ECE 574 – Cluster Computing
Lecture 8

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

21 September 2026

Announcements

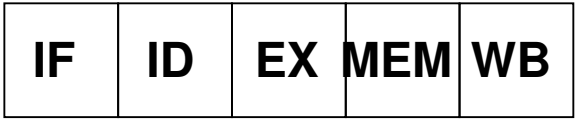
- HW#3 Posted. Fixed an issue with it.
- HW#1 Grades Sent Out

More Computer Architecture Review

- Maybe you've seen this before, but UMaine doesn't have an explicit class for grad-level Computer Architecture
- I sometimes teach ECE571 which covers this all in more detail, but I don't know if/when it will be taught again
- It used to be my primary area but I got a bit cynical about it, moreso after the big ISCA conference peer-review scandal

Pipelined CPUs

- 5-stage MIPS pipeline
- From 2-stage to Pentium 4 31-stage
- Example – single instruction always take 5 cycles? But what about on average? (Theoretical max IPC 1.0)



- Overlap execution of 5 instructions at a time

Pipeline Stages

- IF = Instruction Fetch.
  Fetch 32-bit instruction from L1-cache
- ID = Decode
- EX = execute (ALU, maybe shifter, multiplier, divide)
  Memory address calculated
- MEM = Memory – if memory had to be accessed, happens now.
- WB = register values written back to the register file

Data Hazards

Happen because instructions might depend on results from instructions ahead of them in the pipeline that haven't been written back yet.

- RAW – "true" dependency – problem. Bypassing?
- WAR – "anti" dependency – not a problem if commit in order
- WAW – "output" dependency – not a problem as long as ordered
- RAR – not a problem

Structural Hazards

- CPU can't just provide. Not enough multipliers for example

Control Hazards

- How quickly can we know outcome of a branch/jump?
- Maybe not fast enough to avoid a pipeline stall
- Ways to avoid:
  - On original 5-stage pipeline they had the "Branch Delay Slot"
  - ARM32 has conditional execution / x86 conditional move
  - Neither of the above play well with OoO processors
  - Modern processors use branch prediction

Branch Prediction

- Predict (guess) if a branch is taken or not.
- What do we do if guess wrong? (have to have some way to cancel and start over)
- Modern predictors can be very good, greater than 99%

Branch Prediction – Brief Overview

- Designs are complex and could fill an entire class
- Simple "static" predictors (backward always taken) can do remarkably well on loops
- Dynamic predictors that track branch history with branch state and hash tables (some multi-level) are better
- Winner of competitions are "perceptron" (neural-network/AI) and some modern processors use those, though it can be hard to find info on them

Branch Prediction Mitigation

- What can you do if bad branch prediction?
- Some processors let you give "hints" that go in the assembly code. It turns out often people are bad at giving hints and are worse than nothing

Memory Delay

- Memory/cache is slow
- Need to bubble / Memory Delay Slot

The Memory Wall

- Wulf and McKee
- Processors getting faster more quickly than memory
- Processors can spend large amounts of time waiting for memory to be available
- How do we hide this?

Caches (can teach whole class on these)

- Basic idea is that you have small, faster memories that are closer to the CPU and much faster
- Data from main memory is cached in these caches
- Data is automatically brought in as needed.
- Modern systems often have multiple levels of cache. Usually a small (32k or so each) L1 instruction and data, a larger (128k?) shared L2, then L3 and even L4.
- Modern systems also might share caches between processors, more on that later

Cache Background

- Temporal Locality – if you use memory once, likely to use it again soon
- Spatial Locality – if you use memory, likely to use something nearby
- Miss Types
  - Cold/Compulsory (first time accessed)
  - Capacity (cache not big enough)
  - Conflict (different addresses hashing to same cache location)
  - Coherency (issues with multi-core)

Other Cache Topics

- Skipping various other relevant topics
- Tags (how you can tell if location in cache matches location in main memory)
- Virtual Memory
- Cache-ways (direct-mapped vs set associative caches)
- Cache Replacement Policies
- Inclusive/Exclusive
- Write-back vs Write-through

Pre-fetching

- To avoid cold misses, data can be pre-fetched, either explicitly by program or by the hardware guessing.
- What are the downsides of pre-fetching?
  - If wrong, waste time/energy
  - If overly aggressive can make performance worse

Branch Cache Mitigation

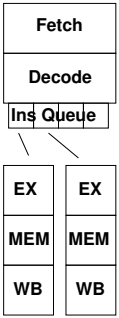
- What can you do if bad cache behavior?
- Cache conscious data placement. Try to keep things small and close together
- Change order of loops to make sure accessing things in row order
- There are specific things you can do if parallel code having trouble (avoid false sharing)

Exploiting Parallelism

- How can we take advantage of parallelism in the control stream?
- Can we execute more than one instruction at a time?

Multi-Issue (Super-Scalar)

- Decode up to X instructions at a time, and if no dependencies issue at same time.
- Dual issue example. Can have theoretical IPC of 2.0
- Can have unequal pipelines.



Out-of-Order

- Tries to exploit instruction-level parallelism
- Instead of being stuck waiting for a resource to become available for an instruction (cache, multiplier, etc) keep executing instructions beyond as long as there are no dependencies
- Need to ensure that instructions commit in order

Out-of-Order Challenges

- Can have hundreds of instructions "in-flight"
- What happens on exception? (interrupt)
  Has to somehow reconstruct the in-order state of the processor to give you a roughly accurate idea of what instruction caused the problem (source of skip, "precise" exceptions)
- Speculative execution / Branch Prediction Mispredict?
  Have to somehow stop everything and replay

Out-of-Order Challenges

- What if you run out of registers? Register Renaming
- How do you keep finished instructions but make sure they actually commit processors state in order? Re-order buffer/RAT
- How do you handle loads/stores happening out of order?
  load/store buffer

Out-of-Order Notes

- Massively complicated, but almost always a performance win
- Enough so that even small embedded processors these days might use it
- Can have security issues if you aren't careful about handling speculative loads/stores (see Meltdown/Spectre)