37.1 OVERVIEW

Intel® Software Guard Extensions (Intel® SGX) is a set of instructions and mechanisms for memory accesses added to Intel® Architecture processors. Intel SGX can encompass two collections of instruction extensions, referred to as SGX1 and SGX2, see Table 37-4. The SGX1 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 37-1), 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. The SGX2 extensions allow additional flexibility in runtime management of enclave resources and thread execution within an enclave.

Chapter 38 covers main concepts, objects and data structure formats that interact within the Intel SGX architecture. Chapter 39 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 40 describes the behavior of Asynchronous Enclave Exit (AEX) caused by events while executing enclave code. Chapter 41 covers the syntax and operational details of the instruction and associated leaf functions available in Intel SGX. Chapter 42 describes interaction of various aspects of IA32 and Intel® 64 architectures with Intel SGX. Chapter 43 covers Intel SGX support for application debug, profiling and performance monitoring.

37.2 ENCLAVE INTERACTION AND PROTECTION

Intel SGX allows the protected portion of an application to be distributed in the clear. Before the enclave is built, the enclave code and data are free for inspection and analysis. The protected portion is loaded into an enclave where its code and data is measured. Once the application's protected portion of the code and data are loaded into an enclave, 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

---

1. For additional information, see white papers on Intel SGX at http://software.intel.com/en-us/intel-isa-extensions.
Intel SGX introduces two significant capabilities to the Intel Architecture. First is the change in enclave memory access semantics. The second is protection of the address mappings of the application.

### 37.3 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 to adjust dynamically without overusing memory resources when enclave needs are low. Commitment adds physical pages to the enclave. An operating system may support separate allocate and commit operations.

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

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

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

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

An active enclave consumes resource from the Enclave Page Cache (EPC, see Section 37.5). Intel SGX provides the EREMOVE instruction that an EPC manager can use to reclaim EPC pages committed to an enclave. The EPC manager uses EREMOVE on every enclave page when the enclave is torn down. After successful execution of EREMOVE the EPC page is available for allocation to another enclave.

### 37.4 DATA STRUCTURES AND ENCLAVE OPERATION

There are 2 main data structures associated with operating an enclave, the SGX Enclave Control Structure (SECS, see Section 38.7) and the Thread Control Structure (TCS, see Section 38.8).

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

Every enclave contains one or more TCS structures. The TCS contains meta-data used by the hardware to save and restore thread specific information when entering/exiting the enclave. There is one field, FLAGS, that may be accessed by software. This field can only be accessed by debug enclaves. The flag bit, DBGOPTIN, allows to single step into the thread associated with the TCS. (see Section 38.8.1)

The SECS is created when ECREATE (see Table 37-1) is executed. The TCS can be created using the EADD instruction or the SGX2 instructions (see Table 37-2).

### 37.5 ENCLAVE PAGE CACHE

The Enclave Page Cache (EPC) is the secure storage used to store enclave pages when they are a part of an executing enclave. For an EPC page, hardware performs additional access control checks to restrict access to the page. After the current page access checks and translations are performed, the hardware checks that the EPC page
is accessible to the program currently executing. Generally an EPC page is only accessed by the owner of the executing enclave or an instruction which is setting up an EPC page.

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

Pages in the EPC can either be valid or invalid. Every valid page in the EPC belongs to one enclave instance. Each enclave instance has an EPC page that holds its SECS. The security metadata for each EPC page is held in an internal micro-architectural structure called Enclave Page Cache Map (EPCM, see Section 37.5.1).

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

### 37.5.1 Enclave Page Cache Map (EPCM)

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

- The status of EPC page with respect to validity and accessibility.
- An SECS identifier (see Section 38.19) 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 managed by the processor as part of various instruction flows.

### 37.6 ENCLAVE INSTRUCTIONS AND INTEL® SGX

The enclave instructions available with Intel 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 the respective leaf function index.

Each leaf function index is also associated with a unique, leaf-specific mnemonic. A long-form expression of Intel SGX instruction takes the form of ENCLx[LEAF_MNEMONIC], where `x` is either 'S' 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 used (omitting the ENCLx for convenience) throughout in this document.

Details of Individual SGX leaf functions are described in Chapter 41. Table 37-1 provides a summary of the instruction leaves that are available in the initial implementation of Intel SGX, which is introduced in the 6th generation Intel Core processors. Table 37-1 summarizes enhancement of Intel SGX for future Intel processors.

<table>
<thead>
<tr>
<th>Supervisor Instruction</th>
<th>Description</th>
<th>User Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCLS[ECREATE]</td>
<td>Create an enclave.</td>
<td>ENCLU[EGETKEY]</td>
<td>Create a cryptographic key.</td>
</tr>
<tr>
<td>ENCLS[EDBGRD]</td>
<td>Read data from a debug enclave by debugger.</td>
<td>ENCLU[EREPORT]</td>
<td>Create a cryptographic report.</td>
</tr>
<tr>
<td>ENCLS[EDBGWR]</td>
<td>Write data into a debug enclave by debugger.</td>
<td>ENCLU[ERESUME]</td>
<td>Re-enter an enclave.</td>
</tr>
</tbody>
</table>
## 37.7 DISCOVERING SUPPORT FOR INTEL® SGX AND ENABLING ENCLAVE INSTRUCTIONS

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

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

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

- The available and configured Intel SGX resources enumerated by the sub-leaves of CPUID.(EAX=12H) depend on the state of BIOS configuration.

### 37.7.1 Intel® SGX Opt-In Configuration

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

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

### Table 37-3. Intel® SGX Opt-in and Enabling Behavior

<table>
<thead>
<tr>
<th>CPUID.(07H,0H):EBX.SGX</th>
<th>CPUID.(12H)</th>
<th>FEATURE_CONTROL.LOCK</th>
<th>FEATURE_CONTROL.SGX_ENABLE</th>
<th>Enclave Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalid</td>
<td>X</td>
<td>X</td>
<td>#UD</td>
</tr>
<tr>
<td>1</td>
<td>Valid*</td>
<td>X</td>
<td>X</td>
<td>#UD**</td>
</tr>
</tbody>
</table>
INTRODUCTION TO INTEL® SOFTWARE GUARD EXTENSIONS

37.7.2 Intel® SGX Resource Enumeration Leaves

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

A properly configured processor exposes Intel SGX functionality with CPUID.EAX=12H reporting valid information (non-zero content) in three or more sub-leaves, see Table 37-4.

- CPUID.(EAX=12H, ECX=0H) enumerates Intel SGX capability, including enclave instruction opcode support.
- CPUID.(EAX=12H, ECX=1H) enumerates Intel SGX capability of processor state configuration and enclave configuration in the SECS structure (see Table 38-3).
- CPUID.(EAX=12H, ECX >1) enumerates available EPC resources.

<table>
<thead>
<tr>
<th>Table 37-3. Intel® SGX Opt-in and Enabling Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUID.(07H,0H):EBX.SGX</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

* Leaf 12H enumeration results are dependent on enablement.

** See list of conditions in the #UD section of the reference pages of ENCLS and ENCLU

Table 37-4. CPUID Leaf 12H, Sub-Leaf 0 Enumeration of Intel® SGX Capabilities

<table>
<thead>
<tr>
<th>CPUID.(EAX=12H,ECX=0)</th>
<th>Description Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Bits</td>
</tr>
<tr>
<td>EAX</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>SGX2: If 1, indicates leaf functions of SGX2 instruction listed in Table 37-2 are supported.</td>
</tr>
<tr>
<td>31:2</td>
<td>Reserved (0)</td>
</tr>
<tr>
<td>EBX</td>
<td>31:0</td>
</tr>
<tr>
<td>ECX</td>
<td>31:0</td>
</tr>
<tr>
<td>EDX</td>
<td>7:0</td>
</tr>
<tr>
<td></td>
<td>15:8</td>
</tr>
<tr>
<td></td>
<td>31:16</td>
</tr>
</tbody>
</table>

Table 37-5. CPUID Leaf 12H, Sub-Leaf 1 Enumeration of Intel® SGX Capabilities

<table>
<thead>
<tr>
<th>CPUID.(EAX=12H,ECX=1)</th>
<th>Description Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Bits</td>
</tr>
<tr>
<td>EAX</td>
<td>31:0</td>
</tr>
<tr>
<td>EBX</td>
<td>31:0</td>
</tr>
</tbody>
</table>
On processors that support Intel SGX1 and SGX2, CPUID leaf 12H sub-leaf 2 report physical memory resources available for use with Intel SGX. These physical memory sections are typically allocated by BIOS as Processor Reserved Memory, and available to the OS to manage as EPC.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Bits</th>
<th>Description Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECX</td>
<td>3:0</td>
<td>Report the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE. SECS.ATTRIBUTES[n+64] can be set to 1 using ECREATE only if ECX[n] is 1, where n &lt; 32.</td>
</tr>
<tr>
<td>EDX</td>
<td>3:0</td>
<td>Report the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE. SECS.ATTRIBUTES[n+96] can be set to 1 using ECREATE only if EDX[n] is 1, where n &lt; 32.</td>
</tr>
</tbody>
</table>

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

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

Table 37-5. CPUID Leaf 12H, Sub-Leaf 1 Enumeration of Intel® SGX Capabilities

<table>
<thead>
<tr>
<th>CPUID.(EAX=12H,ECX=1)</th>
<th>Description Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Bits</td>
</tr>
<tr>
<td>ECX</td>
<td>31:0</td>
</tr>
<tr>
<td>EDX</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Table 37-6. CPUID Leaf 12H, Sub-Leaf Index 2 or Higher Enumeration of Intel® SGX Resources

<table>
<thead>
<tr>
<th>CPUID.(EAX=12H,ECX &gt; 1)</th>
<th>Description Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Bits</td>
</tr>
<tr>
<td>EAX</td>
<td>3:0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>EBX</td>
<td>19:0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ECX</td>
<td>3:0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td>19:0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
38.1 OVERVIEW OF ENCLAVE EXECUTION ENVIRONMENT

When an enclave is created, it has a range of linear addresses that the processor applies enhanced access control. This ranged is called the ELRANGE (see Section 37.3). When an enclave generates a memory access, the existing IA32 segmentation and paging architecture are applied. Additionally, linear addresses inside the ELRANGE must map to an EPC page otherwise when an enclave attempts to access that linear address a fault is generated.

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

Enclave entry must happen through specific enclave instructions:

- ENCLU[EENTER], ENCLU[ERESUME].

Enclave exit must happen through specific enclave instructions or events:

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

Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations. As an example a read of an enclave page may result in the return of all one’s or return of cyphertext of the cache line. Writing to an enclave page may result in a dropped write or a machine check at a later time. The processor will provide the protections as described in Section 38.4 and Section 38.5 on such accesses.

38.2 TERMINOLOGY

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

Memory accesses initiated by certain Intel® SGX instruction leaf functions such as ECREATE, EADD, EDBGRD, EDBGWR, ELDU/ELDB, EWB, EREMOVE, EENTER, and ERESUME to EPC pages are called Indirect Enclave Accesses (Indirect EA). Table 38-1 lists additional details of the indirect EA of SGX1 and SGX2 extensions.

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

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

38.3 ACCESS-CONTROL REQUIREMENTS

Enclave accesses have the following access-control attributes:

- All memory accesses must conform to segmentation and paging protection mechanisms.
- Code fetches from inside an enclave to a linear address outside that enclave result in a #GP(0) exception.
- Non-enclave accesses to EPC memory result in undefined behavior. EPC memory is protected as described in Section 38.4 and Section 38.5 on such accesses.
- EPC pages of page types PT_REG, PT_TCS and PT_TRIM must be mapped to ELRANGE at the linear address specified when the EPC page was allocated to the enclave using ENCLS[EADD] or ENCLS[EAUG] leaf functions. Enclave accesses through other linear address result in a #PF with the PFEC.SGX bit set.
- Direct EAs to any EPC pages must conform to the currently defined security attributes for that EPC page in the EPCM. These attributes may be defined at enclave creation time (EADD) or when the enclave sets them using SGX2 instructions. The failure of these checks results in a #PF with the PFEC.SGX bit set.
ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

- Target page must belong to the currently executing enclave.
- Data may be written to an EPC page if the EPCM allow write access.
- Data may be read from an EPC page if the EPCM allow read access.
- Instruction fetches from an EPC page are allowed if the EPCM allows execute access.
- Target page must not have a restricted page type (PT_SECS, PT_TCS, PT_VA, or PT_TRIM).
- The EPC page must not be BLOCKED.
- The EPC page must not be PENDING.
- The EPC page must not be MODIFIED.

38.4 SEGMENT-BASED ACCESS CONTROL

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

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

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

38.5 PAGE-BASED ACCESS CONTROL

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

Intel SGX builds on the processor’s paging mechanism to provide page-granular access-control for enclave pages. Enclave pages are only accessible from inside the currently executing enclave if they belong to that enclave. In addition, enclave accesses must conform to the access control requirements described in Section 38.3. or through certain Intel SGX instructions. Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations.

38.5.2 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.

38.5.3 Implicit vs. Explicit Accesses

Memory accesses originating from Intel SGX instruction leaf functions are categorized as either explicit accesses or implicit accesses. Table 38-1 lists the implicit and explicit memory accesses made by Intel SGX leaf functions.

---

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

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

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

The interaction of explicit memory accesses with data breakpoints is leaf-function-specific, and is documented in
Section 43.3.4.

38.5.3.2  Implicit Accesses

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

These accesses do not trigger any access-control faults/exits or data breakpoints. Table 38-1 lists memory objects
that Intel SGX instruction leaf functions access either by explicit access or implicit access. The addresses of explicit
access objects are passed via register operands with the second through fourth column of Table 38-1 matching
implicitly encoded registers RBX, RCX, RDX.

Physical addresses used in different implicit accesses are cached via different instructions and for different dura-
tions. The physical address of SECS associated with each EPC page is cached at the time the page is added to the
enclave via ENCLS[EADD] or ENCLS[EAUG], or when the page is loaded to EPC via ENCLS[ELDB] or ENCLS[ELDU].
This binding is severed when the corresponding page is removed from the EPC via ENCLS[EREMOVE] or
ENCLS[EWB]. Physical addresses of TCS and SSA pages are cached at the time of most-recent enclave entry. Exit
from an enclave (ENCLU[EEXIT] or AEX) flushes this caching. Details of Asynchronous Enclave Exit is described in
Chapter 40.

The physical addresses that are cached for use by implicit accesses are derived from logical (or linear) addresses
after checks such as segmentation, paging, EPT, and APIC virtualization checks. These checks may trigger excep-
tions 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>
<thead>
<tr>
<th>Instr. Leaf</th>
<th>Enum.</th>
<th>Explicit 1</th>
<th>Explicit 2</th>
<th>Explicit 3</th>
<th>Implicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCEPT</td>
<td>SGX2</td>
<td>SECINFO</td>
<td>EPCPAGE</td>
<td></td>
<td>SECS</td>
</tr>
<tr>
<td>EACCEPTCOPY</td>
<td>SGX2</td>
<td>SECINFO</td>
<td>EPCPAGE (Src)</td>
<td>EPCPAGE (Dst)</td>
<td></td>
</tr>
<tr>
<td>EADD</td>
<td>SGX1</td>
<td>PAGEINFO and linked structures</td>
<td>EPCPAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EAUG</td>
<td>SGX2</td>
<td>PAGEINFO and linked structures</td>
<td>EPCPAGE</td>
<td></td>
<td>SECS</td>
</tr>
<tr>
<td>EBLOCK</td>
<td>SGX1</td>
<td>EPCPAGE</td>
<td></td>
<td></td>
<td>SECS</td>
</tr>
<tr>
<td>ECREATE</td>
<td>SGX1</td>
<td>PAGEINFO and linked structures</td>
<td>EPCPAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDBGDRD</td>
<td>SGX1</td>
<td>EPCADDR</td>
<td>Destination</td>
<td></td>
<td>SECS</td>
</tr>
<tr>
<td>EDBGWR</td>
<td>SGX1</td>
<td>EPCADDR</td>
<td>Source</td>
<td></td>
<td>SECS</td>
</tr>
<tr>
<td>EEENTER</td>
<td>SGX1</td>
<td>TCS and linked SSA</td>
<td></td>
<td></td>
<td>SECS, TCS</td>
</tr>
<tr>
<td>EEEXIT</td>
<td>SGX1</td>
<td>SECS</td>
<td>EPCPAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEEXTEND</td>
<td>SGX1</td>
<td>SECS</td>
<td>EPCPAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EGETKEY</td>
<td>SGX1</td>
<td>KEYREQUEST</td>
<td>KEY</td>
<td></td>
<td>SECS</td>
</tr>
<tr>
<td>EINIT</td>
<td>SGX1</td>
<td>SIGSTRUCT</td>
<td>SECS</td>
<td>EINITTOKEN</td>
<td></td>
</tr>
<tr>
<td>ELDB/ELDU</td>
<td>SGX1</td>
<td>PAGEINFO and linked structures, PMCID</td>
<td>EPCPAGE</td>
<td>VAPAGE</td>
<td></td>
</tr>
<tr>
<td>EMODPE</td>
<td>SGX2</td>
<td>SECINFO</td>
<td>EPCPAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMODPR</td>
<td>SGX2</td>
<td>SECINFO</td>
<td>EPCPAGE</td>
<td></td>
<td>SECS</td>
</tr>
<tr>
<td>EMODT</td>
<td>SGX2</td>
<td>SECINFO</td>
<td>EPCPAGE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
38.6 INTEL® SGX DATA STRUCTURES OVERVIEW

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

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

Details of the top-level data structures and associated sub-structures are listed in Section 38.7 through Section 38.19.

38.7 SGX ENCLAVE CONTROL STRUCTURE (SECS)

The SECS data structure requires 4K-Bytes alignment.

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE</td>
<td>0</td>
<td>8</td>
<td>Size of enclave in bytes; must be power of 2.</td>
</tr>
<tr>
<td>BASEADDR</td>
<td>8</td>
<td>8</td>
<td>Enclave Base Linear Address must be naturally aligned to size.</td>
</tr>
<tr>
<td>SSAFRAMESIZE</td>
<td>16</td>
<td>4</td>
<td>Size of one SSA frame in pages, including XSAVE, pad, GPR, and MISC (if CPUID.(EAX=12H, ECX=0):EBX != 0).</td>
</tr>
<tr>
<td>MISCESELECT</td>
<td>20</td>
<td>4</td>
<td>Bit vector specifying which extended features are saved to the MISC region (see Section 38.7.2) of the SSA frame when an AEX occurs.</td>
</tr>
</tbody>
</table>

*Details of Asynchronous Enclave Exit (AEX) is described in Section 40.4
ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

38.7.1 ATTRIBUTES

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

38.7.2 SECS.MISCSELECT Field

CPUID.(EAX=12H, ECX=0):EBX[31:0] enumerates which extended information that the processor can save into the MISC region of SSA when an AEX occurs. An enclave writer can specify via SIGSTRUCT how to set the SECS.MISCSELECT field. The bit vector of MISCSELECT selects which extended information is to be saved in the MISC region of the SSA frame when an AEX is generated. The bit vector definition of extended information is listed in Table 38-4.

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

The SECS.MISCSELECT field determines the size of MISC region of the SSA frame, see Section 38.9.2.

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESERVED</td>
<td>24</td>
<td>24</td>
<td>Attributes of the Enclave, see Table 38-3.</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td>48</td>
<td>16</td>
<td>Attributes of the Enclave, see Table 38-3.</td>
</tr>
<tr>
<td>MRENCLAVE</td>
<td>64</td>
<td>32</td>
<td>Measurement Register of enclave build process. See SIGSTRUCT for format.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>96</td>
<td>32</td>
<td>Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for format.</td>
</tr>
<tr>
<td>MRSIGNER</td>
<td>128</td>
<td>32</td>
<td>Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for format.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>160</td>
<td>96</td>
<td>Security version number (SVN) of the enclave.</td>
</tr>
<tr>
<td>ISVPRODID</td>
<td>256</td>
<td>2</td>
<td>Product ID of enclave.</td>
</tr>
<tr>
<td>ISVSVN</td>
<td>258</td>
<td>2</td>
<td>Security version number (SVN) of the enclave.</td>
</tr>
</tbody>
</table>
| RESERVED      | 260           | 3836        | The RESERVED field consists of the following:
- EID: An 8 byte Enclave Identifier. Its location is implementation specific.
- PAD: A 352 bytes padding pattern from the Signature (used for key derivation strings). It's location is implementation specific.
- The remaining 3476 bytes are reserved area.
The entire 3836 byte field must be cleared prior to executing ECREATE or EREPORT. |

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>0</td>
<td></td>
<td>This bit indicates if the enclave has been initialized by EINIT. It must be cleared when loaded as part of ECREATE. For EREPORT instruction, TARGET_INFO.ATTRIBUTES[ENIT] must always be 1 to match the state after EINIT has initialized the enclave.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>1</td>
<td></td>
<td>If 1, the enclave permit debugger to read and write enclave data using EDBGWR and EDBGWR.</td>
</tr>
<tr>
<td>MODE64BIT</td>
<td>2</td>
<td></td>
<td>Enclave runs in 64-bit mode.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>3</td>
<td></td>
<td>Must be Zero.</td>
</tr>
<tr>
<td>PROVISIONKEY</td>
<td>4</td>
<td></td>
<td>Provisioning Key is available from EGETKEY.</td>
</tr>
<tr>
<td>EINITTOKENKEY</td>
<td>5</td>
<td></td>
<td>EINIT token key is available from EGETKEY.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>63:6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XFRM</td>
<td>127:64</td>
<td></td>
<td>XSAVE Feature Request Mask. See Section 42.7.</td>
</tr>
</tbody>
</table>
38.8 THREAD CONTROL STRUCTURE (TCS)

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

### Table 38-5. Layout of Thread Control Structure (TCS)

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAGE</td>
<td>0</td>
<td>8</td>
<td>Enclave execution state of the thread controlled by this TCS. A value of 0 indicates that this TCS is available for enclave entry. A value of 1 indicates that a processor is currently executing an enclave in the context of this TCS.</td>
</tr>
<tr>
<td>FLAGS</td>
<td>8</td>
<td>8</td>
<td>The thread’s execution flags (see Section 38.8.1).</td>
</tr>
<tr>
<td>OSSA</td>
<td>16</td>
<td>8</td>
<td>Offset of the base of the State Save Area stack, relative to the enclave base. Must be page aligned.</td>
</tr>
<tr>
<td>CSSA</td>
<td>24</td>
<td>4</td>
<td>Current slot index of an SSA frame, cleared by EADD and EACCEPT.</td>
</tr>
<tr>
<td>NSSA</td>
<td>28</td>
<td>4</td>
<td>Number of available slots for SSA frames.</td>
</tr>
<tr>
<td>OENTRY</td>
<td>32</td>
<td>8</td>
<td>Offset in enclave to which control is transferred on EENTER relative to the base of the enclave.</td>
</tr>
<tr>
<td>AEP</td>
<td>40</td>
<td>8</td>
<td>The value of the Asynchronous Exit Pointer that was saved at EENTER time.</td>
</tr>
<tr>
<td>OFSBSAGX</td>
<td>48</td>
<td>8</td>
<td>Offset to add to the base address of the enclave for producing the base address of FS segment inside the enclave. Must be page aligned.</td>
</tr>
<tr>
<td>OGSBSAGX</td>
<td>56</td>
<td>8</td>
<td>Offset to add to the base address of the enclave for producing the base address of GS segment inside the enclave. Must be page aligned.</td>
</tr>
<tr>
<td>FSLIMIT</td>
<td>64</td>
<td>4</td>
<td>Size to become the new FS limit in 32-bit mode.</td>
</tr>
<tr>
<td>GSLIMIT</td>
<td>68</td>
<td>4</td>
<td>Size to become the new GS limit in 32-bit mode.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>72</td>
<td>4024</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

38.8.1 TCS.FLAGS

### Table 38-6. Layout of TCS.FLAGS Field

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBGOPTIN</td>
<td>0</td>
<td>If set, allows debugging features (single-stepping, breakpoints, etc.) to be enabled and active while executing in the enclave on this TCS. Hardware clears this bit on EADD. A debugger may later modify it if the enclave’s ATTRIBUTES.DEBUG is set.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>63:1</td>
<td></td>
</tr>
</tbody>
</table>

38.8.2 State Save Area Offset (OSSA)

The OSSA points to a stack of State Save Area (SSA) frames (see Section 38.9) used to save the processor state when an interrupt or exception occurs while executing in the enclave.
38.8.3 Current State Save Area Frame (CSSA)
CSSA is the index of the current SSA frame that will be used by the processor to determine where to save the processor state on an interrupt or exception that occurs while executing in the enclave. It is an index into the array of frames addressed by OSSA. CSSA is incremented on an AEX and decremented on an ERESUME.

38.8.4 Number of State Save Area Frames (NSSA)
NSSA specifies the number of SSA frames available for this TCS. There must be at least one available SSA frame when EENTER-ing the enclave or the EENTER will fail.

38.9 STATE SAVE AREA (SSA) FRAME
When an AEX occurs while running in an enclave, the architectural state is saved in the thread’s current SSA frame, which is pointed to by TCS.CSSA. An SSA frame must be page aligned, and contains the following regions:
• The XSAVE region starts at the base of the SSA frame, this region contains extended feature register state in an XSAVE/FXSAVE-compatible non-compacted format.
• A Pad region: software may choose to maintain a pad region separating the XSAVE region and the MISC region. Software choose the size of the pad region according to the sizes of the MISC and GPRSGX regions.
• The GPRSGX region. The GPRSGX region is the last region of an SSA frame (see Table 38-7). This is used to hold the processor general purpose registers (RAX ... R15), the RIP, the outside RSP and RBP, RFLAGS and the AEX information.
• The MISC region (If CPUIDEAX=12H, ECX=0):EBX[31:0] != 0). The MISC region is adjacent to the GRPSGX region, and may contain zero or more components of extended information that would be saved when an AEX occurs. If the MISC region is absent, the region between the GPRSGX and XSAVE regions is the pad region that software can use. If the MISC region is present, the region between the MISC and XSAVE regions is the pad region that software can use. See additional details in Section 38.9.2.

<table>
<thead>
<tr>
<th>Region</th>
<th>Offset (Byte)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSAVE</td>
<td>0</td>
<td>Calculate using CPUID leaf 0DH information</td>
<td>The size of XSAVE region in SSA is derived from the enclave's support of the collection of processor extended states that would be managed by XSAVE. The enablement of those processor extended state components in conjunction with CPUID leaf 0DH information determines the XSAVE region size in SSA.</td>
</tr>
<tr>
<td>Pad</td>
<td>End of XSAVE region</td>
<td>Chosen by enclave writer</td>
<td>Ensure the end of GPRSGX region is aligned to the end of a 4KB page.</td>
</tr>
<tr>
<td>MISC</td>
<td>base of GPRSGX - sizeof(MISC)</td>
<td>Calculate from highest set bit of SECS.MISCSELECT</td>
<td>See Section 38.9.2.</td>
</tr>
<tr>
<td>GPRSGX</td>
<td>SSAFRAMESIZE - 176</td>
<td>176</td>
<td>See Table 38-8 for layout of the GPRSGX region.</td>
</tr>
</tbody>
</table>
38.9.1 GPRSGX Region

The layout of the GPRSGX region is shown in Table 38-8.

Table 38-8. Layout of GPRSGX Portion of the State Save Area

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAX</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RCX</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RDX</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RBX</td>
<td>24</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RSP</td>
<td>32</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RBP</td>
<td>40</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RSI</td>
<td>48</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RDI</td>
<td>56</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>64</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>72</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>80</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>88</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>96</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td>104</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R14</td>
<td>112</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R15</td>
<td>120</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>RFLAGS</td>
<td>128</td>
<td>8</td>
<td>Flag register.</td>
</tr>
<tr>
<td>RIP</td>
<td>136</td>
<td>8</td>
<td>Instruction pointer.</td>
</tr>
<tr>
<td>URSP</td>
<td>144</td>
<td>8</td>
<td>Non-Enclave (outside) stack pointer. Saved by EENTER, restored on AEX.</td>
</tr>
<tr>
<td>URBP</td>
<td>152</td>
<td>8</td>
<td>Non-Enclave (outside) RBP pointer. Saved by EENTER, restored on AEX.</td>
</tr>
<tr>
<td>EXITINFO</td>
<td>160</td>
<td>4</td>
<td>Contains information about exceptions that cause AEXs, which might be needed by enclave software (see Section 38.9.1.1).</td>
</tr>
<tr>
<td>RESERVED</td>
<td>164</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>FSBASE</td>
<td>168</td>
<td>8</td>
<td>FS BASE.</td>
</tr>
<tr>
<td>GSBASE</td>
<td>176</td>
<td>8</td>
<td>GS BASE.</td>
</tr>
</tbody>
</table>

38.9.1.1 EXITINFO

EXITINFO contains the information used to report exit reasons to software inside the enclave. It is a 4 byte field laid out as in Table 38-9. The VALID bit is set only for the exceptions conditions which are reported inside an enclave. See Table 38-10 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 38-9. Layout of EXITINFO Field

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

38.9.1.2 VECTOR Field Definition

Table 38-10 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 38-10. Exception Vectors

<table>
<thead>
<tr>
<th>Name</th>
<th>Vector #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#DE</td>
<td>0</td>
<td>Divider exception.</td>
</tr>
<tr>
<td>#DB</td>
<td>1</td>
<td>Debug exception.</td>
</tr>
<tr>
<td>#BP</td>
<td>3</td>
<td>Breakpoint exception.</td>
</tr>
<tr>
<td>#BR</td>
<td>5</td>
<td>Bound range exceeded exception.</td>
</tr>
<tr>
<td>#UD</td>
<td>6</td>
<td>Invalid opcode exception.</td>
</tr>
<tr>
<td>#GP</td>
<td>13</td>
<td>General protection exception. Only reported if SECS.MISCSELECT.EXINFO = 1.</td>
</tr>
<tr>
<td>#PF</td>
<td>14</td>
<td>Page fault exception. Only reported if SECS.MISCSELECT.EXINFO = 1.</td>
</tr>
<tr>
<td>#MF</td>
<td>16</td>
<td>x87 FPU floating-point error.</td>
</tr>
<tr>
<td>#AC</td>
<td>17</td>
<td>Alignment check exceptions.</td>
</tr>
<tr>
<td>#XM</td>
<td>19</td>
<td>SIMD floating-point exceptions.</td>
</tr>
</tbody>
</table>

38.9.2 MISC Region

The layout of the MISC region is shown in Table 38-11. The number of components that the processor supports in the MISC region corresponds to the set bits of CPUID.(EAX=12H, ECX=0):EBX[31:0] set to 1. Each set bit in CPUID.(EAX=12H, ECX=0):EBX[31:0] has a defined size for the corresponding component, as shown in Table 38-11. Enclave writers need to do the following:

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

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

The size of the MISC region is calculated as follows:

- If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISC region is not supported.
- If CPUID.(EAX=12H, ECX=0):EBX[31:0] ≠ 0, the size of MISC region is derived from sum of the highest bit set in SECS.MISCSELECT and the size of the MISC component corresponding to that bit. Offset and size
information of currently defined MISC components are listed in Table 38-11. For example, if the highest bit set in SECS.MISCSELECT is bit 0, the MISC region offset is OFFSET(GPRSGX)-16 and size is 16 bytes.

- The processor saves a MISC component i in the MISC region if and only if SECS.MISCSELECT[i] is 1.

### Table 38-11. Layout of MISC region of the State Save Area

<table>
<thead>
<tr>
<th>MISC Components</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXINFO</td>
<td>Offset(GPRSGX)-16</td>
<td>16</td>
<td>If CPUID.(EAX=12H, ECX=0):EBX[0] = 1, exception information on #GP or #PF that occurred inside an enclave can be written to the EXINFO structure if specified by SECS.MISCSELECT[0] = 1.</td>
</tr>
</tbody>
</table>

#### 38.9.2.1 EXINFO Structure
Table 38-12 contains the layout of the EXINFO structure that provides additional information.

### Table 38-12. Layout of EXINFO Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADDR</td>
<td>0</td>
<td>8</td>
<td>If #PF: contains the page fault linear address that caused a page fault. If #GP: the field is cleared.</td>
</tr>
<tr>
<td>ERRCD</td>
<td>8</td>
<td>4</td>
<td>Exception error code for either #GP or #PF.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>12</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

#### 38.9.2.2 Page Fault Error Codes
Table 38-13 contains page fault error code that may be reported in EXINFO.ERRCD.

### Table 38-13. Page Fault Error Codes

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

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

### 38.10 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.
38.11 SECURITY INFORMATION (SECINFO)

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

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

### Table 38-15. Layout of SECINFO Data Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLAGS</td>
<td>0</td>
<td>8</td>
<td>Flags describing the state of the enclave page.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>8</td>
<td>56</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

38.11.1 SECINFO.FLAGS

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0</td>
<td>If 1 indicates that the page can be read from inside the enclave; otherwise the page cannot be read from inside the enclave.</td>
</tr>
<tr>
<td>W</td>
<td>1</td>
<td>If 1 indicates that the page can be written from inside the enclave; otherwise the page cannot be written from inside the enclave.</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
<td>If 1 indicates that the page can be executed from inside the enclave; otherwise the page cannot be executed from inside the enclave.</td>
</tr>
<tr>
<td>PENDING</td>
<td>3</td>
<td>If 1 indicates that the page is in the PENDING state; otherwise the page is not in the PENDING state.</td>
</tr>
<tr>
<td>MODIFIED</td>
<td>4</td>
<td>If 1 indicates that the page is in the MODIFIED state; otherwise the page is not in the MODIFIED state.</td>
</tr>
<tr>
<td>PR</td>
<td>5</td>
<td>If 1 indicates that a permission restriction operation on the page is in progress, otherwise a permission restriction operation is not in progress.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>7:6</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>PAGE_TYPE</td>
<td>15:8</td>
<td>The type of page that the SECINFO is associated with.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>63:16</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>
38.11.2 PAGE_TYPE Field Definition
The SECINFO flags and EPC flags contain bits indicating the type of page.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT_SECS</td>
<td>0</td>
<td>Page is an SECS.</td>
</tr>
<tr>
<td>PT_TCS</td>
<td>1</td>
<td>Page is a TCS.</td>
</tr>
<tr>
<td>PT_REG</td>
<td>2</td>
<td>Page is a regular page.</td>
</tr>
<tr>
<td>PT_VA</td>
<td>3</td>
<td>Page is a Version Array.</td>
</tr>
<tr>
<td>PT_TRIM</td>
<td>4</td>
<td>Page is in trimmed state.</td>
</tr>
<tr>
<td>All other</td>
<td></td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

38.12 PAGING CRYPTO METADATA (PCMD)
The PCMD structure is used to keep track of crypto meta-data associated with a paged-out page. Combined with PAGEINFO, it provides enough information for the processor to verify, decrypt, and reload a paged-out EPC page. The size of the PCMD structure (128 bytes) is architectural.
EWB calculates the Message Authentication Code (MAC) value and writes out the PCMD. ELDB/U reads the fields and checks the MAC.
The format of PCMD is as follows:

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

38.13 ENCLAVE SIGNATURE STRUCTURE (SIGSTRUCT)
SIGSTRUCT is a structure created and signed by the enclave developer that contains information about the enclave. SIGSTRUCT is processed by the EINIT leaf function to verify that the enclave was properly built.
SIGSTRUCT includes ENCLAVEHASH as SHA256 digest, as defined in FIPS PUB 180-4. The digests are byte strings of length 32. Each of the 8 HASH dwords is stored in little-endian order.
SIGSTRUCT includes four 3072-bit integers (MODULUS, SIGNATURE, Q1, Q2). Each such integer is represented as a byte strings of length 384, with the most significant byte at the position "offset + 383", and the least significant byte at position "offset".
The (3072-bit integer) SIGNATURE should be an RSA signature, where: a) the RSA modulus (MODULUS) is a 3072-bit integer; b) the public exponent is set to 3; c) the signing procedure uses the EMSA-PKCS1-v1.5 format with DER encoding of the "DigestInfo" value as specified in of PKCS#1 v2.1/RFC 3447.
The 3072-bit integers Q1 and Q2 are defined by:
q1 = floor(Signature^2 / Modulus);
q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);
SIGSTRUCT must be page aligned
In column 5 of Table 38-19, 'Y' indicates that this field should be included in the signature generated by the developer.

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
<th>Signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADER</td>
<td>0</td>
<td>16</td>
<td>Must be byte stream 0600000000E100000000000100000000000H</td>
<td>Y</td>
</tr>
<tr>
<td>VENDOR</td>
<td>16</td>
<td>4</td>
<td>Intel Enclave: 00008086H Non-Intel Enclave: 00000000H</td>
<td>Y</td>
</tr>
<tr>
<td>DATE</td>
<td>20</td>
<td>4</td>
<td>Build date is yyyymmdd in hex: yyyy=4 digit year, mm=1-12, dd=1-31</td>
<td>Y</td>
</tr>
<tr>
<td>HEADER2</td>
<td>24</td>
<td>16</td>
<td>Must be byte stream 0101000060000000010000000000010000000H</td>
<td>Y</td>
</tr>
<tr>
<td>SWDEFINED</td>
<td>40</td>
<td>4</td>
<td>Available for software use.</td>
<td>Y</td>
</tr>
<tr>
<td>RESERVED</td>
<td>44</td>
<td>84</td>
<td>Must be zero.</td>
<td>Y</td>
</tr>
<tr>
<td>MODULUS</td>
<td>128</td>
<td>384</td>
<td>Module Public Key (keylength=3072 bits).</td>
<td>N</td>
</tr>
<tr>
<td>EXONENT</td>
<td>512</td>
<td>4</td>
<td>RSA Exponent = 3.</td>
<td>N</td>
</tr>
<tr>
<td>SIGNATURE</td>
<td>516</td>
<td>384</td>
<td>Signature over Header and Body.</td>
<td>N</td>
</tr>
<tr>
<td>MISCESELECT*</td>
<td>900</td>
<td>4</td>
<td>Bit vector specifying Extended SSA frame feature set to be used.</td>
<td>Y</td>
</tr>
<tr>
<td>MISCMASK*</td>
<td>904</td>
<td>4</td>
<td>Bit vector mask of MISCESELECT to enforce.</td>
<td>Y</td>
</tr>
<tr>
<td>RESERVED</td>
<td>908</td>
<td>20</td>
<td>Must be zero.</td>
<td>Y</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td>928</td>
<td>16</td>
<td>Enclave Attributes that must be set.</td>
<td>Y</td>
</tr>
<tr>
<td>ATTRIBUTEMASK</td>
<td>944</td>
<td>16</td>
<td>Mask of Attributes to enforce.</td>
<td>Y</td>
</tr>
<tr>
<td>ENCLAVEHASH</td>
<td>960</td>
<td>32</td>
<td>MRENCLAVE of enclave this structure applies to.</td>
<td>Y</td>
</tr>
<tr>
<td>RESERVED</td>
<td>992</td>
<td>32</td>
<td>Must be zero.</td>
<td>Y</td>
</tr>
<tr>
<td>ISVPROID</td>
<td>1024</td>
<td>2</td>
<td>ISV assigned Product ID.</td>
<td>Y</td>
</tr>
<tr>
<td>ISVSVN</td>
<td>1026</td>
<td>2</td>
<td>ISV assigned SVN (security version number).</td>
<td>Y</td>
</tr>
<tr>
<td>RESERVED</td>
<td>1028</td>
<td>12</td>
<td>Must be zero.</td>
<td>N</td>
</tr>
<tr>
<td>Q1</td>
<td>1040</td>
<td>384</td>
<td>Q1 value for RSA Signature Verification.</td>
<td>N</td>
</tr>
<tr>
<td>Q2</td>
<td>1424</td>
<td>384</td>
<td>Q2 value for RSA Signature Verification.</td>
<td>N</td>
</tr>
</tbody>
</table>

*If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISCESELECT must be 0.
If CPUID.(EAX=12H, ECX=0):EBX[31:0] !=0, enclave writers must specify MISCESELECT such that each cleared bit in MISCMASK must also specify the corresponding bit as 0 in MISCESELECT.

### 38.14 **EINIT TOKEN STRUCTURE (EINITTOKEN)**

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

EINIT token must be 512-Byte aligned.
### Table 38-20. Layout of EINIT Token (EINITTOKEN)

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>MACed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>0</td>
<td>4</td>
<td>Y</td>
<td>Bit 0: 1: Valid; 0: Invalid. All other bits reserved.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>4</td>
<td>44</td>
<td>Y</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td>48</td>
<td>16</td>
<td>Y</td>
<td>ATTRIBUTES of the Enclave.</td>
</tr>
<tr>
<td>MRENCLAVE</td>
<td>64</td>
<td>32</td>
<td>Y</td>
<td>MRENCLAVE of the Enclave.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>96</td>
<td>32</td>
<td>Y</td>
<td>Reserved.</td>
</tr>
<tr>
<td>MRSIGNER</td>
<td>128</td>
<td>32</td>
<td>Y</td>
<td>MRSIGNER of the Enclave.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>160</td>
<td>32</td>
<td>Y</td>
<td>Reserved.</td>
</tr>
<tr>
<td>CPUSVNLE</td>
<td>192</td>
<td>16</td>
<td>N</td>
<td>Launch Enclave’s CPUSVN.</td>
</tr>
<tr>
<td>ISVPRODID</td>
<td>208</td>
<td>02</td>
<td>N</td>
<td>Launch Enclave’s ISVPRODID.</td>
</tr>
<tr>
<td>ISVSVNLE</td>
<td>210</td>
<td>02</td>
<td>N</td>
<td>Launch Enclave’s ISVSVN.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>212</td>
<td>24</td>
<td>N</td>
<td>Reserved.</td>
</tr>
<tr>
<td>MASKEDMISCSEL</td>
<td>236</td>
<td>4</td>
<td></td>
<td>Launch Enclave’s MASKEDMISCSELECT: set by the LE to the resolved MISCELECT value, used by EGETKEY (after applying KEYREQUEST’s masking).</td>
</tr>
<tr>
<td>ECTLE</td>
<td>240</td>
<td>16</td>
<td>N</td>
<td>Launch Enclave’s MASKEDATTRIBUTES: This should be set to the LE’s ATTRIBUTES masked with ATTRIBUTEMASK of the LE’s KEYREQUEST.</td>
</tr>
<tr>
<td>KEYID</td>
<td>256</td>
<td>32</td>
<td>N</td>
<td>Value for key wear-out protection.</td>
</tr>
<tr>
<td>MAC</td>
<td>288</td>
<td>16</td>
<td>N</td>
<td>Message Authentication Code on EINITTOKEN using EINITOKENKEY.</td>
</tr>
</tbody>
</table>

### 38.15 REPORT (REPORT)

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

### Table 38-21. Layout of REPORT

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUSVN</td>
<td>0</td>
<td>16</td>
<td>The security version number of the processor.</td>
</tr>
<tr>
<td>MISCELECT</td>
<td>16</td>
<td>4</td>
<td>Bit vector specifying which extended features are saved to the MISC region of the SSA frame when an AEX occurs.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>20</td>
<td>28</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td>48</td>
<td>16</td>
<td>ATTRIBUTES of the Enclave. See Section 38.7.1.</td>
</tr>
<tr>
<td>MRENCLAVE</td>
<td>64</td>
<td>32</td>
<td>The value of SECS.MRENCLAVE.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>96</td>
<td>32</td>
<td>Reserved.</td>
</tr>
<tr>
<td>MRSIGNER</td>
<td>128</td>
<td>32</td>
<td>The value of SECS.MRSIGNER.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>160</td>
<td>96</td>
<td>Zero.</td>
</tr>
<tr>
<td>ISVPRODID</td>
<td>256</td>
<td>02</td>
<td>Product ID of enclave.</td>
</tr>
<tr>
<td>ISVSVN</td>
<td>258</td>
<td>02</td>
<td>Security version number (SVN) of the enclave.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>260</td>
<td>60</td>
<td>Zero.</td>
</tr>
<tr>
<td>REPORTDATA</td>
<td>320</td>
<td>64</td>
<td>Data provided by the user and protected by the REPORT’s MAC, see Section 38.15.1.</td>
</tr>
<tr>
<td>KEYID</td>
<td>384</td>
<td>32</td>
<td>Value for key wear-out protection.</td>
</tr>
</tbody>
</table>
38.15.1 REPORTDATA

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

38.16 REPORT TARGET INFO (TARGETINFO)

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

Table 38-22. Layout of TARGETINFO Data Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEASUREMENT</td>
<td>0</td>
<td>32</td>
<td>The MRENCLAVE of the target enclave.</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td>32</td>
<td>16</td>
<td>The ATTRIBUTES field of the target enclave.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>48</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>MISCESELECT</td>
<td>52</td>
<td>4</td>
<td>The MISCESELECT of the target enclave.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>56</td>
<td>456</td>
<td></td>
</tr>
</tbody>
</table>

38.17 KEY REQUEST (KEYREQUEST)

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

Table 38-23. Layout of KEYREQUEST Data Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYNAME</td>
<td>0</td>
<td>02</td>
<td>Identifies the Key Required.</td>
</tr>
<tr>
<td>KEYPOLICY</td>
<td>02</td>
<td>02</td>
<td>Identifies which inputs are required to be used in the key derivation.</td>
</tr>
<tr>
<td>ISVSVN</td>
<td>04</td>
<td>02</td>
<td>The ISV security version number that will be used in the key derivation.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>06</td>
<td>02</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>CPUSVN</td>
<td>08</td>
<td>16</td>
<td>The security version number of the processor used in the key derivation.</td>
</tr>
<tr>
<td>ATTRIBUTEMASK</td>
<td>24</td>
<td>16</td>
<td>A mask defining which ATTRIBUTES bits will be included in key derivation.</td>
</tr>
<tr>
<td>KEYID</td>
<td>40</td>
<td>32</td>
<td>Value for key wear-out protection.</td>
</tr>
<tr>
<td>MISCMASK</td>
<td>72</td>
<td>4</td>
<td>A mask defining which MISCESELECT bits will be included in key derivation.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>76</td>
<td>436</td>
<td></td>
</tr>
</tbody>
</table>
38.17.1  KEY REQUEST KeyNames

Table 38-24.  Supported KEYName Values

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINITOKEN_KEY</td>
<td>0</td>
<td>EINIT_TOKEN key</td>
</tr>
<tr>
<td>PROVISION_KEY</td>
<td>1</td>
<td>Provisioning Key</td>
</tr>
<tr>
<td>PROVISION_SEAL_KEY</td>
<td>2</td>
<td>Provisioning Seal Key</td>
</tr>
<tr>
<td>REPORT_KEY</td>
<td>3</td>
<td>Report Key</td>
</tr>
<tr>
<td>SEAL_KEY</td>
<td>4</td>
<td>Seal Key</td>
</tr>
<tr>
<td>All other</td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>

38.17.2  Key Request Policy Structure

Table 38-25.  Layout of KEYPOLICY Field

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRENCLAVE</td>
<td>0</td>
<td>If 1, derive key using the enclave’s MRENCLAVE measurement register.</td>
</tr>
<tr>
<td>MRSIGNER</td>
<td>1</td>
<td>If 1, derive key using the enclave’s MRSIGNER measurement register.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>15:2</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

38.18  VERSION ARRAY (VA)

In order to securely store the versions of evicted EPC pages, Intel SGX defines a special EPC page type called a Version Array (VA). Each VA page contains 512 slots, each of which can contain an 8-byte version number for a page evicted from the EPC. When an EPC page is evicted, software chooses an empty slot in a VA page; this slot receives the unique version number of the page being evicted. When the EPC page is reloaded, there must be a VA slot that must hold the version of the page. If the page is successfully reloaded, the version in the VA slot is cleared. VA pages can be evicted, just like any other EPC page. When evicting a VA page, a version slot in some other VA page must be used to hold the version for the VA being evicted. A Version Array Page must be 4K-Bytes aligned.

Table 38-26.  Layout of Version Array Data Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>OFFSET (Bytes)</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot 0</td>
<td>0</td>
<td>08</td>
<td>Version Slot 0</td>
</tr>
<tr>
<td>Slot 1</td>
<td>8</td>
<td>08</td>
<td>Version Slot 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slot 511</td>
<td>4088</td>
<td>08</td>
<td>Version Slot 511</td>
</tr>
</tbody>
</table>

38.19  ENCLAVE PAGE CACHE MAP (EPCM)

EPCM is a secure structure used by the processor to track the contents of the EPC. The EPCM holds exactly one entry for each page that is currently loaded into the EPC. EPCM is not accessible by software, and the layout of EPCM fields is implementation specific.
### Table 38-27. Content of an Enclave Page Cache Map Entry

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALID</td>
<td>Indicates whether the EPCM entry is valid.</td>
</tr>
<tr>
<td>R</td>
<td>Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.</td>
</tr>
<tr>
<td>W</td>
<td>Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.</td>
</tr>
<tr>
<td>X</td>
<td>Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.</td>
</tr>
<tr>
<td>PT</td>
<td>EPCM page type (PT_SECS, PT_TCS, PT_REG, PT_VA, PT_TRIM).</td>
</tr>
<tr>
<td>ENCLAVESECS</td>
<td>SECS identifier of the enclave to which the EPC page belongs.</td>
</tr>
<tr>
<td>ENCLAVEADDRESS</td>
<td>Linear enclave address of the EPC page.</td>
</tr>
<tr>
<td>BLOCKED</td>
<td>Indicates whether the EPC page is in the blocked state.</td>
</tr>
<tr>
<td>PENDING</td>
<td>Indicates whether the EPC page is in the pending state.</td>
</tr>
<tr>
<td>MODIFIED</td>
<td>Indicates whether the EPC page is in the modified state.</td>
</tr>
<tr>
<td>PR</td>
<td>Indicates whether the EPC page is in a permission restriction state.</td>
</tr>
</tbody>
</table>
The following aspects of enclave operation are described in this chapter:

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

### 39.1 CONSTRUCTING AN ENCLAVE

Figure 39-1 illustrates a typical Enclave memory layout.

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

1. The application hands over the enclave content along with additional information required by the enclave creation API to the enclave creation service running at privilege level 0.
2. The enclave creation service running at privilege level 0 uses the ECREATE leaf function to set up the initial environment, specifying base address and size of the enclave. This address range, the ELRANGE, is part of the application’s address space. This reserves the memory range. The enclave will now reside in this address range.
region. ECREATE also allocates an Enclave Page Cache (EPC) page for the SGX Enclave Control Structure (SECS). Note that this page is not required to be a part of the enclave linear address space and is not required to be mapped into the process.

3. The enclave creation service uses the EADD leaf function to commit EPC pages to the enclave, and use EEXTEND to measure the committed memory content of the enclave. For each page to be added to the enclave:
   — Use EADD to add the new page to the enclave.
   — If the enclave developer requires measurement of the page as a proof for the content, use EEXTEND to add a measurement for 256 bytes of the page. Repeat this operation until the entire page is measured.

4. The enclave creation service uses the EINIT leaf function to complete the enclave creation process and finalize the enclave measurement to establish the enclave identity. Until an EINIT is executed, the enclave is not permitted to execute any enclave code (i.e. entering the enclave by executing EENTER would result in a fault).

### 39.1.1 ECREATE

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

### 39.1.2 EADD and EEXTEND Interaction

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

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

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

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

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

### 39.1.3 EINIT Interaction

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

A cryptographic hash of the log is stored as the **enclave identity**. Correct construction of the enclave results in the cryptographic hash matching the one built by the enclave owner and included as the ENCLAVEHASH field of SIGSTRUCT. The **enclave identity** provided by the EREPORT leaf function can be verified by a remote party.
The EINIT leaf function checks the EINIT token to validate that the enclave has been enabled on this platform. If the enclave is not correctly constructed, or the EINIT token is not valid for the platform, or SIGSTRUCT isn't properly signed, then EINIT will fail. See the EINIT leaf function for details on the error reporting.

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

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

EINIT establishes the sealing identity using the following steps:

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

### 39.1.4 Intel® SGX Launch Control Configuration

Intel® SGX Launch Control is a set of controls that govern the creation of enclaves. Before the EINIT leaf function will successfully initialize an enclave, a designated Launch Enclave must create an EINITTOKEN for that enclave.

Launch Enclaves have SECS.ATTRIBUTES.EINITTOKENKEY = 1, granting them access to the EINITTOKENKEY from the EGETKEY leaf function. EINITTOKENKEY must be used by the Launch Enclave when computing EINITTOKEN.MAC, the Message Authentication Code of the EINITTOKEN.

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

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

IA32_FEATURE_CONTROL bit 17 controls the permissions on the IA32_SGXLEPUBKEYHASH MSRs when CPUID.(EAX=12H, ECX=00H):EAX[0] = 1. If IA32_FEATURE_CONTROL is locked with bit 17 set, IA32_SGXLEPUBKEYHASH MSRs are reconfigurable (writeable). If either IA32_FEATURE_CONTROL is not locked or bit 17 is clear, the MSRs are read only. By leaving these MSRs writable, system SW or a VMM can support a plurality of Launch Enclaves for hosting multiple execution environments. See Section 42.3.2 for more details.

### 39.2 ENCLAVE ENTRY AND EXITING

#### 39.2.1 Controlled Entry and Exit

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

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

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

EENTER performs the following operations:
ENCLAVE OPERATION

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

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

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

39.2.2 Asynchronous Enclave Exit (AEX)

Asynchronous and synchronous events, such as exceptions, interrupts, traps, SMIs, and VM exits may occur while executing inside an enclave. These events are referred to as Enclave Exiting Events (EEE). Upon an EEE, the processor state is securely saved inside the enclave (in the thread’s current SSA frame) and then replaced by a synthetic state to prevent leakage of secrets. The process of securely saving state and establishing the synthetic state is called an Asynchronous Enclave Exit (AEX). Details of AEX is described in Chapter 40, “Enclave Exiting Events”.

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

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

39.2.3 Resuming Execution after AEX

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

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

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

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

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

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

39.3 CALLING ENCLAVE PROCEDURES

39.3.1 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.

39.3.2 Register Preservation

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

39.3.3 Returning to Caller

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

39.4 INTEL® SGX KEY AND ATTESTATION

39.4.1 Enclave Measurement

During the enclave build process, two “measurements” are taken of each enclave and are stored in two 256-bit Measurement Registers (MR): MRENCLAVE and MRSIGNER. MRENCLAVE represents the enclave’s contents and build process. MRSIGNER represents the entity that signed the enclave’s SIGSTRUCT.
The values of the Measurement Registers are included in attestations to identify the enclave to remote parties. The MRs are also included in most keys, binding keys to enclaves with specific MRs.

### 39.4.1.1 MRENCLAVE

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

Figure 39-2 illustrates a simplified flow of changes to the MRENCLAVE register when building an enclave:
- Enclave creation with ECREATE.
- Copying a non-enclave source page into the EPC of an un-initialized enclave with EADD.
- Updating twice of the MRENCLAVE after modifying the enclave’s page content, i.e. EEXTEND twice.
- Finalizing the enclave build with EINIT.

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

![Figure 39-2. Measurement Flow of Enclave Build Process](image)

### 39.4.1.2 MRSIGNER

Each enclave is signed using a 3072 bit RSA key. The signature is stored in the SIGSTRUCT. In the SIGSTRUCT, the enclave’s signer also assigns a product ID (ISVPROMID) and a security version (ISVSvn) to the enclave. MRSIGNER is the SHA-256 hash of the signer’s public key.

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

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

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

39.4.2.1 Enclave Security Version

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

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

39.4.2.2 Hardware Security Version

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

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

39.4.3 Keys

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

Each enclave requests keys using the EGETKEY leaf function. The key is based on enclave parameters such as measurement, the enclave signing key, security attributes of the enclave, and the Hardware Security version of the processor itself. A full list of parameter options is specified in the KEYREQUEST structure, see details in Section 38.17.

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

The EGETKEY leaf function provides several key types. Each key is specific to the processor, CPUSVN, and the enclave that executed EGETKEY. The EGETKEY instruction definition details how each of these keys is derived, see Table 41-56. Additionally,

- **SEAL Key**: The Seal key is a general purpose key for the enclave to use to protect secrets. Typical uses of the Seal key are encrypting and calculating MAC of secrets on disk. There are 2 types of Seal Key described in Section 39.4.3.1.

- **REPORT Key**: This key is used to compute the MAC on the REPORT structure. The EREPORT leaf function is used to compute this MAC, and destination enclave uses the Report key to verify the MAC. The software usage flow is detailed in Section 39.4.3.2.

- **EINITTOKENKEY**: This key is used by Launch Enclaves to compute the MAC on EINITTOKENs. These tokens are then verified in the EINIT leaf function. The key is only available to enclaves with ATTRIBUTE.EINITTOKENKEY set to 1.
• PROVISIONING Key and PROVISIONING SEAL Key: These keys are used by attestation key provisioning software to prove to remote parties that the processor is genuine and identify the currently executing TCB. These keys are only available to enclaves with ATTRIBUTE.PROVISIONKEY set to 1.

39.4.3.1 Sealing Enclave Data

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

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

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

39.4.3.2 Using REPORTs for Local Attestation

SGX provides a means for enclaves to securely identify one another, this is referred to as “Local Attestation”. SGX provides a hardware assertion, REPORT that contains calling enclaves Attributes, Measurements and User supplied data (described in detail in Section 38.15). Figure 39-3 shows the basic flow of information.

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

---

---

Figure 39-3. SGX Local Attestation
39.5  EPC AND MANAGEMENT OF EPC PAGES

EPC layout is implementation specific, and is enumerated through CPUID (see Table 37-6 for EPC layout). EPC is typically configured by BIOS at system boot time.

The exact amount, size, and layout of EPC are model-specific, and depend on BIOS settings.

39.5.1  EPC Implementation

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

39.5.2  OS Management of EPC Pages

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

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

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

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

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

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

39.5.2.1  Enhancement to Managing EPC Pages

On processors supporting SGX2 (i.e. CPUID.(EAX=12H, ECX=0):EAX.SGX2 = 1), the EPC manager can manage EPC resources (while enclave is running) with more flexibility provided by the SGX2 leaf functions. The additional flexibility is described in Section 39.5.7 through Section 39.5.11.

39.5.3  Eviction of Enclave Pages

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

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

1. For each page to be evicted from the EPC:
   a. Select an empty slot in a Version Array (VA) page.
ENCLAVE OPERATION

- If no empty VA page slots exist, create a new VA page using the EPA leaf function.

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

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

2. For each enclave containing pages selected in step 1:
   - Execute an ETRACK leaf function pointing to that enclave’s SECS. This initiates the tracking process that
     ensures that all caching of linear-address to physical-address translations for the blocked pages is cleared.

3. For all logical processors executing in processes (OS) or guests (VMM) that contain the enclaves selected in
   step 1:
   - Issue an IPI (inter-processor interrupt) to those threads. This causes those logical processors to asynchro-
nously exit any enclaves they might be in, and as a result flush cached linear-address to physical-address
   translations that might hold stale translations to blocked pages. There is no need for additional measures
   such as performing a "TLB shootdown".

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

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

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

39.5.4 Loading an Enclave Page

To reload a previously evicted page, system software needs four elements: the VA slot used when the page was
evicted, a buffer containing the encrypted page contents, a buffer containing the page metadata, and the parent
SECS to associate this page with. If the VA page or the parent SECS are not already in the EPC, they must be
reloaded first.
1. Execute ELDB/ELDU (depending on the desired BLOCKED state for the page), passing as parameters: the EPC
   page linear address, the VA slot, the encrypted page, and the page metadata.
2. Create a mapping in the enclave context’s mapping tables (page tables and EPT tables) to allow the application
   to access that page (OS: system page table; VMM: EPT).

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

39.5.5 Eviction of an SECS Page

The eviction of an SECS page is similar to the eviction of an enclave page. The only difference is that an SECS page
cannot be evicted until all other pages belonging to the enclave have been evicted. Since all other pages have been
evicted, there will be no threads executing inside the enclave and tracking with ETRACK isn’t necessary. When
reloading an enclave, the SECS page must be reloaded before all other constituent pages.
1. Ensure all pages are evicted from enclave.
2. Select an empty slot in a Version Array page.
   - If no VA page exists with an empty slot, create a new one using the EPA function leaf.
3. Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC
   page, the VA slot, a 4K byte buffer to hold the encrypted page contents and a 128 byte buffer to hold page
   metadata. The last three elements are tied together cryptographically and must be used to later reload the
   page.
39.5.6 Eviction of a Version Array Page
VA pages do not belong to any enclave and tracking with ETRACK isn’t necessary. When evicting the VA page, a slot in a different VA page must be specified in order to provide versioning of the evicted VA page.
1. Select a slot in a Version Array page other than the page being evicted.
   — If no VA page exists with an empty slot, create a new one using the EPA leaf function.
2. Evict the page using the EWB leaf function with parameters include the effective-address pointer to the EPC page, the VA slot, a 4K byte buffer to hold the encrypted page contents, and a 128 byte buffer to hold page metadata. The last three elements are tied together cryptographically and must be used to later reload the page.

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

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

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

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

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

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

39.5.9 Trimming a Page

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

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

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

39.5.10 Restricting the EPCM Permissions of a Page

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

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

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

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

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

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

39.6 CHANGES TO INSTRUCTION BEHAVIOR INSIDE AN ENCLAVE

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

39.6.1 Illegal Instructions

The instructions listed in Table 39-1 are ring 3 instructions which become illegal when executed inside an enclave. Executing these instructions inside an enclave will generate an exception.

The first row of Table 39-1 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 39-1 enumerates I/O instructions that may cause a fault or a VM exit for emulation. Again, enclave execution cannot be emulated, so execution of any of these instructions inside an enclave results in #UD.

The third row of Table 39-1 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.

The fourth row of Table 39-1 enumerates instructions that provide access to kernel information from user mode and can be used to aid kernel exploits from within enclave. Execution of any these instructions inside an enclave results in #UD.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUID, GETSEC, RDPMC, SGDT, SIDT, SLDT, STR, VMCALL, VMFUNC</td>
<td>#UD</td>
<td>Might cause VM exit.</td>
</tr>
<tr>
<td>IN, INS/INSB/INSW, OUT, OUTS/OUTSB/OUTSW/OUTSD</td>
<td>#UD</td>
<td>I/O fault may not safely recover. May require emulation.</td>
</tr>
<tr>
<td>SMSW</td>
<td>#UD</td>
<td>Might provide access to kernel information.</td>
</tr>
<tr>
<td>ENCLU[EENTER], ENCLU[ERESUME]</td>
<td>#GP</td>
<td>Cannot enter an enclave from within an enclave.</td>
</tr>
</tbody>
</table>
RDTSC and RDTSCP are legal inside an enclave for processors that support SGX2 (subject to the value of CR4.TSD). For processors which support SGX1 but not SGX2, RDTSC and RDTSCP will cause #UD. RDTSC and RDTSCP instructions may cause a VM exit when inside an enclave.
Software developers must take into account that the RDTSC/RDTSCP results are not immune to influences by other software, e.g. the TSC can be manipulated by software outside the enclave.

### 39.6.2 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 Intel SGX will not set “RDRAND exiting” to 1.

### 39.6.3 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 Intel SGX will not set “PAUSE exiting” to 1.

### 39.6.4 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 43.4.1 for details.

### 39.6.5 INVD Handling when Enclaves Are Enabled

Once processor reserved memory protections are activated (see Section 39.5), any execution of INVD will result in a #GP(0).
Certain events, such as exceptions and interrupts, incident to (but asynchronous with) enclave execution may cause control to transition outside of enclave mode. (Most of these also cause a change of privilege level.) To protect the integrity and security of the enclave, the processor will exit the enclave (and enclave mode) before invoking the handler for such an event. For that reason, such events are called 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**.

### 40.1 Compatible Switch to the Exiting Stack of AEX

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

The stack to be used is chosen using the same rules as for non-SGX mode:
- If there is a privilege level change, the stack will be the one associated with the new ring.
- If there is no privilege level change, the current application stack is used.
- If the IA-32e IST mechanism is used, the exit stack is chosen using that method.

![Figure 40-1. Exit Stack Just After Interrupt with Stack Switch](image-url)
In all cases, the choice of exit stack and the information pushed onto it is consistent with non-SGX operation. Figure 40-1 shows the Application and Exiting Stacks after an exit with a stack switch. An exit without a stack switch uses the Application Stack. The ERESUME leaf index value is placed into RAX, the TCS pointer is placed in RBX and the AEP (see below) is placed into RCX to facilitate resuming the enclave after the exit.

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

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

### 40.2 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 40-2.

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

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

When an AEX occurs, hardware selects the SSA frame to use by examining TCS.CSSA. Processor state is saved into the SSA frame (see Section 40.4) and loaded with a synthetic state (as described in Section 40.3.1) to avoid leaking secrets, RSP and RP are restored to their values prior to enclave entry, and TCS.CSSA is incremented. As will be described later, if an exception takes the last slot, it will not be possible to reenter the enclave to handle the excep-
tion from within the enclave. A subsequent ERESUME restores the processor state from the current SSA frame and frees the SSA frame.

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

### 40.3 SYNTHETIC STATE ON ASYNCHRONOUS ENCLAVE EXIT

#### 40.3.1 Processor Synthetic State on Asynchronous Enclave Exit

Table 40-1 shows the synthetic state loaded on AEX. The values shown are the lower 32 bits when the processor is in 32 bit mode and 64 bits when the processor is in 64 bit mode.

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

#### 40.3.2 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.)
40.3.3 Synthetic State for MISC Features

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

40.4 AEX FLOW

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

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

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

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

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

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

   The cause of the AEX is saved in the EXITINFO field. See Table 38-9 for details and values of the various fields.

   The state of those miscellaneous features (see Section 38.7.2) specified by SECS.MISCSELECT are stored into the MISC area of the current SSA frame.

2. Synthetic state is created for a number of processor registers to present an opaque view of the enclave state. Table 40-1 shows the values for GPRs, x87, SSE, FS, GS, Debug and performance monitoring on AEX. The synthetic state for other extended features (those controlled by XCR0[62:2]) is set to their respective INIT states when their corresponding bit of SECS.ATTRIBUTES.XFRM is set. The INIT state is that state as defined by the behavior of the XRSTOR instruction when HEADER.XSTATE_BV[n] is 0. Synthetic state of those miscellaneous features specified by SECS.MISCSELECT depends on the miscellaneous feature. There is no synthetic state required for the miscellaneous state controlled by SECS.MISCSELECT[0].

3. Any code and data breakpoints that were suppressed at the time of enclave entry are unsuppressed when exiting the enclave.

4. RFLAGS.TF is set to the value that it had at the time of the most recent enclave entry (except for the situation that the entry was opt-in for debug; see Section 43.2). In the SSA, RFLAGS.TF is set to 0.

5. RFLAGS.RF is set to 0 in the synthetic state. In the SSA, the value saved is the same as what would have been saved on stack in the non-SGX case (architectural value of RF). Thus, AEXs due to interrupts, traps, and code breakpoints save RF unmodified into SSA, while AEXs due to other faults save RF as 1 in the SSA.

   If the event causing AEX happened on intermediate iteration of a REP-prefix instruction, then RF=1 is saved on SSA, irrespective of its priority.

6. Any performance monitoring activity (including PEBS) or profiling activity (LBR, Tracing using Intel PT) on the exiting thread that was suppressed due to the enclave entry on that thread is unsuppressed. Any counting that had been demoted from AnyThread counting to MyThread counting (on one logical processor) is promoted back to AnyThread counting.
40.4.1 AEX Operational Detail

Temp Variables in AEX Operational Flow

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

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 41-4 for list of CREGs used to describe internal operation within Intel SGX *)
        SSA.RFLAGS.TF ← 0;
    ELSE    (* TMP_MODE64 = 1 *)
        Save RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8-R15, RFLAGS, RIP into the current SSA frame using CR_GPR_PA;
        SSA.RFLAGS.TF ← 0;
    FI;
Save FS and GS BASE into SSA using CR_GPR_PA;

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

(* Clear bytes 8 to 23 of XSAVE_HEADER, i.e. the next 16 bytes after XHEADER_BV *)
CR_XSAVE_PAGE_0.XHEADER_BV[191:64] ← 0;

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

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

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

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

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

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

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

(* Set EAX to the ERESUME leaf index *)
EAX ← 3;
(* Put the TCS LA into RBX for later use by ERESUME *)
RBX ← CR_TCS_LA;

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

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

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

Un-suppress all code breakpoints that are outside ELRANGE

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

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

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

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

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

(* end_of_flow *)
(* Execution continues with normal event processing. *)
ENCLAVE EXITING EVENTS
This chapter describes the supervisor and user level instructions provided by Intel® Software Guard Extensions (Intel® SGX). In general, a various functionality is 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.

### 41.1 INTEL® 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.

#### 41.1.1 ENCLS Register Usage Summary

Table 41-1 summarizes the implicit register usage of supervisor mode enclave instructions.

<table>
<thead>
<tr>
<th>Instr. Leaf</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
<th>RDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECREATE</td>
<td>00H</td>
<td>PAGEINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EADD</td>
<td>01H</td>
<td>PAGEINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EINIT</td>
<td>02H</td>
<td>SIGSTRUCT (In, EA)</td>
<td>SECS (In, EA)</td>
<td>EINITTOKEN (In, EA)</td>
</tr>
<tr>
<td>EREMOVE</td>
<td>03H</td>
<td></td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EDBGDRD</td>
<td>04H</td>
<td>Result Data (Out)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EDBGWR</td>
<td>05H</td>
<td>Source Data (In)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EEXTEND</td>
<td>06H</td>
<td>SECS (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>ELDB</td>
<td>07H</td>
<td>PAGEINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td>VERSION (In, EA)</td>
</tr>
<tr>
<td>ELDU</td>
<td>08H</td>
<td>PAGEINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td>VERSION (In, EA)</td>
</tr>
<tr>
<td>EBLOCK</td>
<td>09H</td>
<td></td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EPA</td>
<td>0AH</td>
<td>PT_VA (In)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EWB</td>
<td>0BH</td>
<td>PAGEINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td>VERSION (In, EA)</td>
</tr>
<tr>
<td>ETRACK</td>
<td>0CH</td>
<td></td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EAUG</td>
<td>0DH</td>
<td>PAGEINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td>LINADDR</td>
</tr>
<tr>
<td>EMODPR</td>
<td>0EH</td>
<td>SECINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EMODT</td>
<td>0FH</td>
<td>SECINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
</tbody>
</table>

EA: Effective Address

#### 41.1.2 ENCLU Register Usage Summary

Table 41-2 summarized the implicit register usage of user mode enclave instructions.
### 41.1.3 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 41-3 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 41-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

<table>
<thead>
<tr>
<th>Instr. Leaf</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
<th>RDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>EREPORT</td>
<td>00H (In)</td>
<td>TARGETINFO (In, EA)</td>
<td>REPORTDATA (In, EA)</td>
<td>OUTPUTDATA (In, EA)</td>
</tr>
<tr>
<td>EGETKEY</td>
<td>01H (In)</td>
<td>KEYREQUEST (In, EA)</td>
<td>KEY (In, EA)</td>
<td></td>
</tr>
<tr>
<td>ENTER</td>
<td>02H (In)</td>
<td>TCS (In, EA)</td>
<td>AEP (In, EA)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EBX.CSSA (Out)</td>
<td></td>
<td>Return (Out, EA)</td>
<td></td>
</tr>
<tr>
<td>ERESUME</td>
<td>03H (In)</td>
<td>TCS (In, EA)</td>
<td>AEP (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>04H (In)</td>
<td>Target (In, EA)</td>
<td>Current AEP (Out)</td>
<td></td>
</tr>
<tr>
<td>EACCEPT</td>
<td>05H (In)</td>
<td>SECINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EMODPE</td>
<td>06H (In)</td>
<td>SECINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
<tr>
<td>EACCEPTCOPY</td>
<td>07H (In)</td>
<td>SECINFO (In, EA)</td>
<td>EPCPAGE (In, EA)</td>
<td></td>
</tr>
</tbody>
</table>

EA: Effective Address

#### Table 41-3. Error or Information Codes for Intel® SGX Instructions

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Returned By</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SGX_INVALID_SIG_STRUCT</td>
<td>1</td>
<td>EINIT</td>
</tr>
<tr>
<td>SGX_INVALID_ATTRIBUTE</td>
<td>2</td>
<td>EINIT, EGETKEY</td>
</tr>
<tr>
<td>SGX_BSTATE</td>
<td>3</td>
<td>EBLOCK</td>
</tr>
<tr>
<td>SGX_INVALID_MEASUREMENT</td>
<td>4</td>
<td>EINIT</td>
</tr>
<tr>
<td>SGX_NOTBLOCKABLE</td>
<td>5</td>
<td>EBLOCK</td>
</tr>
<tr>
<td>SGX_PG_INVLD</td>
<td>6</td>
<td>EBLOCK</td>
</tr>
<tr>
<td>SGX_LOCKFAIL</td>
<td>7</td>
<td>EBLOCK, EMODPR, EMODT</td>
</tr>
<tr>
<td>SGX_INVALID_SIGNATURE</td>
<td>8</td>
<td>EINIT</td>
</tr>
<tr>
<td>SGX_MACCOMPARE_FAIL</td>
<td>9</td>
<td>ELDB, ELDU</td>
</tr>
<tr>
<td>SGX_PAGE_NOT_BLOCKED</td>
<td>10</td>
<td>EWB</td>
</tr>
<tr>
<td>SGX_NOT_TRACKED</td>
<td>11</td>
<td>EWB, EACCEPT</td>
</tr>
<tr>
<td>SGX_VA_SLOT_OCCUPIED</td>
<td>12</td>
<td>EWB</td>
</tr>
<tr>
<td>SGX_CHILD_PRESENT</td>
<td>13</td>
<td>EWB, EREMOVE</td>
</tr>
<tr>
<td>SGX_ENCLAVE_ACT</td>
<td>14</td>
<td>EREMOVE</td>
</tr>
<tr>
<td>SGX_ENTRYEPOCH_LOCKED</td>
<td>15</td>
<td>EBLOCK</td>
</tr>
<tr>
<td>SGX_INVALID_EINIT_TOKEN</td>
<td>16</td>
<td>EINIT</td>
</tr>
<tr>
<td>SGX_PREV_TRK_INCMPL</td>
<td>17</td>
<td>ETRACK</td>
</tr>
<tr>
<td>SGX_PG_IS_SEC</td>
<td>18</td>
<td>EBLOCK</td>
</tr>
<tr>
<td>SGX_PAGE_ATTRIBUTES_MISMATCH</td>
<td>19</td>
<td>EACCEPT, EACCEPTCOPY</td>
</tr>
<tr>
<td>SGX_PAGE_NOT_MODIFIABLE</td>
<td>20</td>
<td>EMODPR, EMODT</td>
</tr>
</tbody>
</table>
| SGX_PAGE_NOT_DEBUGGABLE     | 21    | EDEGRD, EDBGWR
41.1.4 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 Intel SGX instruction.
These registers are not software visible and are implementation specific. The values in Table 41-4 appear at various
places in the pseudo-code of this document. They are used to enhance understanding of the operations.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Returned By</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX_INVALID_CPUSVN</td>
<td>32</td>
<td>EINIT, EGETKEY</td>
</tr>
<tr>
<td>SGX_INVALID_ISVSVN</td>
<td>64</td>
<td>EGETKEY</td>
</tr>
<tr>
<td>SGX_UNMASKED_EVENT</td>
<td>128</td>
<td>EINIT</td>
</tr>
<tr>
<td>SGX_INVALID_KEYNAME</td>
<td>256</td>
<td>EGETKEY</td>
</tr>
</tbody>
</table>

### Table 41-3. Error or Information Codes for Intel® SGX Instructions

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Returned By</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX_INVALID_CPUSVN</td>
<td>32</td>
<td>EINIT, EGETKEY</td>
</tr>
<tr>
<td>SGX_INVALID_ISVSVN</td>
<td>64</td>
<td>EGETKEY</td>
</tr>
<tr>
<td>SGX_UNMASKED_EVENT</td>
<td>128</td>
<td>EINIT</td>
</tr>
<tr>
<td>SGX_INVALID_KEYNAME</td>
<td>256</td>
<td>EGETKEY</td>
</tr>
</tbody>
</table>

41.1.5 Concurrent Operation Restrictions

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

- For example, Intel SGX disallows the following leafs to concurrently operate on the same EPC page.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Returned By</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX_INVALID_CPUSVN</td>
<td>32</td>
<td>EINIT, EGETKEY</td>
</tr>
<tr>
<td>SGX_INVALID_ISVSVN</td>
<td>64</td>
<td>EGETKEY</td>
</tr>
<tr>
<td>SGX_UNMASKED_EVENT</td>
<td>128</td>
<td>EINIT</td>
</tr>
<tr>
<td>SGX_INVALID_KEYNAME</td>
<td>256</td>
<td>EGETKEY</td>
</tr>
</tbody>
</table>

### Table 41-4. List of Internal CREG

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (Bits)</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR_ENCLAVE_MODE</td>
<td>1</td>
<td>LP</td>
</tr>
<tr>
<td>CR_DBGOPTIN</td>
<td>1</td>
<td>LP</td>
</tr>
<tr>
<td>CR_TCS_LA</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_TCS_PH</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_ACTIVE_SECS</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_ELRANGE</td>
<td>128</td>
<td>LP</td>
</tr>
<tr>
<td>CR_SAVE_TF</td>
<td>1</td>
<td>LP</td>
</tr>
<tr>
<td>CR_SAVE_FS</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_GPR_PA</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_XSAVE_PAGE_n</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_SAVE_DR7</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_SAVE_PERF_GLOBAL_CTRL</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_SAVE_DEBUGCTL</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_SAVE_PEBBS_ENABLE</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_CPUSVN</td>
<td>128</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>CSR_SGX_OWNEREPOCH</td>
<td>128</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>CR_SAVE_XCRO</td>
<td>64</td>
<td>LP</td>
</tr>
<tr>
<td>CR_SGX_ATTRIBUTES_MASK</td>
<td>128</td>
<td>LP</td>
</tr>
<tr>
<td>CR_PAGING_VERSION</td>
<td>64</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>CR_VERSION_THRESHOLD</td>
<td>64</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>CR_NEXT_EID</td>
<td>64</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>CR_BASE_PK</td>
<td>128</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>CR_SEAL_FUSES</td>
<td>128</td>
<td>PACKAGE</td>
</tr>
</tbody>
</table>
— ECREATE, EADD, and EREMOVE are not allowed to operate on the same EPC page concurrently with themselves.

— EADD, EEXTEND, and EINIT leafs are not allowed to operate on the same SECS concurrently.

• Intel SGX disallows the EREMOVE leaf from removing pages from an enclave that is in use.

• Intel SGX disallows entry (EENTER and ERESUME) to an enclave while a page from that enclave is being removed.

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

41.1.5.1 Concurrency Tables of Intel® SGX Instructions

Concurrent restriction of an individual leaf function (ENCLS or ENCLU) with another Intel SGX instruction leaf functions is listed under the Concurrency Restriction paragraph of the respective reference pages of the leaf function.

Each cell in the table for a given Intel SGX Instruction leaf details the concurrency restriction when that instruction references the same EPC page (as an explicit or an implicit parameter) as referenced by a concurrent instruction leaf executed on another logical processor. The concurrency behavior of the instruction leaf if focus shown in a given row is denoted by the following:

• 'N': The instructions listed in a given row heading may not execute concurrently with the instruction leaf shown in the respective column. Software should serialize them. For example, multiple ETRACK operations on the same enclave are not allowed to execute concurrently on the same SECS page.

• 'Y': The instruction leaf listed in a given row may execute concurrently with the instruction leaf shown in the respective column. For instance, multiple ELDB/ELDUs are allowed to execute concurrently as long as the selected EPC page is not the same page.

• 'C': The instruction leaf listed in a given row heading may return an error code when executed concurrently with the instruction leaf shown in the respective column.

• 'U': These two instruction leaves may complete, but the occurrence these two simultaneous flows are considered a user program error for which the processor does not enforce any restriction.

• A grey cell indicates the concurrency behavior of the instruction in focus (in the row header) may be different than that of the concurrent instruction (in the column header). The concurrent instruction's behavior is detailed in its respective concurrency table. For example, EBLOCK's SECS parameter is implicit, thus it is always shown as 'Y' in the table. However a concurrent instruction may return an error code when accessing the same page.

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.

41.2 INTEL® SGX INSTRUCTION REFERENCE
ENCLS—Execute an Enclave System Function of Specified Leaf Number

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCLS</td>
<td>NP</td>
<td>V/V</td>
<td>SGX1</td>
<td>This instruction is used to execute privileged Intel SGX leaf functions that are used for managing and debugging the enclaves.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Operand 3</th>
<th>Implicit Register Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>See Section 41.3</td>
</tr>
</tbody>
</table>

**Description**

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

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

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

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

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

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

**Operation**

IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

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

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

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

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

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

Jump to leaf specific flow

**Flags Affected**
See individual leaf functions

**Protected Mode Exceptions**

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

**Real-Address Mode Exceptions**

- **#UD** ENCLS is not recognized in real mode.

**Virtual-8086 Mode Exceptions**

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

**Compatibility Mode Exceptions**

Same exceptions as in protected mode.

**64-Bit Mode Exceptions**

- **#UD** If any of the LOCK/OSIZE/REP/VEX prefix is used.
  - If current privilege level is not 0.
  - If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
  - If logical processor is in SMM.
- **#GP(0)** If IA32_FEATURE_CONTROL.LOCK = 0.
  - If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
  - If input value in EAX encodes an unsupported leaf.
ENCLU—Execute an Enclave User Function of Specified Leaf Number

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OF 01 D7</td>
<td>NP</td>
<td>V/V</td>
<td>SGX1</td>
<td>This instruction is used to execute non-privileged Intel SGX leaf functions.</td>
</tr>
</tbody>
</table>

Description

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

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

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

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

Operation

IN_64BIT_MODE ← 0;
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

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

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

IF CPL < 3
    THEN #UD; FI;

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

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

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

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

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

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

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 CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
If logical processor is in SMM.
#GP(0) If IA32_FEATURE_CONTROL.LOCK = 0.
If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
If input value in EAX encodes an unsupported leaf.
If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1.
If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE
and ENCLAVE_MODE = 0.
If operating in 16-bit mode.
If data segment is in 16-bit mode.
If CR0.PG = 0 or CR0.NE= 0.
#NM If CR0.TS = 1.

Real-Address Mode Exceptions
#UD ENCLS is not recognized in real mode.

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

Compatibility Mode Exceptions
Same exceptions as in protected mode.

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

#NM If CR0.TS = 1.
41.3 INTEL® SGX SYSTEM LEAF FUNCTION REFERENCE

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

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

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

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EADD (In)</td>
<td>Address of a PAGEINFO (In)</td>
<td>Address of the destination EPC page (In)</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>PAGEINFO</th>
<th>PAGEINFO.SECs</th>
<th>PAGEINFO.SRC PG</th>
<th>PAGEINFO.SECINFO</th>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read/Write access permitted by Enclave</td>
<td>Read access permitted by Non Enclave</td>
<td>Read access permitted by Non Enclave</td>
<td>Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

The instruction faults if any of the following:

**EADD Faulting Conditions**

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

**Concurrency Restrictions**

Table 41-5. Concurrency Restrictions of EADD with Other Intel® SGX Operations 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEIXT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGDRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
</tr>
<tr>
<td>EADD</td>
<td>Targ</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SECS</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Table 41-6. Concurrency Restrictions of EADD with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EADD</td>
<td>Targ</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SECS</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Temp Variables in EADD Operational Flow

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

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

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

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

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); Fl;

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

SCRATCH_SECINFO ← DS: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); Fl;
(* Check the EPC page for concurrency *)
IF (EPC page in use)
    THEN #GP(0); FI;

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

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

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

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

CASE (SCRATCH_SECINFO.FLAGS.PT)
{
    PT_TCS:
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
        IF (DS:TMP_SECS.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 *)
SGX INSTRUCTION REFERENCES

TMP_ENCLAVEOFFSET ← Tmp_LinAddr - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] ← 0000000044444145H; // “EADD”
TMPUPDATEFIELD[127:64] ← TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] ← SCRATCH_SECINFO[375:0]; // 48 bytes
DS:TMP_SECS.MRENCLAVE ← SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

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

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

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

Flags Affected
None

Protected Mode Exceptions

#GP(0)  If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If an enclave memory operand is outside of the EPC.
If an enclave memory operand is the wrong type.
If a memory operand is locked.
If the enclave is initialized.
If the enclave's MRENCLAVE is locked.
If the TCS page reserved bits are set.

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

64-Bit Mode Exceptions

#GP(0)  If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If an enclave memory operand is outside of the EPC.
If an enclave memory operand is the wrong type.
If a memory operand is locked.
If the enclave is initialized.
If the enclave's MRENCLAVE is locked.
If the TCS page reserved bits are set.

#PF(error code)  If a page fault occurs in accessing memory operands.
If the EPC page is valid.
EAUG—Add a Page to an Initialized Enclave

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode</th>
<th>CPUID Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 0DH</td>
<td>IR</td>
<td>V/V</td>
<td>SGX2</td>
<td>This leaf function adds a page to an initialized enclave.</td>
</tr>
<tr>
<td>ENCLS[EAUG]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EAX (In)</td>
<td>Address of a SECINFO (In)</td>
<td>Address of the destination EPC page (In)</td>
</tr>
</tbody>
</table>

**Description**

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

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

**EAUG Memory Parameter Semantics**

<table>
<thead>
<tr>
<th>PAGEINFO</th>
<th>PAGEINFO.SECS</th>
<th>PAGEINFO.SRCPGE</th>
<th>PAGEINFO.SECSINFO</th>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read/Write access permitted by Enclave</td>
<td>Must be zero</td>
<td>Read access permitted by Non Enclave</td>
<td>Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

The instruction faults if any of the following:

**EAUG Faulting Conditions**

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

**Concurrency Restrictions**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREAT</th>
<th>EDBGRAWR</th>
<th>EENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SSA</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>EAUG</td>
<td>Targ</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Operation

Temp Variables in EAUG Operational Flow

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

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

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

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

TMP_SECS ← DS:RBX:SECS;
TMP_LINADDR ← DS:RBX:LINADDR;

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

IF ( (DS:RBX:SRCPAGE is not 0) or (DS:RBX:SECINFO is not 0) )
    THEN #GP(0); Fl;

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

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

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

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

Table 41-8. Concurrency Restrictions of EAUG with Other Intel* SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
</tr>
<tr>
<td>EAUG</td>
<td></td>
<td>N</td>
<td>N</td>
<td></td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table 49-8. Concurrency Restrictions of EAUG with Other Intel* SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
</tr>
<tr>
<td>EAUG</td>
<td></td>
<td>N</td>
<td>N</td>
<td></td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table 49-8. Concurrency Restrictions of EAUG with Other Intel* SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
</tr>
<tr>
<td>EAUG</td>
<td></td>
<td>N</td>
<td>N</td>
<td></td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>
IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

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

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

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

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

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

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

Flags Affected
None

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

64-Bit Mode Exceptions
#GP(0)   If a memory operand is non-canonical form.
         If a memory operand is not properly aligned.
         If a memory operand is locked.
         If the enclave is not initialized.
#PF(error code) If a page fault occurs in accessing memory operands.
**EBLOCK—Mark a page in EPC as Blocked**

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

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EBLOCK (In)</td>
<td>Return error code (Out)</td>
</tr>
</tbody>
</table>

**Description**

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

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

An error code is returned in RAX.

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

**EBLOCK Memory Parameter Semantics**

- **EPCPAGE**: Read/Write access permitted by Enclave

The error codes are:

**Table 41-9. EBLOCK Return Value in RAX**

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EBLOCK successful</td>
</tr>
<tr>
<td>SGX_BLKSTATE</td>
<td>Page already blocked. This value is used to indicate to a VMM that the page was already in BLOCKED state as a result of EBLOCK and thus will need to be restored to this state when it is eventually reloaded (using ELDB).</td>
</tr>
<tr>
<td>SGX_ENTRYEPOCH_LOCKED</td>
<td>SECS locked for Entry Epoch update. This value indicates that an ETRACK is currently executing on the SECS. The EBLOCK should be reattempted.</td>
</tr>
<tr>
<td>SGX_NOTBLOCKABLE</td>
<td>Page type is not one which can be blocked</td>
</tr>
<tr>
<td>SGX_PG_INVLD</td>
<td>Page is not valid and cannot be blocked</td>
</tr>
<tr>
<td>SGX_LOCKFAIL</td>
<td>Page is being written by EADD, EALUG, ECREATE, ELDU/B, EMODT, or EWBB</td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

**Table 41-10. Concurrency Restrictions of EBLOCK with Other Intel® SGX Operations 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEEXIT</th>
<th>EADDR</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGDRD/WR</th>
<th>EEENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>Targ VA</td>
</tr>
<tr>
<td>EBLOCK</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>Y</td>
<td>C</td>
<td>Y</td>
<td>C</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Table 41-11. Concurrency Restrictions of EBLOCK with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBLOCK</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ SFC</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>PARAM</td>
<td>Targ</td>
<td>SECS</td>
<td>SRC</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SRC SFC</td>
</tr>
<tr>
<td>PARAM</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SRC SFC</td>
</tr>
</tbody>
</table>

Operation

Temp Variables in EBLOCK Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_BLKSTATE</td>
<td>Integer</td>
<td>64</td>
<td>Page is already blocked.</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

(* at this point, the page must be valid and PT_TCS or PT_REG or PT_TRIM*)
IF (TMP_BLKSTATE = 1)
    THEN
        RFLAGS.CF ← 1;
        RAX ← SGX_BLKSTATE;
    ELSE
        EPCM(DS:RCX).BLOCKED ← 1
    Fl;
DONE:
Flags Affected
Sets ZF if SECS is in use or invalid, otherwise cleared. Sets CF if page is BLOCKED or not blockable, otherwise cleared. Clears PF, AF, OF, SF.

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

64-Bit Mode Exceptions
#GP(0)  If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If the specified EPC resource is in use.
#PF(error code)  If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
**ECREATE—Create an SECS page in the Enclave Page Cache**

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>ECREATE (In)</td>
<td>Address of a PAGEINFO (In)</td>
<td>Address of the destination SECS page (In)</td>
</tr>
</tbody>
</table>

**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 a 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**

<table>
<thead>
<tr>
<th>PAGEINFO</th>
<th>PAGEINFO.SRCPGE</th>
<th>PAGEINFO.SECINFO</th>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read access permitted by Non Enclave</td>
<td>Read access permitted by Non Enclave</td>
<td>Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

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 42.7.

**Concurrency Restrictions**

**Table 41-12. Concurrency Restrictions of ECREATE with Other Intel® SGX Operations 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGDRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
</tr>
<tr>
<td>ECREATE</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
</tr>
</tbody>
</table>

**Table 41-13. Concurrency Restrictions of ECREATE with Other Intel® SGX Operations 2 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EwB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODDT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Tar g</td>
<td>SECS</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>ECREATE</td>
<td>SECS</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Operation

Temp Variables in ECREATE Operational Flow

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

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

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

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

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); Fl;

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

(* 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); Fl;

TMP_SECS ← RCX;

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

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

(* 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); Fl;

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

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

(* Compute size of MISC area *)
TMP_MISC_SIZE ← compute_misc_region_size();

(* Compute the size required to save state of the enclave on async exit, see Section 42.7.2.2*)
TMP_XSIZE ← compute_xsave_size(DS:TMP_SECS.ATTRIBUTES.XFRM) + GPR_SIZE + TMP_MISC_SIZE;

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

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

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.BASEADDR and 0FFFFFFFD0000000H) )
   THEN #GP(0); Fi;

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

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

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

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

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

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

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).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).PR ← 0;
EPCM(DS:RCX).VALID ← 1;

Flags Affected
None

Protected Mode Exceptions

#GP(0)  If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If the reserved fields are not zero.
If PAGEINFO.SECS is not zero.
If PAGEINFO.LINADDR is not zero.
If the SECS destination is locked.
If SECS.SSAFRAMESIZE is insufficient.

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

64-Bit Mode Exceptions

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

#PF(error code)  If a page fault occurs in accessing memory operands.
If the SECS destination is outside the EPC.
EDBGRD—Read From a Debug Enclave

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

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EDBGRD (In)</td>
<td>Data read from a debug enclave (Out)</td>
<td>Address of source memory in the EPC (In)</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>EPCQW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Enclave</td>
</tr>
</tbody>
</table>

The error codes are:

**Table 41-14. EDBGRD Return Value in RAX**

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EDBGRD successful</td>
</tr>
<tr>
<td>SGX_PAGE_NOT_DEBUGGABLE</td>
<td>The EPC page cannot be accessed because it is in the PENDING or MODIFIED state</td>
</tr>
</tbody>
</table>

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.
### Concurrency Restrictions

#### Table 41-15. Concurrency Restrictions of EDBGRD with Other Intel® SGX Operations 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
</tr>
<tr>
<td>EDBGRD</td>
<td>Targ</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

#### Table 41-16. Concurrency Restrictions of EDBGRD with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODEP</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EAACCEPT</th>
<th>EAACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
</tr>
<tr>
<td>EDBGRD</td>
<td>Targ</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Temp Variables in EDBGRD Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_MODE64</td>
<td>Binary</td>
<td>1</td>
<td>(((IA32_EFER.LMA = 1) &amp;&amp; (CS.L = 1)))</td>
</tr>
<tr>
<td>TMP_SECS</td>
<td></td>
<td>64</td>
<td>Physical address of SECS of the enclave to which source operand belongs</td>
</tr>
</tbody>
</table>

\[
\text{TMP\_MODE64} \leftarrow \(((\text{IA32\_EFER.LMA} = 1) \&\& (\text{CS.L} = 1))\);
\]

IF ( (TMP\_MODE64 = 1) and (DS:RCX is not 8Byte Aligned) )
    THEN #GP(0); Fi;

IF ( (TMP\_MODE64 = 0) and (DS:RCX is not 4Byte Aligned) )
    THEN #GP(0); Fi;

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

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

IF (EPCM(DS:RCX), VALID = 0)
    THEN #PF(DS:RCX); Fi;

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

(* If source is a TCS, then make sure that the offset into the page is not beyond the TCS size*)
IF ( (EPCM(DS:RCX), PT = PT_TCS) and ((DS:RCX) & FFFH ≥ SGX\_TCS\_LIMIT) )
    THEN #GP(0); Fi;
(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF ((EPCM(DS:RCX).PT = PT_REG) or (EPCM(DS:RCX).PT = PT_TCS))
  THEN
    TMP_SECS ← GET_SECS_ADDRESS;
    IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
      THEN #GP(0); FI;
    IF ((TMP_MODE64 = 1))
      THEN RBX[63:0] ← (DS:RCX)[63:0];
      ELSE EBX[31:0] ← (DS:RCX)[31:0];
      FI;
  ELSE
    TMP_64BIT_VAL[63:0] ← (DS:RCX)[63:0] & (~07H); // Read contents from VA slot
    IF (TMP_MODE64 = 1)
      THEN
        IF (TMP_64BIT_VAL ≠ 0H)
          THEN RBX[63:0] ← 0FFFFFFFFFFFFFFFH;
          ELSE RBX[63:0] ← 0H;
          FI;
      ELSE
        IF (TMP_64BIT_VAL ≠ 0H)
          THEN EBX[31:0] ← 0FFFFFFFFH;
          ELSE EBX[31:0] ← 0H;
          FI;
      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.
#PF(error code) If a page fault occurs in accessing memory operands.
If the address in RCX points to a non-EPC page.
If the address in RCX points to an invalid EPC page.

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

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

### Instruction Operand Encoding

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EDBGWR (In)</td>
<td>Data to be written to a debug enclave (In)</td>
<td>Address of Target memory in the EPC (In)</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>EPCQW</th>
</tr>
</thead>
</table>
Write access permitted by Enclave

The instruction faults if any of the following:

### EDBGWR Faulting Conditions

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

The error codes are:

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EDBGWR successful</td>
</tr>
<tr>
<td>SGX_PAGE_NOT_DEBUGGABLE</td>
<td>The EPC page cannot be accessed because it is in the PENDING or MODIFIED state</td>
</tr>
</tbody>
</table>

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.
Concurrency Restrictions

**Table 41-18. Concurrency Restrictions of EDBGWR with Other Intel® SGX Operations 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGWR/WR</th>
<th>EENTER/ERESUME</th>
<th>EEXIT</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDBGWR</td>
<td>Targ</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Table 41-19. Concurrency Restrictions of EDBGWR with Other Intel® SGX Operations 2 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>ERMOVE</th>
<th>REPORT</th>
<th>ETRACK</th>
<th>EwB</th>
<th>EAUG</th>
<th>EMODEPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDBGWR</td>
<td>Targ</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Operation**

**Temp Variables in EDBGWR Operational Flow**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_MODE64</td>
<td>Binary</td>
<td>1</td>
<td>((IA32_EFER.LMA = 1) &amp;&amp; (CS.L = 1)).</td>
</tr>
<tr>
<td>TMP_SECS</td>
<td>64</td>
<td></td>
<td>Physical address of SECS of the enclave to which source operand belongs.</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

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

(* Locate the SECS for the enclave to which the DS:RCX page belongs *)
TMP_SECS ← GET_SECS_PHYS_ADDRESS(EPCM(DS:RCX).ENCLAVESCES);
```
SGX INSTRUCTION REFERENCES

(* 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.

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

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

#PF(error code) If a page fault occurs in accessing memory operands.
    If the address in RCX points to a non-EPC page.
    If the address in RCX points to an invalid EPC page.
**EEXTEND—Extend Uninitialized Enclave Measurement by 256 Bytes**

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

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th></th>
<th>EAX</th>
<th>EBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op/En</td>
<td>EEXTEND (In)</td>
<td>Effective address of the SECS of the data chunk (In)</td>
<td>Effective address of a 256-byte chunk in the EPC (In)</td>
</tr>
</tbody>
</table>

**Description**

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

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

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

**EEXTEND Memory Parameter Semantics**

| EPC[RCX] |
| Read access by Enclave |

The instruction faults if any of the following:

**EEXTEND Faulting Conditions**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RBX points to an address not 4KBytes aligned.</td>
<td>RBX does not resolve to an SECS.</td>
</tr>
<tr>
<td>RBX does not point to an SECS page.</td>
<td>RBX does not point to the SECS page of the data chunk.</td>
</tr>
<tr>
<td>RCX points and address not 256B aligned.</td>
<td>RCX does not resolve in an EPC page.</td>
</tr>
<tr>
<td>RCX does not resolve in an EPC page.</td>
<td>If SECS is locked.</td>
</tr>
<tr>
<td>If the SECS is already initialized.</td>
<td>May page fault.</td>
</tr>
<tr>
<td>CPL &gt; 0.</td>
<td></td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREAT</th>
<th>EDBGRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEXTEND</th>
<th>EGGET</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
</tr>
<tr>
<td>EEXIT</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>EADD</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>EBLOCK</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ECREAT</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>EDBGRD/WR</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>EENTER/ERESUME</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 41-20. Concurrency Restrictions of EEXTEND with Other Intel® SGX Operations 1 of 2

Vol. 3D 41-31
Table 41-21. Concurrency Restrictions of EEXTEND with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEXTEND</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ S</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Operation

Temp Variables in EEXTEND Operational Flow

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

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

IF (DS:RBX does resolve to an EPC page)
    THEN #PF(DS:RBX); FI;

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

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

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

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

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

(* 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 \*)

\[
\text{TMPUPDATEFIELD}[63:0] \leftarrow 00444E4554584545H; // "EEXTEND"
\]

\[
\text{TMPUPDATEFIELD}[127:64] \leftarrow \text{TMP\_ENCLAVE\_OFFSET};
\]

\[
\text{TMPUPDATEFIELD}[511:128] \leftarrow 0; // 48 bytes
\]

\[
\text{TMP\_SECS\_MRENCLAVE} \leftarrow \text{SHA256\_UPDATE} (\text{TMP\_SECS\_MRENCLAVE, TMPUPDATEFIELD})
\]

INC enclave's MRENCLAVE update counter;

(\*Add 256 bytes to MRENCLAVE, 64 byte at a time \*)

\[
\text{TMP\_SECS\_MRENCLAVE} \leftarrow \text{SHA256\_UPDATE} (\text{TMP\_SECS\_MRENCLAVE, DS:RCX}[511:0]);
\]

\[
\text{TMP\_SECS\_MRENCLAVE} \leftarrow \text{SHA256\_UPDATE} (\text{TMP\_SECS\_MRENCLAVE, DS:RCX}[1023:512]);
\]

\[
\text{TMP\_SECS\_MRENCLAVE} \leftarrow \text{SHA256\_UPDATE} (\text{TMP\_SECS\_MRENCLAVE, DS:RCX}[1535:1024]);
\]

\[
\text{TMP\_SECS\_MRENCLAVE} \leftarrow \text{SHA256\_UPDATE} (\text{TMP\_SECS\_MRENCLAVE, DS:RCX}[2047:1536]);
\]

INC enclave's MRENCLAVE update counter by 4;

**Flags Affected**

None

**Protected Mode Exceptions**

\#GP(0) If the address in RBX is outside the DS segment limit.

If RBX points to an SECS page which is not the SECS of the data chunk.

If the address in RCX is outside the DS segment limit.

If RCX points to a memory location not 256Byte-aligned.

If another instruction is accessing MRENCLAVE.

If another instruction is checking or updating the SECS.

If the enclave is already initialized.

\#PF(error code) If a page fault occurs in accessing memory operands.

If the address in RBX points to a non-EPC page.

If the address in RCX points to a page which is not PT\_TCS or PT\_REG.

If the address in RCX points to a non-EPC page.

If the address in RCX points to an invalid EPC page.

**64-Bit Mode Exceptions**

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

If RBX points to an SECS page which is not the SECS of the data chunk.

If RCX is non-canonical form.

If RCX points to a memory location not 256 Byte-aligned.

If another instruction is accessing MRENCLAVE.

If another instruction is checking or updating the SECS.

If the enclave is already initialized.

\#PF(error code) If a page fault occurs in accessing memory operands.

If the address in RBX points to a non-EPC page.

If the address in RCX points to a page which is not PT\_TCS or PT\_REG.

If the address in RCX points to a non-EPC page.

If the address in RCX points to an invalid EPC page.
EINIT—Initialize an Enclave for Execution

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINIT</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function initializes the enclave and makes it ready to execute enclave code.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>Error code (Out)</th>
<th>RBX</th>
<th>Address of SIGSTRUCT (In)</th>
<th>RCX</th>
<th>Address of SECS (In)</th>
<th>RDX</th>
<th>Address of EINITTOKEN (In)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EINIT</td>
<td>(In)</td>
<td>RBX</td>
<td>(In)</td>
<td>RCX</td>
<td>(In)</td>
<td>RDX</td>
<td>(In)</td>
</tr>
</tbody>
</table>

**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:

\[
q_1 = \text{floor}(\text{Signature}^2 / \text{Modulus});
\]

\[
q_2 = \text{floor}((\text{Signature}^3 - q_1 \times \text{Signature} \times \text{Modulus}) / \text{Modulus});
\]

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 41-1. Relationships Between SECS, SIGSTRUCT and EINITTOKEN**
EINIT performs the following steps, which can be seen in Figure 41-1:
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 controlled ATTRIBUTES bits are set in SIGSTRUCT.ATTRIBUTES unless the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
Checks that SIGSTRUCT.ATTRIBUTES equals the result of logically and-ing SIGSTRUCT.ATTRIBUTESMASK with SECS.ATTRIBUTES.
If EINITTOKEN.VALID is 0, checks that the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
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 includes external interrupts, non-maskable interrupts, system-management interrupts, machine checks, INIT signals, and the VMX-preemption timer. EINIT does not fail if the pending event is inhibited (e.g., external interrupts could be inhibited due to blocking by MOV SS blocking or by STI).
The following bits in RFLAGS are cleared: CF, PF, AF, OF, and SF. When the instruction completes with an error, RFLAGS.ZF is set to 1, and the corresponding error bit is set in RAX. If no error occurs, RFLAGS.ZF is cleared and RAX is set to 0.
The error codes are:

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EINIT successful</td>
</tr>
<tr>
<td>SGX_INVALID_SIG_STRUCT</td>
<td>If SIGSTRUCT contained an invalid value</td>
</tr>
<tr>
<td>SGX_INVALID_ATTRIBUTE</td>
<td>If SIGSTRUCT contains an unauthorized attributes mask</td>
</tr>
<tr>
<td>SGX_INVALID_MEASUREMENT</td>
<td>If SIGSTRUCT contains an incorrect measurement</td>
</tr>
<tr>
<td>SGX_INVALID_MEASUREMENT</td>
<td>If EINITTOKEN contains an incorrect measurement</td>
</tr>
<tr>
<td>SGX_INVALID_SIGNATURE</td>
<td>If signature does not validate with enclosed public key</td>
</tr>
<tr>
<td>SGX_INVALID_LICENSE</td>
<td>If license is invalid</td>
</tr>
<tr>
<td>SGX_INVALID_CPUSVN</td>
<td>If license SVN is unsupported</td>
</tr>
<tr>
<td>SGX_UNMASKED_EVENT</td>
<td>If an unmasked event is received before the instruction completes its operation</td>
</tr>
</tbody>
</table>
Concurrence Restrictions

**Table 41-23. Concurrency Restrictions of EINIT with Other Intel® SGX Operations 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGDRD/WR</th>
<th>ENTERERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINIT</td>
<td>SECS</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Table 41-24. Concurrency Restrictions of EINIT with Other Intel® SGX Operations 2 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPCTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINIT</td>
<td>SECS</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Operation**

**Temp Variables in EINIT Operational Flow**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_SIG</td>
<td>SIGSTRUCT</td>
<td>1808Bytes</td>
<td>Temp space for SIGSTRUCT.</td>
</tr>
<tr>
<td>TMP_TOKEN</td>
<td>EINITTOKEN</td>
<td>304Bytes</td>
<td>Temp space for EINITTOKEN.</td>
</tr>
<tr>
<td>TMP_MRENCLAVE</td>
<td></td>
<td>32Bytes</td>
<td>Temp space for calculating MRENCLAVE.</td>
</tr>
<tr>
<td>TMP_MRSIGNER</td>
<td></td>
<td>32Bytes</td>
<td>Temp space for calculating MRSIGNER.</td>
</tr>
<tr>
<td>CONTROLLED_ATTRIBUTES</td>
<td>ATTRIBUTES</td>
<td>16Bytes</td>
<td>Constant mask of all ATTRIBUTE bits that can only be set for authorized enclaves.</td>
</tr>
<tr>
<td>TMP_KEYDEPENDENCIES</td>
<td>Buffer</td>
<td>224Bytes</td>
<td>Temp space for key derivation.</td>
</tr>
<tr>
<td>TMP_EINITTOKENKEY</td>
<td>Buffer</td>
<td>16Bytes</td>
<td>Temp space for the derived EINITTOKEN Key.</td>
</tr>
<tr>
<td>TMP_SIG_PADDING</td>
<td>PKCS Padding Buffer</td>
<td>352Bytes</td>
<td>The value of the top 352 bytes from the computation of Signature³ modulo MRSIGNER.</td>
</tr>
</tbody>
</table>

(* make sure SIGSTRUCT and SECS are aligned *)

IF ( (DS:RBX is not 4KByte Aligned) or (DS:RCX is not 4KByte Aligned) )

THEN #GP(0); FI;

(* make sure the EINITTOKEN is aligned *)

IF (DS:RDX is not 512Byte Aligned)

THEN #GP(0); FI;

(* make sure the SECS is inside the EPC *)

IF (DS:RCX does not resolve within an EPC)

THEN #PF(DS:RCX); FI;

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

(* Verify SIGSTRUCT Header. *)

IF ( (TMP_SIG.HEADER ≠ 06000000E1000000000001000000000000h) or

((TMP_SIG.VENDOR ≠ 0) and (TMP_SIG.VENDOR ≠ 0008086 Eh)) or

(TMP_SIG HEADER2 ≠ 010100006000000600000000001000000h) or


41-36 Vol. 3D
(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 Intel SGX instruction is modifying SECS*)

IF (Other instructions modifying SECS)
    THEN #GP(0);

FI

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

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTUETS.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 controlled ATTRIBUTES are set, SIGSTRUCT must be signed using an authorized key *)

CONTROLLED_ATTRIBUTES ← 0000000000000020H;

IF ( ((DS:RCX.ATTRIBUTES & CONTROLLED_ATTRIBUTES) ≠ 0) and (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH) )
    RFLAG.ZF ← 1;
    RAX ← SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;

Fi;

(* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
SGX INSTRUCTION REFERENCES

IF ( DS:RCX.ATTRIBUTES & TMP.SIG.ATTRIBEMASK ≠ (TMP.SIG.ATTRIBUTE & TMP.SIG.ATTRIBEMASK) )
  RFLAG.ZF ← 1;
  RAX ← SGX_INVALID_ATTRIBUTE;
  GOTO EXIT;
FI;

(* Verify SIGSTRUCT.MISCSELECT requirements are met *)
IF ( DS:RCX.MISCSELECT & TMP.SIG.MISC_MASK ≠ (TMP.SIG.MISCSELECT & TMP.SIG.MISC_MASK) )
  THEN
    RFLAGS.ZF ← 1;
    RAX ← SGX_INVALID_ATTRIBUTE;
    GOTO EXIT
  FI;

(* if EINITTOKEN.VALID[0] is 0, verify the enclave is signed by an authorized key *)
IF (TMP_TOKEN.VALID[0] = 0)
  IF (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH)
    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 ← EINITTOKEN_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 ← IA32_SGXLEPUBKEYHASH;
TMP_KEYDEPENDENCIES.KEYID ← TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.SEO1_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← TMP_TOKEN.CPUSVN;
TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_TOKEN.MASKEDMISCSELECTLE;
TMP_KEYDEPENDENCIES.MISCMASK ← 0;
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]))
   IFLAG.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)
   IFLAG.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)
   IFLAG.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*)
   IFLAG.ZF ← 0;
   RAX ← 0;
EXIT;
RFLAGS.CF,PF,AF,OF,SF ← 0;
Flags Affected
ZF is cleared if successful, otherwise ZF is set and RAX contains the error code. CF, PF, AF, OF, SF are cleared.

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

64-Bit Mode Exceptions
#GP(0) If a memory operand is not properly aligned.
If another instruction is modifying the SECS.
If the enclave is already initialized.
If the SECS.MRENCLAVE is in use.
#PF(error code) If a page fault occurs in accessing memory operands.
If RCX does not resolve in an EPC page.
If the memory address is not a valid, uninitialized SECS.
ELDB/ELDU—Load an EPC page and Marked its State

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 07H ENCLS[ELDB]</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function loads, verifies an EPC page and marks the page as blocked.</td>
</tr>
<tr>
<td>EAX = 08H ENCLS[ELDU]</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function loads, verifies an EPC page and marks the page as unblocked.</td>
</tr>
</tbody>
</table>

Instruction Operand Encoding

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
<th>RDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>(In)</td>
<td>Address of the PAGEINFO (In)</td>
<td>Address of the EPC page (In)</td>
<td>Address of the version-array slot (In)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>PAGEINFO</th>
<th>PAGEINFO.SRCPGE</th>
<th>PAGEINFO.PCMD</th>
<th>PAGEINFO.SECS</th>
<th>EPCPAGE</th>
<th>Version-Array Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-enclave read access</td>
<td>Non-enclave read access</td>
<td>Non-enclave read access</td>
<td>Enclave read/write access</td>
<td>Read/Write access permitted by Enclave</td>
<td>Read/Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

The error codes are:

Table 41-25. ELDB/ELDU Return Value in RAX

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>ELDB/ELDU successful</td>
</tr>
<tr>
<td>SGX_MAC_COMPARE_FAIL</td>
<td>If the MAC check fails</td>
</tr>
</tbody>
</table>

Concurrence Restrictions

Table 41-26. Concurrence Restrictions of ELDB/ELDU with Intel® SGX Instructions - 1 of 2
Temp Variables in ELDB/ELDU Operational Flow

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

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

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

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

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

TMP_SRCPGE ← DS:RBX.SRCPGE;
TMP_SECS ← DS:RBX.SECS;
TMP_PCMD ← DS: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 Intel 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)
  THEN #PF(DS:RCX); FI;

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

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

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

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

(* Verify various attributes of SECS parameter *)
IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) )
  THEN
    IF (DS:TMP_SECS is not 4KByte aligned)
      THEN #GP(0) FI;
    IF (DS:TMP_SECS does not resolve within an EPC)
      THEN #PF(DS:TMP_SECS) FI;
    IF (Other instructions modifying SECS)
      THEN #GP(0) FI;
    IF ( (EPCM(DS:TMP_SECS).VALID = 0) or (EPCM(DS:TMP_SECS).PT ≠ PT_SECS) )
      THEN #PF(DS:TMP_SECS) FI;
    ELSEIF ( (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;
  FI;

IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) )
  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
SGX INSTRUCTION REFERENCES

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.SECINFO.FLAGS.RWX;
EPCM(DS:RCX).PENDING ← TMP_HEADER.SECINFO.FLAGS.PENDING;
EPCM(DS:RCX).MODIFIED ← TMP_HEADER.SECINFO.FLAGS.MODIFIED;
EPCM(DS:RCX).PR ← TMP_HEADER.SECINFO.FLAGS.PR;
EPCM(DS:RCX).ENCLAVEADDRESS ← TMP_HEADER.LINADDR;

IF ( (EAX = 07H) and (TMP_HEADER.SECINFO.FLAGS.PT is NOT PT_SECS or PT_VA))
  THEN
    EPCM(DS:RCX).BLOCKED ← 1;
  ELSE
    EPCM(DS:RCX).BLOCKED ← 0;
FI;

EPCM(DS:RCX).VALID ← 1;
RAX ← 0;
RFLAGS.ZF ← 0;
ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF ← 0;

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

Protected Mode Exceptions
#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If the instruction’s EPC resource is in use by others.
If the instruction fails to verify MAC.
If the version-array slot is in use.
If the parameters fail consistency checks.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand expected to be in EPC does not resolve to an EPC page.
If one of the EPC memory operands has incorrect page type.
If the destination EPC page is already valid.

64-Bit Mode Exceptions
#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If the instruction’s EPC resource is in use by others.
If the instruction fails to verify MAC.
If the version-array slot is in use.
If the parameters fail consistency checks.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand expected to be in EPC does not resolve to an EPC page.
If one of the EPC memory operands has incorrect page type.
If the destination EPC page is already valid.
EMODPR—Restrict the Permissions of an EPC Page

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 0EH</td>
<td>IR</td>
<td>V/V</td>
<td>SGX2</td>
<td>This leaf function restricts the access rights associated with an EPC page in an initialized enclave.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EMODPR (In)</td>
<td>Return Error Code (Out)</td>
<td>Address of a SECINFO (In)</td>
</tr>
</tbody>
</table>

**Description**

This leaf function restricts the access rights associated with an EPC page in an initialized enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not restrict the page permissions will have no effect. This instruction can only be executed when current privilege level is 0.

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

**EMODPR Memory Parameter Semantics**

<table>
<thead>
<tr>
<th>SECINFO</th>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read/Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

The instruction faults if any of the following:

**EMODPR Faulting Conditions**

- The operands are not properly aligned. If unsupported security attributes are set.
- The Enclave is not initialized. SECS is locked by another thread.
- The EPC page is locked by another thread. RCX does not contain an effective address of an EPC page in the running enclave.
- The EPC page is not valid.

The error codes are:

**Table 41-28. EMODPR Return Value in RAX**

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EMODPR successful</td>
</tr>
<tr>
<td>SGX_PAGE_NOT_MODIFIABLE</td>
<td>The EPC page cannot be modified because it is in the PENDING or MODIFIED state</td>
</tr>
<tr>
<td>SGX_LOCKFAIL</td>
<td>Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB</td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

**Table 41-29. Concurrency Restrictions of EMODPR with Other Intel® SGX Operations 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBG/RD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EP A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>Targ SECS</td>
<td>SSA SECS</td>
<td>Targ SECS</td>
<td>Param SECS</td>
<td>SECS Targ VA SECS VA</td>
</tr>
<tr>
<td>EMODPR</td>
<td>Targ</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Table 41-30. Concurrency Restrictions of EMODPR with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EW</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS NFO</td>
</tr>
<tr>
<td>EMODPR</td>
<td>Targ</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Operation

Temp Variables in EMODPR Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_SECS</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Physical address of SECS to which EPC operand belongs.</td>
</tr>
<tr>
<td>SCRATCH_SECINFO</td>
<td>SECINFO</td>
<td>512</td>
<td>Scratch storage for holding the contents of DS:RBX.</td>
</tr>
</tbody>
</table>

IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

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

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

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF ((SCRATCH_SECINFO reserved fields are not zero ) or
    (SCRATCH_SECINFO.FLAGS.W is not 0) )
    THEN #GP(0); FI;

(* Check concurrency with SGX1 or SGX2 instructions on the EPC page *)
IF (SGX1 or other SGX2 instructions accessing EPC page)
    THEN #GP(0); FI;

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

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN
        RFLAGS ← 1;
        RAX ← SGX_LOCKFAIL;
        GOTO DONE;
    FI;

IF ( (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS ← 1;
        RAX ← SGX_PAGE_NOT_MODIFIABLE;
        GOTO DONE;
SGX INSTRUCTION REFERENCES

Fl;

IF (EPCM(DS:RCX).PT is not PT_REG)
    THEN #PF(DS:RCX); Fl;

TMP_SECS ← GET_SECS_ADDRESS

IF (TMP_SECS_ATTRIBUTES.INIT = 0)
    THEN #GP(0); Fl;

(* Set the PR bit to indicate that permission restriction is in progress *)
EPCM(DS:RCX).PR ← 1;

(* Update EPCM permissions *)

RFLAGS.ZF ← 0;
RAX ← 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;

Flags Affected
Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

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

64-Bit Mode Exceptions
#GP(0) If a memory operand is non-canonical form.
    If a memory operand is not properly aligned.
    If a memory operand is locked.
#PF(error code) If a page fault occurs in accessing memory operands.
    If a memory operand is not an EPC page.
EMODT—Change the Type of an EPC Page

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode</th>
<th>CPUID Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 0FH</td>
<td>IR</td>
<td>V/V</td>
<td>SGX2</td>
<td>This leaf function changes the type of an existing EPC page.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EMODT (In)</td>
<td>Return Error Code (Out)</td>
<td>Address of a SECINFO (In)</td>
</tr>
</tbody>
</table>

**Description**

This leaf function modifies the type of an EPC page. The security attributes are configured to prevent access to the EPC page at its new type until a corresponding invocation of the EACCEPT leaf confirms the modification. This instruction can only be executed when current privilege level is 0.

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

**EMODT Memory Parameter Semantics**

<table>
<thead>
<tr>
<th>SECINFO</th>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read/Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

The instruction faults if any of the following:

**EMODT Faulting Conditions**

- The operands are not properly aligned.
- If unsupported security attributes are set.
- The Enclave is not initialized.
- SECS is locked by another thread.
- The EPC page is locked by another thread.
- RCX does not contain an effective address of an EPC page in the running enclave.
- The EPC page is not valid.

The error codes are:

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EMODT successful</td>
</tr>
<tr>
<td>SGX_PAGE_NOT_MODIFIABLE</td>
<td>The EPC page cannot be modified because it is in the PENDING or MODIFIED state</td>
</tr>
<tr>
<td>SGX_LOCKFAIL</td>
<td>Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODPR, or EWB</td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EP A</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMODT</td>
<td>Targ</td>
<td>Y</td>
<td>Y</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>
Table 41-33. Concurrency Restrictions of EMODT with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODEP</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
</tr>
<tr>
<td>EMODT</td>
<td>Targ</td>
<td>C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Operation

Temp Variables in EMODT Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_SECS</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Physical address of SECS to which EPC operand belongs.</td>
</tr>
<tr>
<td>SCRATCH_SECINFO</td>
<td>SECINFO</td>
<td>512</td>
<td>Scratch storage for holding the contents of DS:RBX.</td>
</tr>
</tbody>
</table>

IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); Fi;

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

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

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or
    (SCRATCH_SECINFO.FLAGS.PT is PT_TCS or SCRATCH_SECINFO.FLAGS.PT is PT_TRIM) )
    THEN #GP(0); Fi;

(* Check concurrency with SGX1 instructions on the EPC page *)
IF (other SGX1 instructions accessing EPC page)
    THEN #GP(0); Fi;

IF (EPCM(DS:RCX).VALID is 0 or
    (EPCM(DS:RCX).PT is PT_REG or EPCM(DS:RCX).PT is PT_TCS))
    THEN #PF(DS:RCX); Fi;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN #GP(0); Fi;

(* Check for mis-configured SECINFO flags*)
IF ( (EPCM(DS:RCX).R = 0) and (SCRATCH_SECINFO.FLAGS.R = 0) and (SCRATCH_SECINFO.FLAGS.W ≠ 0) )
    THEN
        RFLAGS ← 1;
        RAX ← SGX_LOCKFAIL;
        GOTO DONE;
    Fi;

41-50  Vol. 3D
IF ( (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
THEN
   RFLAGS ← 1;
   RAX ← SGX_PAGE_NOT_MODIFIABLE;
   GOTO DONE;
FI;

TMP_SECS ← GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
THEN #GP(0); FI;

(* Check concurrency with ETRACK *)
IF (ETRACK executed concurrently)
THEN #GP(0); FI;

(* Update EPCM fields *)
EPCM(DS:RCX).PR ← 0;
EPCM(DS:RCX).MODIFIED ← 1;
EPCM(DS:RCX).R ← 0;
EPCM(DS:RCX).W ← 0;
EPCM(DS:RCX).X ← 0;
EPCM(DS:RCX).PT ← SCRATCH_SECINFO.FLAGS.PT;

RFLAGS.ZF ← 0;
RAX ← 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;

Flags Affected
Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

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

64-Bit Mode Exceptions
#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If a memory operand is locked.
#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
**EPA—Add Version Array**

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 0AH ENCLS[EPA]</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function adds a Version Array to the EPC.</td>
</tr>
</tbody>
</table>

**InstructionOperand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EPA (In)</td>
<td>PT_VA (In, Constant)</td>
<td>Effective address of the EPC page (In)</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

Table 41-34. Concurrency Restrictions of EPA with Other Intel® SGX Operations 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>EPA</td>
<td>VA</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 41-35. Concurrency Restrictions of EPA with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EwB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECI NFO</td>
</tr>
<tr>
<td>EPA</td>
<td>VA</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Operation**

If (RBX ≠ PT_VA or DS:RCX is not 4KByte Aligned)

THEN #GP(0); Fi;

If (DS:RCX does not resolve within an EPC)

THEN #PF(DS:RCX); Fi;

(*) Check concurrency with other Intel SGX instructions (*)

If (Other Intel SGX instructions accessing the page)

THEN #GP(0); Fi;
(* Check EPC page must be empty *)
IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Clears EPC page *)
DS:RCX[32767:0] ← 0;

EPCM(DS:RCX).PT ← PT_VA;
EPCM(DS:RCX).ENCLAVEADDRESS ← 0;
EPCM(DS:RCX).BLOCKED ← 0;
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).PR ← 0;
EPCM(DS:RCX).RWX ← 0;
EPCM(DS:RCX).VALID ← 1;

**Flags Affected**
None

**Protected Mode Exceptions**

#GP(0) If a memory operand effective address is outside the DS segment limit.  
If a memory operand is not properly aligned.  
If another Intel SGX instruction is accessing the EPC page.  
If RBX is not set to PT_VA.

#PF(error code) If a page fault occurs in accessing memory operands.  
If a memory operand is not an EPC page.  
If the EPC page is valid.

**64-Bit Mode Exceptions**

#GP(0) If a memory operand is non-canonical form.  
If a memory operand is not properly aligned.  
If another Intel SGX instruction is accessing the EPC page.  
If RBX is not set to PT_VA.

#PF(error code) If a page fault occurs in accessing memory operands.  
If a memory operand is not an EPC page.  
If the EPC page is valid.
EREMOVE—Remove a page from the EPC

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32</th>
<th>CPUID Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 03H</td>
<td>IR</td>
<td>V/V</td>
<td>V/V</td>
<td>SGX1</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EREMOVE (In)</td>
<td>Effective address of the EPC page (In)</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>EPCPAGE</th>
</tr>
</thead>
</table>

Write access permitted by Enclave

The instruction faults if any of the following:

**EREMOVE Faulting Conditions**

<table>
<thead>
<tr>
<th>The memory operand is not properly aligned.</th>
<th>The memory operand does not resolve in an EPC page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refers to an invalid SECS.</td>
<td>Refers to an EPC page that is locked by another thread.</td>
</tr>
<tr>
<td>Another Intel SGX instruction is accessing the EPC page.</td>
<td>RCX does not contain an effective address of an EPC page.</td>
</tr>
<tr>
<td>the EPC page refers to an SECS with associations.</td>
<td></td>
</tr>
</tbody>
</table>

The error codes are:

**Table 41-36. EEREMOVE Return Value in RAX**

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EEREMOVE successful</td>
</tr>
<tr>
<td>SGX_CHILD_PRESENT</td>
<td>If the SECS still have enclave pages loaded into EPC</td>
</tr>
<tr>
<td>SGX_ENCLAVE_ACT</td>
<td>If there are still logical processors executing inside the enclave</td>
</tr>
</tbody>
</table>
Concurrency Restrictions

### Table 41-37. Concurrency Restrictions of EREMOVE with Other Intel® SGX Operations 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>EREMOVE</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SECS</td>
<td>C</td>
<td>C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Table 41-38. Concurrency Restrictions of EREMOVE with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EwB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SecI/Info</td>
</tr>
<tr>
<td>EREMOVE</td>
<td>Targ</td>
<td>N</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>C</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>C</td>
</tr>
</tbody>
</table>

### Temp Variables in EREMOVE Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_SECS</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Effective address of the SECS destination page.</td>
</tr>
</tbody>
</table>

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

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

TMP_SECS ← Get_SECS_ADDRESS();

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

(* if DS:RCX is already unused, nothing to do*)
IF ((EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PT = PT_TRIM AND EPCM(DS:RCX).MODIFIED = 0))
THEN GOTO DONE;
Fl;

IF (EPCM(DS:RCX).PT = PT_VA)
THEN
EPCM(DS:RCX).VALID ← 0;
GOTO DONE;
Fl;

IF (EPCM(DS:RCX).PT = PT_SECS)
THEN
IF (DS:RCX has an EPC page associated with it)
THEN
RFLAGS.ZF ← 1;
SGX INSTRUCTION REFERENCES

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,AOF,SF ← 0;

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

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

64-Bit Mode Exceptions
#GP(0) If the memory operand is non-canonical form.
    If a memory operand is not properly aligned.
    If another Intel SGX instruction is accessing the page.
#PF(error code) If a page fault occurs in accessing memory operands.
    If the memory operand is not an EPC page.
ETRACK—Activates EBLOCK Checks

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 0CH</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function activates EBLOCK checks.</td>
</tr>
<tr>
<td>ENCLS[ETRACK]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write access permitted by Enclave</td>
</tr>
</tbody>
</table>

The error codes are:

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>ETRACK successful</td>
</tr>
<tr>
<td>SGX_PREV_TRK_INCMPL</td>
<td>All processors did not complete the previous shoot-down sequence</td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

**Table 41-40. Concurrency Restrictions of ETRACK with Other Intel® SGX Operations 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEODULE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMOPDE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
</tr>
<tr>
<td>ETRACK</td>
<td>SECS</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

**Table 41-41. Concurrency Restrictions of ETRACK with Other Intel® SGX Operations 2 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMOPDE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
</tr>
<tr>
<td>ETRACK</td>
<td>SECS</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Operation**

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

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

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

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

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

(* All processors must have completed the previous tracking cycle*)
IF ( (DS:RCX).TRACKING ≠ 0 )
    THEN
        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 another thread is concurrently using the tracking facility on this SECS.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.

64-Bit Mode Exceptions
#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If the specified EPC resource is in use.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.

<table>
<thead>
<tr>
<th>Table 41-42. ETRACK Return Value in RAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Code (see Table 41-3)</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
<tr>
<td>No Error</td>
</tr>
<tr>
<td>SGX_PREV_TRK_INCMPL</td>
</tr>
</tbody>
</table>
**EWB—Invalidate an EPC Page and Write out to Main Memory**

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 0BH ENCLS[EWB]</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function invalidates an EPC page and writes it out to main memory.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
<th>RDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EWB (In)</td>
<td>Error code (Out)</td>
<td>Address of an PAGEINFO (In)</td>
<td>Address of the EPC page (In)</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>PAGEINFO</th>
<th>PAGEINFO.SRCPGE</th>
<th>PAGEINFO.PCMD</th>
<th>EPCPAGE</th>
<th>VASLOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-EPC R/W access</td>
<td>Non-EPC R/W access</td>
<td>Non-EPC R/W access</td>
<td>EPC R/W access</td>
<td>EPC R/W access</td>
</tr>
</tbody>
</table>

The error codes are:

**Table 41-43. EWB Return Value in RAX**

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EWB successful</td>
</tr>
<tr>
<td>SGX_PAGE_NOT_BLOCKED</td>
<td>If page is not marked as blocked</td>
</tr>
<tr>
<td>SGX_NOT_TRACKED</td>
<td>If EWB is racing with ETRACK instruction</td>
</tr>
<tr>
<td>SGX_VA_SLOT_OCCUPIED</td>
<td>Version array slot contained valid entry</td>
</tr>
<tr>
<td>SGX_CHILD_PRESENT</td>
<td>Child page present while attempting to page out enclave</td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

**Table 41-44. Concurrency Restrictions of EWB with Intel* SGX Instructions - 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGRD/WR</th>
<th>CENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELOB/ELODU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EWB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VA</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

**Table 41-45. Concurrency Restrictions of EWB with Intel* SGX Instructions - 2 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETРАRK</th>
<th>EwB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>SECS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SGX INSTRUCTION REFERENCES

Table 41-45. Concurrency Restrictions of EWB with Intel® SGX Instructions - 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ENTRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EWB</td>
<td>N</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>VA</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Temp Variables in EWB Operational Flow**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_SRCPGE</td>
<td>Memory page</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>TMP_PCMD</td>
<td>PCMD</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>TMP_SECS</td>
<td>SECS</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>TMP_BPEPOCH</td>
<td>UINT64</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>TMP_BPREFCOUNT</td>
<td>UINT64</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>TMP_HEADER</td>
<td>MAC Header</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>TMP_PCMD_ENCLAVEID</td>
<td>UINT64</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>TMP_VER</td>
<td>UINT64</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>TMP_PK</td>
<td>UINT128</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

IF (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned)
THEN #GP(0); Fi;

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

IF (DS:RDX is not 8Byte Aligned)
THEN #GP(0); Fi;

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

(* EPCPAGE and VASLOT should not resolve to the same EPC page*)
IF (DS:RCX and DS:RDX resolve to the same EPC page)
THEN #GP(0); Fi;

TMP_SRCPGE ← DS:RBX.SRCPGE;
(* Note PAGEINFO.PCMD is overlaid on top of PAGEINFO.SECINFO *)
TMP_PCMD ← DS:RBX.PCMD;

IF (DS:RBX.LINADDR ≠ 0) OR (DS:RBX.SECS ≠ 0)
THEN #GP(0); Fi;

IF (DS:TMP_PCMD is not 128Byte Aligned) or (DSTMP_SRCPGE is not 4KByte Aligned) )
THEN #GP(0); Fi;

(* Check for concurrent Intel SGX instruction access to the page *)
IF (Other Intel SGX instruction is accessing page)

41-60 Vol. 3D
THEN #GP(0); Fl;

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

(* Verify that EPCPAGE and VASLOT page are valid EPC pages and DS:RDX is VA *)
IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); Fl;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~FFFH).PT is not PT_VA) )
    THEN #PF(DS:RDX); Fl;

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

Fl:
    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;

(* Zero out TMP_HEADER*)
    TMP_HEADER[ sizeof(TMP_HEADER)-1 : 0] ← 0;
    TMP_HEADER.LINADDR ← EPCM(DS:RCX).ENCLAVEADDRESS;
    TMP_HEADER.SECINFO.FLAGS.PT ← EPCM(DS:RCX).PT;
    TMP_HEADER.SECINFO.FLAGS.RwX ← EPCM(DS:RCX).RwX;
    TMP_HEADER.SECINFO.FLAGS.PENDING ← EPCM(DS:RCX).PENDING;
    TMP_HEADER.SECINFO.FLAGS.MODIFIED ← EPCM(DS:RCX).MODIFIED;
    TMP_HEADER.SECINFO.FLAGS.PR ← EPCM(DS:RCX).PR;

(* Encrypt the page, DS:RCX could be encrypted in place. AES-GCM produces 2 values, (ciphertext, MAC). *)
(* AES-GCM input parameters: key, GCM Counter, MAC_HDR, MAC_HDR_SIZE, SRC, SRC_SIZE)*
    {DS:TMP_SRCPGE, DS:TMP_PCMD.MAC} ← AES_GCM_ENC(CR_BASE_PK), (TMP_VER << 32),
    (DS:TMP_PCMD.SECINFO.FLAGS.PT ← EPCM(DS:RCX).PT);
    DS:TMP_PCMD.SECINFO.FLAGS.RwX ← EPCM(DS:RCX).RwX;
    DS:TMP_PCMD.SECINFO.FLAGS.PENDING ← EPCM(DS:RCX).PENDING;
    DS:TMP_PCMD.SECINFO.FLAGS.MODIFIED ← EPCM(DS:RCX).MODIFIED;
    DS:TMP_PCMD.SECINFO.FLAGS.PR ← EPCM(DS:RCX).PR;
    DS:TMP_PCMD.RESERVED ← 0;
    DS:TMP_PCMD.ENCLAVEID ← TMP_PCMD_ENCLAVEID;
    DS:RBX.LINADDR ← EPCM(DS:RCX).ENCLAVEADDRESS;

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

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

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

Flags Affected
ZF is set if page is not blocked, not tracked, or a child is present. Otherwise cleared.
CF is set if VA slot is previously occupied, Otherwise cleared.
**Protected Mode Exceptions**

#GP(0)  If a memory operand effective address is outside the DS segment limit.
         If a memory operand is not properly aligned.
         If the EPC page and VASLOT resolve to the same EPC page.
         If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
         If the tracking resource is in use.
         If the EPC page or the version array page is invalid.
         If the parameters fail consistency checks.

#PF(error code)  If a page fault occurs in accessing memory operands.
                If a memory operand is not an EPC page.
                If one of the EPC memory operands has incorrect page type.

**64-Bit Mode Exceptions**

#GP(0)  If a memory operand is non-canonical form.
        If a memory operand is not properly aligned.
        If the EPC page and VASLOT resolve to the same EPC page.
        If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
        If the tracking resource is in use.
        If the EPC page or the version array page is invalid.
        If the parameters fail consistency checks.

#PF(error code)  If a page fault occurs in accessing memory operands.
                 If a memory operand is not an EPC page.
                 If one of the EPC memory operands has incorrect page type.
41.4 INTEL® SGX USER LEAF FUNCTION REFERENCE

41.4.1 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.
**EACCEPT—Accept Changes to an EPC Page**

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 05H</td>
<td>IR</td>
<td>V/V</td>
<td>SGX2</td>
<td>This leaf function accepts changes made by system software to an EPC page in the running enclave.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EACCEPT (In)</td>
<td>Return Error Code (Out)</td>
<td>Address of a SECINFO (In)</td>
</tr>
<tr>
<td></td>
<td>RBX</td>
<td>Address of the destination EPC page (In)</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

This leaf function accepts changes to a page in the running enclave by verifying that the security attributes specified in the SECINFO match the security attributes of the page in the EPCM. This instruction leaf can only be executed when inside the enclave.

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

**EACCEPT Memory Parameter Semantics**

<table>
<thead>
<tr>
<th>SECINFO</th>
<th>EPCPAGE (Destination)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read access permitted by Enclave</td>
</tr>
</tbody>
</table>

The instruction faults if any of the following:

**EACCEPT Faulting Conditions**

- The operands are not properly aligned.
- The SECINFO page is locked by another thread.
- The EPC page is not valid.
- SECINFO contains an invalid request.

- If security attributes of the SECINFO page make the page inaccessible.
- RBX does not contain an effective address in an EPC page in the running enclave.
- RCX does not contain an effective address of an EPC page in the running enclave.
- Page type is PT_REG and MODIFIED bit is 0.
- Page type is PT_TCS or PT_TRIM and PENDING bit is 0 and MODIFIED bit is 1.

The error codes are:

**Table 41-46. EACCEPT Return Value in RAX**

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EACCEPT successful</td>
</tr>
<tr>
<td>SGX_PAGE_ATTRIBUTES_MISMATCH</td>
<td>The attributes of the target EPC page do not match the expected values</td>
</tr>
<tr>
<td>SGX_NOT_TRACKED</td>
<td>The OS did not complete an ETRACK on the target page</td>
</tr>
</tbody>
</table>

**Concurrency Restrictions**

**Table 41-47. Concurrency Restrictions of EACCEPT with Intel® SGX Instructions - 1 of 2**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGORD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td></td>
</tr>
<tr>
<td>Targ</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>Targ</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>Targ</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td></td>
</tr>
</tbody>
</table>
## Temp Variables in EACCEPT Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_SECS</td>
<td>Effective</td>
<td>32/64</td>
<td>Physical address of SECS to which EPC operands belongs.</td>
</tr>
<tr>
<td>SCRATCH_SECINFO</td>
<td>SECINFO</td>
<td>512</td>
<td>Scratch storage for holding the contents of DS:RBX.</td>
</tr>
</tbody>
</table>

### IF (DS:RBX is not 64Byte Aligned)

THEN #GP(0); Fl;

IF (DS:RBX is not within CR_ELRANGE)

THEN #GP(0); Fl;

IF (DS:RBX does not resolve within an EPC)

THEN #PF(DS:RBX); Fl;

IF ( (EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or (EPCM(DS:RBX &~FFFH).PT # PT_REG) or (EPCM(DS:RBX &~FFFH).ENCLAVÈSECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ (DS:RBX & FFFH)) )

THEN #PF(DS:RBX); Fl;

(* Copy 64 bytes of contents *)

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)

IF (SCRATCH_SECINFO reserved fields are not zero )

THEN #GP(0); Fl;

IF (DS:RCX is not 4KByte Aligned)

THEN #GP(0); Fl;

IF (DS:RCX is not within CR_ELRANGE)

THEN #GP(0); Fl;
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check that the combination of requested PT, PENDING and MODIFIED is legal *)
IF (NOT ( (SCRATCH_SECINFO.FLAGS.PT is PT_REG) and (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) or
    (SCRATCH_SECINFO.FLAGS.PT is PT_TCS or PT_TRIM) and (SCRATCH_SECINFO.FLAGS.PENDING is 0) and
    (SCRATCH_SECINFO.FLAGS.MODIFIED is 1)) )
THEN #GP(0); FI;

(* Check security attributes of the destination EPC page *)
If ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).BLOCKED is not 0) or
    (EPCM(DS:RCX).PT is not PT_REG) and (EPCM(DS:RCX).PT is not PT_TCS) and (EPCM(DS:RCX).PT is not PT_TRIM) or
    (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS)
THEN #PF(DS:RCX); FI;

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

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
THEN #PF(DS:RCX); FI;

(* Verify that accept request matches current EPC page settings *)
IF ( (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX) or (EPCM(DS:RCX).PENDING ≠ SCRATCH_SECINFO.FLAGS.PENDING) or
    (EPCM(DS:RCX).MODIFIED ≠ SCRATCH_SECINFO.FLAGS.MODIFIED) or (EPCM(DS:RCX).R ≠ SCRATCH_SECINFO.FLAGS.R) or
    (EPCM(DS:RCX).W ≠ SCRATCH_SECINFO.FLAGS.W) or (EPCM(DS:RCX).X ≠ SCRATCH_SECINFO.FLAGS.X) or
    (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) )
THEN
    RFLAGS ← 1;
    RAX ← SGX_PAGE_ATTRIBUTES_MISMATCH;
    GOTO DONE;
FI;

(* Check that all required threads have left enclave *)
IF (Tracking not correct)
    THEN
        RFLAGS.ZF ← 1;
        RAX ← SGX_NOT_TRACKED;
        GOTO DONE;
    FI;

(* Get pointer to the SECS to which the EPC page belongs *)
TMP_SECS = << Obtain physical address of SECS through EPCM(DS:RCX)>>

(* For TCS pages, perform additional checks *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
    FI;

(* Check that TCS.FLAGS.DBGOPTIN, TCS stack, and TCS status are correctly initialized *)
IF ( ((DS:RCX).FLAGS.DBGOPTIN is not 0) or ((DS:RCX).CSSA ≥ (DS:RCX).NSSA) or ((DS:RCX).AEP is not 0) or ((DS:RCX).STATE is not 0)
    THEN #GP(0); FI;
SGX INSTRUCTION REFERENCES

(* Check consistency of FS & GS Limit *)
IF ( (TMP_SECS.ATTRIBUTES.MODE64BIT is 0) and ((DS:RCX.FSLIMIT & FFFH ≠ FFFH) or (DS:RCX.GSLIMIT & FFFH ≠ FFFH)) )
  THEN #GP(0); FI;

(* Clear PENDING/MODIFIED flags to mark accept operation complete *)
EPCM(DS:RCX).PENDING ← 0;
EPCM(DS:RCX).MODIFIED ← 0;
EPCM(DS:RCX).PR ← 0;

(* Clear EAX and ZF to indicate successful completion *)
RFLAGS.ZF ← 0;
RAX ← 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;

Flags Affected
Sets ZF if page cannot be accepted, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions
#GP(0) If a memory operand effective address is outside the DS segment limit.
   If a memory operand is not properly aligned.
   If a memory operand is locked.
#PF(error code) If a page fault occurs in accessing memory operands.
   If a memory operand is not an EPC page.
   If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions
#GP(0) If a memory operand is non-canonical form.
   If a memory operand is not properly aligned.
   If a memory operand is locked.
#PF(error code) If a page fault occurs in accessing memory operands.
   If a memory operand is not an EPC page.
   If EPC page has incorrect page type or security attributes.
EACCEPTCOPY—Initialize a Pending Page

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 07H ENCLU[EACCEPTCOPY]</td>
<td>IR V/V</td>
<td>SGX2</td>
<td>This leaf function initializes a dynamically allocated EPC page from another page in the EPC.</td>
<td></td>
</tr>
</tbody>
</table>

Instruction Operand Encoding

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
<th>RDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR EACCEPTCOPY (In)</td>
<td>Return Error Code (Out)</td>
<td>Address of a SECINFO (In)</td>
<td>Address of the destination EPC page (In)</td>
<td>Address of the source EPC page (In)</td>
</tr>
</tbody>
</table>

Description

This leaf function copies the contents of an existing EPC page into an uninitialized EPC page (created by EAUG). After initialization, the instruction may also modify the access rights associated with the destination EPC page. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX and RDX each contain the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPTCOPY leaf function.

**EACCEPTCOPY Memory Parameter Semantics**

<table>
<thead>
<tr>
<th>SECINFO</th>
<th>EPCPAGE (Destination)</th>
<th>EPCPAGE (Source)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read/Write access permitted by Enclave</td>
<td>Read access permitted by Enclave</td>
</tr>
</tbody>
</table>

The instruction faults if any of the following:

**EACCEPTCOPY Faulting Conditions**

- The operands are not properly aligned.
- The EPC page is locked by another thread.
- The EPC page is not valid.
- SECINFO contains an invalid request.
- If security attributes of the SECINFO page make the page inaccessible.
- If security attributes of the source EPC page make the page inaccessible.
- RBX does not contain an effective address in an EPC page in the running enclave.
- RCX/RDX does not contain an effective address of an EPC page in the running enclave.

The error codes are:

**Table 41-49. EACCEPTCOPY Return Value in RAX**

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>EACCEPTCOPY successful</td>
</tr>
<tr>
<td>SGX_PAGE_ATTRIBUTES_MISMATCH</td>
<td>The attributes of the target EPC page do not match the expected values</td>
</tr>
</tbody>
</table>
Concurrent Restrictions

Table 41-50. Concurrency Restrictions of EACCEPTCOPY with Intel® SGX Instructions - 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGWRD/WR</th>
<th>ENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EIINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>EACCEPTCOPY</td>
<td>Targ</td>
<td>U</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Src</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECINFO</td>
<td>U</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 41-51. Concurrency Restrictions of EACCEPTCOPY with Intel® SGX Instructions - 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODEPE</th>
<th>EMODEPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
</tr>
<tr>
<td>EACCEPTCOPY</td>
<td>Targ</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Src</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECINFO</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Operation

**Temp Variables in EACCEPTCOPY Operational Flow**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCRATCH_SECINFO</td>
<td>SECINFO</td>
<td>512</td>
<td>Scratch storage for holding the contents of DS:RBX.</td>
</tr>
</tbody>
</table>

IF (DS:RBX is not 64Byte Aligned)

THEN #GP(0); Fl;

IF (DS:RCX is not 4KByte Aligned) or (DS:RDX is not 4KByte Aligned)

THEN #GP(0); Fl;

IF (DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) or (DS:RDX is not within CR_ELRANGE)

THEN #GP(0); Fl;

IF (DS:RBX does not resolve within an EPC)

THEN #PF(DS:RBX); Fl;

IF (DS:RCX does not resolve within an EPC)

THEN #PF(DS:RCX); Fl;

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

THEN #PF(DS:RDX); Fl;

IF ((EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or (EPCM(DS:RBX &~FFFH).PT # PT_REG) or (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ DS:RBX))

THEN #PF(DS:RBX); Fl;
(* Copy 64 bytes of contents *)

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or ((SCRATCH_SECINFO.FLAGS.R=0) AND(SCRATCH_SECINFO.FLAGS.W#0 ) or (SCRATCH_SECINFO.FLAGS.PT is not PT_REG ) )
    THEN #GP(0); FI;

(* Check security attributes of the source EPC page *)
IF ( (EPCM(DS:RDX).VALID = 0) or (EPCM(DS:RDX).PENDING ≠ 0) or (EPCM(DS:RDX).MODIFIED ≠ 0) or (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RDX).ENCLAVEADDRESS ≠ DS:RDX))
    THEN #PF(DS:RDX); FI;

(* Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN RFLAGS ← 1;
    RAX ← SGX_PAGE_ATTRIBUTE_MISMATCH;
    GOTO DONE;
    FI;

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

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or (EPCM(DS:RCX).BLOCKED ≠ 1) or (EPCM(DS:RCX).W ≠ 1) or (EPCM(DS:RCX).X ≠ 0) or (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
    THEN #PF(DS:RCX); FI;

(* Copy 4KBbytes form the source to destination EPC page*)
DS:RCX[32767:0] ← DS:RDX[32767:0];

(* Update EPCM permissions *)
EPCM(DS:RCX).PENDING ← 0;

RFLAGS.ZF ← 0;
RAX ← 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF ← 0;

Flags Affected
Sets ZF if page is not modifiable, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions
#GP(0) If a memory operand effective address is outside the DS segment limit.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If EPC page has incorrect page type or security attributes.

**64-Bit Mode Exceptions**

#GP(0) If a memory operand is non-canonical form.
If a memory operand is not properly aligned.
If a memory operand is locked.

#PF(error code) If a page fault occurs in accessing memory operands.
If a memory operand is not an EPC page.
If EPC page has incorrect page type or security attributes.
**EENTER—Enters an Enclave**

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 02H ENCLU[EENTER]</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function is used to enter an enclave.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EENTER (In)</td>
<td>Address of a TCS (In)</td>
<td>Address of IP following EENTER (Out)</td>
</tr>
<tr>
<td></td>
<td>Content of RBX.CSSA (Out)</td>
<td>Address of AEP (In)</td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>TCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enclave access</td>
</tr>
</tbody>
</table>

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

- Address in RBX is not properly aligned.
- TCS pointed to by RBX is not valid or available or locked.
- The SECS is in use.
- Any one of DS, ES, CS, SS is not zero.
- CR4.OSFXSR ≠ 1.

Any TCS.FLAGS's must-be-zero bit is not zero.
Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
Either of TCS-specified FS and GS segment is not a subset of the current DS segment.
If XSAVE available, CR4.OSXS SAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
If CR4.OSXS SAVE = 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.OSXS SAVE == 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 43.1.2):
  - 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 43.2.5).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 43.2.2).
- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.
- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 43.2.3):
SGX INSTRUCTION REFERENCES

— All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
— PEBS is suppressed.
— AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set
— If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 41-52. Concurrency Restrictions of EENTER with Intel® SGX Instructions - 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGREG/WR</th>
<th>EENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EENTER</td>
<td>TCS</td>
<td>N</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>TCS</td>
<td>SSA</td>
<td>Param</td>
<td>Secs</td>
<td>Secs</td>
</tr>
<tr>
<td>SSA</td>
<td>U</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 41-53. Concurrency Restrictions of EENTER with Intel® SGX Instructions - 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRA CK</th>
<th>EWBG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>N</td>
<td>TCS</td>
<td>N</td>
<td>TCS</td>
<td>N</td>
<td>TCS</td>
<td>N</td>
<td>TCS</td>
</tr>
<tr>
<td>SSA</td>
<td>U</td>
<td></td>
<td>U</td>
<td>Y</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Temp Variables in EENTER Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_FSBASE</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Proposed base address for FS segment.</td>
</tr>
<tr>
<td>TMP_GSBASE</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Proposed base address for FS segment.</td>
</tr>
<tr>
<td>TMP_FSLIMIT</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Highest legal address in proposed FS segment.</td>
</tr>
<tr>
<td>TMP_GSLIMIT</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Highest legal address in proposed GS segment.</td>
</tr>
<tr>
<td>TMP_XSIZE</td>
<td>integer</td>
<td>64</td>
<td>Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.</td>
</tr>
<tr>
<td>TMP_SSA_PAGE</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Pointer used to iterate over the SSA pages in the current frame.</td>
</tr>
<tr>
<td>TMP_GPR</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Address of the GPR area within the current SSA frame.</td>
</tr>
</tbody>
</table>

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); Fl;

(* Check that CS, SS, DS, ES.base is 0 *)
IF (TMP_MODE64 = 0)
THEN
  IF (CS.base ≠ 0 or DS.base ≠ 0) #GP(0); Fl;
IF (ES usable and ES.base ≠ 0) #GP(0); Fi;
IF (SS usable and SS.base ≠ 0) #GP(0); Fi;
IF (SS usable and SS.B = 0) #GP(0); Fi;

FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); Fi;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); Fi;

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

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions is operating on TCS)
    THEN #GP(0); Fi;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); Fi;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); Fi;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS # DS:RBX) or (EPCM(DS:RBX).PT # PT_TCS) )
    THEN #PF(DS:RBX); Fi;

IF ( (EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1) )
    THEN #PF(DS:RBX); Fi;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); Fi;

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

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

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    THEN
        TMP_FSBASE ← (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_GSBASE ← (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        (* 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); Fl;
Fl;
(* if GS wrap-around, make sure DS has no holes*)
IF (TMP_GSLIMIT < TMP_GSBASE)
THEN
IF (DS.limit < 4GB) THEN #GP(0); Fl;
ELSE
IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); Fl;
ELSE
TMP_FSBASE ← (DS:RBX).OFBASE + TMP_SECS.BASEADDR;
TMP_GSBASE ← (DS:RBX).OGBASE + TMP_SECS.BASEADDR;
IF ((TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
THEN #GP(0); Fl;
Fl;
(* Ensure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & & FFFFFFFFFFFFFFFFEH) ≠ 0)
THEN #GP(0); Fl;
(* SECS must exist and enclave must have previously been EINITed *)
IF (the enclave is not already initialized)
THEN #GP(0); Fl;
(* make sure the logical processor’s operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
THEN #GP(0); Fl;
IF (CR4.OSFXSR = 0)
THEN #GP(0); Fl;
(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
THEN
IF (TMP_SECS.ATTRIBUTES.XFRM # 03H) THEN #GP(0); Fl;
ELSE
IF ((TMP_SECS.ATTRIBUTES.XFRM & XCR0) ≠ TMP_SECS.ATTRIBUTES.XFRM) THEN #GP(0); Fl;
Fl;
(* Make sure the SSA contains at least one more frame *)
THEN #GP(0); Fl;
(* Compute linear address of SSA frame *)
TMP_XSIZE ← compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);
FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
(* Check page is read/write accessible *)
Check that DS:TMP_SSA_PAGE is read/write accessible;
If a fault occurs, release locks, abort and deliver that fault;
IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
THEN #PF(DS:TMP_SSA_PAGE); Fl;
41-76 Vol. 3D
IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
THEN #PF(DS:TMP_SSA_PAGE); FI;
IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
THEN #PF(DS:TMP_SSA_PAGE); FI;
IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
THEN #PF(DS:TMP_SSA_PAGE); FI;
IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
   (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
   (EPCM(DS:TMP_SECS).R = 0) or (EPCM(DS:TMP_SECS).W = 0) )
THEN #PF(DS:TMP_SSA_PAGE); FI;

CR_XSAVE_PAGE_n ← Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR ← TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE -- sizeof(GPRSGX_AREA);
If a fault occurs; release locks, abort and deliver that fault;

IF (DS:TMP_GPR does not resolve to EPC page)
  THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
  THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
  THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
  THEN #PF(DS:TMP_GPR); FI;
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
  THEN #PF(DS:TMP_GPR); FI;
IF (TMP_MODE64 = 0)
THEN
  IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
FI;

CR_GPR_PA ← Physical_Address (DS: TMP_GPR);

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

(* 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);
SGX INSTRUCTION REFERENCES

(* 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;
        Suppress_any_pending_debug_exceptions;
        Suppress_any_pending_MTF_VM_exit;
    ELSE
        IF RFLAGS.TF = 1
            THEN pend a single-step #DB at the end of EENTER; FI;
        IF the “monitor trap flag” VM-execution control is set
            THEN pend an MTF VM exit at the end of EENTER; FI;
    FI;

Flush_linear_context;
Allow_front_end_to_begin_fetch_at_new_RIP;

Flags Affected
RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions
#GP(0) If DS:RBX is not page aligned.
If the enclave is not initialized.
If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
If the thread is not in the INACTIVE state.
If CS, DS, ES or SS bases are not all zero.
If executed in enclave mode.
If any reserved field in the TCS FLAG is set.
If the target address is not within the CS segment.
If CR4.OSFXSR = 0.
If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

#PF(error code) If a page fault occurs in accessing memory.
If DS:RBX does not point to a valid TCS.
If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions
#GP(0) If DS:RBX is not page aligned.
If the enclave is not initialized.
If the thread is not in the INACTIVE state.
If CS, DS, ES or SS bases are not all zero.
If executed in enclave mode.
If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
If the target address is not canonical.
If CR4.OSFXSR = 0.
If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

#PF(error code) If a page fault occurs in accessing memory operands.
If DS:RBX does not point to a valid TCS.
If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.
EEXIT—Exits an Enclave

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.

Concurrency Restrictions

Table 41-54. Concurrency Restrictions of EEXIT with Intel® SGX Instructions - 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGDR/WR</th>
<th>ENTER/ERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td></td>
<td>Targ</td>
<td>VA</td>
<td>SECS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEXIT</td>
<td>TCS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SSA</td>
<td></td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 41-55. Concurrency Restrictions of EEXIT with Intel® SGX Instructions - 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRAC CK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SRC</td>
<td>VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>EEXIT</td>
<td>TCS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SSA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
**SGX INSTRUCTION REFERENCES**

**Operation**

**Temp Variables in EEXIT Operational Flow**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_RIP</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Saved copy of CRIP for use when creating LBR.</td>
</tr>
</tbody>
</table>

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;
    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_monior_trap_flag;
        UnSuppress_LBR_Generation;
        UnSuppress_performance_monitoring_activity;
        Restore performance monitoring counter AnyThread demotion to MyThread in enclave back to AnyThread
    FI;

IF RFLAGS.TF = 1
    THEN Pend Single-Step #DB at the end of EEXIT;
    FI;
IF the “monitor trap flag” VM-execution control is set THEN pend a MTF VM exit at the end of EEXIT;
FI;

CR_ENCLAVE_MODE <- 0;
CR_TCS_PA.STATE <- INACTIVE;

(* Assure consistent translations *)
Flush_linear_context;

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

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

64-Bit Mode Exceptions
#GP(0) If executed outside an enclave.
    If RBX is not canonical.
#PF(error code) If a page fault occurs in accessing memory operands.
EGETKEY—Retrieves a Cryptographic Key

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 04H</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function retrieves a cryptographic key.</td>
</tr>
<tr>
<td>ENCLU[EGETKEY]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instruction Operand Encoding

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EGETKEY (In)</td>
<td>Address to a KEYREQUEST (In)</td>
<td>Address of the OUTPUTDATA (In)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>KEYREQUEST</th>
<th>OUTPUTDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enclave read access</td>
<td>Enclave write access</td>
</tr>
</tbody>
</table>

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 41-56
- Computes derived key using the derivation data and package specific value
- Outputs the calculated key to the address in RCX

The instruction fails with #GP(0) if the operands are not properly aligned. Successful completion of the instruction will clear RFLAGS.(ZF, CF, AF, OF, SF, PF). The instruction returns an error code if the user tries to request a key based on an invalid CPUSVN or ISVSVN (when the user request is accepted, see the table below), requests a key for which it has not been granted the attribute to request, or requests a key that is not supported by the hardware. These checks may be performed in any order. Thus, an indication by error number of one cause (for example, invalid attribute) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same Enclave. The correctness of software should not rely on the order resulting from the checks documented in this section. In such cases the ZF flag is set and the corresponding error bit (SGX_INVALID_SVN, SGX_INVALID_ATTRIBUTE, SGX_INVALID_KEYNAME) is set in RAX and the data at the address specified by RCX is unmodified.

Requesting Keys

The KEYREQUEST structure (see Section 38.17.1) 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 41-56) 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 41-56 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.
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.

Several keys are access controlled. Access to the Provisioning Key and Provisioning Seal key requires the enclave's ATTRIBUTES.PROVISIONKEY be set. The EINITTOKEN Key requires ATTRIBUTES.EINITTOKENKEY be set and SECS.MRSIGNER equal IA32_SGXLEPUBKEYHASH.

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

\[
\text{HARDCODED_PKCS1_5_PADDING}[15:0] \leftarrow 0100H; \\
\text{HARDCODED_PKCS1_5_PADDING}[2655:16] \leftarrow \text{SignExtend330Byte}(-1); // 330 bytes of 0FFH \\
\text{HARDCODED_PKCS1_5_PADDING}[2815:2656] \leftarrow 2004000501020403650148866009060D30313000H;
\]

The error codes are:

### Table 41-57. EGETKEY Return Value in RAX

<table>
<thead>
<tr>
<th>Error Code (see Table 41-3)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>0</td>
<td>EGETKEY successful</td>
</tr>
<tr>
<td>SGX_INVALID_ATTRIBUTES</td>
<td></td>
<td>The KEYREQUEST contains a KEYNAME for which the enclave is not authorized</td>
</tr>
<tr>
<td>SGX_INVALID_CPUSVN</td>
<td></td>
<td>If KEYREQUEST.CPUSVN is an unsupported platforms CPUSVN value</td>
</tr>
<tr>
<td>SGX_INVALID_ISVSVN</td>
<td></td>
<td>If KEYREQUEST.ISVSVN is greater than the enclave's ISV_SVN</td>
</tr>
<tr>
<td>SGX_INVALID_KEYNAME</td>
<td></td>
<td>If KEYREQUEST.KEYNAME is an unsupported value</td>
</tr>
</tbody>
</table>

### Concurrency Restrictions

#### Table 41-58. Concurrency Restrictions of EGETKEY with Other Intel® SGX Operations 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREATE</th>
<th>EDBGORD/WR</th>
<th>EEENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>Secs</td>
<td>Secs</td>
<td>Secs</td>
<td>Secs</td>
<td>Secs</td>
<td>Secs</td>
<td>Secs</td>
</tr>
<tr>
<td>EGETKEY</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Table 41-59. Concurrency Restrictions of EGETKEY with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>Targ</td>
<td>SRC</td>
<td>Targ</td>
<td>Targ</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>EGETKEY</td>
<td>Param</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>U</td>
<td></td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Operation

Temp Variables in EGETKEY Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (Bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_CURRENTSECS</td>
<td>Address of the SECS for the currently executing enclave.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMP_KEYDEPENDENCIES</td>
<td>Temp space for key derivation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMP_ATTRIBUTES</td>
<td>128</td>
<td>Temp Space for the calculation of the sealable Attributes.</td>
<td></td>
</tr>
<tr>
<td>TMP_OUTPUTKEY</td>
<td>128</td>
<td>Temp Space for the calculation of the key.</td>
<td></td>
</tr>
</tbody>
</table>

(* 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 #PF(DS:RBX); FI;

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

(* Check page parameters for correctness *)
IF ((EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH)) or (EPCM(DS:RBX).R = 0))
    THEN #PF(DS:RBX);
FI;

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

(* Make sure DS:RCX is an EPC address and the EPC page is valid *)
IF ( (DS:RCX does not resolve to an EPC address) or (EPCM(DS:RCX).VALID = 0) )
    THEN #PF(DS:RCX); FI;

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

(* Check page parameters for correctness *)
IF ((EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH)) or (EPCM(DS:RCX).W = 0))
    THEN #PF(DS:RCX);
FI;
(* Verify RESERVED spaces in KEYREQUEST are valid *)
IF ((DS:RBX).RESERVED ≠ 0) or (DS:RBX.KEYPOLICY.RESERVED ≠ 0) )
    THEN #GP(0); FI;

TMP_CURRENTSECS ← CR_ACTIVE_SECS;

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

(* Compute MISCE SELECT fields to be included *)
TMP_MISCE SELECT ← DS:RBX.MISCMA N & TMP_CURRENTSECS.MISCE SELECT

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.ISVPROID ← TMP_CURRENTSECS.ISVPROID;
    TMP_KEYDEPENDENCIES.ISVSVN ← DS:RBX.ISVSVN;
    TMP_KEYDEPENDENCIES.OWNEREP OCH ← 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.S EAL_KEY_FUSES ← CR_SEAL_FUSES;
    TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
    TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
    TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_MISCE SELECT;
    TMP_KEYDEPENDENCIES.MISCMA N ← ~DS:RBX.MISCMA N;
    BREAK;
REPORT_KEY:
// Determine values key is based on
TMP_KEYDEPENDENCIES.KEYNAME ← REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVPRODID ← 0;
TMP_KEYDEPENDENCIES.ISVSVN ← 0;
TMP_KEYDEPENDENCIES.OwnEPOCH ← CSR_SEOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_CURRENTSECS.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
TMP_KEYDEPENDENCIES.MRENCALVE ← TMP_CURRENTSECS.MRENCALVE;
TMP_KEYDEPENDENCIES.MRSIGNER ← 0;
TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
TMP_KEYDEPENDENCIES.SELF_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING ← HARDCODED_PKCS1_5_PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_CURRENTSECS.MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK ← 0;
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.OwnEPOCH ← CSR_SEOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK ← 0;
TMP_KEYDEPENDENCIES.MRENCALVE ← 0;
TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
TMP_KEYDEPENDENCIES.KEYID ← DS:RBX.KEYID;
TMP_KEYDEPENDENCIES.SELF_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT ← TMP_MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK ← 0;
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.ATTRIBUTESMASK;
TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
TMP_KEYDEPENDENCIES.KEYID ← 0;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← 0;
TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCMASK ← DS:RBX.MISCMASK;
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;
SGX INSTRUCTION REFERENCES

(* 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.OWNEREPCH ← 0;
TMP_KEYDEPENDENCIES.ATTRIBUTES ← TMP_ATTRIBUTES;
TMP_KEYDEPENDENCIES.MRENCLAVE ← 0;
TMP_KEYDEPENDENCIES.MRSIGNER ← TMP_CURRENTSECS.MRSIGNER;
TMP_KEYDEPENDENCIES.KEYID ← 0;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES ← CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN ← DS:RBX.CPUSVN;
TMP_KEYDEPENDENCIES.PADDING ← TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCMASK ← ~DS:RBX.MISCMASK;
BREAK;
DEFAULT:
(* The value of KEYNAME is invalid *)
RFLAGS.ZF ← 1;
RAX ← SGXINVALID_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(error 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(error code) If a page fault occurs in accessing memory operands.
EMODPE—Extend an EPC Page Permissions

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 06H ENCLU[EMODPE]</td>
<td>IR</td>
<td>V/V</td>
<td>SGX2</td>
<td>This leaf function extends the access rights of an existing EPC page.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EMODPE (In)</td>
<td>Address of a SECINFO (In)</td>
<td>Address of the destination EPC page (In)</td>
</tr>
</tbody>
</table>

**Description**

This leaf function extends the access rights associated with an existing EPC page in the running enclave. The RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not extend the page permissions will have no effect. This instruction leaf can only be executed when inside the enclave.

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

**EMODPE Memory Parameter Semantics**

<table>
<thead>
<tr>
<th>SECINFO</th>
<th>EPCPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access permitted by Non Enclave</td>
<td>Read access permitted by Enclave</td>
</tr>
</tbody>
</table>

The instruction faults if any of the following:

**EMODPE Faulting Conditions**

- The operands are not properly aligned.
- If security attributes of the SECINFO page make the page inaccessible.
- The EPC page is locked by another thread.
- RBX does not contain an effective address in an EPC page in the running enclave.
- The EPC page is not valid.
- RCX does not contain an effective address of an EPC page in the running enclave.
- SECINFO contains an invalid request.

**Concurrency Restrictions**

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECCRETATE</th>
<th>EDBGRED/WR</th>
<th>CENTER/ERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMODPE</td>
<td>Targ</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>SECINFO</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 41-60. Concurrency Restrictions of EMODPE with Other Intel® SGX Operations 1 of 2**

**Table 41-61. Concurrency Restrictions of EMODPE with Other Intel® SGX Operations 2 of 2**

41-92 Vol. 3D
Operation

Temp Variables in EMODPE Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCRATCH_SECINFO</td>
<td>SECINFO</td>
<td>512</td>
<td>Scratch storage for holding the contents of DS:RBX.</td>
</tr>
</tbody>
</table>

IF (DS:RBX is not 64Byte Aligned)
   THEN #GP(0); Fi;

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

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE))
   THEN #GP(0); Fi;

IF (DS:RBX does not resolve within an EPC)
   THEN #PF(DS:RBX); Fi;

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

IF ((EPCM(DS:RBX).VALID = 0) or (EPCM(DS:RBX).PENDING ≠ 0) or (EPCM(DS:RBX).MODIFIED ≠ 0) or
    (EPCM(DS:RBX).BLOCKED ≠ 0) or (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX))
   THEN #PF(DS:RBX); Fi;

SCRATCH_SECINFO ← DS:RBX;

(* Check for mis-configured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero)
   THEN #GP(0); Fi;

(* Check security attributes of the EPC page *)
IF ((EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS))
   THEN #PF(DS:RCX); Fi;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
   THEN #GP(0); Fi;

(* Re-Check security attributes of the EPC page *)
IF ((EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
   THEN #PF(DS:RCX); Fi;

(* Check for mis-configured SECINFO flags*)
IF ((EPCM(DS:RCX).R = 0) and (SCRATCH_SECINFO.FLAGS.R = 0) and (SCRATCH_SECINFO.FLAGS.W ≠ 0))
   THEN #GP(0); Fi;
SGX INSTRUCTION REFERENCES

(* Update EPCM permissions *)

Flags Affected
None

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

64-Bit Mode Exceptions
#GP(0) If a memory operand is non-canonical form.
       If a memory operand is not properly aligned.
       If a memory operand is locked.
#PF(error code) If a page fault occurs in accessing memory operands.
EREPORT—Create a Cryptographic Report of the Enclave

<table>
<thead>
<tr>
<th>Opcode/Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 00H</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function creates a cryptographic report of the enclave.</td>
</tr>
</tbody>
</table>

Instruction Operand Encoding

<table>
<thead>
<tr>
<th>Op/En</th>
<th>EAX</th>
<th>RBX</th>
<th>RCX</th>
<th>RDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>EREPORT (In)</td>
<td>Address of TARGETINFO (In)</td>
<td>Address of REPORTDATA (In)</td>
<td>Address where the REPORT is written to in an OUTPUTDATA (In)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>TARGETINFO</th>
<th>REPORTDATA</th>
<th>OUTPUTDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access by Enclave</td>
<td>Read access by Enclave</td>
<td>Read/Write access by Enclave</td>
</tr>
</tbody>
</table>

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

The instruction fails if the operands are not properly aligned.
CR_REPORT_KEYID, used to provide key wearout protection, is populated with a statistically unique value on boot of the platform by a trusted entity within the SGX TCB.

The instruction faults if any of the following:
EREPORT Faulting Conditions

<table>
<thead>
<tr>
<th>An effective address not properly aligned.</th>
<th>An memory address does not resolve in an EPC page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>If accessing an invalid EPC page.</td>
<td>If the EPC page is blocked.</td>
</tr>
<tr>
<td>May page fault.</td>
<td></td>
</tr>
</tbody>
</table>

Concurrent Restrictions

Table 41-62. Concurrency Restrictions of EREPORT with Other Intel® SGX Operations 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECREDATE</th>
<th>EDBGRD/WR</th>
<th>EENTER/ERESUME</th>
<th>EEEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>TCS</td>
<td>SSA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>EREPORT</td>
<td>Param</td>
<td>SSA</td>
<td>SECS</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 41-63. Concurrency Restrictions of EREPORT with Other Intel® SGX Operations 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREPORT</th>
<th>ETRACK</th>
<th>EAUG</th>
<th>EMOODEPR</th>
<th>EMOODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>SRC</td>
<td>Sec</td>
<td>Sec</td>
<td>Sec</td>
<td>Sec</td>
</tr>
<tr>
<td>EREPORT</td>
<td>Param</td>
<td>SECS</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
<td>SSA</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Temp Variables in EREPORT Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_ATTRIBUTES</td>
<td>32</td>
<td>32</td>
<td>Physical address of SECS of the enclave to which source operand belongs.</td>
</tr>
<tr>
<td>TMP_CURRENTSECS</td>
<td>32</td>
<td>32</td>
<td>Address of the SECS for the currently executing enclave.</td>
</tr>
<tr>
<td>TMP_KEYDEPENDENCIES</td>
<td></td>
<td></td>
<td>Temp space for key derivation.</td>
</tr>
<tr>
<td>TMP_REPORTKEY</td>
<td>128</td>
<td>128</td>
<td>REPORTKEY generated by the instruction.</td>
</tr>
<tr>
<td>TMP_REPORT</td>
<td>3712</td>
<td>3712</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{TMP\_MODE64} \leftarrow ((\text{IA32\_EFER\_LMA} = 1) \&\& (\text{CS.L} = 1));
\]

(* Address verification for TARGETINFO (RBX) *)

IF ( (DS:RBX is not 128Byte Aligned) or (DS:RBX is not within CR\_ELRANGE) )

THEN #GP(0); Fl;

IF (DS:RBX does not resolve within an EPC)

THEN #PF(DS:RBX); Fl;

IF (EPCM(DS:RBX).VALID = 0)

THEN #PF(DS:RBX); Fl;

IF (EPCM(DS:RBX).BLOCKED = 1))

THEN #PF(DS:RBX); Fl;

(* Check page parameters for correctness *)

IF ( (EPCM(DS:RBX).PT \# PT\_REG) or (EPCM(DS:RBX).ENCLAVESECS \# CR\_ACTIVE\_SECS) or (EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS \# (DS:RBX & \~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
THEN #PF(DS:RBX);
FI;

(* Address verification for REPORTDATA (RCX) *)
IF ( (DS:RCX is not 128Byte Aligned) or (DS:RCX is not within CR_EL RANGE) )
   THEN #GP(0); FI;

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

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

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

(* Check page parameters for correctness *)
IF ((EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
   (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH)) or (EPCM(DS:RCX).R = 0))
   THEN #PF(DS:RCX);
FI;

(* Address verification for OUTPUTDATA (RDX) *)
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RDX is not within CR_EL RANGE) )
   THEN #GP(0); FI;

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

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

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

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

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

(* 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_REPORT[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(error 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(error code) If a page fault occurs in accessing memory operands.
ERSUME—Re-Enters an Enclave

### Opcode/Instruction

<table>
<thead>
<tr>
<th>Opcode/ Instruction</th>
<th>Op/En</th>
<th>64/32 bit Mode Support</th>
<th>CPUID Feature Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 03H ENCLU[ERSUME]</td>
<td>IR</td>
<td>V/V</td>
<td>SGX1</td>
<td>This leaf function is used to re-enter an enclave after an interrupt.</td>
</tr>
</tbody>
</table>

### Instruction Operand Encoding

<table>
<thead>
<tr>
<th>Op/En</th>
<th>RAX</th>
<th>RBX</th>
<th>RCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>ERESUME (In)</td>
<td>Address of a TCS (In)</td>
<td>Address of AEP (In)</td>
</tr>
</tbody>
</table>

### Description

The ENCLU[ERSUME] 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

<table>
<thead>
<tr>
<th>TCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enclave read/write access</td>
</tr>
</tbody>
</table>

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 ≠ 3.
- CR4.OSFXSR ≠ 1. If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
- Offsets 520-535 of the XSAVE area not 0. The bit vector stored at offset 512 of the XSAVE area must be a subset of SECS.ATTRIBUTES.XFRM.
- The SSA frame is not valid or in use.

The following operations are performed by ERESUME:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or an asynchronous exit due to any Interrupt event.
- The AEP contained in RCX is stored into the TCS for use by AEExs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSOFBASE (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 43.1.2):
  - On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEEx). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 43.2.5).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 43.2.3).
- 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 43.2.3):

- All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
- PEBS is suppressed.
- AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set.
- If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Concurrence Restrictions of ERESUME with Intel® SGX Instructions - 1 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EEXIT</th>
<th>EADD</th>
<th>EBLOCK</th>
<th>ECRE ATE</th>
<th>EDBGRD/ WR</th>
<th>ENTER/ ERESUME</th>
<th>EEXTEND</th>
<th>EGETKEY</th>
<th>EINIT</th>
<th>ELDB/ELDU</th>
<th>EPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERESUME</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS SSA</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>SSA</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>SECS</td>
<td>Targ</td>
<td>VA</td>
<td>SECS</td>
<td>SECS</td>
<td>SECS</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Concurrence Restrictions of ERESUME with Intel® SGX Instructions - 2 of 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>EREMOVE</th>
<th>EREPORT</th>
<th>ETRA Ck</th>
<th>EWB</th>
<th>EAUG</th>
<th>EMODPE</th>
<th>EMODPR</th>
<th>EMODT</th>
<th>EACCEPT</th>
<th>EACCEPTCOPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>Targ</td>
<td>SECS</td>
<td>Param</td>
<td>SECS</td>
<td>SECS</td>
<td>SRC VA</td>
<td>SECS</td>
<td>Targ</td>
<td>SECS</td>
<td>Targ SECS</td>
</tr>
<tr>
<td>ERESUME</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
<td>TCS N</td>
</tr>
<tr>
<td>SSA</td>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SECS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Operation

Temp Variables in ERESUME Operational Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMP_FSBASE</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Proposed base address for FS segment.</td>
</tr>
<tr>
<td>TMP_GSBASE</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Proposed base address for FS segment.</td>
</tr>
<tr>
<td>TMP_FSLIMIT</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Highest legal address in proposed FS segment.</td>
</tr>
<tr>
<td>TMP_GSLIMIT</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Highest legal address in proposed GS segment.</td>
</tr>
<tr>
<td>TMP_TARGET</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Address of first instruction inside enclave at which execution is to resume.</td>
</tr>
<tr>
<td>TMP_SECS</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Physical address of SECS for this enclave.</td>
</tr>
<tr>
<td>TMP_SSA</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Address of current SSA frame.</td>
</tr>
<tr>
<td>TMP_XSIZE</td>
<td>integer</td>
<td>64</td>
<td>Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.</td>
</tr>
<tr>
<td>TMP_SSA_PAGE</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Pointer used to iterate over the SSA pages in the current frame.</td>
</tr>
<tr>
<td>TMP_GPR</td>
<td>Effective Address</td>
<td>32/64</td>
<td>Address of the GPR area within the current SSA frame.</td>
</tr>
<tr>
<td>TMP_BRANCH_RECORD</td>
<td>LBR Record</td>
<td></td>
<td>From/to addresses to be pushed onto the LBR stack.</td>
</tr>
</tbody>
</table>

\[\text{TMP\_MODE64} \leftarrow ((\text{IA32\_EFER.LMA} = \text{1}) \&\& (\text{CS.L} = \text{1}));\]
(* Make sure DS is usable, expand up *)
IF (TMP_MODE64 = 0 and (DS not usable or ((DS[5] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1)))
    THEN #GP(0); Fl;

(* 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); Fl;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); Fl;

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

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions is operating on TCS)
    THEN #GP(0); Fl;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); Fl;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); Fl;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); Fl;

IF ((DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); Fl;

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

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

(* Make sure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ((DS:RBX).FLAGS & & FFFFFFFFFFFFFF7E) ≠ 0)
    THEN #GP(0); Fl;
SGX INSTRUCTION REFERENCES

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

(* make sure the logical processor’s operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
    THEN #GP(0); FI;

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

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

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

(* Compute linear address of SSA frame *)
TMP_SSA ← (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * ( (DS:RBX).CSSA - 1);
TMP_XSIZE ← compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);
FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;
    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
         (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
         (EPCM(DS:TMP_SECS).R = 0) or (EPCM(DS:TMP_SECS).W = 0) )
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    CR_XSAVE_PAGE_n ← Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR ← TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE -- sizeof(GPRSGX_AREA);
Check that DS:TMP_SSA_PAGE is read/write accessible;
If a fault occurs, release locks, abort and deliver that fault;
IF (DS:TMP_GPR does not resolve to EPC page)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
    THEN #PF(DS:TMP_GPR); FI;
\[
\begin{align*}
&\text{IF} \ (\text{EPCM(DS:TMP\_GPR).BLOCKED} = 1) \\
&\quad \text{THEN} \ #PF(\text{DS:TMP\_GPR}); \text{FI;}
\\
&\text{IF} \ ((\text{EPCM(DS:TMP\_GPR).PENDING} = 1) \text{ or } (\text{EPCM(DS:TMP\_GPR).MODIFIED} = 1)) \\
&\quad \text{THEN} \ #PF(\text{DS:TMP\_GPR}); \text{FI;}
\\
&\text{IF} \ ((\text{EPCM(DS:TMP\_GPR).ENCLAVEADDRESS} \neq \text{DS:TMP\_GPR}) \text{ or } (\text{EPCM(DS:TMP\_GPR).PT} \neq \text{PT\_REG}) \text{ or } \\
&\quad (\text{EPCM(DS:TMP\_GPR).SECS} \neq \text{EPCM(DS:RBX).SECS}) \text{ or } (\text{EPCM(DS:TMP\_GPR).R} = 0) \text{ or } (\text{EPCM(DS:TMP\_GPR).W} = 0)) \\
&\quad \text{THEN} \ #PF(\text{DS:TMP\_GPR}); \text{FI;}
\\
&\text{IF} \ (\text{TMP\_MODE64} = 0) \\
&\quad \text{THEN} \\
&\quad \quad \text{IF} \ (\text{TMP\_GPR} + (\text{GPR\_SIZE} - 1) \text{ is not in DS segment}) \text{ THEN} \ #GP(0); \text{FI;}
&\quad \text{FI;}
\\
&\quad \text{CR\_GPR\_PA} \leftarrow \text{Physical\_Address(\text{DS:TMP\_GPR});}
\\
&\quad \text{TMP\_TARGET} \leftarrow (\text{DS:TMP\_GPR}).\text{RIP;}
\\
&\quad \text{IF} \ (\text{TMP\_MODE64} = 1) \\
&\quad \quad \text{THEN} \\
&\quad \quad \quad \text{IF} \ (\text{TMP\_TARGET} \text{ is not canonical}) \text{ THEN} \ #GP(0); \text{FI;}
&\quad \quad \text{ELSE} \\
&\quad \quad \quad \text{IF} \ (\text{TMP\_TARGET} > \text{CS limit}) \text{ THEN} \ #GP(0); \text{FI;}
&\quad \text{FI;}
\\
&\quad \quad \text{(* Check proposed FS/GS segments fall within DS *)}
\\
&\quad \text{IF} \ (\text{TMP\_MODE64} = 0) \\
&\quad \quad \text{THEN} \\
&\quad \quad \quad \text{TMP\_FSBASE} \leftarrow (\text{DS:RBX}).\text{OFSBASE} + \text{TMP\_SECS.BASEADDR;}
&\quad \quad \quad \text{TMP\_FSLIMIT} \leftarrow (\text{DS:RBX}).\text{OFSBASE} + \text{TMP\_SECS.BASEADDR} + (\text{DS:RBX}).\text{FSLIMIT;}
&\quad \quad \quad \text{TMP\_GSBASE} \leftarrow (\text{DS:RBX}).\text{OGSBASE} + \text{TMP\_SECS.BASEADDR;}
&\quad \quad \quad \text{TMP\_GSLIMIT} \leftarrow (\text{DS:RBX}).\text{OGSBASE} + \text{TMP\_SECS.BASEADDR} + (\text{DS:RBX}).\text{GSLIMIT;}
&\quad \quad \quad \text{(* if FS wrap-around, make sure DS has no holes*)}
&\quad \quad \quad \text{IF} \ (\text{TMP\_FSLIMIT} < \text{TMP\_FSBASE}) \\
&\quad \quad \quad \quad \text{THEN} \\
&\quad \quad \quad \quad \quad \text{IF} \ (\text{DS.limit} < 4GB) \text{ THEN} \ #GP(0); \text{FI;}
&\quad \quad \quad \quad \text{ELSE}
&\quad \quad \quad \quad \quad \text{IF} \ (\text{TMP\_FSLIMIT} > \text{DS.limit}) \text{ THEN} \ #GP(0); \text{FI;}
&\quad \quad \quad \quad \text{FI;}
&\quad \quad \quad \quad \text{(* if GS wrap-around, make sure DS has no holes*)}
&\quad \quad \quad \quad \text{IF} \ (\text{TMP\_GSLIMIT} < \text{TMP\_GSBASE}) \\
&\quad \quad \quad \quad \quad \text{THEN} \\
&\quad \quad \quad \quad \quad \quad \text{IF} \ (\text{DS.limit} < 4GB) \text{ THEN} \ #GP(0); \text{FI;}
&\quad \quad \quad \quad \quad \text{ELSE}
&\quad \quad \quad \quad \quad \quad \text{IF} \ (\text{TMP\_GSLIMIT} > \text{DS.limit}) \text{ THEN} \ #GP(0); \text{FI;}
&\quad \quad \quad \quad \quad \quad \text{FI;}
&\quad \quad \quad \quad \text{ELSE}
&\quad \quad \quad \quad \quad \quad \text{TMP\_FSBASE} \leftarrow (\text{DS:RBX}).\text{OFSBASE} + \text{TMP\_SECS.BASEADDR;}
&\quad \quad \quad \quad \quad \quad \text{TMP\_GSBASE} \leftarrow (\text{DS:RBX}).\text{OGSBASE} + \text{TMP\_SECS.BASEADDR;}
&\quad \quad \quad \quad \quad \quad \text{IF} \ ((\text{TMP\_FSBASE} \text{ is not canonical}) \text{ or } (\text{TMP\_GSBASE} \text{ is not canonical})) \\
&\quad \quad \quad \quad \quad \quad \quad \quad \text{THEN} \ #GP(0); \text{FI;}
&\quad \quad \quad \quad \text{FI;}
\\
&\quad \quad \text{(* Ensure the enclave is not already active and this thread is the only one using the TCS*)}
&\quad \quad \text{IF} \ (\text{DS:RBX\_STATE} = \text{ACTIVE})
\\
&\end{align*}
\]
THEN #GP(0); Fl;

(* 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);
    Fl;

CR_ENCALVE_MODE ← 1;
CR_ACTIVE_SECS ← TMP_SECS;
CR_EL RANGE ← (TMP_SECS.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;

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

Restore_GPRs from DS:TMP_GPR;

(*Restore the RFLAGS values from SSA*)
RFLAGS.CF ← DS:TMP_GPR.RFLAGS.CF;
RFLAGS.PF ← DS:TMP_GPR.RFLAGS.PF;
RFLAGS.AF ← DS:TMP_GPR.RFLAGS.AF;
RFLAGS.ZF ← DS:TMP_GPR.RFLAGS.ZF;
RFLAGS.SF ← DS:TMP_GPR.RFLAGS.SF;
RFLAGS.DF ← DS:TMP_GPR.RFLAGS.DF;
RFLAGS.OF ← DS:TMP_GPR.RFLAGS.OF;
RFLAGS.NT ← DS:TMP_GPR.RFLAGS.NT;
RFLAGS.AC ← DS:TMP_GPR.RFLAGS.AC;
RFLAGS.ID ← DS:TMP_GPR.RFLAGS.ID;
RFLAGS.RF ← DS:TMP_GPR.RFLAGS.RF;
RFLAGS.VM ← 0;
IF (RFLAGS.IOP = 3)
    THEN RFLAGS.IF = DS:TMP_GPR.IF; Fl;

IF (TCS.FLAGS.OPTIN = 0)
THEN RFLAGS.TF = 0; Fi;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
   CR_SAVE_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 any MTF VM exits during execution of the enclave;
      Clear all pending debug exceptions;
      Clear any pending MTF VM exit;
   ELSE
      Clear all pending debug exceptions;
      Clear pending MTF VM exits;
SGX INSTRUCTION REFERENCES

Fl;

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

Flags Affected
RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0) If DS:RBX is not page aligned.
If the enclave is not initialized.
If the thread is not in the INACTIVE state.
If CS, DS, ES or SS bases are not all zero.
If executed in enclave mode.
If part or all of the FS or GS segment specified by TCS is outside the DS segment.
If any reserved field in the TCS FLAG is set.
If the target address is not within the CS segment.
If CR4.OSFXSR = 0.
If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

#PF(error code) If a page fault occurs in accessing memory.
If DS:RBX does not point to a valid TCS.
If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0) If DS:RBX is not page aligned.
If the enclave is not initialized.
If the thread is not in the INACTIVE state.
If CS, DS, ES or SS bases are not all zero.
If executed in enclave mode.
If part or all of the FS or GS segment specified by TCS is outside the DS segment.
If any reserved field in the TCS FLAG is set.
If the target address is not canonical.
If CR4.OSFXSR = 0.
If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
If CR4.OSXSAVE = 1 and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

#PF(error code) If a page fault occurs in accessing memory operands.
If DS:RBX does not point to a valid TCS.
If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.
This page was intentionally left blank.
SGX INSTRUCTION REFERENCES
Intel® SGX provides Intel® Architecture with a collection of enclave instructions for creating protected execution environments on processors supporting IA32 and Intel® 64 architectures. These Intel SGX instructions are designed to work with legacy software and the various IA32 and Intel 64 modes of operation.

### 42.1 Intel® SGX Availability in Various Processor Modes

The Intel SGX extensions (see Table 37-1) 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.

### 42.2 IA32_FEATURE_CONTROL

IA32_FEATURE_CONTROL MSR provides two new bits related to two aspects of Intel SGX: using the instruction extensions and launch control configuration.

#### 42.2.1 Availability of Intel SGX

IA32_FEATURE_CONTROL[bit 18] allows BIOS to control the availability of Intel SGX extensions. For Intel SGX extensions to be available on a logical processor, bit 18 in the IA32_FEATURE_CONTROL MSR on that logical processor must be set, and IA32_FEATURE_CONTROL MSR on that logical processor must be locked (bit 0 must be set). See Section 37.7.1 for additional details. OS is expected to examine the value of bit 18 prior to enabling Intel SGX on the thread, as the settings of bit 18 is not reflected by CPUID.

#### 42.2.2 Intel SGX Launch Control Configuration

The IA32_SGXLEPUBKEYHASHn MSRs used to configure authorized launch enclaves' MRSIGNER digest value. They are present on logical processors that support the collection of SGX1 leaf functions (i.e. CPUID.(EAX=12H, ECX=00H):EAX[0] = 1) and that CPUID.(EAX=07H, ECX=00H):ECX[30] = 1. IA32_FEATURE_CONTROL[bit 17] allows BIOS to enable write access to these MSRs. If IA32_FEATURE_CONTROL.LE_WR (bit 17) is set to 1 and IA32_FEATURE_CONTROL is locked on that logical processor, IA32_SGXLEPUBKEYHASHn MSRs on that logical processor then the IA32_SGXLEPUBKEYHASHn MSR are writeable. If this bit 17 is not set or IA32_FEATURE_CONTROL is not locked, IA32_SGXLEPUBKEYHASHn MSRs are read only. See Section 39.1.4 for additional details.

### 42.3 Interactions with Segmentation

#### 42.3.1 Scope of Interaction

Intel SGX extensions are available only when the processor is executing in a protected mode operation (see Section 42.1 for Intel SGX availability in various processor modes). Enclaves abide by all the segmentation policies set up by the OS, but they can be more restrictive than the OS.

Intel SGX interacts with segmentation at two levels:

- The Intel SGX instruction (see the enclave instruction in Table 37-1).
42.3.2 Interactions of Intel® SGX Instructions with Segment, Operand, and Addressing Prefixes

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

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

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

42.3.3 Interaction of Intel® SGX Instructions with Segmentation

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

The Intel SGX leaf functions used for entering the enclave (ENCLU[EENTER] and ENCLU[ERESUME]) operate as follows:

- All usable segment registers except for FS and GS have a zero base.
- The contents of the FS/GS segment registers (including the hidden portion) is saved in the processor.
- New FS and GS values compatible with enclave security are loaded from the TCS
- The linear ranges and access rights available under the newly-loaded FS and GS must abide to OS policies by ensuring they are subsets of the linear-address range and access rights available for the DS segment.
- The CS segment mode (64-bit, compatible, or 32 bit modes) must be consistent with the segment mode for which the enclave was created, as indicated by the SECS.ATTRIBUTES.MODE64 bit, and that the CPL of the logical processor is 3

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

42.3.4 Interactions of Enclave Execution with Segmentation

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

Additionally, any attempt by software executing inside an enclave to modify the processor’s segmentation state (e.g. via MOV seg register, POP seg register, LDS, far jump, etc; excluding WRFSBASE/WRGSBASE) results in the generation of a #UD. See Section 39.6.1 for more information.

Upon enclave entry via the EENTER leaf function, FS is loaded from the (TCS.OFSBASE + SECS.BASEADDR) and TCS.FSLIMIT fields and GS is loaded from the (TCS.OGSBASE + SECS.BASEADDR) and TCS.GSLIMIT fields.

Execution of WRFSBASE and WRGSBASE from inside a 64-bit enclave is allowed. The processor will save the new values into the current SSA frame on an asynchronous exit (AEX) and restore them back on enclave entry via ENCLU[ERESUME] instruction.

42.4 INTERACTIONS WITH PAGING

Intel SGX instructions are available only when the processor is executing in a protected mode of operation. Additionally, all Intel SGX leaf functions except for EDBGRD and EDBGWR are available only if paging is enabled. Any attempt to execute these leaf functions with paging disabled results in an invalid-opcode exception (#UD). As with
segmentation, enclaves abide by all the paging policies set up by the OS, but they can be more restrictive than the OS.

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

Since the ENCLU[ENTER] and ENCLU[ERESUME] can only be executed when paging is enabled, and since paging cannot be disabled by software running inside an enclave (recall that enclaves always run with CPL = 3), enclave execution is always subject to paging-based access control. The Intel SGX access control itself is implemented as an extension to the existing paging modes. See Section 38.5 for details.

Execution of Intel SGX instructions may set accessed and dirty flags on accesses to EPC pages that do not fault even if the instruction later causes a fault for some other reason.

### 42.5 INTERACTIONS WITH VMX

Intel SGX functionality (including SGX1 and SGX2) can be made available to software running in either VMX root operation or VMX non-root operation, as long as the processor is using a legal mode of operation (see Section 42.1).

A VMM has the flexibility to configure a VMCS to permit a guest to use any subset of the ENCLS leaf functions. Availability of the ENCLU leaf functions in VMX non-root operation has the same requirement as ENCLU leaf functions outside of a virtualized environment.

Details of the VMCS control to allow VMM to configure support of Intel SGX in VMX non-root operation is described in Section 42.5.1.

#### 42.5.1 VMM Controls to Configure Guest Support of Intel® SGX

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

Some of Intel SGX resources are exposed/controlled via model-specific registers (see Section 37.7). VMMs can virtualize these MSRs for the guests using the MSR bitmaps referenced by pointers in the VMCS.

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

The VMM can set the “enable ENCLS exiting” VM-execution controls to cause a VM exit when the ENCLS instruction is executed in VMX non-root operation. If the “enable ENCLS exiting” control is 0, all of the ENCLS leaf functions are permitted in VMX non-root operation. If the “enable ENCLS exiting” control is 1, execution of ENCLS leaf functions in VMX non-root operation is governed by consulting the bits in a new 64-bit VM-execution control field called the ENCLS-exiting bitmap (Each bit in the bitmap corresponds to an ENCLS leaf function with an EAX value that is identical to the bit’s position). When bits in the “ENCLS-exiting bitmap” are set, attempts to execute the corresponding ENCLS leaf functions in VMX non-root operation causes VM exits. The checking for these VM exits occurs immediately after checking that CPL = 0.

#### 42.5.2 Interactions with the Extended Page Table Mechanism (EPT)

Intel SGX instructions are fully compatible with the extended page-table mechanism (EPT; see Section 28.2).

All the memory operands passed into Intel SGX instructions are interpreted as offsets within the DS segment, and the linear addresses generated by combining these offsets with DS segment register are subject to paging and EPT. As with paging, enclaves abide by all the policies set up by the VMM.

The Intel SGX access control itself is implemented as an extension to paging and EPT, and may be more restrictive. See Section 42.4 for details of this extension.

An execution of an Intel SGX instruction may set accessed and dirty flags for EPT (when enabled; see Section 28.2.4) on accesses to EPC pages that do not fault or cause VM exits even if the instruction later causes a fault or VM exit for some other reason.
42.5.3 Interactions with APIC Virtualization

This section applies to Intel SGX in VMX non-root operation when the "virtualize APIC accesses" VM-execution control is 1.

A memory access by an enclave instruction that implicitly uses a cached physical address is never checked for overlap with the APIC-access page. Such accesses never cause APIC-access VM exits and are never redirected to the virtual-APIC page. Implicit memory accesses can only be made to the SECS, the TCS, or the SSA of an enclave (see Section 38.5.3.2).

An explicit Enclave Access (a linear memory access which is either from within an enclave into its ELRANGE, or an access by an Intel SGX instruction that is expected to be in the EPC) that overlaps with the APIC-access page causes a #PF exception (APIC page is expected to be outside of EPC).

Non-Enclave accesses made either by an Intel SGX instruction or by a logical processor inside an enclave to an address that without SGX would have caused redirection to the virtual-APIC page instead cause an APIC-access VM exit.

Other than implicit accesses made by Intel SGX instructions, guest-physical and physical accesses are not considered "enclave accesses"; consequently, such accesses result in undefined behavior if these accesses eventually reach EPC. This applies to any non-enclave physical accesses.

While a logical processor is executing inside an enclave, an attempt to execute an instruction outside of ELRANGE results in a #GP(0), even if the linear address would translate to a physical address that overlaps the APIC-access page.

42.6 INTEL® SGX INTERACTIONS WITH ARCHITECTURALLY-VISIBLE EVENTS

All architecturally visible vectored events (IA32 exceptions, interrupts, SMI, NMI, INIT, VM exit) can be detected while inside an enclave and will cause an asynchronous enclave exit if they are not blocked. Additionally, INT3, and the SignalTXTMsg[SENTER] (i.e. GETSEC[SENTER]'s rendezvous event message) events also cause asynchronous enclave exits. Note that SignalTXTMsg[SEXIT] (i.e. GETSEC[SEXIT]'s teardown message) does not cause an AEX.

On an AEX, information about the event causing the AEX is stored in the SSA (see Section 40.4 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.

42.7 INTERACTIONS WITH THE PROCESSOR EXTENDED STATE AND MISCELLANEOUS STATE

42.7.1 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 Intel SGX:

- On an AEX, the Intel SGX architecture must protect the processor extended state and miscellaneous state by saving them in the enclave's state-save area (SSA), and clear the secrets from the processor extended state that is used by an enclave.
- Intel SGX architecture must verify that the SSA frame size is large enough to contain all the processor extended states and miscellaneous state used by the enclave.
- Intel SGX architecture must ensure that enclaves can only use processor extended state that is enabled by system software in XCR0.
• Enclave software should be able to discover only those processor extended state and miscellaneous state for which such protection is enabled.

• The processor extended states that are enabled inside the enclave must be approved by the enclave developer:
  — Certain processor extended state (e.g., Memory Protection Extensions, see Chapter 17, “Intel® MPX” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1) modify the behavior of the legacy ISA software. If such features are enabled for enclaves that do not understand those features, then such a configuration could lead to a compromise of the enclave's security.

• The processor extended states that are enabled inside the enclave must form an integral part of the enclave's identity. This requirement has two implications:
  — Service providers may decide to assign different trust level to the same enclave depending on the ISA features the enclave is using.

To meet these requirements, the Intel SGX architecture defines a sub-field called X-Feature Request Mask (XFRM) in the ATTRIBUTES field of the SECS. On enclave creation (ENCLS[ECREATE] leaf function), the required SSA frame size is calculated by the processor from the list of enabled extended and miscellaneous states and verified against the actual SSA frame size defined by SECS.SSAFRAMESIZE.

On enclave entry, after verifying that XFRM is only enabling features that are already enabled in XCR0, the value in the XCR0 is saved internally by the processor, and is replaced by the XFRM. On enclave exit, the original value of XCR0 is restored. Consequently, while inside the enclave, the processor extended states enabled in XFRM are in enabled state, and those that are disabled in XFRM are in disabled state.

The entire ATTRIBUTES field, including the XFRM subfield is integral part of enclave's identity (i.e., its value is included in reports generated by ENCLU[EREPORT], and select bits from this field can be included in key-derivation for keys obtained via the ENCLU[EGETKEY] leaf function).

Enclave developers can create their enclave to work with certain features and fallback to another code path in case those features aren't available (e.g. optimize for AVX and fallback to SSE). For this purpose Intel SGX provides the following fields in SIGSTRUCT: ATTRIBUTES, ATTRIBUTESMASK, MISCSELECT, and MISCMASK. EINIT ensures that the final SECS.ATTRIBUTES and SECS.MISCSELECT comply with the enclave developer's requirements as follows:

SIGSTRUCT.ATTRIBUTES & SIGSTRUCT.ATTRIBUTESMASK = SECS.ATTRIBUTES & SIGSTRUCT.ATTRIBUTESMASK
SIGSTRUCT.MISCSELECT & SIGSTRUCT.MISCMASK = SECS.MISCSELECT & SIGSTRUCT.MISCMASK.

On an asynchronous enclave exit, the processor extended states enabled by XFRM are saved in the current SSA frame, and overwritten by synthetic state (see Section 40.3 for the definition of the synthetic state). When the interrupted enclave is resumed via the ENCLU[ERESUME] leaf function, the saved state for processor extended states enabled by XFRM is restored.

### 42.7.2 Relevant Fields in Various Data Structures

#### 42.7.2.1 SECS.ATTRIBUTES.XFRM

The ATTRIBUTES field of the SECS data structure (see Section 38.7) contains a sub-field called XSAVE-Feature Request Mask (XFRM). Software populates this field at the time of enclave creation according to the features that are enabled by the operating system and approved by the enclave developer.

Intel SGX architecture guarantees that during enclave execution, the processor extended state configuration of the processor is identical to what is required by the XFRM sub-field. All the processor extended states enabled in XFRM are saved on AEX from the enclave and restored on ERESUME.

The XFRM sub-field has the same layout as XCR0, and has consistency requirements that are similar to those for XCR0. Specifically, the consistency requirements on XFRM values depend on the processor implementation and the set of features enabled in CR4.

Legal values for SECS.ATTRIBUTES.XFRM conform to these requirements:

• XFRM[1:0] must be set to 0x3.

• If the processor does not support XSAVE, or if the system software has not enabled XSAVE, then XFRM[63:2] must be zero.

• If the processor does support XSAVE, XFRM must contain a value that would be legal if loaded into XCR0.
The various consistency requirements are enforced at different times in the enclave's life cycle, and the exact enforcement mechanisms are elaborated in Section 42.7.3 through Section 42.7.6.

On processors not supporting XSAVE, software should initialize XFRM to 0x3. On processors supporting XSAVE, software should initialize XFRM to be a subset of XCR0 that would be present at the time of enclave execution. Because bits 0 and 1 of XFRM must always be set, the use of Intel SGX requires that SSE be enabled (CR4.OSFXSR = 1).

42.7.2.2 SECS.SSAFRAMESIZE

The SSAFRAMESIZE field in the SECS data structure specifies the number of pages which software allocated\(^1\) for each SSA frame, including both the GPRSGX area, MISC area, the XSAVE area (x87 and XMM states are stored in the latter area), and optionally padding between the MISC and XSAVE area. The GPRSGX area must hold all the general-purpose registers and additional Intel SGX specific information. The MISC area must hold the Miscellaneous state as specified by SECS.MISCSELECT, the XSAVE area holds the set of processor extended states specified by SECS.ATTRIBUTES.XFRM (see Section 38.9 for the layout of SSA and Section 42.7.3 for ECREATE's consistency checks). The SSA is always in non-compacted format.

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

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

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

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

The size of the MISC region depends on the setting of SECS.MISCSELECT and can be calculated using the layout information described in Section 38.9.2

42.7.2.3 XSAVE Area in SSA

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

42.7.2.4 MISC Area in SSA

The MISC area of an SSA frame is positioned immediately before the GPRSGX region.

42.7.2.5 SIGSTRUCT Fields

Intel SGX provides the flexibility for an enclave developer to choose the enclave's code path according to the features that are enabled on the platform (e.g. optimize for AVX and fallback to SSE). See Section 42.7.1 for details.

\(^1\) It is the responsibility of the enclave to actually allocate this memory.
SIGSTRUCT includes the following fields:
SIGSTRUCT.ATTRIBUTES, SIGSTRUCT.ATTRIBUTEMASK, SIGSTRUCT.MISCSELECT, SIGSTRUCT.MISCMASK.

### 42.7.2.6 REPORT.ATTRIBUTES.XFRM and REPORT.MISCSELECT

The processor extended states and miscellaneous states that are enabled inside the enclave form an integral part of the enclave's identity and are therefore included in the enclave's report, as provided by the ENCLU[EREPORT] leaf function. The REPORT structure includes the enclave's XFRM and MISCSELECT configurations.

### 42.7.2.7 KEYREQUEST

An enclave developer can specify which bits out of XFRM and MISCSELECT ENCLU[EGETKEY] should include in the derivation of the sealing key by specifying ATTRIBUTESMASK and MISCMASK in the KEYREQUEST structure.

### 42.7.3 Processor Extended States and ENCLS[ECREATE]

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

- SECS.ATTRIBUTES.XFRM[1:0] is not 3.
- The processor does not support XSAVE and any of the following is true:
  - SECS.ATTRIBUTES.XFRM[63:2] is not 0.
  - SECS.SSAFRAMESIZE is 0.
- The processor supports XSAVE and any of the following is true:
  - XSETBV would fault on an attempt to load XFRM into XCR0.
  - XFRM[63]=1.
  - The SSAFRAME is too small to hold required, enabled states (see Section 42.7.2.2).

### 42.7.4 Processor Extended States and ENCLU[EENTER]

#### 42.7.4.1 Fault Checking

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

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

#### 42.7.4.2 State Loading

If ENCLU[EENTER] is successful, the current value of XCR0 is saved internally by the processor and replaced by SECS.ATTRIBUTES.XFRM.
42.7.5 Processor Extended States and AEX

42.7.5.1 State Saving

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

Note that enclave entry ensures that if CR4.OSXSAVE is set to 0, then SECS.ATTRIBUTES.XFRM is set to 3. It should also be noted that it is not possible to enter an enclave with FXSAVE disabled.

42.7.5.2 State Synthesis

After saving the extended state, the processor restores XCR0 to the value it held at the time of the most recent enclave entry.

The state of features corresponding to bits set in XFRM is synthesized. In general, these states are initialized. Details of state synthesis on AEX are documented in Section 40.3.1.

42.7.6 Processor Extended States and ENCLU[ERESUME]

42.7.6.1 Fault Checking

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

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

A successful execution of ENCLU[ERESUME] loads state from the XSAVE area of the SSA frame in a fashion similar to that used by the XRSTOR instruction. Data in the XSAVE area that would cause the XRSTOR instruction to fault will cause the ENCLU[ERESUME] leaf function to fault. Examples include, but are not restricted to the following:

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

Any of these conditions will cause ERESUME to fault, even if CR4.OSXSAVE=0.

42.7.6.2 State Loading

If ENCLU[ERESUME] is successful, the current value of XCR0 is saved internally by the processor and replaced by SECS.ATTRIBUTES.XFRM.

State is loaded from the XSAVE area of the SSA frame as if the XRSTOR instruction were executed with XCR0=XFRM, EDX:EAX = XFRM, with the memory operand being the XSAVE area, and (for 64-bit enclaves) as if REX.W=1.

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

42.7.7 Processor Extended States and ENCLU[EEXIT]

The ENCLU[EEXIT] leaf function does not perform any X-feature specific consistency checks, nor performs any state synthesis. It is the responsibility of enclave software to clear any sensitive data from the registers before
executing EEXIT. However, successful execution of the ENCLU[EEXIT] leaf function restores XCR0 to the value it held at the time of the most recent enclave entry.

42.7.8 Processor Extended States and ENCLU[EREPORT]

The ENCLU[EREPORT] leaf function creates the MAC-protected REPORT structure that reports on the enclave’s identity. ENCLU[EREPORT] includes in the report the values of SECS.ATTRIBUTES.XFRM and SECS.MISCSELECT.

42.7.9 Processor Extended States and ENCLU[EGETKEY]

The ENCLU[EGETKEY] leaf function returns a cryptographic key based on the information provided by the KEYREQUEST structure. Intel SGX provides the means for isolation between different operating conditions by allowing an enclave developer to select which bits out of XFRM and MISCSELECT need to be included in the derivation of the keys.

42.8 INTERACTIONS WITH SMM

42.8.1 Availability of Intel® SGX instructions in SMM

Enclave instructions are not available in SMM, and any attempt to execute ENCLS or ENCLU instructions inside SMM results in an invalid-opcode exception (#UD).

42.8.2 SMI while Inside an Enclave

If the logical processor executing inside an enclave receives an SMI, the logical processor exits the enclave asynchronously. The response to an SMI received while executing inside an enclave depends on whether the dual-monitor treatment is enabled. For detailed discussion of transfer to SMM, see Chapter 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).

If the logical processor executing inside an enclave receives an SMI when dual-monitor treatment is enabled, the logical processor exits the enclave asynchronously, and transfers the control to the SMM monitor via SMM VM exit. The SMM VM exit sets the “Enclave Interruption” bit in the Exit Reason (see Table 42-1) and in the Guest Interruptibility State field (see Table 42-2) of the SMM VMCS.

42.8.3 SMRAM Synthetic State of AEX Triggered by SMI

All processor registers saved in the SMRAM have the same synthetic values listed in Section 40.3. Additional SMRAM fields that are treated specially on SMI are:

<table>
<thead>
<tr>
<th>Position Field</th>
<th>Value</th>
<th>Writable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMRAM Offset 07EE0H.Bit 1</td>
<td>ENCLAVE INTERRUPTION</td>
<td>Set to 1 if exit occurred in enclave mode</td>
</tr>
</tbody>
</table>
42.9 INTERACTIONS OF INIT, SIPI, AND WAIT-FOR-SIPI WITH INTEL® SGX

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

INIT received inside an enclave, while the logical processor (LP) is in VMX root operation, follows regular Intel Architecture behavior and is blocked.

INIT received inside an enclave, while the logical processor is in VMX non-root operation, causes an AEX. Subsequent to the AEX, the INIT causes a VM exit with the Enclave Interruption bit in the exit-reason field in the VMCS. A processor cannot be inside an enclave in the wait-for-SIPI state. Consequently, a SIPI received while inside an enclave is lost.

Intel SGX does not change the behavior of the processor in the wait-for-SIPI state.

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

- EPC Settings: Unchanged
- EPCM: Unchanged
- CPUID.LEAF_12H.*: Unchanged
- ENCLAVE_MODE: 0 (LP exits enclave asynchronously)
- MEE state: Unchanged

Software should be aware that following INIT-SIPI-SIPI, the EPC might contain valid pages and should take appropriate measures such as initialize the EPC with the EREMOVE leaf function.

42.10 INTERACTIONS WITH DMA

DMA is not allowed to access any Processor Reserved Memory.

42.11 INTERACTIONS WITH TXT

42.11.1 Enclaves Created Prior to Execution of GETSEC

Enclaves which have been created before the GETSEC[SENTER] leaf function are available for execution after the successful completion of GETSEC[SENTER] and the corresponding SINIT ACM. Actions that a TXT Launched Environment performs in preparation to execute code in the Launched Environment, also applies to enclave code that would run after GETSEC[SENTER].

42.11.2 Interaction of GETSEC with Intel® SGX

All leaf functions of the GETSEC instruction are illegal inside an enclave, and results in an invalid-opcode exception (#UD).

Responding Logical Processors (RLP) which are executing inside an enclave at the time a GETSEC[SENTER] event occurs perform an AEX from the enclave and then enter the Wait-for-SIPI state.

RLP executing inside an enclave at the time of GETSEC[SEXIT], behave as defined for GETSEC[SEXIT]-that is, the RLPs pause during execution of SEXIT and resume after the completion of SEXIT.

The execution of a TXT launch does not affect Intel SGX configuration or security parameters.
42.11.3 Interactions with Authenticated Code Modules (ACMs)

Intel SGX only allows launching ACMs with an Intel SGX SVN that is at the same level or higher than the expected Intel SGX SVN. The expected Intel SGX SVN is specified by BIOS and locked down by the processor on the first successful execution of an Intel SGX instruction that doesn’t return an error code. Intel SGX provides interfaces for system software to discover whether a non-faulting Intel SGX instruction has been executed, and evaluate the suitability of the Intel SGX SVN value of any ACM that is expected to be launched by the OS or the VMM.

These interfaces are provided through a read-only MSR called the IA32_SGX_SVN_STATUS MSR (MSR address 500h). The IA32_SGX_SVN_STATUS MSR has the format shown in Table 42-2.

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Name</th>
<th>ACM Module ID</th>
<th>Value</th>
</tr>
</thead>
</table>
| 0            | Lock        | N.A.          | • If 1, indicates that a non-faulting Intel SGX instruction has been executed, consequently, launching a properly signed ACM but with Intel SGX SVN value less than the BIOS specified Intel SGX SVN threshold would lead to an TXT shutdown.  
• If 0, indicates that the processor will allow a properly signed ACM to launch irrespective of the Intel SGX SVN value of the ACM. |
| 15:1         | RSVD        | N.A.          | 0                                                                    |
| 23:16        | SGX_SVN_SINIT | SINIT ACM     | • If CPUID.01H:ECX.SMX = 1, this field reflects the expected threshold of Intel SGX SVN for the SINIT ACM.  
• If CPUID.01H:ECX.SMX = 0, this field is reserved (0). |
| 63:24        | RSVD        | N.A.          | 0                                                                    |

OS/VMM that wishes to launch an architectural ACM such as SINIT is expected to read the IA32_SGX_SVN_STATUS MSR to determine whether the ACM can be launched or a new ACM is needed:

• If either the Intel SGX SVN of the ACM is greater than the value reported by IA32_SGX_SVN_STATUS, or the lock bit in the IA32_SGX_SVN_STATUS is not set, then the OS/VMM can safely launch the ACM.

• If the Intel SGX SVN value reported in the corresponding component of the IA32_SGX_SVN_STATUS is greater than the Intel SGX SVN value in the ACM’s header, and if bit 0 of IA32_SGX_SVN_STATUS is 1, then the OS/VMM should not launch that version of the ACM. It should obtain an updated version of the ACM either from the BIOS or from an external resource.

However, OSVs/VMMs are strongly advised to update their version of the ACM any time they detect that the Intel SGX SVN of the ACM carried by the OS/VMM is lower than that reported by IA32_SGX_SVN_STATUS MSR, irrespective of the setting of the lock bit.

42.12 INTERACTIONS WITH CACHING OF LINEAR-ADDRESS TRANSLATIONS

Entering and exiting an enclave causes the logical processor to flush all the global linear-address context as well as the linear-address context associated with the current VPID and PCID. The MONITOR FSM is also cleared.

42.13 INTERACTIONS WITH INTEL® TRANSACTIONAL SYNCHRONIZATION EXTENSIONS (INTEL® TSX)

1. ENCLU or ENCLS instructions inside an HLE region will cause the flow to be aborted and restarted non-speculatively. ENCLU or ENCLS instructions inside an RTM region will cause the flow to be aborted and transfer control to the fallback handler.

2. If XBEGIN is executed inside an enclave, the processor does NOT check whether the address of the fallback handler is within the enclave.

3. If an RTM transaction is executing inside an enclave and there is an attempt to fetch an instruction outside the enclave, the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.
4. If an RTM transaction is executing inside an enclave and there is a data access to an address within the enclave that denied due to EPCM content (e.g., to a page belonging to a different enclave), the transaction is aborted and control is transferred to the fallback handler. No #GP is delivered.

5. If an RTM transaction executing inside an enclave aborts and the address of the fallback handler is outside the enclave, a #GP is delivered after the abort (EIP reported is that of the fallback handler).

### 42.13.1 HLE and RTM Debug

RTM debug will be suppressed on opt-out enclave entry. After opt-out entry, the logical processor will behave as if IA32_DEBUG_CTL[15] = 0. Any #DB detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if either DR7[11] = 0 OR IA32_DEBUGCTL[15] = 0, any #DB or #BP detected inside an RTM transaction region will just cause an abort with no exception delivered.

After opt-in entry, if DR7[11] = 1 AND IA32_DEBUGCTL[15] = 1, any #DB or #BP detected inside an RTM translation will:

- terminate speculative execution,
- set RIP to the address of the XBEGIN instruction, and
- be delivered as #DB (implying an Intel SGX AEX; any #BP is converted to #DB).

DR6[16] will be cleared, indicating RTM debug (if the #DB causes a VM exit, DR6 is not modified but bit 16 of the pending debug exceptions field in the VMCS will be set).

### 42.14 INTEL® SGX INTERACTIONS WITH S STATES

Whenever an Intel SGX enabled processor enters S3-S5 state, enclaves are destroyed. This is due to the EPC being destroyed when power down occurs. It is the application runtime’s responsibility to re-instantiate an enclave after a power transition for which the enclaves were destroyed.

### 42.15 INTEL® SGX INTERACTIONS WITH MACHINE CHECK ARCHITECTURE (MCA)

#### 42.15.1 Interactions with MCA Events

All architecturally visible machine check events (#MC and CMCI) that are detected while inside an enclave cause an asynchronous enclave exit.

Any machine check exception (#MC) that occurs after Intel SGX is first enables causes Intel SGX to be disabled, (CPUID.SGX_Leaf.0:EAX[SGX1] == 0). It cannot be enabled until after the next reset.

#### 42.15.2 Machine Check Enables (IA32_MCi_CTL)

All supported IA32_MCi_CTL bits for all the machine check banks must be set for Intel SGX to be available (CPUID.SGX_Leaf.0:EAX[SGX1] == 1). Any act of clearing bits from '1 to '0 in any of the IA32_MCi_CTL register may disable Intel SGX (set CPUID.SGX_Leaf.0:EAX[SGX1] to 0) until the next reset.

#### 42.15.3 CR4.MCE

CR4.MCE can be set or cleared with no interactions with Intel SGX.
### 42.16 INTEL® SGX INTERACTIONS WITH PROTECTED MODE VIRTUAL INTERRUPTS

ENCLS[EENTER] modifies neither EFLAGS.VIP nor EFLAGS.VIF.

ENCLS[ERESUME] loads EFLAGS in a manner similar to that of an execution of IRET with CPL = 3. This means that ERESUME modifies neither EFLAGS.VIP nor EFLAGS.VIF regardless of the value of the EFLAGS image in the SSA frame.

AEX saves EFLAGS.VIP and EFLAGS.VIF unmodified into the EFLAGS image in the SSA frame. AEX modifies neither EFLAGS.VIP nor EFLAGS.VIF after saving EFLAGS.

If CR4.PVI = 1, CPL = 3, EFLAGS.VM = 0, IOPL < 3, EFLAGS.VIP = 1, and EFLAGS.VIF = 0, execution of STI causes a #GP fault. In this case, STI modifies neither EFLAGS.IF nor EFLAGS.VIF. This behavior applies without change within an enclave (where CPL is always 3). Note that, if IOPL = 3, STI always sets EFLAGS.IF without fault; CR4.PVI, EFLAGS.VIP, and EFLAGS.VIF are neither consulted nor modified in this case.

### 42.17 INTEL SGX INTERACTION WITH PROTECTION KEYS

SGX interactions with PKRU are as follows:

- CPUID.(EAX=12H, ECX=1):ECX.PKRU indicates whether SECS.ATTRIBUTES.XFRM.PKRU can be set. If SECS.ATTRIBUTES.XFRM.PKRU is set, then PKRU is saved and cleared as part of AEX and is restored as part of ERESUME. If CR4.PKE is set, an enclave can execute RDPKRU and WRKRU independent of whether SECS.ATTRIBUTES.XFRM.PKRU is set.

SGX interactions with domain permission checks are as follows:

1. If CR4.PKE is not set, then legacy and SGX permission checks are not effected.
2. If CR4.PKE is set, then domain permission checks are applied to all non-enclave access and enclave accesses to user pages in addition to legacy and SGX permission checks at a higher priority than SGX permission checks.
3. Implicit accesses aren't subject to domain permission checks.
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.

43.1 CONFIGURATION AND CONTROLS

43.1.1 Debug Enclave vs. Production Enclave

The SECS of each enclave provides a bit, SECS.ATTRIBUTES.DEBUG, indicating whether the enclave is a debug enclave (if set) or a production enclave (if 0). If this bit is set, software outside the enclave can use EDBGRD/EDBGWR to access the EPC memory of the enclave. The value of DEBUG is not included in the measurement of the enclave and therefore doesn't require an alternate SIGSTRUCT to be generated to debug the enclave.

The ATTRIBUTES field in the SECS is reported in the enclave's attestation, and is included in the key derivation. Enclave secrets that were protected by the enclave using Intel SGX keys when it ran as a production enclave will not be accessible by the debug enclave. A debugger needs to be aware that special debug content might be required for a debug enclave to run in a meaningful way.

EPC memory belonging to a debug enclave can be accessed via the EDBGRD/EDBGWR leaf functions (see Section 41.4), while that belonging to a non-debug enclave cannot be accessed by these leaf functions.

43.1.2 Tool-Chain Opt-in

The TCS.FLAGS.DBGOPTIN bit controls interactions of certain debug and profiling features with enclaves, including code/data breakpoints, TF, RF, monitor trap flag, BTF, LBRs, BTM, BTS, Intel Processor Trace, and performance monitoring. This bit is forced to zero when EPC pages are added via EADD. A debugger can set this bit via EDBGWR to the TCS of a debug enclave.

An enclave entry through a TCS with the TCS.FLAGS.DBGOPTIN set to 0 is called an opt-out entry. Conversely, an enclave entry through a TCS with TCS.FLAGS.DBGOPTIN set to 1 is called an opt-in entry.

43.2 SINGLE STEP DEBUG

43.2.1 Single Stepping ENCLS Instruction Leafs

If the RFLAGS.TF bit is set at the beginning of ENCLS, then a single-step debug exception is pending as a trap-class exception on the instruction boundary immediately after the ENCLS instruction. Additionally, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending on the instruction boundary immediately after the instruction if the instruction does not fault.

43.2.2 Single Stepping ENCLU Instruction Leafs

The interactions of the unprivileged Intel SGX instruction ENCLU are leaf dependent.

An enclave entry via EENTER/ERESUME leaf functions of the ENCLU, in certain cases, may mask the RFLAGS.TF bit, and mask the setting of the “monitor trap flag” VM-execution control. In such situations, an exit from the enclave, either via the EEXIT leaf function or via an AEX unmasks the RFLAGS.TF bit and the “monitor trap flag” VM-execu-
tion control. The details of this masking/unmasking and the pending of single stepping events across EENTER/ERESUME/EEXIT/AEX are covered in detail in Section 43.2.3.

If the EFLAGS.TF bit is set at the beginning of EREPORT or EGGETKEY leafs, and if the EFLAGS.TF is not masked by the preceding enclave entry, then a single-step debug exception is pending on the instruction boundary immediately after the ENCLU instruction. Additionally, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, and if the monitor trap flag is not masked by the preceding enclave entry, then an MTF VM exit is pending on the instruction boundary immediately after the instruction.

If the instruction under consideration results in a fault, then the control flow goes to the fault handler, and no single-step debug exception is asserted. In such a situation, if the instruction is executed in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending after the delivery of the fault (or any nested exception). No MTF VM exit occurs if another VM exit occurs before reaching that boundary on which an MTF VM exit would be pending.

### 43.2.3 Single-Stepping Enclave Entry with Opt-out Entry

#### 43.2.3.1 Single Stepping without AEX

Figure 43-1 shows the most common case for single-stepping after an opt-out entry.

![Figure 43-1. Single Stepping with Opt-out Entry - No AEX](image)

In this scenario, if the RFLAGS.TF bit is set at the time of the enclave entry, then a single step debug exception is pending on the instruction boundary after EEXIT. Additionally, if the enclave is executing in VMX non-root operation and the "monitor trap flag" VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after EEXIT.

The value of the RFLAGS.TF bit at the end of EEXIT is the same as the value of RFLAGS.TF at the time of the enclave entry.

#### 43.2.3.2 Single Step Preempted by AEX Due to Non-SMI Event

Figure 43-2 shows the interaction of single stepping with AEX due to a non-SMI event after an opt-out entry.
In this scenario, if the enclave is executing in VMX non-root operation and the “monitor trap flag” VM-execution control is 1, an MTF VM exit is pending on the instruction boundary after the AEX. No MTF VM exit occurs if another VM exit happens before reaching that instruction boundary.

The value of the RFLAGS.TF bit at the end of AEX is the same as the value of RFLAGS.TF at the time of the enclave entry.

### 43.2.4 RFLAGS.TF Treatment on AEX

The value of EFLAGS.TF at the end of AEX from an opt-out enclave is same as the value of EFLAGS.TF at the time of the enclave entry. The value of EFLAGS.TF at the end of AEX from an opt-in enclave is unmodified. The EFLAGS.TF saved in GPR portion of the SSA on an AEX is 0. For more detail see EENTER and ERESUME in Chapter 5.

### 43.2.5 Restriction on Setting of TF after an Opt-Out Entry

Enclave entered through an opt-out entry is not allowed to set EFLAGS.TF. The POPF instruction forces RFLAGS.TF to 0 if the enclave was entered through opt-out entry.

### 43.2.6 Trampoline Code Considerations

Any AEX from the enclave which results in the RFLAGS.TF = 1 on the reporting stack will result in a single-step #DB after the first instruction of the trampoline code if the trampoline is entered using the IRET instruction.

### 43.3 CODE AND DATA BREAKPOINTS

#### 43.3.1 Breakpoint Suppression

Following an opt-out entry:

- Instruction breakpoints are suppressed during execution in an enclave.
- Data breakpoints are not triggered on accesses to the address range defined by ELRANGE.
- Data breakpoints are triggered on accesses to addresses outside the ELRANGE

![Figure 43-2. Single Stepping with Opt-out Entry -AEX Due to Non-SMI Event Before Single-Step Boundary](image-url)
Following an opt-in entry instruction and data breakpoints are not suppressed.
The processor does not report any matches on debug breakpoints that are suppressed on enclave entry. However, the processor does not clear any bits in DR6 that were already set at the time of the enclave entry.

43.3.2 Reporting of Instruction Breakpoint on Next Instruction on a Debug Trap
A debug exception caused by the single-step execution mode or when a data breakpoint condition was met causes the processor to perform an AEX. Following such an AEX, the processor reports in the debug status register (DR6) matches of the new instruction pointer (the AEP address) in a breakpoint address register setup to detect instruction execution.

43.3.3 RF Treatment on AEX
RF flag value saved in SSA is the same as what would have been pushed on stack if the exception or event causing the AEX occurred when executing outside an enclave (see Section 17.3.1.1). Following an AEX, the RF flag is 0 in the synthetic state.

43.3.4 Breakpoint Matching in Intel® SGX Instruction Flows
Implicit accesses made by Intel SGX instructions to EPC regions do not trigger data breakpoints. Explicit accesses made by ENCLS[ECREATE], ENCLS[EADD], ENCLS[EEXTEND], ENCLS[EINIT], ENCLS[EREMOVE], ENCLS[ETRACK], ENCLS[EBLOCK], ENCLS[EPA], ENCLS[EB], ENCLS[ELD], ENCLS[EDBGRD], ENCLS[EDBGWR], ENCLU[EENTER], and ENCLU[ERESUME] to the EPC operands do not trigger data breakpoints.
Explicit accesses made by the Intel SGX instructions (ENCLU[EGETKEY] and ENCLU[EREPORT]) executed by an enclave following an opt-in entry, trigger data breakpoints on accesses to their EPC operands. All Intel SGX instructions trigger data breakpoints on accesses to their non-EPC operands.

43.4 INT3 CONSIDERATION

43.4.1 Behavior of INT3 Inside an Enclave
Inside an enclave, INT3 delivers a fault-class exception and thus does not require the CPL to be less than DPL in the IDT gate 3. Following opt-out entry, the instruction delivers #UD. Following opt-in entry, INT3 delivers #BP.
The RIP saved in the SSA on AEX is that of the INT3 instruction. The RIP saved on the stack (or in the TSS or VMCS) is that of the AEP.
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.

43.4.2 Debugger Considerations
The INT3 is fault-like inside an enclave and the RIP saved in the SSA on AEX is that of the INT3 instruction. Consequently, the debugger must not decrement SSA.RIP for #BP coming from an enclave to re-execute the instruction at the RIP of the INT3 instruction on a subsequent enclave entry.

43.4.3 VMM Considerations
As described above, INT3 executed by enclave delivers #BP with “interruption type” of 3. A VMM that re-injects #BP into the guest can obtain the VM entry interruption information from appropriate VMCS fields (as recommended in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C).
VMMs that create the VM-entry interruption information based on the interruption vector should use event type of 3 (instead of 6) when they detect a VM exit incident to enclave mode that is due to an event with vector 3.

### 43.5 BRANCH TRACING

#### 43.5.1 BTF Treatment

When software enables single-stepping on branches then:

- Following an opt-in entry using EENTER the processor generates a single step debug exception.
- Following an EEXIT the processor generates a single-step debug exception

Enclave entry using ERESUMEx (opt-in or opt-out) and an AEX from the enclave do not cause generation of the single-step debug exception.

#### 43.5.2 LBR Treatment

##### 43.5.2.1 LBR Stack on Opt-in Entry

Following an opt-in entry into an enclave, last branch recording facilities if enabled continued to store branch records in the LBR stack MSRs as follows:

- On enclave entry using EENTER/ERESUME, the processor pushes the address of EENTER/ERESUME instruction into MSR_LASTBRANCH_n_FROM_IP, and the destination address of the EENTER/ERESUME into MSR_LASTBRANCH_n_TO_IP.
- On EEXIT, the processor pushes the address of EEXIT instruction into MSR_LASTBRANCH_n_FROM_IP, and the address of EEXIT destination into MSR_LASTBRANCH_n_TO_IP.
- On AEX, the processor pushes RIP saved in the SSA into MSR_LASTBRANCH_n_FROM_IP, and the address of AEP into MSR_LASTBRANCH_n_TO_IP.
- For every branch inside the enclave, a branch record is pushed on the LBR stack.

Figure 43-3 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.
43.5.2.2 LBR Stack on Opt-out Entry

An opt-out entry into an enclave suppresses last branch recording facilities, and enclave exit after an opt-out entry un-suppresses last branch recording facilities.

Opt-out entry into an enclave does not push any record on LBR stack.

If last branch recording facilities were enabled at the time of enclave entry, then EEXIT following such an enclave entry pushes one record on LBR stack. The MSR_LASTBRANCH_n_FROM_IP of such record holds the linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while the MSR_LASTBRANCH_n_TO_IP of such record holds linear address of the destination of EEXIT.

Additionally, if last branch recording facilities were enabled at the time of enclave entry, then an AEX after such an entry pushes one record on LBR stack, before pushing record for the event causing the AEX if the event pushes a record on LBR stack. The MSR_LASTBRANCH_n_FROM_IP of the new record holds linear address of the instruction (EENTER or ERESUME) that was used to enter the enclave, while MSR_LASTBRANCH_n_TO_IP of the new record holds linear address of the AEP. If the event causing AEX pushes a record on LBR stack, then the MSR_LASTBRANCH_n_FROM_IP for that record holds linear address of the AEP.

Figure 43-4 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.
43.5.2.3 Mispredict Bit, Record Type, and Filtering

All branch records resulting from Intel SGX instructions/AEXs are reported as predicted branches, and consequently, bit 63 of MSR_LASTBRANCH_n_FROM_IP for such records is set. Branch records due to these Intel SGX operations are always non-HLE/non-RTM records.

EENTER, ERESUME, EEXIT, and AEX are considered to be far branches. Consequently, bit 8 in MSR_LBR_SELECT controls filtering of the new records introduced by Intel SGX.

43.6 INTERACTION WITH PERFORMANCE MONITORING

43.6.1 IA32_PERF_GLOBAL_STATUS Enhancement

On processors supporting Intel SGX, the IA32_PERF_GLOBAL_STATUS MSR provides a bit indicator, known as “Anti Side-channel Interference” (ASCI) at bit position 60. If this bit is 0, the performance monitoring data in various performance monitoring counters are accumulated normally as defined by relevant architectural/microarchitectural conditions. If the ASCI bit is set, the contents in various performance monitoring counters can be affected by the direct or indirect consequence of Intel SGX protection of enclave code executing in the processor.

43.6.2 Performance Monitoring with Opt-in Entry

An opt-in enclave entry allow performance monitoring logic to observe the contribution of enclave code executing in the processor. Thus the contents of performance monitoring counters does not distinguish between contribution originating from enclave code or otherwise. All counters, events, precise events, etc. continue to work as defined in the IA32/Intel 64 Software Developer Manual. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is not set.
ENCLAVE CODE DEBUG AND PROFILING

43.6.3 Performance Monitoring with Opt-out Entry

In general, performance monitoring activities are suppressed when entering an opt-out enclave. This applies to all thread-specific, configured performance monitoring, except for the cycle-counting fixed counter, IA32_FIXED_CTR1 and IA32_FIXED_CTR2. Upon entering an opt-out enclave, IA32_FIXED_CTR0, IA32_PMCx will stop accumulating counts. Additionally, if PEBS is configured to capture PEBS record for this thread, PEBS record generation will also be suppressed. Consequently, bit 60 of IA32_PERF_GLOBAL_STATUS MSR is set.

Performance monitoring on the sibling thread may also be affected. Any one of IA32_FIXED_CTRx or IA32_PMCx on the sibling thread configured to monitor thread-specific eventing logic with AnyThread =1 is demoted to count only MyThread while an opt-out enclave is executing on the other thread.

43.6.4 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.

43.6.5 PEBS Record Generation on Intel® SGX Instructions

All leaf functions of the ENCLS instruction report “Eventing RIP” of the ENCLS instruction if a PEBS record is generated at the end of the instruction execution. Additionally, the EGETKEY and EREPORT leaf functions of the ENCLU instruction report “Eventing RIP” of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution.

If the EENTER and ERESUME leaf functions are performing an opt-in entry report “Eventing RIP” of the ENCLU instruction if a PEBS record is generated at the end of the instruction execution. On the other hand, if these leaf functions are performing an opt-out entry, then these leaf functions result in PEBS being suppressed, and no PEBS record is generated at the end of these instructions.

A PEBS record is generated if there is a PEBS event pending at the end of EEXIT (due to a counter overflowing during enclave execution or during EEXIT execution). This PEBS record contains the architectural state of the logical processor at the end of EEXIT. If the enclave was entered via an opt-in entry, then this record reports the “Eventing RIP” as the linear address of the ENCLU[EEXIT] instruction. If the enclave was entered via an opt-out entry, then the record reports the “Eventing RIP” as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A PEBS record is generated after the AEX if there is a PEBS event pending at the end of AEX (due to a counter overflowing during enclave execution or during AEX execution). This PEBS record contains the synthetic state of the logical processor that is established at the end of AEX. For opt-in entry, this record has the EVENTING_RIP set to the RIP saved in the SSA. For opt-out entry, the record has the EVENTING_RIP set to the linear address of EENTER/ERESUME used for the last enclave entry.

If the enclave was entered via an opt-in entry, then this record reports the “Eventing RIP” as the linear address in the SSA of the enclave (a.k.a., the “Eventing LIP” inside the enclave). If the enclave was entered via an opt-out entry, then the record reports the “Eventing RIP” as the linear address of the ENCLU[EENTER/ERESUME] instruction that performed the last enclave entry.

A second PEBS event may be pended during the Enclave Exiting Event (EEE). If the PEBS event is taken at the end of delivery of the EEE then the “Eventing RIP” in this second PEBS record is the linear address of the AEP.

43.6.6 Exception-Handling on PEBS/BTS Loads/Stores after AEX

The operating system should allocate sections of the DS save area from a non-paged pool, and mark them as accessed and dirty. If the loads/stores to any section of the DS save area incur faults then such faults are reported to the OS/VMM immediately, and generation of the PEBS/BTS record is skipped and may leave the buffers in a state where they have a partial PEBS or BTS records.

However, any events that are detected during PEBS/BTS record generation at the end of AEX and before delivering the Enclave Exiting Event (EEE) cannot be reported immediately to the OS/VMM, as an event window is not open at
the end of AEX. Consequently, fault-like events such as page faults, EPT faults, EPT mis-configuration, and accesses to APIC-access page detected on stores to the PEBS/BTS buffer are not reported, and generation of the PEBS and/or BTS record at the end of AEX is aborted (this may leave the buffers in a state where they have partial PEBS or BTS records). Trap-like events detected on stores to the PEBS/BTS buffer (such as debug traps) are pended until the next instruction boundary, where they are handled according to the architecturally defined priority. The processor continues the handling of the Enclave Exit Event (SMI, NMI, interrupt, exception delivery, VM exit, etc.) after aborting the PEBS/BTS record generation.

43.6.6.1 Other Interactions with Performance Monitoring

For opt-in entry, EENTER, ERESUME, EEXIT, and AEX are all treated as predicted far branches, and any counters that are counting such branches are incremented by 1 as a part of retirement of these instructions. Retirement of these instructions is also counted in any counters configured to count instructions retired.

For opt-out entry, execution inside an enclave is treated as a single predicted branch, and all branch-counting performance monitoring counters are incremented accordingly. Additionally, such execution is also counted as a single instruction, and all performance monitoring counters counting instructions are incremented accordingly. Enclave entry does not affect any performance monitoring counters shared between cores.
APPENDIX A

VMX CAPABILITY REPORTING FACILITY

The ability of a processor to support VMX operation and related instructions is indicated by CPUID.1:ECX.VMX[bit 5] = 1. A value 1 in this bit indicates support for VMX features.

Support for specific features detailed in Chapter 26 and other VMX chapters is determined by reading values from a set of capability MSRs. These MSRs are indexed starting at MSR address 480H. VMX capability MSRs are read-only; an attempt to write them (with WRMSR) produces a general-protection exception (#GP(0)). They do not exist on processors that do not support VMX operation; an attempt to read them (with RDMSR) on such processors produces a general-protection exception (#GP(0)).

A.1 BASIC VMX INFORMATION

The IA32_VMX_BASIC MSR (index 480H) consists of the following fields:

- Bits 30:0 contain the 31-bit VMCS revision identifier used by the processor. Processors that use the same VMCS revision identifier use the same size for VMCS regions (see subsequent item on bits 44:32).
- Bit 31 is always 0.
- Bits 44:32 report the number of bytes that software should allocate for the VMXON region and any VMCS region. It is a value greater than 0 and at most 4096 (bit 44 is set if and only if bits 43:32 are clear).
- Bit 48 indicates the width of the physical addresses that may be used for the VMXON region, each VMCS, and data structures referenced by pointers in a VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions). If the bit is 0, these addresses are limited to the processor’s physical-address width. If the bit is 1, these addresses are limited to 32 bits. This bit is always 0 for processors that support Intel 64 architecture.
- If bit 49 is read as 1, the logical processor supports the dual-monitor treatment of system-management interrupts and system-management mode. See Section 34.15 for details of this treatment.
- Bits 53:50 report the memory type that should be used for the VMCS, for data structures referenced by pointers in the VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions), and for the MSEG header. If software needs to access these data structures (e.g., to modify the contents of the MSR bitmaps), it can configure the paging structures to map them into the linear-address space. If it does so, it should establish mappings that use the memory type reported bits 53:50 in this MSR.

As of this writing, all processors that support VMX operation indicate the write-back type. The values used are given in Table A-1.

<table>
<thead>
<tr>
<th>Value(s)</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Uncacheable (UC)</td>
</tr>
<tr>
<td>1-5</td>
<td>Not used</td>
</tr>
<tr>
<td>6</td>
<td>Write Back (wB)</td>
</tr>
<tr>
<td>7-15</td>
<td>Not used</td>
</tr>
</tbody>
</table>

1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field in bits 31:0 of this MSR. For all processors produced prior to this change, bit 31 of this MSR was read as 0.
2. On processors that support Intel 64 architecture, the pointer must not set bits beyond the processor’s physical address width.
3. Alternatively, software may map any of these regions or structures with the UC memory type. (This may be necessary for the MSEG header.) Doing so is discouraged unless necessary as it will cause the performance of software accesses to those structures to suffer.
If software needs to access these data structures (e.g., to modify the contents of the MSR bitmaps), it can configure the paging structures to map them into the linear-address space. If it does so, it should establish mappings that use the memory type reported in this MSR.\(^1\)

- If bit 54 is read as 1, the processor reports information in the VM-exit instruction-information field on VM exits due to execution of the INS and OUTS instructions (see Section 27.2.4). This reporting is done only if this bit is read as 1.
- Bit 55 is read as 1 if any VMX controls that default to 1 may be cleared to 0. See Appendix A.2 for details. It also reports support for the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3.1, Appendix A.3.2, Appendix A.4, and Appendix A.5 for details.
- The values of bits 47:45 and bits 63:56 are reserved and are read as 0.

### A.2 RESERVED CONTROLS AND DEFAULT SETTINGS

As noted in Chapter 26, “VM Entries”, certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its default setting.

Software can discover the default setting of a reserved control by consulting the appropriate VMX capability MSR (see Appendix A.3 through Appendix A.5).

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control's new functionality should set the control to its default setting. For that reason, it is useful for software to know the default settings of the reserved controls.

Default settings partition the various controls into the following classes:

- **Always-flexible.** These have never been reserved.
- **Default0.** These are (or have been) reserved with a default setting of 0.
- **Default1.** They are (or have been) reserved with a default setting of 1.

As noted in Appendix A.1, a logical processor uses bit 55 of the IA32_VMX_BASIC MSR to indicate whether any of the default1 controls may be 0:

- If bit 55 of the IA32_VMX_BASIC MSR is read as 0, all the default1 controls are reserved and must be 1. VM entry will fail if any of these controls are 0 (see Section 26.2.1).
- If bit 55 of the IA32_VMX_BASIC MSR is read as 1, not all the default1 controls are reserved, and some (but not necessarily all) may be 0. The CPU supports four (4) new VMX capability MSRs: IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3 through Appendix A.5 for details. (These MSRs are not supported if bit 55 of the IA32_VMX_BASIC MSR is read as 0.)

See Section 31.5.1 for recommended software algorithms for proper capability detection of the default1 controls.

### A.3 VM-EXECUTION CONTROLS

There are separate capability MSRs for the pin-based VM-execution controls, the primary processor-based VM-execution controls, and the secondary processor-based VM-execution controls. These are described in Appendix A.3.1, Appendix A.3.2, and Appendix A.3.3, respectively.

---

1. Alternatively, software may map any of these regions or structures with the UC memory type. (This may be necessary for the MSEG header.) Doing so is discouraged unless necessary as it will cause the performance of software accesses to those structures to suffer. The processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with the exceptions noted.
A.3.1 Pin-Based VM-Execution Controls

The IA32_VMX_PINBASED_CTLS MSR (index 481H) reports on the allowed settings of most of the pin-based VM-execution controls (see Section 24.6.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the pin-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the pin-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 2, and 4; the corresponding bits of the IA32_VMX_PINBASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any pin-based VM-execution control in the default1 class is 0.
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (see below) reports which of the pin-based VM-execution controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (index 48DH) reports on the allowed settings of all of the pin-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the pin-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_PINBASED_CTLS MSR. (The IA32_VMX_TRUE_PINBASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PINBASED_CTLS MSR. Assuming that software knows that the default1 class of pin-based VM-execution controls contains bits 1, 2, and 4, there is no need for software to consult the IA32_VMX_PINBASED_CTLS MSR.

A.3.2 Primary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS MSR (index 482H) reports on the allowed settings of most of the primary processor-based VM-execution controls (see Section 24.6.2):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary processor-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the primary processor-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 4–6, 8, 13–16, and 26; the corresponding bits of the IA32_VMX_PROCBASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any of the primary processor-based VM-execution controls in the default1 class is 0.
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLS MSR (see below) reports which of the primary processor-based VM-execution controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.
If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLS MSR (index 48EH) reports on the allowed settings of all of the primary processor-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary processor-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_PROCBASED_CTLS MSR. (The IA32_VMX_TRUE_PROCBASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the processor-based VM-execution controls is contained in the IA32_VMX_TRUE_PROCBASED_CTLS MSR. Assuming that software knows that the default1 class of processor-based VM-execution controls contains bits 1, 4–6, 8, 13–16, and 26, there is no need for software to consult the IA32_VMX_PROCBASED_CTLS MSR.

### A.3.3 Secondary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS2 MSR (index 48BH) reports on the allowed settings of the secondary processor-based VM-execution controls (see Section 24.6.2). VM entries perform the following checks:

- Bits 31:0 indicate the allowed 0-settings of these controls. These bits are always 0. This fact indicates that VM entry allows each bit of the secondary processor-based VM-execution controls to be 0 (reserved bits must be 0).
- Bits 63:32 indicate the allowed 1-settings of these controls; the 1-setting is not allowed for any reserved bit. VM entry allows control X (bit X of the secondary processor-based VM-execution controls) to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X and the "activate secondary controls" primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLS2 MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1).

### A.4 VM-EXIT CONTROLS

The IA32_VMX_EXIT_CTLS MSR (index 483H) reports on the allowed settings of most of the VM-exit controls (see Section 24.7.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the VM-exit controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. Exceptions are made for the VM-exit controls in the default1 class (see Appendix A.2). These are bits 0–8, 10, 11, 13, 14, 16, and 17; the corresponding bits of the IA32_VMX_EXIT_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-exit control in the default1 class is 0.
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (see below) reports which of the VM-exit controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (index 48FH) reports on the allowed settings of all of the VM-exit controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
• Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-exit controls:
• If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-exit controls is contained in the IA32_VMX_EXIT_CTLS MSR. (The IA32_VMX_TRUE_EXIT_CTLS MSR is not supported.)
• If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-exit controls is contained in the IA32_VMX_TRUE_EXIT_CTLS MSR. Assuming that software knows that the default1 class of VM-exit controls contains bits 0–8, 10, 11, 13, 14, 16, and 17, there is no need for software to consult the IA32_VMX_EXIT_CTLS MSR.

A.5 VM-ENTRY CONTROLS

The IA32_VMX_ENTRY_CTLS MSR (index 484H) reports on the allowed settings of most of the VM-entry controls (see Section 24.8.1):
• Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the VM-entry controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. Exceptions are made for the VM-entry controls in the default1 class (see Appendix A.2). These are bits 0–8 and 12; the corresponding bits of the IA32_VMX_ENTRY_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:
  — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-entry control in the default1 class is 0.
  — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLS MSR (see below) reports which of the VM-entry controls in the default1 class can be 0 on VM entry.
• Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-entry controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLS MSR (index 490H) reports on the allowed settings of all of the VM-entry controls:
• Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
• Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-entry controls:
• If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_ENTRY_CTLS MSR. (The IA32_VMX_TRUE_ENTRY_CTLS MSR is not supported.)
• If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_TRUE_ENTRY_CTLS MSR. Assuming that software knows that the default1 class of VM-entry controls contains bits 0–8 and 12, there is no need for software to consult the IA32_VMX_ENTRY_CTLS MSR.

A.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:
• Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.
• If bit 5 is read as 1, VM exits store the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control; see Section 27.2 for more details. This bit is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.

• Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
  — Bit 6 reports (if set) the support for activity state 1 (HLT).
  — Bit 7 reports (if set) the support for activity state 2 (shutdown).
  — Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. All implementations support VM entry to activity state 0 (active).

• If bit 14 is read as 1, Intel® Processor Trace (Intel PT) can be used in VMX operation. If the processor supports Intel PT but does not allow it to be used in VMX operation, execution of VMXON clears IA32_RTIT_CTL.TraceEn (see “VMXON—Enter VMX Operation” in Chapter 30); any attempt to set that bit while in VMX operation (including VMX root operation) using the WRMSR instruction causes a general-protection exception.

• If bit 15 is read as 1, the RDMSR instruction can be used in system-management mode (SMM) to read the IA32_SMBASE MSR (MSR address 9EH). See Section 34.15.6.3.

• Bits 24:16 indicate the number of CR3-target values supported by the processor. This number is a value between 0 and 256, inclusive (bit 24 is set if and only if bits 23:16 are clear).

• Bits 27:25 is used to compute the recommended maximum number of MSRs that should appear in the VM-exit MSR-store list, the VM-exit MSR-load list, or the VM-entry MSR-load list. Specifically, if the value bits 27:25 of IA32_VMX_MISC is N, then 512 * (N + 1) is the recommended maximum number of MSRs to be included in each list. If the limit is exceeded, undefined processor behavior may result (including a machine check during the VMX transition).

• If bit 28 is read as 1, bit 2 of the IA32_SMM_MONITOR_CTL can be set to 1. VMXOFF unblocks SMIs unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 34.14.4).

• If bit 29 is read as 1, software can use VMWRITE to write to any supported field in the VMCS; otherwise, VMWRITE cannot be used to modify VM-exit information fields.

• If bit 30 is read as 1, VM entry allows injection of a software interrupt, software exception, or privileged software exception with an instruction length of 0.

• Bits 63:32 and bit 31 are reserved and are read as 0.

### A.7 VMX-FIXED BITS IN CR0

The IA32_VMX_CR0_FIXED0 MSR (index 486H) and IA32_VMX_CR0_FIXED1 MSR (index 487H) indicate how bits in CR0 may be set in VMX operation. They report on bits in CR0 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit of CR0 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit of CR0 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit is also 1 in IA32_VMX_CR0_FIXED1; if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit is also 0 in IA32_VMX_CR0_FIXED0. Thus, each bit in CR0 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in IA32_VMX_CR0_FIXED0 and 1 in IA32_VMX_CR0_FIXED1).

### A.8 VMX-FIXED BITS IN CR4

The IA32_VMX_CR4_FIXED0 MSR (index 488H) and IA32_VMX_CR4_FIXED1 MSR (index 489H) indicate how bits in CR4 may be set in VMX operation. They report on bits in CR4 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit of CR4 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit of CR4 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit is also 1 in IA32_VMX_CR4_FIXED1; if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit is also 0 in IA32_VMX_CR4_FIXED0. Thus, each bit in CR4 is either fixed to...
A.9 VMCS ENUMERATION

The IA32_VMX_VMCS_ENUM MSR (index 48AH) provides information to assist software in enumerating fields in the VMCS.

As noted in Section 24.11.2, each field in the VMCS is associated with a 32-bit encoding which is structured as follows:

- Bits 31:15 are reserved (must be 0).
- Bits 14:13 indicate the field’s width.
- Bit 12 is reserved (must be 0).
- Bits 11:10 indicate the field’s type.
- Bits 9:1 is an index field that distinguishes different fields with the same width and type.
- Bit 0 indicates access type.

IA32_VMX_VMCS_ENUM indicates to software the highest index value used in the encoding of any field supported by the processor:

- Bits 9:1 contain the highest index value used for any VMCS encoding.
- Bit 0 and bits 63:10 are reserved and are read as 0.

A.10 VPID AND EPT CAPABILITIES

The IA32_VMX_EPT_VPID_CAP MSR (index 48CH) reports information about the capabilities of the logical processor with regard to virtual-processor identifiers (VPIDs, Section 28.1) and extended page tables (EPT, Section 28.2):

- If bit 0 is read as 1, the processor supports execute-only translations by EPT. This support allows software to configure EPT paging-structure entries in which bits 1:0 are clear (indicating that data accesses are not allowed) and bit 2 is set (indicating that instruction fetches are allowed).\(^1\)
- Bit 6 indicates support for a page-walk length of 4.
- If bit 8 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be uncacheable (UC); see Section 24.6.11.
- If bit 14 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be write-back (WB).
- If bit 16 is read as 1, the logical processor allows software to configure a EPT PDE to map a 2-Mbyte page (by setting bit 7 in the EPT PDE).
- If bit 17 is read as 1, the logical processor allows software to configure a EPT PDPTPE to map a 1-Gbyte page (by setting bit 7 in the EPT PDPTPE).
- Support for the INVEPT instruction (see Chapter 30 and Section 28.3.3.1).
  - If bit 20 is read as 1, the INVEPT instruction is supported.
  - If bit 25 is read as 1, the single-context INVEPT type is supported.
  - If bit 26 is read as 1, the all-context INVEPT type is supported.
- If bit 21 is read as 1, accessed and dirty flags for EPT are supported (see Section 28.2.4).

\(^1\) If the “mode-based execute control for EPT” VM-execution control is 1, setting bit 0 indicates also that software may also configure EPT paging-structure entries in which bits 1:0 are both clear and in which bit 10 is set (indicating a translation that can be used to fetch instructions from a supervisor-mode linear address or a user-mode linear address).
• If bit 22 is read as 1, the processor reports advanced VM-exit information for EPT violations (see Section 27.2.1). This reporting is done only if this bit is read as 1.
• Support for the INVVPID instruction (see Chapter 30 and Section 28.3.3.1).
  — If bit 32 is read as 1, the INVVPID instruction is supported.
  — If bit 40 is read as 1, the individual-address INVVPID type is supported.
  — If bit 41 is read as 1, the single-context INVVPID type is supported.
  — If bit 42 is read as 1, the all-context INVVPID type is supported.
  — If bit 43 is read as 1, the single-context-retaining-globals INVVPID type is supported.
• Bits 5:1, bit 7, bits 13:9, bit 15, bits 19:18, bits 24:23, bits 31:27, bits 39:33, and bits 63:44 are reserved and are read as 0.

The IA32_VMX_EPT_VPID_CAP MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1) and that support either the 1-setting of the “enable EPT” VM-execution control (only if bit 33 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1) or the 1-setting of the “enable VPID” VM-execution control (only if bit 37 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1).

A.11 VM FUNCTIONS

The IA32_VMX_VMFUNC MSR (index 491H) reports on the allowed settings of the VM-function controls (see Section 24.6.14). VM entry allows bit X of the VM-function controls to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if bit X of the VM-function controls, the “activate secondary controls” primary processor-based VM-execution control, and the “enable VM functions” secondary processor-based VM-execution control are all 1.

The IA32_VMX_VMFUNC MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1) and the 1-setting of the “enable VM functions” secondary processor-based VM-execution control (only if bit 45 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1).
Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 24.11.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.).

B.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 24.11.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.1.1 16-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-1 enumerates the 16-bit control fields.

Table B-1. Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual-processor identifier (VPID)(^1)</td>
<td>00000000B</td>
<td>00000000H</td>
</tr>
<tr>
<td>Posted-interrupt notification vector(^2)</td>
<td>00000001B</td>
<td>00000002H</td>
</tr>
<tr>
<td>EPTP index(^3)</td>
<td>00000010B</td>
<td>00000004H</td>
</tr>
</tbody>
</table>

NOTES:
1. This field exists only on processors that support the 1-setting of the "enable VPID" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "process posted interrupts" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "EPT-violation #VE" VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest ES selector</td>
<td>00000000B</td>
<td>000000800H</td>
</tr>
<tr>
<td>Guest CS selector</td>
<td>00000001B</td>
<td>000000802H</td>
</tr>
<tr>
<td>Guest SS selector</td>
<td>000000010B</td>
<td>000000804H</td>
</tr>
<tr>
<td>Guest DS selector</td>
<td>000000011B</td>
<td>000000806H</td>
</tr>
<tr>
<td>Guest FS selector</td>
<td>00000010B</td>
<td>000000808H</td>
</tr>
<tr>
<td>Guest GS selector</td>
<td>000000101B</td>
<td>00000080AH</td>
</tr>
<tr>
<td>Guest LDTR selector</td>
<td>000000110B</td>
<td>00000080CH</td>
</tr>
<tr>
<td>Guest TR selector</td>
<td>000000111B</td>
<td>00000080EH</td>
</tr>
</tbody>
</table>
### B.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-3 enumerates the 16-bit host-state fields.

#### Table B-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host ES selector</td>
<td>00000000B</td>
<td>00000C00H</td>
</tr>
<tr>
<td>Host CS selector</td>
<td>00000001B</td>
<td>00000C02H</td>
</tr>
<tr>
<td>Host SS selector</td>
<td>00000001B</td>
<td>00000C04H</td>
</tr>
<tr>
<td>Host DS selector</td>
<td>00000001B</td>
<td>00000C06H</td>
</tr>
<tr>
<td>Host FS selector</td>
<td>00000010B</td>
<td>00000C08H</td>
</tr>
<tr>
<td>Host GS selector</td>
<td>00000010B</td>
<td>00000C0AH</td>
</tr>
<tr>
<td>Host TR selector</td>
<td>00000010B</td>
<td>00000C0CH</td>
</tr>
</tbody>
</table>

#### NOTES:
1. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.

### B.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 24.11.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

#### B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

#### Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address of I/O bitmap A (full)</td>
<td>00000000B</td>
<td>00002000H</td>
</tr>
<tr>
<td>Address of I/O bitmap A (high)</td>
<td>00000000B</td>
<td>00002001H</td>
</tr>
<tr>
<td>Address of I/O bitmap B (full)</td>
<td>00000001B</td>
<td>00002002H</td>
</tr>
<tr>
<td>Address of I/O bitmap B (high)</td>
<td>00000001B</td>
<td>00002003H</td>
</tr>
<tr>
<td>Address of MSR bitmaps (full)</td>
<td>00000010B</td>
<td>00002004H</td>
</tr>
<tr>
<td>Address of MSR bitmaps (high)</td>
<td>00000010B</td>
<td>00002005H</td>
</tr>
<tr>
<td>VM-exit MSR-store address (full)</td>
<td>00000011B</td>
<td>00002006H</td>
</tr>
<tr>
<td>VM-exit MSR-store address (high)</td>
<td>00000011B</td>
<td>00002007H</td>
</tr>
</tbody>
</table>
## Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM-exit MSR-load address (full)</td>
<td>000000100B</td>
<td>00002008H</td>
</tr>
<tr>
<td>VM-exit MSR-load address (high)</td>
<td>000000101B</td>
<td>00002009H</td>
</tr>
<tr>
<td>VM-entry MSR-load address (full)</td>
<td>000000110B</td>
<td>0000200AH</td>
</tr>
<tr>
<td>VM-entry MSR-load address (high)</td>
<td>000000111B</td>
<td>0000200BH</td>
</tr>
<tr>
<td>Executive-VMCS pointer (full)</td>
<td></td>
<td>00002000C</td>
</tr>
<tr>
<td>Executive-VMCS pointer (high)</td>
<td></td>
<td>0000200DH</td>
</tr>
<tr>
<td>PML address (full)</td>
<td>000000100B</td>
<td>00002000E</td>
</tr>
<tr>
<td>PML address (high)</td>
<td>000000101B</td>
<td>0000200FH</td>
</tr>
<tr>
<td>TSC offset (full)</td>
<td>000000110B</td>
<td>00002010H</td>
</tr>
<tr>
<td>TSC offset (high)</td>
<td>000000111B</td>
<td>00002011H</td>
</tr>
<tr>
<td>Virtual-APIC address (full)</td>
<td>000001000B</td>
<td>00002012H</td>
</tr>
<tr>
<td>Virtual-APIC address (high)</td>
<td>000001001B</td>
<td>00002013H</td>
</tr>
<tr>
<td>APIC-access address (full)</td>
<td>000001010B</td>
<td>00002014H</td>
</tr>
<tr>
<td>APIC-access address (high)</td>
<td>000001011B</td>
<td>00002015H</td>
</tr>
<tr>
<td>Posted-interrupt descriptor address (full)</td>
<td>000001011B</td>
<td>00002016H</td>
</tr>
<tr>
<td>Posted-interrupt descriptor address (high)</td>
<td>000001012B</td>
<td>00002017H</td>
</tr>
<tr>
<td>VM-function controls (full)</td>
<td>000001100B</td>
<td>00002018H</td>
</tr>
<tr>
<td>VM-function controls (high)</td>
<td>000001101B</td>
<td>00002019H</td>
</tr>
<tr>
<td>EPT pointer (EPTP; full)</td>
<td>000001100B</td>
<td>0000201AH</td>
</tr>
<tr>
<td>EPT pointer (EPTP; high)</td>
<td>000001101B</td>
<td>0000201BH</td>
</tr>
<tr>
<td>EOI-exit bitmap 0 (EOI_EXIT0; full)</td>
<td>000001110B</td>
<td>0000201CH</td>
</tr>
<tr>
<td>EOI-exit bitmap 0 (EOI_EXIT0; high)</td>
<td>000001111B</td>
<td>0000201DH</td>
</tr>
<tr>
<td>EOI-exit bitmap 1 (EOI_EXIT1; full)</td>
<td>000001111B</td>
<td>0000201EH</td>
</tr>
<tr>
<td>EOI-exit bitmap 1 (EOI_EXIT1; high)</td>
<td>000001112B</td>
<td>0000201FH</td>
</tr>
<tr>
<td>EOI-exit bitmap 2 (EOI_EXIT2; full)</td>
<td>000001112B</td>
<td>00002020H</td>
</tr>
<tr>
<td>EOI-exit bitmap 2 (EOI_EXIT2; high)</td>
<td>000001113B</td>
<td>00002021H</td>
</tr>
<tr>
<td>EOI-exit bitmap 3 (EOI_EXIT3; full)</td>
<td>000001113B</td>
<td>00002022H</td>
</tr>
<tr>
<td>EOI-exit bitmap 3 (EOI_EXIT3; high)</td>
<td>000001114B</td>
<td>00002023H</td>
</tr>
<tr>
<td>EPTP-list address (full)</td>
<td>000001000B</td>
<td>00002024H</td>
</tr>
<tr>
<td>EPTP-list address (high)</td>
<td>000001001B</td>
<td>00002025H</td>
</tr>
<tr>
<td>VMREAD-bitmap address (full)</td>
<td>000001000B</td>
<td>00002026H</td>
</tr>
<tr>
<td>VMREAD-bitmap address (high)</td>
<td>000001001B</td>
<td>00002027H</td>
</tr>
<tr>
<td>VMWRITE-bitmap address (full)</td>
<td>000001000B</td>
<td>00002028H</td>
</tr>
<tr>
<td>VMWRITE-bitmap address (high)</td>
<td>000001001B</td>
<td>00002029H</td>
</tr>
<tr>
<td>Virtualization-exception information address (full)</td>
<td>000001010B</td>
<td>0000202AH</td>
</tr>
<tr>
<td>Virtualization-exception information address (high)</td>
<td>000001011B</td>
<td>0000202BH</td>
</tr>
<tr>
<td>XSS-exiting bitmap (full)</td>
<td>000001010B</td>
<td>0000202CH</td>
</tr>
<tr>
<td>XSS-exiting bitmap (high)</td>
<td>000001011B</td>
<td>0000202DH</td>
</tr>
</tbody>
</table>
FIELD ENCODING IN VMCS

B.2.2 64-Bit Read-Only Data Field

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. There is only one such 64-bit field as given in Table B-5. (As with other 64-bit fields, this one has two encodings.)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCLS-exiting bitmap (full)</td>
<td>000010111B</td>
<td>00002022EH</td>
</tr>
<tr>
<td>ENCLS-exiting bitmap (high)</td>
<td></td>
<td>0000202FH</td>
</tr>
<tr>
<td>TSC multiplier (full)</td>
<td>000011001B</td>
<td>00002032H</td>
</tr>
<tr>
<td>TSC multiplier (high)</td>
<td></td>
<td>00002033H</td>
</tr>
</tbody>
</table>

NOTES:
1. This field exists only on processors that support the 1-setting of the “use MSR bitmaps” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
6. This field exists only on processors that support the 1-setting of the “enable VM functions” VM-execution control.
7. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
8. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
9. This field exists only on processors that support the 1-setting of the “EPTP switching” VM-function control.
10. This field exists only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control.
11. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
12. This field exists only on processors that support the 1-setting of the “enable XSAVE/XRSSORS” VM-execution control.
13. This field exists only on processors that support the 1-setting of the “enable ENCLS exiting” VM-execution control.
14. This field exists only on processors that support the 1-setting of the “use TSC scaling” VM-execution control.

Table B-5. Encodings for 64-Bit Read-Only Data Field (0010_00xx_xxxx_xxxAb)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest-physical address (full)</td>
<td>000000000B</td>
<td>00002400H</td>
</tr>
<tr>
<td>Guest-physical address (high)</td>
<td></td>
<td>00002401H</td>
</tr>
</tbody>
</table>

NOTES:
1. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMCS link pointer (full)</td>
<td>000000000B</td>
<td>00002800H</td>
</tr>
<tr>
<td>VMCS link pointer (high)</td>
<td></td>
<td>00002801H</td>
</tr>
<tr>
<td>Guest IA32_DEBUGCTL (full)</td>
<td>000000001B</td>
<td>00002802H</td>
</tr>
<tr>
<td>Guest IA32_DEBUGCTL (high)</td>
<td></td>
<td>00002803H</td>
</tr>
</tbody>
</table>
B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest IA32_PAT (full)¹</td>
<td>000000010B</td>
<td>00002804H</td>
</tr>
<tr>
<td>Guest IA32_PAT (high)¹</td>
<td>000000011B</td>
<td>00002807H</td>
</tr>
<tr>
<td>Guest IA32_EFER (full)²</td>
<td>000000100B</td>
<td>00002808H</td>
</tr>
<tr>
<td>Guest IA32_EFER (high)²</td>
<td>000000101B</td>
<td>00002809H</td>
</tr>
<tr>
<td>Guest IA32_PERF_GLOBAL_CTRL (full)³</td>
<td>000000100B</td>
<td>00002808H</td>
</tr>
<tr>
<td>Guest IA32_PERF_GLOBAL_CTRL (high)³</td>
<td>000000101B</td>
<td>00002809H</td>
</tr>
<tr>
<td>Guest PDPTE0 (full)⁴</td>
<td>000000010B</td>
<td>00002804H</td>
</tr>
<tr>
<td>Guest PDPTE0 (high)⁴</td>
<td>000000011B</td>
<td>00002807H</td>
</tr>
<tr>
<td>Guest PDPTE1 (full)⁴</td>
<td>000000010B</td>
<td>00002804H</td>
</tr>
<tr>
<td>Guest PDPTE1 (high)⁴</td>
<td>000000011B</td>
<td>00002807H</td>
</tr>
<tr>
<td>Guest PDPTE2 (full)⁴</td>
<td>000000010B</td>
<td>00002804H</td>
</tr>
<tr>
<td>Guest PDPTE2 (high)⁴</td>
<td>000000011B</td>
<td>00002807H</td>
</tr>
<tr>
<td>Guest PDPTE3 (full)⁴</td>
<td>000000010B</td>
<td>00002804H</td>
</tr>
<tr>
<td>Guest PDPTE3 (high)⁴</td>
<td>000000011B</td>
<td>00002807H</td>
</tr>
<tr>
<td>Guest IA32_BNDCFGS (full)⁵</td>
<td>000000010B</td>
<td>00002804H</td>
</tr>
<tr>
<td>Guest IA32_BNDCFGS (high)⁵</td>
<td>000000011B</td>
<td>00002807H</td>
</tr>
</tbody>
</table>

NOTES:
1. This field exists only on processors that support either the 1-setting of the "load IA32_PAT" VM-entry control or that of the "save IA32_PAT" VM-exit control.
2. This field exists only on processors that support either the 1-setting of the "load IA32_EFER" VM-entry control or that of the "save IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-entry control.
4. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.
5. This field exists only on processors that support either the 1-setting of the "load IA32_BNDCFGS" VM-entry control or that of the "clear IA32_BNDCFGS" VM-exit control.

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host IA32_PAT (full)¹</td>
<td>000000000B</td>
<td>00002C00H</td>
</tr>
<tr>
<td>Host IA32_PAT (high)¹</td>
<td>000000001B</td>
<td>00002C01H</td>
</tr>
<tr>
<td>Host IA32_EFER (full)²</td>
<td>000000000B</td>
<td>00002C02H</td>
</tr>
<tr>
<td>Host IA32_EFER (high)²</td>
<td>000000001B</td>
<td>00002C03H</td>
</tr>
<tr>
<td>Host IA32_PERF_GLOBAL_CTRL (full)³</td>
<td>000000000B</td>
<td>00002C00H</td>
</tr>
<tr>
<td>Host IA32_PERF_GLOBAL_CTRL (high)³</td>
<td>000000001B</td>
<td>00002C01H</td>
</tr>
</tbody>
</table>

NOTES:
1. This field exists only on processors that support the 1-setting of the "load IA32_PAT" VM-entry control.
2. This field exists only on processors that support the 1-setting of the "load IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-exit control.

### B.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 24.11.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

#### B.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-8 enumerates the 32-bit control fields.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin-based VM-execution controls</td>
<td>000000000B</td>
<td>00004000H</td>
</tr>
<tr>
<td>Primary processor-based VM-execution controls</td>
<td>000000001B</td>
<td>00004002H</td>
</tr>
<tr>
<td>Exception bitmap</td>
<td>000000010B</td>
<td>00004004H</td>
</tr>
<tr>
<td>Page-fault error-code mask</td>
<td>000000011B</td>
<td>00004006H</td>
</tr>
<tr>
<td>Page-fault error-code match</td>
<td>000000100B</td>
<td>00004008H</td>
</tr>
<tr>
<td>CR3-target count</td>
<td>000000101B</td>
<td>0000400AH</td>
</tr>
<tr>
<td>VM-exit controls</td>
<td>000000110B</td>
<td>0000400CH</td>
</tr>
<tr>
<td>VM-exit MSR-store count</td>
<td>000000111B</td>
<td>0000400EH</td>
</tr>
<tr>
<td>VM-exit MSR-load count</td>
<td>000001000B</td>
<td>00004010H</td>
</tr>
<tr>
<td>VM-entry controls</td>
<td>000001001B</td>
<td>00004012H</td>
</tr>
<tr>
<td>VM-entry MSR-load count</td>
<td>000001010B</td>
<td>00004014H</td>
</tr>
<tr>
<td>VM-entry interruption-information field</td>
<td>000001011B</td>
<td>00004016H</td>
</tr>
<tr>
<td>VM-entry exception error code</td>
<td>000001100B</td>
<td>00004018H</td>
</tr>
<tr>
<td>VM-entry instruction length</td>
<td>000001101B</td>
<td>0000401AH</td>
</tr>
<tr>
<td>TPR threshold&lt;sup&gt;1&lt;/sup&gt;</td>
<td>000001110B</td>
<td>0000401CH</td>
</tr>
<tr>
<td>Secondary processor-based VM-execution controls&lt;sup&gt;2&lt;/sup&gt;</td>
<td>000011111b</td>
<td>0000401EH</td>
</tr>
<tr>
<td>PLE_Gap&lt;sup&gt;3&lt;/sup&gt;</td>
<td>000010000b</td>
<td>00004020H</td>
</tr>
<tr>
<td>PLE_Window&lt;sup&gt;3&lt;/sup&gt;</td>
<td>000010001b</td>
<td>00004022H</td>
</tr>
</tbody>
</table>

**NOTES:**

1. This field exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "PAUSE-loop exiting" VM-execution control.
B.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-9 enumerates the 32-bit read-only data fields.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM-instruction error</td>
<td>00000000B</td>
<td>00004400H</td>
</tr>
<tr>
<td>Exit reason</td>
<td>000000001B</td>
<td>00004402H</td>
</tr>
<tr>
<td>VM-exit interruption information</td>
<td>000000010B</td>
<td>00004404H</td>
</tr>
<tr>
<td>VM-exit interruption error code</td>
<td>000000011B</td>
<td>00004406H</td>
</tr>
<tr>
<td>IDT-vectoring information field</td>
<td>000000100B</td>
<td>00004408H</td>
</tr>
<tr>
<td>IDT-vectoring error code</td>
<td>000000101B</td>
<td>0000440AH</td>
</tr>
<tr>
<td>VM-exit instruction length</td>
<td>000000110B</td>
<td>0000440CH</td>
</tr>
<tr>
<td>VM-exit instruction information</td>
<td>000000111B</td>
<td>0000440EH</td>
</tr>
</tbody>
</table>

B.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-10 enumerates the 32-bit guest-state fields.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest ES limit</td>
<td>00000000B</td>
<td>00004800H</td>
</tr>
<tr>
<td>Guest CS limit</td>
<td>000000001B</td>
<td>00004802H</td>
</tr>
<tr>
<td>Guest SS limit</td>
<td>000000010B</td>
<td>00004804H</td>
</tr>
<tr>
<td>Guest DS limit</td>
<td>000000011B</td>
<td>00004806H</td>
</tr>
<tr>
<td>Guest FS limit</td>
<td>000000100B</td>
<td>00004808H</td>
</tr>
<tr>
<td>Guest GS limit</td>
<td>000000101B</td>
<td>0000480AH</td>
</tr>
<tr>
<td>Guest LDTR limit</td>
<td>000000110B</td>
<td>0000480CH</td>
</tr>
<tr>
<td>Guest TR limit</td>
<td>000000111B</td>
<td>0000480EH</td>
</tr>
<tr>
<td>Guest GDTR limit</td>
<td>000001000B</td>
<td>00004810H</td>
</tr>
<tr>
<td>Guest IDTR limit</td>
<td>000001001B</td>
<td>00004812H</td>
</tr>
<tr>
<td>Guest ES access rights</td>
<td>000001010B</td>
<td>00004814H</td>
</tr>
<tr>
<td>Guest CS access rights</td>
<td>000001011B</td>
<td>00004816H</td>
</tr>
<tr>
<td>Guest SS access rights</td>
<td>000001100B</td>
<td>00004818H</td>
</tr>
<tr>
<td>Guest DS access rights</td>
<td>000001101B</td>
<td>0000481AH</td>
</tr>
<tr>
<td>Guest FS access rights</td>
<td>000001110B</td>
<td>0000481CH</td>
</tr>
<tr>
<td>Guest GS access rights</td>
<td>000001111B</td>
<td>0000481EH</td>
</tr>
<tr>
<td>Guest LDTR access rights</td>
<td>000010000B</td>
<td>00004820H</td>
</tr>
<tr>
<td>Guest TR access rights</td>
<td>000010001B</td>
<td>00004822H</td>
</tr>
<tr>
<td>Guest interruptibility state</td>
<td>000010010B</td>
<td>00004824H</td>
</tr>
<tr>
<td>Guest activity state</td>
<td>000010011B</td>
<td>00004826H</td>
</tr>
</tbody>
</table>
The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

### B.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table B-11.

### B.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 24.11.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

#### B.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-12 enumerates the natural-width control fields.

### Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B) (Contd.)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest SMBASE</td>
<td>000010100B</td>
<td>0000482BH</td>
</tr>
<tr>
<td>Guest IA32_SYSENTER_CS</td>
<td>000010101B</td>
<td>0000482AH</td>
</tr>
<tr>
<td>VMX-preemption timer value(^1)</td>
<td>000010111B</td>
<td>0000482EH</td>
</tr>
</tbody>
</table>

**NOTES:**
1. This field exists only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control.

### Table B-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host IA32_SYSENTER_CS</td>
<td>000000000B</td>
<td>00004C00H</td>
</tr>
</tbody>
</table>

### Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR0 guest/host mask</td>
<td>000000000B</td>
<td>00006000H</td>
</tr>
<tr>
<td>CR4 guest/host mask</td>
<td>000000001B</td>
<td>00006002H</td>
</tr>
<tr>
<td>CR0 read shadow</td>
<td>000000010B</td>
<td>00006004H</td>
</tr>
<tr>
<td>CR4 read shadow</td>
<td>000000011B</td>
<td>00006006H</td>
</tr>
<tr>
<td>CR3-target value 0</td>
<td>000000100B</td>
<td>00006008H</td>
</tr>
<tr>
<td>CR3-target value 1</td>
<td>000000101B</td>
<td>0000600AH</td>
</tr>
<tr>
<td>CR3-target value 2</td>
<td>000000110B</td>
<td>0000600CH</td>
</tr>
<tr>
<td>CR3-target value 3(^1)</td>
<td>000000111B</td>
<td>0000600EH</td>
</tr>
</tbody>
</table>

**NOTES:**
1. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.
B.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-13 enumerates the natural-width read-only data fields.

Table B-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit qualification</td>
<td>000000000B</td>
<td>00006400H</td>
</tr>
<tr>
<td>I/O RCX</td>
<td>000000001B</td>
<td>00006402H</td>
</tr>
<tr>
<td>I/O RSI</td>
<td>000000010B</td>
<td>00006404H</td>
</tr>
<tr>
<td>I/O RDI</td>
<td>000000011B</td>
<td>00006406H</td>
</tr>
<tr>
<td>I/O RIP</td>
<td>000000100B</td>
<td>00006408H</td>
</tr>
<tr>
<td>Guest-linear address</td>
<td>000000101B</td>
<td>0000640AH</td>
</tr>
</tbody>
</table>

B.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-14 enumerates the natural-width guest-state fields.

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest CR0</td>
<td>000000000B</td>
<td>00006800H</td>
</tr>
<tr>
<td>Guest CR3</td>
<td>000000001B</td>
<td>00006802H</td>
</tr>
<tr>
<td>Guest CR4</td>
<td>000000010B</td>
<td>00006804H</td>
</tr>
<tr>
<td>Guest ES base</td>
<td>000000011B</td>
<td>00006806H</td>
</tr>
<tr>
<td>Guest CS base</td>
<td>000000100B</td>
<td>00006808H</td>
</tr>
<tr>
<td>Guest SS base</td>
<td>000000101B</td>
<td>0000680AH</td>
</tr>
<tr>
<td>Guest DS base</td>
<td>000000110B</td>
<td>0000680CH</td>
</tr>
<tr>
<td>Guest FS base</td>
<td>000000111B</td>
<td>0000680EH</td>
</tr>
<tr>
<td>Guest GS base</td>
<td>000001000B</td>
<td>00006810H</td>
</tr>
<tr>
<td>Guest LDTR base</td>
<td>000001001B</td>
<td>00006812H</td>
</tr>
<tr>
<td>Guest TR base</td>
<td>000001010B</td>
<td>00006814H</td>
</tr>
<tr>
<td>Guest GDTR base</td>
<td>000001011B</td>
<td>00006816H</td>
</tr>
<tr>
<td>Guest IDTR base</td>
<td>000001100B</td>
<td>00006818H</td>
</tr>
<tr>
<td>Guest DR7</td>
<td>000001101B</td>
<td>0000681AH</td>
</tr>
<tr>
<td>Guest RSP</td>
<td>000001110B</td>
<td>0000681CH</td>
</tr>
<tr>
<td>Guest RIP</td>
<td>000001111B</td>
<td>0000681EH</td>
</tr>
<tr>
<td>Guest RFLAGS</td>
<td>000010000B</td>
<td>00006820H</td>
</tr>
<tr>
<td>Guest pending debug exceptions</td>
<td>000010001B</td>
<td>00006822H</td>
</tr>
<tr>
<td>Guest IA32_SYSENTER_ESP</td>
<td>000010010B</td>
<td>00006824H</td>
</tr>
<tr>
<td>Guest IA32_SYSENTER_EIP</td>
<td>000010011B</td>
<td>00006826H</td>
</tr>
</tbody>
</table>

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.
B.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-15 enumerates the natural-width host-state fields.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host CR0</td>
<td>000000000B</td>
<td>00006C00H</td>
</tr>
<tr>
<td>Host CR3</td>
<td>000000001B</td>
<td>00006C02H</td>
</tr>
<tr>
<td>Host CR4</td>
<td>000000010B</td>
<td>00006C04H</td>
</tr>
<tr>
<td>Host FS base</td>
<td>000000011B</td>
<td>00006C06H</td>
</tr>
<tr>
<td>Host GS base</td>
<td>000000100B</td>
<td>00006C08H</td>
</tr>
<tr>
<td>Host TR base</td>
<td>000000101B</td>
<td>00006C0AH</td>
</tr>
<tr>
<td>Host GDTR base</td>
<td>000000110B</td>
<td>00006C0CH</td>
</tr>
<tr>
<td>Host IDTR base</td>
<td>000000111B</td>
<td>00006C0EH</td>
</tr>
<tr>
<td>Host IA32_SYSENTER_ESP</td>
<td>000001000B</td>
<td>00006C10H</td>
</tr>
<tr>
<td>Host IA32_SYSENTER_EIP</td>
<td>000001001B</td>
<td>00006C12H</td>
</tr>
<tr>
<td>Host RSP</td>
<td>000001010B</td>
<td>00006C14H</td>
</tr>
<tr>
<td>Host RIP</td>
<td>000001011B</td>
<td>00006C16H</td>
</tr>
</tbody>
</table>
Every VM exit writes a 32-bit exit reason to the VMCS (see Section 24.9.1). Certain VM-entry failures also do this (see Section 26.7). The low 16 bits of the exit-reason field form the basic exit reason which provides basic information about the cause of the VM exit or VM-entry failure.

Table C-1 lists values for basic exit reasons and explains their meaning. Entries apply to VM exits, unless otherwise noted.

<table>
<thead>
<tr>
<th>Basic Exit Reason</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0                 | Exception or non-maskable interrupt (NMI). Either:  
|                   | 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1.  
<p>|                   | 2: An NMI was delivered to the logical processor and the &quot;NMI exiting&quot; VM-execution control was 1. This case includes executions of BOUND that cause #BR, executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD2 (they cause #UD). |
| 1                 | External interrupt. An external interrupt arrived and the &quot;external-interrupt exiting&quot; VM-execution control was 1. |
| 2                 | Triple fault. The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap. |
| 3                 | INIT signal. An INIT signal arrived |
| 4                 | Start-up IPI (SIPI). A SIPI arrived while the logical processor was in the &quot;wait-for-SIPI&quot; state. |
| 5                 | I/O system-management interrupt (SMI). An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 34.15.2). |
| 6                 | Other SMI. An SMI arrived and caused an SMM VM exit (see Section 34.15.2) but not immediately after retirement of an I/O instruction. |
| 7                 | Interrupt window. At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the &quot;interrupt-window exiting&quot; VM-execution control was 1. |
| 8                 | NMI window. At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the &quot;NMI-window exiting&quot; VM-execution control was 1. |
| 9                 | Task switch. Guest software attempted a task switch. |
| 10                | CPUID. Guest software attempted to execute CPUID. |
| 11                | GETSEC. Guest software attempted to execute GETSEC. |
| 12                | HLT. Guest software attempted to execute HLT and the &quot;HLT exiting&quot; VM-execution control was 1. |
| 13                | INVD. Guest software attempted to execute INVD. |
| 14                | INVLPG. Guest software attempted to execute INVLPG and the &quot;INVLPG exiting&quot; VM-execution control was 1. |
| 15                | RDPMC. Guest software attempted to execute RDPMC and the &quot;RDPMC exiting&quot; VM-execution control was 1. |
| 16                | RDTSC. Guest software attempted to execute RDTSC and the &quot;RDTSC exiting&quot; VM-execution control was 1. |
| 17                | RSM. Guest software attempted to execute RSM in SMM. |
| 18                | VMCALL. VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 34.15.2). |
| 19                | VMCLEAR. Guest software attempted to execute VMCLEAR. |
| 20                | VMLAUNCH. Guest software attempted to execute VMLAUNCH. |
| 21                | VMPTRLD. Guest software attempted to execute VMPTRLD. |
| 22                | VMPTRST. Guest software attempted to execute VMPTRST. |
| 23                | VMREAD. Guest software attempted to execute VMREAD. |</p>
<table>
<thead>
<tr>
<th>Basic Exit Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>VMRESUME. Guest software attempted to execute VMRESUME.</td>
</tr>
<tr>
<td>25</td>
<td>VMWRITE. Guest software attempted to execute VMWRITE.</td>
</tr>
<tr>
<td>26</td>
<td>VMXOFF. Guest software attempted to execute VMXOFF.</td>
</tr>
<tr>
<td>27</td>
<td>VMXON. Guest software attempted to execute VMXON.</td>
</tr>
<tr>
<td>28</td>
<td>Control-register accesses. Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 25.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1.</td>
</tr>
<tr>
<td>29</td>
<td>MOV DR. Guest software attempted a MOV to or from a debug register and the “MOV-DR exiting” VM-execution control was 1.</td>
</tr>
</tbody>
</table>
| 30               | I/O instruction. Guest software attempted to execute an I/O instruction and either:  
1: The “use I/O bitmaps” VM-execution control was 0 and the “unconditional I/O exiting” VM-execution control was 1.  
2: The “use I/O bitmaps” VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1. |
| 31               | RDMSR. Guest software attempted to execute RDMSR and either:  
1: The “use MSR bitmaps” VM-execution control was 0.  
2: The value of RCX is neither in the range C00000000H – C0001FFFFH nor in the range C00000000H – C0001FFFFH and the nth bit in read bitmap for high MSRs is 1, where n is the value of RCX.  
3: The value of RCX was in the range 00000000H – 00001FFFFH and the nth bit in read bitmap for low MSRs is 1, where n is the value of RCX & 00001FFFFH.  
4: The value of RCX is in the range 00000000H – 00001FFFFH and the nth bit in read bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFFH. |
| 32               | WRMSR. Guest software attempted to execute WRMSR and either:  
1: The “use MSR bitmaps” VM-execution control was 0.  
2: The value of RCX is neither in the range C00000000H – C0001FFFFH nor in the range C00000000H – C0001FFFFH and the nth bit in write bitmap for low MSRs is 1, where n is the value of RCX.  
3: The value of RCX was in the range 00000000H – 00001FFFFH and the nth bit in write bitmap for low MSRs is 1, where n is the value of RCX & 00001FFFFH.  
4: The value of RCX is in the range C00000000H – C0001FFFFH and the nth bit in write bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFFH. |
<p>| 33               | VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 26.3.1. |
| 34               | VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs. See Section 26.4. |
| 36               | MWAIT. Guest software attempted to execute MWAIT and the “MWAIT exiting” VM-execution control was 1. |
| 37               | Monitor trap flag. A VM entry occurred due to the 1-setting of the “monitor trap flag” VM-execution control and injection of an MTF VM exit as part of VM entry. See Section 25.5.2. |
| 39               | MONITOR. Guest software attempted to execute MONITOR and the “MONITOR exiting” VM-execution control was 1. |
| 40               | PAUSE. Either guest software attempted to execute PAUSE and the “PAUSE exiting” VM-execution control was 1 or the “PAUSE-loop exiting” VM-execution control was 1 and guest software executed a PAUSE loop with execution time exceeding PLE_Window (see Section 25.1.3). |
| 41               | VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 26.8). |
| 43               | TPR below threshold. The logical processor determined that the value of bits 7:4 of the byte at offset 080H on the virtual-APIC page was below that of the TPR threshold VM-execution control field while the “use TPR shadow” VM-execution control was 1 either as part of TPR virtualization (Section 29.1.2) or VM entry (Section 26.6.7). |
| 44               | APIC access. Guest software attempted to access memory at a physical address on the APIC-access page and the “virtualize APIC accesses” VM-execution control was 1 (see Section 29.4). |
| 45               | Virtualized EOI. EOI virtualization was performed for a virtual interrupt whose vector indexed a bit set in the EOI-exit bitmap. |</p>
<table>
<thead>
<tr>
<th>Basic Exit Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>46 Access to GDTR or IDTR.</td>
<td>Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the “descriptor-table exiting” VM-execution control was 1.</td>
</tr>
<tr>
<td>47 Access to LDTR or TR.</td>
<td>Guest software attempted to execute LLDT, LTR, SLDT, or STR and the “descriptor-table exiting” VM-execution control was 1.</td>
</tr>
<tr>
<td>48 EPT violation.</td>
<td>An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.</td>
</tr>
<tr>
<td>49 EPT misconfiguration.</td>
<td>An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.</td>
</tr>
<tr>
<td>50 INVEPT.</td>
<td>Guest software attempted to execute INVEPT.</td>
</tr>
<tr>
<td>51 RDTSCP.</td>
<td>Guest software attempted to execute RDTSCP and the “enable RDTSCP” and “RDTSC exiting” VM-execution controls were both 1.</td>
</tr>
<tr>
<td>52 VMX-preemption timer expired.</td>
<td>The preemption timer counted down to zero.</td>
</tr>
<tr>
<td>53 INVVPID.</td>
<td>Guest software attempted to execute INVVPID.</td>
</tr>
<tr>
<td>54 WBINVD.</td>
<td>Guest software attempted to execute WBINVD and the “WBINVD exiting” VM-execution control was 1.</td>
</tr>
<tr>
<td>55 XSETPB.</td>
<td>Guest software attempted to execute XSETPB.</td>
</tr>
<tr>
<td>56 APIC write.</td>
<td>Guest software completed a write to the virtual-APIC page that must be virtualized by VMM software (see Section 29.4.3.3).</td>
</tr>
<tr>
<td>57 RDRAND.</td>
<td>Guest software attempted to execute RDRAND and the “RDRAND exiting” VM-execution control was 1.</td>
</tr>
<tr>
<td>58 INVPCID.</td>
<td>Guest software attempted to execute INVPID and the “enable INVPCID” and “INVLPG exiting” VM-execution controls were both 1.</td>
</tr>
<tr>
<td>59 VMFUNC.</td>
<td>Guest software invoked a VM function with the VMFUNC instruction and the VM function either was not enabled or generated a function-specific condition causing a VM exit.</td>
</tr>
<tr>
<td>60 ENCLS.</td>
<td>Guest software attempted to execute ENCLS and “enable ENCLS exiting” VM-execution control was 1 and either (1) EAX &lt; 63 and the corresponding bit in the ENCLS-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the ENCLS-exiting bitmap is 1.</td>
</tr>
<tr>
<td>61 RDSEED.</td>
<td>Guest software attempted to execute RDSEED and the “RDSEED exiting” VM-execution control was 1.</td>
</tr>
<tr>
<td>62 Page-modification log full.</td>
<td>The processor attempted to create a page-modification log entry and the value of the PML index was not in the range 0–511.</td>
</tr>
<tr>
<td>63 XSAVE.</td>
<td>Guest software attempted to execute XSAVE, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.</td>
</tr>
<tr>
<td>64 XRSTORS.</td>
<td>Guest software attempted to execute XRSTORS, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.</td>
</tr>
</tbody>
</table>
VMX BASIC EXIT REASONS
### Numerics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit code, mixing with 32-bit code</td>
<td>21-1</td>
</tr>
<tr>
<td>32-bit code, mixing with 16-bit code</td>
<td>21-1</td>
</tr>
<tr>
<td>32-bit physical addressing</td>
<td>3-6</td>
</tr>
<tr>
<td>Overview</td>
<td>3-6</td>
</tr>
<tr>
<td>36-bit physical addressing</td>
<td>3-6</td>
</tr>
<tr>
<td>Overview</td>
<td>3-6</td>
</tr>
<tr>
<td>64-bit mode</td>
<td></td>
</tr>
<tr>
<td>Call gates</td>
<td>5-14</td>
</tr>
<tr>
<td>Code segment descriptors</td>
<td>5-12</td>
</tr>
<tr>
<td>Control registers</td>
<td>2-13</td>
</tr>
<tr>
<td>CR0 register</td>
<td>2-13</td>
</tr>
<tr>
<td>D flag</td>
<td>5-4</td>
</tr>
<tr>
<td>Debug registers</td>
<td>2-7</td>
</tr>
<tr>
<td>Descriptors</td>
<td>5-5</td>
</tr>
<tr>
<td>DPL field</td>
<td>5-4</td>
</tr>
<tr>
<td>Exception handling</td>
<td>6-16</td>
</tr>
<tr>
<td>External interrupts</td>
<td>10-31</td>
</tr>
<tr>
<td>Fast system calls</td>
<td>5-22</td>
</tr>
<tr>
<td>GDTR register</td>
<td>2-12</td>
</tr>
<tr>
<td>GP faults, causes of</td>
<td>6-38</td>
</tr>
<tr>
<td>IDTR register</td>
<td>2-12</td>
</tr>
<tr>
<td>Initialization process</td>
<td>2-8</td>
</tr>
<tr>
<td>Interrupt and trap gates</td>
<td>6-16</td>
</tr>
<tr>
<td>Interrupt controller</td>
<td>10-31</td>
</tr>
<tr>
<td>Interrupt descriptors</td>
<td>2-5</td>
</tr>
<tr>
<td>Interrupt handling</td>
<td>6-16</td>
</tr>
<tr>
<td>Interrupt stack table</td>
<td>6-19</td>
</tr>
<tr>
<td>IRET instruction</td>
<td>6-18</td>
</tr>
<tr>
<td>L flag</td>
<td>3-12</td>
</tr>
<tr>
<td>Logical address translation</td>
<td>3-7</td>
</tr>
<tr>
<td>MOV CRn</td>
<td>2-13</td>
</tr>
<tr>
<td>MSRs</td>
<td>2-25</td>
</tr>
<tr>
<td>Null segment checking</td>
<td>5-6</td>
</tr>
<tr>
<td>Paging</td>
<td>2-6</td>
</tr>
<tr>
<td>Reading counters</td>
<td>2-25</td>
</tr>
<tr>
<td>Reading &amp; writing MSRs</td>
<td>2-25</td>
</tr>
<tr>
<td>Registers and mode changes</td>
<td>9-12</td>
</tr>
<tr>
<td>RFLAGS register</td>
<td>2-11</td>
</tr>
<tr>
<td>Segment descriptor tables</td>
<td>3-16</td>
</tr>
<tr>
<td>Segment loading instructions</td>
<td>3-9</td>
</tr>
<tr>
<td>Segments</td>
<td>5-3</td>
</tr>
<tr>
<td>Stack switching</td>
<td>5-19</td>
</tr>
<tr>
<td>SYSCALL and SYSEXIT</td>
<td>5-21</td>
</tr>
<tr>
<td>System registers</td>
<td>2-7</td>
</tr>
<tr>
<td>Task gate</td>
<td>7-16</td>
</tr>
<tr>
<td>Task priority</td>
<td>2-18</td>
</tr>
<tr>
<td>Task register</td>
<td>2-13</td>
</tr>
<tr>
<td>TSS</td>
<td></td>
</tr>
<tr>
<td>Stack pointers</td>
<td>7-17</td>
</tr>
<tr>
<td>See also: IA-32e mode, compatibility mode</td>
<td></td>
</tr>
<tr>
<td>8086</td>
<td></td>
</tr>
<tr>
<td>Emulation, support for</td>
<td>20-1</td>
</tr>
<tr>
<td>Processor, exceptions and interrupts</td>
<td>20-6</td>
</tr>
<tr>
<td>8086/8088 processor</td>
<td>22-6</td>
</tr>
<tr>
<td>8087 math coprocessor</td>
<td>22-7</td>
</tr>
<tr>
<td>82489DX</td>
<td>22-26</td>
</tr>
<tr>
<td>Local APIC and I/O APICs</td>
<td>10-4</td>
</tr>
</tbody>
</table>

### A

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A20M# signal</td>
<td>20-2</td>
</tr>
<tr>
<td>Aborts</td>
<td>22-33</td>
</tr>
<tr>
<td>Checking</td>
<td>23-4</td>
</tr>
<tr>
<td>Description of</td>
<td>6-5</td>
</tr>
<tr>
<td>Reverting a program or task after</td>
<td>6-5</td>
</tr>
<tr>
<td>AC (alignment check) flag</td>
<td>2-11</td>
</tr>
<tr>
<td>EFLAGS register</td>
<td>6-45</td>
</tr>
<tr>
<td>AC (alignment check) flag</td>
<td>22-6</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B (bus)</td>
<td>2-22</td>
</tr>
<tr>
<td>Description of</td>
<td>6-5</td>
</tr>
<tr>
<td>Reverting a program or task after</td>
<td>6-5</td>
</tr>
<tr>
<td>AC (alignment check) flag</td>
<td>2-11</td>
</tr>
<tr>
<td>EFLAGS register</td>
<td>6-45</td>
</tr>
<tr>
<td>Access rights</td>
<td>22-6</td>
</tr>
<tr>
<td>Checking</td>
<td>2-22</td>
</tr>
<tr>
<td>Checking caller privileges</td>
<td>5-26</td>
</tr>
<tr>
<td>Description of</td>
<td>5-24</td>
</tr>
<tr>
<td>Invalid values</td>
<td>22-18</td>
</tr>
<tr>
<td>ADC instruction</td>
<td>8-3</td>
</tr>
<tr>
<td>ADD instruction</td>
<td>8-3</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Size prefix</td>
<td>21-1</td>
</tr>
<tr>
<td>Space of task</td>
<td>7-14</td>
</tr>
<tr>
<td>Address translation</td>
<td></td>
</tr>
<tr>
<td>In real-address mode</td>
<td>20-2</td>
</tr>
<tr>
<td>Logical to linear</td>
<td>3-7</td>
</tr>
<tr>
<td>Overview</td>
<td>3-6</td>
</tr>
<tr>
<td>Addressing, segments</td>
<td>1-7</td>
</tr>
<tr>
<td>Advanced power management</td>
<td></td>
</tr>
<tr>
<td>C-state and Sub C-state</td>
<td>14-18</td>
</tr>
<tr>
<td>NowAIT extensions</td>
<td>14-18</td>
</tr>
<tr>
<td>See also: thermal monitoring</td>
<td></td>
</tr>
<tr>
<td>Advanced programmable interrupt controller</td>
<td></td>
</tr>
<tr>
<td>See also: I/O APIC or Local APIC</td>
<td></td>
</tr>
<tr>
<td>Alignment</td>
<td></td>
</tr>
<tr>
<td>Check exception</td>
<td>2-11</td>
</tr>
<tr>
<td>6-45, 22-11, 22-20</td>
<td></td>
</tr>
<tr>
<td>Checking</td>
<td>5-27</td>
</tr>
<tr>
<td>AM (alignment mask) flag</td>
<td>2-14</td>
</tr>
<tr>
<td>CR0 control register</td>
<td>2-14</td>
</tr>
<tr>
<td>AND instruction</td>
<td>8-3</td>
</tr>
<tr>
<td>APIC</td>
<td>10-39</td>
</tr>
<tr>
<td>10-40, 10-41</td>
<td></td>
</tr>
<tr>
<td>APIC bus</td>
<td></td>
</tr>
<tr>
<td>Arbitration mechanism and protocol</td>
<td>10-25</td>
</tr>
<tr>
<td>10-32</td>
<td></td>
</tr>
<tr>
<td>Bus message format</td>
<td>10-33</td>
</tr>
<tr>
<td>10-47</td>
<td></td>
</tr>
<tr>
<td>Diagram of</td>
<td>10-2</td>
</tr>
<tr>
<td>10-3</td>
<td></td>
</tr>
<tr>
<td>EOI message format</td>
<td>10-14</td>
</tr>
<tr>
<td>10-47</td>
<td></td>
</tr>
<tr>
<td>Nonfocused lowest priority message</td>
<td>10-49</td>
</tr>
<tr>
<td>Short message format</td>
<td>10-48</td>
</tr>
<tr>
<td>SMI message</td>
<td>34-2</td>
</tr>
<tr>
<td>Status cycles</td>
<td>10-60</td>
</tr>
<tr>
<td>Structure of</td>
<td>10-4</td>
</tr>
<tr>
<td>See also: local APIC</td>
<td></td>
</tr>
<tr>
<td>APIC flag</td>
<td>10-7</td>
</tr>
<tr>
<td>CPUID instruction</td>
<td>10-7</td>
</tr>
<tr>
<td>APIC ID</td>
<td>10-40</td>
</tr>
<tr>
<td>10-44, 10-46</td>
<td></td>
</tr>
<tr>
<td>APIC (see I/O APIC or Local APIC)</td>
<td></td>
</tr>
<tr>
<td>ARPIL instruction</td>
<td>2-22</td>
</tr>
<tr>
<td>5-27</td>
<td></td>
</tr>
<tr>
<td>Not supported in 64-bit mode</td>
<td>2-22</td>
</tr>
<tr>
<td>Atomic operations</td>
<td></td>
</tr>
<tr>
<td>Automatic bus locking</td>
<td>8-3</td>
</tr>
<tr>
<td>Effects of a locked operation on internal processor caches</td>
<td>8-5</td>
</tr>
<tr>
<td>Guaranteed, description of</td>
<td>8-2</td>
</tr>
<tr>
<td>Overview of, 8-3</td>
<td></td>
</tr>
<tr>
<td>Software-controlled bus locking</td>
<td>8-3</td>
</tr>
<tr>
<td>At-retirement</td>
<td></td>
</tr>
<tr>
<td>Counting</td>
<td>18-21</td>
</tr>
<tr>
<td>18-96</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>18-21</td>
</tr>
<tr>
<td>18-85, 18-86, 18-96</td>
<td></td>
</tr>
<tr>
<td>18-101</td>
<td></td>
</tr>
<tr>
<td>Auto HALT restart</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>34-14</td>
</tr>
<tr>
<td>SMM</td>
<td>34-13</td>
</tr>
<tr>
<td>Automatic bus locking</td>
<td>8-3</td>
</tr>
<tr>
<td>Automatic thermal monitoring mechanism</td>
<td>14-19</td>
</tr>
</tbody>
</table>

### Vol. 3D INDEX-1
Vol. 3D INDEX-3

self-modifying code, effect on, 11-18, 22-29
snooping, 11-6
store buffer, 11-20
TLBs, 11-5
UC (strong uncacheable) memory type, 11-6
UC- (uncacheable) memory type, 11-6
WB (write back) memory type, 11-7
WC (write combining) memory type, 11-7
WP (write protected) memory type, 11-7
write-back caching, 11-6
WT (write through) memory type, 11-7

Call gates:
16-bit, interlevel return from, 22-32
accessing a code segment through, 5-15
description of, 5-13
for 16-bit and 32-bit code modules, 21-1
IA-32e mode, 5-14
introduction to, 2-4
mechanism, 5-15
privilege level checking rules, 5-16

CALL instruction, 2-5, 3-9, 5-10, 5-15, 5-20, 7-2, 7-9, 7-10, 21-5
Caller access privileges, checking, 5-26

Calls
16 and 32-bit code segments, 21-3
controlling operand-size attribute, 21-5
returning from, 5-20

Capability MSR
See VMX capability MSR

Catastrophic shutdown detector
Thermal monitoring
catastrophic shutdown detector, 14-20
catastrophic shutdown detector, 14-19

CCD and CC1 (counter control) fields, CESR MSR (Pentium processor), 18-121
CD (cache disable) flag, CR0 control register, 2-14, 9-7, 11-10, 11-12, 11-13, 11-16, 11-32, 22-17, 22-18, 22-29
CESR (control and event select) MSR (Pentium processor), 18-120, 18-121
CLFLUSH feature flag, CPUID instruction, 9-8
CLFLUSH instruction, 2-15, 9-8, 11-17
CU instruction, 6-7

Clocks
counting processor clocks, 18-105
Hyper-Threading Technology, 18-105
nominal CPI, 18-105
non-halted clockticks, 18-105
non-halted CPI, 18-105
non-sleep Clockticks, 18-105
time stamp counter, 18-105

CLTS instruction, 2-22, 5-24, 25-2, 25-6
Cluster model, local APIC, 10-24
CMOVcc instructions, 22-4
CMPXCHG instruction, 8-3, 22-4
CMPXCHG instruction, 8-3, 8-4, 22-4
Code modules
16 bit vs. 32 bit, 21-1
mixing 16-bit and 32-bit code, 21-1
sharing data, mixed-size code segs, 21-3
transferring control, mixed-size code segs, 21-3

Code segments
accessing data in, 5-9
accessing through a call gate, 5-15
description of, 3-12
descriptor format, 5-2
descriptor layout, 5-2
direct calls or jumps to, 5-10
paging of, 2-6
pointer size, 21-4
privilege level checks
transferring control between code segs, 5-10

Compatibility
IA-32 architecture, 22-1
software, 1-6

Compatibility mode
code segment descriptor, 5-3
code segment descriptors, 9-12
code segment descriptors, 9-12
code segment descriptors, 9-12
control registers, 2-13
CSL and CSW, 9-12
default registers, 2-23
EFLAGS register, 2-11
exception handling, 2-5
gates, 2-4
GDTR register, 2-12, 2-13
GL (global lock) flag, 2-12
interrupt handling, 2-5
L flag, 3-12, 5-4
memory management, 2-6
operation, 9-12
segment loading instructions, 3-9
segments, 3-5
switching to, 9-12
SYSCALL and SYSRET, 5-22
SYSENTER and SYSEXIT, 5-21
system flags, 2-11
system registers, 2-7
task register, 2-12
See also: 64-bit mode, IA-32e mode

Condition code flags, x87 FPU status word
compatibility information, 22-7

Conforming code segments
accessing, 5-12
C (conforming) flag, 5-11
description of, 3-13
Context, task (see Task state)

Control registers
64-bit mode, 2-13
CR0, 2-13
CR0 (reserved), 2-13
CR1 (reserved), 2-13
CR2, 2-13
CR3 (PDBR), 2-6, 2-13
CR4, 2-13
description of, 2-13
introduction to, 2-6
VMX operation, 31-17

Coprocessor segment
overrun exception, 6-30, 22-11

Counter mask field
PerfEvtSel0 and PerfEvtSel1 MSRs (P6 family processors), 18-4, 18-119

CPL
description of, 5-7
field, CS segment selector, 5-2
CPUID instruction
availability, 22-4
counting features, 2-19
detecting features, 22-2
serializing instructions, 8-17
syntax for data, 1-8

CR0 control register, 22-7
description of, 2-13
introduction to, 2-6
state following processor reset, 9-2

CR1 control register (reserved), 2-13

CR2 control register
description of, 2-13
introduction to, 2-6

CR3 control register (PDBR)
associated with a task, 7-1, 7-3
description of, 2-13
in TSS, 7-4, 7-14
introduction to, 2-6
loading during initialization, 9-10
memory management, 2-6
page directory base address, 2-6
Error code, 16-3, 16-7, 16-10, 16-13, 16-15
architectural MCA, 16-1, 16-3, 16-7, 16-10, 16-13, 16-15
decoding IA32_MCR_STATUS, 16-1, 16-3, 16-7, 16-10, 16-13, 16-15
exception, description of, 6-14
external bus, 16-1, 16-3, 16-7, 16-10, 16-13, 16-15
memory hierarchy, 16-3, 16-7, 16-10, 16-13, 16-15
pushing on stack, 22-31
watchdog timer, 16-1, 16-3, 16-7, 16-10, 16-13, 16-15

Error numbers
VM-instruction error field, 30-29
Error signals, 22-10
Error-reporting bank registers, 15-2
ERROR#
input, 22-15
output, 22-15
ESO and ES1 (event select) fields, CESR MSR (Pentium processor), 18-121
ESR

Exception handler
calling, 6-11
defined, 6-1
flag usage by handler procedure, 6-14
machine-check exception handler, 15-27
machine-check exceptions (#MC), 15-27
machine-error logging utility, 15-27
procedures, 6-11
protection of handler procedures, 6-13
task, 6-14, 7-2

Exceptions
alignment check, 22-11
classifications, 6-4
compound error codes, 15-20
description during a task switch, 7-11
coprocessor segment overrun, 22-11
description of, 2-5, 6-1
device not available, 22-11
double fault, 6-28
derror code, 6-14
exception bitmap, 32-1
execute-disable bit, 5-32
floating-point error, 22-11
general protection, 22-11
handler mechanism, 6-11
handler procedures, 6-11
handing, 6-11
handling in real-address mode, 20-4
handling in SMCL, 3-10
handling in virtual-8086 mode, 20-11
handling through a task gate in virtual-8086 mode, 20-14
handling through a trap or interrupt gate in virtual-8086 mode, 20-12
IA-32e mode, 2-5
IDT, 6-9
initializing for protected-mode operation, 9-10
invalid-opcode, 22-5
masking debug exceptions, 6-7
masking when switching stack segments, 6-7
MCA error codes, 15-20
MMX instructions, 12-1
notation, 1-9
overview of, 6-1
priority among simultaneous exceptions and interrupts, 6-8
priority of, 22-21
priority of, x87 FPU exceptions, 22-10
reference information on all exceptions, 6-19
reference information, 64-bit mode, 6-16
restarting a task or program, 6-5
segment not present, 22-11
simple error codes, 15-20
sources of, 6-4
summary of, 6-2
vectors, 6-1

Executable, 3-11
Execute-disable bit capability
conditions for, 5-30
CPUID flag, 5-30
detecting and enabling, 5-30
exception handling, 5-32
pageFault exceptions, 6-40
protection matrix for IA-32e mode, 5-31
protection matrix for legacy modes, 5-31
reserved bit checking, 5-31

Execution events, 19-196
Exit-reason numbers
VM entries & exits, C-1
Expand-down data segment type, 3-11
Extended signature table, 9-31
extended signature table, 9-31
External bus errors, detected with machine-check architecture, 15-26

F
F2XM1 instruction, 22-13
Family 06H, 16-1
Family 0FH, 16-1
microcode update facilities, 9-28
Faults
description of, 6-5
restarting a program or task after, 6-5
FCMOVcc instructions, 22-4
FCOMI instruction, 22-4
FCOMP instruction, 22-4
FCOS instruction, 22-11
FDISI instruction (obsolete), 22-11
FDIV instruction, 22-11, 22-12
FE (fixed MTRRs enabled) flag, IA32_MTRR_DEF_TYPE MSR, 11-23
Feature
determination of, processor, 22-2
information, processor, 22-2
FENI instruction (obsolete), 22-14
FIX (fixed range registers supported) flag, IA32_MTRRCAPMSR, 11-22
Fixed-range MTRRs
description of, 11-23
Flat segmentation mode, 3-3
FLD instruction, 22-13
FLDENV instruction, 22-11
FLDL2E instruction, 22-13
FLDL2T instruction, 22-13
FLDLNZ instruction, 22-13
FLDLP2 instruction, 22-13
Floating-point error exception (#MF), 22-11
Floating-point exceptions
denormal operand exception (#D), 22-9
invalid operation (#I), 22-13
numeric overflow (#O), 22-9
numeric underflow (#U), 22-10
saved CS and EIP values, 22-10
FLUSH# pin, 6-3
FNSAVE instruction, 12-4
Focus processor, local APIC, 10-25
FORCEPR# log, 14-25, 14-29
FORCPR# interrupt enable bit, 14-27
FPATAN instruction, 22-13
FPREM instruction, 22-7, 22-11, 22-12
INDEX

FPREM1 instruction, 22-7, 22-12
FPTAN instruction, 22-7, 22-12
Front_end events, 19-196
FRSTOR instruction, 12-4, 22-11
FSAVE instruction, 12-3, 12-4
FSAVE/FNSAVE instructions, 22-11, 22-14
FSCALE instruction, 22-12
FSIN instruction, 22-12
FSINCOS instruction, 22-12
FSQRT instruction, 22-12
FSTENV instruction, 22-12
FSTENV/FNSTENV instructions, 22-12
FUCOM instruction, 22-12
FUCOMI instruction, 22-12
FUCOMPI instruction, 22-12
FUCOMPP instruction, 22-12
FXAM instruction, 6-27
FSCALE instruction, 12-3
FRSTOR instruction, 22-12
FSTENV instruction, 12-3
FSTENV/FNSTENV instructions, 22-14
FTAN instruction, 22-7
FUCOM instruction, 22-12
FUCOMI instruction, 22-4
FUCOMIP instruction, 22-4
FUCOMP instruction, 22-12
FUCOMPP instruction, 22-12
FXAM instruction, 22-13, 22-14
FXRSTOR instruction, 2-17, 2-18, 9-8, 12-3, 12-4, 13-2, 13-6
FXSAVE instruction, 2-17, 2-18, 9-8, 12-3, 12-4, 13-2, 13-6
FSX feature flag, CPUID instruction, 9-8
FXTRACT instruction, 22-9, 22-13

G
G (global) flag
- page-directory entries, 11-13
- page-table entries, 11-13
G (granularity) flag
- segment descriptor, 3-10, 3-11, 5-2, 5-4
G0-G3 (global breakpoint enable) flags
- DR7 register, 17-4
Gate descriptors
- call gates, 5-13
- description of, 5-13
- IA-32e mode, 5-14
Gates, 2-4
- IA-32e mode, 2-4
GD (general detect enable) flag
- DR7 register, 17-4, 17-9
GDT
- description of, 2-3, 3-15
- IA-32e mode, 2-4
- index field of segment selector, 3-7
- initializing, 9-10
- paging of, 2-6
- pointers to exception/interrupt handlers, 6-11
- segment descriptors in, 3-9
- selecting with TI flag of segment selector, 3-7
- task switching, 7-9
- task-gate descriptor, 7-8
- TSS descriptors, 7-5
- use in address translation, 3-6
GDTR register
- description of, 2-3, 2-6, 2-12, 3-15
- IA-32e mode, 2-4, 2-12
- limit, 5-5
- loading during initialization, 9-10
- storing, 3-15
GE (global exact breakpoint enable) flag
- DR7 register, 17-4, 17-9
General-detect exception condition, 17-9
General-protection exception (#GP), 3-12, 5-6, 5-7, 5-11, 5-12, 6-9, 6-13, 6-37, 7-5, 17-3, 22-11, 22-20, 22-33, 22-34
General-purpose registers, saved in TSS, 7-4
Global control MSRs, 15-2
Global descriptor table register (see GDTR)
Global descriptor table (see GDT)

H
HALT state
- relationship to SMI interrupt, 34-3, 34-13
Hardware reset
- description of, 9-1
- processor state after reset, 9-2
- state of MTRRs following, 11-20
- value of SMBASE following, 34-4
Hexadecimal numbers, 1-7
- high-temperature interrupt enable bit, 14-27, 14-30
HitM line, 11-6
HLT instruction, 2-24, 5-24, 6-29, 25-2, 34-13, 34-14
Hyper-Threading Technology
- architectural state of a logical processor, 8-32
- architecture description, 8-28
- caches, 8-30
data registers, 8-29
description of, 8-24, 22-3, 22-4
detecting, 8-35, 8-39, 8-40
executing multiple threads, 8-26
execution-based timing loops, 8-52
external signal compatibility, 8-31
halting logical processors, 8-50
handling interrupts, 8-26
HLT instruction, 8-46
IA32_MISC_ENABLE MSR, 8-29, 8-32
initializing IA-32 processors with, 8-25
- introduction of into the IA-32 architecture, 22-3, 22-4
- local a, 8-27
- local APIC
- functionality in logical processor, 8-28
- logical processors, identifying, 8-35
- machine check architecture, 8-28
- managing idle and blocked conditions, 8-46
- mapping resources, 8-33
- memory ordering, 8-29
- microcode update resources, 8-29, 8-32, 9-35
- MP systems, 8-26
- MTRRs, 8-28, 8-32
- multi-threading feature flag, 8-24
- multi-threading support, 8-24
- PAT, 8-28
- PAUSE instruction, 8-46, 8-47
- performance monitoring, 18-100, 18-108
- performance monitoring counters, 8-29, 8-32
- placement of locks and semaphores, 8-52
- required operating system support, 8-48
- scheduling multiple threads, 8-51
- self modifying code, 8-30
- serializing instructions, 8-29
- spin-wait loops
- PAUSE instructions in, 8-49, 8-51
- thermal monitor, 8-31
- TLBs, 8-30

I
IA-32 Intel architecture
- compatibility, 22-1
- processors, 22-1
IA32e mode
- registers and mode changes, 9-12
IA-32e mode
- call gates, 5-14
code segment descriptor, 5-3
D flag, 5-4
data structures and initialization, 9-11
- debug registers, 2-7
data structures and initialization, 9-11
debug store area
descriptors, 2-4
DPL field, 5-4
exceptions during initialization, 9-12
Interrupt gates
16-bit, interlevel return from, 22-32
clearing IF flag, 6-7, 6-14
difference between interrupt and trap gates, 6-14
for 16-bit and 32-bit code modules, 21-1
handling a virtual-8086 mode interrupt or exception through, 20-12
in IDT, 6-10
introduction to, 2-4, 2-5
layout of, 6-10
Interrupt handler
calling, 6-11
defined, 6-1
flag usage by handler procedure, 6-14
procedures, 6-11
protection of handler procedures, 6-13
task, 6-14, 7-2
Interrupts
automatic bus locking, 22-34
control transfers between 16- and 32-bit code modules, 21-6
description of, 2-5, 6-1
destination, 10-26
distribution mechanism, local APIC, 10-24
enabling and disabling, 6-6
handling, 6-11
handling in real-address mode, 20-4
handling in SMM, 34-10
handling in virtual-8086 mode, 20-11
handling multiple NMI's, 6-6
handling through a task gate in virtual-8086 mode, 20-14
handling through a trap or interrupt gate in virtual-8086 mode, 20-12
IA-32e mode, 2-5, 2-12
IDT, 6-9
IDTR, 2-12
initializing for protected-mode operation, 9-10
interrupt descriptor table register (see IDTR)
interrupt descriptor table (see IDT)
list of, 6-2, 20-6
local APIC, 10-1
maskable hardware interrupts, 2-10
masking maskable hardware interrupts, 6-6
masking when switching stack segments, 6-7
message signalled interrupts, 10-34
on-die sensors for, 14-19
overview of, 6-1
priorities among simultaneous exceptions and interrupts, 6-8
priority, 10-28
propagation delay, 22-26
real-address mode, 20-6
restarting a task or program, 6-5
software, 6-51
sources of, 10-1
summary of, 6-2
thermal monitoring, 14-19
user defined, 6-1, 6-51
valid APIC interrupts, 10-14
vectors, 6-1
virtual-8086 mode, 20-6
INTO instruction, 2-5, 3-9, 6-4, 6-24, 17-10
INTR# pin, 6-2, 6-6
Invalid opcode exception (#UD), 2-16, 6-26, 6-48, 12-1, 17-3, 22-5, 22-10, 22-19, 22-20, 34-3
Invalid TSS exception (#TS), 6-31, 7-6
Invalid-operation exception, x87 FPU, 22-11, 22-13
INV/VD instruction, 2-23, 5-24, 11-17, 22-4
INVLPG instruction, 2-23, 5-24, 22-4, 25-2, 32-3, 32-4
IOPL (I/O privilege level) field, EFLAGS register
description of, 2-10
on return from exception, interrupt handler, 6-13
sensitive instructions in virtual-8086 mode, 20-10
virtual interrupt, 2-11
IPI (see interprocessor interrupt)
IRET instruction, 3-9, 6-7, 6-13, 6-14, 6-18, 7-10, 8-17, 20-5, 20-19, 25-7
IRET instruction, 2-10, 8-17
IRR
Interrupt Request Register, 10-39, 10-41, 10-47
IRR (interrupt request register), local APIC, 10-29
ISR
In Service Register, 10-38, 10-41, 10-47
I/O
breakpoint exception conditions, 17-9
in virtual-8086 mode, 20-10
instruction restart flag
SMM revision identifier field, 34-15
instruction restart flag, SMM revision identifier field, 34-15
IO_SMI bit, 34-12
I/O permission bit map, TSS, 7-5
map base address field, TSS, 7-5
restarting following SMI interrupt, 34-15
saving I/O state, 34-12
SMM state save map, 34-12
I/O APIC, 10-26
bus arbitration, 10-25
description of, 10-1
effect of using write-through memory
external interrupts, 6-3
information about, 10-1
interrupt sources, 10-2
local APIC and I/O APIC, 10-2, 10-3
overview of, 10-1
valid interrupts, 10-14
See also: local APIC
J
JMP instruction, 2-5, 3-9, 5-10, 5-15, 7-2, 7-9, 7-10
K
KEN# pin, 11-13, 22-35
L
LO-L3 (local breakpoint enable) flags
DR7 register, 17-4
L1 (level 1) cache
caching methods, 11-6
CPUID feature flag, 11-18
description of, 11-4
effect of using write-through memory, 11-8
introduction of, 22-29
invalidating and flushing, 11-17
MESI cache protocol, 11-9
shared and adaptive mode, 11-18
L2 (level 2) cache
caching methods, 11-6
description of, 11-4
disabling, 11-17
effect of using write-through memory, 11-8
introduction of, 22-29
invalidating and flushing, 11-17
MESI cache protocol, 11-9
L3 (level 3) cache
caching methods, 11-6
description of, 11-4
disabling and enabling, 11-13, 11-17
effect of using write-through memory, 11-8
introduction of, 22-30
invalidating and flushing, 11-17
MESI cache protocol, 11-9
LAR instruction, 2-23, 5-24
Larger page sizes
introduction of, 22-30
support for, 22-18
Last branch
INDEX

LVT (see Local vector table)

M
Machine check architecture
VMX considerations, 33-11
Machine-check architecture
availability of MCA and exception, 15-18
compatibility with Pentium processor, 15-1
compound error codes, 15-20
CPUID flags, 15-18
cpuID error codes, 15-20
error-reporting bank registers, 15-2
error-reporting MSRs, 15-5
extended machine check state MSRs, 15-11
external bus errors, 15-26
first introduced, 22-21
global MSRs, 15-2
initialization of, 15-18
introduction of in IA-32 processors, 22-35
logging correctable errors, 15-28, 15-30, 15-34
machine-check exception handler, 15-27
machine-check exception (#MCE), 15-1
MSRs, 15-2
overview of MCA, 15-1
Pentium processor exception handling, 15-28
Pentium processor style error reporting, 15-12
simple error codes, 15-20
VMX considerations, 33-8, 33-9
writing machine-check software, 15-26
Mapping of shared resources, 8-33
Maskable hardware interrupts
description of, 6-3
handling with virtual interrupt mechanism, 20-15
masking, 2-10, 6-6
MCA flag, CPUID instruction, 15-18
MCE flag, CPUID instruction, 15-18
MCE (machine-check enable) flag
CR4 control register, 2-17, 22-17
MDA (message destination address)
local APIC, 10-23
Memory, 11-1
Memory management
introduction to, 2-6
overview, 3-1
paging, 3-1, 3-2
registers, 2-11
segments, 3-1, 3-2, 3-7
virtualization of, 32-2
Memory ordering
in IA-32 processors, 22-33
overview, 8-5
processor ordering, 8-5
strengthening or weakening, 8-15
write ordering, 8-5
Memory type range registers (see MTRRs)
Memory types
caching methods, defined, 11-6
choosing, 11-8
MTRR types, 11-21
selecting for Pentium III and Pentium 4 processors, 11-15
selecting for Pentium Pro and Pentium II processors, 11-14
UC (strong uncacheable), 11-6
UC (uncacheable), 11-6
WB (write back), 11-7
WC (write combining), 11-7
WP (write protected), 11-7
writing values across pages with different memory types, 11-16
WT (write through), 11-7
MemTypeGet() function, 11-29
MemTypeSet() function, 11-31
MESI cache protocol, 11-5, 11-9
Message address register, 10-34
Message data register format, 10-35
Message signalled interrupts
message address register, 10-34
message data register format, 10-34
MFENCE instruction, 2-15, 8-6, 8-15, 8-16, 8-17
Microcode update facilities
authenticating an update, 9-37
BIOS responsibilities, 9-38
calling program responsibilities, 9-39
checksum, 9-33
extended signature table, 9-31
family OFH processors, 9-28
field definitions, 9-28
format of update, 9-28
function 00H presence test, 9-42
function 01H write microcode update data, 9-43
function 02H microcode update control, 9-46
function 03H read microcode update data, 9-47
general description, 9-28
HT Technology, 9-35
INT 15H-based interface, 9-42
overview, 9-27
process description, 9-28
processor identification, 9-32
processor signature, 9-32
return codes, 9-48
update loader, 9-34
update signature and verification, 9-36
update specifications, 9-37
VMX non-root operation, 25-9, 32-8
VMX support
early loading, 32-8
late loading, 32-8
virtualization issues, 32-8
Mixing 16-bit and 32-bit code
in IA-32 processors, 22-32
overview, 21-1
MMX technology
debbugging MMX code, 12-5
effect of MMX instructions on pending x87 floating-point exceptions, 12-5
emulation of the MMX instruction set, 12-1
exceptions that can occur when executing MMX instructions, 12-1
introduction of into the IA-32 architecture, 22-2
register aliasing, 12-1
state, 12-1
state, saving and restoring, 12-3
system programming, 12-1
task or context switches, 12-4
using TS flag to control saving of MMX state, 13-7
Mode switching
effect, 12-6
example, 9-14
real-address and protected mode, 9-13
to SMM, 34-2
Model and stepping information, following processor initialization or reset
Model-specific registers (see MSRs)
Modes of operation (see Operating modes)
MONITOR instruction, 25-3
MOV instruction, 3-8, 5-8
MOV (control registers) instructions, 2-22, 5-24, 8-17, 9-13
MOV (debug registers) instructions, 2-23, 5-24, 8-17, 17-9
MOVNTDIQ instruction, 8-6, 11-17
MOVNTI instruction, 2-15, 8-6, 11-17
MOVNTPD instruction, 8-6, 11-17
MOVNTPS instruction, 8-6, 11-17
MOVNTQ instruction, 8-6, 11-17
MP (monitor coprocessor) flag
CR0 control register, 2-15, 2-16, 6-27, 9-6, 9-7, 12-1, 22-7
MSR
Vol. 3D
INDEX

Model Specific Register, 10-36, 10-37

MSRs
architectural, 35-2
description of, 9-7
introduction of in IA-32 processors, 22-35
introduction to, 2-6
list of, 35-1
machine-check architecture, 15-2
P6 family processors, 35-319
Pentium 4 processor, 35-43, 35-57, 35-172, 35-188, 35-204, 35-277, 35-301
Pentium processors, 35-327, 35-403
reading and writing, 2-19, 2-20, 2-25
reading & writing in 64-bit mode, 2-25
virtualization support, 31-14
VMX support, 31-14

MSR_TC_PRECISE_EVENT MSR, 19-196
MSR_DEBUBCTLB MSR, 17-12, 17-27, 17-36, 17-37
MSR_DEBUGCTL MSR, 17-11, 17-17, 17-23, 17-24, 17-32, 17-33, 18-4, 18-8, 18-21, 18-25, 18-31, 18-56, 18-67, 35-289
MSR_EBC_HARD_POWERON MSR, 35-114
MSR_EBC_FREQUENCY_ID MSR, 35-280
MSR_EBC_SOFT_POWERON MSR, 35-279, 35-348
MSR_IFSB_CNT0 MSR, 18-111
MSR_IFSB_CNTL6 MSR, 18-111
MSR_IFSB_DRRDY MS0, 18-110
MSR_IFSB_DRRDY1 MSR, 18-110
MSR_IFSB_IBUSQ0 MSR, 18-109
MSR_IFSB_IBUSQ1 MSR, 18-109
MSR_IFSB_ISNPOD MSR, 18-110
MSR_IFSB_ISNPO1 MSR, 18-110
MSR_LASTBRANCH_TOS MSR, 35-289
MSR_LASTBRANCH_0_TO_IP MSR, 35-300
MSR_LASTBRANCH_n MSR, 17-17, 17-34, 35-289, 35-352
MSR_LASTBRANCH_n_FROM_IP MSR, 17-16, 17-17, 17-34, 17-35, 35-298
MSR_LASTBRANCH_n_TO_IP MSR, 17-16, 17-17, 17-34, 17-35
MSR_LASTBRANCH_n_TO_LIP MSR, 35-300
MSR_LASTBRANCH_TOS MSR, 17-34
MSR_LER_FROM_LIP MSR, 17-15, 17-36, 35-288
MSR_LER_TO_LIP MSR, 17-35, 35-288
MSR_PBS_MATRIX_VERT MSR, 19-196
MSR_PBS_MATRIX_VERT MSR, 35-295, 35-381
MSRPLATFORM_BRV, 35-288, 35-384
MTRR feature flag, CPUID instruction, 11-21
MTRRcap MSR, 11-21
MTRRfix MSR, 11-23
MTRRs, 8-15
base & mask calculations, 11-26, 11-27
cache control, 11-13
description of, 9-8, 11-20
dual-core processors, 8-32
enabling caching, 9-7
feature identification, 11-21
fixed-range registers, 11-23
IA32_MTRR_CAP MSR, 11-21
IA32_MTRR_DEF_TYPE MSR, 11-22
initialization of, 11-29
introduction of in IA-32 processors, 22-35
introduction to, 2-6
large page size considerations, 11-33
logical processors, 8-32
mapping physical memory with, 11-21
memory types and their properties, 11-21
MemTypeGet() function, 11-29
MemTypeSet() function, 11-31
multiple-processor considerations, 11-32
precedence of cache controls, 11-13
precedences, 11-28
programming interface, 11-29
remapping memory types, 11-29
state of following a hardware reset, 11-20
variable-range registers, 11-23, 11-25

Multi-core technology
See multi-threading support

Multi-threading support
Multiple-processor management
bus locking, 8-3
guaranteed atomic operations, 8-2
initialization
MP protocol, 8-18
procedure, 8-53
local APIC, 10-1
memory ordering, 8-5
MP protocol, 8-18
overview of, 8-1
SMM considerations, 34-16
VMM design, 31-10
asymmetric, 31-10
CPUID emulation, 31-12
external data structures, 31-11
index-data registers, 31-11
initialization, 31-11
moving between processors, 31-11
symmetric, 31-10

Multi-processor system
local APIC and I/O APICs, Pentium 4, 10-3
local APIC and I/O APIC, P6 family, 10-3

Multisegment model, 3-4

Multitasking
initialization for, 9-10, 9-11
initializing IA-32e mode, 9-11
linking tasks, 7-12
mechanism, description of, 7-2
overview, 7-1
setting up TSS, 9-10
setting up TSS descriptor, 9-10

Multi-threading support
executing multiple threads, 8-26
handling interrupts, 8-26
logical processors per package, 8-24
mapping resources, 8-33
microcode updates, 8-32
performance monitoring counters, 8-32
programming considerations, 8-33
See also Hyper-Threading Technology and dual-core technology

MWAIT instruction, 25-3
power management extensions, 14-18

MXCSR register, 6-48, 9-8, 13-6

N
NaN, compatibility, IA-32 processors, 22-8
NE (numeric error) flag
-CR0 control register, 2-15, 6-43, 9-6, 9-7, 22-7, 22-17
NEG instruction, 8-3
NetBurst microarchitecture (see Intel NetBurst microarchitecture)

NMI interrupt, 2-24, 10-3
description of, 6-2
handling during initialization, 9-9
handling in SMM, 34-11
handling multiple NMIIs, 6-6
masking, 22-26
receiving when processor is shutdown, 6-29
reference information, 6-22
vector, 6-2

NMIP# pin, 6-2, 6-22
Nominal CPI method, 18-106
Nonconforming code segments
accessing, 5-11
C (conforming) flag, 5-11
description of, 3-13
Non-halted clockticks, 18-105
INDEX

setting up counters, 18-106
Non-Halted CPI method, 18-106
Nonmaskable interrupt (see NMI)
Non-precise event-based sampling
O (defined, 18-86
OR instruction, 18-86
used for at-retirement counting, 18-97
writing an interrupt service routine for, 17-24
Non-retirement events, 18-85, 19-173
Non-sleep clockticks, 18-105
NOT instruction, 8-3

Notation
bit and byte order, 1-6
conventions, 1-6
exceptions, 1-9
hexadecimal and binary numbers, 1-7
Instructions
operands, 1-7
reserved bits, 1-6
segmented addressing, 1-7
NT (nested task) flag
EFLAGS register, 2-10, 7-10, 7-12
Null segment selector, checking for, 5-6
Numeric overflow exception (#O), 22-9
Numeric underflow exception (#U), 22-10
NV (invert) flag, PerfEvtSel0 MSR
(P6 family processors), 18-4, 18-119
NW (not write-through) flag
CR0 control register, 2-14, 9-7, 11-12, 11-13, 11-16, 11-32, 22-17, 22-18, 22-29
NXE bit, 5-30

O
Obsolete instructions, 22-5, 22-14
OF flag, EFLAGS register, 6-24
On-die digital thermal sensor, 14-25
relevant MSRs, 14-25
sensor enumeration, 14-25
On-Demand
clock modulation enable bits, 14-23
On-demand
clock modulation duty cycle bits, 14-23
On-die sensors, 14-19
Opcodes
undefined, 22-5
Operands
instruction, 1-7
operand-size prefix, 21-1
Operating modes
64-bit mode, 2-7
compatibility mode, 2-7
IA-32e mode, 2-7, 2-8
introduction to, 2-7
protected mode, 2-7
SMM (system management mode), 2-7
transitions between, 2-8
virtual-8086 mode, 2-7
VMX operation
enabling and entering, 23-3
guest environments, 31-1
OR instruction, 8-3
OS (operating system mode) flag
PerfEvtSel0 and PerfEvtSel1 MSRs (P6 only), 18-4, 18-118
OSFXSR (FXSAVE/FXRSTOR support) flag
CR4 control register, 2-17, 9-8, 13-2
OSXMMEXCP (SIMD floating-point exception support) flag, CR4 control register, 2-18, 6-48, 9-8, 13-3
OUT instruction, 8-15, 25-2
Out-of-spec status bit, 14-25, 14-29
Out-of-spec status log, 14-26, 14-29
OUTS/OUTSB/OUTSW/OUTSD instruction, 17-9, 25-2
Overflow exception (#OF), 6-24
Overheat interrupt enable bit, 14-27, 14-30

P
P (present) flag
page-directory entry, 6-40
page-table entry, 6-40
segment descriptor, 3-11
P6 family processors
compatibility with FP software, 22-6
description of, 1-1
last branch, interrupt, and exception recording, 17-38
list of performance-monitoring events, 19-204
MSR supported by, 35-319
PAE paging
feature flag, CR4 register, 2-17
flag, CR4 control register, 3-6, 22-17, 22-18
Page attribute table (PAT)
compatibility with earlier IA-32 processors, 11-36
detecting support for, 11-34
IA32_PAT MSR, 11-34
introduction to, 11-33
memory types that can be encoded with, 11-34
MSR, 11-13
precedence of cache controls, 11-14
programming, 11-35
selecting a memory type with, 11-35
Page directories, 2-6
Page directory
base address (PDBR), 7-5
introduction to, 2-6
overview, 3-2
setting up during initialization, 9-10
Page directory pointers, 2-6
Page frame (see Page)
Page tables, 2-6
introduction to, 2-6
overview, 3-2
setting up during initialization, 9-10
Page-directory entries, 8-3, 11-5
Page-fault exception (#PF), 4-47, 6-40, 22-20
Pages
disabling protection of, 5-1
enabling protection of, 5-1
introduction to, 2-6
overview, 3-2
PG flag, CR0 control register, 5-1
split, 22-14
Page-table entries, 8-3, 11-5, 11-19
Paging
combining segment and page-level protection, 5-29
combining with segmentation, 3-5
defined, 3-1
IA-32e mode, 2-6
initializing, 9-10
introduction to, 2-6
large page size MTRR considerations, 11-33
mapping segments to pages, 4-47
page boundaries regarding TSS, 7-5
page-fault exception, 6-40, 6-50
page-level protection, 5-2, 5-3, 5-27
page-level protection flags, 5-28
virtual-8086 tasks, 20-7
Parameter
passing, between 16- and 32-bit call gates, 21-6
translation, between 16- and 32-bit code segments, 21-6
PAUSE instruction, 2-15, 25-3

Vol. 3D INDEX-13
PERFECTSEL0 and PERFECTSEL1 MSRs

PC (pin control) flag, PERFECTSEL0 and PERFECTSEL1 MSRs (P6 family processors), 18-4, 18-119

PCD pin (Pentium processor), 11-13

PCD (page-level cache disable) flag
    CR3 control register, 2-16, 11-13, 22-17, 22-29
    page-directory entries, 9-7, 11-13, 11-33
    page-table entries, 9-7, 11-13, 11-33, 22-30

PCE (performance monitoring counter enable) flag, CR4 control register, 2-17, 5-24, 18-88, 18-119

PCE (performance monitoring counter enable) flag, CR4 control register, 22-17

PDBR (see CR3 control register)

PE (protection enabled) flag, CR0 control register, 2-16, 5-1, 9-10, 9-13, 34-9

PEBS records, 17-21

PEBS (precise event-based sampling) facilities
    availability of, 18-99
    description of, 18-86, 18-99
    DS save area, 17-17
    IA-32e mode, 17-21
    PEBS buffer, 17-18, 18-19, 18-89
    PEBS records, 17-17, 17-20
    writing a PEBS interrupt service routine, 18-99
    writing interrupt service routine, 17-24

PEBS_UNAVAILABLE flag

IA32_MISC_ENABLE MSR, 17-18, 35-287

Pentium 4 processor, 1-1
    compatibility with FP software, 22-6
    last branch, interrupt, and exception recording, 17-32
    list of performance-monitoring events, 19-2, 19-173
    MSRs supported, 35-43, 35-57, 35-68, 35-83, 35-85, 35-277, 35-301
    time-stamp counter, 17-40

Pentium II processor, 1-2

Pentium III processor, 1-2

Pentium M processor
    last branch, interrupt, and exception recording, 17-37
    MSRs supported by, 35-312
    time-stamp counter, 17-40

Pentium Pro processor, 1-2

Pentium processor, 1-1, 22-6
    compatibility with MCA, 15-1
    list of performance-monitoring events, 19-213
    MSR supported by, 35-327
    performance-monitoring counters, 18-120

PERFCTR0 and PERFCTR1 MSRs

(P6 family processors), 18-118, 18-119

PERFCTSEL0 and PERFCTSEL1 MSRs

(P6 family processors), 18-118

PERFCTSEL0 and PERFCTSEL1 MSRs (P6 family processors), 18-118

Performance events
    architectural, 18-1
    Intel Core Solo and Intel Core Duo processors, 18-1
    non-architectural, 18-1
    non-retirement events (Pentium 4 processor), 19-173
    P6 family processors, 19-204
    Pentium 4 and Intel Xeon processors, 17-32
    Pentium M processors, 17-37
    Pentium processor, 19-213

Performance state, 14-1

Performance-monitoring counters
    counted events (P6 family processors), 19-204
    counted events (Pentium 4 processor), 19-2, 19-173
    counted events (Pentium processors), 18-122
    description of, 18-1, 18-2
    events that can be counted (Pentium processors), 19-213
    interrupt, 10-1
    introduction of in IA-32 processors, 22-36
    monitoring counter overflow (P6 family processors), 18-120
    overflow, monitoring (P6 family processors), 18-120
    overview of, 2-7
    P6 family processors, 18-117
    Pentium II processor, 18-117
    Pentium IV processor, 18-117
    Pentium Pro processor, 18-117
    Pentium processor, 18-120
    reading, 2-24, 18-119
    setting up (P6 family processors), 18-118
    software drivers for, 18-120
    starting and stopping, 18-119

PG (paging) flag
    CR0 control register, 2-14, 5-1
    PG (paging) flag, CR0 control register, 9-10, 9-13, 22-31, 34-9
    PGE (page global enable) flag, CR4 control register, 2-17, 11-13, 22-17, 22-18

PhysBase field, IA32_MTRR_PHYSBASEn MTRR, 11-24, 11-26

Physical address extension
    introduction to, 3-6

Physical address space
    4 GBytes, 3-6
    64 GBytes, 3-6
    addressing, 2-6
    defined, 3-1
    description of, 3-6
    guest and host spaces, 32-2
    IA-32e mode, 3-6
    mapped to a task, 7-14
    mapping with variable-range MTRRs, 11-23, 11-25
    memory virtualization, 32-2
    See also: VMX, VMX

Physical destination mode, local APIC, 10-22

PhysMask
    IA32_MTRR_PHYSMASKn MTRR, 11-24, 11-26

PMO/BPO and PM1/BP1 (performance-monitor) pins (Pentium processor), 18-121, 18-122

PML4 tables, 2-6

Pointers
    code-segment pointer size, 21-4
    limit checking, 5-25
    validation, 5-24

POP instruction, 3-8

POPF instruction, 6-7, 17-9

Power consumption
    software controlled clock, 14-19, 14-23

Precise event-based sampling (see PEBS)

PREFETCHh instruction, 2-15, 11-17

Previous task link field, TSS, 7-4, 7-12, 7-13

Privilege levels
    checking when accessing data segments, 5-8
    checking, for call gates, 5-15
    checking, when transferring program control between code segments, 5-10
    description of, 5-6
    protection rings, 5-8

Privileged instructions, 5-23

Processor families
    0GH, 16-1
    OFH, 16-1

Processor management
    initialization, 9-1
    local APIC, 10-1
    microcode update facilities, 9-27

overview of, 8-1

See also: multiple-processor management

Processor ordering, description of, 8-5

PROCHOT# log, 14-25, 14-29

PROCHOT# or FORCEPR# event bit, 14-25, 14-29

Protected mode
    IDT initialization, 9-10
    initialization for, 9-9
    mixing 16-bit and 32-bit code modules, 21-1
    mode switching, 9-13
INDEX

PE flag, CR0 register, 5-1
switching to, 5-1, 9-13
system data structures required during initialization, 9-9

Protection
combining segment & page-level, 5-29
disabling, 5-1
enabling, 5-1
flags used for page-level protection, 5-2, 5-3
flags used for segment-level protection, 5-2
IA-32e mode, 5-3
of exception, interrupt-handler procedures, 6-13
overview of, 5-1
page level, 5-1, 5-27, 5-28, 5-30
page level, overriding, 5-29
page-level protection flags, 5-28
read/write, page level, 5-28
segment level, 5-1
user/Supervisor type, 5-28

Protection rings, 5-8
PSE (page size extension) flag
CR4 control register, 2-17, 11-20, 22-17, 22-18
PSE-36 page size extension, 3-6

Pseudo-functions
VMfail, 30-2
VMfailInvalid, 30-2
VMfailValid, 30-2
VMsucceed, 30-2

Pseudo-infinity, 22-9
Pseudo-NaN, 22-9
Pseudo-zero, 22-9
P-state, 14-1

PUSH instruction, 22-6
PUSHF instruction, 6-7, 22-6

PVI (protected-mode virtual interrupts) flag
CR4 control register, 2-17, 2-17, 17-17

PwT (Pentium Processor) flag
PwT (page-level write-through) flag
CR3 control register, 2-16, 11-13, 22-17, 22-29
page-directory entries, 9-7, 11-13, 11-33
page-table entries, 9-7, 11-33, 22-30

Q
QNaN, compatibility, IA-32 processors, 22-8

R
RDPMC instruction, 2-24, 5-24, 18-87, 18-118, 18-119, 22-4, 22-17, 22-36, 25-4
in 64-bit mode, 2-25
RDTSC instruction, 2-24, 5-24, 14-41, 22-4, 25-4, 25-9
in 64-bit mode, 2-25
reading sensors, 14-25

Read/write
protection, page level, 5-28
rights, checking, 5-25

Real-address mode
8086 emulation, 20-1
address translation in, 20-2
description of, 20-1
exceptions and interrupts, 20-6
IDT initialization, 9-8
IDT, changing base and limit of, 20-5
IDT, structure of, 20-5
IDT, use of, 20-4
initialization, 9-8
instructions supported, 20-3
interrupt and exception handling, 20-4
interrupts, 20-6
introduction to, 2-7

mode switching, 9-13
native 16-bit mode, 21-1
overview of, 20-1
registers supported, 20-3
switching to, 9-14
Recursive task switching, 7-13
Related literature, 1-9
Replay events, 19-196
Requested privilege level (see RPL)
Reserved bits, 1-6, 22-2
RESET# pin, 6-3, 22-15
RESET# signal, 2-24
Resolution in degrees, 14-26
Restarting program or task, following an exception or interrupt, 6-5
Restricting addressable domain, 5-28
RET instruction, 5-10, 5-20, 21-6
Returning
from a called procedure, 5-20
from an interrupt or exception handler, 6-13
RF (resume) flag
EFLAGS register, 2-10, 6-7
RPL
description of, 3-8, 5-8
field, segment selector, 5-2
RsvdZ, 10-39
R/S# pin, 6-3
R/W (read/write) flag
page-directory entry, 5-1, 5-2, 5-28
page-table entry, 5-1, 5-2, 5-28
R/W0-R/W3 (read/write) fields
DR7 register, 17-4, 22-19

S
S (descriptor type) flag
segment descriptor, 3-11, 3-12, 5-2, 5-5
SBB instruction, 8-3

Segment descriptors
access rights, 5-24
access rights, invalid values, 22-18
automatic bus locking while updating, 8-3
base address fields, 3-10
code type, 5-2
data type, 5-2
description of, 2-4, 3-9
DPL (descriptor privilege level) field, 3-11, 5-2
d/B (default operation size/default stack pointer size and/or upper bound) flag, 3-11, 5-4
E (expansion direction) flag, 5-2, 5-4
G (granularity) flag, 3-11, 5-2, 5-4
limit field, 5-2, 5-4
loading, 22-19
P (segment-present) flag, 3-11
S (descriptor type) flag, 3-11, 3-12, 5-2, 5-5
segment limit field, 3-10
system type, 5-2
tables, 3-14
tSS descriptor, 7-5, 7-6
type field, 3-10, 3-12, 5-2, 5-5
type field, encoding, 3-14
when P (segment-present) flag is clear, 3-11

Segment limit
checking, 2-22
field, segment descriptor, 3-10
Segment not present exception (#NP), 3-11
Segment registers
description of, 3-8
IA-32e mode, 3-9
saved in TSS, 7-4
Segment selectors
Stack-fault exception (#SS)

SYSRET instruction, 2-7, 5-22, 31-15

System
- architecture, 2-1, 2-2
- data structures, 2-2
- instructions, 2-7, 2-20
- registers in IA-32e mode, 2-7
- registers, introduction to, 2-6
- segment descriptor, layout of, 5-2
- segments, paging of, 2-6

System programming
- MMX technology, 12-1
- virtualization of resources, 32-1
- System-management mode (see SMM)

T (debug trap) flag, TSS, 7-5

Task gates
descriptor, 7-8
executing a task, 7-2
handling a virtual-8086 mode interrupt or exception through, 20-14
IA-32e mode, 2-6
in IDT, 6-10
introduction for IA-32e, 2-4
introduction to, 2-4, 2-5
layout of, 6-10
referencing of TSS descriptor, 6-14

Task management, 7-1
data structures, 7-3
mechanism, description of, 7-2

Task register, 3-16
description of, 2-13, 7-1, 7-7
IA-32e mode, 2-13
initializing, 9-11
introduction to, 2-6

Task switching
description of, 7-3
exception condition, 17-10
operation, 7-10
preventing recursive task switching, 7-13
saving MMX state on, 12-4
saving SSE/SSE2/SSE3 state
on task or context switches, 13-6
T (debug trap) flag, 7-5

Tasks
- address space, 7-14
description of, 7-1
extinction-handler task, 6-11
executing, 7-2
Intel 286 processor tasks, 22-36
interrupt-handler task, 6-11
interrupts and exceptions, 6-14
linking, 7-12
logical address space, 7-15
management, 7-1
mapping linear and physical address space, 7-14
reset following an exception or interrupt, 6-5
state (context), 7-2, 7-3
structure, 7-1
switching, 7-3
task management data structures, 7-3


Thermal monitoring

INDEX
advanced power management, 14-18
automatic, 14-20
automatic thermal monitoring, 14-19
catastrophic shutdown detector, 14-19, 14-20
clock-modulation bits, 14-23
c-state, 14-18
detection of facilities, 14-24
Enhanced Intel SpeedStep Technology, 14-1
IA32_APERF MSR, 14-2
IA32_MPERF MSR, 14-1
IA32_THERM_INTERRUPT MSR, 14-25
IA32_THERM_STATUS MSR, 14-25
interrupt enable/disable flags, 14-22
interrupt mechanisms, 14-19
MWAIT extensions for, 14-18
on die sensors, 14-19, 14-24
overview of, 14-1, 14-19
performance state transitions, 14-21
sensor interrupt, 10-1
setting thermal thresholds, 14-25
software controlled clock modulation, 14-19, 14-23
status flags, 14-21
status information, 14-21, 14-22
stop clock mechanism, 14-19
thermal monitor 1 (TM1), 14-20
thermal monitor 2 (TM2), 14-20
TM flag, CPUID instruction, 14-24
Thermal status bit, 14-25, 14-28
Thermal status log bit, 14-25, 14-29
Thermal threshold #1 log bit, 14-25
Thermal threshold #1 status, 14-26, 14-29
Thermal threshold #2 log bit, 14-26, 14-29
Thermal threshold #2 status, 14-26, 14-29
THERMTRIP# interrupt enable bit, 14-27, 14-30
thread timeout indicator, 16-3, 16-7, 16-10, 16-13, 16-15
Threshold #1 interrupt enable bit, 14-27, 14-30
Threshold #1 value, 14-27, 14-30
Threshold #2 interrupt enable, 14-27, 14-30
Threshold #2 value, 14-27, 14-30
TI (table indicator) flag, segment selector, 3-7
timer, local APIC, 10-16
time-stamp counter

counting clockticks, 18-105
description of, 17-40
IA32_TIME_STAMP_COUNTER MSR, 17-40
RDTSI instruction, 17-40
reading, 2-24
software drivers for, 18-120
TSC flag, 17-40
TSD flag, 17-40
TLBs
description of, 11-1, 11-5
flushing, 11-19
invalidating (flushing), 2-23
relationship to PGE flag, 22-18
relationship to PSE flag, 11-20
virtual TLBs, 32-3
TM1 and TM2
See: thermal monitoring, 14-20
TMR
Trigger Mode Register, 10-31, 10-38, 10-41, 10-47
TMR (Trigger Mode Register), local APIC, 10-29
TPR
Task Priority Register, 10-38, 10-41
TR (trace message enable) flag
DEBUGCTLMSR MSR, 17-11, 17-33, 17-36, 17-37, 17-39
Trace cache, 11-4, 11-5
Transcendental instruction accuracy, 22-7, 22-14
Translation lookaside buffer (see TLB)
Trap gates
difference between interrupt and trap gates, 6-14
for 16-bit and 32-bit code modules, 21-1
handling a virtual-8086 mode interrupt or exception through, 20-12
in IDT, 6-10
introduction for IA-32e, 2-4
introduction to, 2-4, 2-5
layout of, 6-10
Traps
description of, 6-5
resetting a program or task after, 6-5
TS (task switched) flag
CR0 control register, 2-15, 2-22, 6-27, 12-1, 13-3, 13-7
TSD (time-stamp counter disable) flag
CR4 control register, 2-17, 5-24, 17-41, 22-17
TSS
16-bit TSS, structure of, 7-15
32-bit TSS, structure of, 7-3
64-bit mode, 7-16
CR3 control register (PDBR), 7-4, 7-14
description of, 2-4, 2-5, 7-1, 7-3
EFLAGS register, 7-4
EFLAGS. NT, 7-12
EIP, 7-4
executing a task, 7-2
floating-point save area, 22-11
format in 64-bit mode, 7-16
general-purpose registers, 7-4
IA-32e mode, 2-5
initialization for multitasking, 9-10
interrupt stack table, 7-17
invalid TSS exception, 6-31
IRET instruction, 7-12
I/O map base address field, 7-5, 22-28
I/O permission bit map, 7-5, 7-17
LDT segment selector field, 7-4, 7-14
link field, 6-14
order of reads/writes to, 22-28
pointed to by task-gate descriptor, 7-8
previous task link field, 7-4, 7-12, 7-13
privilege-level 0, 1, and 2 stacks, 5-17
referred by task gate, 6-14
segment registers, 7-4
T (debug trap) flag, 7-5
task register, 7-7
using 16-bit TSSs in a 32-bit environment, 22-28
virtual-mode extensions, 22-28
TSS descriptor
B (busy) flag, 7-5
busy flag, 7-13
initialization for multitasking, 9-10
structure of, 7-5, 7-6
TSS Segment selector
field, task-gate descriptor, 7-8
writes, 22-28
Type
checking, 5-5
field, IA32_MTRR_DEF_TYPE MSR, 11-22
field, IA32_MTRR_PHYSBASEn MTRR, 11-24, 11-26
field, segment descriptor, 3-10, 3-12, 3-14, 5-2, 5-5
of segment, 5-5

U
UC (uncacheable) memory type, 11-6
UD2 instruction, 22-4
Uncached (UC) memory type, 11-8
Uncached (UC) memory type (see Strong uncacheable (UC) memory type)
Undefined opcodes, 22-5
Unit mask field, PerfEvtSel0 and PerfEvtSel1 MSRs (P6 family processors)
, 18-3, 18-7, 18-9, 18-10, 18-11, 18-13, 18-19, 18-20, 18-42,
18-44, 18-52, 18-53, 18-71, 18-73, 18-74, 18-118
Un-normal number, 22-9
User mode
description of, 5-28
INDEX

U/S (user/supervisor) flag, 5-28
User-defined interrupts, 6-1, 6-51
USR (user mode) flag, PerfEvtSel0 and PerfEvtSel1 MSRs (P6 family processors), 18-3, 18-7, 18-9, 18-10, 18-11, 18-19, 18-20, 18-42, 18-44, 18-52, 18-53, 18-71, 18-73, 18-74, 18-118
U/S (user/supervisor) flag
page-directory entry, 5-1, 5-2, 5-28
page-table entries, 20-8
page-table entry, 5-1, 5-2, 5-28

V
V (valid) flag
IA32_MTRR_PHYSMASKn MTRR, 11-24, 11-26
Variable-range MTRRs, description of, 11-23, 11-25
VCNT (variable range registers count) field, IA32_MTRRCAPI MSR, 11-22

Vectors
exceptions, 6-1
interrupts, 6-1
VERR instruction, 2-23, 5-25
VERW instruction, 2-23, 5-25
VIF (virtual interrupt) flag
EFLAGS register, 2-11, 22-5, 22-6
VIF (virtual interrupt pending) flag
EFLAGS register, 2-11, 22-5, 22-6
Virtual memory, 2-6, 3-1, 3-2
Virtual-8086 mode
8086 emulation, 20-1
description of, 20-5
emulating 8086 operating system calls, 20-18
enabling, 20-6
entering, 20-8
exception and interrupt handling overview, 20-11
exceptions and interrupts, handling through a task gate, 20-14
exceptions and interrupts, handling through a trap or interrupt gate, 20-12
handling exceptions and interrupts through a task gate, 20-14 interrupts, 20-6
introduction to, 2-7
I/OPL sensitive instructions, 20-10
I/O-port-mapped I/O, 20-11
leaving, 20-9
memory mapped I/O, 20-11
native 16-bit mode, 21-1
overview of, 20-1
paging of virtual-8086 tasks, 20-7
protection within a virtual-8086 task, 20-8
special I/O buffers, 20-11
structure of a virtual-8086 task, 20-7
virtual I/O, 20-10
VM flag, EFLAGS register, 2-10

Virtual-8086 tasks
paging of, 20-7
protection within, 20-8
structure of, 20-7
Virtualization
debugging facilities, 32-1
interrupt vector space, 33-3
memory, 32-2
microcode update facilities, 32-8
operating modes, 32-2
page faults, 32-5
system resources, 32-1
TLBs, 32-3
VM
OSs and application software, 31-1
programming considerations, 31-1
VM entries
basic VM-entry checks, 26-2
checking guest state
control registers, 26-8
debug registers, 26-8
descriptor-table registers, 26-11
MSRs, 26-8
non-register state, 26-12
RIP and RFLAGS, 26-11
segment registers, 26-9
checks on controls, host-state area, 26-2
registers and MSRs, 26-6
segment and descriptor-table registers, 26-7
VMX control checks, 26-2
exit-reason numbers, C-1
loading guest state, 26-14
control and debug registers, MSRs, 26-15
RIP, RSP, RFLAGS, 26-16
segment & descriptor-table registers, 26-16
loading MSRs, 26-17
failure cases, 26-17
VM-entry MSR-load area, 26-17
overview of failure conditions, 26-1
overview of steps, 26-1
VMLAUNCH and VMRESUME, 26-1
See also: VMCS, VMM, VM exits

VM exits
architectural state
existing before exit, 27-1
updating state before exit, 27-1
basic VM-exit information fields, 27-4
basic exit reasons, 27-4
exit qualification, 27-4
exception bitmap, 27-1
exceptions (faults, traps, and aborts), 25-5
exit-reason numbers, C-1
external interrupts, 25-5
handling of exits due to exceptions, 31-8
IA-32 faults and VM exits, 25-1
INITS, 25-5
instructions that cause:
conditional exits, 25-2
unconditional exits, 25-2
interrupt-window exiting, 25-6
non-maskable interrupts (NMIs), 25-5
page faults, 25-5
reflecting exceptions to guest, 31-8
resuming guest after exception handling, 31-9
start-up IPIs (SIPIs), 25-5
task switches, 25-5
See also: VMCS, VMM, VM entries
VM (virtual-8086 mode) flag
EFLAGS register, 2-8, 2-10
VMCALL instruction, 30-1
VMCLEAR instruction, 30-1, 31-7
VMCS
error numbers, 30-29
field encodings, 1-6, B-1
16-bit guest-state fields, B-1
16-bit host-state fields, B-2
32-bit control fields, B-1, B-6
32-bit guest-state fields, B-7
32-bit read-only data fields, B-7
64-bit control fields, B-2
64-bit guest-state fields, B-4, B-5
natural-width control fields, B-8
natural-width guest-state fields, B-9
natural-width host-state fields, B-10
natural-width read-only data fields, B-9
format of VMCS region, 24-2
guest-state area, 24-3, 24-4
guest non-register state, 24-5
guest register state, 24-4
host-state area, 24-3, 24-8
introduction, 24-1
migrating between processors, 24-24
software access to, 24-24
INDEX

SMM
- CR4.VMxE reserved, 34-18
- overview, 34-1
- RSM instruction, 34-18
- VMCS pointer, 34-17
- VMX-critical state, 34-17
- testing for support, 23-2
- virtual TLBs, 32-3
- virtual-machine control structure (VMCS), 23-2
- virtual-machine monitor (VMM), 23-1
- virtualization of system resources, 32-1
- VM entries and exits, 23-1
- VM exits, 27-1
- VMCS pointer, 23-2
- VMM life cycle, 23-2
- VMXOFF instruction, 23-3
- VMXON instruction, 23-3
- VMXON region, 23-3
- VMXON pointer, 23-3
- See also: VMM, VMCS, VM entries, VM exits

VMXOFF instruction, 23-3, 30-1
VMXON instruction, 23-3, 30-1

WAIT/FWAIT instructions, 6-27, 22-7, 22-14, 22-15
WB (write back) memory type, 8-16, 11-7, 11-8
WB (write-back) pin (Pentium processor), 11-13
WBINVD instruction, 2-23, 5-24, 11-16, 11-17, 22-4
WB/WT# pins, 11-13
WC buffer (see Write combining (WC) buffer)
WC (write combining)
- flag, IA32_MTRRCAP MSR, 11-22
- memory type, 11-7, 11-8
WP (write protected) memory type, 11-7
WP (write protect) flag
CR0 control register, 2-15, 5-28, 22-17
Write
- hit, 11-5
Write combining (WC) buffer, 11-4, 11-7
Write-back caching, 11-6
wRMSR instruction, 2-19, 2-20, 2-25, 5-24, 8-17, 17-33, 17-38, 17-41, 18-88, 18-118, 18-119, 18-120, 22-4, 22-35, 25-9
WT (write through) memory type, 11-7, 11-8
WT# (write-through) pin (Pentium processor), 11-13

X
- x2APIC ID, 10-40, 10-41, 10-44, 10-46
- x2APIC Mode, 10-31, 10-37, 10-39, 10-40, 10-41, 10-44, 10-45, 10-46
- x87 FPU
  - compatibility with IA-32 x87 FPs and math coprocessors, 22-6
  - configuring the x87 FPU environment, 9-6
  - device-not-available exception, 6-27
  - effect of MMX instructions on pending x87 floating-point exceptions, 12-5
  - effects of MMX instructions on x87 FPU state, 12-3
  - effects of MMX, x87 FPU, FXSAVE, and FXRSTOR instructions on x87 FPU
    - tag word, 12-3
  - error signals, 22-10
  - initialization, 9-5
  - instruction synchronization, 22-15
  - register stack, aliasing with MMX registers, 12-2
  - setting up for software emulation of x87 FPU functions, 9-6
  - using TS flag to control saving of x87 FPU state, 13-7
  - x87 floating-point error exception (#MF), 6-43
- x87 FPU control word
  - compatibility, IA-32 processors, 22-8
  - x87 FPU floating-point error exception (#MF), 6-43
  - x87 FPU status word
  - condition code flags, 22-7
  - x87 FPU tag word, 22-8

XADD instruction, 8-3, 22-4
xAPIC, 10-37, 10-40
- determining lowest priority processor, 10-24
- interrupt control register, 10-21
- introduction to, 10-4
- message passing protocol on system bus, 10-32
- new features, 22-27
- spurious vector, 10-32
- using system bus, 10-4
xAPIC Mode, 10-31, 10-37, 10-41, 10-44, 10-46
XCHG instruction, 8-3, 8-16
XCR0, 2-18
XGETBV, 2-18, 2-21, 2-22
XMM registers, saving, 13-6
XOR instruction, 8-3
XSAVE, 2-18, 13-7, 13-8, 13-9, 13-10
XSETBV, 2-18, 2-19, 2-21, 2-25

Z
- ZF flag, EFLAGS register, 5-25