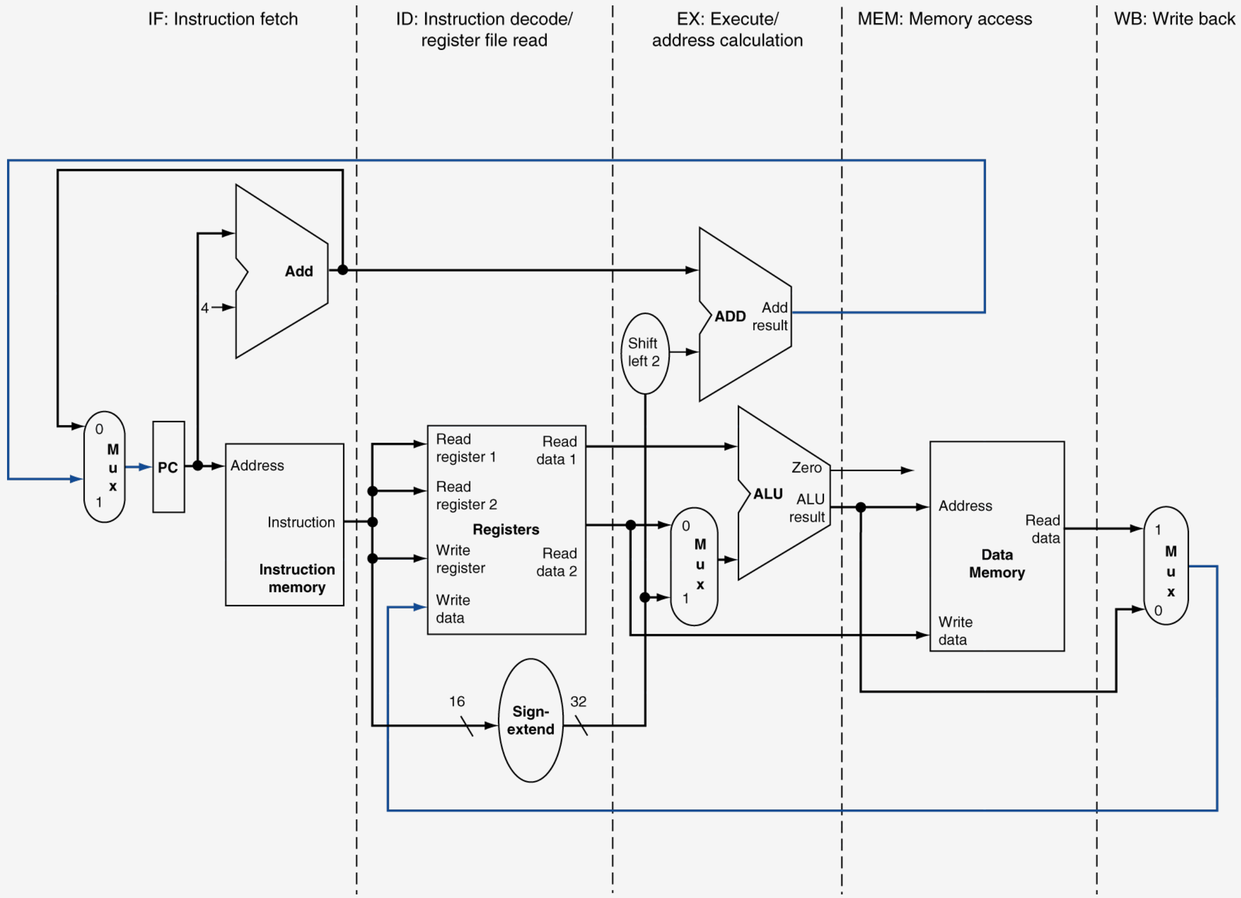
# ECE260: Fundamentals of Computer Engineering

## Pipelined Datapath and Control

James Moscola
Dept. of Engineering & Computer Science
York College of Pennsylvania

# MIPS Pipelined Datapath

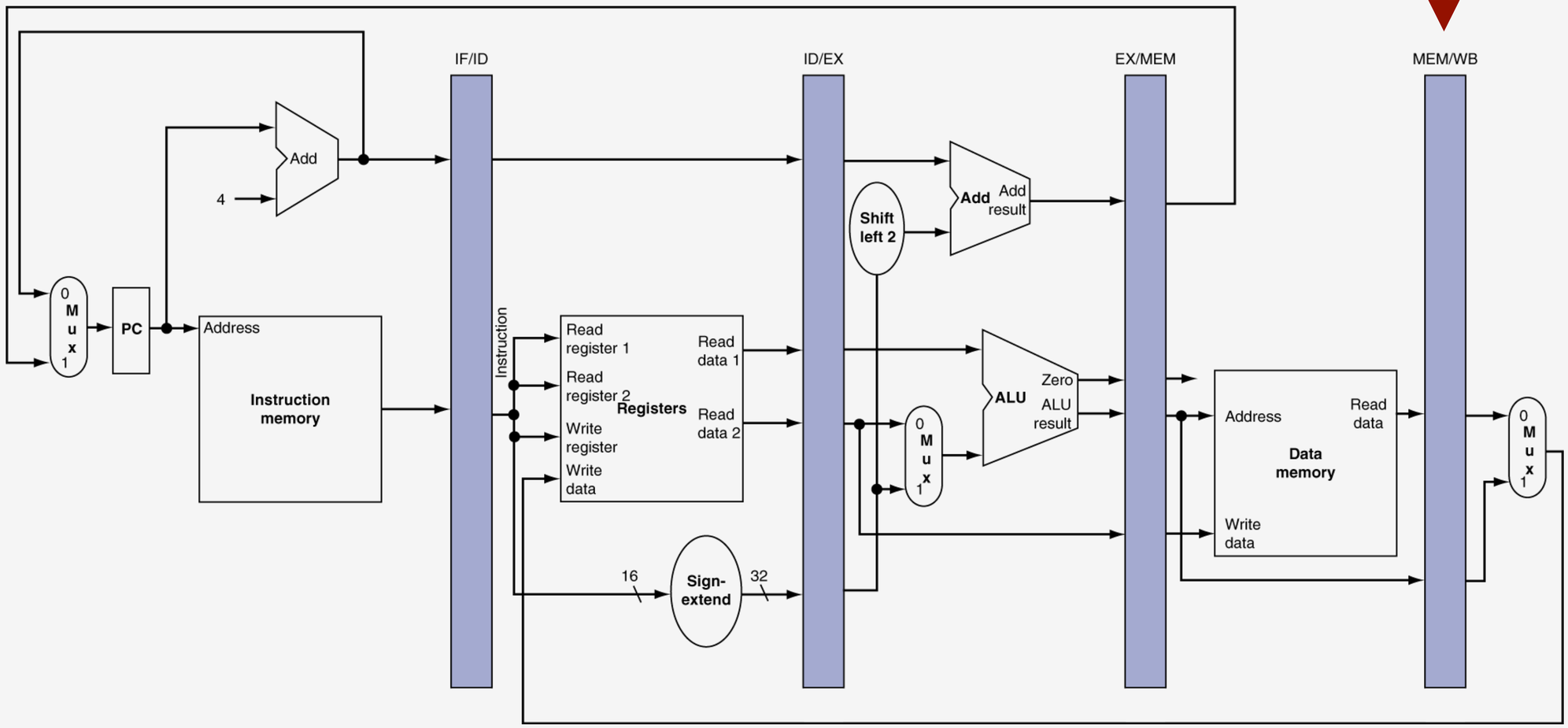
- Single-cycle MIPS datapath can be divided into the classic 5-stage pipeline
  - ***IF:*** Instruction Fetch
  - ***ID:*** Instruction Decode and Register Read
  - ***EX:*** Execution (or address calculation)
  - ***MEM:*** Data Memory Access
  - ***WB:*** Write Back
- Divide single-cycle datapath to reduce the amount of combinational logic between sequential elements
  - Place registers between each stage




**Single-cycle datapath with dashed lines to indicate where stages will be separated**

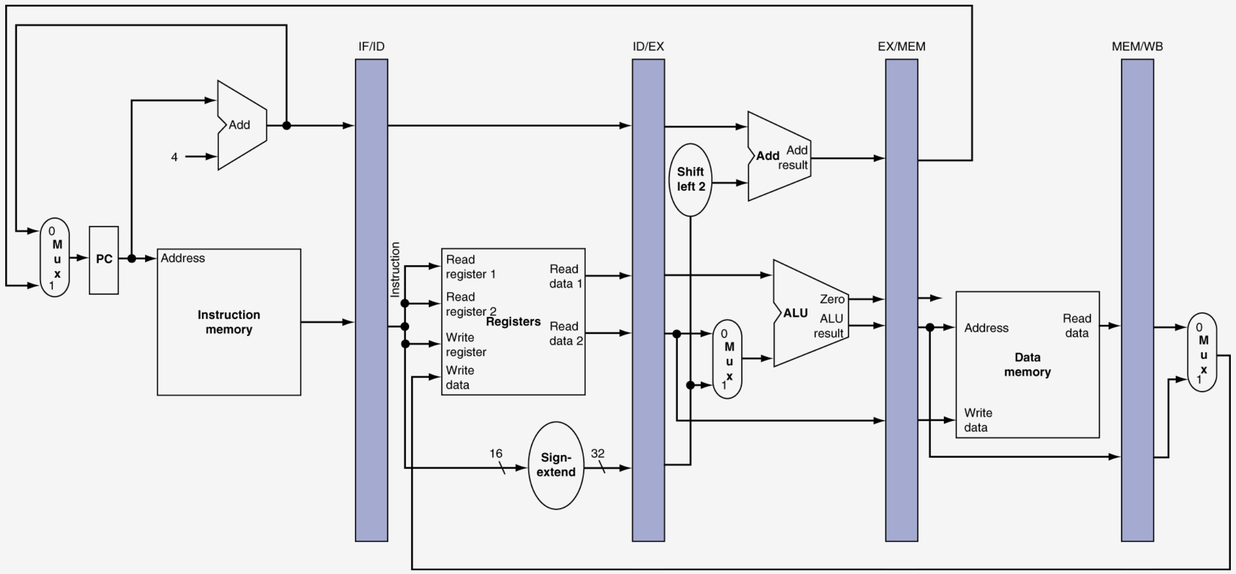
# Pipeline Registers

**Registers are named for the two pipeline stages that are separated by the register**



# Pipeline Registers

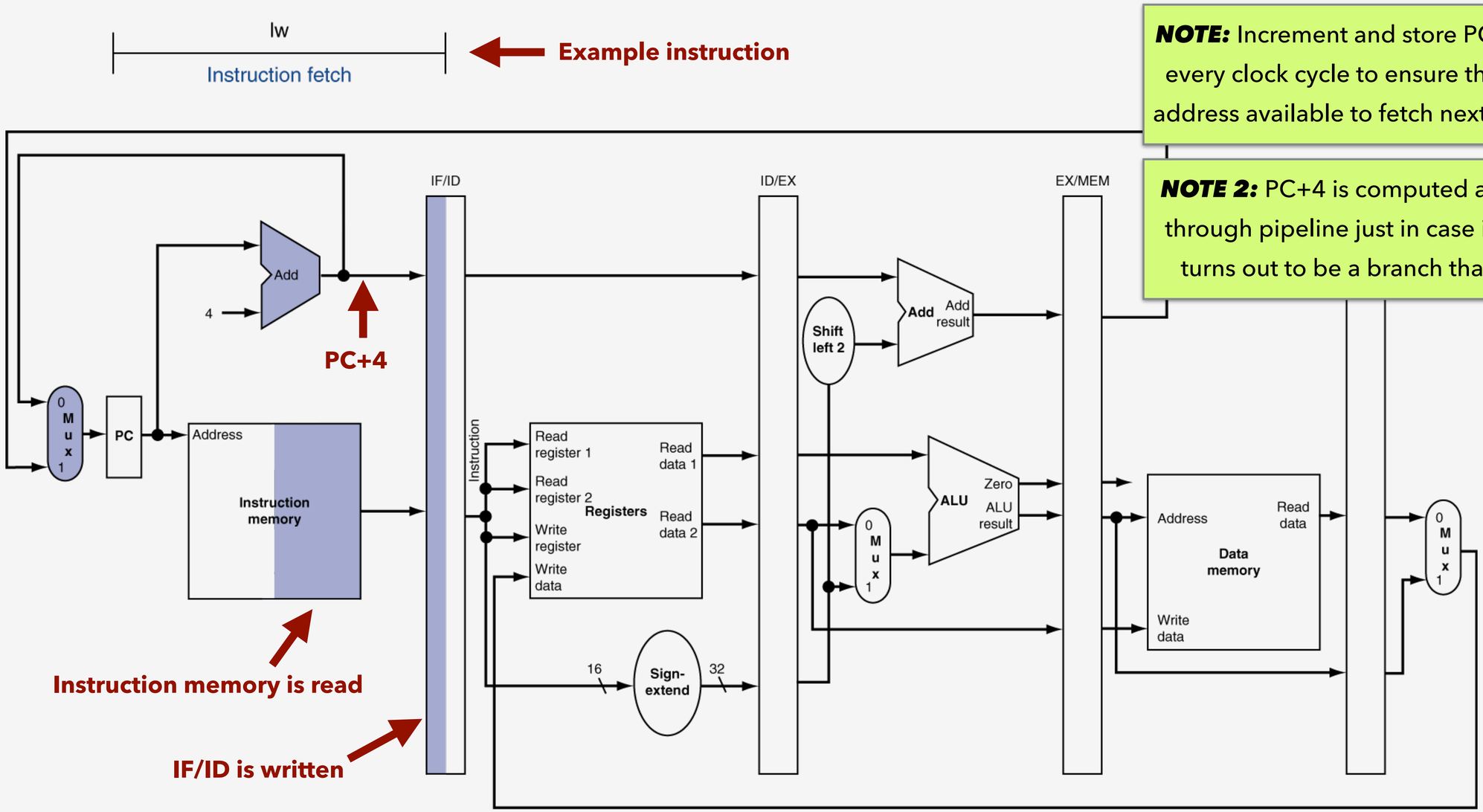
- Registers divide datapath into smaller (faster) chunks of combinational logic
  - Decreases latency between stages and supports higher clock rates
- Output of a pipeline stage is passed to next stage through pipeline registers
  - Stored into register at rising clock edge
  - Available in next stage for remainder of clock cycle
- Pipeline registers vary in size
  - Example: IF/ID register is 64 bits:
    32 bits for instruction + 32 bits for PC+4



# MIPS Pipelined Datapath

- ***Recall:*** for ***all instructions***, the first two steps are the same:
  - Use the Program Counter (PC) to access program memory and fetch an instruction
  - Read the source registers (one or two) to be used for the instruction – encoded into instruction
- During IF and ID pipeline stages, the instruction is still being fetched/decoded so no way to perform instruction-specific control
- Control during EX, MEM, and WB pipeline stages varies depending on instruction
  - Example: No need to write data memory while performing an R-Type instruction

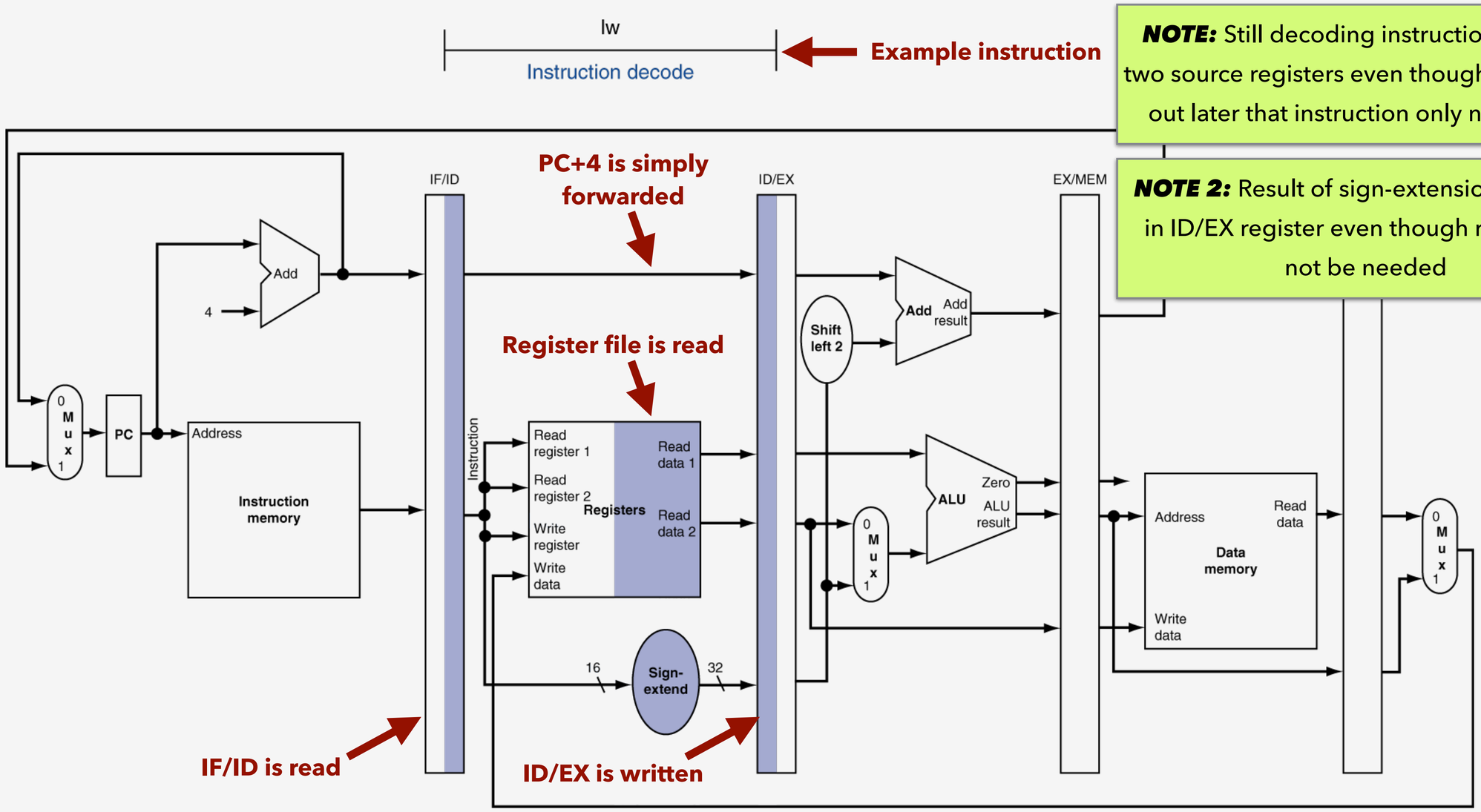
# IF Pipeline Stage (Same for **All** Instructions)




***NOTE:*** Increment and store PC=PC+4 on every clock cycle to ensure that new PC address available to fetch next instruction

***NOTE 2:*** PC+4 is computed and passed through pipeline just in case instruction turns out to be a branch that is taken

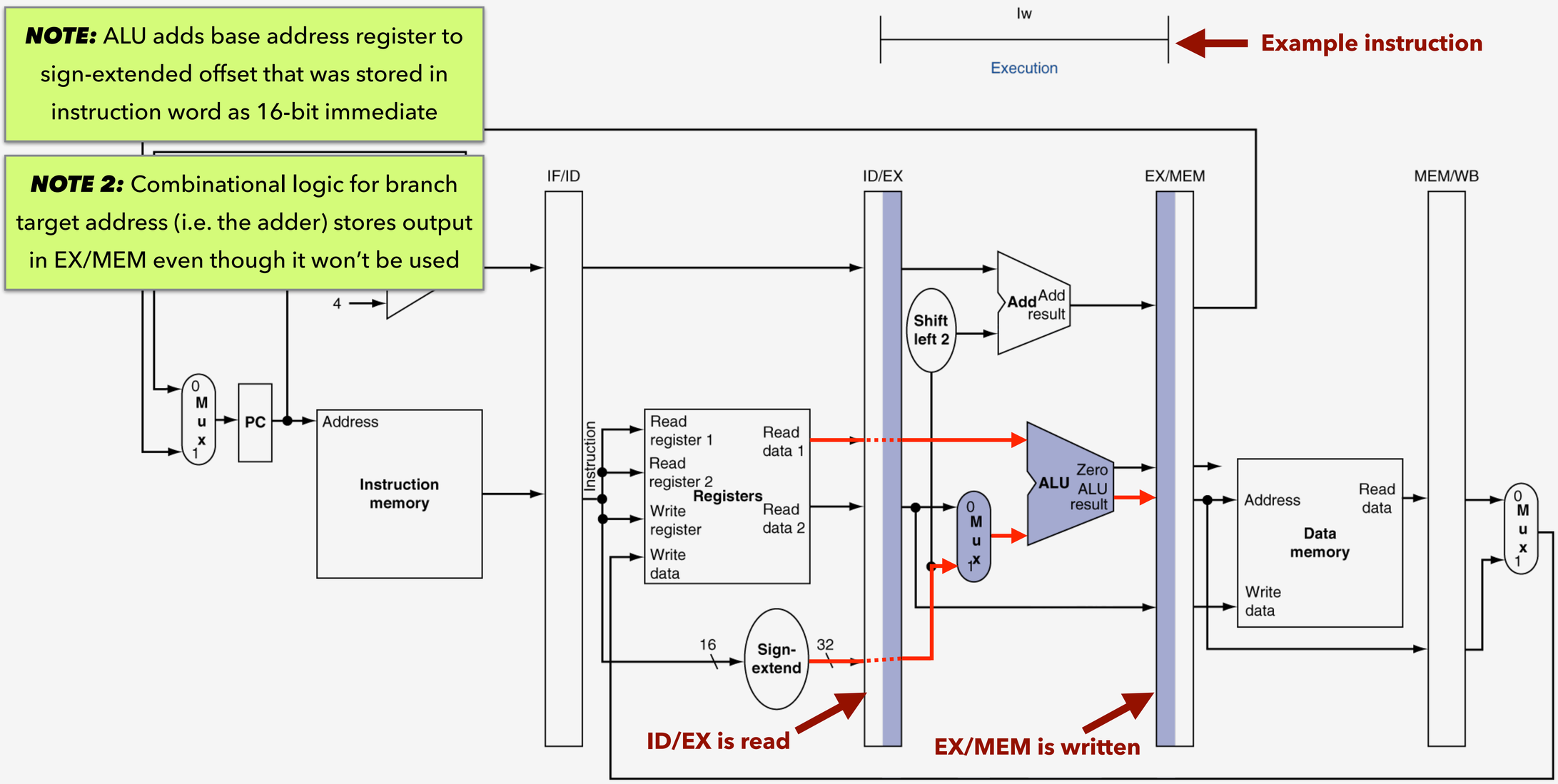
# ID Pipeline Stage (Same for **All** Instructions)



# EX Pipeline Stage for **Load** Instruction

***NOTE:*** ALU adds base address register to sign-extended offset that was stored in instruction word as 16-bit immediate

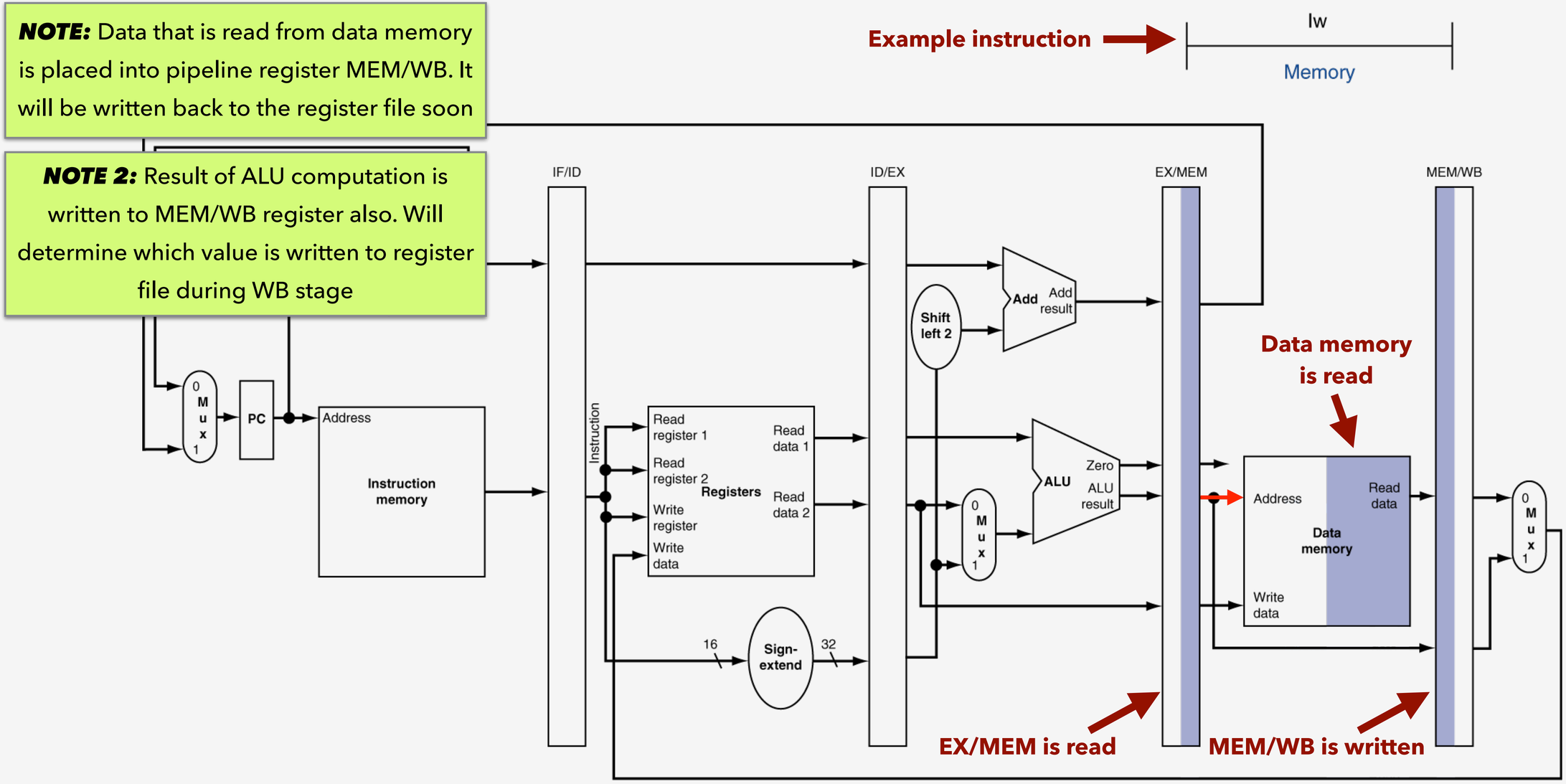
***NOTE 2:*** Combinational logic for branch target address (i.e. the adder) stores output in EX/MEM even though it won't be used



# MEM Pipeline Stage for **Load** Instruction

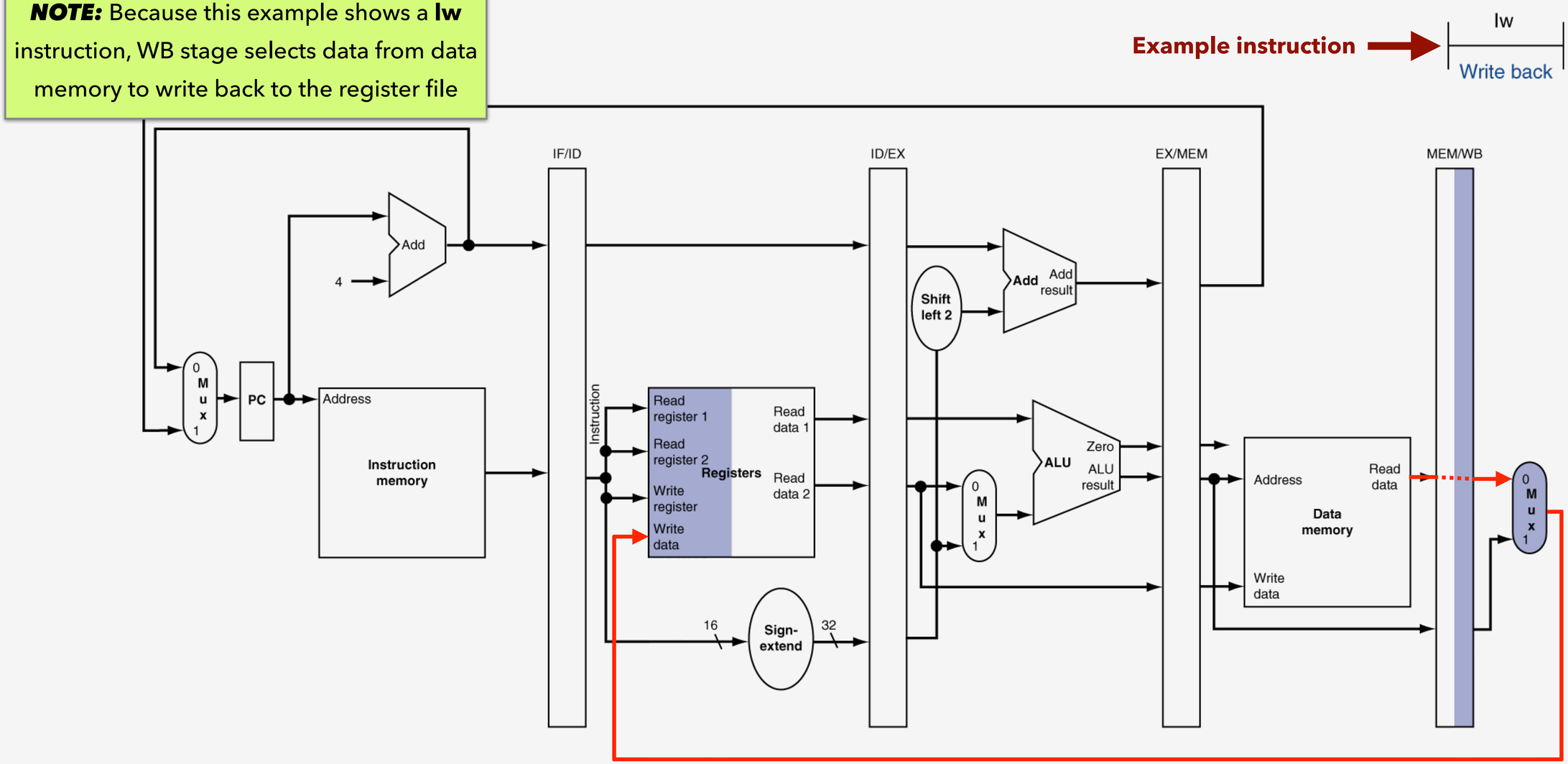
***NOTE:*** Data that is read from data memory is placed into pipeline register MEM/WB. It will be written back to the register file soon

***NOTE 2:*** Result of ALU computation is written to MEM/WB register also. Will determine which value is written to register file during WB stage



# WB Pipeline Stage for **Load** Instruction (sort of)

***NOTE:*** Because this example shows a **lw** instruction, WB stage selects data from data memory to write back to the register file

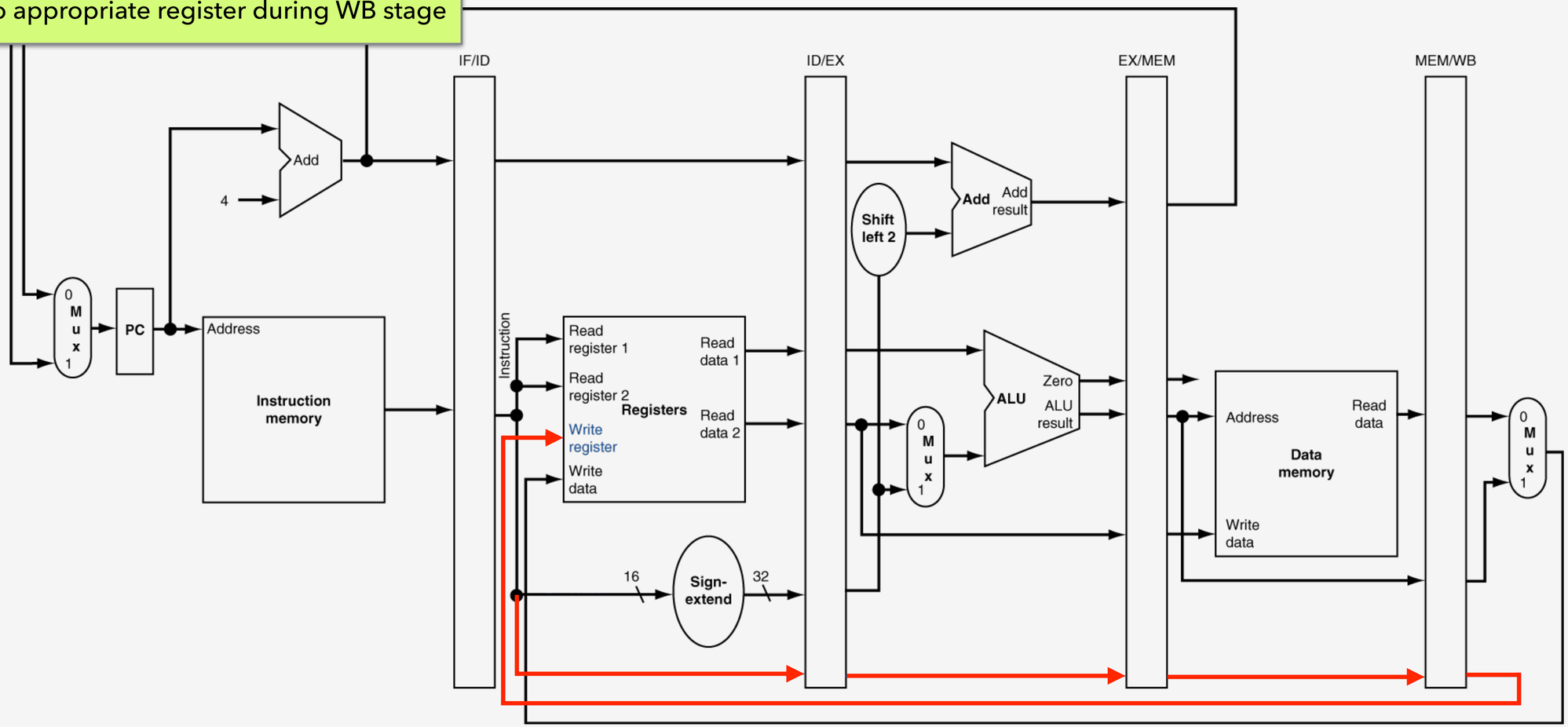


# Correcting the Single-Cycle Datapath for Pipelining

- Thus far, single-cycle datapath and pipeline pictures have shown that "Write Register" input of register file is determined directly from instruction
  - ***Problem:*** by the time an instruction reaches the WB stage, a new instruction is in the ID stage
    - The new instruction will have a destination register of it's own
    - Want to write the result for the original instruction into the destination register that was specified in the original instruction
  - ***Solution:*** must carry the "Write Register" information through the pipeline with an instruction and use that delayed information to select the write register in the register file
    - The register file "Read Register" inputs and "Read Data" outputs are determined by the instruction currently in the ID stage
    - The register file "Write Register" and "Write Data" inputs are determined by the instruction currently in the WB stage

# Corrected Datapath for Writing Register File

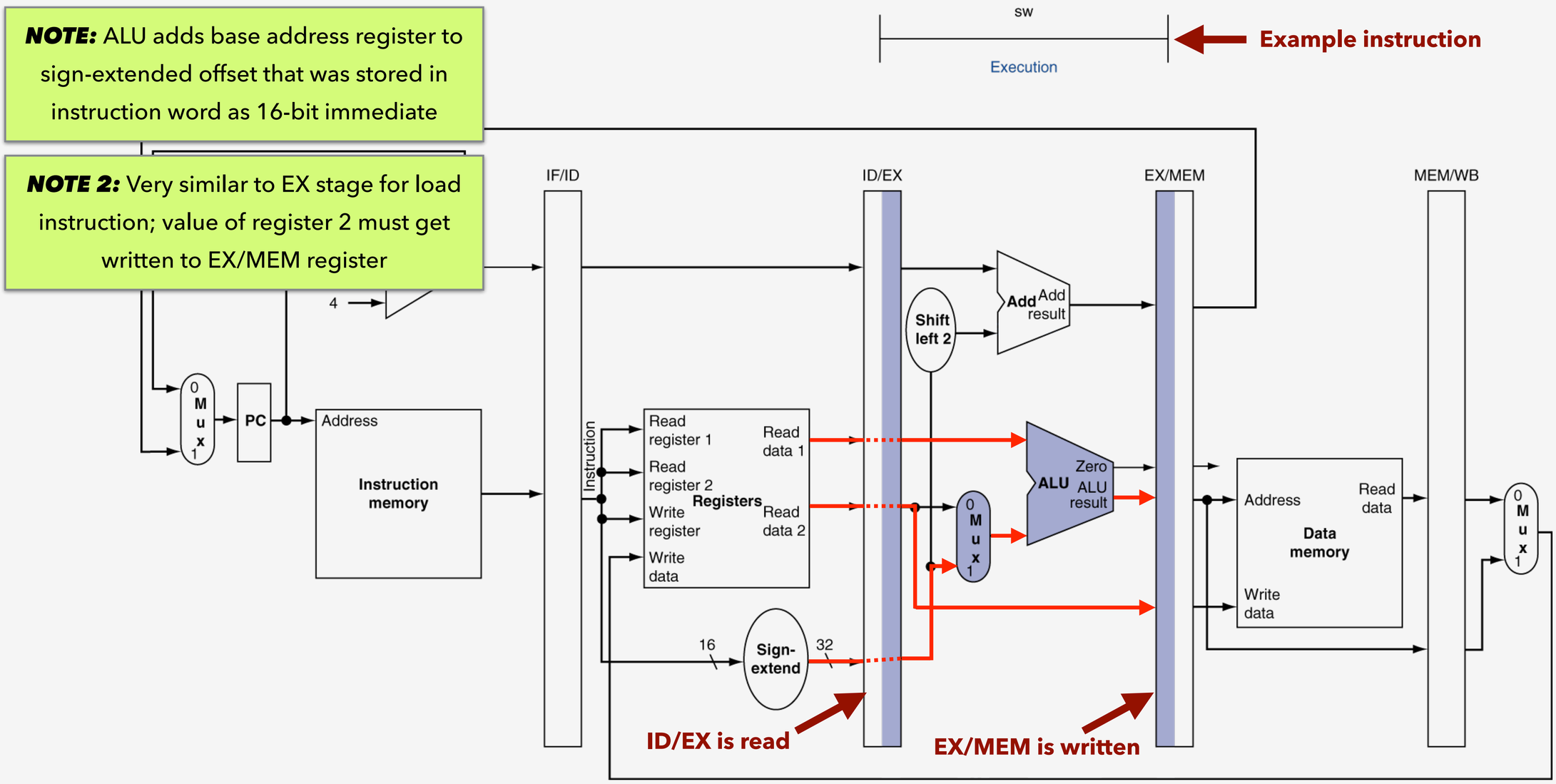
***NOTE:*** "Write Register" information must be delayed through pipeline with the rest of the instruction so result (i.e. "Write Data") can be written into appropriate register during WB stage



# EX Pipeline Stage for **Store** Instruction

***NOTE:*** ALU adds base address register to sign-extended offset that was stored in instruction word as 16-bit immediate

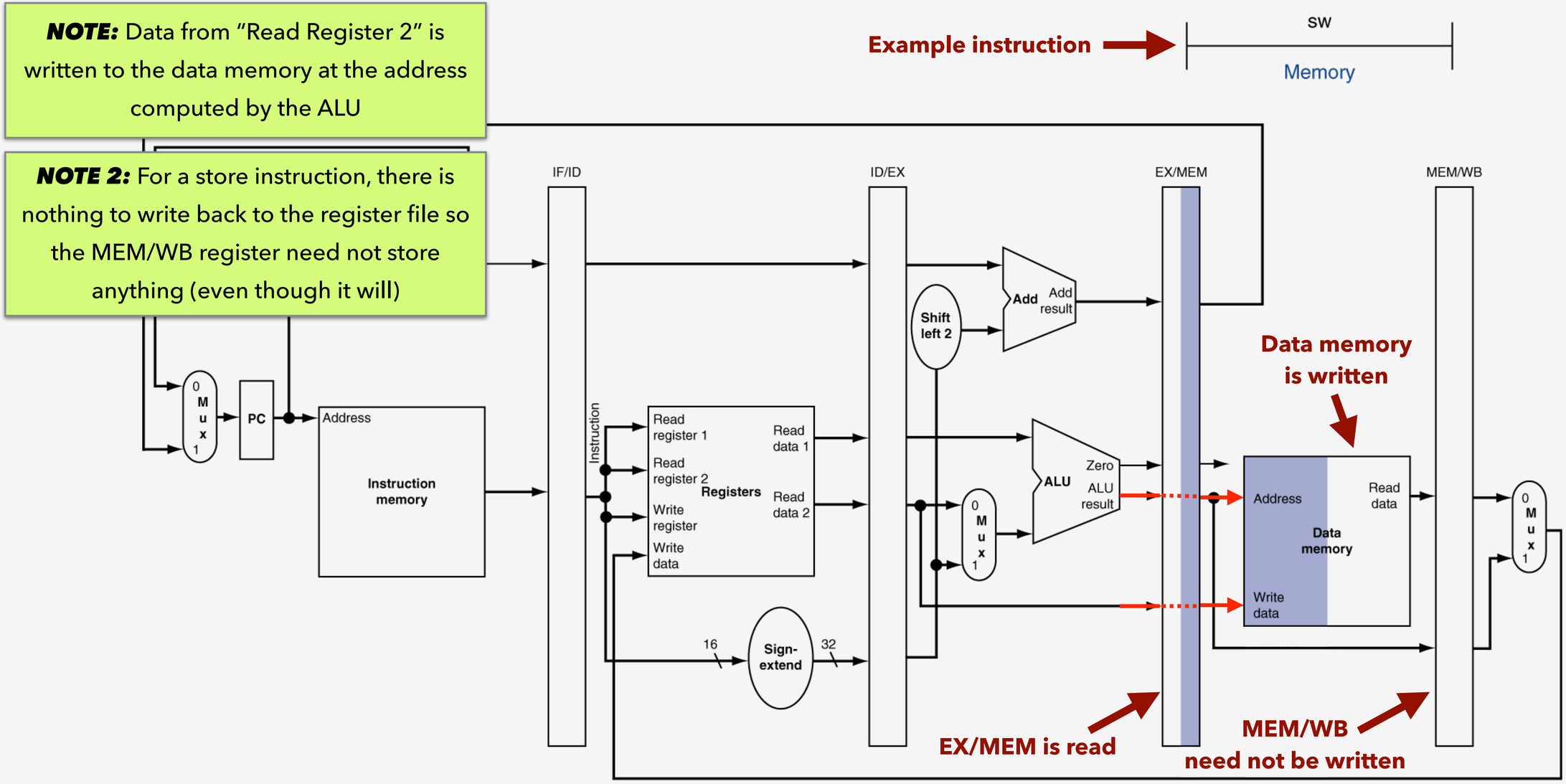
***NOTE 2:*** Very similar to EX stage for load instruction; value of register 2 must get written to EX/MEM register



# MEM Pipeline Stage for **Store** Instruction

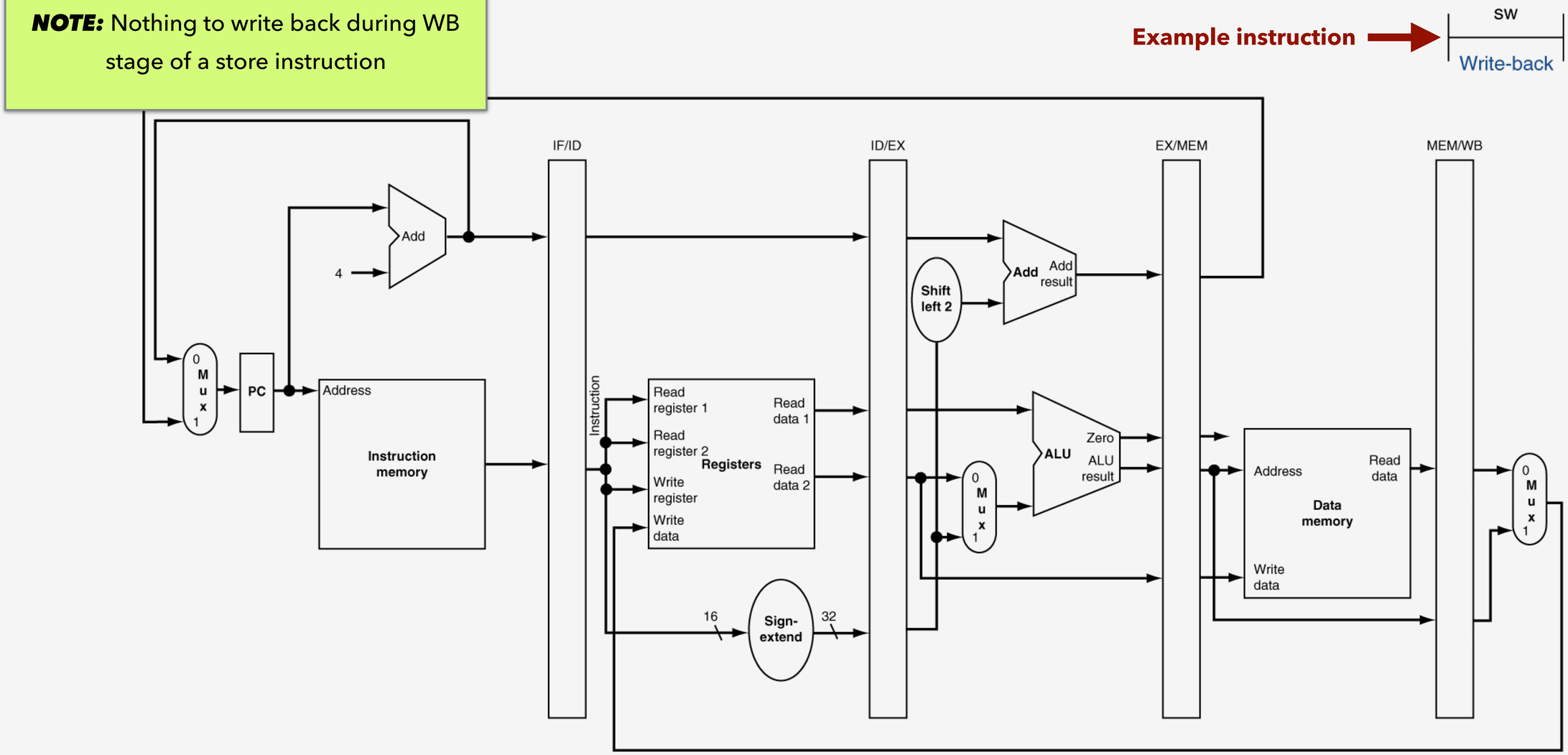
***NOTE:*** Data from "Read Register 2" is written to the data memory at the address computed by the ALU

***NOTE 2:*** For a store instruction, there is nothing to write back to the register file so the MEM/WB register need not store anything (even though it will)



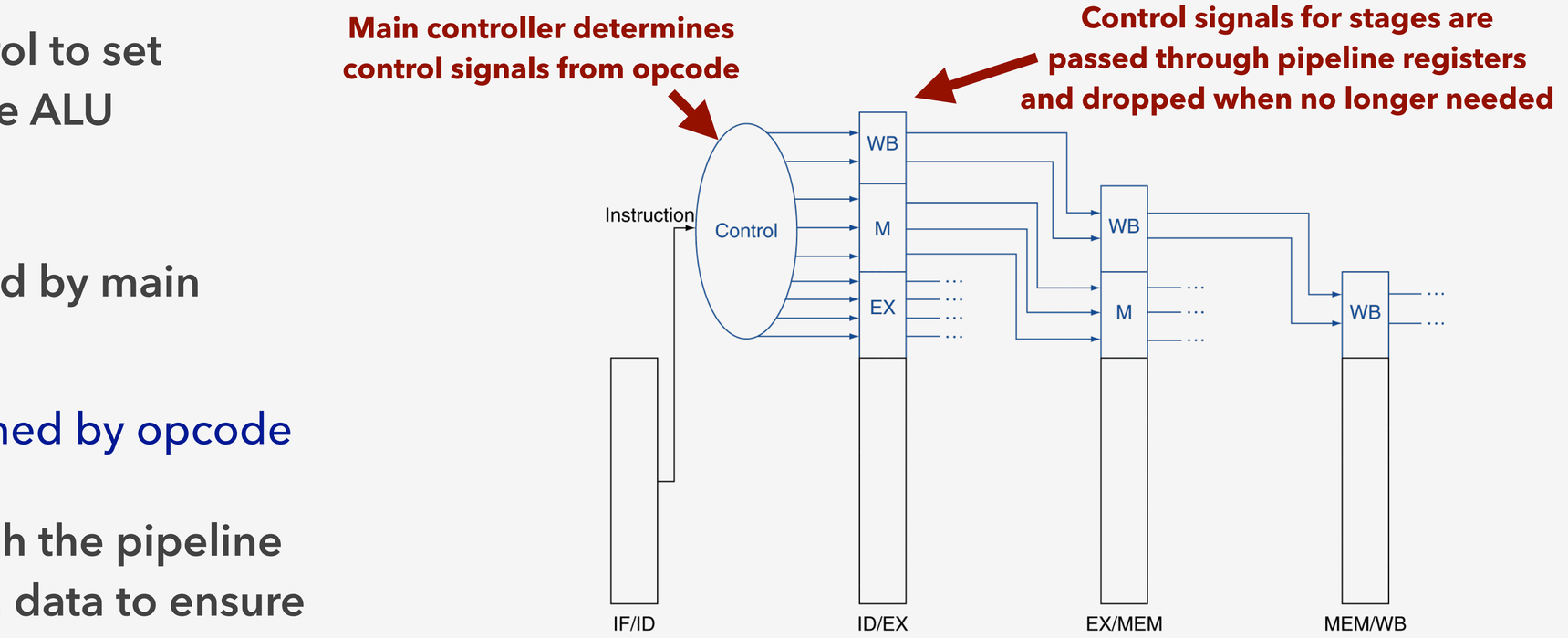
# WB Pipeline Stage for **Store** Instruction

***NOTE:*** Nothing to write back during WB stage of a store instruction



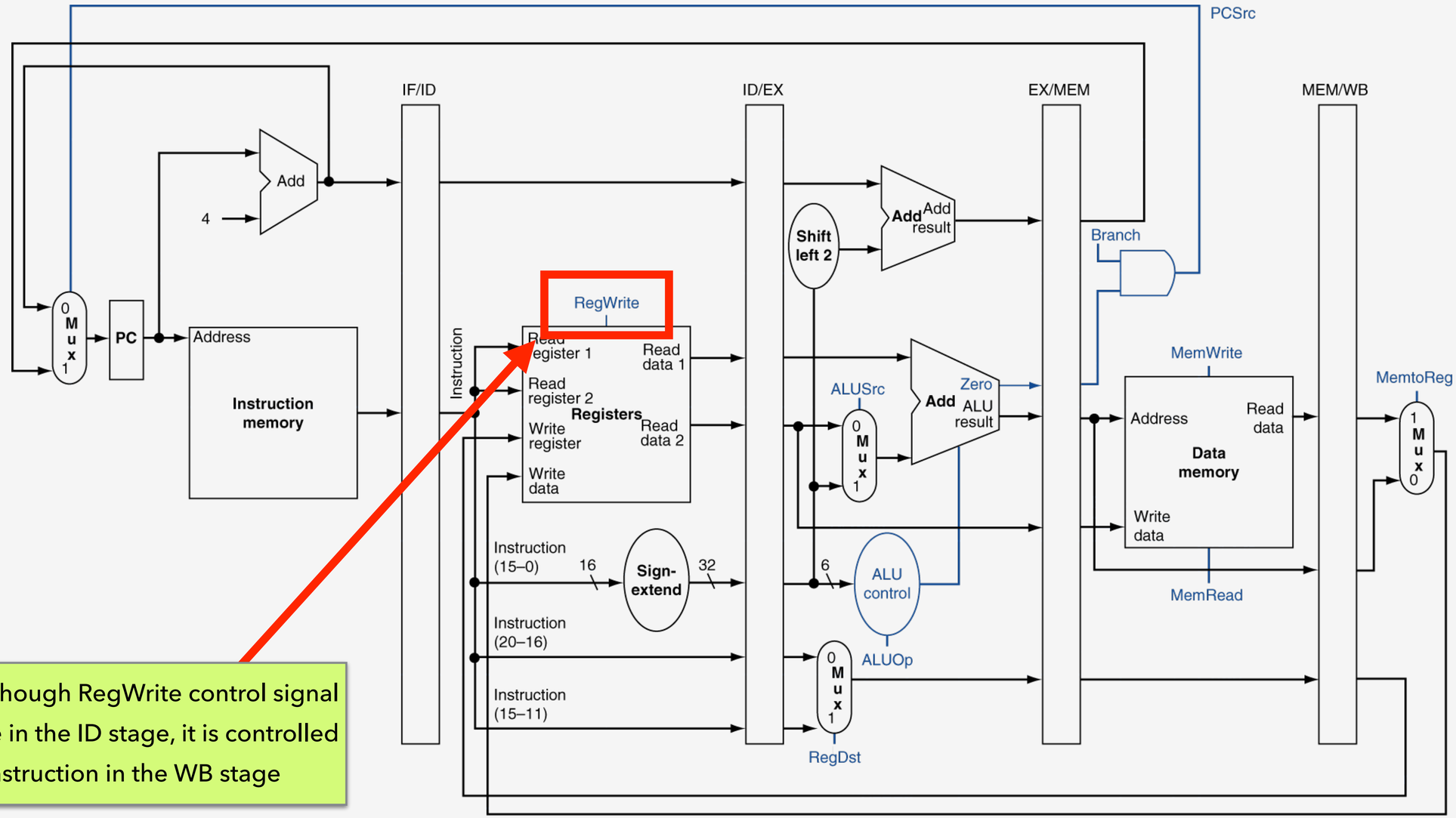
# Pipelined Control

- Each instruction needs control to set multiplexers and appropriate ALU operations
- Control signals are generated by main controller during ID stage
  - Signal values are determined by opcode
- Control signals travel through the pipeline registers with the instruction data to ensure that control is "synced" with the instruction
  - Signals are "dropped" when they are no longer needed
    - Example: don't need ALUOp bits after EX stage, so don't forward them to MEM



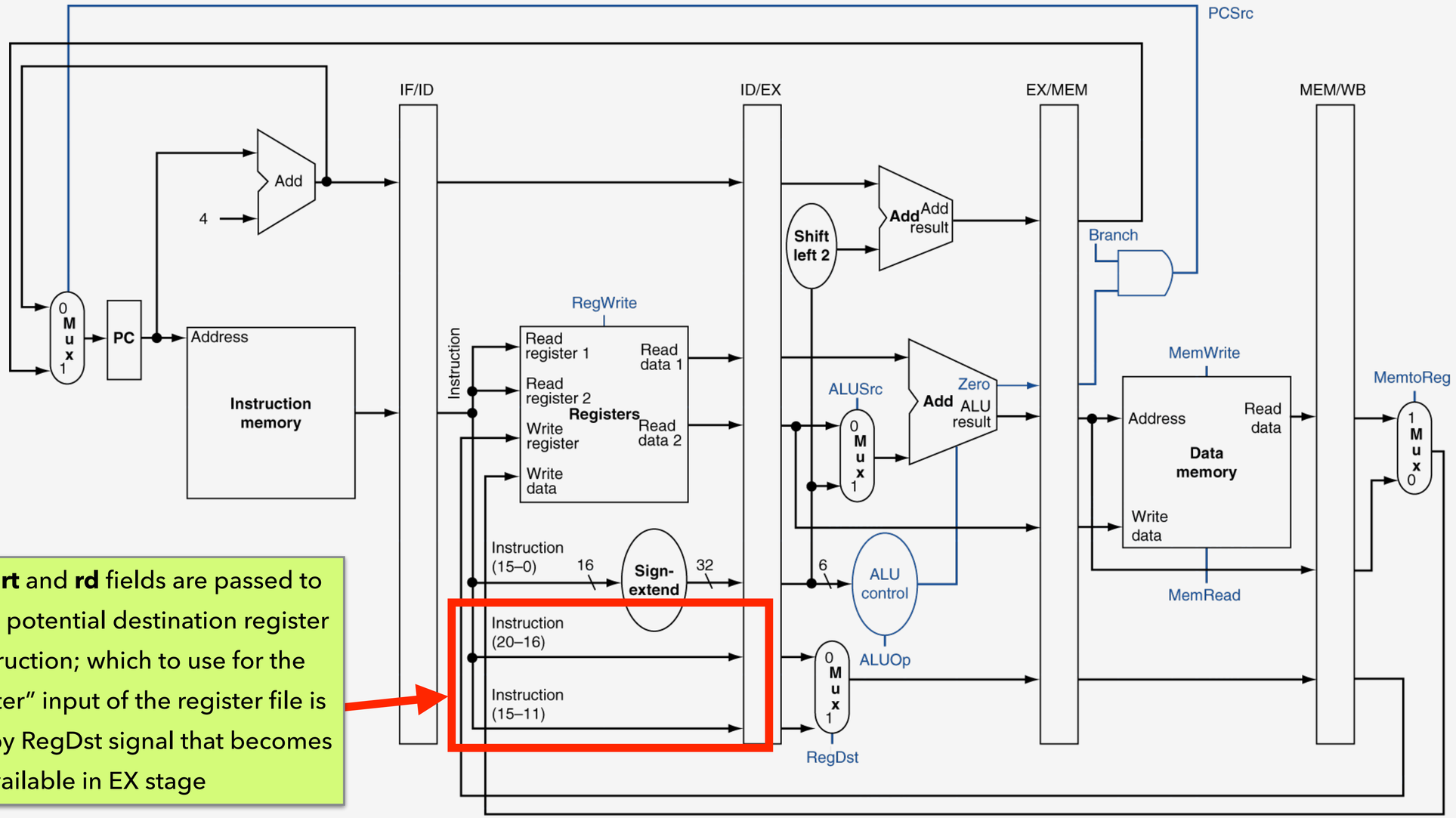

| Instruction | Execution/address calculation stage control lines | | | | Memory access stage control lines | | | Write-back stage control lines | |
|---|---|---|---|---|---|---|---|---|---|
| | RegDst | ALUOp1 | ALUOp0 | ALUSrc | Branch | Mem-Read | Mem-Write | Reg-Write | Memto-Reg |
| R-format | 1 | 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 (Simplified)



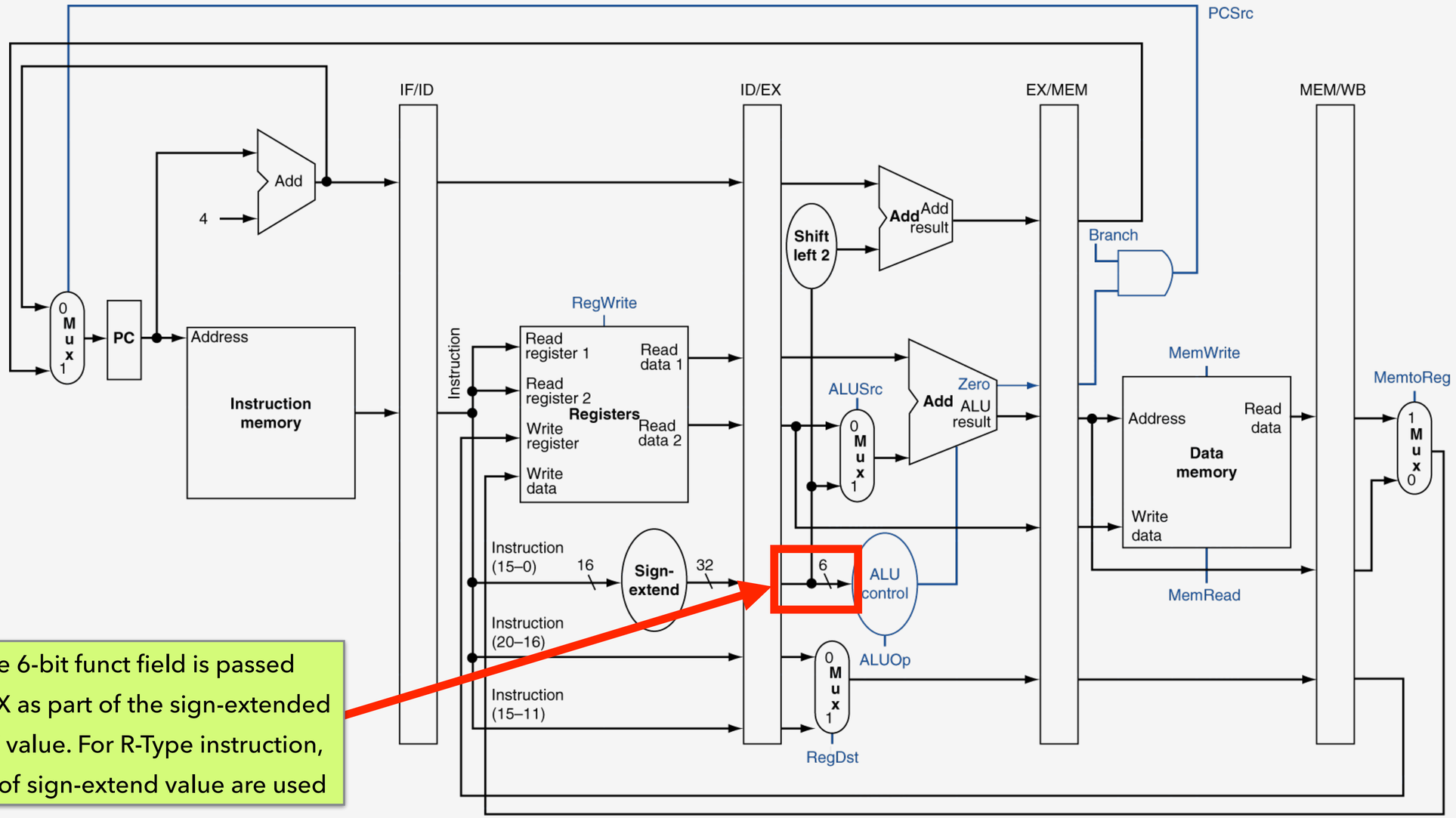

***NOTE:*** Even though RegWrite control signal appears to be in the ID stage, it is controlled by the instruction in the WB stage

# Pipelined Control (Simplified)



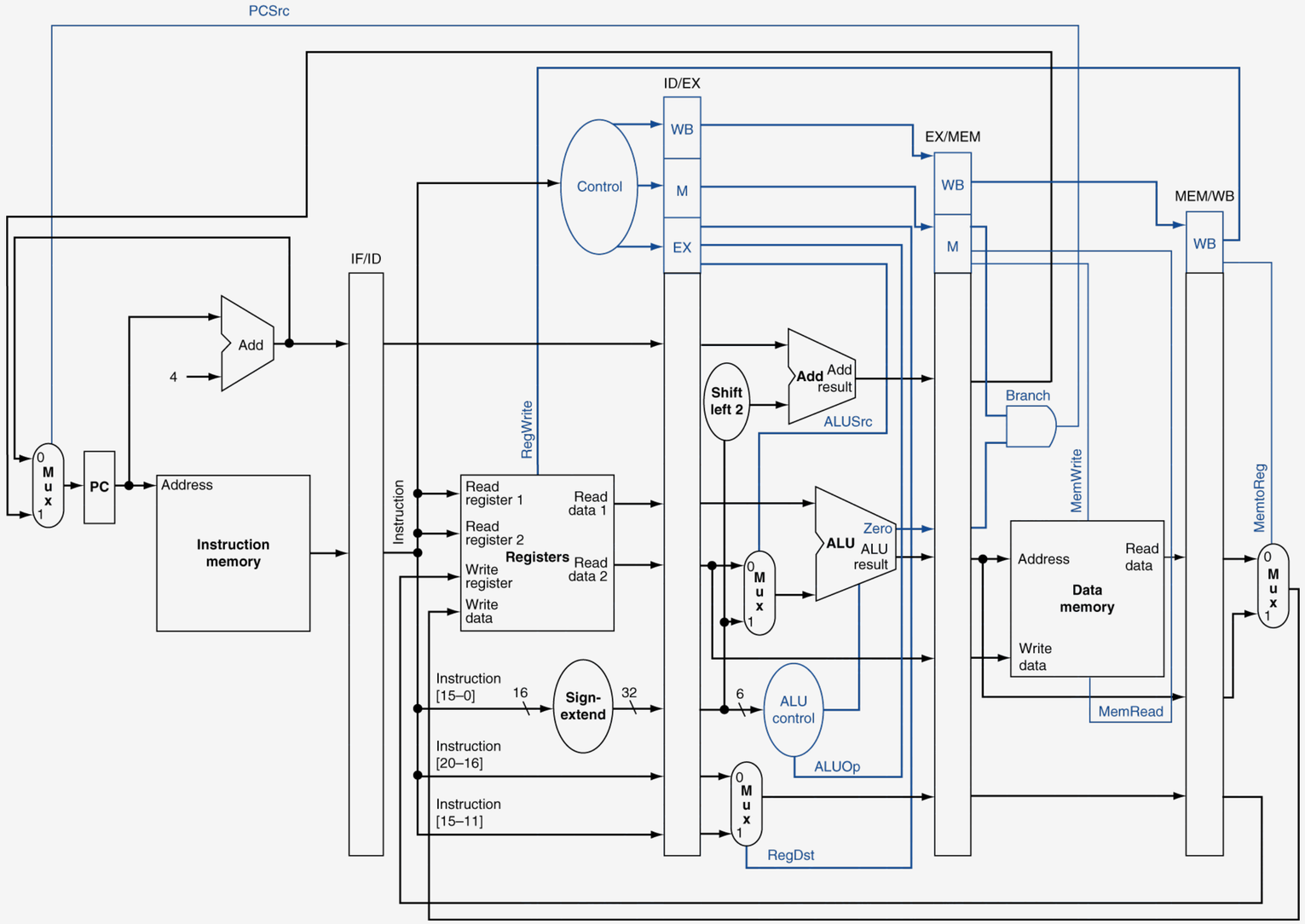

***NOTE:*** Both **rt** and **rd** fields are passed to EX stage as a potential destination register for the instruction; which to use for the "Write Register" input of the register file is determined by RegDst signal that becomes available in EX stage

# Pipelined Control (Simplified)



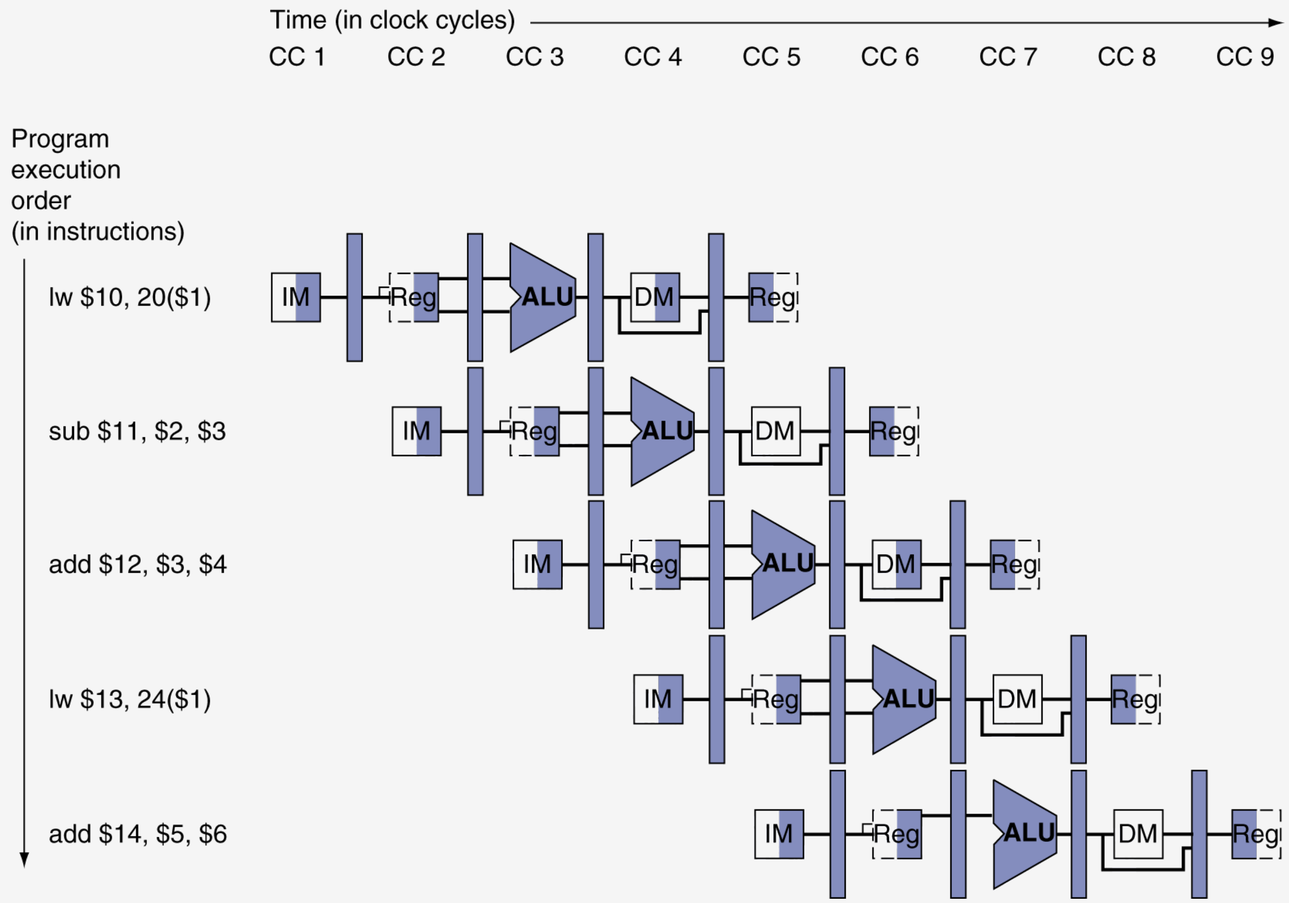

***NOTE:*** The 6-bit funct field is passed through ID/EX as part of the sign-extended "immediate" value. For R-Type instruction, only bits 5:0 of sign-extend value are used

# Pipelined Control



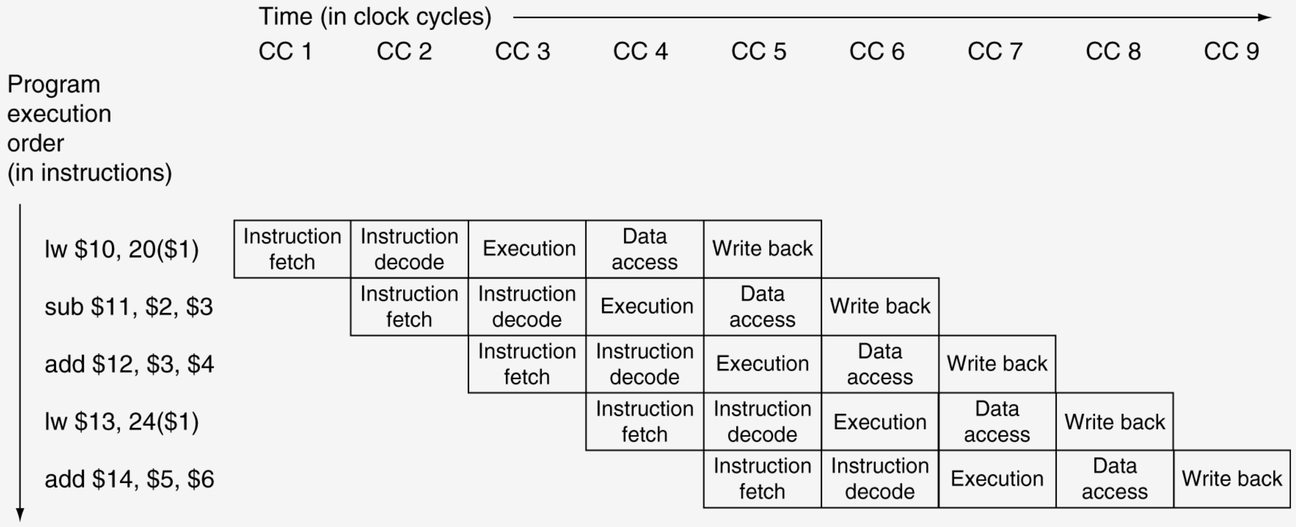
# Multi-Cycle Pipeline Diagram

- A representation of a pipeline ***over multiple clock cycles***
- Illustrates which instruction is in which pipeline stage at each clock cycle
- Useful for visualizing pipeline behavior and how instructions interact with each other
  - i.e. finding hazards



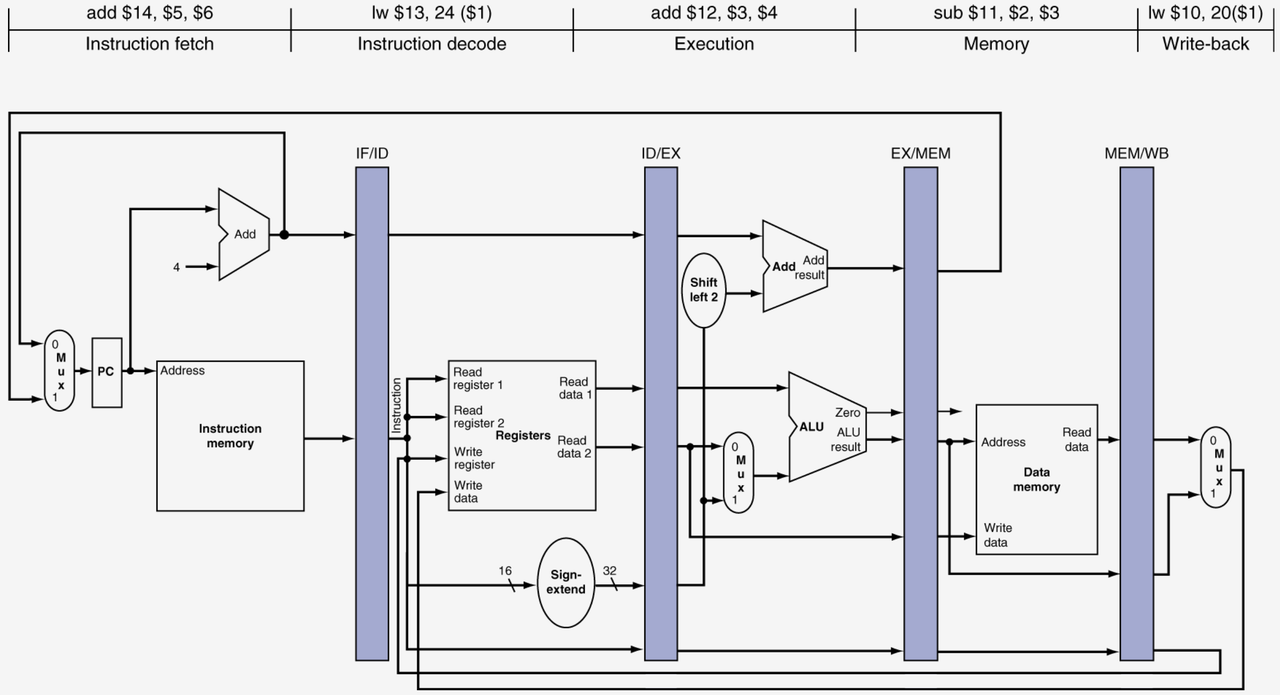
# Multi-Cycle Pipeline Diagram (less stylized version)

- A representation of a pipeline ***over multiple clock cycles***
- Illustrates which instruction is in which pipeline stage at each clock cycle
- Useful for visualizing pipeline behavior and how instructions interact with each other
  - i.e. finding hazards

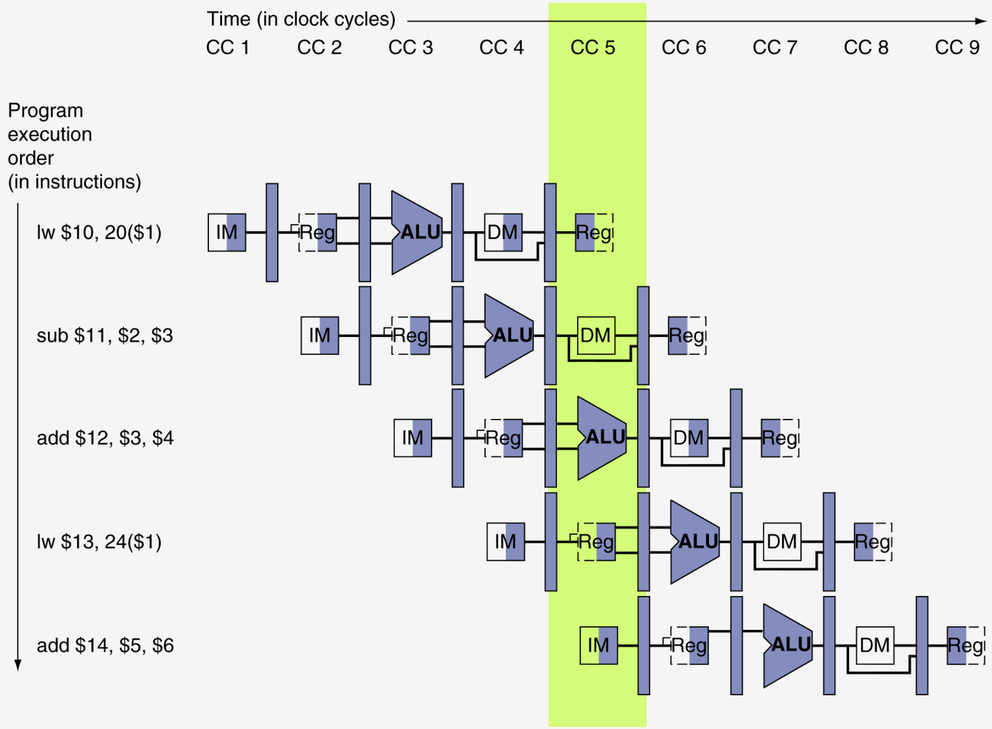


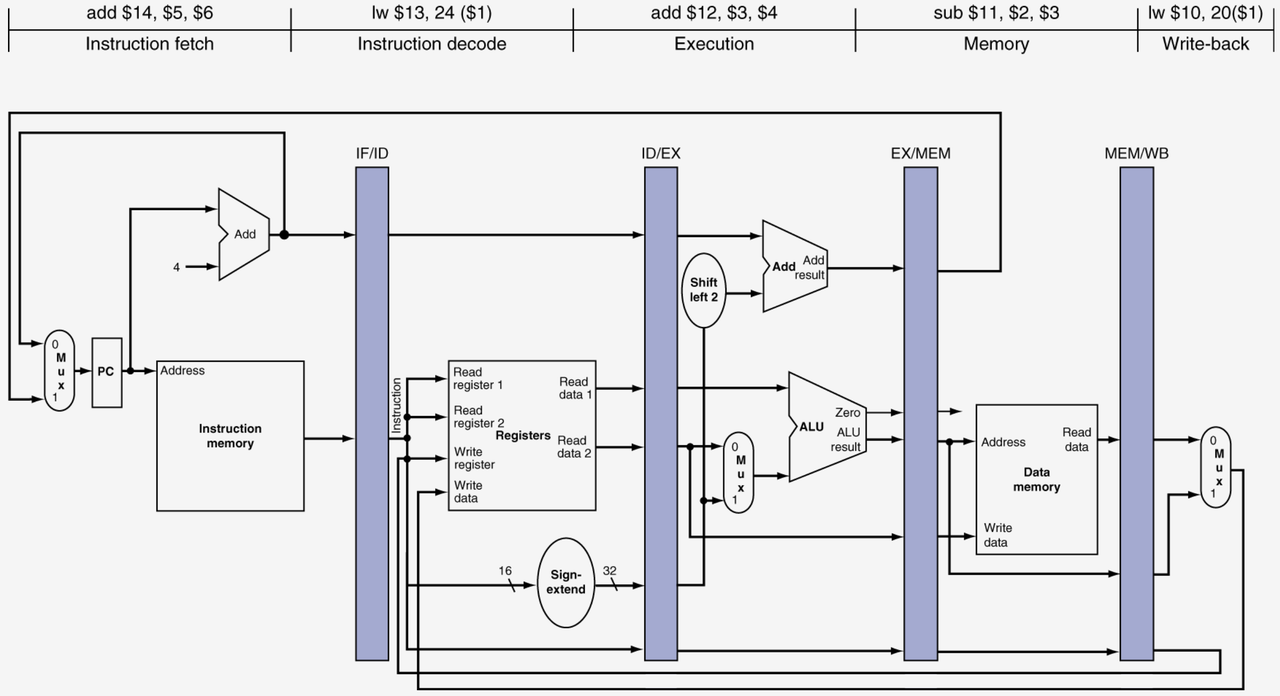
# Single-Cycle Pipeline Diagram

- A representation of a pipeline ***during a single clock cycle***
- Shows what instructions are in the pipeline and at what stage for a given clock cycle
  - Represents a vertical slice through a set of multi-cycle pipeline diagrams showing the usage of the datapath by each of the instructions in the pipeline at the designated clock cycle



# Pipeline Diagrams – a Snapshot in Time




***NOTE:*** Single-cycle pipeline diagram shows the state of the pipeline during clock cycle 5 of the multi-cycle diagram