CSCI 210: Computer Architecture
Lecture 26: Control Path

Stephen Checkoway

Slides from Cynthia Taylor

CS History: Apple Lisa

- First mass-market PC that used a graphical user interface
- Released in 1983
- Cost $9,995 (equivalent to $29,400 in 2022)
- Used the Motorola 68000 CPU, the first 32-bit CPU
- Shipped with 1 MB of RAM

Control Path

- Our data path is complicated, and we don't use each element for each instruction or use it in different ways, e.g.,
  - add and addi use the ALU but not the data memory;
  - lw, sw, add, and addi require the ALU to perform an addition whereas beq and sub require the ALU to perform a subtraction

- How do we know which elements to use or what operation to have them perform? The information is encoded in the instruction itself!
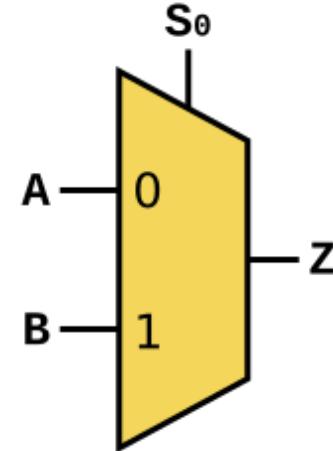
What control signals do we need?

Our data path consists of four major components
1. Instruction memory and PC register
2. Register file
3. ALU
4. Data memory
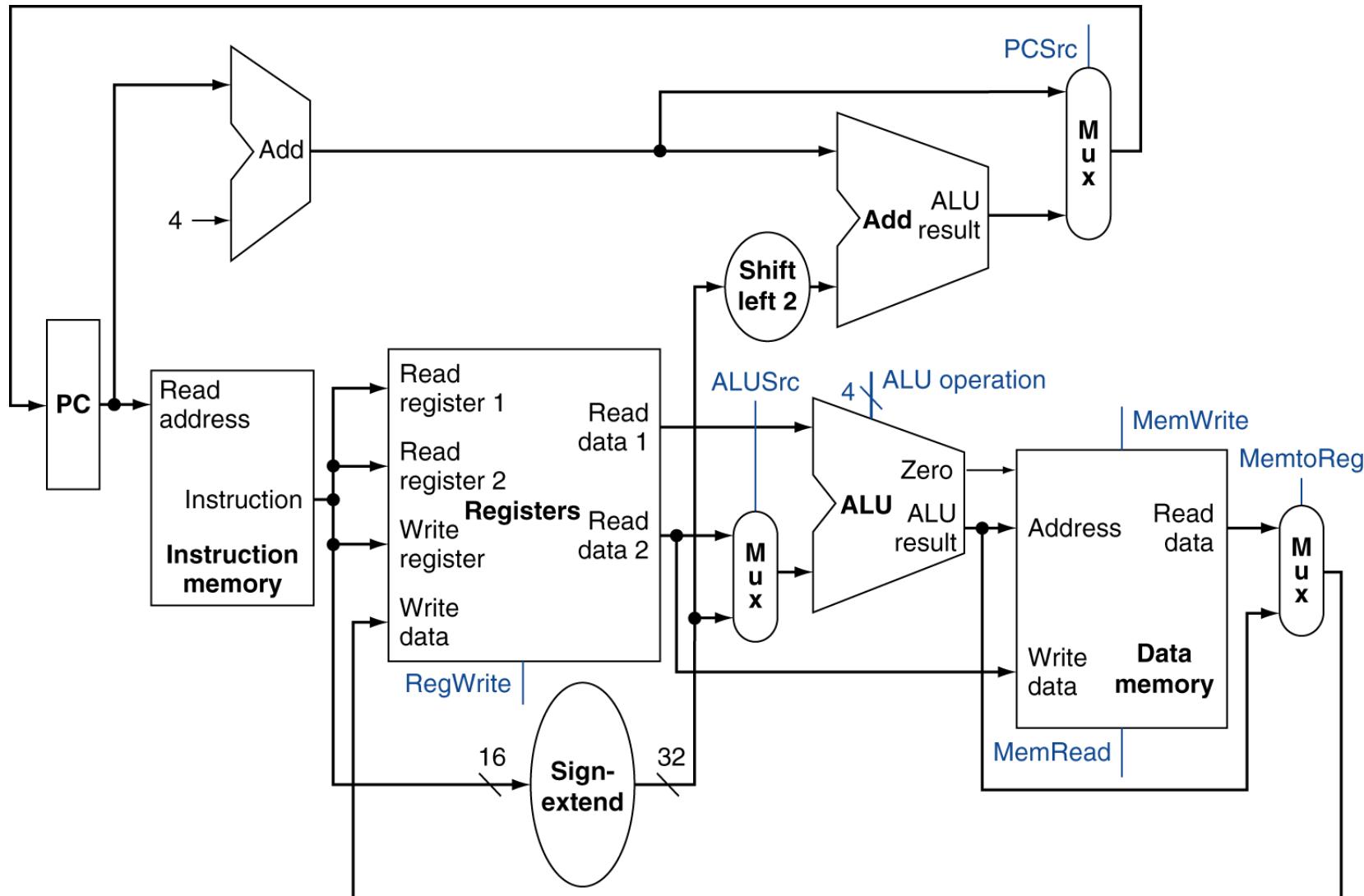
For each component, we're going to consider what control signals it needs to perform the correct operation for the instruction on the correct data

Warmup: Consider the multiplexer below. What value must the select input S_0 have for the output Z to have the same value as input B?
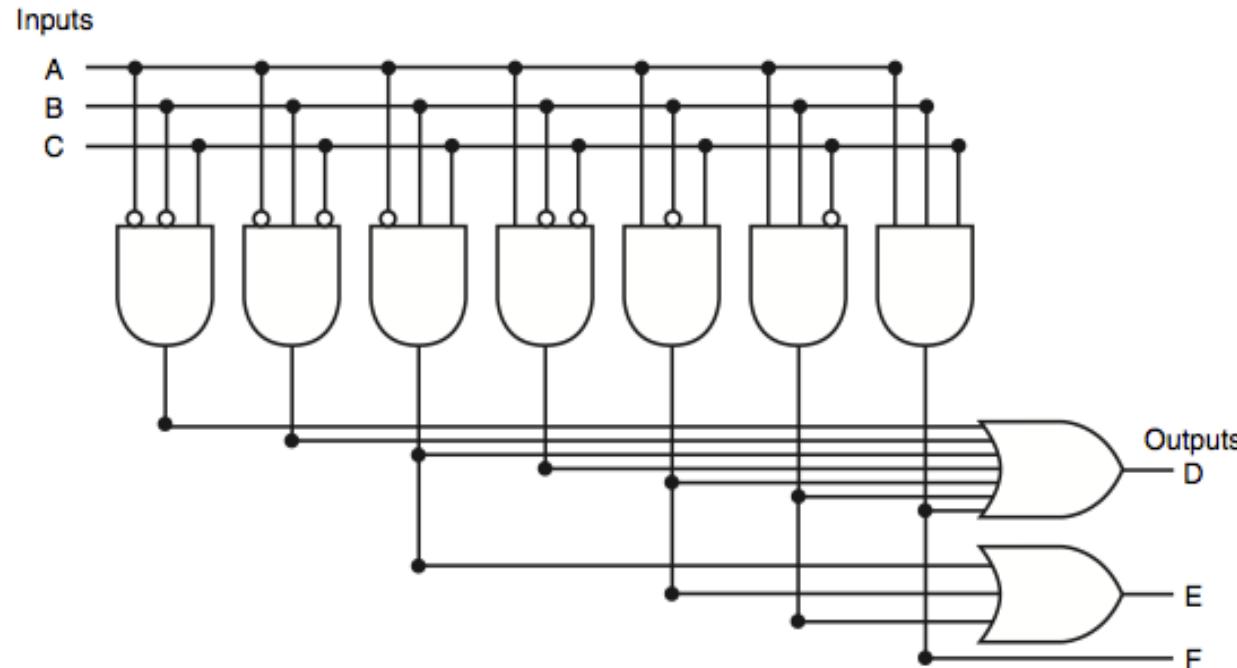

A. 0

B. 1

C. 2

D. It depends on the value of A

E. It depends on the value of both A and B

Blue inputs are some of the control signals
# Today, we're going to hook some logic up to them

Recall: PLAs

Inputs

A
B
C

Outputs
D

E

F
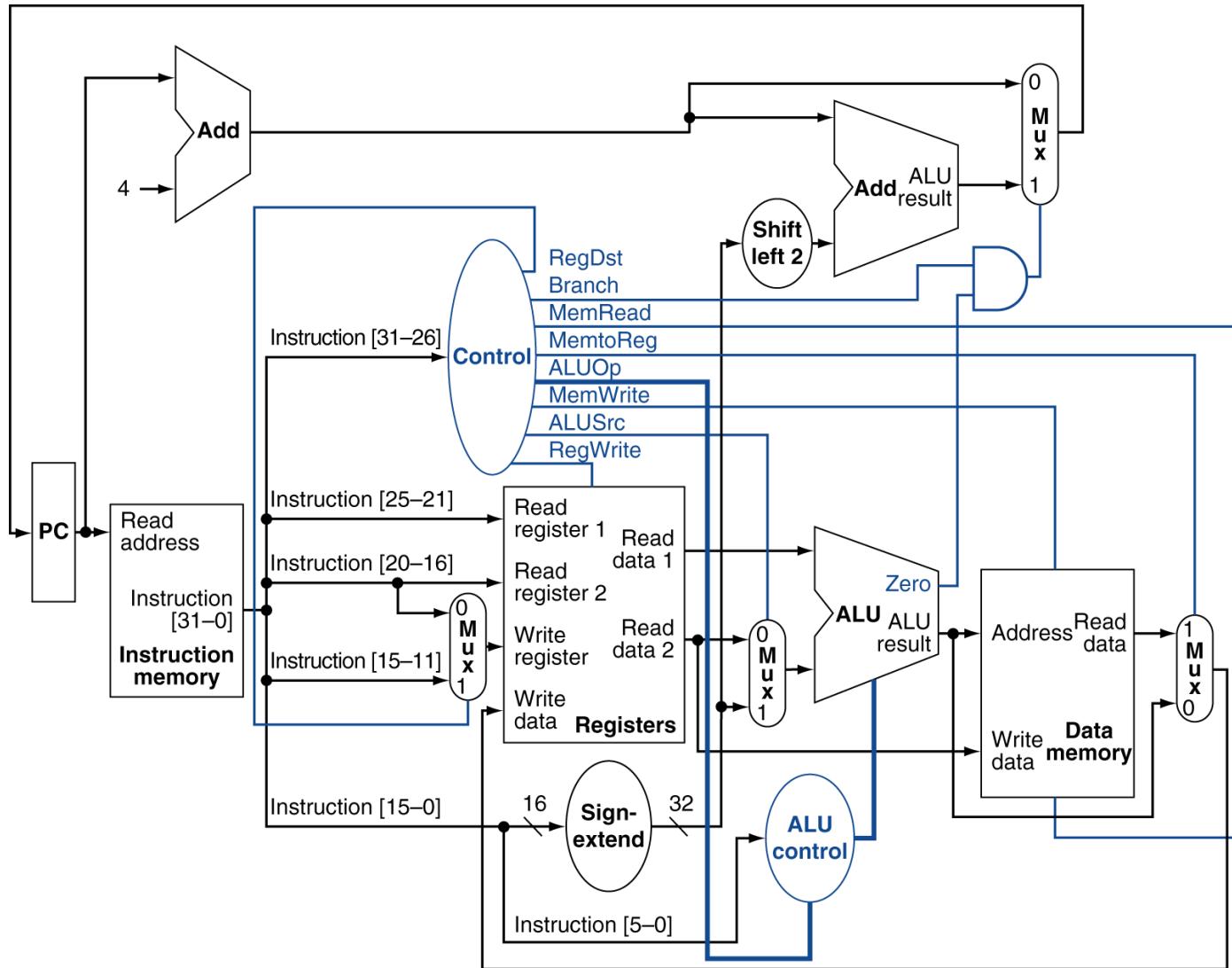
- Derived from truth table using sum of products

- Allow us to encode arbitrary functions

- **Used to derive control signals in the data path from bits in the instruction**
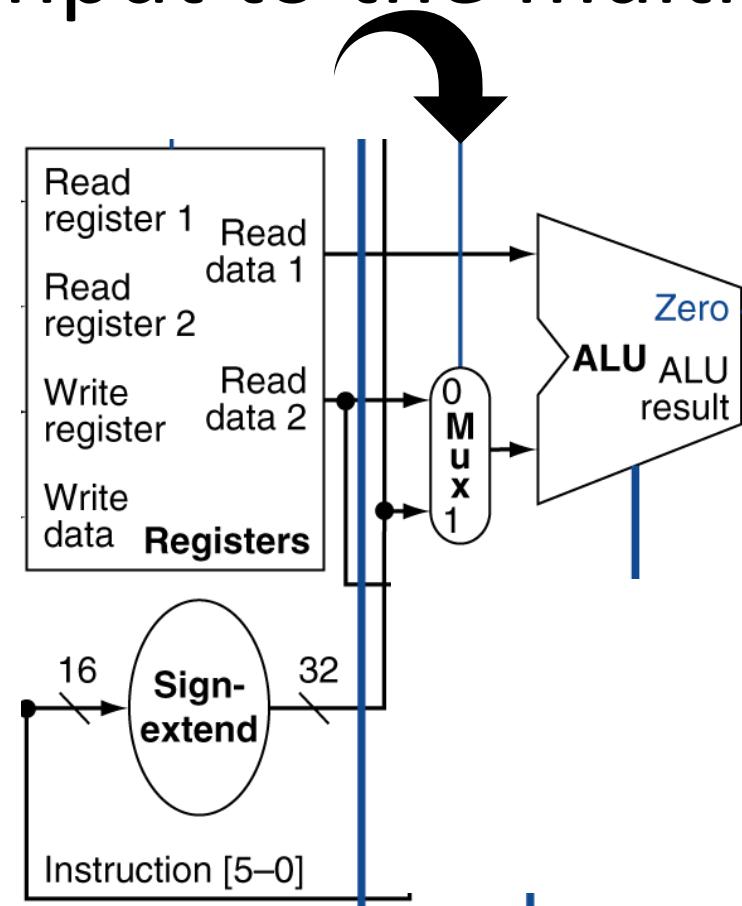
Datapath With Two Control Units

When the processor is executing the instruction

`or $t0, $t1, $t2`
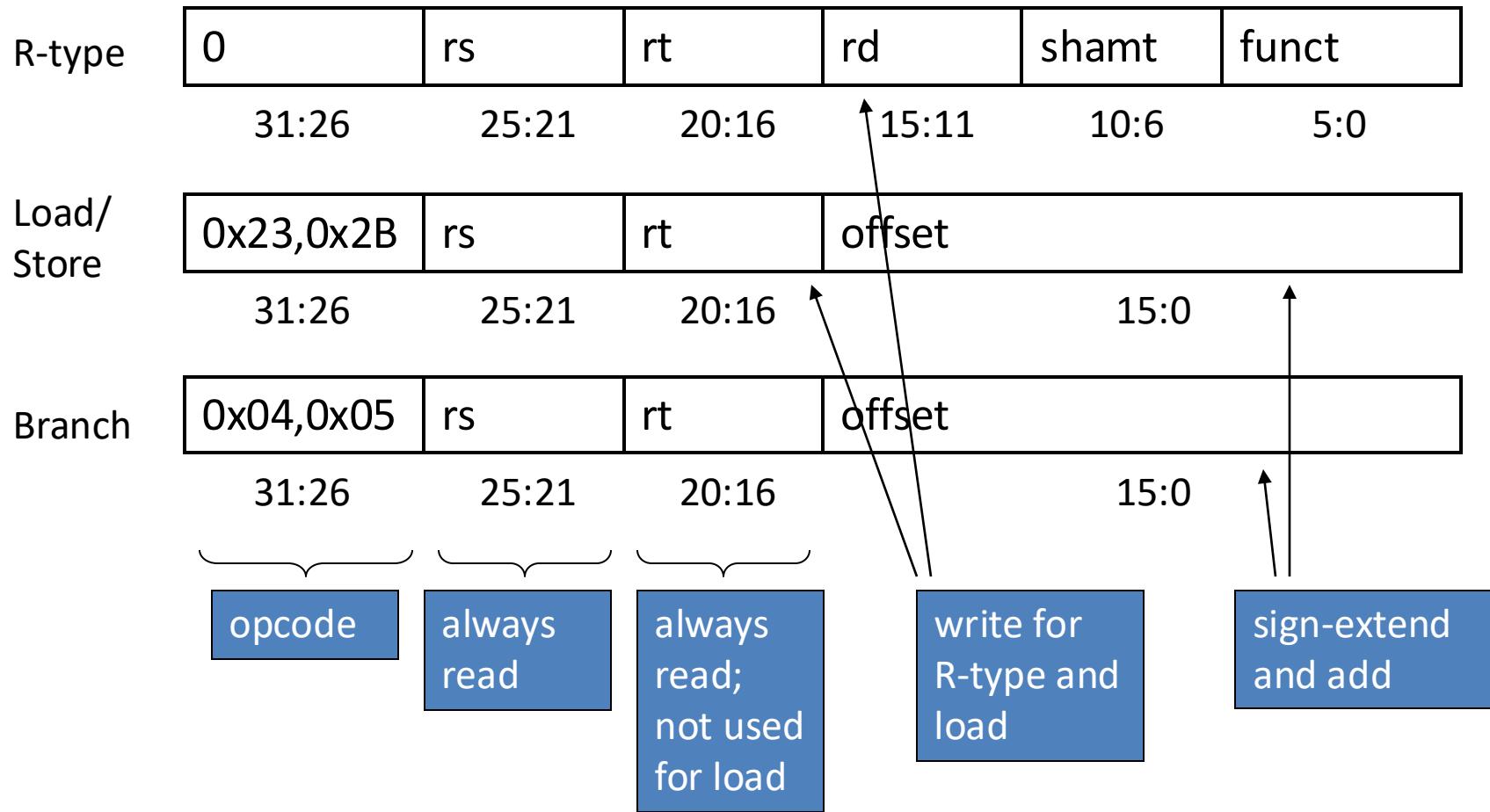
what value does the select input to the multiplexer need to have?


A. 0

B. 1

C. X (a "don't care")

The Main Control Unit

Control signals derived from instruction opcode

R-type

0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

Load/ Store

0x23,0x2B	rs	rt	offset
31:26	25:21	20:16	15:0

Branch

0x04,0x05	rs	rt	offset
31:26	25:21	20:16	15:0

opcode

always read

always read; not used for load

write for R-type and load

sign-extend and add
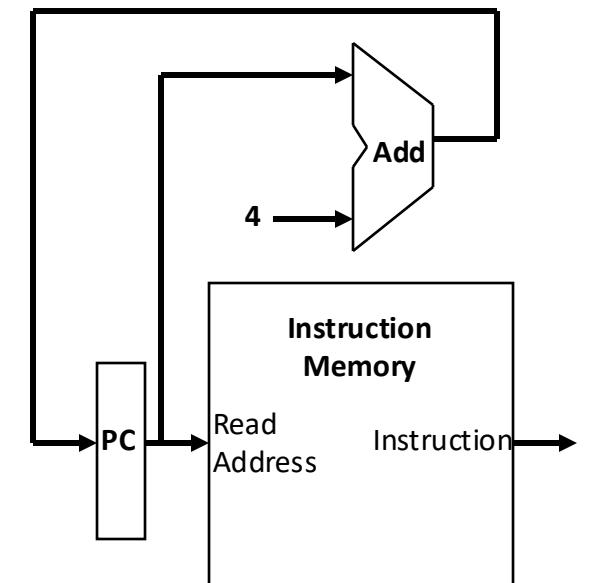
Let's derive some control signals!

- We'll tackle this systematically by following the data as it flows through the data path

- Recall that every instruction has 3 steps to be executed:
  1. Fetch the instruction from instruction memory and update the PC
  2. Decode the instruction and get the operands for the ALU out of the register file or from an immediate field in the instruction
  3. Execute the instruction and store the result back into the register file
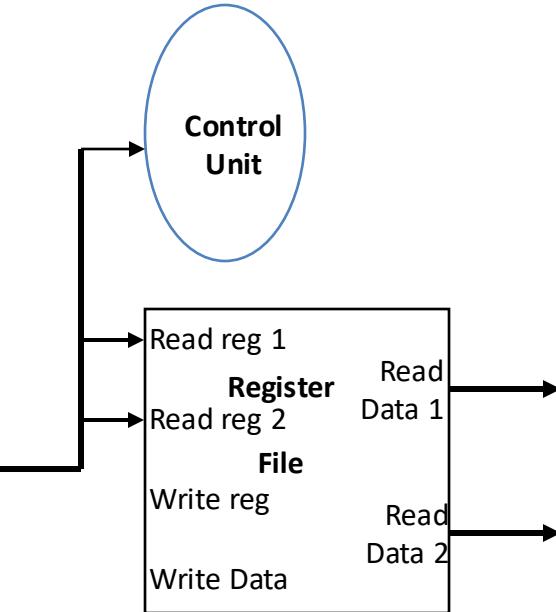
Fetching Instructions

- Read instruction from instruction memory

- Update PC value to address of next (sequential) instruction

- PC is updated every clock cycle, so it does not need an explicit write control signal just a clock signal

- Read from memory each time, so **we don't need an explicit control signal for the instruction memory or PC**

Add

4

PC

Instruction
Memory

Read
Address

Instruction

Decoding Instructions

Control Unit

Read reg 1

Register
Read reg 2
File

Write reg

Write Data

Read Data 1

Read Data 2

- Send fetched instruction's opcode to the Instruction main control unit

R-type	0	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0
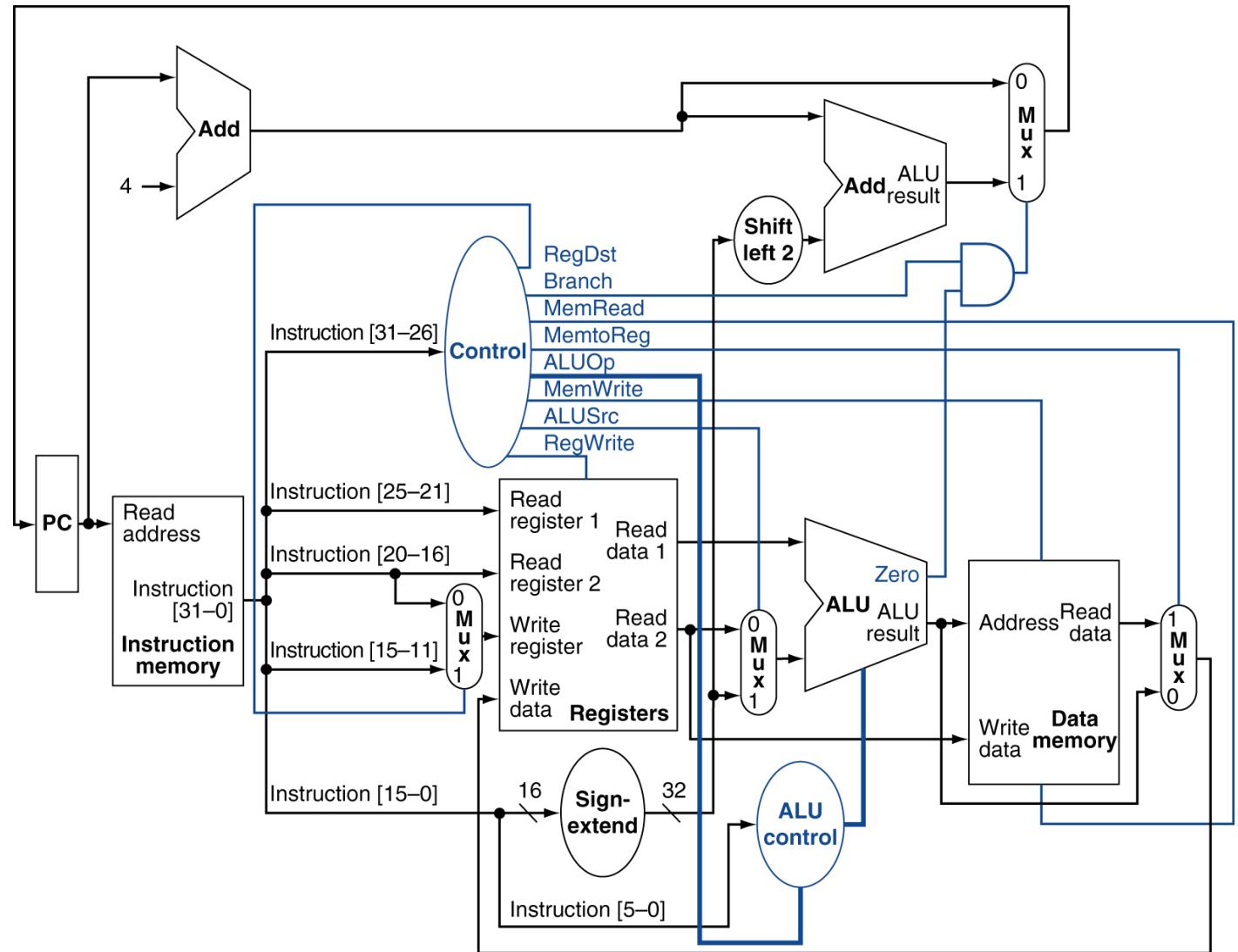
| Load/ Store	0x23,0x2B	rs	rt	address
	31:26	25:21	20:16	15:0

- Read two values from the register file
- Register numbers are contained in the instruction (rs and rt)
- **Decode happens for every instruction; this is where control signals are produced**

Producing control signals

After reading opcode

- Produce most control signals
- Includes the ALUOp control signal—which goes to the ALU control unit—and the ALUSrc control signal which selects the ALU's second operand
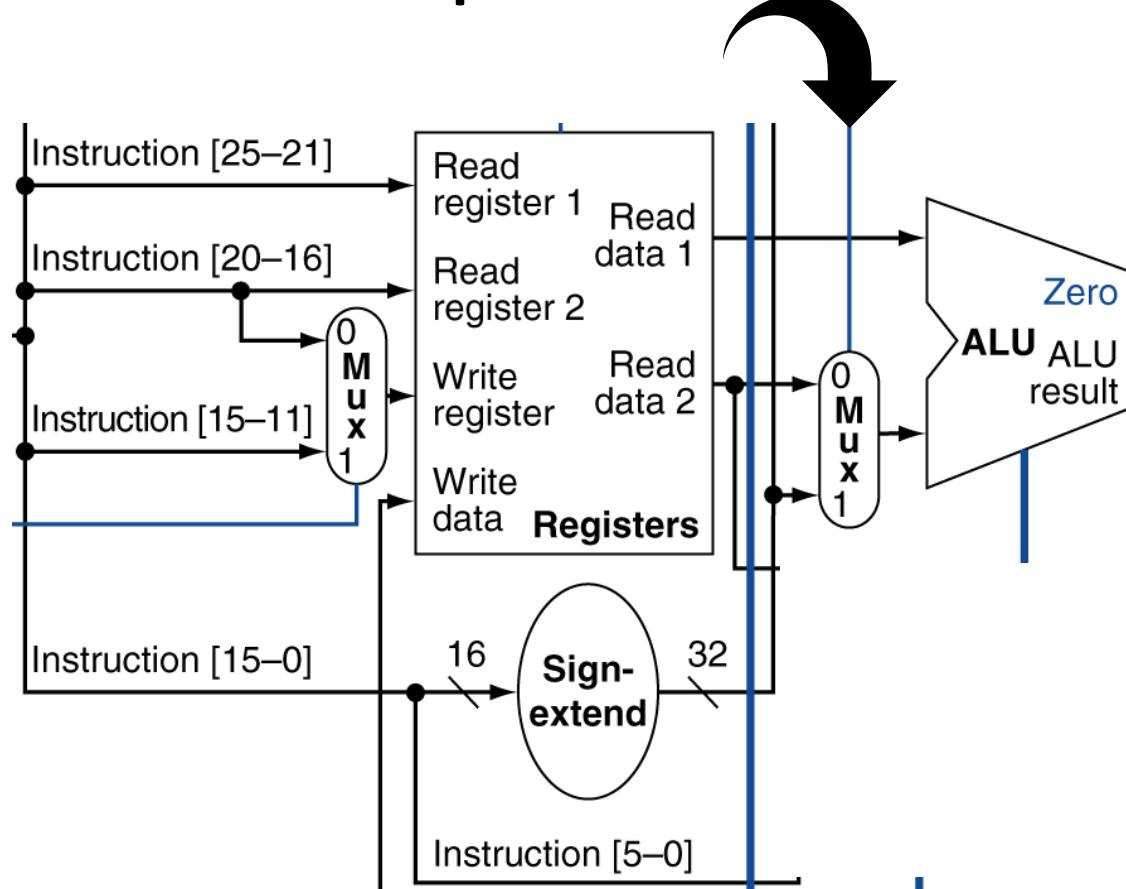
When the processor is executing the instruction

`sw $s0, 28($sp)`

what value does the select input to this multiplexer
need to have?

A. 0

B. 1
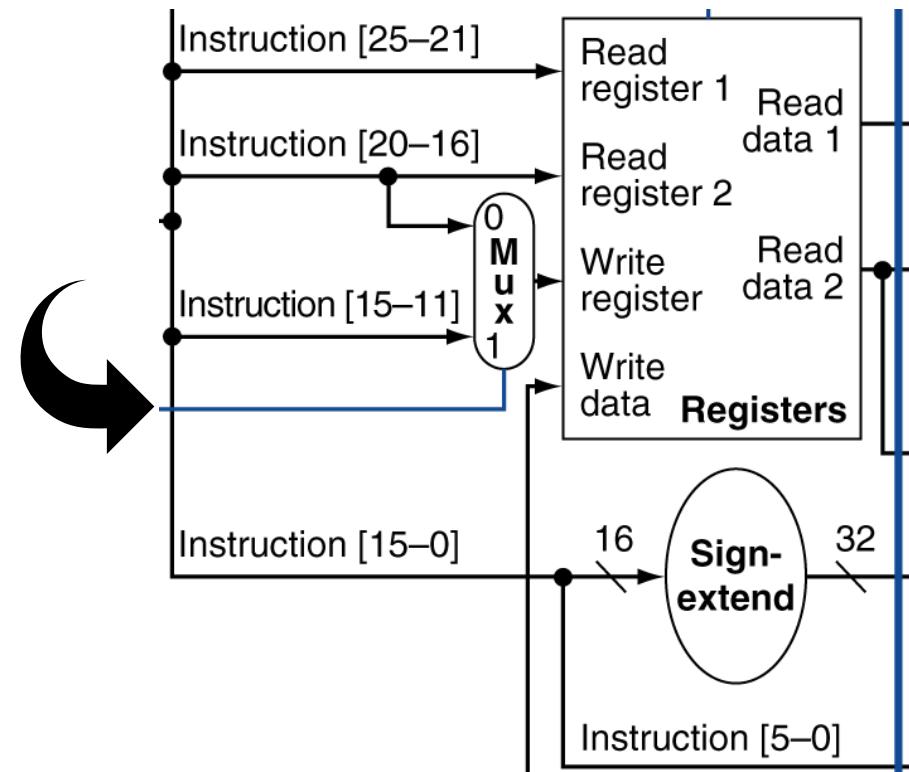
C. X (a "don't care")

When the processor is executing the instruction

`lw $s0, 28($sp)`

what value does the select input to this multiplexer need to have?

A. 0

B. 1

C. X (a "don't care")

| Opcode	rs	rt	address
31:26	25:21	20:16	15:0
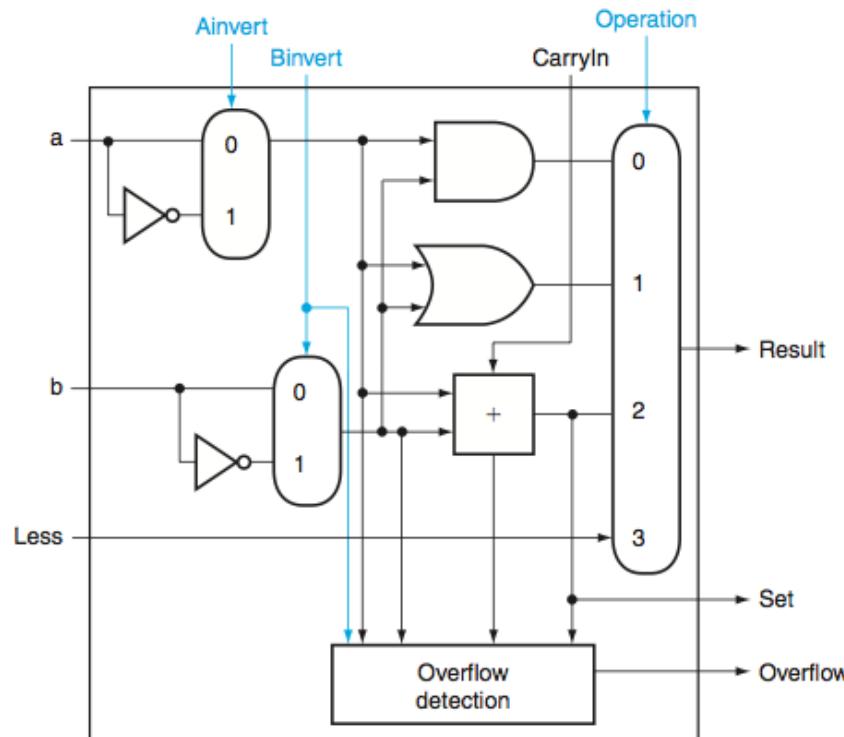
For load/store, our ALU operation will be

A. Add

B. And

C. Set less than

D. Subtract

E. None of the above
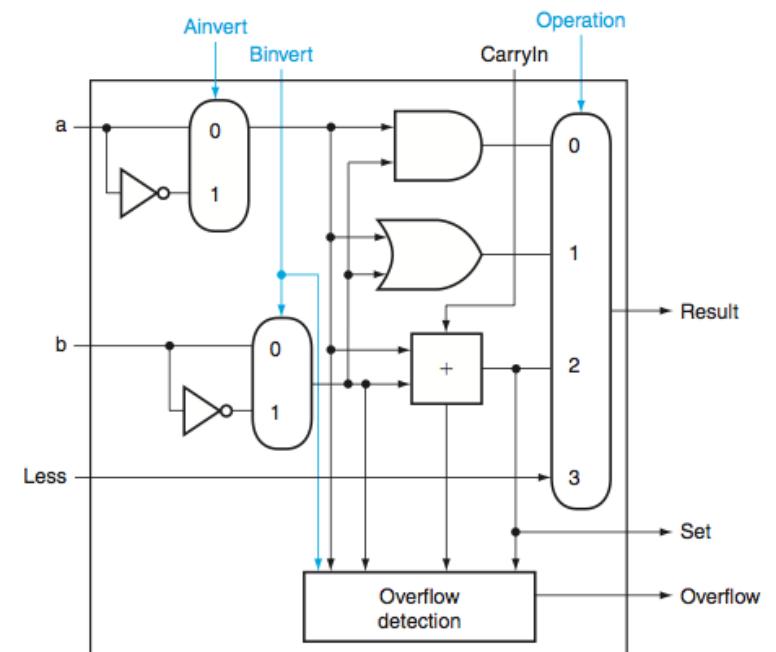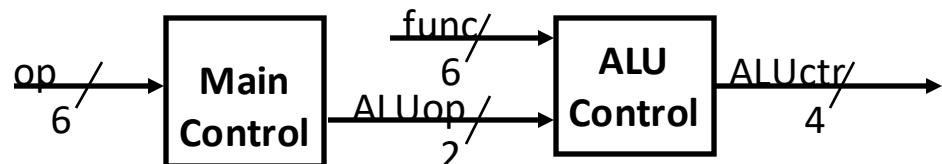
lw $t0, 4($t1)

ALU Control Unit
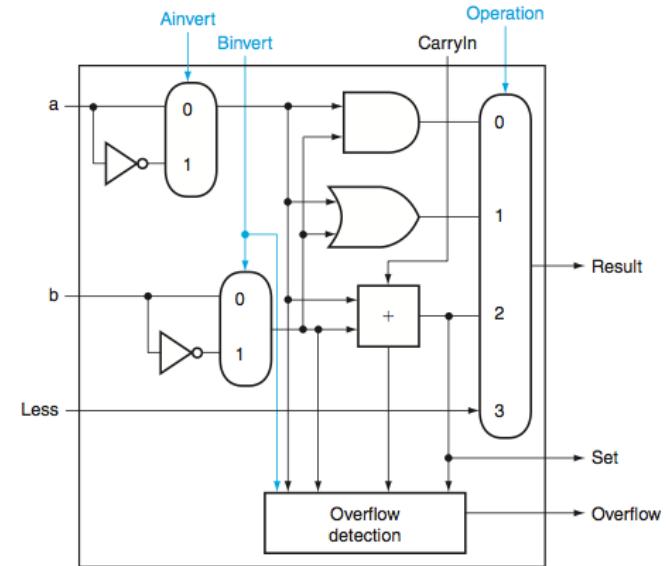
- Combinational logic (the main control unit) derives 2-bit ALUOp signal from opcode

- ALU Control Unit takes ALUOp and instruction funct field as inputs and derives a 4-bit ALU control signal

opcode	ALUOp	Operation	ALU function
lw	00	load word	add
sw	00	store word	add
beq	01	branch equal	subtract
R-type	10	arithmetic/logic	depends on funct

# ALU Control signal
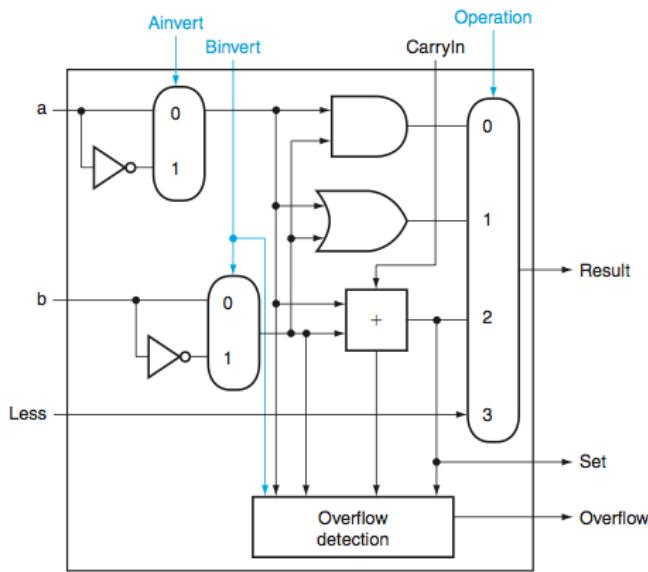
- ALU used for
  - Load/Store: op = add
  - Branch: op = subtract
  - R-type: op depends on funct field

| ALU control	Function	Ainvert	Binvert/CarryIn0	Operation
0000	AND	0	0	00
0001	OR	0	0	01
0010	add	0	0	10
0110	subtract	0	1	10
0111	set-on-less-than	0	1	11
1100	NOR	1	1	00

ALU Control
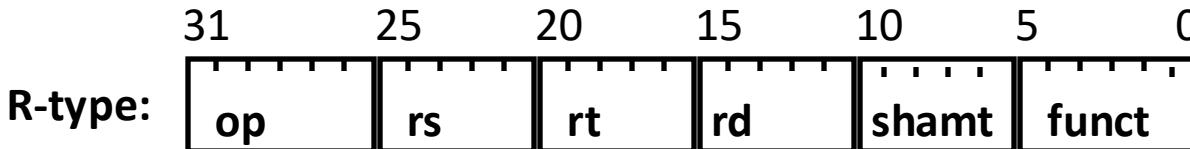
Takes as input 2-bit ALUop (derived from opcode) and 6-bit funct field; outputs 4 bits
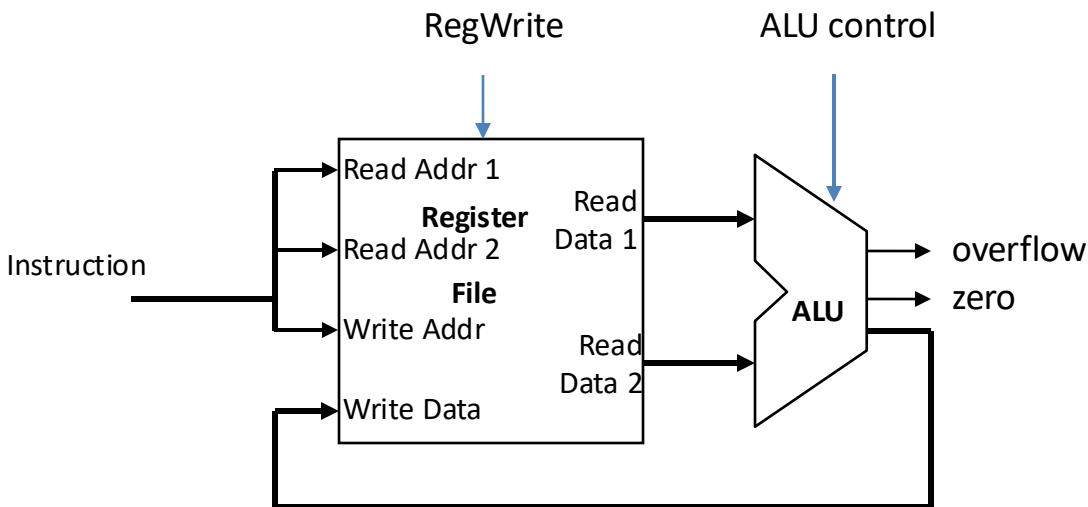
Instruction	ALUOp	funct	ALU function	Ainvert	Binvert	ALU operation
load word	00 (add)	XXXXXX	add	0	0	10 (add)
store word	00 (add)	XXXXXX	add	0	0	10 (add)
branch equal	01 (subtract)	XXXXXX	subtract	0	1	10 (add)
add	10 (r-type)	100000	add	0	0	10 (add)
subtract		100010	subtract	0	1	10 (add)
AND		100100	AND	0	0	00 (and)
OR		100101	OR	0	0	01 (or)
NOR		100111	NOR	1	1	00 (and)
set-on-less-than		101010	set-on-less-than	0	1	11 (less)

# Executing R Format Operations

- R format operations (`add, sub, slt, and, or`)
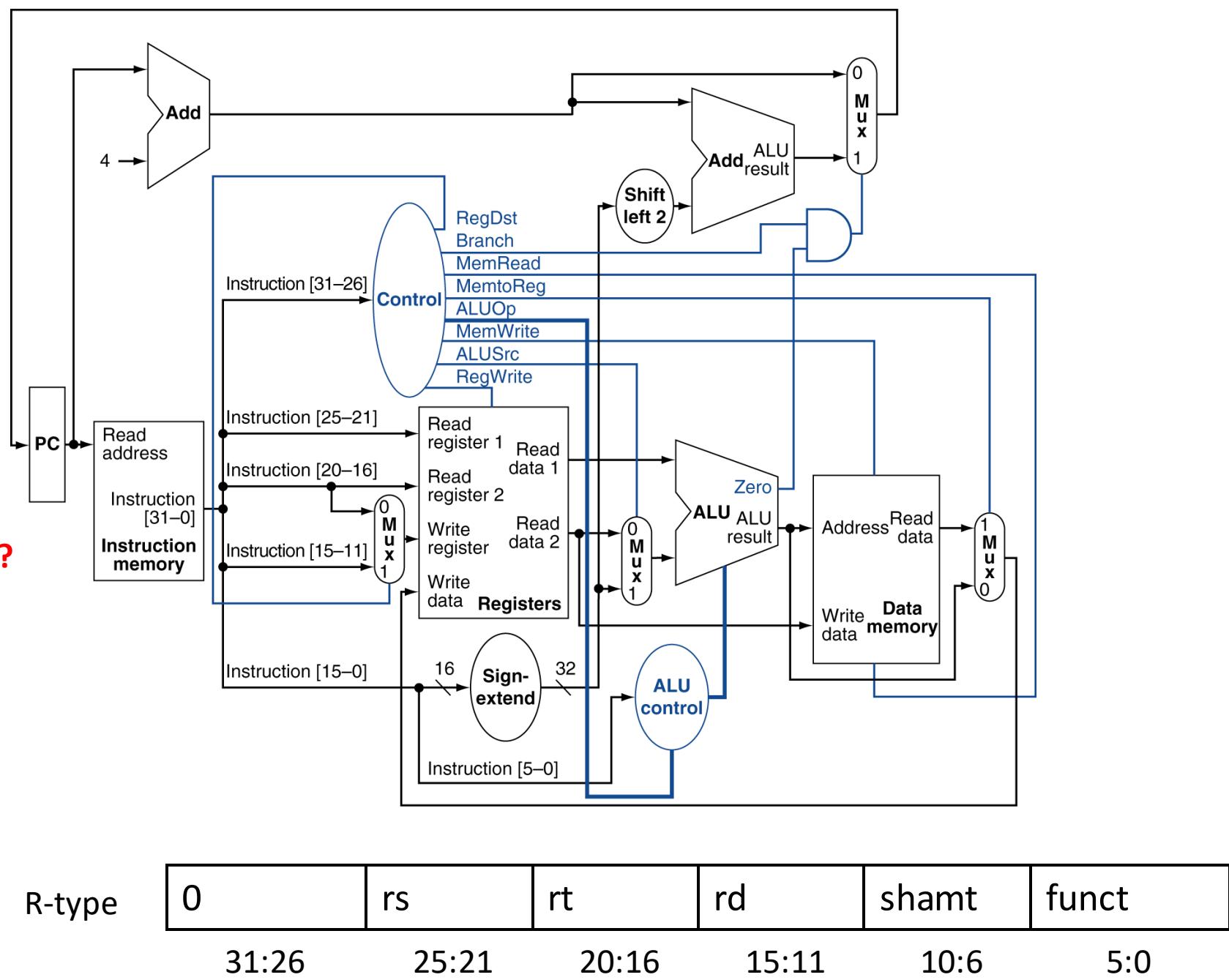
- perform operation (specified by funct) on values in rs and rt
- store the result back into the Register File (into location rd)

Note that Register File is not written every cycle (e.g., `sw`), so we need an explicit write control signal for the Register File

instruction control signals for ADD?
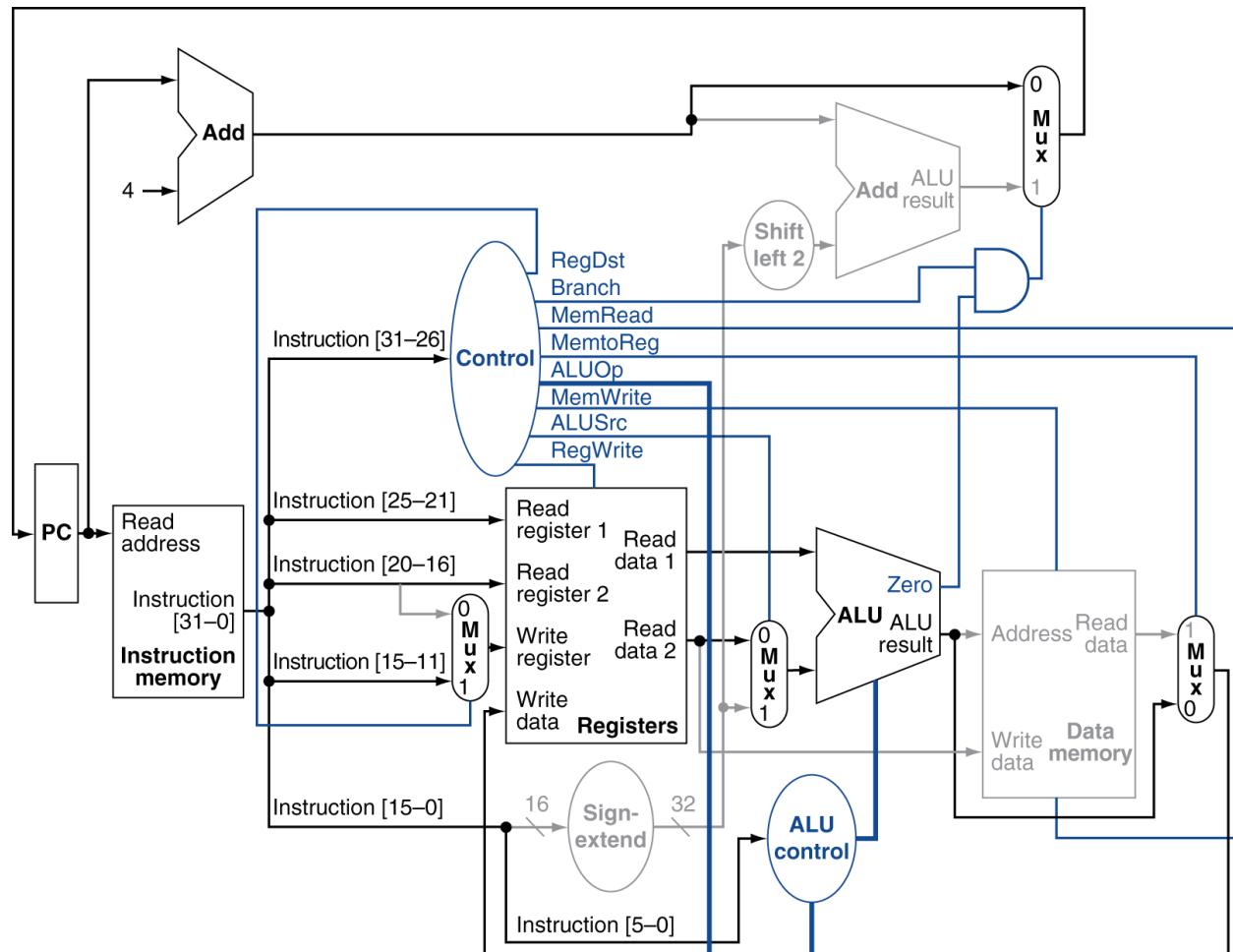
Select	RegDst	MemToReg
A	0	X
B	1	X
C	0	1
D	1	0
E	None of the above	

R-type:

0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0
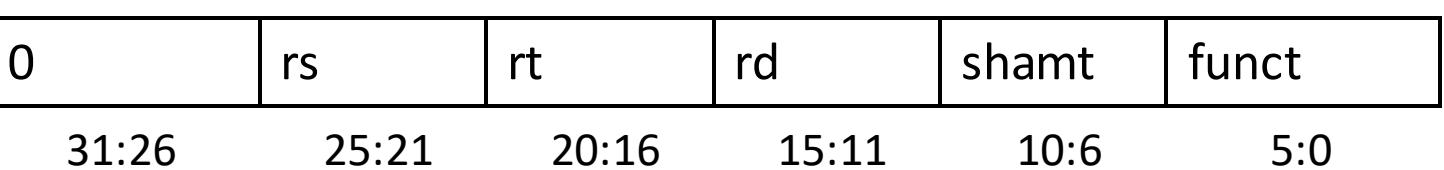
R-Type Instruction

ALUOp = 00 (add)
ALUOp = 01 (subtract)
ALUOp = 10 (R-type)

RegDst	
Branch	
MemRead	
MemtoReg	
ALUOp	
MemWrite	
ALUSrc	
RegWrite	

| R-type	0	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0
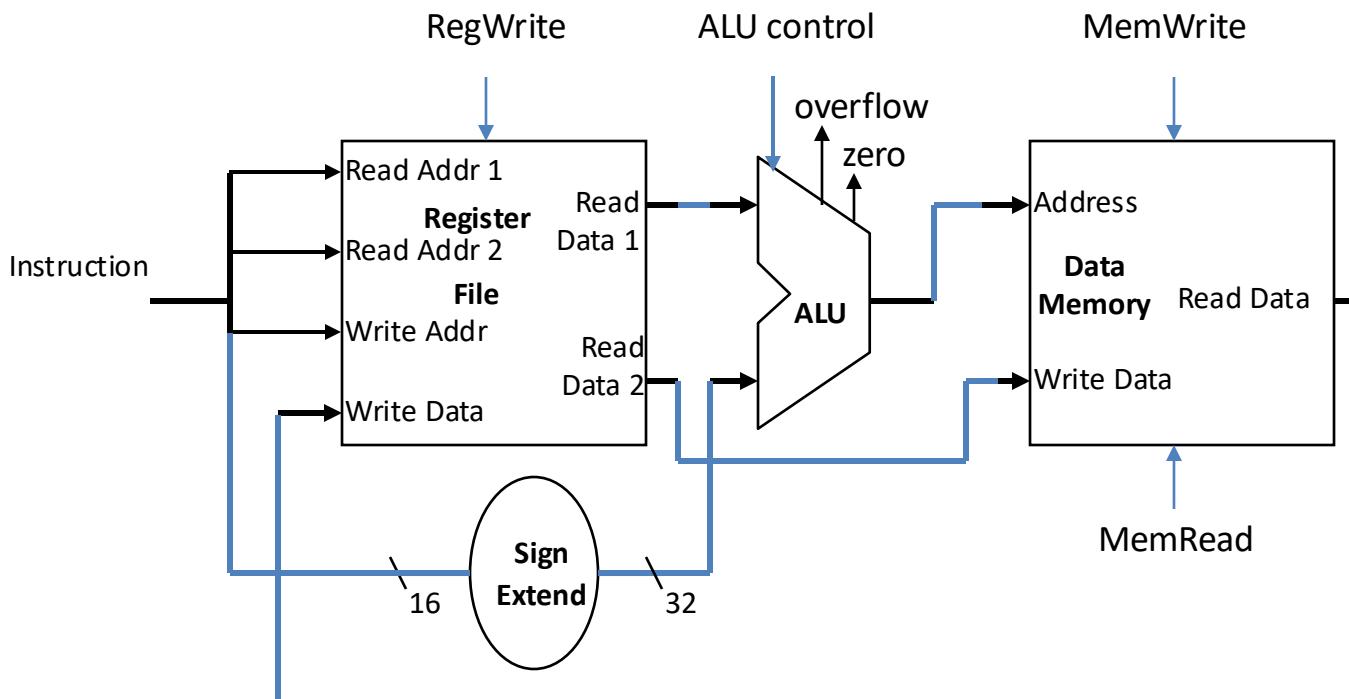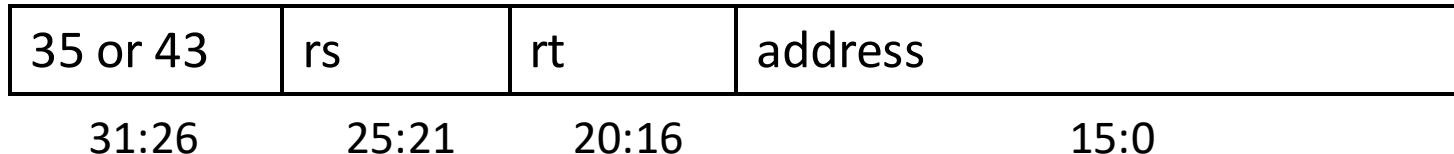
Executing Load and Store Operations
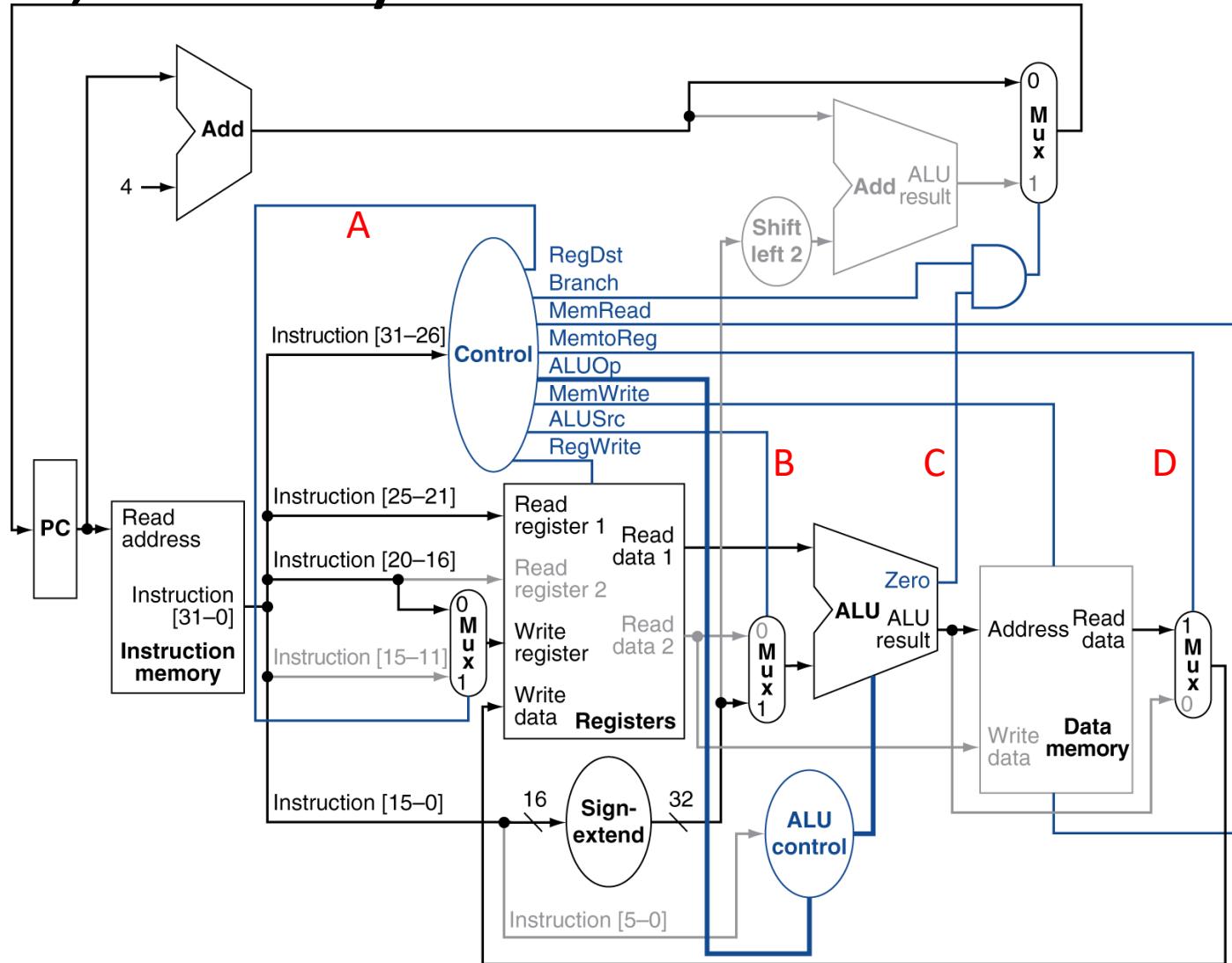
- compute memory address by adding base register to 16-bit signed-extended offset field
- store value written to the Data Memory
- load value read from the Data Memory, written to the Register File

Load/ Store	35 or 43	rs	rt	address
	31:26	25:21	20:16	15:0

Which wire, if always set to 1 would break lw?

Load/Store
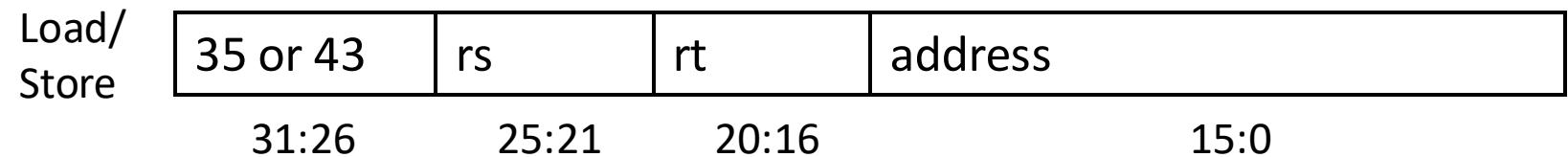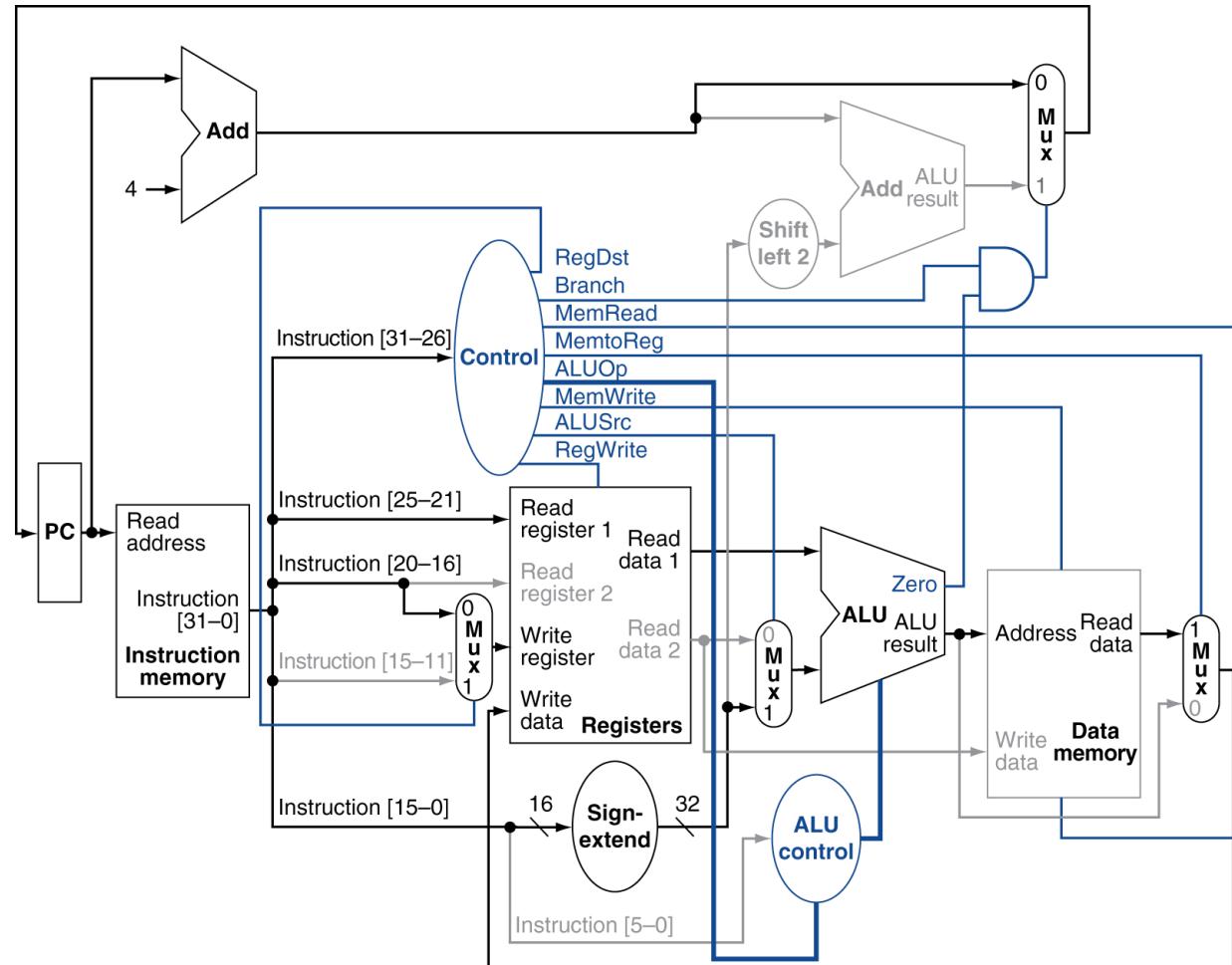
35 or 43	rs	rt	address
31:26	25:21	20:16	15:0

Load Instruction
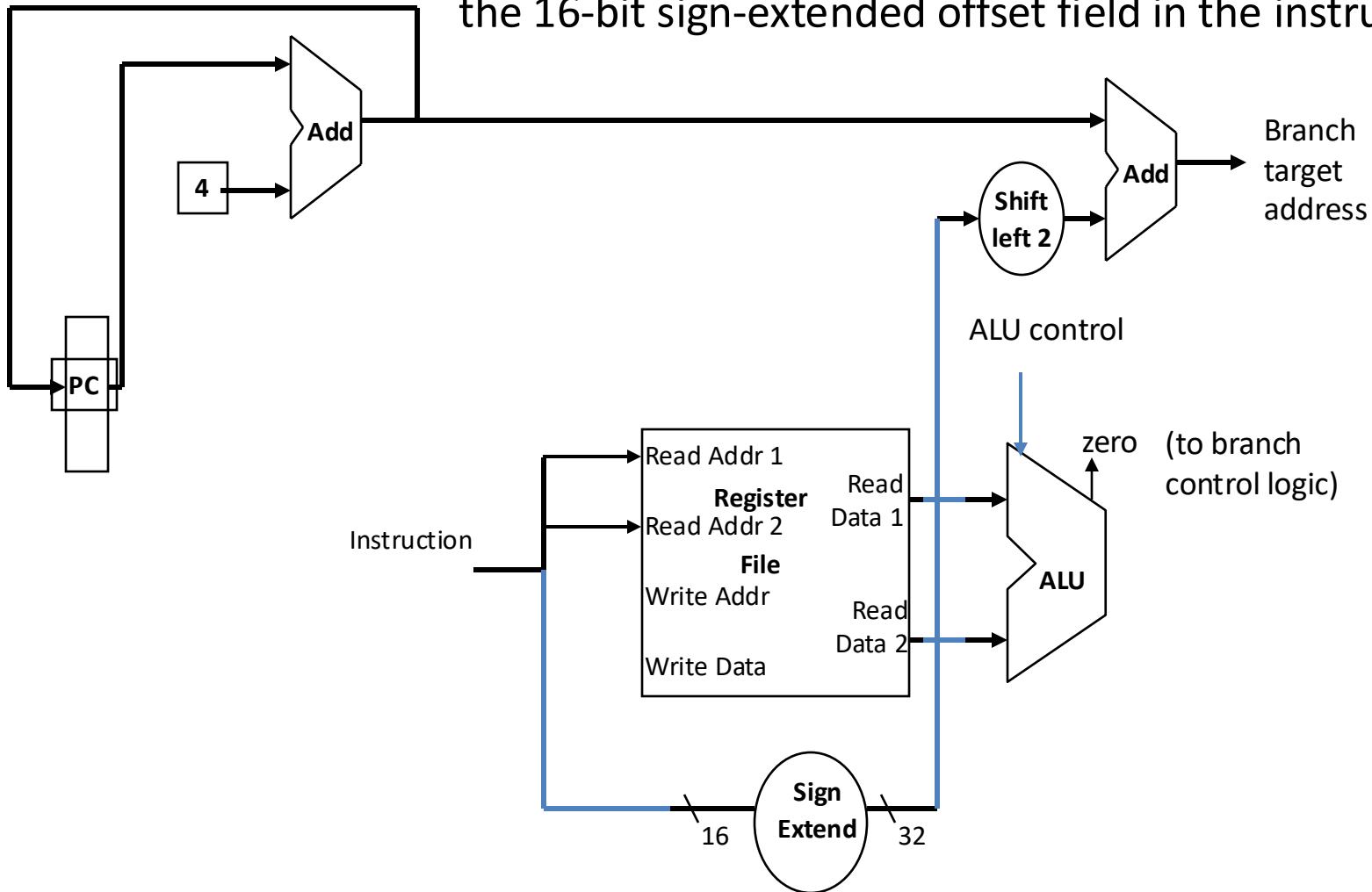
ALUOp = 00 (add)
ALUOp = 01 (subtract)
ALUOp = 10 (R-type)

RegDst	
Branch	
MemRead	
MemtoReg	
ALUOp	
MemWrite	
ALUSrc	
RegWrite	

Load/Store	35 or 43	rs	rt	address
	31:26	25:21	20:16	15:0

Executing Branch Operations
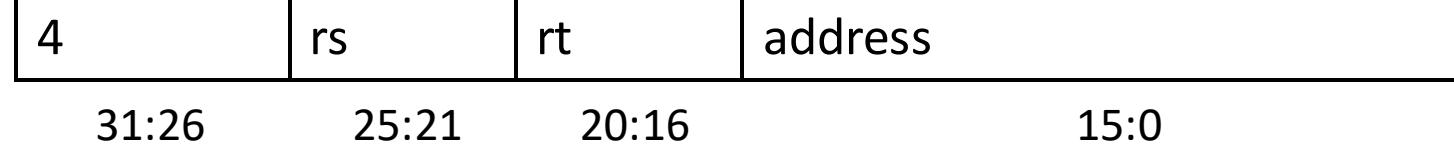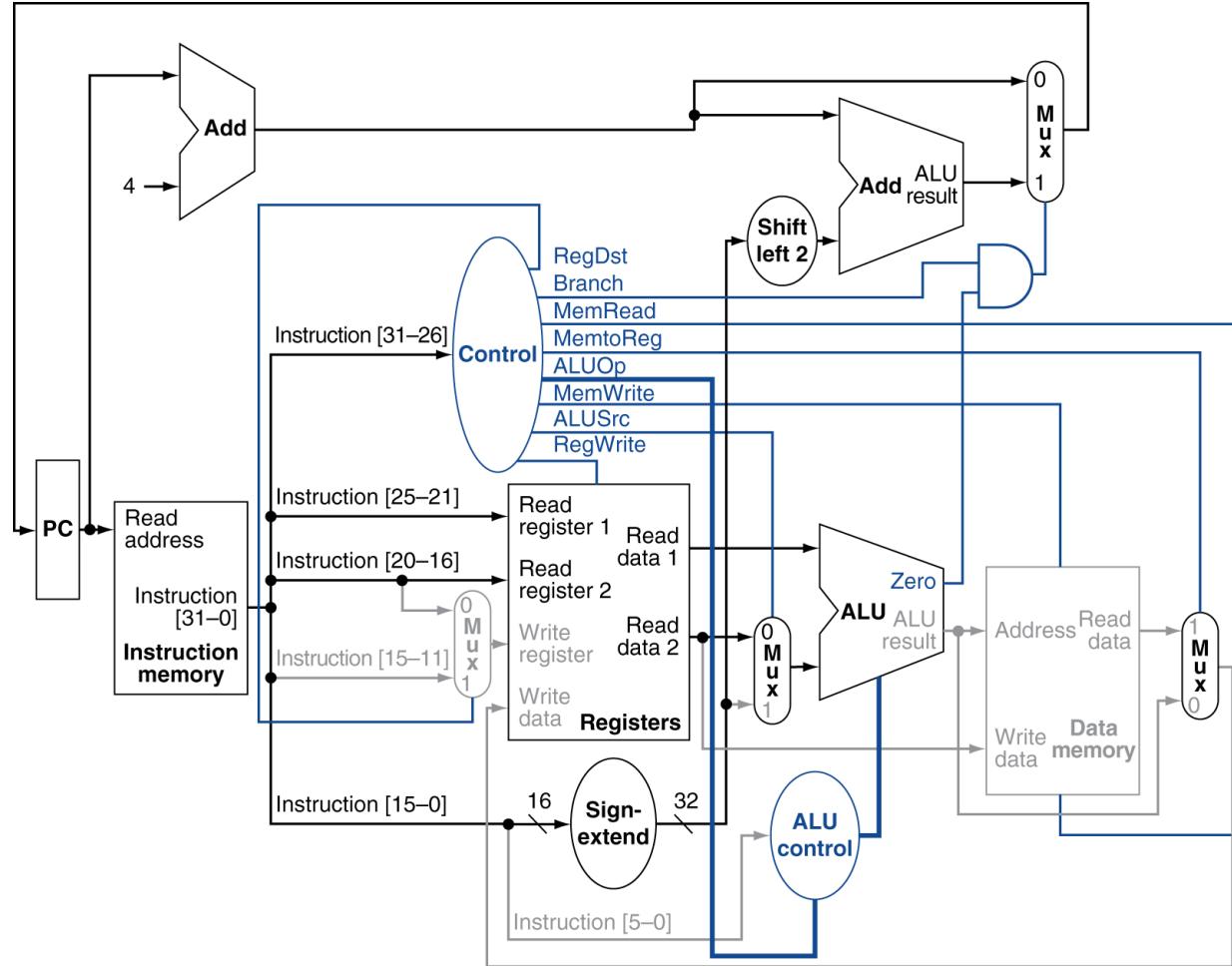
- Branch operations involve
  - compare the operands read from the Register File during decode for equality (**zero** ALU output)
  - compute the branch target address by adding the updated PC to the 16-bit sign-extended offset field in the instruction

# Branch-on-Equal Instruction

ALUOp = 00 (add)
ALUOp = 01 (subtract)
ALUOp = 10 (R-type)

RegDst	
Branch	
MemRead	
MemtoReg	
ALUOp	
MemWrite	
ALUSrc	
RegWrite	

4	rs	rt	address
31:26	25:21	20:16	15:0

Control Truth Table
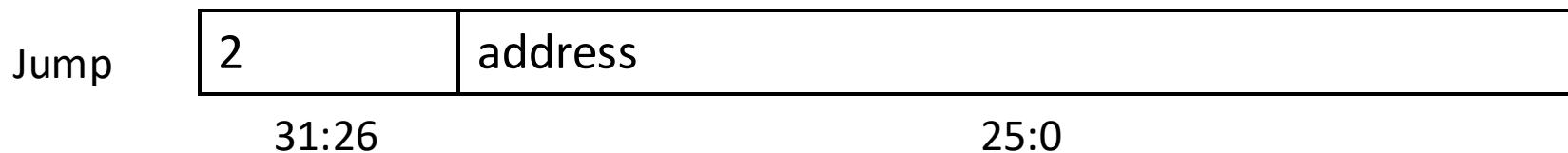
		R-format	lw	sw	beq
Opcode		000000	100011	101011	000100
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1
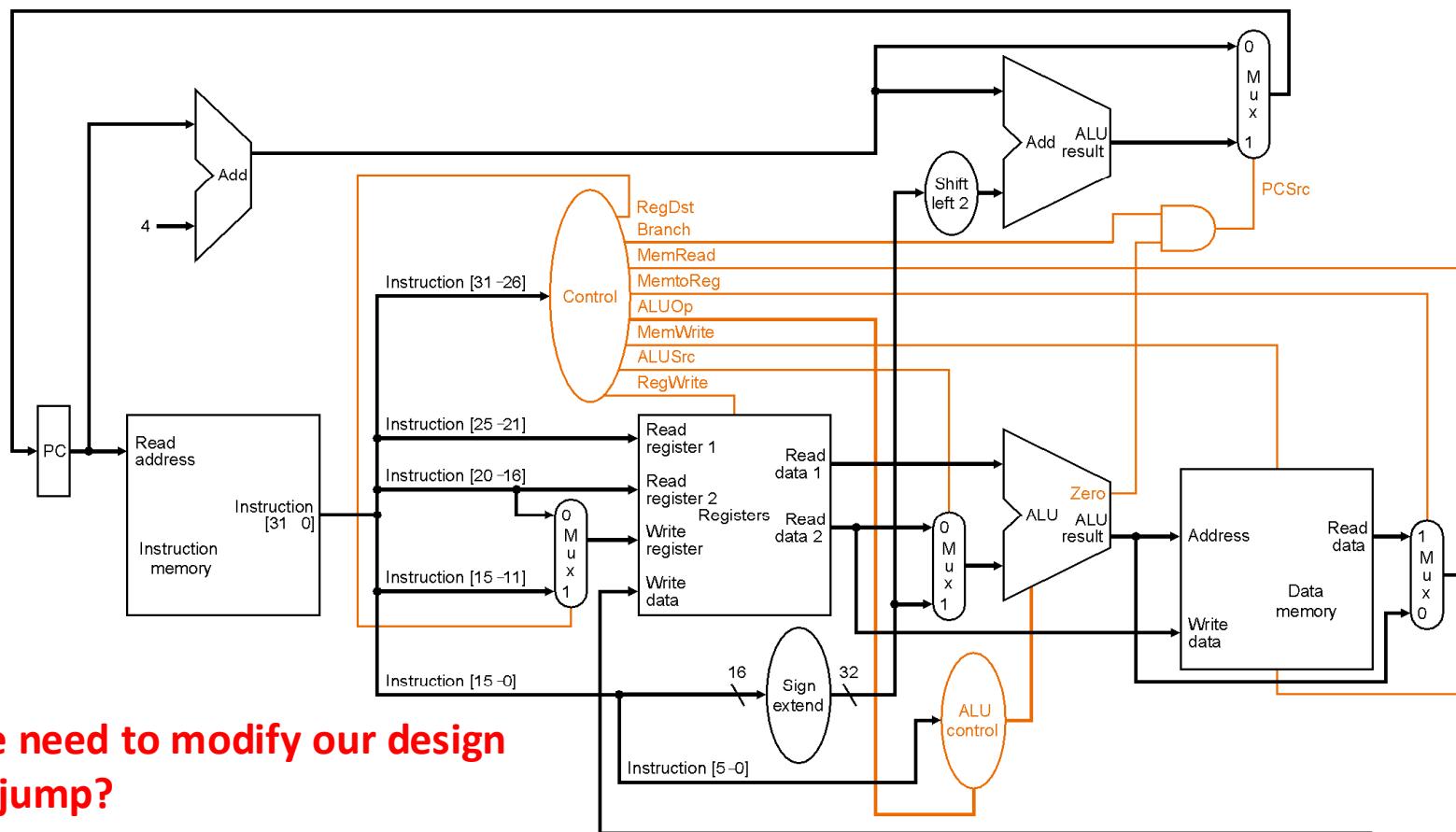
ALUOp = 00 (add)
ALUOp = 01 (subtract)
ALUOp = 10 (R-type, use funct to determine ALUCtrl signals)

Implementing Jumps

Jump	2	address
	31:26	25:0
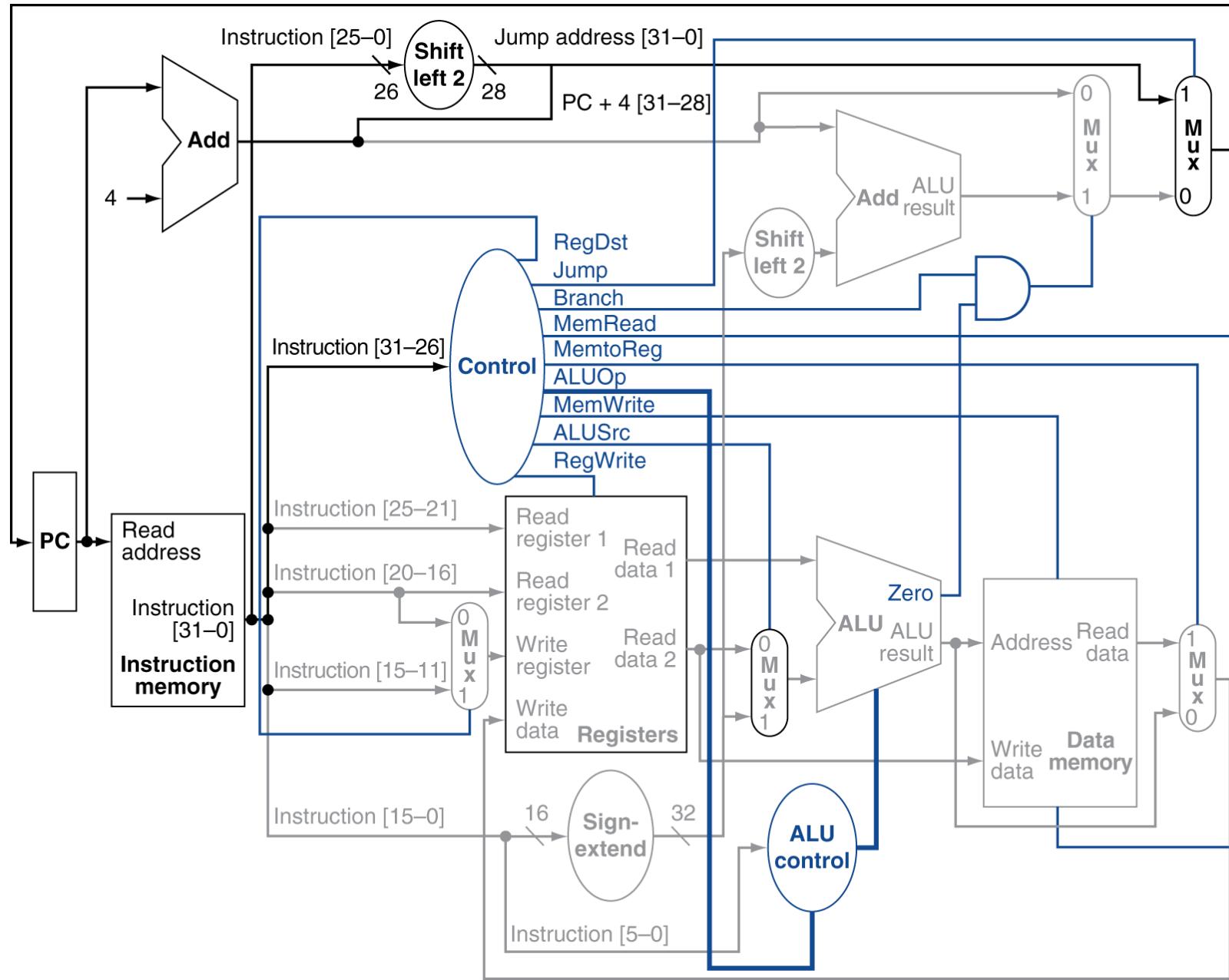
- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of PC + 4
  - 26-bit jump address
  - 00

Do we need to modify our design to do jump?

Select	Best Answer
A	Yes – we need both new control and datapath.
B	Yes – we need just datapath.
C	No – but we should for better performance.
D	No – just changing control signals is fine.
E	Single cycle can't do jump register.
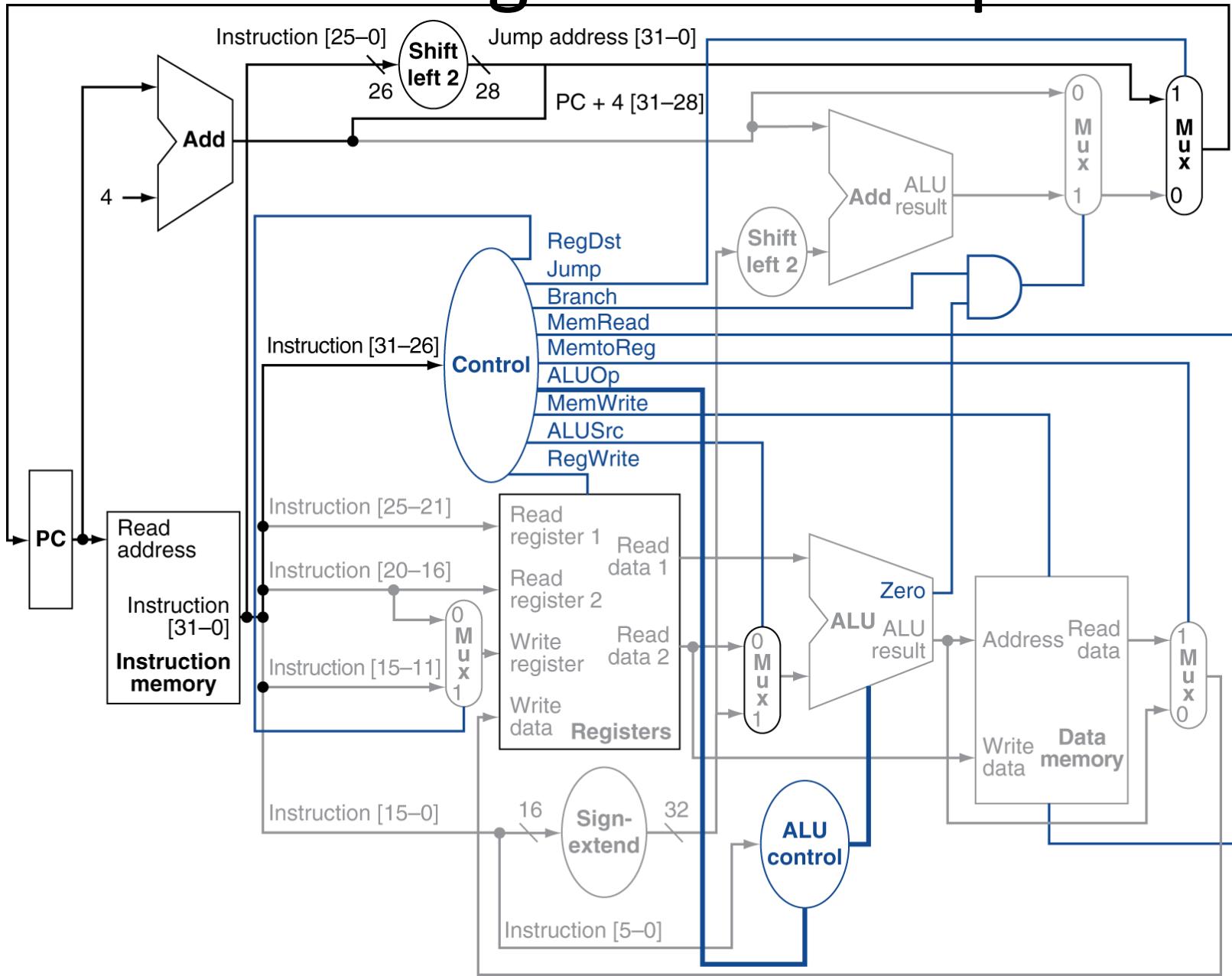
Datapath With Jumps Added
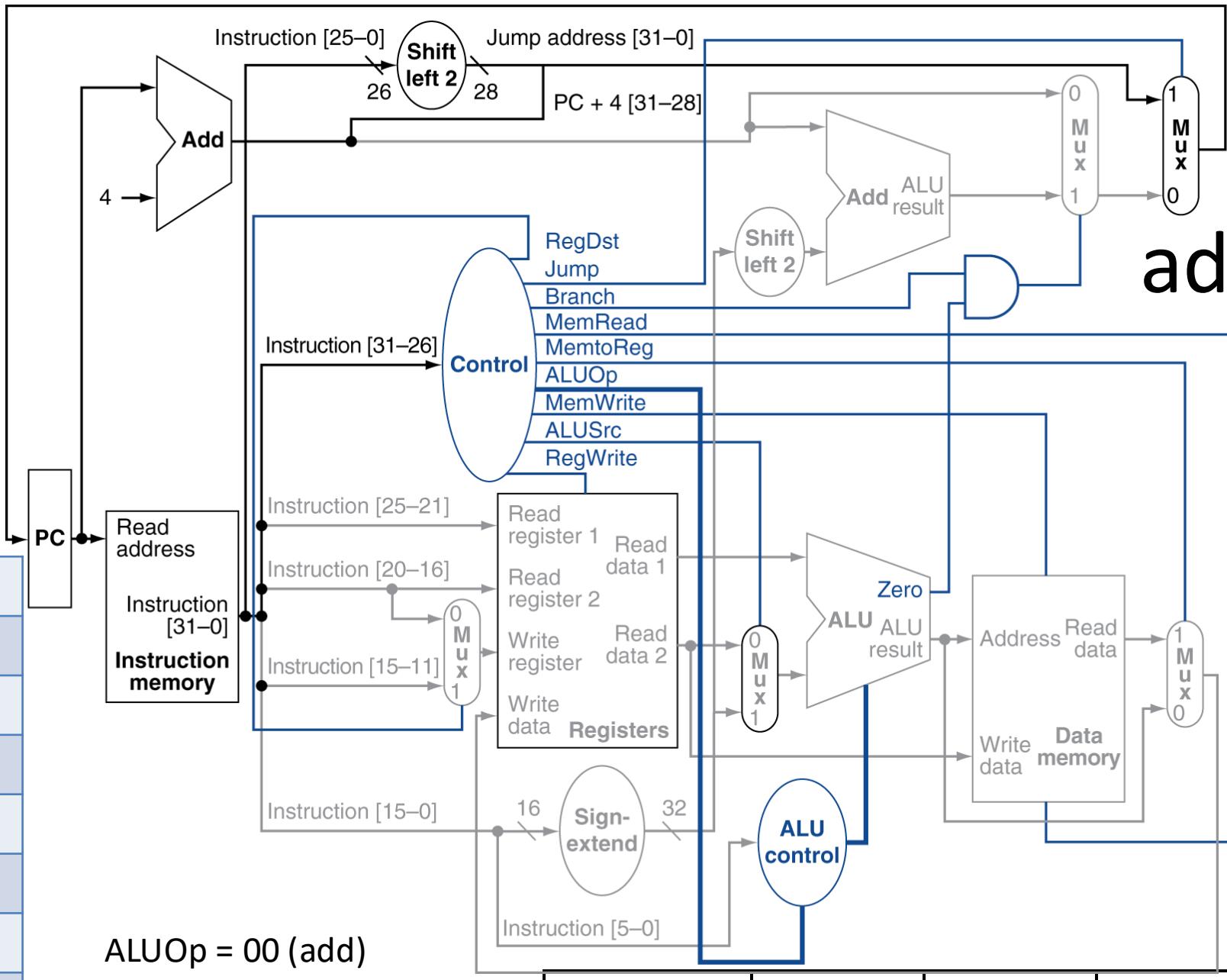
What will the Signals for Jump be?

ALUOp = 00 (add)
ALUOp = 01 (subtract)
ALUOp = 10 (R-type)

RegDst	
Jump	
Branch	
MemRead	
MemtoReg	
ALUOp	
MemWrite	
ALUSrc	
RegWrite	

addi $t1, $t2, 6

$t2 holds 5

ALUOp = 00 (add)
ALUOp = 01 (subtract)
ALUOp = 10 (R-type)

RegDst	
Jump	
Branch	
MemRead	
MemtoReg	
ALUOp	
MemWrite	
ALUSrc	
RegWrite	

Op = 0x08	rs = 10	Rt = 9	Imm = 6
31:26	25:21	20:16	15:0