Transient Execution Emulator

Meltdown and Spectre Behind the Scenes

Felix Betke, Lennart Hein, Melina Hoffmann, Jan-Niklas Sohn

2022-04-01

Rheinische Friedrich-Wilhelms-Universität Bonn

Structure

- Topic
- Background
- Our task
- Our approach
- Implementation
- Demo
- Conclusion

Topic

- Lab builds on SCA lecture
- Meltdown and Spectre mostly patched
- Difficult to experiment with
    - Personal computer often times not usable
- Goal: Vulnerable CPU Emulator that runs on many systems
    - Should offer a gdb-like interface

Background

CPU

- Frontend:
    - Fetches/Decodes instructions, maintains queue
    - Branch prediction
- Execution Engine:
    - Multiple sets of execution units
- Memory Subsystem:
    - Handles memory operations
    - Maintains L1 cache
    - Ensures data is loaded from other caches/memory

Out-of-order execution

- Independent instruction streams
- Tomasulo algorithm:
    - Reservation stations
    - Common Data Bus
- Rollbacks

Speculative execution

- Predict results of branch instructions
- Prevent stalls
- BPU maintains counters
- Rollbacks

Meltdown

- Abuses out-of-order execution
- Meltdown-US-L1:
    - Define oracle array
    - Perform illegal read to steal secret
    - Embded secret-dependent oracle entry into cache
    - Await rollback and measure oracle access times
- Small time window

Spectre

- Abuses speculative execution
- Different variations. Here: prediction of branch instrs.
- Spectre v1: Deliberately train BPU used by victim process
- Make victim leak secret into cache
- Direct consequence of speculative execution

Mitigations: Meltdown

- Disable out-of-order execution
- Intel's microcode mitigation
    - Microprograms
- OS mitigations

Mitigations: Spectre

- Disable speculative execution:
    - Completely disable
    - fence instructions
- Flush entire cache after rollback

Our task

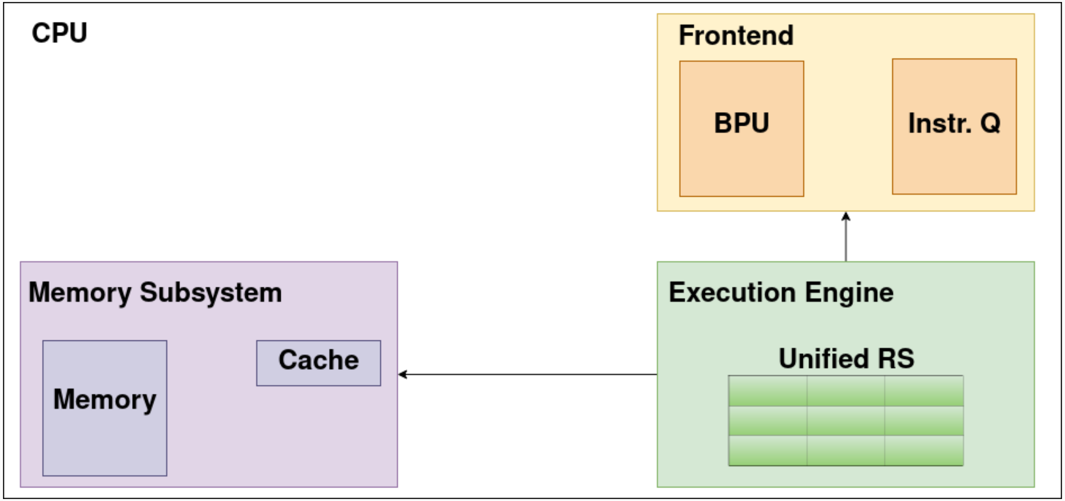
- Develop graphical CPU emulator vulnerable to:
    - Our version of Meltdown-US-L1
    - Spectre v1
- Must support single step, out-of-order, and speculative execution
- Implement Intel's microcode mitigation
- Other mitigations via microprograms
- Target audience: SCA students
    - Or anyone with basic knowledge of TE attacks

Our approach

How we started

- Must-haves, nice-to-haves, future work
- At time of Meltdown/Spectre publication: Skylake
- Filter components needed for our Meltdown/Spectre versions
- Build simplified CPU

Our version

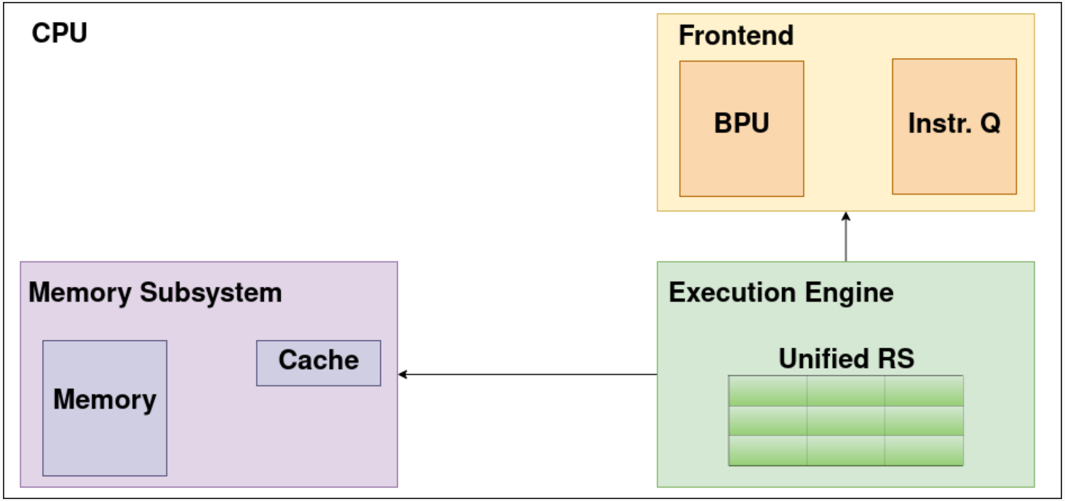

CPU
Frontend
BPU
Instr. Q
Memory Subsystem
Memory
Cache
Execution Engine
Unified RS

Implementation of our emulator

Implementation of our emulator

- overview over our whole emulator
- out-of-order execution
- speculative execution
- fault handling and rollbacks

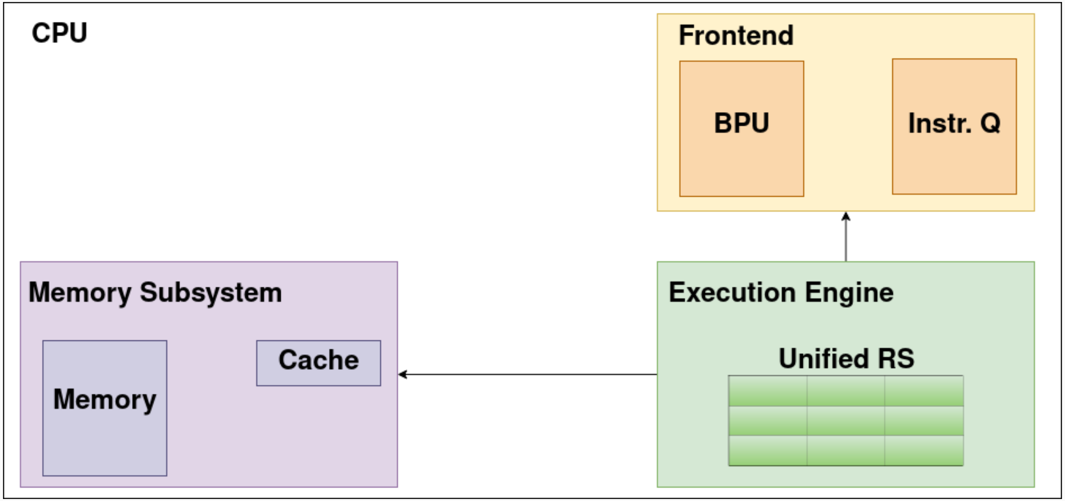
CPU class


CPU
Frontend
BPU
Instr. Q
Memory Subsystem
Memory
Cache
Execution Engine
Unified RS

Parser

- assembler style source code
- arithmetic, branch and memory instructions, fence, rtdsc
- provides an instruction list
- only one type of instructions

CPU components



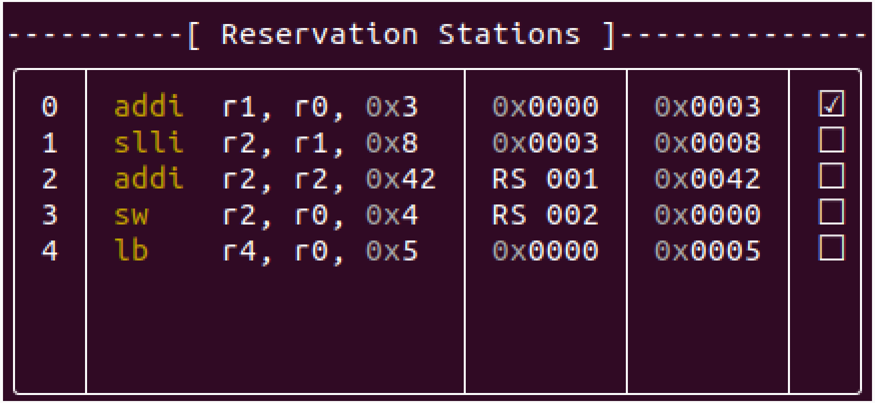
Out-of-order execution

- Execution Engine
- Tomasulos algorithm
    - unified reservation station
    - instructions wait for their operands
    - keeping track of operands and results

Issuing instructions

- resolve operands and target register
- two kinds of register values: Word and SlotID
- put register content into operand list
- put SlotID into target register

Example Reservation Station



```
----------[ Reservation Stations ]--------------

 0   addi  r1, r0, 0x3     0x0000   0x0003   ☑
 1   slli  r2, r1, 0x8     0x0003   0x0008   ☐
 2   addi  r2, r2, 0x42    RS 001   0x0042   ☐
 3   sw    r2, r0, 0x4     RS 002   0x0000   ☐
 4   lb    r4, r0, 0x5     0x0000   0x0005   ☐
```

Common Data Bus (CDB)

- execute instructions in reservation station
- broadcast the result over the CDB
    - registers
    - reservation station slots
    - at most once per cycle

Speculative execution

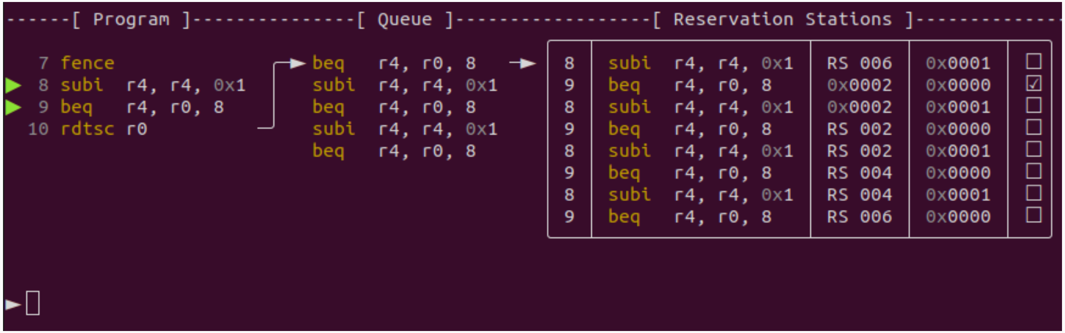
- predict outcome of branch instructions
- resume execution based on this prediction
- two central components
    - branch prediction unit (BPU)
    - CPU frontend with instruction queue

Branch Prediction Unit (BPU)

- simplified version
- array of predictions
- 2-bit-saturating counter to handle predictions

CPU frontend with instruction queue

- interface btw. instruction list and execution unit
- especially wrt speculative execution
- manages instruction queue



```
------[ Program ]--------------[ Queue ]------------------[ Reservation Stations ]------------

    7 fence                 ┌─► beq    r4, r0, 8    ─►  │ 8 │ subi  r4, r4, 0x1 │ RS 006 │ 0x0001 │ ☐
 ▶  8 subi  r4, r4, 0x1     │   subi   r4, r4, 0x1      │ 9 │ beq   r4, r0, 8   │ 0x0002 │ 0x0000 │ ☑
 ▶  9 beq   r4, r0, 8       │   beq    r4, r0, 8        │ 8 │ subi  r4, r4, 0x1 │ 0x0002 │ 0x0001 │ ☐
   10 rdtsc r0            ──┘   subi   r4, r4, 0x1      │ 9 │ beq   r4, r0, 8   │ RS 002 │ 0x0000 │ ☐
                                beq    r4, r0, 8        │ 8 │ subi  r4, r4, 0x1 │ RS 002 │ 0x0001 │ ☐
                                                        │ 9 │ beq   r4, r0, 8   │ RS 004 │ 0x0000 │ ☐
                                                        │ 8 │ subi  r4, r4, 0x1 │ RS 004 │ 0x0001 │ ☐
                                                        │ 9 │ beq   r4, r0, 8   │ RS 006 │ 0x0000 │ ☐


►▯
```

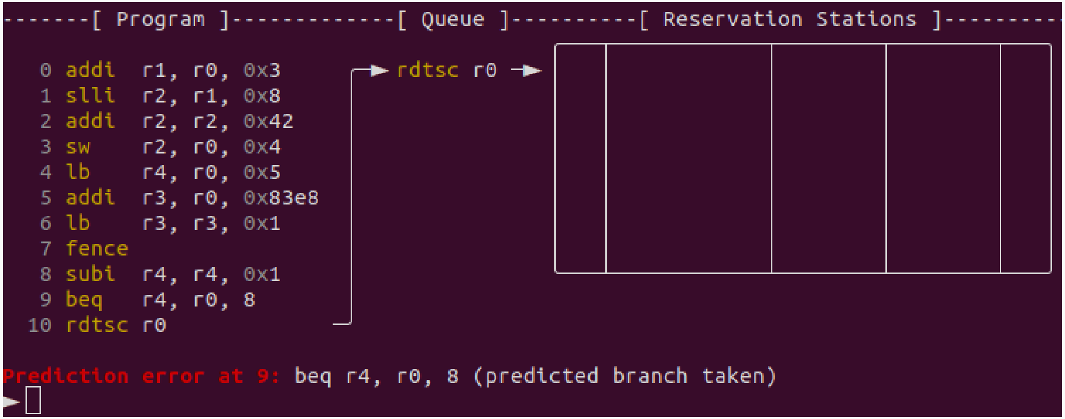
Faults and Rollbacks

- microarchitectural fault situation that has to be handled bevore we can resume our execution
    - mispredicted branches
    - attempt to access inaccessible memory
- have to handle the effects of transient execution

Rollback

- only rollback the register state and the memory contents
- no rollback in Cache and BPU
- restore register state via snapshots
- prevent memory rollbacks by executing stores in-order
- handle faults in program order

Rollback after mispredicted branch



```
-------[ Program ]------------[ Queue ]---------[ Reservation Stations ]---------

   0 addi  r1, r0, 0x3            rdtsc r0
   1 slli  r2, r1, 0x8
   2 addi  r2, r2, 0x42
   3 sw    r2, r0, 0x4
   4 lb    r4, r0, 0x5
   5 addi  r3, r0, 0x83e8
   6 lb    r3, r3, 0x1
   7 fence
   8 subi  r4, r4, 0x1
   9 beq   r4, r0, 8
  10 rdtsc r0

Prediction error at 9: beq r4, r0, 8 (predicted branch taken)
▶
```

Thank you for your attention

Do you have any questions so far?

Demo

Demo - Meltdown

```
lb r1, r0, 0xc000
slli r1, r1, 4                Encode byte into oracle array
lb r2, r1, 0x1000

fence
addi r1, r0, 0
addi r2, r0, 0xFFFF
addi r3, r0, 0x0000
addi r4, r0, 0x0FF0

probe:
fence
rdtsc r5
lb r7, r3, 0x1000             Measure access time
fence
rdtsc r6
sub r5, r6, r5

bgtu r5, r2, skip
addi r1, r3, 0                update shortest
addi r2, r5, 0
skip:

addi r3, r3, 0x10             increment and loop
bgeu r4, r3, probe

srli r1, r1, 4                shift result
```

r1	r2	r3	r4
shortest load	shortest load time	current offset	last offset

Spectre-Type Attack Demo

- Demonstrate mechanism behind Spectre-type attacks
- BPU can be trained for targeted misprediction
- Requires code sequence that encodes leaked value into cache

Spectre-Type Attack: Overview

- Prepare victim array: 8 elements, all zero
    - Followed by secret value `0x41`
- Victim loops over the array and encodes each value in the cache
    - BPU is trained to predict that the loop continues
- Final loop condition will be mispredicted
    - During transient execution: Additional iteration with out-of-bounds index
    - Secret value accessed and encoded into cache

Spectre-Type Attack: Preparation

```
// Set up array at 0x1000, 8 elements, all zero
addi r1, r0, 0x1000
sb r0, r1, 0
sb r0, r1, 1
sb r0, r1, 2
sb r0, r1, 3
sb r0, r1, 4
sb r0, r1, 5
sb r0, r1, 6
sb r0, r1, 7
// Followed by one out-of-bounds 0x41 value
addi r2, r0, 0x41
sb r2, r1, 8
```

Spectre-Type Attack: Execution

```
// Loop over array, encode every value in cache
addi r2, r0, 0    // r2: Loop index
addi r3, r0, 8    // r3: Array length
loop:
// Load array element
lb r4, r2, 0x1000
// Encode value in cache
slli r4, r4, 4
lb r4, r4, 0x2000
// Increment loop index
addi r2, r2, 1
fence
// Loop while index is in bounds
bne r2, r3, loop
```

Attack Demo

Spectre-Type Attack: Mitigation

- Flush cache after rollback
- Prevents using cache as transmission channel
- Implementation: Inject microcode after rollback
    - Inject `flushall` instruction after mispredicted branch

Mitigation Demo

Conclusion

- Goal: CPU Emulator
    - Out-of-Order Execution
    - Branch Prediction
    - Transient Execution Attacks
    - Mitigations

Further Work

References

- ja