CPT_S 260 Intro to Computer Architecture
Lecture 35

Pipeline Datapath
April 11, 2021

Ganapati Bhat
School of Electrical Engineering and Computer Science
Washington State University

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

Control Hazard on Branches Three Stage Stall

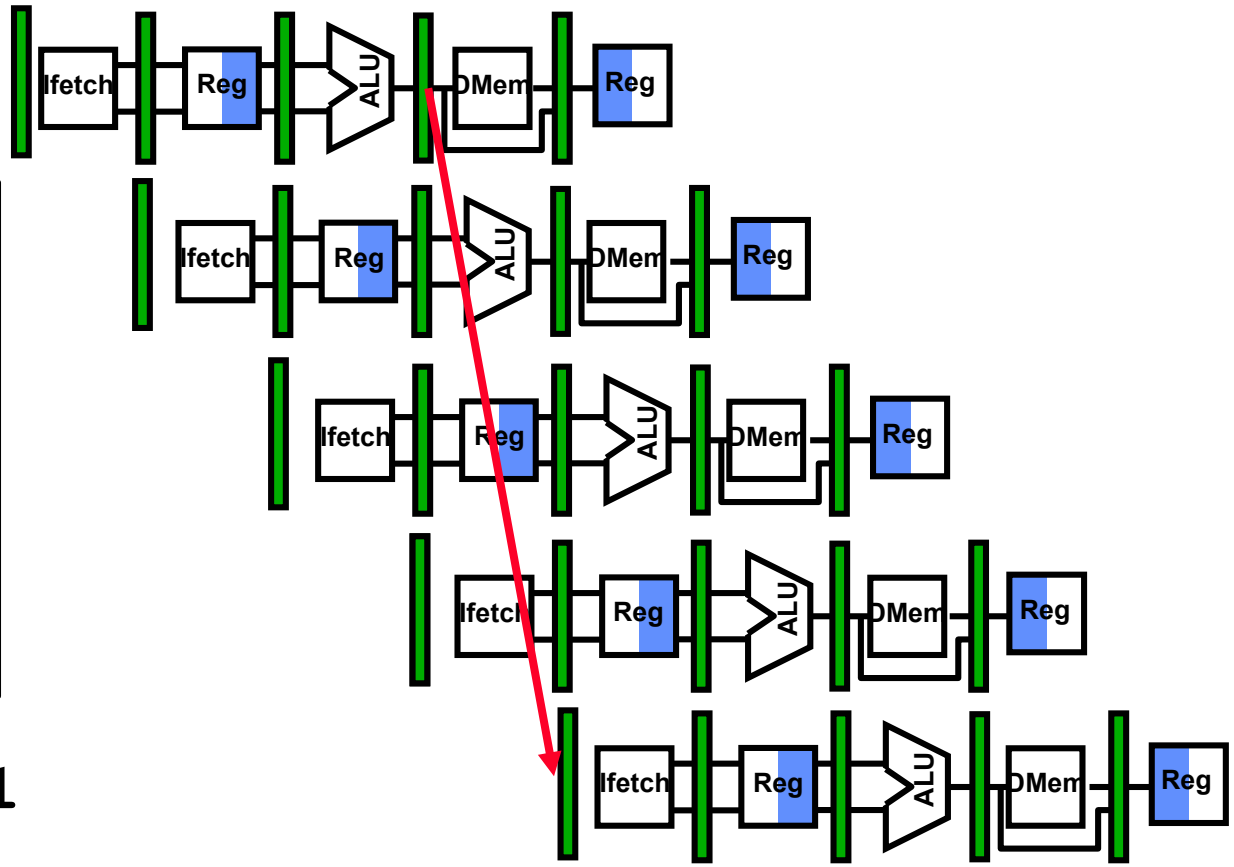
```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```



What do you do with the 3 instructions in between?

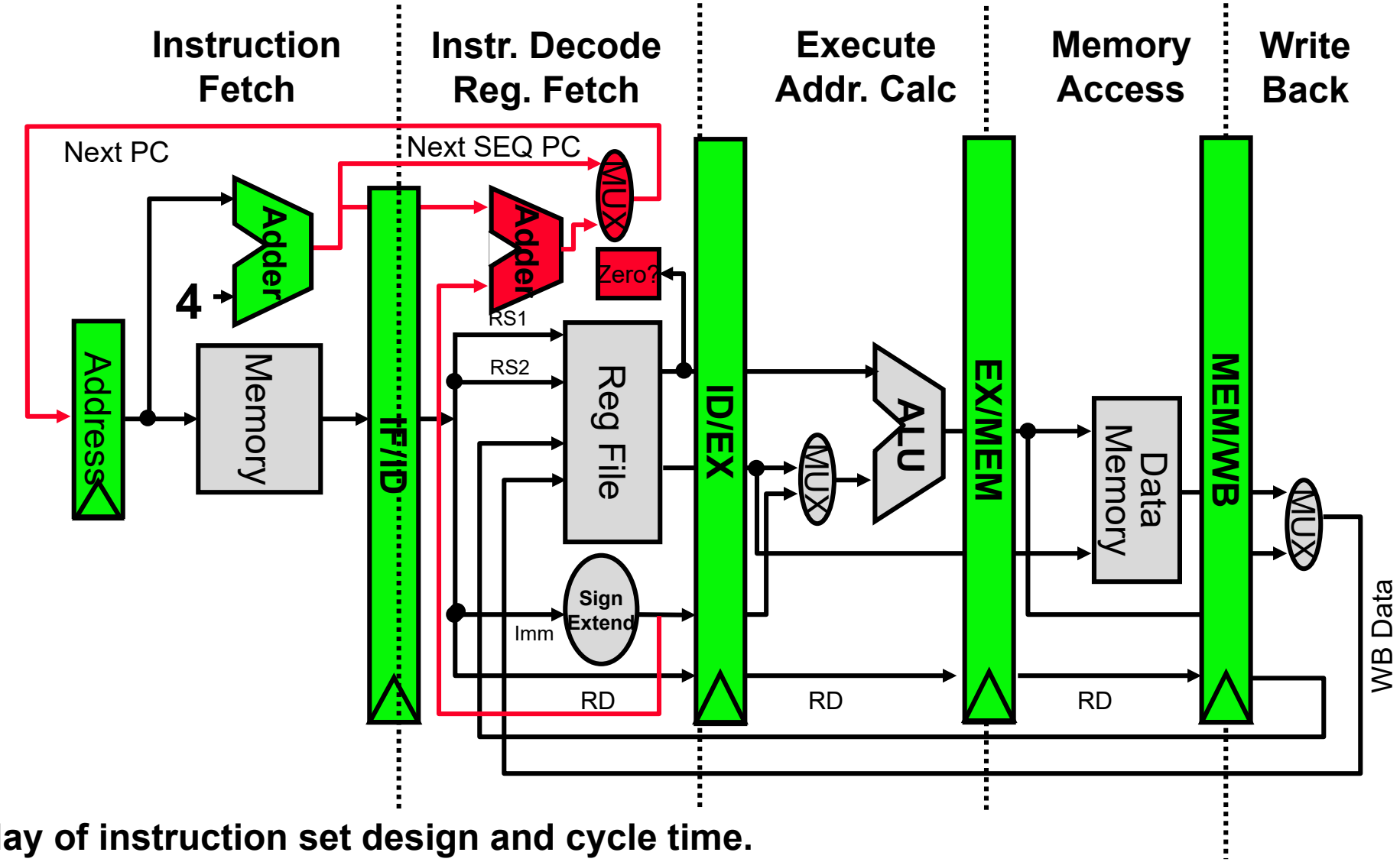
How do you do it?

Where is the “commit”?

Branch Stall Impact

- **If CPI = 1, 30% branch,**
  Stall 3 cycles => new CPI = 1.9!
- **Two-part solution:**
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- **MIPS branch tests if register = 0 or ≠ 0**
- **MIPS Solution:**
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Pipelined MIPS Datapath



- Interplay of instruction set design and cycle time.

Four Branch Hazard Alternatives

1: Stall until branch direction is clear

2: Predict Branch Not Taken

- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% MIPS branches not taken on average
- PC+4 already calculated, so use it to get next instruction

3: Predict Branch Taken

- 53% MIPS branches taken on average
- But haven't calculated branch target address in MIPS
  - » MIPS still incurs 1 cycle branch penalty
  - » Other machines: branch target known before outcome
- What happens when hit not-taken branch?

Four Branch Hazard Alternatives

4: Delayed Branch

– Define branch to take place AFTER a following instruction

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

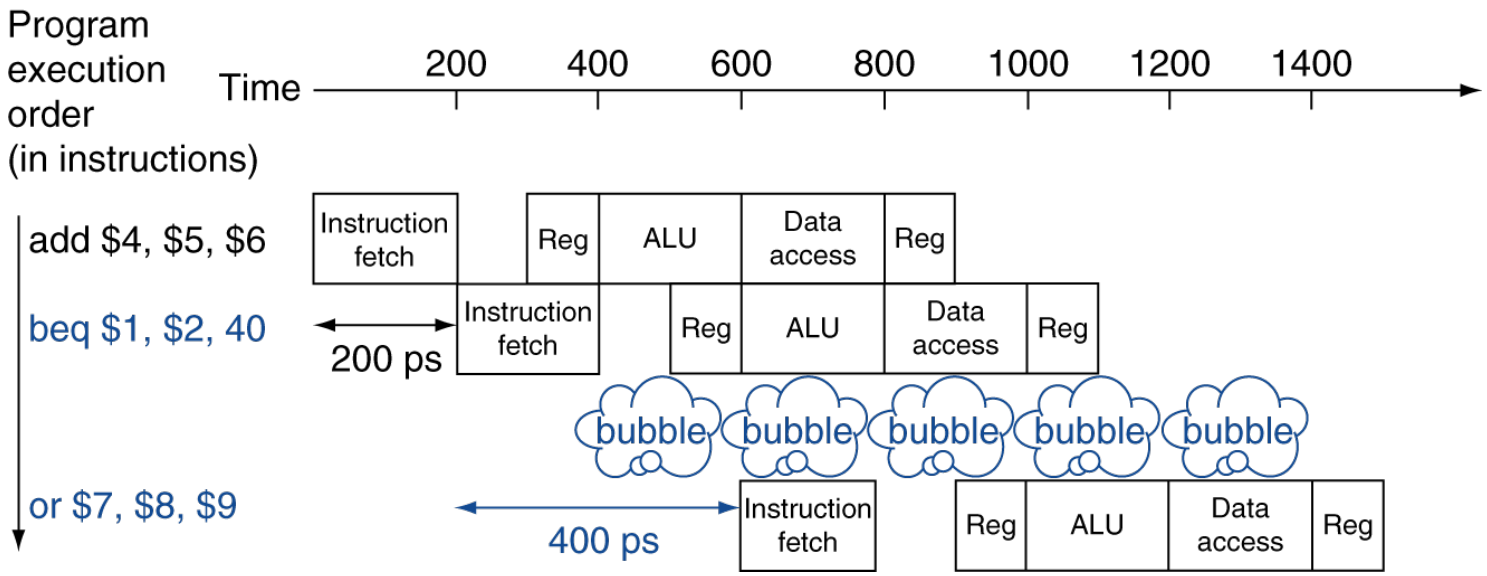
Branch delay of length *n*

– 1 slot delay allows proper decision and branch target address in 5 stage pipeline

– MIPS uses this

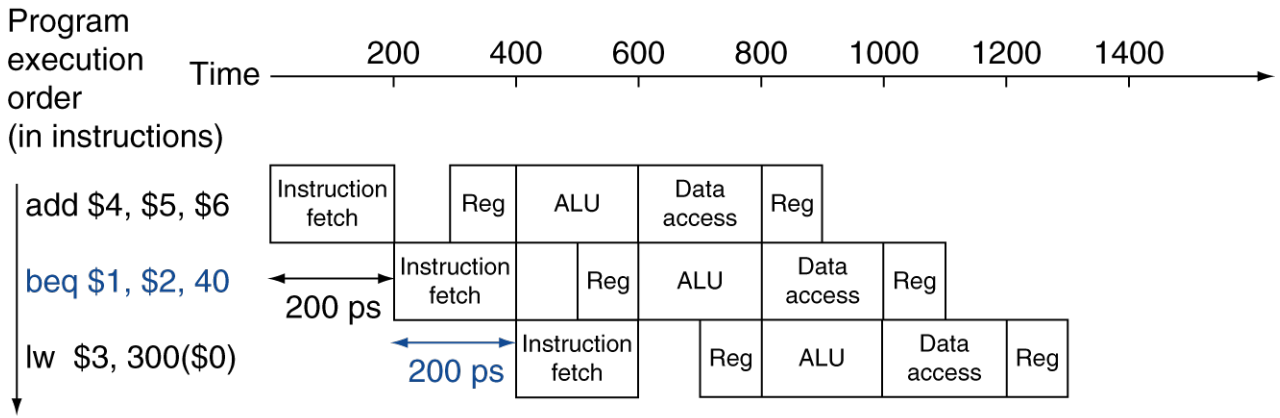
Stall on Branch

- **Wait until branch outcome determined before fetching next instruction**

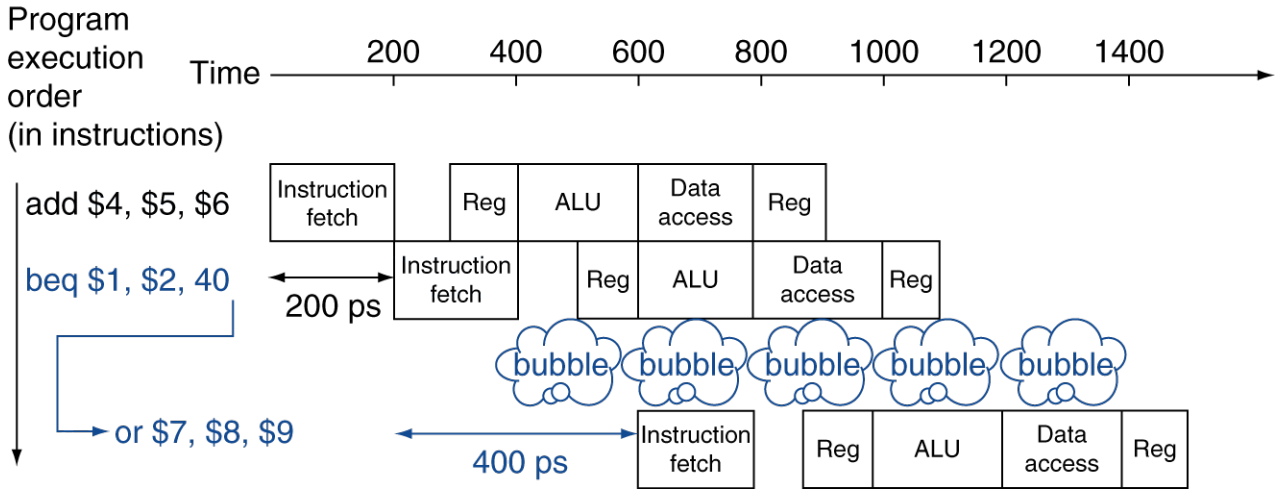


MIPS with Predict Not Taken

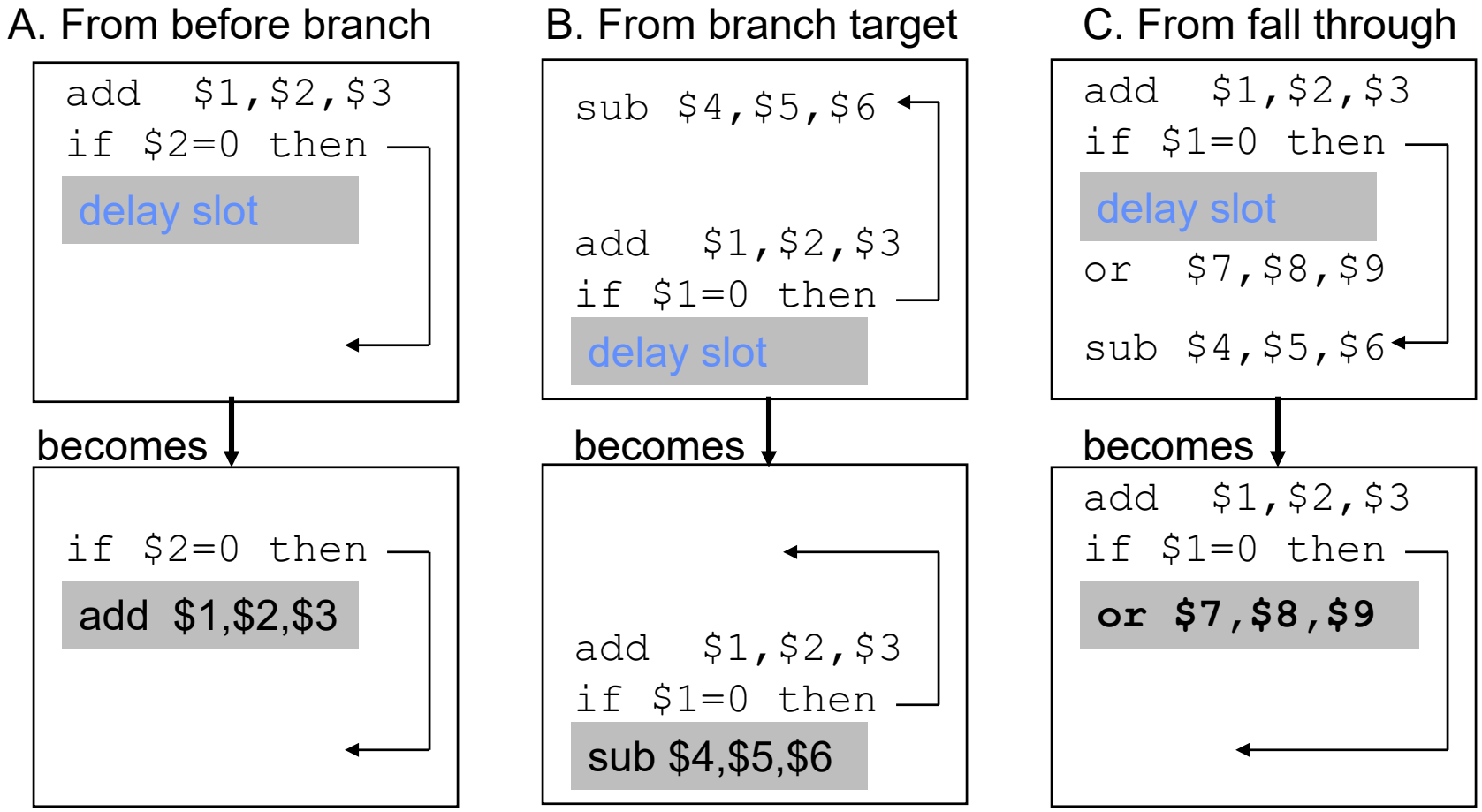
Prediction correct



Prediction incorrect



Scheduling Branch Delay Slots



- **A is the best choice, fills delay slot & reduces instruction count (IC)**
- **In B, the `sub` instruction may need to be copied, increasing IC**
- **In B and C, must be okay to execute `sub`/`or` when branch fails**

More-Realistic Branch Prediction

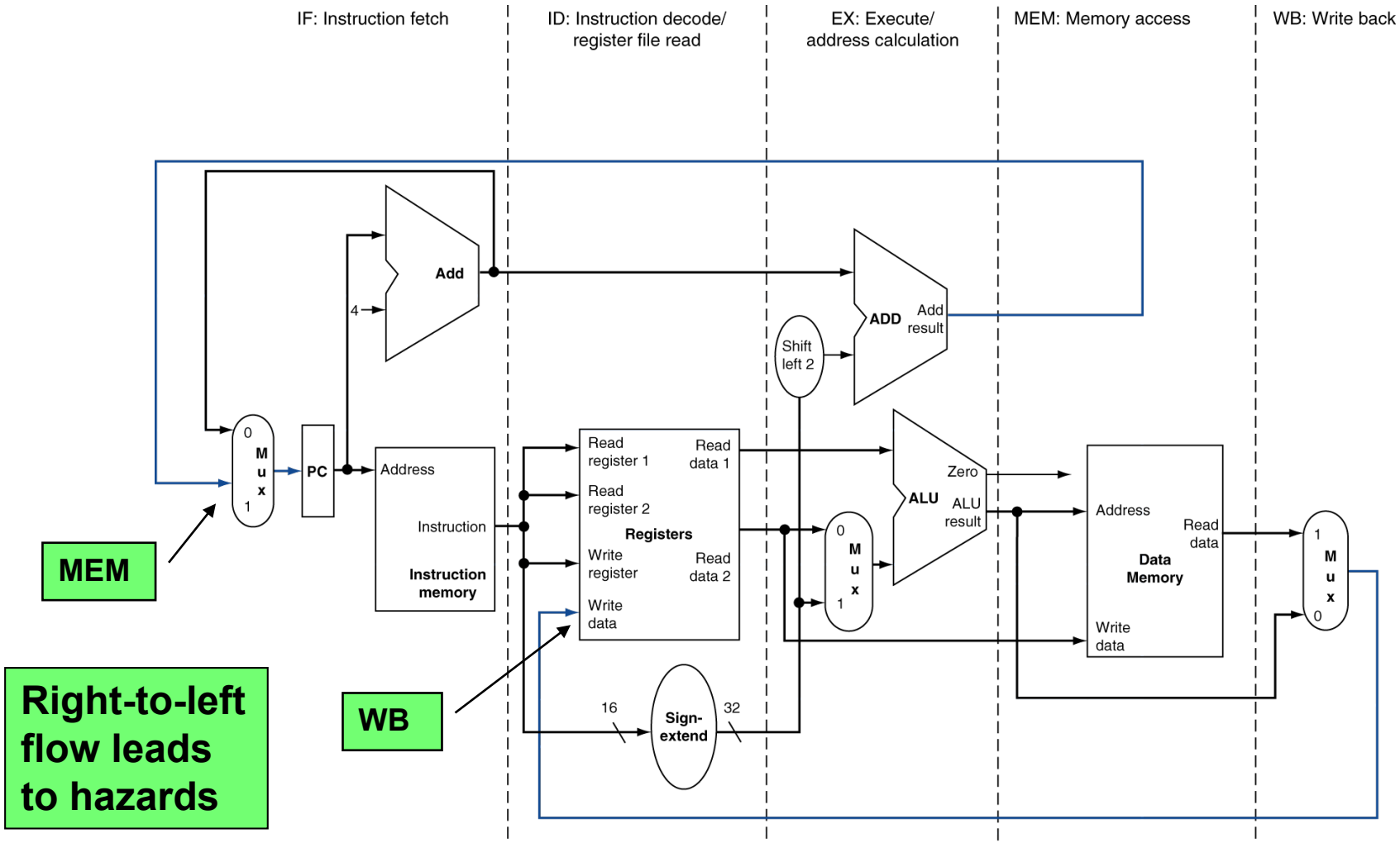
- **Static branch prediction**
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - » Predict backward branches taken
    - » Predict forward branches not taken
- **Dynamic branch prediction**
  - Hardware measures actual branch behavior
    - » e.g., record recent history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

Delayed Branch

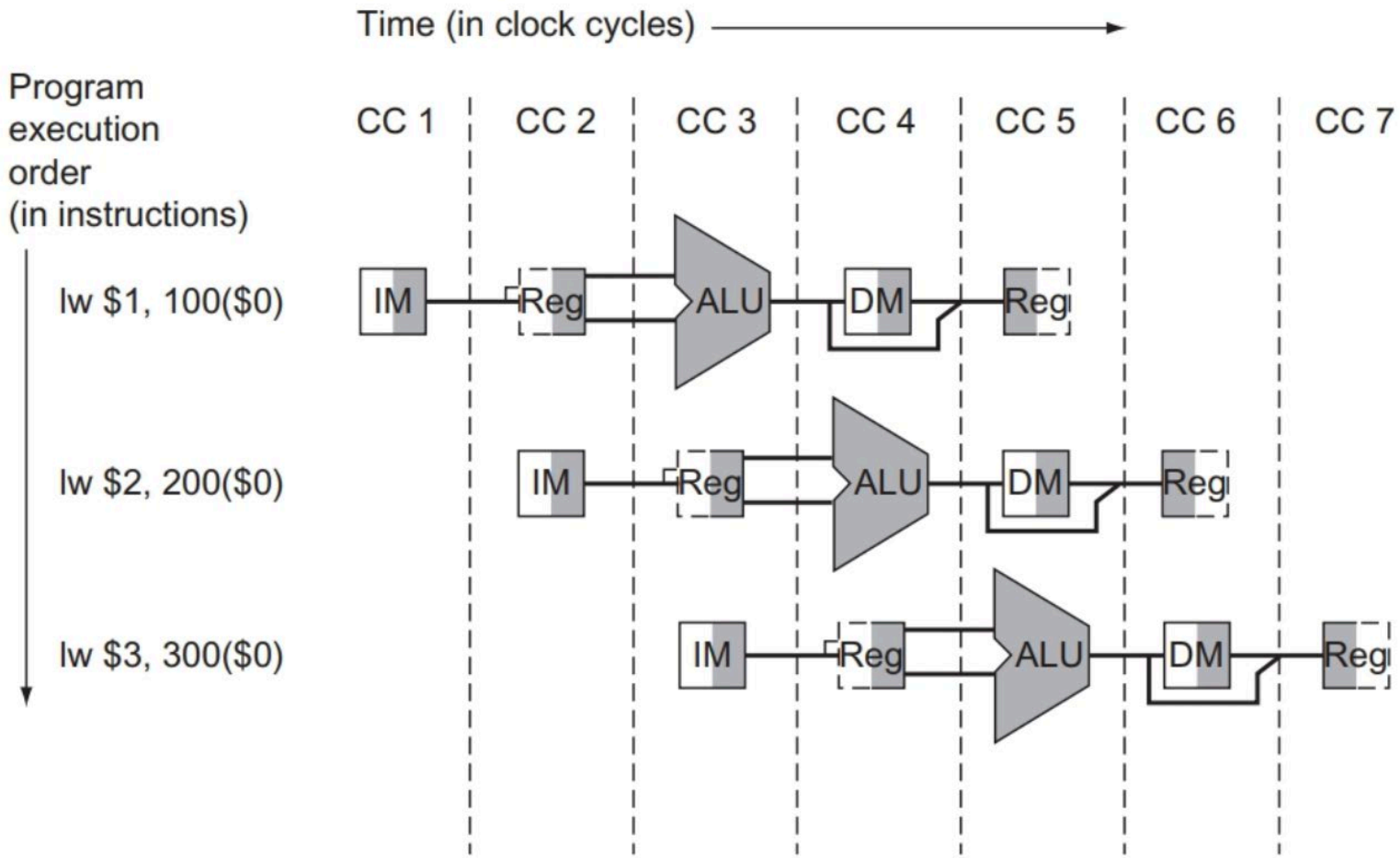
- **Compiler effectiveness for single branch delay slot:**
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled
- **Delayed Branch downside: As processor go to deeper pipelines and multiple issue, the branch delay grows and need more than one delay slot**
  - Delayed branching has lost popularity compared to more expensive but more flexible dynamic approaches
  - Growth in available transistors has made dynamic approaches relatively cheaper

Pipeline Datapath

MIPS Pipelined Datapath

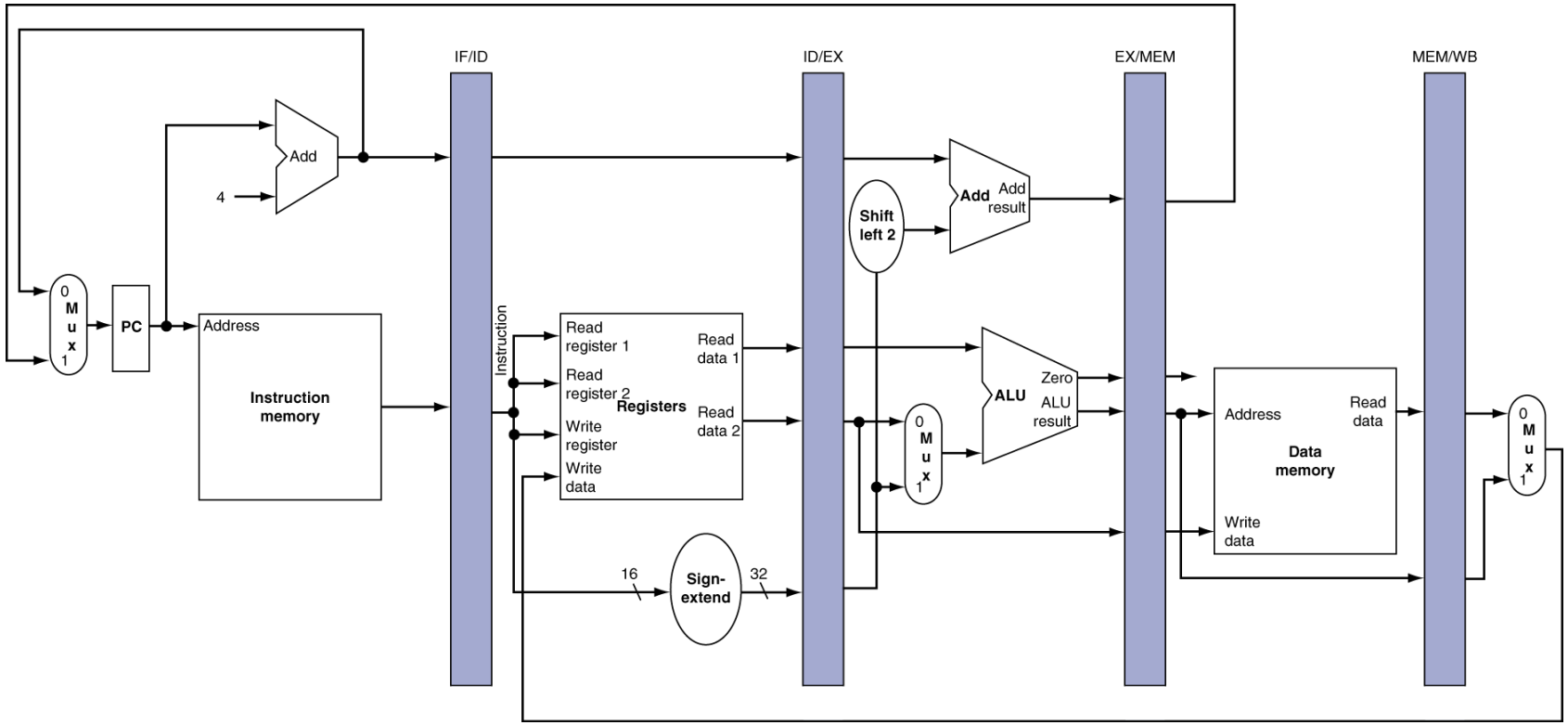


MIPS Pipelined Datapath



Pipeline registers

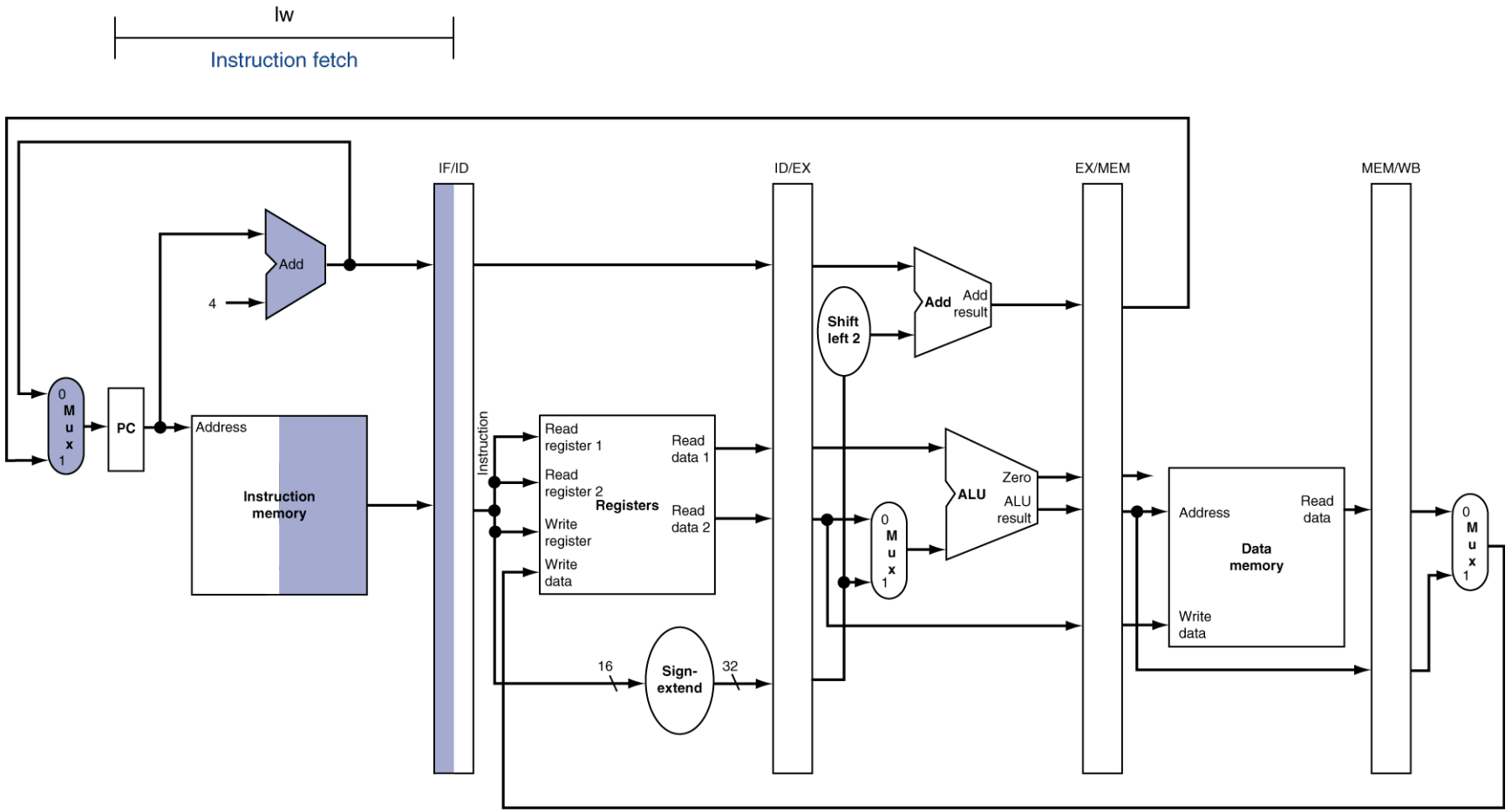
- **Need registers between stages**
  - To hold information produced in previous cycle



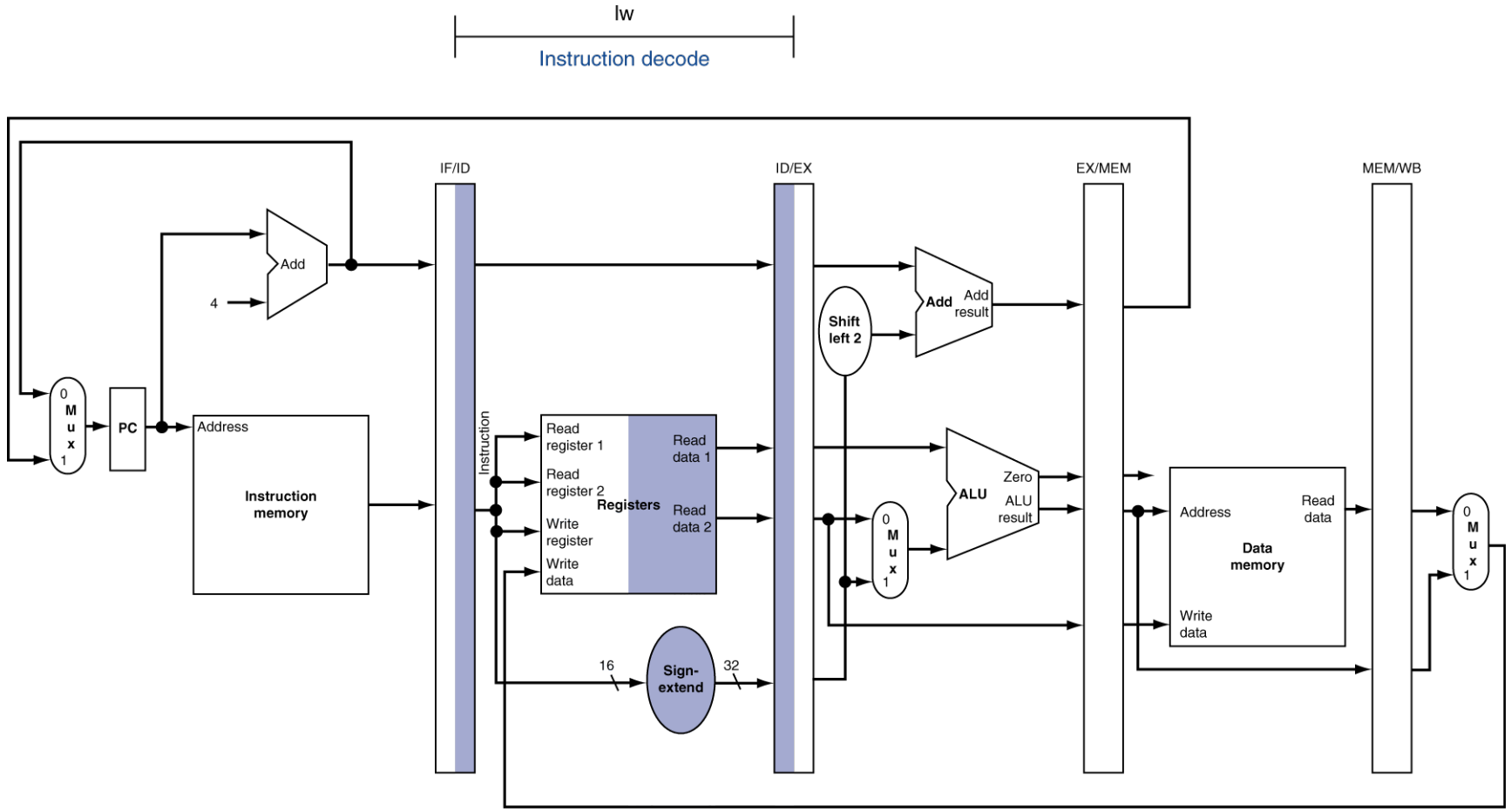
Pipeline Operation

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

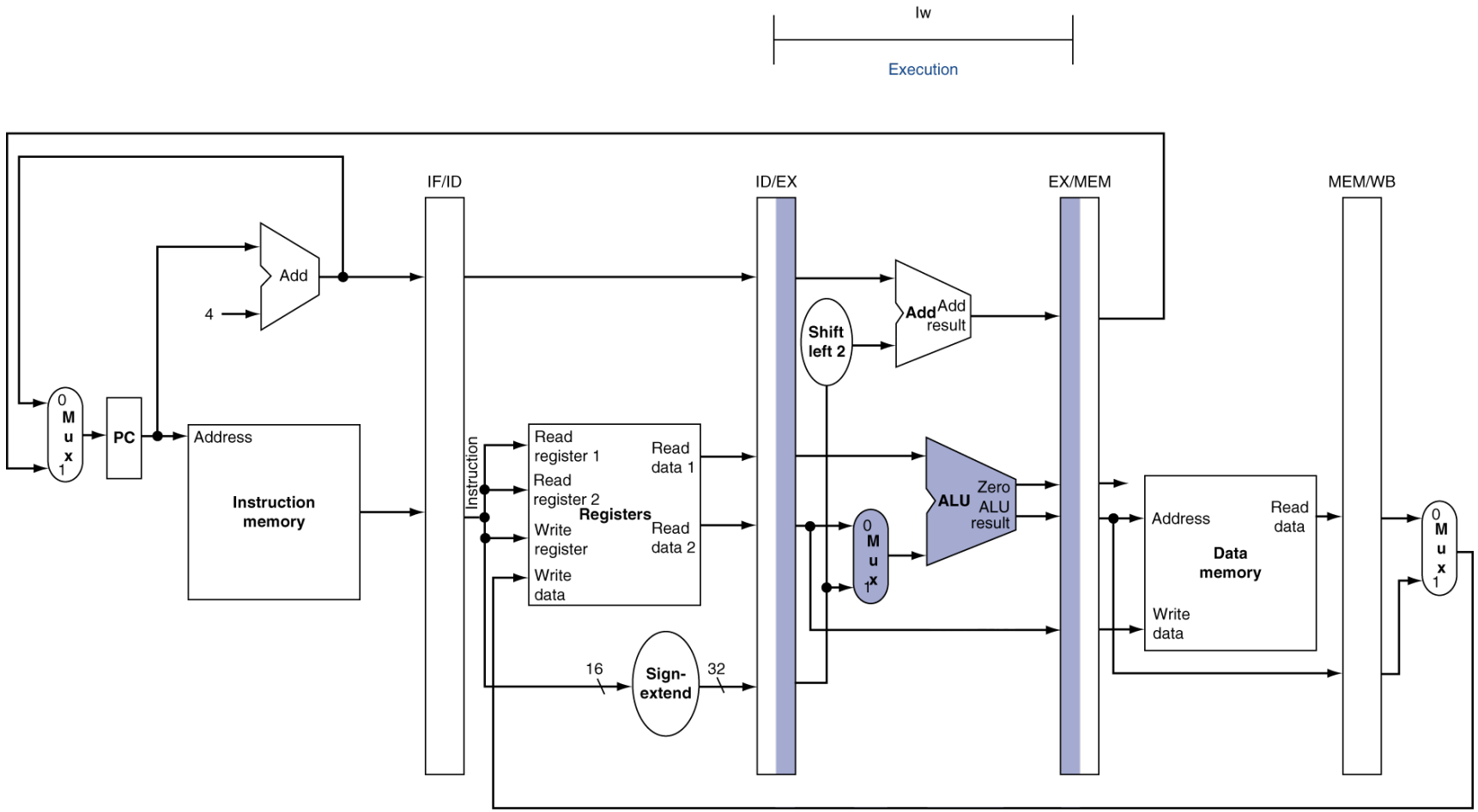
IF for Load, Store, …



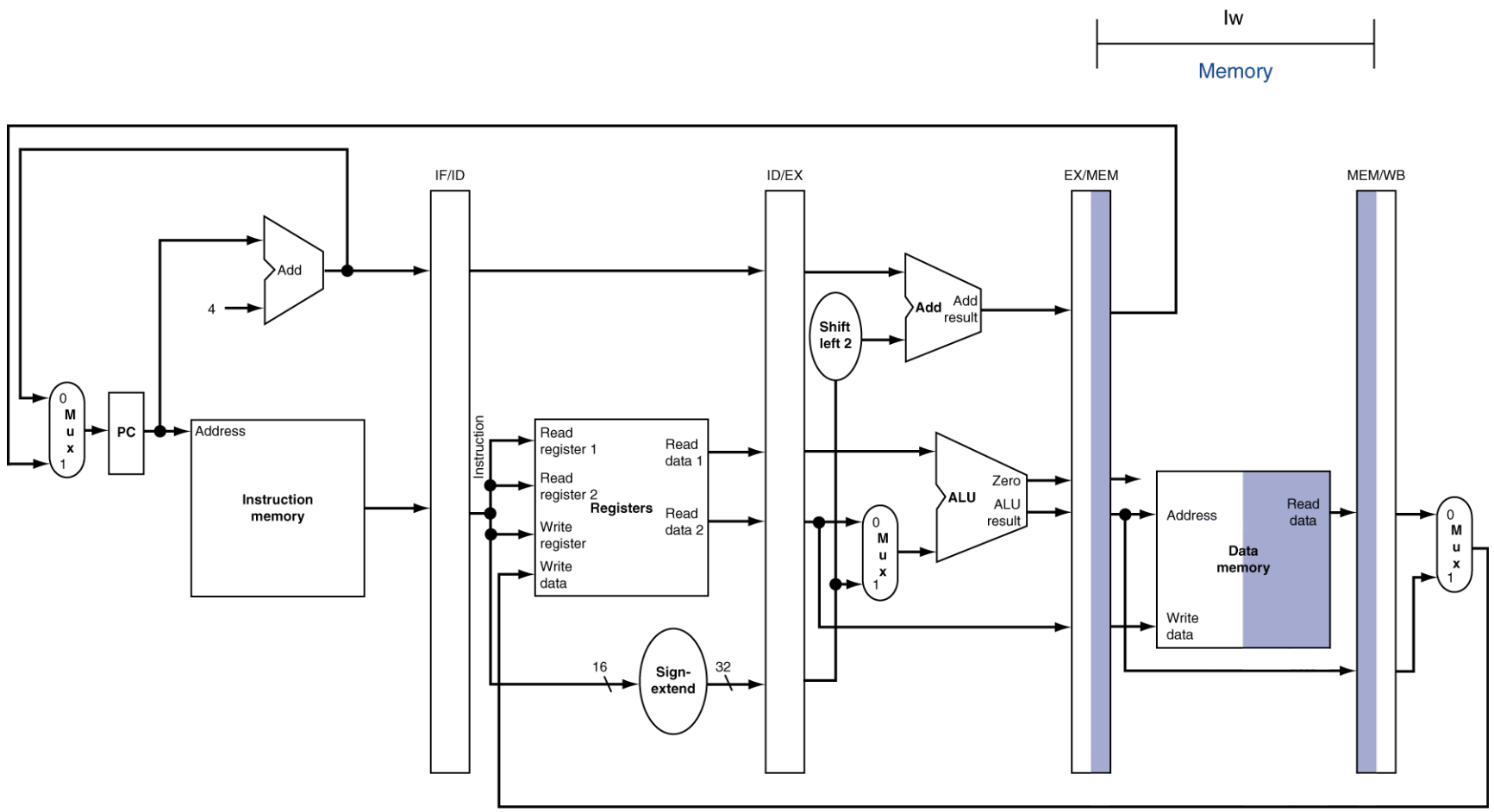
ID for Load, Store, …



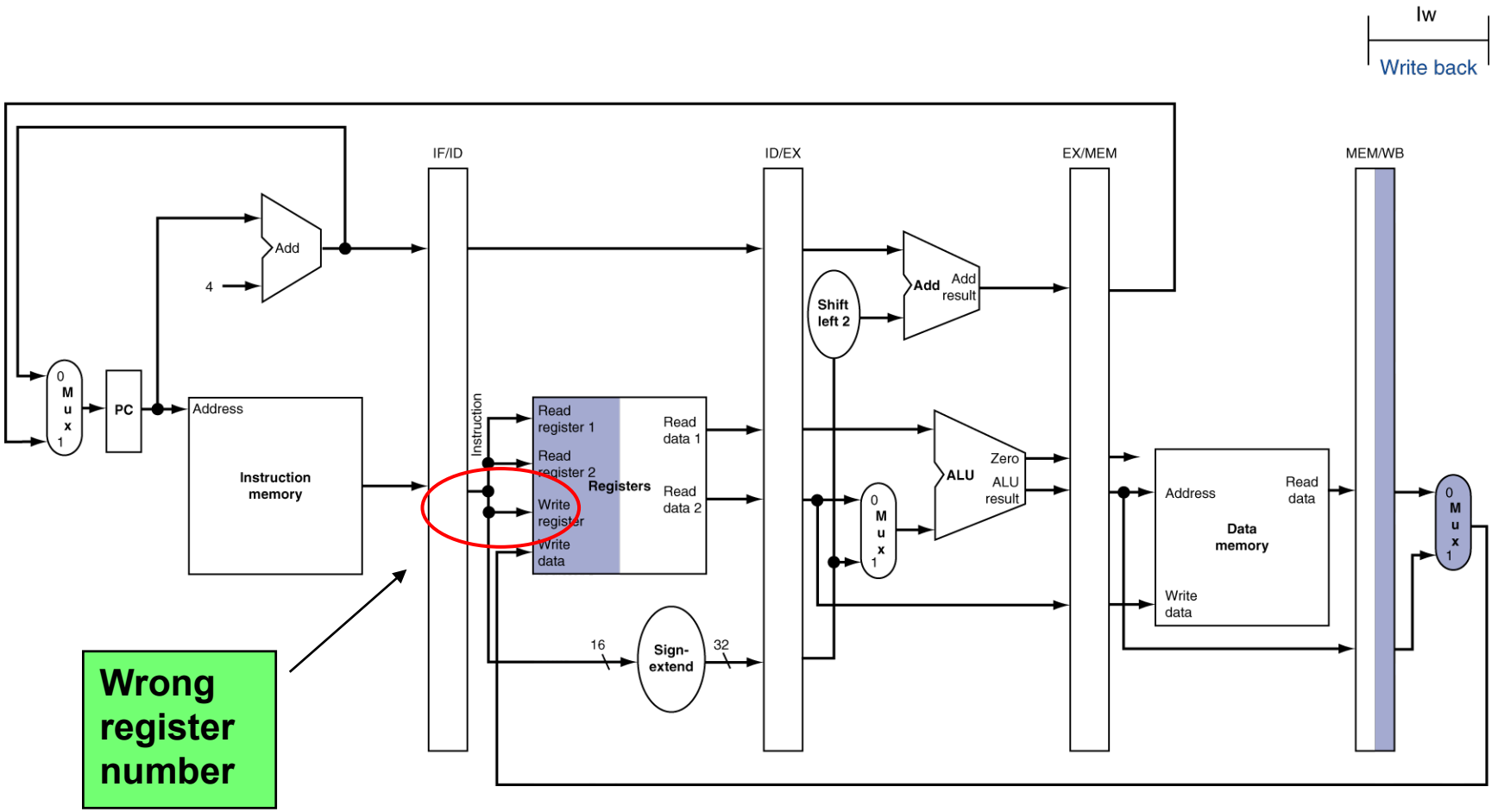
EX for Load


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
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

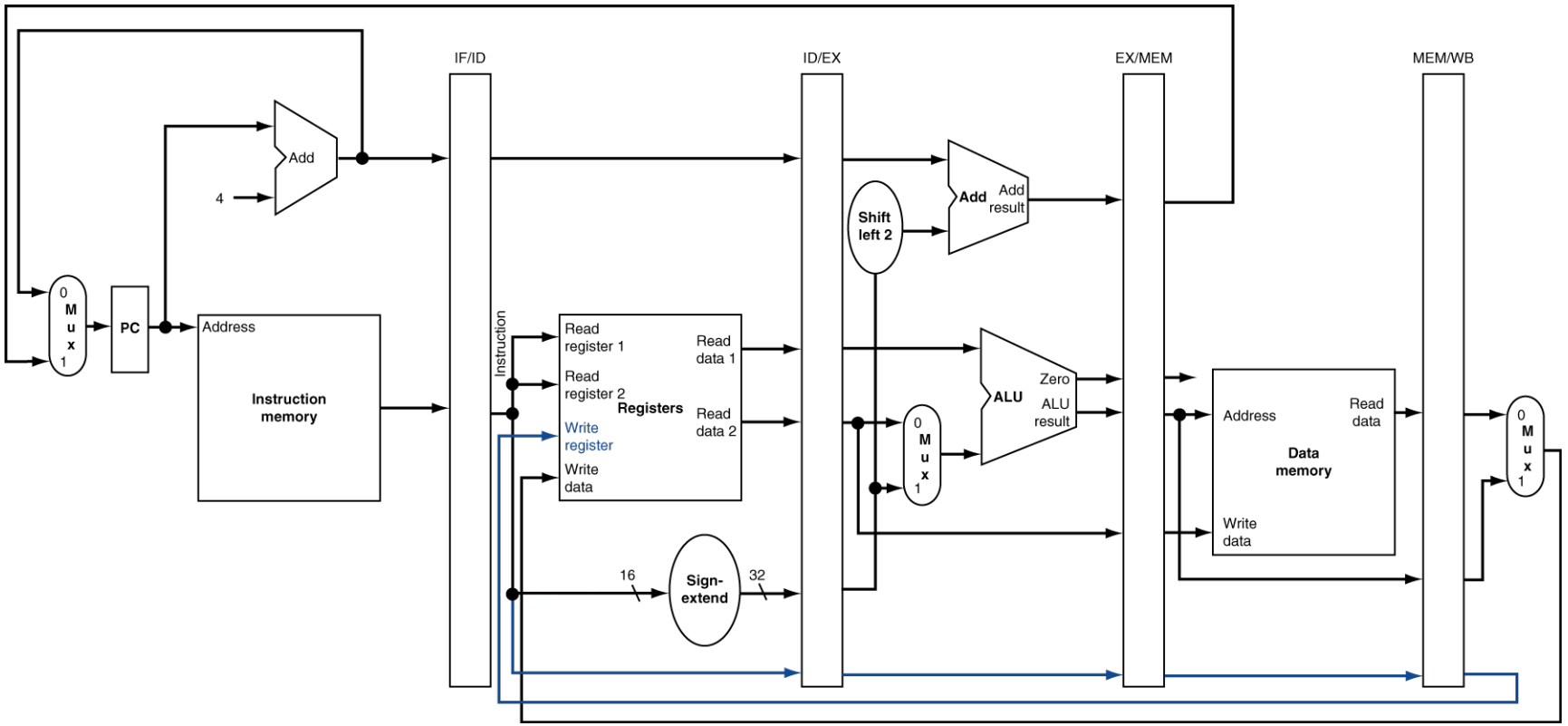
MEM for Load


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
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

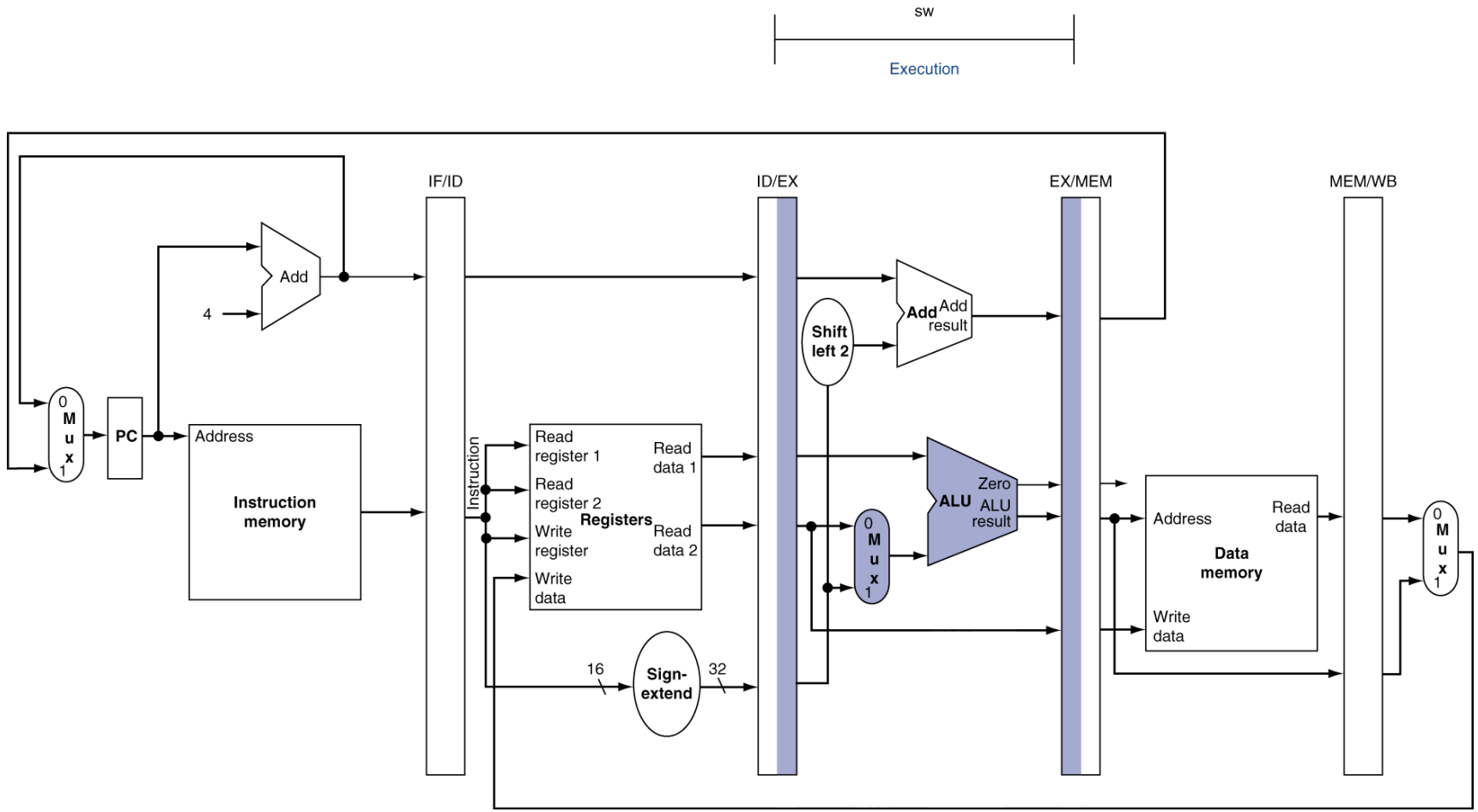
WB for Load


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
16
Sign-extend
32
Address
Data memory
Read data
Write data
Wrong register number

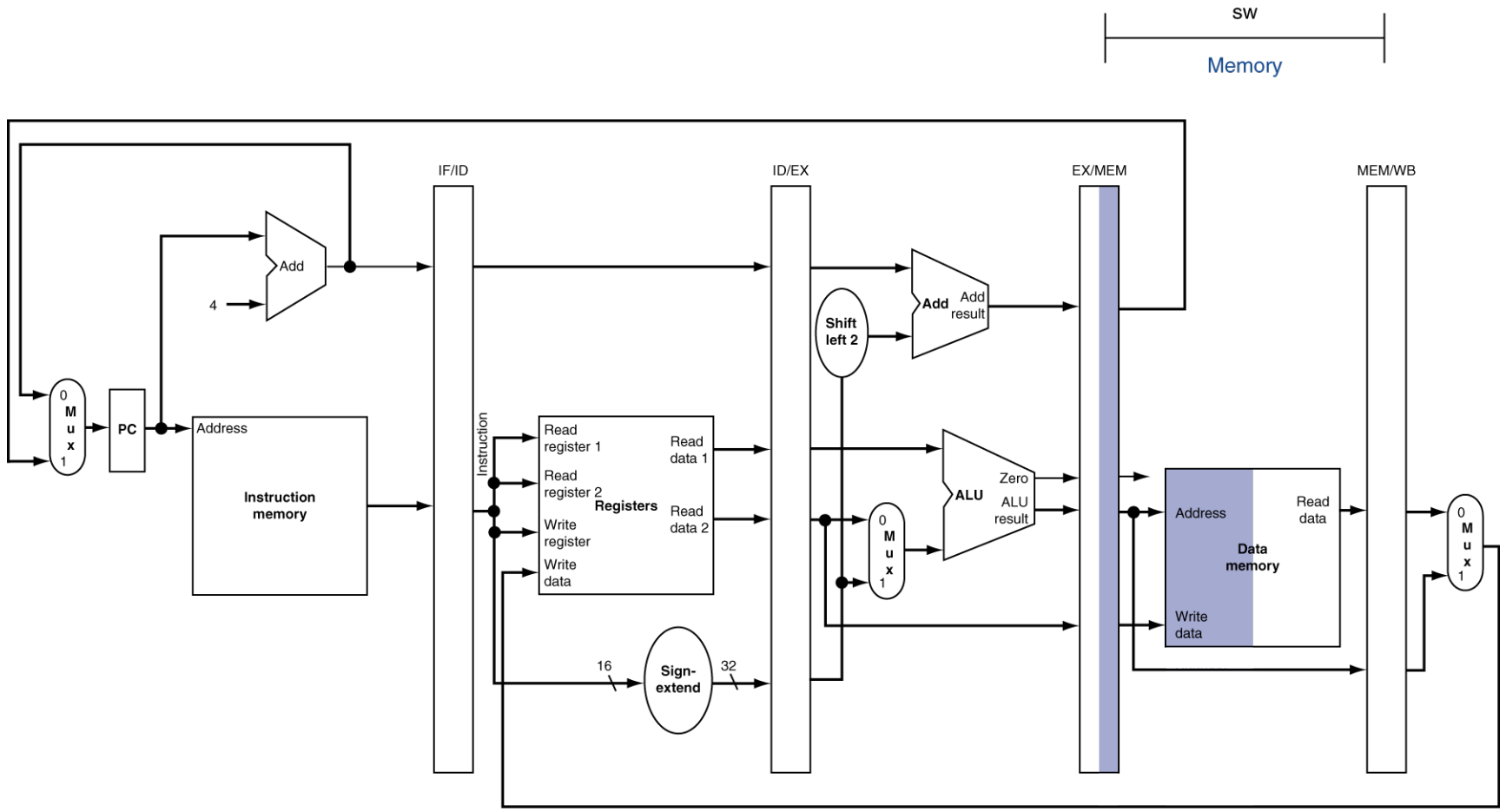
Corrected Datapath for Load



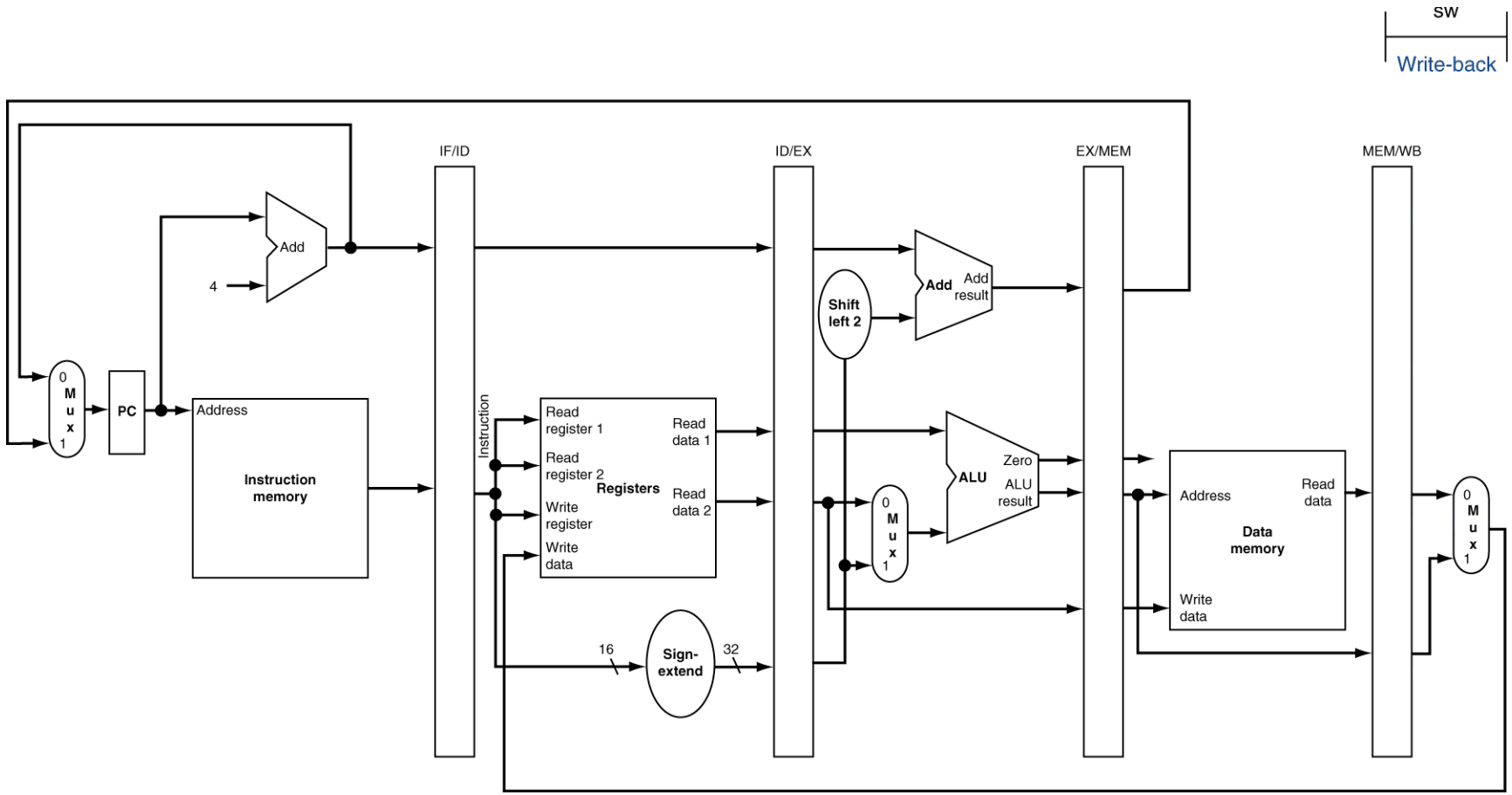
EX for Store


sw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
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
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

MEM for Store



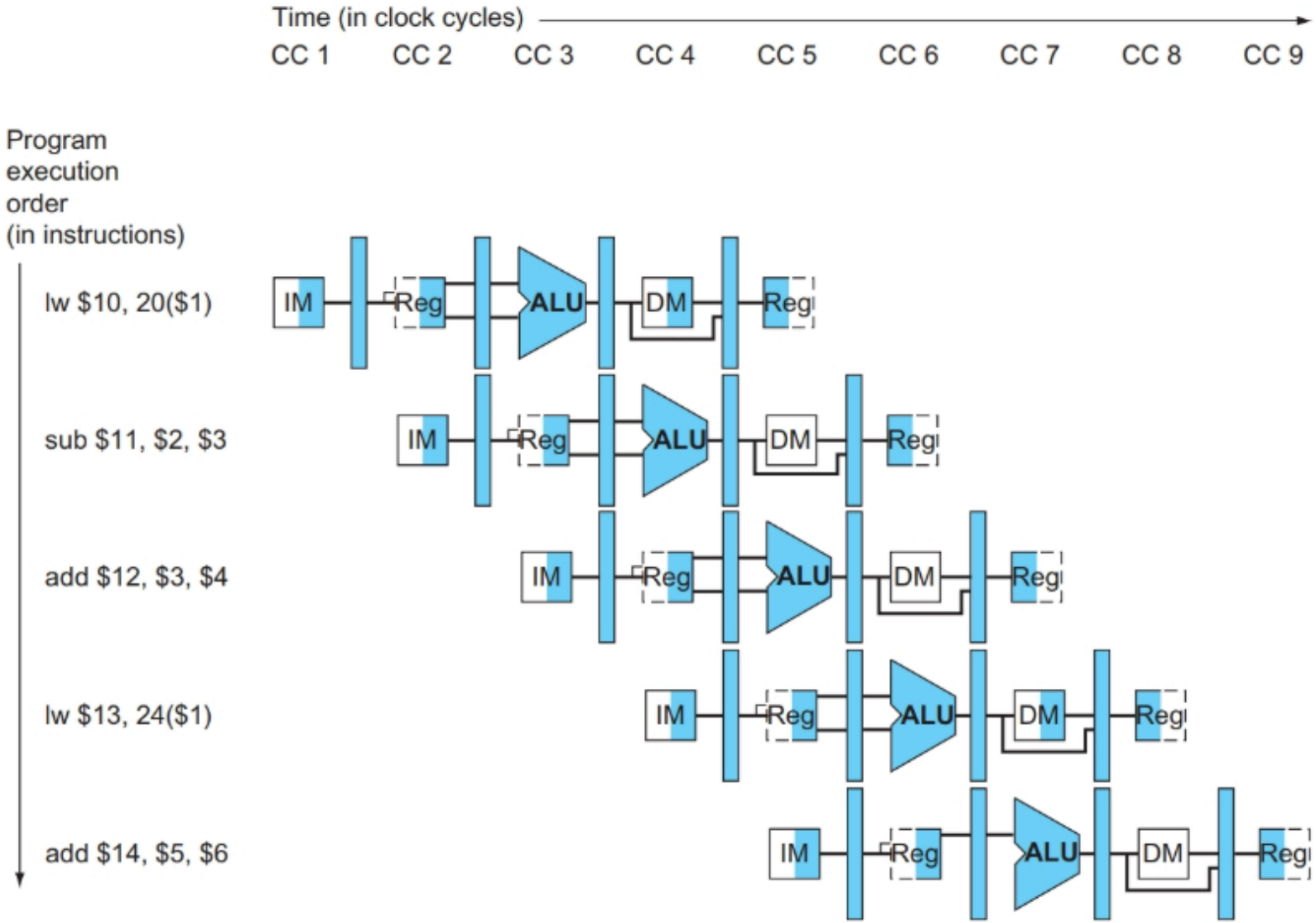
WB for Store


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
Address
Read data
Data memory
Write data

Example

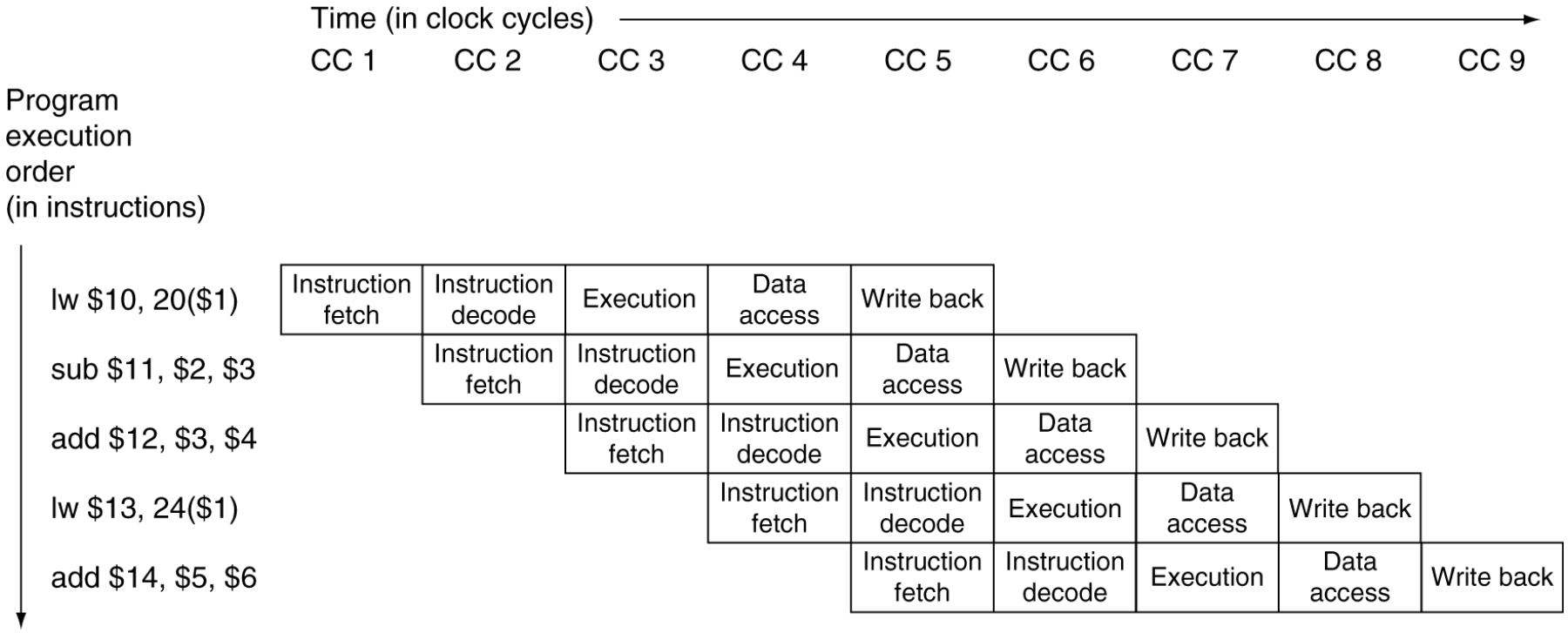
```
lw      $10, 20($1)
sub     $11, $2, $3
add     $12, $3, $4
lw      $13, 24($1)
add     $14, $5, $6
```

Multi-Cycle Pipeline Diagram



Multi-Cycle Pipeline Diagram

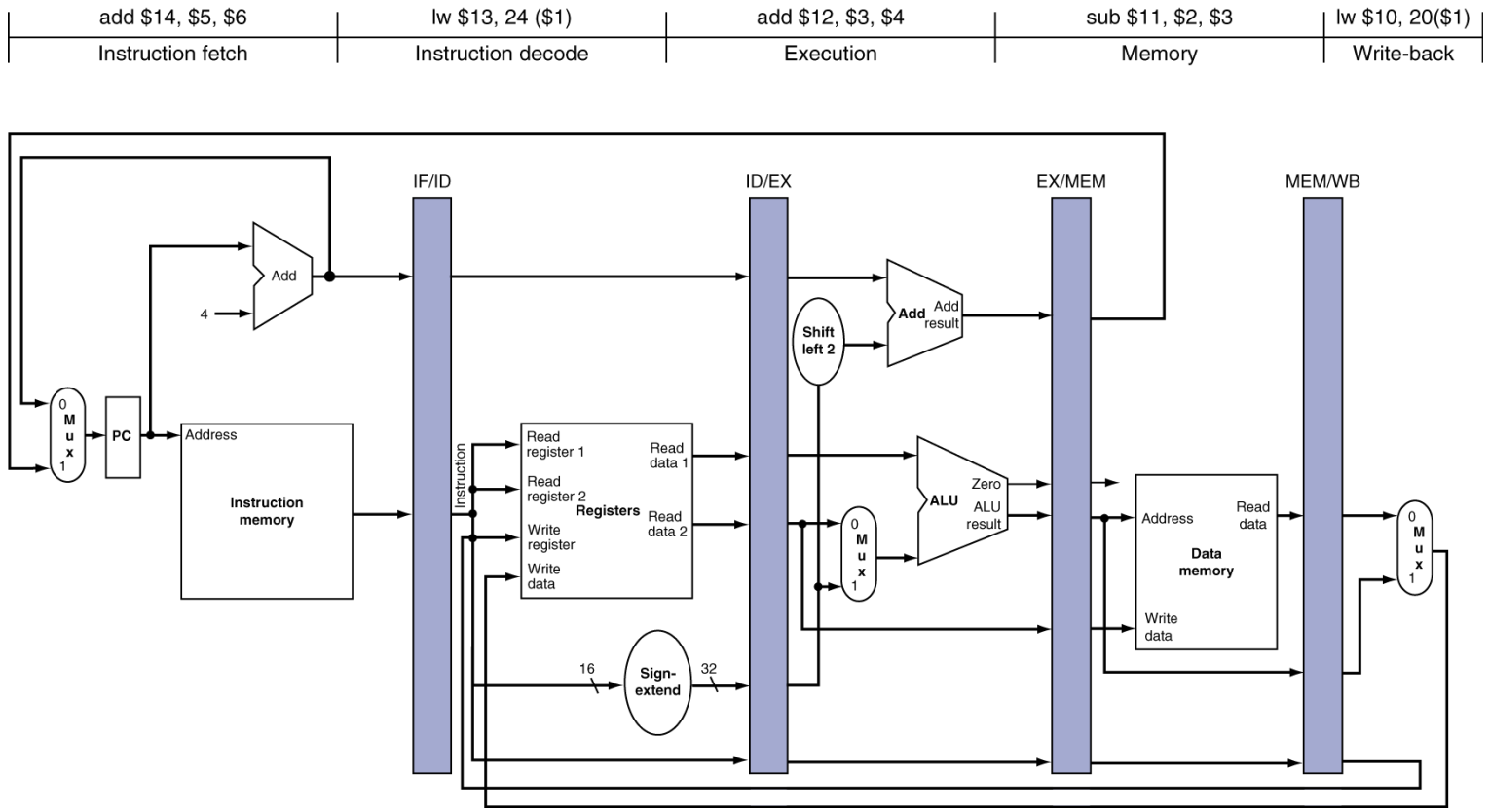
Traditional form



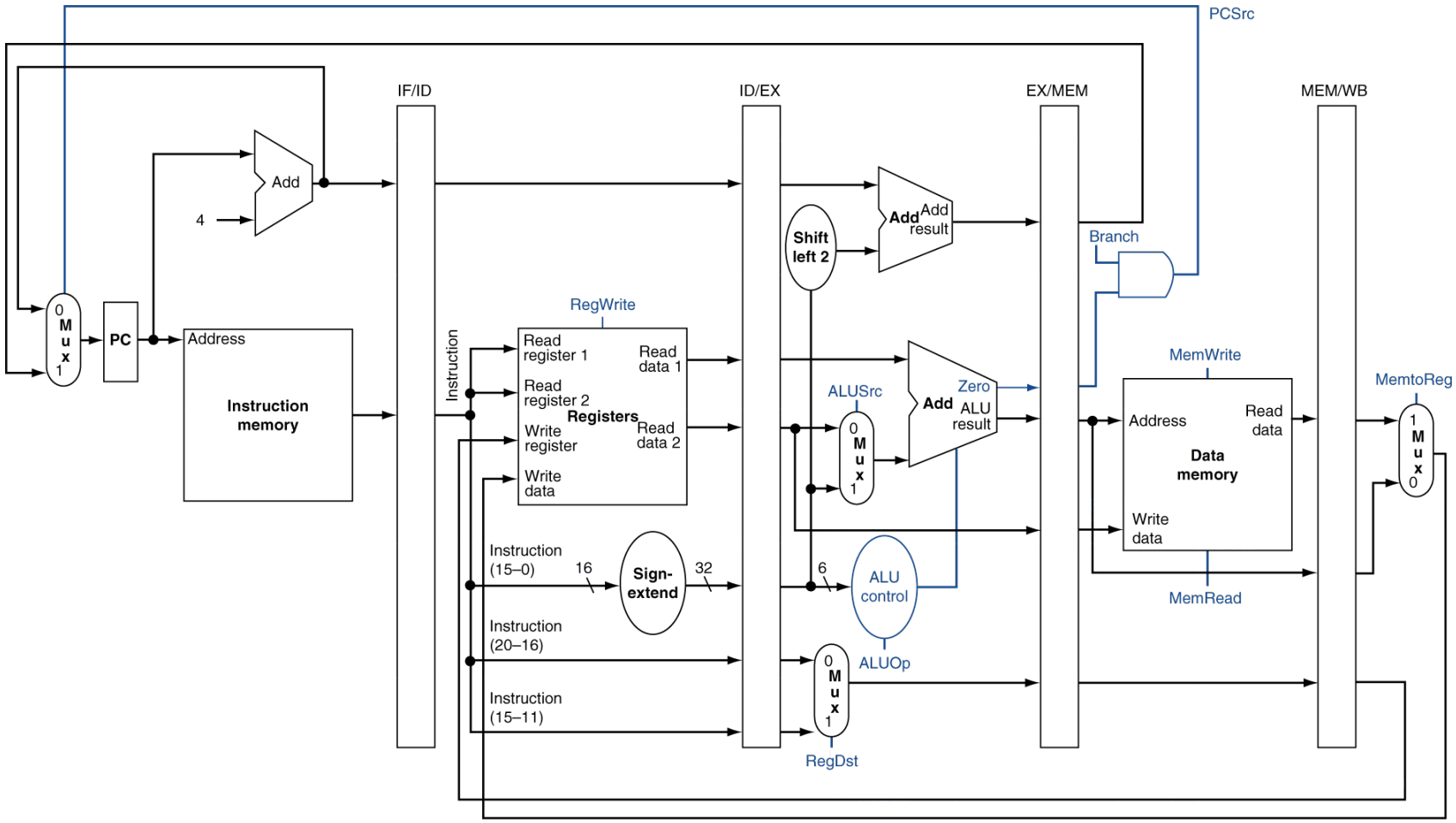
Single-Cycle Pipeline Diagram

- **State of pipeline in a given cycle (Clock cycle 5)**

```
lw      $10, 20($1)
sub     $11, $2, $3
add     $12, $3, $4
lw      $13, 24($1)
add     $14, $5, $6
```



Pipelined Control (Simplified)

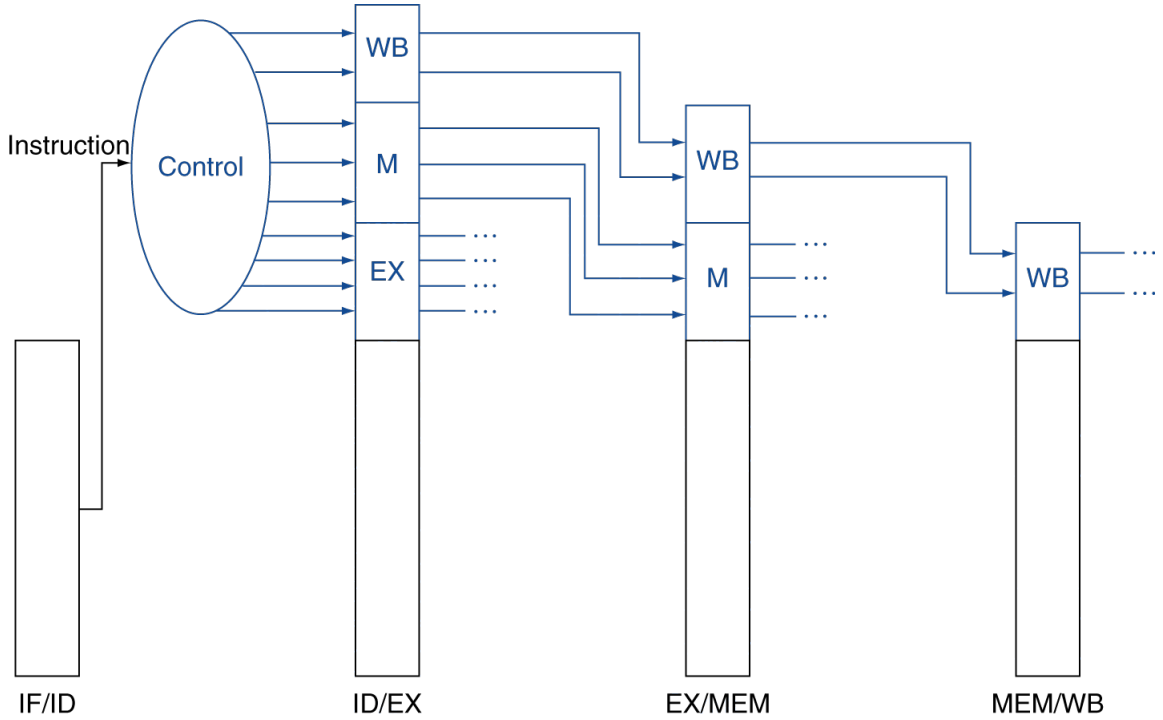


Control Signals

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Pipelined Control

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



Pipelined Control