Lecture 19: Branches

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

PoP/PoC Summary

Without bypassing:

PoP is typically whenever the register file write is completed

PoC is typically at the start of register file read

With bypassing:

PoP is when the value to be written to the register is available

- For an Add, right after the ALU stage
- For a Load, right after the DM stage
- For an FP-Add, right after all the FP-Add stages have finished

PoC is right before one of the compute units needs its input

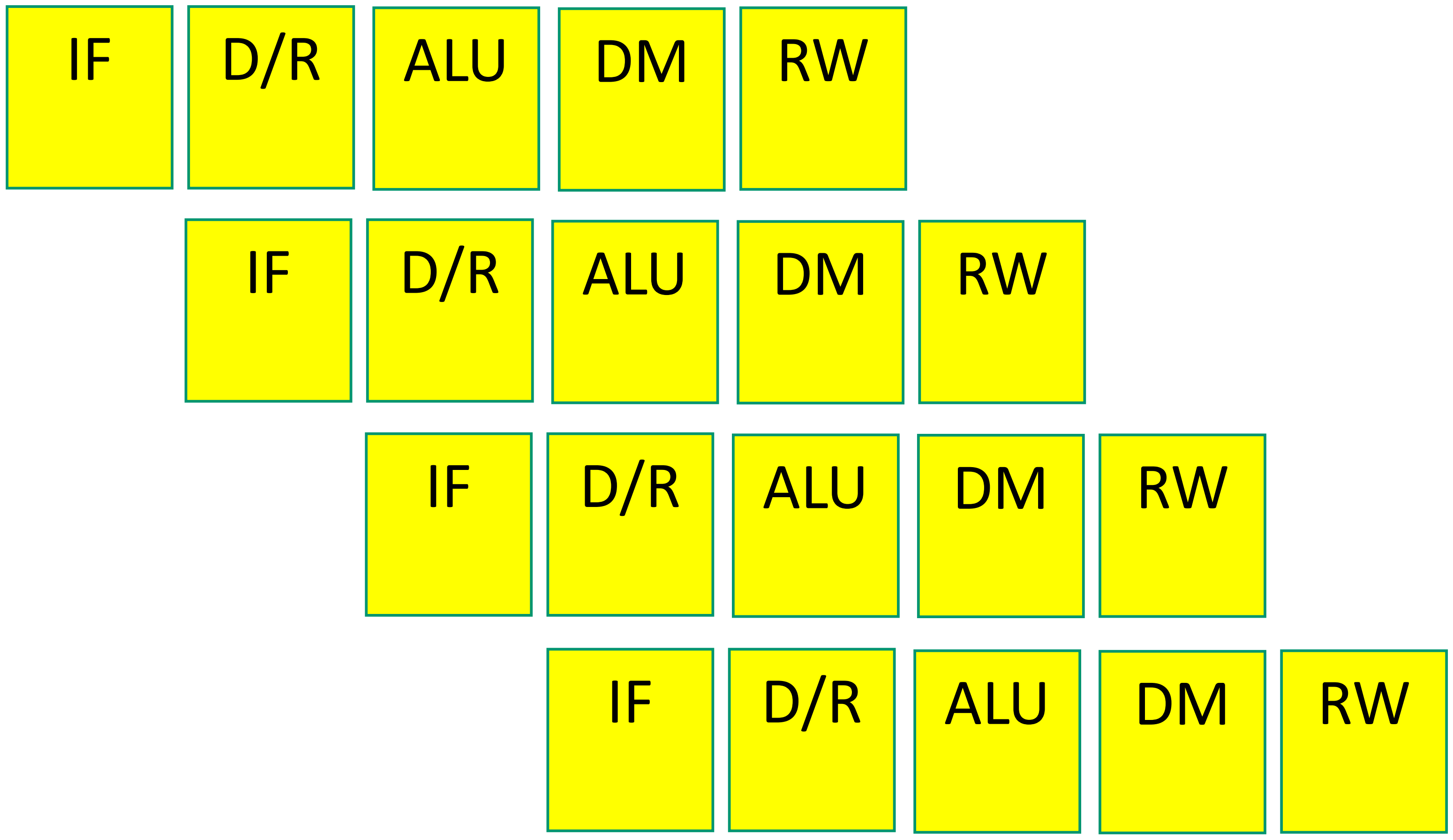
- For an Add, right before the ALU stage
- For a Load, right before the ALU stage
- For a store, one operand is needed right before ALU stage
  one operand is needed right before DM stage

Pipeline Depth

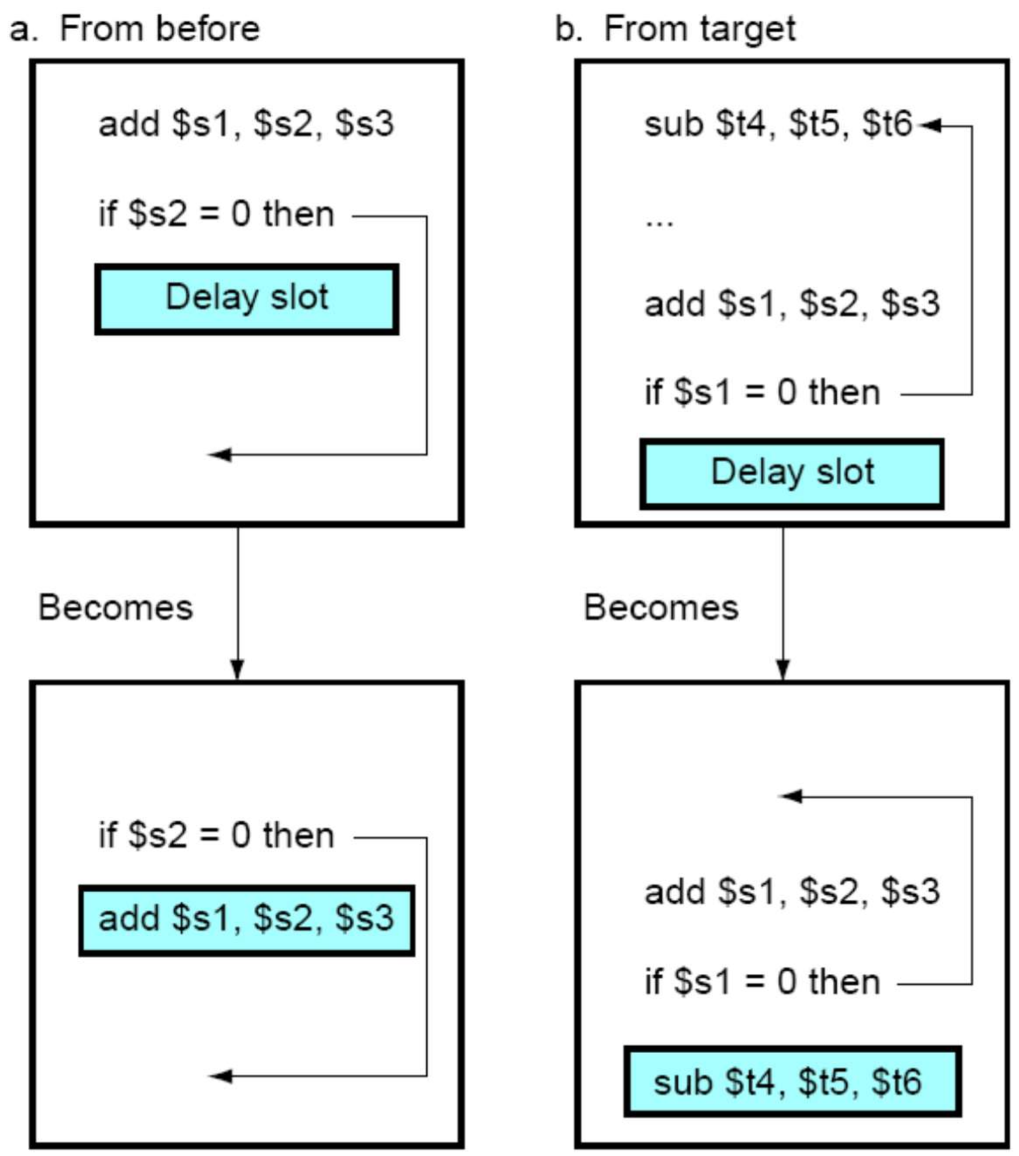
Control Hazards

- 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

Control Hazards

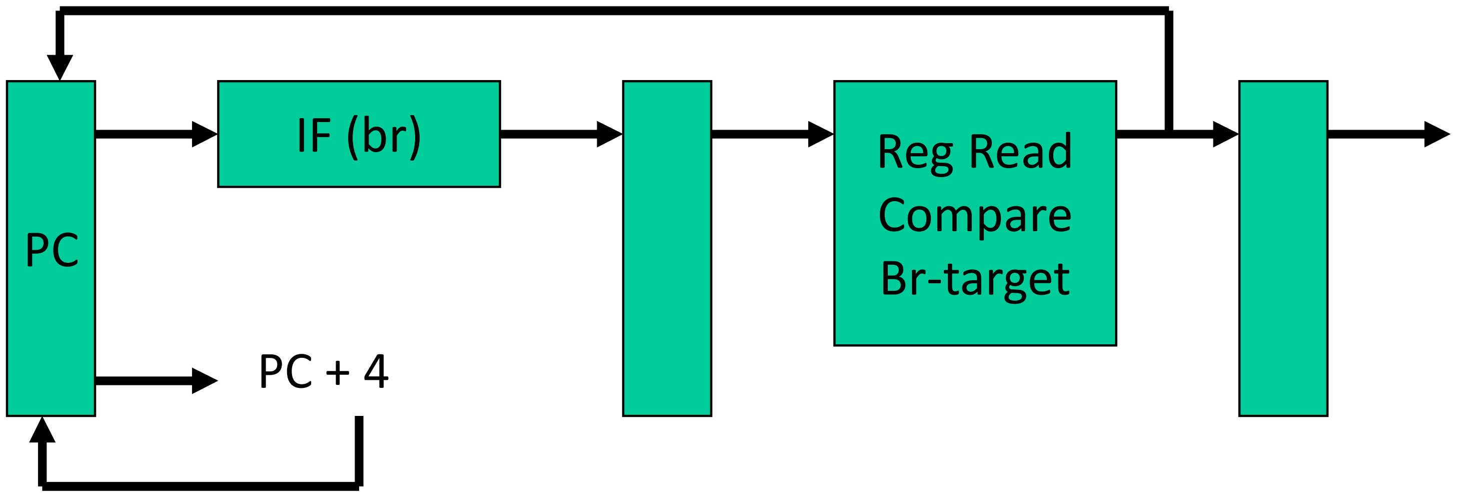


Branch Delay Slots

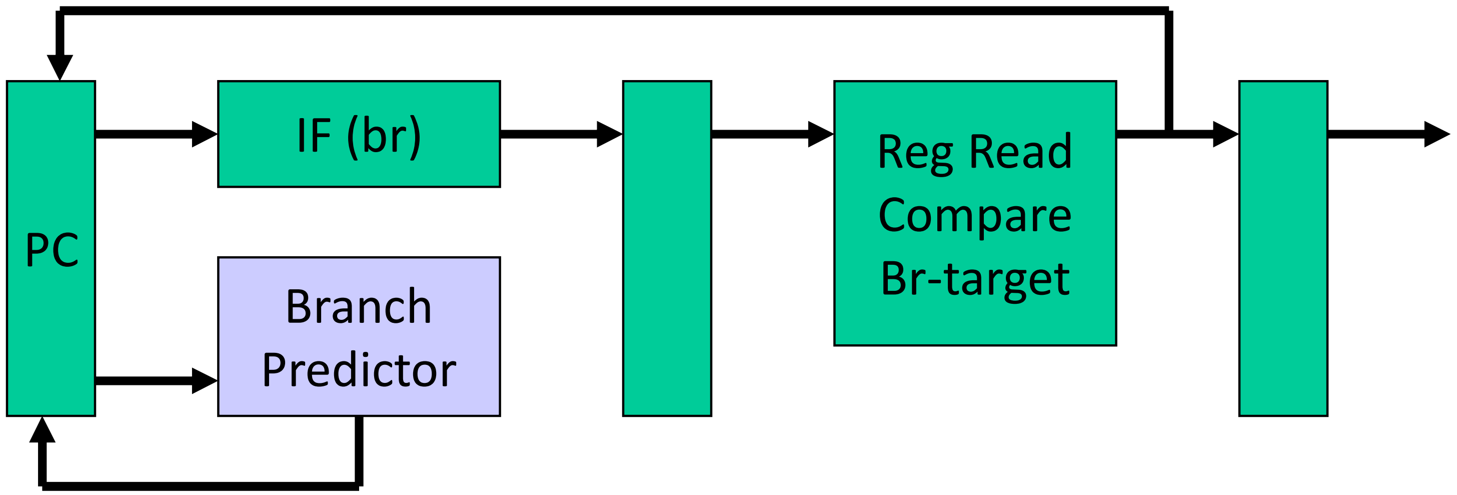


Source: H&P textbook

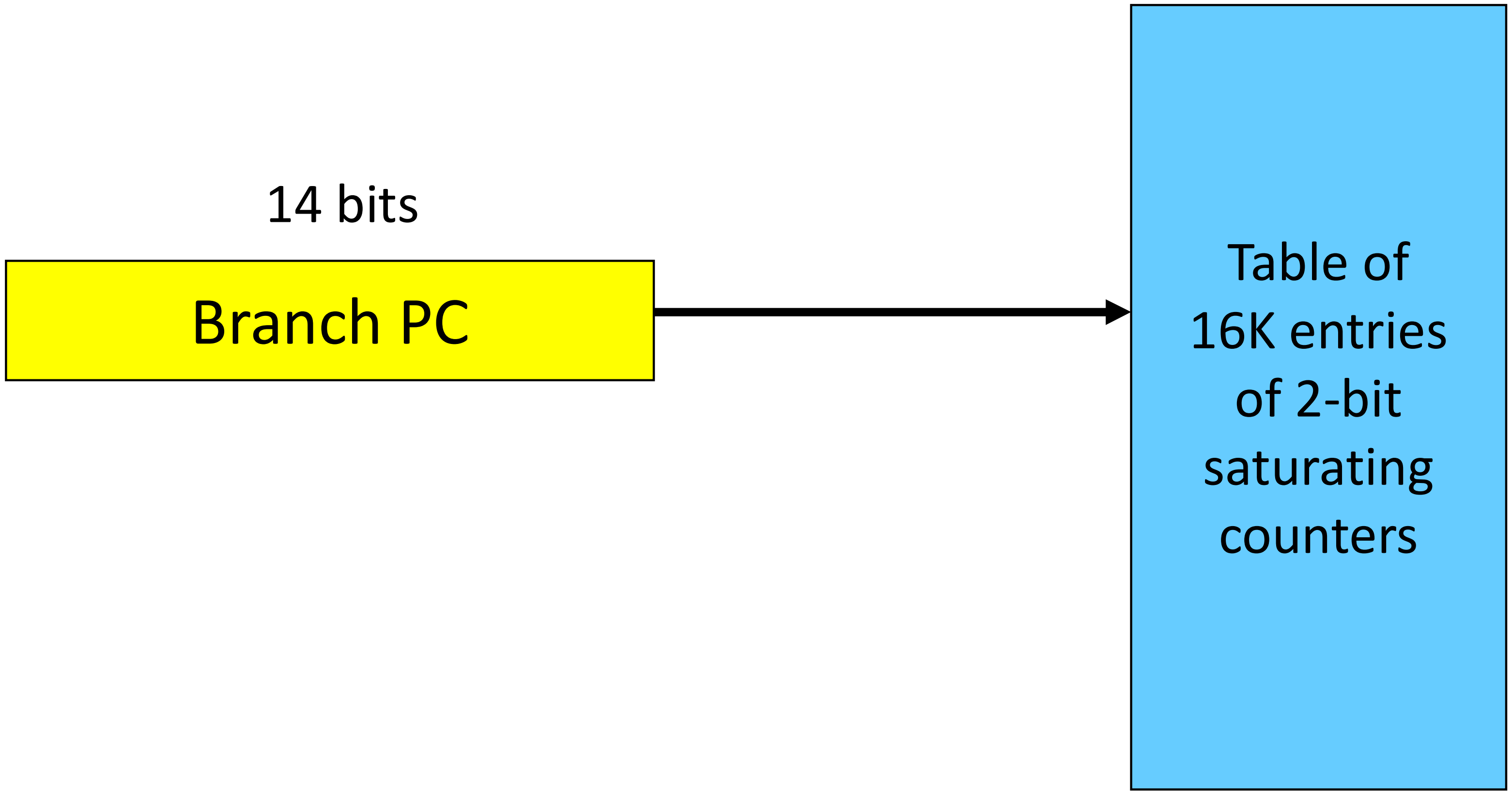
Pipeline without Branch Predictor



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

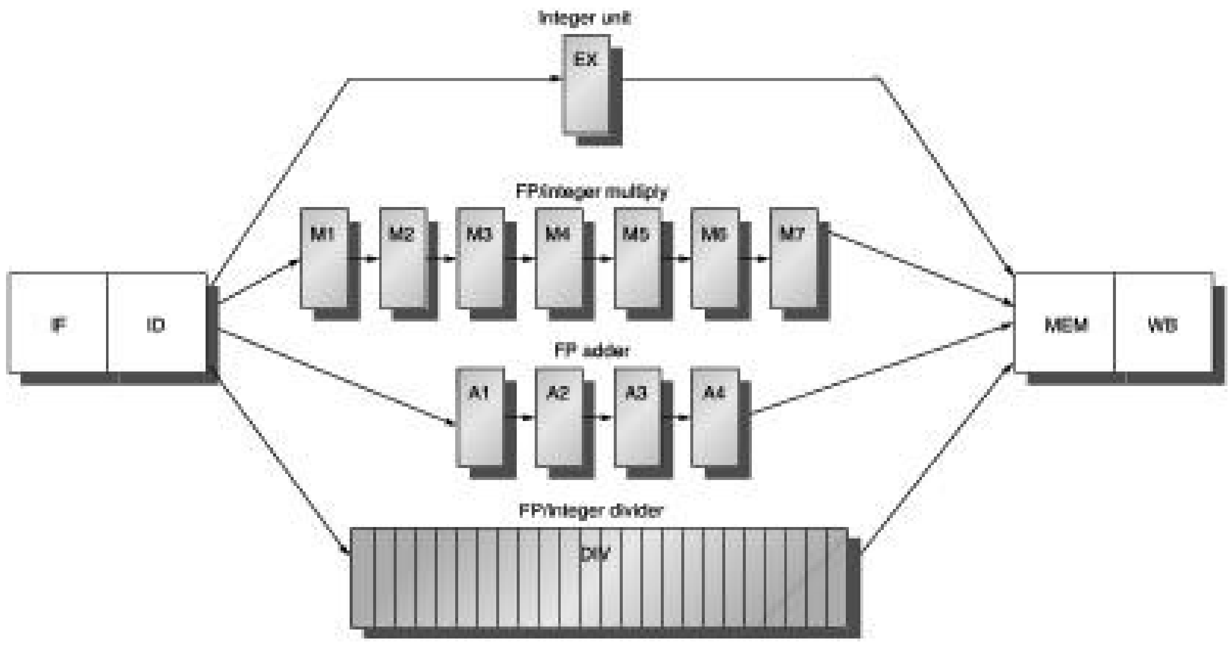
- The counter attempts to capture the common case for each branch

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