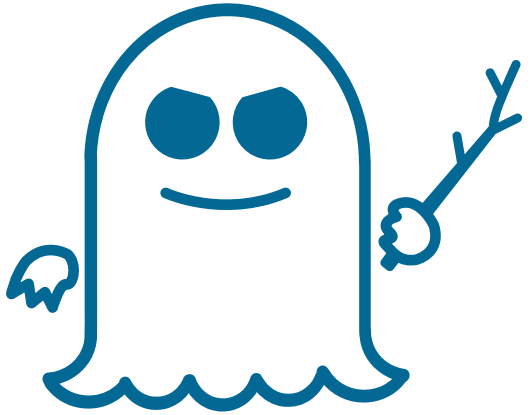
Spectre/Meltdown



Wojciech Romaszkan

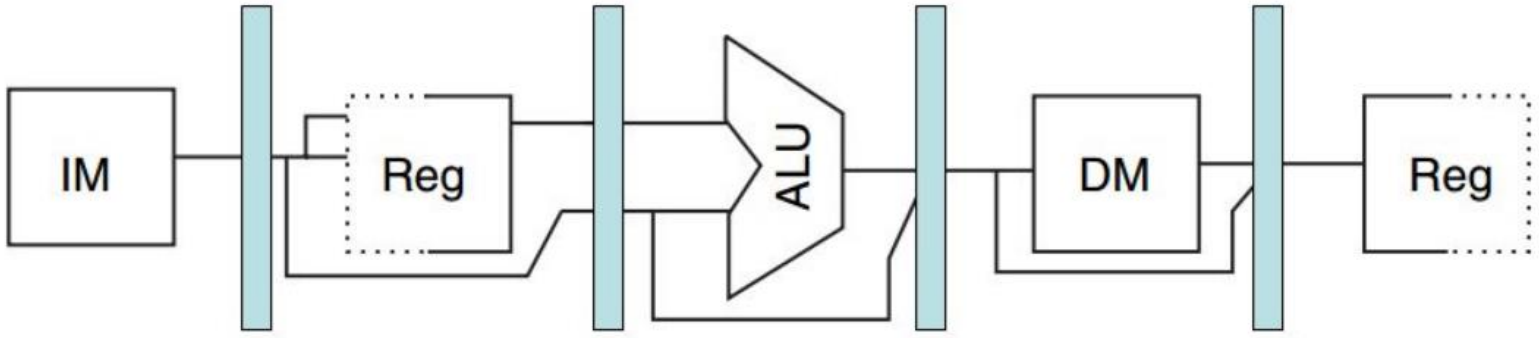
NanoCAD Lab

Outline

- Quick Computer Architecture Refresher
  - Branch prediction, out-of-order execution, speculation, virtual memory
- Cache side-channel attacks
- Meltdown
- Spectre
- Comparison Table
- References

How does a processor work?

- Comp Arch 101:



- What's wrong with that?
  - Memory is slow,
  - Serial execution is slow,
  - Branches (control hazards),
  - And many more,

How to make the processor run faster?

- Mitigate memory latency – Branch Prediction
  - Assume whether the branch will or will not be taken and fetch next instruction based on that.
  - Avoid stalling on every branch, potentially higher penalty on a misprediction.
  - So we want a high ratio of predicted to mispredicted branches.

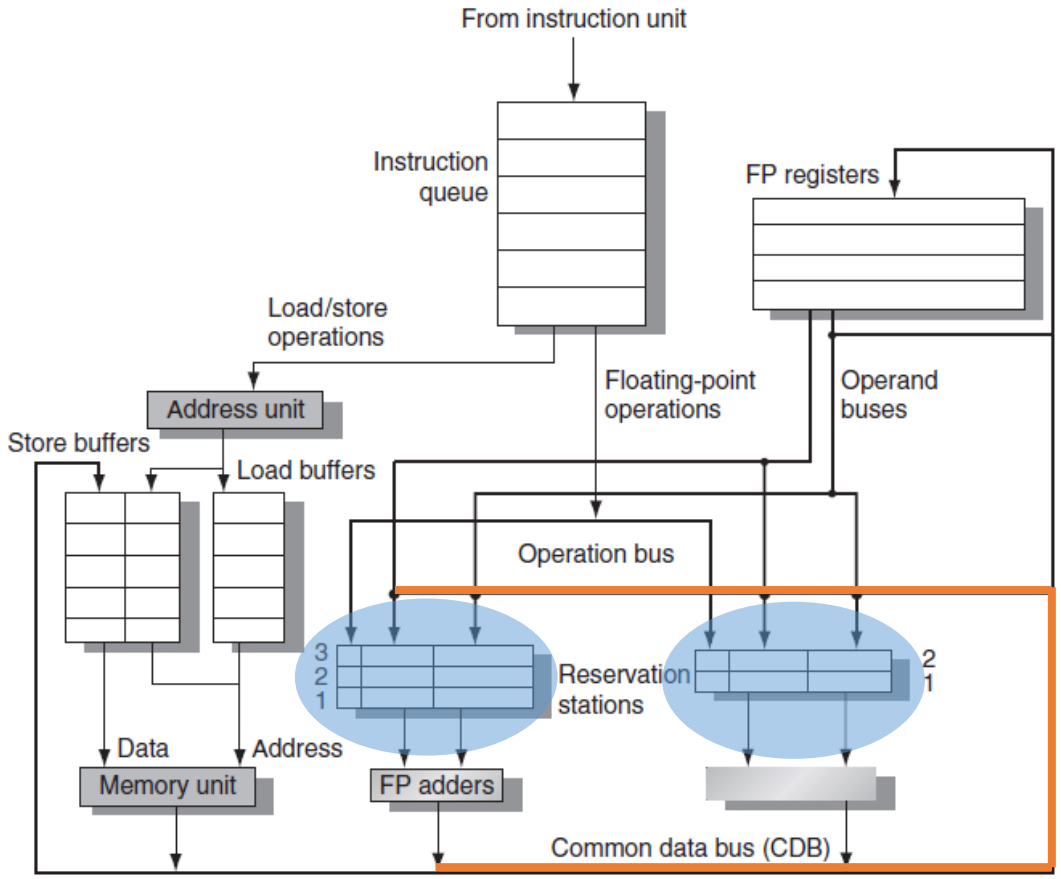
```
if (x < 10) {
…
}

x = 1,2,7,5,4,1,3,2,7,4,5,6,2,3,4,2,1,4,5,8,2,99999999
```

Out-of-order execution

```
DIV F0,F2,F4
ADD F3,F0,F8
SUB F7,F8,F6
```

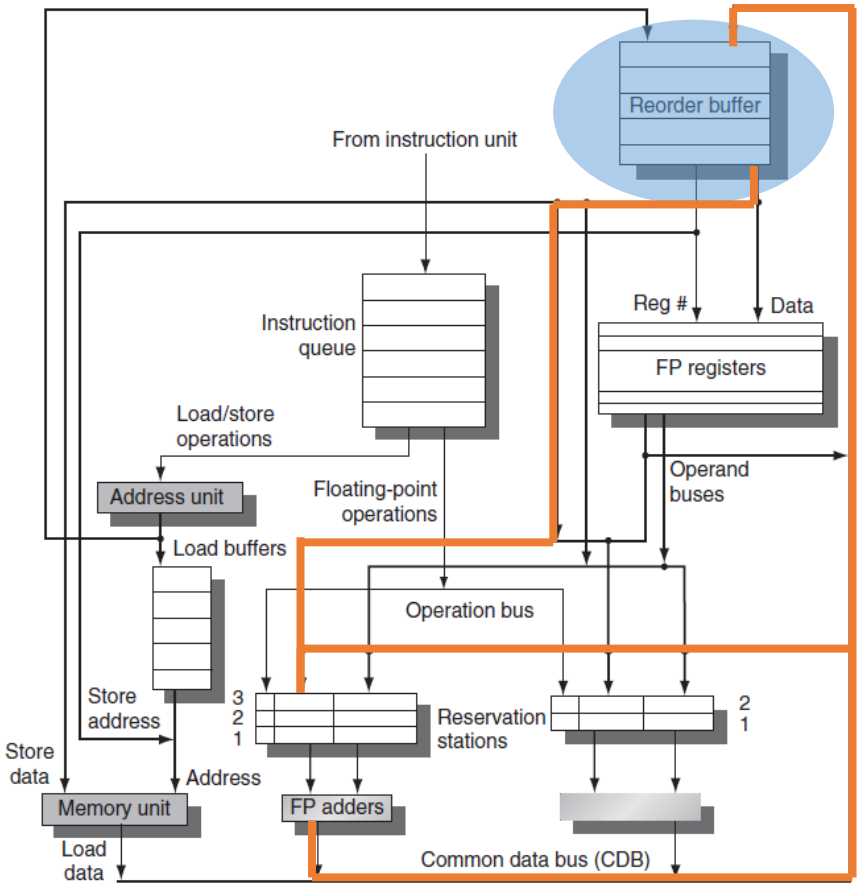
- Single pipeline in-order execution
  - One instruction can stall the upstream
  - Can we “bypass” instructions?
- Split the pipeline into multiple execution units
  - Tomasulo’s algorithm
  - Reservation stations
  - Common data bus
  - Register renaming
- What happens with exceptions?



Hennessy J., Patteron D., Computer Architecture: A Quantitative Approach, 5th Edition

Speculation

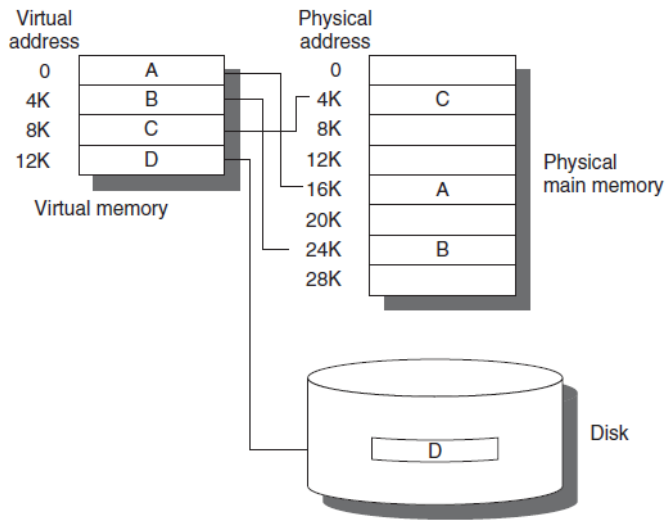
- Branch prediction only fetches the data
- It doesn't execute until we know the direction of the branch
- Why not extend that to actual speculative execution?
- Need a roll-back mechanism
- Re-order buffer
  - Instruction can execute, but not commit



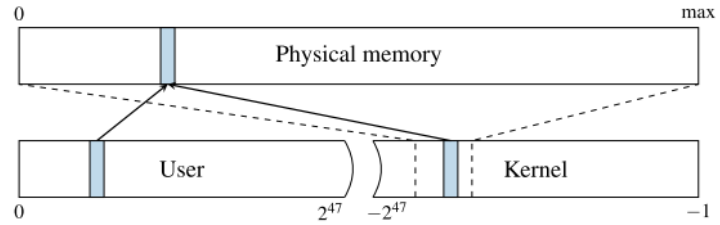
Hennessy J., Patteron D., Computer Architecture: A Quantitative Approach, 5th Edition

Virtual Memory

- Cache/mapping between main memory and secondary storage
  - Allows for sharing of memory between multiple processes
  - Protects process memory
- Each process has a virtual address space, split into user and kernel
  - Kernel address space has the whole physical memory mapped into it
  - Makes context switching faster
  - Randomization schemes exist (KASLR) but can be broken



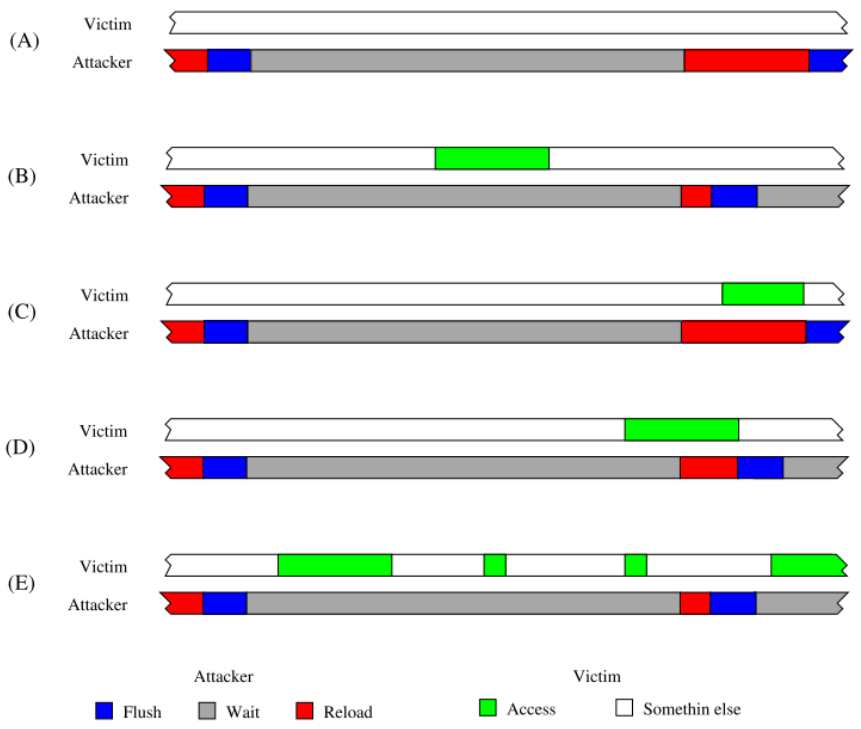
Hennessy J., Patteron D., Computer Architecture: A Quantitative Approach, 5th Edition



Lipp. M et.al., Meltdown

Cache Side-Channel Attacks

- Instead of breaking the security scheme itself, observe effects associated with it
  - e.g. timing, power, acoustic etc.
  - Cache side-channel attacks utilize timing differences
- Flush+Reload
  - Uses L3 cache – attacker and victim don't need to run on the same core
  - Attacker flushes the cache
  - Waits
  - Reloads the cache line and checks how long it takes
    - If the victim accessed that line, it is chached – shorter access
    - Attacker and victim need to share pages
    - Attacker needs to know the victim's program layout

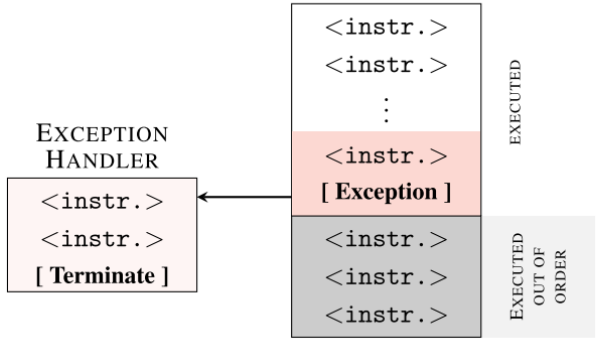


Meltdown – exploiting OOOE

- In an in-order execution pipeline an exception changes the control flow and subsequent instructions are not executed
- This is not necessarily true for OOOE processors
  - Instructions execute but are not committed
- From architectural point of view it's like they were not executed
  - Registers and memory content are discarded
- But not the cache – side channel

```
1 raise_exception();
2 // the line below is never reached
3 access(probe_array[data * 4096]);
```

Listing 1: A toy example to illustrate side-effects of out-of-order execution.



Lipp. M et.al., Meltdown

How to exploit OOOE & cache side-channel?

- Physical memory is mapped in kernel address space
- Unprivileged process trying to access kernel memory will normally raise an exception
- Because of OOOE we might be able to run a few more instructions before the exception actually triggers
  - Race condition
- We will lose the data eventually, but maybe we can set up a side channel?

Meltdown – attack sequence

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

Lipp. M et.al., Meltdown

- #4 will trigger an exception when it gets committed
- #5-7 are ready to execute as soon as data from #4 comes back (not when it gets committed)
- If there are enough instructions waiting to get committed in the ROB before #4, there's a decent chance #5-7 will execute
- Once #4 tries to commit, exception is raised and everything is discarded

Meltdown – setting up the side channel

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

Lipp. M et.al., Meltdown

- Flush+Reload
- Create a “probe” array in memory, and flush the corresponding part of the cache
  - Secrets are read 8 bits at a time, value is multiplied by page size (e.g. 4k)
  - We need 256x4096B array

Use the secret/kernel value to calculate an address into this array

  - Load a value based on this address

- “retry” jump deals with situations when we’re to slow vs the exception

Meltdown – retrieving the secret

- After that we loop through the probe array and measure access time
  - Address to the fastest access is the secret
- Do it byte by byte, we can read the **whole** physical memory of the machine
  - Slow, but not impossible
  - 503 KB/s with 0.02% error rate
- Needs to be able to execute code on the machine, but doesn’t need privileged access
- Has full control of the covert channel (doesn’t need a victim process)

Meltdown – how bad is it?

- Theoretically affects all architectures, practically shown only on Intel CPUs
  - There's a special class of instructions on Intel CPUs that make it easier to exploit the race condition and exception suppression
  - ARM lists some cores as vulnerable (A-15/57/72)
- Disable OOOE?
  - Not a viable solution
- KASLR (Kernel Address Space Layout Randomization) makes it harder but not impossible
- KAISER (Kernel Address Isolation to have Side-channels Efficiently Removed) prevents Meltdown
  - Not 100% as parts of kernel memory still need to mapped to user space. Architecture limitation.

Spectre

- Based on return-oriented programming
  - Software technique that analyses the victim binary, finds parts of code that can leak information and somehow manipulates them into doing that
  - e.g. stack overflow overwriting return address
- Spectre tricks the CPU into incorrectly executing victim’s instructions and leaking information
- Information is retrieved using cover channel
  - Flush+reload, same as Meltdown

Spectre – training branch predictors

- Find a conditional code snippet (‘gadget’) where attacker can control the data
  - e.g. range check
  - gadget can attacker’s own code (e.g. if it’s sandboxed)

```
if (x < array1_size) {
   y = array2[array1[x] * 256]
}
```

- Invoke it N times to train branch prediction for this particular branch
- Also flush the cache to prepare the covert channel

Spectre – attack phase

- After the branch predictor is trained, the code is executed once more with the x value out of bounds
- Branch prediction + speculation means that the code will be executed, but not commited
- From this point onward it’s pretty much the same as Meltdown
  - fetch secret data using out-of-bound x as an address
  - use that data as an address to the probe array

Spectre – how bad is it?

- Works on Intel, AMD, ARM.
- Can’t read kernel/physical memory, can only read other processes information
- Harder to exploit than Meltdown (requires finding and using ‘gadgets’)
  - Not impossible and most likely would target browsers
- Harder to mitigate
  - It’s a class of attacks, not a single one (2 variants at the moment)
  - Browser updates (e.g. site isolation)
  - Processor microcode updates (performance drops)
  - Google’s “retpoline” (compiler level)

Comparison

- Meltdown uses OOOE+Speculation, Spectre uses Branch Prediction+Speculation
- Both use Flush+Reload as covert channel (other options possible)
- Meltdown can read the whole physical memory, Spectre only the memory of the process it's attacking
- Meltdown is mitigated with KAISER, Spectre mitigation techniques are not foolproof and have performance impact

	Meltdown	Spectre
Allows kernel memory read	Yes	No
Was patched with KAISER/KPTI	Yes	No
Leaks arbitrary user memory	Yes	Yes
Could be executed remotely	Sometimes	Definitely
Most likely to impact	Kernel integrity	Browser memory
Practical attacks against	Intel	Intel, AMD, ARM

References

- Lipp M., et.al, Meltdown, ArXiv e-prints, Jan 2018
- Kocher P., et.al, Spectre Attacks: Exploiting Speculative Execution, ArXiv e-prints, Jan 2018
- Yarom Y., Falkner K., Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack