Computer Engineering Department

CC 311- Computer Architecture

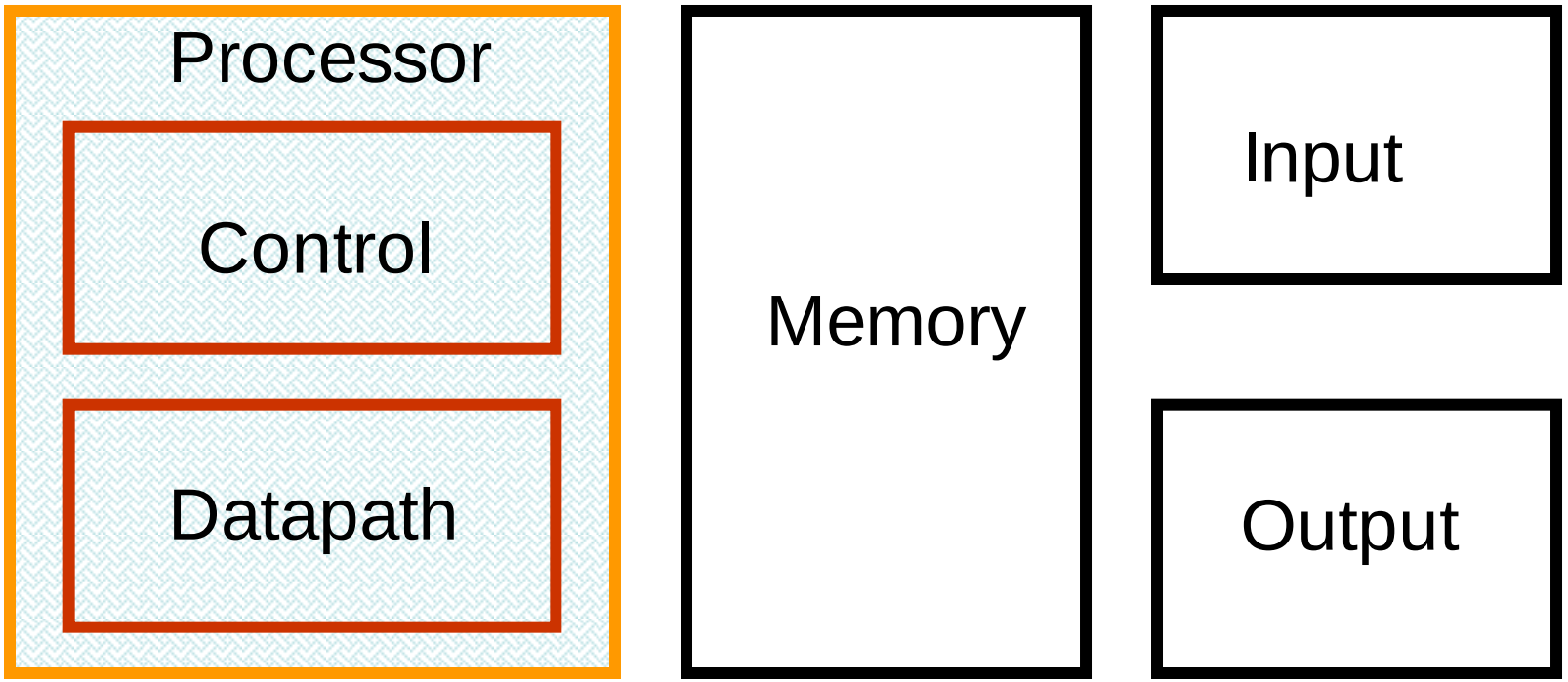
Chapter 4

The Processor: Datapath and Control

"Single Cycle"

Introduction

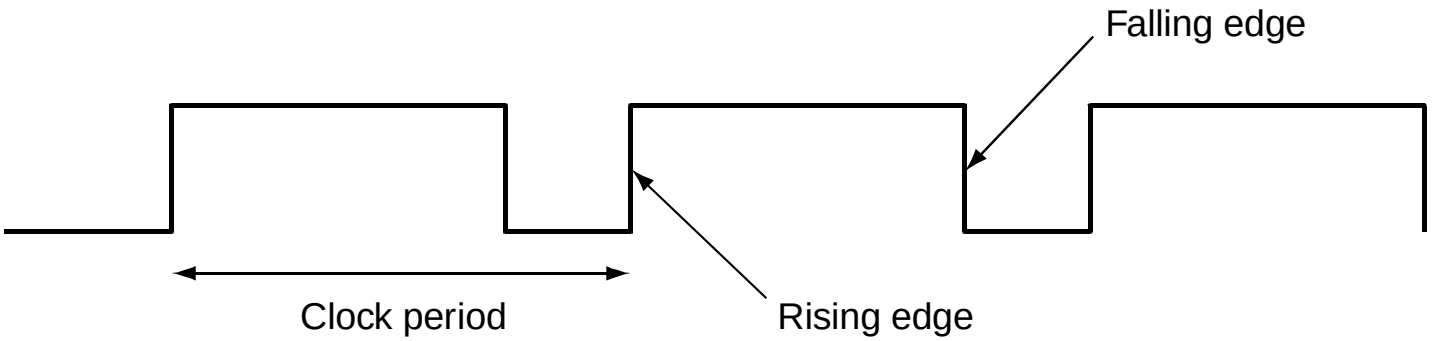
- The 5 classic components of a computer



- Chapter 4 discusses the processor
  - Part-a: Datapath
  - Part-b: Control

Review: Edge-triggered Methodology

- Input:
  - Values written in a previous clock cycle
- Output:
  - Values to be used in the following clock cycle
- Prevents reading the signal in the same time it is written
- More than one action can take place in the same clock cycle

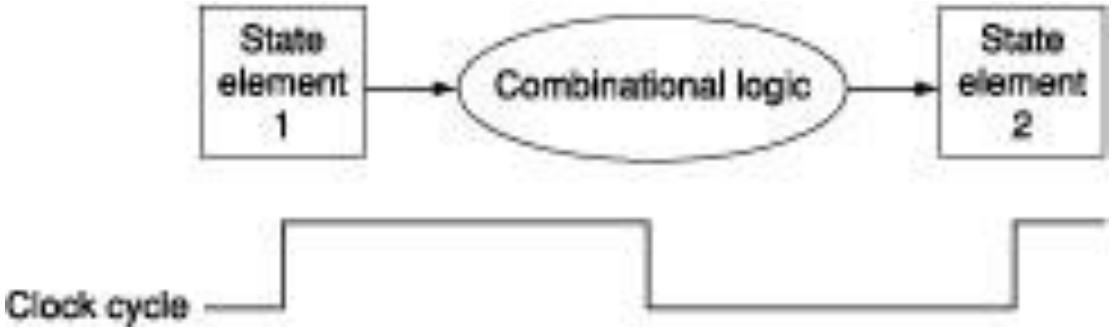


Review: Timing Methodology

- Clock cycle(Tick/Period):
  - Time for one clock period, usually of the processor, which runs at a constant rate
- Memory Access time:
  - Time between the initiation of a read request and when the desired word arrives
- Memory Cycle time:
  - Minimum time between requests to memory
  - Should be greater than access time to keep address line stable between accesses

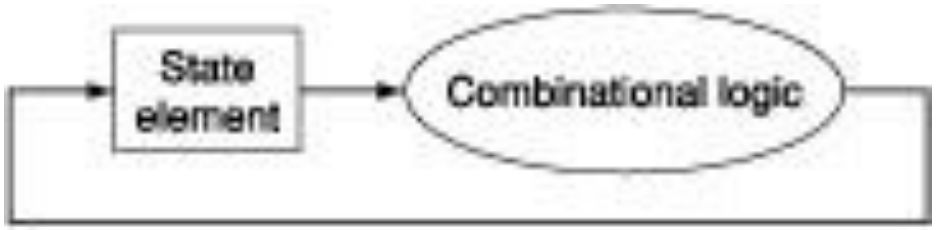
Review: Timing Methodology

- Typical execution cycle:
  - Read contents of some state elements,
  - Send values through some combinational logic
  - Write results to one or more state elements
  - Clock period should cover all these activities
- All signals must propagate from state element1 to state element2 in the time of one clock cycle
- If the state element is not updated on every clock, an explicit write control signal is required, in which case the state element is changed only when the control signal is asserted and the clock edge occurs

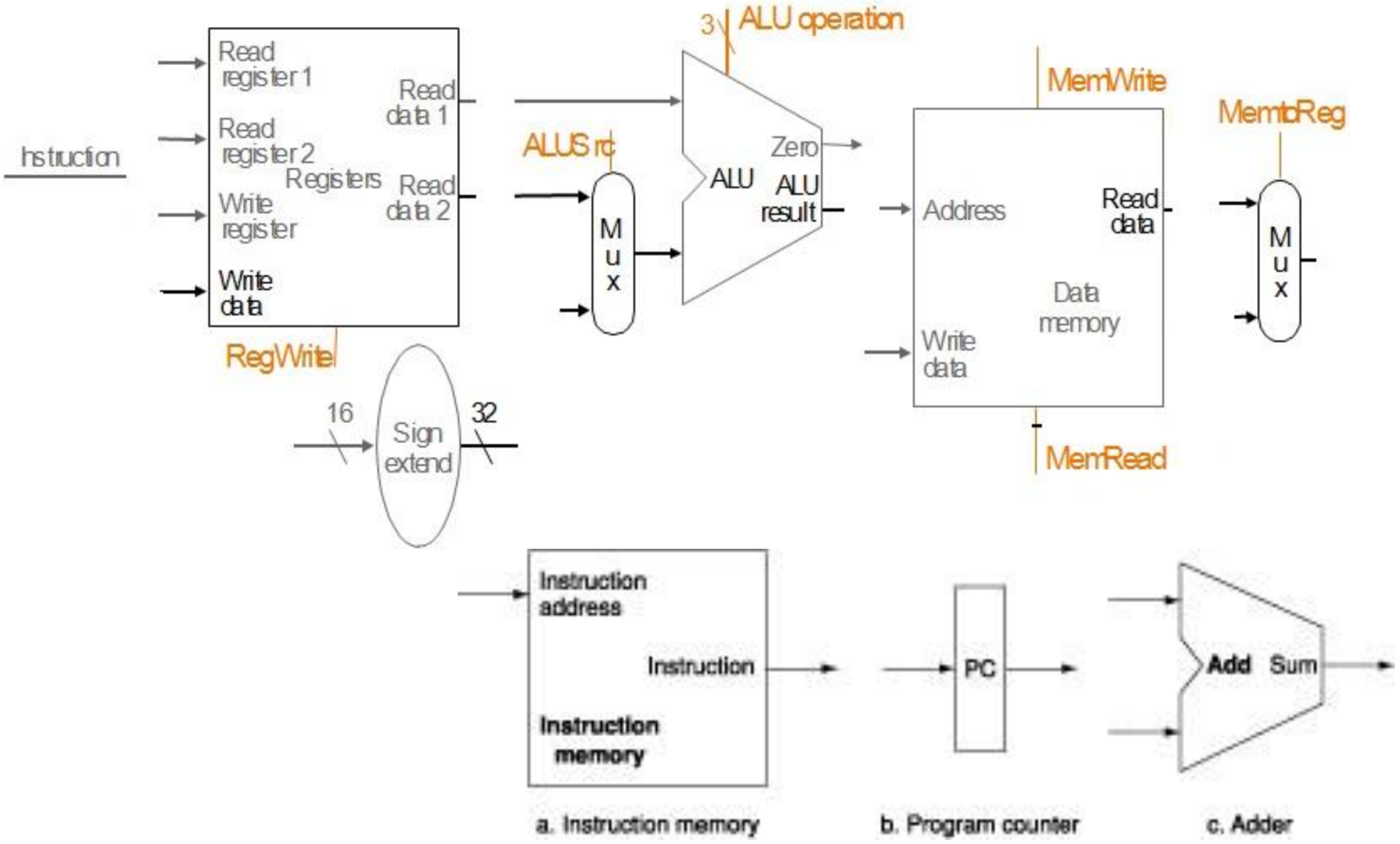


Review: Timing Methodology

- Edge triggered methodology allows a state element to be read and written in the same clock cycle
- The clock must be long enough to allow the stability of input value before the active edge occurs



Building the Datapath



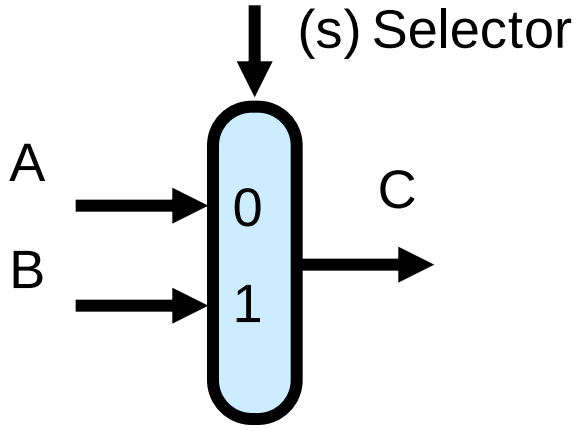
Review: 2 x 1 MUX

- MUX Operation:
  - Selects one of the (A or B) inputs to be the output, based on a control (select) input (S)
  - We need a 2x1 MUX
  - The MUX has 3 inputs
- Equivalent C-Operation

If (s == 0)

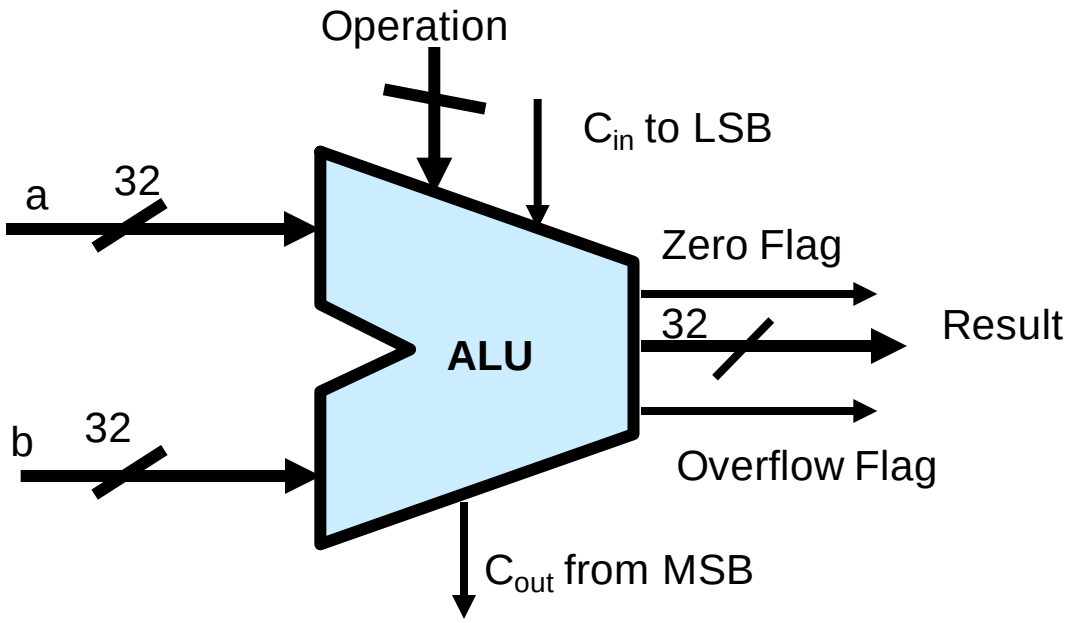
C:= A;

else C = B;



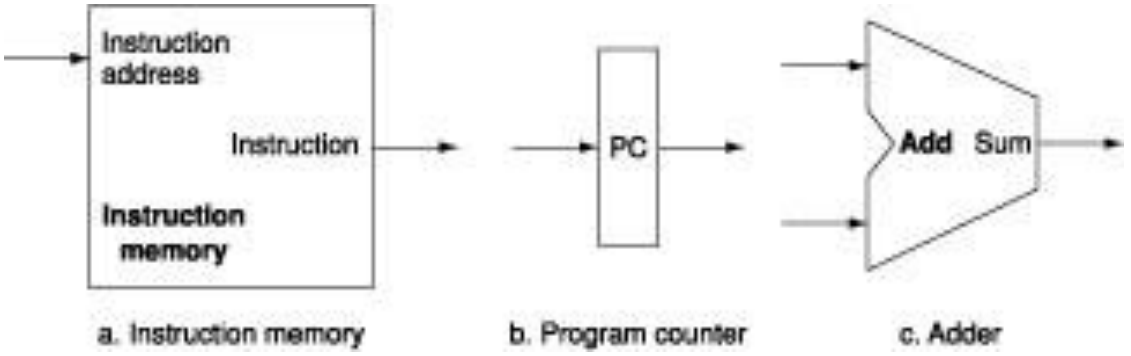
Universal ALU

- Symbolic representation:
  - (Sometimes used to represent adders as well)



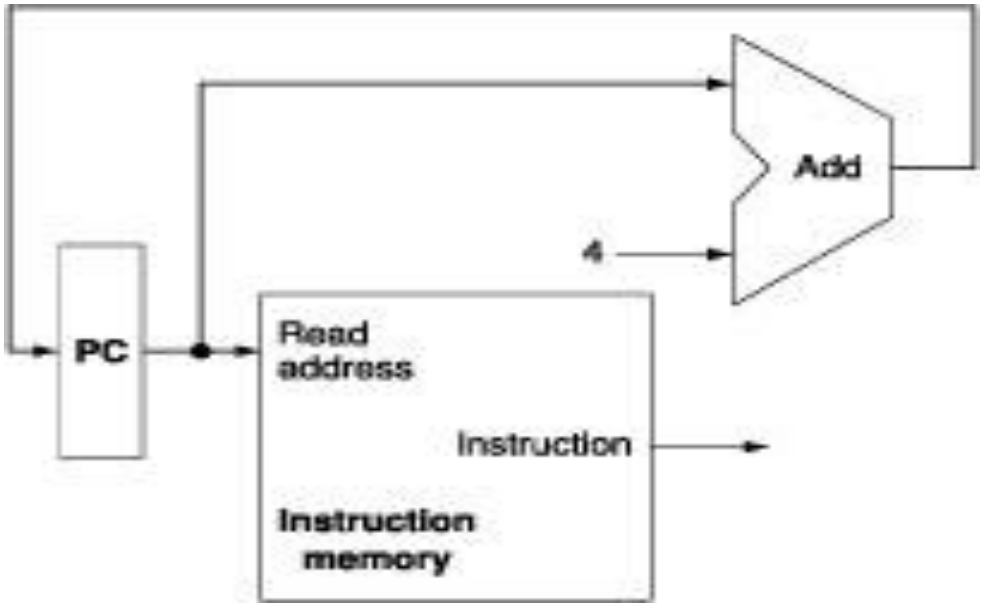
Building the Datapath

- The major components required to execute each class of MIPS instructions?
  - A memory unit to store the instructions
  - PC to store the address of the current instruction
  - Adder to increment PC to address of next instruction



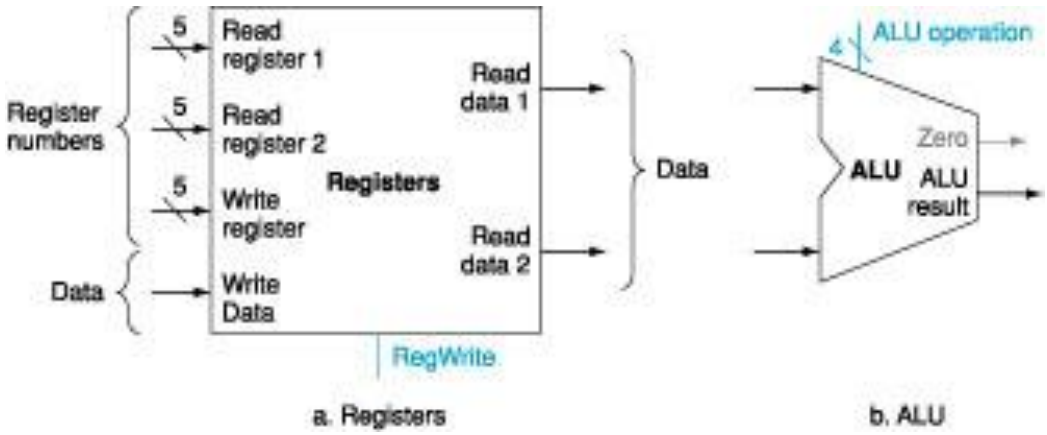
Building the Datapath

- How are these components connected?



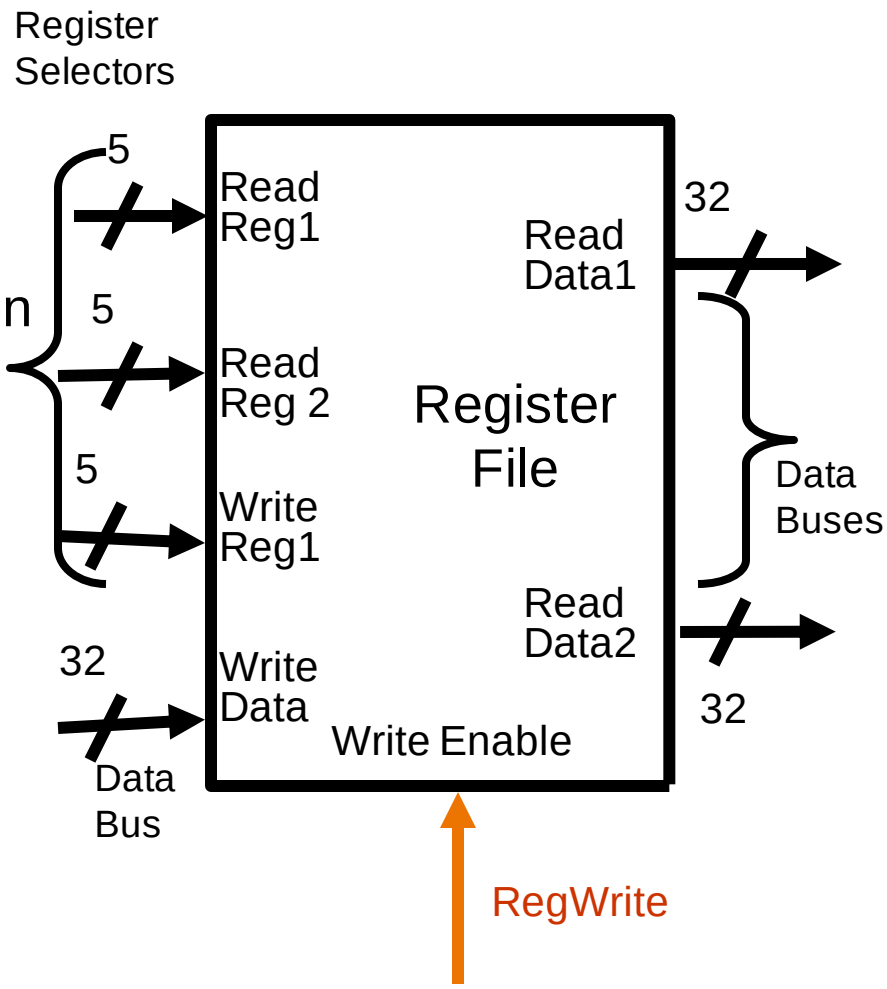
Building the Datapath

- The major components required to execute R-Format instructions
  - add, sub, and, or, slt
  - Three register operands
    - Two read ports to read two registers from register file
    - One write port to write into one register
  - Data lines 32-bit
  - Register lines
  - ALUOp



Review: Register File

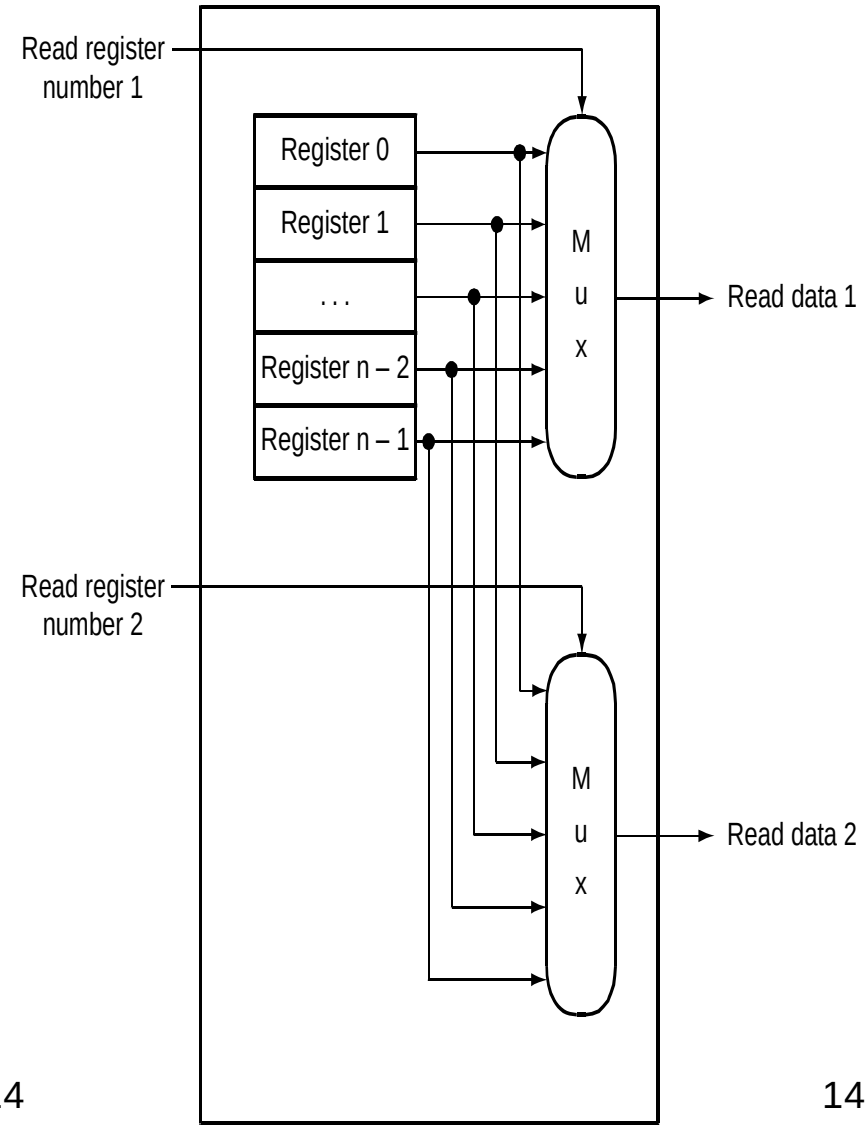
- A set of 32 registers
  - 5-inputs
    - 2 read-ports to supply source register numbers
    - 1 write-port to supply destination register number
    - 1 Data bus
    - 1 Register write enable control signal
  - 2-outputs
    - Data from register 1
    - Data from register 2
  - 1-Control signal
    - RegWrite
    - No need to read-enable



13

Review: Reading from Register File

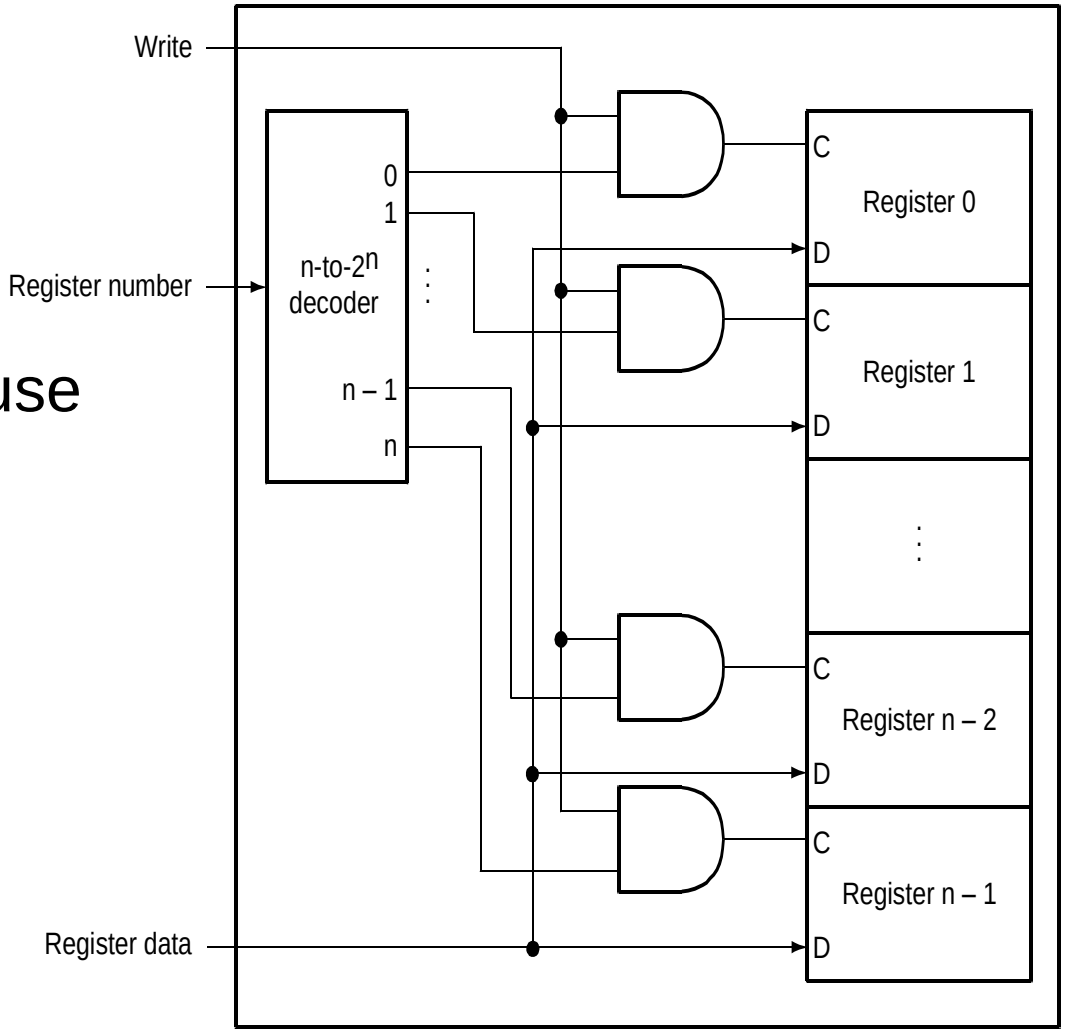
- Need to submit:
  - Register #
- For N-registers we need:
  - n x 1 MUX



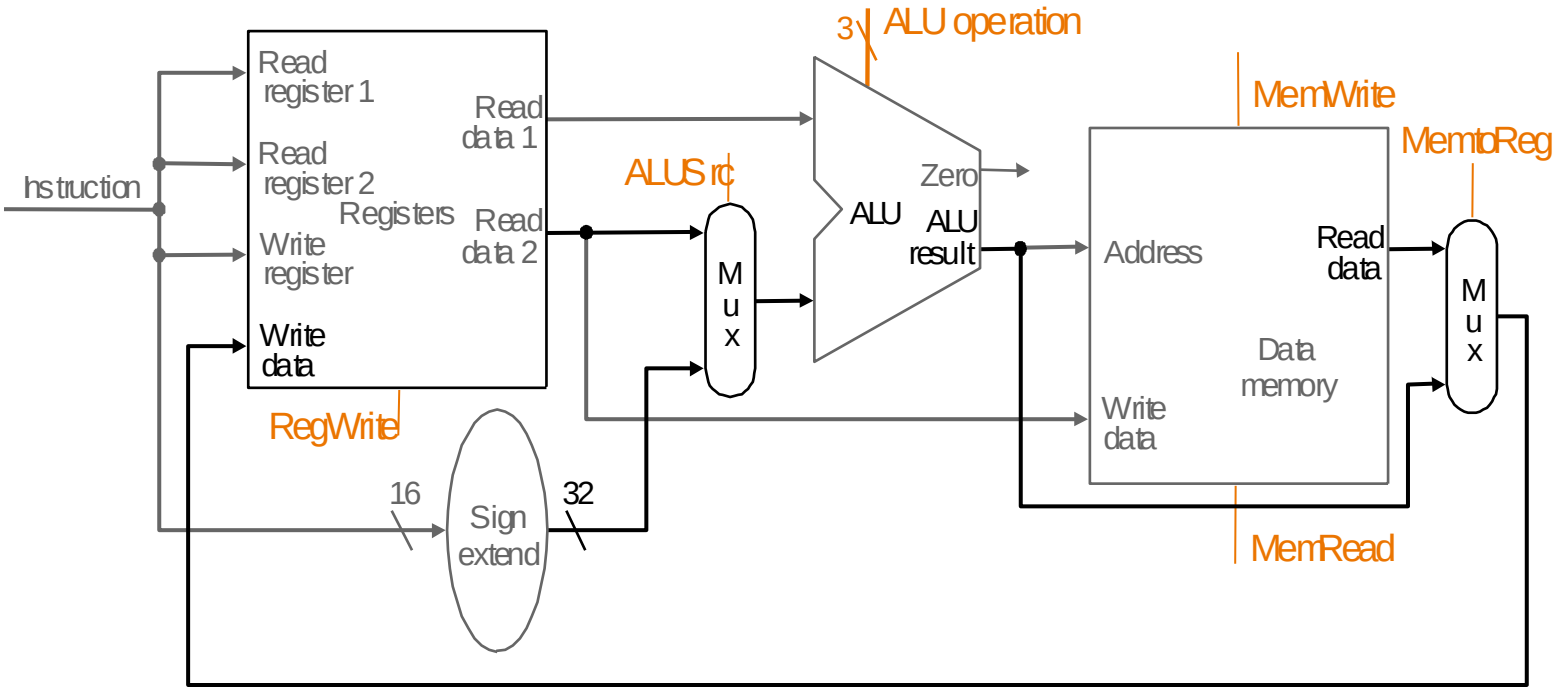
14

Review: Writing to Register File

- Need to submit:
  - Register #
  - Data
  - Write control signal
- To choose a register, use
  - Decoder
- To determine when to write, use the clock
- Note:
  - C means control signal
  - D means data lines



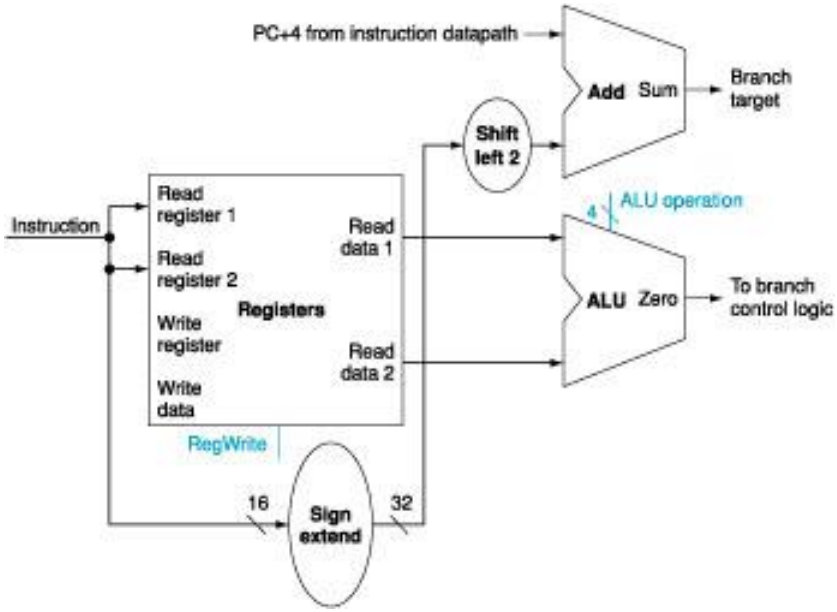
Building the Datapath (lw, sw)



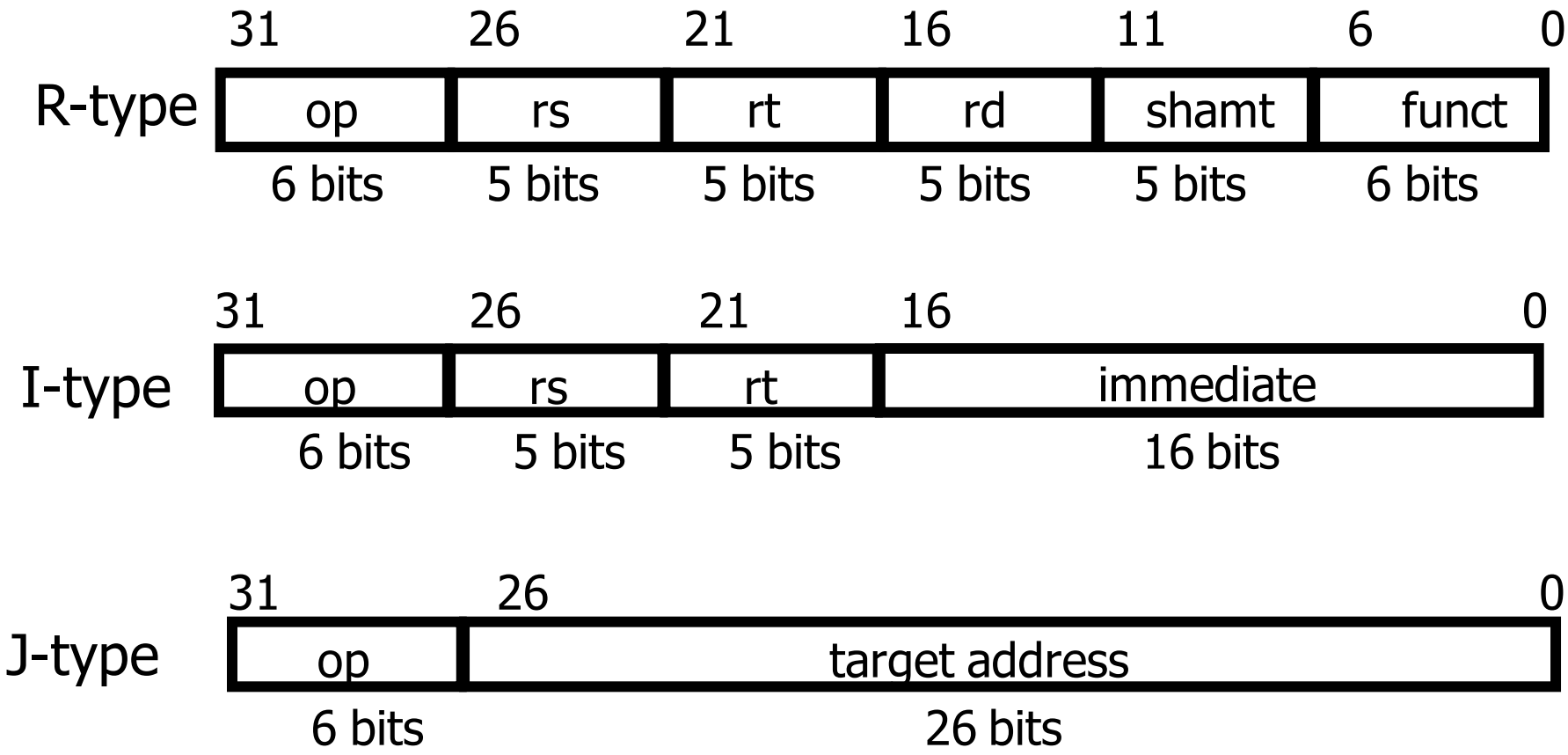
Building the Datapath (beq)

- For branch instructions,

  two operations are needed
  - Compare register contents
  - ALU Zero signal returns the result of comparison
  - Compute branch target address
    - A sign-extend unit is required
    - If branch is taken
      - Branch target address becomes the new PC contents
    - If branch is not taken
      - PC+4 is the new value for PC



Review: MIPS Instruction Format



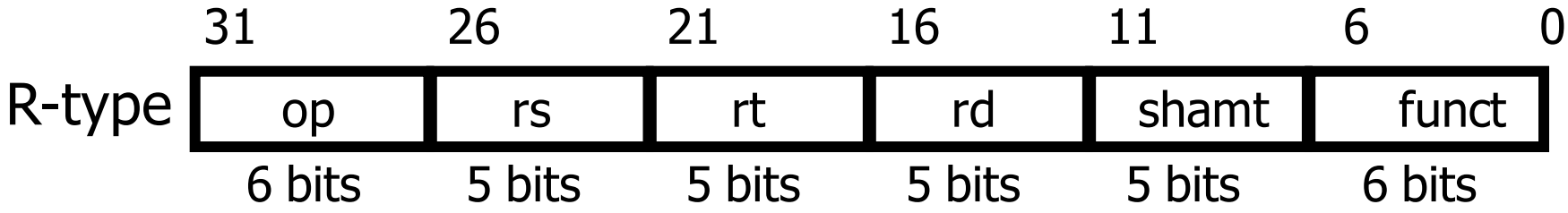
Addressing Modes

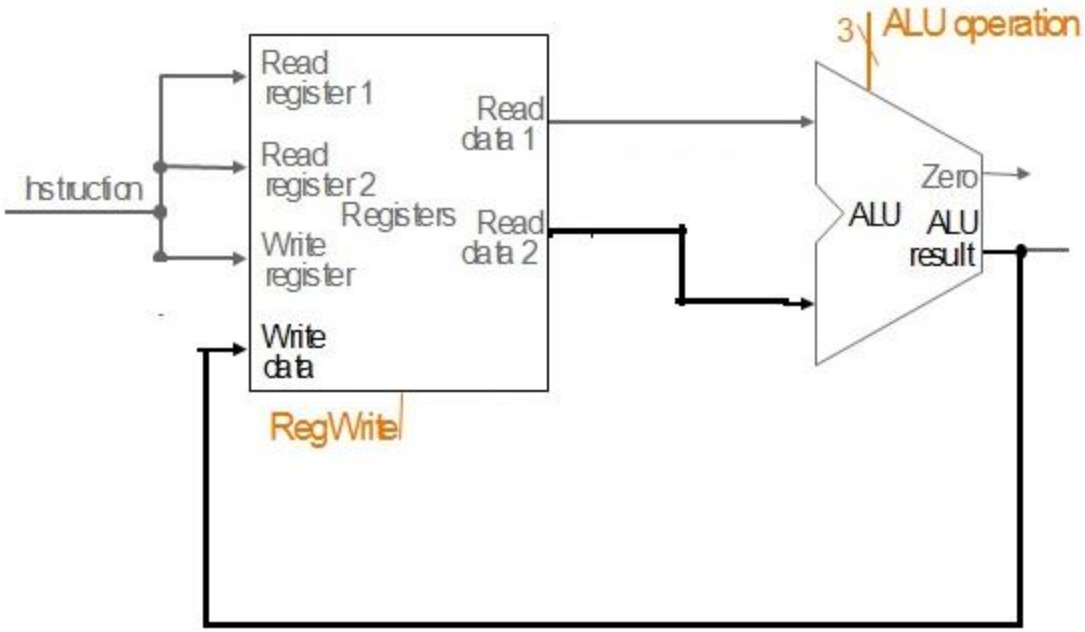
- Register addressing
- Base or displacement addressing
- Immediate addressing
- PC-relative addressing
- Pseudo-direct addressing

Addressing Modes (1)

- Register addressing
  - Operand is a register
  - Value is the contents of the register

Building the Datapath



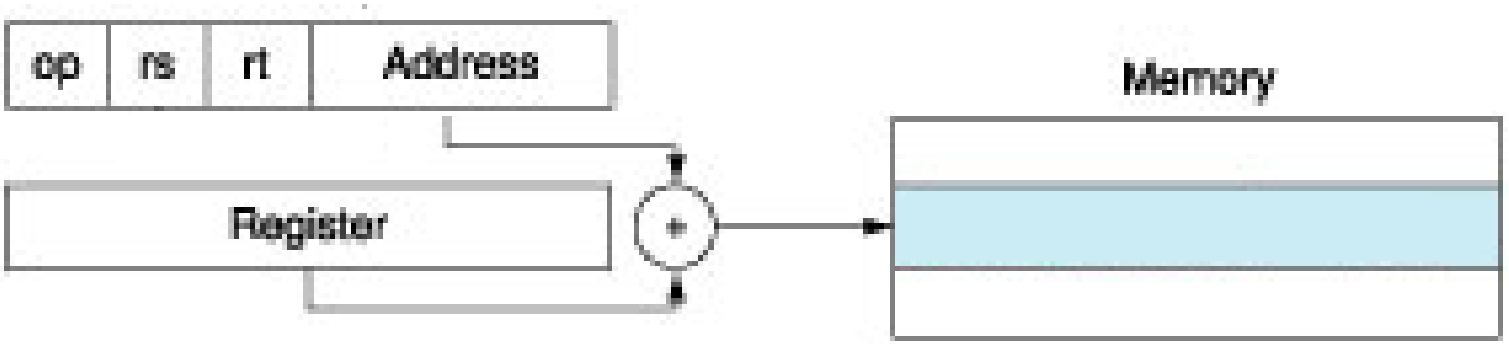


Addressing Modes

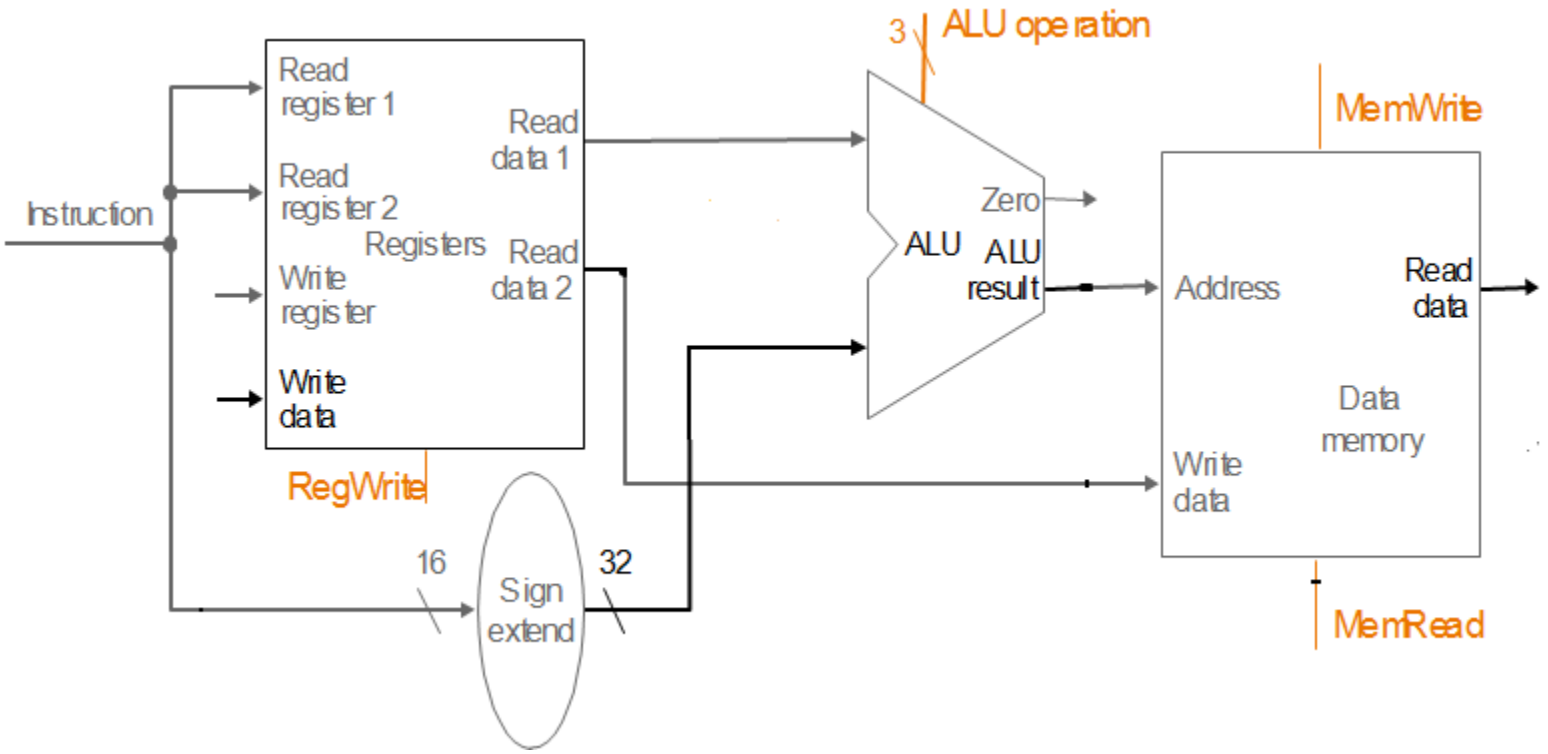
- Register addressing
- Base or displacement addressing
- Immediate addressing
- PC-relative addressing
- Pseudo-direct addressing

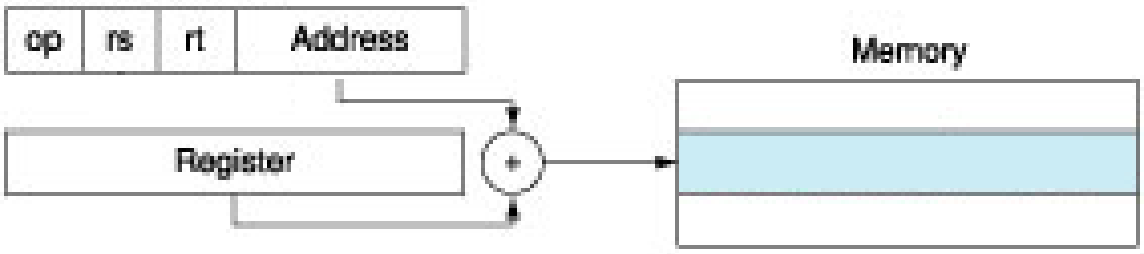
Addressing Modes (2)

- Base or displacement addressing
  - Operand location =
    register + constant (offset) in the instruction

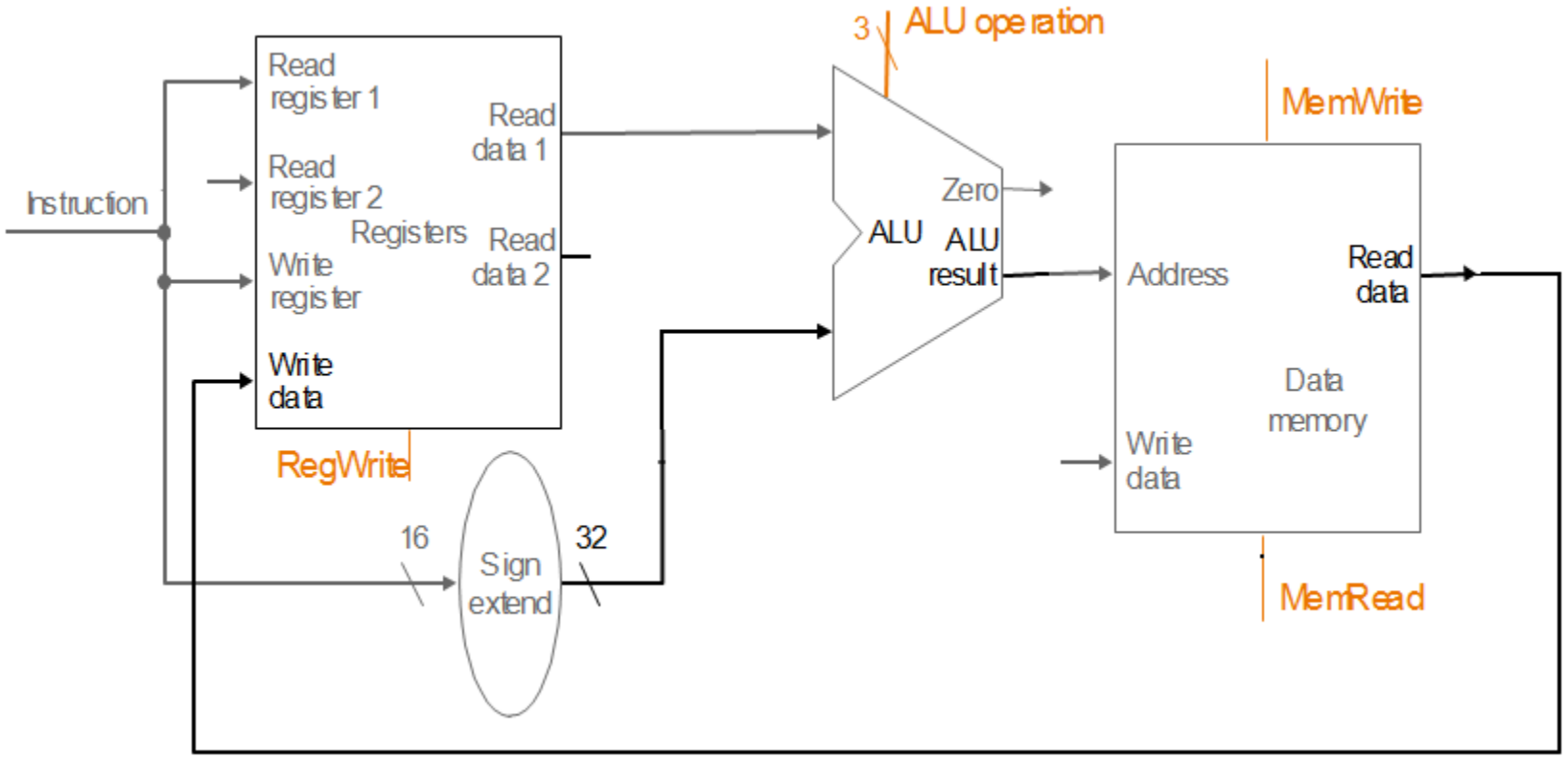


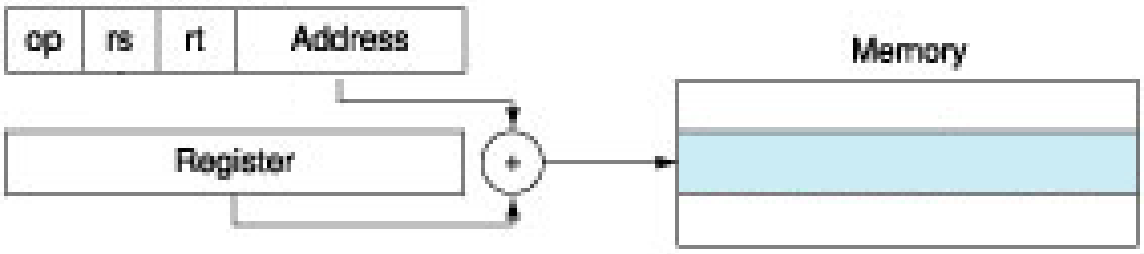
Building the Datapath (sw)



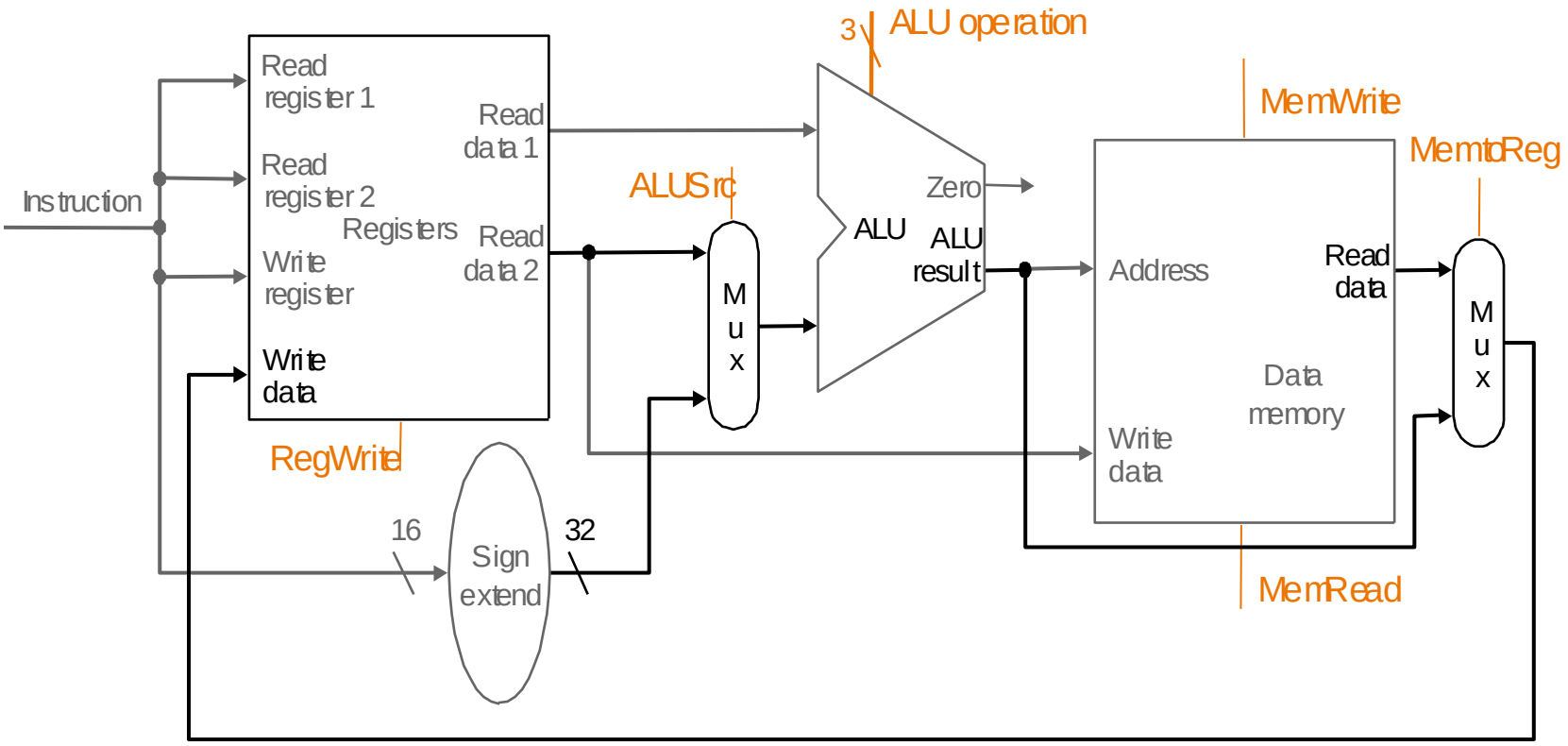


Building the Datapath (lw)

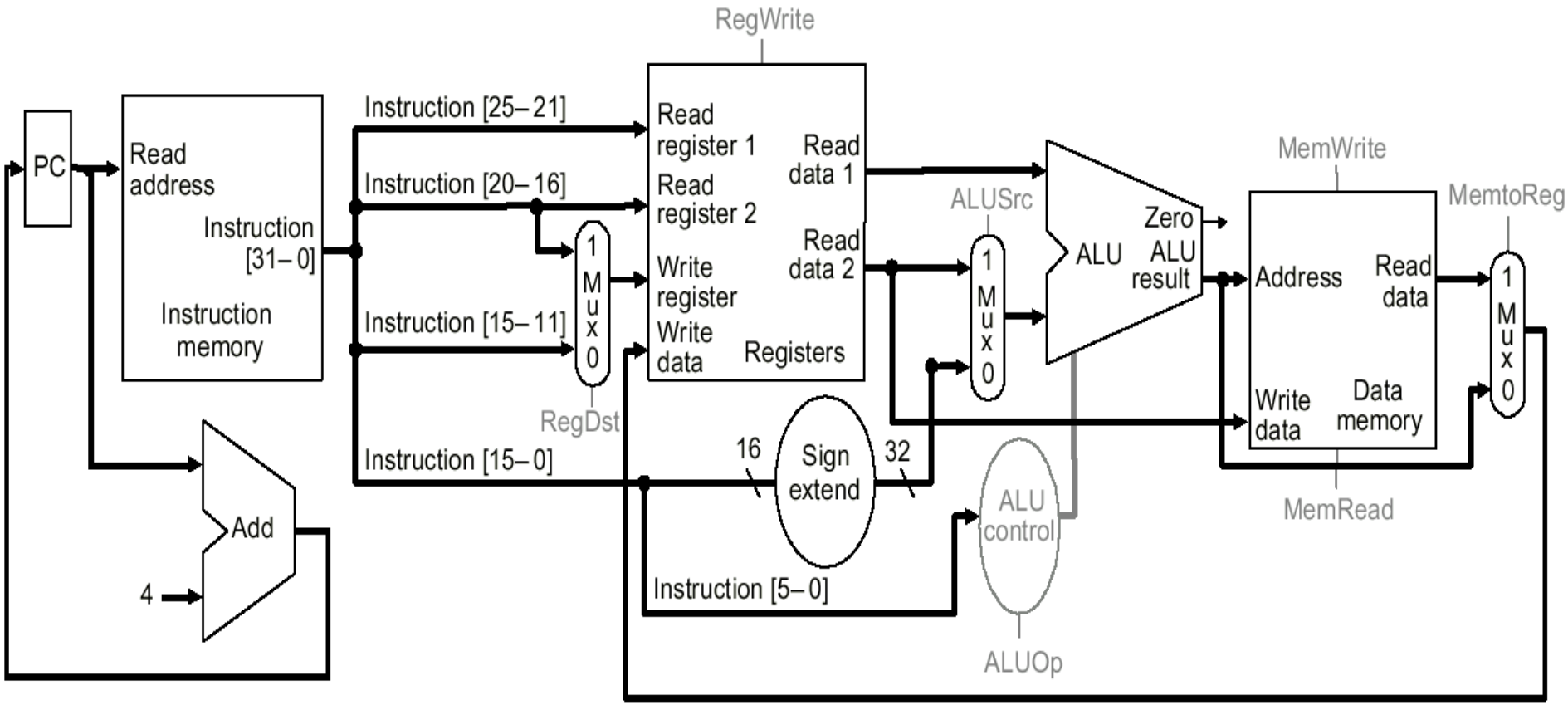




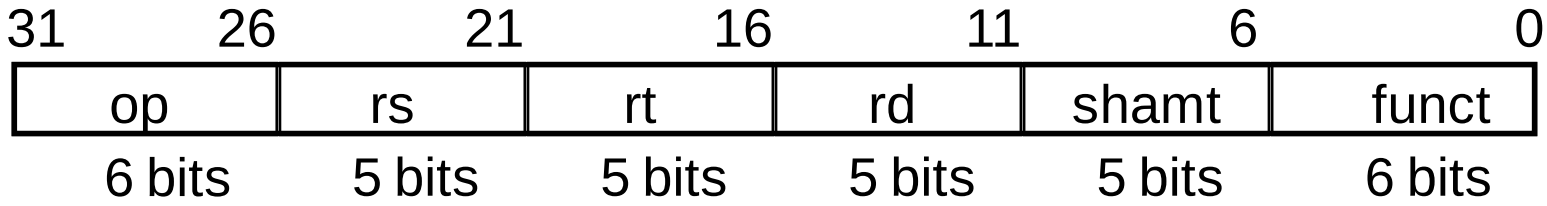
Building Datapath (R-type, lw, sw)



Building Datapath (R-type, lw, sw)



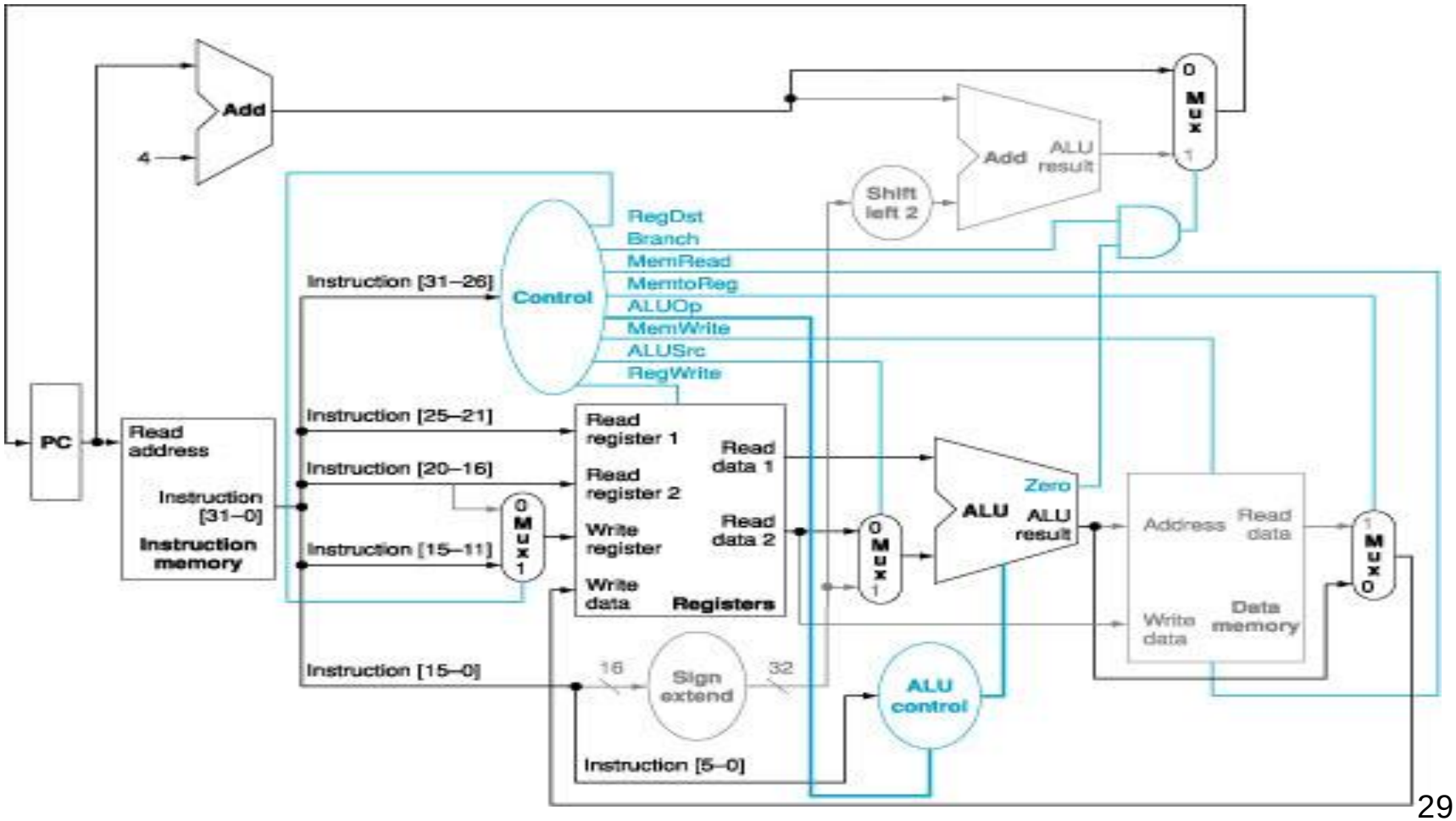
Datapath Operation for R-Format ***(add)***



- Step1: ***Add $t1, $t2, $t3***
  - Fetch instruction from instruction memory
  - Increment program counter
- Step2:
  - Decode result is ***Add*** operation
  - Read two registers ***$t2 & $t3*** from register file
- Step 3:
  - ALU operates on data, using ***"funct"*** field code to generate the ALU function
- Step4:
  - Write result of step 3 from ALU into register ***$t1***

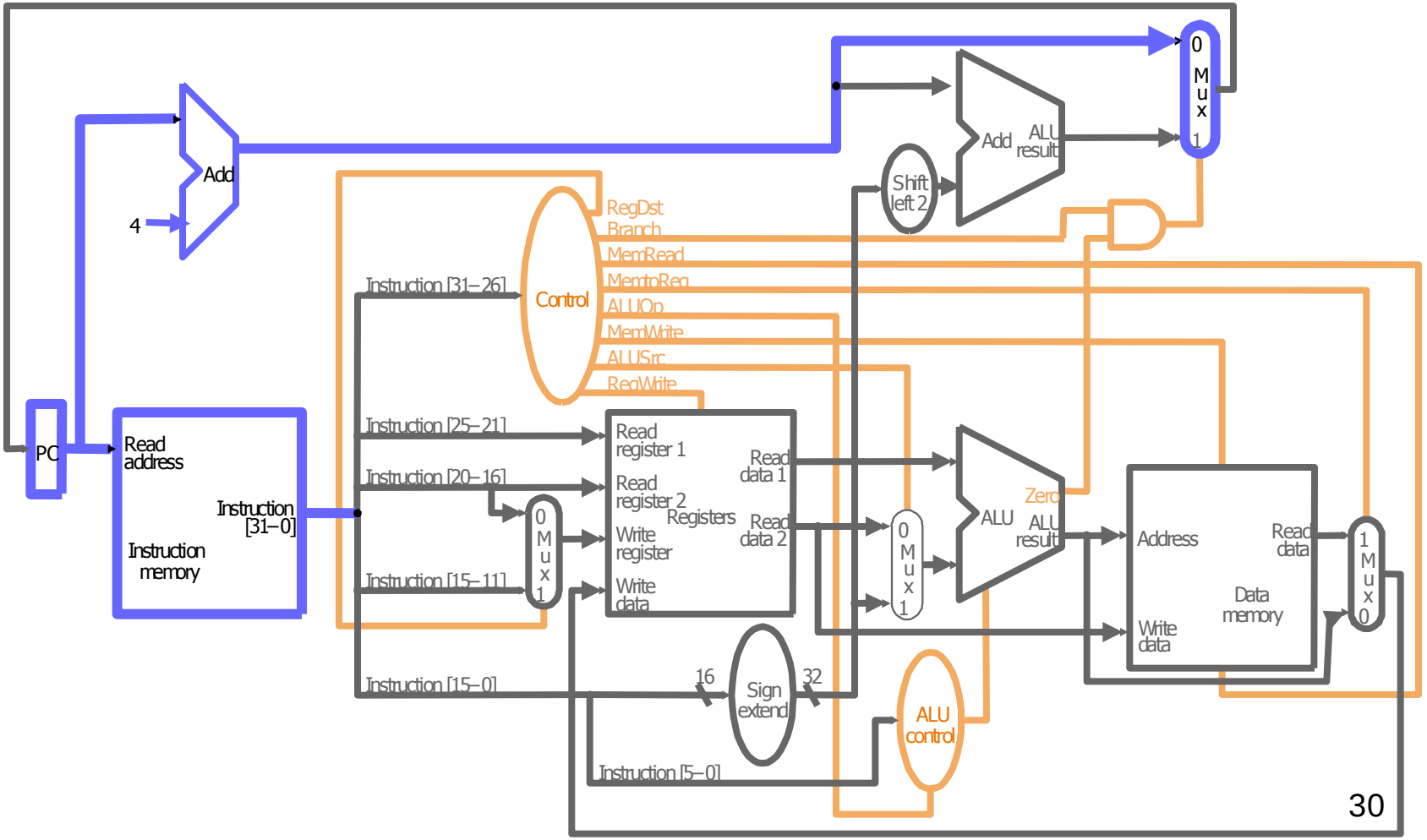
Datapath Operation for R-Format ***(add)***

Add $t1, $t2, $t3 (Fig. 5.19, p. 322)



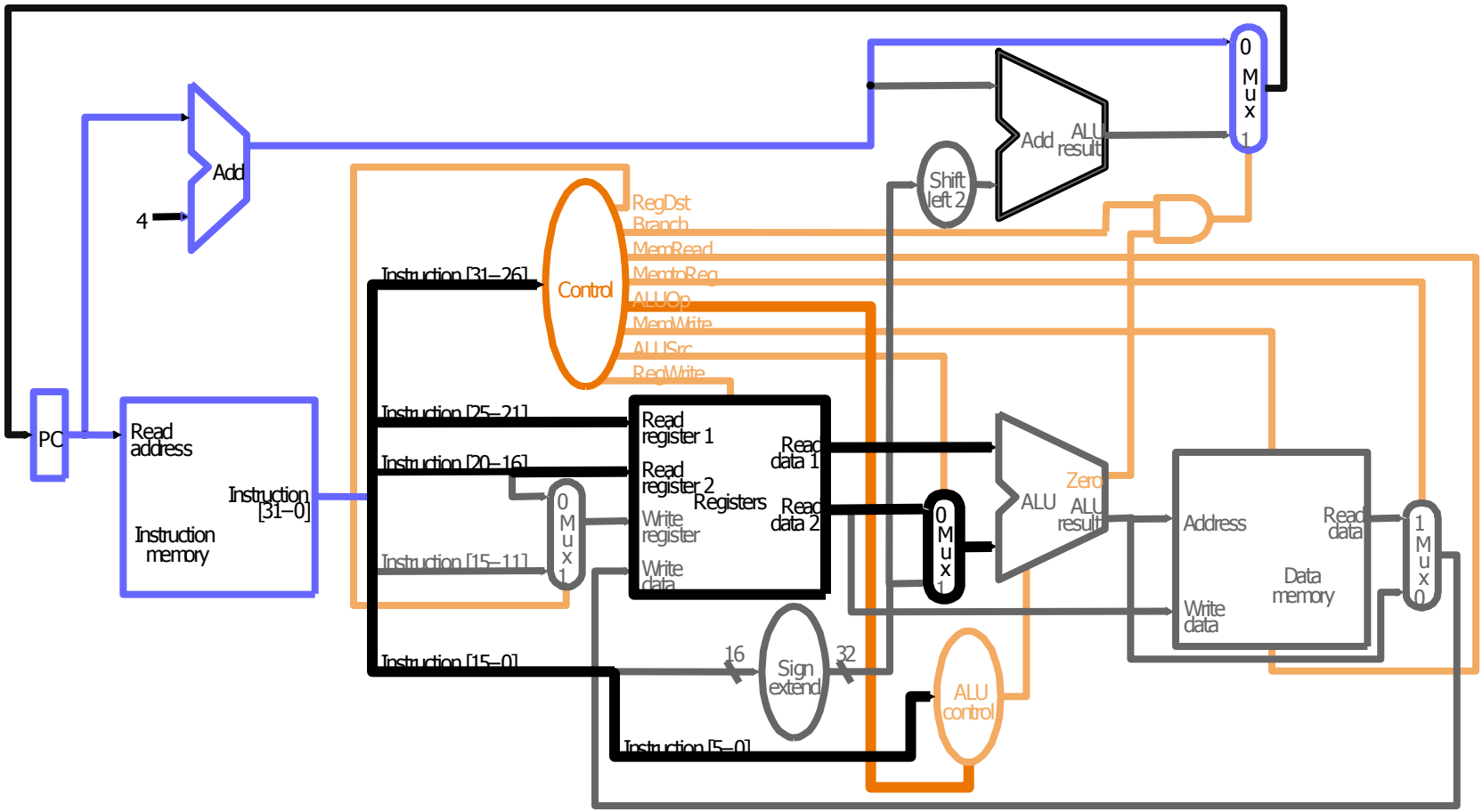
Datapath Operation for R-Format ***(add)***

- Step 1: Fetch instruction & increment PC



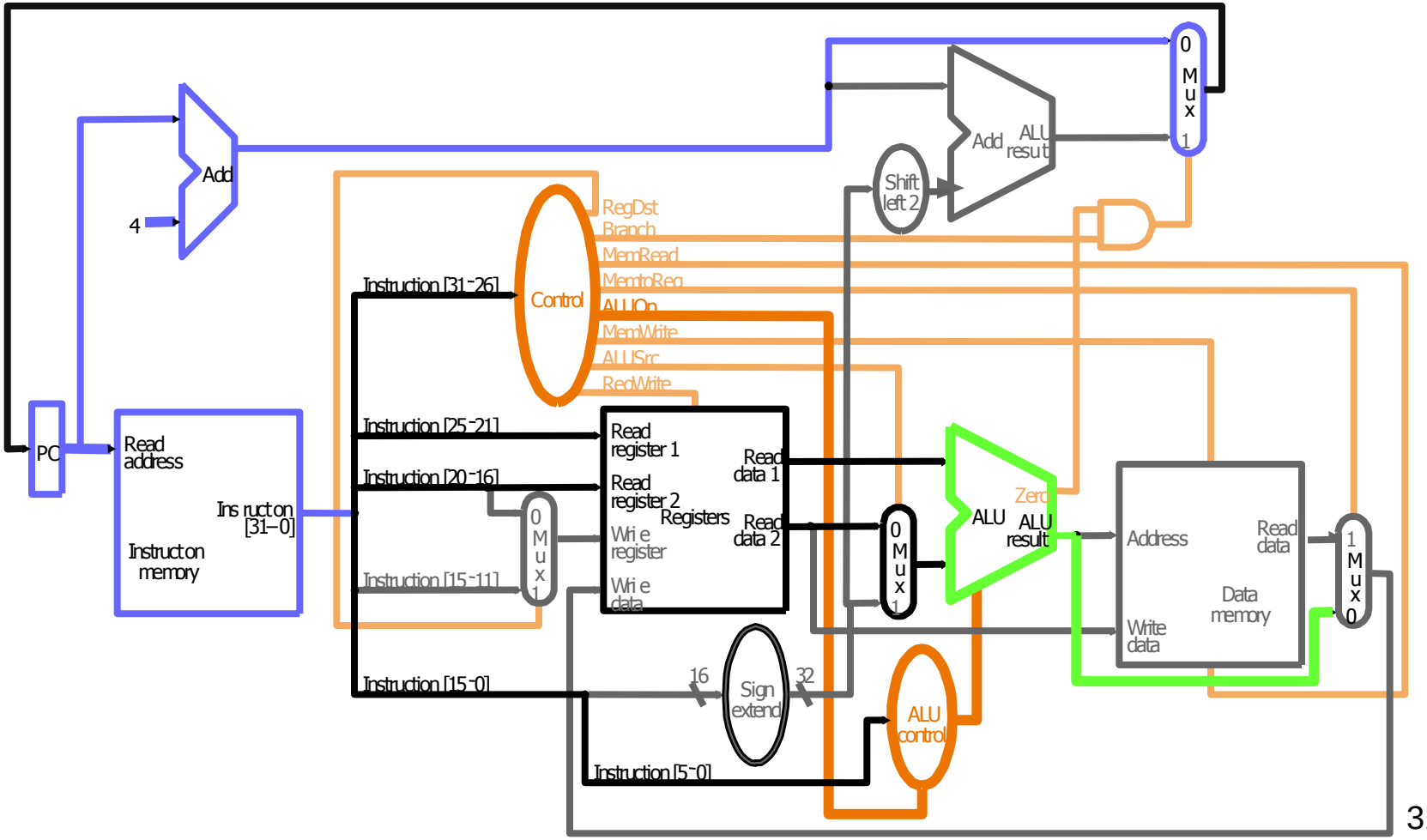
Datapath Operation for R-Format ***(add)***

- Step2: Read registers from register file



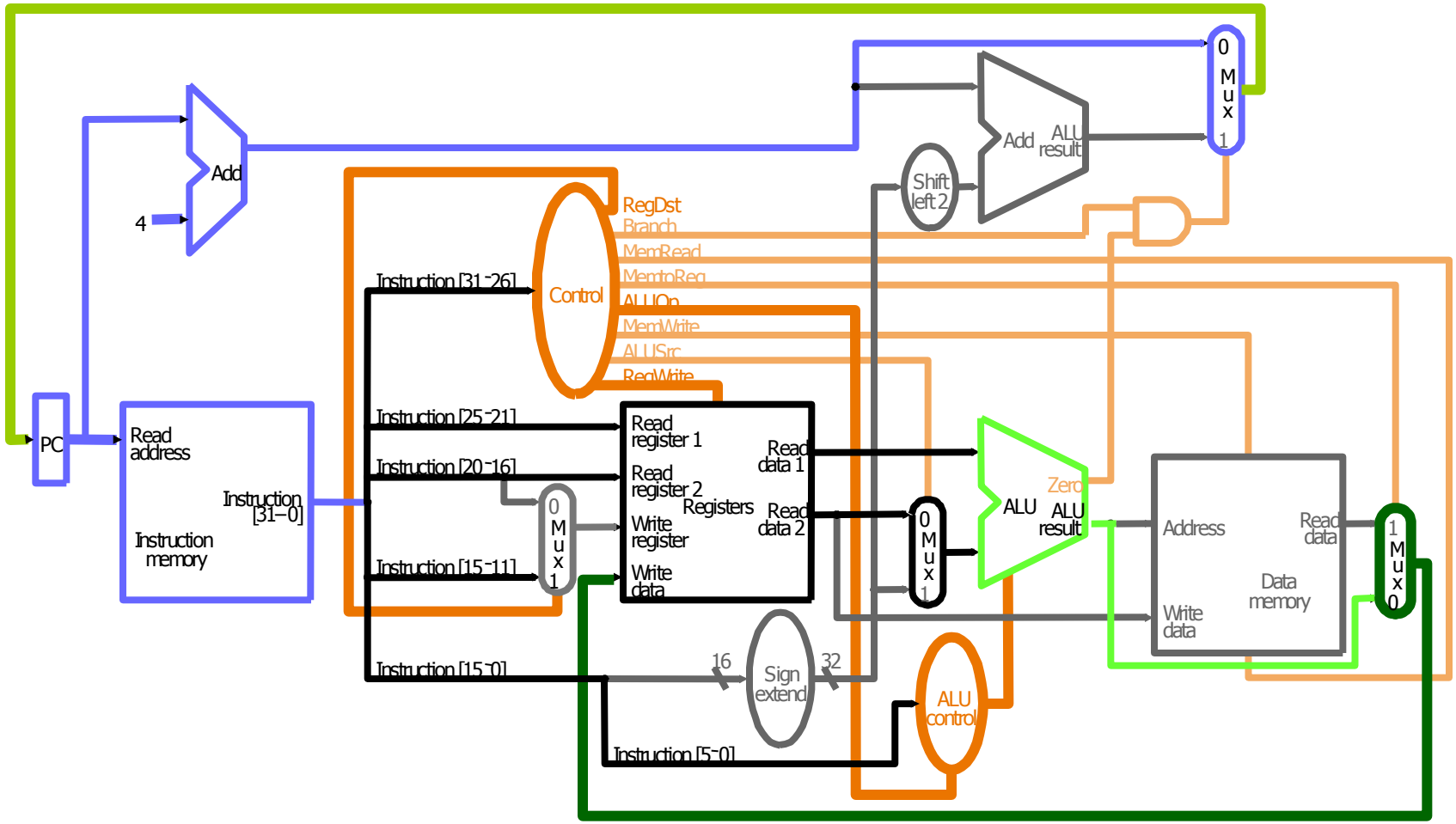
Datapath Operation for R-Format ***(add)***

- Step 3: Perform ALU operation

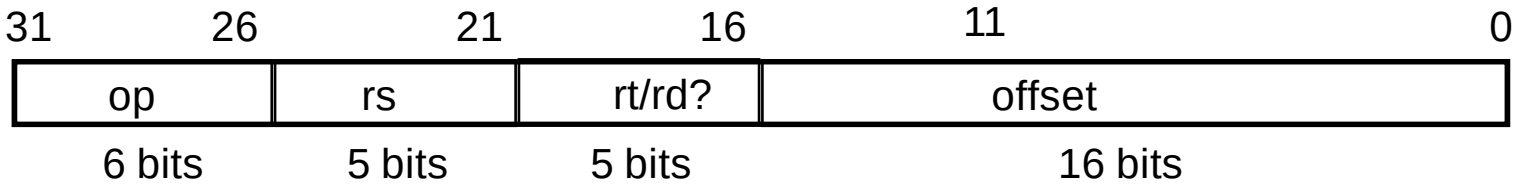


Datapath Operation for R-Format ***(add)***

- Step 4: Write result from ALU into register



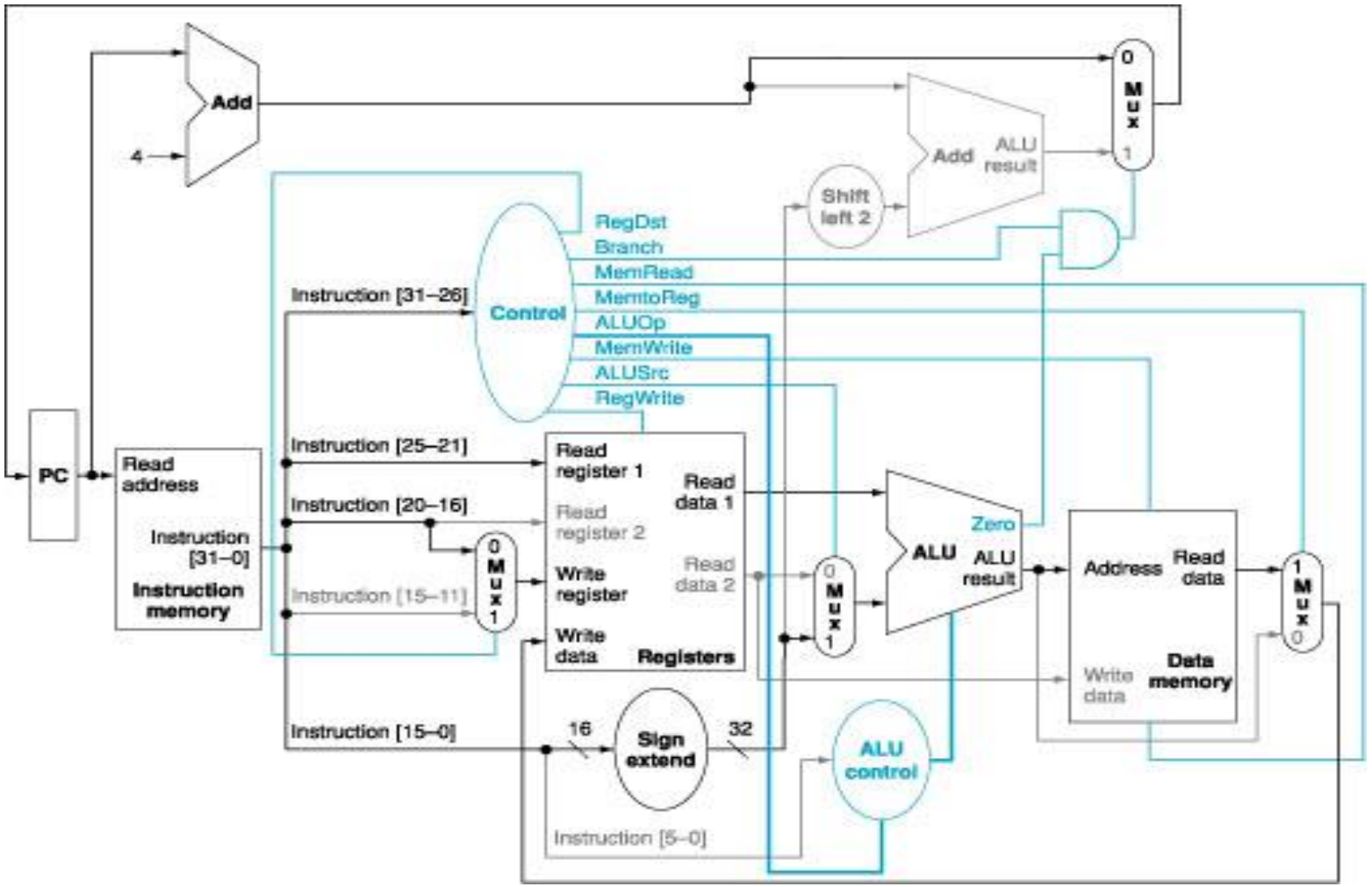
Datapath Operation for I-Format ***(lw)***



- Step 1: ***lw $t1, offset($t2)***
  - Fetch instruction from instruction memory
  - Increment program counter
- Step 2:
  - Decode result is ***lw*** operation
  - Read register ***$t2*** from register file
- Step 3:
  - ALU operates on data to build an address
  - Compute sum of ***$t2*** & sign-extended 16-bits (offset) of instruction
- Step 4: Retrieve data located in the calculate address from memory
- Step5: Write memory contents into destination register $t1

Datapath Operation for I-Format ***(lw)***

lw $t1, offset($t2) - Exercise: Highlight the steps!



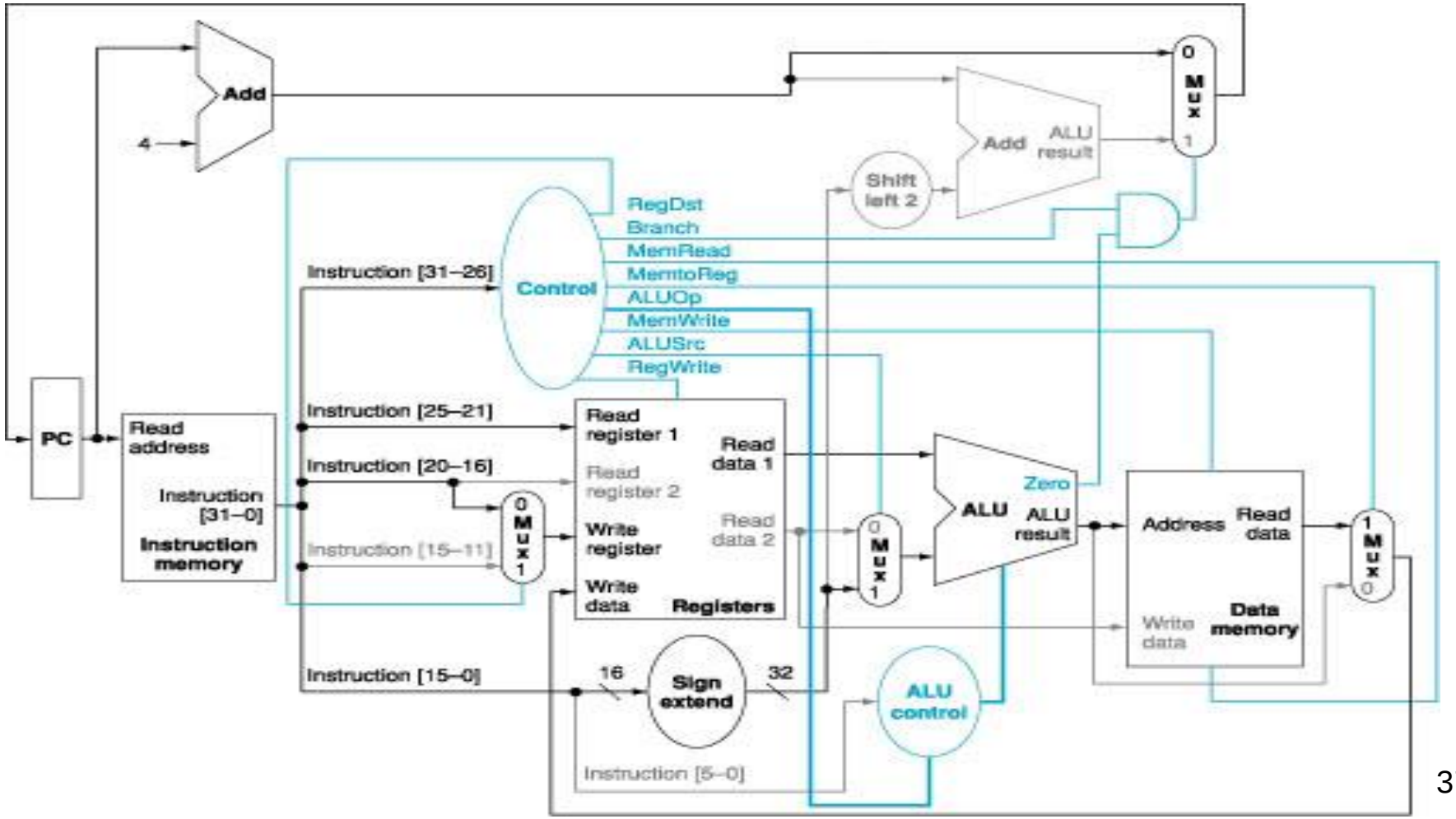
Datapath Operation for I-Format ***(sw)***

sw $t1, 33($t2)

- Step 1:
  - Fetch instruction from instruction memory
  - Increment program counter
- Step 2:
  - Decode result is ***SW*** operation
  - Read register contents of ***$t1*** and ***$t2*** from register file
- Step 3:
  - ALU operates on data to build an address
  - Compute sum of ***$t2*** & sign-extended 16-bits (offset) of instruction
- Step 4:
  - Write data into the computed memory address

Datapath Operation for I-Format ***(sw)***

sw $t1, offset($t2)

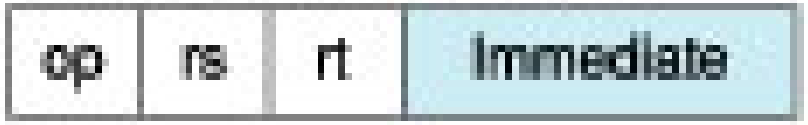


Addressing Modes

- Register addressing
- Base or displacement addressing
- Immediate addressing
- PC-relative addressing
- Pseudo-direct addressing

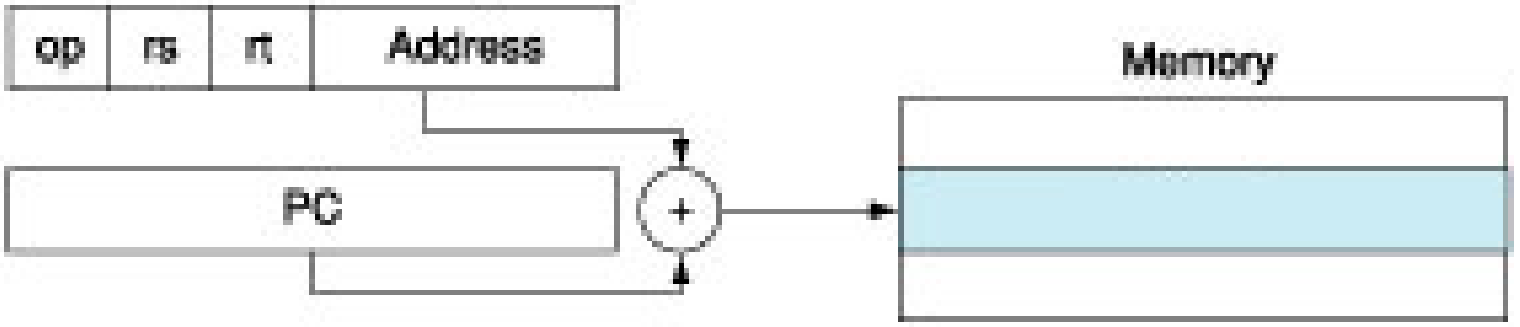
Addressing Modes (3)

- Immediate addressing
  - Operand is a constant within the instruction itself



Addressing Modes (4)

- PC-relative addressing
  - Address = PC (program counter)
    + constant in the instruction

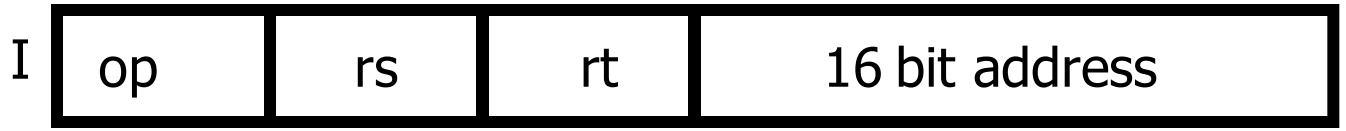


Branch instruction

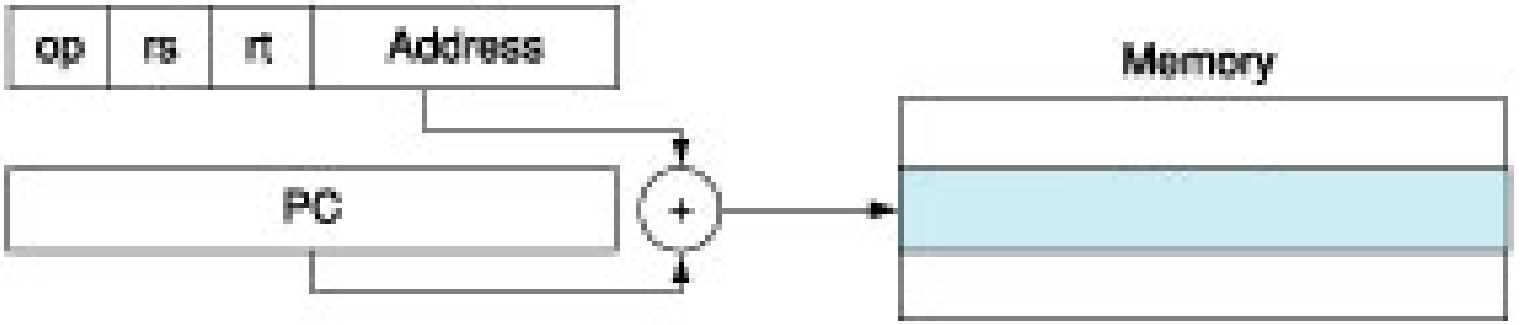
bne $t4,$t5,Label # Next instruction is at Label if $t4 ≠ $t5

beq $t4,$t5,Label # Next instruction is at Label if $t4 = $t5

- Formats:



– Use Instruction Address Register (PC = program counter)

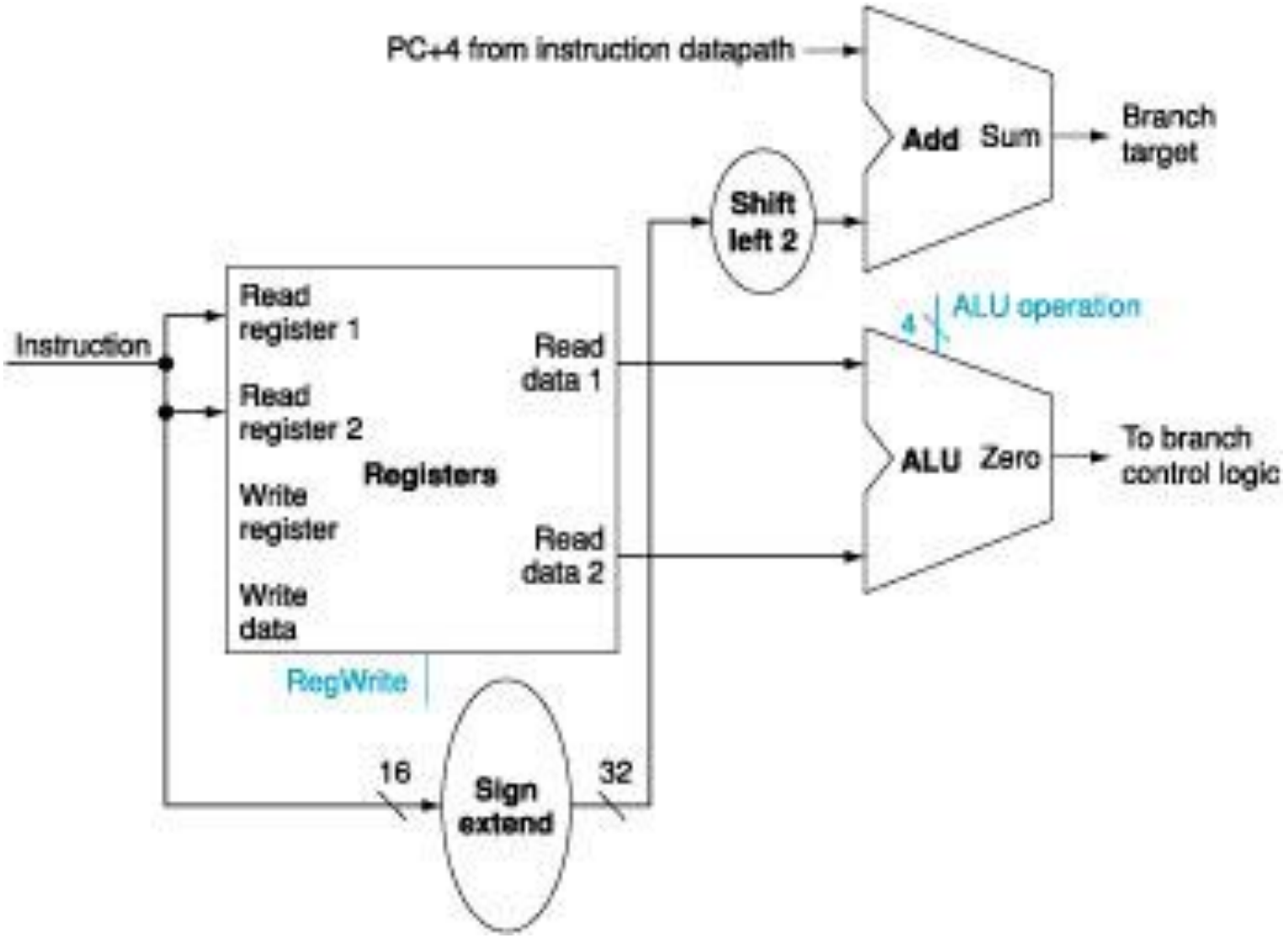


Datapath Operation for I-Format ***(beq)***

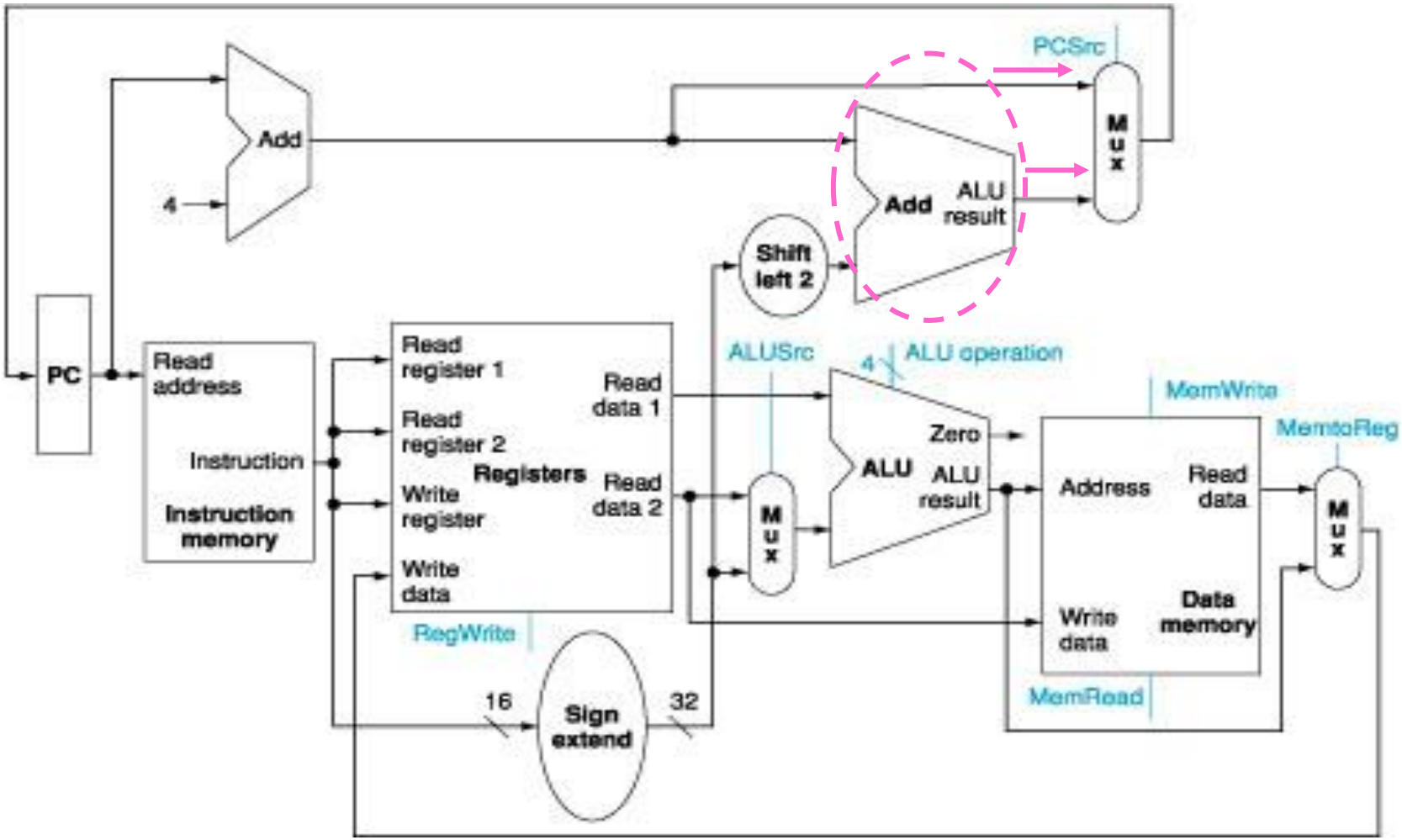
beq $t1, $t2, offset (Fig. 5.21, p. 311)

- Step 1:
  - Fetch instruction from instruction memory
  - Increment PC
- Step 2:
  - Decode instruction result in ***beq***
  - Read contents of ***$t1, $t2*** registers from register file
- Step 3:
  - ALU subtract the two values from registers
  - Add PC+4 to sign-extended lower 16-bits of offset
- Step 4:
  - The zero result from ALU is used to decide which adder result to store into PC

Branch instruction

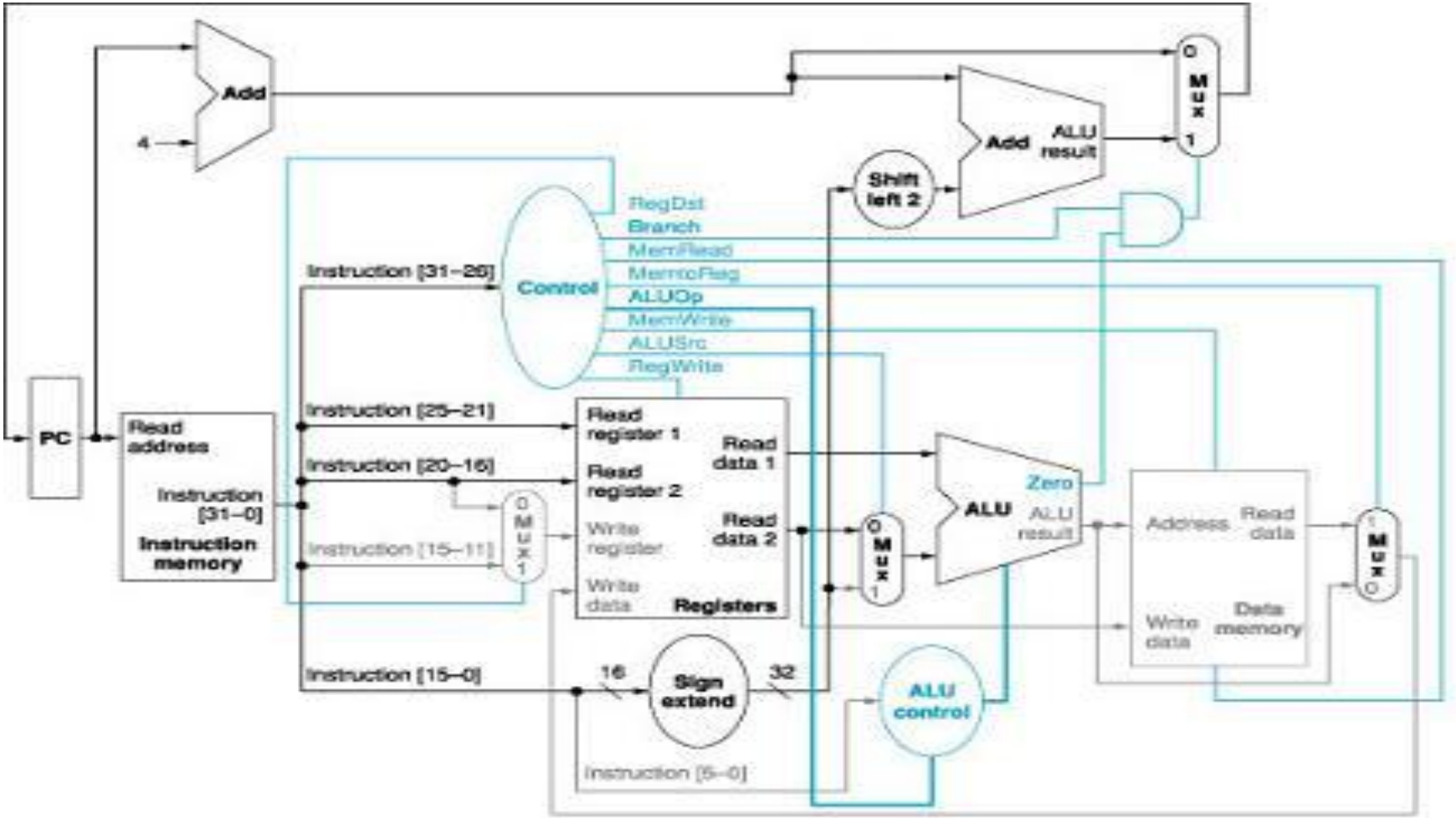


Branch instruction



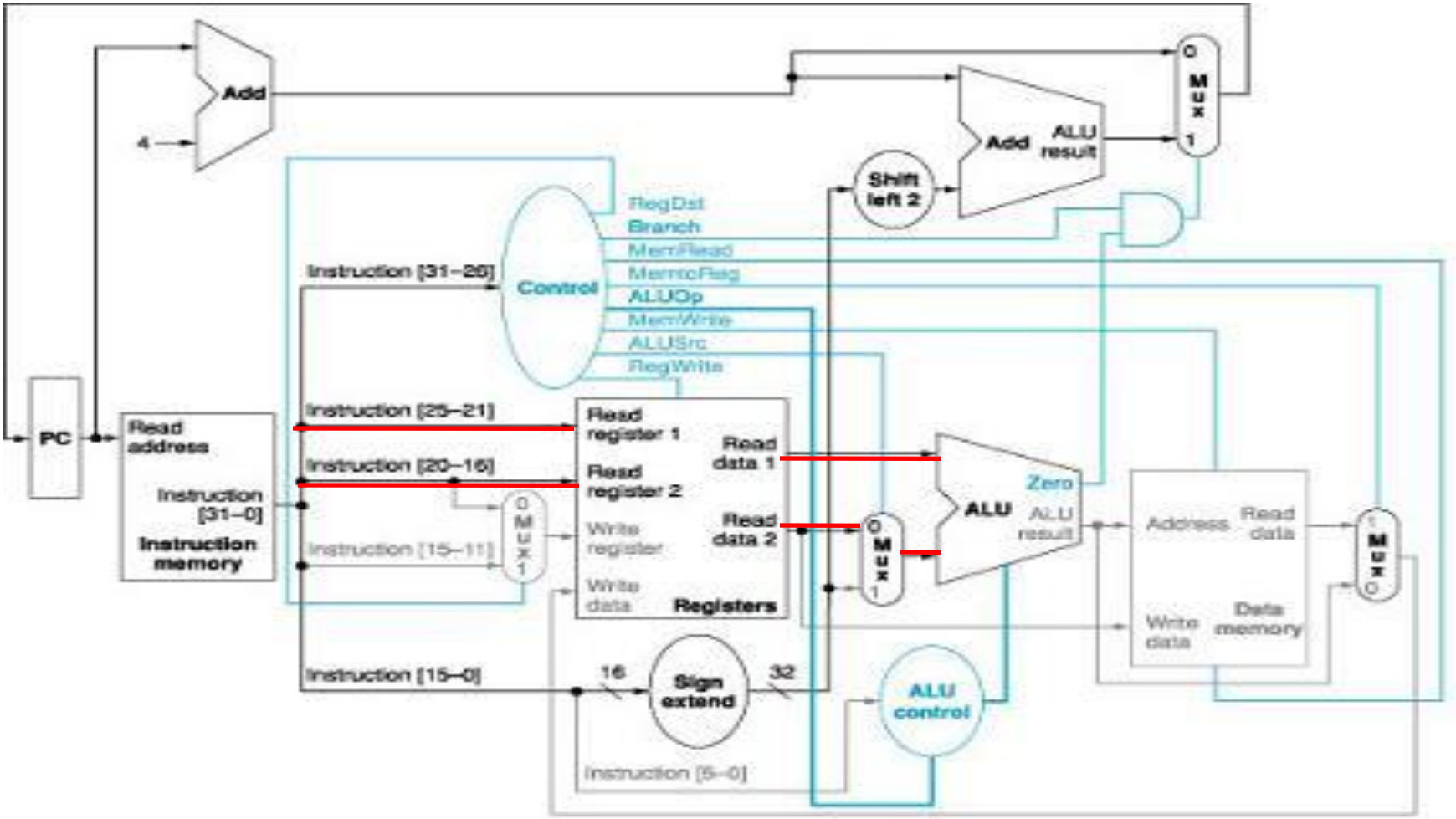
Datapath Operation for ***(beq)***

beq $t1, $t2, offset



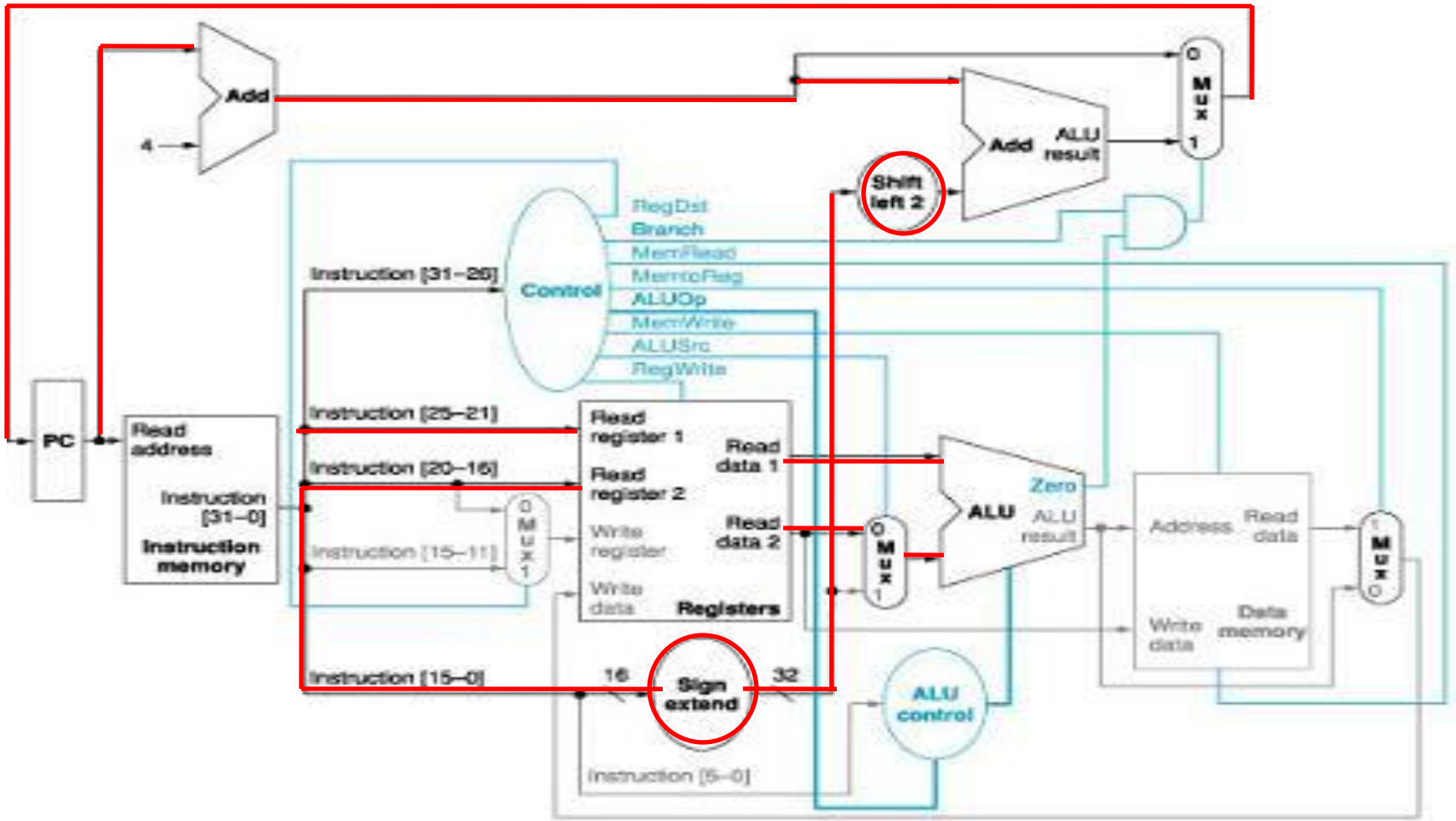
Datapath Operation for ***(beq)***

beq $t1, $t2, offset



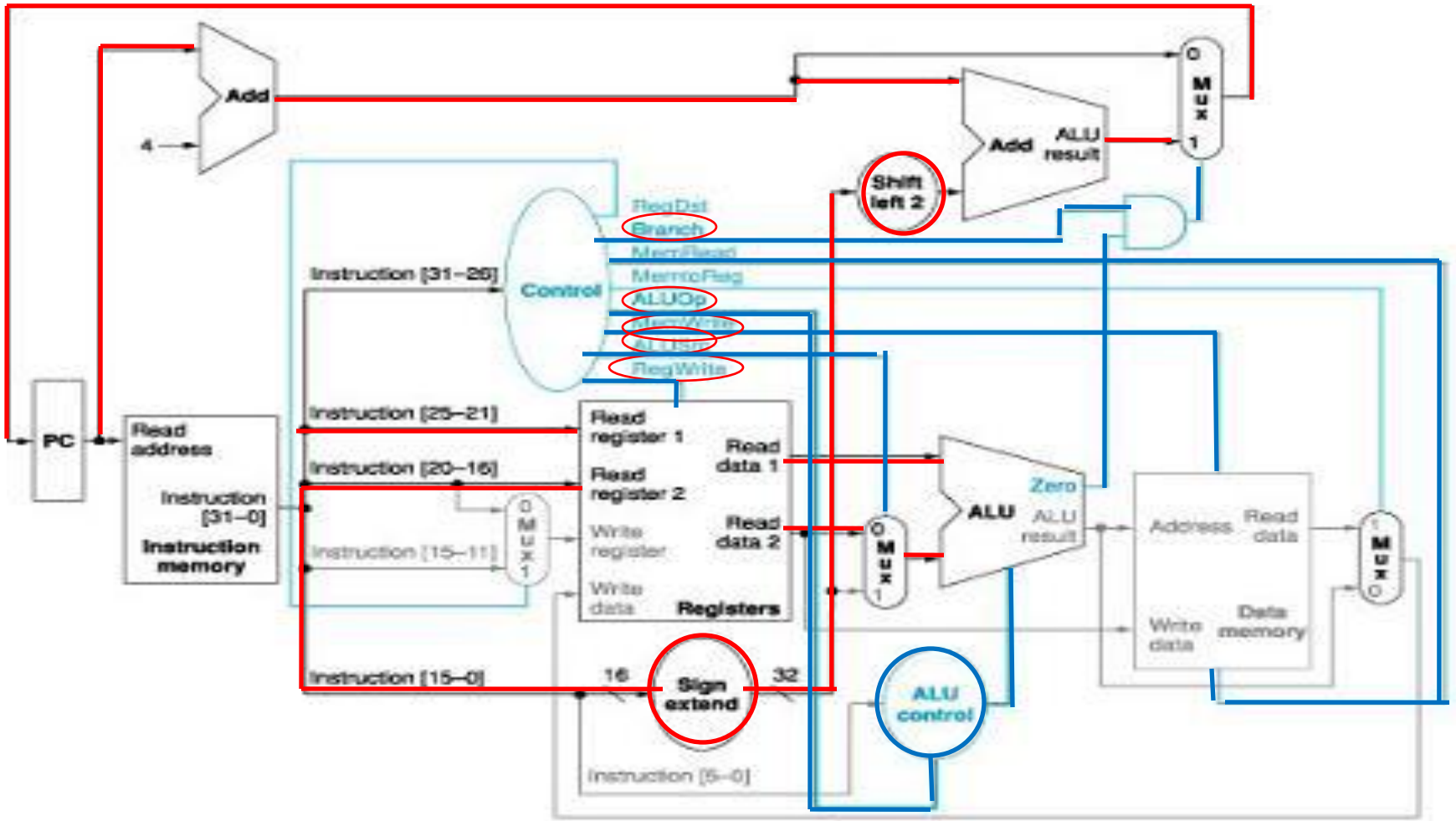
Datapath Operation for ***(beq)***

beq $t1, $t2, offset



Datapath Operation for *(beq)*

beq $t1, $t2, offset

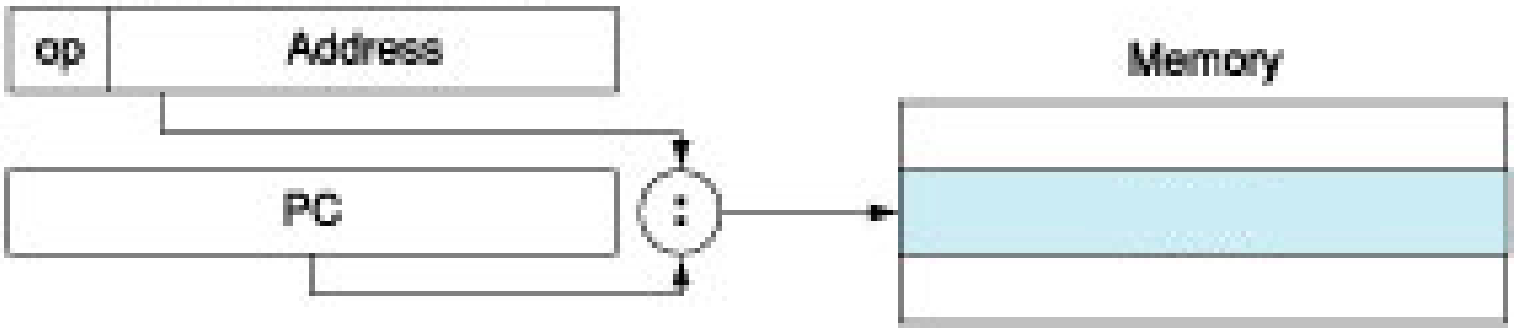


Addressing Modes

- Register addressing
- Base or displacement addressing
- Immediate addressing
- PC-relative addressing
- Pseudo-direct addressing

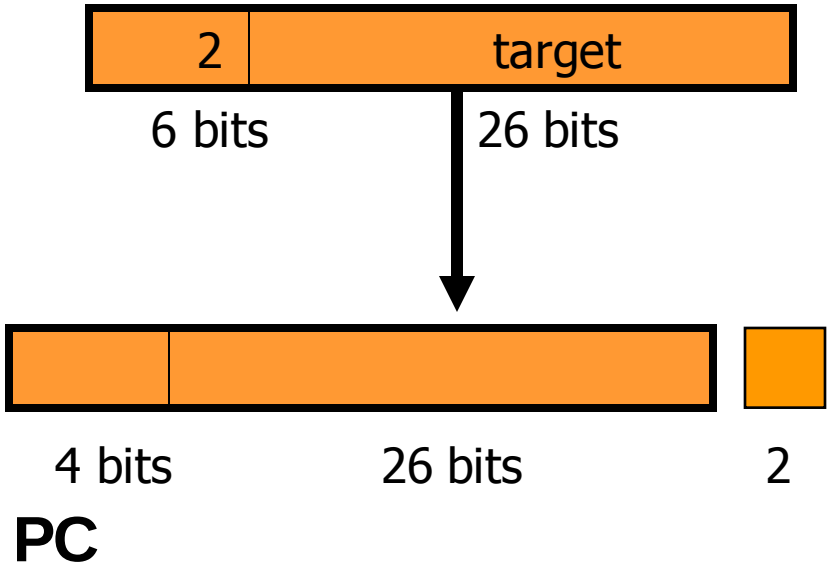
Addressing Modes (5)

- Pseudo-direct addressing
  - Jump address = 26 bits of the instruction
    + upper bits of the PC

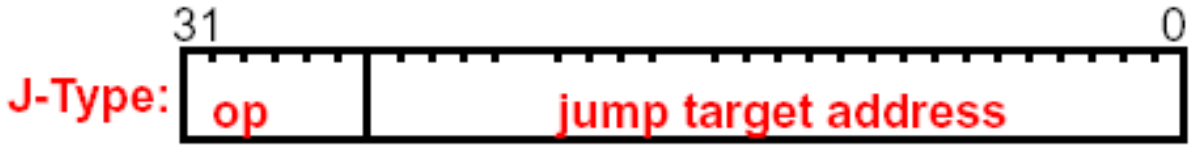


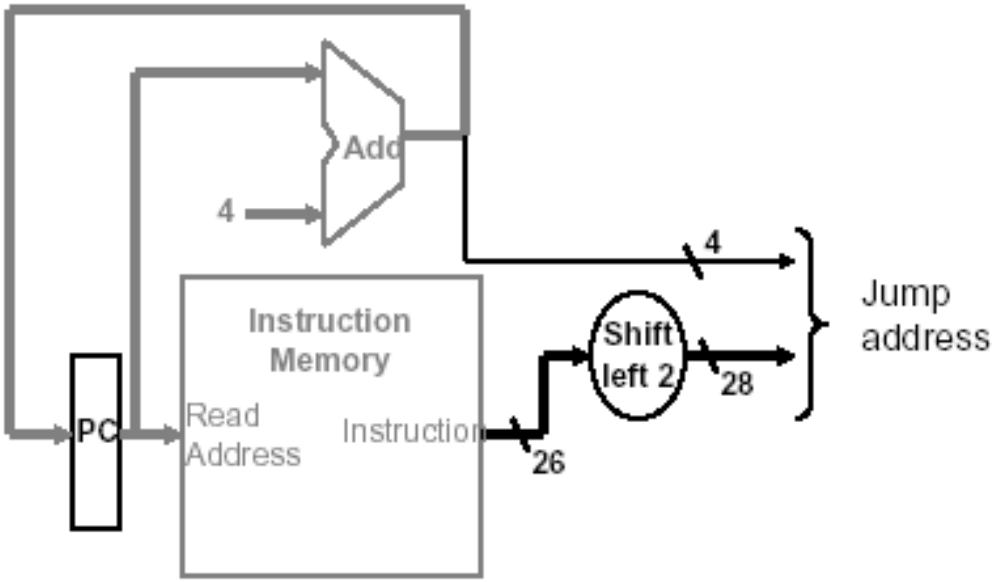
Jump Instruction (J-Format)

- Jump instruction uses high order bits of PC
  - address boundaries of 256 MB



Jump Instruction (J-Format)



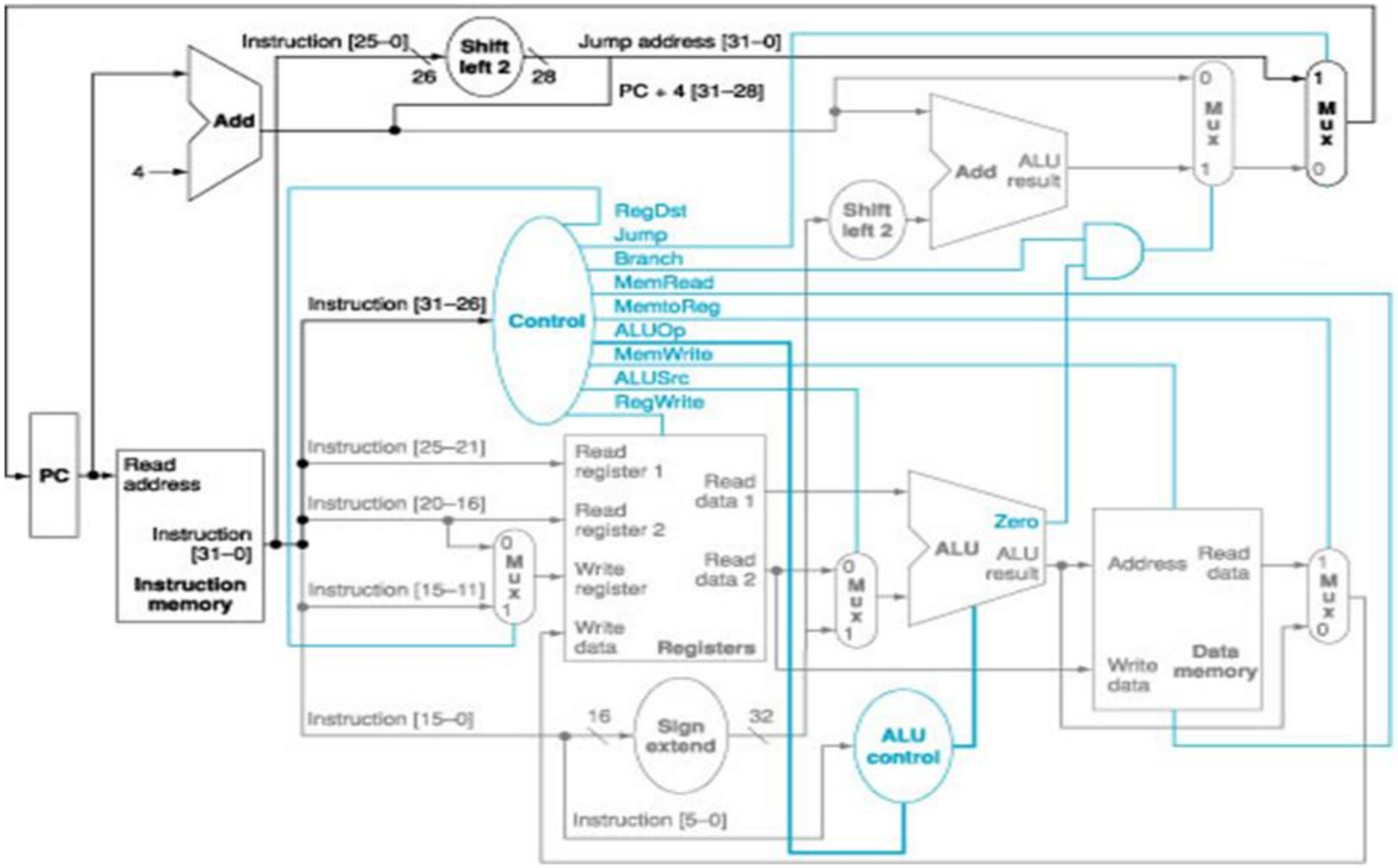


Datapath Operation for J-Format ***(j)***

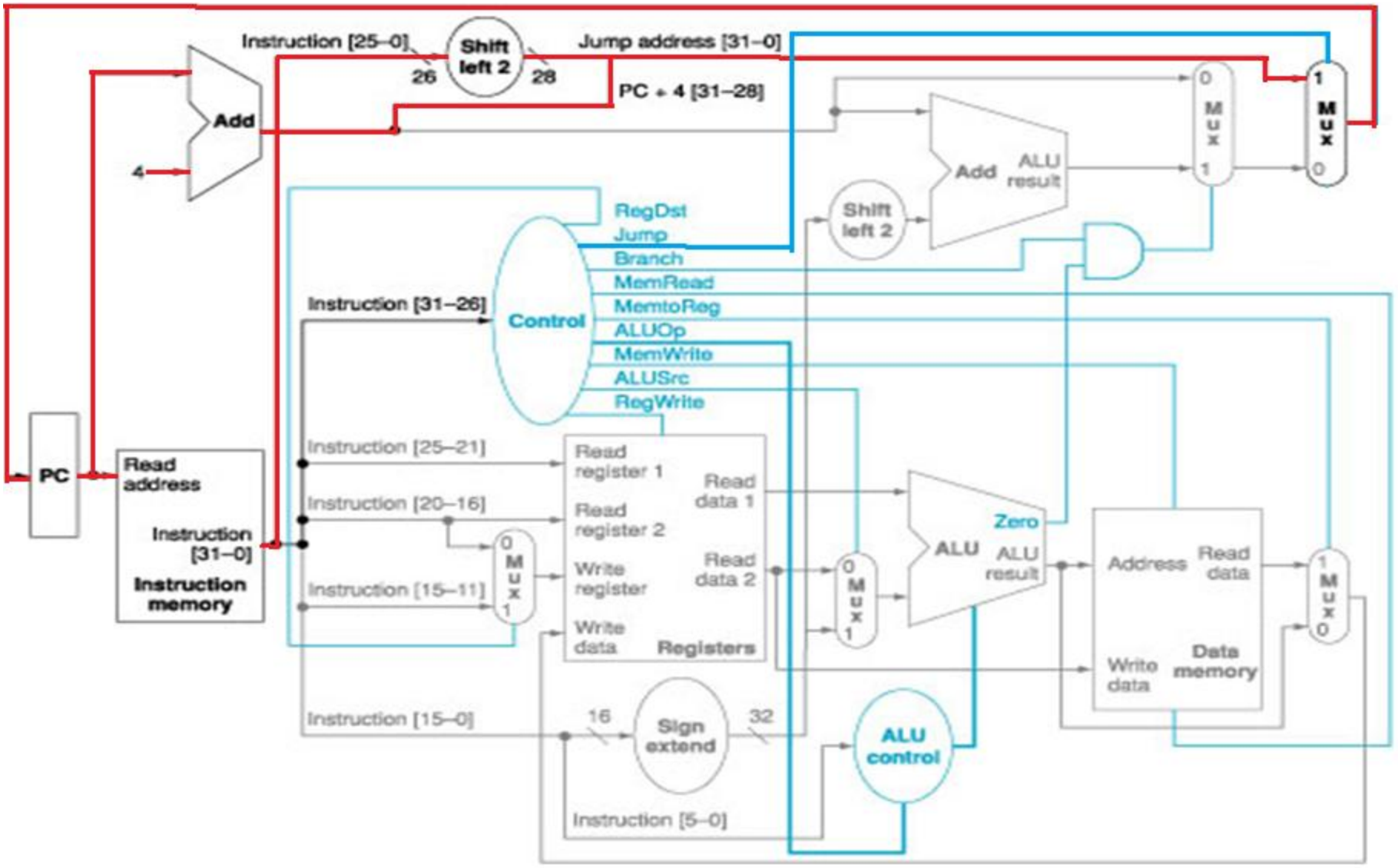
j label

- Step 1:
  - Fetch instruction from instruction memory
  - Increment PC by 4
- Step 2:
  - Decode instruction result is ***j***
  - Retrieve value of jump target (label)
- Step 3:
  - Shift the label 26 bits left by 2 bits (to get byte count instead of word count)
  - Concatenate the upper 4 bits of PC +4 as the high-order bits (to get the complete memory address)
- Step 4:
  - Store the calculated address into PC

Datapath Operation for J-Format



Datapath Operation for J-Format

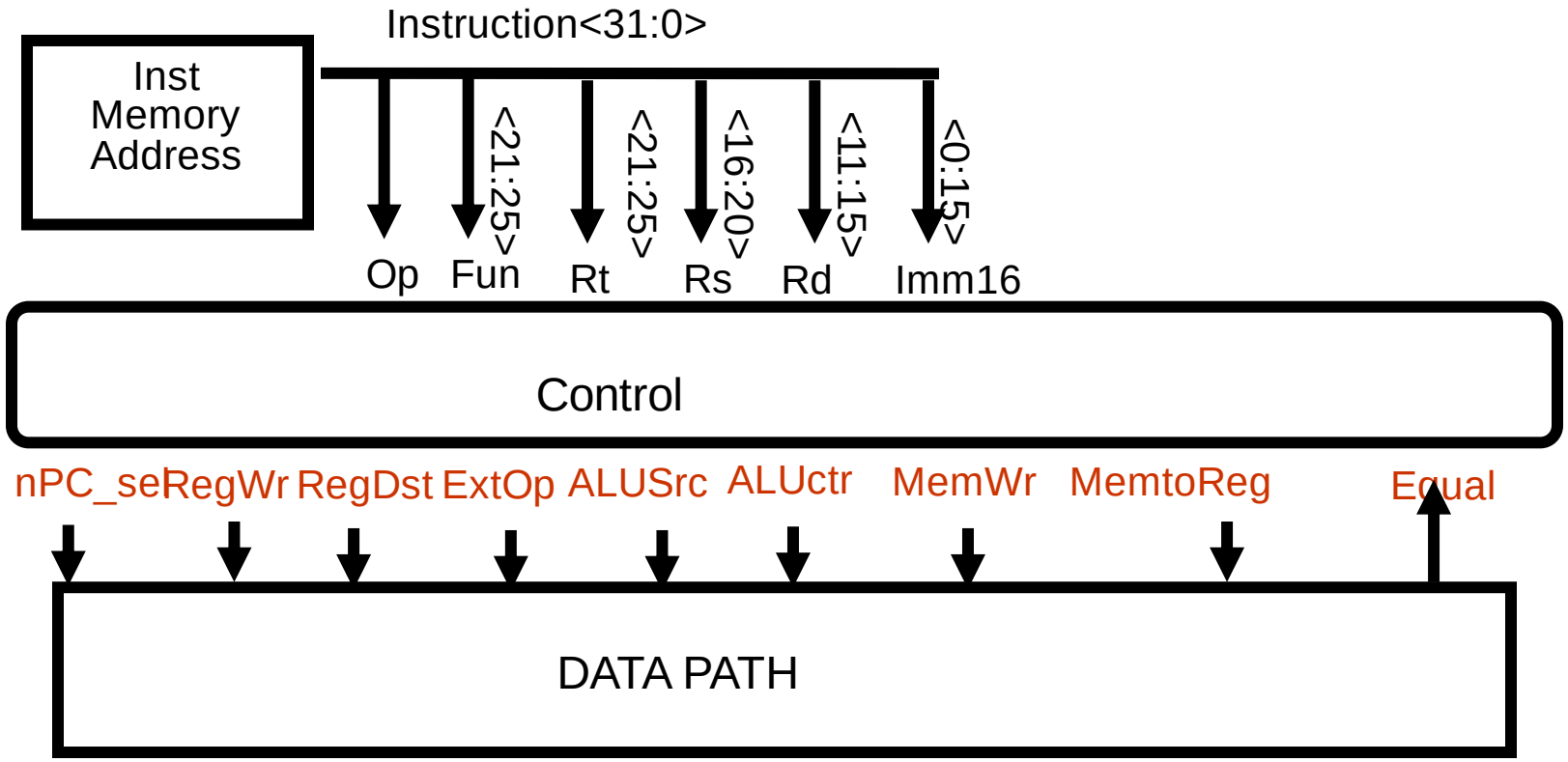


Processor Design

- Control Signals
  - Register selection: Rs, Rt, Rd and Imed16 hardwired into datapath
  - Operation selection:
  - ALU data source (register or immediate address) (ALUsrc)
  - ALU Operation selection (ALUctr)
  - Memory Write (MemWr)
  - MemtoReg (1 => Mem)
  - RegDst (0 => ‟rt”; 1 => ‟rd”)
  - RegWr (write dest. register)

Processor Design Step 4

- Control Unit
  - Analyze implementation of each instruction to determine setting of control points



Processor Design

Control Signals needed by each instruction:

inst	*Register Transfer*	
ADD	*R[rd] <– R[rs] + R[rt];*	*PC <– PC + 4*
	ALUsrc=R[rs], ALUctr="add", RegDst=rd, RegWr, nPC_sel="+4"	
SUB	*R[rd] <– R[rs] – R[rt];*	*PC <– PC + 4*
	ALUsrc=R[rs], ALUctr="sub", RegDst=rd, RegWr, nPC_sel="+4"	
LOAD	*R[rt] <– MEM[R[rs] + sign_ext(Imm16)];*	*PC <– PC + 4*
	ALUsrc = Im, ALUctr = "add",	
	MemtoReg, RegDst = rt, RegWr,	*nPC_sel = "+4"*
STORE	*MEM[R[rs] + sign_ext(Imm16)] <– R[rs];*	*PC <– PC + 4*
	ALUsrc=Im, ALUctr="add", MemWr, nPC_sel="+4"	
BEQ	*if(R[rs]== R[rt]) then PC<– PC+sign_ext(Imm16)] || 00*	*else PC <– PC + 4*
	nPC_sel = EQUAL, ALUctr = "sub"	

Main Control Unit

- Effect of the control signals when they are asserted or de-asserted respectively
- See fig. 4.16, p. 321

Signal name	If de-asserted (0)	If asserted (1)
RegDst	Destination register <= rt field (20-16)	Destination register <= rd field (15-11)
RegWrite	None	Write data input => Write register
ALUSrc	Reg Data2 => 2nd ALU operand	Sign-extnd 16-bits of instruction => 2nd ALU operand
PCSrc	PC <= PC + 4 (from adder)	PC <= Branch target (from adder)
MemRead	None	Mem[Address] => to Read data output
MemWrite	None	Mem[Address] <= value on Write data input
MemtoReg	Value to Write data input comes from ALU	Value to Write data input comes from data memory

Main Control Unit

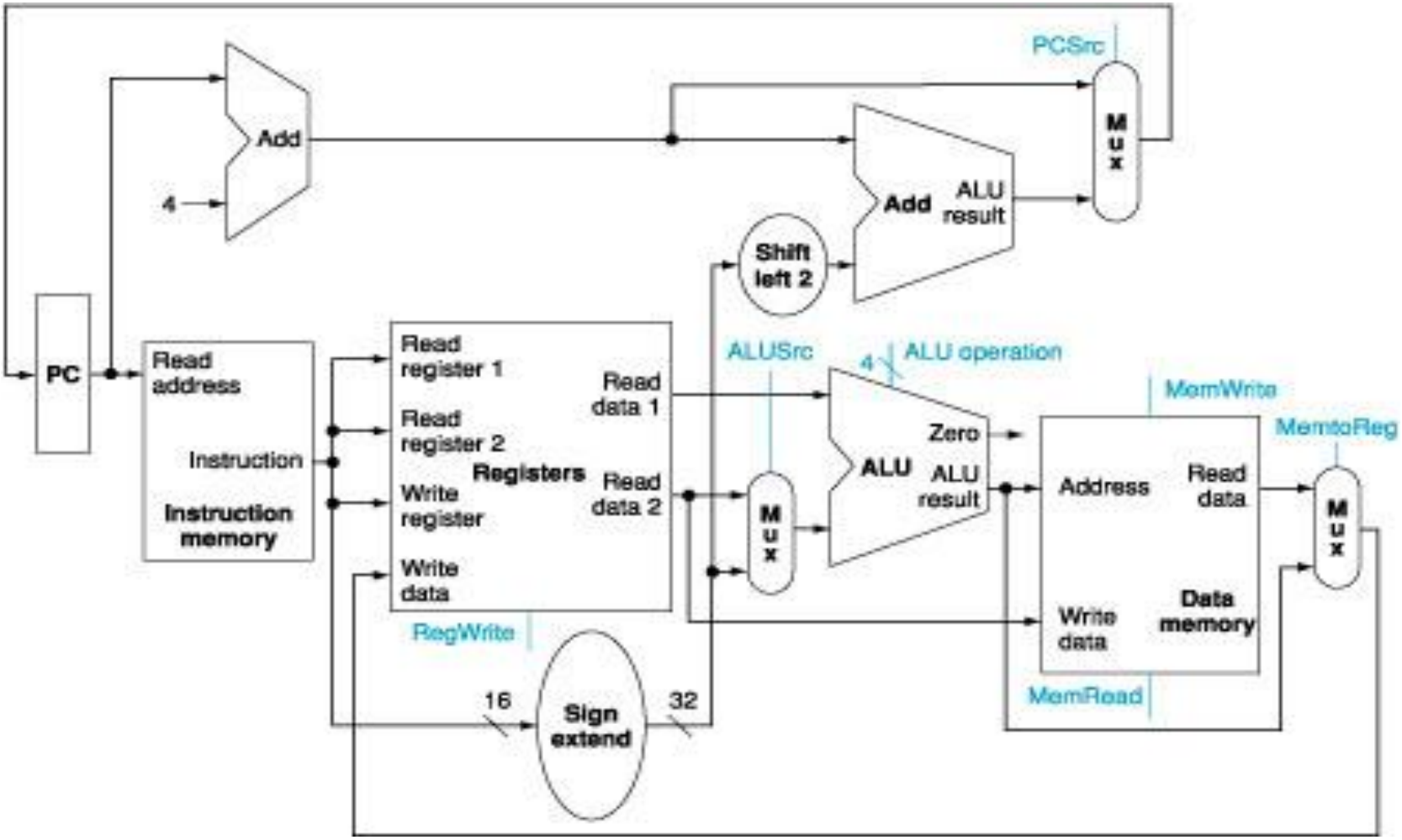
- The control signals based on different Opcode

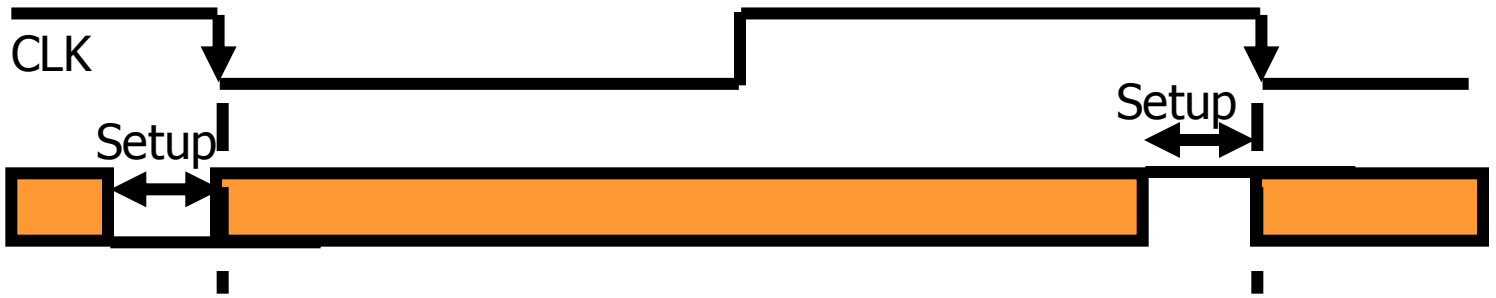
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Main Control Unit

- Control lines needed by each instruction (See Fig 5.18, p. 308)
  - R-Format (***add, sub, AND, OR, & slt***)
    - Source register : ***rs & rt***
    - Destination register: ***rd***
    - Writes a register (RegWrite = 1) but neither reads nor writes data memory
    - ALUOp for R-Type format = 10 indicating that ALU control should be generated from function field
  - When Branch:
    - Control signal = 0, PC<= PC +4;
    - Otherwise it is replaced by Branch target
  - For lw MemRead = 1,
  - For sw MemWrite = 1

Instruction	RegDest	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Banch	ALUOp1	ALUOp0
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


PCSrc
Add
4
M
u
x
Add
ALU
result
Shift
left 2
PC
Read
address
Instruction
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU operation
M
u
x
ALU
Zero
ALU
result
MemWrite
MemtoReg
Address
Read
data
Write
data
Data
memory
MemRead
M
u
x
16
Sign
extend
32


CLK
Setup
Setup

EXAMPLE

Performance of Single-Cycle Machines

Assume that the operation times for the major functional units in this implementation are the following:

- Memory units: 200 picoseconds (ps)
- ALU and adders: 100 ps
- Register file (read or write): 50 ps

Assuming that the multiplexors, control unit, PC accesses, sign extension unit, and wires have no delay, which of the following implementations would be faster and by how much?

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				

Using these critical paths, we can compute the required length for each instruction class:

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

Single Cycle Implementation

- Exercise: Assuming we have the following instruction mix:

Load	25%
Store	10%
ALU instructions	45%
Branch	15%
Jump	5%

Calculate cycle time and total program (with **L** instructions) execution time

Assuming negligible delays except for the following:

- memory (200 ps) (ps = Pico Second)
- ALU and adders (100 ps)
- Register file access (50 ps)
- Note: For single-cycle instruction, **CPI = 1**

- Answer:

(Complete answer)

Single Cycle Problems

- Single cycle instruction is not used in modern computers
- **Disadvantages:**
  - Inefficient in both performance and HW cost
  - Clock cycle must have the same length for every instruction
  - Clock cycle determined by the longest possible path
  - For complicated instruction like floating point, it will be harder
  - Functional units must be duplicated, since each can be used only once per cycle