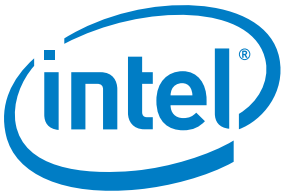


Mitigations for Jump Conditional Code Erratum

White Paper

Revision 1.0
November 2019

Document Number: 341810-001

Contents

Revision History

Date	Revision	Description
Nov 11, 2019	1.0	Initial release.

§

1.0 Introduction

Starting with second generation Intel® Core™ Processors and Intel® Xeon E3-1200 Series Processors (formerly codenamed Sandy Bridge) and later processor families, the Intel® microarchitecture introduces a microarchitectural structure called the Decoded ICache (also called the Decoded Streaming Buffer or DSB). The Decoded ICache caches decoded instructions, called micro-ops (µops), coming out of the legacy decode pipeline. The next time the processor accesses the same code, the Decoded ICache provides the µops directly, speeding up program execution.

1.1 Description of Jump Conditional Code (JCC) Erratum

In some Intel processors there is an erratum (SKX102 in https://www.intel.com/content/www/us/en/processors/xeon/scalable/xeon-scalable-spec-update.html), which may occur under complex microarchitectural conditions involving jump instructions that span 64-byte boundaries (cross cache lines).

1.2 Impact

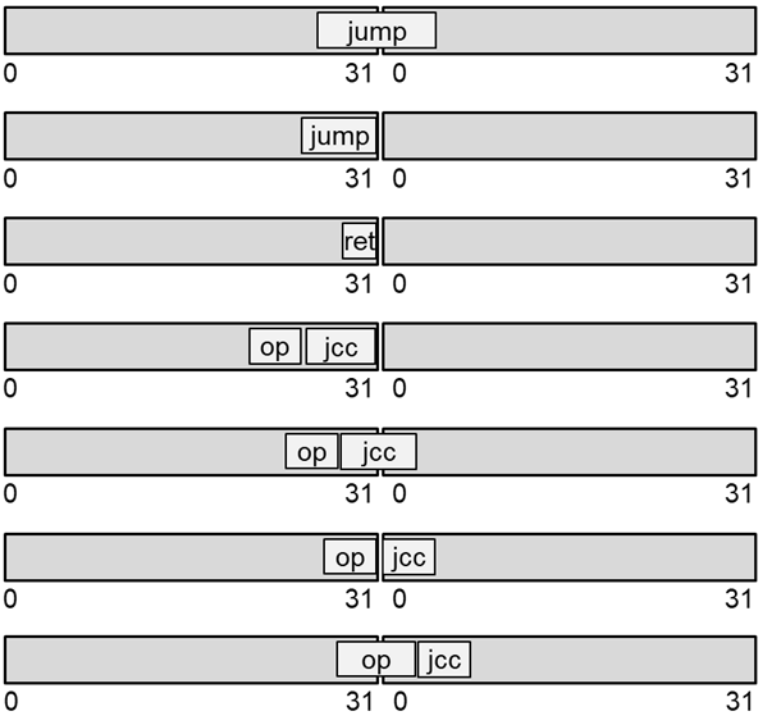
The erratum may result in unpredictable behavior when certain multiple dynamic microarchitectural conditions are met. Refer to the Affected Processors section for a full list of processors affected by this erratum. Future processors may include a fix for this erratum in the hardware.

2.0 Mitigation Strategy

2.1 Microcode Update (MCU) to Mitigate JCC Erratum

This erratum can be prevented by a microcode update (MCU). The MCU prevents jump instructions from being cached in the Decoded ICache when the jump instructions cross a 32-byte boundary or when they end on a 32-byte boundary. In this context, *Jump Instructions* include all jump types: conditional jump (`Jcc`), macro-fused `op-Jcc` (where *op* is one of `cmp`, `test`, `add`, `sub`, `and`, `inc`, or `dec`), direct unconditional jump, indirect jump, direct/indirect call, and return.

Figure 1: Jumps and 32-byte boundary



You can find the MCU that fixes this erratum on GitHub*.

2.2 Potential Performance Effects of the MCU

The JCC erratum MCU workaround will cause a greater number of misses out of the Decoded ICache and subsequent switches to the legacy decode pipeline. This occurs since branches that overlay or end on a 32-byte boundary are unable to fill into the Decoded ICache.

Intel has observed performance effects associated with the workaround ranging from 0-4% on many industry-standard benchmarks.[1] In subcomponents of these benchmarks, Intel has observed outliers higher than the 0-4% range. Other workloads not observed by Intel may behave differently. Intel has in turn developed software-based tools to minimize the impact on potentially affected applications and workloads.

The potential performance impact of the JCC erratum mitigation arises from two different sources:

1. A switch penalty that occurs when executing in the Decoded ICache and switching over to the legacy decode pipeline.
2. Inefficiencies that occur when executing from the legacy decode pipeline that are potentially hidden by the Decoded ICache.

2.3 Detecting Performance Effects of the MCU

Collect the following events to detect the performance effects of the MCU:

1. `CPU_CLK_UNHALTED.THREAD` = Core clock cycles in C0.
2. `IDQ.DSB_UOPS` = µops coming from the Decoded ICache.
3. `DSB2MITE_SWITCHES.PENALTY_CYCLES` = Penalty cycles introduced into the pipeline from switching from the Decoded ICache.
4. `FRONTEND_RETIRED.DSB_MISS_PS` = Precise frontend retired DSB miss will tag where modules, functions, and branches cause the DSB to miss.
5. `IDQ.MS_UOPS` = µops coming from the microcode sequencer.
6. `IDQ.MITE_UOPS` = µops coming from the legacy decode pipeline (also called the Micro Instruction Translation Engine)
7. `LSD.UOPS` = µops coming from the Loop Stream Detector (LSD)

   Note: The LSD is only available on some cores. The LSD.UOPS event can be excluded from calculations if not present as an event.

The ratios of interest are the following:

---

1 Data measured on Intel internal reference platform for research/educational purposes. Server benchmarks include SPECrate2017_int_base compiler with Intel Compiler Version 19 update 4, SPECrate2017_fp_base compiler with Intel Compiler Version 19 update 4, Linpack, Stream Triad, FIO (rand7030_4K_04_workers_Q32/seq7030_64K_04_workers_Q32), HammerDB-Postgres, SPECjbb2015, SPECvirt. Client benchmarks include: SPECrate2017_int_base compiler with Intel Compiler Version 19 update 4, SPECrate2017_fp_base compiler with Intel Compiler Version 19 update 4, SYSmark 2018, PCmark 10, 3Dmark Sky Diver, WebXPRT v3, Cinebench R20.

1. Determining the penalty of the switch from the Decoded ICache to the legacy decode pipeline as a percentage of core clocks:

   - `IFU_SWITCH_PENALTY% = 100*DSB2MITE_SWITCHES.PENALTY_CYCLES/CPU_CLK_UNHALTED.THREAD`

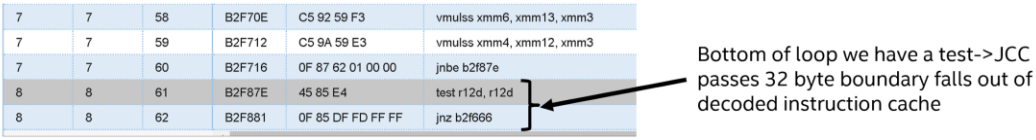
2. Determining the percentage of µops coming from the Decoded ICache:

   Note: Applications with >40% of µops coming from the Decoded ICache will be more susceptible to performance degradation. The JCC erratum mitigation can cause the percentage of µops coming from the Decoded ICache to decrease.

   - `DECODED_ICACHE_UOPS% = 100*(IDQ.DSB_UOPS/(IDQ.MS_UOPS+IDQ.MITE_UOPS+IDQ.DSB_UOPS+LSD.UOPS))`

The `FRONTEND_RETIRED.DSB_MISS_PS` event tags Decoded ICache misses to the module, function, and source lines including misses caused by the branches that overlay or end on a 32-byte boundary. This precise event is guaranteed to tag in the vicinity of decoded instruction cache misses and usually to the beginning of execution entry to an aligned 64-byte chunk.

Figure 2: Identification of the branch which overlays the 32-byte 0x80 boundary with a macro-fused test->conditional jump



2.4 Software Guidance and Optimization Methods

Software can compensate for the performance effects of the workaround for this erratum with optimizations that align the code such that jump instructions (and macro-fused jump instructions) do not cross 32-byte boundaries or end on a 32-byte boundary. Such aligning can reduce or eliminate the performance penalty caused by the transition of execution from Decoded ICache to the legacy decode pipeline.

In the following code example, the two-byte jump instruction `jae` starting at offset `1f` spans a 32-byte boundary and can cause a transition from the Decoded ICache to the legacy decode pipeline.

2.4.1 Code Without JCC Mitigation

```
0000000000000000 <fn1>:

   0: 55                    push   %rbp
```

```
 1: 41 54                    push   %r12
 3: 48 89 e5                 mov    %rsp,%rbp
 6: c5 f8 10 04 0f           vmovups (%rdi,%rcx,1),%xmm0
 b: c5 f8 11 04 0a           vmovups %xmm0,(%rdx,%rcx,1)
10: c5 f8 10 44 0f 10        vmovups 0x10(%rdi,%rcx,1),%xmm0
16: c5 f8 11 44 0a 10        vmovups %xmm0,0x10(%rdx,%rcx,1)
1c: 48 39 fe                 cmp    %rdi,%rsi
1f: 73 09                    jae    2a <fn1+0x2a>
21: e8 00 00 00 00           callq  26 <fn1+0x26>
26: 41 5c                    pop    %r12
28: c9                       leaveq
29: c3                       retq
2a: e8 00 00 00 00           callq  2f <fn1+0x2f>
2f: 41 5c                    pop    %r12
31: c9                       leaveq
32: c3                       retq
```

The advice to software developers is to align the `jae` instruction so that it does not cross a 32-byte boundary. In the example, this is done by adding the benign prefix `0x2e` four times before the first `push %rbp` instruction so that the `cmp` instruction, which started at offset `1c`, will instead start at offset `20`. Hence the macro-fused `cmp + jae` instruction will not cross a 32-byte boundary.

2.4.2 Code With JCC Mitigation

```
0000000000000000 <fn1>:
   0: 2e 2e 2e 2e 55         cs cs cs cs push %rbp
   5: 41 54                  push   %r12
   7: 48 89 e5               mov    %rsp,%rbp
   a: c5 f8 10 04 0f         vmovups (%rdi,%rcx,1),%xmm0
   f: c5 f8 11 04 0a         vmovups %xmm0,(%rdx,%rcx,1)
  14: c5 f8 10 44 0f 10      vmovups 0x10(%rdi,%rcx,1),%xmm0
```


```
1a: c5 f8 11 44 0a 10       vmovups %xmm0,0x10(%rdx,%rcx,1)
20: 48 39 fe                cmp    %rdi,%rsi
23: 73 09                   jae    2e <fn1+0x2e>
25: e8 00 00 00 00          callq  2a <fn1+0x2a>
2a: 41 5c                   pop    %r12
2c: c9                      leaveq
2d: c3                      retq
2e: e8 00 00 00 00          callq  33 <fn1+0x33>
33: 41 5c                   pop    %r12
35: c9                      leaveq
36: c3                      retq
```

3.0 Software Tools to Improve Performance

Intel is working with the community on tools that will help developers align the branches and has observed that recompilation with the updated tools can help recover most of the performance loss that might be otherwise observed in selected applications.

The release schedule of individual tools can vary, but Intel expects the updated tools to be released in the next few months.

3.1 Options for GNU Assembler

3.1.1 -mbranches-within-32B-boundaries

This is the recommended option for affected processors.[2] This option aligns conditional jumps, fused conditional jumps, and unconditional jumps within a 32-byte boundary with up to 5 segment prefixes on an instruction. It is equivalent to the following:

- `-malign-branch-boundary=32`
- `-malign-branch=jcc+fused+jmp`
- `-malign-branch-prefix-size=5`

The default doesn't align branches.

3.1.2 -malign-branch-boundary=NUM

This option controls how the assembler should align branches with segment prefixes or `NOP`. `NUM` must be a power of 2. It should be 0 or at least 32. Branches will be aligned within the `NUM` byte boundary. The default `-malign-branch-boundary=0` doesn't align branches.

3.1.3 -malign-branch=TYPE[+TYPE...]

This option specifies types of branches to align. `TYPE` is combination of the following:

- `jcc`, which aligns conditional jumps
- `fused`, which aligns fused conditional jumps
- `jmp`, which aligns unconditional jumps
- `call`, which aligns calls
- `ret`, which aligns returns

2 Note that some processors which are not affected may take longer to decode instructions with more than 3 or 4 prefixes (for example Silvermont and Goldmont processors as noted in the Intel® 64 and IA-32 Architectures Optimization Reference Manual).

- `indirect`, which aligns indirect jumps and calls

The default is `-malign-branch-boundary=jcc+fused+jmp`.

3.1.4 -malign-branch-prefix-size=NUM

This option specifies the maximum number of prefixes on an instruction to align branches. `NUM` should be between 0 and 5. The default `NUM` is 5.

4.0 Affected Processors

To find the mapping between a processor's CPUID and its Family/Model number, refer to the Intel® Software Developer's Manual, Vol 2A, table 3-8 and the *INPUT EAX = 01H: Returns Model, Family, Stepping Information* section.

Family_Model	Stepping	Processor Families/Processor Number series
06_8EH	9	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Amber Lake Y
06_8EH	C	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Amber Lake Y
06_55	7	2nd Generation Intel® Xeon® Scalable Processors based on microarchitecture code name Cascade Lake (server)
06_9EH	A	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Coffee Lake H
06_9EH	A	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Coffee Lake S
06_8EH	A	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Coffee Lake U43e
06_9EH	B	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Coffee Lake S (4+2)
06_9EH	B	Intel® Celeron® Processor G Series based on microarchitecture code name Coffee Lake S (4+2)
06_9EH	A	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Coffee Lake S (6+2) x/KBP
06_9EH	A	Intel® Xeon® Processor E Family based on microarchitecture code name Coffee Lake S (6+2)
06_9EH	A	Intel® Xeon® Processor E Family based on microarchitecture code name Coffee Lake S (6+2)
06_9EH	A	Intel® Xeon® Processor E Family based on microarchitecture code name Coffee Lake S (6+2)
06_9EH	A	Intel® Xeon® Processor E Family based on microarchitecture code name Coffee Lake S (4+2)

Family_Model	Stepping	Processor Families/Processor Number series
06_9EH	A	Intel® Xeon® Processor E Family based on microarchitecture code name Coffee Lake S (4+2)
06_9EH	A	Intel® Xeon® Processor E Family based on microarchitecture code name Coffee Lake S (4+2)
06_9EH	D	9th Generation Intel® Core™ Processor Family based on microarchitecture code name Coffee Lake H (8+2)
06_9EH	D	9th Generation Intel® Core™ Processor Family based on microarchitecture code name Coffee Lake S (8+2)
06_8EH	C	10th Generation Intel® Core™ Processor Family based on microarchitecture code name Comet Lake U42
06_A6H	0	10th Generation Intel® Core™ Processor Family based on microarchitecture code name Comet Lake U62
06_9EH	9	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Kaby Lake G
06_9EH	9	7th Generation Intel® Core™ Processor Family based on microarchitecture code name Kaby Lake H
06_AEH	A	8th Generation Intel® Core™ Processor Family based on microarchitecture code name Kaby Lake Refresh U (4+2)
06_9EH	9	7th Generation Intel® Core™ Processor Family based on microarchitecture code name Kaby Lake S
06_8EH	9	7th Generation Intel® Core™ Processor Family based on microarchitecture code name Kaby Lake U
06_8EH	9	7th Generation Intel® Core™ Processor Family based on microarchitecture code name Kaby Lake U23e
06_9EH	9	Intel® Core™ X-series Processors based on microarchitecture code name Kaby Lake X
06_9EH	9	Intel® Xeon® Processor E3 v6 Family Kaby Lake Xeon E3
06_8EH	9	7th Generation Intel® Core™ Processor Family based on microarchitecture code name Kaby Lake Y
06_55H	4	Intel® Xeon® Processor D Family based on microarchitecture code name Skylake D, Bakerville

Family_Model	Stepping	Processor Families/Processor Number series
06_5E	3	6th Generation Intel® Core™ Processor Family based on microarchitecture code name Skylake H
06_5E	3	6th Generation Intel® Core™ Processor Family based on microarchitecture code name Skylake S
06_55H	4	Intel® Xeon® Scalable Processors based on microarchitecture code name Skylake Server
06_4E	3	6th Generation Intel® Core™ Processors based on microarchitecture code name Skylake U
06_4E	3	6th Generation Intel® Core™ Processor Family based on microarchitecture code name Skylake U23e
06_55H	4	Intel® Xeon® Processor W Family based on microarchitecture code name Skylake W
06_55H	4	Intel® Core™ X-series Processors based on microarchitecture code name Skylake X
06_55H	4	Intel® Xeon® Processor E3 v5 Family based on microarchitecture code name Skylake Xeon E3
06_4E	3	6th Generation Intel® Core™ Processors based on microarchitecture code name Skylake Y
06_8EH	B	8th Generation Intel® Core™ Processors based on microarchitecture code name Whiskey Lake U
06_8EH	C	8th Generation Intel® Core™ Processors based on microarchitecture code name Whiskey Lake U