# The Processor

Computer Architecture

2019 1학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

# Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

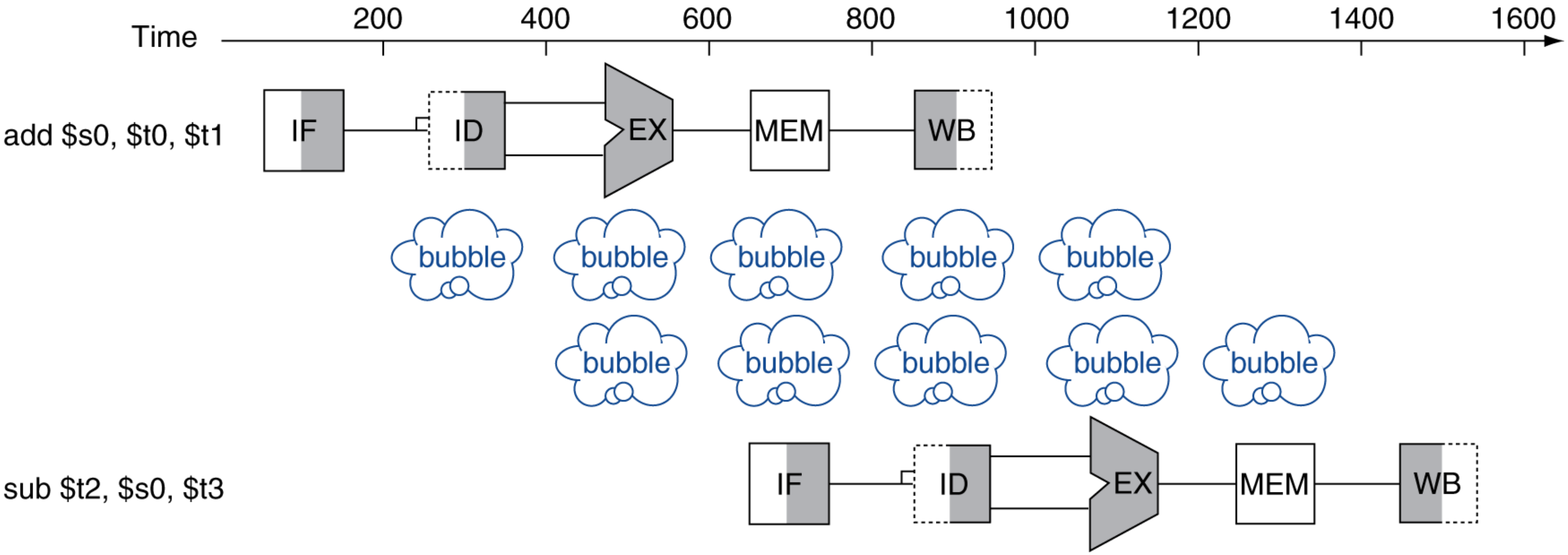
# Structure Hazards

- Conflict for use of a 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 Hazards

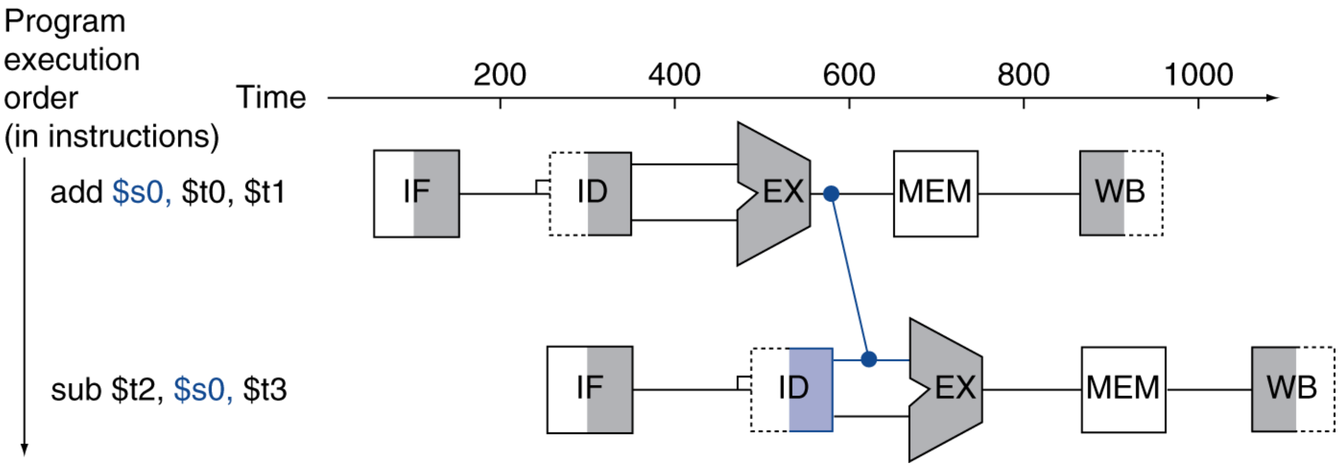
- An instruction depends on completion of data access by a previous instruction

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



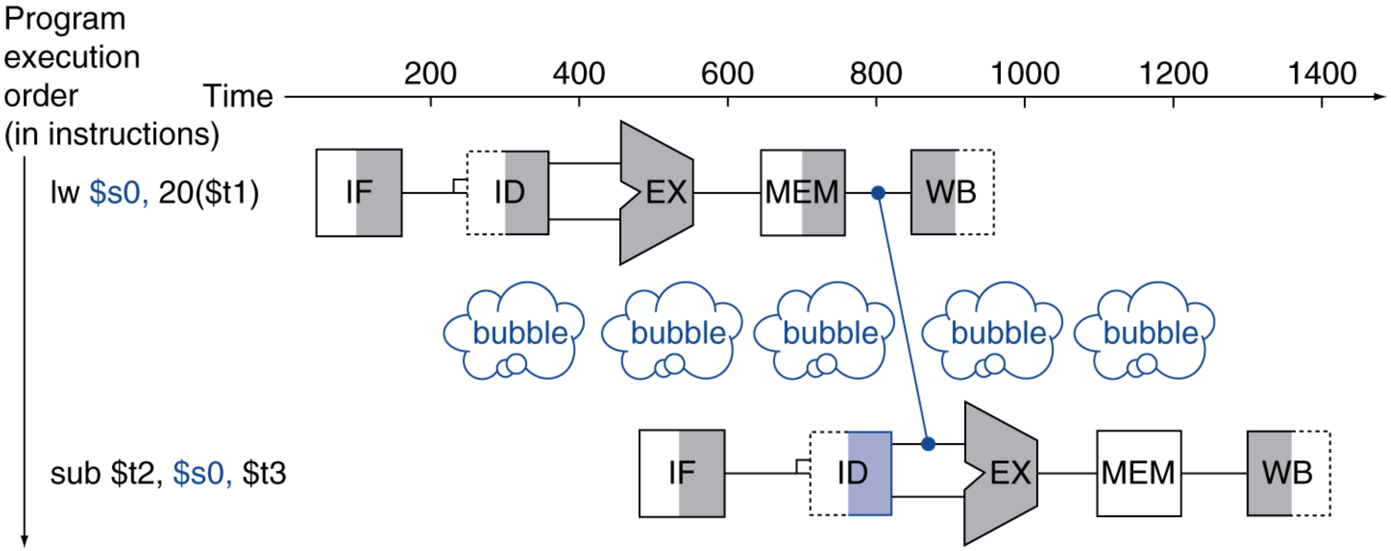
# Forwarding (aka Bypassing)

- Use result when it is computed
  - Don’t wait for it to be stored in a register
  - Requires extra connections in the datapath



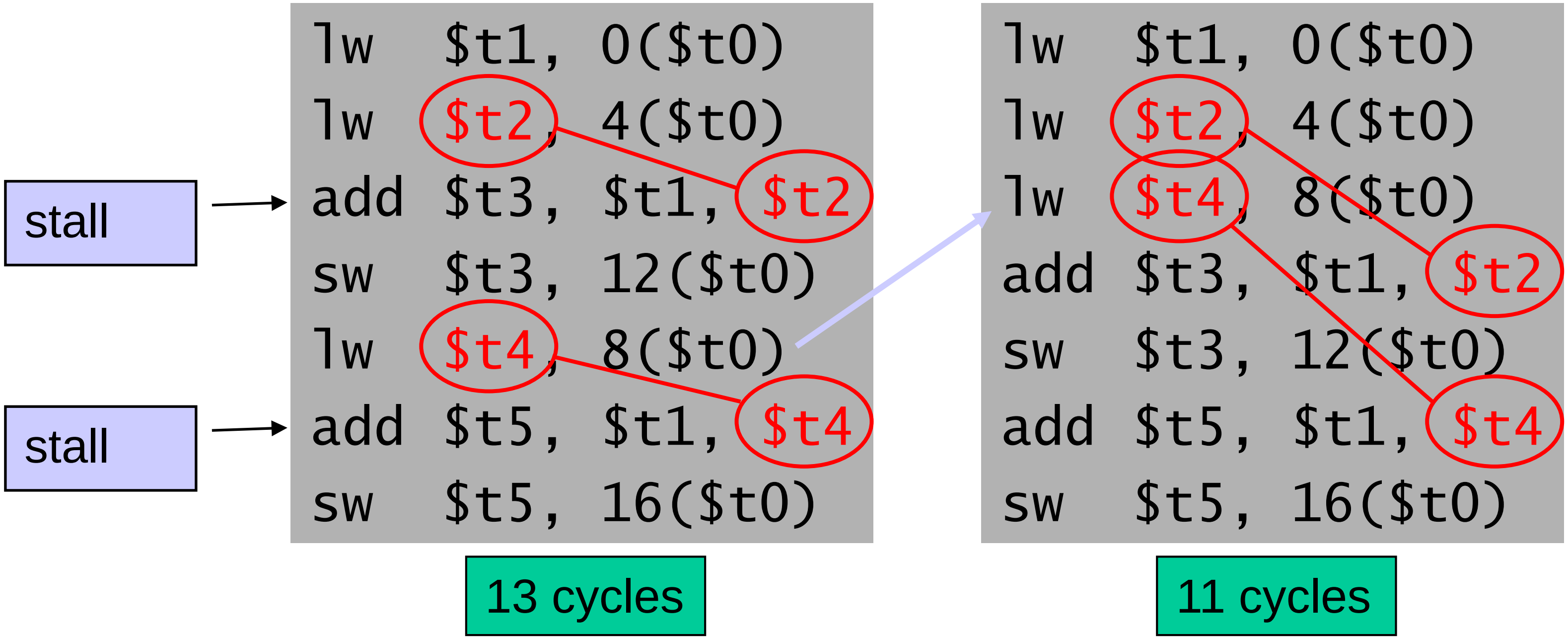
# Load-Use Data Hazard

- 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;`



stall

stall

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

13 cycles

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

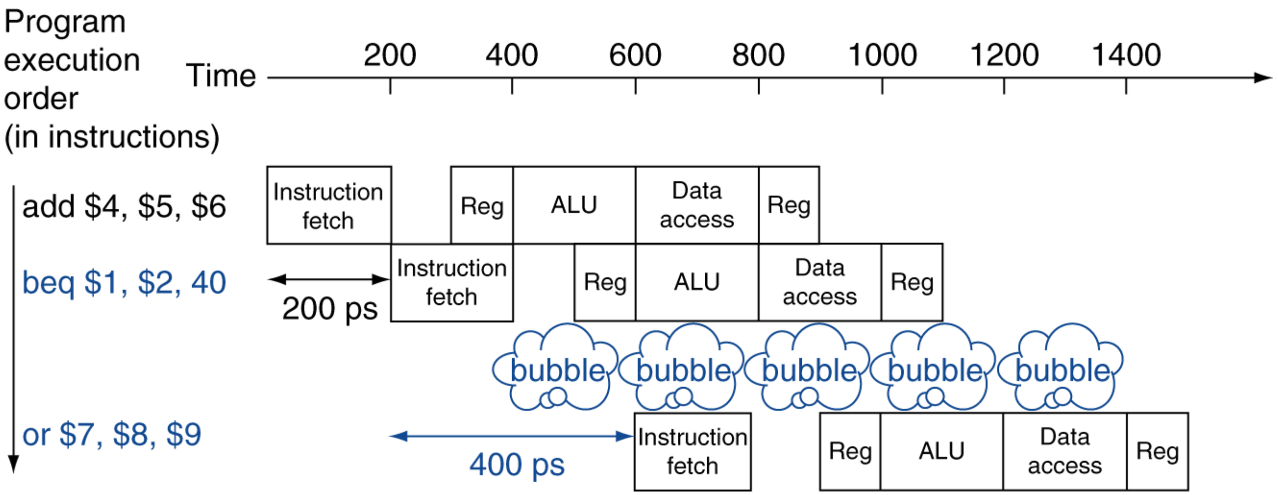
11 cycles

# Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - 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

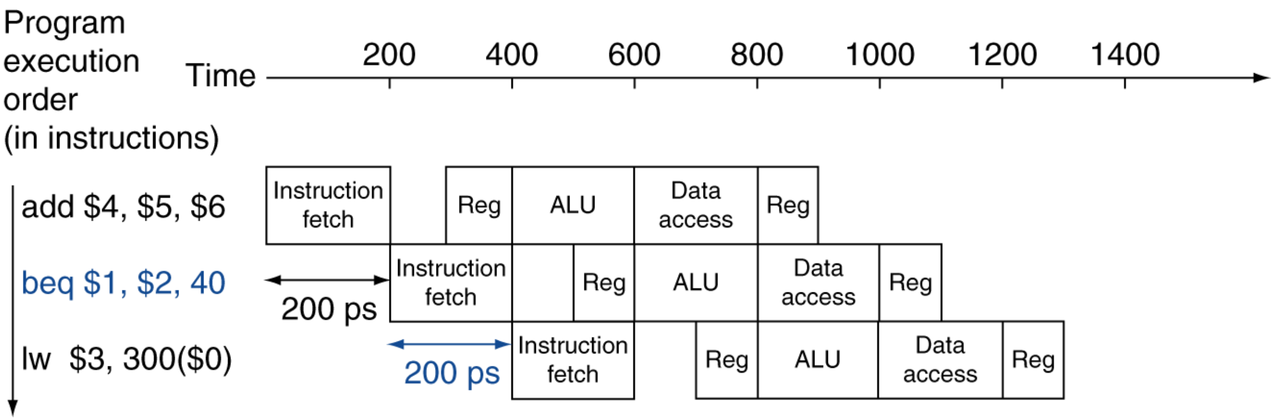


# Branch Prediction

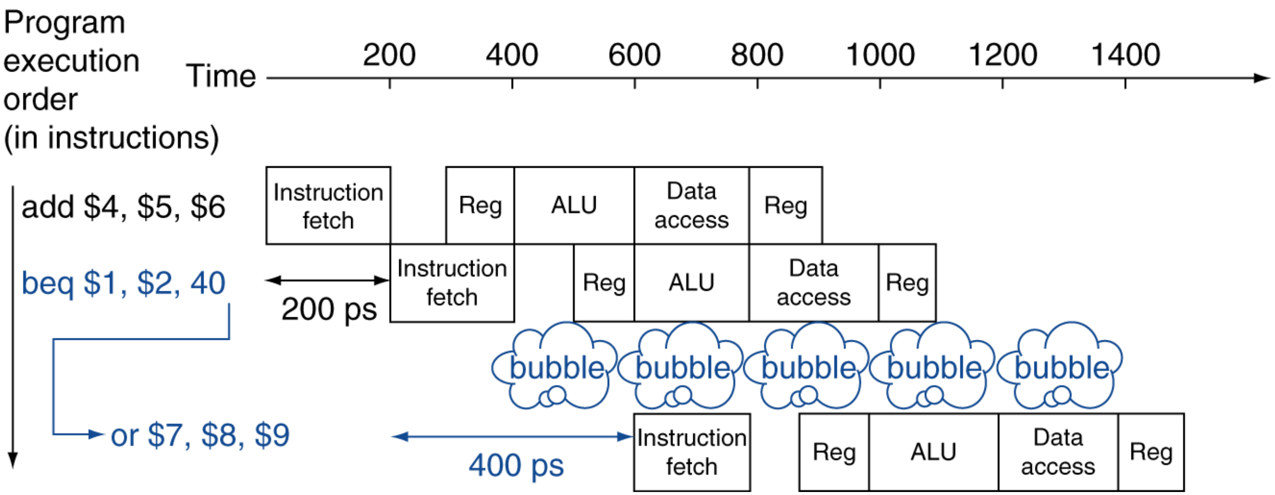
- Longer pipelines can’t readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In MIPS pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay

# MIPS with Predict Not Taken

Prediction correct



Prediction incorrect



# More-Realistic Branch Prediction

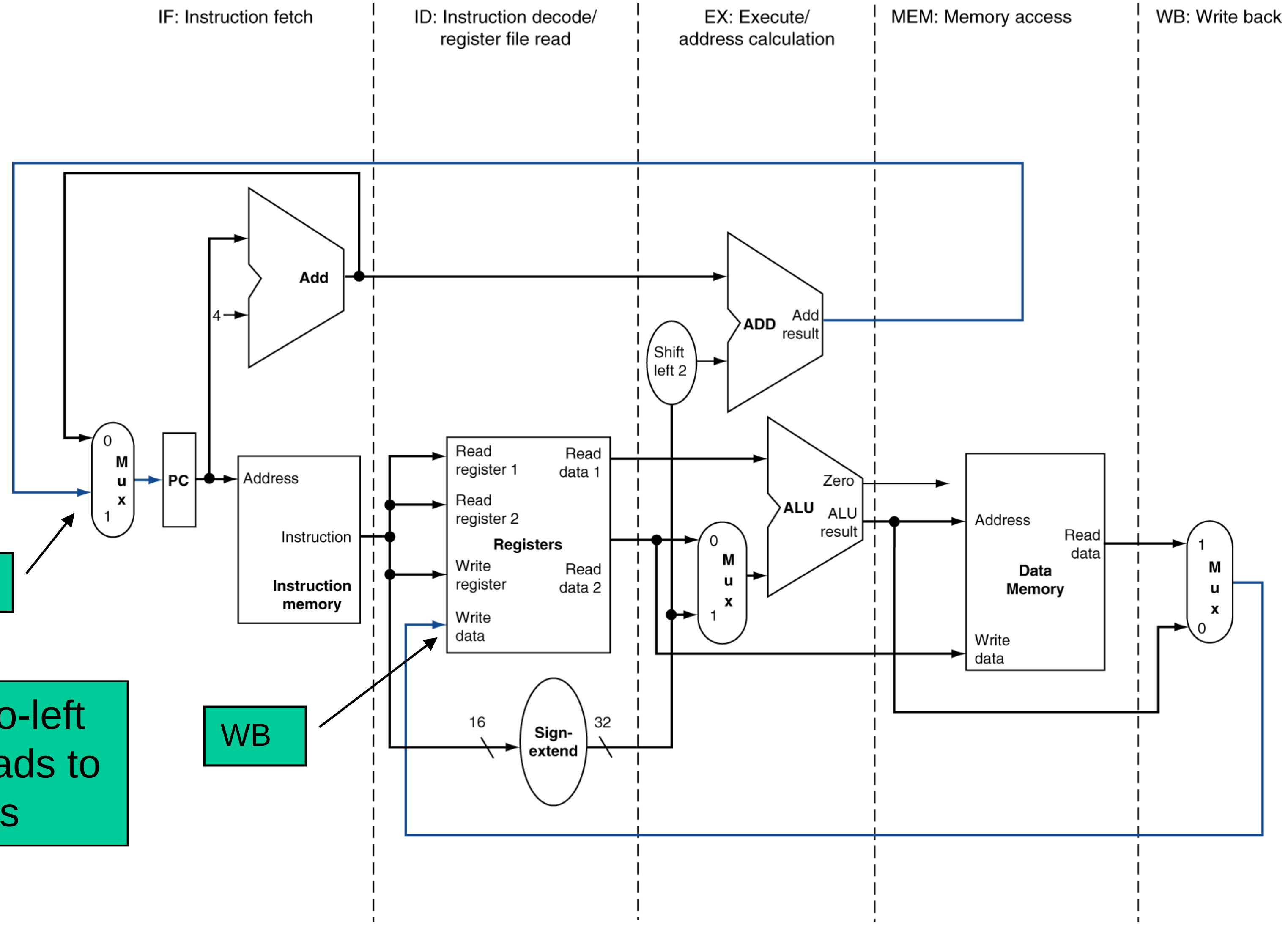
- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

# Pipeline Summary

**The BIG Picture**

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

# MIPS Pipelined Datapath



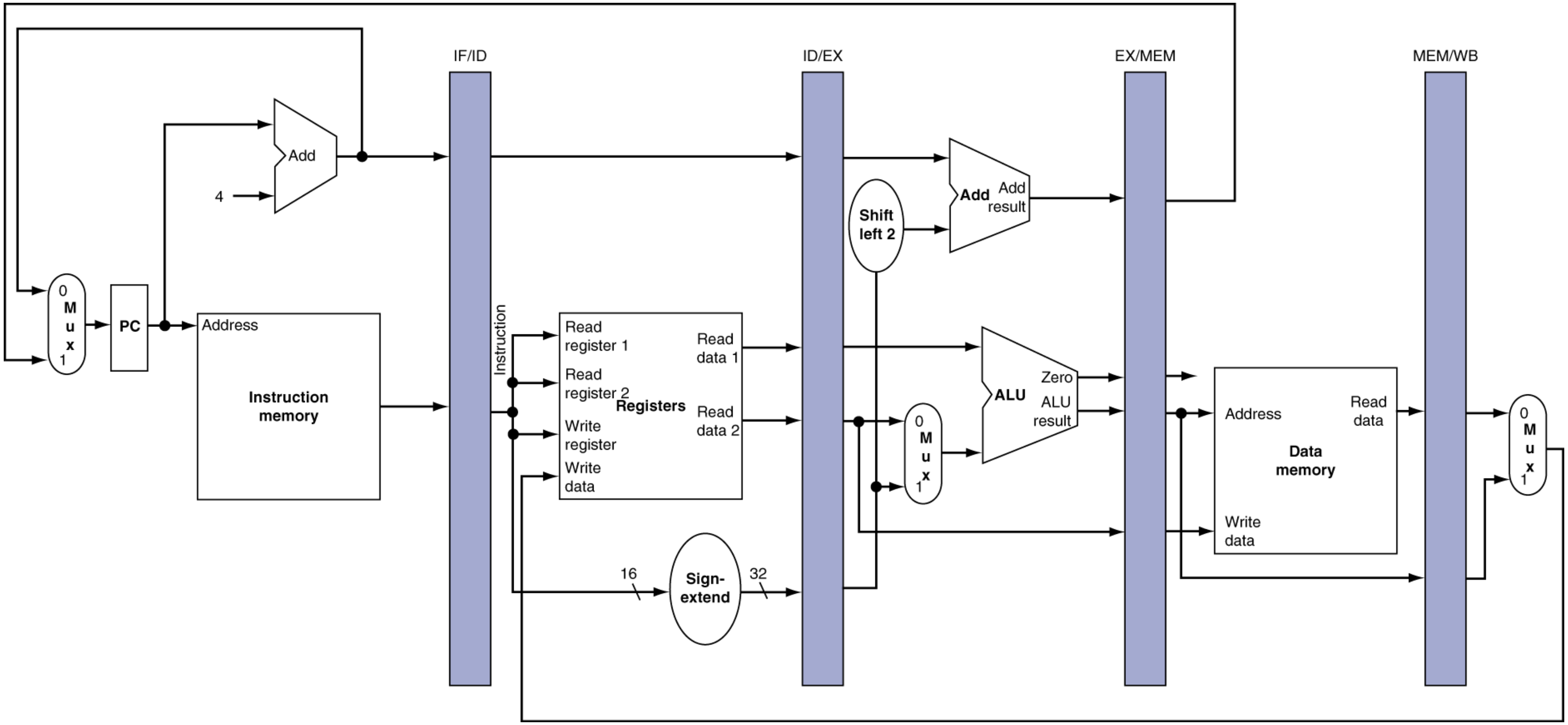
MEM

WB

Right-to-left flow leads to hazards

# Pipeline registers

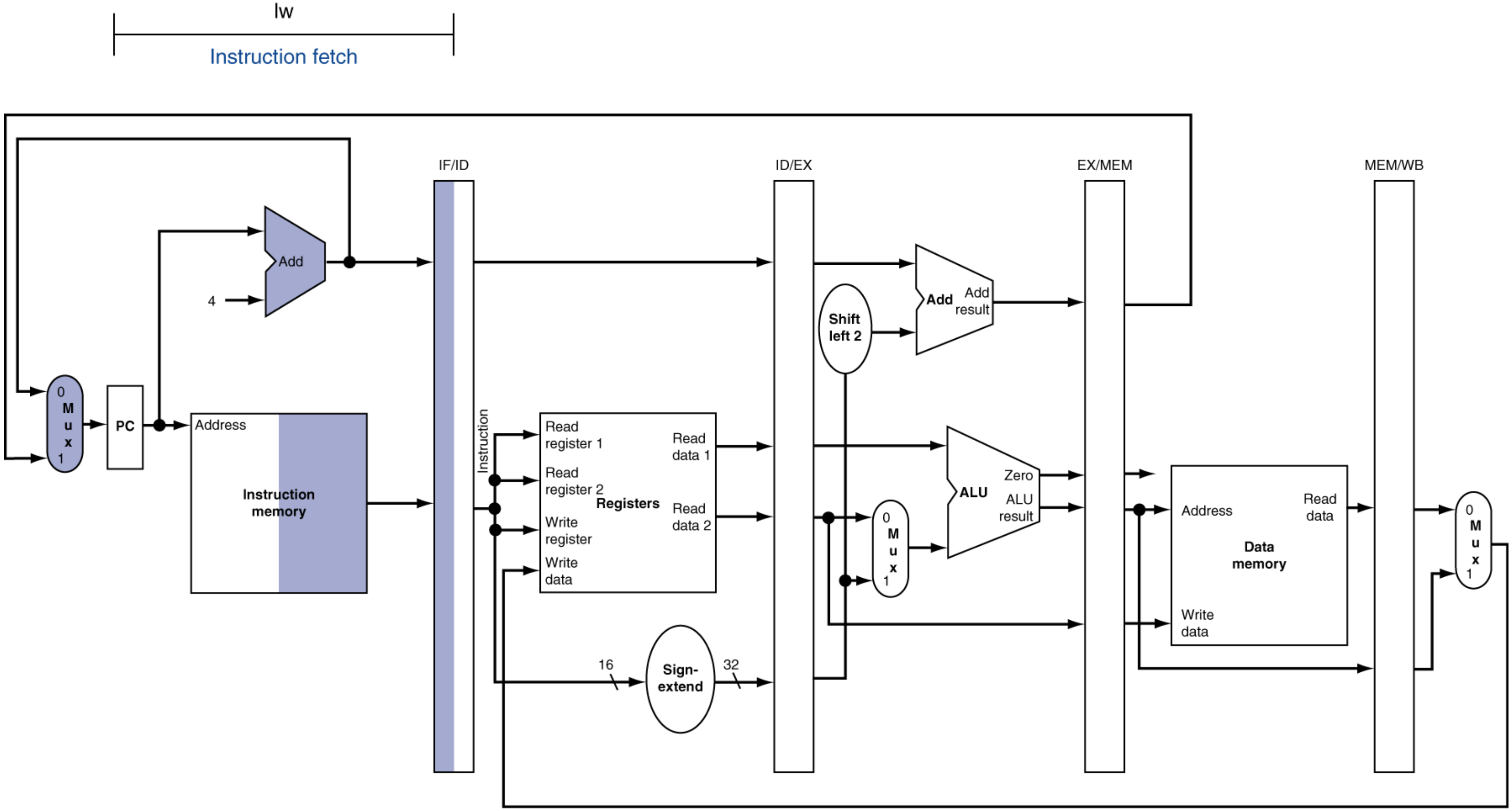
- Need registers between stages
  - To hold information produced in previous cycle



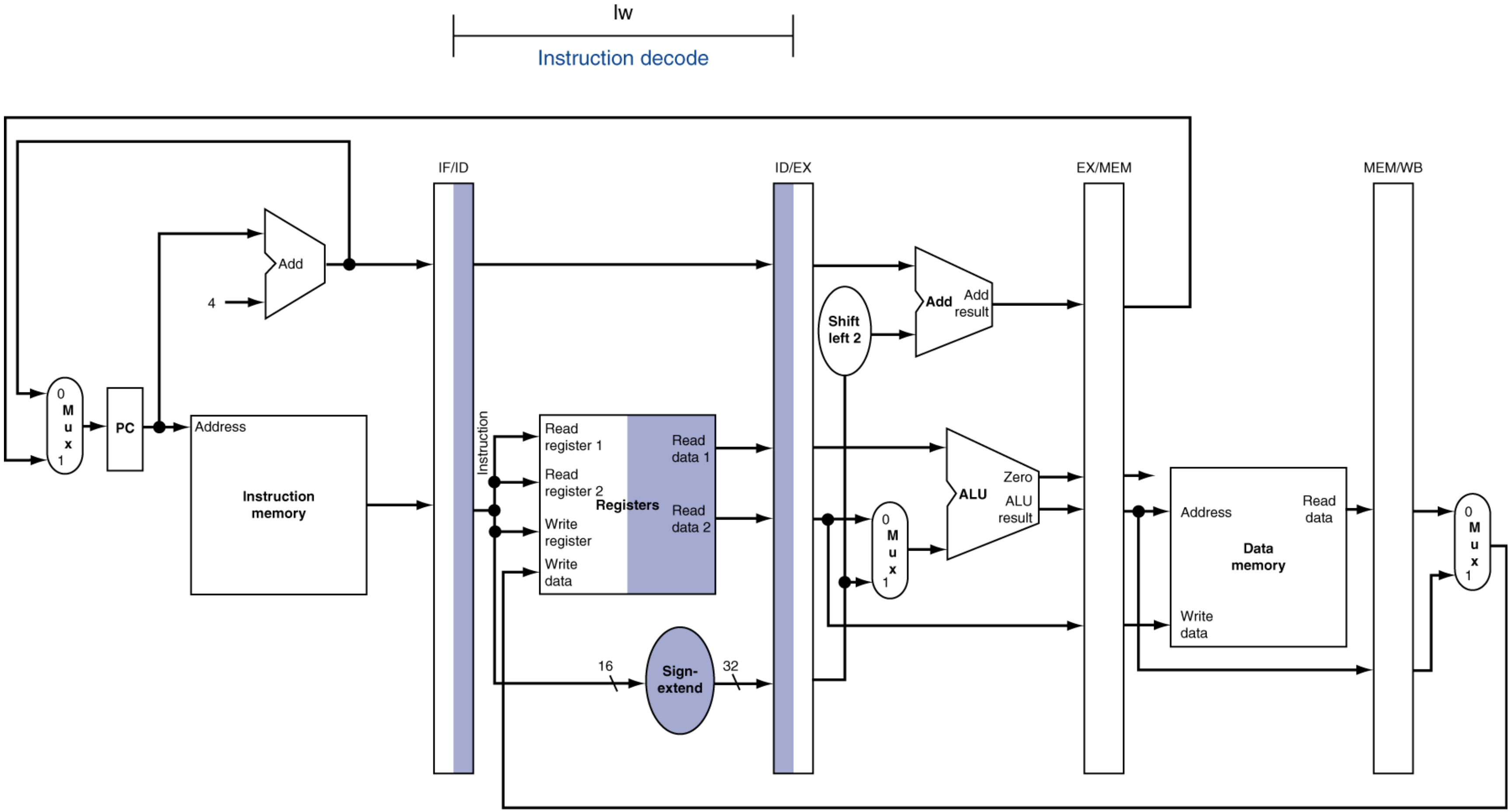
# Pipeline Operation

- Cycle-by-cycle flow of instructions through the pipelined datapath
  - "Single-clock-cycle" pipeline diagram
    - *Shows pipeline usage in a single cycle*
    - *Highlight resources used*
  - c.f. "multi-clock-cycle" diagram
    - *Graph of operation over time*
- We'll look at "single-clock-cycle" diagrams for load & store

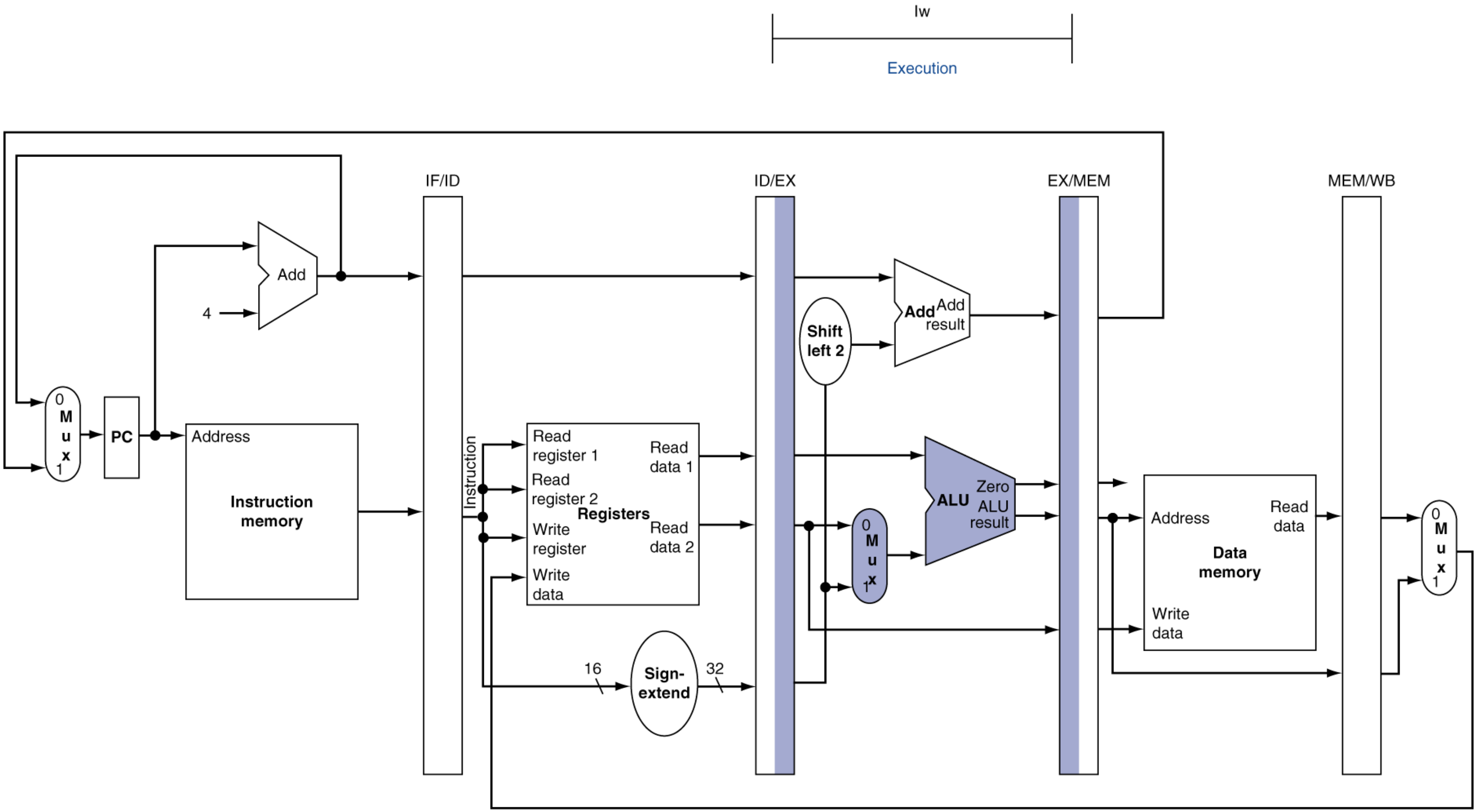
# IF for Load, Store, …



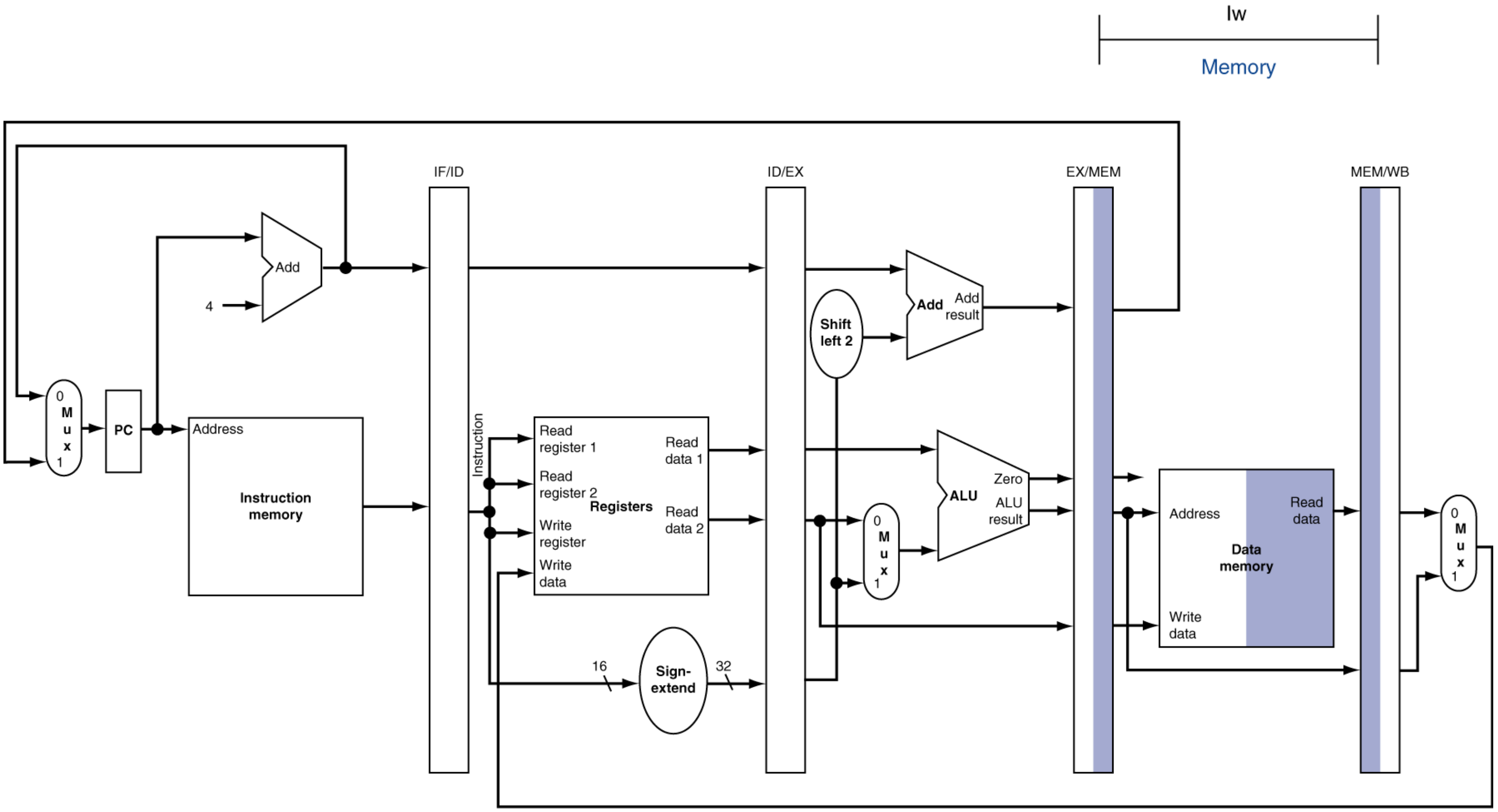
# ID for Load, Store, …



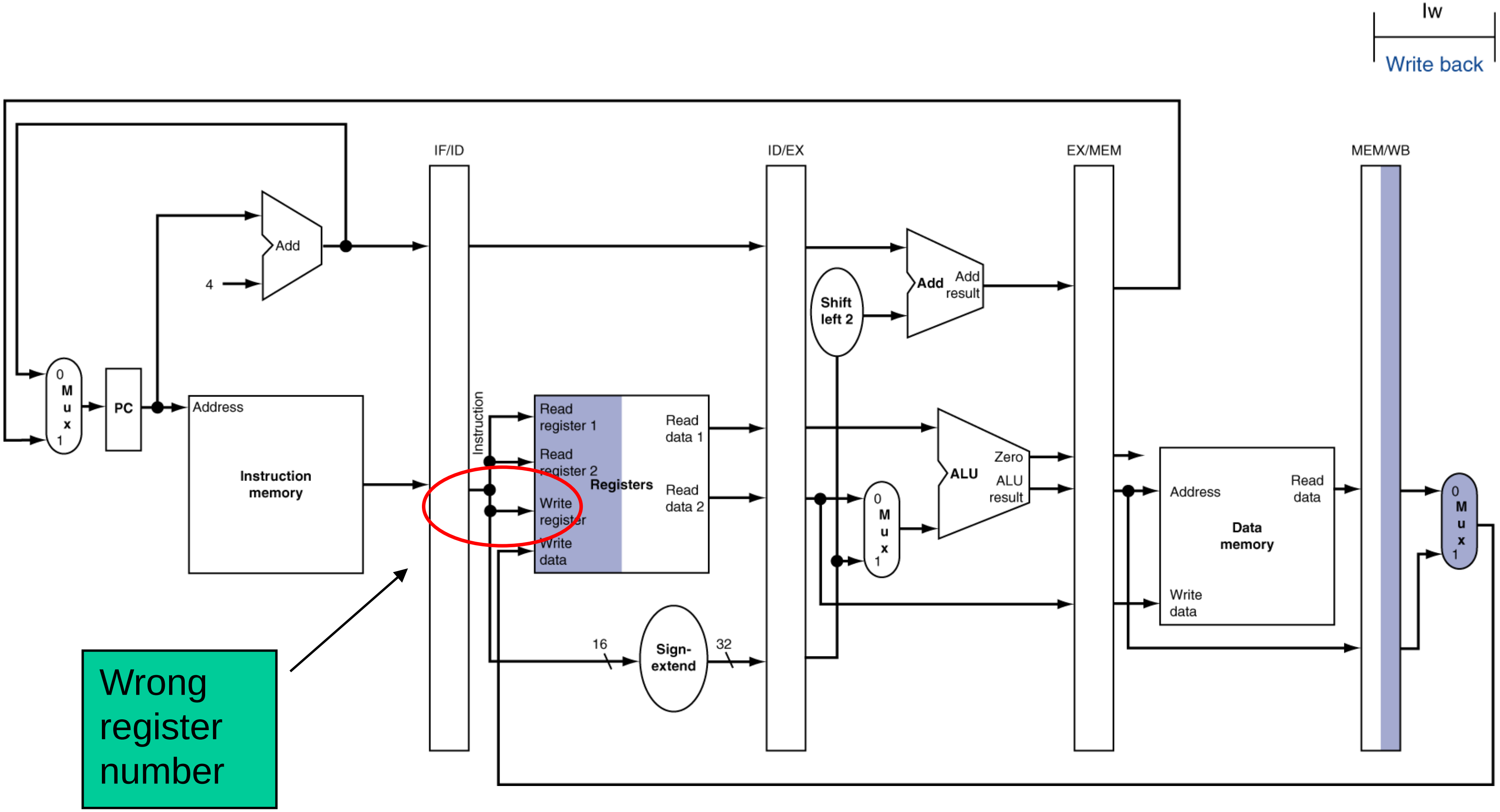
# EX for Load


lw
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
16
Sign-extend
32
Shift left 2
Add
Add result
0
M
u
x
1
ALU
Zero
ALU result
Address
Data memory
Write data
Read data
0
M
u
x
1

# MEM for Load

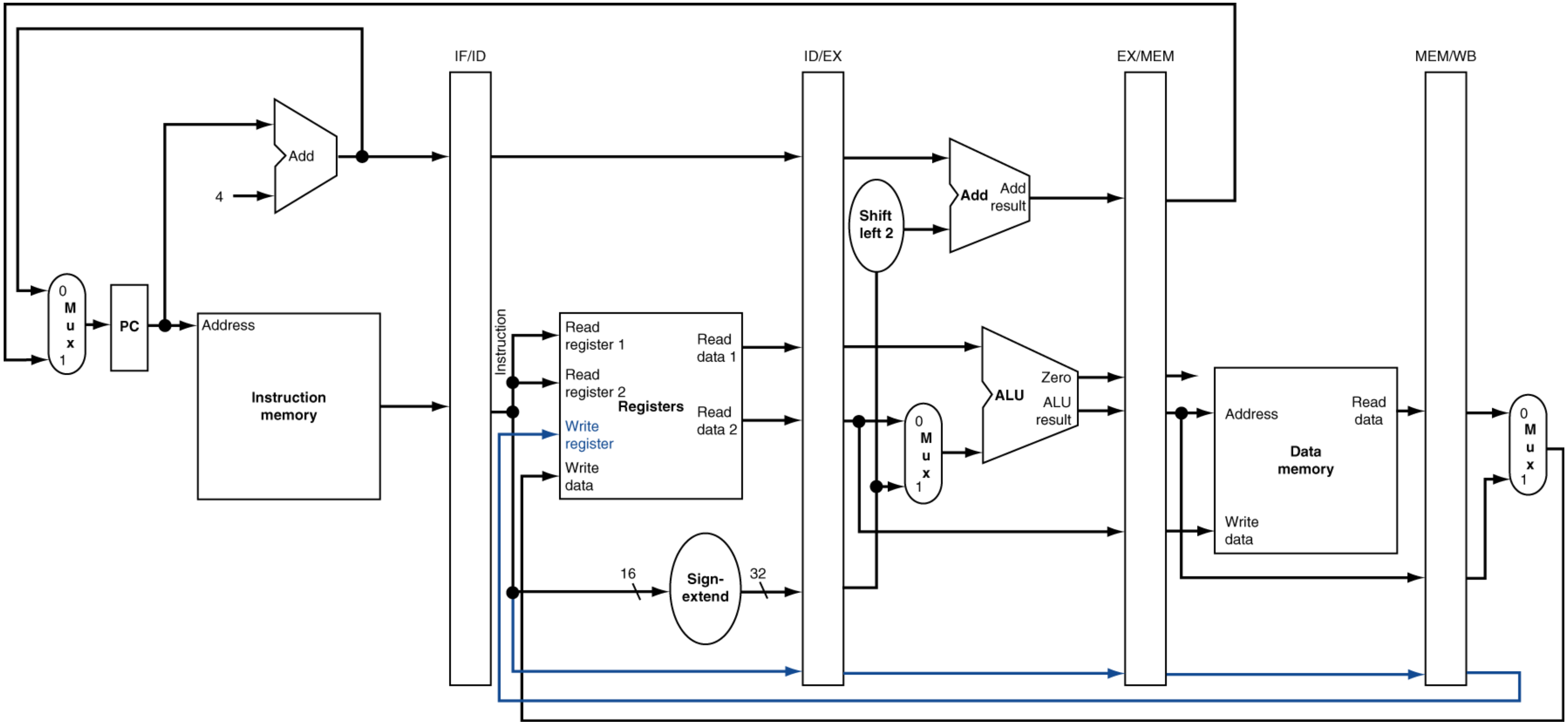

lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
0
M
u
x
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data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

# WB for Load

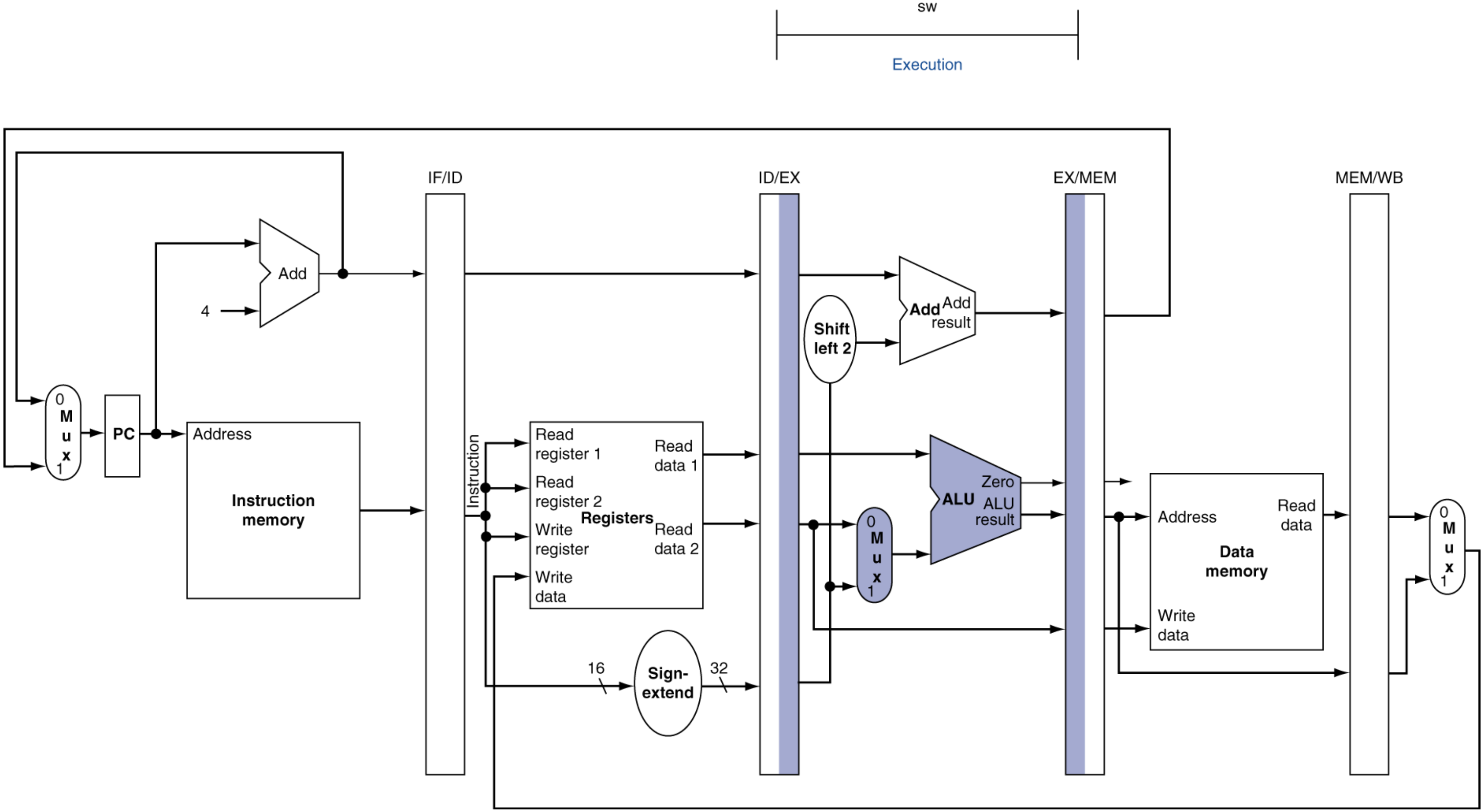


Wrong register number

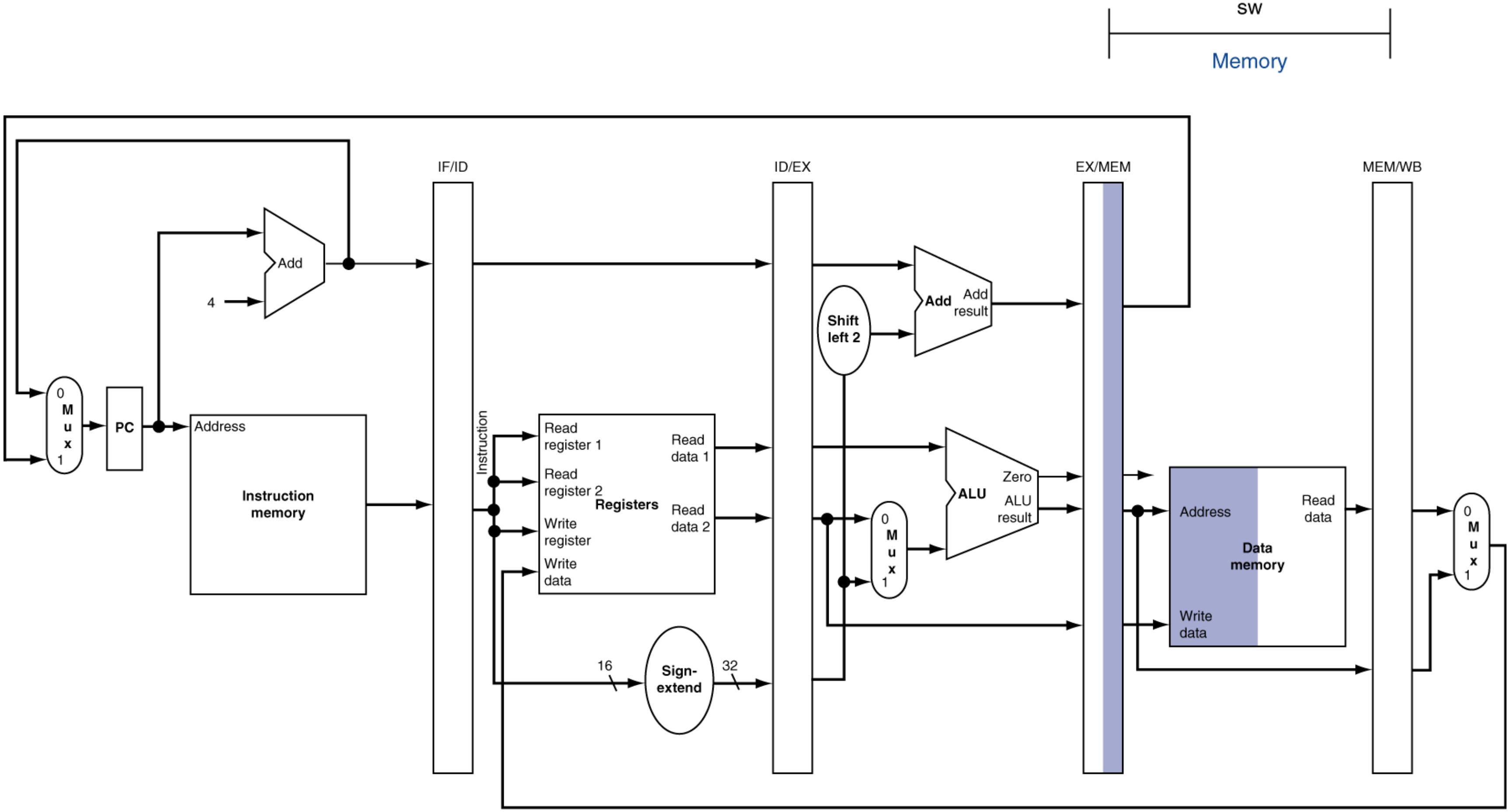
# Corrected Datapath for Load



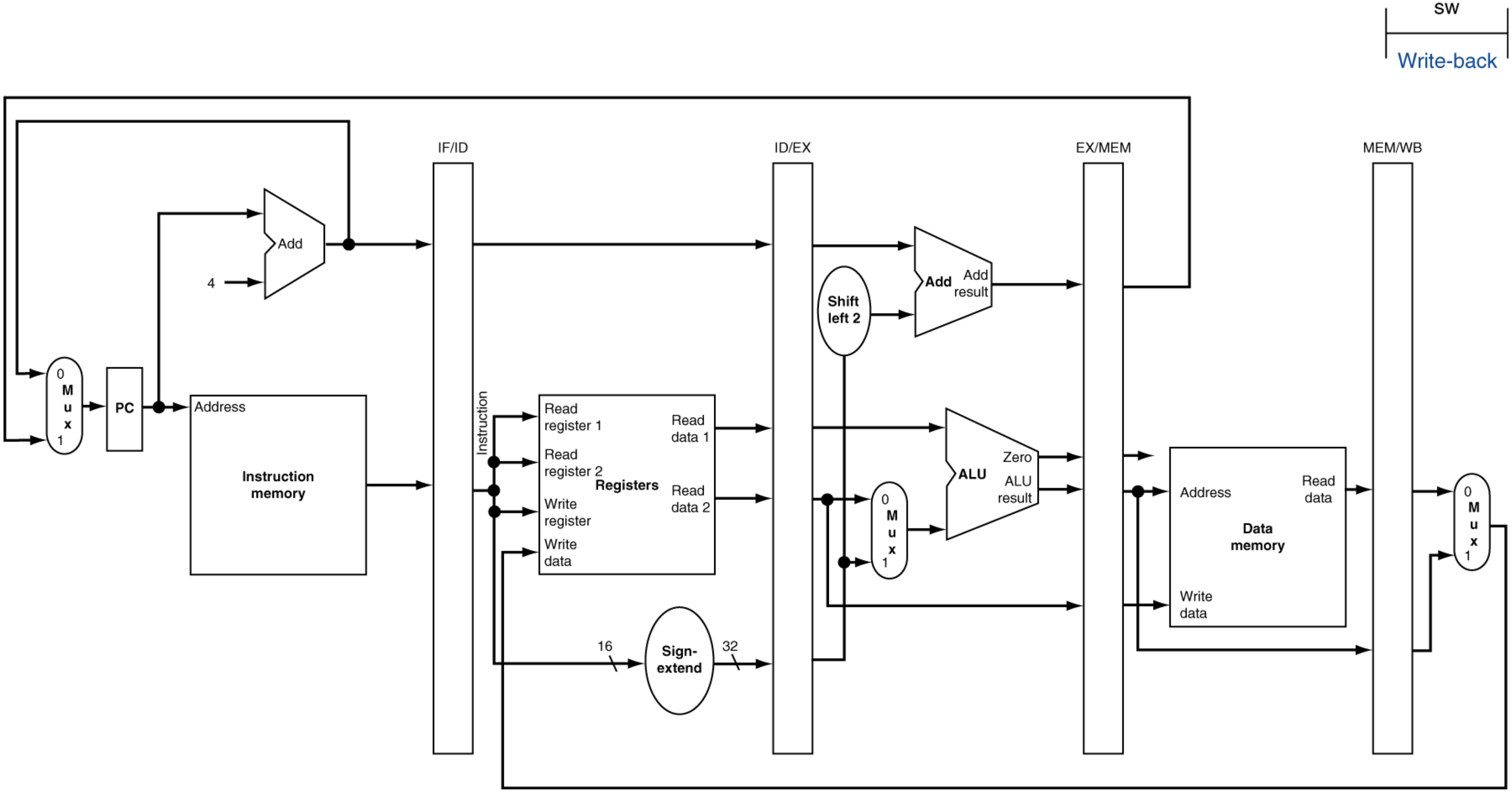
# EX for Store


sw
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
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
0
M
u
x
1
ALU
Zero
ALU result
Address
Data memory
Write data
Read data
0
M
u
x
1

# MEM for Store

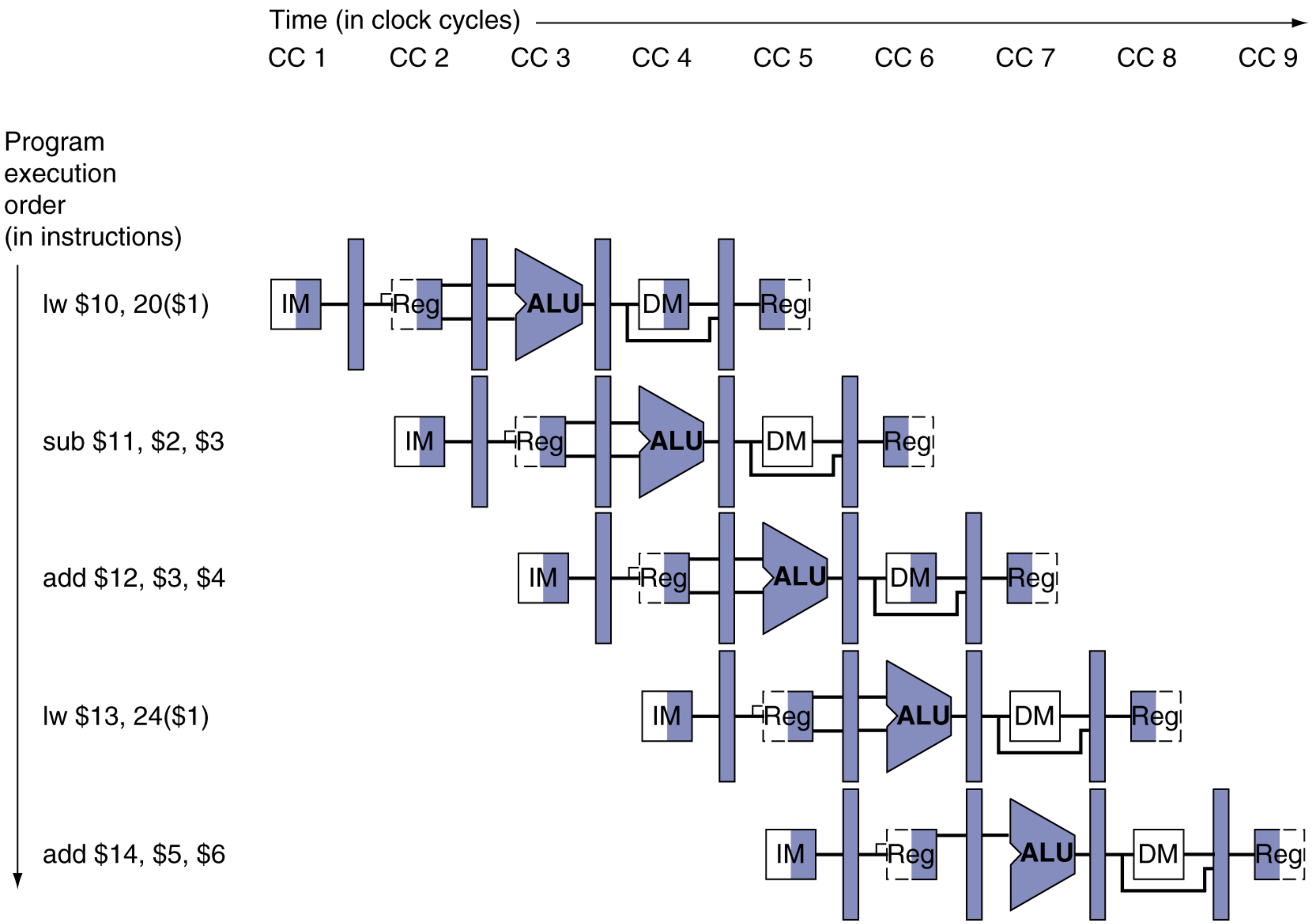


# WB for Store


sw
Write-back
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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
0
M
u
x
1
Address
Read data
Data memory
Write data
0
M
u
x
1

# Multi-Cycle Pipeline Diagram

- Form showing resource usage



Time (in clock cycles)

CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9

Program execution order (in instructions)

lw $10, 20($1)

sub $11, $2, $3

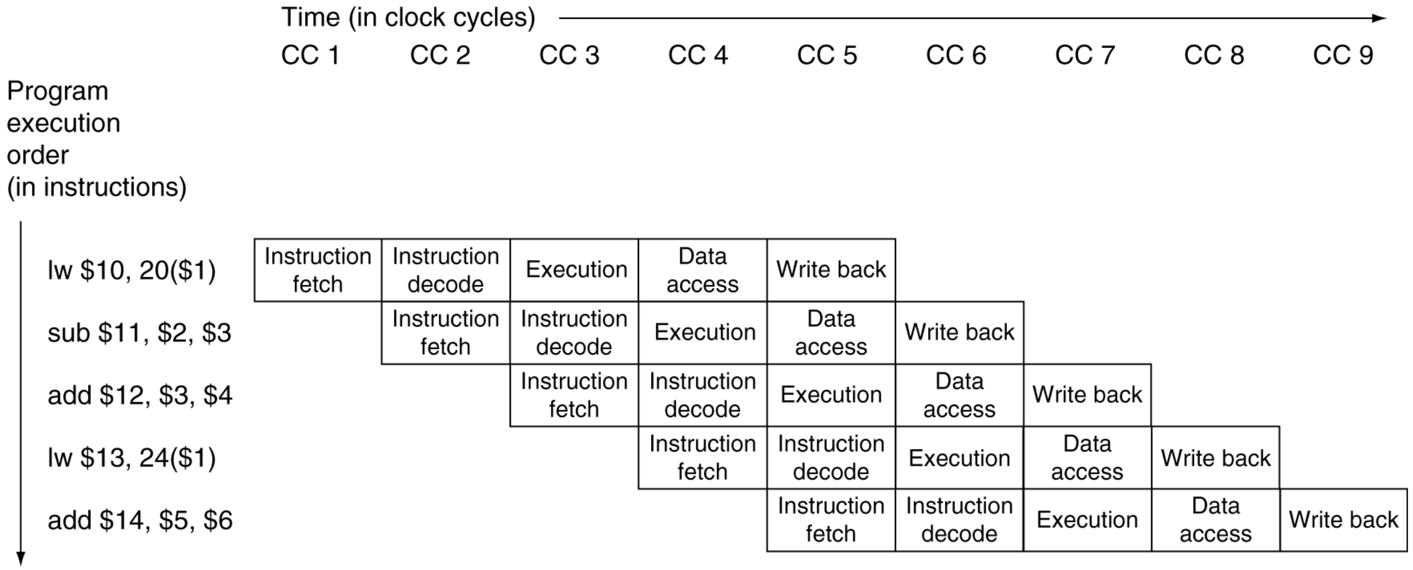
add $12, $3, $4

lw $13, 24($1)

add $14, $5, $6

# Multi-Cycle Pipeline Diagram

- Traditional form



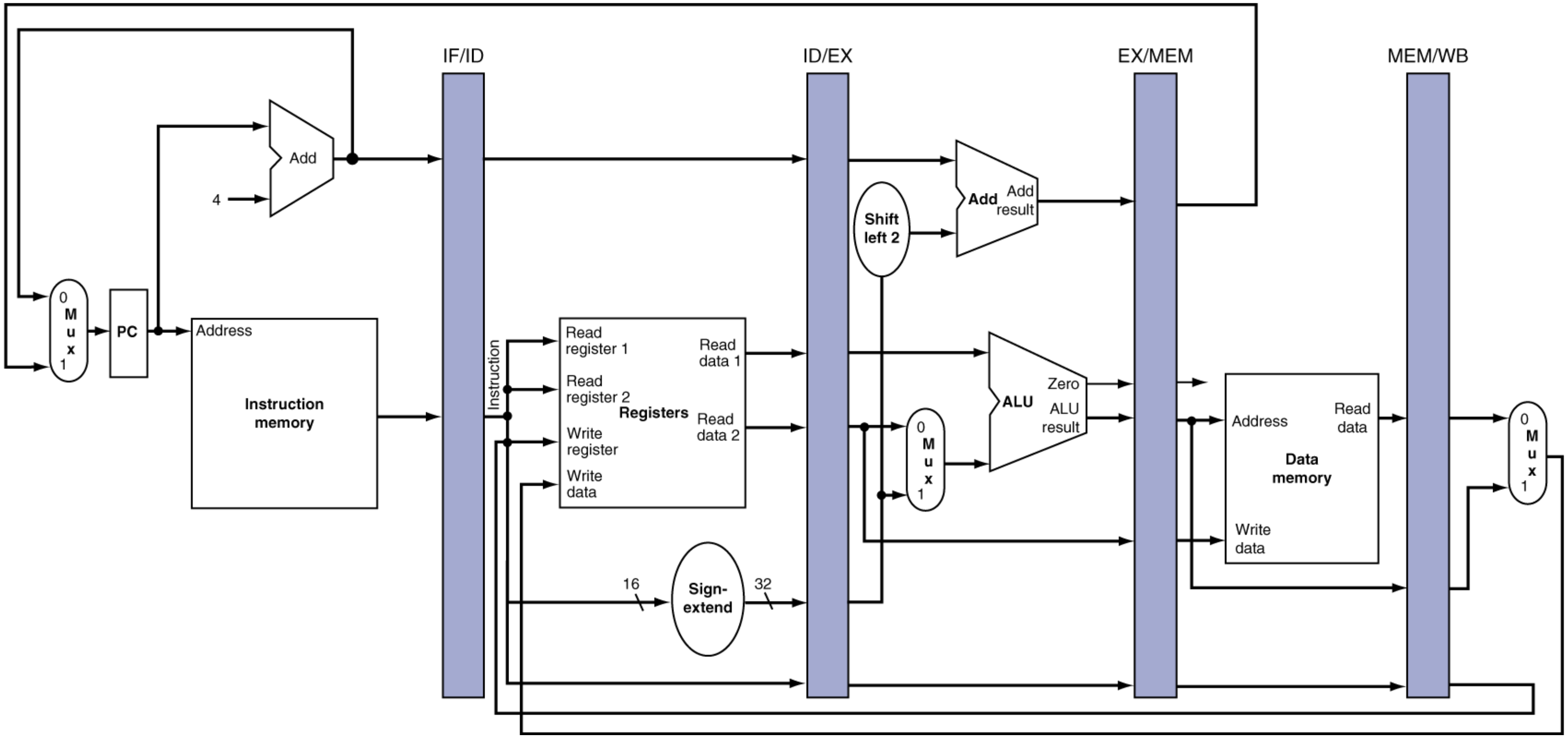
Time (in clock cycles)

| Program execution order (in instructions) | CC 1 | CC 2 | CC 3 | CC 4 | CC 5 | CC 6 | CC 7 | CC 8 | CC 9 |
|---|---|---|---|---|---|---|---|---|---|
| lw $10, 20($1) | Instruction fetch | Instruction decode | Execution | Data access | Write back | | | | |
| sub $11, $2, $3 | | Instruction fetch | Instruction decode | Execution | Data access | Write back | | | |
| add $12, $3, $4 | | | Instruction fetch | Instruction decode | Execution | Data access | Write back | | |
| lw $13, 24($1) | | | | Instruction fetch | Instruction decode | Execution | Data access | Write back | |
| add $14, $5, $6 | | | | | Instruction fetch | Instruction decode | Execution | Data access | Write back |

# Single-Cycle Pipeline Diagram

- 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 |


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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
0
M
u
x
1
Address
Data memory
Write data
Read data
0
M
u
x
1