# Chapter 4

The Processor

Introduction

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware

- We will examine two MIPS implementations
  - A simplified version
  - A more realistic 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 $\rightarrow$ instruction memory, fetch instruction

- Register numbers $\rightarrow$ 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

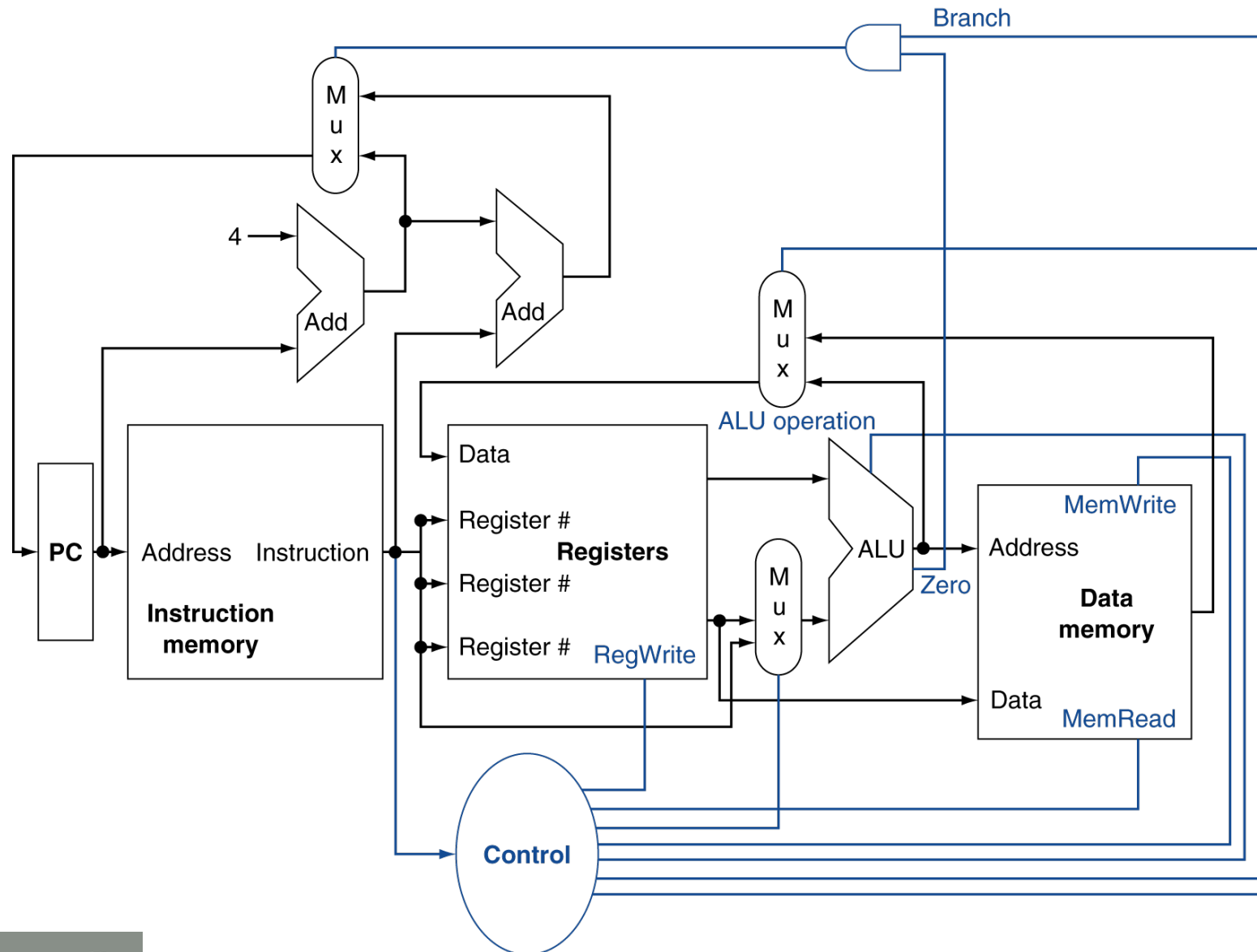 - PC $\leftarrow$ target address or PC + 4

CPU Overview
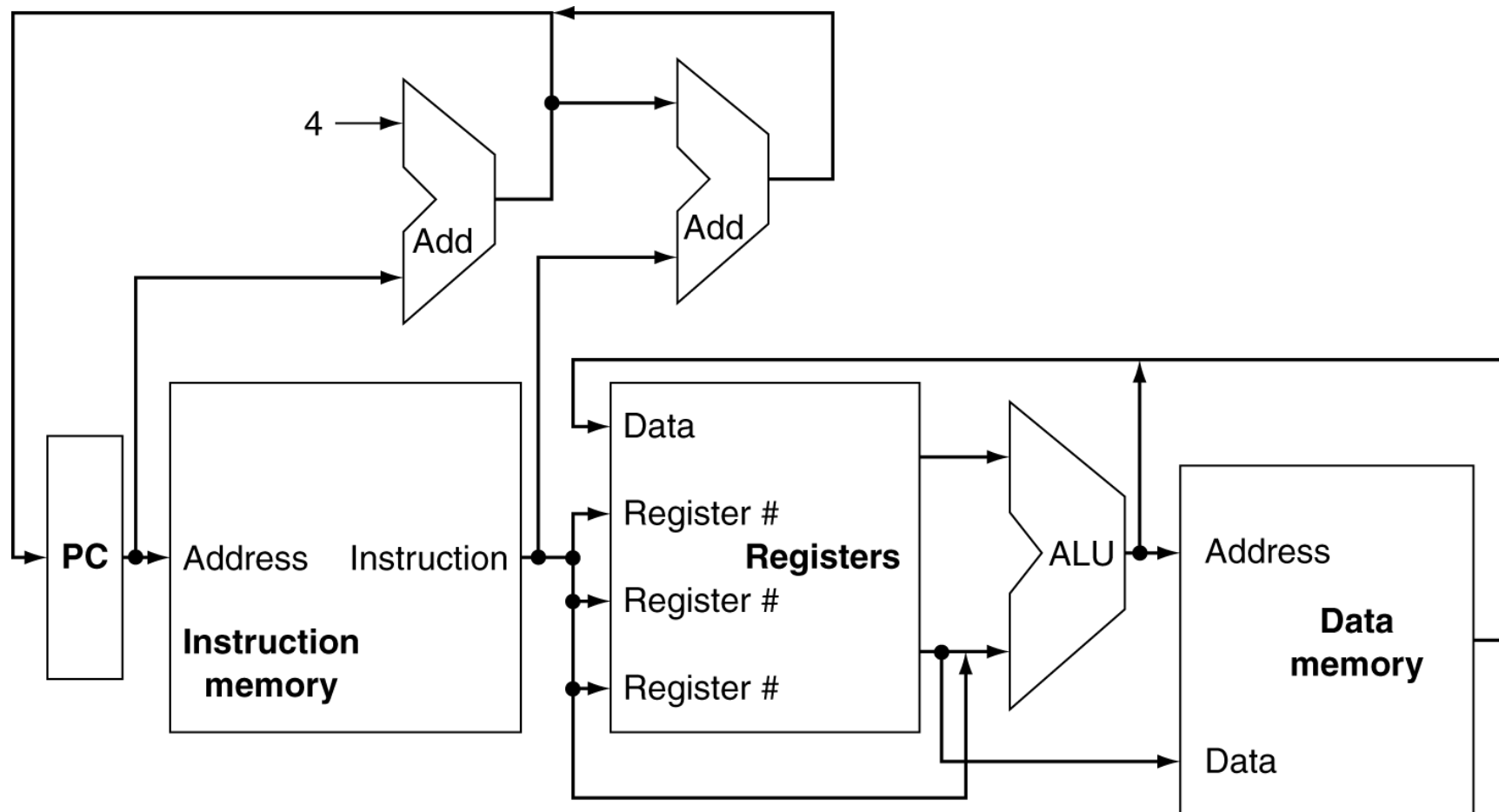
Multiplexers
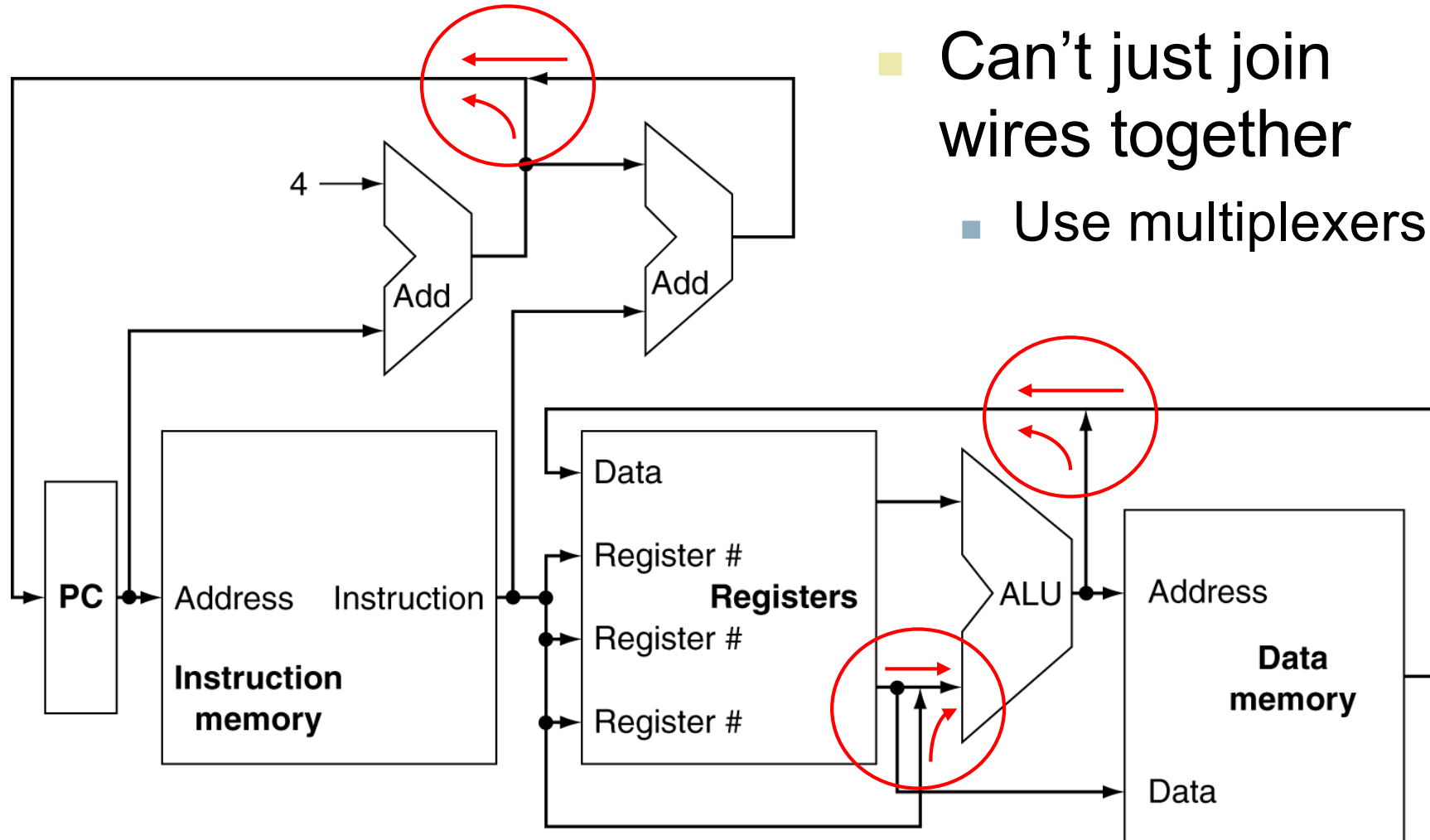
- Can't just join wires together
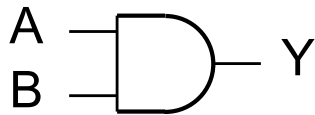 - Use multiplexers

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
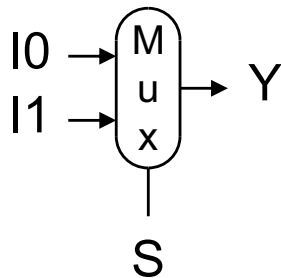 - Store information

Combinational Elements
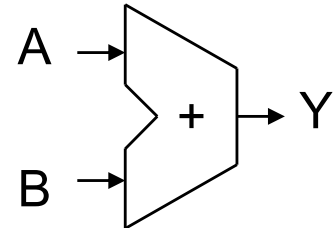
- ## AND-gate
  - Y = A & B
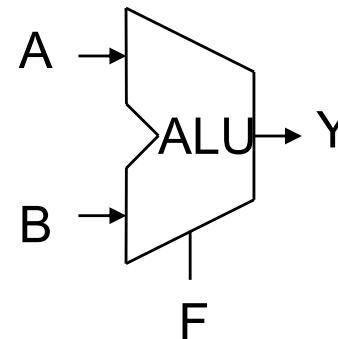
  A —
  B —
  ⟩— Y

- ## Multiplexer
  - Y = S ? I1 : I0

  I0 →
  I1 →
  Mux
  → Y

  S

- ## Adder
  - Y = A + B

  A →
  B →
  + → Y

- ## Arithmetic/Logic Unit
  - Y = F(A, B)

  A →
  B →
  ALU → Y
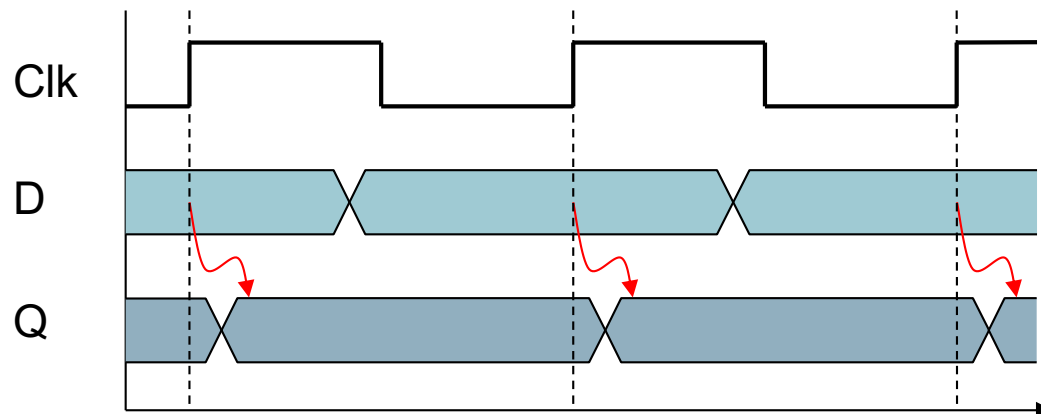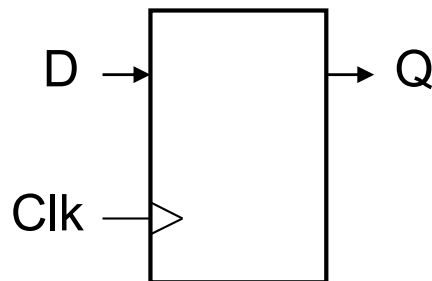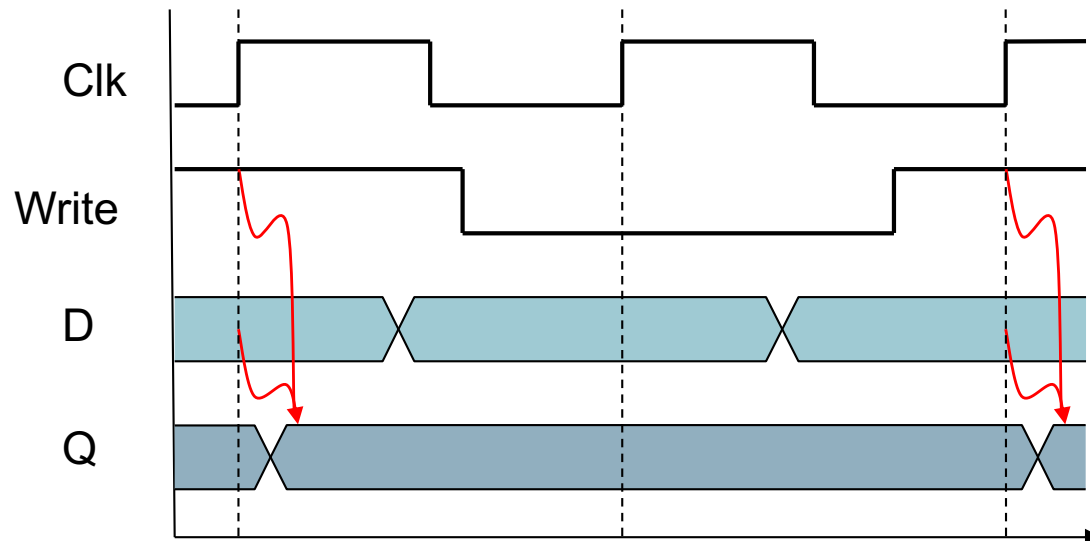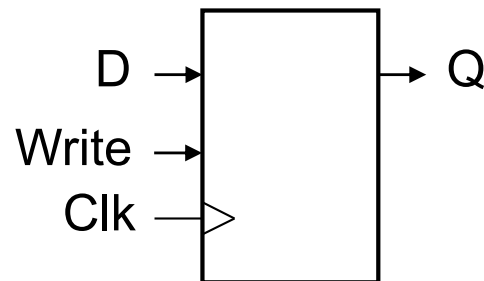
  F

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
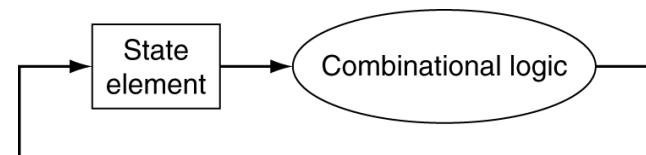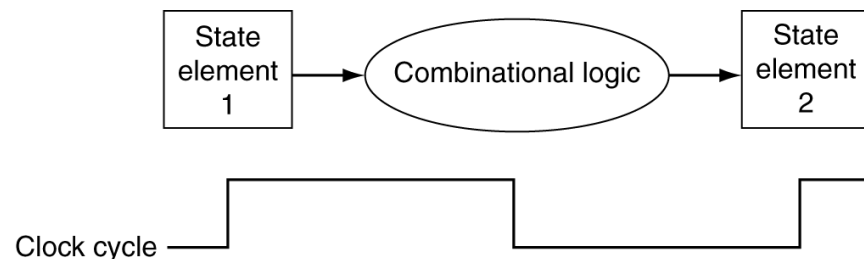
Sequential Elements

- 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

Building a Datapath

- ## Datapath

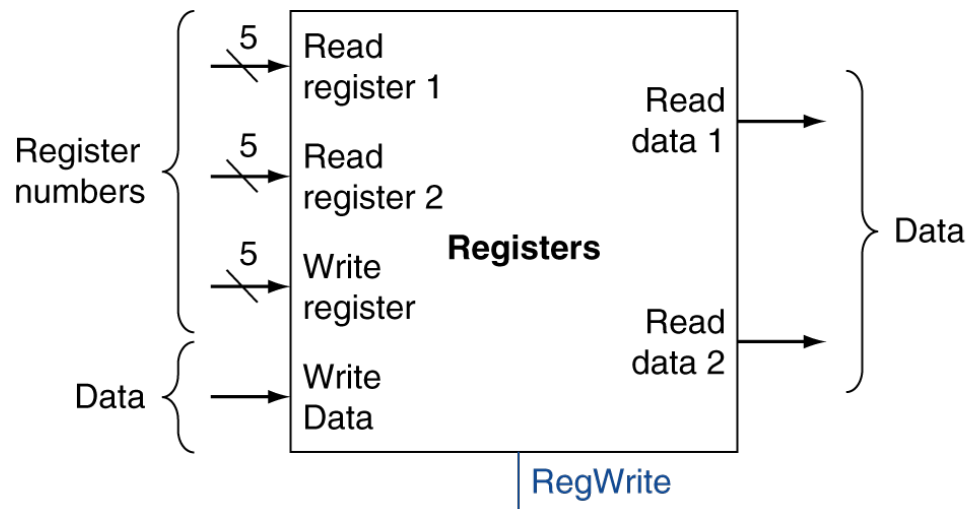 - ### Elements that process data and addresses in the CPU

    - Registers, ALUs, mux's, memories, …

- ## We will build a MIPS datapath incrementally

  - ### Refining the overview design

Instruction Fetch

32-bit register

Increment by 4 for next instruction
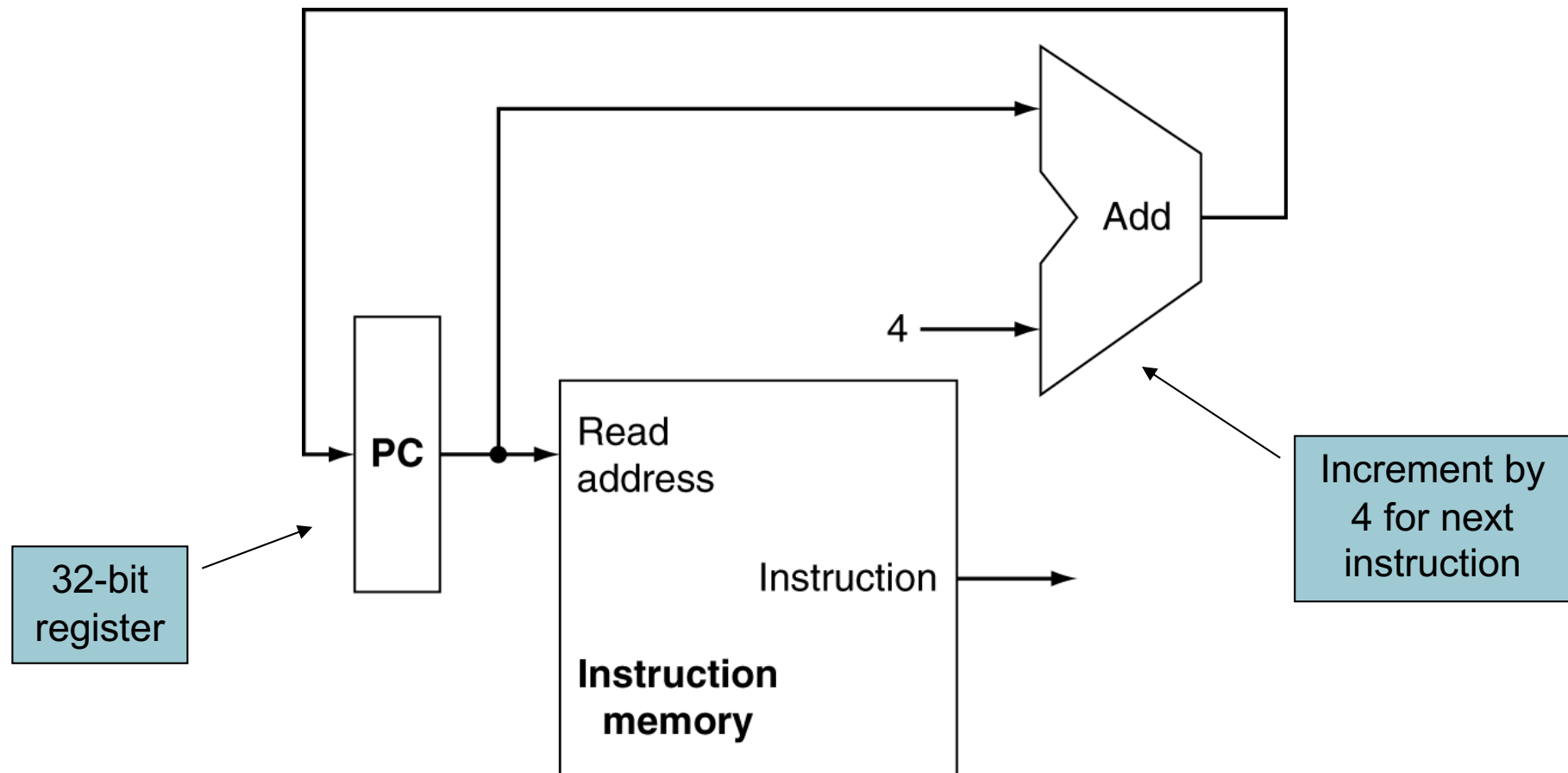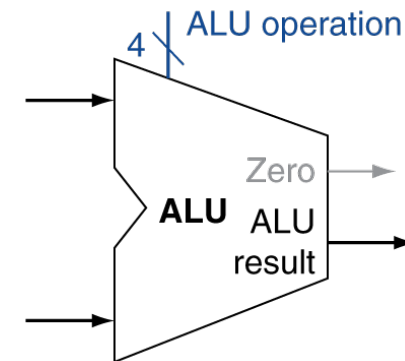
R-Format Instructions

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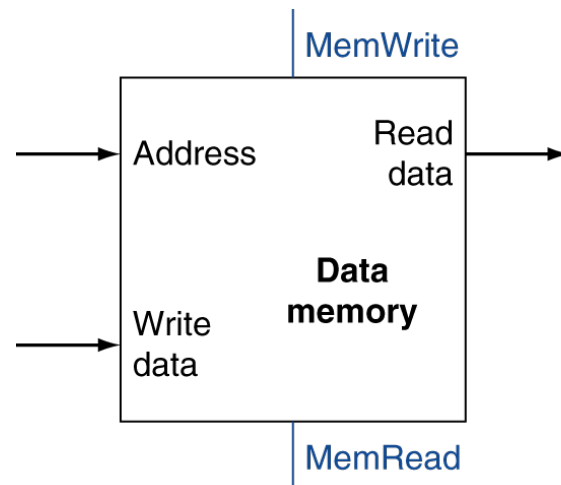- Write register result

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

a. Data memory unit
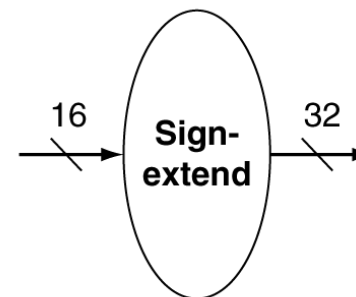
b. Sign extension unit

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

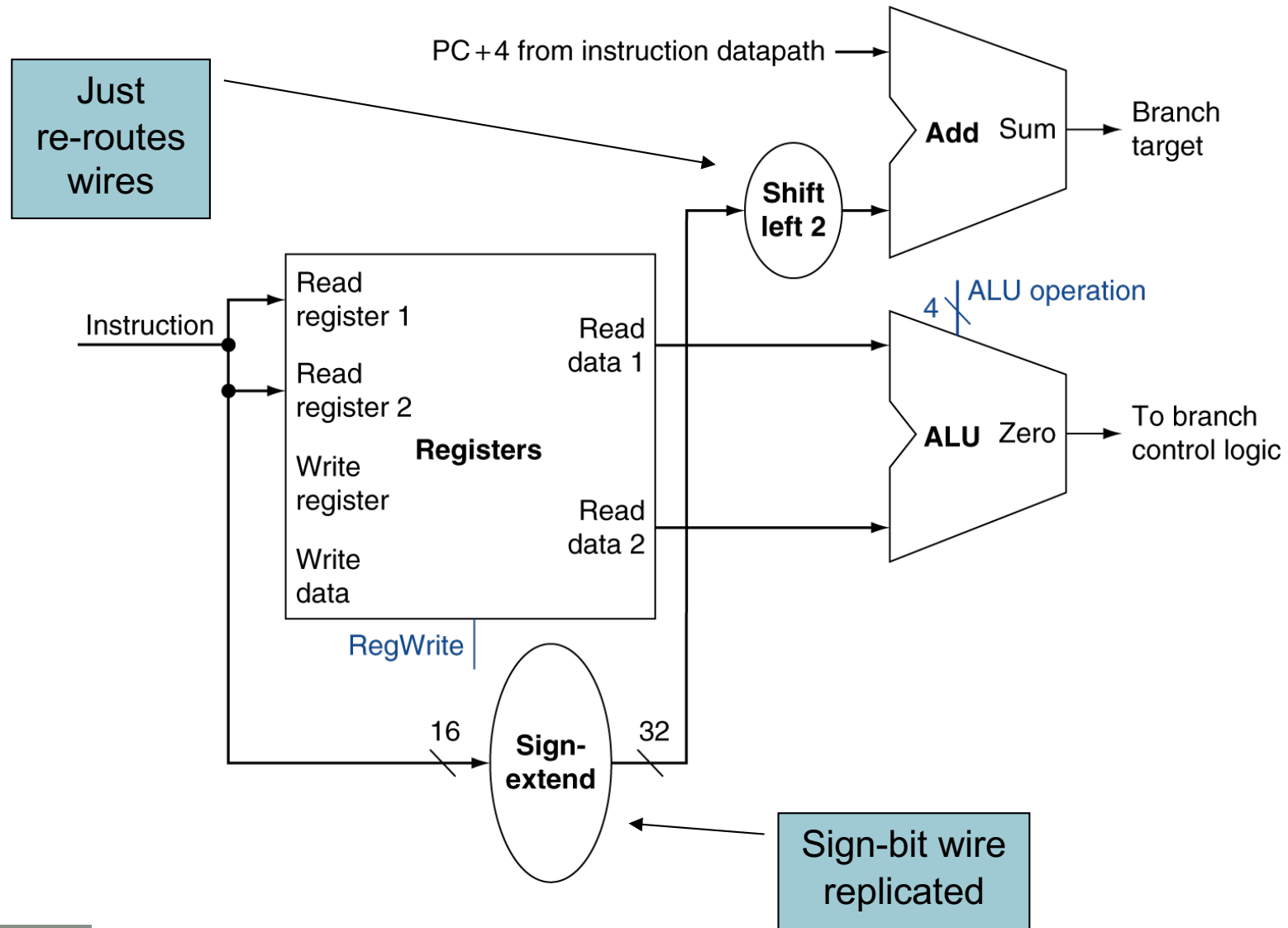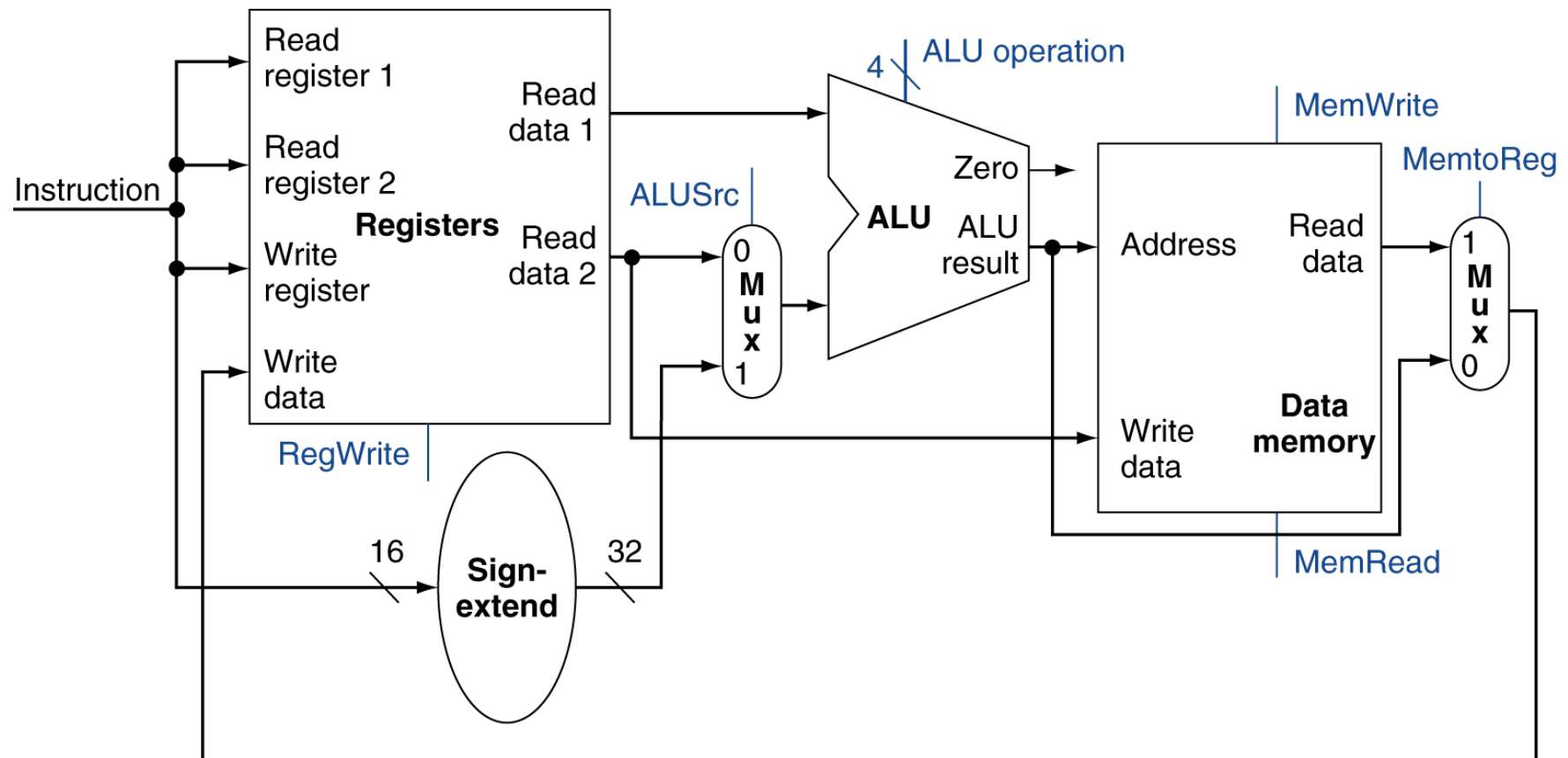- First-cut data path does an instruction in one clock cycle

  - 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

Full Datapath
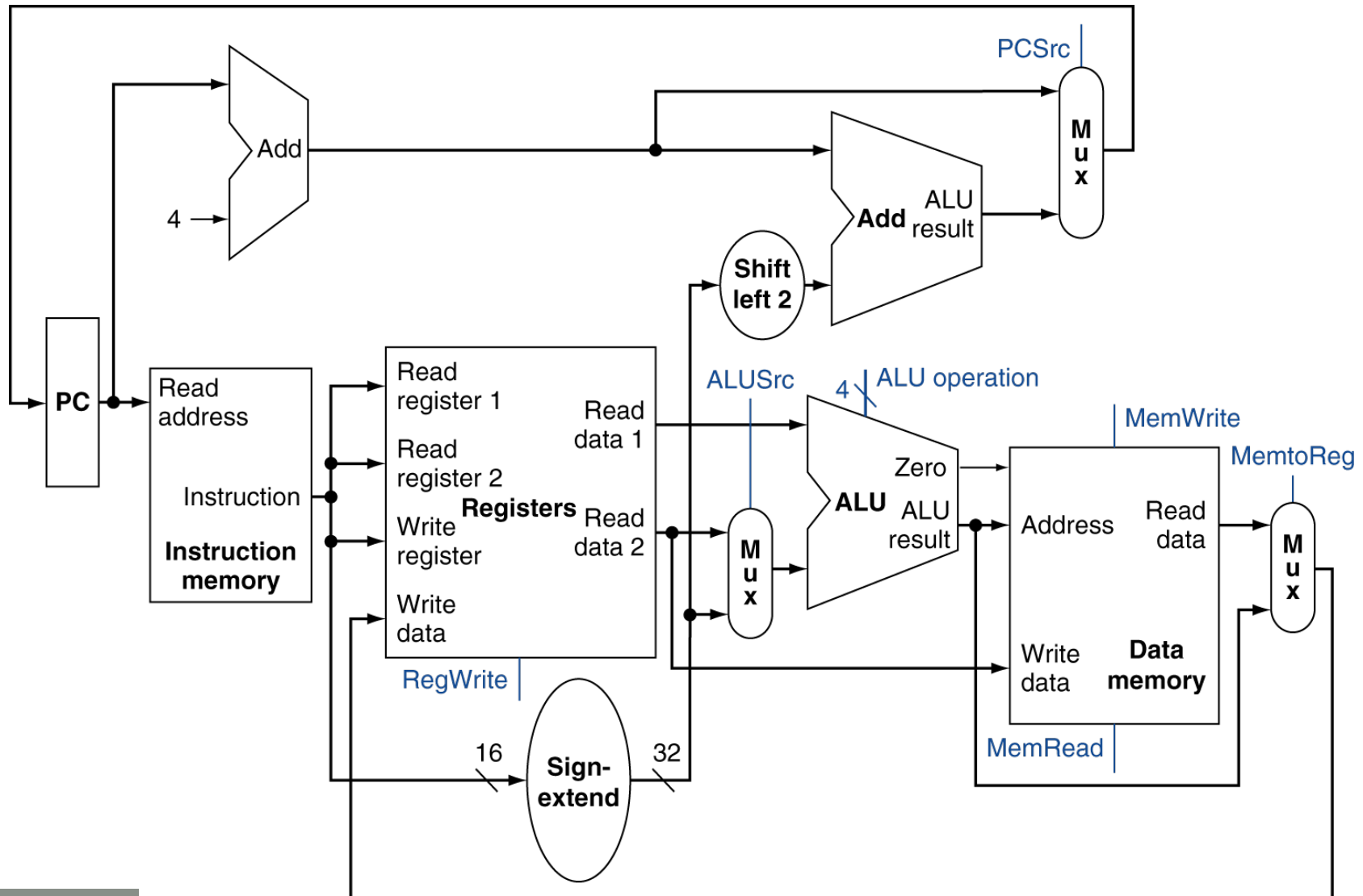
ALU Control

- ## 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
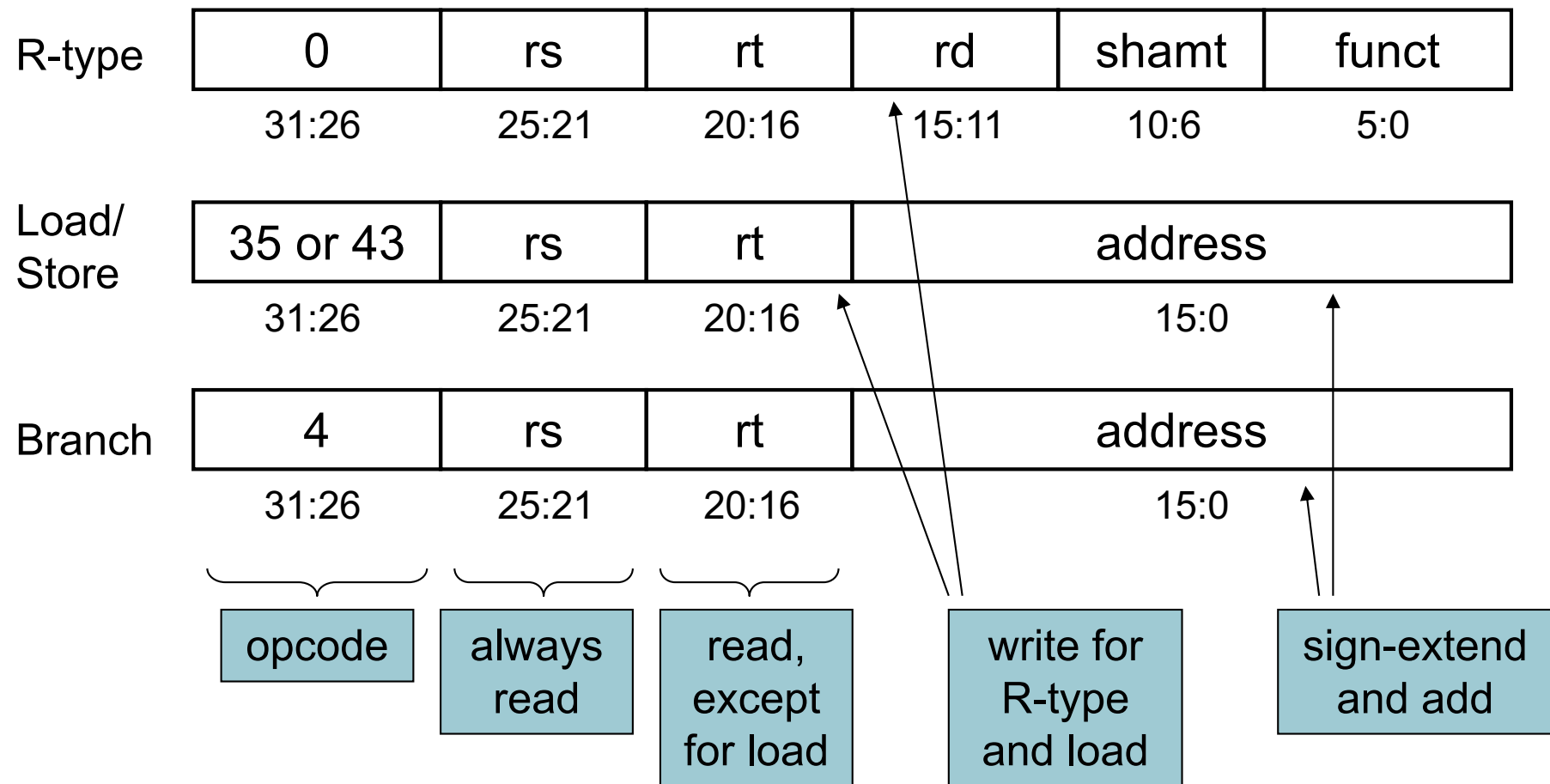 - Combinational logic derives ALU control

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

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

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

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

opcode   always read   read, except for load   write for R-type and load   sign-extend and add
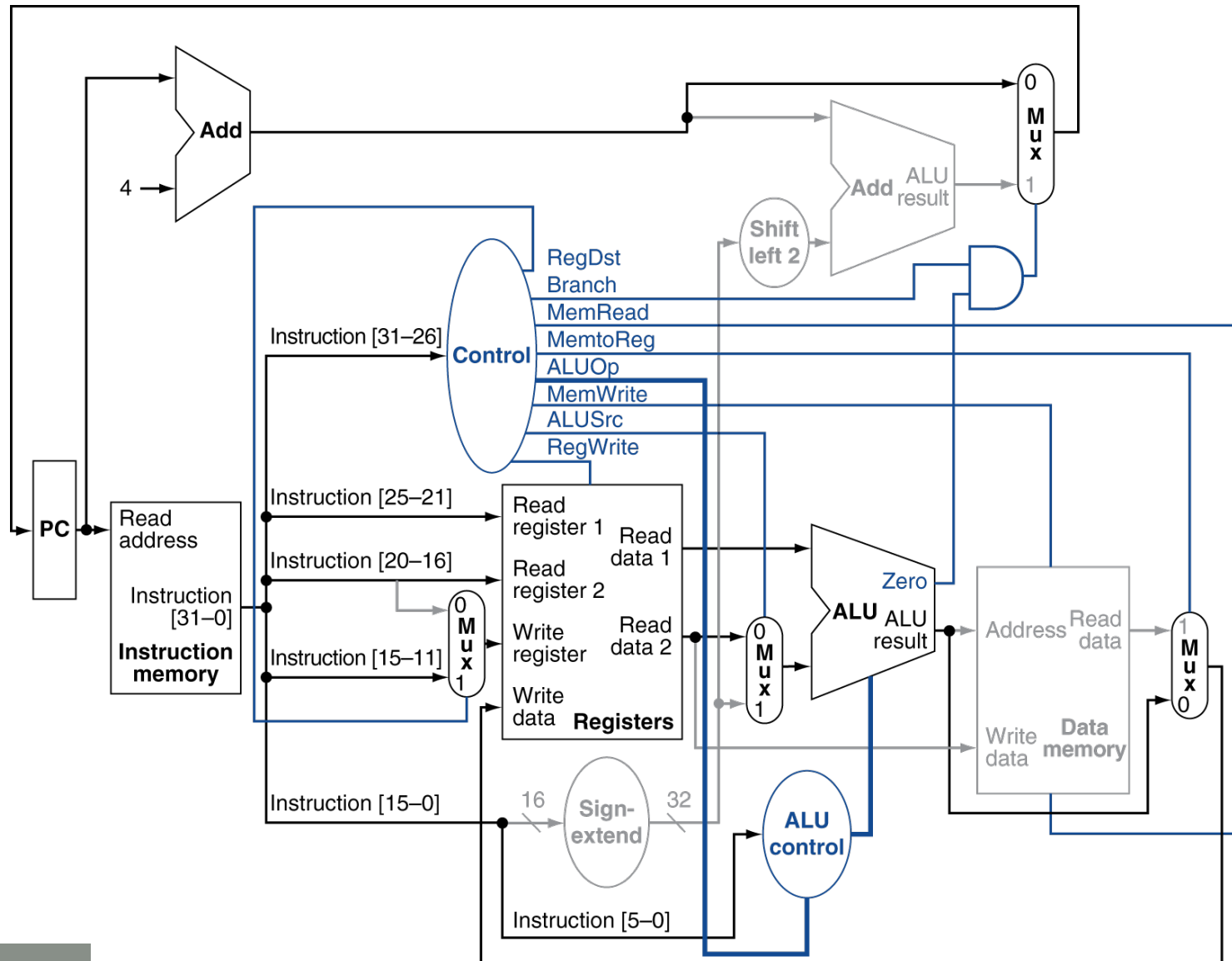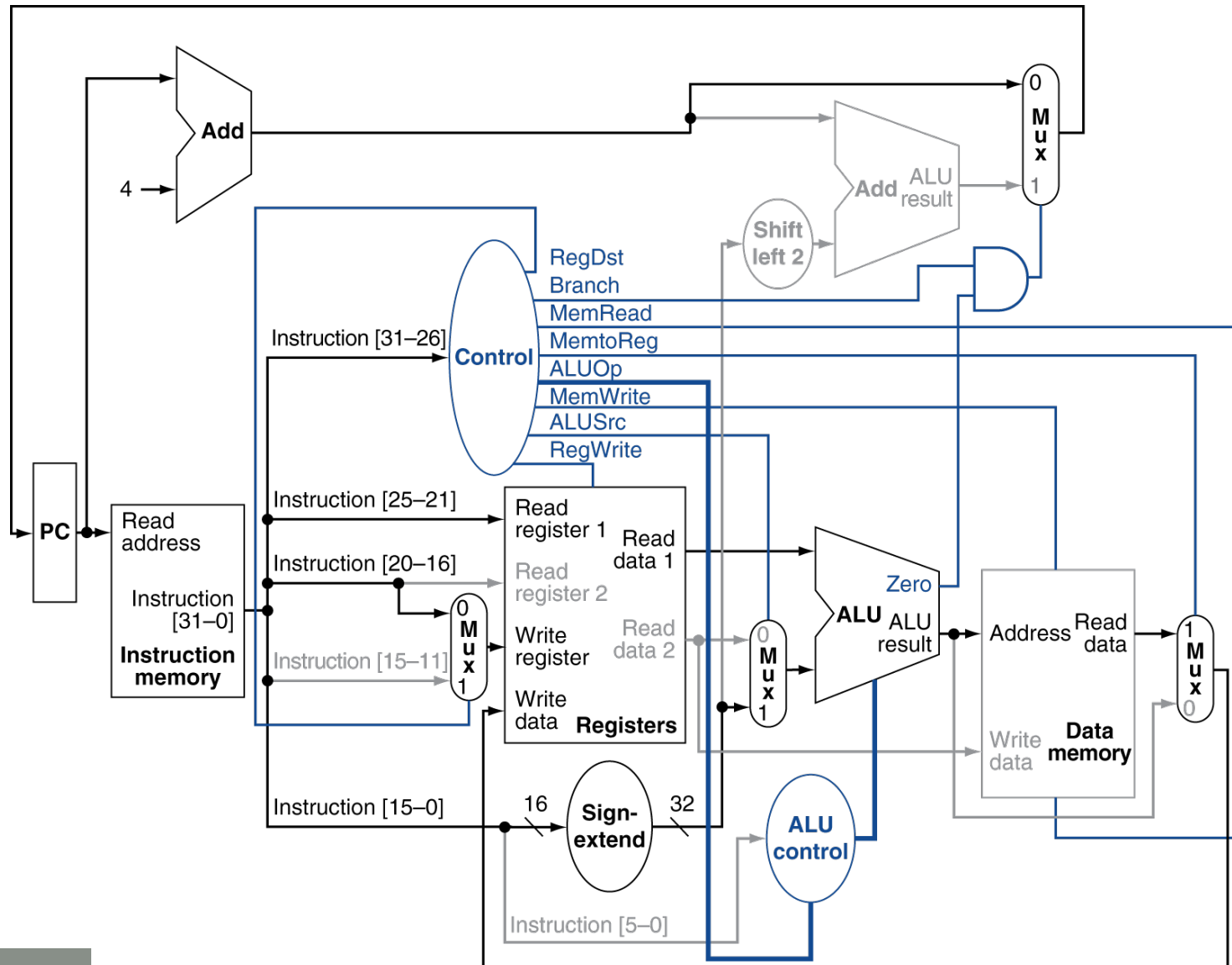
Datapath With Control
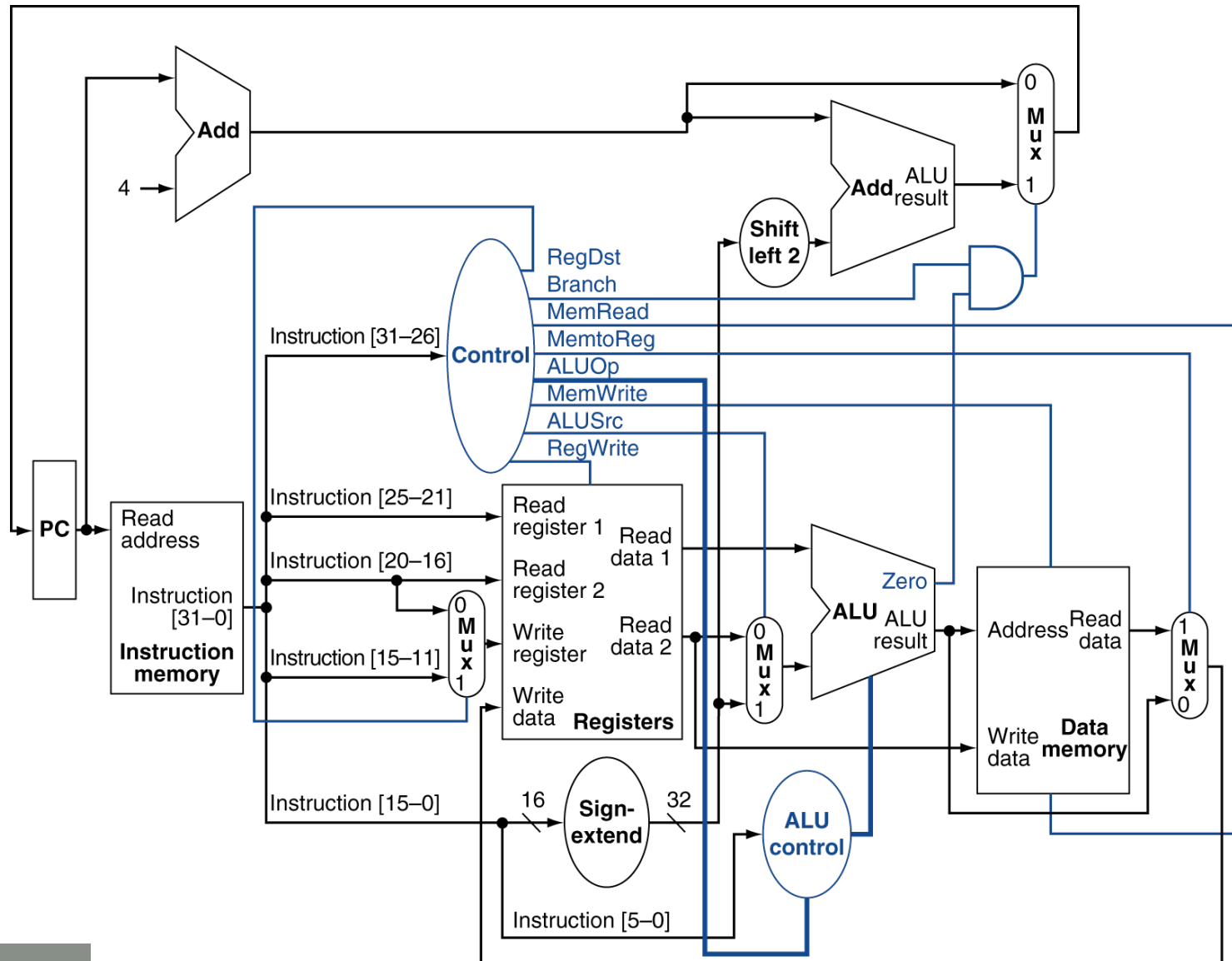
R-Type Instruction

Load Instruction

Branch-on-Equal Instruction
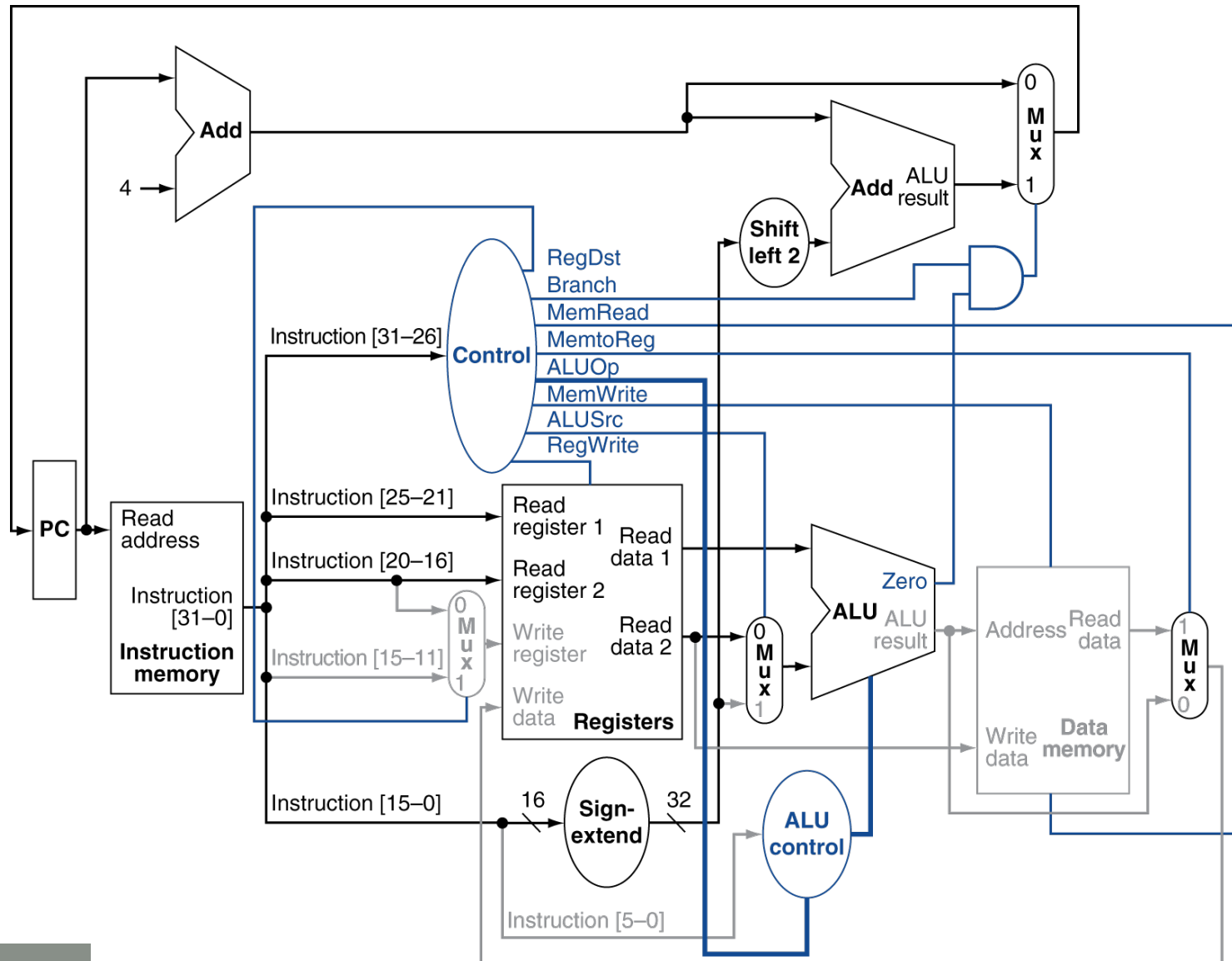
Implementing Jumps

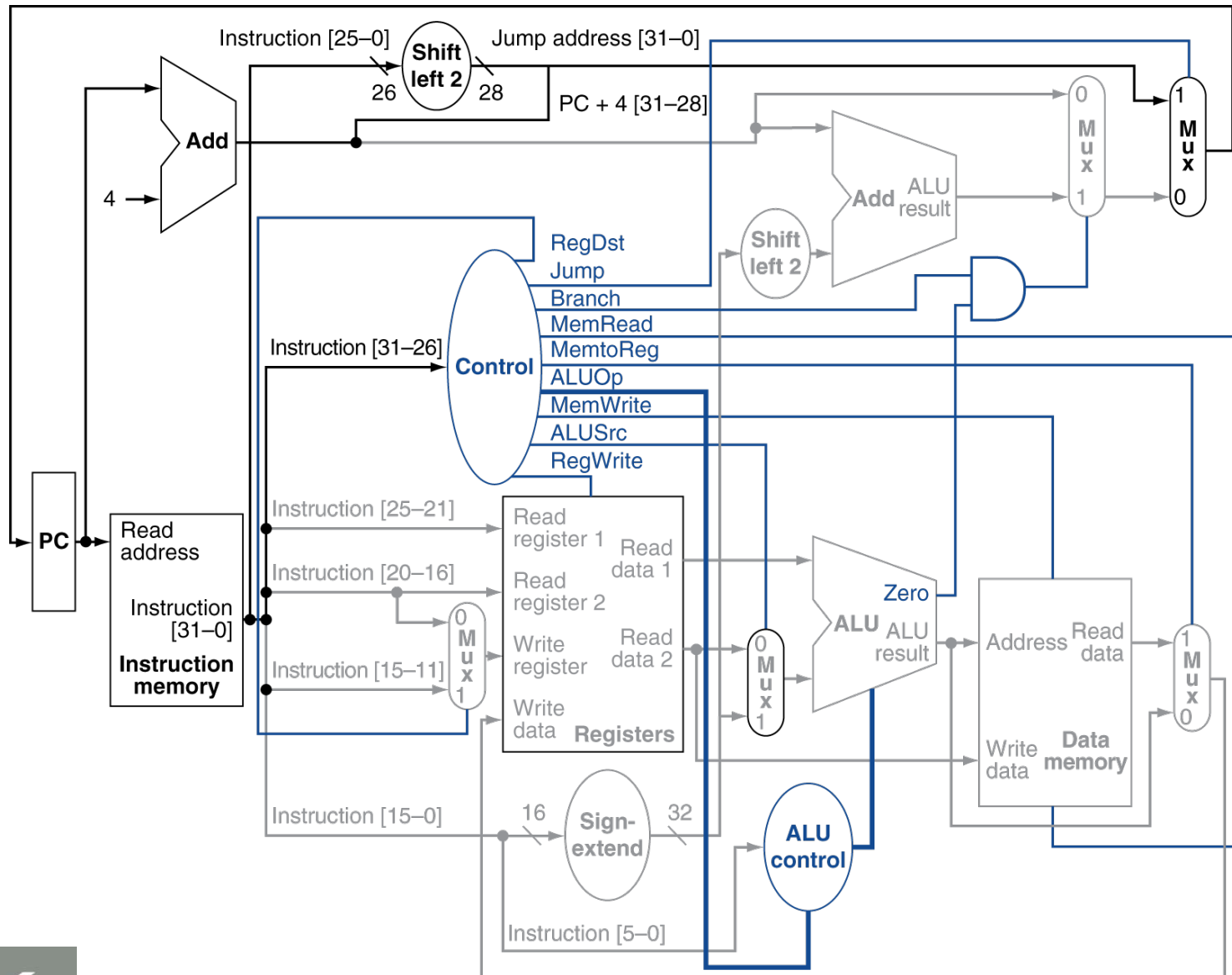| Jump	2	address
	31:26	25:0
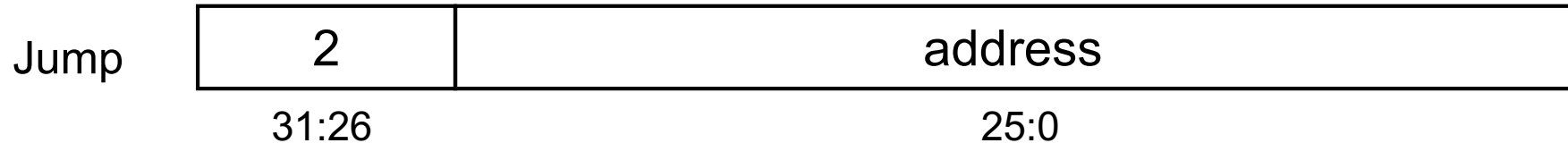
- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode

Datapath With Jumps Added

Performance Issues

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory $\rightarrow$ register file $\rightarrow$ ALU $\rightarrow$ data memory $\rightarrow$ register file
- Not feasible to vary period for different instructions
- Violates design principle
  - Making the common case fast
- We will improve performance by pipelining