

Chapter 4

The Processor



Introduction

- CPU performance factors
 - Instruction count
 - CPI and Cycle time
- We will examine two MIPS implementations
 - A single-cycle implementation
 - A pipelined version
- Simple subset, shows most aspects
 - Memory reference: lw, sw
 - Arithmetic/logical: add, sub, and, or, slt
 - Control transfer: beq, j

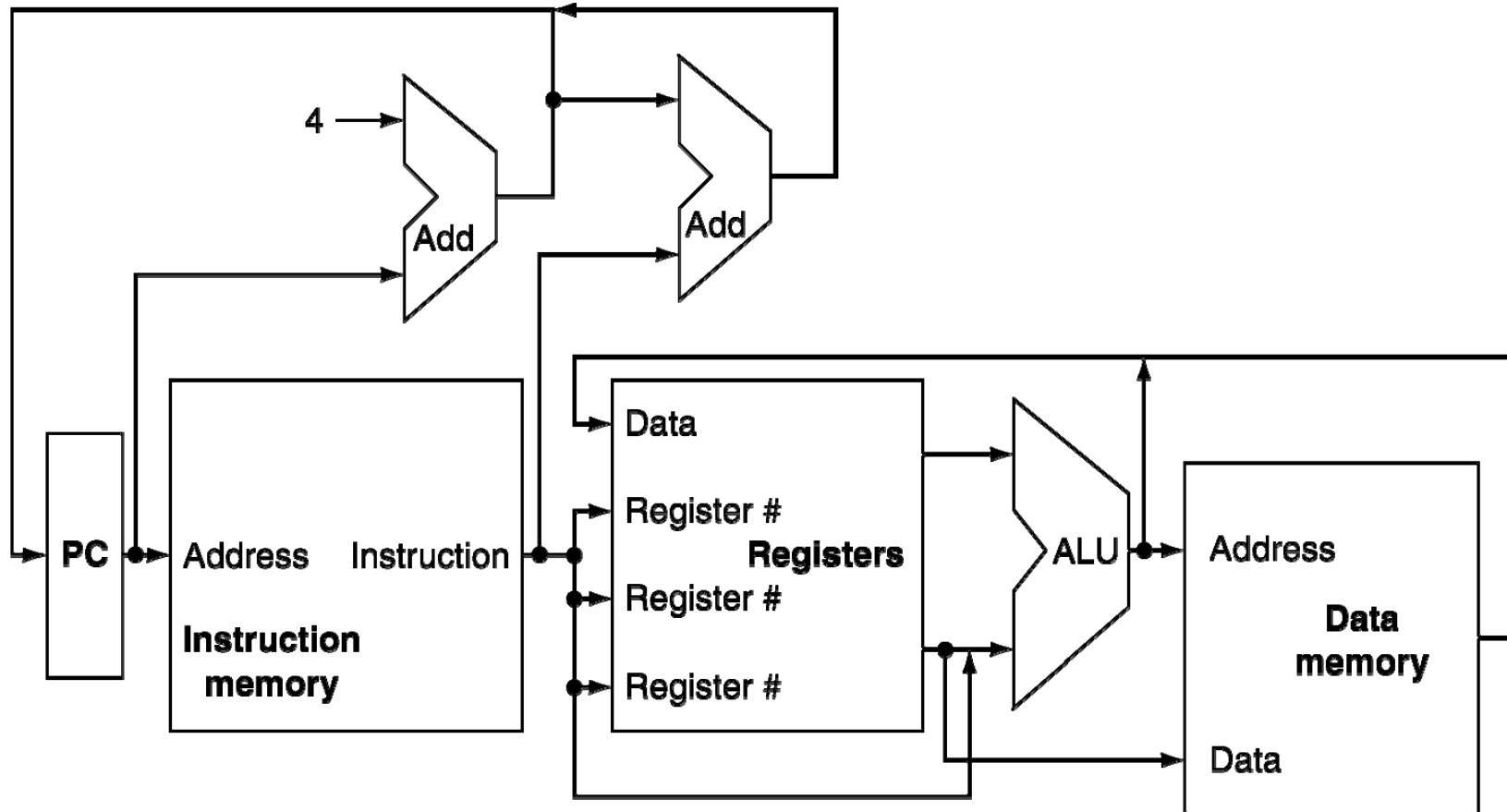


Instruction Execution

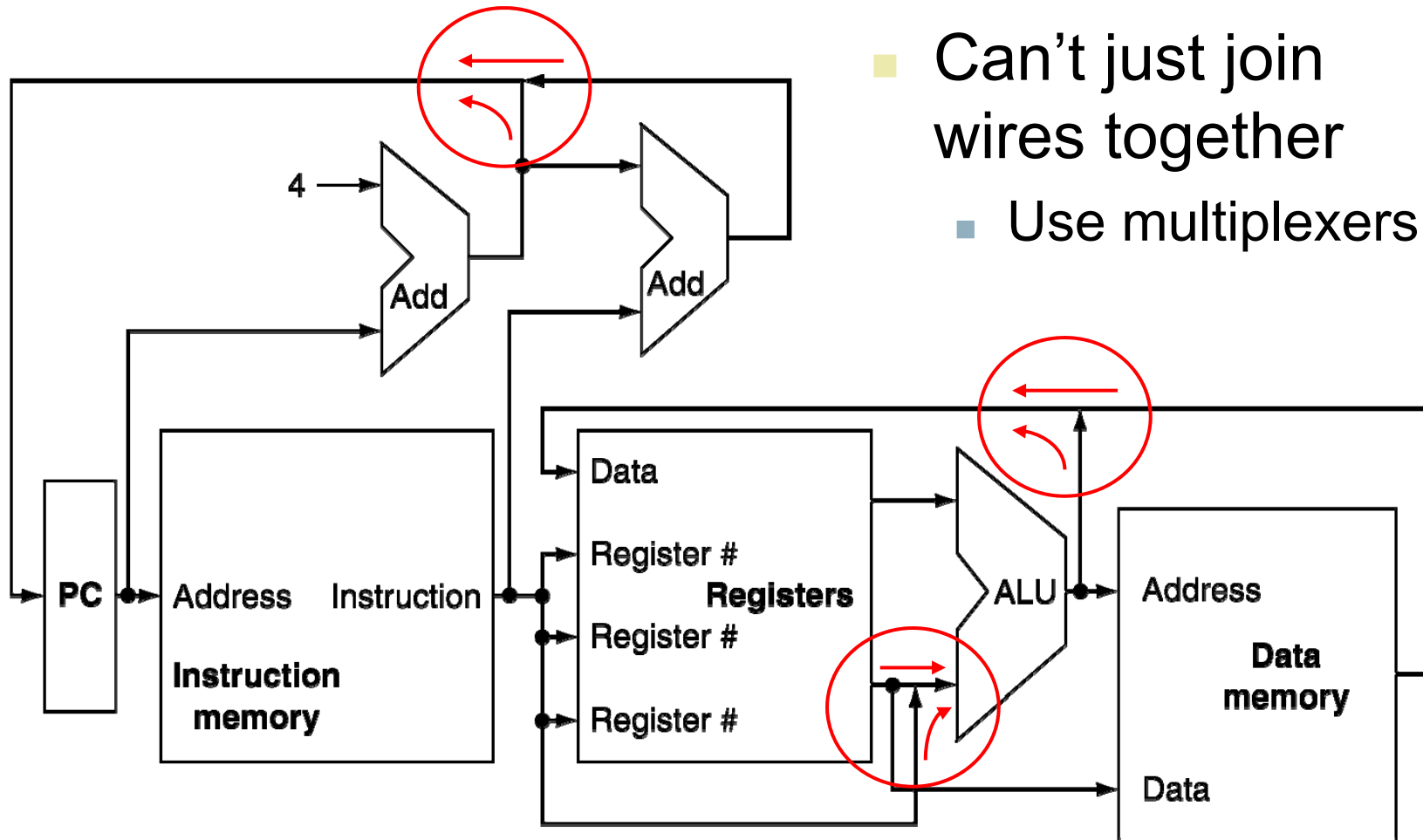
- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
 - Use ALU to calculate
 - Arithmetic result
 - Memory address for load/store
 - Branch target address
 - Access data memory for load/store
 - $PC \leftarrow \text{target address or } PC + 4$



CPU Overview



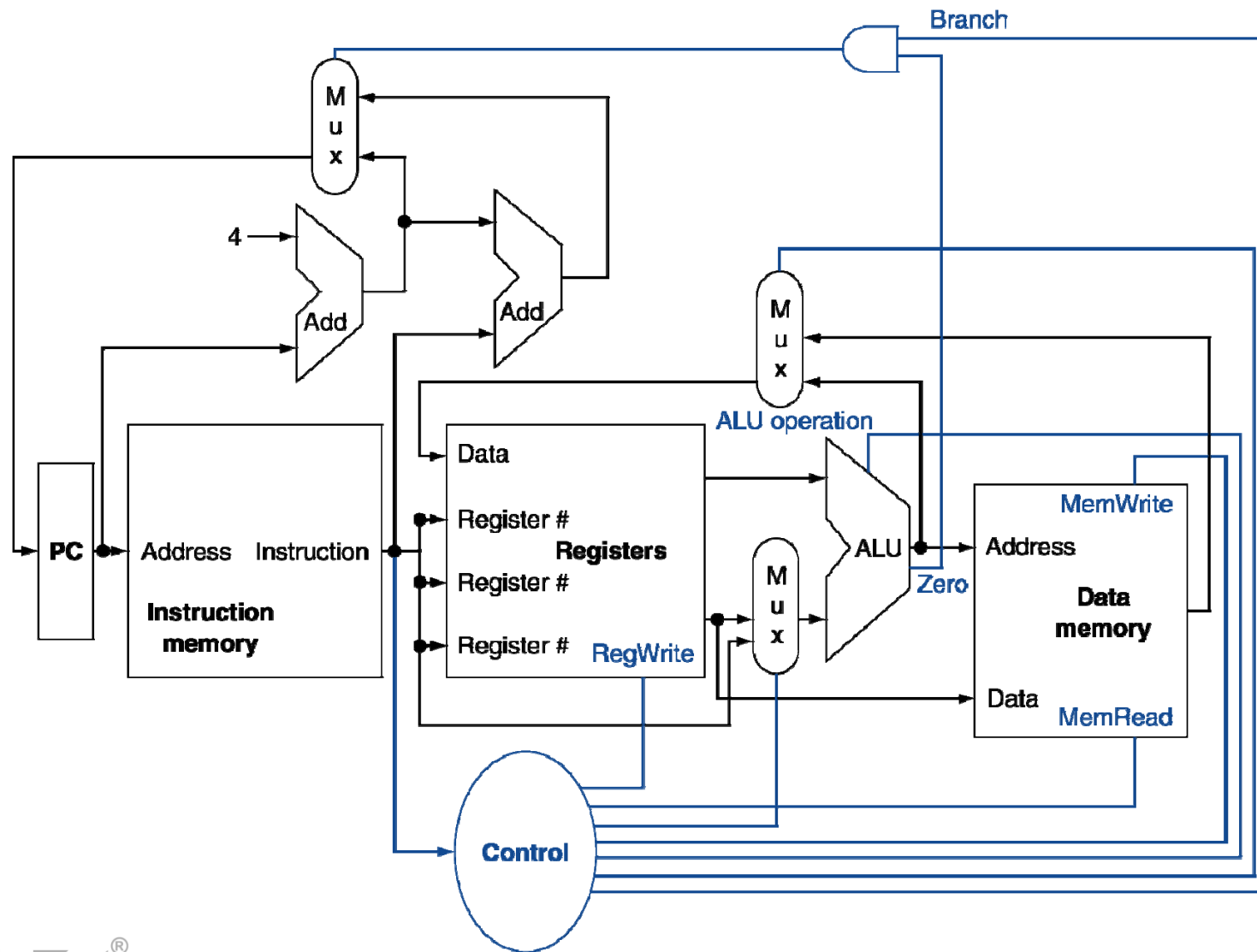
Multiplexers



- Can't just join wires together
 - Use multiplexers



Control



Logic Design Basics

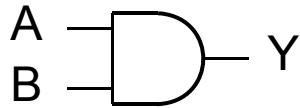
- Information encoded in binary
 - Low voltage = 0, High voltage = 1
 - One wire per bit
 - Multi-bit data encoded on multi-wire buses
- Combinational element
 - Operate on data
 - Output is a function of input
- State (sequential) elements
 - Store information



Combinational Elements

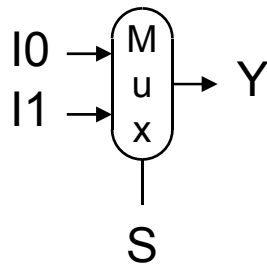
- AND-gate

- $Y = A \& B$



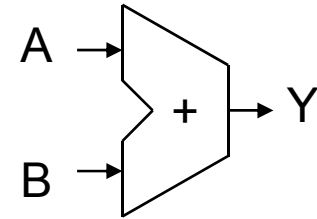
- Multiplexer

- $Y = S ? I1 : I0$



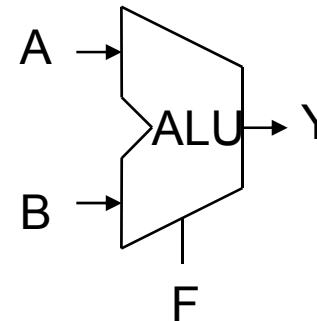
- Adder

- $Y = A + B$



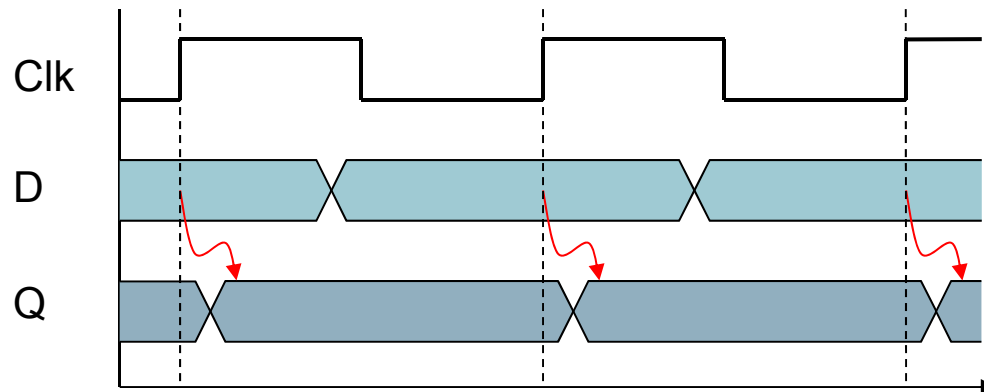
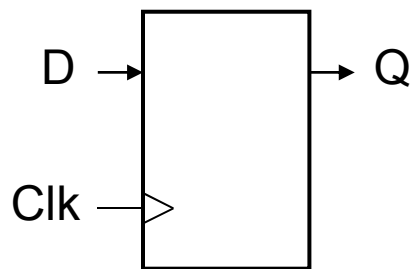
- Arithmetic/Logic Unit

- $Y = F(A, B)$



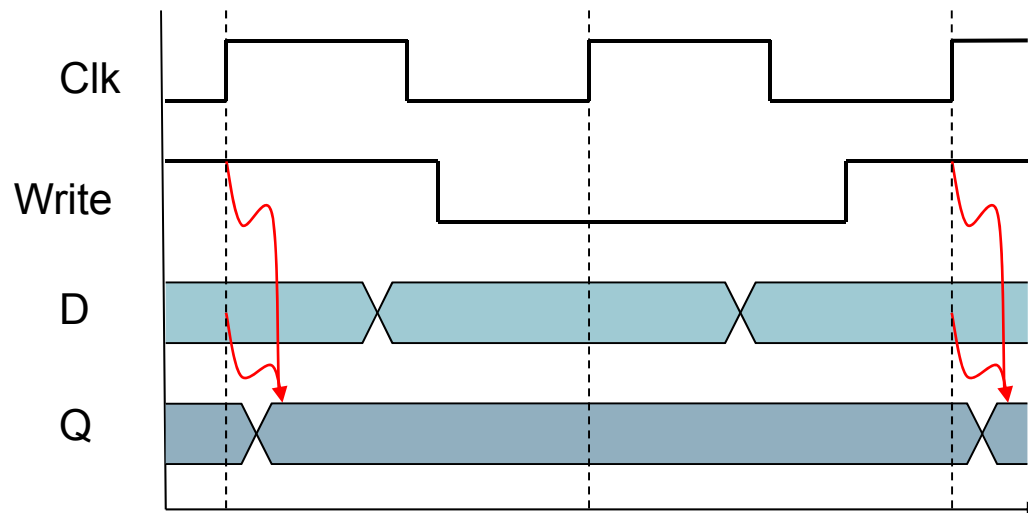
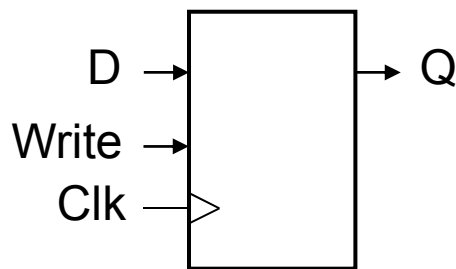
Sequential Elements

- Register: stores data in a circuit
 - Uses a clock signal to determine when to update the stored value
 - Edge-triggered: update when Clk changes from 0 to 1



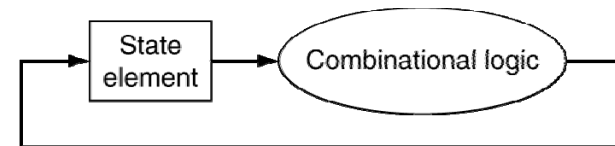
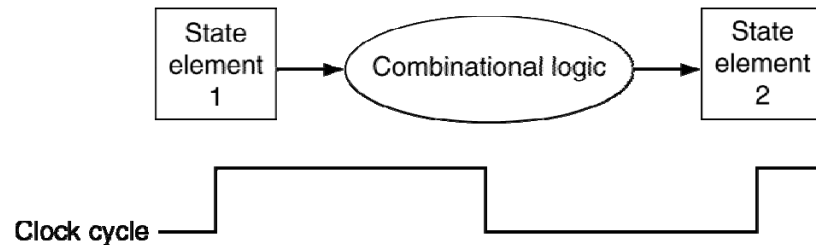
Sequential Elements

- Register with write control
 - Only updates on clock edge when write control input is 1
 - Used when stored value is required later



Clocking Methodology

- Combinational logic transforms data during clock cycles
 - Between clock edges
 - Input from state elements, output to state element
 - Longest delay determines clock period



Chapter 4

The Processor

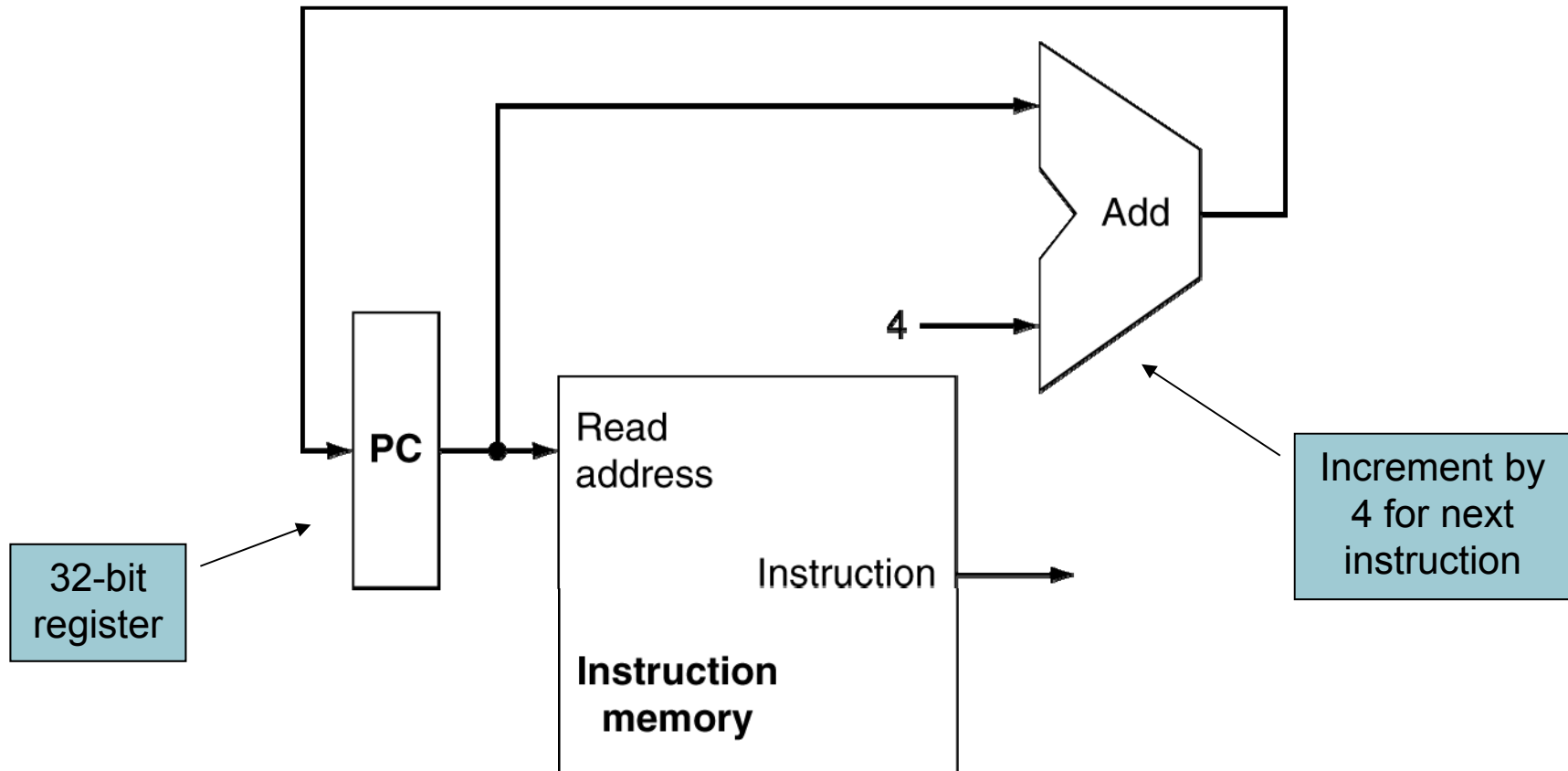


Building a Datapath

- Datapath
 - Elements that process data and addresses in the CPU
 - Registers, ALUs, mux's, memories, ...
- We will build a MIPS datapath incrementally
 - Refining the overview design

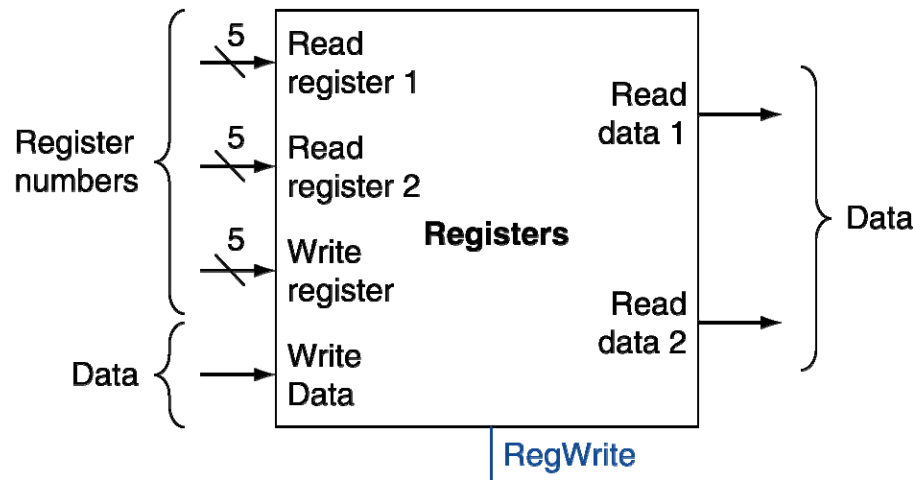


Instruction Fetch

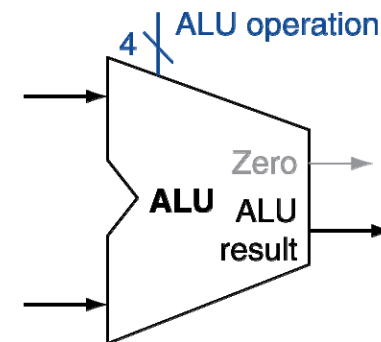


R-Format Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result



a. Registers

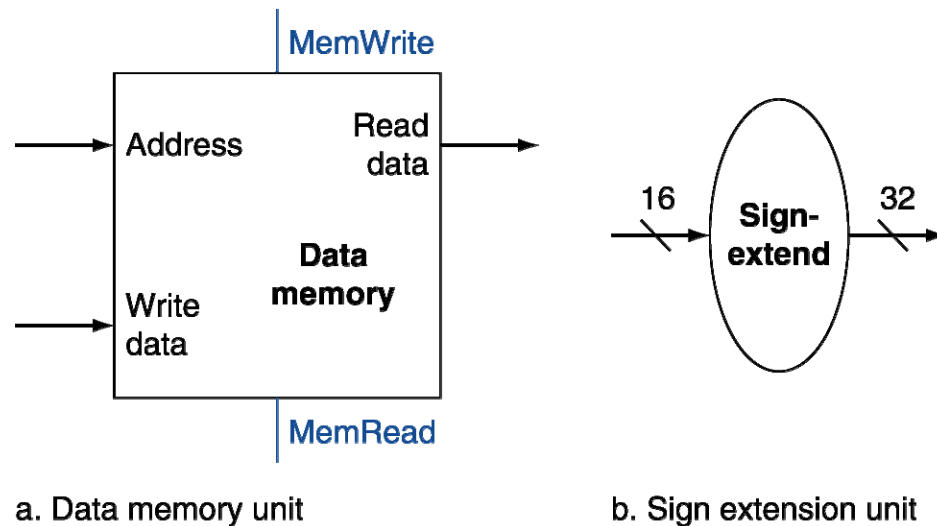


b. ALU



Load/Store Instructions

- Read register operands
- Calculate address using 16-bit offset
 - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory

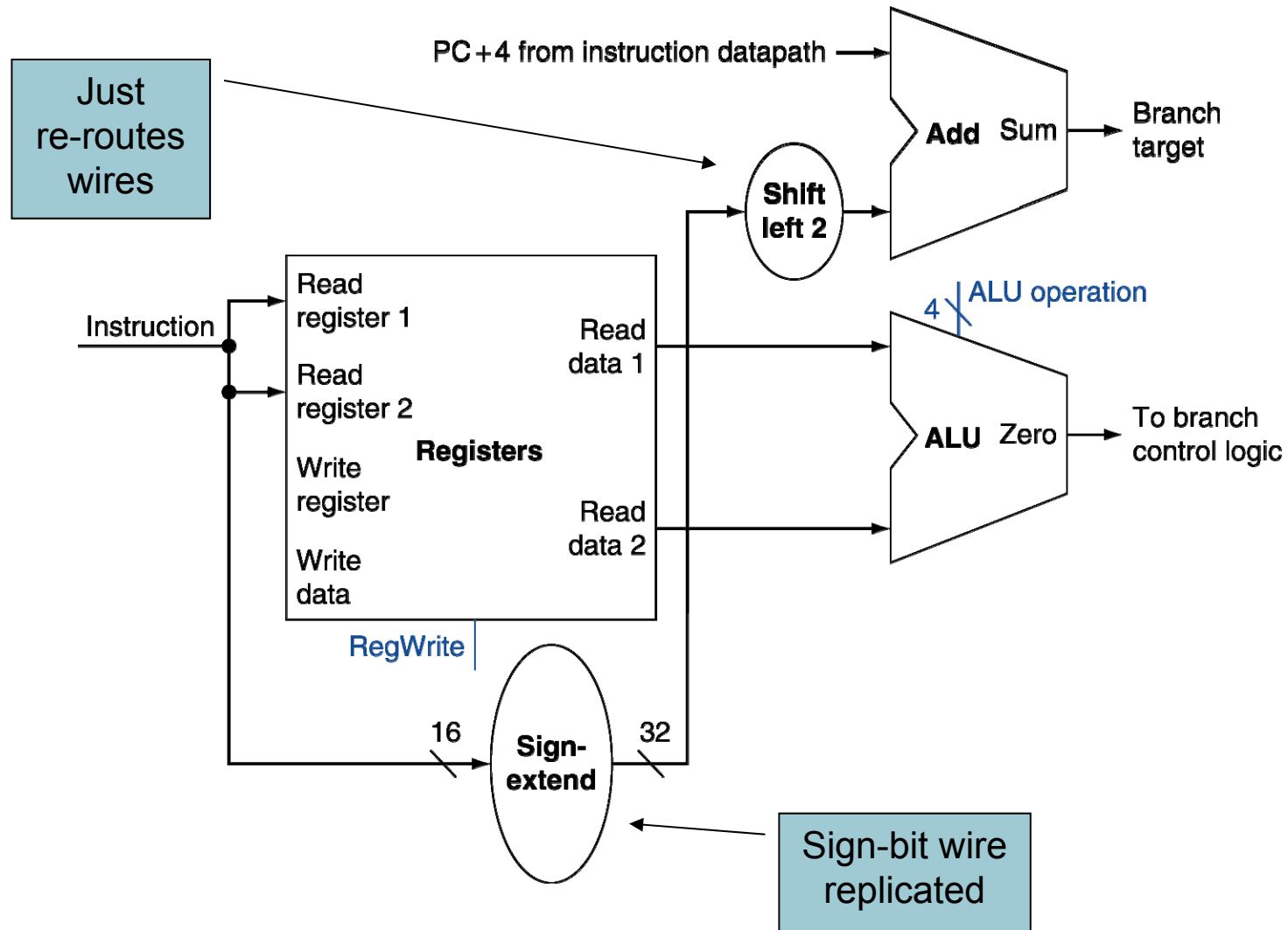


Branch Instructions

- Read register operands
- Compare operands
 - Use ALU, subtract and check Zero output
- Calculate target address
 - Sign-extend displacement
 - Shift left 2 places (word displacement)
 - Add to PC + 4
 - Already calculated by instruction fetch



Branch Instructions

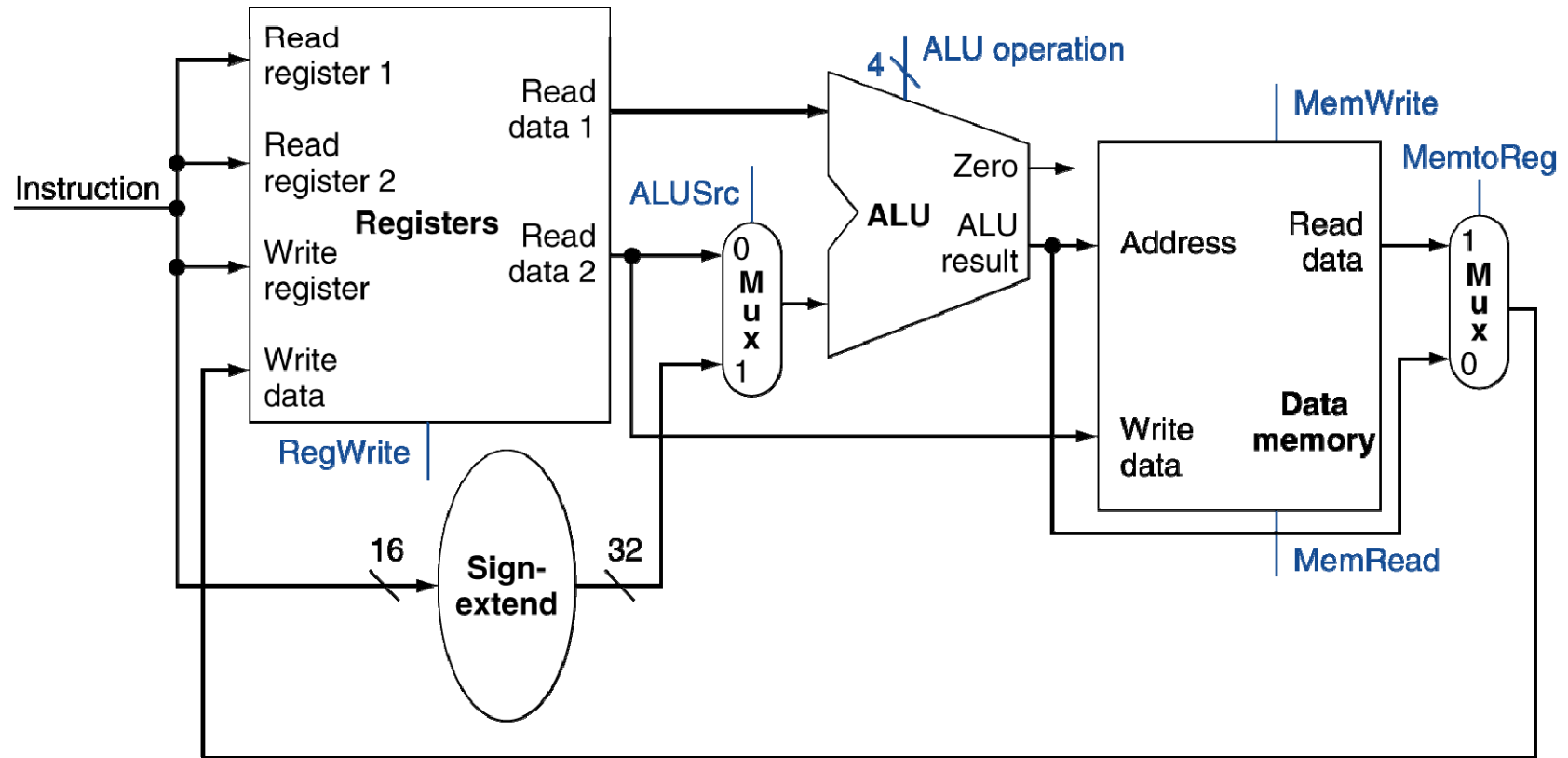


Composing the Elements

- First-cut data path does an instruction in one clock cycle
 - Each datapath element can only do one function at a time
 - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions



R-Type/Load/Store Datapath

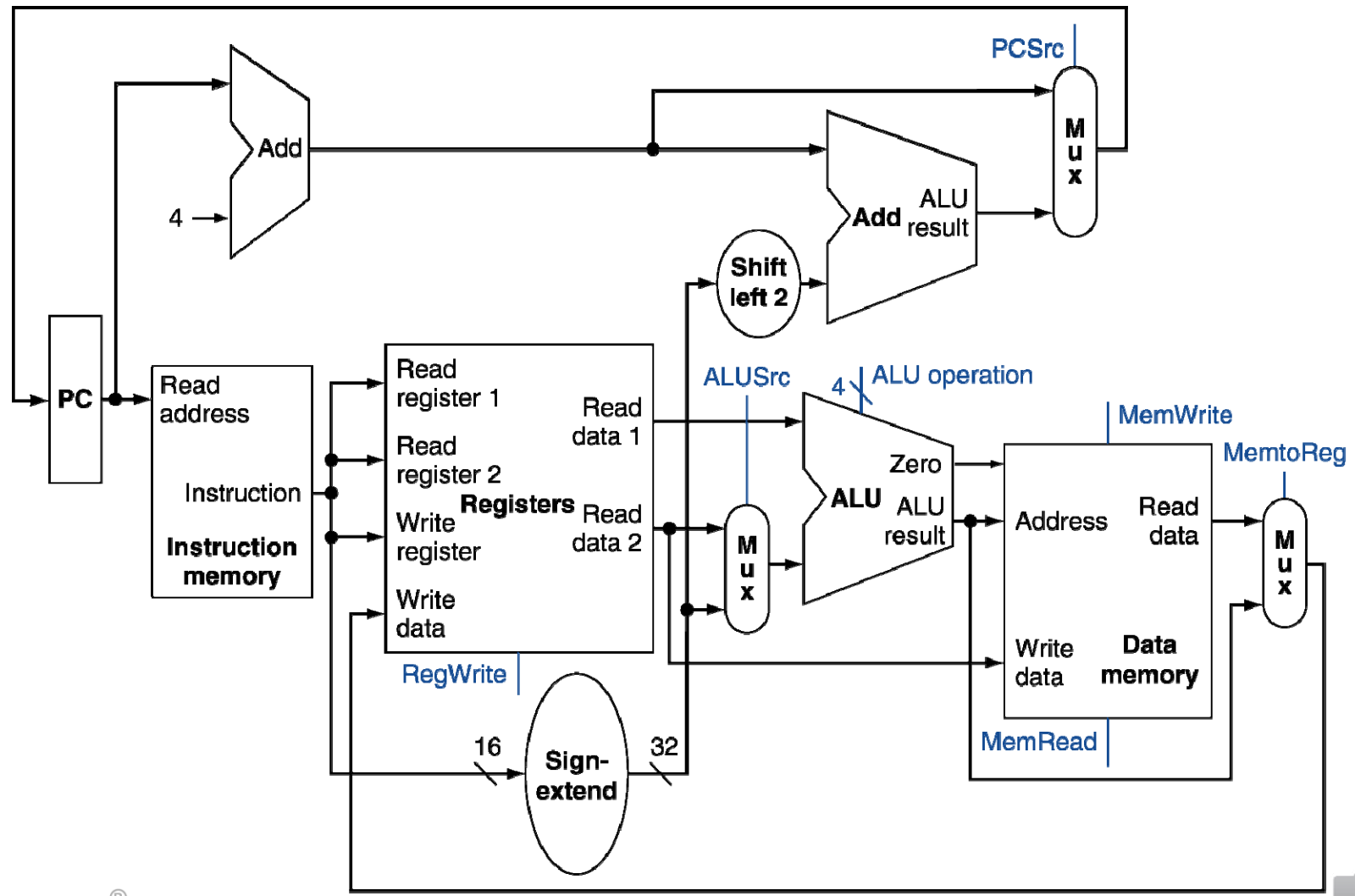


Chapter 4

The Processor



Full Datapath



ALU Control

- ALU used for
 - Load/Store: F = add
 - Branch: F = subtract
 - R-type: F depends on funct field

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



ALU Control

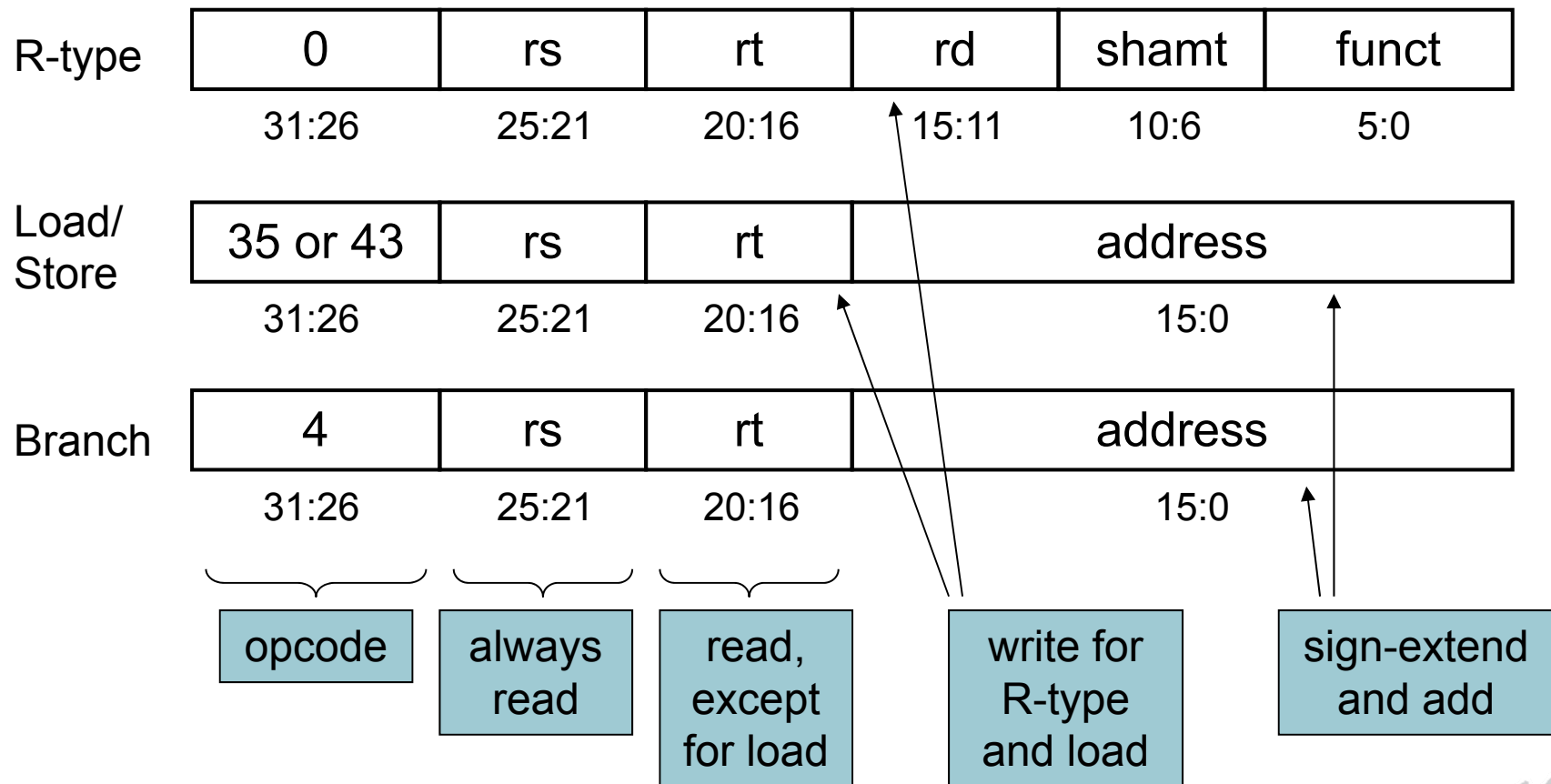
- Assume 2-bit ALUOp derived from opcode
 - Combinational logic derives ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

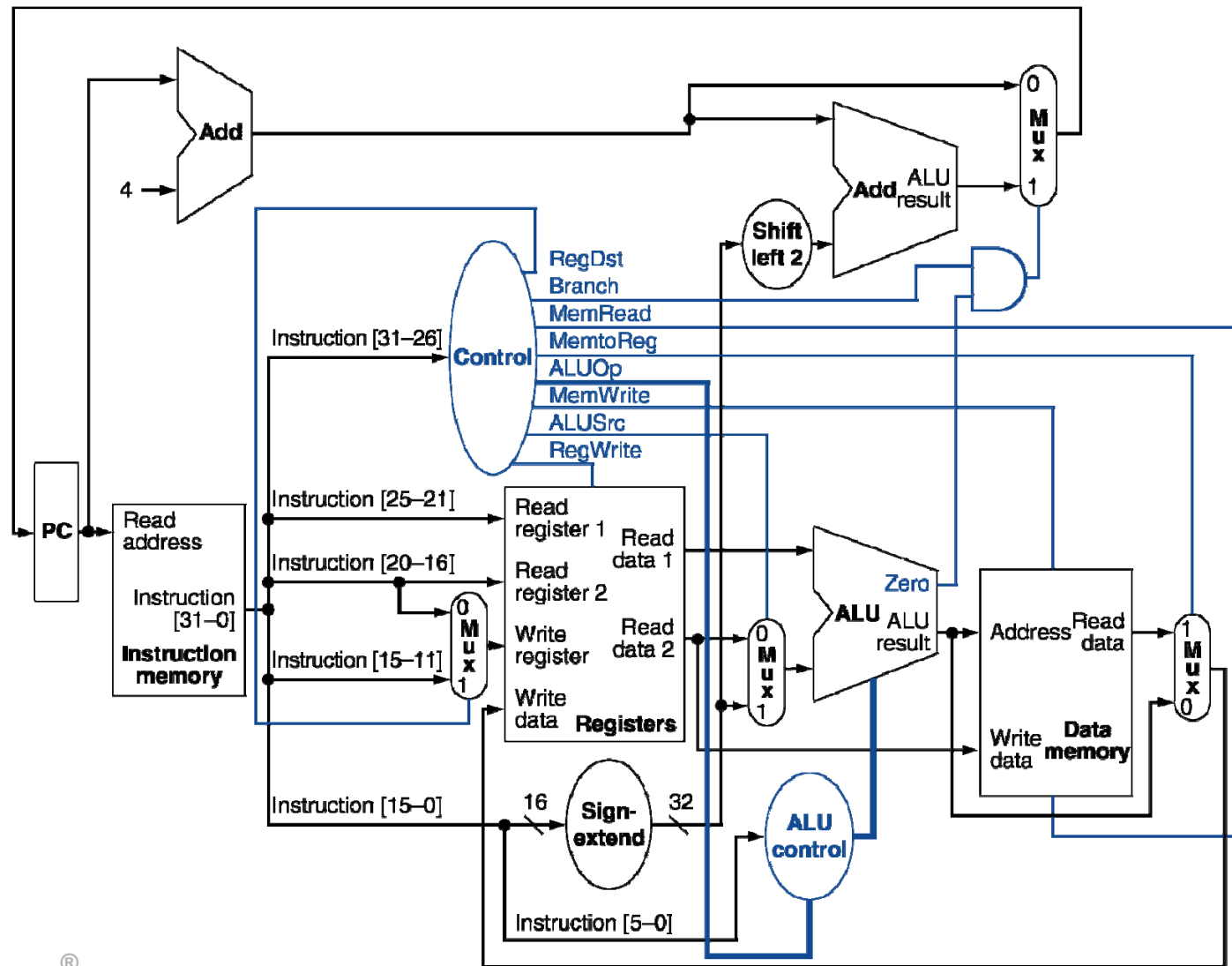


The Main Control Unit

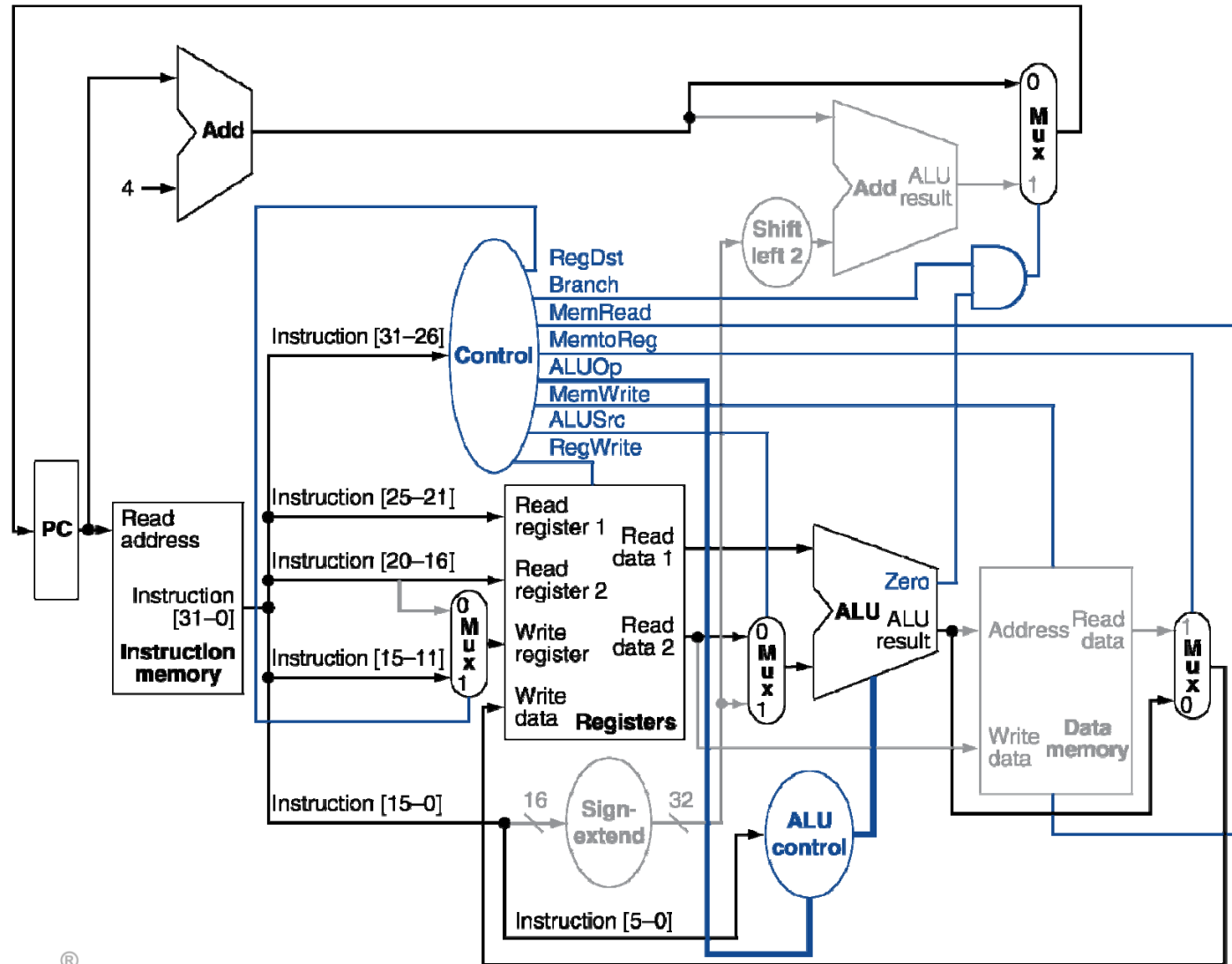
- Control signals derived from instruction



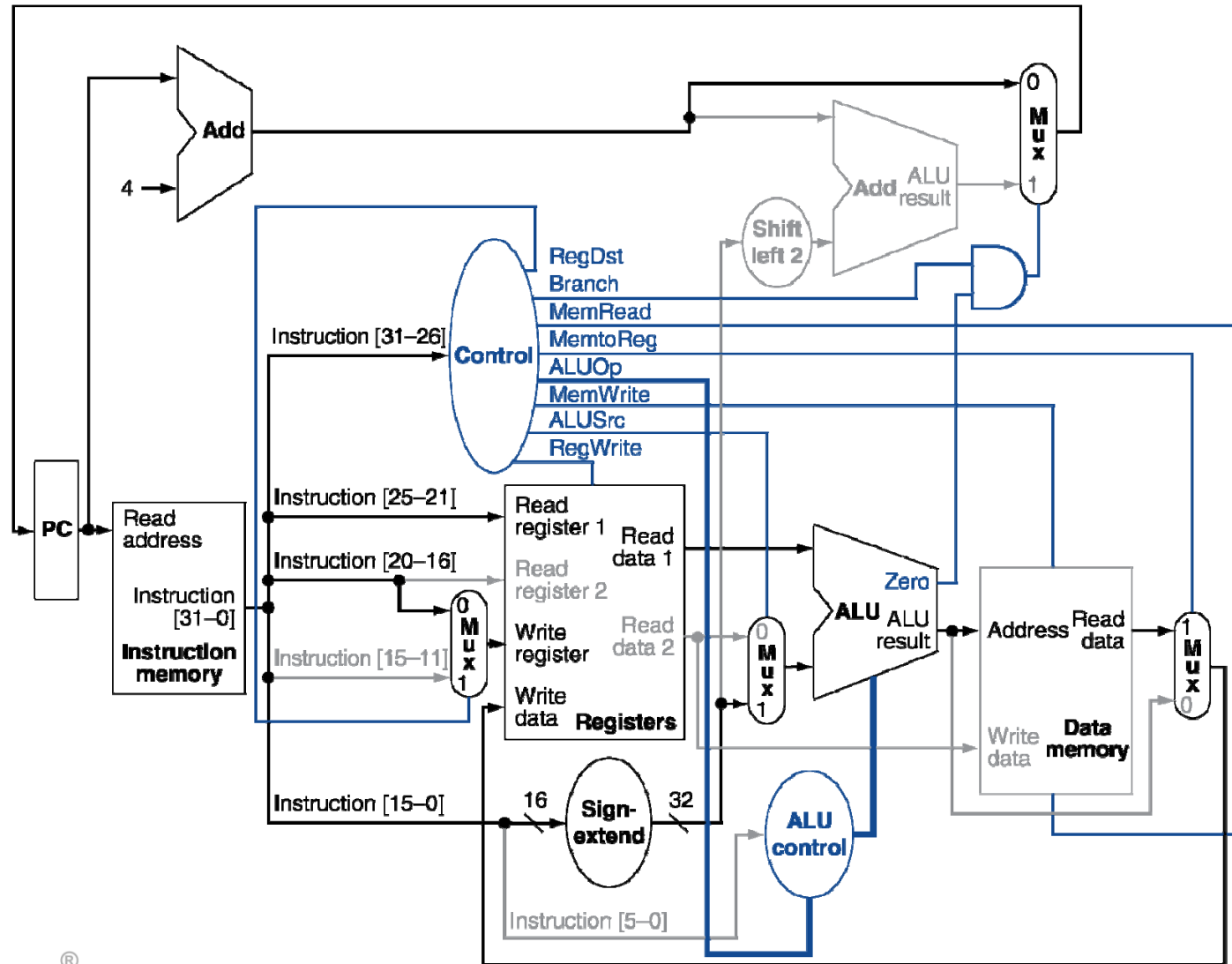
Datapath With Control



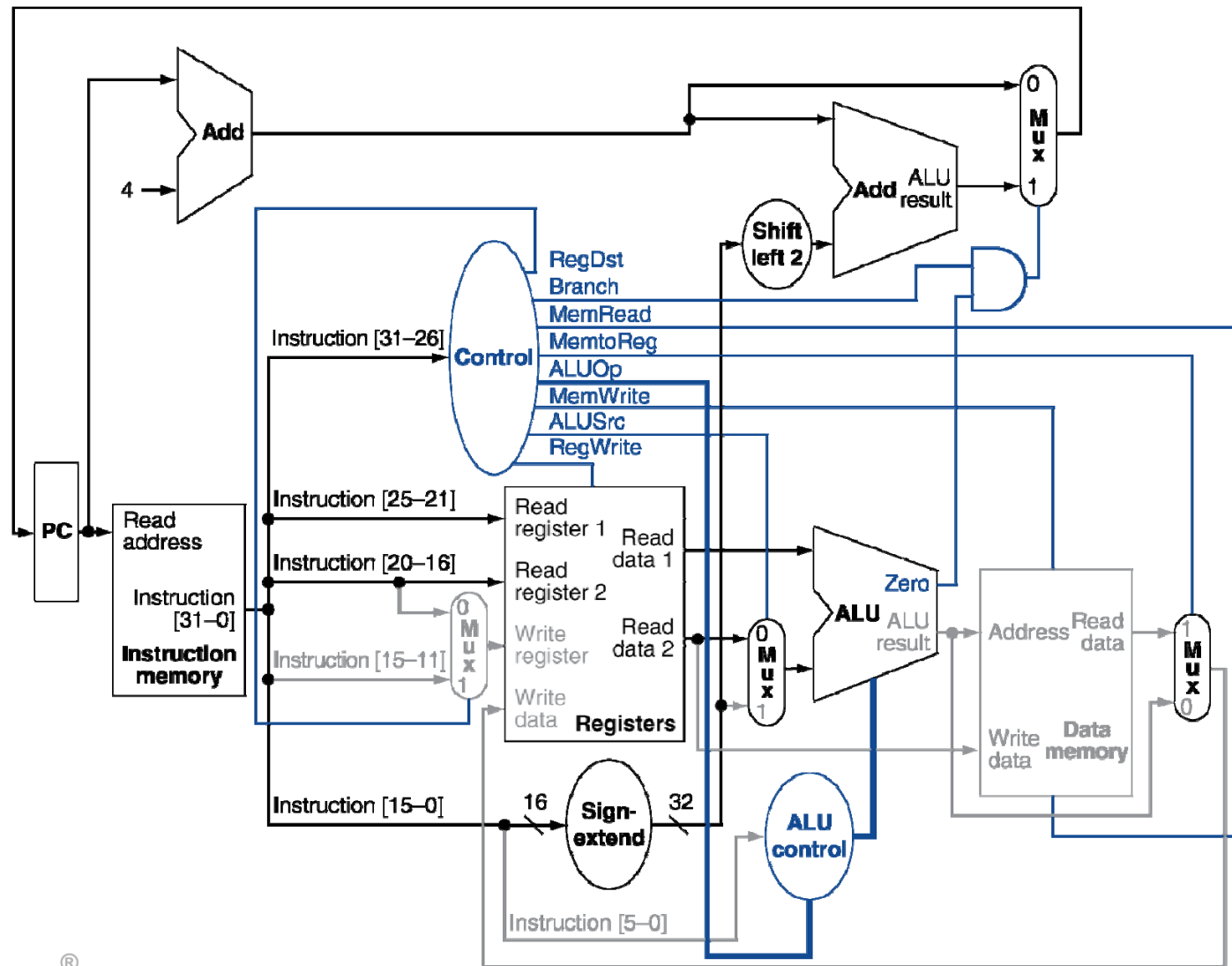
R-Type Instruction



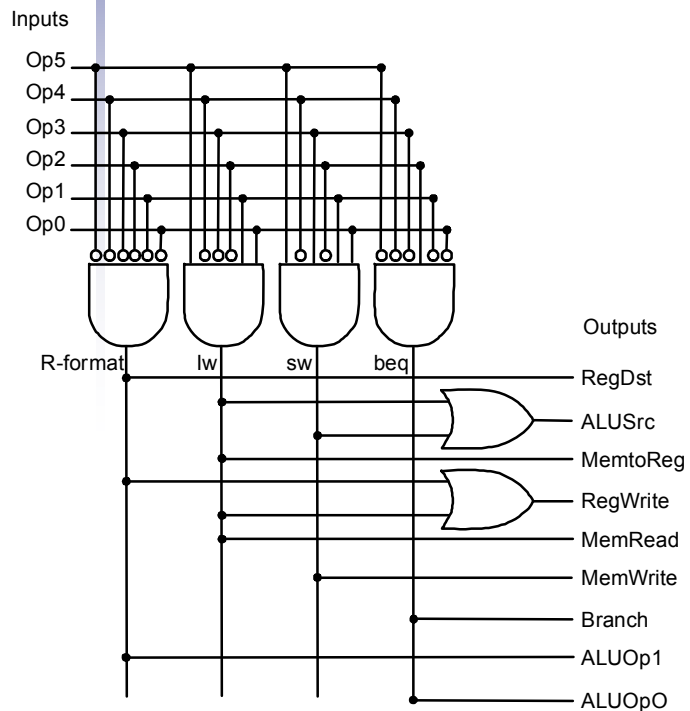
Load Instruction



Branch-on-Equal Instruction



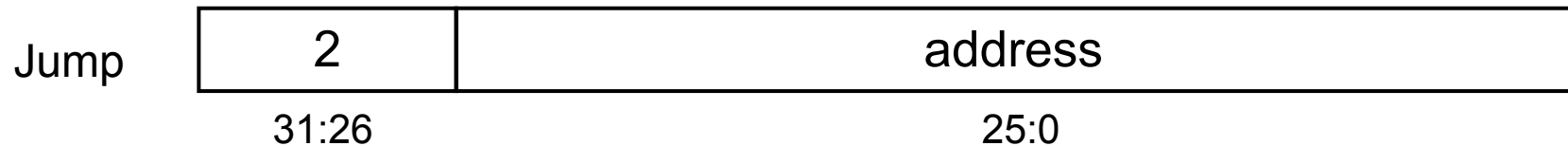
Controller



		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
	ALUOp2	0	0	0	1



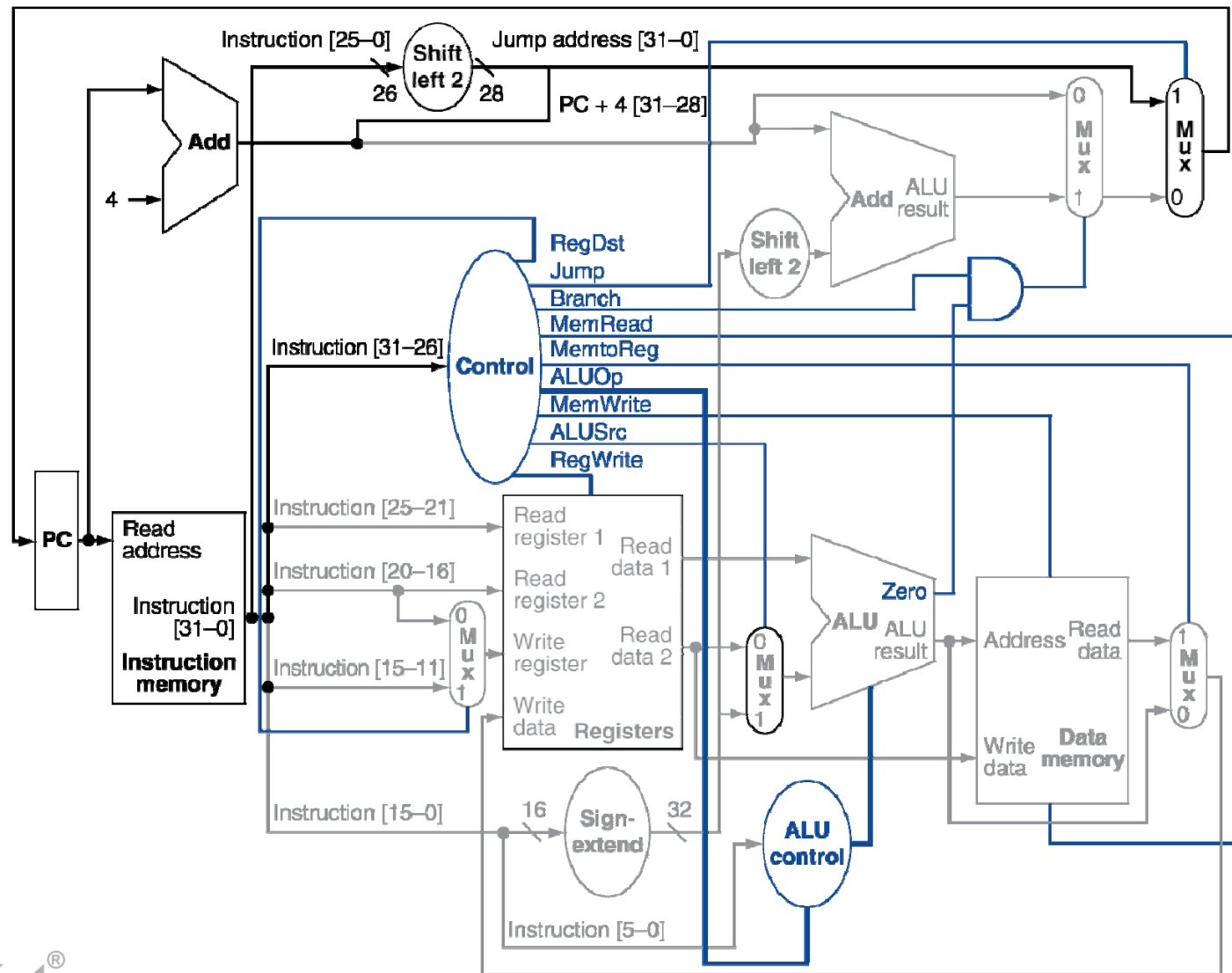
Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
 - Top 4 bits of old PC
 - 26-bit jump address
 - 00
- Need an extra control signal decoded from opcode



Datapath With Jumps Added



Up Next

- Pipelined Implementation
- Why isn't single cycle enough?
 - control is relatively simple
 - CPI is 1, but cycle time must be long enough for every instruction to complete!
 - branch instruction versus load instruction
 - loads require instruction fetch, register access, ALU, memory access, register access
 - branches require instruction fetch, register access, ALU
 - and this is for a simplified processor!
 - no floating point ops, no multiply or divide

