Introduction to Pipelining

Araddhana Arvind Deshmukh

Pipelining Outline

- Introduction \
  - Defining Pipelining
  - Pipelining Instructions
- Hazards
  - Structural hazards
  - Data Hazards
  - Control Hazards
- Performance
- Controller implementation

What is Pipelining?

- A way of speeding up execution of instructions

- *Key idea*:

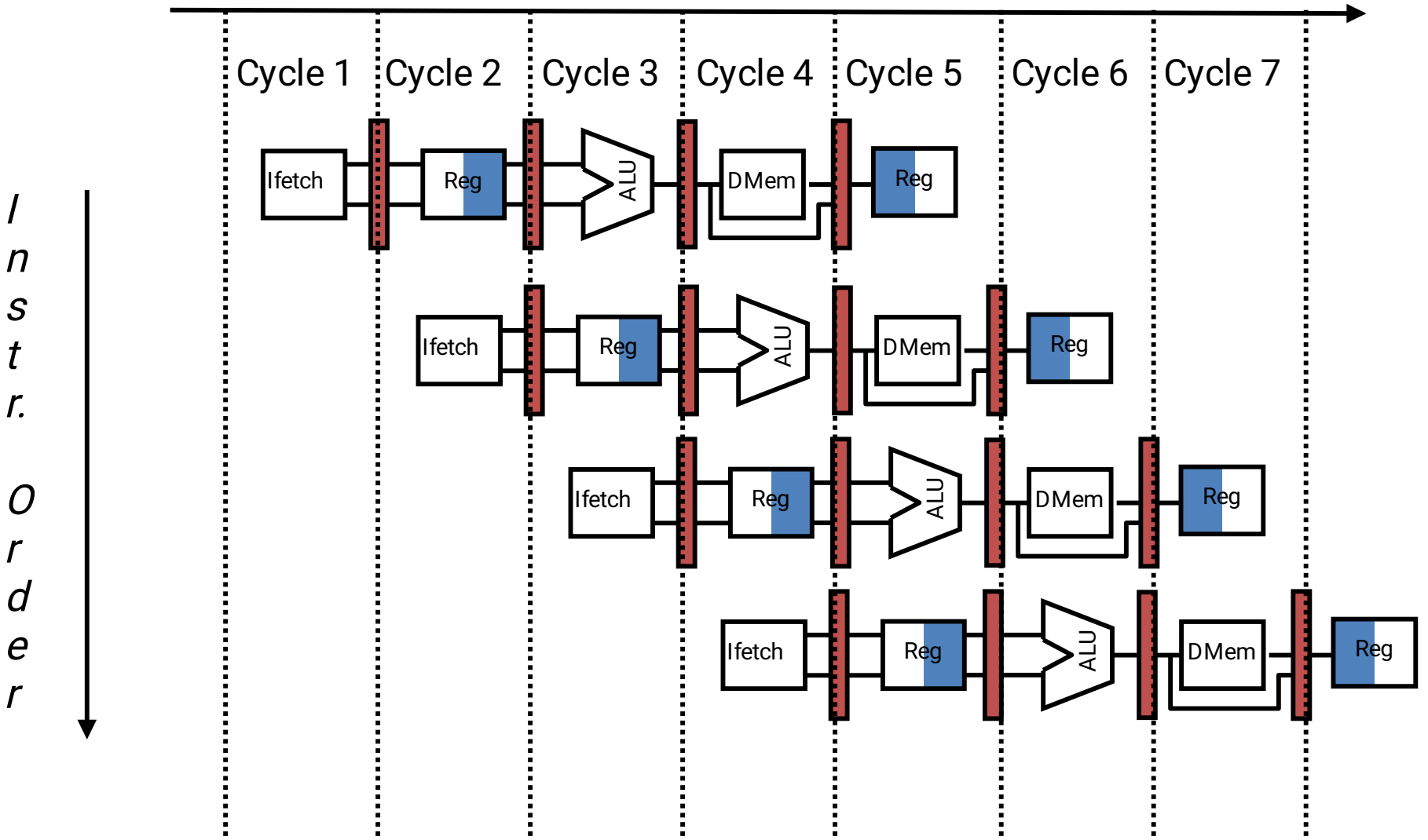
overlap execution of multiple instructions

Pipelining a Processor

- Recall the 5 steps in instruction execution:
  1. Instruction Fetch (IF)
  2. Instruction Decode and Register Read (ID)
  3. Execution operation or calculate address (EX)
  4. Memory access (MEM)
  5. Write result into register (WB)

- Review: Single-Cycle Processor
  - All 5 steps done in a single clock cycle
  - Dedicated hardware required for each step

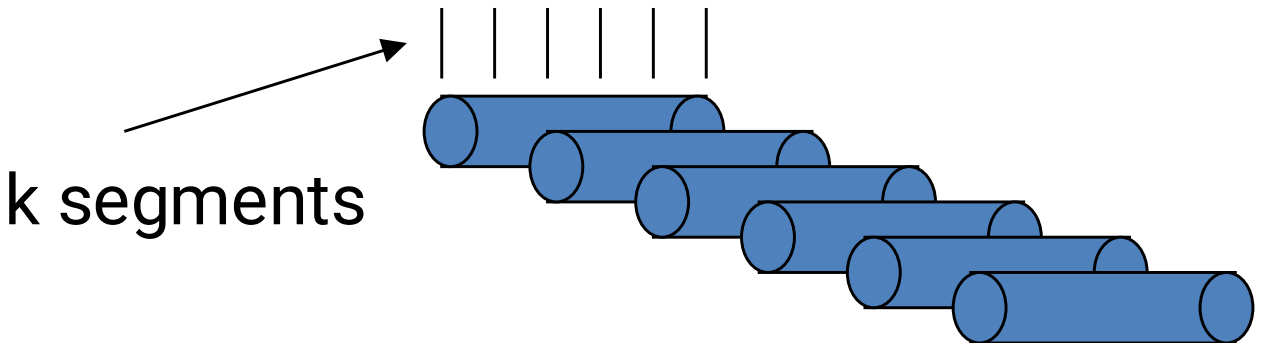
The Basic Pipeline For MIPS



What do we need to add to actually split the datapath into stages?

Pipelining

- Instruction execution is divided into k segments or stages
  - Instruction exits pipe stage k-1 and proceeds into pipe stage k
  - All pipe stages take the same amount of time; called one processor cycle
  - Length of the processor cycle is determined by the slowest pipe stage



Some definitions

- **Pipeline:** is an implementation technique where multiple instructions are overlapped in execution.

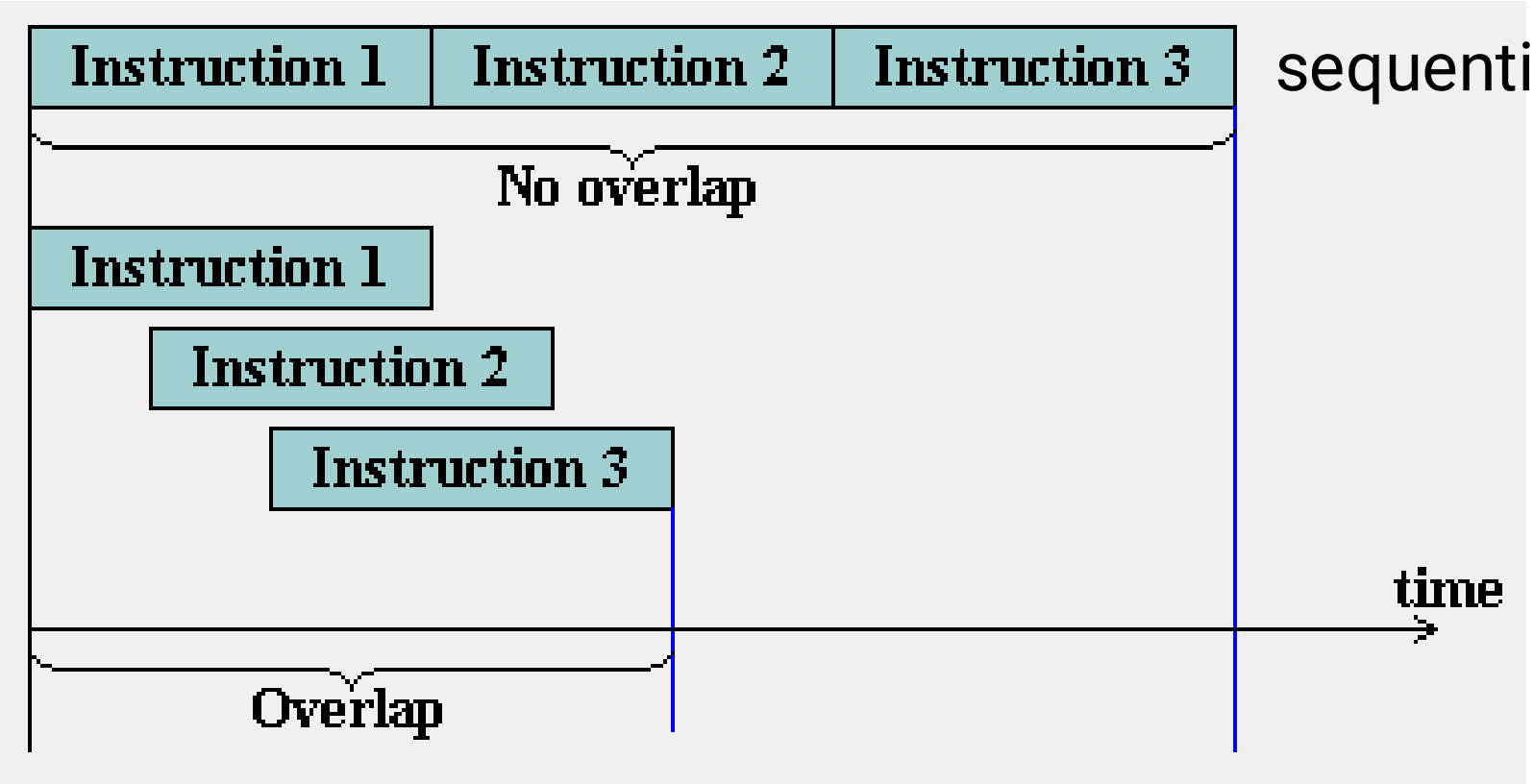
- **Pipeline stage**: The computer pipeline is to divided instruction processing into **stages**.
  - Each stage completes a part of an instruction and loads a new part in parallel.

Some definitions

Throughput of the instruction pipeline is determined by how often an instruction exits the pipeline. Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput.

Machine cycle . The time required to move an instruction one step further in the pipeline. The ***length*** of the machine cycle is determined by the time required for the slowest pipe stage.

Instruction pipeline versus sequential processing

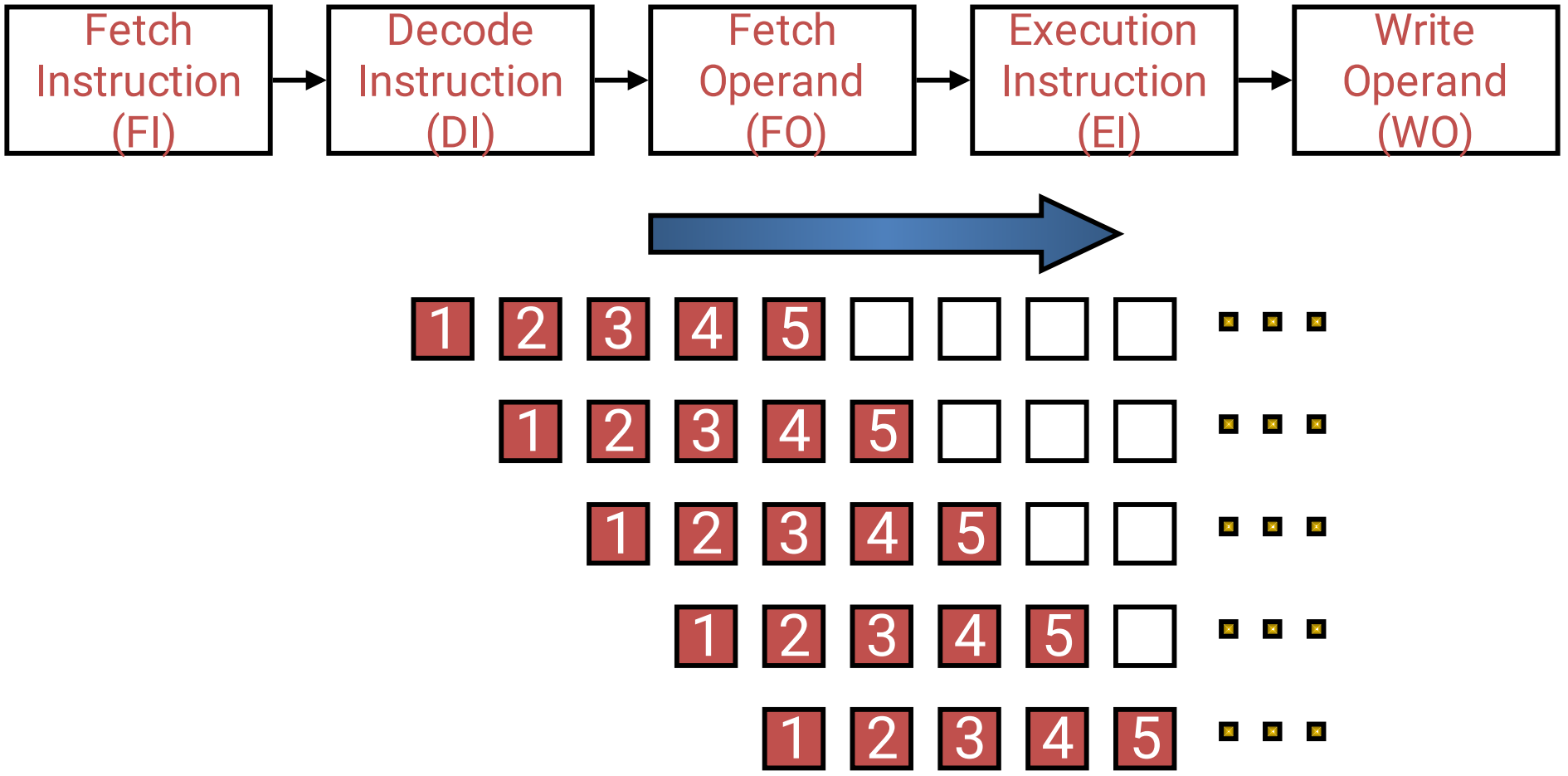


Instruction pipeline

Instructions seperate

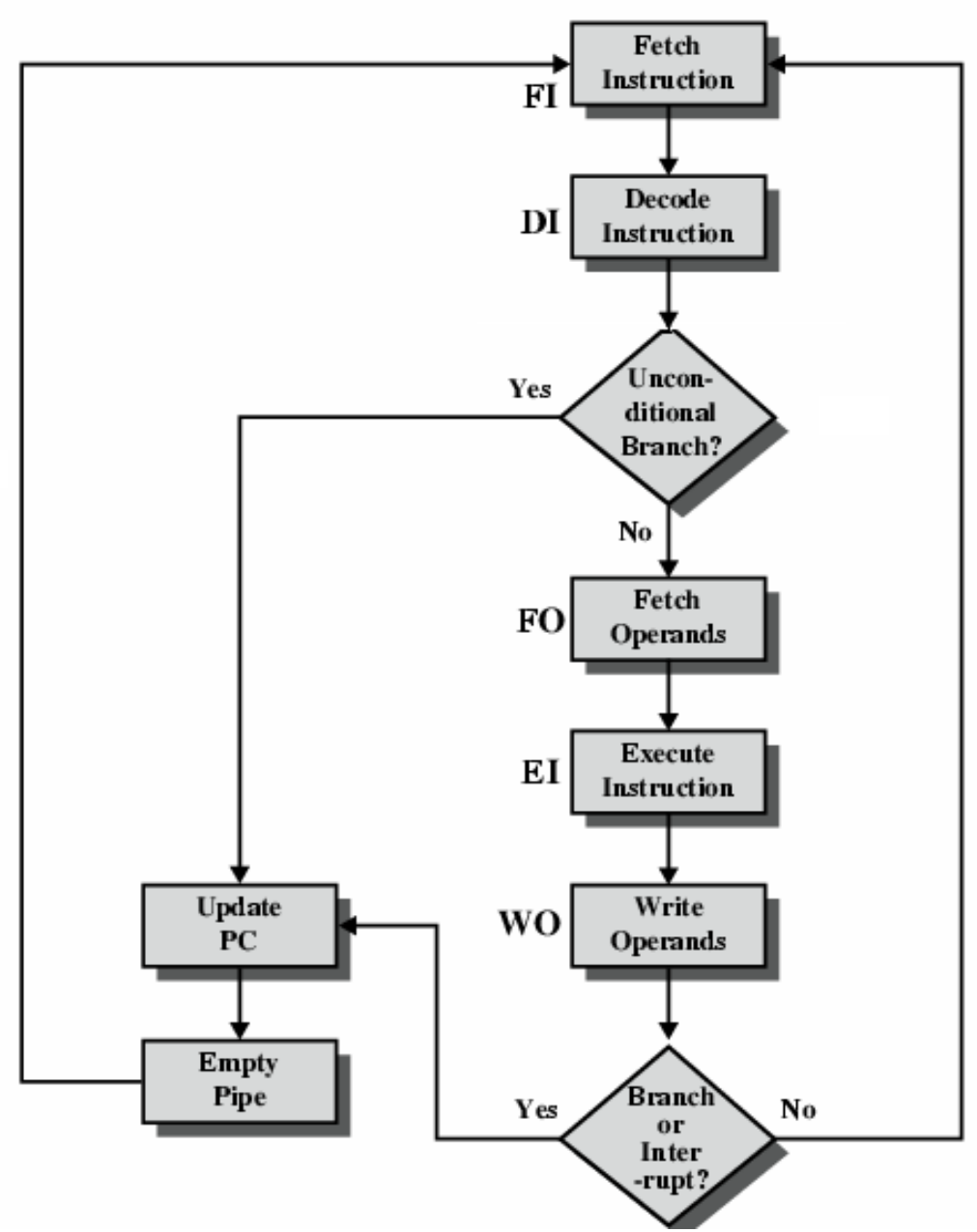
- 1. Fetch the instruction
- 2. Decode the instruction
- 3. Fetch the operands from memory
- 4. Execute the instruction
- 5. Store the results in the proper place

5-Stage Pipelining

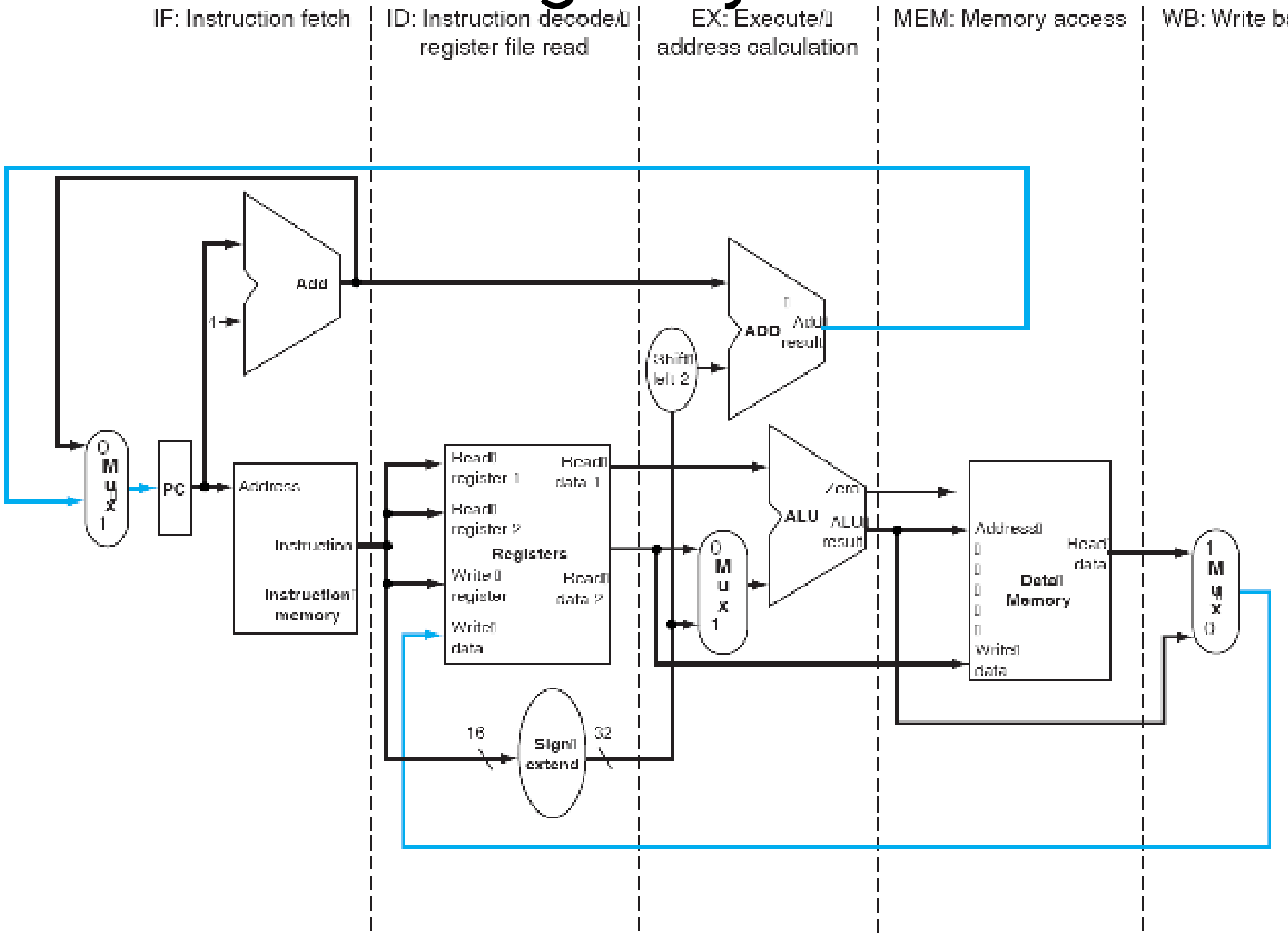


Five Stage Instruction Pipelin

- Fetch instruction
- Decode instruction
- Fetch operands
- Execute instructions
- Write result

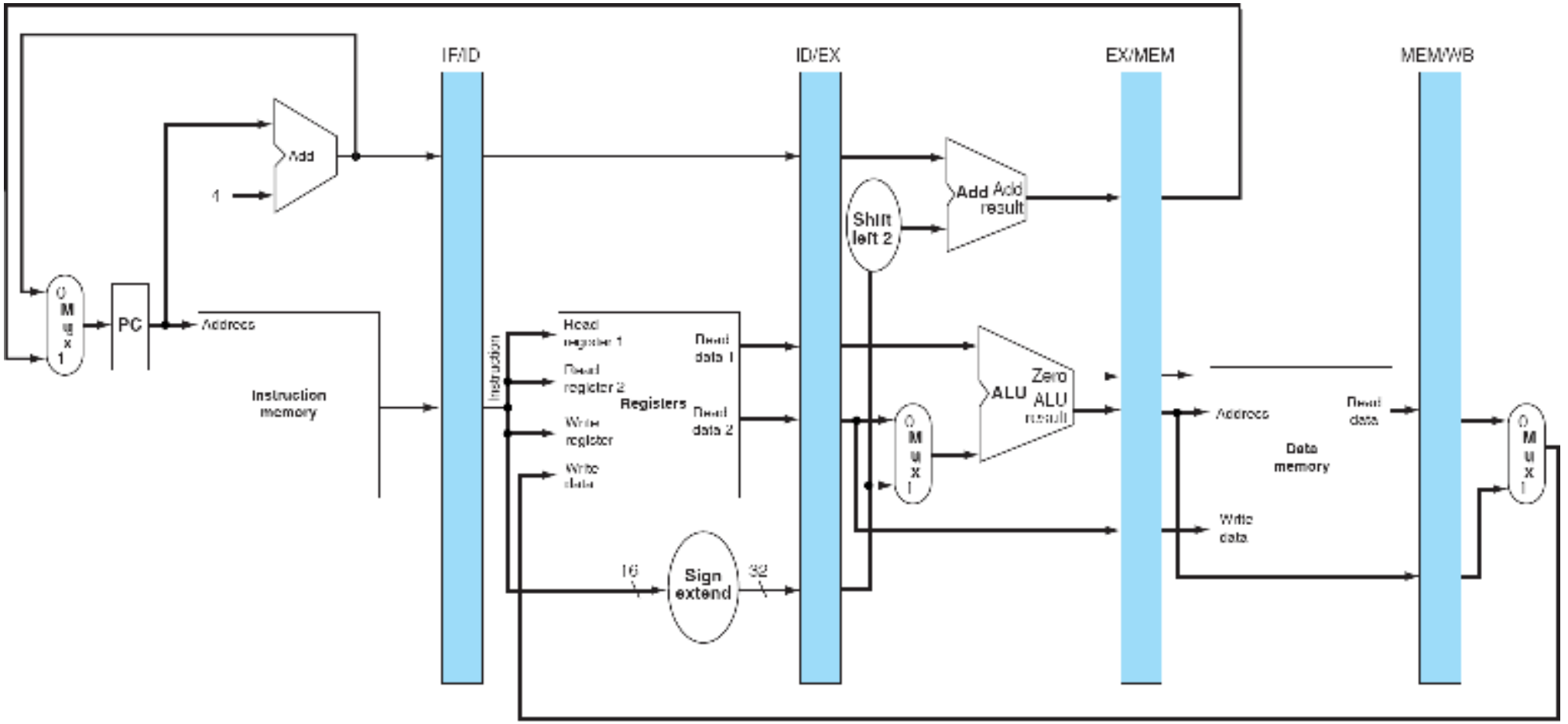


Review - Single-Cycle Processor



- *What do we need to add to actually split the datapath into stages?*

Basic Pipelined Processor

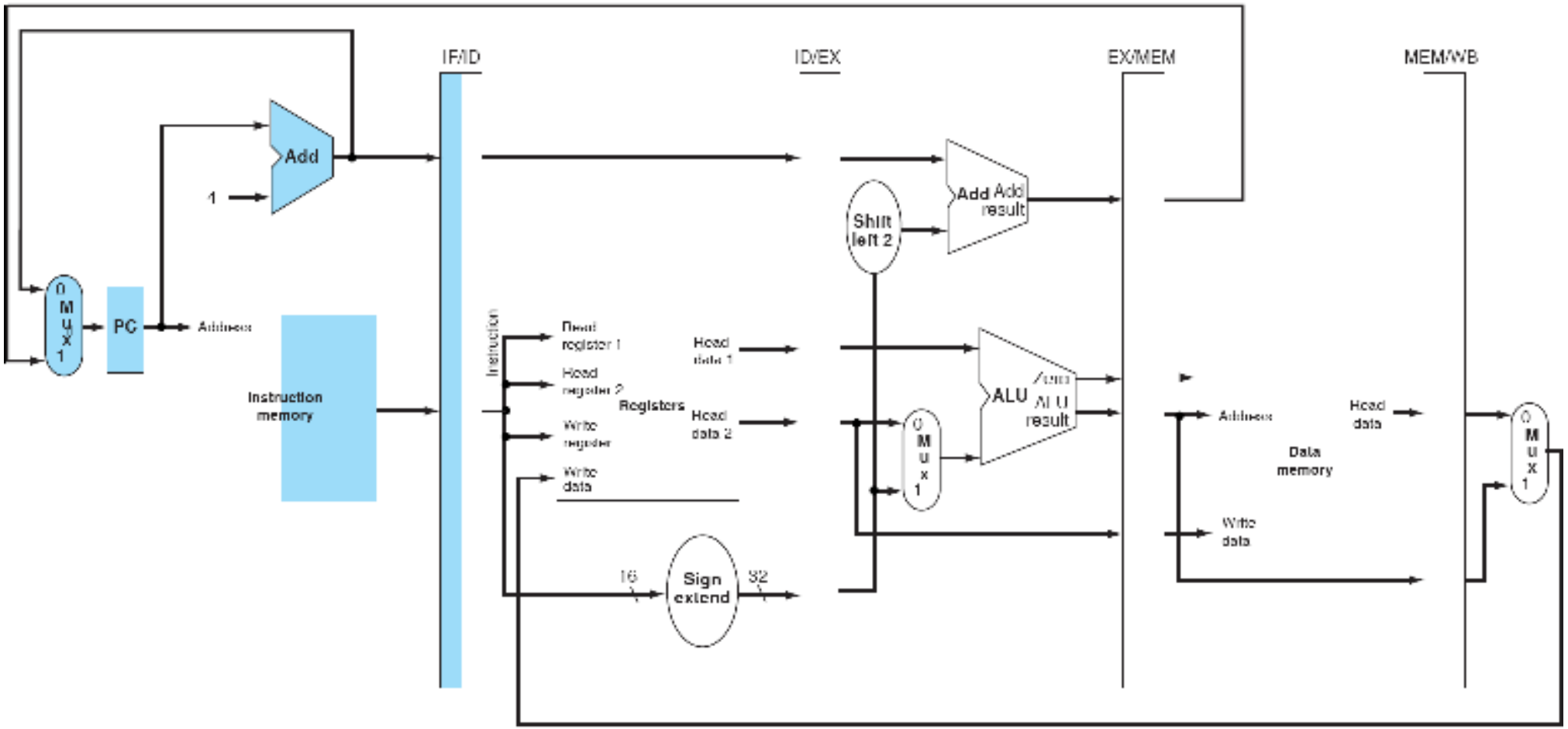


Pipeline example: lw

lw

Instruction fetch

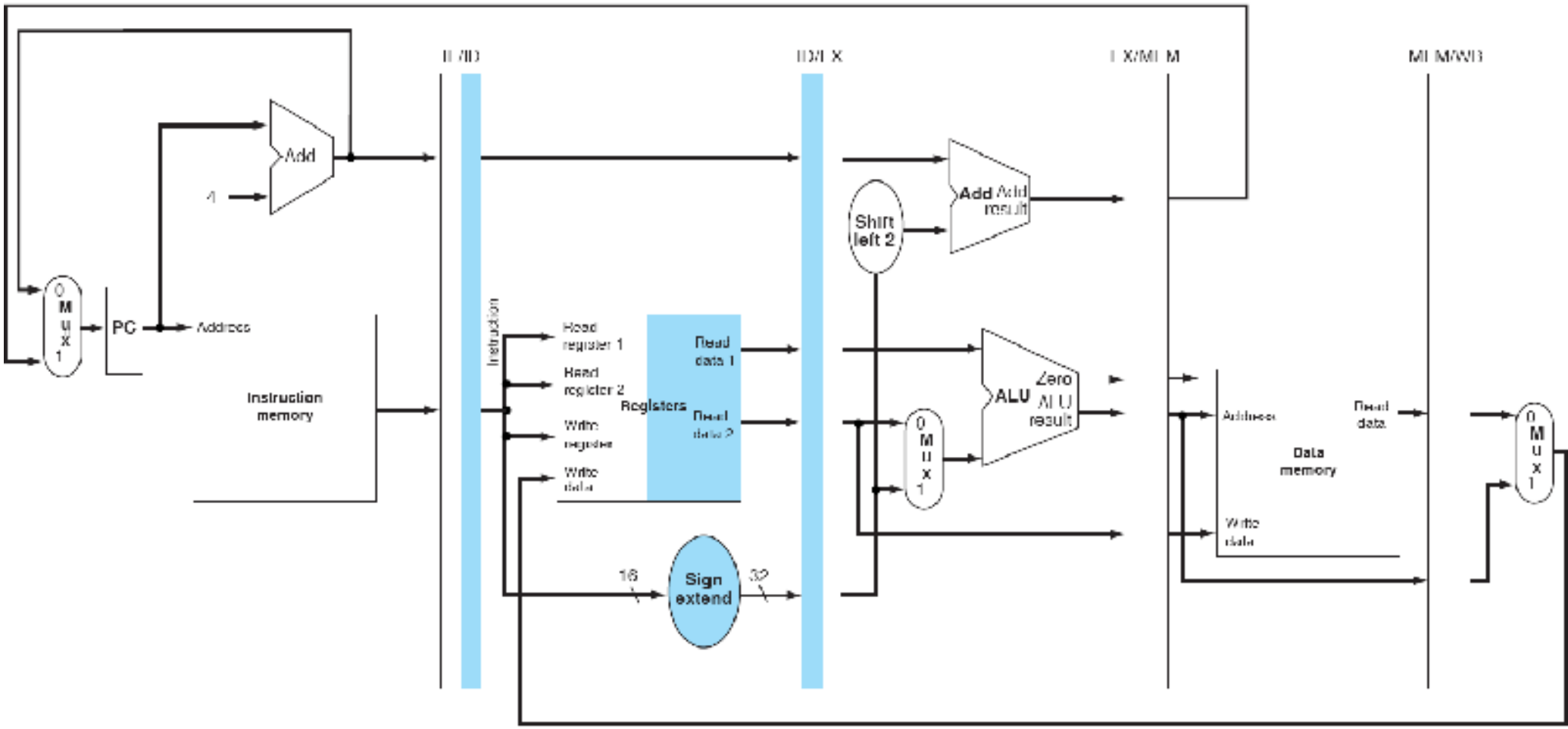
IF



Pipeline example: lw

lw ID

Instruction decode

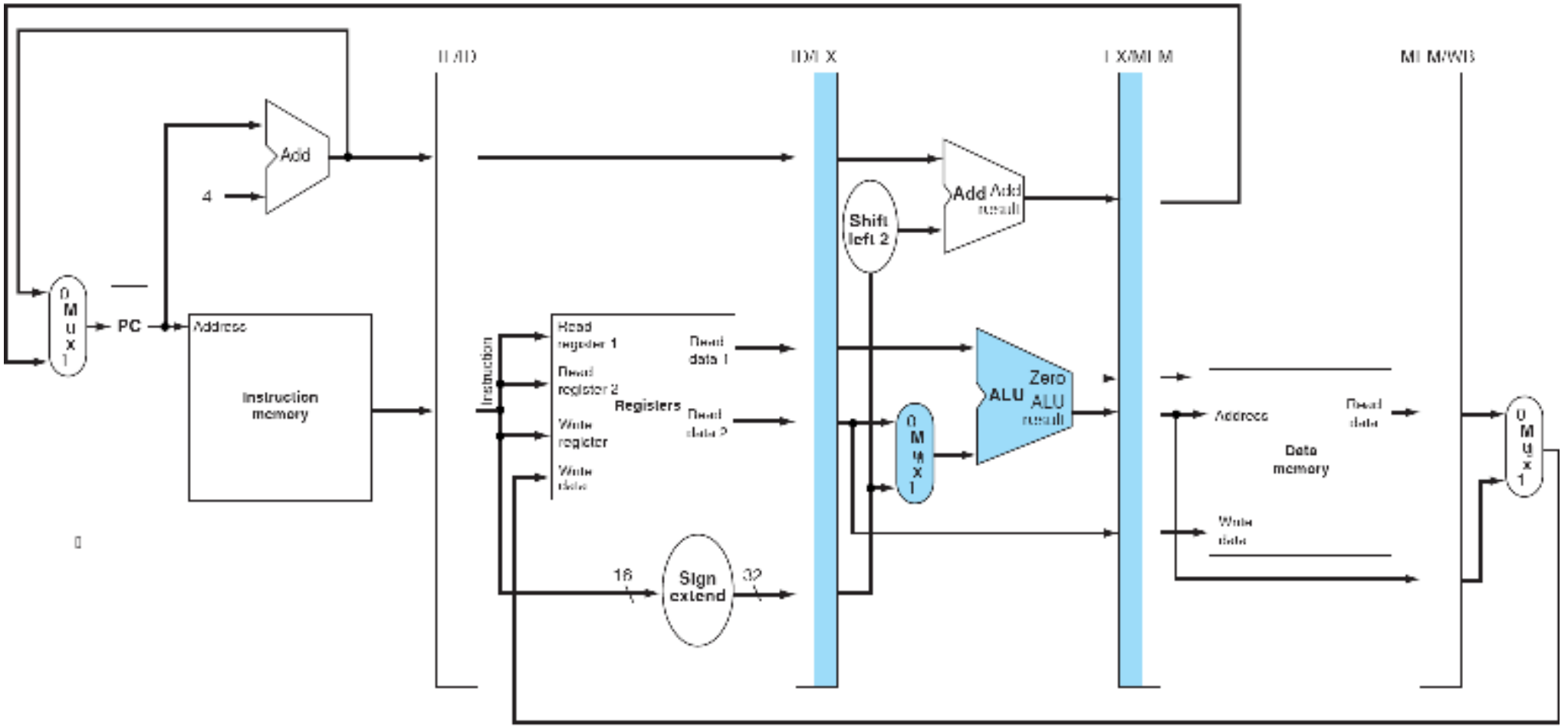


Pipeline example: lw

EX

lw

Execution

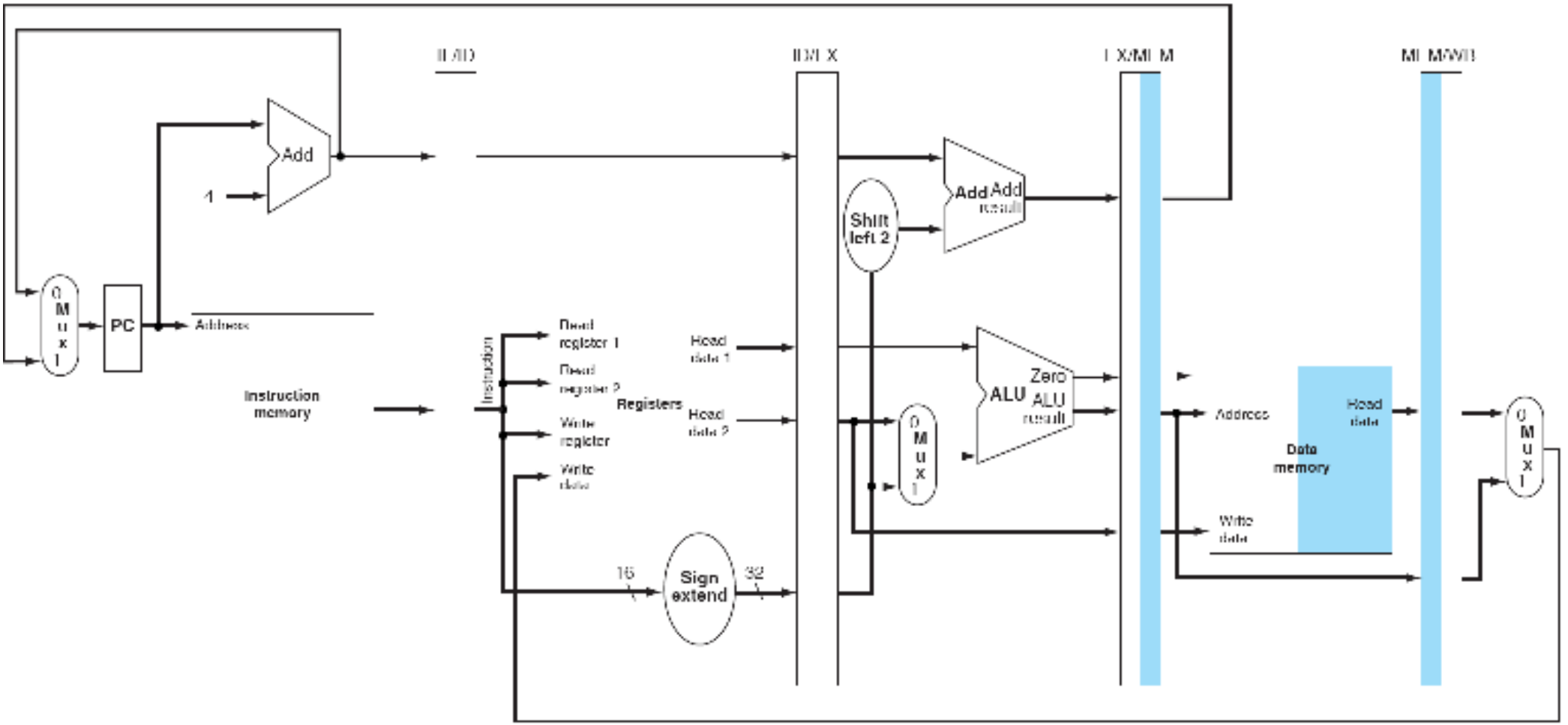


Pipeline example: lw

MEM

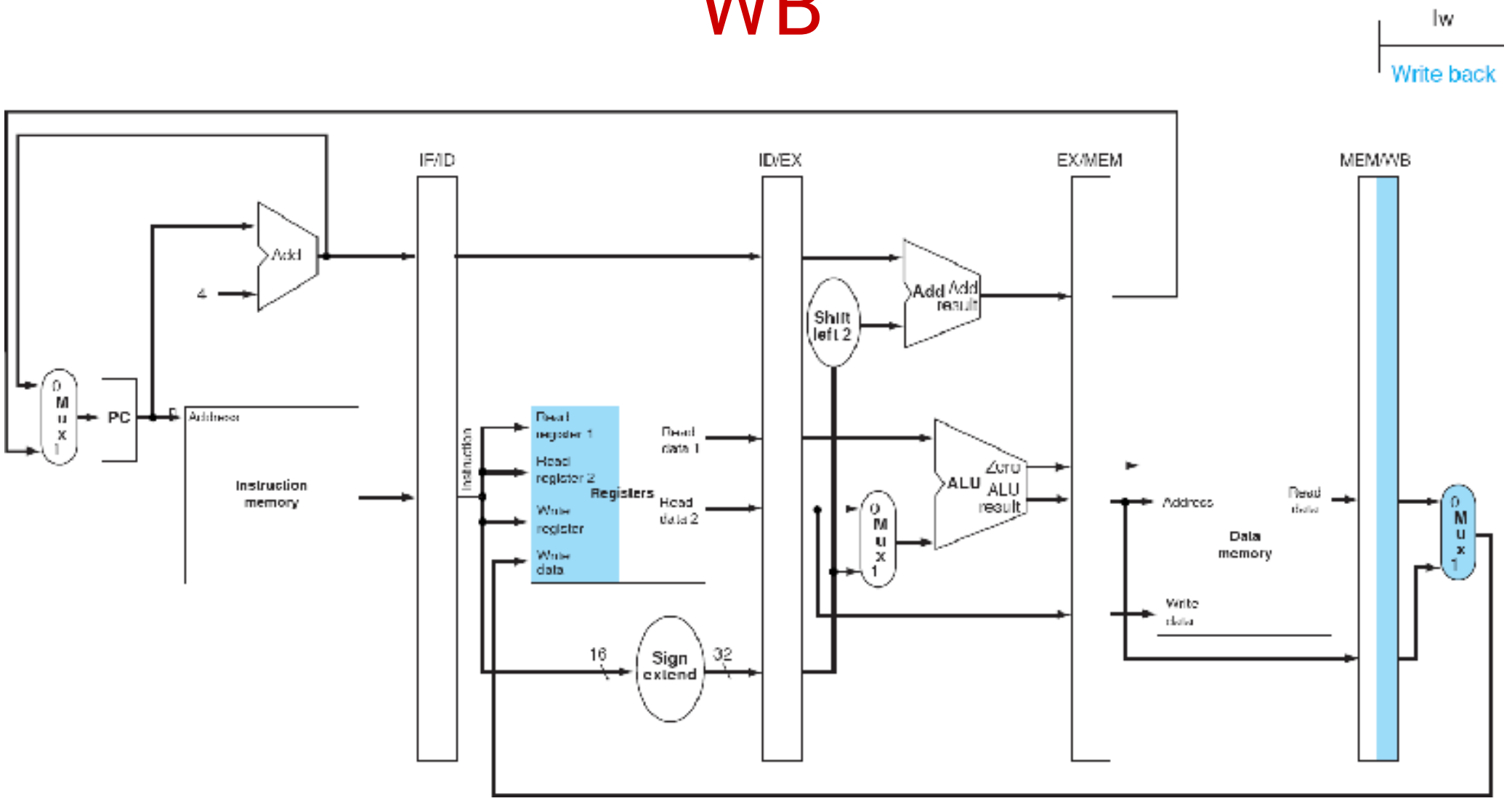
lw

Memory



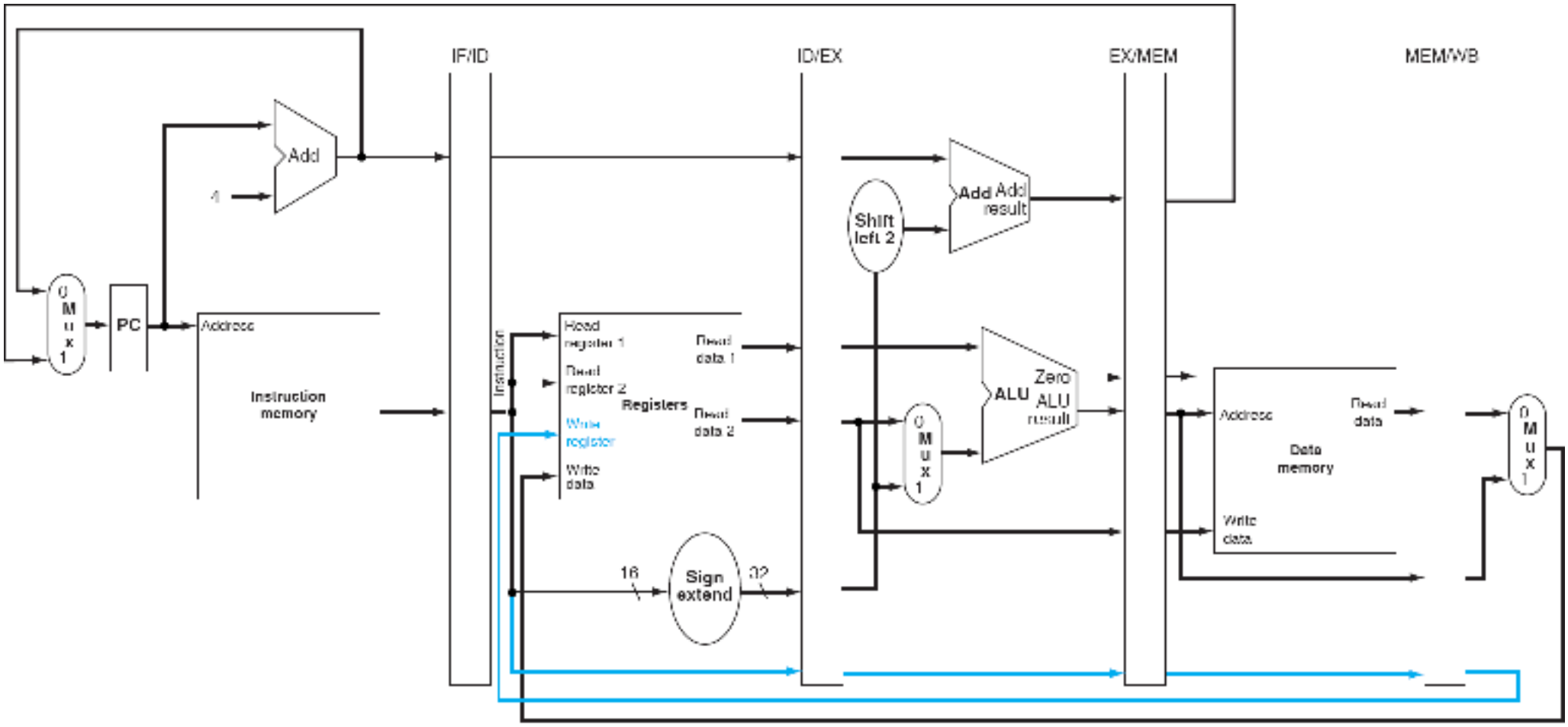
Pipeline example: lw

WB



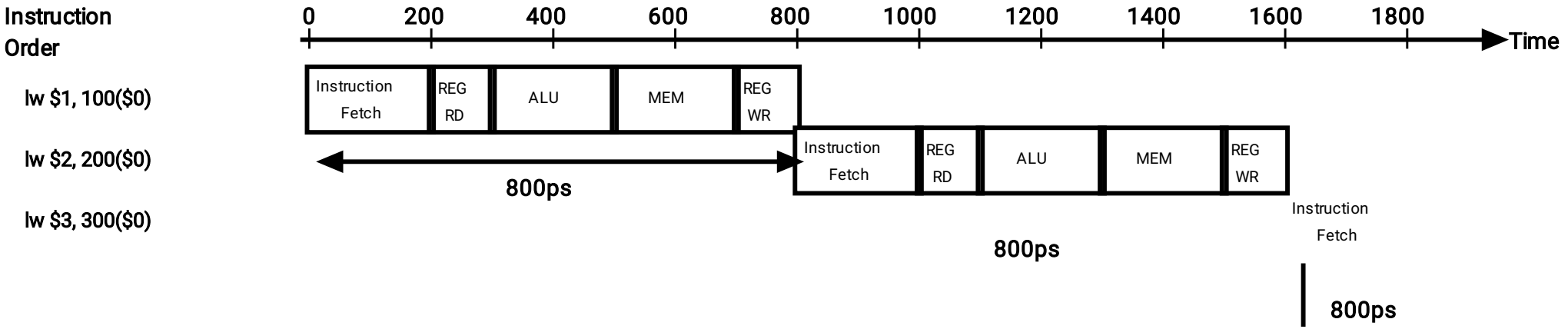
Can you find a problem?

Basic Pipelined Processor (Corrected)

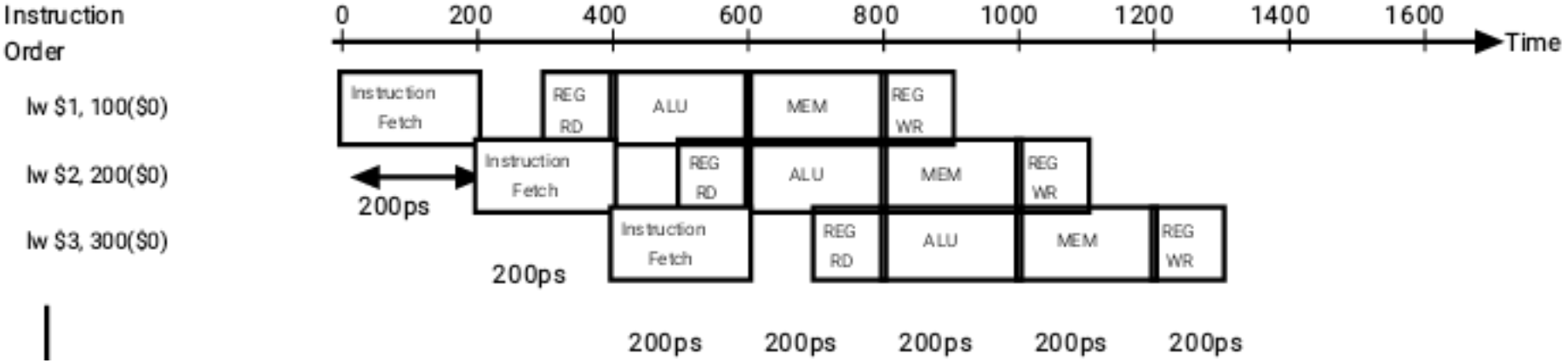


Single-Cycle vs. Pipelined Execution

Non-Pipelined



Pipelined



Difficulties...

- If a complicated memory access occurs in stage 1, stage 2 will be delayed and the rest of the pipe is *stalled*.
- If there is a branch, **if.. and jump**, then some of the instructions that have already entered the pipeline should not be processed.
- We need to deal with these difficulties to keep the pipeline moving

Comments about Pipelining

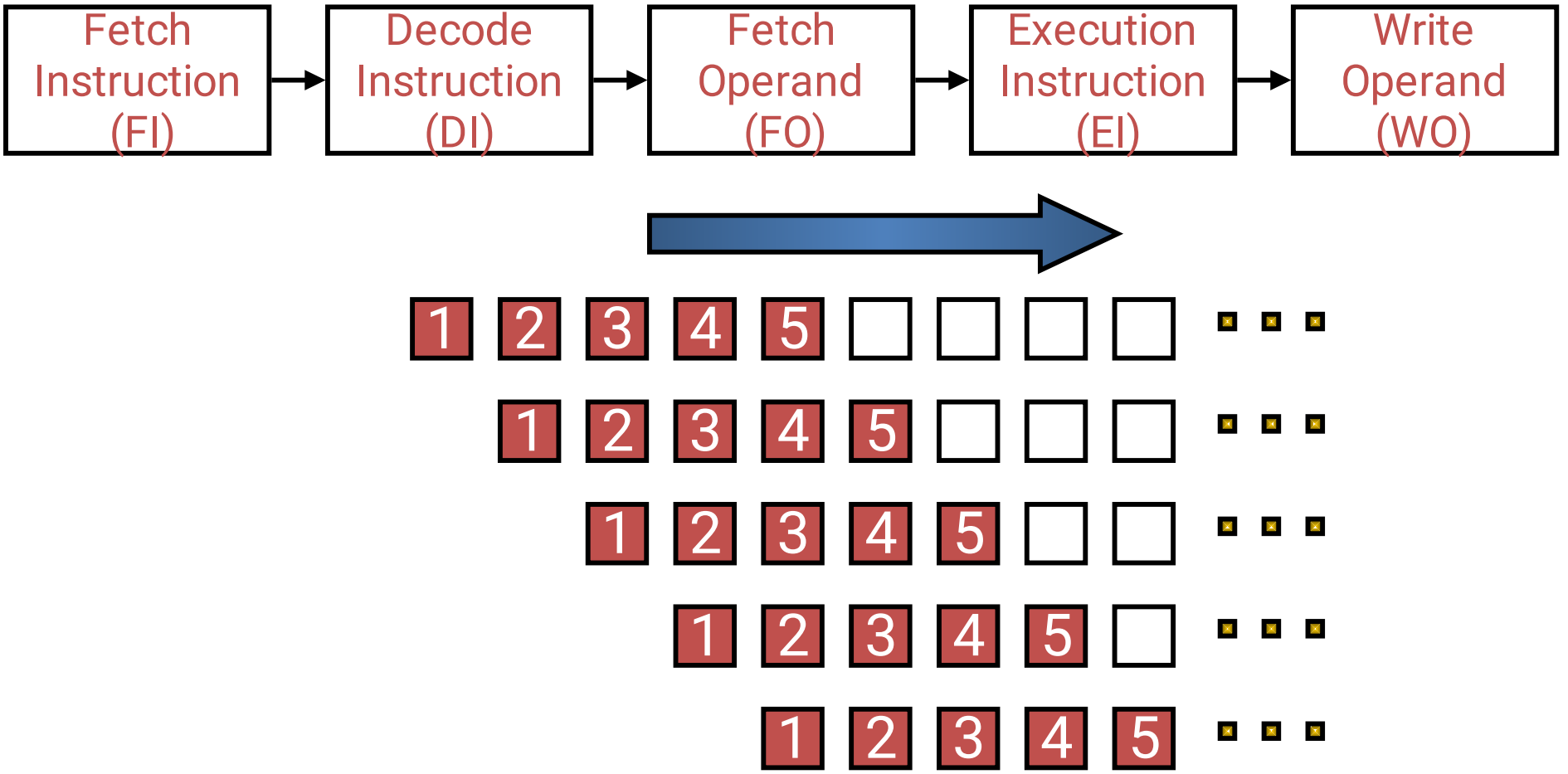
- The good news
  - Multiple instructions are being processed at same time
  - This works because stages are isolated by registers
  - Best case speedup of N
- The bad news
  - Instructions interfere with each other - hazards
    - Example: different instructions may need the same piece of hardware (e.g., memory) in same clock cycle
    - Example: instruction may require a result produced by an earlier instruction that is not yet complete

Pipeline Hazards

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: two different instructions use same h/w in same cycle
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Pipelining of branches & other instructions that change the PC

Structural hazard

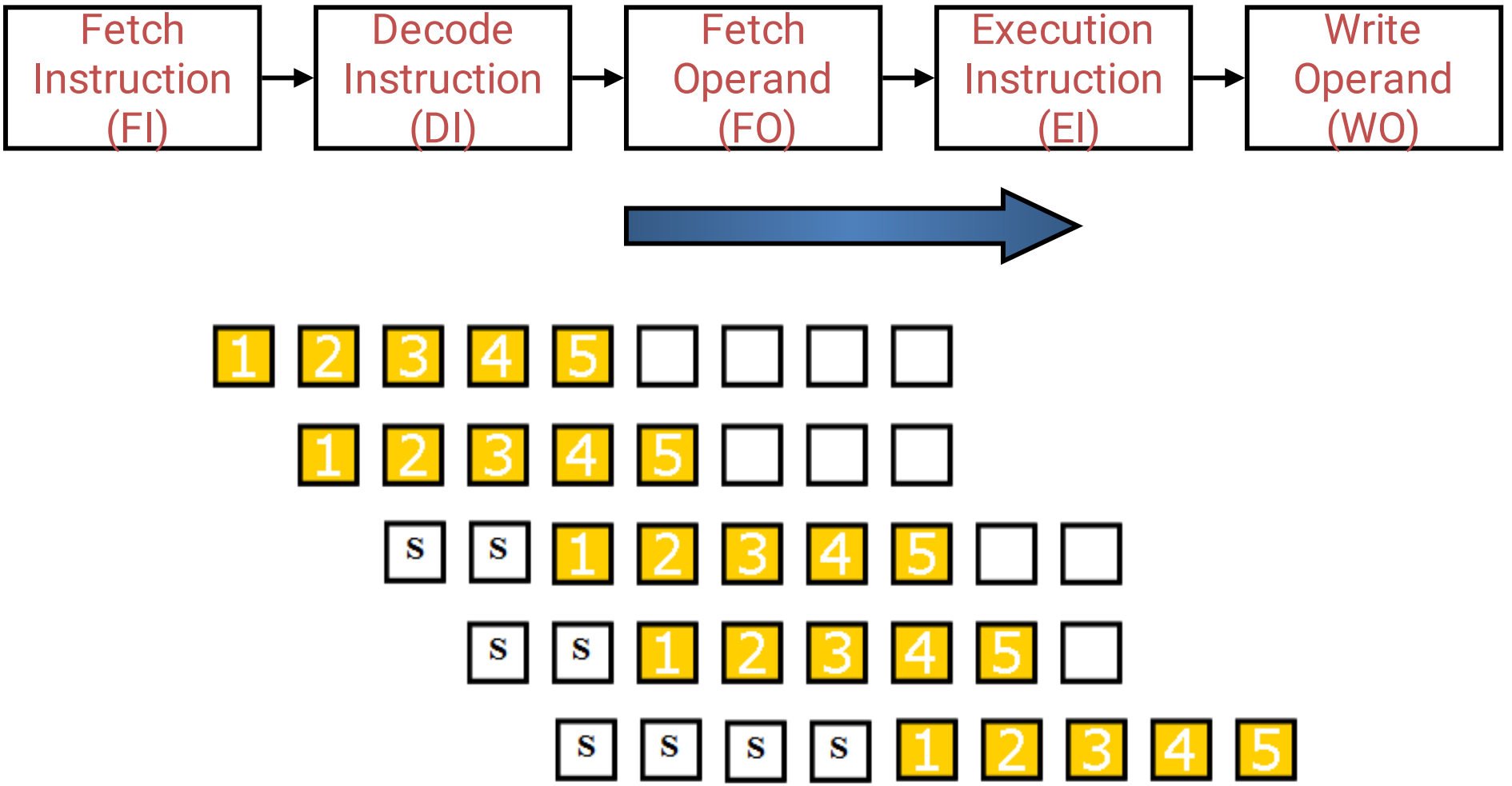
Memory data fetch requires on FI and FO



Structural hazard

- To solve this hazard, we “stall” the pipeline until the resource is freed
- A stall is commonly called pipeline bubble, since it floats through the pipeline taking space but carry no useful work

Structural hazard



Data hazard

Example:

ADD R1 R2 R3

SUB R4 R1-R5

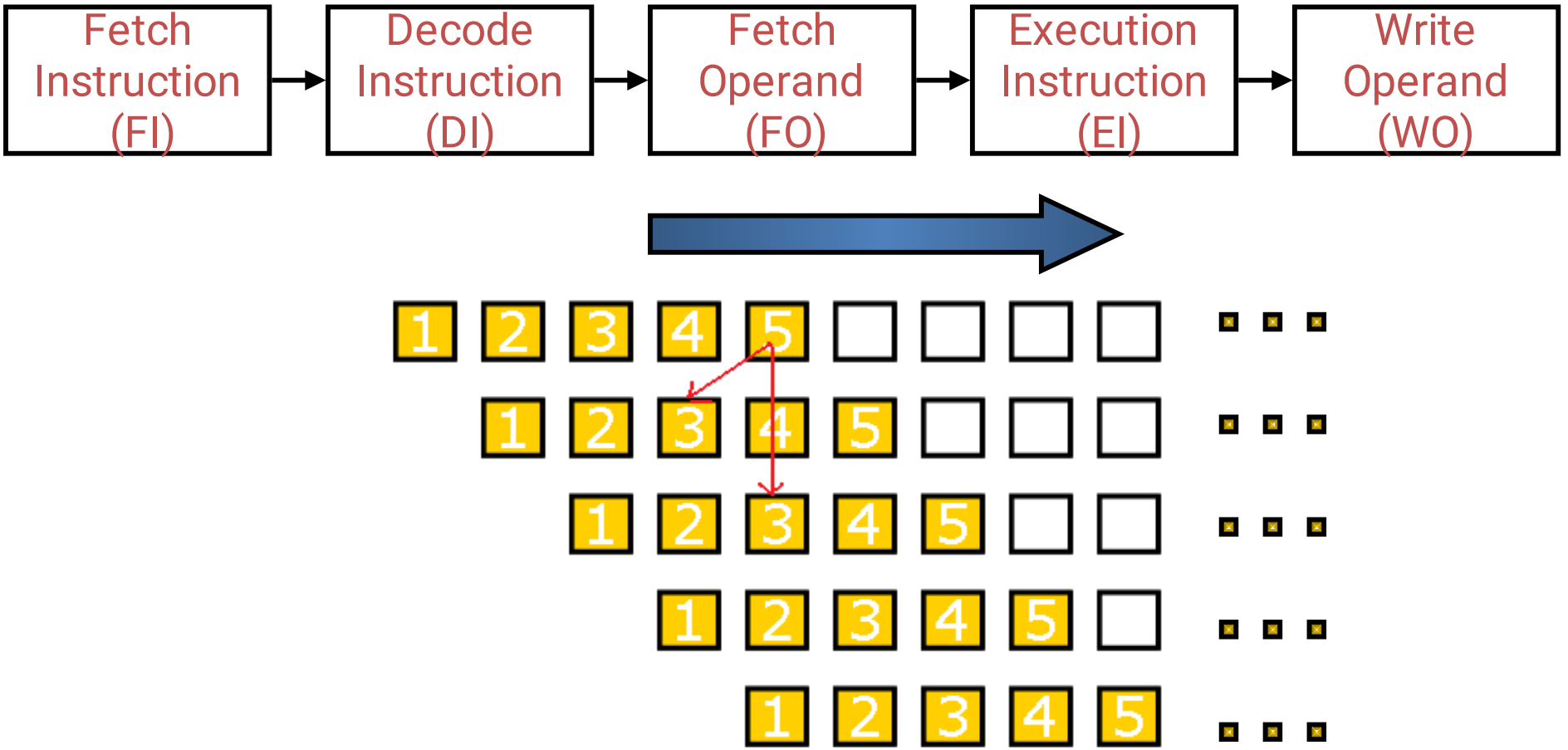
AND R6 R1 AND R7

OR R8 R1 OR R9

XOR R10 R1 XOR R11

Data hazard

FO: fetch data value WO: store the executed value

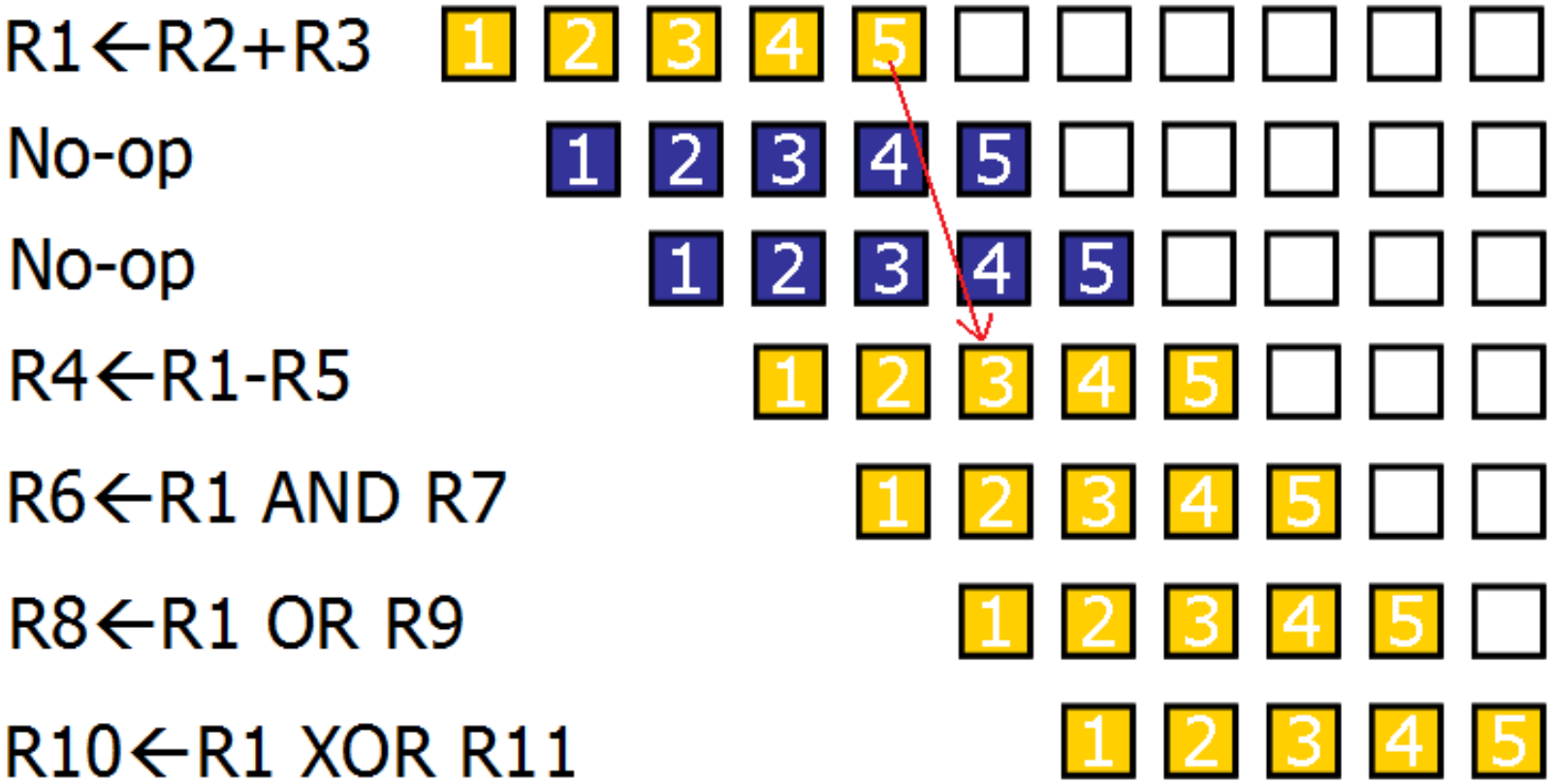


Data hazard

- Delay load approach inserts a no-operation instruction to avoid the data conflict

```
ADD     R1□R2□R3
No-op
No-op
SUB     R4□R1-R5
AND     R6□R1 AND R7
OR     R8□R1 OR R9
XOR       R10□R1 XOR R11
```

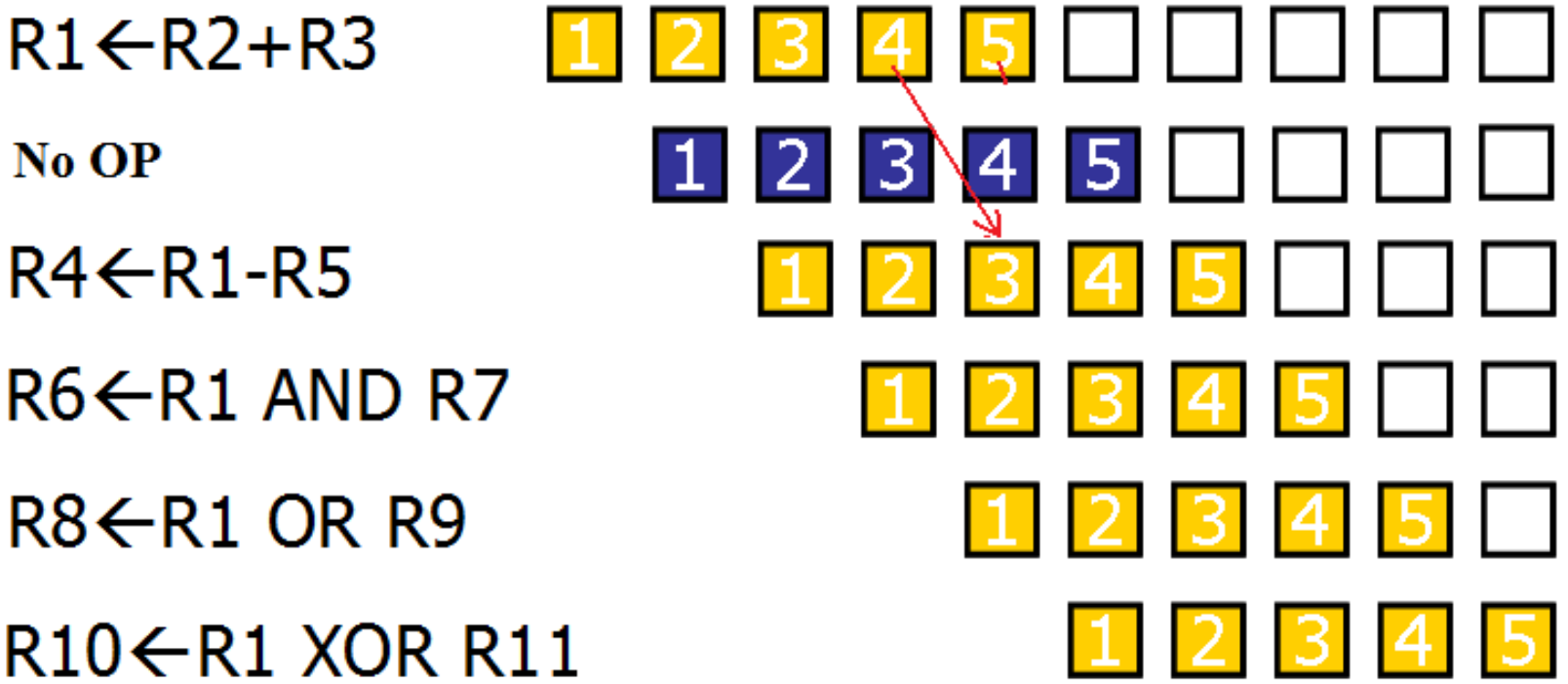
Data hazard


R1←R2+R3
No-op
No-op
R4←R1-R5
R6←R1 AND R7
R8←R1 OR R9
R10←R1 XOR R11
1 2 3 4 5

Data hazard

- It can be further solved by a simple hardware technique called *forwarding* (also called *bypassing* or *short-circuiting*)
- The insight in forwarding is that the result is not really needed by SUB until the ADD execute completely
- If the forwarding hardware detects that the previous ALU operation has written the register corresponding to a source for the current ALU operation, control logic selects the results in ALU instead of from memory

Data hazard



Branch hazards

- Branch hazards can cause a greater performance loss for pipelines
- When a branch instruction is executed, it may or may not change the PC
- If a branch changes the PC to its target address, it is a *taken* branch
- Otherwise, it is *untaken*

Branch hazards

- There are **FOUR** schemes to handle branch hazards
  - Freeze scheme
  - Predict-untaken scheme
  - Predict-taken scheme
  - Delayed branch

Summary - Pipelining Overview

- Pipelining increase throughput (but not latency)
- Hazards limit performance
  - Structural hazards
  - Control hazards
  - Data hazards