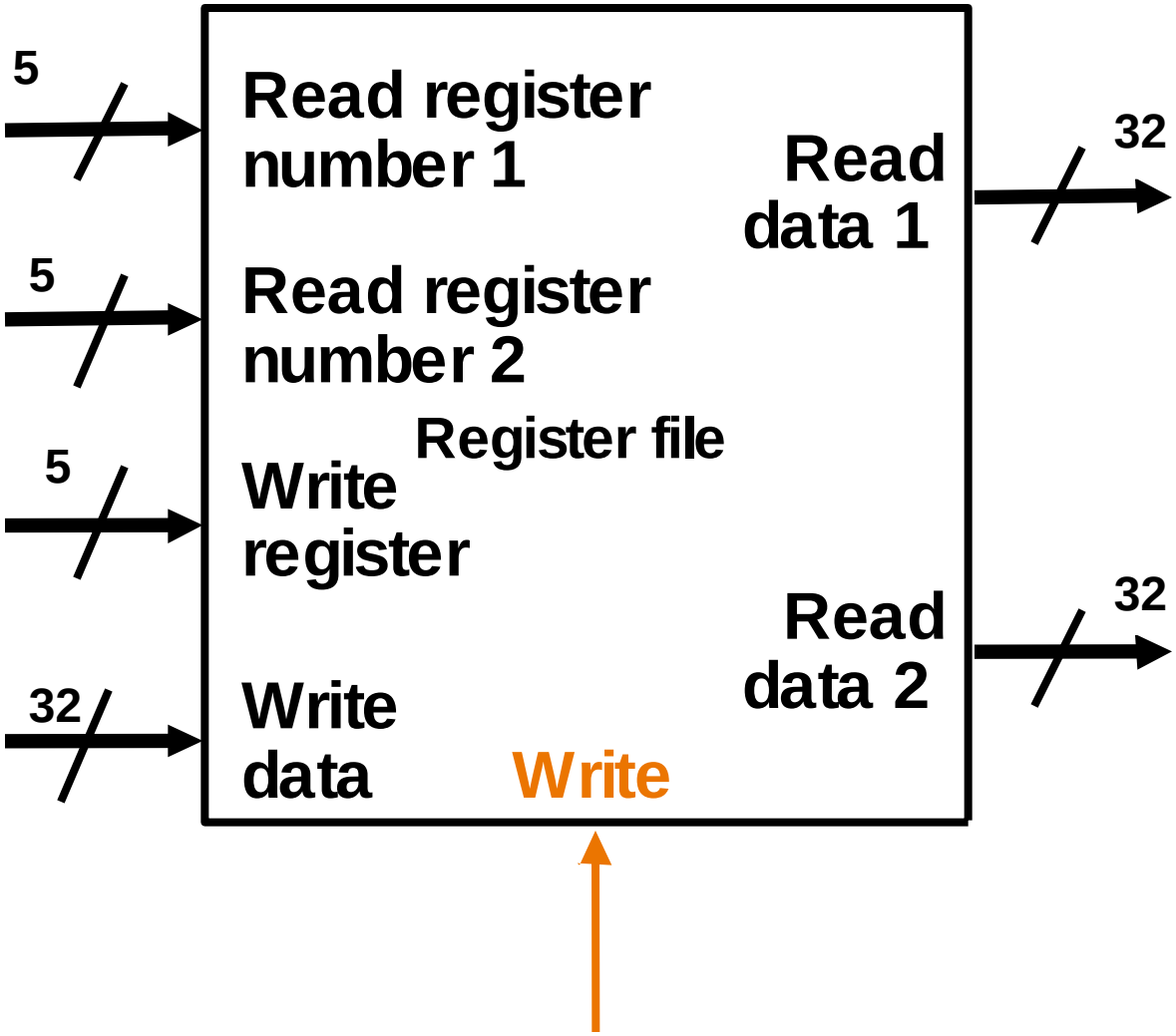
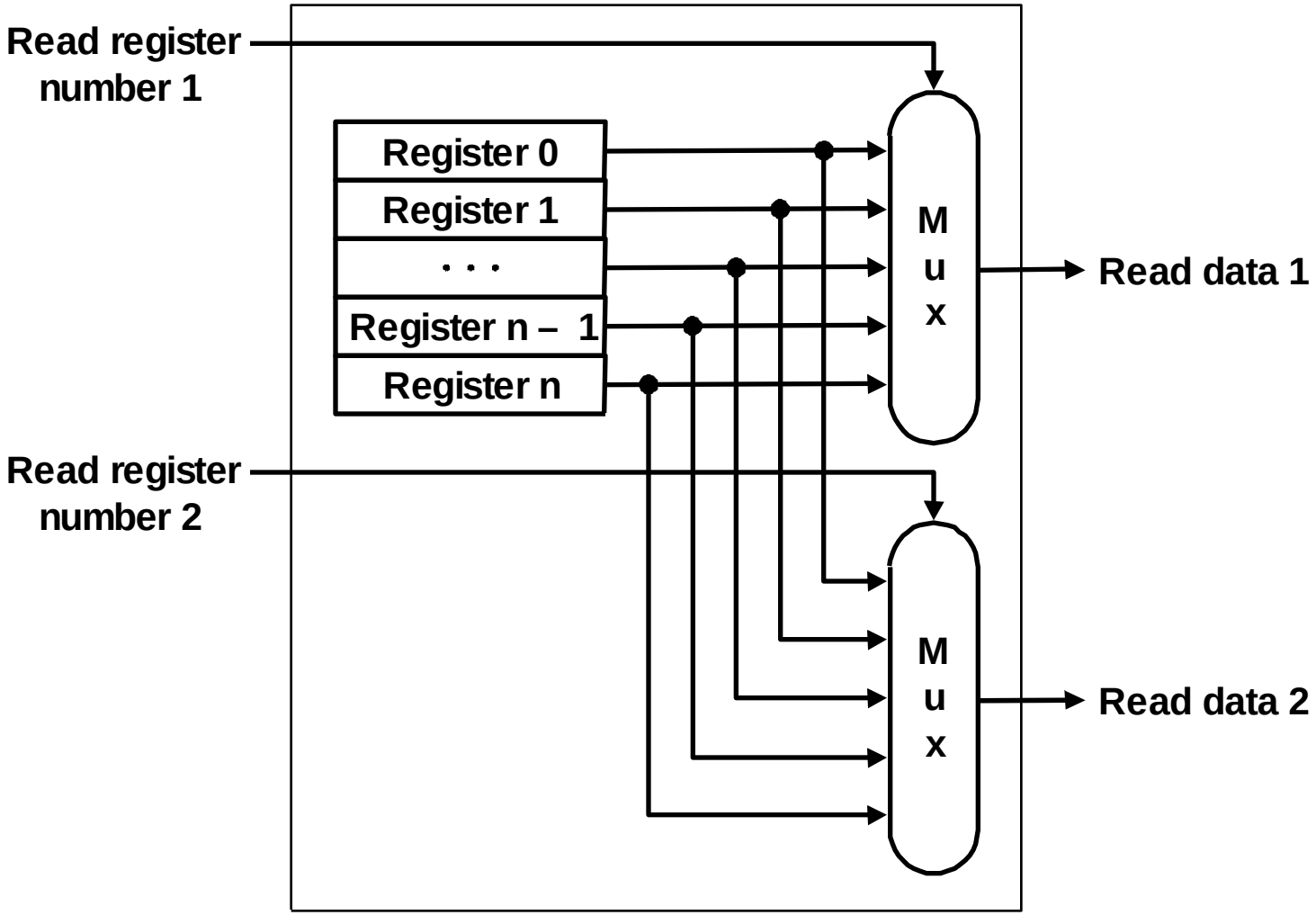
CS 220

The Processor
Single Cycle Datapath
4.1 – 4.4

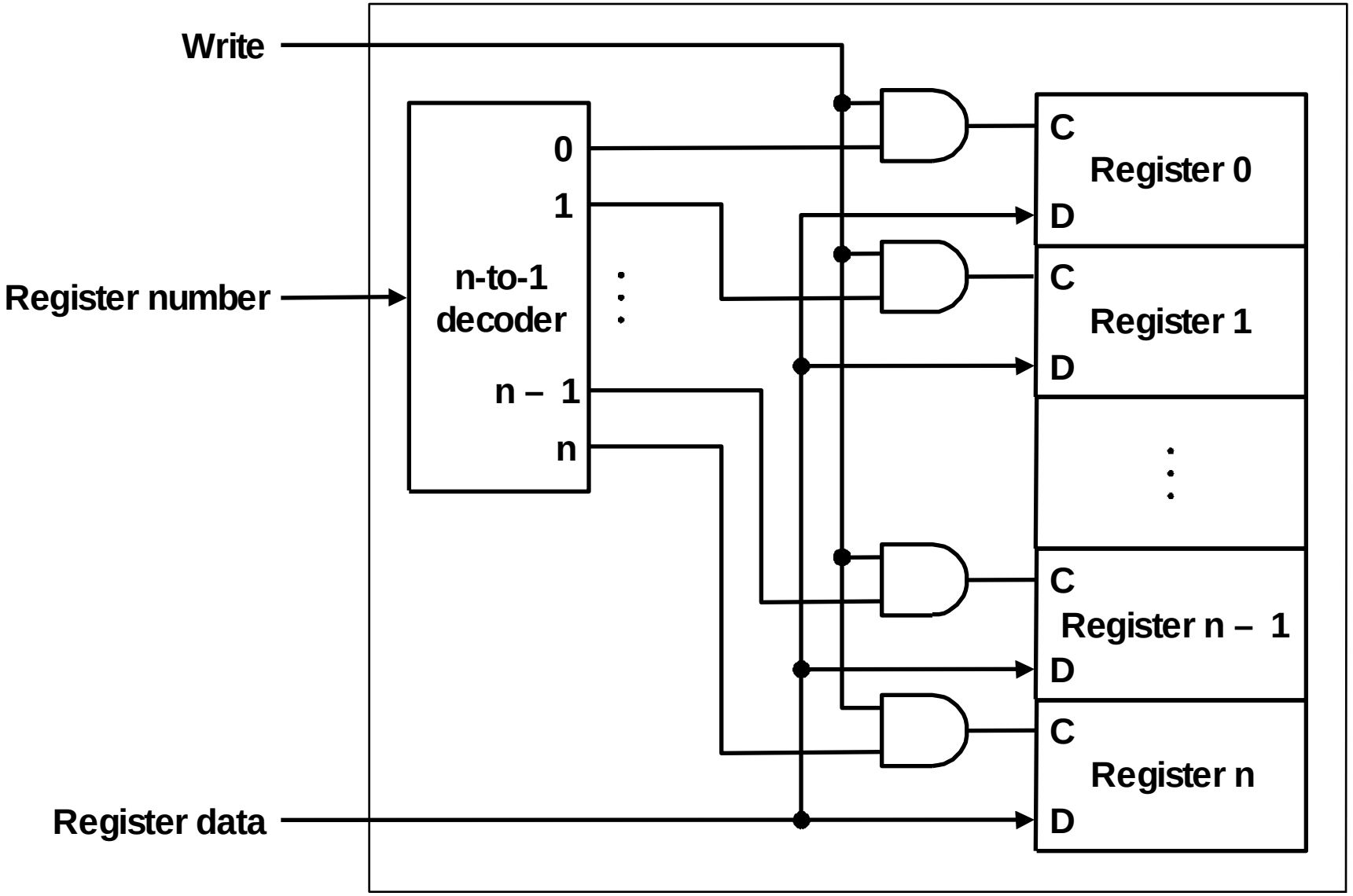
Register Files



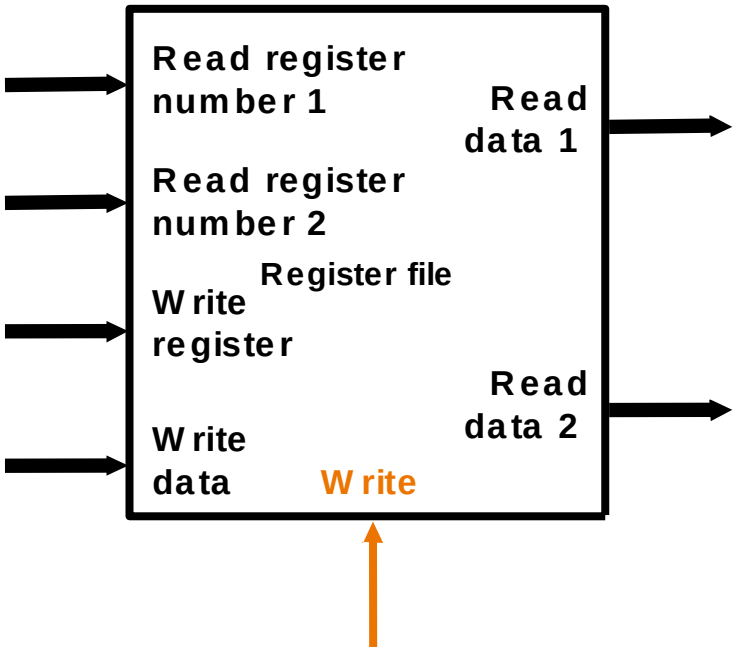
Register Files: The Read Ports

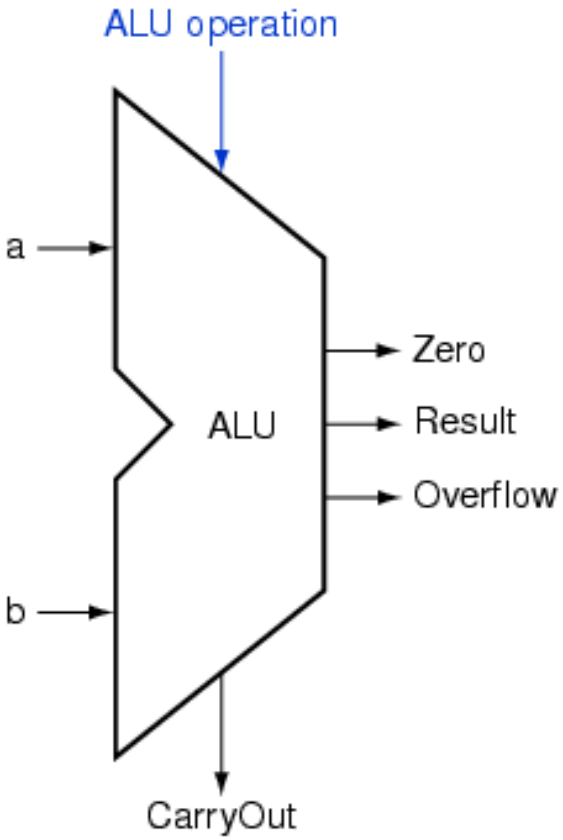


Register Files: The Write Port



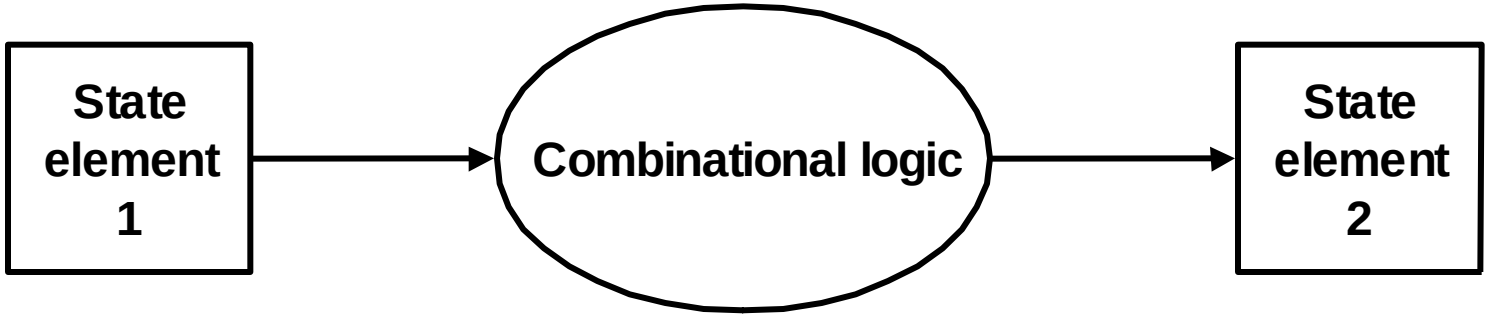
Register Files





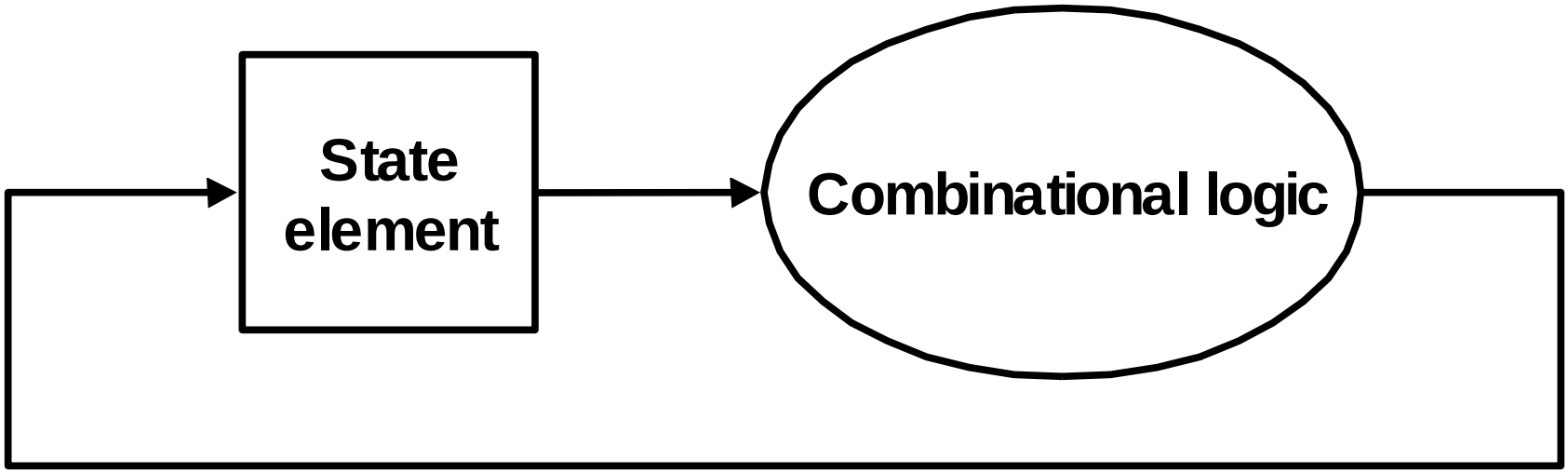
Sequential Logic

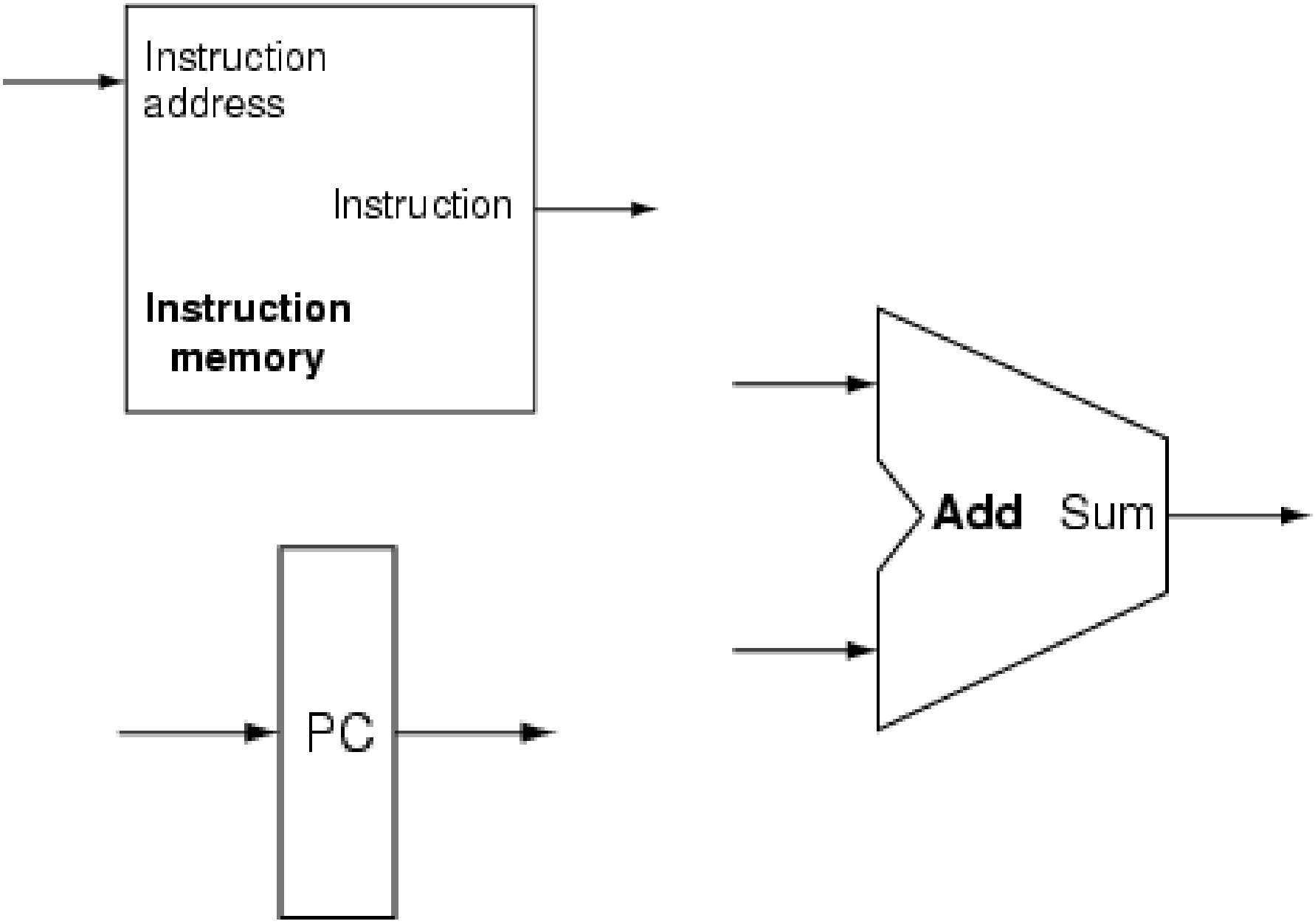
- Usually use **edge triggered** timing
- Changes appear on the rising or falling edge.
  - Assume changes ready on next rising edge



Sequential Logic

- Can the same **state element** be read and written at the same time?
- **`add $t0, $t0, $t0`**

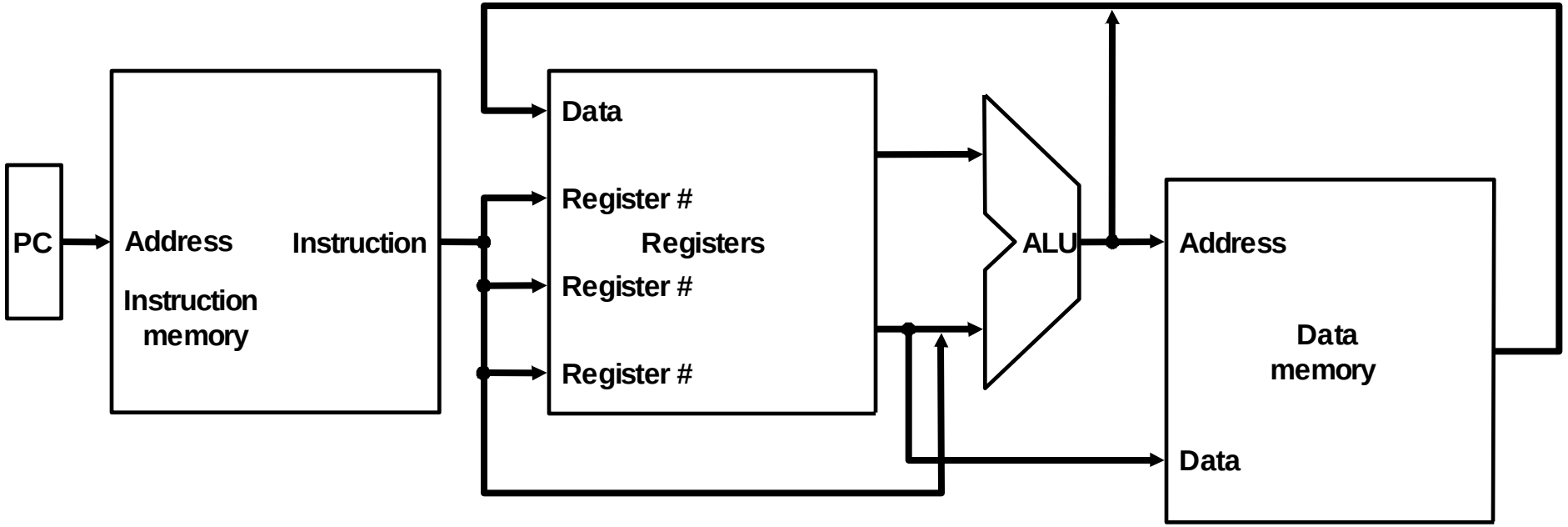



Instruction address
Instruction
Instruction memory
Add
Sum
PC

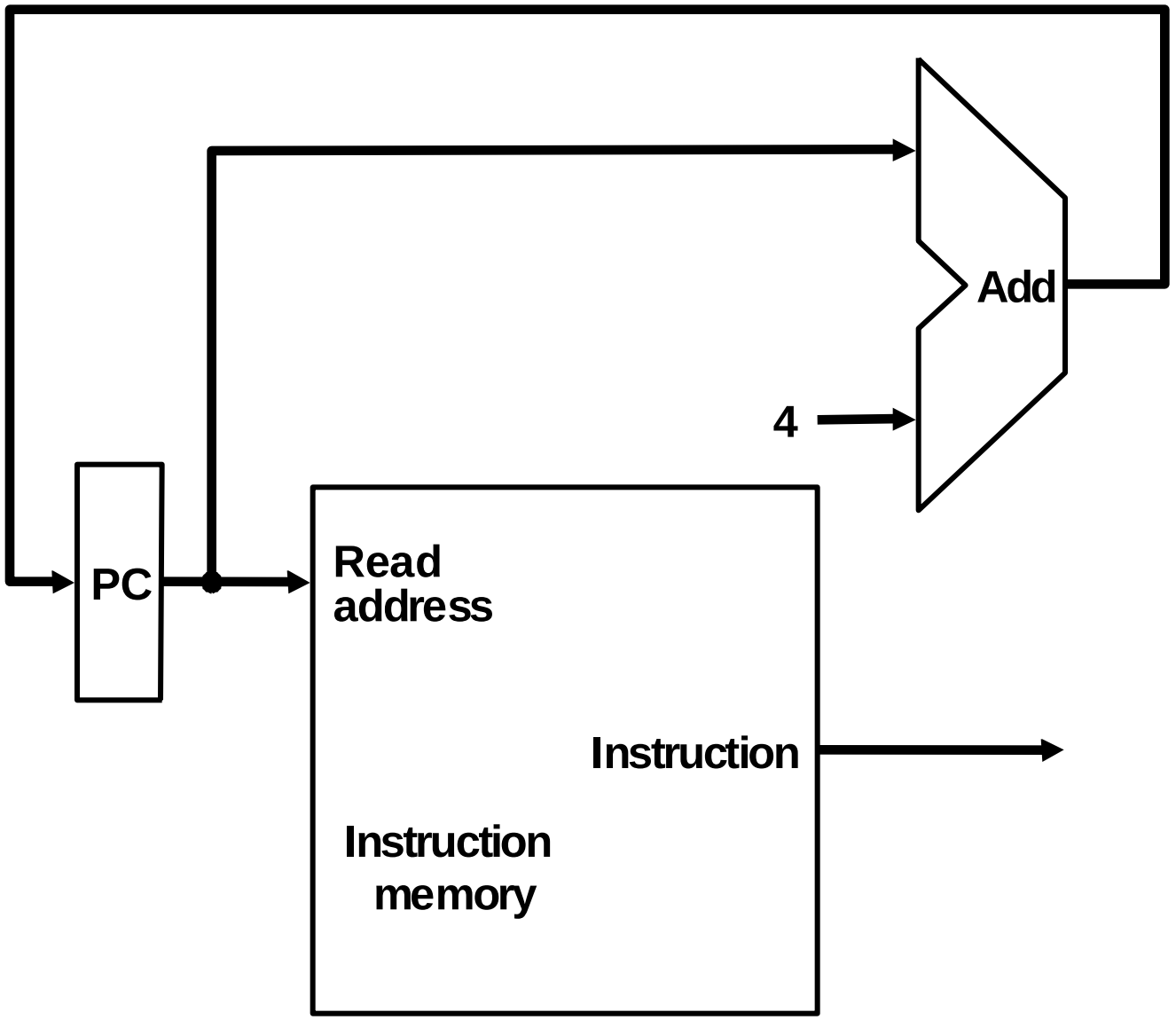
MIPS Subset

- Memory instructions lw/sw
  - **`lw $t0, 4($sp)`**
- Simple arithmetic add, sub, slt, ...
  - **`add $t0, $t1, $t2`**
  - **`addi $t0, $t1, 234`**
- Branches beq, j
  - beq $zero, $s0, offset
  - j address

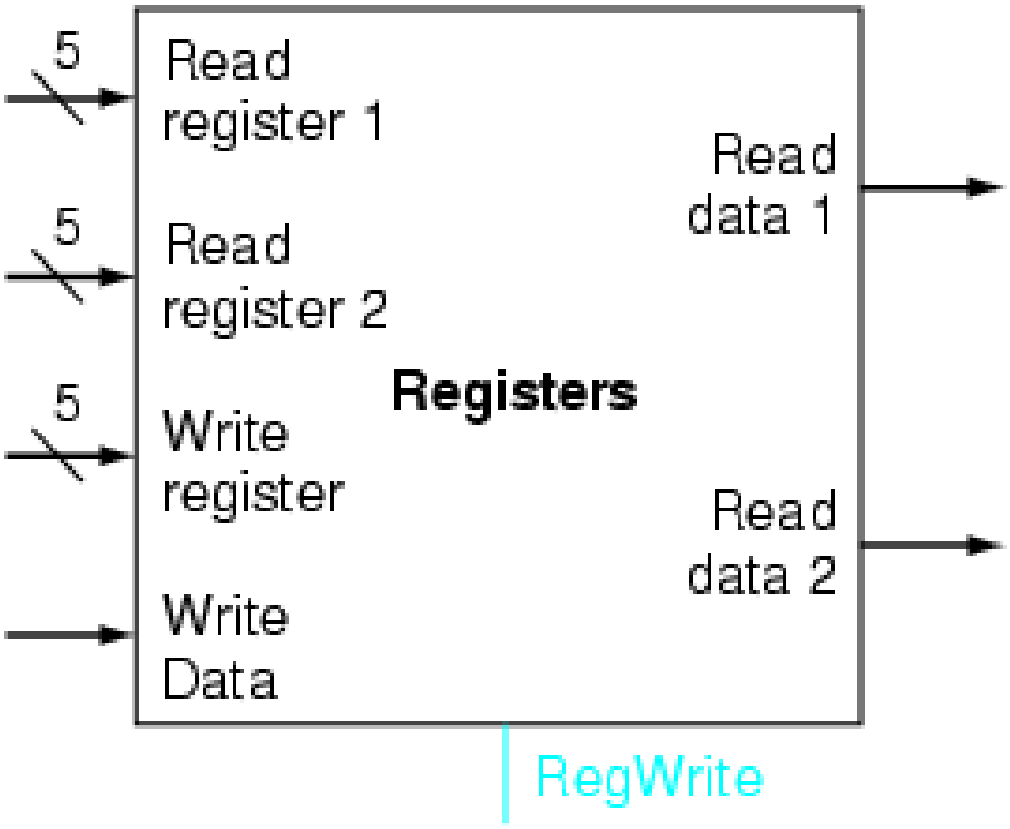
Abstract View of MIPS processor



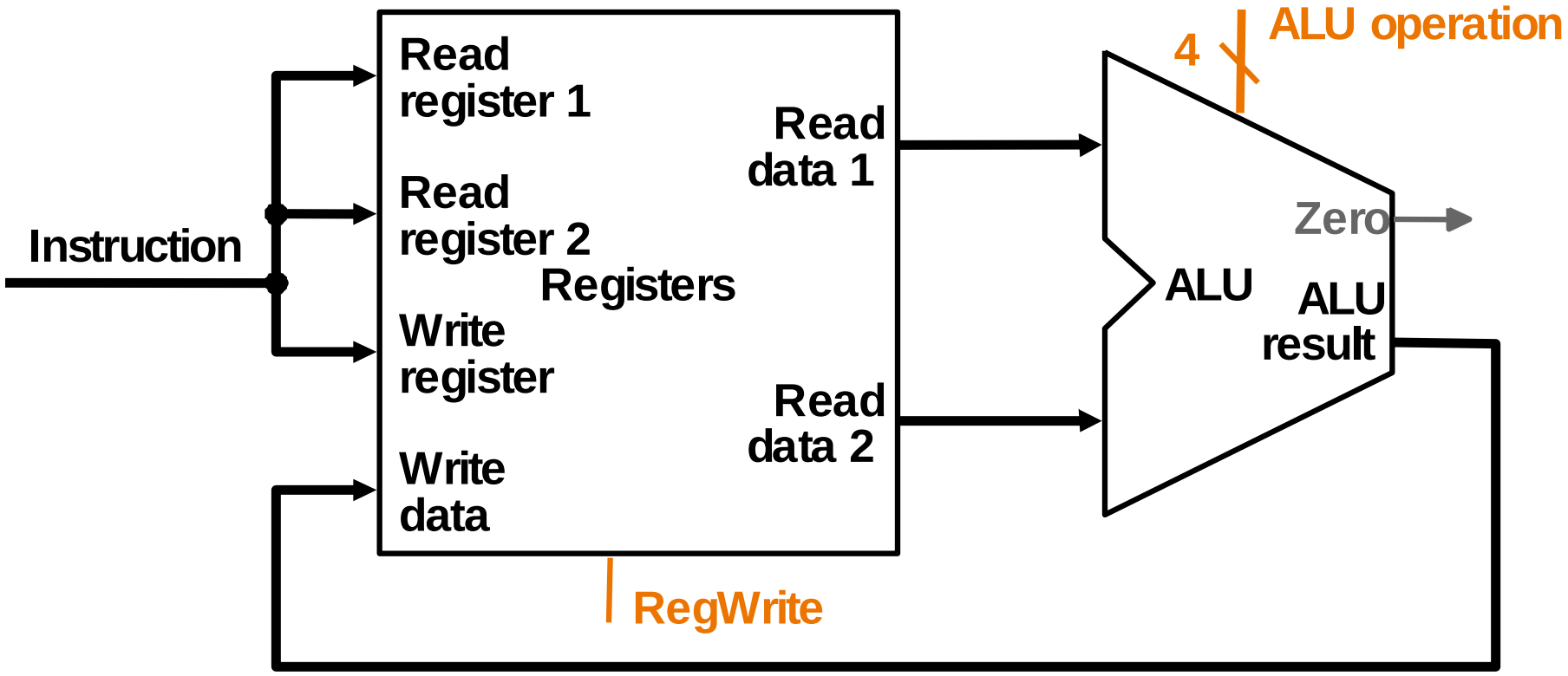
Fetching instruction from Memory



Recall the MIPS Registers



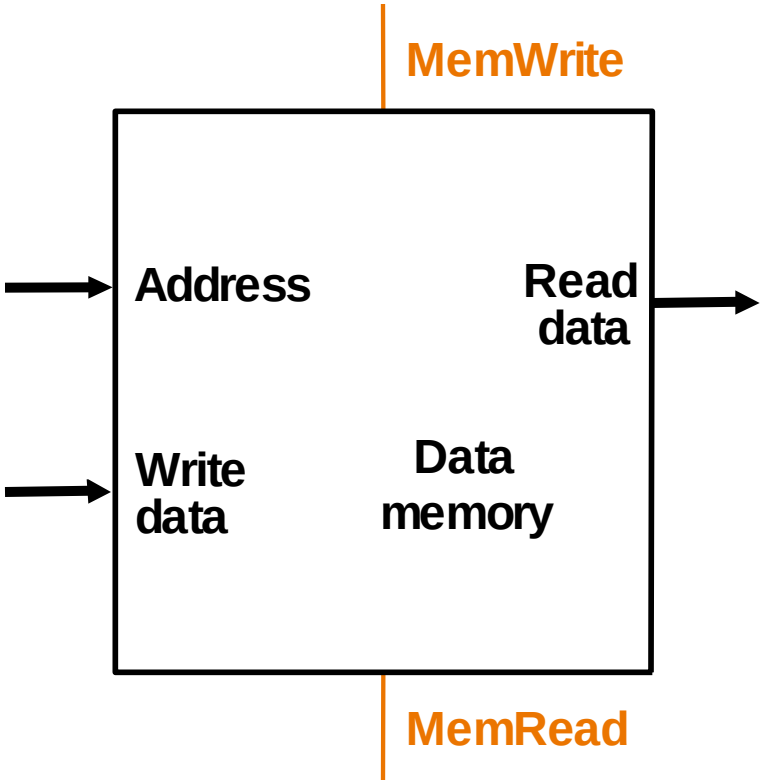
Executing R-Type Instructions



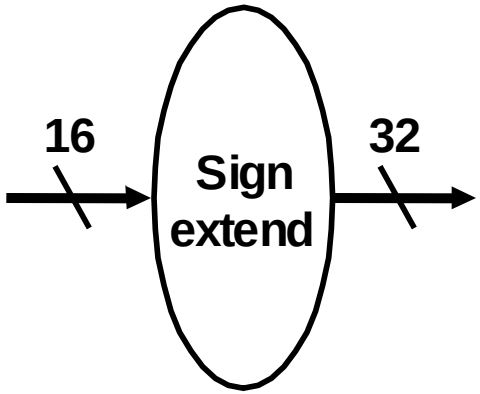
Executing Load/Store instructions

- A load/store does a
  1. register access then
  2. a memory address calculation
  3. read or write from memory
- Load/Store Instruction format is an I-format
  - lw $1, 100($2)
  - 16 bit signed offset is added to register

Load/Store Instructions

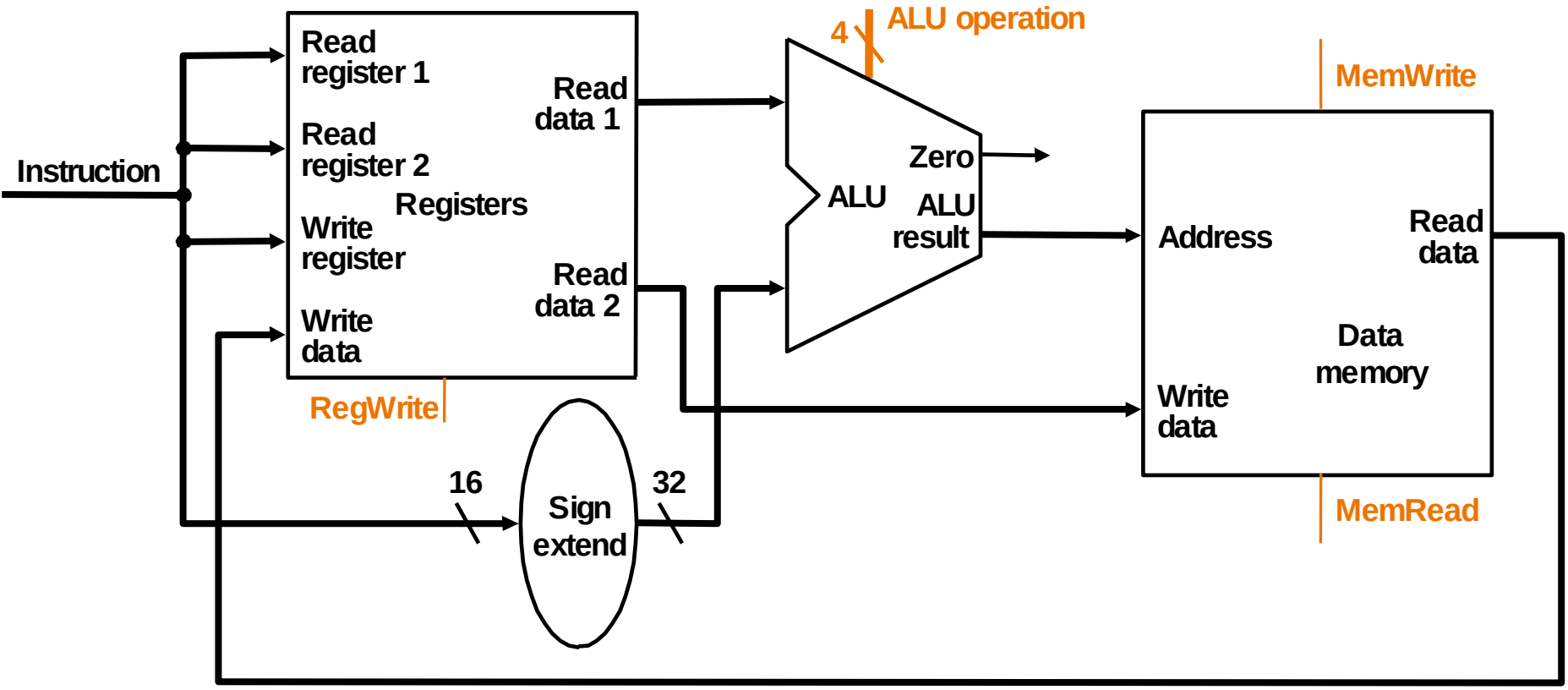


a. Data memory unit



b. Sign-extension unit

Datapath for Load/Store Instructions



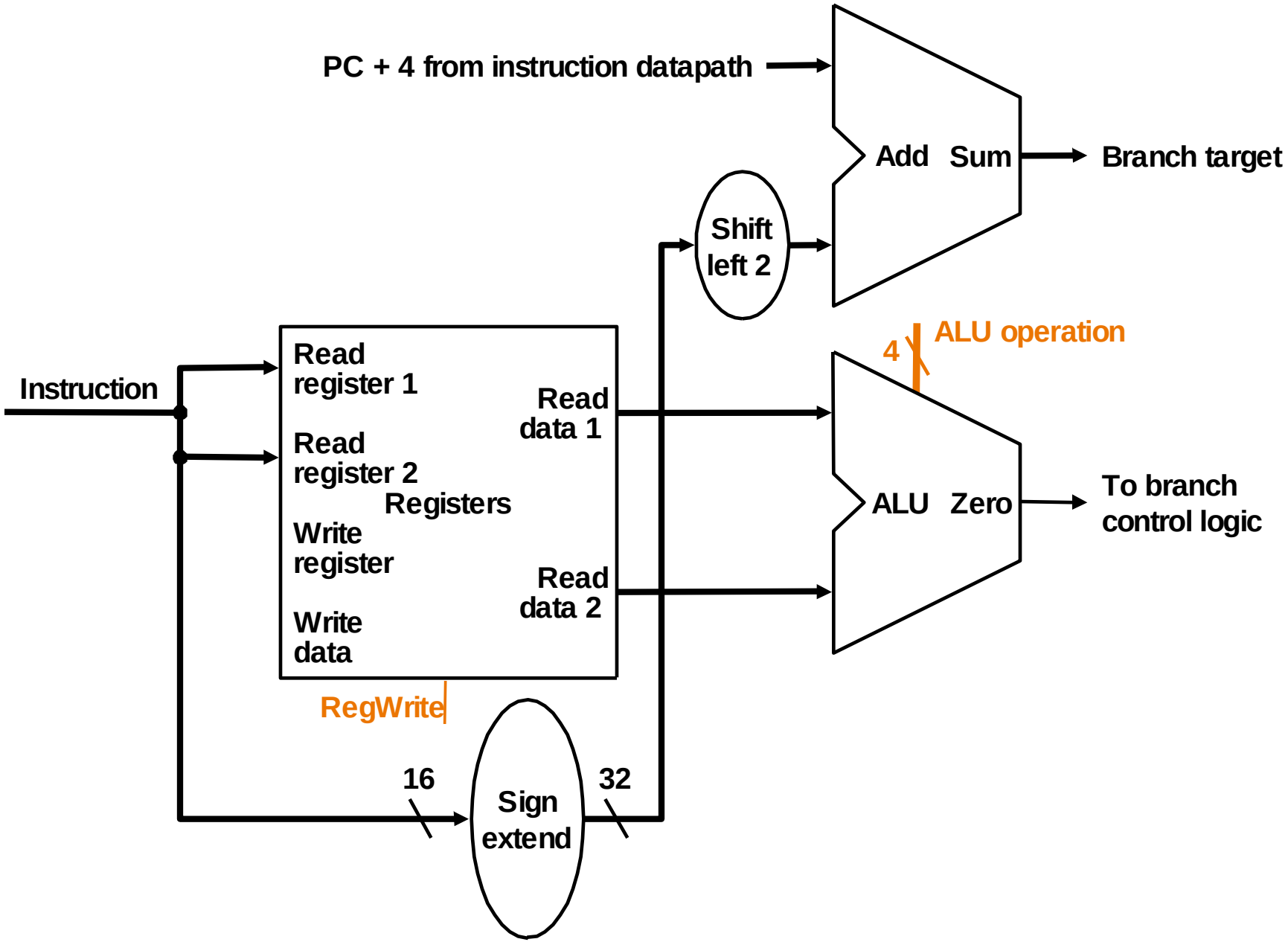
PC-relative addressing

```
loop: addi $s0, $s0, -1
      bne $s0, $zero, loop
```

Gets translated to

```
loop: addi $16, $16, -1
      bne $16, $0, _____
```

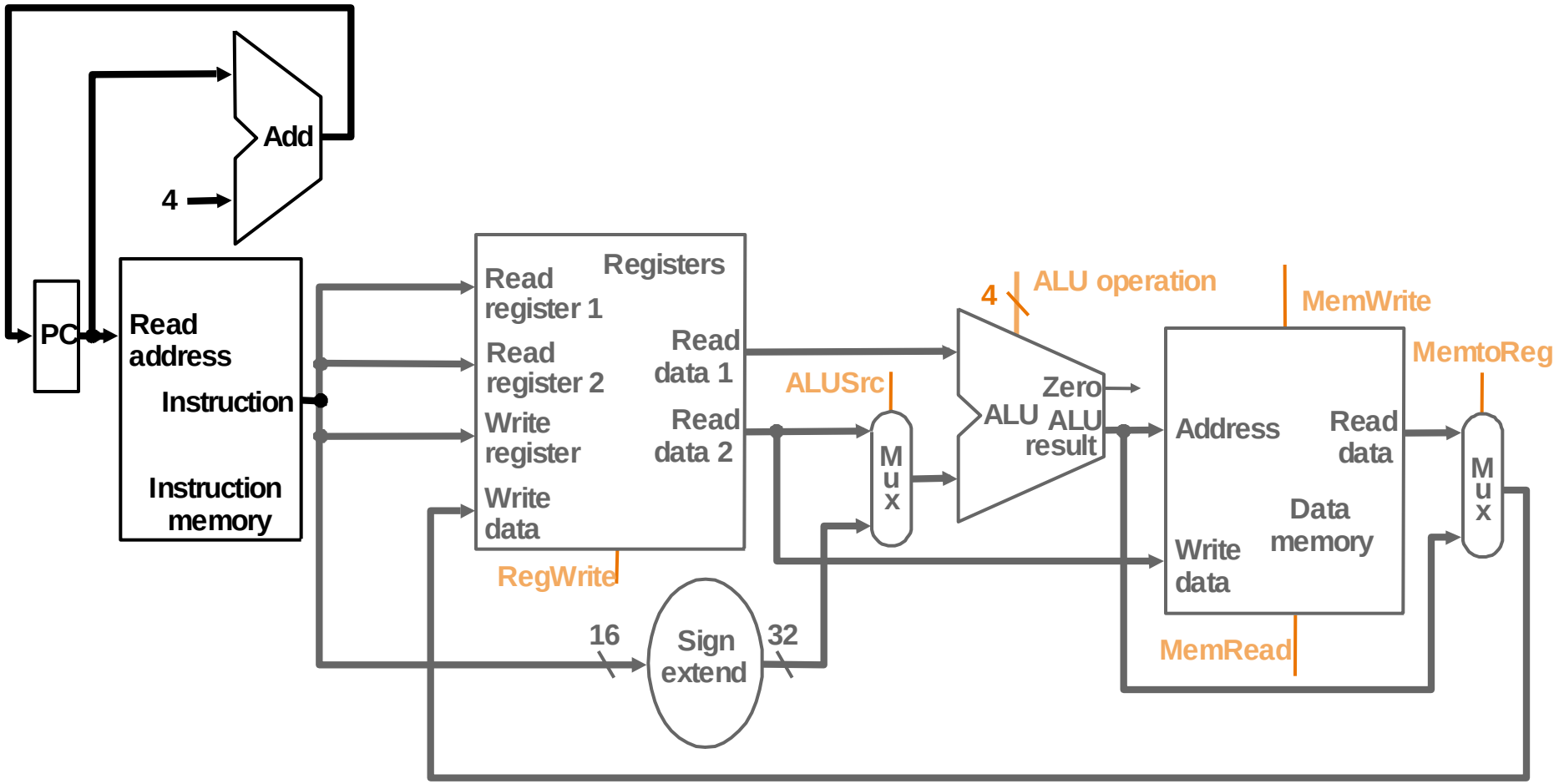
Datapath for branch instructions



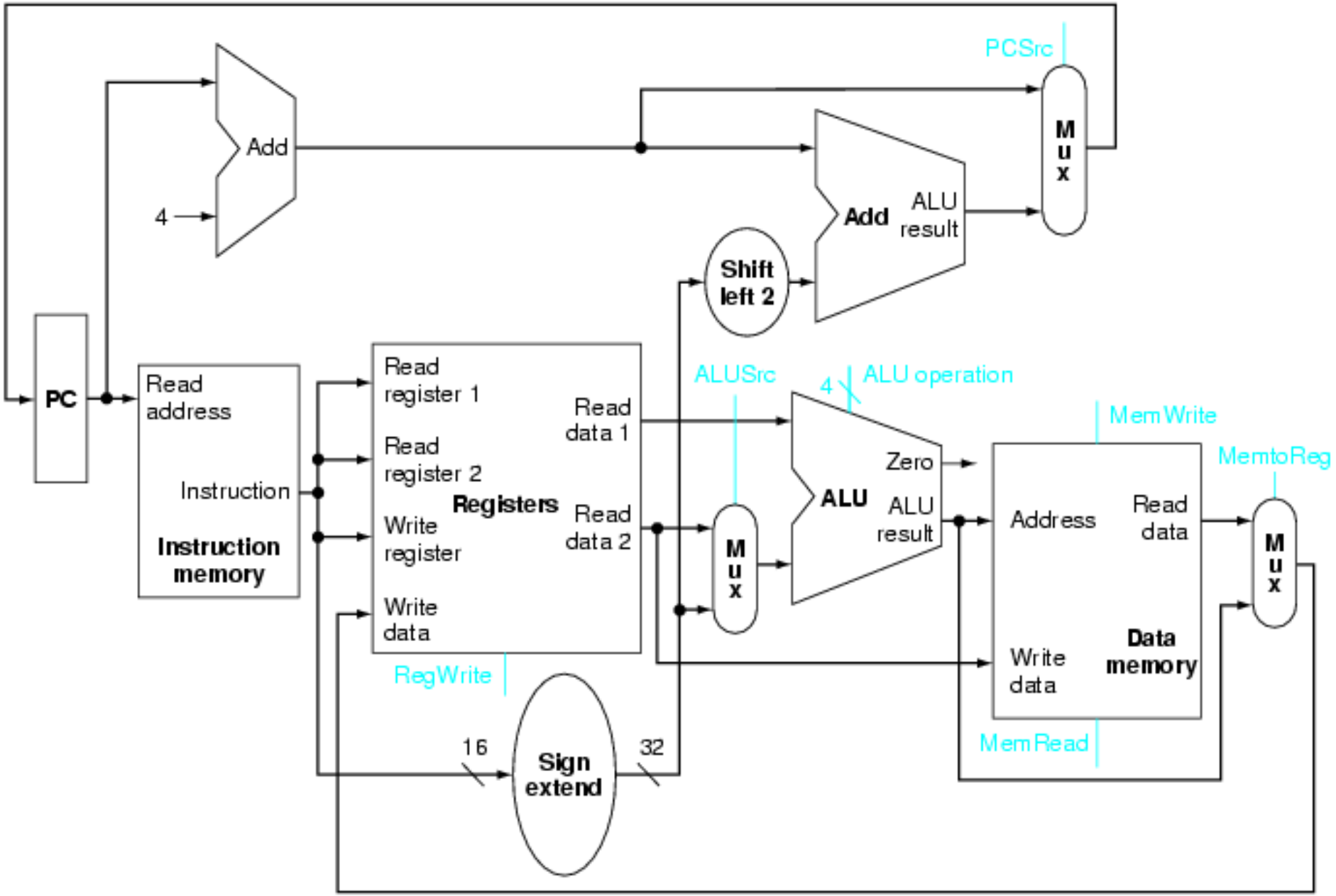
Combining Datapaths

- We have built datapaths for
  - R-type instructions
  - and Some I-type instructions
    - lw/sw instructions
    - beq
- Need to combine these
  - into a single datapath
  - allows us to reuse registers, ALU, etc.

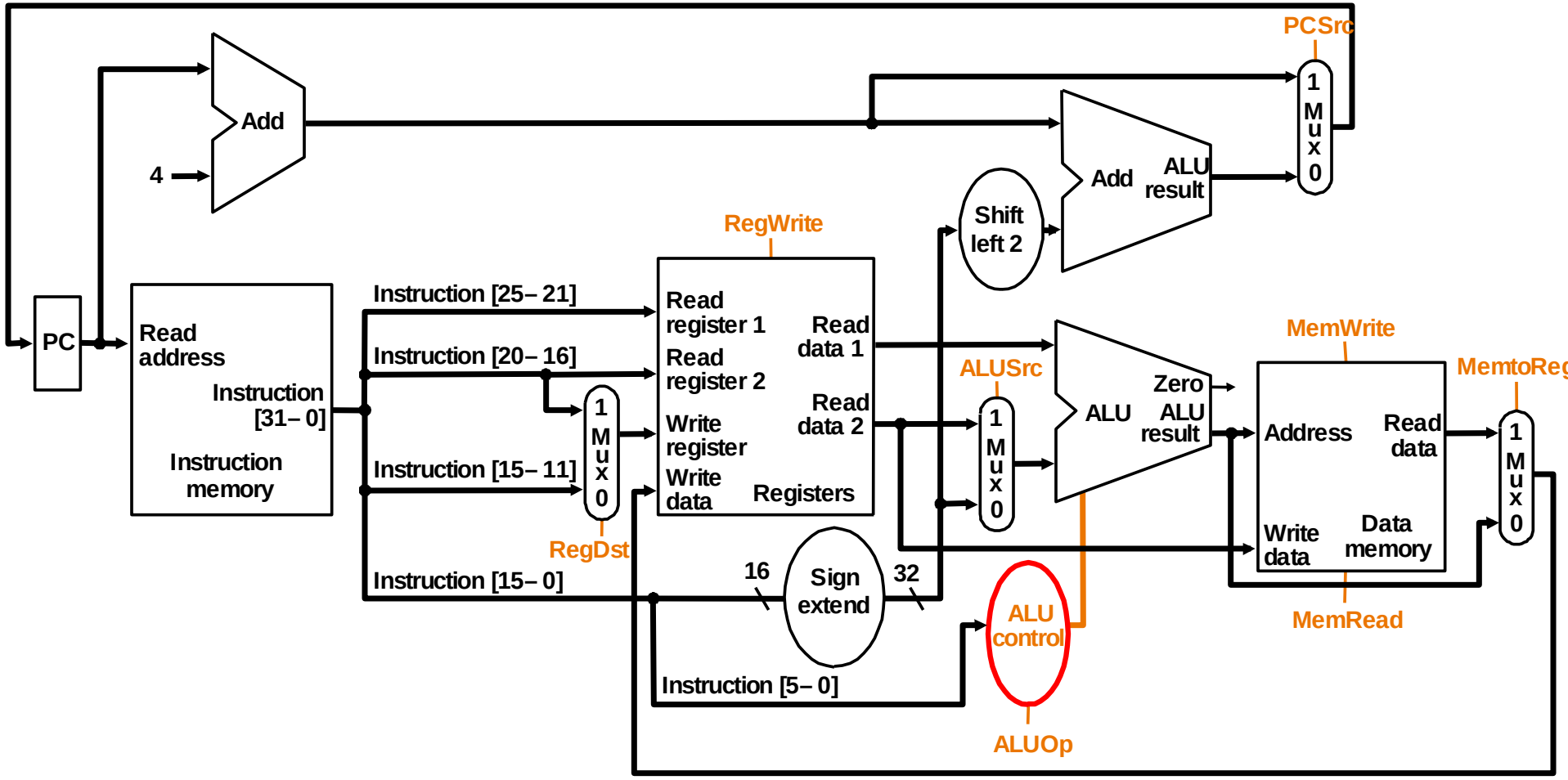
Combining Datapaths (R and lw/sw)



R type + lw/sw + branch



Controlling the ALU (page 316)



ALU Control (page 316)

- AND 0000
- OR 0001
- add 0010
- subtract 0110
- set on less than 0111
- NOR 1100

Controlling the ALU

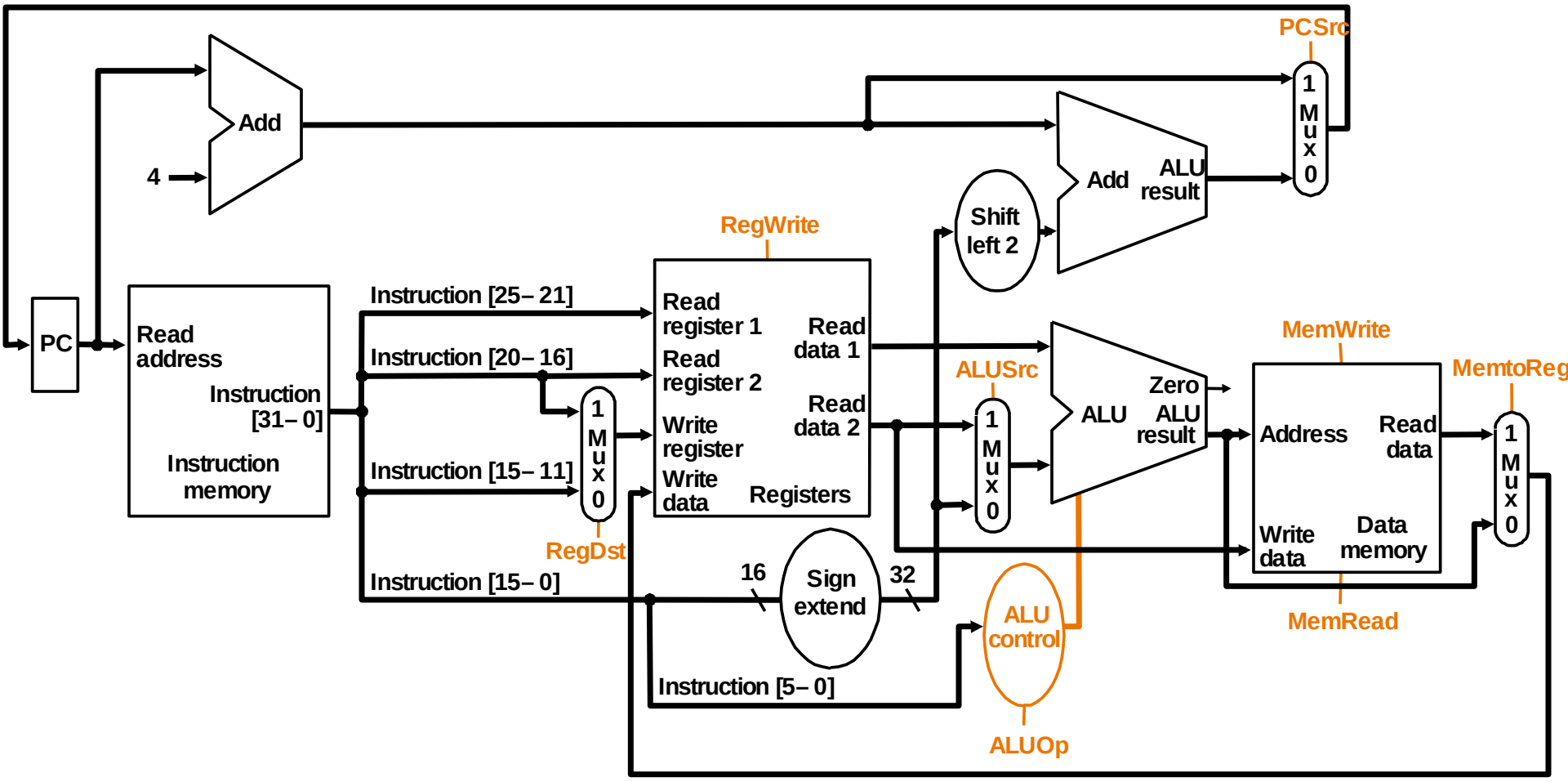
Opcode	ALUop	Oper-ation	Funct field	ALU Action	ALU Control
lw	00	lw	X		
sw	00	sw	X		
beq	01	beq	X		
R	10	Add	100000		
R	10	Sub	100010		
R	10	And	100100		
R	10	Or	100101		
R	10	Slt	101010		

Controlling the ALU: truth table

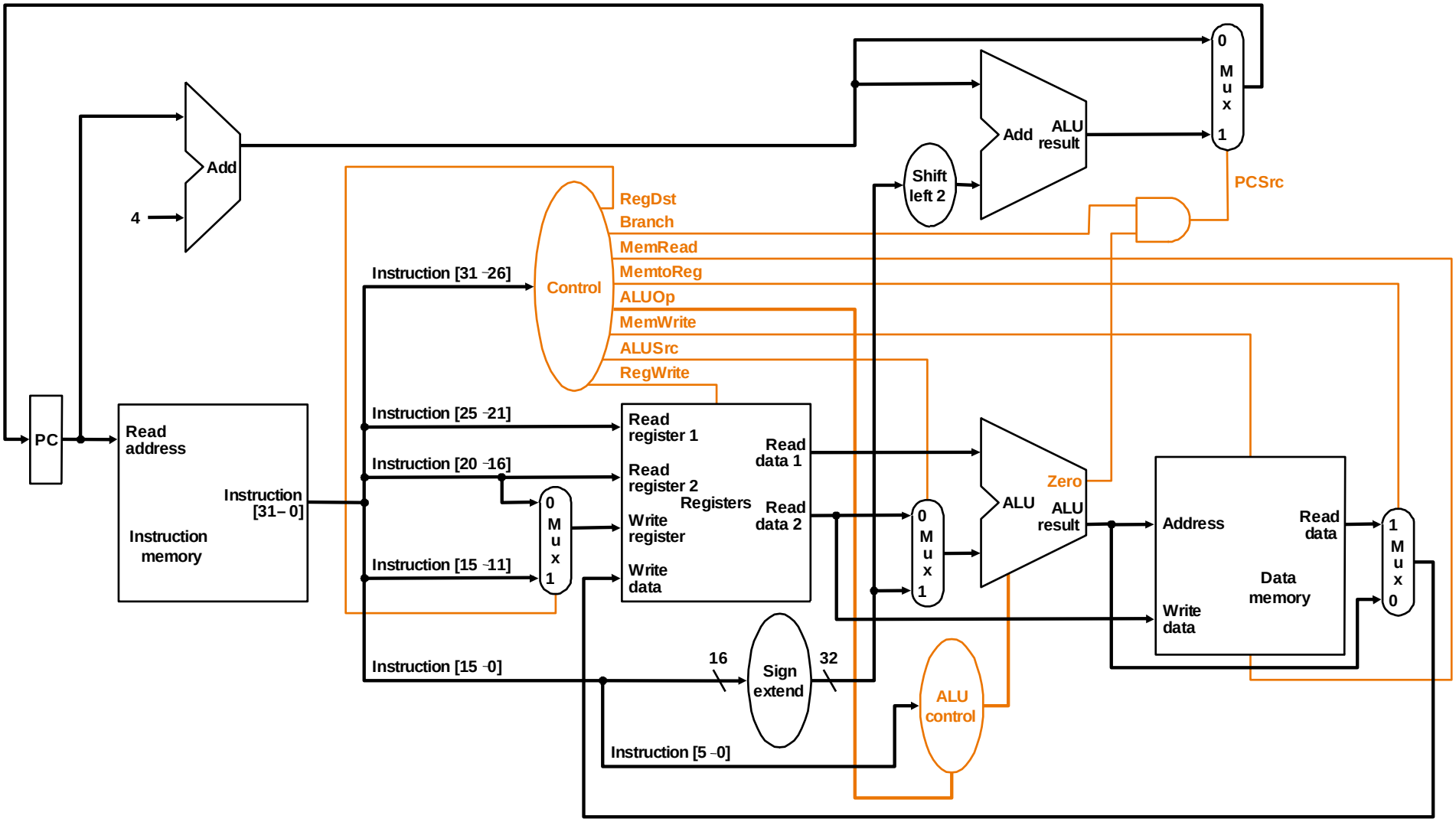
ALUop		Funct Field						ALU Operation
Op_1	Op_0	f_5	f_4	f_3	f_2	f_1	f_0	ALU_3 ALU_2 ALU_1 ALU_0
0	0	x	x	x	x	x	x	0010
0	1	x	x	x	x	x	x	0110
1	0	x	x	0	0	0	0	0010
1	0	x	x	0	0	1	0	0110
1	0	x	x	0	1	0	0	0000
1	0	x	x	0	1	0	1	0001
1	0	x	x	1	0	1	0	0111

The Main Control Unit

We are left with 9 control signals that need to be set to 0 or 1 on the basis of 6 remaining opcode bits.



The Main Control Unit



Designing the Main Control Unit

- What are the different opcodes we are supporting?
- See table page 308

Ins	Reg Dst	ALU Src	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R									
lw									
sw									
beq									

Can we write the logic equations?