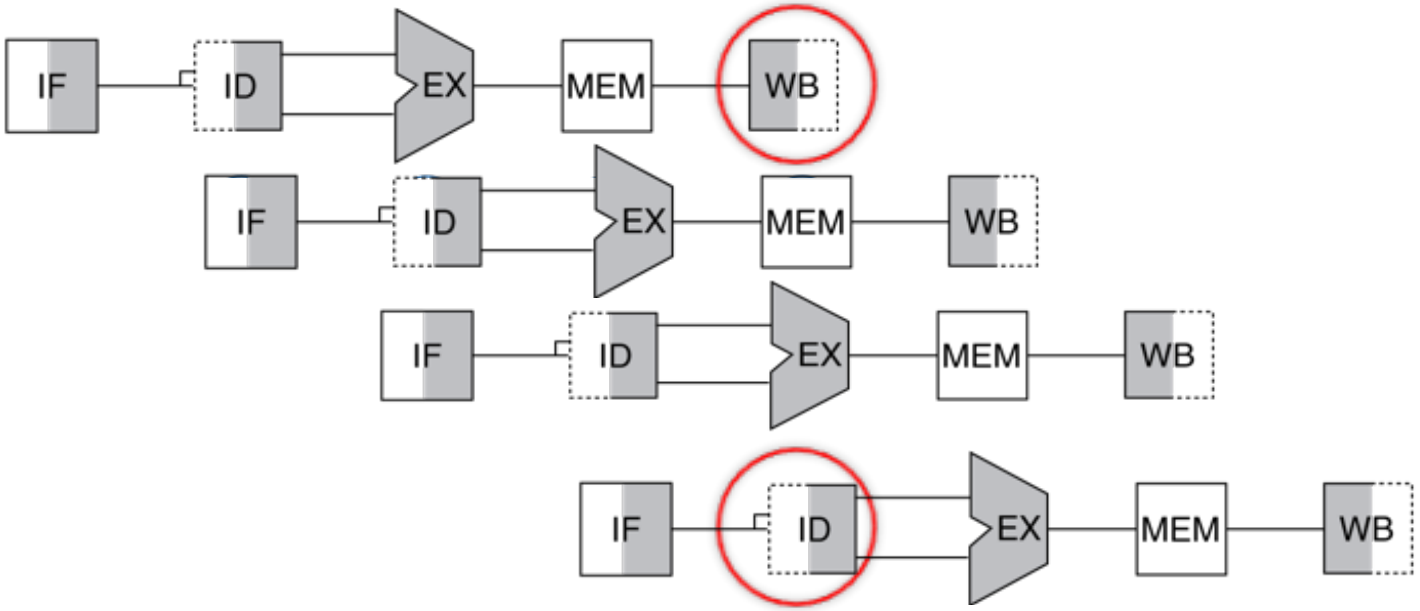
Review: Pipelined CPU

- Overlapped execution of multiple instructions
  - Improved overall throughput
  - Effective CPI=1 (ideal case)
- Each on a different stage using a different major functional unit in datapath
  - IF, ID, EX, MEM, WB
  - Same number of stages for all instruction types
- Hazards
  - Structural / data / control

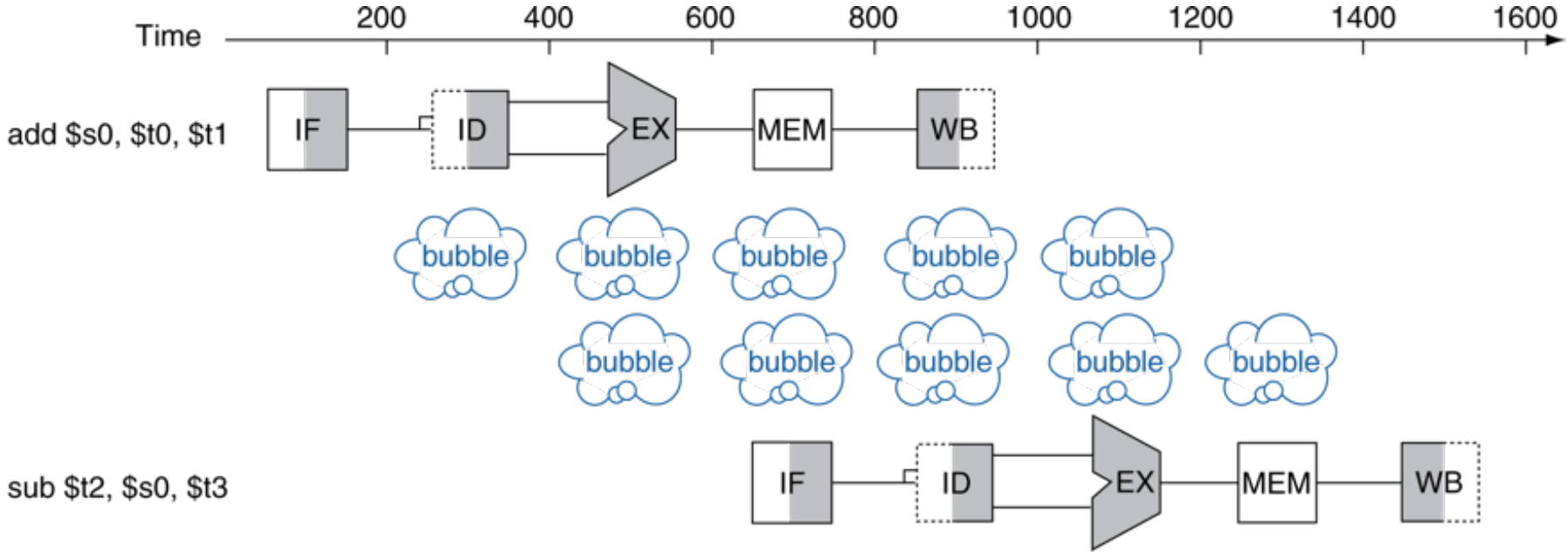
Register Access

- Divide one cycle into two halves:
  - Register updating in the first half; register reading in the second half: conflict avoided!



Data Hazards

- An instruction depends on data generated by a previous instruction
  - `add $s0, $t0, $t1`
    `sub $t2, $s0, $t3`



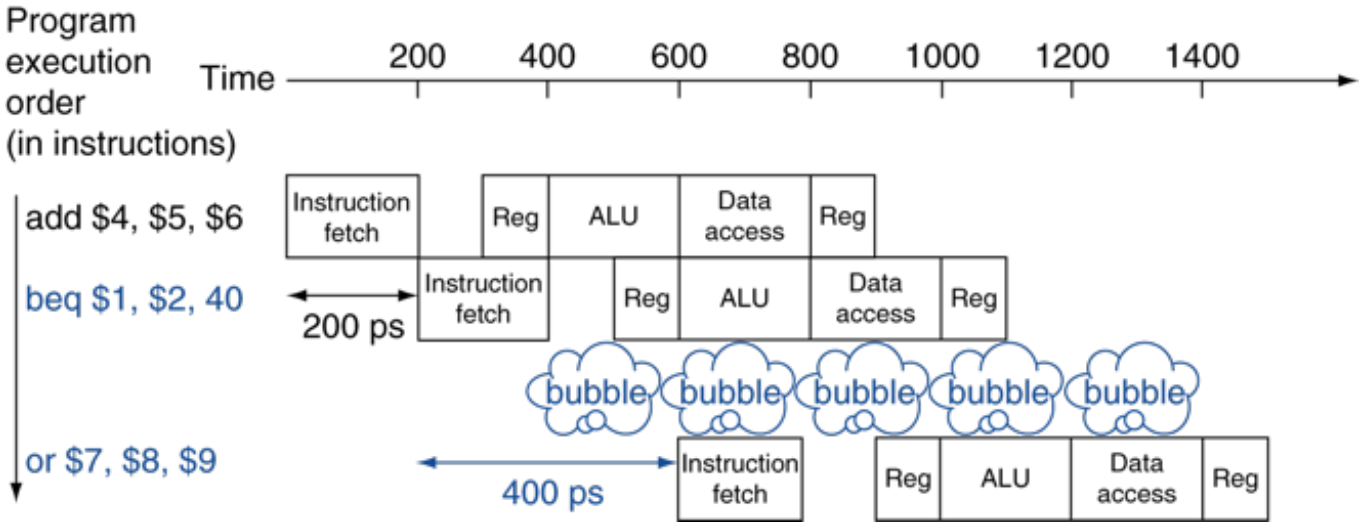
- Better approach: forwarding; reordering

Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - If branch decision is made in EX stage, fetch from the correct addr in MEM stage of the branch instead of ID stage: 2-cycle delay
- How to reduce the delay?
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage
    - Still 1-cycle delay

Stall on Branch

- Wait until branch outcome determined before fetching next instruction



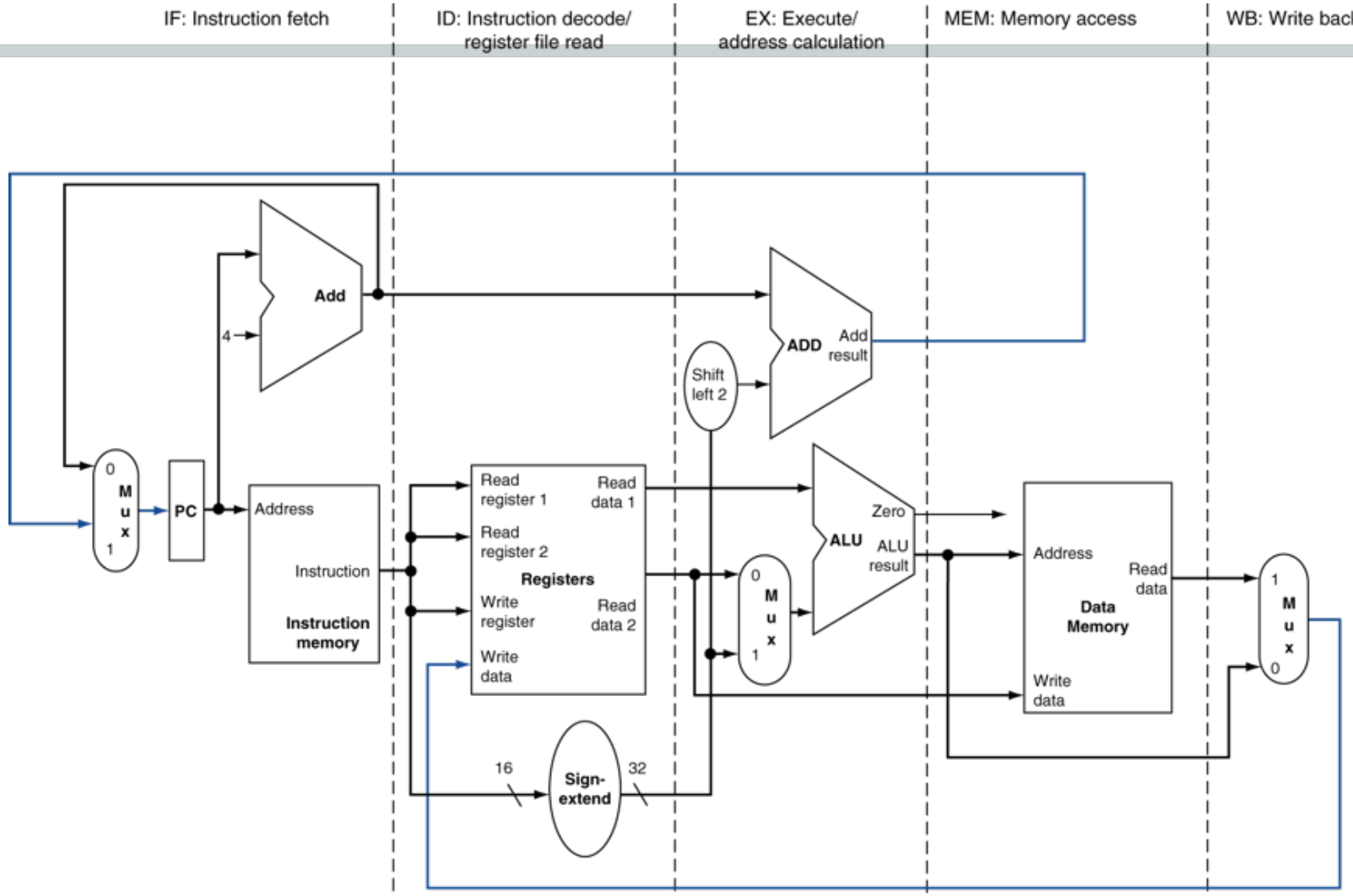
- Better approach: guess/predict branch outcome; branch delay slot

Pipeline Concept

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
  - All hazards can be solved by stall
  - Other approaches: forwarding, prediction, reordering
- Next: hardware implementation
  - Instruction set design affects complexity of pipeline implementation

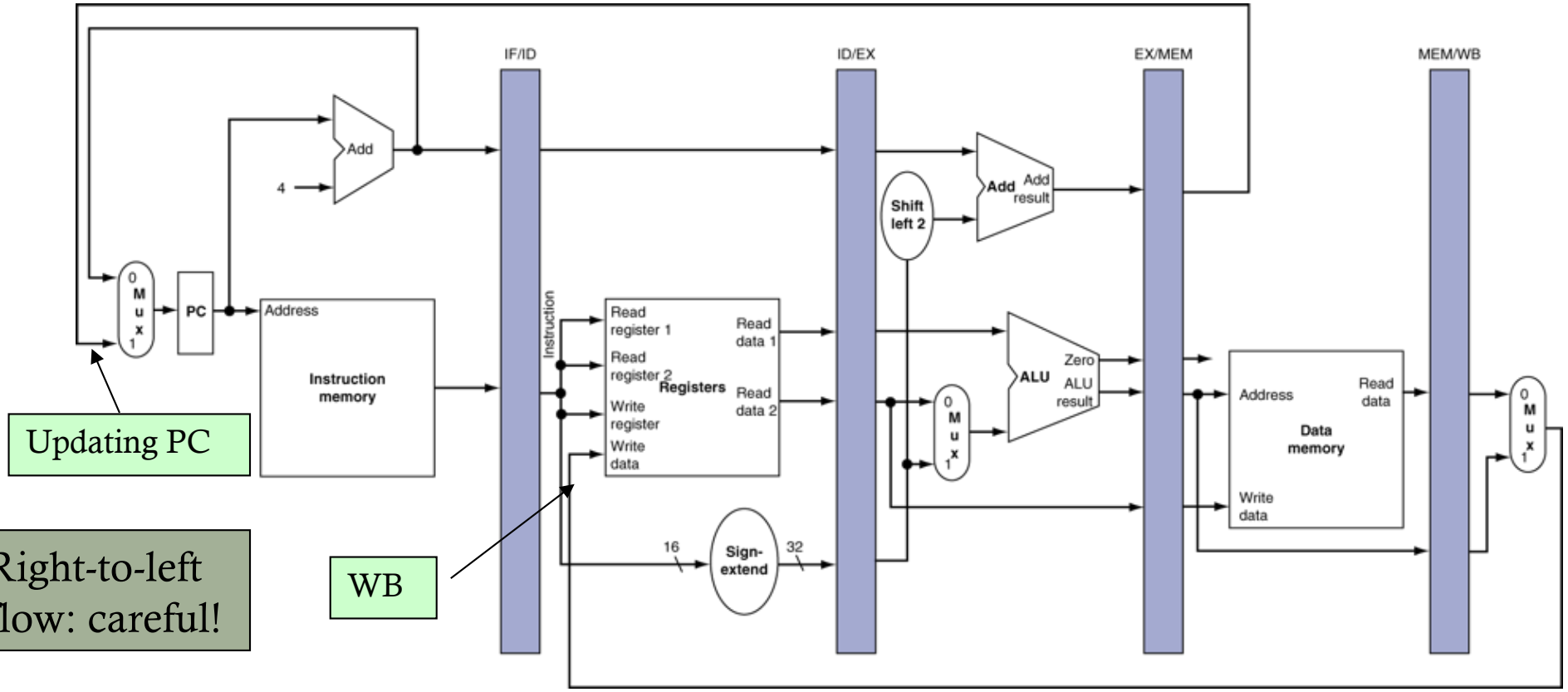
MIPS Pipelined Datapath



- What do we need to add in order to split the datapath into multi-cycle stages?

Pipeline Registers

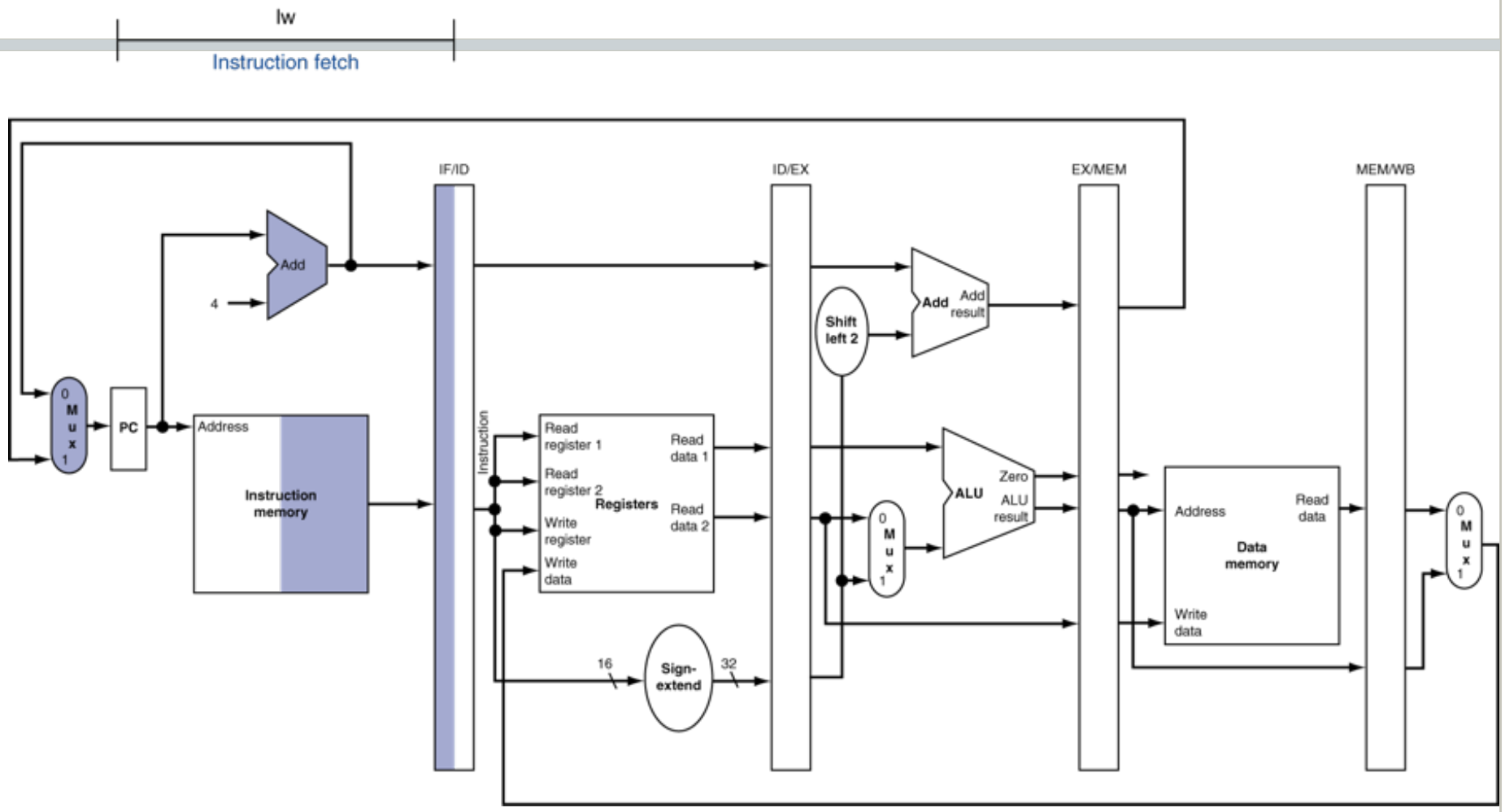
- Need registers between stages
  - To hold information produced in previous cycle



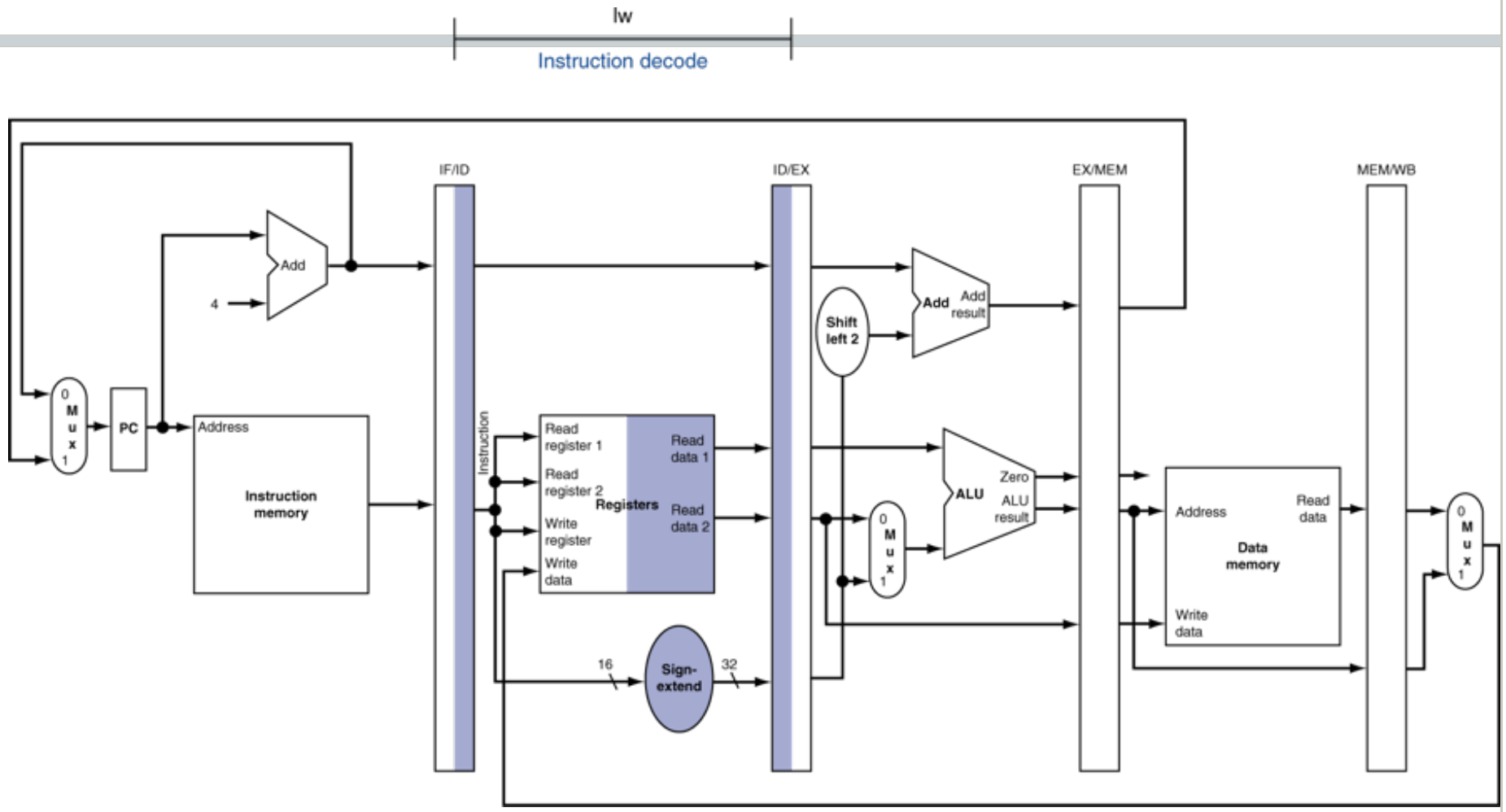
Observations

- 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 and the selection of PC
    - May lead to data hazards and control hazards
- Why there is no pipeline register at the end of the WB stage?
  - Instructions must update either register file, or memory, or PC
- Let's check how load/store is executed in pipeline

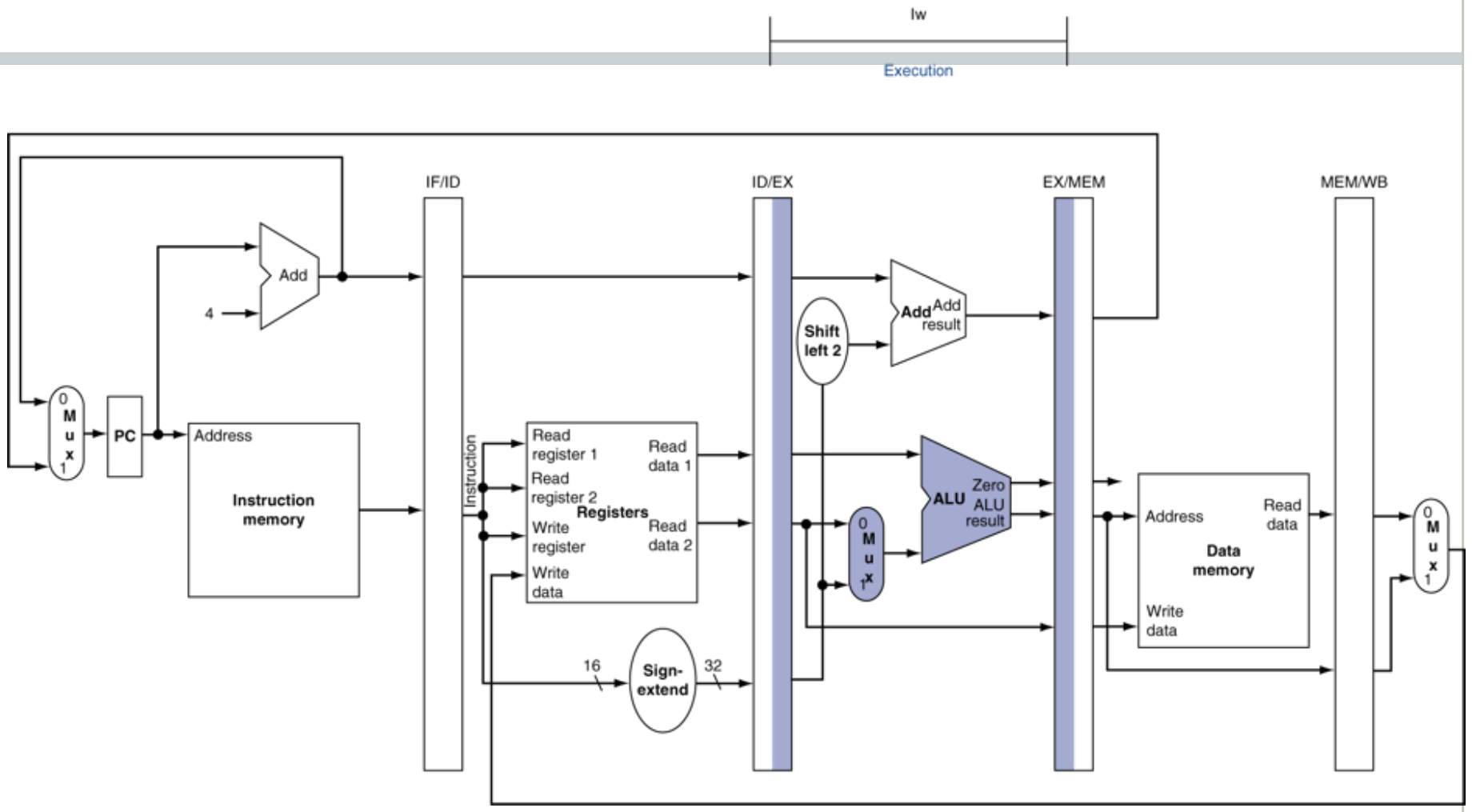
IF for Load, Store, …



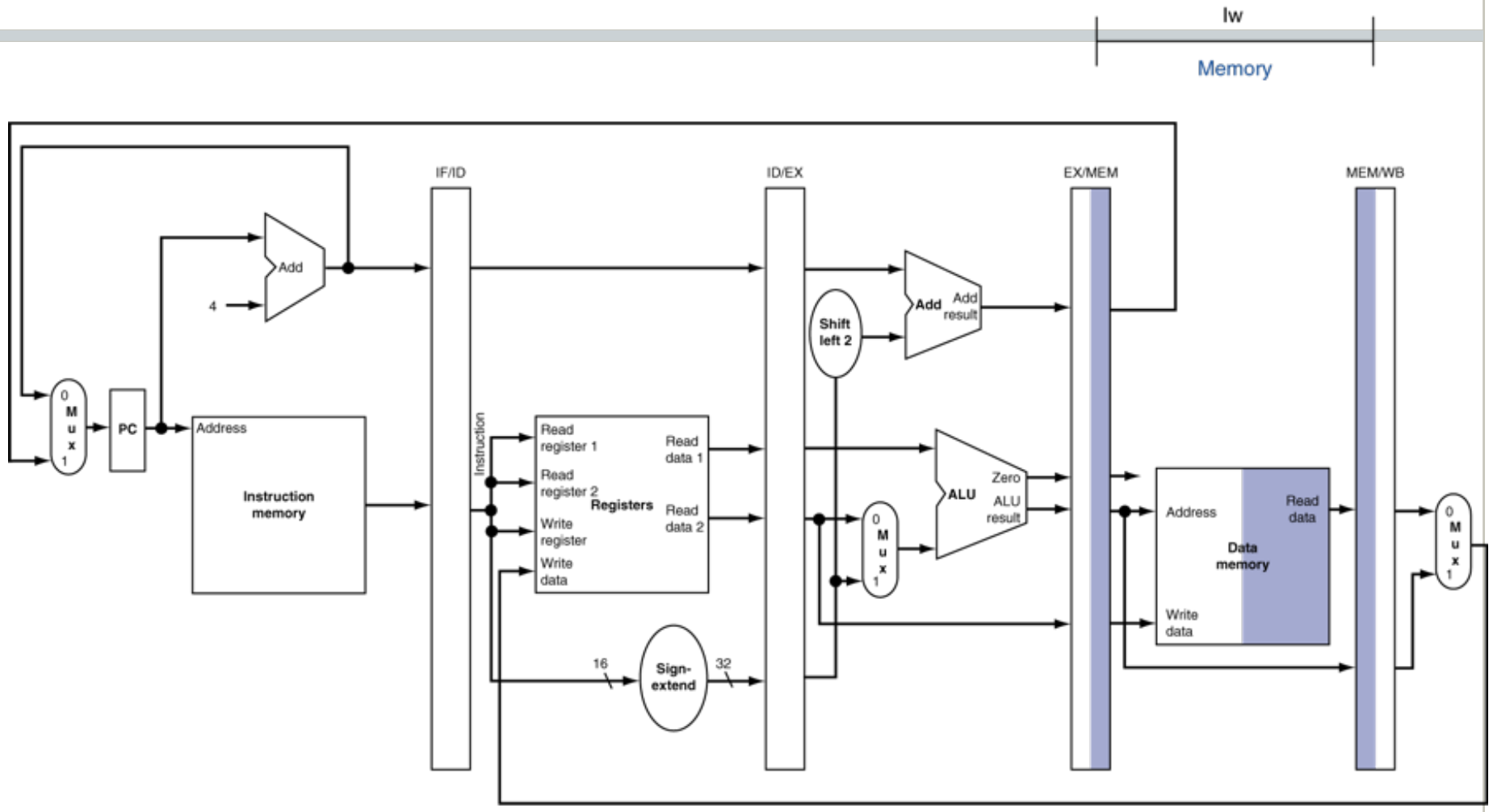
ID for Load, Store, …



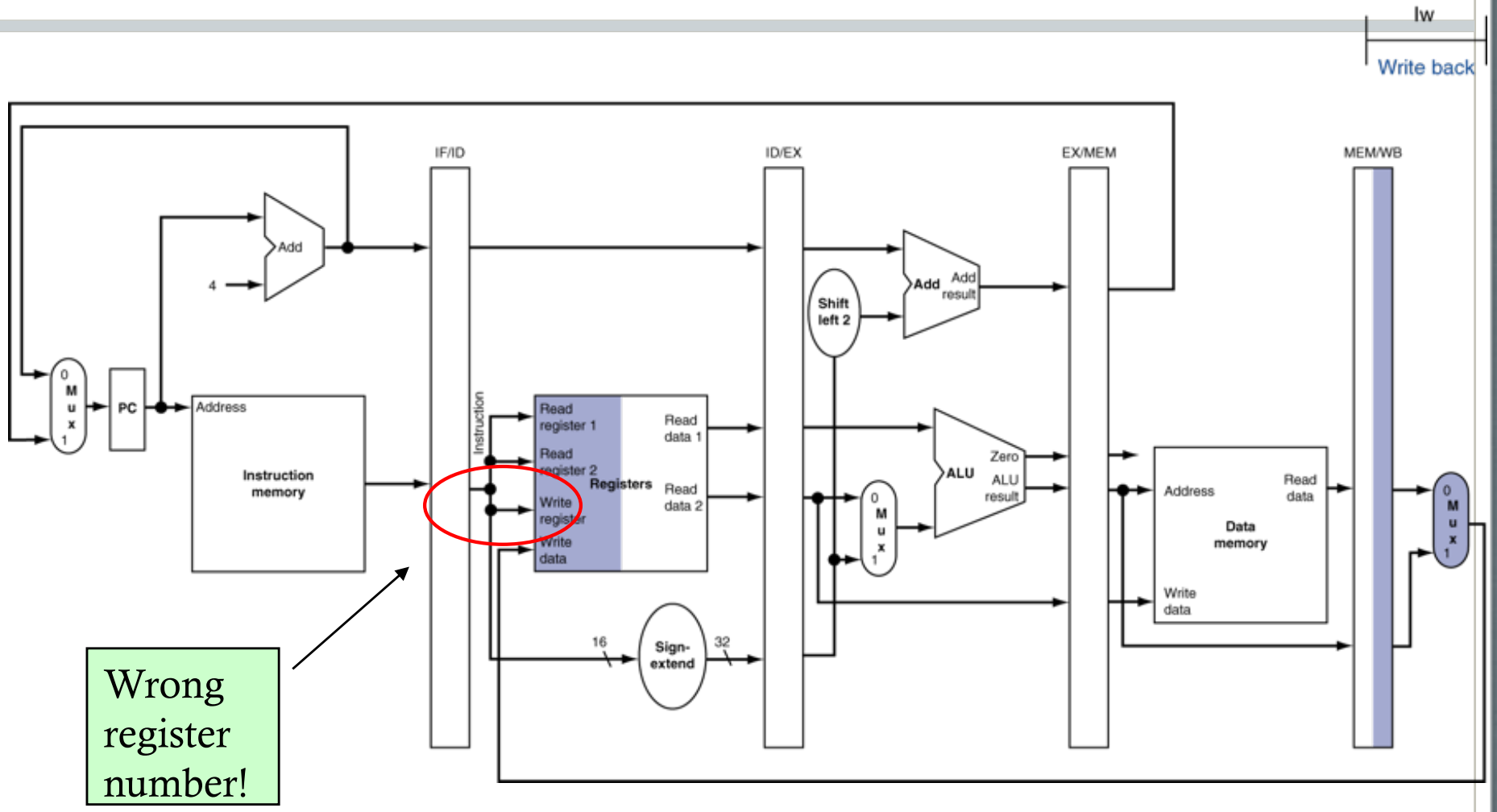
EX for Load


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
Read data
Data memory
Write data
16
Sign-extend
32

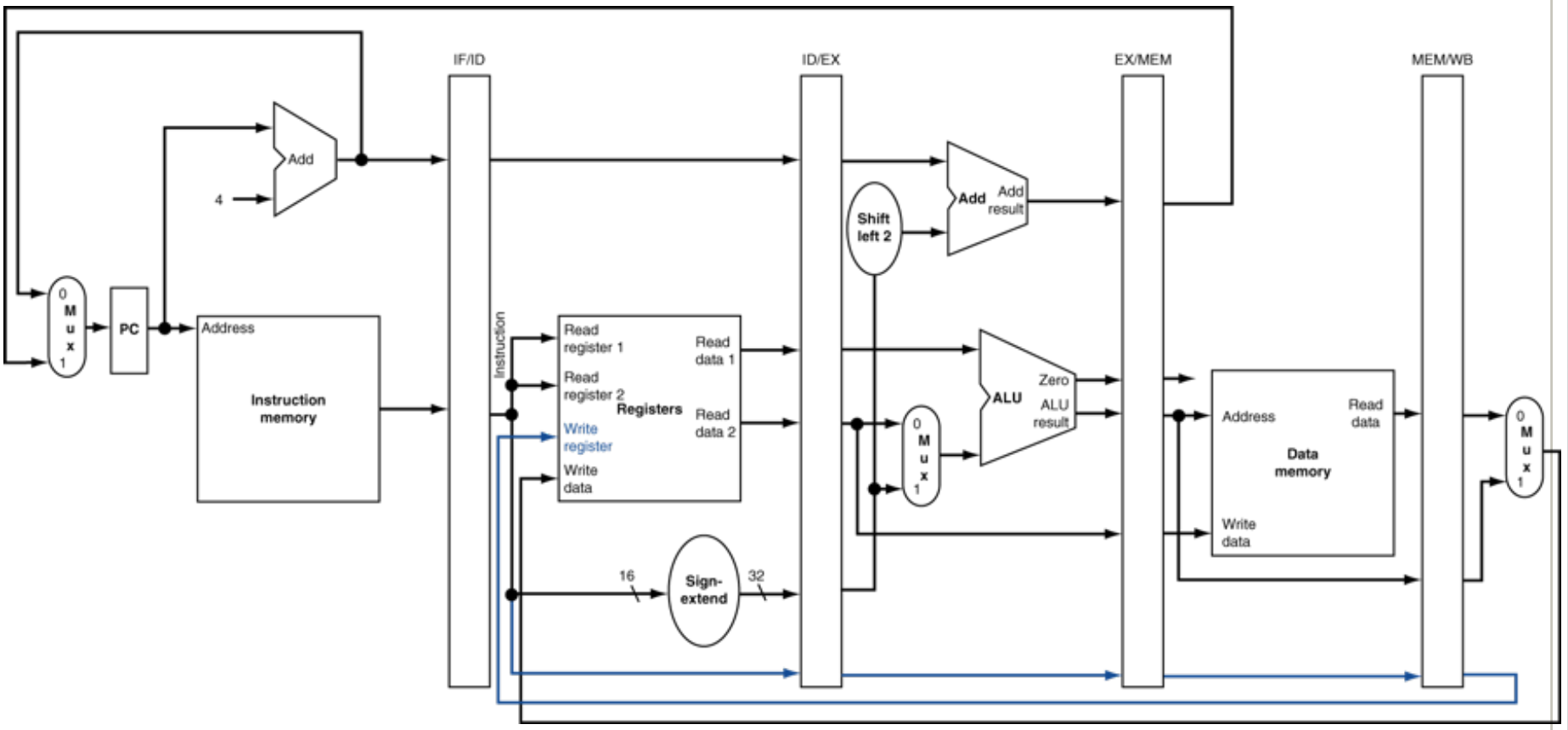
MEM for Load


lw
Memory
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
Data memory
Read data
Write data
16
Sign-extend
32

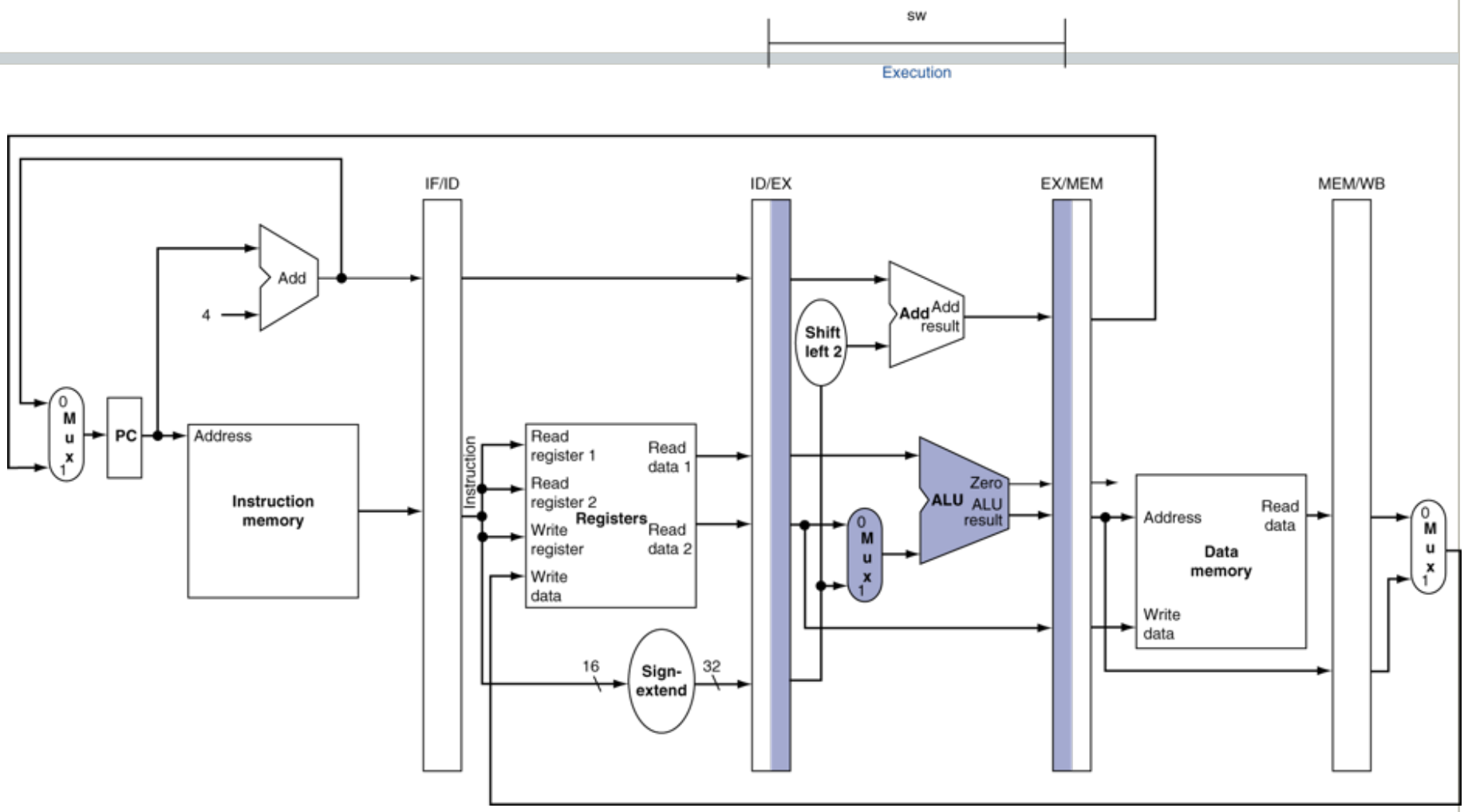
WB for Load


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
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
Read data
Data memory
Write data
16
Sign-extend
32
Wrong register number!

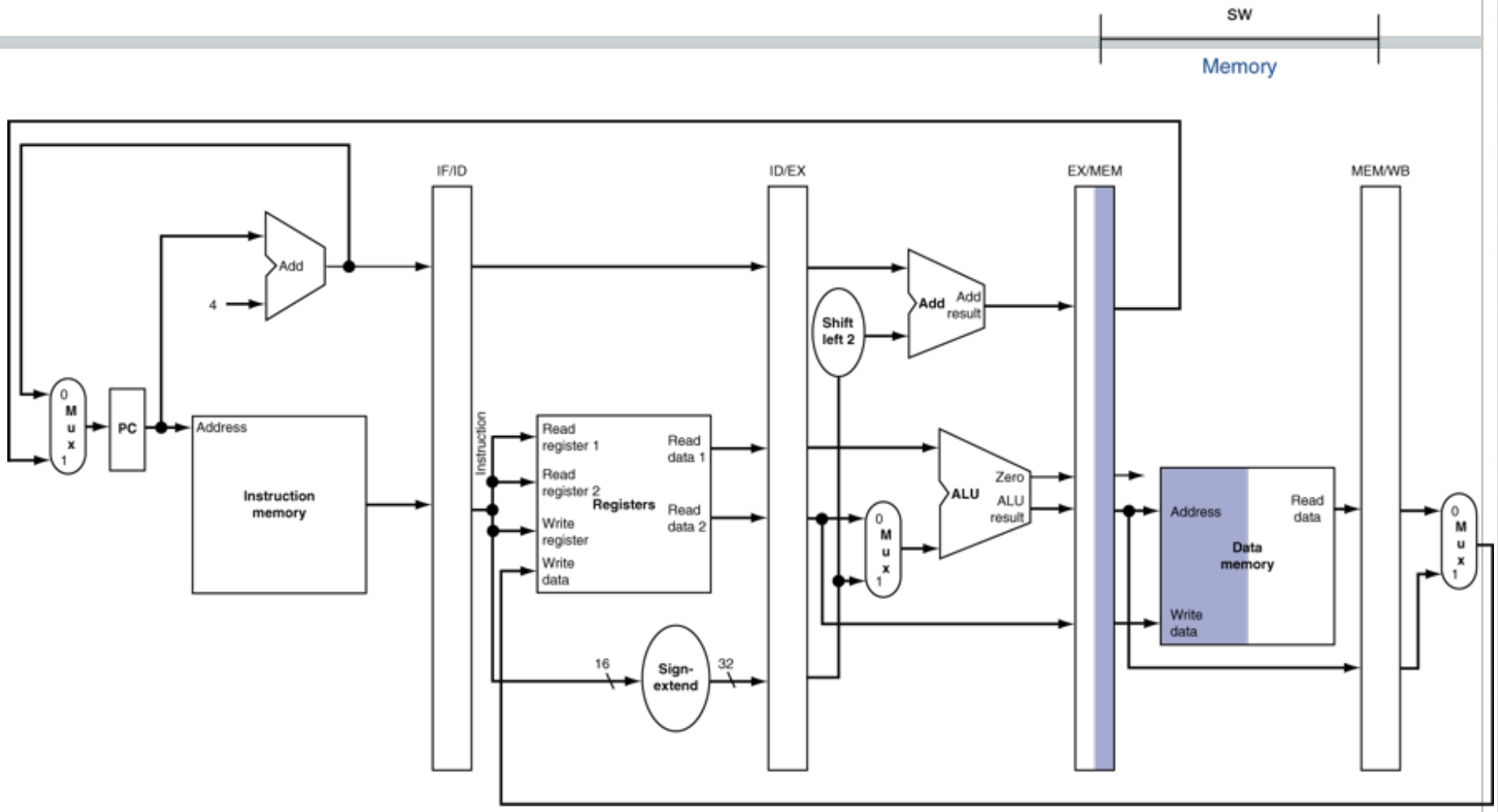
Corrected Datapath for Load



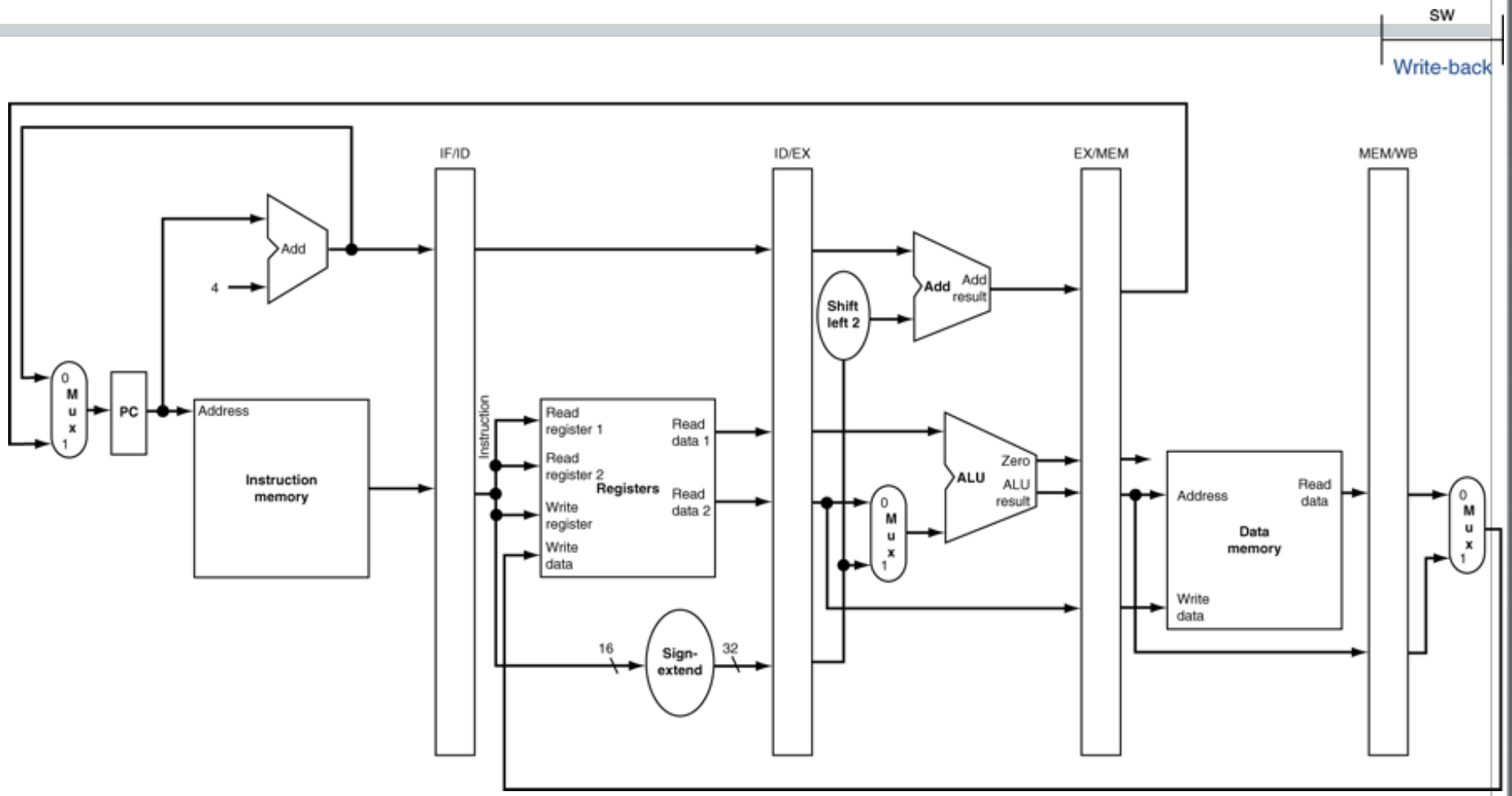
EX for Store


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
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32

MEM for Store

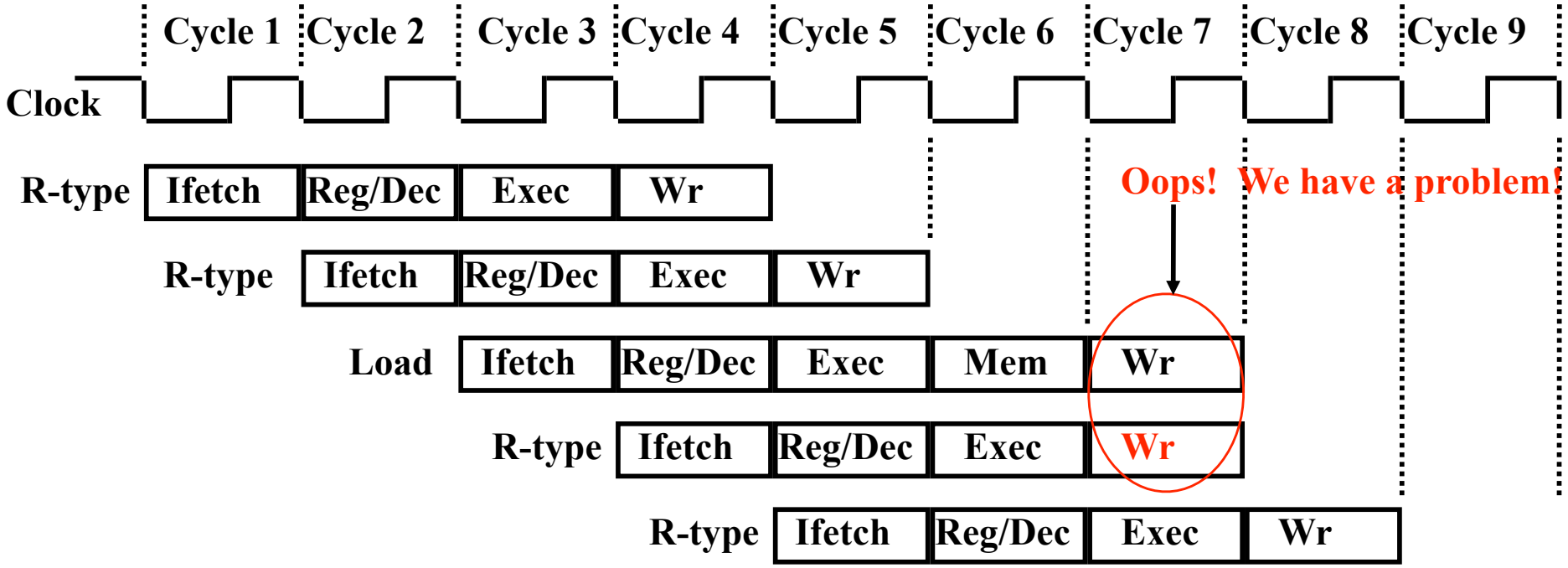


WB for Store


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
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign-extend
32
Address
Read data
Data memory
Write data

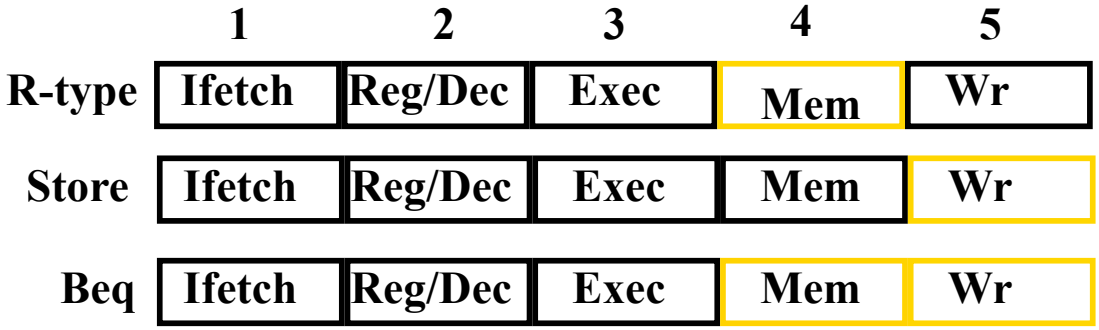
Pipeline for R-type

- R-type instructions only need IF, ID, EX, WB
- Can we skip MEM stage for R-type instructions?



Observations

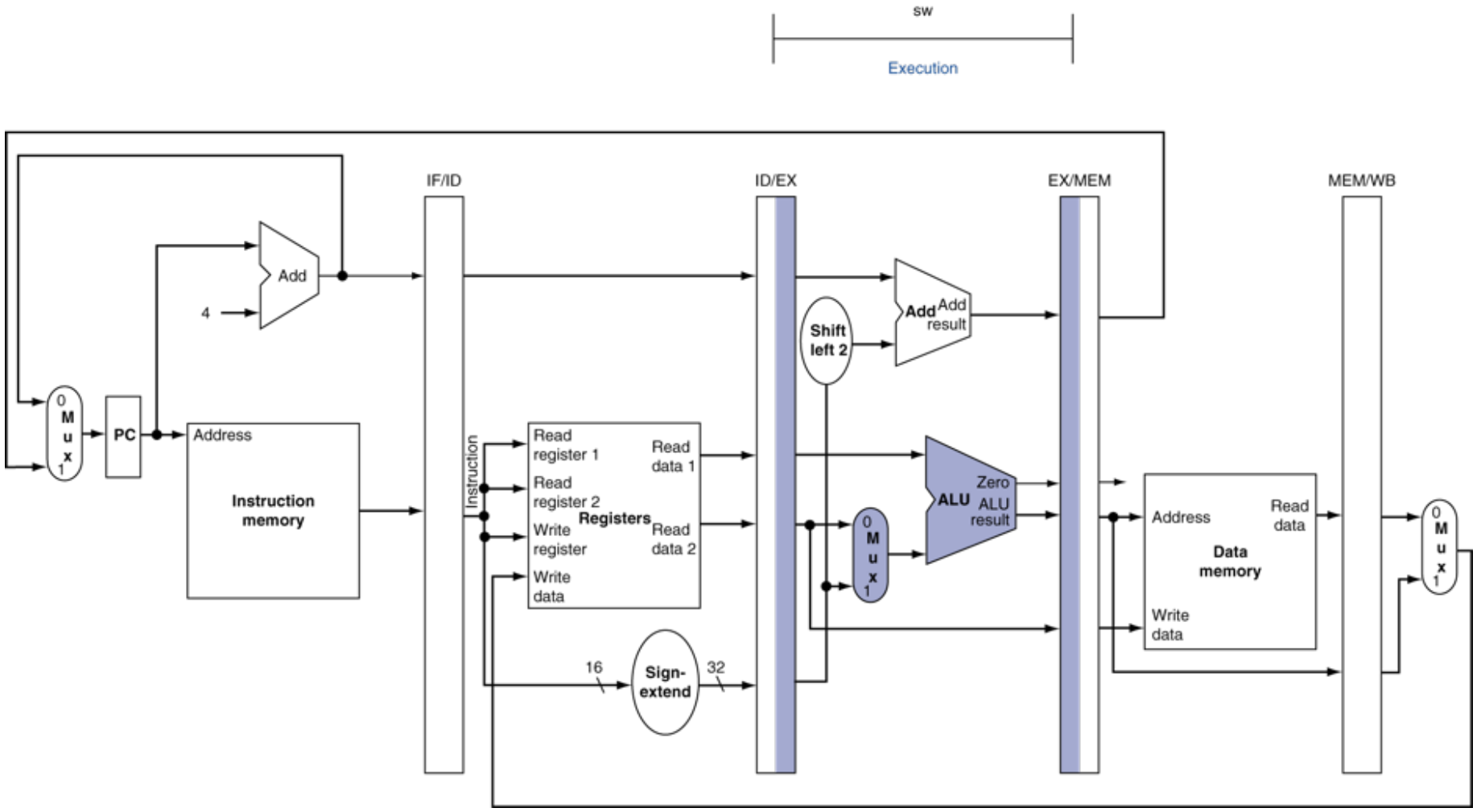
- Each functional unit can only be used once per instruction
- Each functional unit must be used at the same stage for all instructions
- All instruction types have five pipeline stages
- Some stages may be wasted for some instructions
  - Mem stage is a NO-OP stage for R-type instructions: not needed but still must go through



Pipeline Representations

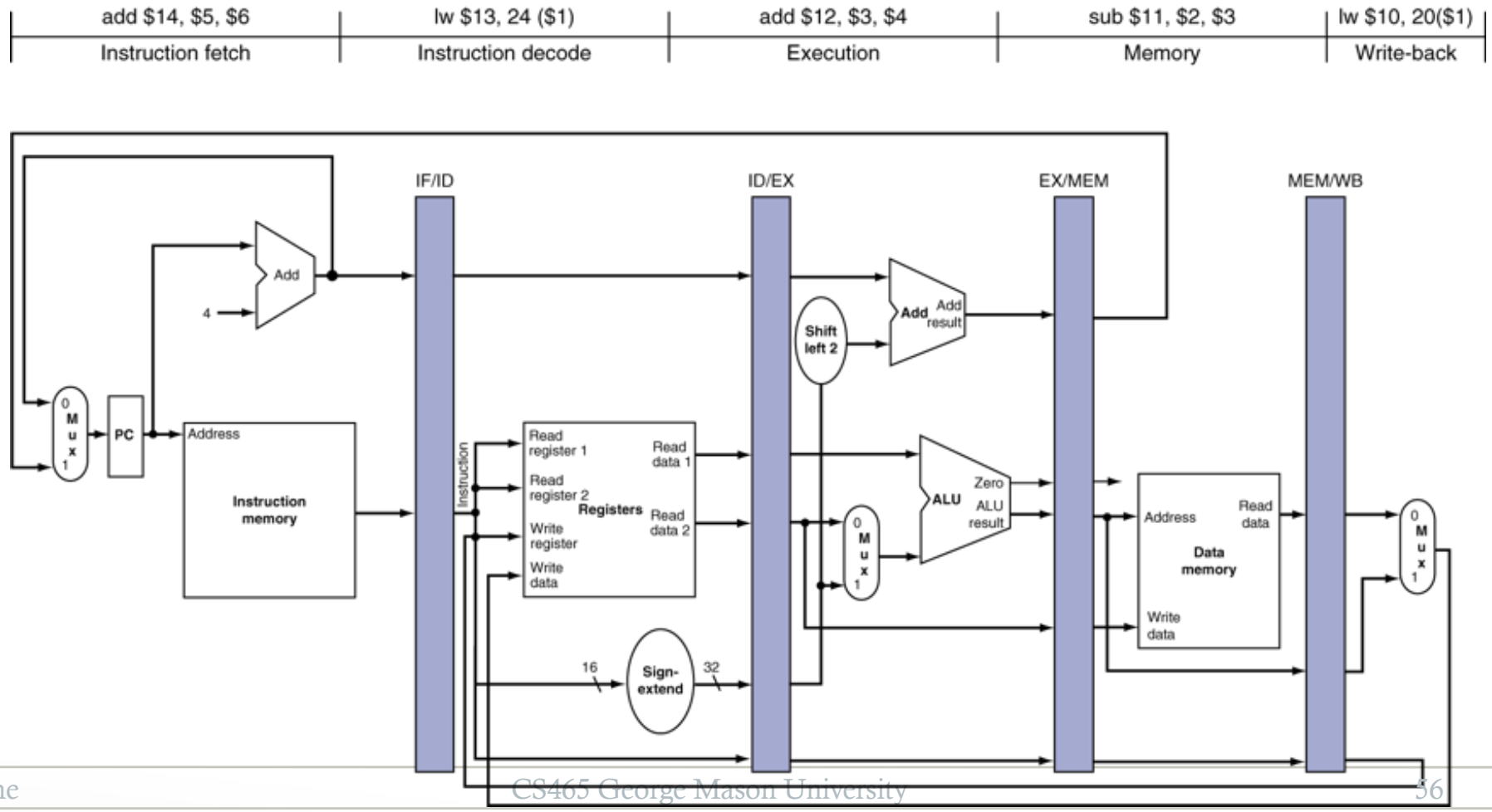
- Cycle-by-cycle flow of instructions through the pipelined datapath
- Two main types
  - "Single-clock-cycle" pipeline diagram
    - Shows pipeline usage in a single cycle
    - Highlight resources used
  - "Multi-clock-cycle" diagram
    - Graph of operation over time

Single-Cycle Diagram Example



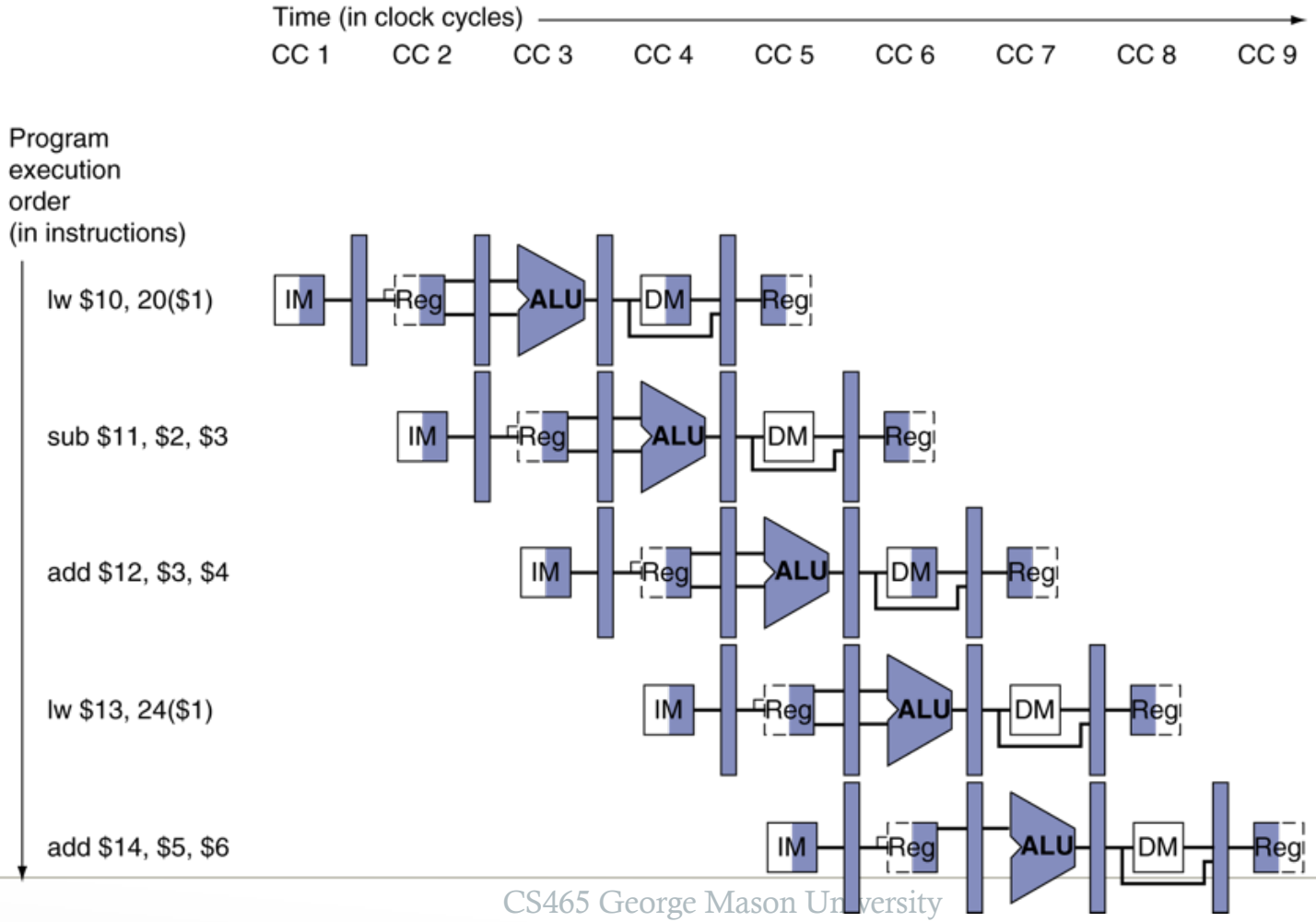
Single-Cycle Diagram Example

- State of pipeline in a given cycle (multiple instructions)



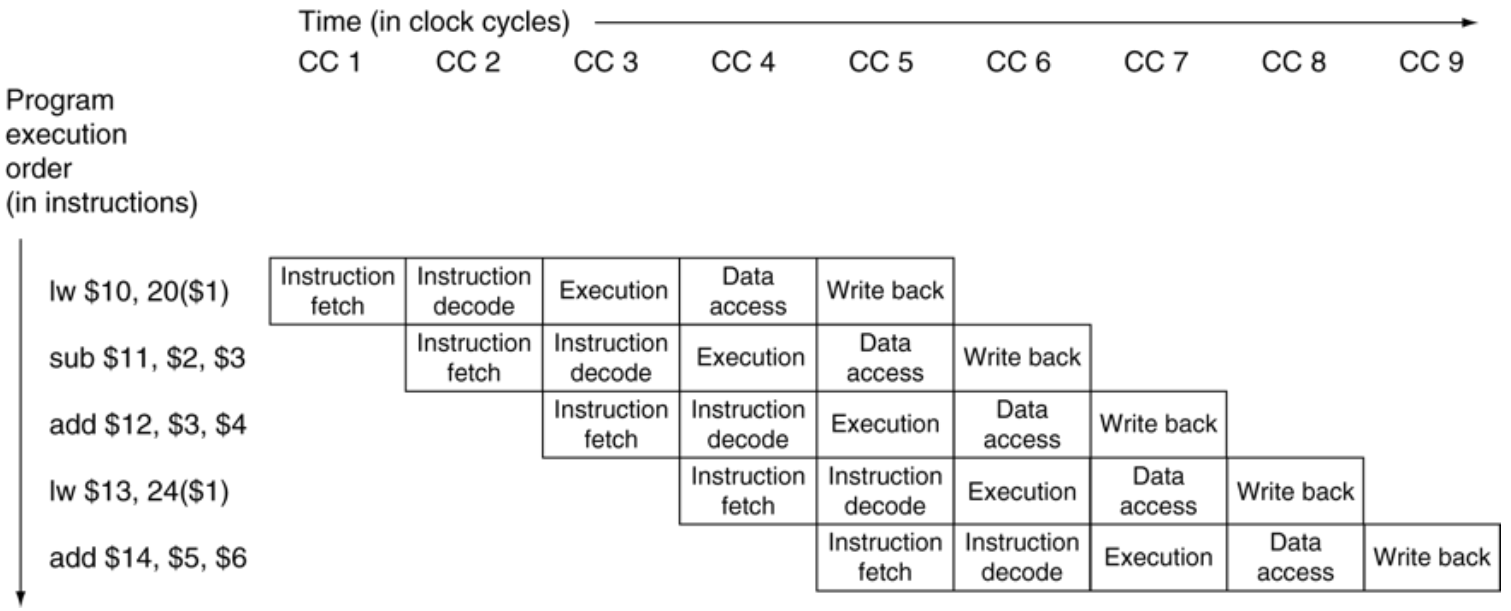
Multi-Cycle Pipeline Diagram

- Form showing resource usage



Multi-Cycle Pipeline Diagram

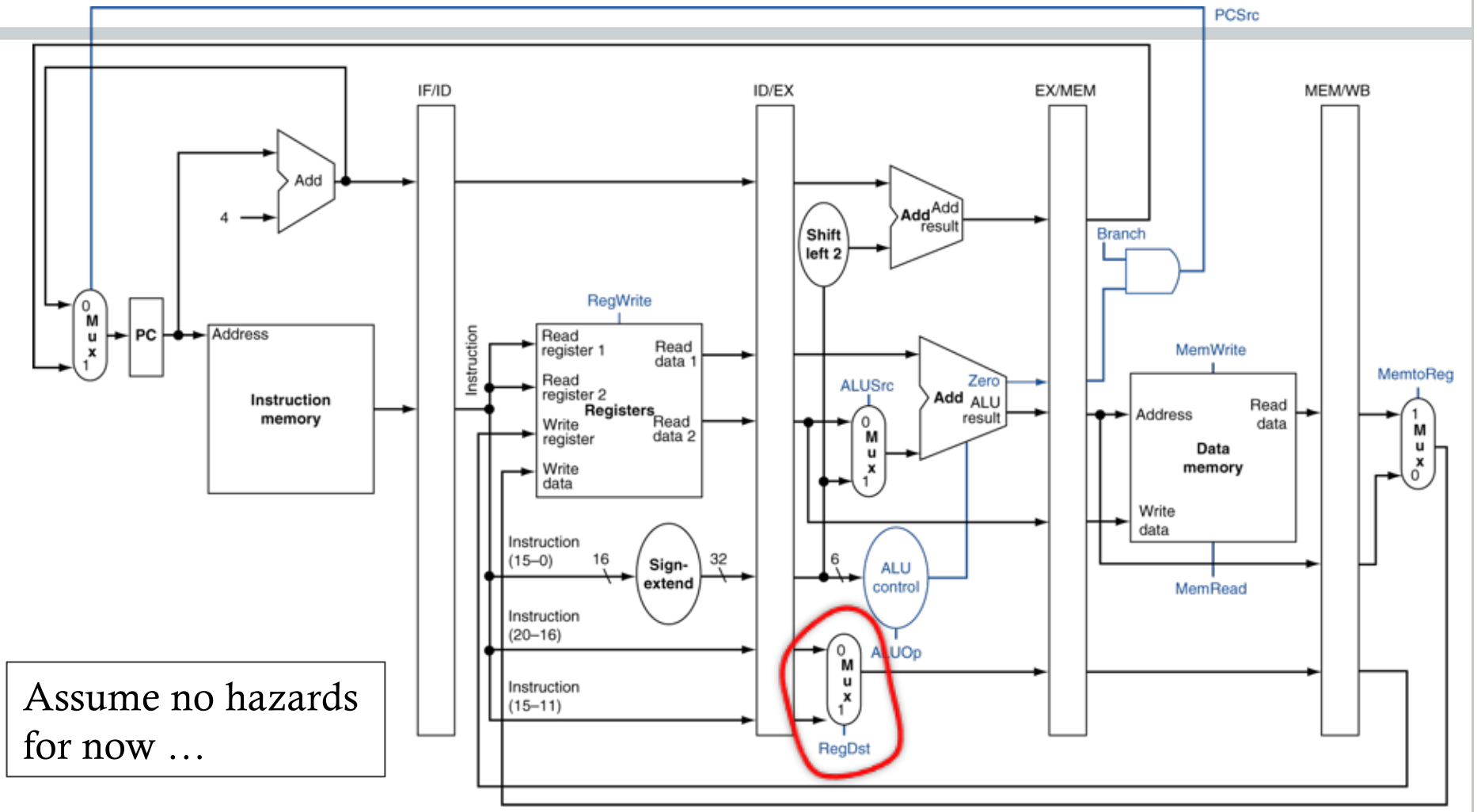
- Traditional form



Outline

- Overview of pipeline
  - Stages
  - Hazards
- Pipelined datapath
  - Pipeline registers
  - Pipelined execution
- Pipelined control
  - Different signals for different stages
  - Propagate control signals

Pipelined Control (Simplified)



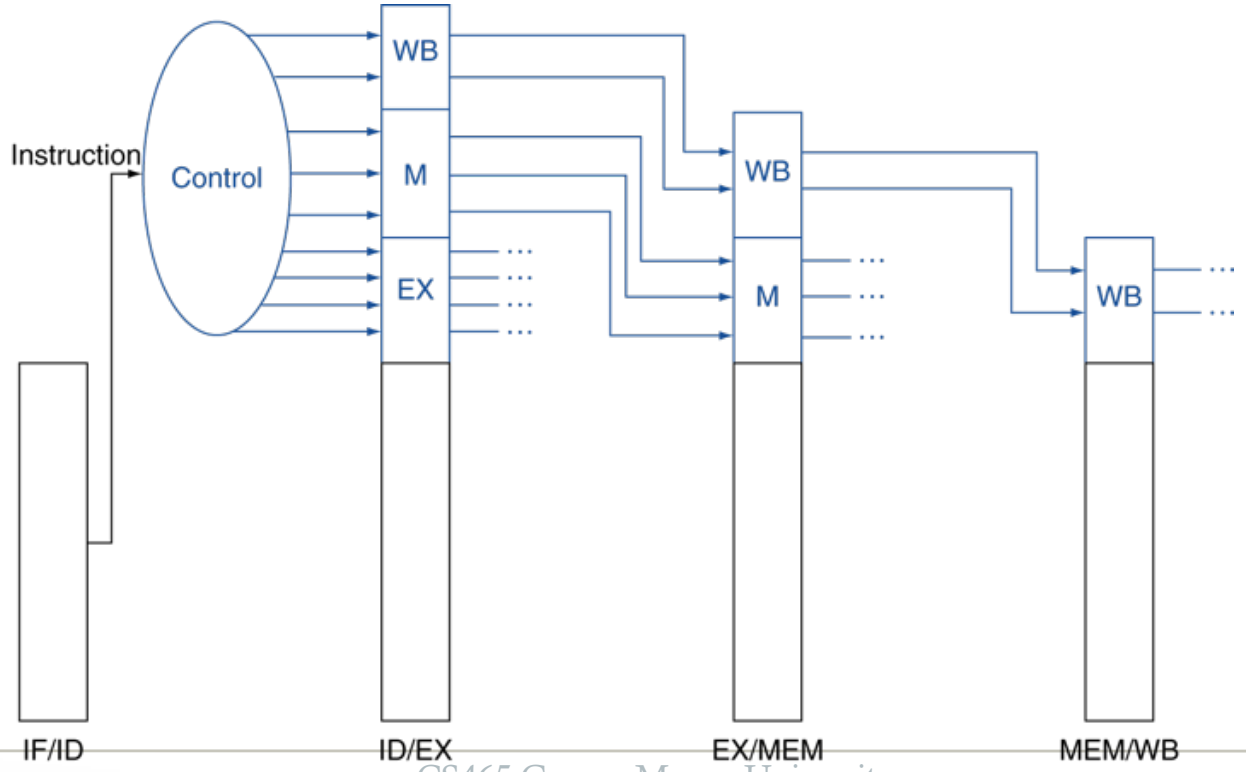
Assume no hazards for now …

Observations

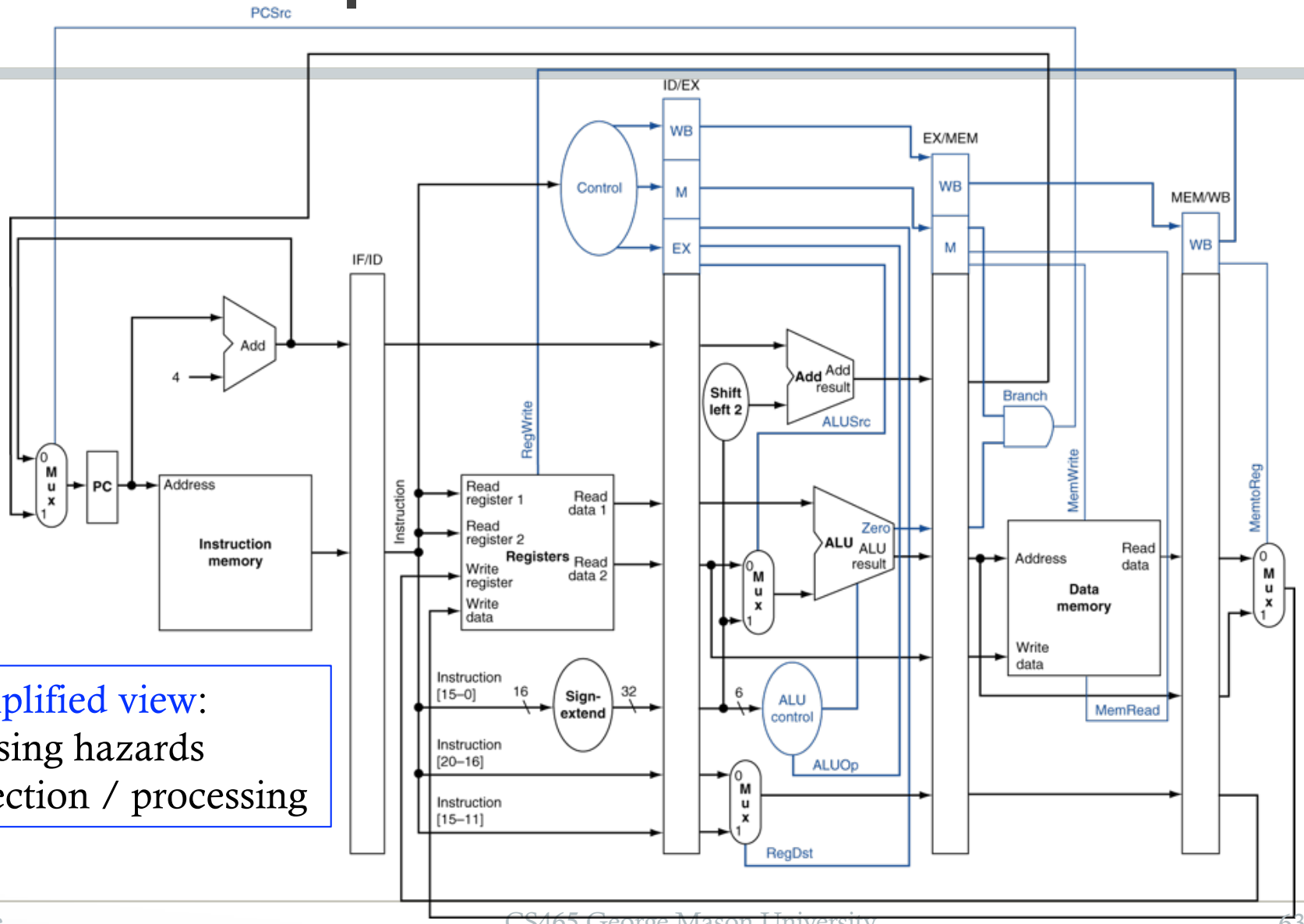
- No write control for all pipeline registers and PC since they are updated at every clock cycle (for now)
- Control signals used in different pipeline stages:
  - IF – NONE
  - ID – NONE
  - EXE – RegDst, ALUOp, ALUSrc
  - MEM – Branch, MemRead, MemWrite
  - WB – MemtoReg, RegWrite
- Group these nine control lines into 3 subsets:
  - ALUControl, MEMControl, WBControl
- Control signals are generated at ID stage, how to pass them to other stages?

Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation
- Expand pipeline registers to include control signals



Pipelined Control



Simplified view:
missing hazards
detection / processing

Summary

- Overview of pipeline
  - Stages
  - Hazards
- Pipelined datapath
  - Pipeline registers
  - Pipelined execution
- Pipelined control
  - Different signals for different stages
  - Propagate control signals