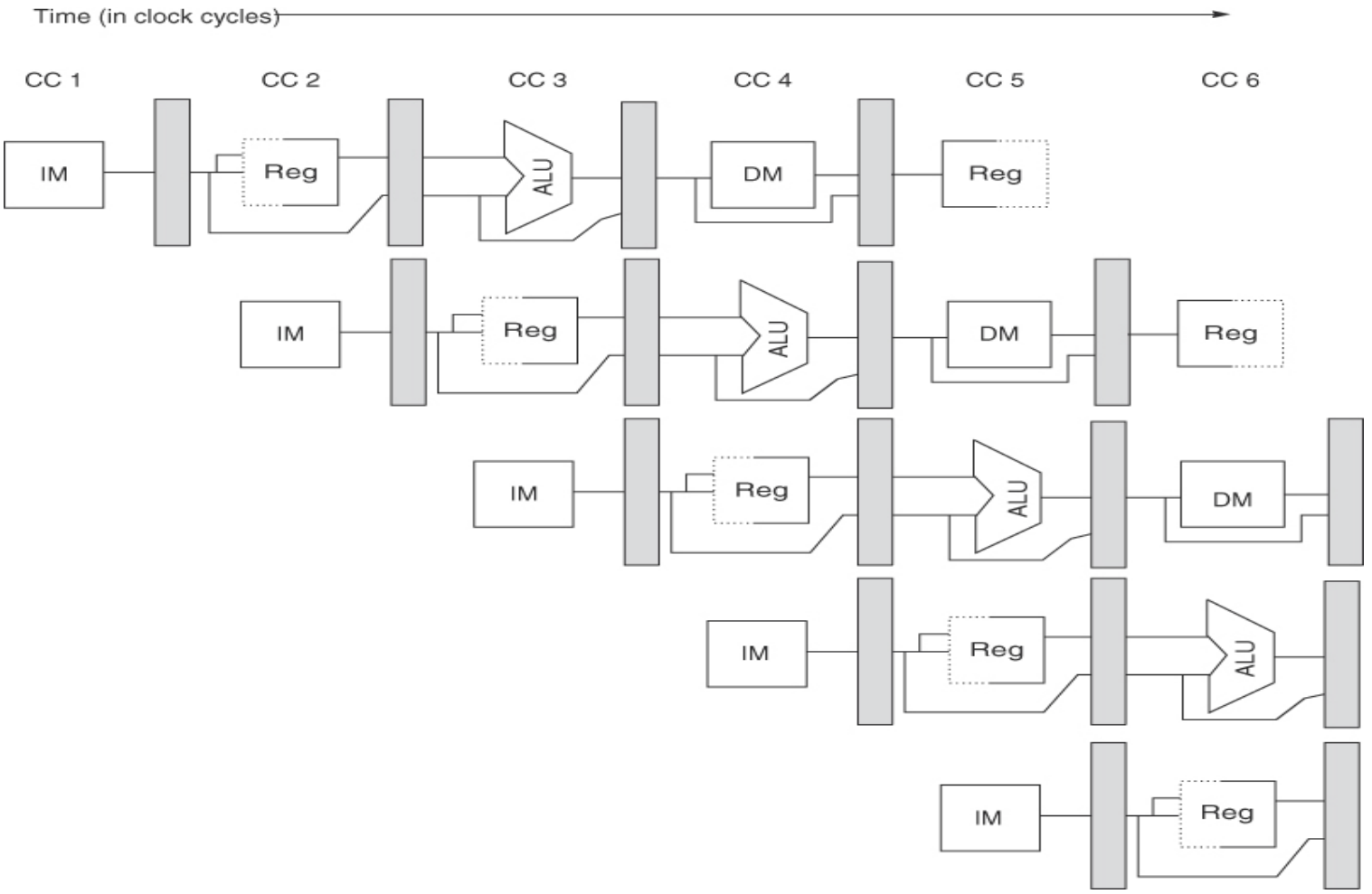
Lecture: Pipelining Hazards

- Topics: structural and data hazards
- HW2 posted later today; due in a week

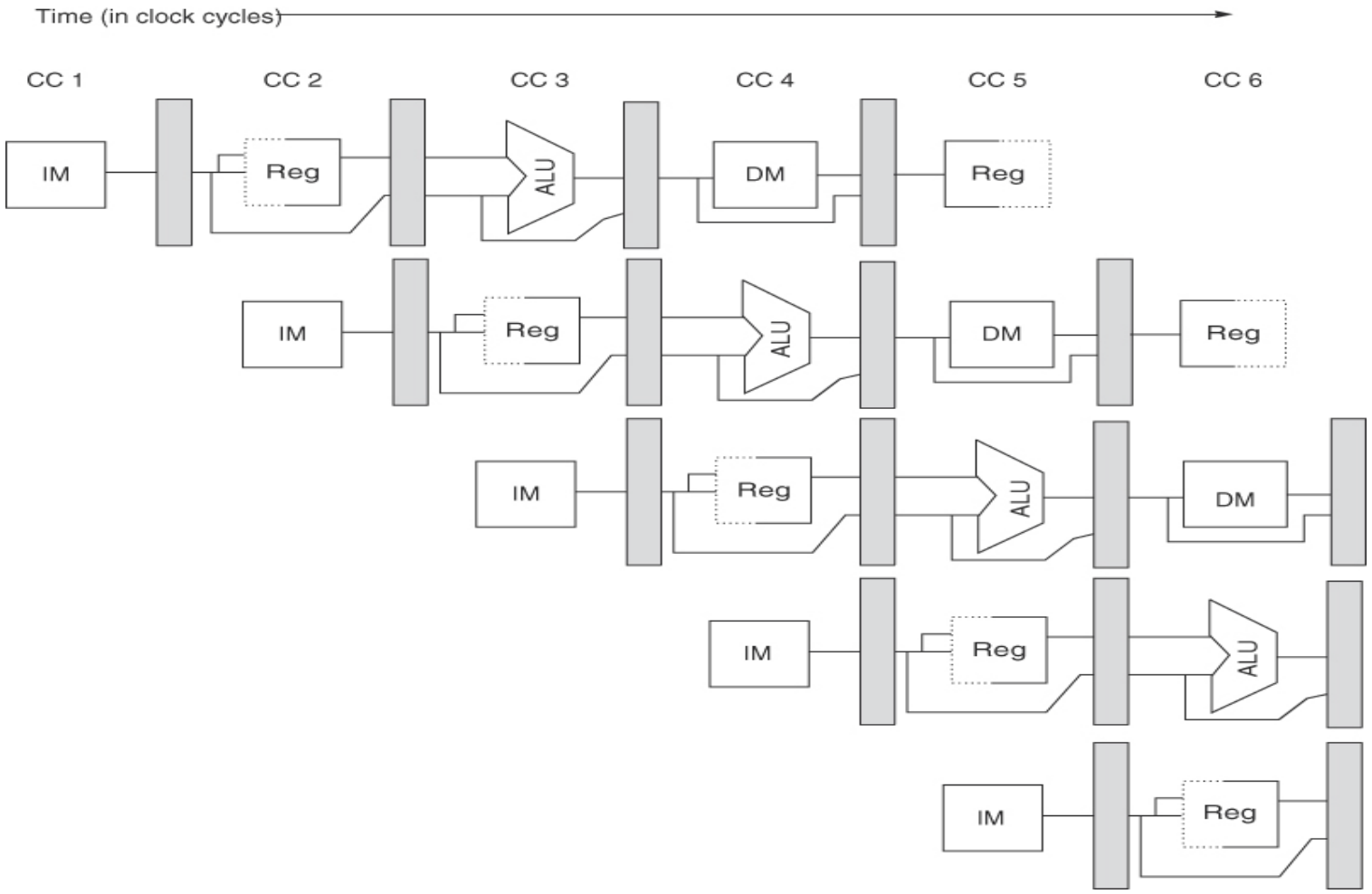
A 5-Stage Pipeline



Source: H&P textbook

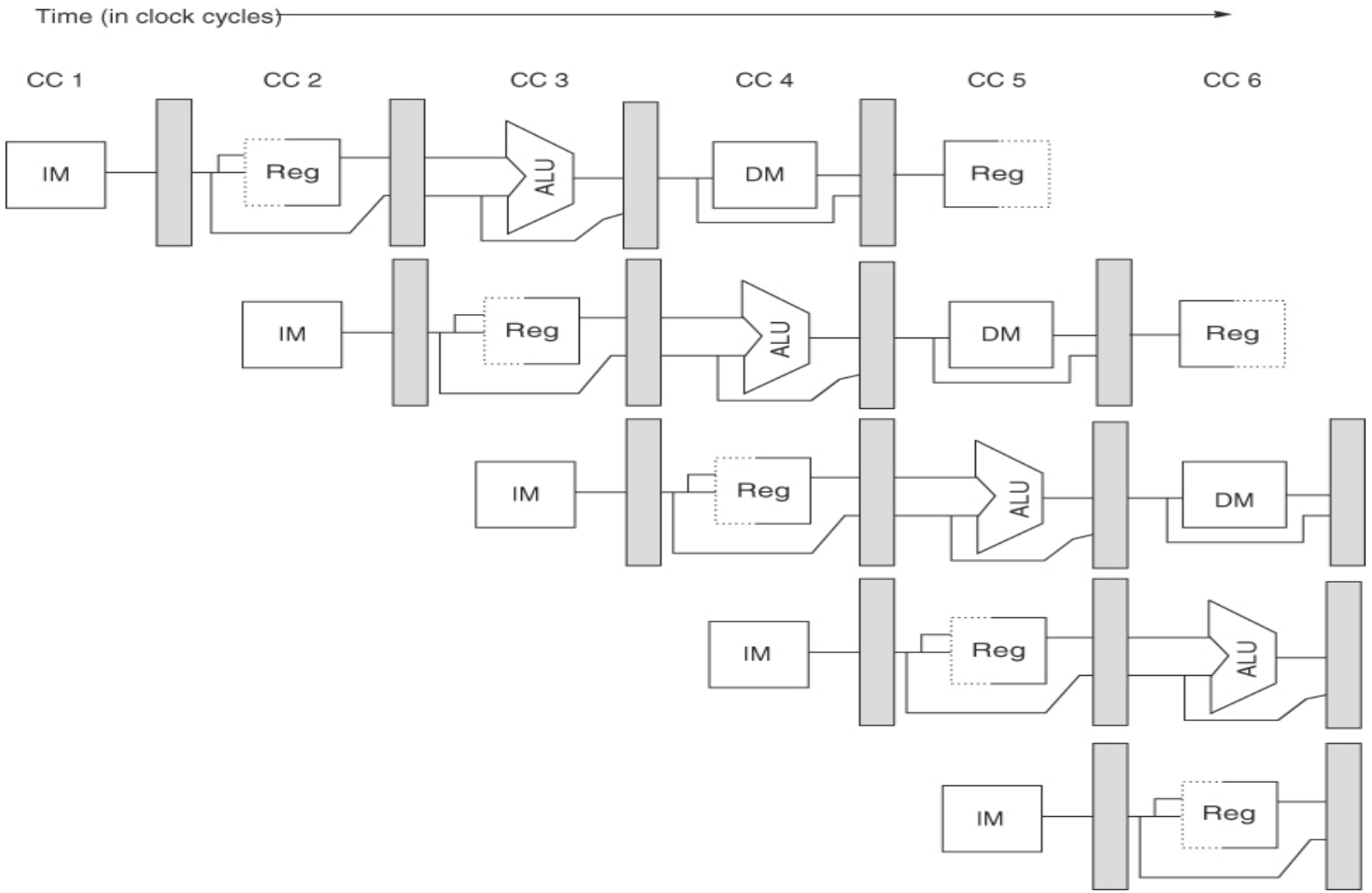
A 5-Stage Pipeline

Use the PC to access the I-cache and increment PC by 4



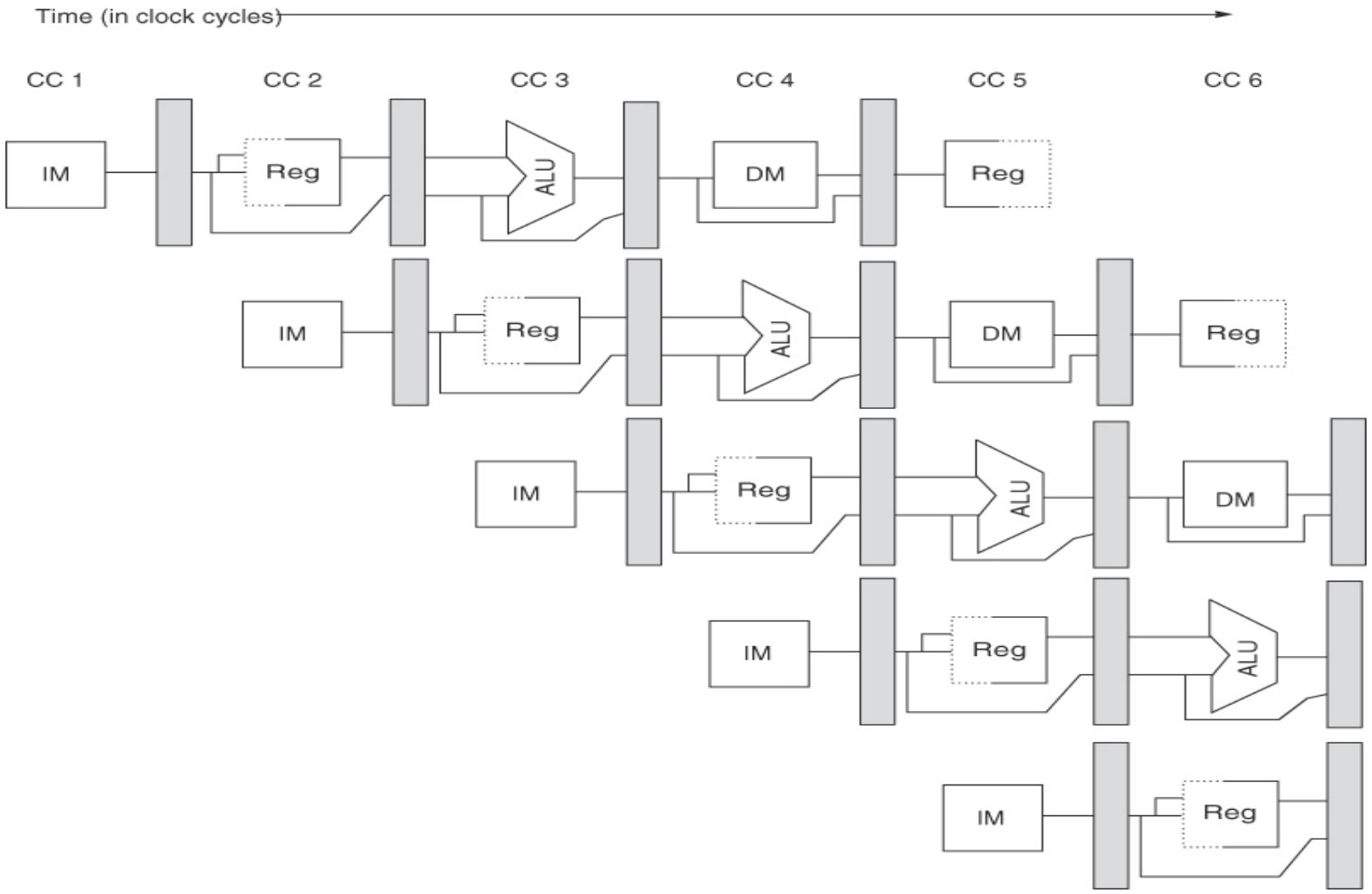
A 5-Stage Pipeline

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



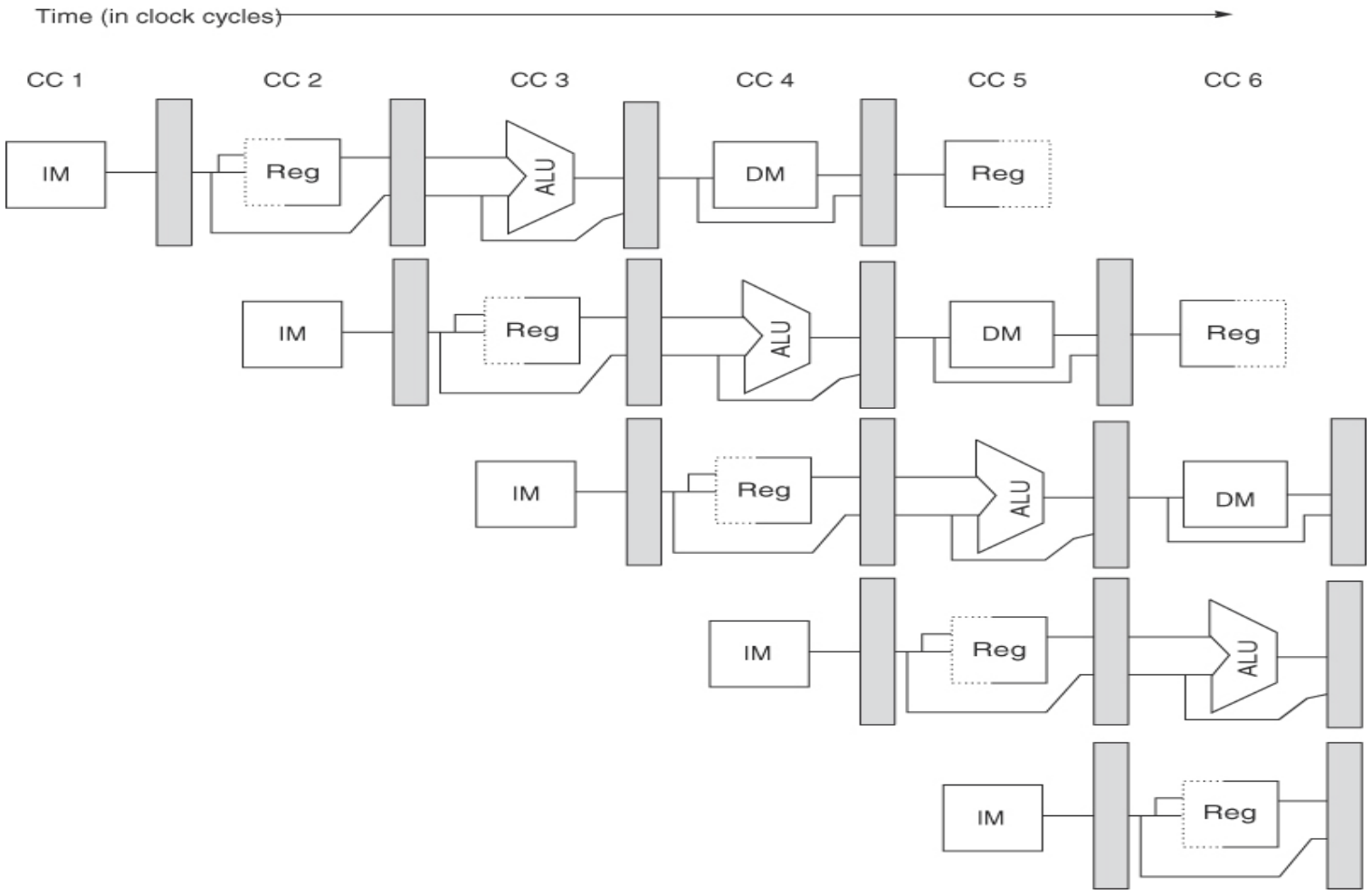
A 5-Stage Pipeline

ALU computation, effective address computation for load/store



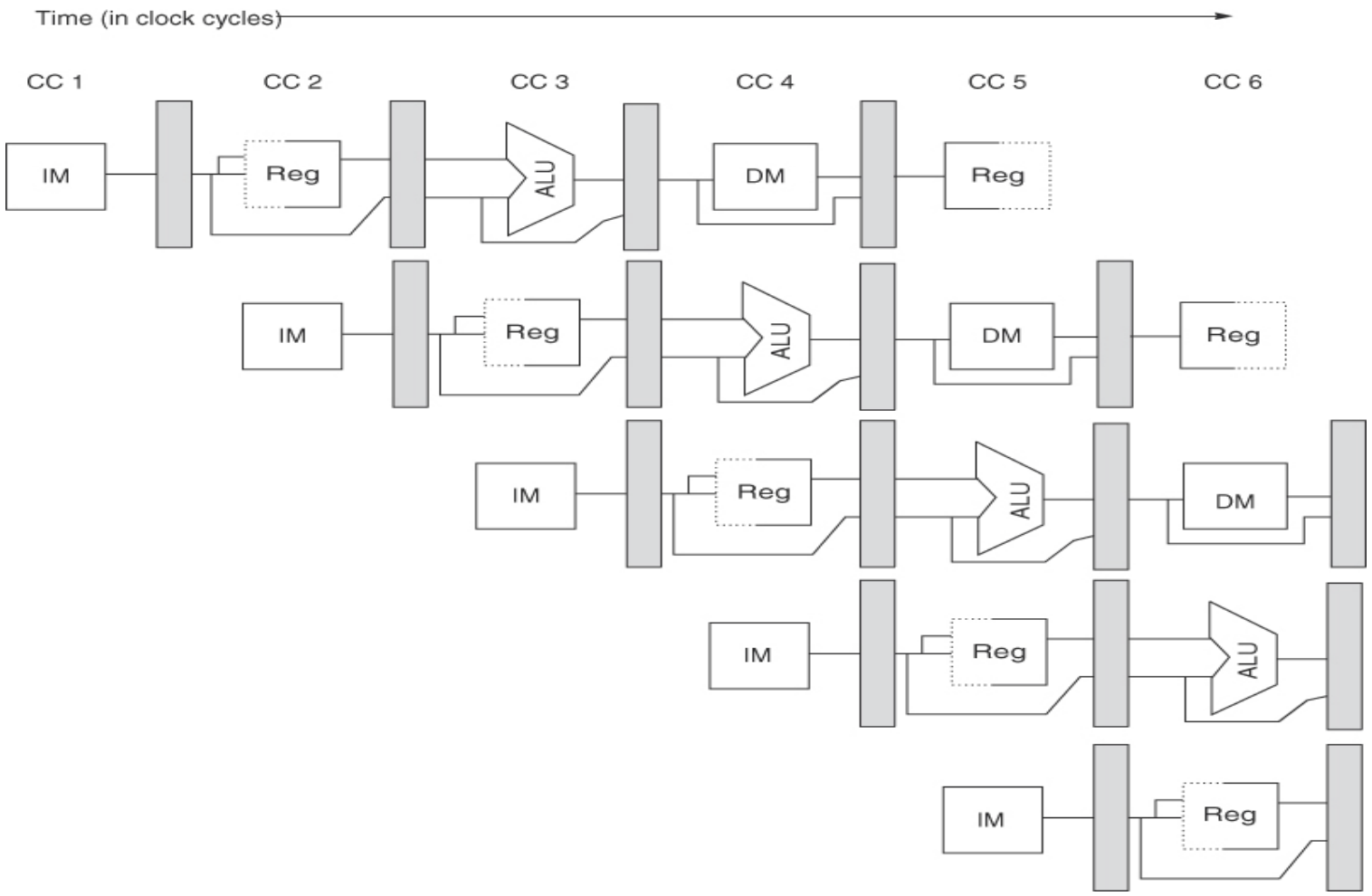
A 5-Stage Pipeline

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



RISC/CISC Loads/Stores

Registers and memory
Complex and reduced instrs
Format of a load/store

Problem 3

- Convert this C code into equivalent RISC assembly instructions

```
a[i] = b[i] + c[i];
```

Problem 3

- Convert this C code into equivalent RISC assembly instructions

```
a[i] = b[i] + c[i];

LD  R2, [R1]          # R1 has the address for variable i
MUL  R3, R2, 8        # the offset from the start of the array
ADD  R7, R3, R4       # R4 has the address of a[0]
ADD  R8, R3, R5       # R5 has the address of b[0]
ADD  R9, R3, R6       # R6 has the address of c[0]
LD R10, [R8]          #  Bringing b[i]
LD R11, [R9]          #  Bringing c[i]
ADD R12, R11, R10     # Sum is in R12
ST  R12, [R7]         # Putting result in a[i]
```

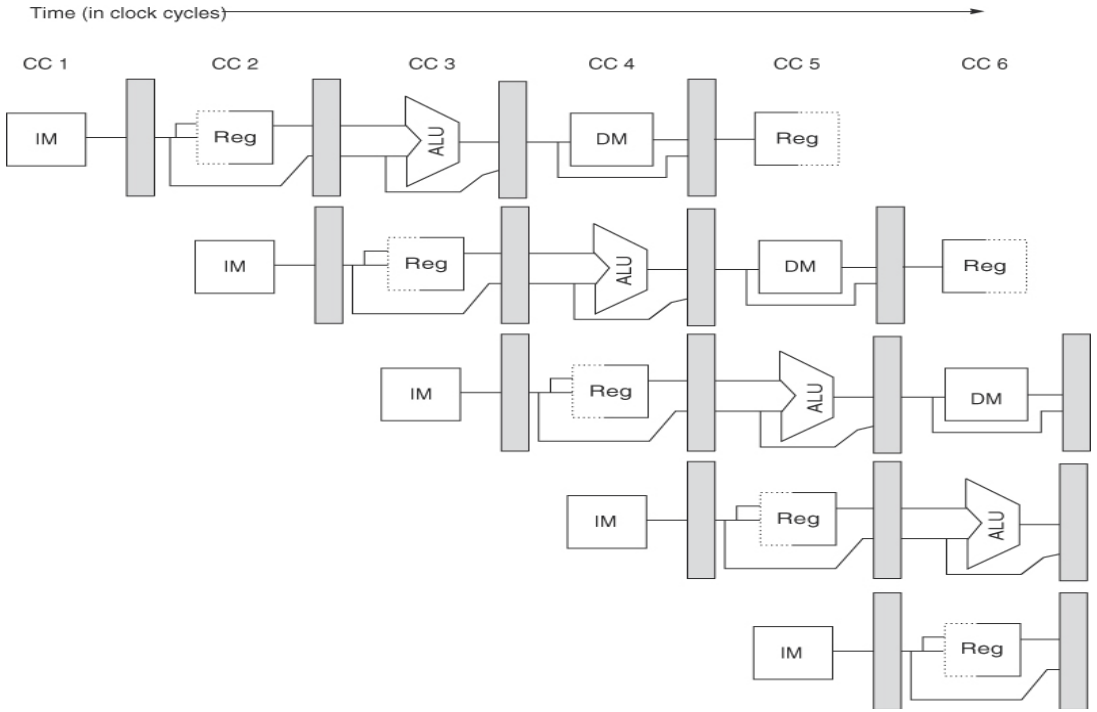
Pipeline Summary

	RR	ALU	DM	RW
ADD R3 ← R1, R2	Rd R1,R2	R1+R2	--	Wr R3
BEZ R1, [R5]	Rd R1, R5 Compare, Set PC	--	--	--
LD R6 ← 8[R3]	Rd R3	R3+8	Get data	Wr R6
ST R6 → 8[R3]	Rd R3,R6	R3+8	Wr data	--

Problem 4

- For the following code sequence, show how the instrs flow through the pipeline:

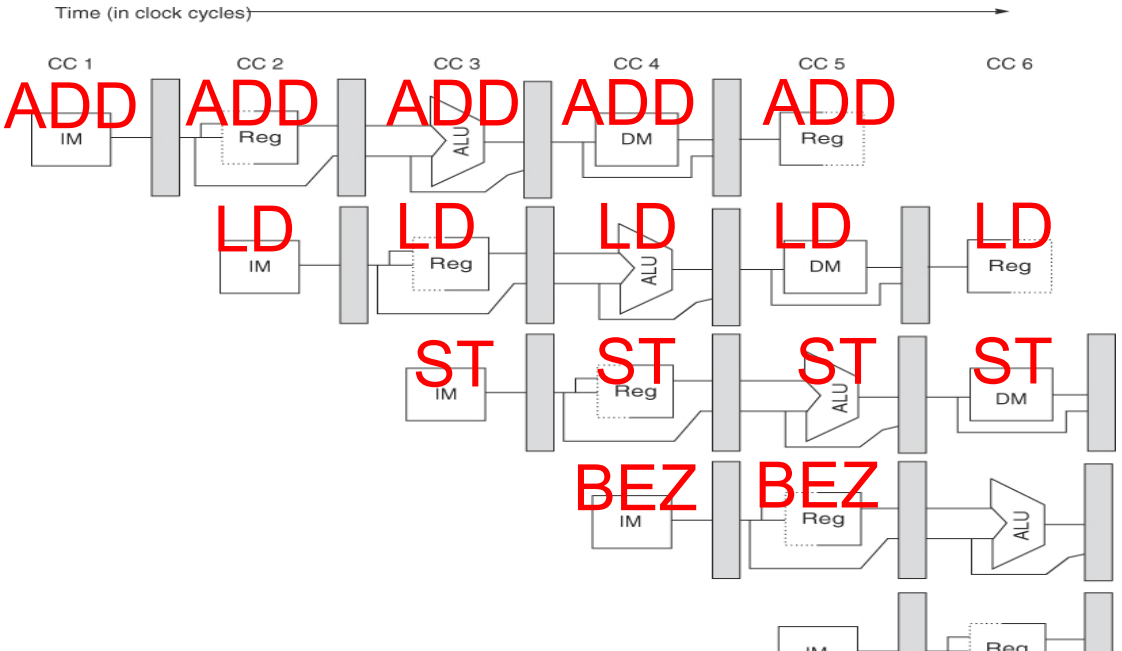
  ADD R3 ← R1, R2
  LD R7 ← 8[R6]
  ST R9 → 4[R8]
  BEZ R4, [R5]



Problem 4

- For the following code sequence, show how the instrs flow through the pipeline:

  ADD R3 ← R1, R2
  LD R7 ← 8[R6]
  ST R9 → 4[R8]
  BEZ R4, [R5]



Hazards

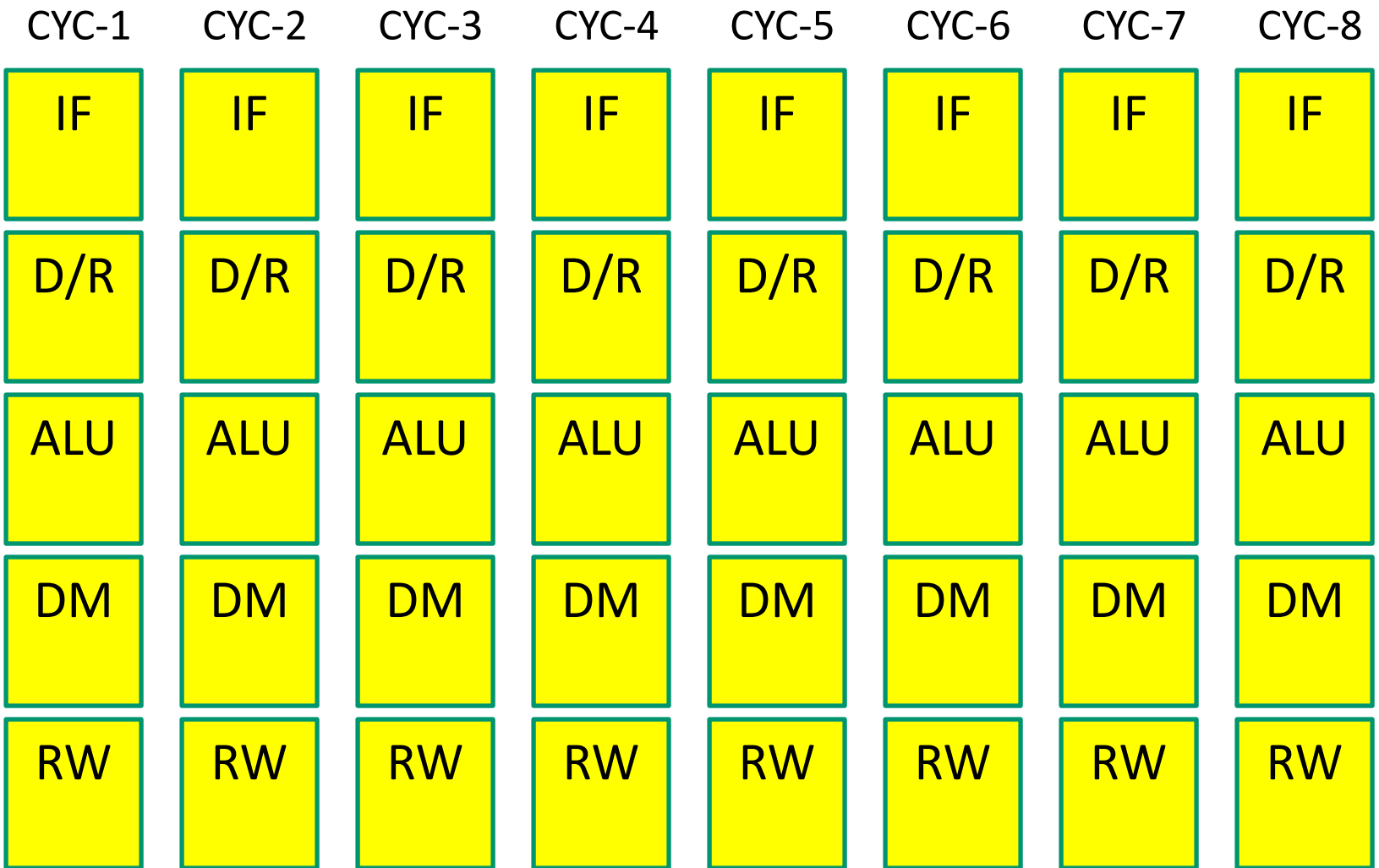
- Structural hazards: different instructions in different stages (or the same stage) conflicting for the same resource
- Data hazards: an instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction
- Control hazard: fetch cannot continue because it does 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)

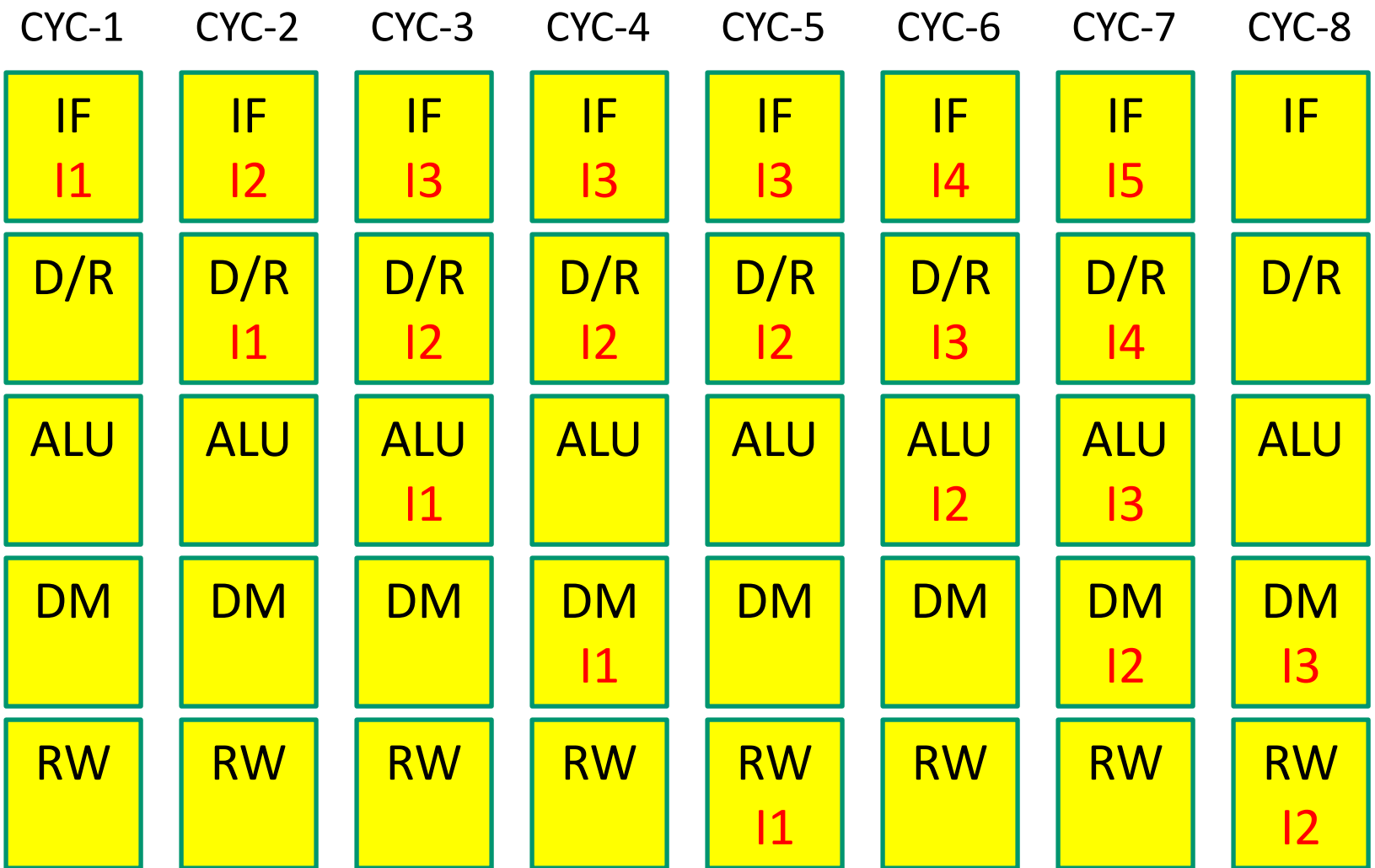
Problem 5

- Show the instruction occupying each stage in each cycle (no bypassing)
  if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R7+R8→R9

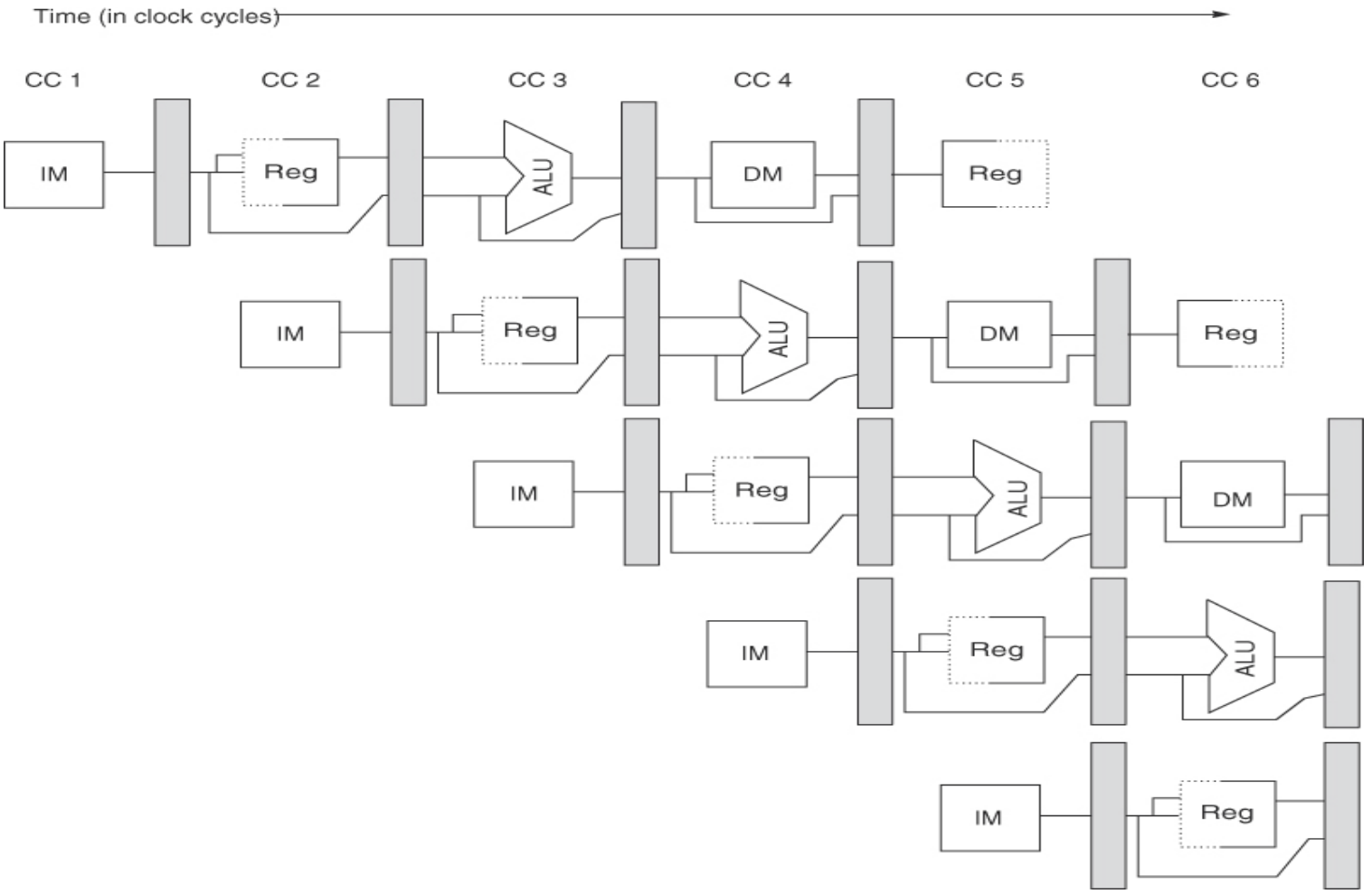


Problem 5

- Show the instruction occupying each stage in each cycle (no bypassing)
  if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R7+R8→R9



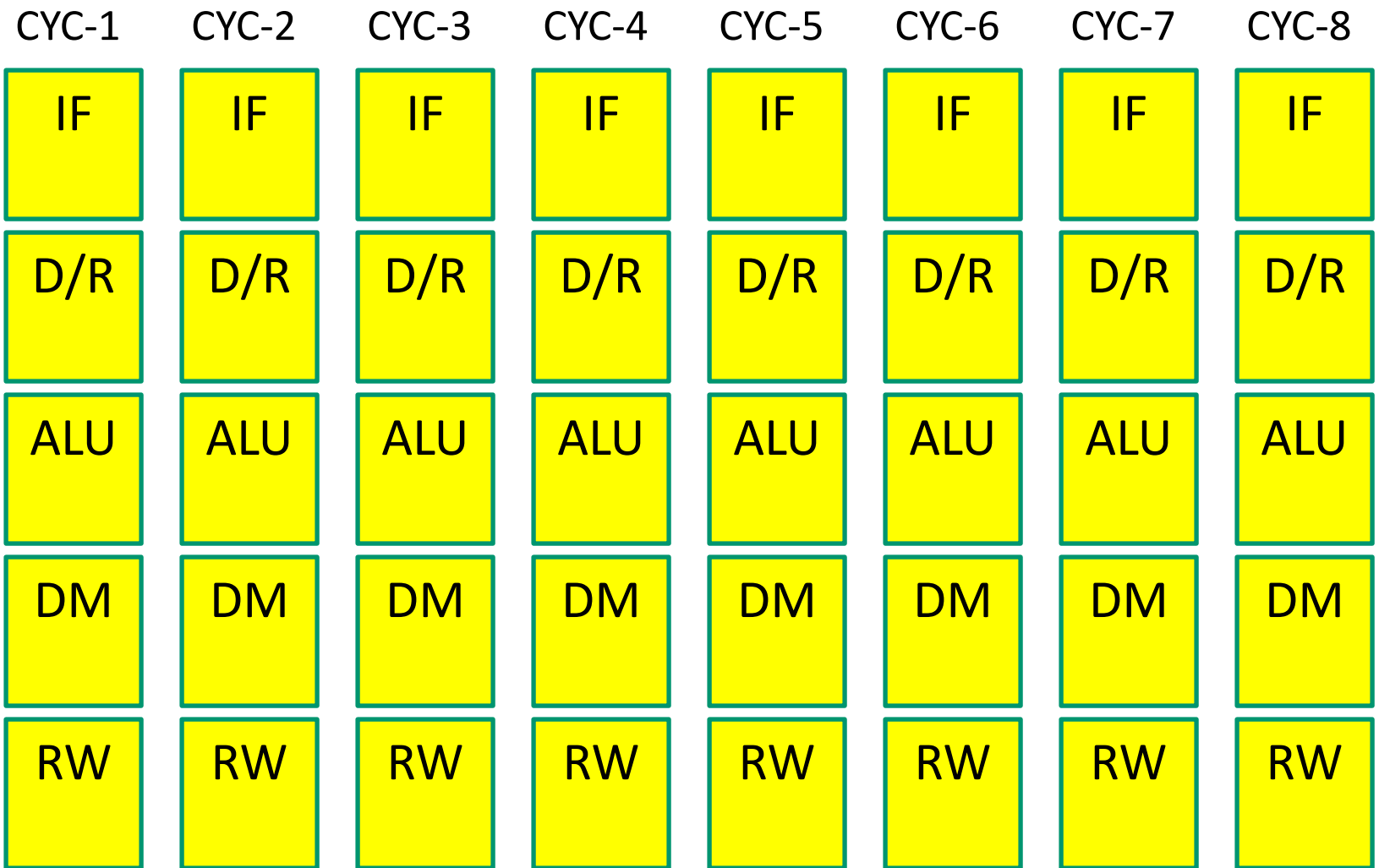
Bypassing: 5-Stage Pipeline



Source: H&P textbook

Problem 6

- Show the instruction occupying each stage in each cycle (with bypassing) if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R3+R8→R9. Identify the input latch for each input operand.



Problem 6

- Show the instruction occupying each stage in each cycle (with bypassing) if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R3+R8→R9. Identify the input latch for each input operand.