EECS2021 Computer Organization

Chapter 4 Part 1

The Processor

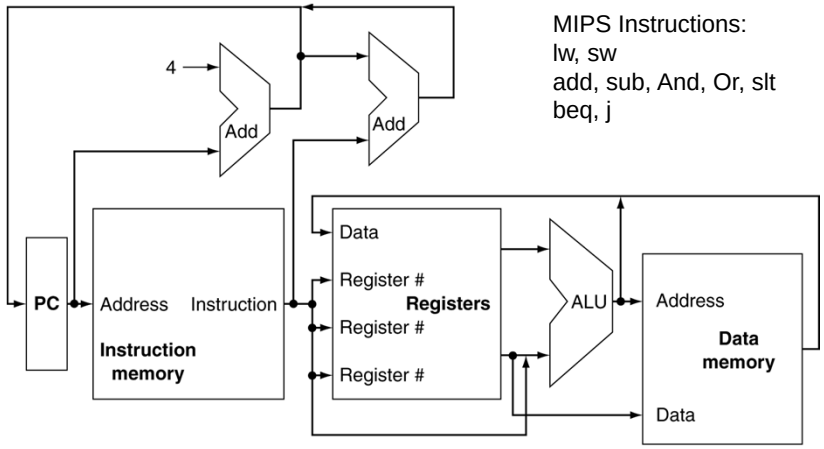
Outline

- CPU overview
- Single cycle MIPS implementation
  - Simple subset
  - Memory reference: lw, sw
  - Arithmetic/logical: add, sub, and, or, slt
  - Control transfer: beq, j
- Pipelined MIPS implementation (Part 2)

MIPS Core Instructions

Arithmetic	add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
	subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
	add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
	add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
	subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
	add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
	Move fr. copr. reg.	mfc0 $1,$epc	$1 = $epc	Used to get exception PC
	multiply	mult $2,$3	Hi, Lo = $2 ¥ $3	64-bit signed product in Hi, Lo
	multiply unsigned	multu $2,$3	Hi, Lo = $2 ¥ $3	64-bit unsigned product in Hi, Lo
	divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
	divide unsigned	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quotient and remainder
	Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
	Move from Lo	mflo $1	$1 = Lo	Use to get copy of Lo
Logical	and	and $1,$2,$3	$1 = $2 & $3	3 register operands; logical AND
	or	or $1,$2,$3	$1 = $2 | $3	3 register operands; logical OR
	and immediate	and $1,$2,100	$1 = $2 & 100	Logical AND register, constant
	or immediate	or $1,$2,100	$1 = $2 | 100	Logical OR register, constant
	shift left logical	sll $1,$2,10	$1 = $2 << 10	Shift left by constant
	shift right logical	srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
Data transfer	load word	lw $1,100($2)	$1 = Memory[$2+100]	Data from memory to register
	store word	sw $1,100($2)	Memory[$2+100] = $1	Data from register to memory
	load upper imm.	lui $1,100	$1 = 100 x 2^{16}	Loads constant in upper 16 bits

MIPS CPU Overview

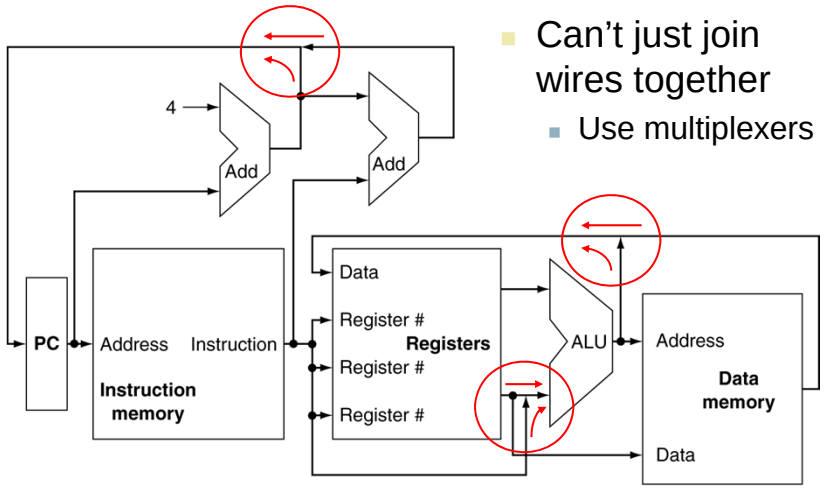


MIPS Instructions:
lw, sw
add, sub, And, Or, slt
beq, j

Instruction Execution

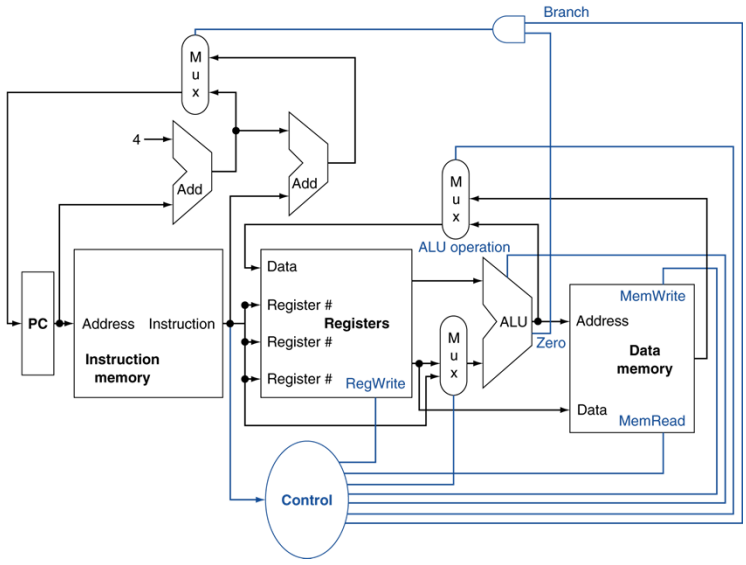
- PC (program counter)
  - provides the instruction memory address
  - used to fetch instruction
  - Instruction is fetched from instruction memory based on address in the PC
- Register numbers are specified by the instruction
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC → target address or PC + 4

The Use of Multiplexers



- Can't just join wires together
  - Use multiplexers

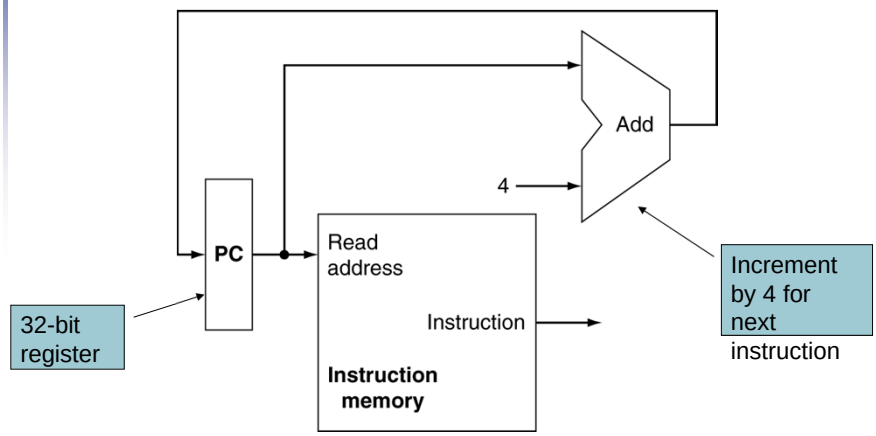
Use of Control in CPU



Building a Datapath

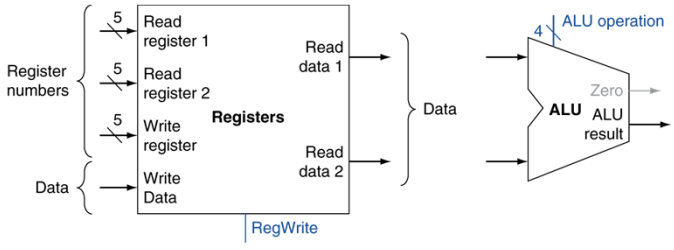
- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, mux's, memories, …
- We will build a MIPS datapath incrementally
  - Refining the overview design

Instruction Fetch



R-Format Instructions

- R-type instructions: add, sub, and, or, slt
- Example: add $s1, $s2, $s3
  1. Read two register operands ($s2, $s3)
  2. Perform arithmetic/logical operation (add, sub, and, or, slt)
  3. Write register result ($s1)

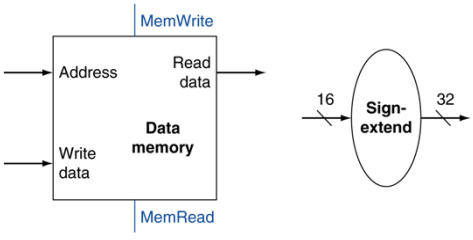


Load/Store Instructions

- lw $s1, offset($s2)
  - Read register ($s2) specified in the instruction
  - Offset is sign extended to 32 bits
  - ALU adds offset to register to obtain data memory address
  - Data memory transfers data memory to specified register ($s1)
- sw $s1, offset(Ss2)
  - Read two register operands ($s1, $s2)
  - Offset is sign extended to 32 bits
  - ALU adds offset to register ($s2) to obtain data memory address
  - Write register value ($s1) to memory

Load/Store Instructions (2)

- Load/Store requires additional components
  - Data memory
  - A sign extension unit

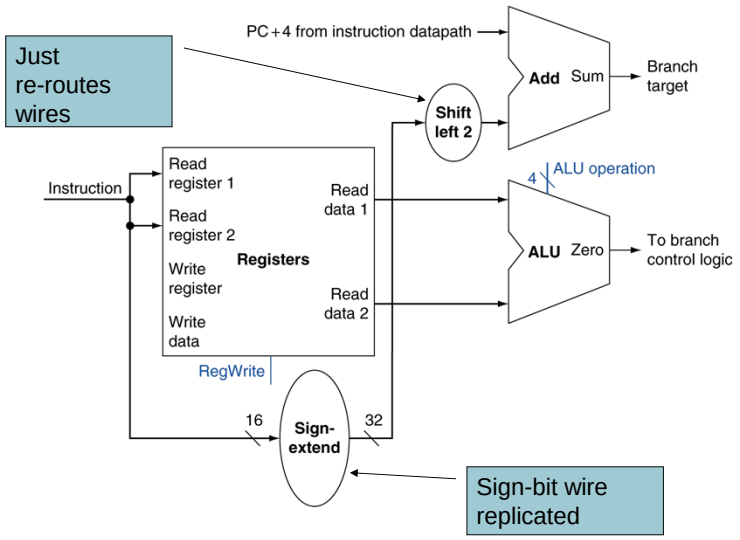


a. Data memory unit b. Sign extension unit

Branch Instructions

- beq $s1, $s2, Loop
  - If $s1==$s2, goto (PC+4)+4*offset
- 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

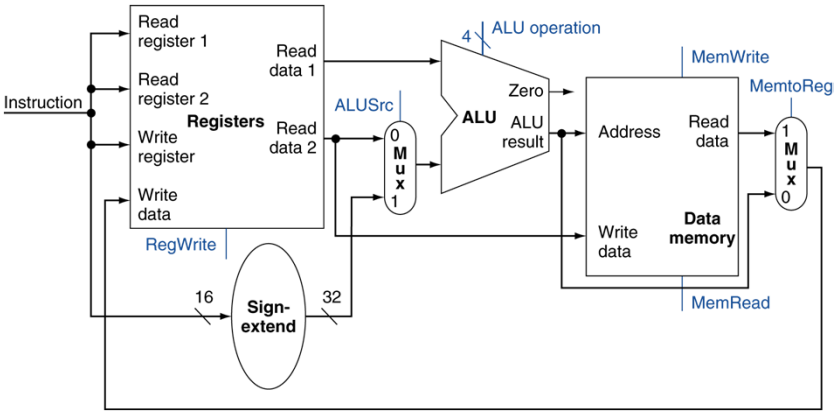
Branch Instructions



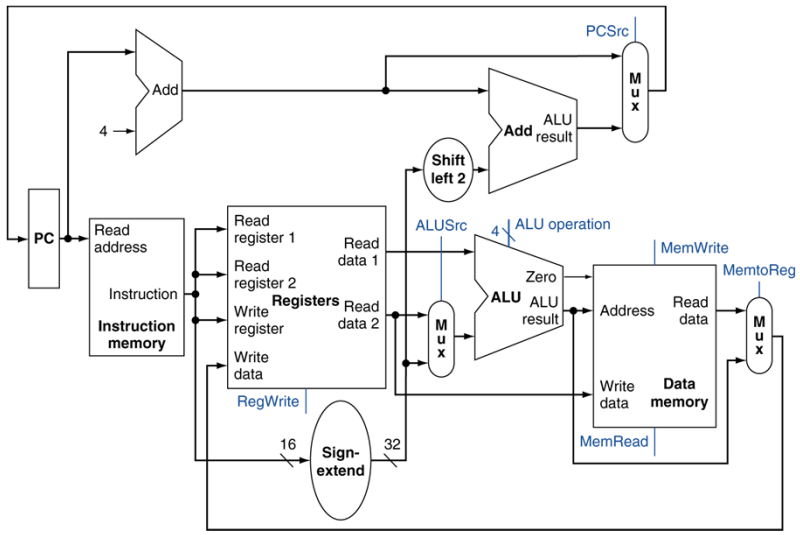
Composing the Elements

- 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: Function = add
  - Branch: Function = subtract
  - R-type: Function depends on funct field

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

MIPS Instruction Format

- R-format

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

- Example

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

MIPS Instruction Format

- I-format

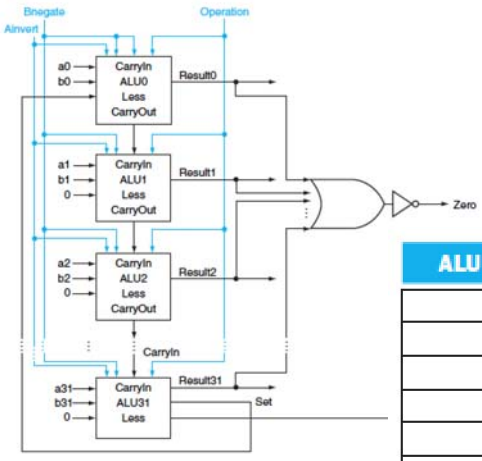
op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Example

lw $t0, 32($s3)

op	rs	rt	Constant or address
35	19	8	32
100011	10011	01000	0000,0000,0010,0000

Control of 32-Bit ALU



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

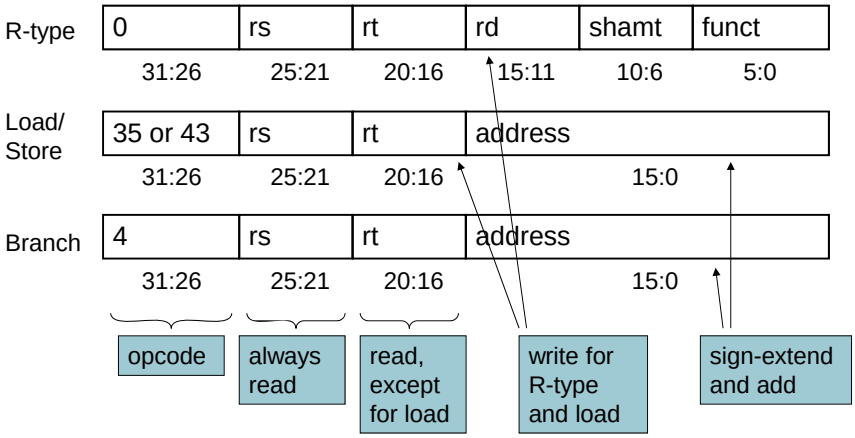
ALU Control

- 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

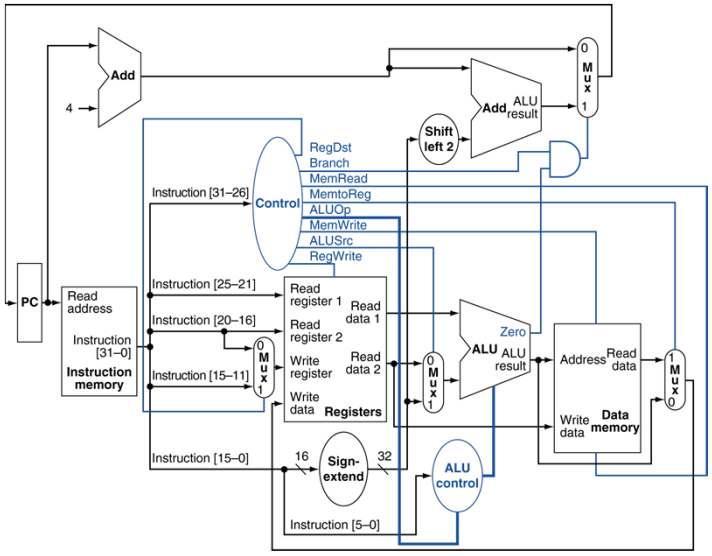


ALU Control

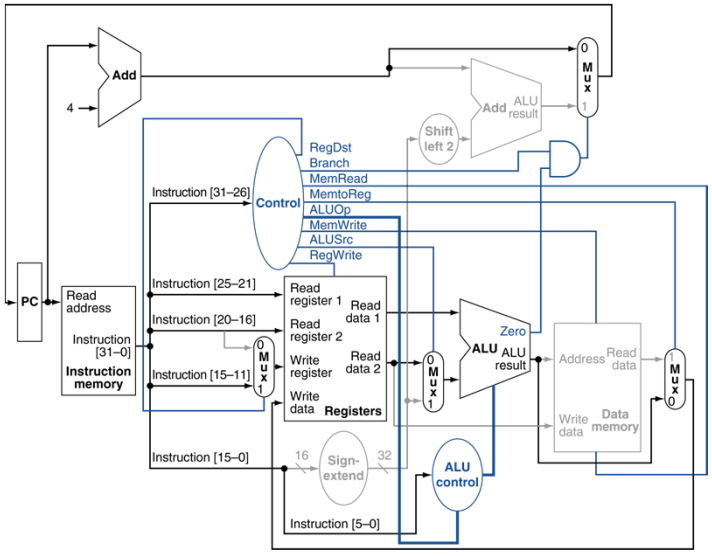
- Truth table for ALU control

ALUOp		Funct field						ALU control
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	1	0	0	0	0	0	0010
1	0	1	0	0	0	1	0	0110
1	0	1	0	0	1	0	0	0000
1	0	1	0	0	1	0	1	0001
1	0	1	0	1	0	1	0	0111

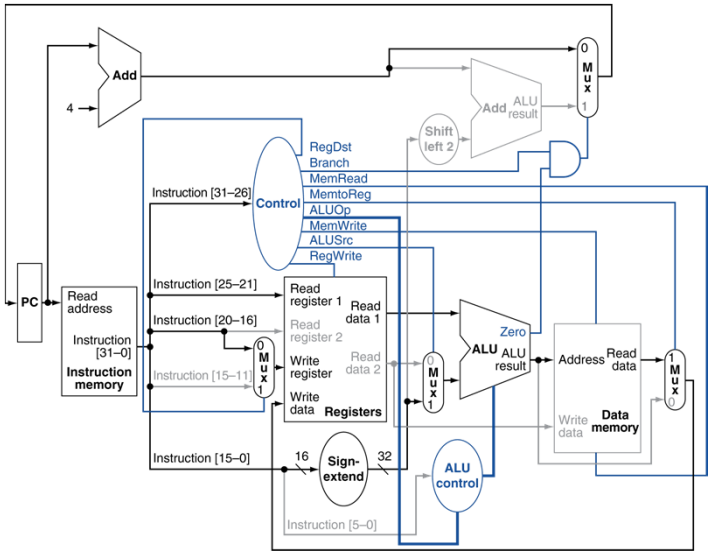
Datapath With Control



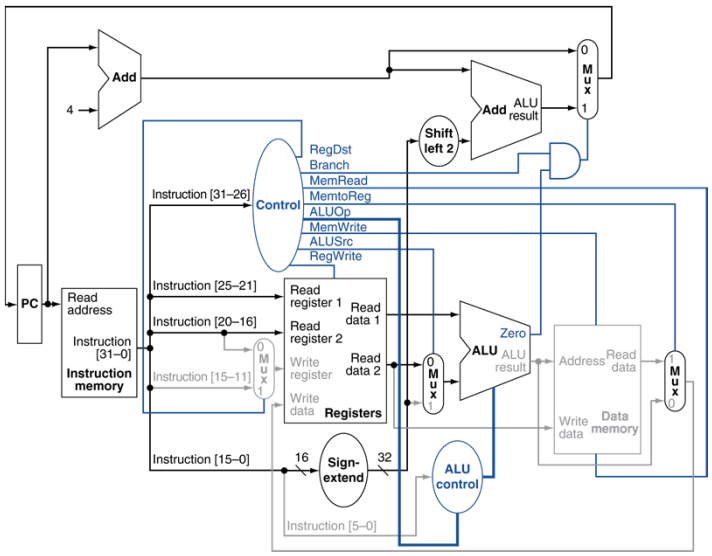
R-Type Instruction



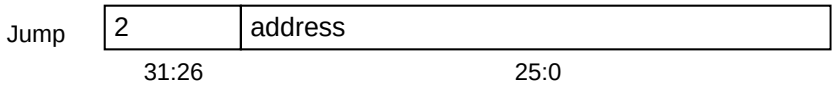
Load Instruction



Branch-on-Equal Instruction

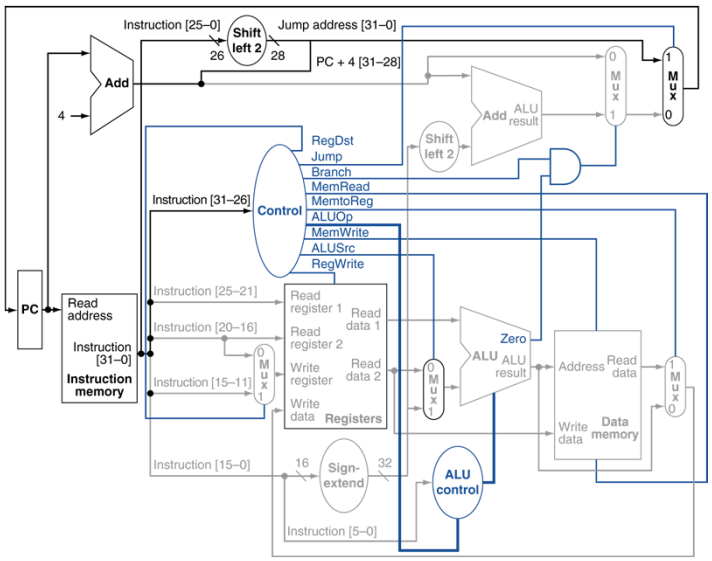


Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00 for low two bits
- 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 → 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