X86S

External Architectural Specification

Rev. 1.2
June 2024

Document Number: 351407-002

Notice: This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

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

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

Your costs and results may vary.

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

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

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

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

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

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

Contents

Figures

Tables

(This page intentionally left blank)

1 About This Document

1.1 Audience

This document is intended for software development for the X86S ISA.

To provide feedback, email x86s_feedback@intel.com.

1.2 Document Revision History

Revision History for this Document

Revision No.	Revision Description	Revision Date
1.0	Initial release	Apr 2023
1.1	Change name to X86S. Add SMX chapter. Simplify state and checks for limited segmentation. Describe VMEntry and VMExit. Add CPUIDs and MSR numbers. No fallback in ERETU. Document init and reset state and remove FIT references. Various fixes to pseudo code and descriptions. Remove 5 level switch. Cleanups to 64-bit SIPI and INIT. Clarify behavior on asize overrides. Re-add some RPL checks. Add tables for descriptor types. Fix IA32_SIPI_ENTRY_STRUCT_PTR definition. Various clarifications.	Nov 2023
1.2	64bit INIT sets SS=8 matching reset. Clarify behavior on different variants of 0x67 jumps. Remove references to segment access bits in pseudo code. Truncate RIP when entering compat mode in pseudo code. Clarify SS.B behavior. Fixes to IRET pseudo code. Document ACM header and MLE join structures. Clarify SS.B/DPL for STM and NMI blocking after SIPI. Allow hypervisors to inject #SS and #NP. CS.DPL is written on VM exit. Editorial changes. Fix EBP and CS value for ACM exit. Document host driver compatibility. Fix description of R10 vector transfer. Fix CS type for Intel64 64bit SIPI.	Jun 2024

(This page intentionally left blank)

2 Introduction

X86S is a legacy-reduced-OS ISA that removes outdated execution modes and operating system ISA.

The presence of the X86S ISA is enumerated by a single, main CPUID feature LEGACY_REDUCED_ISA in CPUID 7.1.ECX[2] which implies all the ISA removals described in this document. A new, 64-bit "start-up" interprocessor interrupt (SIPI) has a separate CPUID feature flag.

Changes in the X86S ISA consist of:

- restricting the CPU to be always in paged mode
- removing 32-bit ring 0, as well as vm86 mode
- removing ring 1 and ring 2
- removing 16-bit real and protected modes
- removing 16-bit addressing
- removing fixed MTRRs
- removing user-level I/O and string I/O
- removing CR0 Write-Through mode
- removing legacy FPU control bits in CR0
- removing ring 3 interrupt flag control
- removing the CR access instruction
- rearchitecting INIT/SIPI
- removing XAPIC and only supporting x2APIC
- removing APIC support for the 8259 interrupt controller
- removing the disabling of NX or SYSCALL or long mode in the EFER MSR
- removing the #SS and #NP exceptions
- supporting a subset of segmentation architecture
    - a subset of IDT event delivery is implemented with FRED restrictions.
    - 64-bit segmentation is applied to 32-bit compatibility mode:
        - base only for FS, GS
        - base and limit for GDT, IDT, LDT, and TSS
        - no limit on data or code fetches in 32-bit mode.
    - there are no access rights or unusable selector checking on CS, DS, ES, FS, and GS on data or code fetches in any mode.
    - there is no support for changing rings for far call, far return, and far jump (like FRED).
    - IRET can only stay in-ring or change from ring 0 to ring 3.
    - there is a reduction of segmentation state. Only a subset of the state is loaded/stored on VMX entry/exit.
    - there is reduced checking on descriptor loads.
    - the Accessed bit in a descriptor is not set.
    - the Busy bit in the TSS descriptor is not used or checked.

(This page intentionally left blank)

3 Architectural Changes

3.1 Removal of 32-Bit Ring 0

32-bit ring 0 is not supported anymore and cannot be entered.

3.2 Removal of Ring 1 and Ring 2

Ring 1 and 2 are not supported anymore and cannot be entered.

3.3 Removal of 16-Bit and 32-Bit Protected Mode

16-bit and 32-bit protected mode are not supported anymore and cannot be entered. The CPU always operates in long mode. The 32-bit submode of Intel64 (compatibility mode) still exists. An attempt to load a descriptor into CS that has CS.L==0 and CS.D==0 will generate a #GP(sel) exception.

3.4 Removal of 16-Bit Addressing and Address Size Overrides

For 32-bit compatibility mode, the 16-bit address size override prefix (0x67) triggers a #GP(0) exception when it leads to an unmasked memory reference that is not a stack access. The #GP exception takes precedence over other memory-related exceptions. Jumps follow different rules specified below.

Jumps with a 16-bit operand size prefix (0x66) that previously did truncate the RIP to 16 bits (Jump Short 0x7*, Jump Near 0x0f 8*, LOOP 0xE0-2, JECZ 0xE3, JMP near 0xE9 and 0xEB, CALL rel 0xE8, JMP near 0xFF/4, CALL indirect near 0xFF/2, RET near 0xC2-3, JMP far 0xEA and 0xFF/5, CALL indirect far 0xFF/3, CALL far 0x9A, RET far 0xCA-B, IRET) will now generate a #UD exception.

Jumps with a 0x67 prefix that previously did truncate to 16 bits (CALL indirect near mem 0xFF/2 mem, JMP far 0xFF/5, CALL indirect far 0xFF/3, JMP near indirect mem 0xFF/4) will now generate a #GP(0) exception There is no fault for operations which do not modify memory or jump, like LEA or NOPs.

SS.B is ignored in compatibility mode, and stack accesses in compatibility mode always use 32-bit operand sizes unless the instruction has an address size override which may result in a #GP(0) exception. The memory value of SS.B is still saved/restored in the VMCS.

3.5 CPUID

The LEGACY_REDUCED_OS_ISA feature bit in CPUID 7.1.ECX[2] indicates all the ISA removals described in this document.

SIPI64 in CPUID.7.1.ECX[4] indicates support for 64-bit SIPI. A processor that enumerates LEGACY_REDUCED_OS_ISA will also enumerate SIPI64.

3.6 Restricted Subset of Segmentation

X86S supports a subset of segmentation:

- No gates are supported in the GDT/LDT; it only supports data segments, code segments, LDTs, and TSSs (in the GDT).

- Bases are supported for FS, GS, GDT, IDT, LDT, and TSS registers; the base for CS, DS, ES, and SS is ignored for 32-bit mode, the same as 64-bit mode (treated as zero). The processor does not save the state of CS, DS, ES, SS Base. It is neither saved nor restored on VMENTRY/VMEXIT or SMI/RSM.

- Limits are supported for GDT, IDT, LDT, and TSS; the limit for CS, DS, ES, FS, GS, and SS is treated as infinite. The processor does not save the state of CS, DS, ES, FS, GS, and SS.

- The Limit field is neither saved nor restored on VMENTRY/VMEXIT.

  CS and SS are the only descriptors having access rights. For these only CS.L and SS.DPL fields exist at runtime; however, some of the other bits may be checked at initial descriptor load. All other descriptors' access rights (except SS.B) are neither saved nor restored on VMENTRY/VMEXIT. The CPL of the core is always SS.DPL. The access rights are checked on a descriptor load to check the type and DPL, and to create the limit (if applicable) and D (if applicable).

- Expand down, conforming, and unusable segment types are not supported – they are ignored and revert to the base type. What used to be a conforming code segment is now treated as a code segment. Data and code segments are always readable and writable.

- The descriptor.DPL field must be 0 or 3, and the selector RPL must match DPL (except for data segments or for exception entry).

- On loads/stores, R/W access rights and NULL are ignored.

- IRET can switch rings from 0 to 3, or stay within a ring, but cannot cause a task switch nor enter VM86 mode.

- Descriptor accessed bits will not be set in memory, but appear to be set when accessed through the LAR instruction.

- The TSS busy bit is not supported. It is not set in memory by LTR, or checked on VMENTRY.

- #SS exceptions are removed and will signal a #GP instead. However #SS can still be injected when entering a guest.

- #NP exceptions are removed and will signal a #GP instead. However #NP can still be injected when entering a guest.

- The LMSW instruction is removed and will signal a #UD exception.

The three operating modes shown in Table 1 are supported.

Table 1. Supported Operating Modes

	CPL=0	**CPL=3**
LMA=1 CS.L=0	Unsupported	Ring 3 32-bit compatibility mode
LMA=1 CS.L=1	Ring 0 64-bit mode	Ring 3 64-bit mode

3.7 New Checks When Loading Segment Registers

When loading segment registers through a method other than VM Entry/Exit, the following conditions are checked:

- The CS and SS.DPL field must be either 0 or 3.
- In general, DPL must equal CPL for CS/SS. The exception is the CS descriptor popped off the stack for IRET and ERETU. For data segments other than SS, DPL is ignored.
- Code descriptors must be code type, and not be 16-bit in any ring, or 32-bit when DPL==0.
- Data descriptors must not be system type.

A #GP(sel) exception is signaled if these conditions are not met. For non-system segments, limits and bases are ignored.

Pseudocode for the modified instructions can be found in Chapter 4.

3.7.1 Code and Data Segment Types

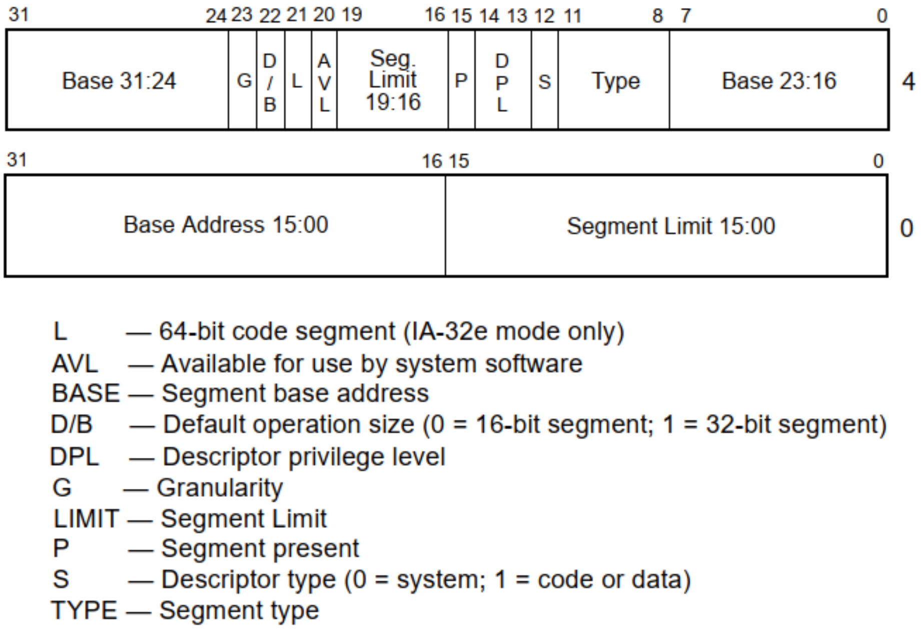
31
24 23 22 21 20 19
16 15 14 13 12 11
8 7
0
Base 31:24
G
D / B
L
A V L
Seg. Limit 19:16
P
D P L
S
Type
Base 23:16
4
31
16 15
0
Base Address 15:00
Segment Limit 15:00
0
L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Figure 1. Segment Descriptor Layout

Bits[10:8] in the descriptor, shown in Figure 1, are ignored for code and data segment types. There are now two types in the group: data segments, and code segments, with 8 encodings each. Code segments can be executed; both data and code segments can be read and written. Table 2 shows the encodings for the Type field.

Table 2. Code/Data Segment Types

Type Field	11	10	9	8	Type	Behavior Load	Behavior Use
0	0	0	0	0	Data	Load for data	Read / Write
1	0	0	0	1	Data	Load for data	Read / Write
2	0	0	1	0	Data	Load for data	Read / Write
3	0	0	1	1	Data	Load for data	Read / Write
4	0	1	0	0	Data	Load for data	Read / Write
5	0	1	0	1	Data	Load for data	Read / Write
6	0	1	1	0	Data	Load for data	Read / Write
7	0	1	1	1	Data	Load for data	Read / Write
8	1	0	0	0	Code	Load for code/data	Execute / Read / Write
9	1	0	0	1	Code	Load for code/data	Execute / Read / Write
10	1	0	1	0	Code	Load for code/data	Execute / Read / Write
11	1	0	1	1	Code	Load for code/data	Execute / Read / Write
12	1	1	0	0	Code	Load for code/data	Execute / Read / Write
13	1	1	0	1	Code	Load for code/data	Execute / Read / Write
14	1	1	1	0	Code	Load for code/data	Execute / Read / Write
15	1	1	1	1	Code	Load for code/data	Execute / Read / Write

3.7.2 System Segment Types (S=0)

Bit[9] (BUSY) for 64-bit TSS is ignored. X86S does not differentiate between busy and available TSS. Both encodings are treated in the same manner.

64-bit call gate has been removed.

There are now four types in this group: LDT, interrupt gate, trap gate (with one encoding), and TSS (with two encodings). Table 3 shows the encodings for the Type field.

Table 3. System Segment Types

Type Field	Description	CR4.FRED=0	CR4.FRED=1
0	Reserved.	#GP	#GP
1	16-bit TSS.	#GP	#GP
2	LDT.	Load with LLDT or #GP	Load with LLDT or #GP
3	Busy 16-bit TSS.	#GP	#GP
4	16-bit call gate.	#GP	#GP
5	Task gate.	#GP	#GP
6	16-bit interrupt gate.	#GP	#GP
7	16-bit trap gate.	#GP	#GP
8	Reserved.	#GP	#GP
9	Available TSS.	Load with LTR or #GP	Load with LTR or #GP

Type Field	Description	CR4.FRED=0	CR4.FRED=1
10	Reserved.	#GP	#GP
11	Busy 32-bit TSS.	Load in LTR or #GP	Load with LTR or #GP
12	32-bit call gate.	#GP	#GP
13	Reserved.	#GP	#GP
14	Interrupt gate.	Follow in IDT or #GP	#GP
15	Trap gate.	Follow in IDT or #GP	#GP

3.8 Removal of #SS and #NP Exceptions

Any faulting stack segment references, both explicit and implicit, do not cause #SS exceptions anymore. Instead, #GP exceptions will be generated. All descriptor loads with desc.P==0 will generate a #GP(sel) exception.

3.9 Fixed Mode Bits

The CPU is always running in the 64-bit submode of Intel64. Real mode, protected mode, or VM86 modes cannot be enabled.

3.9.1 Fixed CR0 Bits

All bits in the CR0 register, shown in Figure 2, except for the TS, WP, AM, and CD bits, are fixed. ET is fixed to 1 but ignored on input. Writing a different value than the fixed value to the PE, MP, EM, NE, NW, or PG bits will produce a #GP(0) exception, but only after causing a VM exit if CR0 exiting is configured. Reading will always return the fixed value with the current value of the flexible bits, unless changed by a VM exit from CR0 exiting.

The CR0 control bits are defined in Table 4.

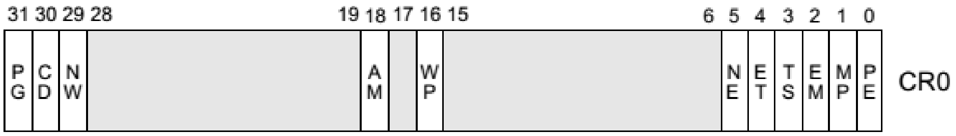

Figure 2. CR0 Layout

Table 4. CR0 Control Bits

CR0 Bit	Fixed Value	Bit	Implication
PE	1	0	Protection enable: always in protected mode.
MP	1	1	Monitor coprocessor: always enabled.
EM	0	2	FP emulation.
TS	-	3	Task switch. Disable FPU. This bit is still flexible.
ET	1	4	Extension type (ignored on input).
NE	1	5	Numeric error.

CR0 Bit	Fixed Value	Bit	Implication
WP	-	16	Write protect page tables. This bit is still flexible.
AM	-	18	Enable alignment checks with RFLAGS.AC. This bit is still flexible.
NW	0	29	Write-through. Always disabled.
CD	-	30	Cache disable. This bit is still flexible.
PG	1	31	Paging is always enabled.

3.9.2 Fixed CR4 Bits

The bits listed in Table 5 are fixed in the CR4 register, shown in Figure 3. Assuming the CR4 write is not intercepted, writing any other value will result in a #GP(0) exception. This does not apply to the VME bit.

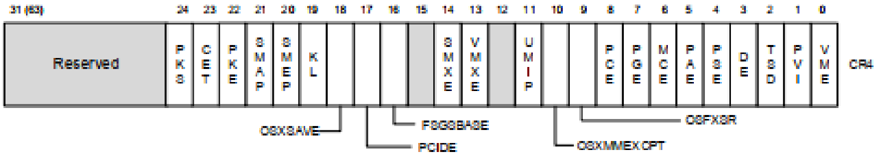

Figure 3. CR4 Layout

Table 5. CR4 Fixed Bits

CR4 Bit	Fixed Value	Bit	Implication
PVI	0	1	No support for protected mode virtual interrupts.
PAE	1	5	8-byte PTEs. Always enabled in 64-bit mode.

3.9.3 Fixed EFER Bits

The bits listed in Table 6 are fixed in the EFER MSR. Writing other values to EFER will produce a #GP exception, except for LMA, which is ignored.

Table 6. Fixed EFER Bits

EFER Bit	Fixed Value	Bit	Implication
SCE	1	0	Syscall is always enabled.
LME	1	8	Always in long mode.
LMA	1	10	Always in long mode, but changes ignored.
NXE	1	11	NX bit for page tables is always enabled.

3.9.4 Removed RFLAGS

RFLAGS Register shows the bits in the RFLAGS register. The IOPL, VM, VIF, and VIP bits are always zero. The rules in Table 7 apply.

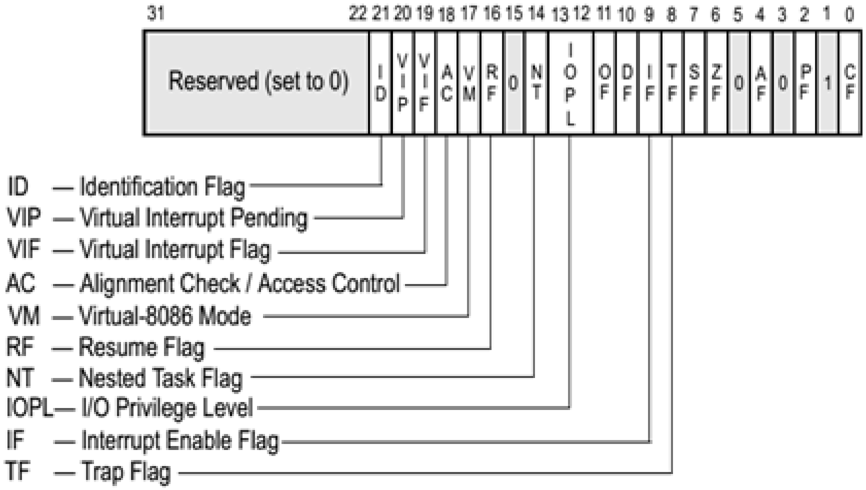

Figure 4. RFLAGS Register

Table 7. Behavior of Removed RFLAGS

Action	Action on newIOPL != 0	Action on newVIF != 0 or newVIP != 0	Action on newVM!=0
POPF CPL3	Ignored	Ignored	Ignored
POPF CPL0	Ignored	Ignored	Ignored
SYSRET	#GP(0)	#GP(0)	N/A (always cleared)
IRET CPL3->CPL3 (CS.DPL=3)	Ignored	Ignored	Ignored
IRET CPL0->CPL0 (CS.DPL=0)	#GP(0)	#GP(0)	Ignored
IRET CPL3->CPU3 (CS.DPL=0)	Ignored	Ignored	Ignored
IRET CPL0->CPL3 (CS.DPL=3)	Ignored	Ignored	Ignored
ERETU	#GP(0)	#GP(0)	#GP(0)
ERETS	#GP(0)	#GP(0)	#GP(0)
VMEntry	Bad Guest State error	Bad Guest State error	Bad Guest State error
SEAMRET	Bad Guest State error	Bad Guest State error	Bad Guest State error
RSM	Forced to 0	Forced to 0	Forced to 0

3.9.5 Removed Status Register Instruction

The LMSW instruction is removed and will result in a #UD fault.

3.9.6 Removal of Ring 3 I/O Port Instructions

There is no concept of user mode I/O port accesses anymore, and using INB/INW/INL/INQ/OUTB/OUTW/OUTL/OUTQ in ring 3 always leads to a #GP(0) exception. The #GP check will be before VM execution or I/O permission bitmap checks. This implies there will be no loads from the I/O permission bitmap.

3.9.7 Removal of String I/O

INS/OUTS are not supported and will result in a #UD exception. This includes the REP variants of the INS/OUTS instructions as well.

3.10 64-Bit SIPI

64-bit SIPI defines an architectural package scope IA32_SIPI_ENTRY_STRUCT_PTR MSR that contains a physical pointer to an entry structure in memory. The entry structure defines the state for entering application processors in 64-bit paged mode.

To trigger 64-bit SIPI, set the enable bit in the IA32_SIPI_ENTRY_STRUCT_PTR MSR, as well in the features field of the memory entry struct, then trigger SIPI using the X2APIC ICR register.

Legacy SIPI is not supported.

The presence of 64-bit SIPI is enumerated by the CPUID.7.1.ECX[4] SIPI64 CPUID feature bit.

3.10.1 IA32_SIPI_ENTRY_STRUCT_PTR

The IA32_SIPI_ENTRY_STRUCT_PTR (0x3C) package scope MSR, shown in Table 8, defines the execution context of the target CPU after receiving a SIPI message. It points to an entry structure in memory.

The MSR is read only after the BIOS_DONE MSR bit is set.

Table 8. IA32_SIPI_ENTRY_STRUCT_PTR MSR (Address 0x3C)

Bits	Field	Attr	Reset Value	Description
63:MAXPA	Reserved	NA	0	-
MAXPA-1:12	SIPI_ENTRY_STRUCT_PTR	RW	0	Bits [MAXPA-1:12] of physical pointer to SIPI_ENTRY_STRUCT.
11:1	Reserved	NA	0	-
0	ENABLED	RW	0	Enable 64-bit SIPI.

After SIPI, NMIs are blocked until explicitly unblocked by ERETS clearing the NMI blocking bit or IRET. On BSP reset, NMIs are not blocked.

On receiving a SIPI, the target CPU loads the register state from the entry struct and starts executing at the specified RIP. The vector from the SIPI message is delivered in R10. The vector delivered in the vector field of the INIT IPI message is ignored.

3.10.2 The SIPI_ENTRY_STRUCT Definition

The entry struct memory table, shown in Table 9, defines the execution context of a CPU receiving a SIPI.

Table 9. SIPI_ENTRY_STRUCT Structure in Memory

Offset (bits)	Size (bits)	Name	Description
0	8	FEATURES	Bit[0] - enable bit (0 - shutdown). Other bits are reserved.
8	8	RIP	New instruction pointer to execute after SIPI. Valid values depend on new CR4.
16	8	CR3	New CR3 value. Must be consistent with new CR4.PCIDE and no reserved bits set.
24	8	CR0	New CR0 value. Non-flexible bits must match fixed values and no reserved bits set.
32	8	CR4	New CR4 value. Non-flexible bits must match fixed value. Must be consistent with new CR3, new RIP, new CR0 and no reserved bits set.

Any consistency check failures on SIPI_ENTRY_STRUCT fields lead to a shutdown on the target CPU.

3.10.3 Pseudocode on Receiving INIT When Not Blocked

```
IF in guest mode THEN
    Trigger exit
FI
RFLAGS = 2 # clear all modifiable bits in RFLAGS
Set CR0 to PE=1, MP=1, ET=1, NE=1, NW=0, PG=1, preserve CR0.CD
Set CR4 to PAE=1
Clear CR3
Clear CR2
Set CS to Selector = 0, CS.L = 1
Set DS, ES Selector = 0
Set SS Selector = 8
Set FS, GS to Selector = 0, Base = 0
Set GDTR/IDTR to Base = 0, Limit = 0xffff
Set LDTR, TR to Selector = 0, Base = 0, Limit = 0xffff,
Set FS/GS BASE MSR to 0
Set EFER to LMA=1, LME=1, NX=1, SC=1 // only relevant for Intel64
Set RDX to 0x000n06xxx, where n is extended model value and x is a stepping number
Clear all other GPRs
Clear DR0/DR1/DR2/DR3
Set DR6 to 0xffff0ff0
Set DR7 to 0x400
Set x87 FPU control word to 0x37f
Set x87 FPU status word to 0
Set x87 FPU tag word to 0xffff
Flush all TLBs
```

```
IF IA32_APICBASE.BSP = 1 THEN
    Force 64bit supervisor mode as in reset
    Execute 64bit reset vector using CR3/RIP value from reset
ELSE
   Enter wait for SIPI state
FI
```

3.10.4 Pseudocode on Receiving SIPI

```
IF IA32_SIPI_ENTRY_STRUCT_PTR.ENABLED = 0 THEN
   Shutdown // On non X86S fall back to legacy SIPI
FI
// following memory reads are done physically with normal ring 0 rights honoring range registers and allowing MKTME keys but not TDX
ENTRY_STRUCT = IA32_SIPI_ENTRY_STRUCT_PTR[12:MAXPA]
IF ENTRY_STRUCT->FEATURES != 1 THEN
   Non triple fault Shutdown // On non X86S fall back to legacy SIPI
FI
# note the order of these checks is not defined
newCR4 = ENTRY_STRUCT->CR4 # read entry_struct.CR4
newRIP = ENTRY_STRUCT->RIP # read entry_struct.RIP
newCR0 = ENTRY_STRUCT->CR0 # read entry_struct.CR0
newCR3 = ENTRY_STRUCT->CR3 # read entry_struct.CR3
IF newCR4.PVI != 0 OR
  OR newCR4.PAE != 1 OR
  newCR4 has reserved bits set OR  // follows same rules as MOV CR4
  newCR0.PE != 1 OR
  newCR0.MP != 1 OR
  newCR0.EM != 0 OR
  newCR0.NE != 1 OR
  newCR0.NW != 0 OR
  newCR0.PG != 1 OR
  newCR3 has reserved bits set OR // follows same rules as MOV CR3
  newRIP is not canonical depending on newCR4.LA57 THEN
   Unbreakable Shutdown
FI
IF LEGACY_REDUCED_OS_ISA CPUID is clear THEN
  // initialize state to be equivalent to X86S
  CS = Selector=0, Base=0, Limit=0xffffff, AR=Present, R/W, DPL=0, Type=11, S=1, G=1, L=1
  SS/ES/FS/GS/DS = Selector = 0, Base = 0, Limit = 0xffffff, AR = Present, R/W, DPL=0, Type=3, S=1, G=1
  EFER = LMA=1, LME=1, SC=1, NX=1
  GDTR/TR.limit = 0
```

FI

newCR0.ET = 1

CR4 = newCR4 ; CR3 = newCR3 ; CR0 = newCR0

Move received SIPI vector zero extended to R10

NMIs are blocked

RIP = newRIP

3.11 64-Bit Reset

The CPU starts executing in 64-bit paged mode with a 4-level page table after reset. No Firmware Interface Table (FIT) is necessary, as the X86S reset state has a fixed RIP and CR3. The fixed reset RIP is the standard reset vector 0xFFFFFFF0 but is entered as 64-bit. The fixed reset CR3 value is 0xFFFFE000.

Reset register states are shown in Table 10.

Table 10. Reset Register States

Register	Intel64 Reset	X86S Reset	X86S INIT
EFLAGS	00000002H	00000002H	00000002H
RIP/EIP	0000FFF0H	FFFFFFF0H	FFFFFFF0H
CR0	60000010H	80000033H	80000033H
CR2	00000000H	00000000H	00000000H
CR3	00000000H	FFFFE000H	FFFFE000H
CR4	00000000H	00000020H	00000020H

CS	Selector=F000H
	Base=FFFF0000H
	Limit=FFFFH
	AR=Present, R/W, Accessed, Type=3
		Selector=0H
		Base=n/a
		Limit=n/a
		AR=L=1
			Selector=0H
			Base=n/a
			Limit=n/a
			AR=L=1

SS	Selector=F000H
	Base=FFFF0000H
	Limit=FFFFH
	AR=Present, R/W, Accessed, Type=3
		Selector= 8
		Base= n/a
		Limit= n/a
		AR=DPL=0, B=0, rest n/a
			Selector= 8
			Base= n/a
			Limit= n/a
			AR=DPL=0, B=0, rest n/a

DS,ES	Selector=0000H
	Base=00000000H
	Limit=FFFFH
	AR=Present, R/W, Accessed, P=1,S=1 Type=3
		Selector= 0
		Base= n/a
		Limit= n/a
		AR=n/a
			Selector= 0
			Base= n/a
			Limit= n/a
			AR=n/a

FS,GS	Selector=0000H
	Base=00000000H
	Limit=FFFFH
	AR=Present, R/W, Accessed, P=1,S=1 Type=3
		Selector= 0
		Base= 00000000H
		Limit= n/a
		AR= n/a
			Selector= 0
			Base= 00000000H
			Limit= n/a
			AR= n/a

EFER	0	LMA=1,LME1=,SC=1,NX=1	LMA=1,LME1=,SC=1,NX=1

Register	Intel64 Reset	X86S Reset	X86S INIT
LDT	Base=0, Limit=0, P=0	Base=0, Limit=0	Base=0, Limit=0
TR	Base=0, Limit=0xffff	Base=0, Limit=0	Base=0, Limit=0
IDTR	Base=0,Limit=0xffff	Base=0, Limit=0	Base=0, Limit=0
GDTR	Base=0,Limit=0xffff	Base=0, Limit=0	Base=0, Limit=0
RDX	0x000nFFxx	0x000nFFxx	0x000nFFxx
RAX	0 if BIST passed	0 if BIST passed	0
RBX, RCX, RDI, RSI, RSP, RBP, R8-R15	0	0	0
ST0-ST7	+0.0	+0.0	unchanged
X87 FPU control word	0x40	0x40	0x37f
X87 FPU status word	0	0	0
X87 FPU	0x5555	0x5555	0xffff
MM0-MM7, YMM0-YMM15	0	0	unchanged
MXCSR	0x1f80	0x1f80	unchanged
DR0-DR3	0	0	0
DR6	0xffff0ff0	0xffff0ff0	0xffff0ff0
DR7	0x400	0x400	0x400
XCR0	1	1	unchanged
IA32_XSS	0	0	unchanged
OPMASK	0	0	unchanged
PKRU	0	0	unchanged
Intel PT MSRs	0	0 (for cold reset)	unchanged
TSC	0	0 (for cold reset)	unchanged
TSC_AUX/TSC_ADJUST/TSC_DEADLINE	0	0	unchanged
SYSENTER_CS/ESP/EIP	0	0	unchanged
STAR/LSTAR	0	0	unchanged
PMXc, PERFEVTSELx	0	0	unchanged
PERF_GLOBAL_CTRL	Set bits for all counters	Set bits for all counters	0
FIXED_CTRx, FIXED_CTR_CTRL	0	0	unchanged
TLBs	-	-	unchanged

Register	Intel64 Reset	X86S Reset	X86S INIT
MTRRs	Disabled	Disabled	unchanged
Machine check banks	Undefined	Undefined	unchanged
LBRs	0	0 (for cold reset)	unchanged

3.12 Removal of Fixed MTRRs

There is no support for fixed MTRRs. The FIX bit, bit[8] in the IA32_MTRRCAP register, is cleared and all the MTRR_FIX_* MSRs are not implemented. MTRR_DEF_TYPE bit[10] is reserved.

Table 11 lists the fixed MTRR MSRs removed.

Table 11. Removed MTRR Registers

Name
IA32_MTRR_FIX64_00000
IA32_MTRR_FIX16_80000
IA32_MTRR_FIX16_a0000
IA32_MTRR_FIX4_c0000
IA32_MTRR_FIX4_c8000
IA32_MTRR_FIX4_d0000
IA32_MTRR_FIX4_d8000
IA32_MTRR_FIX4_e0000
IA32_MTRR_FIX4_e8000
IA32_MTRR_FIX4_f0000
IA32_MTRR_FIX4_f8000

3.13 Removal of XAPIC and ExtInt

The only way to access the X2APIC is through MSR accesses. Virtual XAPIC through VMX is still supported.

The CPU is always in x2APIC mode (IA32_APIC_BASE[EXTD] is 1) and is enabled. Attempts to write IA32_APIC_BASE to disable the APIC or leave x2APIC mode will cause a #GP(0) exception.

This is enumerated to software through the IA32_XAPIC_DISABLE_STATUS[LEGACY_XAPIC_DISABLED] MSR bit being 1 and IA32_ARCH_CAPABILITIES[21] MSR being 1.

For more details, see https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/cpuid-enumeration-and-architectural-msrs.html

The ExtINT decoding in the local APIC is removed.

3.14 Virtualization Changes

This section describes changes to the virtualization state.

“Fixed” fields are consistency checked and VM entry will fail if they do not match the fixed value.

3.14.1 VMCS Guest State

Guest VMCS field changes are listed in Table 12.

For VMEntry, consistency checks on the removed segmentation state do not occur.

Table 12. VMCS Fields Changed (Guest State)

VMCS Field	INDEX	Change	Reason
WFS encoding in Guest activity state		Fixed 0	64-bit SIPI does not support
Guest ES Limit	0x00004800	Ignored	Reduced Segmentation State
Guest CS Limit	0x00004802	Ignored	Reduced Segmentation State
Guest SS Limit	0x00004804	Ignored	Reduced Segmentation State
Guest DS Limit	0x00004806	Ignored	Reduced Segmentation State
Guest FS Limit	0x00004808	Ignored	Reduced Segmentation State
Guest GS Limit	0x0000480A	Ignored	Reduced Segmentation State
Guest ES Access Rights	0x00004814	Ignored	Reduced Segmentation State
Guest DS Access Rights	0x0000481A	Ignored	Reduced Segmentation State
Guest FS Access Rights	0x0000481C	Ignored	Reduced Segmentation State
Guest GS Access Rights	0x0000481E	Ignored	Reduced Segmentation State
Guest LDTR Access Rights	0x00004820	Ignored	Reduced Segmentation State
Guest TR Access Rights	0x00004822	Ignored	Reduced Segmentation State
Guest CS Access Rights	0x00004816	Only L and D saved/loaded; DPL is written to SS.DPL on exit and ignored on entry, the rest written to zero on exit and ignored on entry	Reduced Segmentation State
Guest SS Access Rights	0x00004818	Only DPL and B saved/loaded; the rest written to zero on exit and ignored on entry	Reduced Segmentation State
Guest ES Base	0x00006806	Ignored	Reduced Segmentation State
Guest CS Base	0x00006808	Ignored	Reduced Segmentation State
Guest SS Base	0x0000680A	Ignored	Reduced Segmentation State
Guest DS Base	0x0000680C	Ignored	Reduced Segmentation State
Guest PDPTE0	0x0000280A	Ignored	IA32e mode always enabled
Guest PDPTE1	0x0000280A	Ignored	IA32e mode always enabled
Guest PDPTE2	0x0000280A	Ignored	IA32e mode always enabled
Guest PDPTE3	0x0000280A	Ignored	IA32e mode always enabled

3.14.2 VMCS Exit Controls

VM exit controls that are changed are listed in Table 13.

Table 13. VMCS Exit Control Changes

VMCS Field	Change	Reason
Host Address Space Size (HASS)	Fixed 1	Host is always in 64-bit supervisor mode.

3.14.3 VMCS Entry Controls

VM entry controls that are changed are listed in Table 14.

Table 14. VMCS Entry Control Changes

VMCS Field	Change	Reason
IA-32e mode guest	Fixed 1	Guest is always in long mode.

3.14.4 VMCS Secondary Processor-Based Execution Controls

Changes are listed in Table 15.

Table 15. Secondary Processor-Based Execution Control Changes

VMCS Field	Change	Reason
Unrestricted guest	Fixed 0	Unrestricted guest not supported.

3.14.5 VMX Enumeration

VMX enumeration changes are listed in Table 16.

Table 16. VMX Enumeration Changes

MSR	Bit(s)	Corresponding Field	Value	Notes
IA32_VMX_EXIT_CTLS	9, 41	host address space size	1	EFER LME and LMA are fixed to 1.
IA32_VMX_TRUE_EXIT_CTLS				
IA32_VMX_ENTRY_CTLS	9, 41	IA-32e mode guest	1	Guest is always in long mode.
IA32_VMX_TRUE_ENTRY_CTLS				
IA32_VMX_PROCBASED_CTLS2	39	unrestricted guest	0	No unrestricted guest.
IA32_VMX_MISC	8	supports activity state: wait-for-SIPI	0	Unsupported.
IA32_VMX_CR0_FIXED0	0	PE: protected mode enable	1 (legacy)	

MSR	Bit(s)	Corresponding Field	Value	Notes
	1	MP: monitor coprocessor	1	Always long mode, no legacy FPU modes. Fixed to 0.
	5	NE: numeric error	1 (legacy)	
	31	PG: paging enabled	1 (legacy)	
IA32_VMX_CR0_FIXED1	2	EM: FP emulation	0	These CR0 bits are fixed to 0.
	29	NW: not write-through	0	
IA32_VMX_CR4_FIXED0				No changes.
IA32_VMX_CR4_FIXED1	1	PVI: protected-mode virtual interrupts	0	No support for protected-mode virtual interrupts.

3.15 SMX Changes

The behavior of the following sub-leaves of the GETSEC instruction, ENTERACCS/SENTER, EXITAC as well as the RLP WAKEUP, are modified for X86S. The environment after entering an authenticated code module is X86S-compliant, so the instructions reflect changes to that behavior.

3.15.1 Summary of Changes to SMX in X86S

The following changes have been made:

1. Overall changes:
   a. ACBASE is set to 0FEB00000h by the CPU. The CPU loads the ACM image from a pointer in memory and copies it to an internal memory at location 0FEB00000h.
   b. There is no requirement for ACM to be located in a region with WB memory type.
2. Changes to ENTERACCS/SENTER:
   a. The CodeControl field is removed, and CodeControl checks are removed.
   b. Pre-Entry CR3, CR4, RIP, RSP, and FRED MSRs are saved to a state save area in the internal ACRAM memory.
   c. CR3, RIP, and FRED CONFIG MSRs are loaded from the ACM header. The new format for the ACM header is shown in 0.
   d. All segment state is unmodified.
   e. CR4, FRED_SKTLVLS, and RSP are forced to a fixed value. FRED is forced to 1.
3. EXITAC

   CR3, CR4, RIP, and FRED states are loaded from a storage structure specified by R8.
4. WAKEUP
   a. CR3,CR4, and RIP are loaded from the JOIN structure. The new JOIN structure is shown in Table 18.
   b. Segment state is unmodified.
5. SEXIT

   There is no change to SEXIT itself. However, on X86S, if RLP was in LT_WFS (SENTER_SLEEP) and NEWSIPI is not enabled, INIT will result in a shutdown.

Table 17. X86S-Compliant ACM Header Format (New and Modified Fields Highlighted)

Field	Offset	Size (bytes)	Description
ModuleType	0	2	2 – ACM
ModuleSubType	2	2	Module sub-type
HeaderLen	4	4	Header length (in multiples of four bytes) 928
HeaderVersion	8	4	Module format version 10.0
ChipsetID	12	2	Module release identifier
Flags	14	2	Module-specific flags
ModuleVendor	16	4	Module vendor identifier
Date	20	4	Creation date (BCD format: year.month.day)
Size	24	4	Module size (in multiples of four bytes)
ACM_SVN	28	2	ACM *Security Version Number*
TEE_SVN	30	2	*Security Version Number* for "Trusted Execution Environment" features – SGX/TDX
State Save Address (SSA)	32	4	**Physical** address of an Internal ACRAM storage. Used by ENTERACCS/SENTER uCode to store certain state values of a Source environment. (replaces existing CodeControl field)
ACM CR3	36	4	ACM CR3
ACM FRED_CONFIG	40	4	FRED configuration MSR
ACM RIP	44	4	**Linear** address of ACM EntryPoint
Reserved	48	36	Reserved. Must be zero.

Table 18. X86S Measured Launch Environment JOIN Structure

Offset	Field
0	Linear address of target entry point.
8	Target CR3
16	Target CR4

3.15.2 Overview of Changes to State After ENTERACCS/SENTER

These changes, shown in Table 19, provide an X86S-compliant environment.

Table 19. Changes to State After ENTERACCS/SENTER

Register State	Value After ENTERACCS - Legacy	Value After ENTERACCS – X86S
CR0	PG=0, AM=0, WP=0: Others unchanged	PG=1; AM=0; WP=1; Others unchanged
CR4	MCE=0, CET=0, PCIDE=0: Others unchanged	PAE=1, FRED=1; SMXE=1; Others 0
IA32_EFER	0H	Unmodified (EFER has fixed value in X86S)
EIP	AC.base + EntryPoint	ACMHeader[RIP]
[E|R]BX	Pre-ENTERACCS state: Next [E|R]IP prior to GETSEC[ENTERACCS]	Unmodified
ECX	Pre-ENTERACCS state: [31:16]=GDTR.limit;[15:0]=CS.sel	Unmodified

Register State	Value After ENTERACCS - Legacy	Value After ENTERACCS – X86S
[E|R]DX	Pre-ENTERACCS state:GDTR base	Unmodified
EBP	AC.base	0FEB00400h
CS	Sel=[SegSel], base=0, limit=FFFFFh, G=1, D=1, AR=9BH	Unmodified
DS	Sel=[SegSel] +8, base=0, limit=FFFFFh, G=1, D=1, AR=93H	Unmodified
GDTR	Base= AC.base (EBX) + [GDTBasePtr], Limit=[GDTLimit]	Unmodified
CR3	Unmodified	ACMHeader[CR3]
FRED_CONFIG	Unmodified	ACMHeader[FRED_CONFIG]
FRED_STKLVLS	Unmodified	0

3.15.3 ENTERACCS / SENTER Pseudocode in X86S

```
(* The state of the internal flag ACMODEFLAG persists across instruction boundary *)
IF (CR4.SMXE=0)
        THEN #UD;
ELSIF (in VMX non-root operation)
        THEN VM Exit (reason=”GETSEC instruction”);
ELSIF (GETSEC leaf unsupported)
        THEN #UD;
ELSIF ((in VMX operation) or
        (CR0.CD=1) or (CPL>0) or (IA32_APIC_BASE.BSP=0) or
         (TXT chipset not present) or (ACMODEFLAG=1) or (IN_SMM=1))
THEN #GP(0);
IF (GETSEC[PARAMETERS].Parameter_Type = 5, MCA_Handling (bit 6) = 0)
        FOR I = 0 to IA32_MCG_CAP.COUNT-1 DO
                IF (IA32_MC[I]_STATUS = uncorrectable error)
                THEN #GP(0);
        OD;
FI;
IF (IA32_MCG_STATUS.MCIP=1) or (IERR pin is asserted)
        THEN #GP(0); FI
ACBASE := EBX;
ACSIZE := ECX;
IF (((ACBASE MOD 4096) ≠ 0) or ((ACSIZE MOD 64 ) ≠ 0 ) or (ACSIZE < minimum module size) OR (ACSIZE >
authenticated RAM capacity)) or ((ACBASE+ACSIZE) > (2^32 -1)))
        THEN #GP(0); FI
IF (secondary thread(s) CR0.CD = 1) OR ((secondary thread(s) NOT(wait-for-SIPI)) AND
(secondary thread(s) not in SENTER sleep state)
        THEN #GP(0); FI
Mask SMI, INIT, A20M, and NMI external pin events;
IA32_MISC_ENABLE := (IA32_MISC_ENABLE & MASK_CONST*)
(* The hexadecimal value of MASK_CONST may vary due to processor implementations *)
IA32_DEBUGCTL := 0;
Invalidate processor TLB(s);
Drain Outgoing Transactions; ACMODEFLAG := 1;
SignalTXTMessage(ProcessorHold);
Set up entire ACRAM space and load the internal ACRAM from ACBASE to FEB00000h based on the AC module
size;
Set ACBASE := 0FEB00000h;
IF (AC module header version is not supported) OR (ACRAM[ModuleType] ≠ 2)
```

```
        THEN TXT-SHUTDOWN(#UnsupportedACM);
FI
(* Authenticate the AC Module and shutdown with an error if it fails *)
KEY := GETKEY(ACRAM, ACBASE);
KEYHASH := HASH(KEY);
CSKEYHASH := READ(TXT.PUBLIC.KEY);
IF (KEYHASH ≠ CSKEYHASH)
        THEN TXT-SHUTDOWN(#AuthenticateFail);
FI
SIGNATURE := DECRYPT(ACRAM, ACBASE, KEY);
(* The value of SIGNATURE_LEN_CONST is implementation-specific*)
FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
        ACRAM[SCRATCH.I] := SIGNATURE[I];
DONE
COMPUTEDSIGNATURE := HASH(ACRAM, ACBASE, ACSIZE);
FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
        ACRAM[SCRATCH.SIGNATURE_LEN_CONST+I] := COMPUTEDSIGNATURE[I];
DONE
IF (SIGNATURE ≠ COMPUTEDSIGNATURE)
        THEN TXT-SHUTDOWN(#AuthenticateFail);  FI
IF (ACRAM[StateSaveAddress] MOD 64) ≠ 0)
  THEN TXT-SHUTDOWN(#BadACMFormat); FI
If (ACRAM[IA32_FRED_CONFIG] has reserved bits set)
  THEN TXT-SHUTDOWN(#BadACMFormat); FI
(* Save state to StateSaveArea *)
SSAddr[FRED_CONFIG] := IA32_FRED_CONFIG
SSAddr[FRED_STKLVLS] := IA32_FRED_STKLVLS
SSAddr[CR4] := CR4[63:0]
SSAddr[CR3] := CR3[63:0]
SSAddr[RIP] := Pre-ENTERACCS next RIP
SSAddr[RSP] := RSP
CR0.[AM] := 0;
CR0.[PG.WP] := 1;
CR4[FRED,PAE,SMXE]=1; Rest of CR4=0
EFLAGS := 00000002h;
RSP := 0FEB00400h;
CR3 := ZX(ACRAM[CR3], 64);
IA32_FRED_STKLVLS = 0;
IA32_FRED_CONFIG = ZX(ACRAM[IA32_FRED_CONFIG], 64);

DR7 := 00000400h;
IA32_DEBUGCTL := 0;
SignalTXTMsg(OpenPrivate);
SignalTXTMsg(OpenLocality3);
EIP := ACEntryPoint;
END;
```

3.15.4 EXITAC Pseudocode in X86S

```
(* The state of the internal flag ACMODEFLAG and SENTERFLAG persist across instruction boundary *)
IF (CR4.SMXE=0)
        THEN #UD;
ELSIF ( in VMX non-root operation)
        THEN VM Exit (reason=”GETSEC instruction”);
```

```
ELSIF (GETSEC leaf unsupported)
        THEN #UD;
ELSIF ((in VMX operation) OR ( (in 64-bit mode) AND ( RBX is non-canonical) ) OR
(CR0.PE=0) OR (CPL>0) OR (EFLAGS.VM=1) OR (ACMODEFLAG=0) OR (IN_SMM=1)) OR (EDX ≠ 0))
        THEN #GP(0); FI
(* Check that the StateSave address is legal *)
SSAddr := R8
IF ((SSAddr MOD 64) ≠ 0 or beyond MAX_PA)
  THEN #GP(0); FI

TempRIP := SSAddr[RIP]
TempRSP := SSAddr[RSP]
TempCR4 := SSAddr[CR4]
TempCR3 := SSAddr[CR3]
TempFredConfig  := SSAddr[FRED_CONFIG]
TempFredSTKLVLS := SSAddr[FRED_STKLVLS]

(* Perform checks on SSA state *)
IF ((TempCR3 reserved bit set) or
  (TempRIP or TempRSP are non-canonical according to TempCR4.LA57) or
  (TempCR4 & CR4_MASK_CONST ≠ 0 ) or
  (TempFREDConfig has reserved bits set or is not canonical)
THEN #GP(0); FI

Invalidate ACRAM contents;
Invalidate processor TLB(s);
Drain outgoing messages;
SignalTXTMsg(CloseLocality3);
SignalTXTMsg(LockSMRAM);
SignalTXTMsg(ProcessorRelease);
Unmask INIT;
IF (SENTERFLAG=0) THEN
        THEN Unmask SMI, INIT, NMI, and A20M pin event;
ELSIF (IA32_SMM_MONITOR_CTL[0] = 0)
        THEN Unmask SMI pin event;
FI
ACMODEFLAG := 0;
CR3 := TempCR3;
CR4 := TempCR4;
RIP := TempRIP;
RSP := TempRSP;
IA32_FRED_CONFIG := TempFredConfig;
IA32_FRED_STKLVLS := TempFredSTKLVLS;

END;
```

3.15.5 RLP_SIPI_WAKEUP_FROM_SENTER_ROUTINE in X86S: (RLP Only)

```
WHILE (no SignalWAKEUP event) DO DONE;
IF (IA32_SMM_MONITOR_CTL[0] ≠ ILP.IA32_SMM_MONITOR_CTL[0]) THEN
        THEN TXT-SHUTDOWN(#IllegalEvent); FI
IF (IA32_SMM_MONITOR_CTL[0] = 0) THEN
        THEN Unmask SMI pin event;
ELSE
        Mask SMI pin event;
```

```
FI
Mask A20M, and NMI external pin events (unmask INIT);
Mask SignalWAKEUP event;
Invalidate processor TLB(s);
Drain outgoing transactions;
TempRIP := LOAD (LT.MLE.JOIN+0);
TempCR3 := LOAD (LT.MLE.JOIN+8);
TempCR4 := LOAD (LT.MLE.JOIN+16);
IF (TempCR3 reserved bits set) OR
   (TempRIP is non-canonical) OR
   (TempCR4 & CR4_RESERVED_BIT_MASK ≠ 0 )
   THEN TXT-SHUTDOWN(#BadJOINFormat); FI
```

3.16 CET changes

The IA32_PL1_SSP and IA32_PL2_SSP MSRs to configure the shadow stack in ring 1 and ring 2 are preserved even though ring 1 and 2 do not exist anymore. Writes check the existing reserved bits and reads return the previously written value.

3.17 Summary of Removals

A summary of removals is given in Table 20.

Table 20. Summary of Removals

Removal of	Replacement	Implied by
Segment bases (except FS/GS/GDT/IDT/LDT/TSS), limits (except GDT/IDT/TSS/LDT), segment permissions (other than CS.L), unusable checks (other than for SS/CS/TR), segment types (other than S)	-	Limited segmentation
Real mode (big and 16-bit)	64-bit paged mode, 64-bit SIPI	-
16-bit protected mode	-	-
16-bit address override in other modes when address is referenced	-	-
32-bit ring 0, including 2- and 3-level paging modes	64-bit ring 0	-
Disabling FPU through CR0.MP	Use CR0.TS to disable FPU	-
Legacy numeric error handling	-	-
VM86 mode	-	16-bit mode removal
Protected-mode virtual interrupts (PVI)	-	-
Clearing EFER.NXE bit to disable presence of NX bit in page table entries	-	-
Disabling SYSCALL through EFER.SCE	-	-
FAR JMP/RET/CALL changing rings	SYSCALL, INT	Limited segmentation
IRET/SYSCALL/SYSRET entering 16-bit mode, VM86 mode or conforming segments. ERETU supporting non-STAR segments.	-	16-bit mode removal, Limited segmentation
Fixed MTRRs	Variable MTRRs , PAT in page tables	-
MMIO-based XAPIC access	X2APIC access through MSRs	-

Removal of	Replacement	Implied by
APIC ExtInt removal	-	-
Ring 1, ring 2 removal	-	-
Ring 3 I/O port access (IOPL, I/O bitmap)	Ring 0 I/O port access	-
INS and OUTS instructions	IN, OUT instructions in loops	-
#SS exception	#GP(0) exception	Limited segmentation
#NP exception	#GP(0) exception	Limited segmentation
Support for INIT/SIPI on entry in VMCS	-	64-bit SIPI
Support for unrestricted guest in VMCS	-	16-bit mode removal, paging always enabled
VMCS support for 32-bit ring 0	-	32-bit ring 0 removal

3.18 Summary of Additions

New additions in the architecture are given in Table 21.

Table 21. Summary of Additions

Addition of	Reason	Needed by
64-bit SIPI and INIT	Boot application processors in paged 64-bit mode	Real mode removal

3.19 Changed Instructions

The following descriptions pertain only to new behavior of the instructions. A longer list of segmentation-related instructions with changed behavior (if any) is shown in Section 4.3. Some instruction with trivial changes are only documented in the Summaries.

For legacy behavior, please refer to *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4*.

3.19.1 SYSRET

SYSRET will generate a #GP(0) exception if a non-zero value is loaded into RFLAGS.IOPL, RFLAGS.VIP, or RFLAGS.VIF.

3.19.2 IRET

IRET cannot jump to 16-bit mode, task gates. IRET will generate a #GP(0) exception if a non-zero value is loaded into RFLAGS.IOPL, RFLAGS.VIP, or RFLAGS.VIF when in ring 0. The details of the IRET instruction are shown in the pseudocode in Section 4.2.6.

3.19.3 POPF – Pop Stack Into RFLAGS Register

The behavior of POPF is summarized in Table 22.

Table 22. POPF Behavior

Opcode	Instruction	Op/ En	64-Bit Mode	Compatibility/ Legacy Mode	Description
9D	POPFD	ZO	N.E.	Valid	Pop top of stack into EFLAGS.
9D	POPFQ	ZO	Valid	N.E.	Pop top of stack and zero-extend into RFLAGS.

POPF pops a doubleword (POPFD) from the top of the stack (if the current operand size attribute is 32) and stores the value in the EFLAGS register, or pops a word from the top of the stack (if the operand size attribute is 16) and stores it in the lower 16 bits of the EFLAGS register (that is, the FLAGS register). These instructions reverse the operation of the PUSHF/PUSHFD/PUSHFQ instructions.

The IOPL, VM, VIP, and VIF flags are always zero and are ignored on POPF.

The POPF instruction never raises an #SS exception, but only a #GP(0) or a #PF exception.

It changes RFLAGS according to Table 23.

Table 23. RFLAGS Changes with POPF Instruction

Mode	Operand Size	CPL	Flags																
			21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0
			ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
Ring 3 and Ring 0 modes	32, 64	*	S	N	N	S	N	0	S	N	S	S	N	S	S	S	S	S	S

Key	
S	Updated from stack
N	No change in value
0	Value is cleared

Pseudocode:

```
tempFlags = POP // according to 32/64 operand size
IF CPL = 0 THEN
   // modify non reserved flags with “tempFlags” except RF, IOPL, VIP, VIF, VM.
   // RF is cleared.
   // Do not modify flags not popped due to operand size.
ELSE
   // modify non reserved flags with “tempFlags” except RF, IOPL, VIP, VIF, VM, IF.
   // RF is cleared.
   // Do not modify flags not popped due to operand size.
FI
```

3.20 Summary of Removed Instructions

Table 24 shows a summary of removed instructions.

Table 24. Removed Instructions Summary

Instruction	Possible Legacy Usage in Rings	Replacement
INS / OUTS	Ring 3, 1, 2, 0	IN, OUT
64-bit indirect far jump with 0x67 prefix that changes to 16-bit operand will #UD. 32-bit near ret, far call, far ret, far jmp, with 0x67 prefix that changes to 16-bit operand will #UD. 32-bit near jmp, jCC, JECX*, near ret, near call, loop*, far jmp with 0x67 prefix that changes to 16-bit operand will #UD. 32-bit any instruction that references memory and is not a jump with 0x67 prefix that changes to 16-bit operand will #GP.	Ring 3, 1, 2, 0	32-bit/64-bit memory references
LMSW	Ring 3, 1, 2, 0	Mov CR0

3.21 Summary of Changed Instructions

Table 25 shows a summary of changed instructions.

Table 25. Changed Instructions Summary

Instruction	Rings	Change
ERETU	0	No support for non-STAR segments. The RFLAGS IOPL, VM, VIF, VIP bits must be zero.
ERETS	0	The RFLAGS IOPL, VM, VIF, VIP bits must be zero.
IRET	3, 0	No support for 16-bit mode or VM86 or gates or ring 1 or 2. Must stay in ring or go from ring 0 to ring 3. The RFLAGS IOPL, VM, VIF, VIP bits must be zero. Simplified checks.
SYSRET	0	The RFLAGS IOPL, VM, VIF, VIP bits must be zero.
FAR CALL	3, 0	Cannot change rings. #UD on 16-bit operand size. #GP on 0x67 in 32-bit mode and indirect.
FAR JMP	3, 0	Cannot change rings. #UD on 16-bit operand size. #GP on 0x67 in 32-bit mode.
POPF	3, 0	The RFLAGS IOPL, VM, VIF, VIP bits must be zero.
FAR RET	3, 0	Cannot change rings. #UD with 16-bit operand size.
STI	3, 0	No support for ring 3 changes through VM86/PVI.
CLI	3, 0	No support for ring 3 changes through VM86/PVI.
VERW, VERR, MOV to sel, MOV from sel, PUSH sel, POP sel, LAR, LGS, LFS, LES, LFS, LGS, LSS, LDS, LKGS	3, 0	Simplified checks.
IN*, OUT*	3	Removed support for ring 3 port I/O.
JMP short, JMP, LOOP, JECX, CALL, RET, JMP	3, 0	#UD with 16-bit operand size prefix in 32-bit mode.

3.22 Software Compatibility Notes

3.22.1 Emulation of Ring 3 I/O Port Access

If there are legacy uses of ring 3 I/O port accesses using the TSS I/O port bitmap or IOPL, it is possible to emulate this case through a #GP(0) handler that executes IN/OUT in ring 0. INS/OUTS can be emulated in a #UD handler with an appropriate emulation routine.

3.22.2 64-Bit SIPI

The BIOS should always disable 64-bit SIPI in the SIPI_ENTRY_STRUCT ENABLES field before passing control to the OS. The BIOS must initialize and enable the IA32_SIPI_ENTRY_STRUCT_PTR MSR on all packages. On Intel64, this will ensure that a legacy OS can use legacy SIPI. A 64-bit-SIPI-aware OS can enable it. On X86S it is not possible to use legacy SIPI, but the OS owns the enabling of 64-bit SIPI, too, for consistency.

3.22.3 64-Bit Reset

Reset uses the same entry point as Intel64, but uses paged 64-bit mode. When compatibility to Intel64 is desired, the entry code can determine if it is entered on X86S by checking CR0.PG in code that is identical to 32-bit and 64-bit.

3.22.4 Legacy OS Virtualization

The VMM is responsible for setting up the system state and VMCS appropriately so that the necessary VM exits and faults occur for cases where emulation of legacy behavior by the VMM is required. There are also cases where the VMM should not attempt to perform a VM entry, but instead emulate until a supported guest state is reached, for example when entering into 16-bit or 32-bit ring 0 code.

If required for guest compatibility, the VMM is responsible for (a) setting the exception bitmap such that #UD and #GP cause a VM exit and then (b) emulating to determine the cause of the exception and the appropriate response. Some examples:

- Some variants of CLI will spuriously #GP(0), for example, if a legacy guest tried to execute a CLI in ring 3 and RFLAGS.IOPL==3. Since RFLAGS.IOPL is always 0, this ring 3 CLI will always #GP(0). If the guest requires these IOPL semantics, it is up to the VMM to emulate this instruction with the emulated legacy guest RFLAGS.IOPL value. Note that there are un-virtualizable aspects of a non-zero IOPL that are discussed later.
- #SS and #NP are converted to #GP. If the guest expects to see the #SS/#NP, the VMM will need to detect cases where a #GP would have been an #SS or #NP and inject them to the guest.

Some guest CR values are ignored on VMENTRY (they retain the fixed values and are not consistency checked). If required by the guest, the VMM can virtualize differences, some of which are described below:

- CR0.MP is fixed to one. Here, the VMM should diagnose and emulate spurious faulting cases.
- CR4.PVI is fixed to zero. Here, the VMM can diagnose #GPs from STI/CLI and emulate the expected guest behavior.
- CR4.DE is fixed to one. Here, the VMM can diagnose and emulate spurious faulting cases.
- CR4.PSE and CR4.PAE are fixed. Legacy paging modes require shadow paging or emulation.

- EFER.LME is fixed to one. If the guest is in 32-bit CPL0 mode and the VMM wants to do a VM entry, it should use emulation.
- RFLAGS:
    - IOPL is fixed 0
    - VIF, VIP are fixed 0. Some CLI/STI may #GP(0) and can be emulated to handle these appropriately if the guest requires this functionality.

A VMM can choose to emulate legacy functionality as required:

1. VMM changes required for mainstream Intel64 guest using legacy SIPI or non-64-bit boot:
    a. Emulate 16-bit modes (real mode, virtual 8086 mode)
    b. Emulate unpaged modes
    c. Emulate legacy INIT/SIPI.
2. Optional VMM changes for handling uncommon cases:
    a. IOPL != 0 (if guest wants ring 3 I/O port access or ring 3 CLI/STI):
        i. Catch CLI #GP in CPL3 and emulate.
        ii. Catch STI #GP in CPL3 and emulate.
        iii. Catch IN/OUT #GP in CPL3 and emulate.
        iv. IRET in CPL0 will #GP if attempting to change IOPL; catch and emulate.

    Note that that are un-virtualizable aspects of a non-zero IOPL described in the next section.

    b. INS/OUTS instructions are removed: Catch #UD and emulate.
    c. Call gates: VMM needs to catch relevant #GPs and emulate.
    d. #SS removal: VMM can catch relevant #GPs and report #SS back to guest.
    e. #NP removal: VMM can catch relevant #GPs and report #NP back to guest.
    f. CR4.PVI: catch and emulate associated #GPs.
    g. Emulate 16-bit addressing by catching #GPs/#UDs.
    h. CR4.VME, RFLAGS.VM: Emulate v8086 mode.
    i. Emulate 32-bit ring 0 and run 32-bit ring 3 with shadow paging in legacy paging modes.
    j. Support for unsupported obscure segmentation features like expand down or non-conforming code segments: Can be emulated by catching #GPs.
3. Uncommon cases with expensive SW solutions:
    a. CPL1/2 requires partial emulation.
    b. Non-flat CS/DS/ES/SS segments or setting access bits in descriptors in memory requires full emulation triggered by Descriptor Table Exiting and then setting the GDT/LDT limit to zero (or read/write protect GDT/LDT) to catch segmentation instructions.
    c. When EFER.NXE is cleared, a set NX bit in PTE requires shadow paging.
    d. Segmentation permission checking on load/store/execute: this would require full emulation.
4. Cases that are un-virtualizable:
    a. RFLAGS.IOPL != 0: When IOPL is non-zero, most cases where behavior would typically change will instead #GP, which the VMM can catch/emulate (i.e., many cases are virtualizable). The problematic Intel64 cases are as follows:
        i. Ring 0 privileged SW sets IOPL to 3 and changes to ring 3. If ring 3 SW runs PUSHF or SYSCALL, the value with IOPL=3 should go into the memory or register destination. When this sequence runs in a VM, the ring 0 instruction that sets IOPL to 3 would cause a #GP and trigger a VMExit. If the VMM resumes the VM with the "wrong" IOPL, i.e., IOPL==0, the ring 3 PUSHF or SYSCALL would expose this incorrect IOPL through memory or the register. Also, ring3 POPF will not update IF. The preferred scheme is

for VMM to emulate the guest until IOPL is changed back to 0. This case is not expected on modern software.

ii. If the guest attempts to set IOPL to a value greater than zero using a POPF instruction in ring 0, this will be silently ignored. The IOPL value will not be updated and the VMM will be unaware that this occurred. Some subsequent consumers of this value (e.g., CLI/STI/IN/OUT) will generate a #GP, but others will silently continue with different semantics (e.g., IF updating POPF, memory written by PUSHF, flags stored by SYSCALL, etc.)

b. #UD behavior on SYSCALL/SYSEXIT when EFER.SCE is cleared.

3.22.5 Legacy OS Without VMM support

It is possible to boot some 64-bit legacy OS by pointing the ENTRY_STRUCT memory context to a software handler that can execute the 16-bit/32-bit entry points until 64-bit mode is reached. It will retrieve the original 16-bit entry point vector value in the R10 register. This technique can be also used to boot legacy OSes inside VMMs that do not have emulation support.

3.22.6 Migration to Intel64

When migrating a guest from X86S to Intel64, the most permissive segmentation state needs to be filled in for segmentation VMCS fields that are removed in X86S:

Limit: Fill in infinite for removed fields

Base: Fill in 0 for removed bases

CS: If selector is zero, fill in Unusable=1 Else S=1, Type=11, L=from VMCS, CPL=from VMCS, D=!L, G=1, P=1

SS: If selector is zero, fill in Unusable=1 Else S=1, Type=3, B=from VMCS, CPL=from VMCS, P=1, G=1

DS/ES/FS/GS: If selector is zero, fill in Unusable=1 Else S=1, Type=3, G=1, P=1

LDT/GDT/TR: Fill in S=0 and respective type, G bit based on limit value.

Fields not mentioned are 0.

(This page intentionally left blank)

4 Appendix

This appendix gives further details on limited segmentation and exception compatibility.

4.1 Segmentation Instruction Behavior

Note the descriptions only describe the new behavior of the instructions. For legacy behavior please refer to the SDM. The pseudocode might not have the final fault ordering or error codes. If something is not changed from baseline, it will not be mentioned.

```
Check_selector(selector):
  IF CS AND selector is NULL THEN
    #GP(0);
  FI
  IF (selector.TI == 0 AND selector exceeds GDT limit) OR
    (selector.TI == 1 AND selector exceeds LDT limit) OR
    Descriptor address in table is non canonical THEN
    #GP(selector); // OR ZF := 0
  FI
END

Check_CS_desc(selector, Descriptor, newCPL):
  IF Descriptor is not code segment
  OR (Descriptor.L xor Descriptor.D == 0)  // prevents 16b size. Invalid size
  OR Descriptor.DPL == 1 // only needed for gates
  OR Descriptor.DPL == 2 // only needed for gates
  OR selector.RPL != Descriptor.DPL
  OR (Descriptor.DPL != newCPL and not (trap or int gate))
  OR (Descriptor.DPL > newCPL and (trap or int gate)) // gates cannot go out
  OR (Descriptor.L == 0 AND Descriptor.DPL == 0)      // prevent 32-bit ring 0
  OR  (descriptor.P == 0 ) THEN
    #GP(selector);
  FI
END

Check_CS_desc_for_IRET(selector, Descriptor, newCPL):
  IF Descriptor is not code segment
  OR (Descriptor.L xor  Descriptor.D == 0)  // prevents 16b size. Invalid size
  OR Descriptor.DPL == 1
  OR Descriptor.DPL == 2
  OR selector.RPL != Descriptor.RPL
  OR Descriptor.DPL != newCPL                          // IRET cannot go in
  OR (Descriptor.L == 0 AND Descriptor.DPL == 0)  // prevents 32-bit ring 0
```

```
    OR  (descriptor.P == 0 ) THEN
     #GP(selector);
 FI
END

Check_Data_desc(selector, Descriptor):
    IF selector is not NULL THEN
        IF selector exceeds GDT/LDT limit  // does not apply to VMEntry/RSM
        OR selector.RPL < CPL
        OR Descriptor is system type
        OR  (descriptor.P == 0 ) THEN
            #GP(selector); // OR ZF := 0
        FI
    FI
END
// This is used for mov SS, pop SS, LSS
Check_SS_desc(selector, Descriptor):

      IF selector exceeds GDT/LDT limit
        OR (selector is non NULL AND selector.RPL != DPL)
        OR selector is NULL
        OR Descriptor is system type
        OR Descriptor.DPL != CPL
        OR Descriptor.P == 0 THEN
            #GP(selector); // OR ZF := 0
      FI
   // SS.B / Selector / DPL are saved for VMX
END
// This is used only for IRET
Check_SS_desc_for_iret(selector, Descriptor, newCPL):
      IF selector is not NULL THEN
        IF selector exceeds GDT/LDT limit
          OR selector.RPL != Descriptor.DPL
          OR Descriptor is system type
          OR Descriptor.DPL != newCPL
          OR descriptor.P == 0 THEN
              #GP(selector); // OR ZF := 0
          FI
       ELSIF (newCPL == 3 OR  NOT 64b mode) THEN // NULL
           #GP(selector)
        FI
    // SS.B / Selector / DPL are saved for VMX
END

Load_descriptor_from_GDT_LDT(selector):
  IF (selector & 0xFFF8) != 0x0 THEN
    IF selector.TI == 1 THEN BASE := LDT Base;
    ELSE BASE := GDT Base; FI;
    Desc := load_physical_sup(BASE + (selector & 0xFFF8));
    Return Desc;
  ELSE
    Return 0;
  FI
END

Load_descriptor_from_IDT(vector):
  Desc := load physical_sup(IDT base + vector << 4);
  Return Desc;
END
```

4.2 Segmentation Instruction Pseudocode

4.2.1 CALL Far

Far CALLs are intra-level only. Mode restrictions are enforced. The selector must point to a non-conforming code descriptor in the GDT/LDT. The CS.accessed bit is not set. With a 16-bit operand size, the instruction raises an #UD exception. With a 0x67 prefix, and when indirect, and in 32-bit mode, the instruction raises a #GP(0). The #NP and #SS exceptions are replaced with #GP.

```
IF 16bit operand size  THEN #UD ; FI
IF 0x67 prefix AND indirect AND 32bit mode THEN #GP(0); FI
Check_selector(newCS);
newCSdesc := Load_descriptor_from_GDT_LDT(tempCS);
Check_CS_desc(tempCS, newCSdesc, CPL);
IF target mode is compat mode THEN
  newRIP = newRIP & 0xffffffff;
FI
IF newRIP is non-cannonical THEN
  #GP(0)
FI
Push CS;
Push RIP;
CS  := newCS;
RIP := newRIP;
```

```
Save newCSdesc;
Do shadow stack pushes if enabled
Do end branch state transition if enabled
```

4.2.2 ERETU

Enforces RFLAGS restrictions. Mode restrictions are enforced, as well as limits on code selector types. No access bits for descriptors are set. #NP and #SS are replaced with #GP.

```
Beginning of flow the same as Intel64
// Intel64 FRED checks for CS/SS compatible with IA32_STAR
Same FRED code
ELSE IF newCS  OR newSS not compatible with IA32_ STAR THEN
  #GP(0);
FI
Rest of flow is same as Intel64
```

4.2.3 ERETS

Enforces RFLAGS restrictions.

4.2.4 FRED ENTRY FLOW

Enforces RFLAGS restrictions.

4.2.5 Int n, INT3, INTO, External Interrupt, Exceptions with CR4.FRED == 0

Mode restrictions and descriptor type restrictions are enforced. Access bits for descriptors are not set. #NP is replaced with #GP.

```
IF INTO and CS.L = 1 THEN
   #UD;
FI;
IF ((vector_number « 4) + 15) is not in IDT.limit THEN
   #GP(error_code(vector_number,1,EXT));
FI;
gate := Read_descriptor_from_IDT(vector_number);
IF gate.type not in {intGate64, trapGate64} THEN
   #GP(error_code(vector_number,1,EXT));
FI;
IF software interrupt (* does not apply to INT1 *) THEN
   IF gate.DPL < CPL THEN
      #GP(error_code(vector_number,1,0));
   FI;
FI;
IF gate.P == 0 THEN
   #GP(error_code(vector_number,1,EXT));
```

```
FI
newCS := gate.selector;
IF newCS is NULL THEN
    #GP(EXT); (* Error code contains NULL selector *)
FI;
Check_selector(newCS);
newCSdesc := Load_descriptor_from_GDT_LDT(newCS);
Check_CS_desc(newCS, newCSdesc, 0);
IF newCSdesc.DPL < CPL THEN
    GOTO INTER-PRIVILEGE-LEVEL-INTERRUPT;
ELSIF newCSdesc.DPL = CPL THEN
    GOTO INTRA-PRIVILEGE-LEVEL-INTERRUPT;
ELSE
    #GP(error_code(new code-segment selector,0,EXT));
FI
END;
INTER-PRIVILEGE-LEVEL-INTERRUPT:
    IF gate.IST == 0 THEN
        TSSstackAddress := (newCSdesc.DPL « 3) + 4;
    ELSE
        TSSstackAddress := (gate.IST « 3) + 28;
    FI;
    IF (TSSstackAddress + 7) > TSS.limit THEN
        #TS(error_code(TSS.selector,0,EXT);
    FI;
    NewRSP := 8 bytes loaded from (TSS.base + TSSstackAddress);
    NewSS  := newCSdesc.DPL; (* NULL selector with RPL = new CPL *)
    IF gate.IST = 0 THEN
        NewSSP := IA32_PLi_SSP; (* where i = newCSdesc.DPL *)
    ELSE
        NewSSPAddress := IA32_INTERRUPT_SSP_TABLE_ADDR + (gate.IST « 3);
        IF ShadowStackEnabled(CPL0) THEN
                NewSSP := 8 bytes loaded from NewSSPAddress;
        FI;
    FI;
    IF NewRSP is non-canonical THEN
        #GP(EXT); (* Error code contains NULL selector *)
    FI;
    IF gate.IP is non-canonical THEN
        #GP(EXT);  (* Error code contains NULL selector *)
    FI;
```

```
RSP  := NewRSP & FFFFFFFFFFFFFFF0H;
SS   := NewSS;
SSdesc := const;
Push(SS);
Push(RSP);
Push(RFLAGS); (* 8-byte push *)
Push(CS);
PUSH(RIP);
Push(ErrorCode); (* If needed, 8-bytes *)
RIP := gate.RIP;
CS  := newCS;
IF ShadowStackEnabled(CPL) AND CPL == 3 THEN
   IA32_PL3_SSP := LA_adjust(SSP);
FI;
CPL := newCSdesc.DPL;
CS.RPL  := CPL;
IF ShadowStackEnabled(CPL) THEN
   oldSSP := SSP
   SSP    := NewSSP
   IF (SSP & 0x07 != 0) THEN
          #GP(0);
   FI
   IF (CS.L = 0 AND SSP[63:32] != 0) THEN
      #GP(0);
   FI
    FI;
expected_token_value := SSP;           (* busy bit- must be clear *)
new_token_value      := SSP | BUSY_BIT; (* Set the busy bit *)
IF (shadow_stack_lock_cmpxchg8b(SSP, new_token_value,
                        expected_token_value) !=
   expected_token_value) THEN
   #GP(0);
FI;
IF oldSS.DPL != 3
   ShadowStackPush8B(oldCS);
   ShadowStackPush8B(oldRIP);
   ShadowStackPush8B(oldSSP);
FI;
IF EndbranchEnabled (CPL)
   IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
   IA32_S_CET.SUPPRESS = 0
```

```
  FI;
  IF gate.type is intGate64 THEN
    RFLAGS.IF := 0 (* Interrupt flag set to 0, interrupts disabled *);
  FI;
  RFLAGS.TF := 0;
  RFLAGS.RF := 0;
  RFLAGS.NT := 0;
END;

INTRA-PRIVILEGE-LEVEL-INTERRUPT:
  NewSSP      := SSP;
  CHECK_SS_TOKEN := 0;
  IF gate.IST != 0 THEN
    TSSstackAddress := (IDT-descriptor IST « 3) + 28;
    IF (TSSstackAddress + 7) > TSS.limit  THEN
      #TS(error_code(current TSS selector,0,EXT));
    FI;
    NewRSP := 8 bytes loaded from (current TSS base +
                              TSSstackAddress);
  ELSE
    NewRSP := RSP;
  FI;
  IF ShadowStackEnabled(CPL) THEN
   NewSSPAddress  := IA32_INTERRUPT_SSP_TABLE_ADDR + (IDT gate IST « 3)
   NewSSP       := 8 bytes loaded from NewSSPAddress
   CHECK_SS_TOKEN := 1
  FI;
  IF NewRSP is non-canonical THEN
    #GP(EXT); (* Error code contains NULL selector *)
  FI;
  IF gate.RIP is non-canonical THEN
    #GP(EXT); (* Error code contains NULL selector *)
  FI;
  RSP := NewRSP & FFFFFFFFFFFFFFF0H;
  Push(SS);
  Push(RSP);
  Push(RFLAGS); // 8-byte push – including .IF, not affected by IOPL,CPL
  Push(CS);
  PUSH(RIP);
  Push(ErrorCode); (* If needed, 8-bytes *)
  oldCS  := CS;
```

```
oldRIP := RIP;
RIP := gate.RIP;
CS  := newCS;
CS.RPL := CPL;
IF ShadowStackEnabled(CPL) AND CHECK_SS_TOKEN == 1 THEN
   IF NewSSP & 0x07 != 0 THEN
     #GP(0);
       FI;
   IF (CS.L = 0 AND NewSSP[63:32] != 0) THEN
         #GP(0);
   FI;
   expected_token_value := NewSSP (* busy bit – (0)- must be clear *)
   new_token_value      := NewSSP | BUSY_BIT (* Set the busy bit *)
   IF shadow_stack_lock_cmpxchg8b(NewSSP, new_token_value,
                        expected_token_value) !=
     expected_token_value THEN
          #GP(0);
   FI;
FI;
IF ShadowStackEnabled(CPL) THEN
   (* Align to next 8 byte boundary *)
   tempSSP = SSP;
   Shadow_stack_store 4 bytes of 0 to (NewSSP − 4)
   SSP := newSSP & 0xFFFFFFFFFFFFFFF8H;
   ShadowStackPush8B(oldCS);
   ShadowStackPush8B(oldRIP);
   ShadowStackPush8B(tempSSP);
FI;
IF EndbranchEnabled (CPL)
   IF CPL == 3 THEN
      IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH;
      IA32_U_CET.SUPPRESS = 0;
   ELSE
      IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
      IA32_S_CET.SUPPRESS = 0;
   FI;
FI;
IF IDT gate is interrupt gate THEN
   RFLAGS.IF := 0; (* Interrupt flag set to 0; interrupts disabled *)
FI;
RFLAGS.TF := 0;
```

```
  RFLAGS.NT := 0;
  RFLAGS.RF := 0;
END;
```

4.2.6 IRET

IRET cannot enter 16-bit mode or VM86 mode. Task Descriptor access bits are not set. Mode restrictions are enforced. #NP and #SS are replaced with #GP.

```
IF EFLAGS.NT == 1 THEN
  #GP(0);
FI
tempRIP := POP(); // according to operand size
tempCS := POP(); // according to operand size
newCPL :=  tempCS.RPL
tempFlags := POP();// according to operand size

IF newCPL == 3 THEN
  tempFlags(VIP, VIF, IOPL) := (0, 0, 0);
ELSIF tempFlags(VIF, VIP, IOPL) != (0,0,0) THEN
   #GP(0);
FI

Check_selector(tempCS);
Descriptor := Load_descriptor_from_GDT_LDT(tempCS);
Check_CS_desc_for_IRET(tempCS, Descriptor, newCPL);
IF newCPL > CPL THEN
  IF CR4.FRED THEN
    #GP(tempCS);
  ELSE
    GOTO RETURN_TO_OUTER_PRIVLEDGE_LEVEL; // must be level 3
  FI
ELSIF newCPL < CPL THEN
  #GP(newCS);
ELSIF Started_in_64b_mode THEN
  GOTO RETURN_FROM_IA32e;
ELSE
  GOTO RETURN_FROM_SAME_PRIVLEDGE_LEVEL;
FI

RETURN_FROM_SAME_PRIVLEDGE_LEVEL:
  IF target mode is compat mode THEN
    tempRIP = tempRIP & 0xffffffff;
  FI
  IF tempRIP is not canonical THEN
    #GP(0); // Restoring RSP
  FI
  IF ShadowStackEnabled(CPL) THEN
    Perform normal Shadow Stack operations as described in the SDM;
  FI
  CS := tempCS;

  RIP := tempRIP;
  RFLAGS(CF, PF, AF, ZF, SF, TF, DF, OF, NT, RF, AC, IC) := tempFlags;
  IF CPL == 0 THEN EFLAGS(IF) := tempFlags; FI;
  Unmask NMI;
END;

RETURN_FROM_IA32e:
  tempRSP := POP();
  tempSS := POP();
  Check_sel(tempSS);
  tempSSdesc := Load_descriptor_from_GDT_LDT(tempSS);
  check_SS_desc(tempSS, tempSSdesc, newCPL); // Null handling is not required because IA32e is ring 3.
  IF Descriptor is compat mode THEN
   tempRSP = tempRSP & 0xffffffff;
  FI
  SS  := tempSS;
  RSP := tempRSP;
  GOTO RETURN_FROM_SAME_PRIVLEDGE_LEVEL;

RETURN_TO_OUTER_PRIVLEDGE_LEVEL:
  IF newCPL != 3 THEN
    #GP(tempCS);
  FI
  tempRSP := POP();
  tempSS := POP();
  Check_selector(tempSS);
  tempSSdesc := Load_descriptor_from_GDT_LDT(tempSS);
  check_SS_desc_for_IRET(tempSS, tempSSdesc, newCPL);
  IF Descriptor is compat mode THEN
     tempRSP = tempRSP & 0xffffffff;
  FI
  IF target mode is compat mode THEN
    tempRIP = tempRIP & 0xffffffff;
  FI
  IF tempRIP is not canonical THEN
   #GP(0); // Restoring RSP
  FI
  CPL := newCPL;
  IF ShadowStackEnabled() THEN
    Perform normal Shadow Stack operations as described in the SDM;
  FI
  CS  := tempCS;
  RIP := tempRIP;
  SS  := tempSS;
  RSP := tempRSP;
  Save CS.ARbyte
  RFLAGS(CF, PF, AF, ZF, SF, TF, DF, OF, NT, RF, AC, IC) := tempFlags;
```

```
    IF CPL == 0 THEN EFLAGS(IF) := tempFlags(IF); FI;
    Unmask NMI;
END;
```

4.2.7 JMP Far

Far JMPs are intra-level only. Mode restrictions are enforced. The selector must point to a non-conforming code descriptor in the GDT/LDT. The CS.accessed bit will not be set. With a 16-bit operand size the instruction raises an #UD exception. With the 0x67 prefix and when in 32-bit mode, the instruction raises a #GP(0) exception.

Pseudocode:

```
IF 16bit operand size THEN #UD ; FI


IF 0x67 prefix AND 32bit mode THEN #GP(0); FI
Check_selector(newCS);
newCSdesc := Load_descriptor_from_GDT_LDT(tempCS);
Check_CS_desc(tempCS, newCSdesc, CPL);
IF target mode is compat mode THEN
    newRIP = newRIP & 0xffffffff;
FI
IF newRIP is non-cannonical THEN
    #GP(0);
FI
CS  := newCS;
RIP := newRIP;
Save newCSdesc;
Do shadow stack pushes if enabled;
Do end branch state transition if enabled;
```

4.2.8 LSL, LAR, VERW, VERR

Simplified checks. LAR forces the returned access bit to 1.

```
Check_Selector(selector);
// If failure return with ZF := 0
Desc := Load_descriptor_from_GDT_LDT(selector);
// If failure return with ZF := 0
Check_Data_Desc(selector, Desc, CPL);
// if failure return with ZF := 0
// For LAR always return Access = 1
// LSL/LAR/VERW/VERR flow to return information from Desc
```

4.2.9 LDS, LES, LFS, LGS, LSS

The Desc.accessed bit will not be set. Use simplified checks.

Pseudocode:

```
If newSel is NULL AND LSS AND NOT (CPL0 AND CS.L) THEN
   #GP(0);
FI
Check_selector(newSel);
newDesc := Load_descriptor_from_GDT_LDT(newSel);
IF LSS THEN
   Check_SS_desc(newSEL, newDesc);
ELSE
   Check_Data_desc(newSel, newDesc);
FI
Dest(sel) := newSel;
Dest(offset) := offest;
Save newDesc;
```

4.2.10 LGDT

Behaves as described in the SDM.

4.2.11 LLDT

Loading a selector with bits [2:15] set to 0 will clear the LDT base and limit.

4.2.12 LIDT

Behaves as described in the SDM except that the Unusable bit is not set in the AR byte when the selector is NULL. Instead the limit is set to zero, which has the effect of causing a #GP when an access is made to a null LDT.

4.2.13 LKGS

Follows modified selector load checks, similar to MOV to segment register below.

4.2.14 LTR

Behaves as described in the SDM except that the BUSY bit is not checked or set in memory.

4.2.15 MOV from Segment Register

Behaves as described in the SDM.

4.2.16 MOV to Segment Register

Simplified checks.

```
If newSel is NULL AND MOV SS AND NOT (CPL0 AND CS.L) THEN
   #GP(0);
FI
```

```
Check_selector(newSel);
newDesc := Load_descriptor_from_GDT_LDT(newSel);
IF MOV SS THEN
    Check_SS_desc(newSEL, newDesc);
ELSE
    Check_Data_desc(newSel, newDesc);
Dest(sel) := newSel;
IF SS THEN
  MOV SS instruction blocking;
   Save Arbyte
FI;
```

4.2.17 POP Segment Register

Simplified checks.

```
IF 64b mode and POP DS, POP ES, POP SS THEN
    #UD;
FI
newSel := POP
Check_selector(newSel);
newDesc := Load_descriptor_from_GDT_LDT(newSel);
IF POP SS THEN
    Check_SS_desc(newSEL, newDesc);
ELSE
    Check_Data_desc(newSel, newDesc);
Dest:= newSel;
IF SS THEN
   Do POP SS blocking;
  Save Arbyte;
FI
```

4.2.18 POPF

The IOPL, VM, VIP, and VIF flags are always zero and are ignored on POPF.

4.2.19 PUSH Segment Selector

Behaves as described in the SDM.

4.2.20 PUSHF

Behaves as described in the SDM.

4.2.21 RDFSBASE, RDGSBASE

Behaves as described in the SDM.

4.2.22 RET far

Far RETs are intra-level only. The selector must point to a code descriptor in the GDT/LDT. The CS.accssed bit is not set. With a 16-bit operand size, the instruction raises an #UD exception.

```
IF 16bit operand size THEN #UD ; FI
newRIP := POP;
newCS  := POP;
If newCS is NULL THEN
   #GP(0)
FI
Check_selector(newCS);
newCSdesc := Load_descriptor_from_GDT_LDT(tempCS);
Check_CS_desc(tempCS, newCSdesc, CPL);
IF target mode is compat mode THEN
   newRIP = newRIP & 0xffffffff;
FI
IF newRIP is non-cannonical THEN
   #GP(0)
FI
CS  := newCS;
RIP := newRIP;
Save newCSdesc.ARbyte;
Do shadow stack if enabled
Do end branch state transition if enabled
```

4.2.23 SGDT

Behaves as described in the SDM.

4.2.24 SLDT

Behaves as described in the SDM.

4.2.25 SIDT

Behaves as described in the SDM.

4.2.26 STR

Behaves as described in the SDM.

4.2.27 SWAPGS

Behaves as described in the SDM and FRED EAS.

4.2.28 SYSCALL

Does not modify CS/SS.Base/Limit/Type/S/P/G.

4.2.29 SYSENTER

Does not modify CS/SS.Base/Limit/Type/S/P/G.

4.2.30 SYSEXIT

Does not modify CS/SS.Base/Limit/Type/S/P/G.

4.2.31 SYSRET

Does not modify CS/SS.Base/Limit/Type/S/P/G. Triggers #GP(0) if incoming EFLAGS (R11) has non-zero VIF, VIP, or IOPL.

4.2.32 WRFSBASE, WRGSBASE

Behaves as described in the SDM.

4.2.33 VMEntry

For each of CS, SS, DS, ES, FS, GS, TR, and LDTR fields are loaded from the VMCS guest state area as follows:

- TR and LDTR: the selector, base, and limit fields are loaded. The AR bytes including Unusable for TR and LDTR are ignored. The G bit is not used; the limit is always loaded as 32-bit.
- CS: The selector field is loaded, as well as the L bit from the access-rights field, and the D bit. The DPL field is checked, but not loaded. The D bit must be always NOT L. Other bits in the AR byte, including Unusable, are ignored.
- SS, DS, ES, FS, GS: The selector field is loaded. The SS DPL and B are loaded. For FS/GS the base is loaded. The AR bytes, including Unusable, for DS, ES, FS, and GS are ignored.

A VMEntry triggers an Invalid Guest State abort for the following conditions in VMCS:

- CS.L == 0 and CS.D == 0 (16-bit)
- CS.L == 1 and CS.D == 1 (invalid)
- CS.L == 0 and SS.DPL == 0 (32-bit ring 0)
- SS.DPL is 1 or 2
- SS.RPL != SS.DPL
- CS.RPL != SS.DPL
- TR.sel.TI != 0 (no TR in LDT)

- LDTR.sel.TI != 0
- LDTR base is not canonical
- There are no checks on data segments other than SS.

4.2.34 VMExit

For each of CS, SS, DS, ES, FS, GS, LDTR, GDTR, TR:

- For FS/GS/TR/LDTR/IDTR/GDTR the base fields are saved.
- For TR/LDTR/IDTR/GDTR the limit fields are saved. The limit is always saved as expanded 32-bit with the G bit never being set.
- For CS, the L bit is saved and the D bit is set to !L. CS.DPL is set to the value of SS.DPL. The other bits in the same field are undefined.
- For SS, the DPL and B bits are saved. The other bits in the same field are undefined.

For CS, SS, DS, ES, FS, GS, TR, GDTR:

- The selector is loaded from the host selector field. There is no concept of unusable for 0 selectors, except that loading NULL selectors for CS/TR fails consistency checks at entry.
- FS/GS/TR load from the host base following the same rules as Intel64. Other bases are ignored.
- TR limit is set to 0x67.
- SS.DPL is set to zero, SS.B is set to zero.

For LDTR the base and limit are set to zero. GDTR and IDTR base are loaded with their limits set to 0xFFFF.

4.2.35 STM Loading Host State for Dual Monitor Activation

The registers CS, SS, DS, ES, FS, GS are loaded as follows:

- The CS selector is set to 8.
- The selectors for SS/DS/ES/FS/GS are set to 16.
- The base addresses for FS/GS are set to 0.
- The CS.L bit is set to 1.
- CR4.FRED is cleared.
- SS.DPL and SS.B are set to 0

4.3 List of Segmentation Instructions and Associated Behavior

Table 26 gives a list of segmentation instructions and their behaviors.

Table 26. List of Segmentation Instructions and X86S Changes

Instruction	Behavior
SGDT	No change to Intel64 behavior.
SIDT	No change to Intel64 behavior.
SLDT	No change to Intel64 behavior.
STR	No change to Intel64 behavior.
LGDT	No change to Intel64 behavior.
LIDT	No change to Intel64 behavior.
LLDT	Loading a 0 descriptor will clear base/limit.
LTR	Does not check TSS.busy bit.
VERR	Behavior changed to follow the modified segmentation architecture.
VERW	Behavior changed to follow the modified segmentation architecture.
ARPL	No change to Intel64 behavior.
FAR CALL	Behavior changed to follow the modified segmentation architecture. #UD on 16-bit operand size. #GP on 0x67 prefix in 32-bit mode and indirect. Cannot change rings. Enforces mode restrictions.
FAR JMP	Behavior changed to follow the modified segmentation architecture. #UD on 16-bit operand size. #GP on 0x67 prefix in 32-bit mode. Cannot change rings.
FAR RET	Behavior changed to follow the modified segmentation architecture. #UD on 16-bit operand size. Cannot change rings. Enforces mode restrictions.
IRET	Only supports intra-ring and ring 0 to ring 3. Enforce mode and RFLAGS restrictions. Simplified checks.
LDS	Load far pointer in DS with simplified segment check rules
LES	Load far pointer in ES with simplified segment check rules
LFS	Load far pointer in FS with simplified segment check rules
LGS	Load far pointer in GS with simplified segment check rules
LSS	Load far pointer in SS with simplified segment check rules
LKGS	Move to Kernel GS Base with simplified segment check rules
MOV to DS	Move to DS with simplified segment check rules
MOV to ES	Move to ES with simplified segment check rules
MOV to SS	Move to SS with simplified segment check rules
MOV to FS	Move to FS with simplified segment check rules
MOV to GS	Move to GS with simplified segment check rules
MOV from DS	No change to Intel64 behavior.
MOV from ES	No change to Intel64 behavior.
MOV from SS	No change to Intel64 behavior.
MOV from FS	No change to Intel64 behavior.
MOV from GS	No change to Intel64 behavior.
POP DS	Pop top of stack into DS with simplified segment check rules
POP ES	Pop top of stack into ES with simplified segment check rules.
POP SS	Pop top of stack into SS with simplified segment check rules
POP FS	Pop top of stack into FS with simplified segment check rules
POP GS	Pop top of stack into GS with simplified segment check rules
PUSH CS	No change to Intel64 behavior.
PUSH DS	No change to Intel64 behavior.

Instruction	Behavior
PUSH ES	No change to Intel64 behavior.
PUSH SS	No change to Intel64 behavior.
PUSH FS	No change to Intel64 behavior.
PUSH GS	No change to Intel64 behavior.
SWAPGS	No change to Intel64 behavior.
RSM	No changes to segmentation but enforces other mode and RFLAGS restrictions.
WRFSBASE	No change to Intel64 behavior.
WRGSBASE	No change to Intel64 behavior.
RDFSBASE	No change to Intel64 behavior.
RDGSBASE	No change to Intel64 behavior.
SYSENTER	Cannot enter 16-bit mode or VM86.
SYSEXIT	Cannot enter 16-bit mode or VM86.
SYSCALL	Enforces RFLAGS restrictions.
SYSRET	Enforces RFLAGS restrictions.
ERETU	Modified segment check rules. Enforces RFLAGS restrictions.
FRED entry	No change to Intel64 behavior.
IDT entry	Modified segment check rules and FRED restrictions.

4.4 64-Bit SIPI Without LEGACY_REDUCED_OS_ISA

64-bit SIPI can be implemented on systems that do not set the LEGACY_REDUCED_OS_ISA CPUID bit to allow compatibility to X86S systems. In this case, not enabling 64-bit SIPI in the IA32_SIPI_ENTRY_STRUCT_PTR or in the SIPI_ENTRY_STRUCT FEATURES bit will fall back to legacy INIT/SIPI.