COM SCI M151B Week 4

Aidan Jan

October 22, 2024

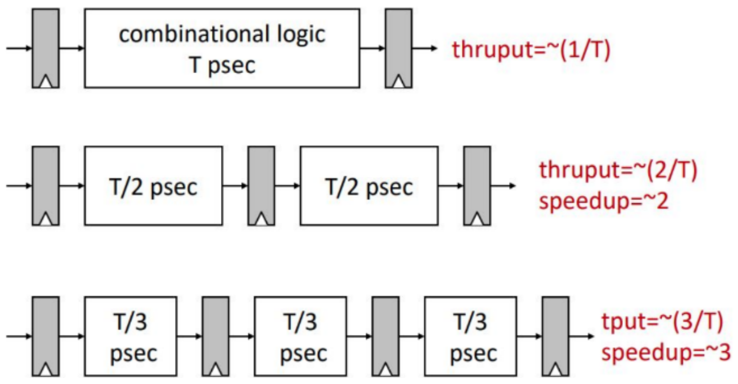
Review

ISA

- First step in the design process
- The ISA (instruction set architecture) is essentially the assembly code. Ex. RISC-V.
- ISA converts to machine code using a standard table.

The Iron Law of Processor Performance

- For single cycle design:
- CPU Time = $InstructionCount \times CyclePerInstruction \times CycleTime$
- Allowing for parallelism (overlaps)

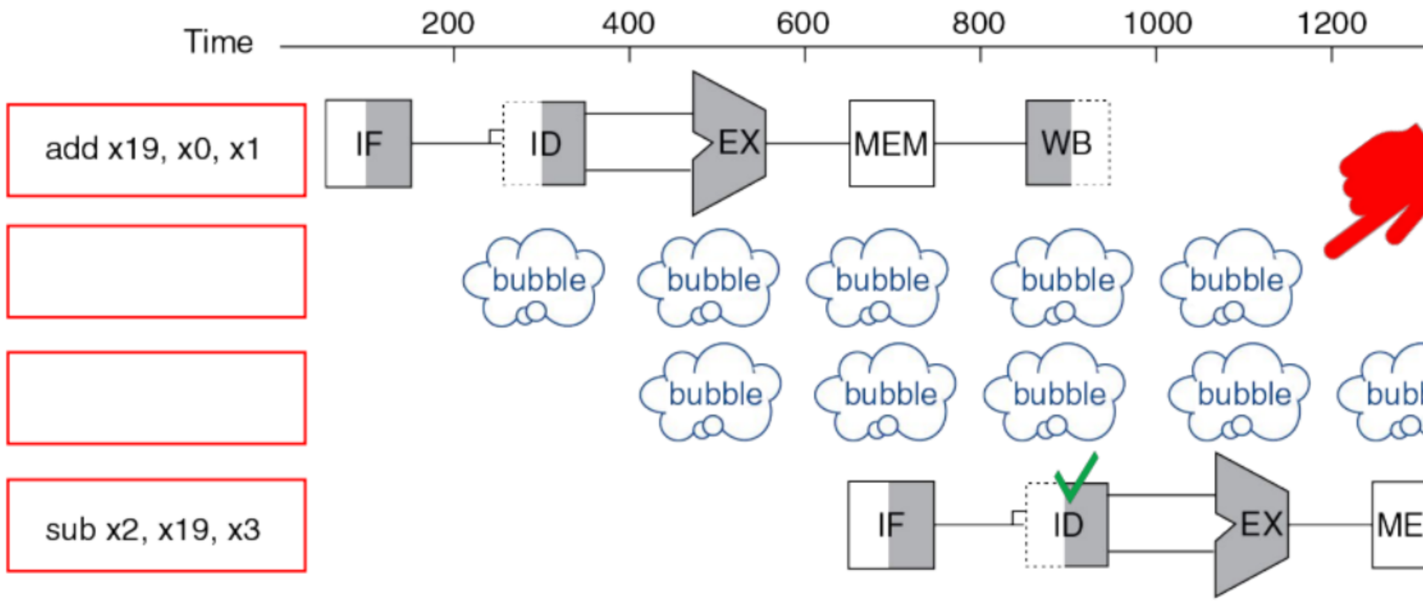


Hazards

Data Hazard - Read after write (RAW)

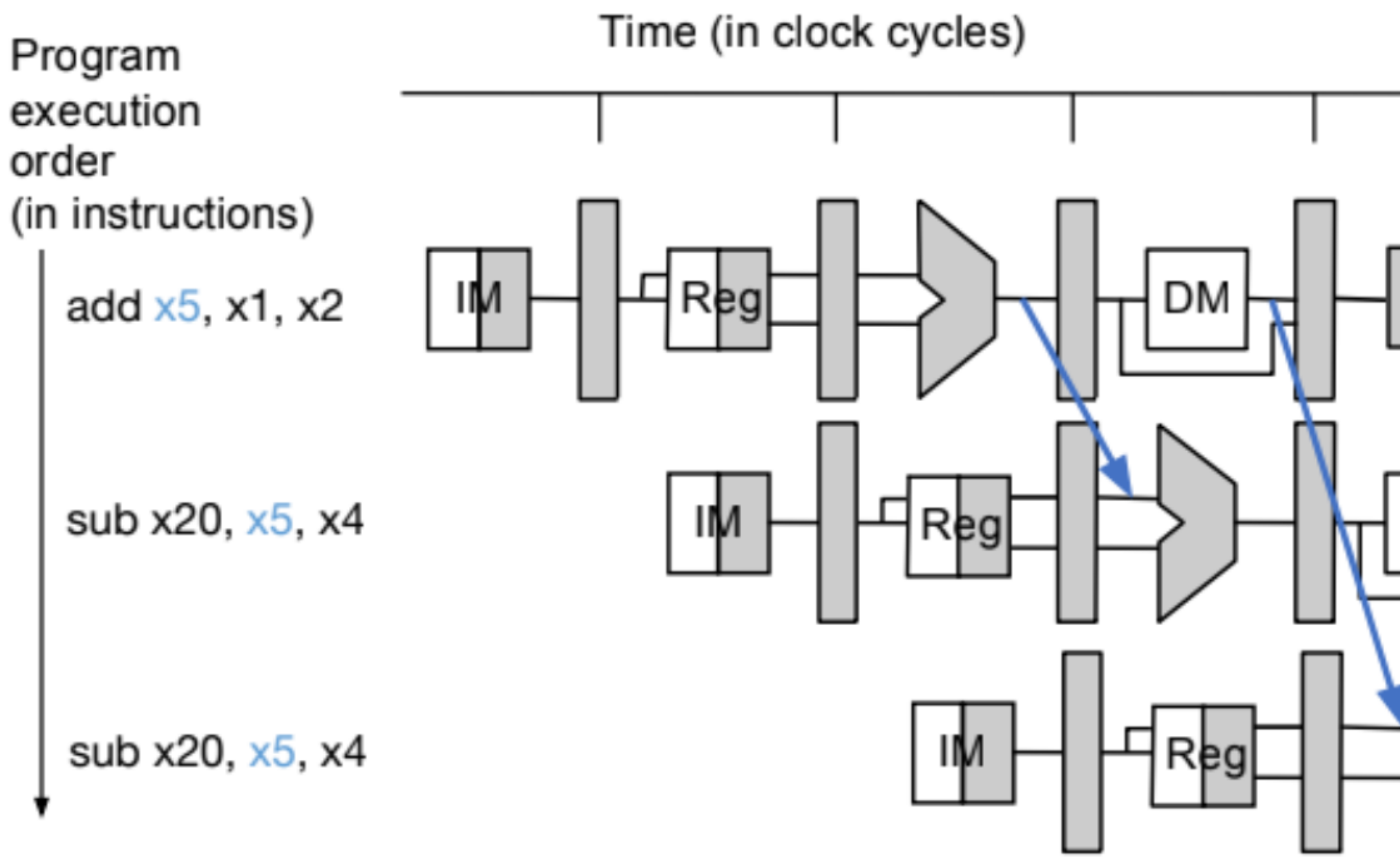
- Writing to a register (rd) and using it (rs1 or rs2) **before** the writing is finished (i.e., rd reaches to the WB stage.)
- To fix this, we either **stall** or **forward**.

Stalling:



- We add bubble instructions (In RISC-V, this is the NOP instruction, which does nothing).
- Alternatively, we can add other, unrelated instructions (e.g., instructions that need to be run that don't involve any of the same registers)

Forwarding:



- Instead of writing the resultant value to the register first, pass it to the ALU directly.
- Forwarding helps prevent stalls!
- See Week 3 Notes for how to detect when to forward.

Branches

- If we get hazards in a branch, we have two options:
  - Stalling
  - Speculation (not forwarding!)

  Stalling:

  - We must stall for 3 cycles!
    - When: End of ID stage
    - Where: End of EX or Beg. of Mem stage
    - Whether: End of Mem stage

  Speculation/Prediction

  - We predict which branch to take. To do this, we predict one as always taken and the other as never taken
  - Not taken is easier to check for, because:

    * We don't need branch addresses; not taken means the next address is PC + 4.
    * Most instructions are not branch so they are not taken too!
  - If we guess the branch incorrectly, we have to flush!
  - The penalty for flushing is 3 cycles, unless we can resolve the branch during the DE stage. This reduces the flush to 1 cycle.

Branch Prediction

How can we improve branch prediction?

- Branch Miss Penalty
  - Resolving branches sooner has reduced this penalty
- Miss rate?
  - Predicting always not-taken has only 30% accuracy.
  - What if we predict always taken?
    * We need nextPC at Fetch stage! (we need to be able to run the next instructions down that branch to save time).

Guessing Always Taken

- Idea: keep track of previous targets and use that to guess!
- If we see a branch instruction before, we know where it jumped. Remember this if we see the same branch again!
- How frequent is this? Is this always correct?

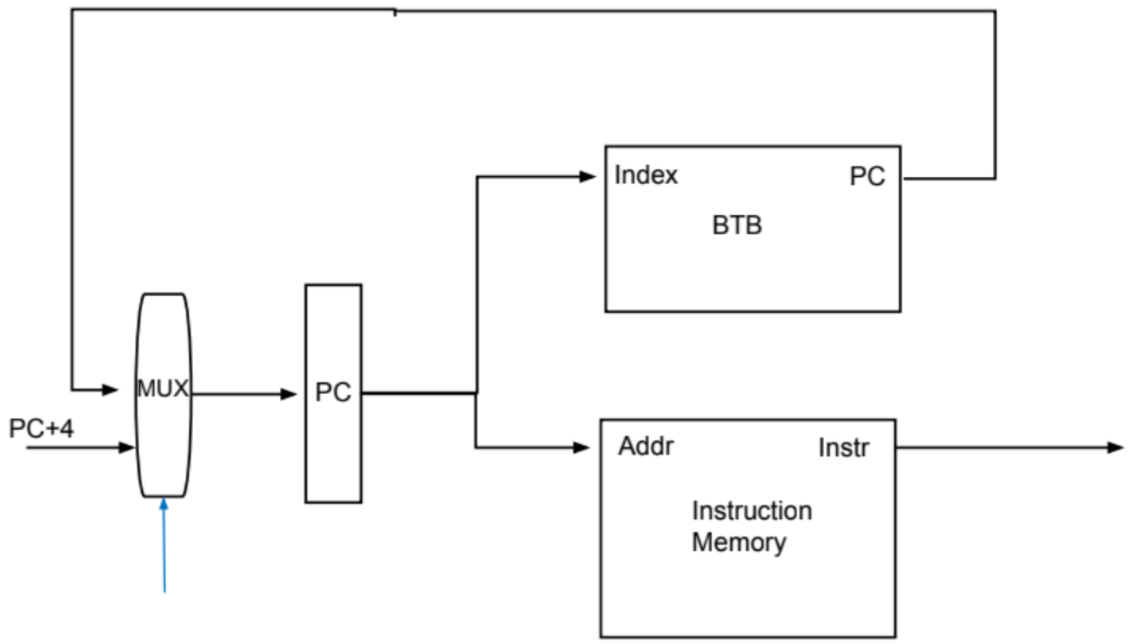
Branch Predictor - Where?

Branch Target Buffer (BTB)

- A table that stores *target addresses.*
- Entries are indexed by PC.
  - The size? Lower bits of PC (to utilize *locality*).

Algorithm for BTB

- For a new PC, record the target address in the table (using the PC as the index).
- Next time (a recurring PC), look up the table (i.e., by using the same index) and predict (i.e., use the stored value as the next PC).


PC+4
MUX
PC
Index
PC
BTB
Addr
Instr
Instruction Memory