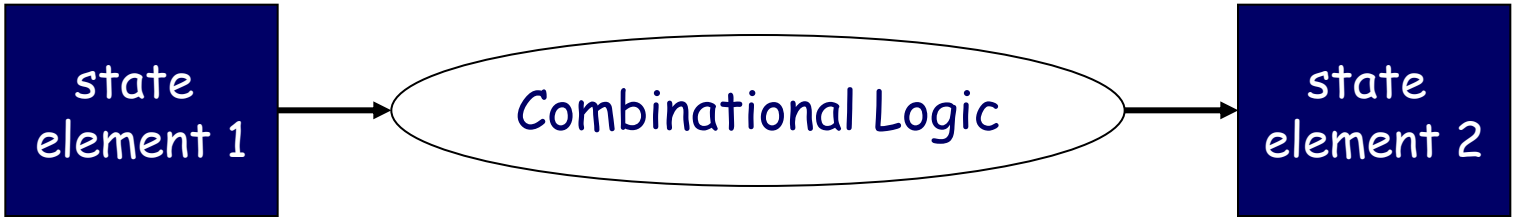
The Processor: Datapath & Control

Implementing Instructions

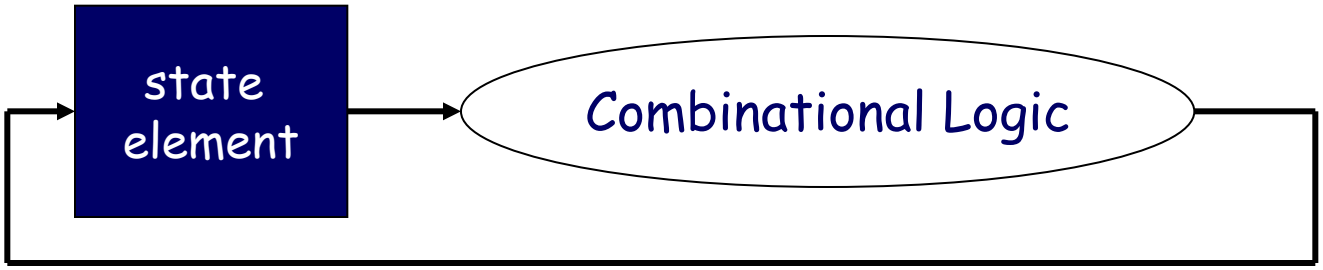
- Simplified instruction set
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq (bne), j (jal)`
- Generic implementation:
  1. PC to supply instruction address
  2. get the instruction from memory
  3. use the instruction to decide what to do
  4. read registers
- Majority of instructions use the ALU
  - the actions differ.

Clocking Methodology 1/2

- Edge-triggered methodology
  - values stored in a sequential logic elements are updated only on clock edge
- Typical execution:
  - read contents of state elements,
  - send values through some combinational logic
  - write results to one or more state elements



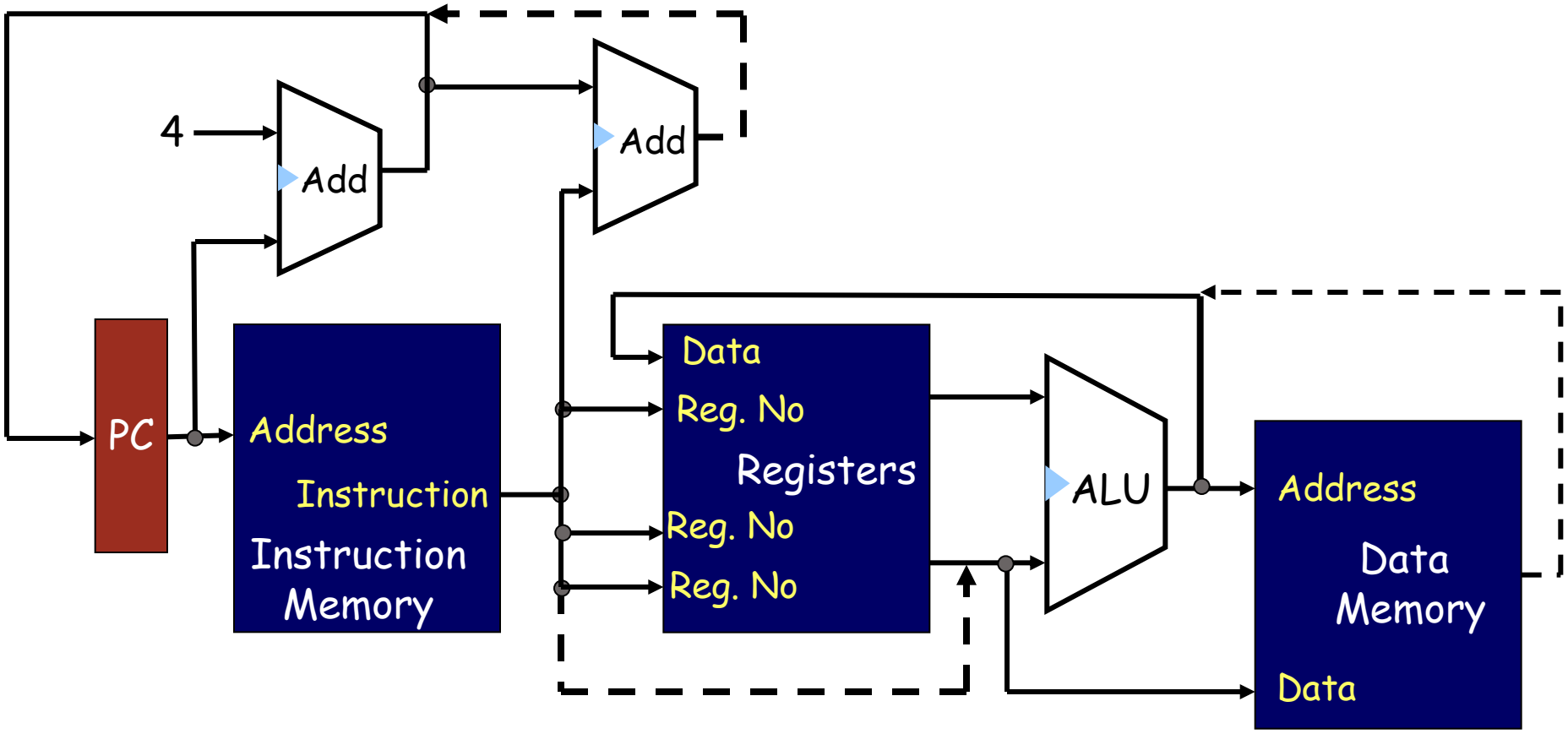
Clocking Methodology 2/2



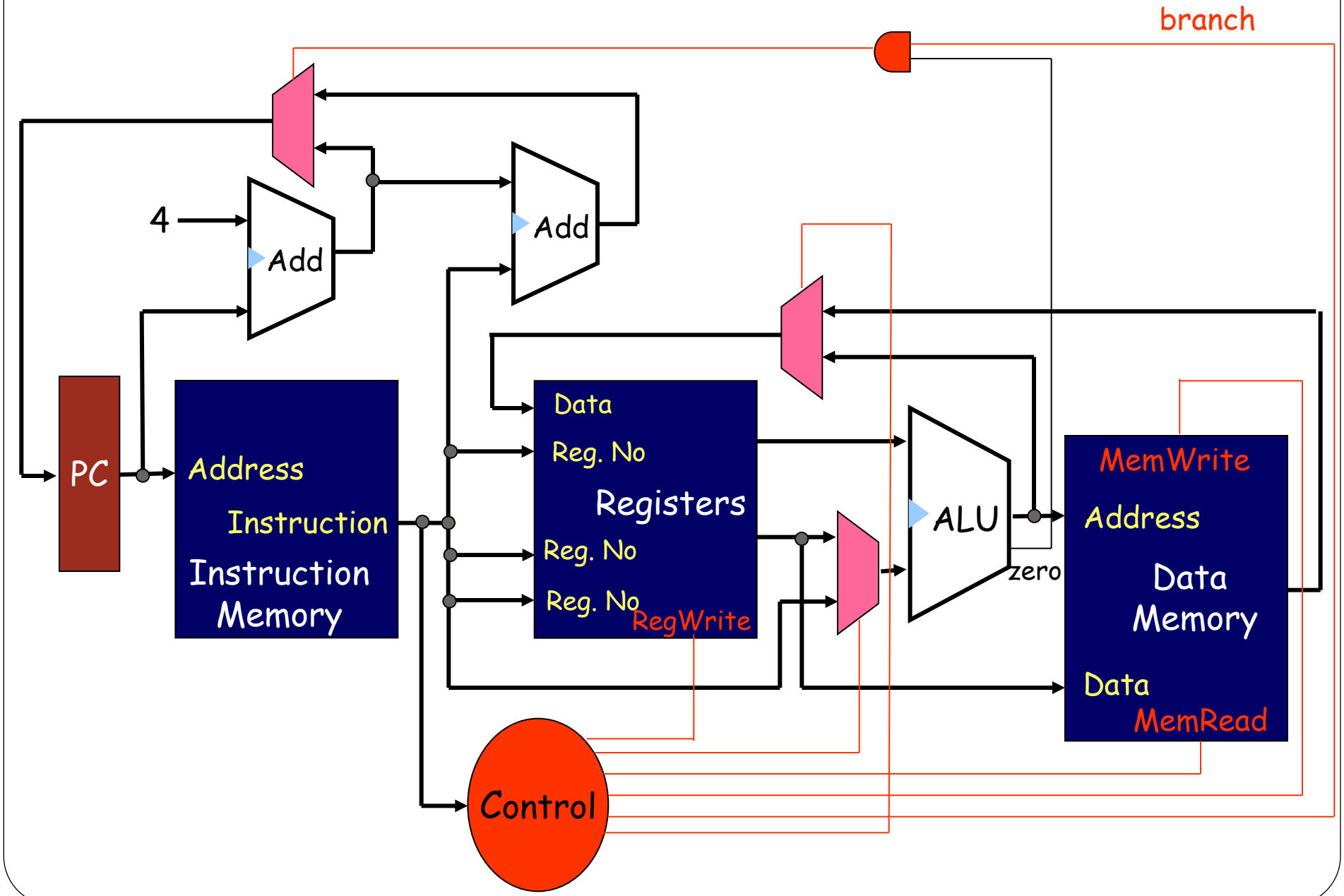
- An *edge-triggered methodology* allows a state element to be read and written in the same clock cycle

Abstract View of Datapath

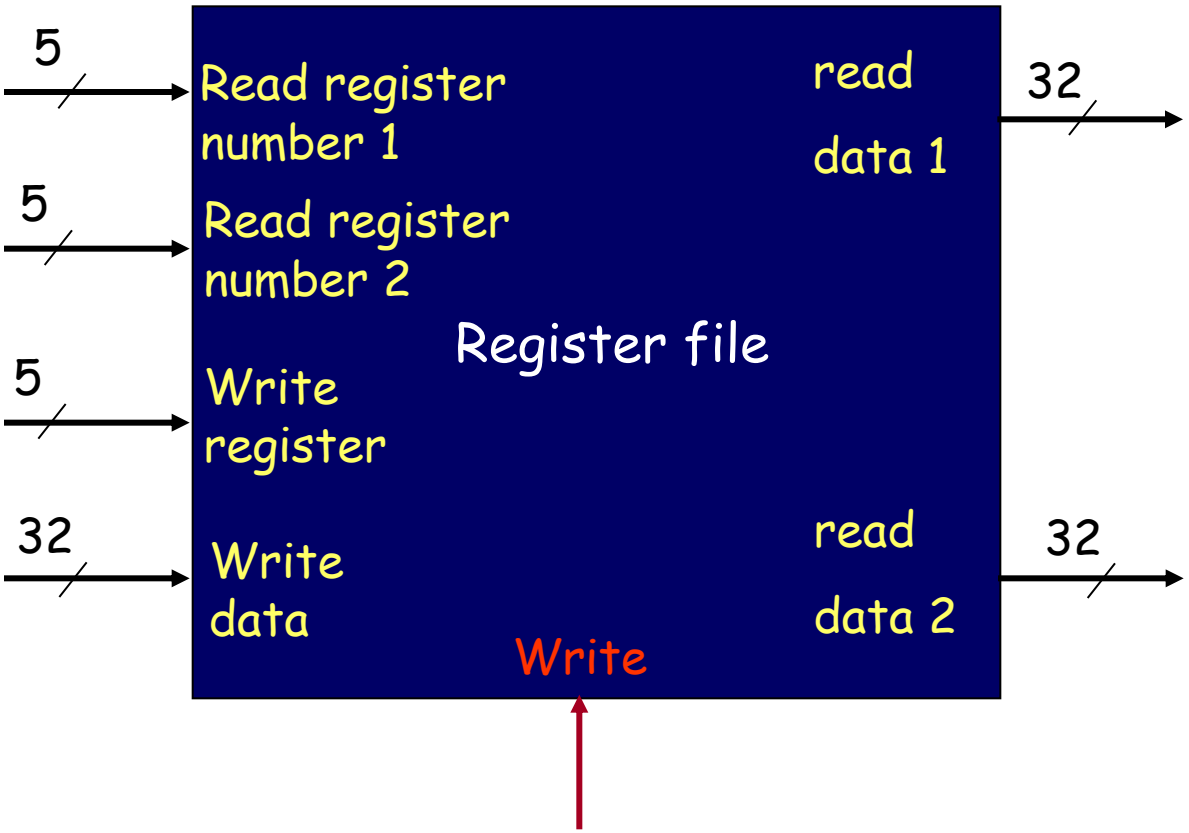
- Two types of functional units:
  - combinational, e.g. ALU
  - sequential, e.g. registers



Datapath Including Control

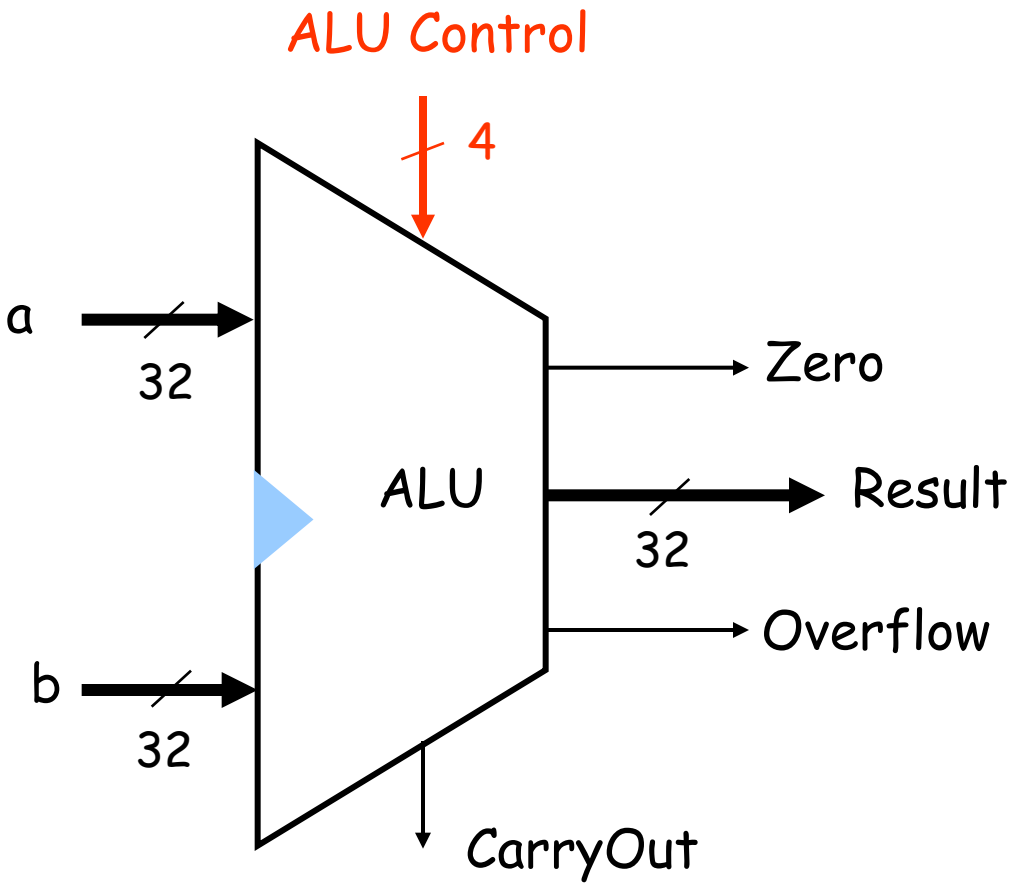


Register File

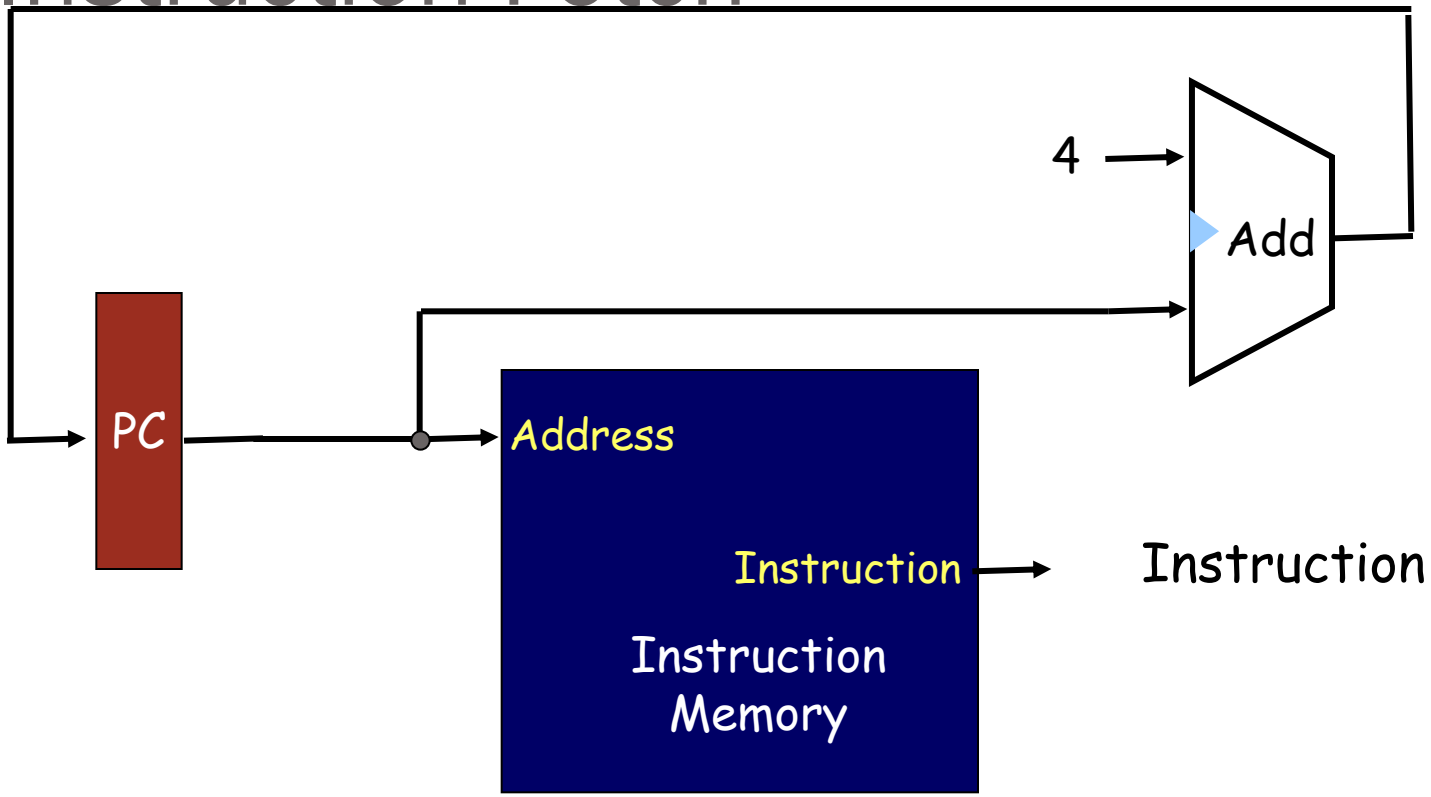


ALU Symbol & Control

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR



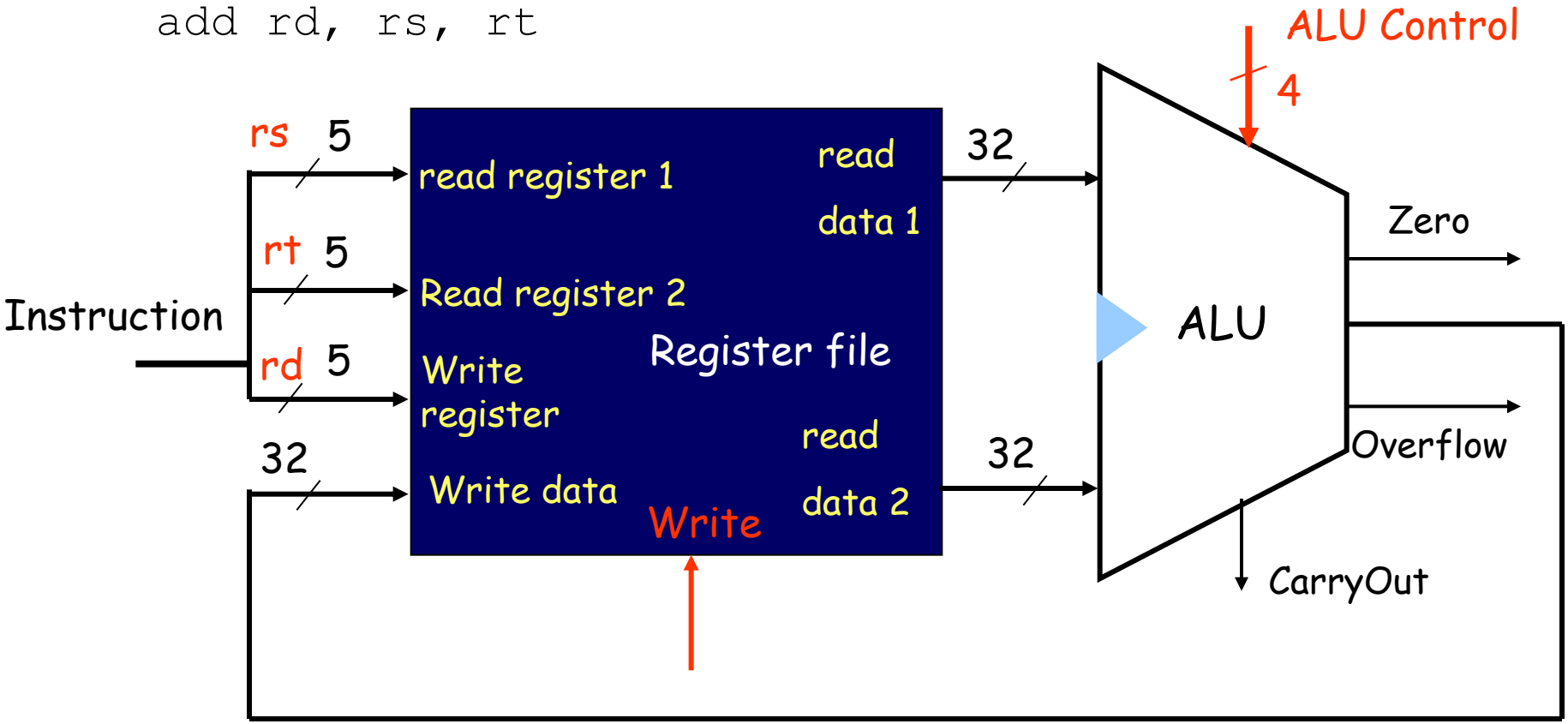
Instruction Fetch



R-Type Instructions

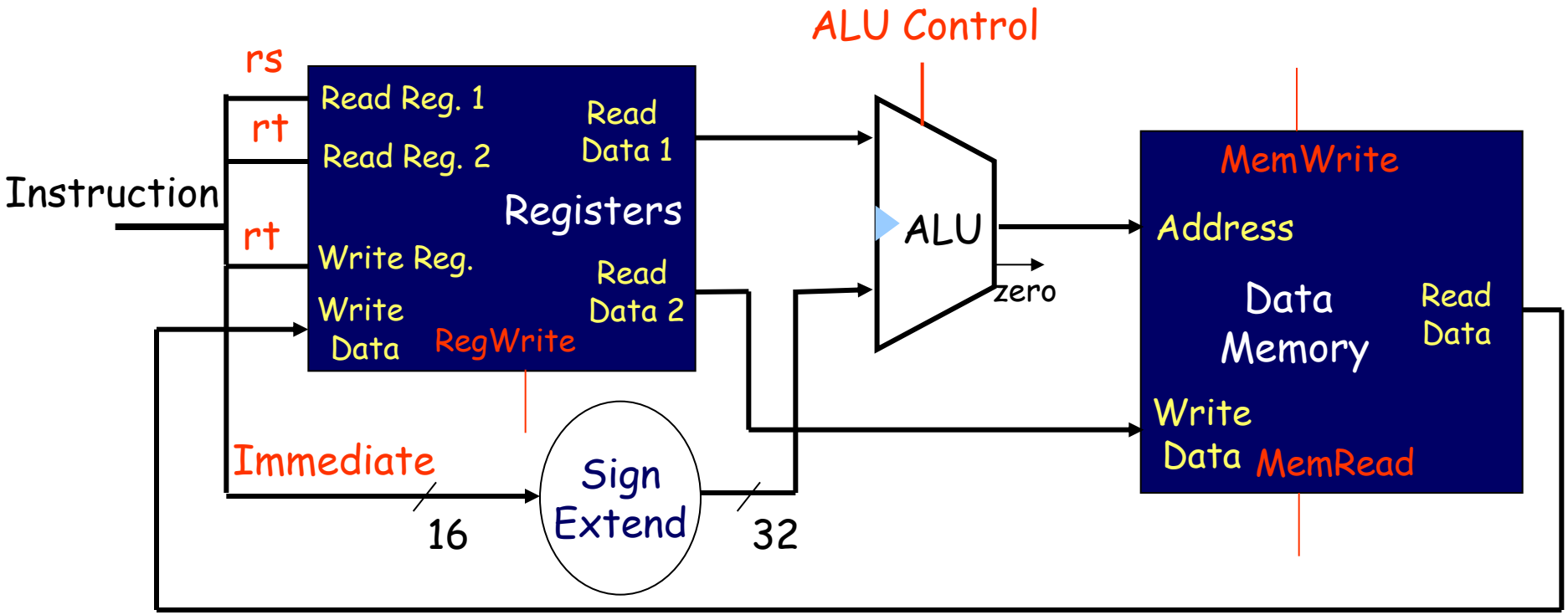
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

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

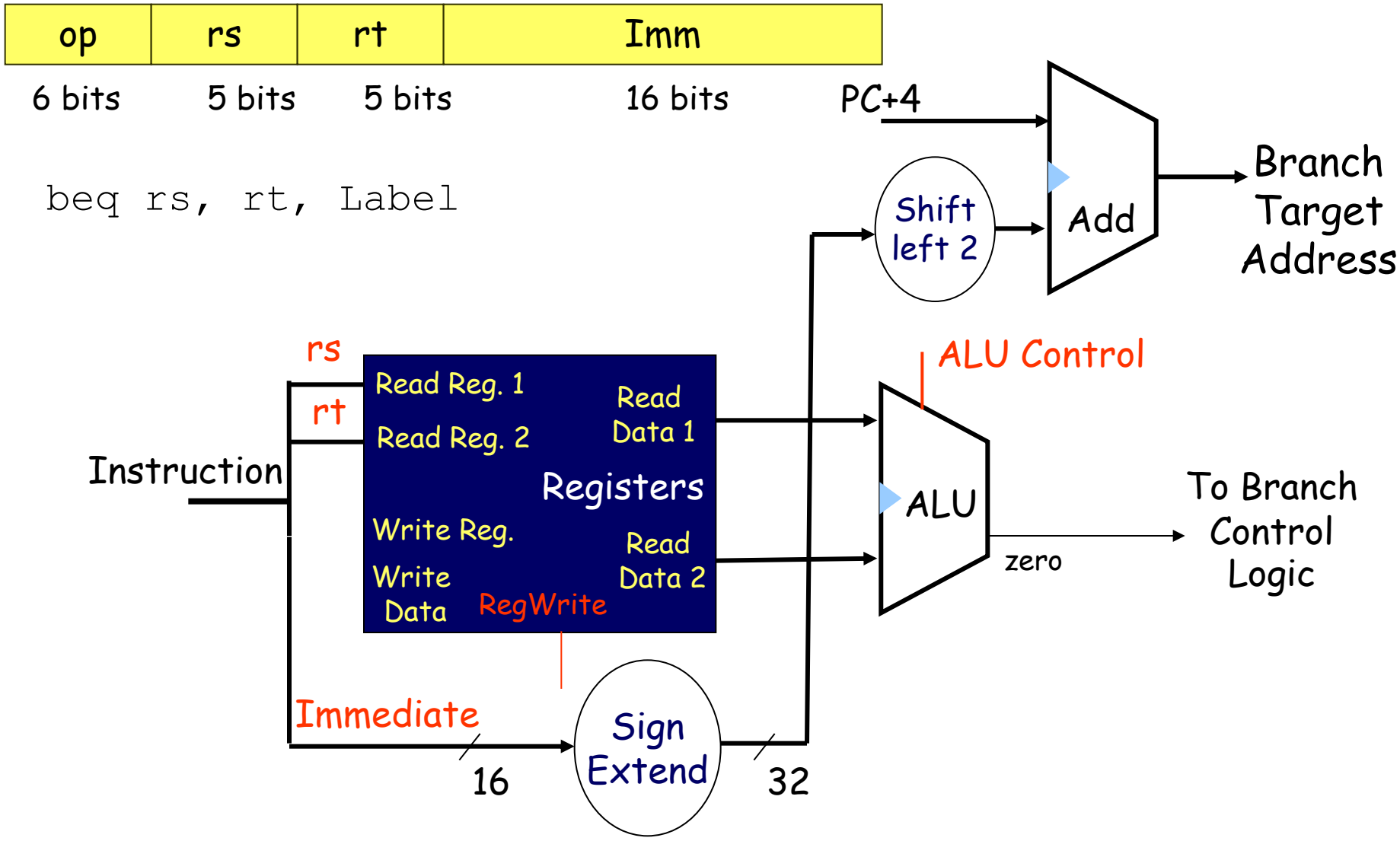


Implementing Load & Stores


```
lw rt, index(rs)
```

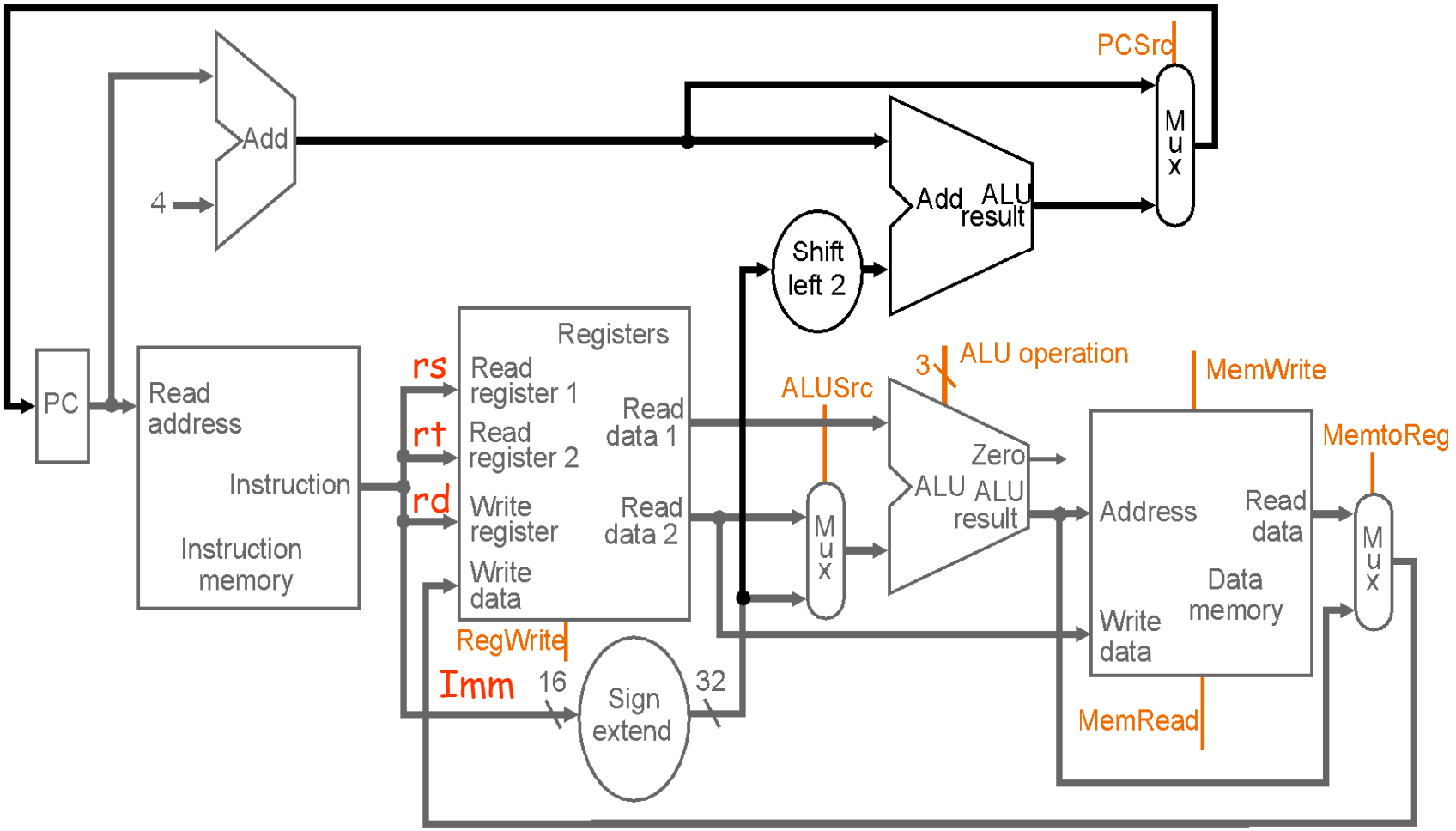


Implementing Branches

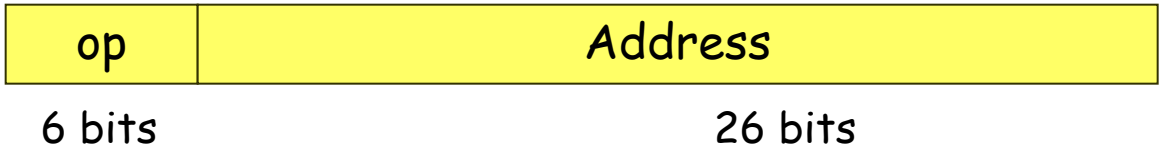


Building the Datapath

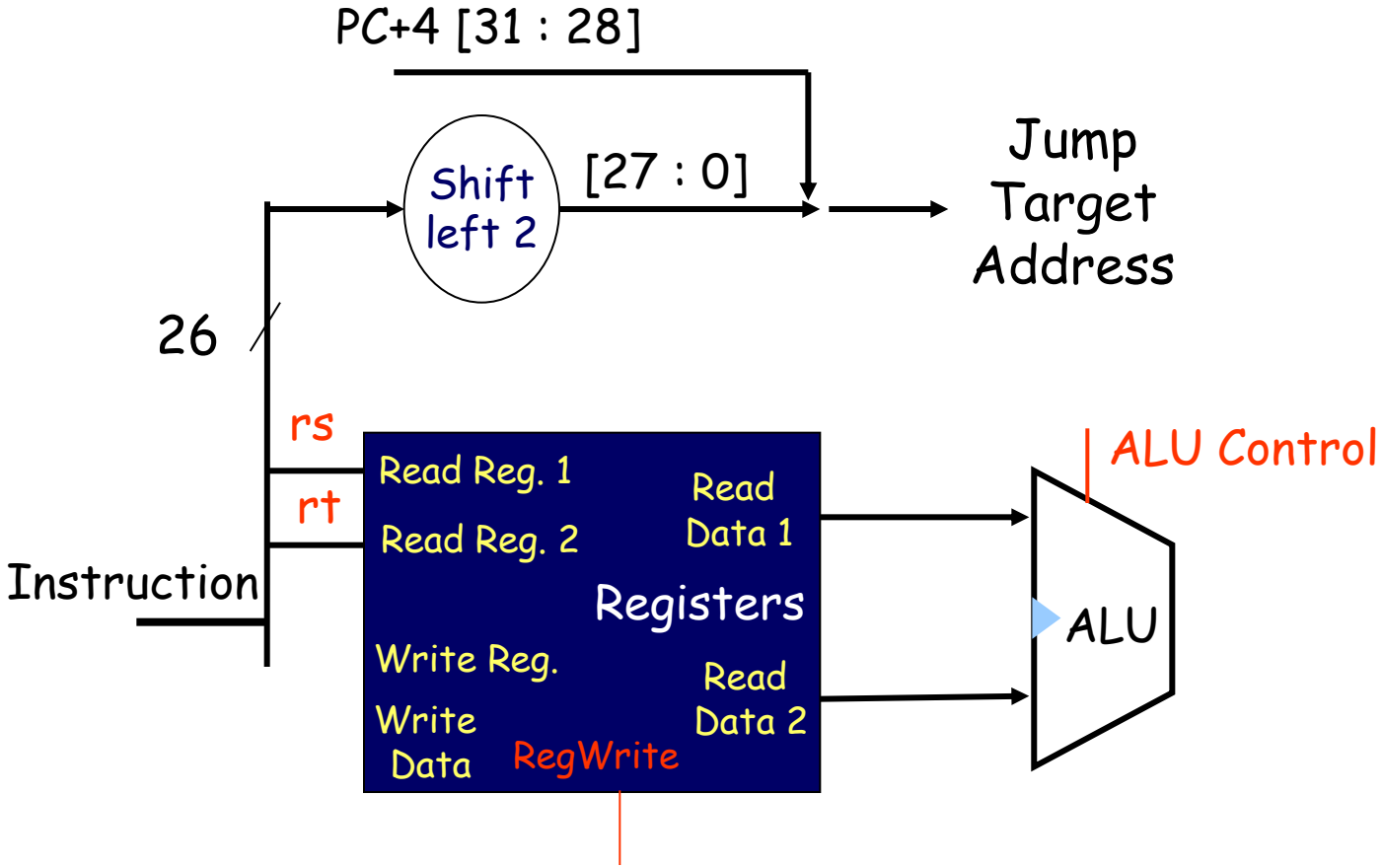
Idea: Use multiplexors to stitch them together



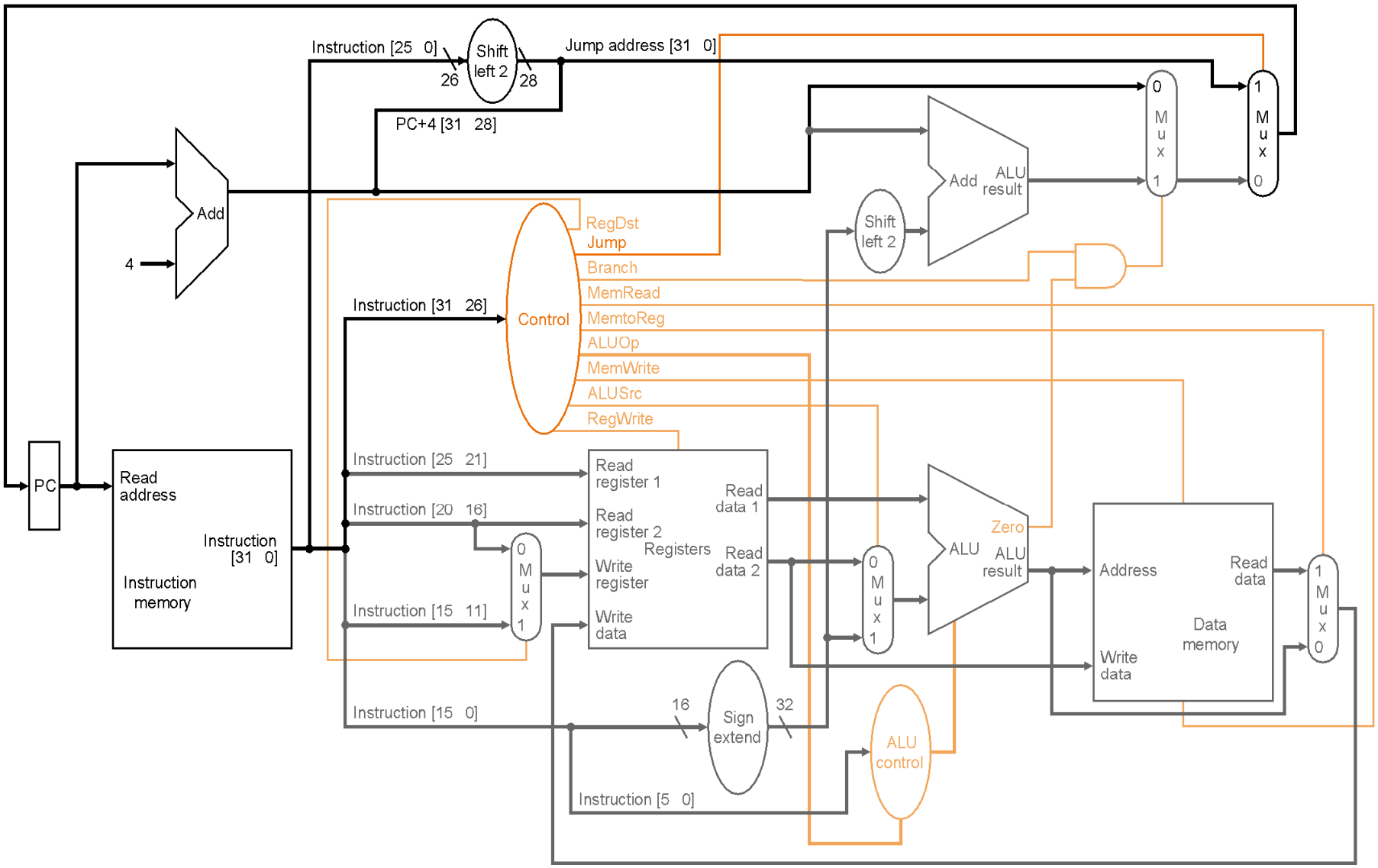
Implementing Jump




```
j Label
```



Complete Datapath with Control



Control

- Control Signals
  - Selecting the operations to perform
  - Controlling the flow of data (via MUX)
  - Read/write enable inputs of memory and register file
- Information comes from the instruction
- Example: `add $t0, $s0, $s1`

- ALU operation is based on instruction type and function code

ALU Control

- Example: `lw $t0, 100($s2)`
- What should the ALU do with this instruction?

```
100101  10010  01000  0000000001100100
```

op rs rt address

ALU Control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

ALU Control Unit

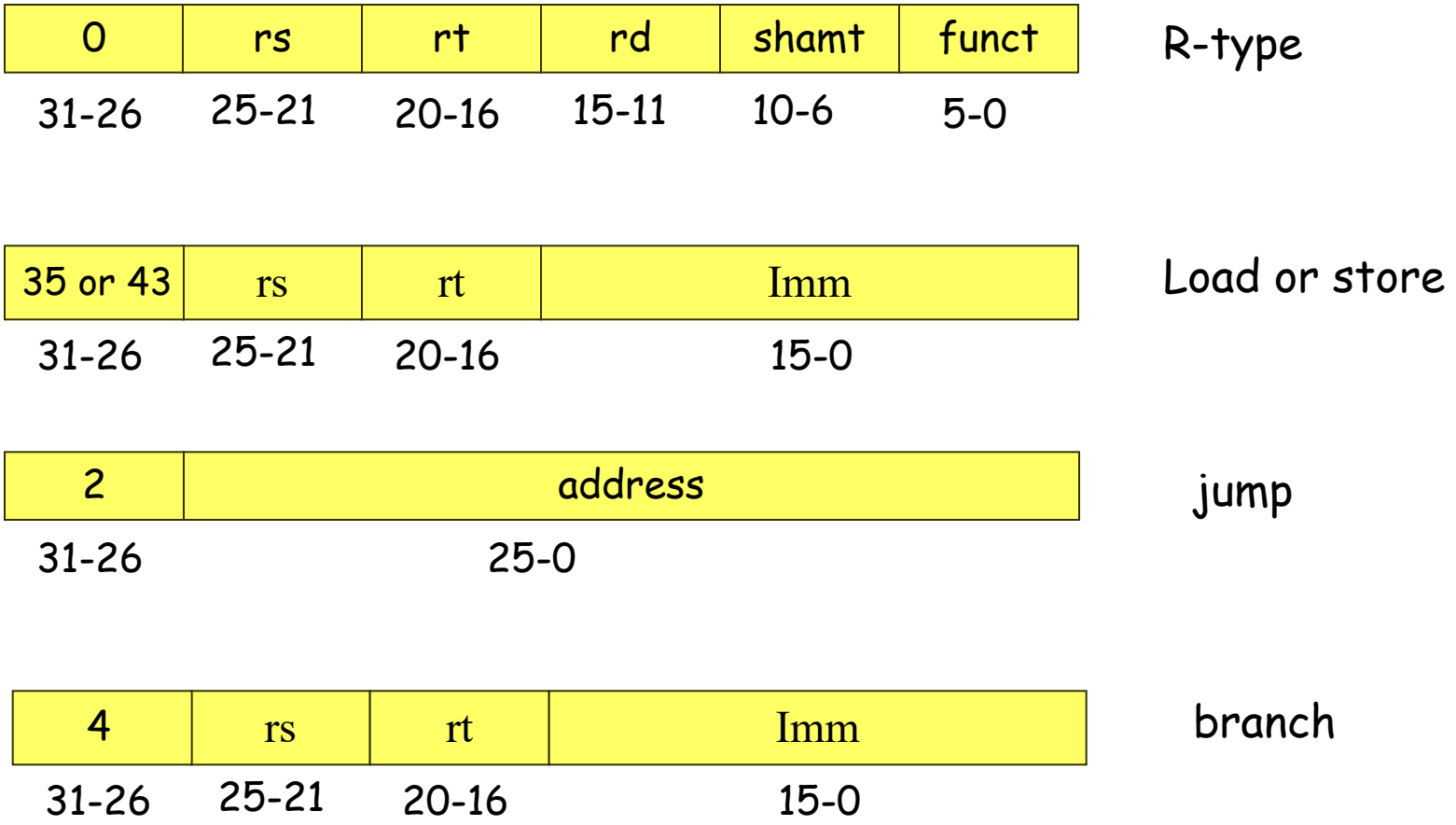
- ALU performs
  - addition for loads and stores
  - subtraction for branches (beq)
  - no operation for jumps
  - or the operation is determined by the function field for R-type instructions.
- ALU Control unit will have the following inputs:
  - 2-bit control field called `ALUOp`
  - 6-bit function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

ALU Control Unit

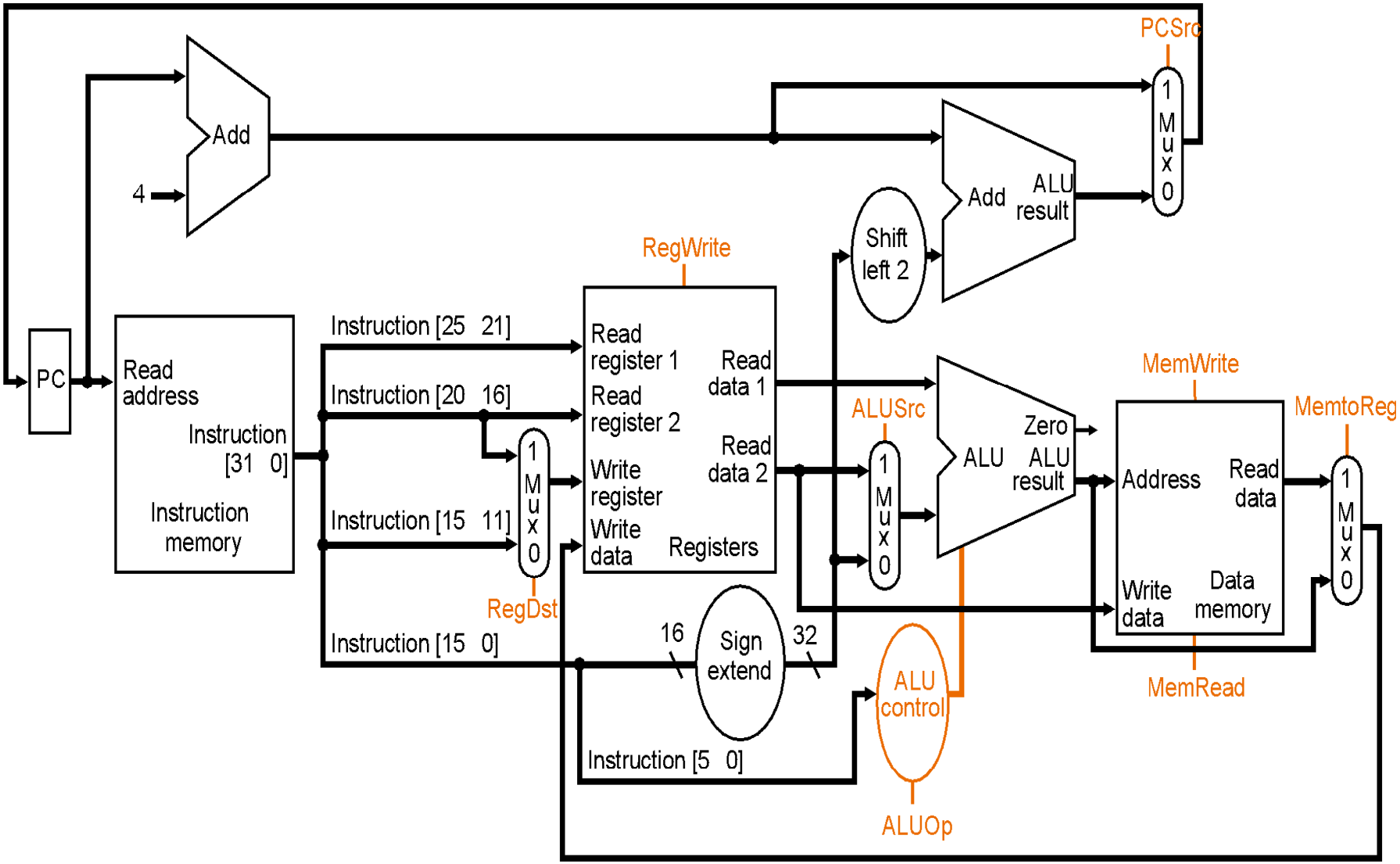
Instruction opcode	Instruction operation	ALUop	Funct field	Desired ALU action	ALU Control
`lw`	Load word	00	xxxxxx	add	0010
`sw`	Store word	00	xxxxxx	add	0010
`beq`	Branch equal	01	xxxxxx	subtract	0110
R-type	Add	10	100000	add	0010
R-type	Subtract	10	100010	subtract	0110
R-type	AND	10	100100	and	0000
R-type	OR	10	100101	or	0001
R-type	slt	10	101010	slt	0111

Main Control Unit

Fields of Different Instruction Classes:



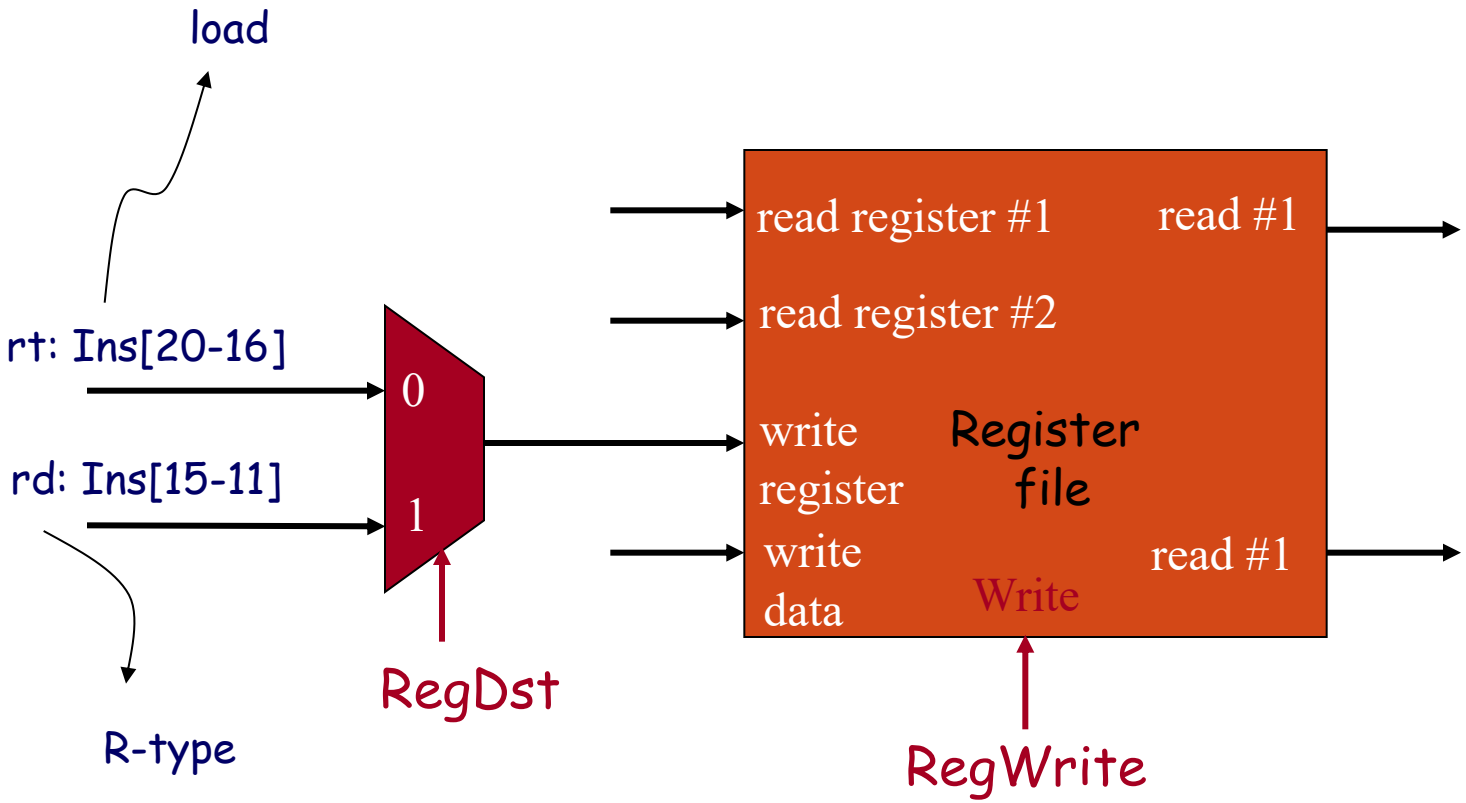
Datapath with Control Signals



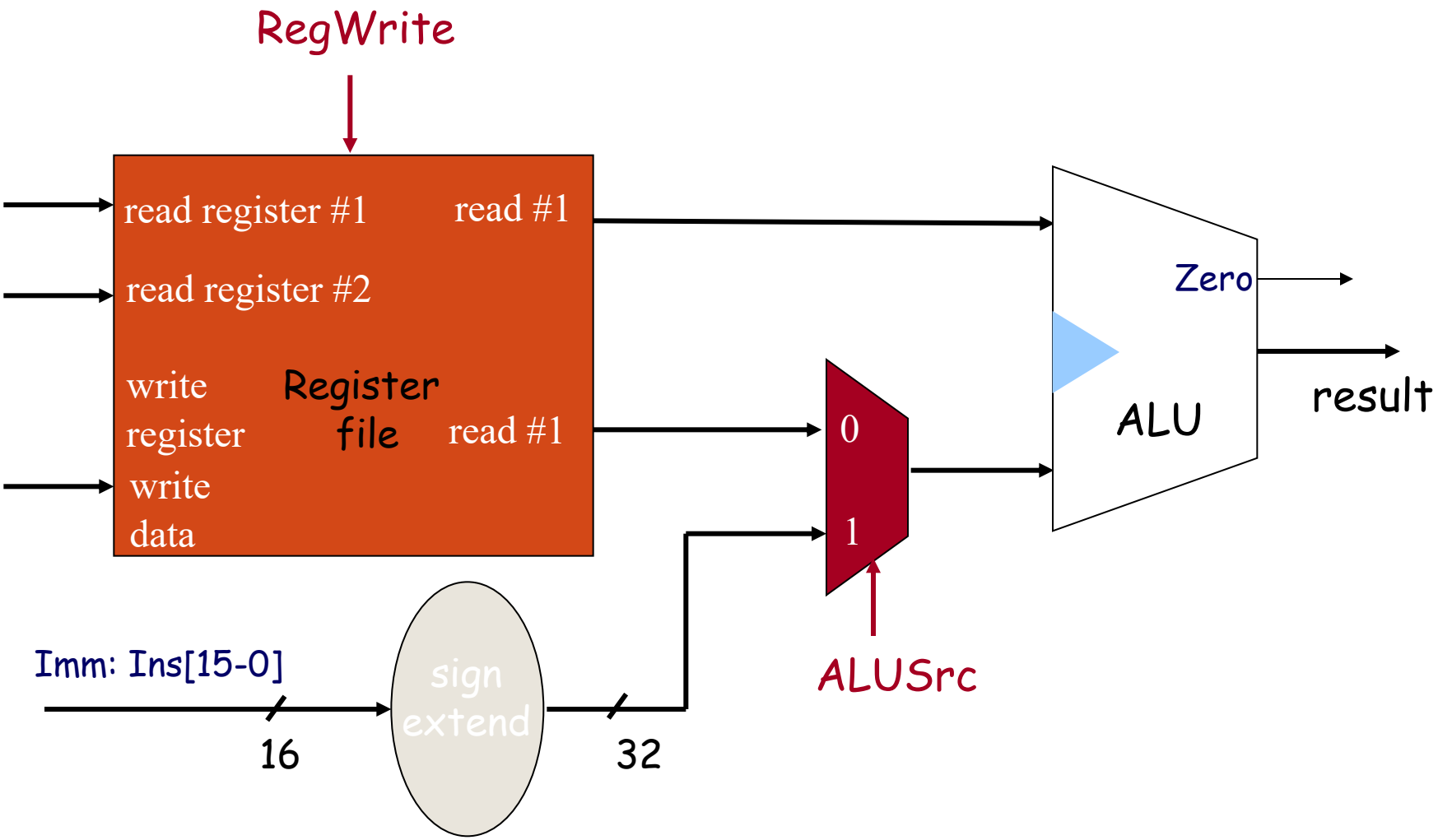
Seven Control Signals

Signal name	Effect when de-asserted	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 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

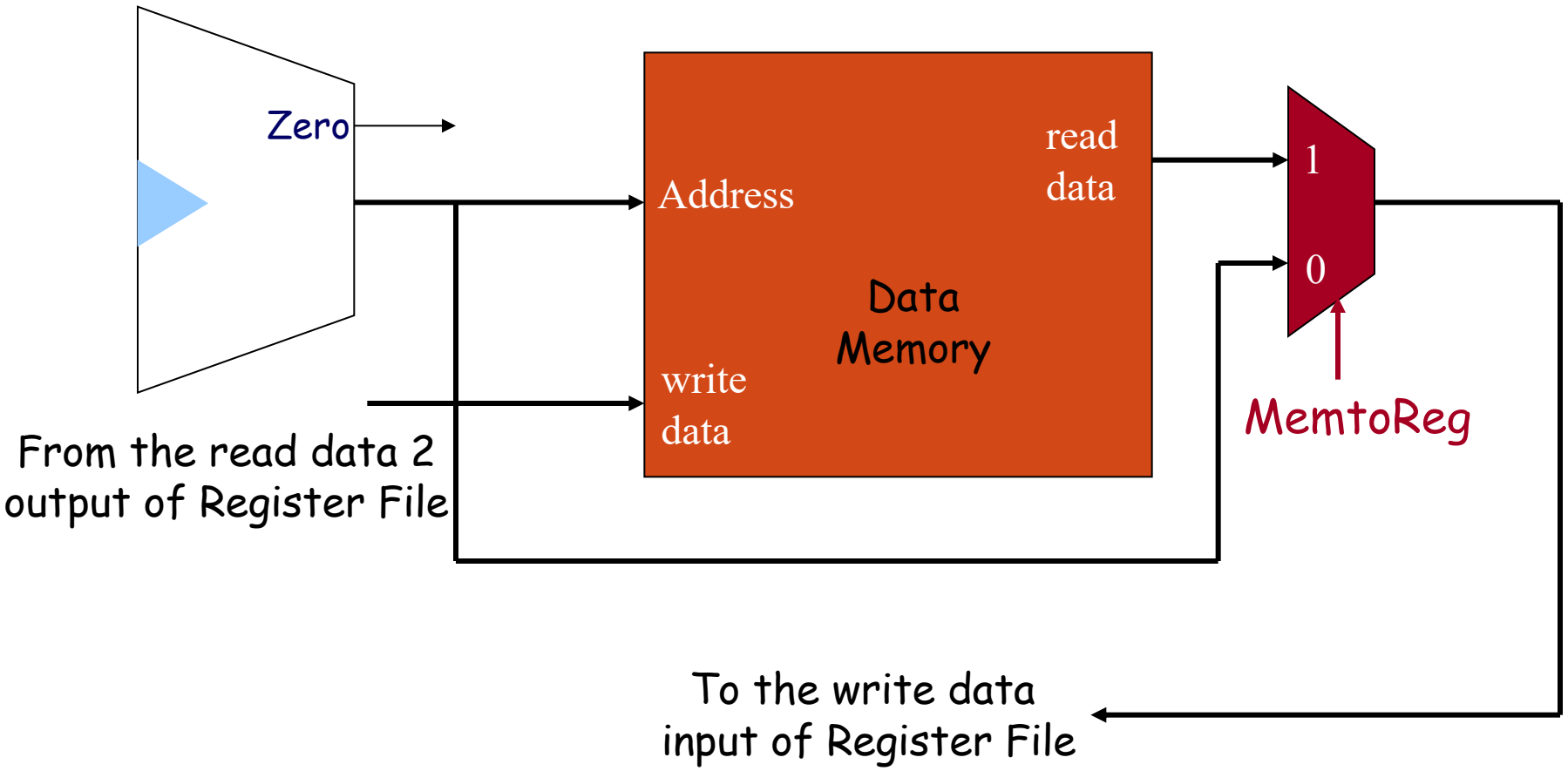
RegDst & RegWrite



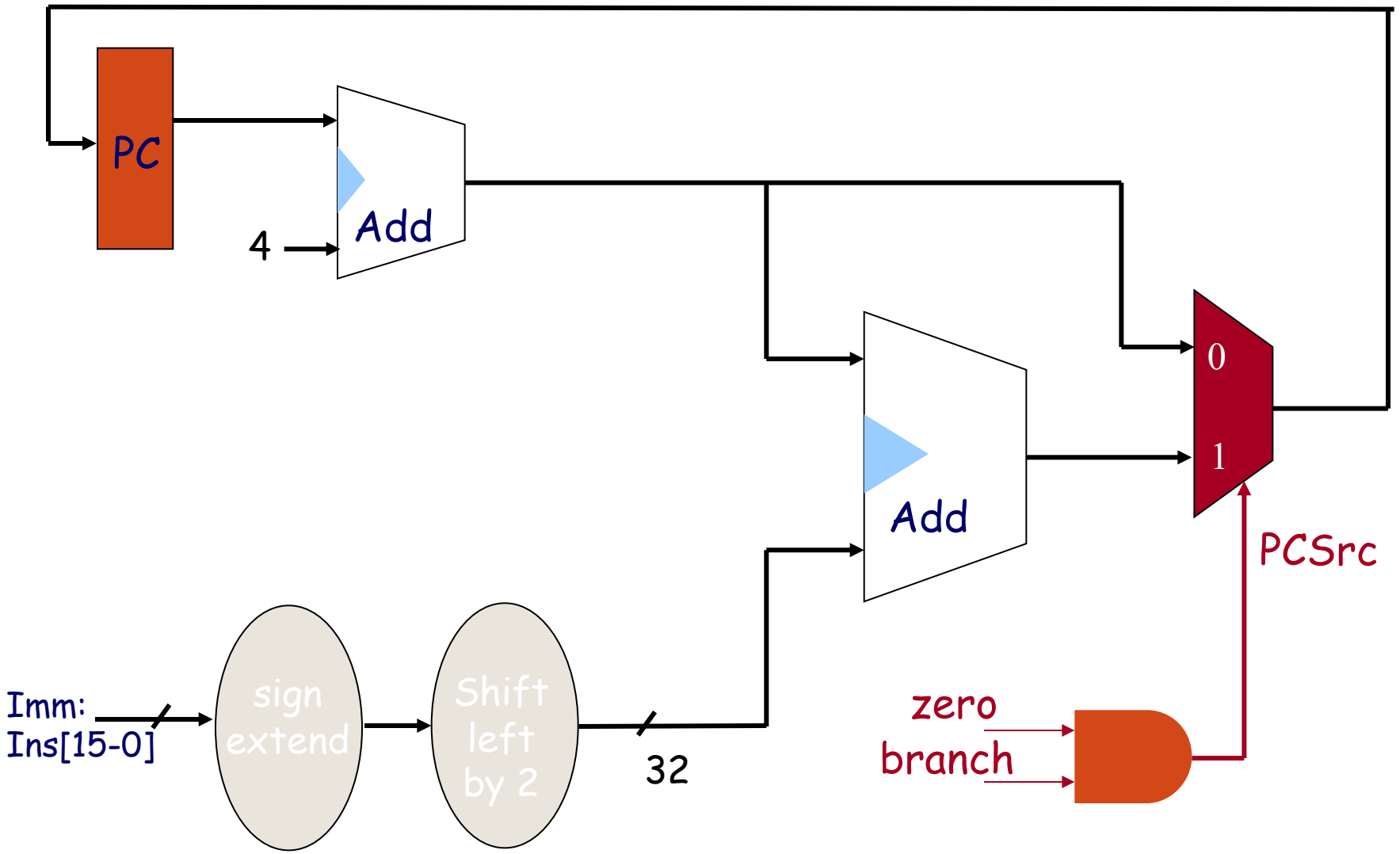
ALUSrc


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

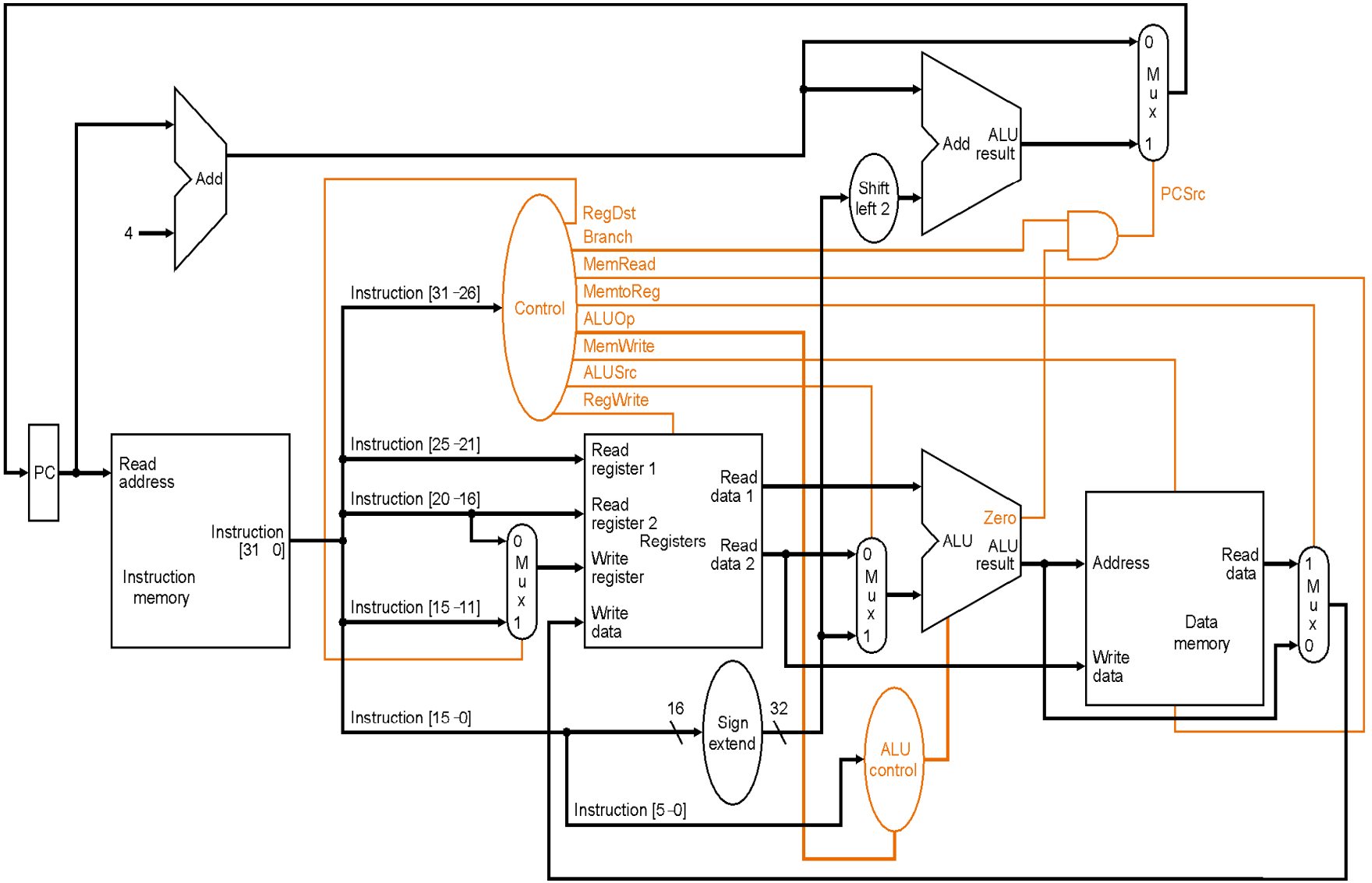
MemtoReg


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`

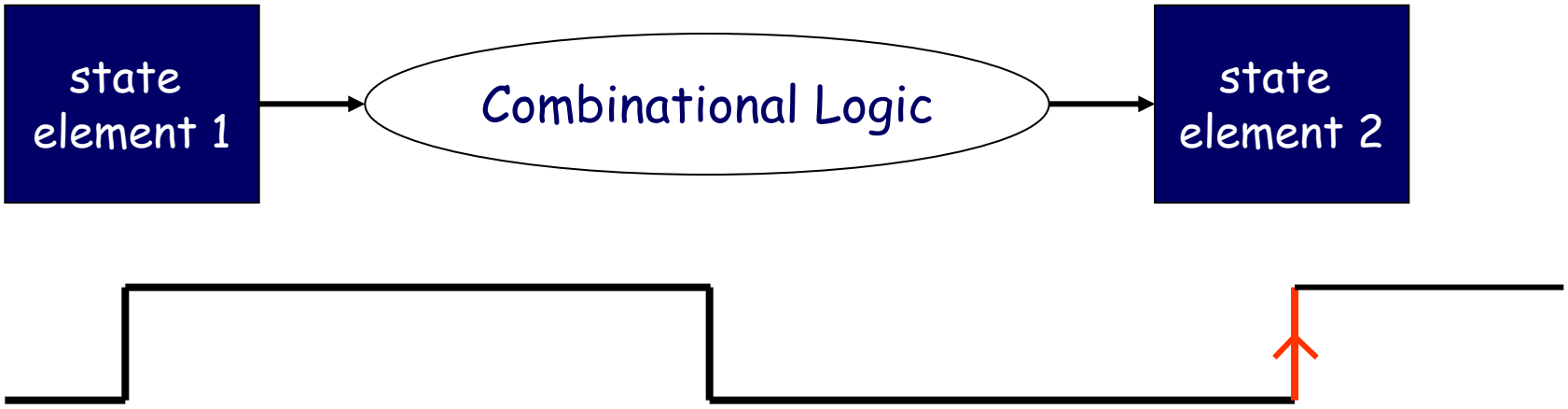
- The instruction is fetched from memory and PC is incremented
- Read two register values
- Subtract one from the other, calculate the branch address
- Use the `zero` signal to determine which of the addresses is to be used for fetching the next instruction

Control Function

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALIOp0	0	0	0	1

Cycle Time

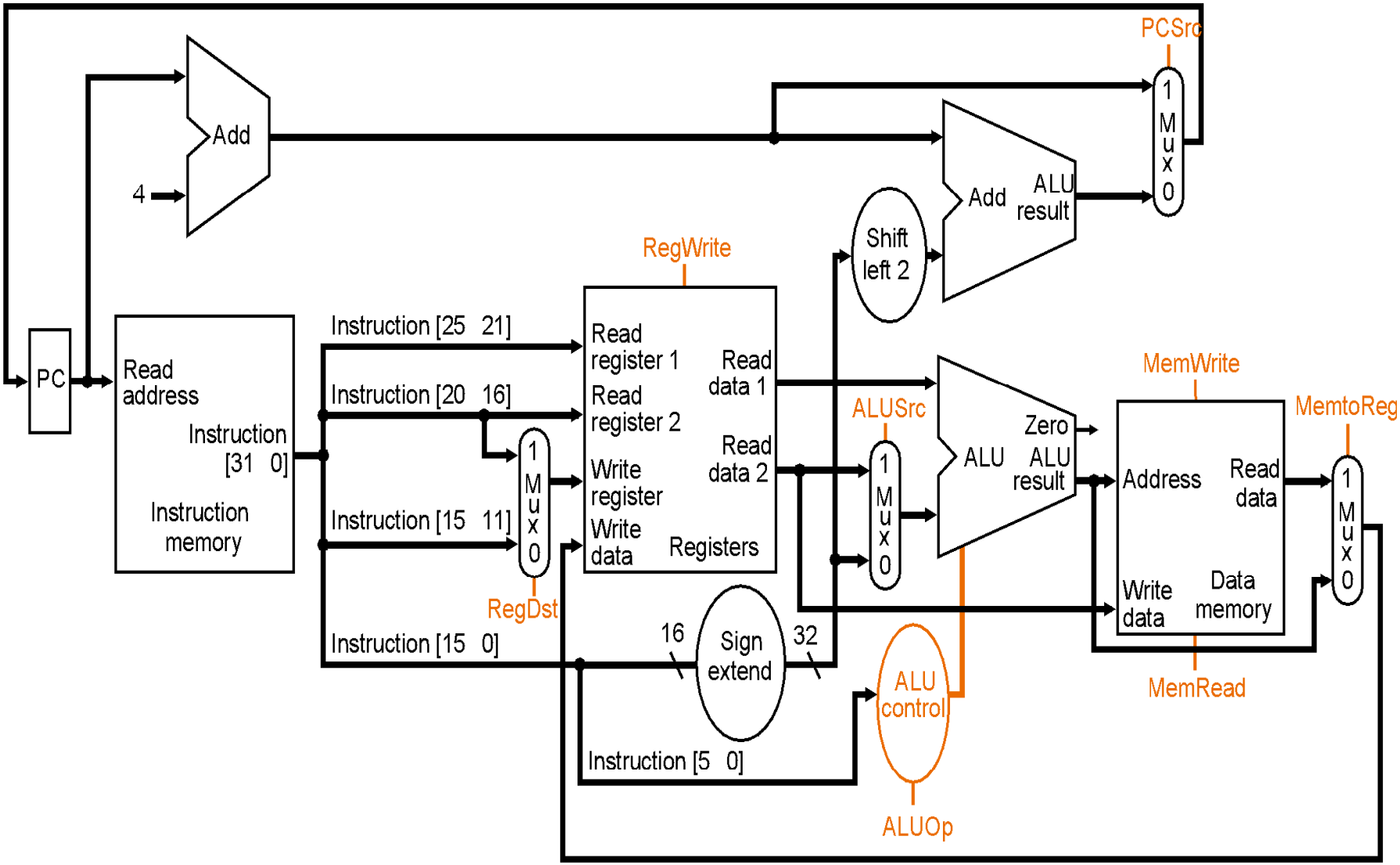
- The control logic is combinational
  - every instruction is executed in one clock cycle
- Cycle time determined by length of the longest path



Single Cycle Approach

- Different instructions have different execution times
  - Add: 3 ns
  - Subtract: 3.5 ns
  - Memory access: 10 ns
  - Multiplication: 20 ns
- In single cycle approach, the slowest instruction determines the clock cycle time.
- Another approach, divide instruction into smaller parts and execute each in a shorter clock cycle

Instruction in Datapath



Instruction Timings

Instruction class	Functional Units used by the instruction class				
R-type	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

Example

- Memory access: 200 ps
- ALU and addition operations: 100 ps
- Register file access (read or write): 50 ps
  - And assume other parts (multiplexors, control units, etc) have no delay.
- Instruction mix: 25% loads, 10% stores, 45% ALU ops, 15% branches, 5% jumps
- Compare two approaches: single fixed clock cycle and multiple clock cycle per instruction, i.e.
- what is the clock period per instruction and CPI for a fixed cycle
- what is the nominal clock period and CPI for the multiple cycle approach
- Execution times?