

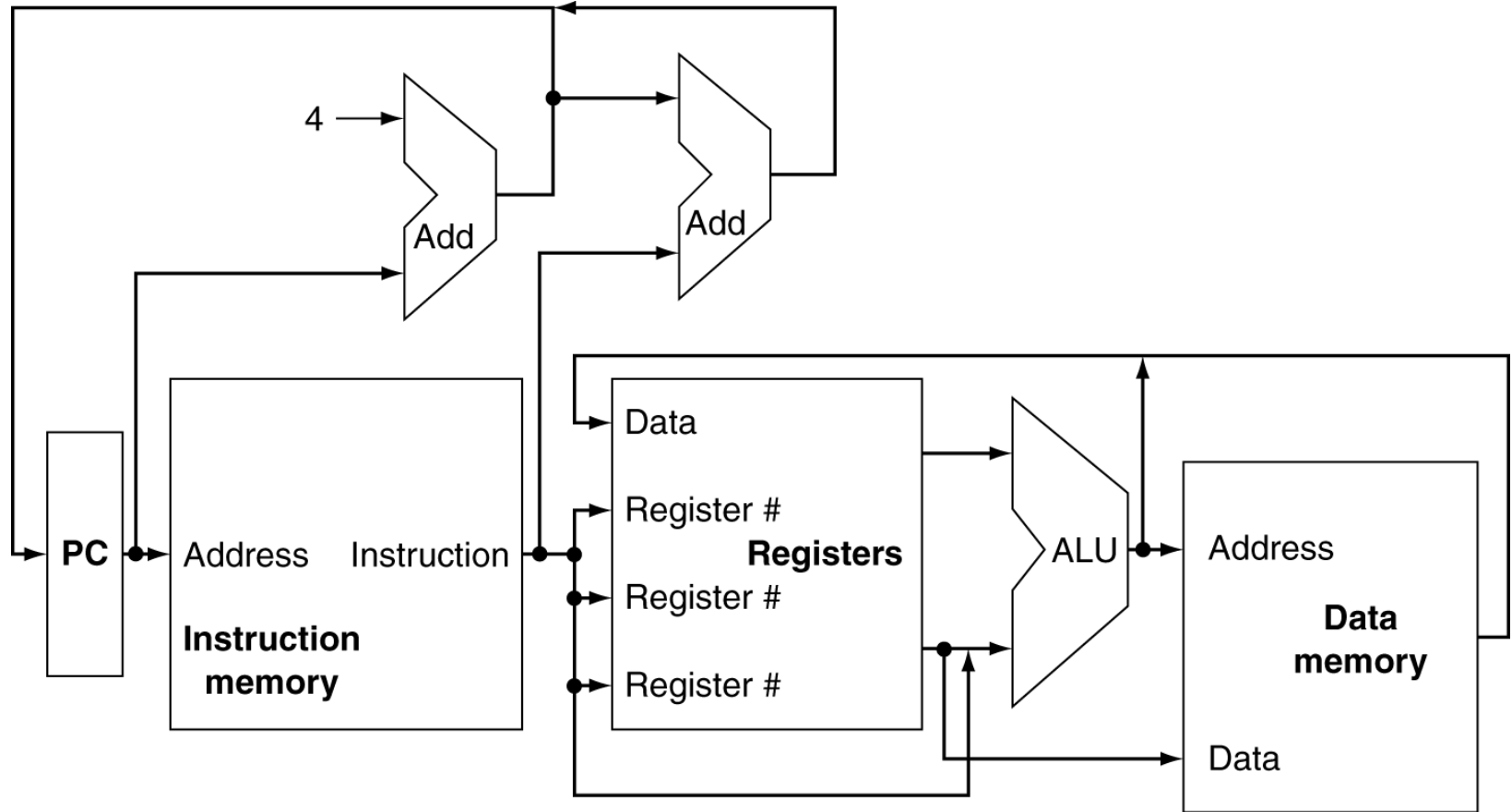
# ARM PROCESSOR

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine two LEGv8 implementations
  - A simplified version
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Memory reference: LDUR, STUR
  - Arithmetic/logical: add, sub, and, or, slt
  - Control transfer: beq, j

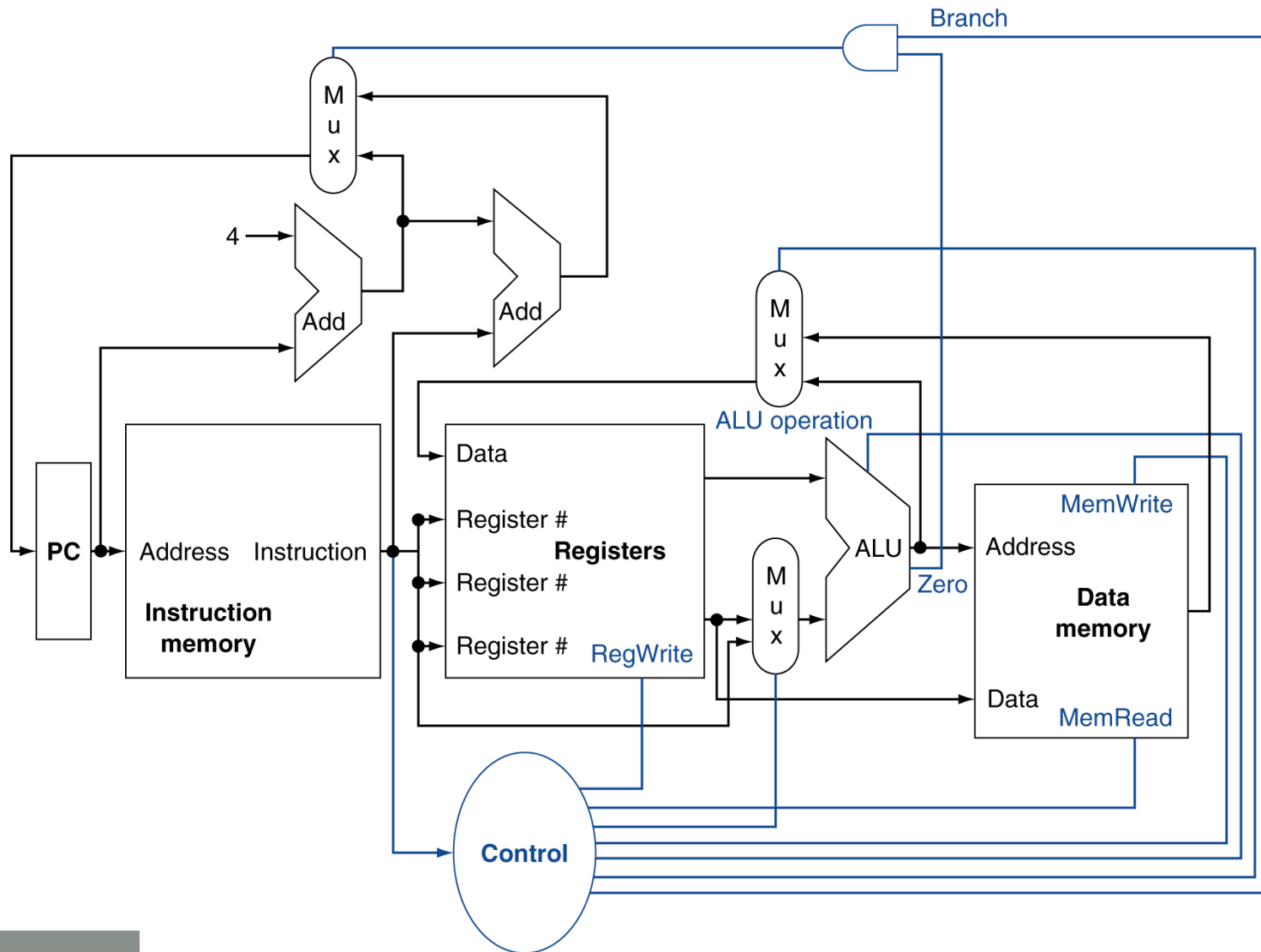
# Instruction Execution

- 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



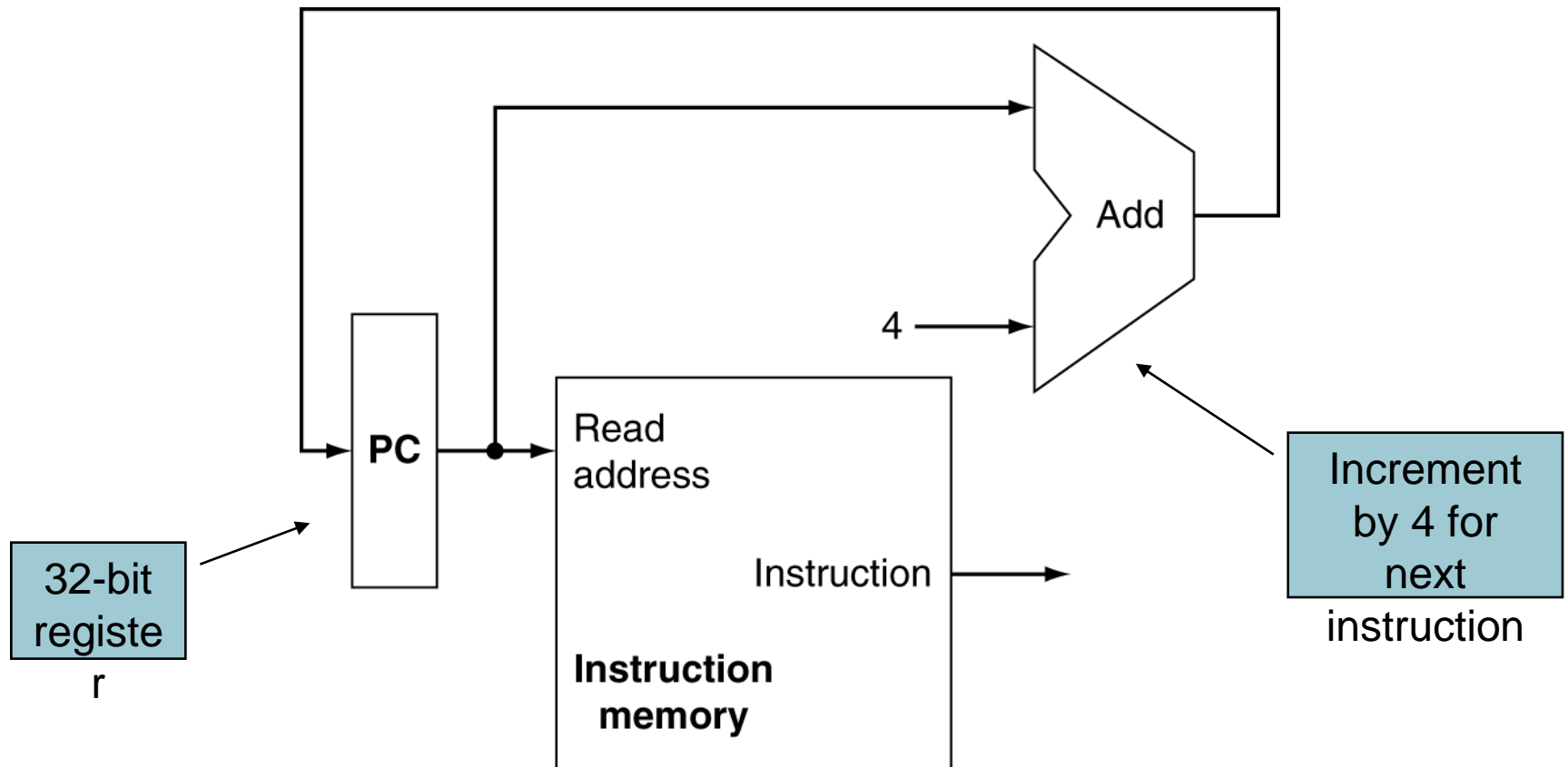
# Control



# Building a Datapath

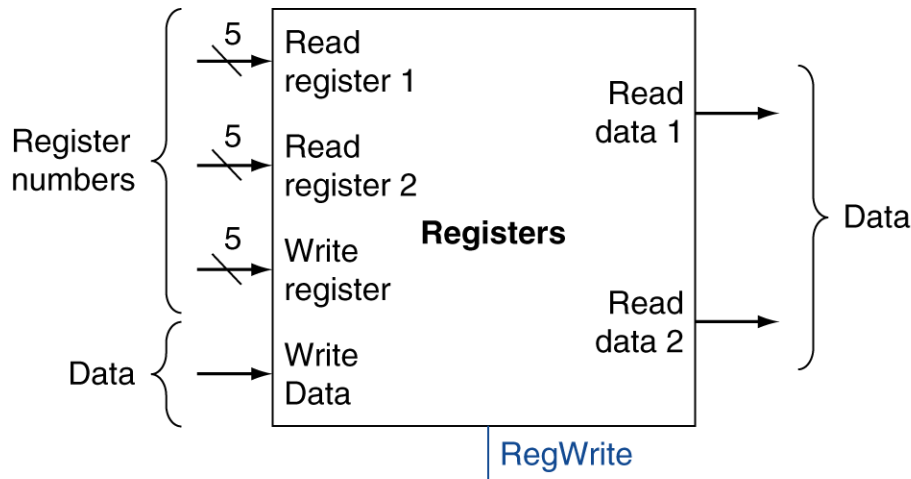
- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, mux's, memories, ...
- We will build a LEGv8 datapath incrementally
  - Refining the overview design

# Instruction Fetch

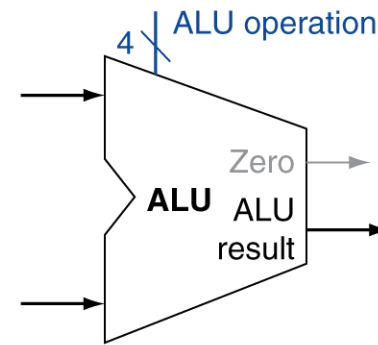


# R-Format Instructions

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



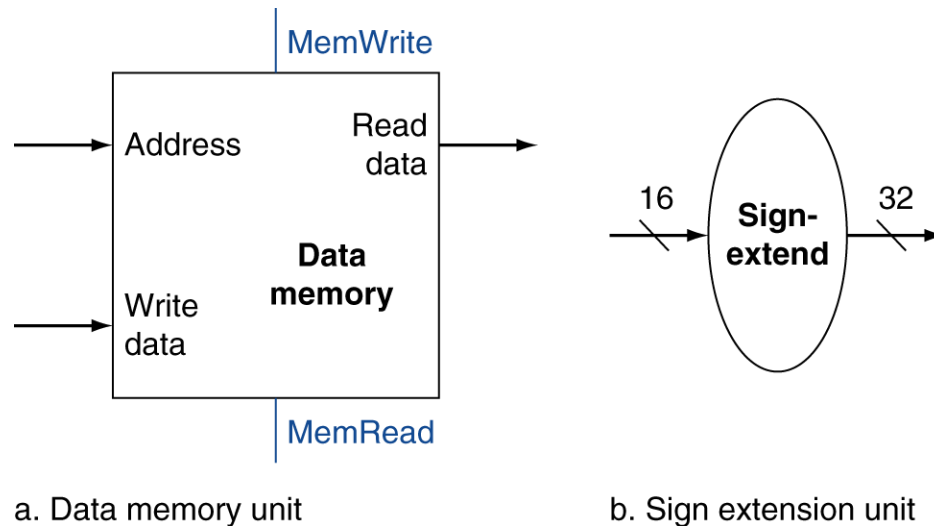
a. Registers



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

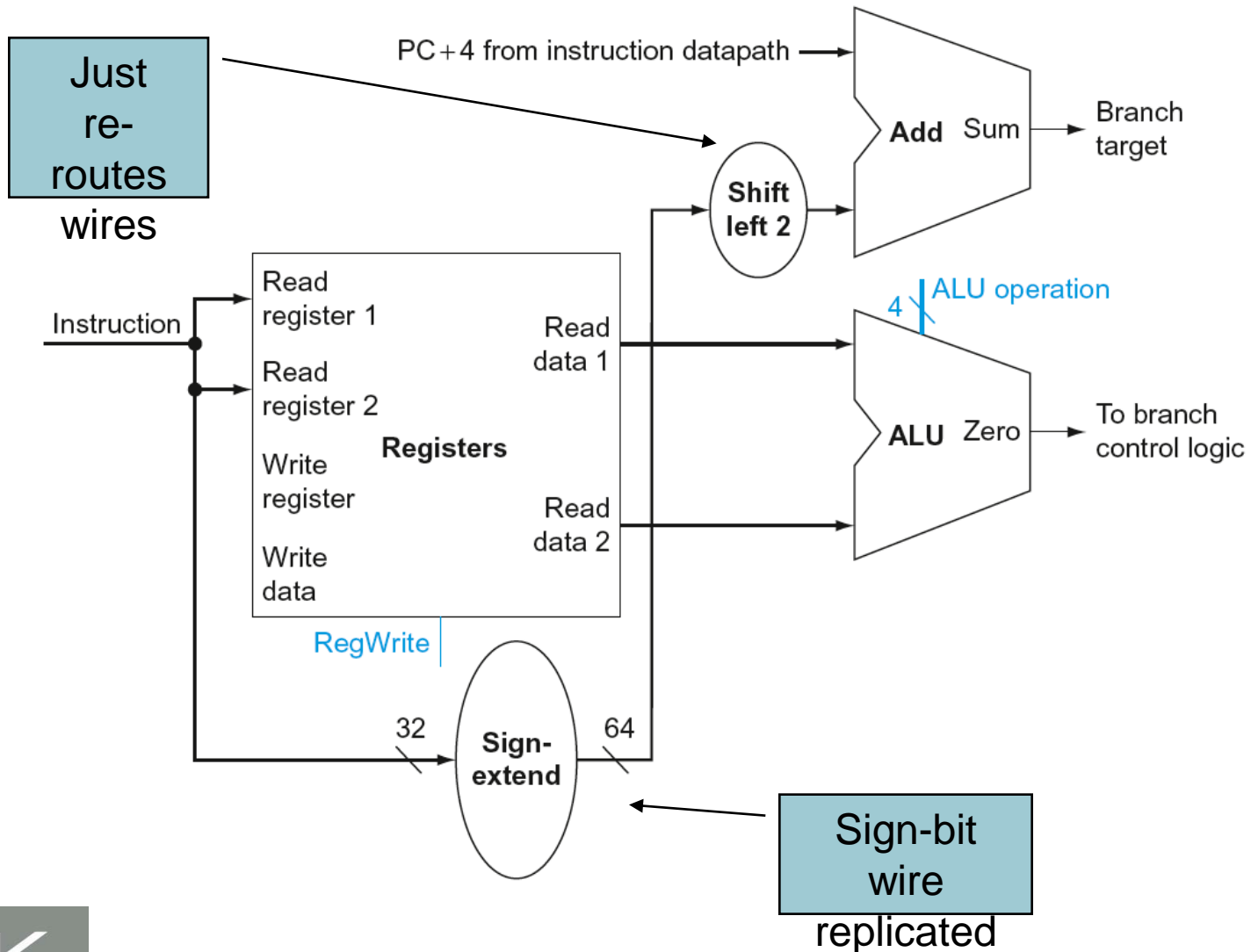




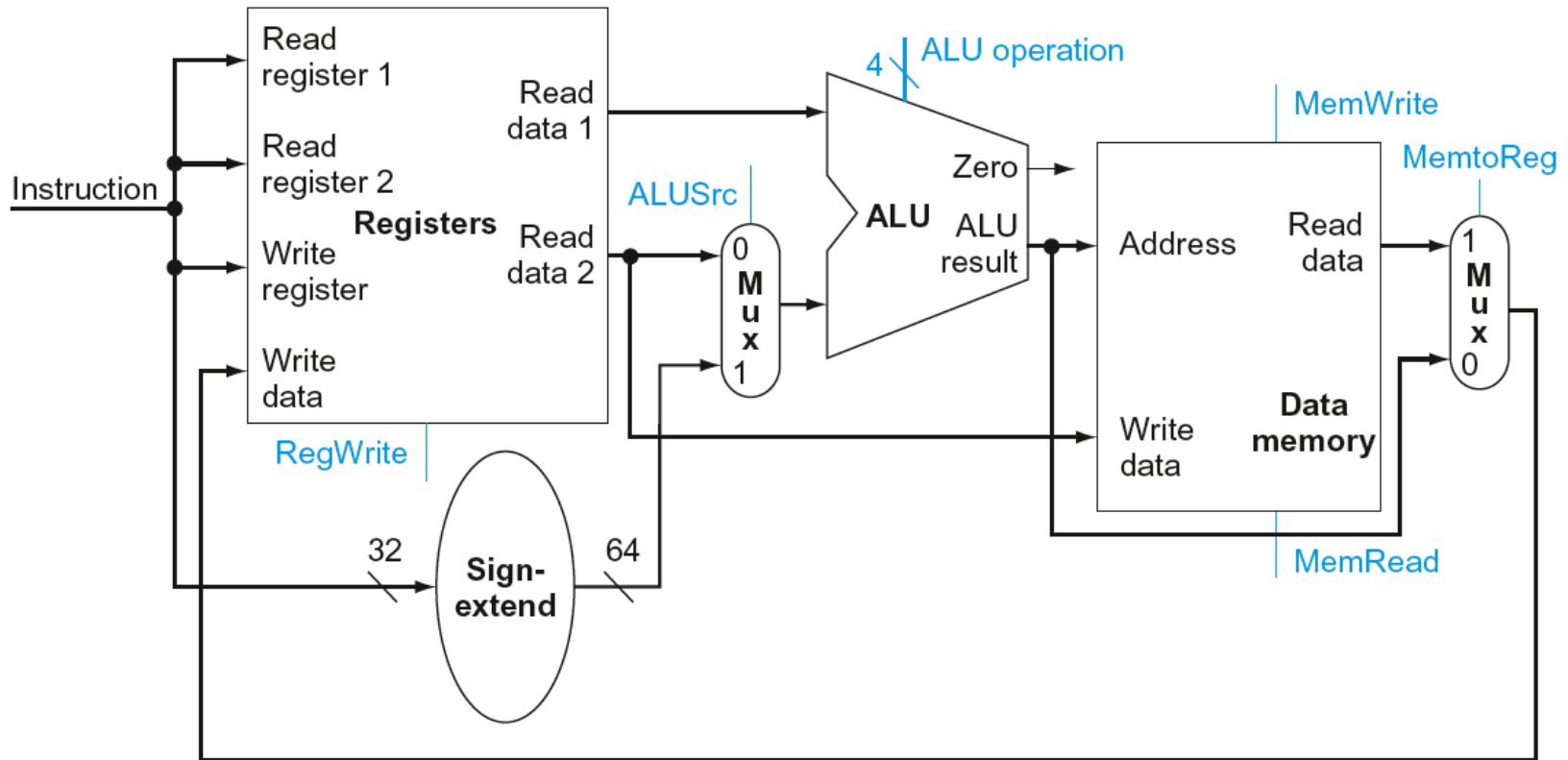
# Branch Instructions

- 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

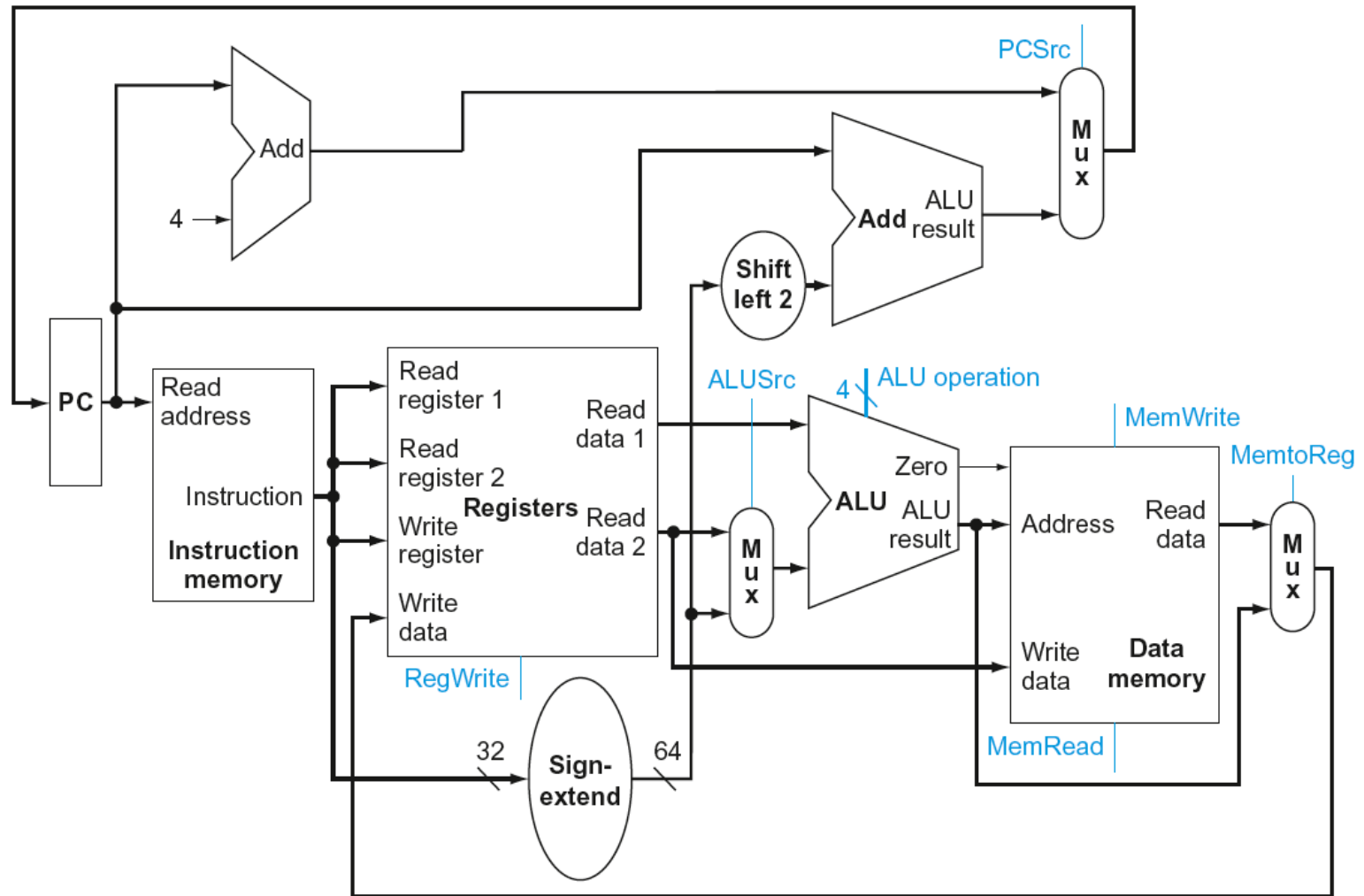
# Branch Instructions



# R-Type/Load/Store Datapath



# Full Datapath



# ALU Control

- ALU used for
  - Load/Store:  $F = \text{add}$
  - Branch:  $F = \text{subtract}$
  - R-type:  $F$  depends on opcode

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

# ALU Control

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

opcode	ALUOp	Operation	Opcode field	ALU function	ALU control
LDUR	00	load register	XXXXXXXXXXXX	add	0010
STUR	00	store register	XXXXXXXXXXXX	add	0010
CBZ	01	compare and branch on zero	XXXXXXXXXXXX	pass input b	0111
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		ORR	100101	OR	0001

# The Main Control Unit

## ■ Control signals derived from instruction

Field	opcode	Rm	shamt	Rn	Rd
Bit positions	31:21	20:16	15:10	9:5	4:0

a. R-type instruction

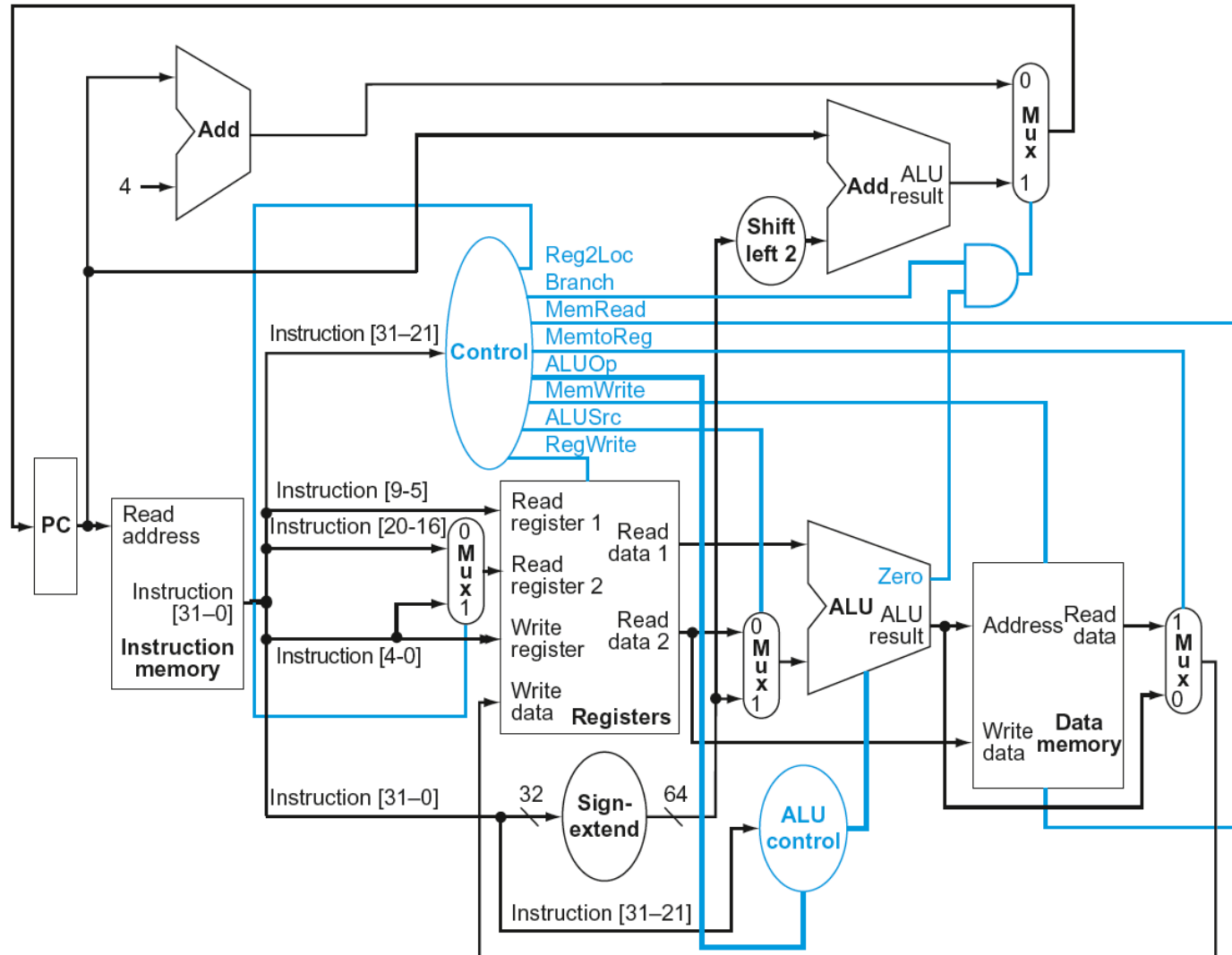
Field	1986 or 1984	address	0	Rn	Rt
Bit positions	31:21	20:12	11:10	9:5	4:0

b. Load or store instruction

Field	180	address	Rt
Bit positions	31:26	23:5	4:0

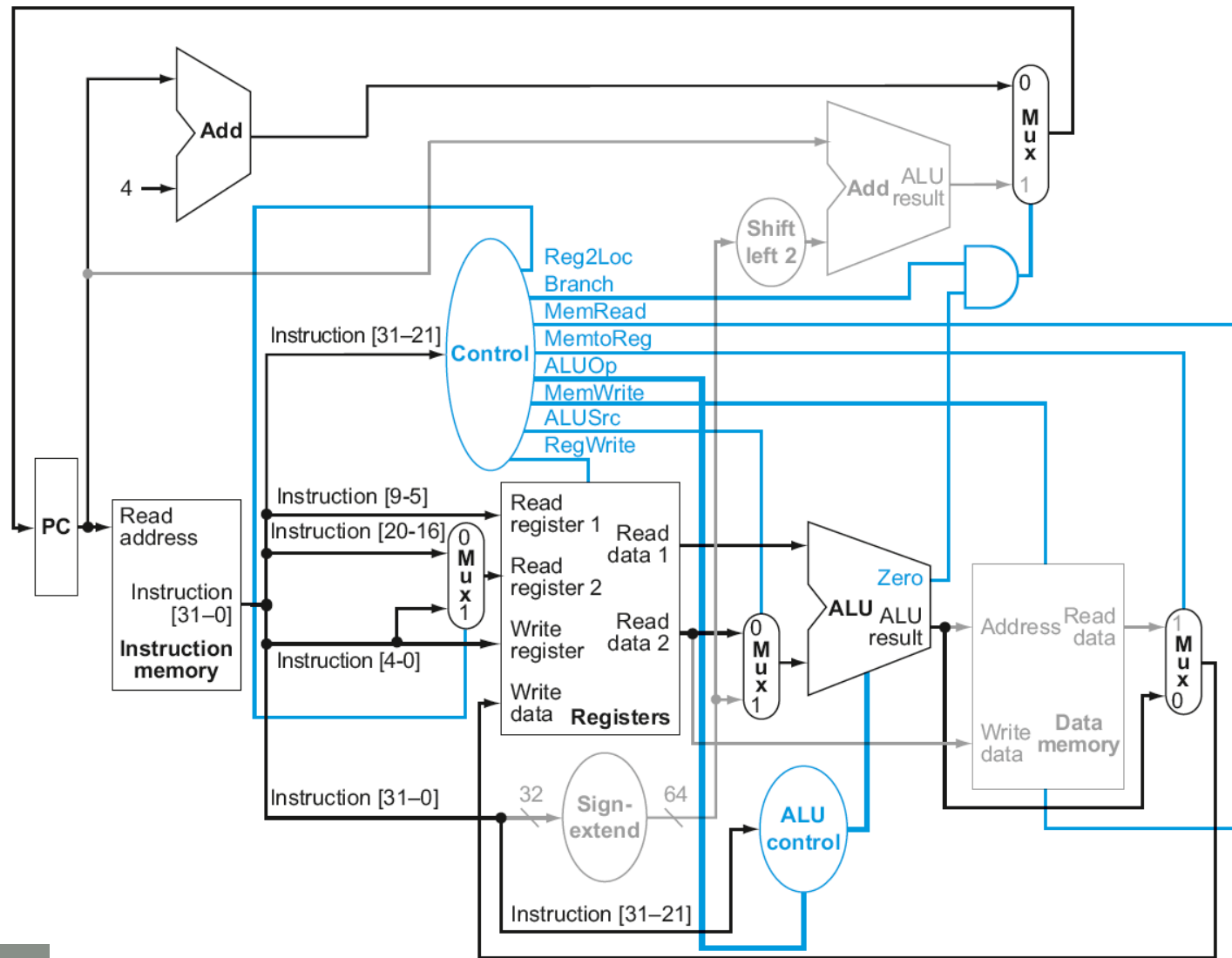
c. Conditional branch instruction

# Datapath With Control

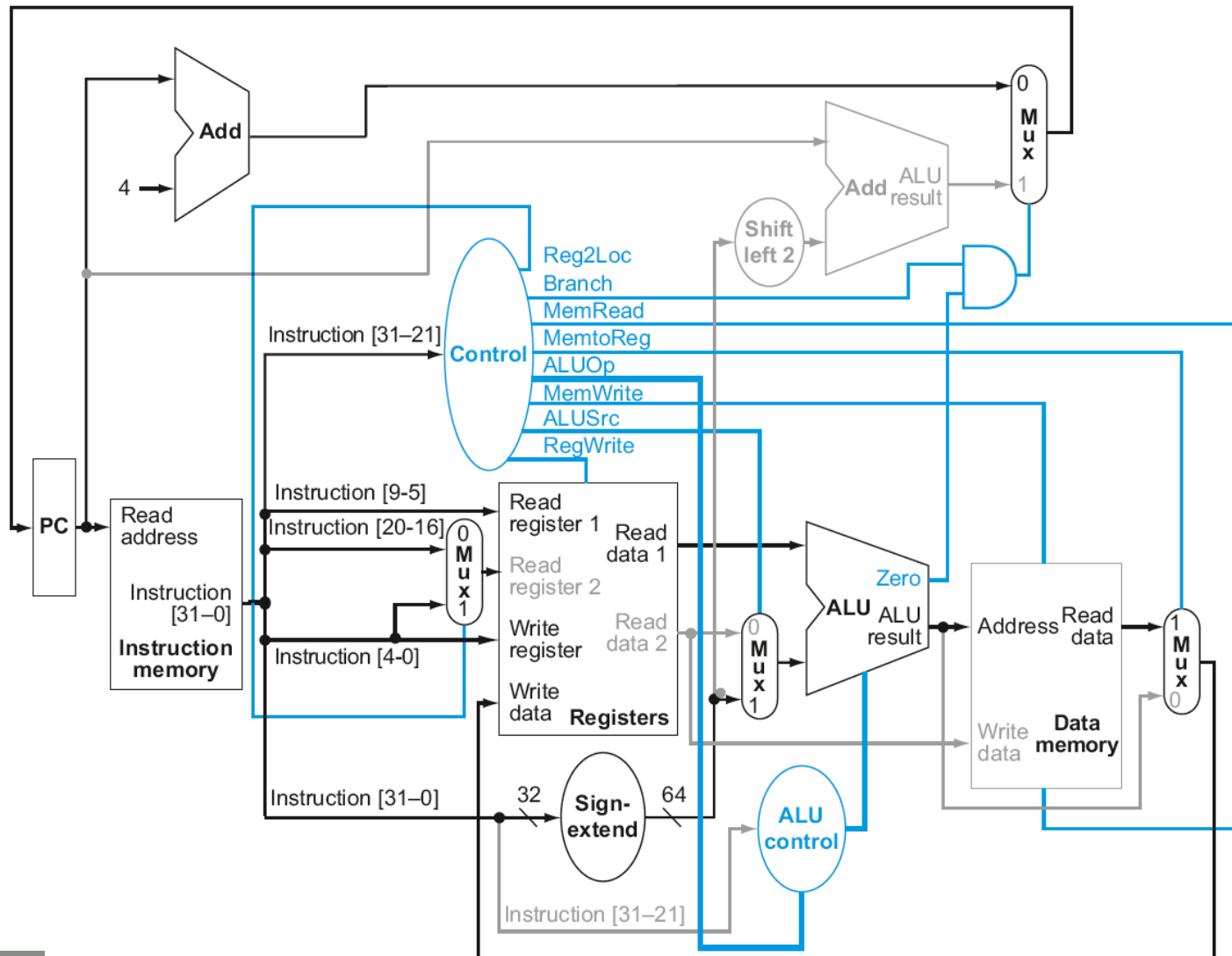




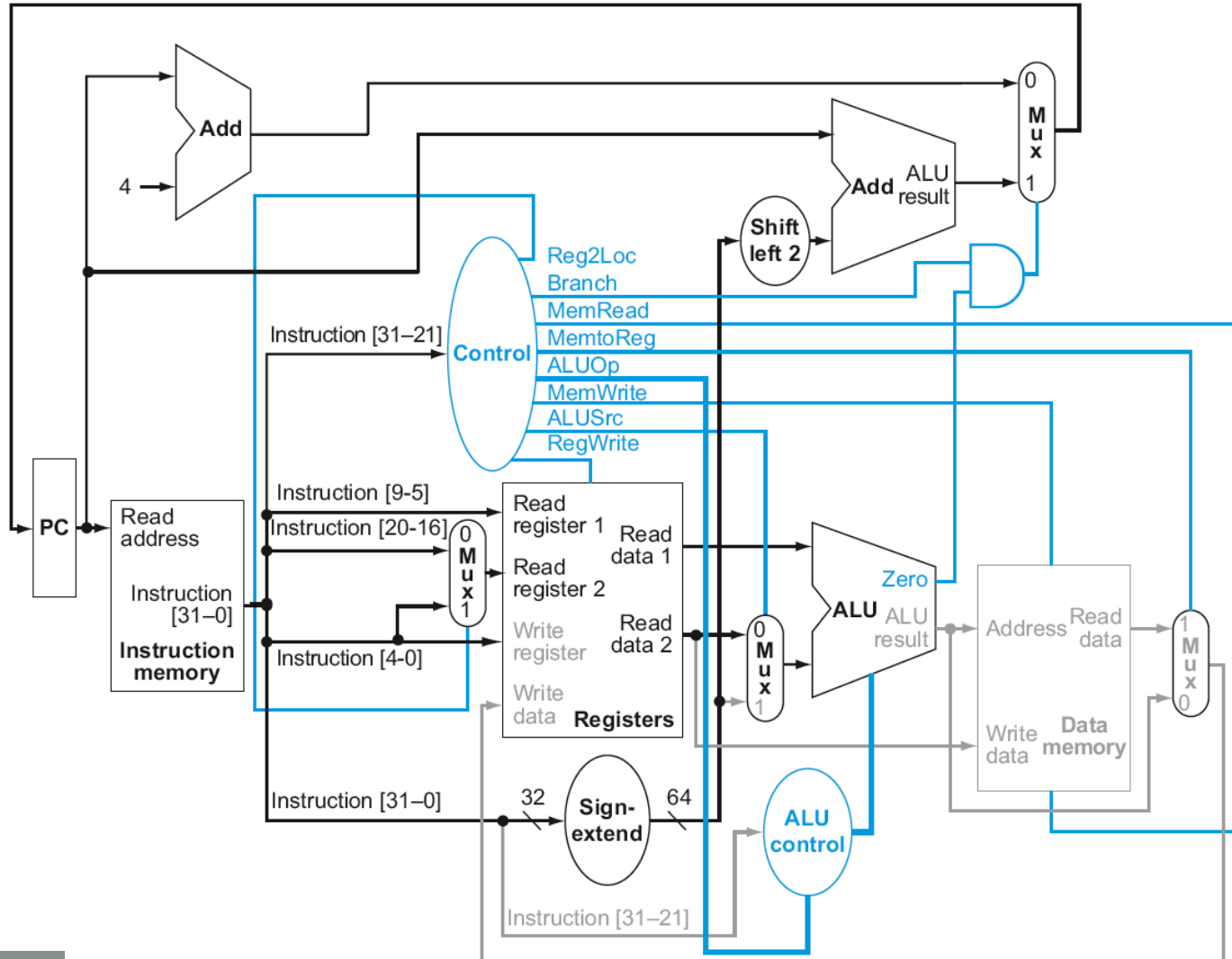
# R-Type Instruction



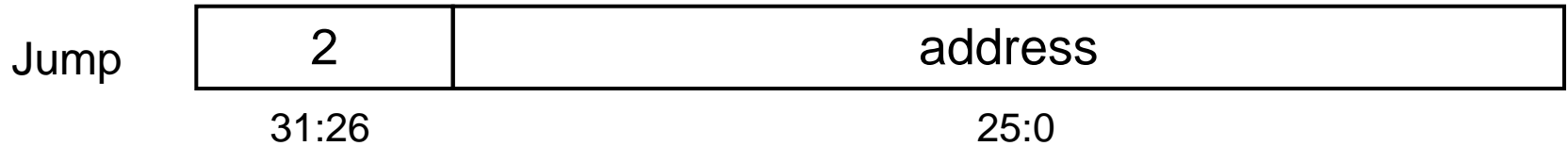
# Load Instruction



# CBZ Instruction



# Implementing Uncnd'l Branch



- 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 B Added

