Topics

- Single Cycle CPU Design (sections 4.1-4.4)

- **Pipelining**
  - Overview (section 4.6)
  - ➔ Pipelined Datapath and control (section 4.7)
  - Data hazard (section 4.8)
  - Control hazard (section 4.9)

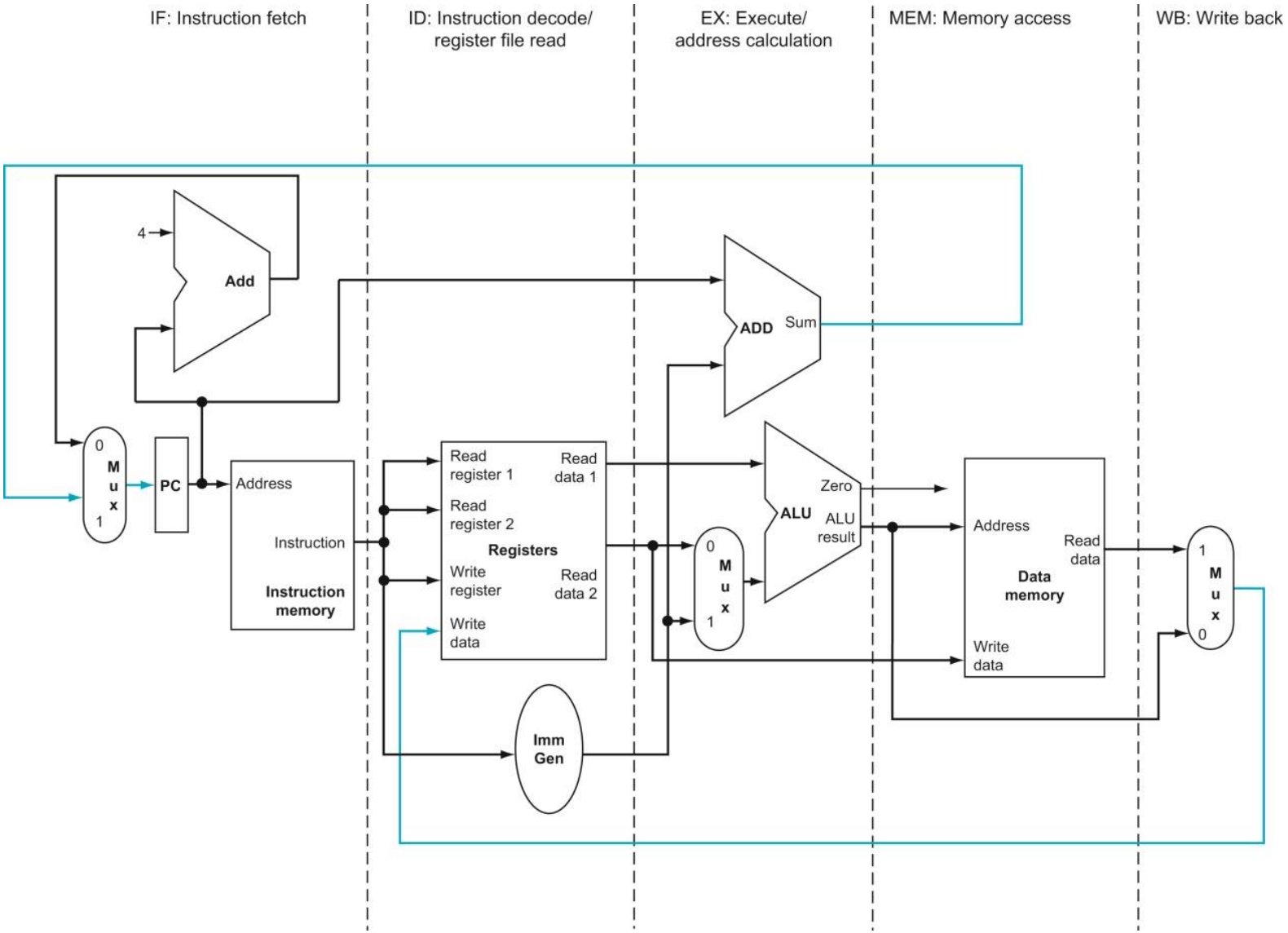
Thus times do shift,
each thing his turn does hold;
New things succeed,
as former things grow old.

Robert Herrick

Topics

- **Five functional units for pipelined datapath**
- **Graphically representing pipelines**
  - Single cycle
  - Multiple cycle
- **Control signals**

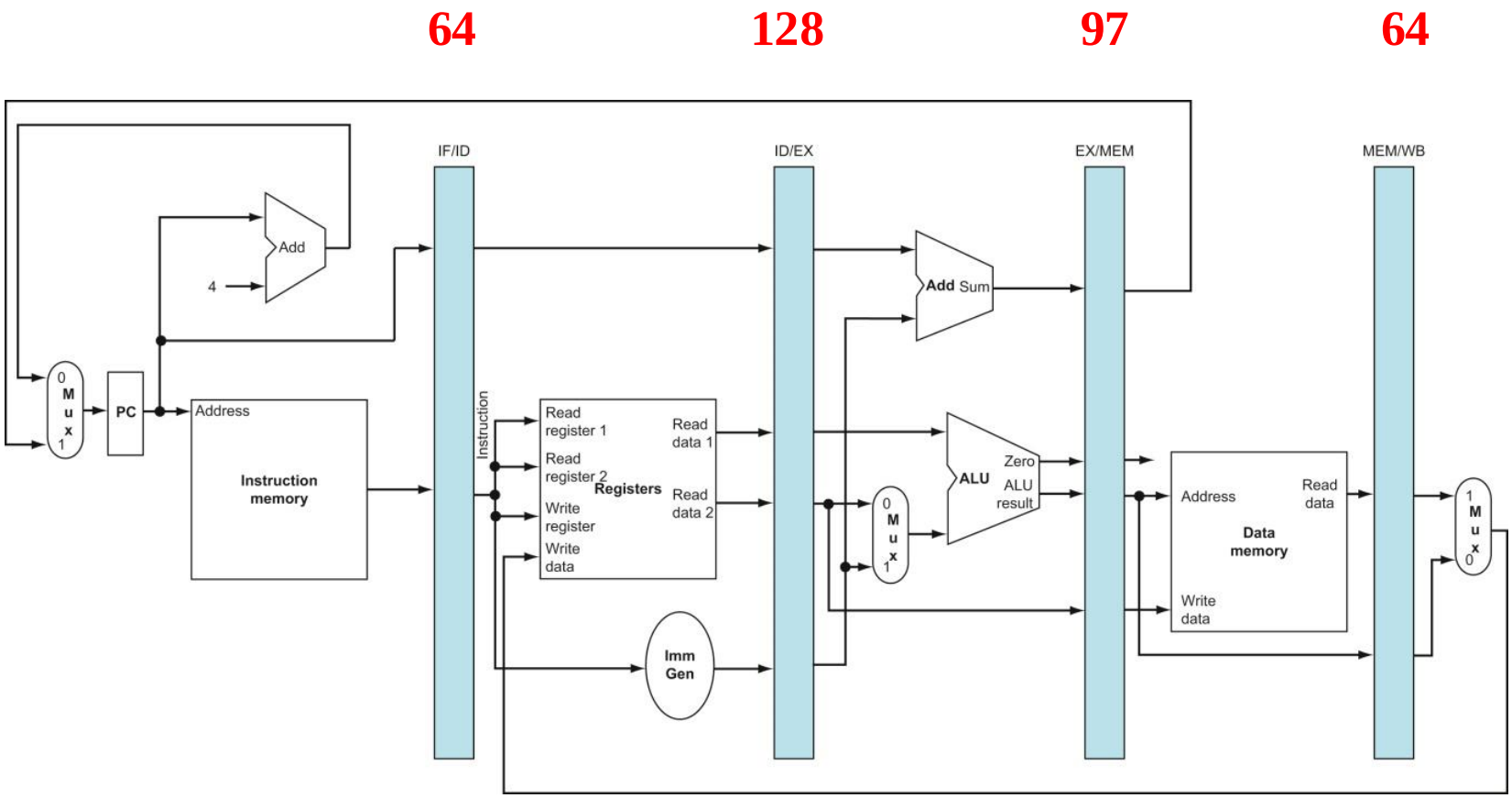
Review: Single-Cycle Datapath



Observations

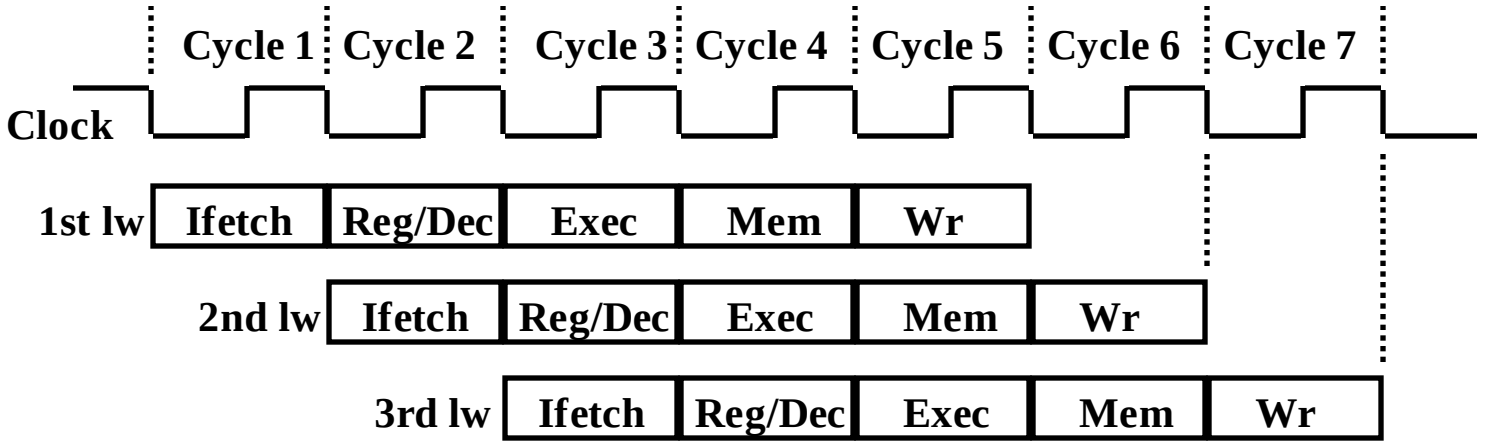
- **5-stage pipeline**
  - IF, ID, EX, MEM, WB
- **Left-to-right flow of instructions**
  - Instructions and data move generally from left to right
  - Two exceptions:
    - WB stage $\Rightarrow$ data hazards
    - the selection of PC $\Rightarrow$ control hazards
- **What to add to split the datapath into stages?**

Pipelined Datapath



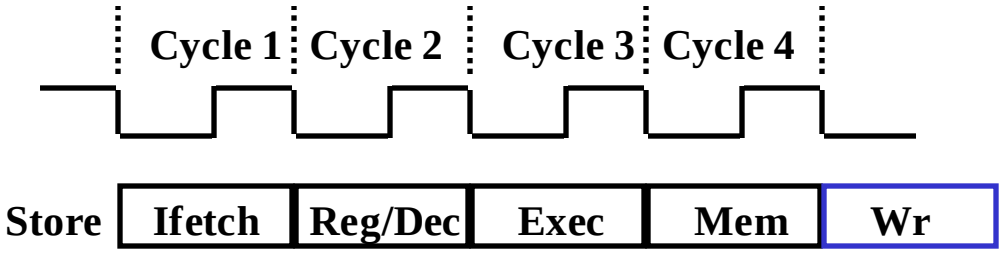
- **Add pipeline registers**
  - How many bits are stored in each pipeline register?
- **Why no pipeline register at the end of the WB stage?**

Pipelining the Load Instruction



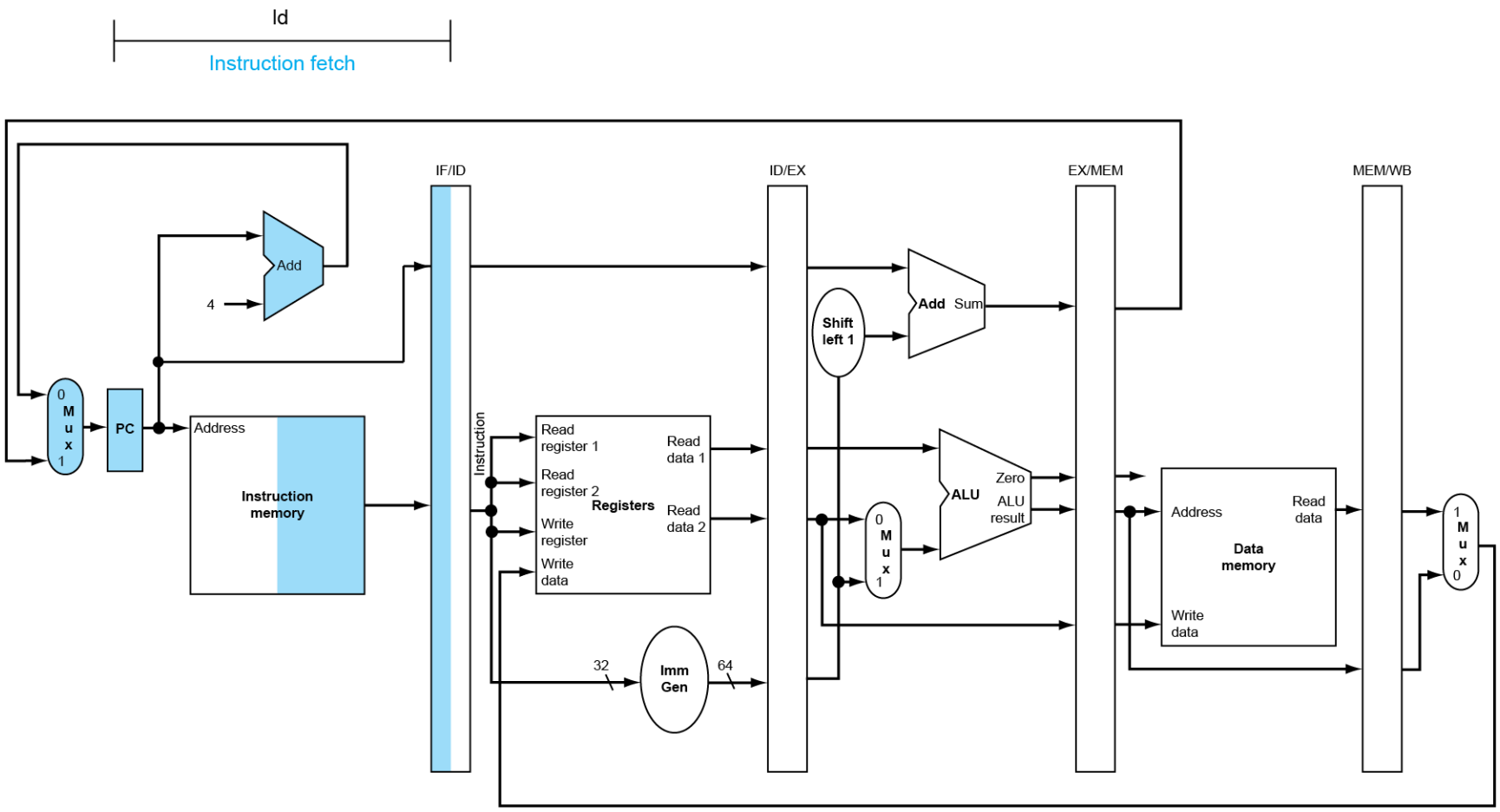
- **The five independent functional units in the pipeline datapath are:**
  - Instruction Memory for the Ifetch stage
  - Register File's Read ports (busA and busB) for the Reg/Dec stage
  - ALU for the Exec stage
  - Data Memory for the Mem stage
  - Register File's Write port (bus W) for the Wr stage
- **Effective (or average) number of cycles per instruction is ONE**

The Four Stages of Store

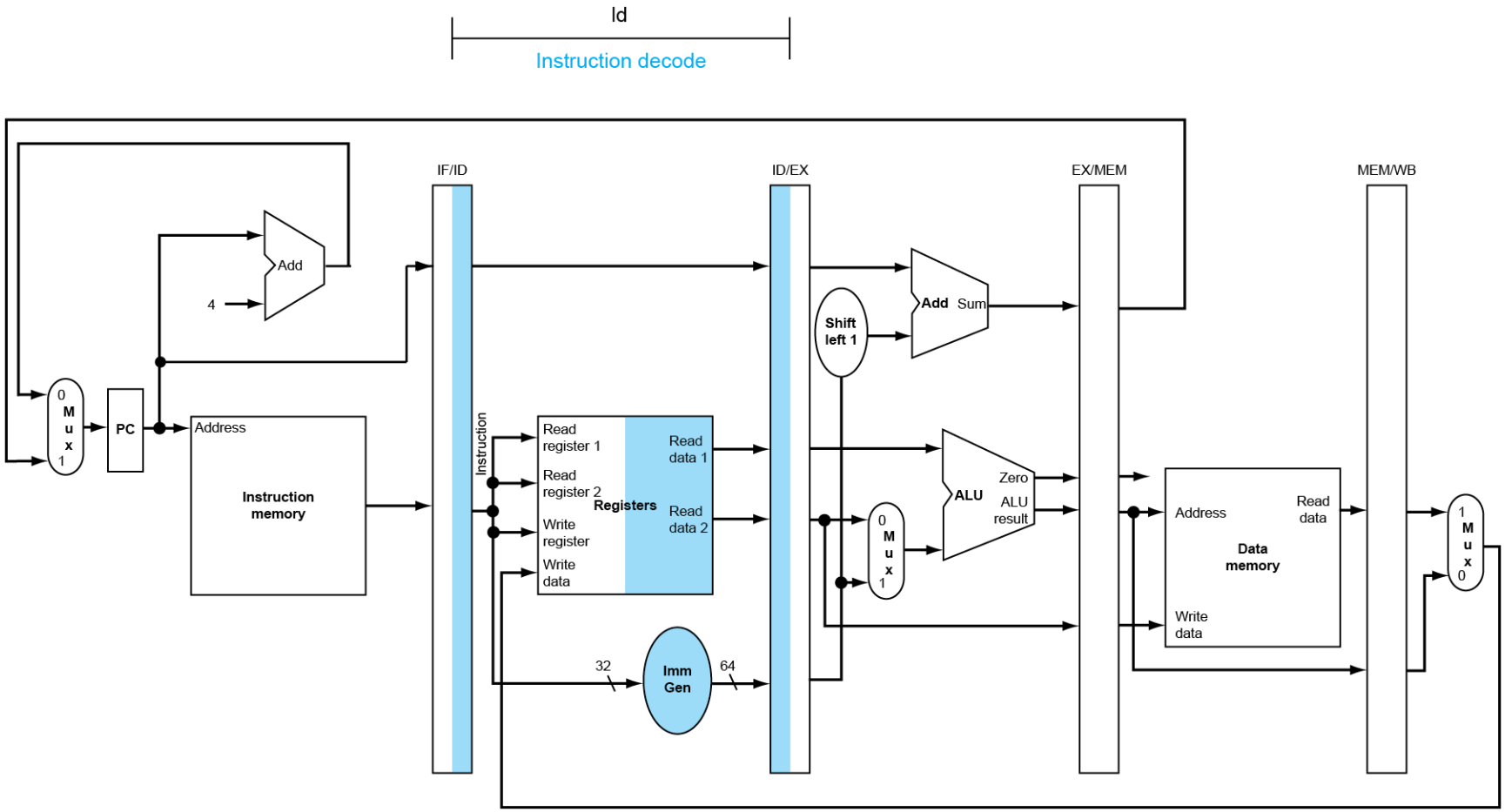


- **Ifetch: Instruction Fetch**
  - Fetch the instruction from the Instruction Memory
- **Reg/Dec: Registers Fetch and Instruction Decode**
- **Exec: Calculate the memory address**
- **Mem: Write the data into the Data Memory**

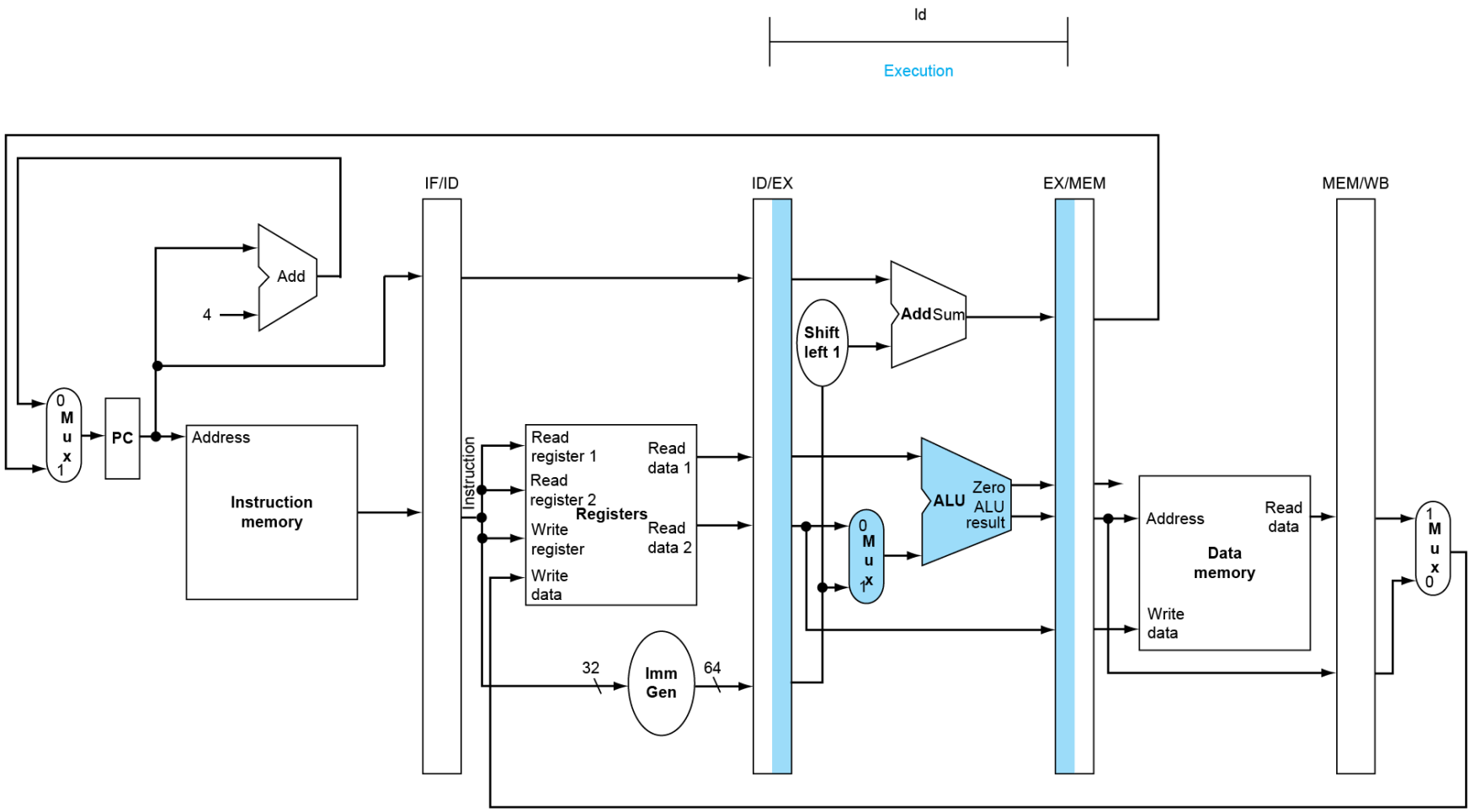
IF for Load, Store, …



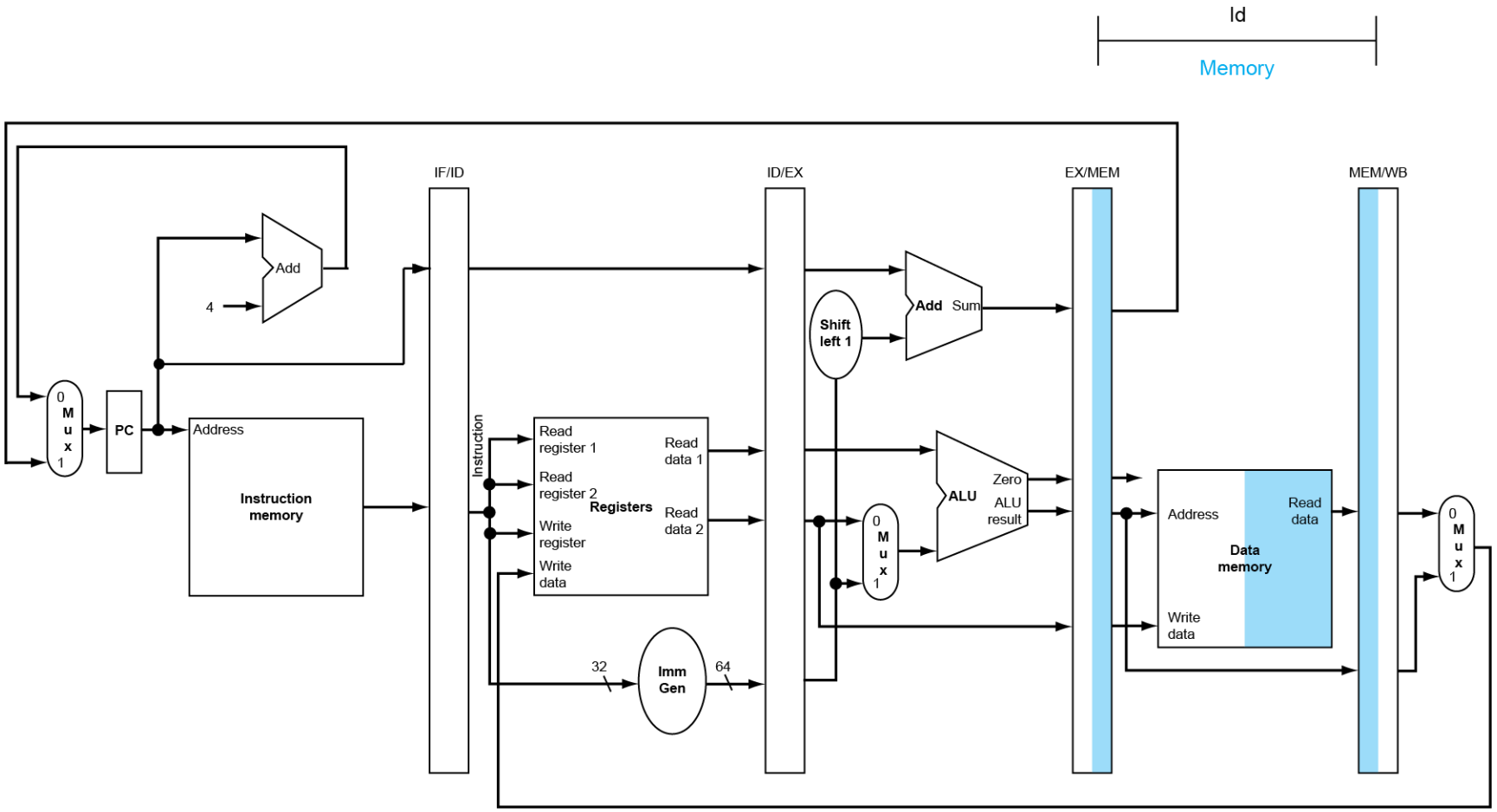
ID for Load, Store, …



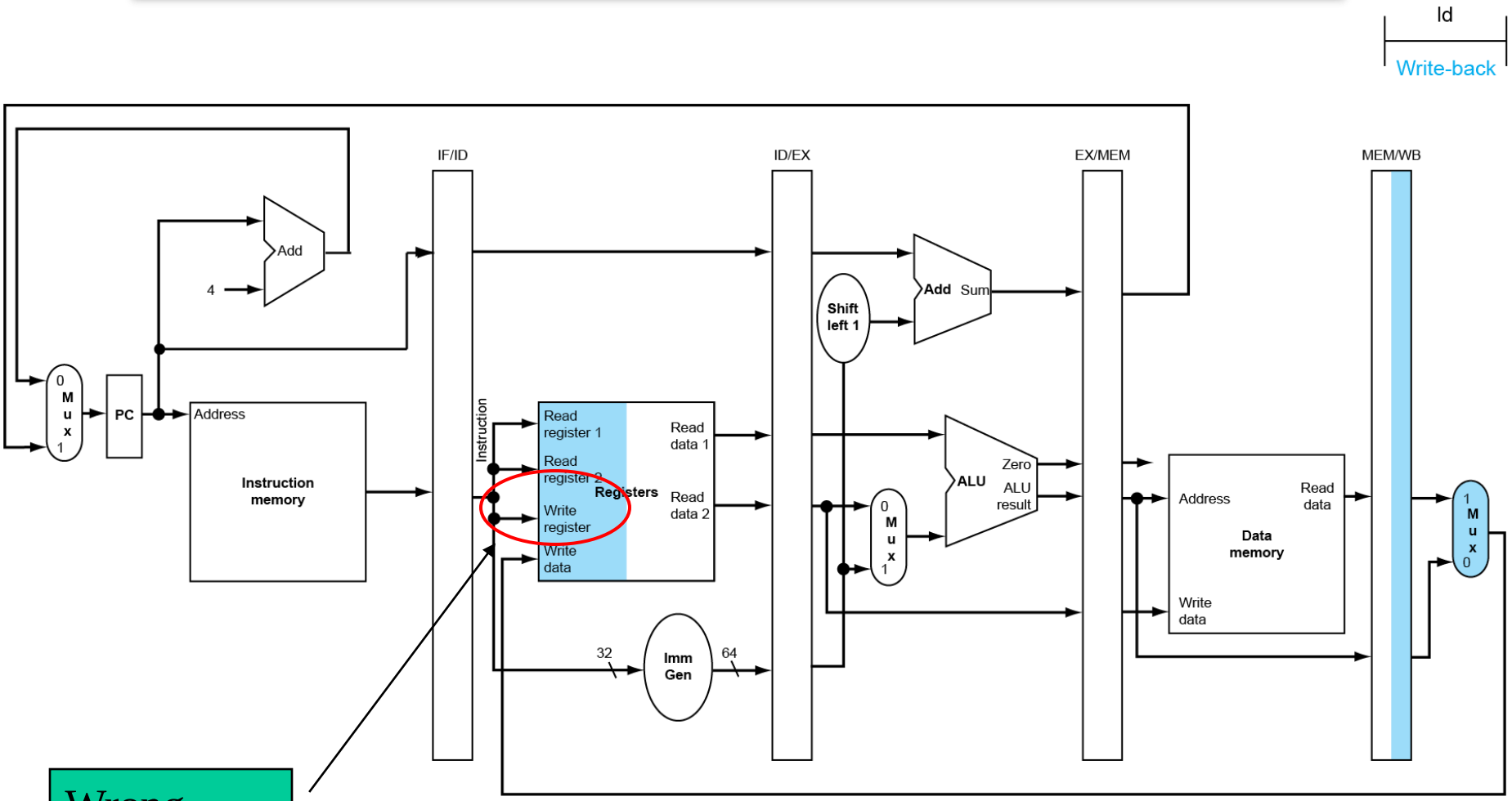
EX for Load


Id
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Imm Gen
64
Shift left 1
Add
Sum
0
M
u
x
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
M
u
x
0

MEM for Load

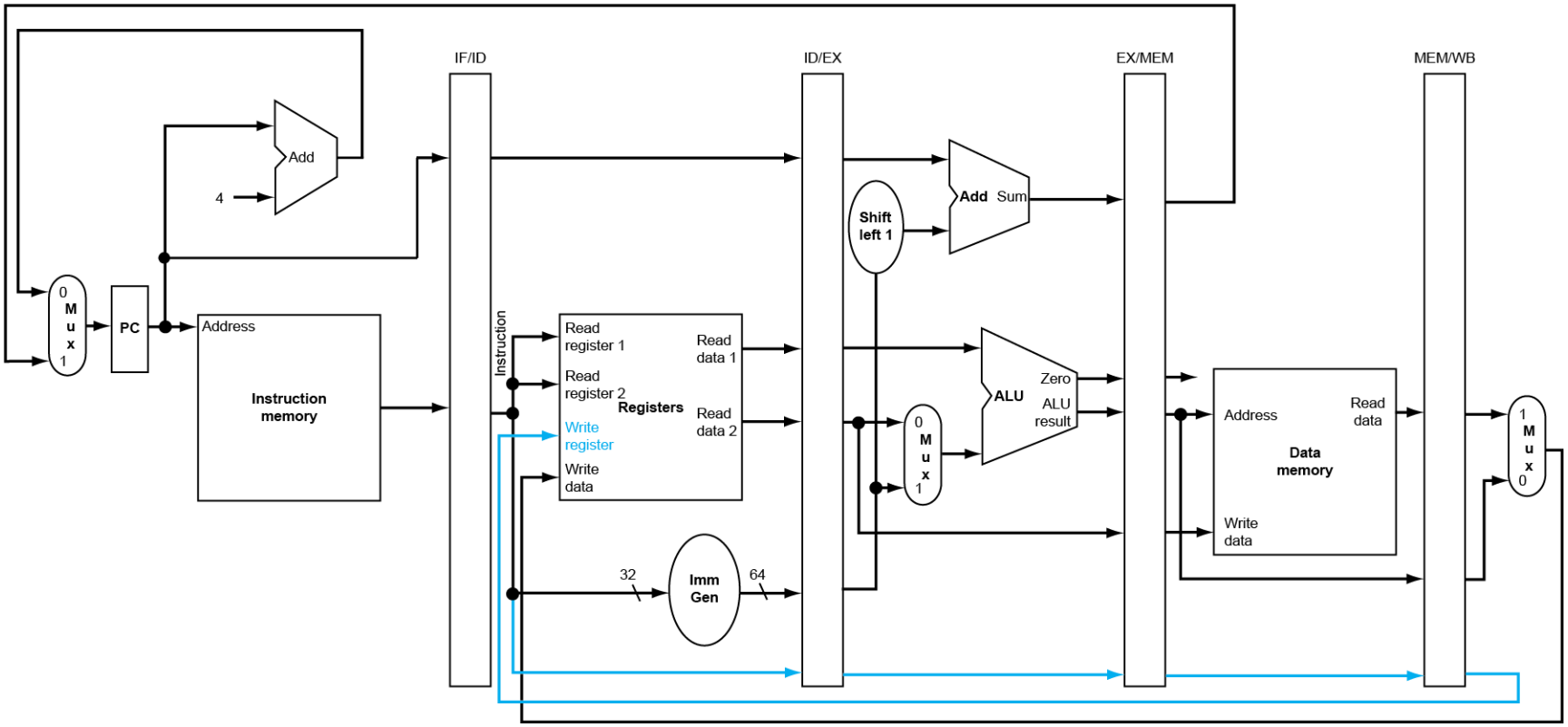

ld
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Imm Gen
64
Shift left 1
Add
Sum
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

WB for Load

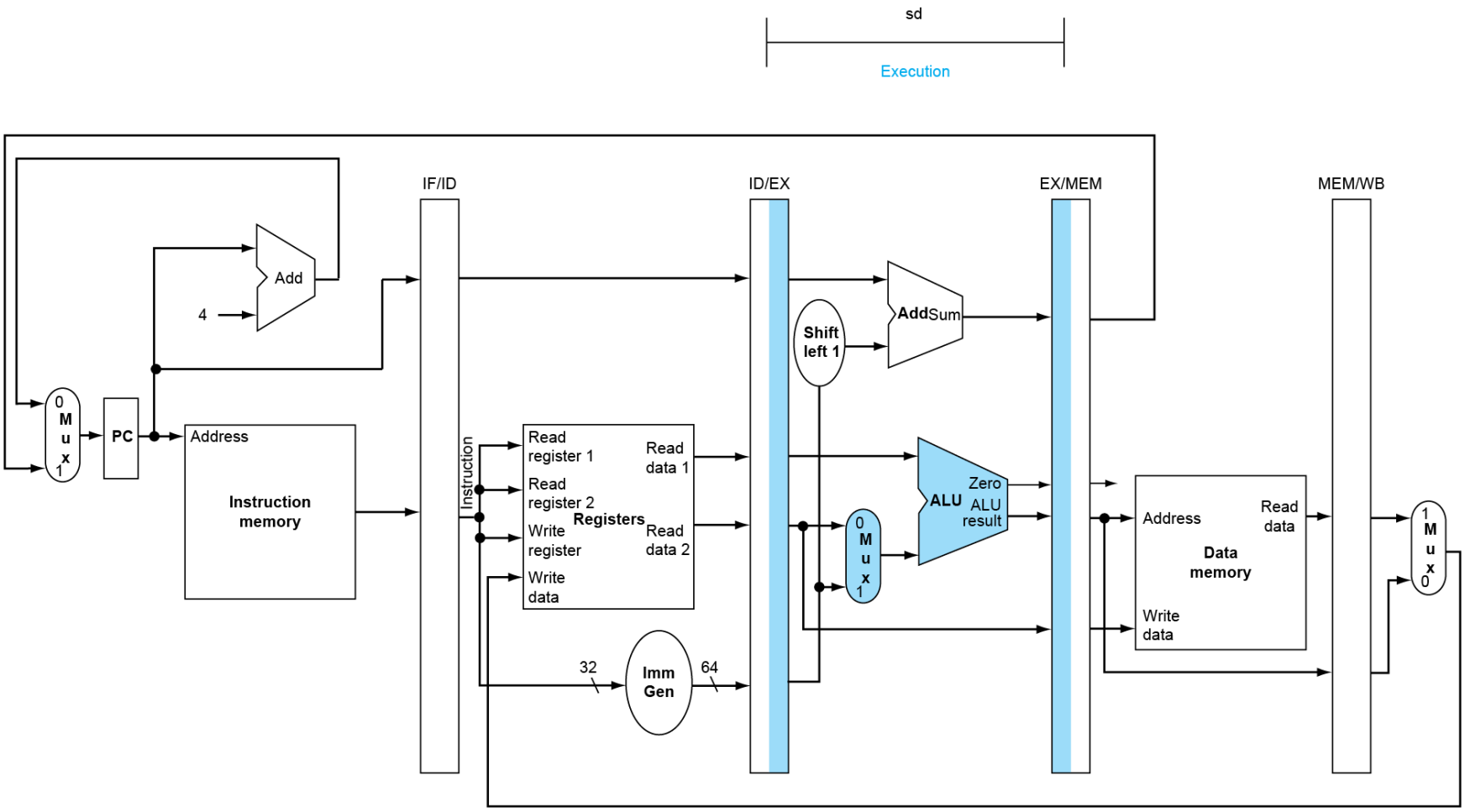


Wrong register number

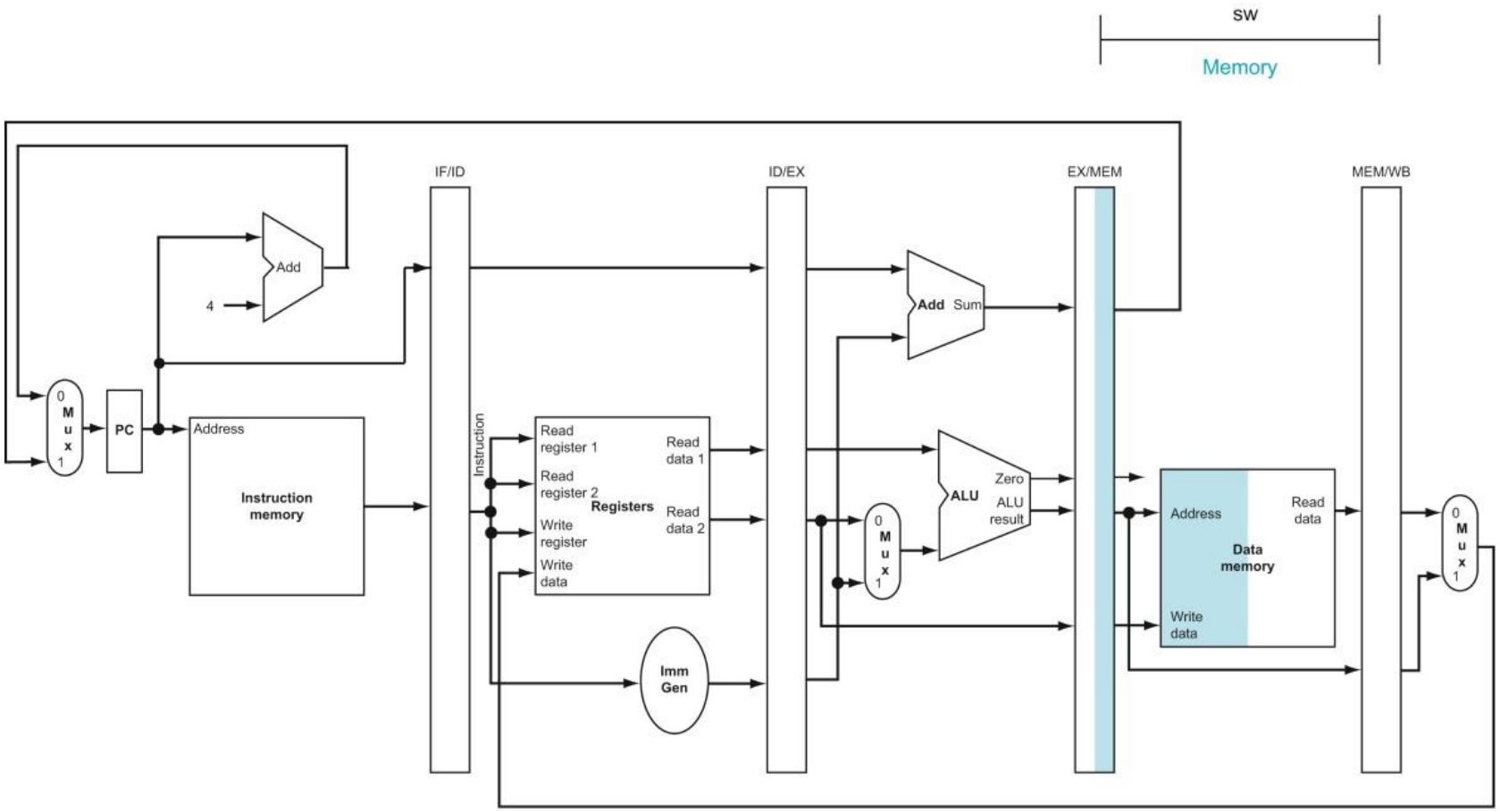
Corrected Datapath for Load



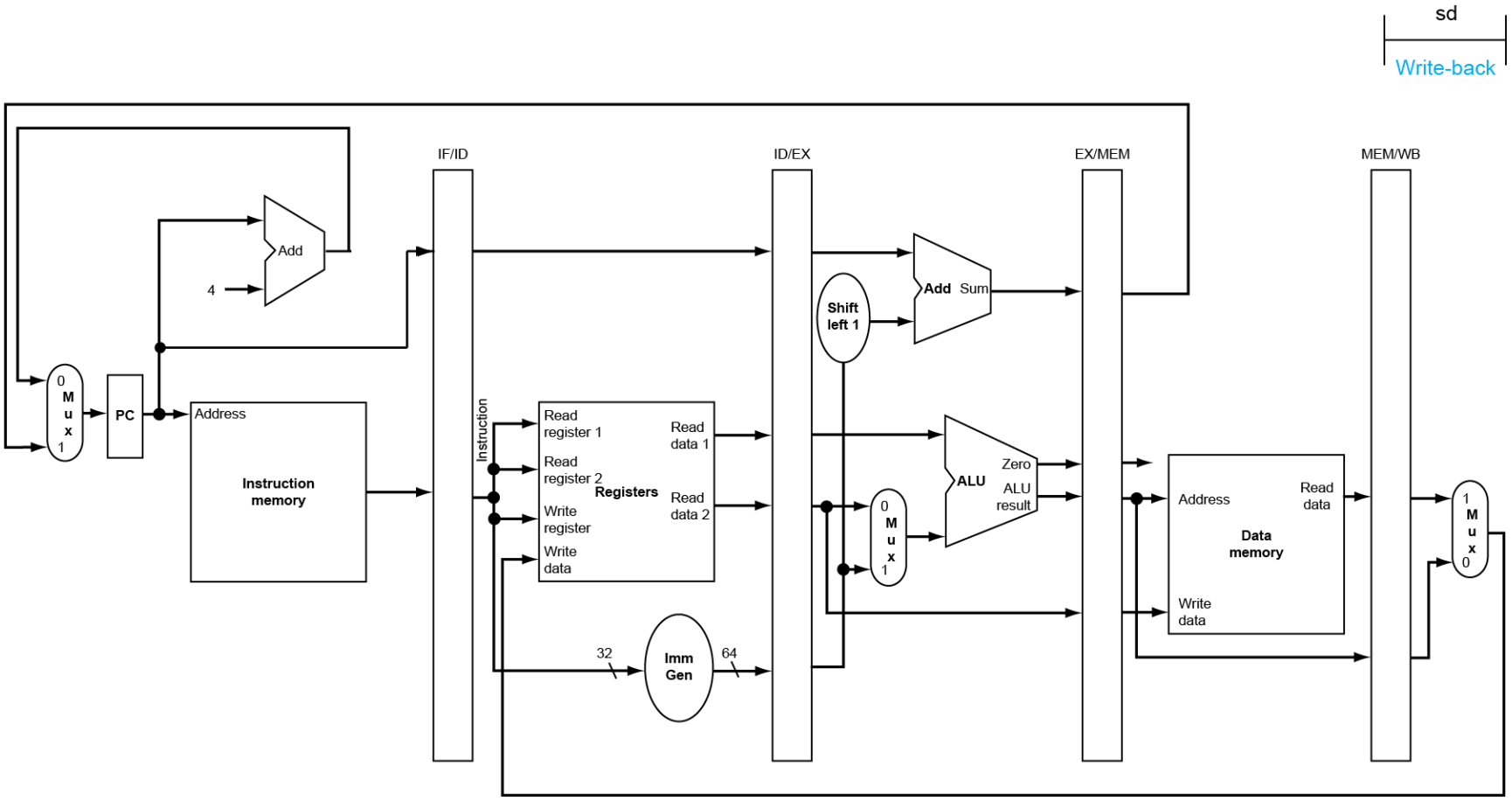
EX for Store


sd
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Imm Gen
64
Shift left 1
AddSum
ALU
Zero
ALU result
0
M
u
x
1
Address
Read data
Data memory
Write data
1
M
u
x
0

MEM for Store



WB for Store

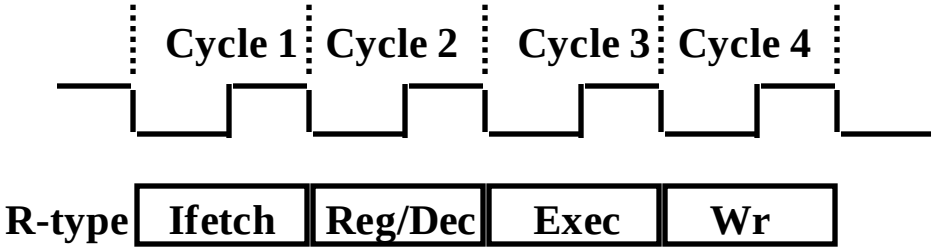

sd
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Imm Gen
64
Shift left 1
Add
Sum
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

Observations from Load and Store

- **Pass information needed from an earlier stage to a later stage**

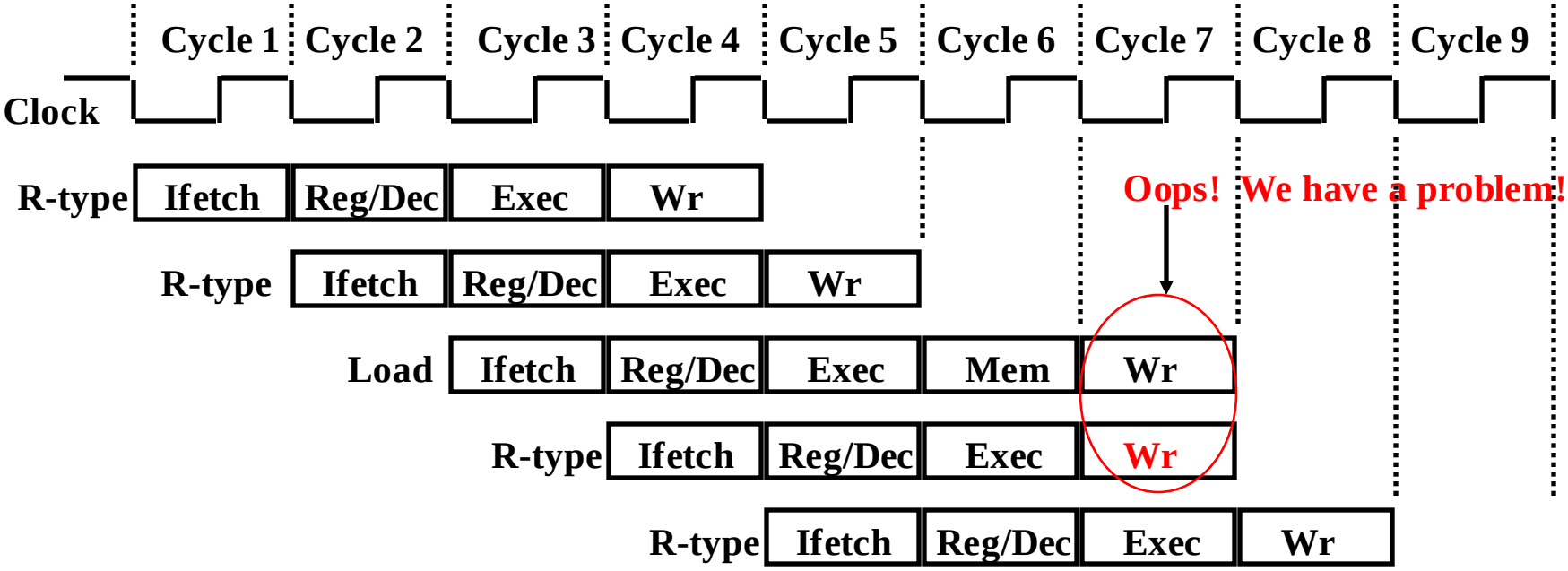
- **Each logical component of the datapath can be used only within a single pipeline stage**
  - functional units: IM, Reg read ports, ALU, DM, Reg write port
  - Otherwise, we would have structural hazard

The Four Stages of R-type



- **Ifetch: Instruction Fetch**
  - Fetch the instruction from the Instruction Memory
- **Reg/Dec: Registers Fetch and Instruction Decode**
- **Exec:**
  - ALU operates on the two register operands
- **Wr: Write the ALU output back to the register file**

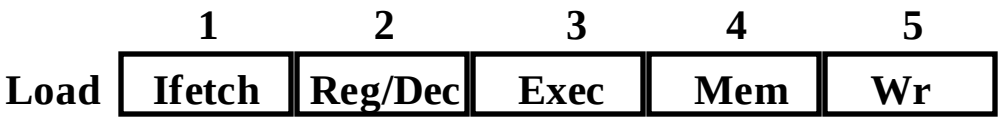
Pipelining the R-type and Load Instructions



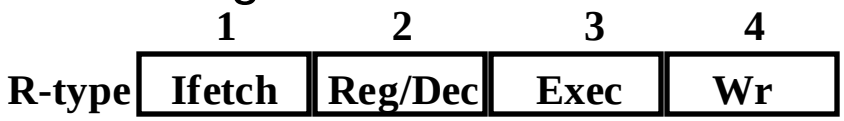
- **We have pipeline conflict or structural hazard:**
  - Two instructions try to write to the register file at the same time!
  - Only one write port

Important Observation

- **Each functional unit can only be used once per instruction**
  - necessary condition for pipelining to work
- **Each functional unit must be used at the same stage for all instructions:**
  - Sufficient condition for pipelining to work
  - Load uses Register File's Write Port during its 5th stage

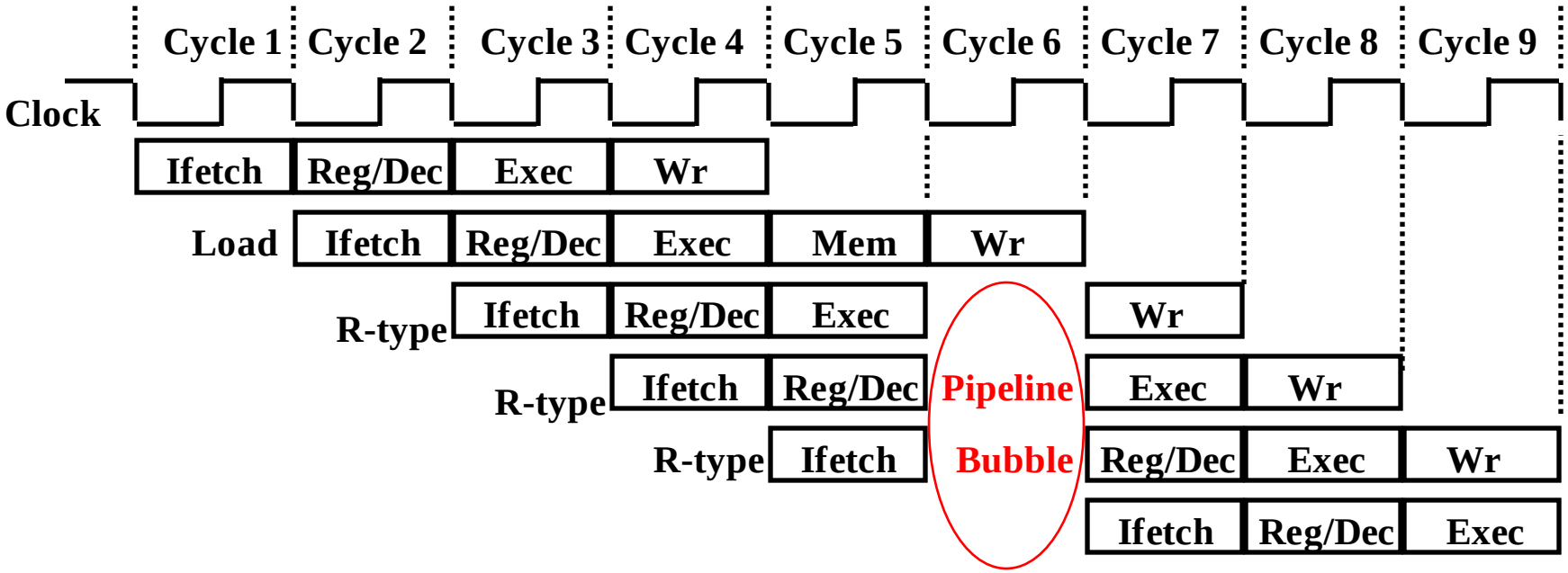


  - R-type uses Register File's Write Port during its 4th stage



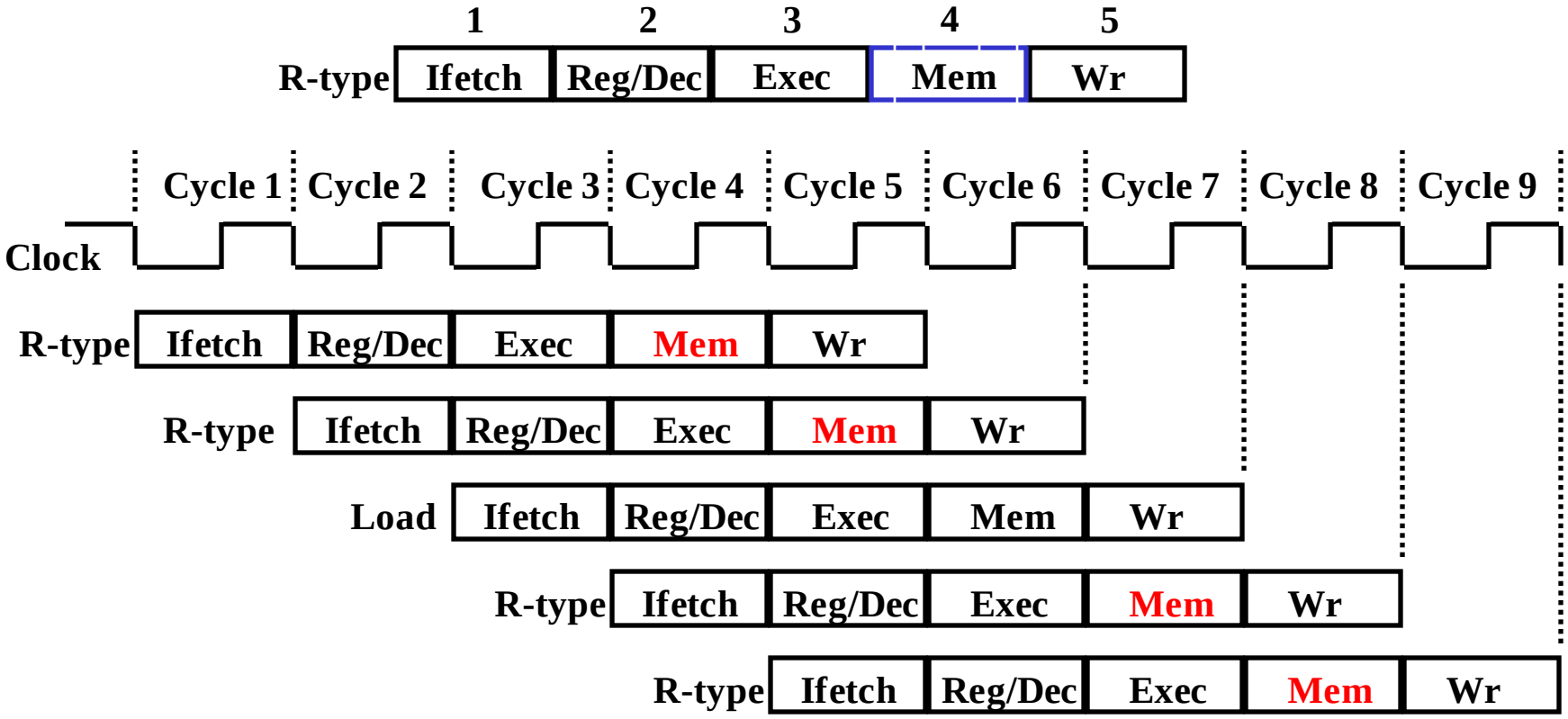
- **2 Ways to resolve the pipeline hazard**

Solution 1: Insert "Bubble" into the Pipeline



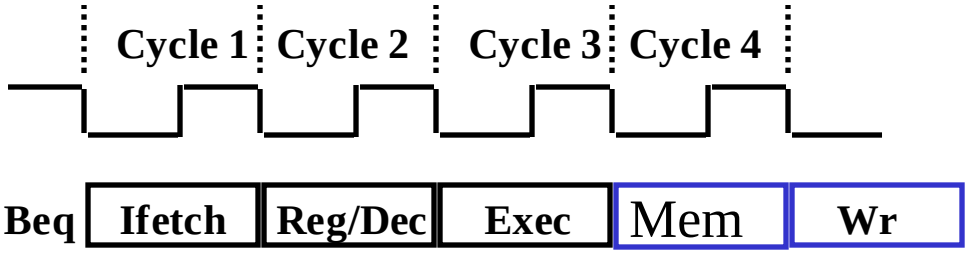
- **Insert a "bubble" into the pipeline after lw to prevent 2 writes at the same cycle**
  - The control logic can be complex
  - Lose instruction fetch and issue opportunity
  - No instruction is started in Cycle 6!

Solution 2: Delay R-type's Write by One Cycle



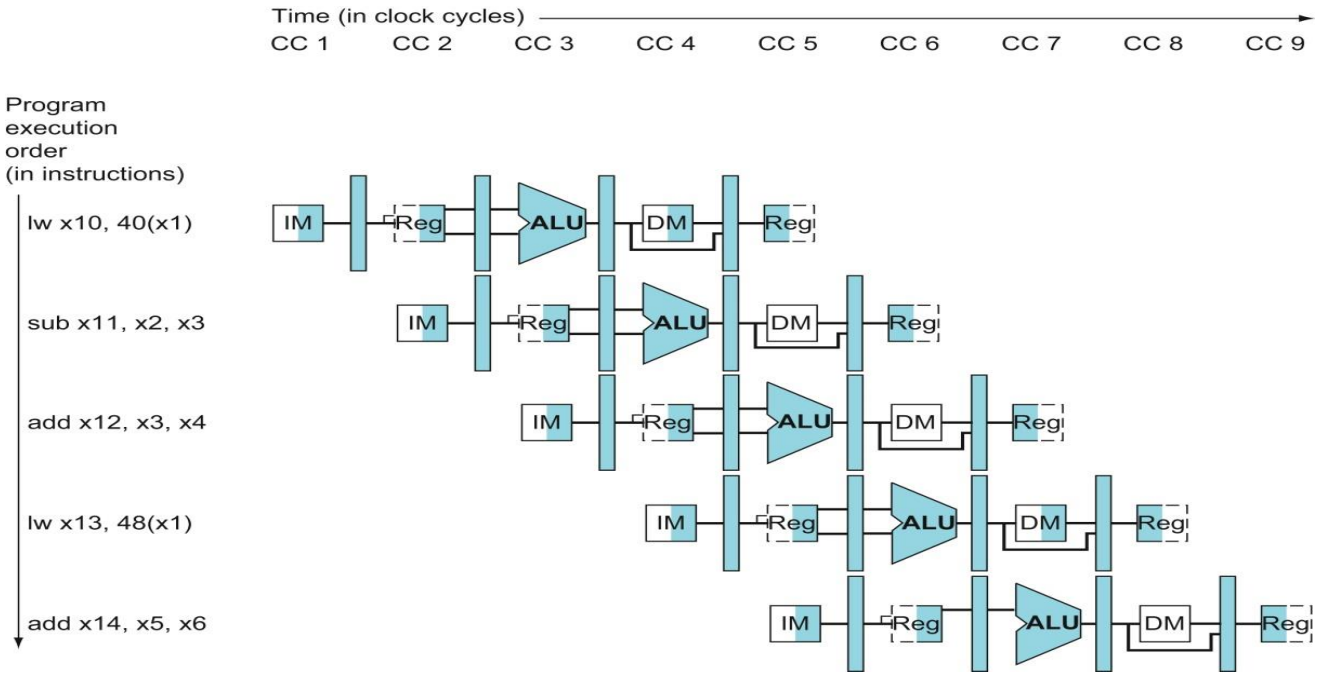
- **Delay R-type's register write by one cycle:**
  - Now R-type instructions also use Reg File's write port at Stage 5
  - Mem stage is a NOOP stage: nothing is being done

The Four Stages of Beq



- **Ifetch: Instruction Fetch**
  - Fetch the instruction from the Instruction Memory
- **Reg/Dec:**
  - Registers Fetch and Instruction Decode
- **Exec:**
  - compares the two register operand,
  - calculate branch target address
- **MEM**
  - Use the Zero result to decide which adder result to store into PC

Multiple-clock-cycle Pipeline Diagram



- **Can help with answering questions like:**
  - how many cycles does it take to execute this code?
  - what is the ALU doing during cycle 4?

Single Clock Cycle Pipeline Diagram

State of pipeline in a given cycle (cycle 5)

add x14, x5, x6	lw x13, 48(x1)	add x12, x3, x4	sub x11, x2, x3	lw x10, 40(x1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back