VHDL
MIPS32

EL 310
Erkay Savaş
Sabancı University

MIPS32 Architecture

- 32 bit architecture
  - registers, ALU, instructions, shifter, data and address are 32 bit
- Load/Store architecture
  - There are special instructions to access memory such as `lw` (load word) and `sw` (store word)
  - No ALU operation involves an operand from the memory.
- RISC based
  - simple instructions
  - most of the instruction can be completed in five clock cycles

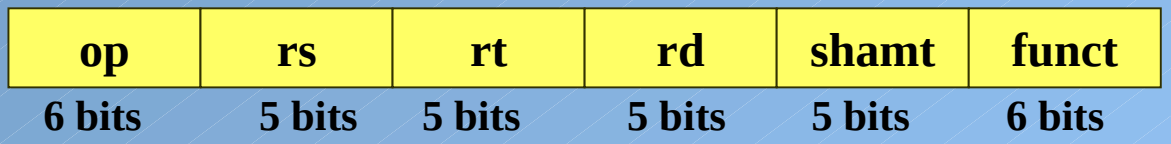
MIPS32 Registers

Name	Register no	Usage	Preserved on call
$zero	0	the constant value 0	n.a.
$v0-$v1	2-3	values for results and expression evaluation	no
$a0-$a3	4-7	arguments	no
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

Instruction Formats

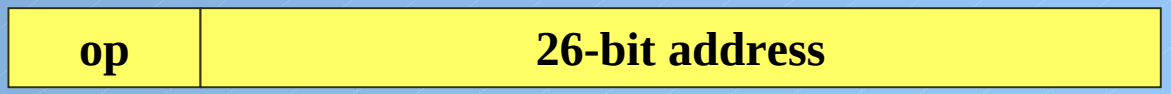
- There are three instruction format types

R type



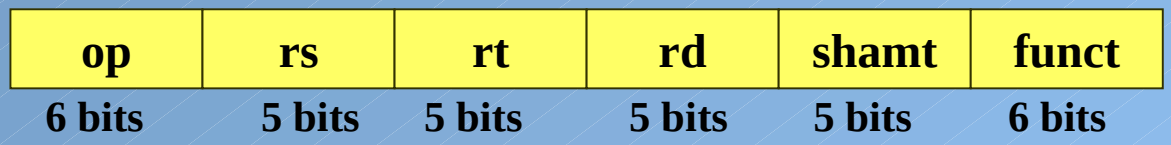
I type

J type



R-type Instruction

R type



name	Example						Comments
add	0	18	19	17	0	32	`add $s1, $s2, $s3`
sub	0	18	19	17	0	34	`sub $s1, $s2, $s3`
slt	0	18	19	17	0	42	`sub $s1, $s2, $s3`
jr	0	31	0	0	0	8	`jr $ra`

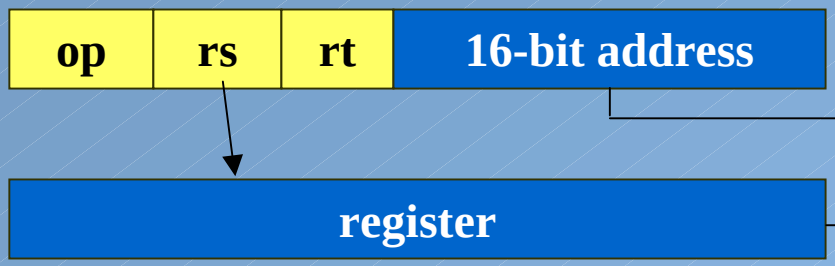
I-type Instruction

I type

name	Example				Comments
lw	35	18	17	Immediate	`lw $s1, 100($s2)`
sw	43	18	17	Immediate	`sw $s1, 100($s2)`
beq	4	17	18	Immediate	`beq $s1, $s2, 100`
bne	5	17	18	Immediate	`bne $s1, $s2, 100`

Addressing Modes

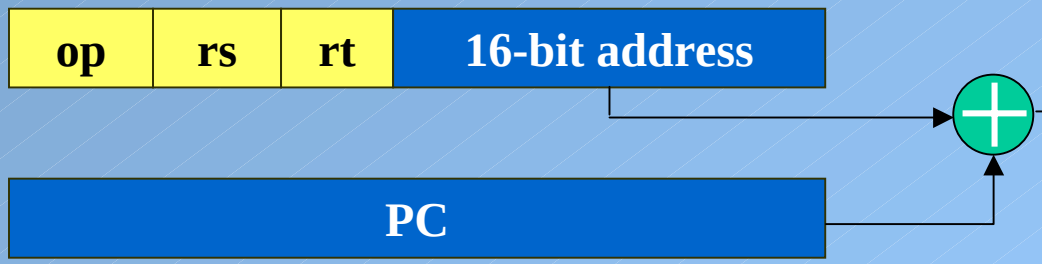
Base addressing



data memory

```
lw $s1, index($s2)
```

PC-relative addressing

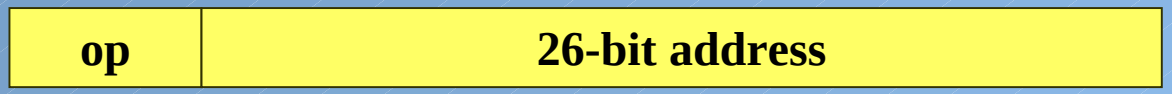


instruction memory

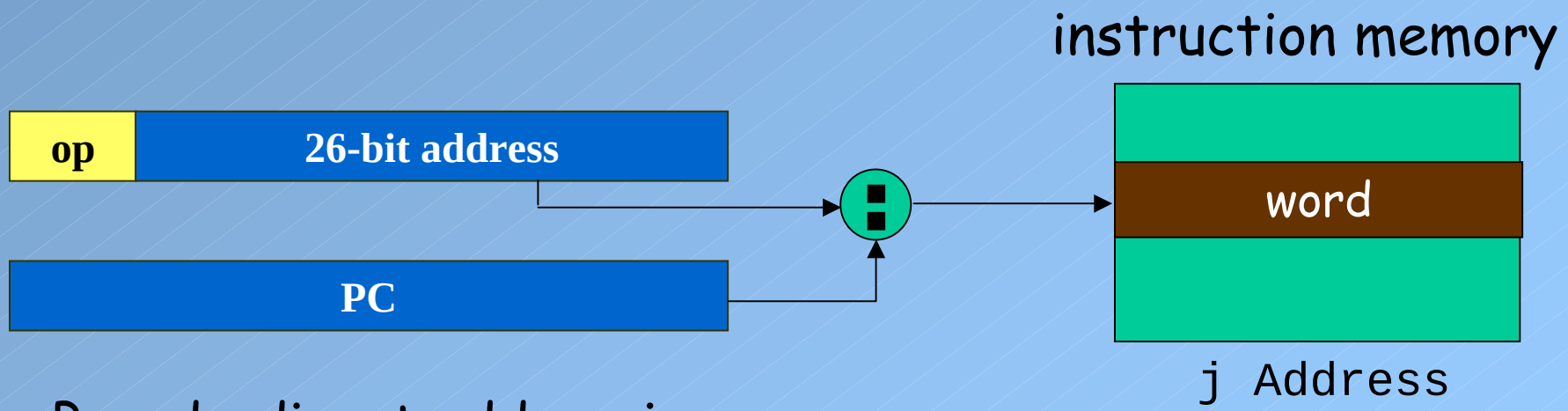
```
beq $s1, $s2, Loop
```

J-type Instructions

J type



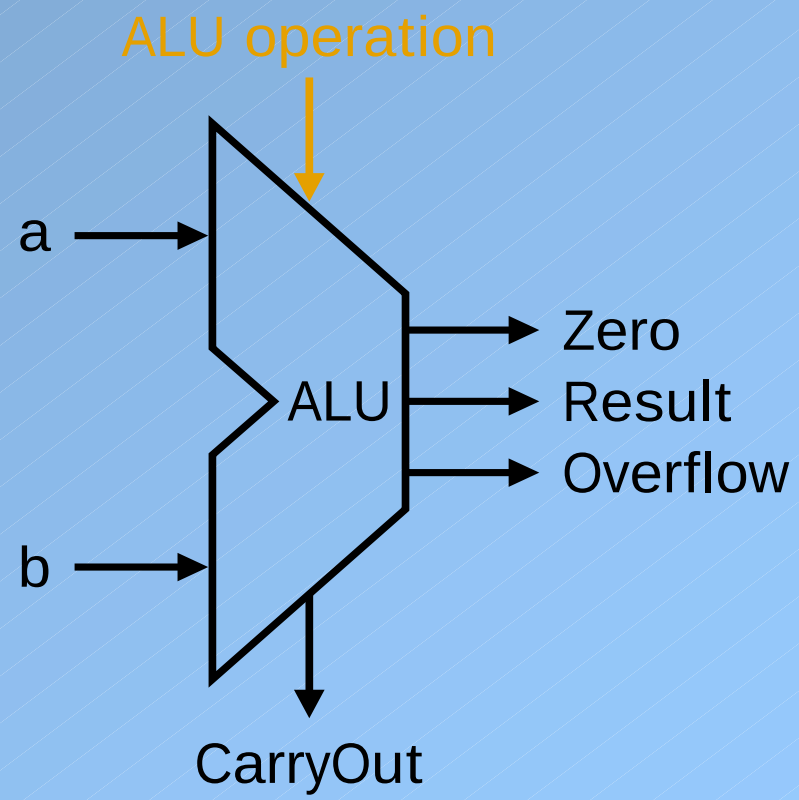
name	Example		Comments
j	2	Immediate	j 10000
jal	3	Immediate	jal 10000



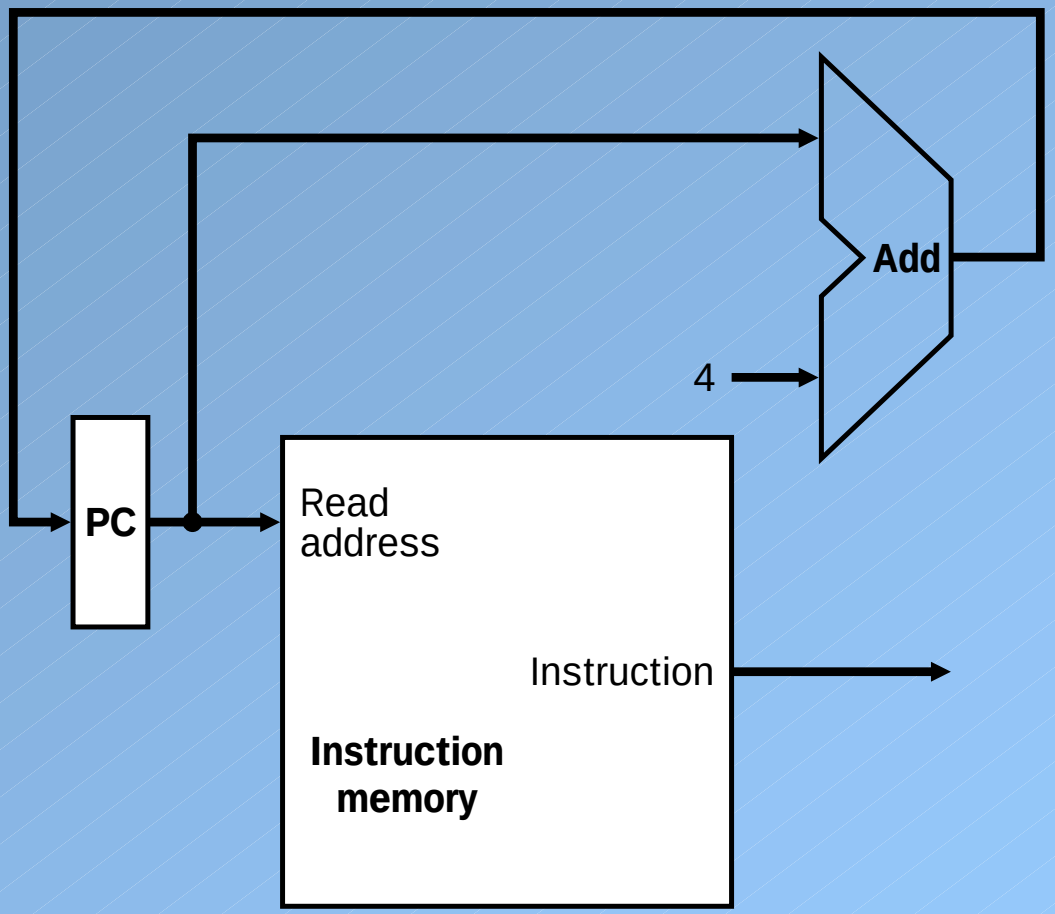
Pseudo-direct addressing

ALU Symbol & Control

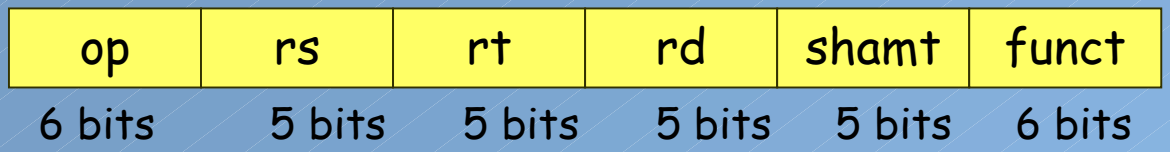
ALU control lines	Function
000	and
001	or
010	add
110	subtract
111	Set on less than



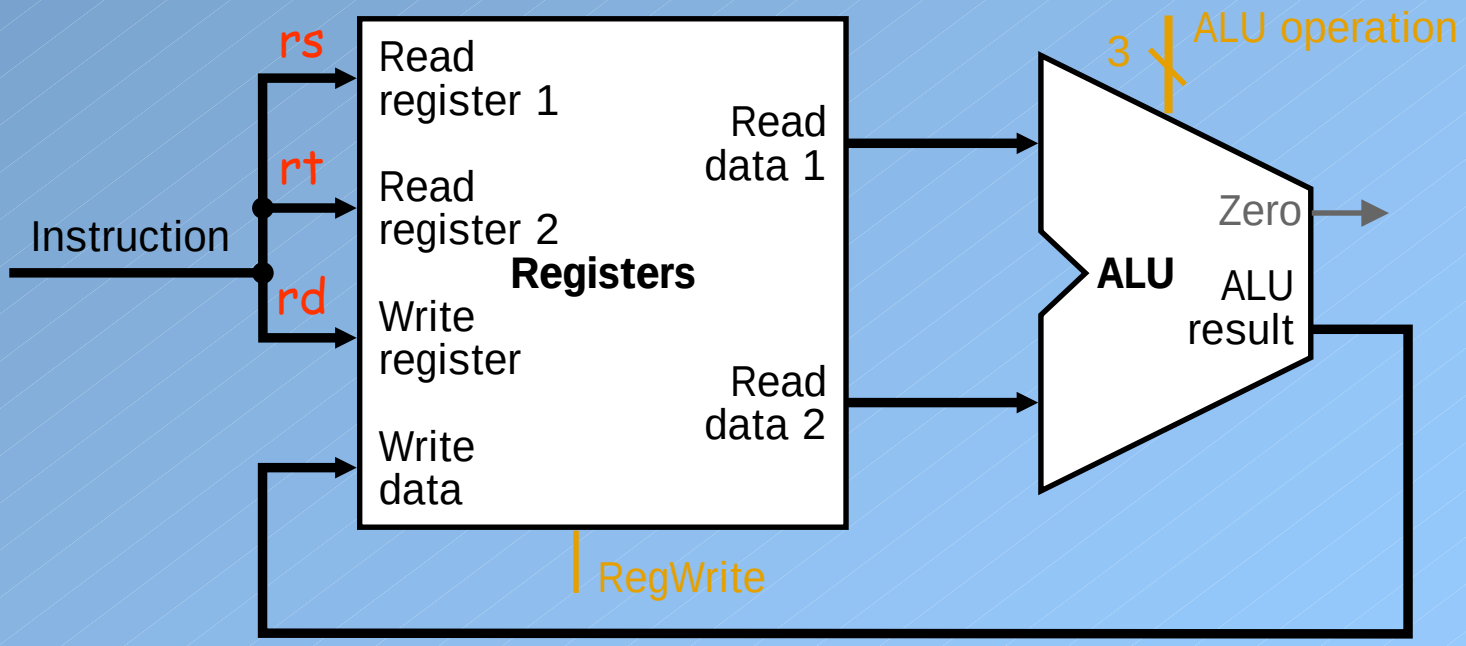
Implementing Instruction Fetch



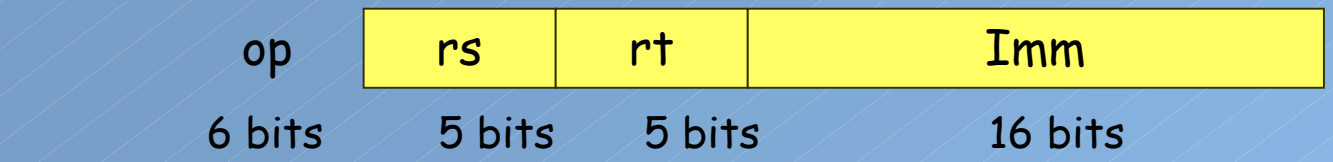
Implementing R-Type ALU Operations



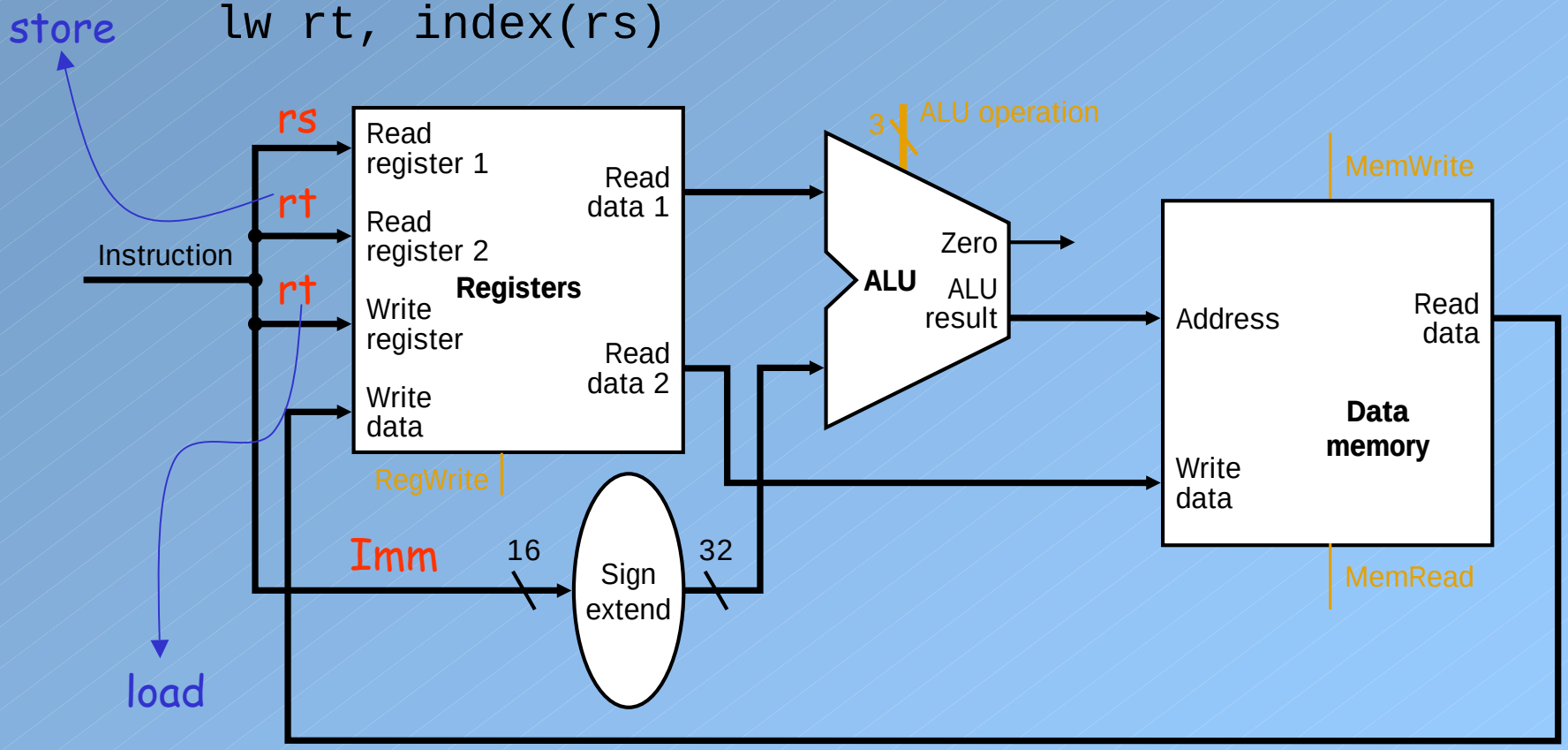

```
add rd, rs, rt
```



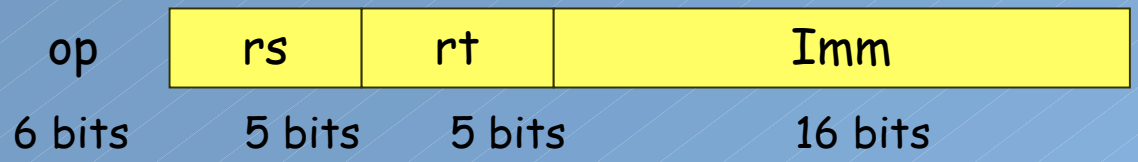
Implementing Load & Stores

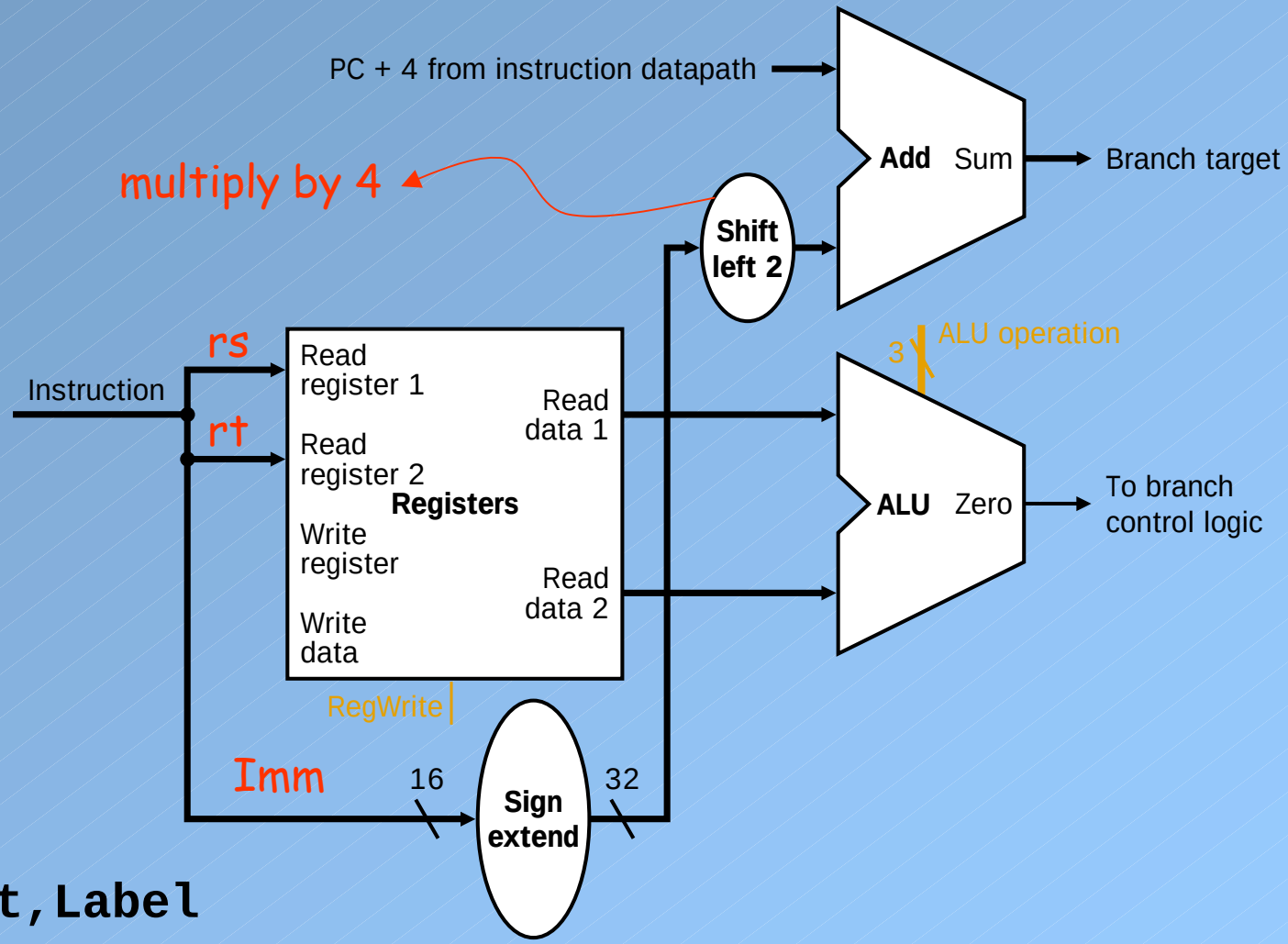


`lw rt, index(rs)`



Implementing Branches

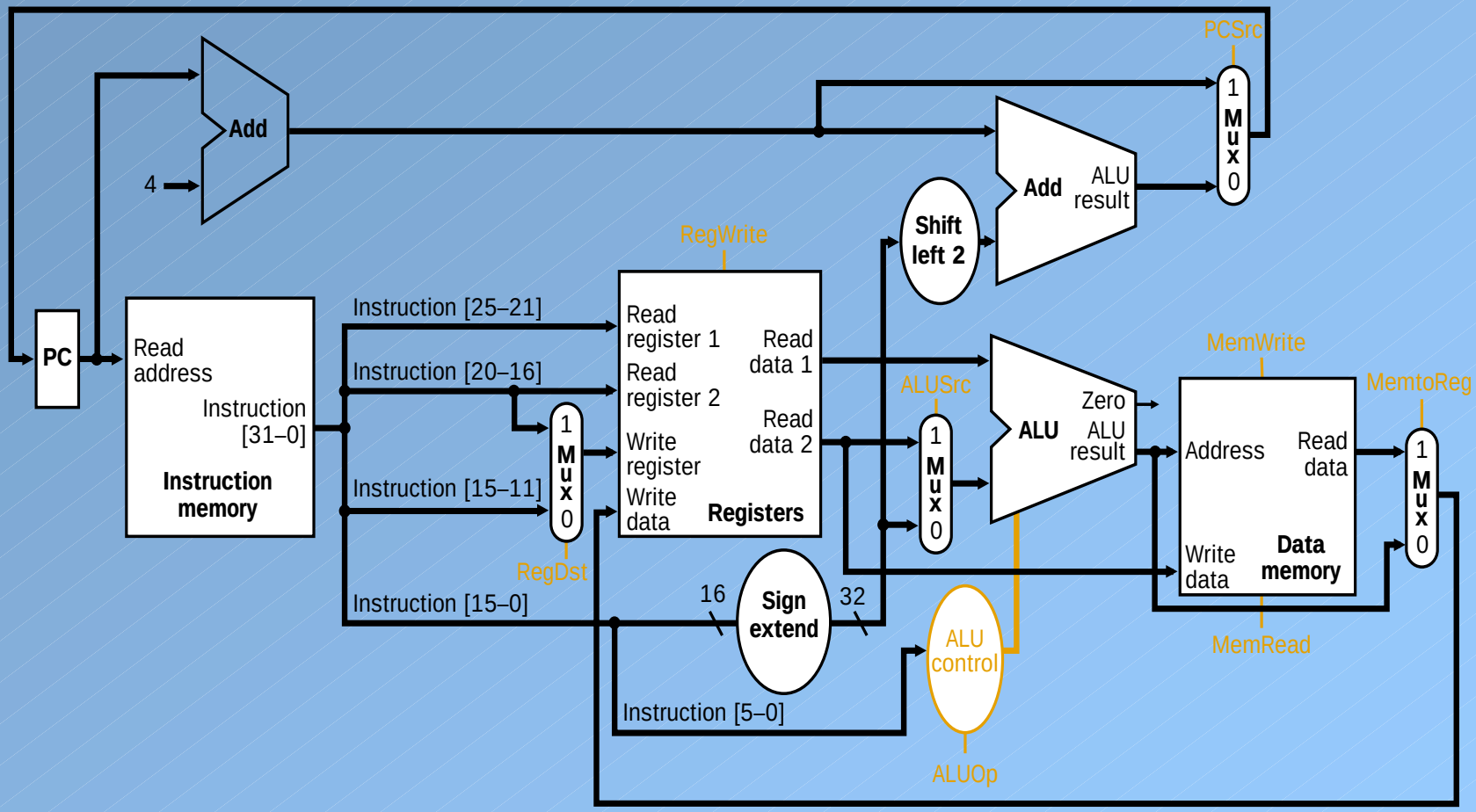




`beq rs, rt,Label`

Building the Datapath

Idea: Use multiplexors to stitch them together



Control

- Control Signals
  - Selecting the operations to perform (ALU, load/store, beq, etc.)
  - Controlling the flow of data (multiplexor's select inputs)
  - Read/write enable inputs of memory and register file
- Information comes from the 32 bits of the instruction

ALU Control Unit

- ALU performs
  - addition for loads and stores
  - subtraction for branches (beq)
  - no operation for jumps
  - or the operation is determined by the function field for R-type of information.
- ALU Control unit will have the following inputs:
  - two-bit control field called ALUOp
  - and Function field

ALU Control Unit

Instruction opcode	ALUop	Instruction operation	Funct field	Desired ALU action	ALU control output
lw	00	Load word	xxxxxx	add	010
sw	00	Store word	xxxxxx	add	010
beq	01	Branch equal	xxxxxx	subtract	110
R-type	10	Add	100000	add	010
R-type	10	Subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	slt	101010	slt	111

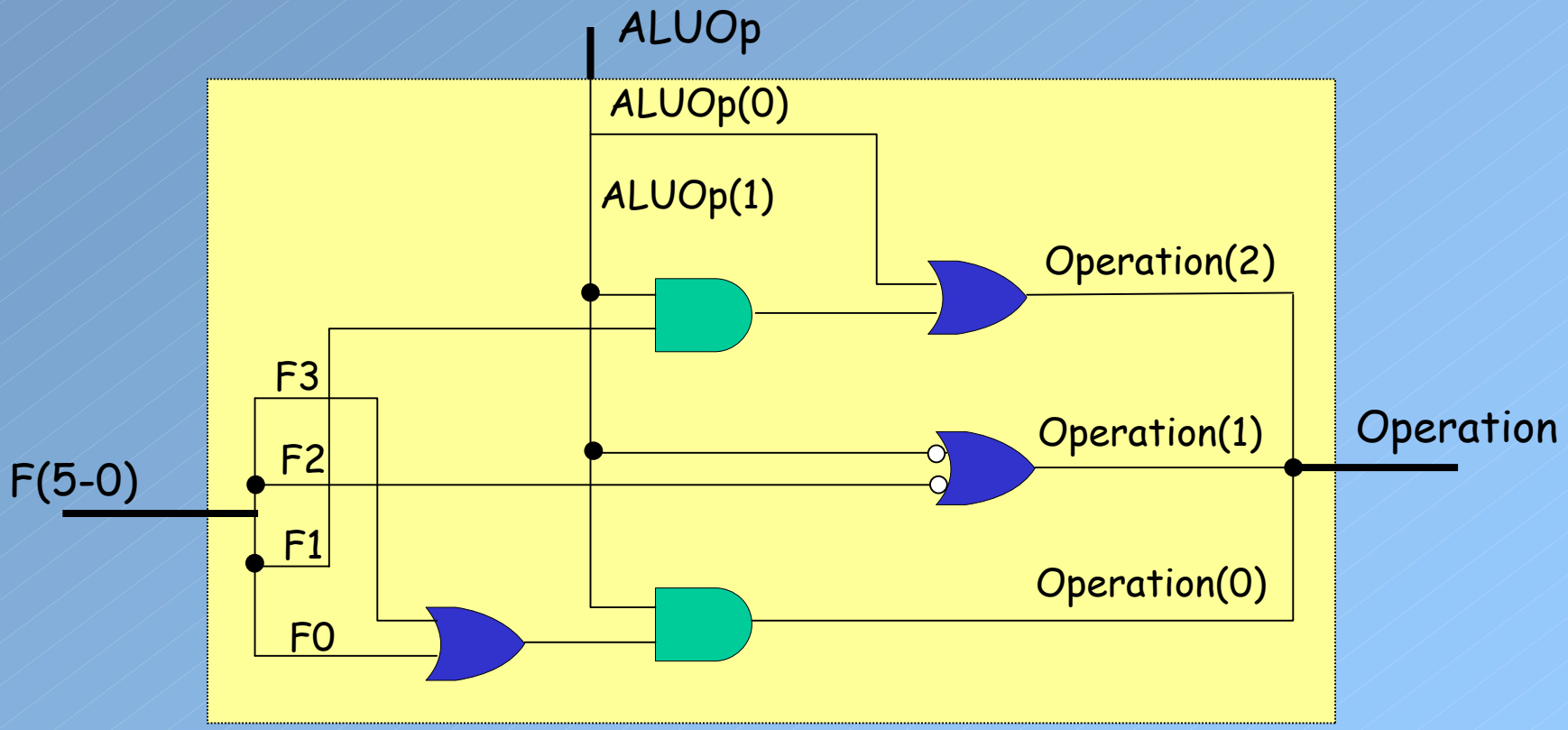
Truth Table for ALU Control Unit

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

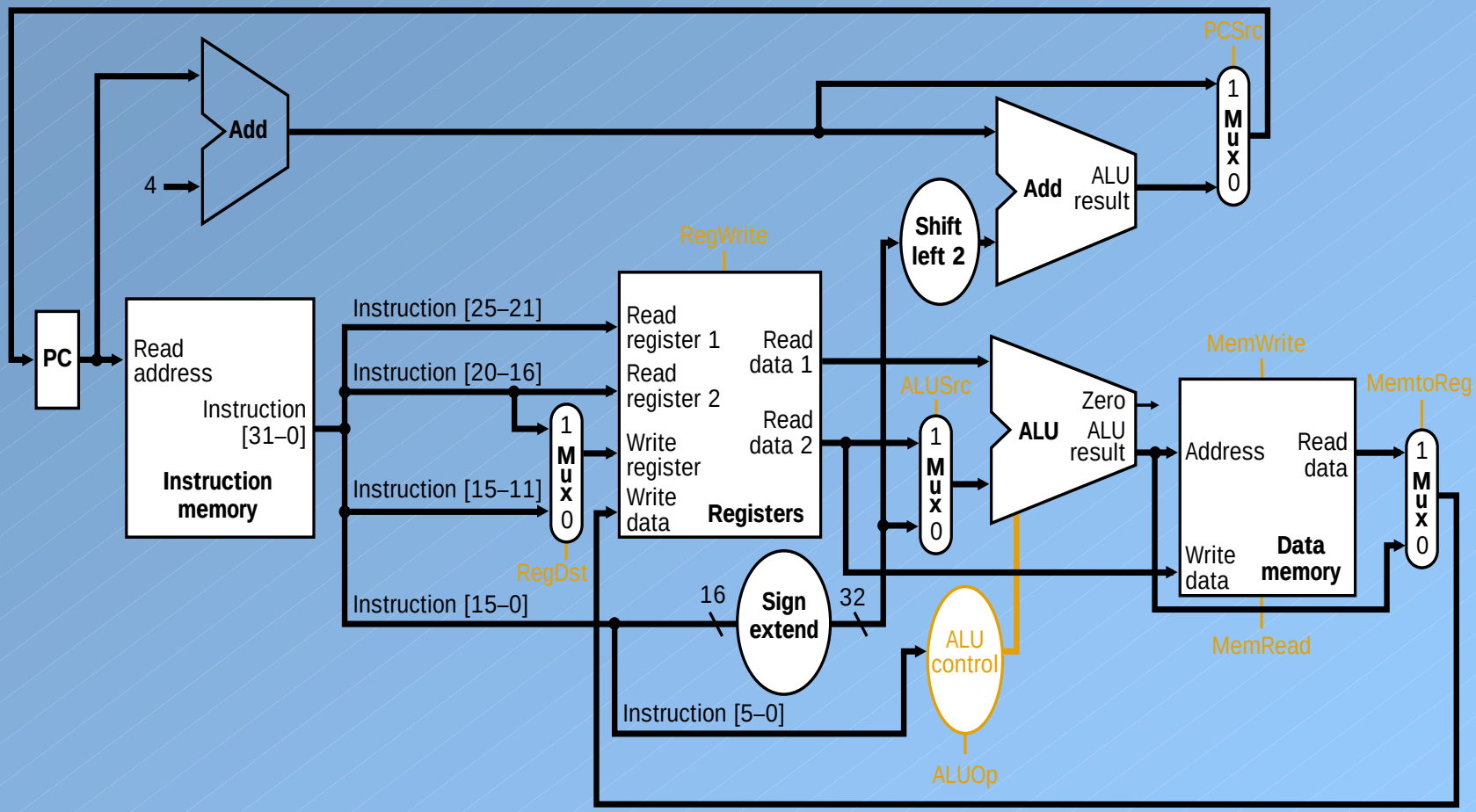
You can map the truth table to gates

Implementing the Control

- ALU Control Block



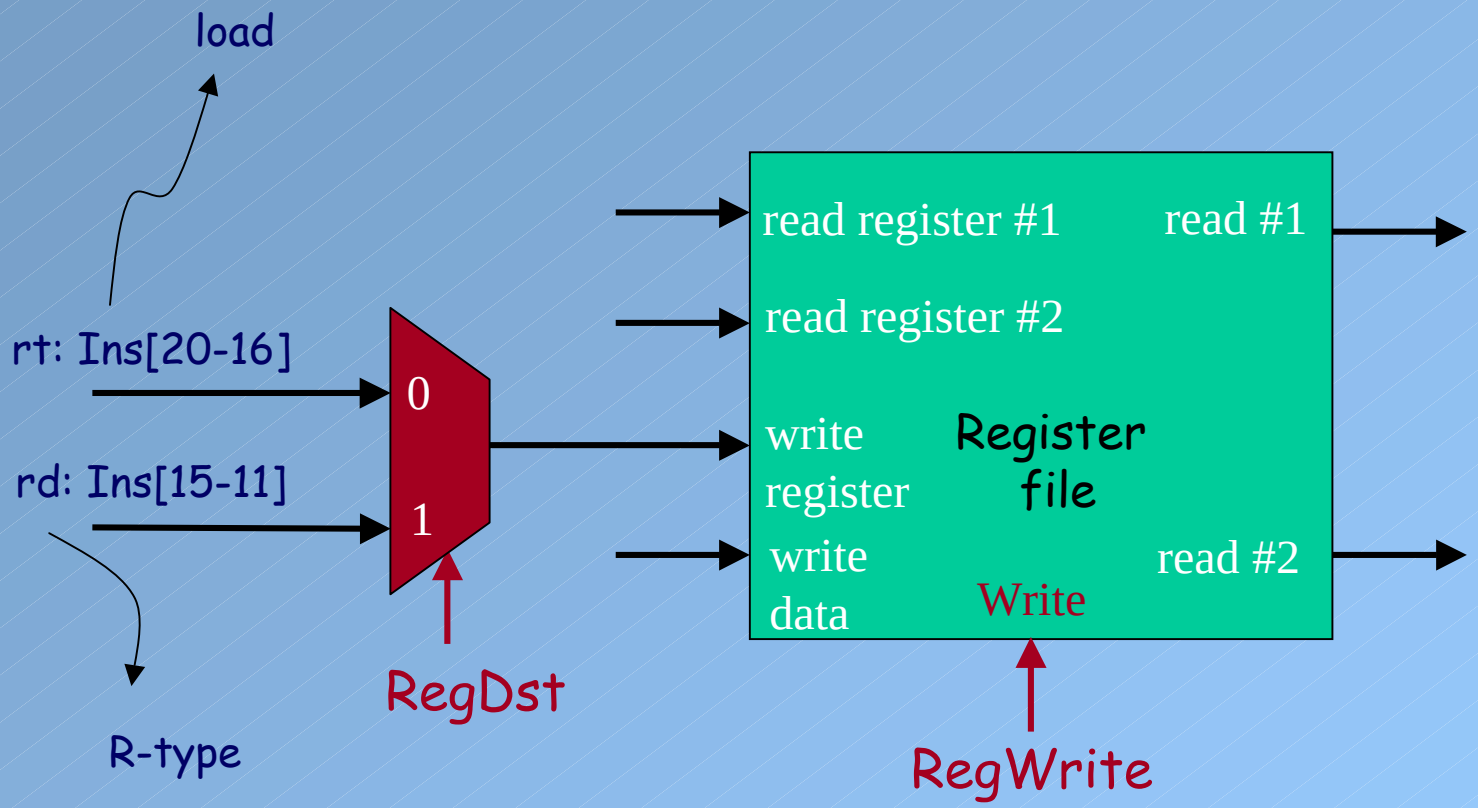
Datapath with the ALU Control Unit



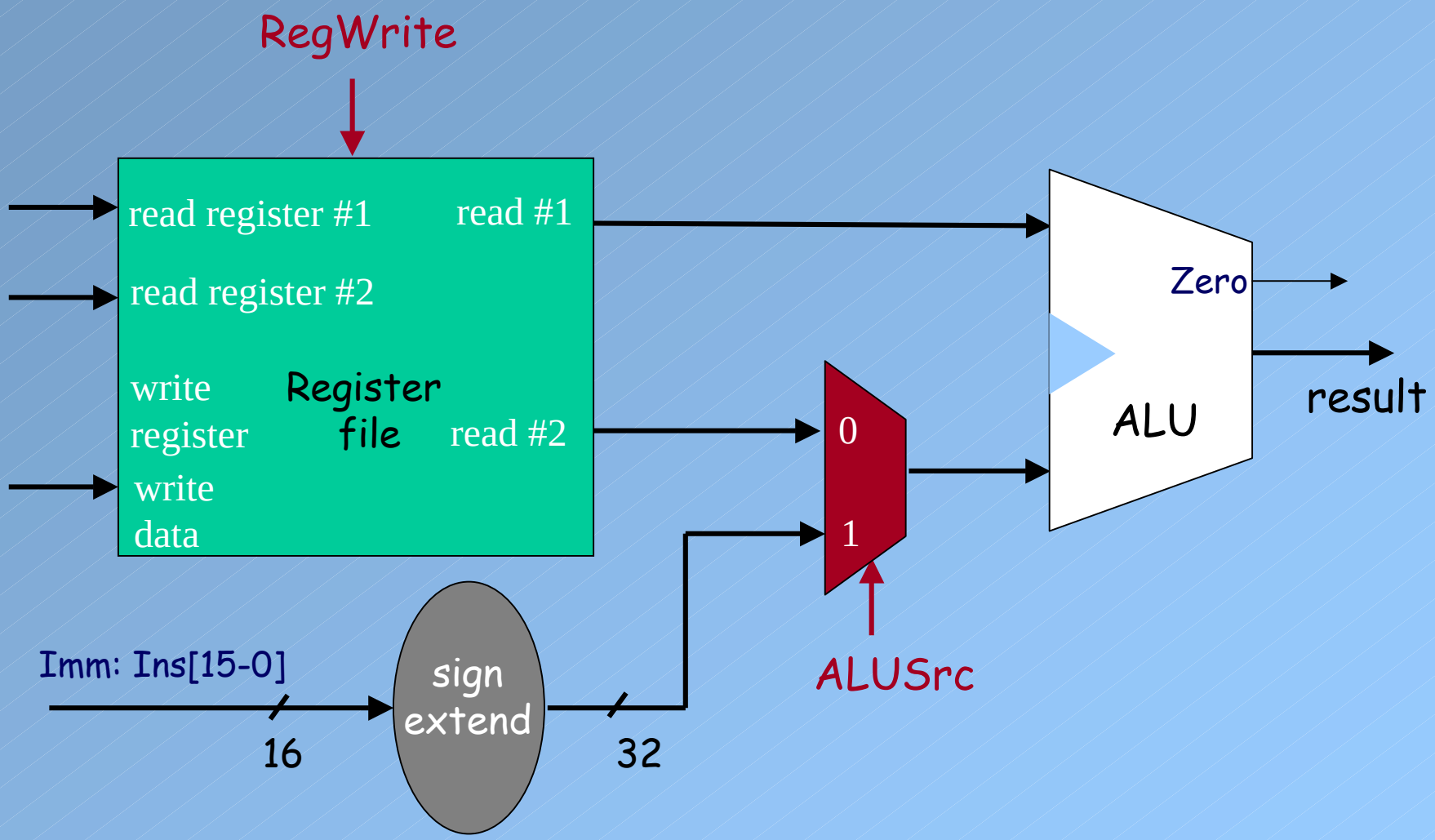
Seven Control Signals

Signal name	Effect when de-asserted	Effect when asserted
RegDst	The destination register no comes from `rt`.	The destination register no comes from `rd`.
RegWrite	None	Destination register is written with value on Writedata
ALUSrc	2nd ALU operand comes from `Read_Data_2`	2nd ALU operand is the sign extended, lower 16 bit of the instruction
PCSrc	The PC is replaced by PC + 4	The PC is replaced by the branch target address
MemRead	None	Memory is read
MemWrite	None	Memory is written
MemtoReg	The value to the register `Writedata` input comes from the ALU.	The value to the register `Writedata` input comes from the data memory

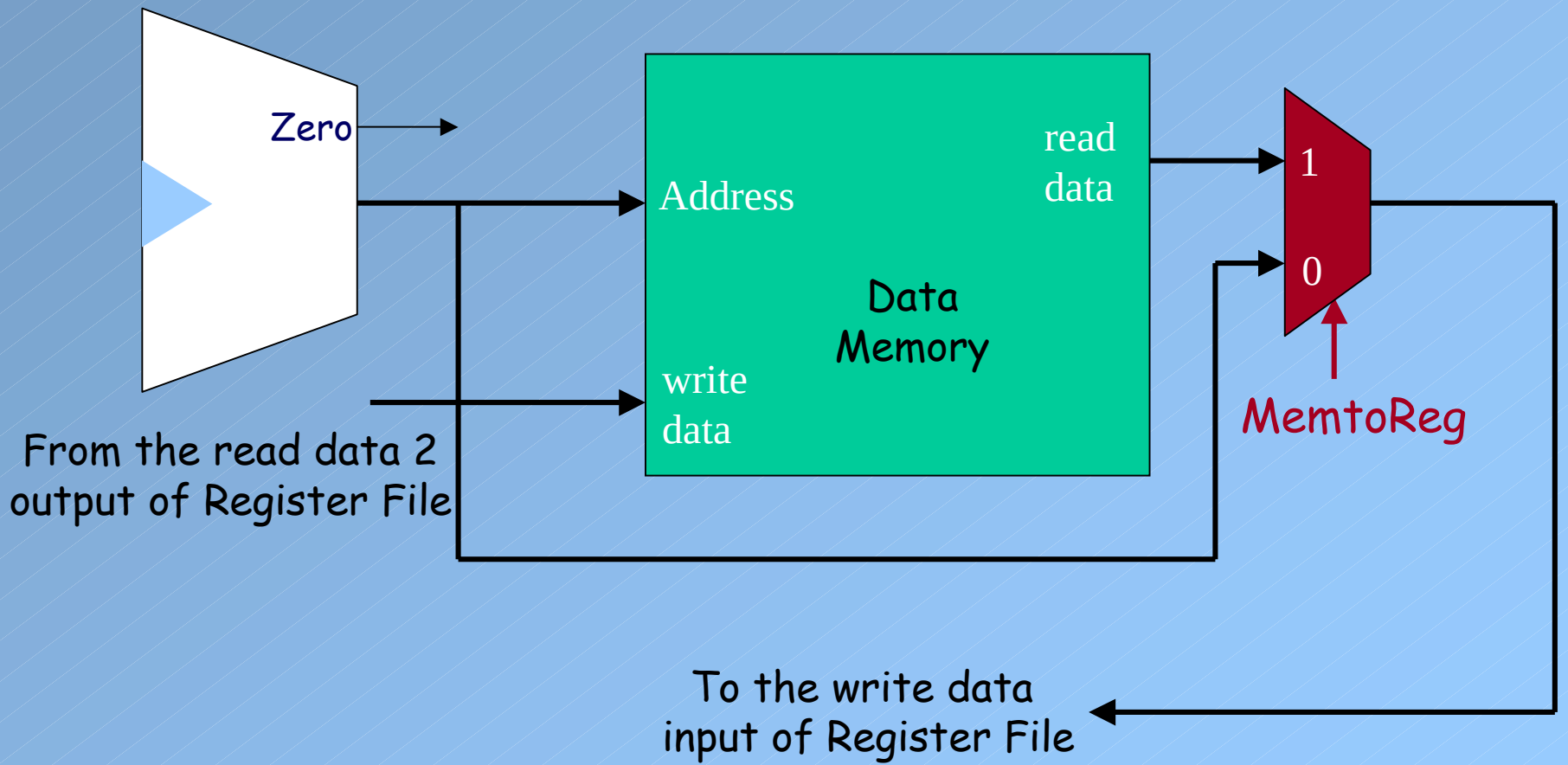
RegDst & RegWrite



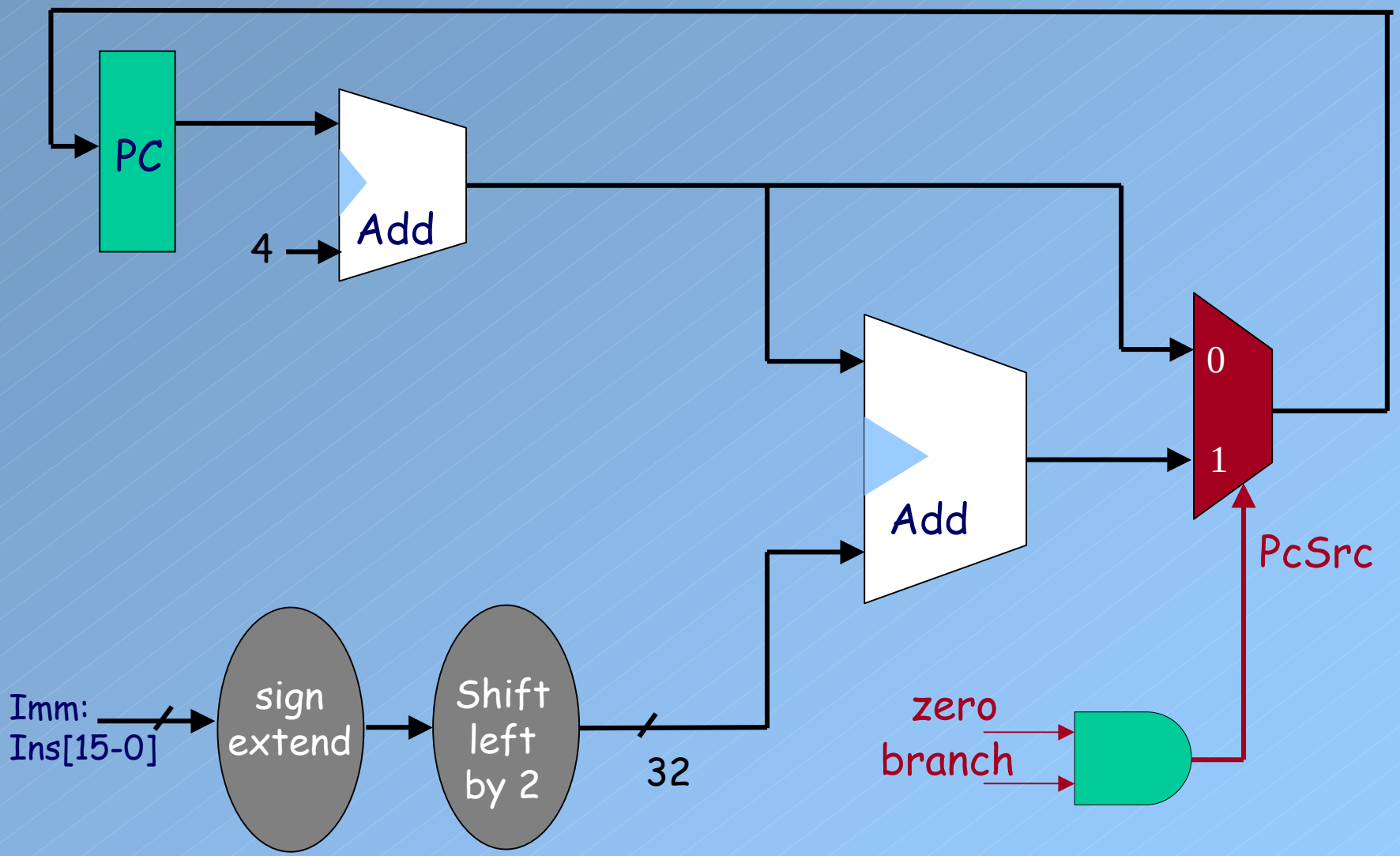
ALUSrc


RegWrite
read register #1
read #1
read register #2
write register
Register file
read #2
write data
0
1
ALU
Zero
result
ALUSrc
Imm: Ins[15-0]
16
sign extend
32

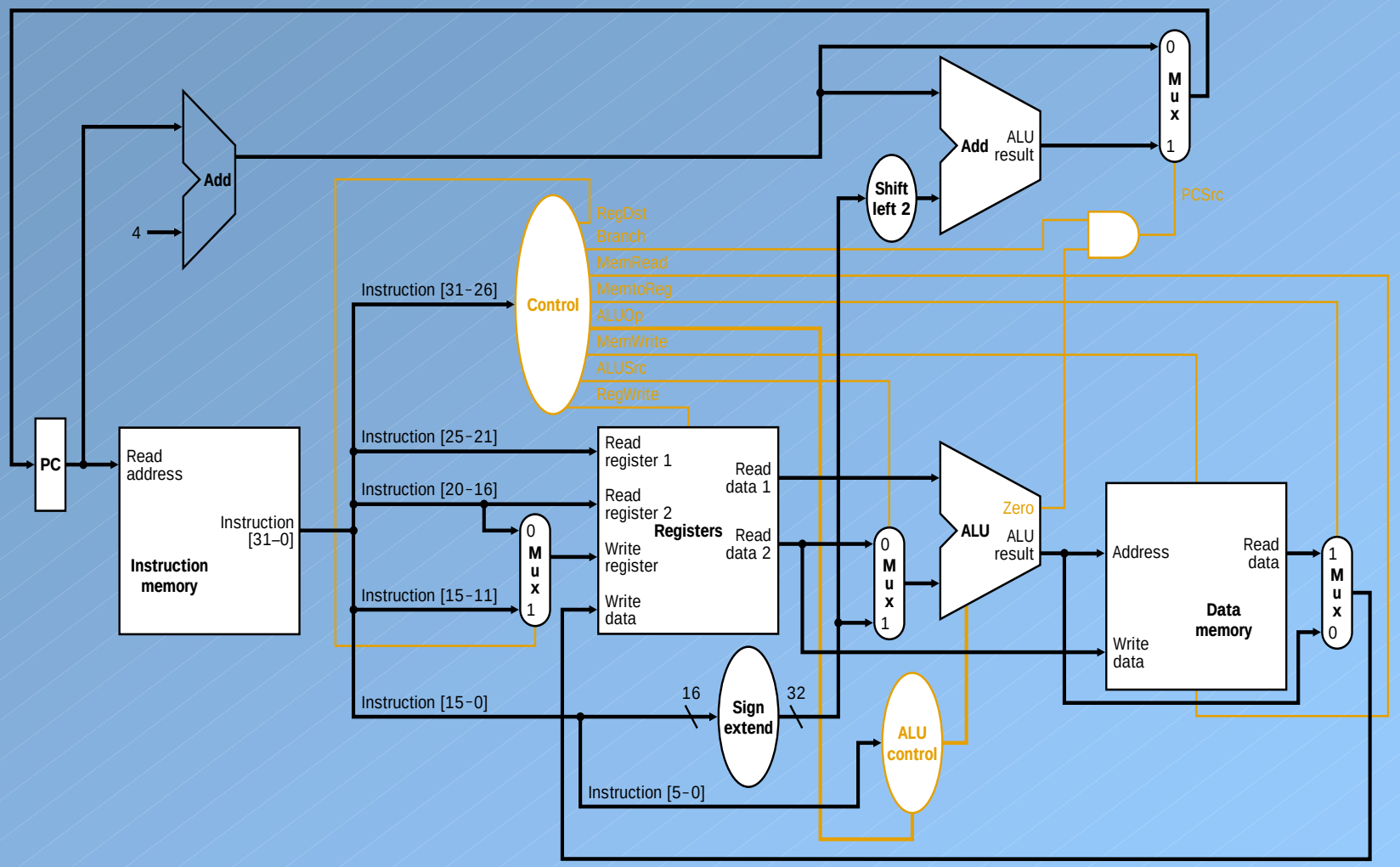
MemtoReg


Zero
Address
read
data
Data
Memory
write
data
1
0
MemtoReg
From the read data 2
output of Register File
To the write data
input of Register File

PCSrc


PC
Add
4
Add
0
1
PcSrc
Imm:
Ins[15-0]
sign
extend
Shift
left
by 2
32
zero
branch

Datapath & Control



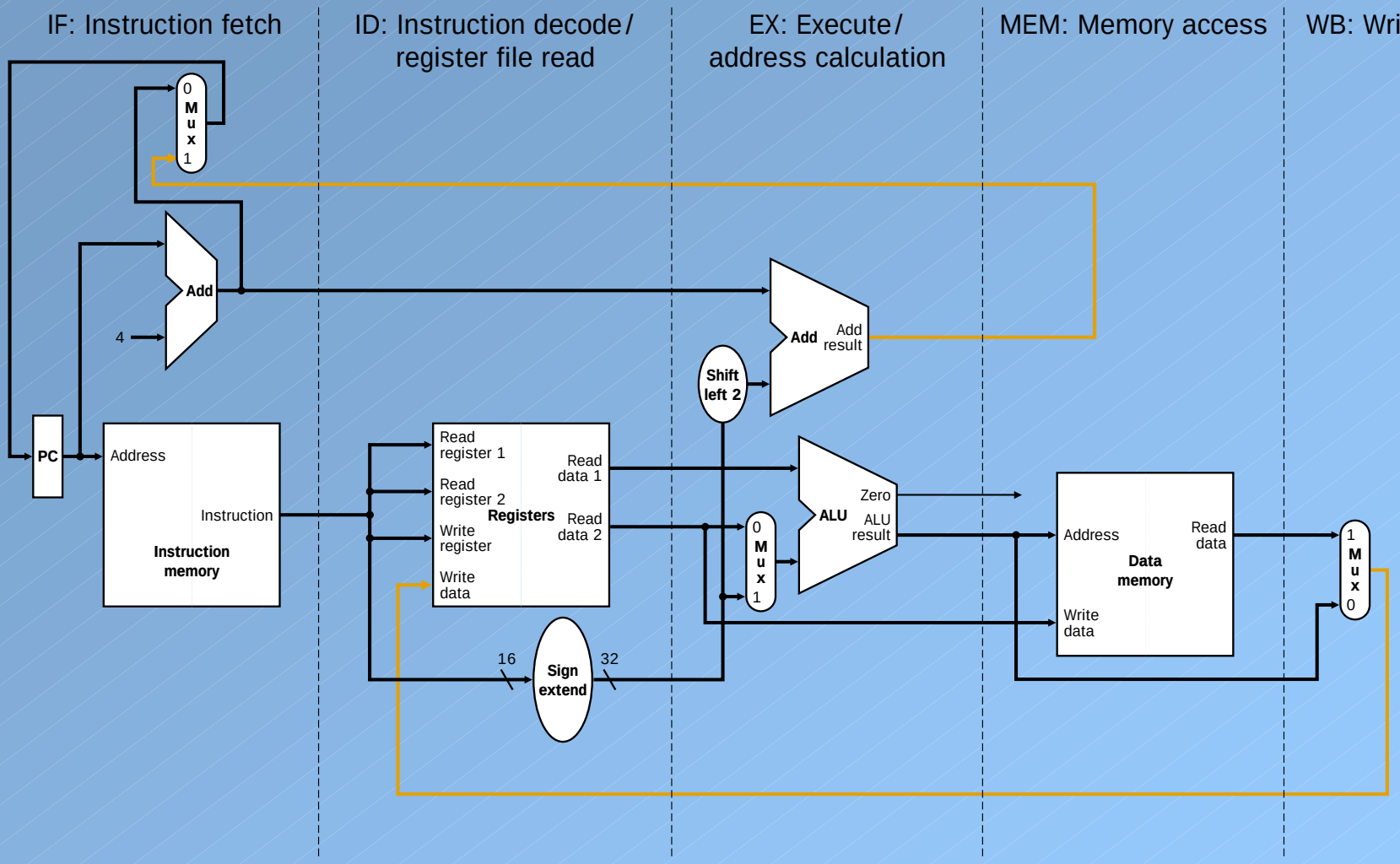
Operation of the Datapath

Instruction	RegDest	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0
R-format	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

Example Flow: beq $s0, $s1, address

- The instruction is fetched from memory and PC is incremented
- Read two register values
- Subtract one from the other, calculate the branch address
- Use the zero signal to determine which of the addresses is to be used for fetching the next instruction

Single Cycle Datapath



The Stages of an Instruction

- MIPS instructions classically take five steps:

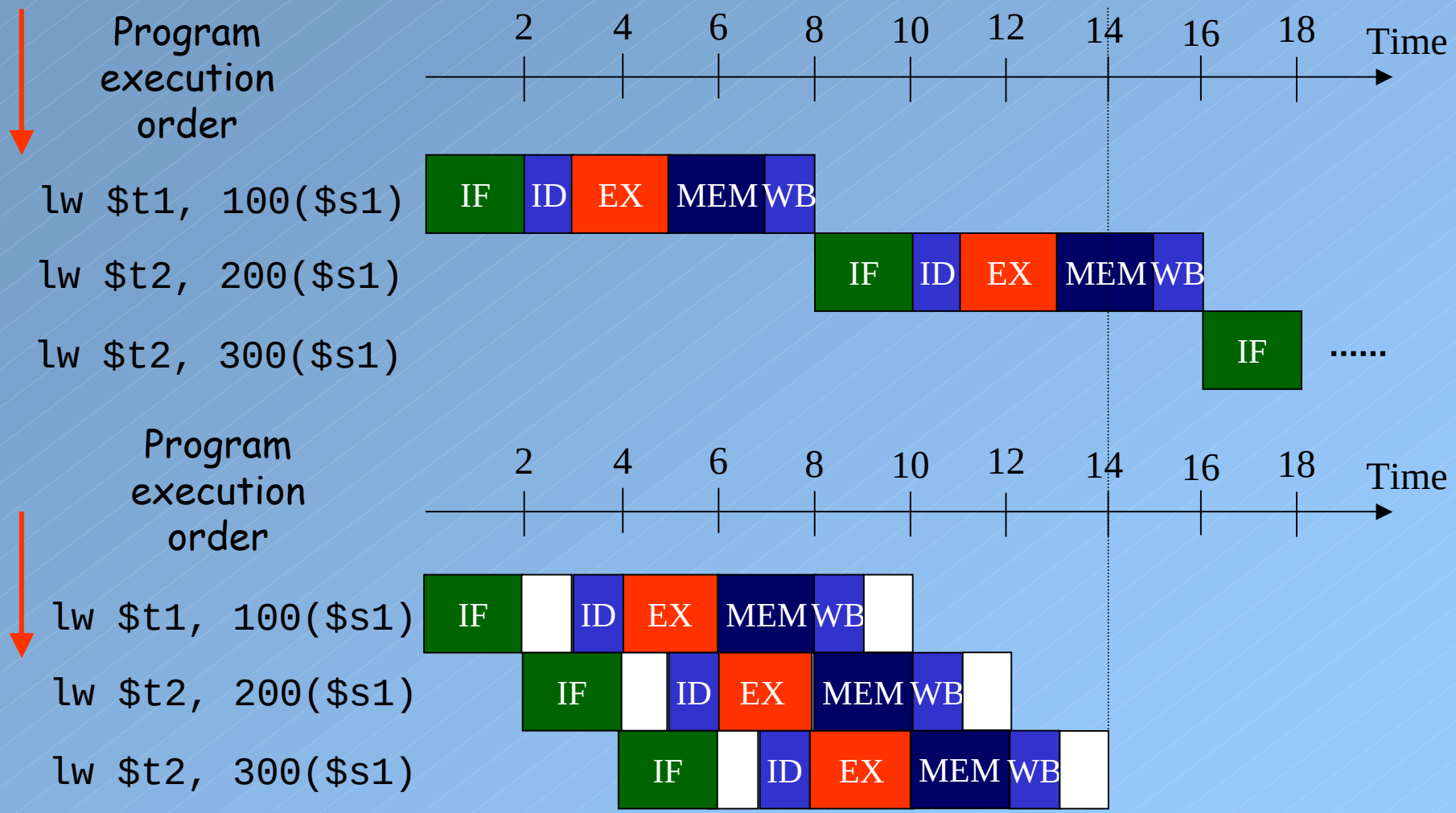
1. Fetch instruction from memory (**IF**)
2. Read registers while decoding the instruction (**ID**)
3. Execute the operation or calculate an address (**EX**)
4. Access an operand in memory (**MEM**)
5. Write result into a register (**WB**)

Times of Different Instruction Steps

Instruction class	Inst. Fetch	Register Read	ALU op	Data access	Register write	Total time
lw	2 ns	1ns	2 ns	2 ns	1 ns	8 ns
sw	2 ns	1ns	2 ns	2 ns		7 ns
R-format	2 ns	1ns	2 ns		1 ns	6 ns
Branch	2 ns	1ns	2 ns			5 ns

Clock period that can be used in single-cycle implementation will be 8 ns for all instruction.

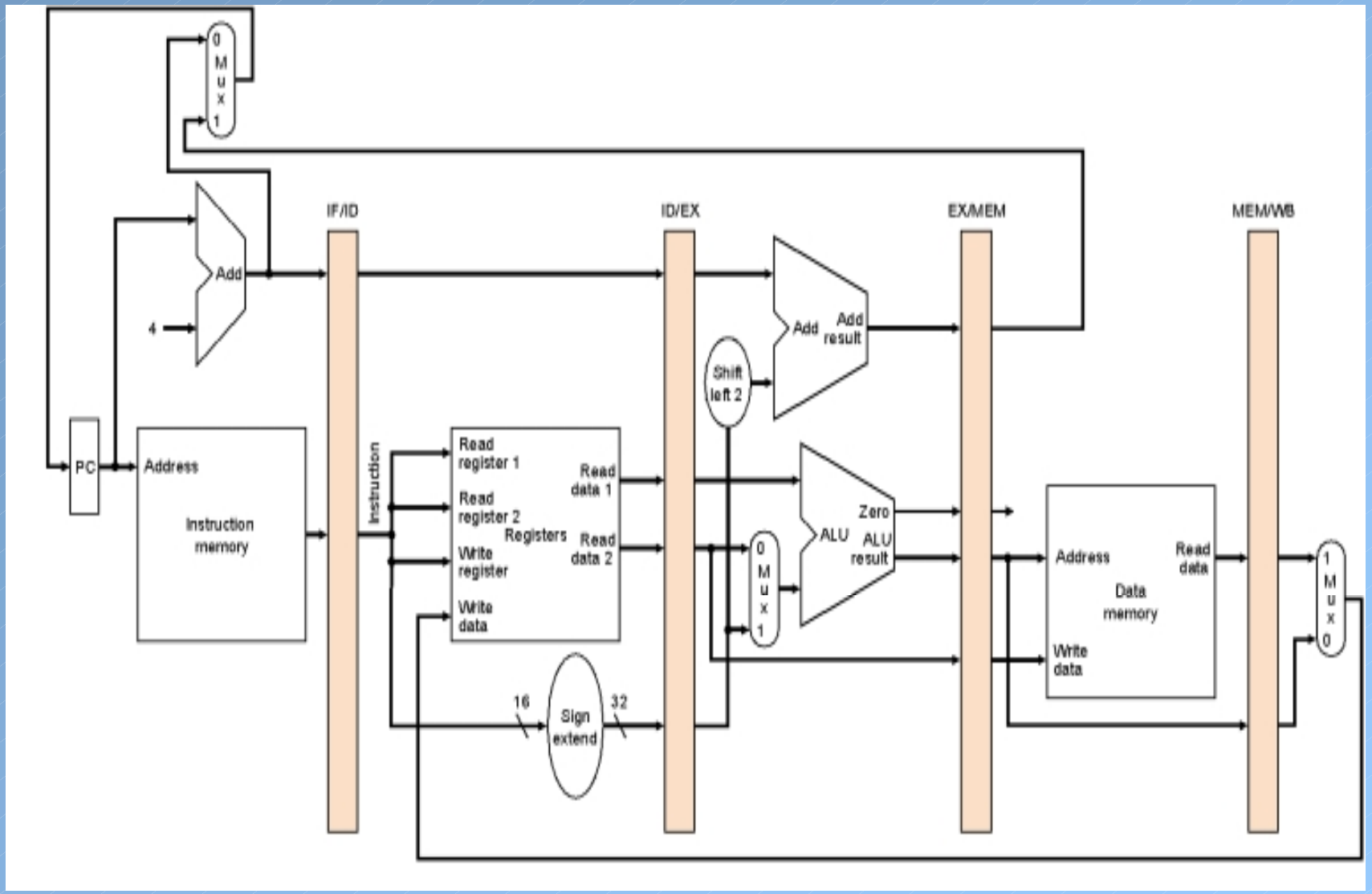
Pipelined vs. Nonpipelined



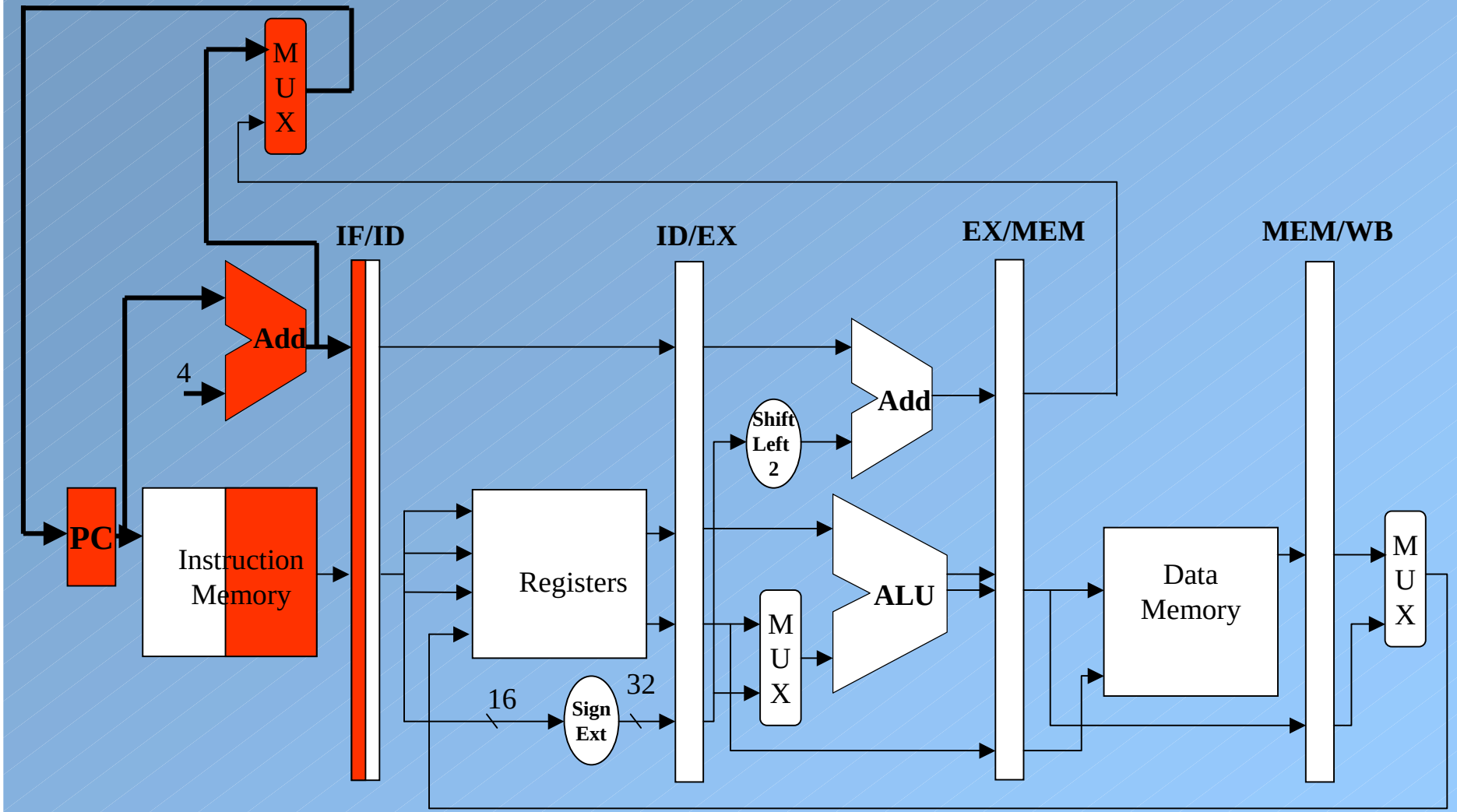
Design of Pipelined Datapath

- Break the datapath into smaller segments
- Portions of datapath can be shared by instructions
- Use registers between two consecutive segments of the datapath to hold the intermediate results.
- Pipeline registers
- Data transfer between the stages happens through the pipeline registers

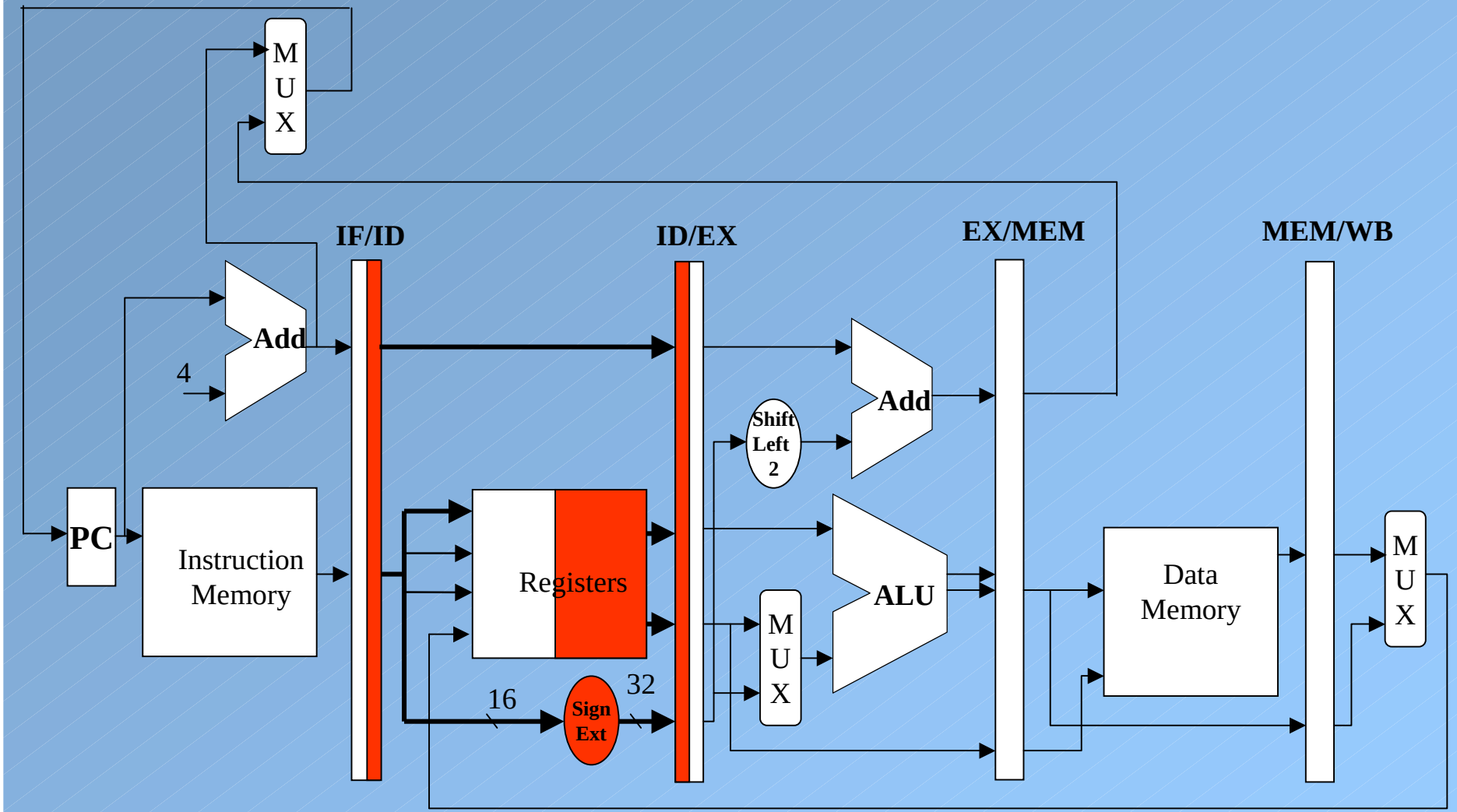
Pipelined Datapath



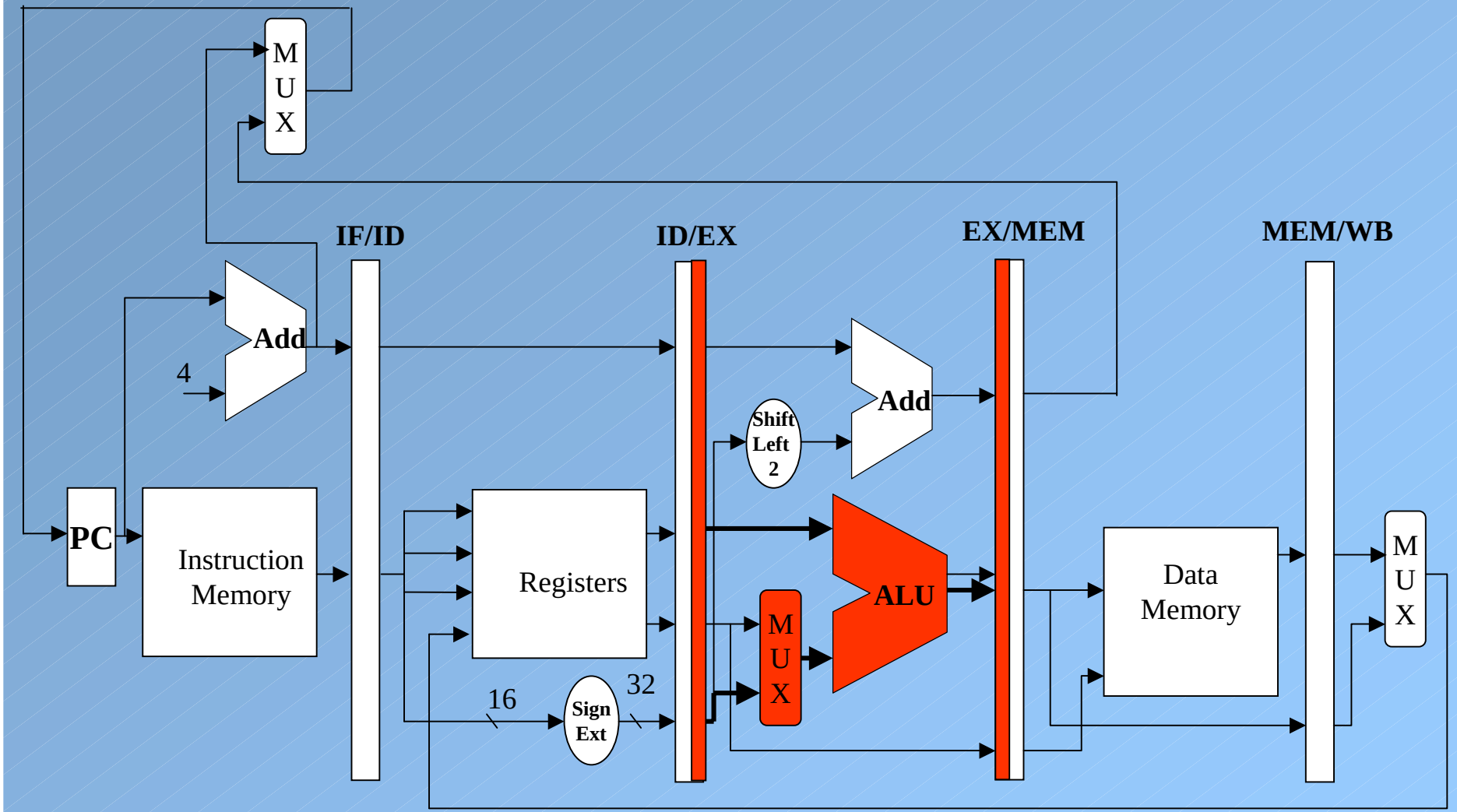
Example: IF of lw



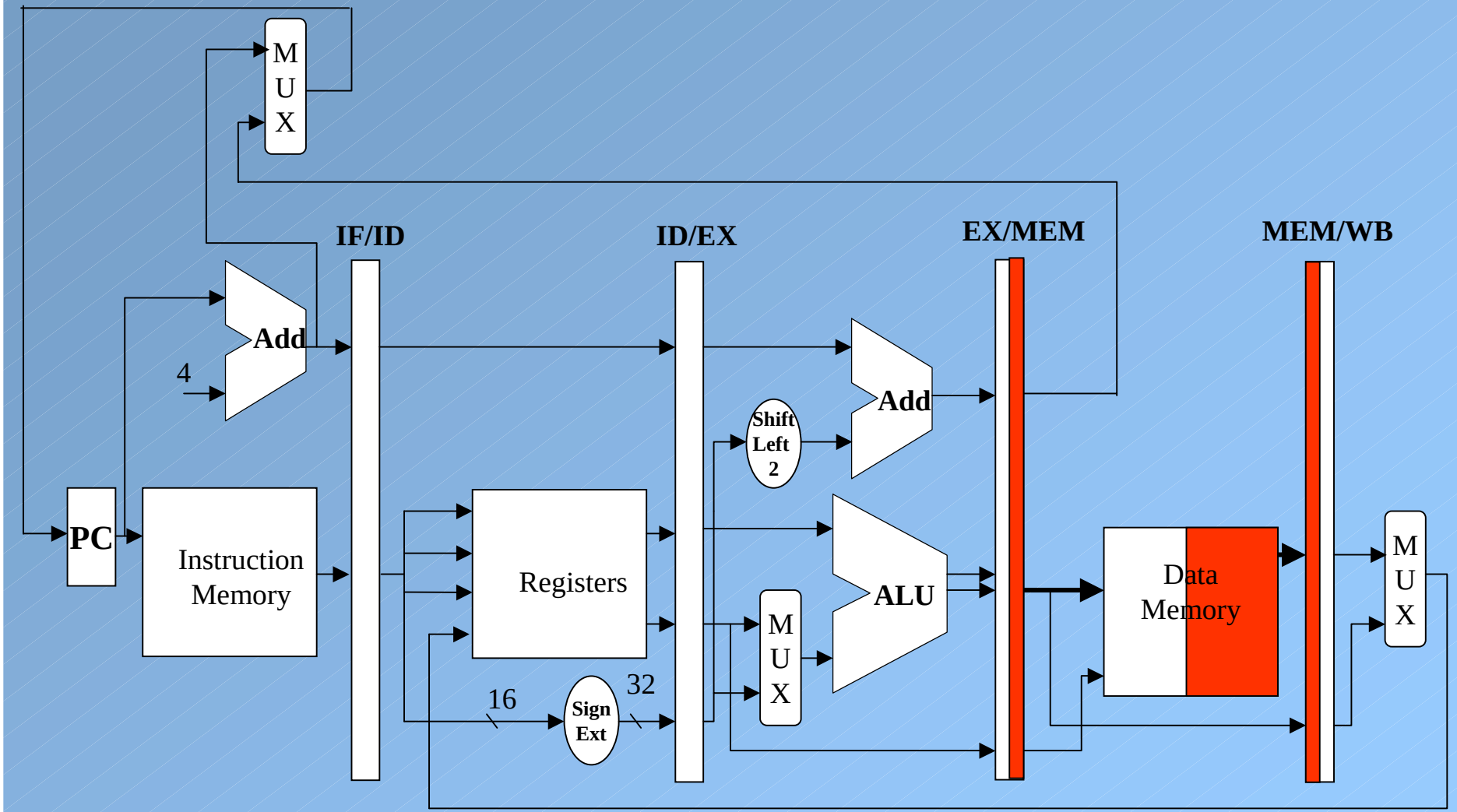
Example: ID of lw



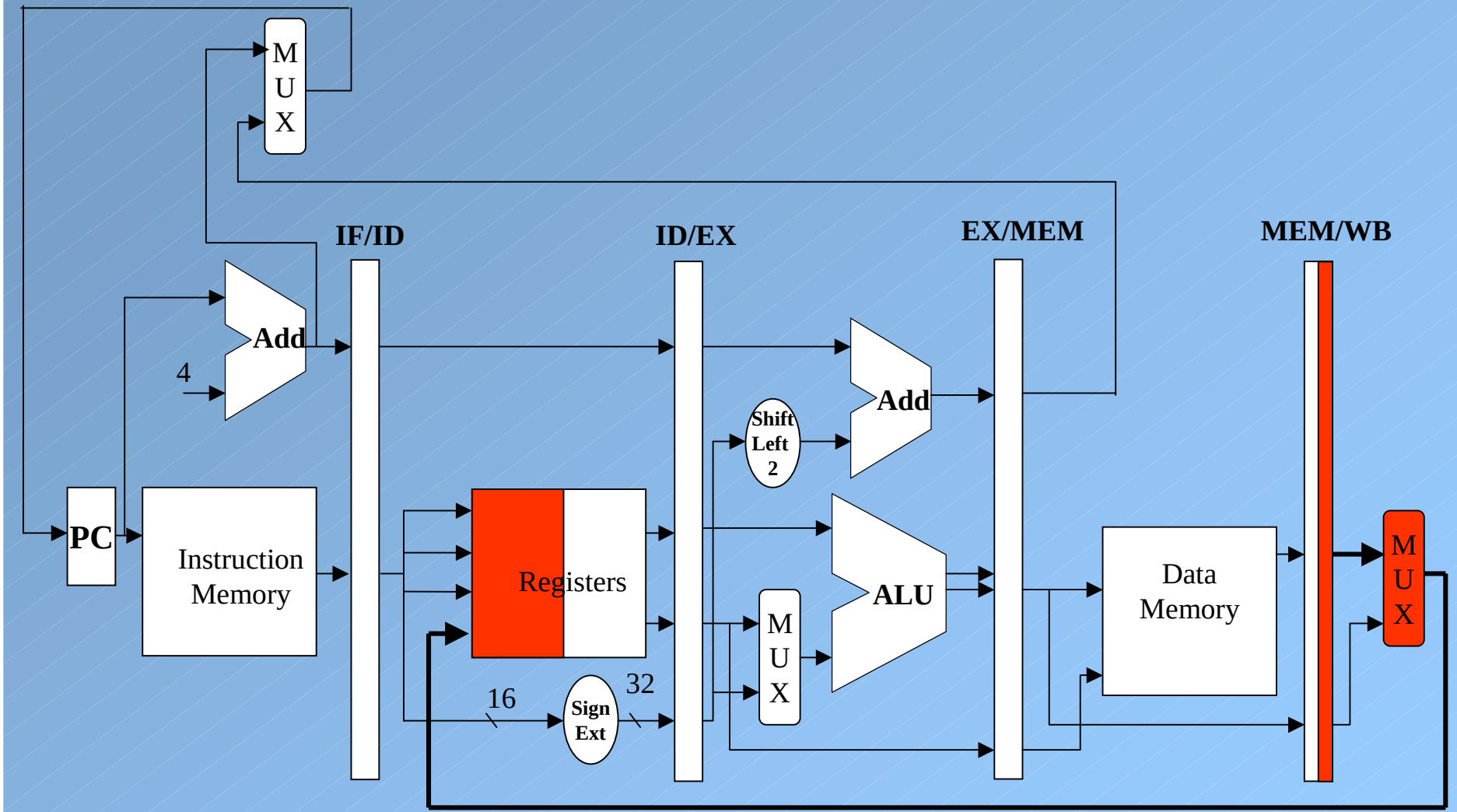
Example: EX of lw



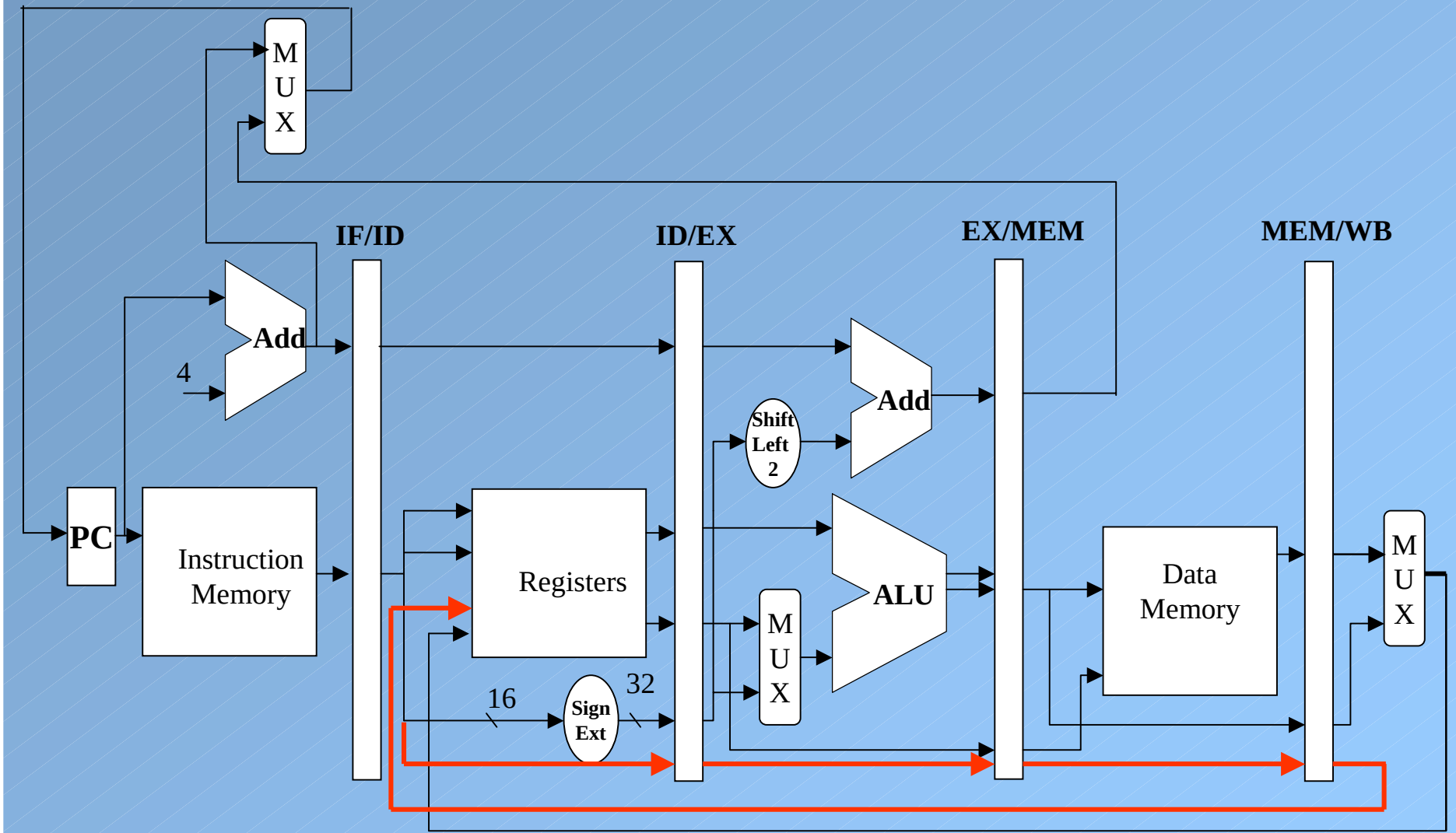
Example: MEM of lw



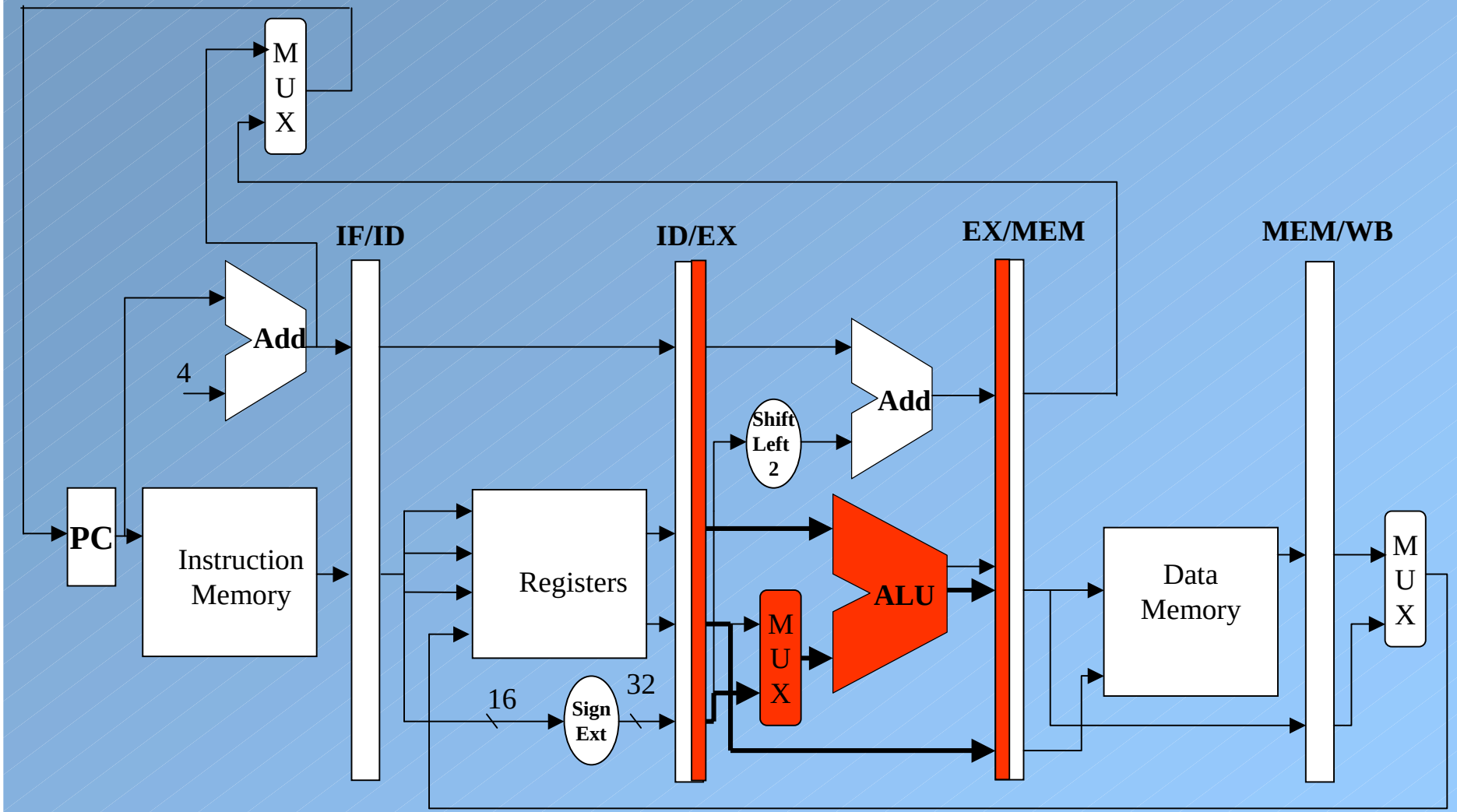
Example: WB of `lw`



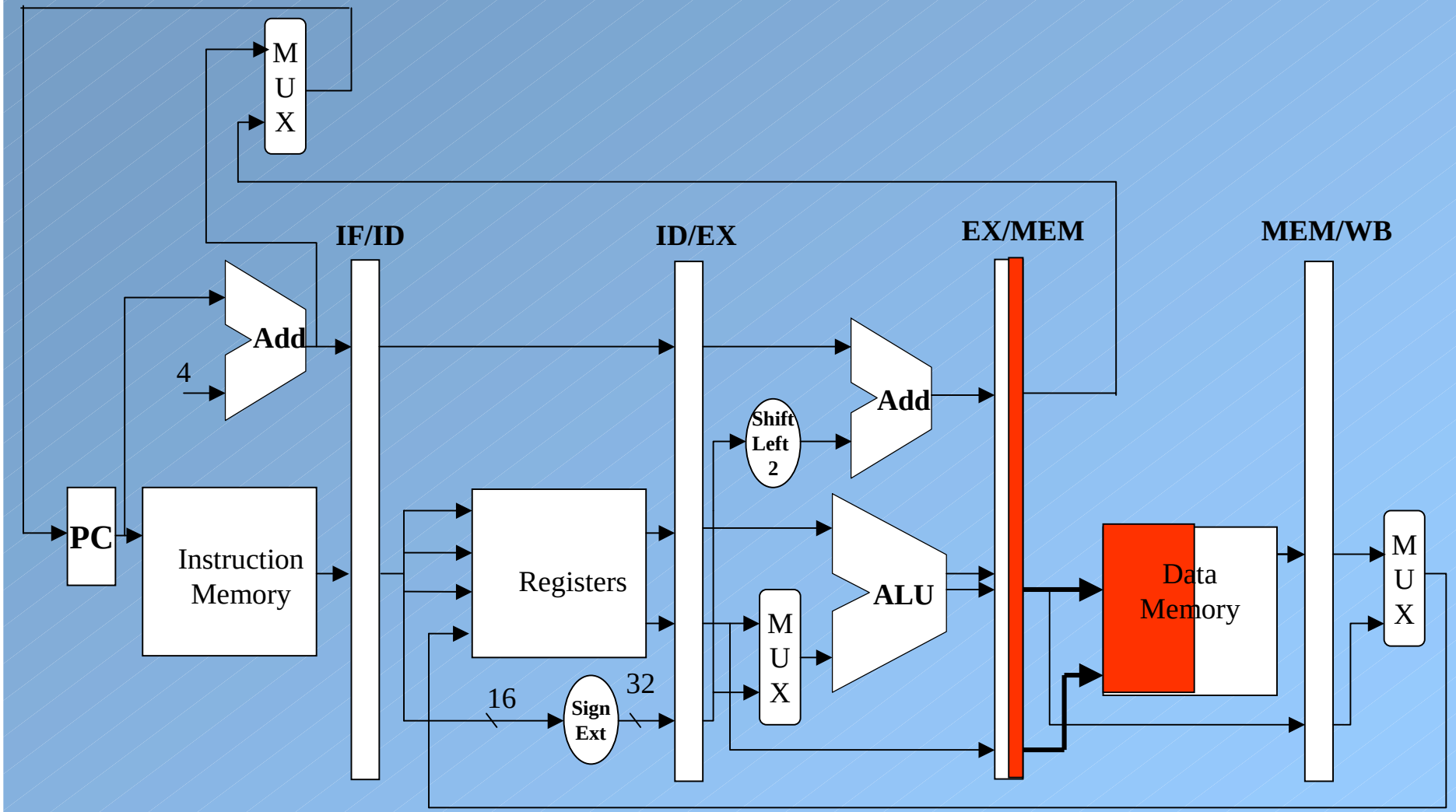
Corrected Datapath



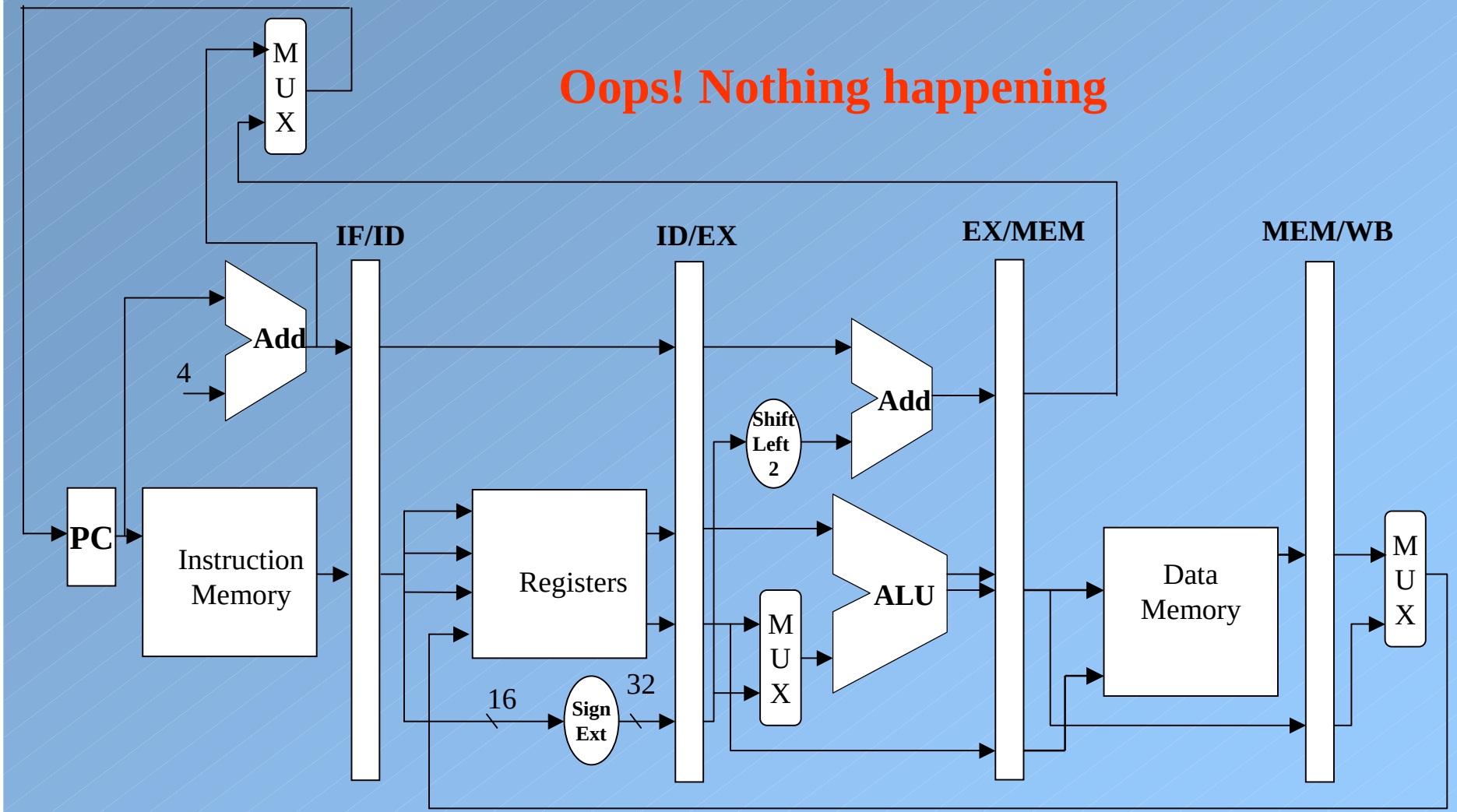
Example 2: Store word, EX stage



Example 2: MEM of sw



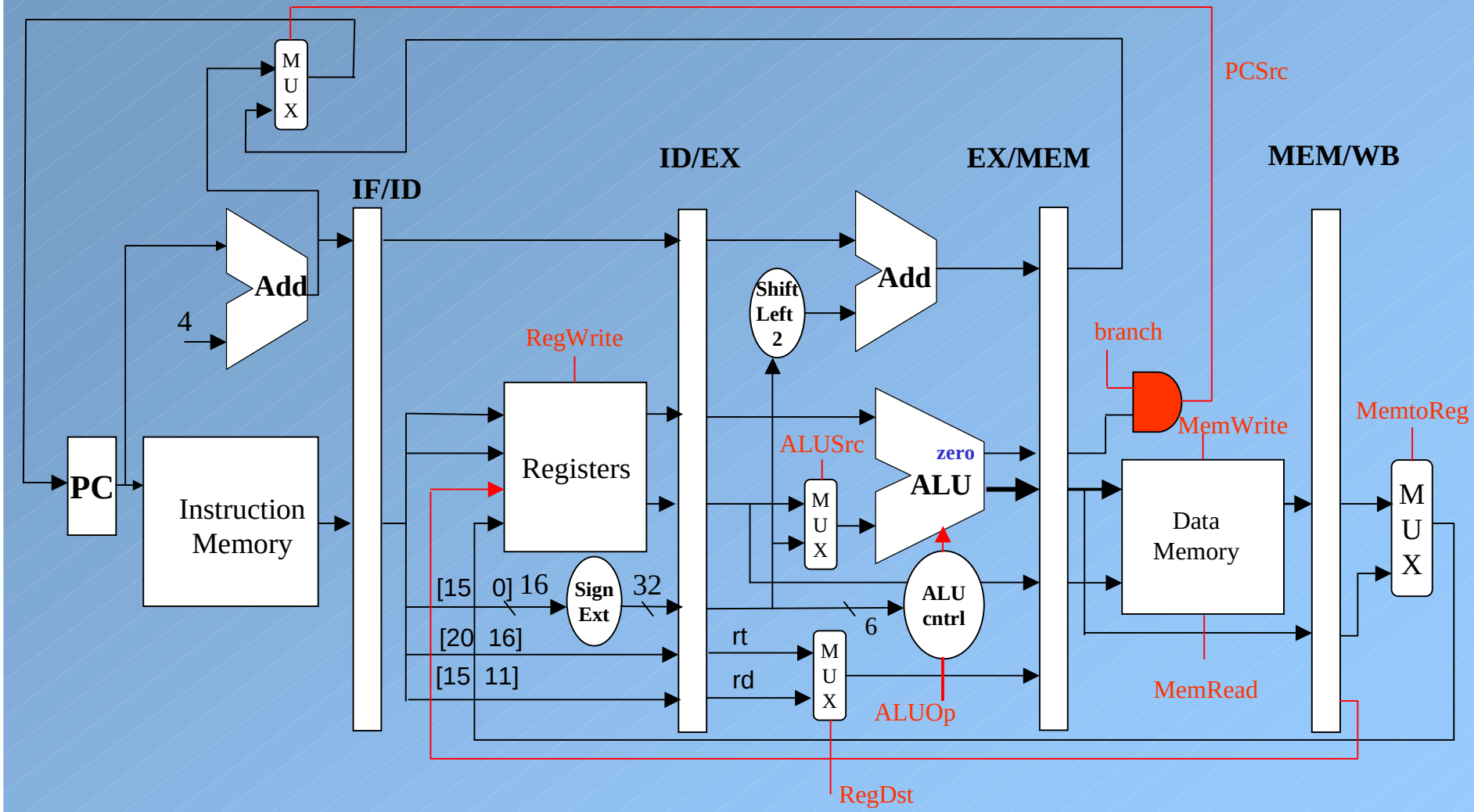
Example 2: WB of sw



Pipelined Control

- PC is written on each clock cycle (no write signal)
- No write signals for pipeline registers.
- IF stage: no control signal since instruction is read and PC is updated each cycle
- ID stage: No control signals
- EX stage: `RegDst, ALUOp, ALUSrc`
- MEM stage: `branch, MemRead, MemWrite`
- WB stage: `MemtoReg, RegWrite`

Pipelined Datapath with Control Signals



Reminder: Seven Control Signals

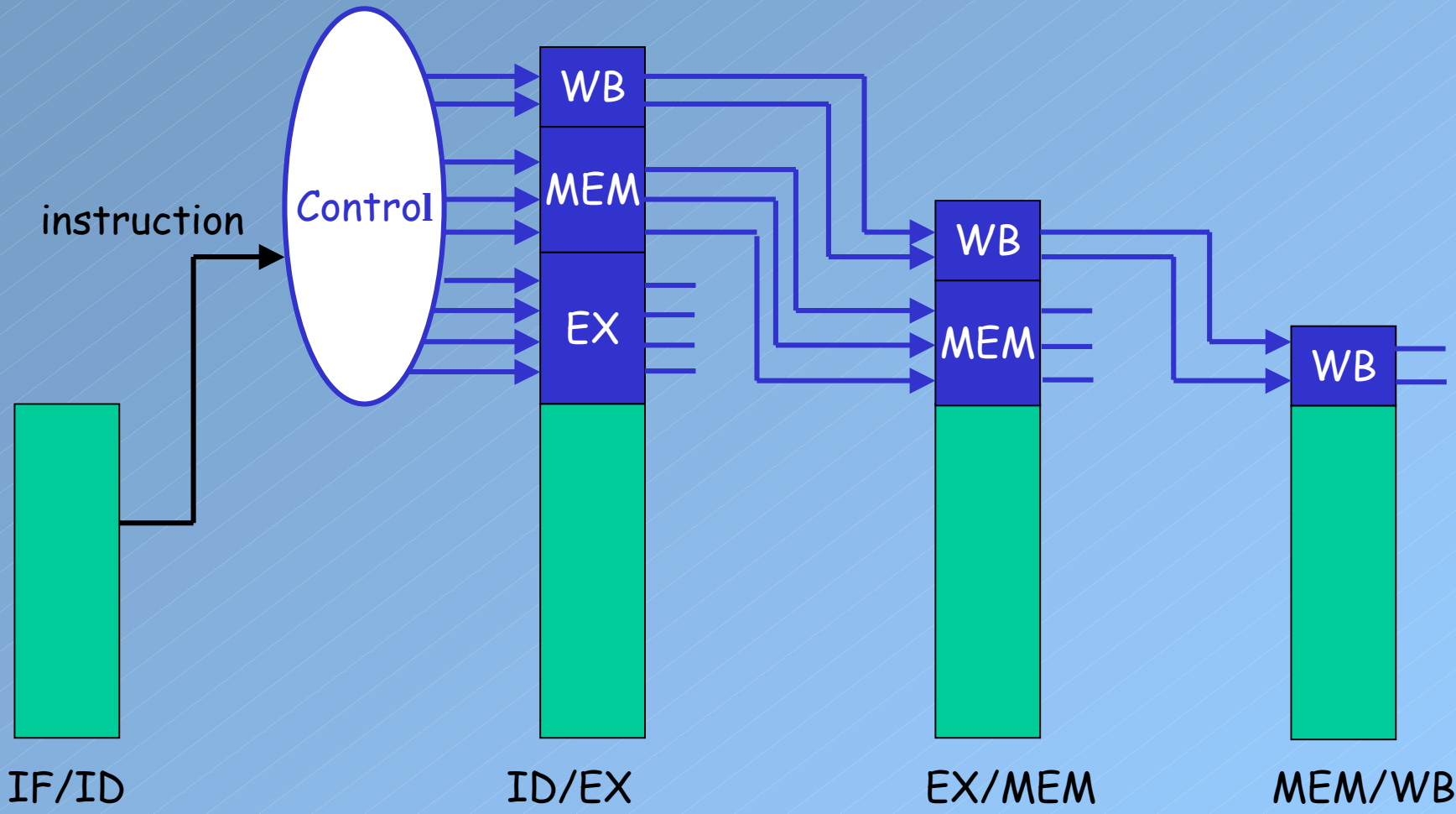
Signal name	Effect when deasserted	Effect when asserted
RegDst	The destination register number comes from rt.	The destination register number comes from rd.
RegWrite	None	Destination register is written with value on Writedata
ALUSrc	2nd ALU op comes from Read data 1 (i.e. rt)	2nd ALU op is the sign extended, lower 16 bit of the instruction
PCSrc	The PC is replaced by PC + 4	The PC is replaced by the branch target address
MemRead	None	Memory is read
MemWrite	None	Memory is written
MemtoReg	The value to the register Writedata input comes from the ALU.	The value to the register Writedata input comes from the data memory

Control Signals for Instructions

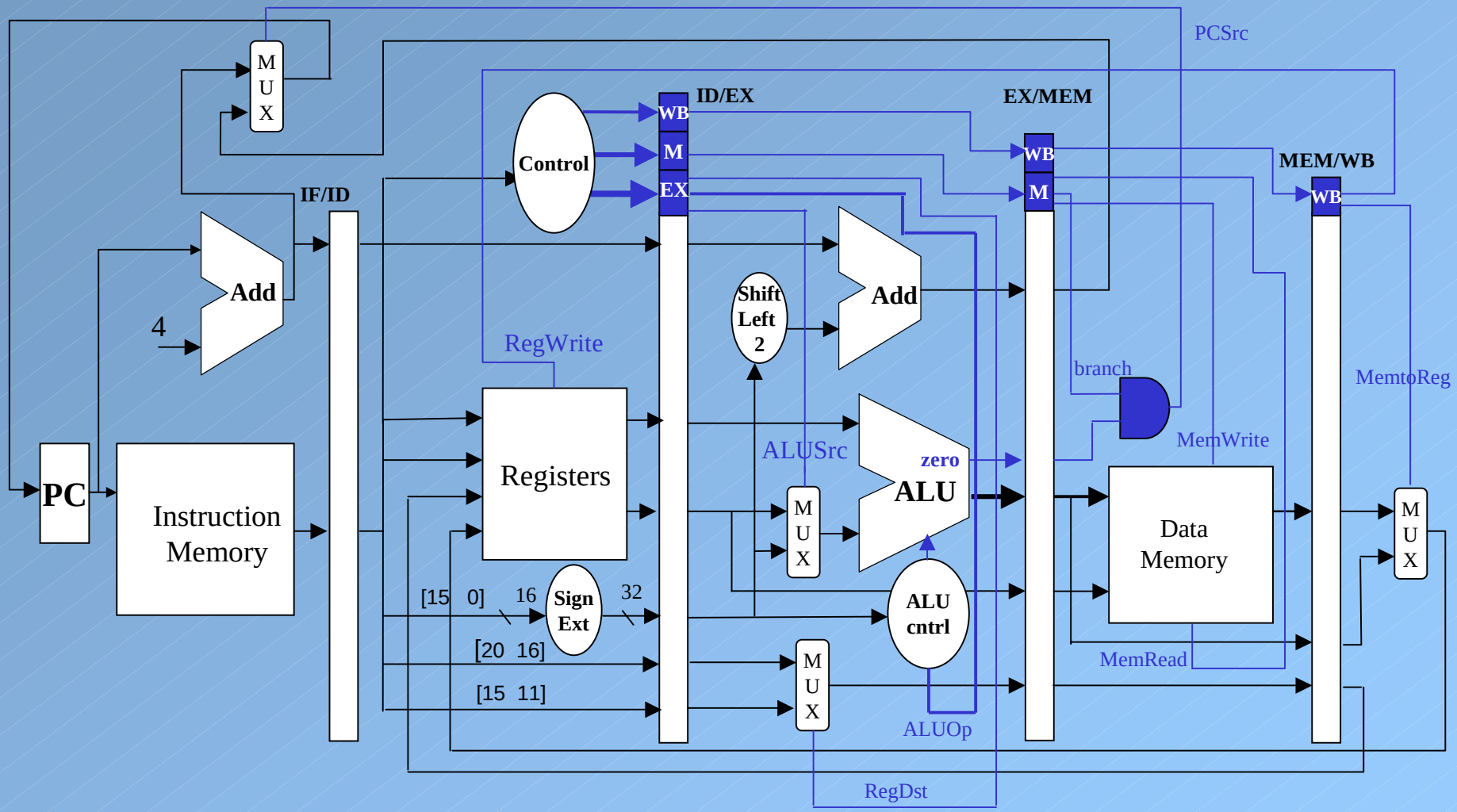
- Nine control signals that start in the EX stage
- Main control unit generates the control signals during the ID stage.

Instruction	EX Stage				MEM Stage			WB Stage	
	RegDest	ALU Op1	ALU Op0	ALU Src	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

Control Lines for the Last Three Stages



Pipelined Datapath with Control Signals

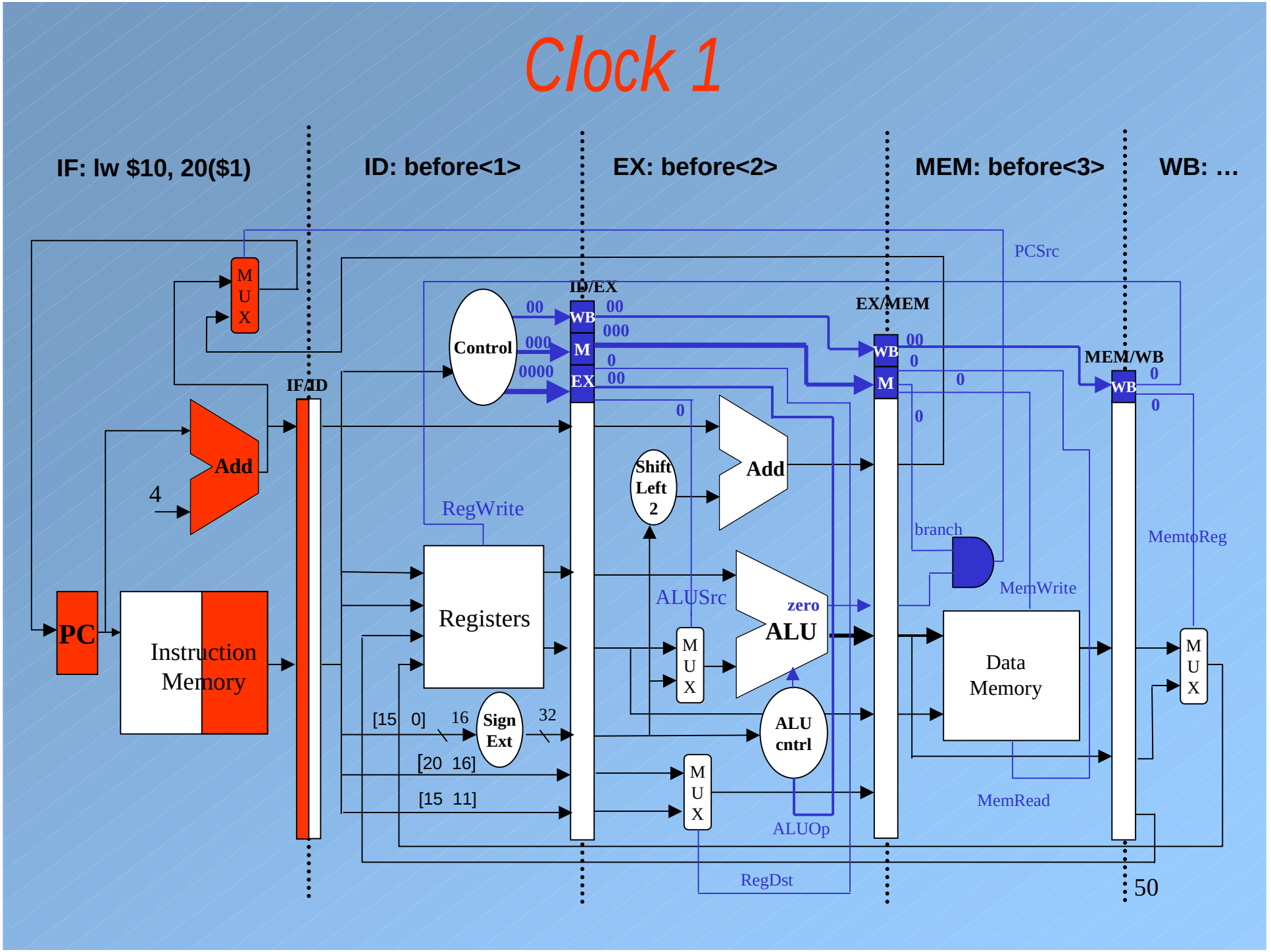


Example

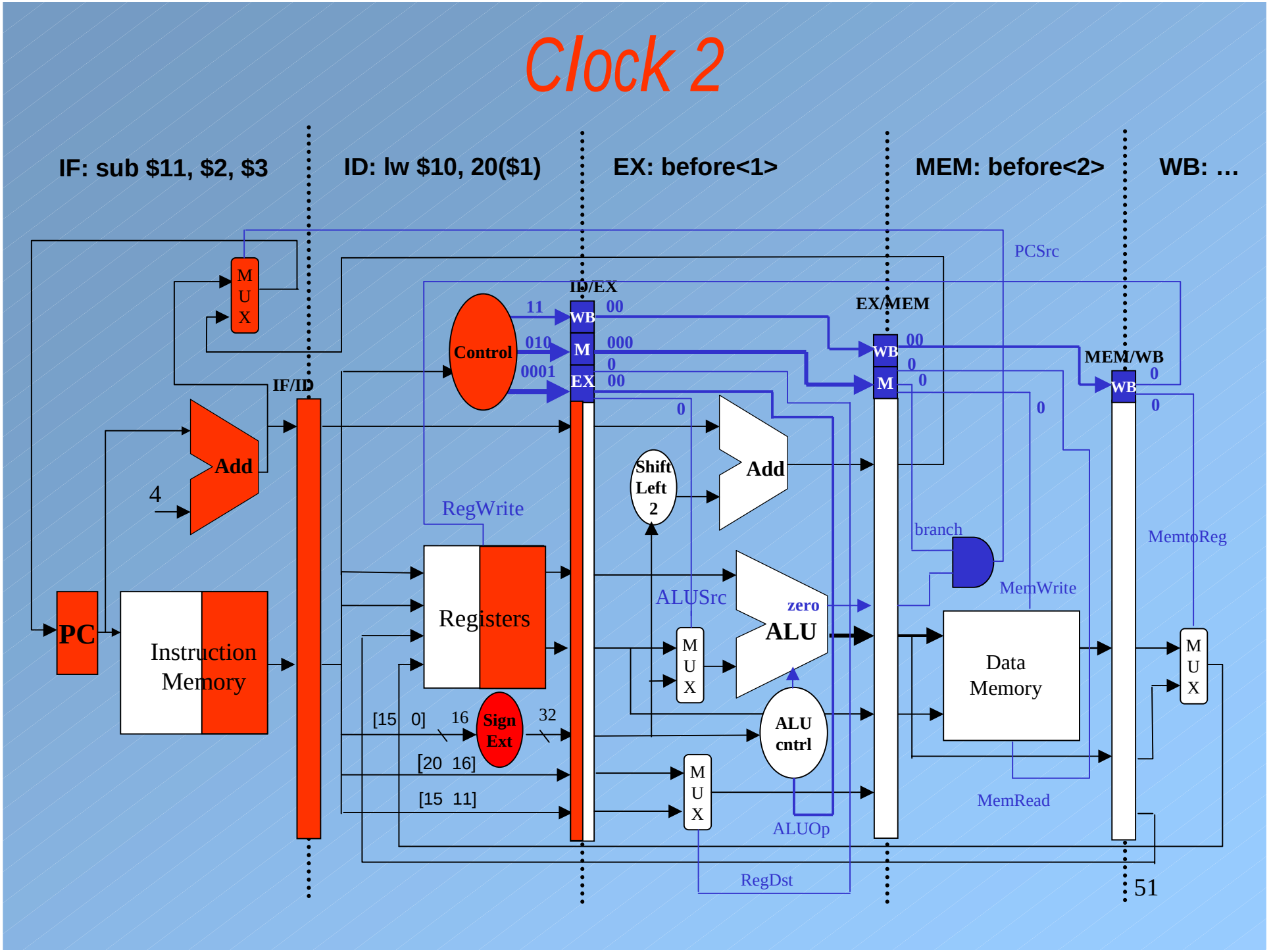
- Show these five instructions going through the pipeline:

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

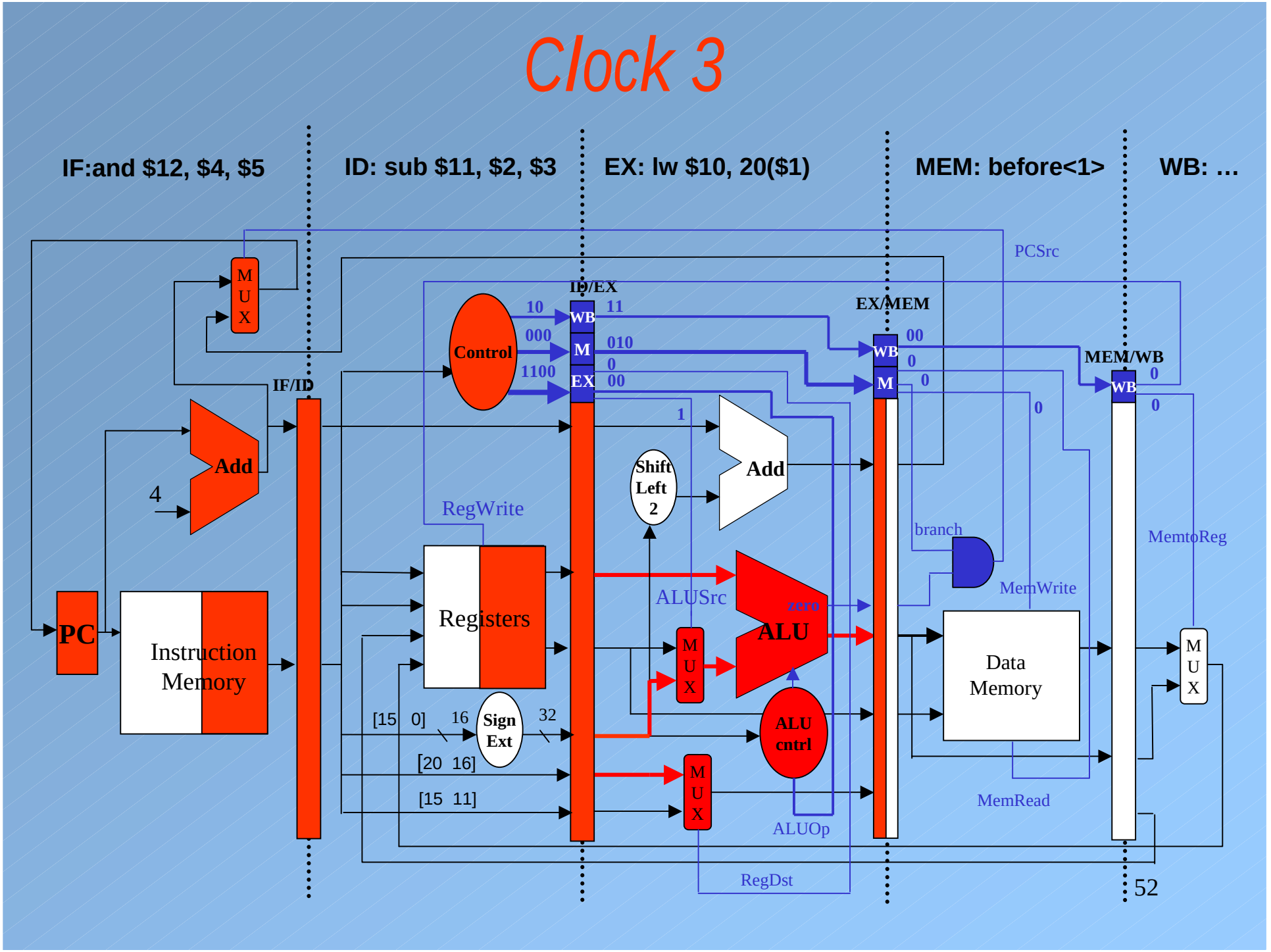
Clock 1


IF: lw $10, 20($1)
ID: before<1>
EX: before<2>
MEM: before<3>
WB: …
PCSrc
M U X
ID/EX
EX/MEM
MEM/WB
Control
00
000
0000
WB
M
EX
IF/ID
Add
4
PC
Instruction Memory
RegWrite
Registers
Shift Left 2
Add
branch
MemtoReg
ALUSrc
zero
ALU
MemWrite
Data Memory
[15 0]
16
Sign Ext
32
[20 16]
[15 11]
ALU cntrl
MemRead
ALUOp
RegDst

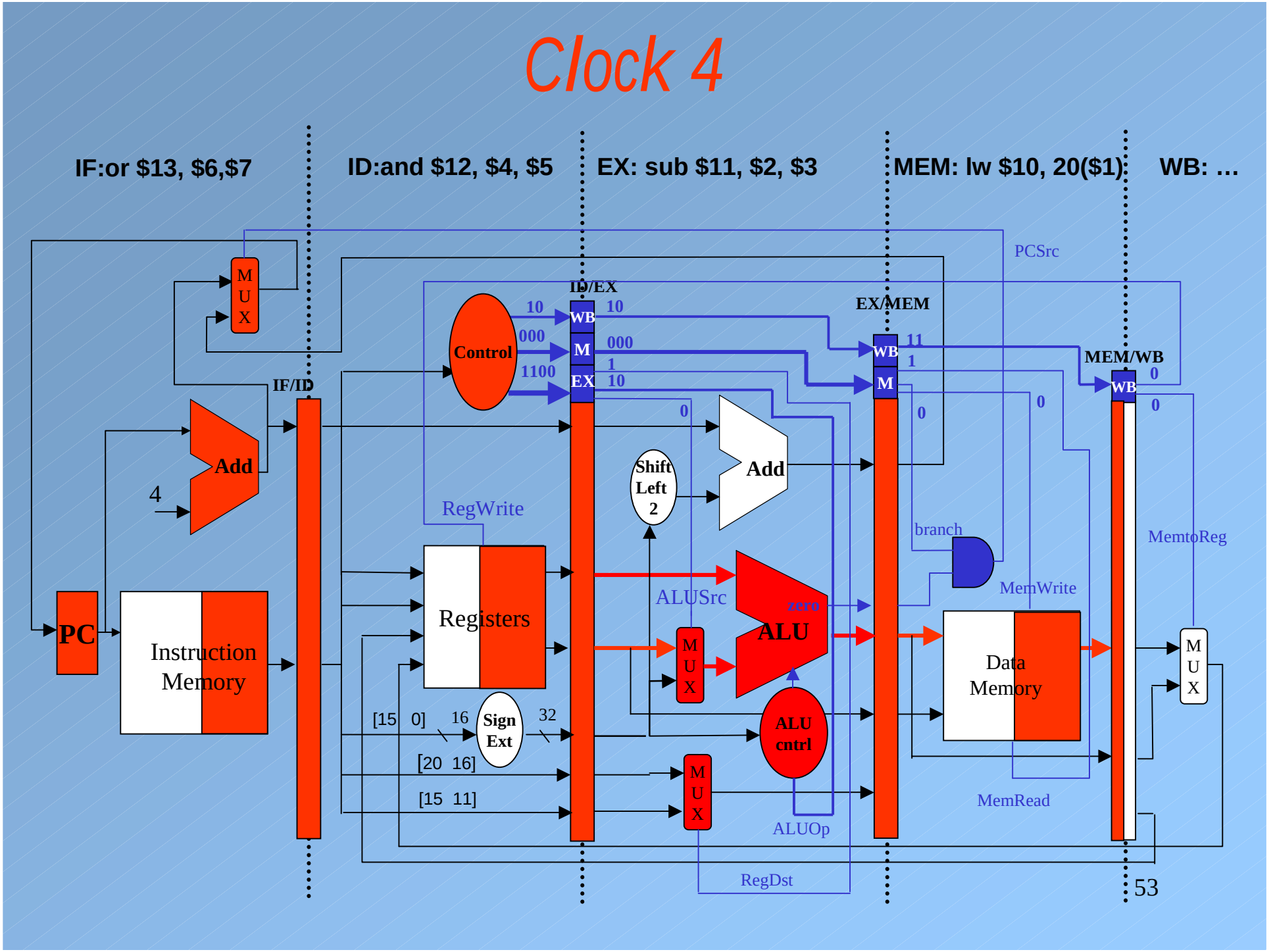
Clock 2


IF: sub $11, $2, $3
ID: lw $10, 20($1)
EX: before<1>
MEM: before<2>
WB: …
PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
11
010
0001
WB
M
EX
00
000
0
Add
4
Shift Left 2
RegWrite
branch
MemtoReg
MemWrite
ALUSrc
zero
ALU
PC
Instruction Memory
Registers
[15 0]
16
Sign Ext
32
[20 16]
[15 11]
ALU cntrl
ALUOp
MemRead
RegDst
Data Memory
MUX

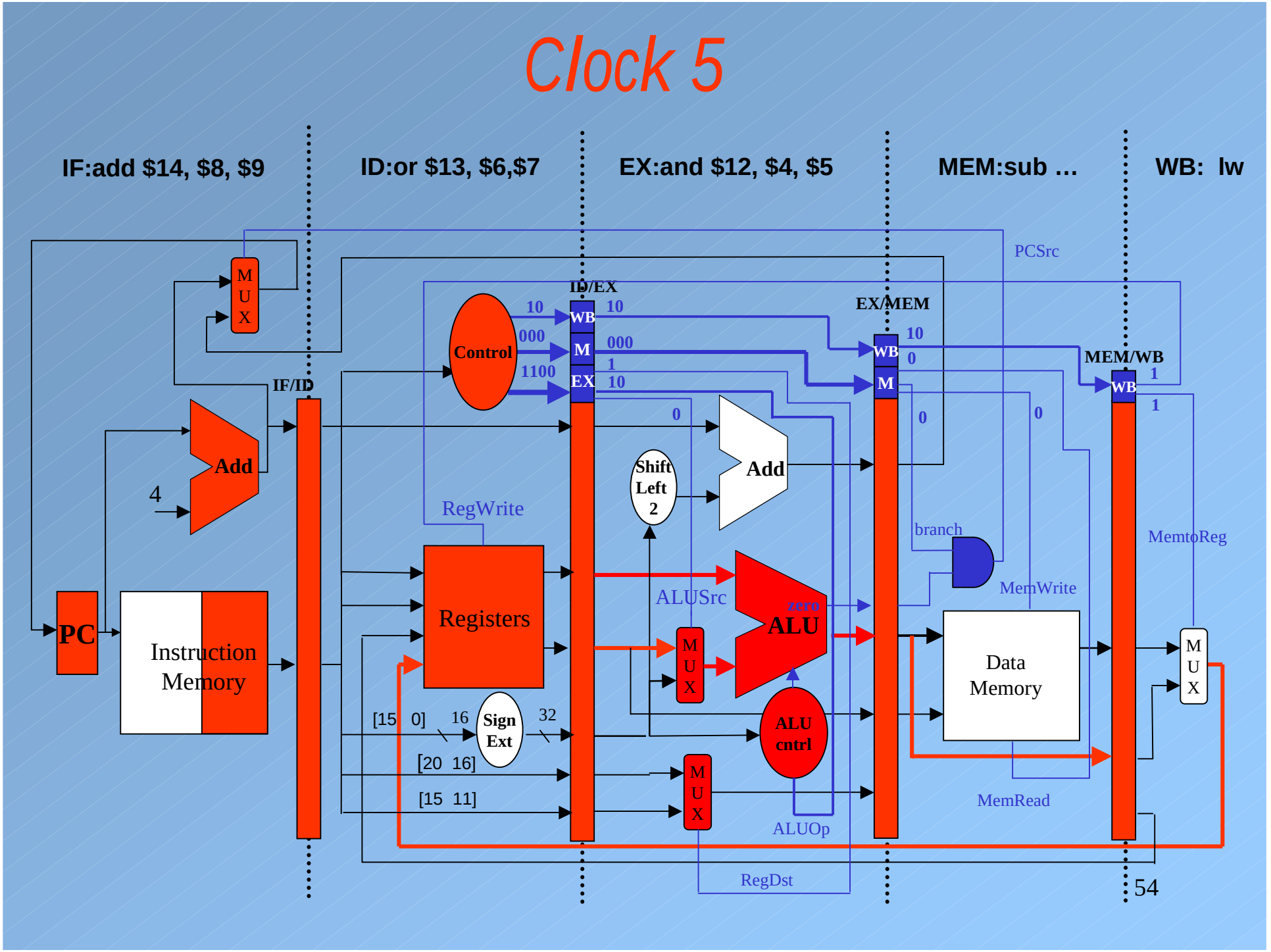
Clock 3


IF:and $12, $4, $5
ID: sub $11, $2, $3
EX: lw $10, 20($1)
MEM: before<1>
WB: …
PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
10
000
1100
WB
M
EX
11
010
00
00
0
0
0
0
0
0
1
Add
4
Add
Shift Left 2
RegWrite
branch
MemtoReg
MemWrite
ALUSrc
zero
ALU
PC
Instruction Memory
Registers
M U X
Data Memory
[15 0]
16
Sign Ext
32
[20 16]
[15 11]
ALU cntrl
MemRead
ALUOp
RegDst

Clock 4


IF:or $13, $6,$7
ID:and $12, $4, $5
EX: sub $11, $2, $3
MEM: lw $10, 20($1)
WB: …
PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
10
000
1100
WB
M
EX
Add
4
Shift Left 2
RegWrite
branch
MemtoReg
MemWrite
ALUSrc
zero
ALU
PC
Instruction Memory
Registers
MUX
[15 0]
16
Sign Ext
32
[20 16]
[15 11]
ALU cntrl
ALUOp
RegDst
Data Memory
MemRead

Clock 5

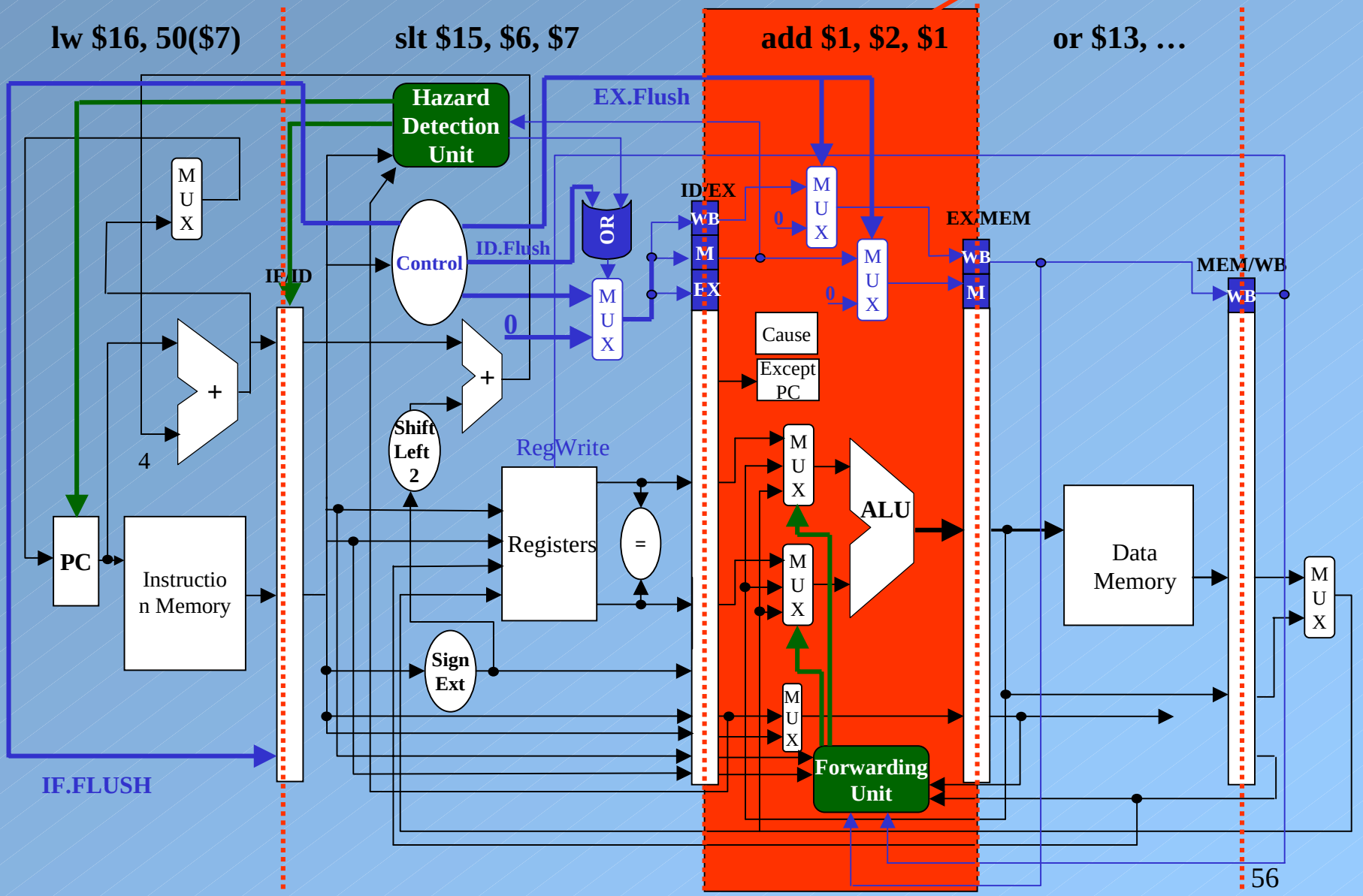

IF:add $14, $8, $9
ID:or $13, $6,$7
EX:and $12, $4, $5
MEM:sub …
WB: lw
PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
10
000
1100
WB
M
EX
Add
4
Shift Left 2
RegWrite
branch
MemtoReg
MemWrite
ALUSrc
zero
ALU
PC
Instruction Memory
Registers
MUX
Data Memory
[15 0]
16
Sign Ext
32
[20 16]
[15 11]
ALU cntrl
MemRead
ALUOp
RegDst

Example: Exception in the Pipelined Datapath

- ```
  0x40    sub  $11, $2,  $4
  0x44    and  $12, $2,  $5
  0x48    or   $13, $2,  $6
  0x4C    add  $1,  $2,  $1
  0x50    slt  $15, $6,  $7
  0x54    lw   $16, 50($7)
  ```
- Suppose an overflow exception occurs in the add instruction.
- Exception handling routine

  ```
  0x40000040   sw   $25, 1000($0)
  0x40000044   sw   $26, 1004($0)
  ```
  ```

Example: clock 6



Example: clock 7