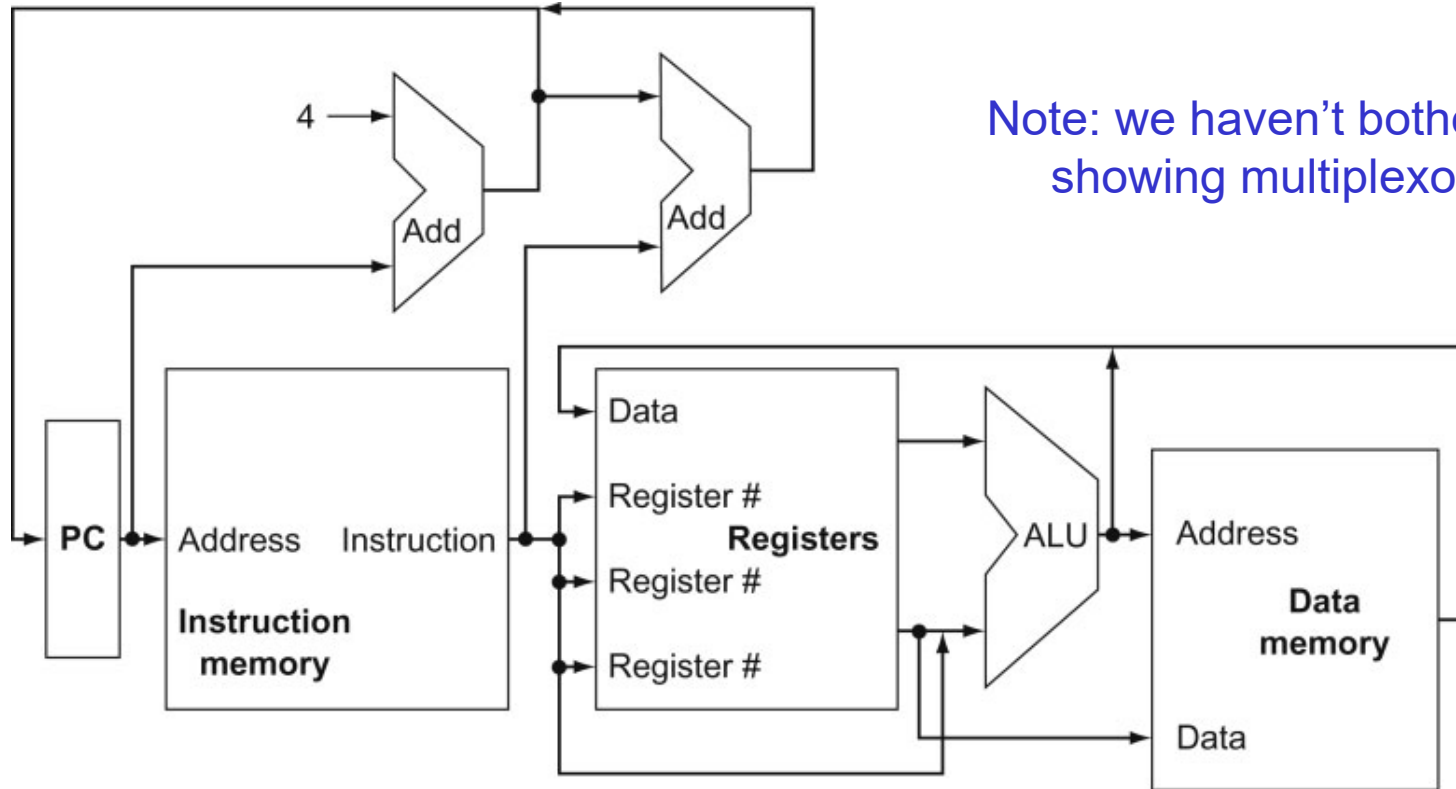


View from 30,000 Feet

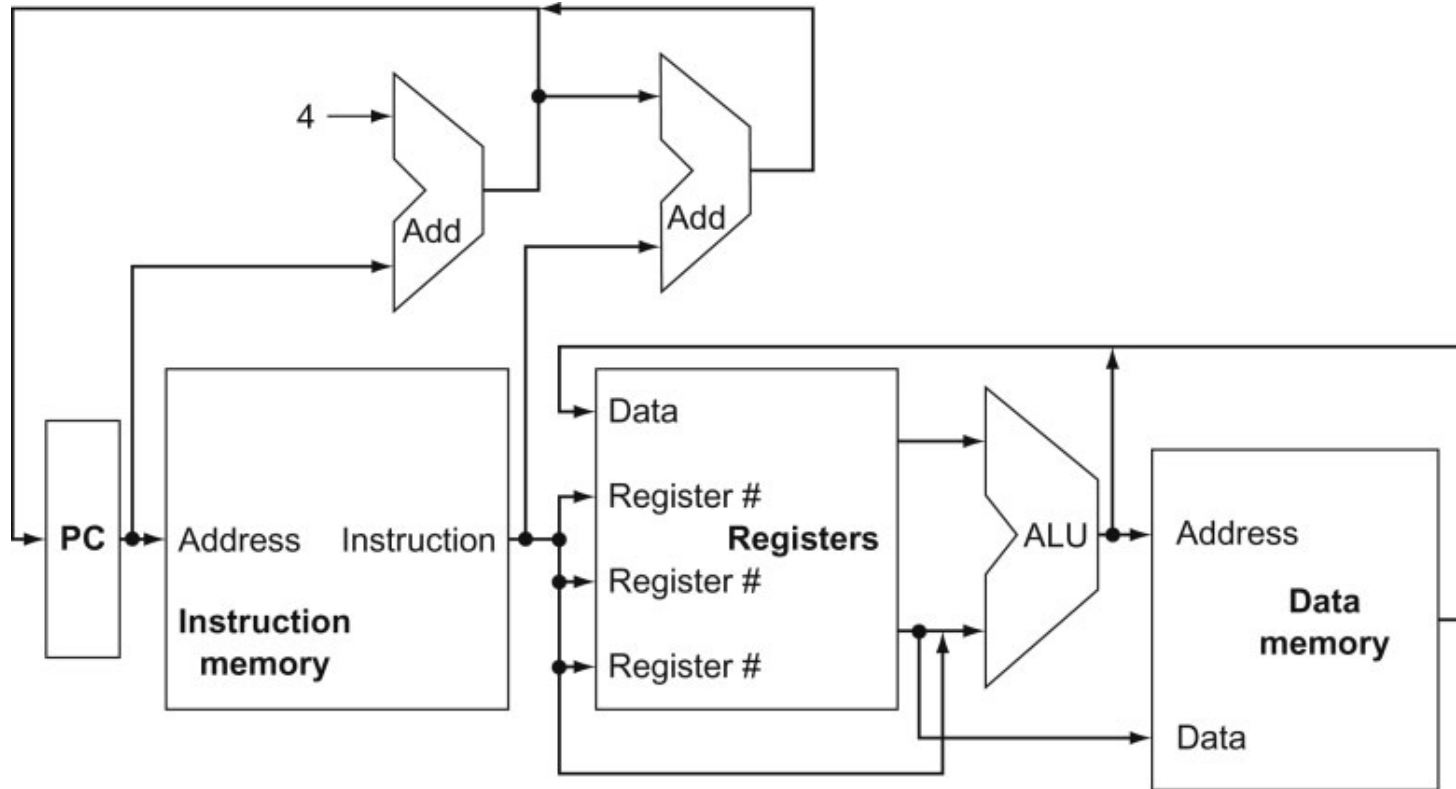


Note: we haven't bothered showing multiplexors

- 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

Source: H&P textbook

Clocking Methodology



Source: H&P textbook

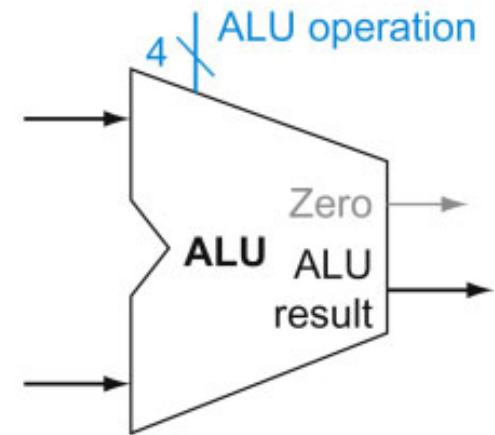
- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?
Keep in mind that the latched value remains there for an entire cycle

Implementing R-type Instructions

- Instructions of the form `add $t1, $t2, $t3`
- Explain the role of each signal



a. Registers

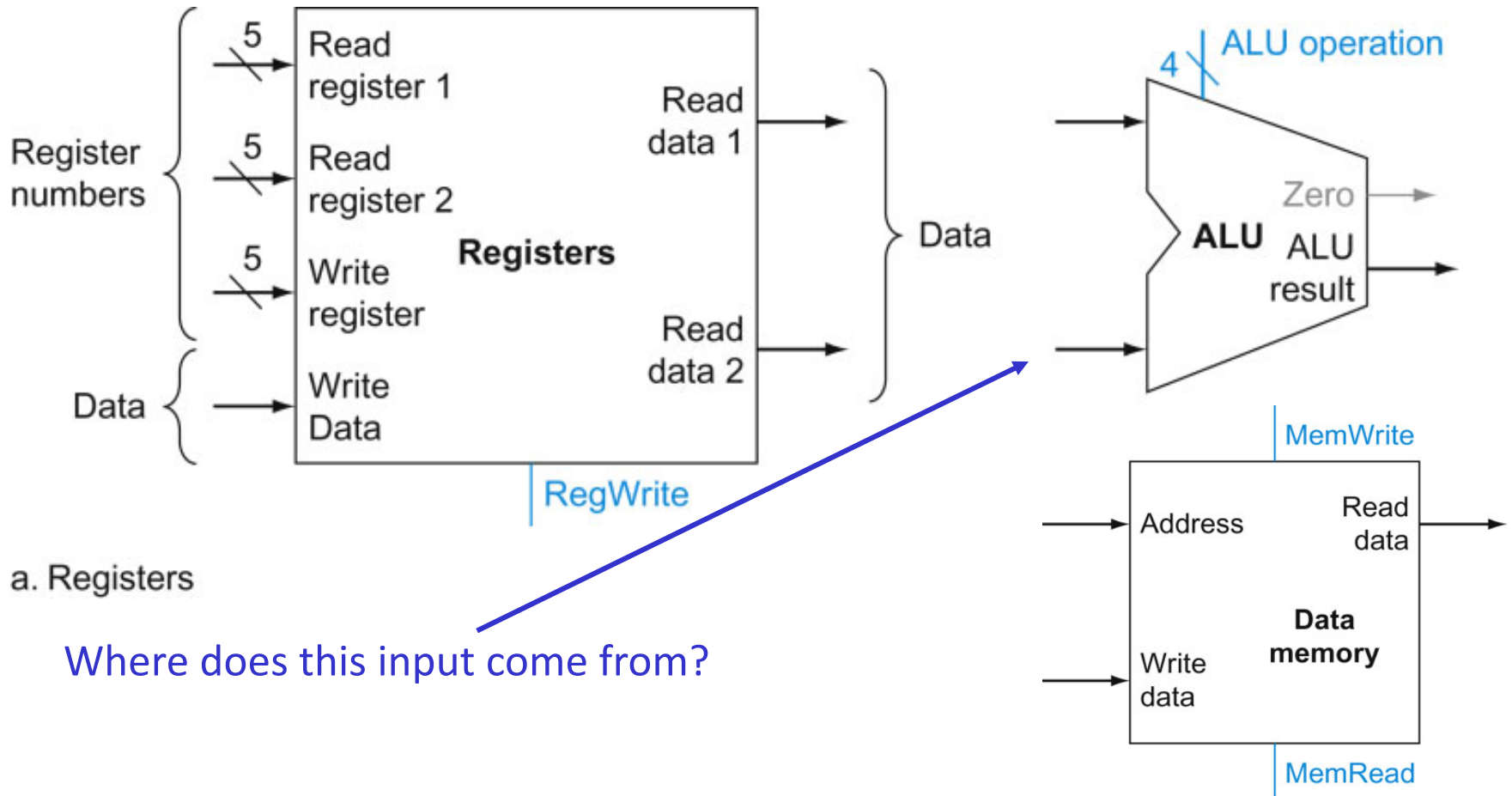


b. ALU

Source: H&P textbook

Implementing Loads/Stores

- Instructions of the form `lw $t1, 8($t2)` and `sw $t1, 8($t2)`



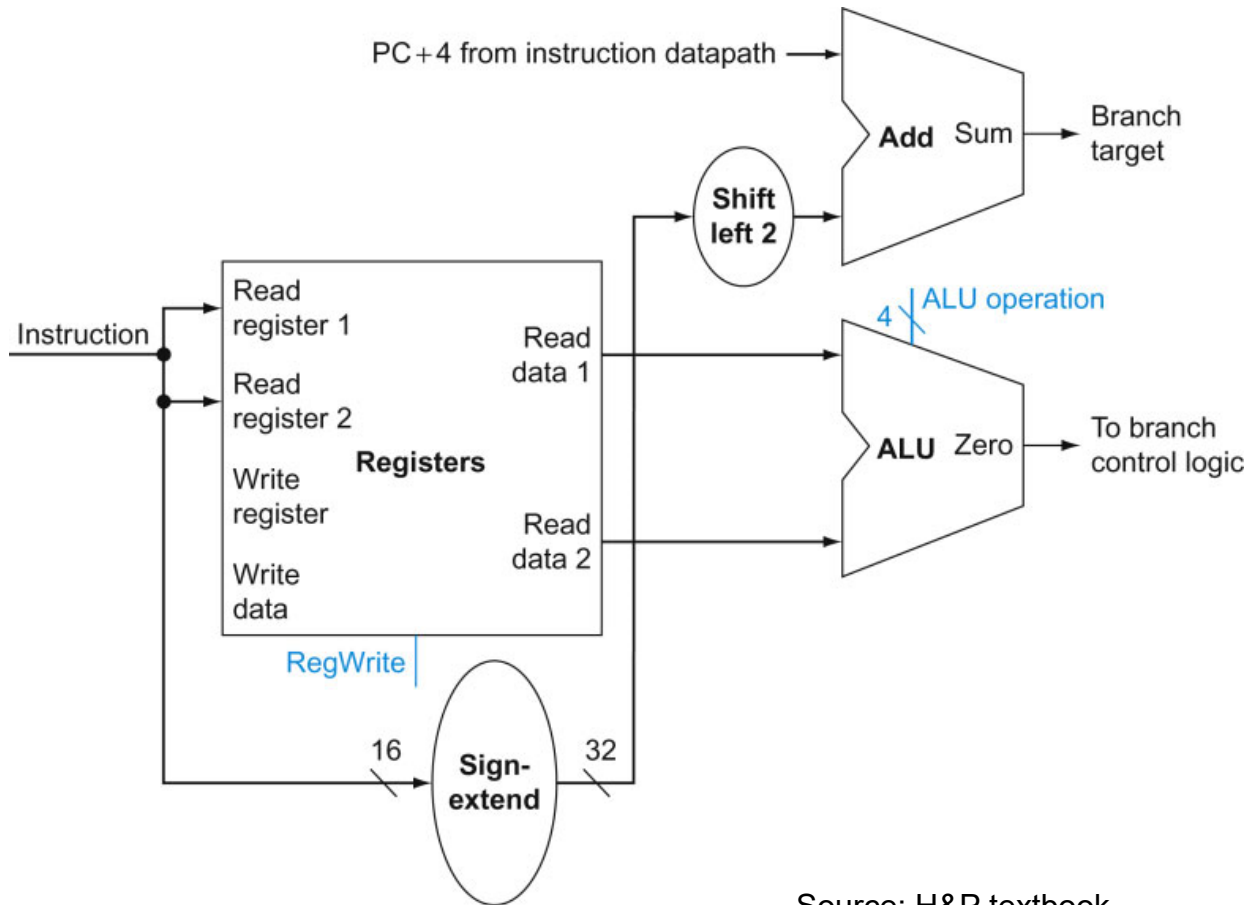
a. Registers

Where does this input come from?

a. Data memory unit Source: H&P textbook

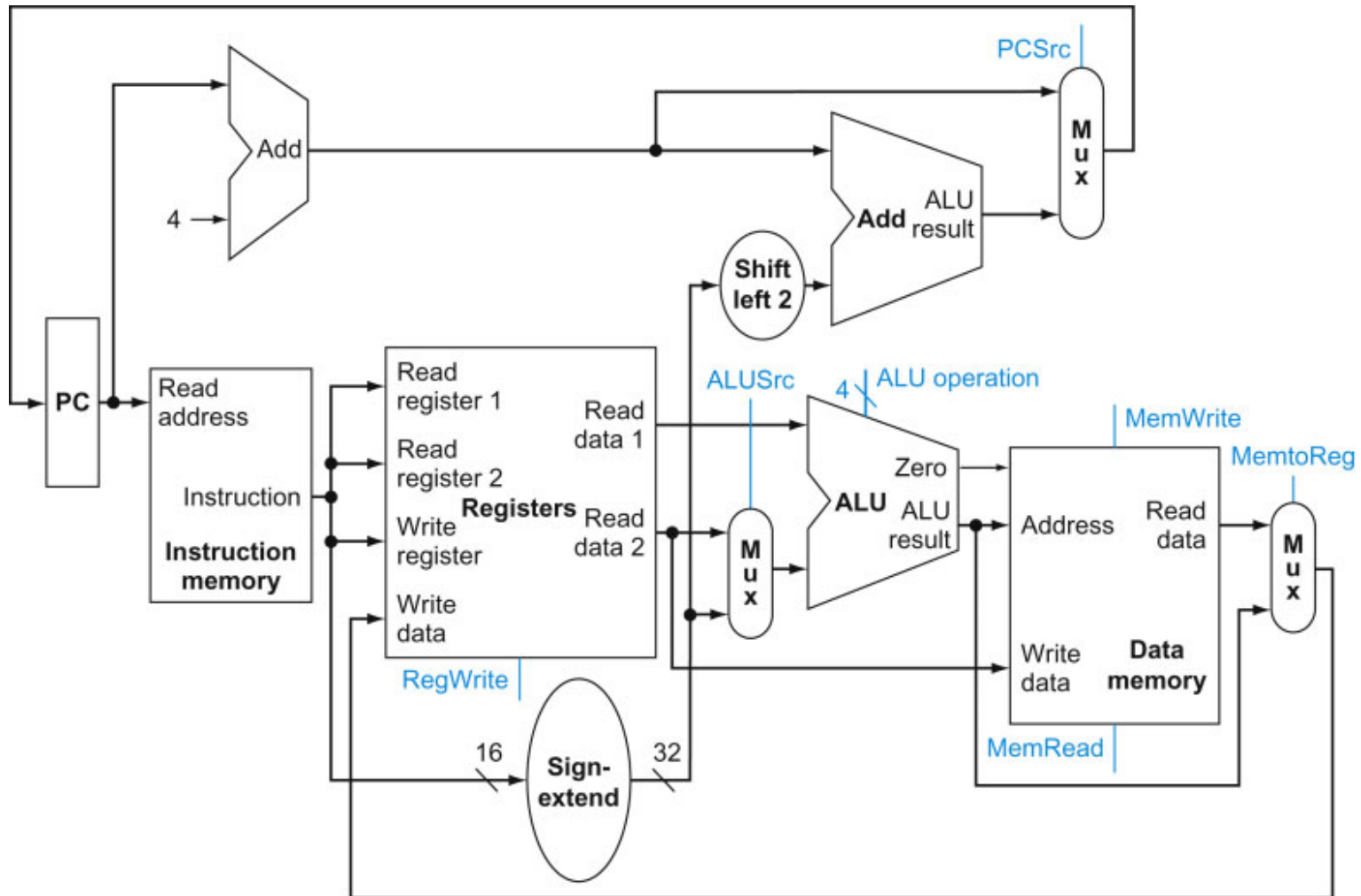
Implementing J-type Instructions

- Instructions of the form `beq $t1, $t2, offset`

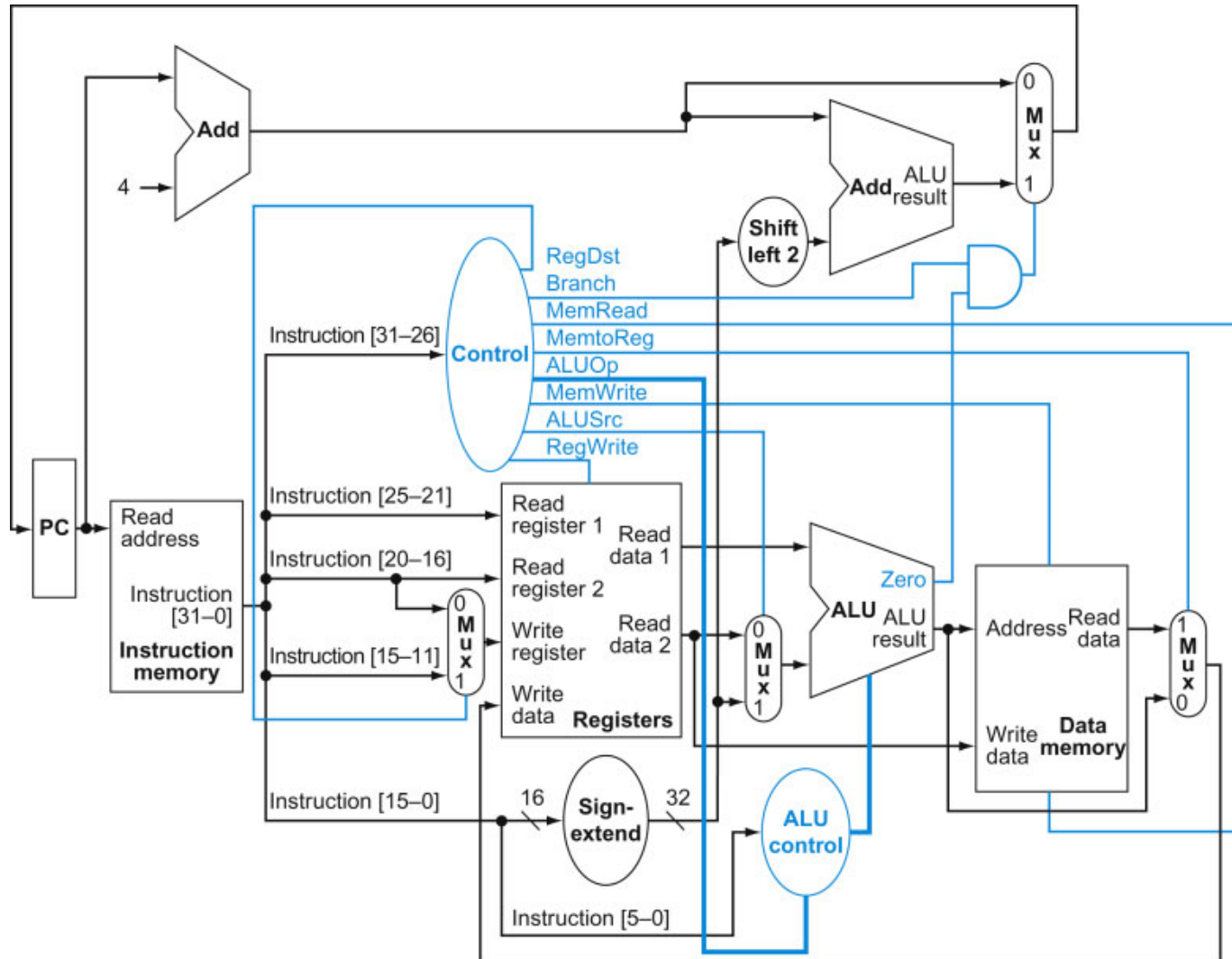


Source: H&P textbook

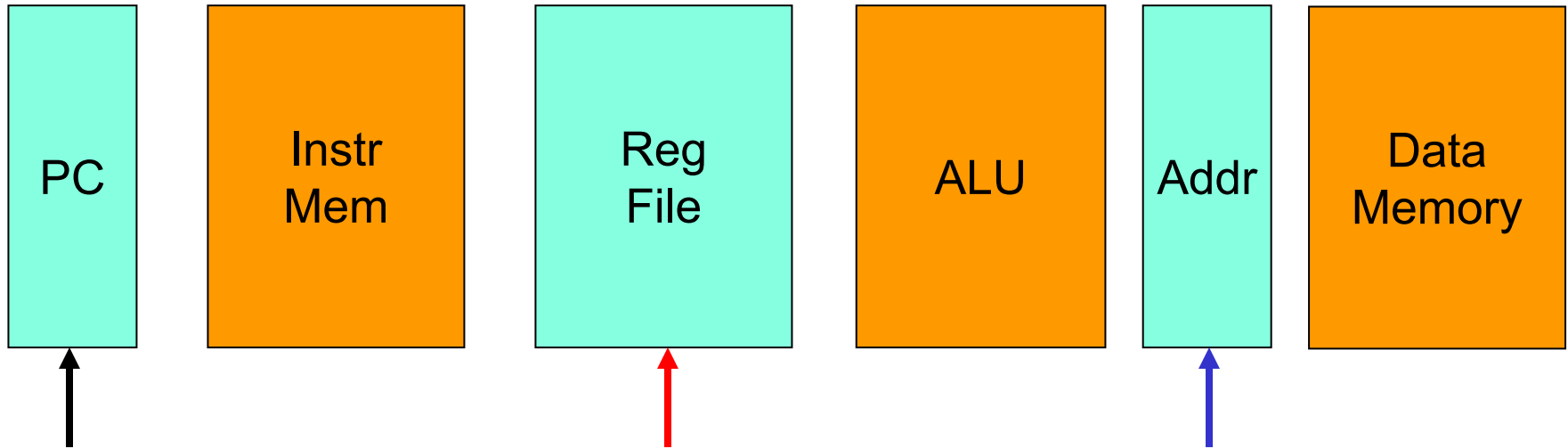
View from 10,000 Feet







View from 5,000 Feet



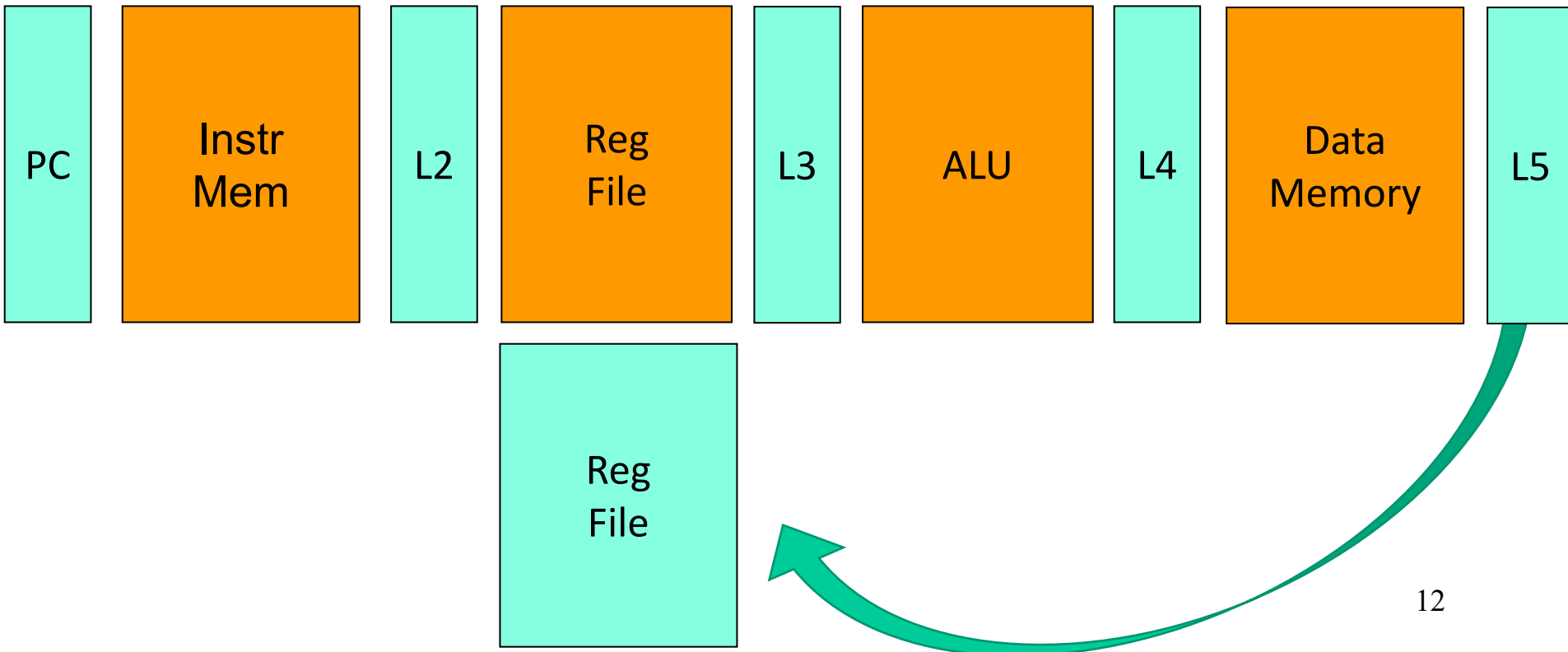
Latches and Clocks in a Single-Cycle Design



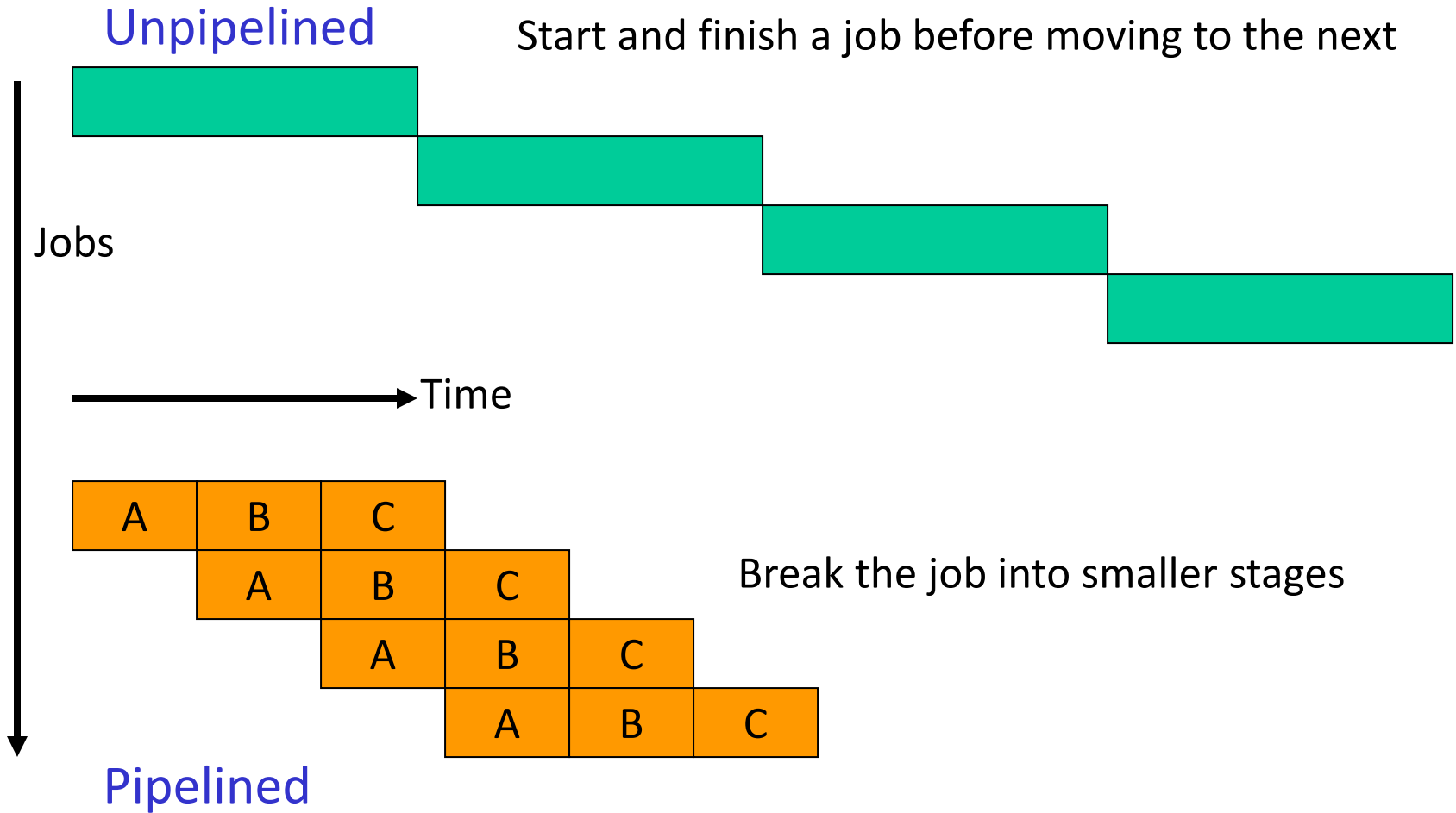
- The entire instruction executes in a single cycle
- Green blocks are latches
- At the rising edge, a new PC is recorded 
- At the rising edge, the result of the previous cycle is recorded 
- At the falling edge, the address of LW/SW is recorded so  we can access the data memory in the 2nd half of the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

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



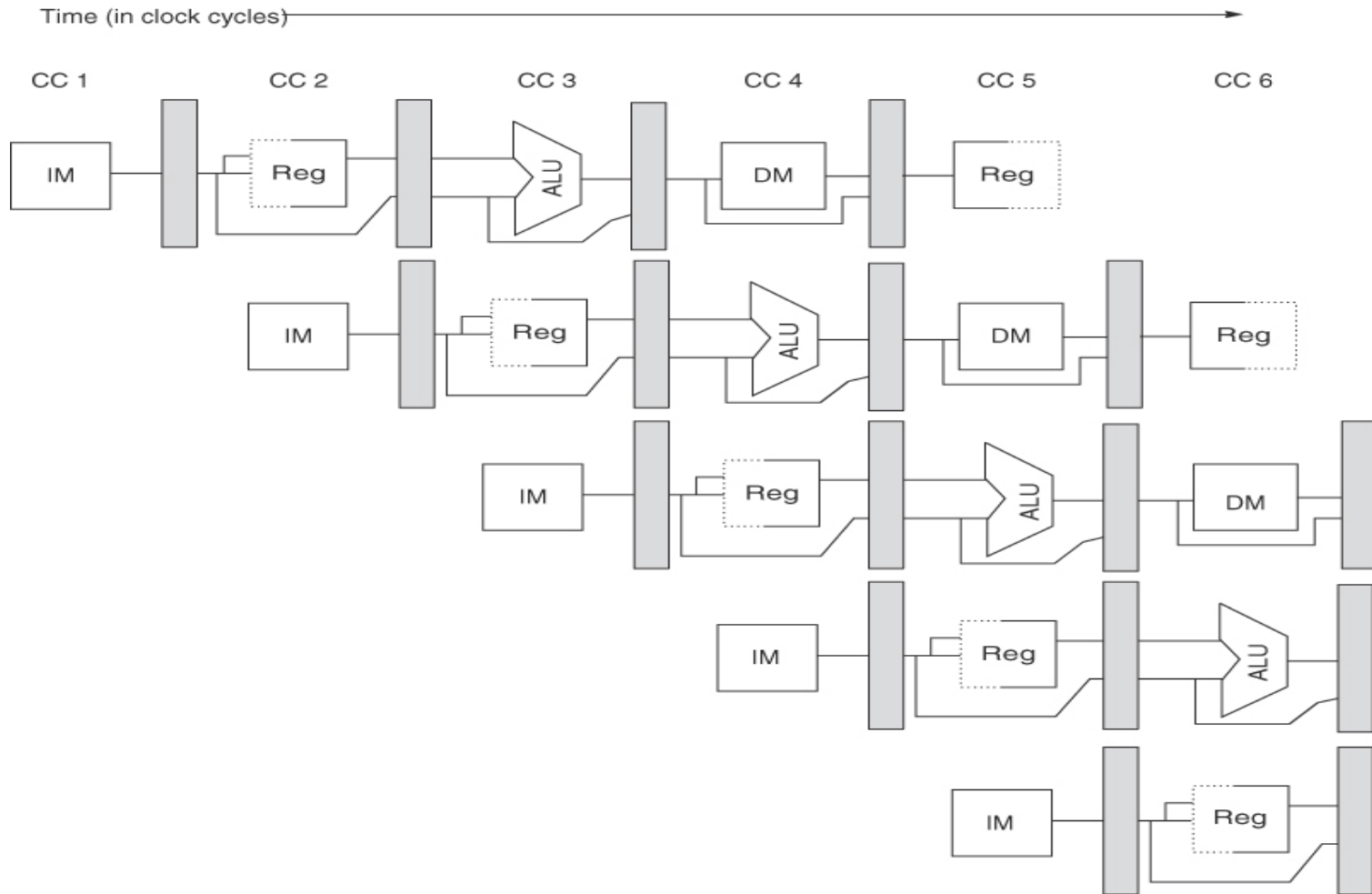
The Assembly Line



Performance Improvements?

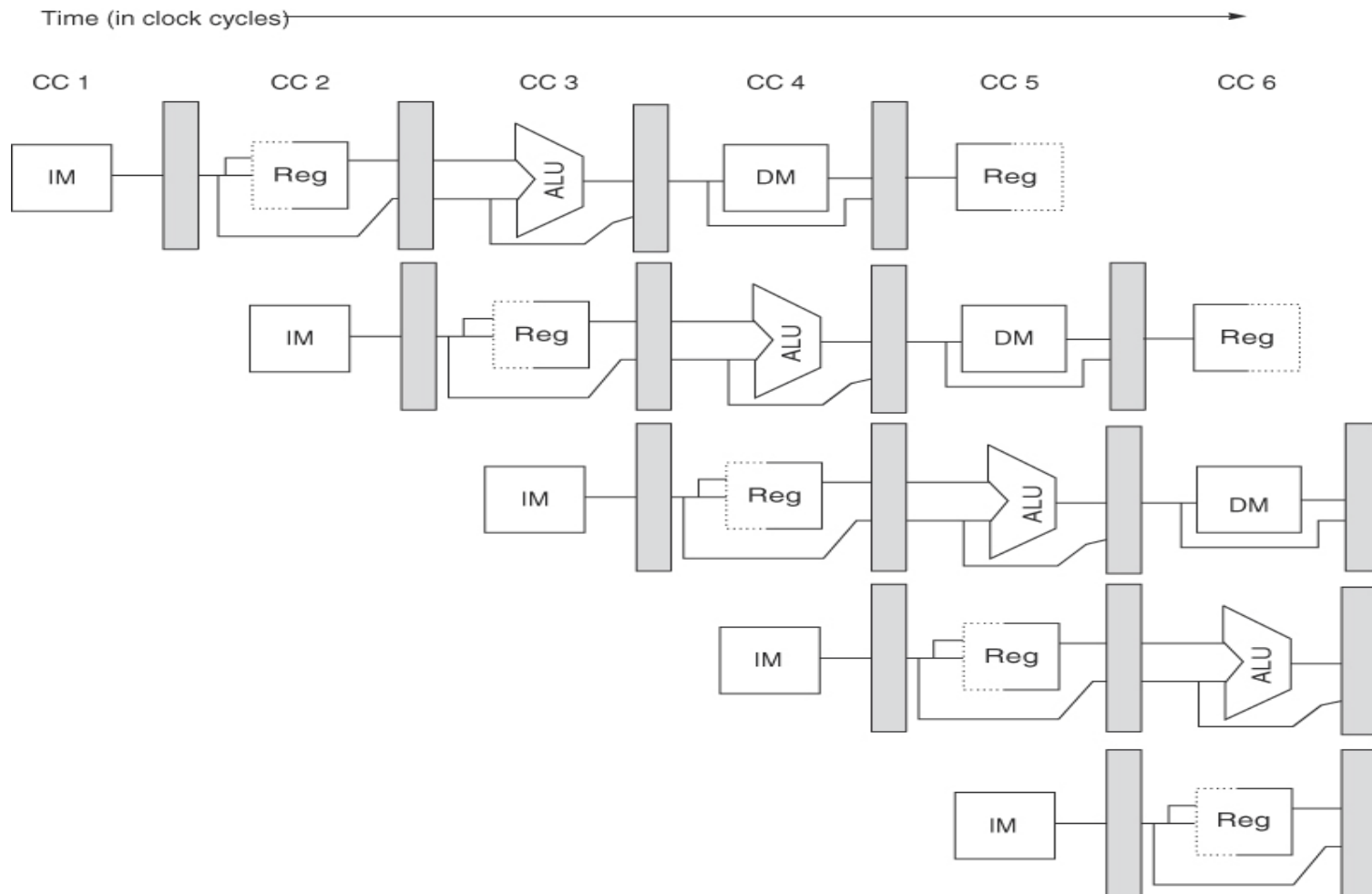
- Does it take longer to finish each individual job?
- Does it take shorter to finish a series of jobs?
- What assumptions were made while answering these questions?
- Is a 10-stage pipeline better than a 5-stage pipeline?

A 5-Stage Pipeline



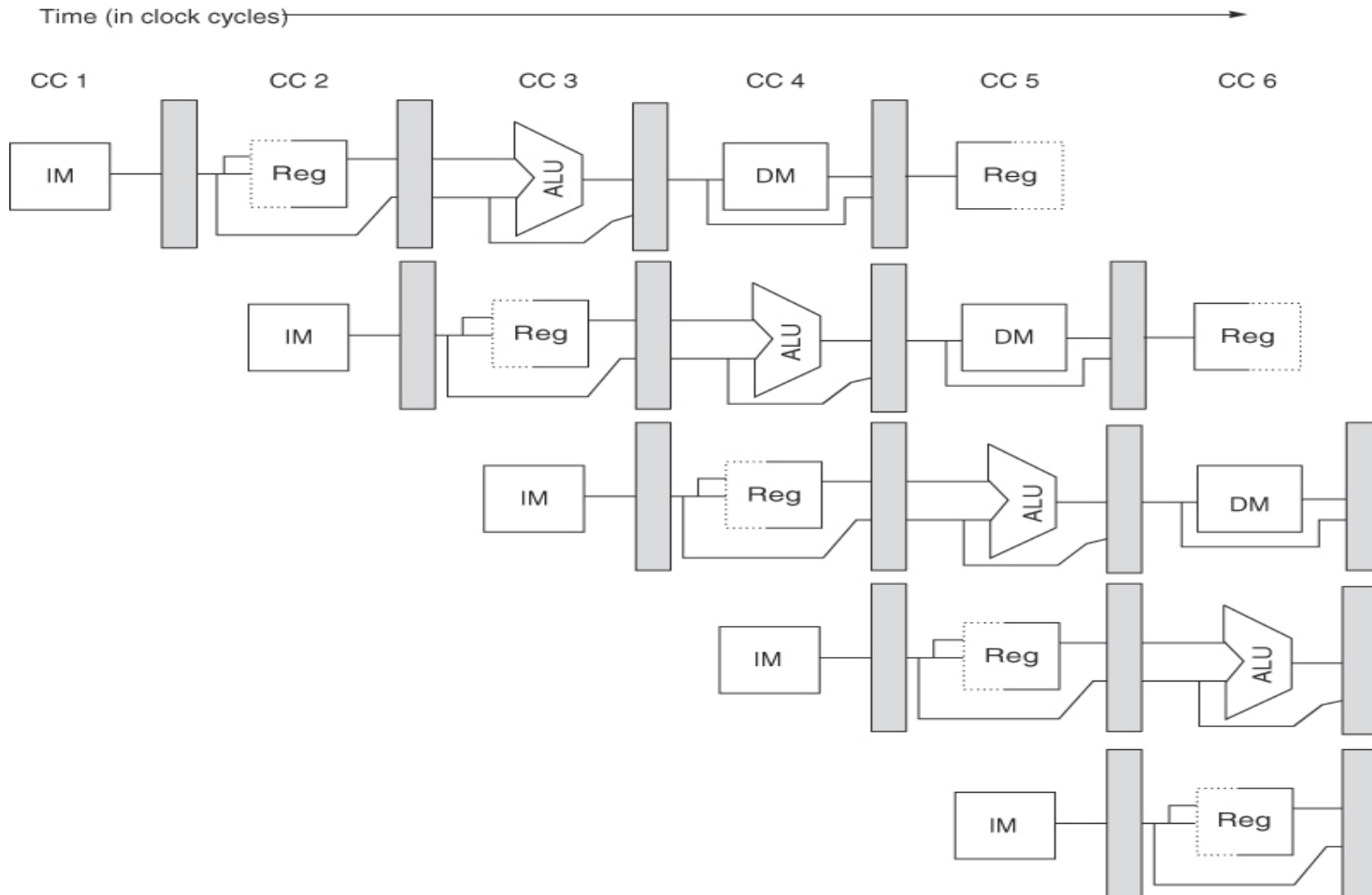
A 5-Stage Pipeline

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



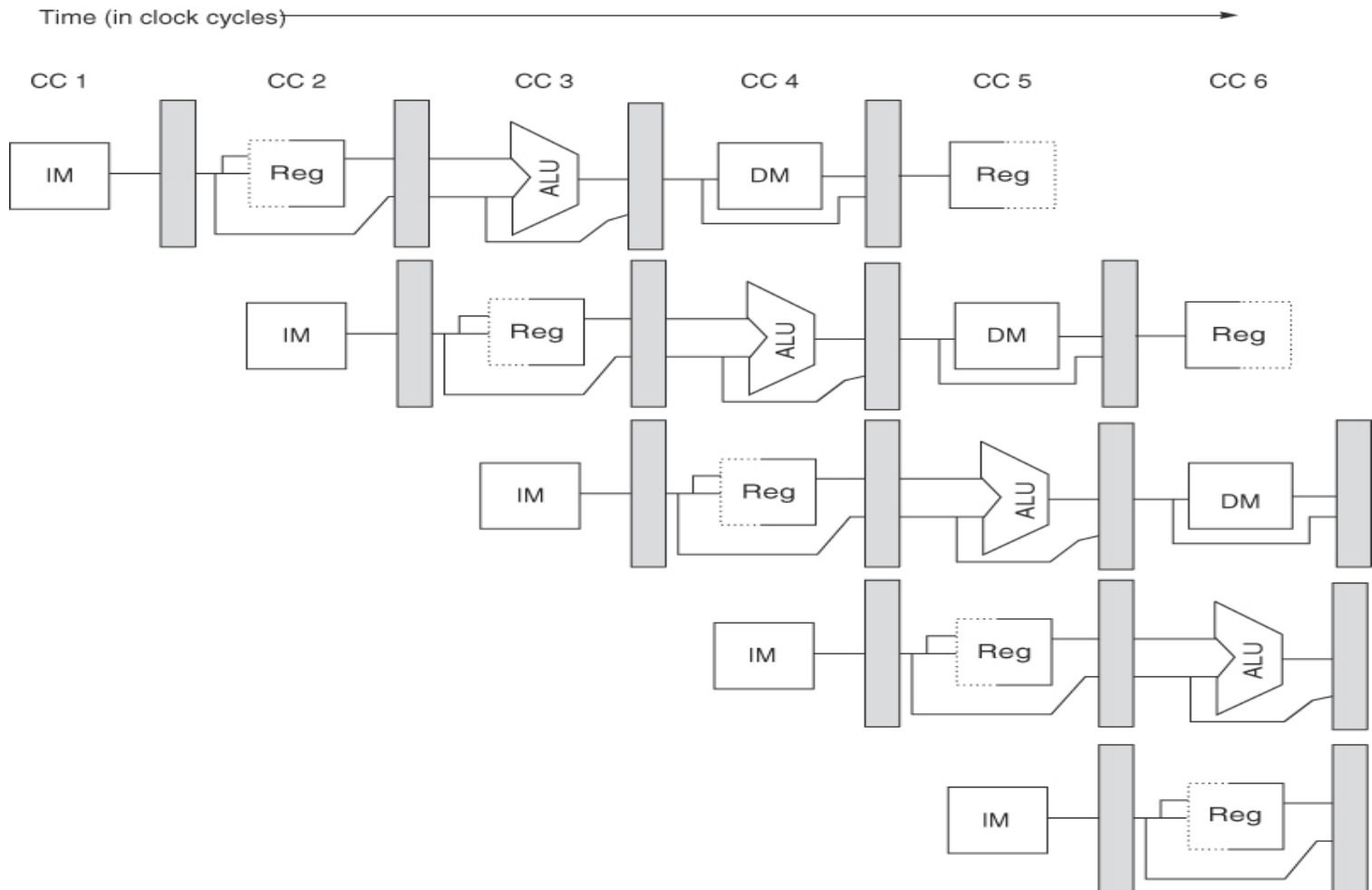
A 5-Stage Pipeline

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



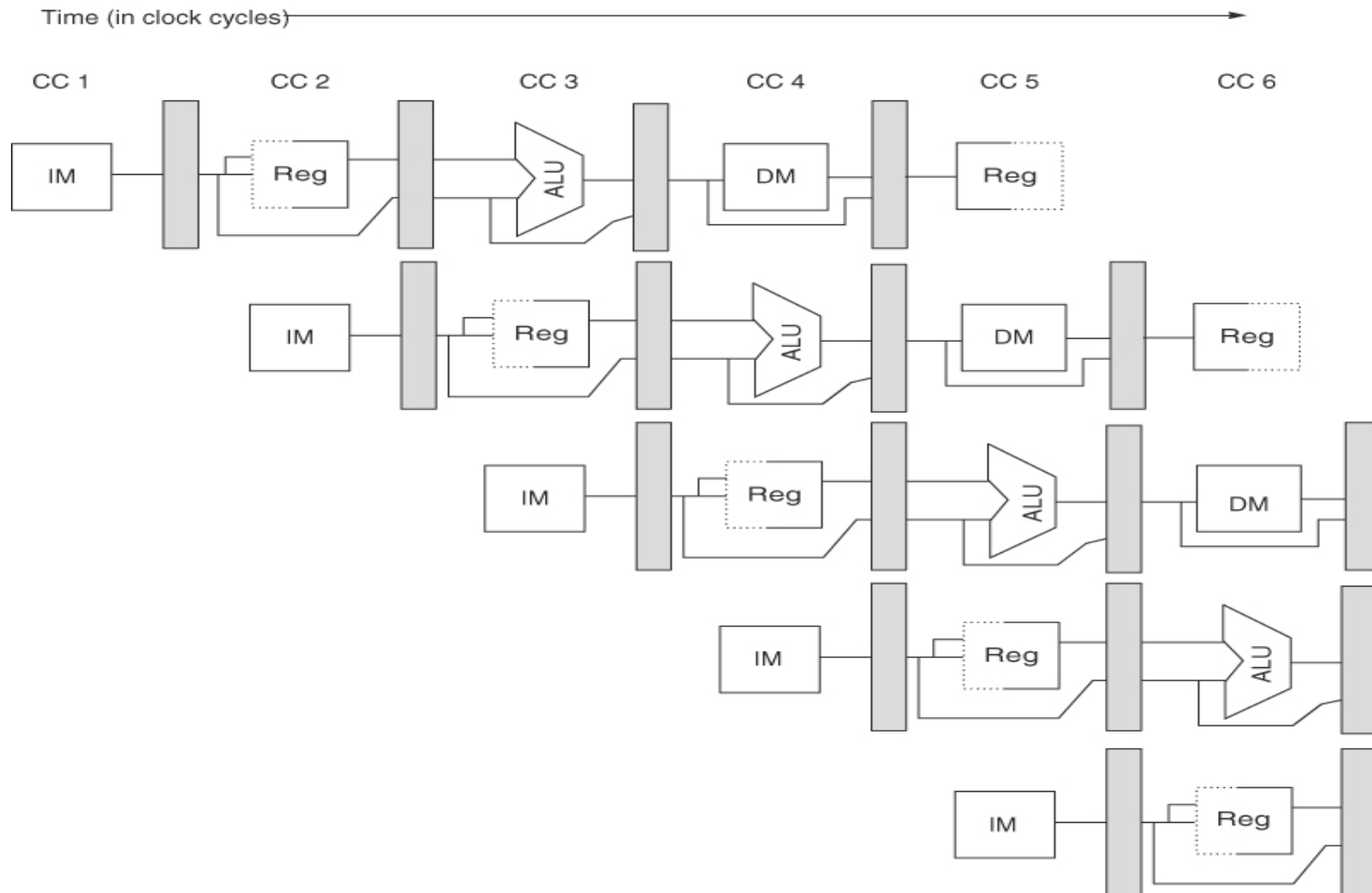
A 5-Stage Pipeline

ALU computation, effective address computation for load/store



A 5-Stage Pipeline

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



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	--

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?
 - No dependences between instructions
 - Easy to partition circuits into uniform pipeline stages
 - No latch overhead
- 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

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

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, add more register ports)

Data Hazards

- An instruction *produces* a value in a given pipeline stage
- A subsequent instruction *consumes* that value in a pipeline stage
- The consumer may have to be delayed so that the time of consumption is later than the time of production

Example 1 – No Bypassing

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

CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF	IF	IF	IF	IF	IF	IF	IF
D/R	D/R	D/R	D/R	D/R	D/R	D/R	D/R
ALU	ALU	ALU	ALU	ALU	ALU	ALU	ALU
DM	DM	DM	DM	DM	DM	DM	DM
RW	RW	RW	RW	RW	RW	RW	RW

Example 1 – No Bypassing

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

CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF I1	IF I2	IF I3	IF I3	IF I3	IF I4	IF I5	IF
D/R	D/R I1	D/R I2	D/R I2	D/R I2	D/R I3	D/R I4	D/R
ALU	ALU	ALU I1	ALU	ALU	ALU I2	ALU I3	ALU
DM	DM	DM	DM I1	DM	DM	DM I2	DM I3
RW	RW	RW	RW	RW I1	RW	RW	RW I2

Example 2 – Bypassing

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

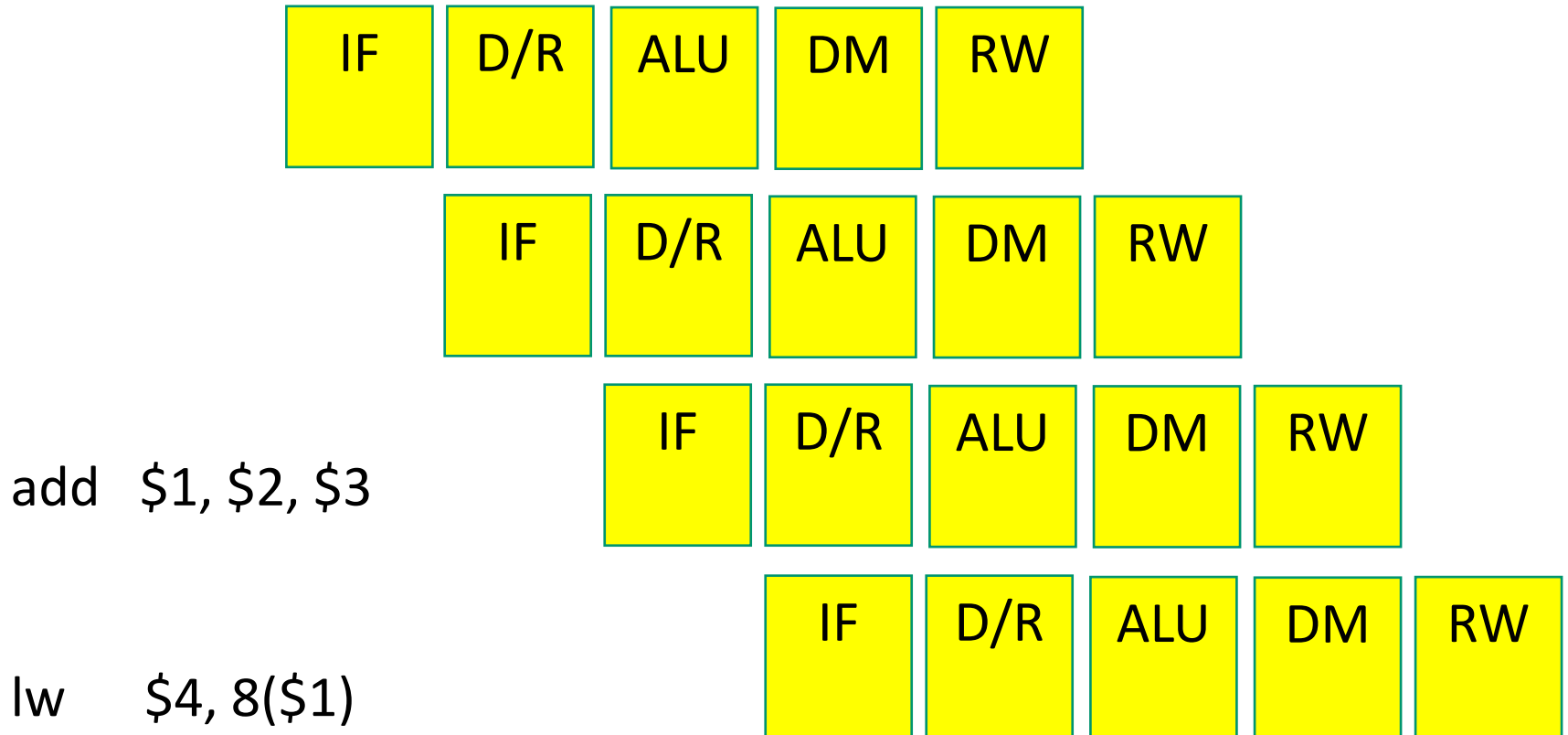
CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF	IF	IF	IF	IF	IF	IF	IF
D/R	D/R	D/R	D/R	D/R	D/R	D/R	D/R
ALU	ALU	ALU	ALU	ALU	ALU	ALU	ALU
DM	DM	DM	DM	DM	DM	DM	DM
RW	RW	RW	RW	RW	RW	RW	RW

Example 2 – Bypassing

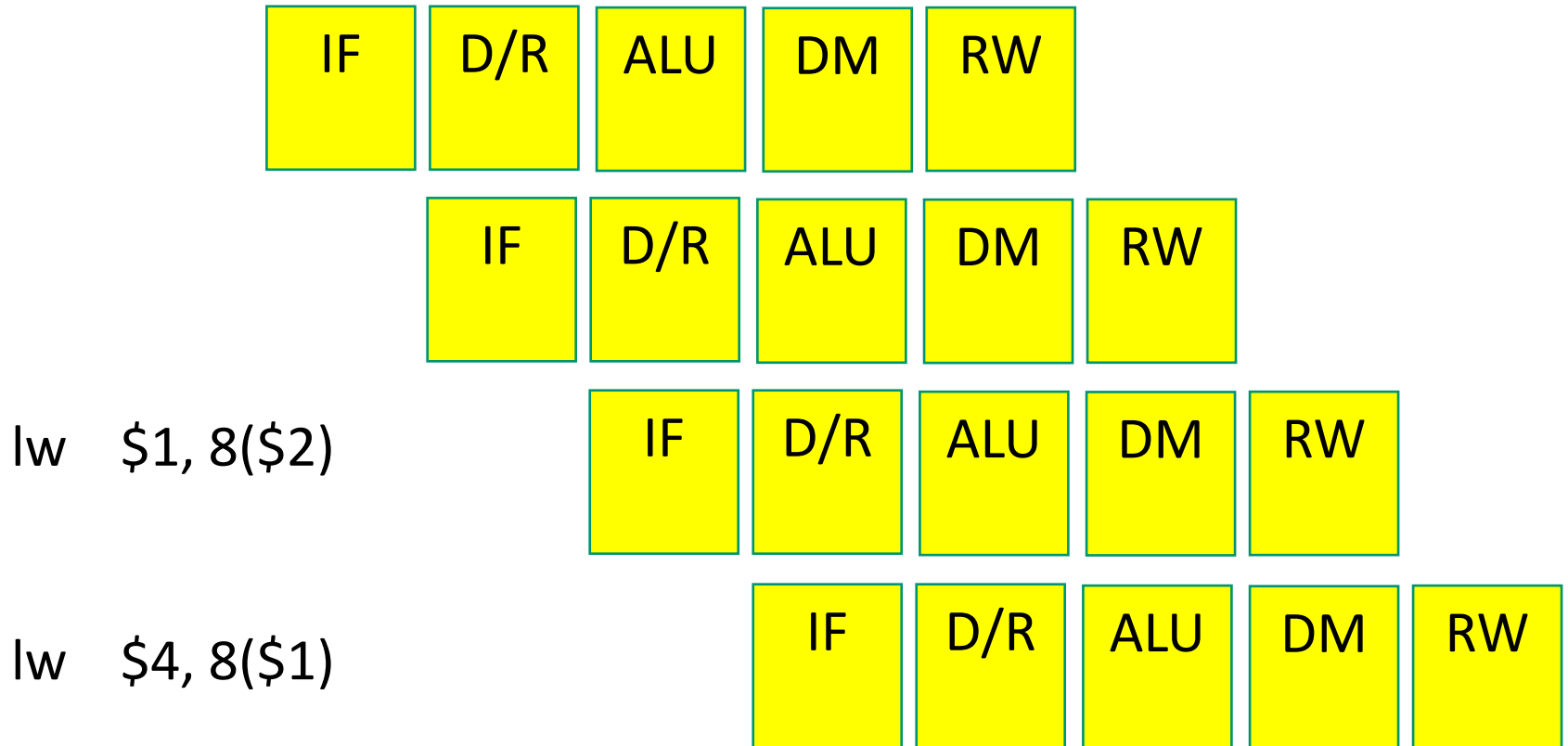
- Show the instruction occupying each stage in each cycle (with bypassing) if I1 is $R1+R2 \rightarrow R3$ and I2 is $R3+R4 \rightarrow R5$ and I3 is $R3+R8 \rightarrow R9$. Identify the input latch for each input operand.

CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF I1	IF I2	IF I3	IF I4	IF I5	IF	IF	IF
D/R	D/R I1	D/R I2	D/R I3	D/R I4	D/R	D/R	D/R
ALU	ALU	ALU I3 I3	ALU I4 I3	ALU I5 I3	ALU	ALU	ALU
DM	DM	DM	DM I1	DM I2	DM I3	DM	DM
RW	RW	RW	RW	RW I1	RW I2	RW I3	RW

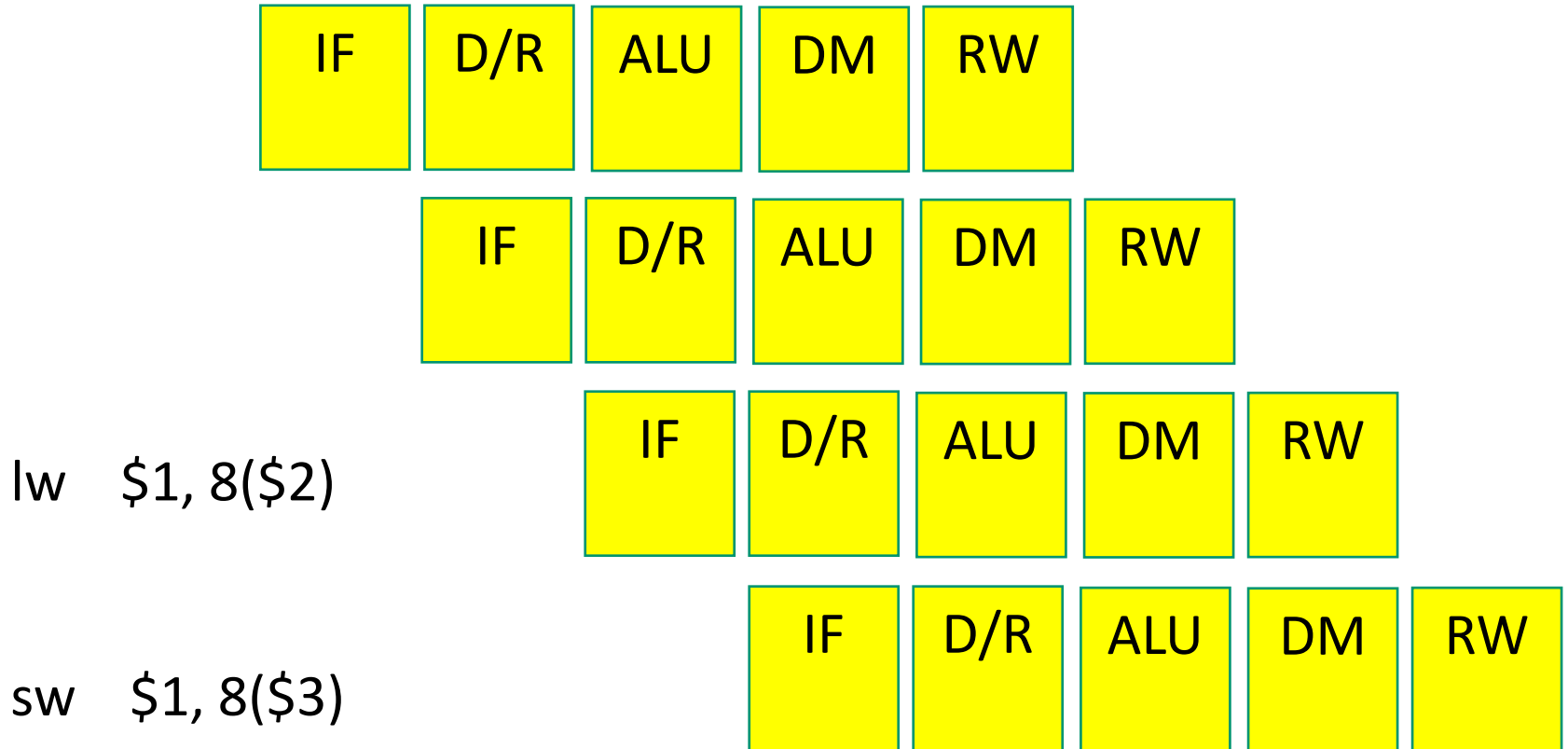
Problem 1



Problem 2

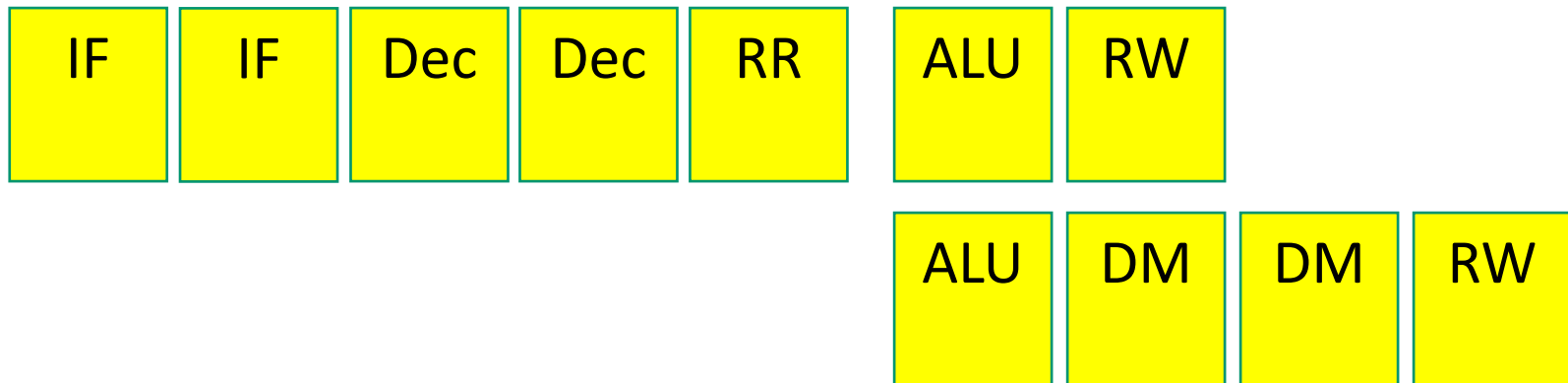


Problem 3



Problem 4

A 7 or 9 stage pipeline, RR and RW take an entire stage

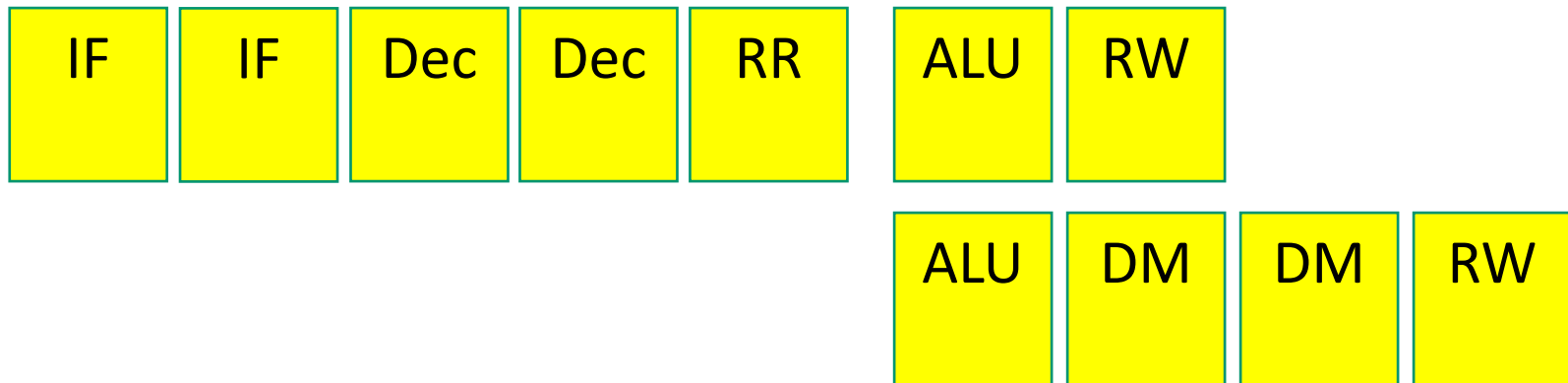


lw \$1, 8(\$2)

add \$4, \$1, \$3

Problem 4

A 7 or 9 stage pipeline, RR and RW take an entire stage



`lw $1, 8($2)`

`add $4, $1, $3`

Problem 4

Without bypassing: 4 stalls

IF:IF:DE:DE:RR:AL:DM:DM:RW

IF: IF :DE:DE:DE:DE: DE :DE:RR:AL:RW

With bypassing: 2 stalls

IF:IF:DE:DE:RR:AL:DM:DM:RW

IF: IF :DE:DE:DE:DE: RR :AL:RW

