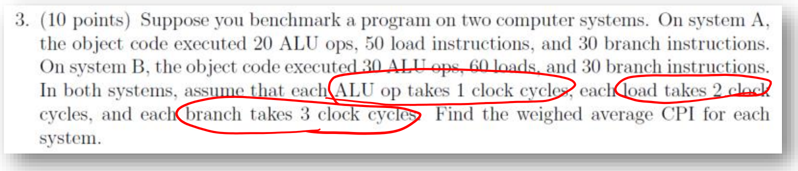
The Processor

COMPUTER ORGANIZATION AND ARCHITECTURE

Introduction

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



3. (10 points) Suppose you benchmark a program on two computer systems. On system A, the object code executed 20 ALU ops, 50 load instructions, and 30 branch instructions. On system B, the object code executed 30 ALU ops, 60 loads, and 30 branch instructions. In both systems, assume that each ALU op takes 1 clock cycles, each load takes 2 clock cycles, and each branch takes 3 clock cycles. Find the weighed average CPI for each system.

- 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`

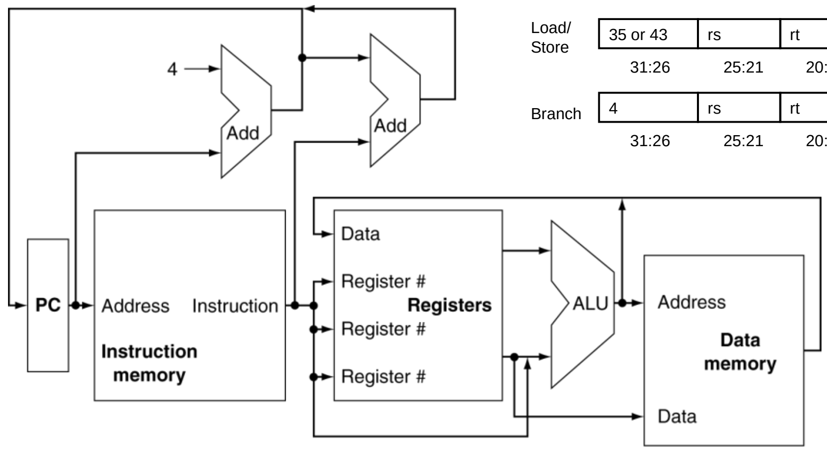
CSULB

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 ← target address or PC + 4

CSULB

CPU Overview



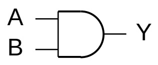
R-type	0	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0
Load/ Store	35 or 43	rs	rt	address		
	31:26	25:21	20:16	15:0		
Branch	4	rs	rt	address		
	31:26	25:21	20:16	15:0		

sw $t0, 4($t1)

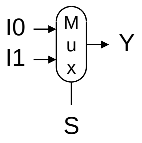
CSULB

Combinational Elements

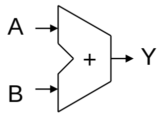
- AND-gate
  - Y = A & B



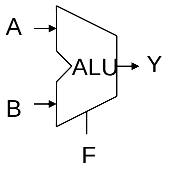
- Multiplexer
  - Y = S ? I1 : I0



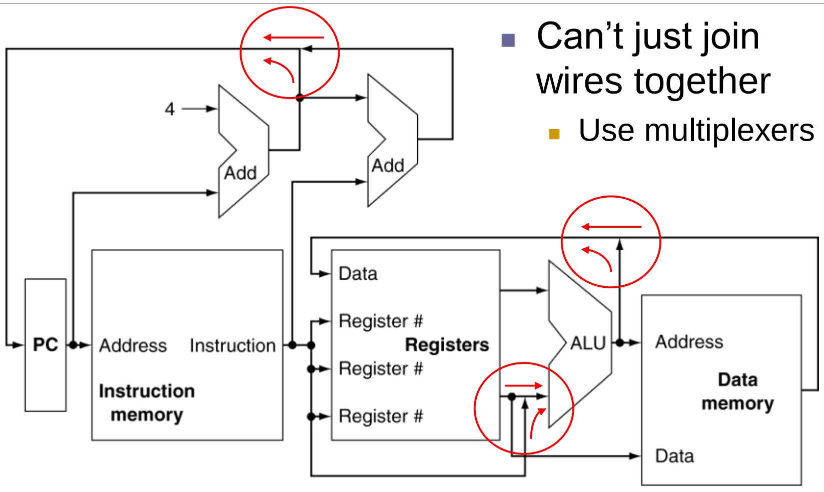
- Adder
  - Y = A + B



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

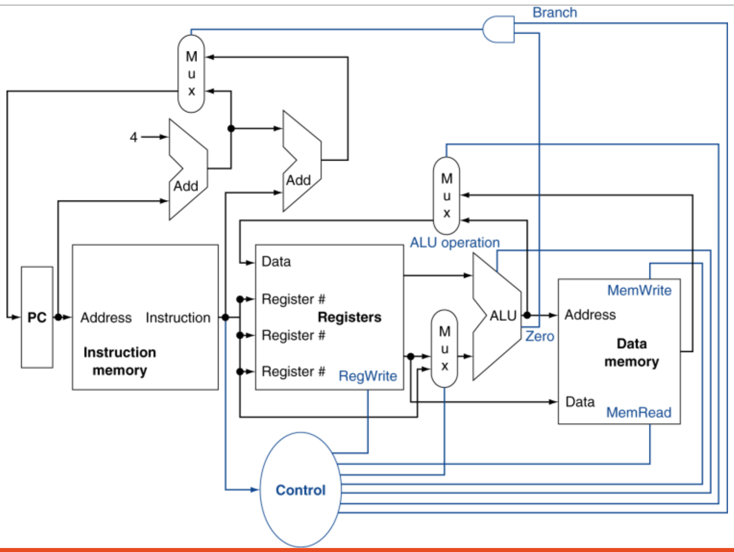


Multiplexers



- Can't just join wires together
  - Use multiplexers

Control

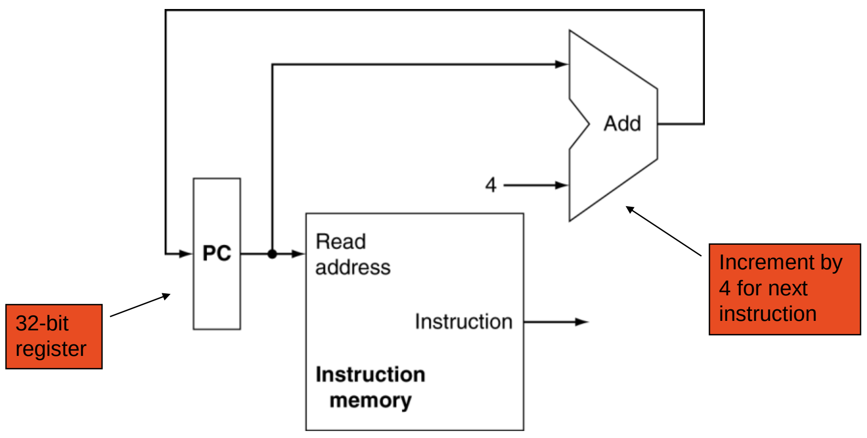


R-type	0	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0
Load/ Store	35 or 43	rs	rt	address		
	31:26	25:21	20:16	15:0		
Branch	4	rs	rt	address		
	31:26	25:21	20:16	15:0		

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



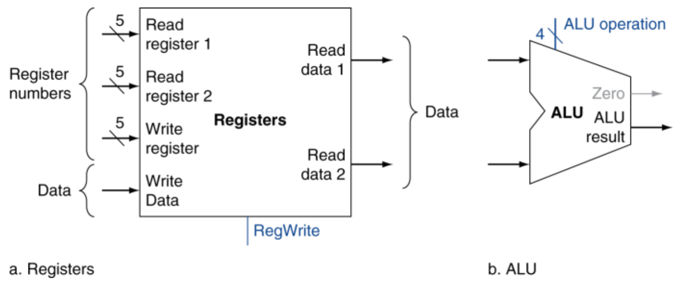
MIPS R-format Instructions

- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount (00000 for now)
  - funct: function code (extends opcode)

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

R-Format Instructions

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



CSULB

MIPS I-format Instructions

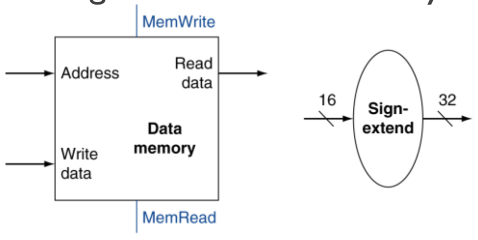
- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15}-1$
  - Address: offset added to base address in rs

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

CSULB

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

$\underbrace{10000 \sim 0}_{16 \text{ bits}}$

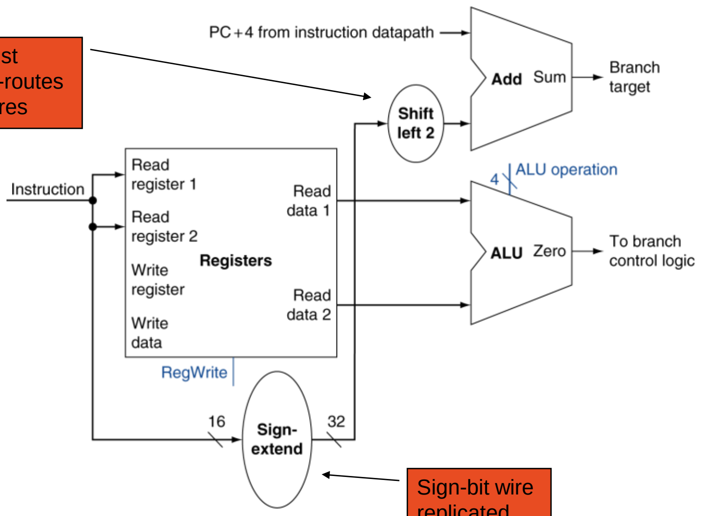
$\underbrace{1 \sim 1}_{16} \underbrace{10000 \sim 0}_{16}$

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

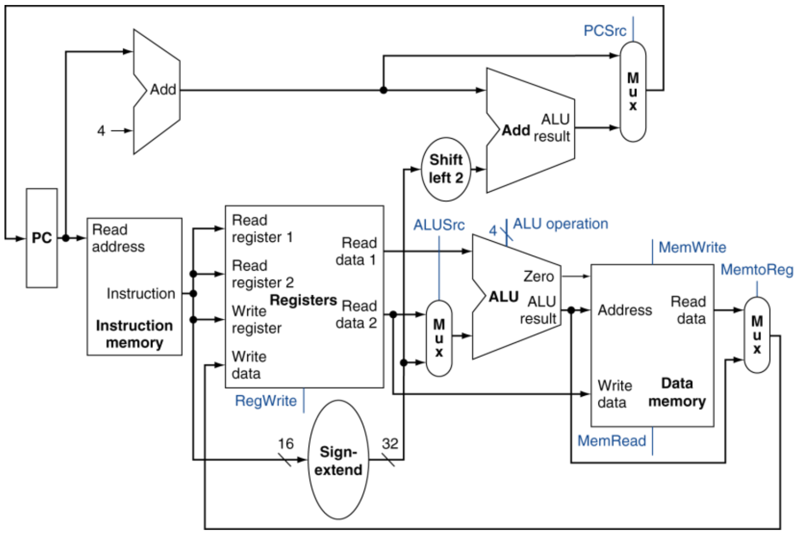
Branch Instructions

Just re-routes wires



Sign-bit wire replicated

R-Type/Load/Store 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

CSULB

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

10/12

CSULB