

Jin-Soo Kim  
([jinsoo.kim@snu.ac.kr](mailto:jinsoo.kim@snu.ac.kr))

Systems Software &  
Architecture Lab.  
Seoul National University

Fall 2020

# Pipelining

Chap. 4.5



# Sequential Processing



# Parallel Processing (Multi-core)

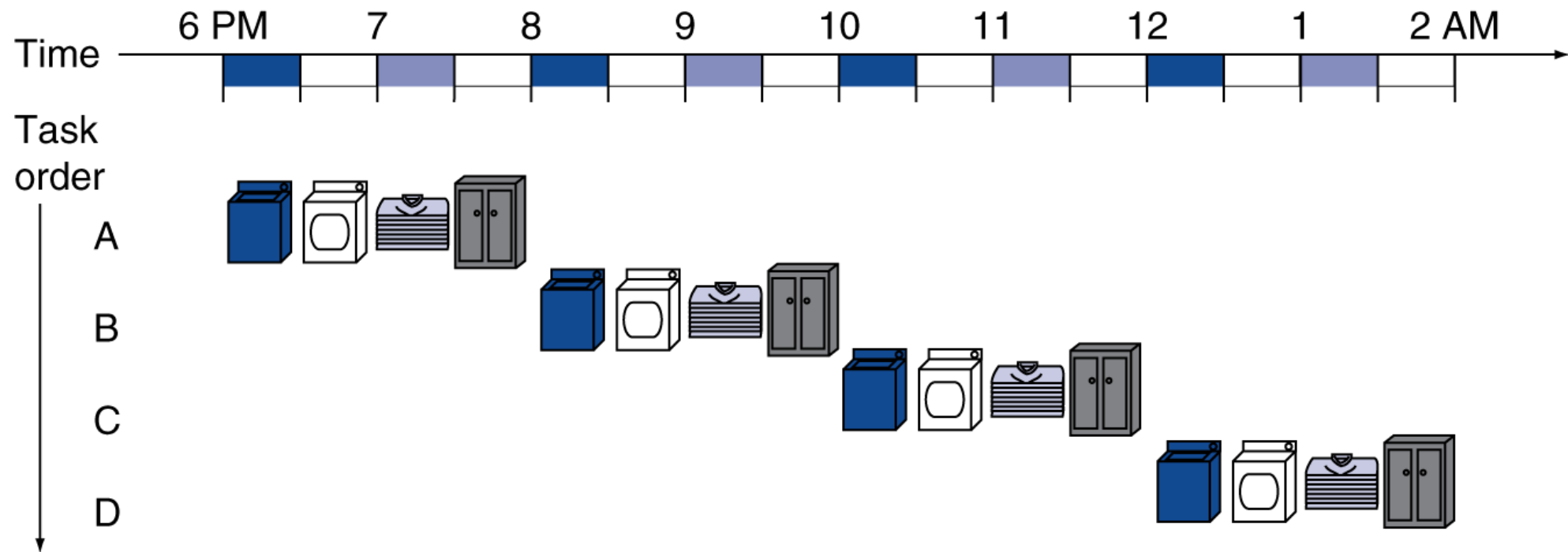


# Pipelining



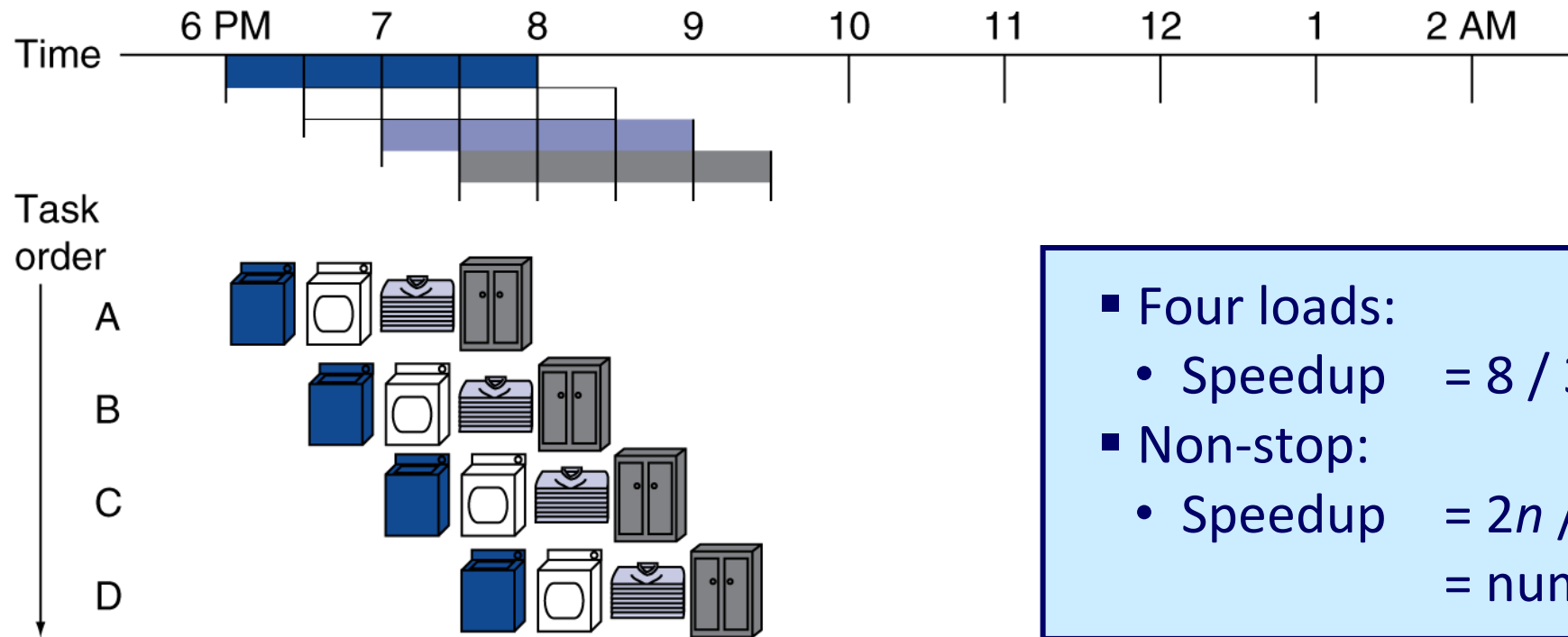
# Laundry Example

- Sequential processing: Wash-Dry-Fold-Store



# Pipelined Laundry Example

- Overlapping execution
- Parallelism improves performance



- Four loads:
  - Speedup =  $8 / 3.5 = 2.3$
- Non-stop:
  - Speedup =  $2n / (0.5n + 1.5) \approx 4$   
= number of stages

# A RISC-V Pipeline

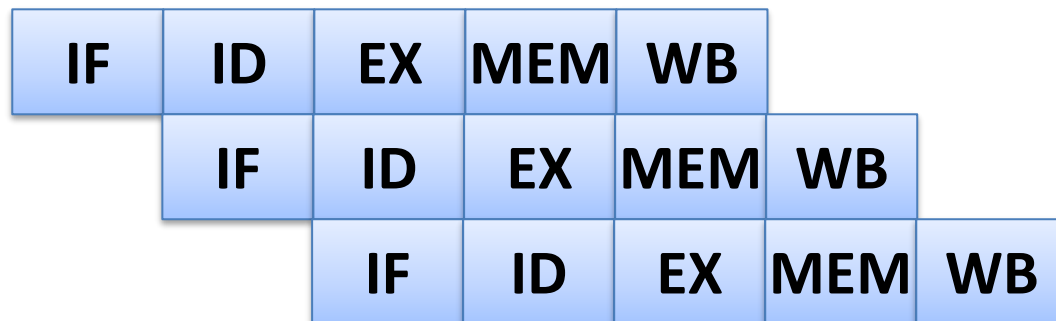
- Five stages, one step per stage
- **IF:** Instruction fetch from memory
- **ID:** Instruction decode & register read
- **EX:** Execute operation or calculate address
- **MEM:** Access memory operand
- **WB:** Write result back to register

# Pipelined Instruction Execution

- Sequential execution



- Pipelined execution



```
add x10, x11, x12
sub x13, x14, x15
and x5, x6, x7
```



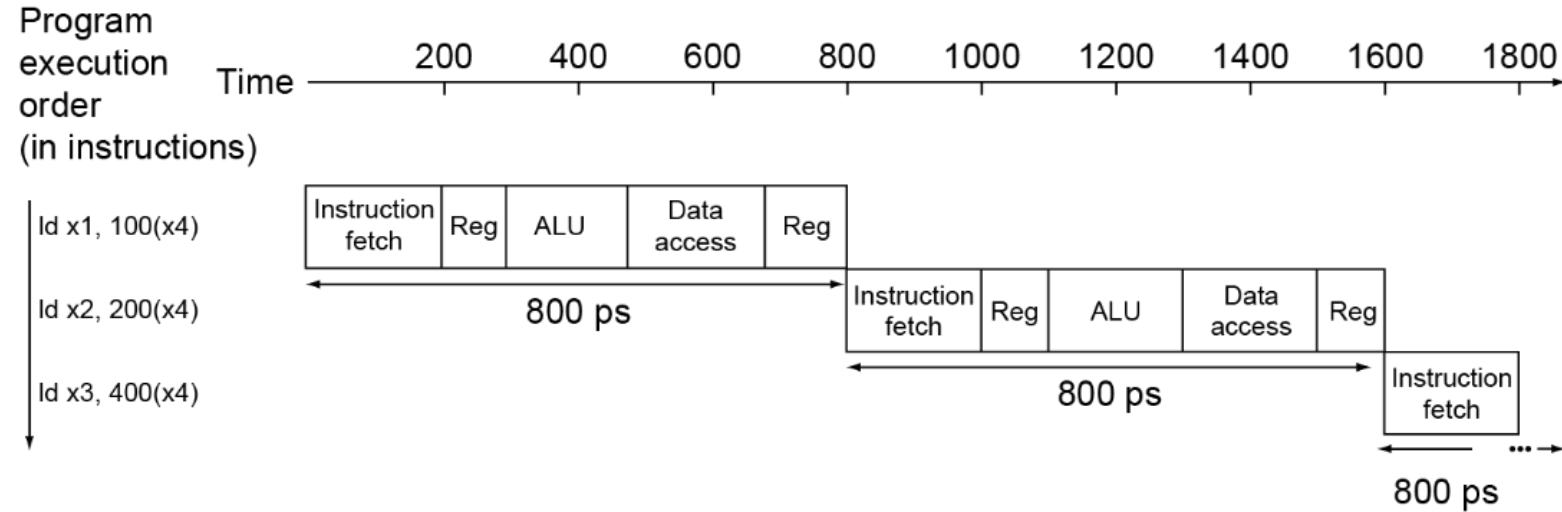
# Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

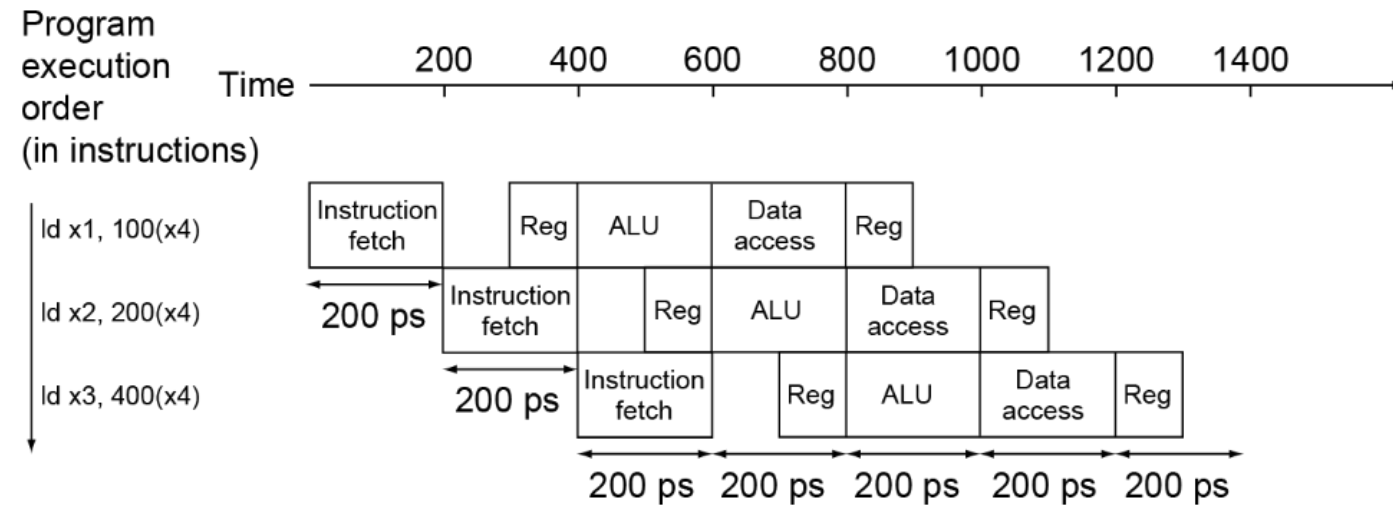
Inst.	Inst. fetch	Register read	ALU op.	Memory access	Register write	Total time
ld	200ps	100ps	200ps	200ps	100ps	800ps
sd	200ps	100ps	200ps	200ps		700ps
R-type	200ps	100ps	200ps		100ps	600ps
beq	200ps	100ps	200ps			500ps

# Pipeline Performance (cont'd)

**Single-cycle**  
( $T_c = 800\text{ps}$ )



**Pipelined**  
( $T_c = 200\text{ps}$ )



# Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time

$$\textit{Time between instructions}_{\textit{pipelined}} = \frac{\textit{Time between instructions}_{\textit{nonpipelined}}}{\textit{Number of stages}}$$

- If not balanced, speedup is less
- Speedup due to increased throughput
- Latency (time for each instruction) does not decrease

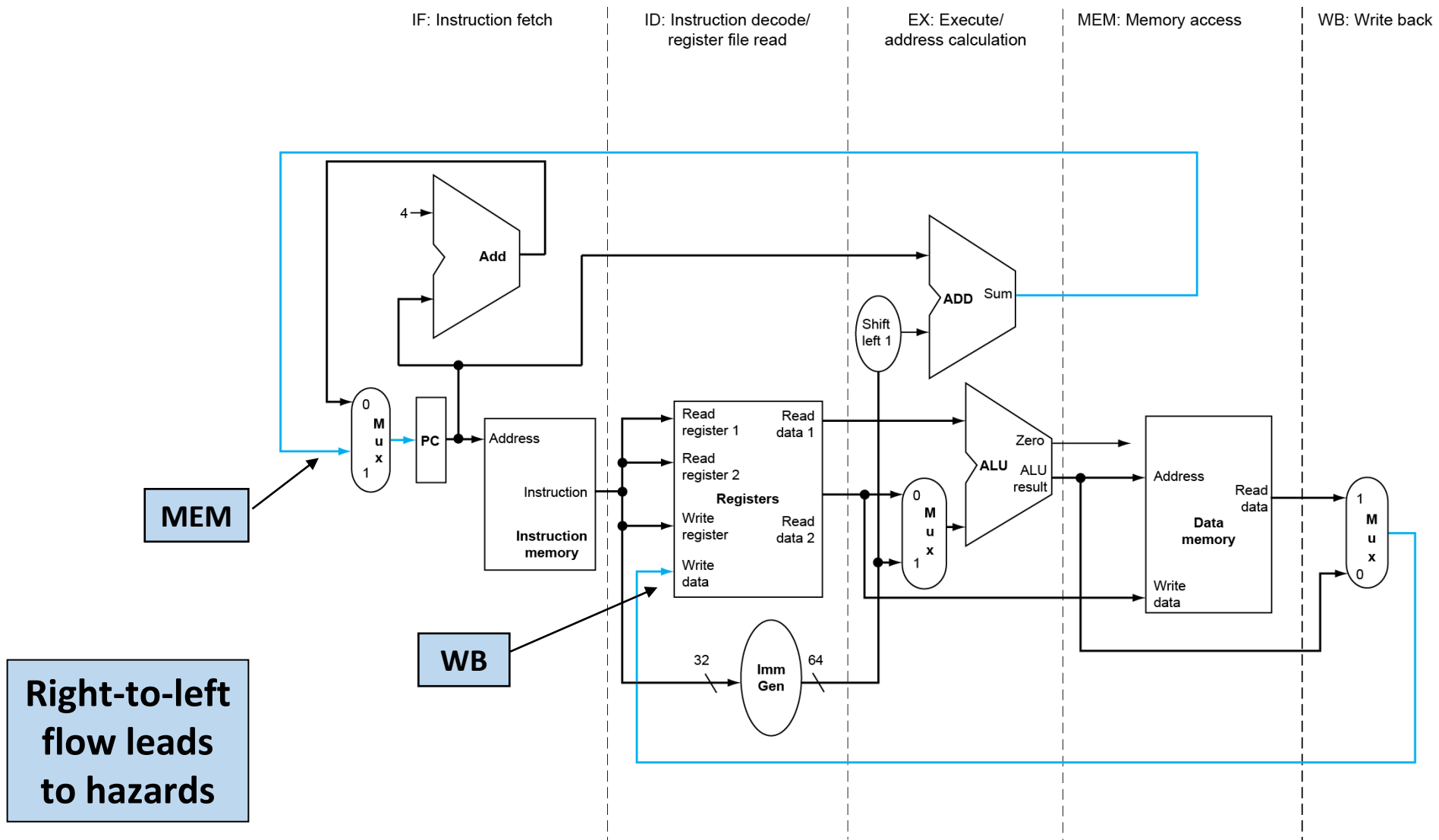
# Pipelining and ISA Design

- RISC-V ISA designed for pipelining
- All instructions are 32-bits
  - Easier to fetch and decode in one cycle
  - (cf.) x86: 1- to 17-byte instructions
- Few and regular instruction formats
  - Source and destination register fields located in the same place
  - Can decode and read registers in one step
- Load/store addressing
  - Can calculate address in 3<sup>rd</sup> EX stage, access memory in 4<sup>th</sup> MEM stage

# Pipelined Datapath and Control

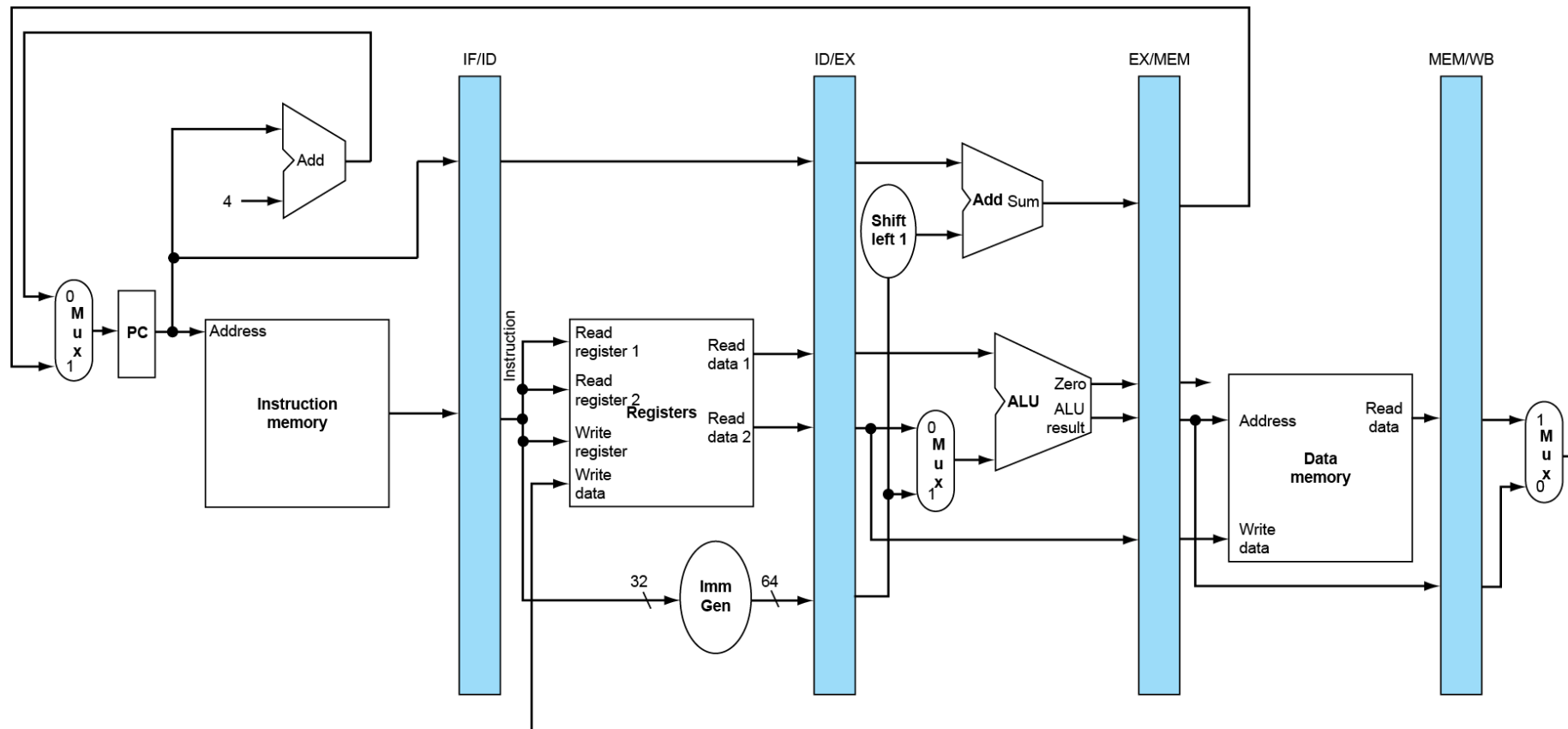
Chap. 4.6

# RISC-V Pipelined Datapath



# Pipeline Registers

- Need registers between stages
  - To hold information produced in previous cycle

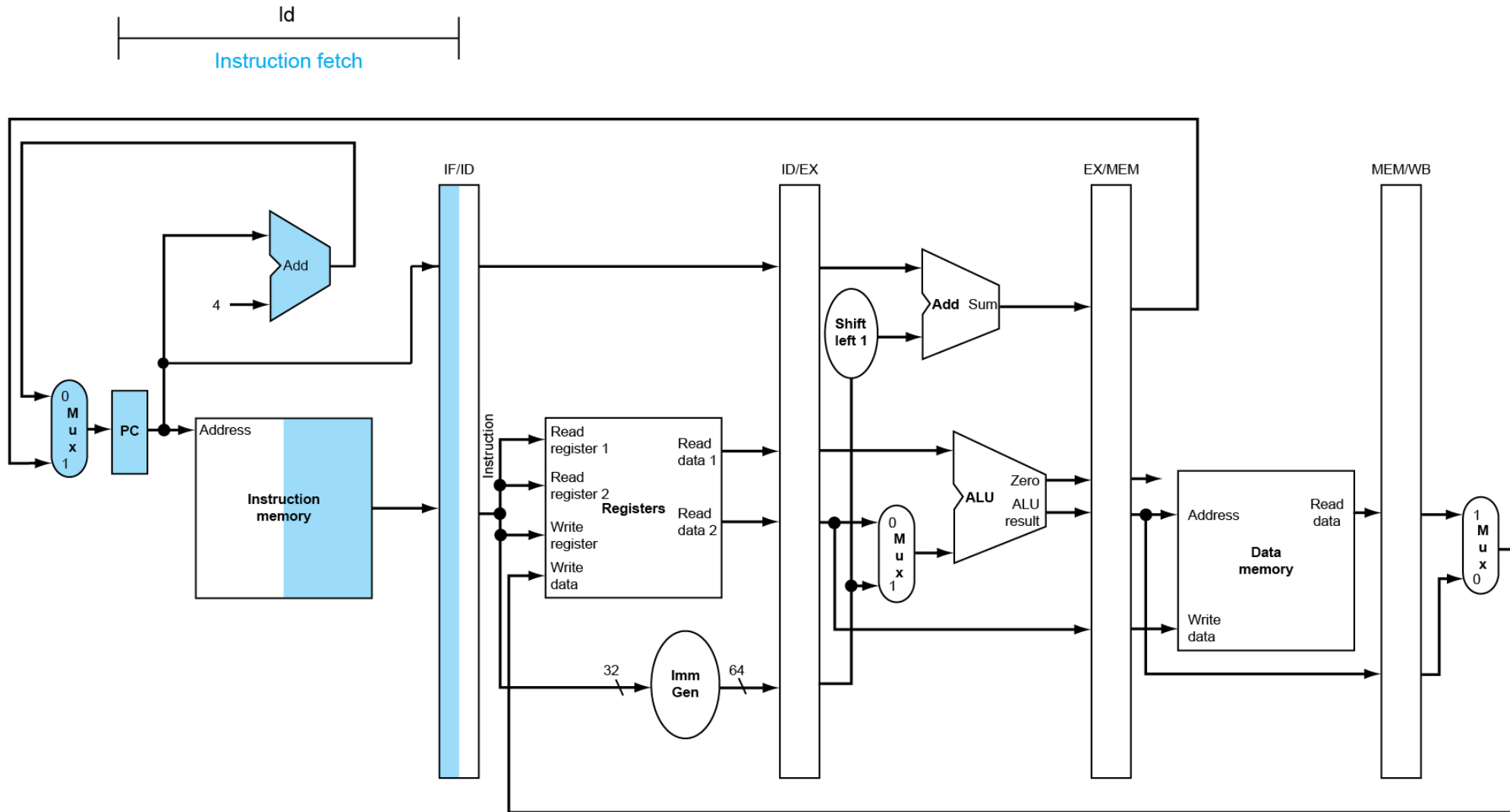


# Pipeline Operation

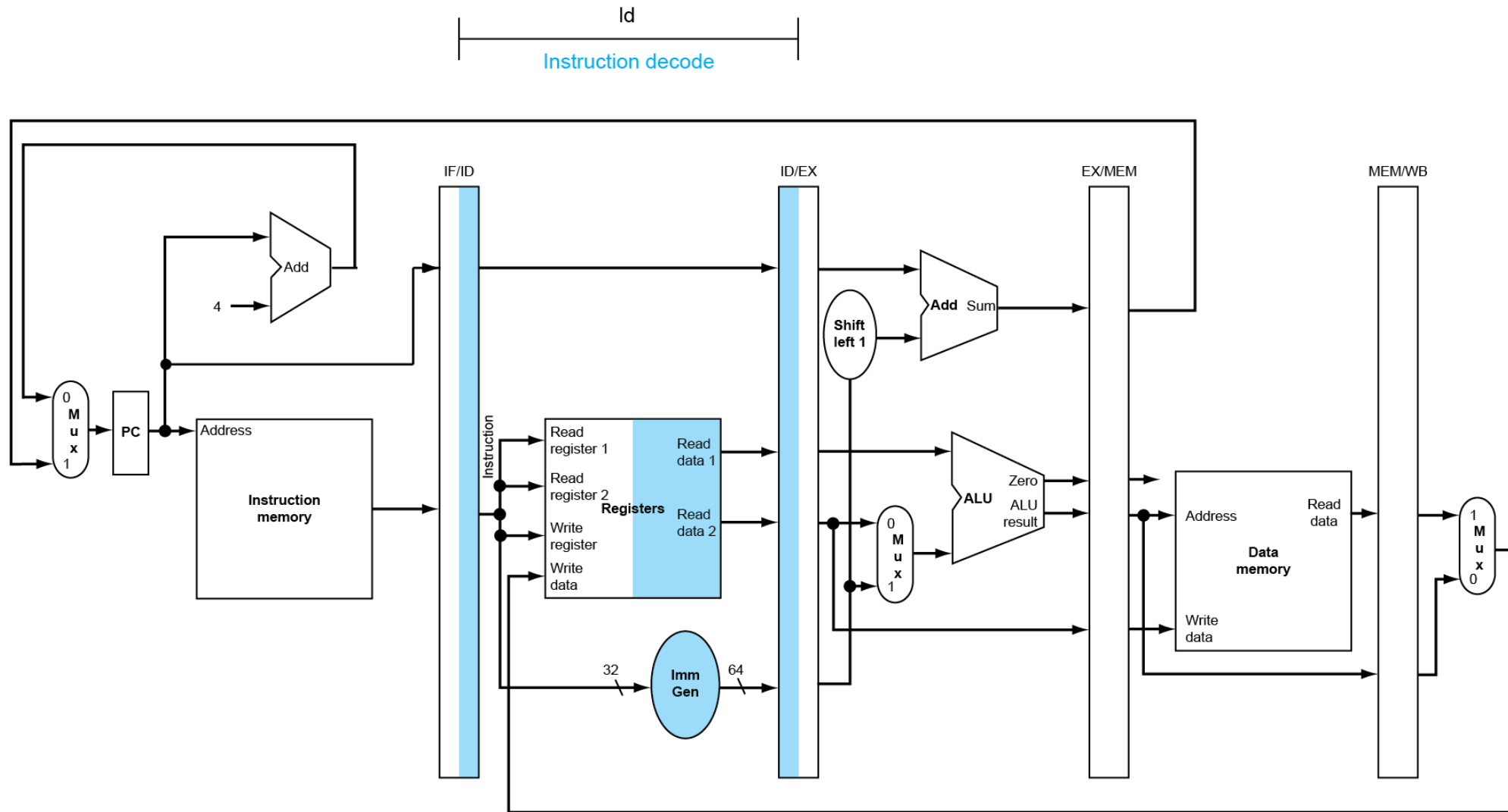
- Cycle-by-cycle flow of instructions through the pipelined datapath
- “Single-clock-cycle” pipeline diagram
  - Shows pipeline usage in a single cycle
  - Highlight resource used
  - (cf.) “multi-clock-cycle” diagram: graph of operation over time
- We’ll look at “single-clock-cycle” diagrams for load & store



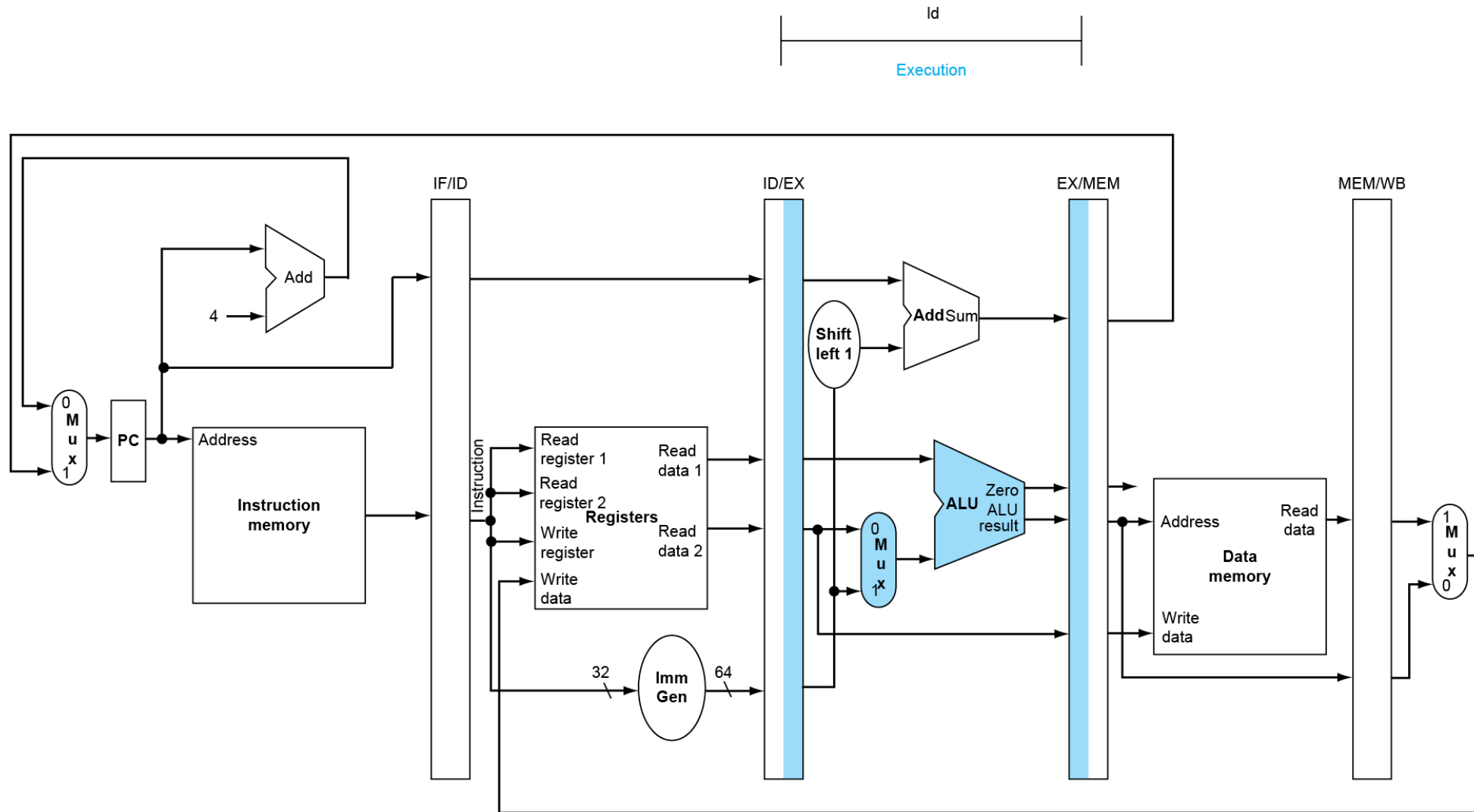
# IF for Load



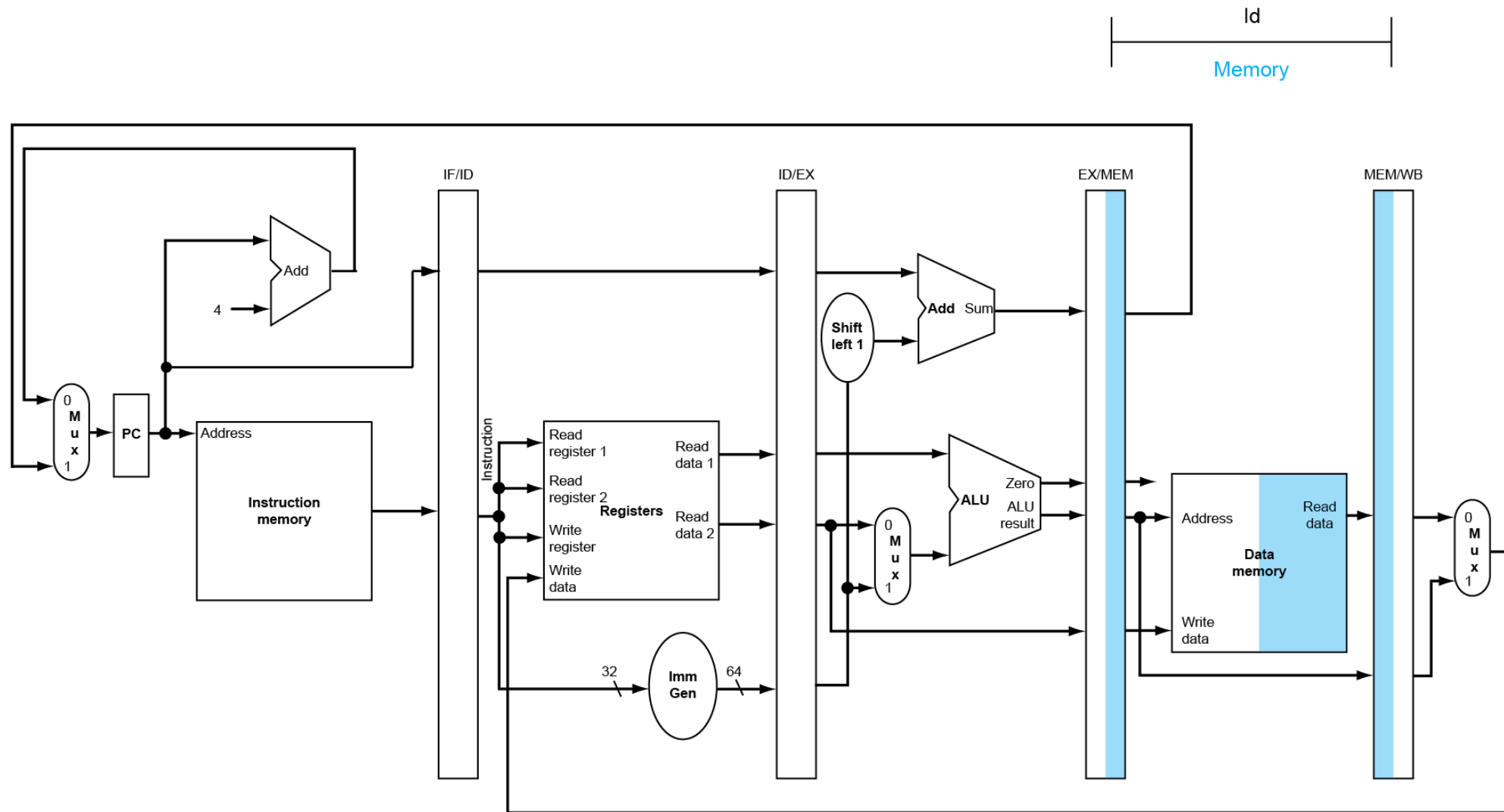
# ID for Load



# EX for Load

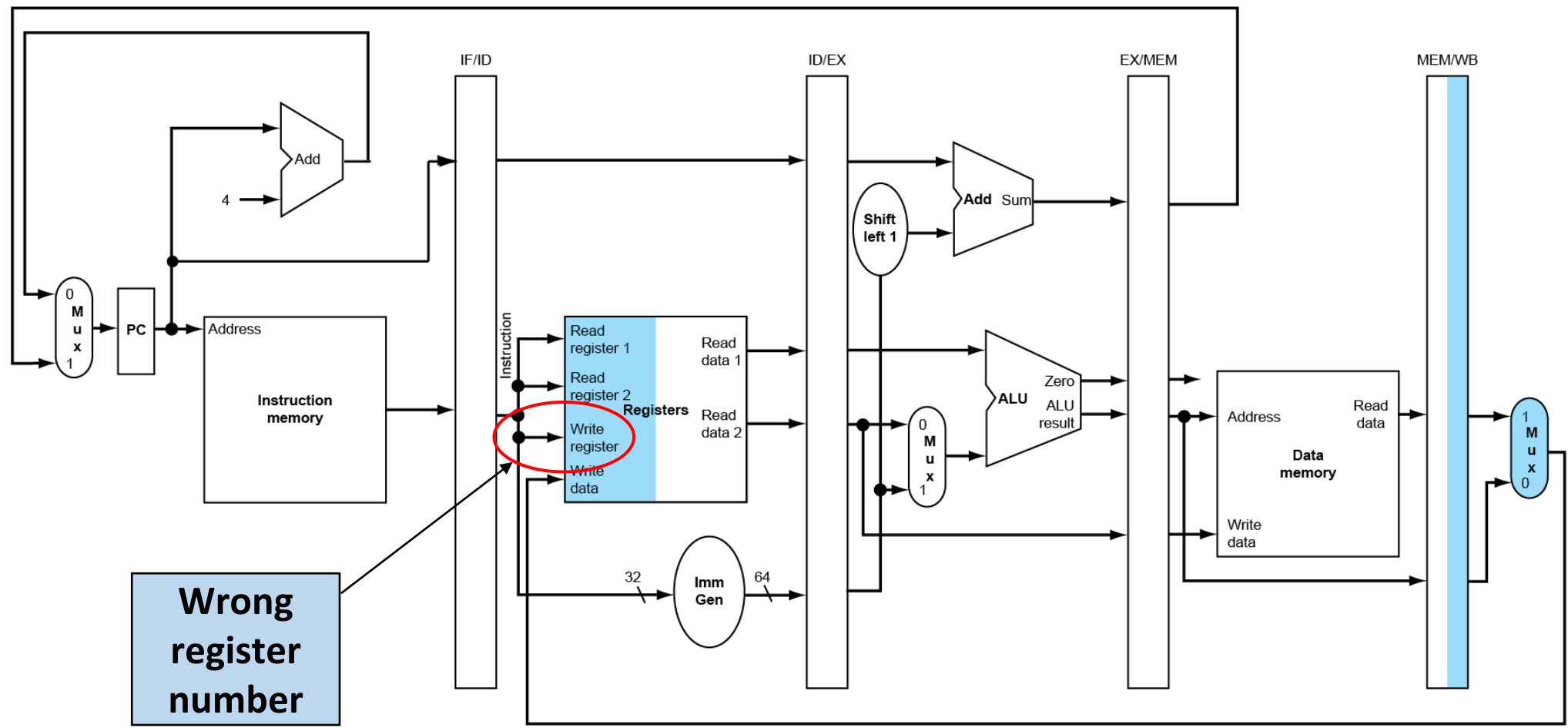


# MEM for Load



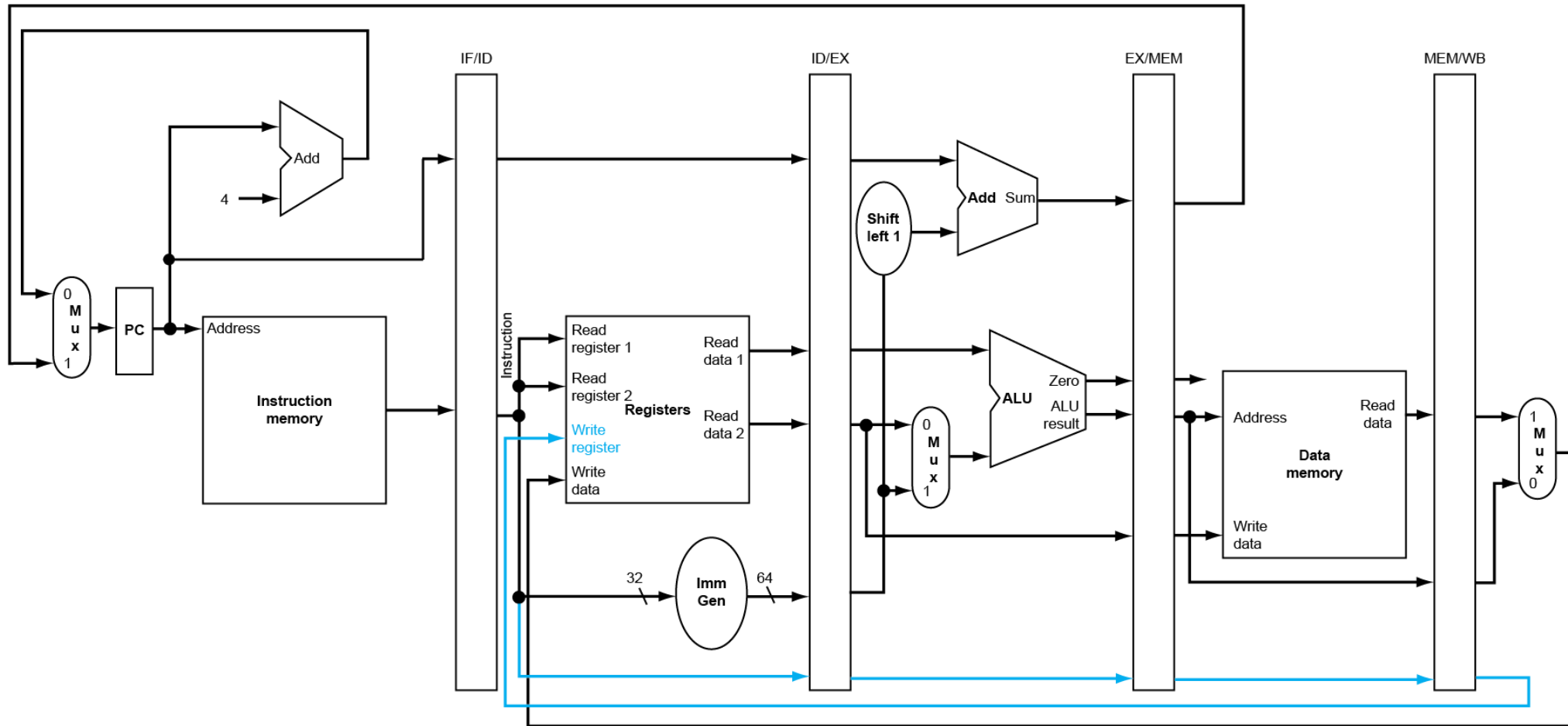
# WB for Load

ld  
Write-back

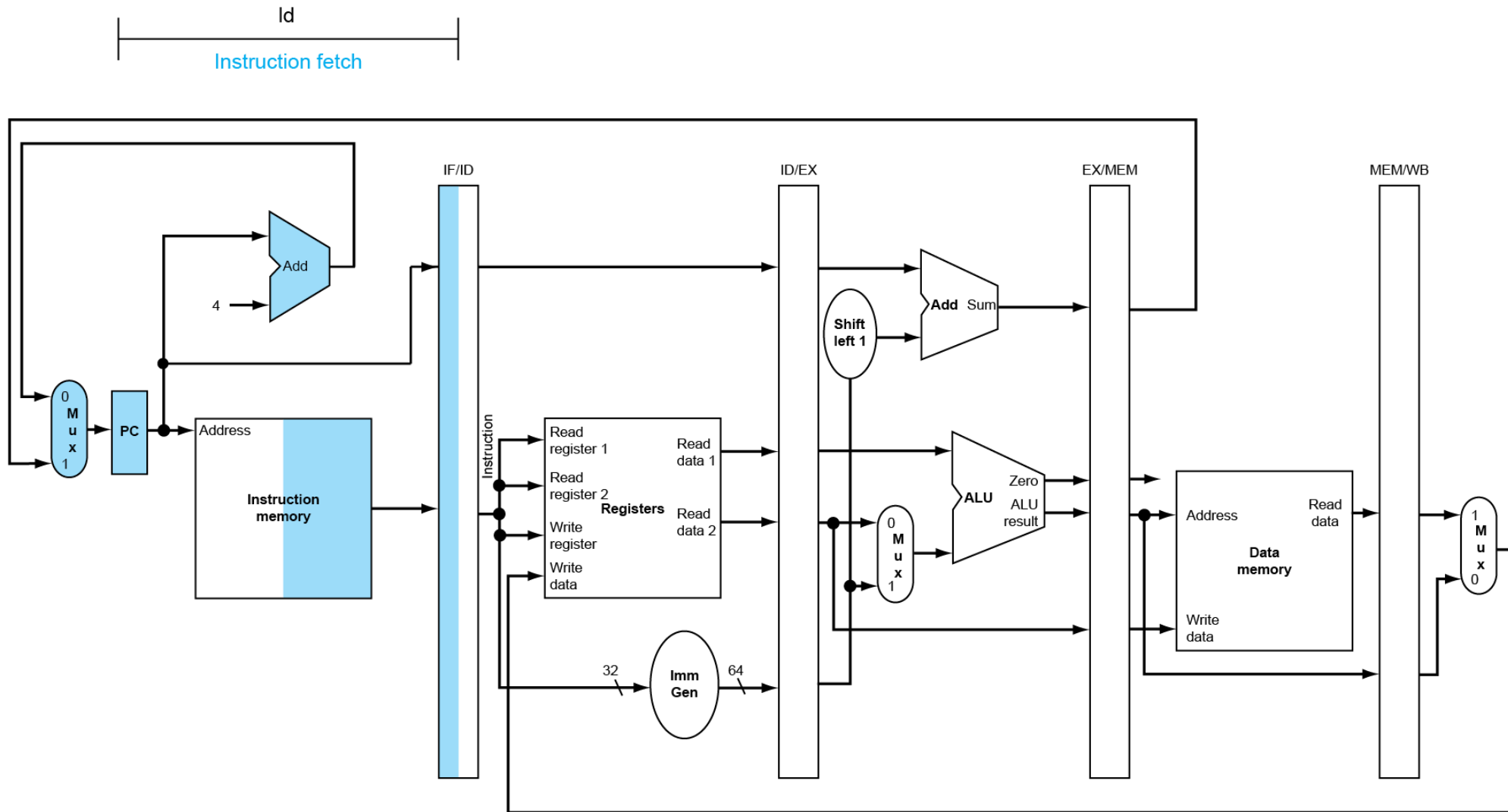


Wrong register number

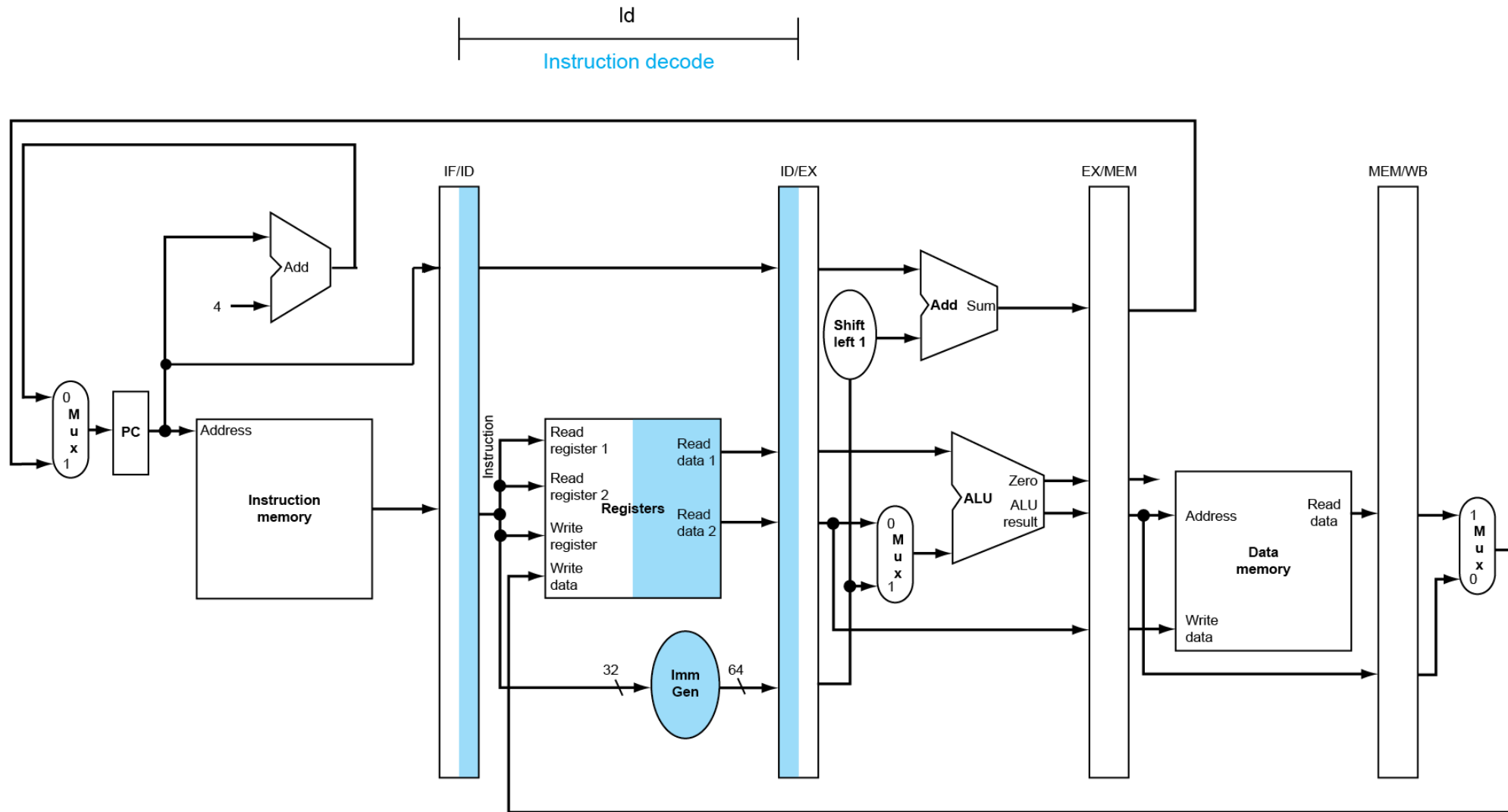
# Corrected Datapath for Load



# IF for Store

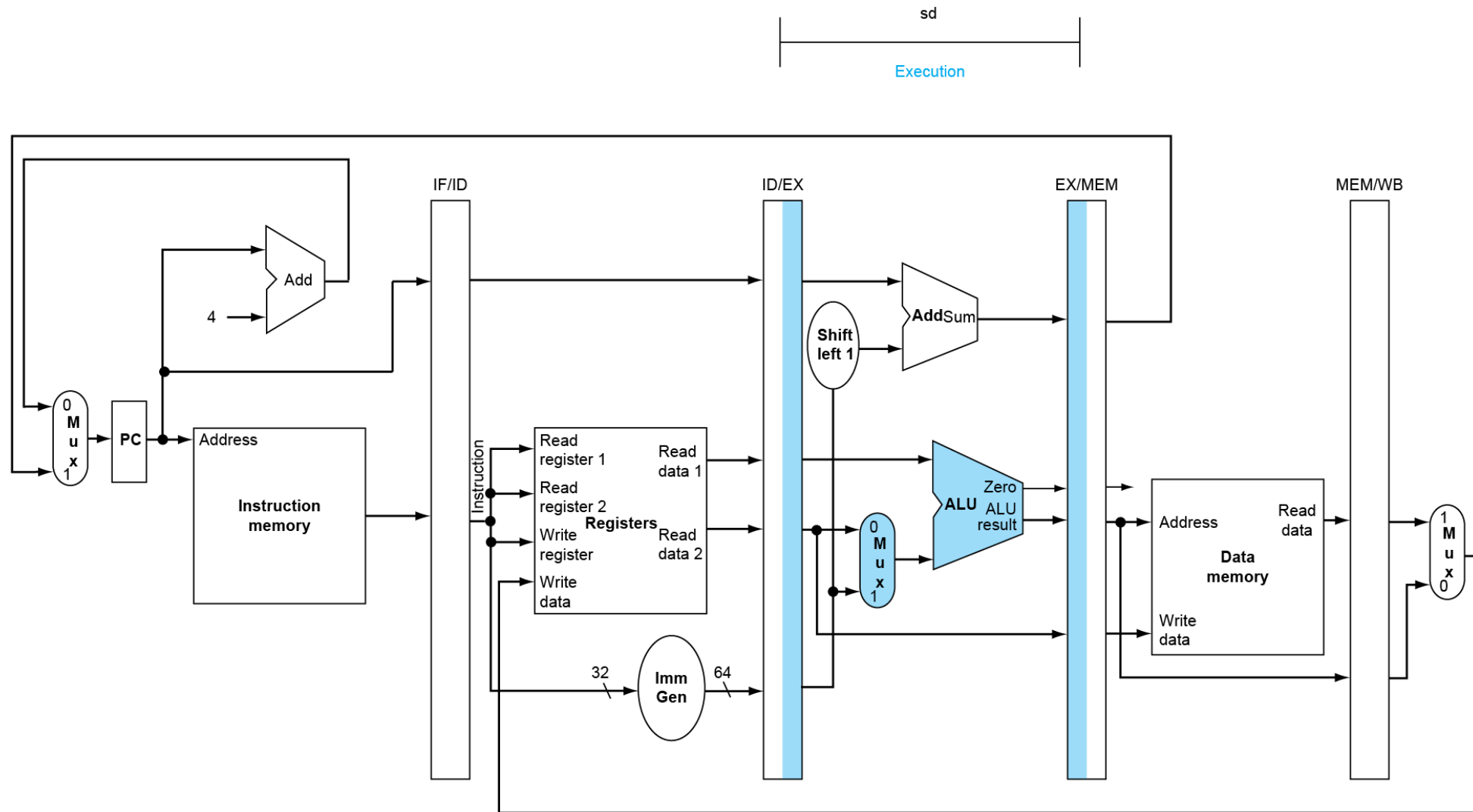


# ID for Store

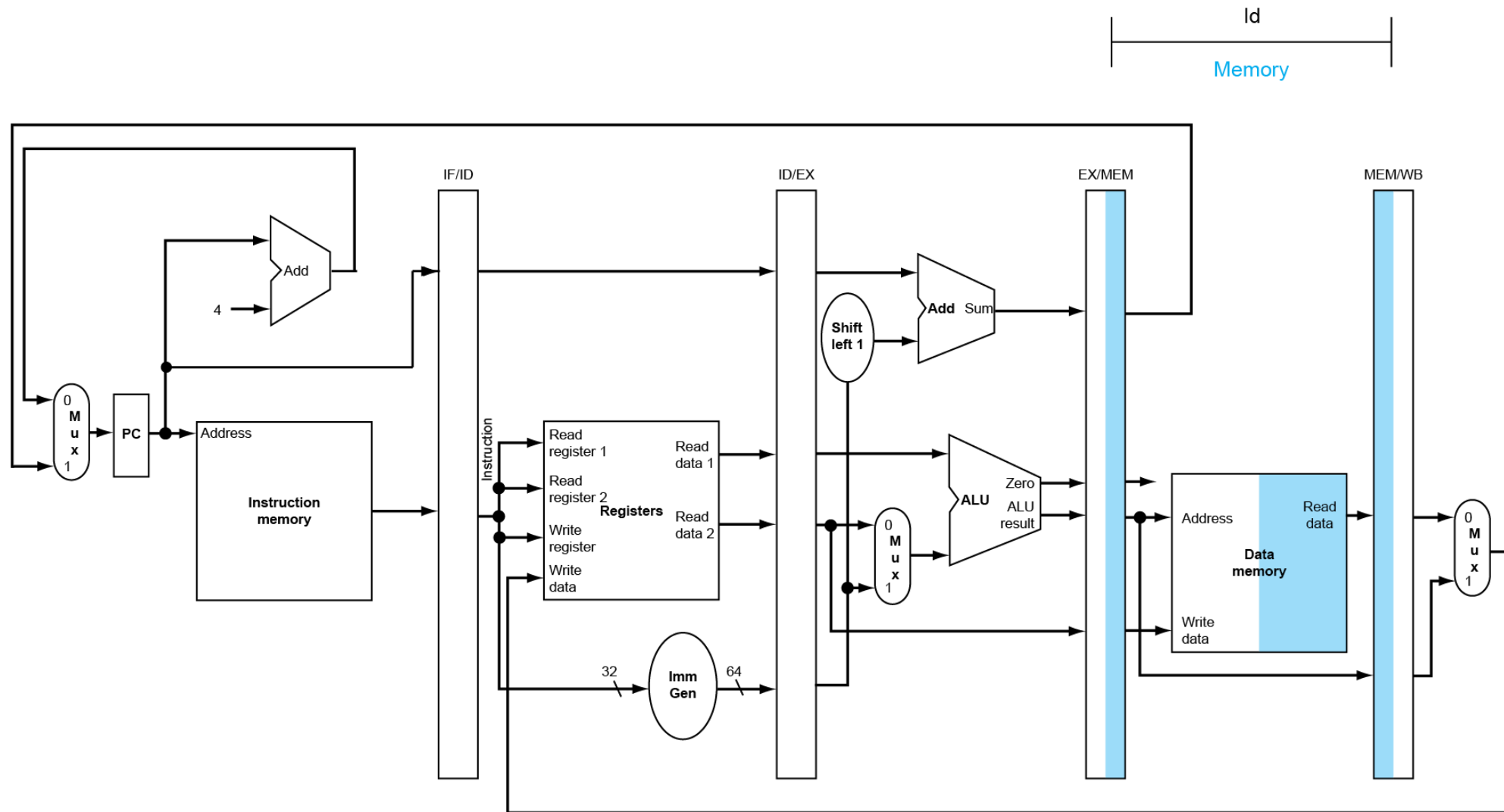




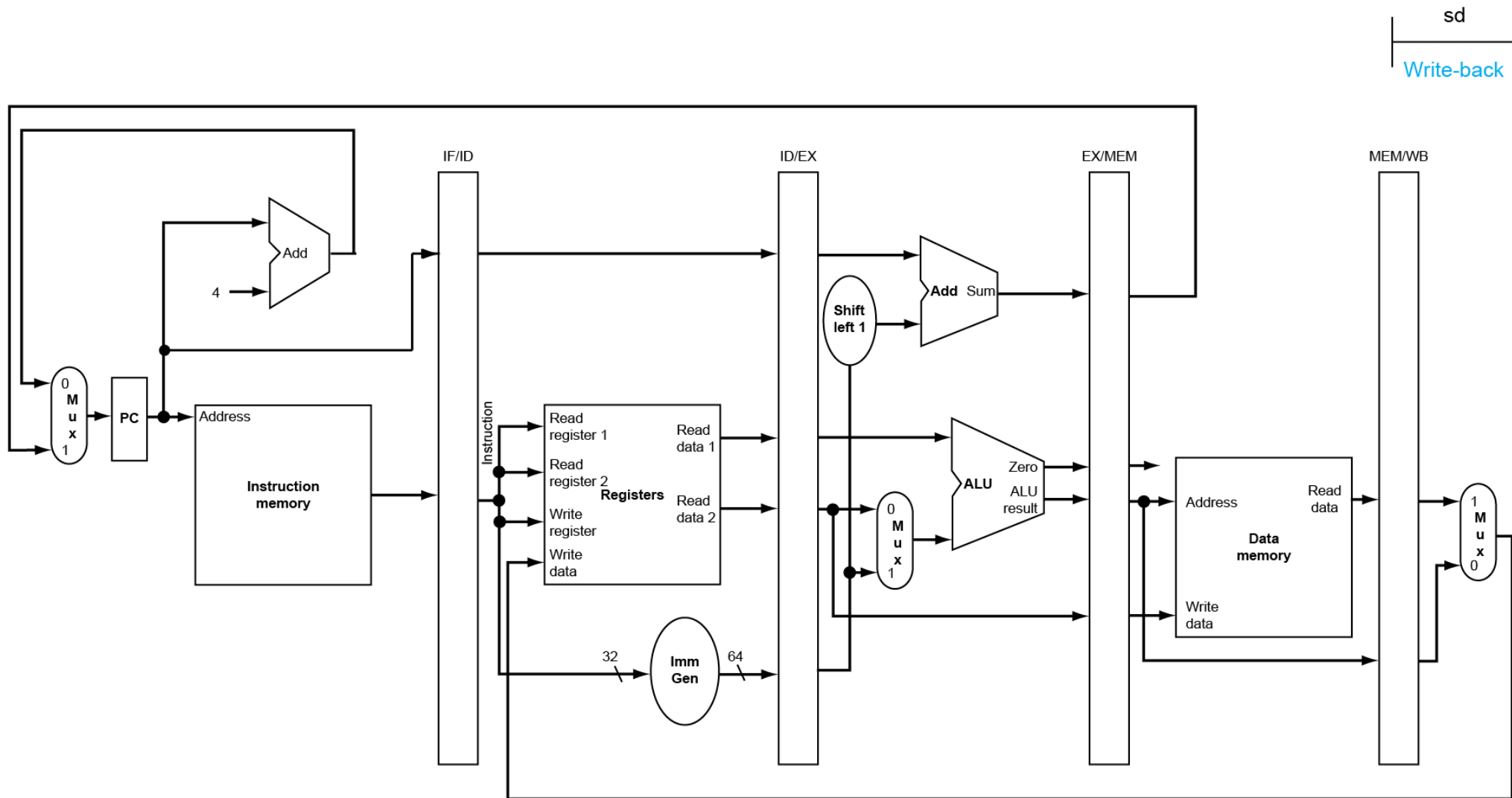
# EX for Store



# MEM for Store

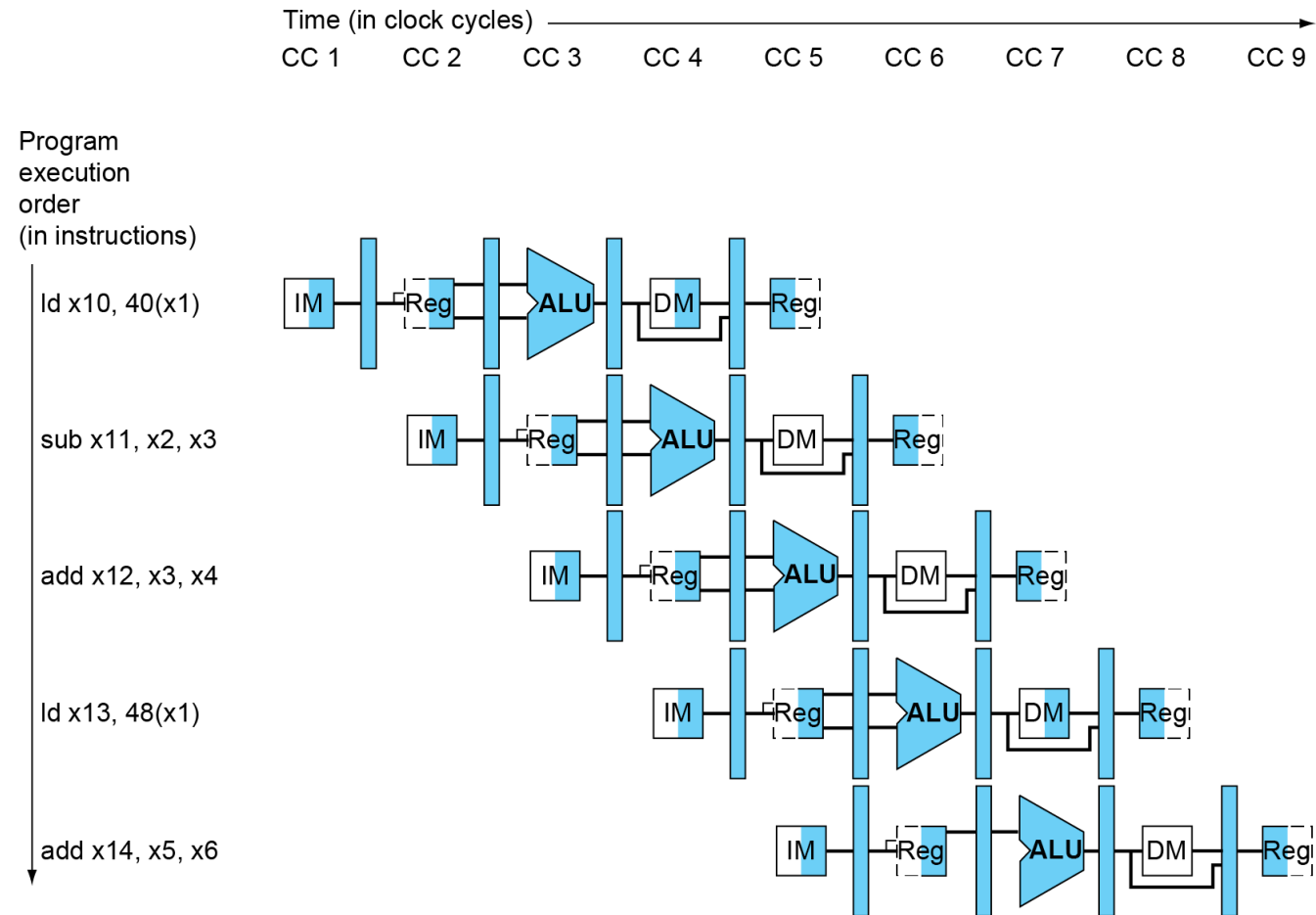


# WB for Store



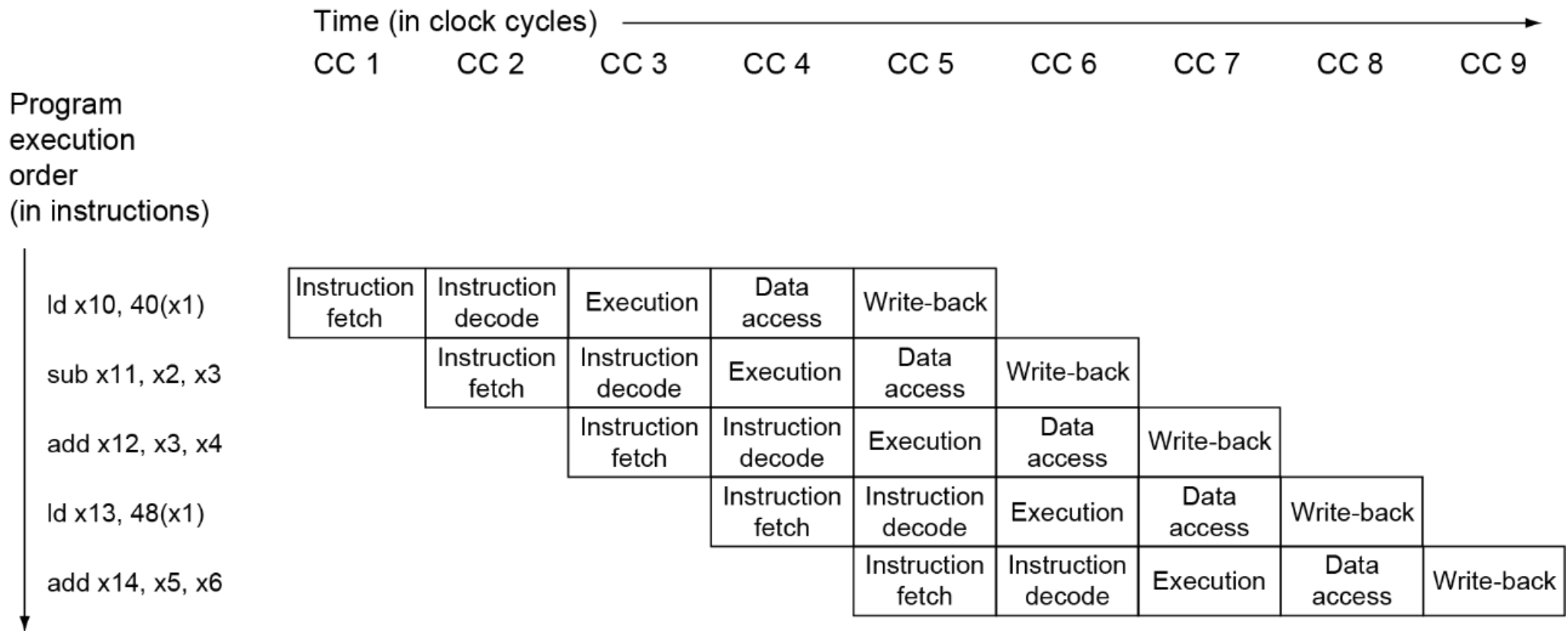
# Multi-Cycle Pipeline Diagram

- Form showing resource usage



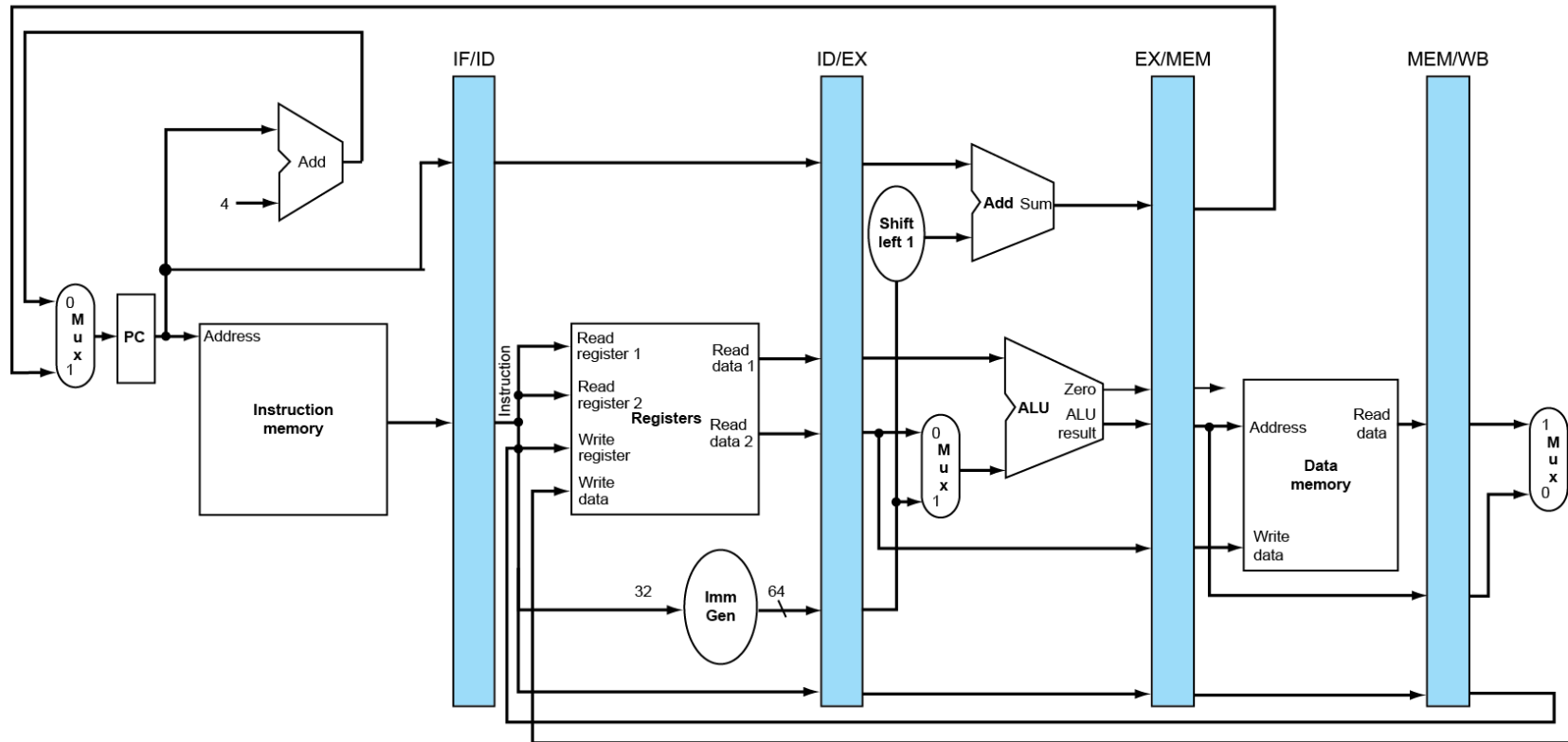
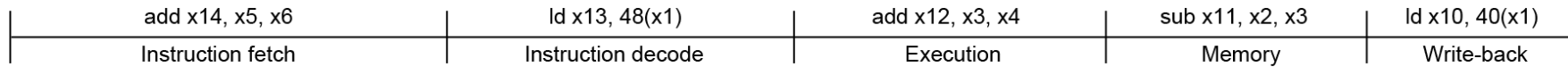
# Multi-Cycle Pipeline Diagram

- Traditional form

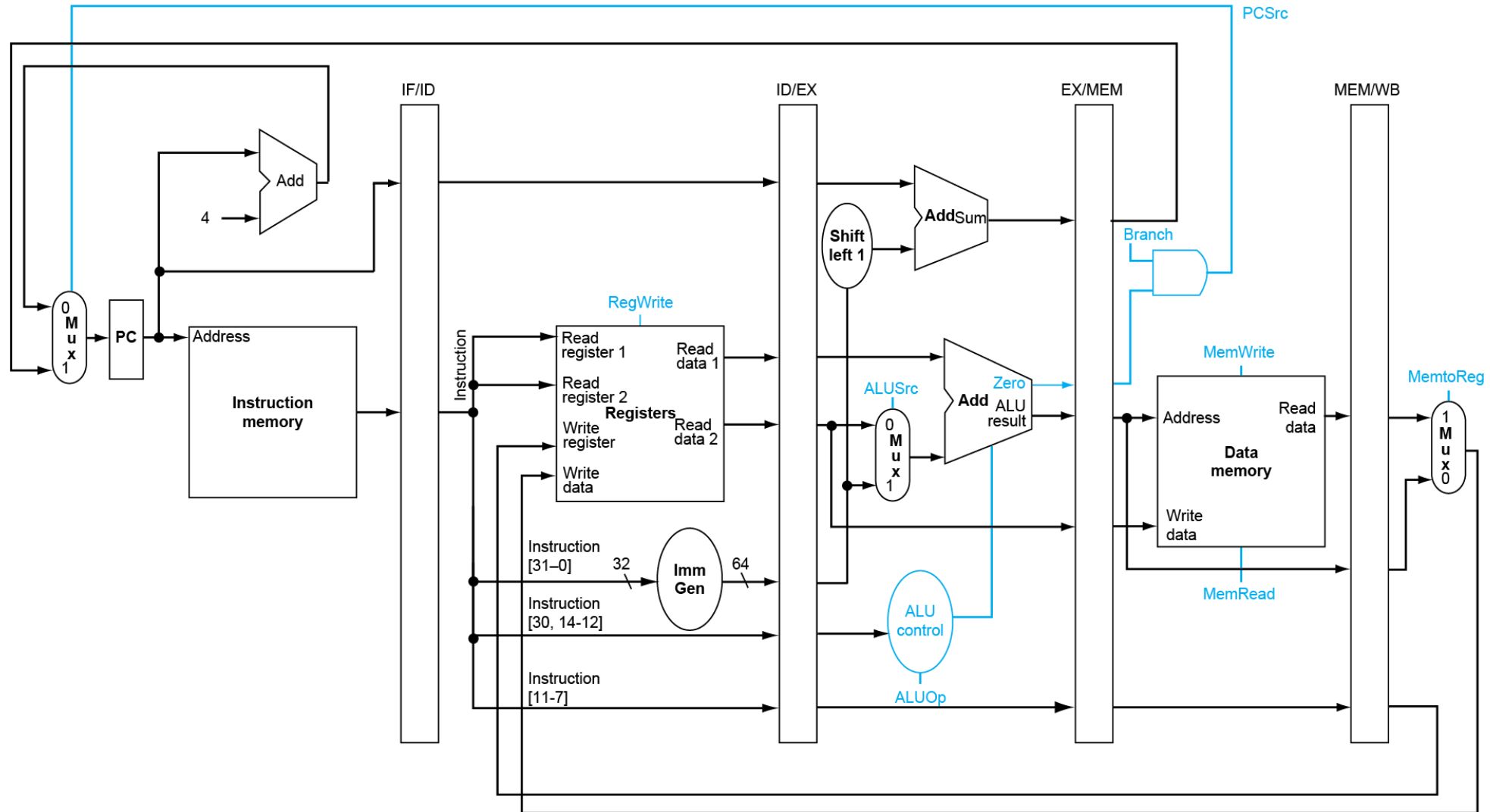


# Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle



# Pipelined Control (Simplified)



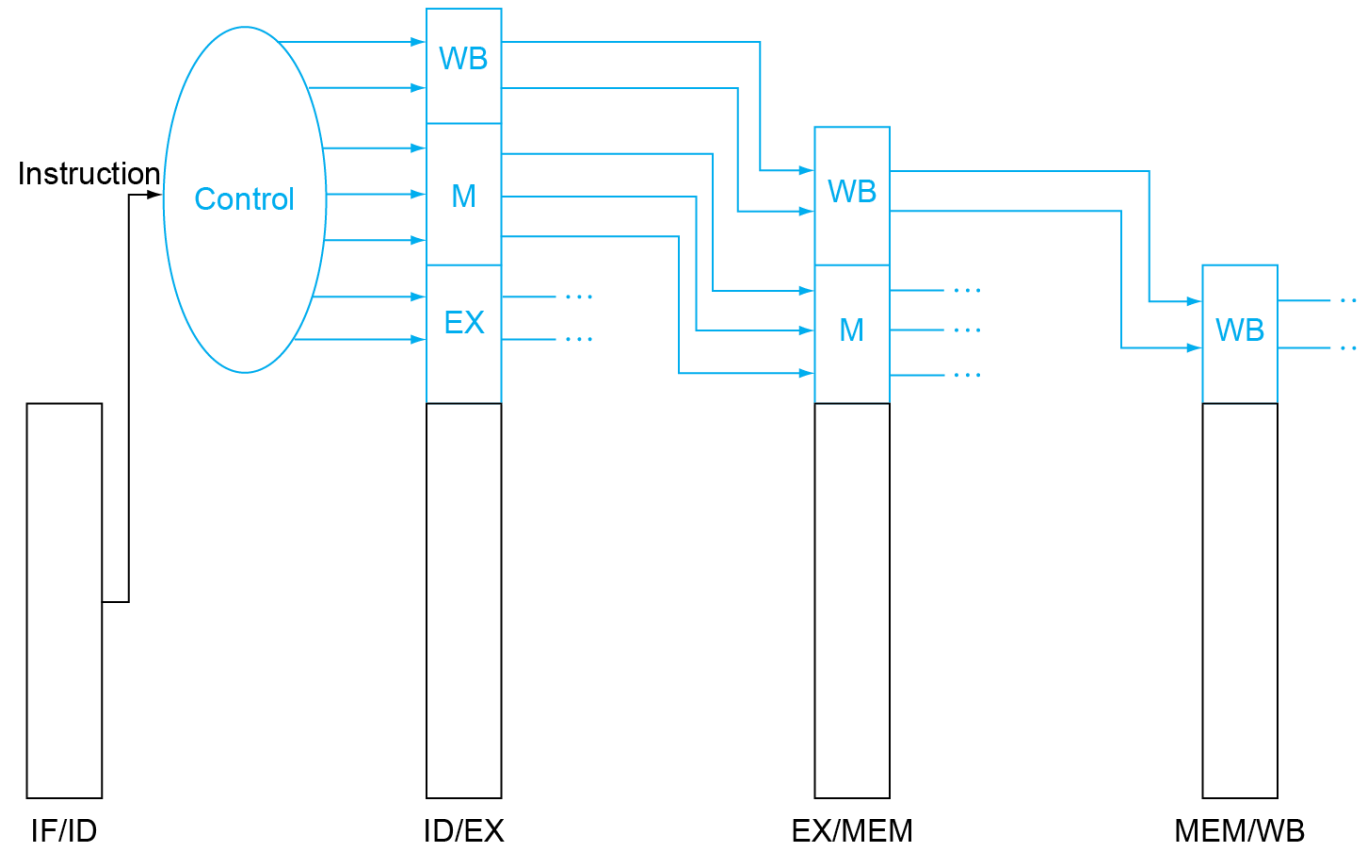
# Generating Control Signals

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



# Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation



# Pipelined Control

