

The Processor

Computer Organization

Sunday, 09 October 2022

Many slides adapted from:
Computer Organization and Design,
Patterson & Hennessy
5th Edition, © 2014, MK
and from Prof. Mary Jane Irwin, PSU



TÉCNICO LISBOA

Summary

- Previous Class
 - Virtual Memory
 - Translation Lookaside Buffer (TLB)
 - Integrated Operation of the Memory System
- Today:
 - The MIPS Architecture
 - Pipeline

Review: MIPS (RISC) Design Principles

- Simplicity favors regularity
 - fixed size instructions
 - small number of instruction formats
 - opcode always the first 6 bits
- Smaller is faster
 - limited instruction set
 - limited number of registers in register file
 - limited number of addressing modes
- Make the common case fast
 - arithmetic operands from the register file (load-store machine)
 - allow instructions to contain immediate operands
- Good design demands good compromises
 - three instruction formats

Introduction

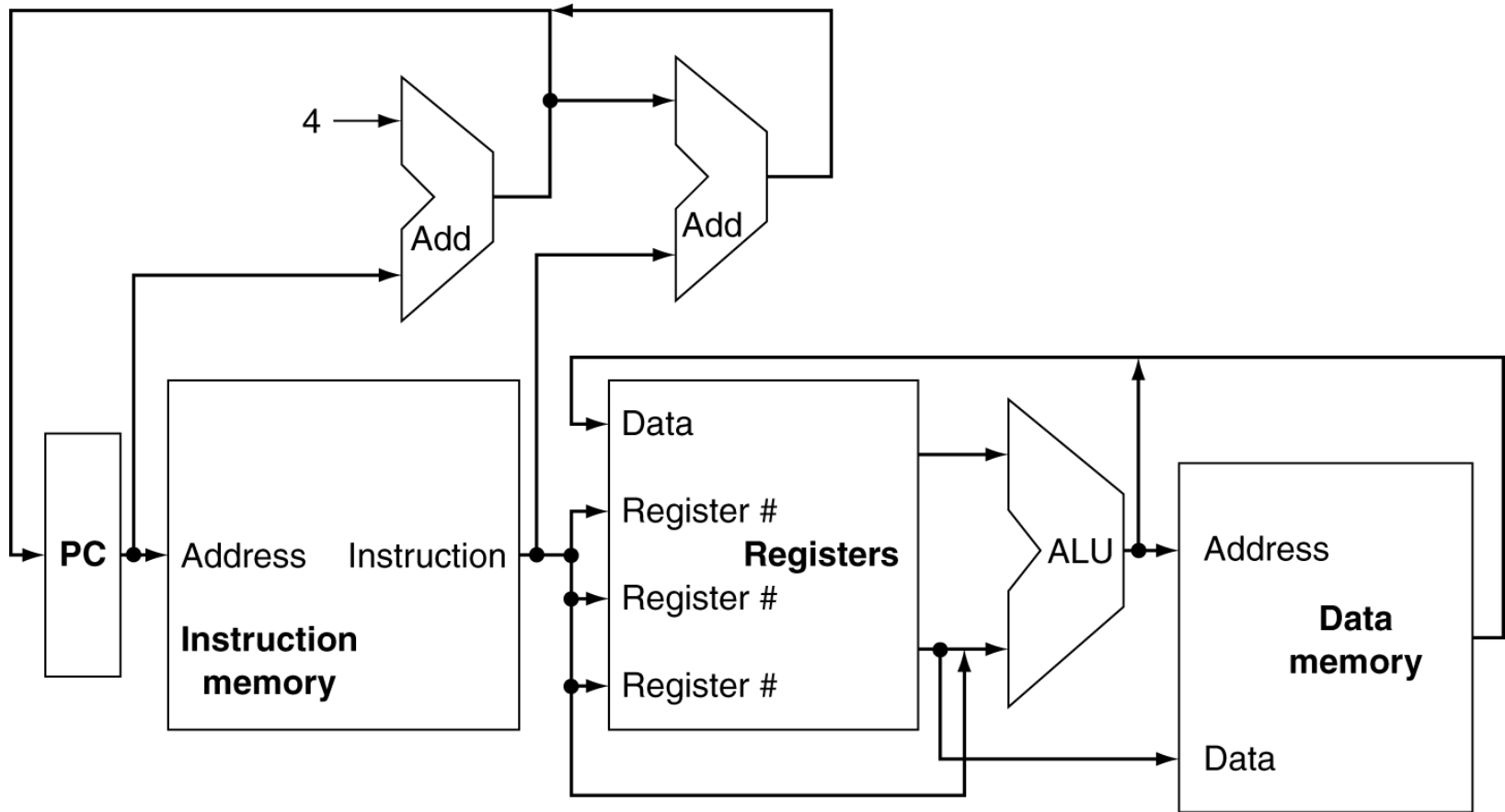
- CPU performance factors
 - Instruction count
 - Determined by ISA and compiler
 - CPI and Cycle time
 - Determined by CPU hardware
- We will examine two MIPS implementations
 - A simplified version
 - A more realistic pipelined version
- Simple subset, shows most aspects
 - Memory reference: `lw`,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

Instruction Execution

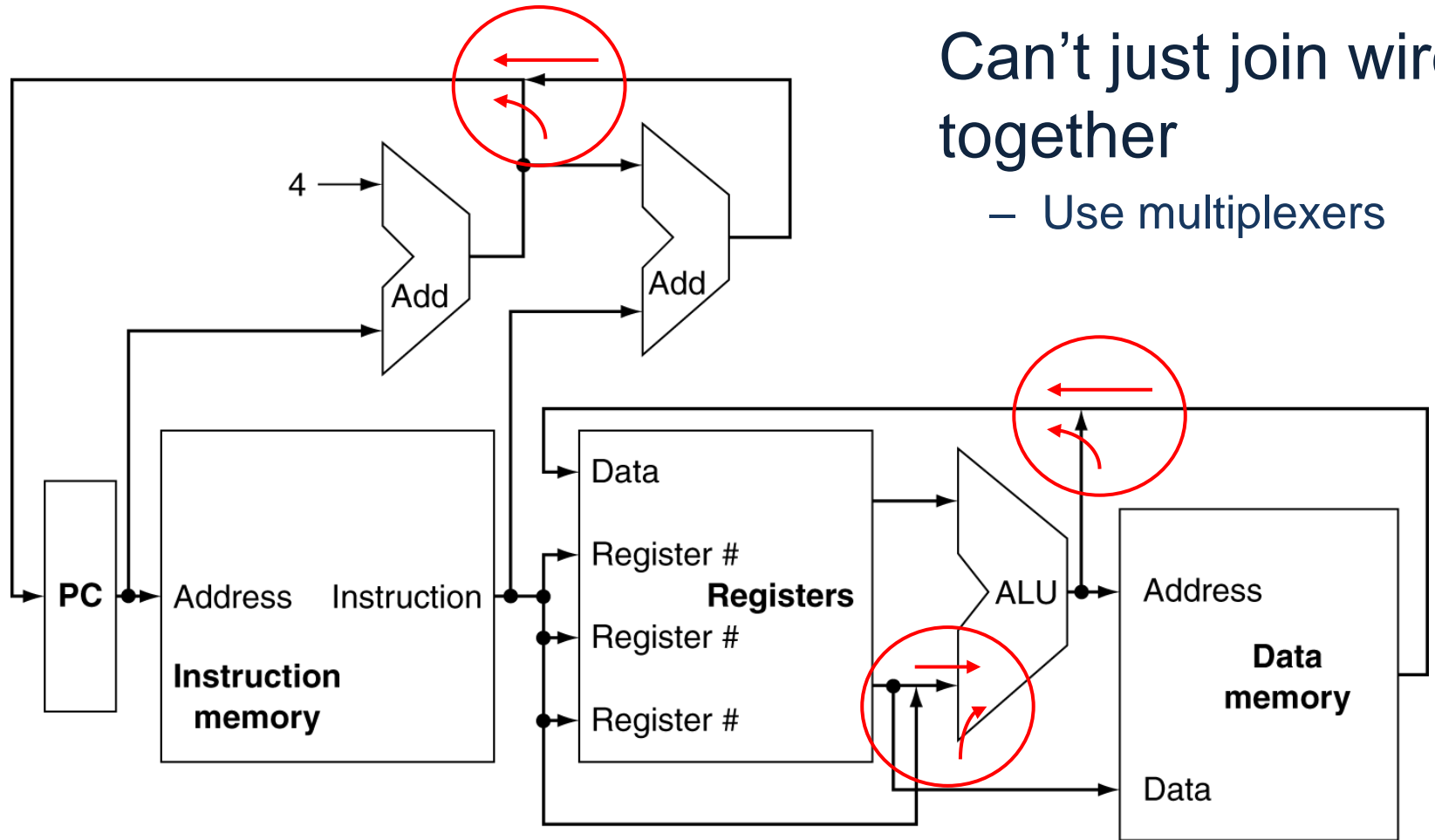
Generic implementation

- PC → instruction memory, fetch instruction
 - use the program counter (PC) to supply the instruction address and fetch the instruction from memory (and update the $PC = PC + 4$)
- Decode the instruction (and read registers)
 - Register numbers → register file, read registers
- Execute the instruction depending on instruction class
 - Use ALU to calculate
 - Arithmetic result
 - Memory address for load/store
 - Branch target address
 - All instructions (except `j`) use the ALU after reading the registers

CPU Overview



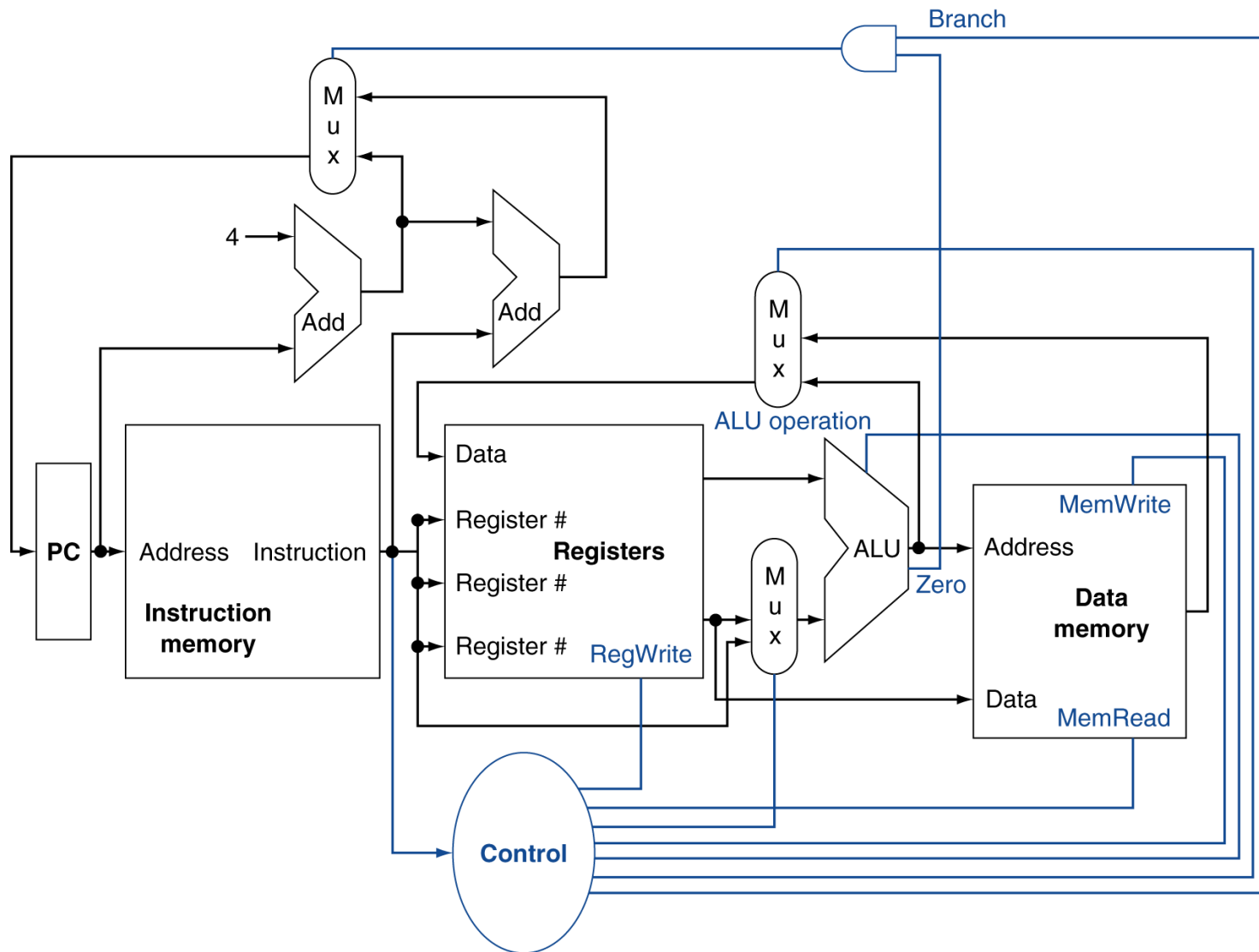
Multiplexers



Can't just join wires together

- Use multiplexers

Control



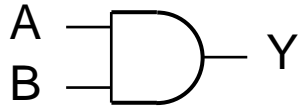
Logic Design Basics

- 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

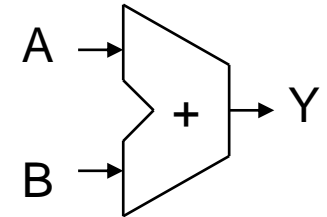
- AND-gate

$$Y = A \& B$$



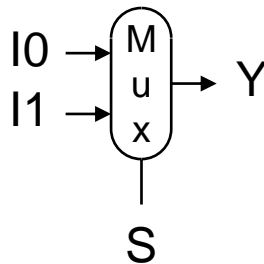
- Adder

$$Y = A + B$$



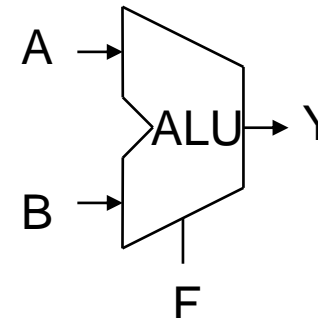
- Multiplexer

$$Y = S ? I1 : I0$$



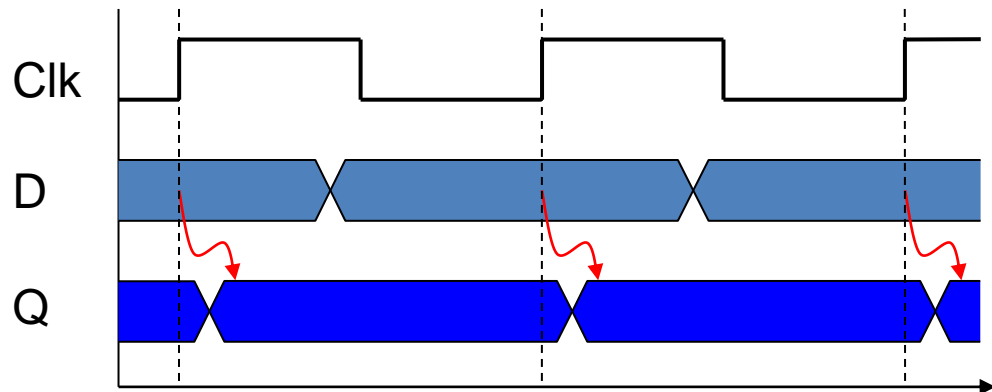
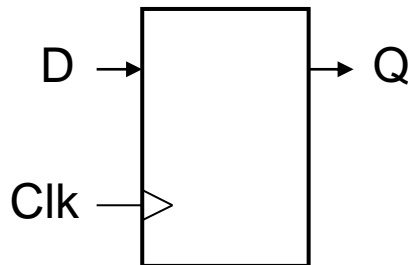
- Arithmetic/Logic Unit

$$Y = F(A, B)$$



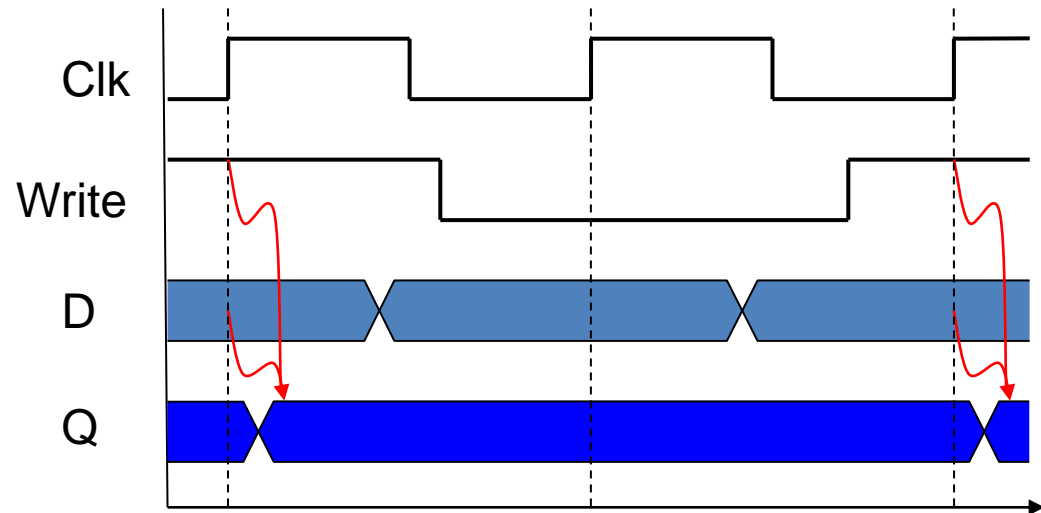
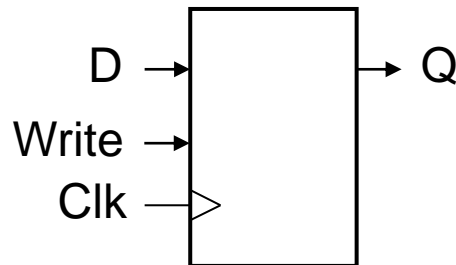
Sequential Elements

- 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



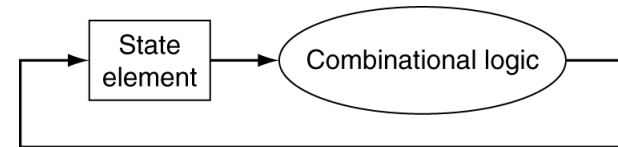
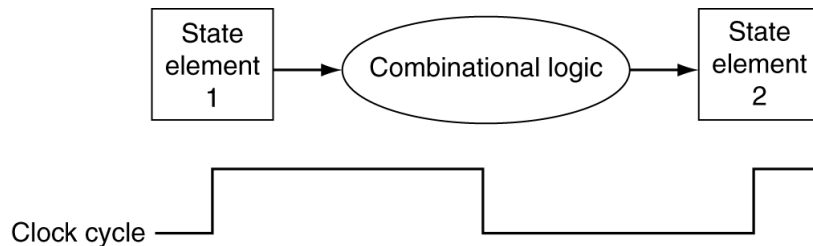
Sequential Elements

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



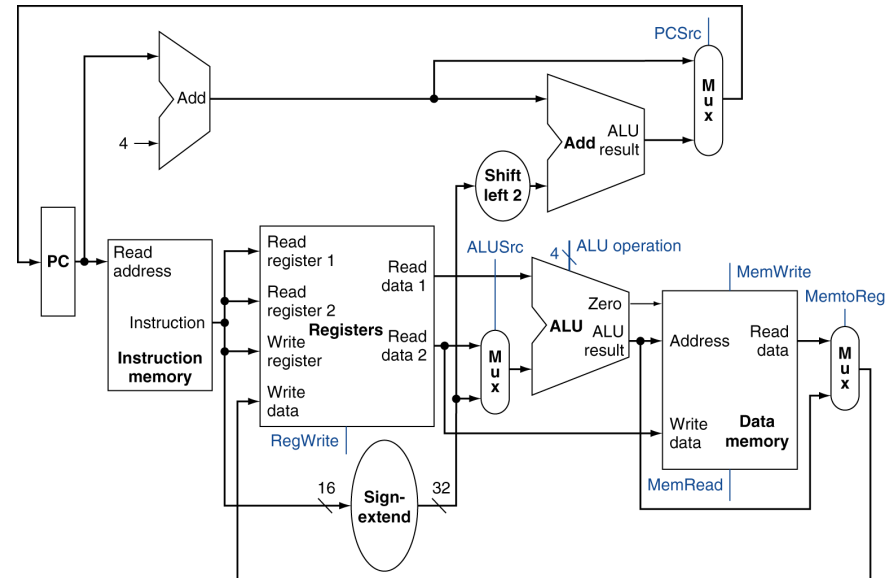
Clocking Methodology

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

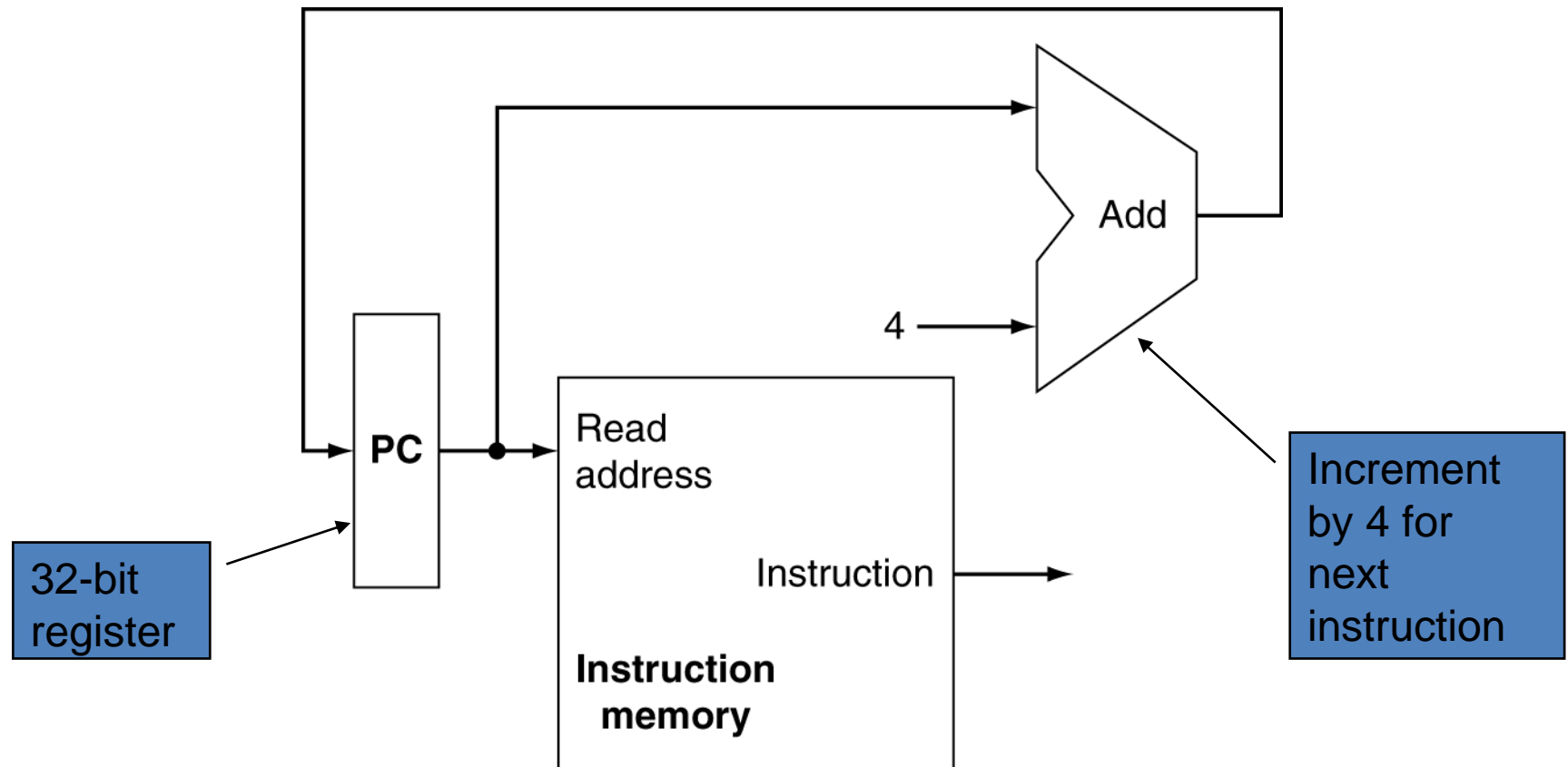


Building a Datapath

- 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
 - Instruction Decode
 - Instruction Execution
 - Instruction flow
 - Branch
 - Jump

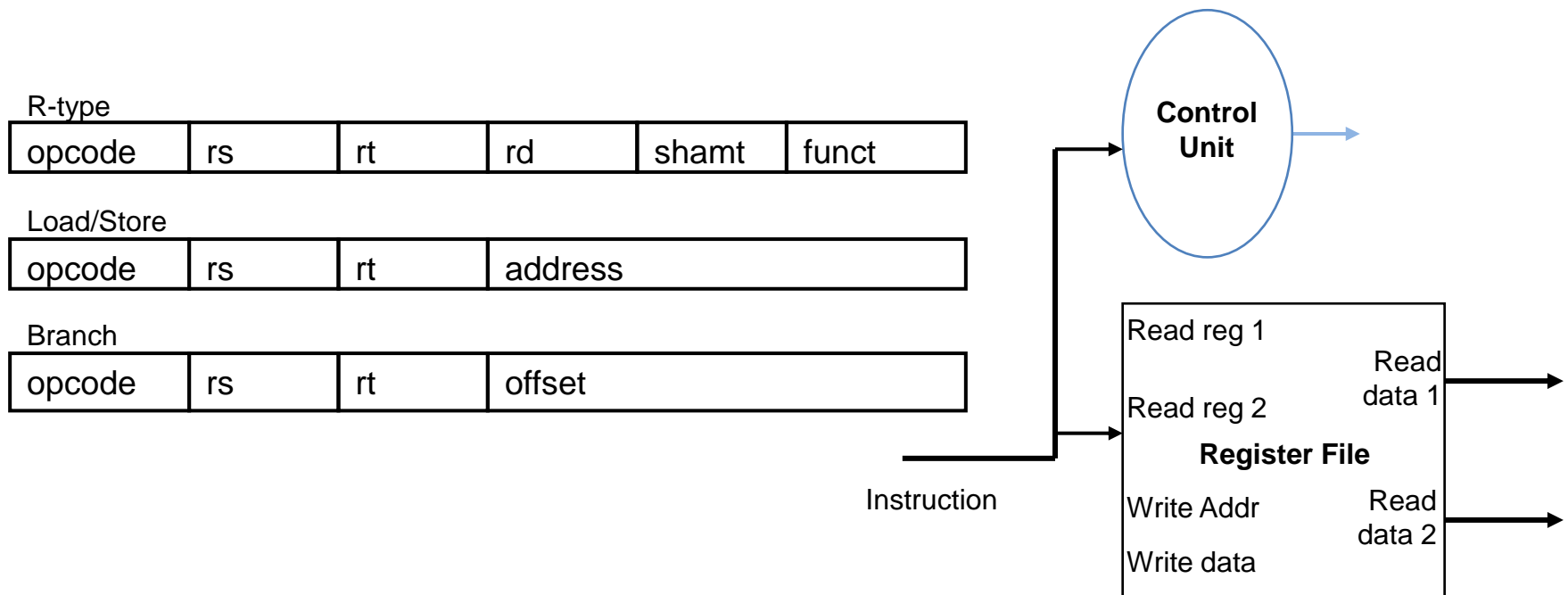


Instruction Fetch



Decoding Instructions

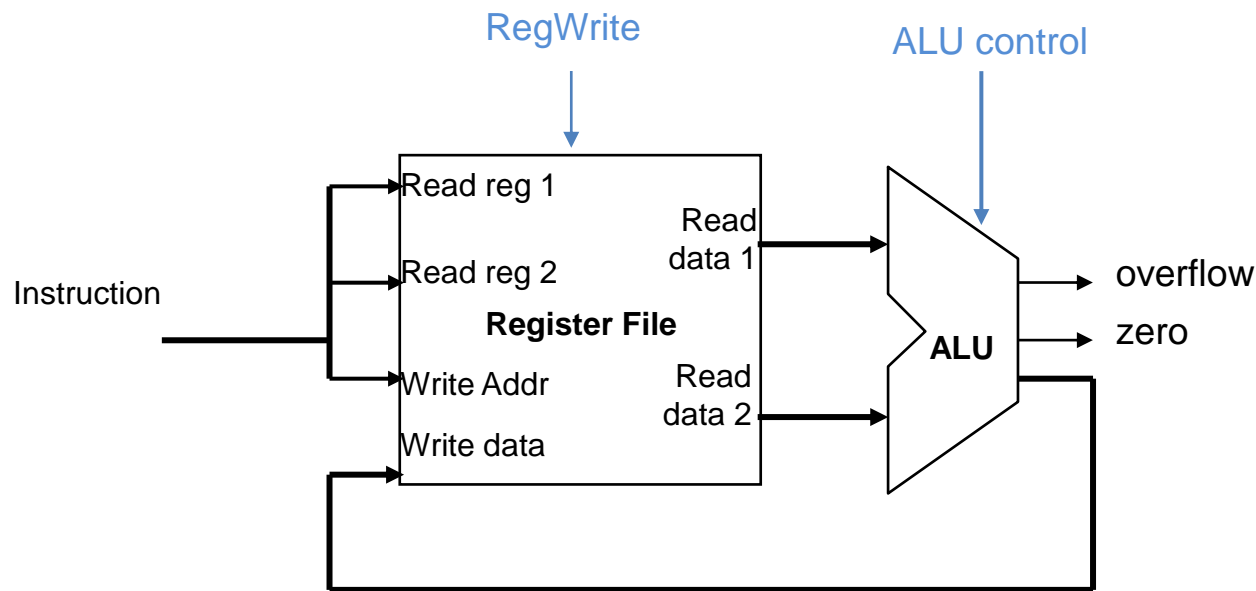
- Decoding instructions involves
 - sending the fetched instruction's opcode and function field bits to the control unit



- reading two values from the Register File
 - Register File addresses are contained in the instruction

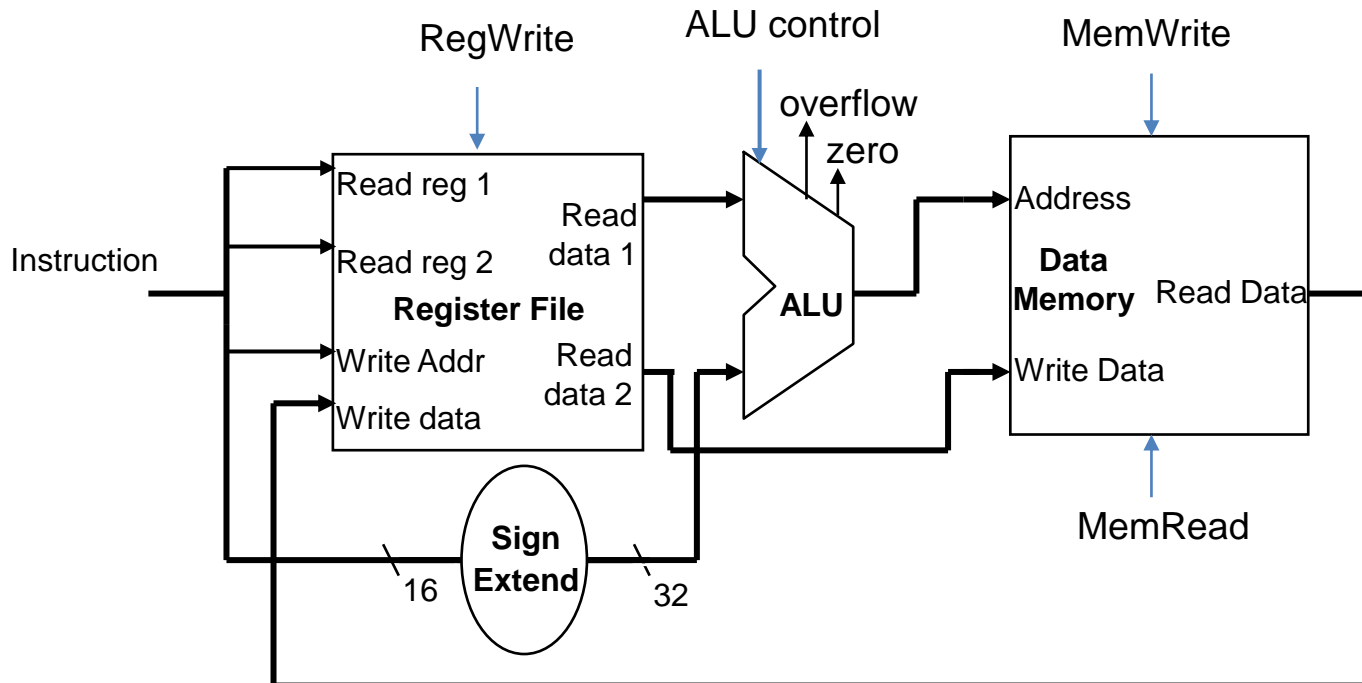
R-Format Instructions

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

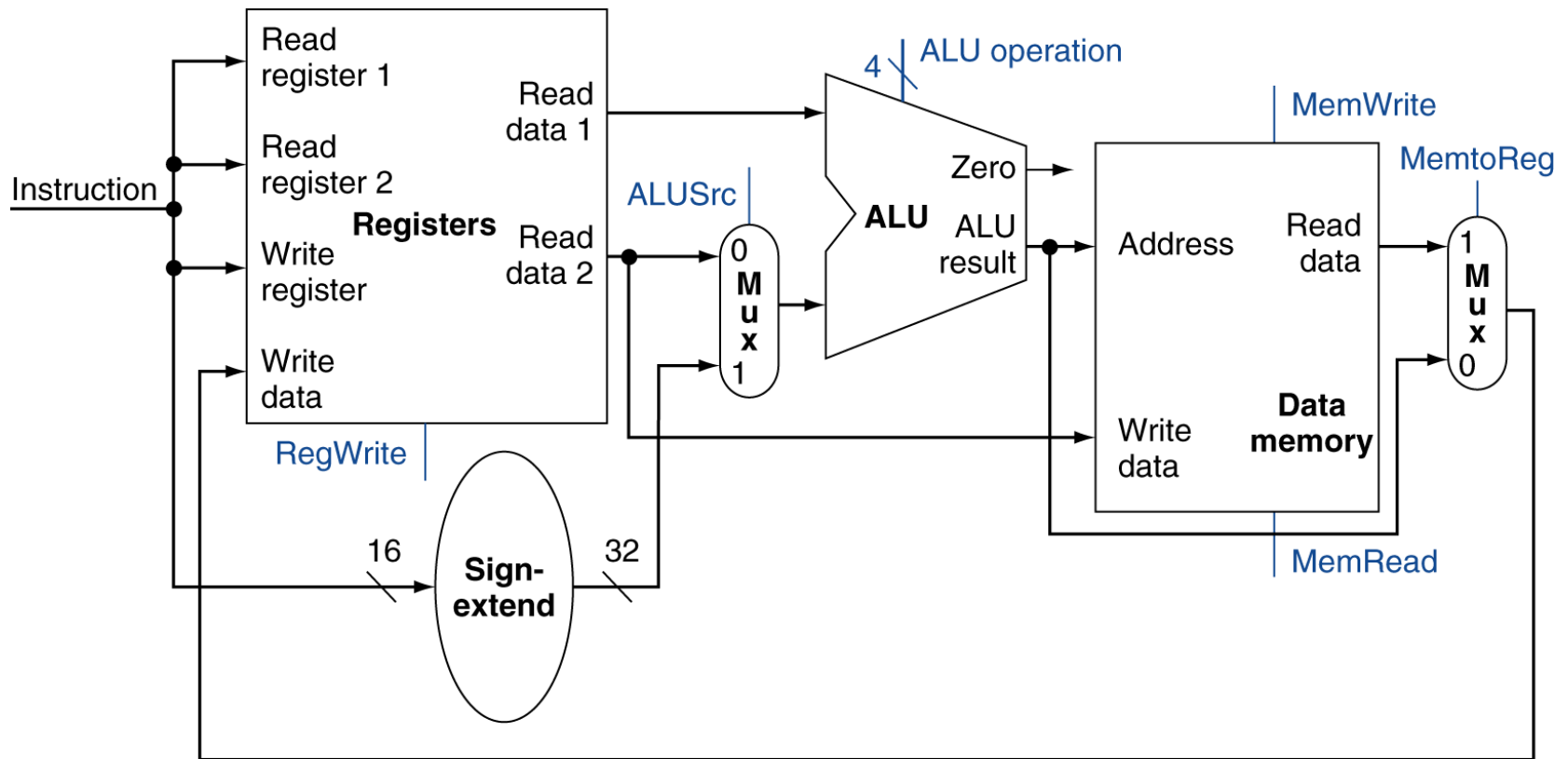


Load/Store Instructions

- Load and store operations involves
 - compute memory address by adding the base register to the 16-bit signed-extended offset field in the instruction
 - store value, from the Register File to the Data Memory
 - load value, from the Data Memory to the Register File



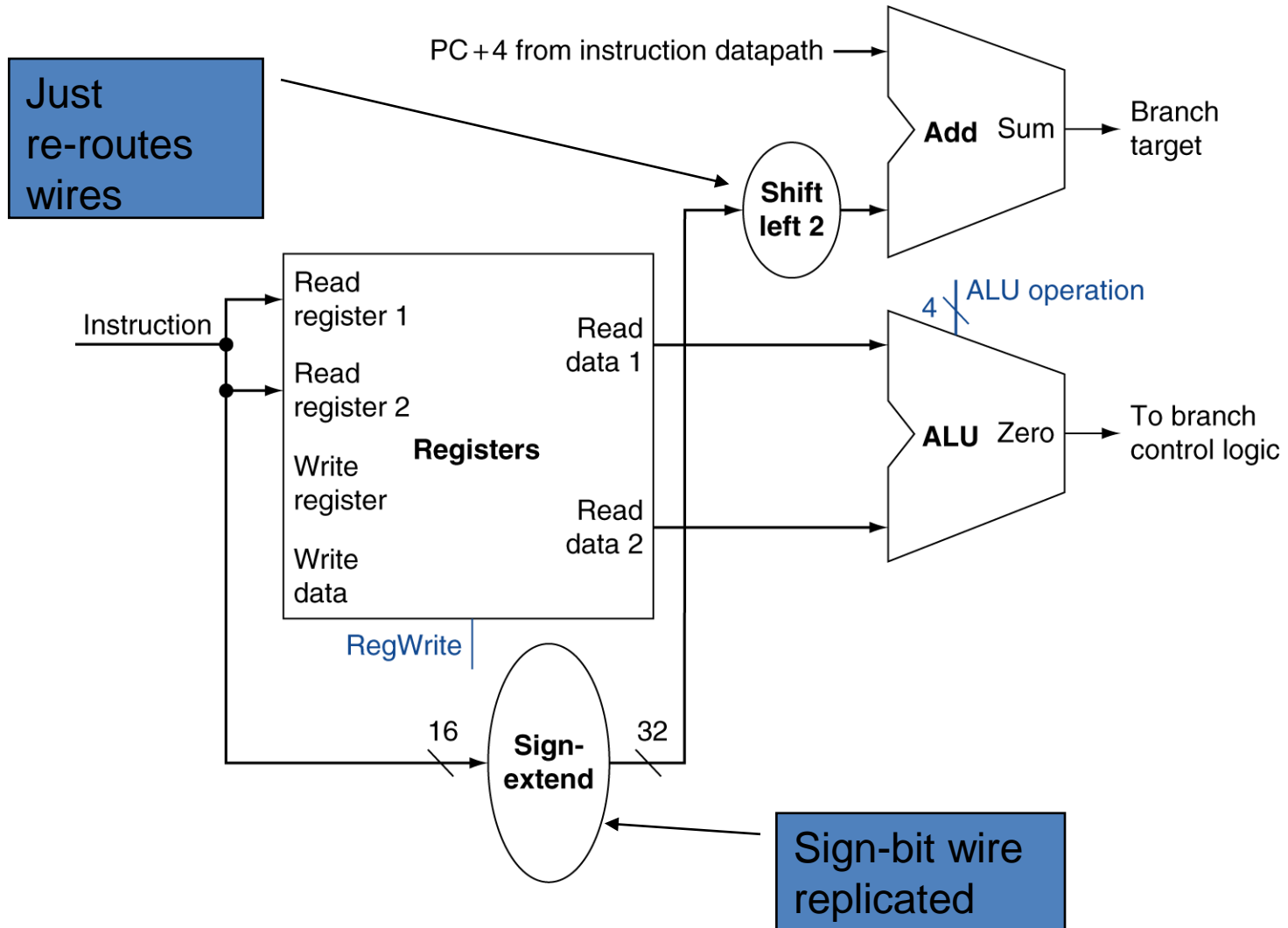
R-Type/Load/Store Datapath



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

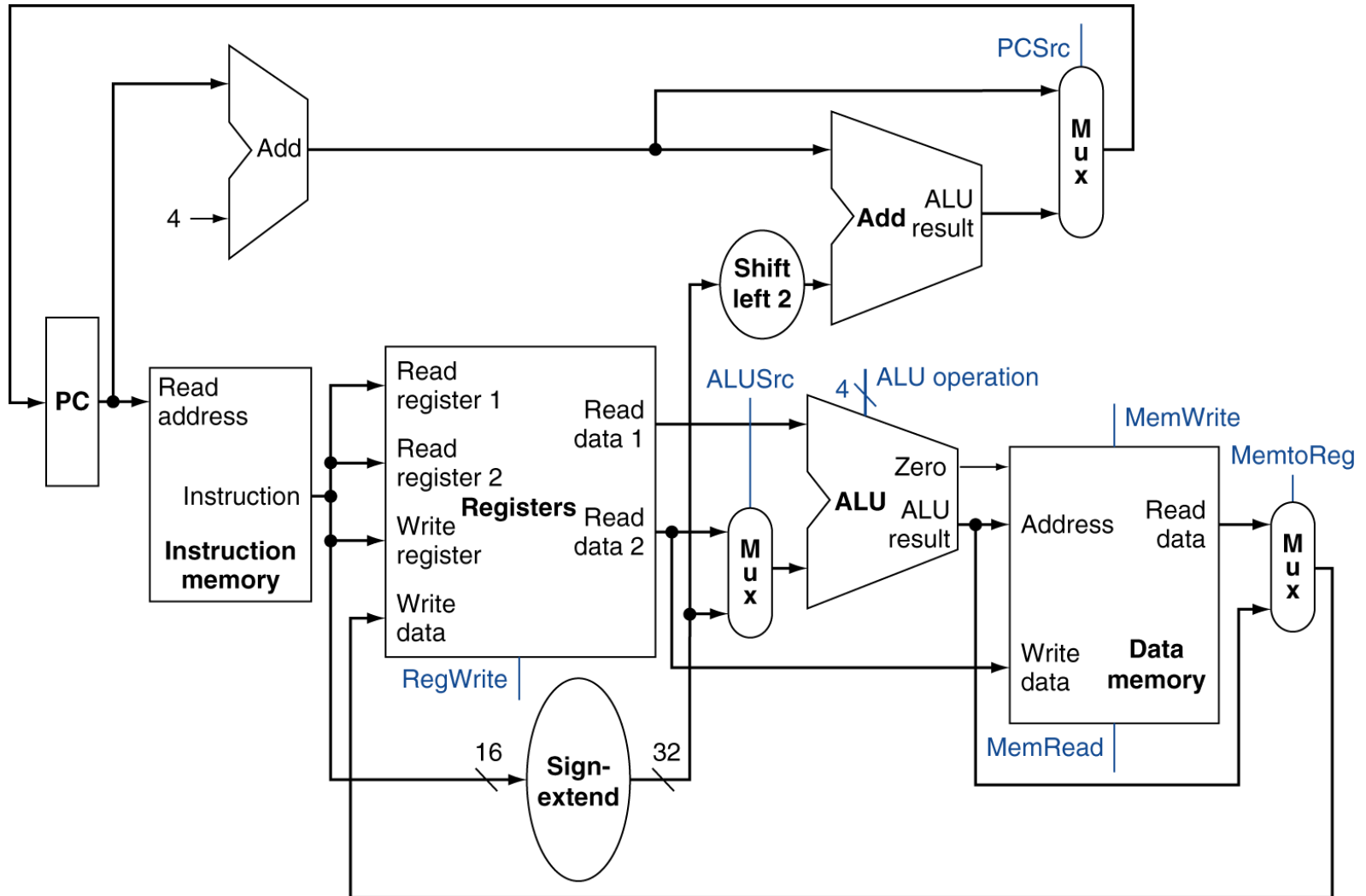
Branch Instructions



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

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

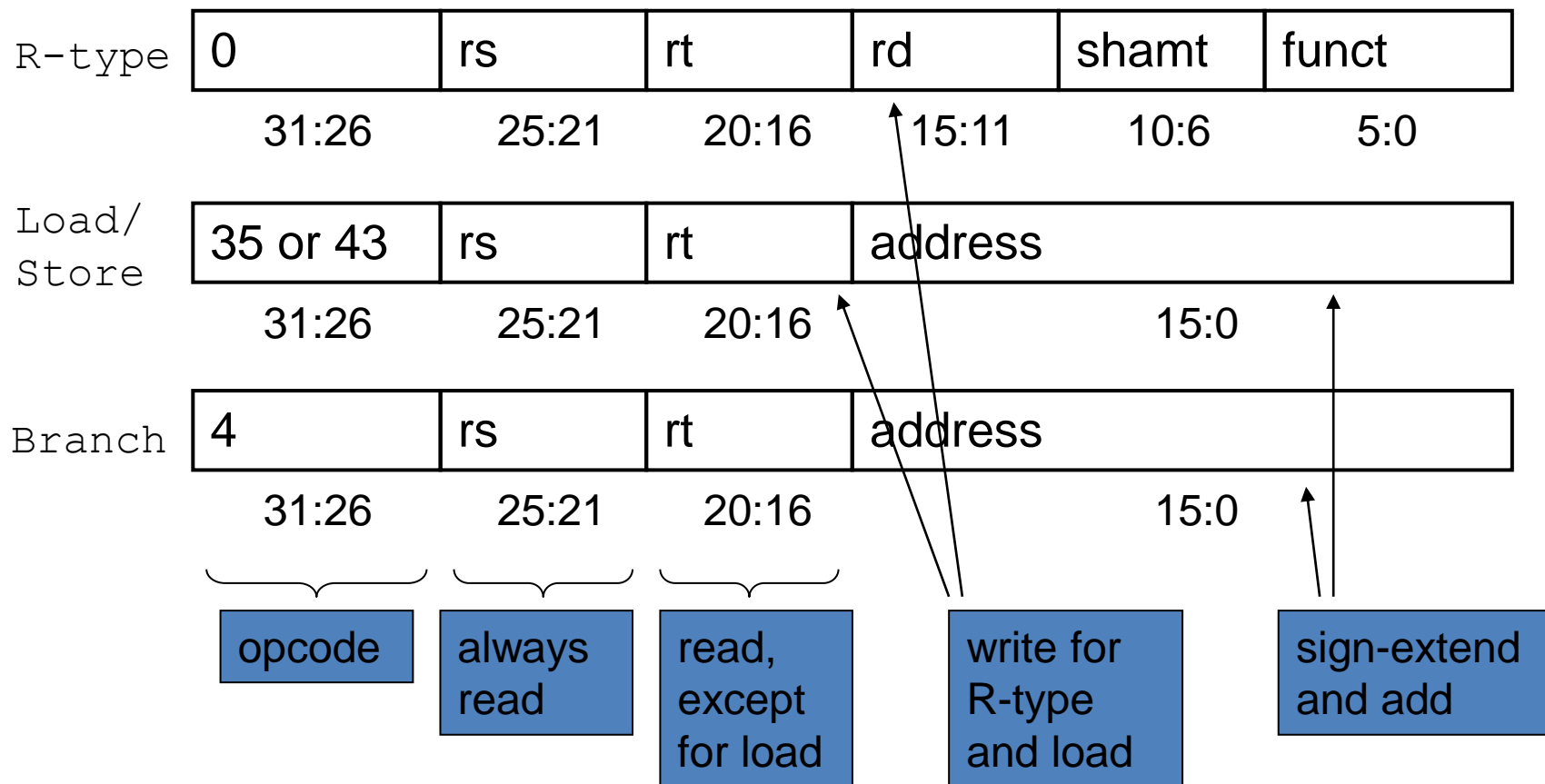
ALU Control

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

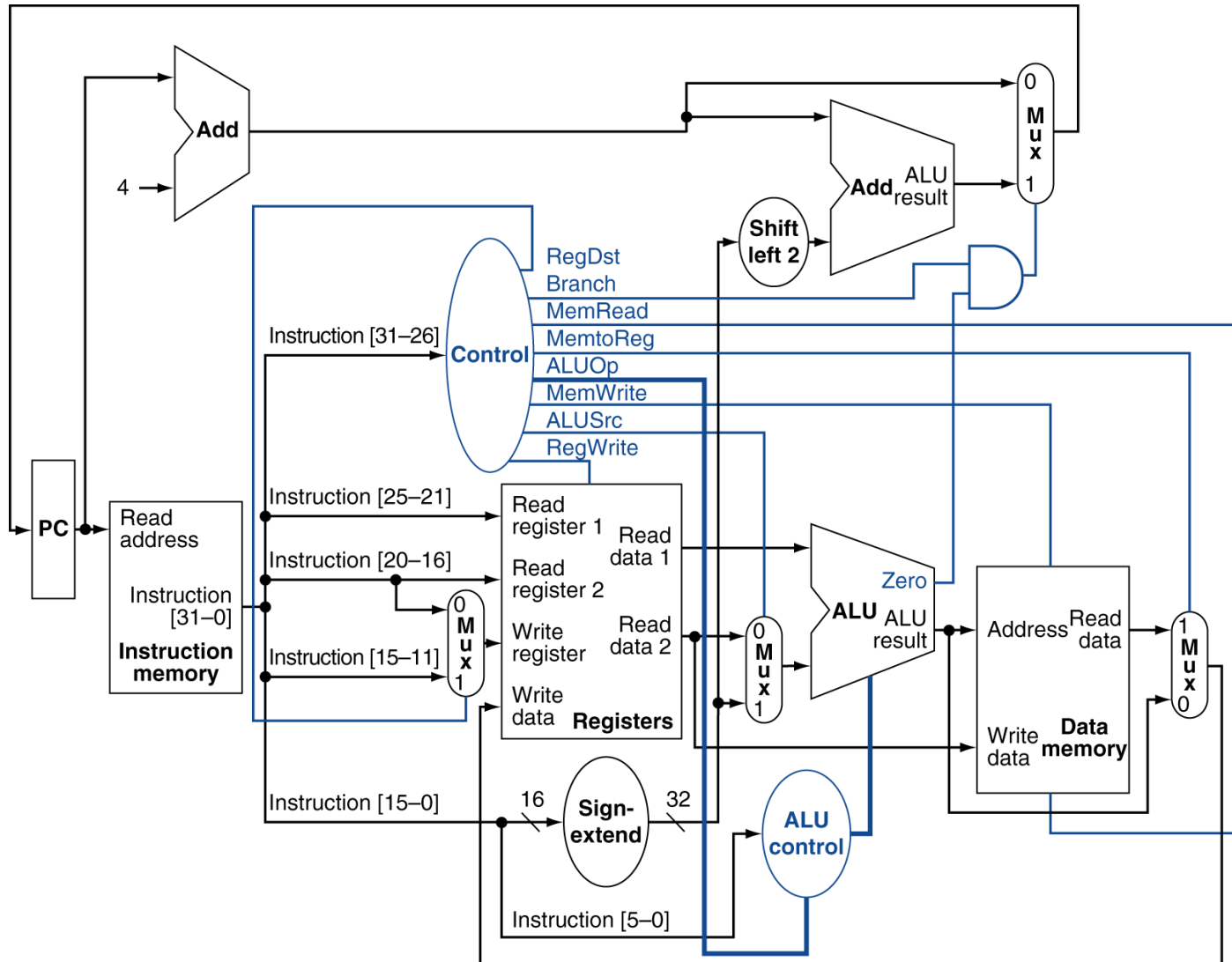
opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than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

The Main Control Unit

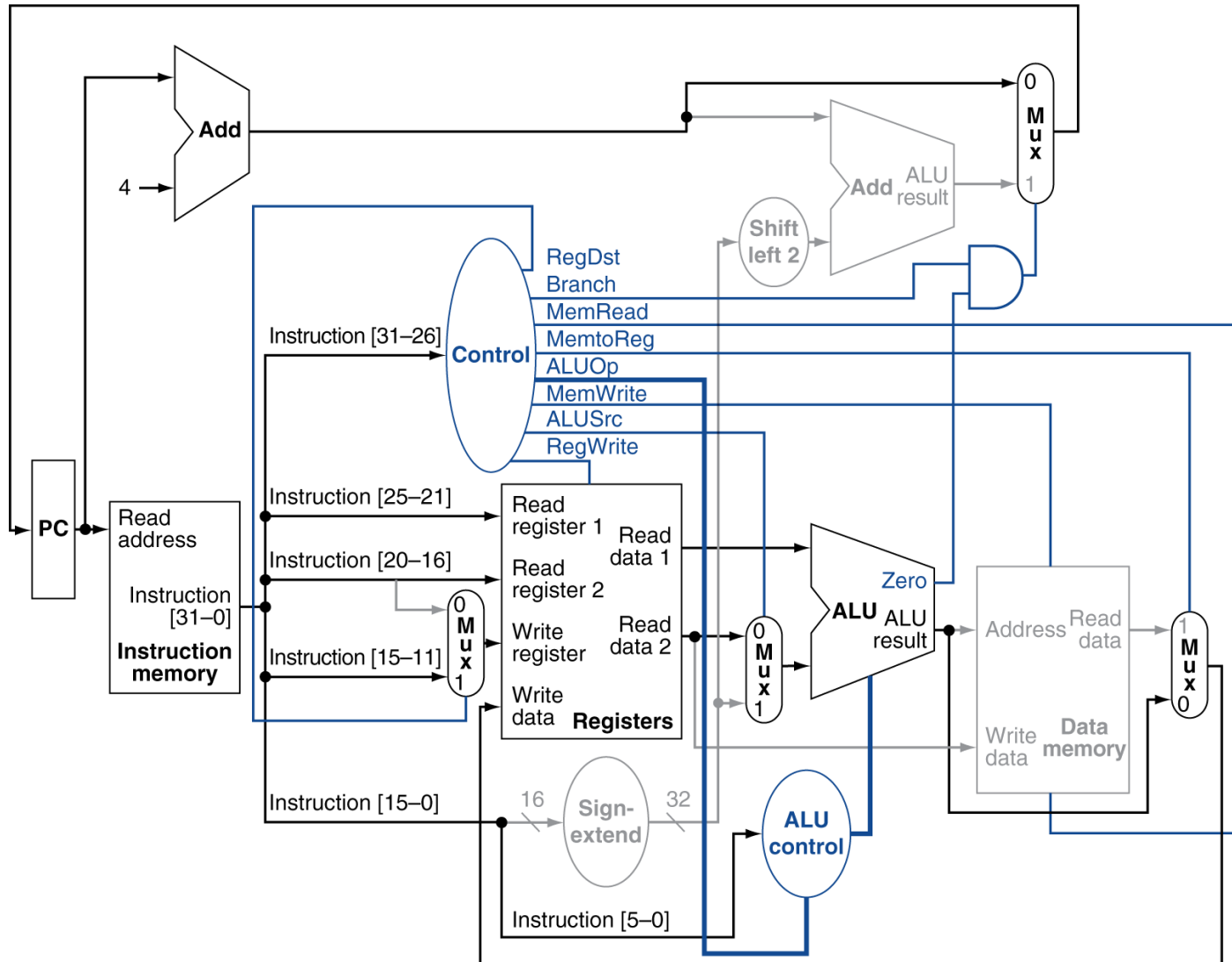
- Control signals derived from instruction



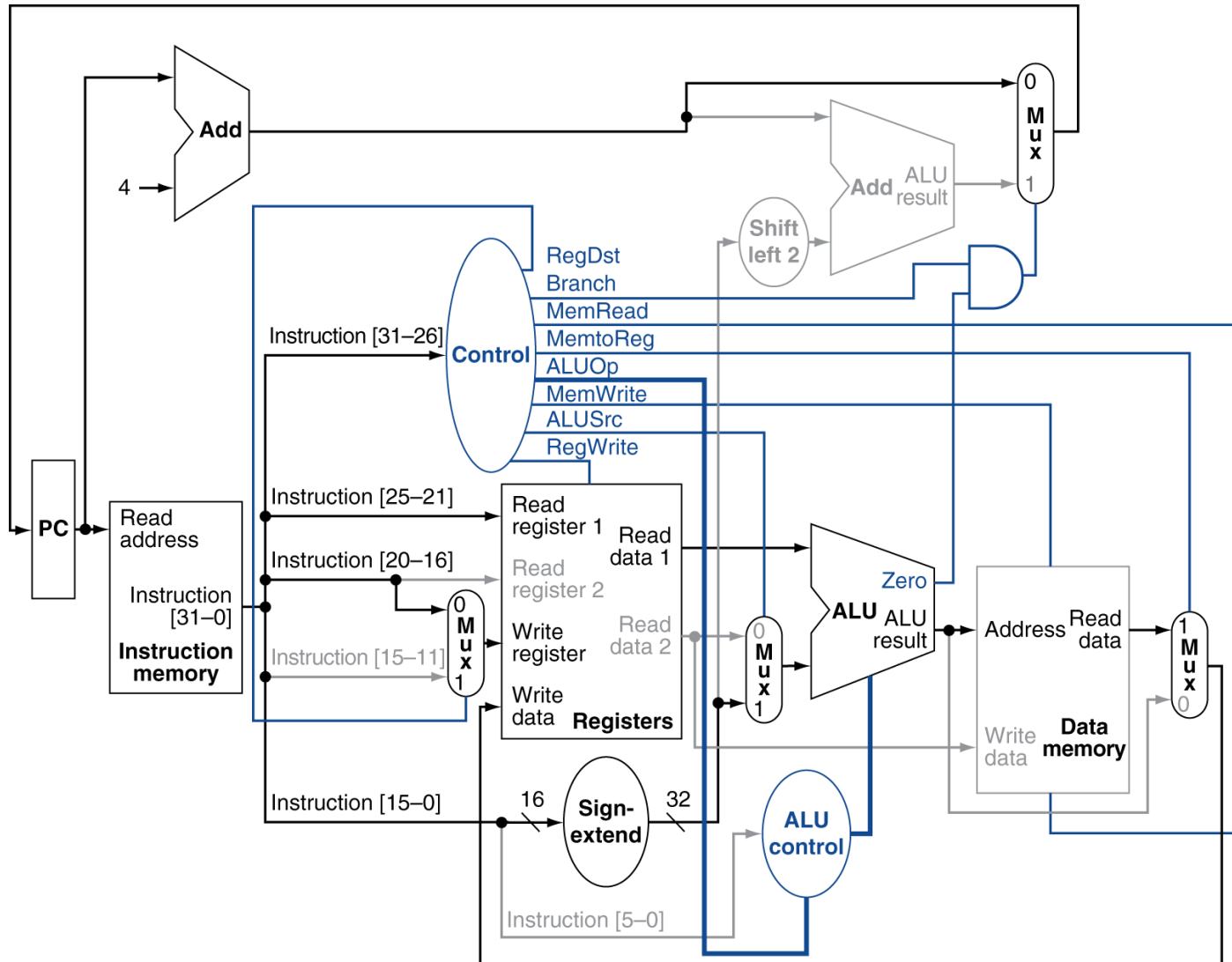
Datapath With Control



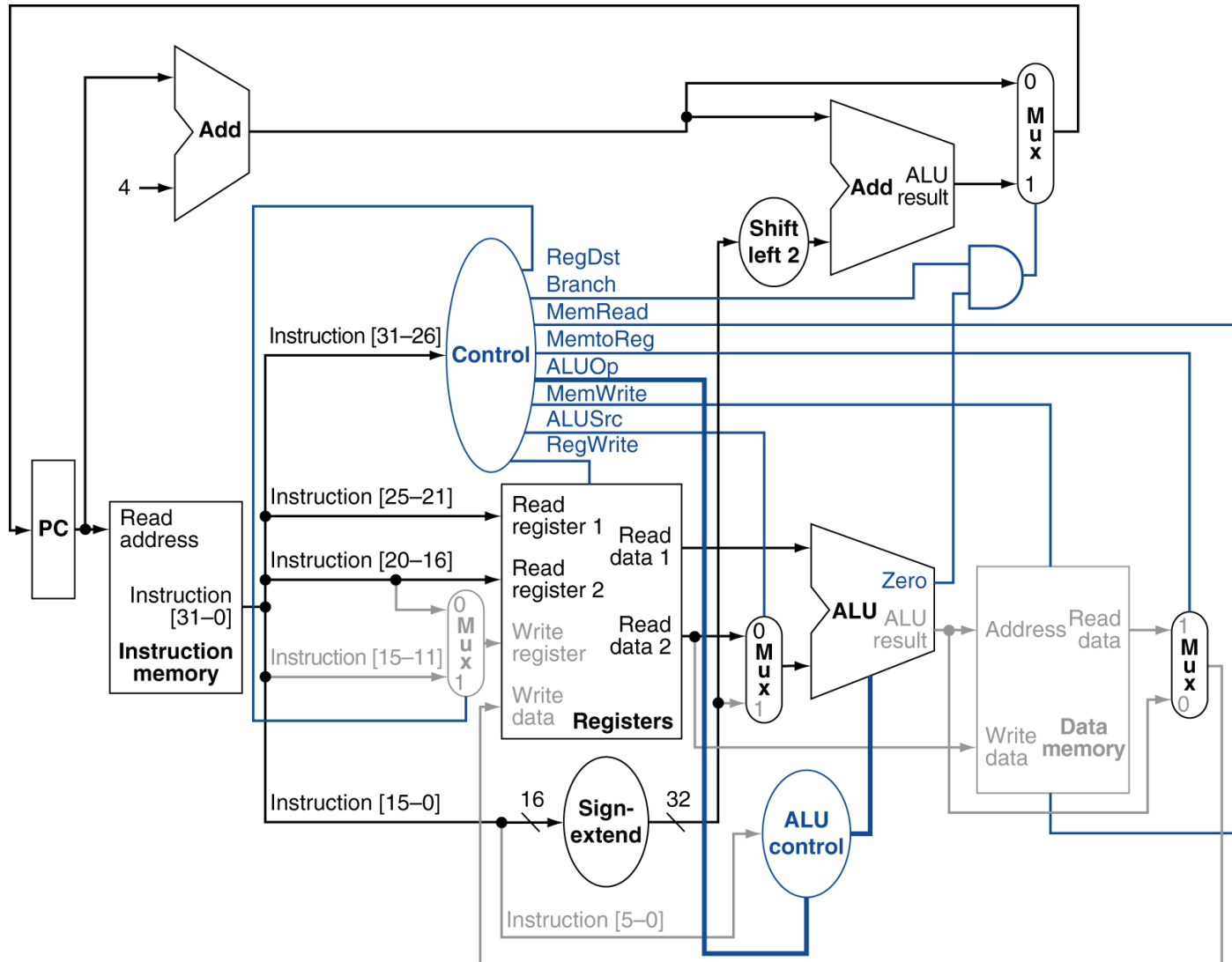
R-Type Instruction



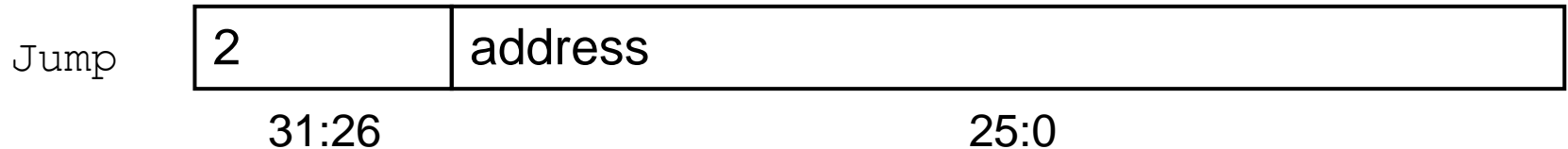
Load Instruction



Branch-on-Equal Instruction



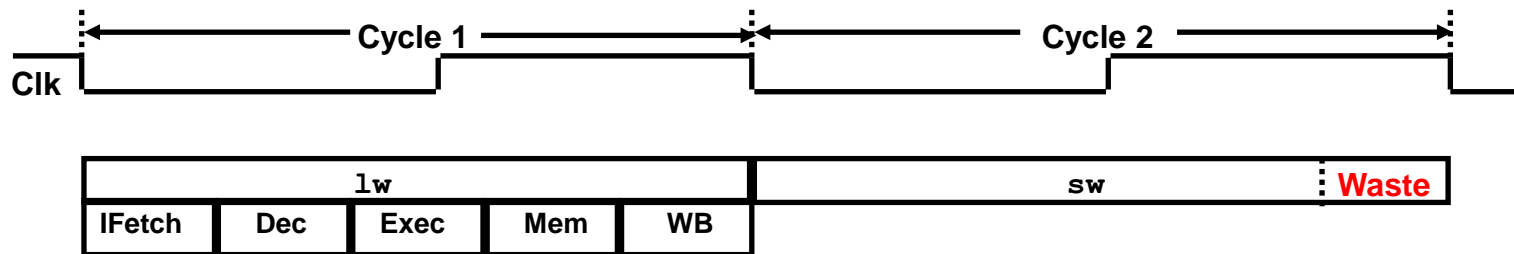
Implementing Jumps



- 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

Performance Issues

- Longest delay determines clock period
 - Critical path: load instruction
 - Instruction memory → register file → ALU → data memory → register file



- Not feasible to vary period for different instructions
- Violates design principle
 - Making the common case fast
- We will improve performance by pipelining

Next Class

- Improving performance
 - Pipeline
 - Pipeline problems

The Processor

Computer Organization

Sunday, 09 October 2022

Many slides adapted from:
Computer Organization and Design,
Patterson & Hennessy
5th Edition, © 2014, MK
and from Prof. Mary Jane Irwin, PSU



TÉCNICO LISBOA