Speculative Execution Side Channel Mitigations

Revision 1.0

January 2018

Document Number: 336996-001

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at www.intel.com.

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.

This document contains information on products, services and/or processes in development. 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.

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.

Intel, the Intel logo, Intel Xeon, Intel Core, Intel Atom, Intel Xeon Phi, Intel Hyper-Threading Technology (Intel HT Technology), and Intel Memory Protection Extensions (Intel MPX) 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

§

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 two side channel methods: *branch target injection* and *bounds check 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.

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].

2.4 Enumeration

Processor support for the new indirect branch control mechanisms is enumerated using the CPUID instruction and the IA32_ARCH_CAPABILITIES MSR.

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

2.4.1 Enumeration by CPUID

The CPUID instruction enumerates support for the indirect branch control mechanisms using three 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[29] enumerates support for the IA32_ARCH_CAPABILITIES MSR.

The indirect branch control 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 2-2. 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: Reserved Bit 29: IA32_ARCH_CAPABILITIES supported Bits 31-30: Reserved

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.

2.4.2 Enumeration by IA32_ARCH_CAPABILITIES

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.

Two bits are currently defined in the IA32_ARCH_CAPABILITIES MSR:

- Bit 0 is defined as RDCL_NO. If RDMSR returns 1 for this bit, the processor is not susceptible to RDCL (rogue data cache load)[2].
- Bit 1 is defined as IBRS_ALL. If RDMSR returns 1 for this bit, the processor supports enhanced IBRS (see Section 2.5.1.3, "Enhanced IBRS").

2.5 Indirect Branch Control Mechanisms

2.5.1 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.

2.5.1.1 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.

2.5.1.2 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 or on another logical processor.[3]

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

[2] See Section 2.2.3 "Rogue Data Cache Load" of the *Intel Analysis of Speculative Execution Side Channels White Paper*, available here: https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.pdf.

[3] 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.

an RSB overwrite sequence[4] 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.

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 2.5.3, "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.

2.5.1.3 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 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 enabling SMEP (for transitions from user mode to supervisor mode) and by maintaining IA32_SPEC_CTRL.IBRS = 1 (for VM exits).

As with ordinary 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 2.5.3, "Indirect Branch Predictor Barrier (IBPB)".

2.5.2 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 prevents the predicted targets of indirect branches from being controlled by software that executes (or executed previously) on another logical processor of the same core.

[4] 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.

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 2.5.1, “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.

2.5.3 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. 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. 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 2.5.1, “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.

2.6 New Architectural MSRs

2.6.1 IA32_SPEC_CTRL MSR

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

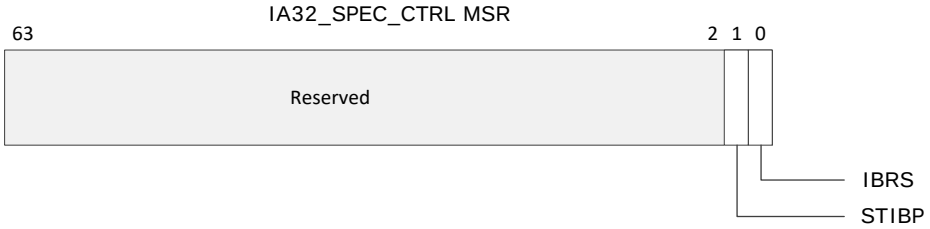
Like IA32_TSC_DEADLINE MSR (MSR index 6E0H), the x2APIC MSRs (MSR indices 802H to 83FH) and IA32_PRED_CMD (MSR index 49H), WRMSR to IA32_SPEC_CTRL (MSR index 48H) is not serializing.

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 2-3. IA32_SPEC_CTRL MSR Details

Register Address		Register Name	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 from being controlled by code executing on a sibling logical processor in the same core.	If CPUID.(EAX=07H, ECX=0):EDX[27]=1.[5]
		63:2	Reserved.	

Figure 2-1. IA32_ SPEC_CTRL MSR



[5] Processors that support IBRS but not STIBP (CPUID.(EAX=07H, ECX=0):EDX[27:26] = 01b) will ignore attempts to set STIBP instead of causing an exception due to setting that reserved bit.

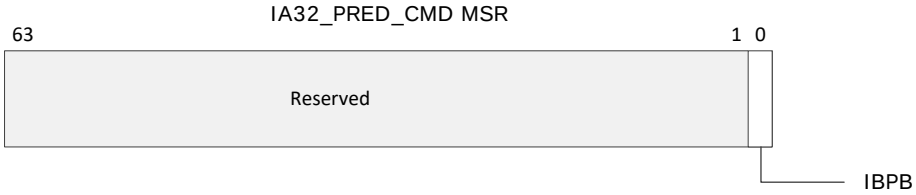
2.6.2 IA32_PRED_CMD MSR

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

Table 2-4. IA32_ PRED_CMD MSR Details

Register Address		Register Name	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.	

Figure 2-2. IA32_ ARCH_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 serializing.

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.

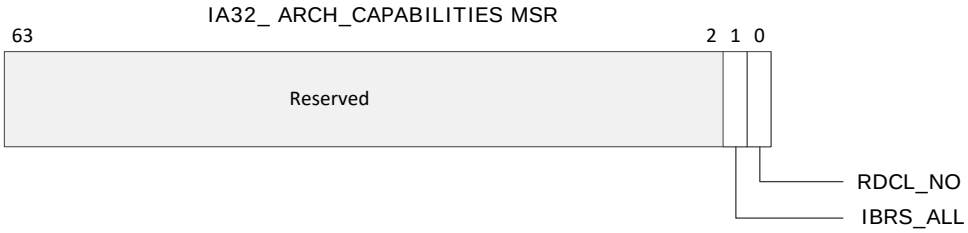
2.6.3 IA32_ARCH_CAPABILITIES MSR

A new read-only enumeration MSR called IA32_ARCH_CAPABILITIES is supported when CPUID.(EAX=07H, ECX=0):EDX[29] is set. It enumerates architectural features to software. See Section 2.4.2, "Enumeration by IA32_ARCH_CAPABILITIES", for details.

Table 2-5. IA32_ ARCH_CAPABILITIES MSR Details

Register Address		Register Name	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.	
		63:2	Reserved.	

Figure 2-3. IA32_ ARCH_CAPABILITIES MSR



[6] See Section 2.2.3, "Rogue Data Cache Load", of the *Intel Analysis of Speculative Execution Side Channels White Paper*, available here: https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.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 will release further guidance on the usage of instructions to constrain speculation in the future before processors with different behavior are released.