SIGPwny

General FA2025 • 2025-11-16

SIGPwny x SIGArch: CPU Security

Presented by Swetha (SIGPwny) and Pratyay (SIGArch) website

SIGARCH

discord

University of Illinois' Premier Computer Architecture Club

We host weekly meetings Wednesdays from 5:00-6:00 pm in ECEB 3015.

Teaching topics including RTL design and simulation, SystemVerilog, transactional memory, branch prediction techniques and cache coherence.

Meeting content can be found at sigpwny.com/meetings.

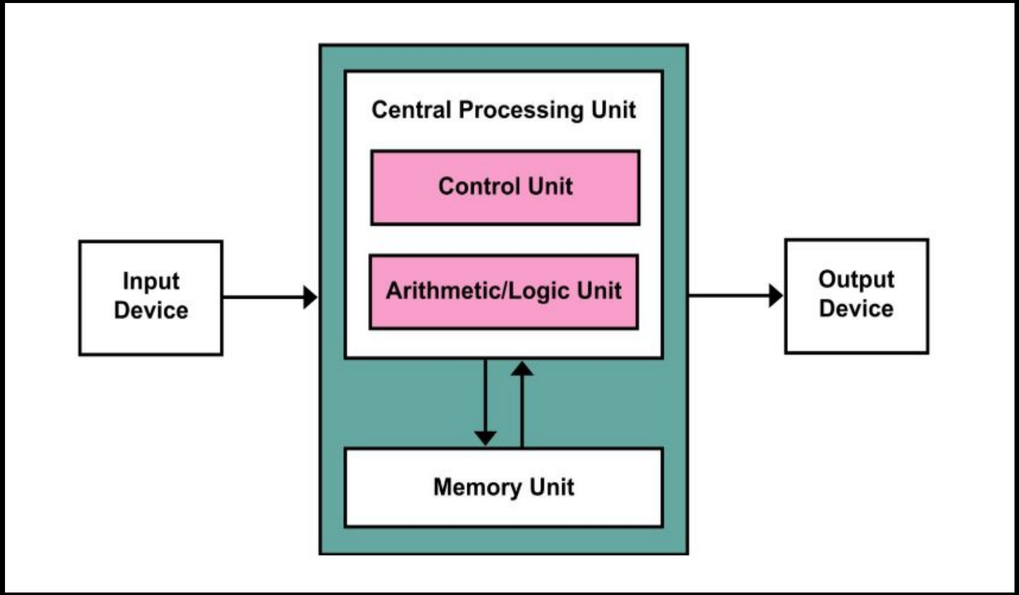
SIGPwny

Agenda

- What is a CPU?
- CPU optimization techniques
- 3 CPU vulnerabilities that exploit these optimizations
- Hands-on experiment! (stay tuned for meeting flag)

What is a CPU?

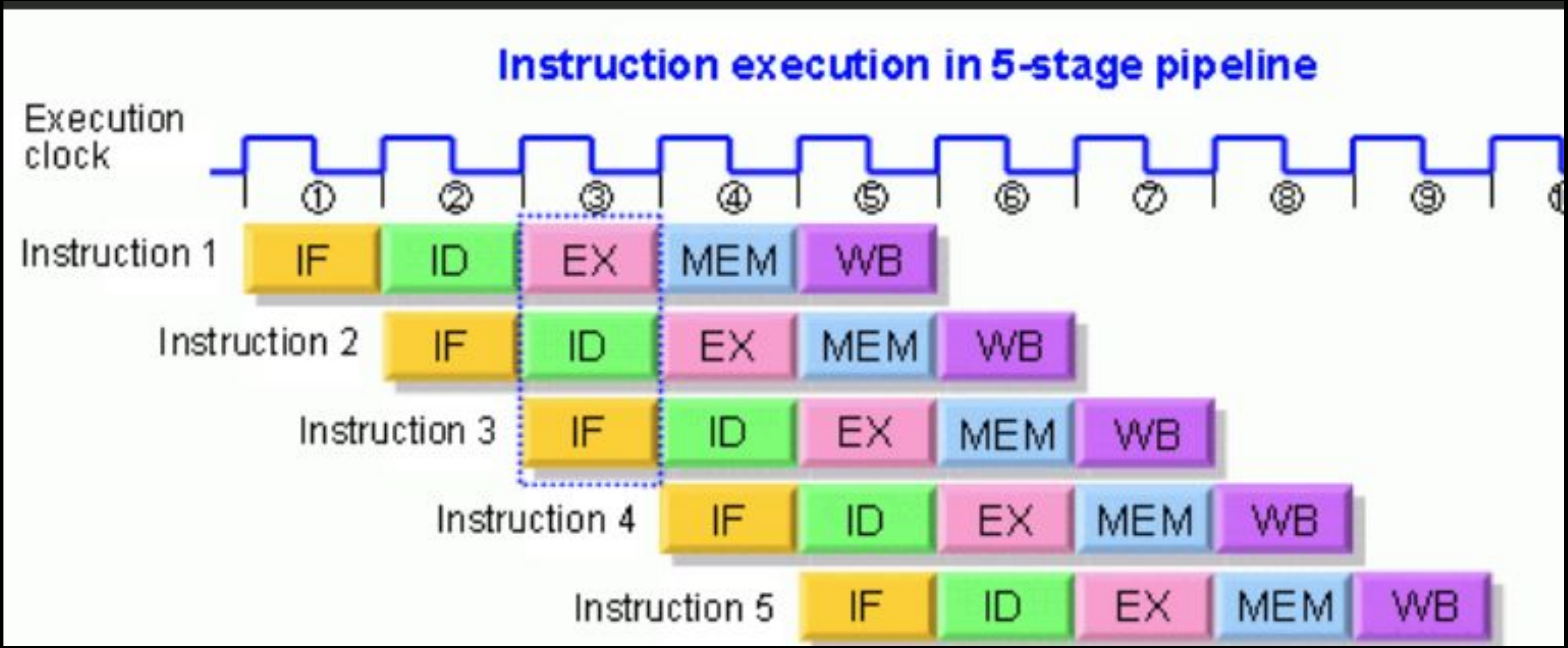
- **Fetch:** The CPU retrieves instructions from the computer's memory (RAM).
- **Decode:** It then decodes the instructions to understand what task it needs to perform.
- **Execute:** The CPU performs the required computation or logic using its internal components, such as the Arithmetic/Logic Unit (ALU) and registers.
- **Memory & Writeback:** The results are then sent back to memory for storage or for use by other components



Pipelining

- Instruction execution is broken down into stages
  - Fetch, Decode, Execute, Memory, Write-Back
- CPU works on multiple instructions simultaneously
- Starts next instruction as soon as previous one moves onto the next stage
  - Instruction A goes to Decode stage → Instruction B goes to Fetch stage
- Reduces number of clock cycles required to run the same set of instructions

Pipelining


Instruction execution in 5-stage pipeline
Execution clock
①
②
③
④
⑤
⑥
⑦
⑧
⑨
Instruction 1
IF
ID
EX
MEM
WB
Instruction 2
IF
ID
EX
MEM
WB
Instruction 3
IF
ID
EX
MEM
WB
Instruction 4
IF
ID
EX
MEM
WB
Instruction 5
IF
ID
EX
MEM
WB

Branch Prediction

- PC (program counter) typically increments sequentially
- Branch: PC “jumps” to a non-consecutive value

- Branching is based on conditions, but conditions take time to evaluate → CPU “predicts” which branch will be taken based on historical patterns, like branch history tables

- CPU speculatively executes instructions from predicted path

- Once branch condition is actually evaluated, the prediction is validated (most CPUs have ~95% prediction accuracy)

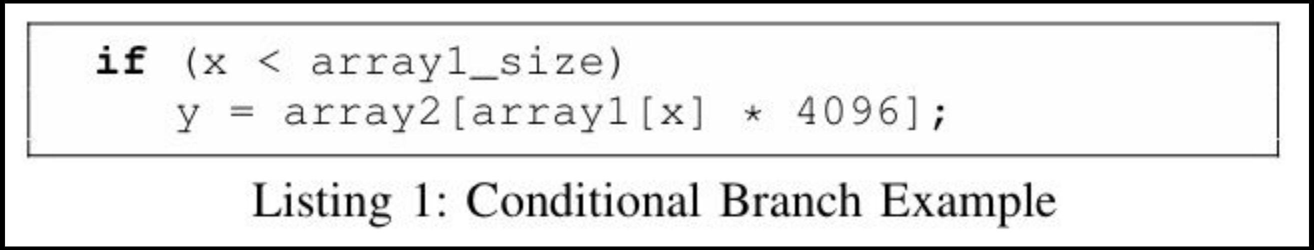
Vulnerability: Spectre

- Discovered in 2017, publicly disclosed in 2018
- Exploits branch prediction to make CPU speculatively execute attacker code paths
- Origin of name: “speculative execution”
- Official MITRE references: CVE-2017-5753 and CVE-2017-5715
- Paper: spectre.pdf

How Spectre Works

- Attacker repeatedly runs certain instructions to train the branch predictor → triggers misprediction
- During the misprediction, the CPU speculatively executes malicious instructions, accessing forbidden memory
- Though the speculative work gets discarded once the misprediction is detected, there are traces of the (secret) data in the CPU cache
- Attackers can then obtain the secret data from the cache via side channel attacks and ROP attacks

Example of Branch Misprediction



```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

Listing 1: Conditional Branch Example

- Repeatedly call this chunk of code with valid (within bounds) values of x → trains branch predictor
- Flush the cache (important for after speculative execution)
- Then provide a malicious value of x (accessing secret memory)
- While the if-statement is being evaluated, the CPU speculatively executes the array access → y value is in the cache
- Side-channel analysis on the cache → see which memory access is the fastest → contains the secret data

Out-of-Order Execution

- CPU executes instructions in a different order than they appear in the program

- Reduce performance penalties by eliminating false dependencies between nearby instructions

OOOE Example

1. Load data from memory into register A	takes 200 clock cycles
2. Add 5 to the value in register B	takes 1 clock cycle
3. Multiply value in register C by 2	takes 3 clock cycles
4. Use value in register A for a calculation	

How can we optimize the execution of these instructions?

Execute instructions #2 and #3 while #1 is happening, since there is no actual dependency between them

Vulnerability: Meltdown

- Discovered in 2017, publicly disclosed in 2018
- Exploits out-of-order execution to read kernel memory from the user space
- Primarily affected Intel CPUs
- Origin of name: “melts down” security boundaries
- Official MITRE reference: CVE-2017-5754

How Meltdown Works

- Attacker's malicious instruction tries to read from kernel memory - which is supposed to cause an exception

- Due to out-of-order execution, the CPU executes the memory access before the permission check completes

- Once the permission check is complete, the work from the out-of-order execution is discarded, but traces are still left in the cache

- Paper: meltdown.pdf

Prefetching

- Prefetching is a computer architecture technique that predicts and fetches data or instructions into the cache before they are requested, aiming to reduce memory access latency and improve performance
- Relies on principles like spatial locality (data near each other in memory is likely to be used next) and temporal locality (recently used data will be used again soon) to predict future needs
- Latency Hiding: This prefetch request runs in parallel with the processor's other computations to prevent unnecessary wait time or stalling.

Vulnerability: GoFetch

- The GoFetch attack is based on a CPU feature called data memory-dependent prefetcher (DMP), which is present in the latest Apple processors.
- Reverse-engineered DMPs on Apple m-series CPUs and found that the DMP gets data loaded from memory that "looks like" a pointer.
- This explicitly violates a requirement of the constant-time programming paradigm, which forbids mixing data and memory access patterns.
- Due to this being a hardware construct, it is hard do disable in software without turning off the fetch

Activity!

- Make an account on EDA Playground
- https://edaplayground.com/x/Adzk

Next Meetings

Meetings continue after fall break!

(No meeting this Thursday)