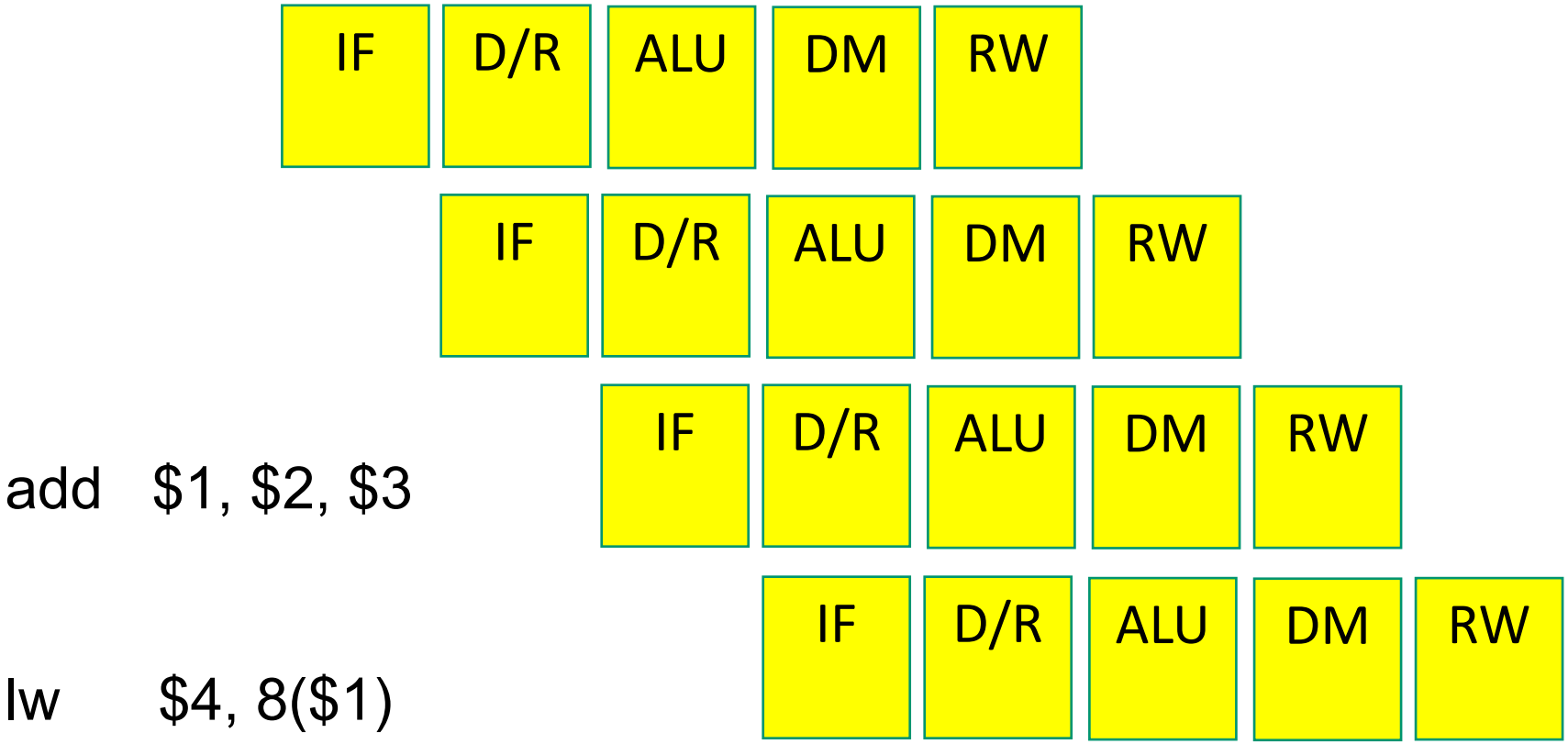
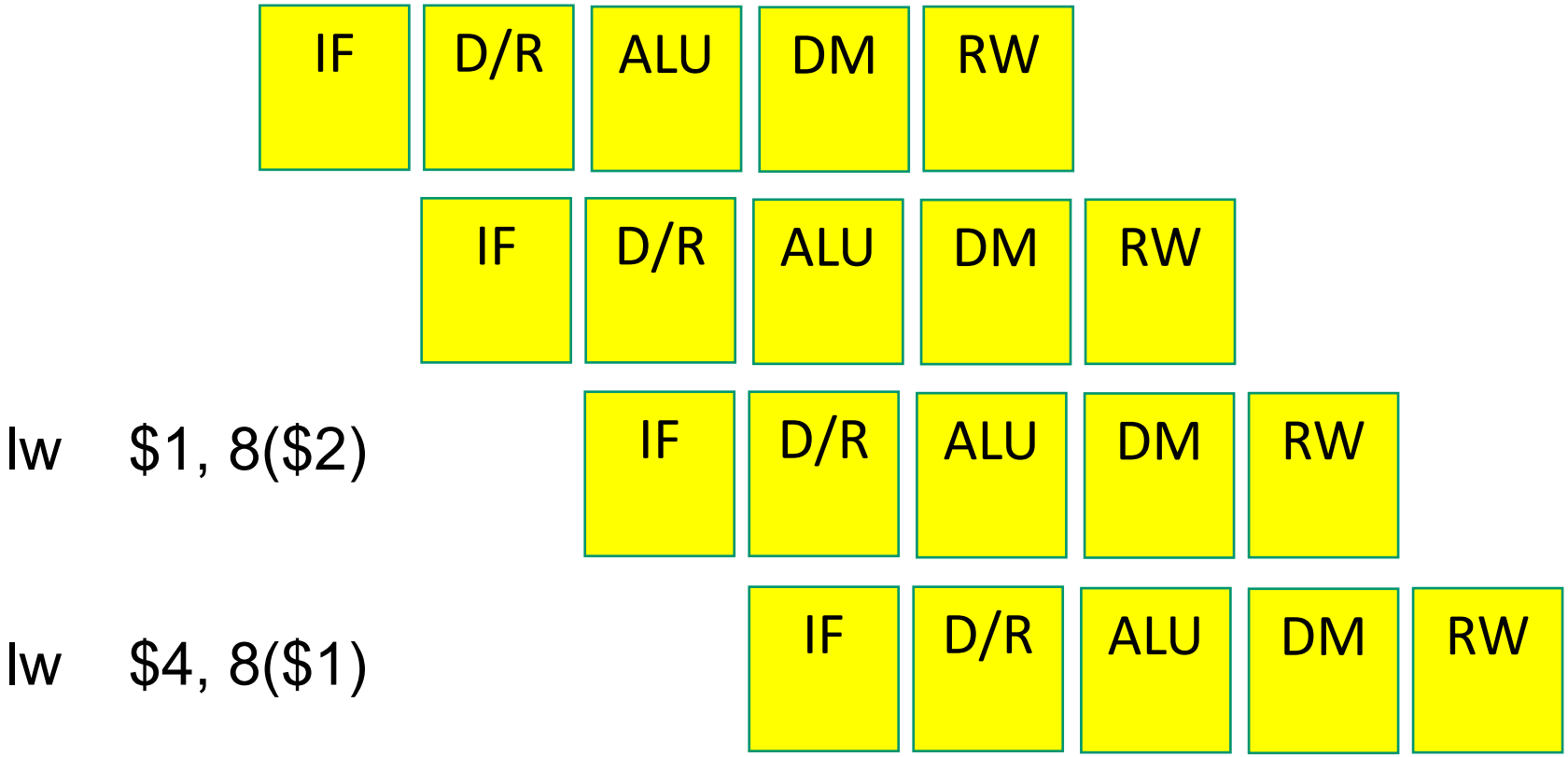
Lecture 19: Pipelining

- Today's topics:
  - Hazards and instruction scheduling
  - Branch prediction
  - Out-of-order execution

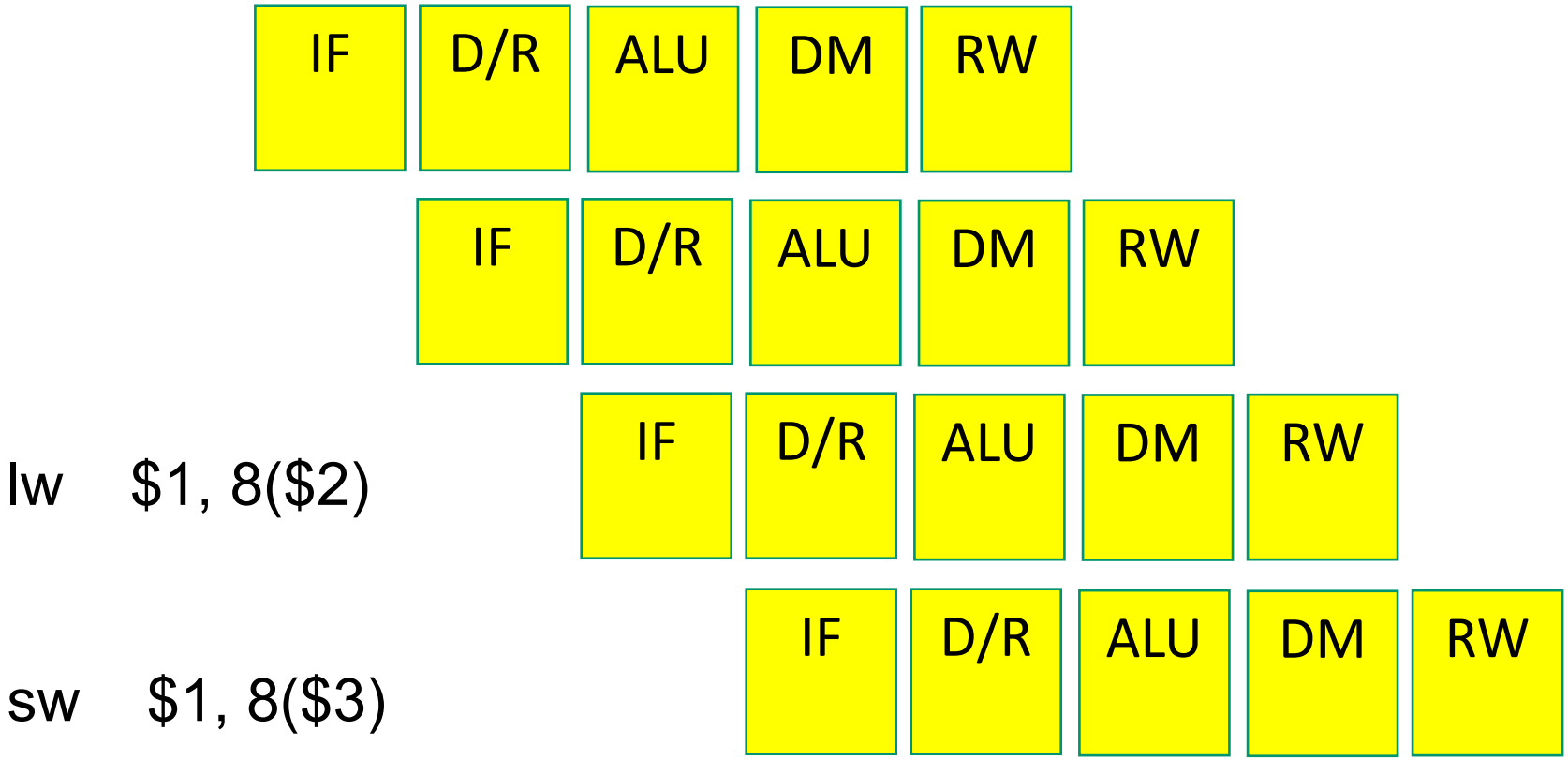
Problem 1


IF
D/R
ALU
DM
RW
IF
D/R
ALU
DM
RW
IF
D/R
ALU
DM
RW
IF
D/R
ALU
DM
RW
add $1, $2, $3
lw $4, 8($1)

Problem 2

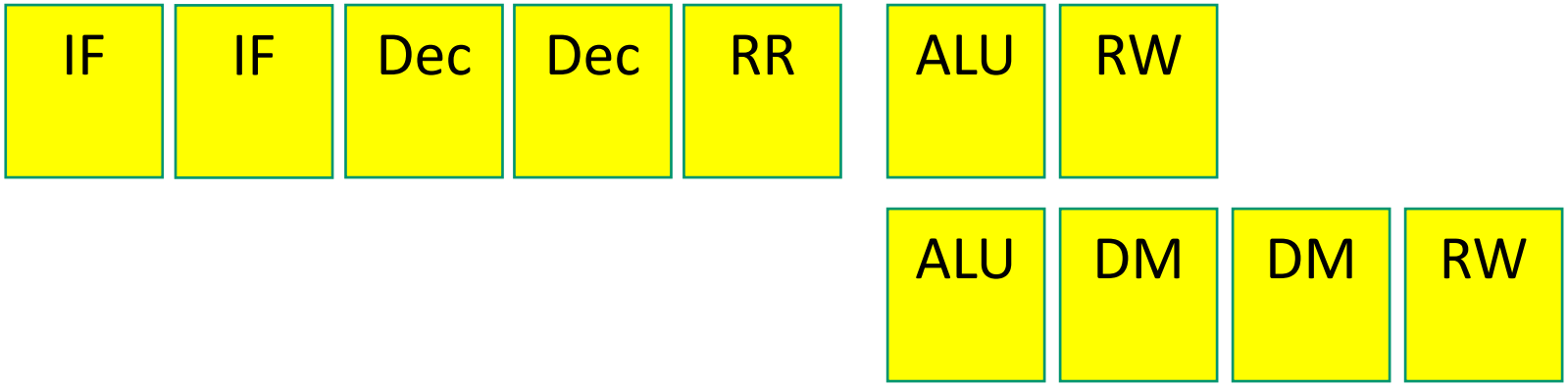

IF
D/R
ALU
DM
RW
IF
D/R
ALU
DM
RW
lw $1, 8($2)
IF
D/R
ALU
DM
RW
lw $4, 8($1)
IF
D/R
ALU
DM
RW

Problem 3


IF
D/R
ALU
DM
RW
IF
D/R
ALU
DM
RW
IF
D/R
ALU
DM
RW
lw $1, 8($2)
IF
D/R
ALU
DM
RW
sw $1, 8($3)

Problem 4

A 7 or 9 stage pipeline



lw $1, 8($2)

add $4, $1, $3

Problem 4

Without bypassing: 4 stalls

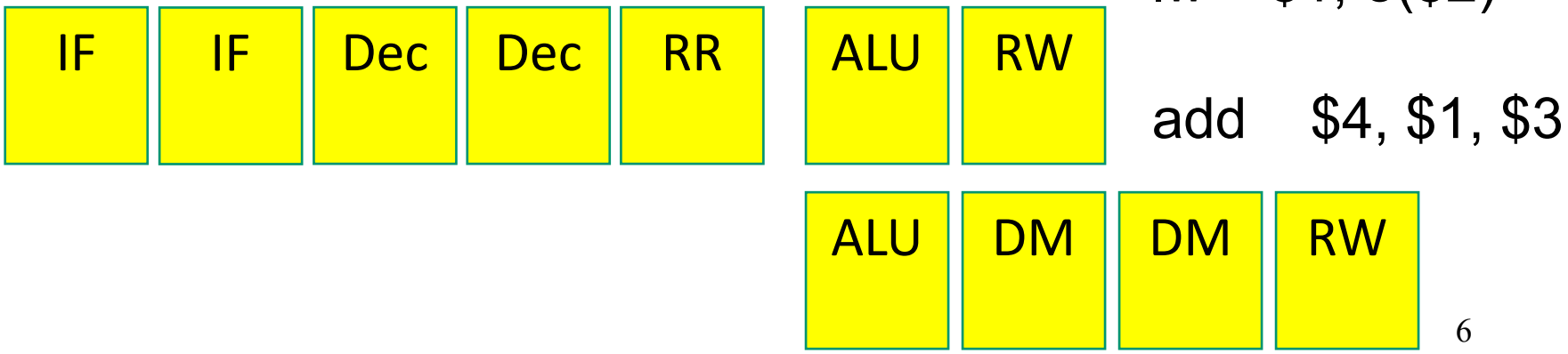
IF:IF:DE:DE:RR:AL:DM:DM:RW

IF: IF :DE:DE:DE:DE:DE :DE:RR:AL:RW

With bypassing: 2 stalls

IF:IF:DE:DE:RR:AL:DM:DM:RW

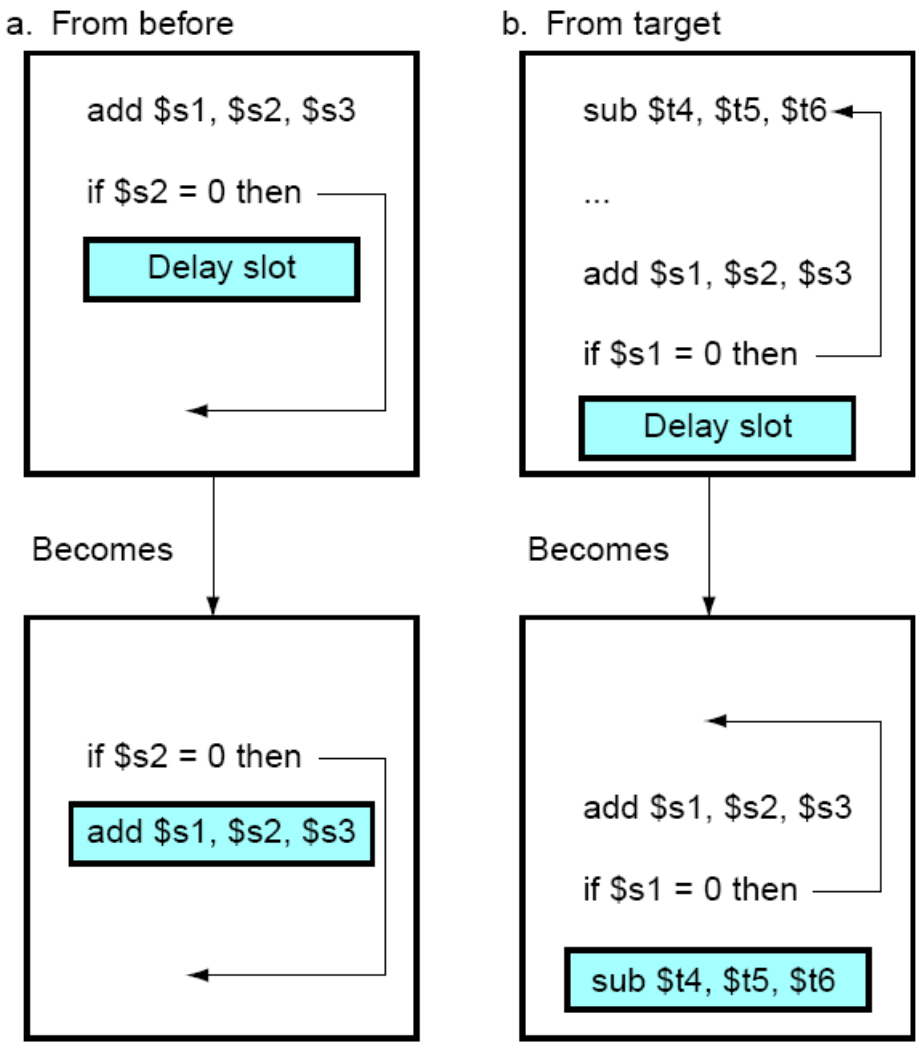
IF: IF :DE:DE:DE:DE:RR :AL:RW



Control Hazards

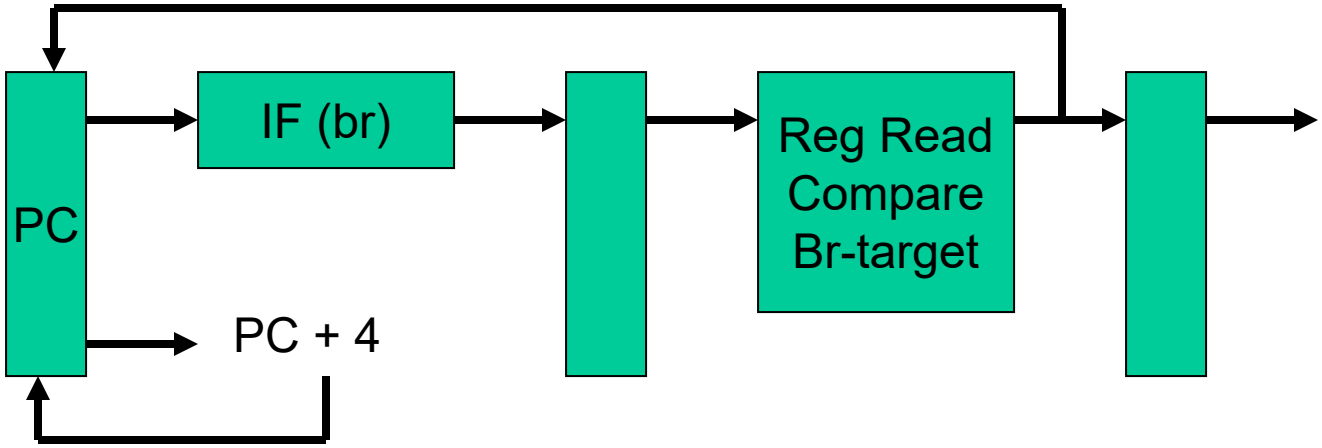
- Simple techniques to handle control hazard stalls:
  - for every branch, introduce a stall cycle (note: every 6th instruction is a branch!)
  - assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instruction
  - 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

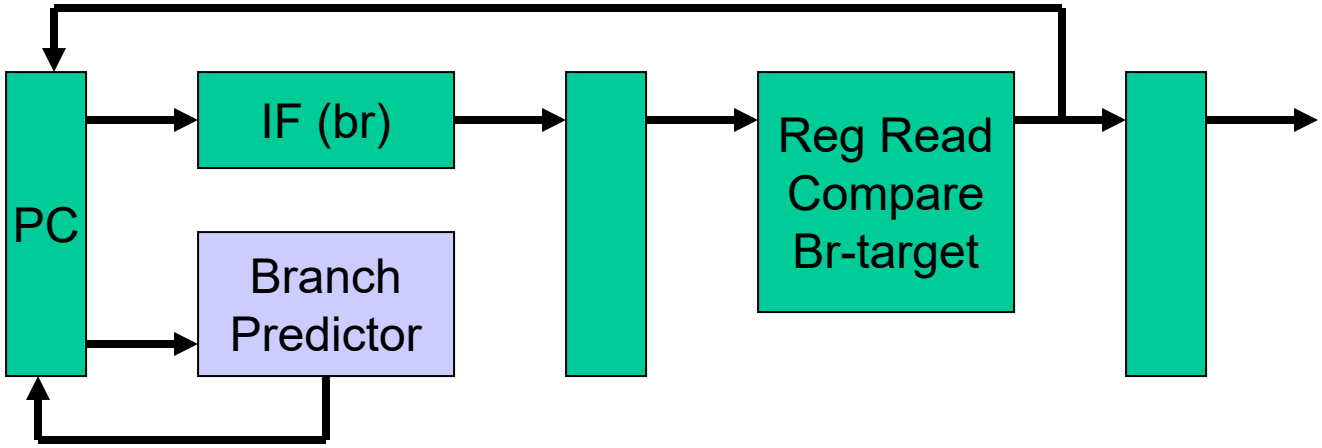


Source: H&P textbook

Pipeline without Branch Predictor



Pipeline with Branch Predictor



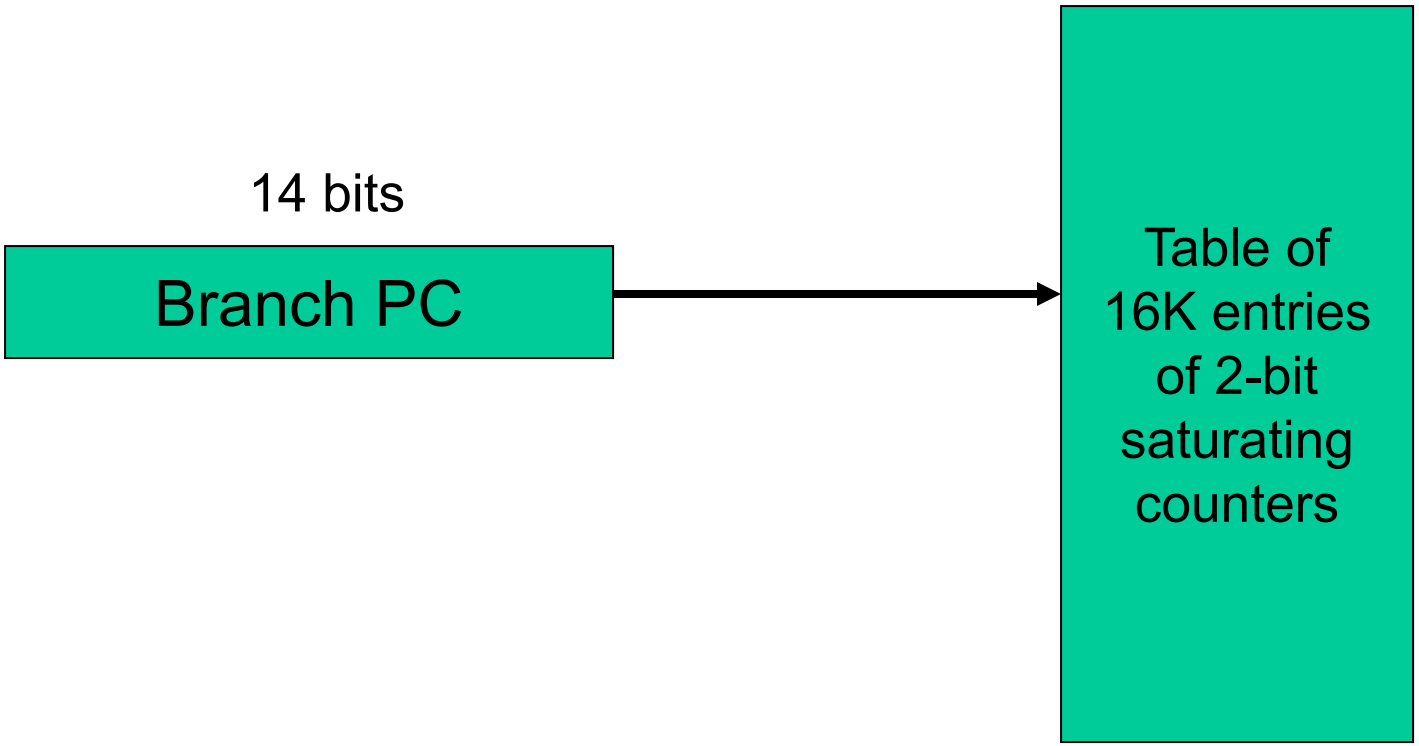
2-Bit Prediction

- 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

Bimodal Predictor



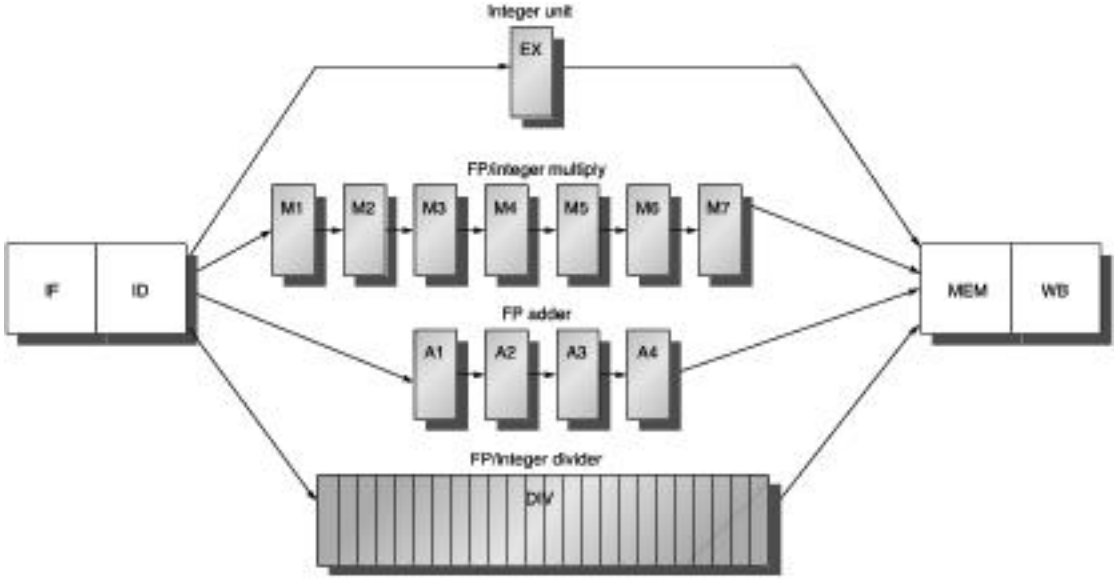
Slowdowns from Stalls

- 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