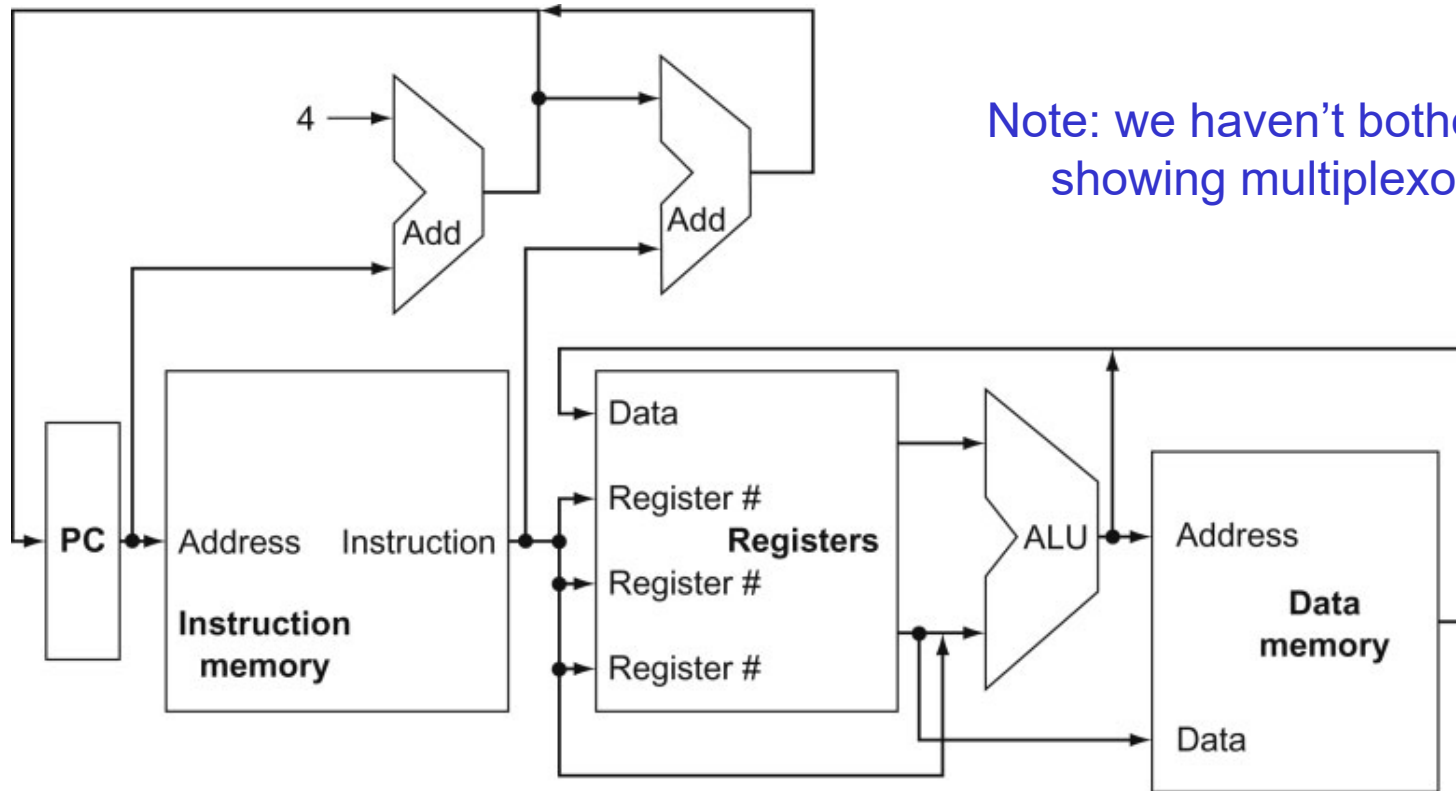# Lecture 17: Basic Pipelining

- Today's topics:

  - 1-stage design
  - 5-stage design
  - 5-stage pipeline
  - Hazards

# View from 30,000 Feet
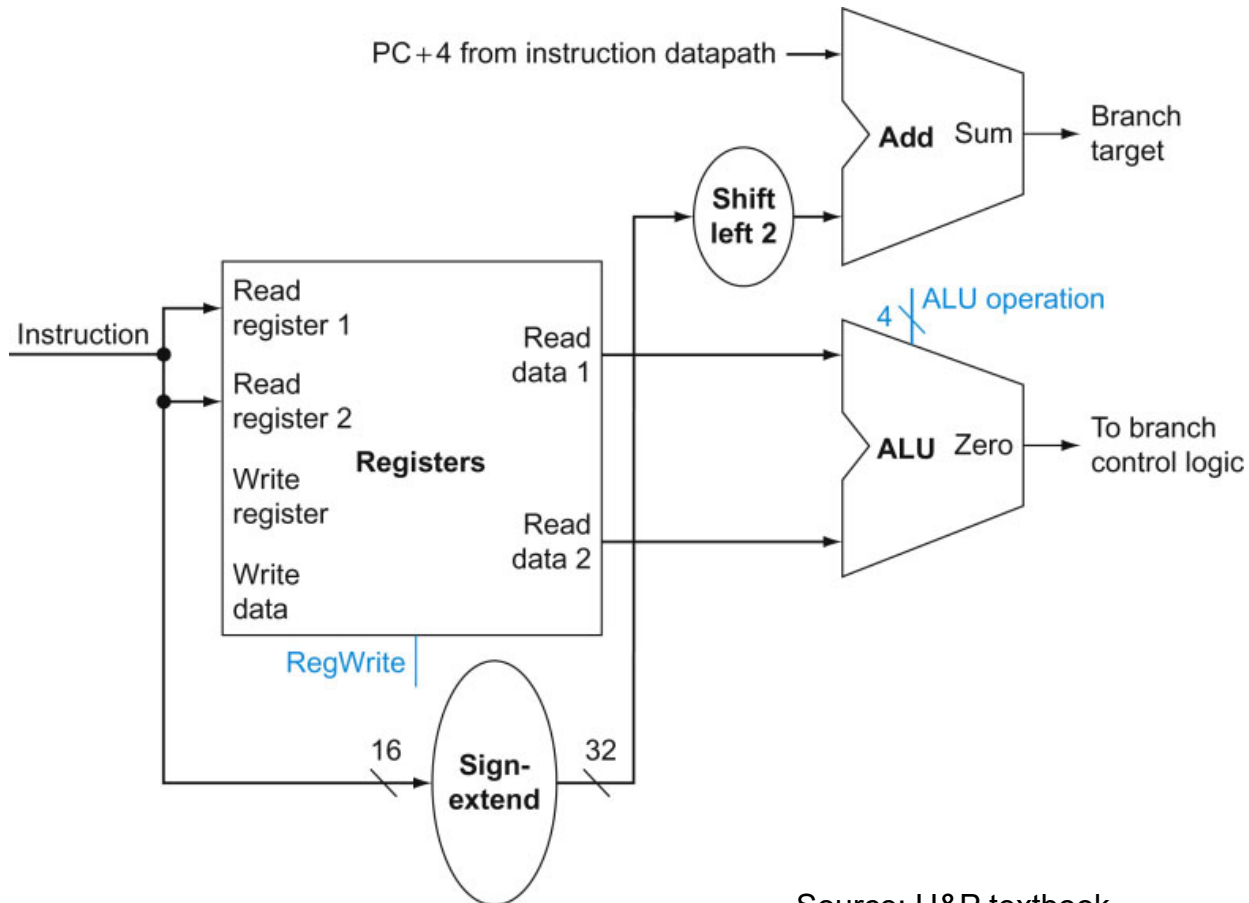


Note: we haven't bothered showing multiplexors

Source: H&P textbook

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

2

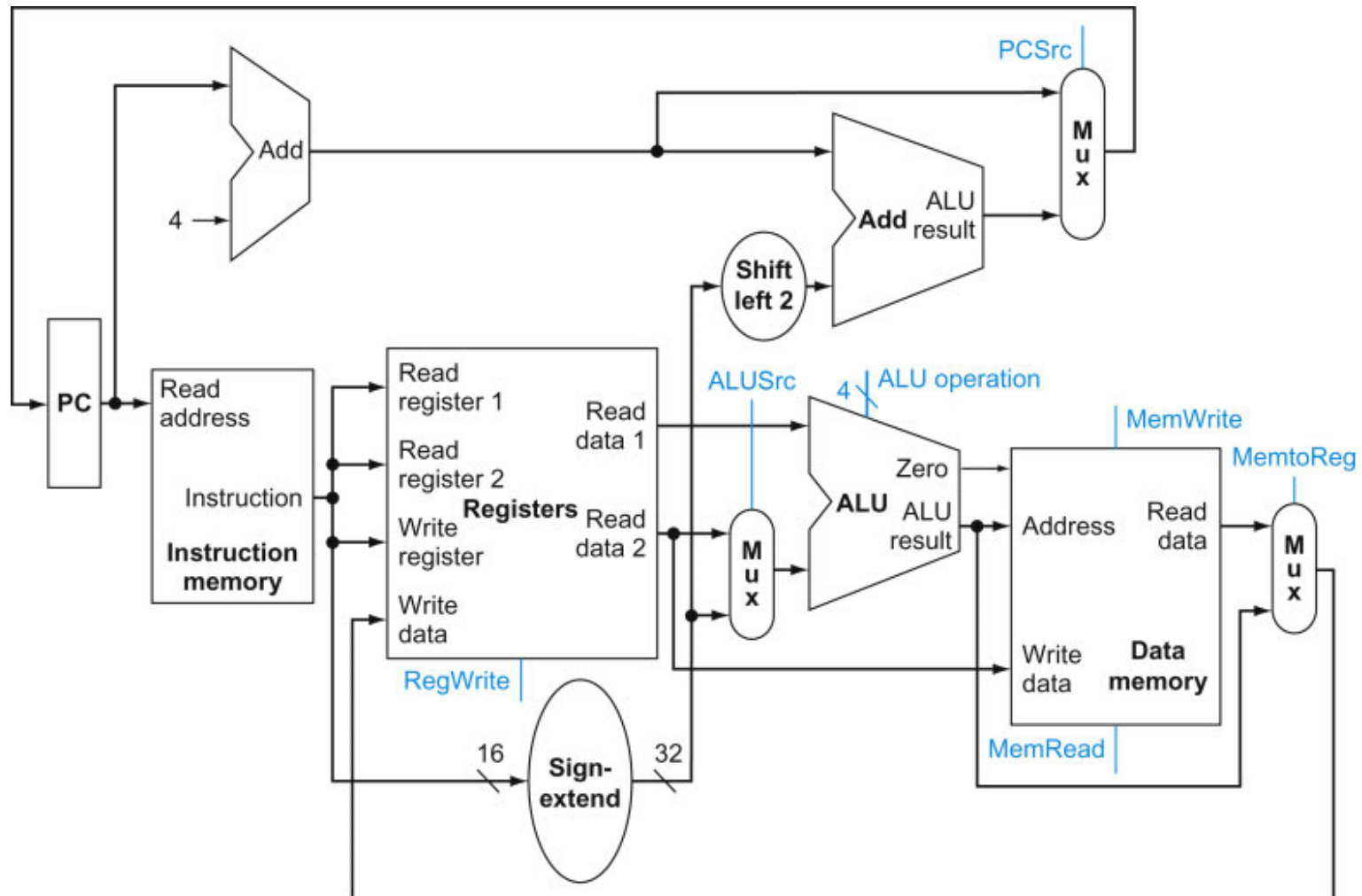# Implementing J-type Instructions

• Instructions of the form   beq $t1, $t2, offset
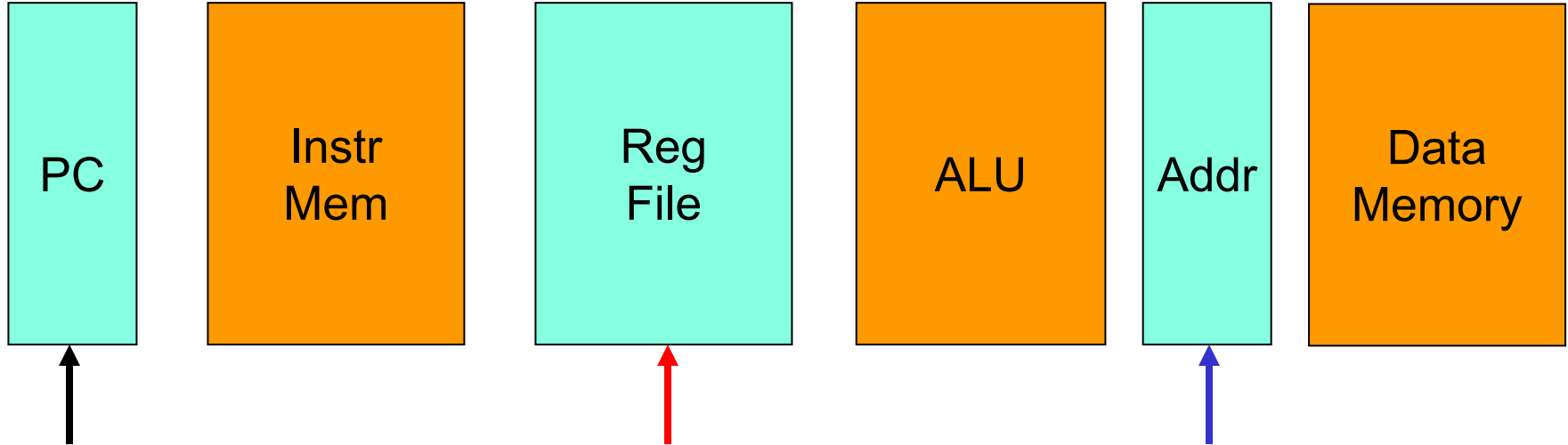


Source: H&P textbook

3

# View from 10,000 Feet



4

Source: H&P textbook

# View from 5,000 Feet

Source: H&P textbook

# Latches and Clocks in a Single-Cycle Design

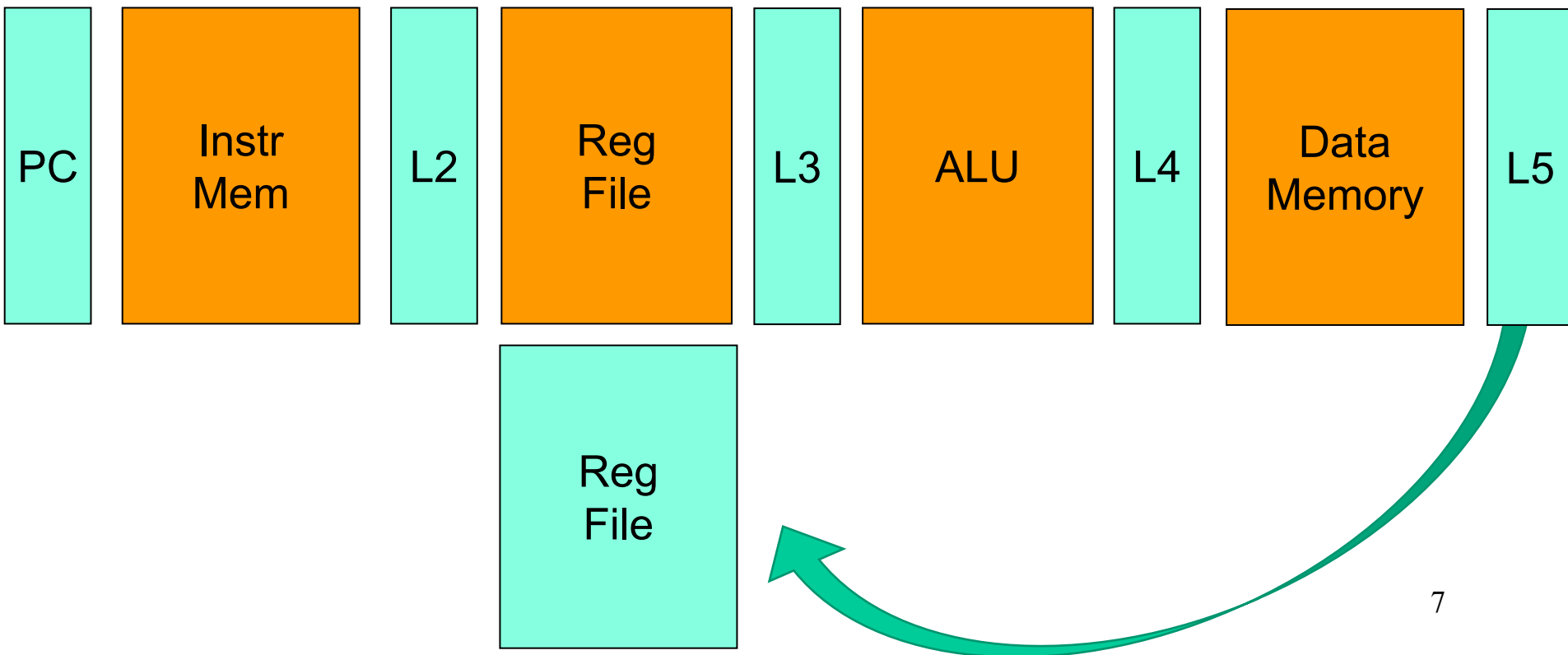| PC | Instr Mem | Reg File | ALU | Addr | Data Memory |
|----|-----------|----------|-----|------|-------------|

- The entire instruction executes in a single cycle
- Green blocks are latches
- At the rising edge, a new PC is recorded
- At the rising edge, the result of the previous cycle is recorded
- At the falling edge, the address of LW/SW is recorded so we can access the data memory in the 2$^{nd}$ half of the cycle
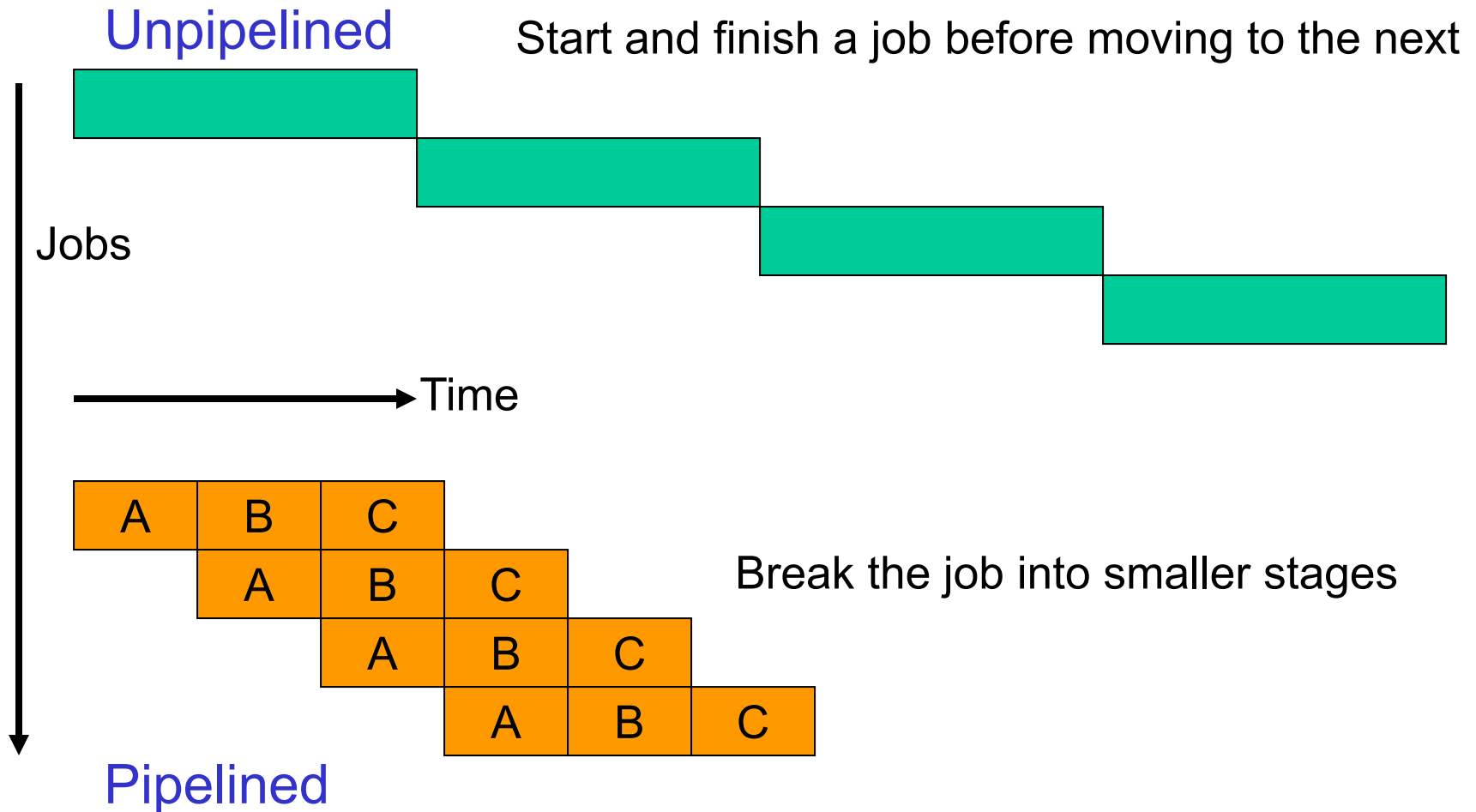
6

# Multi-Stage Circuit

- Instead of executing the entire instruction in a single cycle (a single stage), let's break up the execution into multiple stages, each separated by a latch

# The Assembly Line

Unpipelined  Start and finish a job before moving to the next

Jobs

Time

| A | B | C |   |   |
|---|---|---|---|---|
|   | A | B | C |   |
|   |   | A | B | C |
|   |   |   | A | B | C |

Break the job into smaller stages

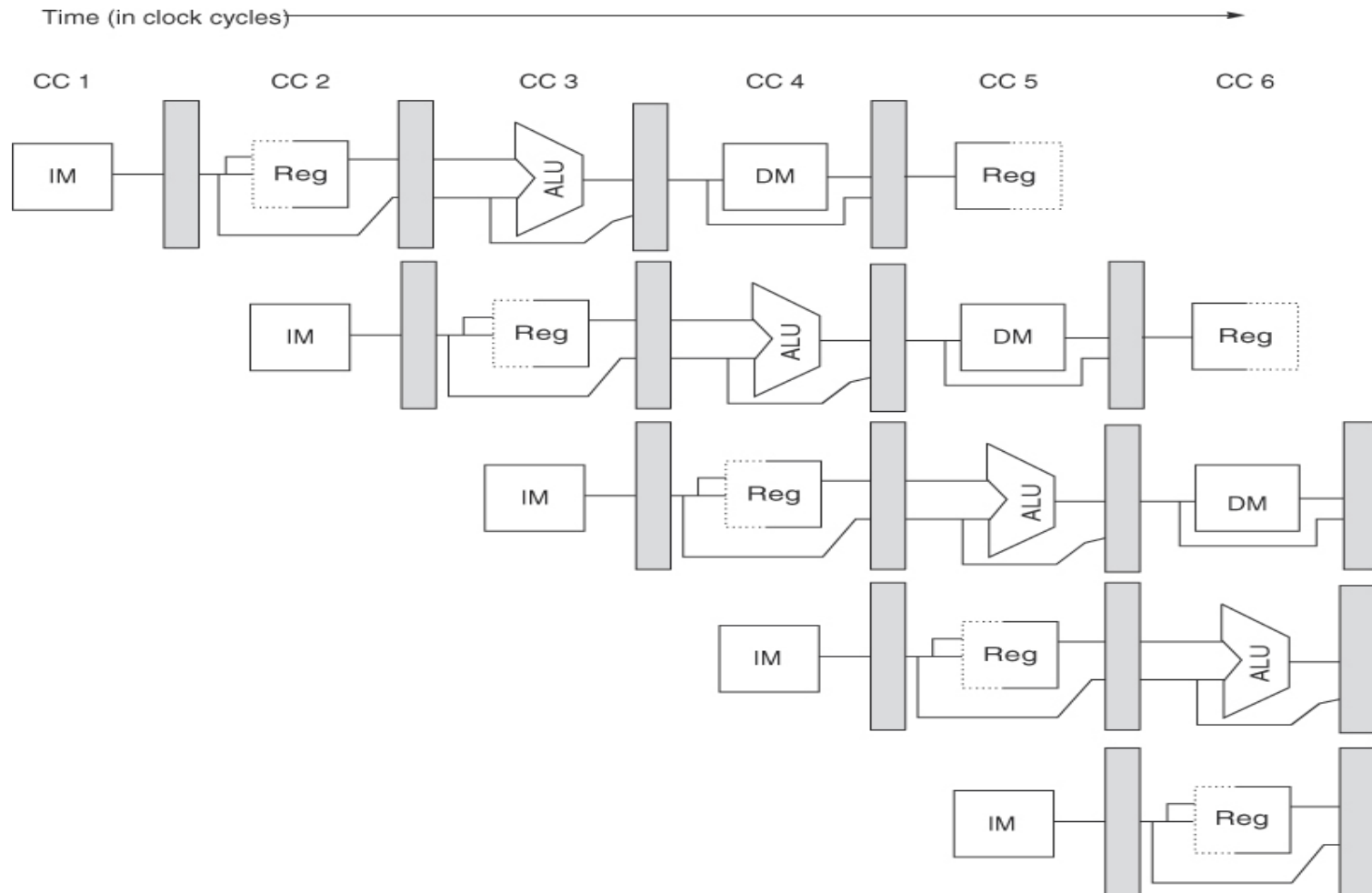Pipelined

# Performance Improvements?

- Does it take longer to finish each individual job?

- Does it take shorter to finish a series of jobs?

- What assumptions were made while answering these questions?

- Is a 10-stage pipeline better than a 5-stage pipeline?

# Quantitative Effects

- As a result of pipelining:
  - ➢ Time in ns per instruction goes up
  - ➢ Each instruction takes more cycles to execute
  - ➢ But… average CPI remains roughly the same
  - ➢ Clock speed goes up
  - ➢ Total execution time goes down, resulting in lower average time per instruction
  - ➢ Under ideal conditions, speedup
    = ratio of *elapsed times between successive instruction completions*
    = number of pipeline stages = increase in clock speed
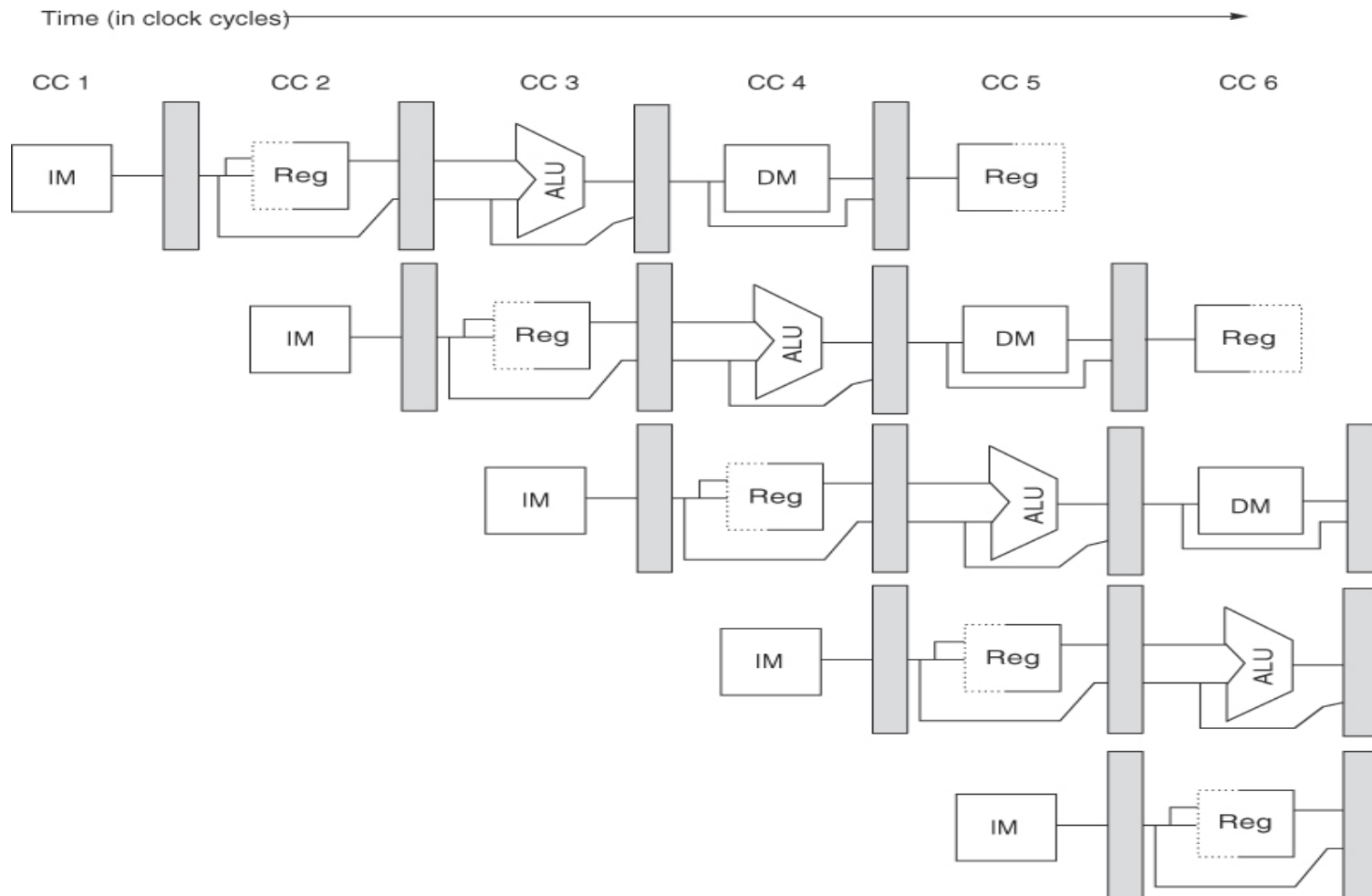
# A 5-Stage Pipeline

# A 5-Stage Pipeline

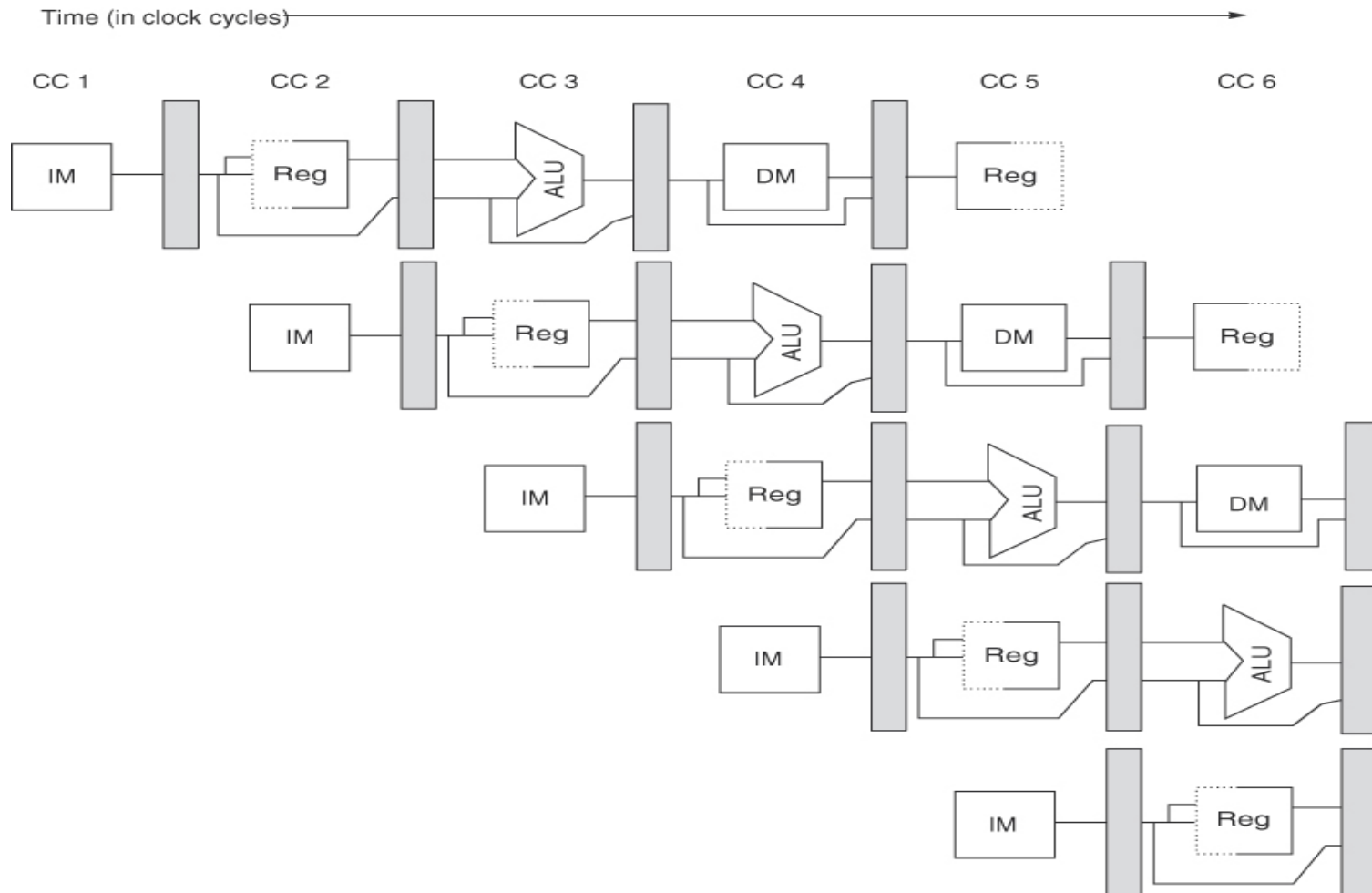Use the PC to access the I-cache and increment PC by 4
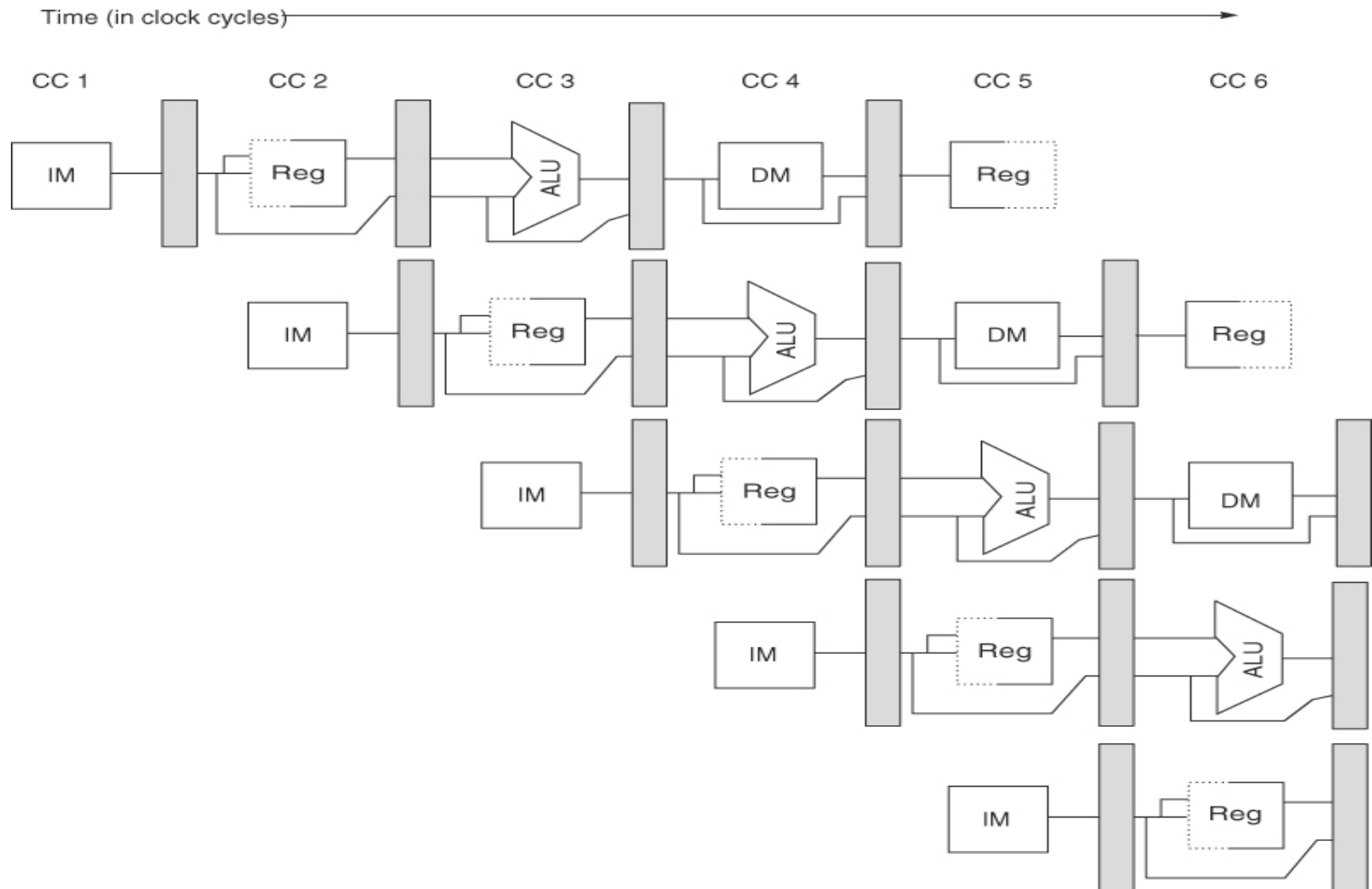
# A 5-Stage Pipeline

Read registers, compare registers, compute branch target; for now, assume branches take 2 cyc (there is enough work that branches can easily take more)
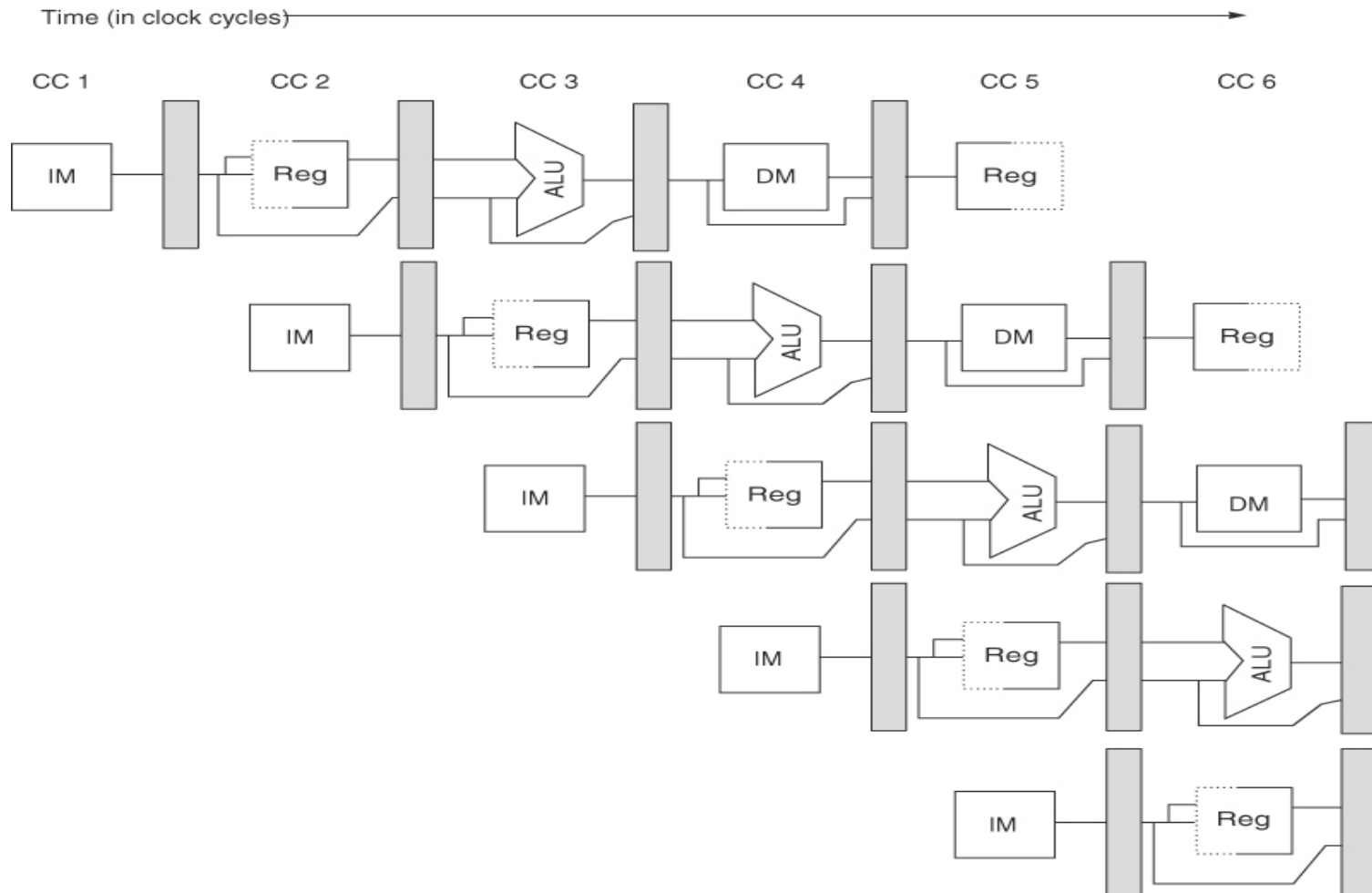
# A 5-Stage Pipeline

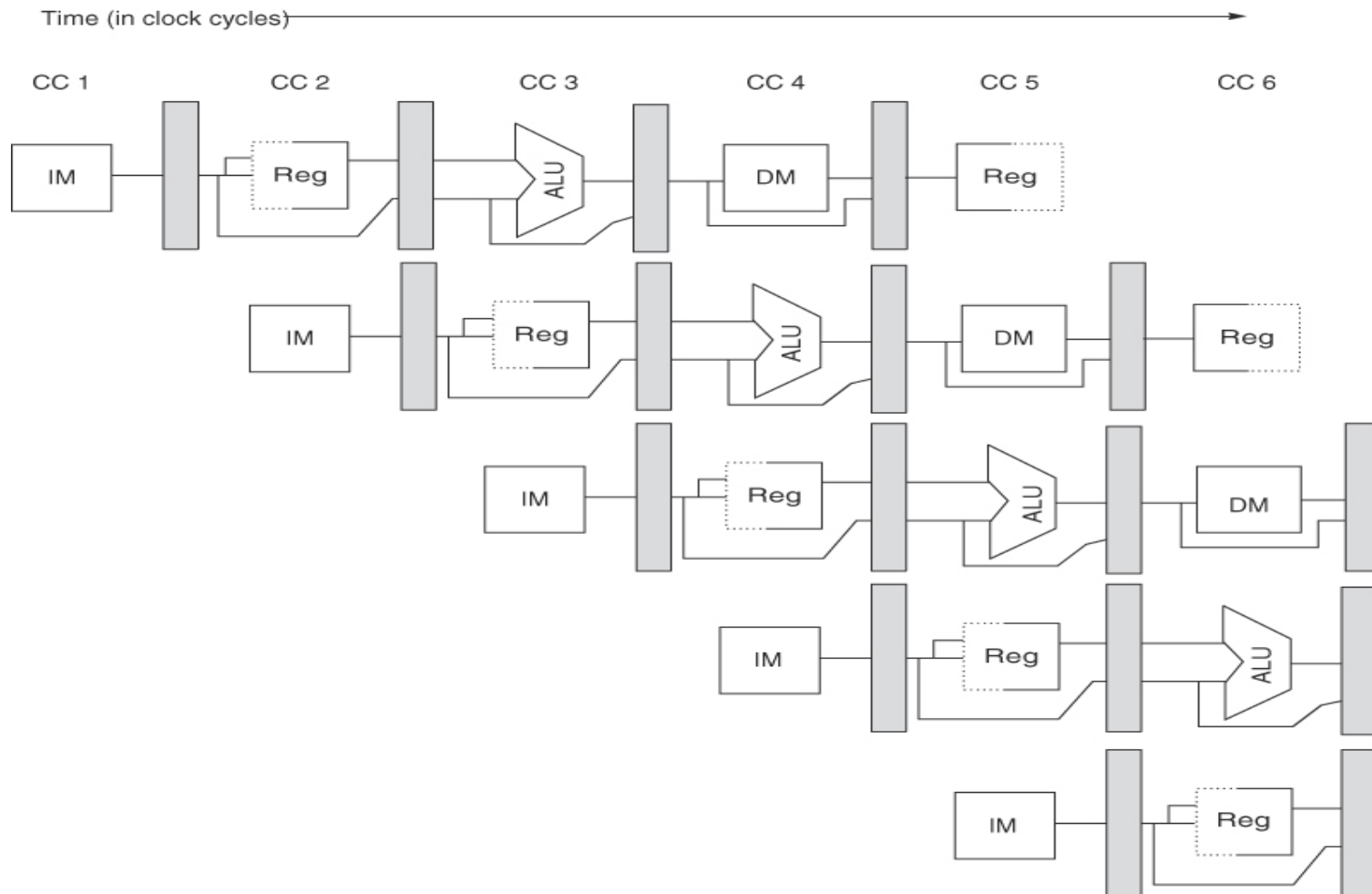ALU computation, effective address computation for load/store

# A 5-Stage Pipeline

Memory access to/from data cache, stores finish in 4 cycles

# A 5-Stage Pipeline

Write result of ALU computation or load into register file

# Pipeline Summary

|  | RR | ALU | DM | RW |
|---|---|---|---|---|
| ADD R1, R2, → R3 | Rd R1,R2 | R1+R2 | -- | Wr R3 |
| BEQ R1, R2, 100 | Rd R1, R2 | -- | -- | -- |
| | Compare, Set PC | | | |
| LD 8[R3] → R6 | Rd R3 | R3+8 | Get data | Wr R6 |
| ST 8[R3] ← R6 | Rd R3,R6 | R3+8 | Wr data | -- |

# Conflicts/Problems

- I-cache and D-cache are accessed in the same cycle – it helps to implement them separately

- Registers are read and written in the same cycle – easy to deal with if register read/write time equals cycle time/2

- Branch target changes only at the end of the second stage -- what do you do in the meantime?

# 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