COMPUTER ORGANIZATION (IS F242)

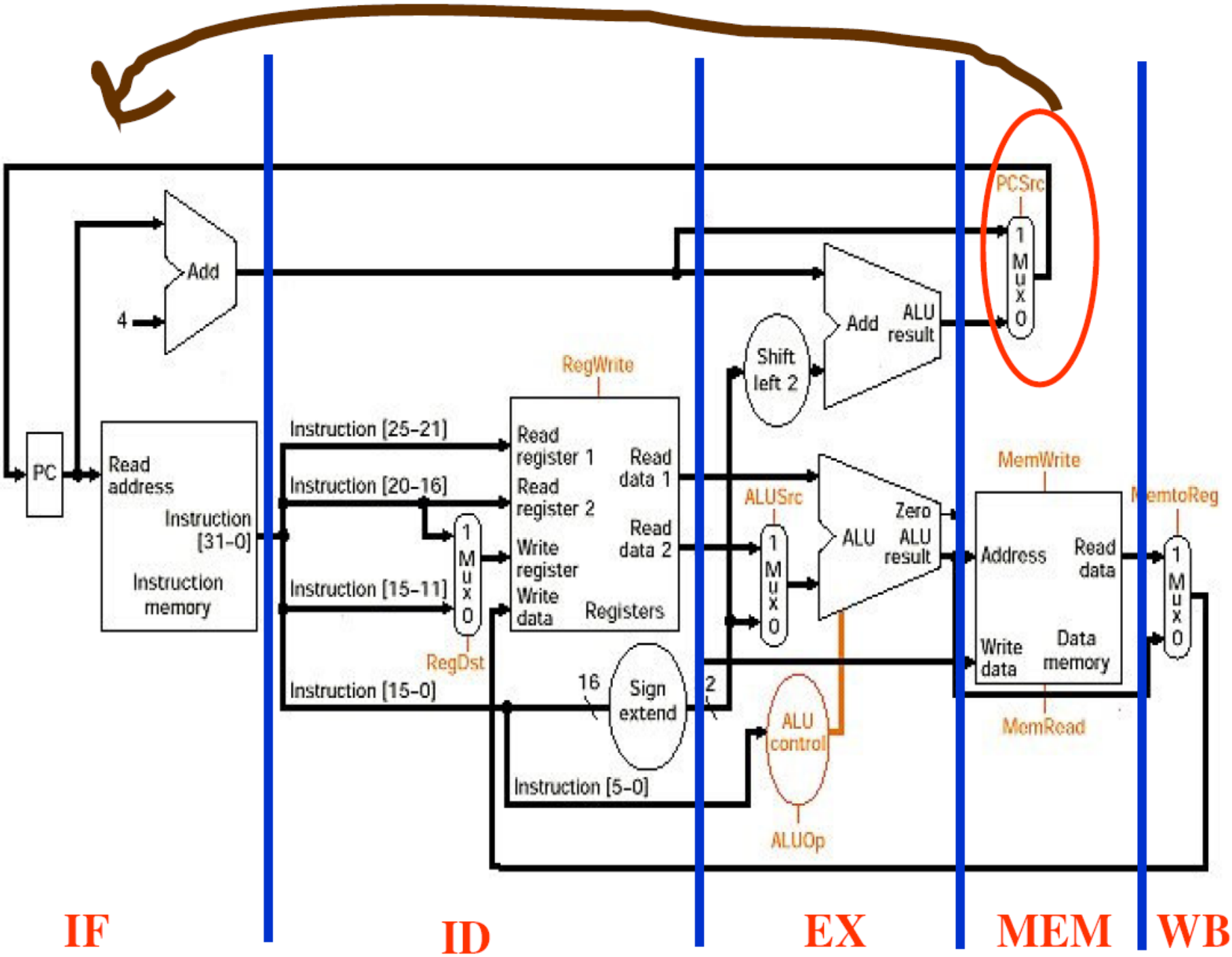
LECT 42: PIPELINING

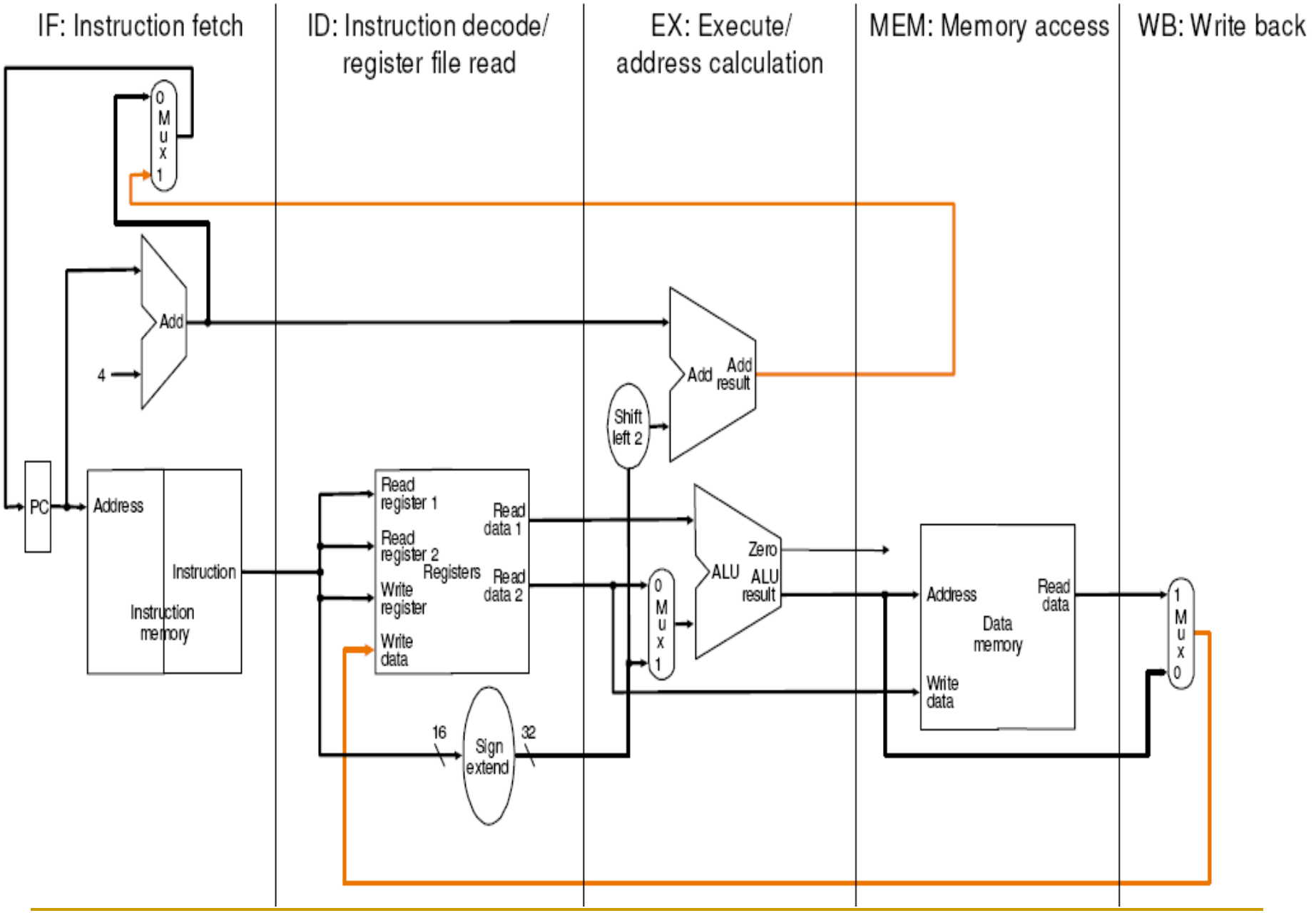
Pipelining and ISA Design

- MIPS ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - 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
  - Alignment of memory operands
    - Memory access takes only one cycle

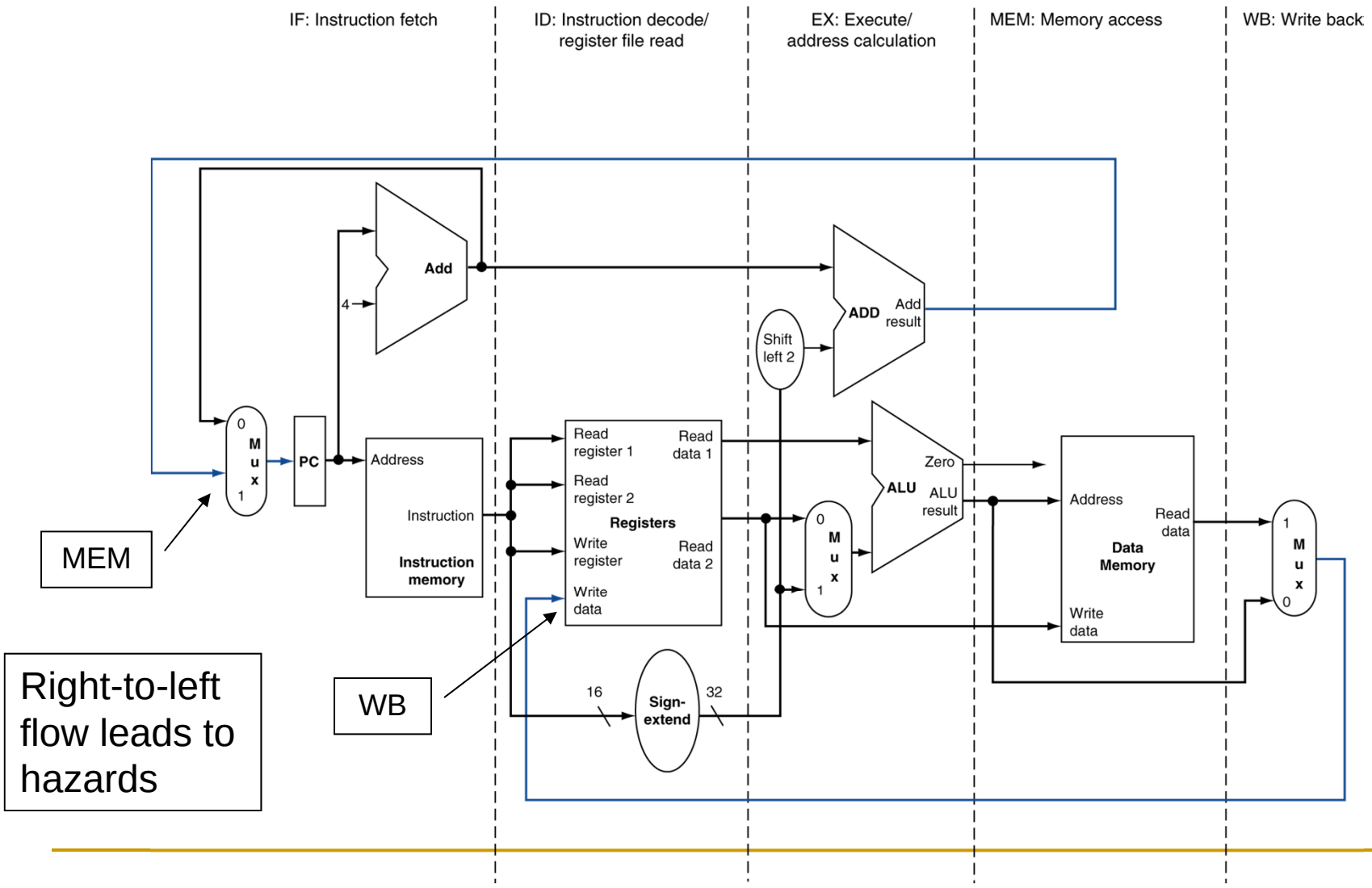
Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle
  - Ensures instructions in different stages of the pipeline do not interfere with one another


PCSrc
Add
4
M
u
x
0
1
Add
ALU result
Shift left 2
RegWrite
PC
Read address
Instruction [31-0]
Instruction memory
Instruction [25-21]
Instruction [20-16]
Instruction [15-11]
Instruction [15-0]
Instruction [5-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
RegDst
ALUSrc
ALU
Zero
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead
MemtoReg
16
Sign extend
2
ALU control
ALUOp
IF
ID
EX
MEM
WB


IF: Instruction fetch
ID: Instruction decode/ register file read
EX: Execute/ address calculation
MEM: Memory access
WB: Write back
0
M
u
x
1
Add
4
PC
Address
Instruction
Instruction memory
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
Read data
Write data
1
M
u
x
0

MIPS Pipelined Datapath



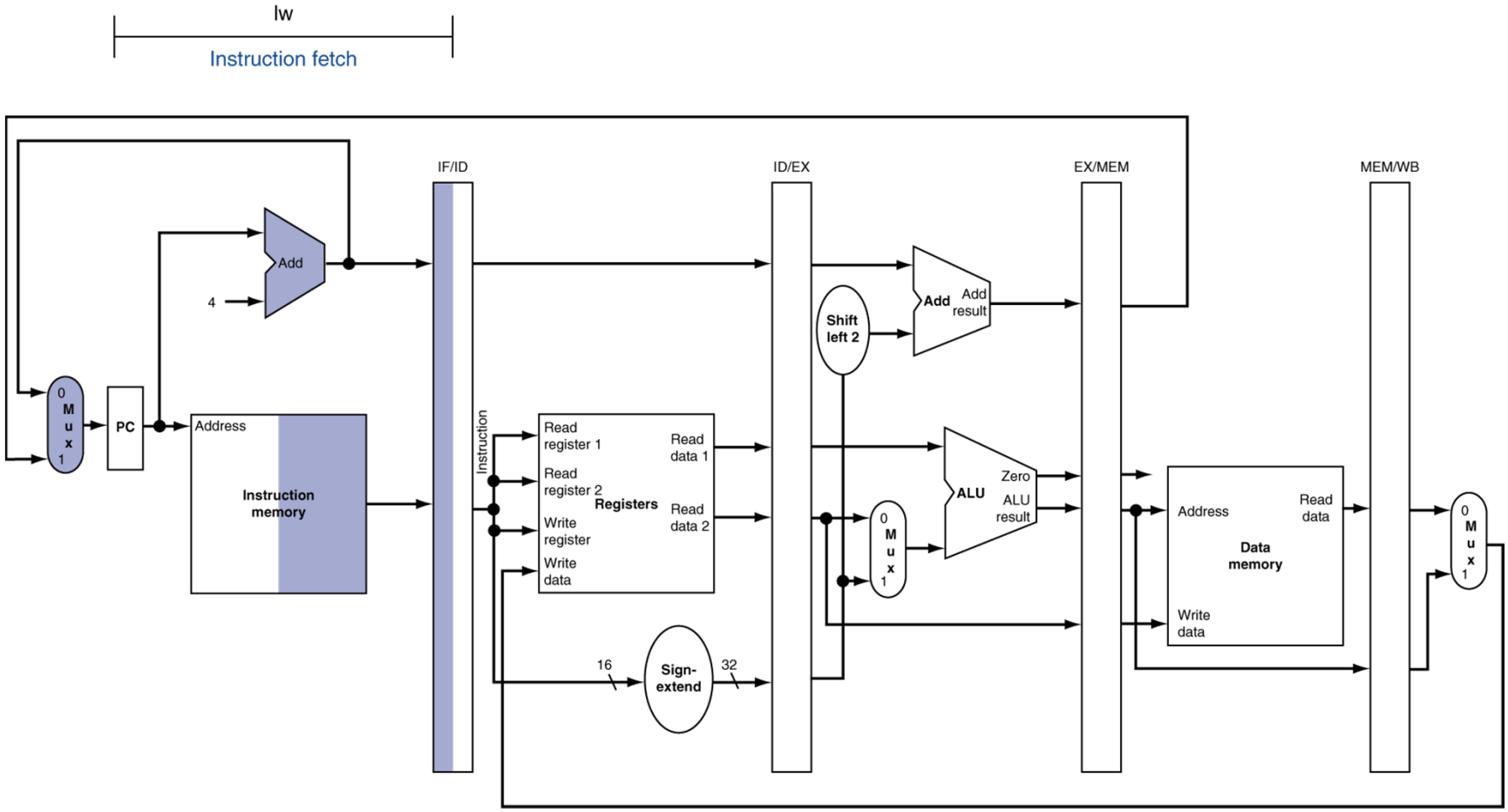
Right to left data flow

- The write back stage, which places the result back in the register file in the middle of the datapath
- The selection of the next value of the PC, choosing between the incremented PC and the branch address from the MEM stage
- The right to left data flow does not affect the current instruction
  - Later instructions in the pipeline are influenced by the right to left data movement

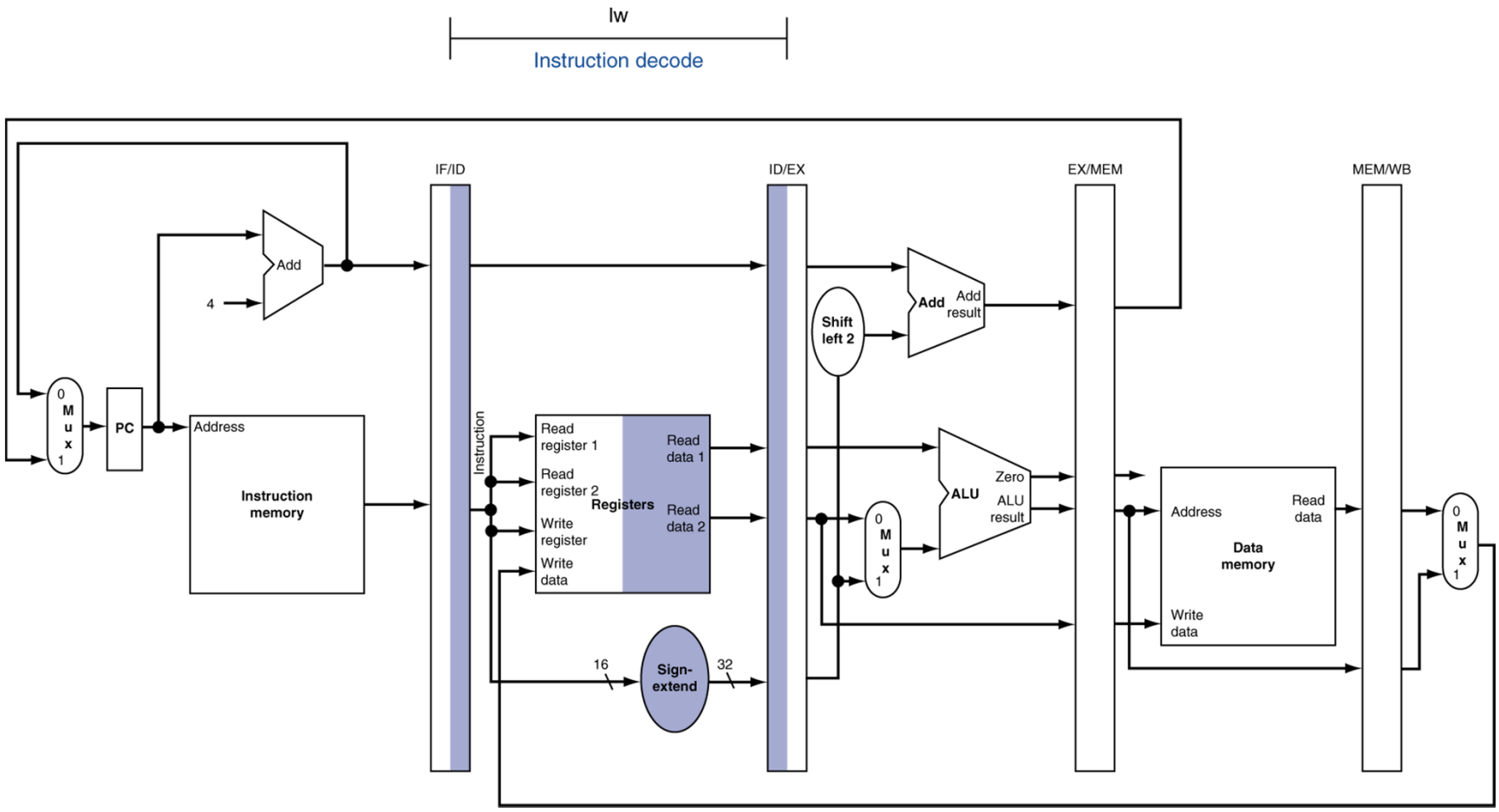
Pipeline Operation

- 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
  - "multi-clock-cycle" diagram
    - Graph of operation over time
- We'll look at "single-clock-cycle" diagrams for load & store

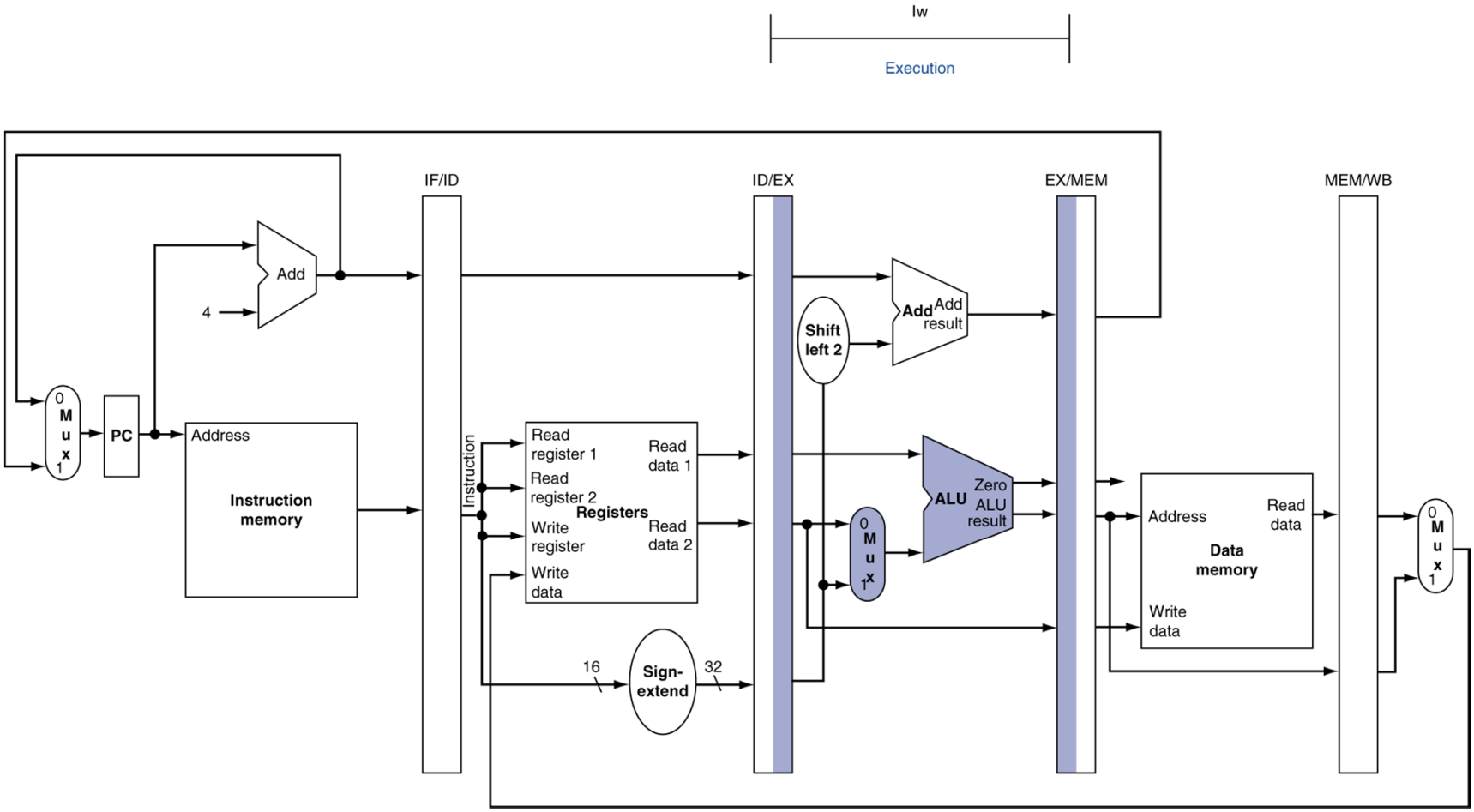
IF for Load, Store, …



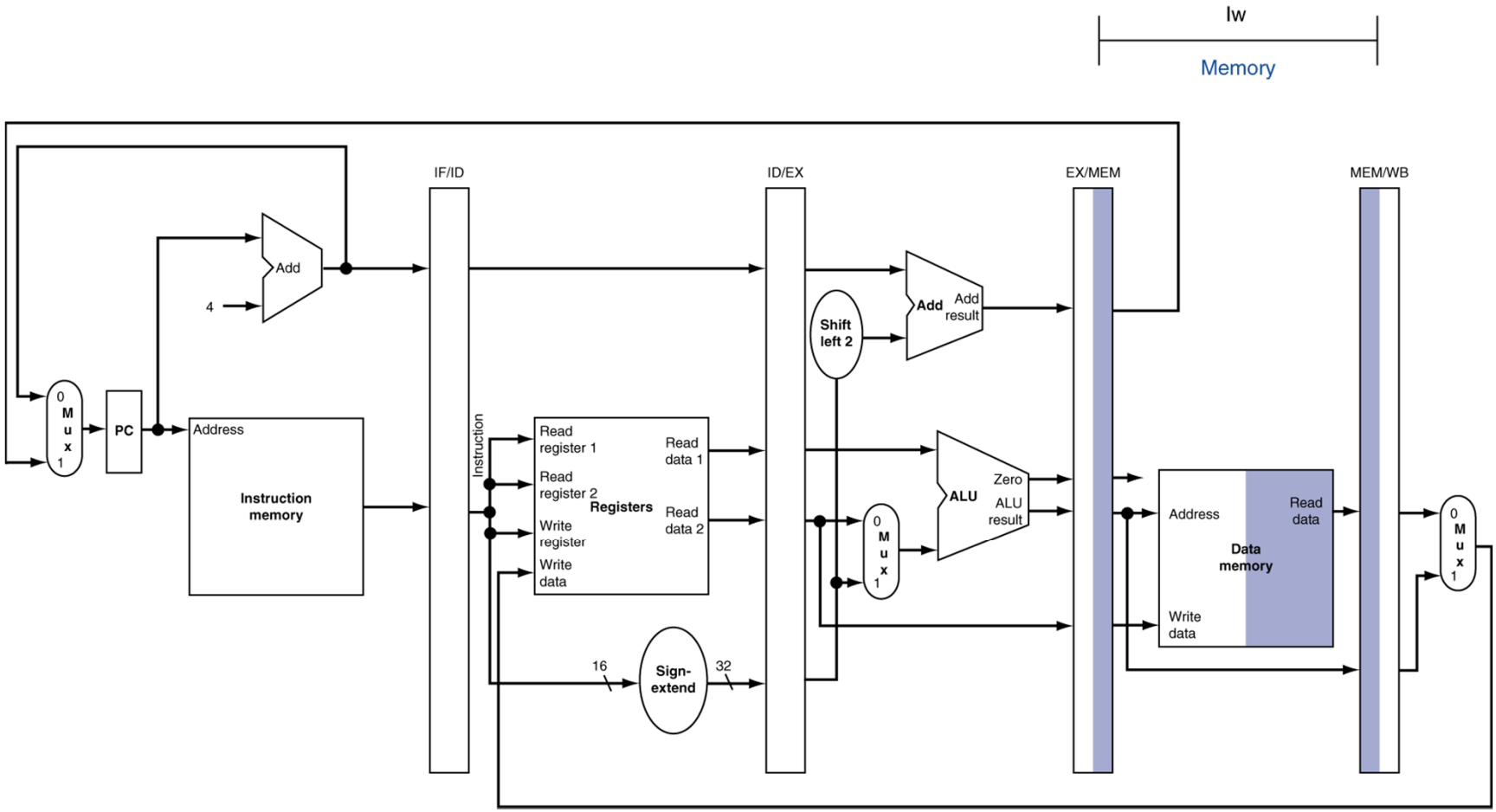
ID for Load, Store, …



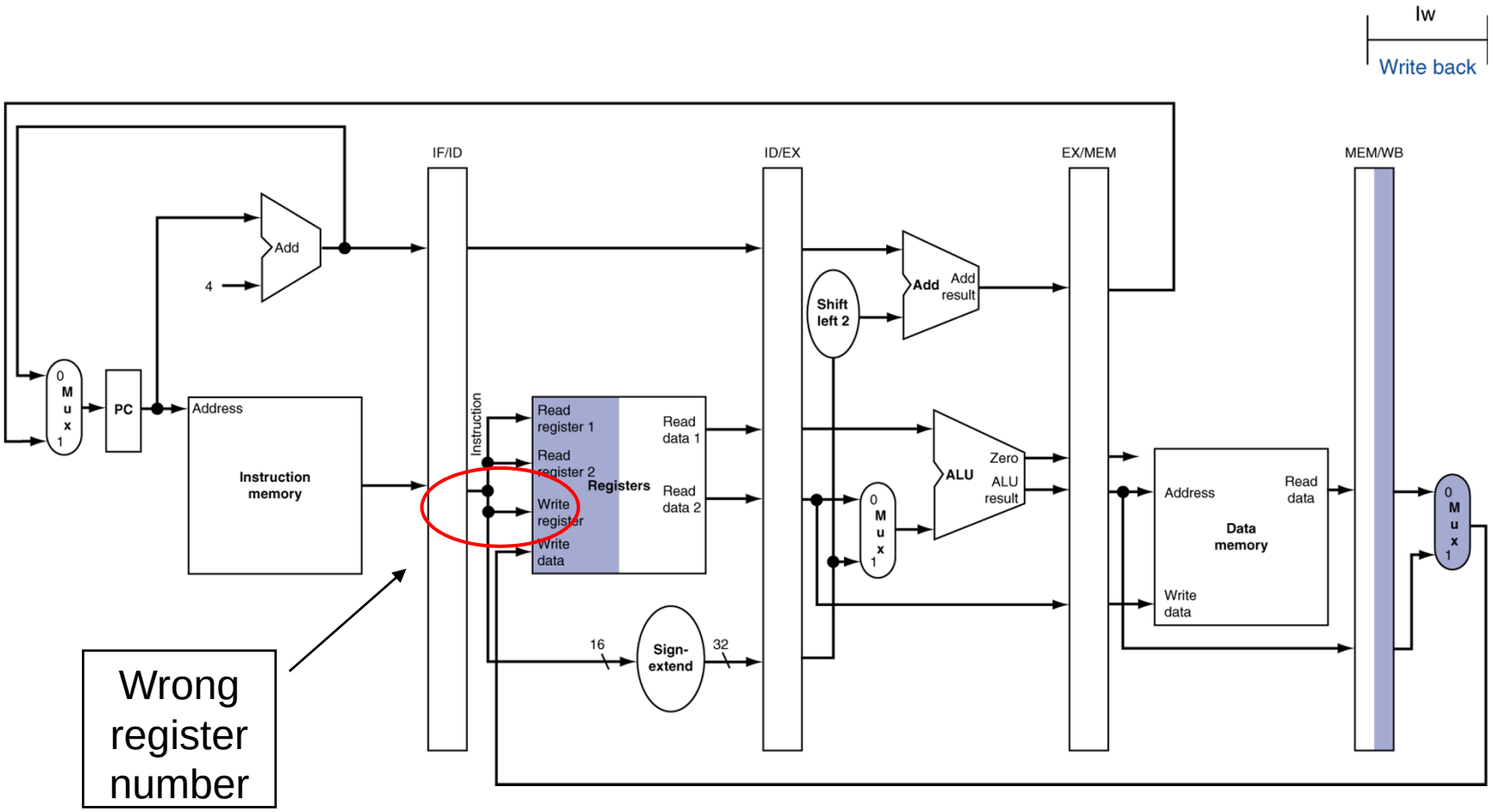
EX for Load


lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
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
Data memory
Read data
Write data
16
Sign-extend
32

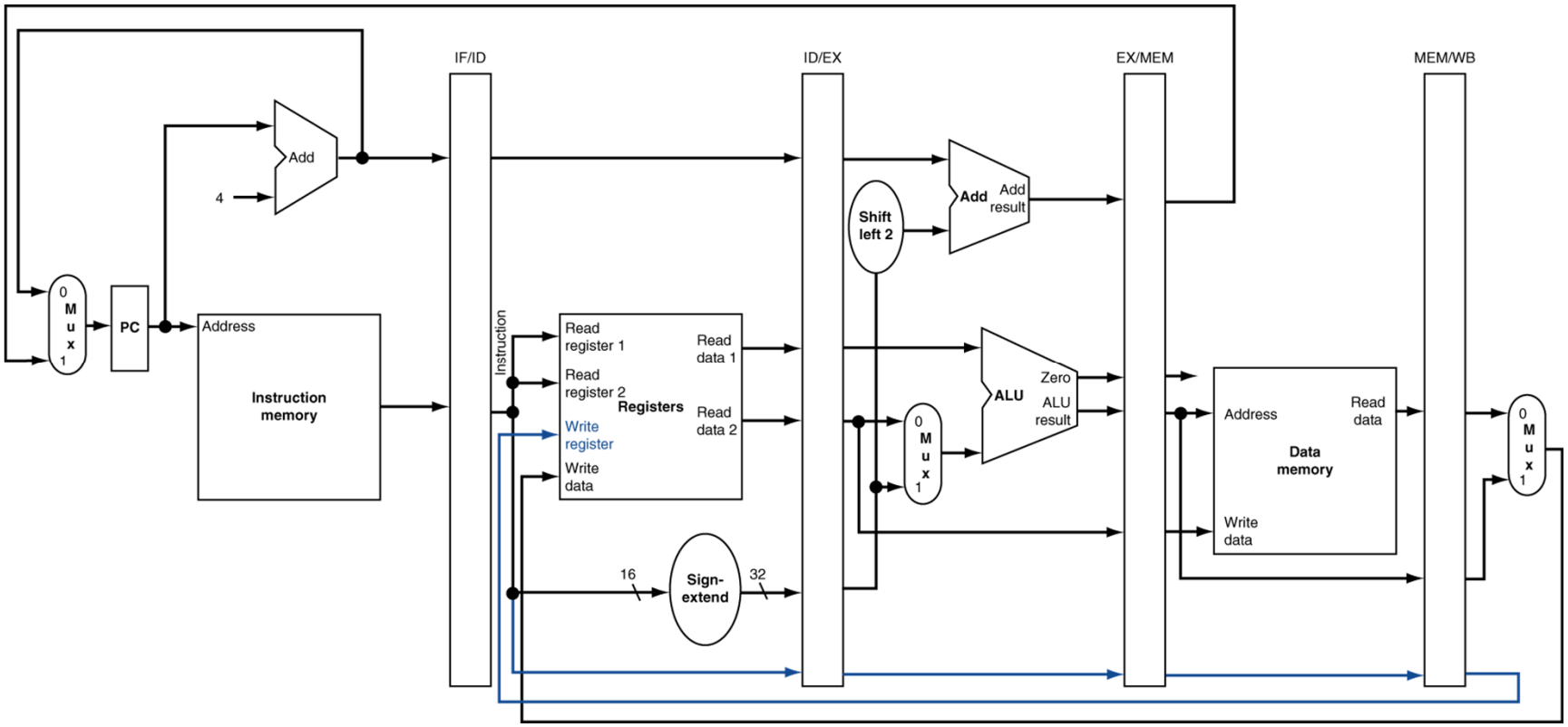
MEM for Load


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
Data memory
Read data
Write data
16
Sign-extend
32

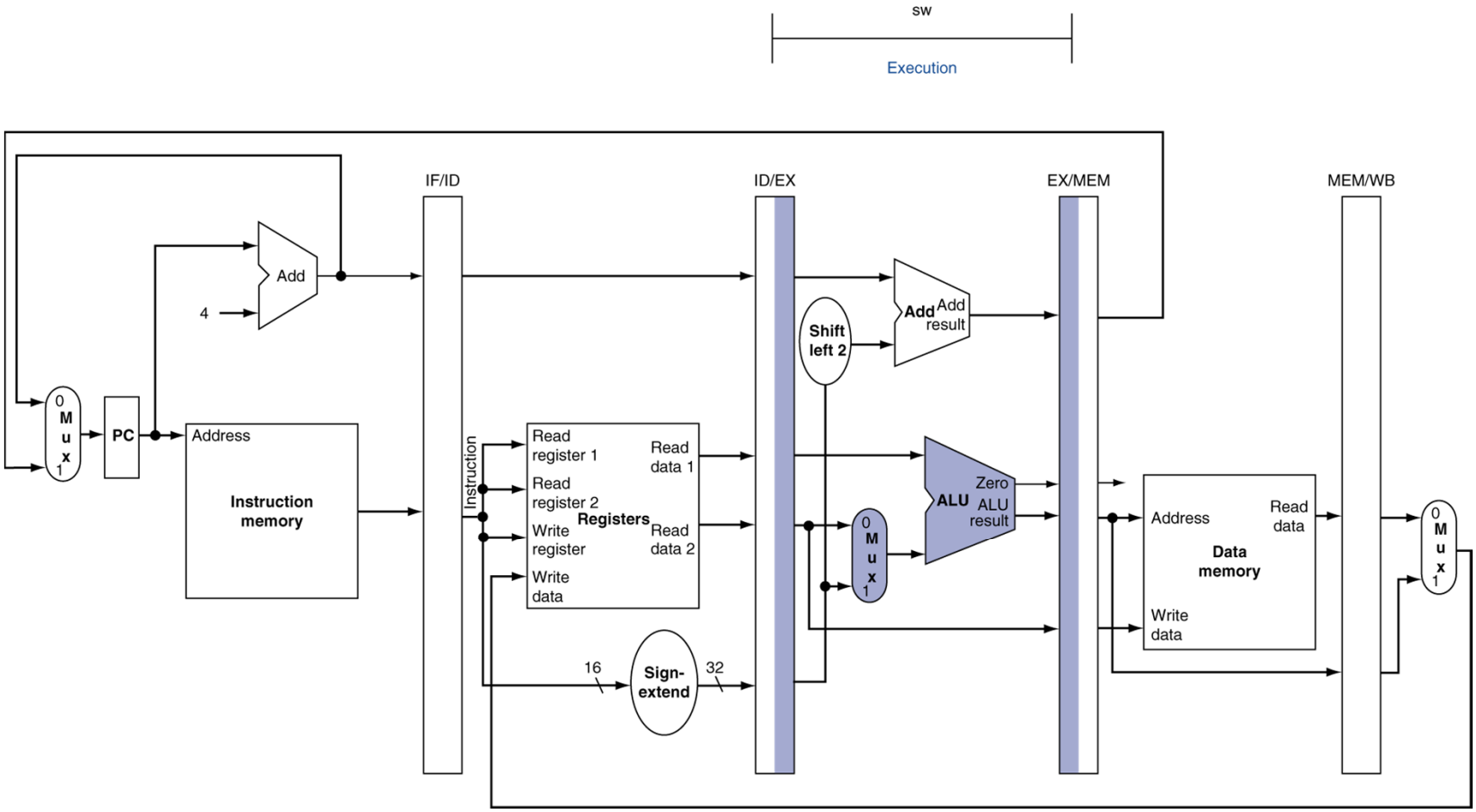
WB for Load


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
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

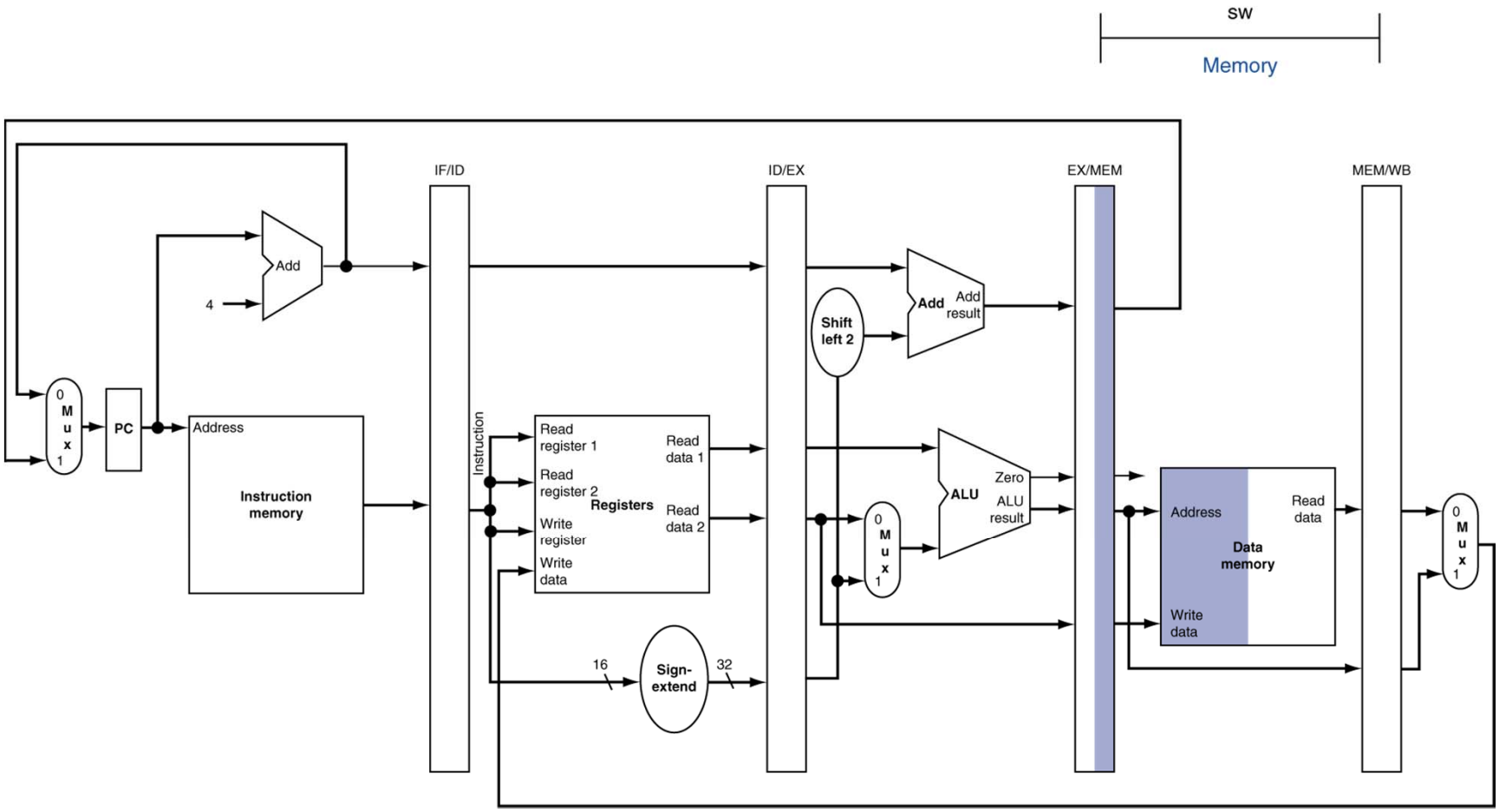
Corrected Datapath for Load



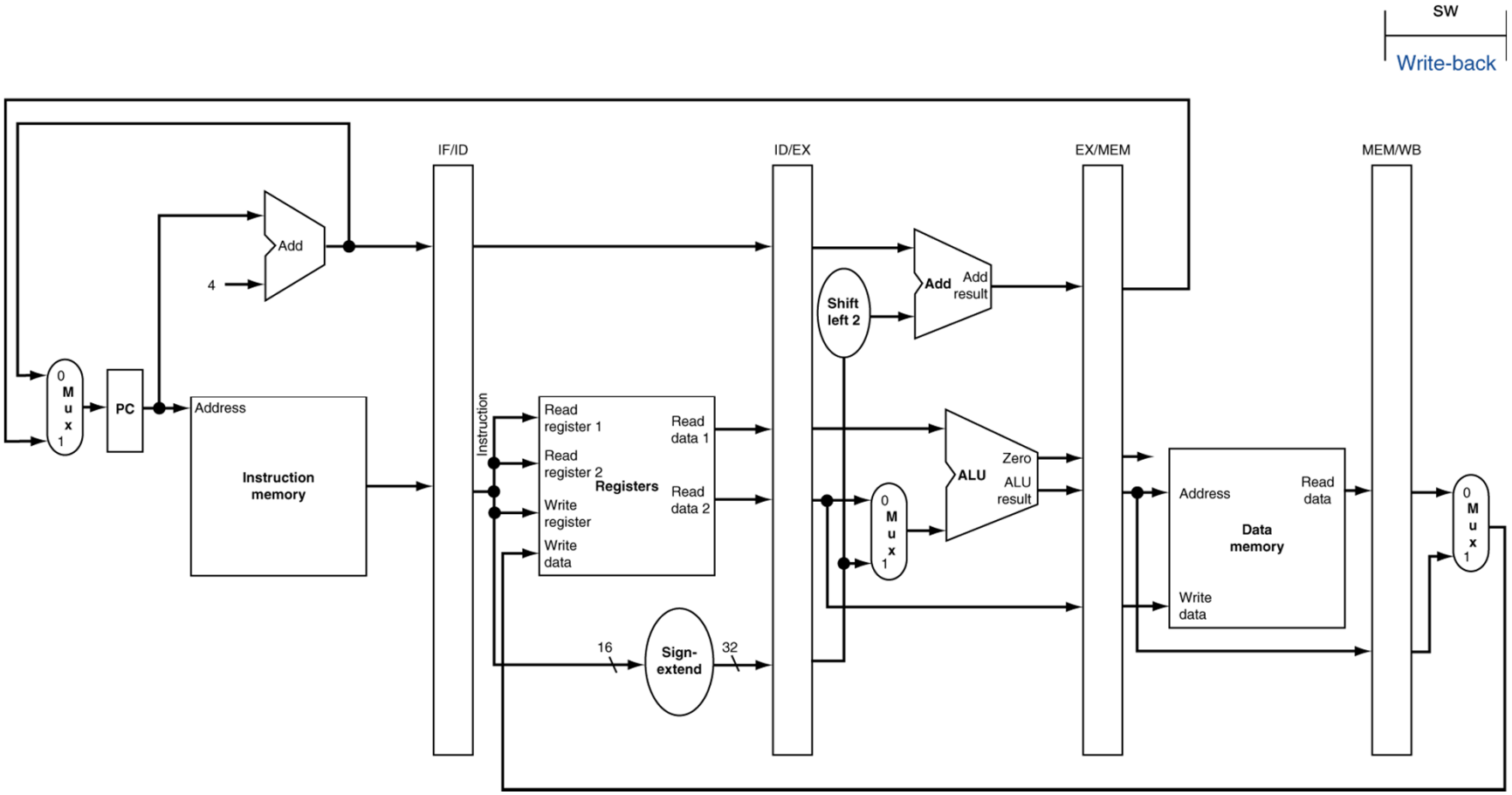
EX for Store


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

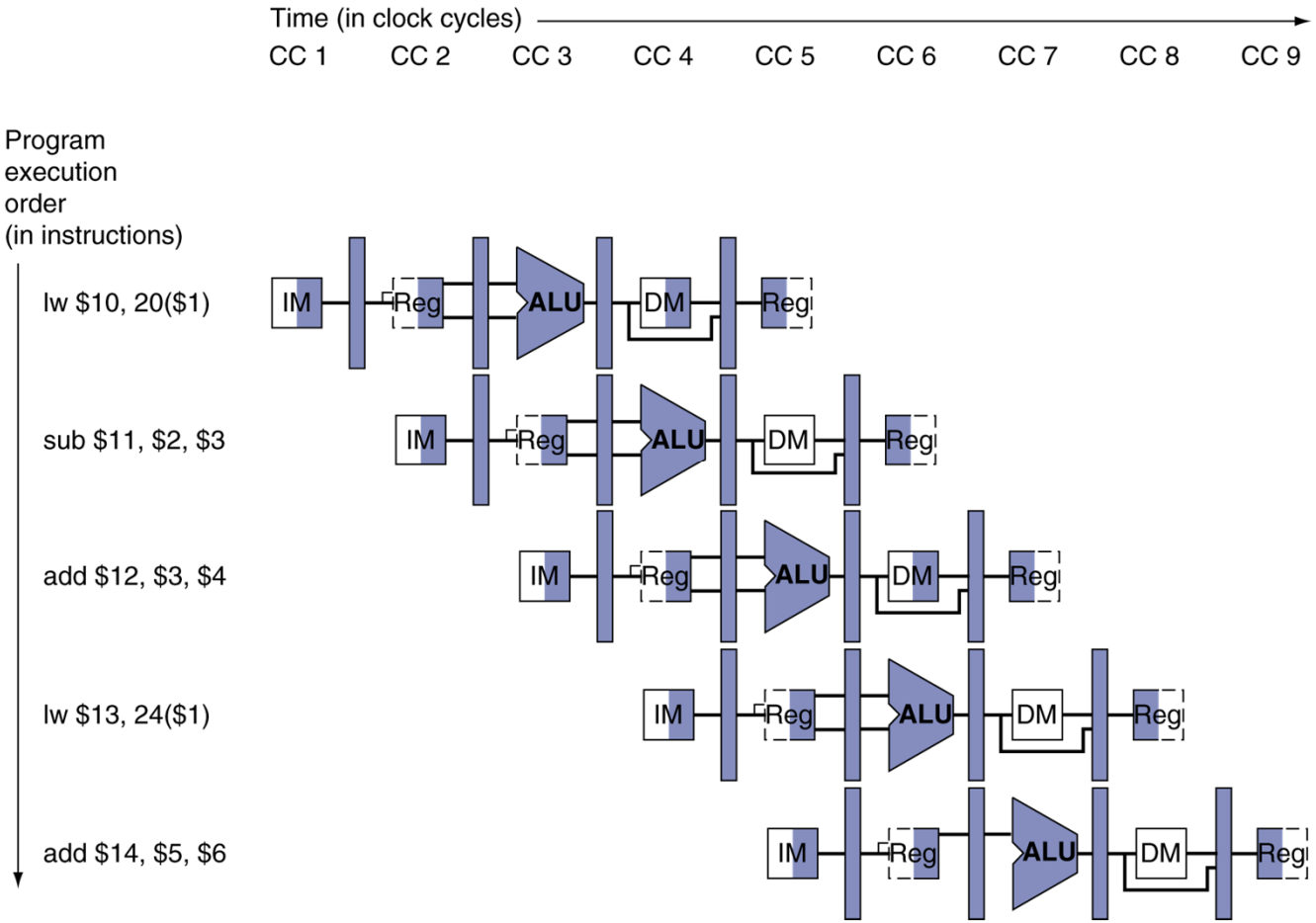
Multiple clock cycle pipeline

Consider the following instruction sequence

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

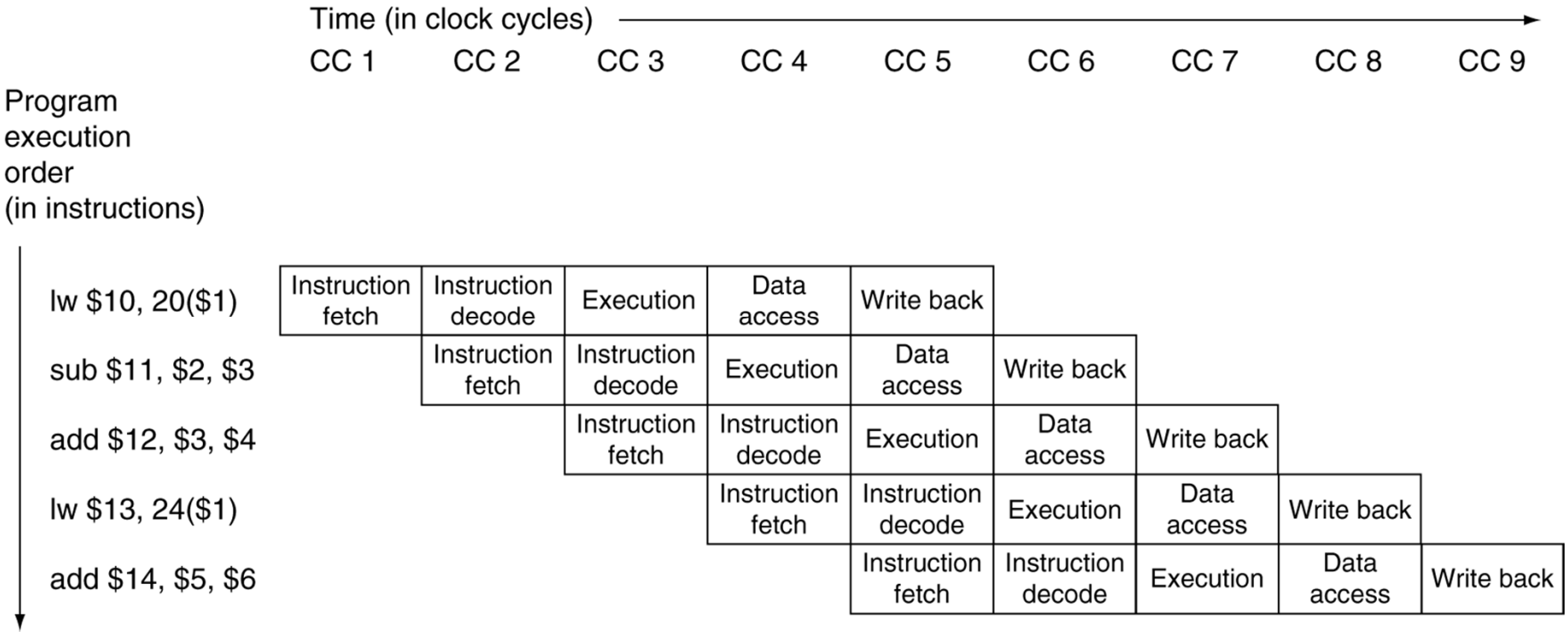
Multi-Cycle Pipeline Diagram

- Form showing resource usage



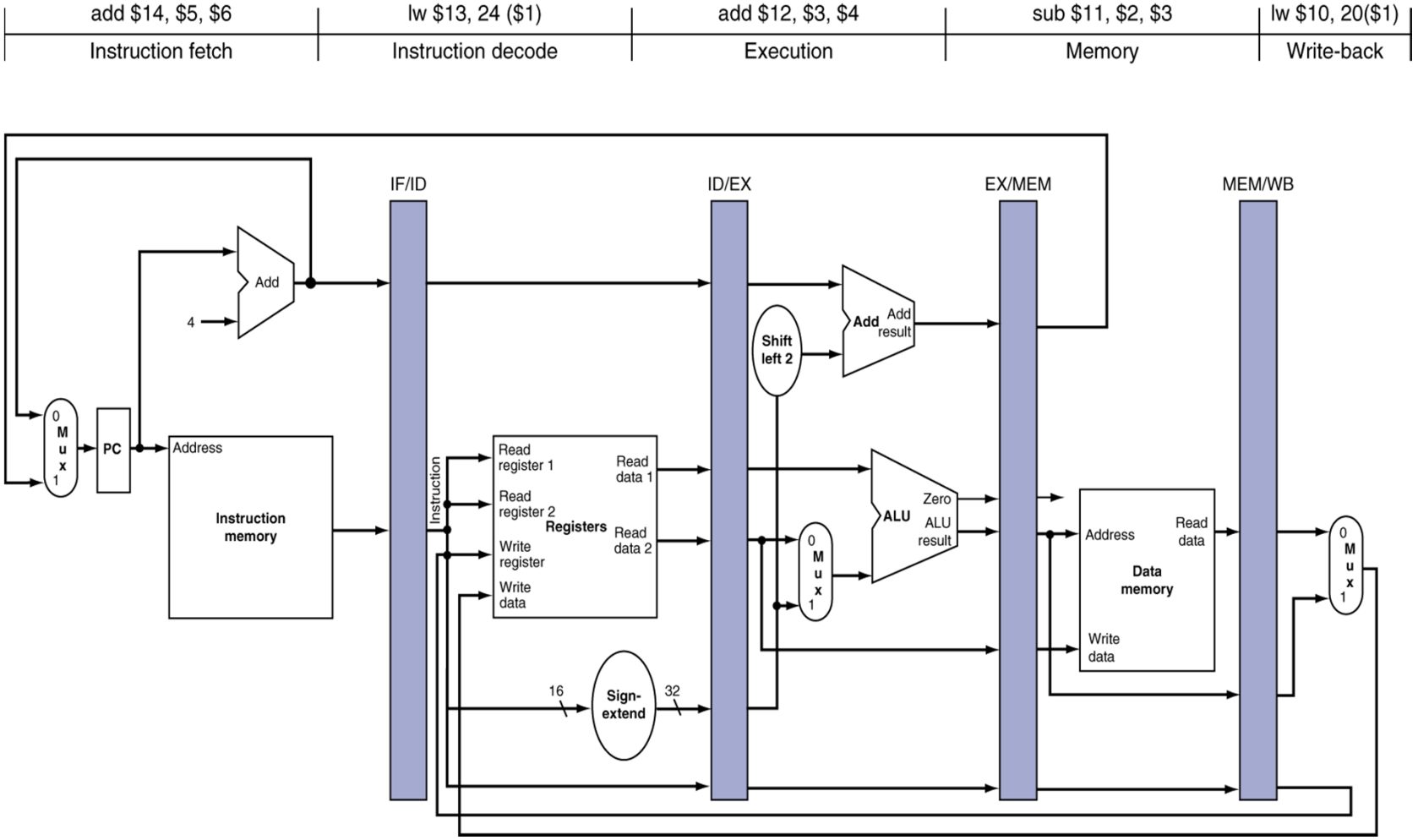
Multi-Cycle Pipeline Diagram

- Traditional form



Single Clock Cycle Pipeline Diagram

- State of pipeline in a given cycle



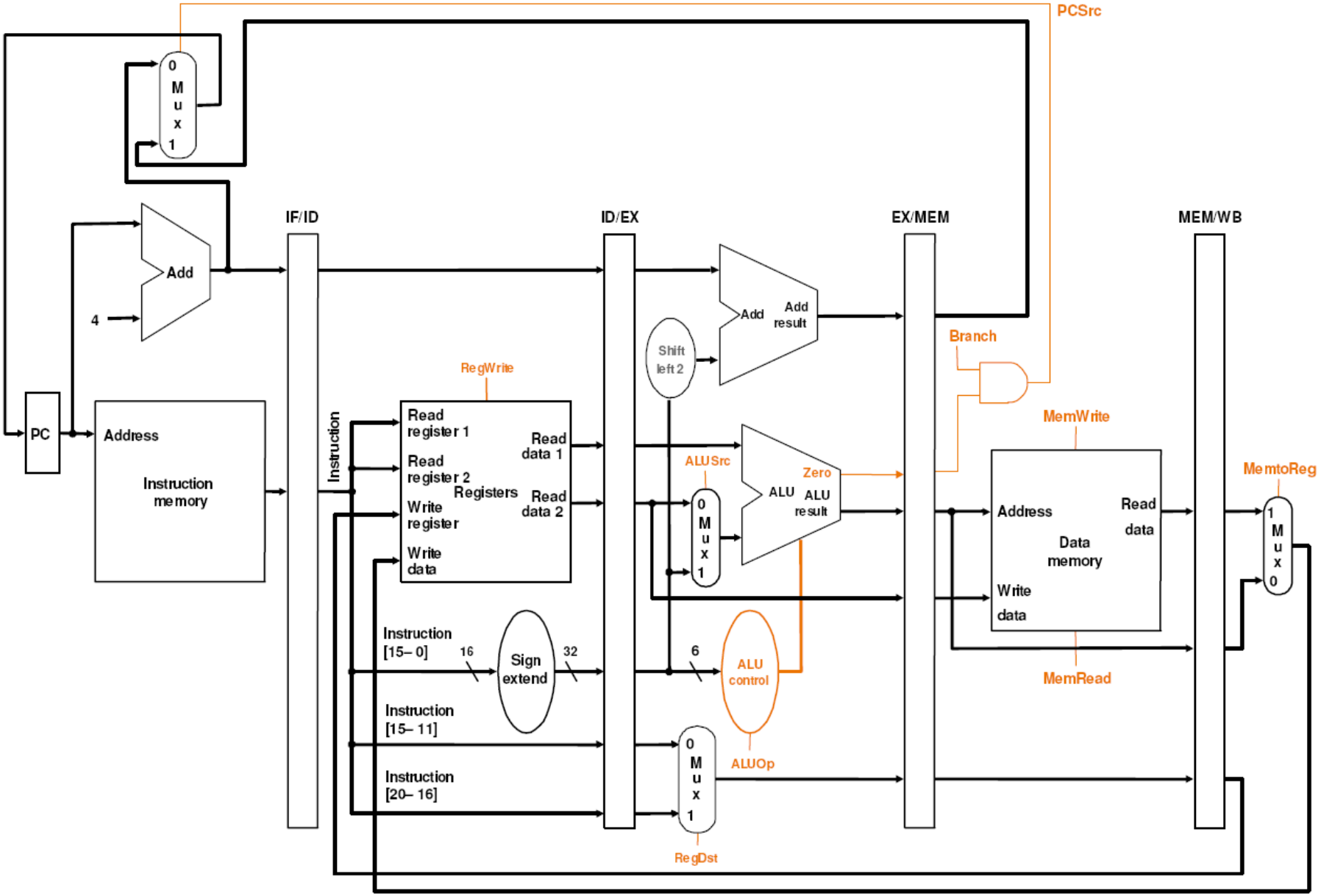
Pipelining Advantages

- Higher maximum throughput
- Higher utilization of CPU resources
- But , more hardware needed, perhaps complex control
- Increases the per instruction execution time of each instruction due to overhead in the control of the pipeline
  - Reduces performance because of Imbalance among pipeline stages & from pipelining overhead
    - Pipeline overhead arises from the combination of pipeline register delay and clock skew

Pipeline Controlz

- FSM isn't really appropriate
- Combinational logic (like single cycle design)
  - Signals generated once in ID stage
  - Follow instruction through the pipeline
  - Get used in whatever stage they are needed

Adding Pipeline Control Points



Pipeline Controlz..

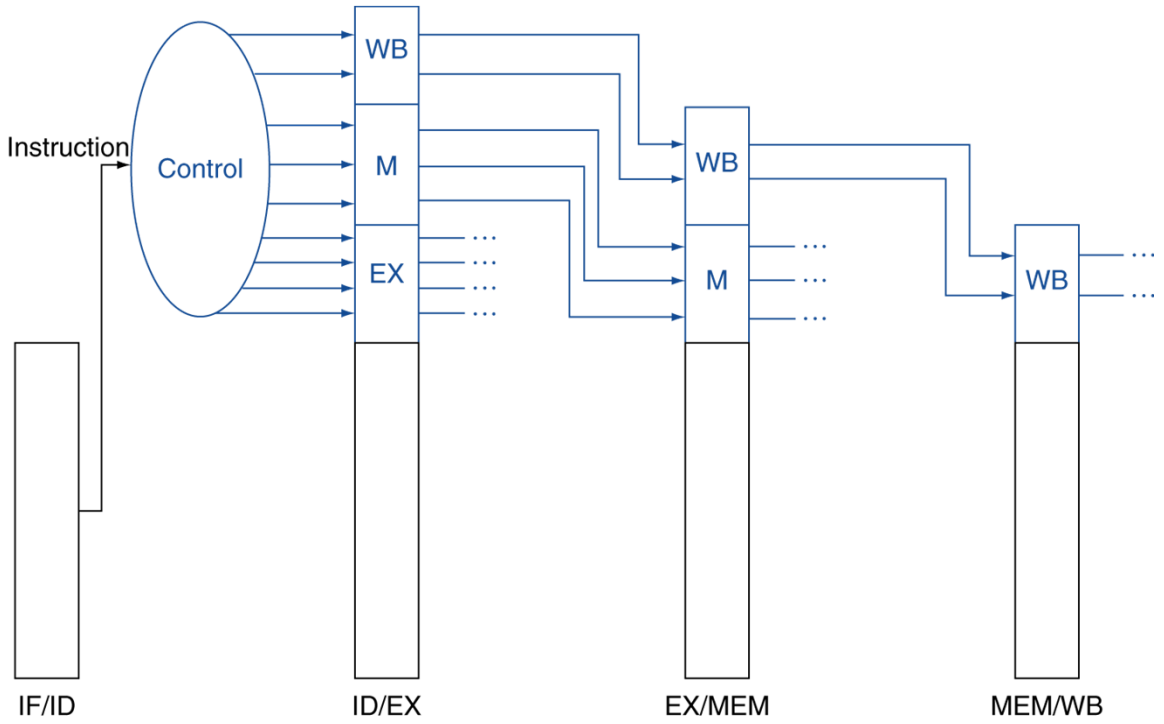
Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X

Pipeline Controlz …

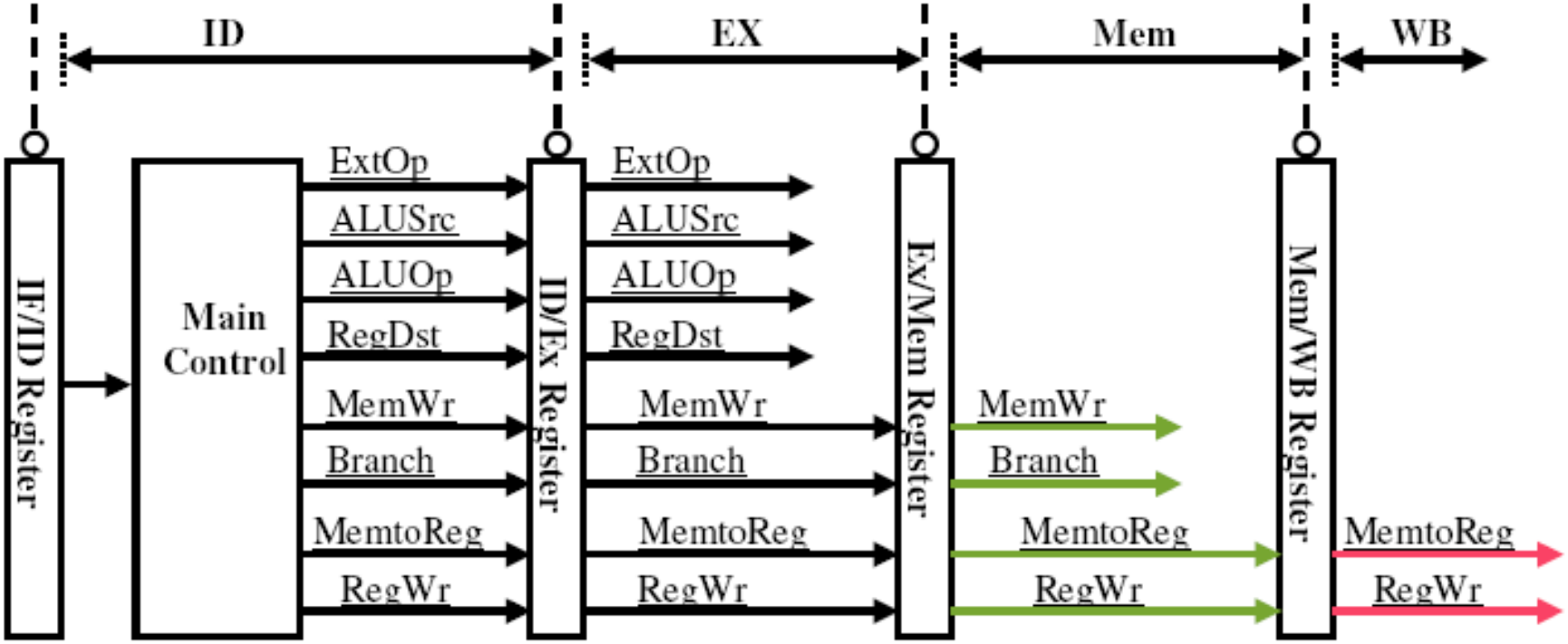
- The main control generates the control signals during ID:
  - Instruction Fetch (IF)
    - Signal to read instruction memory and to write the PC
  - Instruction Decode / Register File Read (ID)
    - No optional control signal to set
  - Execution / address calculation
    - RegDst, ALUOp and ALUSrc
  - Memory Access
    - Branch, MemRead, MemWrite (set by branch equal, load & store)
  - Write Back
    - MemtoReg (decides between sending the ALU result or the memory value to the register file) and RegWrite (writes the chosen value)

Pipelined Control

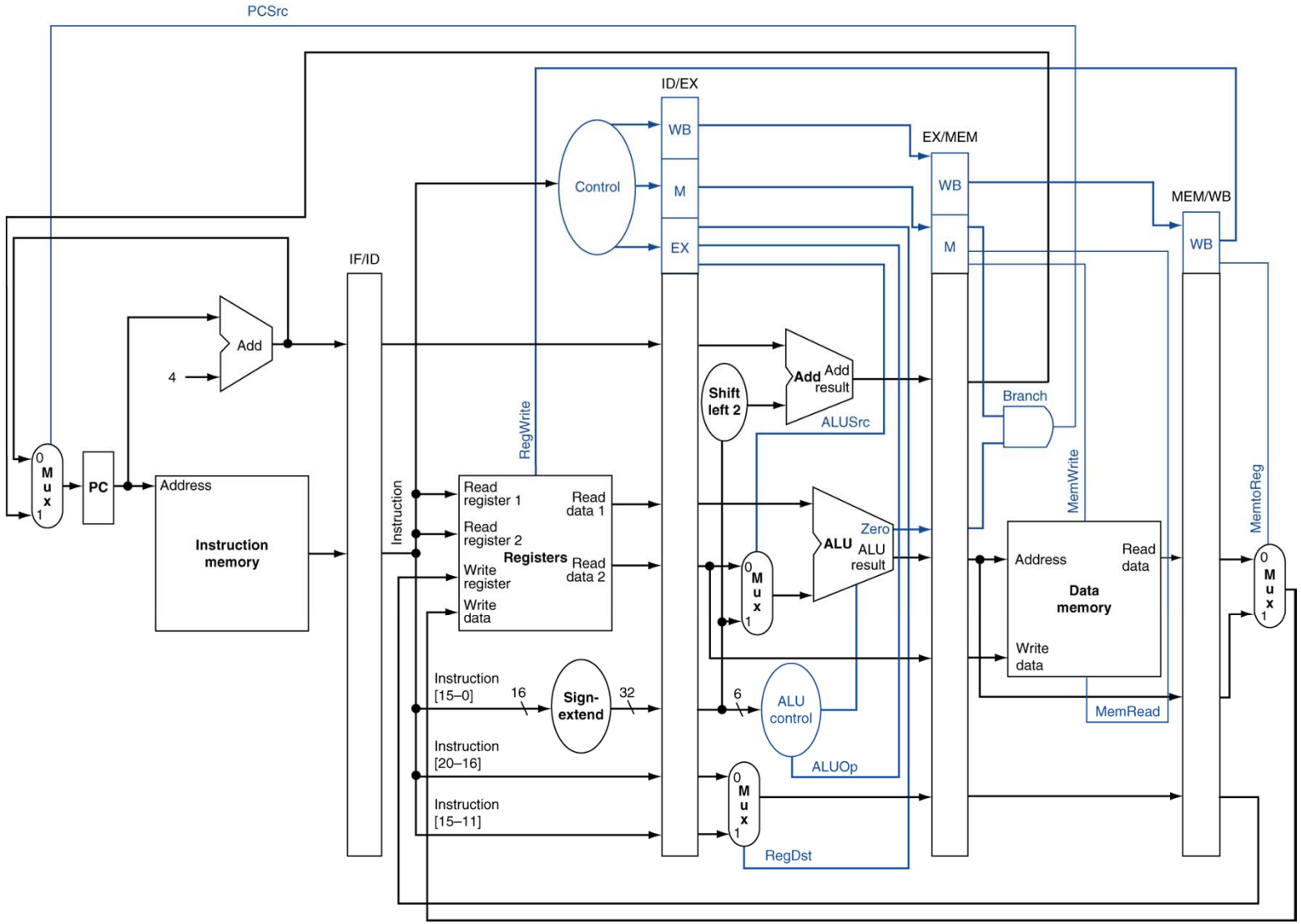
- Control signals derived from instruction
  - As in single-cycle implementation



Pipelined Control



Pipelined Control



Performance

- Pipelining increases the CPU instruction throughput (pipeline throughput)
- Ideally CPI = 1
- Pipelining does not reduce the execution time of an instruction (pipeline latency)
- In fact, it slightly increases the execution time due to the increased control overhead of the pipeline stage register delays

Example

- For a non-pipelined machine:
  - Clock cycle = 10ns
  - 4 cycles for ALU operations and branches, 5 cycles for memory operations with instruction frequencies of 40%, 20% and 40% respectively
  - If pipelining adds 1ns to the machine clock cycle then the speedup in instruction execution from pipelining is:
  - Non-pipelined Average instruction execution time = Clock cycle * Avg CPI = 10ns * [(40%+20%)*4 + 40% * 5]

    = 10ns * 4.4 = 44ns
  - In the pipelined implementation 5 stages are used with average instruction execution time of 10ns + 1ns = 11ns
  - Speedup from pipelining = (instruction execution time non-pipelined / instruction execution time pipelined) = 44/11 = 4