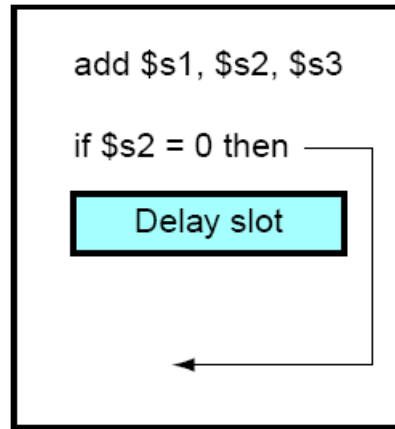


Control Hazards

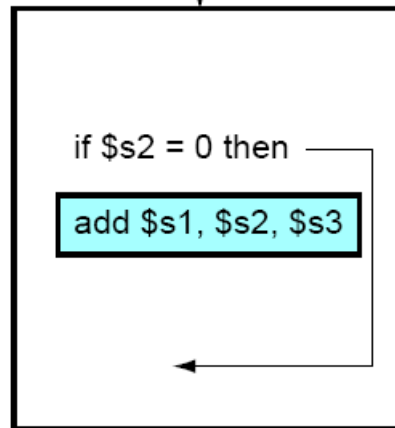
- Simple techniques to handle control hazard stalls:
 - for every branch, introduce a stall cycle (note: every 6th instruction is a branch!)
 - assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instruction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

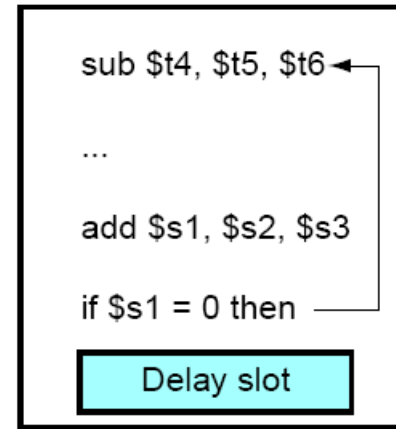
a. From before



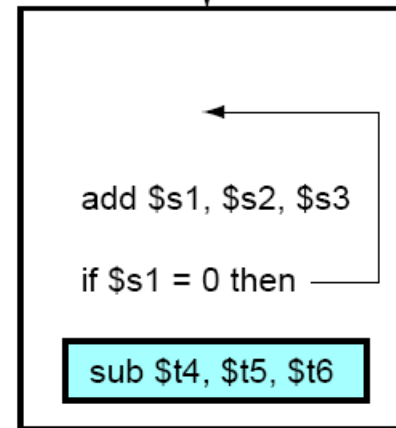
Becomes



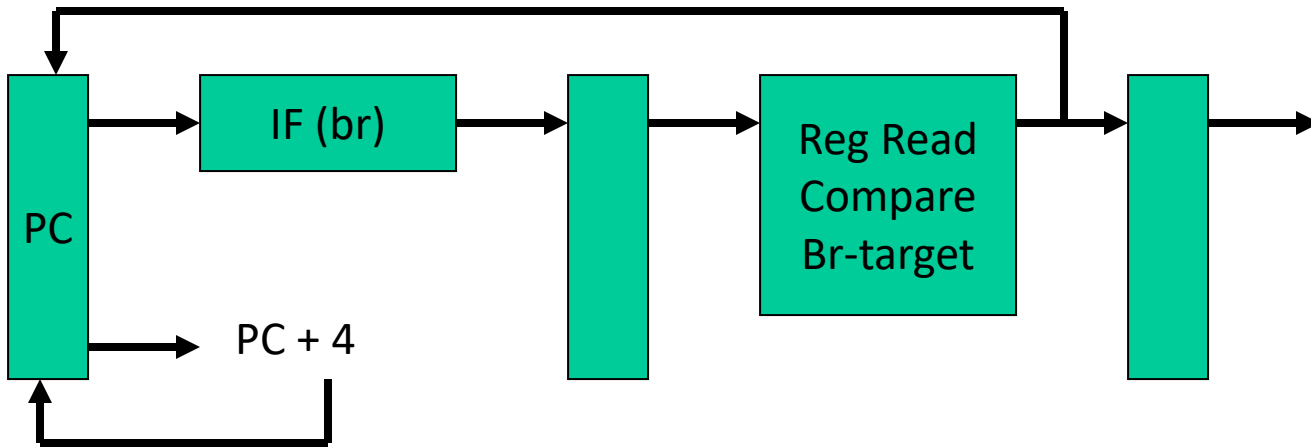
b. From target



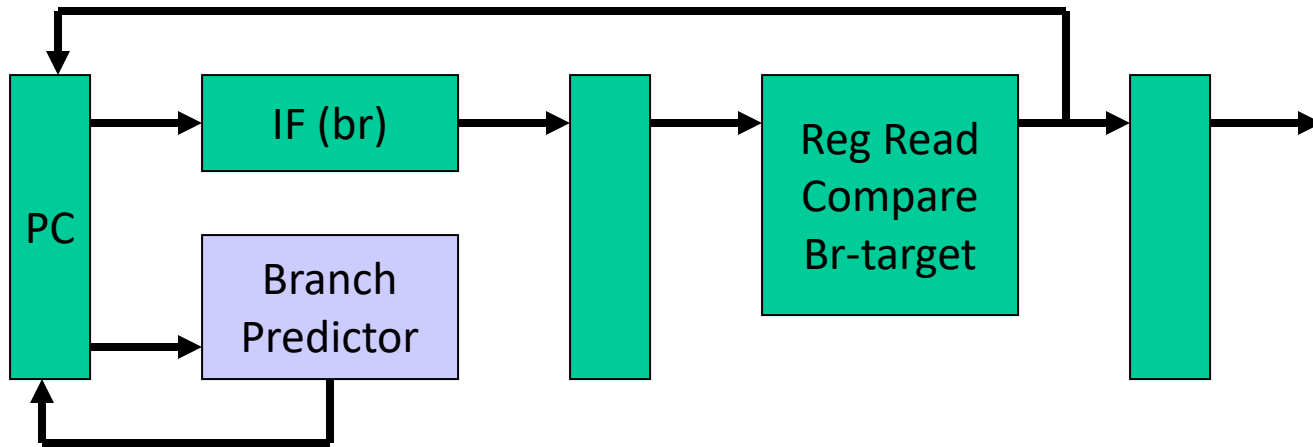
Becomes



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: $\text{counter} = \min(3, \text{counter} + 1)$
if the branch is not taken: $\text{counter} = \max(0, \text{counter} - 1)$
... sound familiar?
- If ($\text{counter} \geq 2$), predict taken, else predict not taken
- The counter attempts to capture the common case for each branch

Slowdowns from Stalls

- Perfect pipelining with no hazards \rightarrow an instruction completes every cycle (total cycles \sim num instructions)
 \rightarrow 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