Speculative Execution Side Channel Mitigations

Revision 3.0

May 2018

Document Number: 336996-003

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at www.intel.com.

Intel provides these materials "as is", with no express or implied warranties.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

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.

Intel, and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Contents

§

Revision History

Document Number	Revision Number	Description	Date
336996-001	1.0	Initial release.	January 2018
336996-002	2.0	Added information on Speculative Store Bypass. Moved architectural MSRs to their own section.	May 2018
336996-002	3.0	Added information about IA32_FLUSH_CMD MSR	July 2018

§

1 Introduction

Side channel methods are techniques that may allow an attacker to gain information through observing the system, such as measuring microarchitectural properties about the system. This document considers multiple side channel methods: *branch target injection, bounds check bypass,* and *speculative store bypass*.

Section 2 describes branch target injection and presents mitigation techniques based on *indirect branch control mechanisms*, which are new interfaces between the processor and system software.

Section 3 describes bounds check bypass as well as mitigation techniques based on software modification.

Section 4 describes speculative store bypass as well as mitigation techniques through *Speculative Store Bypass Disable* or through software modification.

Section 5 describes CPUID enumeration and the architectural MSRs that are available for use in some mitigations.

2 Indirect Branch Control Mitigation

2.1 Overview of Branch Target Injection

Intel processors use *indirect branch predictors* to determine the operations that are speculatively executed after a near indirect branch instruction. *Branch target injection* is a side channel method that takes advantage of the indirect branch predictors. By controlling the operation of the indirect branch predictors ("training"), an attacker can cause certain instructions to be speculatively executed and then use the effects for side channel analysis.

2.2 Overview of Indirect Branch Control Mechanisms

Intel has developed mitigation techniques for branch target injection. One technique uses *indirect branch control mechanisms*, which are new interfaces between the processor and system software. These mechanisms allow system software to prevent an attacker from controlling a victim's indirect branch predictions (e.g., by invalidating the indirect branch predictors at appropriate times).

Three indirect branch control mechanisms are defined in this specification:

- Indirect Branch Restricted Speculation (IBRS): Restricts speculation of indirect branches.
- Single Thread Indirect Branch Predictors (STIBP): Prevents indirect branch predictions from being controlled by a sibling Hyperthread.
- Indirect Branch Predictor Barrier (IBPB): Prevents indirect branch predictions after the barrier from being controlled by software executed before the barrier.

Appropriately written software can use these indirect branch control mechanisms to defend against branch target injection attacks.

2.3 Background and Terminology

2.3.1 Indirect Branch Prediction

The processor uses indirect branch predictors to control only the operation of the branch instructions enumerated in the table below.

Table 2-1. Instructions that use Indirect Branch Predictors

Branch Type	Instruction	Opcode
Near Call Indirect	CALL r/m16, CALL r/m32, CALL r/m64	FF /2
Near Jump Indirect	JMP r/m16, JMP r/m32, JMP r/m64	FF /4
Near Return	RET, RET Imm16	C3, C2 Iw

References in this document to indirect branches are only to near call indirect, near jump indirect and near return instructions.

2.3.2 Indirect Branch Prediction and Intel® Hyper-Threading Technology (Intel® HT Technology)

In a processor supporting Intel® Hyper-Threading Technology, a core (or physical processor) may include multiple logical processors. In such a processor, the logical processors sharing a core may share indirect branch predictors. As a result of this sharing, software on one of a core's logical processors may be able to control the predicted target of an indirect branch executed on another logical processor of the same core.

This sharing occurs only within a core. Software executing on a logical processor of one core cannot control the predicted target of an indirect branch by a logical processor of a different core.

2.3.3 Return Stack Buffer (RSB)

The Return Stack Buffer (RSB) is a microarchitectural structure that holds predictions for execution of near RET instructions.

Each execution of a near CALL instruction with a non-zero displacement[1] adds an entry to the RSB that contains the address of the instruction sequentially following that CALL instruction. The RSB is not used or updated by far CALL, far RET, or IRET instructions.

2.3.4 Predictor Mode

Intel processors support different modes of operation corresponding to different degrees of privilege. VMX root operation (for a virtual-machine monitor, or **host**) is more privileged than VMX non-root operation (for a virtual machine, or **guest**). Within either VMX root operation or VMX non-root operation, **supervisor** mode (CPL < 3) is more privileged than **user** mode (CPL= 3).

To prevent attacks based on branch target injection, it can be important to ensure that less privileged software cannot control use of the branch predictors by more privileged software. For this reason, it is useful to introduce the concept of **predictor mode**. There are four predictor modes: host-supervisor, host-user, guest-supervisor, and guest-user.

The guest predictor modes are considered less privileged than the host predictor modes. Similarly, the user predictor modes are considered less privileged than the supervisor predictor modes.

There are operations that may be used to transition between unrelated software components but which do not change CPL or cause a VMX transition. These operations do not change predictor mode. Examples include MOV to CR3, VMPTRLD, EPTP switching (using VM function 0), and GETSEC[SENTER].

[1] A CALL with a target of the next sequential instruction has zero displacement.

2.4 Indirect Branch Control Mechanisms

2.4.1 Indirect Branch Restricted Speculation (IBRS)

Indirect branch restricted speculation (**IBRS**) is an indirect branch control mechanism that restricts speculation of indirect branches. A processor supports IBRS if it enumerates CPUID.(EAX=7H,ECX=0):EDX[26] as 1.

2.4.1.1 IBRS: Basic Support

Processors that support IBRS provide the following guarantees without any enabling by software:

- The predicted targets of near indirect branches executed in an enclave (a protected container defined by Intel® SGX) cannot be controlled by software executing outside the enclave.
- If the default treatment of SMIs and SMM is active, software executed before a system-management interrupt (SMI) cannot control the predicted targets of indirect branches executed in system-management mode (SMM) after the SMI.

2.4.1.2 IBRS: Support Based on Software Enabling

IBRS provides a method for critical software to protect their indirect branch predictions.

If software sets IA32_SPEC_CTRL.IBRS to 1 after a transition to a more privileged predictor mode, predicted targets of indirect branches executed in that predictor mode with IA32_SPEC_CTRL.IBRS = 1 cannot be controlled by software that was executed in a less privileged predictor mode.[2] Additionally, when IA32_SPEC_CTRL.IBRS is set to 1, the predicted targets of indirect branches cannot be controlled by another logical processor.

If IA32_SPEC_CTRL.IBRS is already 1 before a transition to a more privileged predictor mode, some processors may allow the predicted targets of indirect branches executed in that predictor mode to be controlled by software that executed before the transition. Software can avoid this by using WRMSR on the IA32_SPEC_CTRL MSR to set the IBRS bit to 1 after any such transition, regardless of the bit's previous value. It is not necessary to clear the bit first; writing it with a value of 1 after the transition suffices, regardless of the bit's original value.

Setting IA32_SPEC_CTRL.IBRS to 1 does not suffice to prevent the predicted target of a near return from using an RSB entry created in a less privileged predictor mode. Software can avoid this by using an RSB overwrite sequence[3] following a transition to a more privileged predictor mode. It is not necessary to use such a sequence following a transition from user mode to supervisor mode if supervisor-mode execution prevention (SMEP) is enabled. SMEP prevents execution of code on user mode pages, even speculatively, when in supervisor mode. User mode code can only insert its own return addresses into the RSB; not the return addresses of targets on supervisor mode code pages. On parts without SMEP where separate page tables are used for the OS and applications, the OS page tables can map user code as no-execute. The processor will not speculatively execute instructions from a translation marked no-execute.

---

[2] A transition to a more privileged predictor mode through an INIT# is an exception to this and may not be sufficient to prevent the predicted targets of indirect branches executed in the new predictor mode from being controlled by software operating in a less privileged predictor mode.
[3] An RSB overwrite sequence is a sequence of instructions that includes 32 more near CALL instructions with non-zero displacements than it has near RETs.

Enabling IBRS does not prevent software from controlling the predicted targets of indirect branches of unrelated software executed later at the same predictor mode (for example, between two different user applications, or two different virtual machines). Such isolation can be ensured through use of the IBPB command, described in Section 2.4.3, "Indirect Branch Predictor Barrier (IBPB)".

Enabling IBRS on one logical processor of a core with Intel Hyper-Threading Technology may affect branch prediction on other logical processors of the same core. For this reason, software should disable IBRS (by clearing IA32_SPEC_CTRL.IBRS) prior to entering a sleep state (e.g., by executing HLT or MWAIT) and re-enable IBRS upon wakeup and prior to executing any indirect branch.

2.4.1.3 Enhanced IBRS

Some processors may enhance IBRS by simplifying software enabling and improving performance. A processor supports **enhanced IBRS** if RDMSR returns a value of 1 for bit 1 of the IA32_ARCH_CAPABILITIES MSR.

Enhanced IBRS supports an 'always on' model in which IBRS is enabled once (by setting IA32_SPEC_CTRL.IBRS) and never disabled. If IA32_SPEC_CTRL.IBRS = 1 on a processor with enhanced IBRS, the predicted targets of indirect branches executed cannot be controlled by software that was executed in a less privileged predictor mode or on another logical processor.

As a result, software operating on a processor with enhanced IBRS need not use WRMSR to set IA32_SPEC_CTRL.IBRS after every transition to a more privileged predictor mode. Software can isolate predictor modes effectively simply by setting the bit once. Software need not disable enhanced IBRS prior to entering a sleep state such as MWAIT or HLT.

On processors with enhanced IBRS, an RSB overwrite sequence may not suffice to prevent the predicted target of a near return from using an RSB entry created in a less privileged predictor mode. Software can prevent this by enabling SMEP (for transitions from user mode to supervisor mode) and by having IA32_SPEC_CTRL.IBRS set during VM exits. Processors with enhanced IBRS still support the usage model where IBRS is set only in the OS/VMM for OSes that enable SMEP. To do this, such processors will ensure that guest behavior cannot control the RSB after a VM exit once IBRS is set, even if IBRS was not set at the time of the VM exit. If the guest has cleared IBRS, the hypervisor should set IBRS after the VM exit, just as it would do on processors supporting IBRS but not enhanced IBRS. As with IBRS, enhanced IBRS does not prevent software from affecting the predicted target of an indirect branch executed at the same predictor mode. For such cases, software should use the IBPB command, described in Section 2.4.3, "Indirect Branch Predictor Barrier (IBPB)".

2.4.2 Single Thread Indirect Branch Predictors (STIBP)

Single thread indirect branch predictors (**STIBP**) is an indirect branch control mechanism that restricts the sharing of branch prediction between logical processors on a core. A processor supports STIBP if it enumerates CPUID.(EAX=7H,ECX=0):EDX[27] as 1.

As noted in Section 2.3.2, "Indirect Branch Prediction and Intel® Hyper-Threading Technology (Intel® HT Technology)", the logical processors sharing a core may share indirect branch predictors, allowing one logical processor to control the predicted targets of indirect branches by another logical processor of the same core. Setting bit 1 (STIBP) of the IA32_SPEC_CTRL MSR on a logical processor prevents the predicted targets of indirect branches on any logical processor of that core from being controlled by software that executes (or executed previously) on another logical processor of the same core.

Recall that indirect branch predictors are never shared across cores. Thus, the predicted target of an indirect branch executed on one core can never be affected by software operating on a different core. It is not necessary to set IA32_SPEC_CTRL.STIBP to isolate indirect branch predictions from software operating on other cores.

Many processors do not allow the predicted targets of indirect branches to be controlled by software operating on another logical processor, regardless of STIBP. These include processors on which Intel Hyper-Threading Technology is not enabled and those that do not share indirect branch predictors between logical processors. To simplify software enabling and enhance workload migration, STIBP may be enumerated (and setting IA32_SPEC_CTRL.STIBP allowed) on such processors.

A processor may enumerate support for the IA32_SPEC_CTRL MSR (e.g., by enumerating CPUID.(EAX=7H,ECX=0):EDX[26] as 1) but not for STIBP (CPUID.(EAX=7H,ECX=0):EDX[27] is enumerated as 0). On such processors, execution of WRMSR to IA32_SPEC_CTRL ignores the value of bit 1 (STIBP) and does not cause a general-protection exception (#GP) if bit 1 of the source operand is set. It is expected that this fact will simplify virtualization in some cases.

As noted in Section 2.4.1, "Indirect Branch Restricted Speculation (IBRS)", enabling IBRS prevents software operating on one logical processor from controlling the predicted targets of indirect branches executed on another logical processor. For that reason, it is not necessary to enable STIBP when IBRS is enabled.

Enabling STIBP on one logical processor of a core with Intel Hyper-Threading Technology may affect branch prediction on other logical processors of the same core. For this reason, software should disable STIBP (by clearing IA32_SPEC_CTRL.STIBP) prior to entering a sleep state (e.g., by executing HLT or MWAIT) and re-enable STIBP upon wakeup and prior to executing any indirect branch.

2.4.3 Indirect Branch Predictor Barrier (IBPB)

The **indirect branch predictor barrier** (**IBPB**) is an indirect branch control mechanism that establishes a barrier, preventing software that executed before the barrier from controlling the predicted targets of indirect branches executed after the barrier on the same logical processor. A processor supports IBPB if it enumerates CPUID.(EAX=7H,ECX=0):EDX[26] as 1.

Unlike IBRS and STIBP, IBPB does not define a new mode of processor operation that controls the branch predictors. As a result, it is not enabled by setting a bit in the IA32_SPEC_CTRL MSR. Instead, IBPB is a "command" that software executes when necessary.

Software executes an IBPB command by writing the IA32_PRED_CMD MSR to set bit 0 (IBPB). This can be done either using the WRMSR instruction or as part of a VMX transition that loads the MSR from an MSR-load area. Software that executed before the IBPB command cannot control the predicted targets of indirect branches executed after the command on the same logical processor. The IA32_PRED_CMD MSR is write-only, and it is not necessary to clear the IBPB bit before writing it with a value of 1.

IBPB can be used in conjunction with IBRS to account for cases that IBRS does not cover:

- As noted in Section 2.4.1, "Indirect Branch Restricted Speculation (IBRS)", IBRS does not prevent software from controlling the predicted target of an indirect branch of unrelated software (e.g., a different user application or a different virtual machine) executed at the same predictor mode. Software can prevent such control by executing an IBPB command when changing the identity of software operating at a particular predictor mode (e.g., when changing user applications or virtual machines).
- Software may choose to clear IA32_SPEC_CTRL.IBRS in certain situations (e.g., for execution with CPL = 3 in VMX root operation). In such cases, software can use an IBPB command on certain transitions (e.g., after running an untrusted virtual machine) to prevent software that executed earlier from controlling the predicted targets of indirect branches executed subsequently with IBRS disabled.

2.5 Retpoline

An alternative approach, developed by Google, to using IBRS is a software approach called ‘retpoline’. Details of retpoline are described in Retpoline: A Branch Target Injection Mitigation[4].

[4] https://software.intel.com/sites/default/files/managed/1d/46/Retpoline-A-Branch-Target-Injection-Mitigation.pdf

3 Bounds Check Bypass Mitigation

3.1 Overview of Bounds Check Bypass

Bounds check bypass is a side channel method that takes advantage of the speculative execution that may occur following a conditional branch instruction. Specifically, the method is used in situations in which the processor is checking whether an input is in bounds (e.g., while checking whether the index of an array element being read is within acceptable values). The processor may issue operations speculatively before the bounds check resolves. If the attacker contrives for these operations to access out-of-bound memory, information may be leaked to the attacker in certain circumstances.

3.2 Bounds Check Bypass Mitigation

Bounds check bypass can be mitigated through the modification of software to constrain speculation in confused deputies. Specifically, software can insert a speculation stopping barrier between a bounds check and a later operation that could cause a speculative side channel. The LFENCE instruction, or any serializing instruction, can serve as such a barrier. These instructions suffice regardless of whether the bounds checking is implemented using conditional branches or through the use of bound-checking instructions (BNDCL and BNDCU) that are part of the Intel® Memory Protection Extensions (Intel® MPX).

The LFENCE instruction and the serializing instructions all ensure that no later instruction will execute, even speculatively, until all prior instructions have completed locally. The LFENCE instruction has lower latency than the serializing instructions and thus is recommended.

Other instructions such as CMOVcc, AND, ADC, SBB and SETcc can also be used to prevent bounds check bypass by constraining speculative execution on current family 6 processors (Intel® Core™, Intel® Atom™, Intel® Xeon® and Intel® Xeon Phi™ processors). However, these instructions may not be guaranteed to do so on future Intel processors. Intel intends to release further guidance on the usage of instructions to constrain speculation in the future before processors with different behavior are released.

Memory disambiguation (described in Section 4.1 "Overview of Speculative Store Bypass") can theoretically impact such speculation constraining sequences when they involve a load from memory.

In the following example, a CMOVG instruction is inserted to prevent a side channel from being created with data from any locations beyond the array bounds.

```
CMP RDX, [array_bounds]
JG out_of_bounds_input
MOV RCX, 0
MOV RAX, [RDX + 0x400000]
CMOVG RAX, RCX
<Further code that causes cache movement based on RAX value>
```

As an example, assume the value at 'array_bounds' is 0x20, but that value was only just stored to 'array_bounds and that the prior value at 'array_bounds' was significantly higher, such as 0xFFFF. The processor can execute the CMP instruction speculatively using a value of 0xFFFF for the loaded value due to the memory disambiguation mechanism. The instruction will eventually be re-executed with the intended array bounds of 0x20. This can theoretically cause the above sequence to support the

creation of a side channel that reveals information about the memory at addresses up to 0xFFFF instead of constraining it to addresses below 0x20.

4 Speculative Store Bypass Mitigation

4.1 Overview of Speculative Store Bypass

Many Intel processors use memory disambiguation predictors that allows loads to be executed speculatively before it is known whether the load's address overlaps with a preceding store's address. This may happen if a store's address is unknown when the load is ready to execute. If the processor predicts that the load address will not overlap with the unknown store address, the load may execute speculatively. However, if there was indeed an overlap, then the load may consume stale data. When this occurs, the processor will re-execute the load to ensure a correct result.

Through the memory disambiguation predictors, an attacker can cause certain instructions to be executed speculatively and then use the effects for side channel analysis. For example, consider the following scenario:

> Assume that a key K exists. The attacker is allowed to know the value of M, but not the value of key K. X is a variable in memory.
>
> 1. X = &K; // Attacker manages to get variable with address of K stored into pointer X
>
> <at some later point>
>
> 2. X = &M; // Does a store of address of M to pointer X
>
> 3. Y = Array[*X & 0xFFFF]; // Dereferences address of M which is in pointer X in order to
>
> // load from array at index specified by M[15:0]
>
> When the above code runs, the load from address X that occurs as part of step 3 may execute speculatively and, due to memory disambiguation, initially receive a value of address of K instead of the address of M. When this value of address of K is dereferenced, the array is speculatively accessed with an index of K[15:0] instead of M[15:0]. The CPU will later re-execute the load from address X and use M[15:0] as the index into the array. However, the cache movement caused by the earlier speculative access to the array may be analyzed by the attacker to infer information about K[15:0].

4.2 Speculative Store Bypass Mitigation Mechanisms

Intel has developed mitigation techniques for speculative store bypass. It can be mitigated by software modifications, or if that is not feasible, then the use of *Speculative Store Bypass Disable (SSBD)*, which prevents a load from executing speculatively until the addresses of all older stores are known.

4.2.1 Software-Based Mitigations

Speculative store bypass can be mitigated through numerous software-based approaches. This section describes two such software-based mitigations: process isolation and the selective use of LFENCE.

4.2.1.1 Process Isolation

One approach is to move all secrets into a separate address space from untrusted code. For example, creating separate processes for different websites so that secrets of one website are not mapped into the same address space as code from a different, possibly malicious, website. Similar techniques can be used for other runtime environments that rely on language based security to run trusted and untrusted code within the same process. This may also be useful as a defense in depth to prevent trusted code from being manipulated to create a side channel. Protection Keys can also be valuable in providing such isolation[5].

4.2.1.2 Using LFENCE to Control Speculative Load Execution

Software can insert an LFENCE between a store (for example, the store of address of M in step 2 of Section 4.1) and the subsequent load (for example, the load that dereferences X in step 3 of Section 4.1) to prevent the load from executing before the previous store's address is known. The LFENCE can also be inserted between the load and any subsequent usage of the data returned which might create a side channel (for example, the access to Array in step 3 of Section 4.1). Software should not apply this mitigation broadly, but instead only apply it where there is a realistic risk of an exploit; including that the attacker can control the old value in the memory location, there is a realistic chance of the load executing before the store address is known, and there is a disclosure gadget that reveals the contents of sensitive memory.

4.2.2 Speculative Store Bypass Disable (SSBD)

If the earlier software-based mitigations are not feasible, then employing Speculative Store Bypass Disable (SSBD) will mitigate speculative store bypass.

When SSBD is set, loads will not execute speculatively until the addresses of all older stores are known. This ensures that a load does not speculatively consume stale data values due to bypassing an older store on the same logical processor.

4.2.2.1 Basic Support

Software can disable speculative store bypass on a logical processor by setting IA32_SPEC_CTRL.SSBD to 1.

Both enclave and SMM code will behave as if SSBD is set regardless of the actual value of the MSR bit. The processor will ensure that a load within enclave or SMM code does not speculatively consume stale data values due to bypassing an older store on the same logical processor.

4.2.2.2 Software Usage Guideline

Enabling SSBD can prevent exploits based on speculative store bypass. However, this may reduce performance. Intel provides the following recommendations for the use of such a mitigation.

- Intel recommends software set SSBD for applications and/or execution runtimes relying on language-based security mechanisms. Examples include managed runtimes and just-in-time

[5] See Section 4.4, "Protection Keys", of the *Intel Analysis of Speculative Execution Side Channels White Paper*, available here: https://software.intel.com/sites/default/files/managed/b9/f9/336983-Intel-Analysis-of-Speculative-Execution-Side-Channels-White-Paper.pdf.

translators. If software is not relying on language-based security mechanisms, for example because it is using process isolation, then setting SSBD may not be needed.

- Intel is currently not aware of any practical exploit for Operating Systems, Virtual Machine Monitors, or other applications that do not rely on language-based security. Intel encourages its users to consider their particular security needs in determining whether to set SSBD outside context of language-based security mechanisms.

These recommendations may be updated in the future.

On Intel® Core™ and Intel® Xeon® processors that enable Intel® Hyper-Threading Technology and do not support enhanced IBRS, setting SSBD on a logical processor may impact the performance of a sibling logical processor on the same core. Intel recommends that the SSBD MSR bit be cleared when in an idle state on such processors.

Operating Systems should provide an API through which a process can request it be protected by SSBD mitigation.

Virtual Machine Monitors should allow a guest to determine whether to enable SSBD mitigation by providing direct guest access to IA32_SPEC_CTRL.

5 CPUID Enumeration and Architectural MSRs

Processor support for the new mitigation mechanisms is enumerated using the CPUID instruction and several architectural MSRs.

5.1 Enumeration by CPUID

The CPUID instruction enumerates support for the mitigation mechanisms using five feature flags in CPUID.(EAX=7H,ECX=0):EDX:

- CPUID.(EAX=7H,ECX=0):EDX[26] enumerates support for indirect branch restricted speculation (IBRS) and the indirect branch predictor barrier (IBPB). Processors that set this bit support the IA32_SPEC_CTRL MSR and the IA32_PRED_CMD MSR. They allow software to set IA32_SPEC_CTRL[0] (IBRS) and IA32_PRED_CMD[0] (IBPB).
- CPUID.(EAX=7H,ECX=0):EDX[27] enumerates support for single thread indirect branch predictors (STIBP). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[1] (STIBP).
- CPUID.(EAX=7H,ECX=0):EDX[28] enumerates support for L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_FLUSH).
- CPUID.(EAX=7H,ECX=0):EDX[29] enumerates support for the IA32_ARCH_CAPABILITIES MSR.
- CPUID.(EAX=7H,ECX=0):EDX[31] enumerates support for Speculative Store Bypass Disable (SSBD). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[2] (SSBD).

The mitigation mechanisms may be introduced to a processor by loading a microcode update. In such cases, software should re-evaluate the enumeration after loading that microcode update.

Table 5-1. CPUID Leaf 07H, Sub-leaf 0: Updated EDX Register Details

Initial EAX Value	Information Provided About the Processor	
Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)		
07H	EDX	**NOTES:** Leaf 07H main leaf (ECX = 0). If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Bits 25-00: Reserved Bit 26: IBRS and IBPB supported Bit 27: STIBP supported Bit 28: L1D_FLUSH supported Bit 29: IA32_ARCH_CAPABILITIES supported Bit 30: Reserved Bit 31: SSBD supported

NOTE: The table above is not intended to provide full details of this leaf; see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A (CPUID instruction), for full details on CPUID leaf 07H.

5.2 IA32_ARCH_CAPABILITIES MSR

Additional features are enumerated by the IA32_ARCH_CAPABILITIES MSR (MSR index 10AH). This is a read-only MSR that is supported if CPUID.(EAX=7H,ECX=0):EDX[29] is enumerated as 1.

Table 5-2. IA32_ARCH_CAPABILITIES MSR Details

Register Address		Register Name / Bit Fields	Bit Description	Comment
Hex	Dec			
10AH	266	IA32_ARCH_CAPABILITIES	Enumeration of Architectural Features (RO)	If CPUID.(EAX=07H, ECX=0):EDX[29]=1.
		0	RDCL_NO: The processor is not susceptible to Rogue Data Cache Load (RDCL)[6].	
		1	IBRS_ALL: The processor supports enhanced IBRS.	
		2	RSBA: The processor supports RSB Alternate. Alternative branch predictors may be used by RET instructions when the RSB is empty. SW using retpoline may be affected by this behavior.	
		3	SKIP_L1DFL_VMENTRY: A value of 1 indicates the hypervisor need not flush the L1D on VM entry[7].	
		4	SSB_NO: Processor is not susceptible to Speculative Store Bypass.[8]	
		63:5	Reserved.	

[6] See Section 2.2.3, "Rogue Data Cache Load", of the *Intel Analysis of Speculative Execution Side Channels White Paper*, available here: https://software.intel.com/sites/default/files/managed/b9/f9/336983-Intel-Analysis-of-Speculative-Execution-Side-Channels-White-Paper.pdf.

[7] See section 5.2.4, "Nested VMM environments," of the *Description and mitigation overview for L1 Terminal Fault* white paper.

[8] See Section 2.2.5, "Variant 4: Speculative Store Bypass", of the *Intel Analysis of Speculative Execution Side Channels White Paper*, available here: https://software.intel.com/sites/default/files/managed/b9/f9/336983-Intel-Analysis-of-Speculative-Execution-Side-Channels-White-Paper.pdf.

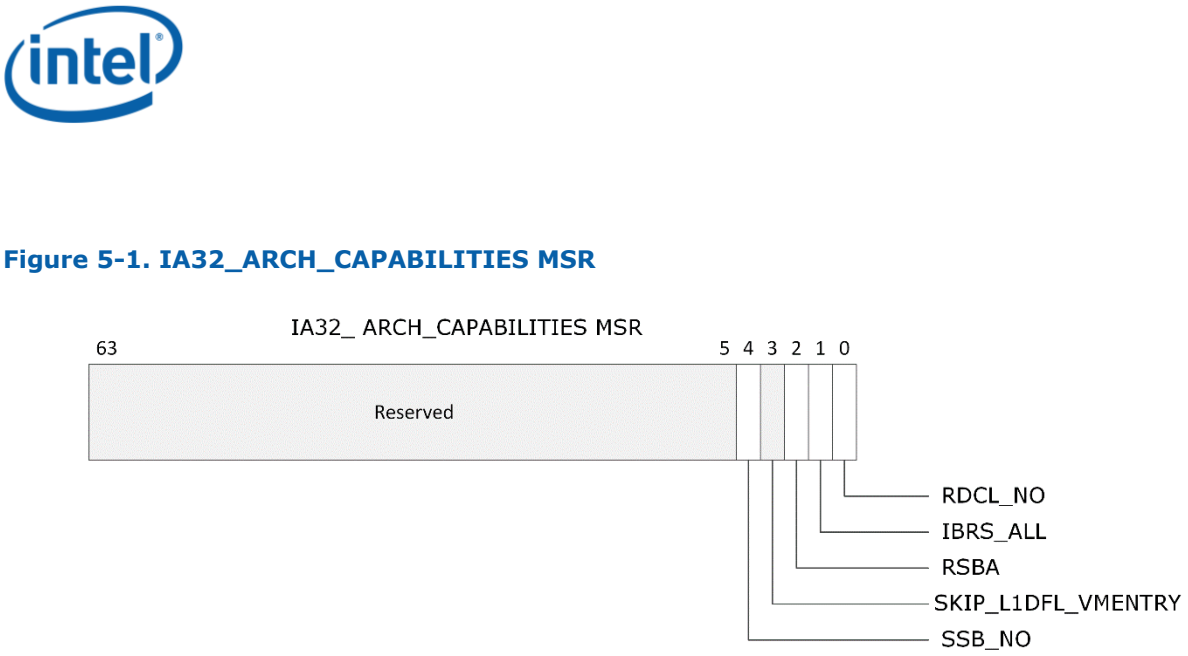


Figure 5-1. IA32_ARCH_CAPABILITIES MSR

5.3 IA32_SPEC_CTRL MSR

The IA32_SPEC_CTRL MSR bits are defined as logical processor scope. On some core implementations, the bits may impact sibling logical processors on the same core.

This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.

Like IA32_TSC_DEADLINE MSR (MSR index 6E0H), the x2APIC MSRs (MSR indices 802H to 83FH) and IA32_PRED_CMD (MSR index 49H), WRMSR to IA32_SPEC_CTRL (MSR index 48H) is not defined as a serializing instruction.

WRMSR to IA32_SPEC_CTRL does not execute until all prior instructions have completed locally and no later instructions begin execution until the WRMSR completes.

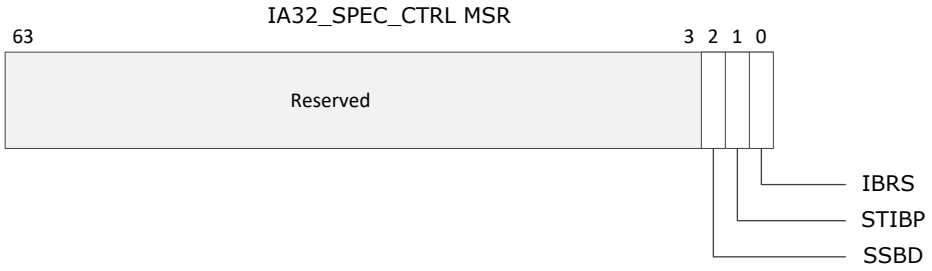
Table 5-3. IA32_SPEC_CTRL MSR Details

Register Address		Register Name / Bit Fields	Bit Description	Comment
Hex	Dec			
48H	72	IA32_SPEC_CTRL	Speculation Control (R/W)	If any one of the enumeration conditions for defined bit field positions holds.
		0	Indirect Branch Restricted Speculation (IBRS). Restricts speculation of indirect branch.	If CPUID.(EAX=07H, ECX=0):EDX[26]=1.
		1	Single Thread Indirect Branch Predictors (STIBP). Prevents indirect branch predictions on all logical processors on the core	If CPUID.(EAX=07H, ECX=0):EDX[27]=1.[9]

[9] Processors that support the IA32_SPEC_CTRL MSR but not STIBP (CPUID.(EAX=07H, ECX=0):EDX[27:26] = 01b) will not cause an exception due to an attempt to set STIBP (bit 1).

			from being controlled by any sibling logical processor in the same core.	
		2	Speculative Store Bypass Disable (SSBD) delays speculative execution of a load until the addresses for all older stores are known.	If CPUID.(EAX=07H, ECX=0):EDX[31]=1.[10]
		63:3	Reserved.	

Figure 5-2. IA32_SPEC_CTRL MSR



5.4 IA32_PRED_CMD MSR

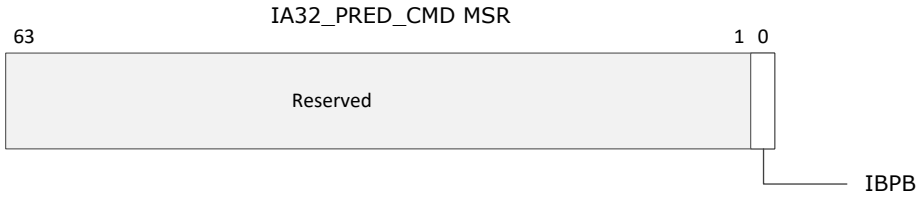
The IA32_PRED_CMD MSR gives software a way to issue commands that affect the state of predictors.

Table 5-4. IA32_PRED_CMD MSR Details

Register Address		Register Name / Bit Fields	Bit Description	Comment
Hex	Dec			
49H	73	IA32_PRED_CMD	Prediction Command (WO)	If any one of the enumeration conditions for defined bit field positions holds.
		0	Indirect Branch Prediction Barrier (IBPB).	If CPUID.(EAX=07H, ECX=0):EDX[26]=1.
		63:1	Reserved.	

[10] Future processor implementations that support the IA32_SPEC_CTRL MSR but without SSBD support will not cause an exception due to an attempt to set SSBD (bit 2). This may not be true for all current processors that support the IA32_SPEC_CTRL MSR, in particular when the latest microcode update has not been loaded.

Figure 5-3. IA32_PRED_CMD MSR



Like IA32_TSC_DEADLINE MSR (MSR index 6E0H), the X2APIC MSRs (MSR indices 802H to 83FH) and IA32_SPEC_CTRL (MSR index 48H), WRMSR to IA32_PRED_CMD (MSR index 49H) is not defined as a serializing instruction.

WRMSR to IA32_PRED_CMD does not execute until all prior instructions have completed locally and no later instructions begin execution until the WRMSR completes.

5.5 IA32_FLUSH_CMD MSR

The IA32_FLUSH_CMD MSR gives software a way to invalidate structures with finer granularity than other architectural methods.

Like the IA32_TSC_DEADLINE MSR (MSR index 6E0H), the X2APIC MSRs (MSR indices 802H to 83FH), and the IA32_SPEC_CTRL MSR (MSR index 48H), WRMSR to the IA32_FLUSH_CMD MSR (MSR index 10BH) is not defined as a serializing instruction.

WRMSR to the IA32_FLUSH_CMD MSR does not execute until all prior instructions have completed locally, and no later instructions begin execution until the WRMSR completes.

Table 5-5. IA32_FLUSH_CMD MSR details

Register address		Register name / Bit fields	Bit decription	Comment
Hex	**Dec**			
10BH	267	IA32_FLUSH_CMD	Flush Command (WO)	If any one of the enumeration conditions for defined bit field positions holds.
		0	L1D_FLUSH: Writeback and invalidate the L1 data cache.	If CPUID.(EAX=07H, ECX=0):EDX[28]=1.

Register address		Register name / Bit fields	Bit decription	Comment
Hex	**Dec**			
		63:1	Reserved	

The L1D_FLUSH command allows for finer granularity invalidation of caching structures than existing mechanisms like WBINVD. It will writeback and invalidate the L1 data cache, including all cachelines brought in by preceding instructions, without invalidating all caches (for example, L2 or LLC). Some processors may also invalidate the first level instruction cache on a L1D_FLUSH command. The L1 data and instruction caches may be shared across the logical processors of a core. This command can be used by a VMM to mitigate the L1TF exploit. Refer to the *Description and mitigation overview for L1 Terminal Fault* white paper.