written. Instead, the PDPEs are accessed on demand as part of a table walk. This has the side-effect that illegal bit combinations in the PDPEs are not signaled at the time that CR3 is written, but instead when the faulty PDPE is accessed as part of a table walk.

This means that an operating system cannot rely on the behavior when the in-memory PDPEs are different than the in-processor copy.

15.25.11 A20 Masking

There is no provision for applying A20 masking to guest physical addresses; the VMM can emulate A20 masking by changing the nested page mappings accordingly.

15.25.12 Detecting Nested Paging Support

Nested Paging is an optional feature of SVM and is not available in all implementations of SVM-capable processors. The CPUID instruction should be used to detect nested paging support on a particular processor. See Section 3.3, “Processor Feature Identification,” on page 63 for more information on using the CPUID instruction.

15.26 Security

SVM provides additional hardware support that is designed to facilitate the construction of trusted software systems. While the security features described in this section are orthogonal to SVM’s virtualization support (and are not required for processor virtualization), the two form building blocks for trusted systems.

SKINIT Instruction. The SKINIT instruction and associated system support (the Trusted Platform Module or TPM) are designed to allow for verifiable startup of trusted software (such as a VMM), based on secure hash comparison.

Security Exception. A security exception (#SX) is used to signal certain security-critical events.

15.27 Secure Startup with SKINIT

The SKINIT instruction is one of the keys to creating a “root of trust” starting with an initially untrusted operating mode. SKINIT reinitializes the processor to establish a secure execution environment for a software component called the secure loader (SL) and starts execution of the SL in a way that cannot be tampered with. SKINIT also copies the secure loader executable image to an external device, such as a Trusted Platform Module (TPM) for verification using unique bus transactions that preclude SKINIT operation from being emulated by software in a way that the TPM could not readily detect. (Detailed operation is described in Section 15.27.4.)

15.27.1 Secure Loader

A secure loader (SL) typically initializes SVM hardware mechanisms and related data structures, and initiates execution of a trusted piece of software such as a VMM (referred to as a Security Kernel, or SK, in this document), after first having validated the identity of that software.

SKINIT allows SVM protections to be reliably enabled after the system is already up and running in a non-trusted mode — there is no requirement to change the typical x86 platform boot process.

Exact details of the handoff from the SL to an SK are dependent on characteristics of the SL, SK and the initial untrusted operating environment. However, there are specific requirements for the SL image, as described in Section 15.27.2.

15.27.2 Secure Loader Image

The secure loader (SL) image contains all code and initialized data sections of a secure loader. This code and initial data are used to initialize and start a security kernel in a completely safe manner, including setting up DEV protection for memory allocated for use by SL and SK. The SL image is loaded into a region of memory called the secure loader block (SLB) and can be no larger than 64Kbyte (see “Secure Loader Block” on page 502). The SL image is defined to start at byte offset 0 in the SLB.

The first word (16 bits) of the SL image must specify the SL entry point as an unsigned offset into the SL image. The second word must contain the length of the image in bytes; the maximum length allowed is 65535 bytes. These two values are used by the SKINIT instruction. The layout of the rest of the image is determined by software conventions. The image typically includes a digital signature for validation purposes. The digital signature hash must include the entry point and length fields. SKINIT transfers the SL image to the TPM for validation prior to starting SL execution (see “SKINIT Operation” on page 504 for further details of this transfer). The SL image for which the hash is computed must be ready to execute without prior manipulation.

15.27.3 Secure Loader Block

The secure loader block is a 64Kbyte range of physical memory which may be located at any 64Kbyte-aligned address below 4Gbyte. The SL image must have been loaded into the SLB starting at offset 0 before executing SKINIT. The physical address of the SLB is provided as an input operand (in the EAX register) to SKINIT, which sets up special protection for the SLB against device accesses (i.e., the DEV need not be activated yet).

The SL must be written to execute initially in flat 32-bit protected mode with paging disabled. A base address can be derived from the value in EAX to access data areas within the SL image using base+displacement addressing, to make the SL code position-independent.

Memory between the end of the SL image and the end of the SLB may be used immediately upon entry by the SL as secure scratch space, such as for an initial stack, before DEV protections are set up for the rest of memory. The amount of space required for this will limit the maximum size of the SL image,

and will depend on SL implementation. SKINIT sets the ESP register to the appropriate top-of-stack value (EAX + 10000h).

Figure 15-14 on page 503 illustrates the layout of the SLB, showing where EAX and ESP point after SKINIT execution. Labels in italics indicate suggested uses; other labels reflect required items.

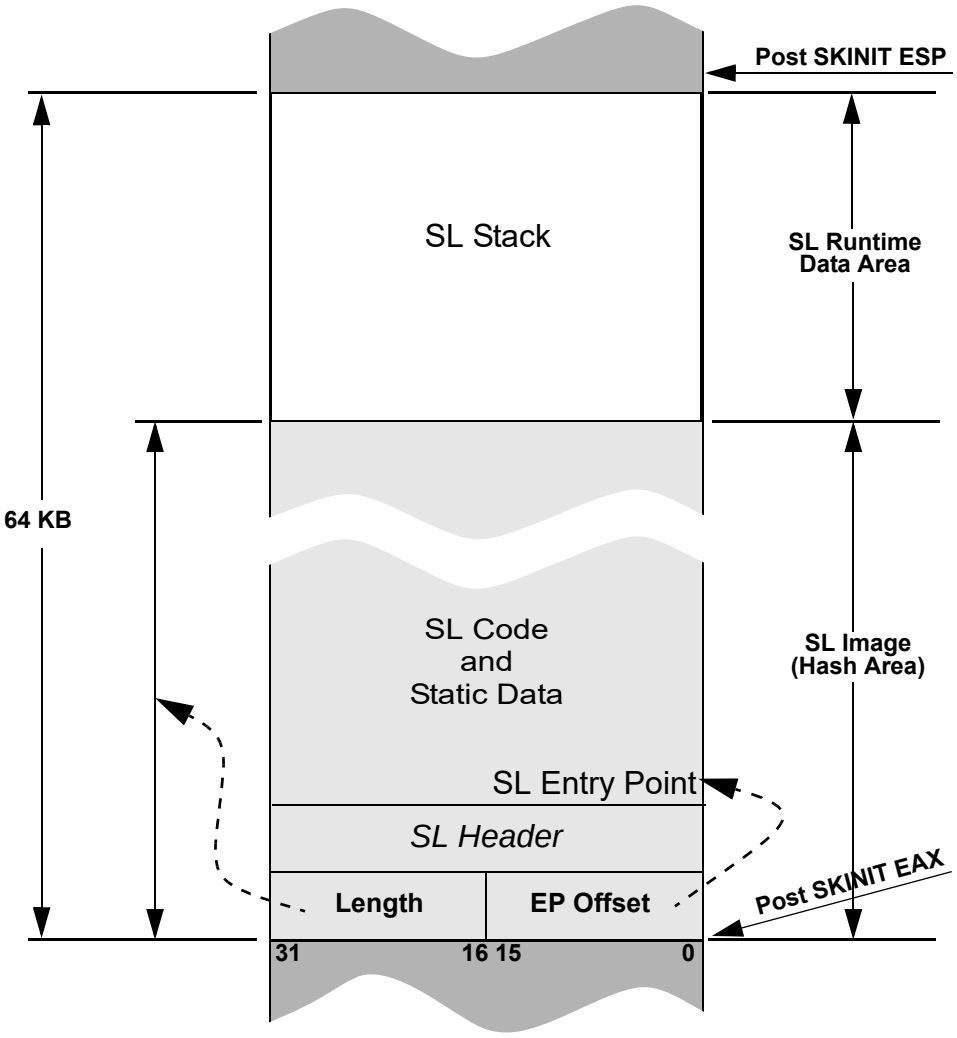

Figure 15-14. SLB Example Layout

15.27.4 Trusted Platform Module

The trusted platform module, or TPM, is an essential part of full trusted system initialization. This device is attached to an LPC link off the system I/O hub. It recognizes special SKINIT transactions,

receives the SL image sent by SKINIT and verifies the signature. Based on the outcome, the device decides whether or not to cooperate with the SL or subsequent SK. The TPM typically contains sealed storage containing cryptographic keys and other high-security information that may be specific to the platform.

15.27.5 System Interface, Memory Controller and I/O Hub Logic

SKINIT uses special support logic in the processor's system interface unit, the internal controller and the I/O hub to which the TPM is attached. SKINIT uses special transactions that are unique to SKINIT, along with this support logic, designed to securely transmit the SL Image to the TPM for validation.

The use of this special protocol is intended to allow the TPM to detect true execution, as opposed to emulation, of a trusted Secure Loader, which in turn provides a means for verifying the subsequent loading and startup of a trusted Security Kernel.

15.27.6 SKINIT Operation

The SKINIT instruction is intended to be used primarily in normal mode prior to the VMM taking control.

SKINIT takes the physical base address of the SLB as its only input operand in EAX, and performs the following steps:

1. Reinitialize processor state in the same manner as for the INIT signal, then enter flat 32-bit protected mode with paging off. The CS selector is set to 8h and CS is read only. The SS selector is set to 10h and SS is read/write and expand-up. The CS and SS bases are cleared to 0 and limits are set to 4G. DS, ES, FS and GS are left as 16-bit real mode segments and the SL must reload these with protected mode selectors having appropriate GDT entries before using them. Initialized data in the SLB may be referenced using the SS segment override prefix until DS is reloaded. The general purpose registers are cleared except for EAX, which points to the start of the secure loader, EDX, which contains model, family and stepping information, and ESP, which contains the initial stack pointer for the secure loader. Cache contents remain intact, as do the x87 and SSE control registers. Most MSRs also retain their values, except those which might compromise SVM protections. The EFER MSR, however, is cleared. The DPD, R_INIT and DIS_A20M flags in the VM_CR register are unconditionally set to 1.
2. Form the SLB base address by clearing bits 15:0 of EAX (EAX is updated), and enable the SL_DEV protection mechanism (see "Secure Initialization Support" on page 493) to protect the 64-Kbyte region of physical memory starting at the SLB base address from any device access.
3. In multiprocessor operation, perform an interprocessor handshake as described in Section 15.27.8 on page 505.
4. Read the SL image from memory and transmit it to the TPM in a manner that cannot be emulated by software.
5. Signal the TPM to complete the hash and verify the signature. If any failures have occurred along the way, the TPM will conclude that no valid SL was started.

6. Clear the Global Interrupt Flag. This disables all interrupts, including NMI, SMI and INIT and ensures that the subsequent code can execute atomically. If the processor enters the shutdown state (due to a triple fault for instance) while GIF is clear, it can only be restarted by means of a RESET.
7. Update the ESP register to point to the first byte beyond the end of the SLB (SLB base + 65536), so that the first item pushed onto the stack by the SL will be at the top of the SLB.
8. Add the unsigned 16-bit entry point offset value from the SLB to the SLB base address to form the SL entry point address, and jump to it.

The validation of the SL image by the TPM is a one-way transaction as far as SKINIT is concerned. It does not depend on any response from the TPM after transferring the SL image before jumping to the SL entry point, and initiates execution of the Secure Loader unconditionally. Because of the processor initialization performed, SKINIT does not honor instruction or data breakpoint traps, or trace traps due to EFLAGS.TF.

Pending interrupts. Device interrupts that may be pending prior to SKINIT execution due to EFLAGS.IF being clear, or that assert during the execution of SKINIT, will be held pending until software subsequently sets GIF to 1. Similarly, SMI, INIT and NMI interrupts that assert after the start of SKINIT execution will also be held pending until GIF is set to 1.

Debug Considerations. SKINIT automatically disables various implementation-specific hardware debug features. A debug version of the SL can reenable those features by clearing the VM_CR.DPD flag immediately upon entry.

15.27.7 SL Abort

If the SL determines that it cannot properly initialize a valid SK, it must cause GIF to be set to 1 and clear the VM_CR MSR to re-enable normal processor operation.

15.27.8 Secure Multiprocessor Initialization

The following standard APIC features are used for secure MP initialization:

- The concept of a single Bootstrap Processor (BSP) and multiple Application Processors (APs).
- The INIT interprocessor interrupt (IPI), which puts the target processors into a halted state (INIT state) which is responsive only to a subsequent Startup IPI.
- The Startup IPI causes target processors to begin execution at a location in memory that is specified by the Boot Processor and conveyed along with the Startup IPI. The operation of the processor in response to a Startup IPI is slightly modified to support secure initialization, as described below.

A Startup IPI normally causes an AP to start execution at a location provided by the IPI. To support secure MP startup, each AP responds to a startup IPI by additionally clearing its GIF and setting the DPD, R_INIT and DIS_A20M flags in the VM_CR register if, and only if, the BSP has indicated that it has executed an SKINIT. All other aspects of Startup IPI behavior remain unchanged.

Software Requirements for Secure MP initialization. The driver that starts the SL must execute on the BSP. Prior to executing the SKINIT instruction, the driver must save any processor-specific system register contents to memory for restoration after reinitialization of the APs. The driver should also put all APs in an idle state. The driver must first confirmed that all APs are idle and then it must issue an INIT IPI to all APs and wait for its local APIC busy indication to clear. This places the APs into a halted state which is responsive only to a subsequent Startup IPI. APs will still respond to snoops for cache coherency. The driver may execute SKINIT at any time after this point. Depending on processor implementation, a fixed delay of no more than 1000 processor cycles may be necessary before executing SKINIT to ensure reliable sensing of APIC INIT state by the SKINIT.

AP Startup Sequence. While the SL starts executing on the BSP, the APs remain halted in APIC INIT state. Either the SL or the SK may issue the Startup IPI for the APs at whatever point is deemed appropriate. The Startup IPI conveys an 8-bit vector specified by the software that issues the IPI to the APs. This vector provides the upper 8 bits of a 20-bit physical address. Therefore, the AP startup code must reside in the lower 1Mbyte of physical memory—with the entry point at offset 0 on that particular page.

In response to the Startup IPI, the APs start executing at the specified location in 16-bit real mode. This AP startup code must set up protections on each processor as determined by the SL or SK. It must also set GIF to re-enable interrupts, and restore the pre-SKINIT system context (as directed by the SL or SK executing on the BSP), before resuming normal system operation.

The SL must guarantee the integrity of the AP startup sequence, for example by including the startup code in the hashed SL image and setting up DEV protection for it before copying it to the desired area. The AP startup code does not need to (and should not) execute SKINIT. Care must also be taken to avoid issuing another INIT IPI from any processor after the BSP executes SKINIT and before all APs have received a Startup IPI, as this could compromise the integrity of AP initialization.

Pending interrupts. Device interrupts that may be pending on an AP prior to the APIC INIT IPI due to EFLAGS.IF being clear, or that assert any time after the processor has accepted the INIT IPI, will be held pending through the subsequent Startup IPI, and remain pending until software sets GIF to 1 on that AP. Similarly, SMI, INIT, and NMI interrupts that assert after the processor has accepted the INIT IPI will also be held pending until GIF is set to 1.

Aborting MP initialization. In the event that the SL or SK on the BSP decides to abort SVM system initialization for any reason, the following clean-up actions must be performed by SL code executing on each processor before returning control to the original operating environment:

- The BSP and all APs that responded to the Startup IPI must restore GIF and clear VM_CR on each processor for normal operation.
- For each processor that has a distinct memory controller associated with it, the SL_DEV_EN flag in the DEV control register must be cleared in order to restore normal device accessibility to the 64KB SL memory range.

Any secure context created by the SL that should not be exposed to untrusted code should be cleaned up as appropriate before these steps are taken.

15.28 Security Exception (#SX)

The Security Exception fault signals security-sensitive events that occur while executing the VMM, in the form of an exception so that the VMM may take appropriate action. (A VMM would typically intercept comparable sensitive events in the guest.) In the current implementation, the only use of the #SX is to redirect external INITs into an exception so that the VMM may — among other possibilities — destroy sensitive information before re-issuing the INIT, this time without redirection. The INIT redirection is controlled by the VM_CR.R_INIT bit.

The #SX exception dispatches to vector 30, and behaves like other fault-class exceptions such as General Protection Fault (#GP). The #SX exception pushes an error code. The only error code currently defined is 1, and indicates redirection of INIT has occurred.

The #SX exception is a contributory fault.

15.29 Advanced Virtual Interrupt Controller

The AMD Advanced Virtual Interrupt Controller (AVIC) is an important enhancement to AMD Virtualization™ Technology (AMD-V). In a virtualized environment, AVIC presents to each guest a virtual interrupt controller that is compliant with the local Advanced Programmable Interrupt Controller (APIC) architecture. See Chapter 16, “Advanced Programmable Interrupt Controller (APIC),” on page 547 for a detailed description of APIC.

15.29.1 Introduction

In a virtualized computer system, each guest operating system needs access to an interrupt controller to send and receive device and interprocessor interrupts. When there is no hardware acceleration, it falls to the virtual machine monitor (VMM) to intercept guest-initiated attempts to access the interrupt controller registers and provide direct emulation of the controller system programming interface allowing the guest to initiate and process interrupts. The VMM uses the underlying physical and virtual interrupt delivery mechanisms of the system to deliver interrupts from I/O devices and virtual processors to the target guest virtual processor and to handle any required end of interrupt processing.

Given the high rate of device and interprocessor interrupt generation in modern computer systems, the emulation of a local APIC is a significant burden for the VMM.

AVIC architecture addresses the overhead of guest interrupt processing in a virtualized environment by applying hardware acceleration to the following components of interrupt processing:

- Providing a guest operating system access to performance-critical interrupt controller registers
- Initiating intra- and inter-processor interrupts (IPIs) in and between virtual processors in a guest

Acceleration of the delivery of virtual interrupts from I/O devices to virtual processors is not addressed directly by AVIC hardware. This acceleration would be provided by an I/O memory management unit (IOMMU). The AVIC architecture is compatible with the AMD I/O Memory Management Unit