숭실대학교
Soongsil University

Computer Architecture

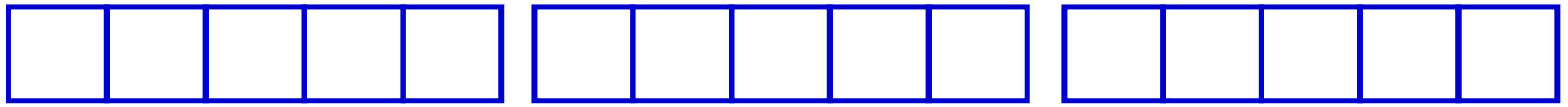
강의 #5: Enhancing Performance with Pipelining (1)

2021년 1학기

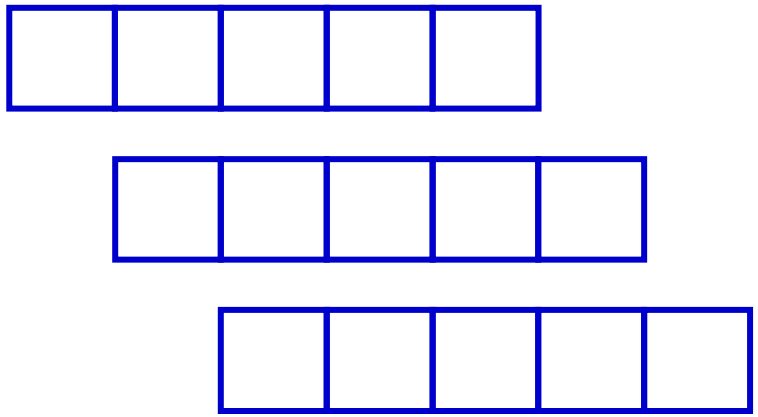
Young Geun Kim (김영근)

Pipelined Instruction Execution

- Sequential Execution



- Pipelined Execution



ADD $3, $1, $2

SUB $4, $5, $6

AND $7, $8, $9

Pipelining

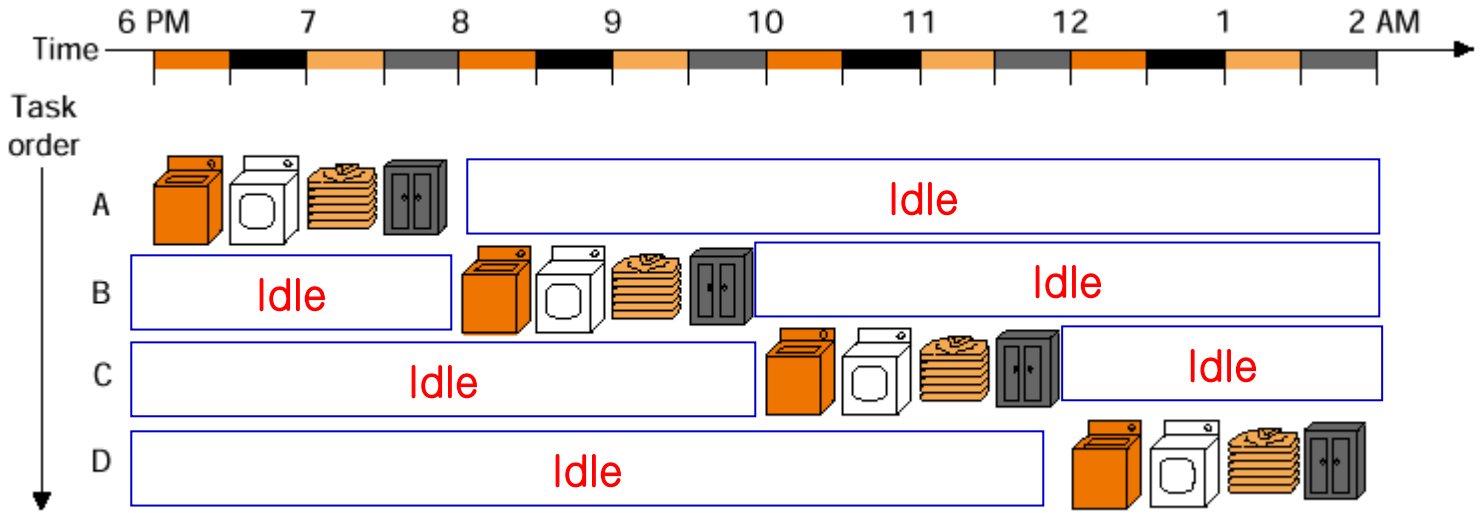
- **Pipelining은 1 Cycle에 여러 명령어가 동시에 수행될 수 있도록 구현하는 기술임**

- **장점**
  - Functional Unit들의 사용률을 더욱 향상시킴
  - 성능을 매우 크게 향상시킴

- **단점**
  - 하드웨어 구성 및 Control Logic이 매우 복잡해짐
  - "Hazard"라는 실행 결과에 영향을 미칠 수 있는 요소가 발생함

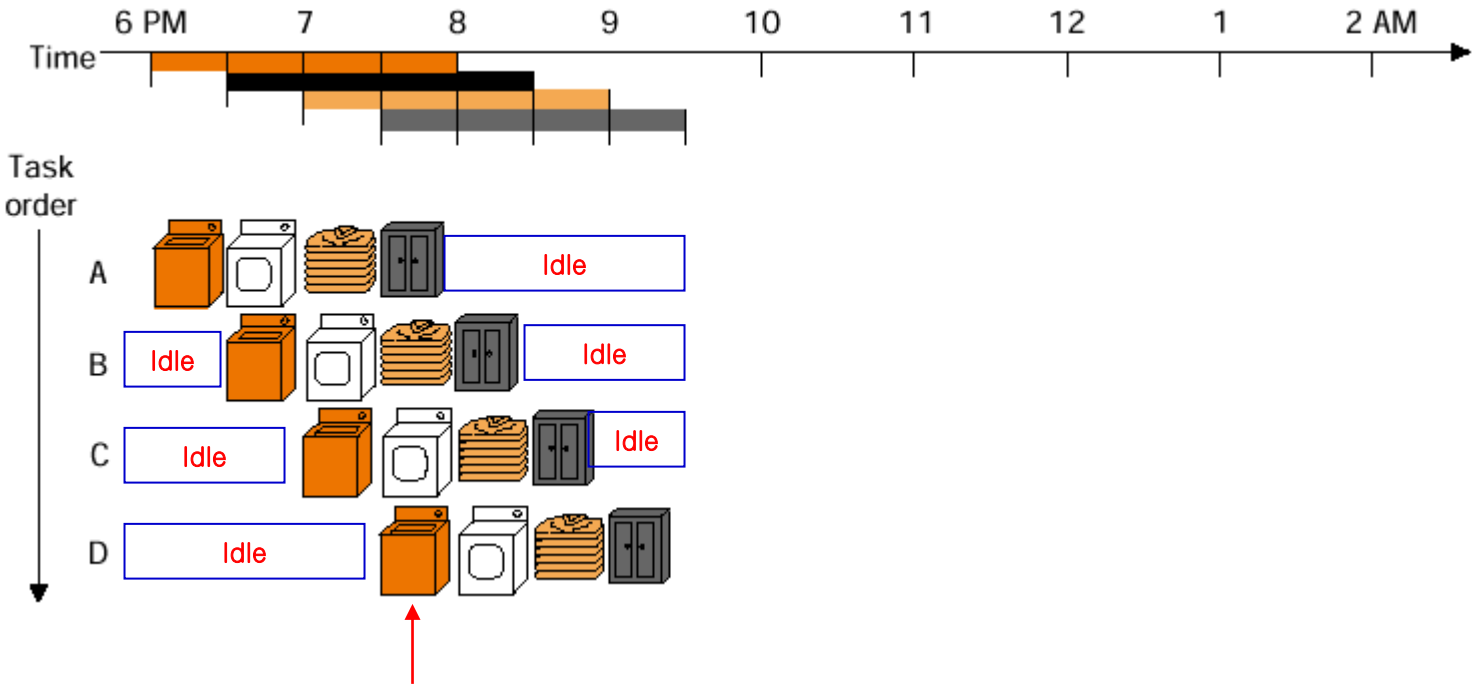
Sequential Processing

- **새로운 작업은 현재 작업이 끝난 후에만 시작될 수 있음**
  - A가 일하는 동안, B는 기다림
  - A가 끝나면, B가 일을 시작함



Pipelined Processing

- **새로운 작업은 현재 작업 Set 중 첫 번째 작업이 끝나면 시작될 수 있음**
  - A가 세탁기를 사용하는 동안 B는 기다림
  - A가 세탁기 사용을 끝내고 건조기를 사용하는 동안 B는 세탁기를 사용할 수 있음



Basic Pipeline

	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

간단한 비교

- **다음과 같은 시스템을 구축한다고 가정**
  - Register File 접근 시간: 1 ns
  - ALU 연산 시간: 2 ns
  - 메모리 접근 시간: 2 ns

- **Single-Cycle과 Multi-Cycle, 그리고 Pipelined Approach를 비교해보자**

간단한 비교 (Cont'd)

- 각 명령어 종류를 고려해보면

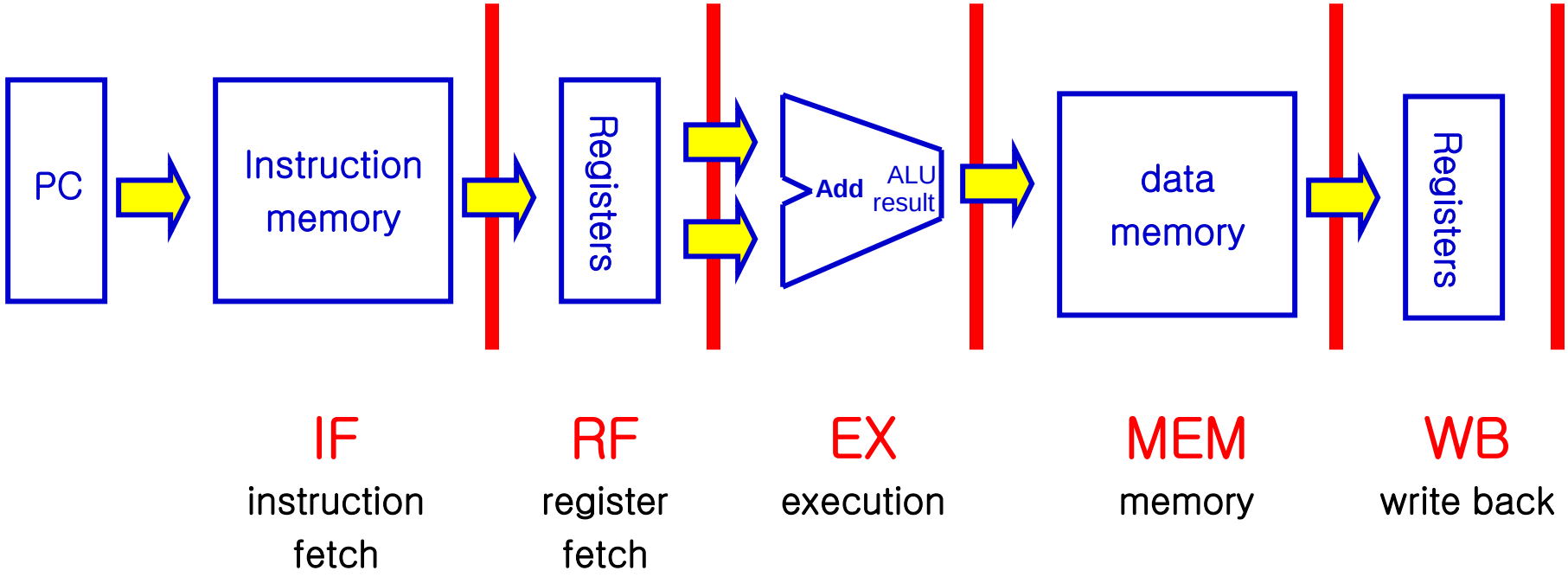
Instruction Class	Instruction Fetch	Register Fetch	Execute (ALU Op)	Memory Access	Register Write	Total time
R-format	2 ns	1 ns	2 ns		1 ns	6 ns
Load word	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
Store word	2 ns	1 ns	2 ns	2 ns		7 ns
Branch	2 ns	1 ns	2 ns			5 ns

간단한 비교 (Cont'd)

- **Single-Cycle**
  - 가장 Clock Cycle Time이 긴 *load* 명령어에 전체 Clock Rate가 맞춰지게 됨 -> 8 ns Clock
- **Multi-Cycle**
  - 가장 Clock Cycle Time이 긴 Step (또는 Stage)에 Clock Rate가 맞춰지게 됨 -> 2 ns Clock
  - 명령어 종류에 따라 Cycle 수가 다음과 같이 달라짐
    - R-format: 4 Cycles
    - Load: 5 Cycles
    - Store: 4 Cycles
    - Branch: 3 Cycles
- **Pipelined Processor**
  - 복잡한 Control로 인한 Delay 증가를 제외하면, Multi-Cycle 대비 4배 가량 성능을 향상시킬 수 있음

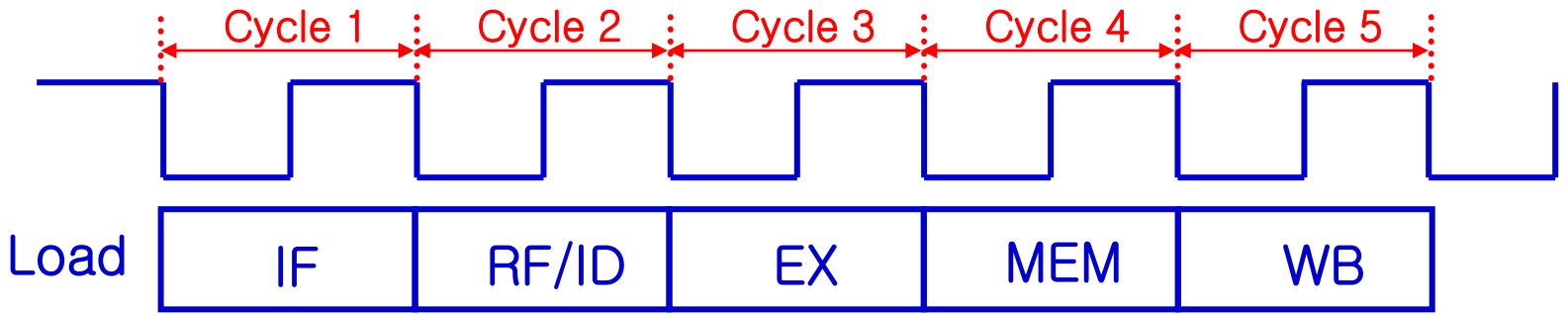
Pipeline Divisions

- (Multi-Cycle 처럼) Datapath를 각각 1 Cycle의 Stage로 나눔
- MIPS Pipeline에서 명령어들은 3-5 Stage로 실행됨



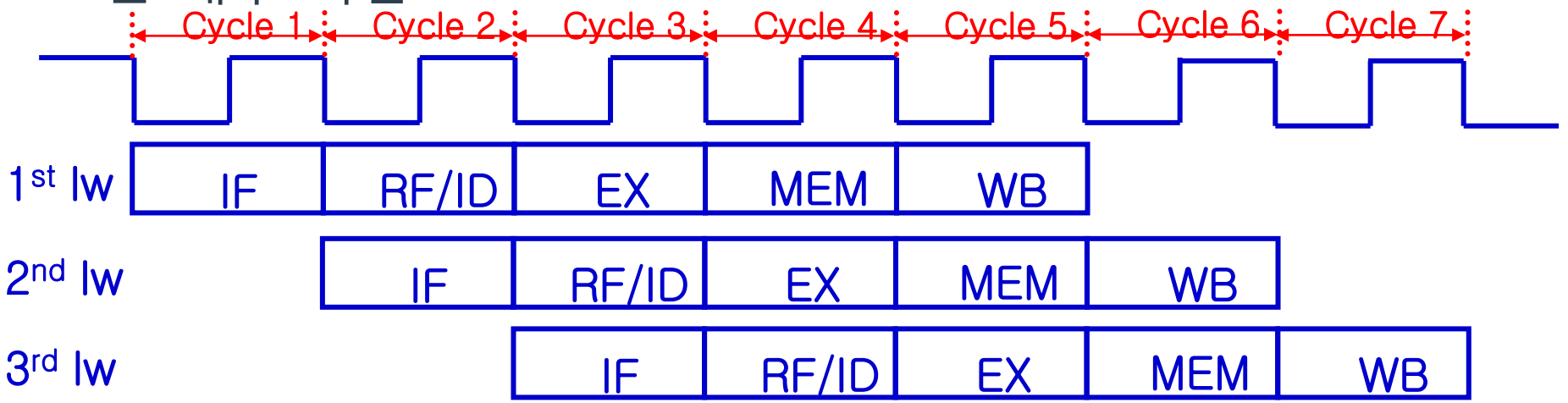
Load 명령어 5 Stages

- **IF: Instruction Fetch**
  - 명령어를 Memory에서 불러옴
  - PC를 증가시킴
- **RF/ID: Register Fetch and Instruction Decode**
  - Register에서 Base Address값을 불러옴
- **EX: Execute**
  - Base Address에 Sign-extended Offset을 더함
- **MEM: Memory**
  - Data Memory에서 Data를 불러옴
- **WB: Write Back**
  - 불러온 Data를 Register File에 저장함



Pipelining Load

- **Load 명령어는 5 Stage 모두 사용함**
  - 다섯 개의 독립적인 Functional Unit이 각 Stage에서 사용됨
    - 각각의 Functional Unit은 오직 한번만 수행됨
  - 또 다른 Load 명령어가 첫 번째 Load 명령어의 IF Stage가 끝나자마자 실행될 수 있음
  - 각각의 Load는 끝날 때 까지 5 Cycle이 걸림
  - Throughput (= 처리량 = 단위 시간 동안 실행된 명령어의 수)은 매우 커짐



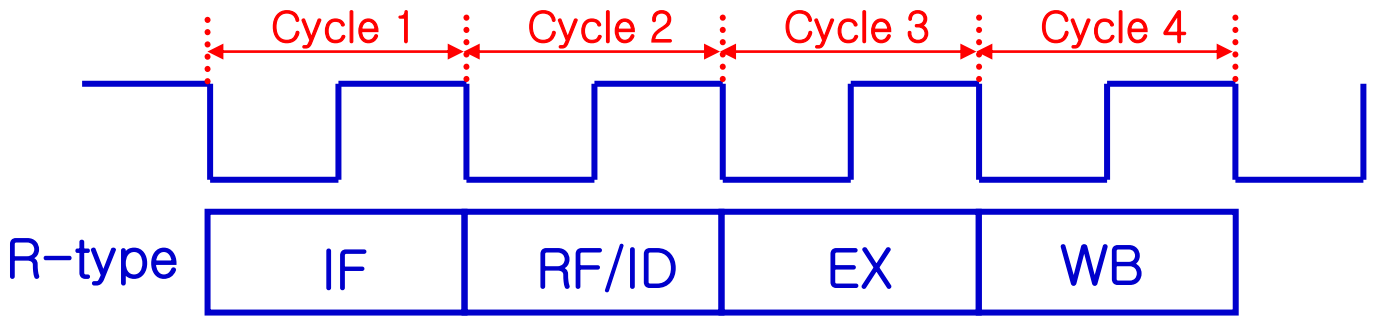
Pipelining Load (Cont'd)

- **Functional Unit들은 독립적으로 사용되어야 함!**
  - IF Stage를 위한 명령어 Memory, PC + 4 계산
  - RF/ID Stage에 Register File을 읽음
  - EX Stage에 ALU가 사용됨
  - Mem Stage에 Data Memory가 사용됨
  - WB Stage에 Register

- **매 Cycle에 명령어가 하나씩 Pipeline에 들어감**
  - 매 Cycle에 명령어가 하나씩 끝남
  - 이상적인 CPI는 결국 1이 됨

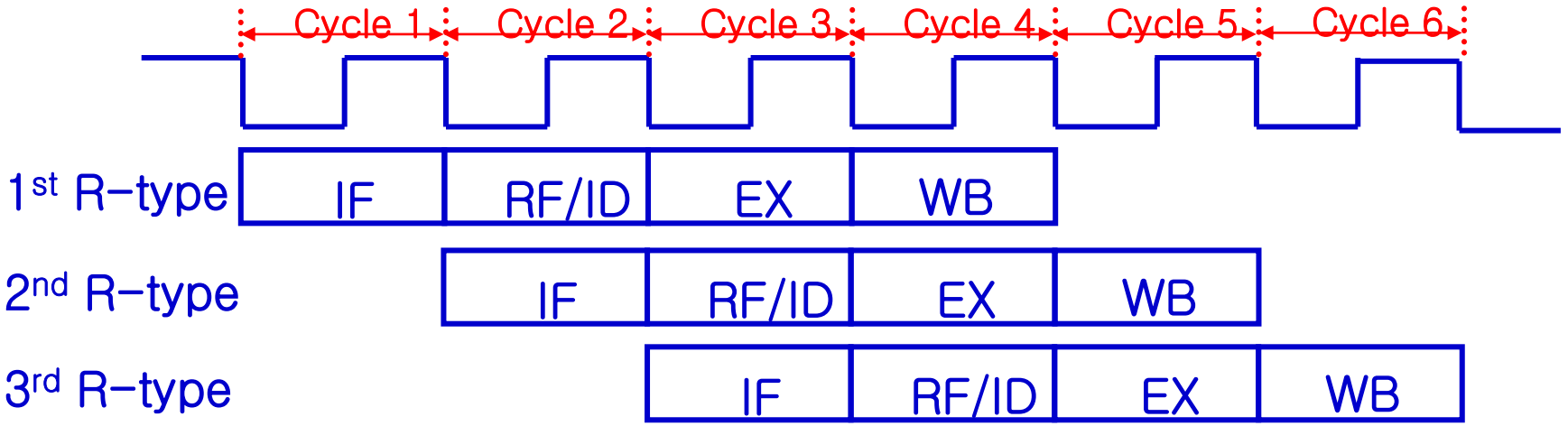
R-Type 명령어 4 Stage

- **IF: Instruction Fetch**
  - 명령어를 Memory에서 불러옴
  - PC를 증가시킴 (다음 명령어가 바로 다음 Cycle에 실행될 수 있도록)
- **RF/ID: Register Fetch and Instruction Decode**
  - Register에서 계산할 값들을 불러옴
- **EX: Execute**
  - Register에서 불러온 값들에 대한 연산을 수행함
- **WB: Write Back**
  - 연산의 결과물을 Register File에 저장함

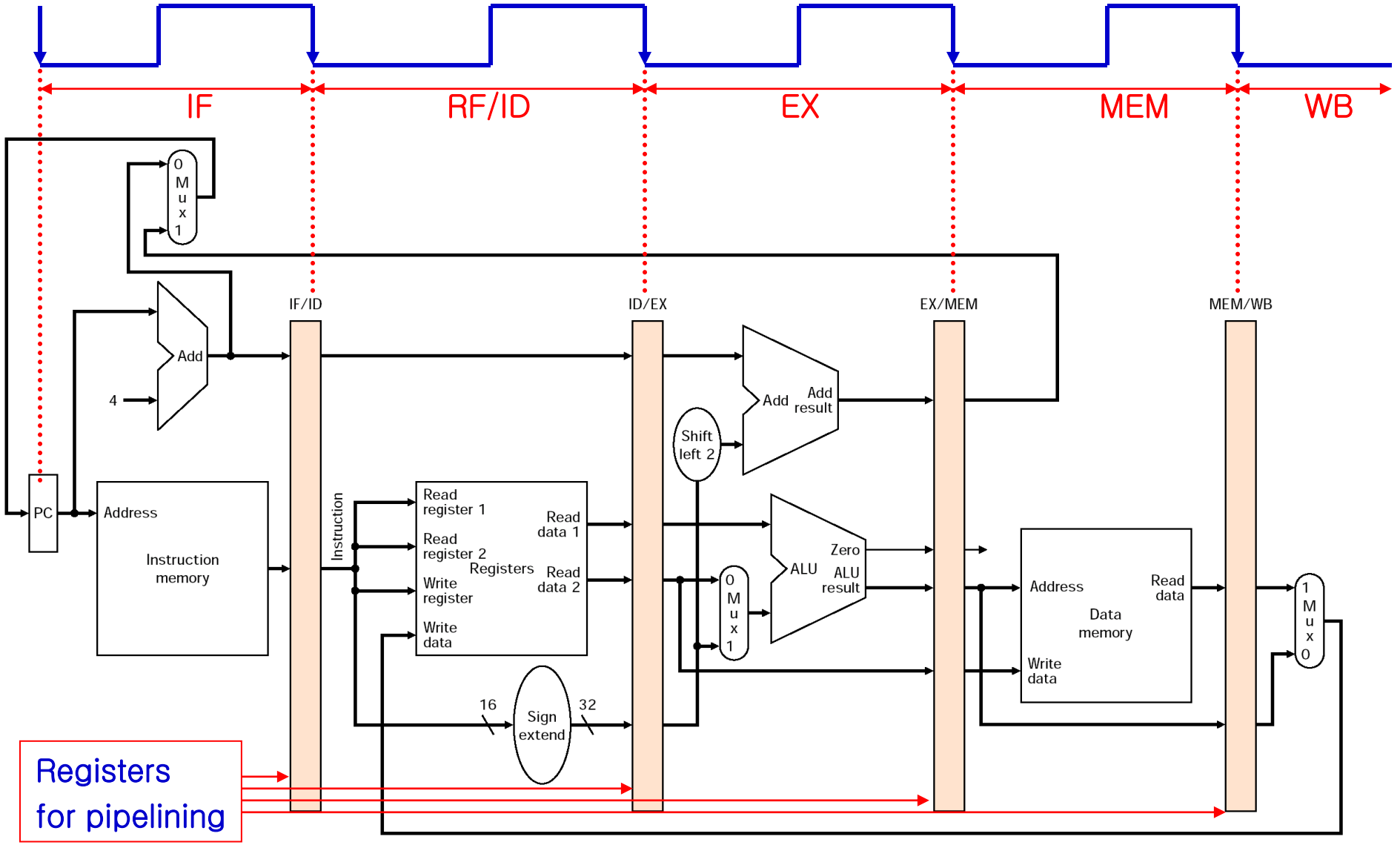


Pipelining R-Type

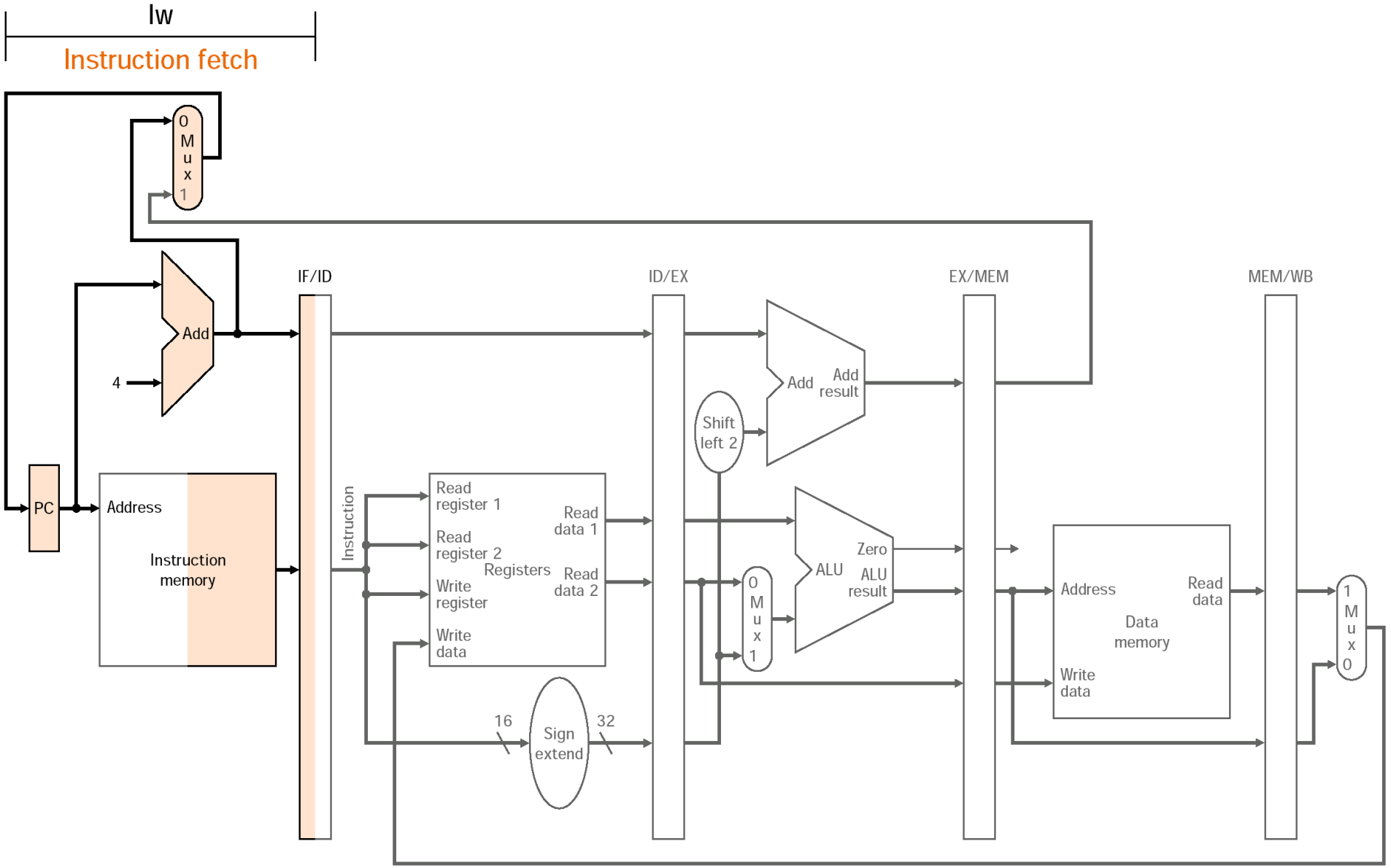
- **R-Type 명령어는 4 Stage를 사용함**
  - 네개의 독립적인 Functional Unit이 각 Stage에서 사용됨
    - 각각의 Functional Unit은 오직 한번만 수행됨
  - 또 다른 R-Type 명령어는 첫 번째 명령어의 IF Stage가 끝나자마자 실행될 수 있음
  - 각각의 R-Type 명령어는 끝날 때 까지 4 Cycle이 걸림
  - Throughput (= 처리량)은 Load와 마찬가지로 매우 커짐



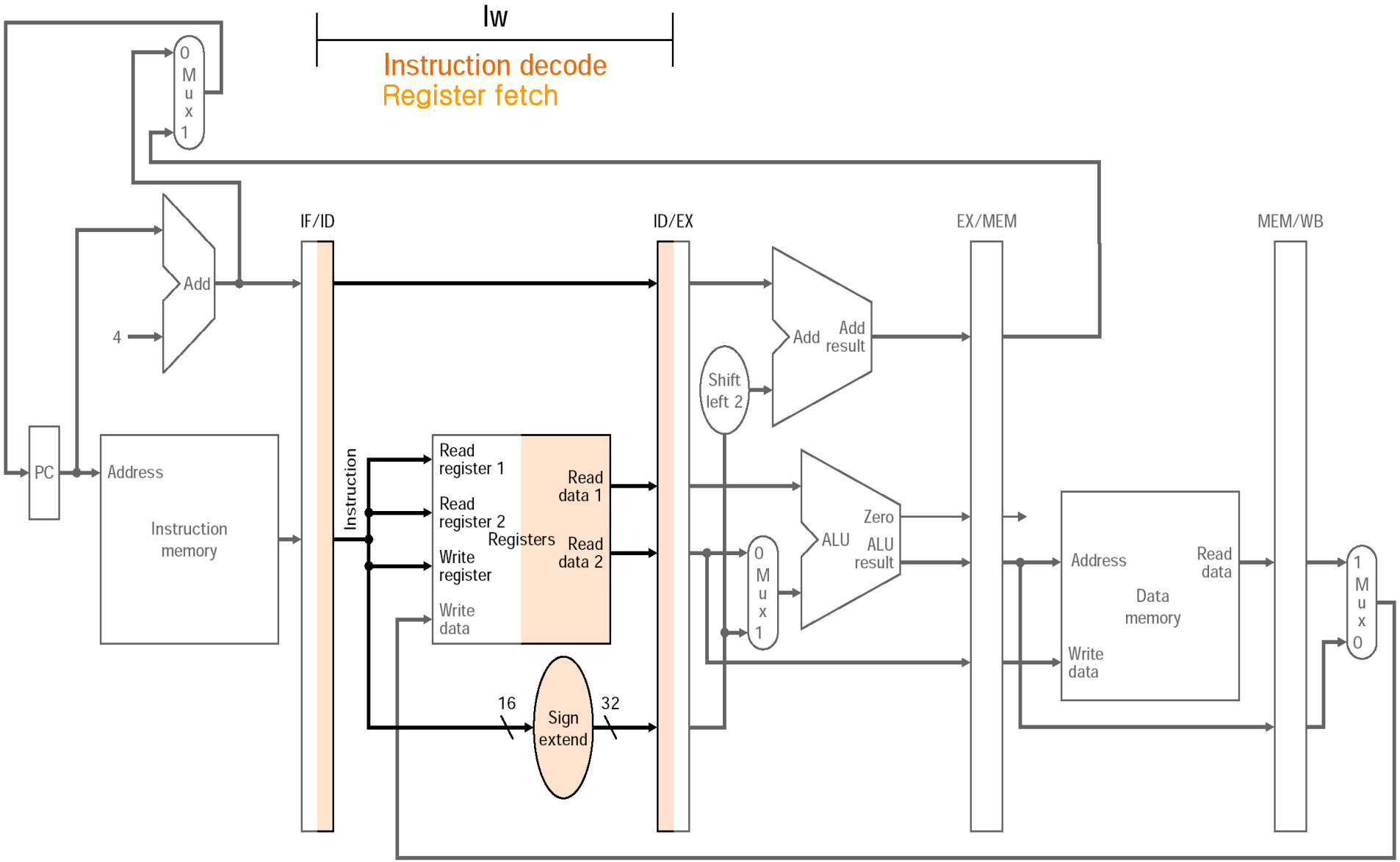
Pipeline Datapath



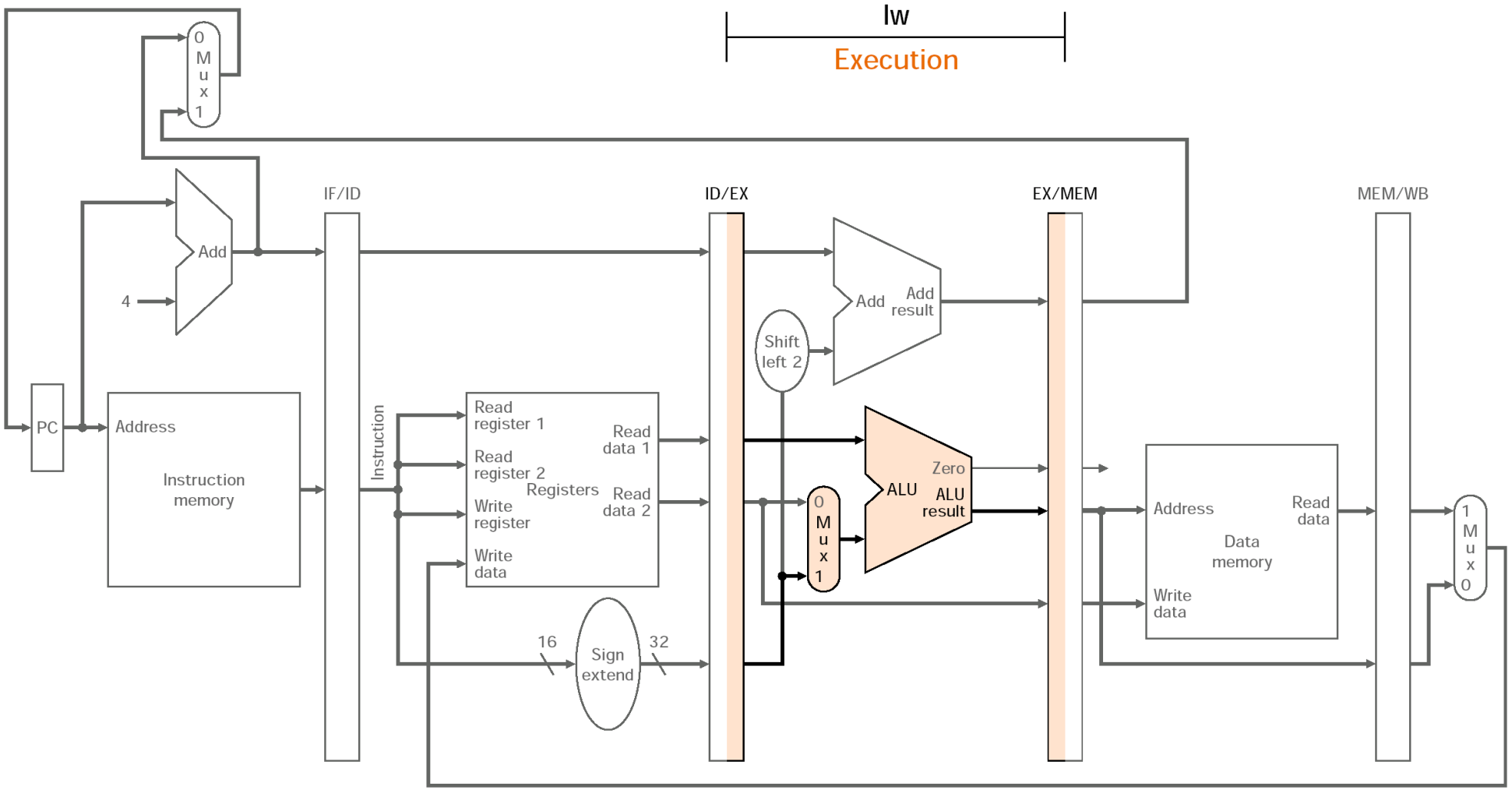
Load Datapath: Stage 1



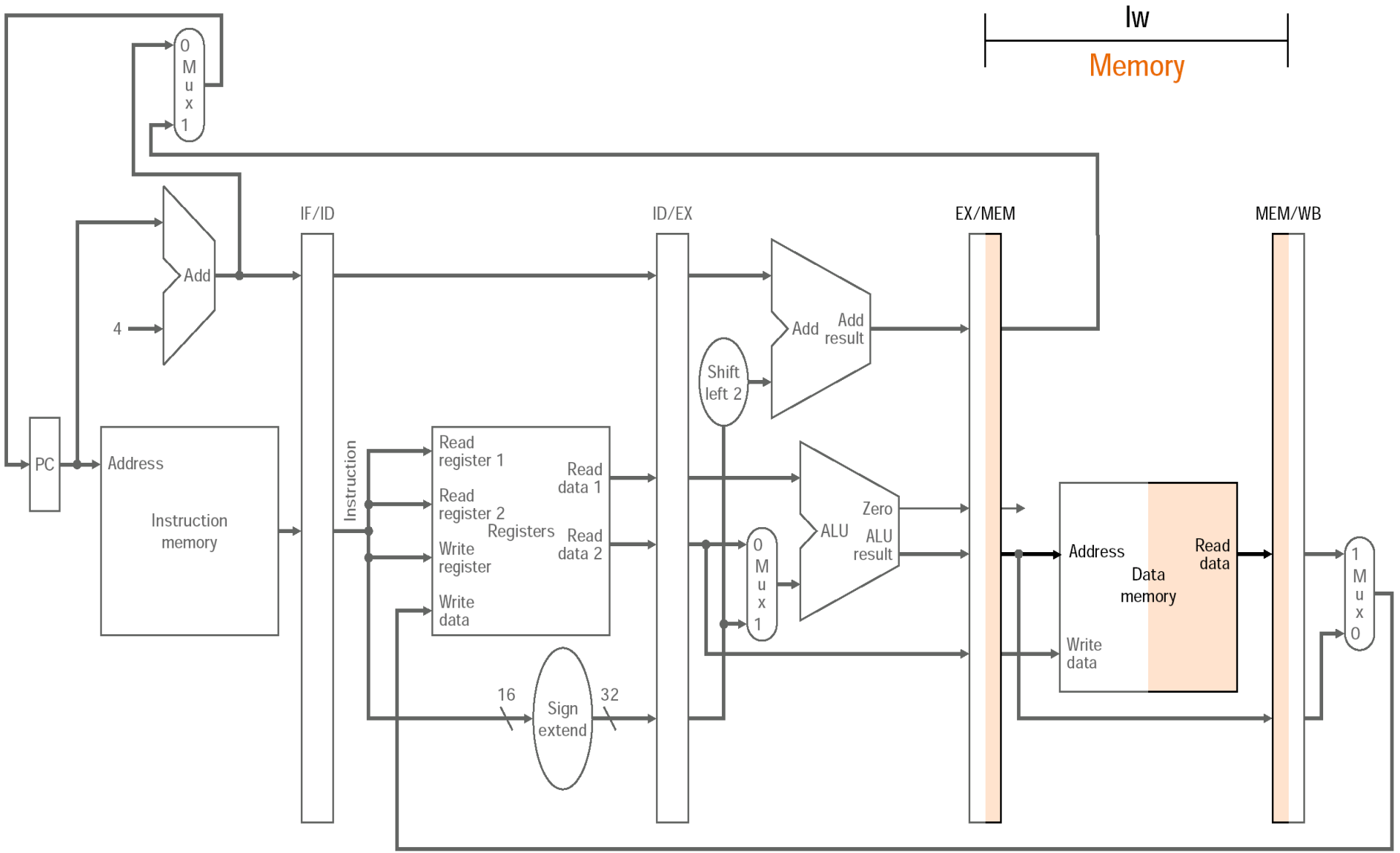
Load Datapath: Stage 2



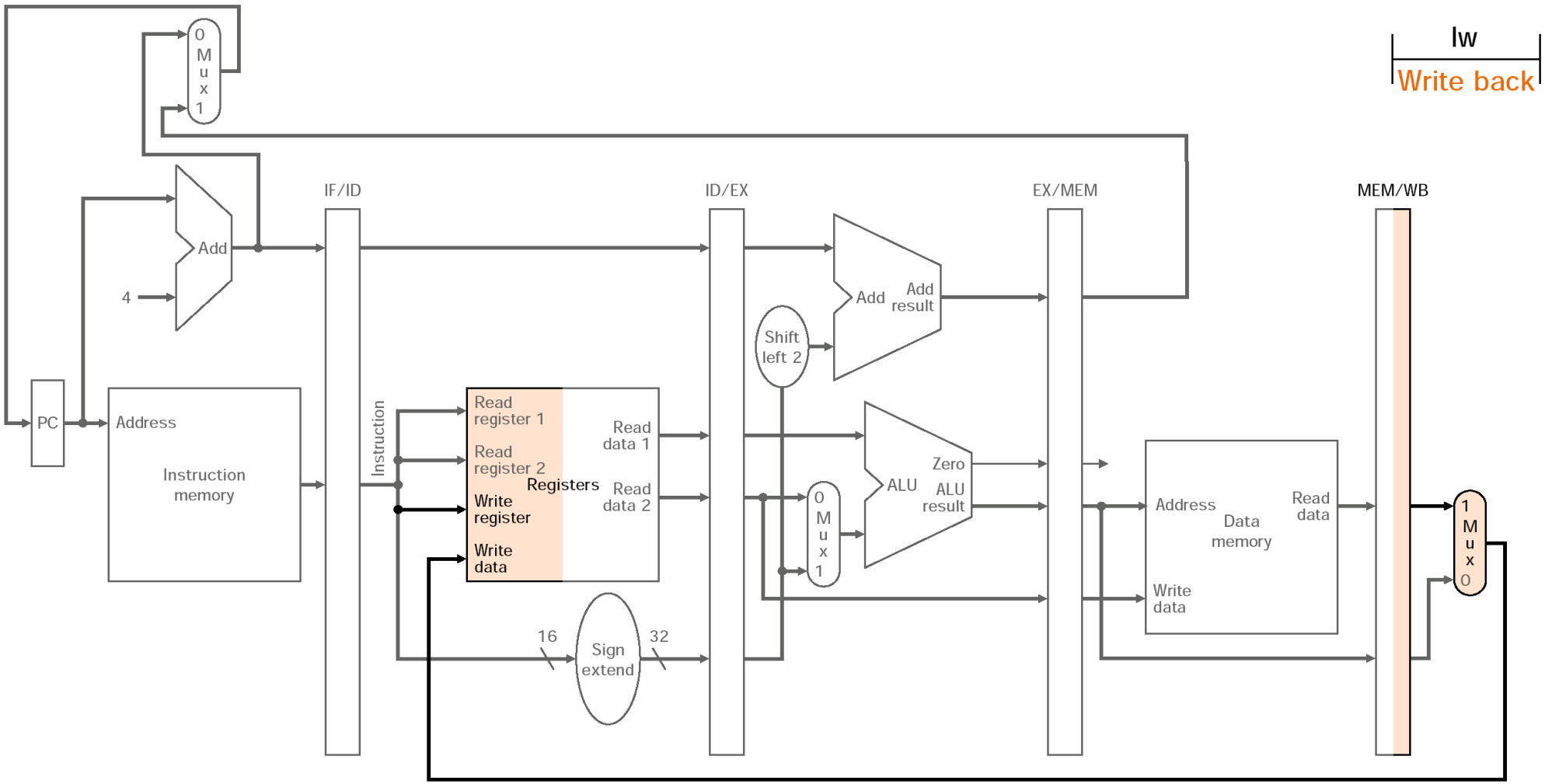
Load Datapath: Stage 3



Load Datapath: Stage 4

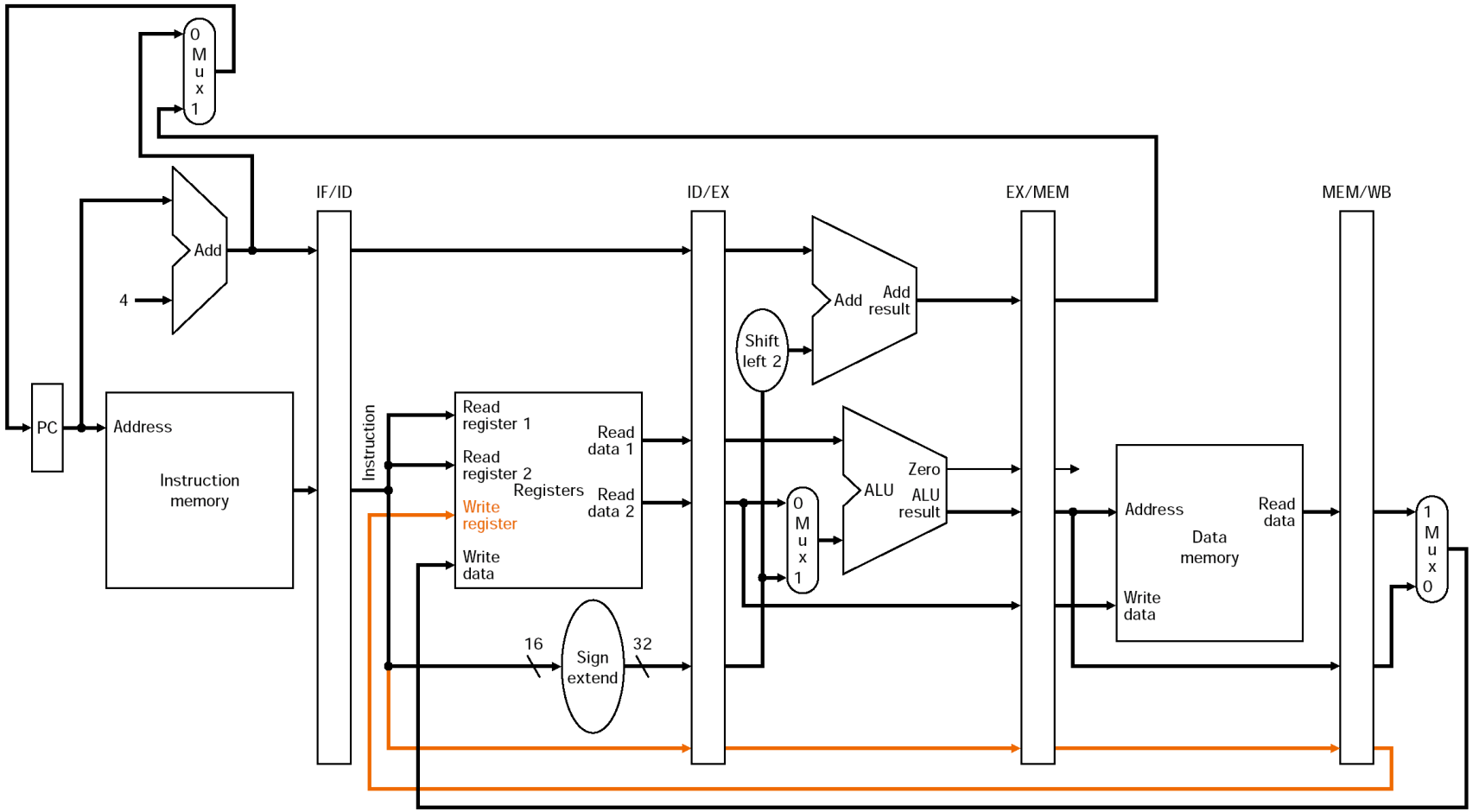


Load Datapath: Stage 5



수정된 Pipelined Datapath

- **Write Register의 번호를 유지하기 위해**
  - Load 명령어가 WB Stage에 있을 때, IF/ID Stage에는 다른 명령어가 실행되고 있으므로, Write Register번호를 유지하기 위해서는..

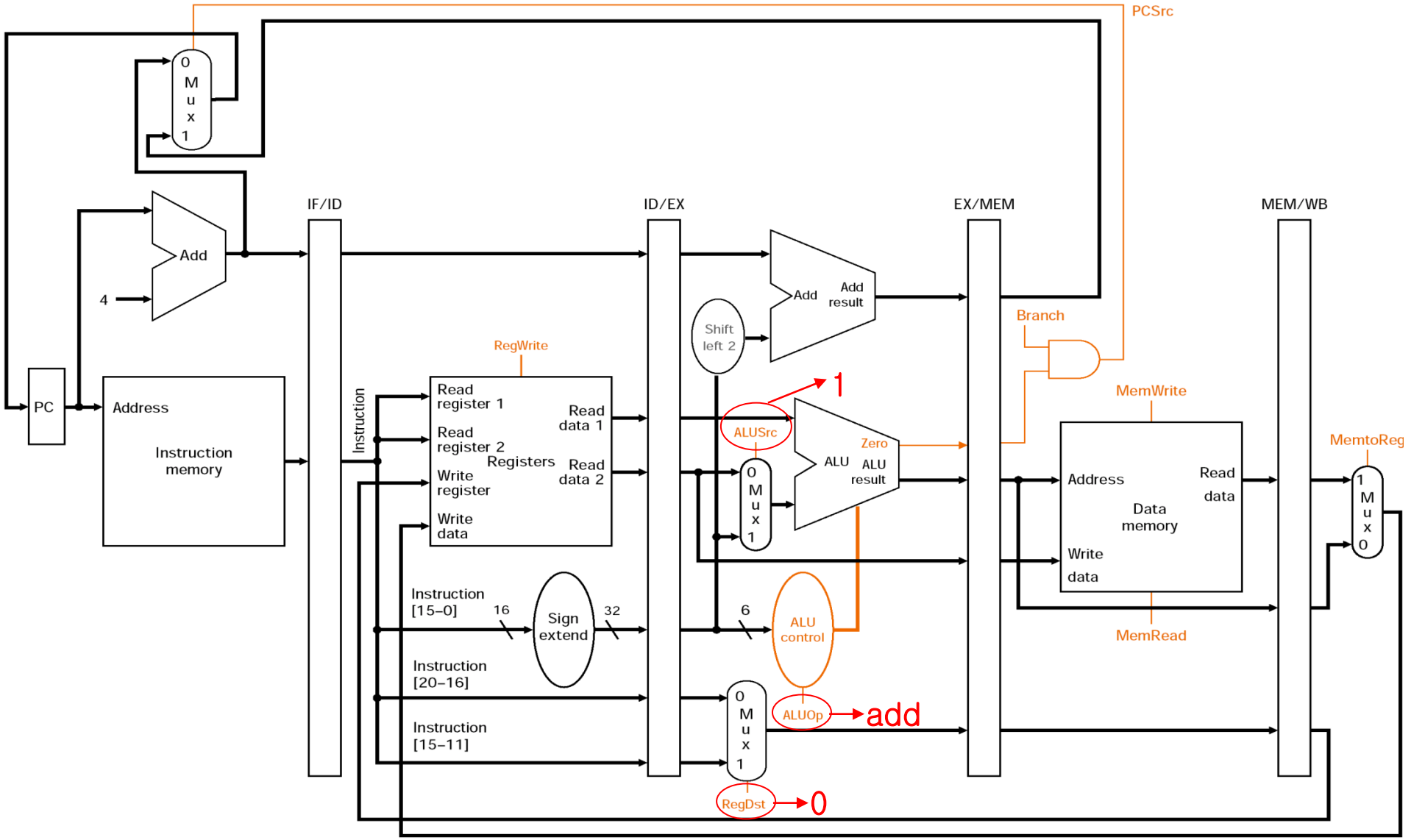


Pipeline Control

- Single-Cycle 및 Multi-Cycle 구현처럼, Pipeline을 위해서도 Control 신호를 추가해야함

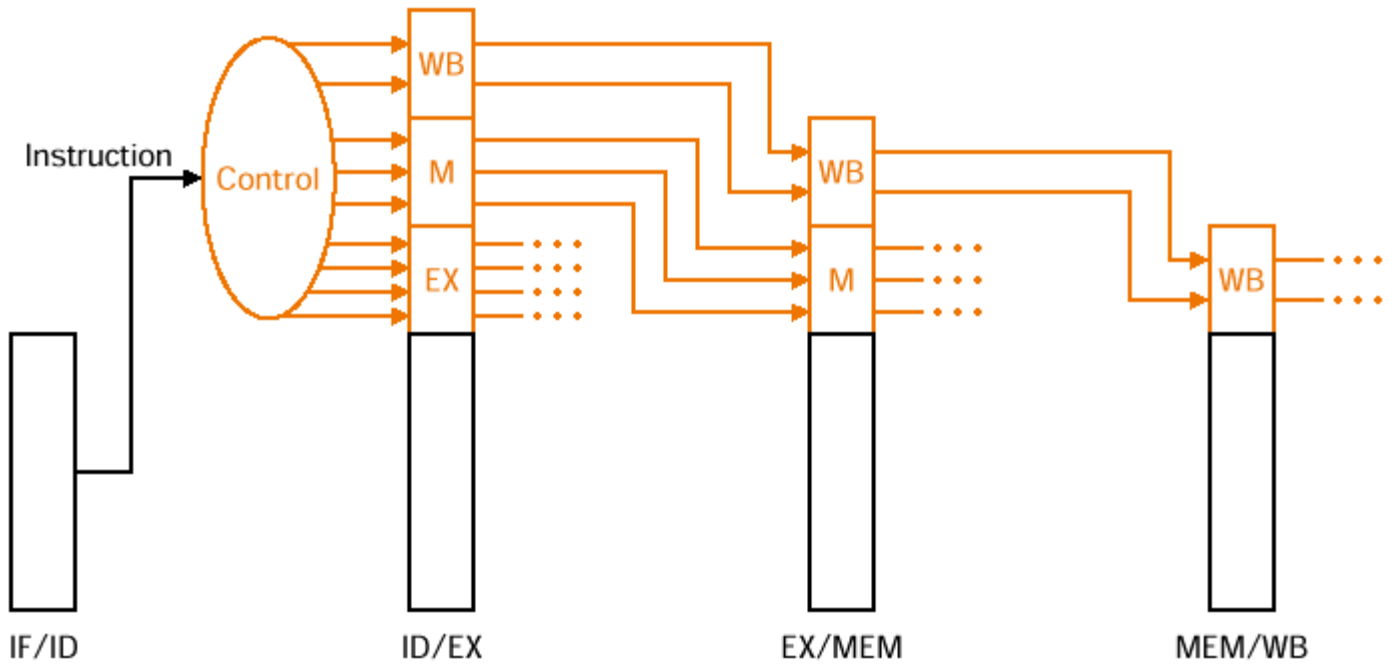
- 단, Single-Cycle이나 Multi-Cycle과는 다르게, 하나의 명령어가 다른 명령어의 Control에 영향을 미치지 않아야 함!

Load Control at Execute



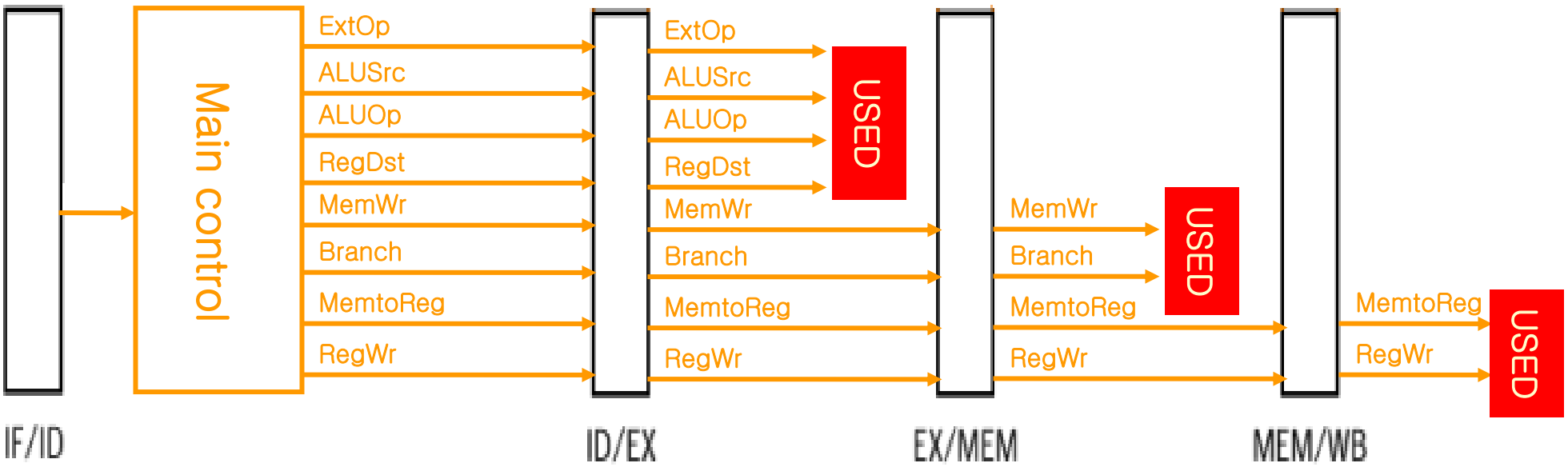
Control Signals

- **RF/ID Stage에서 하나의 Main Control Unit이 해당 명령어를 위해 필요한 Control Signal을 모두 생성함**
  - 1 Cycle 후에 EX Stage에서 쓰일 Control 신호 (ExtOp, ALUSrc, …)
  - 2 Cycles 후에 MEM Stage에서 쓰일 Control 신호 (MemR, Branch, ..)
  - 3 Cycles 후에 WB Stage에서 쓰일 Control 신호 (MemtoReg, MemWr, ..)

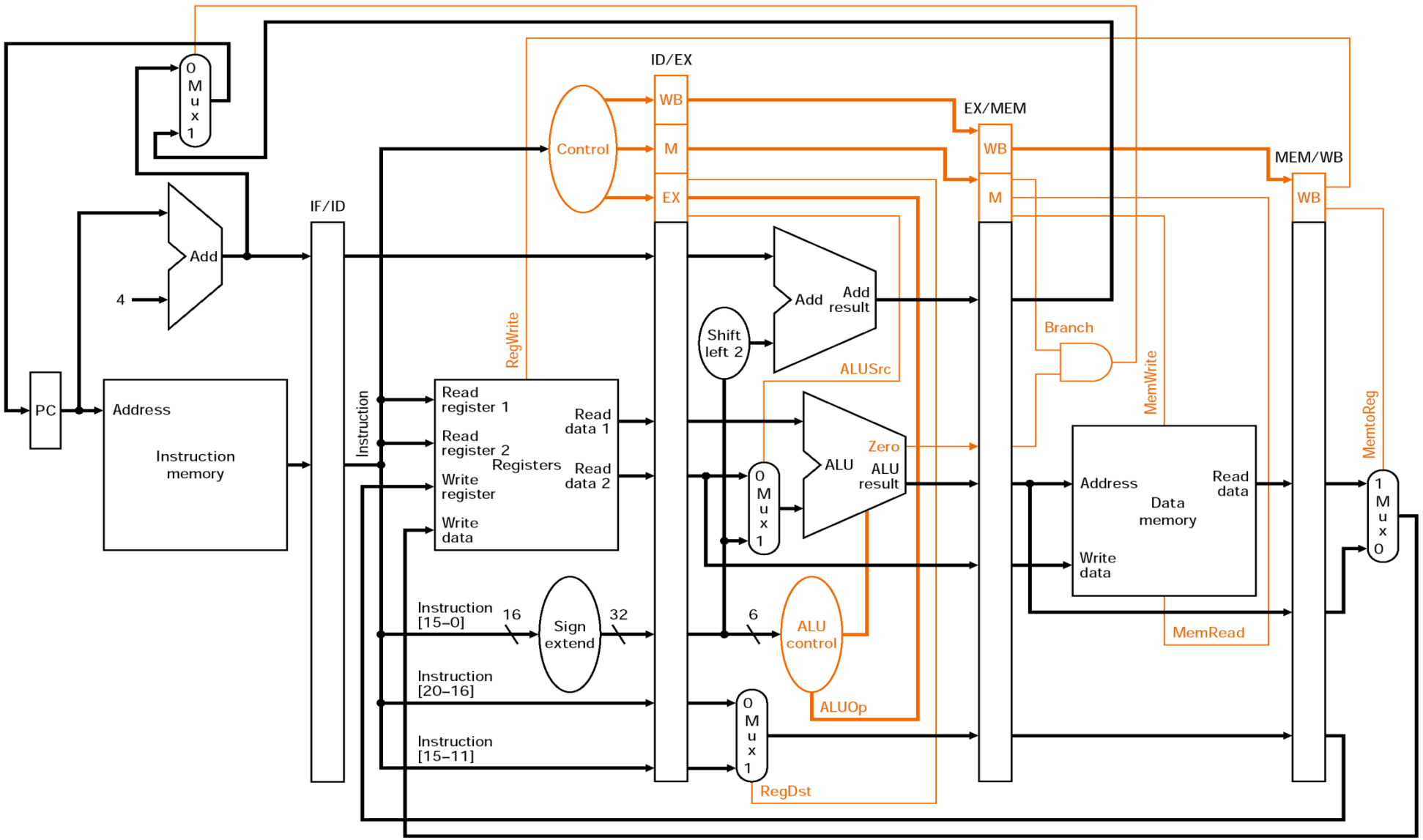


Control Signals (Cont'd)

- 각 Stage에서 사용되는 Control Signal들

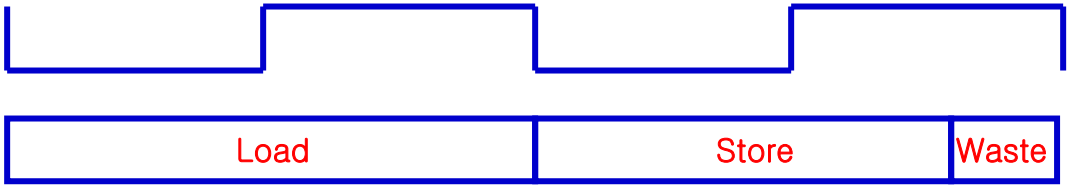


Pipelined Datapath with Controls

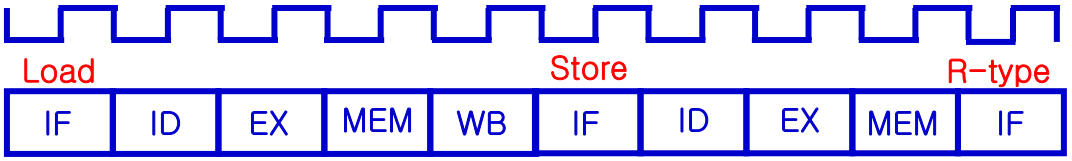


구현 비교

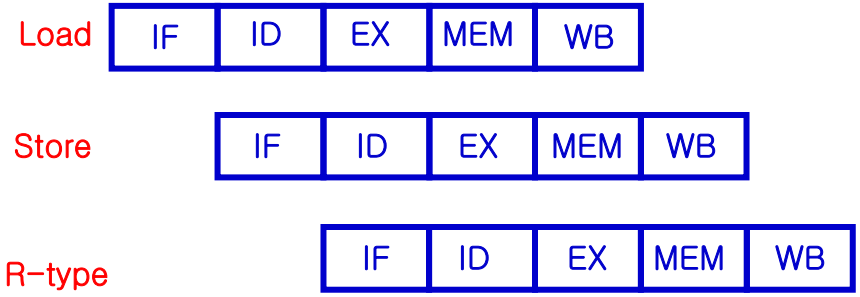
- **Single-Cycle Implementation**



- **Multi-Cycle Implementation**



- **Pipeline Implementation**



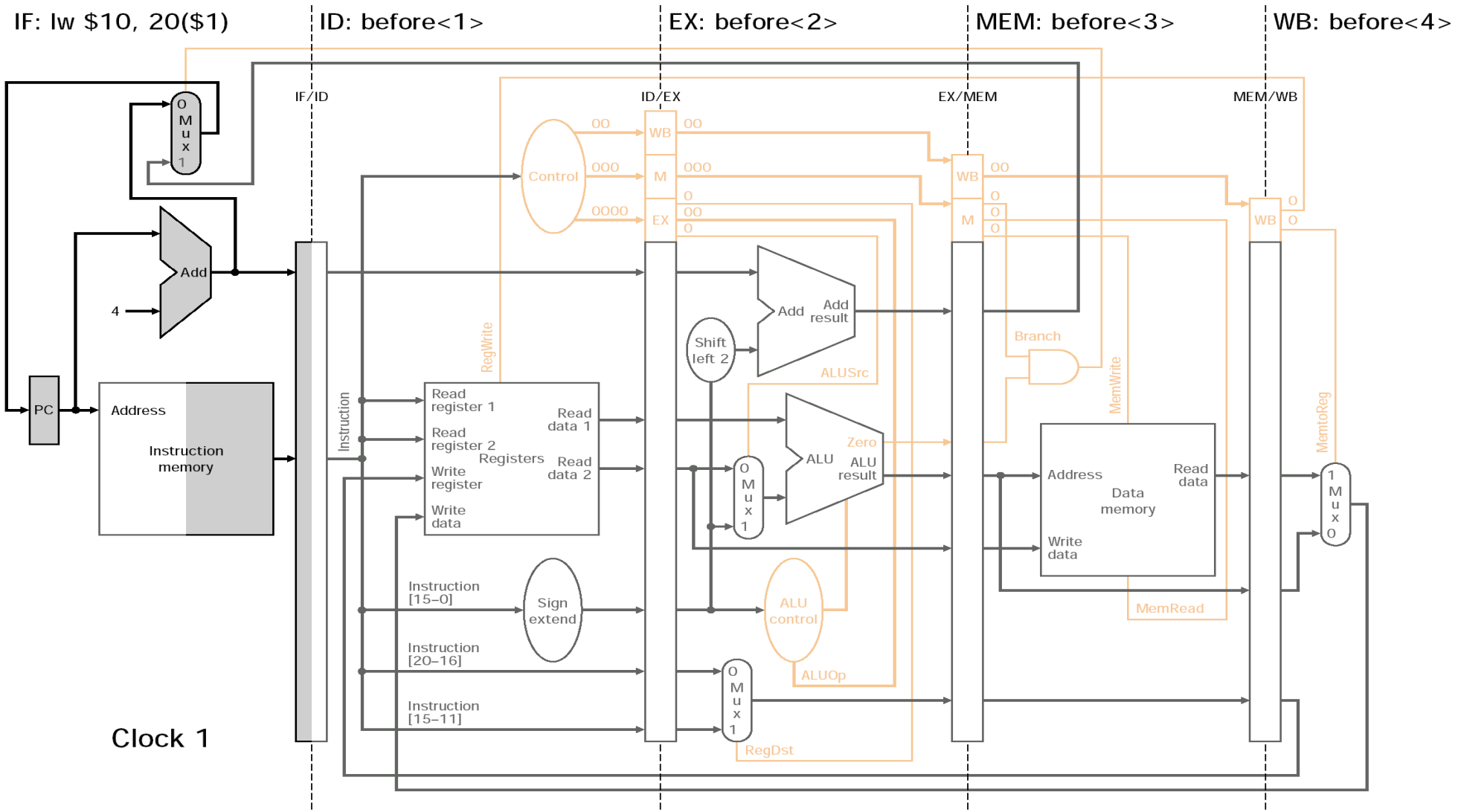
Example

- **다음과 같은 명령어들을 실행한다고 가정**
  - *lw $10, 20 ($1)*
  - *sub $11, $2, $3*
  - *and $12, $4, $5*
  - *or $13, $6, $7*
  - *add $14, $8, $9*

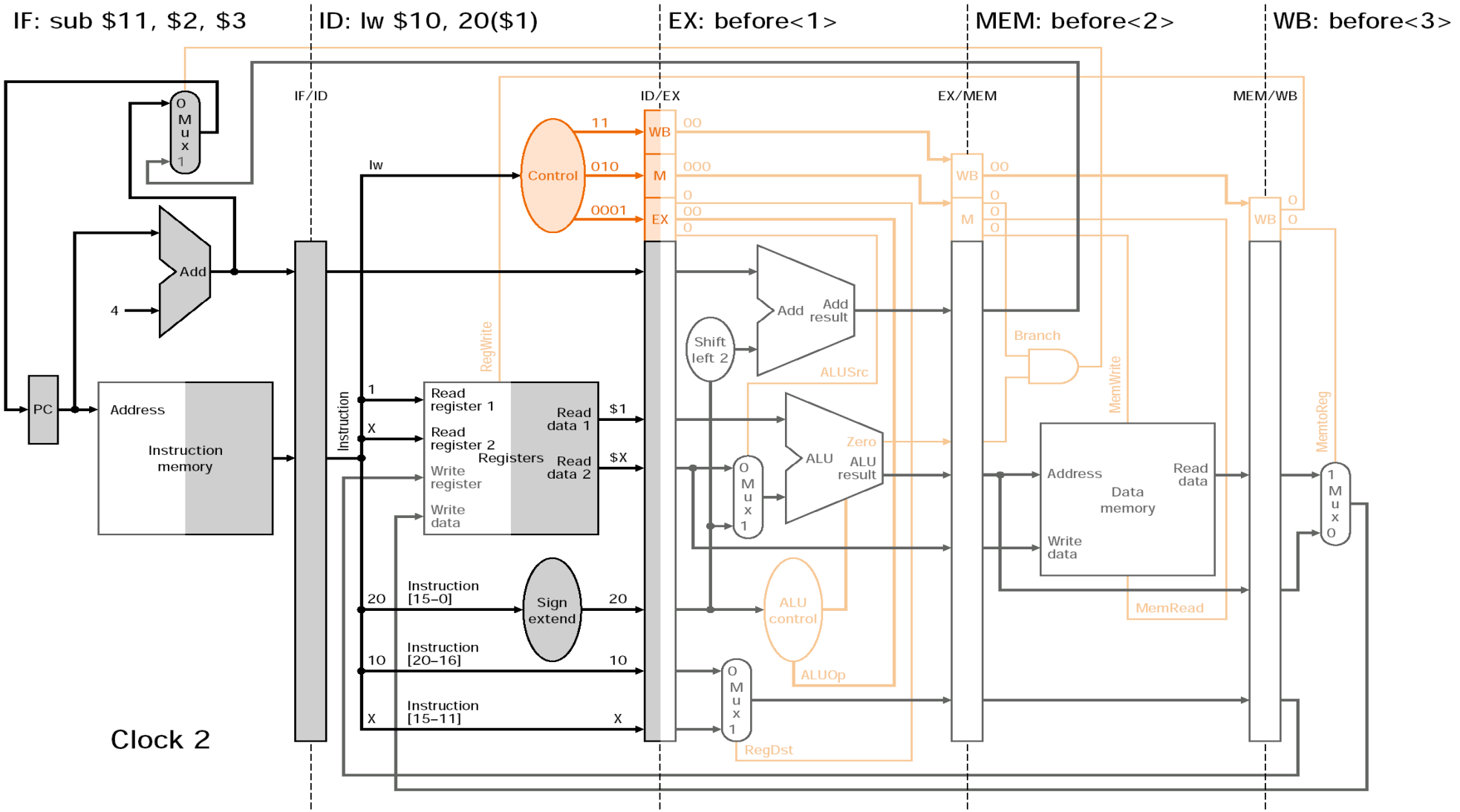
- **가정**
  - Hazard는 없음

- **Hazard**
  - 다음 명령어가 바로 실행될 수 없는 몇몇 경우가 있음
  - 곧 다룰 예정!

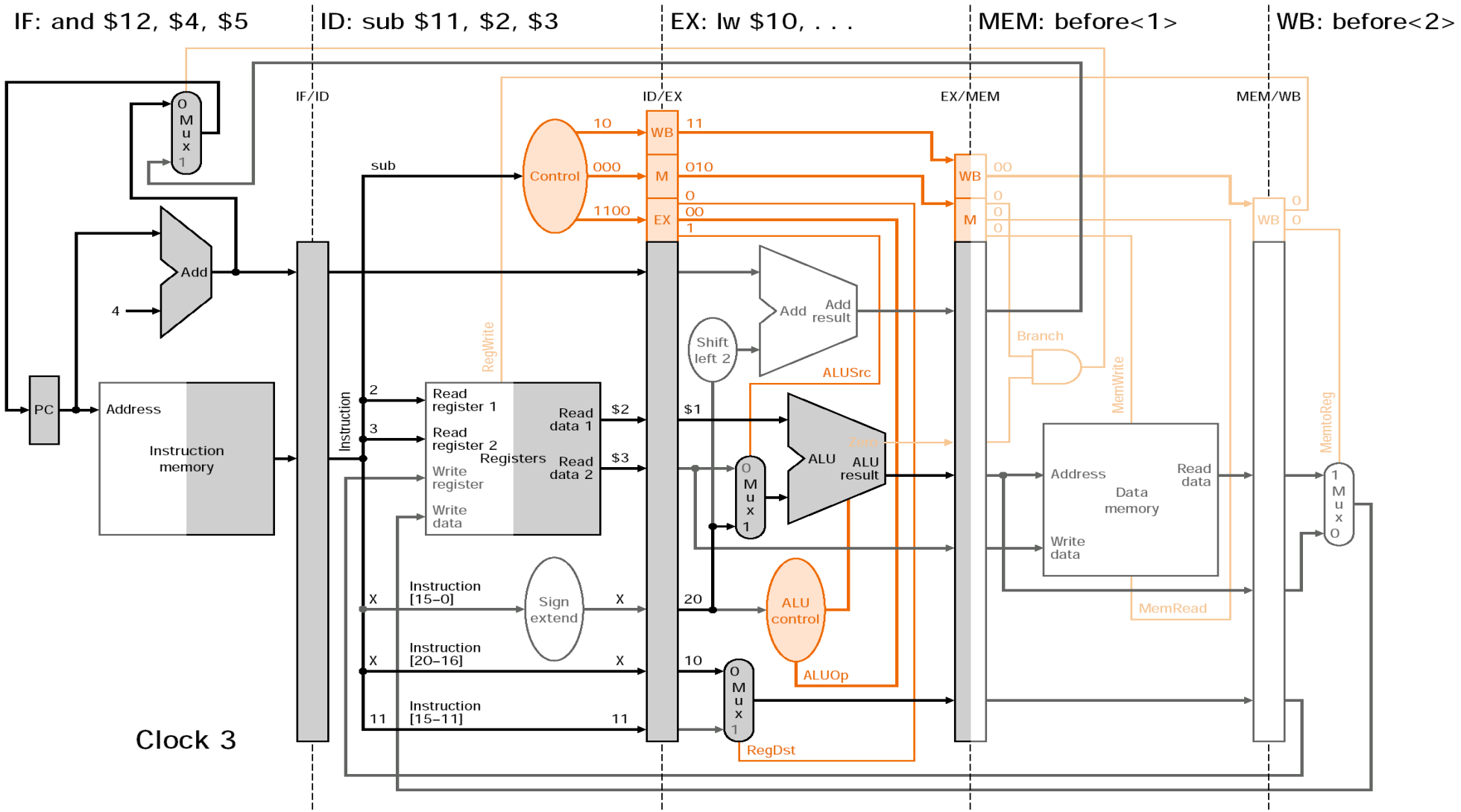
Example


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

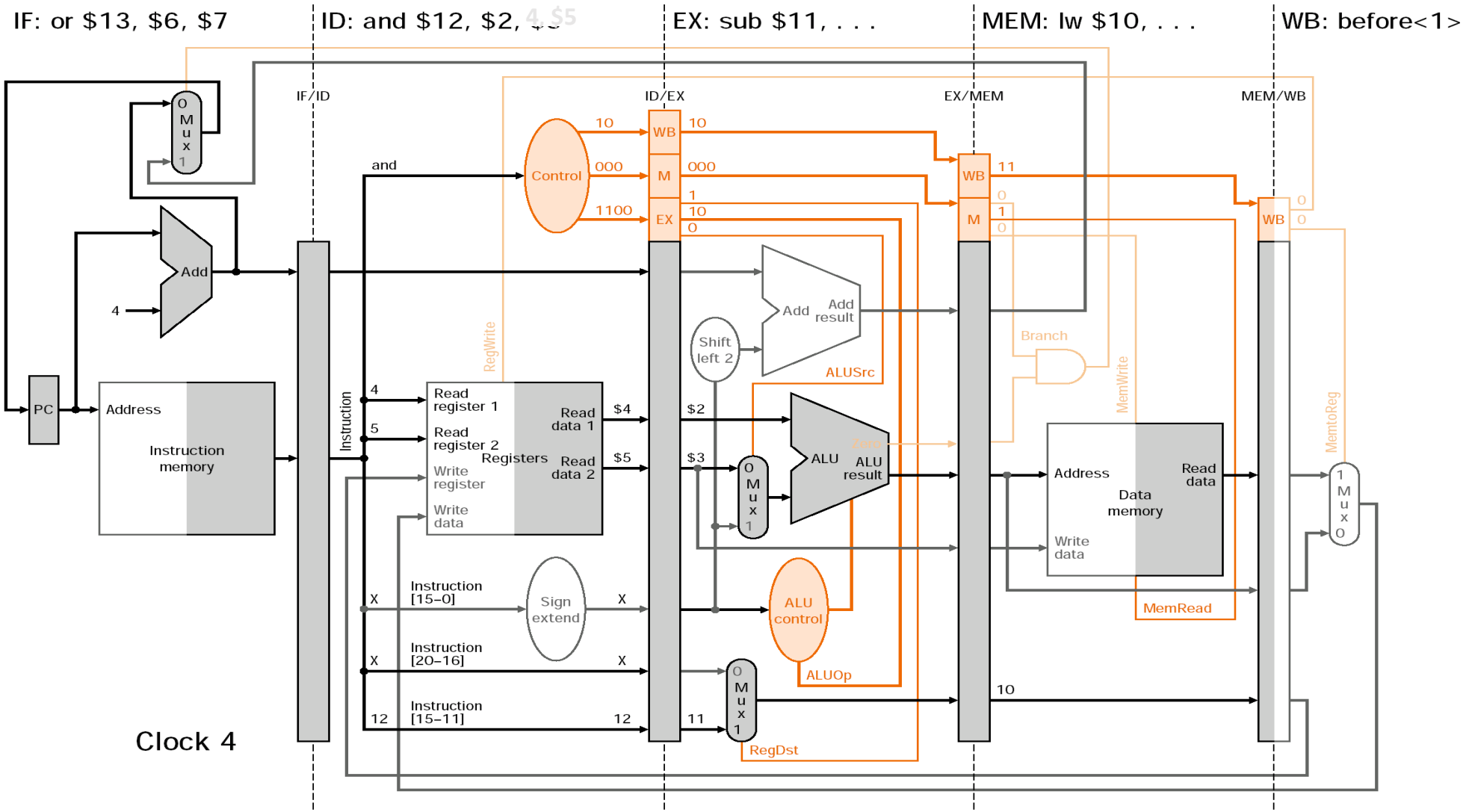
Example


IF: sub $11, $2, $3
ID: lw $10, 20($1)
EX: before<1>
MEM: before<2>
WB: before<3>
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
0
1
Add
4
PC
Address
Instruction memory
Instruction
lw
Control
11
010
0001
WB
M
EX
00
000
0
00
0
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
$1
$X
1
X
20
10
X
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
Sign extend
Shift left 2
Add
Add result
ALUSrc
ALU
Zero
ALU result
ALU control
ALUOp
RegDst
Branch
MemWrite
MemRead
Address
Data memory
Read data
Write data
MemtoReg
Clock 2

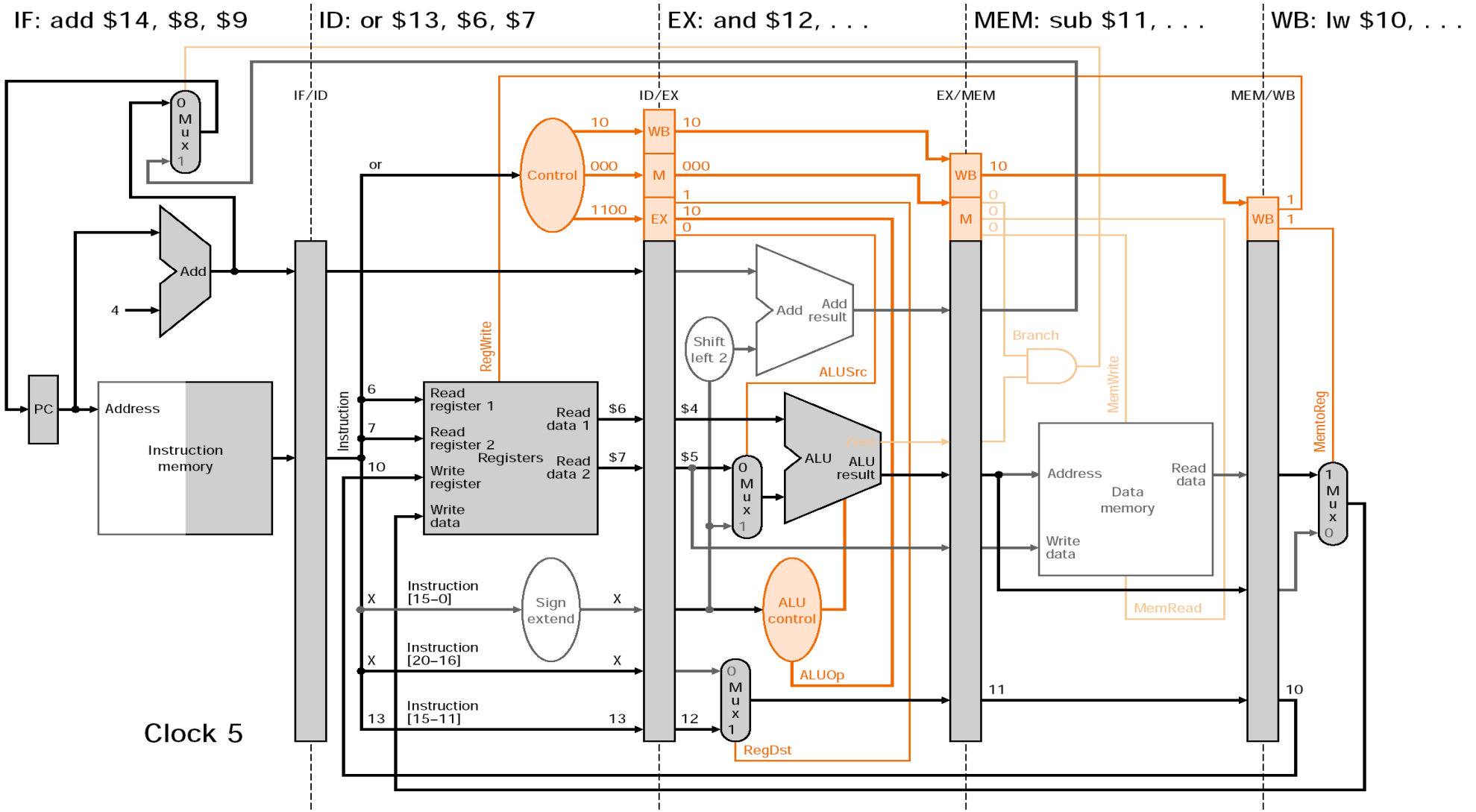
Example


IF: and $12, $4, $5
ID: sub $11, $2, $3
EX: lw $10, . . .
MEM: before<1>
WB: before<2>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 3
PC
Address
Instruction memory
Add
4
M u x
0
1
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
sub
Control
10
000
1100
WB
M
EX
11
010
00
1
Shift left 2
Add
Add result
ALUSrc
ALU
ALU result
Zero
Branch
MemWrite
MemRead
MemtoReg
Address
Data memory
Read data
Write data
ALU control
ALUOp
RegDst
Sign extend
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
2
3
$2
$3
$1
20
10
11
X
00
0

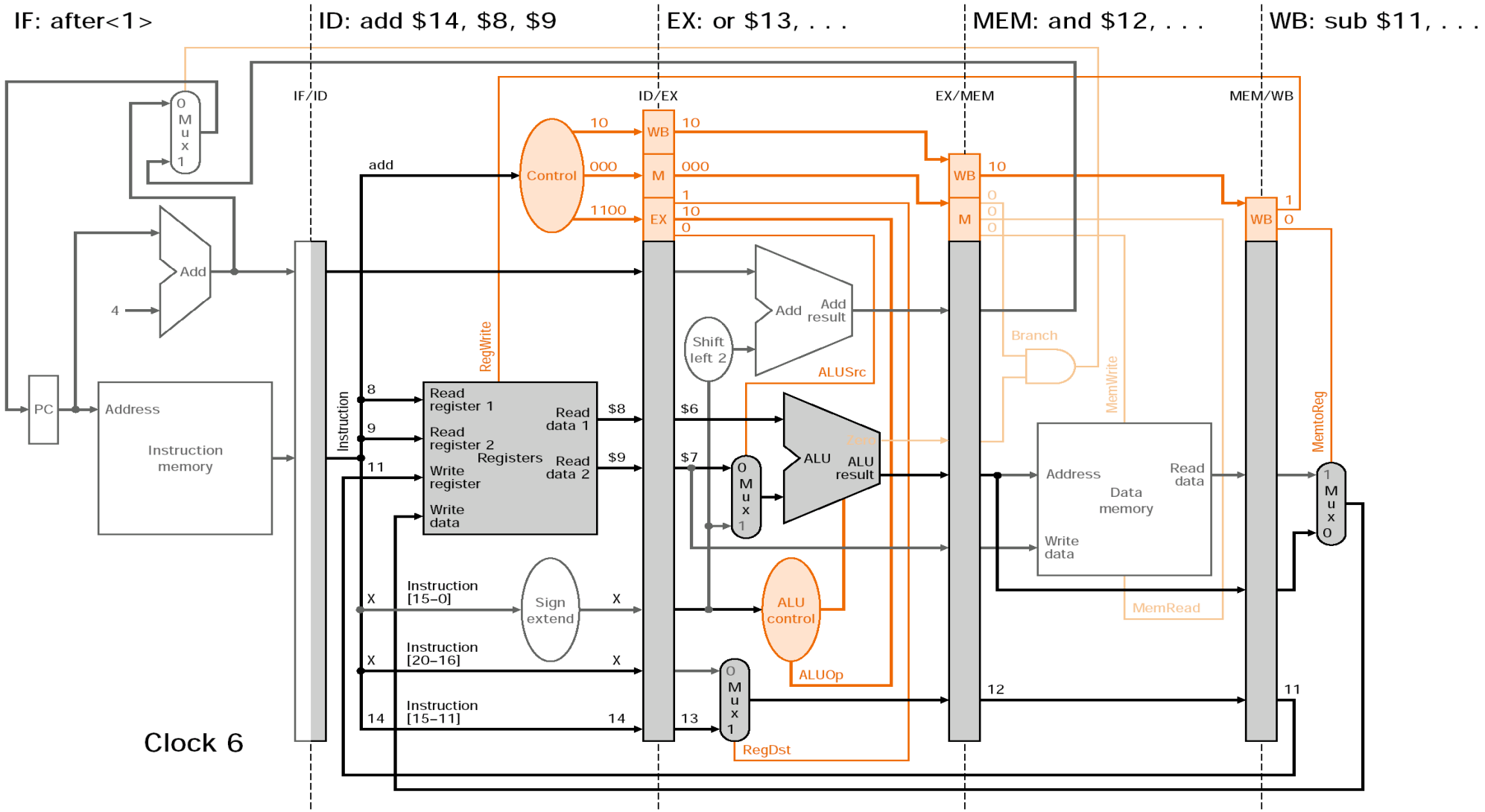
Example


IF: or $13, $6, $7
ID: and $12, $2, $5
EX: sub $11, . . .
MEM: lw $10, . . .
WB: before<1>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 4
PC
Address
Instruction memory
Add
4
M u x
Instruction
and
Control
10
000
1100
WB
M
EX
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
$4
$5
$2
$3
Shift left 2
Add result
ALUSrc
ALU
ALU result
Zero
ALU control
ALUOp
Sign extend
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
RegDst
Branch
MemWrite
Address
Data memory
Read data
Write data
MemRead
MemtoReg

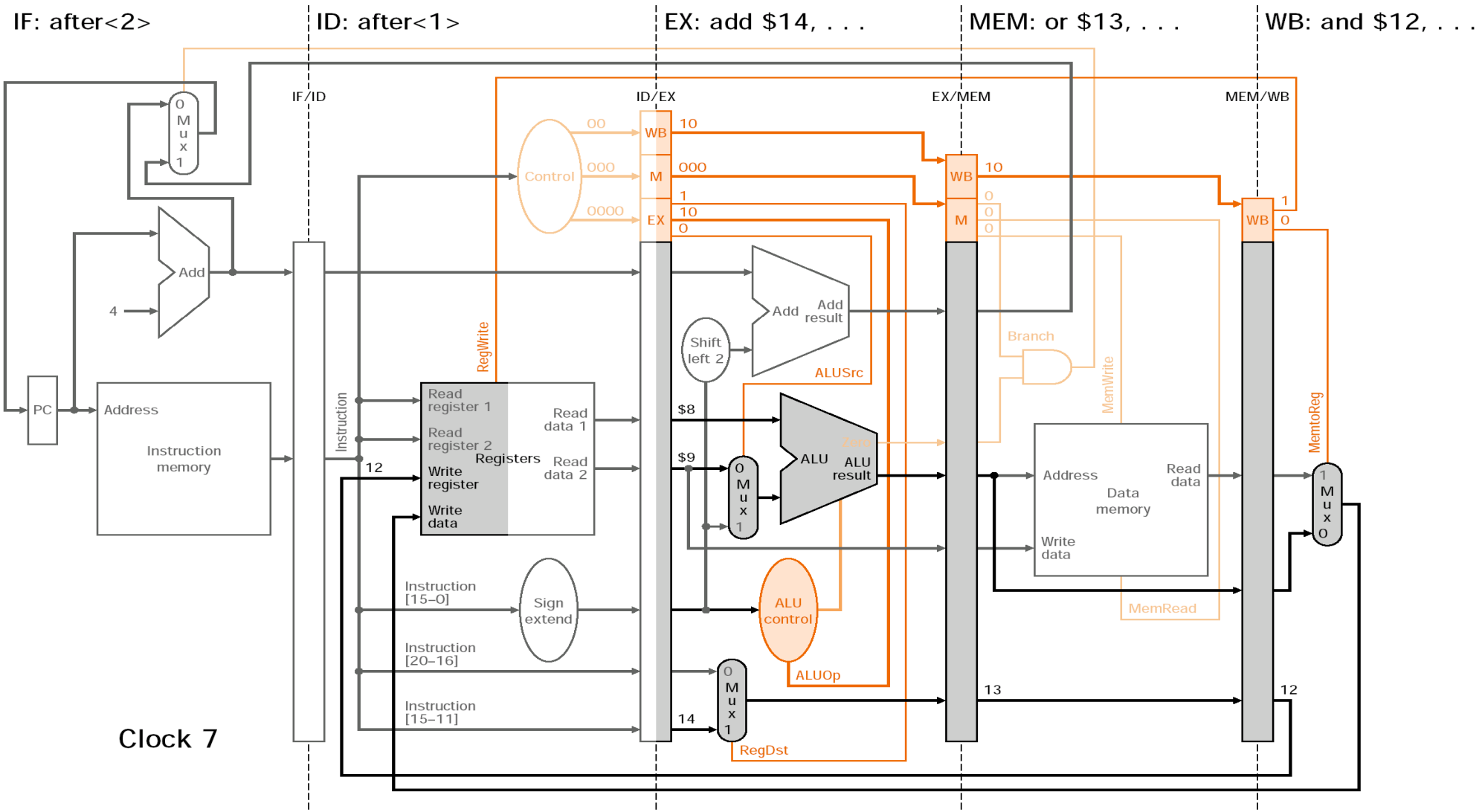
Example


IF: add $14, $8, $9
ID: or $13, $6, $7
EX: and $12, . . .
MEM: sub $11, . . .
WB: lw $10, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 5
PC
Address
Instruction memory
Add
4
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
Control
WB
M
EX
Shift left 2
Add result
ALUSrc
ALU
ALU result
Zero
Branch
MemWrite
MemRead
MemtoReg
Address
Data memory
Read data
Write data
ALU control
ALUOp
Sign extend
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
RegDst
or

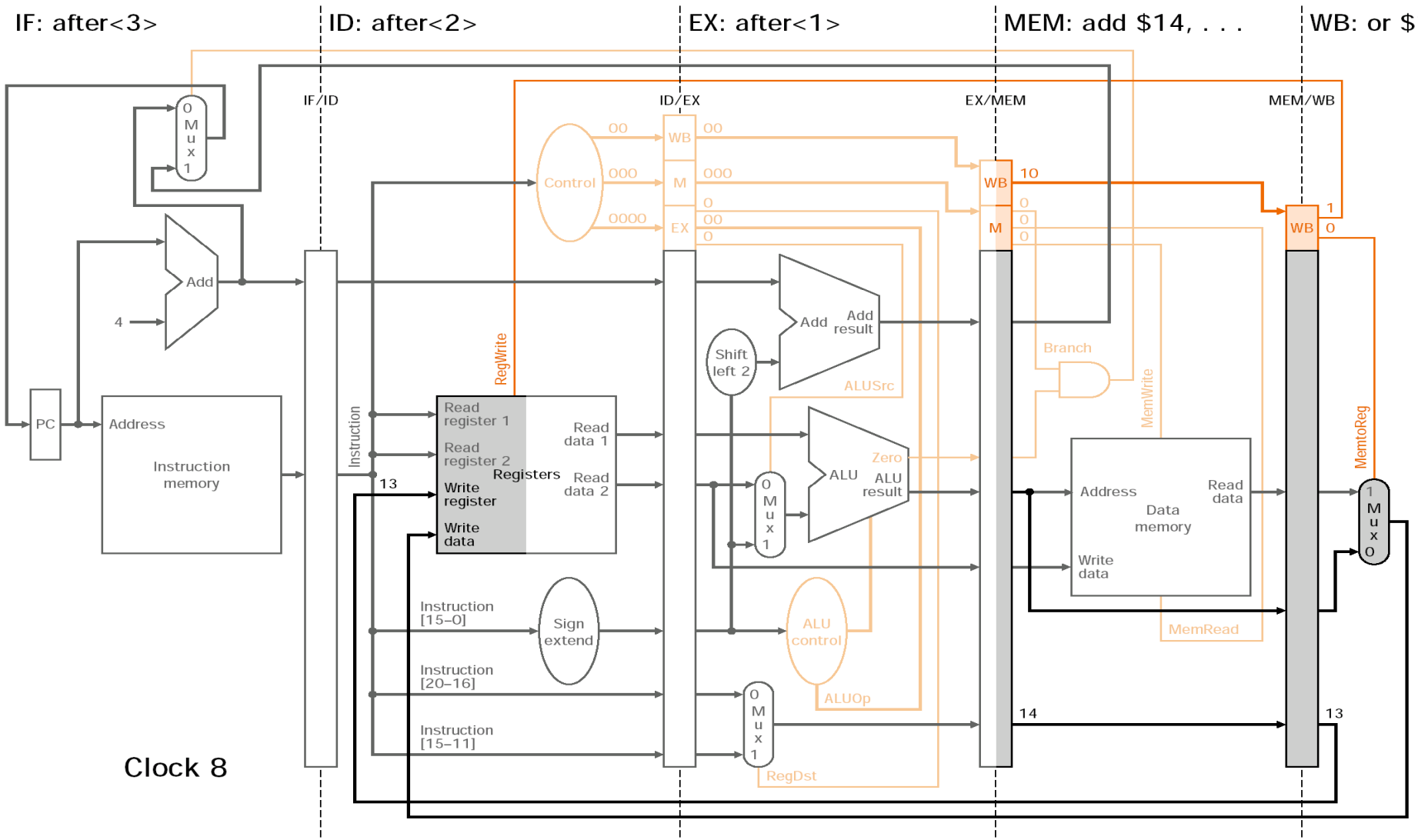
Example


IF: after<1>
ID: add $14, $8, $9
EX: or $13, . . .
MEM: and $12, . . .
WB: sub $11, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
0
1
add
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Control
10
000
1100
WB
M
EX
Shift left 2
Add result
ALUSrc
Branch
MemWrite
MemtoReg
MemRead
8
9
11
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
$8
$9
$6
$7
ALU
ALU result
Zero
Address
Data memory
Read data
Write data
X
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
Sign extend
ALU control
ALUOp
14
13
12
11
RegDst
Clock 6

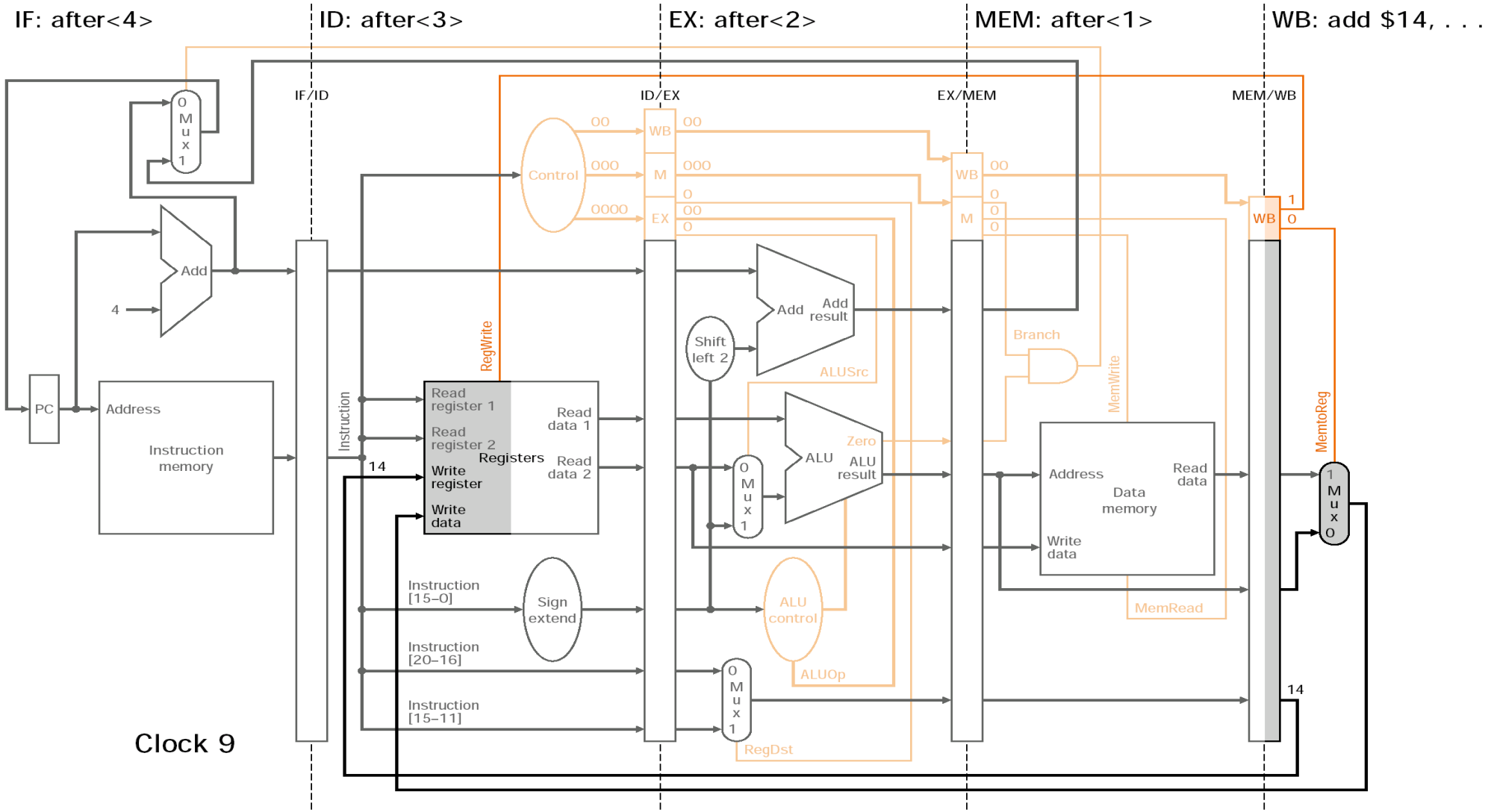
Example


IF: after<2>
ID: after<1>
EX: add $14, . . .
MEM: or $13, . . .
WB: and $12, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
Add
4
PC
Address
Instruction memory
Instruction
12
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Control
00
000
0000
WB
M
EX
10
000
1
10
0
Shift left 2
Add
Add result
ALUSrc
$8
$9
ALU
ALU result
Zero
Branch
MemWrite
Address
Data memory
Read data
Write data
MemRead
MemtoReg
Instruction [15–0]
Sign extend
Instruction [20–16]
Instruction [15–11]
ALU control
ALUOp
14
RegDst
13
12
Clock 7

Example


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

Example


IF: after<4>
ID: after<3>
EX: after<2>
MEM: after<1>
WB: add $14, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 9
PC
Address
Instruction memory
Add
4
Instruction
14
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
Control
WB
M
EX
Shift left 2
Add result
ALUSrc
ALU
Zero
ALU result
ALU control
ALUOp
Sign extend
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
RegDst
Branch
MemWrite
MemRead
Data memory
Read data
MemtoReg