Alternate Injection Support for SEV-SNP Virtual Machines

Jon Lange (jlange@microsoft.com)

June 3, 2024

Background

The Alternate Injection feature of SEV-SNP enhances the security of a confidential VM by preventing the untrusted host from presenting unexpected interrupts or exceptions, while still preserving the standard interrupt dispatch semantics inherent in the x86 architecture. Use of Alternate Injection requires management logic within the guest running at a higher VMPL, such as an SVSM, and also requires cooperation from the host to ensure that pending interrupts can be signaled through the management VMPL. This document describes extensions to the GHCB and SVSM specifications that enable support for the use of Alternate Injection.

Interrupt Delivery Model

Requirements

The primary motivation for supporting Alternate Injection is to ensure that the untrusted hypervisor cannot inject arbitrary interrupt or exception vectors into a guest VM, instead ensuring that interrupt presentation can only be performed by a trusted entity (the SVSM). Satisfying the goal of insulating the guest OS from malicious interrupts injected by the hypervisor, requires insulating the SVSM itself from malicious interrupts injected by the hypervisor. This means that an SVSM that wishes to make use of Alternate Injection should itself be prepared to enforce the use of Restricted Injection. Consequently, any hypervisor that offers support for Alternate Injection must also support Restricted Injection.

Because the SVSM requires its own interrupt sources in addition to the interrupt sources associated with the guest OS, the hypervisor is expected to track a separate set of interrupt sources for each VMPL enabled for a given vCPU. Interrupts associated with GHCB requests initiated by the SVSM (e.g. #HV IPI and #HV timers) are assigned vectors according to the GHCB ABI. Other interrupt sources managed by the host, such as for virtual devices, are associated with the appropriate VMPL using a host-specific virtualization interface not documented here; this document assumes only that any such host-specific interface will carefully prescribe such VMPL association so there is no ambiguity on the part of guest OS or SVSM software.

SVSM involvement is required to present interrupts to the guest OS, so when the hypervisor detects that a guest OS interrupt has become signaled and should be offered to the guest, it is necessary to cause the SVSM to run. Consequently, some sort of preemption mechanism is required to cause the SVSM to take control when required to facilitate handling of guest interrupt state. To take advantage of existing preemption mechanisms, the hypervisor will treat such notification as an interrupt source, and the SVSM must register an interrupt vector with the hypervisor, to be delivered using the existing Restricted Injection interrupt delivery model. The hypervisor is expected to cause the SVSM to run any time delivery of an interrupt would result in injection of #HV to the SVSM.

GHCB Changes

Extended Interrupt Information

The interrupt descriptor at the base of the #HV doorbell page is extended to include two 16-bit words which convey additional information about interrupts or events related to interrupt processing in lower VMPLs. The first 16-bit word ("PendingEvent") retains its existing meaning. The second 16-bit word is defined as "InjectionInfo", and its bits are defined to have the following meaning.

- Bit [0] is NoEoiRequired as defined in the current GHCB specification.
- Bits [7:1] are reserved for future use.
- Bit [8] indicates that interrupt information is available for VMPL 1.
- Bit [9] indicates that interrupt information is available for VMPL 2.
- Bit [10] indicates that interrupt information is available for VMPL 3.
- Bits [15:11] are reserved for future use.

To indicate separate interrupt sources for each VMPL, the #HV doorbell page is further extended to define the first 256 bytes. Bytes 0-31 contain interrupt information associated with SVSM itself. Bytes 32-63 are reserved for future use. Bytes 64-97 contain the extended interrupt descriptor (defined below) associated with VMPL 1. Bytes 128-159 contain the extended interrupt descriptor associated with VMPL 2. Bytes 192-223 contain the extended interrupt descriptor associated with VMPL 3.

Whenever the hypervisor delivers an interrupt for a lower VMPL (or an event, such as a request to revert to host-managed APIC emulation), it will set the appropriate interrupt bit(s) in the extended interrupt descriptor corresponding to the lower VMPL, and it will also set bit 8, 9, or 10 in the second 16-bit word of the SVSM's extended interrupt descriptor to indicate which VMPL has pending interrupt evaluation work. If the bit corresponding to VMPL activity transitions from zero to one, the hypervisor will send an interrupt to the SVSM indicating that guest interrupt processing is required.

The extended interrupt descriptor associated with each non-zero VMPL is defined to include sixteen 16-bit words (for a total of 32 bytes) which together comprise a 256-bit vector describing pending interrupts. Only interrupt vectors in the range 31-255 may be represented in this vector, leaving the low 31 bits available for additional information. Presenting multiple interrupt sources at once permits efficient delivery of interrupts by the SVSM to the guest OS by minimizing the number of interactions that must occur with the hypervisor.

The first 16-bit word in the extended interrupt descriptor, corresponding to bits 0-15, is defined as follows.

- Bits [7:0] describe the vector of a single pending interrupt, which must have a value in the range 31-255. This field is used if only a single vector is pending, or if any level-sensitive interrupt is pending.
- Bit [8] indicates that an NMI is pending.
- Bit [9] indicates that a virtual #MC is pending.
- Bit [10] indicates the trigger mode of the interrupt described in bits [7:0]. A value of one indicates a level-sensitive interrupt, and a value of zero indicates an edge-triggered interrupt.
- Bits [13:11] are reserved for future use.
- Bit [14] indicates whether any interrupt is indicated in bits 31-255 of the extended interrupt descriptor. Only edge-triggered interrupts may be signaled in the extended interrupt descriptor.
- Bit [15] is reserved for future use.

The 16-bit word at byte offset 2, corresponding to bits 16-31, has the following meaning.

- Bits [14:0] are defined for future use.
- Bit [15] corresponds to vector 31 in the 256-bit extended interrupt descriptor.

The 16-bit word at byte offset 4 corresponds to vectors 32-47 in the extended interrupt descriptor, and subsequent 16-bit words have similar meanings.

When the hypervisor presents any edge-triggered interrupt, it will ensure that either a single interrupt is described in bits 7:0 of the page, or that bit 14 is set and the corresponding bits are set in bits 31-255 of the page. Example pseudo-code is included in this document.

When the hypervisor presents a level-sensitive interrupt, it will set bits 7:0 to indicate the highest priority level-sensitive interrupt in progress, as well as setting bit 10 to indicate that a level-sensitive interrupt is pending. The hypervisor may change the value of bits 7:0 if a higher priority level-sensitive interrupt is presented. Example pseudo-code is included in this document.

When the SVSM wants to consume an interrupt for a lower VMPL, it should first check to see whether bits 7:0, as well as bits 10 and 14, describe a single, edge-triggered interrupt. If so, the SVSM can consume the interrupt by atomically clearing bits 7:0, ensuring that bits 10 and 14 have not changed; if this succeeds, the hypervisor will consider the interrupt consumed without requiring any EOI. If bit 14 indicates that multiple interrupts are present, the SVSM can atomically clear bit 14, and then sweep bits 31-255 looking for specific interrupt vectors, atomically clearing those that it finds; these interrupts are considered consumed without requiring any EOI. If the

SVSM observes that a level-sensitive interrupt is present, it should consume what it finds in bits 7:0 and track a level-sensitive interrupt in progress. When handling an EOI for a level-sensitive interrupt, the SVSM must send an explicit EOI to the hypervisor. To resolve possible race conditions between an EOI request and subsequent delivery of a higher-priority level-sensitive interrupt, all level-sensitive EOI requests will specify the vector for which the EOI applies, using a new #HV EOI call. Example pseudo-code is included in this document.

Negotiating Functionality

Enabling Alternate Injection

Hypervisor support for extended interrupt information and Alternate Injection is enumerated in the GHCB FEATURES bitmask. Bit 7 indicates the availability of extended interrupt information. If bit 7 is set, the SVSM can configure interrupt policy to enable the use of Alternate Injection.

If an SVSM detects support for extended information, then the SVSM can request the Alternate Injection feature during VMSA creation. Any time a VMSA is created using the SNP AP Creation GHCB call, and the new VMSA specifies the Alternate Injection feature in the SEV feature mask, the hypervisor will activate extended interrupt delivery for the vCPU.

Alternate Injection is not supported for VMPL 0 itself. Specifying Alternate Injection for any VMSA associated with VMPL 0 is considered an error. In addition, Alternate Injection is not supported unless VMPL 0 is using Restricted Injection. Specifying Alternate Injection for any non-zero VMSA associated with a vCPU that is not using Restricted Injection for VMPL 0 is considered an error.

Supporting Multiple Guest Runtimes

Code executing within the guest OS VMPL typically spans multiple runtime components, such as a virtual firmware as well as an operating system, and each of these components may have different capabilities with respect to APIC handling: some components may know how to make use of the SVSM APIC Protocol and some may rely on the hypervisor to emulate the APIC using the existing X2APIC MSRs conveyed via the GHCB protocol. It must be possible to permit different configurations for each of these components.

The SVSM is required to have knowledge of whether the first component that executes can support the Alternate Injection protocol. If the SVSM does not know that the first component to execute will support the SVSM APIC Protocol, then it must not enable Alternate Injection in the guest VMSA. If the SVSM does know that the first component to execute will support the SVSM APIC Protocol, then it will enable Alternate Injection prior to the first guest entry; the first guest component must either make use of the protocol or it must request that the protocol be disabled. The initial version of the SVSM APIC Protocol permits the enumeration of features and the ability to control their use so that the guest can make this determination and configure itself as required.

When there is a handoff from one component to another (such as from a UEFI virtual firmware to a guest operating system), the component terminating its execution must be prepared to disable the use of Alternate Injection unless it knows that the component taking over is prepared to continue use of Alternate Injection. Since there may not be a mechanism for the two components to negotiate this directly, the SVSM APIC Protocol supports a registration mechanism to determine the behavior across the handoff. Each component that desires to make use of the SVSM APIC Protocol will register its intention to use the protocol, and the SVSM will keep a count of the number of registered requests. As long as the registration count remains non-zero, Alternate Injection will remain active. Once the registration drops to zero, Alternate Injection will be disabled, and cannot be enabled again.

The following flow illustrates how this mechanism might be used in the case of a transition from a UEFI-based firmware to a guest OS.

- Prior to executing ExitBootServices, a guest OS that understands the SVSM APIC Protocol should detect the availability of that protocol and determine whether it is usable. If so, it should register its use of the SVSM APIC Protocol.
- During the execution of ExitBootServices, the firmware (which does not know the intention of the guest OS) should deregister its use of the SVSM APIC Protocol. If the guest OS registered its intention to use Alternate Injection, this call will not affect the current state of Alternate Injection. If the guest OS did not declare its intention to use Alternate Injection, then this call will disable Alternate Injection, and the guest OS can use the X2APIC GHCB protocol as it expects.

Multi-Processor Considerations

The registration count of Alternate Injection is kept as a single count for the entire VM, but Alternate Injection is maintained per vCPU. Consequently, when the guest wishes to deregister its use of Alternate Injection, it must first call the SVSM to perform the deregistration, and after that it must call the SVSM on each active vCPU to update the Alternate Injection state based on the current registration state.

The Alternate Injection state is normally expected to be consistent across processors, except for those periods of time when all vCPUs are in the process of disabling Alternate Injection. Consequently, consistency will be enforced when additional vCPUs are created through the SVSM Core Protocol. When creating a new vCPU via the Core Protocol, the supplied VMSA must set its SEV features to match the Alternate Injection state of the calling vCPU. If a vCPU that has enabled Alternate Injection supplies a VMSA with Alternate Injection disabled, or if a vCPU that has disabled Alternate Injection supplies a VMSA with Alternate Injection enabled, the Create vCPU call will fail with SVSM_ERR_INVALID_PARAMETER.

Scheduling Considerations

When Alternate Injection is active, the SVSM is solely responsible for causing interrupts to be presented to a lower VMPL, but the SVSM is not always aware of the precise time that such interrupts are scheduled for delivery by the hypervisor. It is possible that additional interrupts become scheduled while the SVSM is executing its exit flow to return to a lower VMPL, and has already passed the point where it has made its determination about whether to deliver an interrupt to a lower VMPL. Consequently, the SVSM must be designed so that once it passes the point where it commits to entering the lower VMPL, any delivery of #HV that indicates interrupt delivery to the lower VMPL will cause it to cancel the VMGEXIT that would cause a return to the lower VMPL, so that it can process the interrupt that has been signaled.

Changes to GHCB Guest Non-Automatic Exits

Configure Injection Notification Vector (0x8000_0019)

This call configures the interrupt vector that will be signaled by the hypervisor whenever it needs to notify the SVSM that interrupt injection processing is required.

Upon entry to the hypervisor, SW_EXITINFO1 is defined such that bits [7:0] contain the interrupt vector that will be signaled when the SVSM is notified of pending injection processing; other bits are reserved and should be zero. This interrupt will be delivered as an edge-triggered interrupt.

This call can only be issued by VMPL 0. If this call is issued by any other VMPL, it is considered an error.

Disable Alternate Injection (0x8000_001A)

This call requests the hypervisor to disable Alternate Injection for a specific VMPL. Any interrupts present in the appropriate extended interrupt vector will be placed into the IRR of the hypervisor-emulated APIC and delivered to the target VMPL using direct event injection. As with proxy interrupt messages sent by the hypervisor via the #HV doorbell page, if bits 10 and 14 are clear, then bits 7:0 specify a single, edge-triggered interrupt vector pending delivery, or zero if no interrupt is pending delivery. If bit 14 is set, then only those bits in the extended interrupt vector corresponding to interrupts in the range 31-255 will be placed into the IRR as edge-triggered interrupts. If bit 10 is set, then the interrupt vector indicated by bits 7:0 will be placed into the IRR as a level-sensitive interrupt (the hypervisor is already required to keep track of any other level-sensitive interrupts that are pending delivery. Reverting to hypervisor APIC emulation also requires knowledge of which interrupt(s), if any, have been placed into service. Therefore, the 32 bytes that follow the extended interrupt vector associated with the target VMPL are defined as an ISR vector, indicating which edge-triggered interrupts are already in service (the hypervisor is already required to keep track of which level-sensitive interrupts are currently in service). Only bits corresponding to interrupt vectors in the range 31-255 have meaning; the other bits are reserved. The SVSM must clear any data in this ISR area before writing ISR bits.

This call can only be issued by VMPL 0. If this call is issued by any other VMPL, it is considered an error.

Upon entry to the hypervisor, SW_EXITINFO1 is defined as follows.

- SW_EXITINFO1[19:16] – VMPL for which Alternate Injection is being deactivated.
- SW_EXITINFO1[15:8] – Virtual TPR for the target VMPL.
- SW_EXITINFO1[1] - Interrupt shadow state of the target VMPL.

- SW_EXITINFO1[0] - EFLAGS_IF value of the target VMPL.

#HV Timer (0x8000_0016)

The #HV Timer NAE currently defined is extended to permit signaling of timer interrupts by a lower VMPL. Not all SVSM implementations of Alternate Injection support will choose to implement the timer features of the APIC. In this case, the lower VMPL will continue to make use of the #HV Timer Guest NAE calls to configure timer support. The hypervisor must always interpret this NAE to be specific to the calling VMPL, and must not permit timer requests from VMPL to affect timer state of another VMPL. If the hypervisor determines that the timer interrupt is due for a VMPL that has Alternate Injection active, then it will schedule delivery of the interrupt specified in the Timer LVT entry by proxying that interrupt to the SVSM.

From the point of view of the guest, any timer signaled in this way will appear to be a host-generated interrupt, so the vector must be configured by the guest OS as an allowed vector using the APIC Protocol.

Specific EOI (0x8000_001B)

This call requests the hypervisor to perform an EOI cycle on a level-sensitive interrupt. Such an EOI must be performed on a specific EOI to avoid race conditions in which the hypervisor presents a new level-sensitive interrupt while the SVSM is attempting to complete an EOI on a lower-priority level-sensitive interrupt.

Upon entry to the hypervisor, SW_EXITINFO1 is defined as follows:

- SW_EXITINFO1[19:16] – VMPL for which the EOI applies.
- SW_EXITINFO1[7:0] – The vector for which the EOI applies.

All other bits in SW_EXITINFO1 are reserved and must contain zero. SW_EXITINFO2 must contain zero.

SVSM Protocol Changes

When Alternate Injection is active, only the SVSM can manage interrupt delivery for a lower VMPL. A new SVSM protocol is defined to enable the lower VMPL to configure its APIC state through the SVSM.

In general, the APIC configuration protocol simply requires routing APIC register read and write requests to the SVSM. However, a guest OS generally expects that only certain interrupt vectors can be signaled via the APIC (a guest OS may, for example, assume that certain IDT vectors can only be invoked through execution of a software interrupt, and never through an external hardware interrupt), and therefore the untrusted hypervisor must not have the ability to signal arbitrary interrupt vectors to the lower VMPL. Therefore, the APIC protocol also requires the guest OS to advise the SVSM of which interrupt vectors are permissible to deliver.

SVSM APIC Protocol

Protocol number 3 is defined as the APIC Protocol.

The APIC Protocol is supported only as long as Alternate Injection is enabled. If the SVSM is unable to enable Alternate Injection because the hypervisor does not support the necessary guest/host interaction protocols, the SVSM will suppress availability of the APIC Protocol. Similarly, if Alternate Injection is ever disabled as the result of a guest call, the SVSM will suppress availability of the SVSM protocol on the calling vCPU.

Query Features (Call 0)

This call (SVSM_APIC_QUERY_FEATURES) permits the guest to determine which APIC features are supported. If the APIC protocol is supported at all, then basic APIC functionality related to interrupt delivery is supported, including the following registers: APIC ID, IRR, ISR, LDR, DFR, TMR, TPR, PPR, EOI, ICR, and self-IPI. Only IPIs of the Fixed or NMI types are considered part of basic APIC functionality.

This call has no inputs. Upon completion of the call, EAX contains the return code from the call, and ECX contains a bitmask of supported features.

- Bit 0 indicates support for timer functionality (Timer LVT, divide configuration, initial count, and current count). If timer functionality is not supported, the guest must rely on the hypervisor to emulate timer support through use of the #HV Timer GHCB protocol.
- Bit 1 indicates support for INIT and SIPI delivery. If INIT and SIPI delivery are not supported, the guest may use INIT and SIPI signals to start additional vCPUs within the invoking VMPL. Note that even if INIT and SIPI are supported, the guest must still use the VMSA creation calls of the SVSM Core Protocol to start additional vCPUs so that the Calling Area address for each vCPU can be configured correctly.
- Future bits are reserved and may be defined to include performance counters or other architectural interrupt sources.

APIC Emulation Configuration (Call 1)

This call (SVSM_APIC_CONFIGURE_EMULATION) controls whether the guest can make use of the SVSM APIC Protocol. If Alternate Injection is currently enabled on the calling vCPU, the guest can increment or decrement the registration count of the SVSM APIC Protocol (as described in the section “Supporting Multiple Guest Runtimes”). Three forms of the call are possible.

- ECX[1:0] = 0b00. This will cause Alternate Injection to be disabled on the calling vCPU if the registration count has dropped to zero. If the registration count remains non-zero, then no change will occur.
- ECX[1:0] = 0b01. This will cause Alternate Injection to be deregistered for the guest. If the registration count drops to zero, then Alternate Injection will be disabled on the calling vCPU.
- ECX[1:0] = 0b10. This will cause Alternate injection to be registered for the guest. No change will be made to Alternate injection on the calling vCPU. If the registration count is

already zero, this will fail with error code 0x8000_1000 (SVSM_ERR_APIC_CANNOT_REGISTER).

All other ECX bits are reserved and must be zero. The combination 0b11 for ECX[1:0] is illegal. Passing any illegal combination of bits will cause the call to fail with SVSM_ERR_INVALID_PARAMETER.

Once Alternate Injection is disabled on the calling vCPU, further calls to the SVSM APIC Protocol will return SVSM_ERR_UNSUPPORTED_PROTOCOL.

Read APIC Register (Call 2)

This call (SVSM_APIC_READ_REGISTER) permits the guest to read an APIC register. Upon entry, ECX contains the X2APIC MSR number corresponding to the APIC register, in the range 0x800-0x8FF. Upon successful completion of the call, RDX contains the value of the register. Reading the ICR Low MSR (0x830) will return the entire 64-bit value of the Interrupt Control Register. If ECX contains an illegal or unsupported MSR number, then SVSM_ERR_INVALID_ADDRESS will be returned. If APIC emulation is not active, then SVSM_ERR_INVALID_REQUEST will be returned.

Write APIC Register (Call 3)

This call (SVSM_APIC_WRITE_REGISTER) permits the guest to write an APIC register. Upon entry, ECX contains the X2APIC MSR number corresponding to the APIC register, in the range 0x800-0x8FF, and RDX contains the value of the register. Writing the ICR Low MSR will write the entire 64-bit value of the Interrupt Control Register. If ECX contains an illegal or unsupported MSR number, SVSM_ERR_INVALID_ADDRESS will be returned. If ECX contains an MSR number that does not support writing, then SVSM_ERR_INVALID_PARAMETER will be returned. If APC emulation is not active, then SVSM_ERR_INVALID_REQUEST will be returned.

Configure Interrupt Vector (Call 4)

This call (SVSM_APIC_CONFIGURE_VECTOR) permits the guest to designate an interrupt vector as permissible or impermissible for the hypervisor to deliver. If a vector is designated as permissible, then the SVSM will signal that vector whenever it has been scheduled for delivery by the hypervisor (regardless of whether the source of the interrupt is legitimate or spurious). If a vector is designated as impermissible, then the SVSM will prevent that vector from being delivered to the guest, and will simply acknowledge the interrupt with the hypervisor without processing it any further (if the interrupt is level-sensitive, the SVSM will perform an EOI with the hypervisor without delivering it to the guest OS). Two forms of this call are supported.

- ECX[9]=1. This form requests that all vectors be enabled, or that all vectors be disabled. If ECX[8]=1, then all vectors will be enabled, and if ECX[8]=0, then all vectors will be disabled. ECX[7:0] are ignored. All other bits in ECX are reserved and must be zero.
- ECX[9]=0. This form requests a specific vector to be enabled or disabled. ECX[7:0] contains the vector number. If ECX[8]=1, then the vector will be enabled, and if ECX[8]=0, then the vector will be disabled. All other bits in ECX are reserved and must be zero.

Enabling vector 2 will enable the host to present NMI, and disabling vector 2 will prevent the host from presenting NMI. Specifying a specific vector other than 2 or a vector in the range 0x1F-0xFF will cause SVSM_ERR_INVALID_PARAMETER to be returned. If any reserved bit in ECX is set, then SVSM_ERR_INVALID_PARAMETER will be returned. If APIC emulation is not active, then SVSM_ERR_INVALID_REQUEST will be returned.

Configure Alternate Injection (Call 5)

If ECX[1] is 1, APIC Emulation will be registered, and if ECX[0] is 1, APIC Emulation will be deregistered. Following the call, the state of Alternate Injection on the calling vCPU will be updated to reflect the current registration state of

Core Calling Area Changes

Byte offset 2 of the Calling Area is defined to permit acceleration of End-of-Interrupt signaling without having to invoke the SVSM explicitly. Byte 2 is defined as "No EOI Required", similar to the NoEoiRequired field in the #HV doorbell page.

When the SVSM delivers an interrupt, and no lower priority interrupt is pending, the SVSM will set NoEoiRequired to 1, indicating that it is possible for the guest to complete an EOI without having to write to the APIC EOI register. When the guest is ready to perform an EOI, it must perform an interlocked exchange to write the value zero into the NoEoiRequired field (this must be an interlocked operation in case the SVSM is invoked to perform interrupt processing, interrupting the guest's logic to examine NoEoiPending). If the previous value is non-zero, then the EOI is complete and no further action is required, but if the previous value is zero, then the guest perform an explicit EOI cycle by issuing the SVSM_APIC_WRITE_REGISTER call). If the SVSM delivers an interrupt, and a lower interrupt is also pending, then the SVSM will set NoEoiRequired to zero. Similarly, if the SVSM schedules an interrupt while a higher priority interrupt is already in service, the SVSM will clear the NoEoiPending field to indicate that an explicit EOI cycle will be required to cause the pending lower priority interrupt to be delivered.

Pseudocode

Hypervisor Interrupt Signaling

```
let descriptor := extended descriptor for target VMPL;
if one interrupt is pending
    descriptor[7:0] := vector;
    descriptor[14] := 0;
    perform internal EOI if vector is edge-triggered
else
    descriptor[7:0] := highest pending level-sensitive interrupt, or zero if none;
    if any other edge-triggered interrupts are pending {
        descriptor[14] := 1;
        copy edge-triggered interrupts into descriptor bitmap;
        perform internal EOI for all pending edge-triggered interrupts;
    endif
endif
doorbell.flags[10, 9, or 8] := 1 based on target VMPL;
if doorbell.flags[10, 9, or 8] was previously zero
    signal notification interrupt to SVSM;
endif
```

Consuming interrupts

```
if interlocked test and reset (doorbell.flags[8]) != 0
    process interrupts for VMPL 1
endif
if interlocked test and reset (doorbell.flags[9]) != 0
    process interrupts for VMPL 2
endif
if interlocked test and reset (doorbell.flags[10]) != 0
    process interrupts for VMPL 3
endif

process interrupts:

let flags := interlocked exchange descriptor[15:0] with 0;
if flags[14] == 0
    let vector := descriptor[7:0];
    insert (vector) into the IRR;
    if flags[10] != 0
        insert (vector) into the TMR;
    else
        remove (vector) from the TMR;
    endif
else
    if flags[10] != 0
        let vector := descriptor[7:0];
        insert (vector) into the IRR;
        insert (vector) into the TMR;
    endif

    for each additional 16-bit word in the descriptor {
        let flags := interlocked exchange descriptor word with 0;
        if (flags != 0)
            insert each corresponding vector into the IRR;
            remove each corresponding vector from the TMR;
        endif
    end for
endif
```