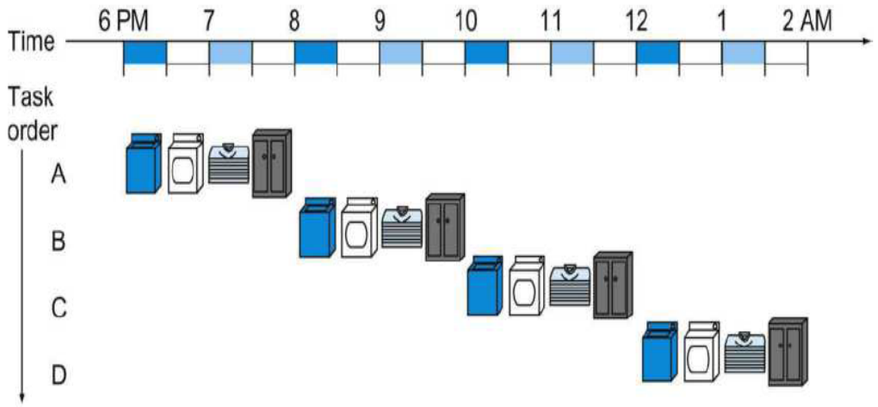
Computer Architecture
Chapter 4

The Processor – 2

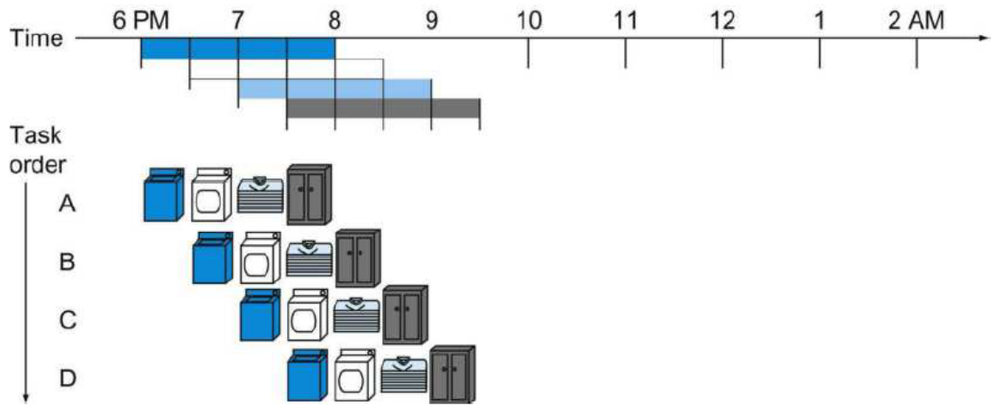
Joonho Kong
School of EE, KNU

Overview of Pipelining

What is more efficient?



Idle resources



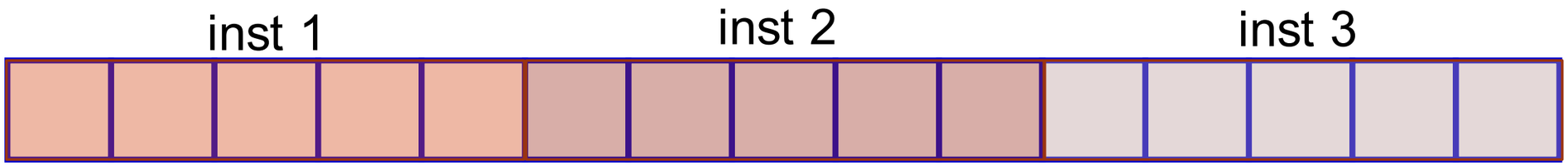
No idle resources

The sequence of washing clothes
1. Put clothes in the washer
2. Put clothes in the dryer
3. Fold clothes
4. Put clothes in the closet

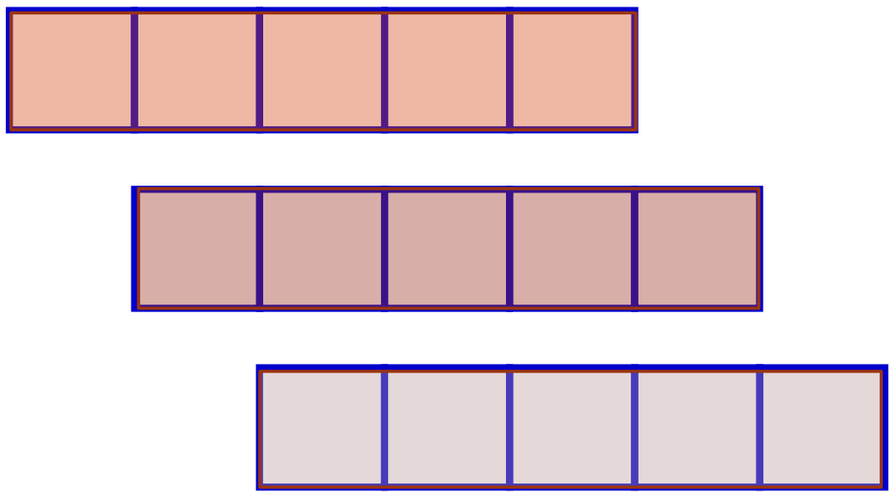
Figure is from “Computer Organization and Design”

Overview of Pipelining

- Single-cycle implementation



- Pipelined Execution



Overview of Pipelining

The Sequence of Five-Stage Pipeline

1. Fetch instruction from memory (IF)
2. Decode the instruction and read registers (ID)
3. Execute the operation or calculate an address (EX)
4. Access data memory (MEM)
5. Write back the result into a register (WB)

	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

Overview of Pipelining

Example

Assume that the operation latencies for the major functional units in this example are listed in below table. Let's compare the pipelined execution time and non-pipelined execution time when 3 instructions are executed and 1,000,000 more instructions are executed

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load doubleword (ld)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store doubleword (sd)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

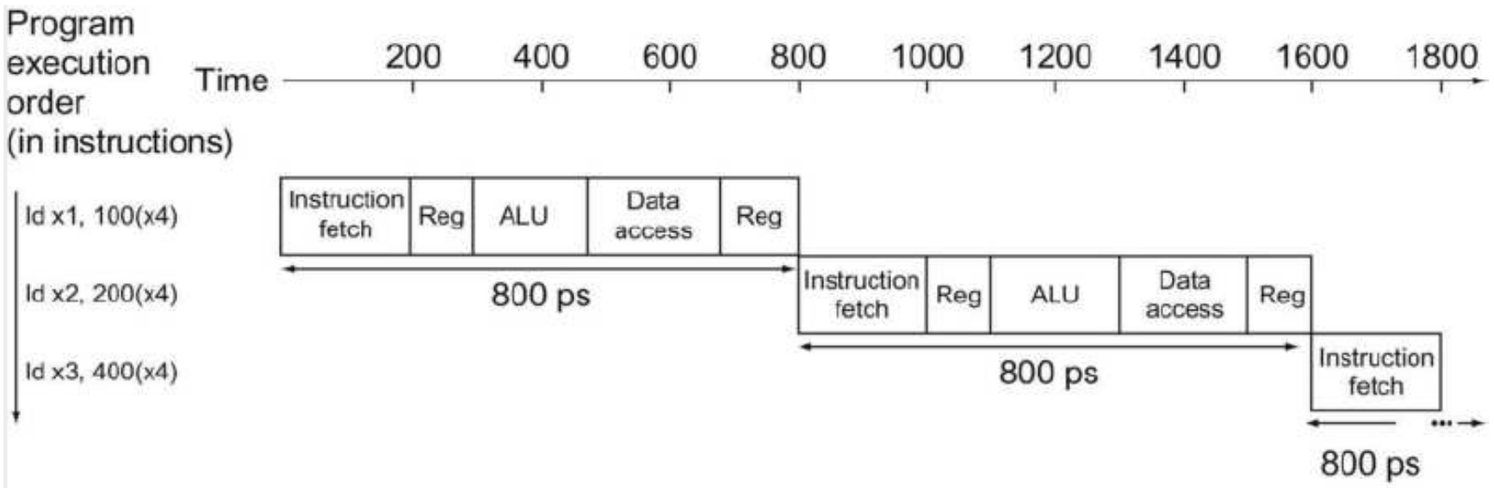
Figure is from "Computer Organization and Design"

Overview of Pipelining

Solution

Let's compare the pipelined execution and non-pipelined execution
In the single-cycle model, every instruction takes exactly one clock cycle

Single cycle, non-pipelined execution



pipelined execution

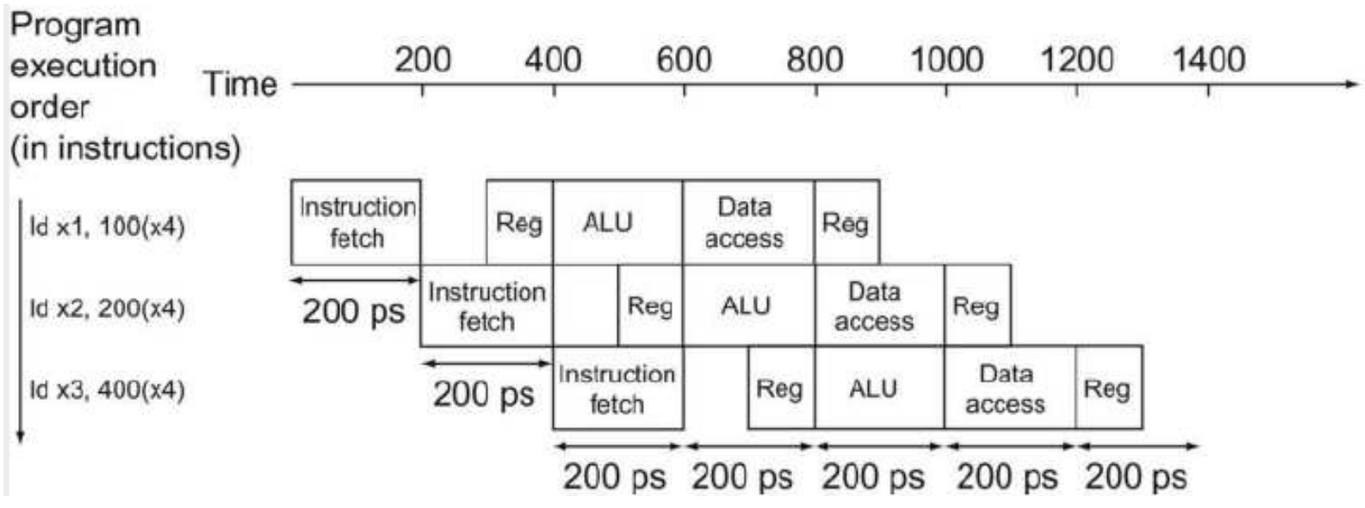


Figure is from "Computer Organization and Design"

Overview of Pipelining

Solution

When 3 ld instructions are executed, each total execution time is:
pipelined execution time = 1400 ps
non-pipelined execution time = 2400 ps
→ Speedup is ~1.7X

How about when 1,000,000 instruction are added?
pipelined execution time = 1,000,000 * 200 ps + 1,400 ps = 200,001,400 ps
non-pipelined execution time = 1,000,000 * 800 ps * 2,400 ps = 800,002,400 ps
→ Speedup is ~4X

※ Real programs execute billions of instructions

Overview of Pipelining

Designing Instruction Sets for Pipelining

Characteristics the design of the RISC-V instruction set for pipeline:
1. All RISC-V instructions are the same length
→ Much easier to fetch instructions and to decode them

2. The source and destination register fields are located in the same place in each instruction
→ Design principle 1: Simplicity favors regularity

3. Memory operands only appear in loads or stores in RISC-V
→ we can calculate the memory address in the execute stage, and then access memory

Overview of Pipelining

Pipeline Hazards

Hazard:
The next instructions cannot execute in the following clock cycle

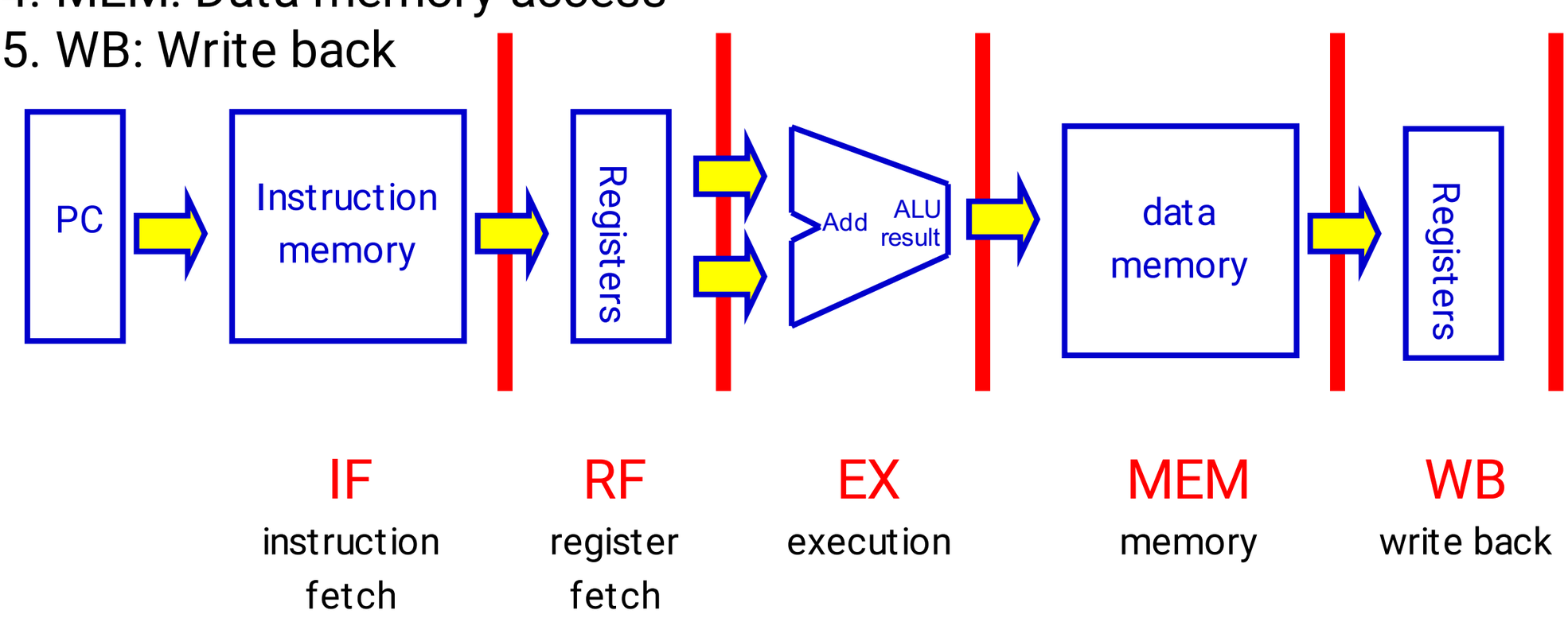
1. Data hazard:
   - It is because of the data dependencies
   - We should reduce any stall (or bubble) in the pipeline caused by data dependencies
2. Structural hazard:
   - A hardware component cannot accommodate two instructions at the same time
   - It is because of the lack of hardware resource
3. Control hazard:
   - It occurs with conditional branches
   - The processor does not know the result of the branch until the values are compared

We will cover a detail of pipeline hazards in the later part of this chapter

Pipelined Datapath and Control

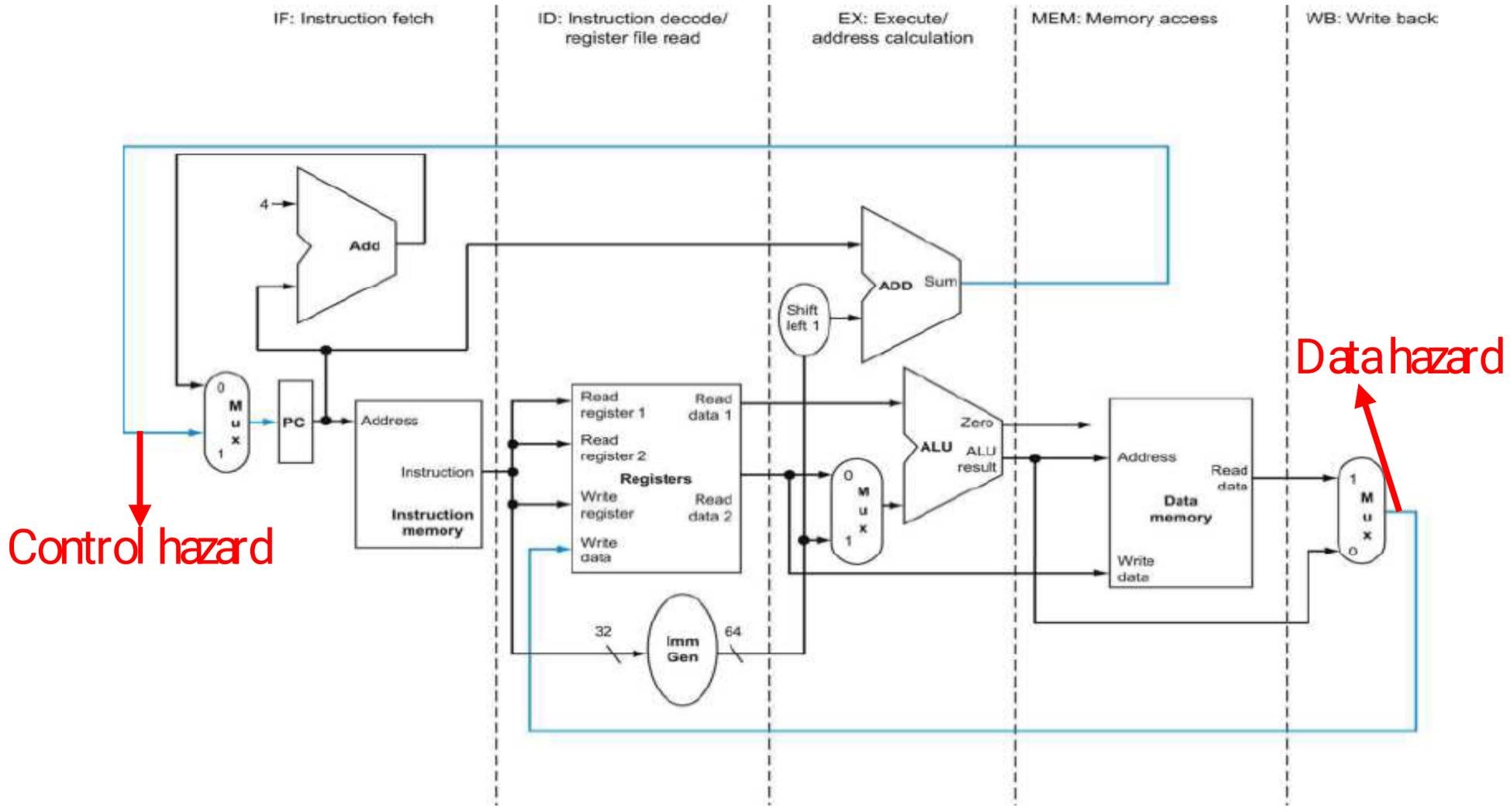
Five-Stage Pipelining

1. IF: Instruction fetch
2. ID: Instruction decode and register file read
3. EX: Execution or address calculation
4. MEM: Data memory access
5. WB: Write back



Pipelined Datapath and Control

Five-Stage Pipelining



- Hazards occur due to signal which move from right to left direction

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

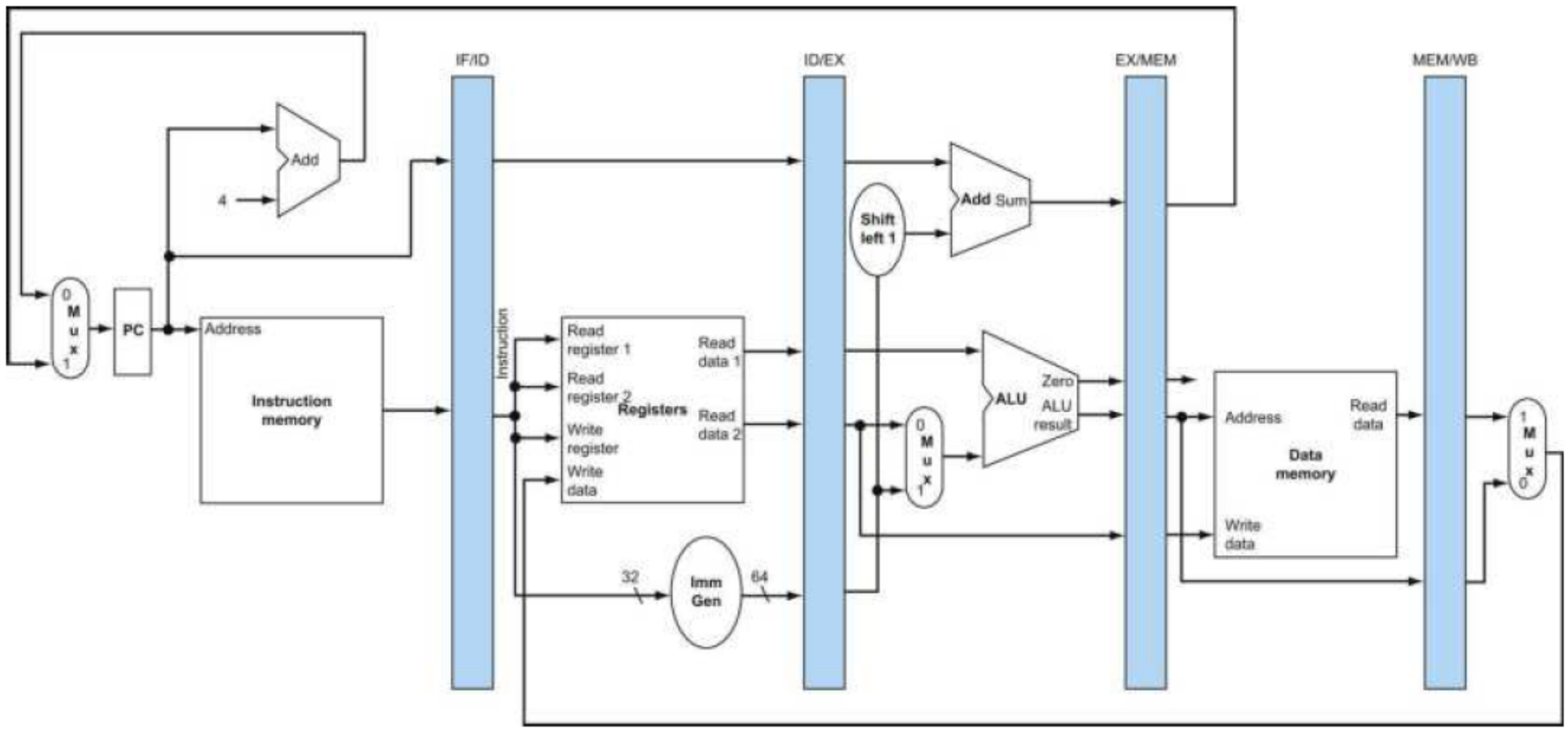
Five-Stage Pipelining

What is the problem in previous pipelined datapath?
- The information is lost when the next instruction enters a certain pipeline stage
- Pipeline stages must store values such as register numbers or control signals for execution of multiple different instructions at the same time
  → We can save them using pipeline registers

- Pipeline registers can be added between every stage
- They temporarily store the intermediate data or control signals between the pipeline stages

Pipelined Datapath and Control

Five-Stage Pipelining



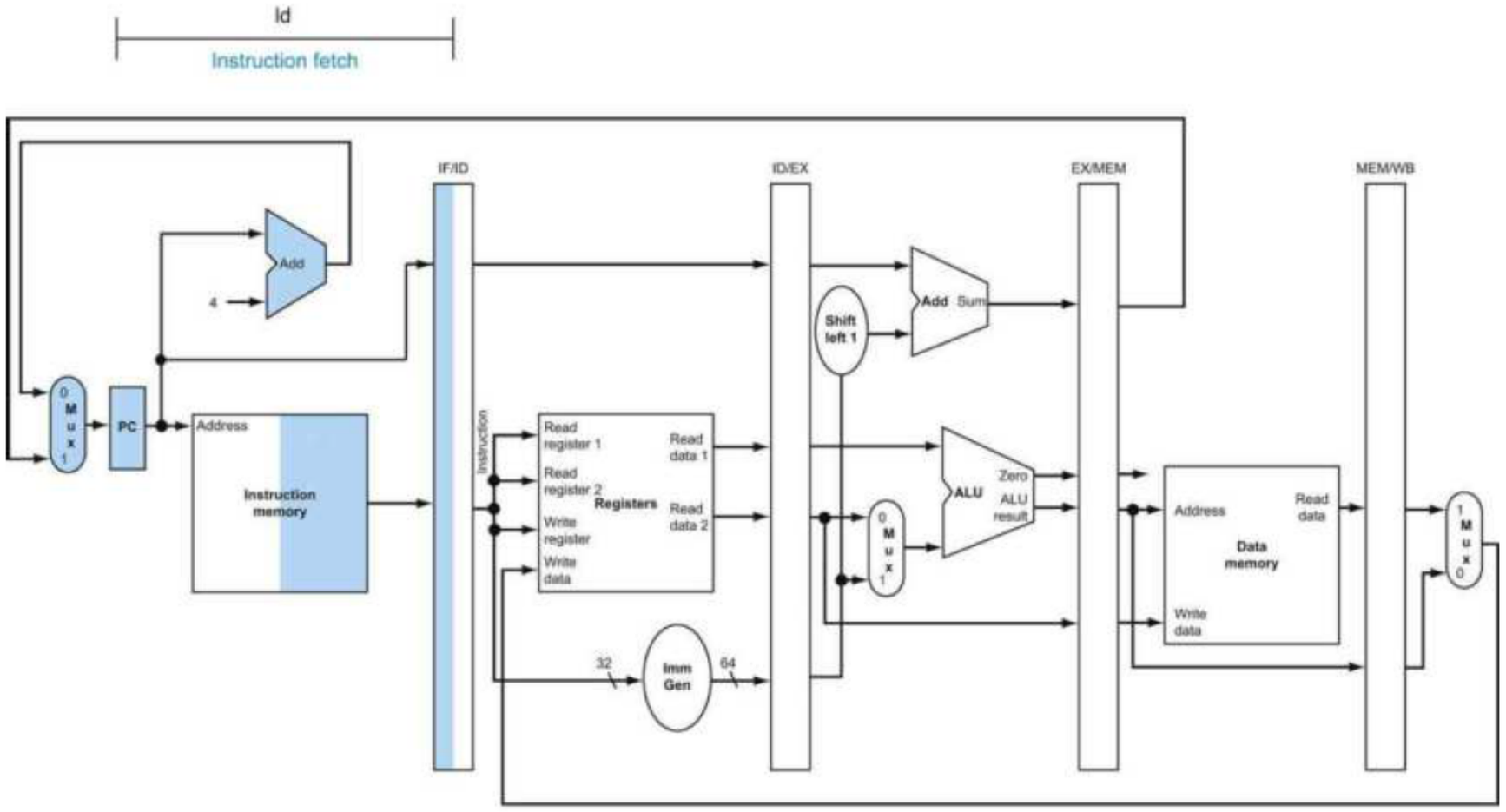
- We can store information to the pipeline registers to execute the different instructions in each stage

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Example) Execution of pipelined processor for load instruction

1. Instruction fetch



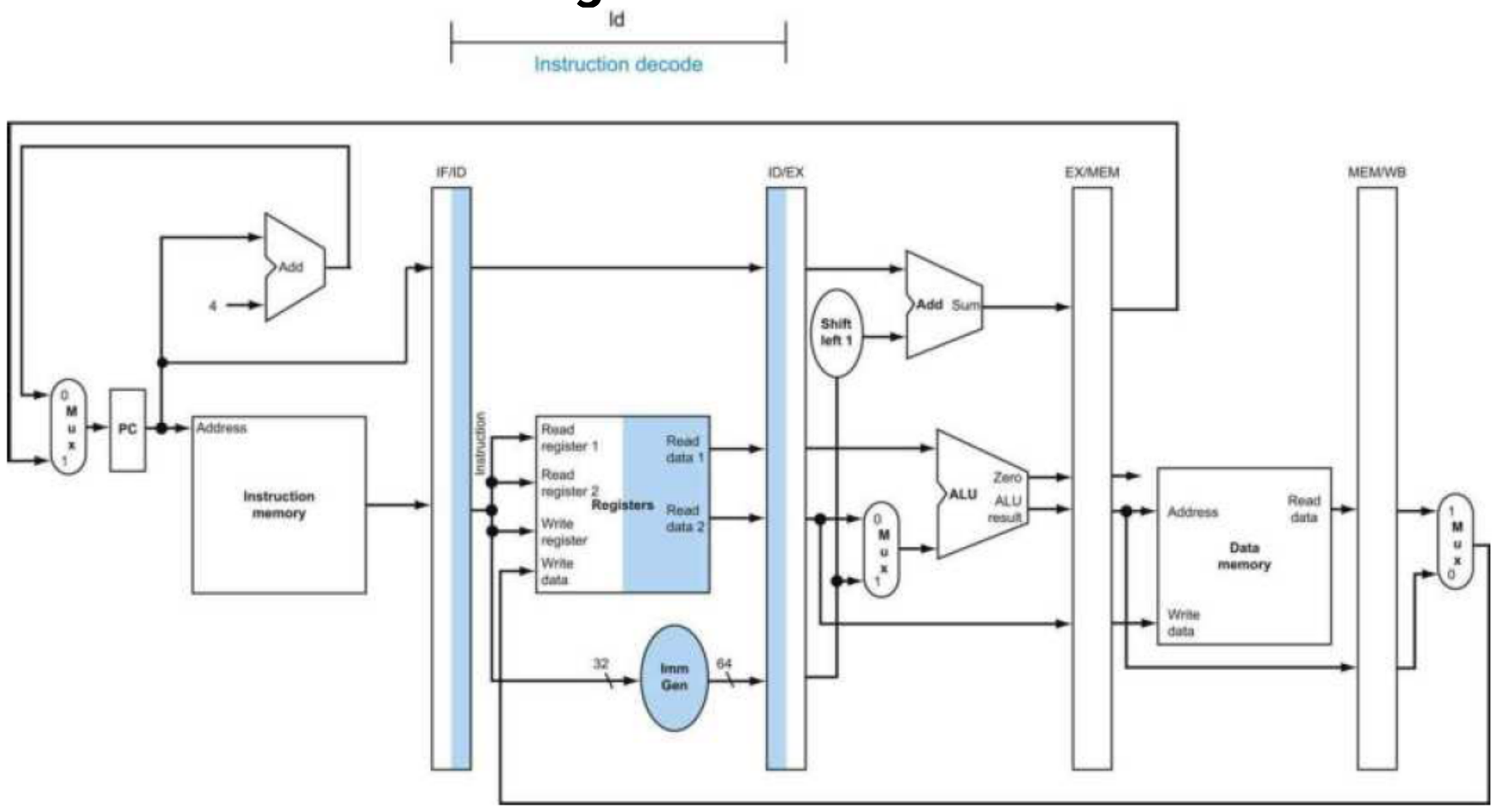
- Instruction is read from memory and then placed in the IF/ID pipeline register

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Example) Execution of pipelined processor for load instruction

2. Instruction decode and register file read



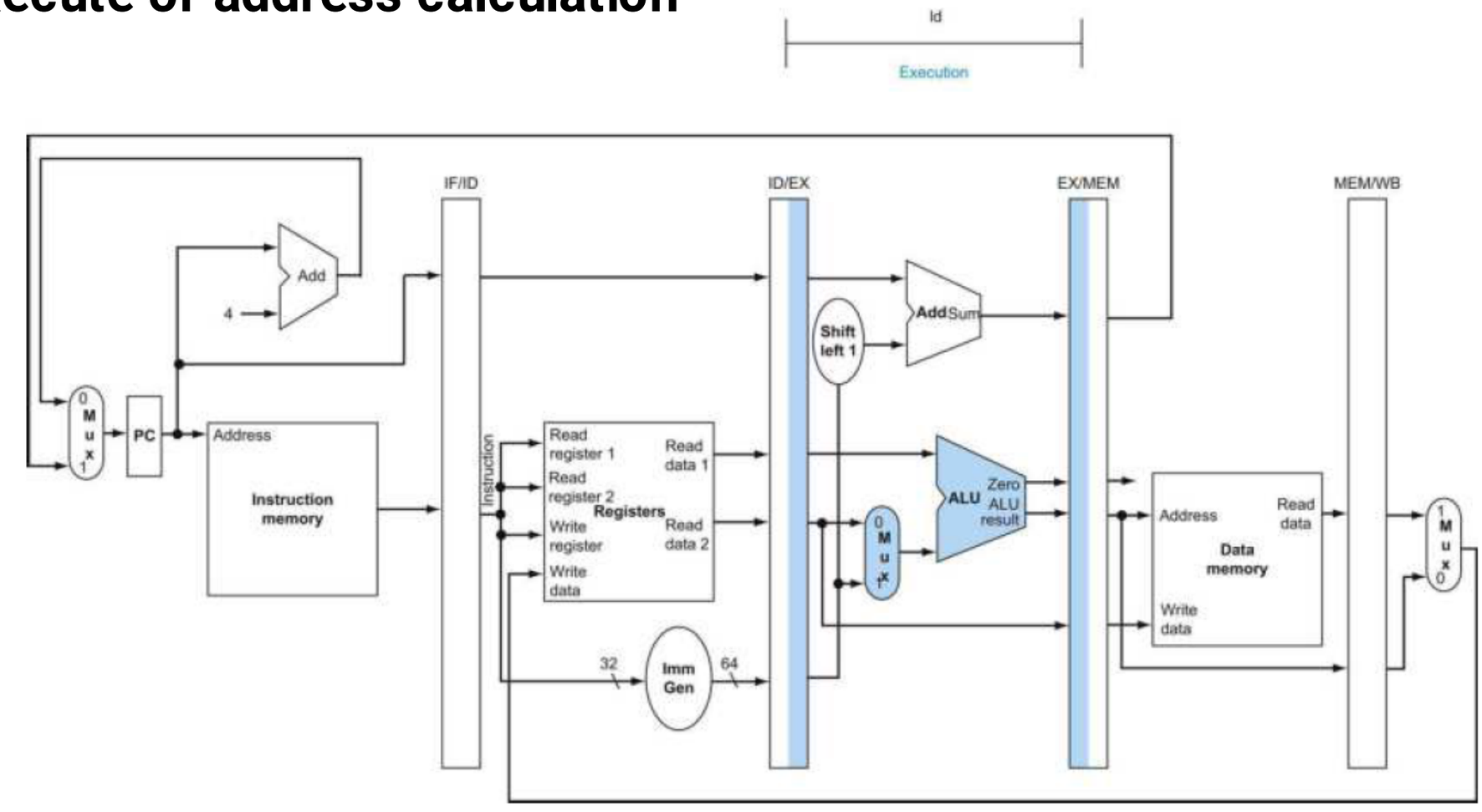
- IF/ID pipeline register supply the immediate field, two register numbers, and PC

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Example) Execution of pipelined processor for load instruction

3. Execute or address calculation



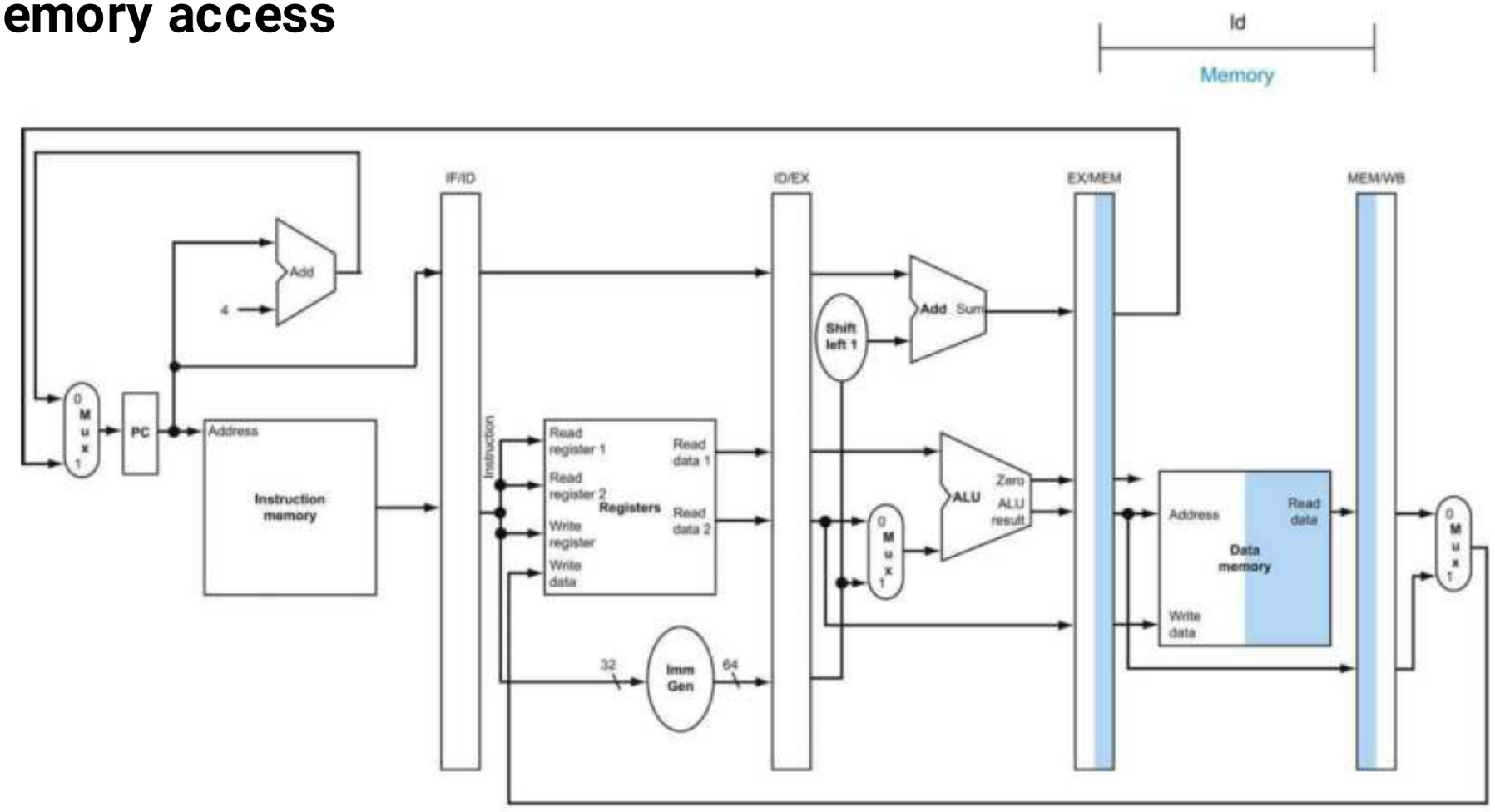
- ID/EX pipeline register supply the values of registers and immediate value
- The result of sum will be placed at EX/MEM pipeline register

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Example) Execution of pipelined processor for load instruction

4. Memory access



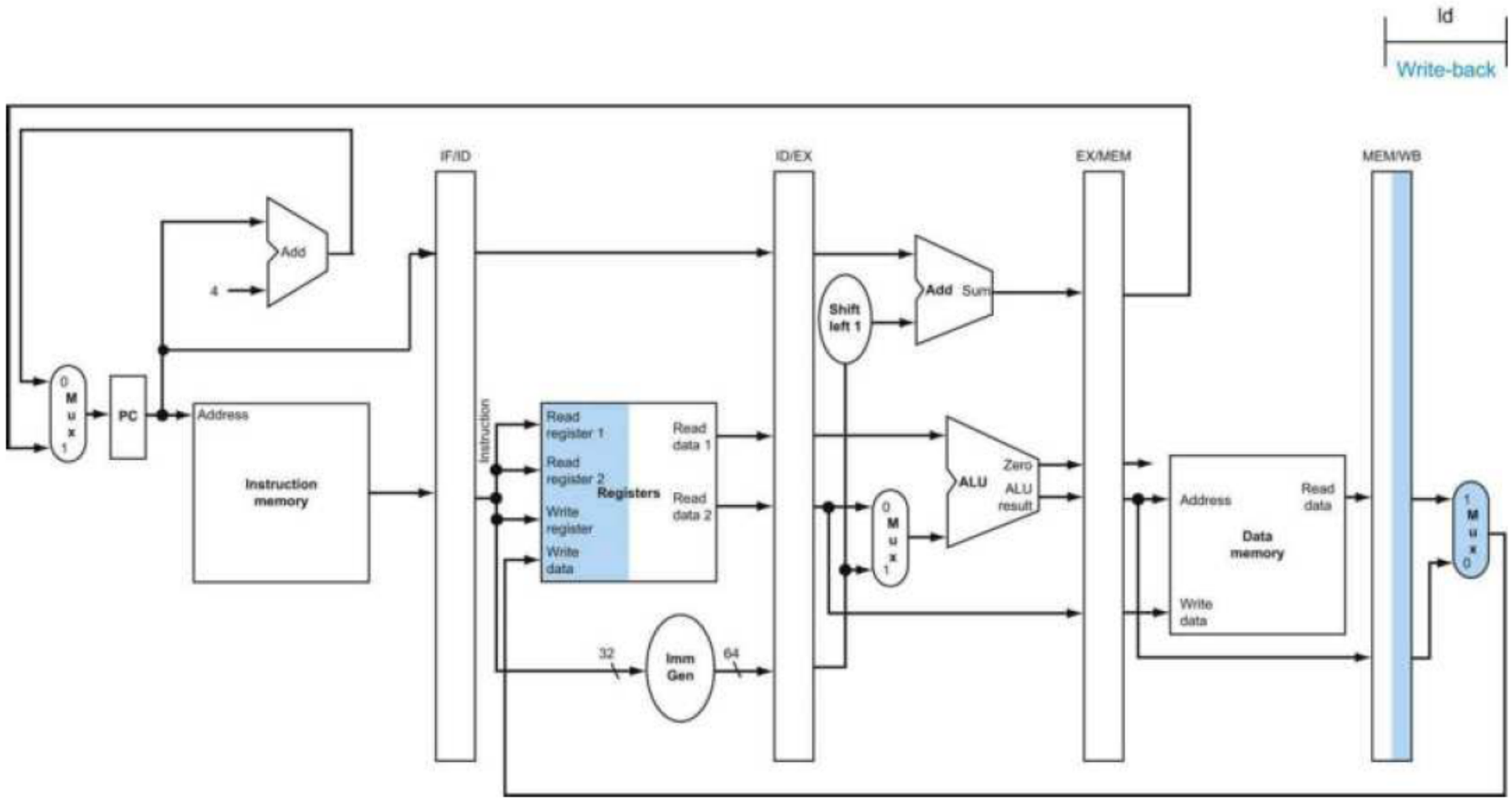
- EX/MEM pipeline register supply the address of memory
- Read the data using the address of memory, and then save it to MEM/WB register

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Example) Execution of pipelined processor for load instruction

5. Write-back



- MEM/WB register supply the data, and then write it into the register file

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Five-Stage Pipelining

Did you find any problem in the previous slides ?

- How processor figures out the destination register number?
  → We need to pass the write register number during execution of load instruction
- Write register number will also be saved and passed into pipeline register and used in the WB stage

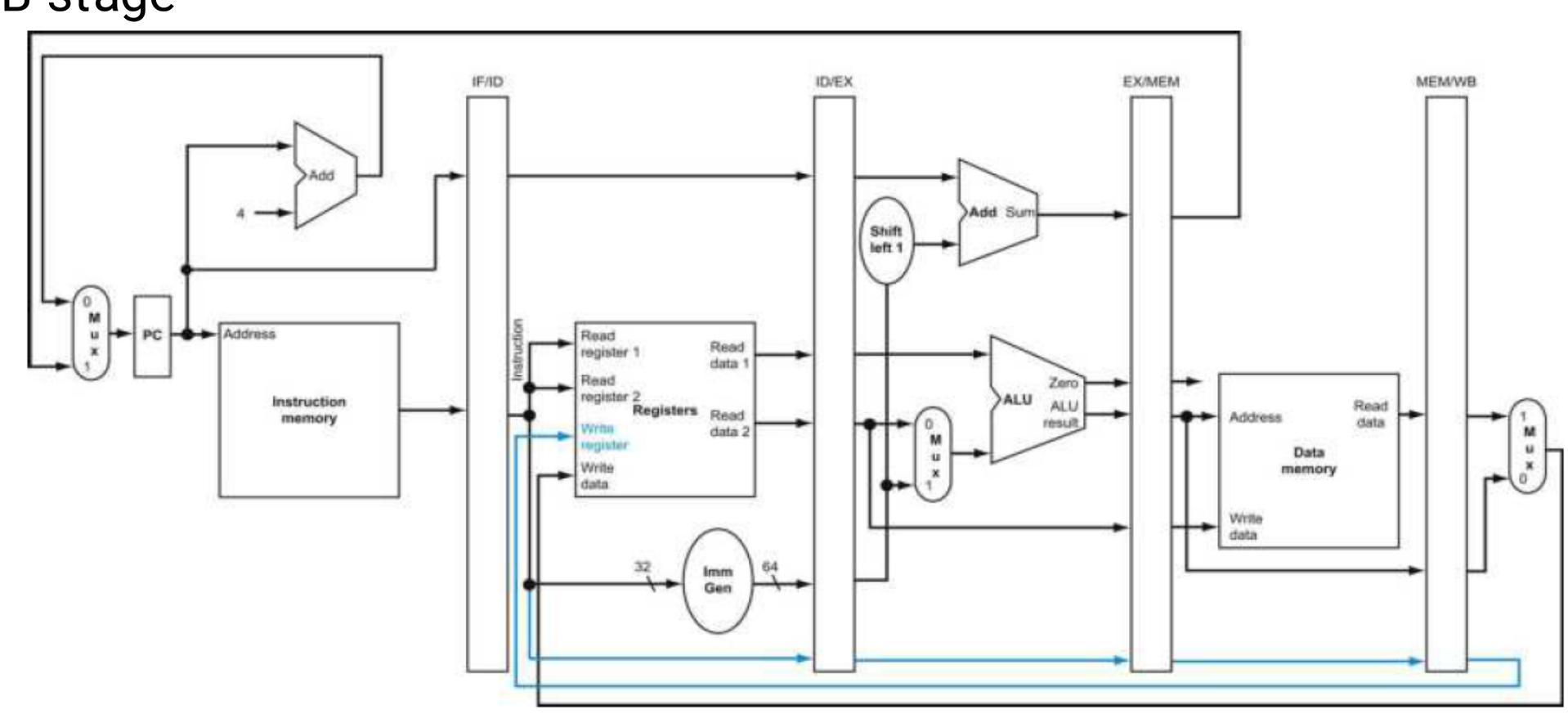


Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

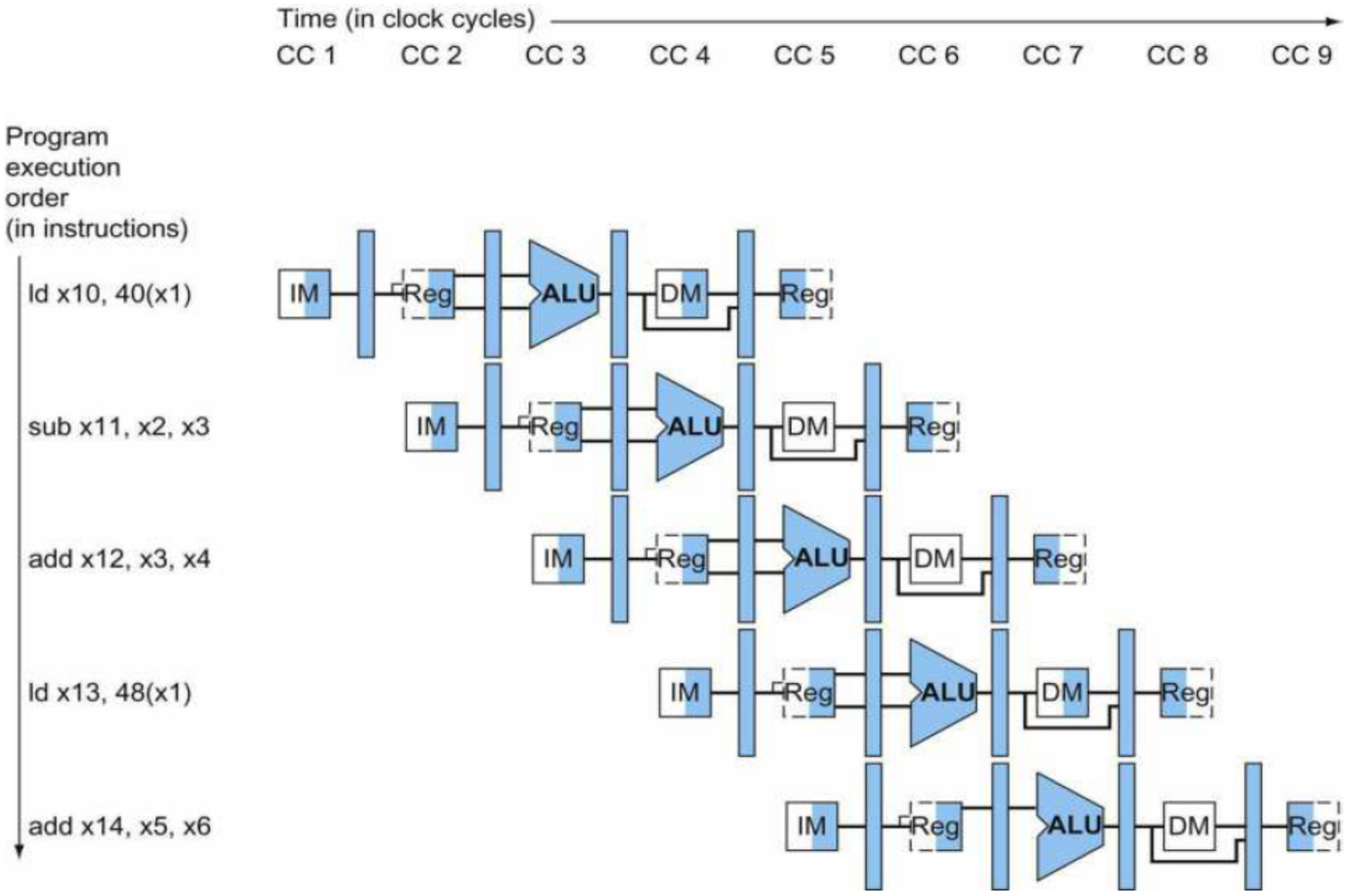
Pipeline Diagram

Let's see the multiple clock cycle pipeline diagram and the single clock cycle diagram corresponding to the following 5 instructions

```
ld      x10, 40(x1)
sub     x11, x2, x3
add     x12, x3, x4
ld      x13, 48(x1)
add     x14, x5, x6
```

Pipelined Datapath and Control

Multi Clock Cycle Pipeline Diagram

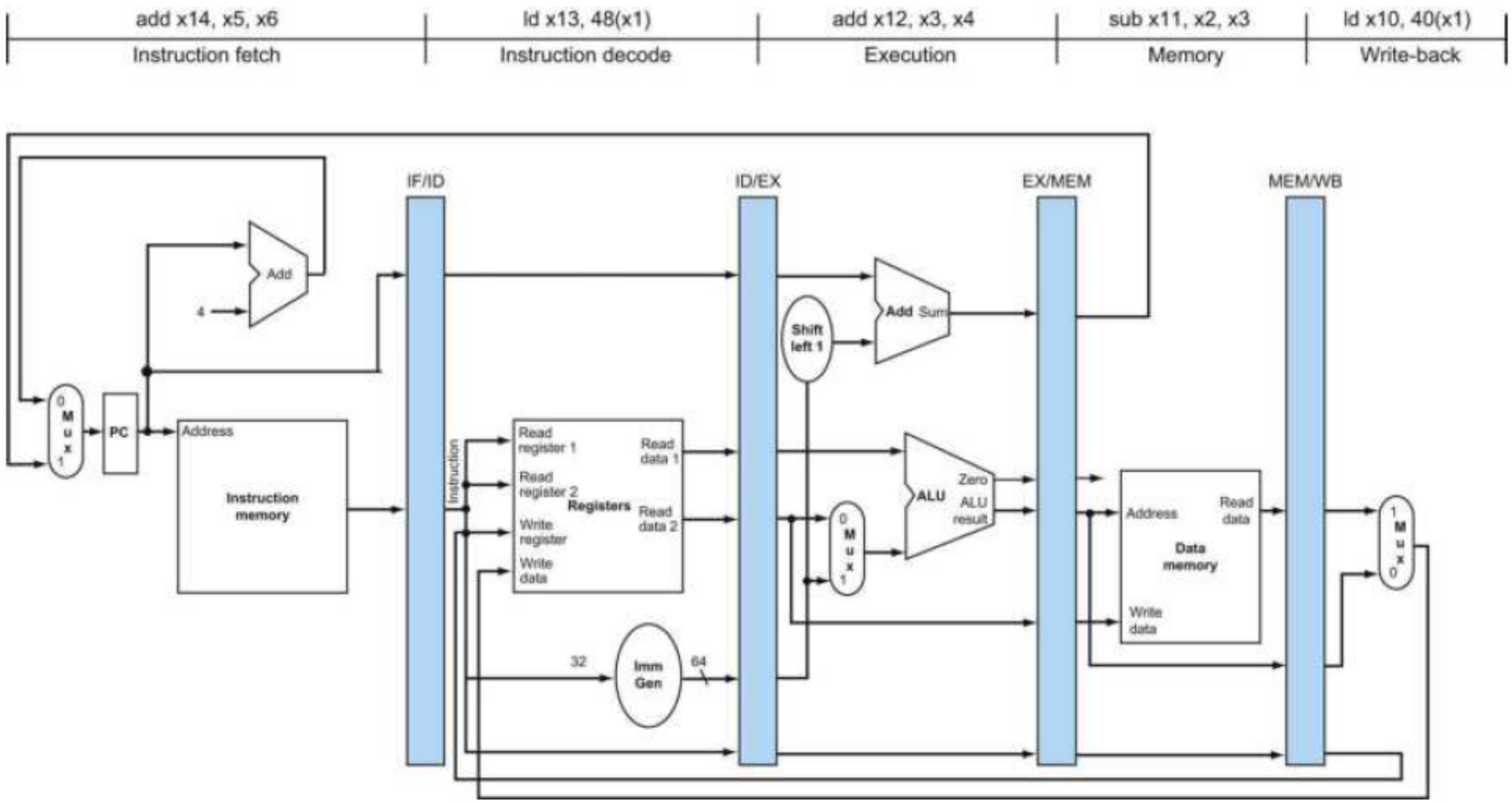


- Frequently used to represent pipeline behaviors

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Single Clock Cycle Pipeline Diagram



- It is like a snapshot of the pipeline execution for a certain clock cycle

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Pipelined Control

Let’s consider the simple datapath with pipeline registers
How can we deliver control signals?

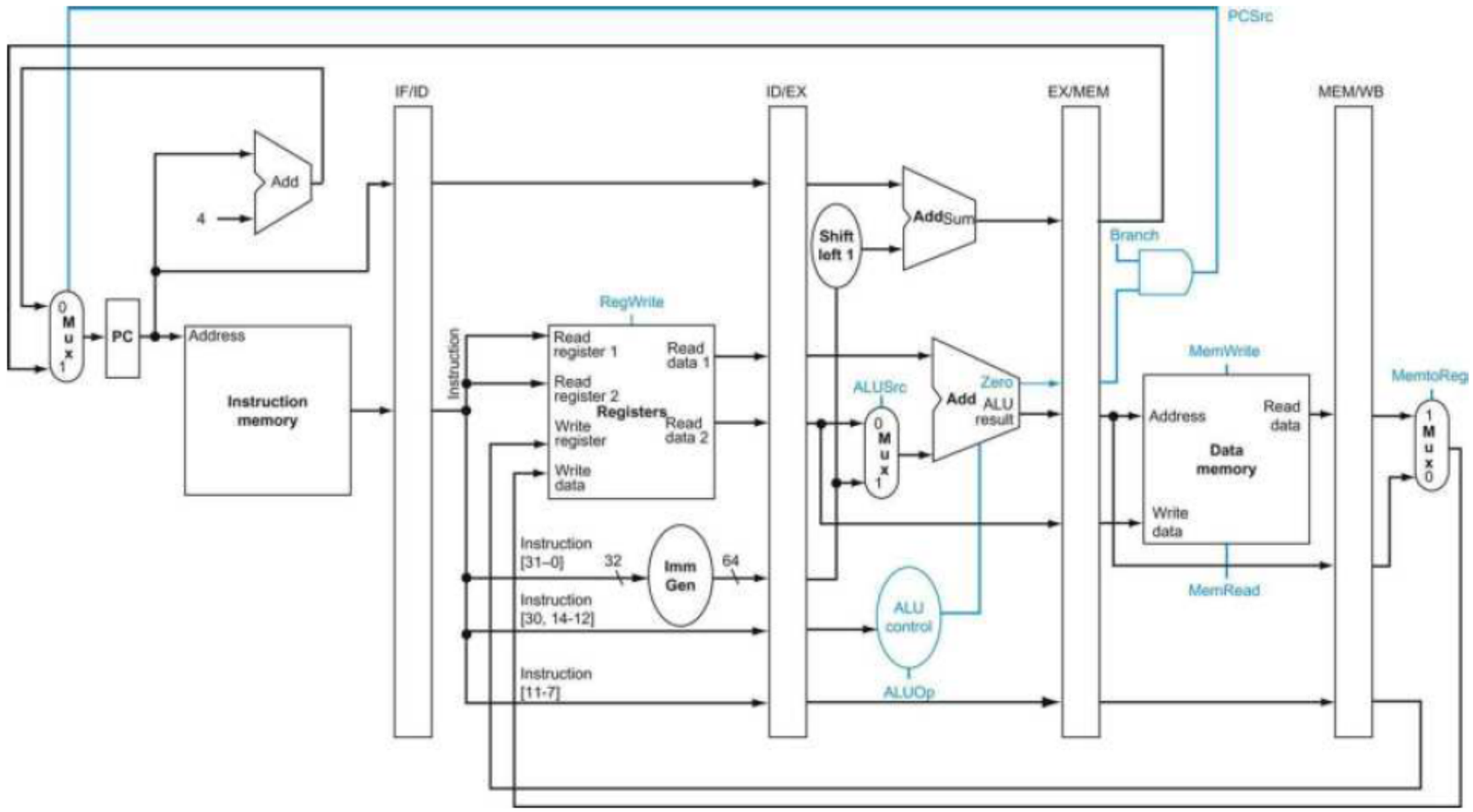


Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Pipelined Control

- The control signals for each instruction also need to be saved into pipeline register
- The following table represents the values of the control lines for each instruction format

Instruction	Execution/address calculation stage control lines		Memory access stage control lines			Write-back stage control lines	
	ALUOp	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	10	0	0	0	0	1	0
ld	00	1	0	1	0	1	1
sd	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X

Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Implementing Control

- Create the control signals during instruction decoding stage (ID)
- Pipeline registers supply the control signals to the corresponding pipeline stage

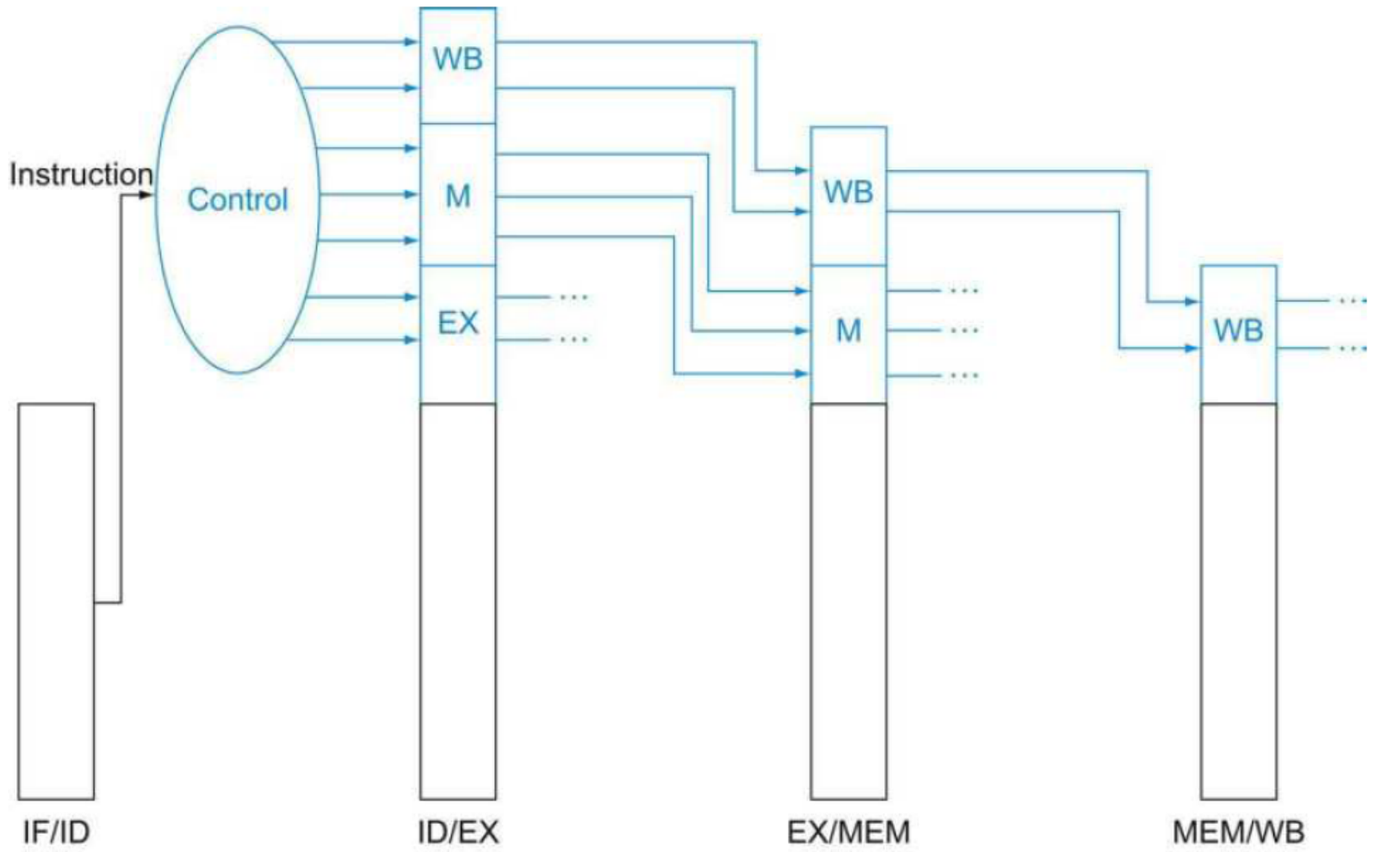


Figure is from “Computer Organization and Design”

Pipelined Datapath and Control

Full Datapath

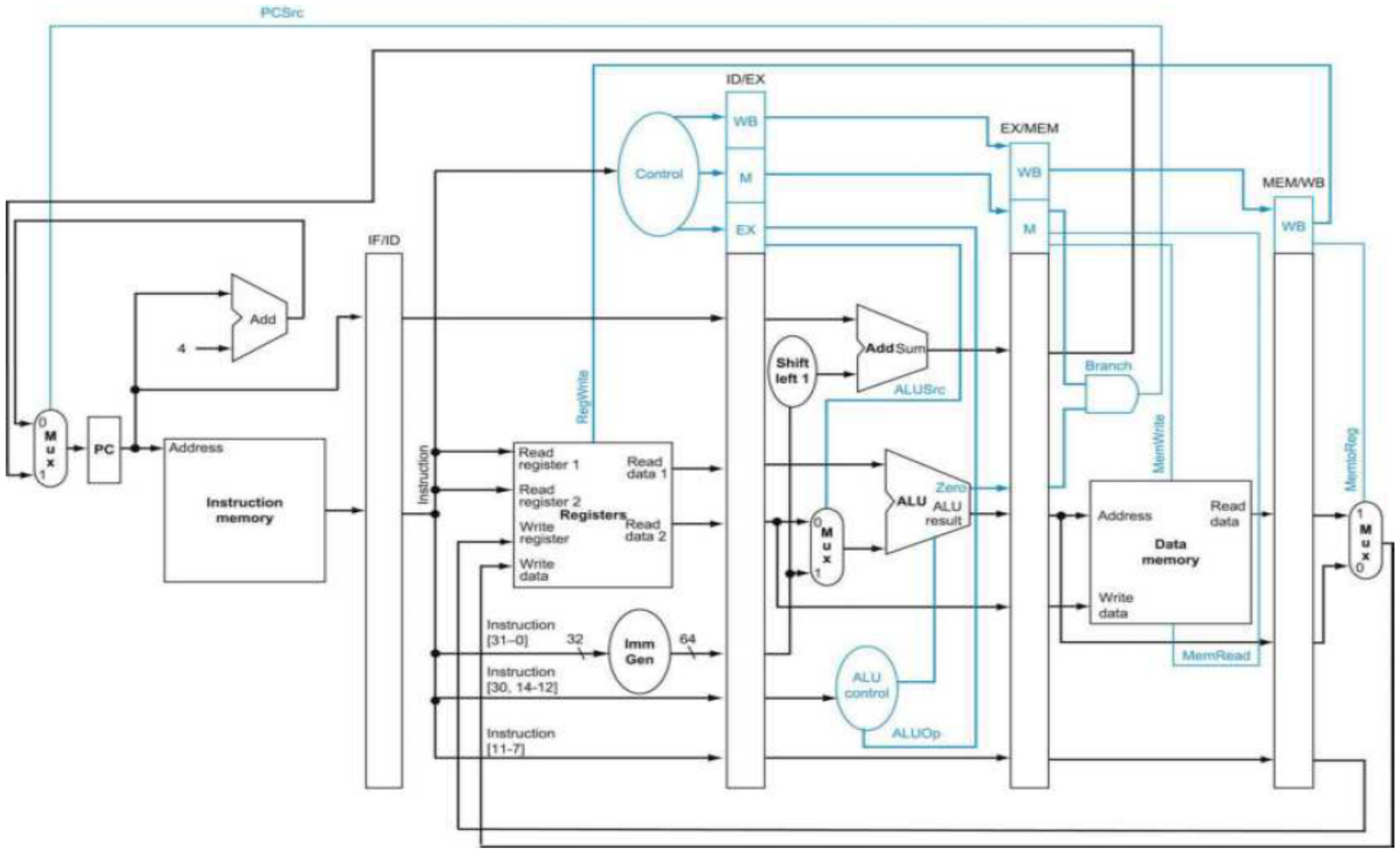


Figure is from “Computer Organization and Design”

Data Hazards

Data Hazard and Forwarding (Bypassing)

Data hazard:
The following instructions must wait (stall or bubble) for another to complete due to data dependencies

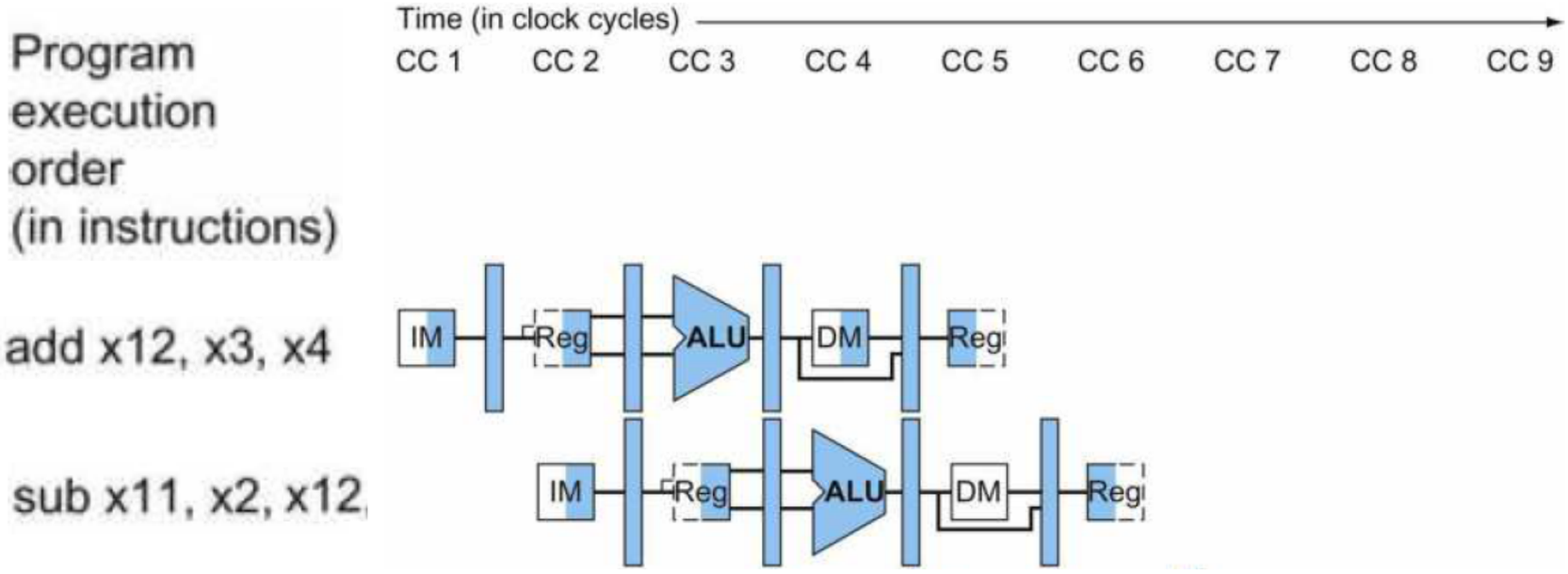
Q) Let's consider the following code:

```
add   x12, x3, x4
sub   x11, x2, x12
```

What if we execute these two instructions without any stall?

Data Hazards

Data Hazard



Let's assume that original value of x12 is 10, it will be updated to 20 by add instruction
→ Then, which value is read for x12 in sub instruction? 10? 20?

Figure is from "Computer Organization and Design"

Data Hazards

Data Hazard

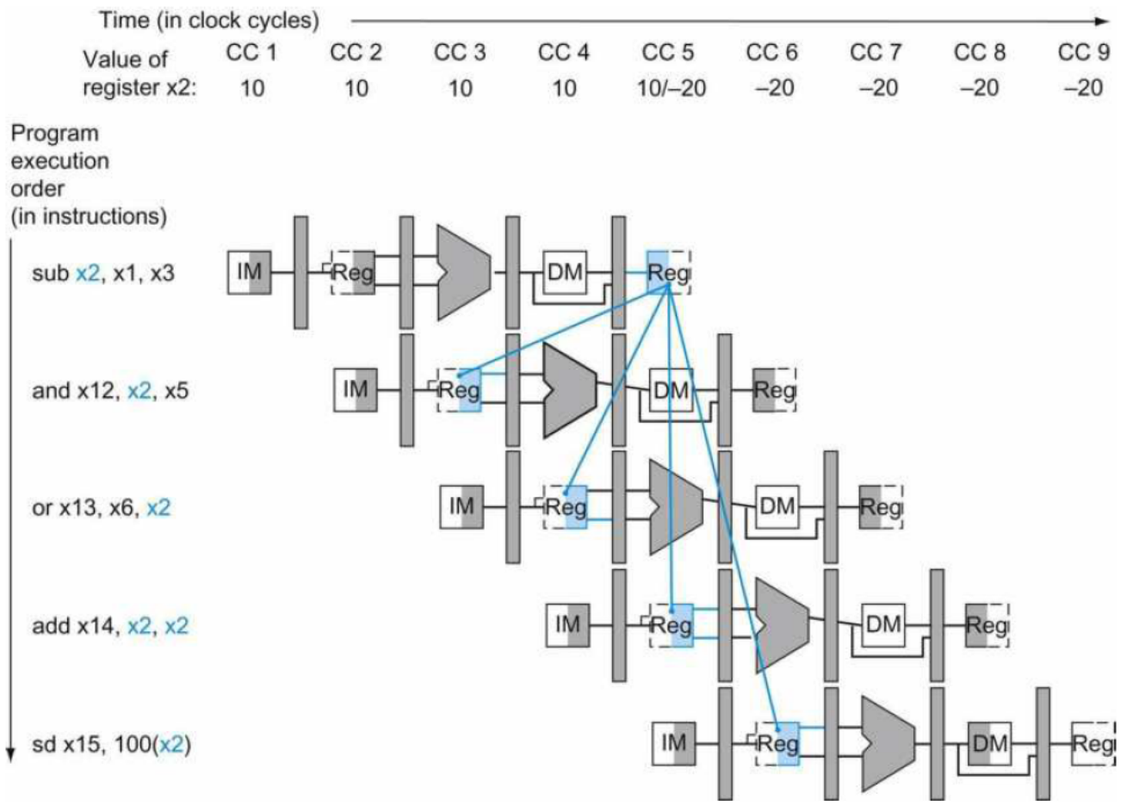
Let's see another example consisting many instructions.
Q) There is an instruction sequence with several data dependences:

```
sub     x2, x1, x3
and     x12, x2, x5
or      x13, x6, x2
add     x14, x2, x2
sd      x15, 100(x2)
```

This RISC-V code will works correctly?
Let's assume that the register write happens in the first half of the clock cycle and the register read happens in the second half

Data Hazards

Data Hazard

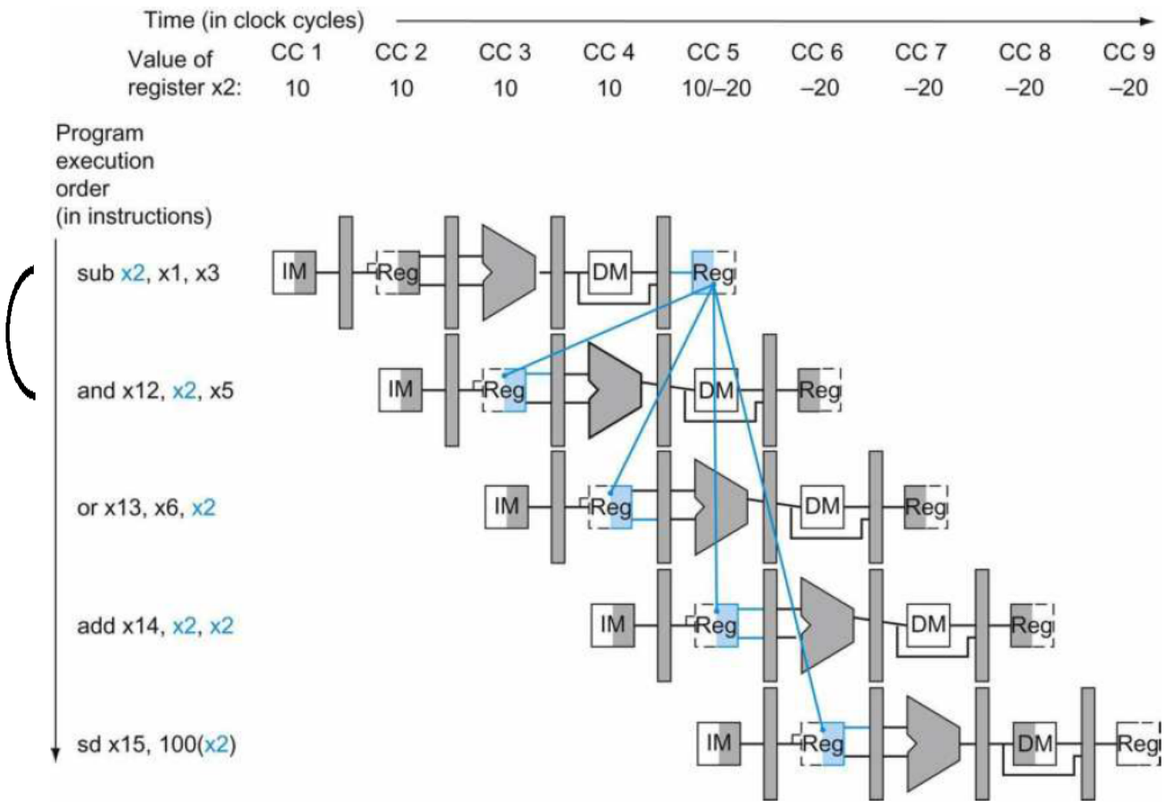


- The second and third instruction will get an old (incorrect) data from register x2
- After updating the value of x2 register in the first instruction (CC 5), the instructions can read exact value from x2 register
- Therefore, x2 register will be read correctly from the fourth instruction (add x14, x2, x2)

Figure is from “Computer Organization and Design”

Data Hazards

Data Hazard Conditions



Condition 1a

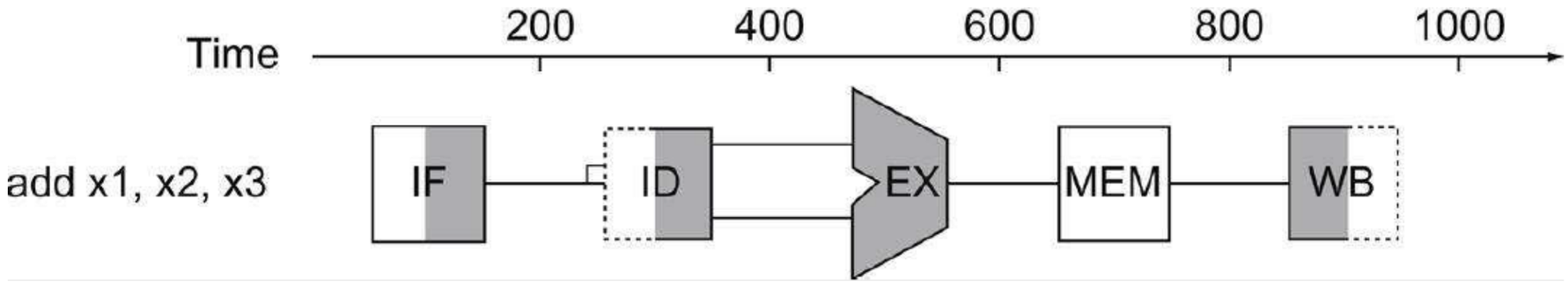
1a. EX/MEM.RegisterRd (≠ x0) = ID/EX.RegisterRs1
1b. EX/MEM.RegisterRd (≠ x0) = ID/EX.RegisterRs2
2a. MEM/WB.RegisterRd (≠ x0) = ID/EX.RegisterRs1
2b. MEM/WB.RegisterRd (≠ x0) = ID/EX.RegisterRs2

Figure is from “Computer Organization and Design”

Data Hazards

Forwarding (Bypassing)

- Pipeline forwards (or bypasses) the data to the following instructions before the destination register is updated
- Extra hardware to retrieve the missing item earlier from the internal resources are required

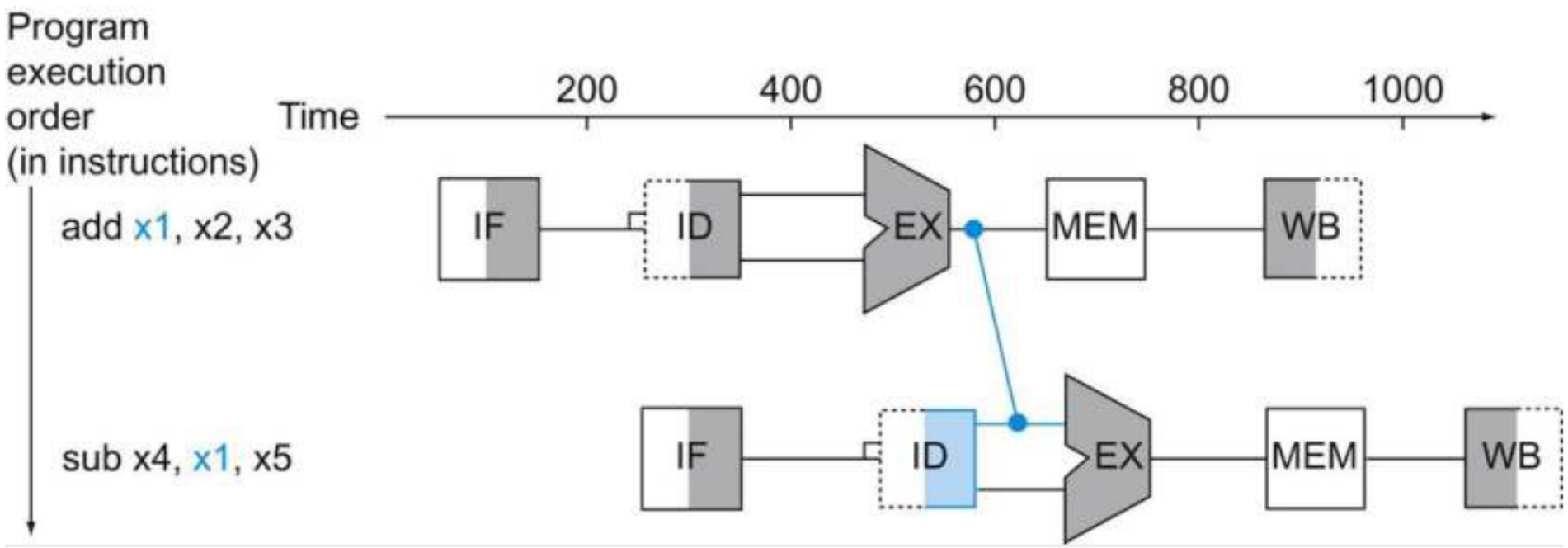


- Where is the result of add instruction?
→ Between EX stage and MEM stage

Figure is from “Computer Organization and Design”

Data Hazards

Forwarding (Bypassing) Example 1



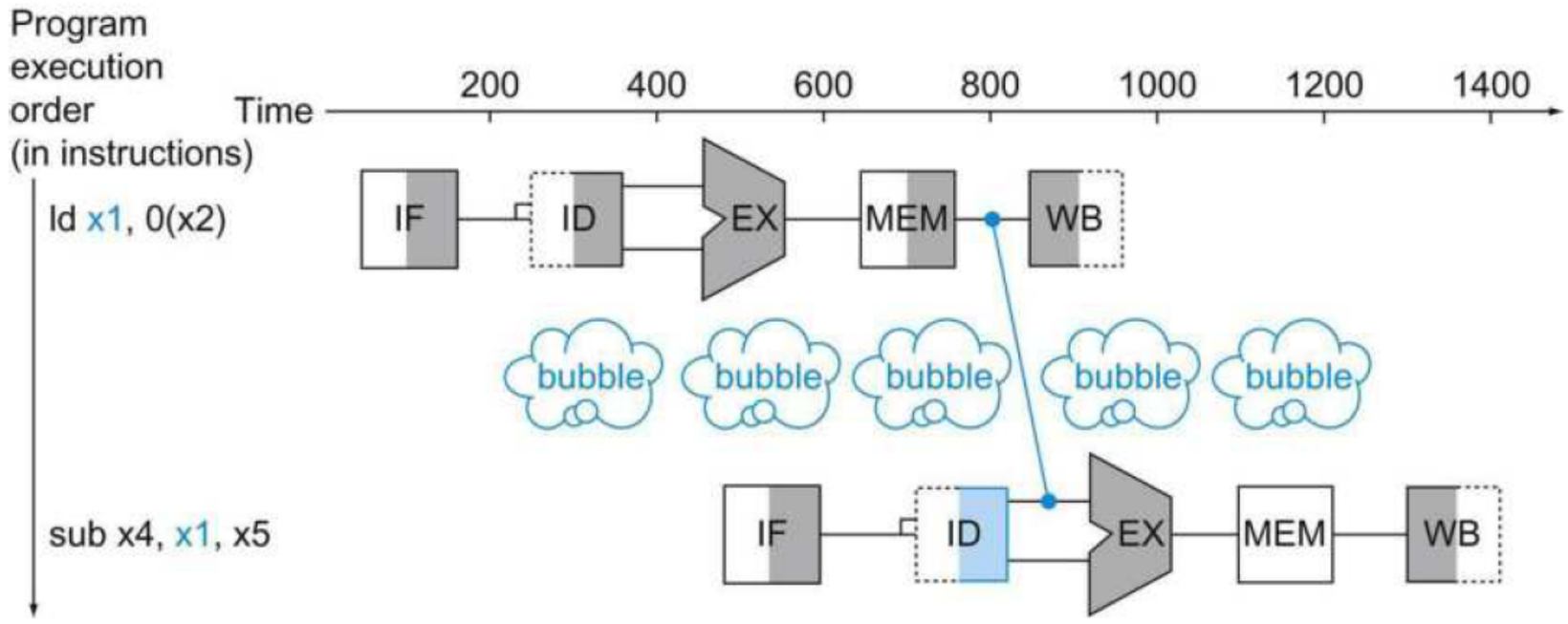
- The updated value can be used before WB stage
- This technique will give correct results for the next instruction without stall (bubble)

Figure is from “Computer Organization and Design”

Data Hazards

Forwarding (Bypassing) Example 2

Can we remove every stall by using forwarding?



- Even though forwarding is used, there is a case we need to stall the pipeline
- Load + instruction that has data dependency to load
- Like above example, do nothing between two instructions is NOP instruction
- NOP instruction can be inserted by changing every control signal in the pipeline register to 0

Figure is from “Computer Organization and Design”

Data Hazards

Forwarding (Bypassing) Example 2

How can we remove stall in following code?

Let's assume that the sequence is changed from left code to right code:

```
ld      x1, 0(x2)                ld      x1, 0(x2)
sub     x4, x1, x5      ⟶       add     x10, x11, x28
add     x10, x11, x28            sub     x4, x1, x5
 ⋮                                ⋮
```

(Original code) (Changed code)

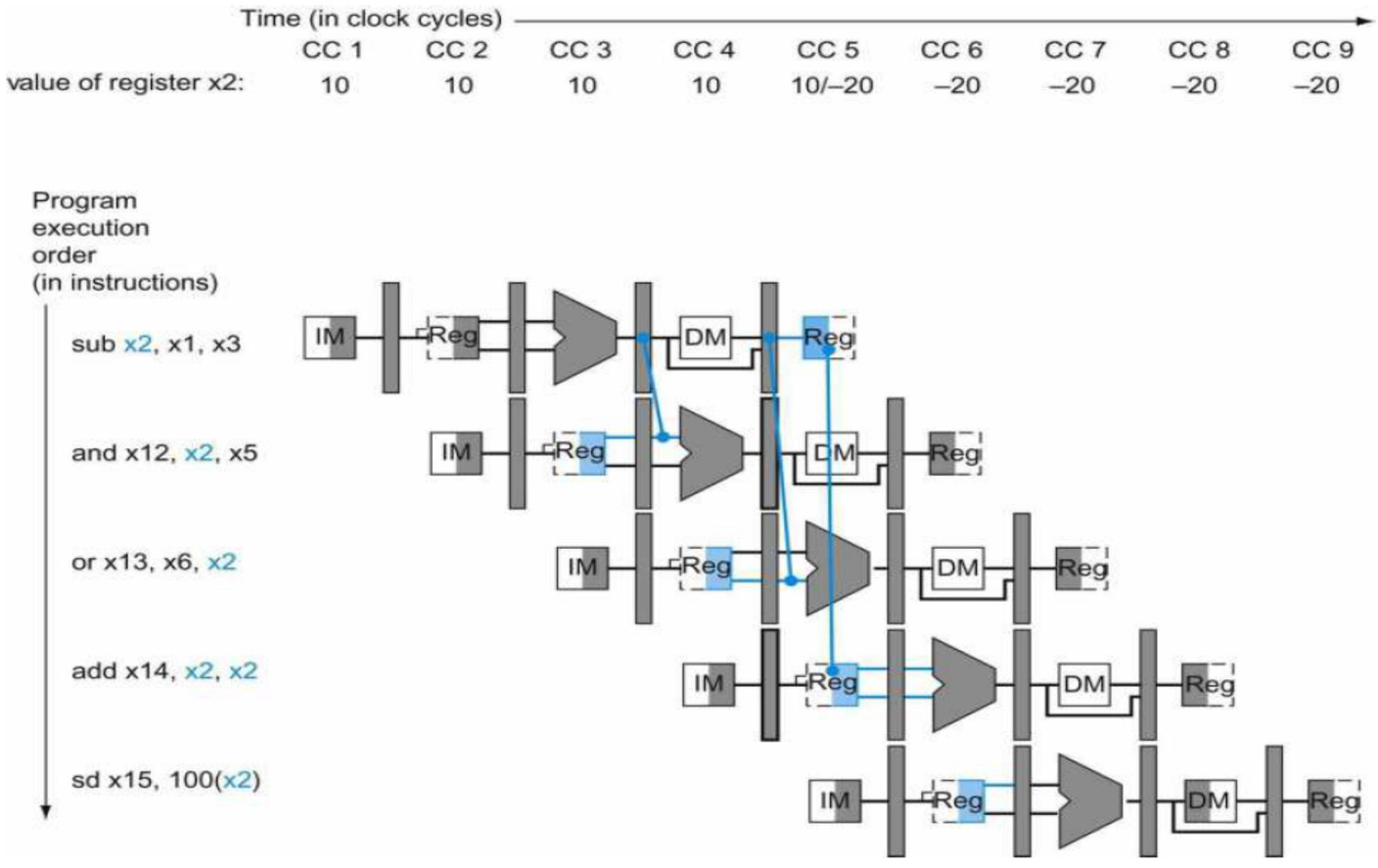
Is it working correctly?

→ We can change the sequence of the instructions if there is no data dependency

→ It requires hardware or compiler supports

Data Hazards

Pipeline Diagram with Forwarding

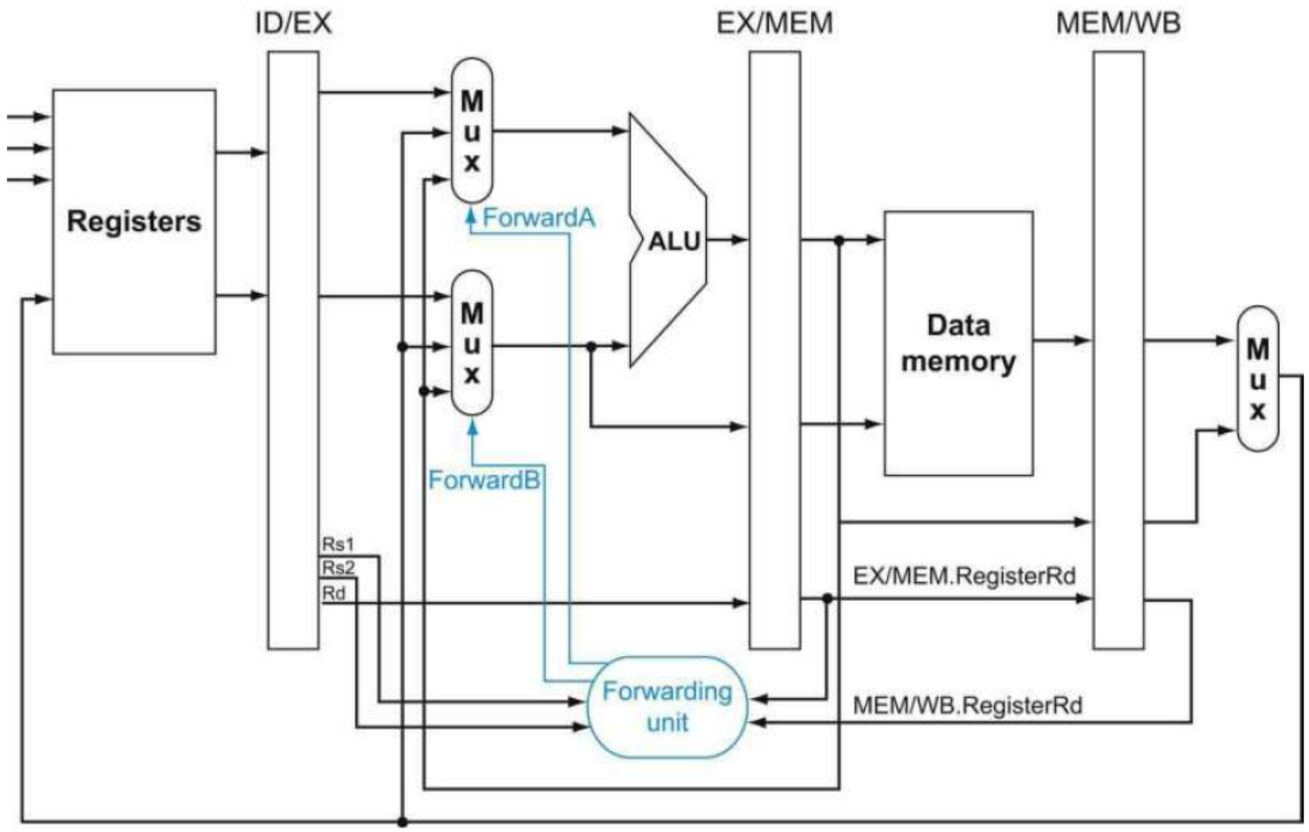


- Every instruction can be executed with the correct value of registers
- To implement forwarding, some multiplexers and controls are needed

Figure is from “Computer Organization and Design”

Data Hazards

Forwarding Unit

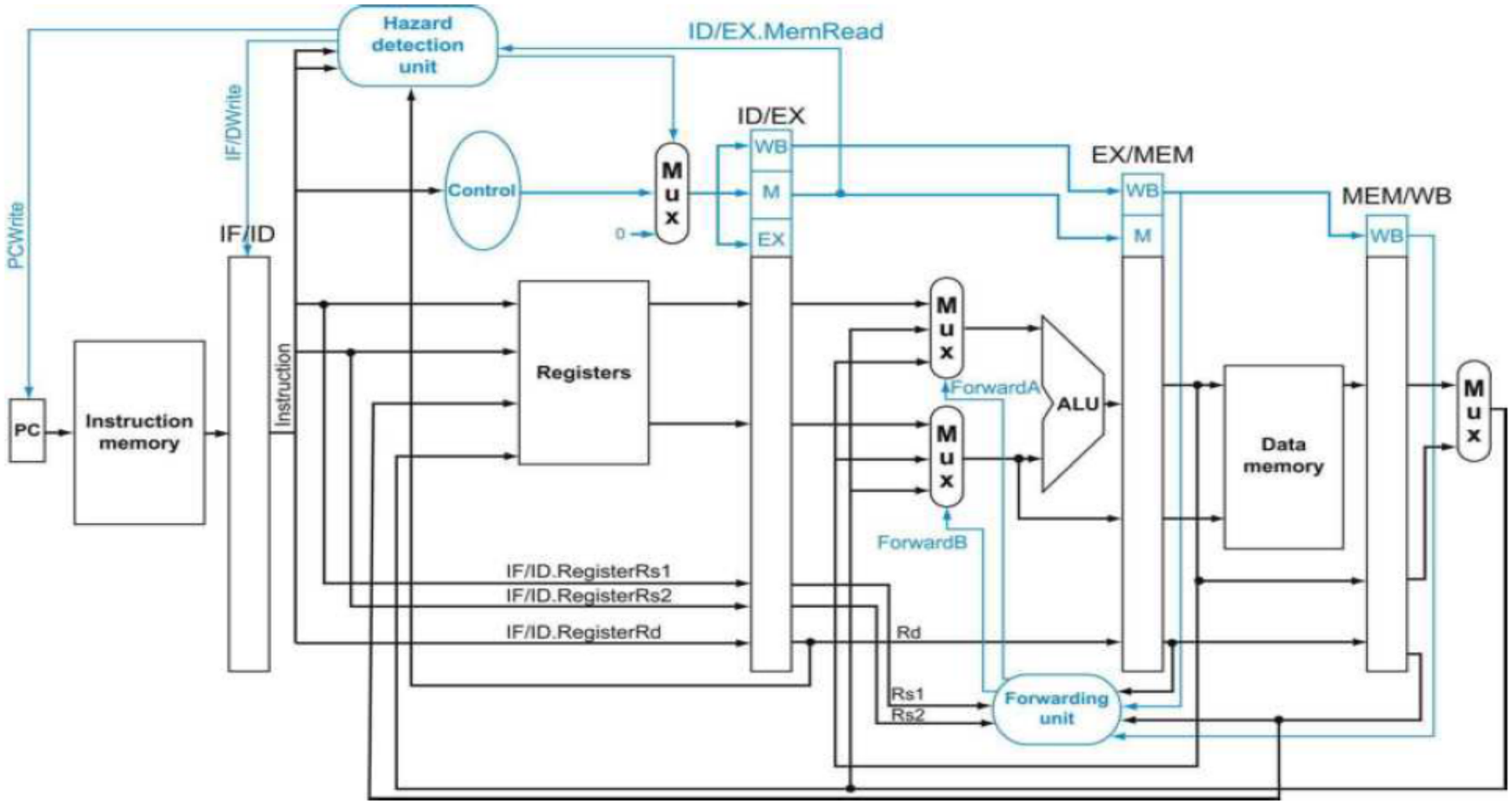


- 2 multiplexors can select the values as inputs of ALU among the 3 types
  1) Forward signal = 00: select the value from register file
  2) Forward signal = 10: select the value from the result of EX stage
  3) Forward signal = 01: select the value from the memory

Figure is from “Computer Organization and Design”

Data Hazards

Pipeline Connection for Hazard



- If the hazard detection unit detects a hazard, this makes every control signal to 0
- Hazard detection unit uses 3 register numbers and MemRead signal

Figure is from “Computer Organization and Design”