Chap. 4   The Processor

Part C
Pipelined Datapath and Control

** NUS, Aaron Tan 교수의 강의자료를 일부 포함하고 있습니다. **

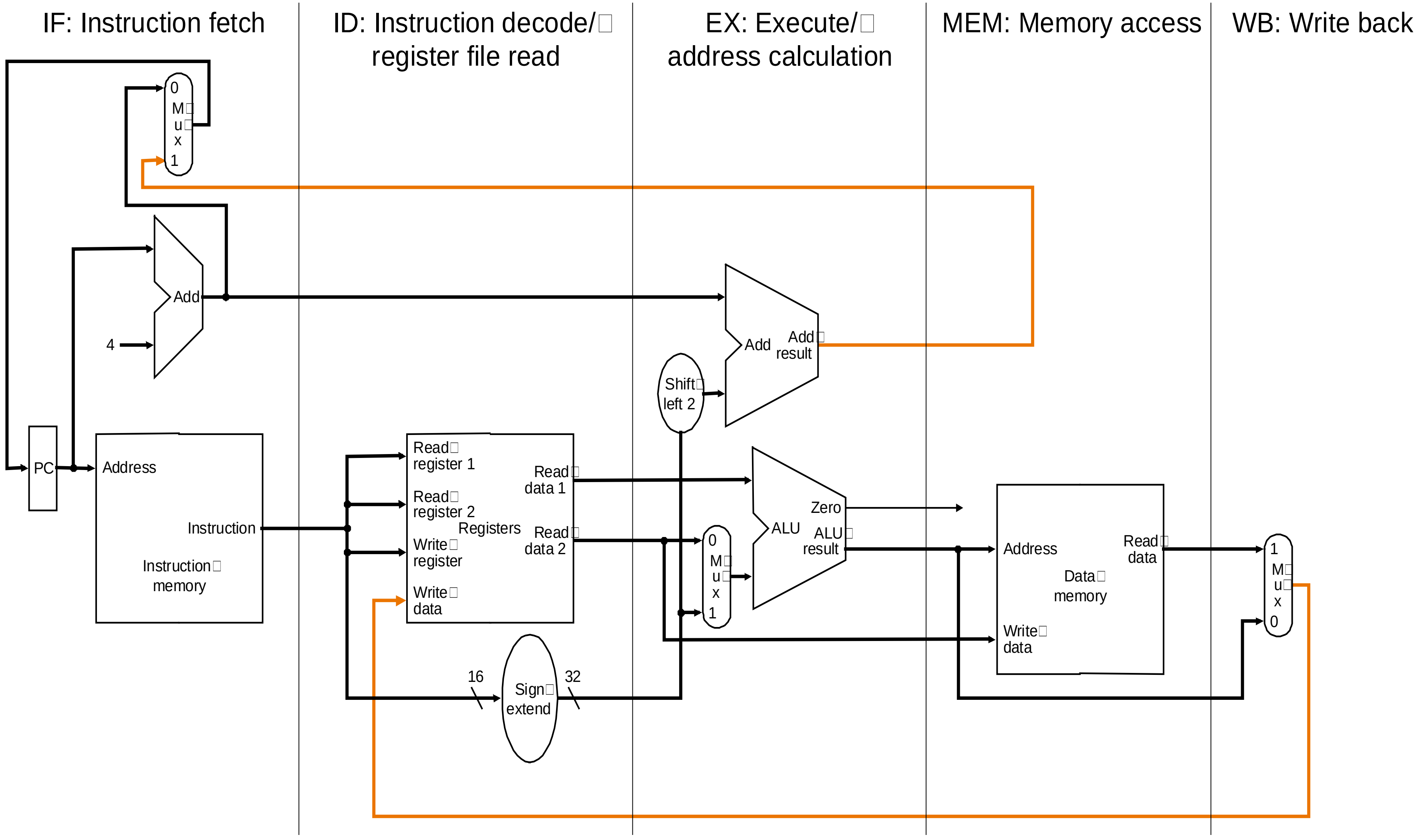
Introduction: Pipelining Lessons

- Pipelining doesn’t help **latency** of single task:
  - It helps the **throughput** of entire workload
- **Multiple** tasks operating simultaneously using different resources
- Pipeline Hazards

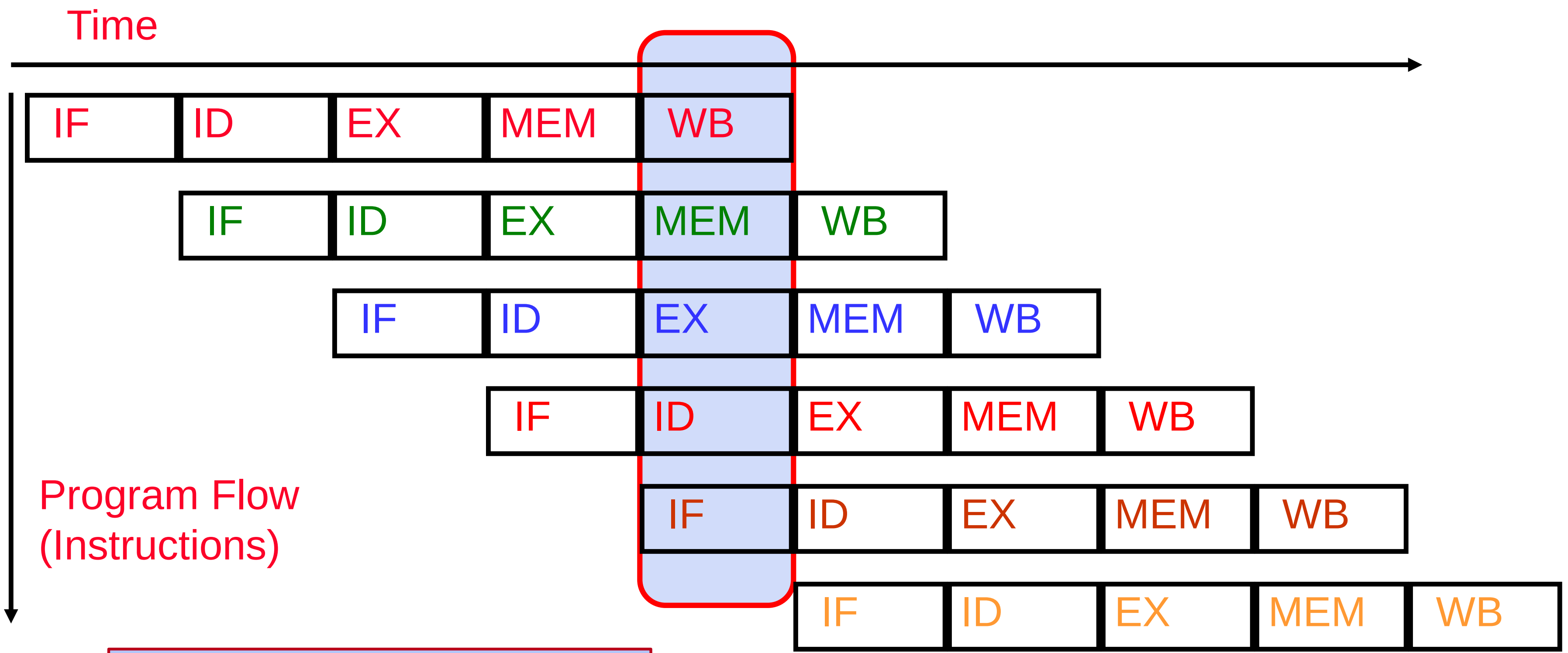
MIPS Pipeline Stages (1/2)

- **Five** Execution Stages
  - **IF**: Instruction Fetch
  - **ID**: Instruction Decode and Register Read
  - **EX**: Execute an operation or calculate an address
  - **MEM**: Access an operand in data memory
  - **WB**: Write back the result into a register
- **Idea:**
  - **Each execution stage takes 1 clock cycle**
  - General flow of data is from one stage to the next
- **Exceptions:**
  - Update of PC and write back of register file – more about this later…

MIPS Pipeline Stages (2/2)



Pipelined Execution: Illustration



Several instructions are in the pipeline simultaneously!

MIPS Pipeline: Datapath (1/3)

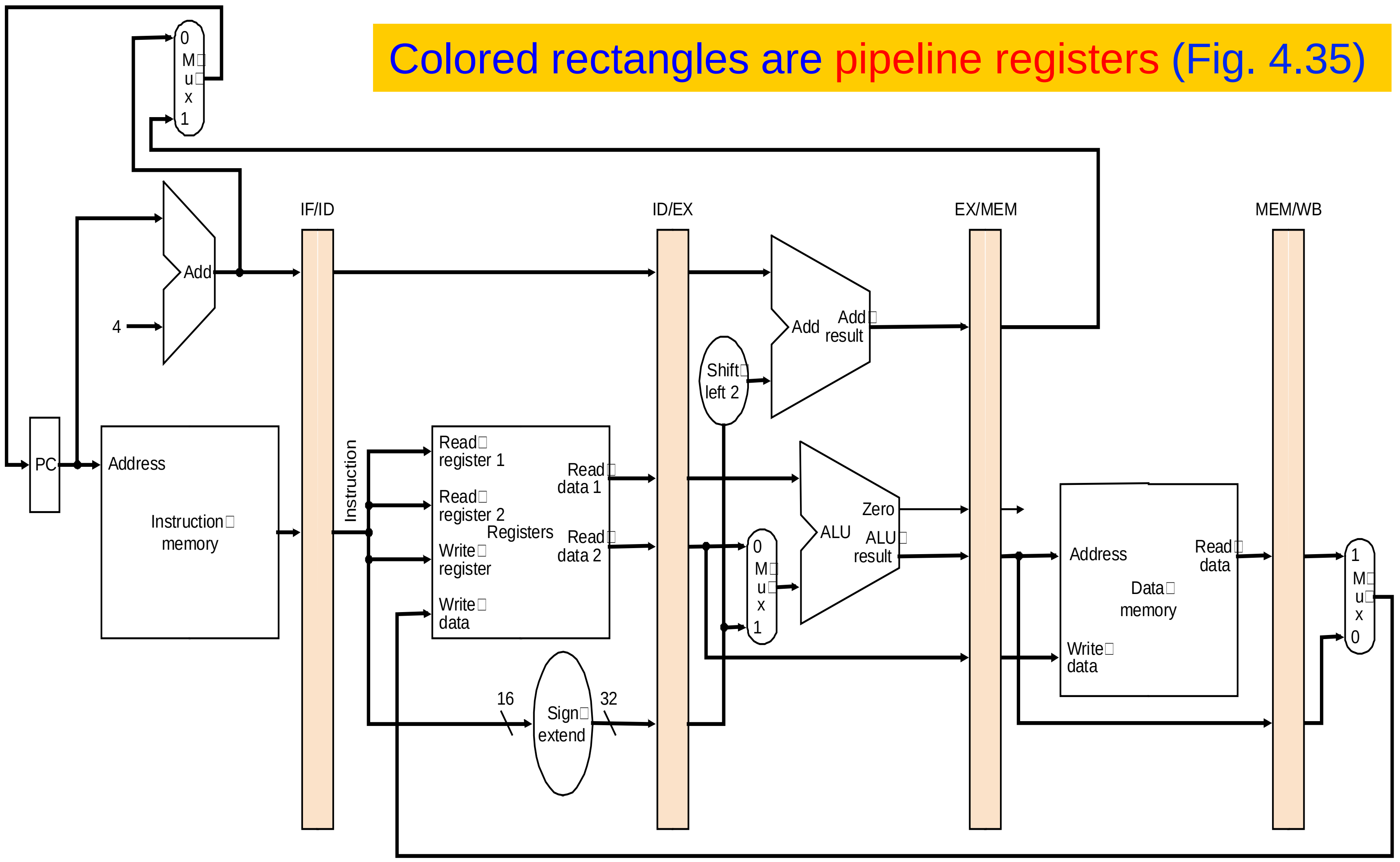
- Single-cycle implementation:
  - Update all state elements (**PC**, register file, data memory) at the end of a clock cycle
- Pipelined implementation:
  - One cycle per pipeline stage
  - Data required for each stage needs to be stored separately (why?)

MIPS Pipeline: Datapath (2/3)

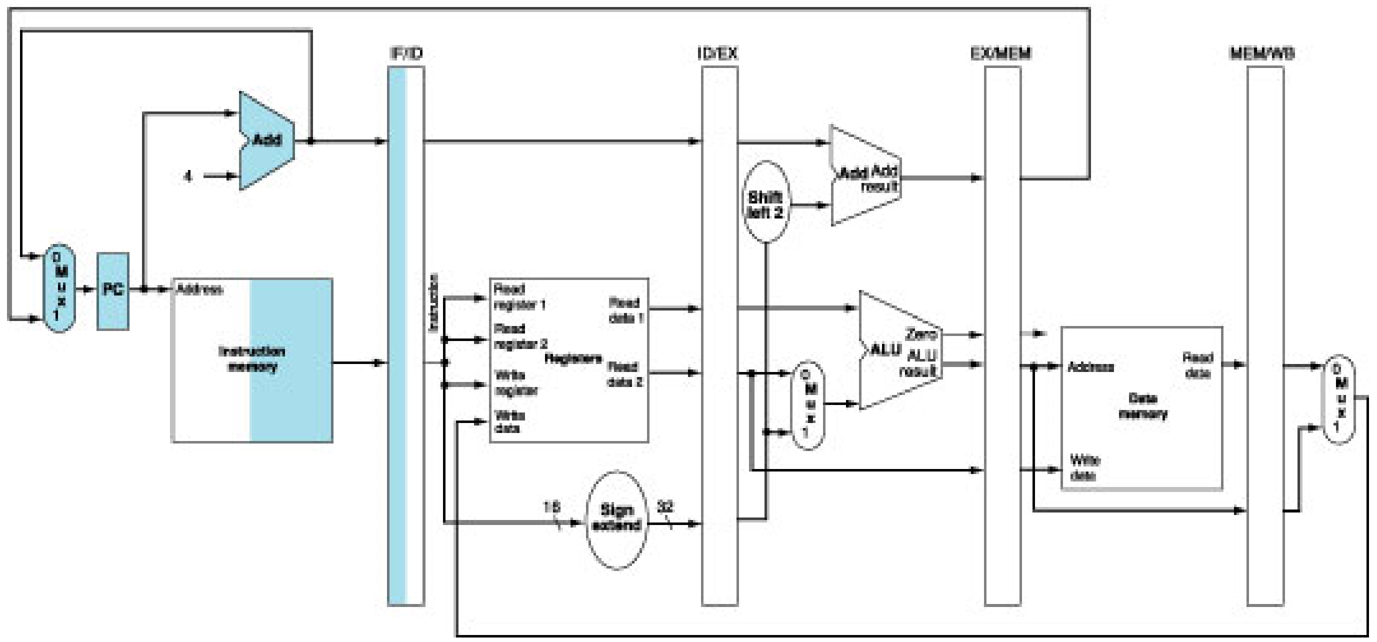
- Data used by **subsequent instructions:**
  - Store in programmer-visible state elements: **`PC`**, register file and memory
- Data used by **same instruction** in later pipeline stages:
  - Additional registers in datapath called **pipeline registers**
  - **`IF/ID`:** register between **`IF`** and **`ID`**
  - **`ID/EX:`** register between **`ID`** and **`EX`**
  - **`EX/MEM:`** register between **`EX`** and **`MEM`**
  - **`MEM/WB`:** register between **`MEM`** and **`WB`**

MIPS Pipeline: Datapath (3/3)

Colored rectangles are pipeline registers (Fig. 4.35)

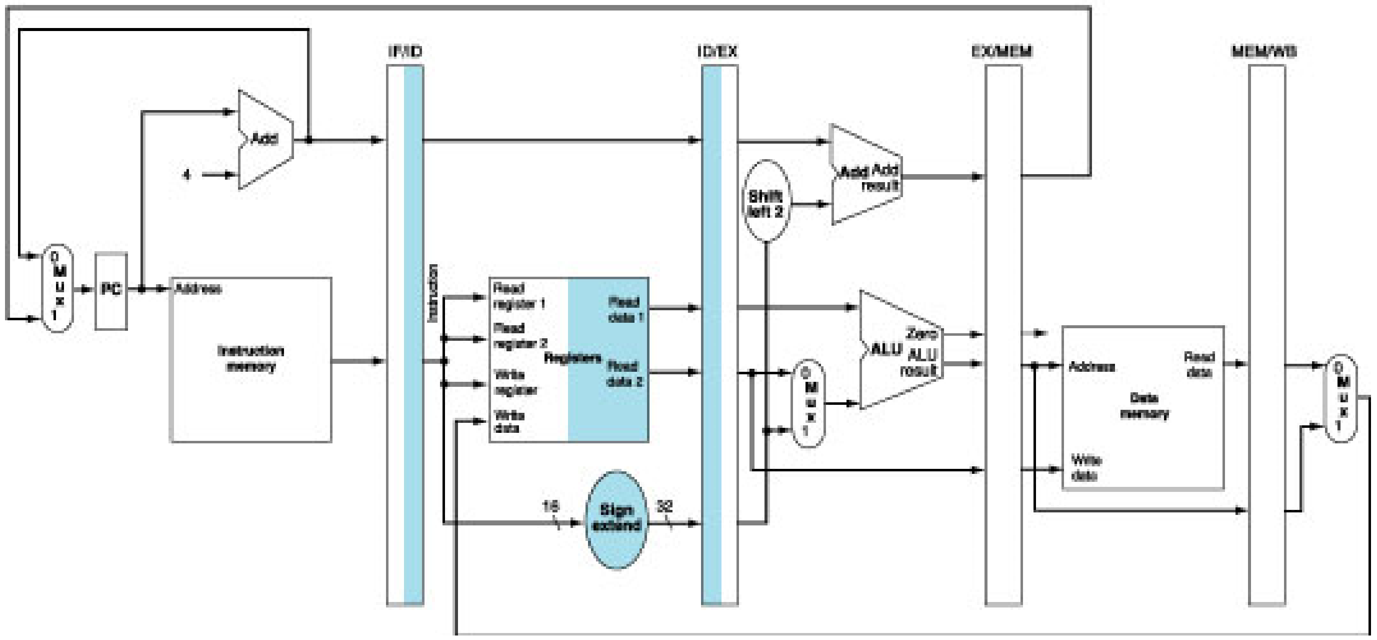


Pipeline Datapath: IF Stage



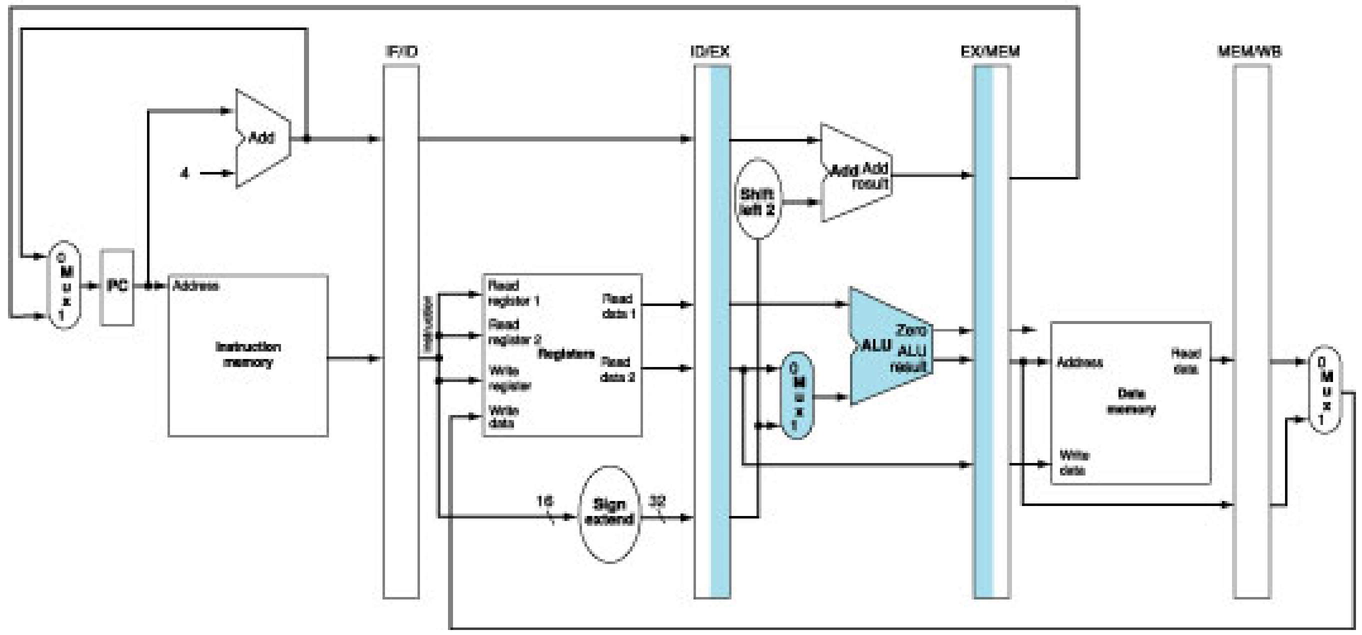
- At the end of a cycle, **IF/ID** receives (stores):
  - Instruction read from InstructionMemory[PC]
  - PC + 4
- PC + 4
  - Also connected to one of the MUX's inputs (another coming later)

Pipeline Datapath: ID Stage



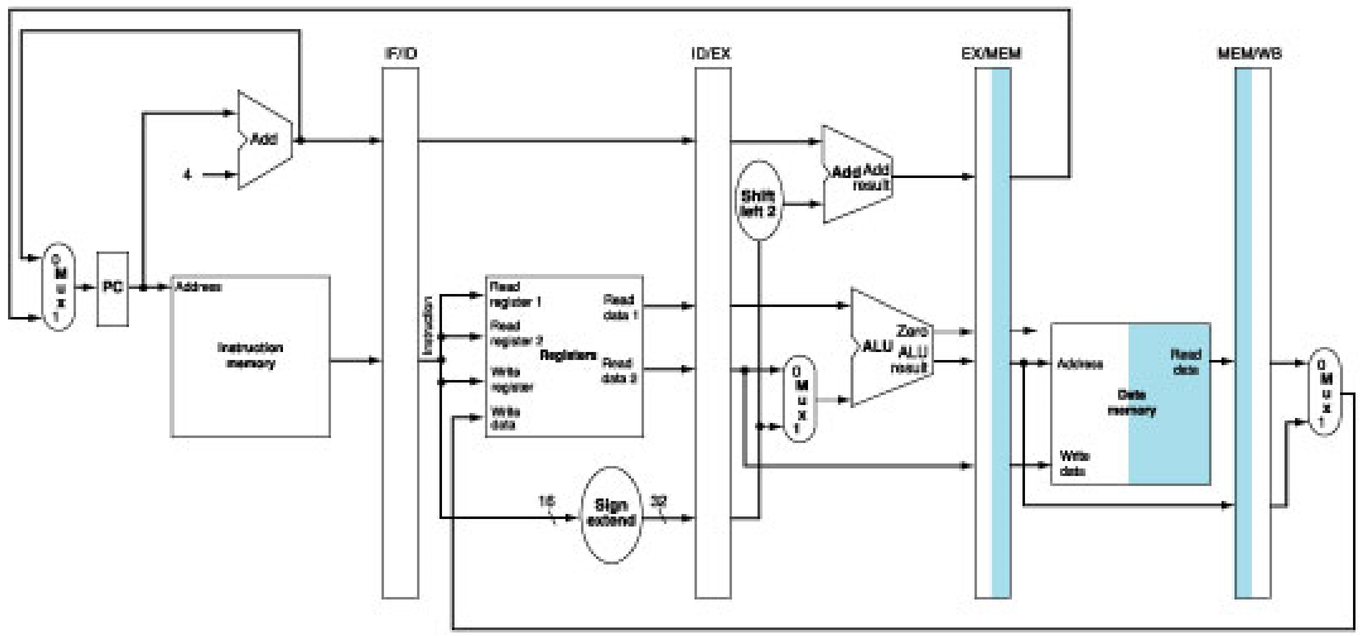
At the beginning of a cycle **IF/ID register supplies:**	At the end of a cycle **ID/EX receives:**
❖ Register numbers for reading two registers ❖ 16-bit offset to be sign-extended to 32-bit	❖ Data values read from register file ❖ 32-bit immediate value ❖ **PC + 4**

Pipeline Datapath: EX Stage



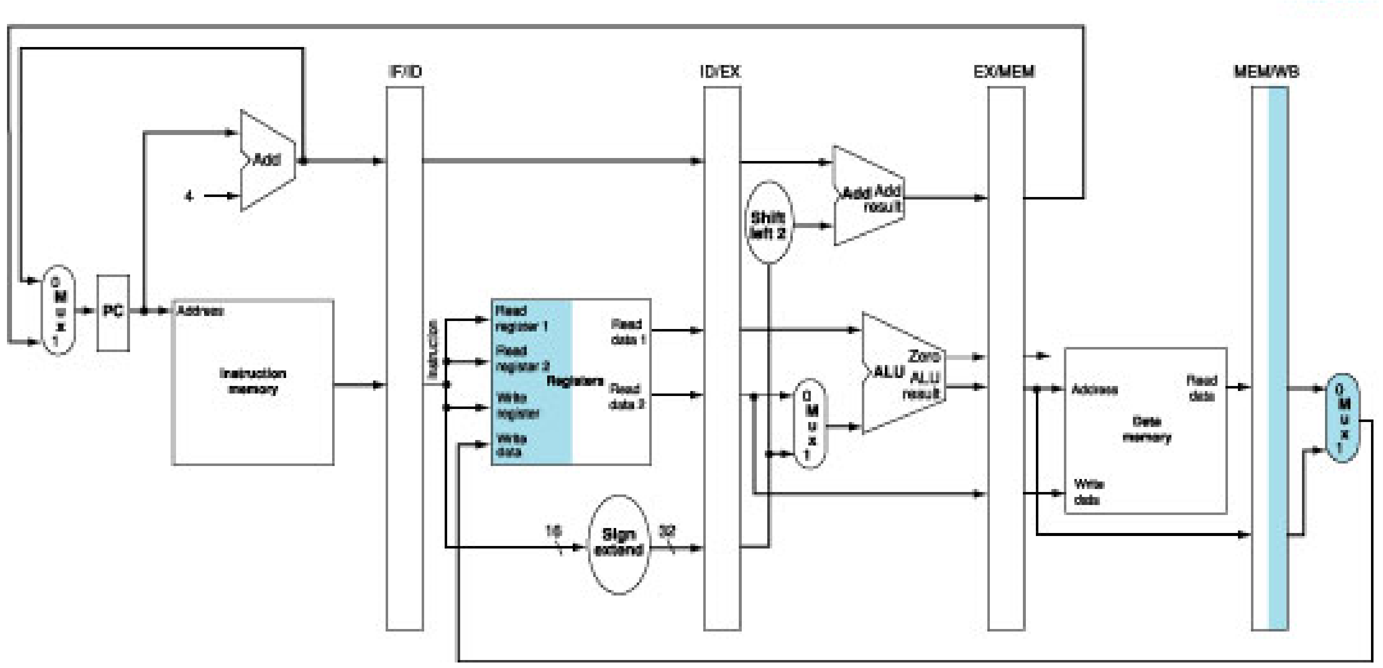
At the beginning of a cycle `ID/EX` register supplies:	At the end of a cycle `EX/MEM` receives:
❖ Data values read from register file ❖ 32-bit immediate value ❖ **`PC + 4`**	❖ (PC + 4) + (Immediate x 4) ❖ ALU result ❖ **`isZero?`** signal ❖ Data Read 2 from register file

Pipeline Datapath: MEM Stage



At the beginning of a cycle EX/MEM register supplies:	At the end of a cycle MEM/WB receives:
❖ (PC + 4) + (Immediate x 4) ❖ ALU result ❖ **`isZero?`** signal ❖ Data Read 2 from register file	❖ ALU result ❖ Memory read data

Pipeline Datapath: WB Stage



At the beginning of a cycle **MEM/WB** register supplies:	At the end of a cycle
❖ ALU result ❖ Memory read data	❖ Result is written back to register file (if applicable) ❖ **There is a bug here…….**

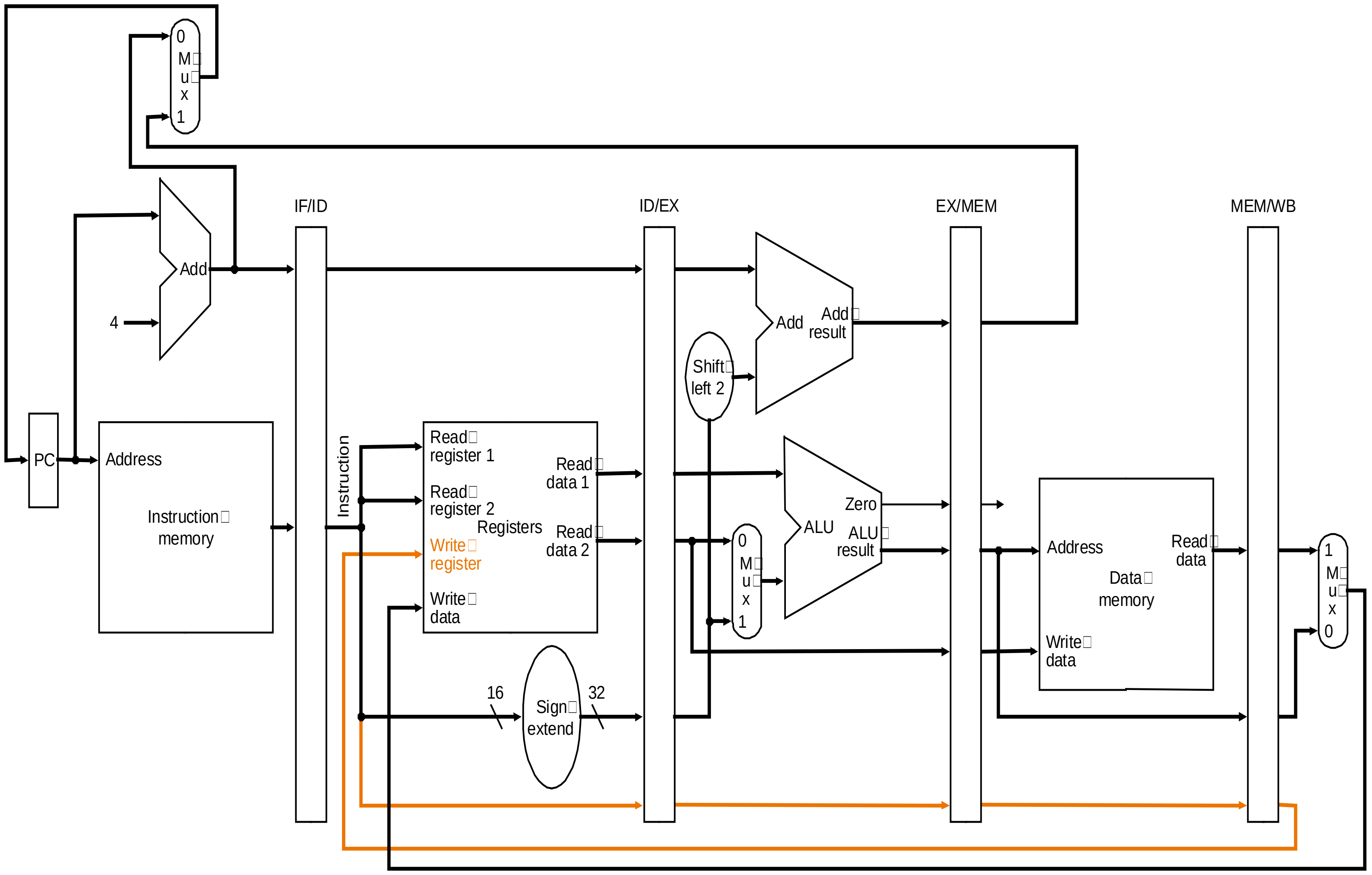
Corrected Datapath (1/2)

- Observe the "Write register" number
  - Supplied by the **`IF/ID`** pipeline register

  ➔ It is NOT the correct write register for the instruction now in **`WB`** stage!

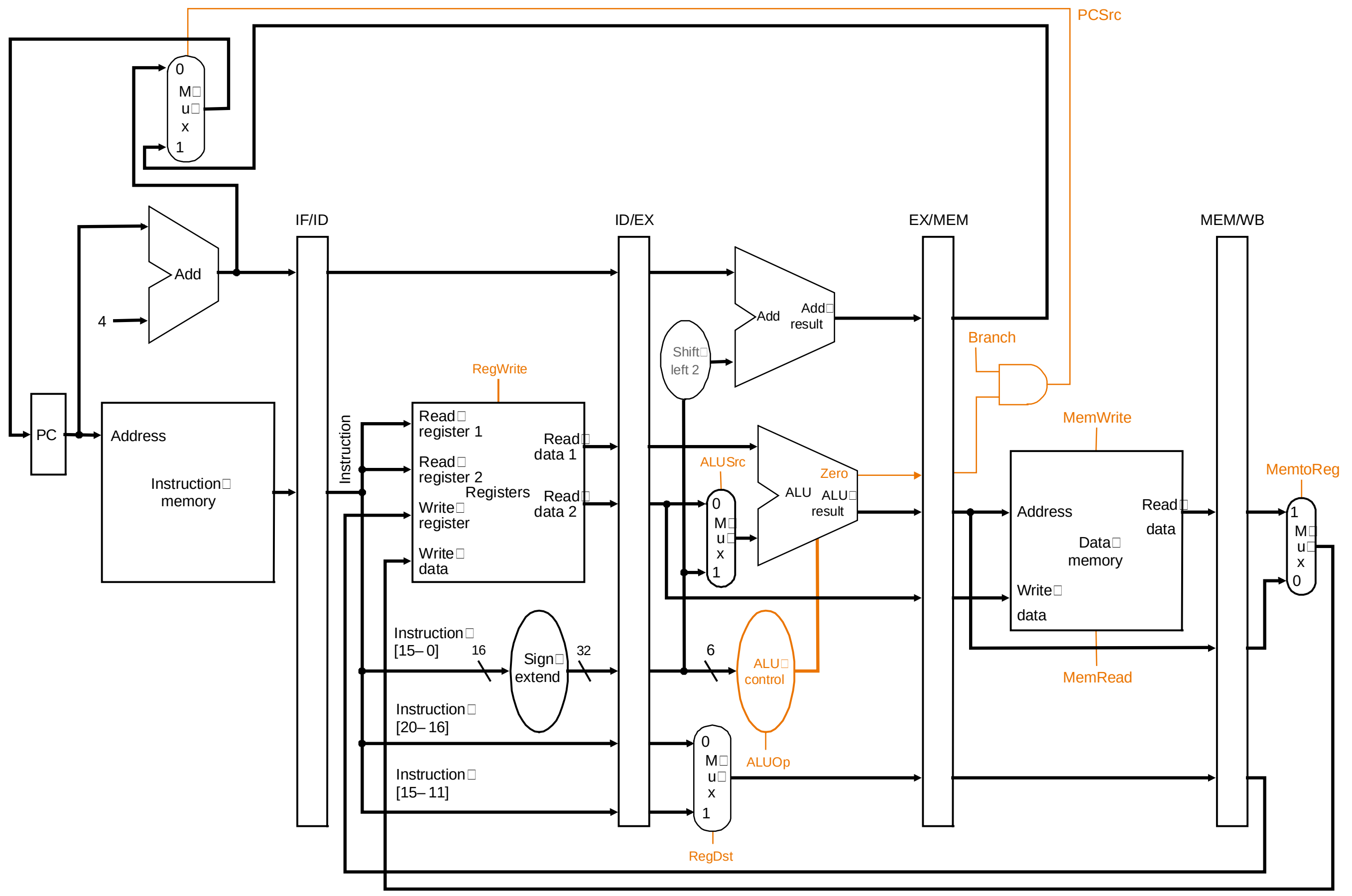
- **Solution:**
  - Pass "Write register" number from **`ID/EX`** through **`EX/MEM`** to **`MEM/WB`** pipeline register for use in **`WB`** stage
  - i.e. let the "Write register" number follows the instruction through the pipeline until it is needed in WB stage

Corrected Datapath (2/2)

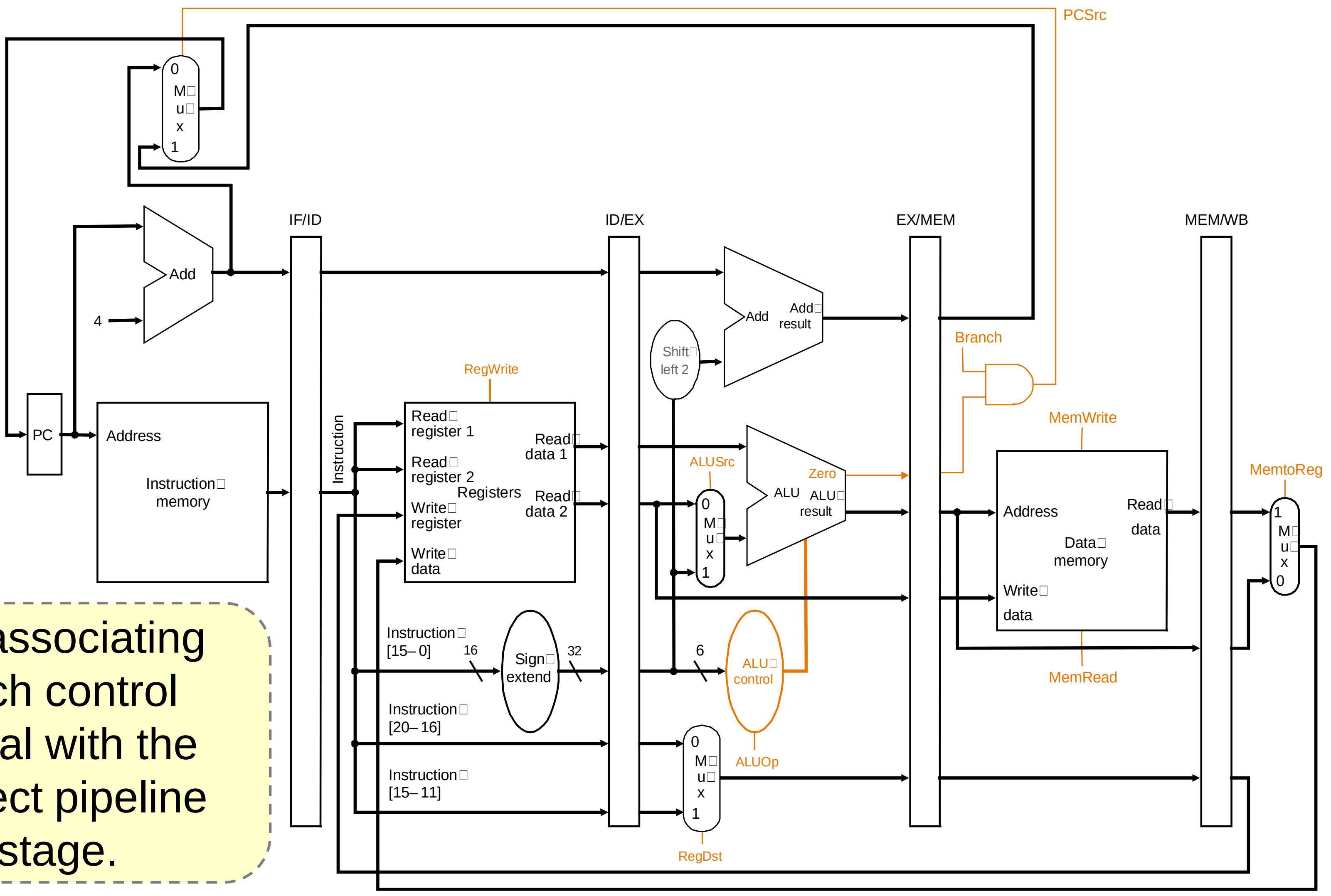


Pipeline Control: Main Idea

- Same control signals as single-cycle datapath
- **Difference:** Each control signal belongs to a particular pipeline stage



Pipeline Control



Try associating each control signal with the correct pipeline stage.

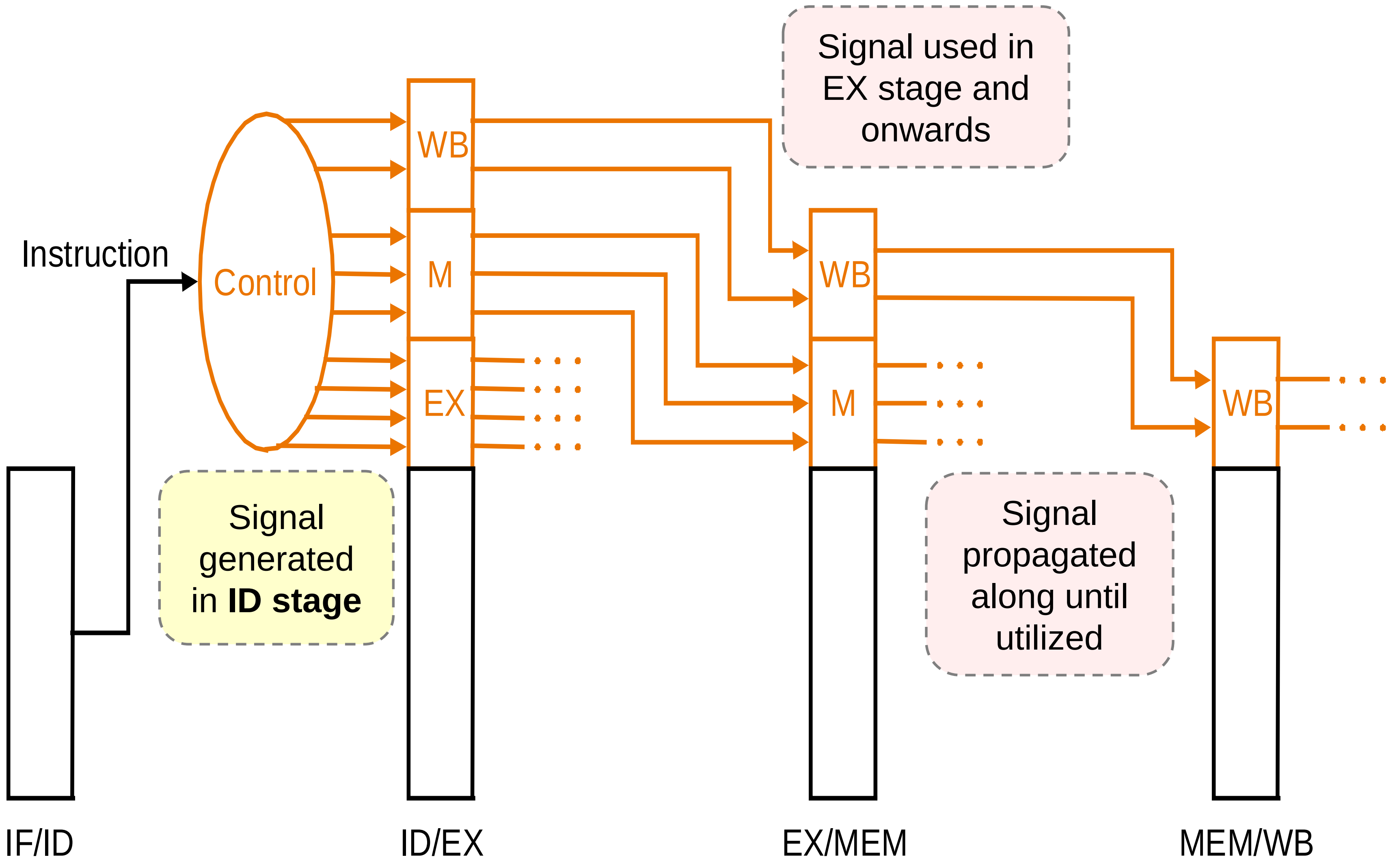
Pipeline Control: Grouping

- Group control signals according to pipeline stage

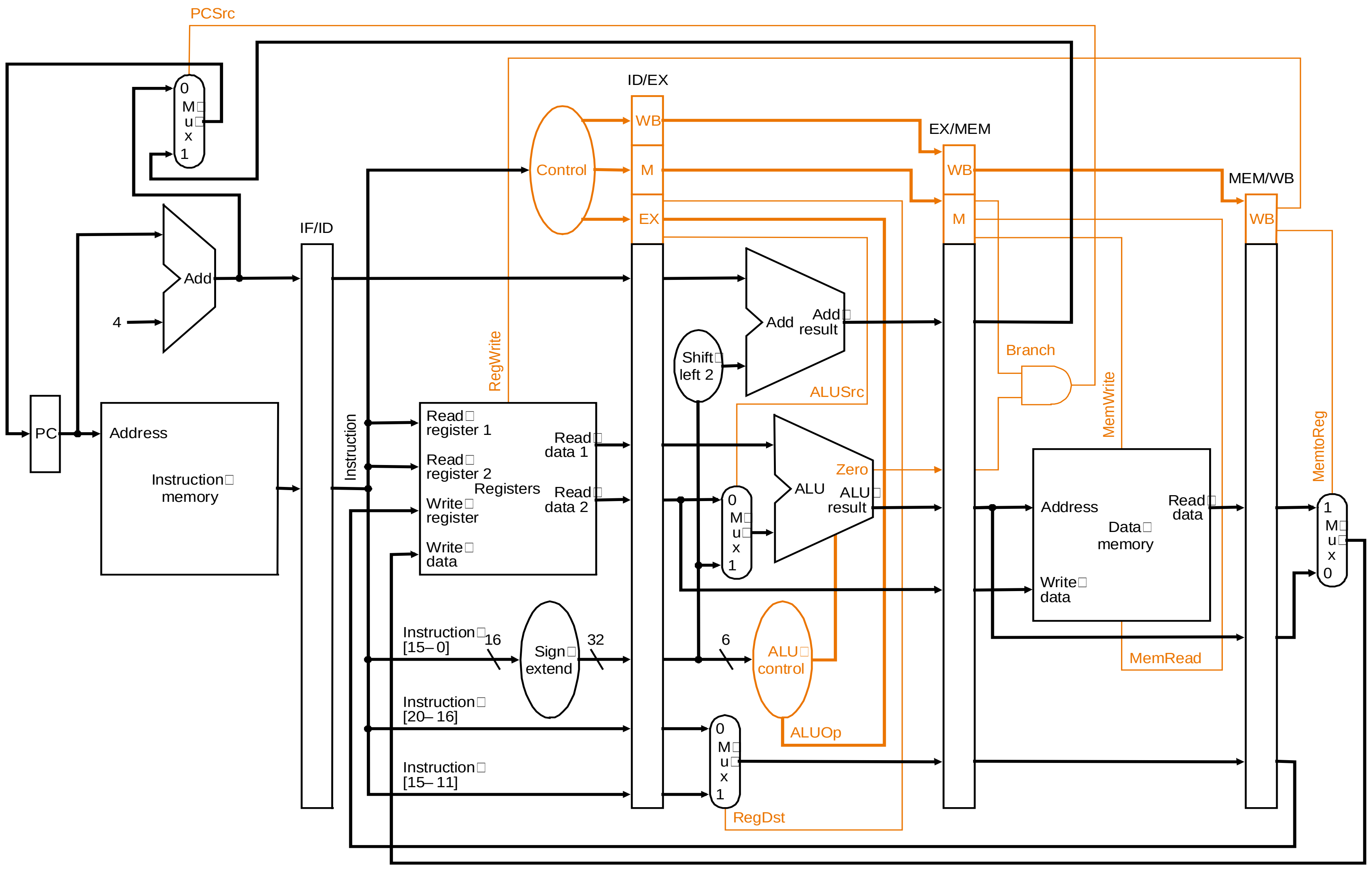
	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

	EX Stage				MEM Stage			WB Stage	
	RegDst	ALUSrc	ALUop		MemRead	MemWrite	Branch	MemToReg	RegWrite
			op1	op0					
R-type	1	0	1	0	0	0	0	0	1
lw	0	1	0	0	1	0	0	1	1
sw	X	1	0	0	0	1	0	X	0
beq	X	0	0	1	0	0	1	X	0

Pipeline Control: Implementation


Signal used in EX stage and onwards
Instruction
Control
WB
M
EX
WB
M
WB
Signal generated in **ID stage**
Signal propagated along until utilized
IF/ID
ID/EX
EX/MEM
MEM/WB

Pipelined Datapath and Control

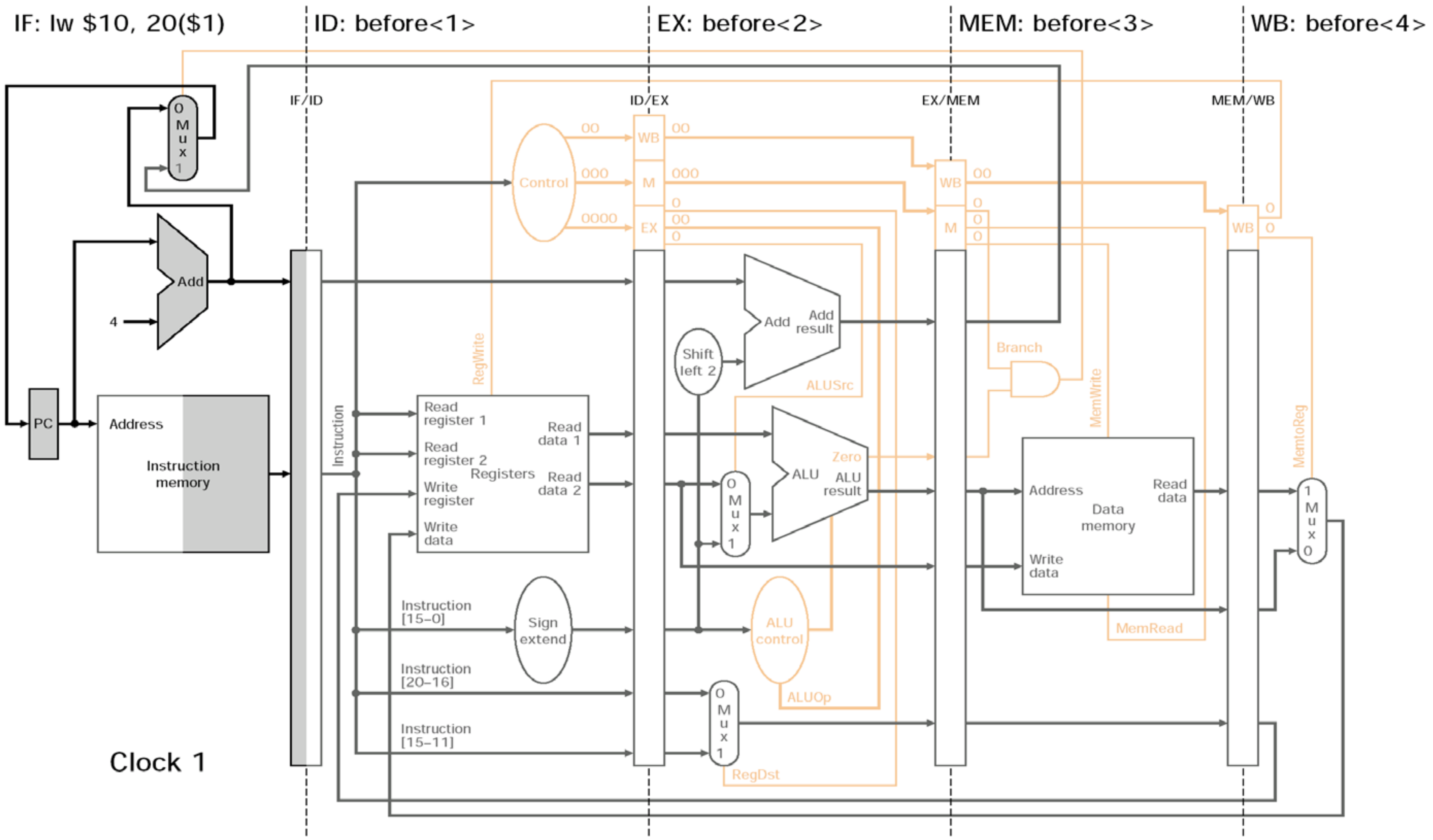


The Grand Example

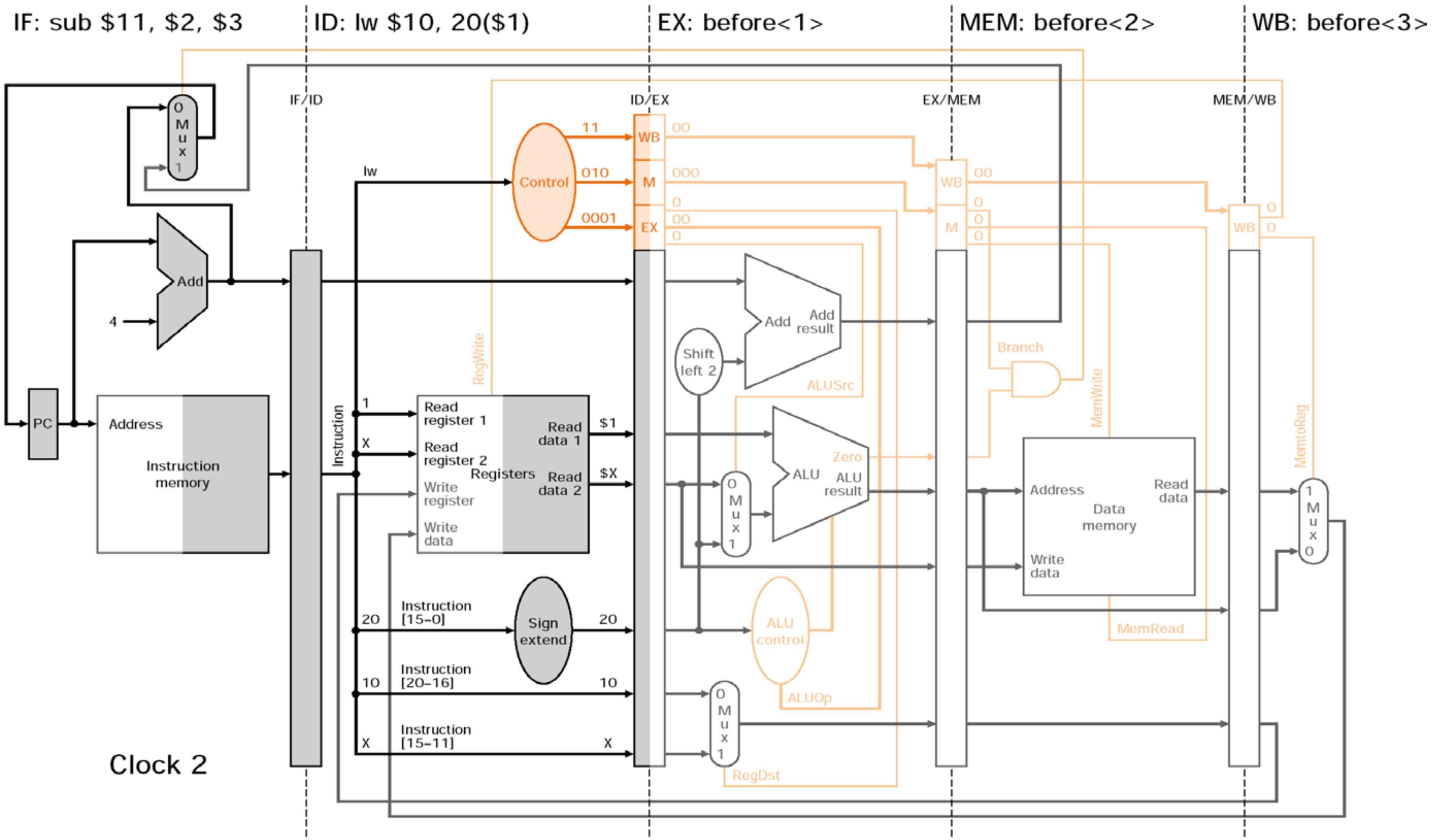
```
lw    $10, 20($1)
sub   $11, $2, $3
and   $12, $4, $5
or    $13, $6, $7
add   $14, $8, $9
```

Assumptions
- No data hazards
- No control hazards

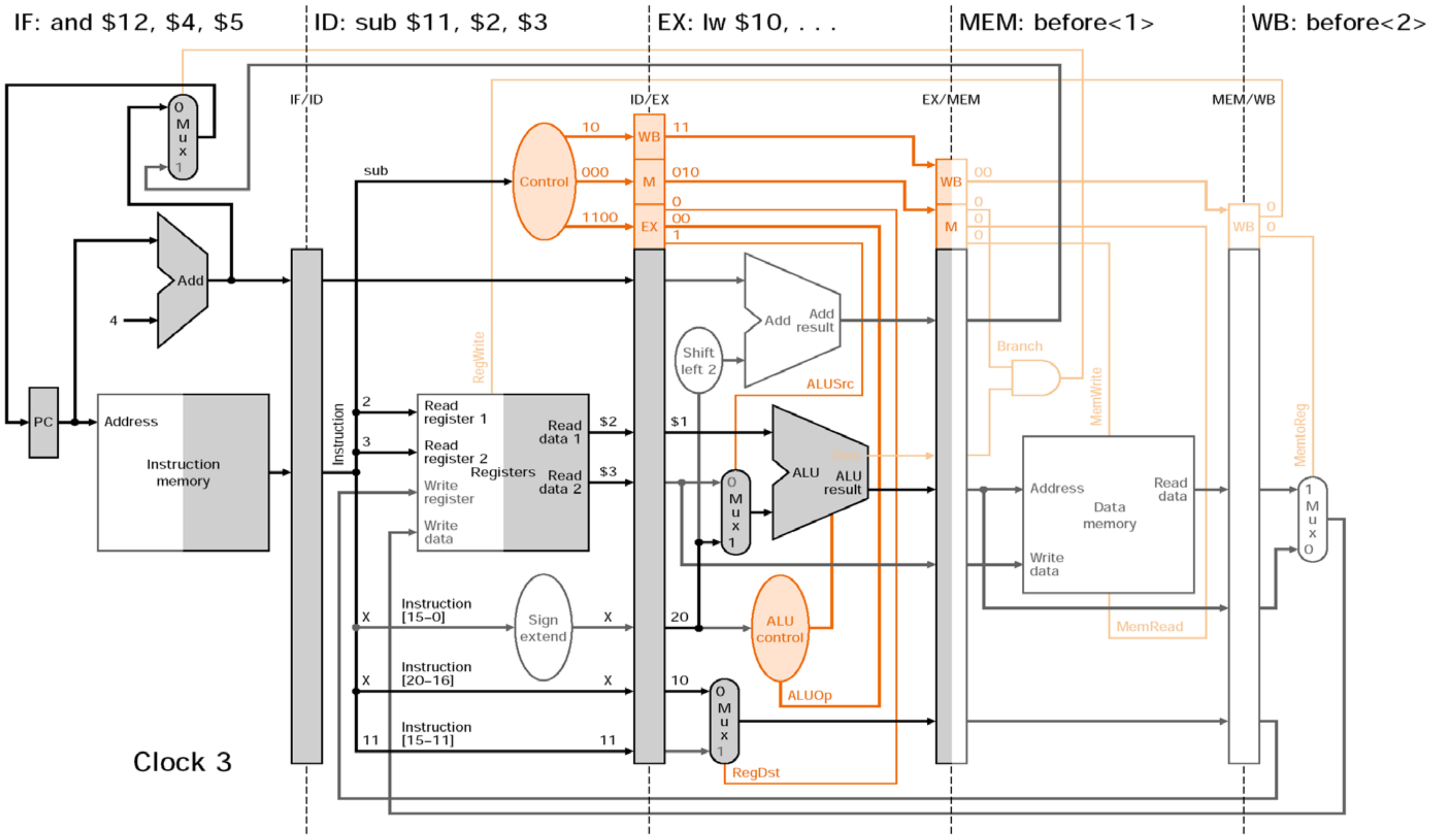
The Grand Example



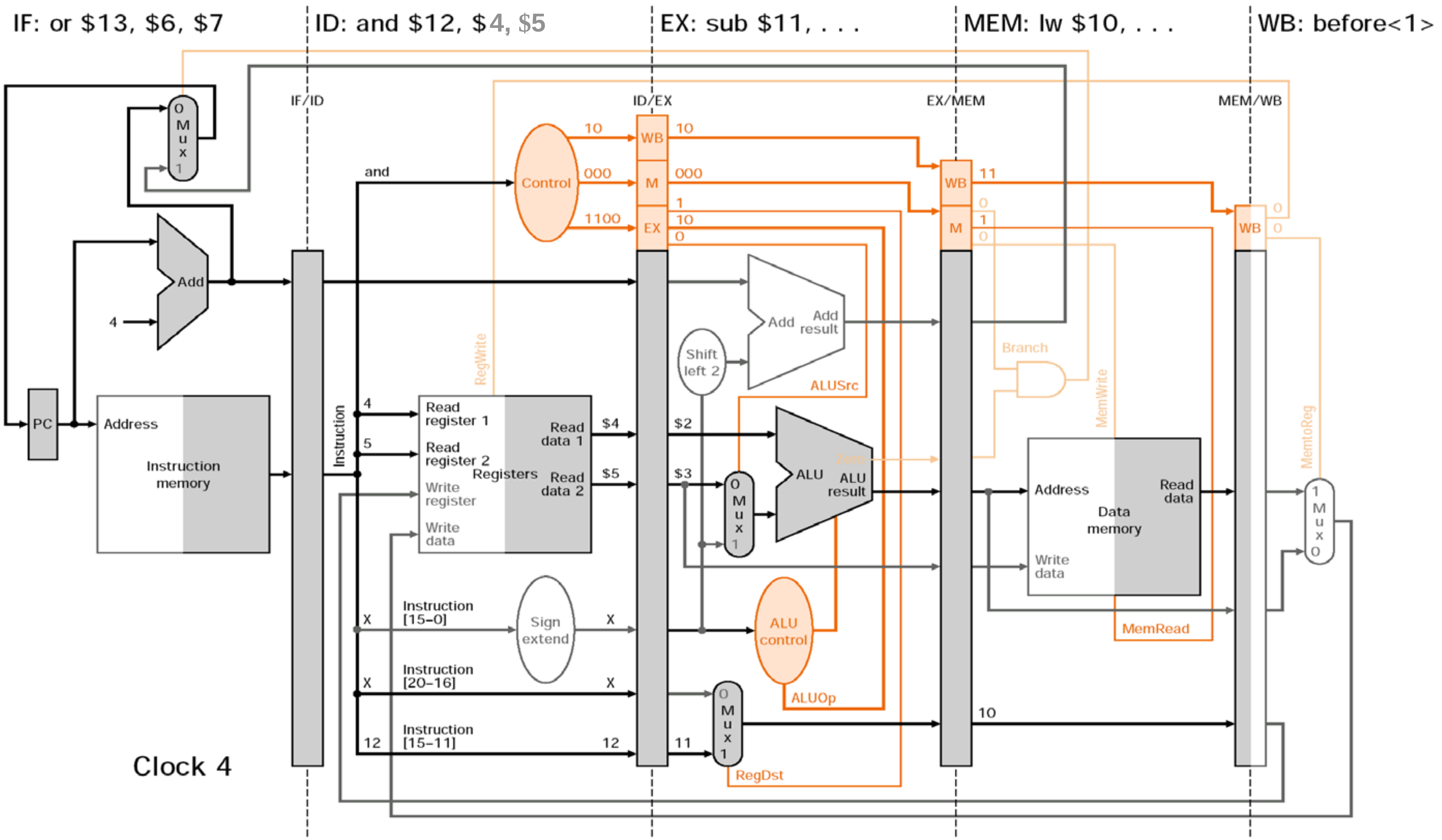
The Grand Example



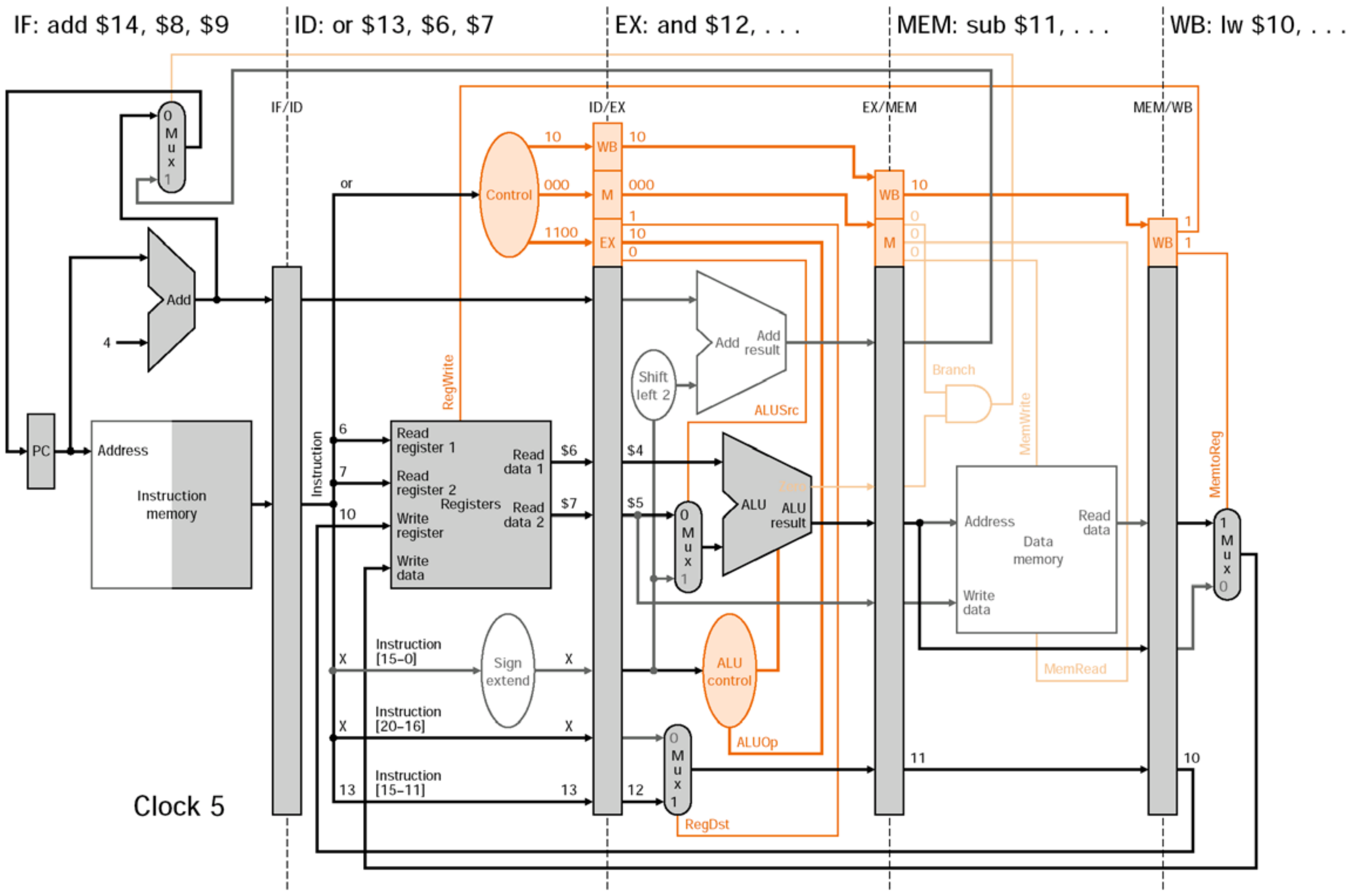
The Grand Example



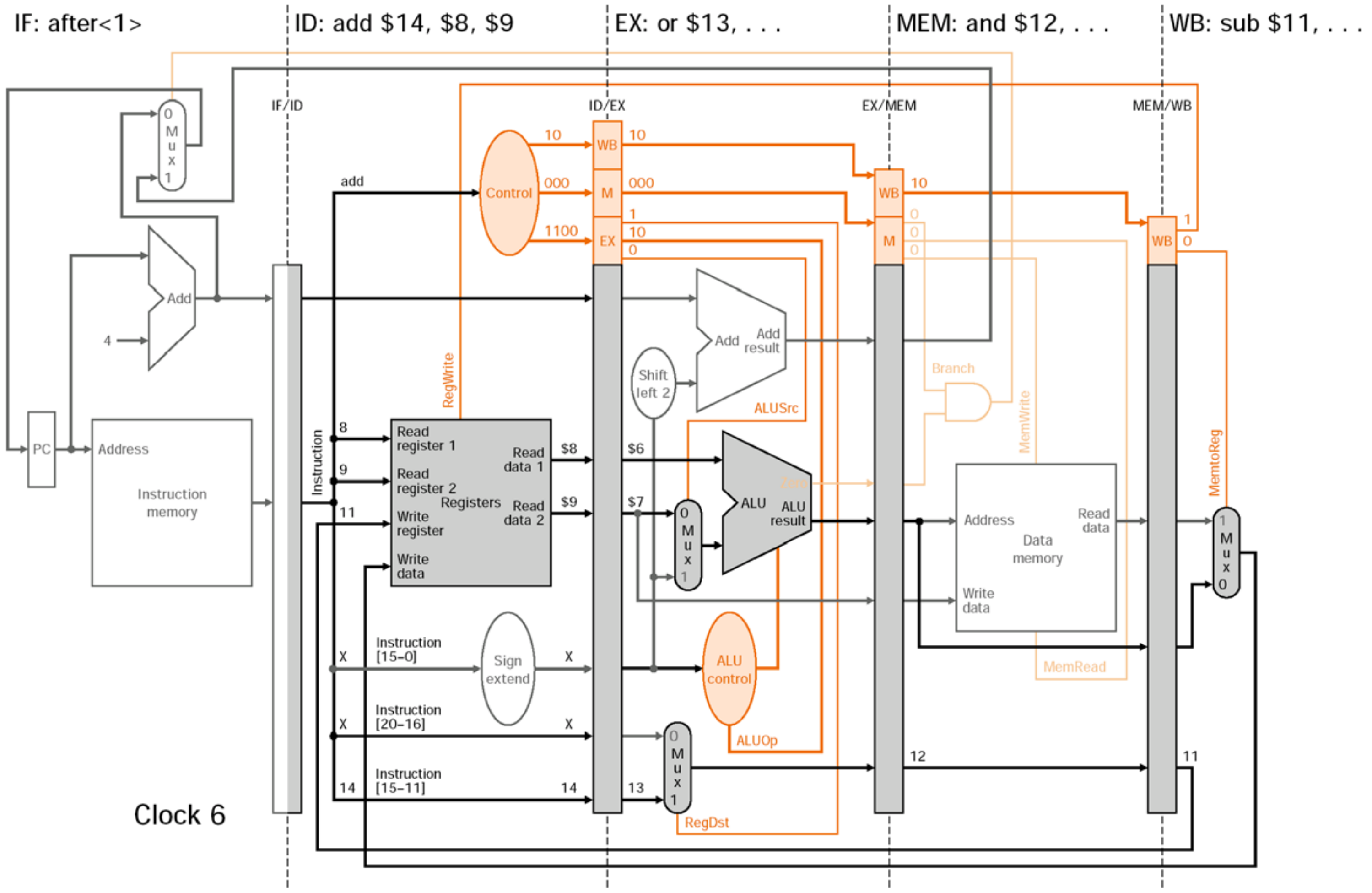
The Grand Example



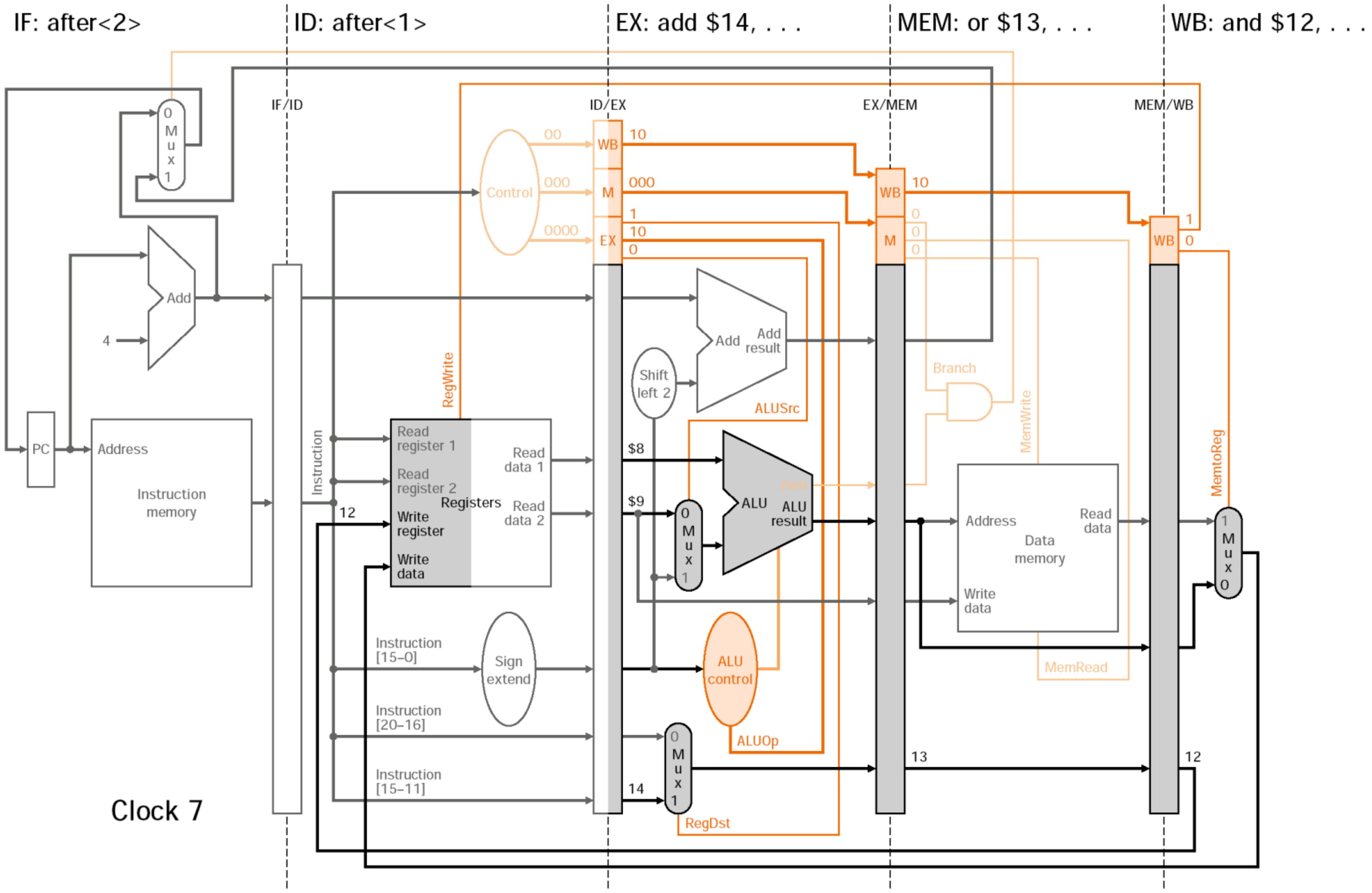
The Grand Example



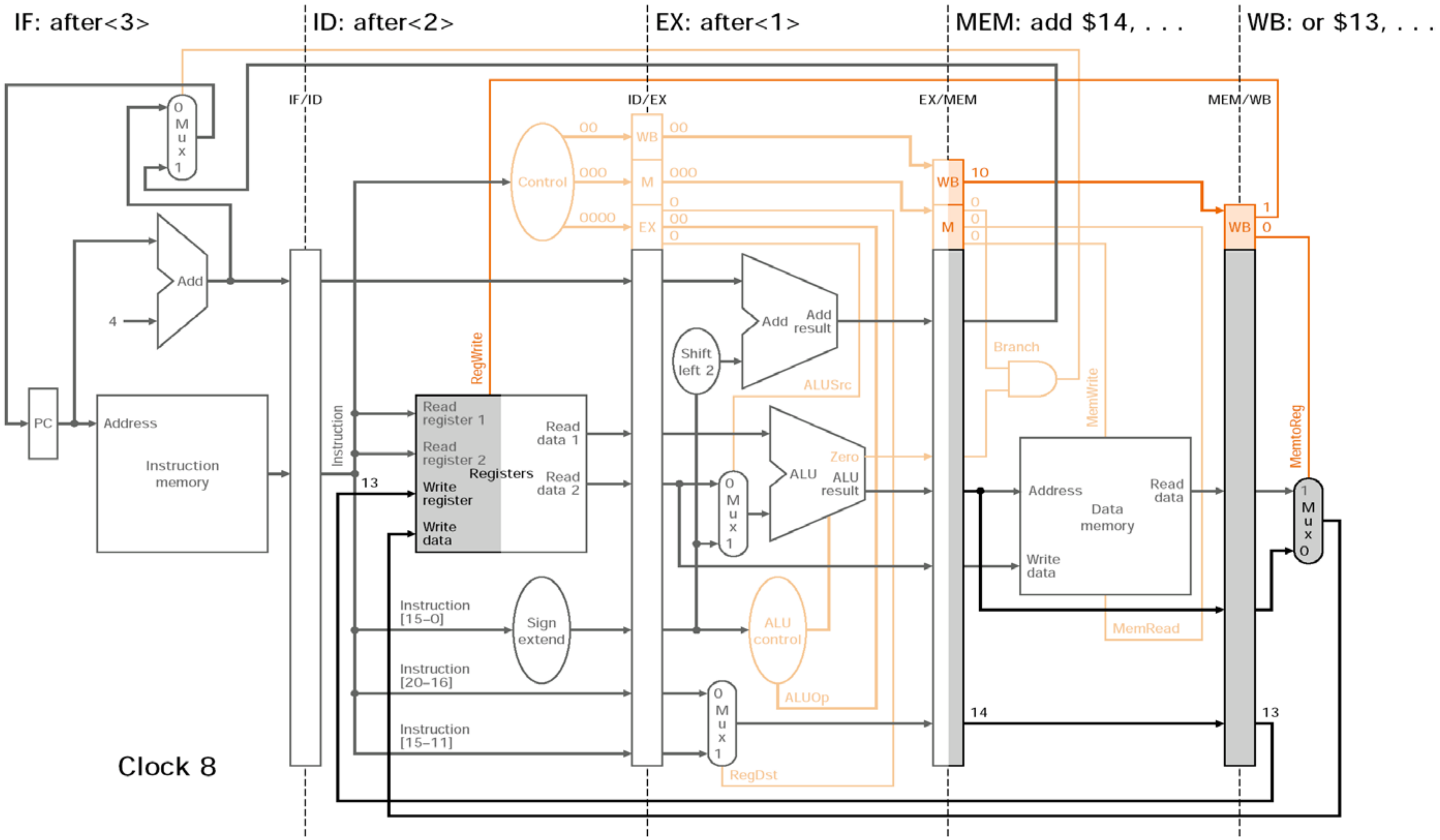
The Grand Example



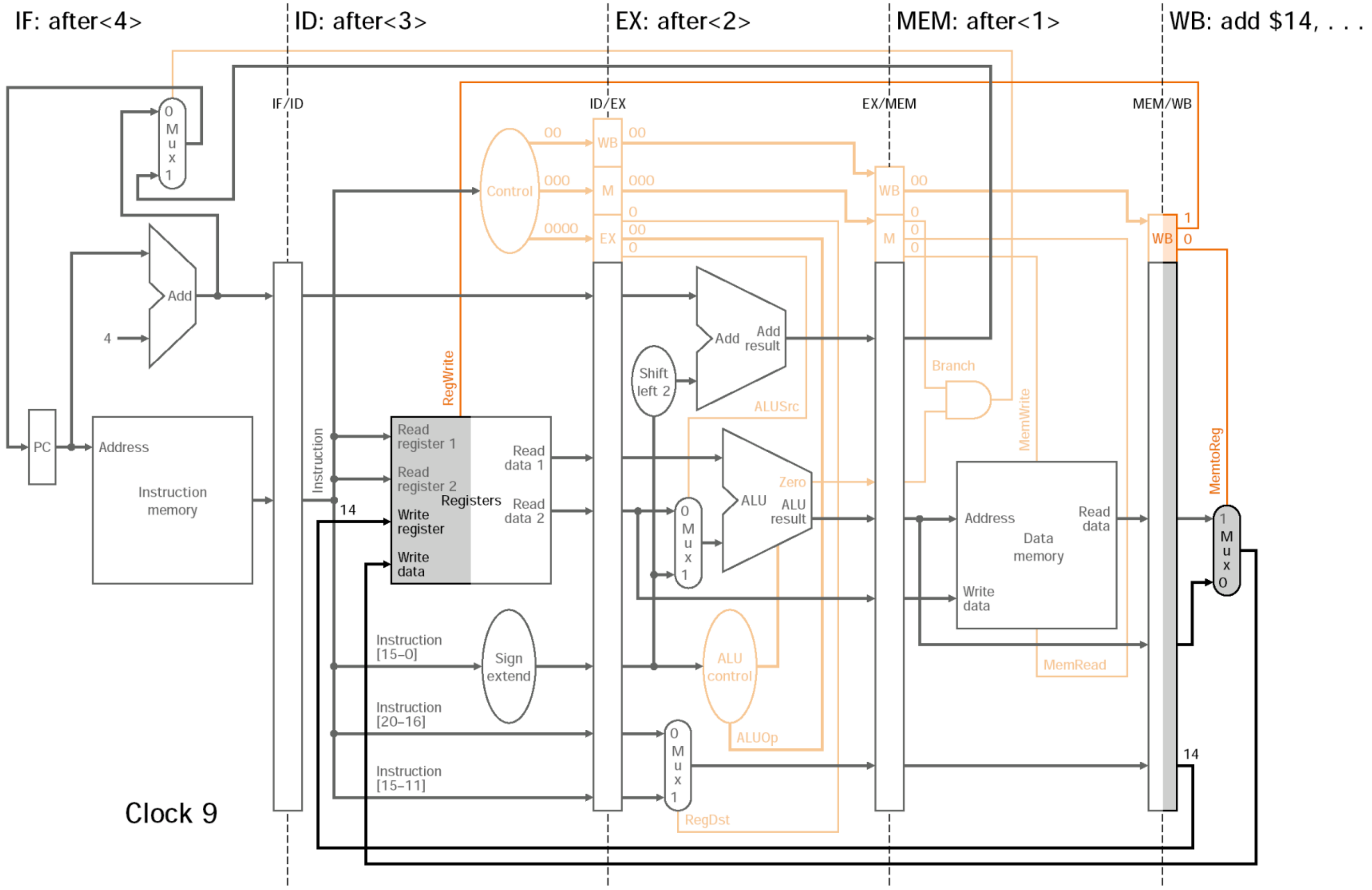
The Grand Example



The Grand Example



The Grand Example



Summary