CENG311 Computer Architecture

Pipelining

IZTECH, Fall 2023

07 December 2023

Single-Cycle Implementation

The clock cycle must have the same length for every instruction

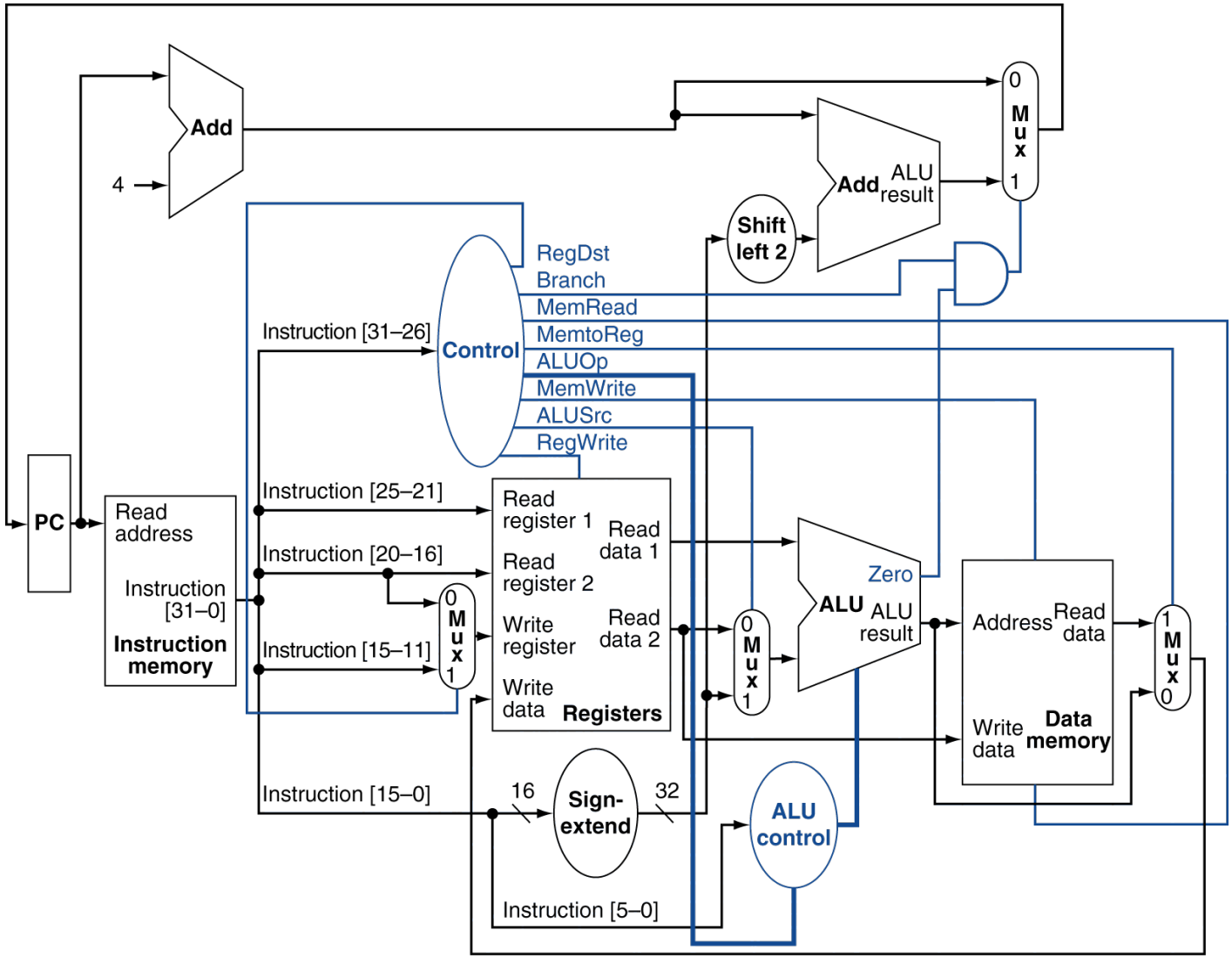
Longest delay determines clock period

- Critical path: Load instruction
- Instruction memory → register file → ALU → data memory → register file

CPI is 1, but the clock cycle is too long

We will improve performance by pipelining

Single-cycle CPU Datapath



Pipelining

Divide the instruction processing cycle into distinct “stages” of processing

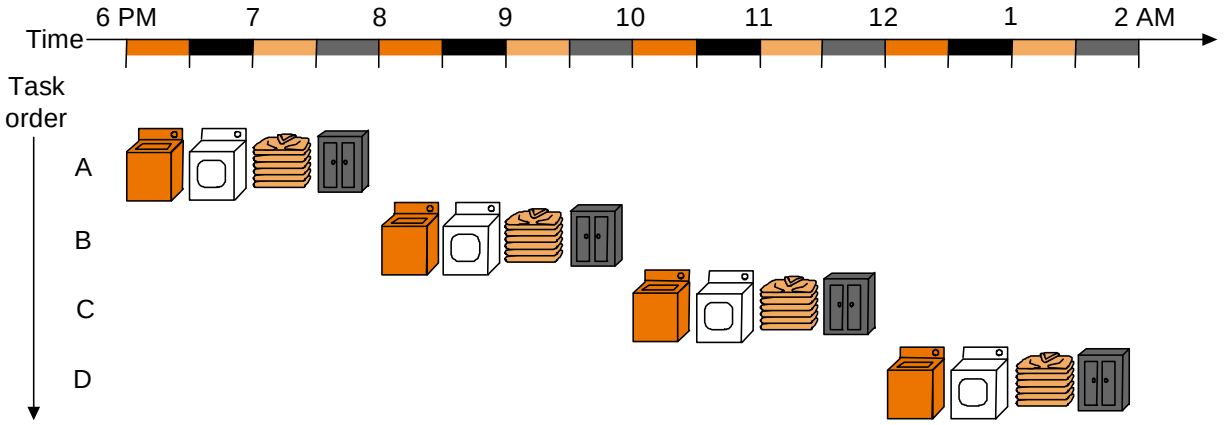
Ensure there are enough hardware resources to process one instruction in each stage

Process a different instruction in each stage

- Instructions consecutive in program order are processed in consecutive stages

Increases instruction processing throughput

Laundry Analogy



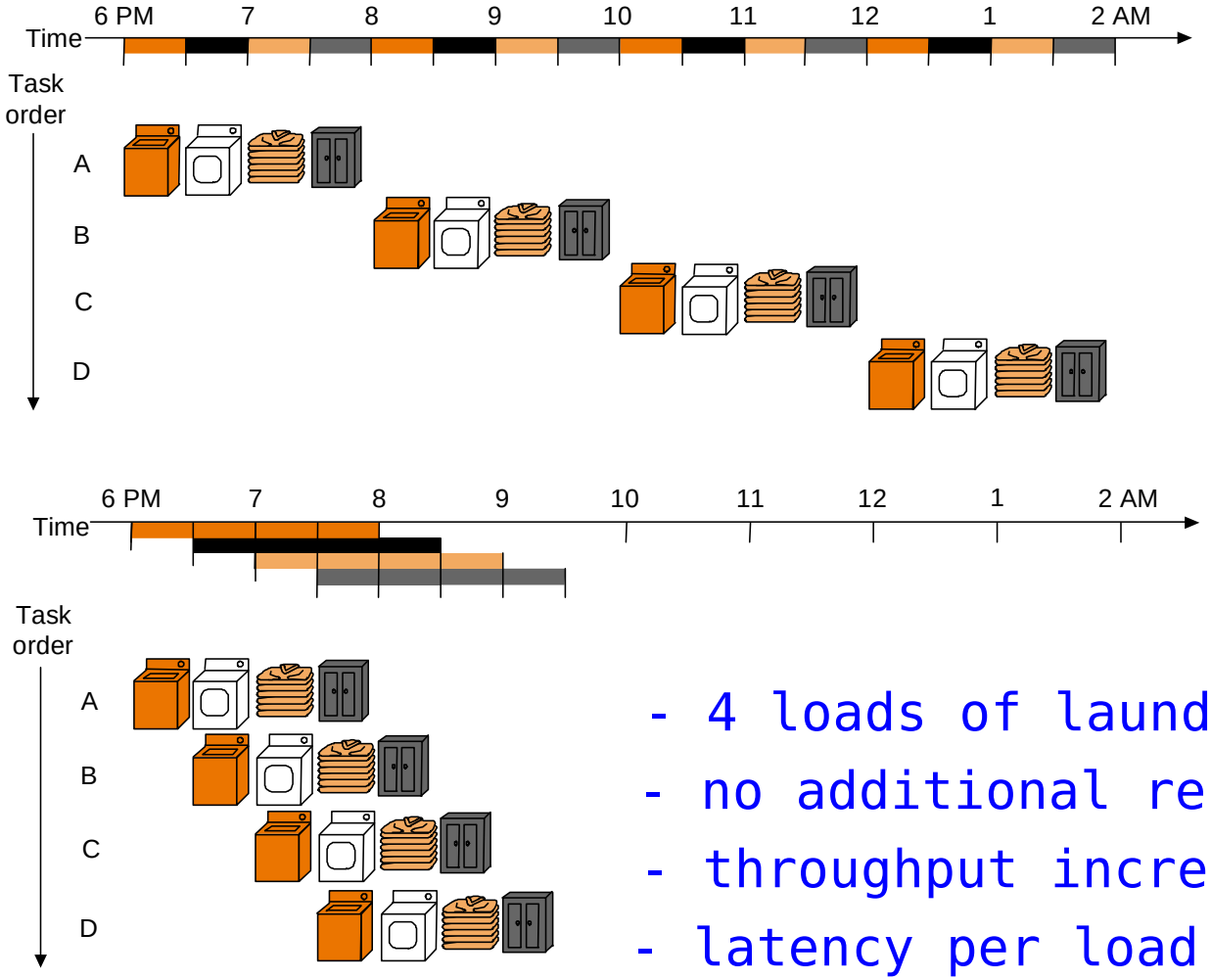
place one dirty load of clothes in the washer

when the washer is finished, place the wet load in the dryer

when the dryer is finished, take out the dry load and fold

when folding is finished, put the clothes away

Pipelining Laundry



- 4 loads of laundry in parallel
- no additional resources
- throughput increased by ~4
- latency per load is the same

MIPS Pipeline

Five stages, one step per stage

IF: Instruction fetch from memory

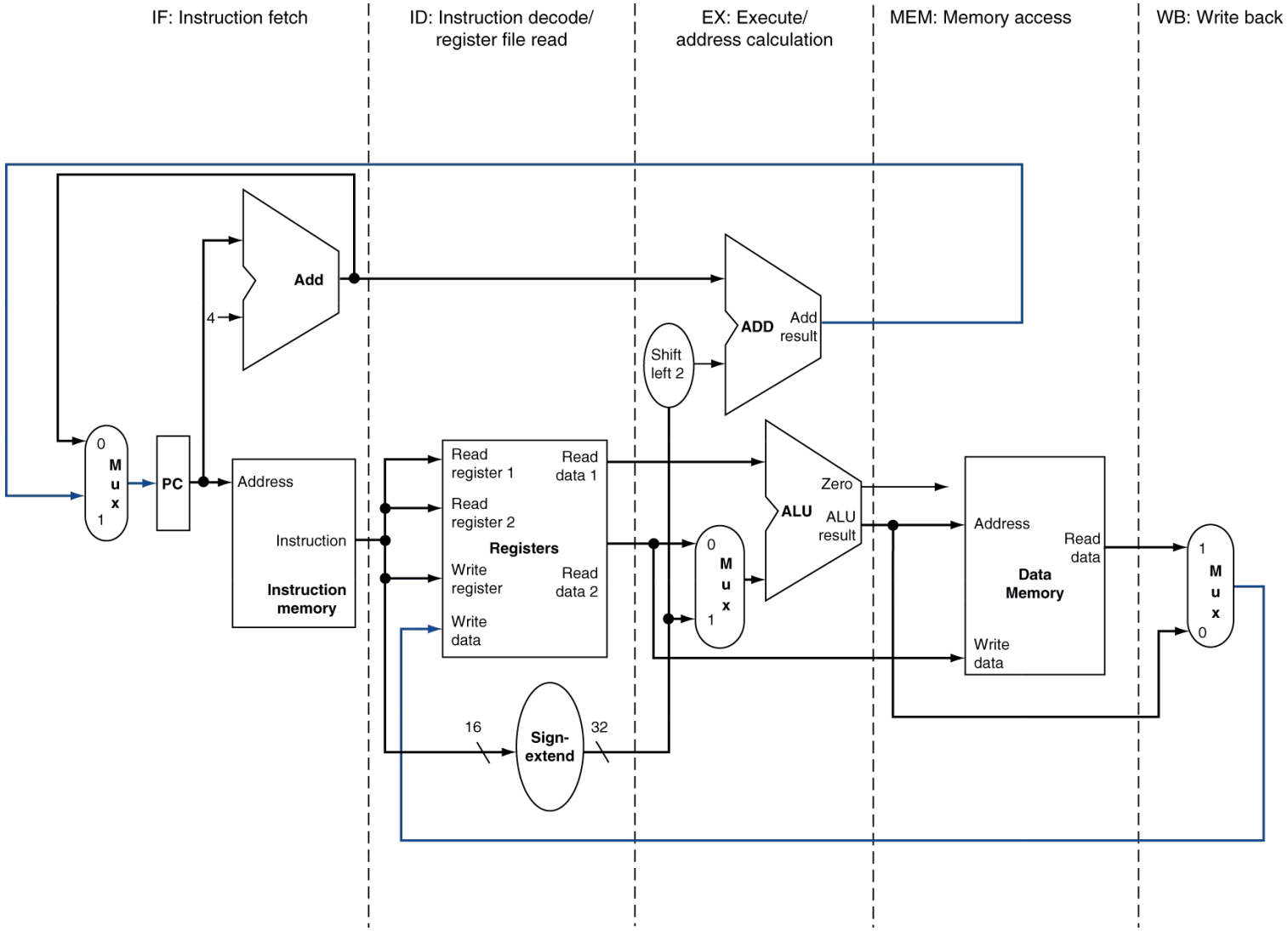
ID: Instruction decode & register read

EX: Execute operation or calculate address

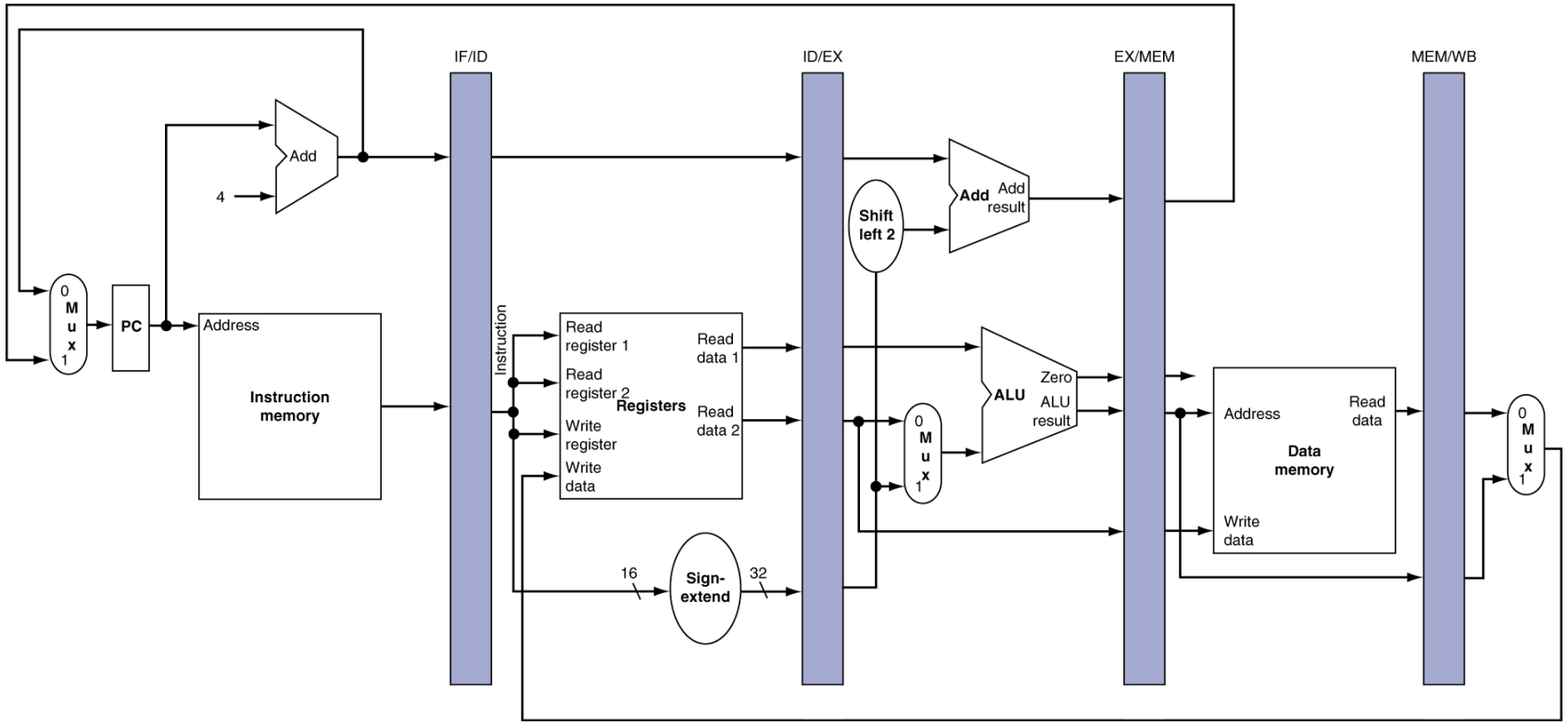
MEM: Access memory operand

WB: Write result back to register

MIPS Pipelined Datapath

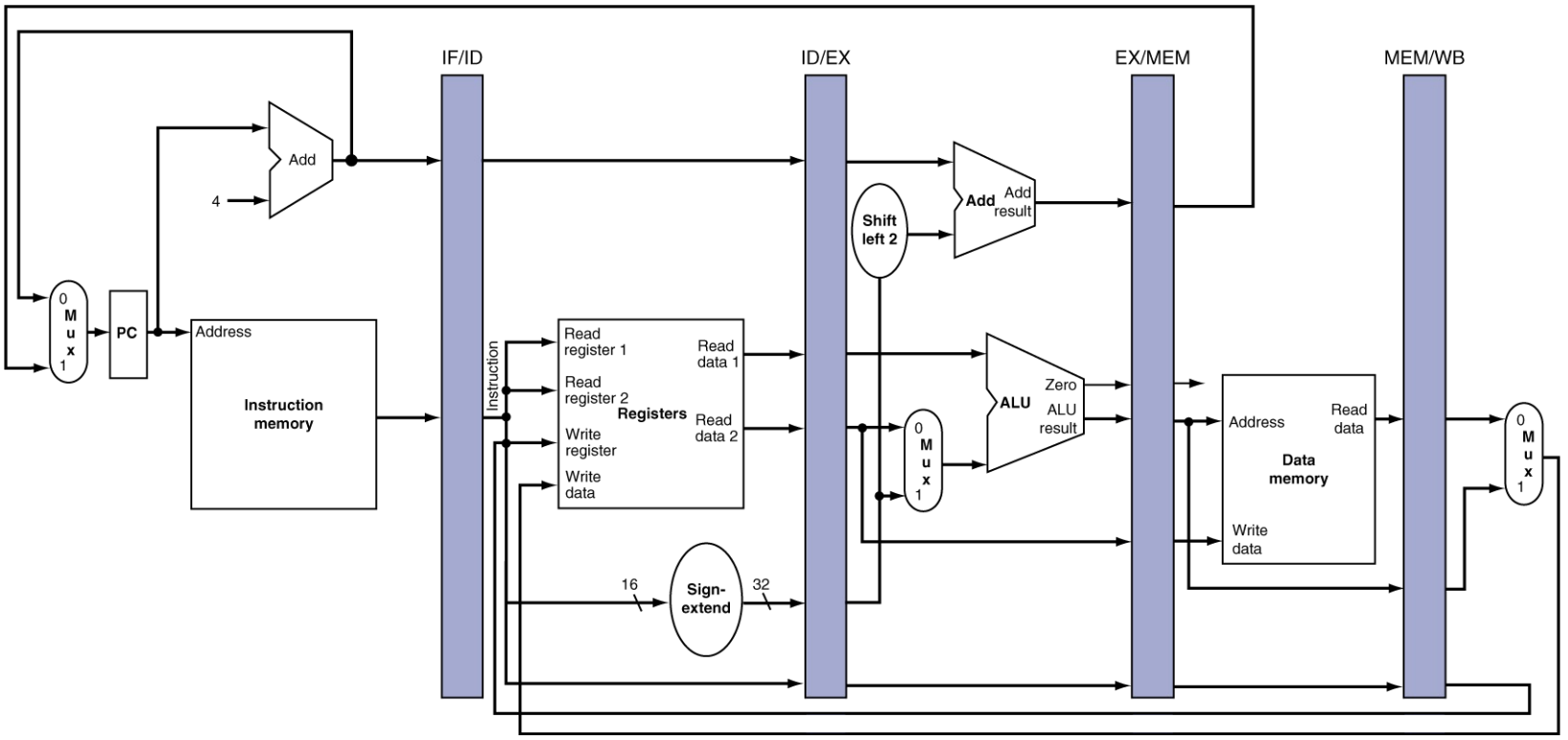


Pipeline Registers

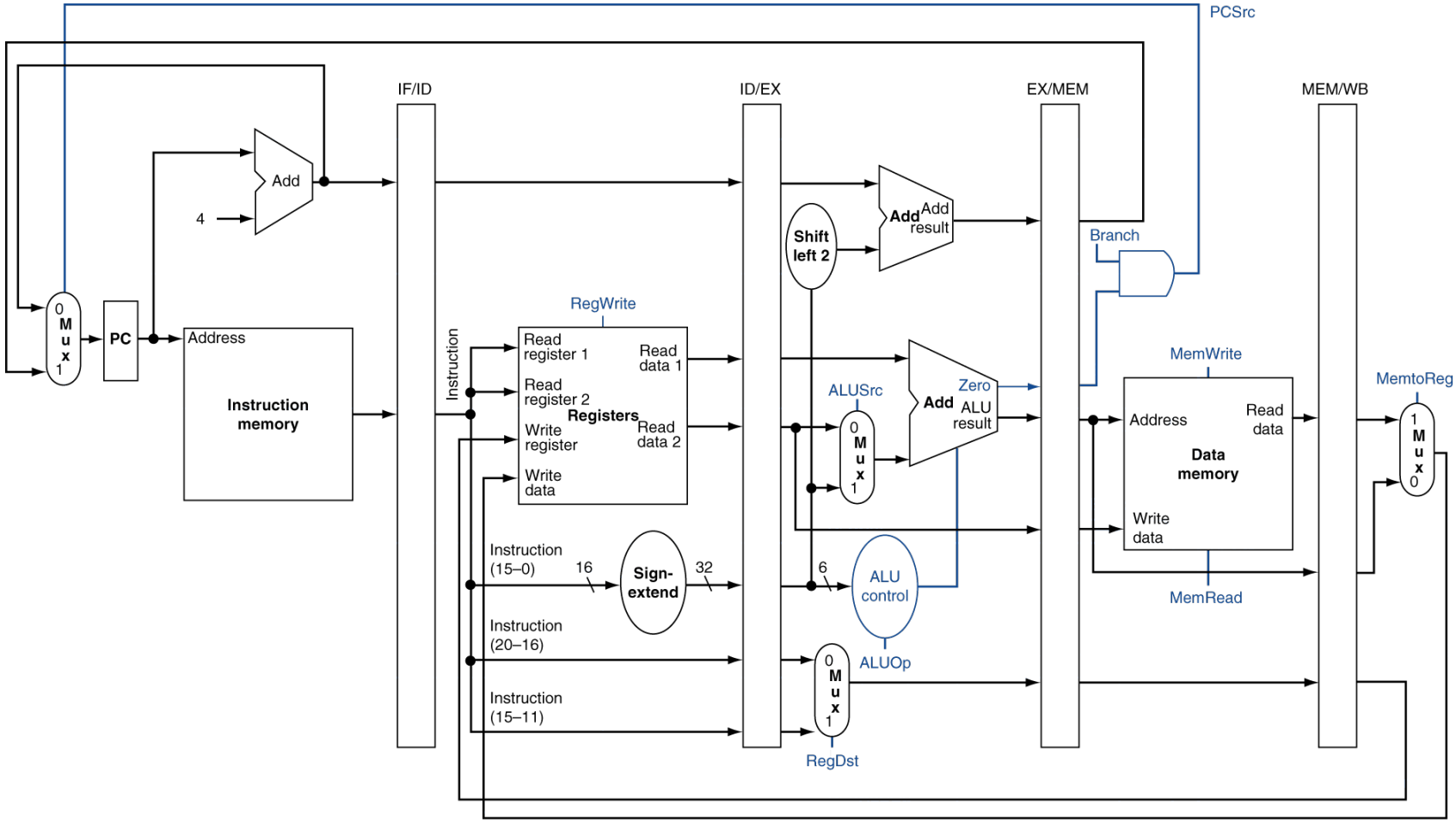


State of Pipeline in a Given Cycle

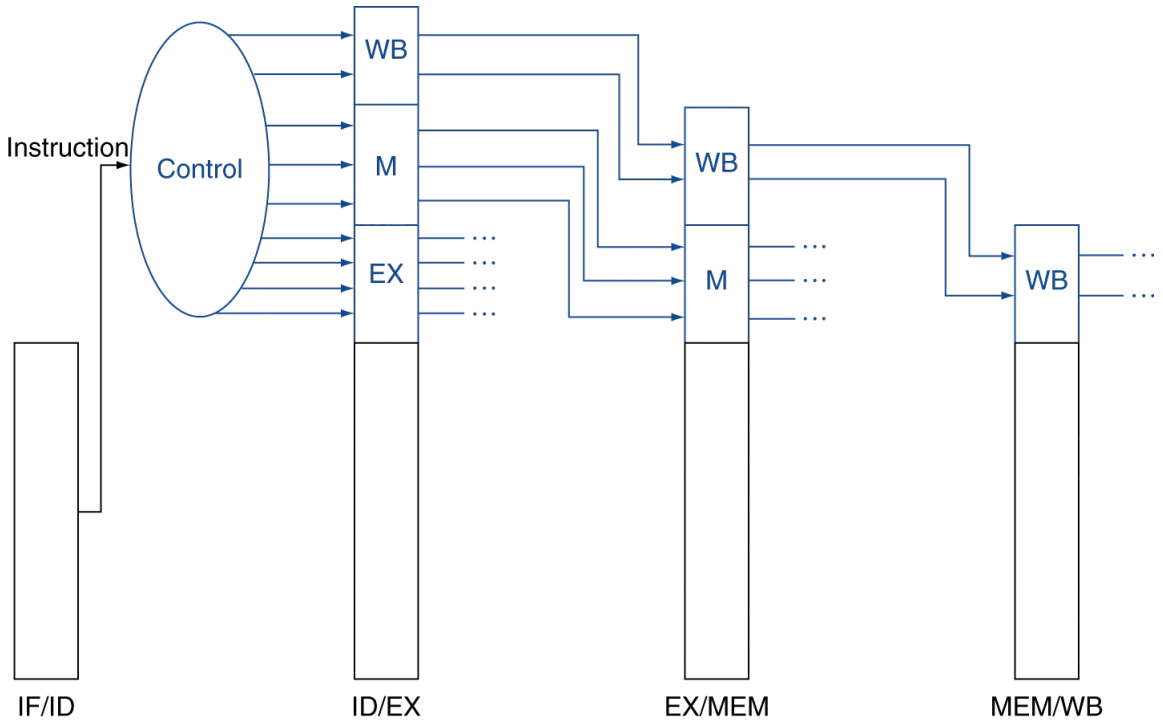
add $14, $5, $6	lw $13, 24 ($1)	add $12, $3, $4	sub $11, $2, $3	lw $10, 20($1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back



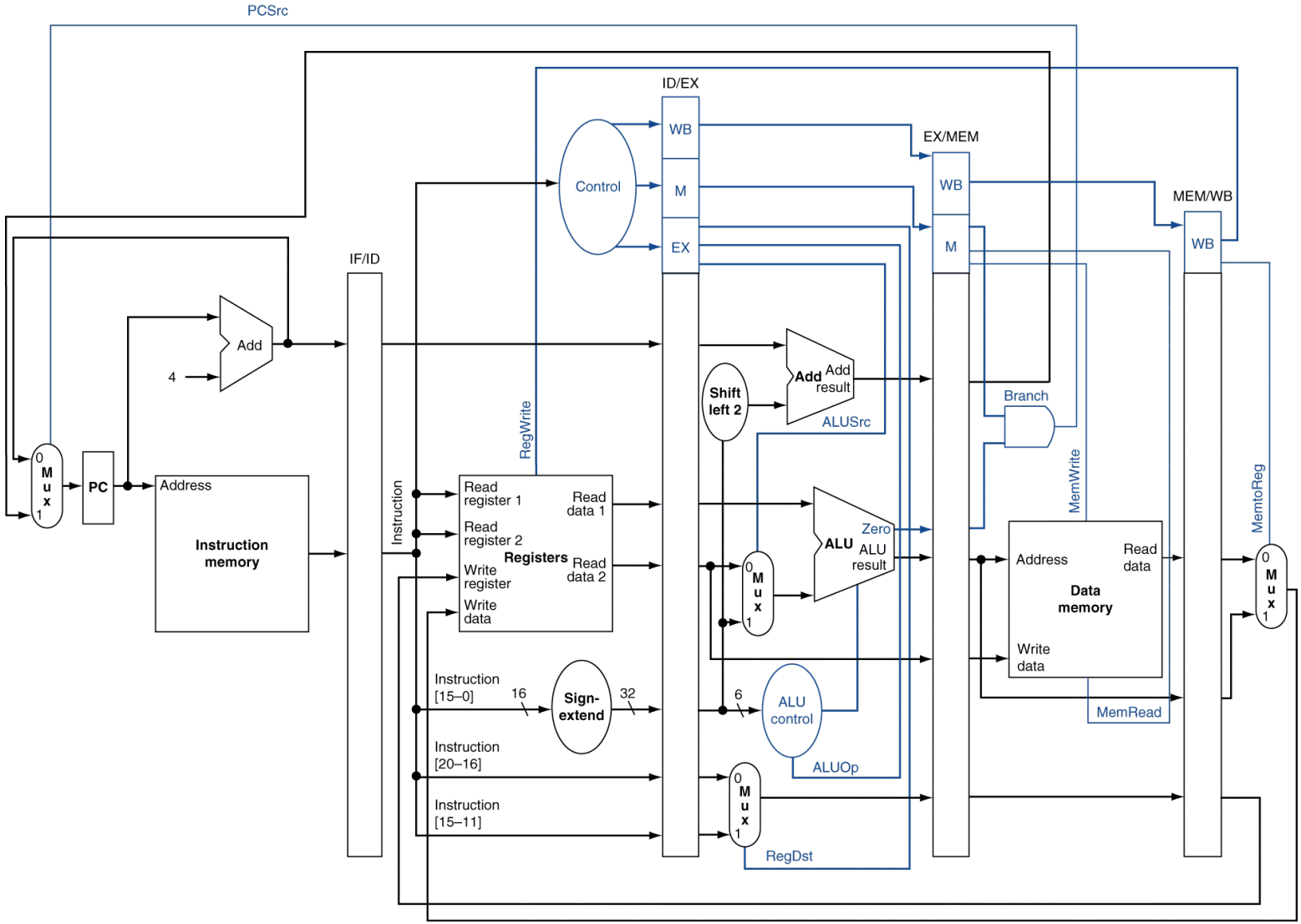
Pipelined Datapath



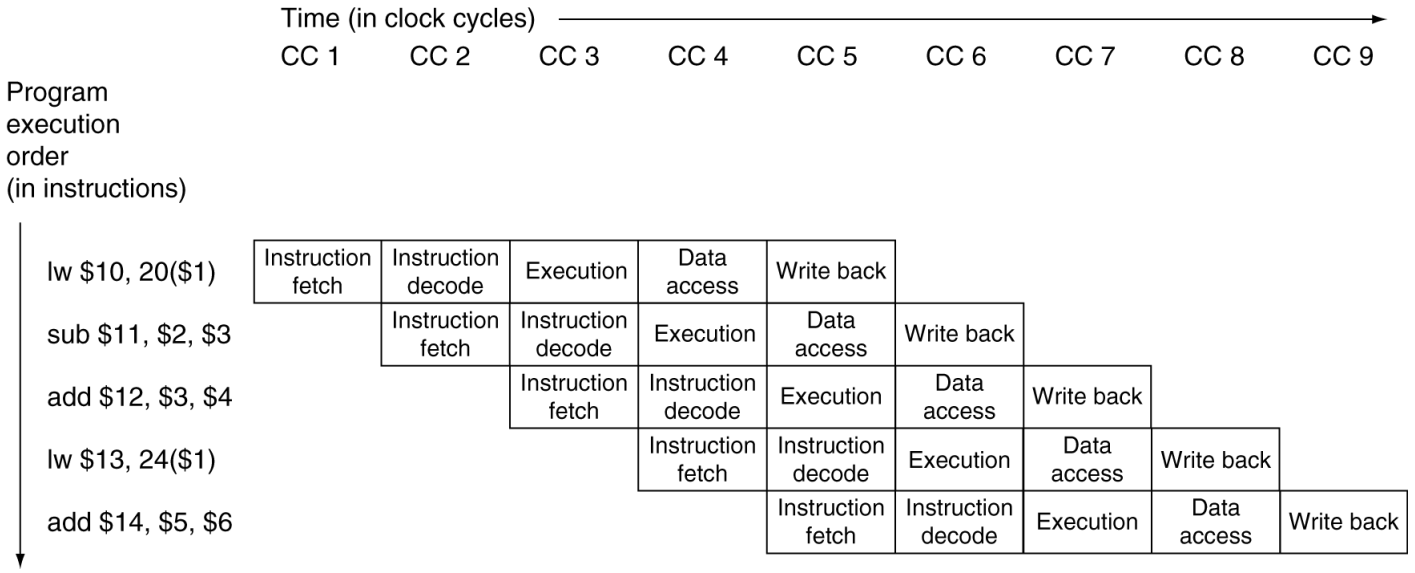
Pipelined Control



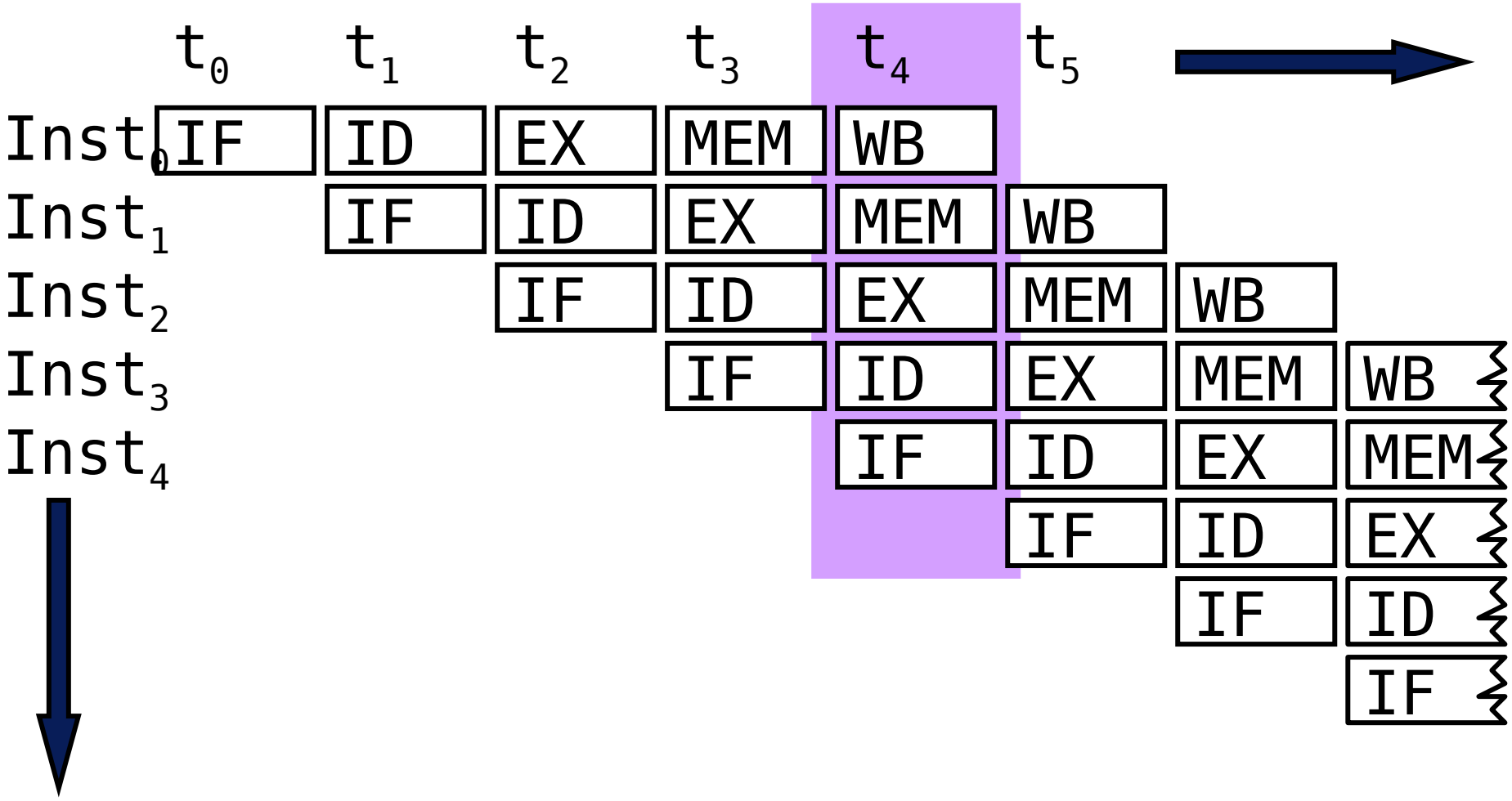
Pipelined Control



Pipeline Diagram



Pipeline Operations



Pipelining and MIPS ISA Design

All instructions are 32-bits

- Easier to fetch and decode in one cycle

Few and regular instruction formats

- Can decode and read registers in one step

Load/store addressing

- Can calculate address in 3rd stage, access memory in 4th stage

Pipeline Stalls

A condition when the pipeline stops moving due to situations that prevent starting the next instruction in the next cycle

Resource contention

A required resource is busy

Data dependencies

Need to wait for previous instruction to complete its data read/write

Control dependencies

Deciding on control action depends on previous instruction

Resource Contention

When instructions in two pipeline stages need the same resource, conflict for use of the resource

In MIPS pipeline with a single memory

- Load/store requires data access
- Instruction fetch would have to stall for that cycle
  - Would cause a pipeline “bubble”

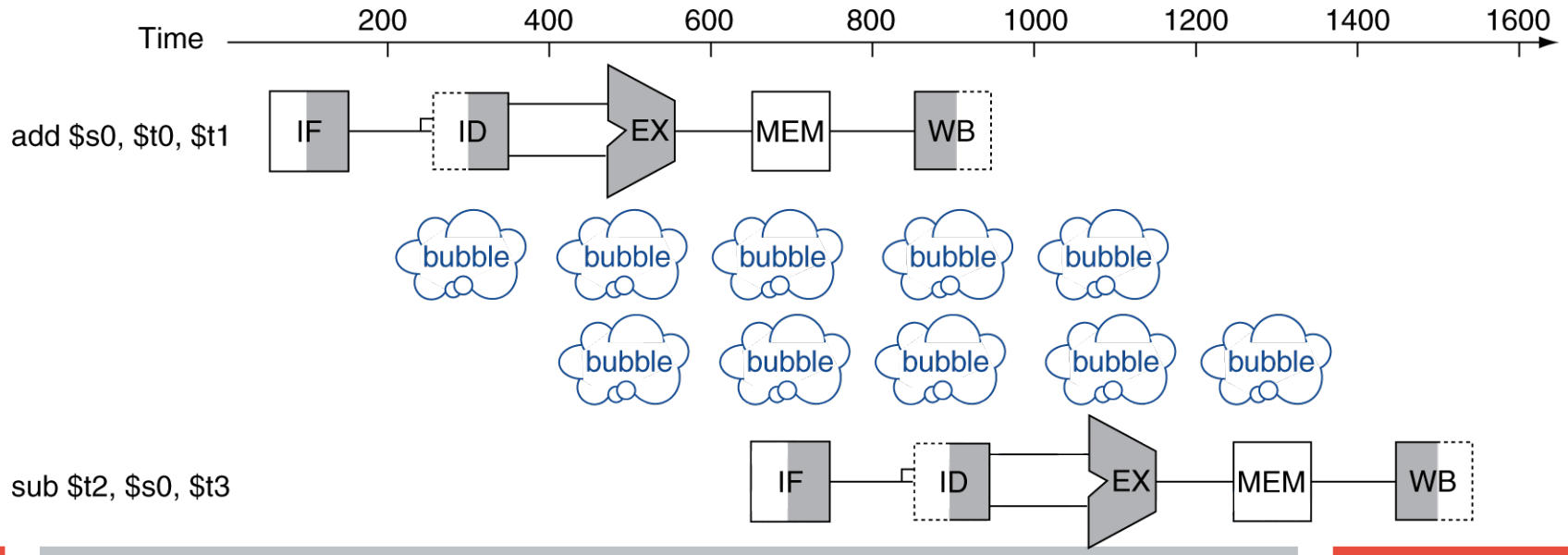
Hence, pipelined datapaths require separate instruction/data memories

- Or separate instruction/data caches

Data Dependencies

An instruction depends on completion of data access by a previous instruction

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
```

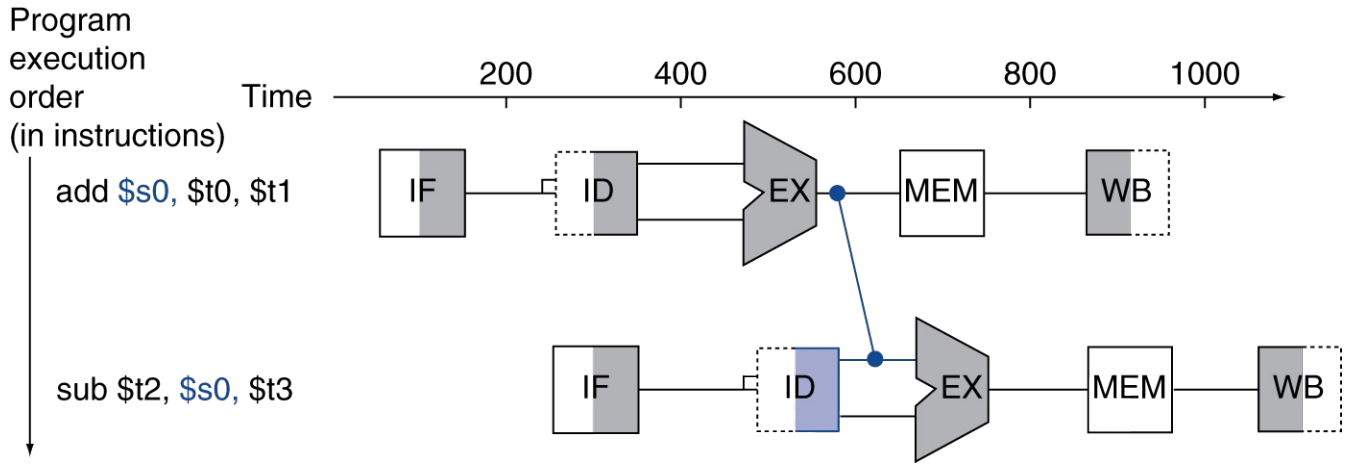


Forwarding

Use result when it is computed

Don't wait for it to be stored in a register

Requires extra connections in the datapath

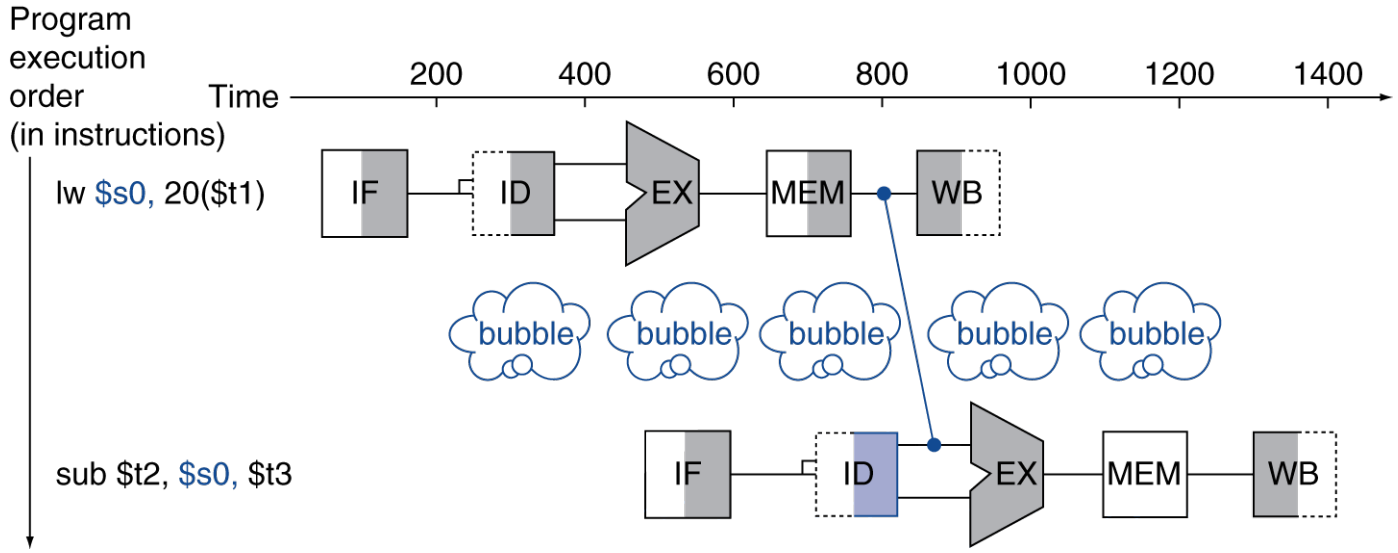


Load-Use Data Dependency

Can’t always avoid stalls by forwarding

If value not computed when needed

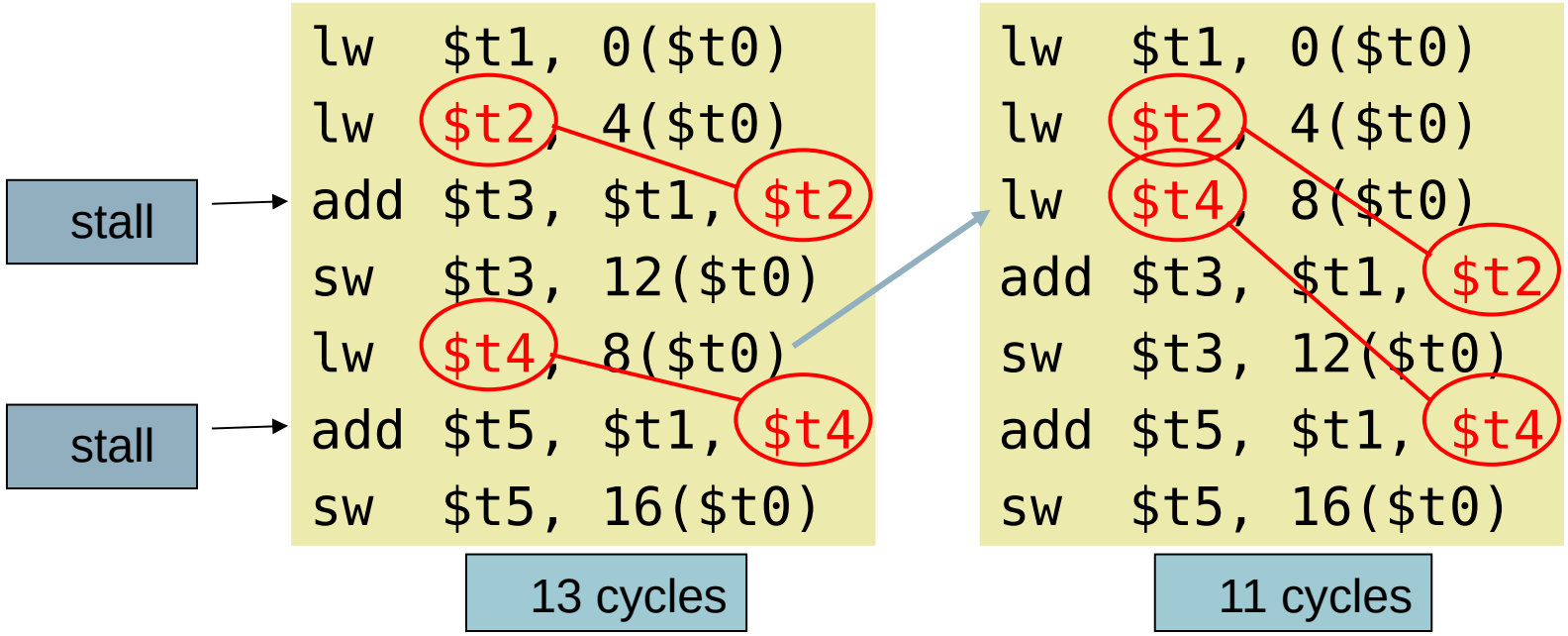
Can’t forward backward in time!



Code Scheduling to Avoid Stalls

Reorder code to avoid use of load result in the next instruction

C code for A = B + E; C = B + F;



Control Dependencies

What should the fetch PC be in the next cycle?

The address of the next instruction

If the fetched instruction is a non-control-flow instruction:

Next Fetch PC is the address of the next-sequential instruction

If the instruction that is fetched is a control-flow instruction:

How do we determine the next Fetch PC?

Control Dependencies

Branch determines flow of control

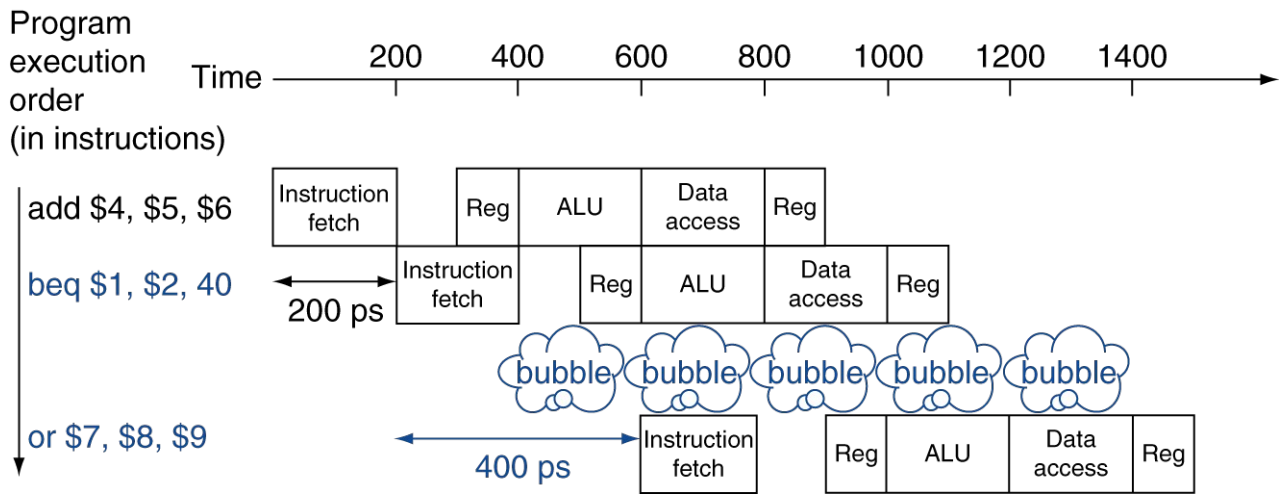
- Fetching next instruction depends on branch outcome
- Pipeline still working on ID stage of branch

In MIPS pipeline

- Need to compare registers and compute target early in the pipeline
- Add hardware to do it in ID stage

Stall on Branch

Wait until branch outcome determined before fetching next instruction



Branch Prediction

Longer pipelines can’t readily determine branch outcome early

- Stall penalty becomes unacceptable

Predict outcome of branch

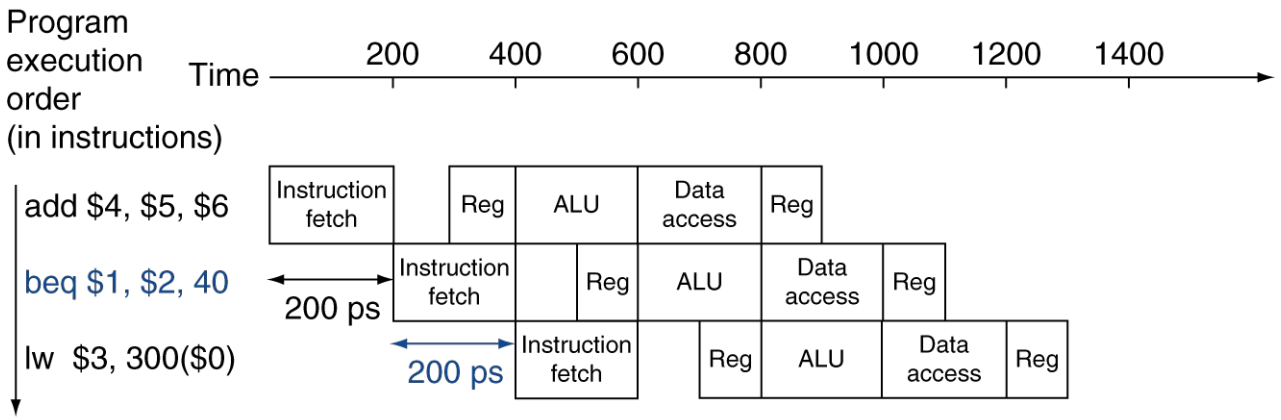
- Only stall if prediction is wrong

In MIPS pipeline

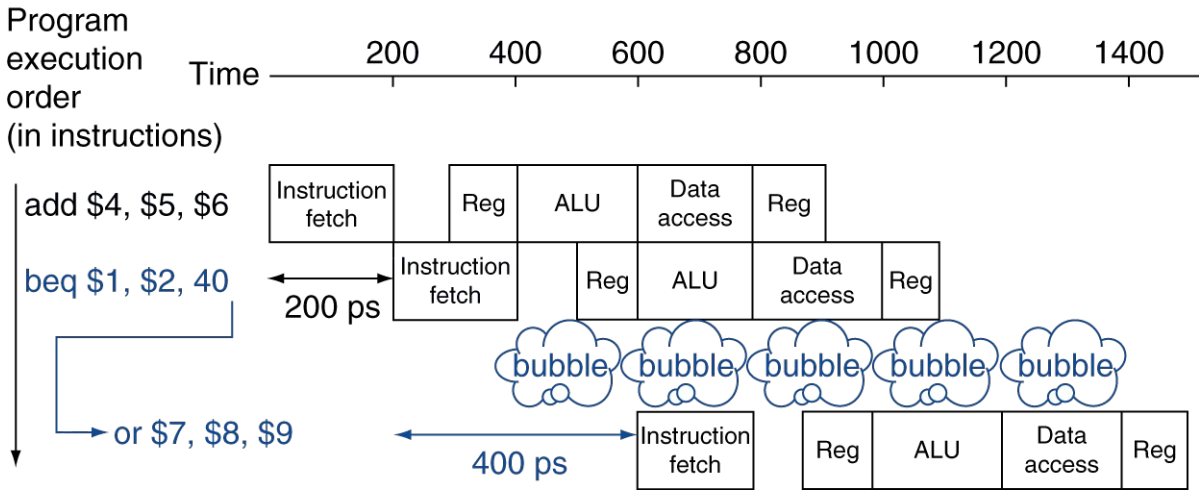
- Can predict branches not taken
- Fetch instruction after branch, with no delay

MIPS with Predict Not Taken

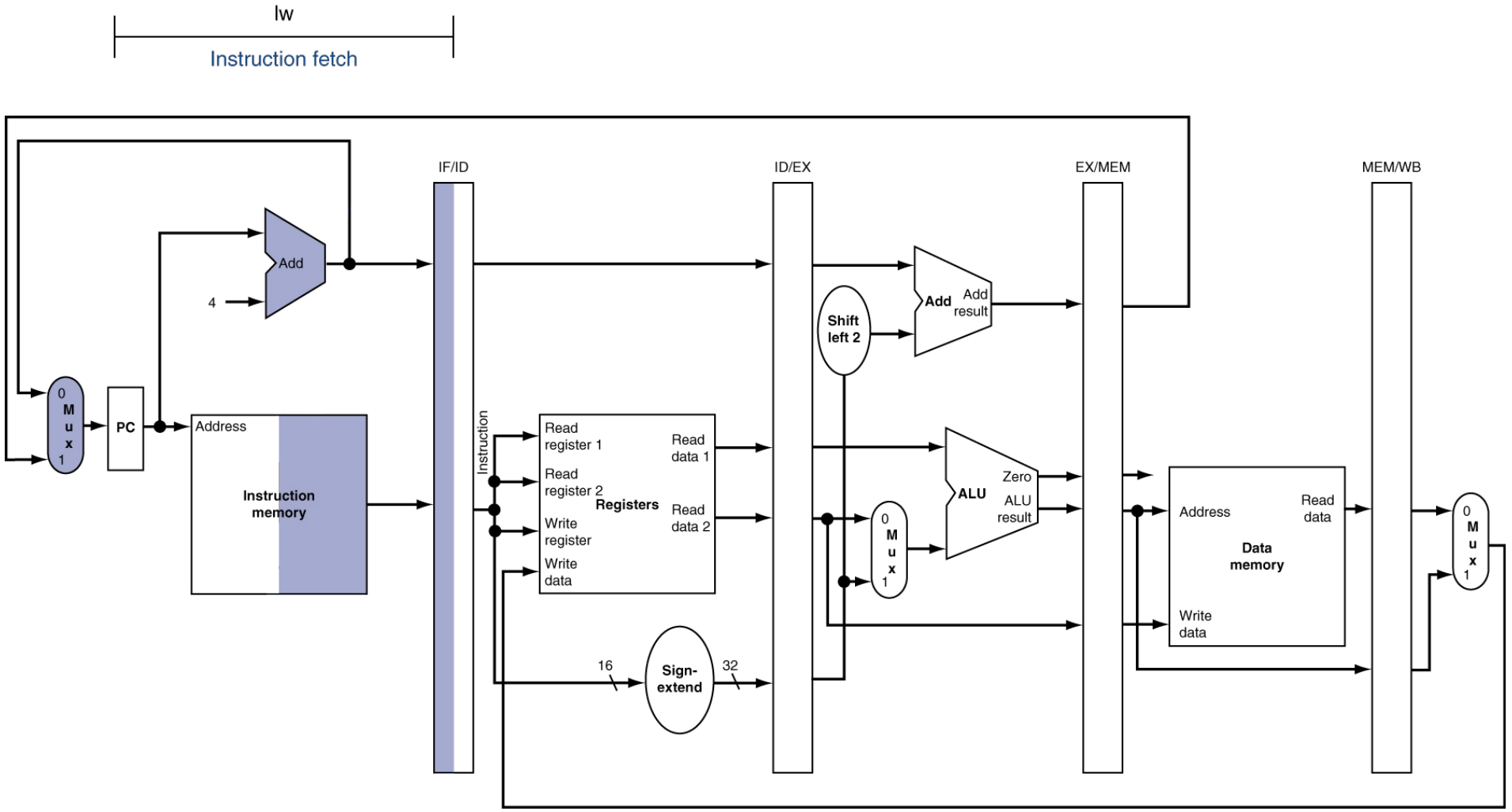
Prediction correct



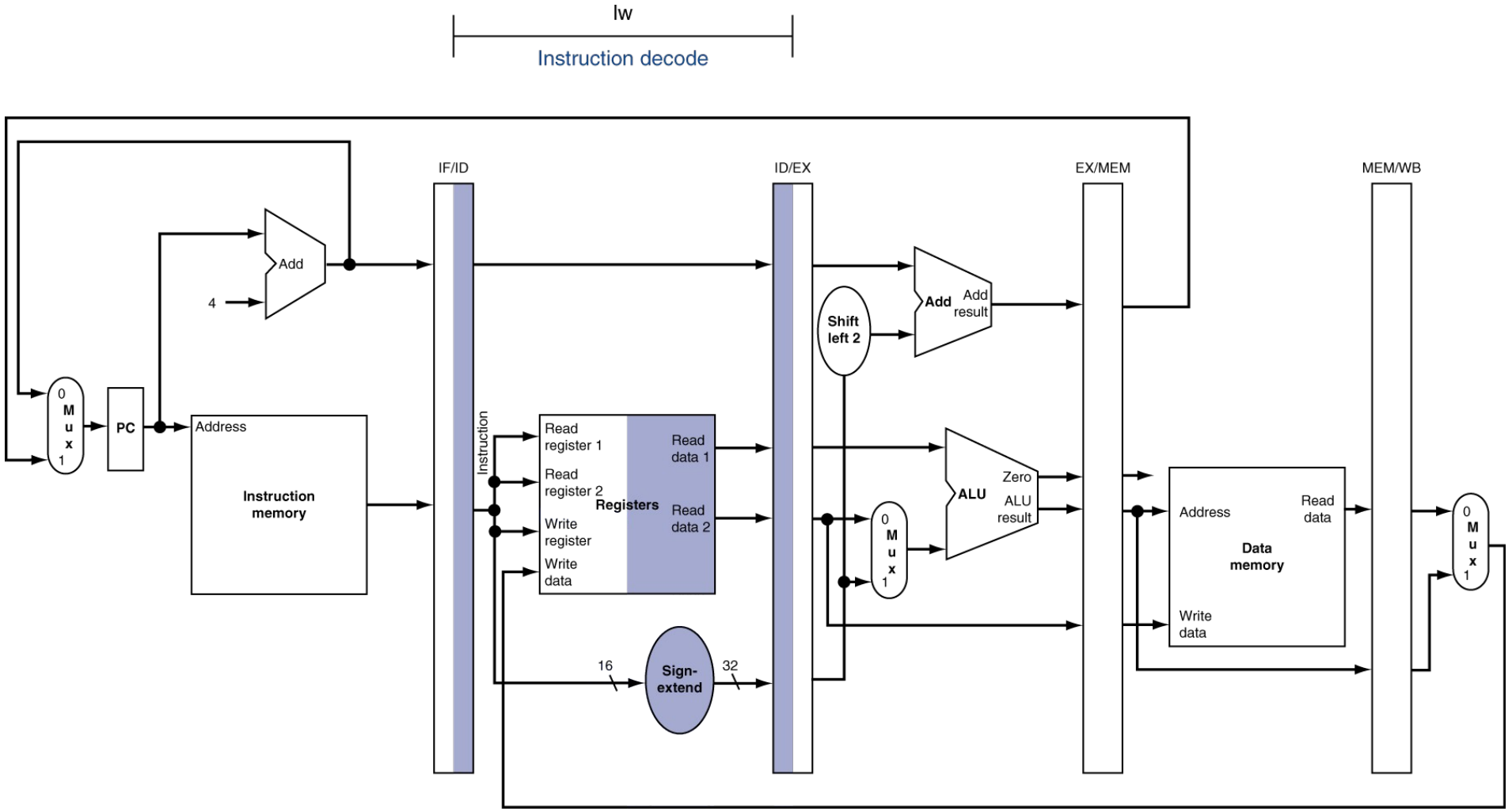
Prediction incorrect



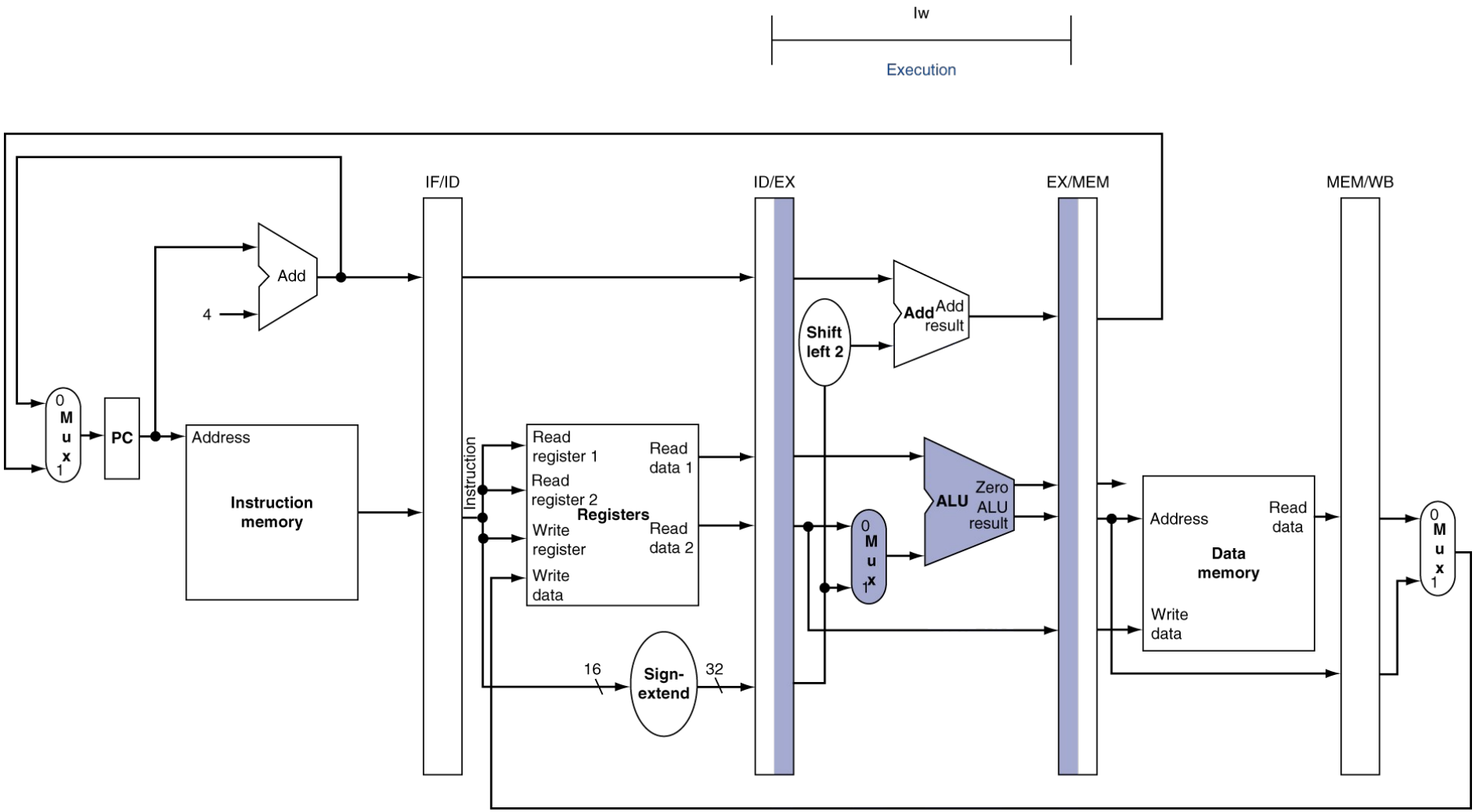
IF for Load, Store, …



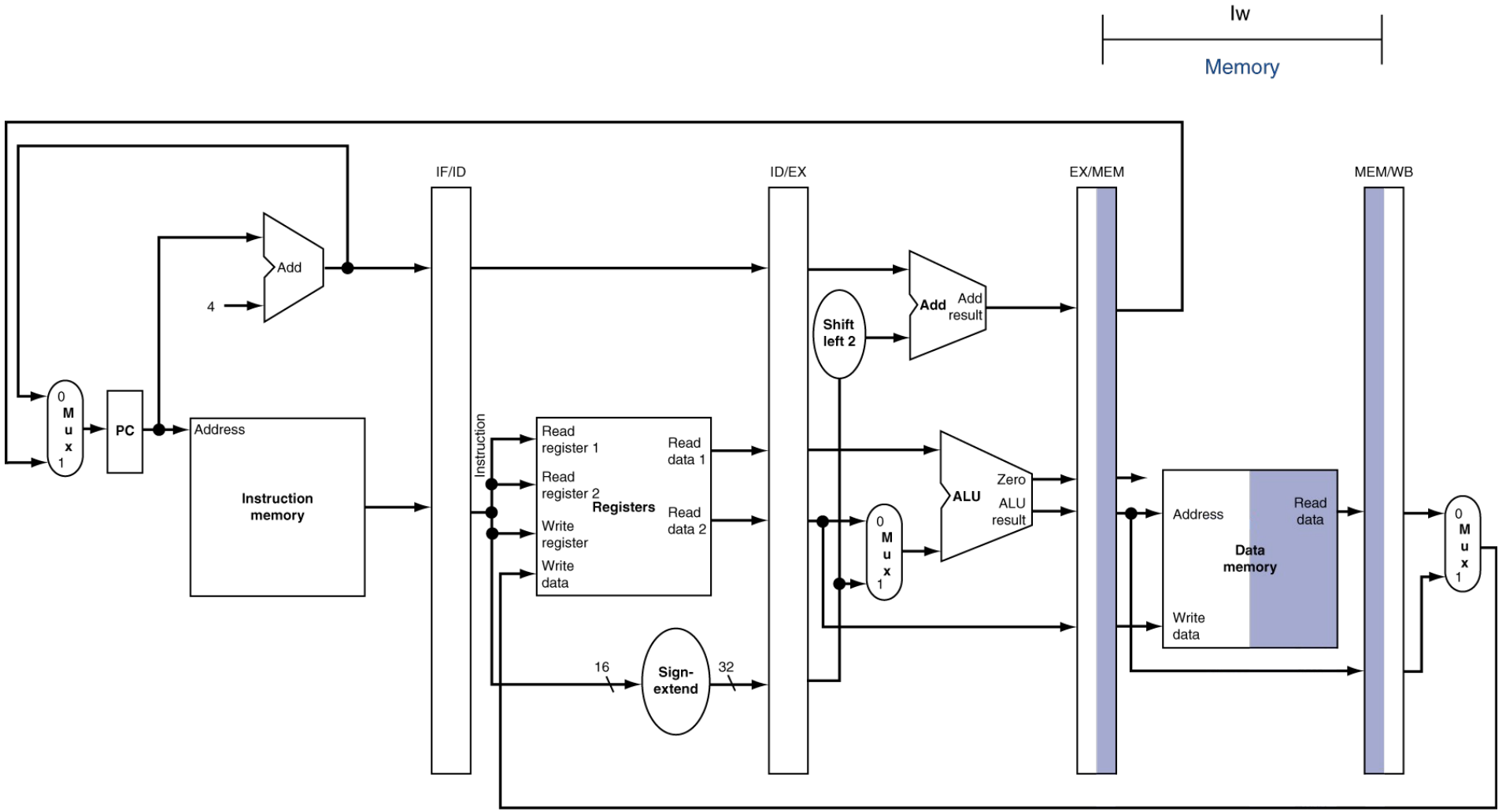
ID for Load, Store, …



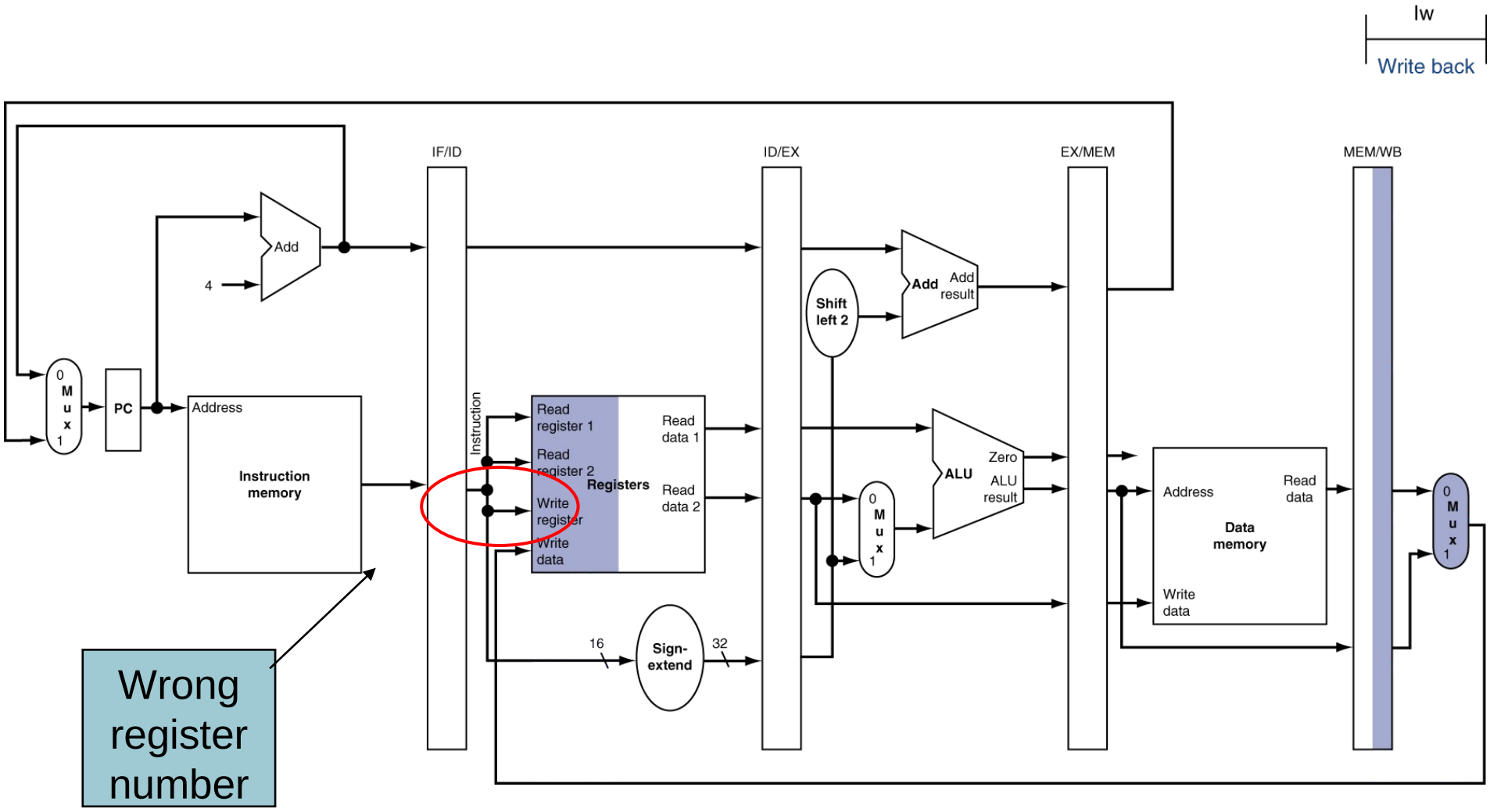
EX for Load


lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32

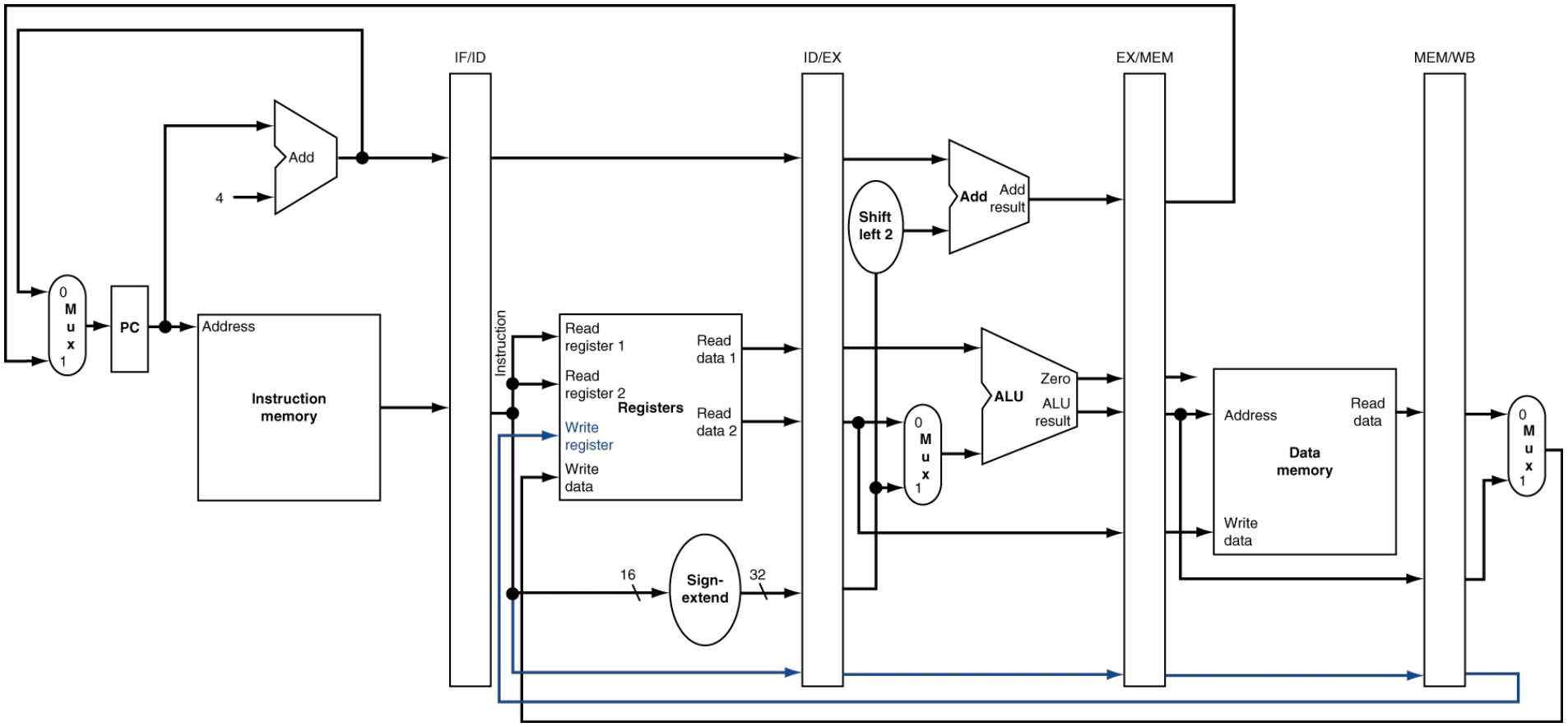
MEM for Load


lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32

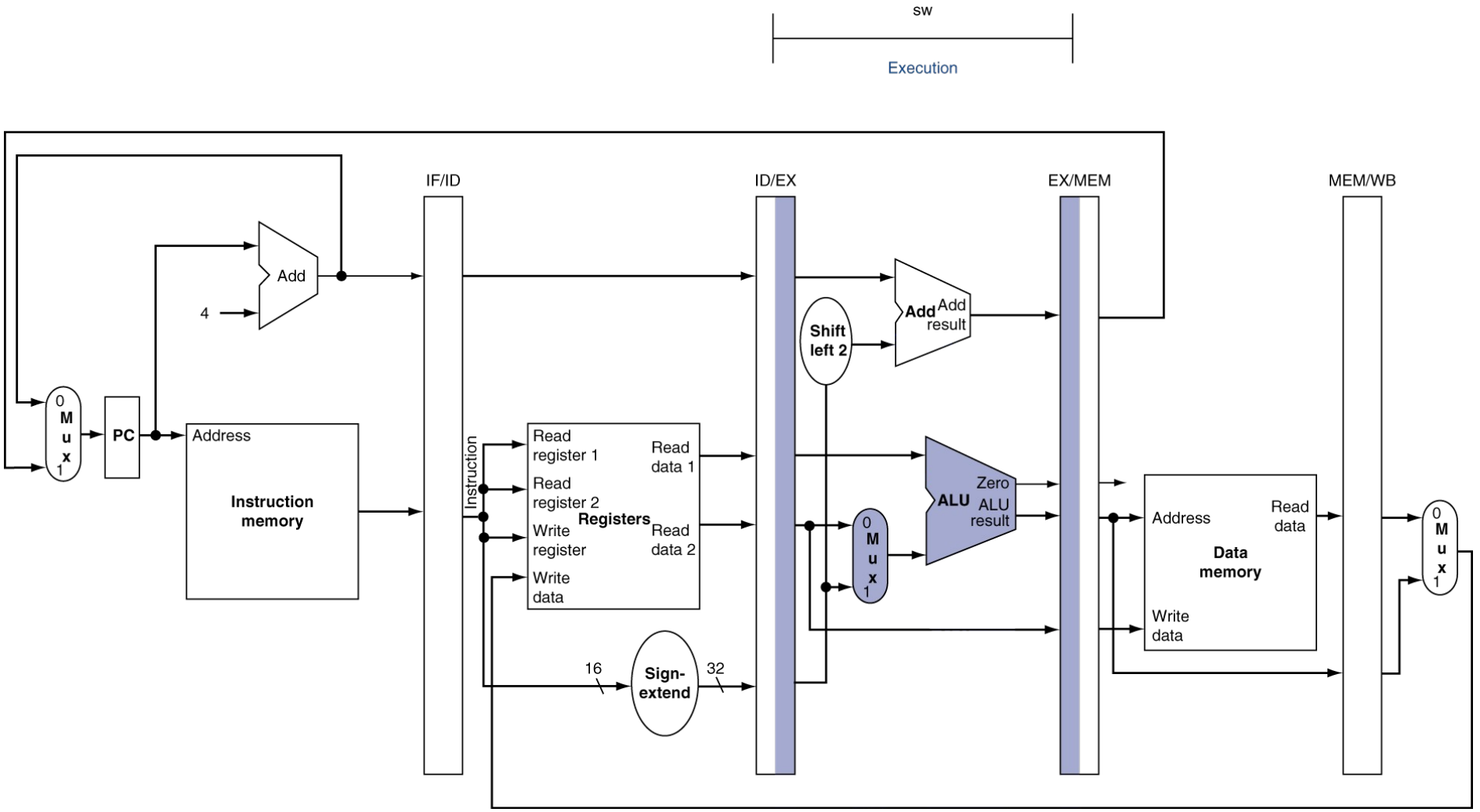
WB for Load


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32
Wrong register number

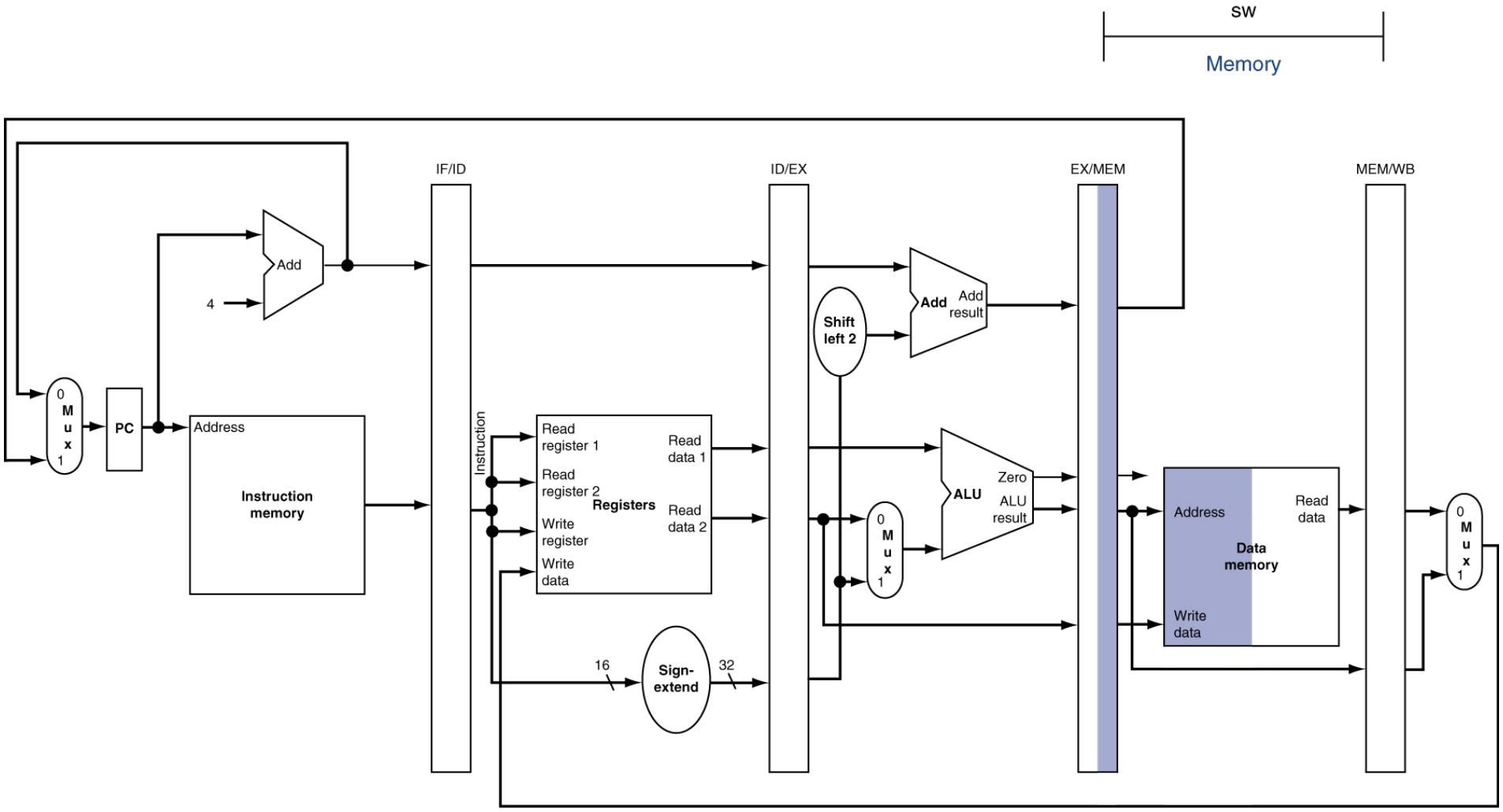
Corrected Datapath for Load



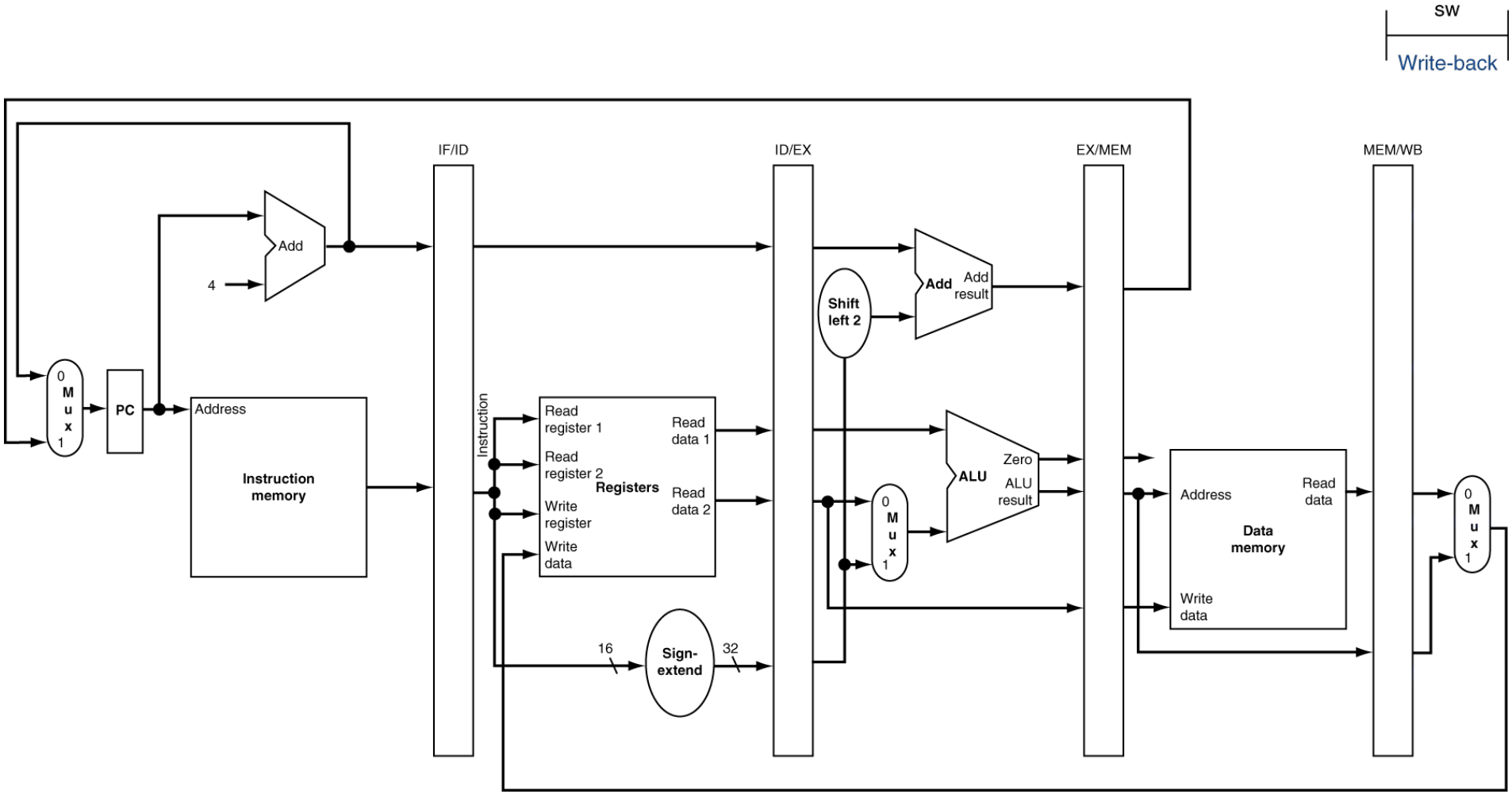
EX for Store


sw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

MEM for Store



WB for Store


SW
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

References

Chapter 4.5 - 4.8

(Computer Organization and Design: The Hardware/Software Interface by Hennessy/Patterson, 5th edition)