# CS202: COMPUTER ORGANIZATION

## Lecture 8

### The Processor

# Outline

- Implementation overview
- Logic design basics
- Detailed implementation for every instruction

# Introduction

CPU Time = Instruction Count × CPI × Clock Cycle Time

- 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

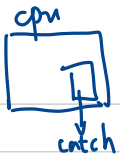
# Basic MIPS Architecture

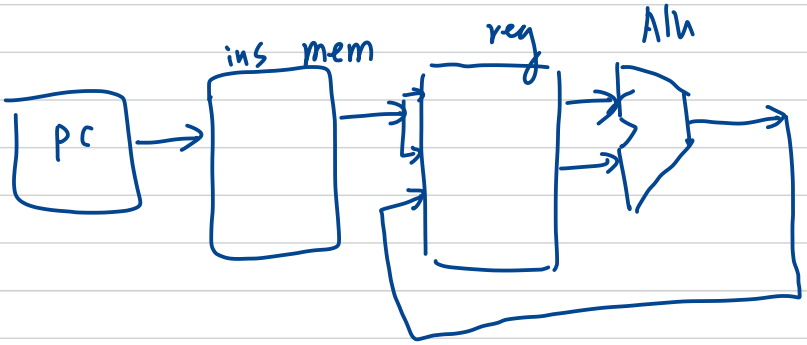
- We have known the instructions a CPU should execute, we'll design a simple CPU that executes:
  - basic math (add, sub, and, or, slt)
  - memory access (lw and sw)
  - branch and jump instructions (beq and j)

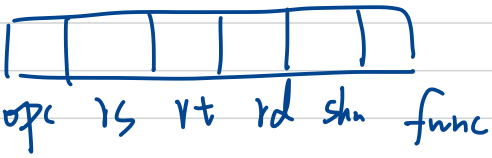
add $s0 $s1 $s2

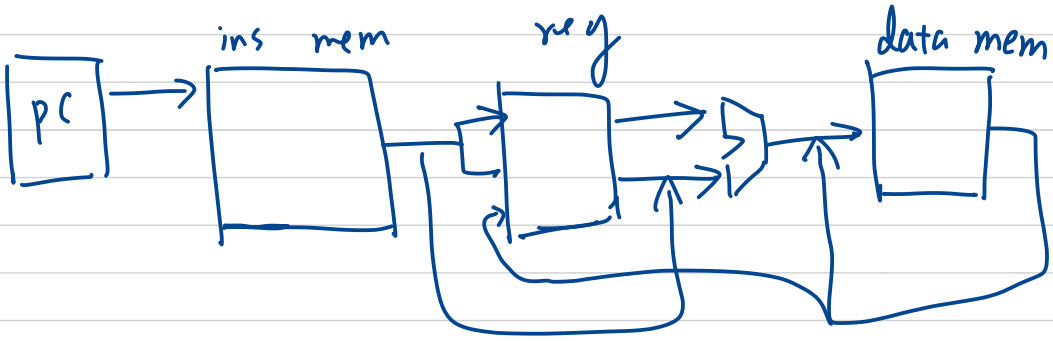
the instruction is in the memory, which is not in the cpu

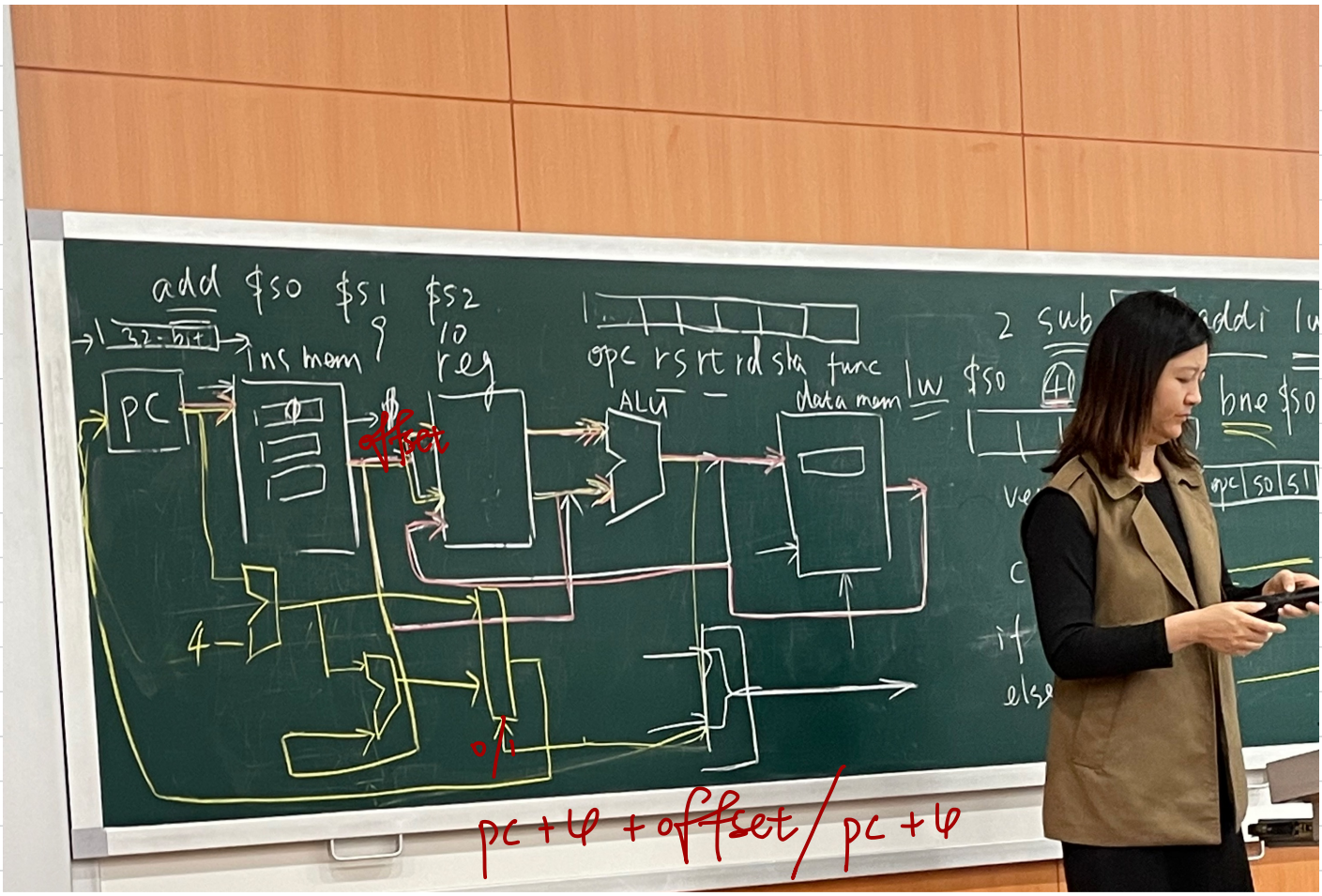
catch:



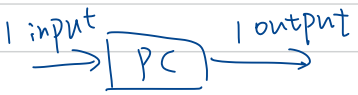



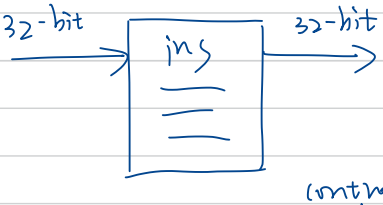

add $s0 $s1 $s2
32-bit
ins mem
reg
opc rs rt rd sha func
ALU
data mem
lw $s0
2 sub
addi
bne $s0
pc
offset
4
pc + 4 + offset / pc + 4

- pc will be update every edge

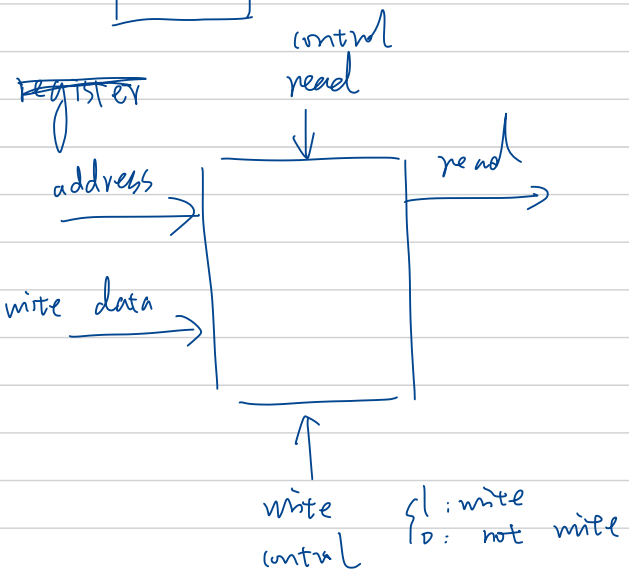
⟹ NO write for PC




- instruction mem
  the code will NOT be changed



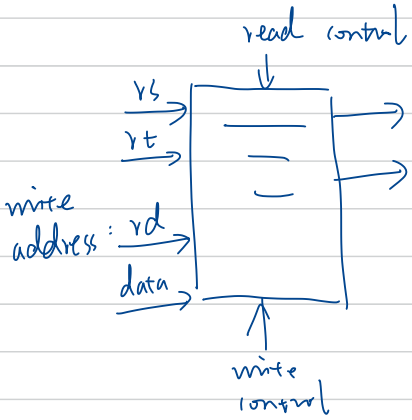

- ~~register~~




- registers

  read at least 2

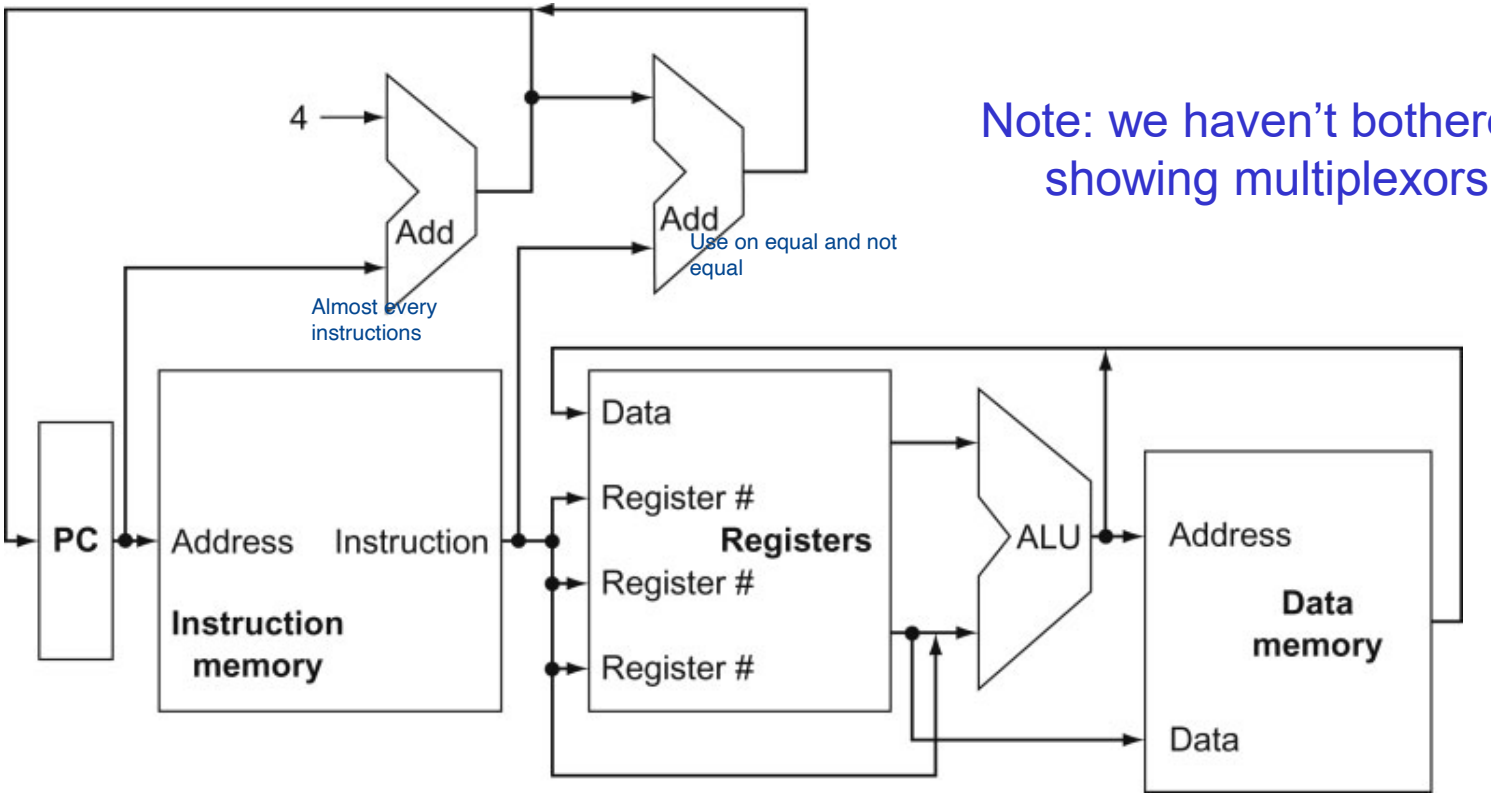
  read all the time : can omit as



# Implementation Overview

- We need memory
  - to store instructions
  - to store data
  - for now, let's make them separate units
- We need registers, ALU, and a whole lot of control logic
- CPU operations common to all instructions:
  - use the program counter (PC) to pull instruction out of instruction memory
  - read register values

# View from 30,000 Feet




Note: we haven't bothered showing multiplexors

Source: H&P textbook

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

# Outline

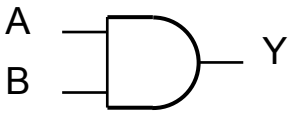
- Implementation overview
- Logic design basics
- Detailed implementation for every instruction

# Logic Design Basics

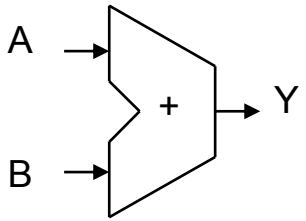
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

# Combinational Elements

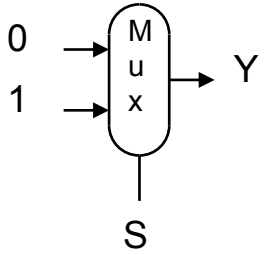
- And gate
  - Y = A & B



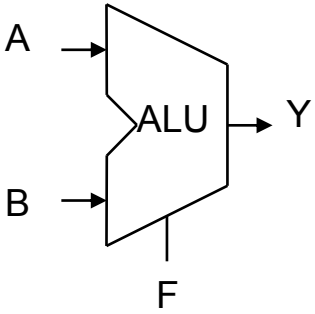

- Adder
  - Y = A + B




- Multiplexer
  - Y = S ? 1 : 0

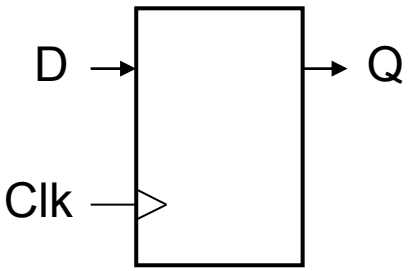


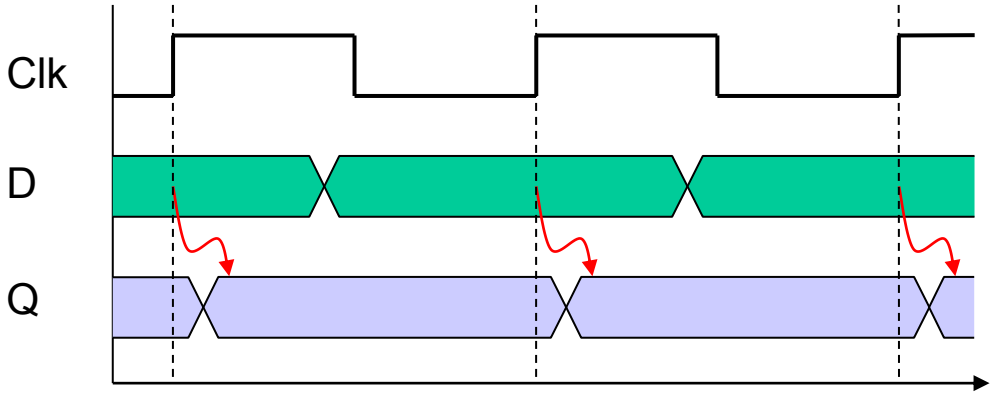

- Arithmetic/Logic Unit
  - Y = F(A, B)



# State Elements (sequential elements)

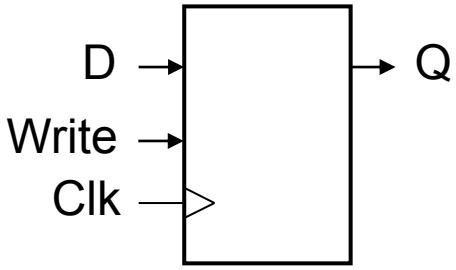
- State element
  - The state element has a pre-stored state
  - It has some internal storage
  - Has at least two inputs and one output (e.g. D-type flip-flop):
    - The data to be written into the element
    - The clock which determines when the data is written
    - The output: the value that was written in earlier cycle
  - Examples: register and memory

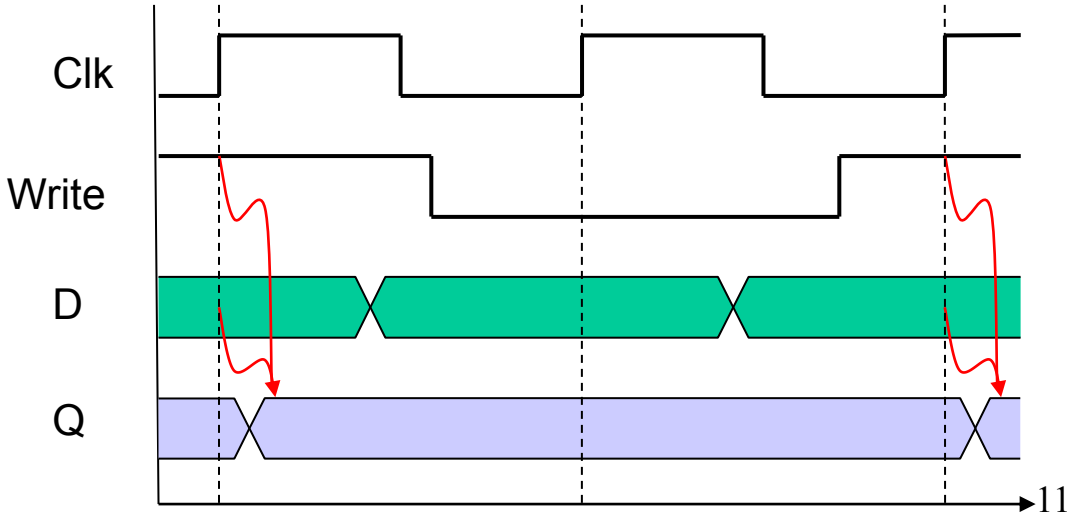


# State Elements with write control

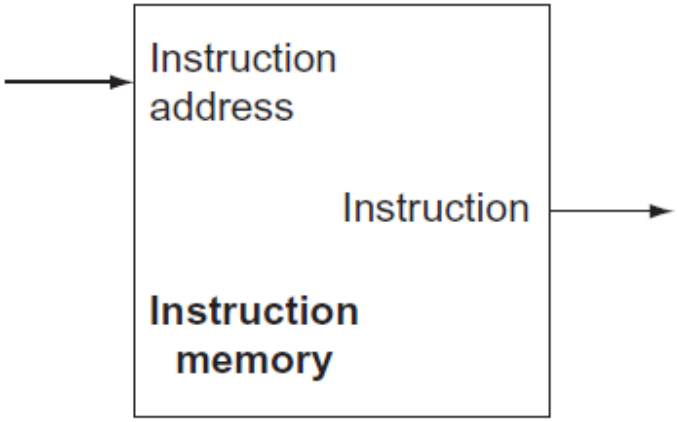
- Register without write control (e.g. program counter)
  - Uses a clock signal to determine when to update
  - Edge-triggered: update when Clk changes from 0 to 1
- Register with write control (e.g. data memory/register)
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later



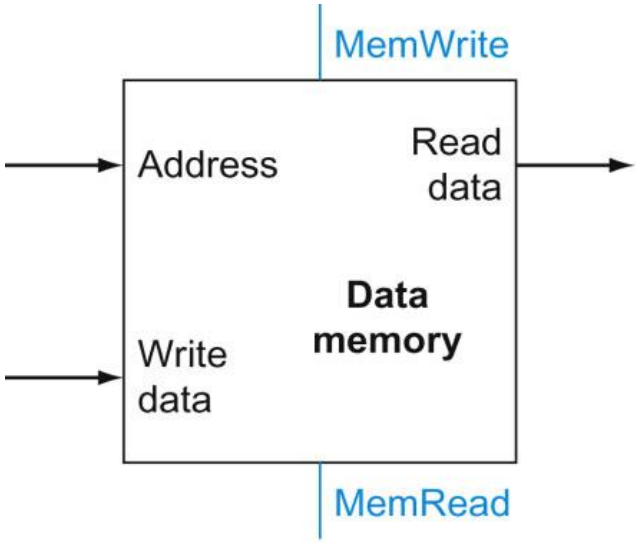
# Instruction Memory

- A state element with
  - Input: InstructionAddress (32-bit), clock
  - Output: Instructions (32-bit)
  - State: the instructions stored in the memory (n*32-bit)
- Why no control signals?
  - No write operation
  - Every clock read an instruction



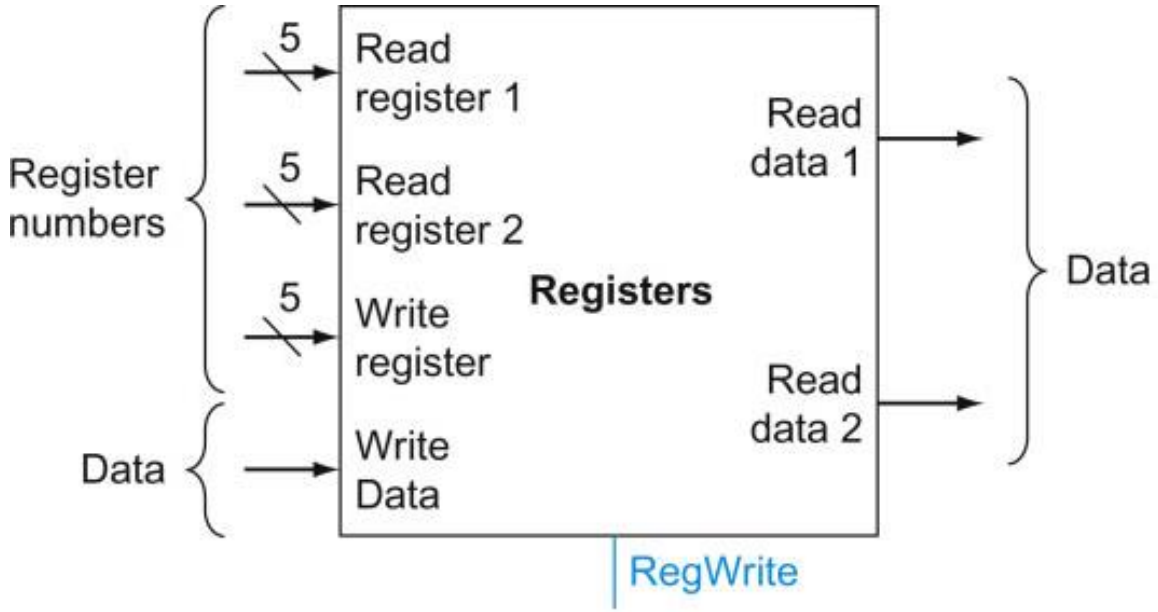
# Data Memory

- A state element with
  - Input: Address (32-bit), Write-in data (32-bit), MemWrite (1-bit), MemRead (1-bit), clock
  - Output: Read-out data (32-bit)
  - State: the data stored in the memory (n*32-bit)



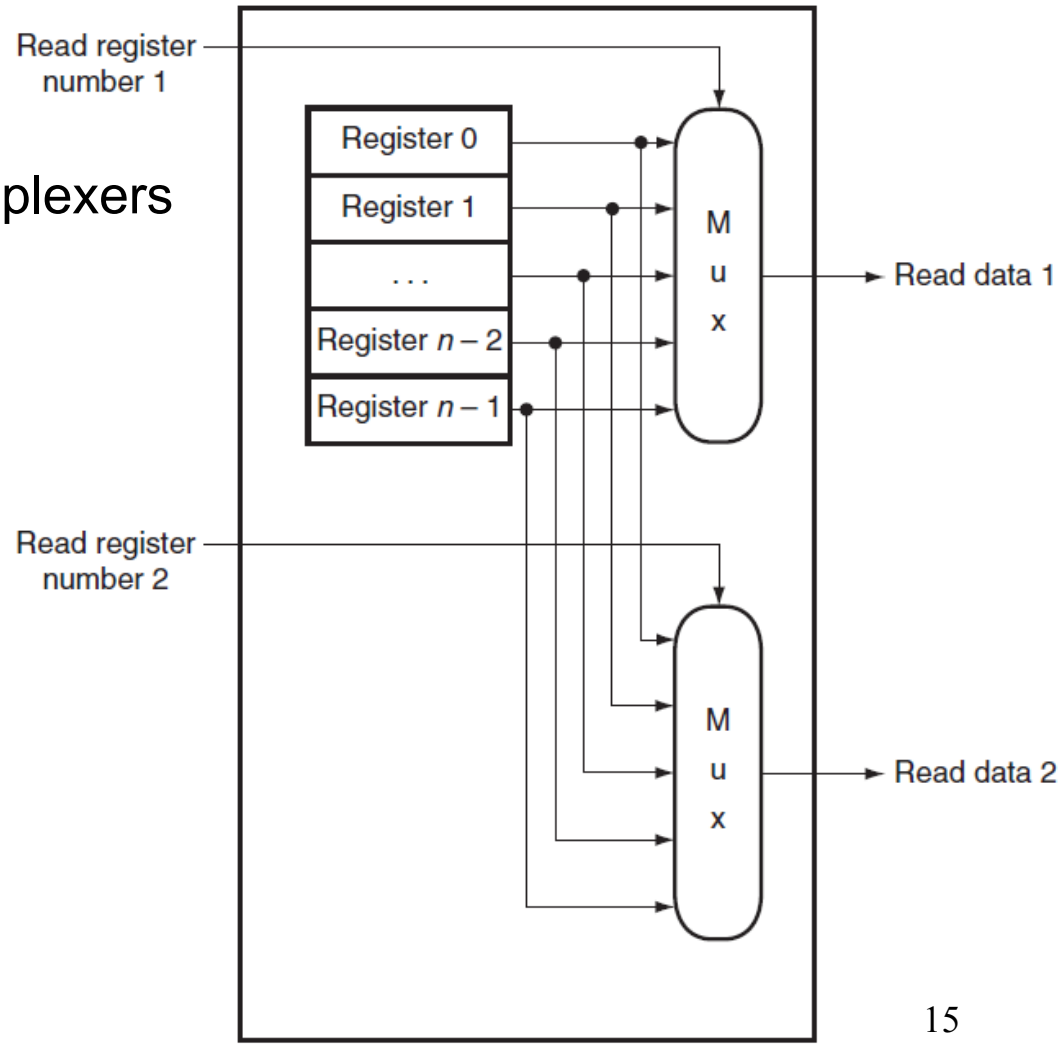
# Registers

- State element
  - Input: three register numbers (5-bit *3), write-in data (32-bit), RegWrite (1-bit)
  - Output: readdata1 (32-bit), readdata 2 (32-bit)
  - State: 32 * 32-bit data



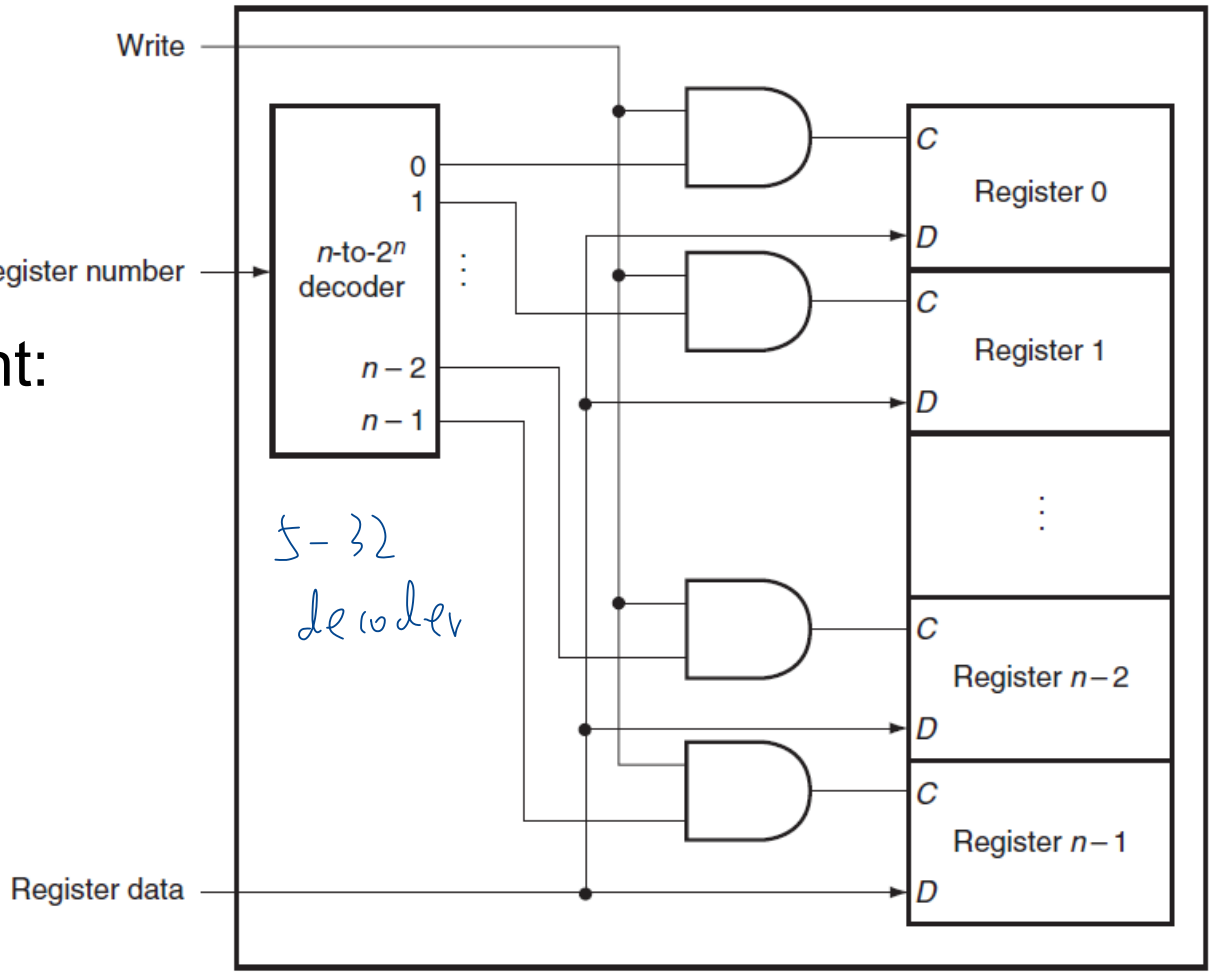
# Register Read

- Input: two addresses
- Output: two data
- Key component: two multiplexers



# Register Write

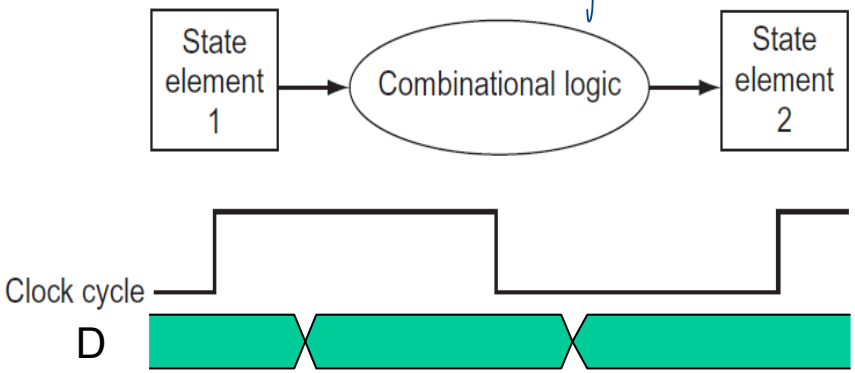
- Input:
  - write control
  - address
  - writing data
- Key component:
  - decoder



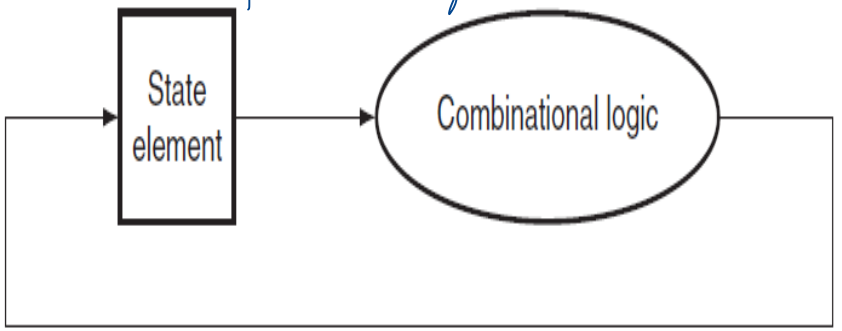
# Clocking Methodology

- Defines when signals can be read and when they can be written
- Edge-triggered clocking: all state changes occur on a clock edge.
- Clock time > the time needed for signals to propagate from SE1 through combinatorial element to SE2
- A state element can be read and written in the same clock cycle without creating a race, but the clock cycle should be long enough

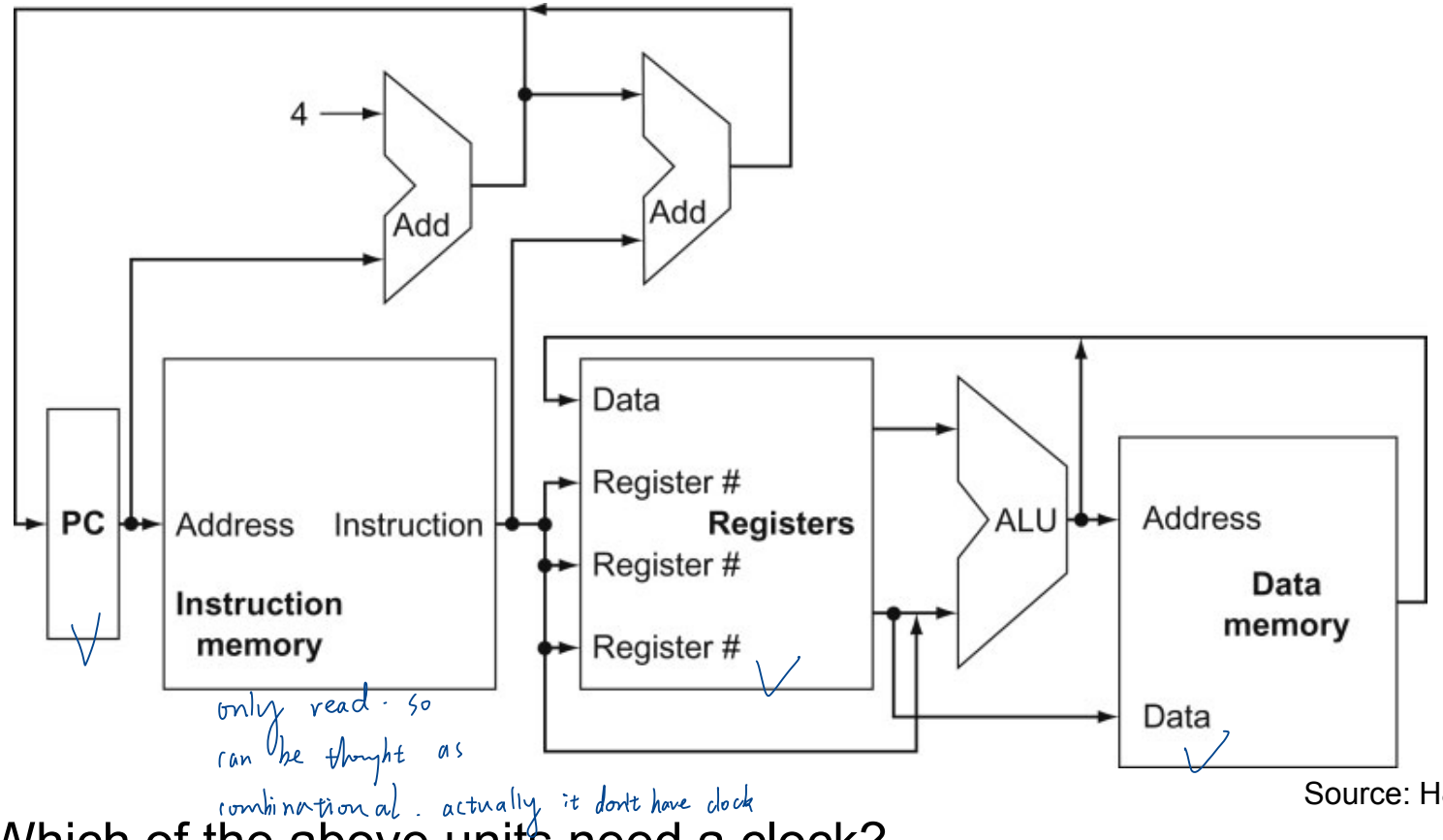
change smaller than 1 cycle




else
blz: can't implement as it can't completed in 1 cycle



# Clocking Methodology




Source: H&P textbook

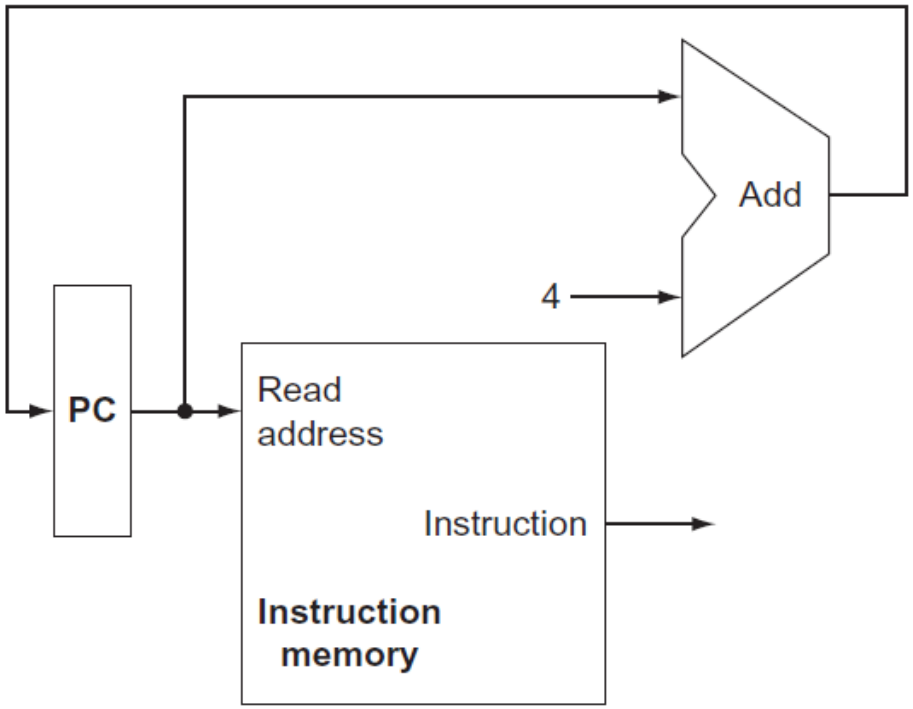
- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?
  Keep in mind that the latched value remains there for an entire cycle

# Outline

- Implementation overview
- Logic design basics
- Detailed implementation for every instruction
  - Fetch instructions
  - R-type
  - Load/store-type
  - J-type

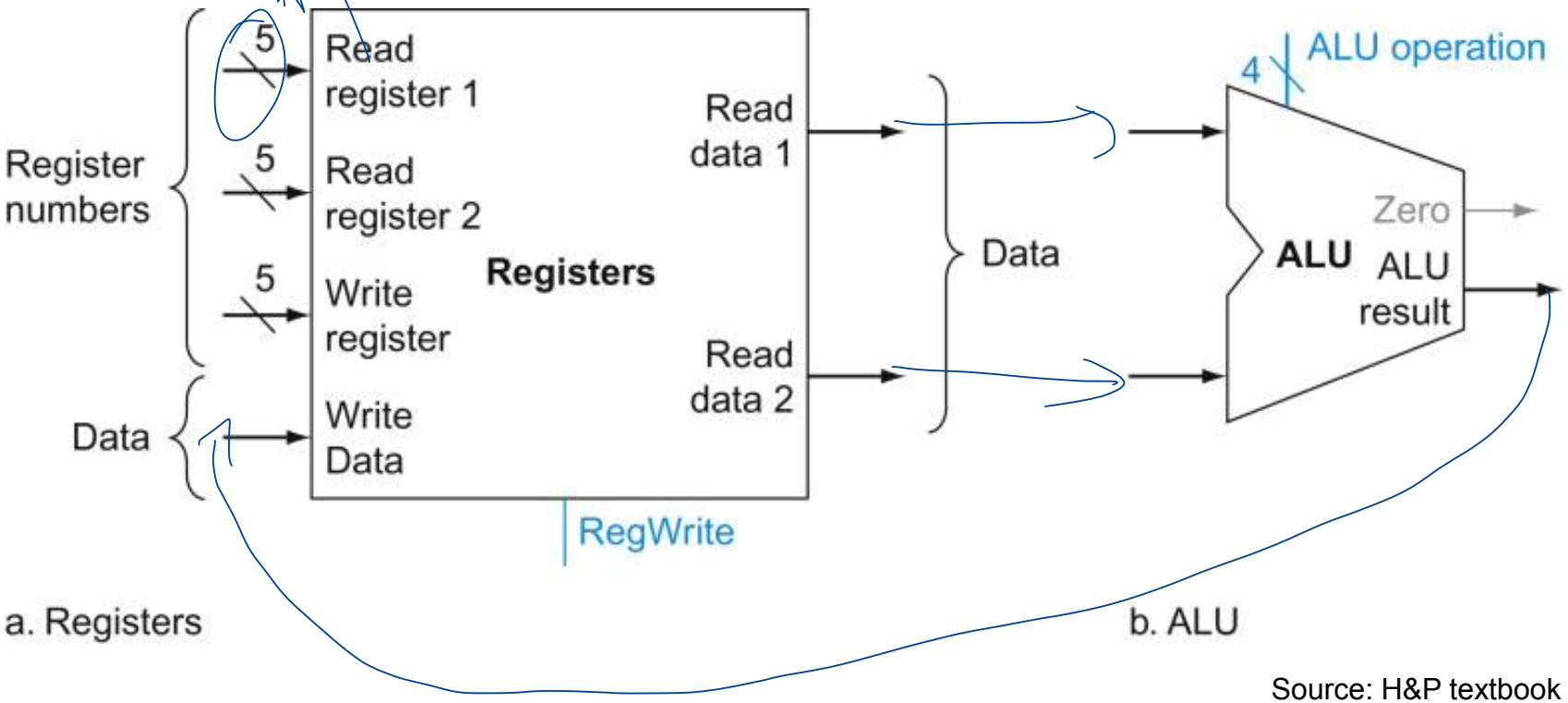
# Fetch Instructions

- The instruction is fetched from I-mem and the PC is added by 4
- Components: program counter, instruction memory, adder
  - The PC is a 32-bit register, written at every positive edge of the clock, thus, it does not need write control signal.



# Implementing R-type Instructions

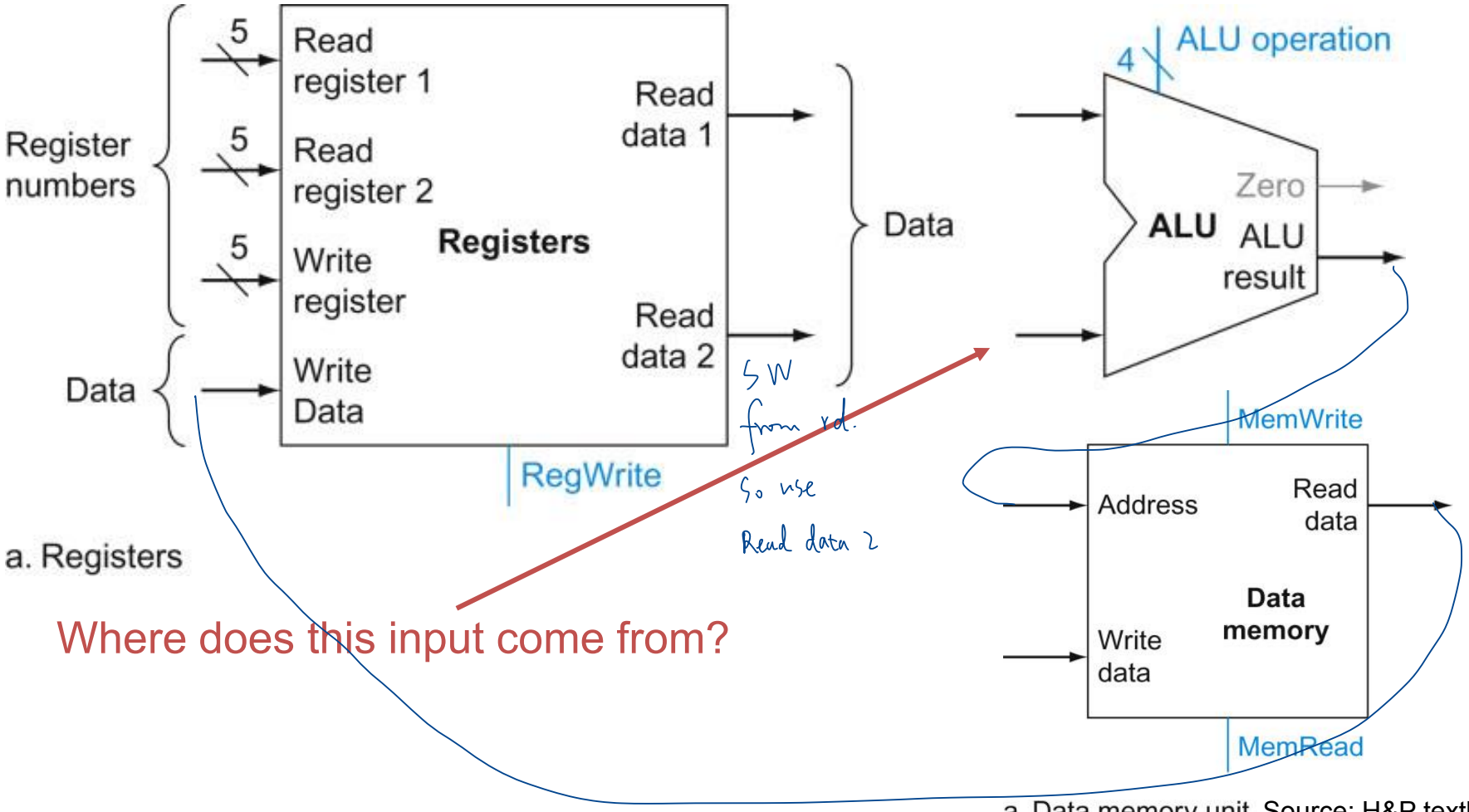
- Instructions of the form add $t1, $t2, $t3
- Explain the role of each signal




Source: H&P textbook

# Implementing Loads/Stores

- Instructions of the form lw $t1, 8($t2) and sw $t1, 8($t2)

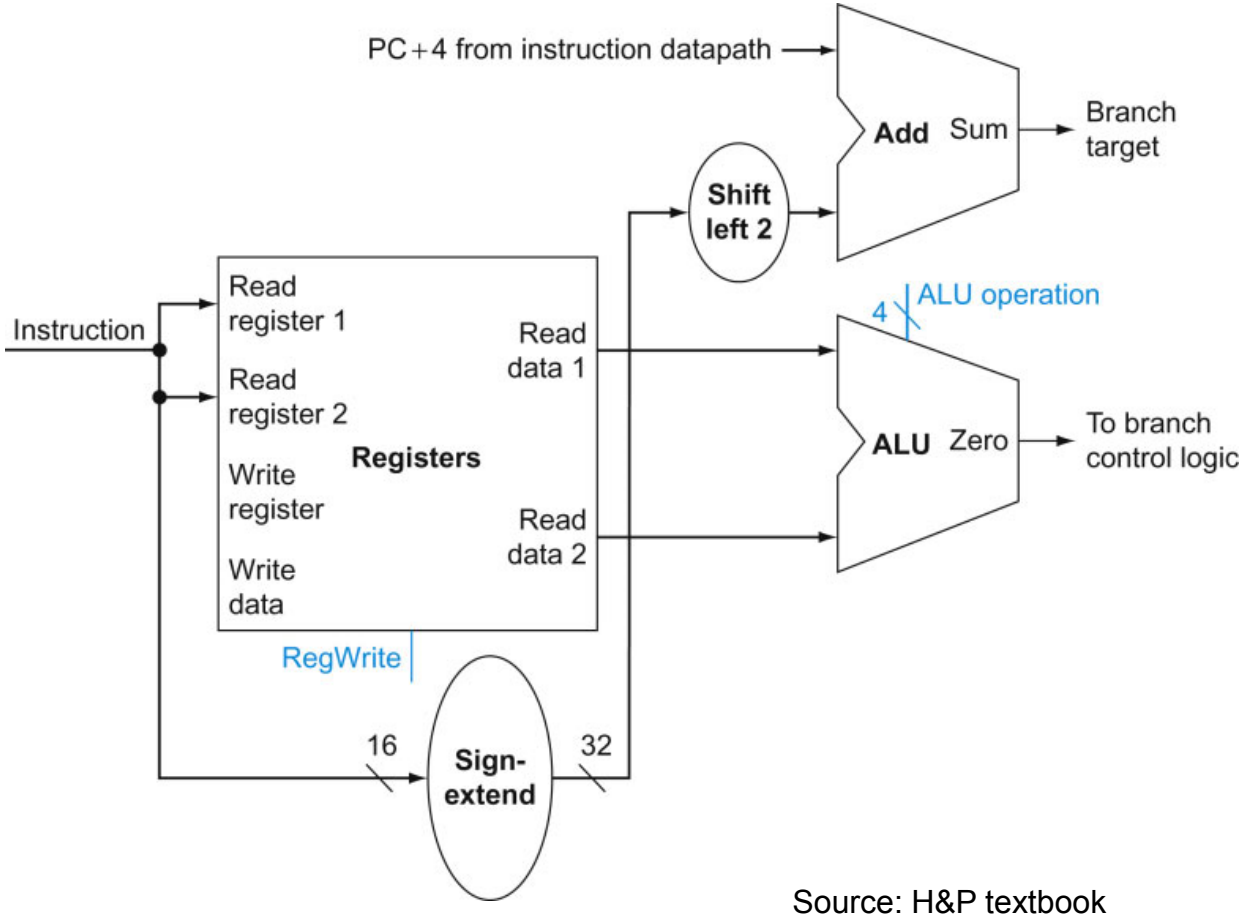



Where does this input come from?

Source: H&P textbook

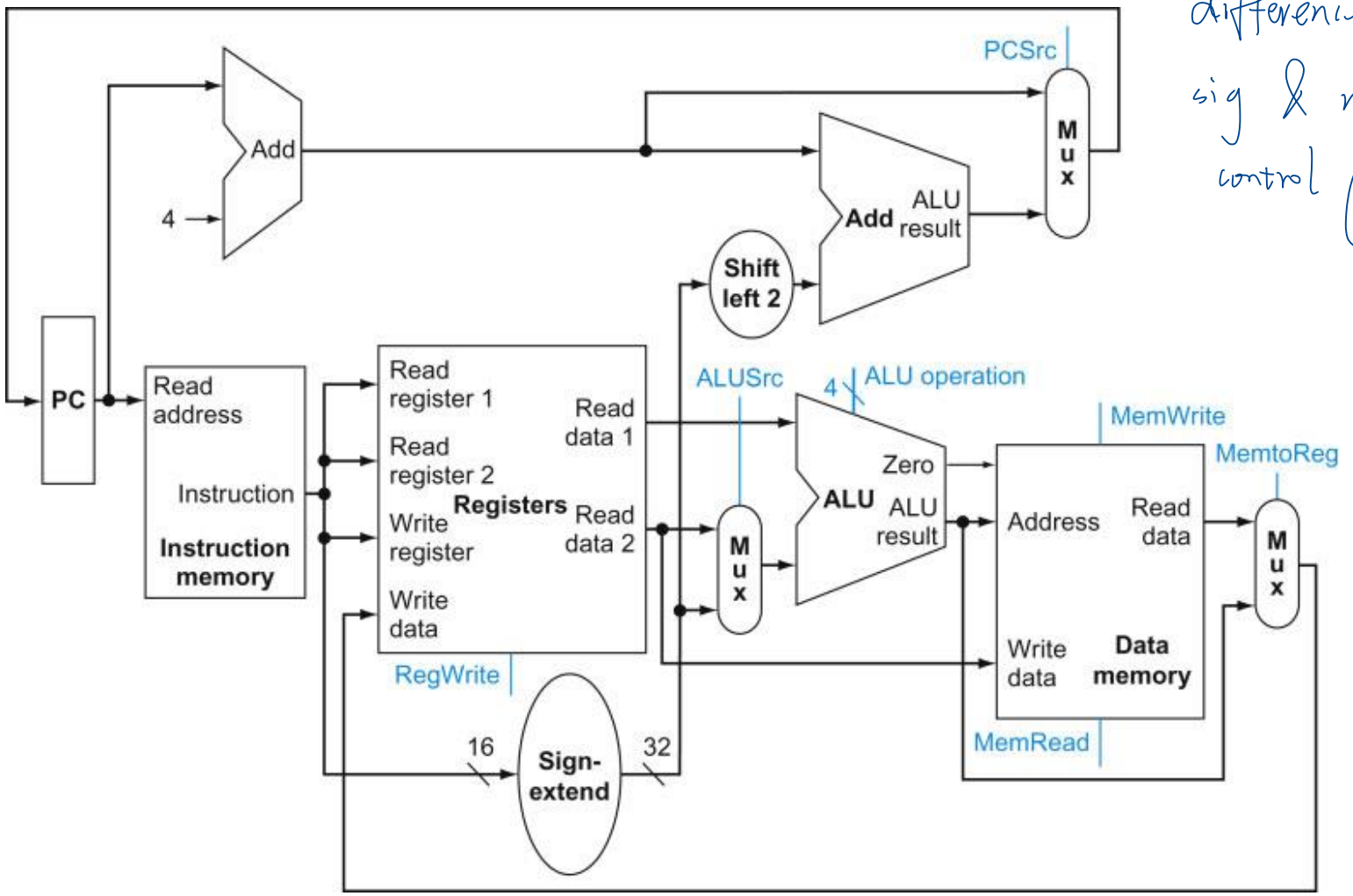
# Implementing Cond-Branch Instructions

- Instructions of the form beq $t1, $t2, offset




Source: H&P textbook

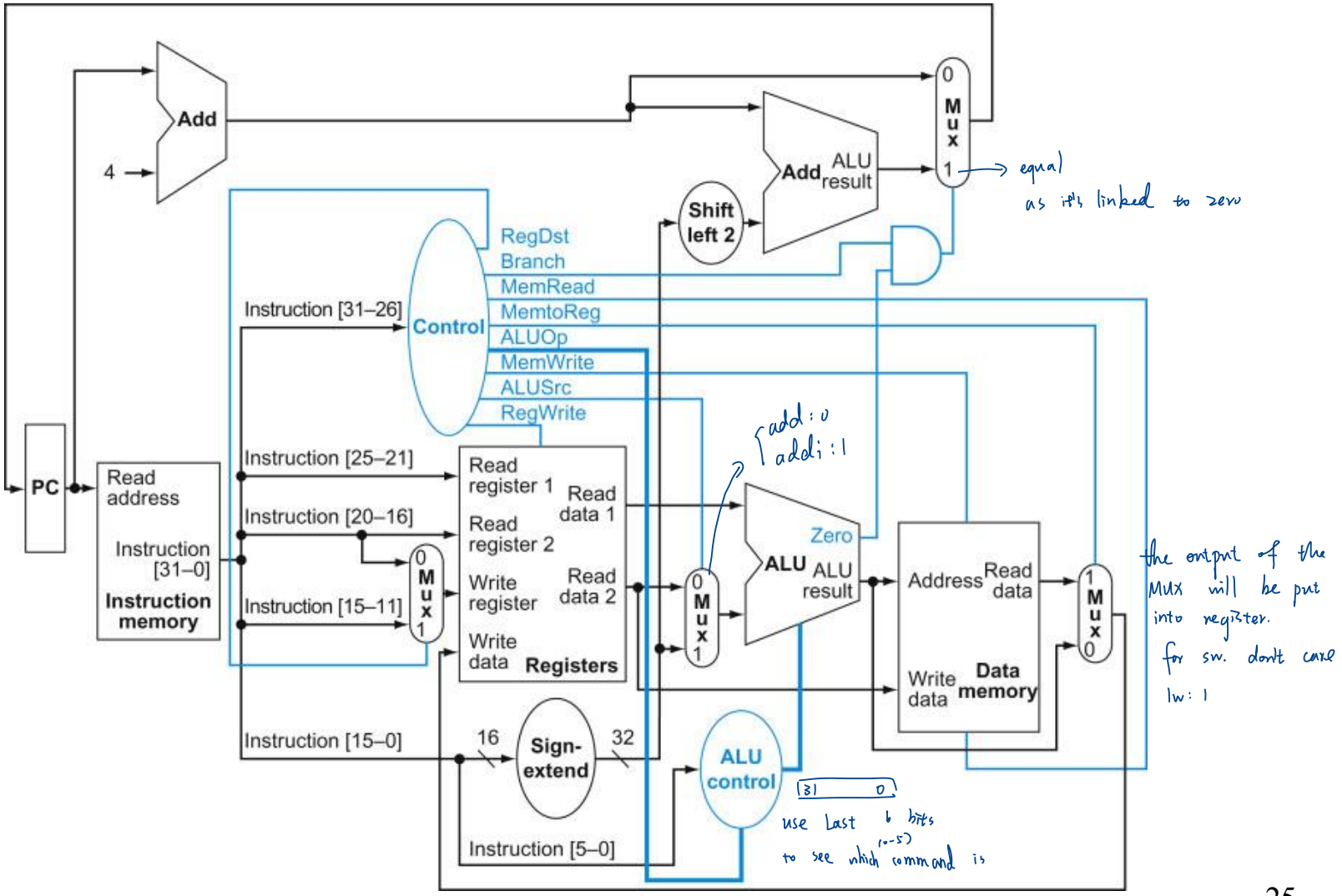
# View from 10,000 Feet




difference sig & mux control (in blue color)

Source: H&P textbook

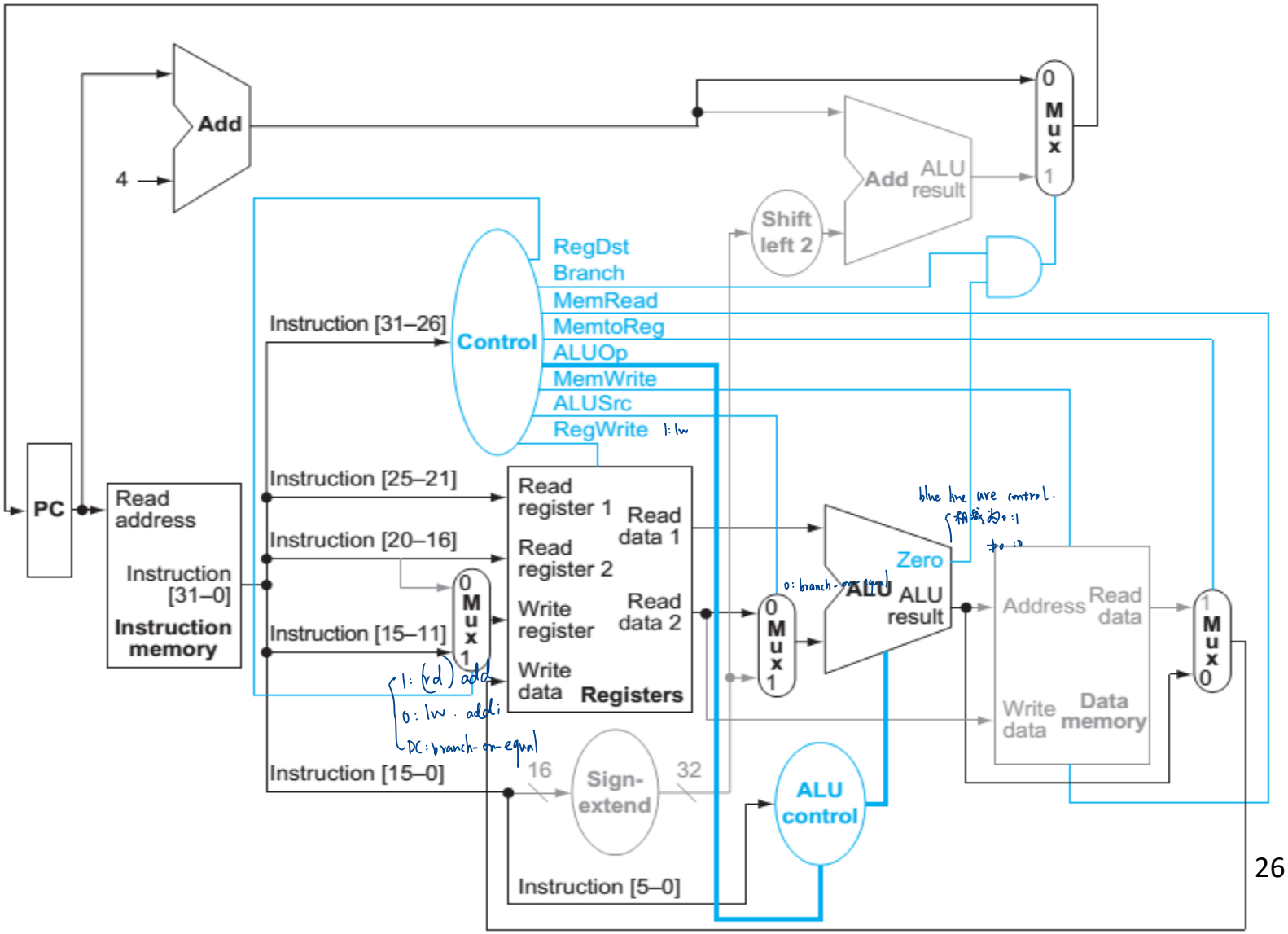
# View from 5,000 Feet



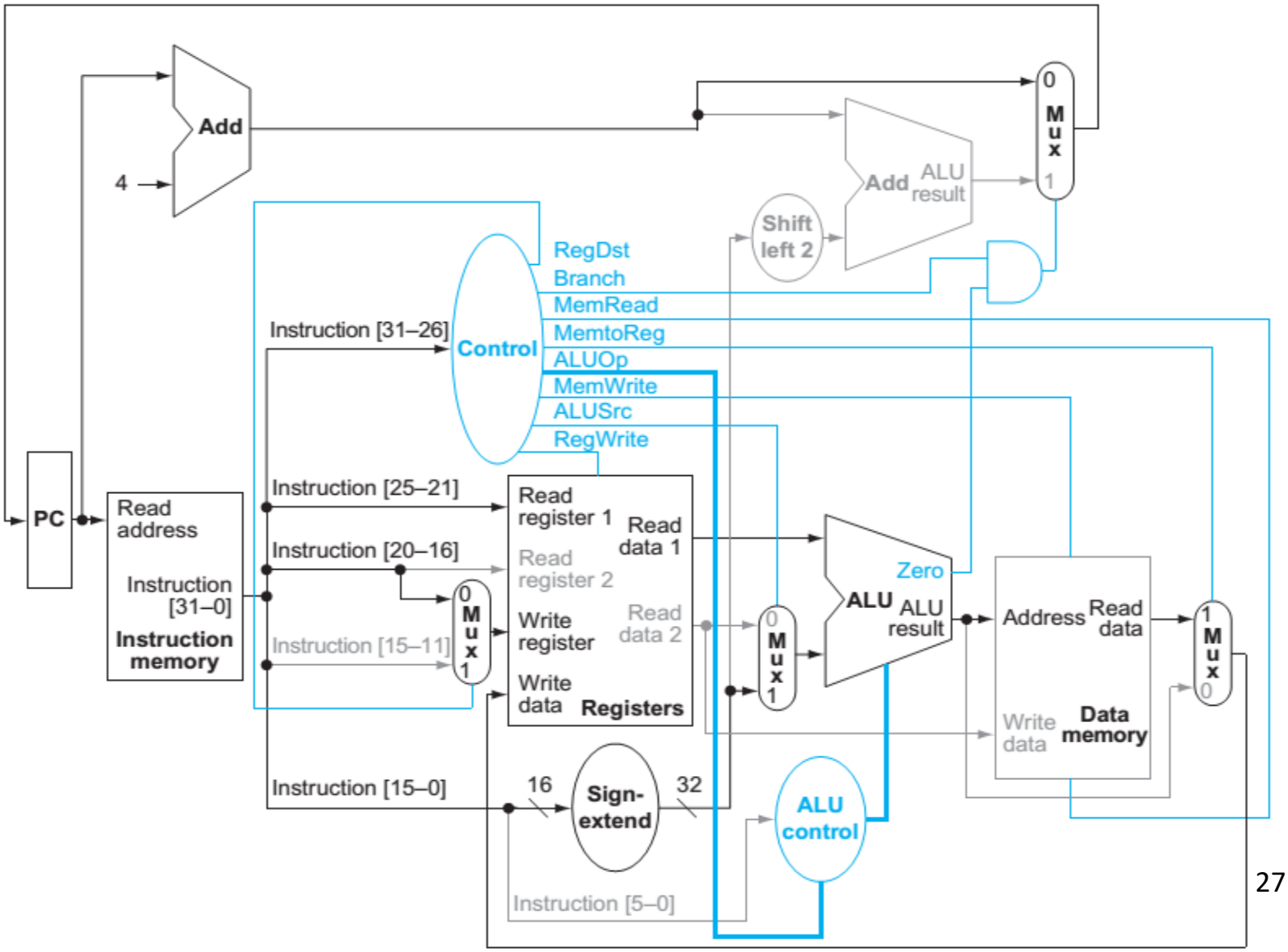



Source: H&P textbook

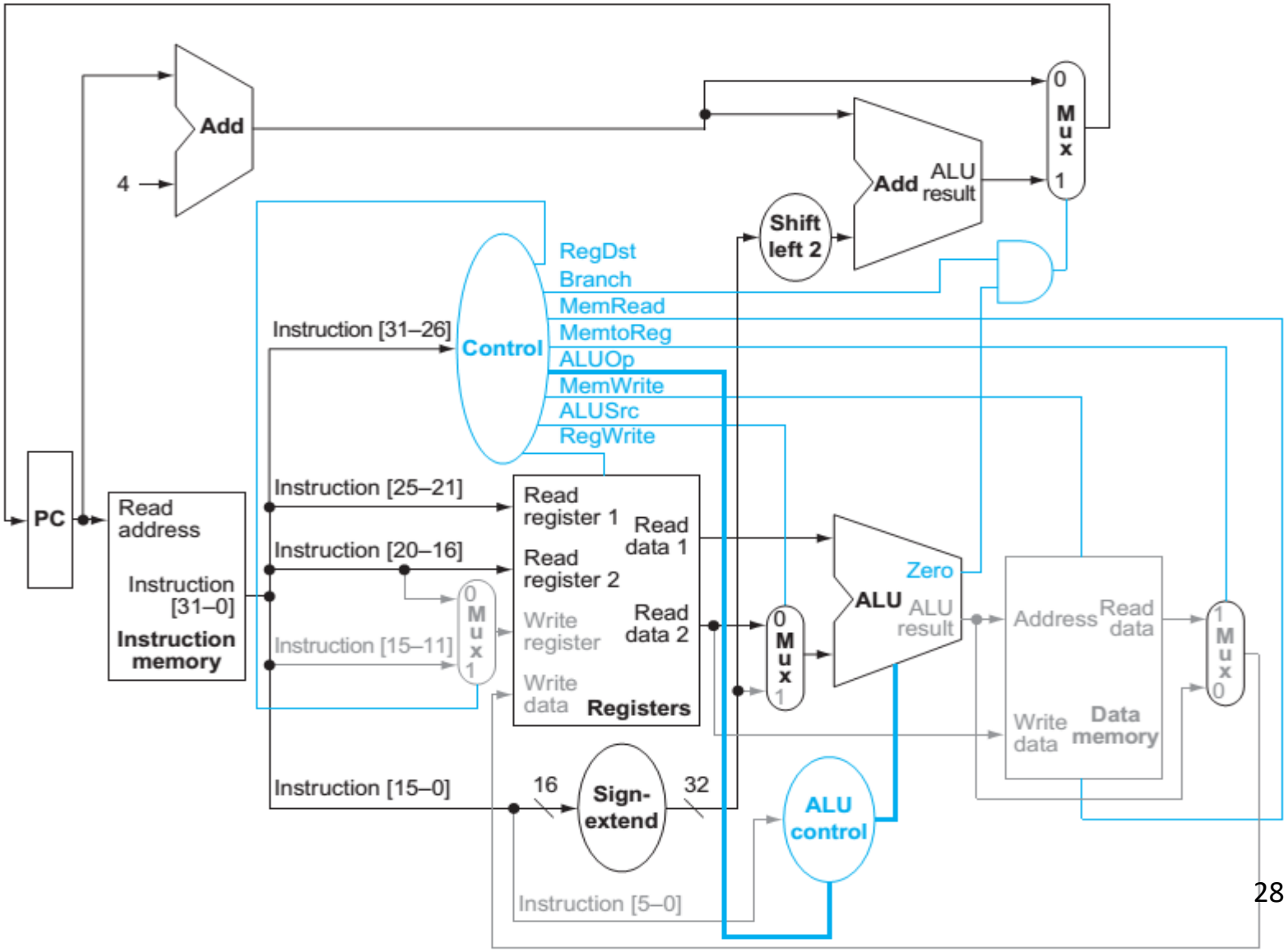
# Datapath for an R-type Instruction



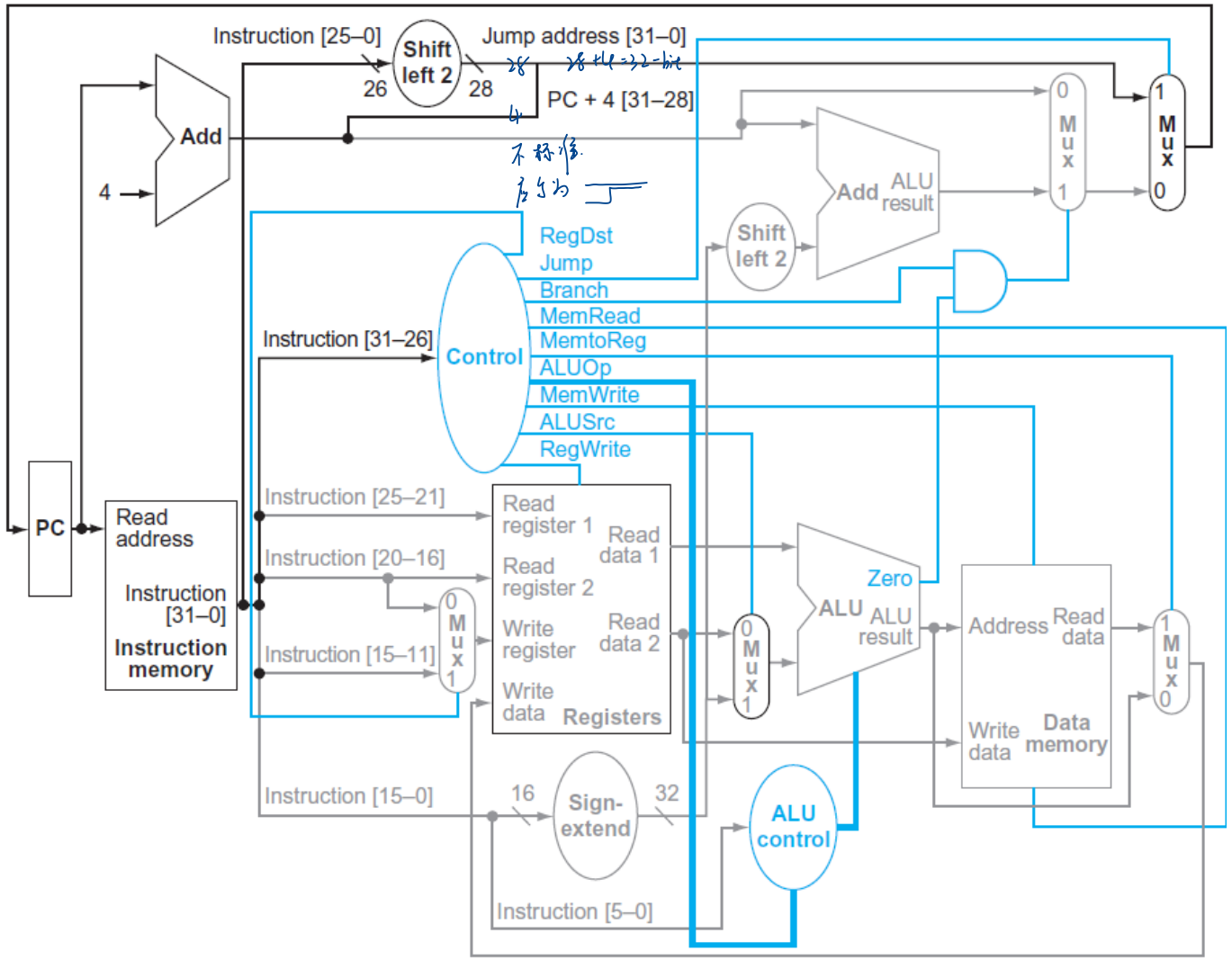
# Datapath for a load Instruction



# Datapath for a Branch-on-equal Instruction



# Datapath for Jump



# Truth Table for Control Unit

| Input or output | Signal name | R-format | lw | sw | beq |
|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 |
| first 6 bit operation code | 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 |

# Midterm

- April 15 (Sat.) 16:30-18:30, room 106,107, 3rd Teaching Building
- Closed-book, no calculator allowed, a reference page will be provided.
- Content: week 1-8, Section 1.1-4.4
- No lecture on week 9.
- Q&A session: April 14 (Fri.) 19:00-21:00, room 443B, CoE South