

## **The Processor**

**Işıl ÖZ, IZTECH, Fall 2023**

**23 November 2023**



# Basic MIPS Implementation

## Subset of the MIPS instruction set

### Memory reference instructions

lw, sw

### Arithmetic-logical instructions

add, sub, and, or, slt

### Control transfer instructions

beq, j

# Instruction Execution

**PC → Instruction memory, fetch instruction**

**Register numbers → register file, read registers**

**Use ALU to calculate**

Arithmetic result (ALU instructions)

Memory address calculation (Memory instructions)

Comparison (Branch instructions)

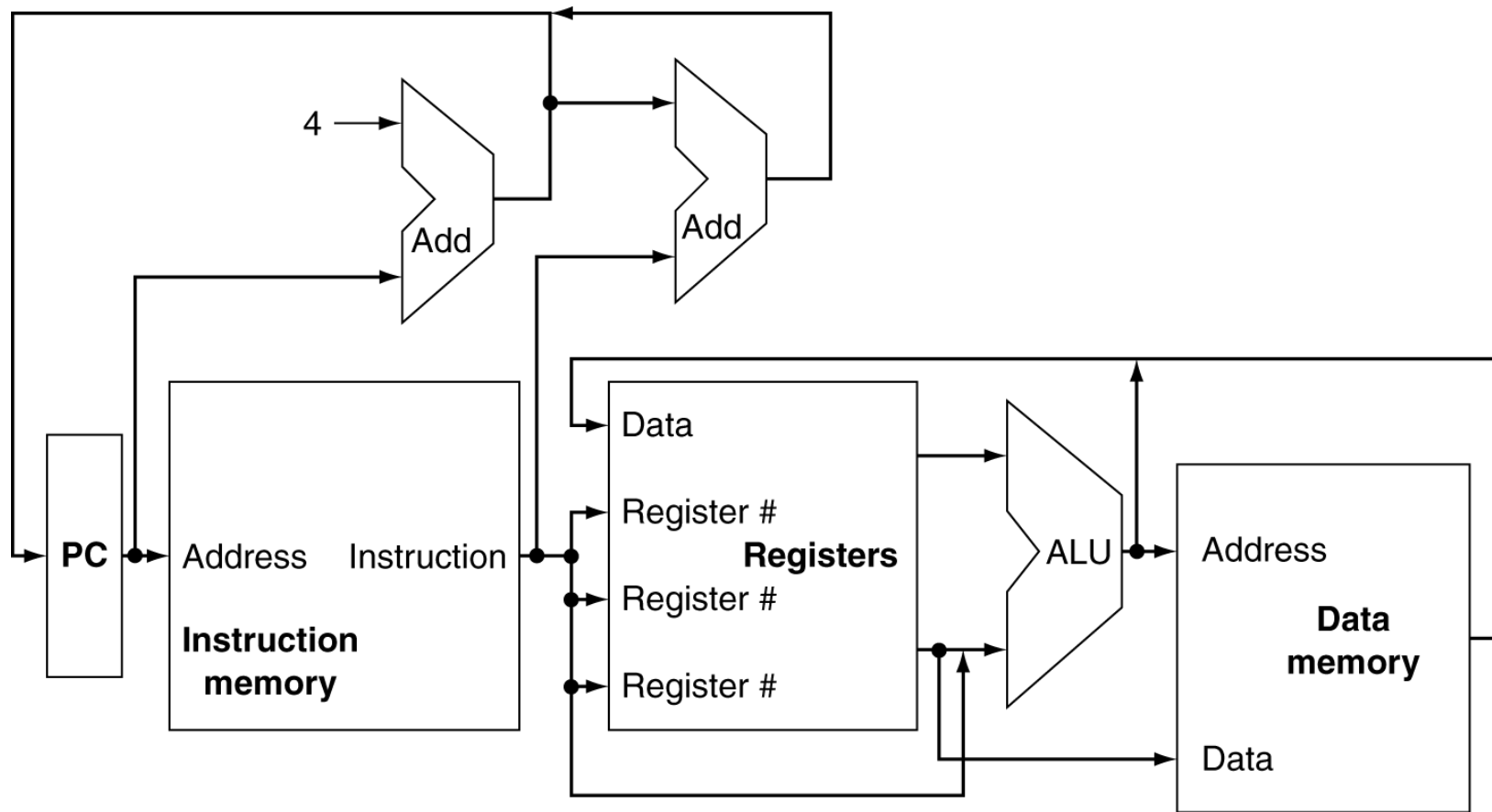
**Access the data memory for load/store**

**Write the data from ALU to register for ALU**

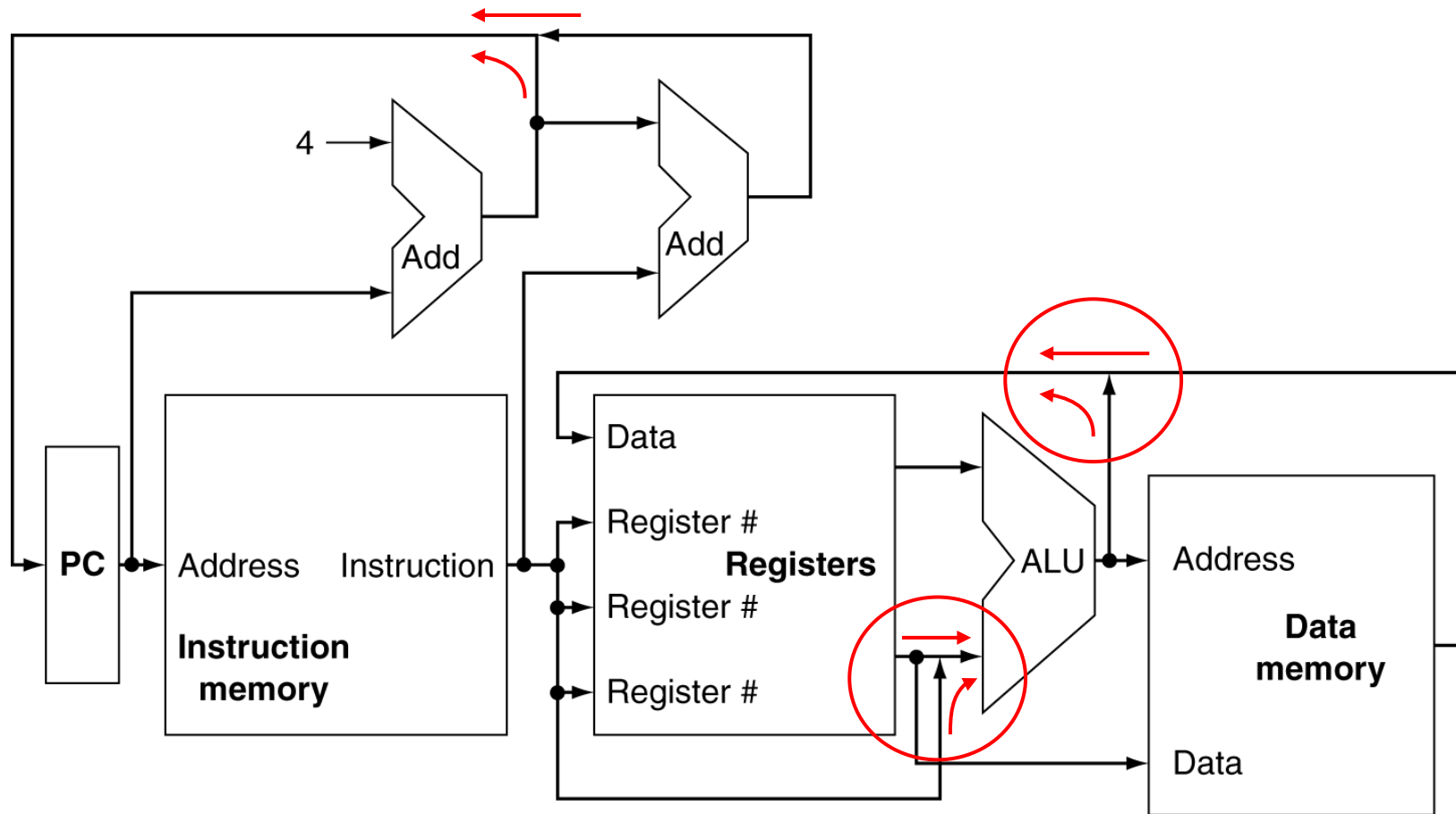
**Change the PC based on comparison for branch**

**PC ← target address or PC + 4**

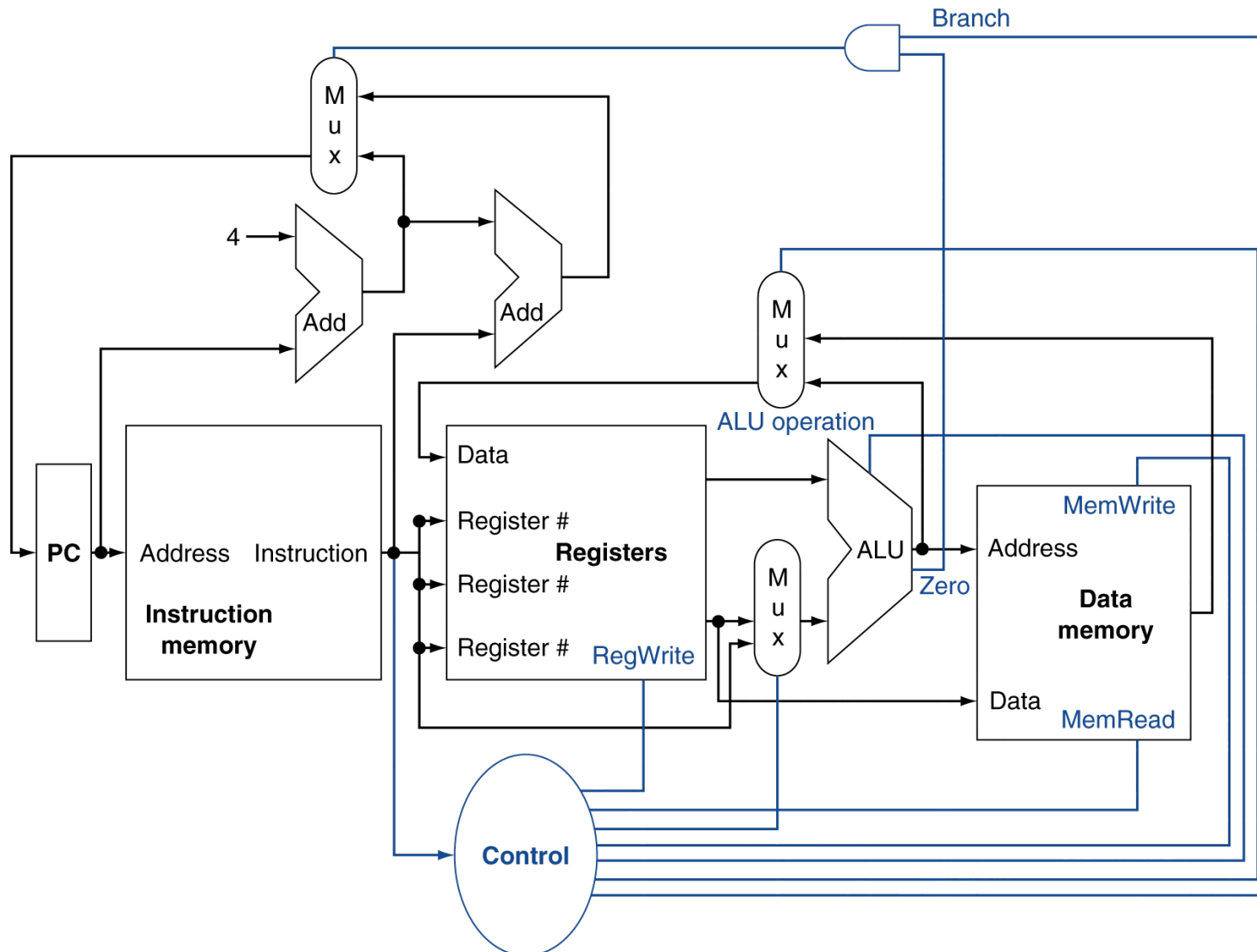
# Abstract CPU Overview



# Multiplexers



# Control



# Logic Design Basics

## Information encoded in binary

Low voltage = 0, High voltage = 1

One wire per bit

Multi-bit on multi-wire buses

## Combinational element

Operate on data

Input vs output

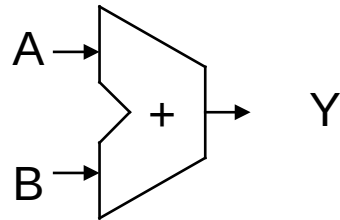
## Sequential element

Store information (state)

# Combinational Elements

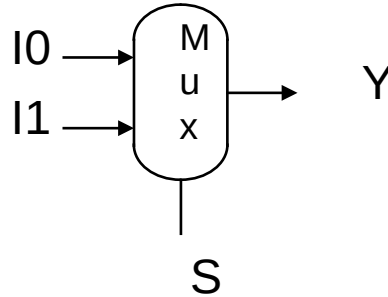
## Adder

$$Y = A + B$$



## Multiplexer

$$Y = S ? I1 : I0$$





# Sequential Elements

## Register file

# Building a Datapath

**Datapath elements that process data and addresses in the CPU**

**Build the MIPS datapath incrementally**

# Instruction Fetch

**Fetch the instruction from instruction memory**

**Increment the PC so that it points at the next instruction, 4 bytes later**

**Instruction memory**

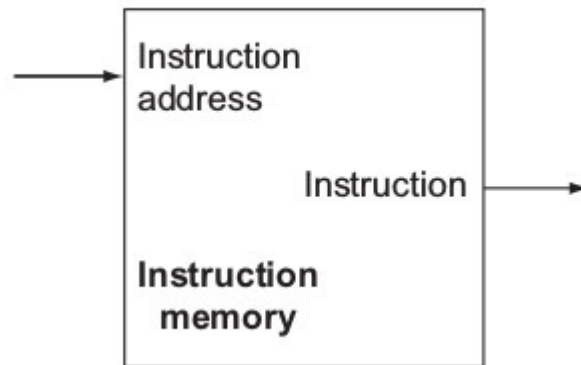
**PC**

**Adder**

# Instruction Memory

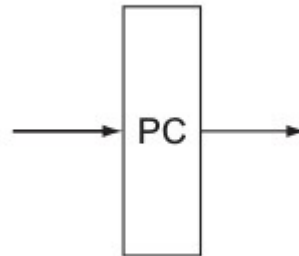
**Assume that we write all instructions when we load the program**

**Only reads from instruction memory (like a combinational element: the output is the content of the location specified by the address input)**



# Program Counter

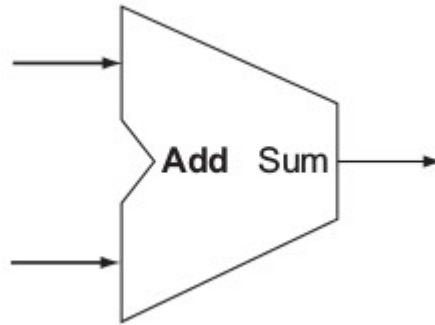
**A 32-bit register that is written at the end of every clock cycle**



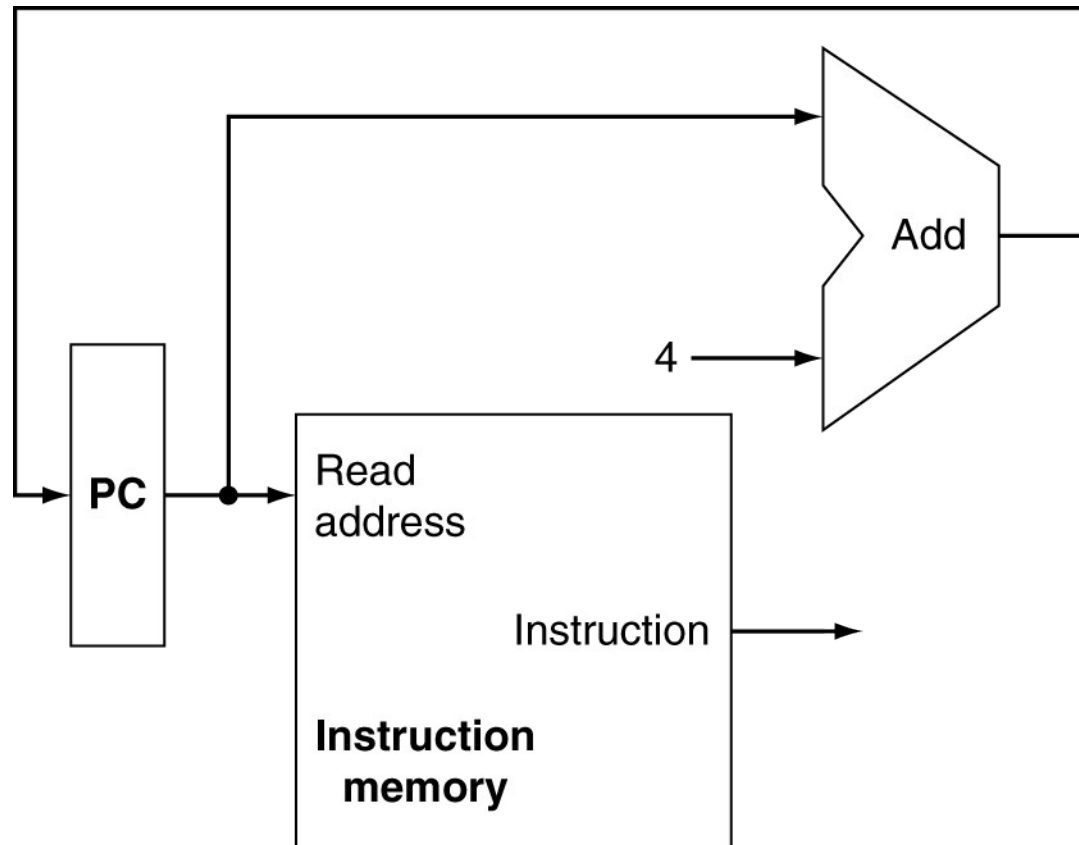
# Adder

**An ALU wired to always add its two 32-bit inputs  
and place the sum on its output**

**Increments PC by 4 for the next instruction**



# Instruction Fetch



# Instruction Fetch

```
Loop: sll  $t1, $s3, 2      80000
      add  $t1, $t1, $s6    80004
      lw   $t0, 0($t1)      80008
      bne  $t0, $s5, Exit   80012
      addi $s3, $s3, 1      80016
      j    Loop            80020
Exit: ...                  80024
```

0	0	19	9	2	0
0	9	22	9	0	32
35	9	8	0		
5	8	21	2		
8	19	19	1		
2	20000				



# R-Format Instructions

**Read two registers**

**Perform an arithmetic/logical operation on the contents of the registers**

**Write the result to a register**

**add, sub, and, or, slt**

`add $t1, $t2, $t3`

**Register file**

**ALU**

# Register File

**Contains all the registers**

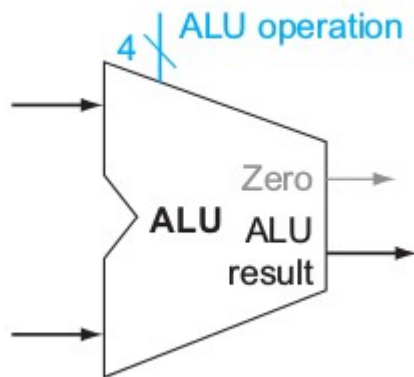
**Has two read ports, one write port**

**Outputs the contents of the registers on the outputs**



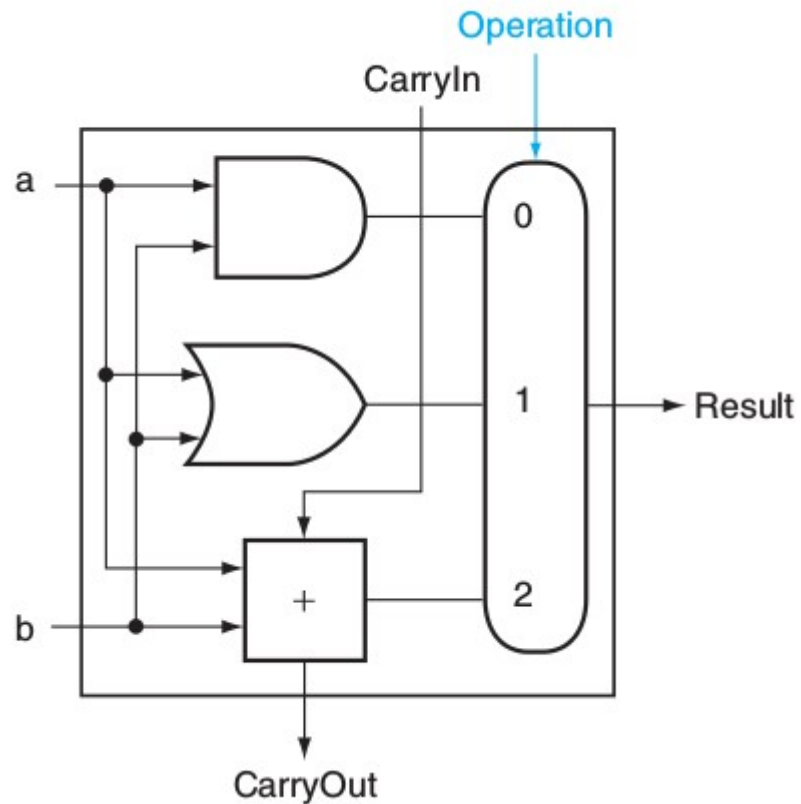
# ALU

**The operation to be performed by the ALU is controlled with the ALU operation signal**



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

# 1-bit ALU for AND, OR, add



# Load/Store Instructions

**Read register operands**

**Compute a memory address by adding the base register, to the signed offset (sign-extend) value**

**Load/store the value from/to memory into/from register**

`lw $t1, offset ($t2)`

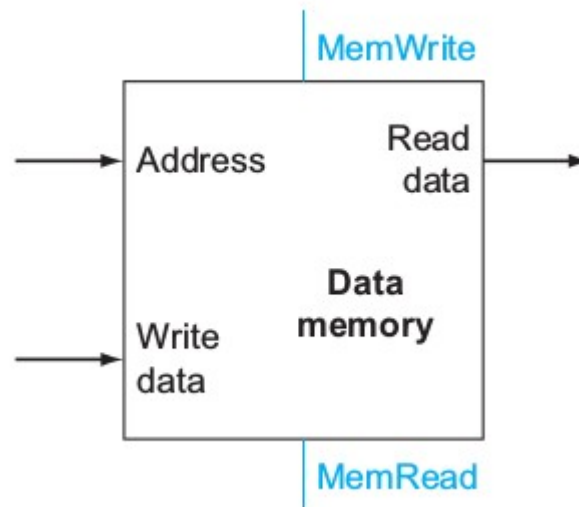
`sw $t1, offset ($t2)`

**Data memory unit**

**Sign extension unit**

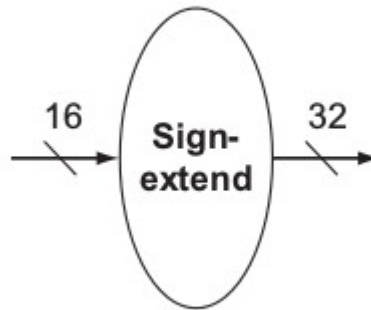
# Data Memory

**A state element with inputs for the address and the write data, output for the read result**



# Sign Extension

# Sign extension replicates the most significant bit loaded into the upper 16 bits



1111111111111111->

**111**

# 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)

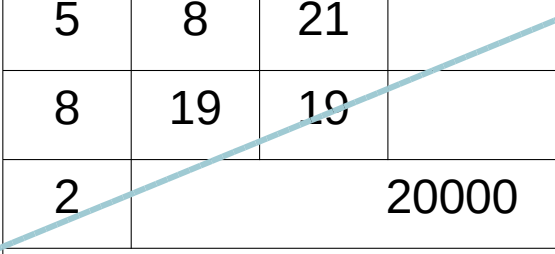
Add to PC + 4



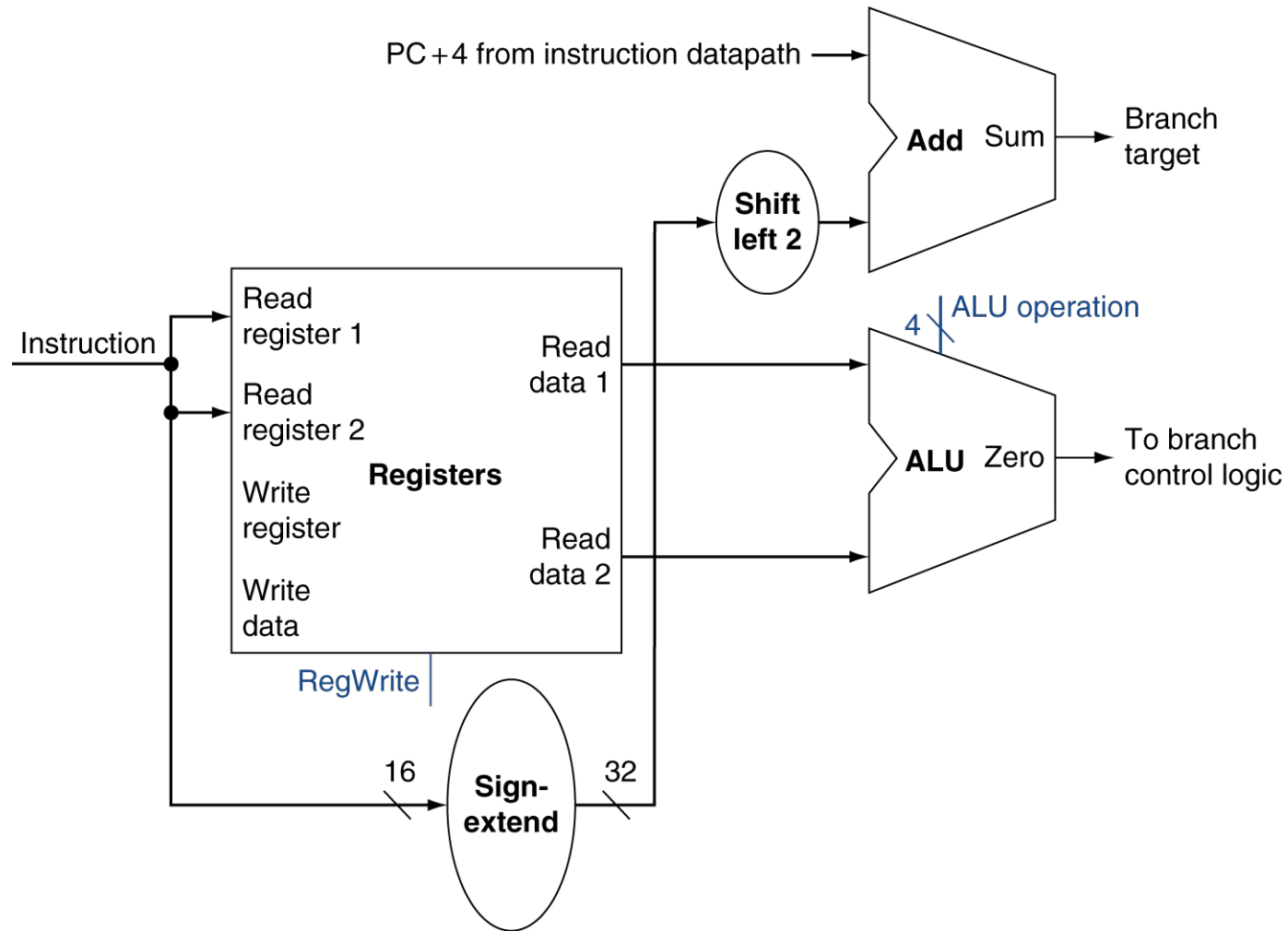
# Target Addressing Example

## Assume Loop at location 80000

Loop: sll	\$t1, \$s3, 2	80000	0	0	19	9	2	0
add	\$t1, \$t1, \$s6	80004	0	9	22	9	0	32
lw	\$t0, 0(\$t1)	80008	35	9	8	0		
bne	\$t0, \$s5, Exit	80012	5	8	21	2		
addi	\$s3, \$s3, 1	80016	8	19	19	1		
j	Loop	80020	2	20000				
Exit: ...		80024						



# Branch Instructions

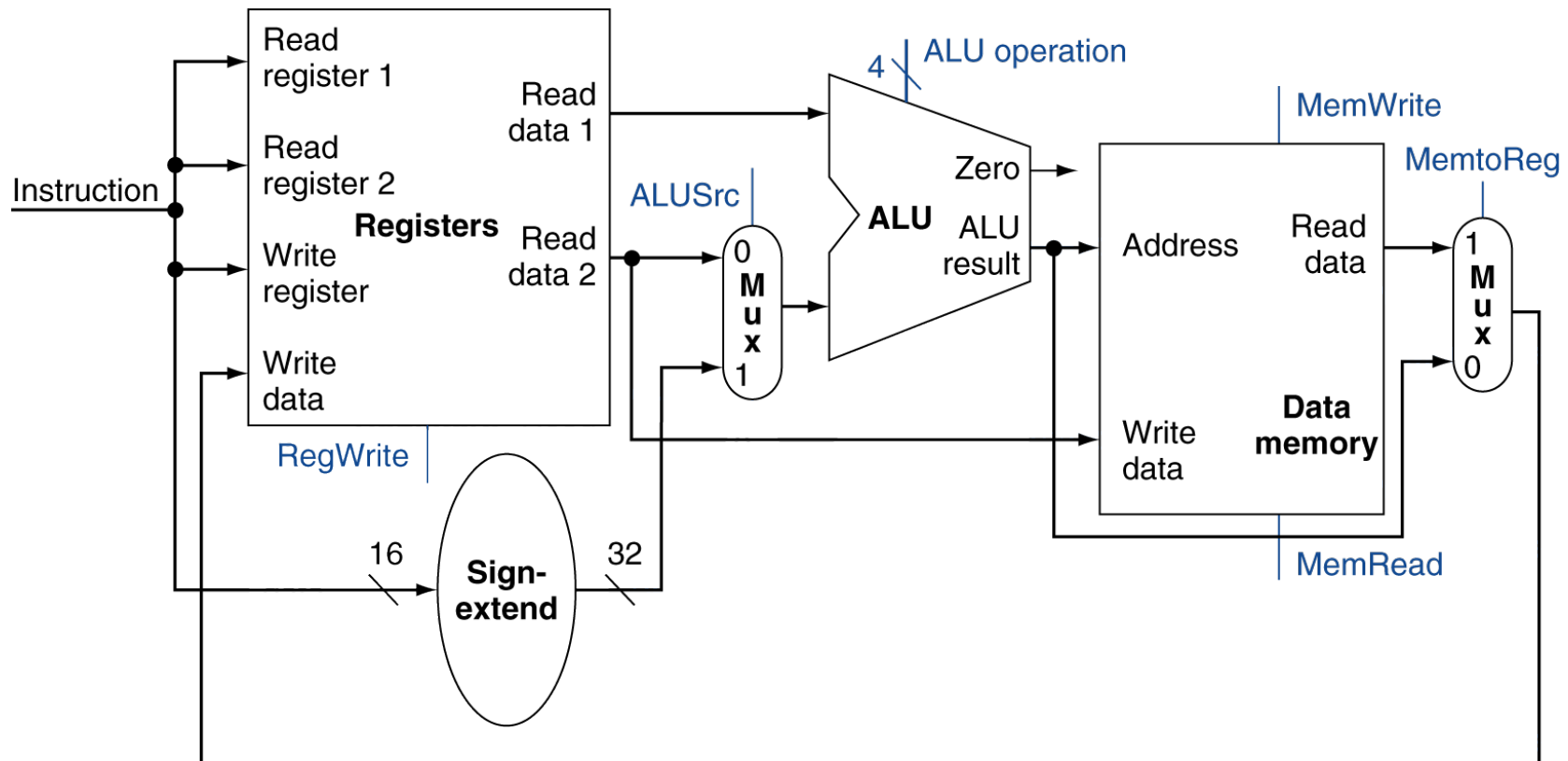


# A Single Datapath

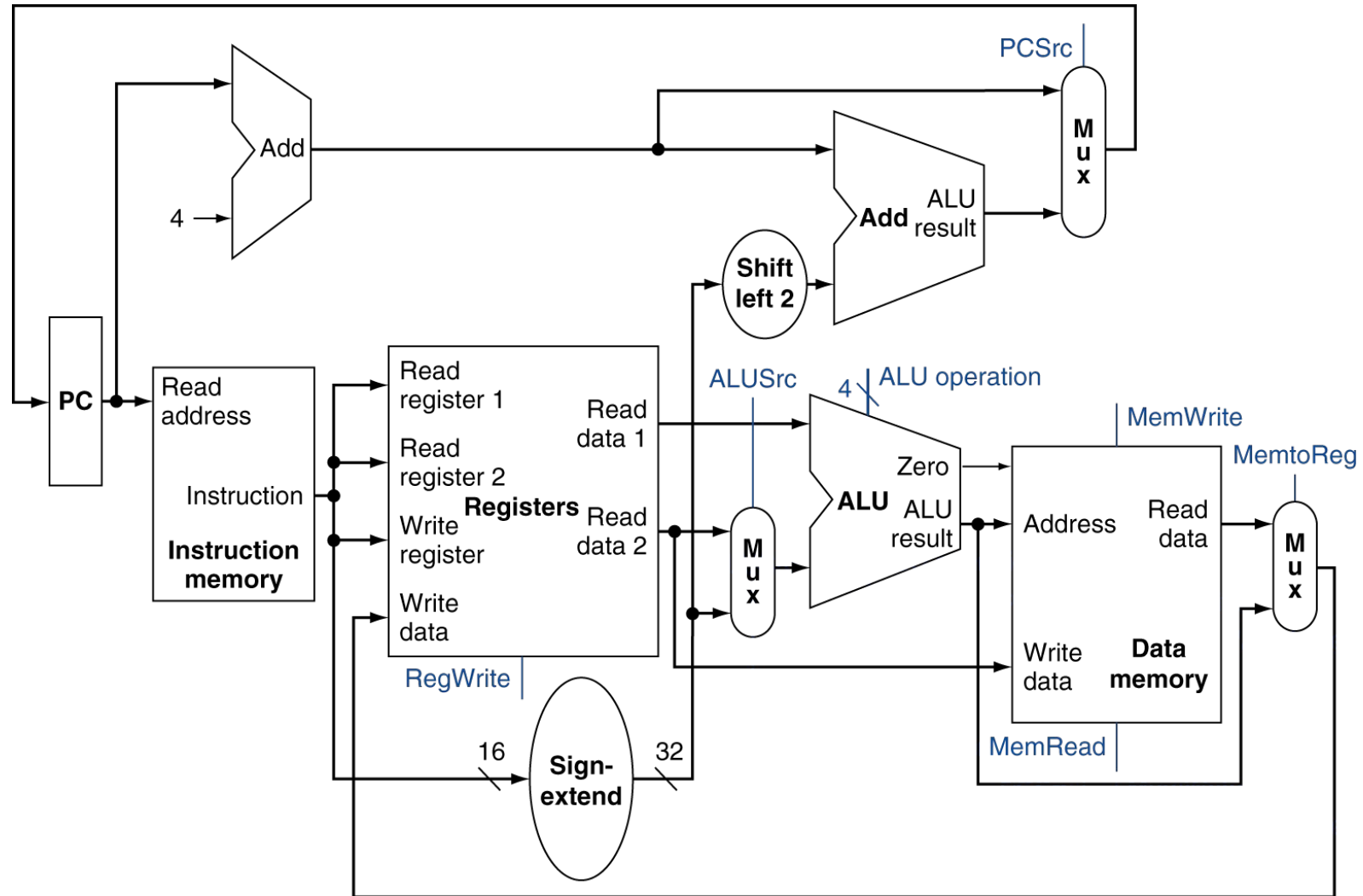
**Execute all instructions in one clock cycle**

**To share a datapath element between two different instruction classes, allow multiple connections to the input of an element (multiplexer and control signal to select among multiple inputs)**

# Datapath for Memory and R-Type Instructions



# Full Datapath



# References

**Chapter 4.1, 4.2, 4.3**

**(Computer Organization and Design: The  
Hardware/Software Interface by  
Hennessy/Patterson, 5th edition)**