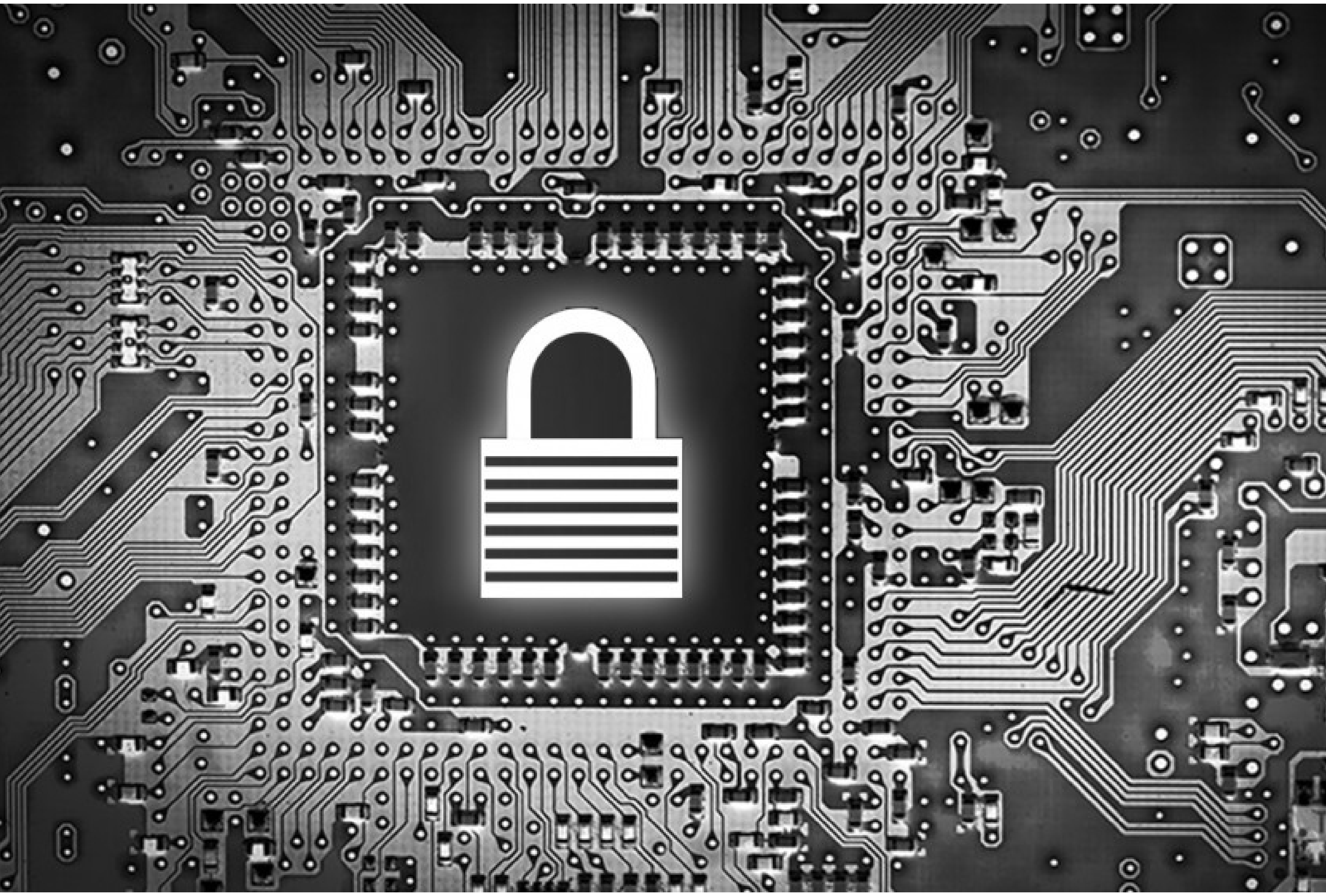


White Paper |

SPECULATION BEHAVIOR IN AMD MICRO-ARCHITECTURES

5.14.19

INTRODUCTION

This document provides in depth descriptions of AMD CPU micro-architecture and how it handles speculative execution in a variety of architectural scenarios. This document is referring to the latest Family 17h processors which include AMD's Ryzen™ and EPYC™ processors, unless otherwise specified. This document does necessarily describe general micro-architectural principles that exist in all AMD microprocessors.

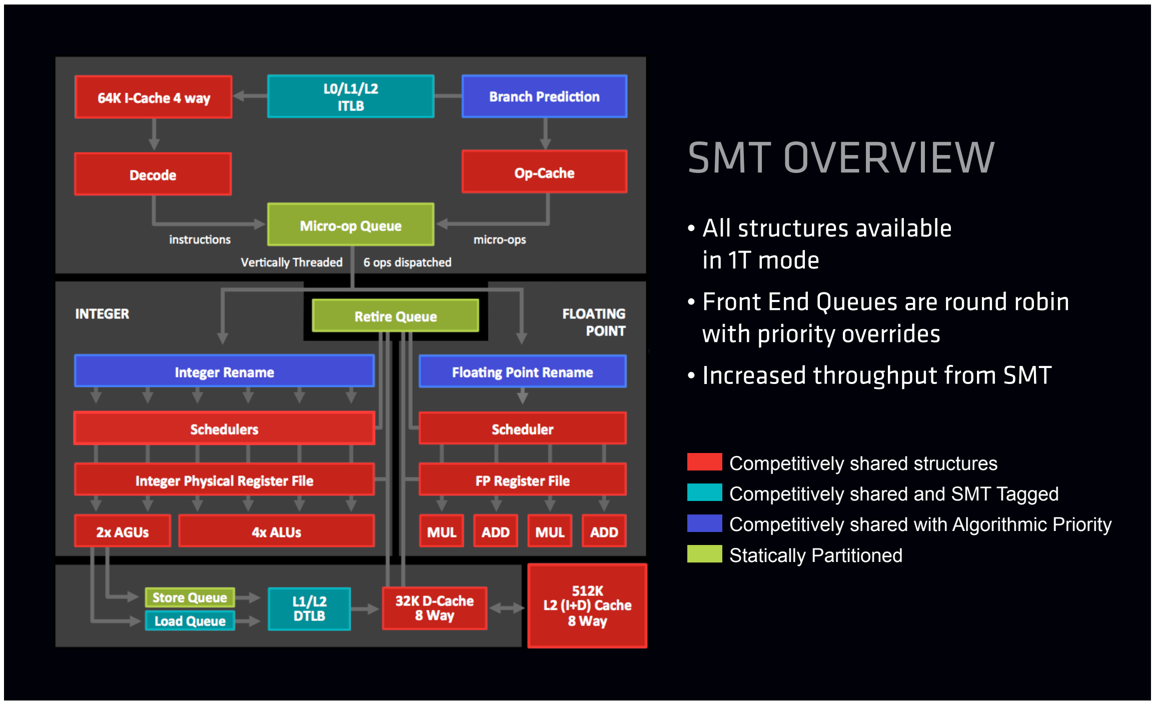
AMD's processor architecture includes hardware protection checks that AMD believes help AMD processors not be affected by many side-channel vulnerabilities. These checks happen in various speculation scenarios including during TLB validation, architectural exception handling, loads and floating point operations.

TLB ARCHITECTURE

The x86 architecture uses virtual addressing and hierarchical page tables to map the virtual address to the physical memory address used to reference caches and memory. This mapping allows privileged system software, whether the operating system or a hypervisor, to isolate different software environments by only allowing certain areas of the memory system to be accessed by each respective environment. This isolation is achieved by creating unique page tables for each environment. These page tables are isolated by either marking the page tables as not-present in the page table entry or using the protection attribute fields in the page table entry to restrict access.

For performance reasons, processors store a copy of these virtual to physical translations in a Translation Lookaside Buffer (TLB). AMD processors store translations in the TLB with a valid bit and all the protection bits from the page table which include user/supervisor, read/write bits along with other information. On each instruction that uses virtual addresses to access memory, AMD processors access the TLB and use the valid bit and the protection attributes to decide whether to access the caches. If the protection check fails, AMD processors operate as if the memory address is invalid and no data is accessed from either the cache or memory. This occurs whether the access is speculative or non-speculative. When the instruction becomes the oldest in the machine, a page fault exception will occur. A validated address is required for AMD processors to access data from both the caches and memory. **The result is AMD processors are designed to not speculate into memory that is not valid in the current virtual address memory range defined by the software defined page tables.**

For AMD processors that support multiple hardware threads running on the same core (such as AMD processors using simultaneous multithreading (SMT) or AMD Family 17h processors), the TLB entries are additionally tagged with the thread ID of the thread that created that TLB entry. The TLB adds this thread bit into the validation of the address such that TLB entries can only be used by the thread that created the TLB entry. A thread ID mismatch on an access to the TLB is treated as a miss in the TLB such that no physical address is delivered to the caches. The caches inside the processor all use physical addresses and therefore utilize the TLB thread tagging to prevent access of data by an incorrect thread. Accesses from the load queue that may hit on the store queue also require a TLB hit to validate both the load and the store address before the load can forward data to speculative instructions. The load and store queues are also tagged per thread, which is designed to prevent any store-to-load forwarding of data from the wrong thread.



SMT OVERVIEW

- All structures available in 1T mode
- Front End Queues are round robin with priority overrides
- Increased throughput from SMT

🟥 Competitively shared structures
🟦 Competitively shared and SMT Tagged
🟪 Competitively shared with Algorithmic Priority
🟩 Statically Partitioned

The above diagram shows how all major blocks are shared within the SMT architecture.

Most structures are competitively shared to improve performance which include the schedulers, register files and caches. Some structures are competitively shared but tagged per thread including the Load Queue, TLBs, and the Branch Predictor when Single Thread Indirect Branch Predictors (STIBP) is enabled. These tags help data created by one thread from being consumed by another thread, even speculatively. Some structures are competitively shared, but algorithms are used to maintain a good balance between the threads. Finally, certain structures are statically partitioned whenever two threads are active, isolating each thread. This includes the micro-op queue, the retire queue, and the store queue.

For AMD processors that support two cores in a compute unit (Family 15h), the shared structures are the same as Family 17h except this family also statically partitioned the load queues, L1 data TLB, L1 data cache, integer physical register files, integer schedulers, integer Address Generation Units (AGU), and integer Arithmetic Logic Units (ALU).

Some AMD processors do not share resources inside the core but may share a last level cache (LLC).

ARCHITECTURAL EXCEPTION HANDLING

When exceptions happen within the processor this provides a window for speculation. The most common exception in the processor is a page fault due to a memory reference that is either to an unmapped page or a page that is being protected from access. AMD processors do not speculate on data from accesses that will result in page faults. **Therefore, AMD processors are designed not to forward data to other speculative operations when the data is not allowed to be accessed by the current processor context.**

In addition to page faults, there are many other types of exceptions in the processor. The processor supports another mode of memory protection through the segmentation architecture. Segmentation uses segment registers with access rights and address limits that can lead to Segment not present faults (#NP), Stack faults (#SS), General Protection faults (#GP) faults, and Invalid-TSS faults (#TS). All these segmentations faults may provide data to younger instructions which allows speculative execution and so segmentation is not recommended to be used to prevent speculative execution. Segmentation with limits and access rights is only available in 32-bit mode and does not exist architecturally in 64 bit mode in the x86 architecture. Modern operating systems do not utilize the segmentation mechanism to restrict data access. Similarly, the BOUND instruction and the BOUND exception (#BR) it generates are only available in 32 bit mode in the x86 architecture. The BOUND instruction in the case that it generates an exception does not prevent younger instructions from speculatively using the data pointer that was checked.

Some faults are detected as the processor is decoding the instruction. These include instruction breakpoints (#DB), invalid opcode (#UD), instruction page fault (#PF) and device not available (#NM). These fault types do not allow dispatch of the current instruction on which the fault is detected or any younger instruction. Some instructions are only allowed in certain modes such as privileged instructions that can only be executed when the CPL is zero. When the instruction is not allowed in the current mode a general protection violation occurs (#GP) and the processor restricts speculative use of the data associated with the faulting instruction.

Some faults are unique to specific instructions. The divide by zero (#DE) fault is signaled on the integer divide instructions. No data is forwarded to younger, dependent operations for speculative execution on this fault. The overflow fault (#OF) is related to the INTO instruction. It does not prevent younger instructions from performing speculative operation on the OF flag. The int3 breakpoint (#BP) is generated from the int3 instruction. This instruction does not allow speculative dispatch or execution of the instructions immediately after it in its sequential path.

The alignment check fault (#AC) is detected if alignment checking is enabled and the address is misaligned. This fault does not prevent the data referenced from being delivered to younger instructions for speculative execution.

The Single Instruction, Multiple Data (SIMD) floating point exception (#XF) can be signaled for a multitude of reasons but none of them prevent the data referenced from being delivered to younger instructions for speculative execution.

The x87 floating point fault (#MF) mechanism is handled as a trap. The instruction that signals the fault (#MF) still retires and subsequent integer instructions continue to execute and retire. When the processor reaches the next floating point instruction that is error sensitive, it prevents speculative dispatch and execution of that floating point instruction and the instructions after it and enters the (#MF) fault handler.

There are several flavors of debug traps (#DB) in the x86 architecture. These include address match breakpoints on data addresses, single stepping with the trap flag, and single stepping taken branches with the trap flag. These traps do not prevent speculative execution from occurring and the data address breakpoint does not prevent the data referenced from being delivered to younger instructions for speculative execution.

Here is a summarizing table:

VECTOR	FAULT/TRAP TYPE	DESIGNED TO STOP SPECULATIVE EXECUTION	SPECULATIVELY FORWARDS DATA TO YOUNGER DEPENDENT INSTRUCTIONS
0	Divide by Zero(#DE)	NO	NO
1	Debug Trap(#DB)	NO	YES
1	Debug Instruction(#DB)	YES	N/A
3	INT3 Breakpoint(#BP)	YES	N/A
4	Overflow (#OF)	NO	YES
5	Bound(#BR)	NO	YES
6	Invalid Opcode(#UD)	YES	N/A
7	Device Not Available(#NM)	YES	N/A
8	Double Fault(#DF)	NO	YES
10	Invalid TSS(#TS)	NO	YES
11	Segment not Present(#NP)	NO	YES
12	Stack Fault(#SS)	NO	YES
13	General Protection Data Access(#GP)	NO	YES
13	General Protection Instruction Mode(#GP)	NO	NO
14	Page Fault(#PF)	NO	NO
16	X87 Floating-Point Pending(#MF)	YES	N/A
17	Alignment Check(#AC)	NO	YES

LOAD SPECULATION

As defined in the *Architectural Programmer's Manual*, Volume 2[1], Section 7.4, all loads to cacheable memory that have a valid translation from the TLB may speculatively bring lines into all caches in the memory hierarchy. AMD processors employ multiple hardware memory "prefetchers" that try to predict what memory will be needed by future instructions and brings those cache lines in to the processor's cache hierarchy. The "prefetchers" will only make accesses to memory for physical addresses that have already been validated by the TLB for this software thread and that have a cacheable memory type.

Another mechanism some AMD processors use to improve performance of a load operation allows loads to bypass older stores that do not yet have an address calculated. The load is allowed to access the cache and provide that data to younger instructions for speculative execution. In this case, the valid load address passes through the TLB and is determined to be valid for this context and to cacheable memory. When the store address is generated it is checked against the load address of the instructions that already sent data speculatively. If they match, the processor schedules the load again and provides the correct data to all the dependent instructions. This behavior can be disabled architecturally on processors that support the speculative store bypass disable feature. For more information on this feature, refer to the AMD document, AMD64 Speculative Store Bypass Disable.[2]

FLOATING POINT SPECULATION

To improve performance of some floating point routines, AMD processors may predict the value of the x87 floating point control word (FCW) fields precision control (PC) and rounding control (RC) when a FLDCW instruction is executed. A similar prediction is made for the SSE and AVX register MXCSR with the rounding control (RC), flush to zero (FTX) and denormals as zero (DAZ) mode bits on a LDMXCSR instruction. In both cases, younger instructions may speculatively calculate results with the wrong rounding or precision control. When the real value of the FCW or MXCSR is known and does not match the predicted value, the processor will cause an internal exception to flush the younger instructions and re-issue them with the correct mode. Software that is sensitive to this type of speculation can place an LFENCE after the FLDCW or LDMXCSR instruction to restrict speculation.

CONCLUSION

In conclusion, AMD microprocessors have many micro-architectural mechanisms that allow for speculative execution. For software that cannot use the natural isolation that the TLB provides to prevent speculative execution into memory, AMD provides other software techniques to prevent speculative execution. These are described in the Software techniques for Managing Speculation on AMD processors.[3]

AMD believes that our hardware paging architecture and protection checks help AMD processors not be affected by many side-channel vulnerabilities, regardless of whether Simultaneous Multi-Threading (SMT) is enabled or disabled.

ABOUT AMD

For 50 years AMD (NASDAQ: AMD) has driven innovation in high-performance computing, graphics and visualization technologies—the building blocks for gaming, immersive platforms and the datacenter. Hundreds of millions of consumers, leading Fortune 500 businesses and cutting-edge scientific research facilities around the world rely on AMD technology daily to improve how they live, work and play. AMD employees around the world are focused on building great products that push the boundaries of what is possible.

AMD.com

1. https://www.amd.com/system/files/TechDocs/24593.pdf
2. https://developer.amd.com/wp-content/resources/124441_AMD64_SpeculativeStoreBypassDisable_Whitepaper_final.pdf
3. https://developer.amd.com/wp-content/resources/90343-B_SoftwareTechniquesforManagingSpeculation_WP_7-18Update_FNL.pdf