

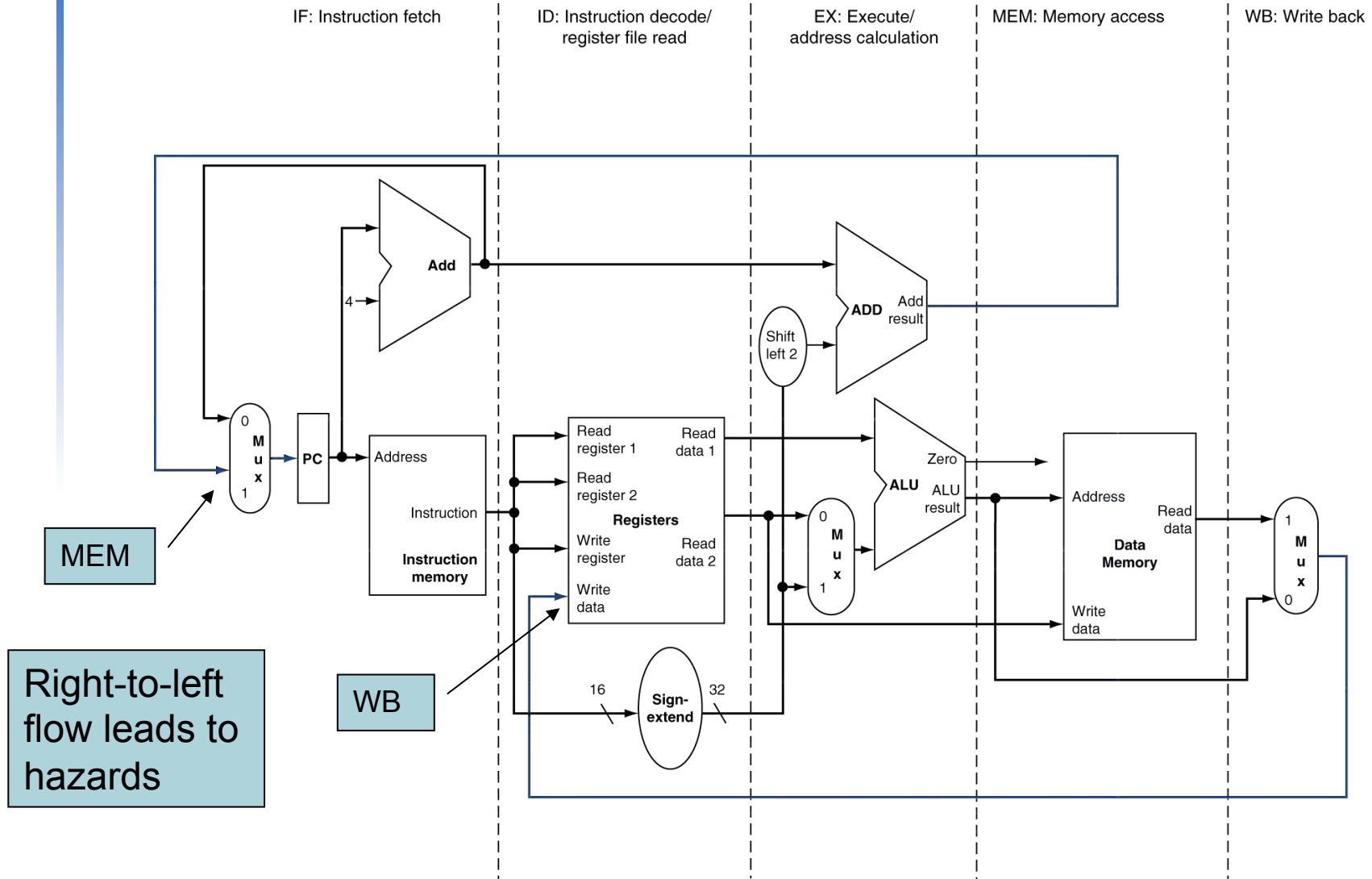
Chapter 4 Part 3

The Processor - Pipelining

Pipeline Summary

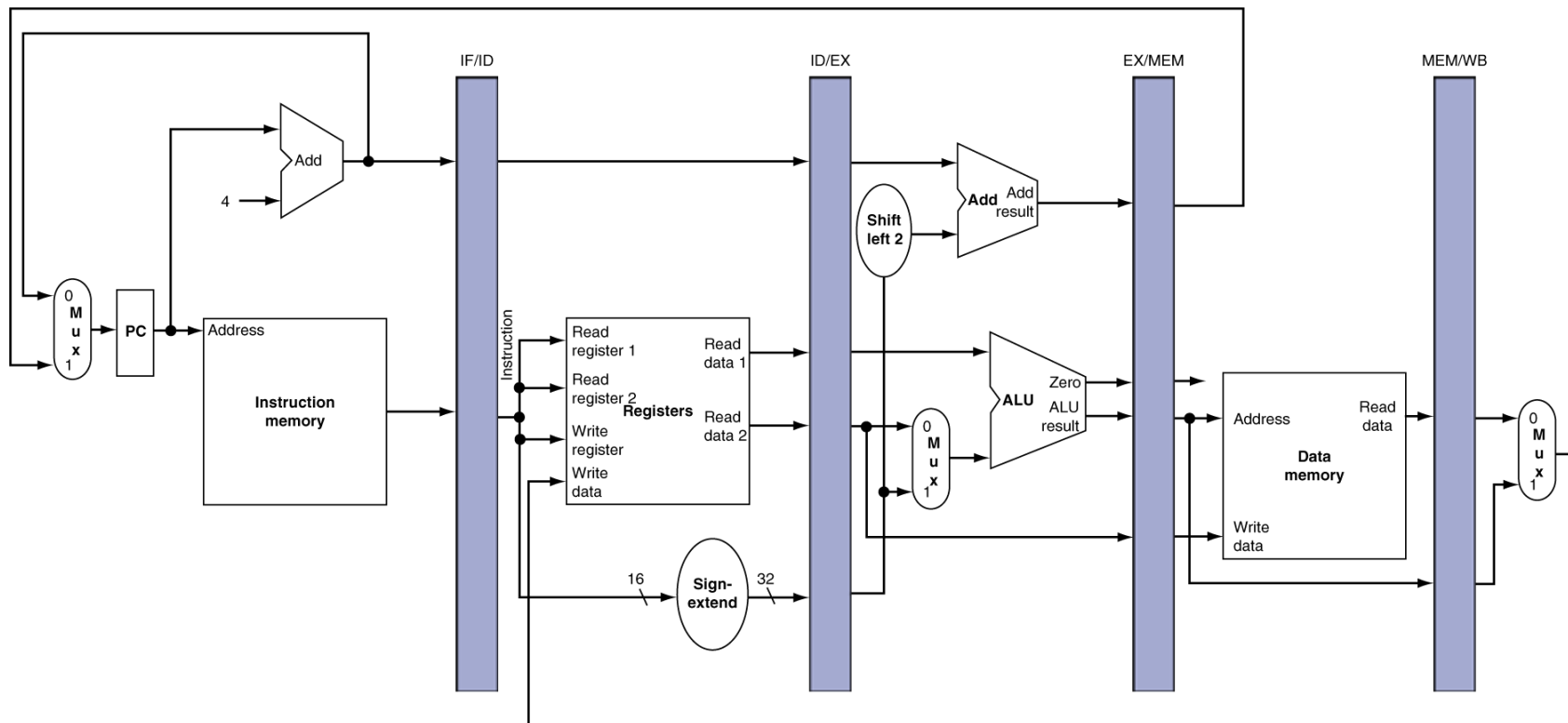
- Pipelining improves performance by increasing instruction throughput
 - Executes multiple instructions in parallel
 - Each instruction has the same latency
- Subject to hazards
 - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

MIPS Pipelined Datapath



Pipeline registers

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

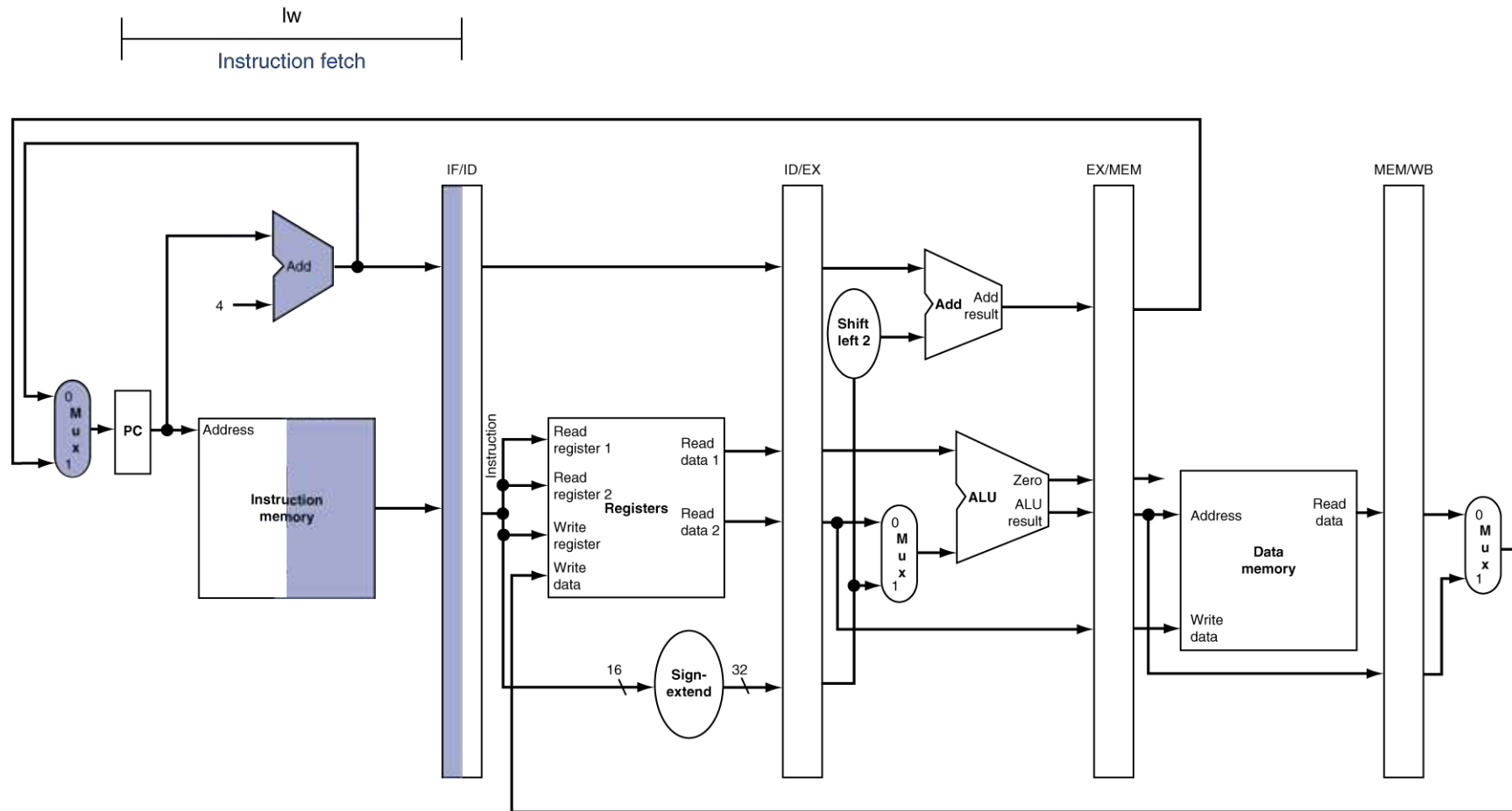


Pipeline Operation

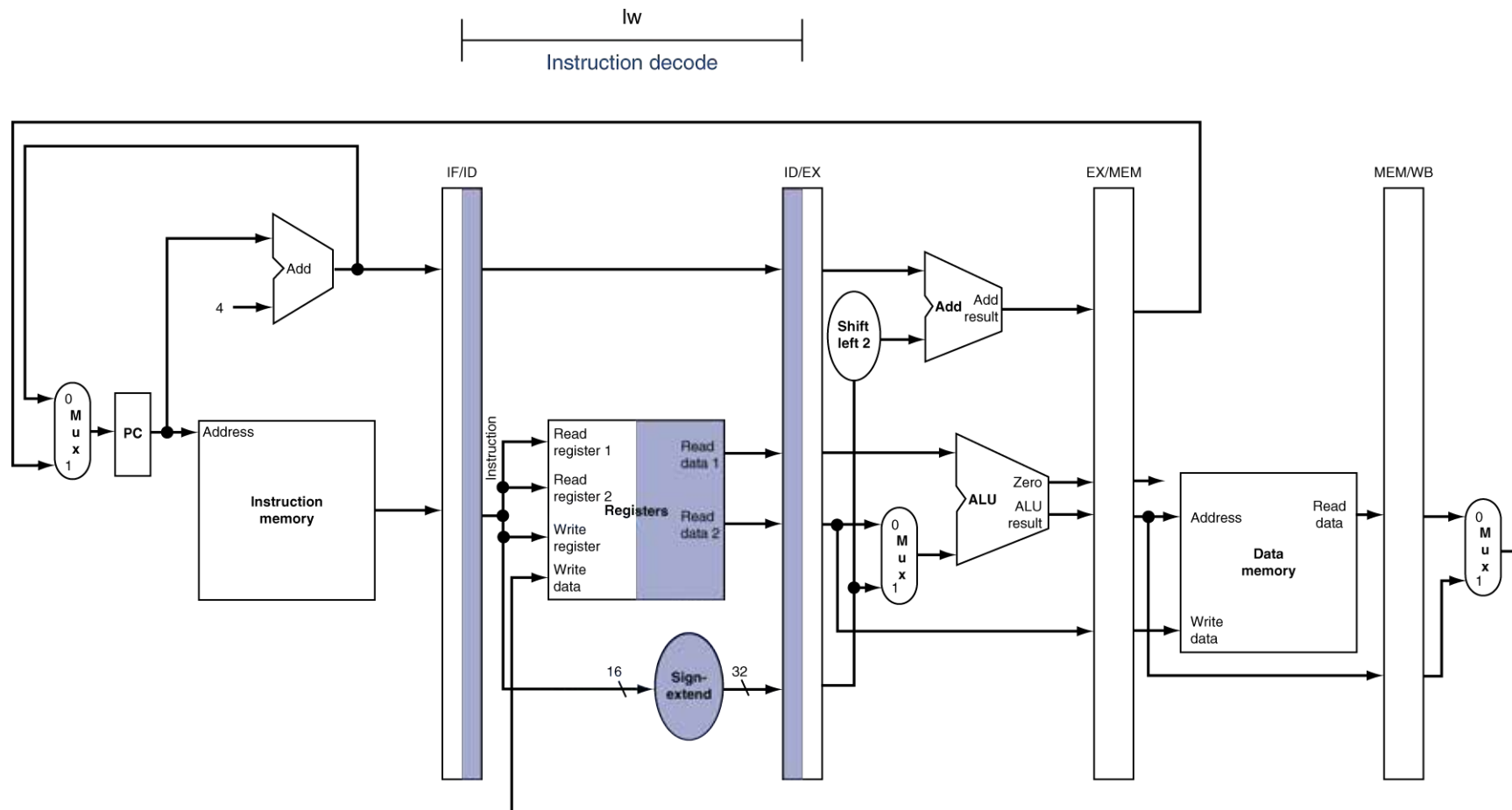
- Cycle-by-cycle flow of instructions through the pipelined datapath
 - “Single-clock-cycle” pipeline diagram
 - Shows pipeline usage in a single cycle
 - Highlight resources used
 - c.f. “multi-clock-cycle” diagram
 - Graph of operation over time
- We’ll look at “single-clock-cycle” diagrams for load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

Single-Clock-Cycle Diagram

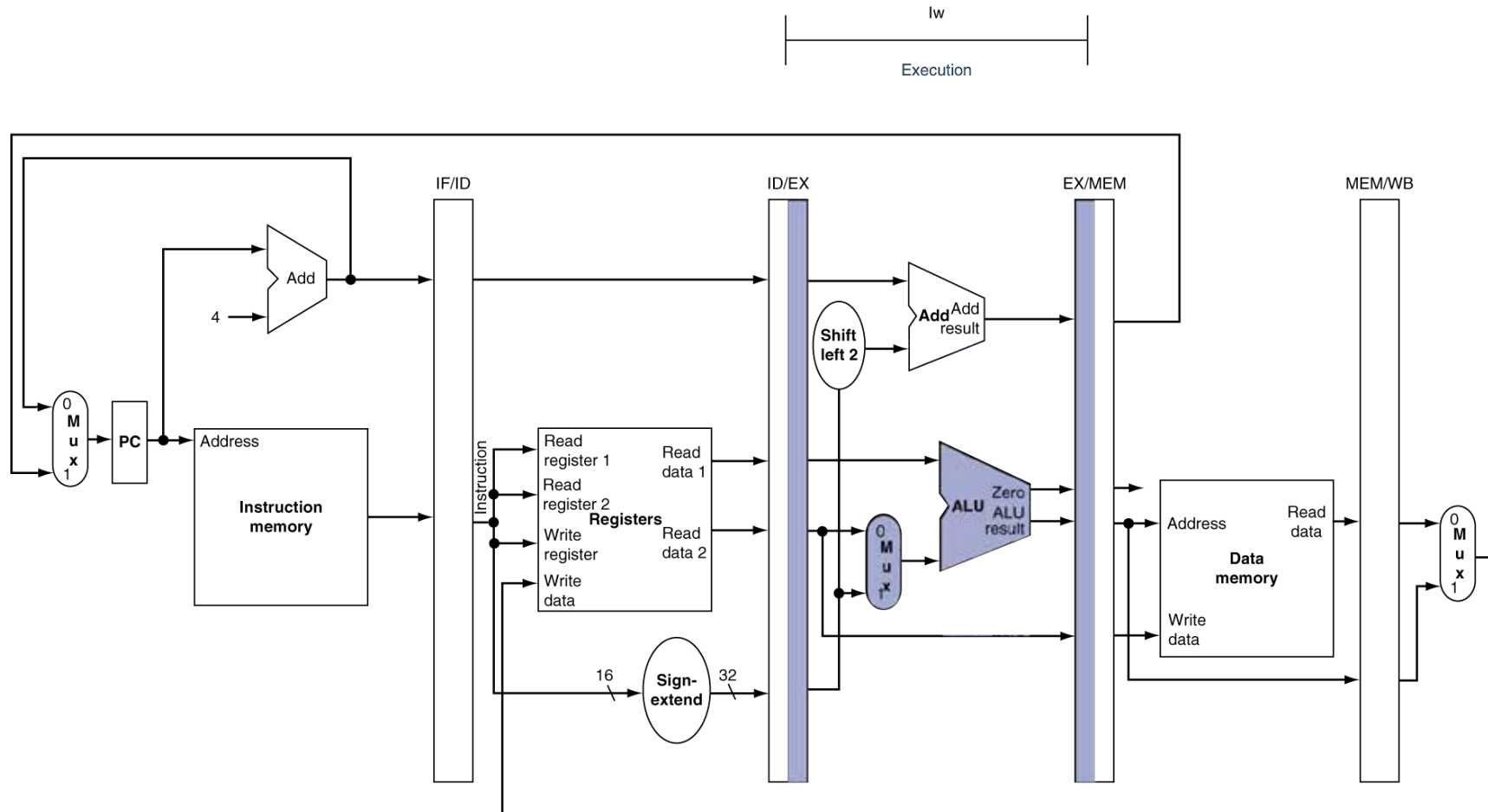
- IF for Load and Store



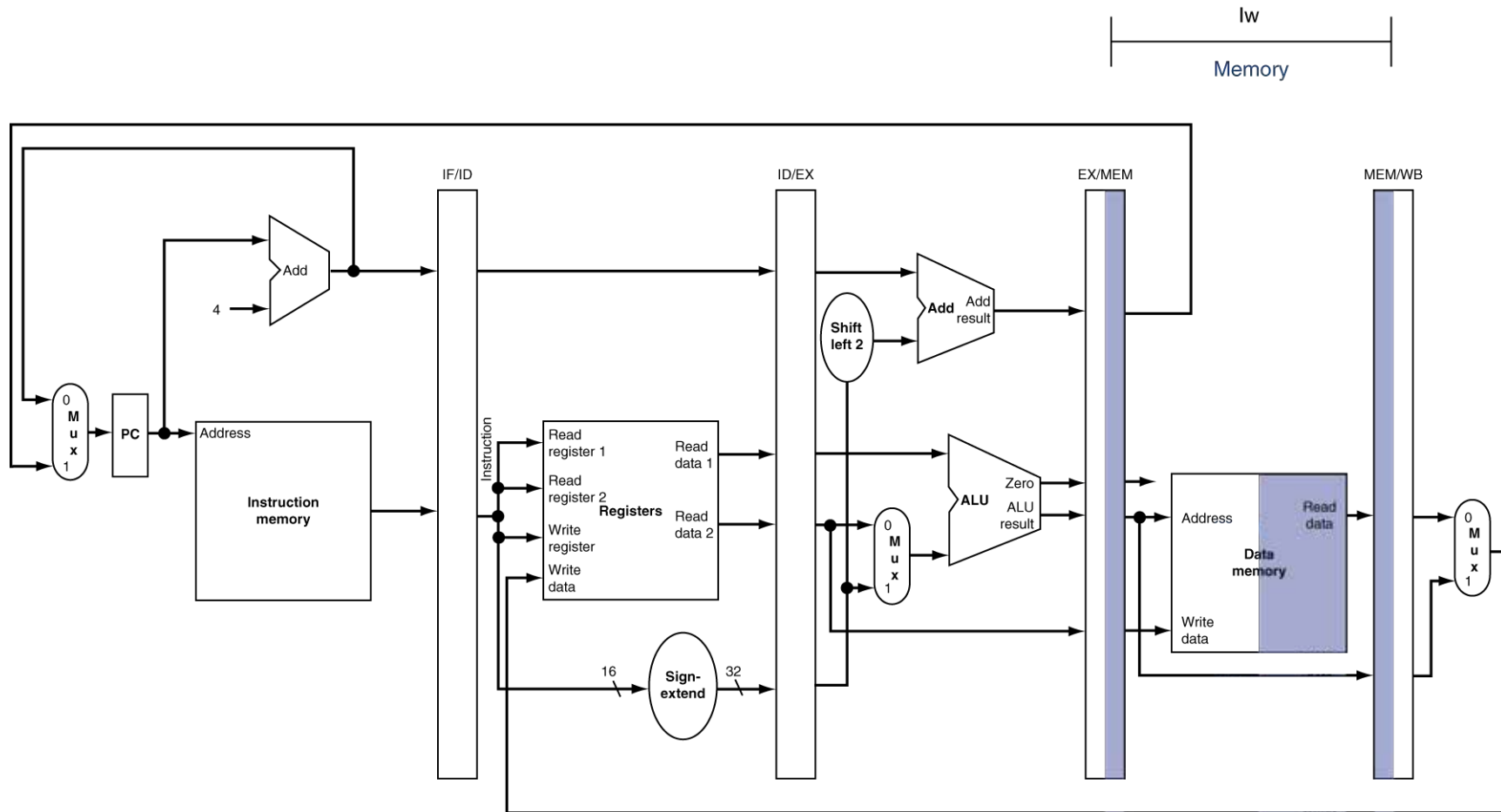
ID for Load, Store, ...



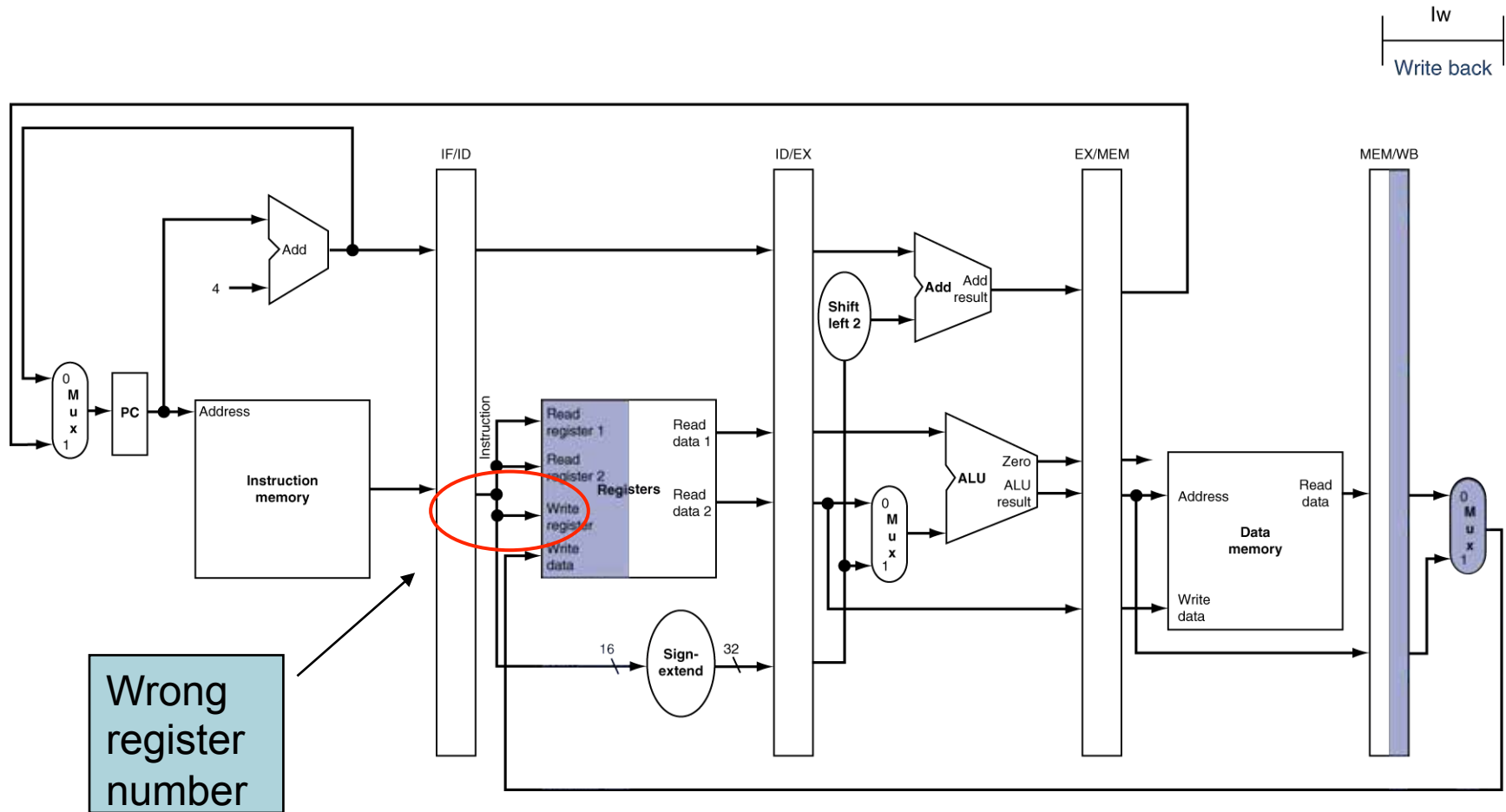
EX for Load



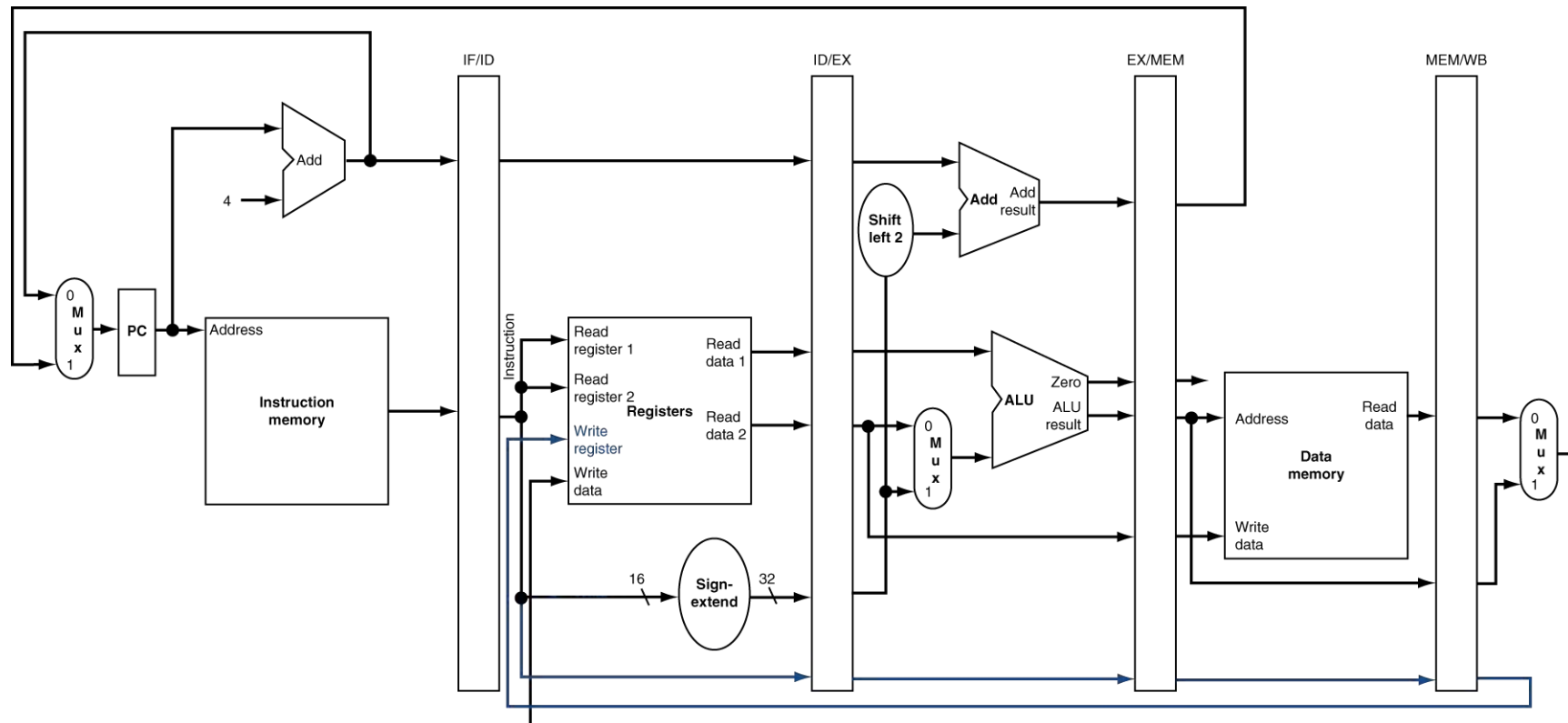
MEM for Load



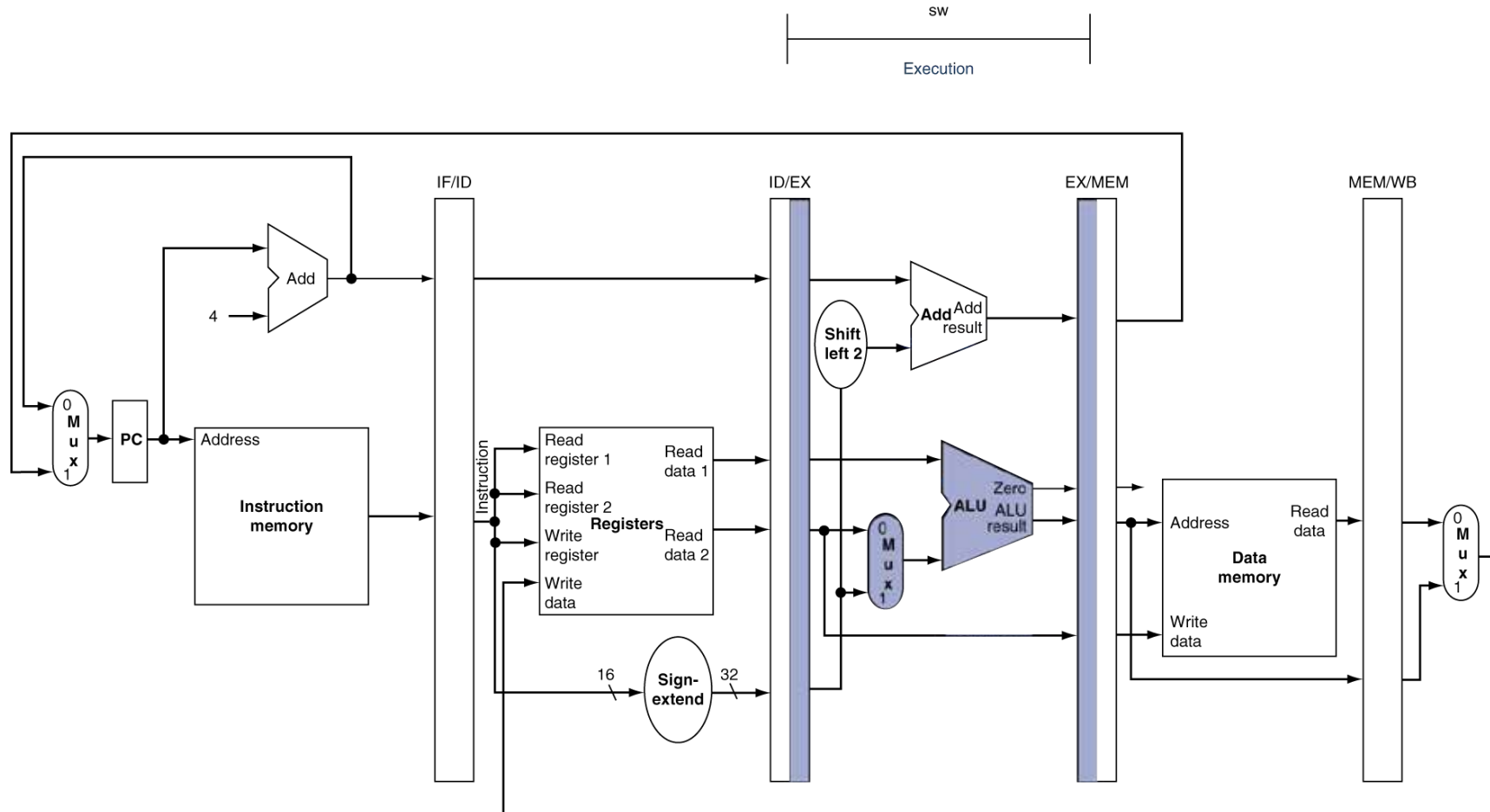
WB for Load



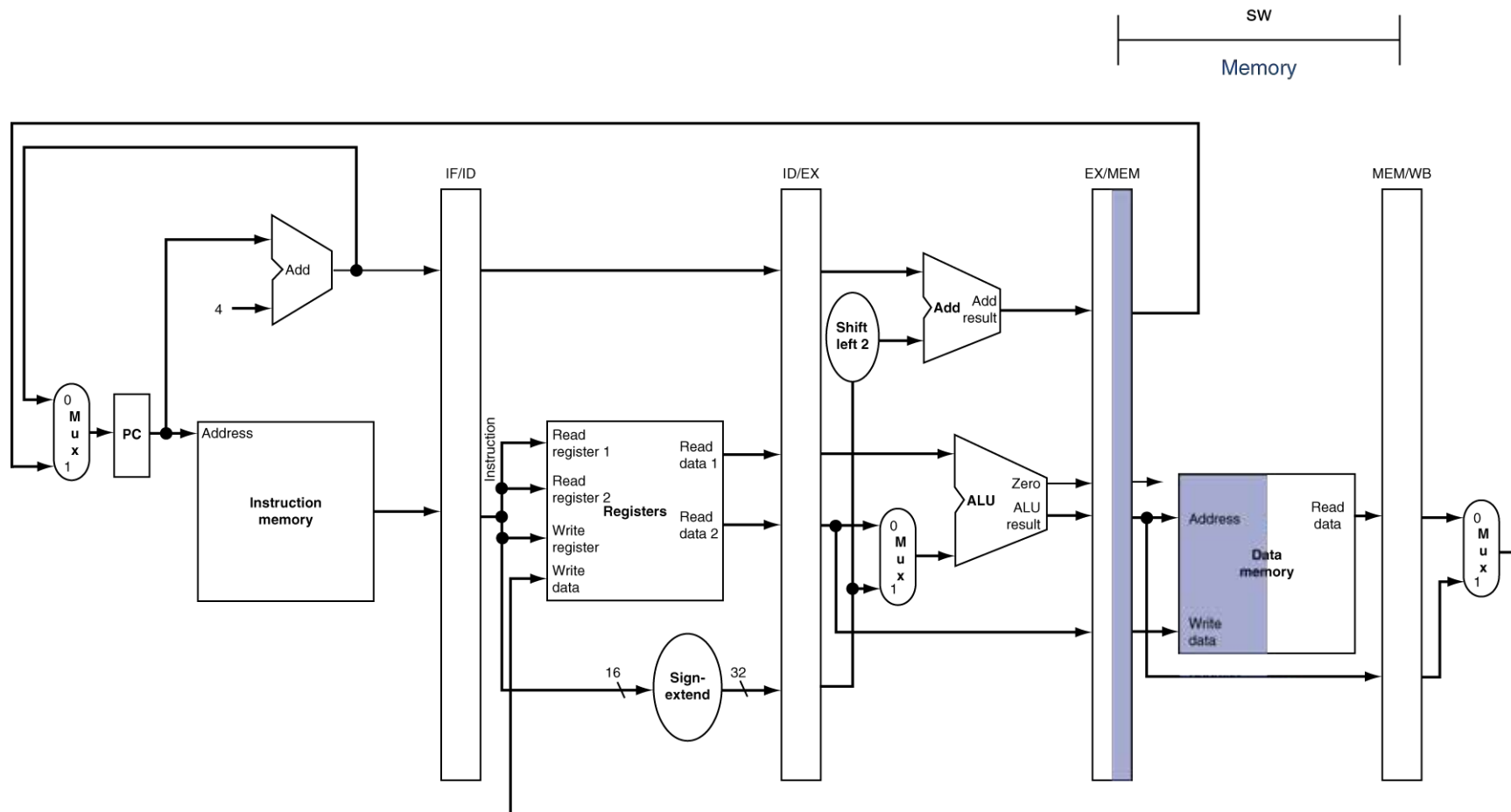
Corrected Datapath for Load



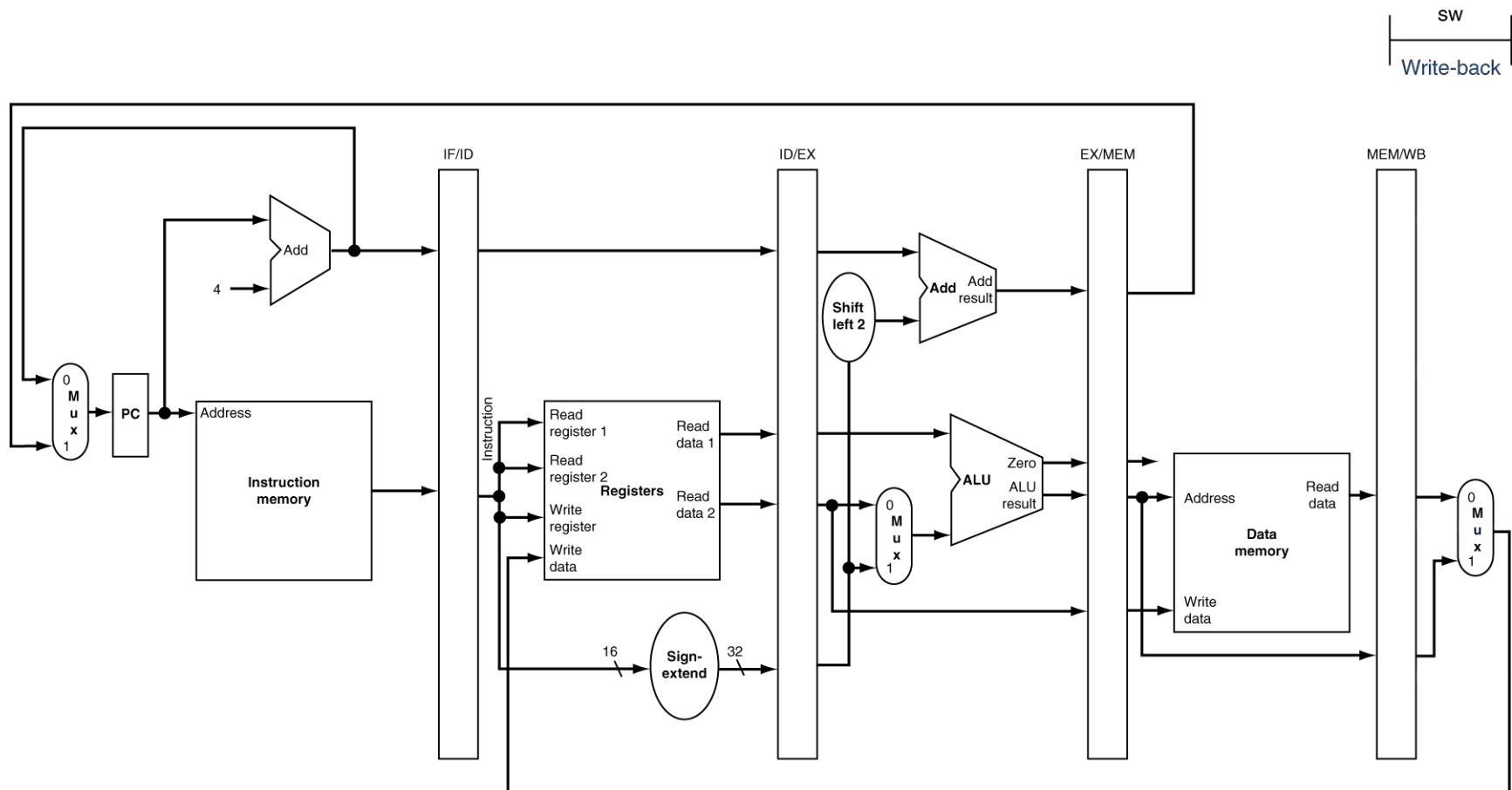
EX for Store



MEM for Store

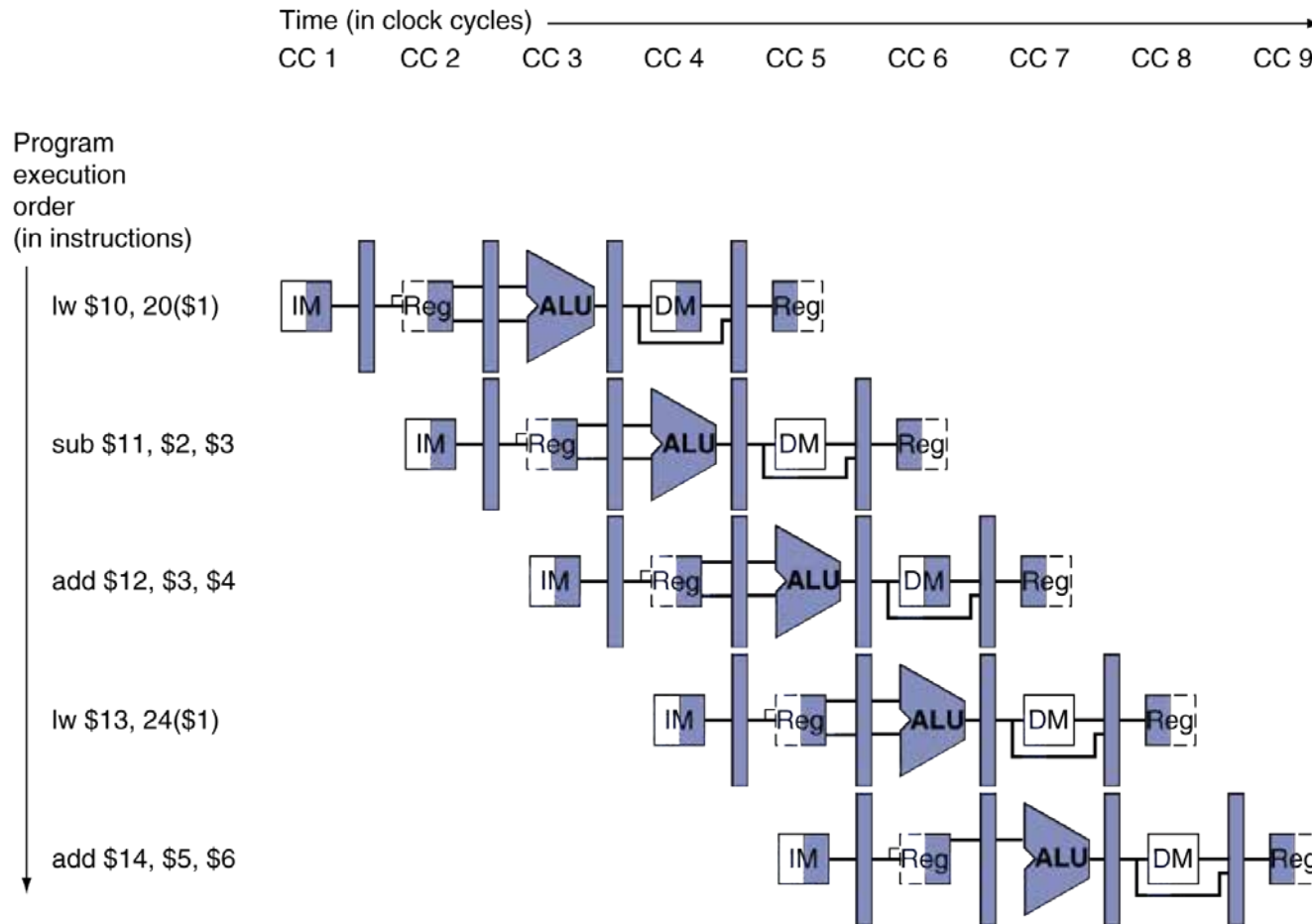


WB for Store



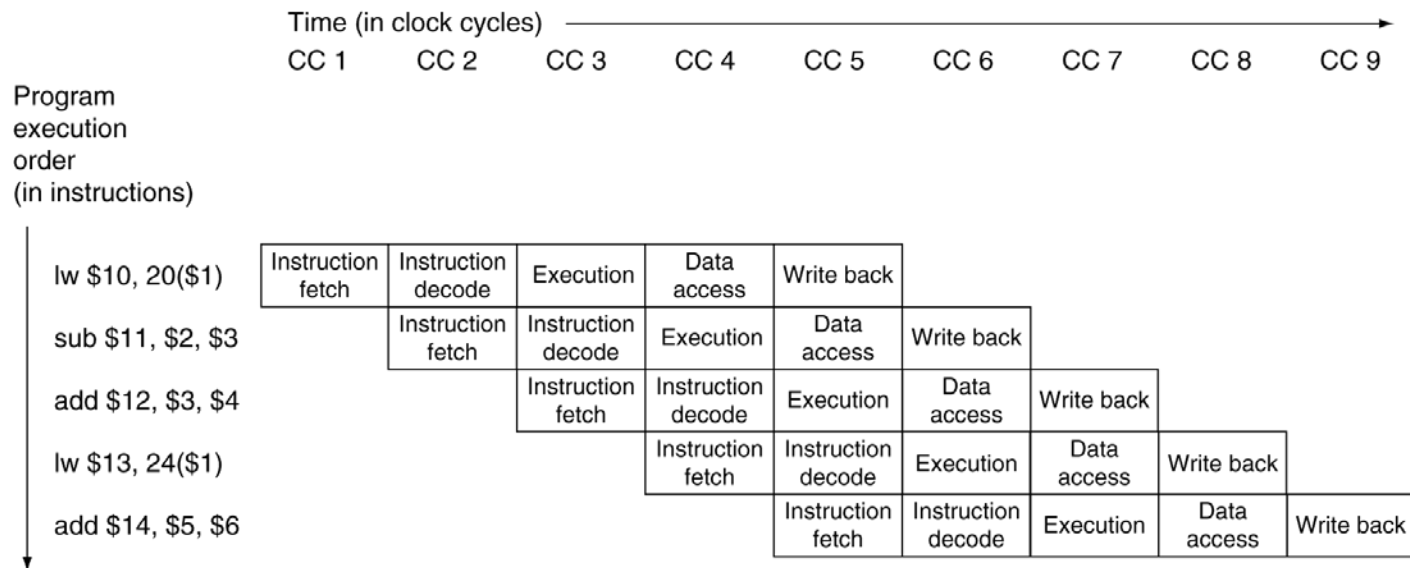
Multi-Cycle Pipeline Diagram

- Form showing resource usage



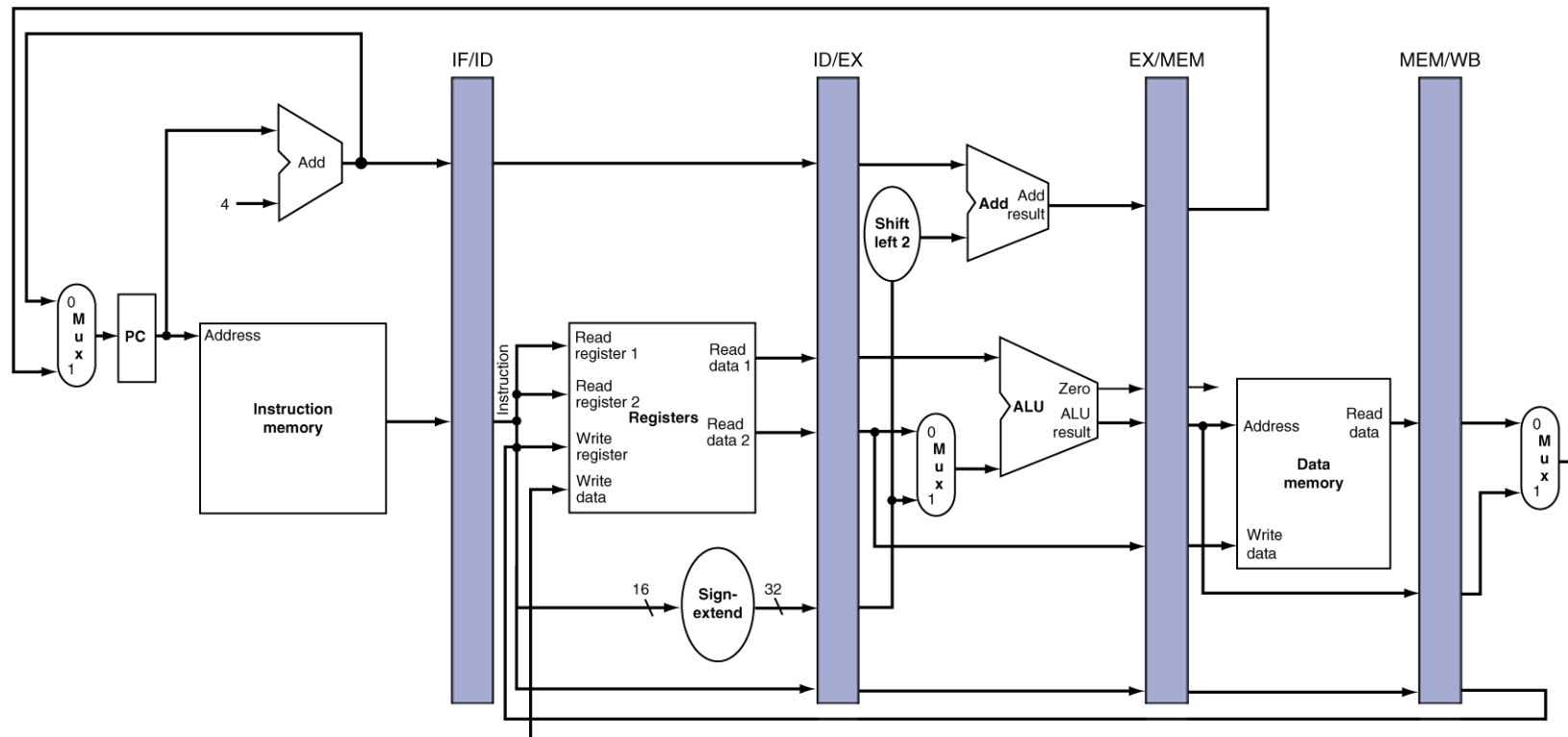
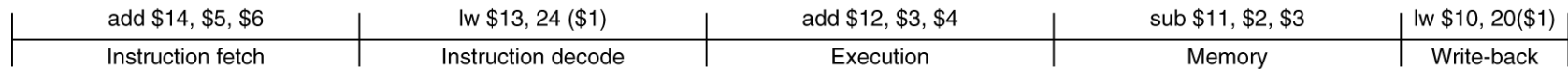
Multi-Cycle Pipeline Diagram

■ Traditional form

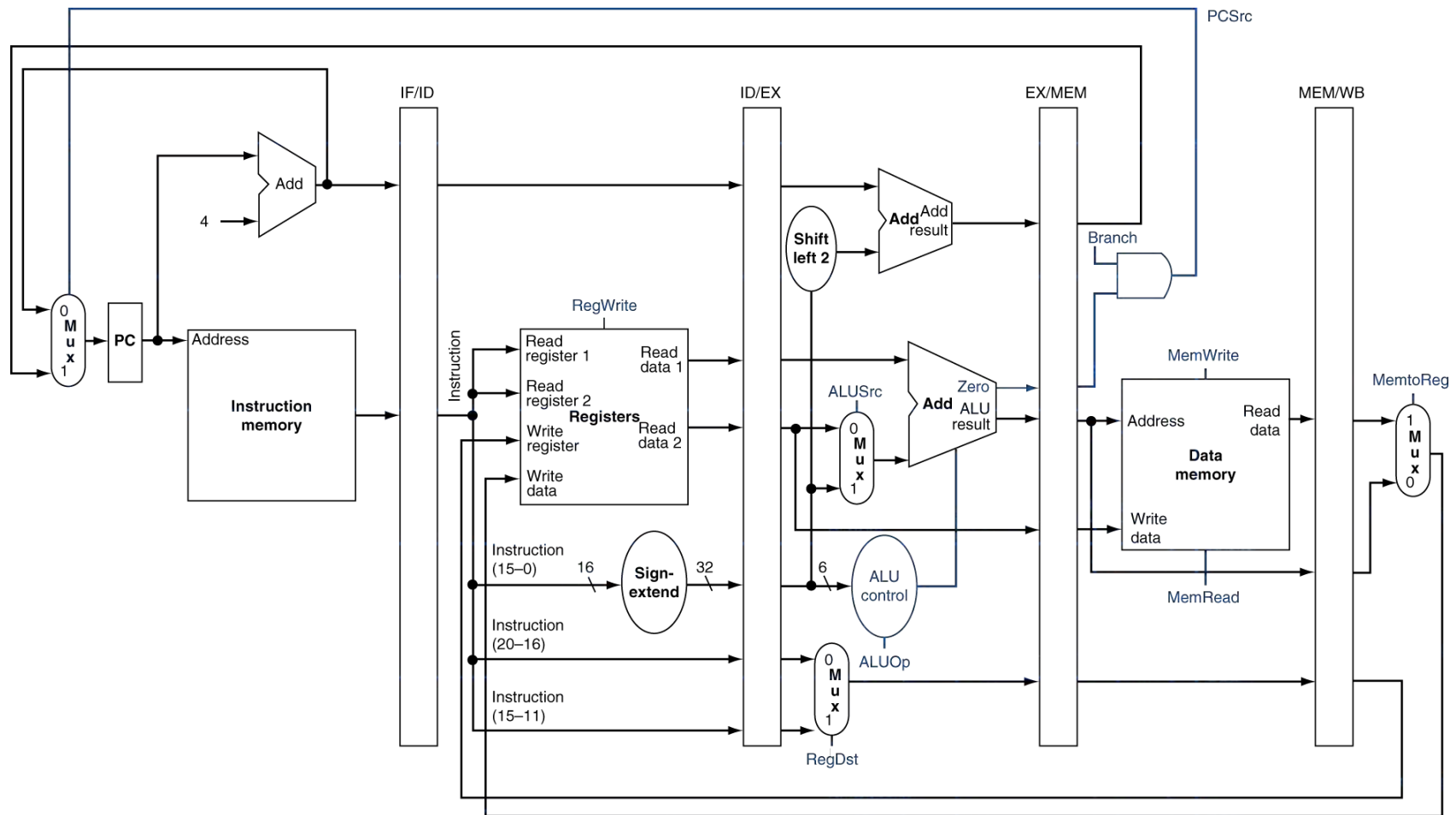


Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle

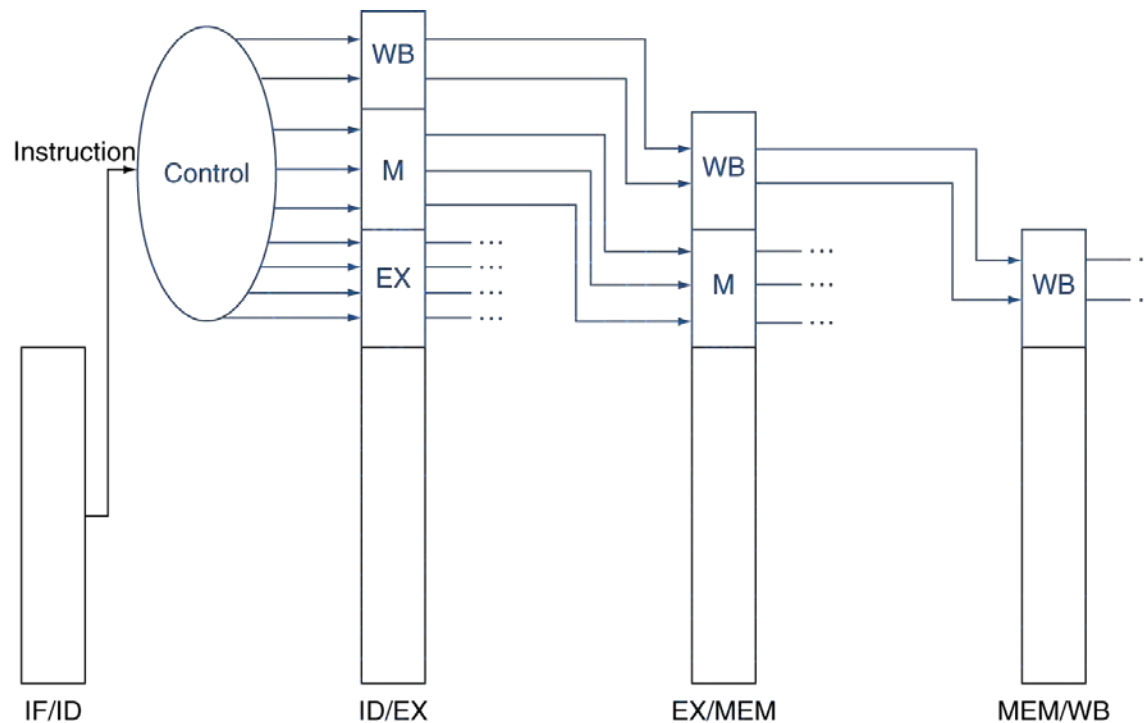


Pipelined Control (Simplified)

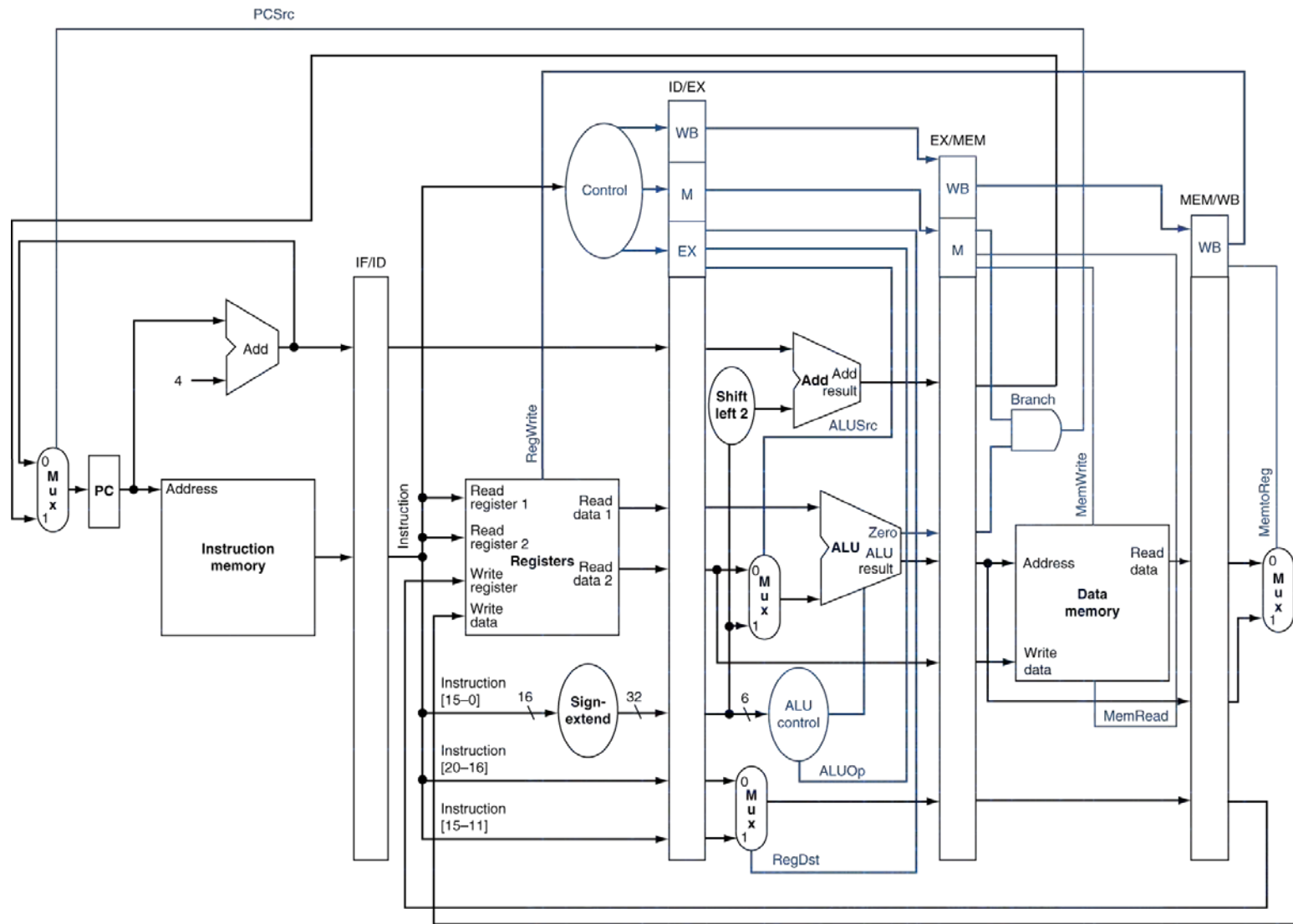


Pipelined Control

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



Pipelined Control



Data Hazards in ALU Instructions

- Consider this sequence:

sub \$2, \$1, \$3

and \$12, \$2, \$5

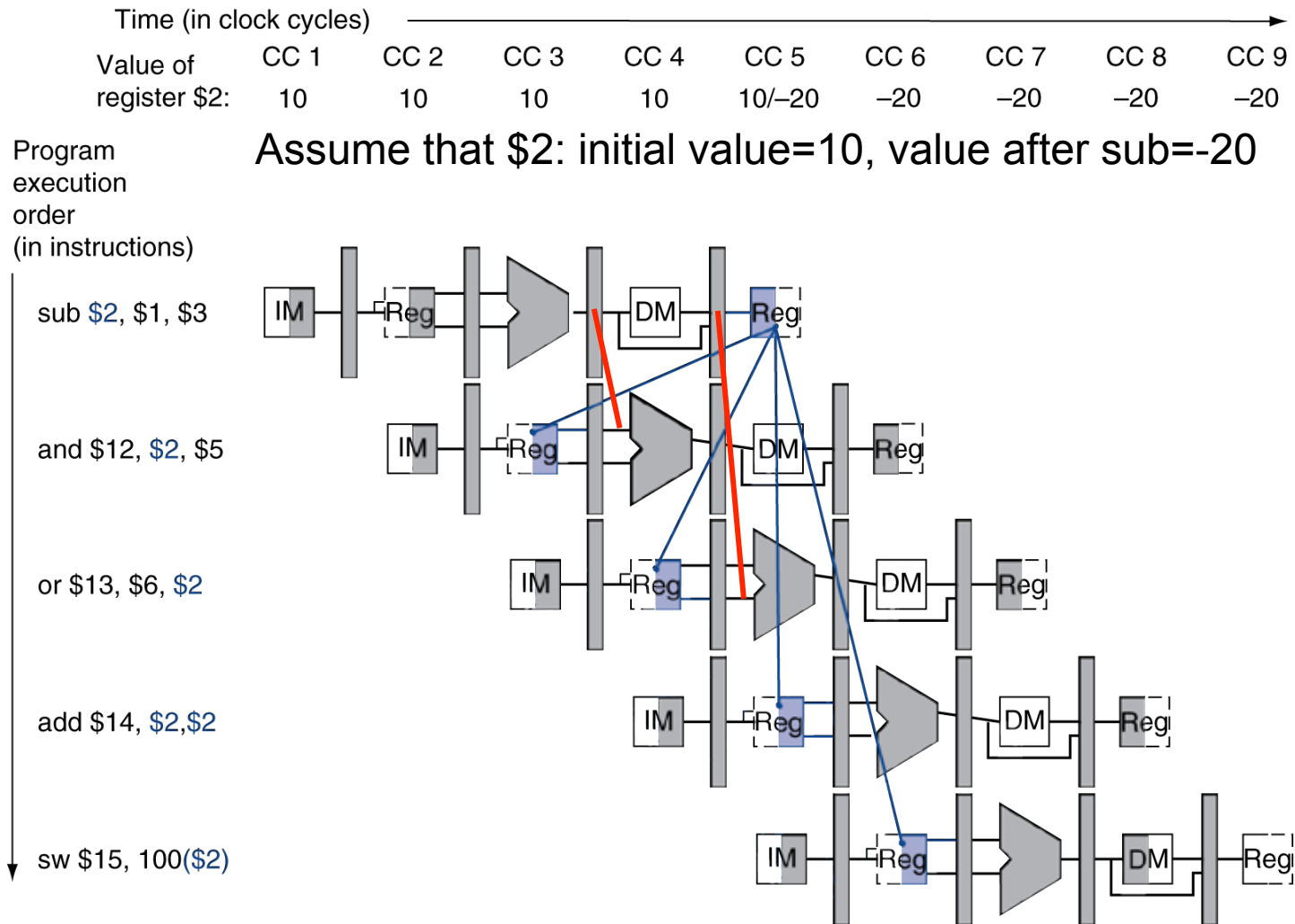
or \$13, \$6, \$2

add \$14, \$2, \$2

sw \$15, 100(\$2)

- We can resolve hazards with forwarding
 - How do we detect when to forward?

Dependencies & Forwarding



Detecting the Need to Forward

- Pass register numbers along pipeline
 - e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
 - ID/EX.RegisterRs, ID/EX.RegisterRt

- Data hazards when

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Fwd from
EX/MEM
pipeline reg

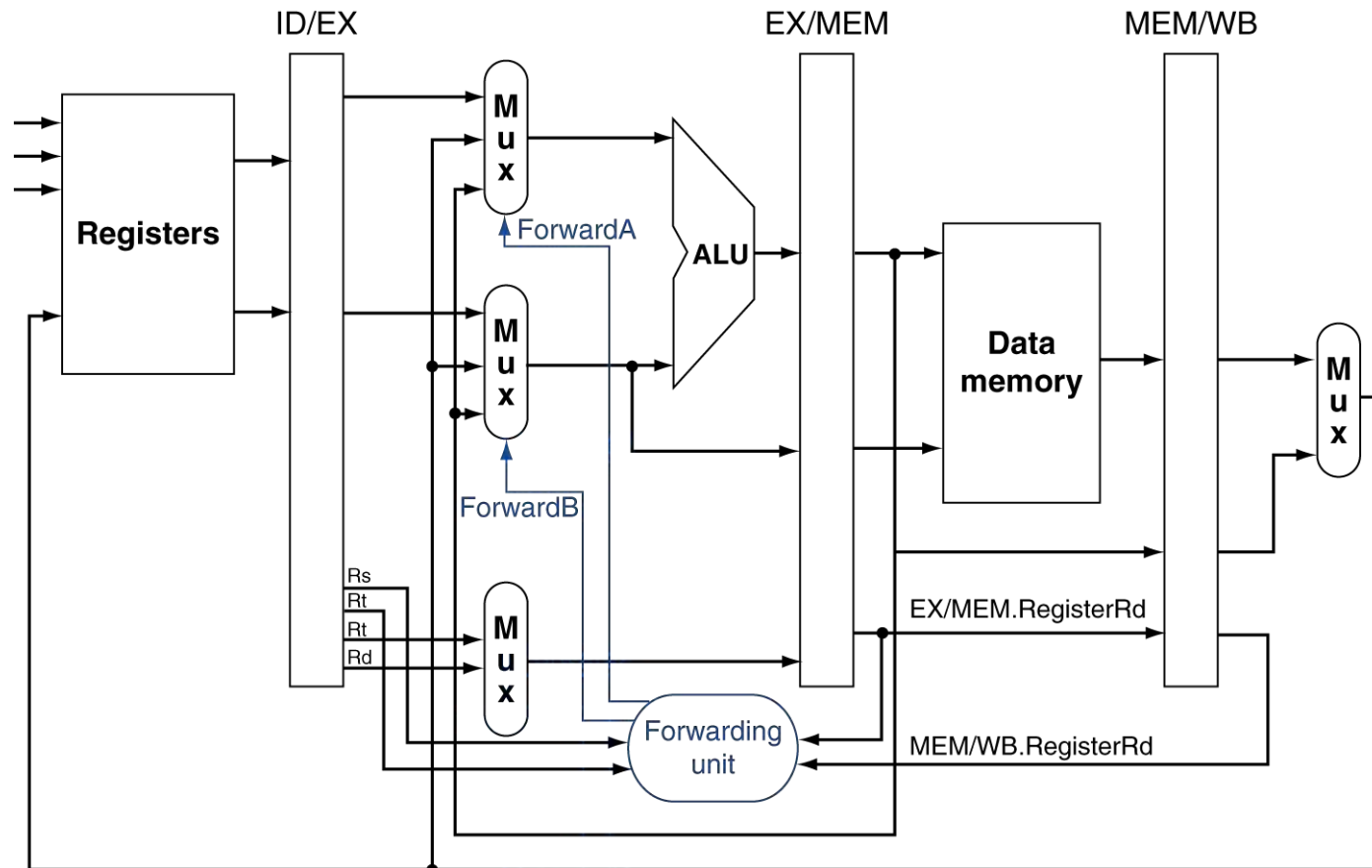
Fwd from
MEM/WB
pipeline reg

**Recall for R-type: add rd, rs, rt, i.e. ALU uses values of rs and rt registers for calculation.*

Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
 - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not \$zero
 - EX/MEM.RegisterRd \neq 0,
MEM/WB.RegisterRd \neq 0

Forwarding Paths

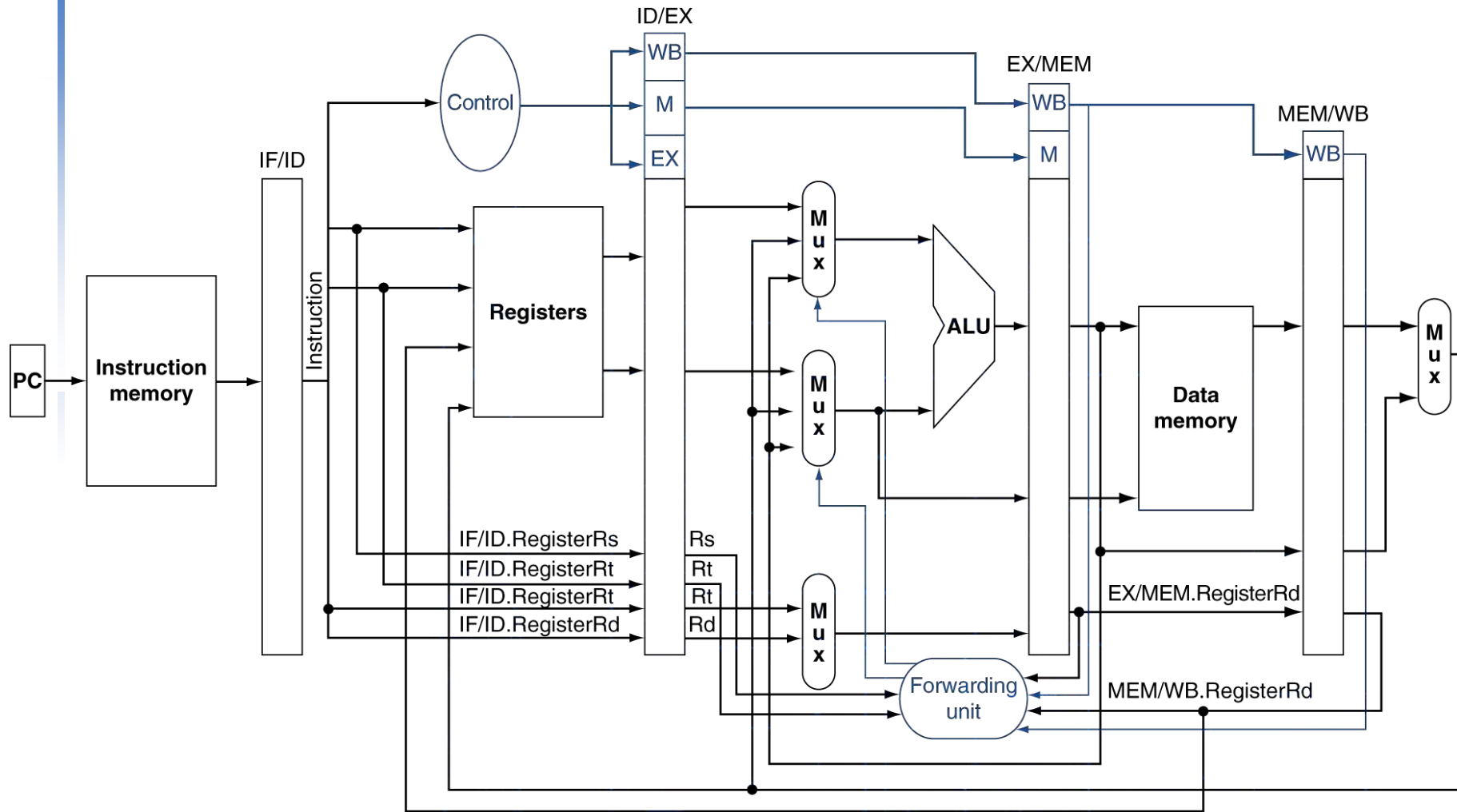


b. With forwarding

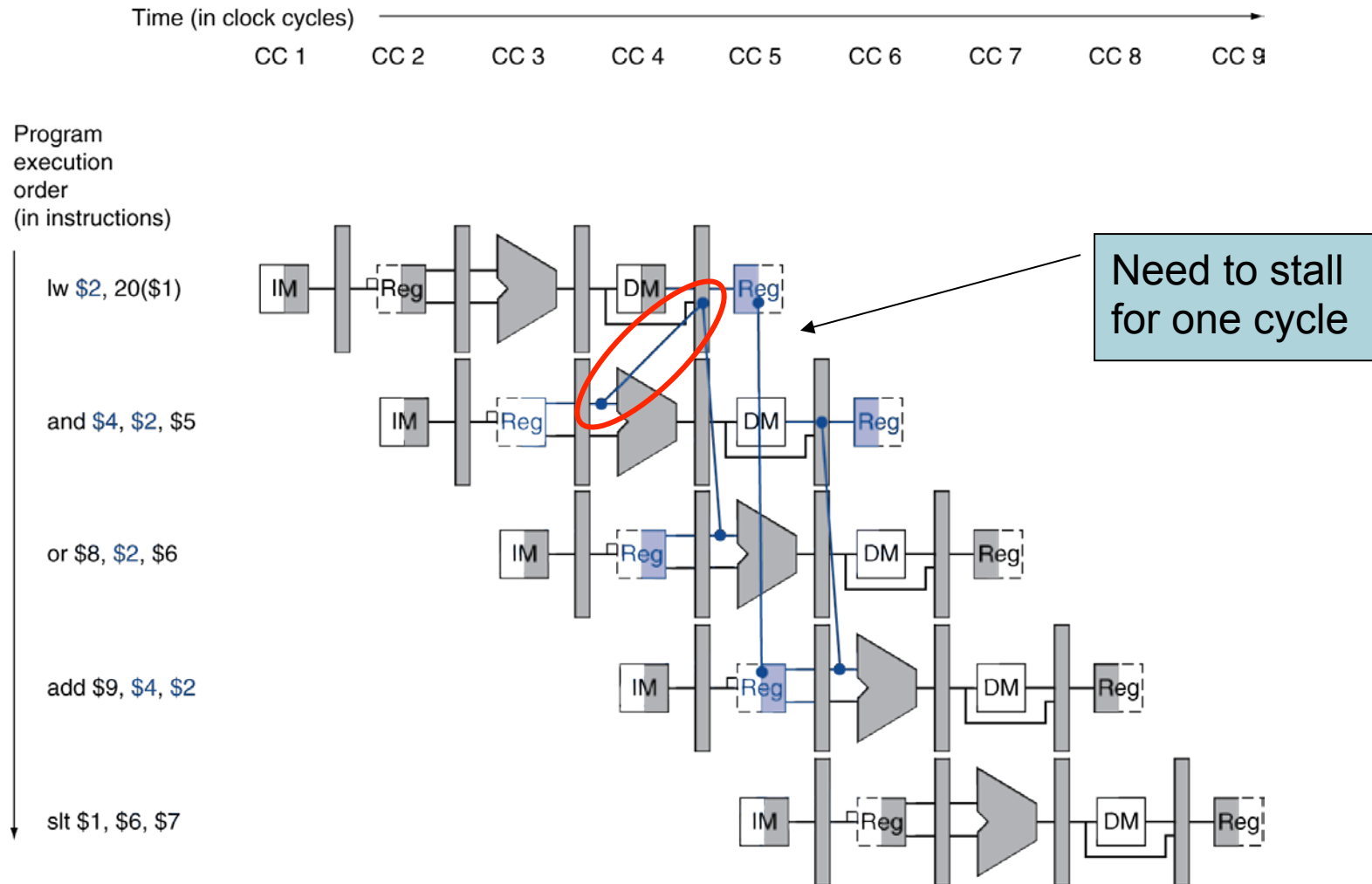
Forwarding Conditions

- EX hazard
 - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
ForwardA = 10
 - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
ForwardB = 10
- MEM hazard
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
ForwardA = 01
 - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd \neq 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
ForwardB = 01

Datapath with Forwarding



Load-Use Data Hazard



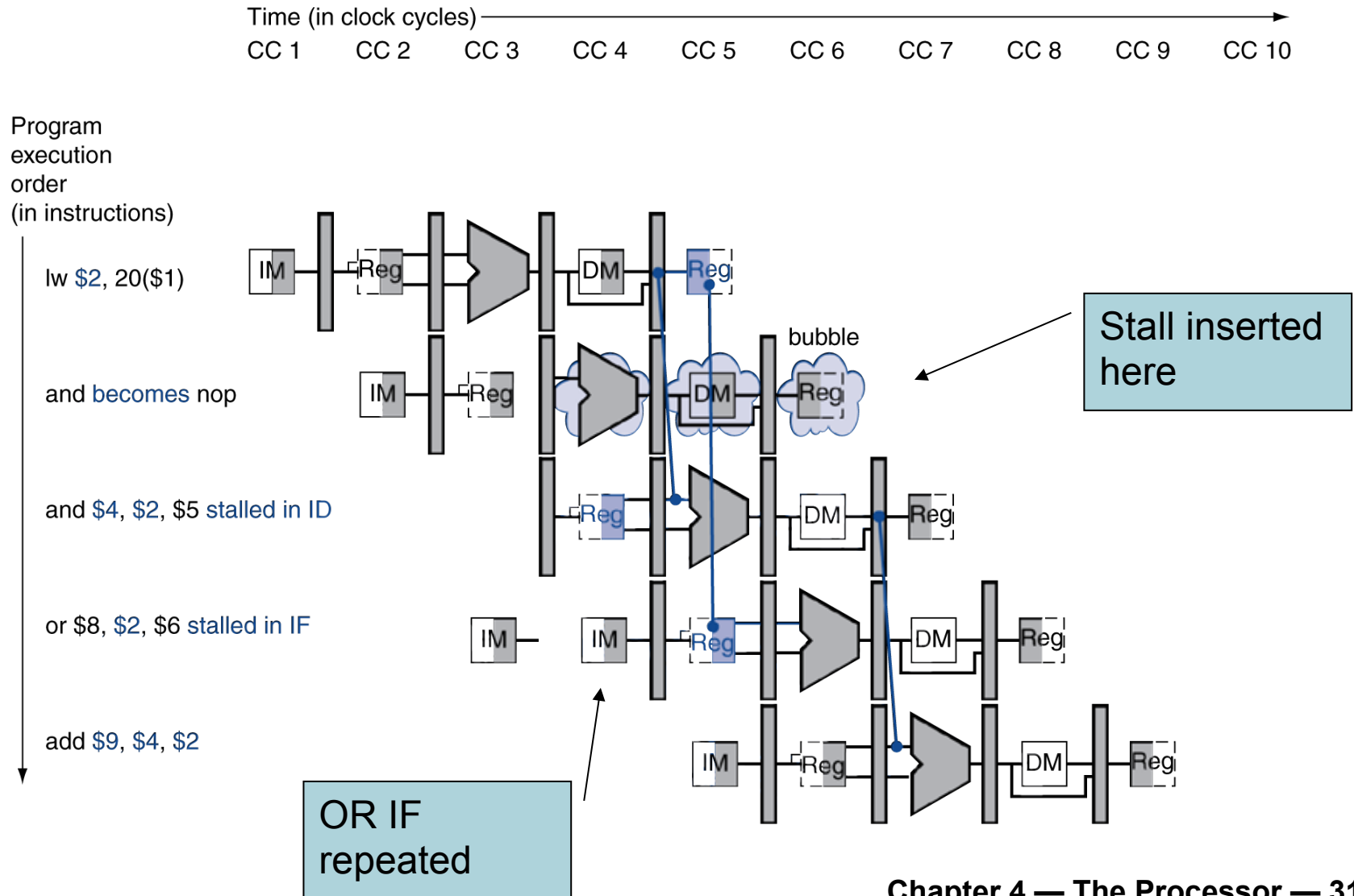
Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
 - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
 - ID/EX.MemRead and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

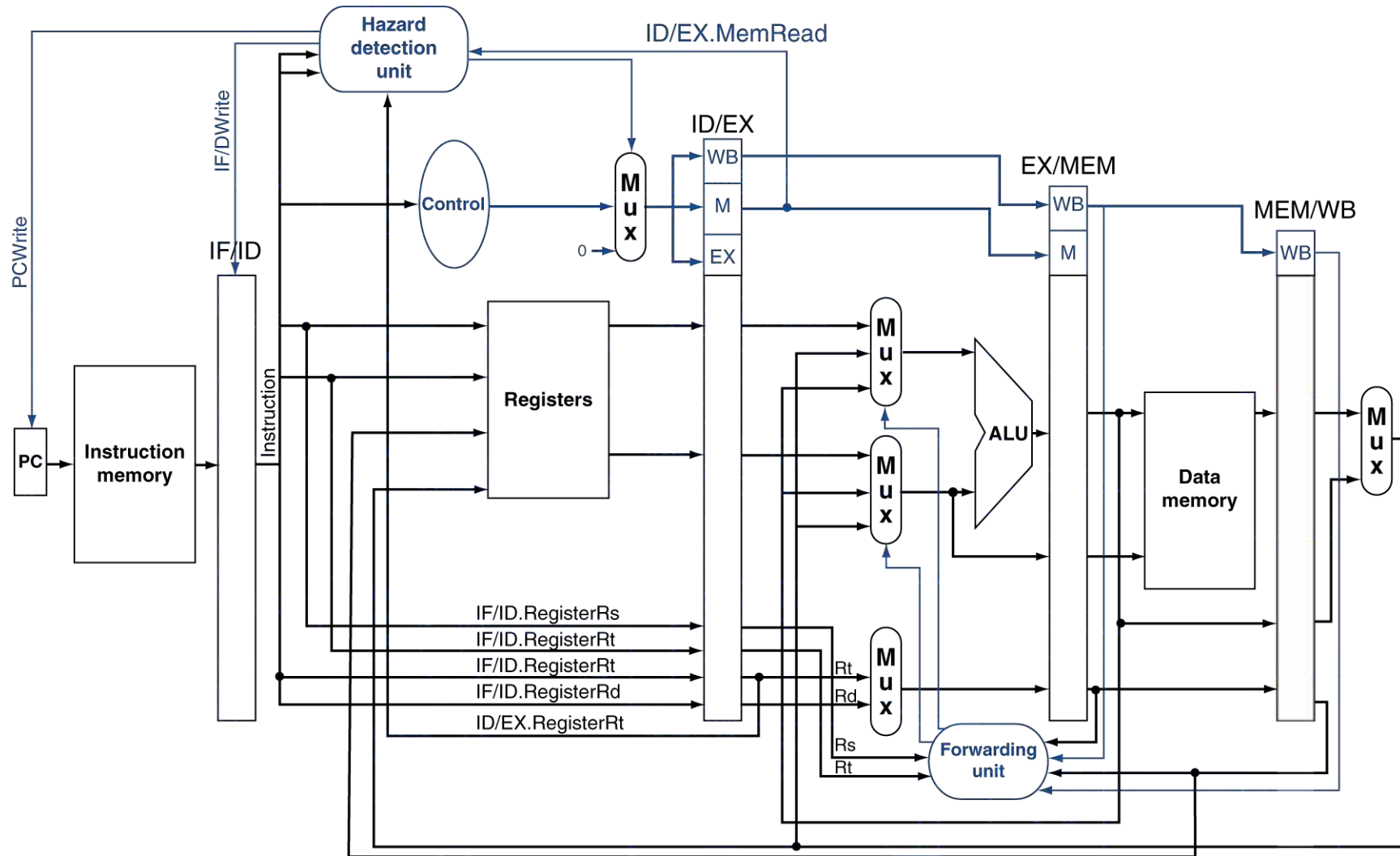
How to Stall the Pipeline

- Force control values in ID/EX register to 0
 - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
 - Using instruction is decoded again
 - Following instruction is fetched again
 - 1-cycle stall allows MEM to read data for $1w$
 - Can subsequently forward to EX stage

Stall/Bubble in the Pipeline



Datapath with Hazard Detection

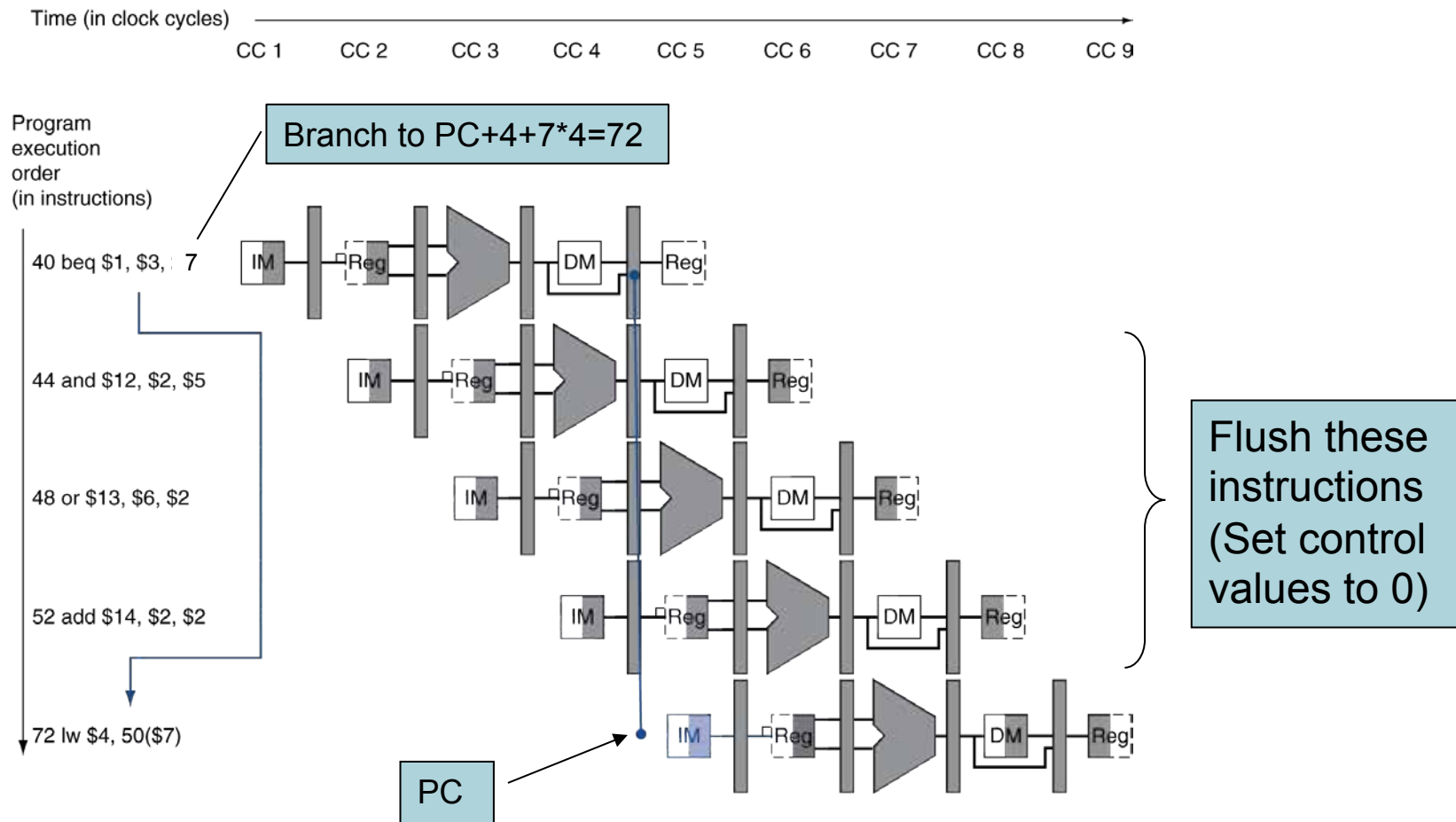


Stalls and Performance

- Stalls reduce performance
 - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
 - Requires knowledge of the pipeline structure

Branch Hazards

- If branch outcome determined in MEM



Solution to Control Hazard

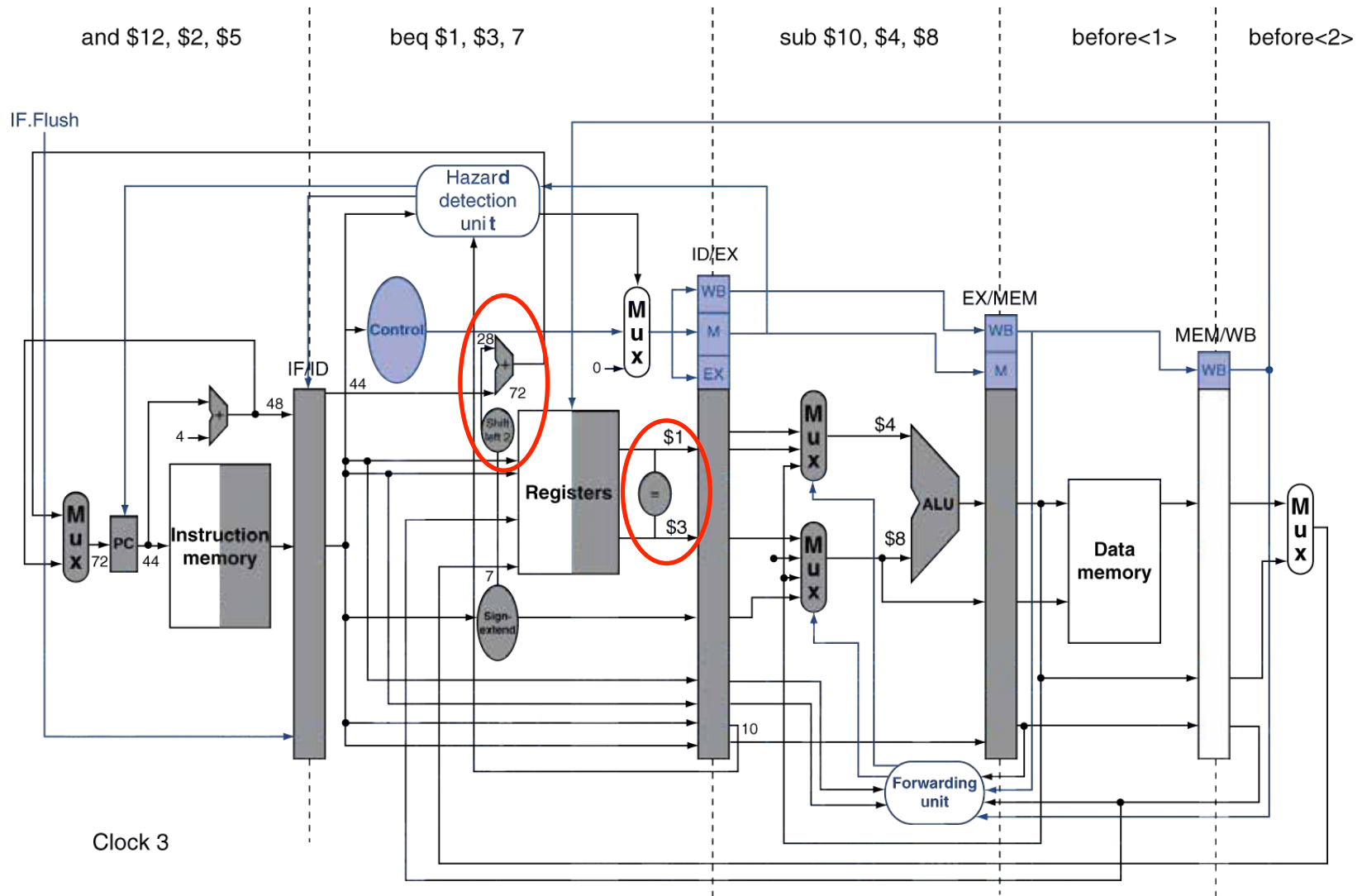
- Example: branch taken

```
36:  sub  $10, $4, $8
40:  beq  $1,  $3,  7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
```

```
    ...
72:  lw   $4, 50($7)
```

- Assume additional hardware to determine outcome of branch in ID stage
 - Target address adder: $PC+4+4*7=72$
 - Register comparator: e.g. if $\$1=\3

Example: Branch Taken



Example: Branch Taken

