Chapter 4

The Processor

Lecture 6

Dr. Karim Emara

Agenda

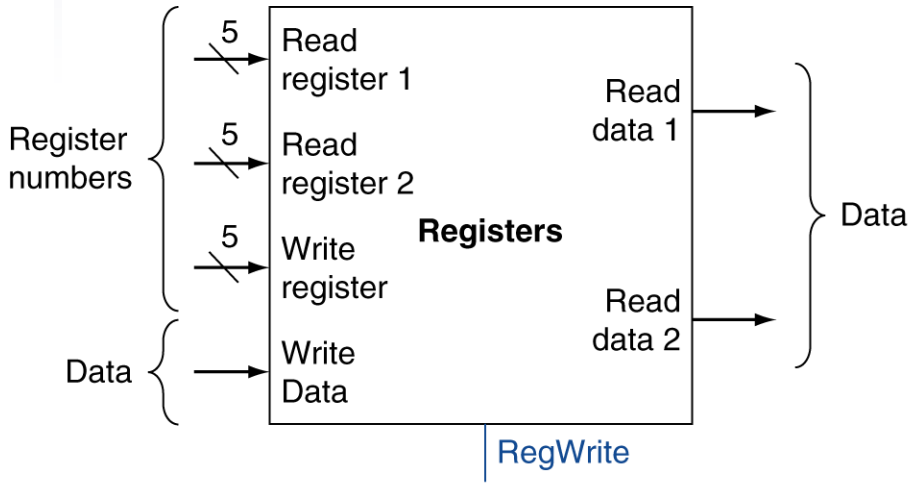
- CPU Design
- MIPS Datapath (single-cycle)
- Control unit
- Adding jump instruction

CPU Design

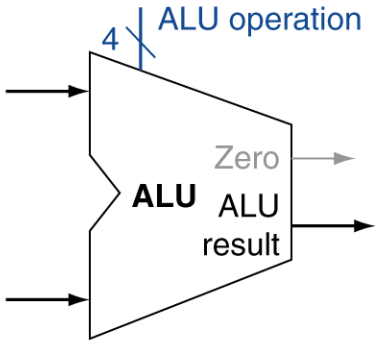
- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine two MIPS implementations
  - **A simplified version** (whole instruction in a single cycle)
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Memory reference: **`lw`, `sw`**
  - Arithmetic/logical: **`add`, `sub`, `and`, `or`, `slt`**
  - Control transfer: **`beq`, `j`**

Register File & ALU

- The 32 registers are arranged as register file where any two registers can be selected and a third can be written
- ALU can perform 6 operations



a. Registers



b. ALU

	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

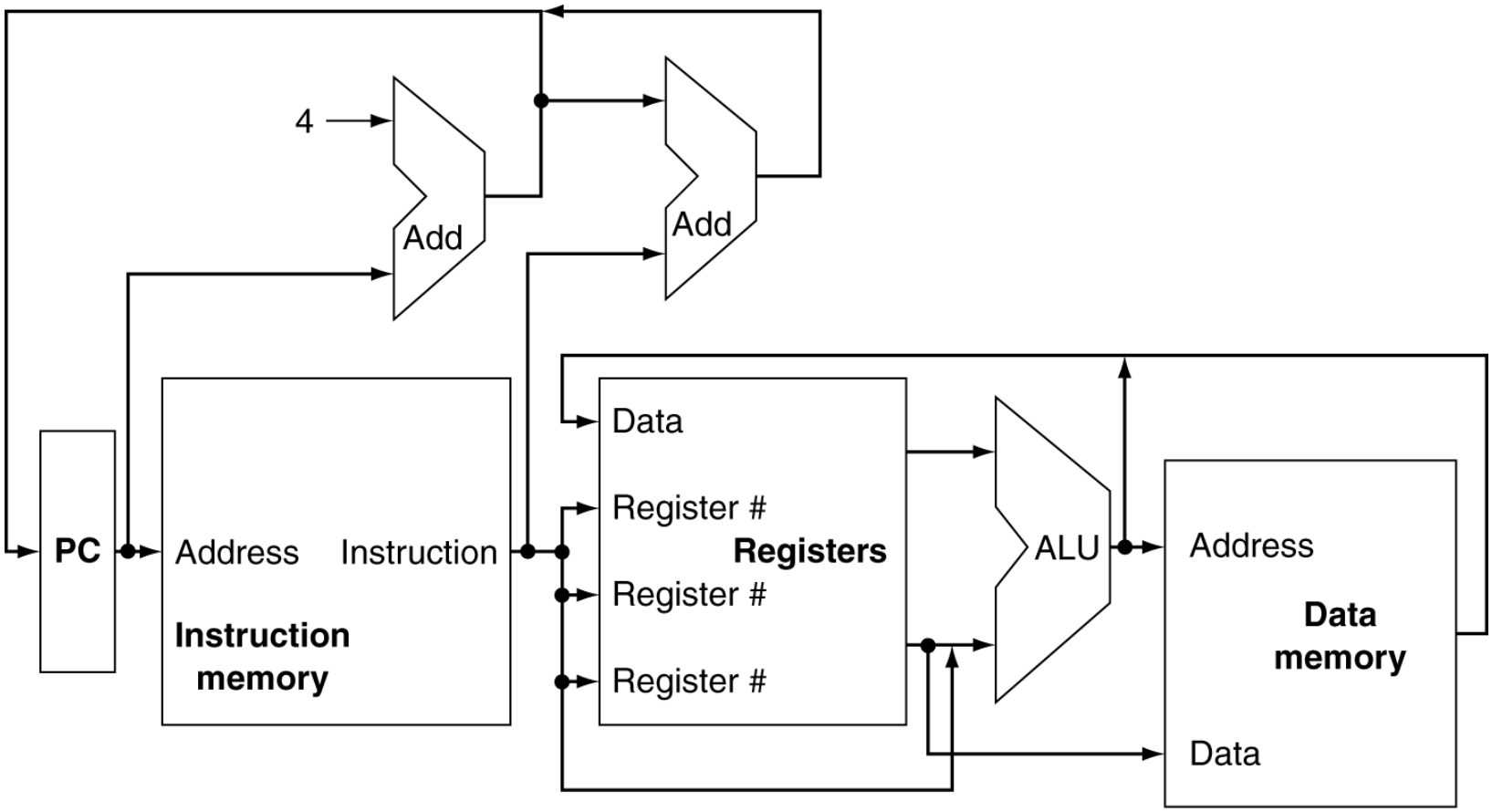
Datapath and control unit

- To design CPU, we need two components:
  - **Datapath** in which the instruction and data move to execute the instructions
    - Analogy: road network
  - **Control unit** which determines which parts of the datapath are currently active
    - Analogy: traffic lights

Instruction Execution

- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result for R-type instructions
    - Memory address for load/store (base address + offset)
    - Compare registers given in beq instruction
  - Access data memory for load/store
  - PC ← target address or PC + 4

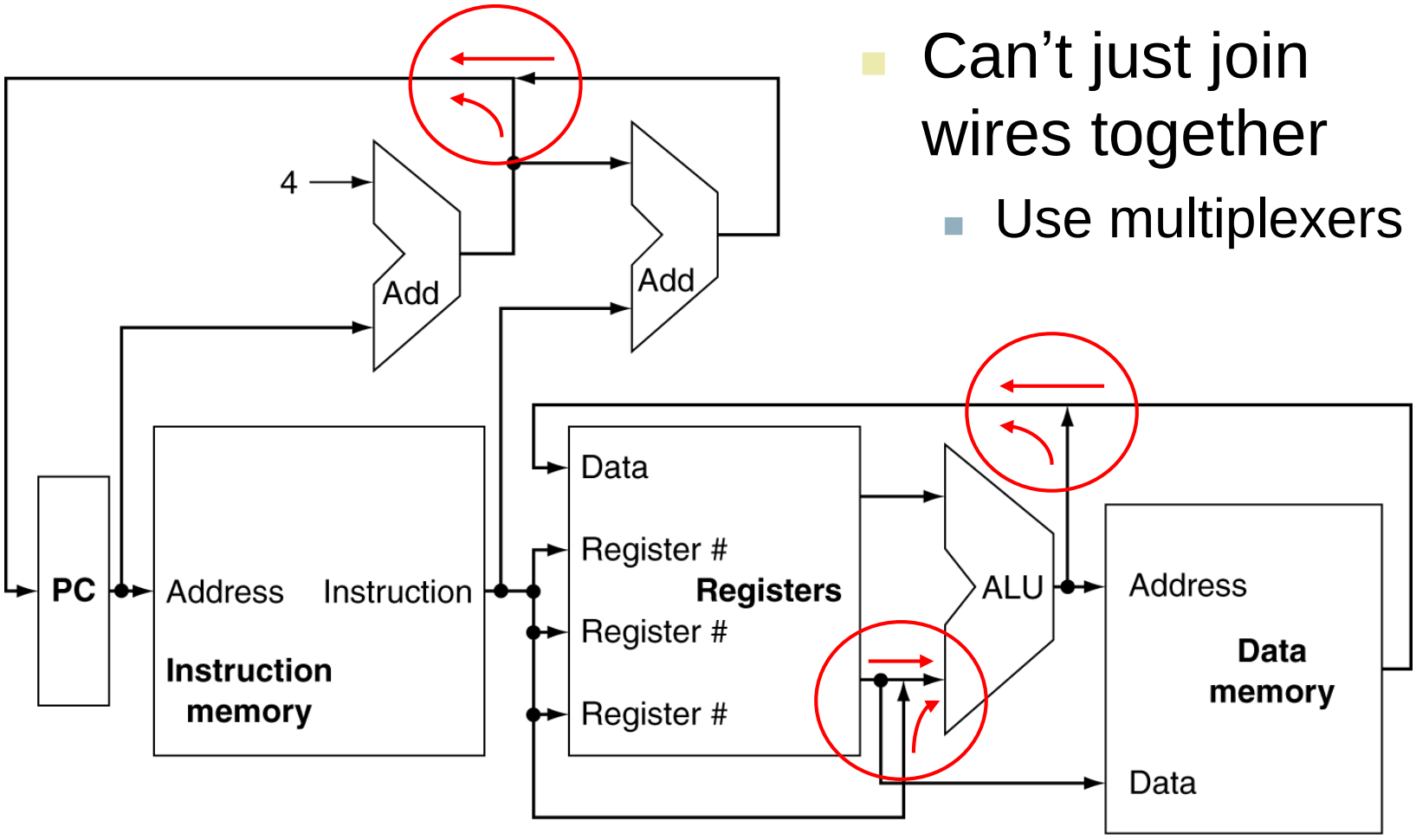
CPU Overview



Composing the Elements

- Simplified 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

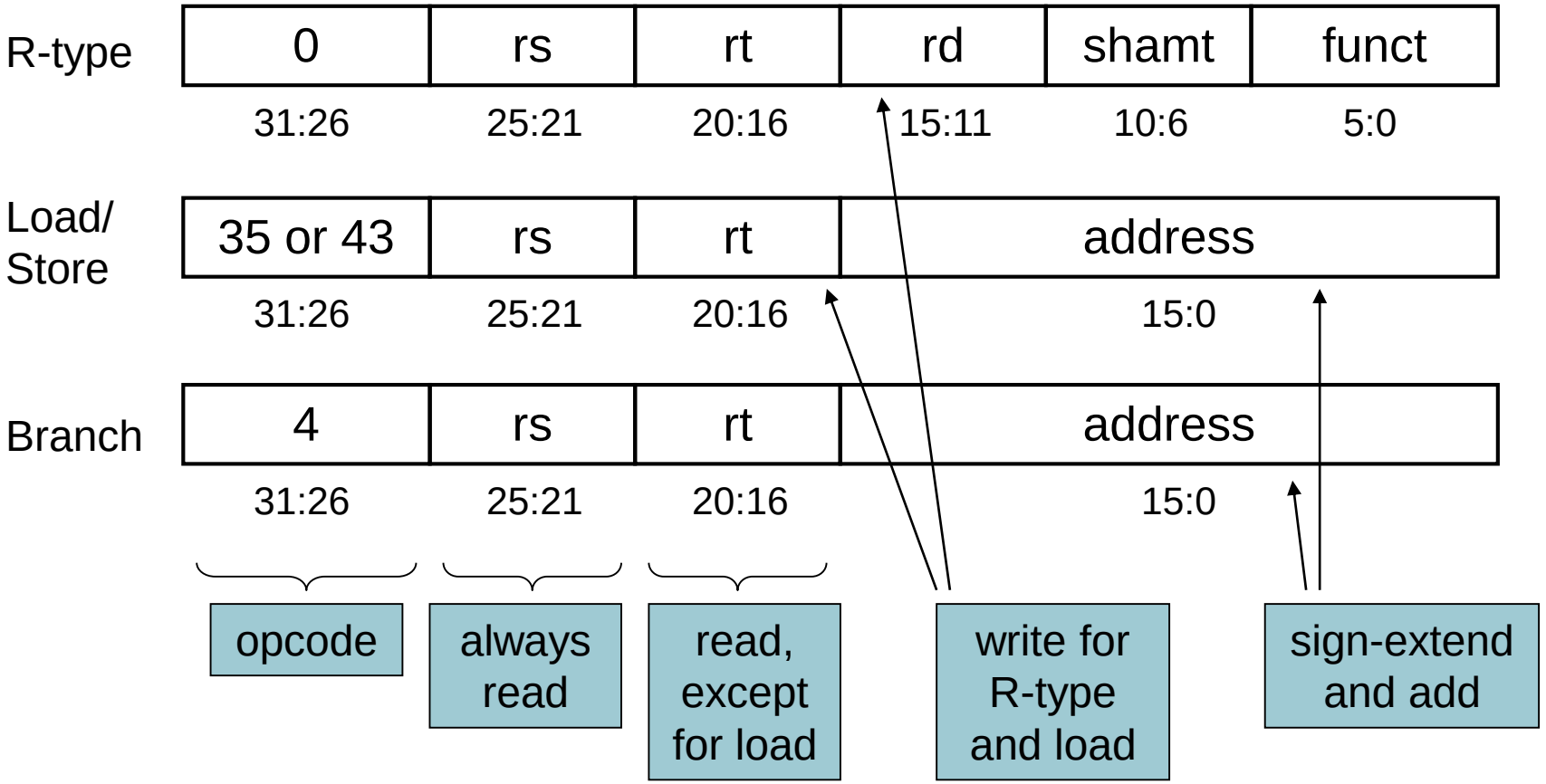
Multiplexers



- Can't just join wires together
  - Use multiplexers

Instruction format

- Control signals derived from instruction



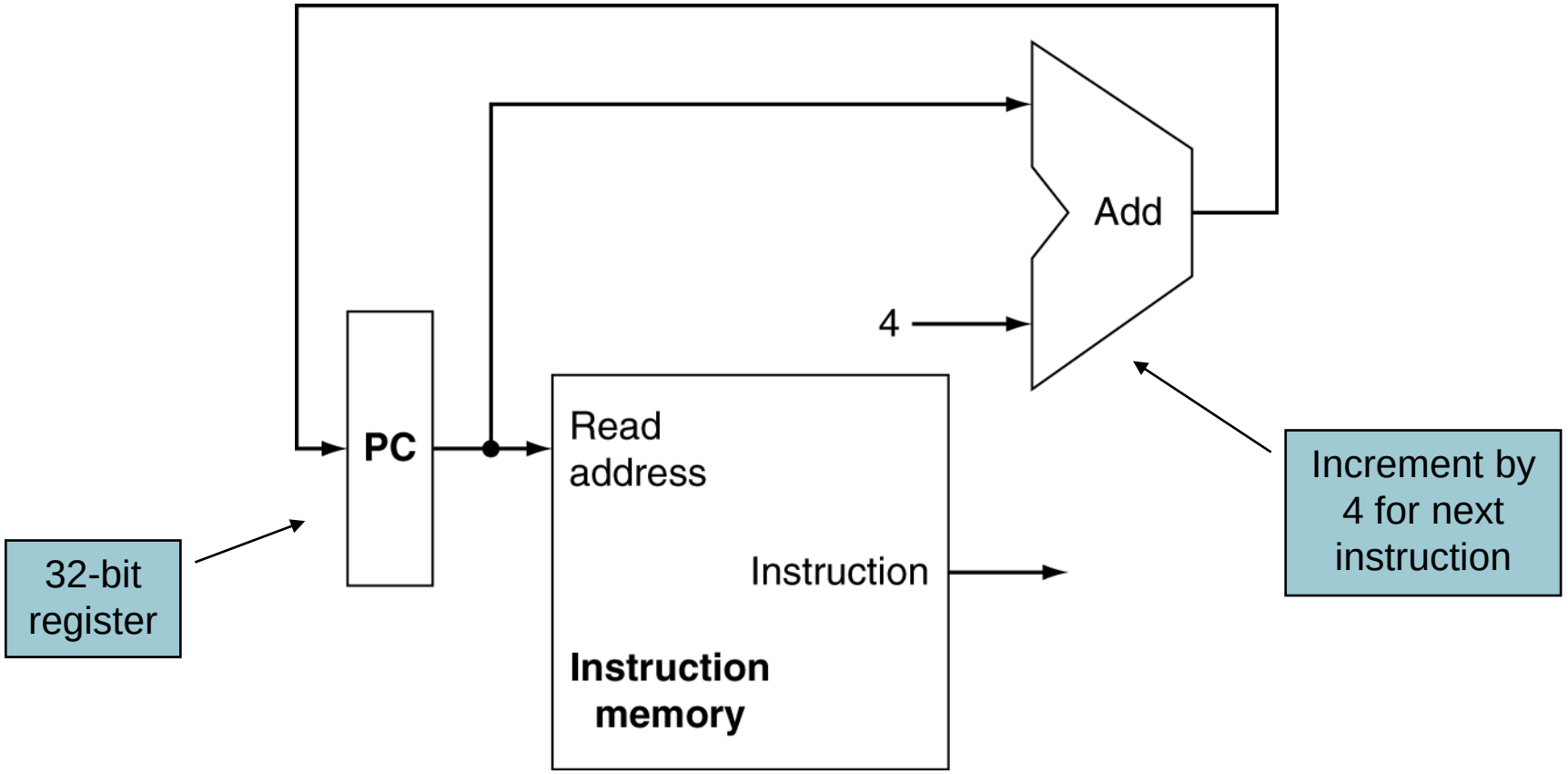
Agenda

- CPU Design
- MIPS Datapath (single-cycle)
- Control unit
- Adding jump instruction

Building a Datapath

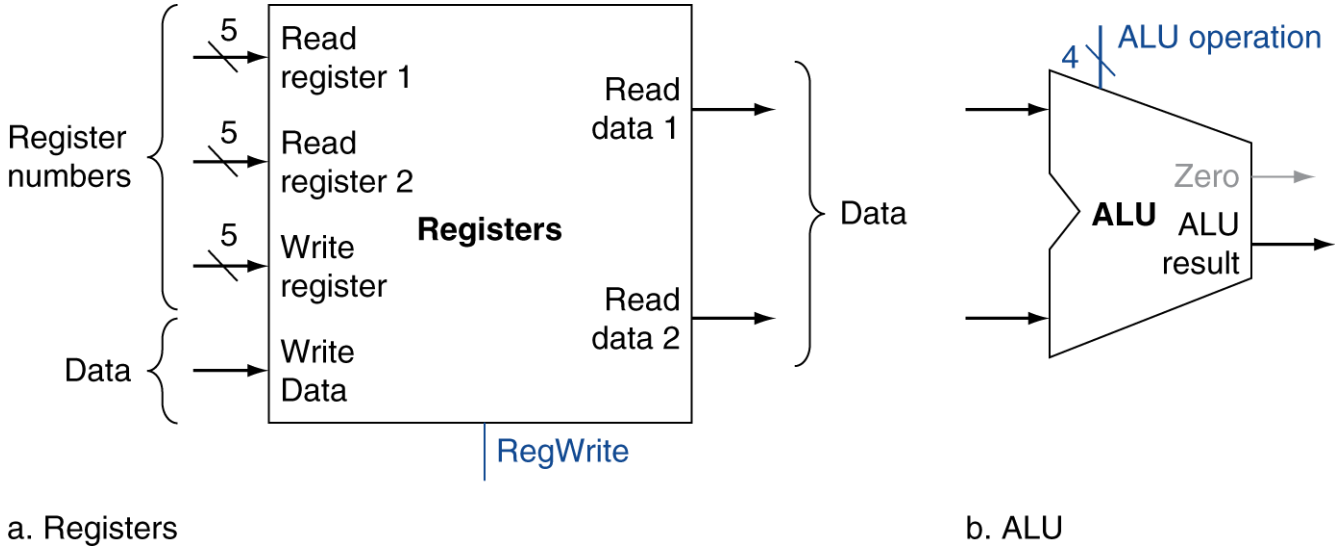
- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, mux's, memories, …
- We will build a MIPS datapath incrementally
  - Refining the overview design

Instruction Fetch



R-Format Instructions

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

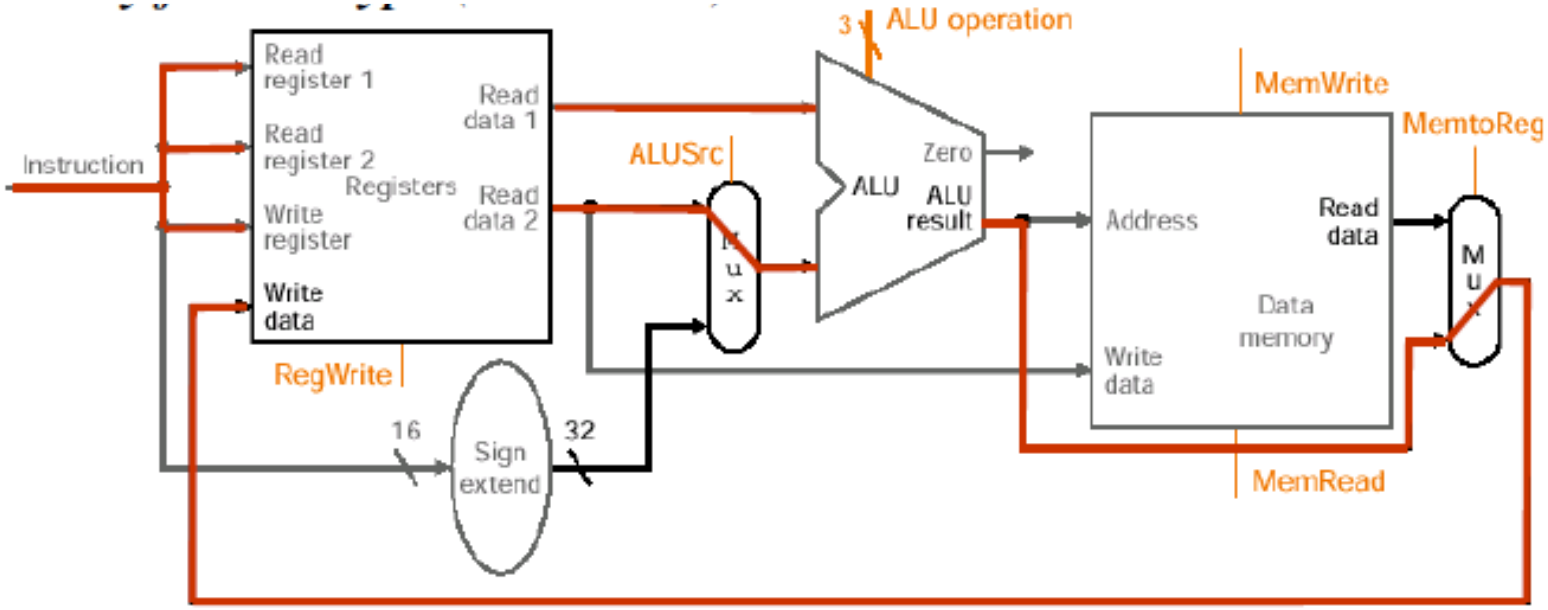


a. Registers

b. ALU

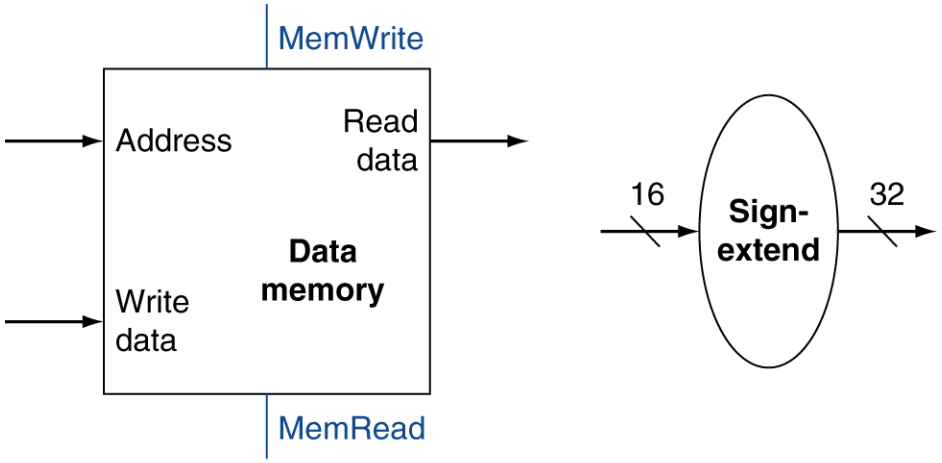
Execute R-type

- R-type:
  - RF: Read rs and rt and put rd at write code
  - ALU: 2 reg data to input, Op based on funct, output to RF write data
  - Memory (no read or write)



Load/Store Instructions

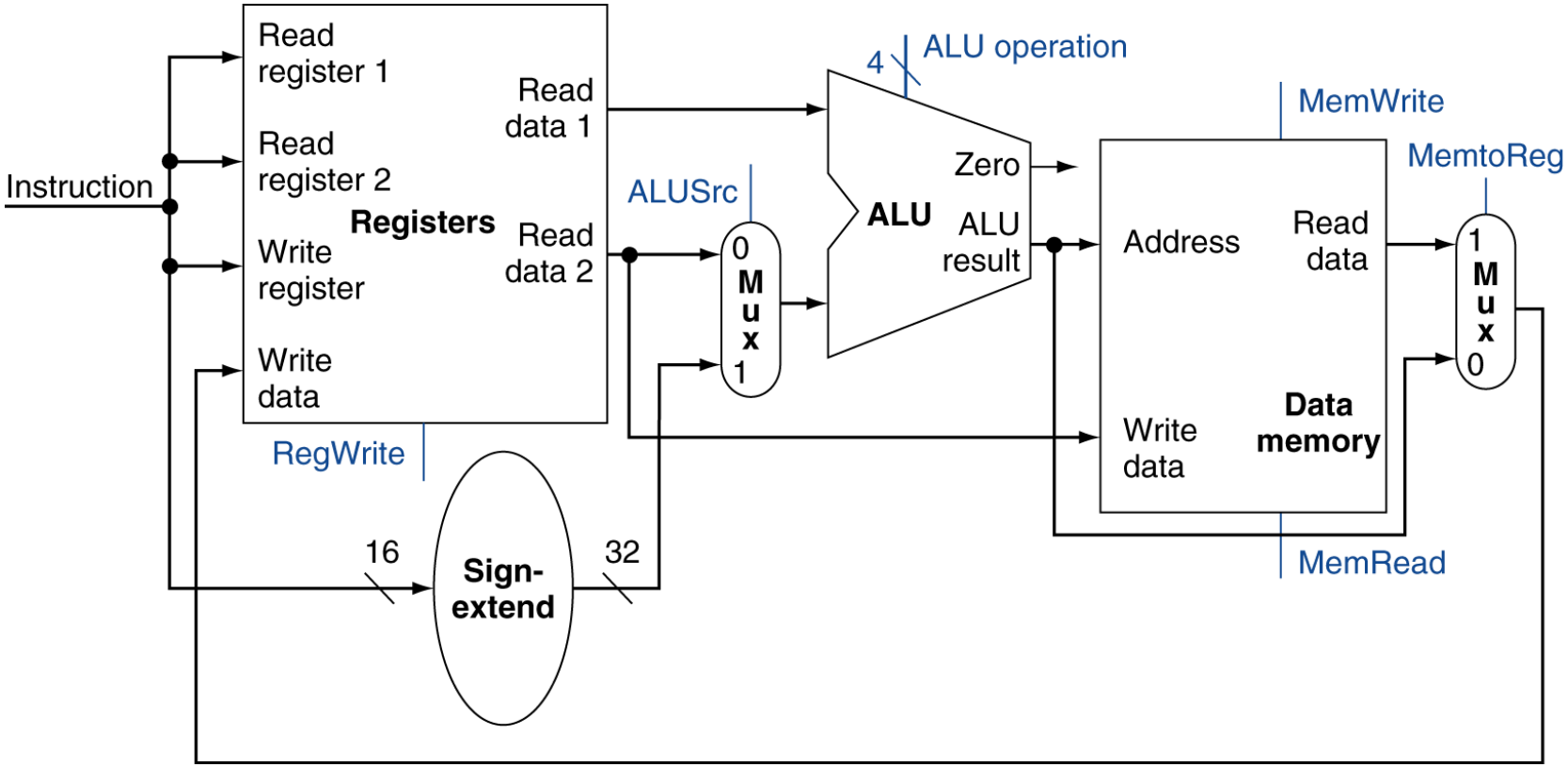
- Read register operands
- Calculate address using 16-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory



a. Data memory unit

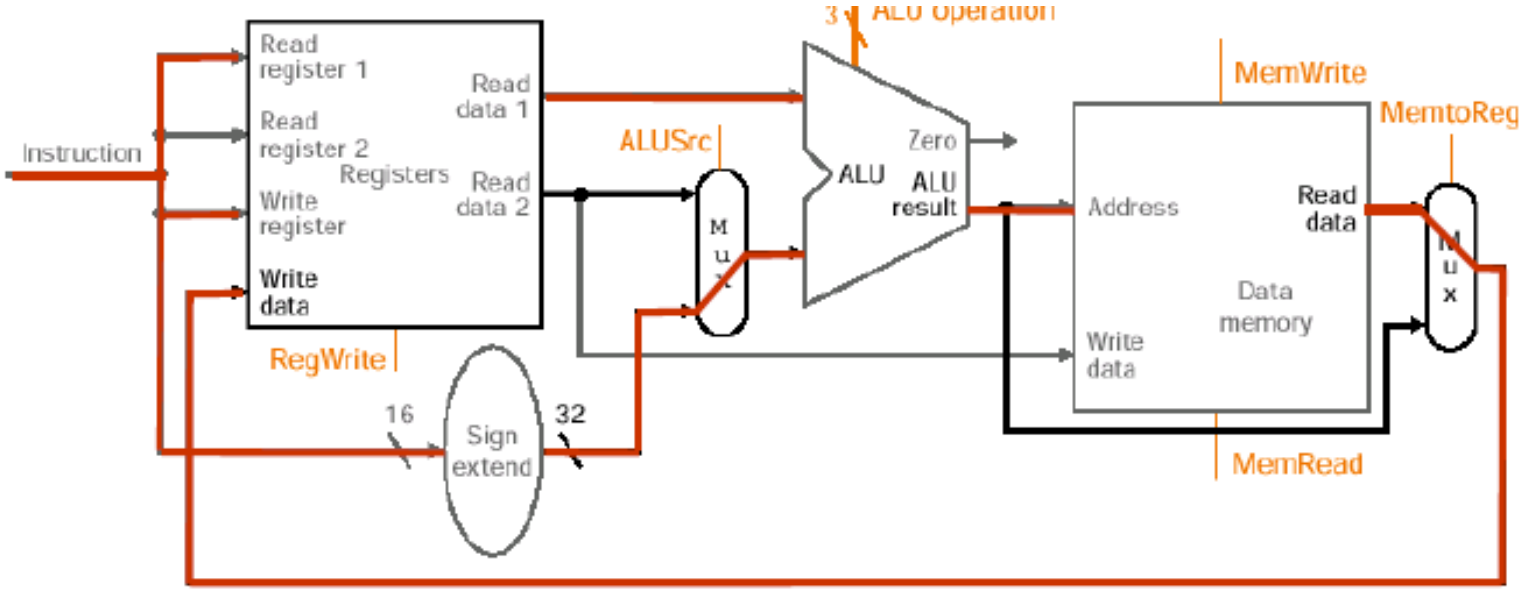
b. Sign extension unit

R-Type/Load/Store Datapath



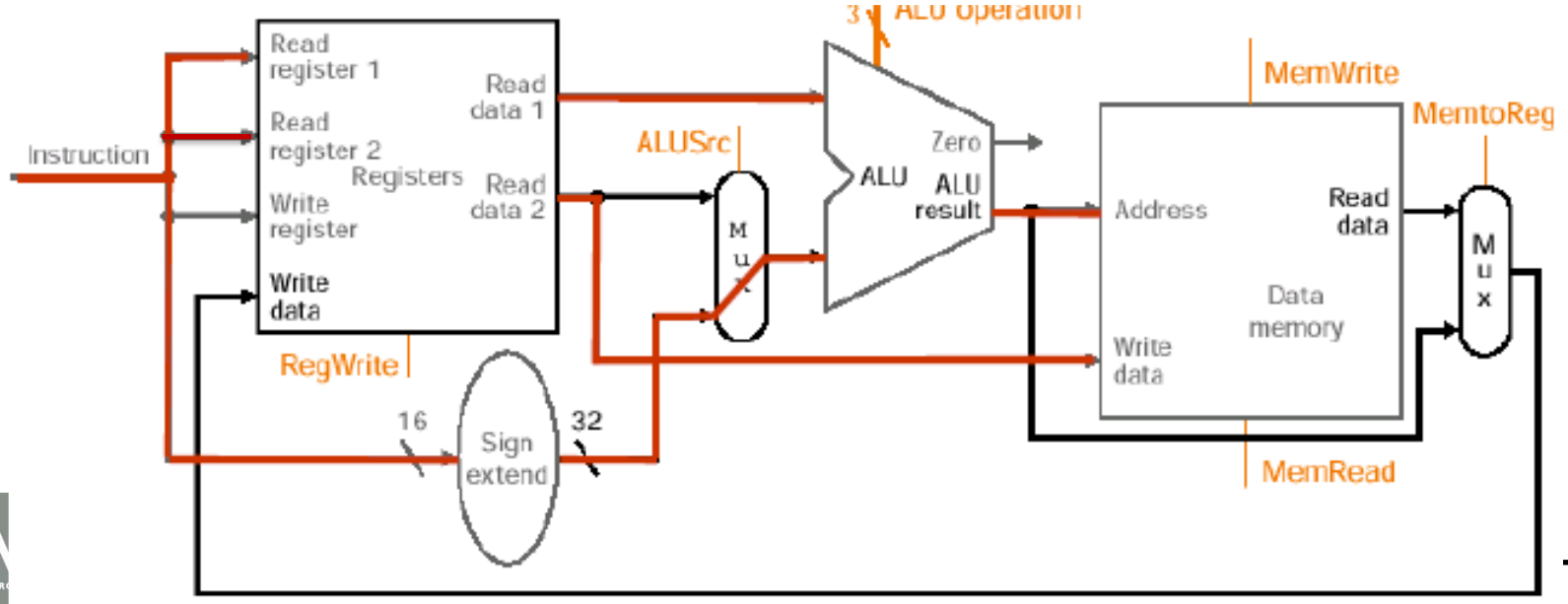
Execute Load

- LW instruction
  - RF: Read rs and Write to rt
  - ALU: rs data, sign-extend to input, Op is add, output to memory address
  - Memory: read, data to RF write data



Execute store

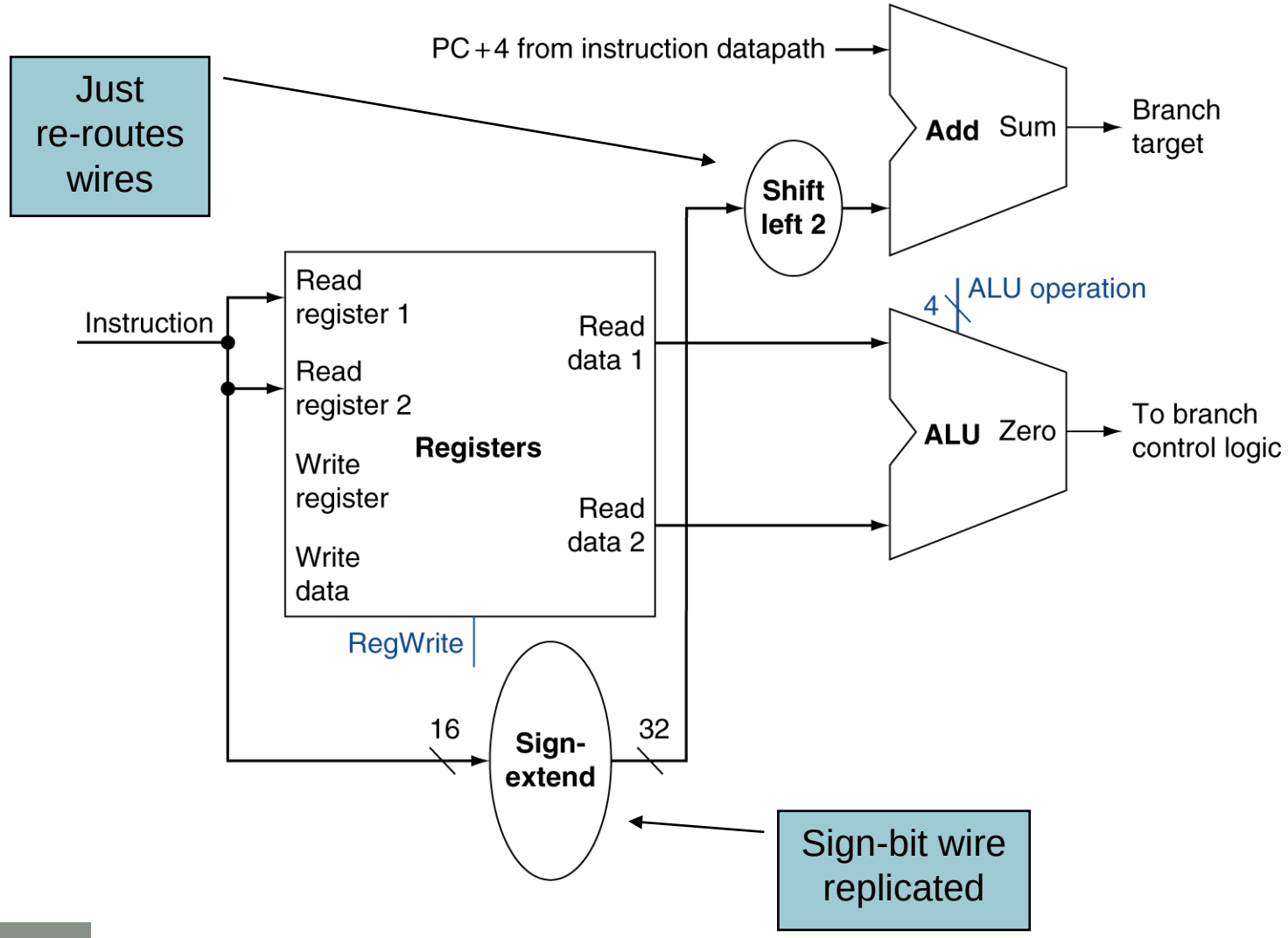
- SW instruction
  - RF: Read rs and rt
  - ALU: rs data, sign-extend to input, Op is add, output to memory address
  - Memory: write, data from RF read data 2 (rt)



Branch Instructions (beq, bne)

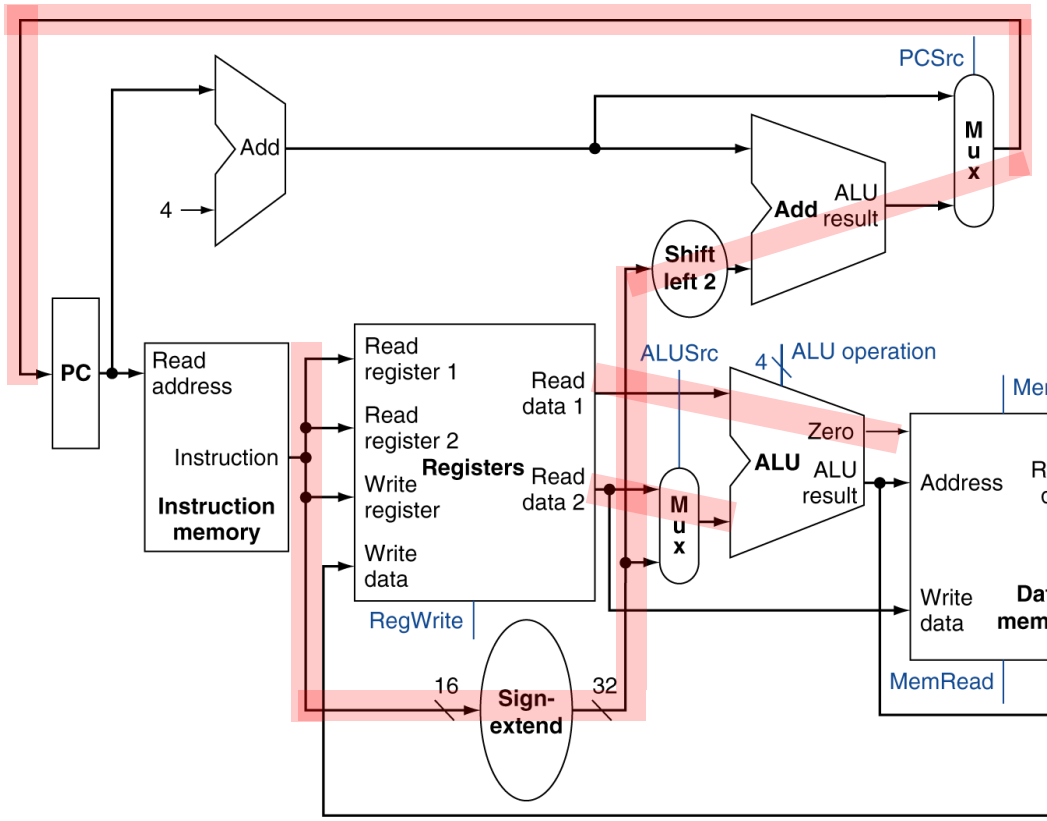
- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places (word displacement)
  - Add to PC + 4
    - Already calculated by instruction fetch

Branch Instructions

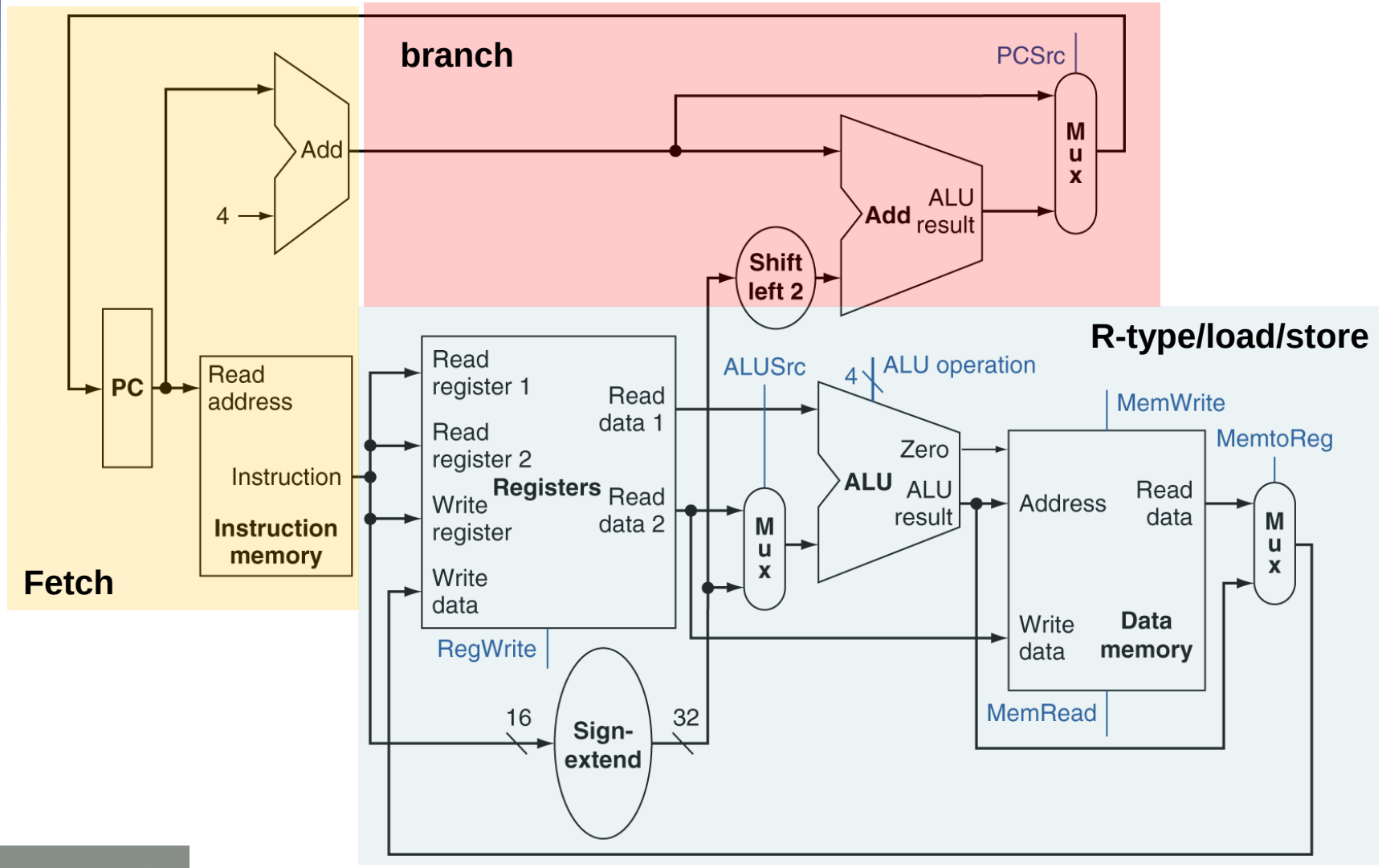


Execute beq

- Beq instruction
  - RF: Read rs and rt
  - ALU: rs and rt data, Op is sub, Zero output to branch mux
  - PC adder: sign-extend -> shift left 2



Full Datapath

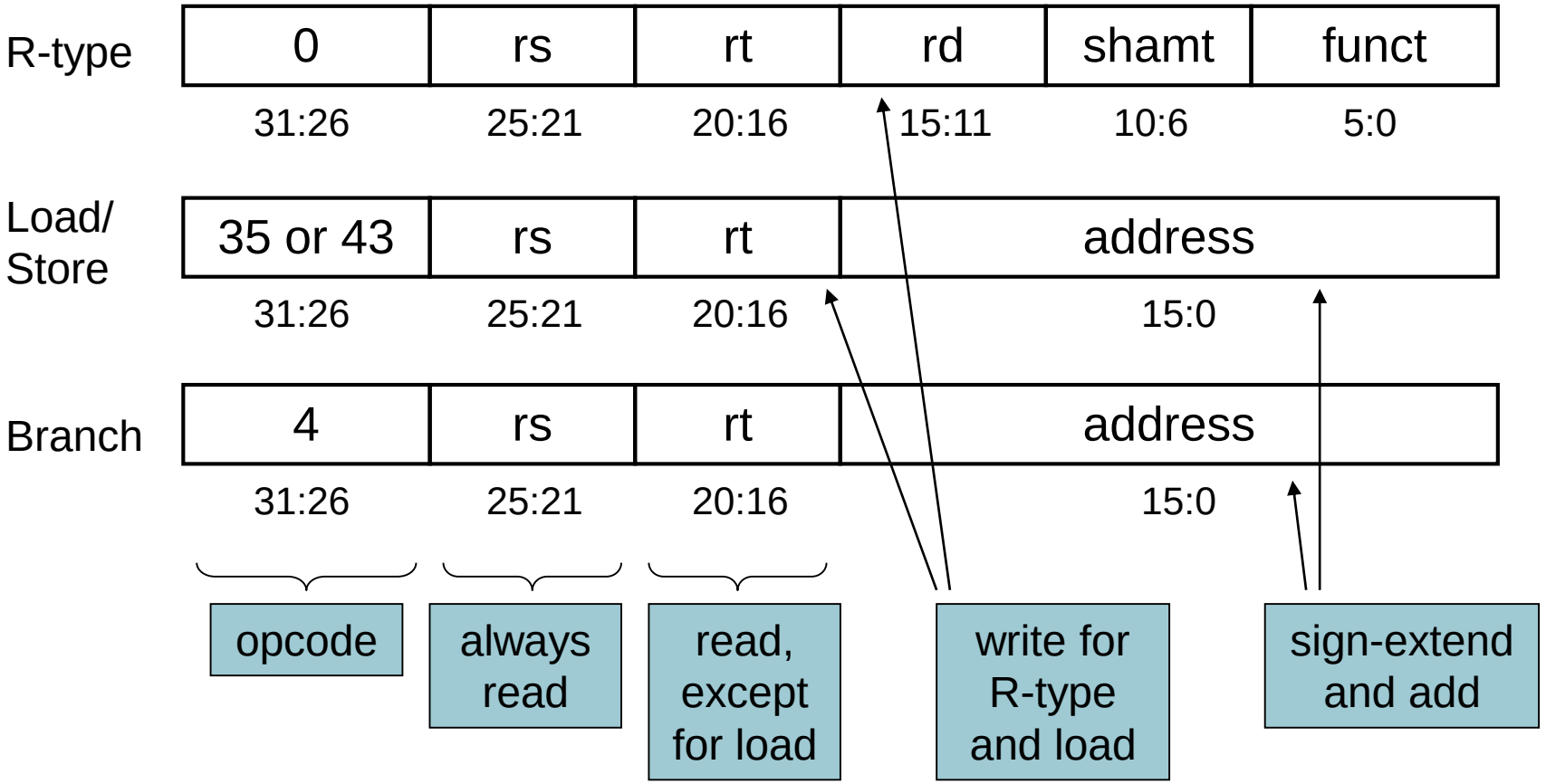


Agenda

- CPU Design
- MIPS Datapath (single-cycle)
- Control unit
- Adding jump instruction

Instruction format

- Control signals derived from instruction



Control Unit

- Decodes instruction to determine what segments will be active in the datapath
- Inputs are from each instruction's
  - op-code field [31:26] for all instructions
  - funct field [5:0] for R-type instructions.
- Outputs are the control lines to control
  - ALU, Multiplexer selectors, Register file, Data Memory
- The control unit includes mainly
  - ALU control unit + Main control unit

ALU Control

- ALU used for
  - Load/Store: F = add (base address + offset)
  - Branch (beq): F = subtract (compare rs, rt)
  - R-type: F depends on funct field
- Remember the ALU control lines:

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
lw	00
sw	00
beq	01
R-type	10

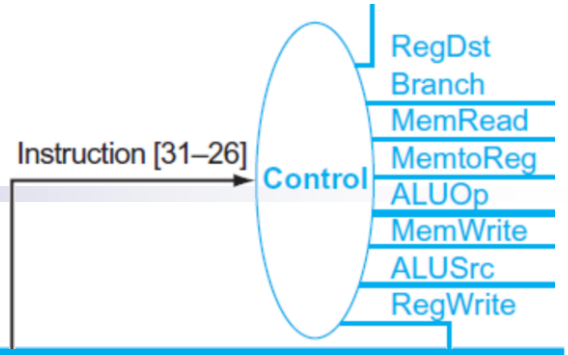
ALU Control

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

Main Control Unit

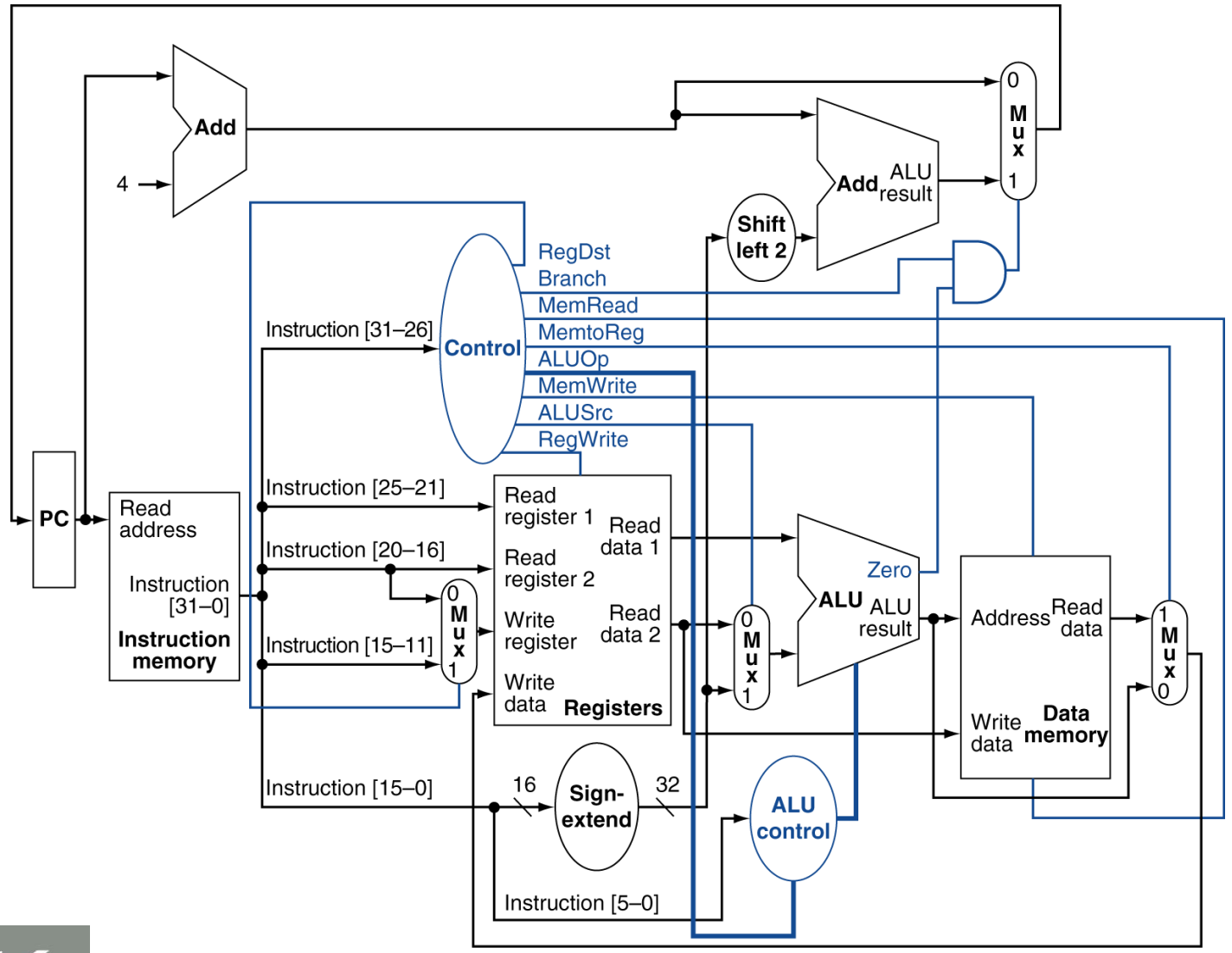
- **Inputs** are 6-bit op-code from all instructions
  - op-code field (6 bits [31:26])
- **Outputs** (9-bit) are the control lines to control
  - Memory (2 bits)
    - MemRead, MemWrite
  - Multiplexers (4 bits)
    - RegDst, Branch, MemtoReg, ALUSrc,
  - Register File (1 bit)
    - RegWrite
  - 2-bit ALUOp (2 bits)
    - ALUOp0, ALUOp1

Control Signal



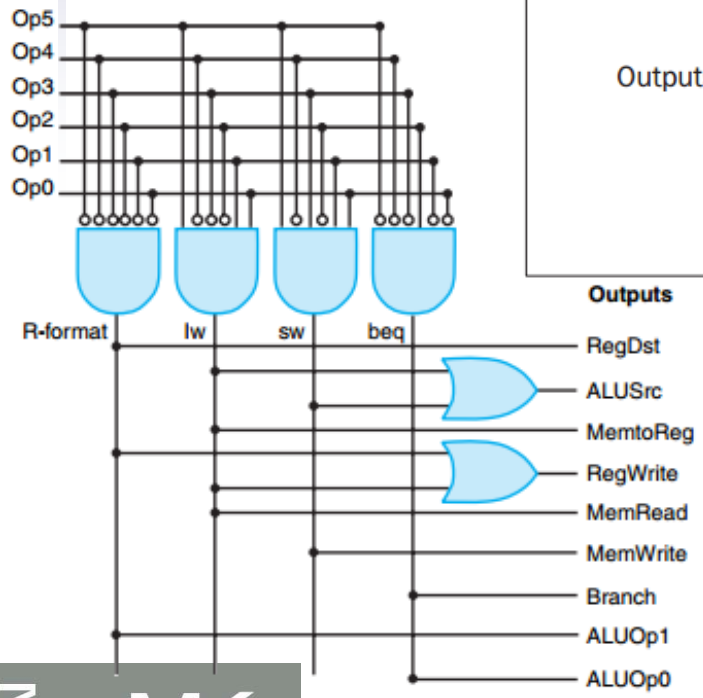
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Datapath With Control



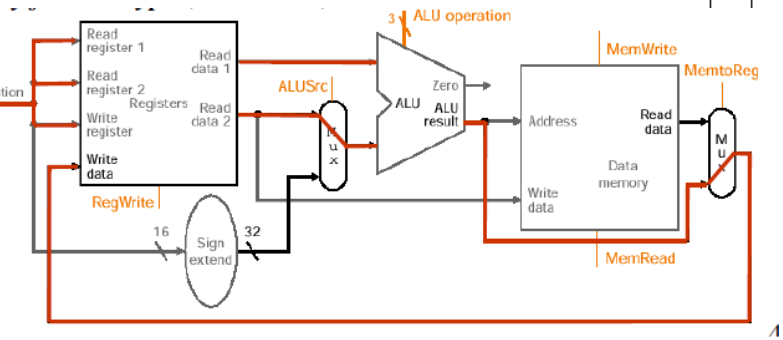
Control Unit Example

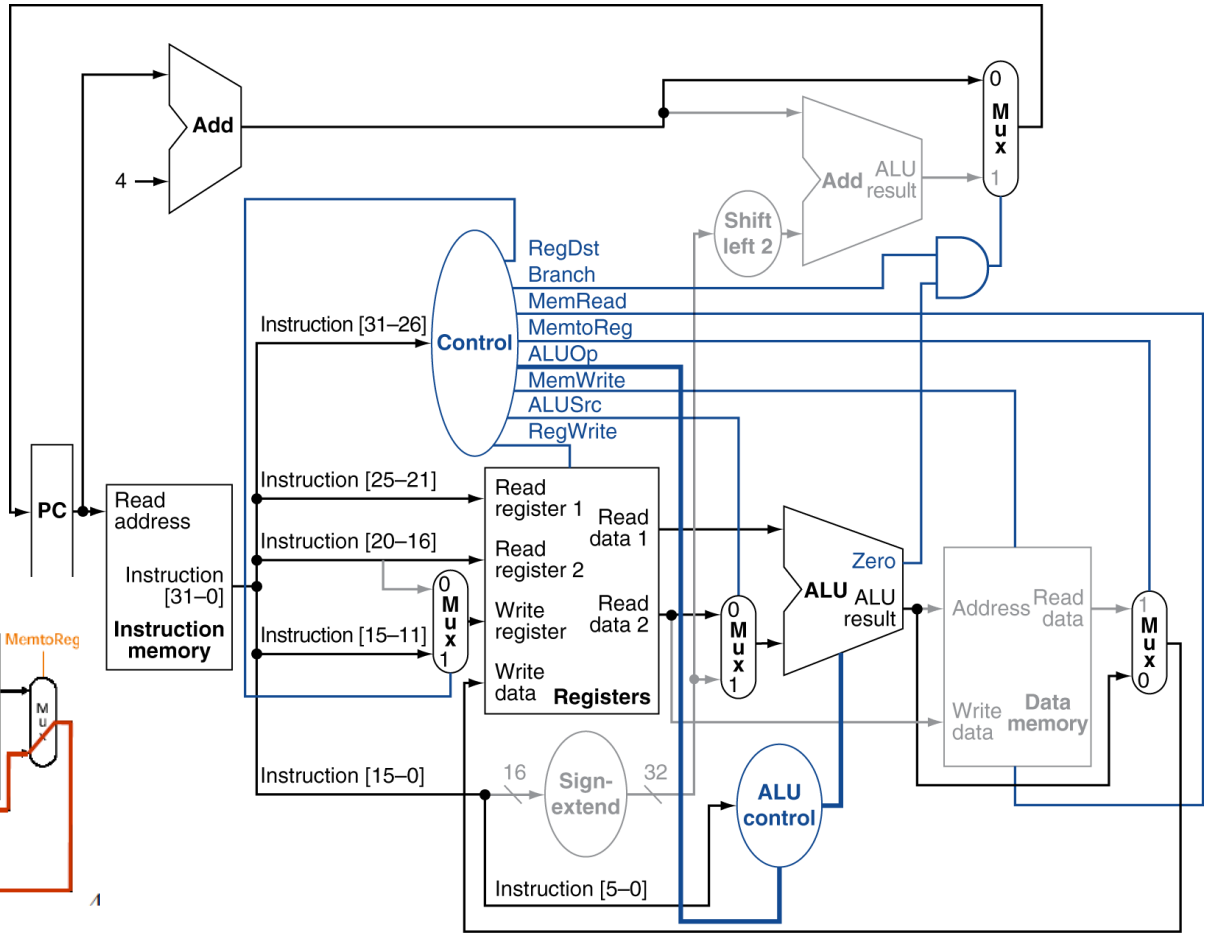
Control	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1



R-Type Instruction

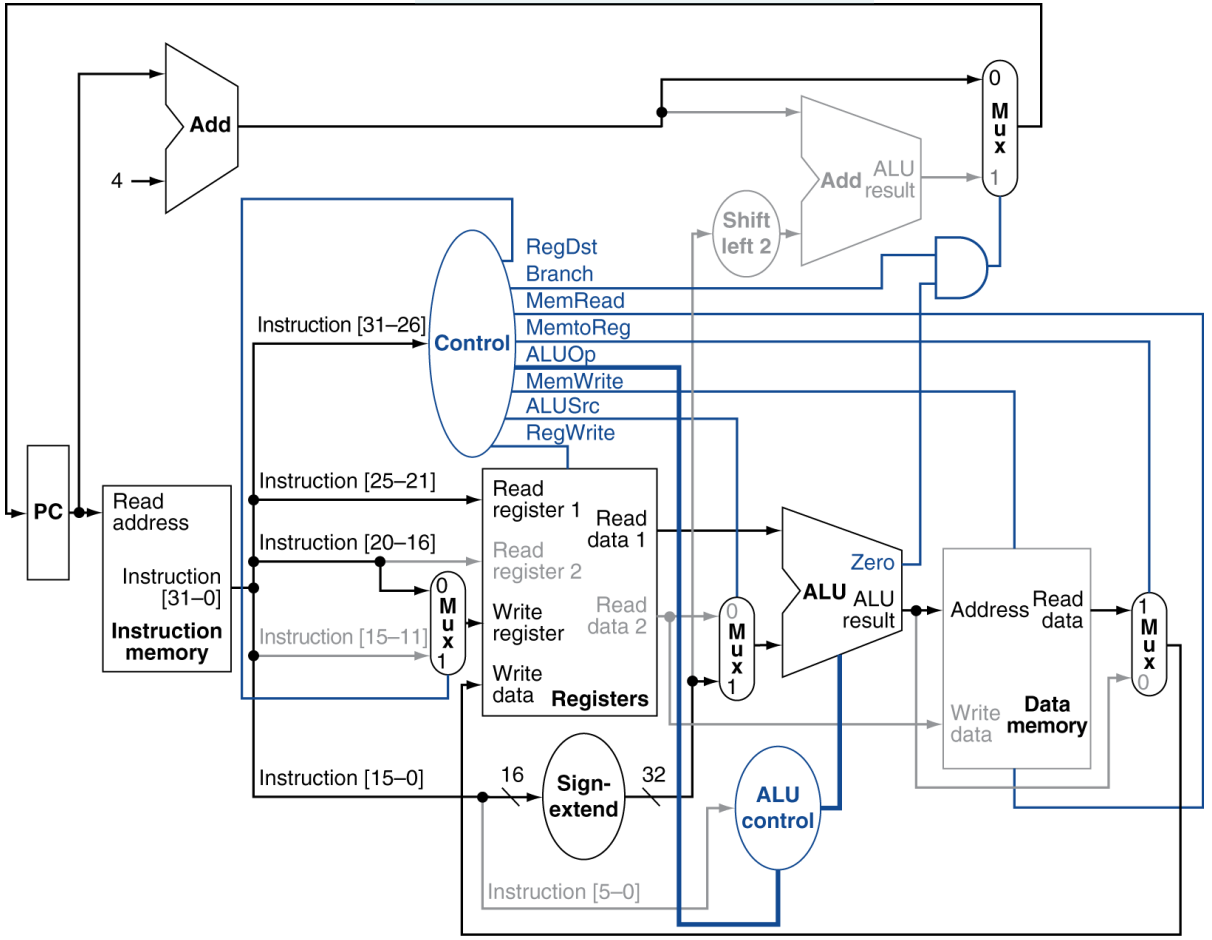
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0





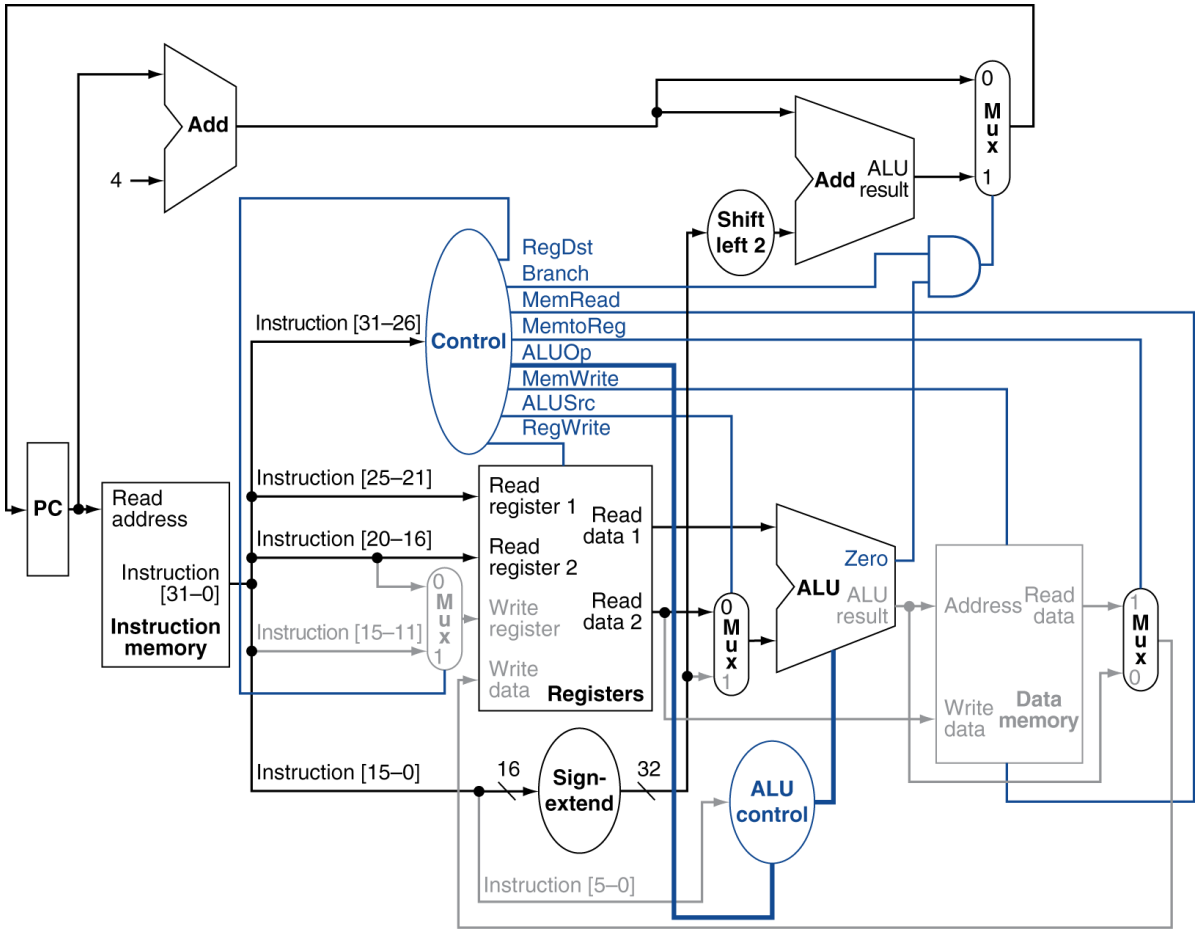
Load Instruction

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0



Branch-on-Equal Instruction

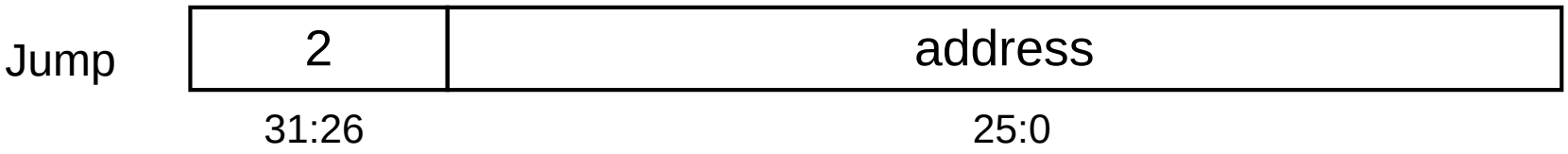
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
beq	X	0	X	0	0	0	1	0	1



Agenda

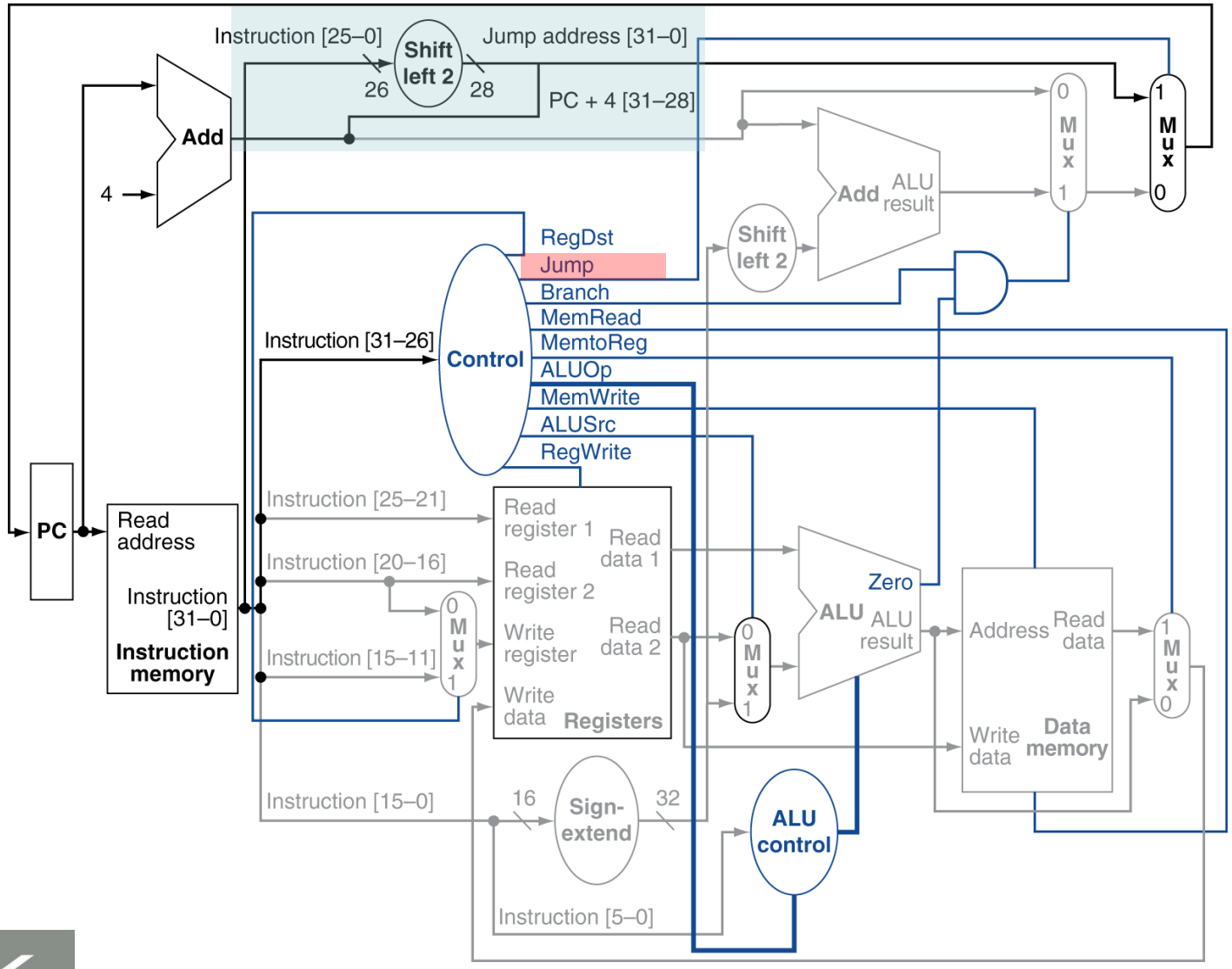
- CPU Design
- MIPS Datapath (single-cycle)
- Control unit
- Adding jump instruction

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



Reading

- Chapter 4: 4.3 – 4.5
- Exercises: 4.1 – 4.8