CMPE 200

COMPUTER ARCHITECTURE

Lecture 4 – The Processor – Pipelining

Bo Yu
Computer Engineering Department
SJSU

Adapted from Computer Organization and Design, 5th Edition, 4th edition, Patterson and Hennessy, MK and Computer Architecture – A Quantitative Approach, 4th edition, Patterson and Hennessy, MK

Lecture 4A Key Concepts Review

- Last Lecture:
  - **4.5 Overview of Pipelining**

Lecture 4A Key Concepts Review

- Polling: Pipeline

Lecture 4A Key Concepts Review

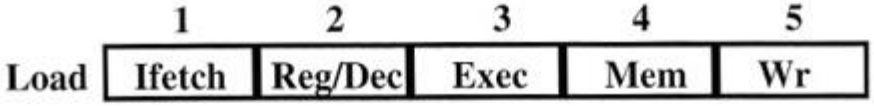
- What is pipelining
  - A technique (for achieving high performance) whereby multiple instructions are overlapped in execution.
  - It's not visible to the programmer
- Throughput of a pipeline
  - Number of instructions that can leave the pipeline each cycle
- Latency
  - The time needed for an instruction to pass through all pipeline stages
  - The 5-stage pipeline takes 5 clock cycles for the instruction to complete
- Pipeline Speedup
  - If all stages are perfectly balanced, Time between instructions$_{pipelined}$ = Time between instructions$_{nonpipelined}$ / Number of stages
  - Speedup due to increased instruction throughput, latency (time for each instruction) does not decrease
- Pipeline Hazards
  - Structure hazards (resource hazard)
  - Data hazards (pipeline data hazard)
  - Control hazards (branch hazard)
- Structure hazards solution
  - Resources requirement for pipelined datapath, i.e., separate instruction and data memory
- Data hazards solution
  - Forwarding (bypassing): retrieve the missing data from internal buffers rather than waiting for it to arrive from registers or memory (require extra connection in datapath)
  - Pipeline stall (bubble): i.e. load-use data hazard
- Control hazards solution
  - Static branch prediction
  - Dynamic branch prediction

Pipelining Definitions

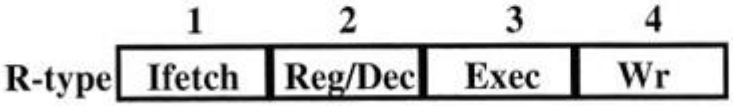
- Pipeline: an implementation technique by which multiple instructions are overlapped in execution. It is not visible to the programmer.
  - ✓ Each stage is called a pipe stage
- Pipeline machine cycle: time required to move an instruction one step down the pipe
- Throughput of a pipeline: number of instructions that can leave the pipeline each cycle
- Latency: the time needed for an instruction to pass through all pipeline stages

Pipelining Important Observation

- Each functional unit can only be used once per instruction
- Each functional unit must be used at the same stage for all instructions:
  - Load uses Register File's Write Port during its 5th stage

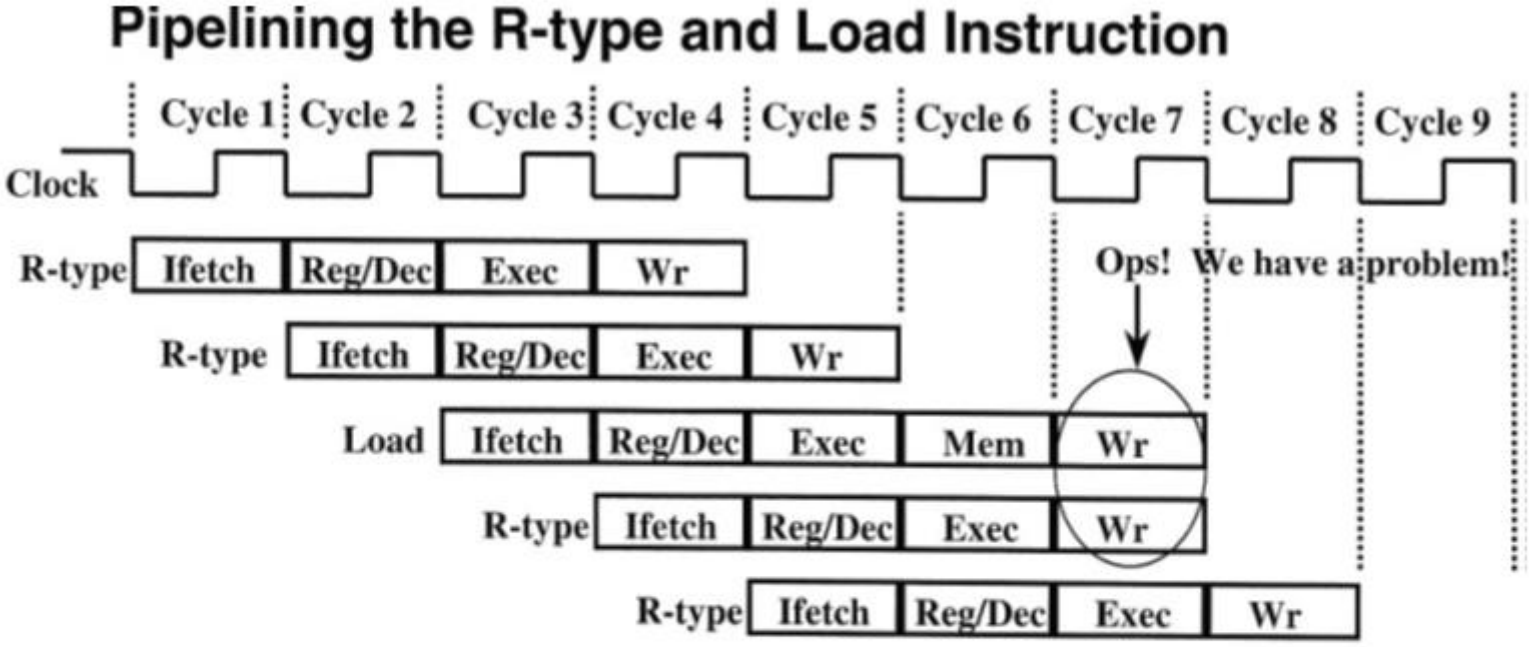


  - R-type uses Register File's Write Port during its 4th stage



- 2 ways to solve this pipeline hazard.

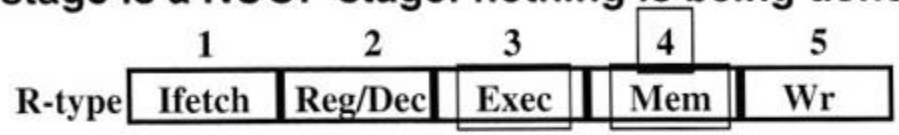
Pipelining Important Observation (resource contention)

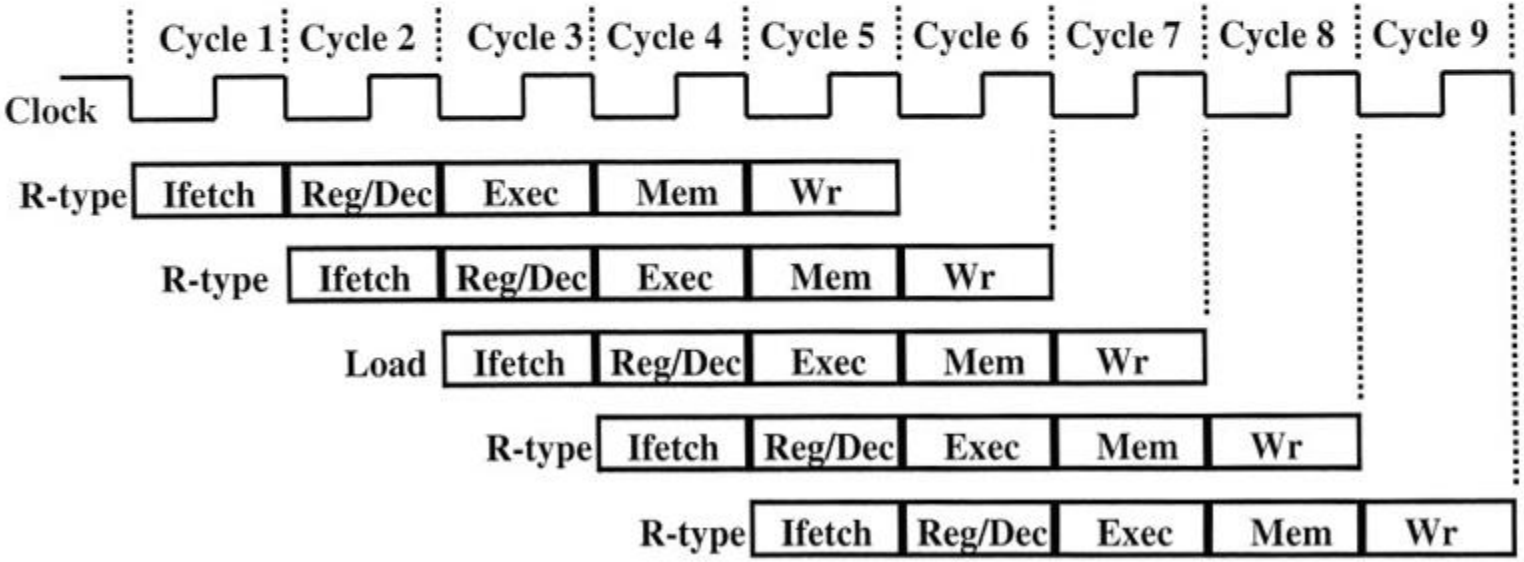


- We have pipeline conflict or structural hazard:
  - Two instructions try to write to the register file at the same time!
  - Only one write port

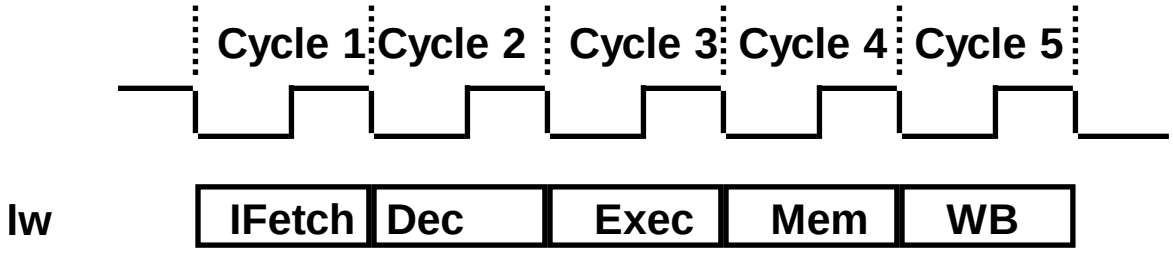
Solution: Delay R-type's Write by One Cycle

- Delay R-type's register write by one cycle:
  - Now R-type instructions also use Reg File's write port at Stage 5
  - Mem stage is a NOOP stage: nothing is being done.





- Five MIPS pipeline stages of Load, one step per stage



- **IF**etch: Instruction fetch from memory and update PC
- **ID**ec: Instruction decode & registers read
- **EX**ec: Execute R-type or calculate memory address
- **MEM**: Read/write the data from/to the Data Memory
- **WB**: Write the result data back into the register file

- Today's Lecture (CH4.6-4.7):
  - Pipelined Datapath and Control
  - Data Hazards: Forwarding vs. Stalling

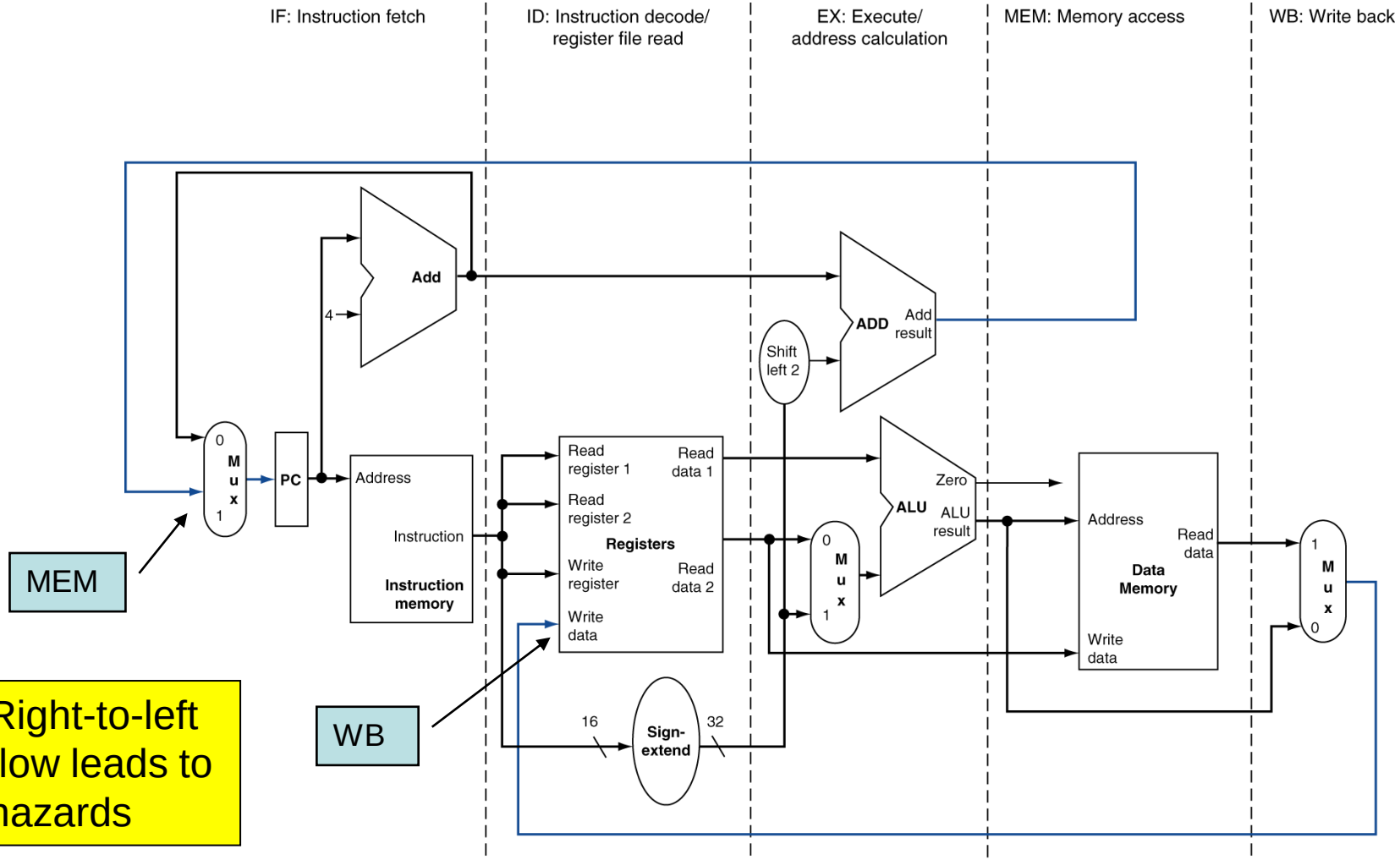
Five Stages Pipeline

- IF: Instruction Fetch
- ID: Instruction decode and register file read
- EX: Execution or address calculation
- MEM: Data memory access
- WB: Write back

An instruction -> five stages -> up to five instructions will be in execution during any single clock cycle

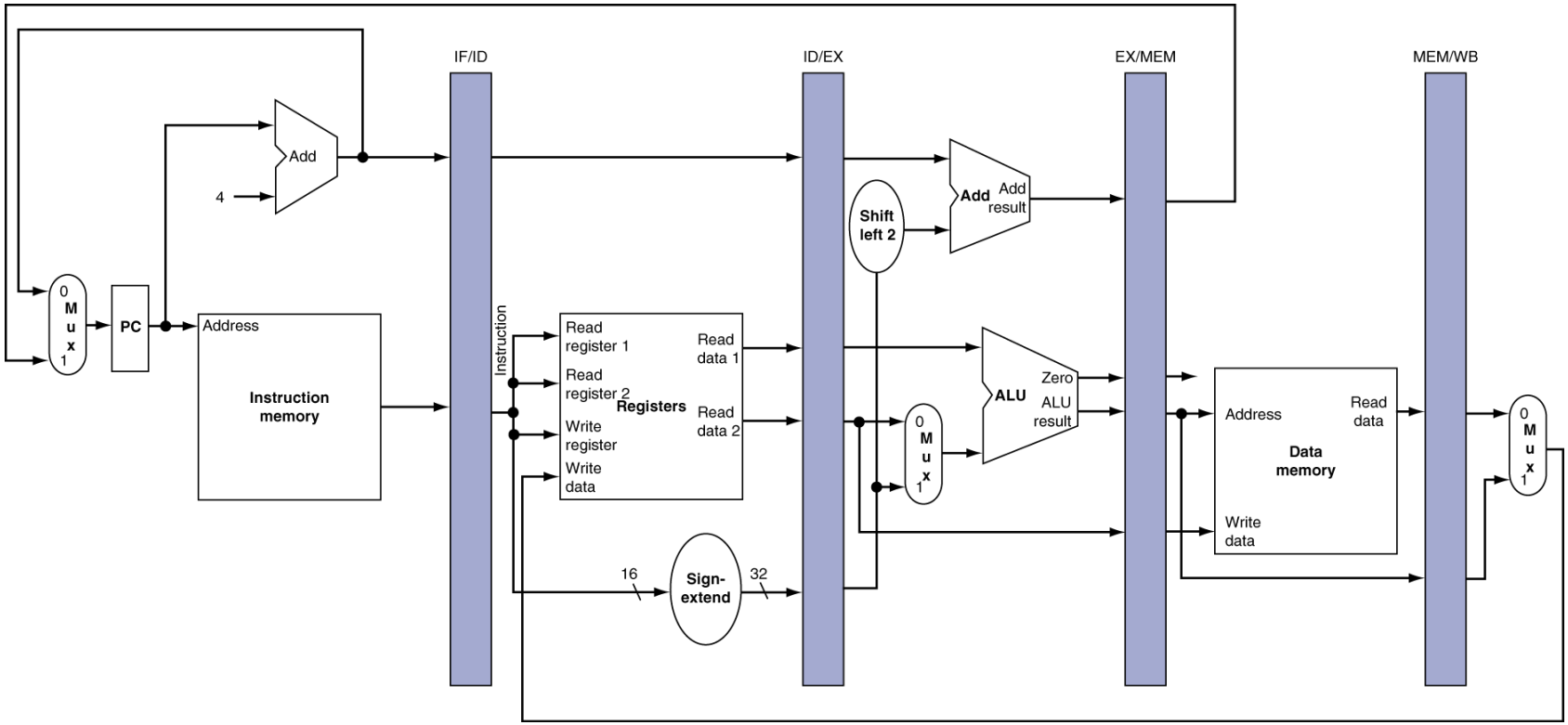
Datapath -> five stages corresponding to the stages of instruction execution

MIPS Pipelined Datapath with Five Stages Identified



Pipeline Registers

- Need registers between stages
  - To hold information produced in previous cycle
  - Must be wide enough to store the data passing through

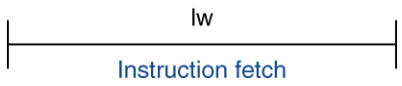


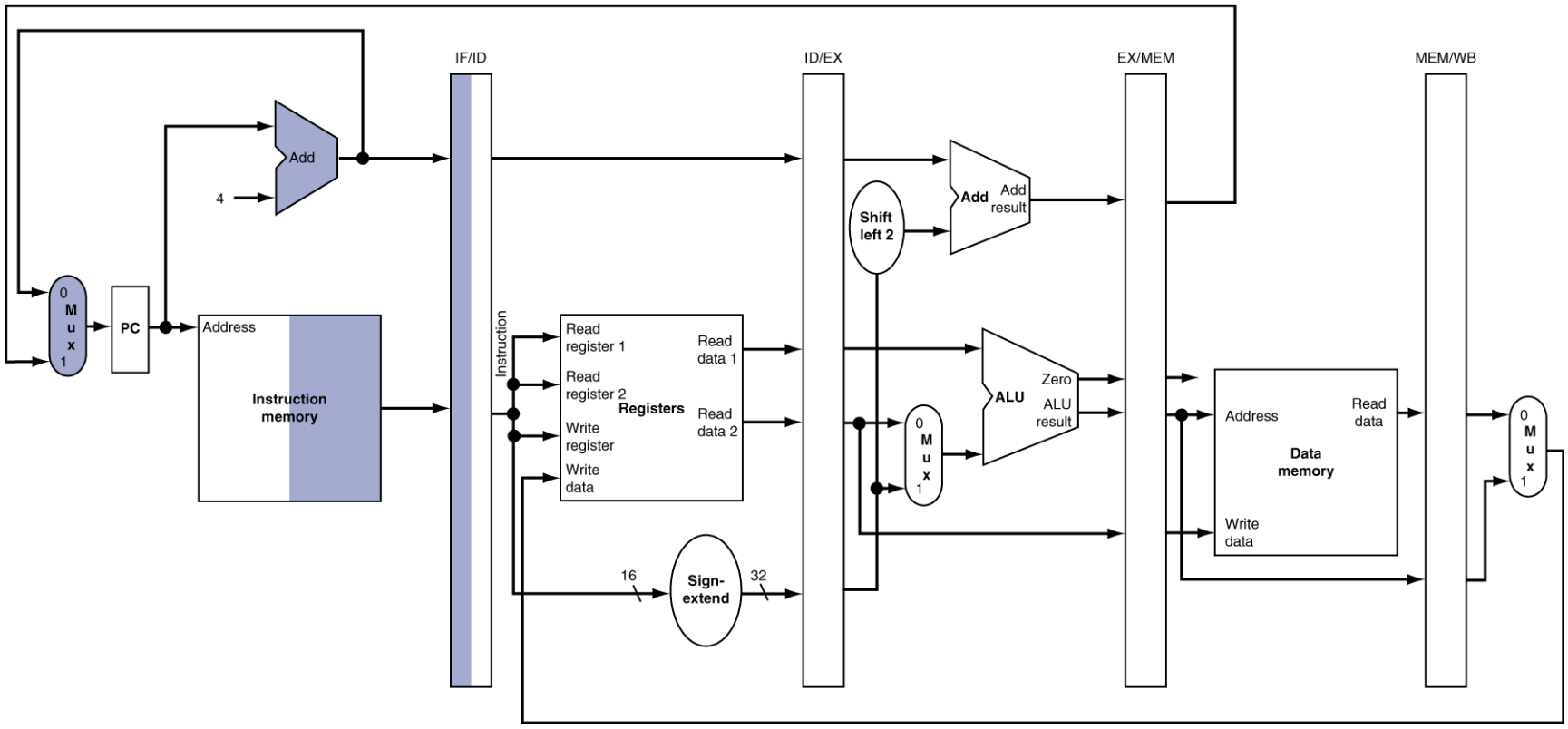
Pipeline Operation

- 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

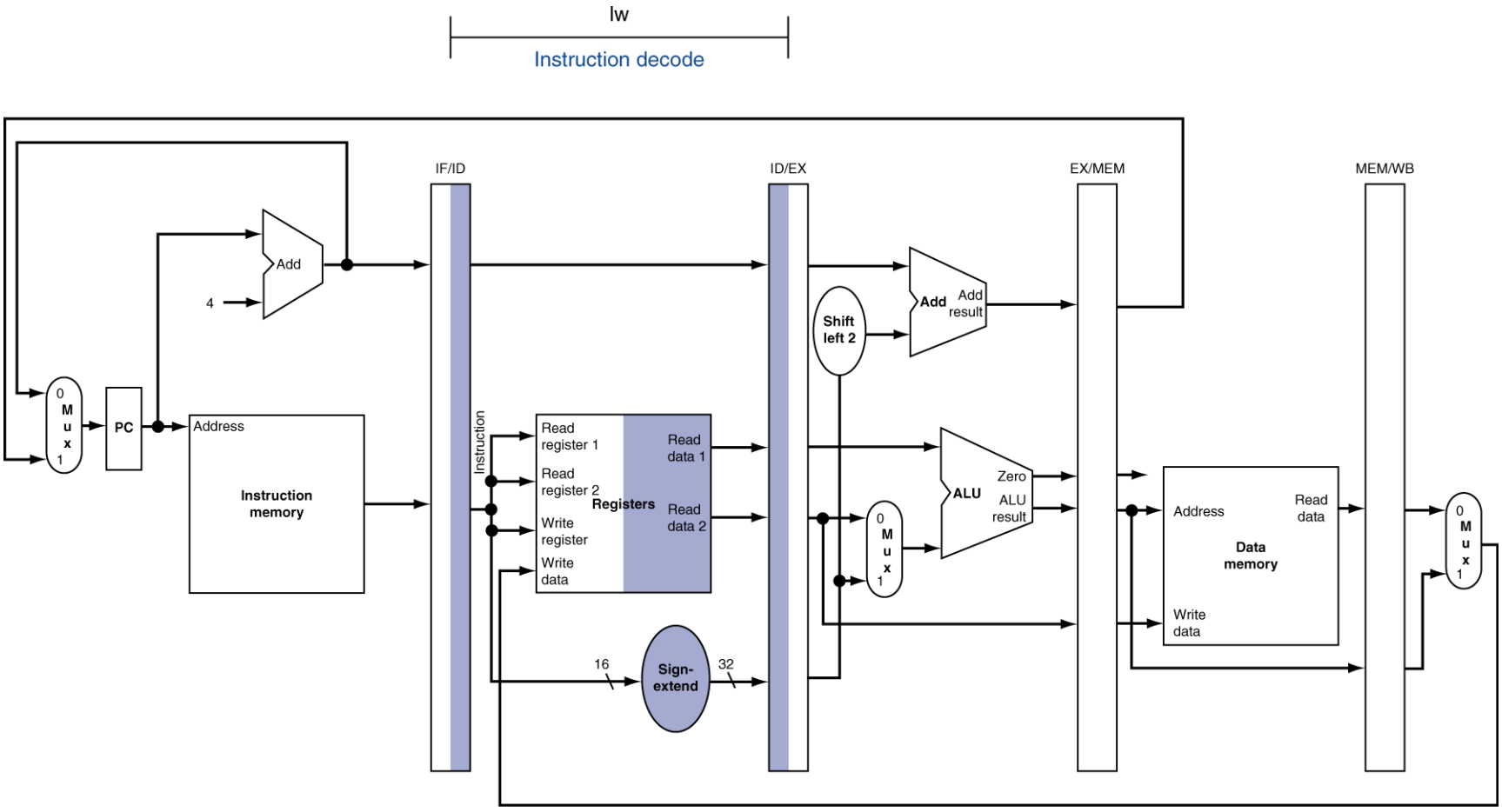
Lecture 4: The Processor - Pipelining

- IF for Load, Store, …



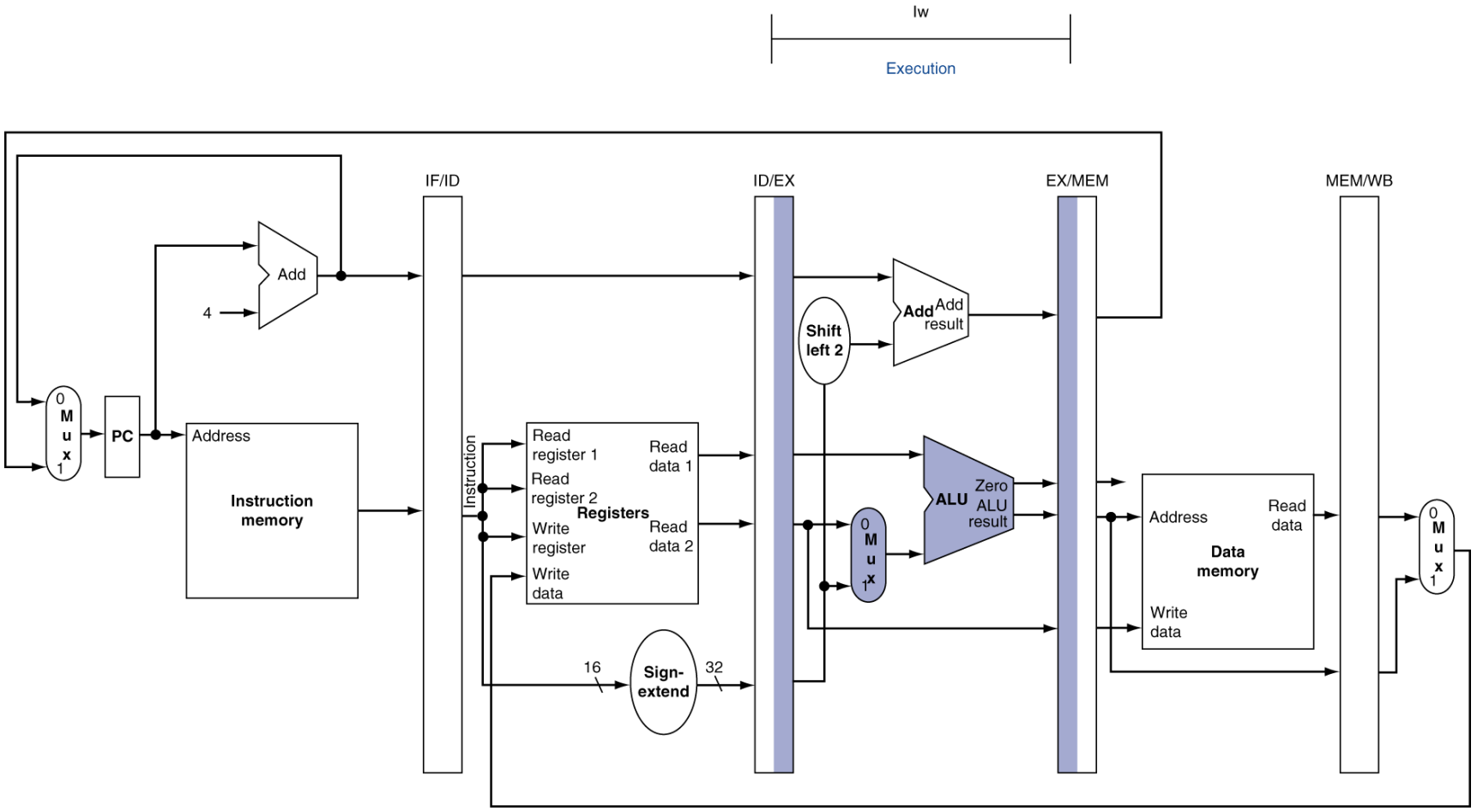


- ID for Load, Store, …

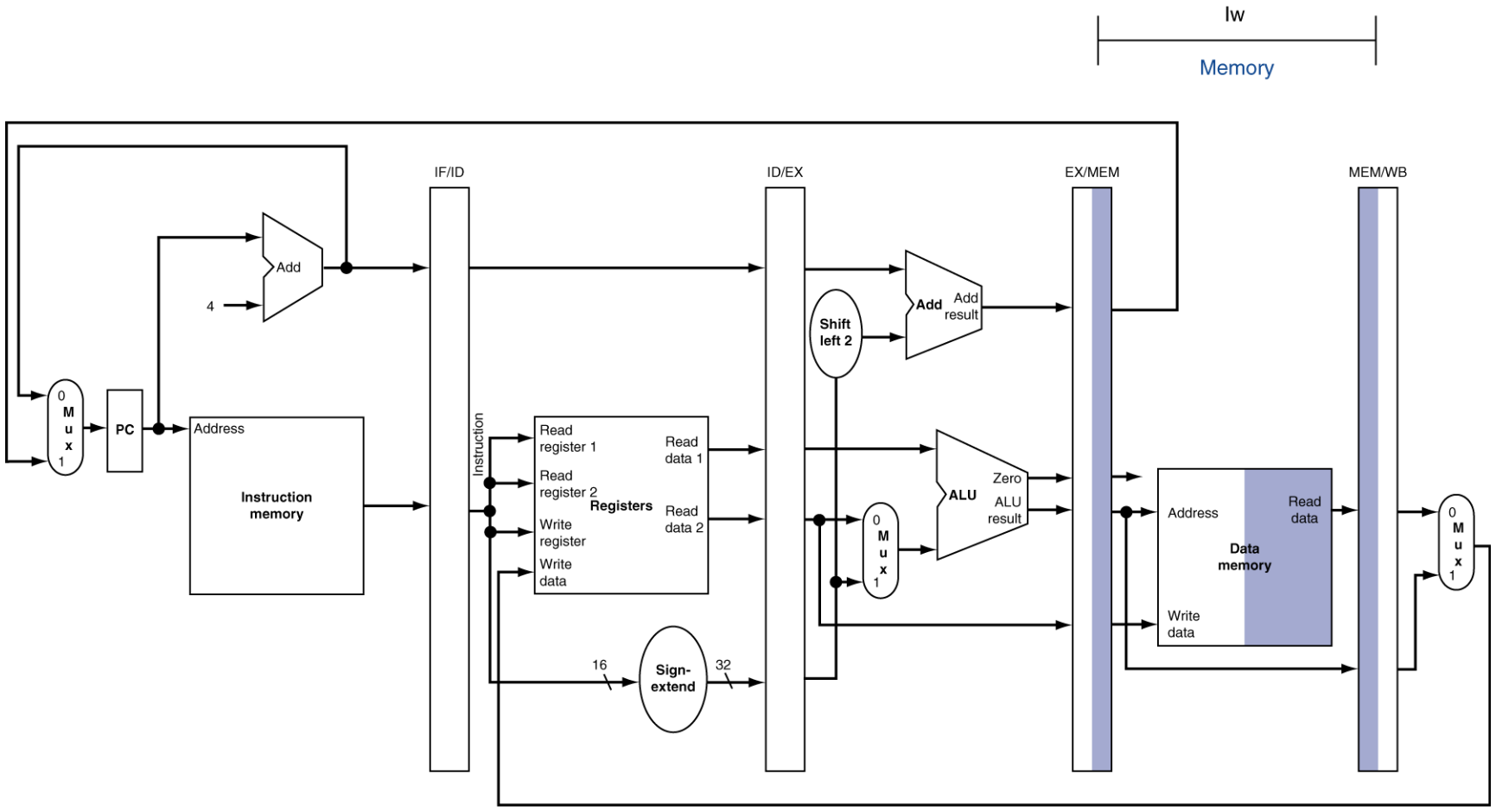


Lecture 4: The Processor - Pipelining

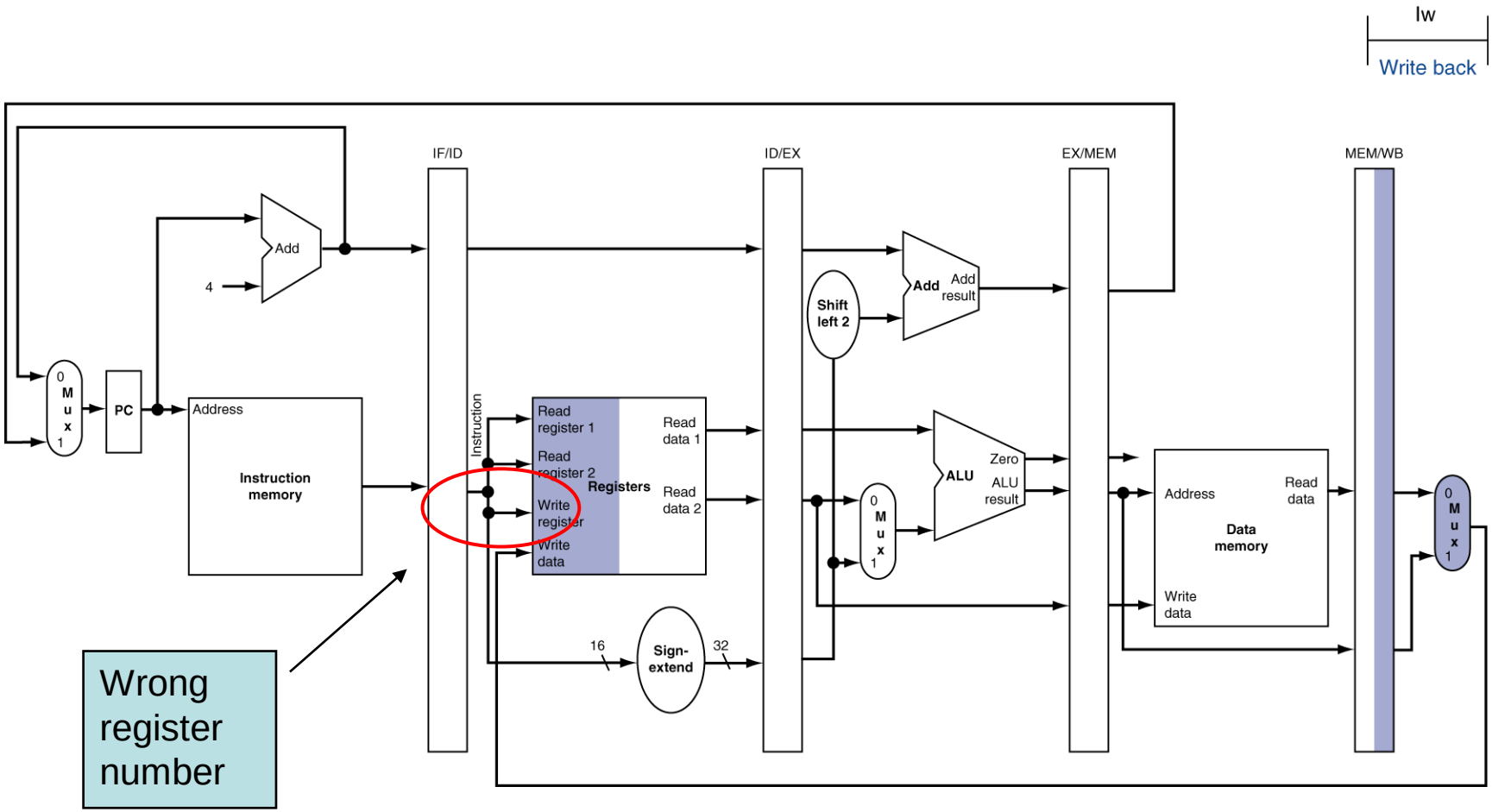
- EX for Load



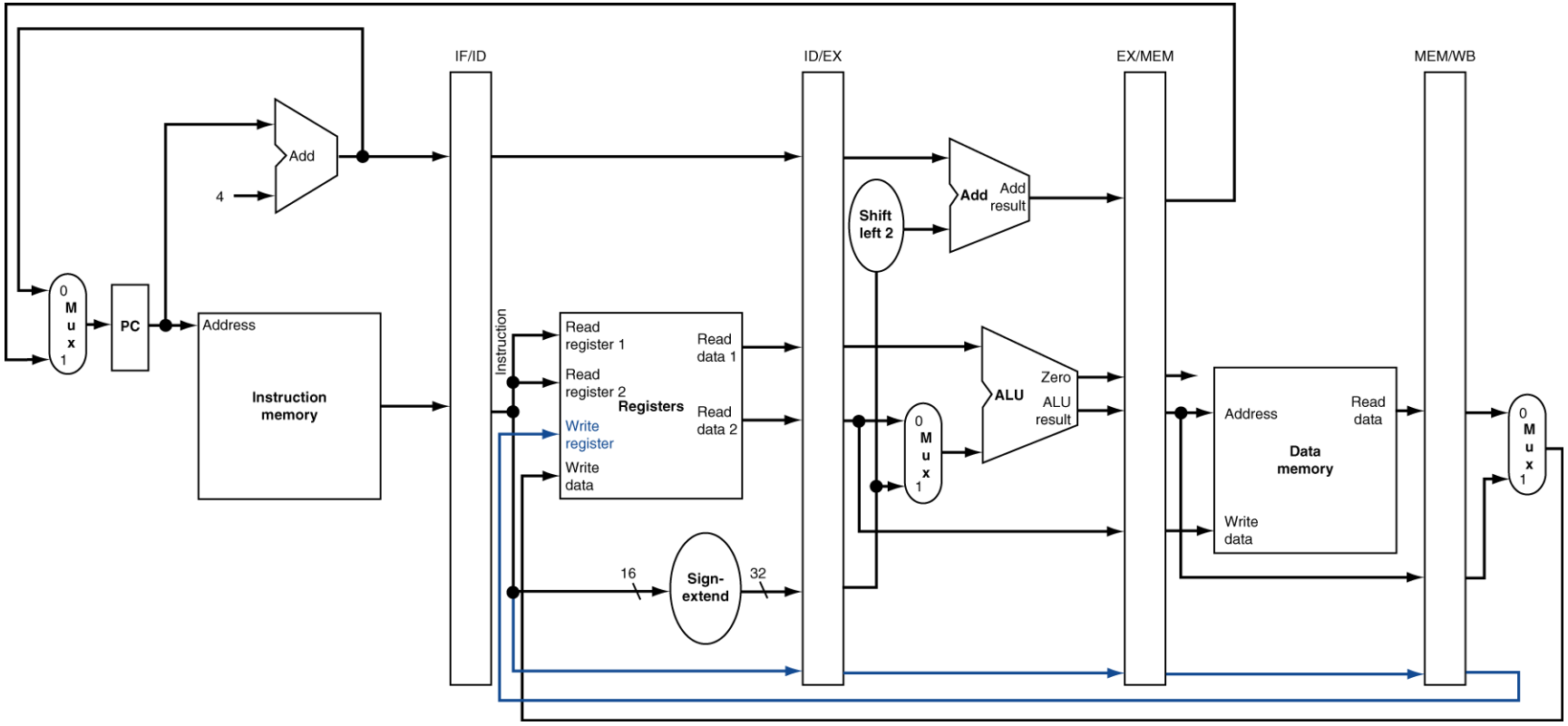
- MEM for Load


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

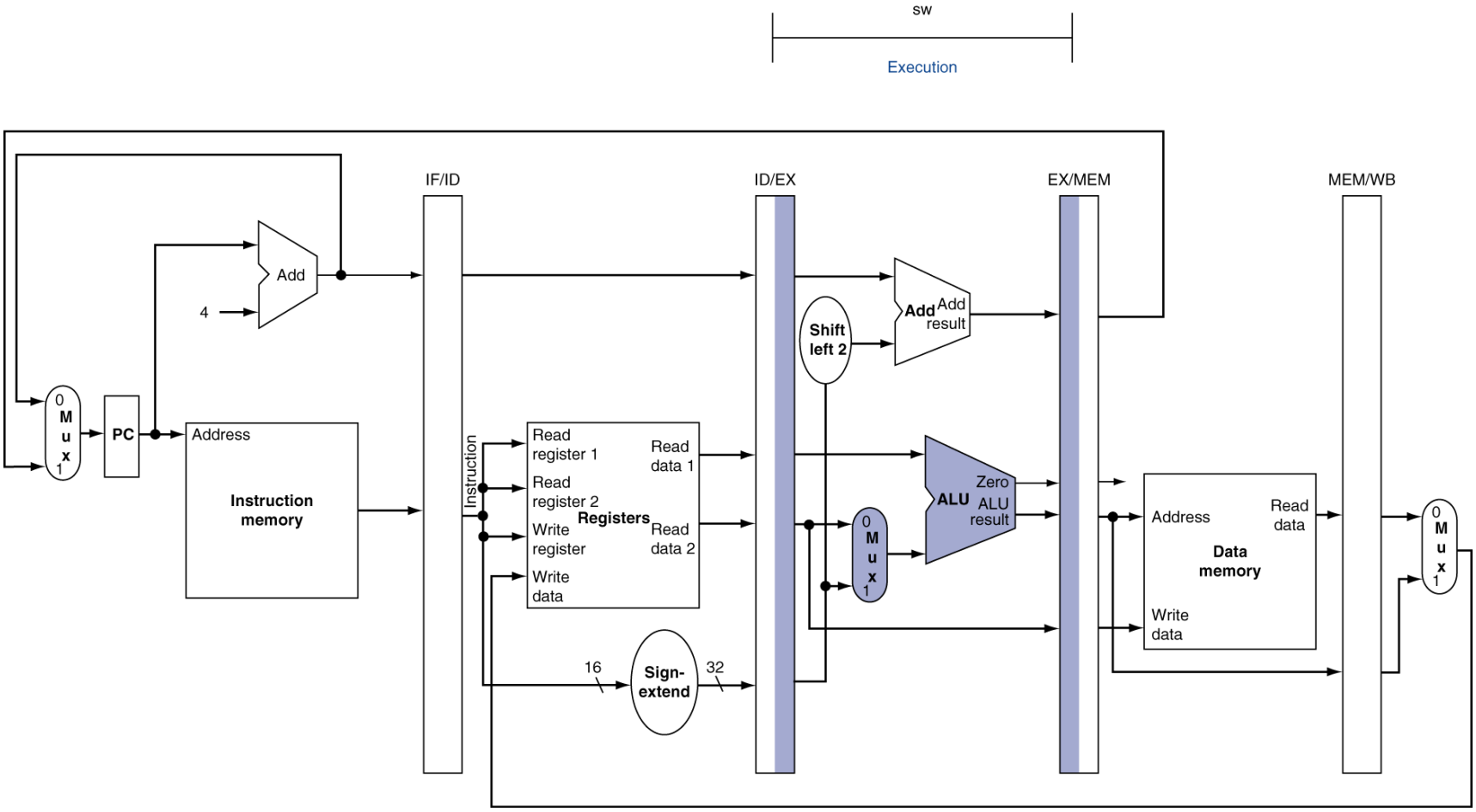

lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32
Wrong register number

- Corrected Datapath for Load

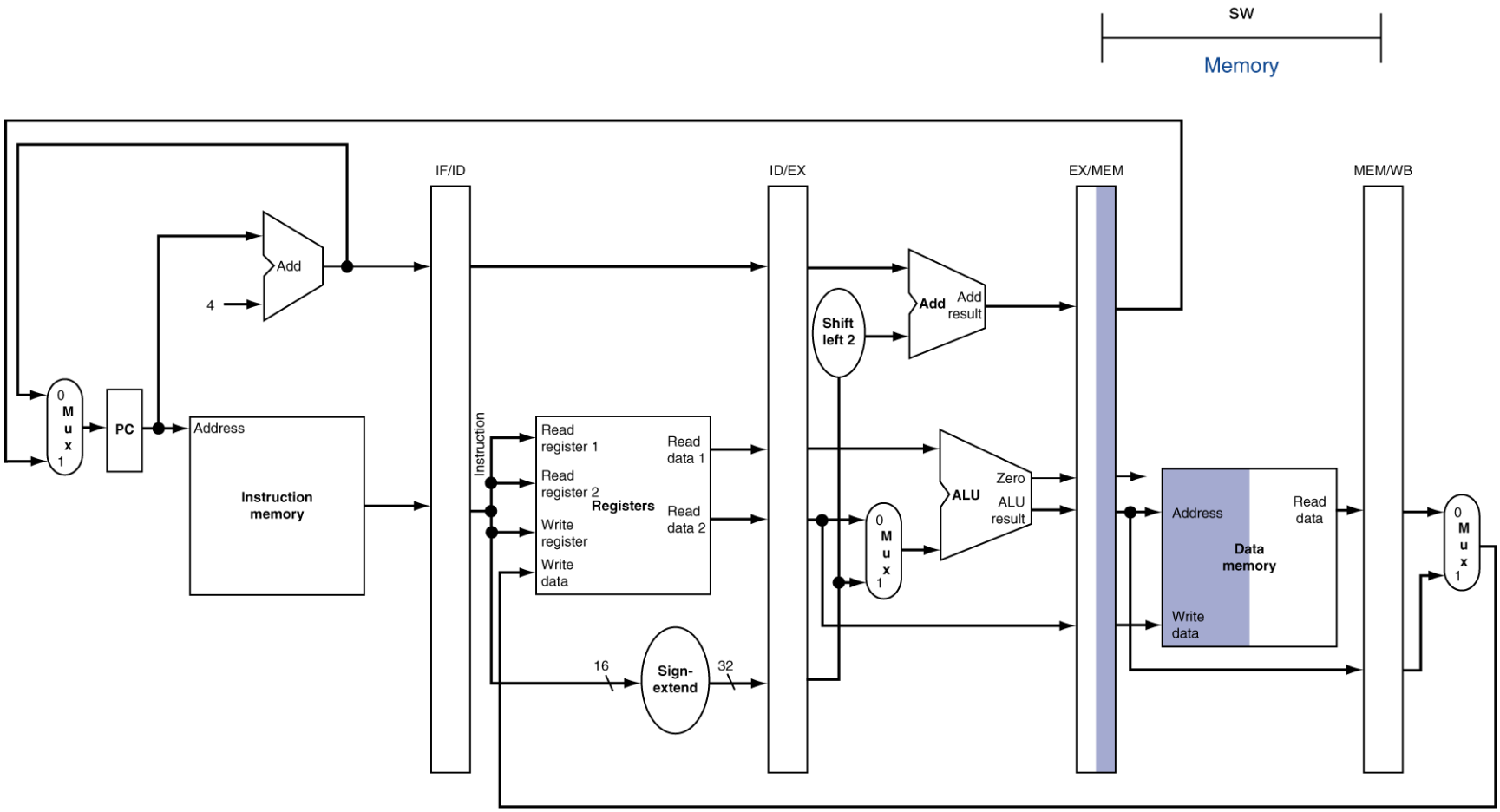


Lecture 4: The Processor - Pipelining

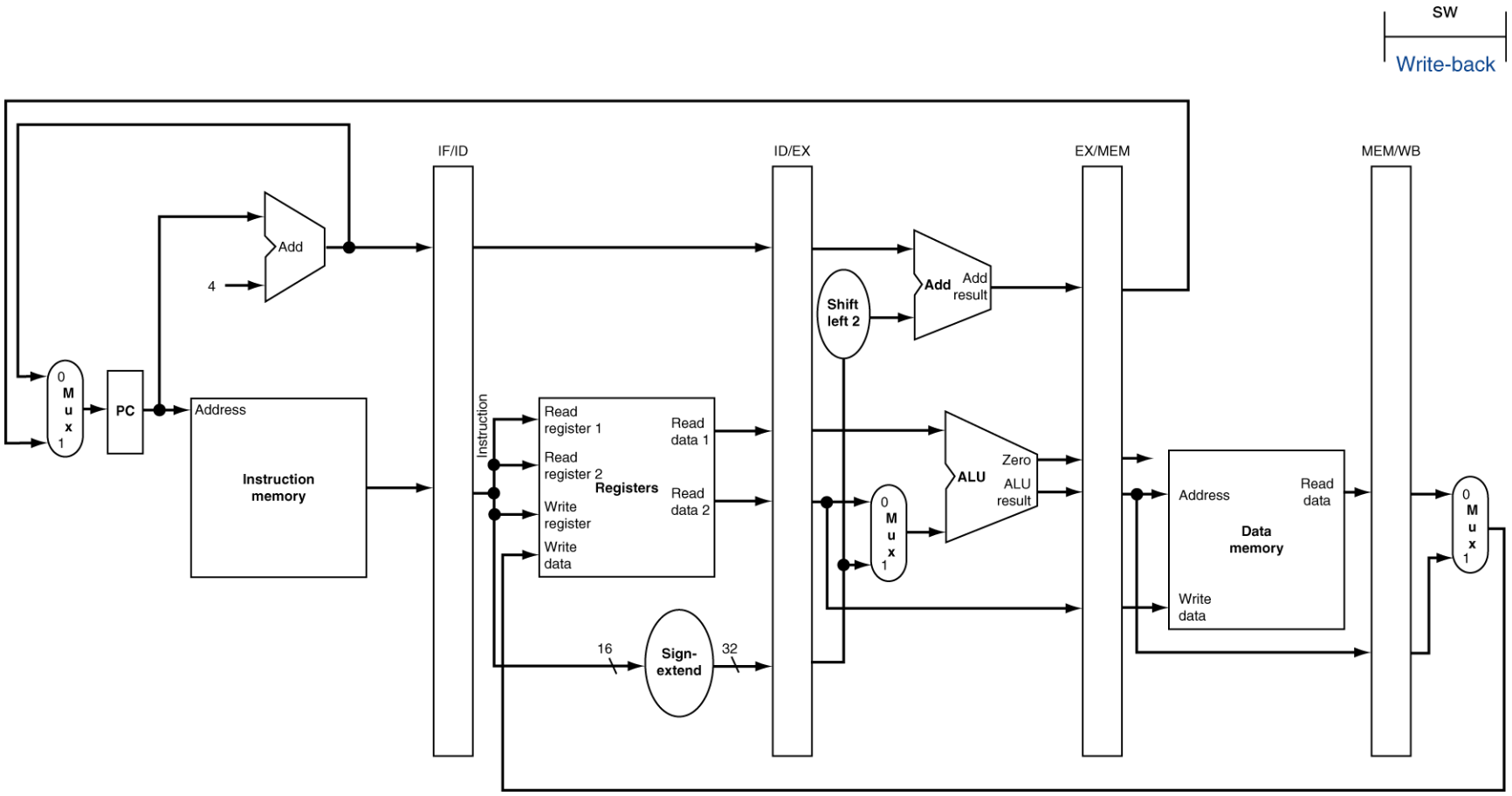
- EX for Store



- MEM for Store



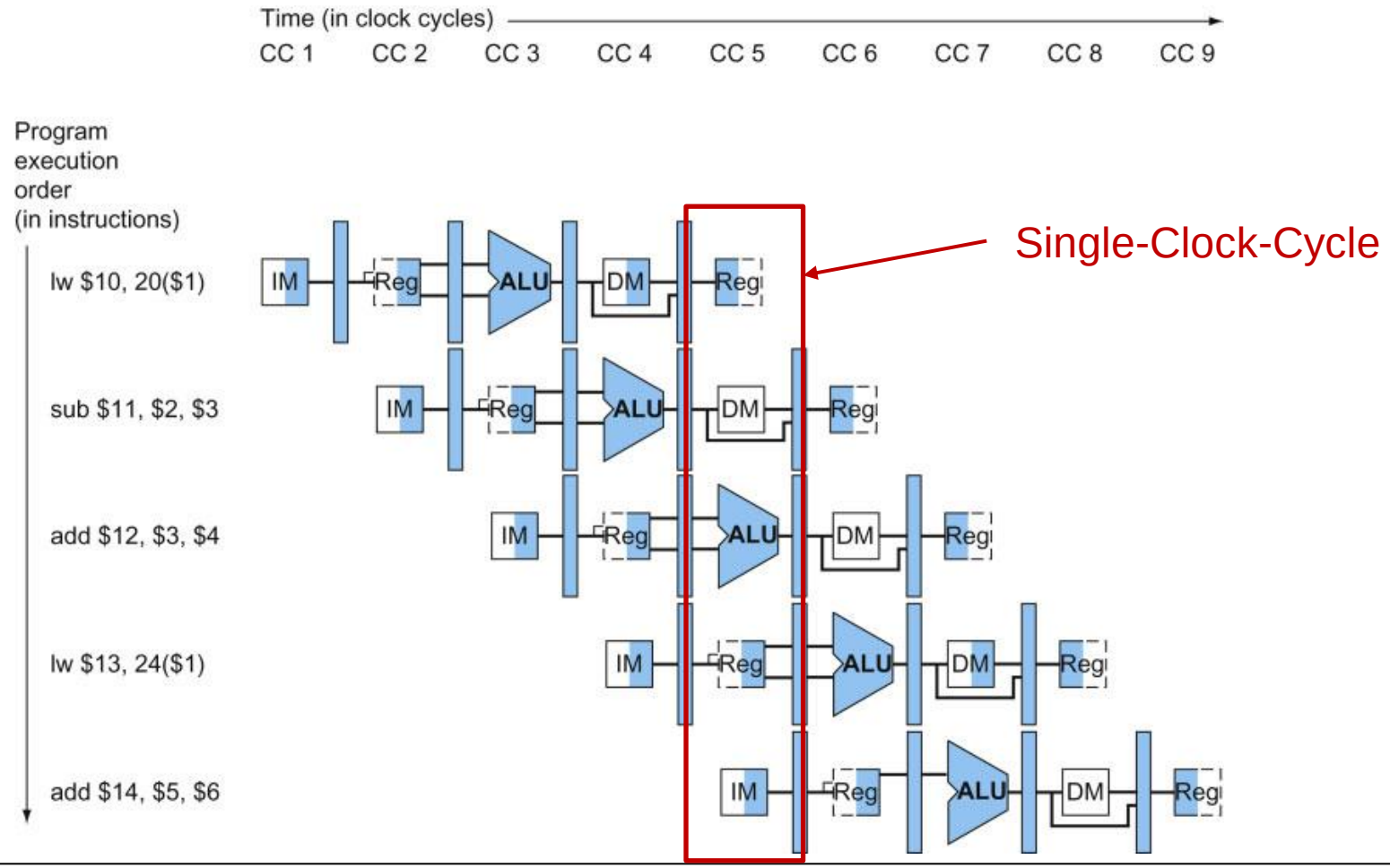
- WB for Store


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
Data memory
Read data
Write data

Two Basic Representations of Pipeline Figures

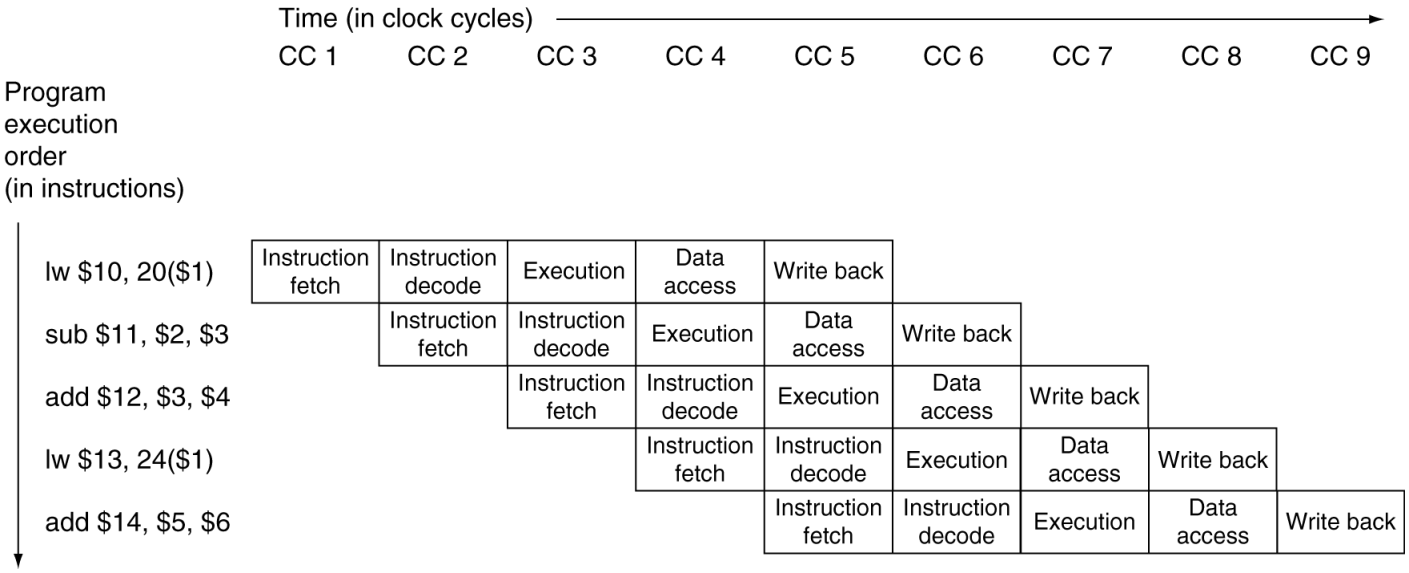
- Multi-Clock-Cycle Pipeline Diagram
  - Time axis (horizontal): CC1, CC2, CC3, …
  - Instruction axis (vertical): Instruction1, Instruction2, …
  - A pipeline stage is placed along instruction axis, occupying the proper clock cycles
  - Overview of pipelining situation
- Single-Clock-Cycle Pipeline Diagram
  - Shows the state of the entire datapath during a single clock cycle
  - Represents a vertical slice through a set of multi-clock-cycle diagrams, showing the usage of datapath by each of instructions in the pipeline at the designated clock cycle

- Multi-Cycle Pipeline Diagram
  - Form showing physical resources usage



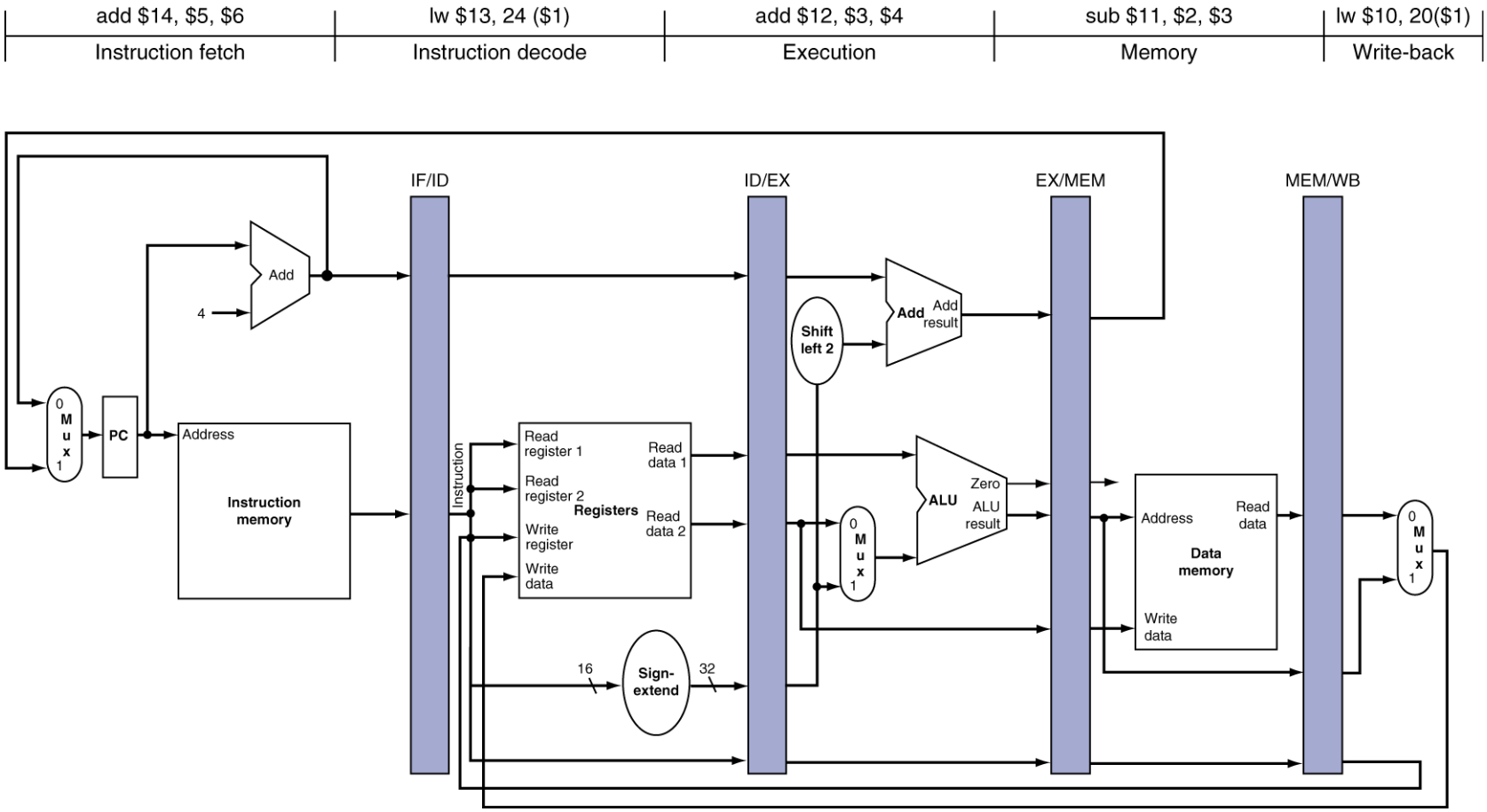
Multi-Cycle Pipeline Diagram

- Traditional form (showing name of each stage)

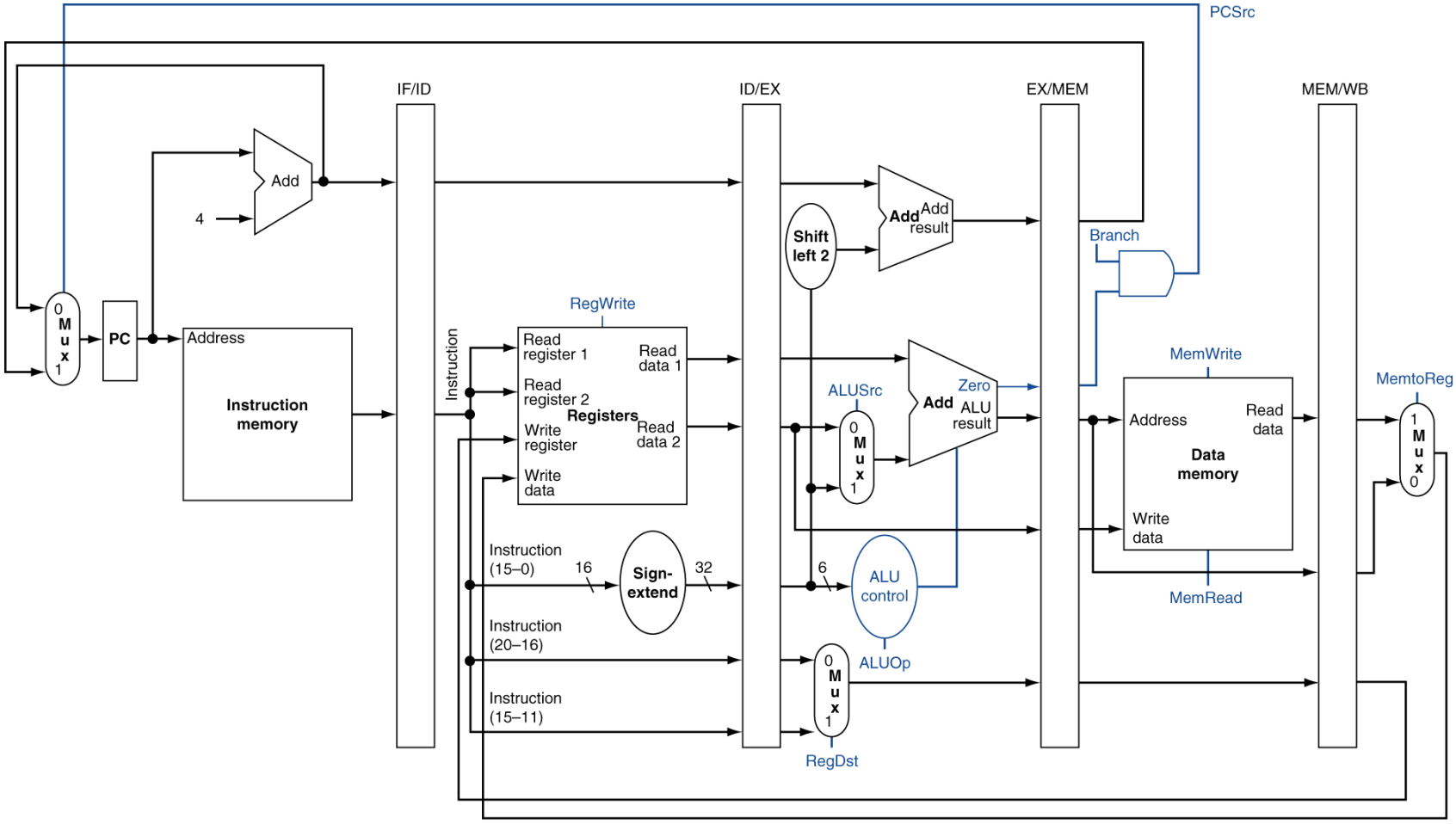


Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle



- Pipelined Control (Simplified)



- ALU Control Input buts are set based on:
  - ALUOp Control bits (2 bits)
  - R-Type instruction function codes (6 bits)

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

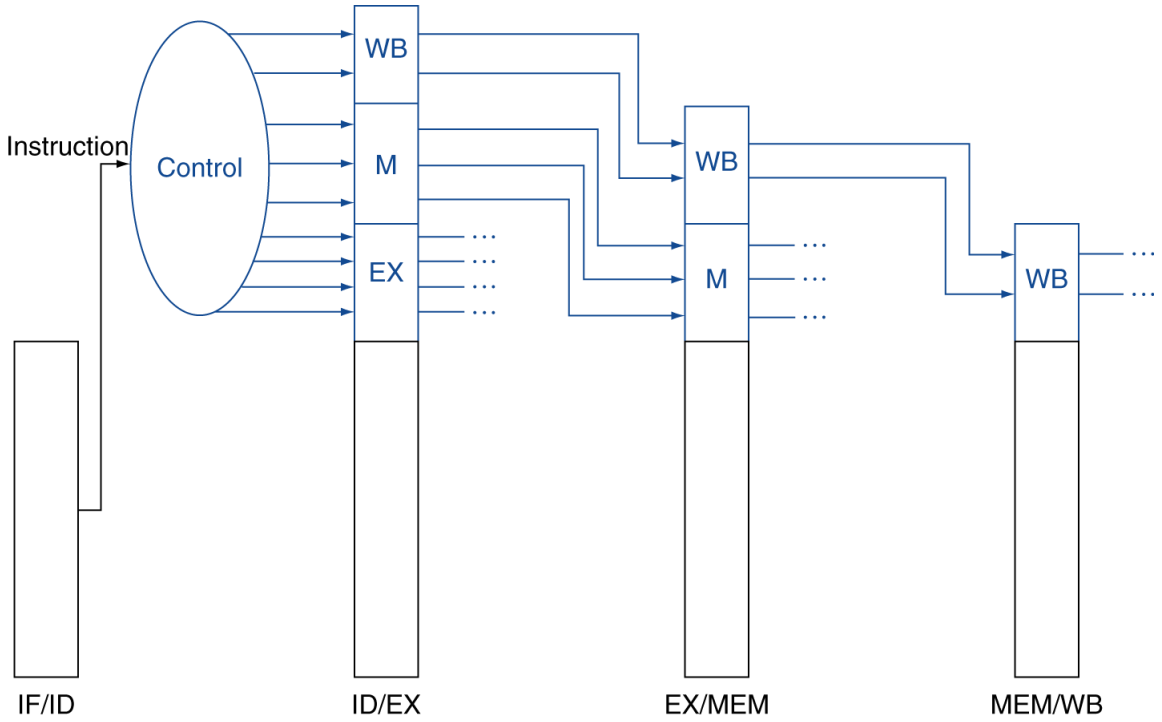
- The Function of Seven Control Signal (ALUOp control lines 2bits are defined separately)

Signal name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

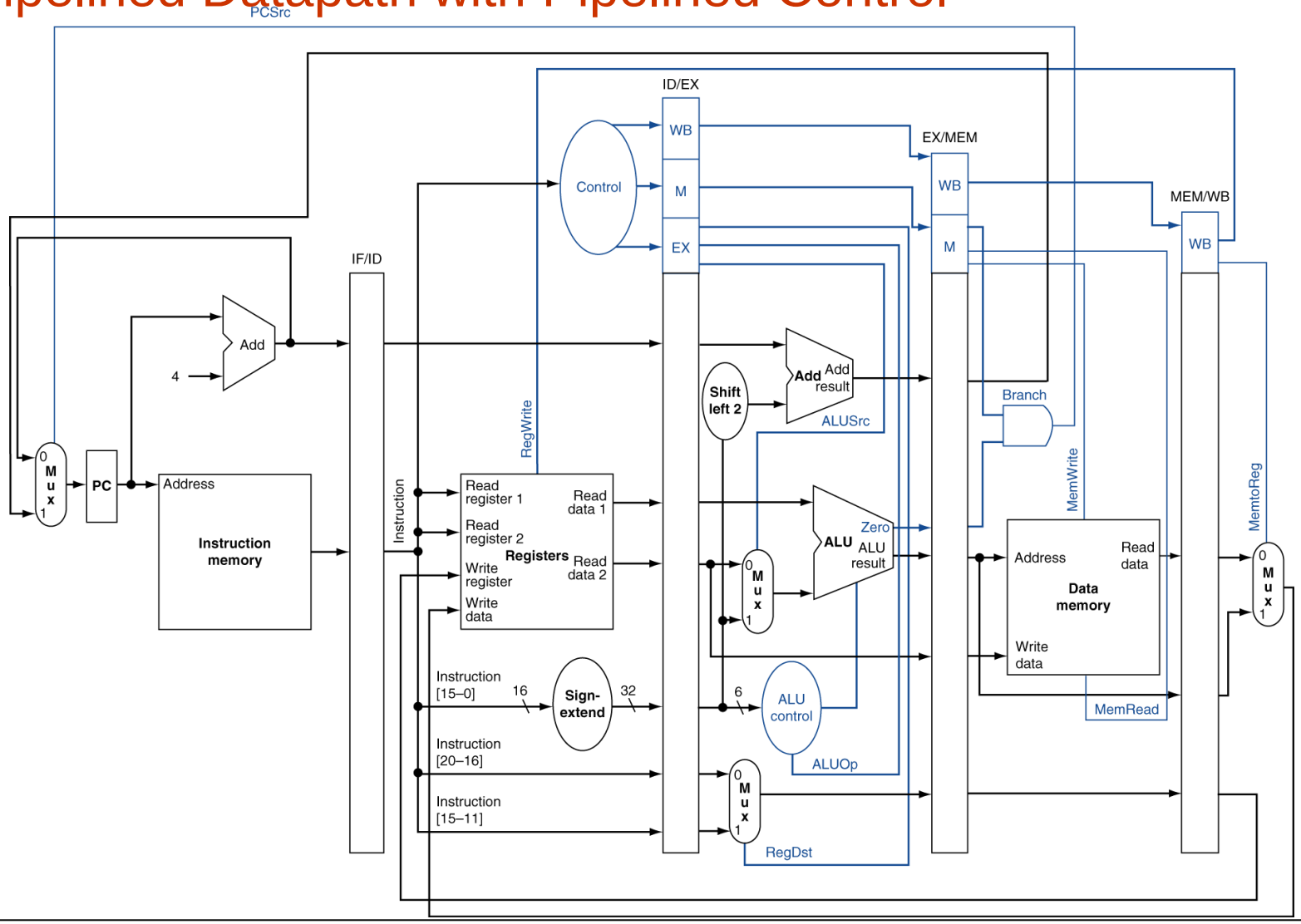
- Nine Control lines grouped by three pipeline stages

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
    - Four used in EX stage, three used in Mem stage, two used in WB stage



- Pipelined Datapath with Pipelined Control



Data Hazards in ALU Instructions

- Consider this sequence:

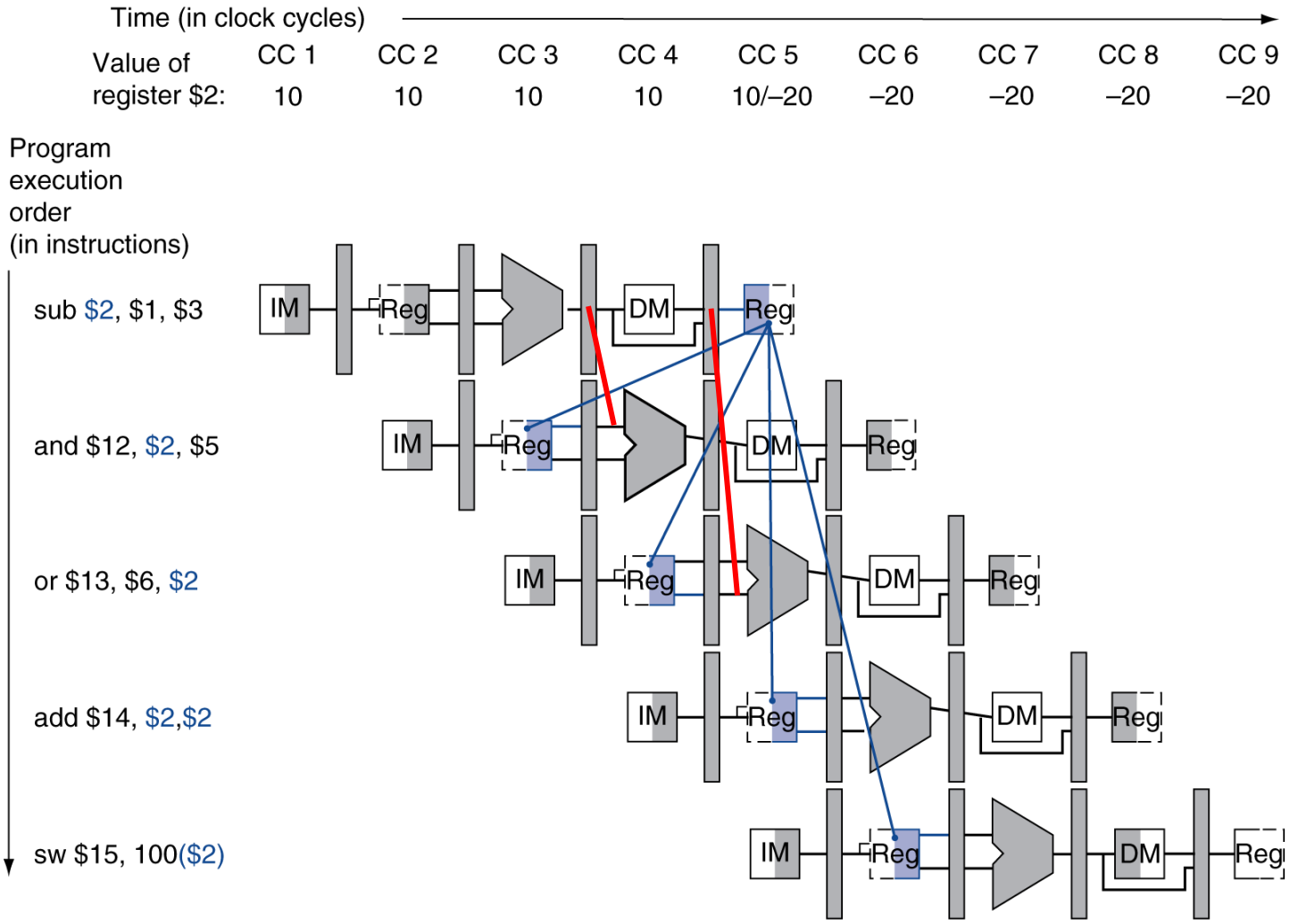
```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- We can resolve hazards with forwarding
  - How do we detect when to forward?
  - $2 had the value 10 before the sub instruction and -20 afterwards. The programmer intends to use -20 in the following instructions

Lecture 4A Key Concepts Review

- Polling: Forwarding

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

  } Fwd from EX/MEM pipeline reg

  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

  } Fwd from MEM/WB pipeline reg

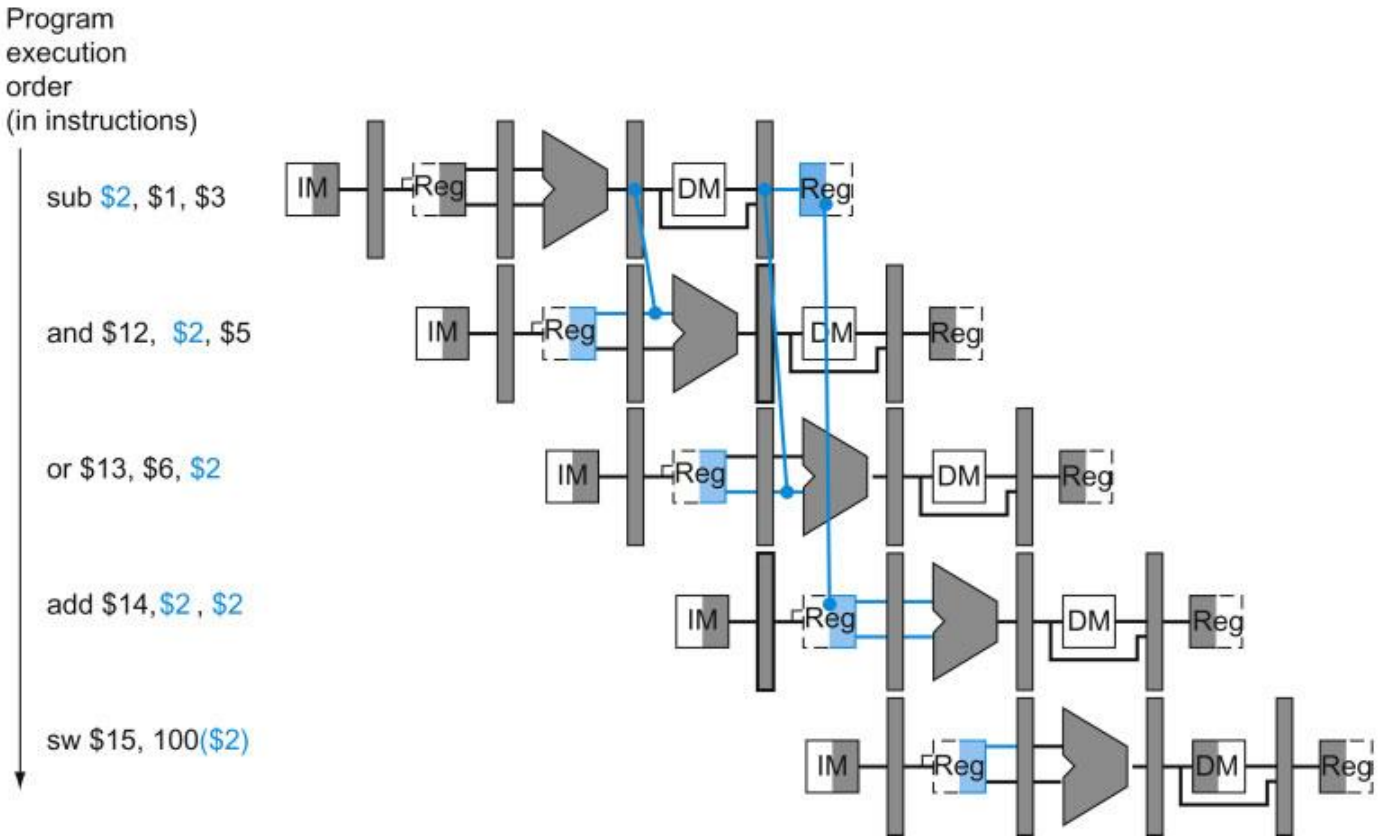
Detecting the Need to Forward

- 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 ≠ 0, MEM/WB.RegisterRd ≠ 0

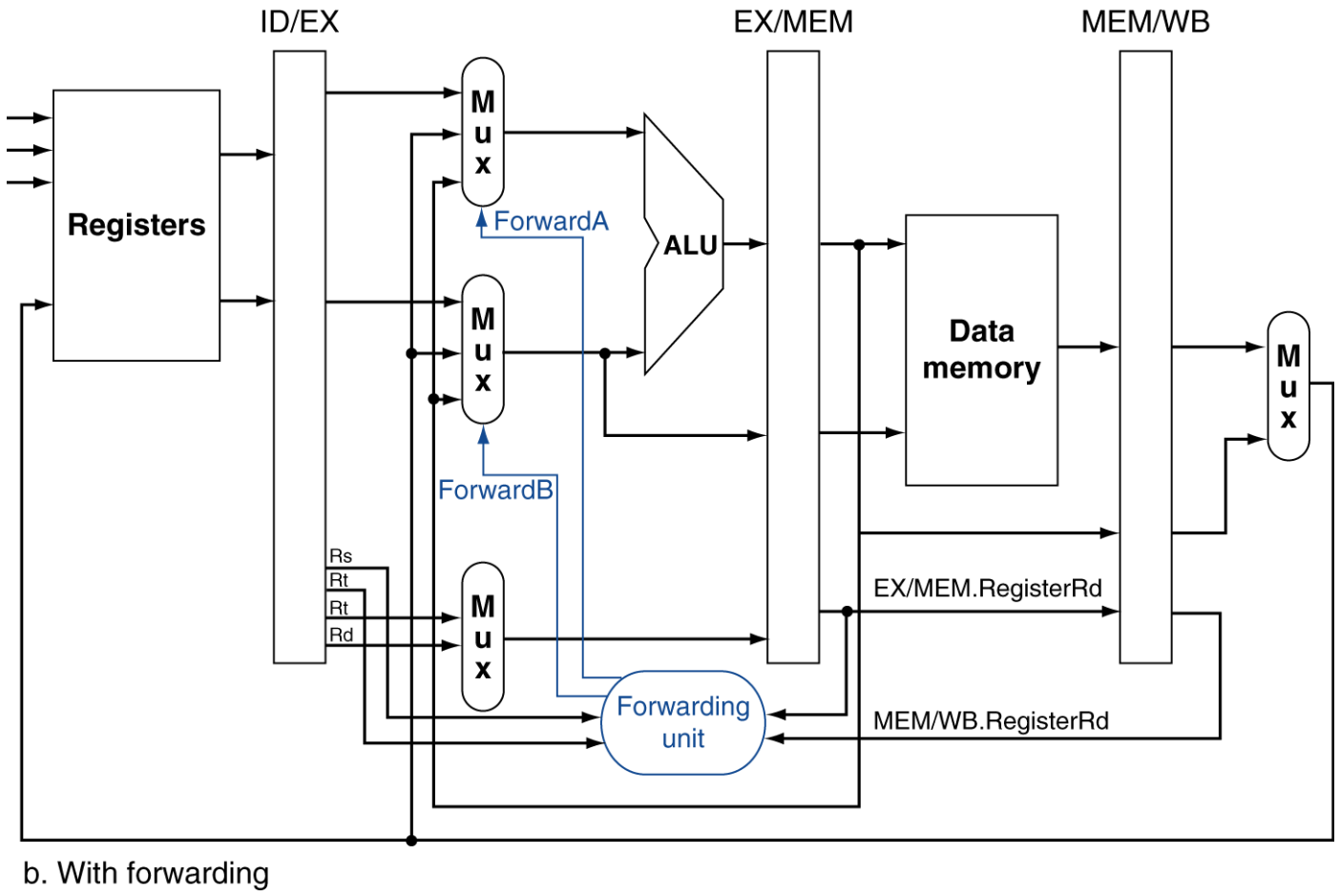
Dependencies & Forwarding

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X



Forwarding Paths



b. With forwarding

The Control Values for the Forwarding Muxes

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Forwarding Conditions

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

Double Data Hazard

- Consider the sequence:

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

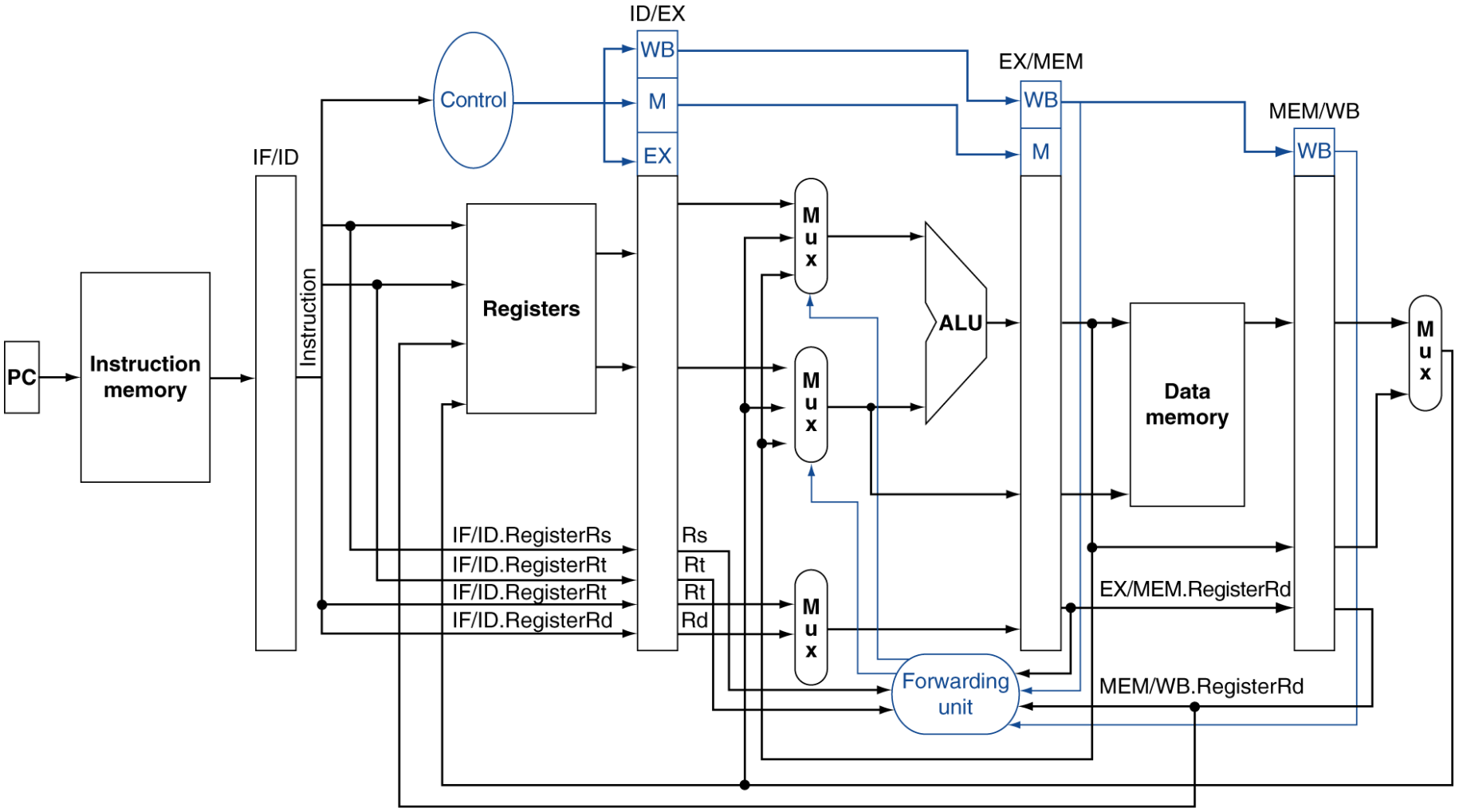
- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

The result is forwarded from MEM stage (most recent result), the control for MEM hazard needs to be revised

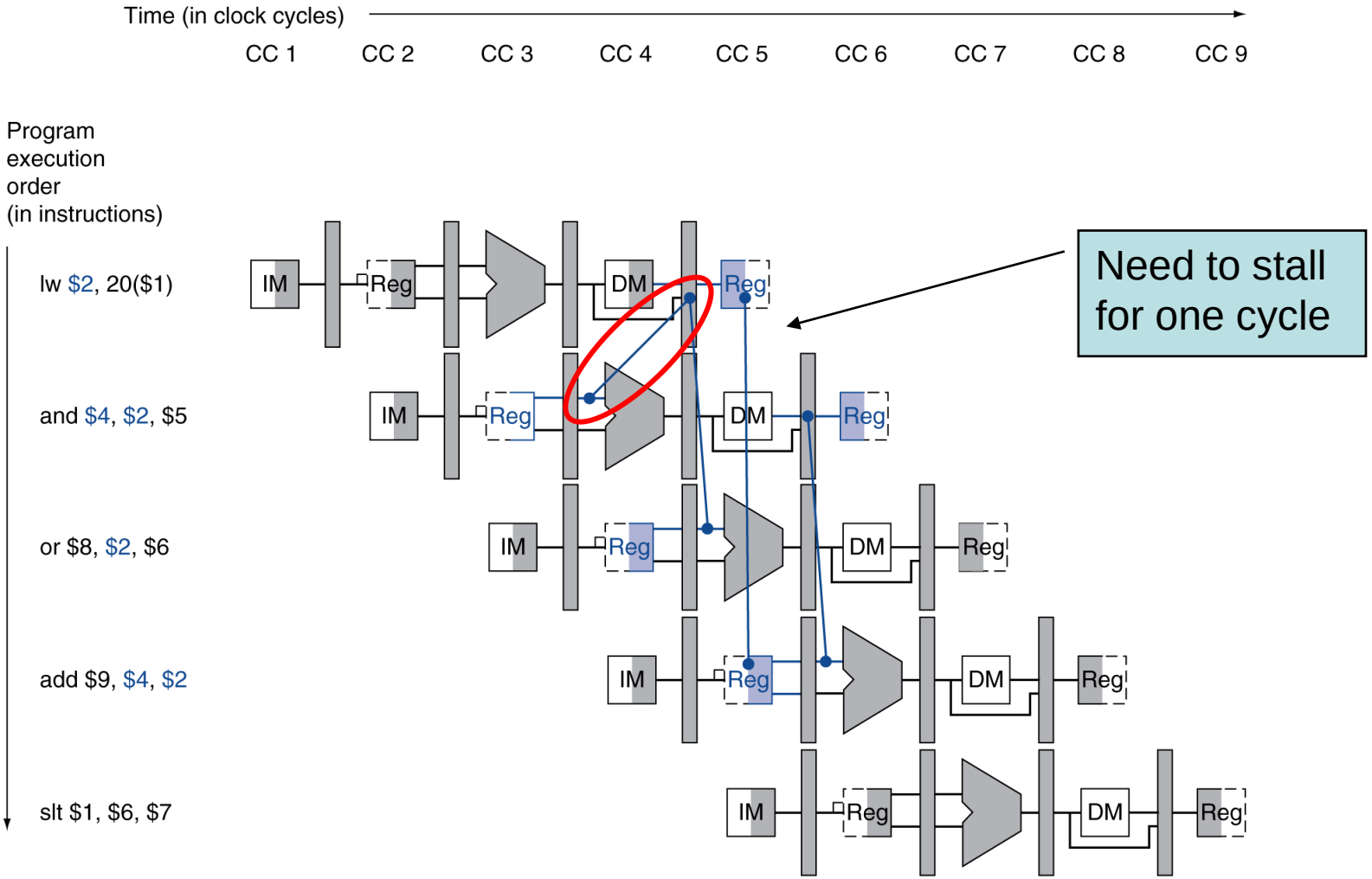
Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Datapath with Forwarding



Load-Use Data Hazard



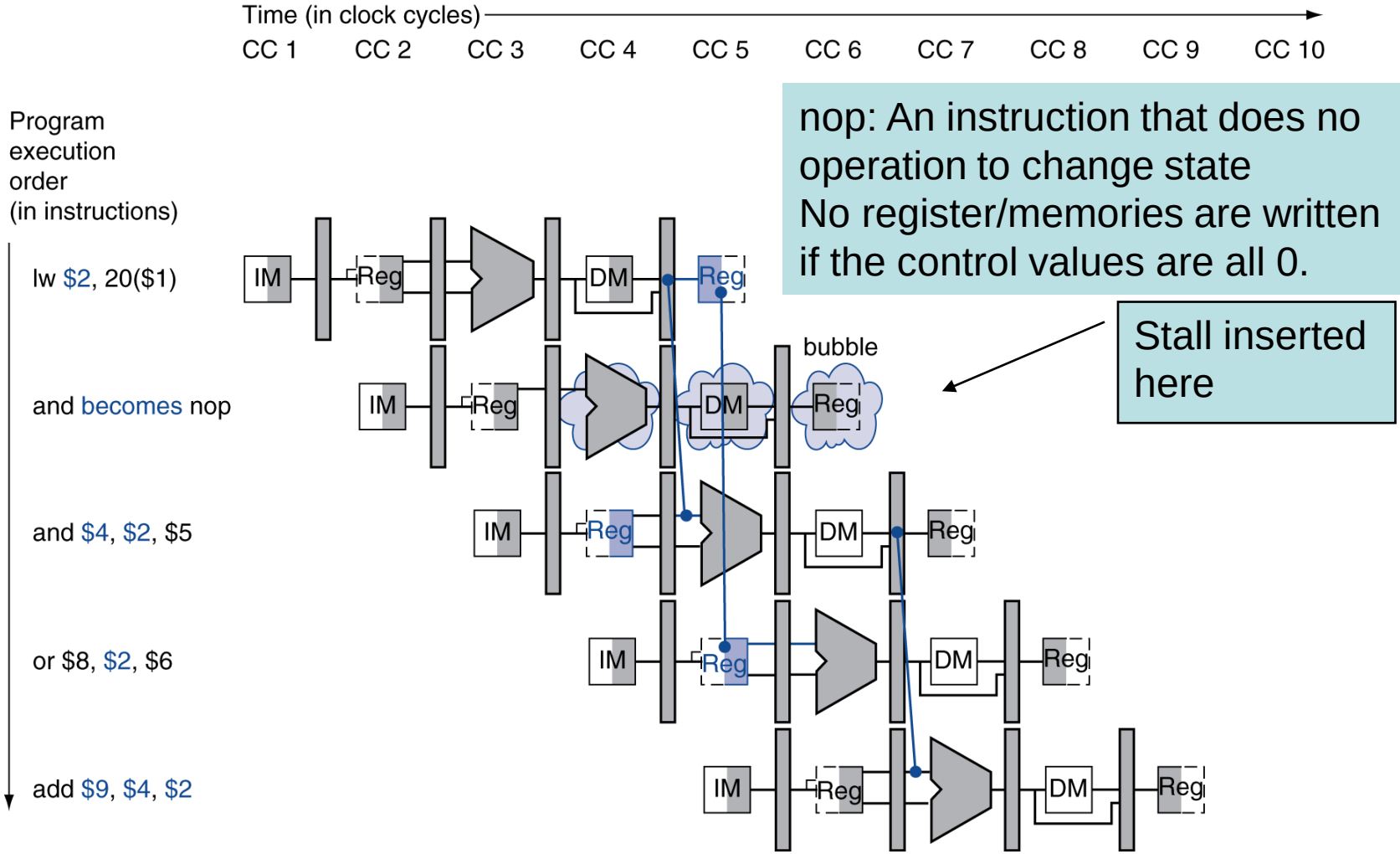
Load-Use Hazard Detection

- 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

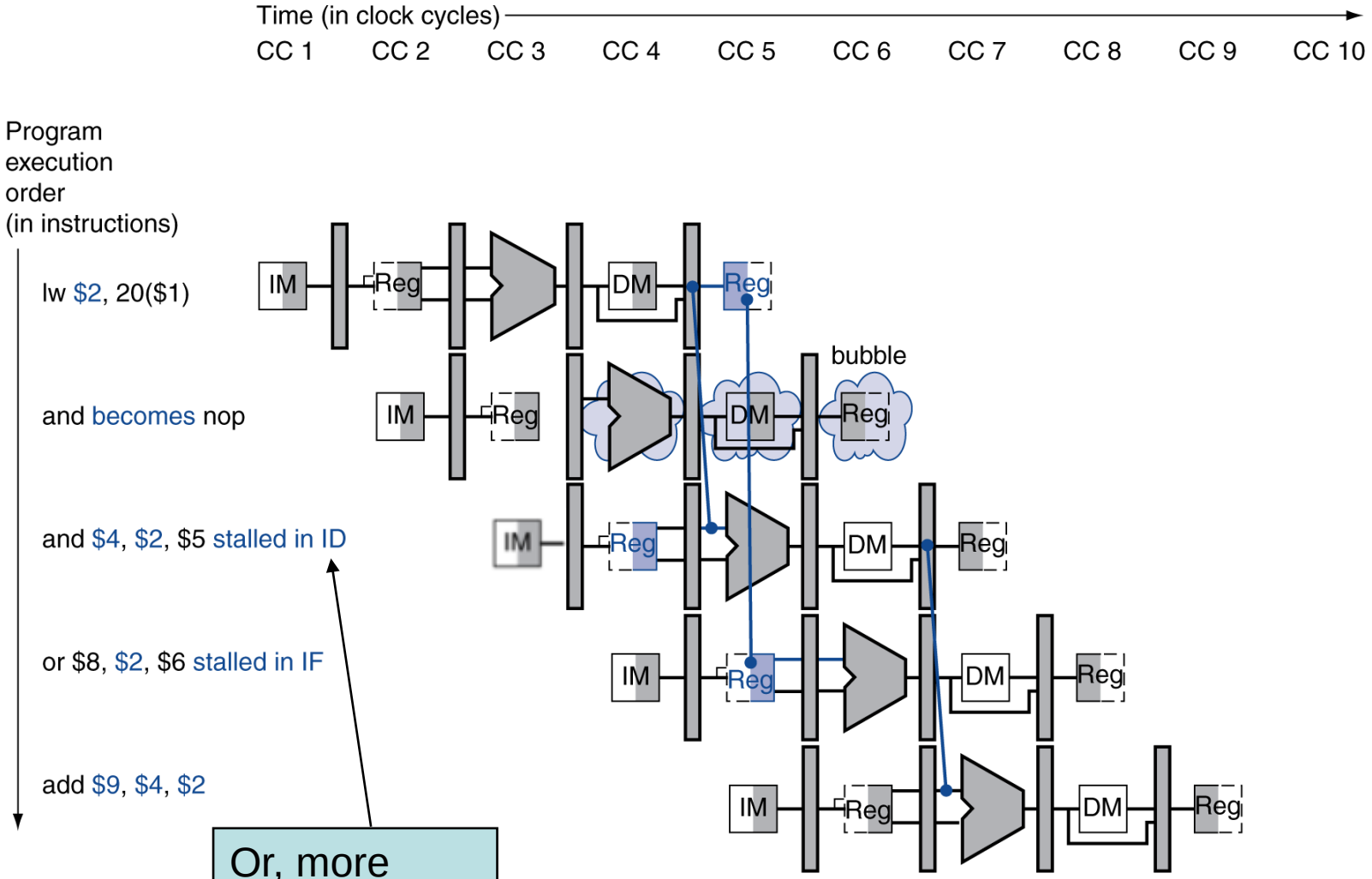
How to Stall the Pipeline

- 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 lw
    - Can subsequently forward to EX stage

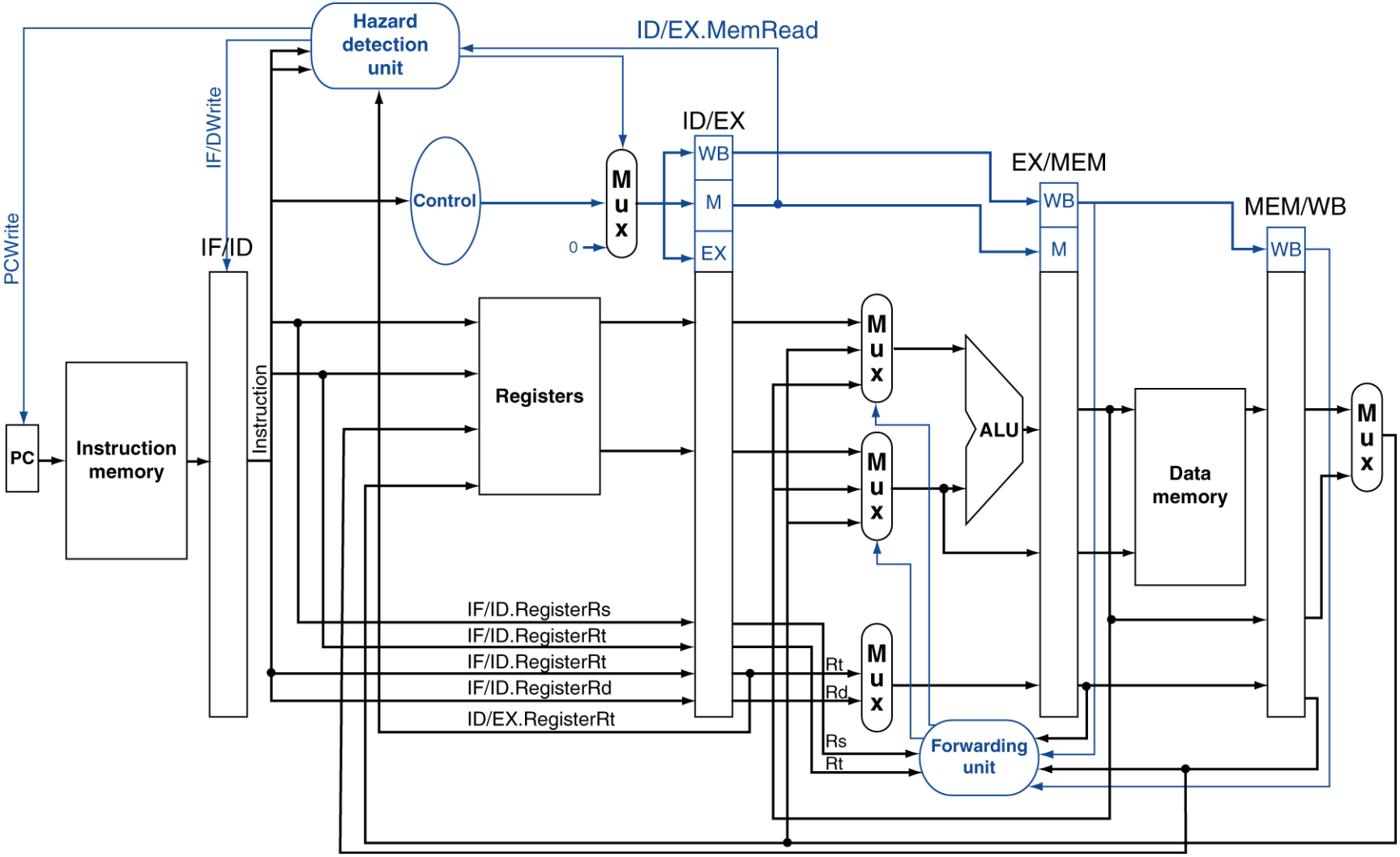
Stall/Bubble in the Pipelined



Stall/Bubble in the Pipeline



Datapath with Hazard Detection



Stalls and Performance

The BIG Picture

- 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

- Next lecture
  - 4.8 Control Hazards
  - 4.9 Exceptions