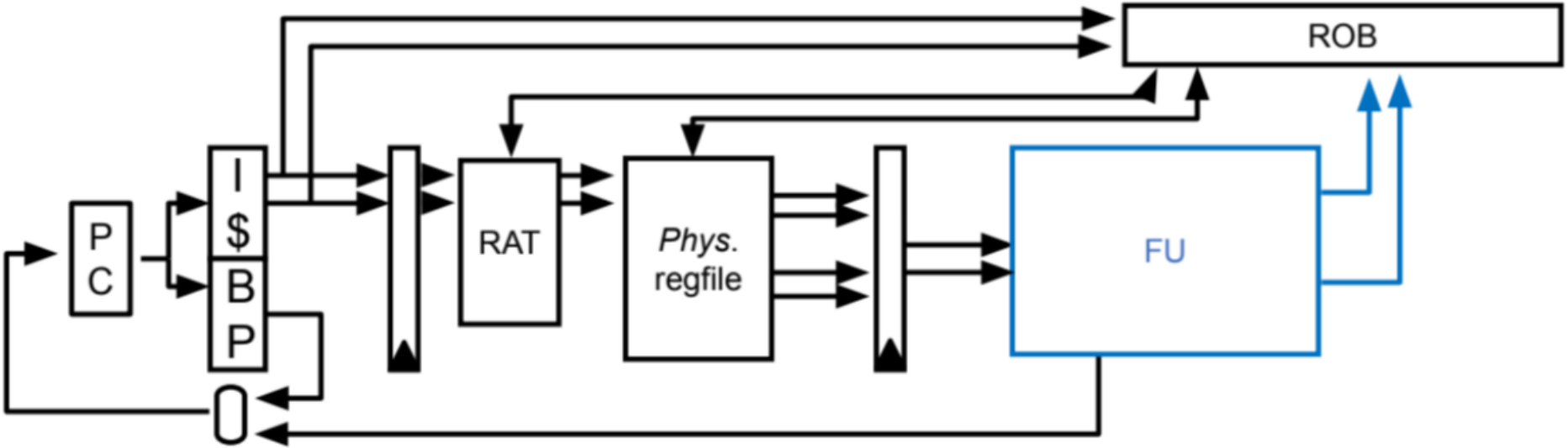
ECE M116C / CS 151B: Week 6 Section

Justin Feng

Architecture



Out of Order Execution Steps

1) Fetch
2) Decode
3) Rename
4) Dispatch
5) Issue
6) Complete
7) Retire

Out of Order Execution Steps

1) **Fetch – Grab instruction from instruction memory**
2) Decode
3) Rename
4) Dispatch
5) Issue
6) Complete
7) Retire

Out of Order Execution Steps

1) Fetch – Grab instruction from instruction memory
2) **Decode – Parse instruction, set control signals, etc. (Don't grab register file values!)**
3) Rename
4) Dispatch
5) Issue
6) Complete
7) Retire

Out of Order Execution Steps

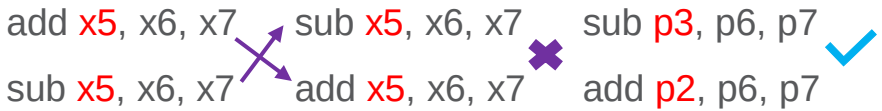
1) Fetch – Grab instruction from instruction memory
2) Decode – Parse instruction, set control signals, etc. (Don't grab register file values!)
3) **Rename – rename architectural register (rd) to a free physical register**
4) Dispatch
5) Issue
6) Complete
7) Retire

Register Renaming

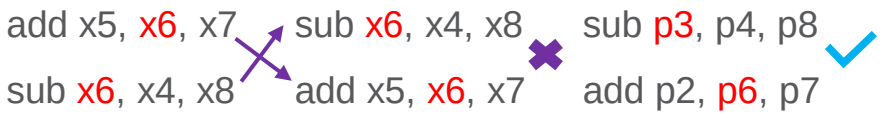
- Architectural Register (x0-x32) vs Physical Register (128 registers)
- Use Register Alias/Map Table (RAT)
  - Destination Register: rename with free physical register
  - Operand: access RAT to find current mapping
- Purpose: removes WAW and WAR dependencies (false dependencies) - works when paired with retiring in order
- When to free: after retire!
  - Say x2 is currently assigned to p2
    - add p3, p2, p5
    - sub x2, p3, p4
  - Sub's x2 will be renamed (say to p6)
    - add p3, p2, p5
    - sub p6, p3, p4
  - Once the sub retires, we can release the old mapping of x2 to p2 as no more instructions below the sub will have that mapping!

Register Renaming

WAW:



WAR:



Register Renaming

-What about RAW?

We only issue when both source operands are ready

and x3, x2, x1

sub x5, x3, x2 -> don’t issue “sub” until “and” is “complete”

Out of Order Execution Steps

1) Fetch – Grab instruction from instruction memory
2) Decode – Parse instruction, set control signals, etc. (Don't grab register file values!)
3) Rename – rename architectural register (rd) to a free physical register
4) **Dispatch – Place instructions in reservation station (RS), and in re-order buffer (ROB). Grab register file values. Mark destination registers as "not ready"**
5) Issue
6) Complete
7) Retire

Out of Order Execution Steps

1) Fetch – Grab instruction from instruction memory
2) Decode – Parse instruction, set control signals, etc. (Don't grab register file values!)
3) Rename – rename architectural register (rd) to a free physical register
4) Dispatch – Place instructions in reservation station (RS), and in re-order buffer (ROB). Grab register file values. Mark destination registers as "not ready"
5) **Issue – If source registers are ready, and functional unit (FU) is available, execute the instruction (remove entry in RS, but not ROB). Mark FU as "not ready"**
6) Complete
7) Retire

Out of Order Execution Steps

1) Fetch – Grab instruction from instruction memory
2) Decode – Parse instruction, set control signals, etc. (Don't grab register file values!)
3) Rename – rename architectural register (rd) to a free physical register
4) Dispatch – Place instructions in reservation station (RS), and in re-order buffer (ROB). Grab register file values. Mark destination registers as "not ready"
5) Issue – If source registers are ready, and functional unit (FU) is available, execute the instruction (remove entry in RS, but not ROB). Mark FU as "not ready"
6) **Complete – Copy result to ROB and mark as complete, release the FU, mark destination register as "ready". Enables recovery of instructions in case of mistake**
7) Retire

Out of Order Execution Steps

1) Fetch – Grab instruction from instruction memory
2) Decode – Parse instruction, set control signals, etc. (Don't grab register file values!)
3) Rename – rename architectural register (rd) to a free physical register
4) Dispatch – Place instructions in reservation station (RS), and in re-order buffer (ROB). Grab register file values**.** Mark destination registers as "not ready"
5) Issue – If source registers are ready, and functional unit (FU) is available, execute the instruction (remove entry in RS, but not ROB). Mark FU as "not ready"
6) Complete – Copy result to ROB and mark as complete, release the FU, mark destination register as "ready". Enables recovery of instructions in case of mistake
7) **Retire – overwrite register file. Retire instructions at top of ROB. Release "old" physical register to architectural register mapping**

Caches

Why Caches?

- Caches allow for quick and easy access for often used items. Closer to CPU.
- If need other items, can go deeper in memory hierarchy, but takes longer to access and results in stalls. Farther from CPU.
- Goal: increase speed of memory accesses over time (reduce stalling)

Caches: Important Info

Formula: Avg Access Time = Hit time + Miss Rate x Miss Penalty

Hit time: time to access L1 cache

Miss Rate: average frequency of L1 cache misses

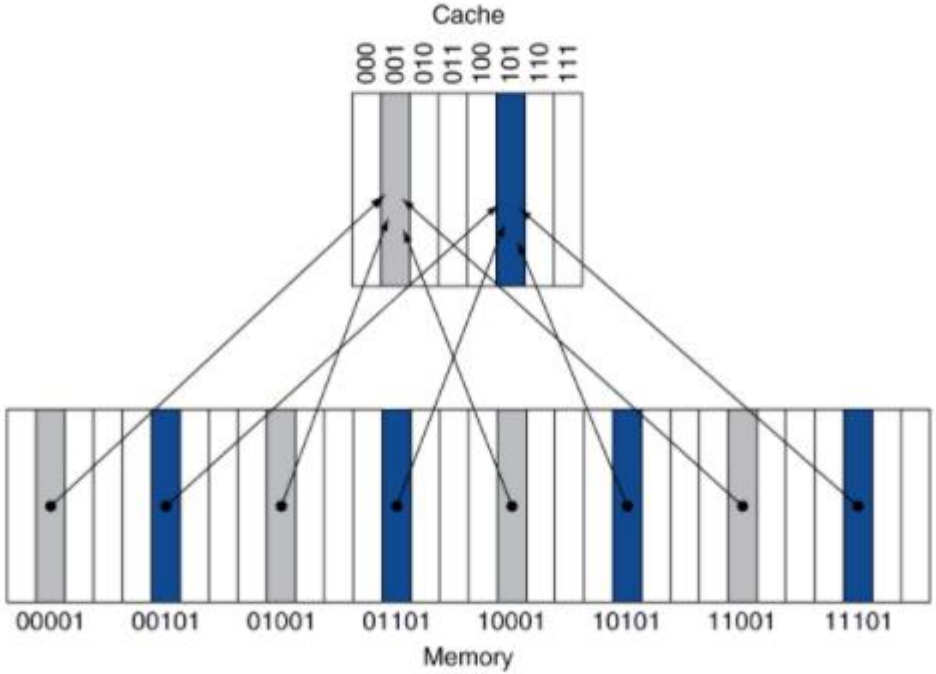
Miss Penalty: time to access main memory

3 Types:

1) Direct Mapped
2) Fully Associative
3) N-way Set Associative

-Caches block addressable (not byte!)

Direct Mapped Cache



Full Comparison