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)

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
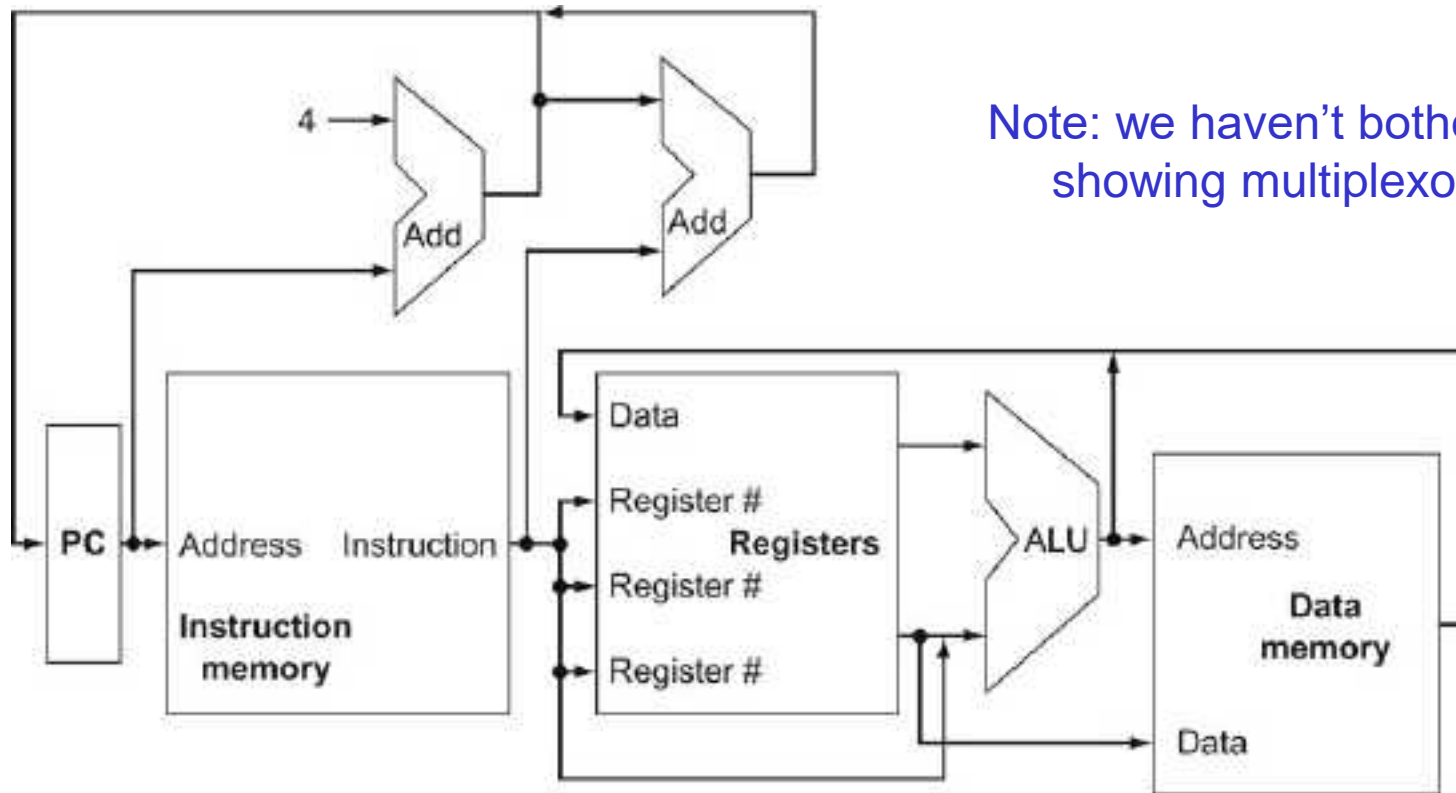
Source: H&P textbook

- 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

6

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
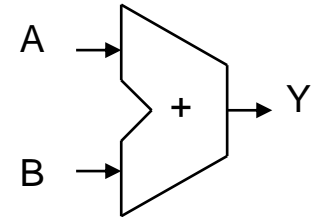   - Store information
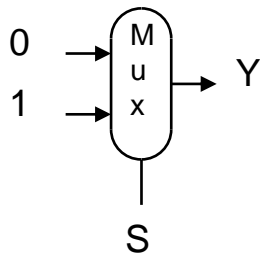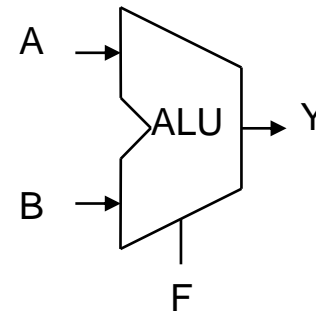
Combinational Elements

- ## And gate
  - Y = A & B

A
B ─── Y

- ## Multiplexer
  - Y = S ? 1 : 0

0 →
1 →
Mux → Y
S

- ## Adder
  - Y = A + B

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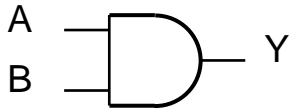
- ## Arithmetic/Logic Unit
  - Y = F(A, B)

A →
B →
ALU → Y
F

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
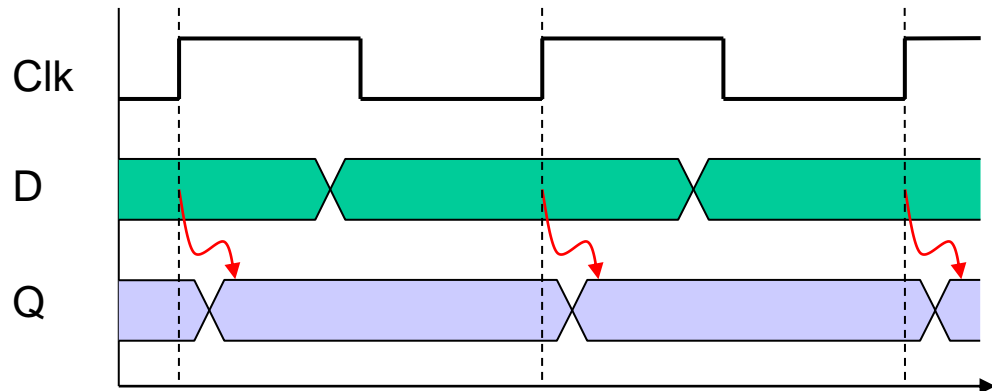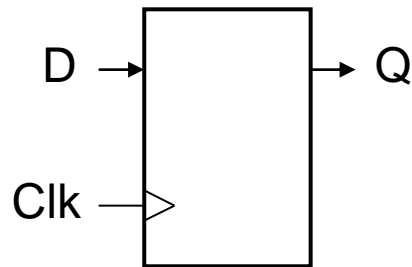
State Elements with write control

- 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
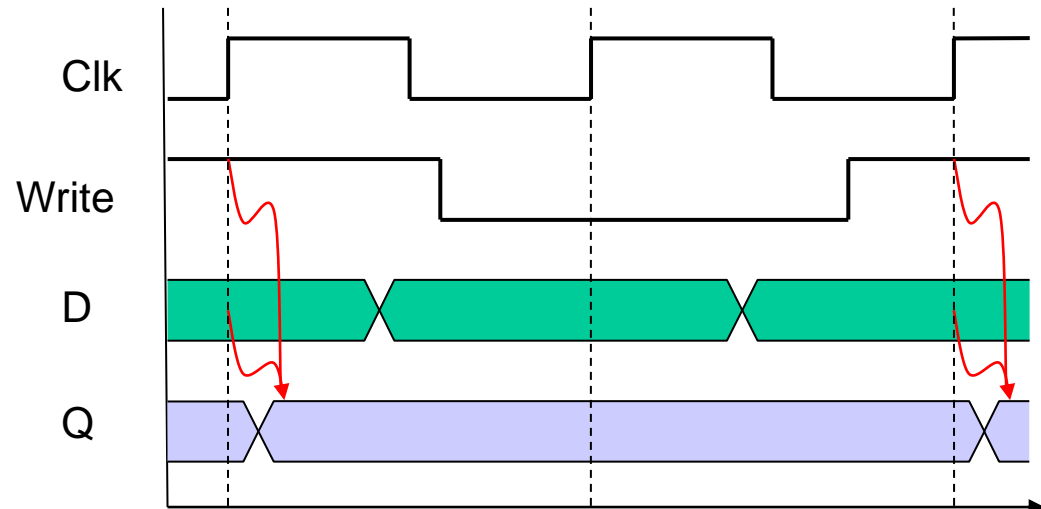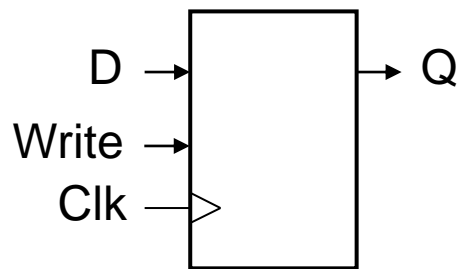
Instruction Memory

- 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

```
          ┌─────────────────┐
          │ Instruction     │
   ──────→│ address         │
          │                 │
          │          Instruction ──────→
          │                 │
          │ Instruction     │
          │  memory         │
          └─────────────────┘
```
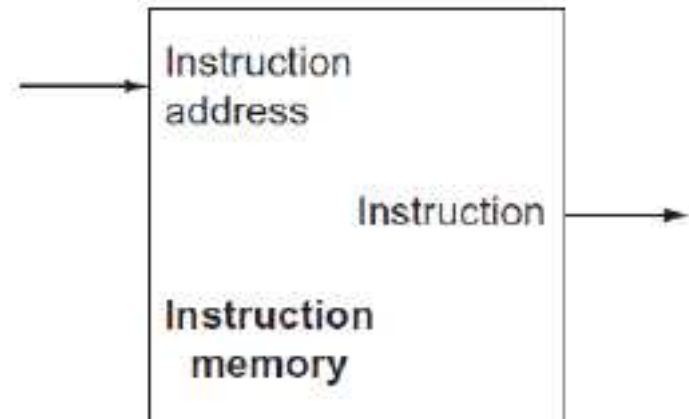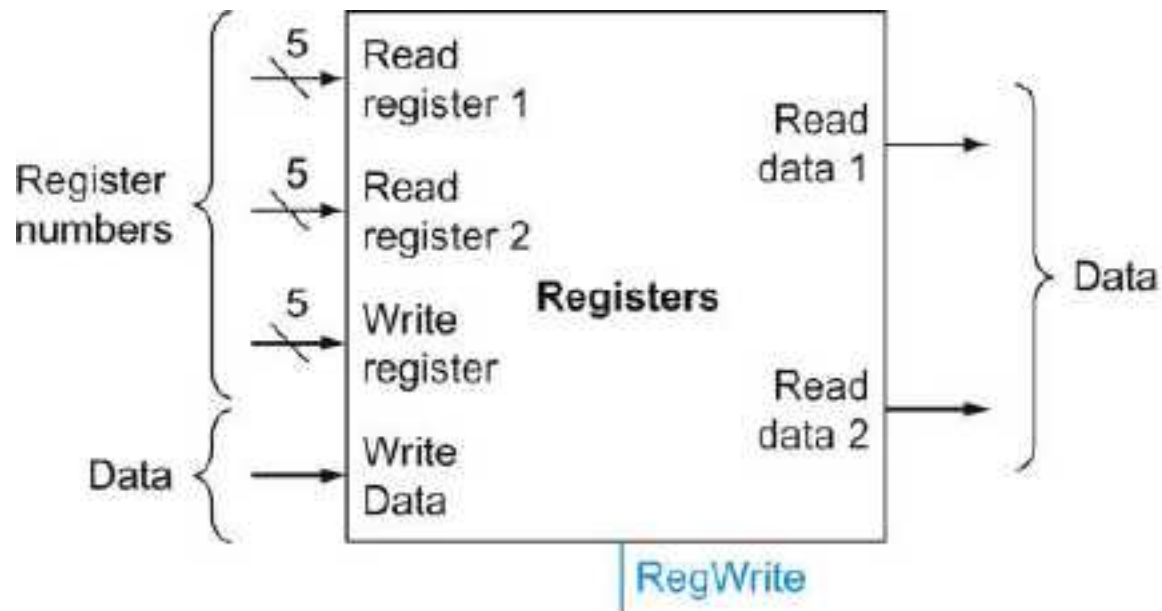
Data Memory

- 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)

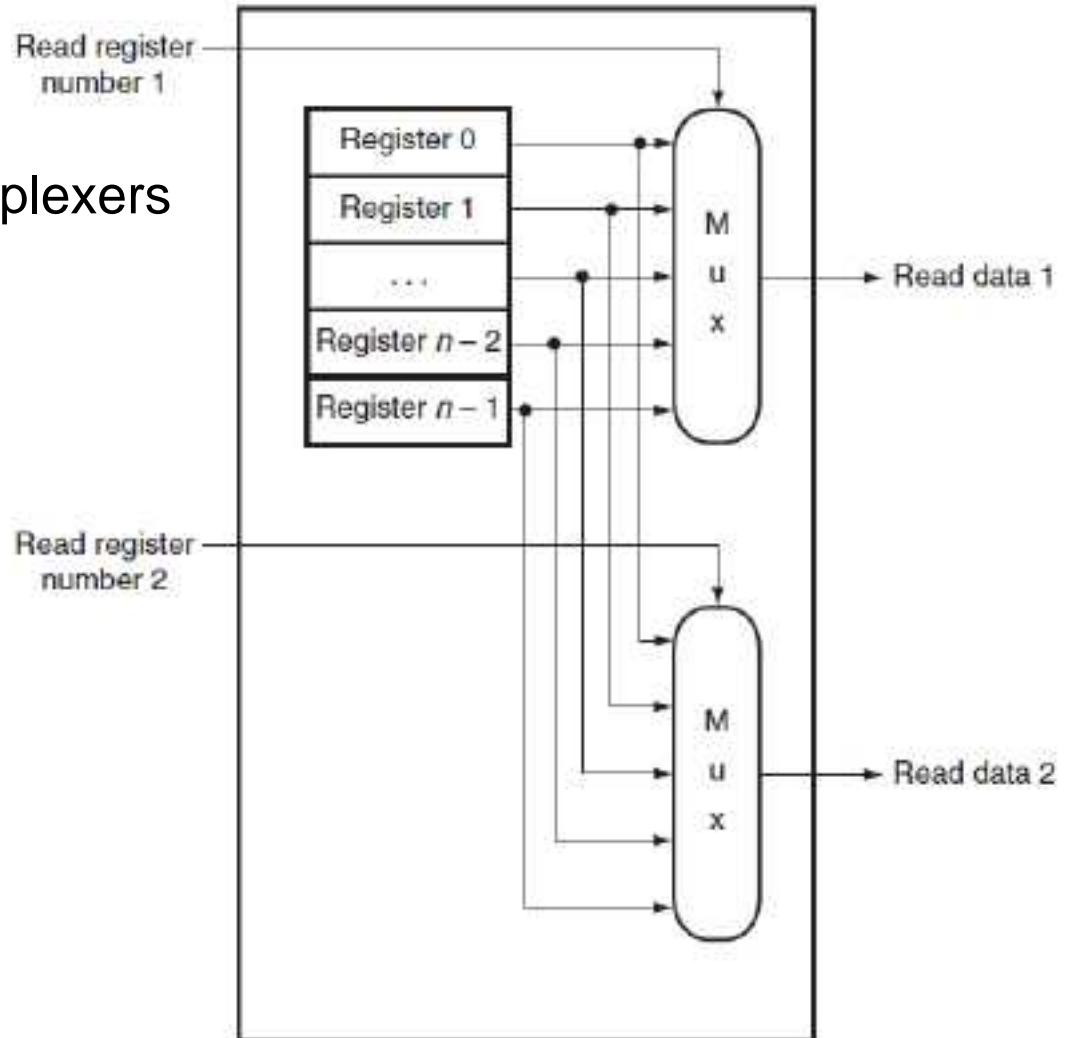- State: the data stored in the memory (n*32-bit)
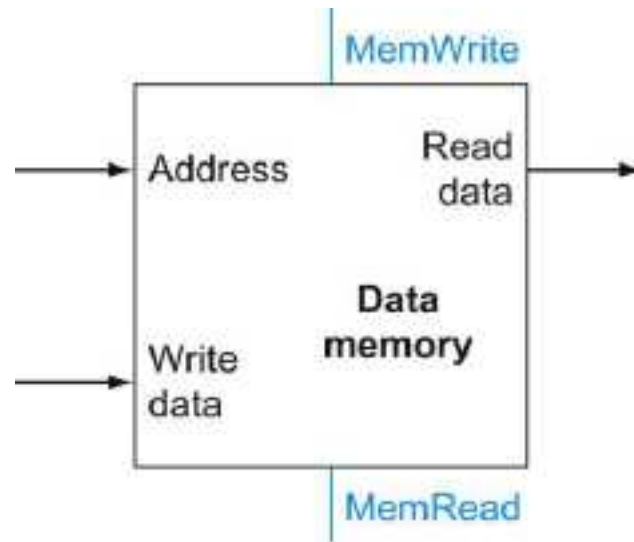
Registers

- State element

- Input: three register numbers (5-bit *3), write-in data (32-bit), RegWrite (1-bit)

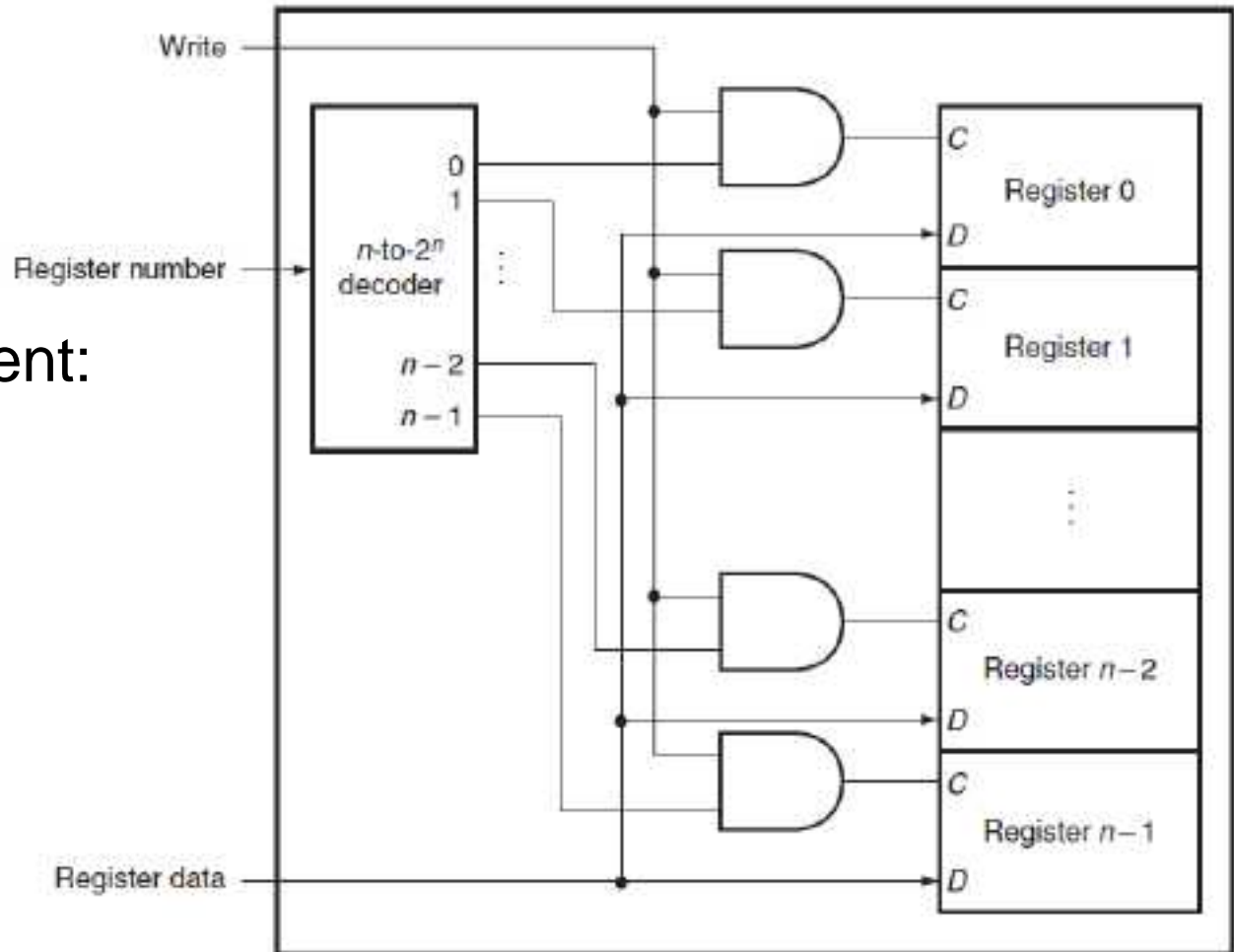- Output: readdata1 (32-bit), readdata 2 (32-bit)

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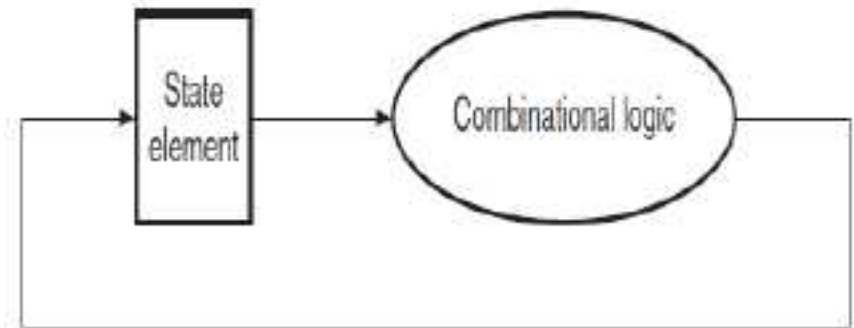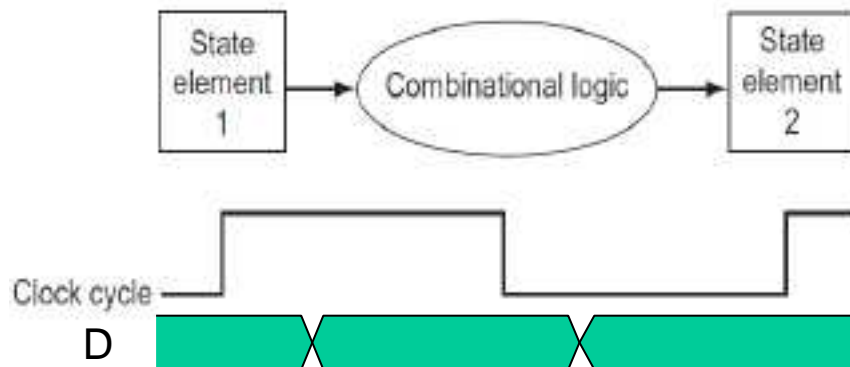
Clocking Methodology
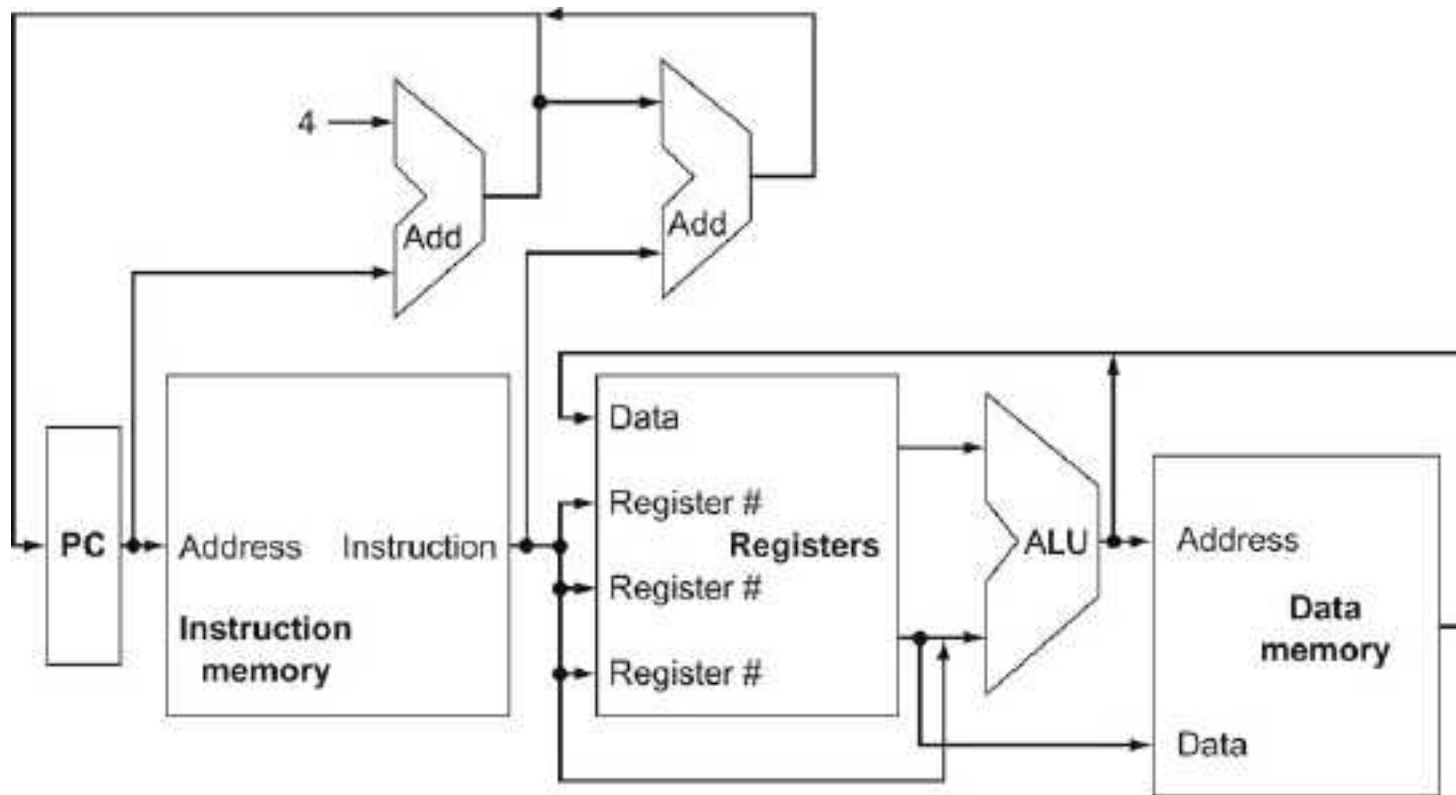
- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?
  Keep in mind that the latched value remains there for an entire cycle

18

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,

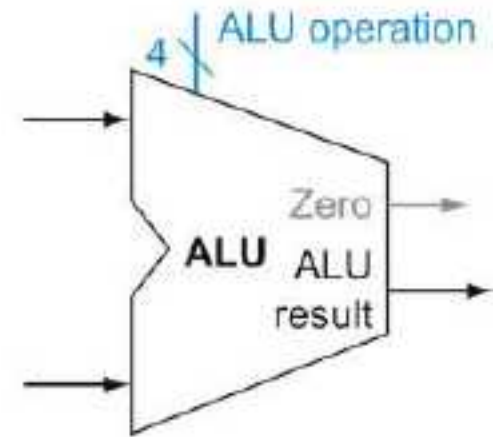   thus, it does not need write control signal.

20
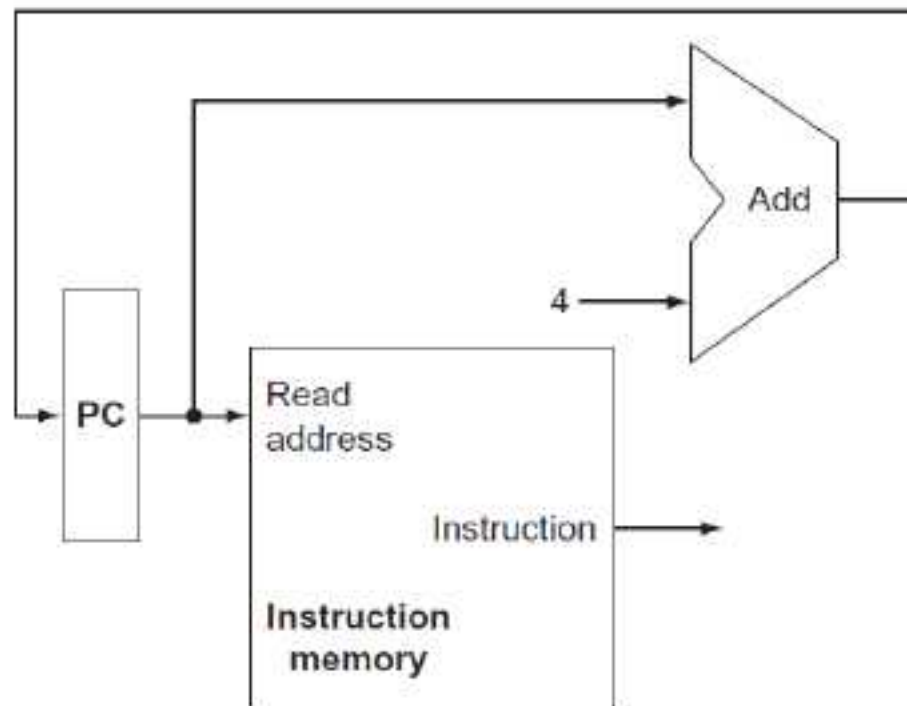
Implementing R-type Instructions

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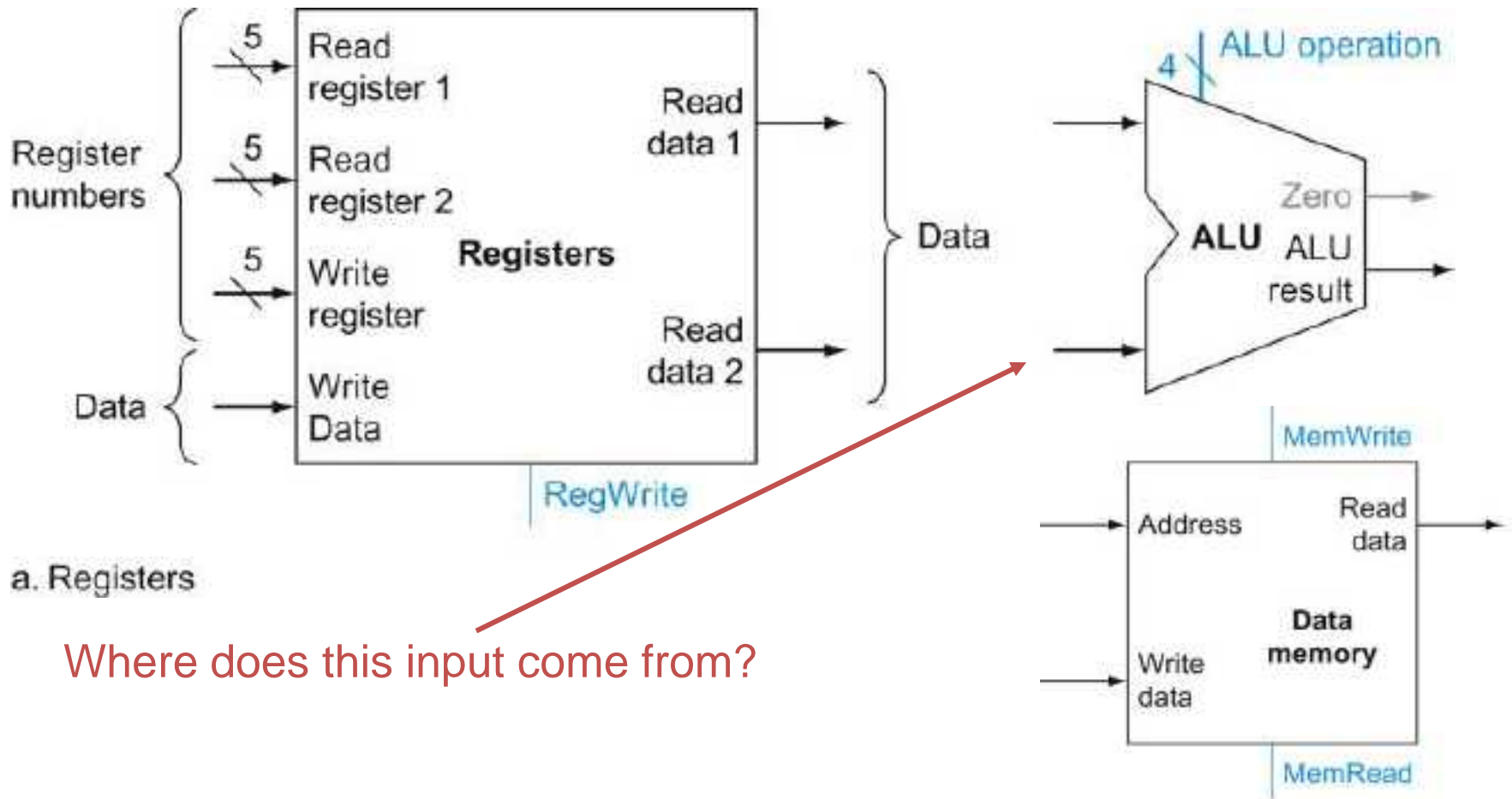- Explain the role of each signal
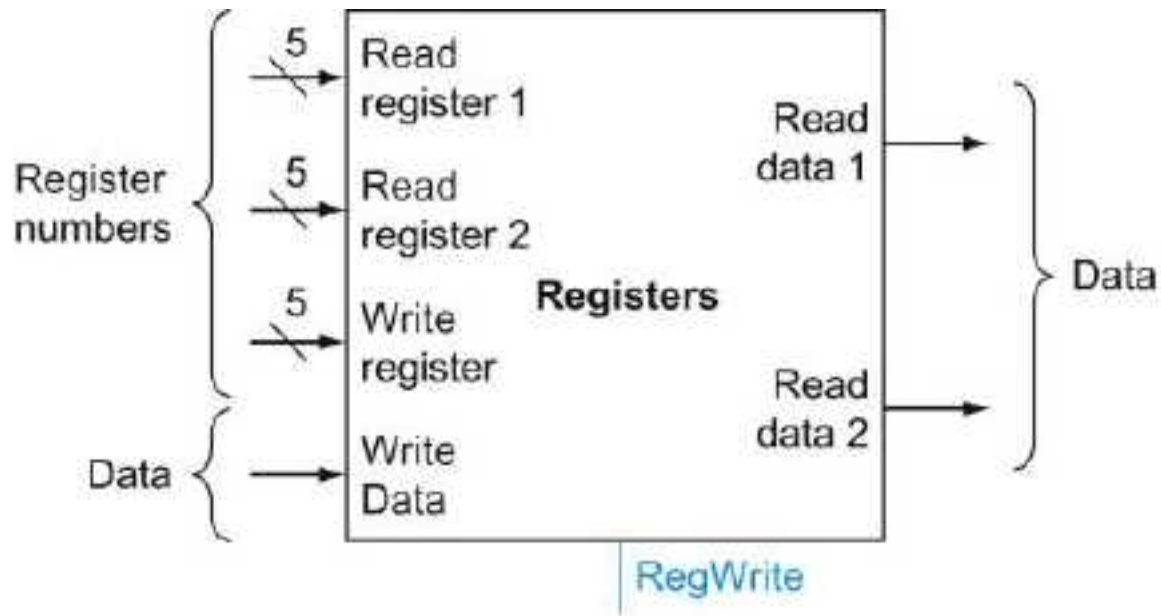
a. Registers

b. ALU
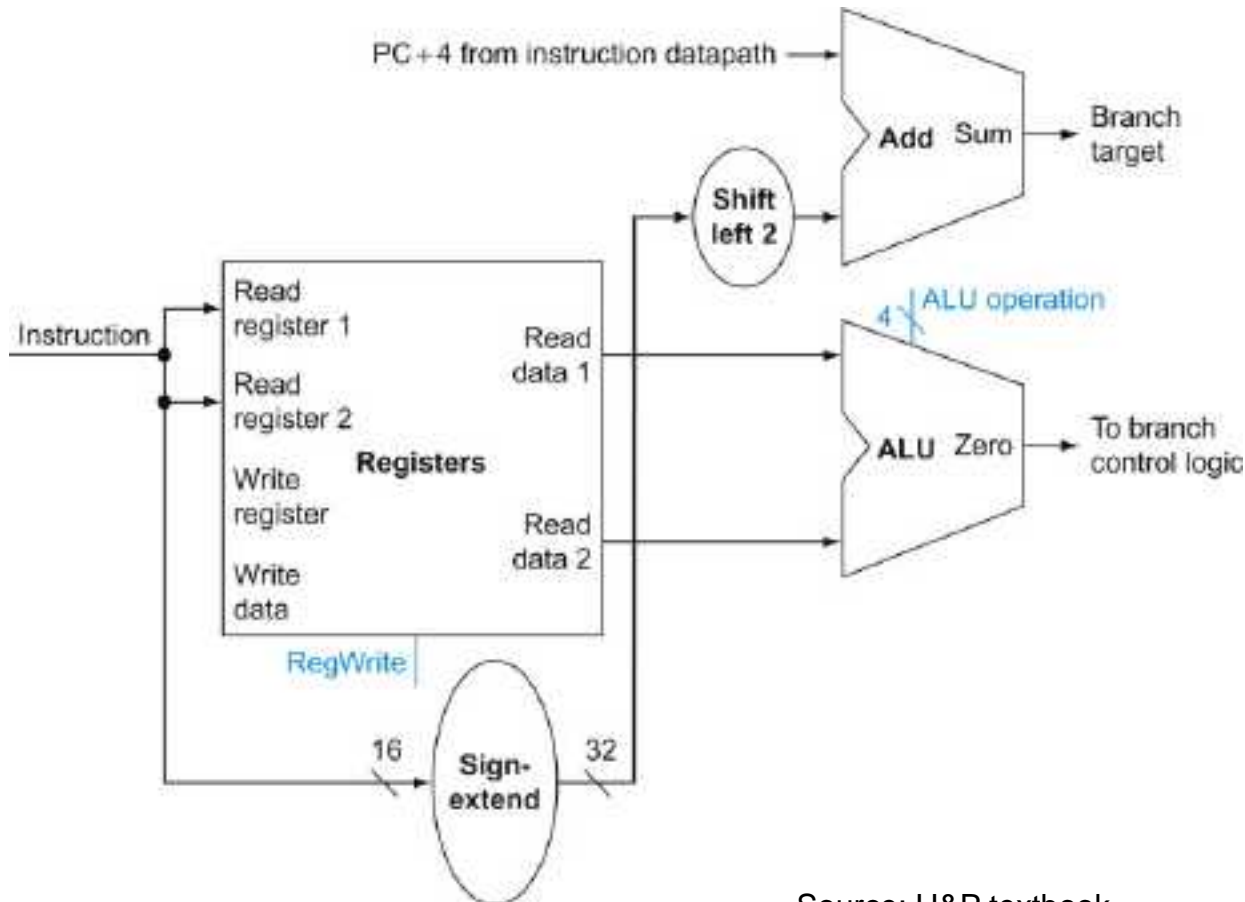
Source: H&P textbook

Implementing Loads/Stores

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

a. Registers

Where does this input come from?

a. Data memory unit Source: H&P textbook
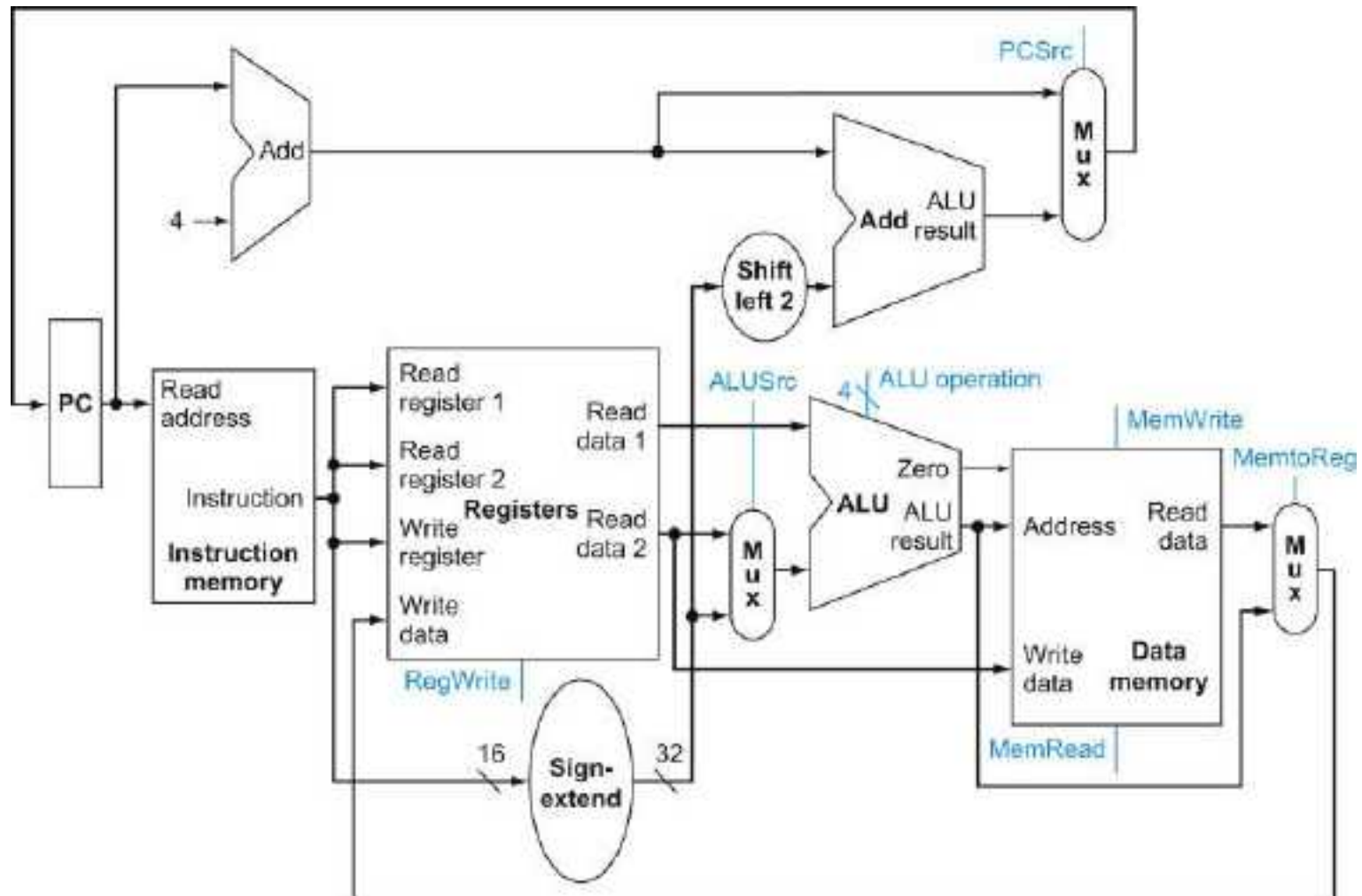
Implementing Cond-Branch Instructions

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

Source: H&P textbook

View from 10,000 Feet

View from 5,000 Feet

Source: H&P textbook

25
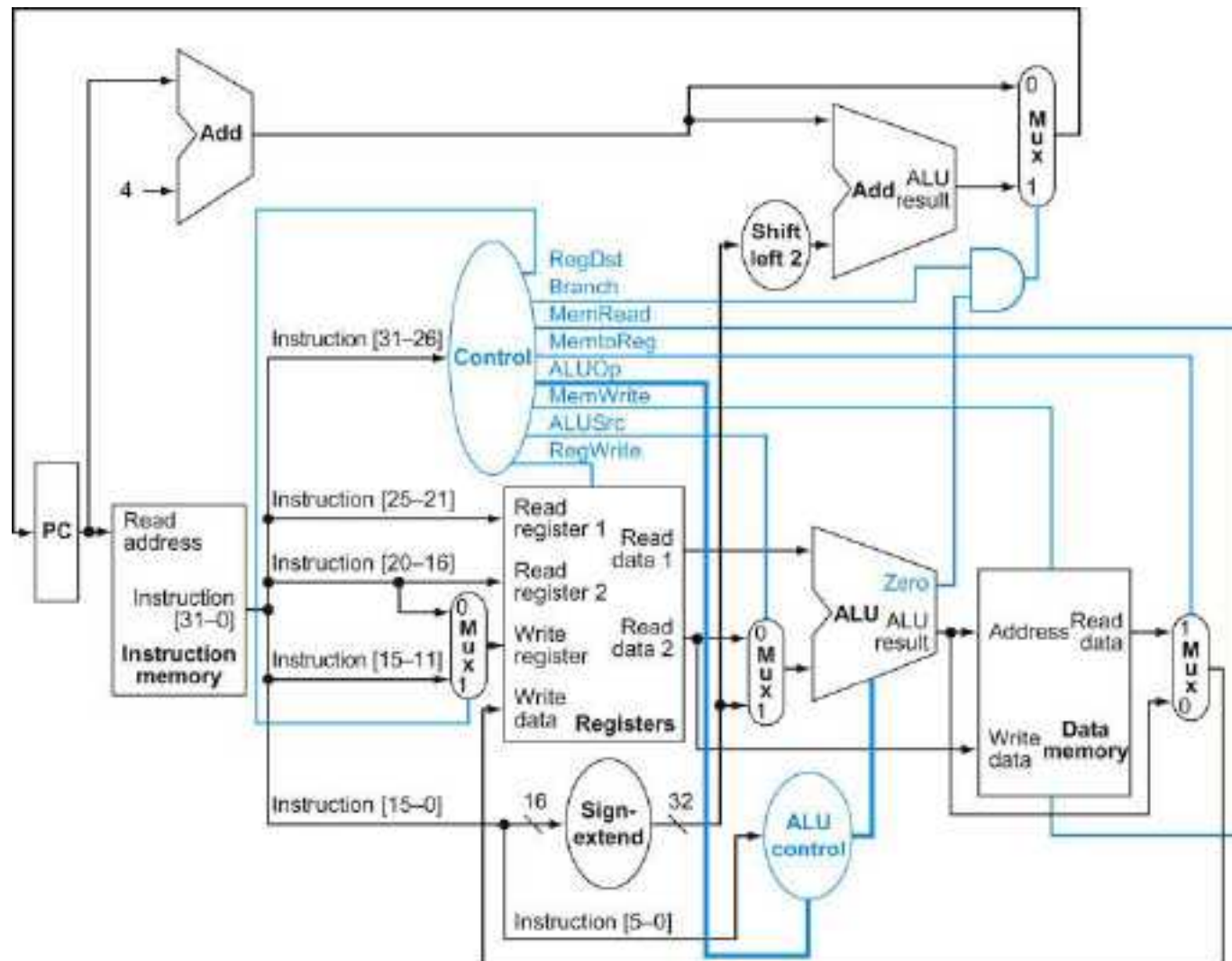
Practice and Midterm

- Practice: Textbook Problem 4.4 4.5 4.7

- Midterm Homework: Apr. 9 (Sat.) 2pm-4pm.

- Content: Section 1.1- 4.4.

- Q & A session: Apr. 8 (Fri.) 9-11pm