Advanced Computer Architecture

Pipelining Hazards

Computer Science

Pipelining Hazards

- Structural hazards: different instructions in different stages (or the same stage) conflicting for the same resource

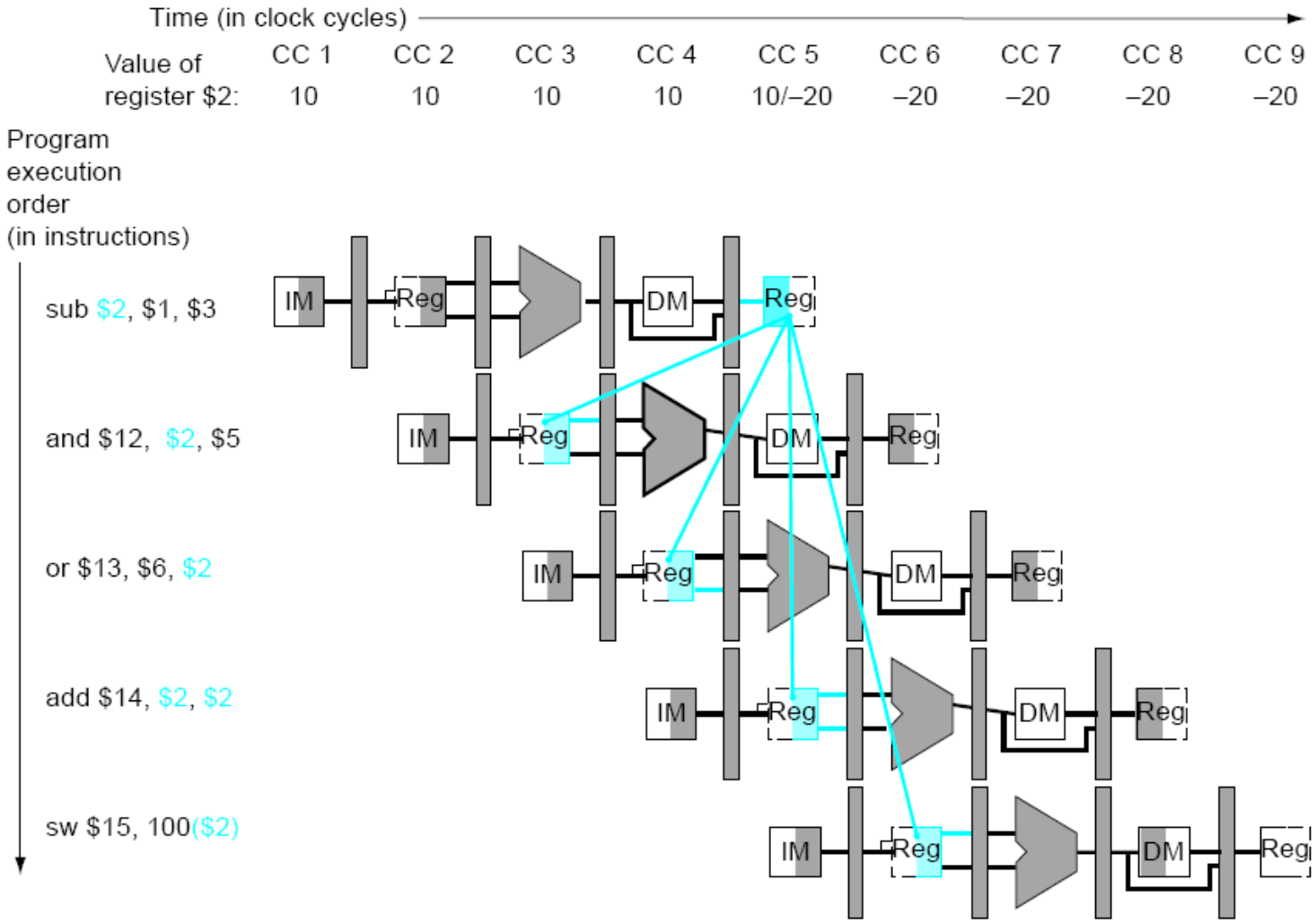
- Data hazards: an instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction

- Control hazard: fetch cannot continue because it does not know the outcome of an earlier branch – special case of a data hazard – separate category because they are treated in different ways

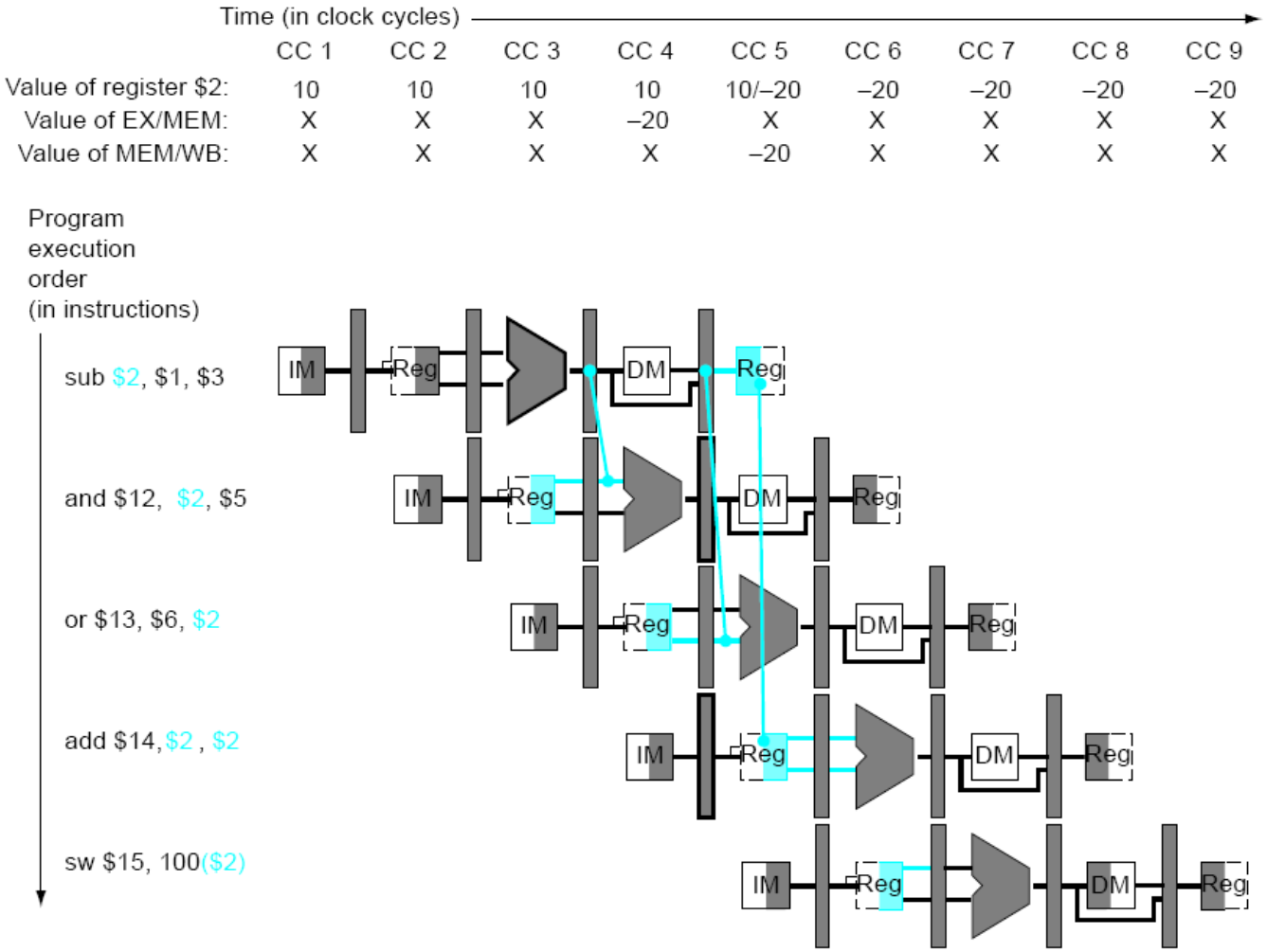
Structural Hazards

- Example: a unified instruction and data cache → stage 4 (MEM) and stage 1 (IF) can never coincide
- The later instruction and all its successors are delayed until a cycle is found when the resource is free → these are pipeline bubbles
- Structural hazards are easy to eliminate – increase the number of resources (for example, implement a separate instruction and data cache)

Data Hazards

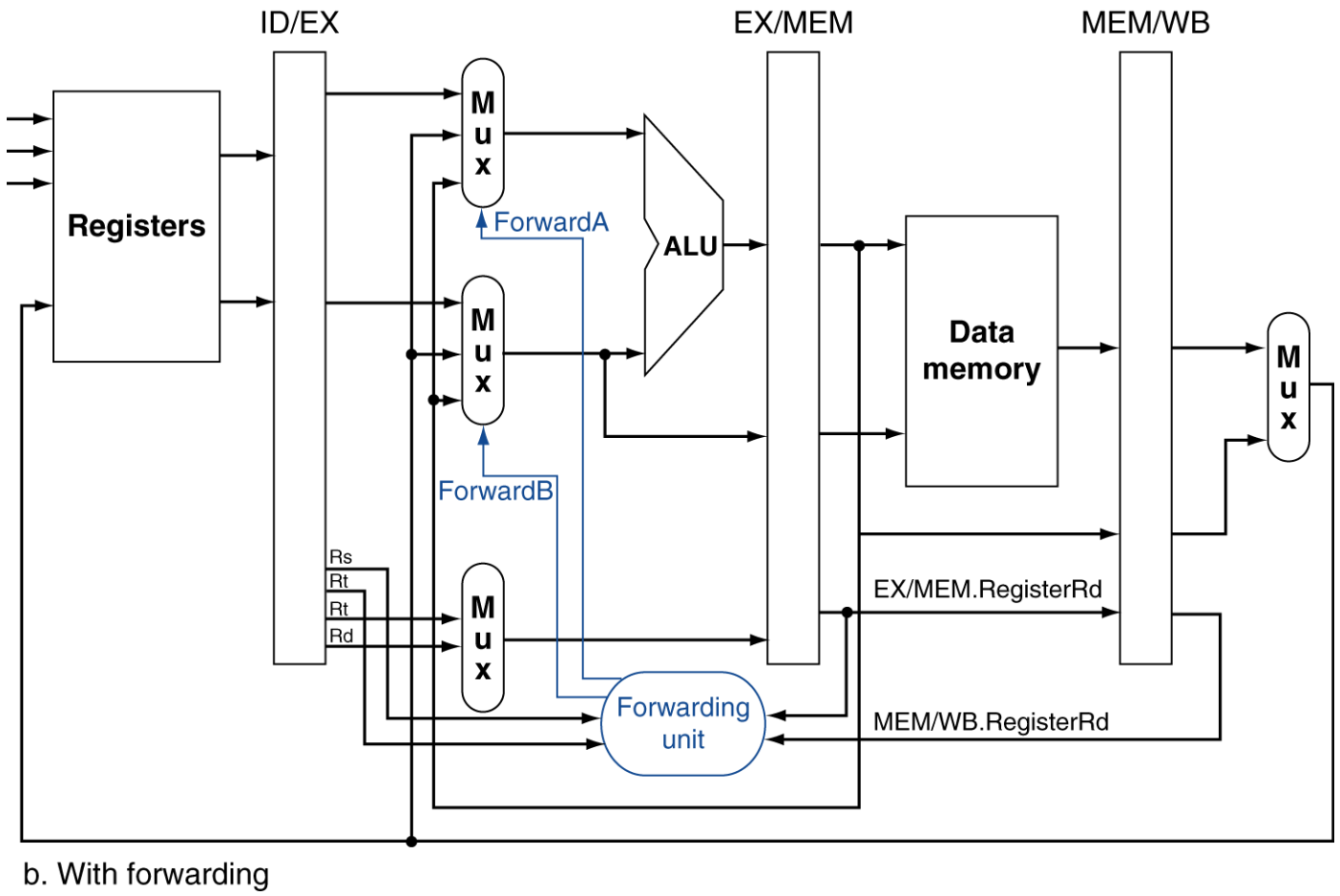


Forwarding



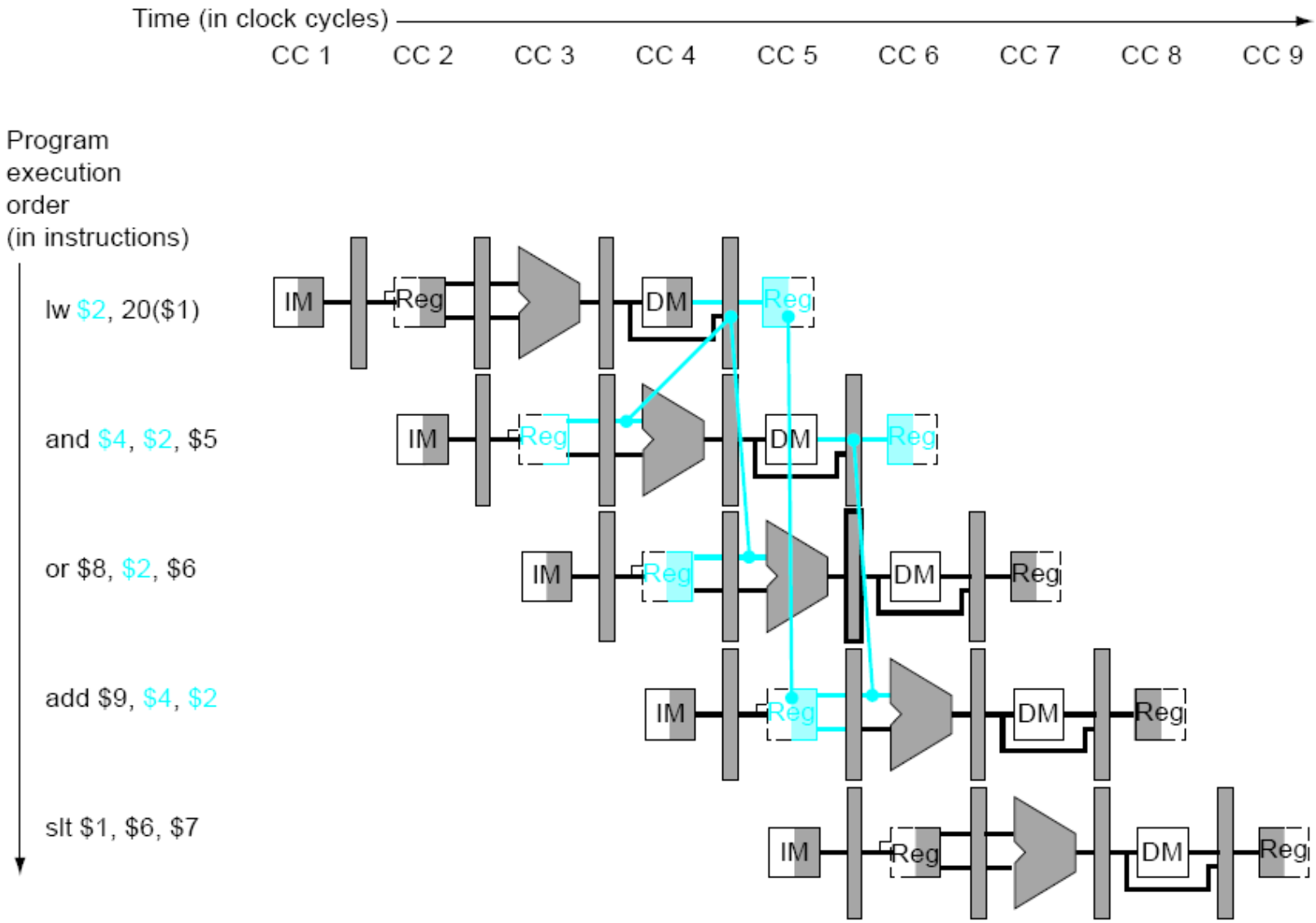
- Some data hazard stalls can be eliminated: bypassing

Forwarding Paths

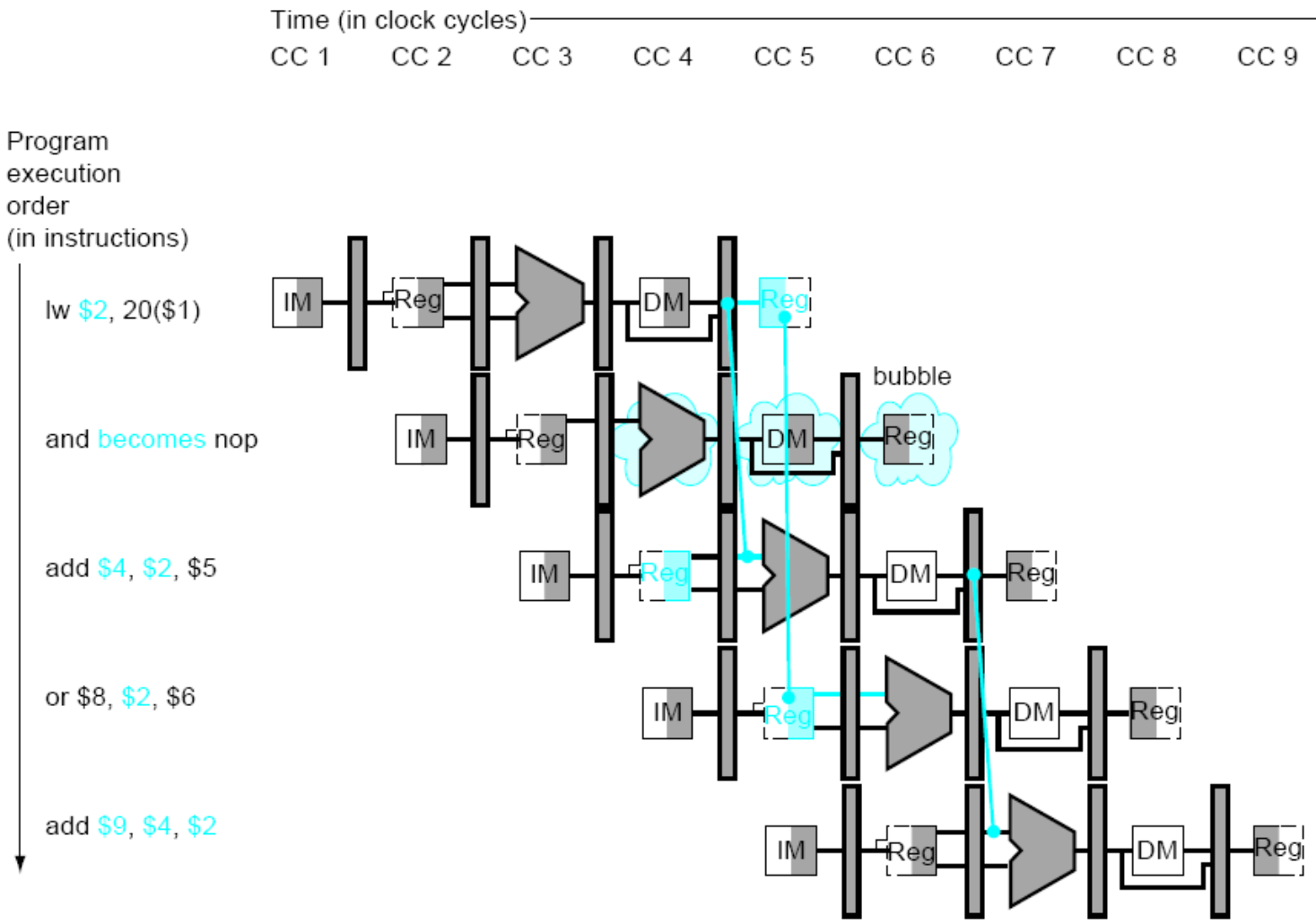


b. With forwarding

Data Hazard Stalls



Data Hazard Stalls



Control Hazards

- Simple techniques to handle control hazard stalls:
  - 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

Slowdowns from Stalls

- Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions)
- 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