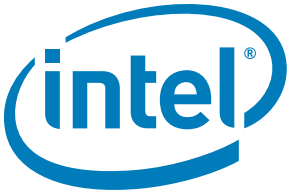

Software Guard Extensions Programming Reference

329298-001US

SEPTEMBER 2013

CHAPTER 1
INTRODUCTION TO SOFTWARE GUARD EXTENSIONS

CHAPTER 2
ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

CHAPTER 3
ENCLAVE OPERATION

CHAPTER 4
ENCLAVE EXITING EVENTS

CHAPTER 5
INSTRUCTION REFERENCES

CHAPTER 6
SGX INTERACTIONS WITH IA32 AND INTEL 64 ARCHITECTURE

CHAPTER 7
ENCLAVE CODE DEBUG AND PROFILING

TABLES

This page was intentionally left blank.

FIGURES

This page was intentionally left blank.

CHAPTER 1
INTRODUCTION TO SOFTWARE GUARD EXTENSIONS

1.1 OVERVIEW

This document describes the Software Guard Extensions (SGX), a set of instructions and mechanisms for memory accesses added to future Intel® Architecture processors. These extensions allow an application to instantiate a protected container, referred to as an enclave. An enclave is a protected area in the application's address space (see Figure 1-1), which provides confidentiality and integrity even in the presence of privileged malware. Accesses to the enclave memory area from any software not resident in the enclave are prevented.

Chapter 2 covers main concepts, objects and data structure formats that interact within the SGX architecture. Chapter 3 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 4 describes the behavior of Asynchronous Enclave Exit (AEX) caused by events while executing enclave code. Chapter 5 covers the syntax and operational details of the instruction and associated leaf functions available in SGX. Chapter 6 describes interaction of various aspects of IA32 and Intel 64 architectures with Intel SGX. Chapter 7 covers SGX support for application debug, profiling and performance monitoring.

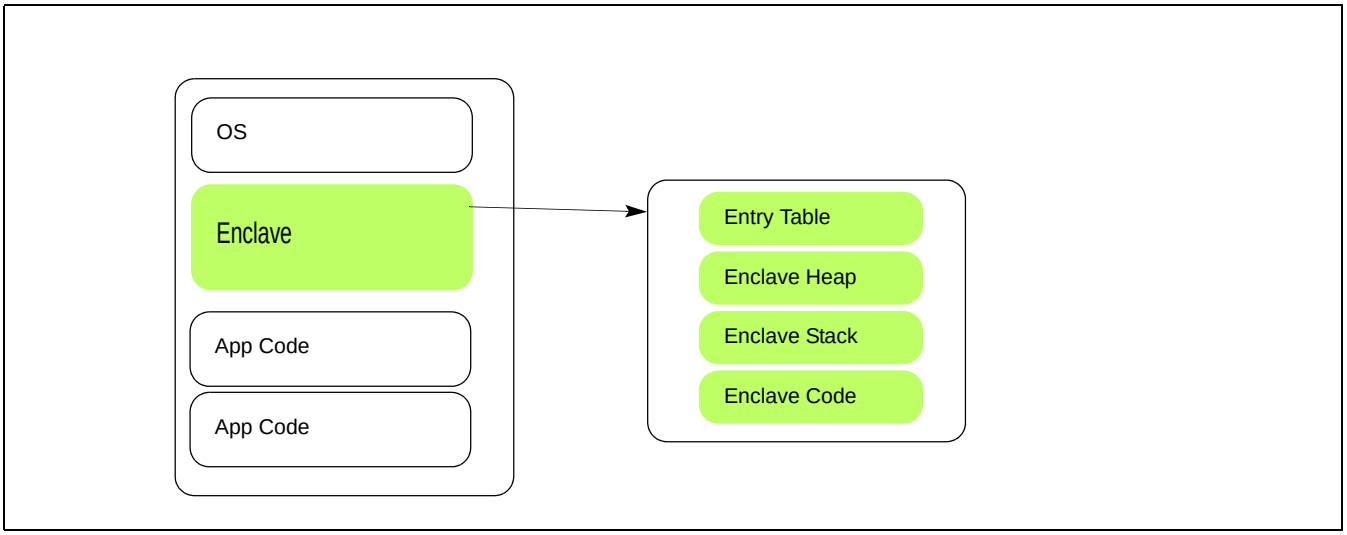

Figure 1-1. An Enclave Within the Application's Virtual Address Space

1.2 ENCLAVE INTERACTION AND PROTECTION

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

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.

1. For additional information, see white papers on Intel SGX at http://software.intel.com/en-us/intel-isa-extensions.

1.3 ENCLAVE LIFE CYCLE

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

Proper memory management procedure for enclave memory access or non-enclave memory access are required throughout the life cycle of an enclave: from creation, use, to destruction.

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

Un-trusted application code starts using an initialized enclave typically by using the SGX EENTER instruction to transfer control to the enclave code residing in protected enclave page cache (EPC). The enclave code returns to the caller via the EEXIT instruction. Upon enclave entry, control is transferred by hardware to software inside the enclave. the software inside the enclave switches the stack pointer to one inside the enclave. When returning back from the enclave, the software swaps the stack pointer then executes the EEXIT instruction.

Calling an external procedure from an enclave could also be done using the EEXIT instruction. EEXIT and a software convention between the trusted section and the un-trusted section.

An active enclave consumes available resource from the EPC. SGX provides the EREMOVE instruction that an EPC manager can use to reclaim resources committed to an enclave no longer in use. The EPC manager uses EREMOVE on every page. After execution of EREMOVE the page is available for allocation to another enclave.

1.4 DATA STRUCTURES AND ENCLAVE OPERATION

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

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

Every enclave contains one or more TCSs. The TCS contains meta-data used by the hardware to save and restore thread specific information when entering/exiting the enclave. There is one field, FLAGS, which may be accessed by software.

1.5 ENCLAVE PAGE CACHE

The Enclave Page Cache (EPC) is a secure storage used by the processor to store enclave pages when they are a part of an executing enclave.

The EPC is divided into chunks of 4KB pages. An EPC page is always aligned on a 4KB boundary.

EPC is used to hold pages belonging to instance of enclaves. Pages in the EPC can either be valid or invalid. Every valid page in the EPC belongs to one enclave instance. Each enclave instance has one EPC page holding its SECS. The security metadata for each EPC page are held in an internal micro-architecture structure called Enclave Page Cache Map (EPCM).

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

1.5.1 Enclave Page Cache Map (EPCM)

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

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

The EPCM structure is used by the CPU in the address-translation flow to enforce access-control on the enclave pages loaded into the EPC. The EPCM structure is described in Table 2-22, and the conceptual access-control flow is described in Section 2.5.

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

1.6 ENCLAVE INSTRUCTIONS AND SGX

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

Each leaf function index is also associated with a unique, leaf-specific mnemonic. A long-form expression of SGX instruction takes the form of ENCLx[LEAF_MNEMONIC], where ‘x’ is either ‘S’ or ‘U’. The long-form expression provides clear association of the privilege-level requirement of a given “leaf mnemonic”. For simplicity, the unique “Leaf_Mnemonic” name is also used interchangeably in this document for brevity.

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

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EADD]	Add a page	ENCLU[EENTER]	Enter an Enclave
ENCLS[EBLOCK]	Block an EPC page	ENCLU[EEXIT]	Exit an Enclave
ENCLS[ECREATE]	Create an enclave	ENCLU[EGETKEY]	Create a cryptographic key
ENCLS[EDBGRD]	Read data by debugger	ENCLU[EREPORT]	Create a cryptographic report
ENCLS[EDBGWR]	Write data by debugger	ENCLU[ERESUME]	Re-enter an Enclave
ENCLS[EEXTEND]	Extend EPC page measurement		
ENCLS[EINIT]	Initialize an enclave		
ENCLS[ELDB]	Load an EPC page as blocked		
ENCLS[ELDU]	Load an EPC page as unblocked		
ENCLS[EPA]	Add version array		
ENCLS[EREMOVE]	Remove a page from EPC		
ENCLS[ETRACK]	Activate EBLOCK checks		
ENCLS[EWB]	Write back/invalidate an EPC page		

1.7 DISCOVERING SUPPORT FOR SGX AND ENABLING ENCLAVE INSTRUCTIONS

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

- Processor support of 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 SGX, and requires opt-in enabling by BIOS via IA32_FEATURE_CONTROL MSR and on-demand enabling by system software via CR4.SEE.

  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 SGX resources.
- The available and configured SGX resources enumerated by the sub-leaves of CPUID.(EAX=12H) depend on the state of opt-in configuration by BIOS and of the CR4.SEE setting by system software.

1.7.1 SGX Opt-In Configuration

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

1.7.2 System Software Enabling of SGX

System software enables SGX by setting CR4.SEE bit = 1. Any attempt to set this bit when CPUID.(EAX=07H, ECX=0H):EBX.SGX is 0 results in a #GP. The CR4.SEE bit is not locked, and can be set/cleared at any time, as long as the operation is consistent with CPUID.(EAX=07H, ECX=0H):EBX.SGX enumeration.

Table 1-2. SGX Opt-in and Enabling Behavior

CPUID.(07H,0H):EBX.SGX	CPUID.(12H)	CR4.SEE	FEATURE_CONTROL.LOCK	FEATURE_CONTROL.SGX_ENABLE	Enclave Instruction
0	Invalid	Reserved	X	X	#UD
1	Valid*	0	X	X	#UD
1	Valid*	1	0	X	#GP
1	Valid*	1	1	0	#GP
1	Valid*	1	1	1	Available
* Leaf 12H enumeration results are dependent on enablement.					

1.7.3 SGX Resource Enumeration Leaves

If CPUID.(EAX=07H, ECX=0H):EBX.SGX = 1, the processor also supports querying CPUID with EAX=12H on 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 SGX instructions are supported on one hardware thread, they are also supported elsewhere.

A properly configured processor exposes SGX functionality with CPUID.EAX=12H reporting valid information (non-zero content) in three or more sub-leaves, see Table 1-3:

- CPUID.(EAX=12H, ECX=0H) enumerates SGX capability, including enclave instruction opcode support.

- CPUID.(EAX=12H, ECX=1H) enumerates SGX capability of processor state configuration and enclave configuration in the SECS structure (see Table 2-3).
- CPUID.(EAX=12H, ECX >1) enumerates available EPC resources.

Table 1-3. CPUID Leaf 12H, Sub-Leaf 0 Enumeration of SGX Capabilities

CPUID.(EAX=12H,ECX=0)		Description Behavior
Register	Bits	
EAX	0	If 1, indicates SGX instruction opcodes listed in Table 1-1are supported.
	31:1	Reserved (0)
EBX	31:0	Reserved (0)
ECX	31:0	Reserved (0)
EDX	31:0	Reserved (0)

Table 1-4. CPUID Leaf 12H, Sub-Leaf 1 Enumeration of SGX Capabilities

CPUID.(EAX=12H,ECX=1)		Description Behavior
Register	Bits	
EAX	31:0	Report the valid bit fields of bits [31:0] of SECS.ATTRIBUTES that software can set with ECREATE
EBX	31:0	Report the valid bit fields of bits [63:32] of SECS.ATTRIBUTES that software can set with ECREATE
ECX	31:0	Report the valid bit fields of bits [95:64] of SECS.ATTRIBUTES that software can set with ECREATE
EDX	31:0	Report the valid bit fields of bits [127:96] of SECS.ATTRIBUTES that software can set with ECREATE

CPUID leaf 12H sub-leaves 2 and higher report physical memory resources available for use with SGX. These physical memory sections are typically configured by BIOS as **Processor Reserved Memory**, and available to the OS to manage as EPC.

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

Table 1-5. CPUID Leaf 12H, Sub-Leaf Index 2 or Higher Enumeration of SGX Resources

CPUID.(EAX=12H,ECX > 1)		Description Behavior
Register	Bits	
EAX	3:0	0000b: This sub-leaf is invalid, EBX:EAX and EDX:ECX report 0. 0001b: This sub-leaf provides information on the Enclave Page Cache (EPC) in EBX:EAX and EDX:ECX. All other encoding are reserved.
	11:4	Reserved (0)
	31:12	If EAX[3:0] = 0001b, these are bits 31:12 of the physical address of the base of the EPC section.
EBX	19:0	If EAX[3:0] = 0001b, these are bits 51:32 of the physical address of the base of the EPC section.
	31:20	Reserved (0)
ECX	3: 0	0000b: Not valid 0001b: The EPC section is confidentiality, integrity and replay protected. All other encoding are reserved.
	11:4	Reserved (0)
	31:12	If EAX[3:0] = 0001b, these are bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory.

Table 1-5. CPUID Leaf 12H, Sub-Leaf Index 2 or Higher Enumeration of 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 (0)

CHAPTER 2
ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

2.1 OVERVIEW OF ENCLAVE EXECUTION ENVIRONMENT

An enclave comprises a contiguous range in the linear address space. An enclave must run from a special area of physical memory called Enclave Page Cache (EPC), which is protected from “non-enclave” memory accesses. An enclave need not be physically contiguous within the EPC. It is up to the EPC manager to allocate EPC pages to various enclaves. Enclaves abide by OS/VMM imposed segmentation and paging policies. OS/VMM-managed page tables and extended page tables provide address translation for the enclave pages, and the hardware guarantees that these pages will be mapped to EPC (any failure generates an exception).

2.2 TERMINOLOGY

A memory access that is initiated by internal enclave code to a linear address inside that enclave is called a Direct Enclave Access (Direct EA).

Memory accesses initiated by certain SGX instruction leaf functions such as ECREATE, EADD, EDBGRD, EDBGWR, ELDU/ELDB, EWB, EREMOVE, EENTER, and ERESUME that need to access EPC data by a non-enclave, managing context are called Indirect Enclave Accesses (Indirect EA).

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.

2.3 ACCESS-CONTROL REQUIREMENTS

Enclave accesses have the following access-control requirements:

- All memory accesses must conform to segmentation and paging policies set by the OS/VMM.
- Enclave entry/exit must happen through specific enclave instructions or events:
  - ENCLU[EENTER], ENCLU[ERESUME]
  - ENCLU[EEXIT], Asynchronous Enclave Exit (AEX).
- Direct jumps from outside an enclave to any linear address that maps to an enclave page do not enable enclave mode and result in abort page semantics and undefined behavior.
- Code fetches from inside an enclave to a linear address outside that enclave result in a #GP(0) exception.
- Non-enclave accesses to EPC memory result in abort page semantics.
- Hardware detects and prevents enclave accesses using logical-to-linear address translations which are different than the original direct EA used to allocate the page. Detection of modified translation results in #GP(0).
- Direct EAs to any EPC pages must conform to the security attributes which were established when those pages were added into the EPC:
  - Target must belong to the same enclave,
  - RWX attributes of the access must be compatible with the RWX permissions with which the target was added to the EPC,
  - Target must not have a restricted page type (PT_SECS, PT_TCS or PT_VA),
  - The EPC page must not be BLOCKED.

NOTE

Read accesses with abort-page semantics return 1 for every bit read. Write accesses with abort-page semantics are ignored.

2.4 SEGMENT-BASED ACCESS CONTROL

SGX architecture does not modify the segment checks performed by a logical processor. All memory accesses arising from a logical processor in protected mode (including one that is inside an enclave) are subject to segmentation checks with the appropriate segment register.

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

On enclave entry either via EENTER or ERESUME, the processor saves the contents of the FS and GS registers, and modifies these registers to enable the enclave's use of these registers for accessing the thread-local storage inside the enclave. FS and GS are loaded from values stored in the TCS at build time. On enclave exit, the contents at time of entry are restored. The details of these modifications can be found in the descriptions of EENTER, ERESUME, EEXIT, and AEX flows.

2.5 PAGE-BASED ACCESS CONTROL

2.5.1 Access-control for Accesses that Originate from non-SGX Instructions

SGX builds on the processor's paging mechanism to afford enclaves a protected execution environment. SGX provides page-granular access-control for enclave pages that are loaded into an EPC. Enclave pages loaded into an EPC are only accessible from inside the same enclave, or through certain SGX instructions.

2.5.2 Memory Accesses that Split across ELRANGE

Memory data accesses are allowed to split across ELRANGE (i.e., a part of the access is inside ELRANGE and a part of the access is outside ELRANGE) while the processor is inside an enclave. If an access splits across ELRANGE, the processor splits the access into two sub-accesses (one inside ELRANGE and the other outside ELRANGE), and each access is evaluated. A code-fetch access that splits across ELRANGE results in a #GP due to the portion that lies outside of the ELRANGE.

2.5.3 Implicit vs. Explicit Accesses

Memory accesses originating from SGX instruction leaf functions are categorized as either explicit accesses or implicit accesses. Table 2-1 lists the implicit and explicit memory accesses made by SGX leaf functions.

2.5.3.1 Explicit Accesses

Accesses to memory locations provided as explicit operands to 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 7.3.5.

2.5.3.2 Implicit Accesses

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

Implicit accesses are made using physical addresses that are cached by the processor. These accesses do not trigger any access-control faults/exits or data breakpoints. Table 2-1 lists memory objects that 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 2-1 matching implicitly encoded registers RBX, RCX, RDX.

Physical addresses used in different implicit accesses are cached via different instructions and for different durations. The physical address of SECS associated with each EPC page is cached at the time the page is added to the enclave via ENCLS[EADD]. This binding is severed when the corresponding page is removed from the EPC via ENCLS[EREMOVE]. Physical addresses of TCS and SSA memory are cached at the time of most-recent enclave entry. Exit from an enclave (ENCLU[EEXIT] or AEX) flushes this caching. Details of Asynchronous Enclave Exit is described in Chapter 4.

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

Table 2-1. List of Implicit and Explicit Memory Access by SGX Enclave Instructions

Instr. Leaf	Explicit 1	Explicit 2	Explicit 3	Implicit
ECREATE	PAGEINFO and linked structures	EPCPAGE		
EADD	PAGEINFO and linked structures	EPCPAGE		
EEXTEND	EPCPAGE			SECS
EINIT	SIGSTRUCT	SECS	EINITTOKEN	
EBLOCK	EPCPAGE			SECS
ETRACK	EPCPAGE			
ELDB/ELDU	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	
EWB	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	SECS
EREMOVE	EPCPAGE			SECS
EDBGRD	EPCADDR	Destination		SECS
EDBGWR	EPCADDR	Source		SECS
EENTER	TCS and linked SSA			SECS
ERESUME	TCS and linked SSA			SECS
EGETKEY	KEYREQUEST	KEY		SECS
EREPORT	TARGETINFO	REPORTDATA	OUTPUTDATA	SECS
EEXIT				SECS, TCS
EPA	EPCADDR			
Asynchronous Enclave Exit*				SECS, TCS, SSA
*Details of Asynchronous Enclave Exit (AEX) is described in Section 4.4				

2.6 SGX DATA STRUCTURES

2.6.1 SGX Enclave Control Structure (SECS)

The SECS data structure requires 4K-Bytes alignment.

Table 2-2. 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 1 SSA frame in pages (including XSAVE).
RESERVED	20	28	
ATTRIBUTES	48	16	Attributes of the Enclave, see Table 2-3
MRENCLAVE	64	32	Measurement Register of enclave build process. See SIGSTRUCT for proper format.
RESERVED	96	32	
MRSIGNER	128	32	Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for proper format.
RESERVED	160	96	
ISVPRODID	256	2	Product ID of enclave
ISVSVN	258	2	Security version number (SVN) of the enclave
EID	Implementation dependent	8	Enclave Identifier
PADDING	Implementation dependent	352	Padding pattern from the Signature (used for key derivation strings)
RESERVED	260	3836	Includes EID, other non-zero reserved field and must-be-zero fields

2.6.1.1 ATTRIBUTES

The ATTRIBUTES data structure is comprised of bit-granular fields that are used in the SECS, CPUID enumeration, the REPORT and the KEYREQUEST structures.

Table 2-3. Layout of ATTRIBUTES Structure

Field	Bit Position	Description
RESERVED	0	
DEBUG	1	If 1, the enclave permit debugger to read and write data to enclave
MODE64BIT	2	Enclave runs in 64-bit mode
RESERVED	3	Must be Zero
PROVISIONKEY	4	Provisioning Key is available from EGETKEY
EINITTOKENKEY	5	EINIT token key is available from EGETKEY
RESERVED	63:6	
XFRM	127:64	XSAVE Feature Request Mask. See Section 6.7.

2.6.2 Thread Control Structure (TCS)

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

Table 2-4. Layout of Thread Control Structure (TCS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
RESERVED	0	8	
FLAGS	8	8	The thread's execution flags.
OSSA	16	8	Offset of the base of the State Save Area stack, relative to the enclave base. Must be page aligned
CSSA	24	4	Current SSA slot, cleared by EADD
NSSA	28	4	Number of SSA slots.
OENTRY	32	8	Offset in enclave to which control is transferred on EENTER relative to the beginning of the enclave.
RESERVED	40	8	
OFSBASGX	48	8	When added to the base address of the enclave, produces the base address FS segment inside the enclave. Must be page aligned.
OGSBASGX	56	8	When added to the base address of the enclave, produces the base address GS segment inside the enclave. Must be page aligned.
FSLIMIT	64	4	Size to become the new FS limit in 32-bit mode
GSLIMIT	68	4	Size to become the new GS limit in 32-bit mode
RESERVED	72	4024	Must-be-zero

2.6.2.1 TCS.FLAGS

Table 2-5. Layout of TCS.FLAGS Field

Field	Bit Position	Description
DBGOPTIN	0	If set, enables debugging features (TF, breakpoints, etc.) while executing in the enclave on this TCS. Hardware clears this bit. A debugger may later modify it.
RESERVED	63:1	

2.6.2.2 State Save Area Offset (OSSA)

The OSSA points to a stack of state save frames used to save the processor state when an interrupt or exception occurs while executing in the enclave.

2.6.2.3 Number of State Save Areas (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.

2.6.2.4 Current State Save Area (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.

2.6.3 State Save Area (SSA)

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

This area contains two regions:

- The GPRSGX region. This is used to hold the processor general purpose registers (RAX ... R15), the RIP, the outside RSP and RBP, RFLAGS and the AEX information.
- The XSAVE region which contains extended feature register state in an XSAVE/FXSAVE-compatible non-compacted format.

Table 2-6. Layout of GPRSGX Portion of the State Save Area

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

2.6.3.1 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 2-7. The VALID bit is set only for the exceptions conditions which are reported inside an enclave. See Table 2-8 for which exceptions are reported inside the enclave. If the exception condition is not one reported inside the enclave then VECTOR and EXIT_TYPE are cleared.

Table 2-7. 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 Supported exceptions are #DE, #DB, #BP, #BR, #UD, #MF, #AC, #XM

2.6.3.2 VECTOR Field Definition

Table 2-8 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 2-8. Exception Vectors

Name	Vector #	Description
#DE	0	DIV and IDIV instructions
#DB	1	For Intel use only
#BP	3	INT 3 instruction
#BR	5	BOUND instruction
#UD	6	UD2 instruction and reserved opcodes
#MF	16	x87 FPU floating-point or WAIT/FWAIT instruction
#AC	17	Any data reference in memory
#XM	19	Any SIMD floating-point exceptions

2.6.4 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 2-9. Layout of PAGEINFO Data Structure

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

2.6.5 Security Information (SECINFO)

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

Table 2-10. Layout of SECINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
FLAGS	0	8	Flags describing the state of the enclave page; R/W by software
RESERVED	8	56	Must be zero;

2.6.5.1 SECINFO.FLAGS

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

Table 2-11. 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
RESERVED	7:3	Must be zero
PAGE_TYPE	15:8	The type of page that the SECINFO is associated with
RESERVED	63:16	Must be zero

2.6.5.2 PAGE_TYPE Field Definition

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

Table 2-12. Supported PAGE_TYPE

TYPE	Value	Description
PT_SECS	0	Page is an SECS
PT_TCS	1	Page is a TCS
PT_REG	2	Page is a normal page
PT_VA	3	Page is a Version Array
	All other	Reserved

2.6.6 Paging Crypto MetaData (PCMD)

The PCMD structure is used to keep track of crypto meta-data associated with a paged-out page. Combined with PAGEINFO, it provides enough information for the processor to verify, decrypt, and reload a paged-out EPC page. The size of the PCMD structure (128 bytes) is architectural. EWB writes out the reserved field and MAC values. ELDB/U reads the fields and checks the MAC.

The format of PCMD is as follows:

Table 2-13. Layout of PCMD Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
SECINFO	0	64	Flags describing the state of the enclave page; R/W by software
ENCLAVEID	64	8	ENCLAVEID

Table 2-13. Layout of PCMD Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
RESERVED	72	40	Must be zero
MAC	112	16	MAC for the page, page meta-data and reserved field

2.6.7 Enclave Signature Structure (SIGSTRUCT)

SIGSTRUCT contains information about the enclave from the enclave signer, and must be 4K-Bytes aligned.

SIGSTRUCT includes ENCLAVEHASH as SHA256 digests as defined in FIPS PUB 180-4. The digests are byte strings of length 32 with the most significant byte of each of the 8 HASH dwords at the left most byte position.

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

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

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

q1 = floor(Signature^2 / Modulus);

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

SIGSTRUCT must be page aligned

In column 5 of Table 2-14, ‘Y’ indicates that this field should be included in the signature generated by the developer.

Table 2-14. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
HEADER	0	16	Must be byte stream 06000000E10000000000010000000000H	Y
VENDOR	16	4	Intel Enclave: 00008086H Non-Intel Enclave: 00000000H	Y
DATE	20	4	Build date is yyyymmdd in hex: yyyy=4 digit year, mm=1-12, dd=1-31	Y
HEADER2	24	16	Must be byte stream 01010000600000006000000001000000H	Y
SWDEFINED	40	4	Available for software use	Y
RESERVED	44	84	Must be zero	Y
MODULUS	128	384	Module Public Key (keylength=3072 bits)	N
EXPONENT	512	4	RSA Exponent = 3	N
SIGNATURE	516	384	Signature over Header and Body	N
RESERVED	900	28	Must be zero	Y
ATTRIBUTES	928	16	Enclave Attributes that must be set	Y
ATTRIBUTEMASK	944	16	Mask of Attributes to enforce	Y
ENCLAVEHASH	960	32	MRENCLAVE of enclave this structure applies to	Y
RESERVED	992	32	Must be zero	Y
ISVPRODID	1024	2	ISV assigned Product ID	Y
ISVSVN	1026	2	ISV assigned SVN (security version number)	Y
RESERVED	1028	12	Must be zero	N

Table 2-14. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
Q1	1040	384	Q1 value for RSA Signature Verification	N
Q2	1424	384	Q2 value for RSA Signature Verification	N

2.6.8 EINIT Token Structure (EINITTOKEN)

The EINIT token is used by EINIT to verify that the enclave is permitted to launch.

EINIT token must be 512-Byte aligned.

Table 2-15. Layout of EINIT Token (EINITTOKEN)

Field	OFFSET (Bytes)	Size (Bytes)	MACed	Description
VALID	0	4	Y	Bits 0: 1: Valid; 0: Debug. All other bits reserved.
RESERVED	4	44	Y	Must be zero
ATTRIBUTES	48	16	Y	ATTRIBUTES of the Enclave
MRENCLAVE	64	32	Y	MRENCLAVE of the Enclave
RESERVED	96	32	Y	Reserved
MRSIGNER	128	32	Y	MRSIGNER of the Enclave
RESERVED	160	32	Y	Reserved
CPUSVNLE	192	16	N	Launch Enclave's CPUSVN
ISVPRODIDLE	208	02	N	Launch Enclave's ISVPRODID
ISVSVNLE	210	02	N	Launch Enclave's ISVSVN
RESERVED	212	28	N	Reserved
MASKEDATTRIBUTESLE	240	16	N	MASKEDATTRIBUTES of Launch Enclave. This should be set to the LE's ATTRIBUTES masked with MASK the LE's KEYREQUEST.
KEYID	256	32	N	Value for key wear-out protection
MAC	288	16	N	A cryptographic MAC on EINITTOKEN using Launch key

2.6.9 Report (REPORT)

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

Table 2-16. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CPUSVN	0	16	The security version number of the processor.
RESERVED	16	32	Must be zero
ATTRIBUTES	48	16	The values of the attributes flags for the enclave. See Section 2.6.1.1 (ATTRIBUTES Bits) for the definitions of these flags.
MRENCLAVE	64	32	The value of SECS.MRENCLAVE
RESERVED	96	32	Reserved
MRSIGNER	128	32	The value of SECS.MRSIGNER
RESERVED	160	96	Zero
ISVPRODID	256	02	Enclave PRODUCT ID
ISVSVN	258	02	The security version number of the Enclave
RESERVED	260	60	Zero

Table 2-16. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
REPORTDATA	320	64	A set of data used for communication between the enclave and the target enclave.This value is provided by the EREPORT call in RCX.
KEYID	384	32	Value for key wear-out protection
MAC	416	16	The CMAC on the report using report key

2.6.9.1 REPORTDATA

The REPORTDATA structure is specifies the address of a 64-Byte input buffer that the EREPORT instruction will use to generate cryptographic report. It requires 128-Byte alignment.

2.6.10 Report Target Info (TARGETINFO)

This structure is an input parameter to the EREPORT instruction leaf. The address of TARGETINFO is specified as an effective address in RBX. It is used to identify the enclave which will be able to cryptographically verify the REPORT structure returned by EREPORT. A TARGETINFO requires 128-Byte alignment.

Table 2-17. 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

2.6.11 Key Request (KEYREQUEST)

This structure is an input parameter to the EGETKEY instruction. It is passed in as an effective address in RBX and requires 128-Byte alignment. It is used for selecting the appropriate key and any additional parameters required in the derivation of that key.

Table 2-18. Layout of KEYREQUEST Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	02	Identifies the Key Required
KEYPOLICY	02	02	Identifies which inputs are required to be used in the key derivation
ISVSVN	04	02	The ISV security version number used in the key derivation
RESERVED	06	02	Must be zero
CPUSVN	08	16	The security version number of the processor used in the key derivation
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in the derivation of the Seal Key
KEYID	40	32	Value for key wear-out protection

2.6.11.1 KEY REQUEST KeyNames

Table 2-19. Supported KEYName Values

Key Name	Value	Description
LAUNCH_KEY	0	Launch key
PROVISION_KEY	1	Provisioning Key
PROVISION_SEAL_KEY	2	Provisioning Seal Key

Table 2-19. Supported KEYName Values

Key Name	Value	Description
REPORT_KEY	3	Report Key
SEAL_KEY	4	Report Key
	All other	Reserved

2.6.11.2 Key Request Policy Structure

Table 2-20. Layout of KEYPOLICY Field

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register
RESERVED	15:2	Must be zero

2.6.12 Version Array (VA)

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

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

Table 2-21. Layout of Version Array Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
Slot 0	0	08	Version Slot 0
Slot 1	8	08	Version Slot 1
...			
Slot 511	4088	08	Version Slot 511

2.6.13 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 are implementation specific.

Table 2-22. Content of Enclave Page Cache Map

Field	Description
VALID	Indicates whether the EPCM entry is valid
R	Read access; indicates whether enclave accesses are allowed for the EPC page
W	Write access; indicates whether enclave accesses are allowed for the EPC page
X	Execute access; indicates whether enclave accesses are allowed for the EPC page
PT	EPCM page type (PT_SECS, PT_TCS, PT_REG, PT_VA)
ENCLAVESECS	SECS identifier of the enclave to which the page belongs

Table 2-22. Content of Enclave Page Cache Map

Field	Description
ENCLAVEADDRESS	Linear enclave address of the page
BLOCKED	Indicates whether the page is in the blocked state

This page was intentionally left blank.

CHAPTER 3
ENCLAVE OPERATION

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

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

3.1 CONSTRUCTING AN ENCLAVE

Figure 3-1 illustrates a typical Enclave memory layout.

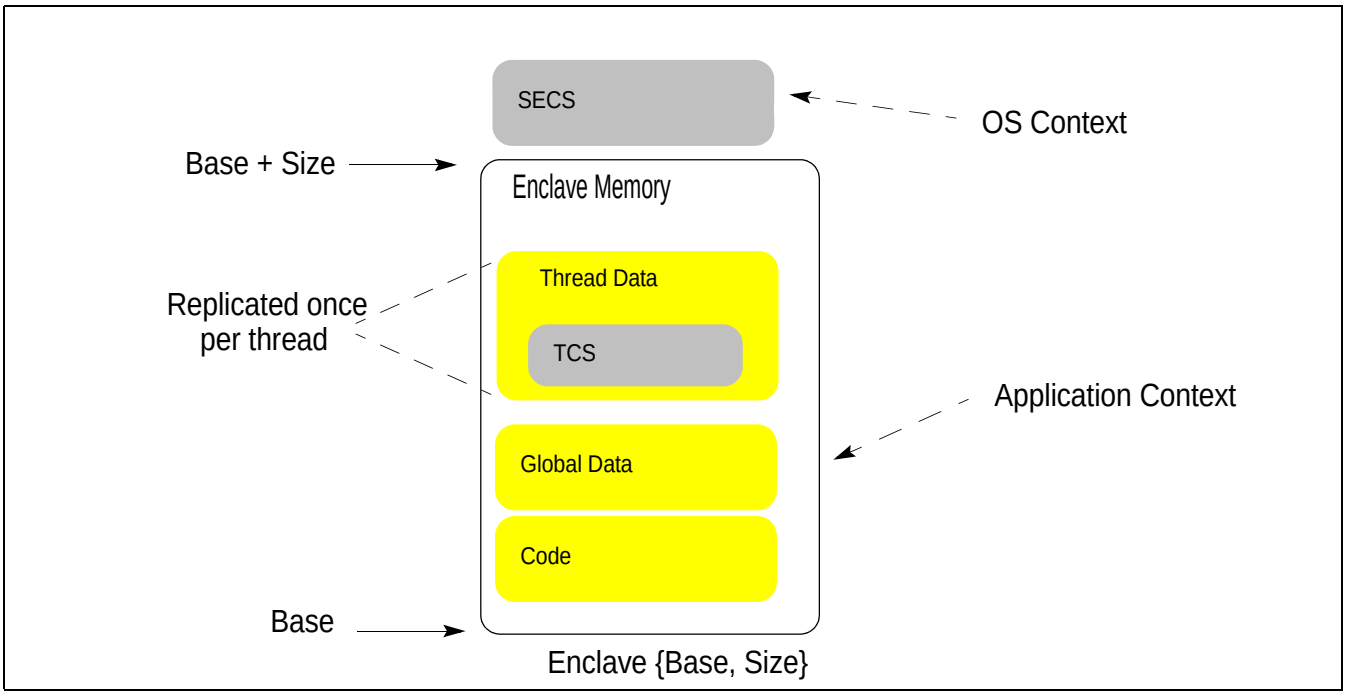

Figure 3-1. Enclave Memory Layout

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

1. The application hands over the enclave content along with additional information required by the enclave creation API to the enclave creation service running at ring-0.
2. Use the ECREATE leaf to set up the initial environment, specifying base address and size of the enclave. This address range, the ELRANGE, is part of the application's address space. This reserves the memory range. The enclave will now reside in this address region. ECREATE also allocates an Enclave Page Cache (EPC) page for

the SGX Enclave Control Structure (SECS). Note that this page is not required to be a part of the enclave linear address space and is not required to be mapped into the process.

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

3.1.1 EADD and EEXTEND Interaction

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

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

System software is responsible for selecting a free EPC page. System software is also responsible for providing the type of page to be added, the attributes the page, the contents of the page, and the SECS (enclave) to which the page is to be added as requested by the application.

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

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

Examples of incorrect construction includes adding multiple pages with the same enclave linear address resulting in an alias, loading modified contents into an enclave page, or not measuring all of the enclave.

3.1.2 EINIT Interaction

Once system software has completed the process of adding and measuring pages, the enclave needs to be initialized by the EINIT instruction. Initializing an enclave prevents the addition or measurement of enclave pages and enables enclave entry. The initialization process finalizes the cryptographic log and establishes the enclave identity and sealing identity used by EGETKEY and EREPORT.

A cryptographic hash of the log is stored. Correct construction results in the cryptographic log matching the one built by the enclave owner in SIGSTRUCT. It can be verified by a remote party.

The enclave is initialized by the EINIT instruction. The EINIT instruction checks the ENIT token to validate that the enclave has been enabled on this platform. If the enclave is not correctly constructed or the EINIT token is not valid for the platform then EINIT will fail. See the EINIT instruction for details on the error reporting.

The enclave identity is a cryptographic hash that reflects the content of the enclave, the order in which it was built, the addresses it occupies in memory, and the security attributes of each page. The Enclave Identity is established by EINIT.

The sealing identity is managed by a sealing authority represented by the hash of a public key used to sign a structure processed by EINIT. The sealing authority assigns a product ID and security version number to a particular enclave identity comprising the attributes of the enclave and the measurement of the enclave.

EINIT establishes the sealing identity using the following steps:

1. Verifies that SIGSTRUCT is signed using the public key enclosed in the SIGSTRUCT

2. Checks that the measurement of the enclave matches the measurement of the enclave specified in SIGSTRUCT

3. Checks that the enclave's attributes are compatible with those specified in SIGSTRUCT

4. Finalizes the measurement of the enclave and records the sealing identity and enclave identity (the sealing authority, product id and security version number) in the SECS

3.2 ENCLAVE ENTRY AND EXITING

3.2.1 Synchronous Entry and Exit

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

When a logical processor enters an enclave, the TCS is considered busy until the logical processors exits the enclave. SGX allows an enclave builder to define multiple TCSs, thereby providing support for multithreaded enclaves.

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

EENTER performs the following operations:

1. Check that TCS is not busy and flush TLB entries for enclave linear addresses in the enclave's ELRANGE.
2. Change the mode of operation to be in enclave mode.
3. Save the RSP, RBP for later restore on AEX.
4. Save XCR0 and replace it with the XFRM value for the enclave.
5. Check if the enclave is debuggable and the software wishes to debug. If not then set hardware so the enclave appears as a single instruction.
6. If the enclave is debuggable and the software wishes to debug, then set hardware to allow traps, breakpoints, and single steps inside the enclave.
7. Set the TCS as busy.
8. Transfer control from outside enclave to predetermined location inside the enclave specified by the TCS.

The EEXIT instruction is the method of leaving the enclave under program control, it performs the following operations:

1. Clear enclave mode and TLB entries for enclave addresses.
2. Mark TCS as not busy.
3. Transfer control from inside the enclave to a location on the outside specified by the register, RBX.

It is the responsibility of enclave software to erase any secret from the registers prior to invoking EEXIT.

3.2.2 Asynchronous Enclave Exit (AEX)

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

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

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

3.2.3 Resuming Execution after AEX

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

If the cause of the exit was an exception or a fault and was not resolved, the event will be triggered again if the enclave is re-entered using ERESUME. For example, if an enclave performs a divide by 0 operation, executing ERESUME will cause the enclave to attempt to re-execute the faulting instruction and result in another divide by 0 exception. In order to handle an exception that occurred inside the enclave, software can enter the enclave at a different location and invoke the exception handler within the enclave by executing the EENTER instruction. The exception handler within the enclave can attempt to resolve the faulting condition or simply return and indicate to software that the enclave should be terminated (e.g. using EEXIT).

3.2.3.1 ERESUME Interaction

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

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

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

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

3.3 CALLING ENCLAVE PROCEDURES

3.3.1 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.

3.3.2 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 opera-

tions that need be performed. It has the additional security result that it ensures that data is scrubbed from any registers that were used to temporarily contain secrets.

3.3.3 Returning to Caller

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

3.4 SGX KEY AND ATTESTATION

To provide cryptographic separation between platforms, SGX provides individual keys to each platform.

Each processor is provisioned with a unique key as the root of the key hierarchy. This is done at manufacturing time. This key is the basis for all keys derived in the EGETKEY instruction. Figure 3-2 shows the hierarchy used to generate keys on the platform.

Each enclave requests keys using the EGETKEY instruction. The key is based on enclave parameters such as measurement or the enclave signing key plus the key derived from the device key and various security version numbers (SVNs). See the EGETKEY instruction for more details.

In order for a remote party to understand the security level of a remote platform, security version numbers are designed into the SGX architecture. Some of the version numbers indicate the patch level of the relevant phases of the processor boot up and system operations that affect the identity of the SGX instructions.

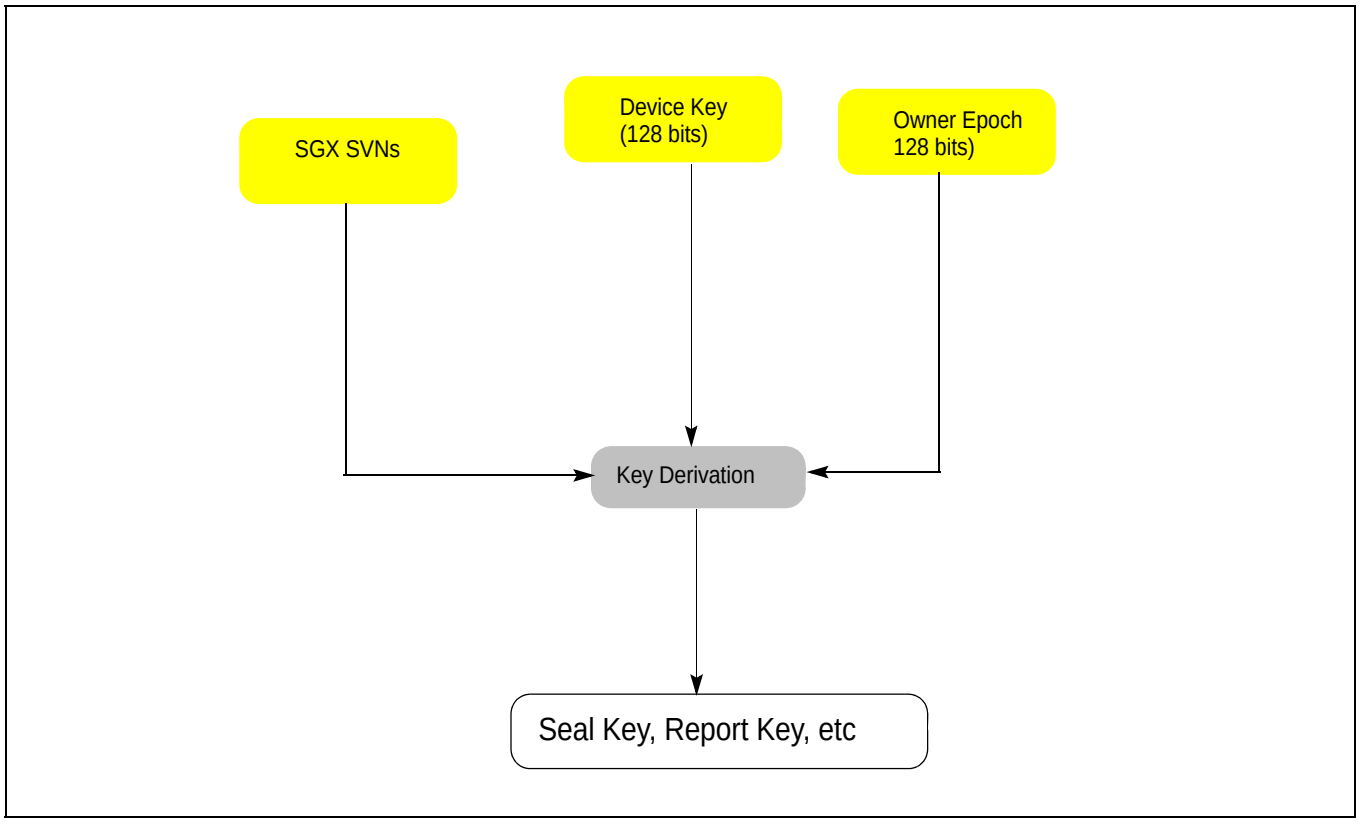

Figure 3-2. SGX Key Overview

The SVN values are reported to the remote user as part of the attestation process. They are part of the EREPORT instruction output.

Owner Epoch is a 128 bit value which is loaded into the SGXOWNEREPOCH0 and SGXOWNEREPOCH1 MSRs when SGX is booted. These registers provide a user with the ability to add personal entropy into the key derivation process.

NOTE: Owner Epoch must be kept the same in order to decrypt data using the EGETKEY instruction. A different Owner Epoch will result in the failure to decrypt files sealed by EGETKEY in a previous boot.

3.5 EPC AND MANAGEMENT OF EPC PAGES

EPC layout is implementation specific, and is enumerated through CPUID (see Table 1-5 for EPC layout). EPC is typically configured by BIOS at system boot time.

A processor that supports SGX (CPUID.(EAX=07H, ECX=0):EBX.SGX= 1) supports the ability for the BIOS to reserve a range of memory called Processor Reserved Memory (PRM). PRM must have a size that is integer power of two, and must be naturally aligned. The BIOS allocates the PRM by setting a pair of MSRs, collectively known as the PRMRR. The exact layout of the PRM and EPC are model-specific, and depend on BIOS settings. Figure 3-3 depicts a conceptual example of the layout of the PRM and available EPC section(s).

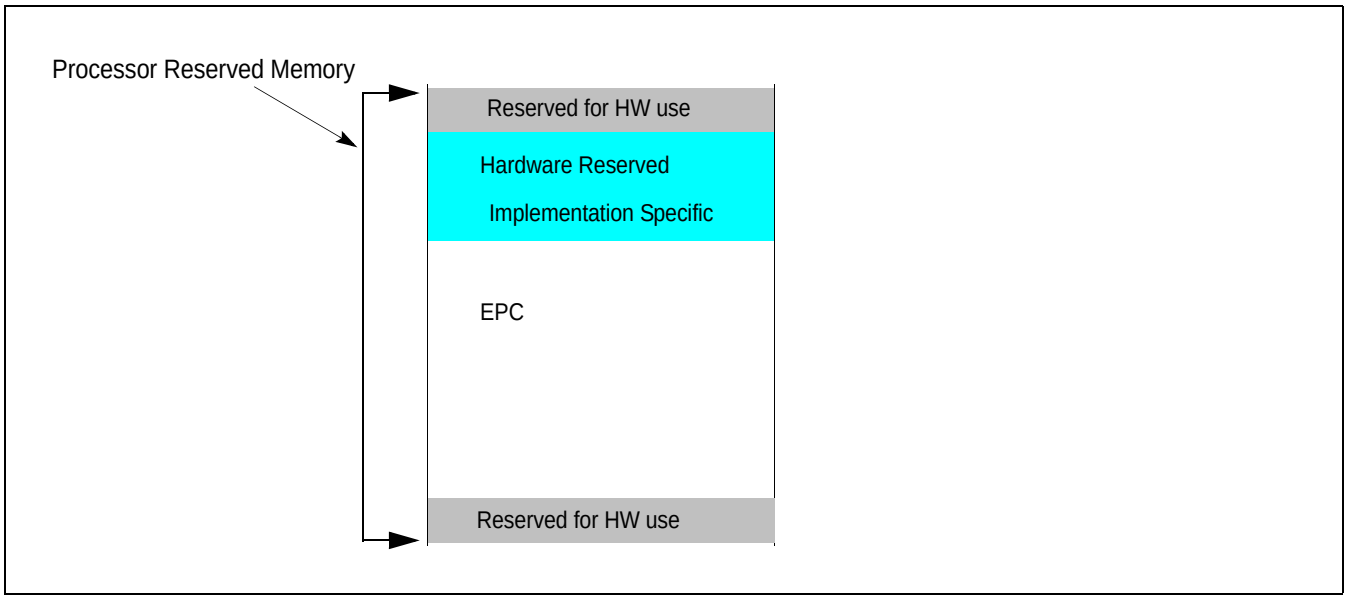

Figure 3-3. Conceptual Layout of Processor Reserved Memory and EPC

3.5.1 EPC Implementation

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

3.5.2 OS Management of EPC Pages

The EPC is a finite resource. To oversubscribe the EPC the EPC manager must keep track of all EPC entries, type and state, context affiliation, SECS affiliation, so that it could manage this resource and properly swap pages out of and into the EPC.

SGX includes the EWB instruction for securely evicting pages out of the EPC. EWB encrypts a page in the EPC and writes it to unprotected memory. In addition, EWB also creates a cryptographic MAC of the page and stores it in unprotected memory. A page can be reloaded back to the processor only if the data and MAC match.

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

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

3.5.3 Eviction of Enclave Pages

SGX paging is optimized to allow the OS to page out multiple EPC pages under a single synchronization.

1. For each enclave page to be evicted:
   a. Select a slot in a Version Array page.
      - If no VA page exists with an empty slot, create a new PT_VA page using the EPA instruction.
   b. Remove mapping from the page table (OS removes from system page table, VMM removes from EPT).
   c. Execute EBLOCK for the target page. This sets the target page state to BLOCKED. At this point no new mappings of the page will be created. Accesses which do not have mapping cached in the TLB will generate a #PF.
2. For each enclave containing pages selected in step 1:
   — Execute an ETRACK on that enclave.
3. For all hardware threads executing in processes (OS) or guests (VMM) that contain the enclaves selected in step 1:
   — Issue an IPI (inter-processor interrupt) to those threads. This causes those hardware threads to exit any enclaves they might be in, and as a result flush all TLB entries that might hold stale translations to blocked pages.
4. After enclaves exit, allow h/w threads to execute normally.
5. For each page to be evicted:
   — Evict the page using the EWB command. Parameters include the EPC page linear address (the OS or VMM needs to use its own, private page mapping for this because of step 1.c), the VA slot, a 4k byte buffer to hold the encrypted page contents, and a buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

At this point, system software has an encrypted copy of each page data and page metadata, both in main memory.

3.5.4 Loading an Enclave Page

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

1. Execute ELDB/ELDU, passing as parameters: the EPC page linear address (again, using a private mapping), the VA slot, the encrypted page, and the page metadata.
2. Create a mapping in the page table to allow the application to access that page (OS: system page table; VMM: EPT).

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

3.5.5 Eviction of an SECS Page

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

1. Ensure all pages are evicted from enclave.
2. Select a slot in a Version Array page.
   — If no VA page exists with an empty slot, create a new one using EPA.
3. Evict the page using the EWB command. Parameters include the EPC page effective address, the VA slot, a 4k byte buffer to hold the encrypted page contents and a buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

3.5.6 Eviction of a Version Array Page

VA pages do not belong to any enclave. When evicting the VA page, a slot in a different VA page must be specified in order to provide versioning of the evicted VA page.

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

3.6 CHANGES TO INSTRUCTION BEHAVIOR INSIDE AN ENCLAVE

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

3.6.1 Illegal Instructions

The instructions listed in Table 3-1 are ring 3 instructions which become illegal when executed inside an enclave. Executing these instructions inside an enclave will generate a #UD fault.

The first row of Table 3-1 enumerates instructions that may cause a VM exit for VMM emulation. Since a VMM cannot emulate enclave execution, execution of any these instructions inside an enclave results in an invalid-opcode exception (#UD) and no VM exit.

The second row of Table 3-1 enumerates I/O instructions that may cause a fault or a VM exit for emulation. Again, enclave execution cannot be emulated, so execution of any these instructions inside an enclave results in #UD.

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

Table 3-1. Illegal Instructions Inside an Enclave

Instructions	Result	Comment
CPUID, GETSEC, RDPMC, RDTSC, RDTSCP, 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.

Table 3-1. Illegal Instructions Inside an Enclave

Instructions	Result	Comment
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.
ENCLU[EENTER], ENCLU[ERESUME]	#GP	Cannot enter an enclave from within an enclave.

3.6.2 RDRAND and RDSEED Instructions

These instructions may cause a VM exit if the "RDRAND exiting" VM-execution control is 1. Unlike other instructions that can cause VM exits, these instructions are legal inside an enclave. As noted in Section 6.5.5, any VM exit originating on an instruction boundary inside an enclave sets bit 27 of the exit-reason field of the VMCS. If a VMM receives a VM exit due to an attempt to execute either of these instructions determines (by that bit) that the execution was inside an enclave, it can do either of two things. It can clear the "RDRAND exiting" VM-execution control and execute VMRESUME; this will result in the enclave executing RDRAND or RDSEED again, and this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

NOTE

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

3.6.3 PAUSE Instruction

The PAUSE instruction may cause a VM exit if either of the "PAUSE exiting" and "PAUSE-loop exiting" VM-execution controls is 1. Unlike other instructions that can cause VM exits, the PAUSE instruction is legal inside an enclave.

If a VMM receives a VM exit due to the 1-setting of "PAUSE-loop exiting", it may take action to prevent recurrence of the PAUSE loop (e.g., by scheduling another virtual CPU of this virtual machine) and then execute VMRESUME; this will result in the enclave executing PAUSE again, but this time the PAUSE loop (and resulting VM exit) will not occur.

If a VMM receives a VM exit due to the 1-setting of "PAUSE exiting", it can do either of two things. It can clear the "PAUSE exiting" VM-execution control and execute VMRESUME; this will result in the enclave executing PAUSE again, but this time a VM exit will not occur. Alternatively, the VMM might choose to discontinue execution of this virtual machine.

NOTE

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

3.6.4 INT 3 Behavior Inside an Enclave

INT3 is legal inside an enclave, however, the behavior inside an enclave is different from its behavior outside an enclave. See Section 7.4.1 for details

3.6.5 INVD Handling when Enclaves Are Enabled

Once processor reserved memory is available (see Section 3.5), any execution of INVD will result in a #GP(0).

This page was intentionally left blank.

CHAPTER 4
ENCLAVE EXITING EVENTS

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

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

4.1 COMPATIBLE SWITCH TO THE EXITING STACK OF AEX

Asynchronous enclave exits push information onto the appropriate stack in a form expected by the operating system. To accomplish this, an address to trampoline code is pushed onto the exiting stack as the RIP. This trampoline code eventually returns to the enclave by means of an ENCLU(ERESUME) instruction.

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

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

In all cases, the choice of exit stack and the information pushed onto it is consistent with non-SGX operation. Figure 4-1 shows the Application and Exiting Stacks after an exit with a stack switch. An exit without a stack switch uses the Application Stack. The ERESUME leaf index is placed into RAX, the TCS pointer is placed in RBX and the AEP (see below) is placed into RCX for later use when resuming the enclave after the exit.

Upon an AEX, the AEP (Asynchronous Exit Pointer) is pushed onto the exit stack as the return RIP. The AEP points to a trampoline code sequence which includes the ERESUME instruction that is later used to reenter the enclave.

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

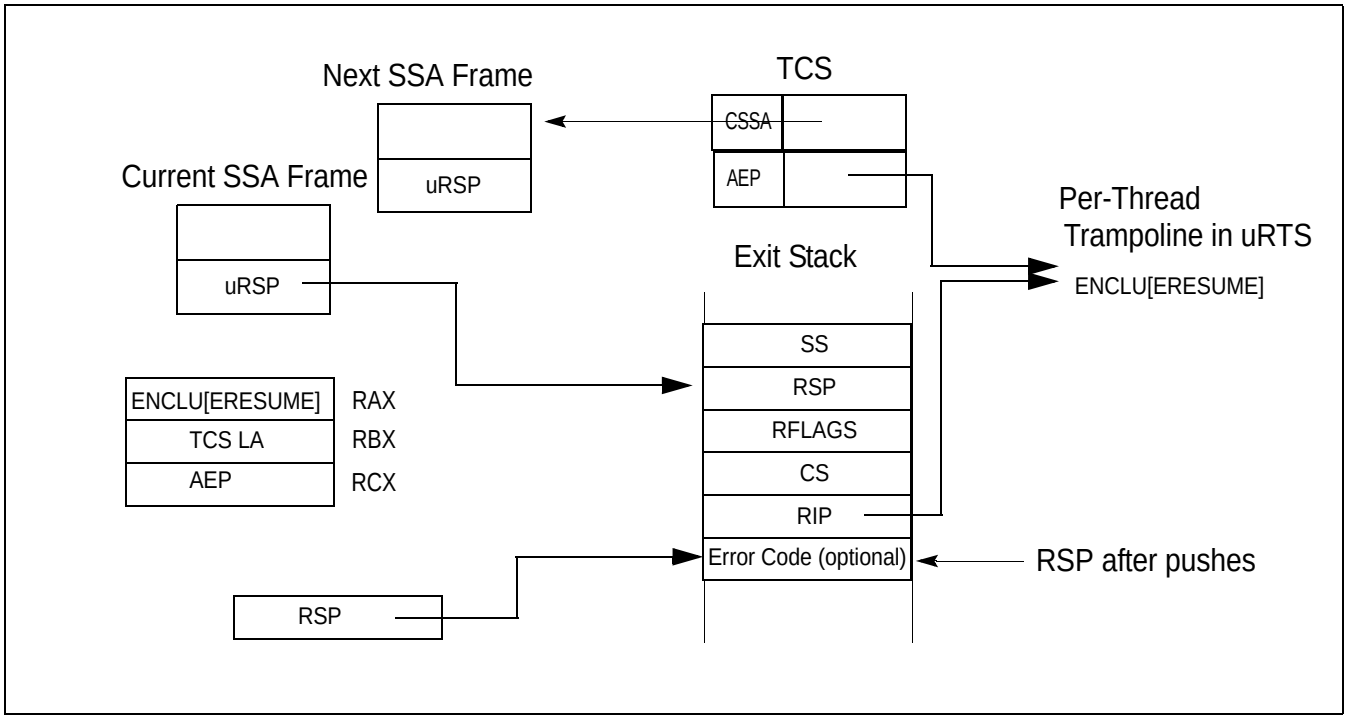

Figure 4-1. Exit Stack Just After Interrupt with Stack Switch

4.2 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 4-2.

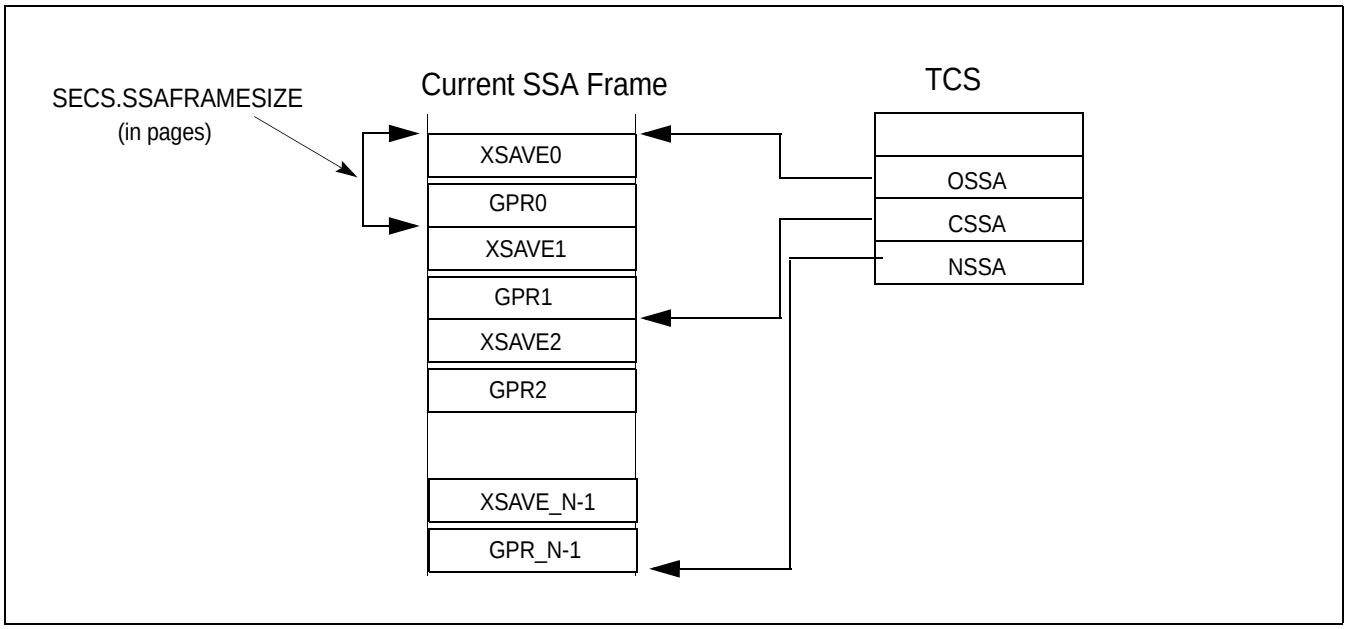

Figure 4-2. The SSA Stack

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

Number of State Save Area Slots (NSSA). Defines the total number of slots in the State Save Area stack.

Current State Save Area Slot (CSSA). Defines the current slot to use on the next exit.

State Save Area (OSSA). Set of save area slots large enough to hold the GPR state and the XSAVE state.

When an AEX occurs while executing on a thread inside the enclave, hardware selects the SSA frame to use by examining TCS.CSSA. Processor state (as described in Section 4.3.1) is saved and loaded with a synthetic state (to avoid leaking secrets) and TCS.CSSA is incremented. As will be described later, if an exception takes the last slot, it will not be possible to reenter the enclave to handle the exception inside the enclave.

The format of the XSAVE section of SSA is identical to the format used by the XSAVE/XRSTOR instructions.

Note: On EENTER, CSSA must be less than NSSA, ensuring that there is at least one Save Area available for exits.

Multiple SSA frames are defined to allow for a variety of behavior. When an AEX occurs the SSA frame is loaded and the pointer incremented. An ERESUME restores the processor state and frees the SSA frame. If after the AEX an EENTER is executed then the next SSA frame is reserved to hold state for another AEX. If there is no free SSA frame when executing EENTER, the entry will fail.

4.3 SYNTHETIC STATE ON ASYNCHRONOUS ENCLAVE EXIT

4.3.1 Processor Synthetic State on Asynchronous Enclave Exit

Table 4-1 shows the synthetic state loaded on AEX. The values written are the lower 32 bits when the processor is in 32 bit mode and 64 bits when the processor is in 64 bit mode.

Table 4-1. GPR, x87 Synthetic States on Asynchronous Enclave Exit

Register	Value
RAX	3 (ENCLU[3] is ERESUME)
RBX	TCS pointer of interrupted enclave thread
RCX	AEP of interrupted enclave thread
RDX, RSI, RDI	0
RSP	Loaded from SSA.uRSP
RBP	Loaded from SSA.uRBP
R8-R15	0 in 64-bit mode; unchanged in 32-bit mode
RIP	AEP of interrupted enclave thread
RFLAGS	CF, PF, AF, ZF, SF, OF, RF bits are cleared. Remaining bits are left unchanged.
x87/SSE State	Unless otherwise listed here, all x87 and SSE state are set to the INIT state. The INIT state is the state that would be loaded by the XRSTOR instruction with bits 1:0 both set in the instruction mask and XCR0, and both clear in XSTATE_BV the XSAVE header.
FCW	On #MF exception: 037EH. On all other exits: 037FH
FSW	On #MF exception: 8081H. On all other exits: 0H
CR2	If the event that caused the AEX is a #PF, and the #PF does not directly cause a VM exit, then the low 12 bits are cleared. If the #PF leads directly to a VM exit, CR2 is not updated (usual IA behavior). Note: The low 12 bits are not cleared if a #PF is encountered during the delivery of the EEE that caused the AEX. This is because it is the AEX that clears those bits, and EEE delivery occurs after AEX. Also, the address of an access causing a #PF during EEE delivery reveals no enclave secrets.
FS, GS	Including hidden portion. Restored to values as of most recent EENTER/ERESUME
DR7	If the last entry was an opt-out entry, restored to value as of most recent EENTER/ERESUME
IA32_DEBUG_CTL	If the last entry was an opt-out entry, restored to value as of most recent EENTER/ERESUME

Table 4-1. GPR, x87 Synthetic States on Asynchronous Enclave Exit

Register	Value
IA32_PERF_GLOBAL_CTRL	If the last entry was an opt-out entry, restored to value as of most recent EENTER/ERESUME
IA32_PEBS_ENABLE	If the last entry was an opt-out entry, restored to value as of most recent EENTER/ERESUME

4.3.2 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.)

4.3.3 VMCS Synthetic State on Asynchronous Enclave Exit

All processor registers saved in the VMCS have the same synthetic values listed above. Additional VMCS fields that are treated specially on VM exit are listed in Table 4-2

Table 4-2. VMCS Synthetic States on Asynchronous Enclave Exit

Field	Value
ENCLAVE_INTERRUPTION	A new bit (bit 4 in the "Guest Interruptibility State" state field). Set to 1 if exit occurred in enclave mode
Guest-linear address	If the event that caused the AEX is an EPT violation that sets bit 7 of the Exit-Qualification field, the low 12 bits are cleared. Note: If the EPT violation occurs during delivery of an event that caused the AEX (e.g., an EPT violation that occurs during IDT vectoring), then the low 12 bits are NOT cleared.
Guest-physical address	If the event that caused the AEX is an EPT violation or mis-configured EPT, then the low 12 bits are cleared. Note: If the EPT violation or misconfiguration occurs during delivery of an event that caused the AEX (e.g., an EPT violation or misconfiguration that occurs during IDT vectoring), then the low 12 bits are NOT cleared.
Exit-Qualification	On page-fault that causes an AEX: low 12 bits are cleared On APIC-access that causes an AEX: low 12 bits are cleared Note: If either the page-fault or APIC-access occurs during delivery of an event that caused the AEX, the low 12 bits are NOT cleared. On EPT violation that causes an AEX, bit 9 will be set if the access was to the EPC, clear on non-EPC access.
VM-exit instruction length	Cleared.
VM-exit instruction information	This field is defined only for VM exits due to or during the execution of specific instructions (i.e. should be reported properly). Most of these instructions do not cause VM exits when executed inside an enclave. Exceptions are INVEPT, INVVPID, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON. Normally, this field is defined for VM exits due to INT3 (or exceptions encountered while delivering INT3). This is not true for INT3 in an enclave, as the instruction becomes fault-like. INT3 Interruption types are reported as hardware exception when invoked inside enclave instead of 6 respectively when invoked outside enclave. This field is cleared for all other VM exits
I/O RCX	Cleared
I/O RSI	Cleared
I/O RDI	Cleared
I/O RIP	Cleared

4.4 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
2. Synthetic state is created for a number of processor registers to present an opaque view of the enclave state. Table 4-1 shows the values for GPRs, x87, SSE, FS, GS, Debug and performance monitoring on AEX. The synthetic state for other extended features (those controlled by XCR0[62:2]) is set to their respective INIT states when the corresponding bit of SECS.ATTRIBUTES.XFRM is set. The INIT state is that state as defined by the behavior of the XRSTOR instruction when HEADER.XSTATE_BV[n] is 0. In addition, on VM exit the VMCS or SMRAM state is initialized as described in Table 4-2.
3. In the current SSA frame, the cause of the AEX is saved in the EXITINFO field. See Table 2-7 for details and values of the various fields.
4. Any code and data breakpoints that were suppressed at the time of enclave entry are unsuppressed when exiting the enclave.
5. RFLAGS.TF is set to the value that it had at the time of the most recent enclave entry (an exception is made if that entry was opt-in; see Section 7.2). In the SSA, RFLAGS.TF is set to 0. However, due to the way TF is handled on enclave entry, this value is irrelevant (see EENTER and ERESUME instructions).
6. RFLAGS.RF is set to 0 in the synthetic state. In the SSA, the value saved is the same as what would have been saved on stack in the non-SGX case (architectural value of RF). Thus, AEXs due to interrupts, traps, and code breakpoints save RF unmodified into SSA, while AEXs due to other faults save RF as 1 in the SSA.

   If the event causing AEX happened on intermediate iteration of a REP-prefixed instruction, then RF=1 is saved on SSA, irrespective of its priority.
7. Any performance monitoring activity (including PEBS) on the exiting thread that was suppressed due to the enclave entry on that thread is unsuppressed. Any counting that had been demoted to MyThread (on other threads) is promoted back to AnyThread.

4.4.1 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 5-4
        SSA.RFLAGS.TF ← 0;
   ELSE    (* TMP_MODE64 = 1 *)
        Save RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8-R15, RFLAGS, RIP into SSA using CR_GPR_PA
        SSA.RFLAGS.TF ← 0;
FI;

(* Use a special version of XSAVE that takes a list of physical addresses of logically sequential pages to
   perform the save. TMP_MODE64 specifies whether to use the 32-bit or 64-bit layout.
   SECS.ATTRIBUTES.XFRM selects the features to be saved.
   CR_XSAVE_PAGE_n specifies a list of 1 or more physical addresses of pages that contain the XSAVE area. *)
XSAVE(TMP_MODE64, SECS.ATTRIBUTES.XFRM, CR_XSAVE_PAGE_n);

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

CR_XSAVE_PAGE_0.XHEADER_BV[191:64] ← 0;

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

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

(* Restore the outside RSP and RBP from the current SSA frame.
   This is where they had been stored on most recent EENTER *)
RSP ← CR_GPR_PA.URSP;
RBP ← CR_GPR_PA.URBP;

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

(* Get exception code from hardware *)
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
      CR_GPR_PA.EXITINFO.VECTOR ← 0;
      CR_GPR_PA.EXITINFO.EXIT_TYPE ← 0
      CR_GPR_PA.REASON.VALID ← 0;
FI;

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

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

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

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

(* Update the SSA frame # *)
CR_TCS_PA.CSSA ← CR_TCS_PA.CSSA + 1;

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

Un-suppress all code breakpoints that are outside ELRANGE

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 LBR generation;
      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;

      (* Create near relative jump record and update appropriate areas *)
      Complete LBR data observable by non-enclave code
   ELSE
      (* Create near relative jump record and update appropriate areas *)
      Complete LBR data observable by non-enclave code
FI;
```

```
IF (VMCS.MTF = 1)
    THEN Pend MTF VM Exit at the end of exit; FI;

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

(* Update the thread context to show not in enclave mode, no valid TCS *)
CR_ENCLAVE_MODE ← 0;

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

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

CHAPTER 5
INSTRUCTION REFERENCES

Supervisor and user level instructions provided by Intel Software Guard Extensions are described in this chapter. In general, a various functionalities are encoded as leaf functions within the ENCLS (supervisor) and ENCLU (user) instruction mnemonics. Different leaf functions are encoded by specifying an input value in the EAX register of the respective instruction mnemonic.

5.1 SGX INSTRUCTION SYNTAX AND OPERATION

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

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

5.1.1 ENCLS Register Usage Summary

Table 5-1 summarizes the implicit register usage of supervisor mode enclave instructions.

Table 5-1. Register Usage of Privileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
ECREATE	00H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EADD	01H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EINIT	02H (In)	SIGSTRUCT (In, EA)	SECS (In, EA)	EINITTOKEN (In, EA)
EREMOVE	03H (In)		EPCPAGE (In, EA)	
EDBGRD	04H (In)	Result Data (Out)	EPCPAGE (In, EA)	
EDBGWR	05H (In)	Source Data (In)	EPCPAGE (In, EA)	
EEXTEND	06H (In)		EPCPAGE (In, EA)	
ELDB	07H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDU	08H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
EBLOCK	09H (In)		EPCPAGE (In, EA)	
EPA	0AH (In)	PT_VA (In)	EPCPAGE (In, EA)	
EWB	0BH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ETRACK	0CH (In)		EPCPAGE (In, EA)	
EA: Effective Address				

5.1.2 ENCLU Register Usage Summary

Table 5-2 Summarized the implicit register usage of user mode enclave instructions.

Table 5-2. 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)	

Table 5-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
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)	
EA: Effective Address				

5.1.3 Information and Error Codes

Information and error codes are reported by various instruction leaf functions to show an abnormal termination of the instruction or provide information which may be useful to the developer. Table 5-3 shows the various codes and the instruction which generated the code. Details of the meaning of the code is provided in the individual instruction.

Table 5-3. Error or Information Codes for SGX Instructions

Name	Value	Returned By
No Error	0	
SGX_INVALID_SIG_STRUCT	1	EINIT
SGX_INVALID_ATTRIBUTE	2	EINIT, EGETKEY
SGX_BLSTATE	3	EBLOCK
SGX_INVALID_MEASUREMENT	4	EINIT
SGX_NOTBLOCKABLE	5	EBLOCK
SGX_PG_INVLD	6	EBLOCK
SGX_LOCKFAIL	7	EBLOCK
SGX_INVALID_SIGNATURE	8	EINIT
SGX_MAC_COMPARE_FAIL	9	ELDB, ELDU
SGX_PAGE_NOT_BLOCKED	10	EWB
SGX_NOT_TRACKED	11	EWB
SGX_VA_SLOT_OCCUPIED	12	EWB
SGX_CHILD_PRESENT	13	EWB, EREMOVE
SGX_ENCLAVE_ACT	14	EREMOVE
SGX_ENTRYEPOCH_LOCKED	15	EBLOCK
SGX_INVALID_EINIT_TOKEN	16	EINIT
SGX_PREV_TRK_INCMPL	17	ETRACK
SGX_PG_IS_SECS	18	EBLOCK
SGX_INVALID_CPUSVN	32	EINIT, EGETKEY
SGX_INVALID_ISVSVN	64	EGETKEY
SGX_UNMASKED_EVENT	128	EINIT
SGX_INVALID_KEYNAME	256	EGETKEY

5.1.4 Internal CREGs

The CREGs as shown in Table 5-4 are hardware specific registers used in this document to indicate values kept by the processor. These values are used while executing in enclave mode or while executing an SGX instruction. These

registers are not software visible and are implementation specific. The values in Table 5-4 appear at various places in the pseudo-code of this document. They are used to enhance understanding of the operations.

Table 5-4. List of Internal CREG

Name	Size (Bits)	Scope
CR_ENCLAVE_MODE	1	LP
CR_TCS_LA	64	LP
CR_TCS_PH	64	LP
CR_ACTIVE_SECS	64	LP
CR_ELRANGE	128	LP
CR_SAVE_TF	1	LP
CR_SAVE_FS	64	LP
CR_GPR_PA	64	LP
CR_XSAVE_PAGE_n	64	LP
CR_SAVE_DR7	64	LP
CR_SAVE_PERF_GLOBAL_CTRL	64	LP
CR_SAVE_DEBUGCTL	64	LP
CR_SAVE_PEBS_ENABLE	64	LP
CR_CPUSVN	128	PACKAGE
CSR_SGX_OWNEREPOCH	128	PACKAGE
CSR_INTELPUBKEYHASH	32	PACKAGE
CR_SAVE_XCR0	64	LP
CR_SGX_ATTRIBUTES_MASK	128	LP
CR_PAGING_VERSION	64	PACKAGE
CR_VERSION_THRESHOLD	64	PACKAGE
CR_NEXT_EID	64	PACKAGE
CR_BASE_PK	128	PACKAGE
CR_SEAL_FUSES	128	PACKAGE

5.1.5 Concurrent Operation Restrictions

To protect the integrity of SGX data structures, under certain conditions, SGX disallows certain leaf functions from operating concurrently. Listed below are some examples of concurrency that are not allowed.

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

When disallowed operation is detected, a leaf function causes an exception. To prevent such exceptions, software must serialize leaf functions or prevent these leaf functions from accessing the same resource.

5.1.5.1 Concurrency Restrictions

Table 5-5 summarizes which pair of instructions are allowed to operate concurrently on the same enclave. The concurrency restriction depends on the type of EPC page and the parameter of the two concurrent instructions each SGX instruction leaf attempts to operate on. The spectrum concurrency behavior is denoted by the following:

- 'N': The processor does not permit this pair of instructions to operate on the respective target EPC page concurrently. Software should serialize them.
- 'Y': The processor permits this pair of instructions to operate on the respective target EPC page concurrently.
- 'C': These two instruction flows will complete without fault. Status/Error code will return respectively.
- 'U': These two instruction flows will complete, but the occurrence these two simultaneous flows are considered a user program error that the processor do not enforce any restriction.
- 'P': The ENCLU flow may fail due to a page fault; the ENCLU flow can be re-executed.

For instance, multiple ELDB/ELDUs are allowed to execute as long as the selected EPC page is not the same page. Multiple ETRACK operations are not allowed to execute concurrently.

Table 5-5. Concurrency Restrictions of SGX Operations AEX - EWB

Operation		AEX			EADD		EBLOCK		ECREATE	EDBGRD/ WR		EENTER/ ERESUME			EEXTEND		EGETKEY		EINIT	ELDB/ELDU			EPA	EREMOVE		EREPORT		ETRACK	EWB		
	Type	TCS	SSA	SECS	Targ	SECS	Targ	SECS	SECS	Targ	SECS	TCS	SSA	SECS	Targ	SECS	Param	SECS	SECS	Targ	VA	SECS	VA	Targ	SECS	Param	SECS	SECS	Src	VA	SECS
AEX	TCS	N	N	N	N	N	Y	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	C	N	N	N	N	C	N	N
	SSA		U	N	N	N	Y	N	N	Y	N	N	U	N	N	N	U	N	N	N	N	N	N	C	N	U	N	N	C	N	N
	SECS			Y	N	N	Y	Y	N	N	Y	N	N	Y	N	N	N	Y	N	N	N	Y	N	C	C	N	Y	Y	C	N	Y
EADD	Targ				N	N	N	N	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	SECS					N	C	Y	N	N	Y	N	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N	Y	N	N	Y
EBLOCK	Targ						C	C	N	Y	C	P	P	C	Y	C	P	C	Y	N	Y	C	N	N	C	P	C	N	N	Y	C
	SECS							C	N	N	Y	N	N	Y	N	Y	N	Y	Y	N	N	Y	N	C	Y	N	Y	N	C	N	Y
ECREATE	SECS								N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
EDBGRD/ WR	Targ									Y	N	Y	Y	N	Y	N	Y	N	N	N	Y	N	N	N	N	Y	N	N	N	Y	N
	SECS										Y	N	N	Y	N	Y	N	Y	Y	N	N	Y	N	C	Y	N	Y	Y	C	N	Y
EENTER/ ERESUME	TCS											N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	SSA												U	N	N	N	U	N	N	N	N	N	N	N	N	U	N	N	N	N	N
	SECS													Y	N	N	N	Y	N	N	N	Y	N	N	C	N	Y	Y	C	N	Y
EEXTEND	Targ														Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	SECS															N	N	N	N	N	N	Y	N	C	Y	N	N	Y	C	N	Y
EGETKEY	Param																U	N	N	N	N	N	N	C	N	U	N	N	P	N	N
	SECS																	Y	N	C	N	Y	N	C	C	N	Y	Y	C	N	Y
EINIT	SECS																		N	N	N	Y	N	N	Y	N	N	Y	N	N	Y
ELDB/ ELDU	Targ																			N	N	N	N	N	N	N	C	N	N	N	N
	VA																				Y	N	N	N	N	N	N	N	N	Y	N
	SECS																					Y	N	N	Y	N	Y	Y	N	N	Y
EPA	VA																						N	N	N	N	N	N	N	N	N
EREMOVE	Targ																							N	C	C	C	N	N	N	C
	SECS																								Y	N	C	Y	C	N	Y
EREPORT	Param																									U	N	N	P	N	N
	SECS																										Y	Y	C	N	Y
ETRACK	SECS																											N	N	N	N
EWB	Src																												N	N	C
	VA																													Y	N
	SECS																														Y

5.2 SGX INSTRUCTION REFERENCE

ENCLS—Execute an Enclave System Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 01 CF ENCLS	NP	V/V	SE1	This instruction is used to execute privileged SGX leaf functions that are used for managing and debugging the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
NP	NA	NA	NA	See Section 5.3

Description

The ENCLS instruction invokes the specified privileged 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 use of ENCLS must be explicitly enabled by setting the SEE bit in Control Register 4 (CR4) to 1. Any attempt to execute ENCLS when CR4.SEE is 0 results in #UD. The instruction also results in a #UD if CR0.PE is 0 or RFLAGS.VM is 1, or if it is executed from inside SMM. Additionally, any attempt to execute this instruction when current privilege level is not 0 results in #UD.

Any attempt to invoke an undefined leaf function results in #GP(0).

If CR0.PG is 0, any attempt to invoke a leaf function other than EDBGRD or EDBGWR results in #GP(0).

In VMX non-root operation, execution of ENCLS results in #UD unless the “Enable ENCLS/ENCLU” VM-execution control is set.

Software in VMX root mode of operation can intercept the invocation of various ENCLS leaf functions from VMX non-root mode by setting bits in a 64-bit VM-execution control called ENCLS_EXITING (encoding pair 0202EH/0202FH). A processor implements this VM-execution control field if IA32_VMX_PROCBASED_CTLS2[47] is read as 1.

The DS segment is used to create linear addresses.

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

Segment prefix override is ignored. Address size prefix (67H) override is ignored.

REX prefix is ignored in 64-bit mode.

Operation

```
IN_64BIT_MODE← 0;
IF TSX_ACTIVE
    Then GOTO TSX_ABORT_PROCESSING; FI;

IF (CR4.SEE = 0 or CR0.PE = 0 or RFLAGS.VM = 1 or IN_SMM or CPUID.SE_LEAF.0.EAX.SE1 = 0 or ENCLS/ENCLU = 0)
    Then #UD; FI;

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

IF (in VMX non-root operation) and ( (EAX > 62 and ENCLS_EXITING[63] = 1) or ENCLS_EXITING[EAX] = 1)
    Then
        Set VMCS.EXIT_REASON = ENCLS;
        Deliver VM exit;
FI;
```

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

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

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

IN_64BIT_MODE ← IA32_EFER.LMA AND CS.L ? 1 : 0;

IF (IN_64BIT_MODE = 0 and (DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1)
    Then #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	If any of the LOCK/OSIZE/REP/VEX prefix is used.
	If current privilege level is not 0.
	If CR4.SEE = 0.
	If CPUID.(EAX=12H,ECX=0):EAX.SE1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If data segment expand down.
	If CR0.PG=0.

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/OSIZE/REP/VEX prefix is used.
	If current privilege level is not 0.
	If CR4.SEE = 0.
	If CPUID.(EAX=12H,ECX=0):EAX.SE1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.

ENCLU—Execute an Enclave User Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 01 D7 ENCLU	NP	V/V	SE1	This instruction is used to execute non-privileged SGX leaf functions that are used for operating the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
NP	NA	NA	NA	See Section 5.4

Description

The ENCLU instruction invokes the specified non-privileged 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 use of ENCLU must be explicitly enabled by setting the SEE bit in Control Register 4 (CR4) to 1. Any attempt to execute ENCLU when CR4.SEE is 0 results in #UD. The instruction also results in a #UD if CR0.PE is 0 or RFLAGS.VM is 1, or if it is executed from inside SMM. Additionally, any attempt to execute this instruction when current privilege level is not 3 results in #UD.

Any attempt to invoke an undefined leaf function results in #GP(0).

In VMX non-root operation, execution of ENCLU results in #UD unless the “Enable ENCLS/ENCLU” VM-execution control is set.

Software in VMX root mode of operation can intercept the invocation of various ENCLU leaf functions from VMX non-root mode by setting bits in a 64-bit VM-execution control called ENCLU_EXITING (encoding pair 0202EH/0202FH). A processor implements this VM-execution control field if IA32_VMX_PROCBASED_CTLS2[47] is read as 1.

The DS segment is used to create linear addresses.

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

Segment prefix override is ignored. Address size prefix (67H) override is ignored.

REX prefix is ignored in 64-bit mode.

Operation

```
IN_64BIT_MODE← 0;
IF TSX_ACTIVE
    Then GOTO TSX_ABORT_PROCESSING; FI;

IF (CR4.SEE = 0 or CR0.PE= 0 or RFLAGS.VM = 1 or IN_SMM or CPUID.SE_LEAF.0.EAX.SE1 = 0 or ENCLS/ENCLU = 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)
    Then #GP(0); FI;

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

IF (CR_ENCLAVE_MODE = 1 and ((EAX = EENTER) or (EAX = ERESUME) ) )
    Then #GP(0); FI;

IF (CR_ENCLAVE_MODE = 0 and ((EAX = EGETKEY) or (EAX = EREPORT) or (EAX = EEXIT)) )
    Then #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	If any of the LOCK/OSIZE/REP/VEX prefix is used.
	If current privilege level is not 3.
	If CR4.SEE = 0.
	If CPUID.(EAX=12H,ECX=0):EAX.SE1 [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 and ENCLAVE_MODE = 0.
	If operating in 16-bit mode.
	If data segment is in 16-bit mode.
	If CR0.PG = 0.
#NM	If CR0.TS = 1.

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

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

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/OSIZE/REP/VEX prefix is used.
	If current privilege level is not 3.
	If CR4.SEE = 0.

	If CPUID.(EAX=12H,ECX=0):EAX.SE1 [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 and ENCLAVE_MODE = 0.
#NM	If CR0.TS = 1.

5.3 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	SE1	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

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 GP(0); 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 GP(0); FI;

SCRATCH_SECINFO ← DS:RBX.TMP_SECINFO;

(* Check for mis-configured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero or
    ! (SCRATCH_SECINFO.FLAGS.PT is PT_REG or SCRATCH_SECINFO.FLAGS.PT is PT_TCS) )
    Then #GP(0); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use)
    Then #GP(0); FI;

IF (EPCM(DS:RCX).VALID != 1)
    Then #GP(0); 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 #GP(0); FI;

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

CASE (SCRATCH_SECINFO.FLAGS.PT)
{
    PT_TCS:
        IF (DS:RCX.RESERVED != 0) #GP(0); FI;
        IF ( (DS:TMP_SECS.ATTIBUTES.MODE64BIT = 0) and
            ((DS:TCS.FSLIMIT & 0FFFH != 0FFFH) or (DS:TCS.GSLIMIT & 0FFFH != 0FFFH) )) #GP(0); FI;
        BREAK;
    PT_REG:
        IF (SCRATCH_SECINFO.FLAGS.W = 1 and SCRATCH_SECINFO.FLAGS.R = 0) #GP(0); FI;
        BREAK;
ESAC;

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

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

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

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

(* Add enclave offset and security attributes to MRENCLAVE *)
TMP_ENCLAVEOFFSET ← TMP_LINADDR - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] ← 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).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 EPC page is valid.
	If the TCS page reserved bits are set.
#PF(fault code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If 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 EPC page is valid.
	If the TCS page reserved bits are set.
#PF(fault code)	If a page fault occurs in accessing memory operands.

EBLOCK—Mark a page in EPC as Blocked

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLS[EBLOCK]	IR	V/V	SE1	This leaf function marks a page in the EPC as blocked.

Instruction Operand Encoding

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

Description

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

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

An error code is returned in RAX.

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

EBLOCK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

EBLOCK Error Codes

0 (No Error)	EBLOCK successful
SGX_BLKSTATE	Page already blocked. This value is used to indicate that the page was already EBLOCKed and thus will need to be restored to this state when it is eventually reloaded (using ELDB).
SGX_ENTRYEPOCH_LOCKED	This value indicates that an ETRACK is currently executing on the SECS. The EBLOCK should be re-attempted.
SGX_NOTBLOCKABLE	Page type is not one which can be blocked
SGX_PG_INVLD	Page is not valid and cannot be blocked
SGX_LOCKFAIL	Page is being written by ECREATE, ELDU/ELDB, or EWB

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 GP(0); FI;

RFLAGS.ZF,CF,PF,AF,OF,SF ← 0;
RAX← 0;

(* Check concurrency with other SGX instructions *)
IF (ETRACK executed concurrently)
    Then
        RAX← SGX_ENTRYEPOCH_LOCKED;
        RFLAGS.ZF ← 1;
        goto Done;
    ELSIF (Other SGX instructions reading or writing EPCM)
        RAX← SGX_LOCKFAIL;
        RFLAGS.ZF ← 1;
        goto Done;
    FI;
FI;

IF (EPCM(DS:RCX). VALID = 0)
    Then
        RFLAGS.ZF ← 1;
        RAX← SGX_PG_INVLD;
        goto Done;
FI;

IF ( (EPCM(DS:RCX).PT != PT_REG) and (EPCM(DS:RCX).PT != PT_TCS) )
    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 *)
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 a memory operand is not an EPC page.
	If the specified EPC resource is in use.
#PF(fault code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

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

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	SE1	This leaf function begins an enclave build by creating an SECS page in EPC.

Instruction Operand Encoding

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

Description

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

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

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

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

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

ECREATE Memory Parameter Semantics

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

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

If the amount of space needed to store the SSA frame is greater than the amount specified in SECS.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 6.7.

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
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 GP(0); FI;

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

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

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

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

TMP_SECS ← RCX;

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

IF (EPCM(DS:RCX).VALID = 1)
    Then #GP(0); 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;
```

```
(* Compute the size required to save state of the enclave on async exit, see Section 6.7.2.2*)
TMP_XSIZE ← compute_xsave_size(DS:TMP_SECS.ATTRIBUTES.XFRM) + GPR_SIZE;

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

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

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

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

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE and 0FFFFFFE000000000H) )
    Then #GP(0); FI;

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

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

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

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

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

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

(* Add "ECREATE" string and SECS fields to MRENCLAVE *)
TMPUPDATEFIELD[63:0] ← 0045544145524345H; // "ECREATE"
TMPUPDATEFIELD[95:64] ← DS:TMP_SECS.SSAFRAMESIZE;
TMPUPDATEFIELD[159:96] ← DS:TMP_SECS.SIZE;
TMPUPDATEFIELD[511:160] ← 0;
SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

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

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

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED ← 0;
EPCM(DS:RCX).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 outside the EPC.
	If the SECS destination is locked.
	If SECS.SSAFRAMESIZE is insufficient.
#PF(fault code)	If a page fault occurs in accessing memory operands.

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

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	SE1	This leaf function reads a dword/quadword from a debug enclave.

Instruction Operand Encoding

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

Description

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

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

EDBGRD Memory Parameter Semantics

EPCQW
Read access permitted by Enclave

The instruction faults if any of the following:

EDBGRD Faulting Conditions

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

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

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 GP(0); FI;

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

IF (EPCM(DS:RCX). VALID = 0)
    Then #GP(0); FI;

(* make sure that DS:RCX (SOURCE) is pointing to a PT_REG or PT_TCS or PT_VA *)
IF ( (EPCM(DS:RCX).PT != PT_REG) and (EPCM(DS:RCX).PT != PT_TCS) and (EPCM(DS:RCX).PT != PT_VA))
    Then #GP(0); FI;

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

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

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights.
	If DS segment is unusable.
	If RCX points to a memory location not 4Byte-aligned.

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

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.
	If 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.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page
#PF(fault code)	If a page fault occurs in accessing memory operands.

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	SE1	This leaf function writes a dword/quadword to a debug enclave.

Instruction Operand Encoding

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

Description

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

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

EDBGWR Memory Parameter Semantics

EPCQW
Write access permitted by Enclave

The instruction faults if any of the following:

EDBGWR Faulting Conditions

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

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

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 GP(0); FI;

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

IF (EPCM(DS:RCX). VALID = 0)
    Then #GP(0); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS *)
IF ( (EPCM(DS:RCX).PT != PT_REG) and (EPCM(DS:RCX).PT != PT_TCS) )
    Then #GP(0); FI;

(* If destination is a TCS, then make sure that the offset into the page can only point to the FLAGS field*)
IF ( ( EPCM(DS:RCX). PT = PT_TCS) and ((DS:RCX) & 0xFF8H != offset_of_FLAGS & 0FF8H) )
    Then #GP(0); FI;

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

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

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

Flags Affected

None

Protected Mode Exceptions

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

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.
	If 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.

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

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	SE1	This leaf function measures 256 bytes of an uninitialized enclave page

Instruction Operand Encoding

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

Description

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

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

EEXTEND Memory Parameter Semantics

EPC[RCX]
Read access by Enclave

The instruction faults if any of the following:

EEXTEND Faulting Conditions

RCX points and address not 256B aligned	RCX points to an unused page or a SECS
RCX does not resolve in an EPC page	If SECS is locked
If the SECS is already initialized	May page fault
CPL != 0	

Operation

Temp Variables in EEXTEND Operational Flow

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

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

IF ( (DS:RCX is not 256Byte Aligned) or (DS:RCX does not resolve within an EPC) )
    Then GP(0); FI;
```

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

IF (EPCM(DS:RCX). VALID = 0)
    Then #GP(0); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS *)
IF ( (EPCM(DS:RCX).PT != PT_REG) and (EPCM(DS:RCX).PT != PT_TCS) )
    Then #GP(0); FI;

TMP_SECS ← Get_SECS_ADDRESS();

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUETS.INIT *)
IF ( (Other instruction accessing MRENCLAVE) or (Other instructions checking or updating the initialized state of the SECS))
    Then #GP(0); FI;

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

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

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

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If the address in RCS is outside the DS segment limit.
	If RCX points to a memory location not 256Byte-aligned.
	If 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.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(fault code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.
	If RCX points to a memory location not 256 Byte-aligned.

	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.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(fault code)	If a page fault occurs in accessing memory operands.

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	SE1	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.

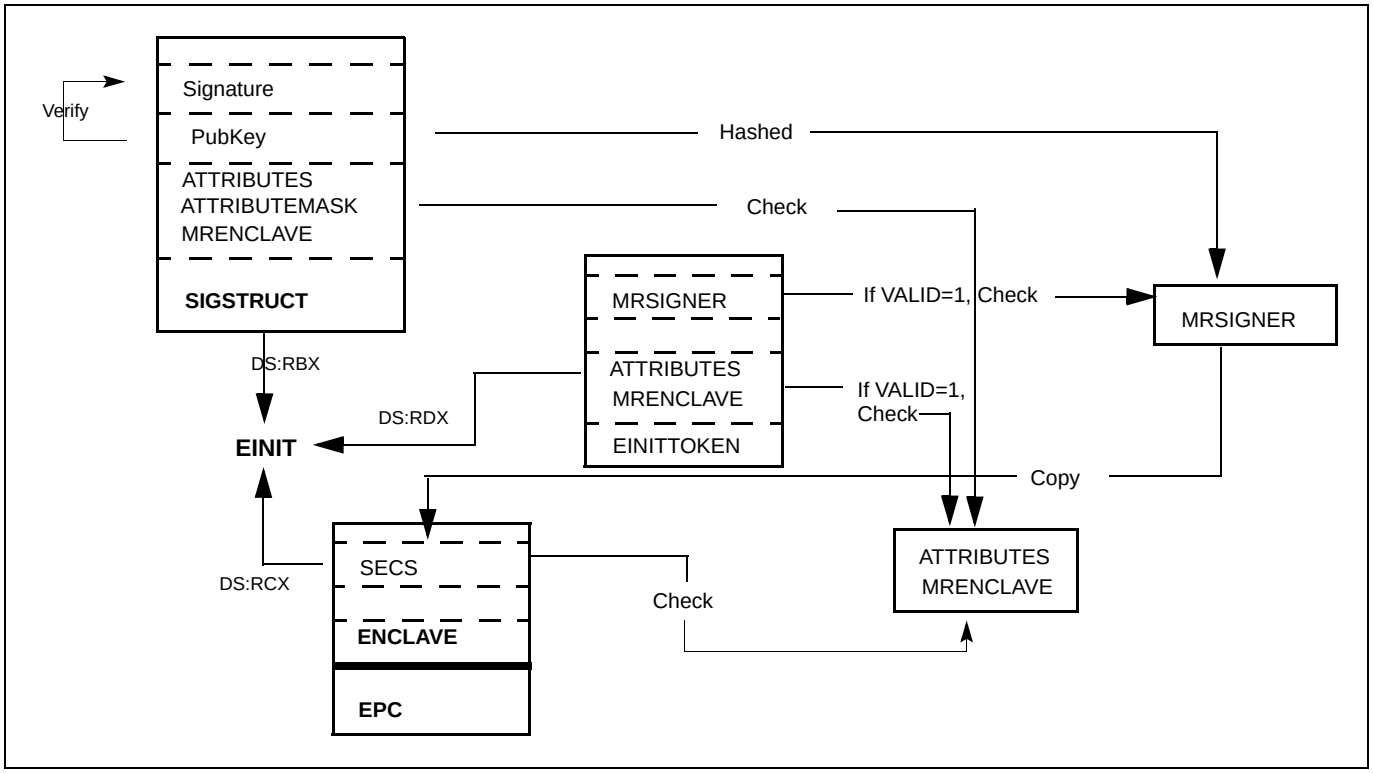

Figure 5-1. Relationships Between SECS, SIGSTRUCT and EINITTOKEN

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 5-1:

Validates that SIGSTRUCT is signed using the enclosed public key.

Checks that the completed computation of SECS.MRENCLAVE equals SIGSTRUCT.HASHENCLAVE.

Checks that no reserved bits are set to 1 in SIGSTRUCT.ATTRIBUTES and no reserved bits in SIGSTRUCT.ATTRIBUTESMASK are set to 0.

Checks that no Intel-only bits are set in SIGSTRUCT.ATTRIBUTES unless SIGSTRUCT was signed by Intel.

Checks that SIGSTRUCT.ATTRIBUTES equals the result of logically and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.

If EINITTOKEN.VALID is 0, checks that SIGSTRUCT is signed by Intel.

If EINITTOKEN.VALID is 1, checks the validity of EINITTOKEN.

If EINITTOKEN.VALID is 1, checks that EINITTOKEN.MRENCLAVE equals SECS.MRENCLAVE.

If EINITTOKEN.VALID is 1 and EINITTOKEN.ATTRIBUTES.DEBUG is 1, SECS.ATTRIBUTES.DEBUG must be 1.

Commits SECS.MRENCLAVE, and sets SECS.MRSIGNER, SECS.ISVSVN, and SECS.ISVPRODID based on SIGSTRUCT.

Update the SECS as Initialized.

Periodically, EINIT polls for certain asynchronous events. If such an event is detected, it completes with failure code (ZF=1 and RAX = SGX_UNMASKED_EVENT), and RIP is incremented to point to the next instruction. These events are INTR, NMI, SMI, INIT, VMX_TIMER, MCAKIND, MCE_SMI, and CMCI_SMI. EINIT does not fail if the pending event is inhibited (e.g., INTR could be inhibited due to MOV/POP SS blocking and STI blocking).

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

Operation

Temp Variables in EINIT Operational Flow

Name	Type	Size	Description
TMP_SIG	SIGSTRUCT	1808Bytes	Temp space for SIGSTRUCT
TMP_TOKEN	EINITTOKEN	304Bytes	Temp space for EINITTOKEN
TMP_MRENCLAVE		32Bytes	Temp space for calculating MRENCLAVE
TMP_MRSIGNER		32Bytes	Temp space for calculating MRSIGNER
INTEL_ONLY_MASK	ATTRIBUTES	16Bytes	Constant mask of all ATTRIBUTE bits that can only be set for Intel enclaves
CSR_INTELPUBKEYHASH		32Bytes	Constant with the SHA256 of the Intel Public key used to sign Architectural Enclaves
TMP_KEYDEPENDENCIES	Buffer	224Bytes	Temp space for key derivation
TMP_EINITTOKENKEY		16Bytes	Temp space for the derived EINITTOKEN Key
TMP_SIG_PADDING	PKCS Padding Buffer	352Bytes	The value of the top 352 bytes from the computation of Signature^3 modulo MRSIGNER

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

```
(* make sure the EINITTOKEN is aligned and SECS is inside the EPC *)
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RCX does not resolve within an EPC) )
    Then GP(0); FI;

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

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

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

(* make sure no other 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 #GP(0); FI;

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

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

(* Verify MRENCLAVE from SIGSTRUCT *)
IF (TMP_SIG.ENCLAVEHASH != TMP_MRENCLAVE)
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_MEASUREMENT;
    goto EXIT;
FI;
```

```
TMP_MRSIGNER ← SHA256(TMP_SIG.MODULUS)

(* if INTEL_ONLY ATTRIBUTES are set, SIGSTRUCT must be signed using the Intel Key *)
INTEL_ONLY_MASK ← 0000000000000020H;
IF ( ( (DS:RCX.ATTRIBUTES & INTEL_ONLY_MASK) != 0) and (TMP_MRSIGNER != CSR_INTELPUBKEYHASH) )
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_ATTRIBUTE;
    goto EXIT;
FI;

(* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
IF ( (DS:RCX.ATTRIBUTES & TMP_SIG.ATTRIBUTEMASK) != (TMP_SIG.ATTRIBUTE & TMP_SIG.ATTRIBUTEMASK) )
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_ATTRIBUTE;
    goto EXIT;
FI;

(* if EINITTOKEN.VALID[0] is 0, verify the enclave is signed by Intel *)
IF (TMP_TOKEN.VALID[0] = 0)
    IF (TMP_MRSIGNER != CSR_INTELPUBKEYHASH)
        RFLAG.ZF ← 1;
        RAX ← SGX_INVALID_EINITTOKEN;
        goto EXIT;
    FI;
    goto COMMIT;
FI;

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

(* Check reserve space in EINIT token includes reserved regions and upper bits in valid field *)
IF (TMP_TOKEN reserved space is not clear)
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_EINITTOKEN;
    goto EXIT;
FI;

(* EINIT token must be <= CR_CPUSVN *)
IF (TMP_TOKEN.CPUSVN > CR_CPUSVN)
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_CPUSVN;
    goto EXIT;
FI;

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

```
TMP_KEYDEPENDENCIES.KEYNAME ← LAUNCH_KEY;
TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_TOKEN.ISVPRODIDLE;
TMP_KEYDEPENDENCIES.ISVSVN ← TMP_TOKEN.ISVSVN;
TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_TOKEN.MASKEDATTRIBUTESLE;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
TMP_KEYDEPENDENCIES.KEYID ← TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← TMP_TOKEN.CPUSVN;
TMP_KEYDEPENDENCIES.PADDING ← HARDCODED_PKCS1_5_PADDING;

(* Calculate the derived key*)
TMP_EINITTOKENKEY ← derivekey(TMP_KEYDEPENDENCIES);

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

(* Verify EINITTOKEN (RDX) is for this enclave *)
IF (TMP_TOKEN.MRENCLAVE != TMP_MRENCLAVE) or (TMP_TOKEN.MRSIGNER != TMP_MRSIGNER) )
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_MEASUREMENT;
    goto EXIT;
FI;

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

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

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

(* Set RAX and ZF for success*)
    RFLAG.ZF ← 0;
    RAX ← 0;
EXIT:
```

RFLAGS.CF,PF,AF,OF,SF ← 0;

Flags Affected

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

Protected Mode Exceptions

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

64-Bit Mode Exceptions

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

ELDB/ELDU—Load an EPC page and Marked its State

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLS[ELDB]	IR	V/V	SE1	This leaf function loads, verifies an EPC page and marks the page as blocked.
EAX = 08H ENCLS[ELDU]	IR	V/V	SE1	This leaf function loads, verifies an EPC page and marks the page as unblocked.

Instruction Operand Encoding

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

Description

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

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

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

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

EBLDB/ELDBU Memory Parameter Semantics

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

The error codes are:

ELDB/ELDU Error Codes

0 (No Error)	EBLOCK successful
SGX_MAC_COMPARE_FAIL	If the MAC check fails

Operation

Temp Variables in ELDB/ELDU Operational Flow

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

```
(* 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 GP(0); 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 GP(0); FI;

TMP_SRCPGE ← DS:RBX.SRCPGE;
TMP_SECS ← DS:RBX.SECS;
TMP_PCMD ← DS:RBXPCMD;

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

(* Check concurrency of EPC and VASLOT by other SGX instructions *)
IF ( (other instructions accessing EPC) or (Other instructions modifying VA slot) )
    Then GP(0); FI;

(* Verify EPCM attributes of EPC page, VA, and SECS *)
IF ( (EPCM(DS:RCX).VALID = 1) or (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~0FFFH).PT != PT_VA) )
    Then GP(0); FI;

(* Copy SECINFO into scratch buffer *)
TMP_HEADER[sizeof(TMP_HEADER)-1: 0]← 0;
TMP_HEADER.SECINFO.FLAGS.PT ← DS:TMP_PCMD.SECINFO.FLAGS.PT;
TMP_HEADER.SECINFO.FLAGS.RWX ← DS:TMP_PCMD.SECINFO.FLAGS.RWX;
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) )
```

```
    Then
        IF ( ( DS:TMP_SECS is not 4KByte aligned) or (DS:TMP_SECS does not resolve within an EPC) )
            THEN #GP(0) FI;
        IF ( ( Other instructions modifying SECS) or (EPCM(DS:TMP_SECS).VALID = 0) or (EPCM(DS:TMP_SECS).PT != PT_SECS) )
            THEN #GP(0) FI;
    ELSIF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_SECS) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_VA) )
        IF ( ( TMP_SECS != 0) )
            THEN #GP(0) FI;
    ELSE
            #GP(0)
FI;

IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) )
    Then
        TMP_HEADER.EID ← DS:TMP_SECS.EID;
    ELSE
        (* These pages do not have any parent, and hence no EID binding *)
        TMP_HEADER.EID ← 0;
FI;

(* Copy 4KBytes SRCPGE to secure location *)
DS:RCX[32767: 0]← DS:TMP_SRCPGE[32767: 0];
TMP_VER ← DS:RDX[63:0];

(* Decrypt and MAC page. AES_GCM_DEC has 2 outputs, {plain text, MAC} *)
(* Parameters for AES_GCM_DEC {Key, Counter, ..} *)
{DS:RCX, TMP_MAC} ← AES_GCM_DEC(CR_BASE_PK, TMP_VER << 32, TMP_HEADER, 128, DS:RCX, 4096);

IF ( (TMP_MAC != DS:TMP_PCMD.MAC) )
    Then
        RFLAGS.ZF ← 1;
        RAX← SGX_MAC_COMPARE_FAIL;
        goto ERROR_EXIT;
FI;

(* Check version before committing *)
IF (DS:RDX != 0)
    Then #GP(0);
    ELSE
        DS:RDX← TMP_VER;
FI;

(* Commit EPCM changes *)
EPCM(DS:RCX).PT ← TMP_HEADER.SECINFO.FLAGS.PT;
EPCM(DS:RCX).RWX ← TMP_HEADER.FLAGS.RWX;
EPCM(DS:RCX).ENCLAVEADDRESS ← TMP_HEADER.LINADDR;

IF (EAX = 07H)
    Then
        EPCM(DS:RCX).BLOCKED ← 1;
    ELSE
        EPCM(DS:RCX).BLOCKED ← 0;
FI;
```

```
EPCM(DS:RCX). VALID ← 1;

RAX← 0;
RFLAGS.ZF ← 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If 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 instruction's EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the destination EPC page is already valid.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(fault code)	If a page fault occurs in accessing memory operands.

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 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 instruction's EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the destination EPC page is already valid.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(fault code)	If a page fault occurs in accessing memory operands.

EPA—Add Version Array

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0AH ENCLS[EPA]	IR	V/V	SE1	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

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 GP(0); FI;

(* Check concurrency with other SGX instructions *)
IF (Other SGX instructions accessing the page)
    THEN #GP(0); FI;

(* Check EPC page must be empty *)
IF (EPCM(DS:RCX). VALID != 0)
    THEN #GP(0); 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).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 a memory operand is not an EPC page.
	If another SGX instruction is accessing the EPC page.
	If the EPC page is valid.
	If RBX is not set to PT_VA.
#PF(fault code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is not an EPC page.
	If another SGX instruction is accessing the EPC page.
	If the EPC page is valid.
	If RBX is not set to PT_VA.
#PF(fault 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	SE1	This leaf function removes a page from the EPC.

Instruction Operand Encoding

Op/En	EAX	RCX
IR	EREMOVE (In)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be un-associated with its SECS and be marked as unused. This instruction leaf can only be executed when the current privilege level is 0.

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

The instruction fails if the operand is not properly aligned or does not refer to an EPC page or the page is in use by another thread, or other threads are running in the enclave to which the page belongs. In addition the instruction fails if the operand refers to an SECS with associations.

EREMOVE Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

The instruction faults if any of the following:

EREMOVE Faulting Conditions

The memory operand is not properly aligned	The memory operand does not resolve in an EPC page
Refers to an invalid SECS	Refers to an EPC page that is locked by another thread
Another SGX instruction is accessing the EPC page	RCX does not contain an effective address of an EPC page
the EPC page refers to an SECS with associations	

The error codes are:

EREMOVE Error Codes

0 (No Error)	EBLOCK 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

Operation

Temp Variables in EREMOVE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page

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

IF (DS:RCX does not resolve within an EPC)
    Then GP(0); FI;

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

SCRATCH_SECINFO ← DS:RBX.TMP_SECINFO;

(* Check the EPC page for concurrency *)
IF (EPC page being referenced by another SGX instruction)
    Then #GP(0); FI;

(* if DS:RCX is already unused, nothing to do*)
IF (EPCM(DS:RCX).VALID = 0)
    Then goto DONE;
FI;

IF (EPCM(DS:RCX).PT = PT_VA)
    Then
        EPCM(DS:RCX).VALID ← 0;
        goto DONE;
FI;

IF (EPCM(DS:RCX).PT = PT_SECS)
    Then
        IF (DS:RCX has an EPC page associated with it)
            Then
                RFLAGS.ZF ← 1;
                RAX← SGX_CHILD_PRESENT;
                goto ERROR_EXIT;
        FI;
        EPCM(DS:RCX).VALID ← 0;
        goto DONE;
FI;

TEMP_SECS ← Get_SECS_ADDRESS();

IF (Other threads active using SECS)
    Then
        RFLAGS.ZF ← 1;
        RAX← SGX_ENCLAVE_ACT;
```

```
        goto ERROR_EXIT;
FI;

DONE:
RAX← 0;
RFLAGS.ZF ← 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

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

64-Bit Mode Exceptions

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

ETRACK—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0CH ENCLS[ETRACK]	IR	V/V	SE1	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	ETRACK (In)	Return error code (Out)	Pointer to the SECS of the EPC page (In)

Description

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

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

ETRACK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

ETRACK Error Codes

0 (No Error)	EBLOCK successful
SGX_PREV_TRK_INCMPL	All logical processors on the platform did not complete the previous tracking cycle.

Operation

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

IF (DS:RCX does not resolve within an EPC)
    Then GP(0); FI;

(* Check concurrency with other SGX instructions *)
IF (Other SGX instructions using tracking facility on this SECS)
    Then GP(0); FI;


IF (EPCM(DS:RCX). VALID = 0)
    Then GP(0); FI;

IF (EPCM(DS:RCX).PT != PT_SECS)
    Then GP(0); FI;

(* All processors must have completed the previous tracking cycle*)
IF ( (DS:RCX).TRACKING != 0) )
```

```
    Then
        RFLAGS.ZF ← 1;
        RAX← SGX_PREV_TRK_INCMPL;
        goto Done;
    ELSE
        RAX← 0;
        RFLAGS.ZF ← 0;
FI;

Done:
RFLAGS.ZF,CF,PF,AF,OF,SF ← 0;
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is not an EPC page.
	If another thread is concurrently using the tracking facility on this SECS.
#PF(fault code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

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

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	SE1	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

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 GP(0); FI;

IF (DS:RDX is not 8Byte Aligned)
    Then GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
```

```
    Then GP(0); 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:TMP_PCMD is not 128Byte Aligned) or (DSTMP_SRCPGE is not 4KByte Aligned) )
    Then GP(0); FI;

(* Check for concurrent SGX instruction access to the page *)
IF (Other SGX instruction is accessing page)
    THEN #GP(0); FI;

(*Check if the VA Page is being removed or changed*)
IF (VA Page is being modified)
    THEN #GP(0); FI;

(* Verify that EPCPAGE and VASLOT page are valid EPC pages and DS:RDX is VA *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~0xFFF).PT is not PT_VA) )
    THEN #GP(0); FI;

(* Perform page-type-specific exception checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) )
    THEN
        TMP_SECS = Obtain SECS through EPCM(DS:RCX)
    (* Check that EBLOCK has occurred correctly *)
    IF (EBLOCK is not correct)
        THEN #GP(0); FI;
FI;

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

(* Perform page-type-specific checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) )
    THEN
        (* check to see if the page is evictable *)
        IF (EPCM(DS:RCX).BLOCKED = 0)
            THEN
                RAX ← SGX_PAGE NOT_BLOCKED;
                RFLAGS.ZF ← 1;
                GOTO ERROR_EXIT;
        FI;
        (* Check if tracking done correctly *)
        IF (Tracking not correct)
            THEN
                RAX ← SGX_NOT_TRACKED;
                RFLAGS.ZF ← 1;
                GOTO ERROR_EXIT;
        FI;
```

```
        (* Obtain EID to establish cryptographic binding between the paged-out page and the enclave *)
        TMP_HEADER.EID ← TMP_SECS.EID;

        (* Obtain EID as an enclave handle for software *)
        TMP_PCMD_ENCLAVEID ← TMP_SECS.EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_SECS)
        (*check that there are no child pages inside the enclave *)
        IF (DS:RCX has an EPC page associated with it)
            THEN
                RAX ← SGX_CHILD_PRESENT;
                RFLAGS.ZF ← 1;
                GOTO ERROR_EXIT;
        FI:
        TMP_HEADER.EID ← 0;
        (* Obtain EID as an enclave handle for software *)
        TMP_PCMD_ENCLAVEID ← (DS:RCX).EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_VA)
        TMP_HEADER.EID ← 0; // Zero is not a special value
        (* No enclave handle for VA pages*)
        TMP_PCMD_ENCLAVEID ← 0;
FI;

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.RSVD ← 0;

(* 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.RESERVED ← 0;
DS:TMP_PCMD.ENCLAVEID ← TMP_PCMD_ENCLAVEID;
DS:RBX.LINADDR ← EPCM(DS:RCX).ENCLAVEADDRESS;

(*Check if version array slot was empty *)
IF ([DS.RDX])
    THEN
        RAX ← SGX_VA_SLOT_OCCUPIED
        RFLAGS.CF ← 1;
FI;

(* Write version to Version Array slot *)
[DS.RDX] ← TMP_VER;

(* Free up EPCM Entry *)
EPCM.(DS:RCX).VALID ← 0;
EXIT:
```

Flags Affected

ZF is set if page is not blocked, not tracked, or a child is present. Otherwise cleared.

CF is set if VA slot is previously occupied, Otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is not an EPC page.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If one of the EPC memory operands has incorrect page type.
	If the tracking resource is in use.
	If the EPC page or the version array page is invalid.
	If the parameters fail consistency checks.
	If the page is not blocked.
	If the page is not properly tracked.
#PF(fault code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is not an EPC page.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If one of the EPC memory operands has incorrect page type.
	If the tracking resource is in use.
	If the EPC page or the version array page in invalid.
	If the parameters fail consistency checks.
	If the page is not blocked.
	If the page is not properly tracked.
#PF(fault code)	If a page fault occurs in accessing memory operands.

5.4 SGX USER LEAF FUNCTION REFERENCE

5.4.1 Instruction Column in the Instruction Summary Table

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

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

EENTER—Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLU[EENTER]	IR	V/V	SE1	This leaf function is used to enter an enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	
IR	EENTER (In)	Content of RBX.CSSA (Out)	Address of a TCS (In)	Address of AEP (In)	Address of IP following EENTER (Out)

Description

The ENCLU[EENTER] instruction transfers execution to an enclave. At the end of the instruction, the logical processor is executing in enclave mode at the RIP computed as EnclaveBase + TCS.OENTRY. If the target address is not within the CS segment (32-bit) or is not canonical (64-bit), a #GP(0) results.

EENTER Memory Parameter Semantics

TCS
Enclave access

EENTER is a serializing instruction. The instruction faults if any of the following occurs:

Address in RBX is not properly aligned	Any TCS.FLAGS's must-be-zero bit is not zero
TCS pointed to by RBX is not valid or available or locked	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64
The SECS is in use	Either of TCS-specified FS and GS segment is not a subsets of the current DS segment
Any one of DS, ES, CS, SS is not zero	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM != 0x3
CR4.OSFXSR != 1	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0

The following operations are performed by EENTER:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or interrupt.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM.The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 7.1.2):
  - 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 7.2.6).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 7.2.3).
- 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 7.2.4):

— All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.

— PEBS is suppressed.

— AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set

— If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

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 GP(0); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (DS:RCX is not canonical) )
    Then #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other SGX instructions is operating on TCS)
    Then #GP(0); FI;

(* TCS verification *)
```

```
IF (EPCM(DS:RBX).VALID = 0)
    Then #GP(0); 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 #GP(0); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    Then GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    Then GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS ← Address of SECS for TCS;

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    Then
        TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_FSLIMIT ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
        TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        TMP_GSLIMIT ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
        (* if FS wrap-around, make sure DS has no holes*)
        IF (TMP_FSLIMIT < TMP_FSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
        (* if GS wrap-around, make sure DS has no holes*)
        IF (TMP_GSLIMIT < TMP_GSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
    ELSE
        TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
            THEN #GP(0); FI;
FI;

(* Ensure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & & 0xFFFFFFFFFFFFFFFE) != 0)
    Then #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    Then #GP(0); FI;
```

```
(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 != TMP_SECS.ATTRIBUTES.MODE64BIT) )
    Then #GP(0); FI;

IF (CR4.OSFXSR = 0)
    Then #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    Then
        IF (TMP_SECS.ATTRIBUES.XFRM != 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUES.XFRM & XCR0) != TMP_SECS.ATTRIBUES.XFRM) THEN #GP(0); FI;
FI;

(* Make sure the SSA contains at least one more frame *)
IF ( (DS:RBX).CSSA >= (DS:RBX).NSSA)
    Then #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA ← (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * (DS:RBX).CSSA;
TMP_XSIZE ← compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;

    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        Then #GP(0); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        Then #GP(0); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        Then #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS != DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT != PT_REG) or
        (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS != EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_SECS).R = 0) or (EPCM(DS:TMP_SECS).W = 0) )
        Then #GP(0); 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 #GP(0); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
    Then #GP(0); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 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 #GP(0); FI;

IF (TMP_MODE64 = 0)
    Then
        IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) Then #GP(0); FI;
FI;

CR_GPR_PA ← Physical_Address (DS: TMP_GPR);

(* Validate TCS.OENTRY *)
TMP_TARGET ← (DS:RBX).OENTRY + TMP_SECS.BASEADDR;
IF (TMP_MODE64 = 1)
    Then
        IF (TMP_TARGET is not canonical) Then #GP(0); FI;
    ELSE
        IF (TMP_TARGET > CS limit) Then #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE))
    Then #GP(0); FI;

CR_ENCALVE_MODE ← 1;
CR_ACTIVE_SECS ← TMP_SECS;
CR_ELRANGE ← (TMPSECS.BASEADDR, TMP_SECS.SIZE);

(* Save state for possible AEXs *)
CR_TCS_PA ← Physical_Address (DS:RBX);
CR_TCS_LA ← RBX;
CR_TCS_LA.AEP ← RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector ← FS.selector;
CR_SAVE_FS_base ← FS.base;
CR_SAVE_FS_limit ← FS.limit;
CR_SAVE_FS_access_rights ← FS.access_rights;
CR_SAVE_GS_selector ← GS.selector;
CR_SAVE_GS_base ← GS.base;
CR_SAVE_GS_limit ← GS.limit;
CR_SAVE_GS_access_rights ← GS.access_rights;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 ← XCR0;
    XCR0 ← TMP_SECS.ATTRIBUTES.XFRM;
FI;

(* Set CR_ENCLAVE_ENTRY_IP *)
CR_ENCLAVE_ENTRY_IP ← CRIP"
RIP ← NRIP;
RAX ← (DS:RBX).CSSA;
(* Save the outside RSP and RBP so they can be restored on interrupt or EEXIT *)
DS:TMP_SSA.U_RSP ← RSP;
```

```
DS:TMP_SSA.U_RBP ← RBP;

(* Do the FS/GS swap *)
FS.base ← TMP_FSBASE;
FS.limit ← DS:RBX.FSLIMIT;
FS.type ← 0001b;
FS.W ← DS.W;
FS.S ← 1;
FS.DPL ← DS.DPL;
FS.G ← 1;
FS.B ← 1;
FS.P ← 1;
FS.AVL ← DS.AVL;
FS.L ← DS.L;
FS.unusable ← 0;
FS.selector ← 0BH;

GS.base ← TMP_GSBASE;
GS.limit ← DS:RBX.GSLIMIT;
GS.type ← 0001b;
GS.W ← DS.W;
GS.S ← 1;
GS.DPL ← DS.DPL;
GS.G ← 1;
GS.B ← 1;
GS.P ← 1;
GS.AVL ← DS.AVL;
GS.L ← DS.L;
GS.unusable ← 0;
GS.selector ← 0BH;

CR_DBGOPTIN ← TSC.FLAGS.DBGOPTIN;
Suppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        Suppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        CR_SAVE_TF ← RFLAGS.TF;
        RFLAGS.TF ← 0;
        Suppress_monitor_trap_flag for the source of the execution of the enclave;
        Clear_all_pending_debug_exceptions;
        Clear_pending_MTF_VM_exit;
    ELSE
        IF (RFLAGS.TF = 1)
            Then Pend_Single-Step_#DB_at_the_end_of_ENTER; FI;
        IF (VMCS.MTF = 1)
            Then Pend_MTF_VM_exit_at_the_end_of_ENTER; FI;
FI;

Flush_linear_context;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If DS:RBX does not point to a valid TCS.
	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 any reserved field in the TCS FLAG is set.
	If the target address is not within the CS segment.
	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.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM != 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(fault code)	If a page fault occurs in accessing memory.
#NM	If CR0.TS is set.

64-Bit Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If DS:RBX does not point to a valid TCS.
	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 one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM != 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(fault code)	If a page fault occurs in accessing memory operands.
#NM	If CR0.TS is set.

EEXIT—Exits an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLU[EEXIT]	IR	V/V	SE1	This leaf function is used to exit an enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EEXIT (In)	Target address outside the enclave (In)	Address of the current AEP (In)

Description

The ENCLU[EEXIT] instruction exits the currently executing enclave and branches to the location specified in RBX. RCX receives the current AEP. If RBX is not within the CS (32-bit mode) or is not canonical (64-bit mode) a #GP(0) results.

EEXIT Memory Parameter Semantics

Target Address
non-Enclave read and execute access

If RBX specifies an address that is inside the enclave, the instruction will complete normally. The fetch of the next instruction will occur in non-enclave mode, but will attempt to fetch from inside the enclave. This has the effect of abort page semantics on the next destination.

If secrets are contained in any registers, it is responsibility of enclave software to clear those registers.

If XCR0 was modified on enclave entry, it is restored to the value it had at the time of the most recent EENTER or ERESUME.

If the enclave is opt-out, RFLAGS.TF is loaded from the value previously saved on EENTER.

Code and data breakpoints are unsuppressed.

Performance monitoring counters are unsuppressed.

Operation

Temp Variables in EEXIT Operational Flow

Name	Type	Size (Bits)	Description
TMP_RIP	Effective Address	32/64	Saved copy of CRIP for use when creating LBR

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

IF (TMP_MODE64 = 1)
    Then
        IF (RBX is not canonical) Then #GP(0); FI;
    ELSE
        IF (RBX > CS limit) Then #GP(0); FI;
FI;

TMP_RIP ← CRIP;
RIP ← RBX;

(* Return current AEP in RCX *)
```

```
RCX ← CR_TCS_PA.AEP;

(* Do the FS/GS swap *)
FS.selector ← CR_SAVE_FS.selector;
FS.base ← CR_SAVE_FS.base;
FS.limit ← CR_SAVE_FS.limit;
FS.access_rights ← CR_SAVE_FS.access_rights;
GS.selector ← CR_SAVE_GS.selector;
GS.base ← CR_SAVE_GS.base;
GS.limit ← CR_SAVE_GS.limit;
GS.access_rights ← CR_SAVE_GS.access_rights;

(* Restore XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
    XCR0 ← CR_SAVE__XCR0;
FI;

Unsuppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        UnSuppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        Restore suppressed breakpoint matches;
        RFLAGS.TF ← CR_SAVE_TF;
        UnSuppress_montior_trap_flag;
        UnSuppress_LBR_Generation;
        UnSuppress_performance monitoring_activity;
        Restore performance monitoring counter AnyThread demotion to MyThread in enclave back to AnyThread
FI;

IF (RFLAGS.TF = 1)
    Pend Single-Step #DB at the end of EEXIT;
FI;

IF (VMCS.MTF = 1)
    Pend MTF VM exit at the end of EEXIT;
FI;

CR_ENCLAVE_MODE ← 0;
CR_TCS_PA.STATE ← INACTIVE;

(* Assure consistent translations *)
Flush_linear_context;
```

Flags Affected

RFLAGS.TF is restored from the value previously saved in EENTER or ERESUME.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RBX is outside the CS segment.
#PF(fault code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RBX is not canonical.
#PF(fault code)	If a page fault occurs in accessing memory operands.

EGETKEY—Retrieves a Cryptographic Key

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLU[EGETKEY]	IR	V/V	SE1	This leaf function retrieves a cryptographic key.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EGETKEY (In)	Address to a KEYREQUEST (In)	Address of the OUTPUTDATA (In)

Description

The ENCLU[EGETKEY] instruction returns a 128-bit secret key from the processor specific key hierarchy. The register RBX contains the effective address of a KEYREQUEST structure, which the instruction interprets to determine the key being requested. The Requesting Keys section below provides a description of the keys that can be requested. The RCX register contains the effective address where the key will be returned. Both the addresses in RBX & RCX should be locations inside the enclave.

EGETKEY derives keys using a processor unique value to create a specific key based on a number of possible inputs. This instruction leaf can only be executed inside an enclave.

EEGETKEY Memory Parameter Semantics

KEYREQUEST	OUTPUTDATA
Enclave read access	Enclave write access

After validating the operands, the instruction determines which key is to be produced and performs the following actions:

- The instruction assembles the derivation data for the key based on the Table 5-6
- 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, and the operands are not inside the currently executing enclave. 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 2.6.11.1) 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 5-6) 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 5-6 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 5-6. Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODID	MRENCLAVE	MRSIGNER	RAND
Source	Key Dependent Constant	Y← SECS.ATTRIBUTES; R←AttribMask & SECS..ATTRIBUTES;	CSR_SEOWNEREPOCH	Y← CPUSVN Register; R← Req.CPUSVN;	R← Req.ISVSVN;	SECS.ISVID	SECS.MRENCLAVE	SECS.MRSIGNER	Req.KEYID
Launch	Yes	Request	Yes	Request	Request	Yes	No	No	Request
Report	Yes	Yes	Yes	Yes	No	No	Yes	No	Request
Seal	Yes	Request	Yes	Request	Request	Yes	Request	Request	Request
Provisioning	Yes	Request	Yes	Request	Request	Yes	No	Yes	Yes
Provisioning Seal	Yes	Request	Yes	Request	Request	Yes	No	Yes	Yes

Keys that permit the specification of a CPU or ISV's code's SVNs have additional requirements. The caller may not request a key for an SVN beyond the current CPU or ISV SVN, respectively.

Some keys are derived based on a hardcode PKCS padding constant (352 byte string):

HARDCODED_PKCS1_5_PADDING[15:0] ß 0100H;

HARDCODED_PKCS1_5_PADDING[2655:16] ß SignExtend330Byte(-1); // 330 bytes of 0FFH

HARDCODED_PKCS1_5_PADDING[2815:2656] ß 2004000501020403650148866009060D30313000H;

The error codes are:

EGETKEY Error Codes

0 (No Error)	EBLOCK successful
SGX_INVALID_ATTRIBUTE	The KEYREQUEST contains a KEYNAME for which the enclave is not authorized
SGX_INVALID_CPUSVN	If KEYREQUEST.CPUSVN is beyond platforms CPUSVN value
SGX_INVALID_ISVSVN	If KEYREQUEST.ISVSVN is greater than the enclave's ISV_SVN
SGX_INVALID_KEYNAME	If KEYREQUEST.KEYNAME is an unsupported value.

Operation

Temp Variables in EGETKEY Operational Flow

Name	Type	Size (Bits)	Description
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave
TMP_KEYDEPENDENCIES			Temp space for key derivation
TMP_ATTRIBUTES		128	Temp Space for the calculation of the sealable Attributes
TMP_OUTPUTKEY		128	Temp Space for the calculation of the key

(* Make sure KEYREQUEST is properly aligned and inside the current enclave *)
IF ((DS:RBX is not 128Byte aligned) or (DS:RBX is within CR_ELRANGE))

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

(* Make sure DS:RBX is an EPC address and the EPC page is valid *)
IF ( (DS:RBX does not resolve to an EPC address) or (EPCM(DS:RBX).VALID = 0) )
    THEN #GP(0); 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).ENCLAVEADDRESS != (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #GP(0);
FI;

(* Make sure OUTPUTDATA is properly aligned and inside the current enclave *)
IF ( (DS:RCX is not 16Byte aligned) or (DS:RCX is within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RCX is an EPC address and the EPC page is valid *)
IF ( (DS:RCX does not resolve to an EPC address) or (EPCM(DS:RCX).VALID = 0) )
    THEN #GP(0); 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).ENCLAVEADDRESS != (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).W = 0) )
    THEN #GP(0);
FI;

(* Verify RESERVED spaces in KEYREQUEST are valid *)
IF ( (DS:RBX).RESERVED != 0) or (DS:RBX.KEYPOLICY.RESERVED != 0) )
    THEN #GP(0); FI;

TMP_CURRENTSECS ← CR_ACTIVE_SECS;

(* Determine which enclave attributes that must be included in the key. Attributes that must always be include INIT & DEBUG *)
REQUIRED_SEALING_MASK[127:0] ← 00000000 00000000 00000000 00000003H;
TMP_ATTRIBUTES ← (DS:RBX.ATTRIBUTEMASK | REQUIRED_SEALING_MASK) & TMP_CURRENTSECS.ATTRIBUTES;

CASE (DS:RBX.KEYNAME)
    SEAL_KEY:
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                goto EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
```

```
                goto EXIT;
        FI;
        // Include enclave identity?
        TMP_MRENCLAVE ← 0;
        IF (DS:RBX.KEYPOLICY.MRENCLAVE = 1)
            THEN TMP_MRENCLAVE ← TMP_CURRENTSECS.MRENCLAVE;
        FI;
        // Include enclave author?
        TMP_MRSIGNER ← 0;
        IF (DS:RBX.KEYPOLICY.MRSIGNER = 1)
            THEN TMP_MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
        FI;
        //Determine values key is based on
        TMP_KEYDEPENDENCIES.KEYNAME ← SEAL_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SEOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← TMP_MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        BREAK;
    REPORT_KEY:
        //Determine values key is based on
        TMP_KEYDEPENDENCIES.KEYNAME ← REPORT_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← 0;
        TMP_KEYDEPENDENCIES.ISVSVN ← 0;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SEOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_CURRENTSECS.ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← TMP_CURRENTSECS.MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
        TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← CR_CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← HARDCODED_PKCS1_5_PADDING;
        BREAK;
    EINITTOKEN_KEY:
        (* Check ENCLAVE has LAUNCH capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.LAUNCHKEY = 0)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ATTRIBUTE;
                goto EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                goto EXIT;
```

```
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
                goto EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME ← EINITTOKEN_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SEOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
        TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
        TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        BREAK;
    PROVISION_KEY: // Check ENCLAVE has PROVISIONING capability
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ATTRIBUTE;
                goto EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                goto EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
                goto EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME ← PROVISION_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← 0;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
        TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID ← 0;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← 0;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        BREAK;
```

```
    PROVISION_SEAL_KEY:
        (* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ATTRIBUTE;
                goto EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_CPUSVN;
                goto EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF ← 1;
                RAX ← SGX_INVALID_ISVSVN;
                goto EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME ← PROVISION_SEAL_KEY;
        TMP_KEYDEPENDENCIES.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.OWNEREPOCH ← 0;
        TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
        TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID ← 0;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
        BREAK;
    DEFAULT:
        (* The value of KEYNAME is invalid *)
        RFLAGS.ZF ← 1;
        RAX ← SGX_INVALID_KEYNAME;
        goto EXIT:
ESAC;

(* Calculate the final derived key and output to the address in RCX *)
TMP_OUTPUTKEY ← derivekey(TMP_KEYDEPENDENCIES);
DS:RCX[15:0] ← TMP_OUTPUTKEY;
RAX ← 0;
RFLAGS.ZF ← 0;

EXIT:
RFLAGS.CF ← 0;
RFLAGS.PF ← 0;
RFLAGS.AF ← 0;
RFLAGS.OF ← 0;
RFLAGS.SF ← 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is outside the DS segment limit.
	If KEYREQUEST format is invalid.
#PF(fault code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is not canonical.
	If KEYREQUEST format is invalid.
#PF(fault code)	If a page fault occurs in accessing memory operands.

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	SE1	This leaf function creates a cryptographic report of the enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX	RDX
IR	EREPORT (In)	Address of TARGETINFO (In)	Address of REPORTDATA (In)	Address where the REPORT is written to in an OUTPUTDATA (In)

Description

This leaf function creates a cryptographic REPORT that describes the contents of the enclave. This instruction leaf can only be executed when inside the enclave. The cryptographic report can be used by other enclaves to determine that the enclave is running on the same platform.

RBX contains the effective address of the MRENCLAVE value of the enclave that will authenticate the REPORT output, using the REPORT key delivered by EGETKEY command for that enclave. RCX contains the effective address of a 64-byte REPORTDATA structure, which allows the caller of the instruction to associate data with the enclave from which the instruction is called. RDX contains the address where the REPORT will be output by the instruction.

EREPORT Memory Parameter Semantics

TARGETINFO	REPORTDATA	OUTPUTDATA
Read access by Enclave	Read access by Enclave	Write access by Enclave

This instruction leaf perform the following:

1. Validate the 3 operands (RBX, RCX, RDX) are inside the enclave;
2. Compute a report key for the target enclave, as indicated by the value located in RBX(TARGETINFO);
3. Assemble the enclave SECS data to complete the REPORT structure (including the data provided using the RCX (REPORTDATA) operand);
4. Computes a crytpographic hash over REPORT structure;
5. Add the computed hash to the REPORT structure;
6. Output the completed REPORT structure to the address in RDX (OUTPUTDATA);

The instruction fails if the operands are not properly aligned

CR_REPORT_KEYID, used to provide key wearout protection, is populated with a statistically unique value on boot of the platform by a trusted entity within the SGX TCB

The instruction faults if any of the following:

EREPORT Faulting Conditions

An effective address not properly aligned	an memory address does not resolve in an EPC page
If accessing an invalid EPC page	If the EPC page is blocked
May page fault	

Operation

Temp Variables in EREPORT Operational Flow

Name	Type	Size (bits)	Description
TMP_ATTRIBUTES		32	Physical address of SECS of the enclave to which source operand belongs
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave
TMP_KEYDEPENDENCIES			Temp space for key derivation
TMP_REPORTKEY		128	REPORTKEY generated by the instruction
TMP_REPORT		3712	

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

(* Address verification for TARGETINFO (RBX) *)
IF ( (DS:RBX is not 128Byte Aligned) or (DS:RBX is not within CR_ELRANGE) or (DS:RBX does not resolve within an EPC) )
    Then GP(0); FI;

IF (EPCM(DS:RBX). VALID = 0)
    Then #GP(0); 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).ENCLAVEADDRESS != (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 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) or (DS:RCX does not resolve within an EPC) )
    Then GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    Then #GP(0); 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).ENCLAVEADDRESS != (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).R = 0) )
    THEN #GP(0);
FI;

(* Address verification for OUTPUTDATA (RDX) *)
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RDX is not within CR_ELRANGE) or (DS:RDX does not resolve within an EPC) )
    Then GP(0); FI;

IF (EPCM(DS:RDX). VALID = 0)
    Then #GP(0); FI;
```

```
IF (EPCM(DS:RDX).BLOCKED = 1) )
    THEN #PF(DS:RDX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RDX).PT != PT_REG) or (EPCM(DS:RDX).ENCLAVESECS != CR_ACTIVE_SECS) or
    (EPCM(DS:RDX).ENCLAVEADDRESS != (DS:RDX & ~0FFFH) ) or (EPCM(DS:RDX).W = 0) )
    THEN #GP(0);
FI;

(* REPORT MAC needs to be computed over data which cannot be modified *)
TMP_REPORT.CPUSVN ← CR_CPUSVN;
TMP_REPORT.ISVPRODID ← TMP_CURRENTSECS.ISVPRODID;
TMP_REPORT.ISVSVN ← TMP_CURRENTSECS..ISVSVN;
TMP_REPORT.ATTRIBUTES ← TMP_CURRENTSECS.ATTRIBUTES;
TMP_REPORT.REPORTDATA ← DS:RCX[511:0];
TMP_REPORT.MRENCLAVE ← TMP_CURRENTSECS.MRENCLAVE;
TMP_REPORT.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
TMP_REPORT.MRRESERVED ← 0;
TMP_REPORT.KEYID[255:0] ← CR_REPORT_KEYID;

(* Derive the report key *)
TMP_KEYDEPENDENCIES.KEYNAME ← REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVPRODID ← 0;
TMP_KEYDEPENDENCIES.ISVSVN ← 0;
TMP_KEYDEPENDENCIES.OWNEREPOCH ← CSR_SGX_OWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← DS:RBX.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
TMP_KEYDEPENDENCIES.MRENCLAVE ← DS:RBX.MEASUREMENT;
TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
TMP_KEYDEPENDENCIES.KEYID ← TMP_REPORT.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;

(* Calculate the derived key*)
TMP_REPORTKEY ← derive_key(TMP_KEYDEPENDENCIES);

(* call cryptographic CMAC function, CMAC data are not including MAC&KEYID *)
TMP_REPORT.MAC ← cmac(TMP_REPORTKEY, TMP_REPORTKEY[3071:0] );
DS:RDX[3455: 0] ← TMP_REPORT;
```

Flags Affected

None

Protected Mode Exceptions

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

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.

	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(fault code)	If a page fault occurs in accessing memory operands.

ERESUME—Re-Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLU[ERESUME]	IR	V/V	SE1	This leaf function is used to re-enter an enclave after an interrupt.

Instruction Operand Encoding

Op/En	RAX	RBX	RCX
IR	ERESUME (In)	Address of a TCS (In)	Address of AEP (In)

Description

The ENCLU[ERESUME] instruction resumes execution of an enclave that was interrupted due to an exception or interrupt, using the machine state previously stored in the SSA.

ERESUME Memory Parameter Semantics

TCS
Enclave read/write access

The instruction faults if any of the following:

Address in RBX is not properly aligned	Any TCS.FLAGS's must-be-zero bit is not zero
TCS pointed to by RBX is not valid or available or locked	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64
The SECS is in use by another enclave	Either of TCS-specified FS and GS segment is not a subset of the current DS segment
Any one of DS, ES, CS, SS is not zero	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM != 0x3
CR4.OSFXSR != 1	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0
Offsets 520-535 of the XSAVE area not 0	The bit vector stored at offset 512 of the XSAVE area must be a subset of SECS.ATTRIBUTES.XFRM
The SSA frame is not valid or in use	

If CR0.TS is set, ERESUME generates a #NM exception.

The following operations are performed by EENTER:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or an asynchronous exit due to any Interrupt event.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM.The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 7.1.2):
  - 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 7.2.6).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 7.2.4).
- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.

- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 7.2.4):
  - All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
  - PEBS is suppressed.
  - AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set
  - If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

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_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 GP(0); FI;

(* Check AEP is canonical*)
```

```
IF (TMP_MODE64 = 1 and (DS:RCX is not canonical) )
    Then #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other SGX instructions is operating on TCS)
    Then #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    Then #GP(0); 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 #GP(0); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    Then GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    Then GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS ← Address of SECS for TCS;

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    Then
        TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_FSLIMIT ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
        TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        TMP_GSLIMIT ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
        (* if FS wrap-around, make sure DS has no holes*)
        IF (TMP_FSLIMIT < TMP_FSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
        (* if GS wrap-around, make sure DS has no holes*)
        IF (TMP_GSLIMIT < TMP_GSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
    ELSE
        TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
            THEN #GP(0); FI;
FI;
```

```
(* Make sure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & & 0xFFFFFFFFFFFFFFFE) != 0)
    Then #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    Then #GP(0); FI;

(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 != TMP_SECS.ATTRIBUTES.MODE64BIT) )
    Then #GP(0); FI;

IF (CR4.OSFXSR = 0)
    Then #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    Then
        IF (TMP_SECS.ATTRIBUES.XFRM != 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUES.XFRM & XCR0) != TMP_SECS.ATTRIBUES.XFRM) THEN #GP(0); FI;
FI;

(* Make sure the SSA contains at least one active frame *)
IF ( (DS:RBX).CSSA = 0)
    Then #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA ← (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * ( (DS:RBX).CSSA - 1);
TMP_XSIZE ← compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;
    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        Then #GP(0); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        Then #GP(0); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        Then #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS != DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT != PT_REG) or
        (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS != EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_SECS).R = 0) or (EPCM(DS:TMP_SECS).W = 0) )
        Then #GP(0); 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 #GP(0); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
    Then #GP(0); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 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 #GP(0); FI;

IF (TMP_MODE64 = 0)
    Then
        IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) Then #GP(0); FI;
FI;

CR_GPR_PA ← Physical_Address (DS: TMP_GPR);

TMP_TARGET ← (DS:TMP_GPR).RIP;
IF (TMP_MODE64 = 1)
    Then
        IF (TMP_TARGET is not canonical) Then #GP(0); FI;
    ELSE
        IF (TMP_TARGET > CS limit) Then #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE))
    Then #GP(0); FI;

(* SECS.ATTRIBUTES.XFRM selects the features to be saved. *)
(* CR_XSAVE_PAGE_n: A list of 1 or more physical address of pages that contain the XSAVE area. *)
XRSTOR(TMP_MODE64, SECS.ATTRIBUTES.XFRM, CR_XSAVE_PAGE_n);

IF (XRSTOR failed with #GP)
    THEN
        DS:RBX.STATE ← INACTIVE;
        #GP(0);
FI;

CR_ENCALVE_MODE ← 1;
CR_ACTIVE_SECS ← TMP_SECS;
CR_ELRANGE ← (TMP_SECS.BASEADDR, TMP_SECS.SIZE);

(* Save sate for possible AEXs *)
CR_TCS_PA ← Physical_Address (DS:RBX);
CR_TCS_LA ← RBX;
CR_TCS_LA.AEP ← RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector ← FS.selector;
CR_SAVE_FS_base ← FS.base;
CR_SAVE_FS_limit ← FS.limit;
CR_SAVE_FS_access_rights ← FS.access_rights;
CR_SAVE_GS_selector ← GS.selector;
```

```
CR_SAVE_GS_base ← GS.base;
CR_SAVE_GS_limit ← GS.limit;
CR_SAVE_GS_access_rights ← GS.access_rights;

(* Set CR_ENCLAVE_ENTRY_IP *)
CR_ENCLAVE_ENTRY_IP ← CRIP"
RIP ← TMP_TARGET;

Restore_GPRs from DS:TMP_GPR;
Restore_Status_Control_Bits_of_RFLAGS_Except_TF from DS:TMP_GPR;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 ← XCR0;
    XCR0 ← TMP_SECS.ATTRIBUTES.XFRM;
FI;

(* Pop the SSA stack*)
(DS:RBX).CSSA ← (DS:RBX).CSSA -1;

(* Do the FS/GS swap *)
FS.base ← TMP_FSBASE;
FS.limit ← DS:RBX.FSLIMIT;
FS.type ← 0001b;
FS.W ← DS.W;
FS.S ← 1;
FS.DPL ← DS.DPL;
FS.G ← 1;
FS.B ← 1;
FS.P ← 1;
FS.AVL ← DS.AVL;
FS.L ← DS.L;
FS.unusable ← 0;
FS.selector ← 0BH;

GS.base ← TMP_GSBASE;
GS.limit ← DS:RBX.GSLIMIT;
GS.type ← 0001b;
GS.W ← DS.W;
GS.S ← 1;
GS.DPL ← DS.DPL;
GS.G ← 1;
GS.B ← 1;
GS.P ← 1;
GS.AVL ← DS.AVL;
GS.L ← DS.L;
GS.unusable ← 0;
GS.selector ← 0BH;

CR_DBGOPTIN ← TSC.FLAGS.DBGOPTIN;
Suppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
```

```
        Suppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        CR_SAVE_TF ← RFLAGS.TF;
        RFLAGS.TF ← 0;
        Suppress_monitor_trap_flag for the source of the execution of the enclave;
        Clear_all_pending_debug_exceptions;
        Clear_pending_MTF_VM_exit;
    ELSE
        Clear all pending debug exceptions;
        Clear pending MTF VM exits;
FI;

(* Assure consistent translations *)
Flush_linear_context;
Clear_Monitor_FSM;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If DS:RBX does not point to a valid TCS.
	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 one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM != 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(fault code)	If a page fault occurs in accessing memory.
#NM	If CR0.TS is set.

64-Bit Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If DS:RBX does not point to a valid TCS.
	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 one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM != 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(fault code)	If a page fault occurs in accessing memory operands.
#NM	If CR0.TS is set.

CHAPTER 6
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 SGX instructions are designed to work with legacy software and the various IA32 and Intel 64 modes of operation.

6.1 SGX AVAILABILITY IN VARIOUS PROCESSOR MODES

The SGX extensions (see Table 1-1) 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.

6.2 IA32_FEATURE_CONTROL

A new bit in IA32_FEATURE_CONTROL MSR (bit 18) is provided to BIOS to control the availability of SGX extensions. For 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 1.7.1 for additional details. OS is expected to examine the value of bit 18 prior to enabling SGX on the thread, as the settings of bit 18 is not reflected by CPUID.

6.3 INTERACTIONS WITH SEGMENTATION

6.3.1 Scope of Interaction

SGX extensions are available only when the processor is executing in a protected mode operation (see Section 6.1 for SGX availability in various processor modes). Enclaves abide by all the segmentation policies set up by the OS.

SGX interacts with segmentation at two levels:

- The SGX instruction (see the enclave instruction in Table 1-1), and
- logical-processor execution inside an enclave (legacy instructions and enclave instructions permitted inside an enclave).

6.3.2 Interactions of SGX Instructions with Instruction Prefixes and Addressing

All the memory operands used by the SGX instructions are interpreted as offsets within the data segment (DS). The segment-override prefix on SGX instructions is ignored.

Operand size is fixed for each enclave instruction. The operand-size prefix is reserved, and results in a #UD exception.

All address sizes are determined by the operating mode of the processor. The address-size prefix is ignored. This implies that while operating in 64-bit mode of operation, the address size is always 64 bits, and while operating in 32-bit mode of operation, the address size is always 32 bits. Additionally, when operating in 16-bit addressing, memory operands used by enclave instructions use 32 bit addressing; the value of CS.D is ignored.

6.3.3 Interaction of SGX Instructions with Segmentation

The SGX instructions used for entering the enclave (ENCLU[EENTER] and ENCLU[ERESUME]) ensure that all usable segment registers (i.e., the segment registers that have “Segment Unusable” bit in “Access Rights” field, a.k.a., “null” bit, set to 0) except for FS and GS have a zero base.

Additionally they save the existing contents of the FS/GS segment registers (including the hidden portion) in the processor, and load those registers with new values. The instructions also ensure that the linear ranges and access rights available under the newly-loaded FS and GS are subsets of the linear-address range/access rights available under DS. See EENTER Leaf and ERESUME Leaf in Chapter 5 for exact details of this computation.

Any exit from the enclave either via ENCLU[EEXIT] or via an AEX restores the saved values of FS/GS segment registers.

The enclave-entry instructions also ensure that the CS segment mode (64-bit vs 32 bit) is consistent with the segment mode for which the enclave was created, as indicated by the SECS.ATTRIBUTES.MODE64 bit, and that the CPL of the logical processor is 3.

Finally, all leaf functions of ENCLU and ENCLS instructions require that the DS segment be usable, and be an expand-up segment. Failing this check results in generation of a #GP(0) exception.

6.3.4 Interactions of Enclave Execution with Segmentation

During the course of execution, enclave code abides by all segmentation policies as dictated by legacy IA32 and Intel 64 Architectures, and generates appropriate exceptions on violations.

Additionally, any attempt by software executing inside an enclave to modify the processor's segmentation state (via MOV seg register, POP seg register, LDS, far jump, etc.) results in the generation of a #UD.

An exception to this rule is the use of WRFSBASE and WRGSBASE instruction from inside an enclave.-Execution of WRFSBASE and WRGSBASE from inside a 64-bit enclave does not generate the #UD exception. However if the software running inside an enclave modifies the segment-base values for these registers using the WRFSBASE and WRGSBASE instructions, the new values are not saved on the next enclave exit. Consequently, enclave writers are discouraged from using these instructions inside an enclave.

6.4 INTERACTIONS WITH PAGING

SGX instructions are available only when the processor is executing in a protected mode of operation. Additionally, all SGX leaf functions except for EDBGRD and EDBGWR are available only in paged mode of operation. Any attempt to execute these leaf functions in non-paged mode of operation results in delivery of #UD to the system software (OS or VMM).

All the memory operands passed into SGX instructions are interpreted as offsets within the data segments, and the linear addresses generated by combining these offsets with DS segment register are subject to paging-based access control, if paging is enabled at the time of the execution of the leaf function.

Since the ENCLU[EENTER] and ENCLU[EEXIT] can only be executed when paging is enabled, and since paging cannot be disabled by software running inside an enclave (recall that enclaves always run with CPL of 3), enclave execution is always subject to paging-based access control. The SGX access control itself is implemented as an extension to the traditional IA-32 and Intel 64 paging state machine. See Section 2.5 for details.

It should be noted that SGX instructions may set the A and D bit on non-faulting EPC pages, even if the instruction may eventually fault due to some other reason.

6.5 INTERACTIONS WITH VMX

All direct and indirect VM exits that originate on an instruction boundary inside an enclave set the “Enclave Interruption” bit in the VMCS. This bit is located in the “Guest Interruptibility State” field (field encoding 4824H, bit position 4) in the VMCS.

An SMI received while the processor is executing inside an enclave sets the “Enclave Interruption” bit (bit position 1) in the SMRAM at offset 7EE0H for opt-out delivery or the “Enclave Interruption” bit in the SMM Transfer VMCS (field encoding 4824H, bit position 4) for opt-in delivery.

If a VM entry with the VMCS “Enclave Interruption” bit set leads to a VM exit (due to vectored-event injection or VM-entry failures), then the VMCS “Enclave Interruption” bit is set on the resulting VM exit.

If a VM entry with the VMCS “Enclave Interruption” bit set or RSM with the SMRAM “Enclave Interruption” bit set pend an MTF VM exit, the resulting MTF VM exit sets the VMCS “Enclave Interruption” bit.

If a VM entry with the VMCS “Enclave Interruption” bit set or RSM with the SMRAM “Enclave Interruption” bit set, pend a debug exception, and if the delivery of that exception leads to a VM exit, either direct or indirect, then the resulting VM exit sets the VMCS “Enclave Interruption” bit.

If a VM entry with the VMCS “Enclave Interruption” bit set or RSM with the SMRAM “Enclave Interruption” bit set, pend a debug exception or an monitor trap flag event, and if the delivery of the debug exception/monitor trap flag event is preempted by a higher-priority event (SMI, INIT, or MC#), then the higher priority event sets the VMCS “Enclave Interruption” bit or the SMRAM “Enclave Interruption” if applicable.

VMM software may modify the VMCS “Enclave Interruption”. However, if the software sets both “Enclave Interruption” and MOV-SS blocking bits simultaneously, then the VM entry results in a late failure (new consistency check on the guest state). It should be noted that, since MOV-SS is an illegal instruction inside an enclave, hardware never sets these two bits simultaneously.

Additionally, VM entry leads to “failed VM-entry VM exit” on processors that enumerate CPUID.(EAX=07H, ECX=0H):EBX.SGX = 0. However, when CPUID.(EAX=07H, ECX=0H):EBX.SGX is 0, RSM ignores the value of the SMRAM.ENCLAVE_INTERRUPTION bit, and behaves as if the value is 0.

6.5.1 Availability of SGX under VMX

SGX functionality can be made available to software running in either VMX-root or VMX-non-root mode, as long as

- The software is not running in SMM mode of operation,
- The software is using a legal mode of operation (see Section 6.1), and
- CR4.SEE is set to 1.

A VMM can use various controls at its disposal for exposing/hiding SGX functionality to/from its guests.

6.5.2 Setting of CR4.SEE Bit under VMX Operation

Processors that support SGX functionality, i.e., CPUID.(EAX=07H, ECX=0):EBX.SGX = 1, also support setting of CR4.SEE bit. Such processors treat CR4.SEE as a flexible CR4 bit, and expose this fact to the VMM by reporting the bit at position 15 as 0 in IA32_VMX_CR4_FIXED0 MSR, and as 1 in IA32_VMX_CR4_FIXED1 MSR.

Note that processors that do not support SGX treat this bit as a RSVD(0) bit, and expose this fact to the VMM by reporting bit at position 15 as 0 in both IA32_VMX_CR4_FIXED0 and IA32_VMX_CR4_FIXED1 MSRs.

6.5.3 VMM Controls on Exposing SGX to the Guest

The SGX capability is primarily exposed to the software via CPUID instruction. VMMs can virtualize CPUID instruction to expose/hide this capability to/from guests.

Next, the various parameters related to SGX resources (such as EPC size, EPC location, etc.) are exposed/controlled via model-specific registers. VMMs can virtualize these MSRs for the guests using standard RDMSR/WRMSR hooks.

System software enables the SGX functionality by setting the CR4.SEE bit to 1. VMM can virtualize the CR4 register or use ENCLS/ENCLU execution control (see Section 6.5.4.2) if it wants to keep those guests from using SGX functionality.

The VMM can partition the Enclave Page Cache, and assign various partitions to (a subset of) its guests via the usual memory-virtualization techniques such as EPTs or shadow page tables.

The VMM can hook into the ENCLS instruction by setting bits in a new 64-bit VM-execution control field called "ENCLS Exiting" (encoding pair 0202EH/0202FH). When bits in this control field are set, execution of the corresponding ENCLS leaf functions in the guest results in a VM exit (see "ENCLS—Execute an Enclave System Function of Specified Leaf Number"). The basic Exit Reason in the VMCS is set to "ENCLS" (basic exit reason 60 (03cH)) and the VM-exit instruction length field is set to the length of the ENCLS instruction. The value of the VM-exit instruction information field is cleared.

6.5.4 VMX Capability Enumeration MSRs and SGX

A summary of the capability registers exposed is shown below

Table 6-1. Summary of VMX Capability Enumeration MSRS for Processors Supporting SGX

Interface	Description
IA32_VMX_CR4_FIXED0.[bit 15]	0
IA32_VMX_CR4_FIXED1.[bit 15]	Mirrors the value of CPUID.(EAX=07H, ECX=0).EBX.SGX
IA32_VMX_PROCBASED_CTLS2[bit 15]	0
IA32_VMX_PROCBASED_CTLS2[bit 47]	Mirrors the value of CPUID.(EAX=07H, ECX=0).EBX.SGX
IA32_VMX_MISC[bit 30]	If 1, VM entry checks that the VM-entry instruction length is in the range 0-15

6.5.4.1 Guest State Area - Guest Non-Register State

Table 6-2. Guest Interruptibility State

Position	Field	Value
0	Blocking by STI	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*
1	Blocking by MOV SS	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*
3	Blocking by SMI	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*
4	Blocking by NMI	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*
5	ENCLAVE_INTERRUPTION	See Section 6.5.8

6.5.4.2 VM-Execution Controls

VM-Execution controls related to SGX include a ENCLS-exiting field (accessed via VMCS encoding pair 0202EH/0202FH) and the Enable ENCLS/ENCLU control at bit 15 of the secondary processor based VM execution controls. The ENCLS exiting controls provides bit fields for VMM to permit which ENCLS leaf functions are permitted in a guest, see "ENCLS—Execute an Enclave System Function of Specified Leaf Number".

Table 6-3. Secondary Processor Based VM Execution Controls

Position	Field	Value
0 through 14	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*	
15	Enable ENCLS/ENCLU	
18	EPT-violation #VE	See Chapter 24 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*

6.5.4.3 Basic Exit Reasons

Table 6-4. Basic Exit Reasons

Basic Exit Reason	Value
0 through 59	See Appendix C of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*
60	ENCLS

6.5.5 VM Exits While Inside an Enclave

All VM exits that originate on an instruction boundary inside an enclave set a new bit called the "Enclave Interruption" bit (bit position 4) in the VMCS Guest Interruptibility State field (field encoding 4824H, Table 6-2) and in the EXIT_REASON field (bit 27) of the VMCS before delivering the VM exit to the VMM. Any VM exit (except for failed VM-entry VM exit) that sets the ENCLAVE_INTERRUPTION bit in GUEST_INTERRUPTIBILITY state, also sets Bit 27 in the EXIT_REASON field. These VM exit conditions include:

- Direct VM exits caused by exceptions, interrupts, and NMIs that happen while the logical processor is executing inside an enclave.
- Indirect VM exits triggered by interrupts, exceptions, and NMIs that happen while the logical processor is executing inside an enclave.
  - This includes VM exits encountered during vectoring due to EPT violations, task switch, etc.
- Parallel VM exits caused by SMI that is received while the logical processor is executing inside an enclave.
- All other VM exits that happen on an instruction boundary that is inside an enclave.

IA32/Intel 64 Architectures define very strict priority ordering between classes of events that are received on the same instruction boundary, and such ordering requires careful attention to cross-interactions between events. See Section 6.6 for details of interactions of architecturally visible events with SGX architecture.

All processor states saved in the VMCS on VM exits from an enclave contain synthetic state. See Table 4-2 for details of the state saved into the VMCS.

A failed VM-entry VM exit will not set the ENCLAVE_INTERRUPTION bit in EXIT_REASON but since it will not save the GUEST_INTERRUPTIBILITY_STATE, the original value of the ENCLAVE_INTERRUPTION bit will remain untouched in GUEST_INTERRUPTIBILITY_STATE.

6.5.6 VM Entry Consistency Checks and SGX

Processors that support SGX will enumerate IA32_VMX_MISC[30] as 1. This implies that, if the VM-entry interruption-information field in the current VMCS is such that the valid bit (bit 31) is 1 and its interruption type (bits 10:8) has value 4 (software interrupt), 5 (privileged software exception), or 6 (software exception), VM entry will not fail due to the VM-entry instruction-length field having value 0.

A VM entry will perform consistency checks according to those described in Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. Additional consistency checks on VM-entry control fields include:

- Checks on the CR4.SEE value in both the guest and the host state areas:
  - If the HOST.CR4.SEE bit is set inconsistently (i.e. this bit is 1 when CPUID.(EAX=07H, ECX=0):EBX.SGX = 0), the VM entry will fail. RFLAGS.ZF will be set to 1, and the control flow will transfer to the instruction after the one that attempted the VM entry (VMFailValid with reason code of Invalid Host State, 08h).
  - If the GUEST.CR4.SEE bit is set inconsistently (i.e. this bit is 1 when CPUID.(EAX=07H, ECX=0):EBX.SGX = 0), the VM entry will fail due to invalid guest state. The appropriate VM exit will be delivered to the VMM (Basic Reason Code 33).

6.5.7 VM Execution Control Setting Checks

A VM entry will perform consistency checks according to those described in Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. Additional consistency check on VM-execution control fields includes:

- If CPUID.(EAX=07H, ECX=0):EBX.SGX = 0, and if any of the bits in the "ENCLS Exiting" VM-execution control field (encoding pair 0202EH/0202FH) are set, then the VM entry fails. Failure sets RFLAGS.ZF=1 and error code=7.
- There is no consistency check on ENCLS Exiting field. There is a consistency check on bit 15 of secondary execution control. The ENCLS exiting field does not exist when CPUID.SE_LEAF.SGX is 0.

6.5.8 Guest Interruptibility State Checks

If the ENCLAVE_INTERRUPTION bit in VM-entry control field is set and if CPUID.(EAX=07H, ECX=0):EBX.SGX = 0, VM entry will fail.

If both the MOV-SS blocking and ENCLAVE_INTERRUPTION bits are set in the interruptibility-state field in the guest-state area of the VMCS, VM entry leads to a Failed VMENTRY/VMEXIT, error code 33. Note that, since the MOV SS and POP SS instructions are illegal inside an enclave, no VM exit will set the interruptibility-state field with both bits set.

If the ENCLAVE_INTERRUPTION bit is set in the interruptibility-state field of the VMCS, and a VM entry leads to a VMEXIT during event injection, then the VM exit sets the ENCLAVE_INTERRUPTION bit. Such a transition does not include an asynchronous enclave exit and consequently, neither the processor's architectural state, nor the state saved in the guest-state area of the VMCS is synthesized as is done during asynchronous enclave exits (for example: there is no clearing of the GPRs or of VMCS fields such as the VM-exit instruction length or the low 12 bits in certain address fields in the VMCS).

6.5.9 Interaction of SGX with Various VMMs

If IA32_VMX_MISC.[bit 30] = 0, permitted VM entry instruction lengths are 1-15 bytes. If IA32_VMX_MISC.[bit 30] = 1, permitted VM entry instruction lengths allow 0 as a legal value for interruption type 4(software interrupt), 5 (privileged software exception), or 6 (software exception).

6.5.10 Interactions with EPTs

SGX instructions are fully compatible with Extended Page Tables.

All the memory operands passed into SGX instructions are interpreted as offsets within the data segments, and the linear addresses generated by combining these offsets with DS segment register are subject to paging and EPT-based access control.

The SGX access control itself is implemented as an extension to the traditional IA-32 paging/EPT state machine. See Section 2.5 for details of this extension.

SGX instructions may set A and D bit on non-faulting EPC pages, even if the instruction may eventually fault due to some other reason, in IA page tables and EPT page tables when enabled.

6.5.11 Interactions with APIC Virtualization

The SGX architecture interacts with APIC virtualization due to its interactions with the APIC access page as well as Virtual APIC Page. See Section 6.11.2 for interactions of the SGX architecture with the Virtual APIC Page, and to Section 6.11.4 for the interactions of SGX architecture with the APIC Access Page.

6.5.12 Interactions with Monitor Trap Flag

The interactions of SGX with the Monitor Trap Flag are documented in Section 7.2.

6.6 SGX INTERACTIONS WITH ARCHITECTURALLY-VISIBLE EVENTS

All architecturally visible vectored events (IA32 exceptions, interrupts, SMI, NMI, INIT, VM exit) that are detected while inside an enclave cause an asynchronous enclave exit. Additionally, INT3, entry/redirection, and the SignalTXTMsg[SENTER] events also cause asynchronous enclave exits. Note that SignalTXTMsg[SEXIT] does not cause an AEX.

On an AEX, information about the event causing the AEX is stored in the SSA (see Section 4.4 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.

6.7 INTERACTIONS WITH THE XSAVE/XRSTOR PROCESSOR EXTENDED STATES

6.7.1 Requirements and Architecture Overview

Processor extended states are the ISA features that are enabled by the settings of CR4.OSXSAVE and the XCR0 register. Processor extended states are normally saved/restored by software via XSAVE/XRSTOR instructions. Details of discovery of processor extended states and management of these states are described in CHAPTER 13 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Additionally, the following requirements apply to SGX:

- On an AEX, the SGX architecture must protect the processor extended state in the state-save area (SSA), and clear the secrets in the processor extended state, if the extended state is being used by an enclave.
- SGX architecture must ensure that erroneous XCR0 and/or XBV_HEADER settings by system software do not result in SSA overflow.
- Enclave software should be able to discover only those processor extended states for which such protection is enabled.
- The processor extended states that are enabled inside the enclave must form an integral part of the enclave's identity. This requirement has two implications:
  - Certain processor extended state (e.g., Memory Protection Extensions, see Chapter 9 of *Intel® Architecture Instruction Set Extensions Programming Reference*) modify the behavior of the legacy ISA software. If such features are enabled for enclaves that do not understand those features, then such a configuration could lead to a compromise of the enclave's security.
  - Service providers may decide to assign different trust level to the same enclave depending on the ISA features the enclave is using.

To meet these requirements, the SGX architecture defines a sub-field called X-feature Request Mask (XFRM) in the ATTRIBUTES field of the SECS. On enclave entry, after certain consistency checks, the value in the XCR0 is saved in a micro-architectural location, and is replaced by the XFRM. On enclave exit, the original value of XCR0 is restored. Consequently, while inside the enclave, the processor extended states enabled in XFRM are in enabled state, and those that are disabled in XFRM are in disabled state. The entire ATTRIBUTES field, including the XFRM subfield is integral part of enclave's identity (i.e., its value is included in reports generated by ENCLU[EREPORT], and select bits from this field can be included in key-derivation for keys obtained via ENCLU[EGETKEY]).

On an asynchronous enclave exit, the processor extended states enabled by XFRM are saved in the current SSA frame, and overwritten by synthetic state (see Section 4.3 for the definition of the synthetic state). When the interrupted enclave is resumed via ENCLU[ERESUME], the saved state for processor extended states enabled by XFRM is restored.

6.7.2 Relevant Fields in Various Data Structures

6.7.2.1 SECS.ATTRIBUTES.XFRM

The ATTRIBUTES field of the SECS data structure (see Section 2.6.1) contains a sub-field called X-Feature Request Mask (XFRM). Software populates this field at the time of enclave creation indicating the processor extended state configuration required by the enclave.

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 6.7.3 through Section 6.7.6.

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 SGX requires that SSE be enabled (CR4.OSFXSR = 1).

6.7.2.2 SECS.SSAFRAMESIZE

The SSAFRAMESIZE field in the SECS data structure specifies the number of pages which software allocated[1] for each SSA frame, including both the GPRSGX area and the XSAVE area (x87 and XMM states are stored in the latter area). The specified size must be large enough to hold all the general-purpose registers, additional SGX specific information, plus the state size of set of processor extended states specified by SECS.ATTRIBUTES.XFRM (see Section 2.6.3 for the layout of SSA). The SSA is always in non-compacted format.

If the processor does not support XSAVE, the XSAVE area will always be 576 bytes; a copy of XFRM (which will be set to 0x3) is saved at offset 512 on an AEX.

If the processor does support XSAVE, the length of the XSAVE area depends on SECS.ATTRIBUTES.XFRM. The length would be equal to what CPUID.(EAX=0DH, ECX= 0):EBX returns if XCR0 were set to XFRM. The following pseudo code illustrates how software can calculate this length using XFRM as the input parameter without modifying XCR0:

```
offset = 576;
size_last_x = 0;
For x=2 to 63
IF (XFRM[x] != 0) Then
   tmp_offset = CPUID.(EAX=0DH, ECX= x):EBX[31:0];
    IF (tmp_offset >= offset + size_last_x) Then
        offset = tmp_offset;
        size_last_x = CPUID.(EAX=0DH, ECX= x):EAX[31:0];
     FI;
FI;
EndFor
return (offset + size_last_x); (* compute_xsave_size(XFRM), see “ECREATE—Create an SECS page in the Enclave Page Cache”*)
```

Where the non-zero bits in XFRM are a subset of non-zero bit fields in XCR0.

1. It is the responsibility of the enclave to actually allocate this memory.

6.7.2.3 XSAVE Area in SSA

The XSAVE area of an SSA frame begins at offset 0 of the frame.

6.7.3 Processor Extended States and ENCLS[ECREATE]

The ECREATE leaf of the ENCLS instruction enforces a number of consistency checks described earlier. The execution of ENCLS[ECREATE] instruction results in a #GP(0) exception in any of the following cases:

- SECS.ATTRIBUTES.XFRM[1:0] is not 3.
- The processor does not support XSAVE and any of the following is true:
  - SECS.ATTRIBUTES.XFRM[63:2] is not 0,
  - SECS.SSAFRAMESIZE is 0.
- The processor supports XSAVE and any of the following is true:
  - XSETBV would fault on an attempt to load XFRM into XCR0.
  - XFRM[63]=1.
  - SSAFRAMESIZE*4096 < 168 + X, where X is the value that would be returned in EBX if CPUID were executed with EAX=0DH, ECX=0, and XCR0 was loaded with the value of XFRM.

6.7.4 Processor Extended States and ENCLU[EENTER]

6.7.4.1 Fault Checking

The EENTER leaf of ENCLU instruction enforces a number of consistency requirements described earlier. Specifically, the ENCLU[EENTER] instruction results in a #GP(0) exception in any of the following cases:

- CR4.OSFXSR=0.
- The processor supports XSAVE and either of the following is true
  - CR4.OSXSAVE=0 and SECS.ATTRIBUTES.XFRM is not 3.
  - (SECS.ATTRIBUTES.XFRM & XCR0) != SECS.ATTRIBUTES.XFRM

6.7.4.2 State Loading

If ENCLU[EENTER] is successful, it saves the current value of XCR0 in a micro-architectural location and sets XCR0 to SECS.ATTRIBUTES.XFRM.

6.7.5 Processor Extended States and AEX

6.7.5.1 State Saving

On an AEX, processor extended states are saved into the XSAVE area of the SSA frame as if the XSAVE instruction was executed with EDX:EAX = SECS.ATTRIBUTES.XFRM, with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if REX.W=1. The XSTATE_BV part of the XSAVE header is saved with 0 for every bit that is 0 in XFRM. Other bits may be saved as 0 if the state saved is initialized.

Note that enclave entry ensures that if CR4.OSXSAVE is set to 0, then SECS.ATTRIBUTES.XFRM is set to 3. It should also be noted that it is not possible to enter an enclave with FXSAVE disabled. While AEX is defined to save data as XSAVE would, implementations may use FXSAVE flows if CR4.OSXSAVE=0. In this case, the implementation ensures that the non-state data is consistent with the XSAVE format, and not the FXSAVE format (e.g., the XSAVE header).

6.7.5.2 State Synthesis

After saving state, AEXs restore XCR0 to the value it held at the time of the most recent enclave entry.

The state of features corresponding to bits set in XFRM is synthesized. In general, these states are initialized. Details of state synthesis on AEX are documented in Section 4.3.1.

6.7.6 Processor Extended States and ENCLU[ERESUME]

6.7.6.1 Fault Checking

The ERESUME leaf of ENCLU instruction enforces a number of consistency requirements described earlier. Specifically, the ENCLU[ERESUME] instruction results in a #GP(0) exception in any of the following cases:

- CR4.OSFXSR=0.
- The processor supports XSAVE and either of the following is true
  - CR4.OSXSAVE=0 and SECS.ATTRIBUTES.XFRM is not 3.
  - (SECS.ATTRIBUTES.XFRM & XCR0) != SECS.ATTRIBUTES.XFRM.

A successful execution of ENCLU[ERESUME] loads state from the XSAVE area of the SSA frame in a fashion similar to that used by the XRSTOR instruction. Data in the XSAVE area that would cause the XRSTOR instruction to fault will cause the ENCLU[ERESUME] instruction to fault. Examples include the following:

- A bit is set in the XSTATE_BV field and clear in XFRM.
- The required bytes in the header are not clear.
- Loading data would set a reserved bit in MXCSR.

Any of these conditions will cause ERESUME to fault, even if CR4.OSXSAVE=0. In this case, it is the responsibility of the processor to generate faults that are caused by XRSTOR and not by FXRSTOR.

6.7.6.2 State Loading

If ENCLU[ERESUME] is successful, it saves the current value of XCR0 microarchitecturally and sets XCR0 to XFRM.

State is loaded from the XSAVE area of the SSA frame as if the XRSTOR instruction were executed with XCR0=XFRM, EDX:EAX = XFRM, with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if REX.W=1. The XSTATE_BV part of the XSAVE header is saved with 0 for every bit that is 0 in XFRM, as a non-compacted buffer. Other bits may be saved as 0 if the state saved is initialized.

ENCLU[ERESUME] ensures that a subsequent execution of XSAVEOPT inside the enclave will operate properly (e.g., by marking all state as modified).

6.7.7 Processor Extended States and ENCLU[EEXIT]

The ENCLU[EEXIT] instruction does not perform any X-feature specific consistency checks. However, successful execution of the ENCLU[EEXIT] instruction restores XCR0 to the value it held at the time of the most recent enclave entry.

6.8 INTERACTIONS WITH SMM

6.8.1 Availability of SGX instructions in SMM

Enclave instructions are not available in SMM, and any attempt to execute ENCLS or ENCLU instructions inside SMM results in a #UD exception.

Transfer to SMM handler when dual-monitor treatment is not enabled saves and clears all bits of CR4, including the CR4.SEE bit. Software is allowed to set the CR4.SEE bit on any processor that enumerates SGX feature flag in

CPUID as 1, and software can set this bit in any processor mode that allows modification of CR4, even if the SGXISA extensions are not available in that particular mode of operation. Consequently, SMM software is allowed to set the CR4.SEE bit without causing any faults, if CPUID.(EAX=07H, ECX=0):EBX.SGX as 1.

6.8.2 SMI while Inside an Enclave

The response to an SMI received while executing inside an enclave depends on whether the dual-monitor treatment is enabled. For detailed discussion of transfer to SMM, see Chapter 34, “System Management Mode” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*.

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is not enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM handler. In addition to saving the synthetic architectural state to the SMRAM State Save Map (SSM), the logical processor also sets the “Enclave Interruption” bit in the SMRAM SSM (bit position 1 in SMRAM field at offset 7EE0H). Transfer to SMM saves the value of CR4 in a reserved location in the SSM, and clears all the bits in CR4-this includes the CR4.SEE bit.

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 Guest Interruptibility State field of the SMM transfer VMCS.

6.8.3 SMRAM Synthetic State of AEX Triggered by SMI

All processor registers saved in the SMRAM have the same synthetic values listed in Section 4.3. Additional SMRAM fields that are treated specially on SMI are:

Table 6-5. SMRAM Synthetic States on Asynchronous Enclave Exit

Position	Field	Value
SMRAM Offset 07EE0H.Bit 1	ENCLAVE_INTERRUPTION	Set to 1 if exit occurred in enclave mode

6.9 INTERACTIONS OF INIT, SIPI, AND WAIT-FOR-SIPI WITH SGX

INIT received inside an enclave, while the logical processor is not in VMX operation, causes the logical processor to exit the enclave asynchronously. After the AEX, the processor's architectural state is initialized to “Power-on” state (Table 9.1 in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). If the logical processor is BSP, then it proceeds to execute the BIOS initialization code. If the logical processor is an AP, it enters Wait-for-SIPI (WFS) state.

INIT received inside an enclave, while the logical processor (LP) is in VMX-root operation, is blocked until either the LP exits VMX operation (via VMXOFF) or enters VMX-non-root operation (via VMLAUNCH or VMRESUME). Since VMXOFF, VMLAUNCH, and VMRESUME cause a CPL-based #GP inside an enclave, such an INIT remains blocked at least until the LP exits the enclave.

INIT received inside an enclave, while the logical processor is in VMX-non-root operation, causes an AEX. Subsequent to the AEX, the INIT is delivered to the VMM via appropriate VM exit with INSIDE_ENCLAVE bit in the VMCS.EXIT_REASON set.

A processor cannot be inside an enclave in WFS state. Consequently, a SIPI received while inside an enclave is lost.

If a processor is in WFS state outside VMX operation, receipt of SIPI vectors the processor to 000VV000H to run BIOS-initialization code. If a processor is in WFS state in VMX-non-root operation, receipt of SIPI causes the LP to deliver appropriate VM exit. A processor cannot be in WFS state in VMX-root operation. In either case, the behavior of the LP on SIPI while in WFS state does not change for SGX.

INIT is considered a warm reset, which keeps all the cache state, RR state, and feature-configuration state unmodified. Consequently, subsequently to INIT, CPUID enumeration of SGX feature remains intact.

The SGX-related processor states after INIT-SIPI-SIPI is as follows:

- PRMRR: Unchanged
- EPCM: Unchanged
- CPUID.LEAF_12H.*: Unchanged
- ENCLAVE_MODE: 0 (LP exits enclave asynchronously)
- CR4.SEE: 0
- MEE state: Unchanged

OSes that use INIT-SIPI-SIPI only during initial boot (i.e., only after reset) can blindly assume that the entire EPC is empty, and every entry in the EPCM is marked invalid. These OSes do not need to use EREMOVE after INIT-SIPI-SIPI.

OSes that use INIT-SIPI-SIPI for dynamic offlining of a processor should use software conventions for communicating the EPCM and other state with the processors that are offlined/onlined dynamically.

6.10 INTERACTIONS WITH DMA

DMA is not allowed to access any Processor Reserved Memory.

6.11 INTERACTIONS WITH MEMORY CONFIGURATION AND VARIOUS MEMORY RANGES

6.11.1 Memory Type Considerations for PRMRR

All enclave accesses to the PRMRR region always use the memory type specified by the PRMRR, unless the CR0.CD bit on one of the logical processors on the core running the enclave is set. In other words, PRMRR memory type overrides memory types coming from overlapping MTRRs and all other architectural range registers, and those coming from PAT and EPTs. All non-enclave accesses to PRMRR region result in abort-page semantics, while all enclave code fetch access to non-PRMRR region result in a #GP(0) exception (see Section 2.3 for description of Access Control).

The TYPE field in the PRMRR_BASE register can only be programmed with values UC(0x0) and WB (0x6). Any attempt to write a value other than these two to the TYPE field of the PRMRR_BASE MSR results in #GP.

At power-on, all bits in PRMRR_BASE are initialized to 0 and mask.

6.11.2 Interactions of PRMRR with Various Memory Regions

6.11.2.1 Interactions of PRMRR with SMRR

SMRR and PRMRR are not allowed to overlap, if SMRR is valid and PRMRR is configured (locked). If either SMRR or PRMRR is written such that the SMRR pair will be valid and the range of SMRR overlaps with the configured (locked) value of PRMRR, then a #GP(0) is signaled. Programming non-contiguous SMRR or PRMRR will cause #GP.

6.11.2.2 Interactions of PRMRR with MTRRs

MTRRs are allowed to overlap with PRMRR. However, for all legal memory accesses to PRMRR region, PRMRR memory type overrides the MTRR memory type.

6.11.2.3 Interactions of PRMRR with MMIO

The MMIO region is not allowed to overlap with PRMRR. If MMIO region overlaps with PRMRR it results in CPUID.(EAX=12H, ECX=0):EAX[bit 0] = 0 (i.e. enclave instructions not available).

6.11.2.4 Interactions of PRMRR with IA32_APIC_BASE

IA32_APIC_BASE and PRMRR are not allowed to overlap if PRMRR is configured (locked). On a write to either IA32_APIC_BASE or PRMRR, if the page defined by IA32_APIC_BASE overlaps with the configured (locked) value of PRMRR, then a #GP(0) is signaled.

6.11.3 Interactions of PRMRR with Virtual APIC Page

Virtual-APIC Page is allowed to overlap with PRMRR. However, if PRMRR overlaps with the Virtual APIC page, then all accesses to Virtual APIC page result in abort-page semantics. See the discussion on PRMRR interactions with Physical Memory Accesses for more details.

6.11.3.1 Interactions of PRMRR with Physical Memory Accesses

Physical memory accesses can be classified into enclave physical accesses and non-enclave physical accesses. All enclave physical memory accesses to an EPC section within PRMRR region are guaranteed to go to the EPC memory, while all non-enclave physical memory accesses result in abort-page behavior.

Enclave physical accesses consist of the implicit memory accesses performed by certain SGX instructions to the enclave SECS, TCS, or SSA by following the physical addresses cached by the processor at various times. See Section 2.3 for the discussion of explicit vs. implicit accesses.

A memory access is called a non-enclave physical access if

1. either (a) the access is generated from software that is not running as a VMX guest; or (b) the “enable EPT” VM-execution control is 0; or (c) the access's physical address is not the result of a translation through the EPT paging structures; and
2. either (a) the access is not generated by a linear address; or (b) the access's physical address is not the translation of its linear address.

Non-enclave physical accesses include the following:

- If VMX is disabled, or if the logical processor is executing in VMX-root mode of operation, or if the “enable EPT” VM-execution control is 0
  - — Reads from the paging structures when translating a linear address.
  - — Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging.
  - — Updates to the accessed and dirty bits in the paging structures.
- If VMX is enabled, logical processor is executing in VMX-non-root mode of operation, and the “enable EPT” VM-execution control is 1, accesses to the EPT paging structures.
- Any of the following accesses made by the processor to support VMX non-root operation:
  - — Accesses to the VMCS region.
  - — Accesses to data structures referenced (directly or indirectly) by physical addresses in VM-execution control fields in the VMCS. These include the I/O bitmaps, the MSR bitmaps, VE info area, VMFUNC list and the virtual-APIC page.
- Accesses that effect transitions into and out of SMM. These include the following:
  - — Accesses to SMRAM during SMI delivery and during execution of RSM.
  - — Accesses during SMM VM exits (including accesses to MSEG) and during VM entries that return from SMM.

6.11.4 Interactions of SGX with APIC Access Address

A memory access by an enclave instruction that implicitly uses a cached physical address is never checked for overlap with the APIC-access page. Such accesses never cause APIC-access VM exits and are never redirected to the virtual-APIC page. Implicit memory accesses can only be made to the SECS, the TCS, or the SSA of an enclave (see Section 2.3). Consequently, all implicit accesses are always targeted at a page inside an EPC.

For all other memory accesses, physical-address matches against the APIC-access address occur before checking for other range register matches.

A memory access that is either made by an enclave instruction or from a logical processor inside an enclave that would have caused redirection to the virtual-APIC page instead causes an APIC-access VM exit.

Other than implicit accesses made by enclave instructions, guest-physical and physical accesses are not considered “enclave accesses”; consequently, such accesses results in abort-page semantics if these accesses eventually reach PRMRR range. This applies to any physical accesses that are redirected to the virtual-APIC page.

While a logical processor is inside an enclave, the checking of the instruction pointer's linear address against the enclave's linear-address range (ELRANGE) is done before checking the physical address to which the linear address translates against the APIC-access address. Thus, an attempt to execute an instruction outside ELRANGE, the instruction fetch results in a #GP(0), even if the linear address would translate to a physical address that overlaps the APIC access address.

6.12 INTERACTIONS WITH TXT

6.12.1 Enclaves Created Prior to Execution of GETSEC

Enclaves which have been created before the GETSEC[SENTER] instruction are available for execution after the successful completion of GETSEC[SENTER] and the corresponding SINIT ACM. GETSEC[SENTER] forces CR4.SEE =0 on all logical processors. SGX will need to be re-enabled by the software launched by GETSEC[SENTER], in addition to any other actions a TXT launched environment performs when preparing to execute code which was running previously to GETSEC[SENTER].

6.12.2 Interaction of GETSEC with SGX

All leaf functions of the GETSEC instruction are illegal inside an enclave, and result in #UD.

Responding Logical Processors (RLP) which are executing inside an enclave at the time a GETSEC[SENTER] event occurs perform an AEX from the enclave and then enter the Wait-for-SIPI state.

RLP threads executing an enclave at the time of GETSEC[SEXIT], behave as defined for GETSEC[SEXIT]-that is, the RLPs pause during execution of SEXIT and resume after the completion of SEXIT.

The execution of a TXT launch does not affect SGX configuration or security parameters.

Processors supporting SGX also require that the ACM-verification key be located on die, and that such ACMs contain a new header field.

6.13 INTERACTIONS WITH CACHING OF LINEAR-ADDRESS TRANSLATIONS

Entering and exiting an enclave causes the logical processor to flush all the global linear-address context as well as the linear-address context associated with the current VPID and PCID. The MONITOR FSM is also cleared.

6.14 INTERACTIONS WITH INTEL® TRANSACTIONAL SYNCHRONIZATION EXTENSIONS (INTEL® TSX)

1. ENCLU or ENCLS instructions inside an HLE region will cause the flow to be aborted and restarted non-speculatively. ENCLU or ENCLS instructions inside an RTM region will cause the flow to be aborted and transfer control to the fallback handler.

2. If XBEGIN is executed inside an enclave, the processor does NOT check whether the address of the fallback handler is within the enclave.

3. If an RTM transaction is executing inside an enclave and there is an attempt to fetch an instruction outside the enclave, the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

4. If an RTM transaction is executing inside an enclave and there is a data access to an address within the enclave that denied due to EPCM content (e.g., to a page belonging to a different enclave), the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

5. If an RTM transaction executing inside an enclave aborts and the address of the fallback handler is outside the enclave, a #GP is delivered after the abort (EIP reported is that of the fallback handler).

6.14.1 HLE and RTM Debug

RTM debug will be suppressed on opt-out enclave entry. After opt-out entry, the logical processor will behave as if IA32_DEBUG_CTL[15]=0. Any #DB detected inside an RTM transaction region will just cause an abort with no exception delivered. After opt-in entry, if either DR7[11] = 0 OR IA32_DEBUGCTL[15] = 0, any #DB or #BP detected inside an RTM transaction region will just cause an abort with no exception delivered. After opt-in entry, if DR7[11] = 1 AND IA32_DEBUGCTL[15] = 1, any #DB or #BP detected inside an RTM translation will terminate speculative execution, set RIP to the address of the XBEGIN instruction, and be delivered as #DB (any #BP is converted to #DB) - imply an SGX AEX. DR6[16] will be cleared, indicating RTM debug (if the #DB causes a VM exit, DR6 is not modified but bit 16 of the pending debug exceptions field in the VMCS will be set).

6.15 SGX INTERACTIONS WITH S STATES

Whenever SGX enabled processor leaves the S0 or S1 state for S2-S5 state, enclaves are destroyed. This is due to the EPC being destroyed when power down occurs.

6.16 SGX INTERACTIONS WITH MACHINE CHECK ARCHITECTURE (MCA)

6.16.1 Interactions with MCA Events

All architecturally visible machine check events (#MC and CMCI) that are detected while inside an enclave cause an asynchronous enclave exit.

Any machine check exception (#MC) that occurs after SGX is first enables causes SGX to be disabled, (CPUID.SE_Leaf.0.EAX[SE1] == 0) . It cannot be enabled until after the next reset.

6.16.2 Machine Check Enables (IA32_MCi_CTL)

All supported IA32_MCi_CTL bits for all the machine check banks must be set for SGX to be available (CPUID.SE_Leaf.0.EAX[SE1] == 1). Any act of clearing bits from '1 to '0 in any of the IA32_MCi_CTL register may disable SGX (set CPUID.SE_Leaf.0.EAX[SE1] to 0) until the next reset.

If SGX instructions are disabled, PRMRR memory protections remain intact and threads in enclave mode may be forced to exit (e.g. when attempting to execute an SE instruction).

6.16.3 CR4.MCE

CR4.MCE can be set or cleared with no interactions with SGX.

CHAPTER 7
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.

7.1 CONFIGURATION AND CONTROLS

7.1.1 Debug Enclave vs. Production Enclave

The SECS of each enclave provides a bit, SECS.ATTRIBUTES.DEBUG, indicating whether the enclave is a debug enclave (if set) or a production enclave (if 0). If this bit is set, software outside the enclave can use EDBGRD/EDBGWR to access the EPC memory of the enclave. The value of DEBUG is not included in the measurement of the enclave and therefore doesn't require a special SIGSTRUCT to be generated for this matter.

The ATTRIBUTES field in the SECS is reported in the enclave's attestation, and is included in the key derivation for the enclave secrets that were protected by the enclave using 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 5.4), while that belonging to a non-debug enclave cannot be accessed by these leaf functions.

7.1.2 Tool-chain Opt-in

The TCS.FLAGS.DBGOPTIN bit controls interactions of certain debug and profiling features with enclaves, including code/data breakpoints, TF, RF, monitor trap flag, BTF, LBRs, BTM, BTS, and performance monitoring. This bit is forced to zero when EPC pages are added via EADD. A debugger can set this bit via EDBGWR to the TCS of a debug enclave.

An enclave entry through a TCS with the TCS.FLAGS.DBGOPTIN set to 0 is called an **opt-out entry**. Conversely, an enclave entry through a TCS with TCS.FLAGS.DBGOPTIN set to 1 is called an **opt-in entry**.

7.2 SINGLE STEP DEBUG

7.2.1 Single Stepping Requirements

The following requirements are identified for the single-stepping architecture:

- The privileged SGX instruction ENCLS must exhibit legacy single-stepping behavior.
- If a debugger is not debugging an enclave, then the enclave should appear as a “giant instruction” to the debugger.
- The architecture must allow an SGX-capable debugger and a debug enclave to single-step within an enclave that it wants to debug in a fashion that is consistent with the IA32/Intel 64 legacy prior to the introduction of SGX.

7.2.2 Single Stepping ENCLS Instruction Leafs

If the RFLAGS.TF bit is set at the beginning of ENCLS, then a single-step debug exception is pending on the instruction boundary immediately after the ENCLS instruction. Additionally, if the instruction is invoked from a VMX guest, and if the monitor trap flag is asserted at the time of the time of invocation, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

7.2.3 Single Stepping ENCLU Instruction Leafs

The interactions of the unprivileged SGX instruction ENCLU are leaf dependent.

An enclave entry via EENTER/ERESUME leaf functions of the ENCLU, in certain cases, may clear the RFLAGS.TF bit, and suppress the monitor trap flag. In such situations, an exit from the enclave, either via the EEXIT leaf function or via an AEX restores the RFLAGS.TF bit and effectiveness of the monitor trap flag. The details of this clearing/suppression and the exact pending of single stepping events across EENTER/ERESUME/EEXIT/AEX are covered in detail in Section 7.2.4.

If the RFLAGS.TF bit is set at the beginning of EREPORT or EGETKEY leafs, then a single-step debug exception is pending on the instruction boundary immediately after the ENCLU instruction. Additionally, if the instruction is invoked from a VMX guest, and if the monitor trap flag is asserted at the time of invocation, and if the monitor trap flag is not suppressed by the preceding enclave entry, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

Consistent with the IA32 and Intel 64 architectures, a pending MTF VM exit takes priority over a pending debug exception. Additionally, if an SMI, an INIT, or an #MC is received on the same instruction boundary, then that event takes priority over both the pending MTF VM exit and the pending debug exception. In such a situation, the pending MTF VM exit and/or pending debug exception are handled in a manner consistent with the IA32 and Intel 64 architectures.

If the instruction under consideration results in a fault, then the control flow goes to the fault handler, and no single-step debug exception is asserted. In such a situation, if the instruction is executed from a VMX guest, and if the VMM has asserted the monitor trap flag, then an MTF VM exit is pending after the delivery of the fault through the IDT (i.e., before the first instruction of the OS handler). If a VM exit occurs before reaching that boundary, then the MTF VM exit is lost.

7.2.4 Single-stepping Enclave Entry with Opt-out Entry

7.2.4.1 Single Stepping without AEX

Figure 7-1 shows the most common case for single-stepping after an opt-out entry.

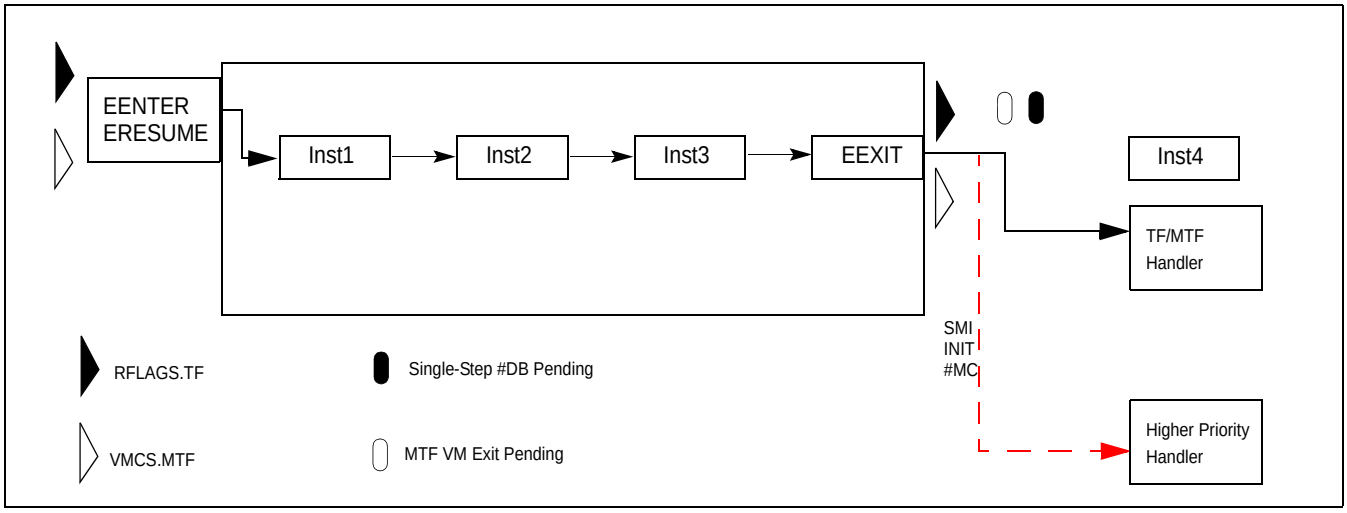

Figure 7-1. Single Stepping with Opt-out Entry - No AEX

In this scenario, if the RFLAGS.TF bit is set at the time of the enclave entry, then a single step debug exception is pending on the instruction boundary after EEXIT. Additionally, if the enclave is executing in a VMX guest, and if the monitor trap flag is asserted at the time of the enclave entry, then an MTF VM exit is pending on the instruction boundary after EEXIT.

The value of the RFLAGS.TF bit at the end of EEXIT is same as the value of RFLAGS.TF at the time of the enclave entry. Similarly, if the enclave is executing inside a VMX guest, then the value of the monitor trap flag after EEXIT is same as the value of that control at the time of the enclave entry.

Consistent with the IA32 and Intel 64 architectures, MTF VM exit, if pending, takes priority over a pending debug exception. If an SMI, an INIT, or an MC# is received on the same instruction boundary, then that event takes priority over both the pending MTF VM exit and the pending debug exception. In such a situation, the pending MTF VM exit and/or pending debug exception are handled in a manner consistent with the IA32 and Intel 64 architecture.

7.2.4.2 Single Step Preempted by AEX due to Non-SMI Event

Figure 7-2 shows the interaction of single stepping with AEX due to a non-SMI event after an opt-out entry.

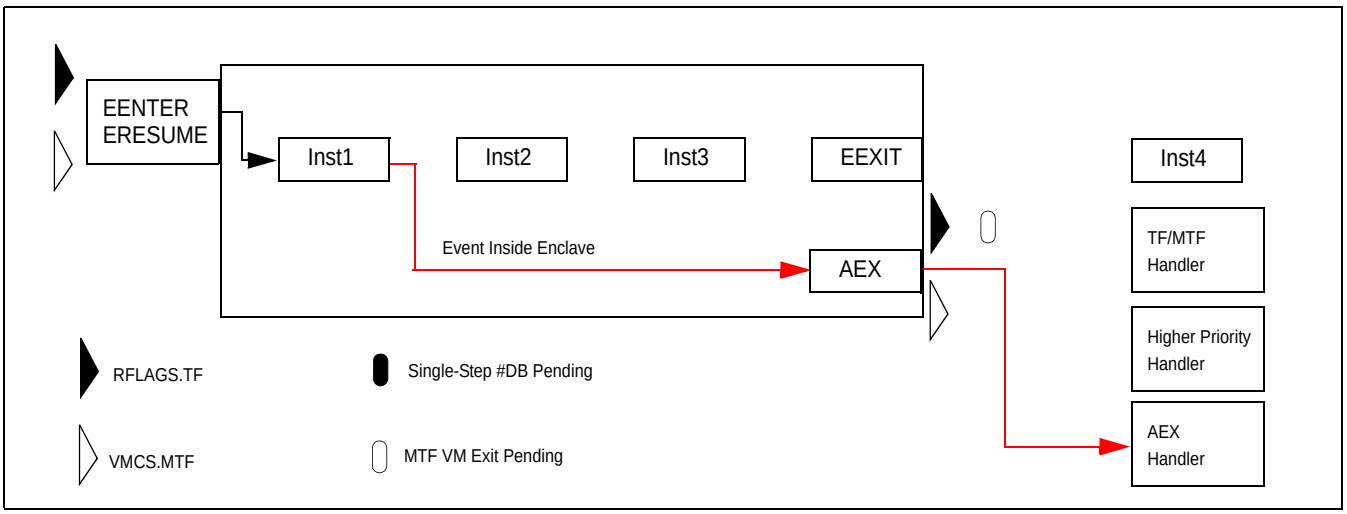

Figure 7-2. Single Stepping with Opt-out Entry -AEX Due to Non-SMI Event Before Single-Step Boundary

In this scenario, if the enclave is executing in a VMX guest, and if the monitor trap flag is asserted at the time of the enclave entry, then an MTF VM exit is pending on the instruction boundary after the delivery of the AEX. Consistent with the IA32 and Intel 64 architectures, if another VM exit happens before reaching that instruction boundary, the MTF VM exit is lost.

The value of the RFLAGS.TF bit at the end of AEX is same as the value of RFLAGS.TF at the time of the enclave entry. Also, if the enclave is executing inside a VMX guest, then the value of the monitor trap flag after AEX is the same as the value of that control at the time of the enclave entry.

7.2.5 RFLAGS.TF Treatment on AEX

When an opt-in enclave takes an AEX, RFLAGS.TF passes unmodified into synthetic state, and is saved as RFLAGS.TF=0 in the GPR portion of the SSA. For opt-out entry, the external value of TF is saved in CR_SAVE_TF, and TF is then cleared. For more detail see EENTER and ERESUME in Chapter 5.

7.2.6 Restriction on Setting of TF after an Opt-out Entry

From an opt-out EENTER or ERESUME until the next enclave exit, enclave is not allowed to set RFLAGS.TF. In such a situation, the POPF instruction forces RFLAGS.TF to 0 if the enclave was entered through TCS with DBGOPTIN=0.

7.2.7 Trampoline Code Considerations

Any AEX from the enclave which results in the RFLAGS.TF =1 on the reporting stack will result in a single-step #DB after the first instruction of the trampoline code if the trampoline is entered using the IRET instruction.

7.3 CODE AND DATA BREAKPOINTS

7.3.1 Breakpoint Suppression

On an opt-out entry into an enclave, all code and data breakpoints that overlap with the ELRANGE are suppressed. On any entry (either opt-in or opt-out) into an enclave, all code breakpoints that do not overlap with ELRANGE are also suppressed.

7.3.2 Breakpoint Match Reporting during Enclave Execution

The processor does not report any new matches on debug breakpoints that are suppressed on enclave entry. However, the processor does not clear any bits in DR6 that were already set at the time of the enclave entry.

SGX architecture specifically forbids reporting of silent matches on any debug breakpoints that overlap with ELRANGE after an opt-out entry.

7.3.3 Reporting of Code Breakpoint on Next Instruction on a Debug Trap

If execution in an enclave encounters a single-step trap or an enabled data breakpoint, the logical processor performs an AEX. Following the AEX, the logical processor checks the new instruction pointer (the AEP address) against any code breakpoints programmed in DR0-DR3. Any matches are reported to software.

If execution in an enclave encounters an enabled code breakpoint, the logical processor checks the current instruction pointer (within the enclave) against any code breakpoints programmed in DR0-DR3. This checking for code breakpoints occurs before the AEX, the SGX breakpoint-suppression architecture applies. Following this, the logical processor performs an AEX, after which any breakpoints matched earlier are reported to software.

7.3.4 RFLAGS.RF Treatment on AEX

RF is always set to 0 in synthetic state. This is because ERESUME after AEX is a new execution attempt.

RF value saved on SSA is the same as what would have been saved on stack in the non-SGX case. AEXs due to interrupts, traps, and code breakpoints save RF unmodified into SSA, while AEXs due to other faults save RF as 1 in the SSA.

7.3.5 Breakpoint Matching in SGX Instruction Flows

None of the implicit accesses made by SGX instructions to EPC regions generate data breakpoints. Explicit accesses made by ENCLS[ECREATE], ENCLS[EADD], ENCLS[EEXTEND], ENCLS[EINIT], ENCLS[EREMOVE], ENCLS[ETRACK], ENCLS[EBLOCK], ENCLS[EPA], ENCLS[EWB], ENCLS[ELD], ENCLS[EDBGRD], ENCLS[EDBGWR], ENCLU[EENTER], and ENCLU[ERESUME] to the EPC parameters do not fire any data breakpoints.

Explicit accesses made by the remaining SGX instructions (ENCLU[EGETKEY] and ENCLU[EREPORT]), trigger precise data breakpoints for their EPC operands. It should also be noted that all SGX instructions trigger precise data breakpoints for their non-EPC operands.

After an opt-out entry, ENCLU[EGETKEY] and ENCLU[EREPORT] do not fire any of the data breakpoints that were suppressed as a part of the enclave entry.

7.4 INT3 CONSIDERATION

7.4.1 Behavior of INT3 inside an Enclave

Inside an enclave, INT3 delivers a fault-class exception. However, the vector delivered as a result of executing the instruction depends on the manner in which the enclave was entered. Following opt-out entry, the instruction delivers #UD. Following opt-in entry, INT3 delivers #BP.

Since the event is a fault-class exception, the delivery flow of the exception does not check CPL against the DPL in the IDT gate. (Normally, delivery of INT3 generates a #GP if CPL is greater than the DPL field in IDT gate 3.) Additionally, the RIP saved in the SSA is always that of the INT3 instruction. The RIP saved on the stack/VMCS is that of the trampoline code as specified by the AEX architecture.

If execution of INT3 in an enclave causes a VM exit, the event type in the VM-exit interruption information field indicates a hardware exception (type 3; not a software exception with type 6) and the VM-exit instruction length field is saved as zero.

7.4.2 Debugger Considerations

The INT3 is always fault-like inside an enclave. Consequently, the debugger must not decrement SSA.RIP for #BP coming from an enclave. INT3 will result in #UD, if the debugger is not attached to the enclave.

7.4.3 VMM Considerations

As described above, INT3 executed by enclave delivers #BP with “interruption type” of 3. This behavior will not cause any problems for VMMs that obtain VM-entry interruption information from appropriate VMCS field (as recommended in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*), and those VMMs will continue to work seamlessly.

VMMs that fabricate the VM-entry interruption information based on the interruption vector need additional enabling. Specifically, such VMMs should be modified to use injection type of 3 (instead of 6) when they see interruption vector 3 along with the VMCS “Enclave Interruption” bit set.

7.5 BRANCH TRACING

7.5.1 BTF Treatment

Any single-step traps pending after EENTER trigger BTF exception, as EENTER is considered a branch instruction. Additionally, any single-step traps pending after EEXIT trigger BTF exception, as EEXIT is also considered a branch instruction. ERESUME does not trigger BTF traps. An AEX does not trigger BTF or TF traps.

7.5.2 LBR Treatment

7.5.2.1 LBR Stack on Opt-in Entry

An opt-in enclave entry does not change the behavior of IA32_DEBUGCTL.LBR bit. Both enclave entry and enclave exit push a record on LBR stack. EENTER/ERESUME with TCS.FLAGS.DBGOPTIN=1, inserts a new LBR record on the LBR stack. The MSR_LASTBRANCH_n_FROM_IP of this record holds linear address of the EENTER/ERESUME instruction, while MSR_LASTBRANCH_n_TO_IP of this record holds linear address of EENTER/ERESUME destination.

On EEXIT a new LBR record is pushed on the LBR stack. The MSR_LASTBRANCH_n_FROM_IP of this record holds linear address of the EEXIT instruction, while MSR_LASTBRANCH_n_TO_IP of this record holds the linear address of EEXIT destination.

On AEX a new LBR record is pushed on the LBR stack. The MSR_LASTBRANCH_n_FROM_IP of this record holds RIP saved in the SSA, while MSR_LASTBRANCH_n_TO_IP of this record holds RIP of the linear address of the AEP.

Additionally, for every branch inside the enclave, one record each is pushed on LBR stack.

Figure 7-3 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.

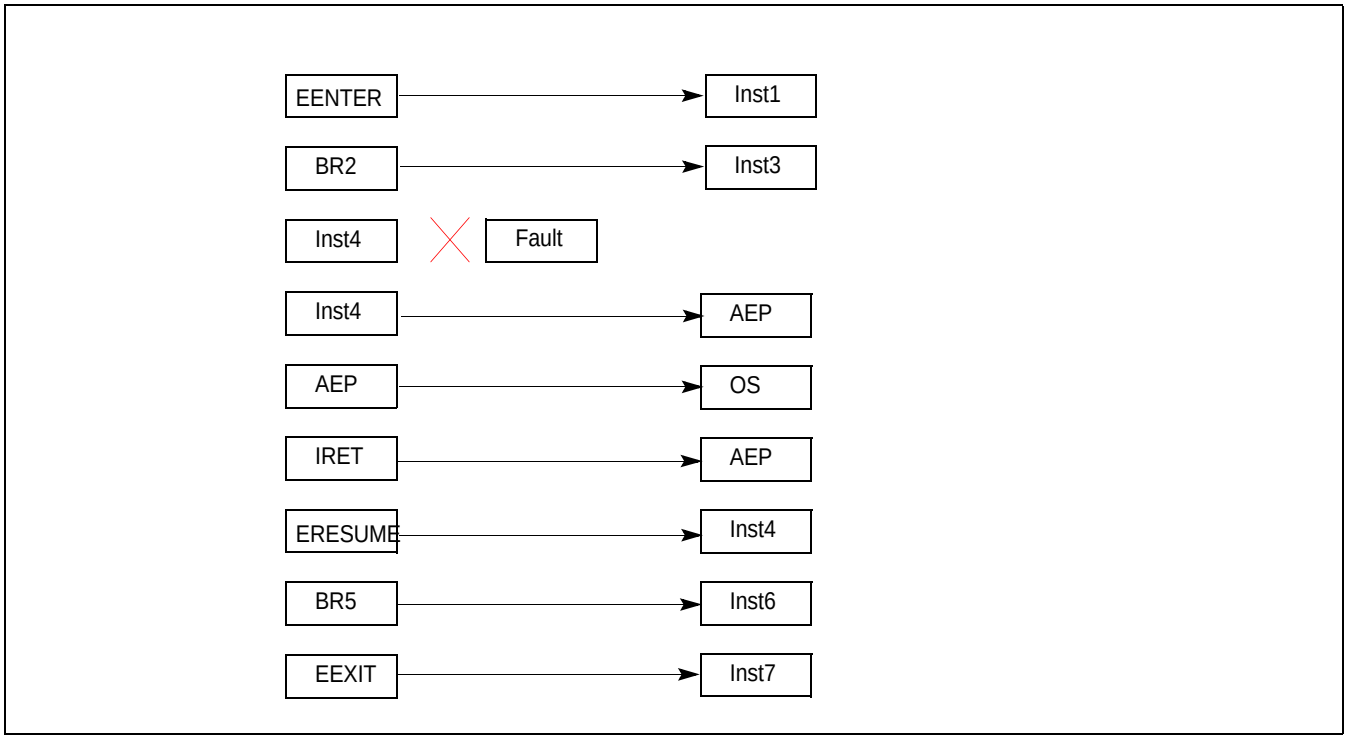

Figure 7-3. LBR Stack Interaction with Opt-in Entry

7.5.2.2 LBR Stack on Opt-out Entry

An opt-out entry into an enclave suppresses IA32_DEBUGCTL.LBR bit, and enclave exit after an opt-out entry un-suppresses the IA32_DEBUGCTL.LBR bit.

Opt-out entry into an enclave does not push any record on LBR stack.

If IA32_DEBUGCTL.LBR is set at the time of enclave entry, then EEXIT following such an enclave entry pushes one record on LBR stack. The MSR_LASTBRANCH_n_FROM_IP of such record holds the linear address of the instruction that took the logical processor into the enclave, while the MSR_LASTBRANCH_n_TO_IP of such record holds linear address of the destination of EEXIT. Additionally, if IA32_DEBUGCTL.LBR is set at the time of enclave entry, then an AEX after such an entry pushes one record on LBR stack, before pushing record for the event causing the AEX. The MSR_LASTBRANCH_n_FROM_IP of the new record holds linear address of the instruction that took the LP into the enclave, while MSR_LASTBRANCH_n_TO_IP of the new record holds linear address of the AEP. If the event causing AEX pushes a record on LBR stack, then the MSR_LASTBRANCH_n_FROM_IP for that record holds linear address of the AEP.

Figure 7-4 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.

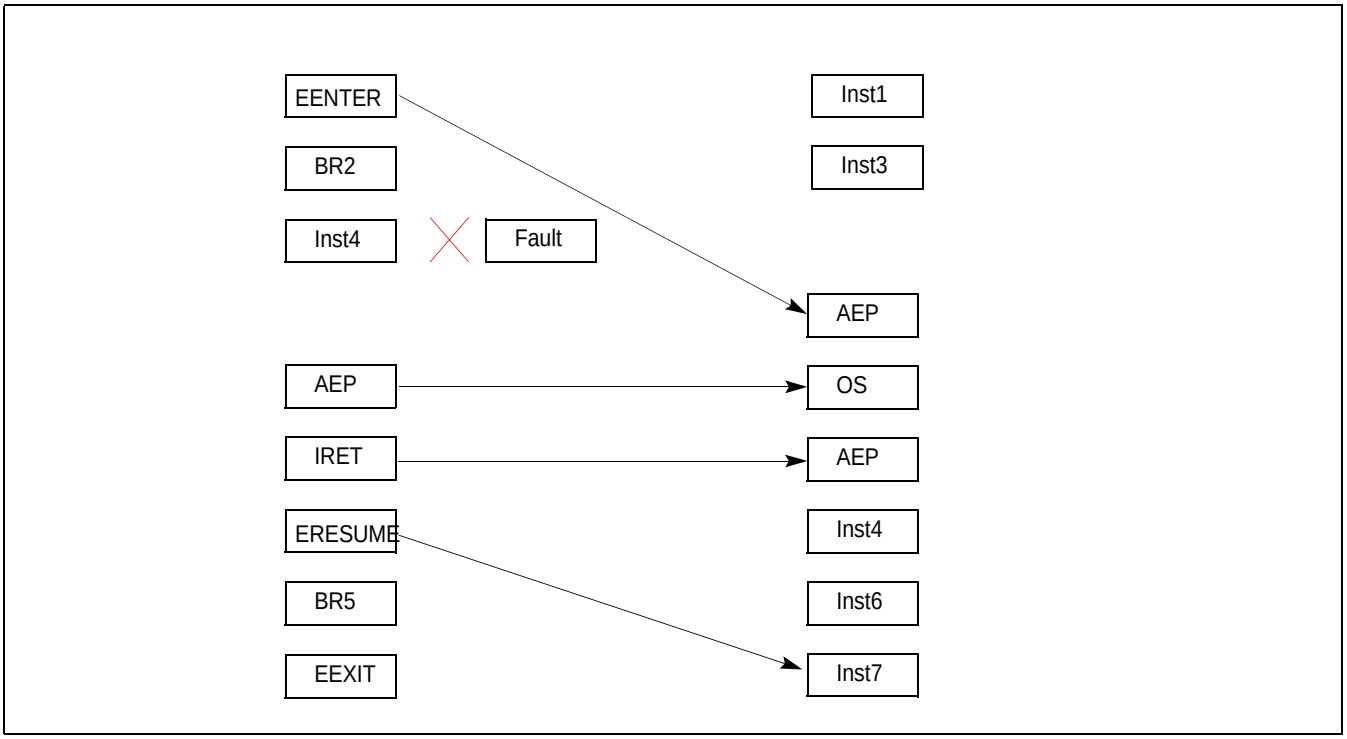

Figure 7-4. LBR Stack Interaction with Opt-out Entry

7.5.2.3 Mispredict Bit, Record Type, and Filtering

All branch records resulting from 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 SGX operations are always non-HLE/non-RTM records.

For LBR filtering, EENTER, ERESUME, EEXIT, and AEX are considered to be far branches. Consequently, bit 8 in MSR_LBR_SELECT controls filtering of the new records introduced by SGX.

7.6 INTERACTION WITH PERFORMANCE MONITORING

7.6.1 IA32_PERF_GLOBAL_STATUS Enhancement

On processors supporting SGX, the IA32_PERF_GLOBAL_STATUS MSR provides a bit indicator, known as “Anti Side-channel Interference” (ASCI) at bit position 60. If this bit is 0, the performance monitoring data in various performance monitoring counters are accumulated normally as defined by relevant architectural/microarchitectural conditions associated with the eventing logic. If the ASCI bit is set, the contents in various performance monitoring counters can be affected by the direct or indirect consequence of SGX protection of enclave code executing in the processor.

7.6.2 Performance Monitoring with Opt-in Entry

An opt-in enclave entry allow performance monitoring eventing logic to observe the contribution of enclave code executing in the processor. Thus the contents of performance monitoring counters does not distinguish between contribution originating from enclave code or otherwise. All counters, events, precise events, etc. continue to work as defined in the IA32/Intel 64 Software Developer Manual. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is always cleared.

7.6.3 Performance Monitoring with Opt-out Entry

In general, performance monitoring activities are suppressed when entering an opt-out enclave. This applies to all thread-specific, configured performance monitoring, except for the cycle-counting fixed counter, IA32_FIXED_CTR1 and IA32_FIXED_CTR2. Upon entering an opt-out enclave, IA32_FIXED_CTR0, IA32_PMCx will stop accumulating counts. Additionally, if PEBS is configured to capture PEBS record for this thread, PEBS record generation will also be suppressed.

Performance monitoring on the sibling thread may also be affected. Any one of IA32_FIXED_CTRx or IA32_PMCx on the sibling thread configured to monitor thread-specific eventing logic with AnyThread =1 is demoted to count only MyThread while an opt-out enclave is executing on the other thread.

7.6.4 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.

7.6.5 PEBS Record Generation on SGX Instructions

All leaf functions of the ENCLS instruction report “Eventing RIP” of the ENCLS instruction if a PEBS record is generated at the end of the instruction execution. Additionally, the EGETKEY and EREPORT leaf functions of the ENCLU instruction report “Eventing RIP” of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution.

The behavior of EENTER and ERESUME leaf functions of the ENCLU instruction depends on whether these leaf functions are performing an opt-in entry or an opt-out entry. If these leaf functions are performing an opt-in entry report “Eventing RIP” of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution. On the other hand, if these leaf functions are performing an opt-out entry, then these leaf functions result in PEBS being suppressed, and no PEBS record is generated at the end of these instructions.

The behavior of the EEXIT leaf function is as follows. A PEBS record is generated if there is a PEBS event pending at the end of EEXIT (due to a counter overflowing during enclave execution or during EEXIT execution). This PEBS record contains the architectural state of the logical processor at the end of EEXIT. If the enclave was entered via an opt-in entry, then this record reports the “Eventing RIP” as the linear address of the ENCLU[EEXIT] instruction (which is inside ELRANGE of the enclave just exited). If the enclave was entered via an opt-out entry, then the record reports the “Eventing RIP” as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A PEBS record is generated immediately after the AEX if there is a PEBS event pending at the end of AEX (due to a counter overflowing during enclave execution or during AEX execution). This PEBS record contains the synthetic state of the logical processor that is established at the end of AEX. For opt-in entry, this record has the EVENTING_RIP set to the eventing LIP in the enclave. For opt-out entry, the record has the EVENTING_RIP set to EENTER/ERESUME LIP.

If the enclave was entered via an opt-in entry, then this record reports the “Eventing RIP” as the linear address in the SSA of the enclave (a.k.a., the “Eventing LIP” inside the enclave). If the enclave was entered via an opt-out entry, then the record reports the “Eventing RIP” as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

It should be noted that a second PEBS event may be pended during the Enclave Exiting Event (EEE). If the PEBS event is taken at the end of the EEE then the “Eventing RIP” in this second PEBS record is the linear address of the AEP.

7.6.6 Exception-Handling on PEBS/BTS Loads/Stores after AEX

The OS/VMM is expected to pin the DS area in virtual memory. If the OS does not pin this area in memory, loads/stores to the PEBS or BTS buffer may incur faults (or other events such as APIC-access VM exit). Usually, such events are reported to the OS/VMM immediately, and generation of the PEBS/BTS record is skipped.

However, any events that are detected during PEBS/BTS record generation cannot be reported immediately to the OS/VMM, as an event window is not open at the end of AEX. Consequently, fault-like events such as page faults, EPT faults, EPT mis-configuration, and accesses to APIC-access page detected on stores to the PEBS/BTS buffer are not reported, and generation of the PEBS and/or BTS record is aborted (this may leave the buffers in a state where they have partial PEBS or BTS records), while trap-like events (such as debug traps) are pended until the next instruction boundary, where they are handled according to the architecturally defined priority. The processor continues the handling of the Enclave Exiting Event (SMI, NMI, interrupt, exception delivery, VM exit, etc.) after aborting the PEBS/BTS record generation.

7.6.6.1 Other Interactions with Performance Monitoring

For opt-in entry, EENTER, ERESUME, EEXIT, and AEX are all treated as predicted branches, and any counters that are counting such branches are incremented by 1 as a part of execution of these instructions. All of these flows are also counted as instructions, and any counters configured appropriately are incremented by 1.

For opt-out entry, execution inside an enclave is treated as a single predicted branch, and all branch-counting performance monitoring counters are incremented accordingly. Additionally, such execution is also counted as a single instruction, and all performance monitoring counters counting instructions are incremented accordingly.

Enclave entry does not affect any performance monitoring counters shared between cores.

EENTER, ERESUME, EEXIT and AEX are classified as far branches.

This page was intentionally left blank.