Lecture 20: Branches, OOO

- Today's topics:
  - Branch prediction
  - Out-of-order execution
  - (Also see class notes on pipelining, hazards, etc.)

Pipelining Example (Recap)

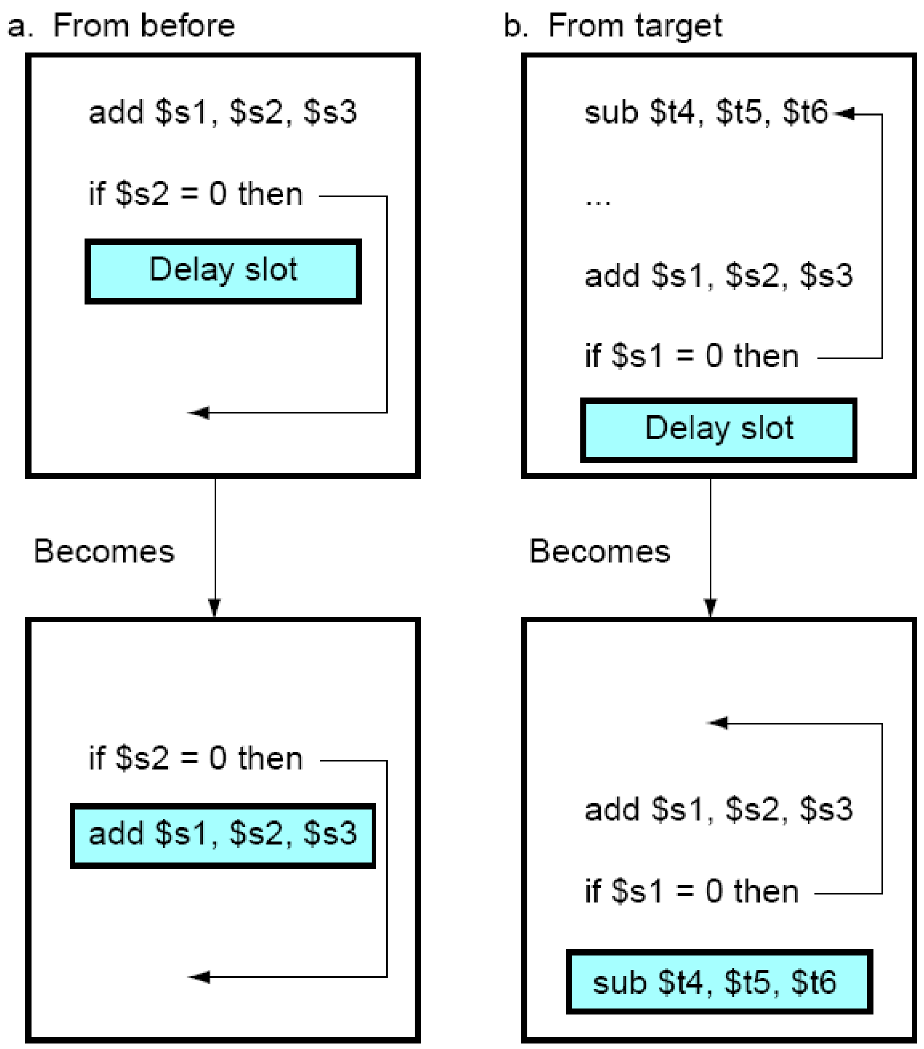
- Unpipelined design: the entire circuit takes 10ns to finish
  Cycle time = 10ns; Clock speed = 1/10ns = 100 MHz
  CPI = 1 (assuming no stalls)
  Throughput in instructions per second =
  #cycles in a second x instructions-per-cycle =
  100 M x 1 = 100 M instrs per second = 0.1 BIPS (billion instrs per sec)

- 5-stage pipeline: under ideal conditions, each stage takes 2ns
  Cycle time = 2ns; Clock speed = 1/2ns = 500 MHz (5x higher)
  CPI = 1 (continuing to assume no stalls)
  Throughput = # cycles in a second x instrs-per-cycle
  = 500 M x 1 = 500 MIPS = 0.5 BIPS
  Under ideal conditions, a 5-stage pipeline gives a 5x speedup.

Control Hazards

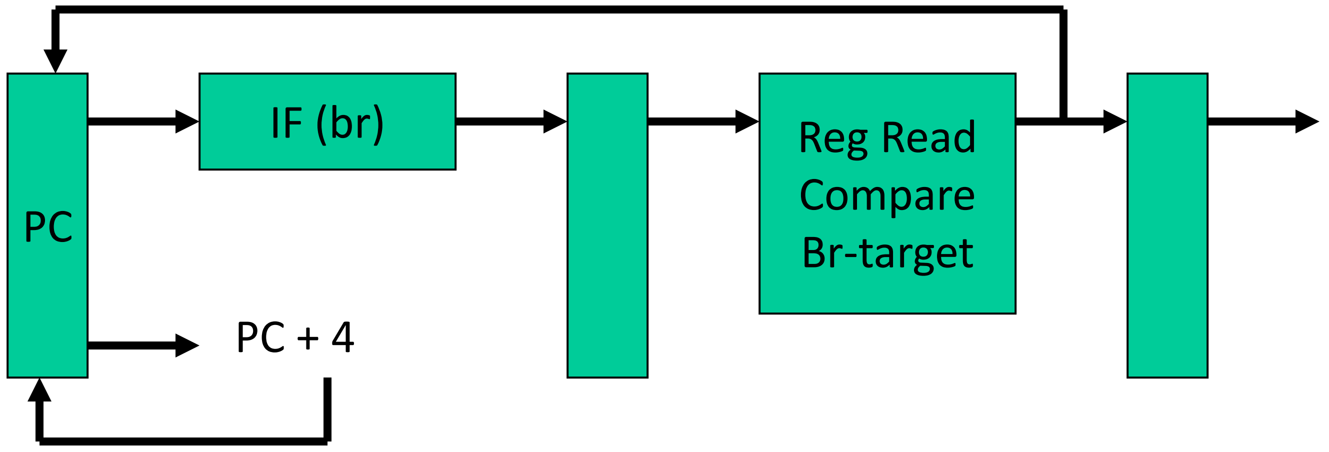
- Simple techniques to handle control hazard stalls:
  - for every branch, introduce a stall cycle (note: every 6th instruction is a branch!)
  - assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instruction
  - fetch the next instruction (branch delay slot) and execute it anyway – if the instruction turns out to be on the correct path, useful work was done – if the instruction turns out to be on the wrong path, hopefully program state is not lost
  - make a smarter guess and fetch instructions from the expected target

Branch Delay Slots

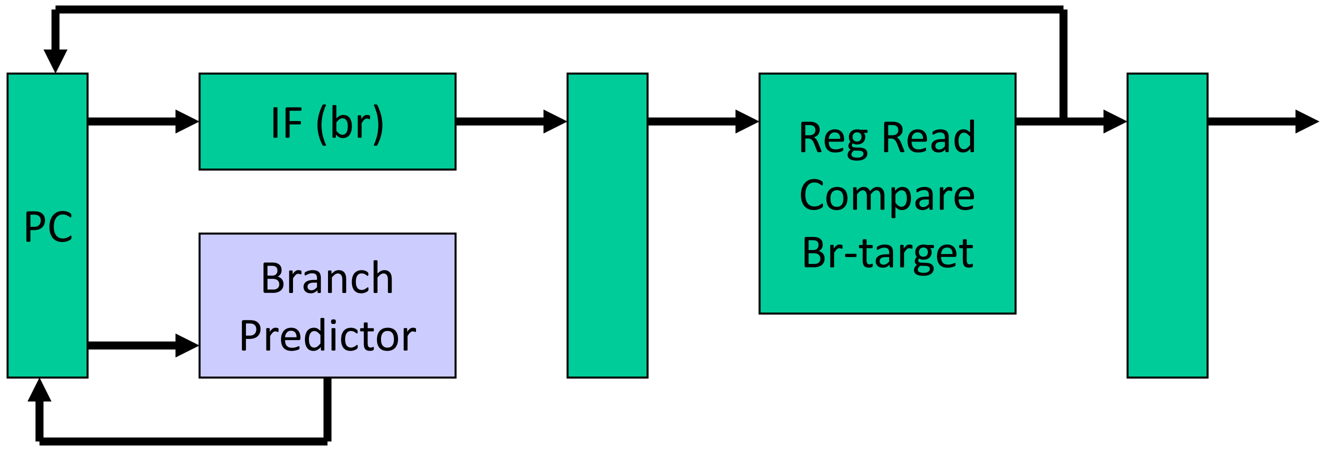


Source: H&P textbook

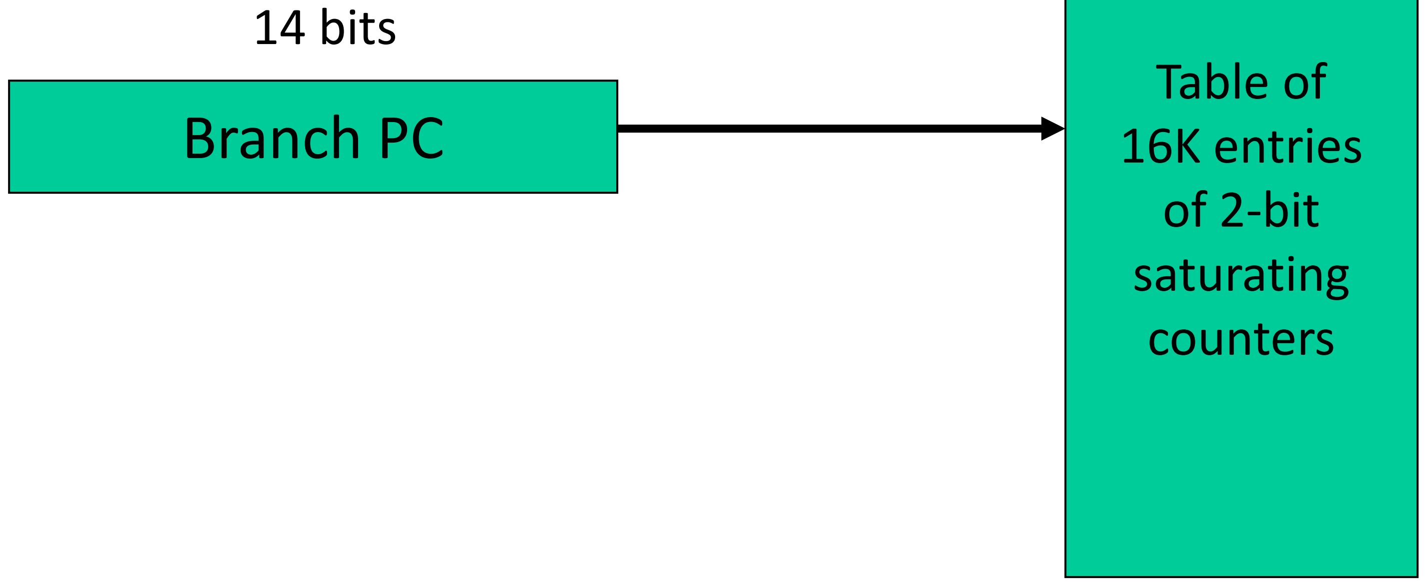
Pipeline without Branch Predictor



Pipeline with Branch Predictor



Bimodal Predictor



2-Bit Prediction

- For each branch, maintain a 2-bit saturating counter:
  if the branch is taken: counter = min(3,counter+1)
  if the branch is not taken: counter = max(0,counter-1)
  … sound familiar?

- If (counter >= 2), predict taken, else predict not taken

- The counter attempts to capture the common case for each branch

Indexing functions
Multiple branch predictors
History, trade-offs

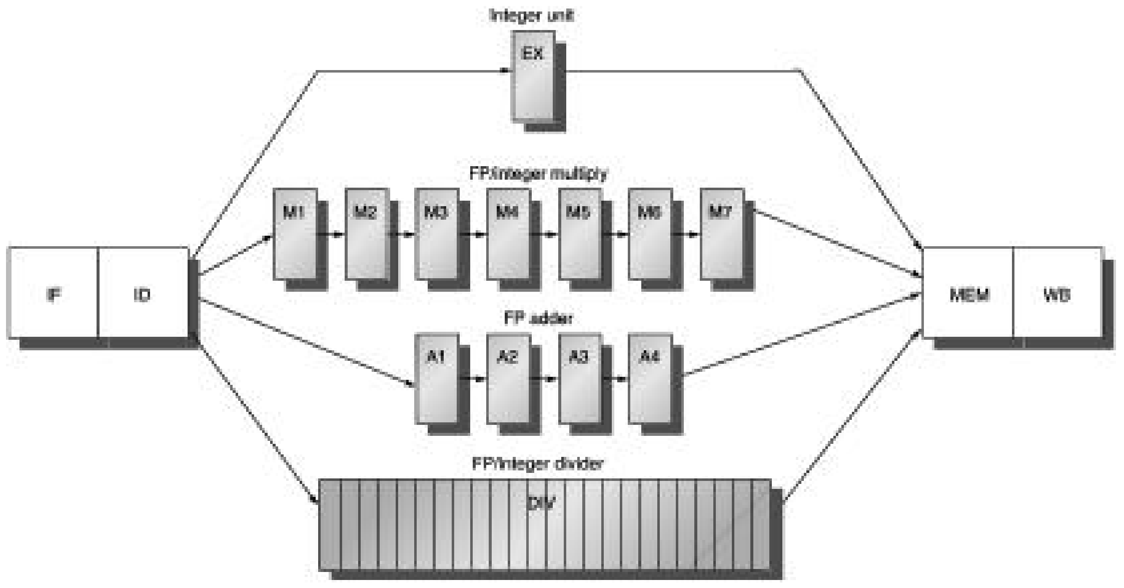
Slowdowns from Stalls

- Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions) → speedup = increase in clock speed = num pipeline stages

- With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes

- Total cycles = number of instructions + stall cycles

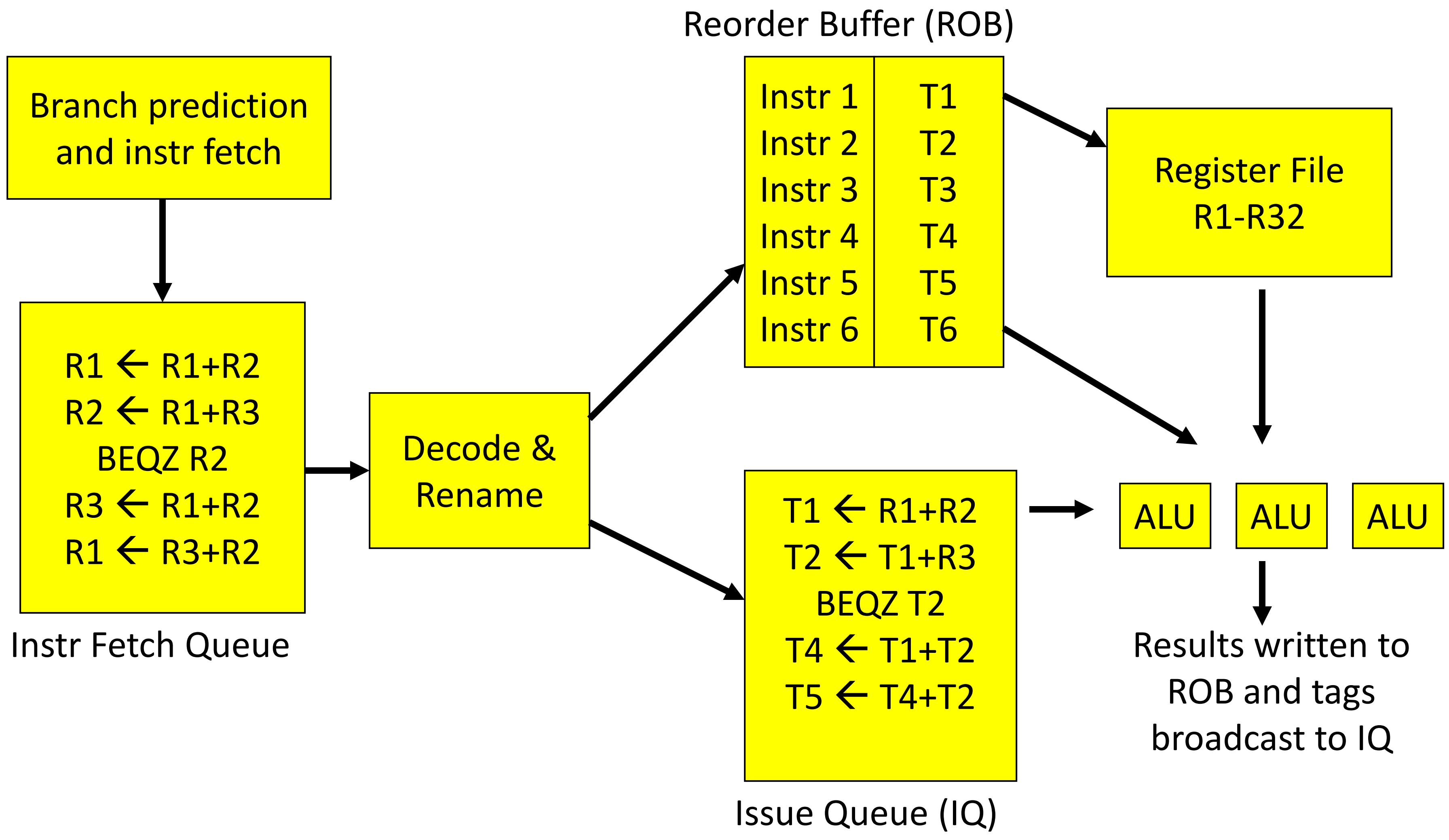
Multicycle Instructions



- Multiple parallel pipelines – each pipeline can have a different number of stages

- Instructions can now complete out of order – must make sure that writes to a register happen in the correct order

An Out-of-Order Processor Implementation



Example Code

Completion times	with in-order	with ooo
ADD R1, R2, R3	5	5
ADD R4, R1, R2	6	6
LW R5, 8(R4)	7	7
ADD R7, R6, R5	9	9
ADD R8, R7, R5	10	10
LW R9, 16(R4)	11	7
ADD R10, R6, R9	13	9
ADD R11, R10, R9	14	10