Pipelining

Pipelining Example 2

- A. A non-pipelined processor takes 5ns to execute an instruction. If I want the processor to be clockable at 2GHz, how many stages should I pipeline this processor into if each latch has a 0.25ns delay?

- B. How many stages if I want to clock the processor at 5GHz?

- C. What is the maximum speedup that can be achieved by the pipelined processor running at 2GHz? (Compared to the original single cycle processor)

- D. What is the average latency of an instruction (belonging to a 23-instruction program) for the pipelined processor?

- E. What is the best case speedup and what is the worst case slowdown?

Hazards

- Situations that prevent starting the next instruction in the next cycle

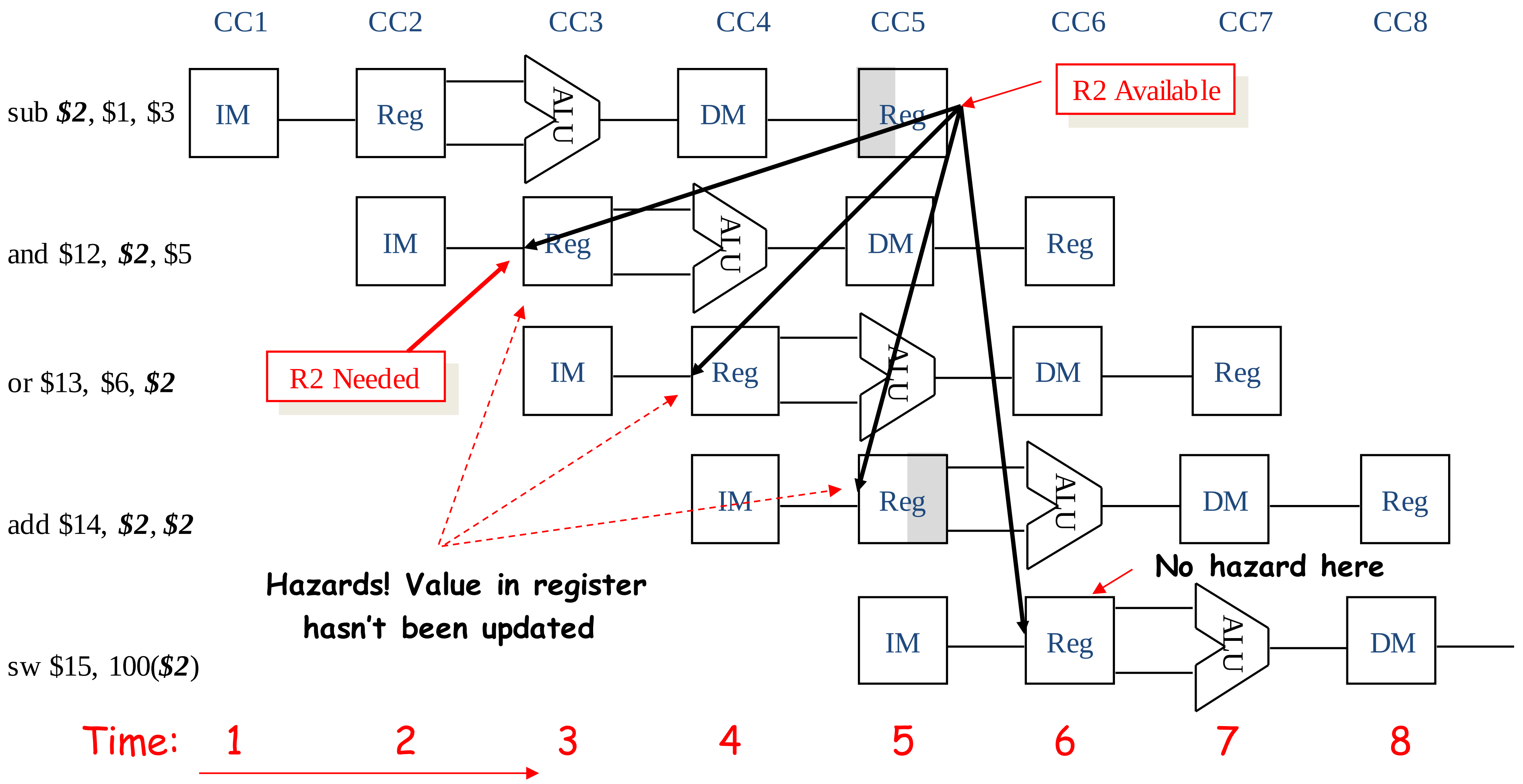
- Structure hazards
  - A required resource is busy

- Data hazard
  - Need to wait for previous instruction to complete its data read/write

- Control hazard
  - Deciding on control action depends on previous instruction

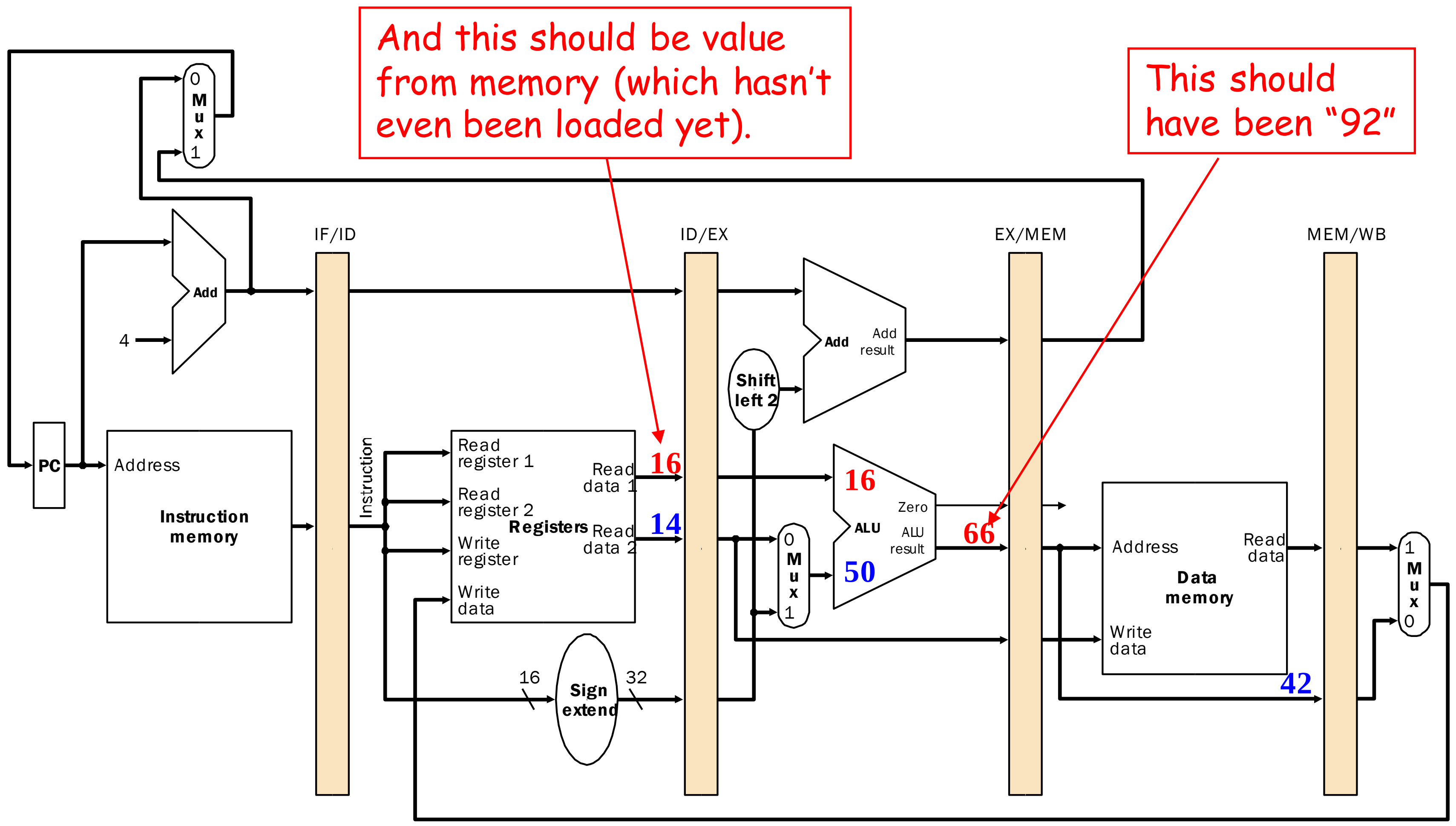
Data Hazards

- When a result is needed in the pipeline before it is available, a "data hazard" occurs.

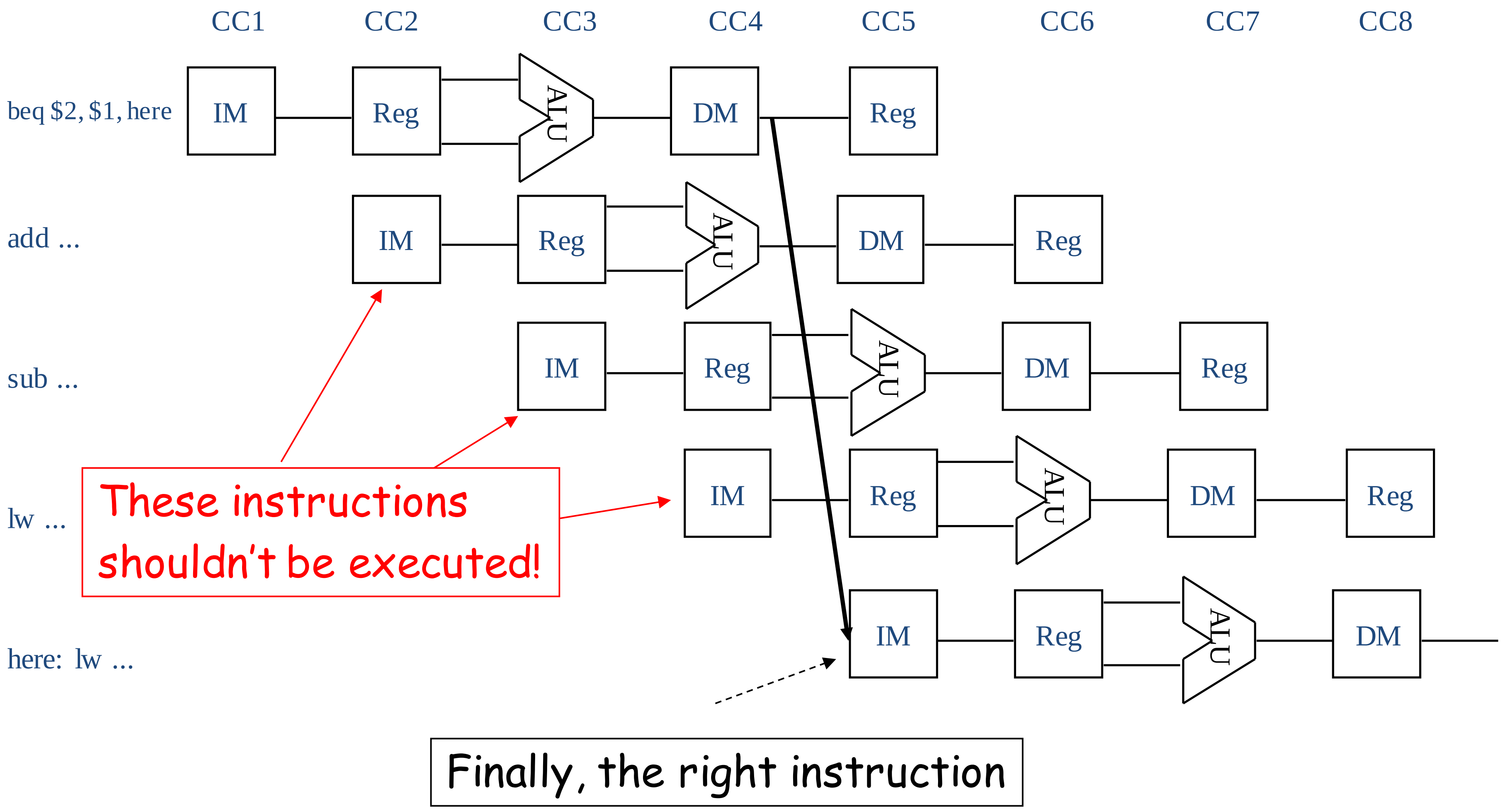


The Pipeline in Execution

add $10, $1, $2   **add $11, $8, $7**   **lw $8, 52($3)**   **add $3, $10, $11**   Write Back



Branch Hazards

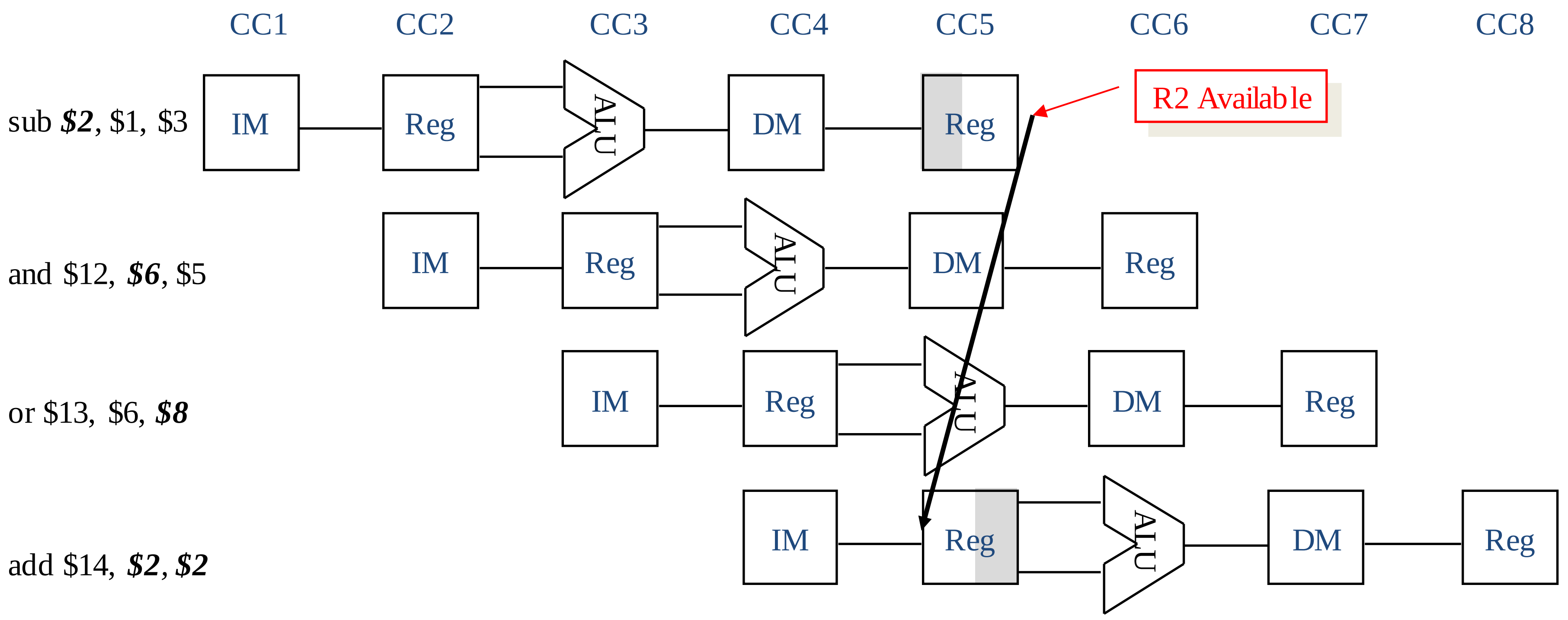


Dealing With Data Hazards

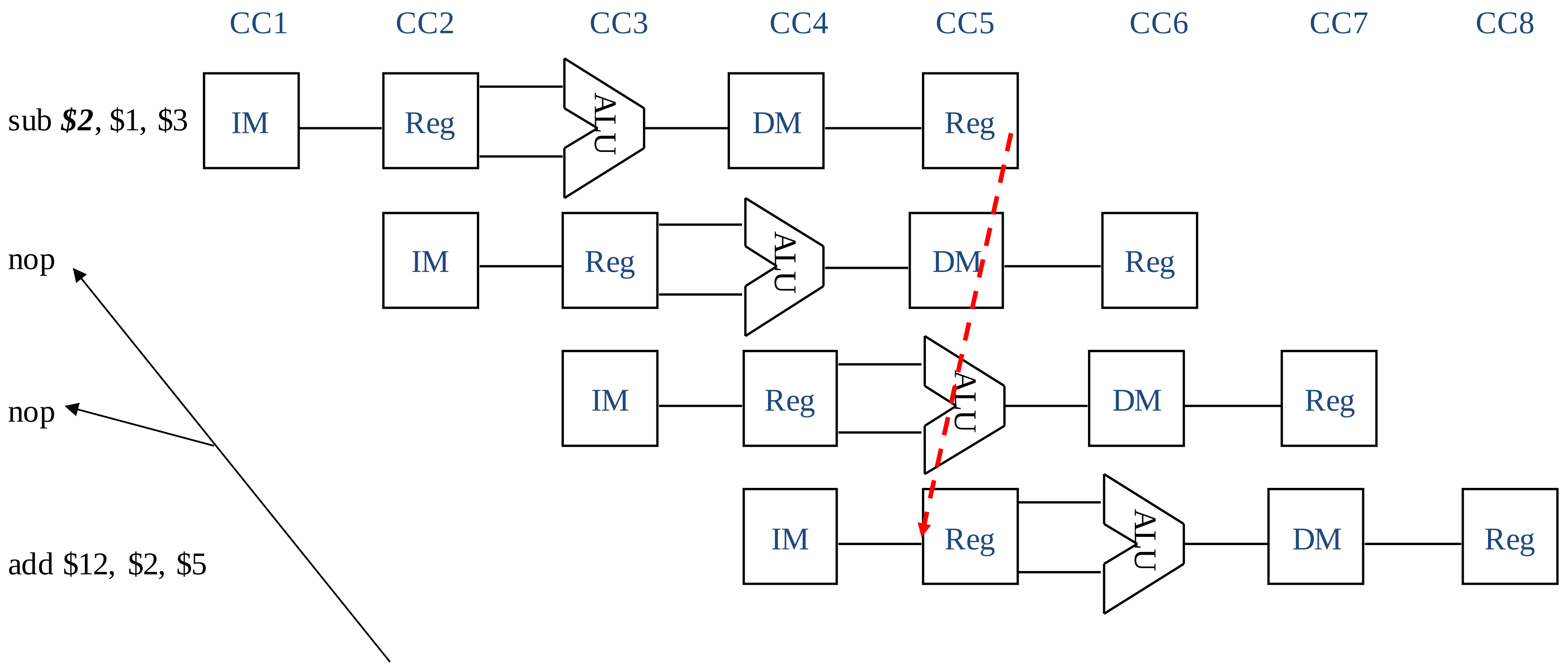
Transparent register file eliminates one hazard.

Use latches rather than flip-flops in Reg file

- First half-cycle of cycle 5: register 2 loaded
- Second half-cycle: new value is read into pipeline state



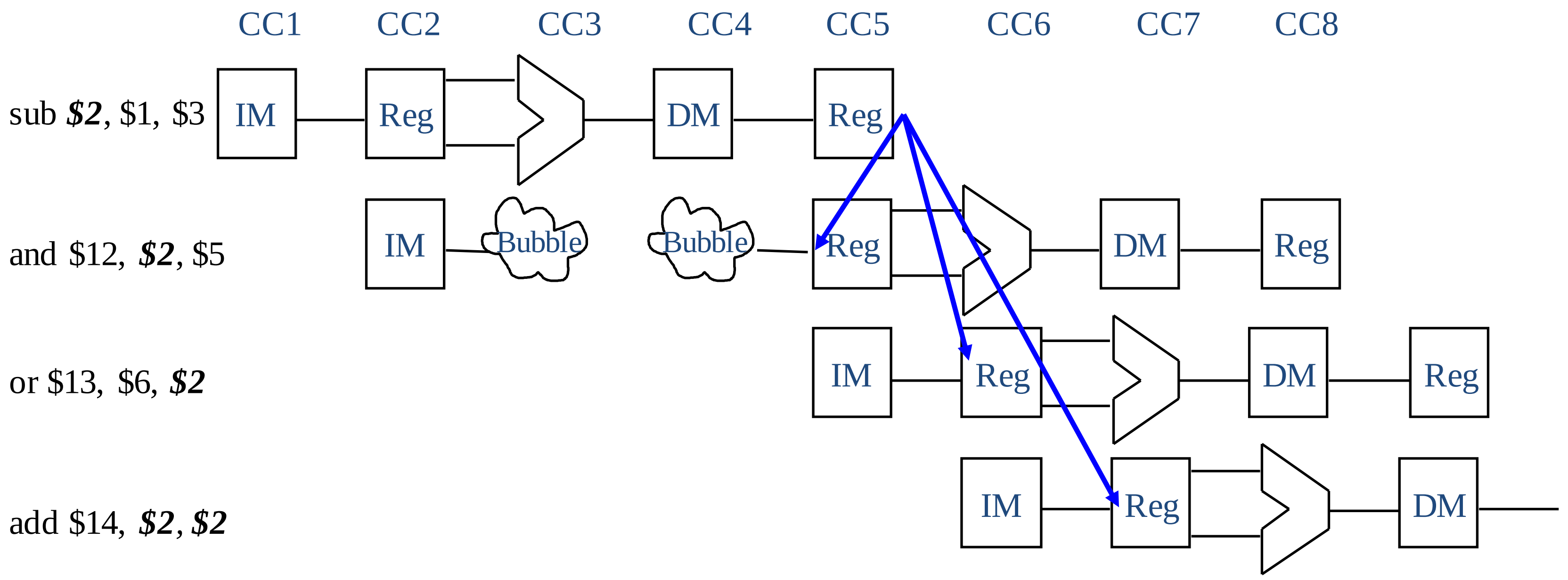
Dealing with Data Hazards in Software



Insert enough no-ops (or other instructions that don't use register 2) so that data hazard doesn't occur,

Handling Data Hazards in Hardware

Stall the pipeline

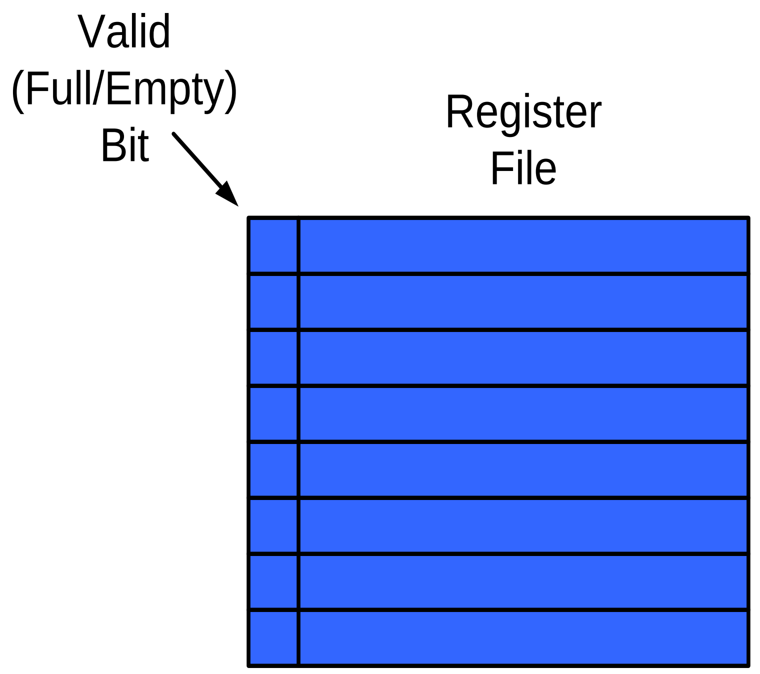


Pipeline Stalls

- To insure proper pipeline execution in light of register dependences, we must:
  - Detect the hazard
  - Stall the pipeline
    - prevent the IF and ID stages from making progress
    - insert“no-ops” into later stages

Register Scoreboard – Tracking Operand Availability

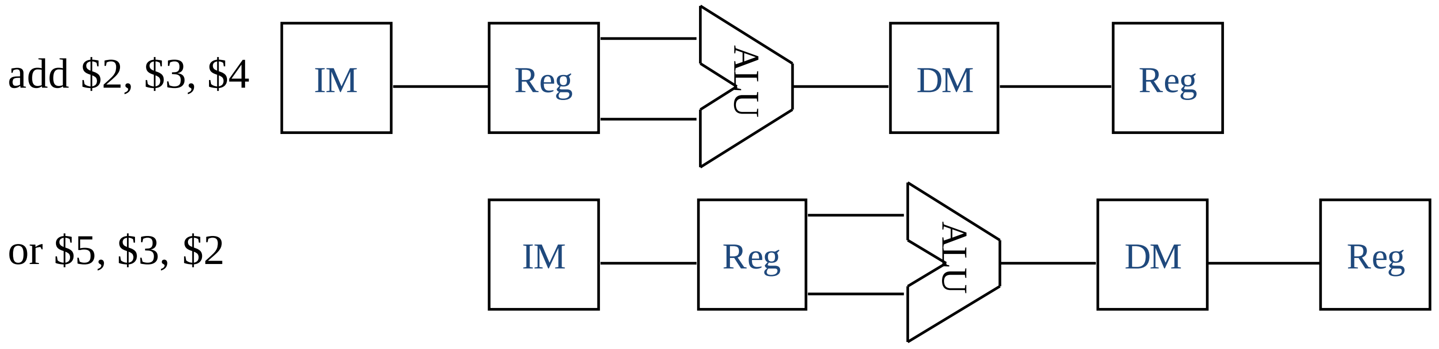
- Add valid bit to each register in the register file
- Hardware clears valid bit when an instruction that writes the register issues (leaves decode/register read stage)
- Hardware sets valid bit when an instruction that writes the register completes
- Instructions not allowed to issue if any of their source registers are invalid



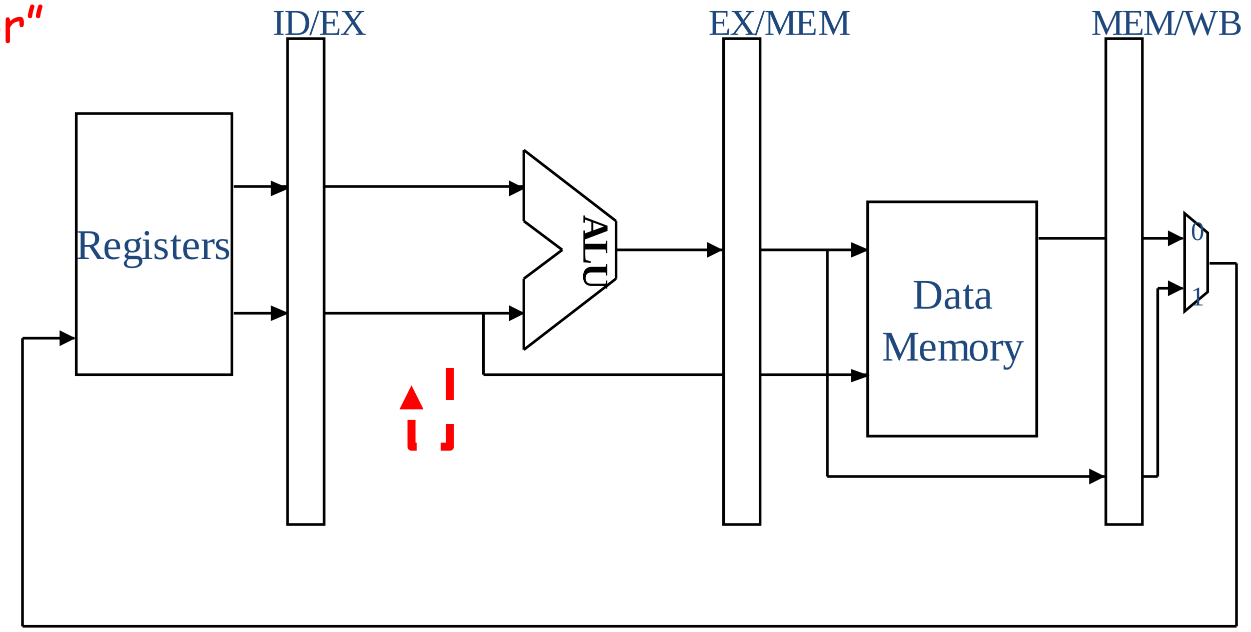
Stalling the Pipeline

- Prevent the IF and ID stages from proceeding
  - don’t write the PC (PCWrite = 0)
  - don’t rewrite IF/ID register (IF/IDWrite = 0)
- Insert “nops”
  - set all control signals propagating to EX/MEM/WB to zero

Reducing Data Hazards Through Forwarding



We could avoid stalling if we could get the ALU output from "add" to ALU input for the "or"


ID/EX
EX/MEM
MEM/WB
Registers
ALU
Data Memory
0
1