Hardware Bugs

Mengjia Yan

mengjia@csail.mit.edu

Recall Debugging Hardware

- In 6.004, you are asked to design a single-cycle CPU
- How to test function correctness?

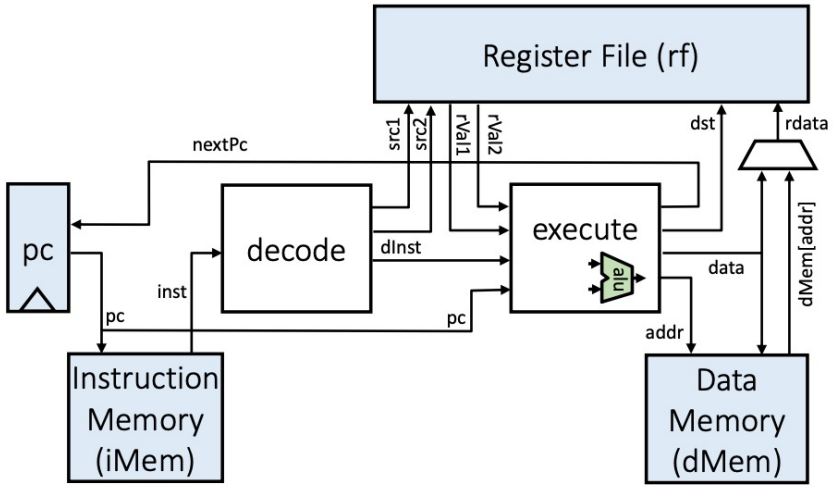


Figure 1: Overall structure of the single-cycle RISC-V processor.

MIT 6.004 ISA Reference Card: Instructions

Instruction	Syntax	Description	Execution
LUI	`lui rd, luiConstant`	Load Upper Immediate	`reg[rd] <= luiConstant « 12`
JAL	`jal rd, label`	Jump and Link	`reg[rd] <= pc + 4` `pc <= label`
JALR	`jalr rd, offset(rs1)`	Jump and Link Register	`reg[rd] <= pc + 4` `pc <= {(reg[rs1] + offset)[31:1], 1'b0}`
BEQ	`beq rs1, rs2, label`	Branch if =	`pc <= (reg[rs1] == reg[rs2]) ? label: pc + 4`
BNE	`bne rs1, rs2, label`	Branch if ≠	`pc <= (reg[rs1] != reg[rs2]) ? label: pc + 4`
BLT	`blt rs1, rs2, label`	Branch if < (Signed)	`pc <= (reg[rs1] <`$_s$` reg[rs2]) ? label: pc + 4`
BGE	`bge rs1, rs2, label`	Branch if ≥ (Signed)	`pc <= (reg[rs1] >=`$_s$` reg[rs2]) ? label: pc + 4`
BLTU	`bltu rs1, rs2, label`	Branch if < (Unsigned)	`pc <= (reg[rs1] <`$_u$` reg[rs2]) ? label: pc + 4`
BGEU	`bgeu rs1, rs2, label`	Branch if ≥ (Unsigned)	`pc <= (reg[rs1] >=`$_u$` reg[rs2]) ? label: pc + 4`
LW	`lw rd, offset(rs1)`	Load Word	`reg[rd] <= mem[reg[rs1] + offset]`
SW	`sw rs2, offset(rs1)`	Store Word	`mem[reg[rs1] + offset] <= reg[rs2]`
ADDI	`addi rd, rs1, constant`	Add Immediate	`reg[rd] <= reg[rs1] + constant`
SLTI	`slti rd, rs1, constant`	Compare < Immediate (Signed)	`reg[rd] <= (reg[rs1] <`$_s$` constant) ? 1 : 0`
SLTIU	`sltiu rd, rs1, constant`	Compare < Immediate (Unsigned)	`reg[rd] <= (reg[rs1] <`$_u$` constant) ? 1 : 0`
XORI	`xori rd, rs1, constant`	Xor Immediate	`reg[rd] <= reg[rs1] ^ constant`
ORI	`ori rd, rs1, constant`	Or Immediate	`reg[rd] <= reg[rs1] | constant`
ANDI	`andi rd, rs1, constant`	And Immediate	`reg[rd] <= reg[rs1] & constant`
SLLI	`slli rd, rs1, shamt`	Shift Left Logical Immediate	`reg[rd] <= reg[rs1] « shamt`
SRLI	`srli rd, rs1, shamt`	Shift Right Logical Immediate	`reg[rd] <= reg[rs1] »`$_u$` shamt`
SRAI	`srai rd, rs1, shamt`	Shift Right Arithmetic Immediate	`reg[rd] <= reg[rs1] »`$_s$` shamt`
ADD	`add rd, rs1, rs2`	Add	`reg[rd] <= reg[rs1] + reg[rs2]`
SUB	`sub rd, rs1, rs2`	Subtract	`reg[rd] <= reg[rs1] - reg[rs2]`
SLL	`sll rd, rs1, rs2`	Shift Left Logical	`reg[rd] <= reg[rs1] « reg[rs2][4:0]`
SLT	`slt rd, rs1, rs2`	Compare < (Signed)	`reg[rd] <= (reg[rs1] <`$_s$` reg[rs2]) ? 1 : 0`
SLTU	`sltu rd, rs1, rs2`	Compare < (Unsigned)	`reg[rd] <= (reg[rs1] <`$_u$` reg[rs2]) ? 1 : 0`
XOR	`xor rd, rs1, rs2`	Xor	`reg[rd] <= reg[rs1] ^ reg[rs2]`
SRL	`srl rd, rs1, rs2`	Shift Right Logical	`reg[rd] <= reg[rs1] »`$_u$` reg[rs2][4:0]`
SRA	`sra rd, rs1, rs2`	Shift Right Arithmetic	`reg[rd] <= reg[rs1] »`$_s$` reg[rs2][4:0]`
OR	`or rd, rs1, rs2`	Or	`reg[rd] <= reg[rs1] | reg[rs2]`
AND	`and rd, rs1, rs2`	And	`reg[rd] <= reg[rs1] & reg[rs2]`

Note: *luiConstant* is a 20-bit value. *offset* and *constant* are signed 12-bit values that are sign-extended to 32-bit values. *label* is a 32-bit memory address or its alias name. *shamt* is a 5-bit unsigned shift amount.

Real-world Processor Design

- RISC-V ISA: 145 pages
  - https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf
- RISC -> CISC
- Single-cycle -> Pipelined -> Out-of-order -> Speculation
- Core -> Memory Hierarchy -> Multi-core
- OS -> Interrupts -> Virtualization

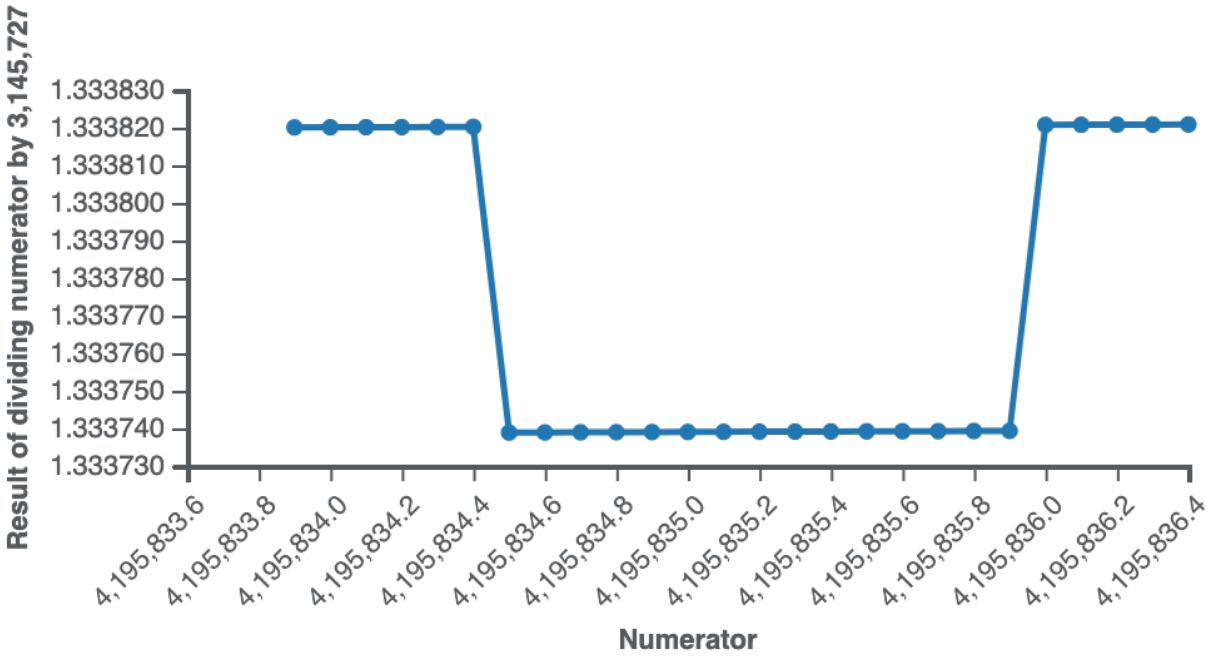
Pentium FDIV bug

- What is it?
  - A hardware bug affects FPU
  - The processor would return incorrect FP results when dividing certain pairs of high-precision numbers.

- Reported in 1994

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.3338204491362410 02$$

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.3337390689020375 89$$



Each data point should be ~3.1789 x 10^{-8} higher on the y-axis than its predecessor to the left, but in the region 4195834.4 < x < 4195835.9 the result differs from the expected value by ~8.14 x 10^{-5}.

From Wikipedia

Consequences/Impacts

- Discovered by Thomas Nicely who was working on computational number theory in June, 1994

- The bug went public in October, 1994

- Intel's bad responses
  - Conditional replacement: disastrous press
  - No-questions-asked replacement: $475M cost in 1994, 10% replacements

Some humor for you:

Q: How many Pentium designers does it take to screw in a light bulb?
A: 1.99904274017, but that's close enough for non-technical people.

Q: What do you get when you cross a Pentium PC with a research grant?
A: A mad scientist.

Do you think it bothers x86 users that the 486 is a functional upgrade to the Pentium?

In response to the Pentium bug, PowerMac officials have announced that they will be adding the control panel "Pentium Switcher" that allows users to decide whether the PowerMac should emulate pre-Pentium or post-Pentium FDIV behaviour.

TOP TEN NEW INTEL SLOGANS FOR THE PENTIUM

9.9999973251 It's a FLAW, Dammit, not a Bug
8.9999163362 It's Close Enough, We Say So
7.9999414610 Nearly 300 Correct Opcodes
6.9999831538 You Don't Need to Know What's Inside
5.9999835137 Redefining the PC--and Mathematics As Well
4.9999999021 We Fixed It, Really
3.9998245917 Division Considered Harmful
2.9991523619 Why Do You Think They Call It *Floating* Point?
1.9999103517 We're Looking for a Few Good Flaws
0.9999999998 The Errata Inside

http://davefaq.com/Opinions/Stupid/Pentium.html#glitch

Bug Explanation

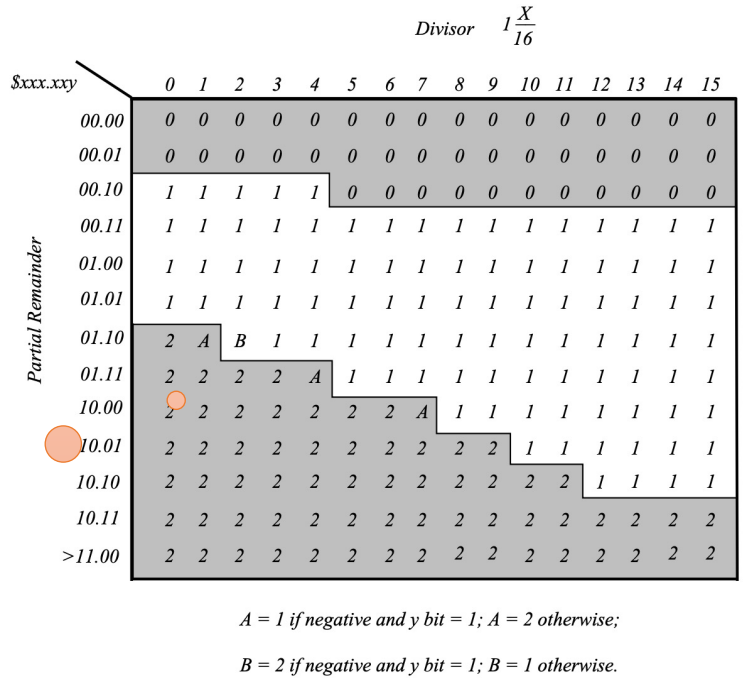
- Shift-and-subtract

Compute quotient of N/D

```
if D = 0 then error(DivisionByZeroException) end
Q := 0 -- Initialize quotient and remainder to zero
R := 0
for i := n − 1 .. 0 do -- Where n is number of bits in N
    R := R << 1 -- Left-shift R by 1 bit
    R(0) := N(i) -- Set the least-significant bit of R equal
                    to bit i of the numerator
    if R ≥ D then
        R := R − D
        Q(i) := 1
    end
end
```

The bug: Five of the 1066 entries had been mistakenly omitted

- Sweeney, Robertson, and Tocher (SRT)



Divisor $1\frac{X}{16}$

Partial Remainder \ $xxx.xy	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00.10	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
00.11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01.00	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01.01	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01.10	2	A	B	1	1	1	1	1	1	1	1	1	1	1	1	1
01.11	2	2	2	2	A	1	1	1	1	1	1	1	1	1	1	1
10.00	2	2	2	2	2	2	2	A	1	1	1	1	1	1	1	1
10.01	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1
10.10	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
10.11	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
>11.00	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

A = 1 if negative and y bit = 1; A = 2 otherwise;

B = 2 if negative and y bit = 1; B = 1 otherwise.

Figure 4: Radix-4 next quotient/root selection table

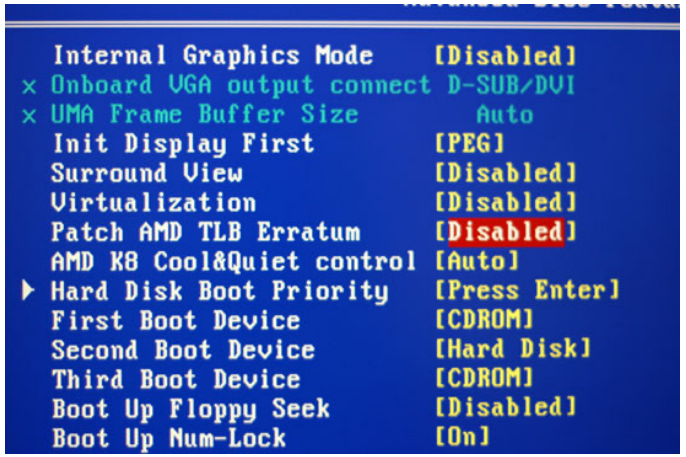
Consequences/Impacts

- Software patches. How?

- Significant impact on verification
  - A marked increase in the use of formal verification and number theory in hardware design

The AMD Phenom TLB Bug

- Affected AMD quad-core Opteron and Phenom processors in 2007
- Can cause random crashes
- AMD gave two confirmed situations in real world usage:
  - 1) Windows Vista 64-bit running SPEC CPU 2006
  - 2) Xen Hypervisor running Windows XP and an unknown configuration of applications
- Described in Erratum 298 as a cache coherence problem
- Bug Explanation
  - TLB
  - Cache coherence
  - Exclusive caches



https://www.anandtech.com/show/2477/2

Errata 298

298 L2 Eviction May Occur During Processor Operation To Set Accessed or Dirty Bit

Description

The processor operation to change the accessed or dirty bits of a page translation table entry in the L2 from 0b to 1b may not be atomic. A small window of time exists where other cached operations may cause the stale page translation table entry to be installed in the L3 before the modified copy is returned to the L2.

In addition, if a probe for this cache line occurs during this window of time, the processor may not set the accessed or dirty bit and may corrupt data for an unrelated cached operation.

Potential Effect on System

One or more of the following events may occur:

- Machine check for an L3 protocol error. The MC4 status register (MSR0000_0410) is B2000000_000B0C0Fh or BA000000_000B0C0Fh. The MC4 address register (MSR0000_0412) is 26h.
- Loss of coherency on a cache line containing a page translation table entry.
- Data corruption.

Suggested Workaround

BIOS should set MSRC001_0015[3] (HWCR[TlbCacheDis]) to 1b and MSRC001_1023[1] to 1b.

In a multiprocessor platform, the workaround above should be applied to all processors regardless of revision when an affected processor is present.

Fix Planned

Yes

https://www.amd.com/system/files/TechDocs/41322_10h_Rev_Gd.pdf

AMD Nested Interrupts DOS

- Allowed a guest VM to fault in a way that would cause the CPU to hang in a microcode infinite loop, allowing any VM to DoS its host.

AMD

Revision Guide for AMD Family 15h Models 00h-0Fh Processors 48063 Rev. 3.24 September 2014

704 Processor May Report Incorrect Instruction Pointer

Description

Under a highly specific and detailed set of internal timing conditions, the processor may store an incorrect instruction pointer (rIP) while processing an interrupt or a debug trap exception (#DB).

Potential Effect on System

Unpredictable system behavior.

Suggested Workaround

Contact your AMD representative for information on a BIOS update.

Fix Planned

No fix planned

Intel Broken Hyper-Threading

- 2017: Intel Skylake and Kabylake, broken hyper-threading
- Errata:
  - Short Loops Which Use AH/BH/CH/DH Registers May Cause Unpredictable System Behavior.
- Problem:
  - Under complex micro-architectural conditions, short loops of less than 64 instructions that use AH, BH, CH or DH registers as well as their corresponding wider register (e.g. RAX, EAX or AX for AH) may cause unpredictable system behavior. This can only happen when both logical processors on the same physical processor are active.
- Implication:
  - Due to this erratum, the system may experience unpredictable system behavior.

Errata

8th and 9th Generation Intel® Core™ P

3.2 Errata Summary Information

Specification U

Supporting 8th
S/H/U Platform

Supporting 9th Ge
Processors for S/
Refresh

November 201

Revision 002

Table 4-3. Errata

ID	B0 42
001	No Fix
002	No Fix
003	No Fix
004	No Fix

ASPLOS'15

SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs

Matthew Hicks
University of Michigan
mdhicks@umich.edu

Cynthia Sturton
University of North Carolina at Chapel Hill
csturton@cs.unc.edu

Samuel T. King
Twitter, Inc.
sking@twitter.com

Jonathan M. Smith
University of Pennsylvania
jms@cis.upenn.edu

Abstract

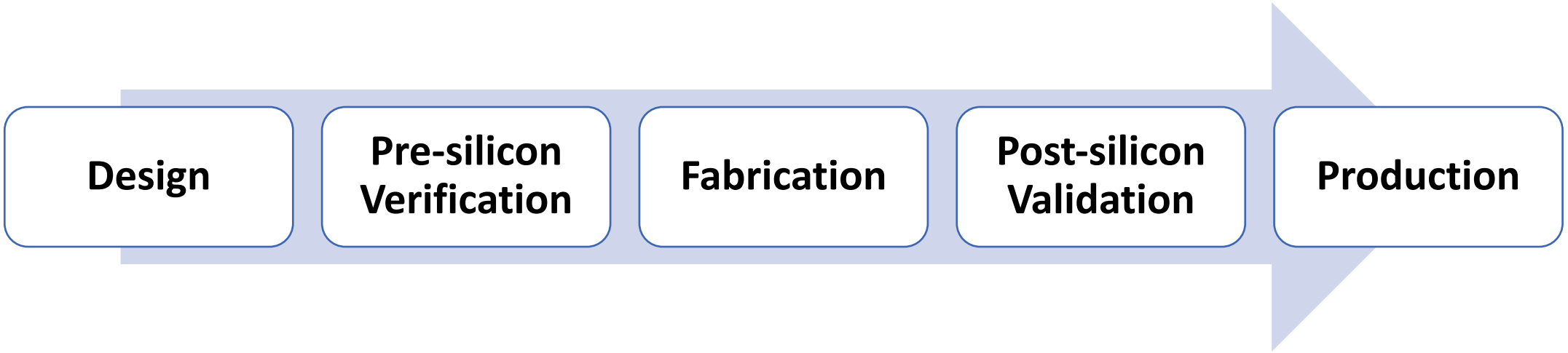
Processor implementation errata remain a problem, and worse, a subset of these bugs are security-critical. We classified 7 years of errata from recent commercial processors to understand the magnitude and severity of this problem, and found that of 301 errata analyzed, 28 are security-critical.

We propose the SECURITY-CRITICAL PROCESSOR ERRATA CATCHING SYSTEM (SPECS) as a low-overhead solution to this problem. SPECS employs a dynamic verification strategy that is made lightweight by limiting protection to only security-critical processor state. As a proof-of-

	Catch All Bugs?	Catch Security-Critical Bugs?	Low Overhead?
SW-only	✗	✗	✓
HW-only	✓	✓	✗
SPECS	✗	✓	✓
SPECS+SW	✓	✓	✓

Table 1: The design space for catching processor bugs: existing software-only approaches are limited, but practical; existing hardware-only approaches are powerful, but impractical; and SPECS, combined with existing software approaches, is both powerful and practical.

Design + Validation Process



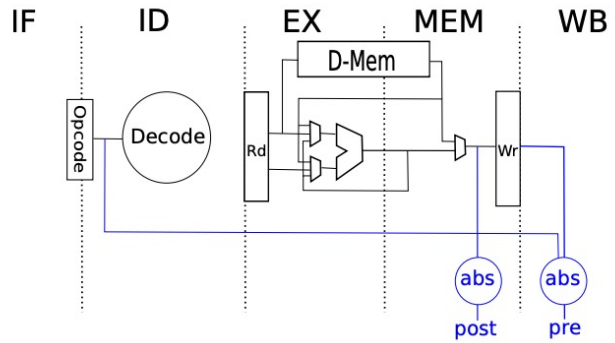
Functional Verification

	Functional	**Formal**
Operate on large designs over 10M nand gates	Yes	No
Verification scheme	Stimulus: - Directed (hand written) - Random (auto generated)	Constraint driven **exhaustive** testing
Designs	Full CPU, IP, SOC, etc.	Small blocks: Multiply, Divide, CRC, Floating-point, etc.
Development work	Testbench, stimulus, reference models, etc	Lemmas, constraint, etc.
Power aware verification	Yes	No

From "David Kaplan: When hardware must just work"

Several Research Efforts

- End-to-End Verification of ARM Processors with ISA-Formal, CAV'16



- Instruction-Level Abstraction (ILA): A Uniform Specification for System-on-Chip (SoC) Verification
  - https://princetonuniversity.github.io/isca22-ila-tutorial/

- Commercial Model Checking Tool: JasperGold

Easy v.s. Hard to Find Bugs

Easy to find bugs

- Basic functional behavior
  - Does this mode work?
  - Are exceptions correctly generated?
- Formal proofs
  - Is multiplier output correct?
- Coverage holes
  - Can all exceptions be generated?
  - Are all instructions executed?

Hard to find bugs

- System level behavior
  - Protocol violations
  - FIFO overruns/underruns
- Multiple random events
  - 5 unlikely asynchronous events in a row
- Long runtime events
- Statistically unlikely matches
  - Multiple loads where 20 address bits match

From "David Kaplan: When hardware must just work"

Hunting Hardware Bugs in the Post-Meltdown Era

Meltdown

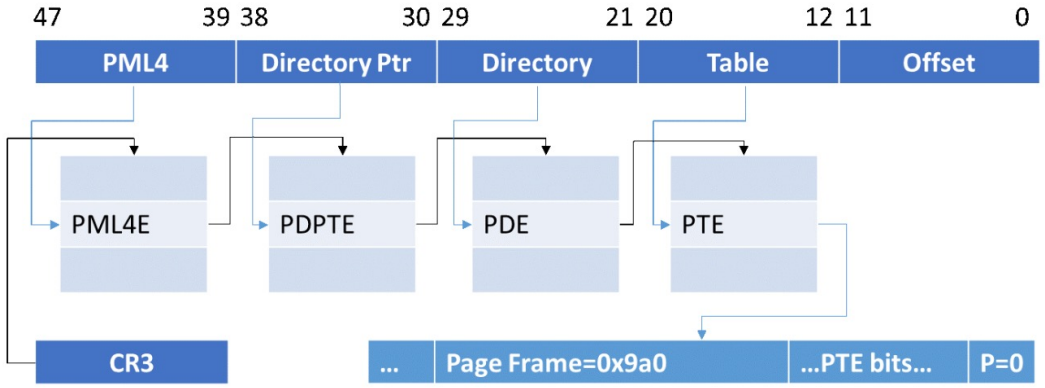
- We generally consider Meltdown as a bug (as oppose to Spectre)
- Should be easy to fix… But …
- Meltdown -> Foreshadow -> MDS -> Zombie Load
- How to end this chain?

Meltdown

```
…
x = load kernel_addr  // trigger a page fault, deferred
y = load array[x*64]
…
```

Foreshadow -- L1 Terminal Fault (L1DT)

```
…
x = load enclave_addr  // trigger a terminal fault
                       // data in L1
y = load array[x*64]
…
```

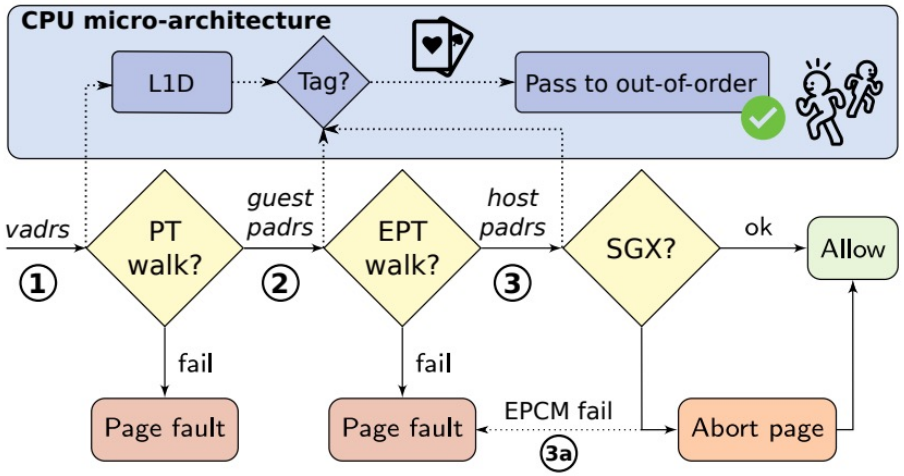


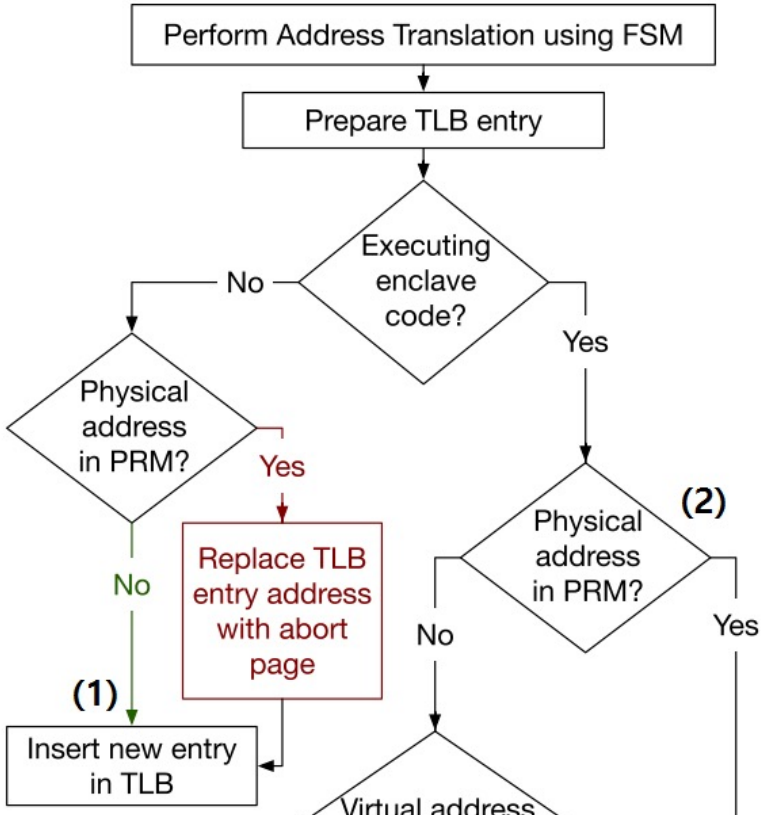
Forshadow-NG

…

```
x = load addr  // incur a terminal fault
               // if data in L1
y = load array[x*64]
```

…





Microarchitecture Data Sampling (MDS)

- Exploiting internal microarch structures:
  - Store buffer, Load port, Fill buffers

- RIDL: Rogue In-Flight Data Load
- Fallout: Leaking Data on Meltdown-resistant CPUs
- ZombieLoad: Cross-Privilege-Boundary Data Sampling

https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-analysis-microarchitectural-data-sampling.html

RIDL Explained

- The processor forwards in-flight data from the LFBs (with no addressing restrictions)

```
…
x = load NULL //invalid or
              //unmapped page
y = load array[x*64]
…
```

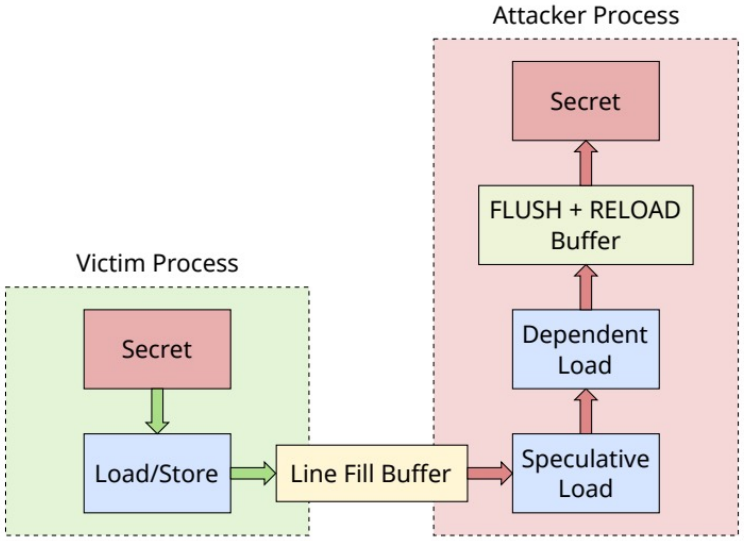


Fig. 2: An overview of a RIDL-class attack.

Meltdown -> Foreshadow -> MDS -> Zombie Load

Consider Software Bugs Hunting/Fixing

- Approach 1: Hire a lot of experts and stare at the code
  - Basically Intel was on it in the last few years **without showing the code**
  - Black-box hacking

- Approach 2: Test suite, but need to be updated. And how to generate test cases?
  - Fuzzing
  - Symbolic Execution => generally require grey-box or white-box

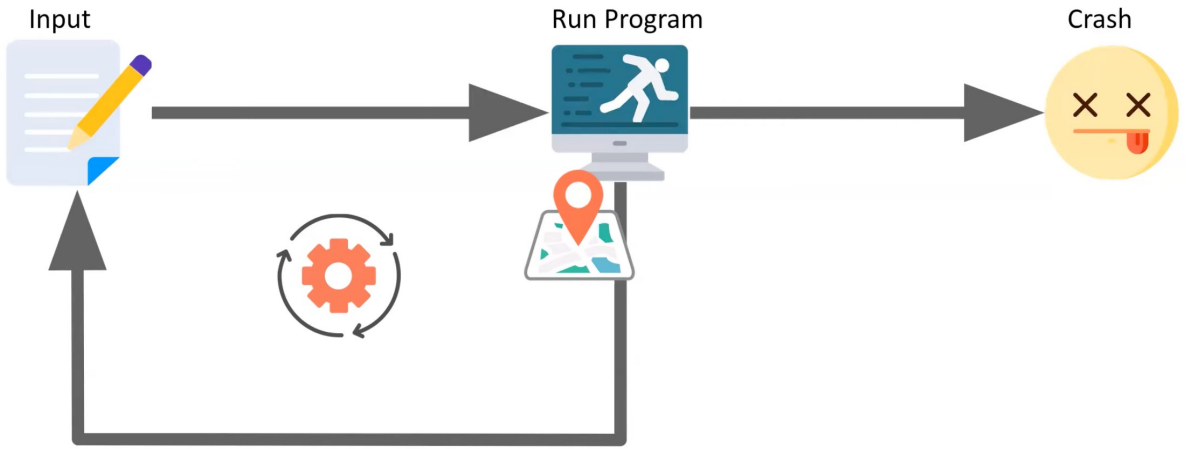
Fuzzing History

- Random: “cat /dev/urandom | program”
  - Class assignment in “Advanced Operating Systems” at University of Wisconsin (1988)
- Generation: Write a BNF spec -> introduce anomalies
  - PROTOS from OUSPG (2002) & Block-based Fuzzing from Dave Aitel (2002)
- Mutation: Assemble valid input corpus -> introduce anomalies
  - Eg. radamsa, zzuf, etc
- Apply random mutations to well-formed inputs and observe the results

Fuzzing In A Nutshell

- Automatic generate test examples
- Crash is generated by assertions/specifications

- Simple yet effective
- Industry standard



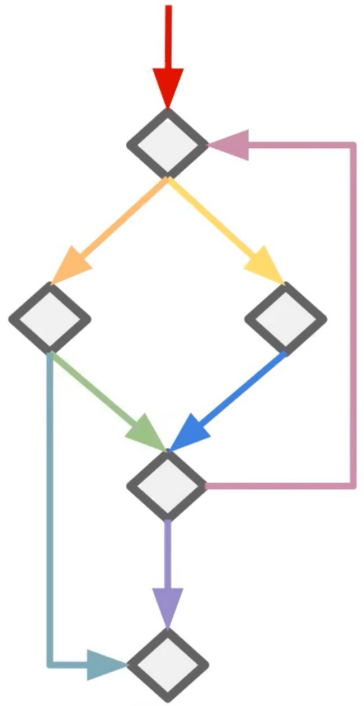
From Riding the Fuzzing Hype Train (RAID'21 Keynote)

Types of Fuzzing

- Blackbox
- Whitebox
- Greybox

```
int obscure(int x, int y)
{
   if (x==hash(y))
      error();
   return 0;

}
```



Collected coverage:

From Blackbox Fuzzing to Whitebox Fuzzing towards Verification; Patrice Godefroid Microsoft Research

Crashes != Unique Bugs

- *The highly-stochastic nature of fuzzing means that PoCs commonly exercise many program behaviors that are orthogonal to the crash's underlying root cause.*

- Examples of misclassification when detecting memory corruption bugs
  - 1. duplicate bugs crash at different addresses, e.g., invalid vptr
  - 2. different bugs crash at same function, e.g., memcpy

Igor: Crash Deduplication Through Root-Cause Clustering; Jiang et al; CCS'21

Idea: Fuzzing for Microarch Vulnerabilities

- What is the input? Instruction sequences
- What is the specification? Timing interference
- Coverage? No idea
- Actual Bug?

The Micro-benchmark Approach (I)

4. Instruction tables

Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs

By Agner Fog. Technical University of Denmark.
Copyright © 1996 – 2016. Last updated 2016-01-09.

Integer instructions

Instruction	Operands	Ops	Latency	Reciprocal throughput	Execution unit	Notes
Move instructions						
MOV	r,r	1	1	1/3	ALU	
MOV	r,i	1	1	1/3	ALU	
MOV	r8,m8	1	4	1/2	ALU, AGU	Any addr. mode. Add 1 clk if code segment base ≠ 0

The Micro-benchmark Approach (II)

uops.info: Characterizing Latency, Throughput, and Port Usage of Instructions on Intel Microarchitectures

Andreas Abel and Jan Reineke
{abel,reineke}@cs.uni-saarland.de
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany

5 Mar 2019

Abstract

Modern microarchitectures are some of the world's most complex man-made systems. As a consequence, it is increasingly difficult to predict, explain, let alone optimize the performance of software running on such microarchitectures. As a basis for performance predictions and optimizations, we would need faithful models of their behavior, which are, unfortunately, seldom available.

In this paper, we present the design and implementation

1 Introduction

Developing tools that predict, explain, or even optimize the performance of software is challenging due to the complexity of today's microarchitectures. Unfortunately, this challenge is exacerbated by the lack of a precise documentation of their behavior. While the high-level structure of modern microarchitectures is well-known and stable across multiple generations, lower-level aspects may differ considerably between microarchitecture generations and are generally not

The Micro-benchmark Approach (III)

Apple M1 Microarchitecture Research by Dougall Johnson

Firestorm: Overview | Base Instructions | SIMD and FP Instructions
Icestorm: Overview | Base Instructions | SIMD and FP Instructions

Firestorm Base Instructions

	LAT	TP	Retire	Int	Mem	FP	Units (ports)
▶ ADC	1	0.333	1	1	-	-	u1-3
▶ ADCS	1	0.333	1	1	-	-	u1-3
▶ ADD (extend)	2	0.333	1	2	-	-	2*u1-6
▶ ADD	1	0.167	1	1	-	-	u1-6
▶ ADD (shift)	2	0.333	1	2	-	-	2*u1-6

Fuzzing for Automatic Bug Detection (I)

ABSynthe: Automatic Blackbox Side-channel Synthesis on Commodity Microarchitectures

Ben Gras*†, Cristiano Giuffrida*, Michael Kurth*, Herbert Bos*, and Kaveh Razavi*

*Vrije Universiteit Amsterdam

†Intel Corporation

***Abstract*—The past decade has seen a plethora of side-channel attacks on various CPU components. Each new attack typically follows a *whitebox* analysis approach, which involves (i) identifying a specific shared CPU component, (ii) reversing its behavior**

cryptographic keys) by examining changes made by a victim's execution to the state of shared microarchitectural components such as caches [1, 2, 3, 4, 5], cache directories [6], TLBs [7], and branch predictors [8, 9]. Such attacks are typically based

Fuzzing for Automatic Bug Detection (II)

Medusa: Microarchitectural Data Leakage via Automated Attack Synthesis

Daniel Moghimi[1], Moritz Lipp[2], Berk Sunar[1], and Michael Schwarz[2]

[1]Worcester Polytechnic Institute, Worcester, MA, USA
[2]Graz University of Technology, Graz, Styria, Austria

Abstract

In May 2019, a new class of transient execution attack based on Meltdown called microarchitectural data sampling (MDS), was disclosed. MDS enables adversaries to leak secrets across security domains by collecting data from shared CPU resources such as data cache, fill buffers, and store buffers

tracting secrets that are only visible in transient states within the CPU [11]. Compared to previous side-channel attacks, the significant impact of transient-execution attacks is that they can leak actual data bits instead of access patterns.

Spectre attacks [21, 34, 35, 37, 40] miss-train branch predictors into executing control paths that might not be taken

Fuzzing for Automatic Bug Detection (III)

Osiris: Automated Discovery of Microarchitectural Side Channels

Daniel Weber, Ahmad Ibrahim, Hamed Nemati, Michael Schwarz, Christian Rossow
CISPA Helmholtz Center for Information Security

7 Jun 2021

Abstract

In the last years, a series of side channels have been discovered on CPUs. These side channels have been used in powerful attacks, e.g., on cryptographic implementations, or as building blocks in transient-execution attacks such as Spec

Side channels often arise from abstraction and optimization [79]. For example, due to the internal complexity of modern CPUs, the actual implementation, *i.e.*, the microarchitecture, is abstracted into the documented architecture. This abstraction also enables CPU vendors to introduce transpar-

Fuzzing for Automatic Bug Detection (IV)

Revizor: Testing Black-Box CPUs against Speculation Contracts

Oleksii Oleksenko*
Christof Fetzer
TU Dresden
Dresden, Germany

Boris Köpf
Microsoft Research
Cambridge, UK

Mark Silberstein
Technion
Haifa, Israel

8 Feb 2022

ABSTRACT

Speculative vulnerabilities such as Spectre and Meltdown expose speculative execution state that can be exploited to leak information across security domains via side-channels. Such vulnerabilities often stay undetected for a long time as we lack the tools for systematic testing of CPUs to find them.

In this paper, we propose an approach to *automatically* detect microarchitectural information leakage in commercial black-box CPUs. We build on speculation contracts, which we employ to specify the permitted side effects of program execution on the CPU's microarchitectural state. We propose a Model-based Relational Testing (MRT) technique to empirically assess the CPU compliance with these specifications.

For software developers, contracts are a foundation for microarchitecturally secure programming: they spell out the assumptions that are required for checking that mitigations are effective and code is free of leaks. For example, a recent survey [9] classifies existing tools for detecting speculative vulnerabilities in the language of contracts. For hardware developers, contracts can provide a target specification that describes the permitted microarchitectural effects of the CPU's operations, without putting further constraints on the hardware implementation. Thus, contracts hold the promise to achieve for speculative vulnerabilities what consistency models have provided for memory consistency [3].

Despite the contracts' potential, so far they have only been used for establishing security guarantees of small white-box models

An Example: Orisis

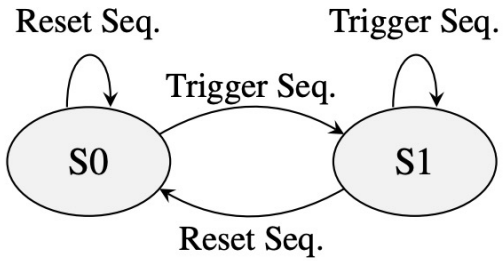


Figure 1: State machine representing different microarchitectural states and transitions between them.

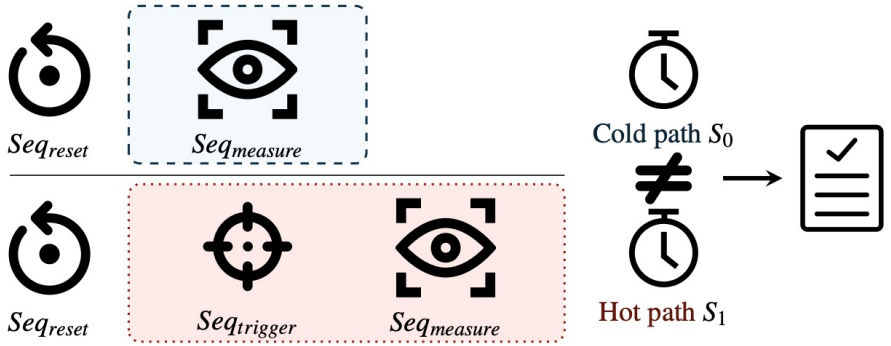


Figure 3: The execution stage receives the triple and executes $Seq_{measure}$ (cold path) and $Seq_{trigger}$, $Seq_{measure}$ (hot path) after Seq_{reset}. Timing differences for the two paths are reported.

Potential Challenges?

Challenges and Potential Solutions

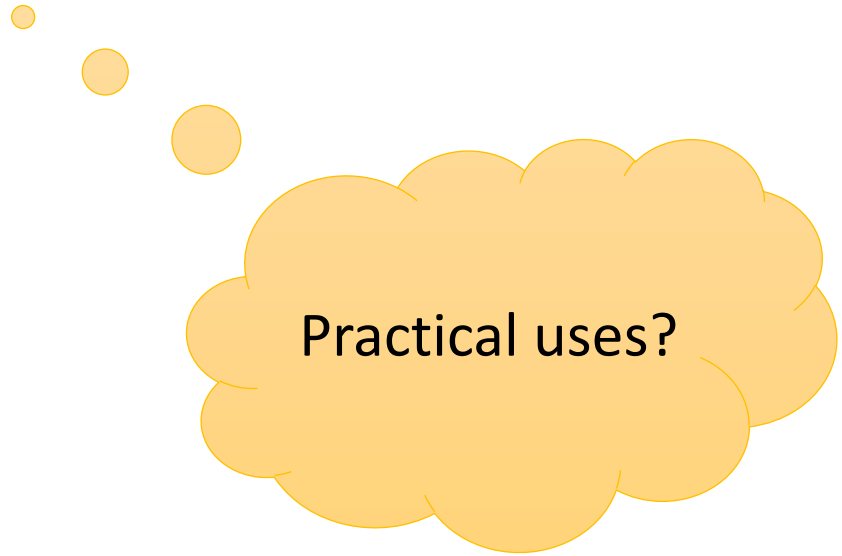
- How to ensure reset works properly?
  - => randomize the ordering of testing the triples. Old, hot ordering.
- How about noise? Frequency, prefetcher, etc.
  - => Repeat
- Applicability to transient execution?
  - Put the trigger in retpoline => perfect mis-prediction

- The majority of what they found are known attacks

Some Findings

- A faster FLUSH+RELOAD
  - This *MOVNTDQ* instruction with a **non-temporal** hint also **evicts** the accessed memory address from the cache. A later load will be faster, potentially due to coherence protocol.

```
- reset: `MOVNTDQ`
- trigger: flush
- measure: load
```



Takeaways

- There are a good number of hardware bugs
- Testing v.s. Formal verification v.s. Fuzzing
- Difficult to fix hardware bugs, e.g., Meltdown-style
  - An example of fuzzing to automatically detect vulnerabilities