MIPS Pipeline

Data and Control Path

MIPS Pipeline

- To Increase the Speed of execution of program
  - **Faster circuit technology** to implement the processor and the main memory
  - Arrange the hardware so that **more than one operation** can be performed at the same time.
    - the number of operations performed per second is increased
    - the time needed to perform any one operation is not changed

- Pipelining is a way of organizing concurrent activity in a computer

MIPS Pipeline

- Five stages, one step per stage

1. IF: Instruction fetch from memory
2. ID: Instruction decode & register read
3. EX: Execute operation or calculate address
4. MEM: Access memory operand
5. WB: Write result back to register

Performance Issues

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- All instructions will follow all 5 stages in pipeline

Pipeline Performance

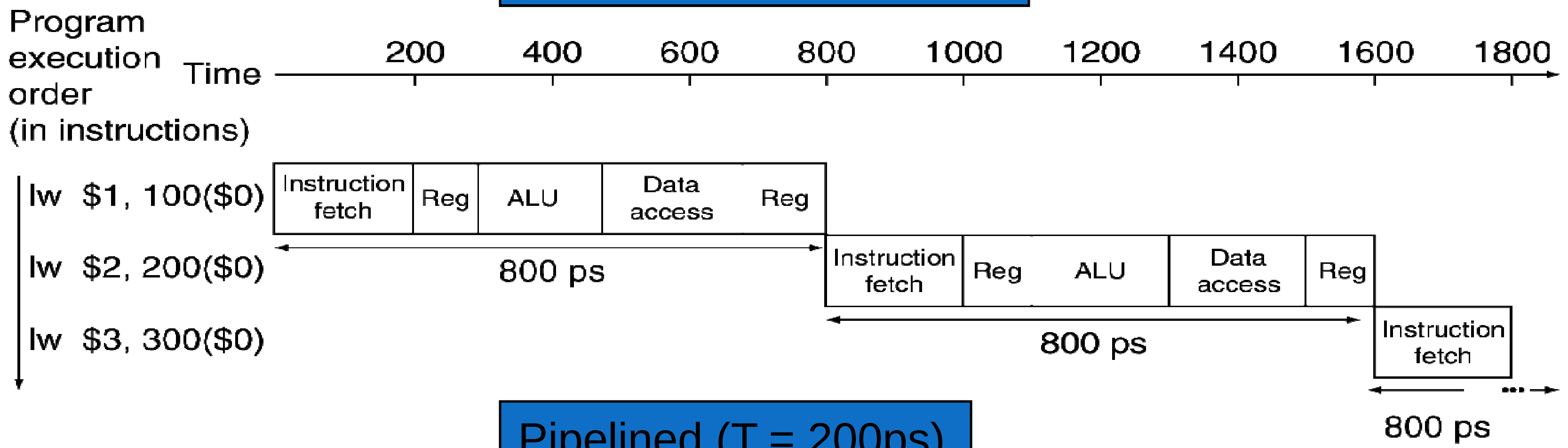
- Assume time for stages are
  - 100ps for register read or write
  - 200ps for other stages
- Single-cycle datapath need worst-case clock cycle of 800 ps,
- Pipelined datapath need worst-case clock cycle of 200 ps

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

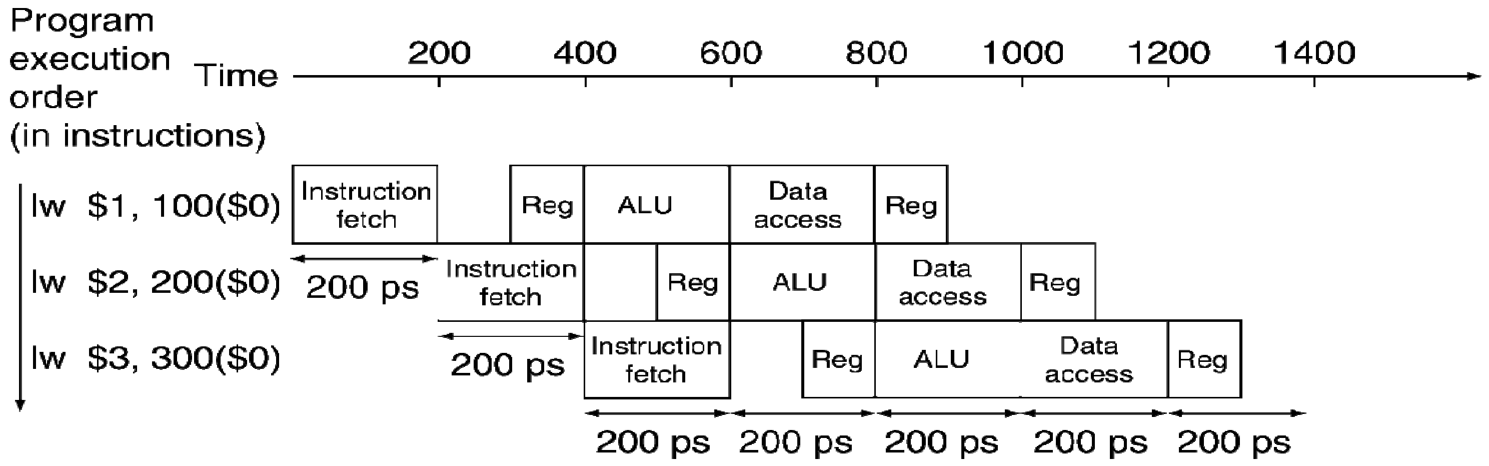
Pipeline Performance

The time between the first and fourth instructions is 3 × 200 ps or 600 ps for Pipeline.

Single-cycle (T_c= 800ps)



Pipelined (T_c= 200ps)



Pipeline Speedup

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instruction}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

- Speedup from pipelining is approximately equal to the number of pipe stages;
- a five-stage pipeline is nearly five times faster.
- 800/5 =160ps clock cycle

- If all stages are balanced (all stage take the same time)
- Pipelining offers a fourfold performance improvement (800/200=4)
- If balanced, speedup is less: in our example 200ps

Pipelining and ISA Design

- MIPS ISA designed for pipelining
  - All instructions are 32-bits
    - fetch and decode each takes only one cycle
  - Few and regular instruction formats
    - Can decode and read registers in one step
      (read register are in same place (rs))
  - Only Load/store use memory operand
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Memory Operand Alignment
    - 4bytes – address multiples of four
    - Memory access takes only one cycle

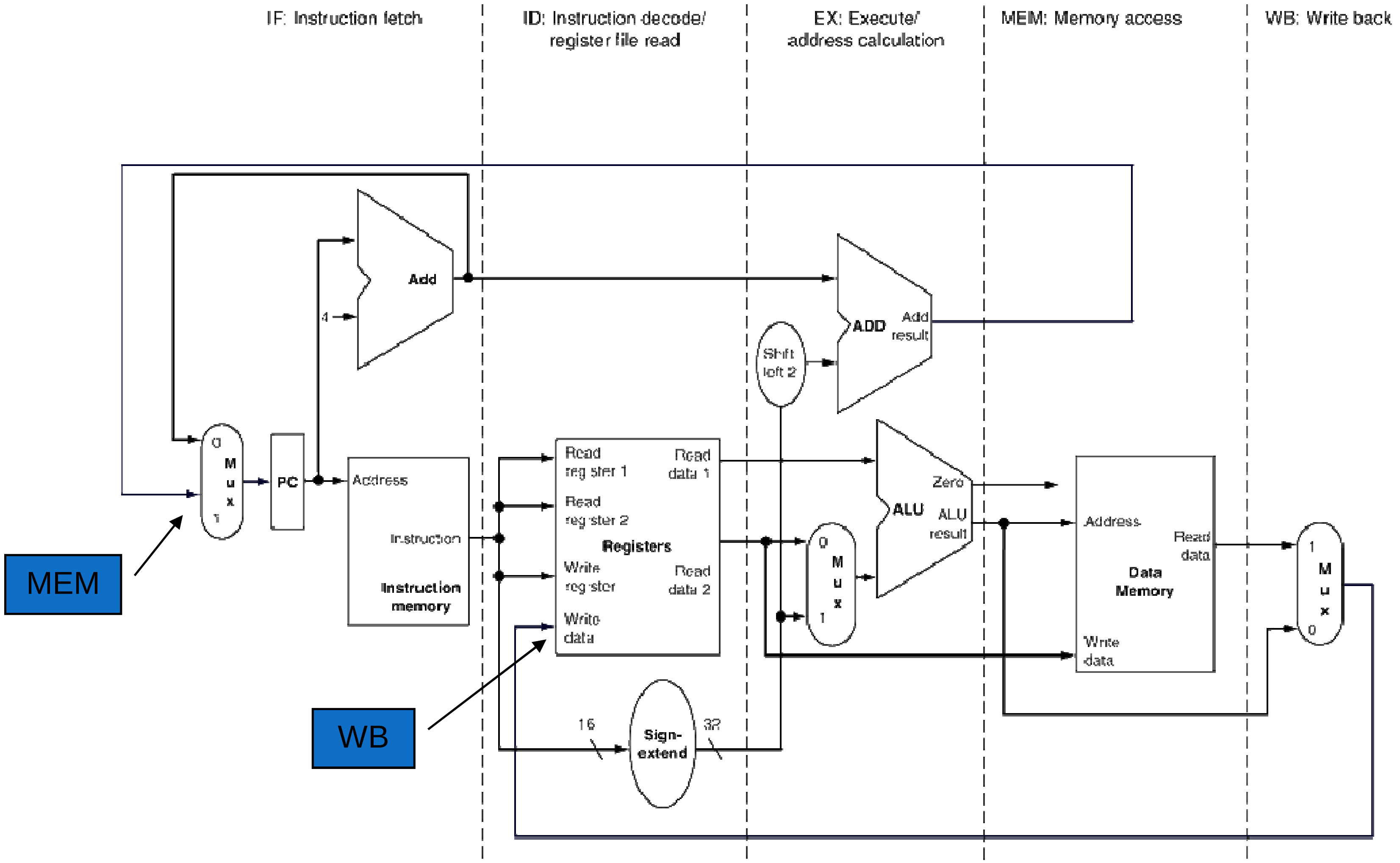
Pipelining Issues

- Hazards
  - A new instruction enter the pipeline in every cycle
  - Situations in pipelining when the next instruction cannot execute in the following clock cycle - called *hazards*
- ***Types***
  - **Structure hazards**
    - A required resource is busy
  - **Data hazard**
    - Need to wait for previous instruction to complete its data read/write
  - **Control hazard**
    - Deciding on control action depends on previous instruction

MIPS Pipelined Datapath

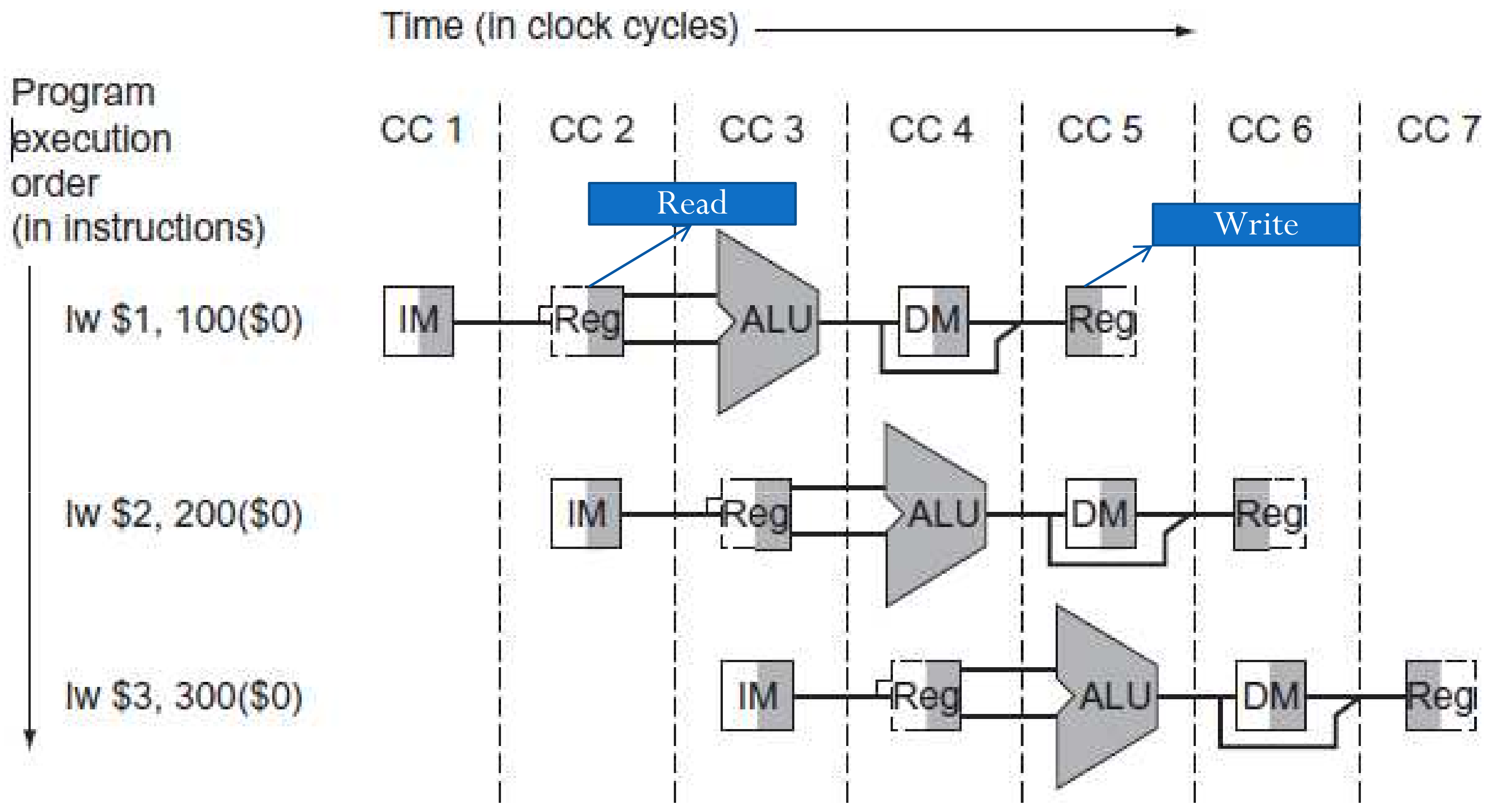
5 stages

1. IF:
2. ID
3. EX:
4. MEM:
5. WB:



Right-to-left flow leads to hazards (MEM – control hazard, WB – data hazard)

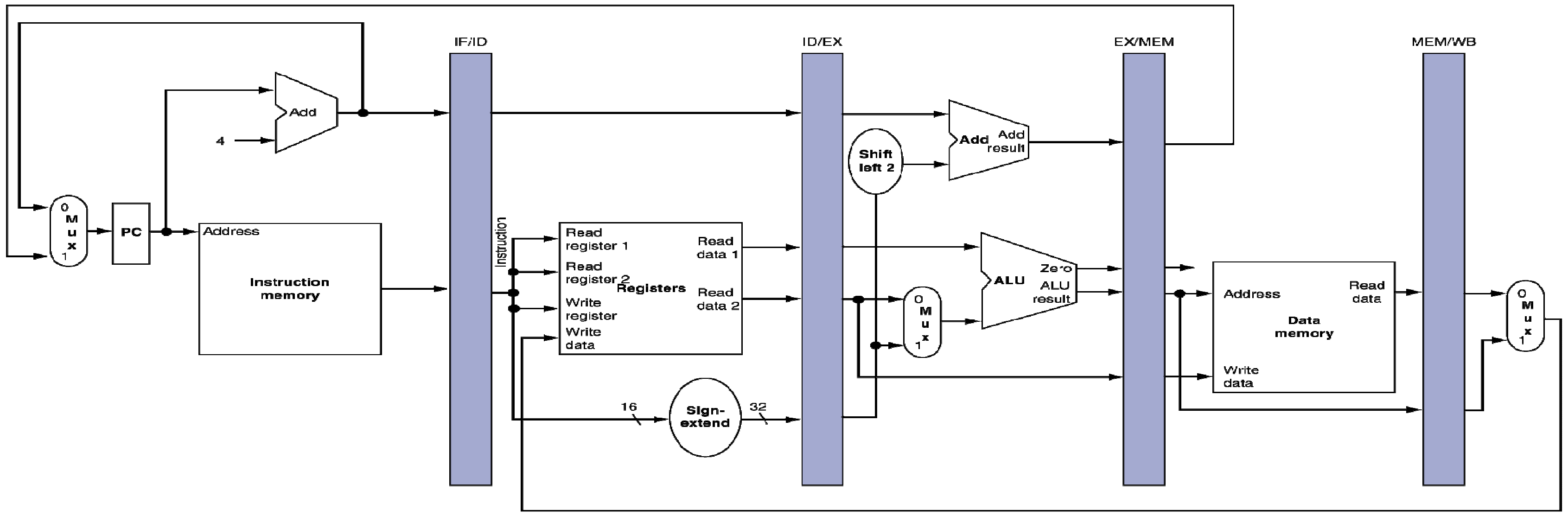
MIPS Pipelined Datapath

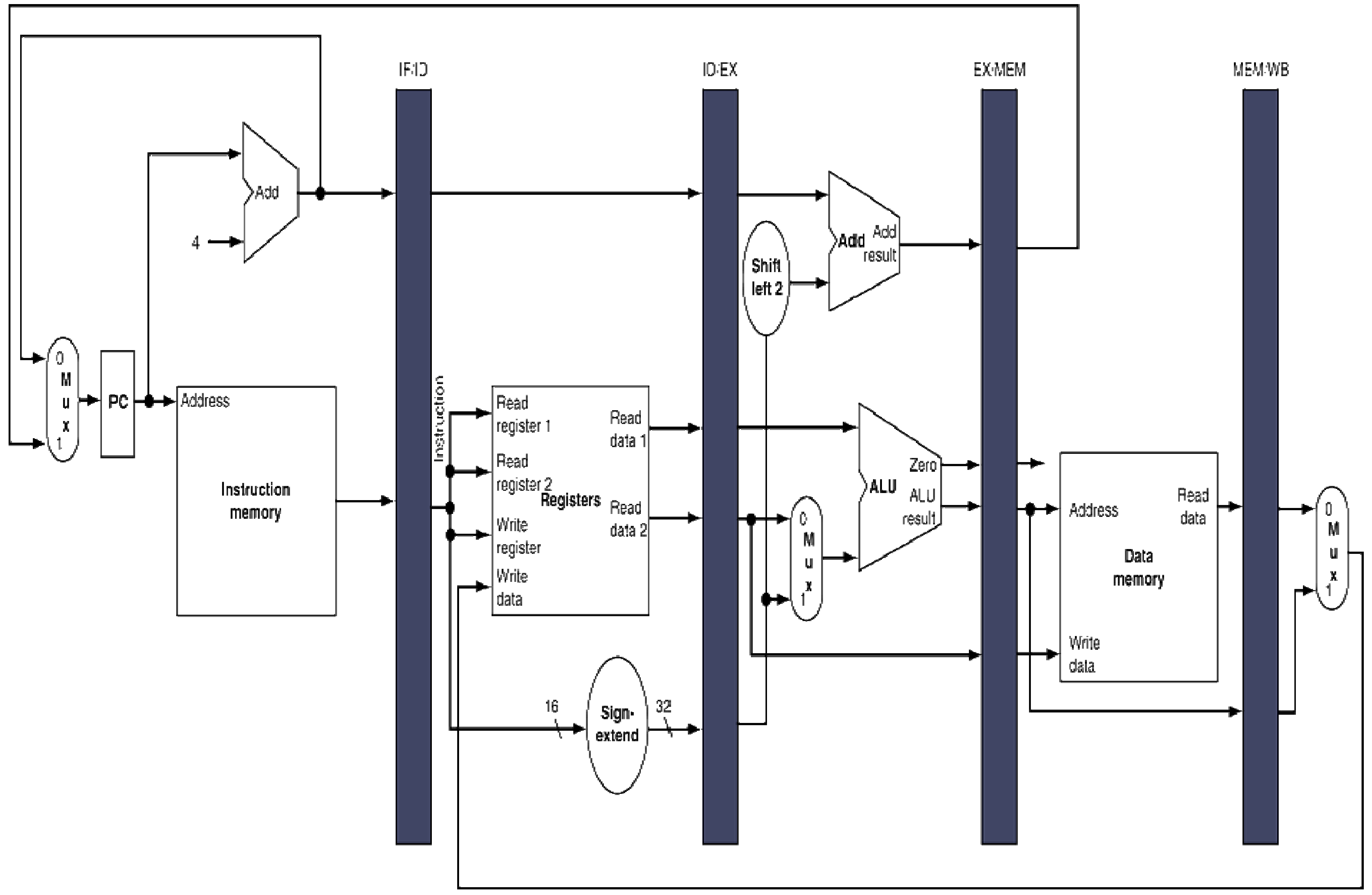


- Three instructions need three datapaths
- Add registers to hold data so that portions of a single datapath can be shared

Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle
- All instructions advance during each clock cycle from one pipeline register to the next.
- No need of pipeline register at the end of the write-back stage as it updates register file, memory, or the PC
- PC is part of the visible architectural state; its contents must be saved when an exception occurs, while the contents of the pipeline registers can be discarded.



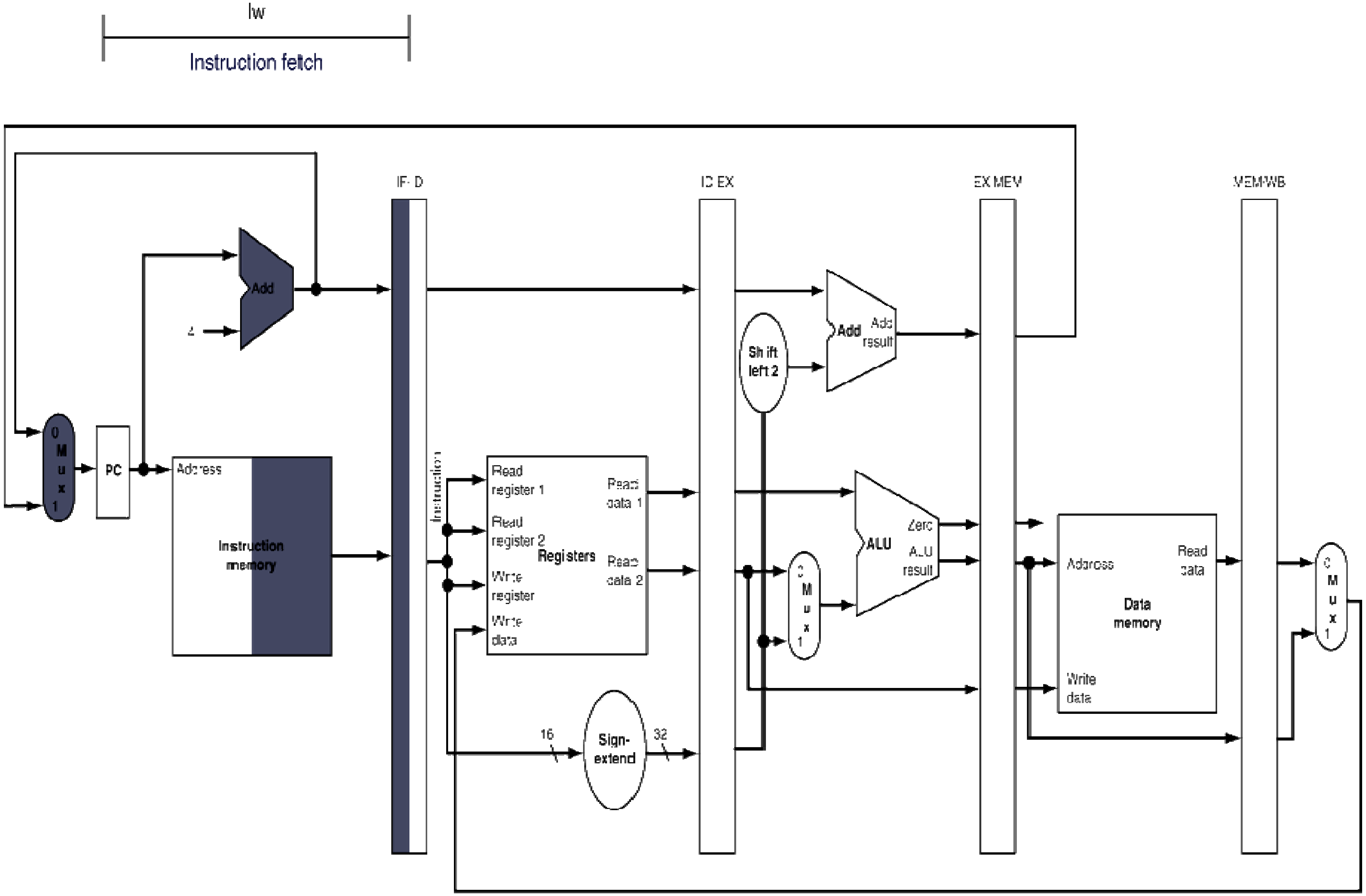


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

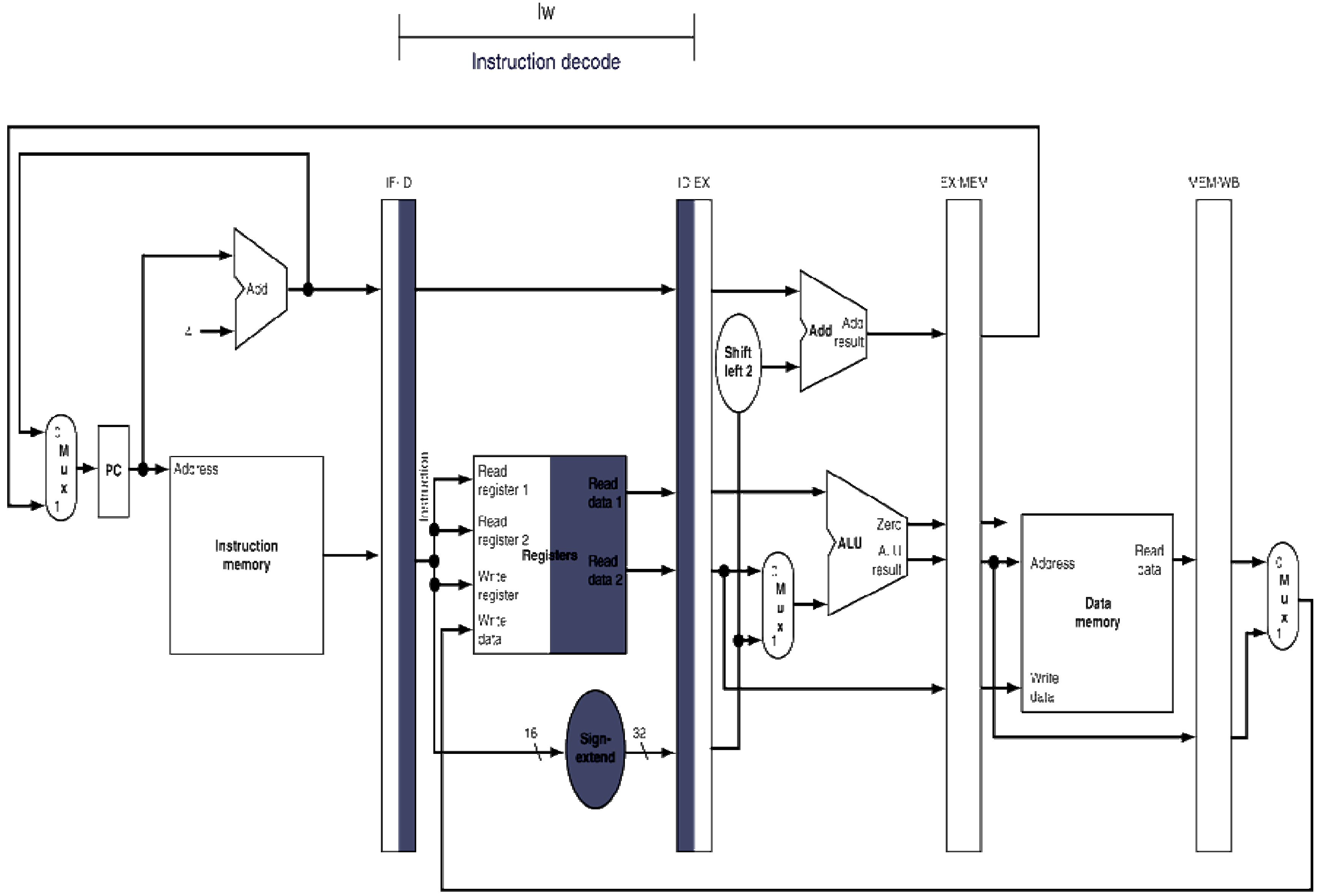
Pipeline Operation

- 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

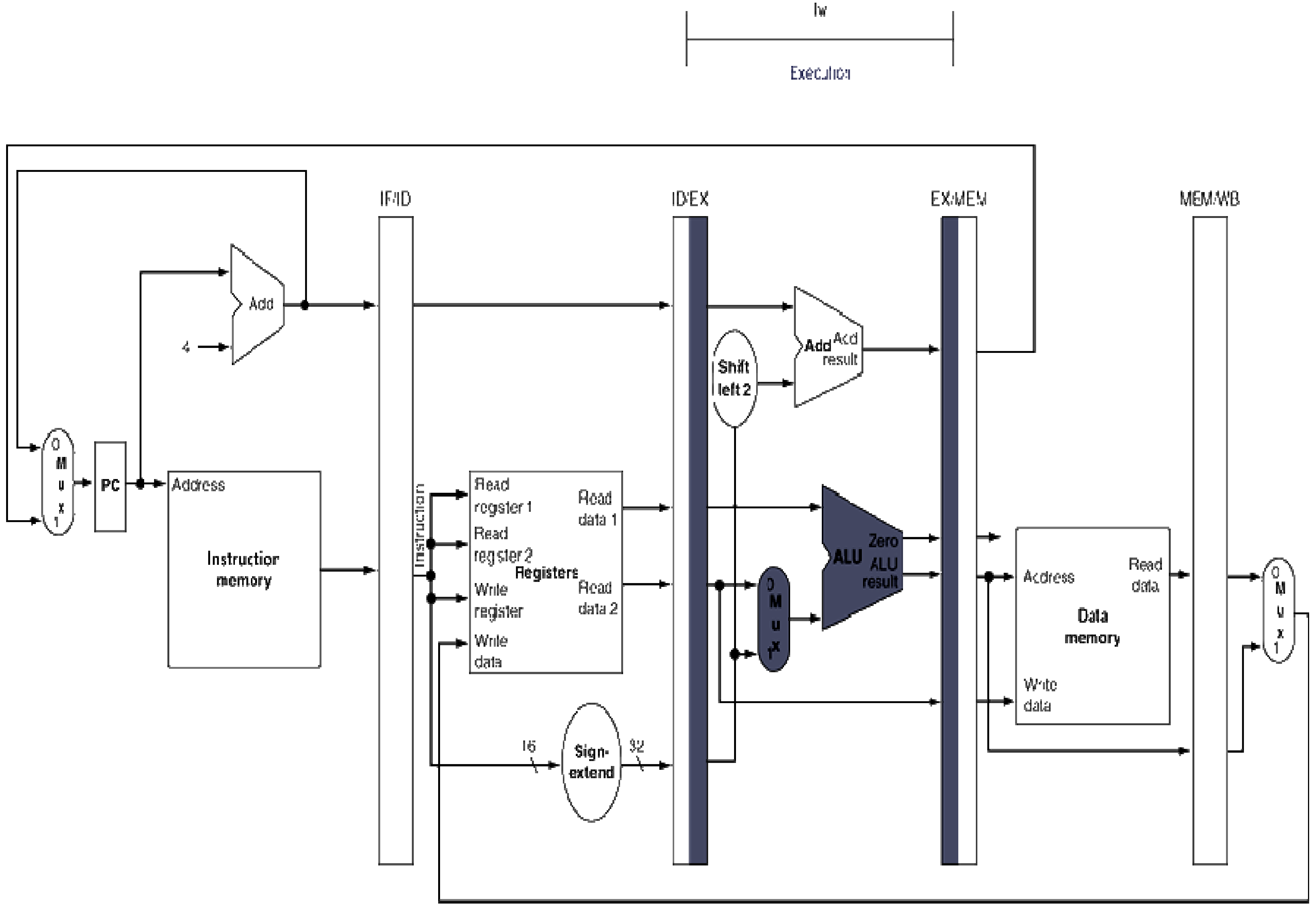
IF for Load, Store, …



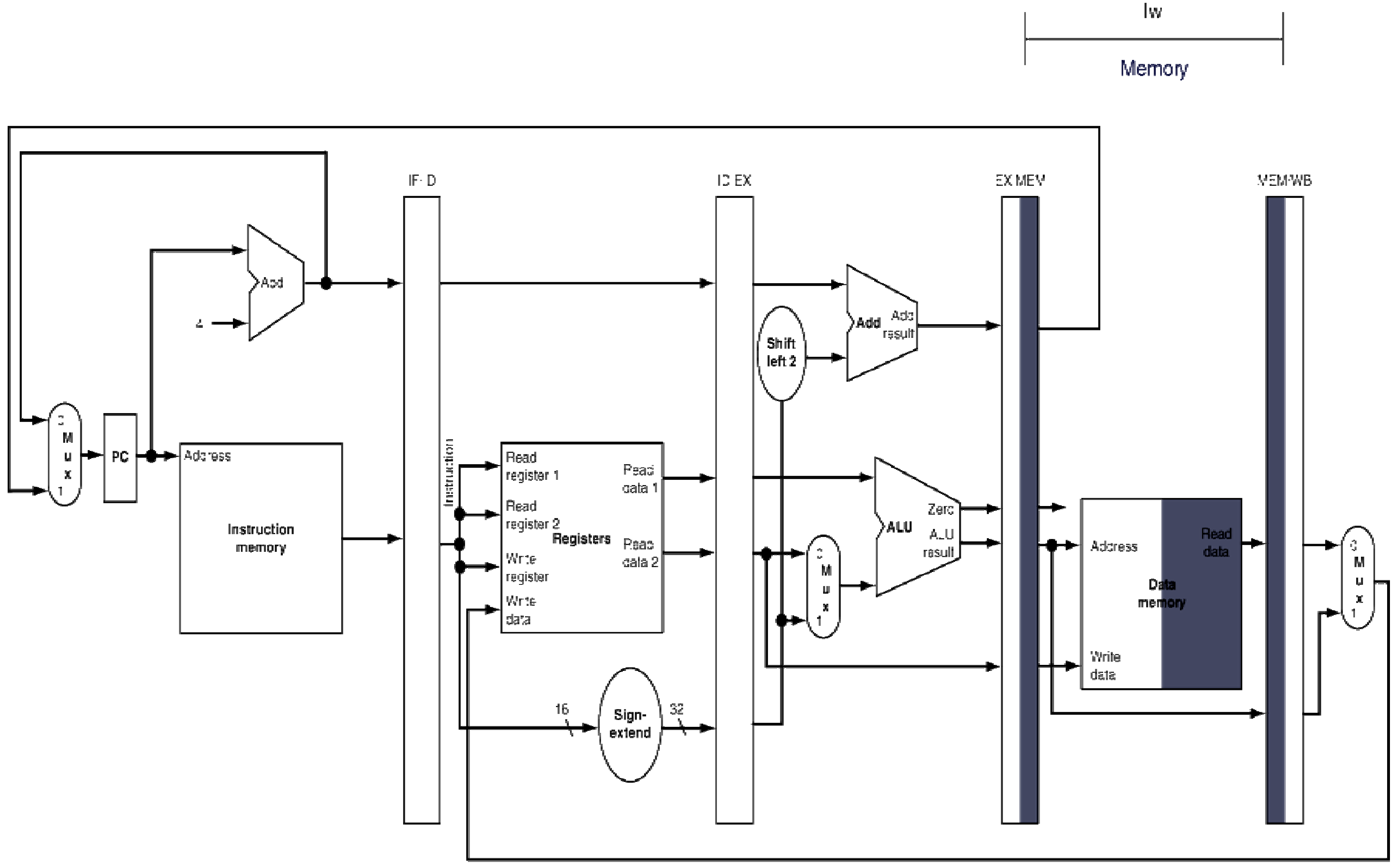
ID for Load, Store, …



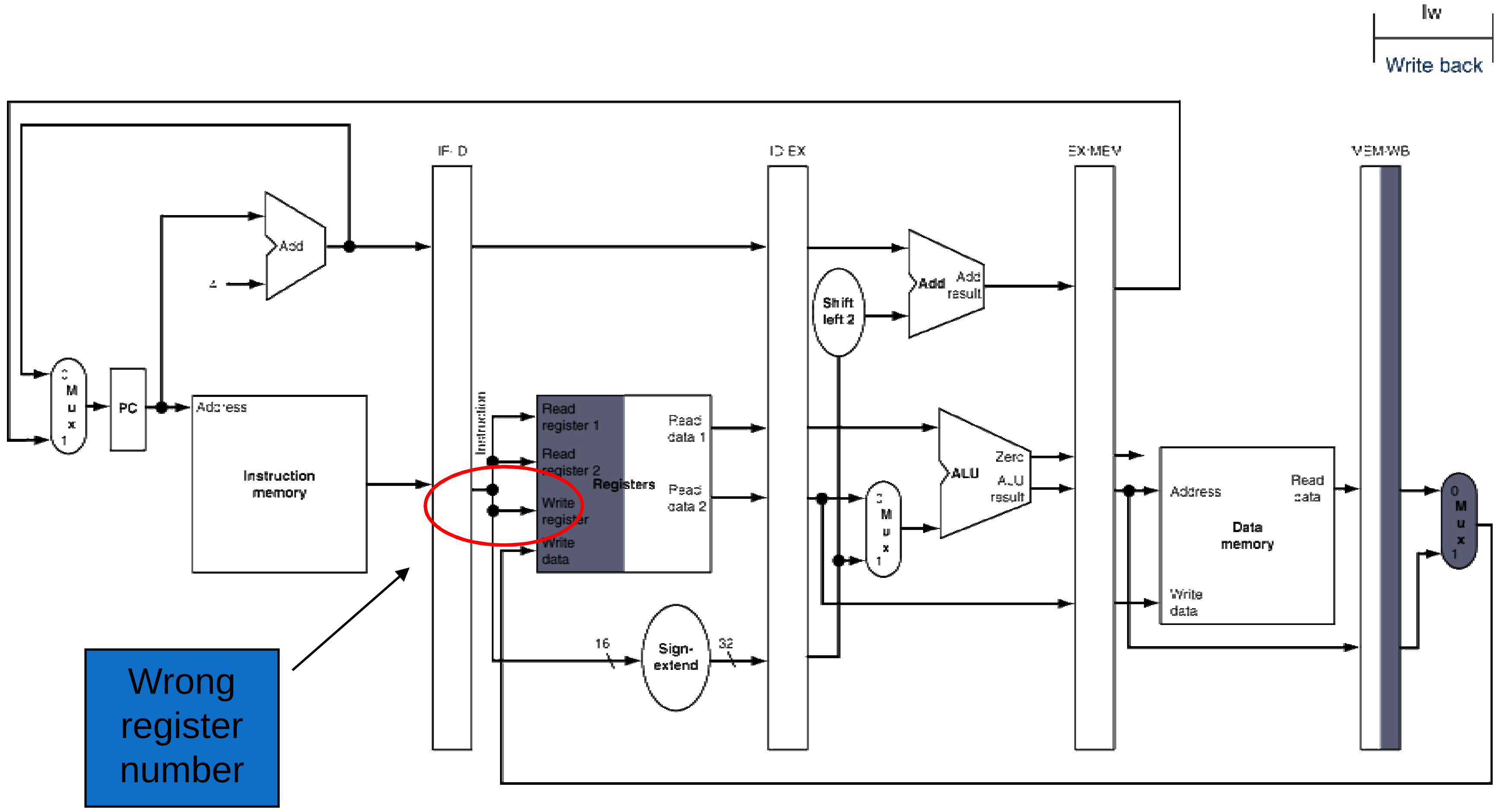
EX for Load


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

MEM for Load

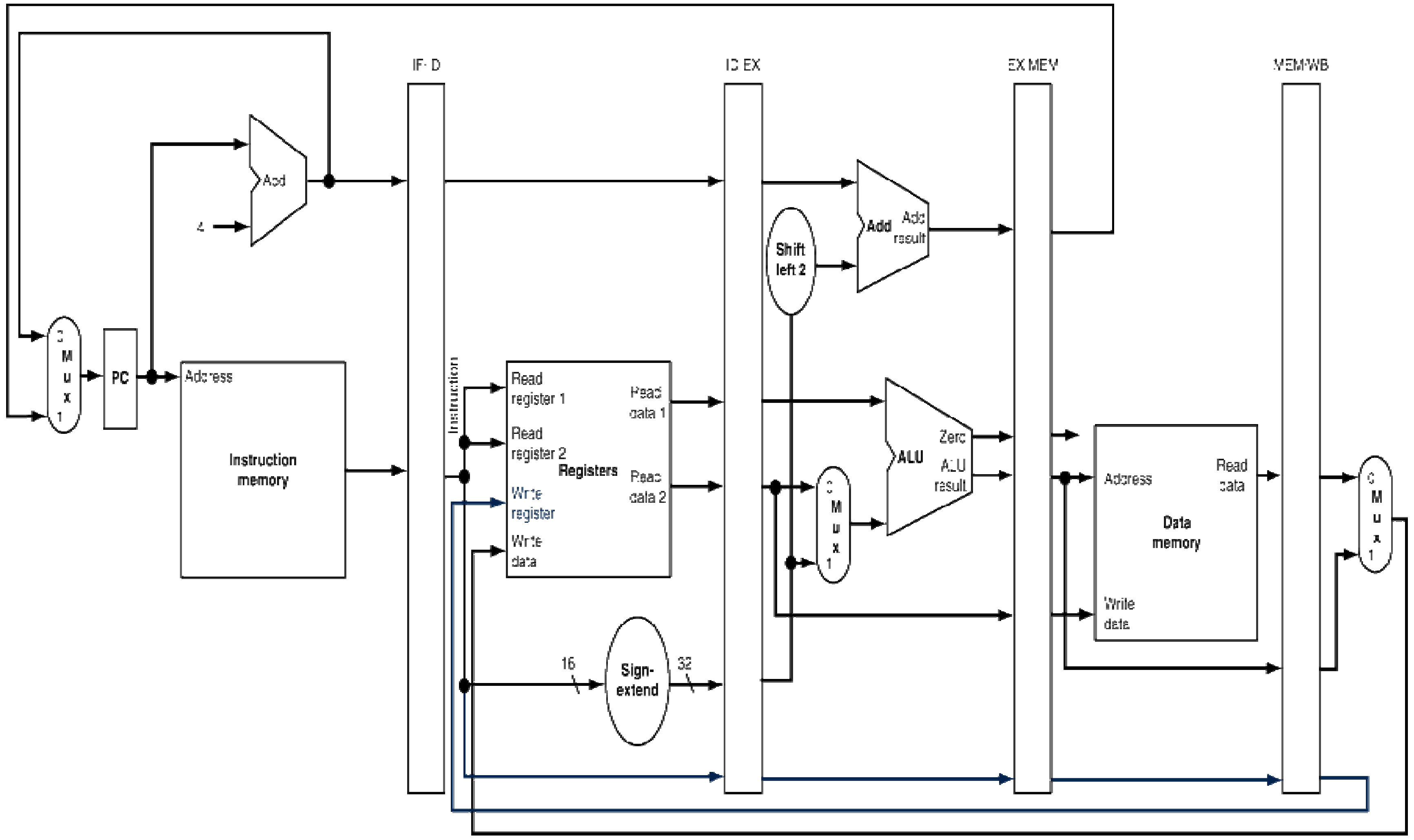

lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
Add
Add result
Shift left 2
0 M u x 1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign-extend
32
0 M u x 1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data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

WB for Load

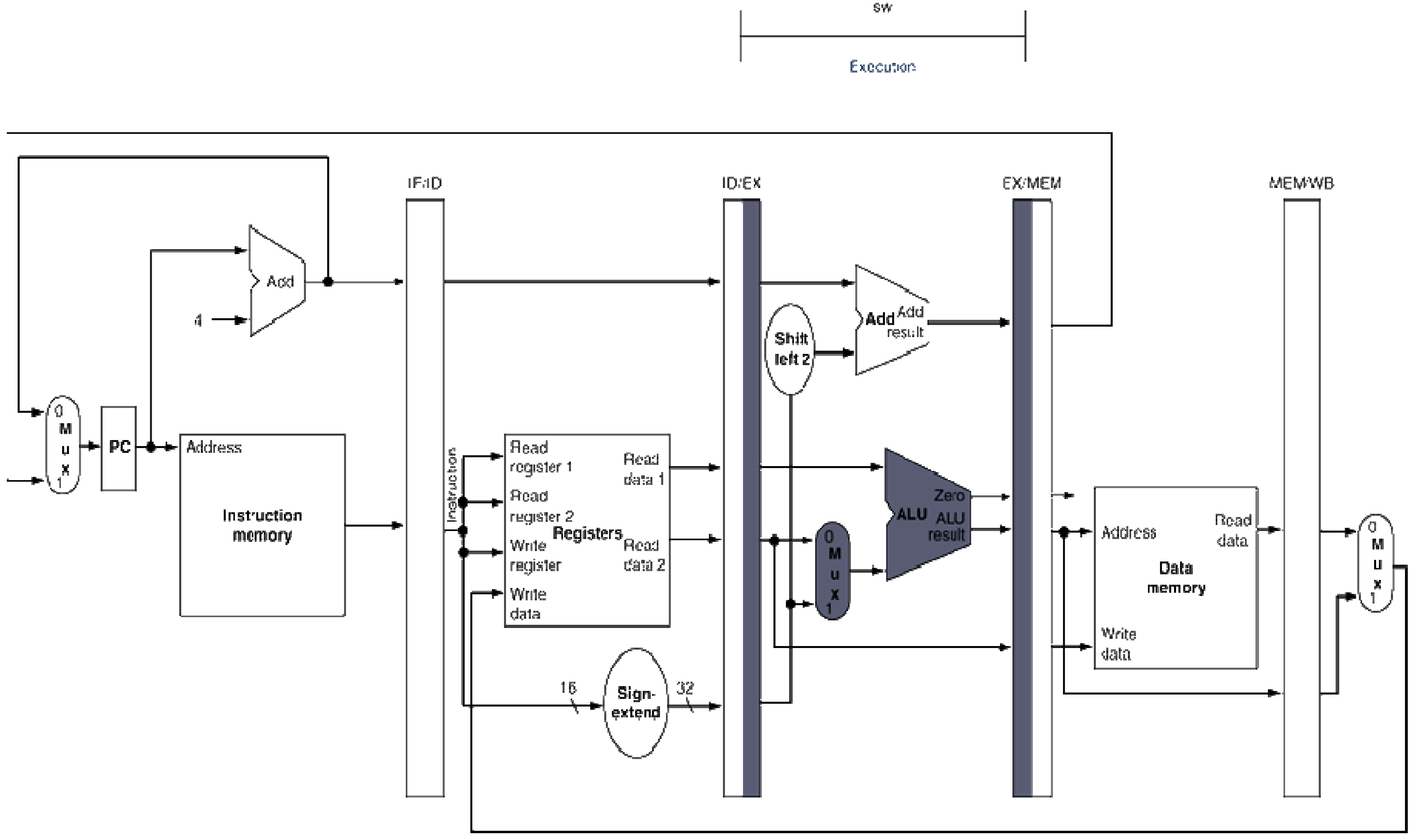


Wrong register number

Corrected Datapath for Load

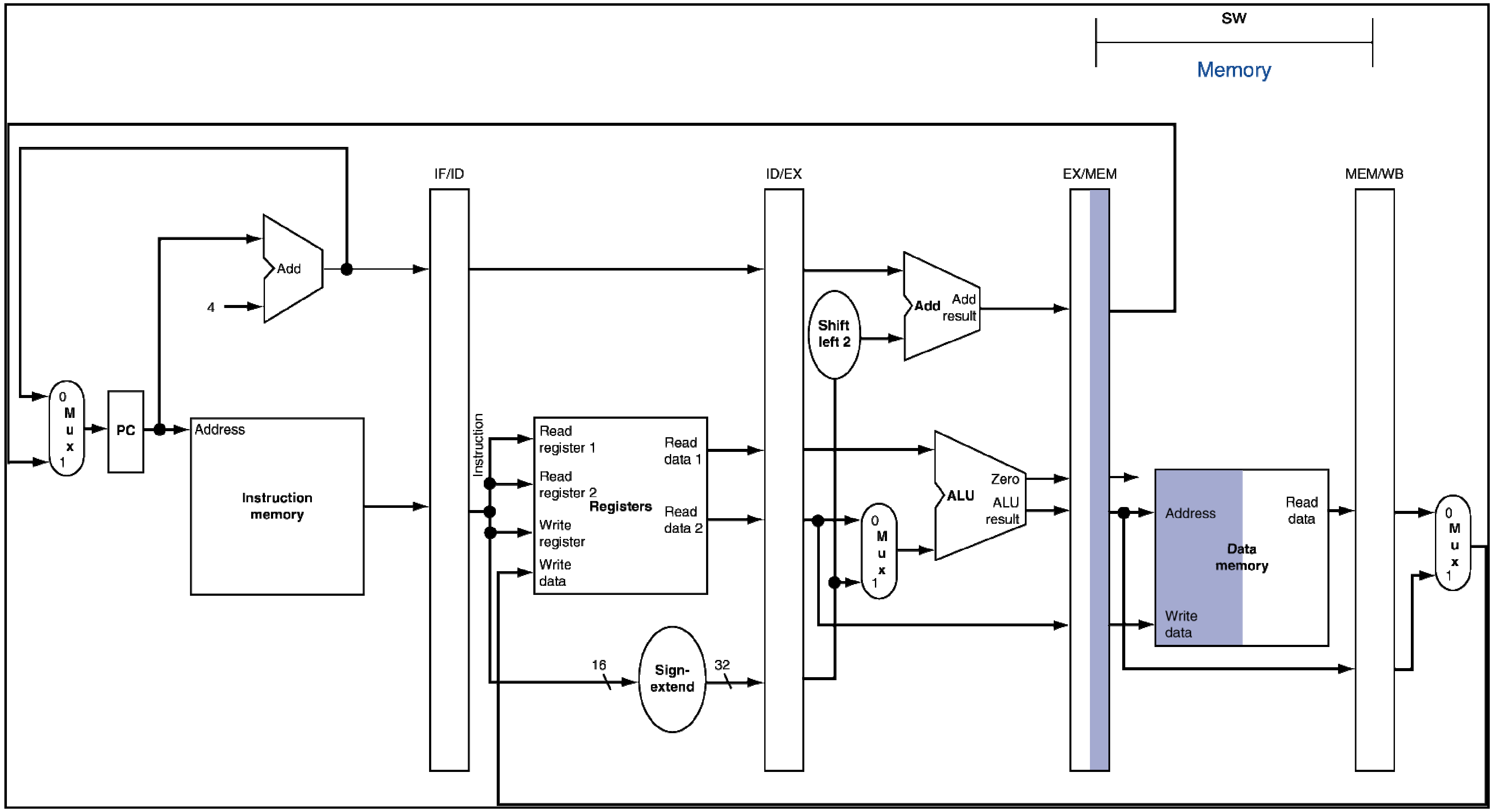


EX for Store



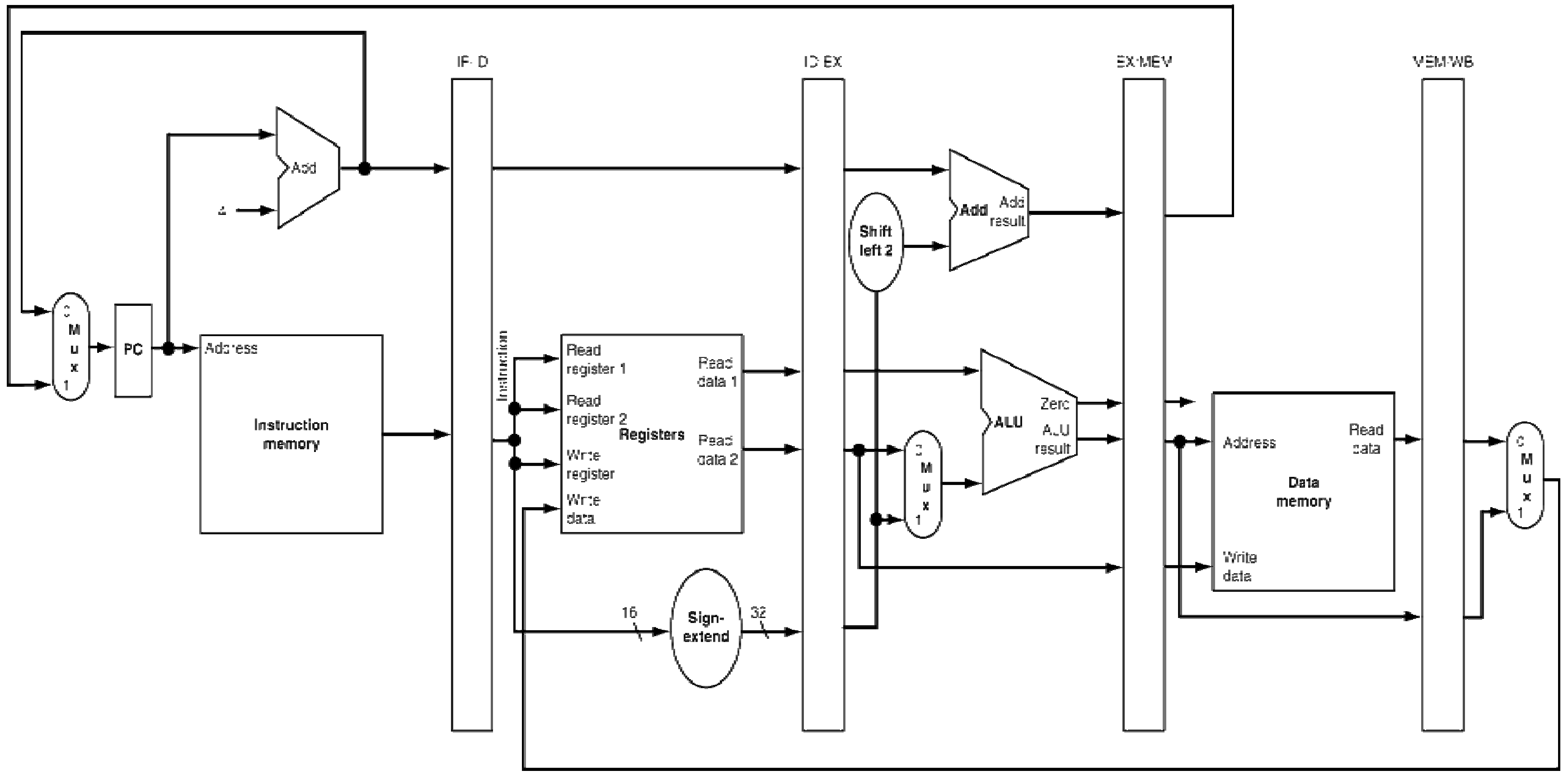
Second register value is loaded into the EX/MEM pipeline register

MEM for Store



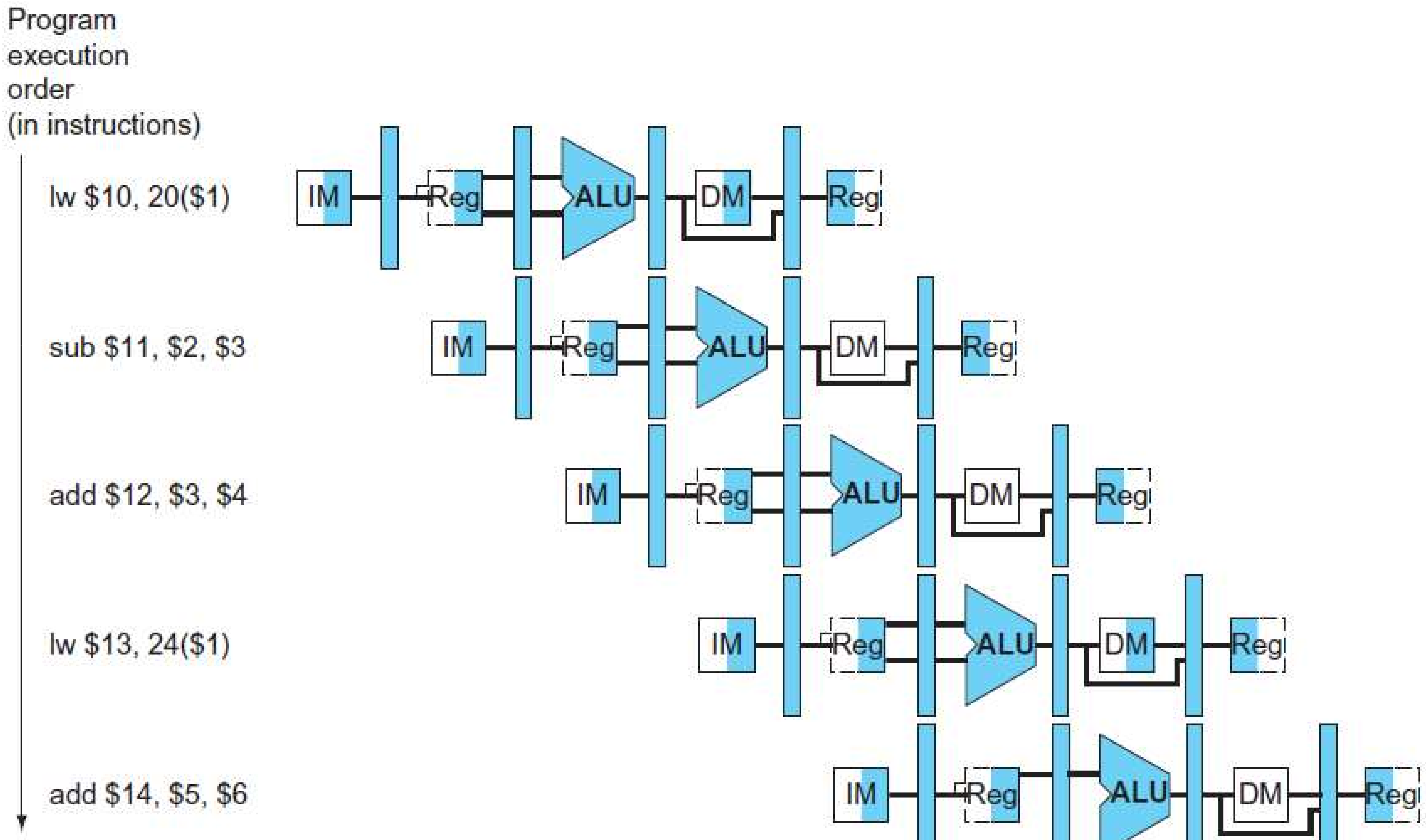
WB for Store

sw
Write-back


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

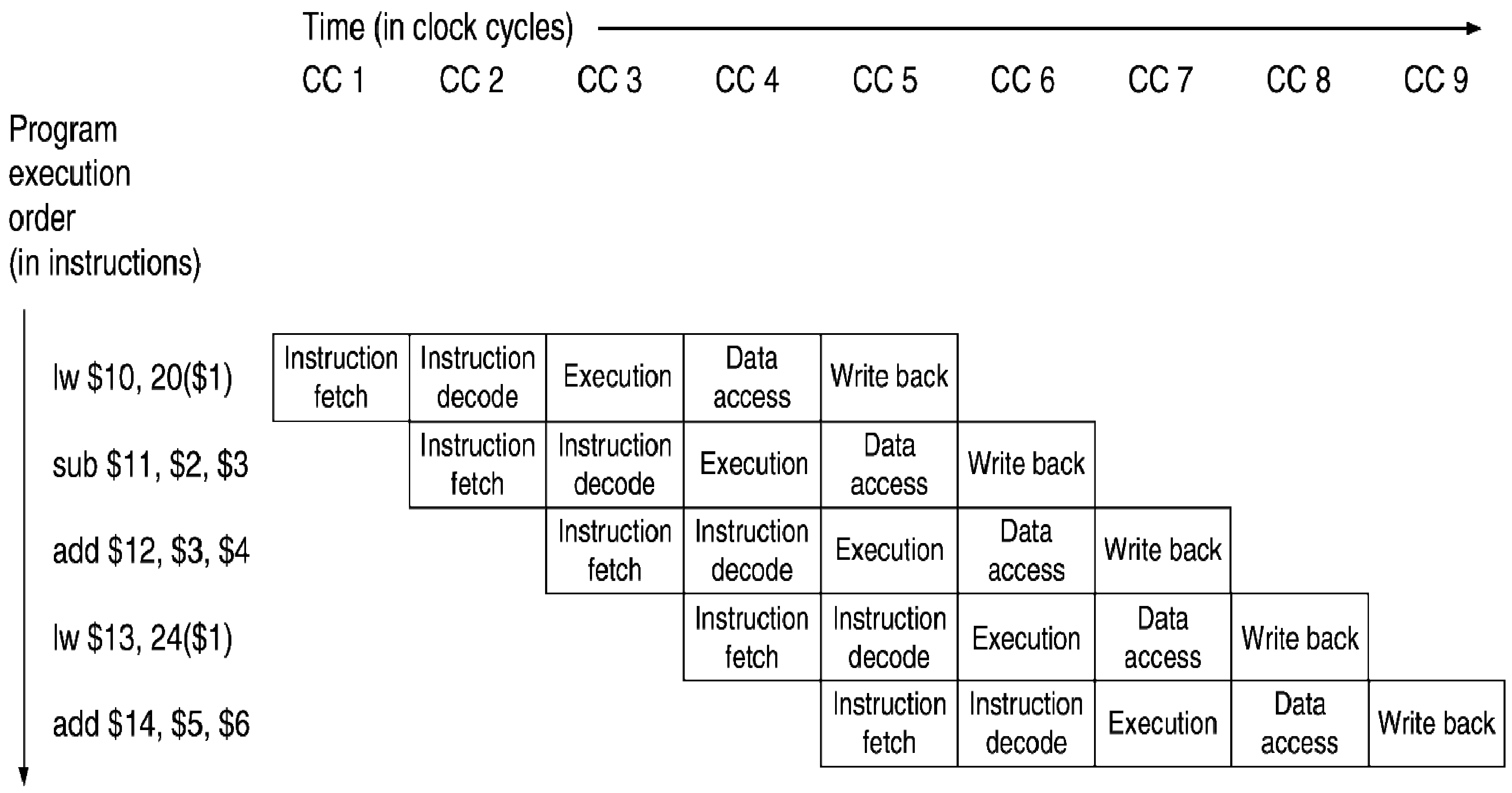
Multi-Cycle Pipeline Diagram

- Form showing resource usage



Multi-Cycle Pipeline Diagram

- Traditional form



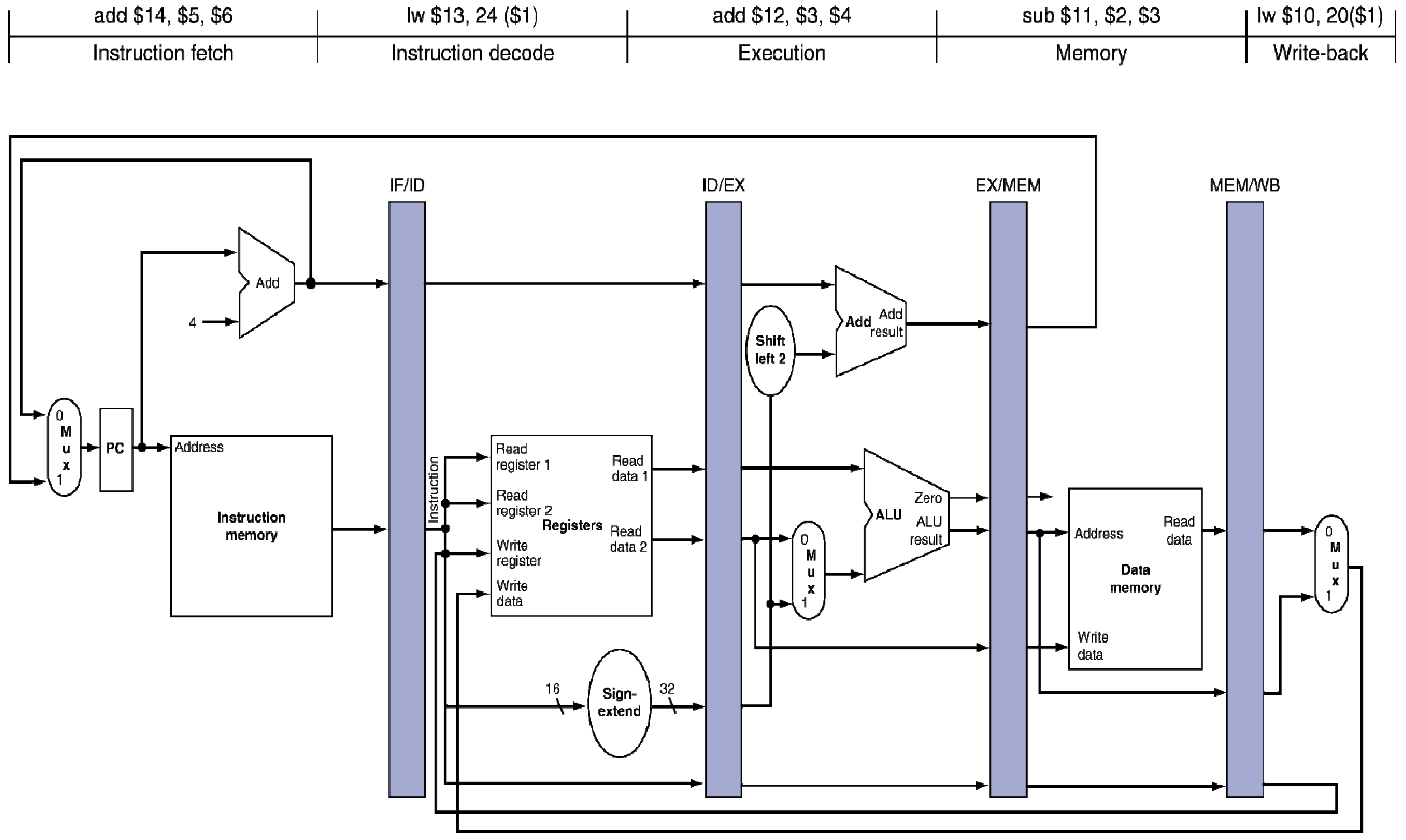
Time (in clock cycles)

Program execution order (in instructions)

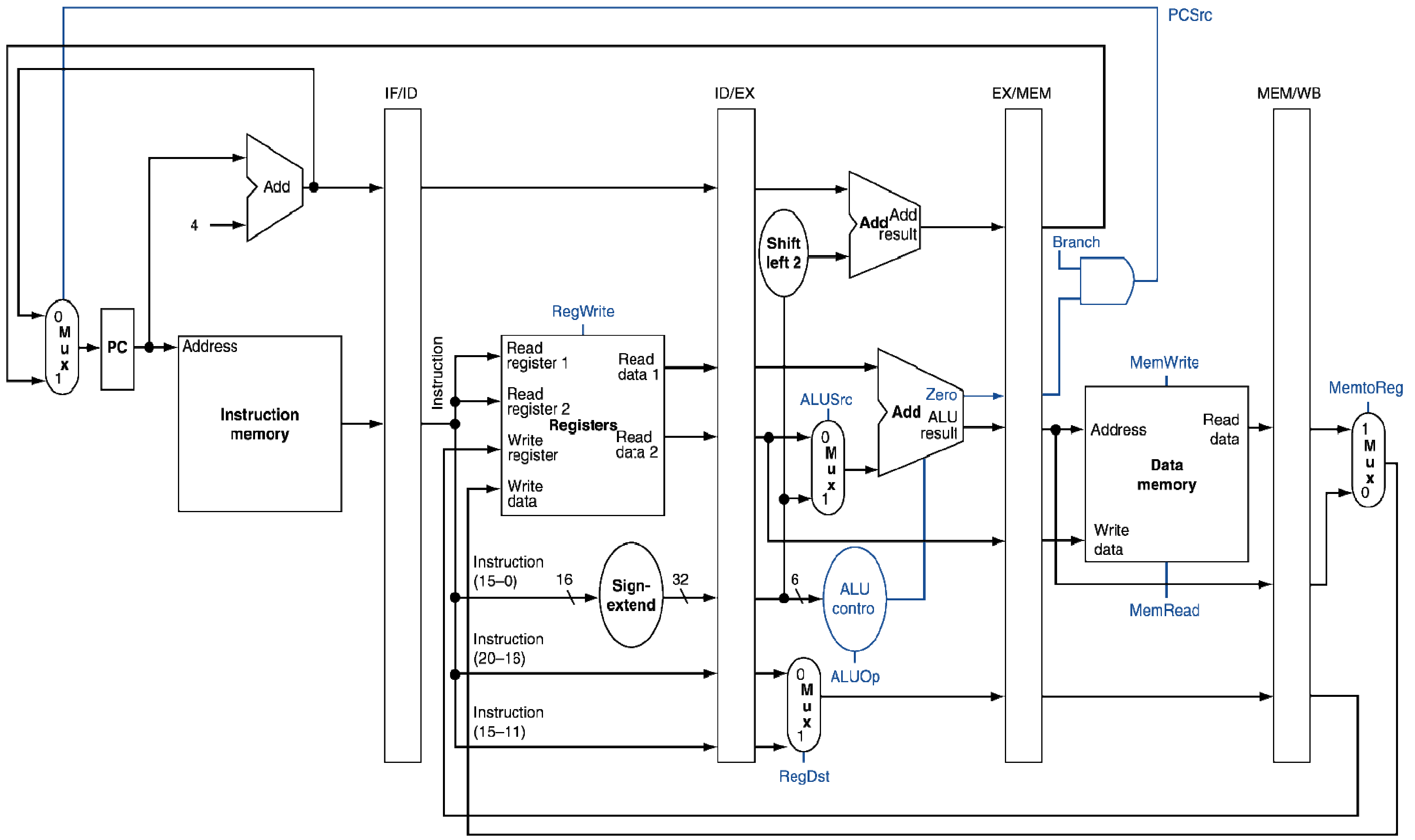
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
lw $10, 20($1)	Instruction fetch	Instruction decode	Execution	Data access	Write back				
sub $11, $2, $3		Instruction fetch	Instruction decode	Execution	Data access	Write back			
add $12, $3, $4			Instruction fetch	Instruction decode	Execution	Data access	Write back		
lw $13, 24($1)				Instruction fetch	Instruction decode	Execution	Data access	Write back	
add $14, $5, $6					Instruction fetch	Instruction decode	Execution	Data access	Write back

Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle



Pipelined Control

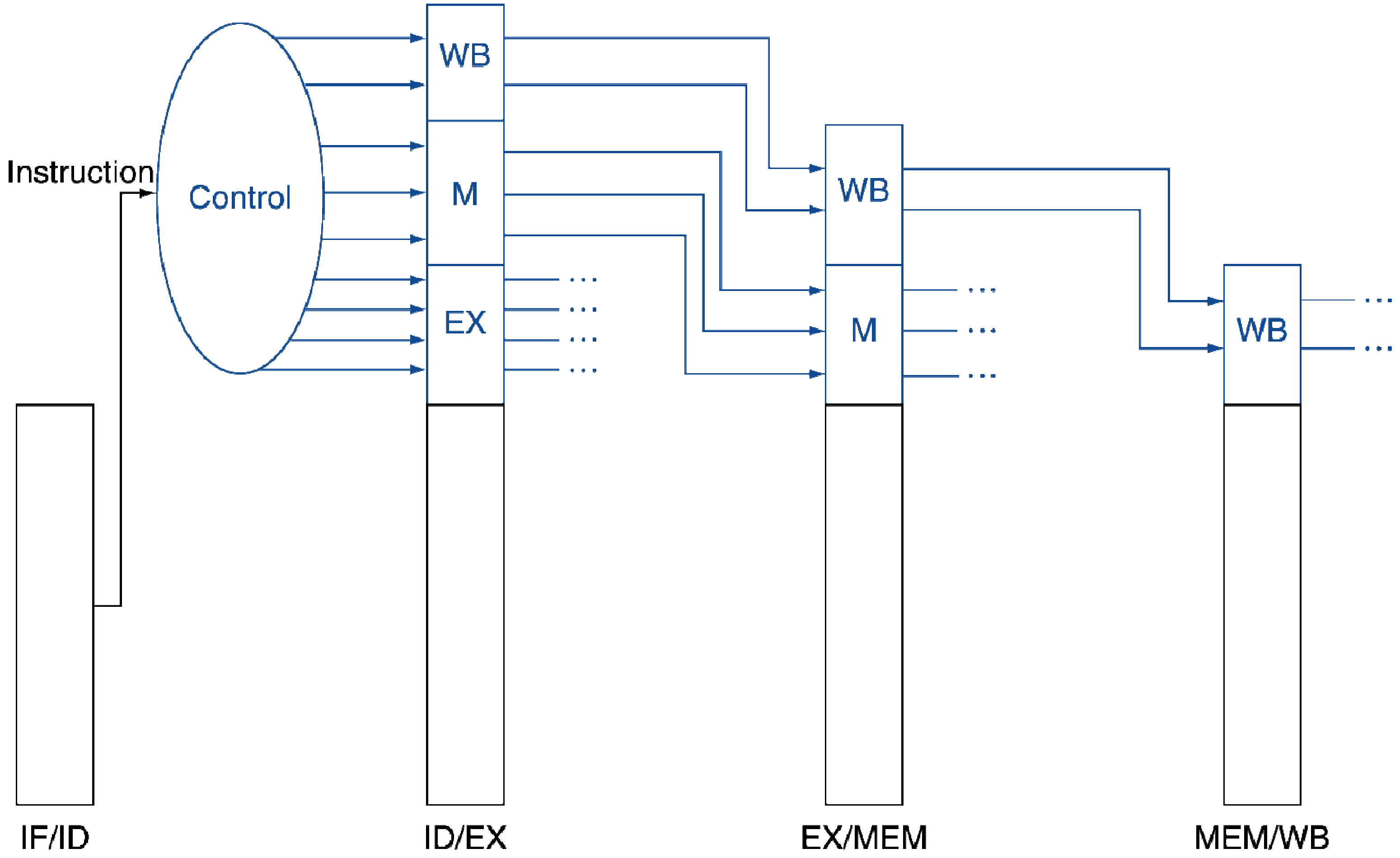


Pipelined Control

- Control signals derived from instruction
- 1. *Instruction fetch:*
  - *The control signals to read instruction memory and to* write the PC are always asserted
- 2. *Instruction decode/register file read:*
  - *As in the previous stage, the same thing* happens at every clock cycle, so there are no optional control lines to set.
- 3. *Execution/address calculation:*
  - *The signals to be set are RegDst, ALUOp,* and ALUSrc
- 4. *Memory access:*
  - *The control lines set in this stage are Branch, MemRead, and* MemWrite
- 5. *Write-back:*
  - *The two control lines are MemtoReg and* Reg- Write

Pipelined Control

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



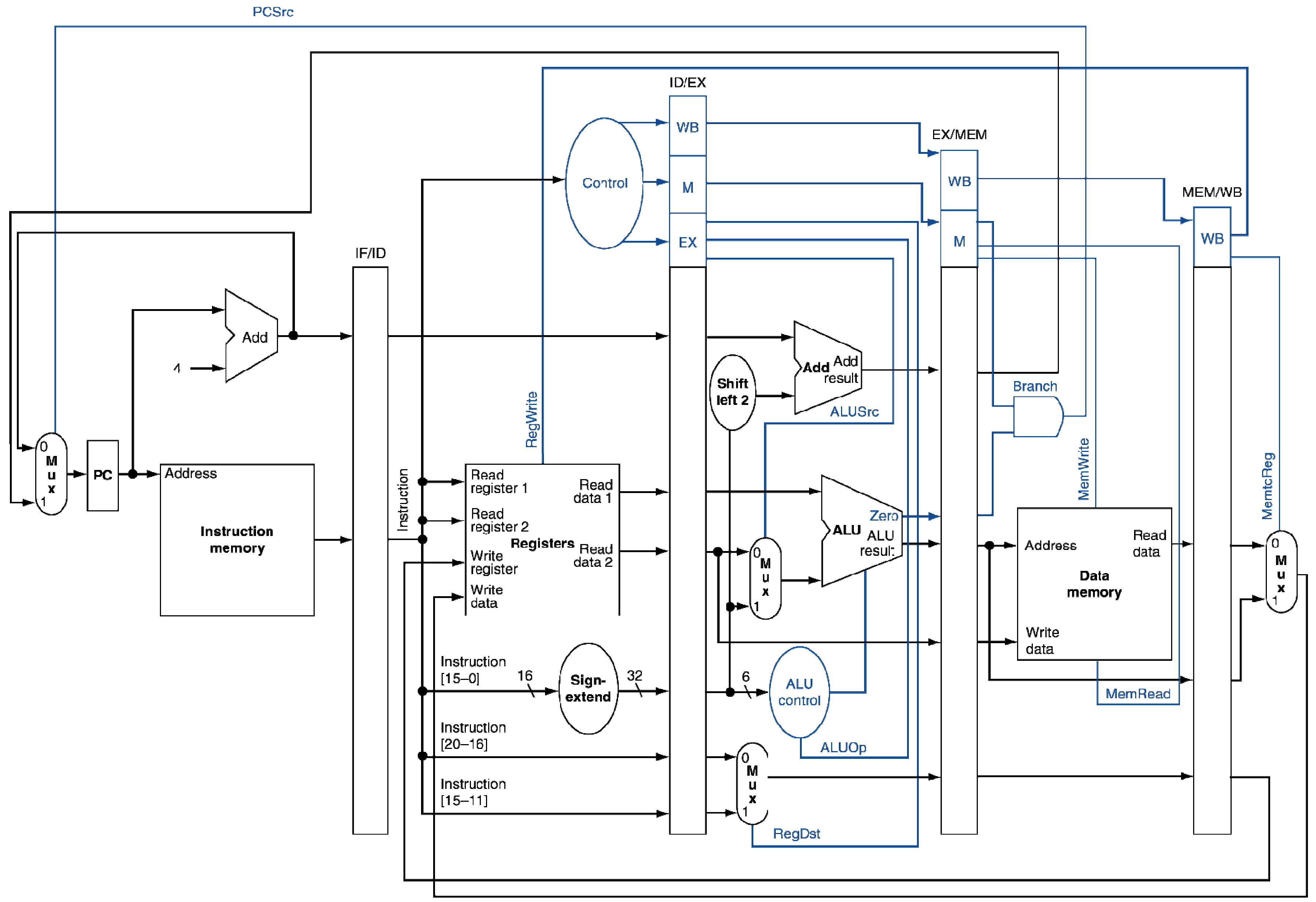
Pipelined Control

- Control signals derived from instruction

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

Ins		ID/EX	EX/MEM	MEM/WB
lw	$10, 20($1)	0001	010	11
sub	$11, $2, $3	1100	000	10
and	$12, $4, $5	1100	000	10
or	$13, $6, $7	1100	000	10
add	$14, $8, $9	1100	000	10

Pipelined Control



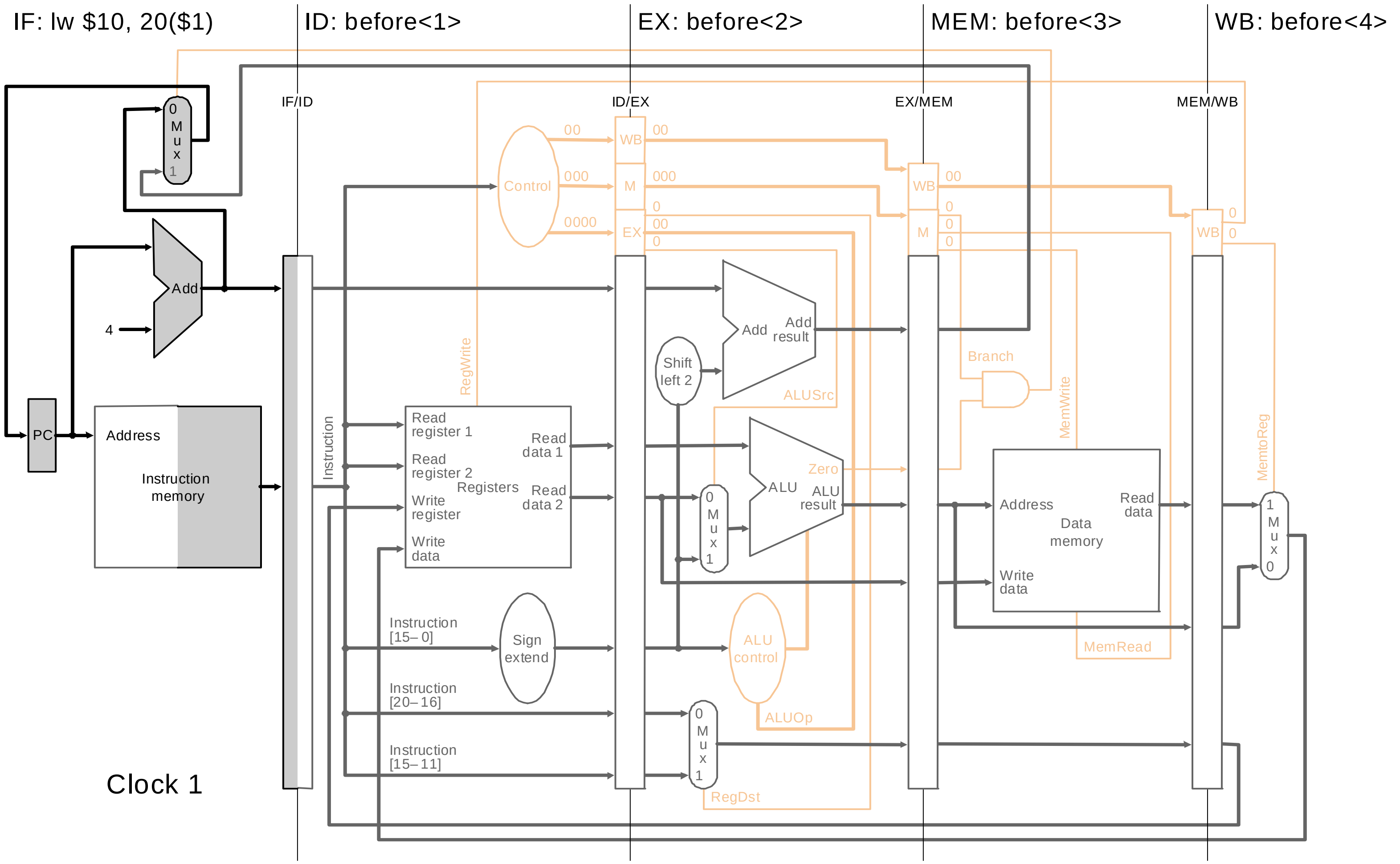
Example

lw	\$10, 20(\$1)
sub	\$11, \$2, \$3
and	\$12, \$4, \$5
or	\$13, \$6, \$7
add	\$14, \$8, \$9

Ins		ID/EX	EX/MEM	MEM/WB
lw	\$10, 20(\$1)	0001	010	11
sub	\$11, \$2, \$3	1100	000	10
and	\$12, \$4, \$5	1100	000	10
or	\$13, \$6, \$7	1100	000	10
add	\$14, \$8, \$9	1100	000	10

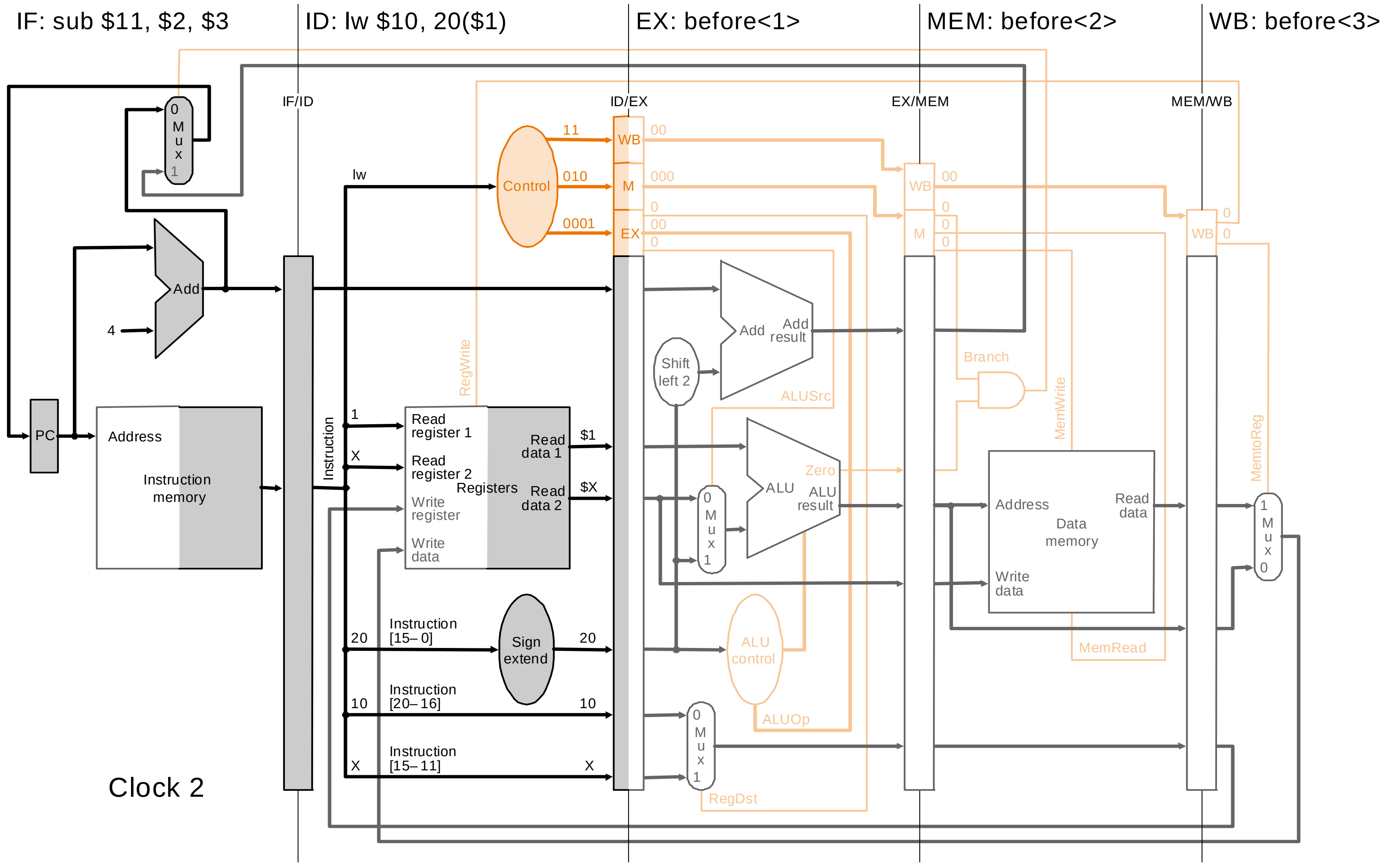
Cycle 1

Ins		**ID/EX**	**EX/MEM**	**MEM/WB**
lw	$10, 20($1)	0001	010	11
sub	$11, $2, $3	1100	000	10
and	$12, $4, $5	1100	000	10
or	$13, $6, $7	1100	000	10
add	$14, $8, $9	1100	000	10



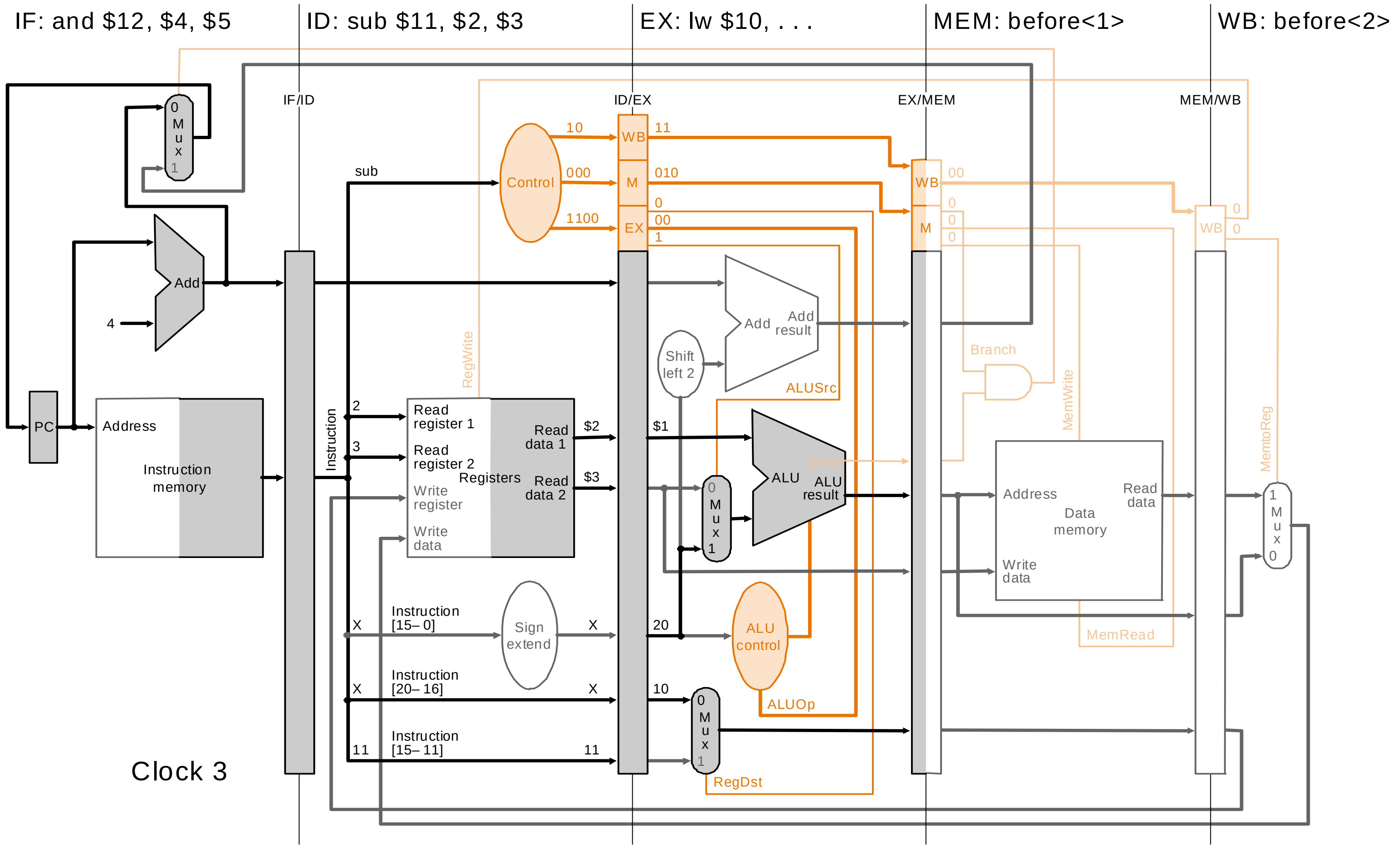
Cycle 2

Ins	ID/EX	EX/MEM	MEM/WB
lw	0001	010	11

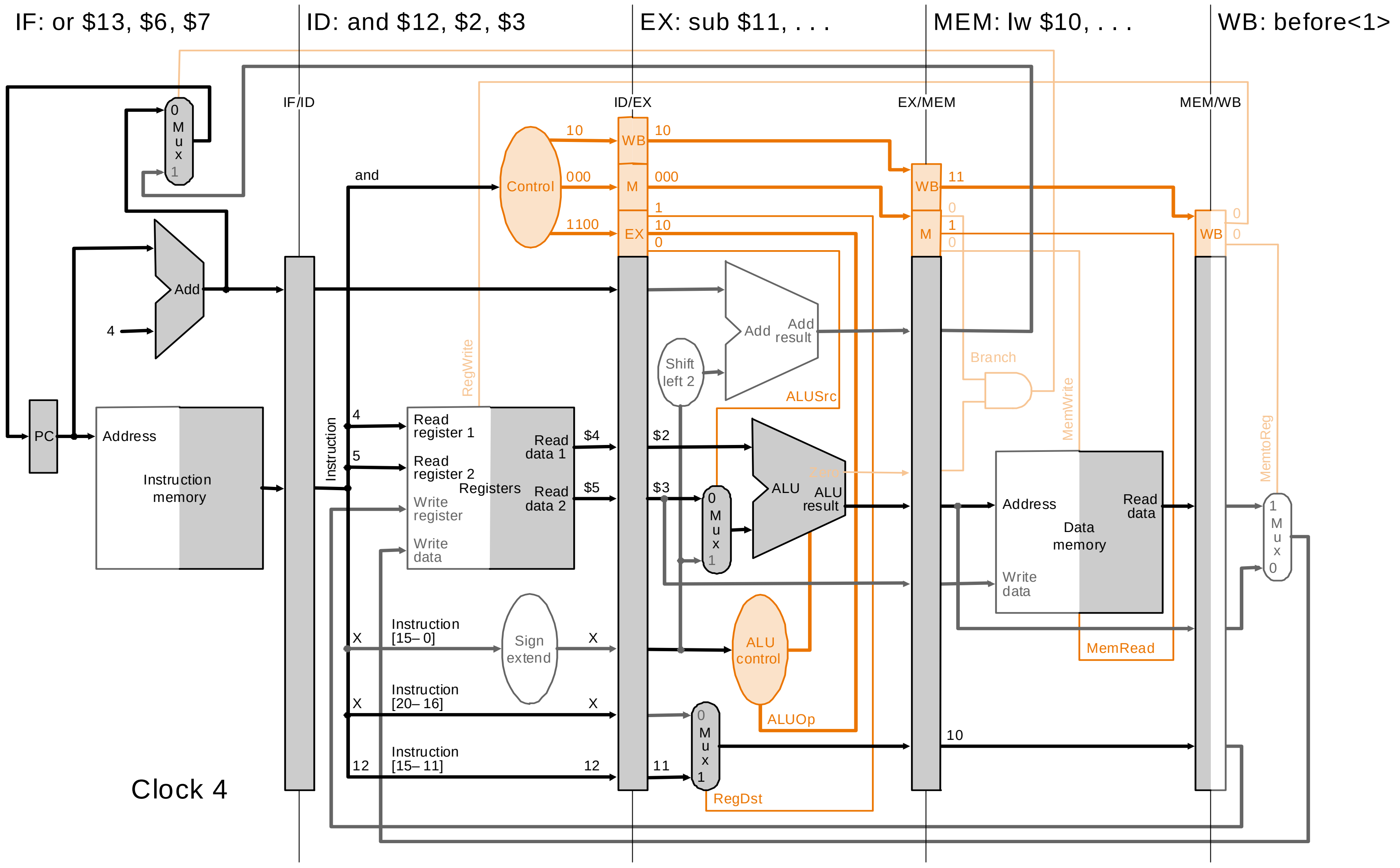


Cycle 3

Ins	ID/EX	EX/MEM	MEM/WB
lw	0001	010	11
sub	1100	000	10

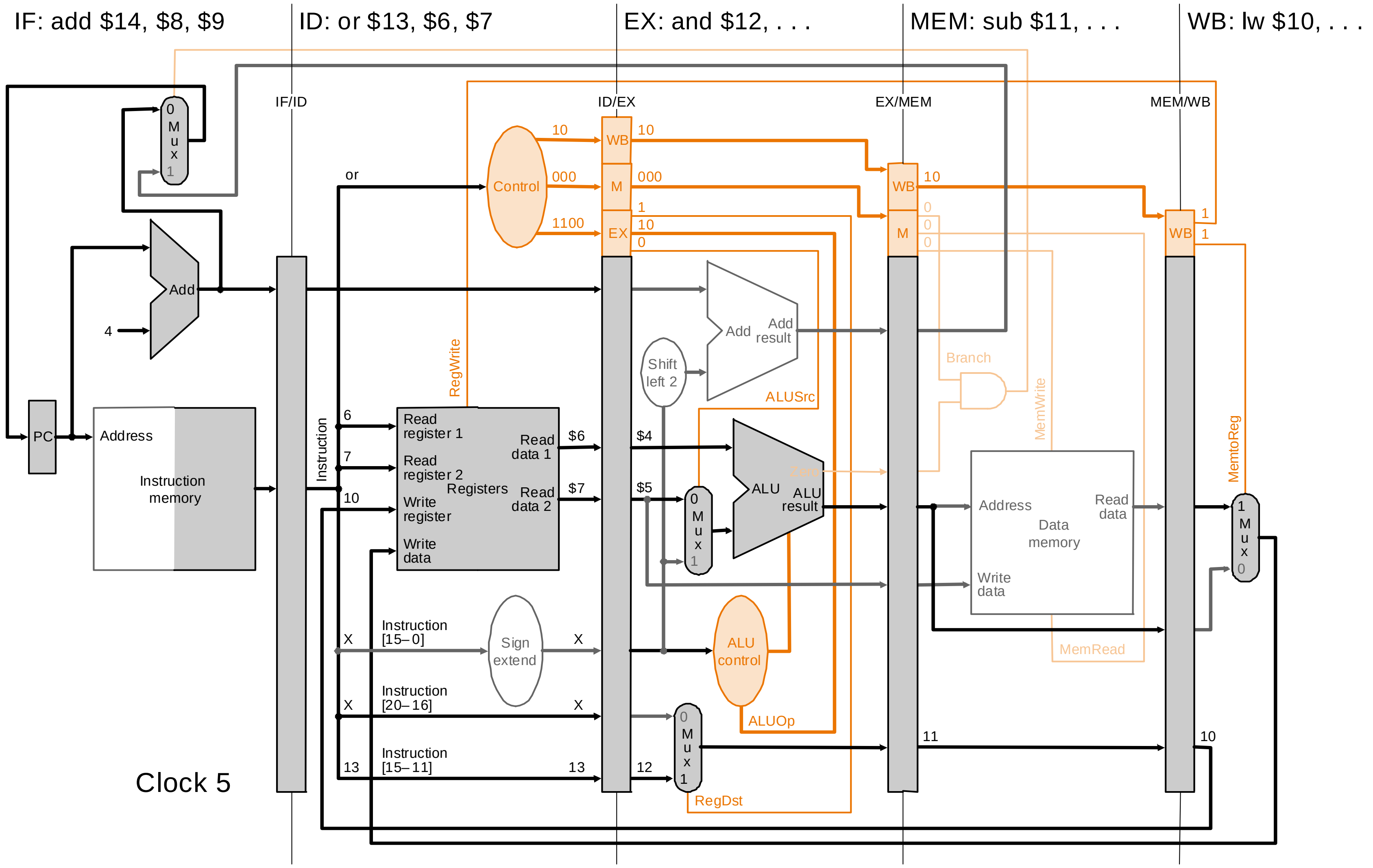


Cycle 4



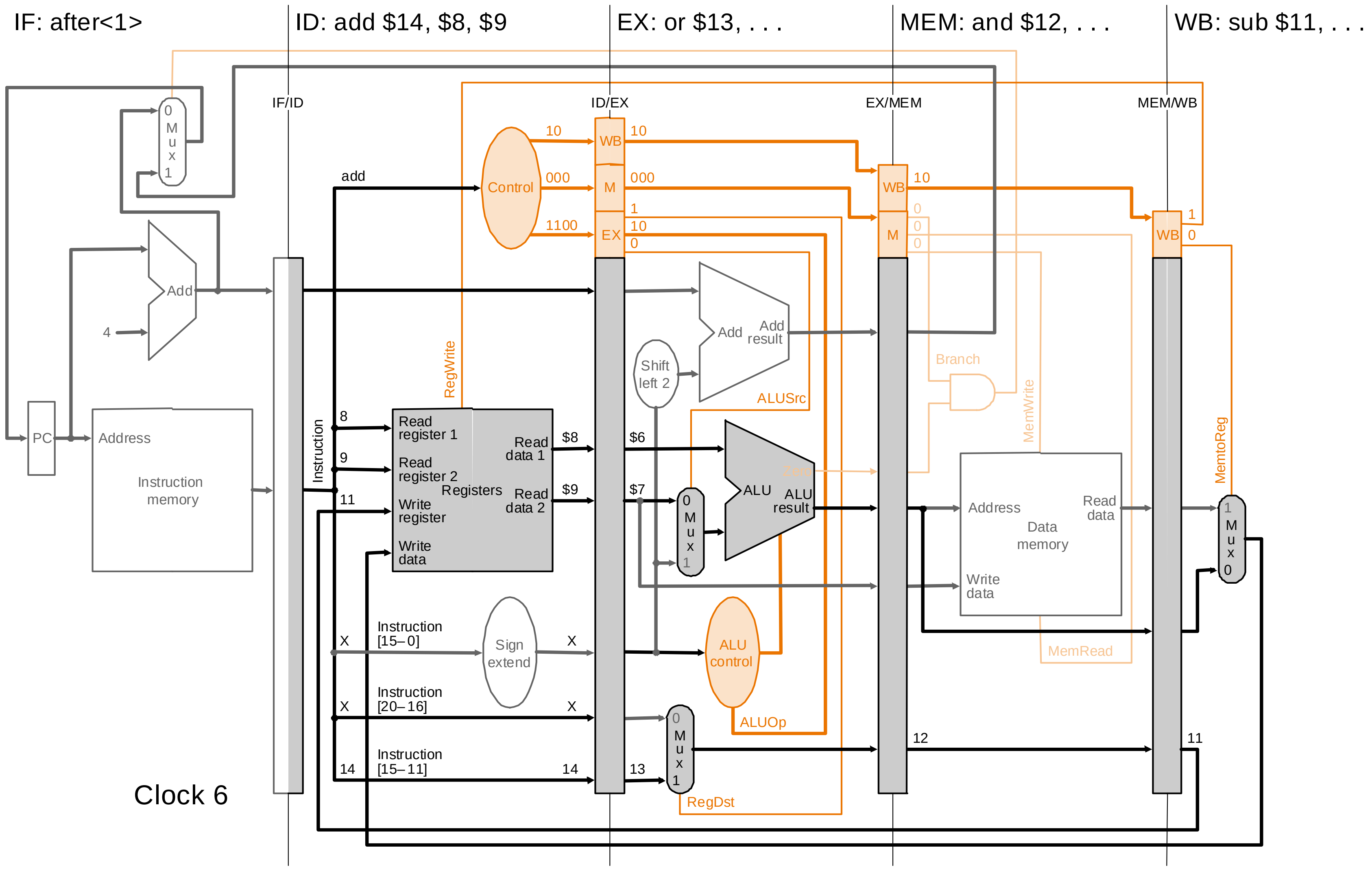
Clock 4

Cycle 5



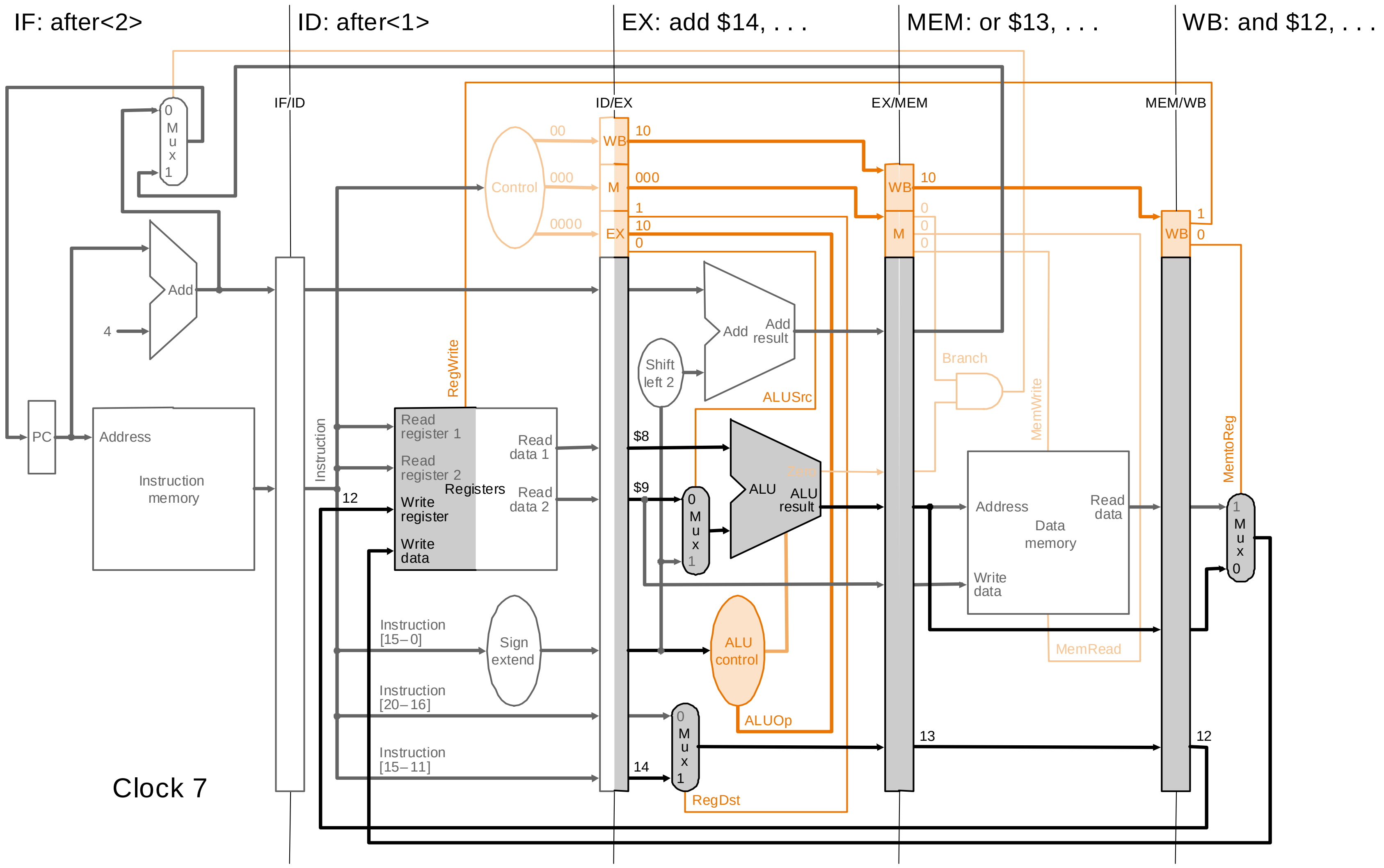
Clock 5

Cycle 6



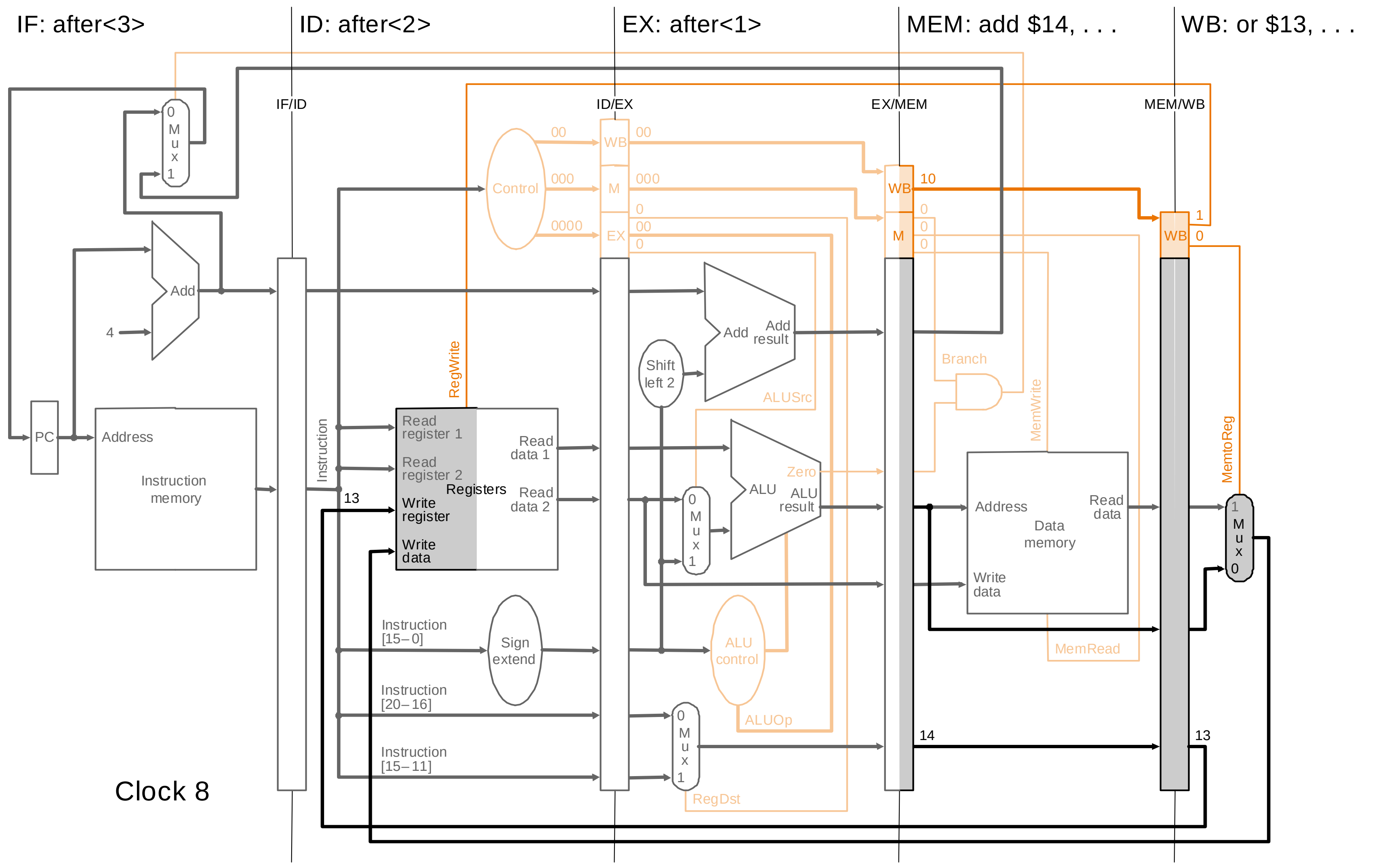
Clock 6

Cycle 7



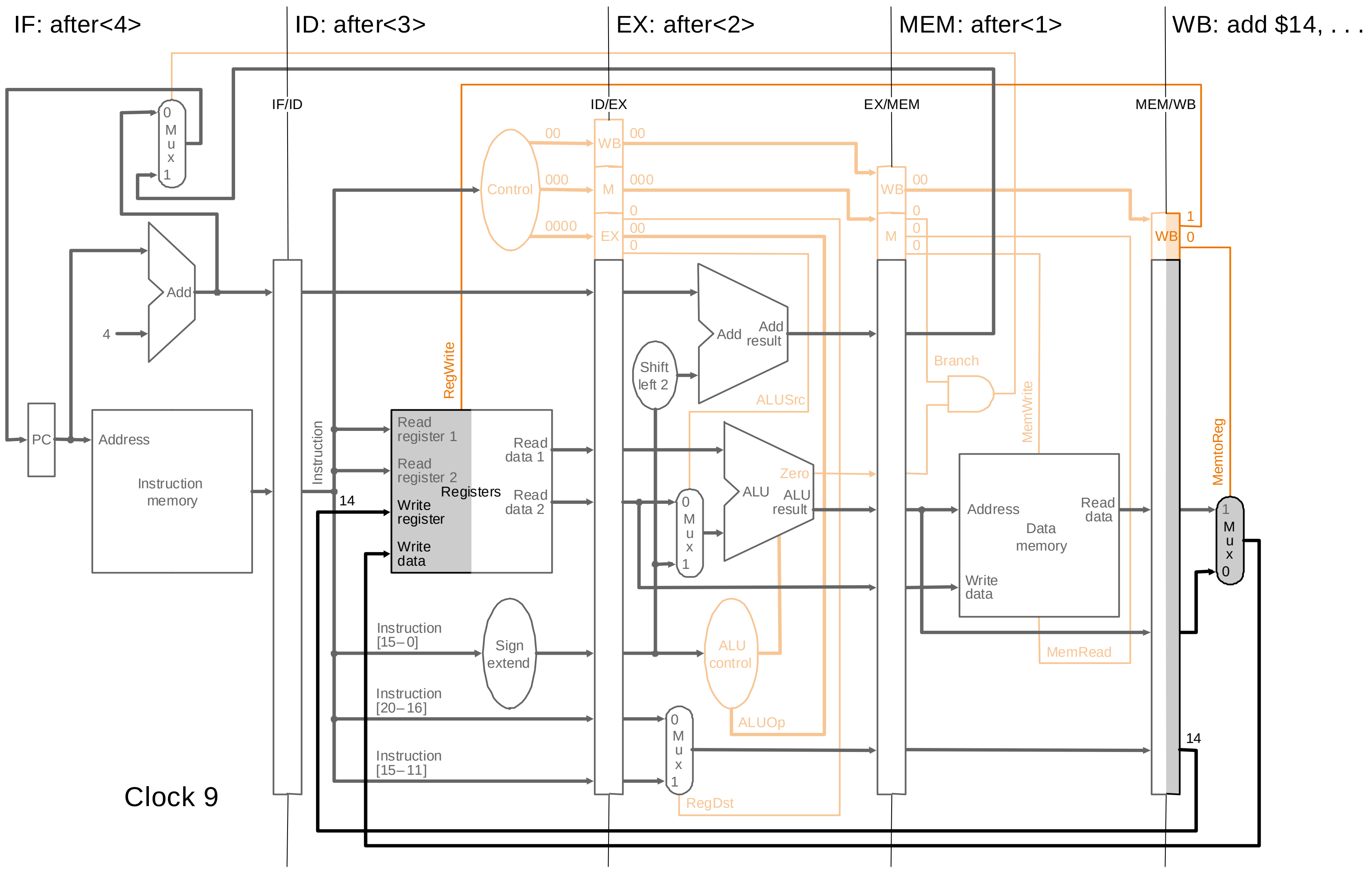
Clock 7

Cycle 8



IF: after<3>
ID: after<2>
EX: after<1>
MEM: add $14, . . .
WB: or $13, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
00
000
0000
10
1
0
14
13
PC
Add
4
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
Sign extend
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
Shift left 2
Add result
ALUSrc
ALU
Zero
ALU result
ALU control
ALUOp
RegDst
Branch
MemWrite
Data memory
Read data
MemRead
MemtoReg
Mux
Clock 8

Cycle 9



Clock 9