2.3 XSAVE Area in SSA

The XSAVE area of an SSA frame begins at offset 0 of the frame.

39.7.2.4 MISC Area in SSA

The MISC area of an SSA frame is positioned immediately before the GPRSGX region.

39.7.2.5 SIGSTRUCT Fields

Intel SGX provides the flexibility for an enclave developer to choose the enclave's code path according to the features that are enabled on the platform (e.g., optimize for AVX and fallback to SSE). See Section 39.7.1 for details.

1. It is the responsibility of the enclave to actually allocate this memory.

SIGSTRUCT includes the following fields:

SIGSTRUCT.ATTRIBUTES, SIGSTRUCT.ATTRIBUTEMASK, SIGSTRUCT.MISCSELECT, SIGSTRUCT.MISCMASK.

39.7.2.6 REPORT.ATTRIBUTES.XFRM and REPORT.MISCSELECT

The processor extended states and miscellaneous states that are enabled inside the enclave form an integral part of the enclave's identity and are therefore included in the enclave's report, as provided by the ENCLU[EREPORT] leaf function. The REPORT structure includes the enclave's XFRM and MISCSELECT configurations.

39.7.2.7 KEYREQUEST

An enclave developer can specify which bits out of XFRM and MISCSELECT ENCLU[EGETKEY] should include in the derivation of the sealing key by specifying ATTRIBUTESMASK and MISCMASK in the KEYREQUEST structure.

39.7.3 Processor Extended States and ENCLS[ECREATE]

The ECREATE leaf function of the ENCLS instruction enforces a number of consistency checks described earlier. The execution of ENCLS[ECREATE] leaf function results in a #GP(0) in any of the following cases:

- SECS.ATTRIBUTES.XFRM[1:0] is not 3.
- The processor does not support XSAVE and any of the following is true:
  - SECS.ATTRIBUTES.XFRM[63:2] is not 0.
  - SECS.SSAFRAMESIZE is 0.
- The processor supports XSAVE and any of the following is true:
  - XSETBV would fault on an attempt to load XFRM into XCR0.
  - XFRM[63]=1.
  - The SSAFRAME is too small to hold required, enabled states (see Section 39.7.2.2).

39.7.4 Processor Extended States and ENCLU[EENTER]

39.7.4.1 Fault Checking

The EENTER leaf function of the ENCLU instruction enforces a number of consistency requirements described earlier. The execution of the ENCLU[EENTER] leaf function results in a #GP(0) in any of the following cases:

- If CR4.OSFXSR=0.
- If The processor supports XSAVE and either of the following is true:
  - CR4.OSXSAVE=0 and SECS.ATTRIBUTES.XFRM is not 3.
  - (SECS.ATTRIBUTES.XFRM & XCR0) != SECS.ATTRIBUTES.XFRM

39.7.4.2 State Loading

If ENCLU[EENTER] is successful, the current value of XCR0 is saved internally by the processor and replaced by SECS.ATTRIBUTES.XFRM.

39.7.5 Processor Extended States and AEX

39.7.5.1 State Saving

On an AEX, processor extended states are saved into the XSAVE area of the SSA frame in a compatible format with XSAVE that was executed with EDX:EAX = SECS.ATTRIBUTES.XFRM, with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if REX.W=1. The XSTATE_BV part of the XSAVE header is saved with 0 for every bit that is 0 in XFRM. Other bits may be saved as 0 if the state saved is initialized.

Note that enclave entry ensures that if CR4.OSXSAVE is set to 0, then SECS.ATTRIBUTES.XFRM is set to 3. It should also be noted that it is not possible to enter an enclave with FXSAVE disabled.

39.7.5.2 State Synthesis

After saving the extended state, the processor restores XCR0 to the value it held at the time of the most recent enclave entry.

The state of features corresponding to bits set in XFRM is synthesized. In general, these states are initialized. Details of state synthesis on AEX are documented in Section 37.3.1.

39.7.6 Processor Extended States and ENCLU[ERESUME]

39.7.6.1 Fault Checking

The ERESUME leaf function of the ENCLU instruction enforces a number of consistency requirements described earlier. Specifically, the ENCLU[ERESUME] leaf function results in a #GP(0) in any of the following cases:

- CR4.OSFXSR=0.
- The processor supports XSAVE and either of the following is true:
  - CR4.OSXSAVE=0 and SECS.ATTRIBUTES.XFRM is not 3.
  - (SECS.ATTRIBUTES.XFRM & XCR0) != SECS.ATTRIBUTES.XFRM.

A successful execution of ENCLU[ERESUME] loads state from the XSAVE area of the SSA frame in a fashion similar to that used by the XRSTOR instruction. Data in the XSAVE area that would cause the XRSTOR instruction to fault will cause the ENCLU[ERESUME] leaf function to fault. Examples include, but are not restricted to the following:

- A bit is set in the XSTATE_BV field and clear in XFRM.
- The required bytes in the header are not clear.
- Loading data would set a reserved bit in MXCSR.

Any of these conditions will cause ERESUME to fault, even if CR4.OSXSAVE=0.

39.7.6.2 State Loading

If ENCLU[ERESUME] is successful, the current value of XCR0 is saved internally by the processor and replaced by SECS.ATTRIBUTES.XFRM.

State is loaded from the XSAVE area of the SSA frame as if the XRSTOR instruction were executed with XCR0=XFRM, EDX:EAX = XFRM, with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if REX.W=1.

ENCLU[ERESUME] ensures that a subsequent execution of XSAVEOPT inside the enclave will operate properly (e.g., by marking all state as modified).

39.7.7 Processor Extended States and ENCLU[EEXIT]

The ENCLU[EEXIT] leaf function does not perform any X-feature specific consistency checks, nor performs any state synthesis. It is the responsibility of enclave software to clear any sensitive data from the registers before

executing EEXIT. However, successful execution of the ENCLU[EEXIT] leaf function restores XCR0 to the value it held at the time of the most recent enclave entry.

39.7.8 Processor Extended States and ENCLU[EREPORT]

The ENCLU[EREPORT] leaf function creates the MAC-protected REPORT structure that reports on the enclave's identity. ENCLU[EREPORT] includes in the report the values of SECS.ATTRIBUTES.XFRM and SECS.MISCSELECT.

39.7.9 Processor Extended States and ENCLU[EGETKEY]

The ENCLU[EGETKEY] leaf function returns a cryptographic key based on the information provided by the KEYREQUEST structure. Intel SGX provides the means for isolation between different operating conditions by allowing an enclave developer to select which bits out of XFRM and MISCSELECT need to be included in the derivation of the keys.

39.8 INTERACTIONS WITH SMM

39.8.1 Availability of Intel® SGX instructions in SMM

Enclave instructions are not available in SMM, and any attempt to execute ENCLS or ENCLU instructions inside SMM results in an invalid-opcode exception (#UD).

39.8.2 SMI while Inside an Enclave

If the logical processor executing inside an enclave receives an SMI, the logical processor exits the enclave asynchronously. The response to an SMI received while executing inside an enclave depends on whether the dual-monitor treatment is enabled. For detailed discussion of transfer to SMM, see Chapter 32, "System Management Mode," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is not enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM handler. In addition to saving the synthetic architectural state to the SMRAM State Save Map (SSM), the logical processor also sets the "Enclave Interruption" bit in the SMRAM SSM (bit position 1 in SMRAM field at offset 7EE0H).

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM monitor via SMM VM exit. The SMM VM exit sets the "Enclave Interruption" bit in the Exit Reason (see Table 39-2) and in the Guest Interruptibility State field (see Table 39-3) of the SMM VMCS.

39.8.3 SMRAM Synthetic State of AEX Triggered by SMI

All processor registers saved in the SMRAM have the same synthetic values listed in Section 37.3. Additional SMRAM fields that are treated specially on SMI are:

Table 39-2. SMRAM Synthetic States on Asynchronous Enclave Exit

Position	Field	Value	Writable
SMRAM Offset 07EE0H.Bit 1	ENCLAVE_INTERRUPTION	Set to 1 if exit occurred in enclave mode	No

39.9 INTERACTIONS OF INIT, SIPI, AND WAIT-FOR-SIPI WITH INTEL® SGX

INIT received inside an enclave, while the logical processor is not in VMX operation, causes the logical processor to exit the enclave asynchronously. After the AEX, the processor's architectural state is initialized to "Power-on" state (Table 9.1 in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A). If the logical processor is BSP, then it proceeds to execute the BIOS initialization code. If the logical processor is an AP, it enters wait-for-SIPI state.

INIT received inside an enclave, while the logical processor (LP) is in VMX root operation, follows regular Intel Architecture behavior and is blocked.

INIT received inside an enclave, while the logical processor is in VMX non-root operation, causes an AEX. Subsequent to the AEX, the INIT causes a VM exit with the Enclave Interruption bit in the exit-reason field in the VMCS.

A processor cannot be inside an enclave in the wait-for-SIPI state. Consequently, a SIPI received while inside an enclave is lost.

Intel SGX does not change the behavior of the processor in the wait-for-SIPI state.

The SGX-related processor states after INIT-SIPI-SIPI is as follows:

- EPC Settings: Unchanged
- EPCM: Unchanged
- CPUID.LEAF_12H.*: Unchanged
- ENCLAVE_MODE: 0 (LP exits enclave asynchronously)
- MEE state: Unchanged

Software should be aware that following INIT-SIPI-SIPI, the EPC might contain valid pages and should take appropriate measures such as initialize the EPC with the EREMOVE leaf function.

39.10 INTERACTIONS WITH DMA

DMA is not allowed to access any Processor Reserved Memory.

39.11 INTERACTIONS WITH TXT

39.11.1 Enclaves Created Prior to Execution of GETSEC

Enclaves which have been created before the GETSEC[SENTER] leaf function are available for execution after the successful completion of GETSEC[SENTER] and the corresponding SINIT ACM. Actions that a TXT Launched Environment performs in preparation to execute code in the Launched Environment, also applies to enclave code that would run after GETSEC[SENTER].

39.11.2 Interaction of GETSEC with Intel® SGX

All leaf functions of the GETSEC instruction are illegal inside an enclave, and results in an invalid-opcode exception (#UD).

Responding Logical Processors (RLP) which are executing inside an enclave at the time a GETSEC[SENTER] event occurs perform an AEX from the enclave and then enter the Wait-for-SIPI state.

RLP executing inside an enclave at the time of GETSEC[SEXIT], behave as defined for GETSEC[SEXIT]-that is, the RLPs pause during execution of SEXIT and resume after the completion of SEXIT.

The execution of a TXT launch does not affect Intel SGX configuration or security parameters.

39.11.3 Interactions with Authenticated Code Modules (ACMs)

Intel SGX only allows launching ACMs with an Intel SGX SVN that is at the same level or higher than the expected Intel SGX SVN. The expected Intel SGX SVN is specified by BIOS and locked down by the processor on the first successful execution of an Intel SGX instruction that doesn't return an error code. Intel SGX provides interfaces for system software to discover whether a non faulting Intel SGX instruction has been executed, and evaluate the suitability of the Intel SGX SVN value of any ACM that is expected to be launched by the OS or the VMM.

These interfaces are provided through a read-only MSR called the IA32_SGX_SVN_STATUS MSR (MSR address 500h). The IA32_SGX_SVN_STATUS MSR has the format shown in Table 39-3.

Table 39-3. Layout of the IA32_SGX_SVN_STATUS MSR

Bit Position	Name	ACM Module ID	Value
0	Lock	N.A.	▪ If 1, indicates that a non-faulting Intel SGX instruction has been executed, consequently, launching a properly signed ACM but with Intel SGX SVN value less than the BIOS specified Intel SGX SVN threshold would lead to an TXT shutdown. ▪ If 0, indicates that the processor will allow a properly signed ACM to launch irrespective of the Intel SGX SVN value of the ACM.
15:1	RSVD	N.A.	0
23:16	SGX_SVN_SINIT	SINIT ACM	▪ If CPUID.01H:ECX.SMX =1, this field reflects the expected threshold of Intel SGX SVN for the SINIT ACM. ▪ If CPUID.01H:ECX.SMX =0, this field is reserved (0).
63:24	RSVD	N.A.	0

OS/VMM that wishes to launch an architectural ACM such as SINIT is expected to read the IA32_SGX_SVN_STATUS MSR to determine whether the ACM can be launched or a new ACM is needed:

- If either the Intel SGX SVN of the ACM is greater than the value reported by IA32_SGX_SVN_STATUS, or the lock bit in the IA32_SGX_SVN_STATUS is not set, then the OS/VMM can safely launch the ACM.
- If the Intel SGX SVN value reported in the corresponding component of the IA32_SGX_SVN_STATUS is greater than the Intel SGX SVN value in the ACM's header, and if bit 0 of IA32_SGX_SVN_STATUS is 1, then the OS/VMM should not launch that version of the ACM. It should obtain an updated version of the ACM either from the BIOS or from an external resource.

However, OSVs/VMMs are strongly advised to update their version of the ACM any time they detect that the Intel SGX SVN of the ACM carried by the OS/VMM is lower than that reported by IA32_SGX_SVN_STATUS MSR, irrespective of the setting of the lock bit.

39.12 INTERACTIONS WITH CACHING OF LINEAR-ADDRESS TRANSLATIONS

Entering and exiting an enclave causes the logical processor to flush all the global linear-address context as well as the linear-address context associated with the current VPID and PCID. The MONITOR FSM is also cleared.

39.13 INTERACTIONS WITH INTEL® TRANSACTIONAL SYNCHRONIZATION EXTENSIONS (INTEL® TSX)

1. ENCLU or ENCLS instructions inside an HLE region will cause the flow to be aborted and restarted non-speculatively. ENCLU or ENCLS instructions inside an RTM region will cause the flow to be aborted and transfer control to the fallback handler.

2. If XBEGIN is executed inside an enclave, the processor does NOT check whether the address of the fallback handler is within the enclave.

3. If an RTM transaction is executing inside an enclave and there is an attempt to fetch an instruction outside the enclave, the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

4. If an RTM transaction is executing inside an enclave and there is a data access to an address within the enclave that denied due to EPCM content (e.g., to a page belonging to a different enclave), the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

5. If an RTM transaction executing inside an enclave aborts and the address of the fallback handler is outside the enclave, a #GP is delivered after the abort (EIP reported is that of the fallback handler).

39.13.1 HLE and RTM Debug

RTM debug will be suppressed on opt-out enclave entry. After opt-out entry, the logical processor will behave as if IA32_DEBUG_CTL[15]=0. Any #DB detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if either DR7[11] = 0 OR IA32_DEBUGCTL[15] = 0, any #DB or #BP detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if DR7[11] = 1 AND IA32_DEBUGCTL[15] = 1, any #DB or #BP detected inside an RTM translation will

- terminate speculative execution,
- set RIP to the address of the XBEGIN instruction, and
- be delivered as #DB (implying an Intel SGX AEX; any #BP is converted to #DB).
- DR6[16] will be cleared, indicating RTM debug (if the #DB causes a VM exit, DR6 is not modified but bit 16 of the pending debug exceptions field in the VMCS will be set).

39.14 INTEL® SGX INTERACTIONS WITH S STATES

Whenever an Intel SGX enabled processor enters S3-S5 state, enclaves are destroyed. This is due to the EPC being destroyed when power down occurs. It is the application runtime's responsibility to re-instantiate an enclave after a power transition for which the enclaves were destroyed.

39.15 INTEL® SGX INTERACTIONS WITH MACHINE CHECK ARCHITECTURE (MCA)

39.15.1 Interactions with MCA Events

All architecturally visible machine check events (#MC and CMCI) that are detected while inside an enclave cause an asynchronous enclave exit.

Any machine check exception (#MC) that occurs after Intel SGX is first enables causes Intel SGX to be disabled, (CPUID.SGX_Leaf.0:EAX[SGX1] == 0). It cannot be enabled until after the next reset.

39.15.2 Machine Check Enables (IA32_MCi_CTL)

All supported IA32_MCi_CTL bits for all the machine check banks must be set for Intel SGX to be available (CPUID.SGX_Leaf.0:EAX[SGX1] == 1). Any act of clearing bits from '1 to '0 in any of the IA32_MCi_CTL register may disable Intel SGX (set CPUID.SGX_Leaf.0:EAX[SGX1] to 0) until the next reset.

39.15.3 CR4.MCE

CR4.MCE can be set or cleared with no interactions with Intel SGX.

39.16 INTEL® SGX INTERACTIONS WITH PROTECTED MODE VIRTUAL INTERRUPTS

ENCLS[EENTER] modifies neither EFLAGS.VIP nor EFLAGS.VIF.

ENCLS[ERESUME] loads EFLAGS in a manner similar to that of an execution of IRET with CPL = 3. This means that ERESUME modifies neither EFLAGS.VIP nor EFLAGS.VIF regardless of the value of the EFLAGS image in the SSA frame.

AEX saves EFLAGS.VIP and EFLAGS.VIF unmodified into the EFLAGS image in the SSA frame. AEX modifies neither EFLAGS.VIP nor EFLAGS.VIF after saving EFLAGS.

If CR4.PVI = 1, CPL = 3, EFLAGS.VM = 0, IOPL < 3, EFLAGS.VIP = 1, and EFLAGS.VIF = 0, execution of STI causes a #GP fault. In this case, STI modifies neither EFLAGS.IF nor EFLAGS.VIF. This behavior applies without change within an enclave (where CPL is always 3). Note that, if IOPL = 3, STI always sets EFLAGS.IF without fault; CR4.PVI, EFLAGS.VIP, and EFLAGS.VIF are neither consulted nor modified in this case.

39.17 INTEL SGX INTERACTION WITH PROTECTION KEYS

SGX interactions with PKRU are as follows:

- CPUID.(EAX=12H, ECX=1):ECX.PKRU indicates whether SECS.ATTRIBUTES.XFRM.PKRU can be set. If SECS.ATTRIBUTES.XFRM.PKRU is set, then PKRU is saved and cleared as part of AEX and is restored as part of ERESUME. If CR4.PKE is set, an enclave can execute RDPKRU and WRKRU independent of whether SECS.ATTRIBUTES.XFRM.PKRU is set.

SGX interactions with domain permission checks are as follows:

1) If CR4.PKE is not set, then legacy and SGX permission checks are not effected.
2) If CR4.PKE is set, then domain permission checks are applied to all non-enclave access and enclave accesses to user pages in addition to legacy and SGX permission checks at a higher priority than SGX permission checks.
3) Implicit accesses aren't subject to domain permission checks.

CHAPTER 40
ENCLAVE CODE DEBUG AND PROFILING

Intel® SGX is architected to provide protection for production enclaves and permit enclave code developers to use an SGX-aware debugger to effectively debug a non-production enclave (debug enclave). Intel SGX also allows a non-SGX-aware debugger to debug non-enclave portions of the application without getting confused by enclave instructions.

40.1 CONFIGURATION AND CONTROLS

40.1.1 Debug Enclave vs. Production Enclave

The SECS of each enclave provides a bit, SECS.ATTRIBUTES.DEBUG, indicating whether the enclave is a debug enclave (if set) or a production enclave (if 0). If this bit is set, software outside the enclave can use EDBGRD/EDBGWR to access the EPC memory of the enclave. The value of DEBUG is not included in the measurement of the enclave and therefore doesn't require an alternate SIGSTRUCT to be generated to debug the enclave.

The ATTRIBUTES field in the SECS is reported in the enclave's attestation, and is included in the key derivation. Enclave secrets that were protected by the enclave using Intel SGX keys when it ran as a production enclave will not be accessible by the debug enclave. A debugger needs to be aware that special debug content might be required for a debug enclave to run in a meaningful way.

EPC memory belonging to a debug enclave can be accessed via the EDBGRD/EDBGWR leaf functions (see Section 38.4), while that belonging to a non-debug enclave cannot be accessed by these leaf functions.

40.1.2 Tool-Chain Opt-in

The TCS.FLAGS.DBGOPTIN bit controls interactions of certain debug and profiling features with enclaves, including code/data breakpoints, TF, RF, monitor trap flag, BTF, LBRs, BTM, BTS, Intel Processor Trace, and performance monitoring. This bit is forced to zero when EPC pages are added via EADD. A debugger can set this bit via EDBGWR to the TCS of a debug enclave.

An enclave entry through a TCS with the TCS.FLAGS.DBGOPTIN set to 0 is called an **opt-out entry**. Conversely, an enclave entry through a TCS with TCS.FLAGS.DBGOPTIN set to 1 is called an **opt-in entry**.

40.1.3 Debugging an Enclave That Uses Asynchronous Enclave Exit Notify

Whenever an opt-in enclave entry is used to perform enclave code debugging or profiling, the debugger or profiling tool may clear TCS.FLAGS.AEXNOTIFY to prevent AEX notifications from being delivered at each interrupt, breakpoint, trap, or other exception.

40.2 SINGLE STEP DEBUG

40.2.1 Single Stepping ENCLS Instruction Leafs

If the RFLAGS.TF bit is set at the beginning of ENCLS, then a single-step debug exception is pending as a trap-class exception on the instruction boundary immediately after the ENCLS instruction. Additionally, if the instruction is executed in VMX non-root operation and the "monitor trap flag" VM-execution control is 1, an MTF VM exit is pending on the instruction boundary immediately after the instruction if the instruction does not fault.

40.2.2 Single Stepping ENCLU Instruction Leafs

The interactions of the unprivileged Intel SGX instruction ENCLU are leaf dependent.

An enclave entry via EENTER/ERESUME leaf functions of the ENCLU, in certain cases, may mask the RFLAGS.TF bit, and mask the setting of the "monitor trap flag" VM-execution control. In such situations, an exit from the enclave, either via the EEXIT leaf function or via an AEX unmasks the RFLAGS.TF bit and the "monitor trap flag" VM-execution control. The details of this masking/unmasking and the pending of single stepping events across EENTER/ERESUME/EEXIT/AEX are covered in detail in Section 40.2.3.

If the EFLAGS.TF bit is set at the beginning of EREPORT or EGETKEY leafs, and if the EFLAGS.TF is not masked by the preceding enclave entry, then a single-step debug exception is pending on the instruction boundary immediately after the ENCLU instruction. Additionally, if the instruction is executed in VMX non-root operation and the "monitor trap flag" VM-execution control is 1, and if the monitor trap flag is not masked by the preceding enclave entry, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

If the instruction under consideration results in a fault, then the control flow goes to the fault handler, and no single-step debug exception is asserted. In such a situation, if the instruction is executed in VMX non-root operation and the "monitor trap flag" VM-execution control is 1, an MTF VM exit is pending after the delivery of the fault (or any nested exception). No MTF VM exit occurs if another VM exit occurs before reaching that boundary on which an MTF VM exit would be pending.

40.2.3 Single-Stepping Enclave Entry with Opt-out Entry

40.2.3.1 Single Stepping without AEX

Figure 40-1 shows the most common case for single-stepping after an opt-out entry.

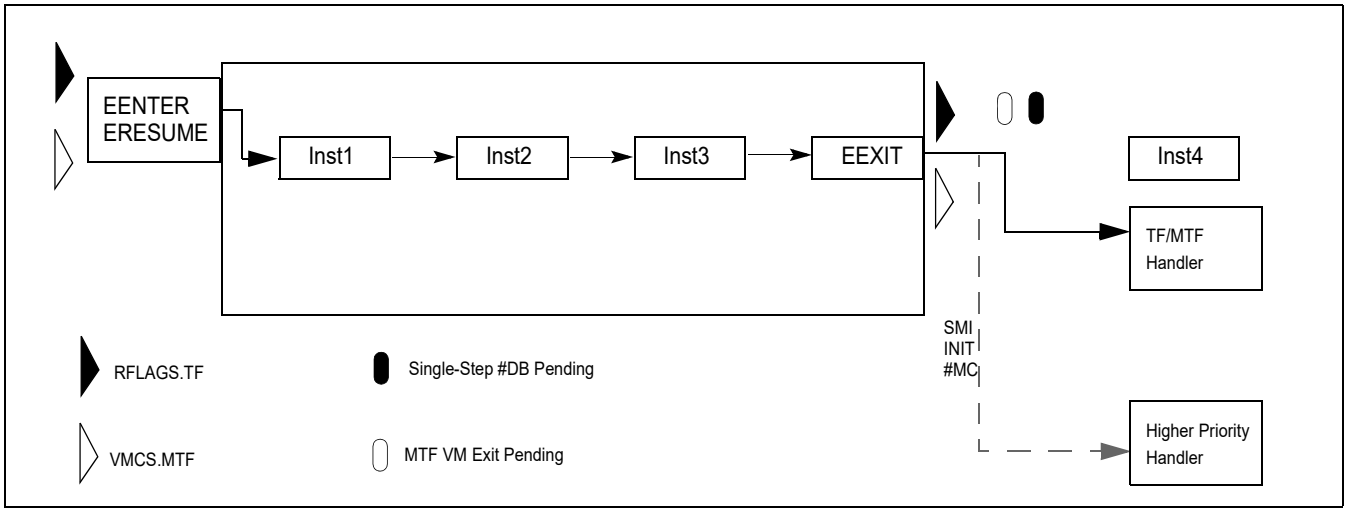

Figure 40-1. Single Stepping with Opt-out Entry - No AEX

In this scenario, if the RFLAGS.TF bit is set at the time of the enclave entry, then a single step debug exception is pending on the instruction boundary after EEXIT. Additionally, if the enclave is executing in VMX non-root operation and the "monitor trap flag" VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after EEXIT.

The value of the RFLAGS.TF bit at the end of EEXIT is the same as the value of RFLAGS.TF at the time of the enclave entry.

40.2.3.2 Single Step Preempted by AEX Due to Non-SMI Event

Figure 40-2 shows the interaction of single stepping with AEX due to a non-SMI event after an opt-out entry.

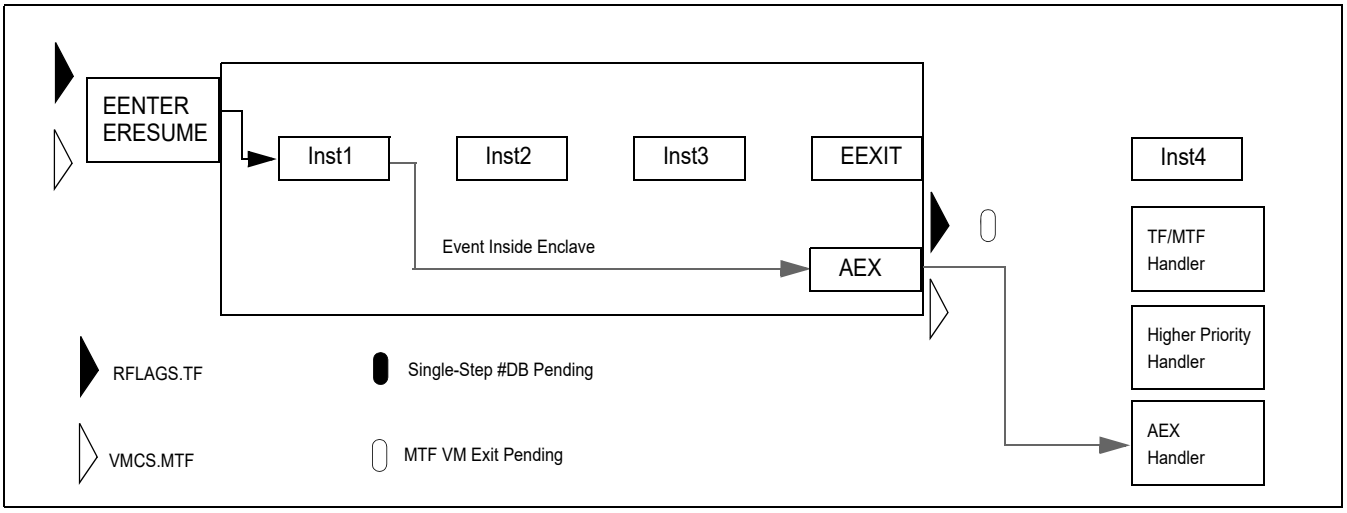

Figure 40-2. Single Stepping with Opt-out Entry -AEX Due to Non-SMI Event Before Single-Step Boundary

In this scenario, if the enclave is executing in VMX non-root operation and the "monitor trap flag" VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after the AEX. No MTF VM exit occurs if another VM exit happens before reaching that instruction boundary.

The value of the RFLAGS.TF bit at the end of AEX is the same as the value of RFLAGS.TF at the time of the enclave entry.

40.2.4 RFLAGS.TF Treatment on AEX

The value of EFLAGS.TF at the end of AEX from an opt-out enclave is same as the value of EFLAGS.TF at the time of the enclave entry. The value of EFLAGS.TF at the end of AEX from an opt-in enclave is unmodified. The EFLAGS.TF saved in GPR portion of the SSA on an AEX is 0. For more detail see EENTER and ERESUME in Chapter 5.

40.2.5 Restriction on Setting of TF after an Opt-Out Entry

Enclave entered through an opt-out entry is not allowed to set EFLAGS.TF. The POPF instruction forces RFLAGS.TF to 0 if the enclave was entered through opt-out entry.

40.2.6 Trampoline Code Considerations

Any AEX from the enclave which results in the RFLAGS.TF =1 on the reporting stack will result in a single-step #DB after the first instruction of the trampoline code if the trampoline is entered using the IRET instruction.

40.3 CODE AND DATA BREAKPOINTS

40.3.1 Breakpoint Suppression

Following an opt-out entry:

- Instruction breakpoints are suppressed during execution in an enclave.
- Data breakpoints are not triggered on accesses to the address range defined by ELRANGE.
- Data breakpoints are triggered on accesses to addresses outside the ELRANGE

Following an opt-in entry instruction and data breakpoints are not suppressed.

The processor does not report any matches on debug breakpoints that are suppressed on enclave entry. However, the processor does not clear any bits in DR6 that were already set at the time of the enclave entry.

40.3.2 Reporting of Instruction Breakpoint on Next Instruction on a Debug Trap

A debug exception caused by the single-step execution mode or when a data breakpoint condition was met causes the processor to perform an AEX. Following such an AEX, the processor reports in the debug status register (DR6) matches of the new instruction pointer (the AEP address) in a breakpoint address register setup to detect instruction execution.

40.3.3 RF Treatment on AEX

RF flag value saved in SSA is the same as what would have been pushed on stack if the exception or event causing the AEX occurred when executing outside an enclave (see Section 18.3.1.1). Following an AEX, the RF flag is 0 in the synthetic state.

40.3.4 Breakpoint Matching in Intel® SGX Instruction Flows

Implicit accesses made by Intel SGX instructions to EPC regions do not trigger data breakpoints. Explicit accesses made by ENCLS[ECREATE], ENCLS[EADD], ENCLS[EEXTEND], ENCLS[EINIT], ENCLS[EREMOVE], ENCLS[ETRACK], ENCLS[EBLOCK], ENCLS[EPA], ENCLS[EWB], ENCLS[ELD], ENCLS[EDBGRD], ENCLS[EDBGWR], ENCLU[EENTER], and ENCLU[ERESUME] to the EPC operands do not trigger data breakpoints.

Explicit accesses made by the Intel SGX instructions (ENCLU[EGETKEY] and ENCLU[EREPORT]) executed by an enclave following an opt-in entry, trigger data breakpoints on accesses to their EPC operands. All Intel SGX instructions trigger data breakpoints on accesses to their non-EPC operands.

40.4 CONSIDERATION OF THE INT1 AND INT3 INSTRUCTIONS

This section considers the operation of the INT1 and INT3 instructions when executed inside an enclave. These are the instructions with opcodes F1 and CC, respectively, and not INT *n* (with opcode CD) with value 1 or 3 for n.

40.4.1 Behavior of INT1 and INT3 Inside an Enclave

An execution of either INT1 or INT3 inside an enclave results in a fault-class exception. Following an opt-out entry, execution of either instruction results in an invalid-opcode exception (#UD). Following opt-in entry, INT1 results in a debug exception (#DB) and INT3 delivers a breakpoint exception (#BP). The normal requirement for INT3 (that the CPL not be greater than the DPL of descriptor 3 in the IDT) is not enforced.

Because execution of INT1 or INT3 inside an enclave results in a fault, the RIP saved in the SSA on AEX references the INT1 or INT3 instruction (and not the following instruction). The RIP value saved on the stack (or in the TSS or VMCS) is that of the AEP.

If execution of INT1 or INT3 inside an enclave causes a VM exit, the event type in the VM-exit interruption information field indicates a hardware exception (type 3),[1] and the VM-exit instruction length field is saved as zero.

40.4.2 Debugger Considerations

A debugger using INT3 inside an enclave should account for the modified behavior described in Section 40.4.1. Because INT3 is fault-like inside an enclave, the RIP saved in the SSA on AEX is that of the INT3 instruction. Conse-

1. INT1 would normally indicate a privileged software exception (type 5), and INT3 would normally indicate a software exception (type 6).

quently, the debugger must not decrement SSA.RIP for #BP coming from an enclave to re-execute the instruction at the RIP of the INT3 instruction on a subsequent enclave entry.

40.4.3 VMM Considerations

As described in Section 40.4.1, execution of INT3 inside an enclave delivers #BP with "interruption type" of 3. A VMM that re-injects #BP into the guest should establish the VM-entry interruption information field using data saved into the appropriate VMCS fields by the VM exit incident to the #BP (as recommended in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C).

VMMs that create the VM-entry interruption information based solely on the exception vector should take care to use event type 3 (instead of 6) when they detect a VM exit incident to enclave mode that is due to an exception with vector 3.

40.5 BRANCH TRACING

40.5.1 BTF Treatment

When software enables single-stepping on branches then:

- Following an opt-in entry using EENTER the processor generates a single step debug exception.
- Following an EEXIT the processor generates a single-step debug exception

Enclave entry using ERESUME (opt-in or opt-out) and an AEX from the enclave do not cause generation of the single-step debug exception.

40.5.2 LBR Treatment

40.5.2.1 LBR Stack on Opt-in Entry

Following an opt-in entry into an enclave, last branch recording facilities if enabled continued to store branch records in the LBR stack MSRs as follows:

- On enclave entry using EENTER/ERESUME, the processor push the address of EENTER/ERESUME instruction into MSR_LASTBRANCH_n_FROM_IP, and the destination address of the EENTER/ERESUME into MSR_LASTBRANCH_n_TO_IP.
- On EEXIT, the processor pushes the address of EEXIT instruction into MSR_LASTBRANCH_n_FROM_IP, and the address of EEXIT destination into MSR_LASTBRANCH_n_TO_IP.
- On AEX, the processor pushes RIP saved in the SSA into MSR_LASTBRANCH_n_FROM_IP, and the address of AEP into MSR_LASTBRANCH_n_TO_IP.
- For every branch inside the enclave, a branch record is pushed on the LBR stack.

Figure 40-3 shows an example of LBR stack manipulation after an opt-in entry. Every arrow in this picture indicates a branch record pushed on the LBR stack. The "From IP" of the branch record contains the linear address of the instruction located at the start of the arrow, while the "To IP" of the branch record contains the linear address of the instruction at the end of the arrow.

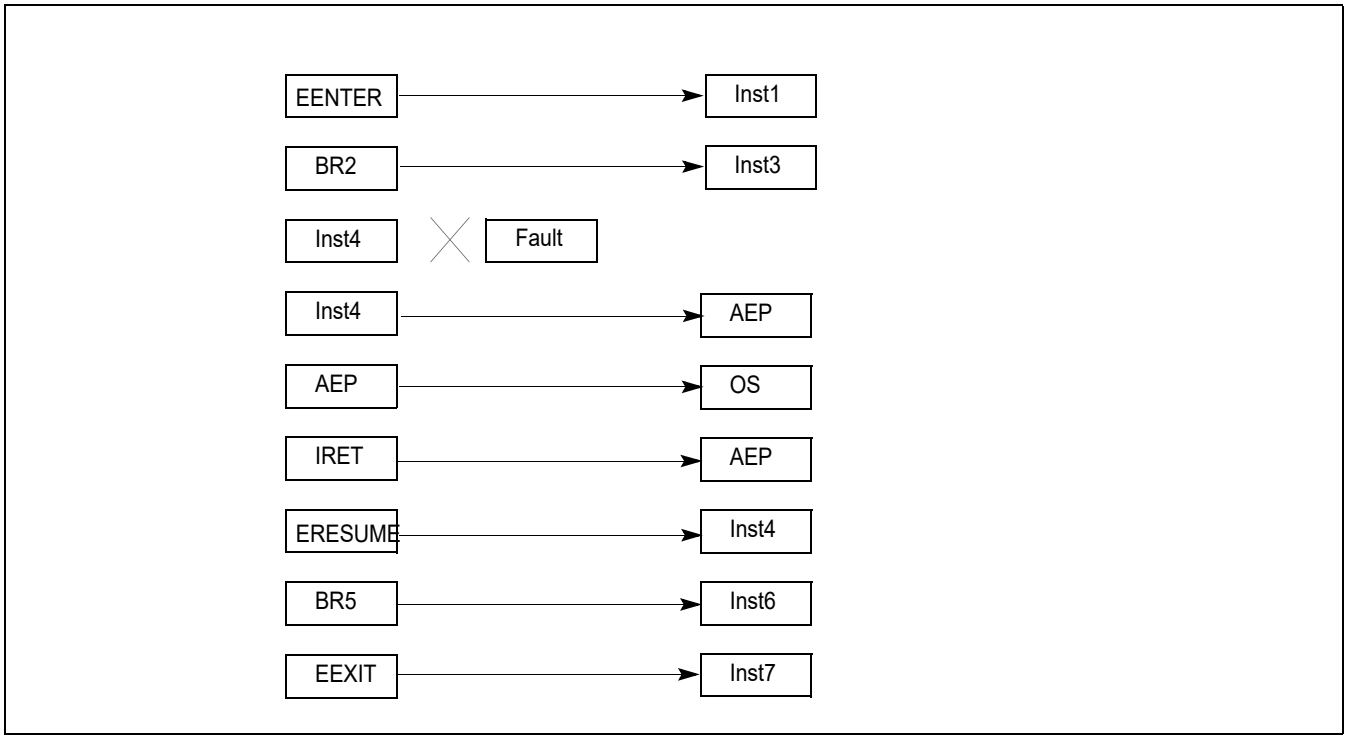

Figure 40-3. LBR Stack Interaction with Opt-in Entry

40.5.2.2 LBR Stack on Opt-out Entry

An opt-out entry into an enclave suppresses last branch recording facilities, and enclave exit after an opt-out entry un-suppresses last branch recording facilities.

Opt-out entry into an enclave does not push any record on LBR stack.

If last branch recording facilities were enabled at the time of enclave entry, then EEXIT following such an enclave entry pushes one record on LBR stack. The MSR_LASTBRANCH_n_FROM_IP of such record holds the linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while the MSR_LASTBRANCH_n_TO_IP of such record holds linear address of the destination of EEXIT.

Additionally, if last branch recording facilities were enabled at the time of enclave entry, then an AEX after such an entry pushes one record on LBR stack, before pushing record for the event causing the AEX if the event pushes a record on LBR stack. The MSR_LASTBRANCH_n_FROM_IP of the new record holds linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while MSR_LASTBRANCH_n_TO_IP of the new record holds linear address of the AEP. If the event causing AEX pushes a record on LBR stack, then the MSR_LASTBRANCH_n_FROM_IP for that record holds linear address of the AEP.

Figure 40-4 shows an example of LBR stack manipulation after an opt-out entry. Every arrow in this picture indicates a branch record pushed on the LBR stack. The “From IP” of the branch record contains the linear address of the instruction located at the start of the arrow, while the “To IP” of the branch record contains the linear address of the instruction at the end of the arrow.

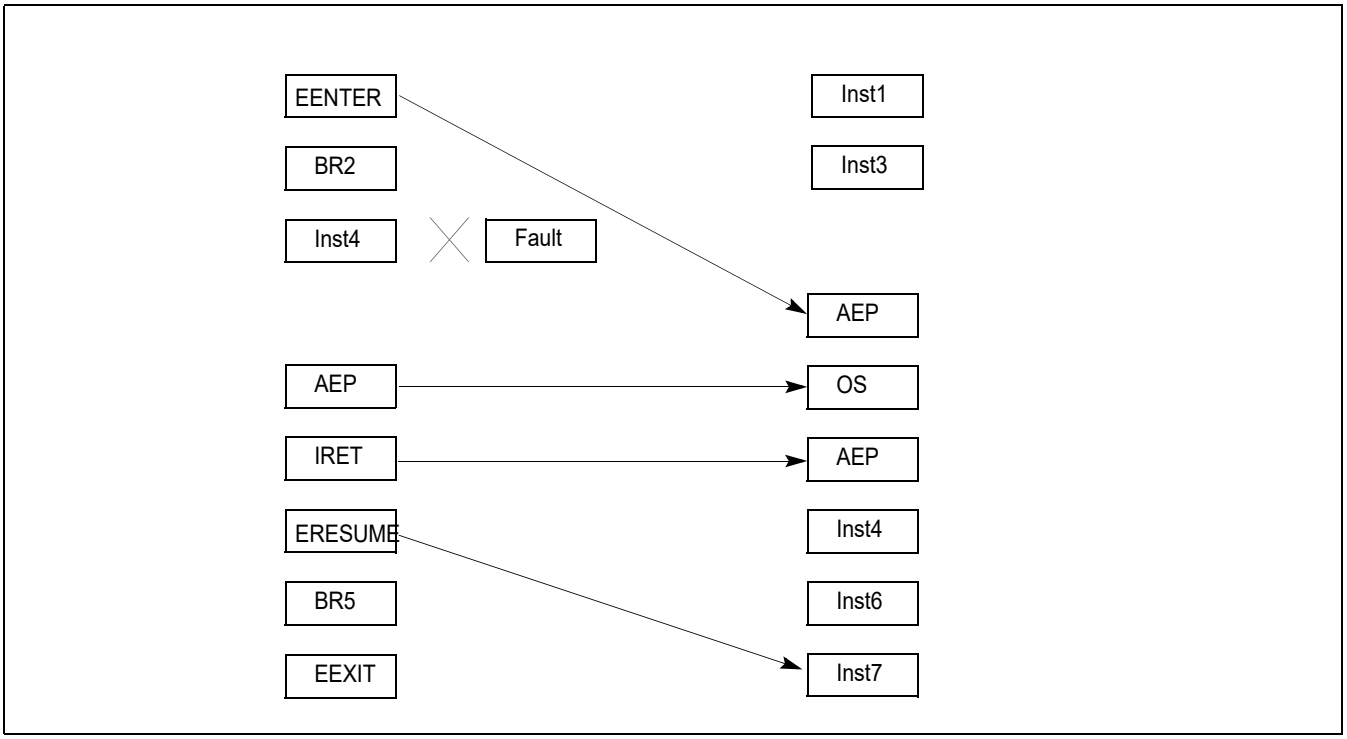

Figure 40-4. LBR Stack Interaction with Opt-out Entry

40.5.2.3 Mispredict Bit, Record Type, and Filtering

All branch records resulting from Intel SGX instructions/AEXs are reported as predicted branches, and consequently, bit 63 of MSR_LASTBRANCH_n_FROM_IP for such records is set. Branch records due to these Intel SGX operations are always non-HLE/non-RTM records.

EENTER, ERESUME, EEXIT, and AEX are considered to be far branches. Consequently, bit 8 in MSR_LBR_SELECT controls filtering of the new records introduced by Intel SGX.

40.6 INTERACTION WITH PERFORMANCE MONITORING

40.6.1 IA32_PERF_GLOBAL_STATUS Enhancement

On processors supporting Intel SGX, the IA32_PERF_GLOBAL_STATUS MSR provides a bit indicator, known as "Anti Side-channel Interference" (ASCI) at bit position 60. If this bit is 0, the performance monitoring data in various performance monitoring counters are accumulated normally as defined by relevant architectural/microarchitectural conditions. If the ASCI bit is set, the contents in various performance monitoring counters can be affected by the direct or indirect consequence of Intel SGX protection of enclave code executing in the processor.

40.6.2 Performance Monitoring with Opt-in Entry

An opt-in enclave entry allow performance monitoring logic to observe the contribution of enclave code executing in the processor. Thus the contents of performance monitoring counters does not distinguish between contribution originating from enclave code or otherwise. All counters, events, precise events, etc. continue to work as defined in the IA32/Intel 64 Software Developer Manual. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is not set.

40.6.3 Performance Monitoring with Opt-out Entry

In general, performance monitoring activities are suppressed when entering an opt-out enclave. This applies to all thread-specific, configured performance monitoring, except for the cycle-counting fixed counter, IA32_FIXED_CTR1 and IA32_FIXED_CTR2. Upon entering an opt-out enclave, IA32_FIXED_CTR0, IA32_PMCx will stop accumulating counts. Additionally, if PEBS is configured to capture PEBS record for this thread, PEBS record generation will also be suppressed. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is set.

Performance monitoring on the sibling thread may also be affected. Any one of IA32_FIXED_CTRx or IA32_PMCx on the sibling thread configured to monitor thread-specific eventing logic with AnyThread =1 is demoted to count only MyThread while an opt-out enclave is executing on the other thread.

40.6.4 Enclave Exit and Performance Monitoring

When a logical processor exits an enclave, either via ENCLU[EEXIT] or via AEX, all performance monitoring activity (including PEBS) on that logical processor that was suppressed is unsuppressed.

Any counters that were demoted from AnyThread to MyThread on the sibling thread are promoted back to AnyThread.

40.6.5 PEBS Record Generation on Intel® SGX Instructions

All leaf functions of the ENCLS instruction report "Eventing RIP" of the ENCLS instruction if a PEBS record is generated at the end of the instruction execution. Additionally, the EGETKEY and EREPORT leaf functions of the ENCLU instruction report "Eventing RIP" of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution.

If the EENTER and ERESUME leaf functions are performing an opt-in entry report "Eventing RIP" of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution. On the other hand, if these leaf functions are performing an opt-out entry, then these leaf functions result in PEBS being suppressed, and no PEBS record is generated at the end of these instructions.

A PEBS record is generated if there is a PEBS event pending at the end of EEXIT (due to a counter overflowing during enclave execution or during EEXIT execution). This PEBS record contains the architectural state of the logical processor at the end of EEXIT. If the enclave was entered via an opt-in entry, then this record reports the "Eventing RIP" as the linear address of the ENCLU[EEXIT] instruction. If the enclave was entered via an opt-out entry, then the record reports the "Eventing RIP" as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A PEBS record is generated after the AEX if there is a PEBS event pending at the end of AEX (due to a counter overflowing during enclave execution or during AEX execution). This PEBS record contains the synthetic state of the logical processor that is established at the end of AEX. For opt-in entry, this record has the EVENTING_RIP set to the RIP saved in the SSA. For opt-out entry, the record has the EVENTING_RIP set to the linear address of EENTER/ERESUME used for the last enclave entry.

If the enclave was entered via an opt-in entry, then this record reports the "Eventing RIP" as the linear address in the SSA of the enclave (a.k.a., the "Eventing LIP" inside the enclave). If the enclave was entered via an opt-out entry, then the record reports the "Eventing RIP" as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A second PEBS event may be pended during the Enclave Exiting Event (EEE). If the PEBS event is taken at the end of delivery of the EEE then the "Eventing RIP" in this second PEBS record is the linear address of the AEP.

40.6.6 Exception-Handling on PEBS/BTS Loads/Stores after AEX

As noted in Section 18.4.9.2, recording in the BTS buffer or in the PEBS buffer may not operate properly if accesses to any of the DS save area sections cause page faults or VM exits. Such page faults or VM exits, if they occur, are delivered immediately to the OS or VMM, and generation of a BTS or PEBS record is skipped and may leave the buffers in a state where they have a partial BTS or PEBS records.

However, any events that are detected during PEBS/BTS record generation at the end of AEX and before delivering the Enclave Exiting Event (EEE) cannot be reported immediately to the OS/VMM, as an event window is not open at

the end of AEX. Consequently, fault-like events such as page faults, EPT faults, EPT mis-configuration, and accesses to APIC-access page detected on stores to the PEBS/BTS buffer are not reported, and generation of the PEBS and/or BTS record at the end of AEX is aborted (this may leave the buffers in a state where they have partial PEBS or BTS records). Trap-like events detected on stores to the PEBS/BTS buffer (such as debug traps) are pended until the next instruction boundary, where they are handled according to the architecturally defined priority. The processor continues the handling of the Enclave Exiting Event (SMI, NMI, interrupt, exception delivery, VM exit, etc.) after aborting the PEBS/BTS record generation.

40.6.6.1 Other Interactions with Performance Monitoring

For opt-in entry, EENTER, ERESUME, EEXIT, and AEX are all treated as predicted far branches, and any counters that are counting such branches are incremented by 1 as a part of retirement of these instructions. Retirement of these instructions is also counted in any counters configured to count instructions retired.

For opt-out entry, execution inside an enclave is treated as a single predicted branch, and all branch-counting performance monitoring counters are incremented accordingly. Additionally, such execution is also counted as a single instruction, and all performance monitoring counters counting instructions are incremented accordingly.

Enclave entry does not affect any performance monitoring counters shared between cores.

APPENDIX A
VMX CAPABILITY REPORTING FACILITY

The ability of a processor to support VMX operation and related instructions is indicated by CPUID.1:ECX.VMX[bit 5] = 1. A value 1 in this bit indicates support for VMX features.

Support for specific features detailed in Chapter 27 and other VMX chapters is determined by reading values from a set of capability MSRs. These MSRs are indexed starting at MSR address 480H. VMX capability MSRs are read-only; an attempt to write them (with WRMSR) produces a general-protection exception (#GP(0)). They do not exist on processors that do not support VMX operation; an attempt to read them (with RDMSR) on such processors produces a general-protection exception (#GP(0)).

A.1 BASIC VMX INFORMATION

The IA32_VMX_BASIC MSR (index 480H) consists of the following fields:

- Bits 30:0 contain the 31-bit VMCS revision identifier used by the processor. Processors that use the same VMCS revision identifier use the same size for VMCS regions (see subsequent item on bits 44:32).[1]
- Bit 31 is always 0.
- Bits 44:32 report the number of bytes that software should allocate for the VMXON region and any VMCS region. It is a value greater than 0 and at most 4096 (bit 44 is set if and only if bits 43:32 are clear).
- Bit 48 indicates the width of the physical addresses that may be used for the VMXON region, each VMCS, and data structures referenced by pointers in a VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions). If the bit is 0, these addresses are limited to the processor's physical-address width.[2] If the bit is 1, these addresses are limited to 32 bits. This bit is always 0 for processors that support Intel 64 architecture.
- If bit 49 is read as 1, the logical processor supports the dual-monitor treatment of system-management interrupts and system-management mode. See Section 32.15 for details of this treatment.
- Bits 53:50 report the memory type that should be used for the VMCS, for data structures referenced by pointers in the VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions), and for the MSEG header. If software needs to access these data structures (e.g., to modify the contents of the MSR bitmaps), it can configure the paging structures to map them into the linear-address space. If it does so, it should establish mappings that use the memory type reported bits 53:50 in this MSR.[3]

  As of this writing, all processors that support VMX operation indicate the write-back type. The values used are given in Table A-1.

Table A-1. Memory Types Recommended for VMCS and Related Data Structures

Value(s)	Field
0	Uncacheable (UC)
1–5	Not used
6	Write Back (WB)
7–15	Not used

1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field in bits 31:0 of this MSR. For all processors produced prior to this change, bit 31 of this MSR was read as 0.
2. On processors that support Intel 64 architecture, the pointer must not set bits beyond the processor's physical address width.
3. Alternatively, software may map any of these regions or structures with the UC memory type. (This may be necessary for the MSEG header.) Doing so is discouraged unless necessary as it will cause the performance of software accesses to those structures to suffer.

- If bit 54 is read as 1, the processor reports information in the VM-exit instruction-information field on VM exits due to execution of the INS and OUTS instructions (see Section 28.2.5). This reporting is done only if this bit is read as 1.
- Bit 55 is read as 1 if any VMX controls that default to 1 may be cleared to 0. See Appendix A.2 for details. It also reports support for the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROC-BASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3.1, Appendix A.3.2, Appendix A.4, and Appendix A.5 for details.
- If bit 56 is read as 1, software can use VM entry to deliver a hardware exception with or without an error code, regardless of vector (see Section 27.2.1.3).
- The values of bits 47:45 and bits 63:57 are reserved and are read as 0.

A.2 RESERVED CONTROLS AND DEFAULT SETTINGS

As noted in Chapter 27, "VM Entries," certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its **default setting**. Software can discover the default setting of a reserved control by consulting the appropriate VMX capability MSR (see Appendix A.3 through Appendix A.5).

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control's new functionality should set the control to its default setting. For that reason, it is useful for software to know the default settings of the reserved controls.

Default settings partition the various controls into the following classes:

- **Always-flexible**. These have never been reserved.
- **Default0**. These are (or have been) reserved with a default setting of 0.
- **Default1**. They are (or have been) reserved with a default setting of 1.

As noted in Appendix A.1, a logical processor uses bit 55 of the IA32_VMX_BASIC MSR to indicate whether any of the default1 controls may be 0:

- If bit 55 of the IA32_VMX_BASIC MSR is read as 0, all the default1 controls are reserved and must be 1. VM entry will fail if any of these controls are 0 (see Section 27.2.1).
- If bit 55 of the IA32_VMX_BASIC MSR is read as 1, not all the default1 controls are reserved, and some (but not necessarily all) may be 0. The CPU supports four (4) new VMX capability MSRs: IA32_VMX_TRUE_PIN-BASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_EN-TRY_CTLS. See Appendix A.3 through Appendix A.5 for details. (These MSRs are not supported if bit 55 of the IA32_VMX_BASIC MSR is read as 0.)

A.3 VM-EXECUTION CONTROLS

There are separate capability MSRs for the pin-based VM-execution controls, the primary processor-based VM-execution controls, the secondary processor-based VM-execution controls, and the tertiary processor-based VM-execution controls. These are described in Appendix A.3.1, Appendix A.3.2, Appendix A.3.3, and Appendix A.3.4, respectively.

A.3.1 Pin-Based VM-Execution Controls

The IA32_VMX_PINBASED_CTLS MSR (index 481H) reports on the allowed settings of **most** of the pin-based VM-execution controls (see Section 25.6.1):

- Bits 31:0 indicate the **allowed 0-settings** of these controls. VM entry allows control X (bit X of the pin-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the pin-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 2, and 4; the corresponding bits of the IA32_VMX_PINBASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

— If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any pin-based VM-execution control in the default1 class is 0.

— If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (see below) reports which of the pin-based VM-execution controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the **allowed 1-settings** of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (index 48DH) reports on the allowed settings of **all** of the pin-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the pin-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_PINBASED_CTLS MSR. (The IA32_VMX_TRUE_PIN-BASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PINBASED_CTLS MSR. Assuming that software knows that the default1 class of pin-based VM-execution controls contains bits 1, 2, and 4, there is no need for software to consult the IA32_VMX_PINBASED_CTLS MSR.

A.3.2 Primary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS MSR (index 482H) reports on the allowed settings of **most** of the primary processor-based VM-execution controls (see Section 25.6.2):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary processor-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

  Exceptions are made for the primary processor-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 4–6, 8, 13–16, and 26; the corresponding bits of the IA32_VMX_PROC-BASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any of the primary processor-based VM-execution controls in the default1 class is 0.

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLS MSR (see below) reports which of the primary processor-based VM-execution controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLS MSR (index 48EH) reports on the allowed settings of **all** of the primary processor-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary processor-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_PROCBASED_CTLS MSR. (The IA32_VMX_TRUE_PROCBASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_TRUE_PROCBASED_CTLS MSR. Assuming that software knows that the default1 class of primary processor-based VM-execution controls contains bits 1, 4–6, 8, 13–16, and 26, there is no need for software to consult the IA32_VMX_PROC-BASED_CTLS MSR.

A.3.3 Secondary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS2 MSR (index 48BH) reports on the allowed settings of the secondary processor-based VM-execution controls (see Section 25.6.2). The following items provide details, including enforcement by VM entry:

- Bits 31:0 indicate the allowed 0-settings of these controls. These bits are always 0. This fact indicates that VM entry allows each bit of the secondary processor-based VM-execution controls to be 0 (reserved bits must be 0)
- Bits 63:32 indicate the allowed 1-settings of these controls; the 1-setting is not allowed for any reserved bit. VM entry allows control X (bit X of the secondary processor-based VM-execution controls) to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X and the "activate secondary controls" primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLS2 MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1).

A.3.4 Tertiary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS3 MSR (index 492H) reports on the allowed 1-settings of the tertiary processor-based VM-execution controls (see Section 25.6.2); the 1-setting is not allowed for any reserved bit.

VM entry allows control X (bit X of the tertiary processor-based VM-execution controls) to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if control X and the "activate tertiary controls" primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLS3 MSR exists only on processors that support the 1-setting of the "activate tertiary controls" VM-execution control (only if bit 49 of the IA32_VMX_PROCBASED_CTLS MSR is 1).

Notice that the organization of this MSR differs from that of IA32_VMX_PROCBASED_CTLS2 (Appendix A.3.3). This is because there are 64 tertiary processor-based VM-execution controls, while there were only 32 secondary processor-based VM-execution controls.

A.4 VM-EXIT CONTROLS

There are separate capability MSRs for the primary VM-exit controls and the secondary VM-exit controls. These are described in Appendix A.4.1 and Appendix A.4.2, respectively.

A.4.1 Primary VM-Exit Controls

The IA32_VMX_EXIT_CTLS MSR (index 483H) reports on the allowed settings of **most** of the primary VM-exit controls (see Section 25.7.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary VM-exit controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the primary VM-exit controls in the default1 class (see Appendix A.2). These are bits 0–8, 10, 11, 13, 14, 16, and 17; the corresponding bits of the IA32_VMX_EXIT_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

— If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any primary VM-exit control in the default1 class is 0.

— If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (see below) reports which of the primary VM-exit controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (index 48FH) reports on the allowed settings of **all** of the primary VM-exit controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary VM-exit controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary VM-exit controls is contained in the IA32_VMX_EXIT_CTLS MSR. (The IA32_VMX_TRUE_EXIT_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the primary VM-exit controls is contained in the IA32_VMX_TRUE_EXIT_CTLS MSR. Assuming that software knows that the default1 class of primary VM-exit controls contains bits 0–8, 10, 11, 13, 14, 16, and 17, there is no need for software to consult the IA32_VMX_EXIT_CTLS MSR.

A.4.2 Secondary VM-Exit Controls

The IA32_VMX_EXIT_CTLS2 MSR (index 493H) reports on the allowed 1-settings of the secondary VM-exit controls (see Section 25.7.1); the 1-setting is not allowed for any reserved bit.

VM entry allows control X (bit X of the secondary VM-exit controls) to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if control X and the "activate secondary controls" primary VM-exit control are both 1.

The IA32_VMX_EXIT_CTLS2 MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-exit control (only if bit 63 of the IA32_VMX_EXIT_CTLS MSR is 1).

Notice that the organization of this MSR differs from that of IA32_VMX_EXIT_CTLS (Appendix A.4.1). This is because there are 64 secondary VM-exit controls, while there were only 32 primary VM-exit controls.

A.5 VM-ENTRY CONTROLS

The IA32_VMX_ENTRY_CTLS MSR (index 484H) reports on the allowed settings of **most** of the VM-entry controls (see Section 25.8.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the VM-entry controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

  Exceptions are made for the VM-entry controls in the default1 class (see Appendix A.2). These are bits 0–8 and 12; the corresponding bits of the IA32_VMX_ENTRY_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-entry control in the default1 class is 0.

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLS MSR (see below) reports which of the VM-entry controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-entry controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLS MSR (index 490H) reports on the allowed settings of **all** of the VM-entry controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-entry controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_ENTRY_CTLS MSR. (The IA32_VMX_TRUE_ENTRY_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_TRUE_ENTRY_CTLS MSR. Assuming that software knows that the default1 class of VM-entry controls contains bits 0–8 and 12, there is no need for software to consult the IA32_VMX_ENTRY_CTLS MSR.

A.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:

- Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.
- If bit 5 is read as 1, VM exits store the value of IA32_EFER.LMA into the "IA-32e mode guest" VM-entry control; see Section 28.2 for more details. This bit is read as 1 on any logical processor that supports the 1-setting of the "unrestricted guest" VM-execution control.
- Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
  - — Bit 6 reports (if set) the support for activity state 1 (HLT).
  - — Bit 7 reports (if set) the support for activity state 2 (shutdown).
  - — Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

  If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. All implementations support VM entry to activity state 0 (active).
- If bit 14 is read as 1, Intel® Processor Trace (Intel PT) can be used in VMX operation. If the processor supports Intel PT but does not allow it to be used in VMX operation, execution of VMXON clears IA32_RTIT_CTL.TraceEn (see "VMXON—Enter VMX Operation" in Chapter 31); any attempt to write IA32_RTIT_CTL while in VMX operation (including VMX root operation) causes a general-protection exception.
- If bit 15 is read as 1, the RDMSR instruction can be used in system-management mode (SMM) to read the IA32_SMBASE MSR (MSR address 9EH). See Section 32.15.6.3.
- Bits 24:16 indicate the number of CR3-target values supported by the processor. This number is a value between 0 and 256, inclusive (bit 24 is set if and only if bits 23:16 are clear).
- Bits 27:25 is used to compute the recommended maximum number of MSRs that should appear in the VM-exit MSR-store list, the VM-exit MSR-load list, or the VM-entry MSR-load list. Specifically, if the value bits 27:25 of IA32_VMX_MISC is N, then 512 * (N + 1) is the recommended maximum number of MSRs to be included in each list. If the limit is exceeded, undefined processor behavior may result (including a machine check during the VMX transition).
- If bit 28 is read as 1, bit 2 of the IA32_SMM_MONITOR_CTL can be set to 1. VMXOFF unblocks SMIs unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 32.14.4).
- If bit 29 is read as 1, software can use VMWRITE to write to any supported field in the VMCS; otherwise, VMWRITE cannot be used to modify VM-exit information fields.

- If bit 30 is read as 1, VM entry allows injection of a software interrupt, software exception, or privileged software exception with an instruction length of 0.
- Bits 63:32 report the 32-bit MSEG revision identifier used by the processor.
- Bits 13:9 and bit 31 are reserved and are read as 0.

A.7 VMX-FIXED BITS IN CR0

The IA32_VMX_CR0_FIXED0 MSR (index 486H) and IA32_VMX_CR0_FIXED1 MSR (index 487H) indicate how bits in CR0 may be set in VMX operation. They report on bits in CR0 that are allowed to be 0 and to be 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit of CR0 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit of CR0 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit is also 1 in IA32_VMX_CR0_FIXED1; if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit is also 0 in IA32_VMX_CR0_FIXED0. Thus, each bit in CR0 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in IA32_VMX_CR0_FIXED0 and 1 in IA32_VMX_CR0_FIXED1).

A.8 VMX-FIXED BITS IN CR4

The IA32_VMX_CR4_FIXED0 MSR (index 488H) and IA32_VMX_CR4_FIXED1 MSR (index 489H) indicate how bits in CR4 may be set in VMX operation. They report on bits in CR4 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit of CR4 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit of CR4 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit is also 1 in IA32_VMX_CR4_FIXED1; if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit is also 0 in IA32_VMX_CR4_FIXED0. Thus, each bit in CR4 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in IA32_VMX_CR4_FIXED0 and 1 in IA32_VMX_CR4_FIXED1).

A.9 VMCS ENUMERATION

The IA32_VMX_VMCS_ENUM MSR (index 48AH) provides information to assist software in enumerating fields in the VMCS.

As noted in Section 25.11.2, each field in the VMCS is associated with a 32-bit encoding which is structured as follows:

- Bits 31:15 are reserved (must be 0).
- Bits 14:13 indicate the field's width.
- Bit 12 is reserved (must be 0).
- Bits 11:10 indicate the field's type.
- Bits 9:1 is an index field that distinguishes different fields with the same width and type.
- Bit 0 indicates access type.

IA32_VMX_VMCS_ENUM indicates to software the highest index value used in the encoding of any field supported by the processor:

- Bits 9:1 contain the highest index value used for any VMCS encoding.
- Bit 0 and bits 63:10 are reserved and are read as 0.

A.10 VPID AND EPT CAPABILITIES

The IA32_VMX_EPT_VPID_CAP MSR (index 48CH) reports information about the capabilities of the logical processor with regard to virtual-processor identifiers (VPIDs, Section 29.1) and extended page tables (EPT, Section 29.3):

- If bit 0 is read as 1, the processor supports execute-only translations by EPT. This support allows software to configure EPT paging-structure entries in which bits 1:0 are clear (indicating that data accesses are not allowed) and bit 2 is set (indicating that instruction fetches are allowed).[1]
- Bit 6 indicates support for a page-walk length of 4.
- If bit 8 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be uncacheable (UC); see Section 25.6.11.
- If bit 14 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be write-back (WB).
- If bit 16 is read as 1, the logical processor allows software to configure a EPT PDE to map a 2-Mbyte page (by setting bit 7 in the EPT PDE).
- If bit 17 is read as 1, the logical processor allows software to configure a EPT PDPTE to map a 1-Gbyte page (by setting bit 7 in the EPT PDPTE).
- Support for the INVEPT instruction (see Chapter 31 and Section 29.4.3.1):
  - — If bit 20 is read as 1, the INVEPT instruction is supported.
  - — If bit 25 is read as 1, the single-context INVEPT type is supported.
  - — If bit 26 is read as 1, the all-context INVEPT type is supported.
- If bit 21 is read as 1, accessed and dirty flags for EPT are supported (see Section 29.3.5).
- If bit 22 is read as 1, the processor reports advanced VM-exit information for EPT violations (see Section 28.2.1). This reporting is done only if this bit is read as 1.
- If bit 23 is read as 1, supervisor shadow-stack control is supported (see Section 29.3.3.2).
- Support for the INVVPID instruction (see Chapter 31 and Section 29.4.3.1):
  - — If bit 32 is read as 1, the INVVPID instruction is supported.
  - — If bit 40 is read as 1, the individual-address INVVPID type is supported.
  - — If bit 41 is read as 1, the single-context INVVPID type is supported.
  - — If bit 42 is read as 1, the all-context INVVPID type is supported.
  - — If bit 43 is read as 1, the single-context-retaining-globals INVVPID type is supported.
- Bits 53:48 enumerate the maximum HLAT prefix size. It is expected that any processor that supports the 1-setting of the "enable HLAT" VM-execution control will enumerate this value as 1. See Section 4.5.1.
- Bits 5:1, bit 7, bits 13:9, bit 15, bits 19:18, bit 24, bits 31:27, bits 39:33, bits 47:44, and bits 63:54 are reserved and are read as 0.

The IA32_VMX_EPT_VPID_CAP MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1) and that support either the 1-setting of the "enable EPT" VM-execution control (only if bit 33 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1) or the 1-setting of the "enable VPID" VM-execution control (only if bit 37 of the IA32_VMX_PROC-BASED_CTLS2 MSR is 1).

1. If the "mode-based execute control for EPT" VM-execution control is 1, setting bit 0 indicates also that software may also configure EPT paging-structure entries in which bits 1:0 are both clear and in which bit 10 is set (indicating a translation that can be used to fetch instructions from a supervisor-mode linear address or a user-mode linear address).

A.11 VM FUNCTIONS

The IA32_VMX_VMFUNC MSR (index 491H) reports on the allowed settings of the VM-function controls (see Section 25.6.14). VM entry allows bit X of the VM-function controls to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if bit X of the VM-function controls, the "activate secondary controls" primary processor-based VM-execution control, and the "enable VM functions" secondary processor-based VM-execution control are all 1.

The IA32_VMX_VMFUNC MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1) and the 1-setting of the "enable VM functions" secondary processor-based VM-execution control (only if bit 45 of the IA32_VMX_PROC-BASED_CTLS2 MSR is 1).

APPENDIX B
FIELD ENCODING IN VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 25.11.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.).

B.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 25.11.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.1.1 16-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-1 enumerates the 16-bit control fields.

Table B-1. Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID)[1]	000000000B	00000000H
Posted-interrupt notification vector[2]	000000001B	00000002H
EPTP index[3]	000000010B	00000004H
HLAT prefix size[4]	000000011B	00000006H
Last PID-pointer index[5]	000000100B	00000008H

NOTES:

1. This field exists only on processors that support the 1-setting of the "enable VPID" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "process posted interrupts" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "EPT-violation #VE" VM-execution control.
4. This field exists only on processors that support the 1-setting of the "enable HLAT" VM-execution control.
5. This field exists only on processors that support the 1-setting of the "IPI virtualization" VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES selector	000000000B	00000800H
Guest CS selector	000000001B	00000802H
Guest SS selector	000000010B	00000804H
Guest DS selector	000000011B	00000806H
Guest FS selector	000000100B	00000808H

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH
Guest TR selector	000000111B	0000080EH
Guest interrupt status[1]	000001000B	00000810H
PML index[2]	000001001B	00000812H
Guest UINV[3]	000001010B	00000814H

NOTES:

1. This field exists only on processors that support the 1-setting of the "virtual-interrupt delivery" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "enable PML" VM-execution control.
3. This 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.

B.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-3 enumerates the 16-bit host-state fields.

Table B-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host ES selector	000000000B	00000C00H
Host CS selector	000000001B	00000C02H
Host SS selector	000000010B	00000C04H
Host DS selector	000000011B	00000C06H
Host FS selector	000000100B	00000C08H
Host GS selector	000000101B	00000C0AH
Host TR selector	000000110B	00000C0CH

B.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 25.11.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)		00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)		00002003H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Address of MSR bitmaps (full)[1]	000000010B	00002004H
Address of MSR bitmaps (high)[1]		00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)		00002007H
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)		00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)		0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)		0000200DH
PML address (full)[2]	000000111B	0000200EH
PML address (high)[2]		0000200FH
TSC offset (full)	000001000B	00002010H
TSC offset (high)		00002011H
Virtual-APIC address (full)[3]	000001001B	00002012H
Virtual-APIC address (high)[3]		00002013H
APIC-access address (full)[4]	000001010B	00002014H
APIC-access address (high)[4]		00002015H
Posted-interrupt descriptor address (full)[5]	000001011B	00002016H
Posted-interrupt descriptor address (high)[5]		00002017H
VM-function controls (full)[6]	000001100B	00002018H
VM-function controls (high)[6]		00002019H
EPT pointer (EPTP; full)[7]	000001101B	0000201AH
EPT pointer (EPTP; high)[7]		0000201BH
EOI-exit bitmap 0 (EOI_EXIT0; full)[8]	000001110B	0000201CH
EOI-exit bitmap 0 (EOI_EXIT0; high)[8]		0000201DH
EOI-exit bitmap 1 (EOI_EXIT1; full)[8]	000001111B	0000201EH
EOI-exit bitmap 1 (EOI_EXIT1; high)[8]		0000201FH
EOI-exit bitmap 2 (EOI_EXIT2; full)[8]	000010000B	00002020H
EOI-exit bitmap 2 (EOI_EXIT2; high)[8]		00002021H
EOI-exit bitmap 3 (EOI_EXIT3; full)[8]	000010001B	00002022H
EOI-exit bitmap 3 (EOI_EXIT3; high)[8]		00002023H
EPTP-list address (full)[9]	000010010B	00002024H
EPTP-list address (high)[9]		00002025H
VMREAD-bitmap address (full)[10]	000010011B	00002026H
VMREAD-bitmap address (high)[10]		00002027H
VMWRITE-bitmap address (full)[10]	000010100B	00002028H
VMWRITE-bitmap address (high)[10]		00002029H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Virtualization-exception information address (full)[11]	000010101B	0000202AH
Virtualization-exception information address (high)[11]		0000202BH
XSS-exiting bitmap (full)[12]	000010110B	0000202CH
XSS-exiting bitmap (high)[12]		0000202DH
ENCLS-exiting bitmap (full)[13]	000010111B	0000202EH
ENCLS-exiting bitmap (high)[13]		0000202FH
Sub-page-permission-table pointer (full)[14]	000011000B	00002030H
Sub-page-permission-table pointer (high)[14]		00002031H
TSC multiplier (full)[15]	000011001B	00002032H
TSC multiplier (high)[15]		00002033H
Tertiary processor-based VM-execution controls (full)[16]	000011010B	00002034H
Tertiary processor-based VM-execution controls (high)[16]		00002035H
ENCLV-exiting bitmap (full)[17]	000011011B	00002036H
ENCLV-exiting bitmap (high)[17]		00002037H
Low PASID directory address (full)[18]	000011100B	00002038H
Low PASID directory address (high)[18]		00002039H
High PASID directory address (full)[18]	000011101B	0000203AH
High PASID directory address (high)[18]		0000203BH
Shared EPT pointer (full)[19]	000011110B	0000203CH
Shared EPT pointer (high)[19]		0000203DH
PCONFIG-exiting bitmap (full)[20]	000011111B	0000203EH
PCONFIG-exiting bitmap (high)[20]		0000203FH
Hypervisor-managed linear-address translation pointer (HLATP; full)[21]	000100000B	00002040H
HLATP (high)[21]		00002041H
PID-pointer table address (full)[22]	000100001B	00002042H
PID-pointer table address (high)[22]		00002043H
Secondary VM-exit controls (full)[23]	000100010B	00002044H
Secondary VM-exit controls (high)[23]		00002045H
IA32_SPEC_CTRL mask (full)[24]	000100101B	0000204AH
IA32_SPEC_CTRL mask (high)[24]		0000204BH
IA32_SPEC_CTRL shadow (full)[24]	000100110B	0000204CH
IA32_SPEC_CTRL shadow (high)[24]		0000204DH

NOTES:

1. This field exists only on processors that support the 1-setting of the "use MSR bitmaps" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "enable PML" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.
4. This field exists only on processors that support the 1-setting of the "virtualize APIC accesses" VM-execution control.
5. This field exists only on processors that support the 1-setting of the "process posted interrupts" VM-execution control.
6. This field exists only on processors that support the 1-setting of the "enable VM functions" VM-execution control.
7. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

8. This field exists only on processors that support the 1-setting of the "virtual-interrupt delivery" VM-execution control.
9. This field exists only on processors that support the 1-setting of the "EPTP switching" VM-function control.
10. This field exists only on processors that support the 1-setting of the "VMCS shadowing" VM-execution control.
11. This field exists only on processors that support the 1-setting of the "EPT-violation #VE" VM-execution control.
12. This field exists only on processors that support the 1-setting of the "enable XSAVES/XRSTORS" VM-execution control.
13. This field exists only on processors that support the 1-setting of the "enable ENCLS exiting" VM-execution control.
14. This field exists only on processors that support the 1-setting of the "sub-page write permissions for EPT" VM-execution control.
15. This field exists only on processors that support the 1-setting of the "use TSC scaling" VM-execution control.
16. This field exists only on processors that support the 1-setting of the "activate tertiary controls" VM-execution control.
17. This field exists only on processors that support the 1-setting of the "enable ENCLV exiting" VM-execution control.
18. This field exists only on processors that support the 1-setting of the "PASID translation" VM-execution control.
19. This field exists only on processors that support the 1-setting of the "shared-EPTP" VM-execution control.
20. This field exists only on processors that support the 1-setting of the "enable PCONFIG" VM-execution control.
21. This field exists only on processors that support the 1-setting of the "enable HLAT" VM-execution control.
22. This field exists only on processors that support the 1-setting of the "IPI virtualization" VM-execution control.
23. This field exists only on processors that support the 1-setting of the "activate secondary controls" VM-exit control.
24. This field exists only on processors that support the 1-setting of the "virtualize IA32_SPEC_CTRL" VM-execution control.

B.2.2 64-Bit Read-Only Data Field

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. There is only one such 64-bit field as given in Table B-5.(As with other 64-bit fields, this one has two encodings.)

Table B-5. Encodings for 64-Bit Read-Only Data Field (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full)[1]	000000000B	00002400H
Guest-physical address (high)[1]		00002401H

NOTES:
1. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)		00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H
Guest IA32_DEBUGCTL (high)		00002803H
Guest IA32_PAT (full)[1]	000000010B	00002804H
Guest IA32_PAT (high)[1]		00002805H
Guest IA32_EFER (full)[2]	000000011B	00002806H
Guest IA32_EFER (high)[2]		00002807H

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Guest IA32_PERF_GLOBAL_CTRL (full)[3]	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high)[3]		00002809H
Guest PDPTE0 (full)[4]	000000101B	0000280AH
Guest PDPTE0 (high)[4]		0000280BH
Guest PDPTE1 (full)[4]	000000110B	0000280CH
Guest PDPTE1 (high)[4]		0000280DH
Guest PDPTE2 (full)[4]	000000111B	0000280EH
Guest PDPTE2 (high)[4]		0000280FH
Guest PDPTE3 (full)[4]	000001000B	00002810H
Guest PDPTE3 (high)[4]		00002811H
Guest IA32_BNDCFGS (full)[5]	000001001B	00002812H
Guest IA32_BNDCFGS (high)[5]		00002813H
Guest IA32_RTIT_CTL (full)[6]	000001010B	00002814H
Guest IA32_RTIT_CTL (high)[6]		00002815H
Guest IA32_LBR_CTL (full)[7]	000001011B	00002816H
Guest IA32_LBR_CTL (high)[7]		00002817H
Guest IA32_PKRS (full)[8]	000001100B	00002818H
Guest IA32_PKRS (high)[8]		00002819H

NOTES:

1. This field exists only on processors that support either the 1-setting of the "load IA32_PAT" VM-entry control or that of the "save IA32_PAT" VM-exit control.
2. This field exists only on processors that support either the 1-setting of the "load IA32_EFER" VM-entry control or that of the "save IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-entry control.
4. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.
5. This field exists only on processors that support either the 1-setting of the "load IA32_BNDCFGS" VM-entry control or that of the "clear IA32_BNDCFGS" VM-exit control.
6. This field exists only on processors that support either the 1-setting of the "load IA32_RTIT_CTL" VM-entry control or that of the "clear IA32_RTIT_CTL" VM-exit control.
7. 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.
8. This field exists only on processors that support the 1-setting of the "load PKRS" VM-entry control.

B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full)[1]	000000000B	00002C00H
Host IA32_PAT (high)[1]		00002C01H

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Host IA32_EFER (full)[2]	000000001B	00002C02H
Host IA32_EFER (high)[2]		00002C03H
Host IA32_PERF_GLOBAL_CTRL (full)[3]	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high)[3]		00002C05H
Host IA32_PKRS (full)[4]	000000011B	00002C06H
Host IA32_PKRS (high)[4]		00002C07H

NOTES:

1. This field exists only on processors that support the 1-setting of the "load IA32_PAT" VM-exit control.
2. This field exists only on processors that support the 1-setting of the "load IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-exit control.
4. This field exists only on processors that support the 1-setting of the "load PKRS" VM-exit control.

B.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 25.11.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-8 enumerates the 32-bit control fields.

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Pin-based VM-execution controls	000000000B	00004000H
Primary processor-based VM-execution controls	000000001B	00004002H
Exception bitmap	000000010B	00004004H
Page-fault error-code mask	000000011B	00004006H
Page-fault error-code match	000000100B	00004008H
CR3-target count	000000101B	0000400AH
Primary VM-exit controls	000000110B	0000400CH
VM-exit MSR-store count	000000111B	0000400EH
VM-exit MSR-load count	000001000B	00004010H
VM-entry controls	000001001B	00004012H
VM-entry MSR-load count	000001010B	00004014H
VM-entry interruption-information field	000001011B	00004016H
VM-entry exception error code	000001100B	00004018H
VM-entry instruction length	000001101B	0000401AH
TPR threshold[1]	000001110B	0000401CH
Secondary processor-based VM-execution controls[2]	000001111B	0000401EH
PLE_Gap[3]	000010000B	00004020H

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
PLE_Window[3]	000010001B	00004022H
Instruction-timeout control[4]	000010010B	00004024H

NOTES:

1. This field exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "PAUSE-loop exiting" VM-execution control.
4. This field exists only on processors that support the 1-setting of the "instruction timeout" VM-execution control.

B.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-9 enumerates the 32-bit read-only data fields.

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
VM-instruction error	000000000B	00004400H
Exit reason	000000001B	00004402H
VM-exit interruption information	000000010B	00004404H
VM-exit interruption error code	000000011B	00004406H
IDT-vectoring information field	000000100B	00004408H
IDT-vectoring error code	000000101B	0000440AH
VM-exit instruction length	000000110B	0000440CH
VM-exit instruction information	000000111B	0000440EH

B.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-10 enumerates the 32-bit guest-state fields.

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES limit	000000000B	00004800H
Guest CS limit	000000001B	00004802H
Guest SS limit	000000010B	00004804H
Guest DS limit	000000011B	00004806H
Guest FS limit	000000100B	00004808H
Guest GS limit	000000101B	0000480AH
Guest LDTR limit	000000110B	0000480CH
Guest TR limit	000000111B	0000480EH
Guest GDTR limit	000001000B	00004810H
Guest IDTR limit	000001001B	00004812H
Guest ES access rights	000001010B	00004814H

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest CS access rights	000001011B	00004816H
Guest SS access rights	000001100B	00004818H
Guest DS access rights	000001101B	0000481AH
Guest FS access rights	000001110B	0000481CH
Guest GS access rights	000001111B	0000481EH
Guest LDTR access rights	000010000B	00004820H
Guest TR access rights	000010001B	00004822H
Guest interruptibility state	000010010B	00004824H
Guest activity state	000010011B	00004826H
Guest SMBASE	000010100B	00004828H
Guest IA32_SYSENTER_CS	000010101B	0000482AH
VMX-preemption timer value[1]	000010111B	0000482EH

NOTES:

1. This field exists only on processors that support the 1-setting of the "activate VMX-preemption timer" VM-execution control.

The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

B.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table B-11.

Table B-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host IA32_SYSENTER_CS	000000000B	00004C00H

B.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 25.11.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-12 enumerates the natural-width control fields.

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR0 guest/host mask	000000000B	00006000H
CR4 guest/host mask	000000001B	00006002H
CR0 read shadow	000000010B	00006004H
CR4 read shadow	000000011B	00006006H

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
CR3-target value 0	000000100B	00006008H
CR3-target value 1	000000101B	0000600AH
CR3-target value 2	000000110B	0000600CH
CR3-target value 3[1]	000000111B	0000600EH

NOTES:

1. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

B.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-13 enumerates the natural-width read-only data fields.

Table B-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
Exit qualification	000000000B	00006400H
I/O RCX	000000001B	00006402H
I/O RSI	000000010B	00006404H
I/O RDI	000000011B	00006406H
I/O RIP	000000100B	00006408H
Guest-linear address	000000101B	0000640AH

B.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-14 enumerates the natural-width guest-state fields.

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CR0	000000000B	00006800H
Guest CR3	000000001B	00006802H
Guest CR4	000000010B	00006804H
Guest ES base	000000011B	00006806H
Guest CS base	000000100B	00006808H
Guest SS base	000000101B	0000680AH
Guest DS base	000000110B	0000680CH
Guest FS base	000000111B	0000680EH
Guest GS base	000001000B	00006810H
Guest LDTR base	000001001B	00006812H
Guest TR base	000001010B	00006814H
Guest GDTR base	000001011B	00006816H

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest IDTR base	000001100B	00006818H
Guest DR7	000001101B	0000681AH
Guest RSP	000001110B	0000681CH
Guest RIP	000001111B	0000681EH
Guest RFLAGS	000010000B	00006820H
Guest pending debug exceptions	000010001B	00006822H
Guest IA32_SYSENTER_ESP	000010010B	00006824H
Guest IA32_SYSENTER_EIP	000010011B	00006826H
Guest IA32_S_CET[1]	000010100B	00006828H
Guest SSP[1]	000010101B	0000682AH
Guest IA32_INTERRUPT_SSP_TABLE_ADDR[1]	000010110B	0000682CH

NOTES:
1. This field is supported only on processors that support the 1-setting of the "load CET state" VM-entry control.

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

B.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-15 enumerates the natural-width host-state fields.

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host CR0	000000000B	00006C00H
Host CR3	000000001B	00006C02H
Host CR4	000000010B	00006C04H
Host FS base	000000011B	00006C06H
Host GS base	000000100B	00006C08H
Host TR base	000000101B	00006C0AH
Host GDTR base	000000110B	00006C0CH
Host IDTR base	000000111B	00006C0EH
Host IA32_SYSENTER_ESP	000001000B	00006C10H
Host IA32_SYSENTER_EIP	000001001B	00006C12H
Host RSP	000001010B	00006C14H
Host RIP	000001011B	00006C16H
Host IA32_S_CET[1]	000001100B	00006C18H
Host SSP[1]	000001101B	00006C1AH
Host IA32_INTERRUPT_SSP_TABLE_ADDR[1]	000001110B	00006C1CH

NOTES:
1. This field is supported only on processors that support the 1-setting of the "load CET state" VM-exit control.

APPENDIX C
VMX BASIC EXIT REASONS

Every VM exit writes a 32-bit exit reason to the VMCS (see Section 25.9.1). Certain VM-entry failures also do this (see Section 27.8). The low 16 bits of the exit-reason field form the basic exit reason which provides basic information about the cause of the VM exit or VM-entry failure.

Table C-1 lists values for basic exit reasons and explains their meaning. Entries apply to VM exits, unless otherwise noted.

Table C-1. Basic Exit Reasons

Basic Exit Reason	Description
0	**Exception or non-maskable interrupt (NMI).** Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. This case includes executions of BOUND that cause #BR, executions of INT1 (they cause #DB), executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD0, UD1, and UD2 (they cause #UD). 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1.
1	**External interrupt.** An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	**Triple fault.** The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	**INIT signal.** An INIT signal arrived
4	**Start-up IPI (SIPI).** A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	**I/O system-management interrupt (SMI).** An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 32.15.2).
6	**Other SMI.** An SMI arrived and caused an SMM VM exit (see Section 32.15.2) but not immediately after retirement of an I/O instruction.
7	**Interrupt window.** At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	**NMI window.** At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	**Task switch.** Guest software attempted a task switch.
10	**CPUID.** Guest software attempted to execute CPUID.
11	**GETSEC.** Guest software attempted to execute GETSEC.
12	**HLT.** Guest software attempted to execute HLT and the "HLT exiting" VM-execution control was 1.
13	**INVD.** Guest software attempted to execute INVD.
14	**INVLPG.** Guest software attempted to execute INVLPG and the "INVLPG exiting" VM-execution control was 1.
15	**RDPMC.** Guest software attempted to execute RDPMC and the "RDPMC exiting" VM-execution control was 1.
16	**RDTSC.** Guest software attempted to execute RDTSC and the "RDTSC exiting" VM-execution control was 1.
17	**RSM.** Guest software attempted to execute RSM in SMM.
18	**VMCALL.** VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 32.15.2).
19	**VMCLEAR.** Guest software attempted to execute VMCLEAR.
20	**VMLAUNCH.** Guest software attempted to execute VMLAUNCH.
21	**VMPTRLD.** Guest software attempted to execute VMPTRLD.
22	**VMPTRST.** Guest software attempted to execute VMPTRST.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
23	**VMREAD.** Guest software attempted to execute VMREAD.
24	**VMRESUME.** Guest software attempted to execute VMRESUME.
25	**VMWRITE.** Guest software attempted to execute VMWRITE.
26	**VMXOFF.** Guest software attempted to execute VMXOFF.
27	**VMXON.** Guest software attempted to execute VMXON.
28	**Control-register accesses.** Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 26.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the "use TPR shadow" VM-execution control is 1. Such VM exits instead use basic exit reason 43.
29	**MOV DR.** Guest software attempted a MOV to or from a debug register and the "MOV-DR exiting" VM-execution control was 1.
30	**I/O instruction.** Guest software attempted to execute an I/O instruction and either: 1: The "use I/O bitmaps" VM-execution control was 0 and the "unconditional I/O exiting" VM-execution control was 1. 2: The "use I/O bitmaps" VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1.
31	**RDMSR.** Guest software attempted to execute RDMSR and either: 1: The "use MSR bitmaps" VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the n^{th} bit in read bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the n^{th} bit in read bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
32	**WRMSR.** Guest software attempted to execute WRMSR and either: 1: The "use MSR bitmaps" VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the n^{th} bit in write bitmap for low MSRs is 1, where n was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the n^{th} bit in write bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
33	**VM-entry failure due to invalid guest state.** A VM entry failed one of the checks identified in Section 27.3.1.
34	**VM-entry failure due to MSR loading.** A VM entry failed in an attempt to load MSRs. See Section 27.4.
36	**MWAIT.** Guest software attempted to execute MWAIT and the "MWAIT exiting" VM-execution control was 1.
37	**Monitor trap flag.** A VM exit occurred due to the 1-setting of the "monitor trap flag" VM-execution control (see Section 26.5.2) or VM entry injected a pending MTF VM exit as part of VM entry (see Section 27.6.2).
39	**MONITOR.** Guest software attempted to execute MONITOR and the "MONITOR exiting" VM-execution control was 1.
40	**PAUSE.** Either guest software attempted to execute PAUSE and the "PAUSE exiting" VM-execution control was 1 or the "PAUSE-loop exiting" VM-execution control was 1 and guest software executed a PAUSE loop with execution time exceeding PLE_Window (see Section 26.1.3).
41	**VM-entry failure due to machine-check event.** A machine-check event occurred during VM entry (see Section 27.9).
43	**TPR below threshold.** The logical processor determined that the value of bits 7:4 of the byte at offset 080H on the virtual-APIC page was below that of the TPR threshold VM-execution control field while the "use TPR shadow" VM-execution control was 1 either as part of TPR virtualization (Section 30.1.2) or VM entry (Section 27.7.7).
44	**APIC access.** Guest software attempted to access memory at a physical address on the APIC-access page and the "virtualize APIC accesses" VM-execution control was 1 (see Section 30.4).
45	**Virtualized EOI.** EOI virtualization was performed for a virtual interrupt whose vector indexed a bit set in the EOI-exit bitmap.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
46	**Access to GDTR or IDTR.** Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the "descriptor-table exiting" VM-execution control was 1.
47	**Access to LDTR or TR.** Guest software attempted to execute LLDT, LTR, SLDT, or STR and the "descriptor-table exiting" VM-execution control was 1.
48	**EPT violation.** An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.
49	**EPT misconfiguration.** An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.
50	**INVEPT.** Guest software attempted to execute INVEPT.
51	**RDTSCP.** Guest software attempted to execute RDTSCP and the "enable RDTSCP" and "RDTSC exiting" VM-execution controls were both 1.
52	**VMX-preemption timer expired.** The preemption timer counted down to zero.
53	**INVVPID.** Guest software attempted to execute INVVPID.
54	**WBINVD or WBNOINVD.** Guest software attempted to execute WBINVD or WBNOINVD and the "WBINVD exiting" VM-execution control was 1.
55	**XSETBV.** Guest software attempted to execute XSETBV.
56	**APIC write.** Guest software completed a write to the virtual-APIC page that must be virtualized by VMM software (see Section 30.4.3.3).
57	**RDRAND.** Guest software attempted to execute RDRAND and the "RDRAND exiting" VM-execution control was 1.
58	**INVPCID.** Guest software attempted to execute INVPCID and the "enable INVPCID" and "INVLPG exiting" VM-execution controls were both 1.
59	**VMFUNC.** Guest software invoked a VM function with the VMFUNC instruction and the VM function either was not enabled or generated a function-specific condition causing a VM exit.
60	**ENCLS.** Guest software attempted to execute ENCLS, "enable ENCLS exiting" VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the ENCLS-exiting bitmap is 1; or (2) EAX ? 63 and bit 63 in the ENCLS-exiting bitmap is 1.
61	**RDSEED.** Guest software attempted to execute RDSEED and the "RDSEED exiting" VM-execution control was 1.
62	**Page-modification log full.** The processor attempted to create a page-modification log entry and the value of the PML index was not in the range 0–511.
63	**XSAVES.** Guest software attempted to execute XSAVES, the "enable XSAVES/XRSTORS" was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
64	**XRSTORS.** Guest software attempted to execute XRSTORS, the "enable XSAVES/XRSTORS" was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
65	**PCONFIG.** Guest software attempted to execute PCONFIG, "enable PCONFIG" VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1; or (2) EAX ? 63 and bit 63 in the PCONFIG-exiting bitmap is 1.
66	**SPP-related event.** The processor attempted to determine an access's sub-page write permission and encountered an SPP miss or an SPP misconfiguration. See Section 29.3.4.2.
67	**UMWAIT.** Guest software attempted to execute UMWAIT and the "enable user wait and pause" and "RDTSC exiting" VM-execution controls were both 1.
68	**TPAUSE.** Guest software attempted to execute TPAUSE and the "enable user wait and pause" and "RDTSC exiting" VM-execution controls were both 1.
69	**LOADIWKEY.** Guest software attempted to execute LOADIWKEY and the "LOADIWKEY exiting" VM-execution control was 1.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
70	**ENCLV.** Guest software attempted to execute ENCLV, the "enable ENCLV exiting" VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the ENCLV-exiting bitmap is 1; or (2) EAX ? 63 and bit 63 in the ENCLV-exiting bitmap is 1.
72	**ENQCMD PASID translation failure.** A VM exit occurred during PASID translation because the present bit was clear in a PASID-directory entry, the valid bit was clear in a PASID-table entry, or one of the entries set a reserved bit.
73	**ENQCMDS PASID translation failure.** A VM exit occurred during PASID translation because the present bit was clear in a PASID-directory entry, the valid bit was clear in a PASID-table entry, or one of the entries set a reserved bit.
74	**Bus lock.** The processor asserted a bus lock while the "bus-lock detection" VM-execution control was 1. (Such VM exits will also set bit 26 of the exit-reason field.)
75	**Instruction timeout.** The "instruction timeout" VM-execution control was 1 and certain operations prevented the processor from reaching an instruction boundary within the amount of time specified by the instruction-timeout control.

INDEX

Numerics

A

D

E

F

G

H

I

M

N

O

P

T

U

V

W

X

Z