

# Clocks

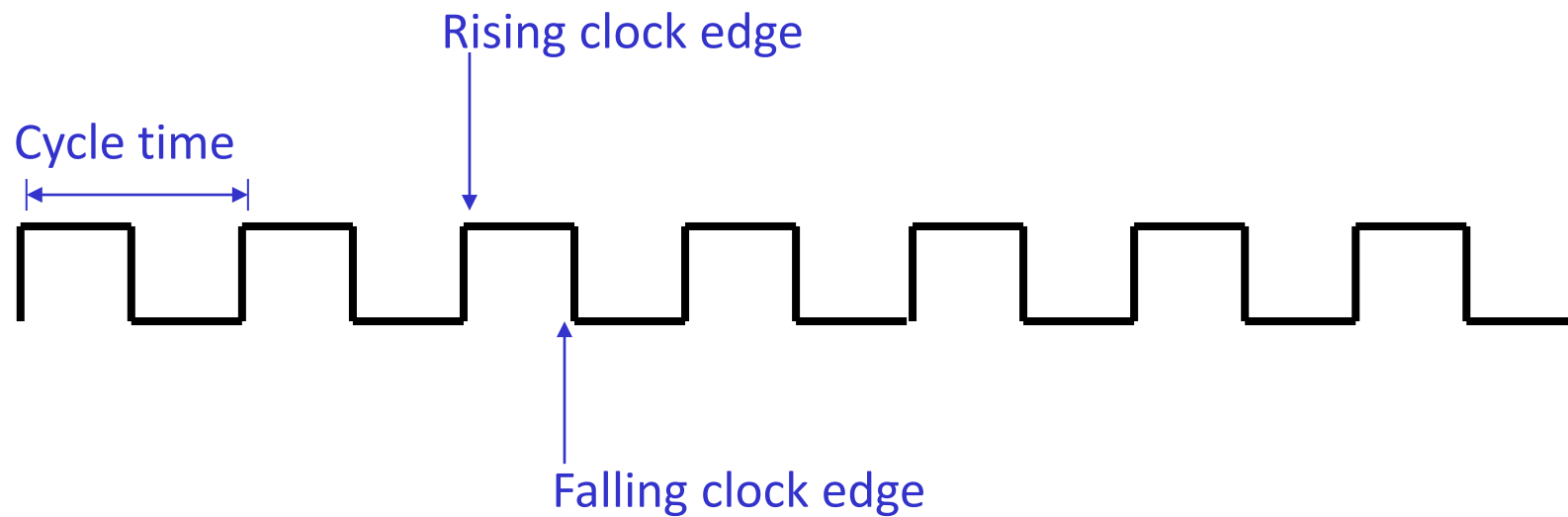
---

- A microprocessor is composed of many different circuits that are operating simultaneously – if each circuit  $X$  takes in inputs at time  $TI_x$ , takes time  $TE_x$  to execute the logic, and produces outputs at time  $TO_x$ , imagine the complications in co-ordinating the tasks of every circuit
- A major school of thought (used in most processors built today): all circuits on the chip share a clock signal (a square wave) that tells every circuit when to accept inputs, how much time they have to execute the logic, and when they must produce outputs



# Clock Terminology

---

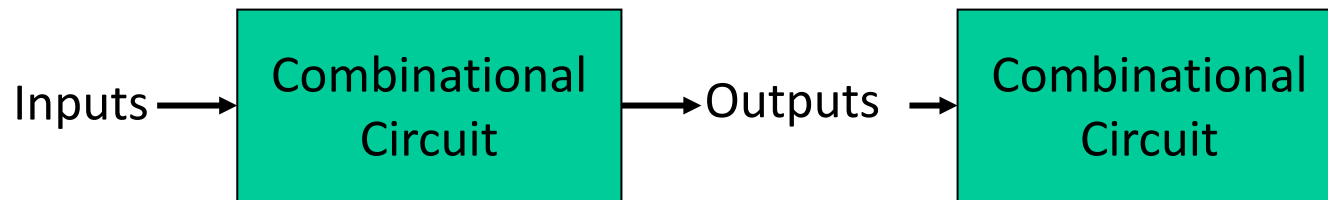


$$4 \text{ GHz} = \text{clock speed} = \frac{1}{\text{cycle time}} = \frac{1}{250 \text{ ps}}.$$

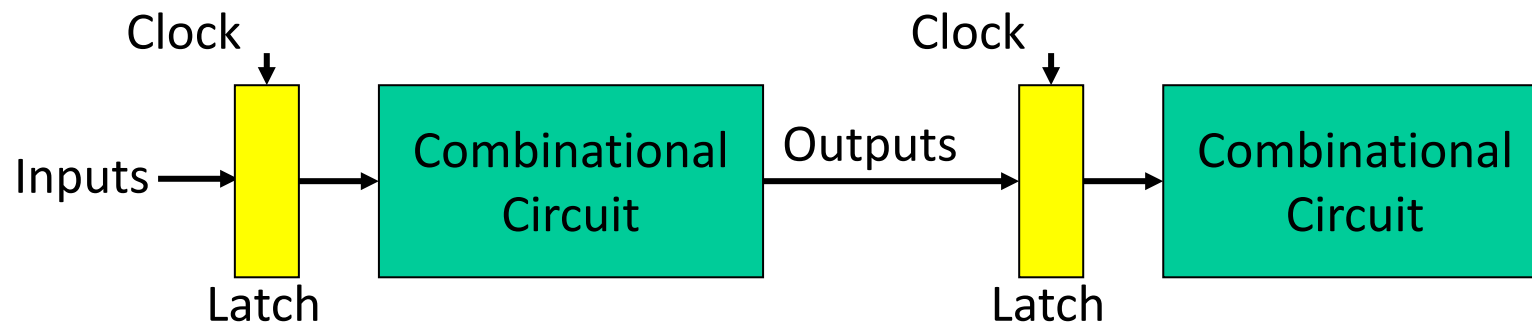
# Sequential Circuits

---

- Until now, circuits were combinational – when inputs change, the outputs change after a while (time = logic delay thru circuit)



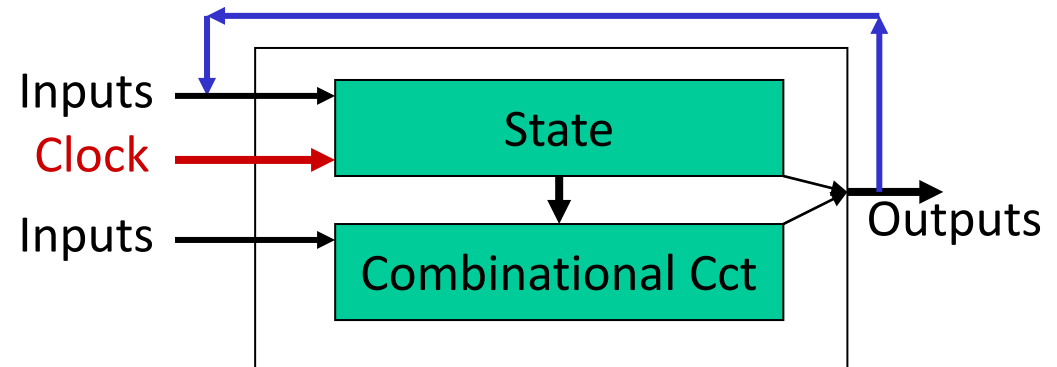
- We want the clock to act like a start and stop signal – a “latch” is a storage device that separates these circuits – it ensures that the inputs to the circuit do not change during a clock cycle



# Sequential Circuits

---

- Sequential circuit: consists of combinational circuit and a storage element
- At the start of the clock cycle, the rising edge causes the “state” storage to store some input values



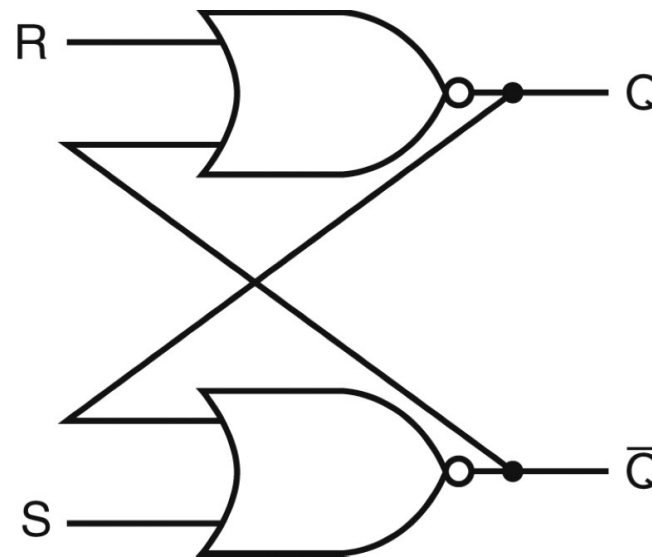
- This state will not change for an entire cycle (until next rising edge)
- The combinational circuit has some time to accept the value of “state” and “inputs” and produce “outputs”
- Some of the outputs (for example, the value of next “state”) may feed back (but through the latch so they’re only seen in the next cycle)

# Designing a Latch

---

- An S-R latch: set-reset latch
  - When Set is high, a 1 is stored
  - When Reset is high, a 0 is stored
  - When both are low, the previous state is preserved (hence, known as a storage or memory element)
  - Both are high – this set of inputs is not allowed

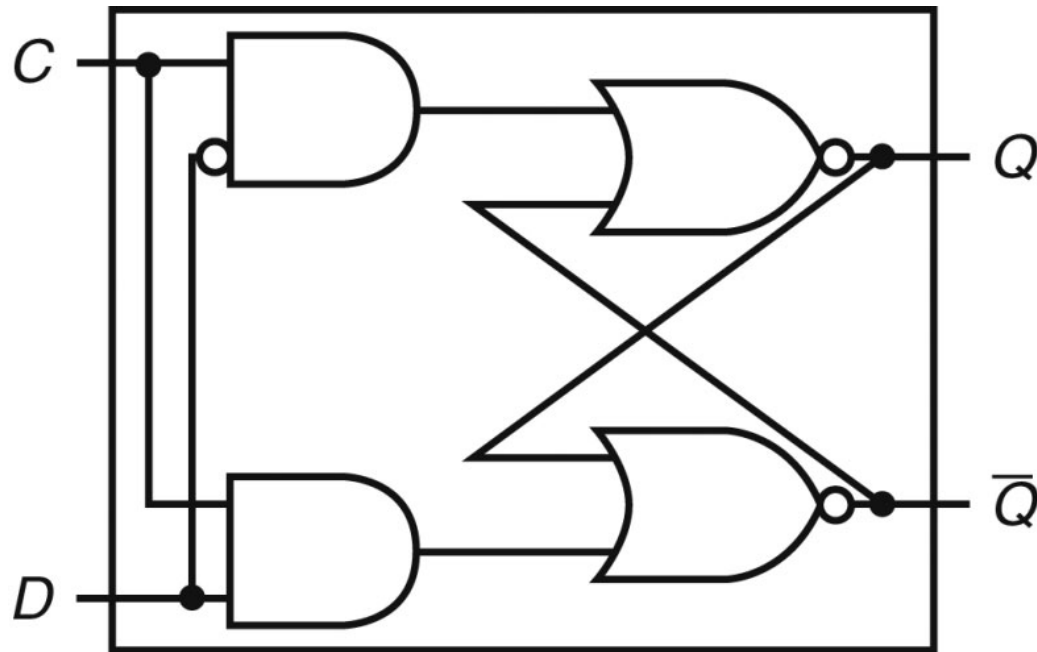
Verify the above behavior!



# D Latch

---

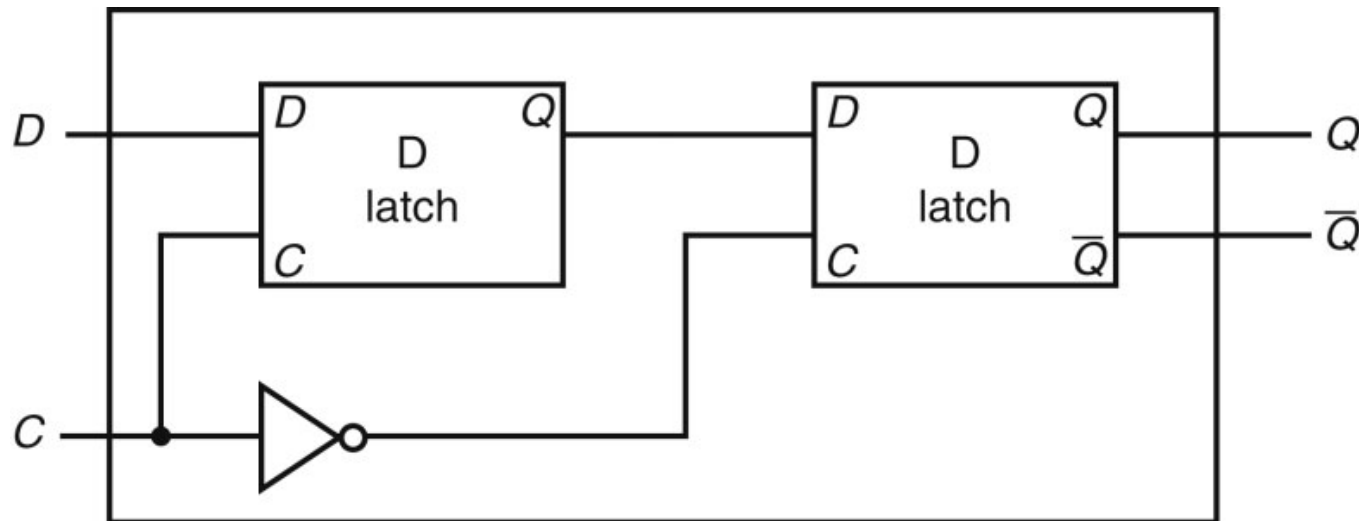
- Incorporates a clock
- The value of the input D signal (data) is stored only when the clock is high – the previous state is preserved when the clock is low



# D Flip Flop

---

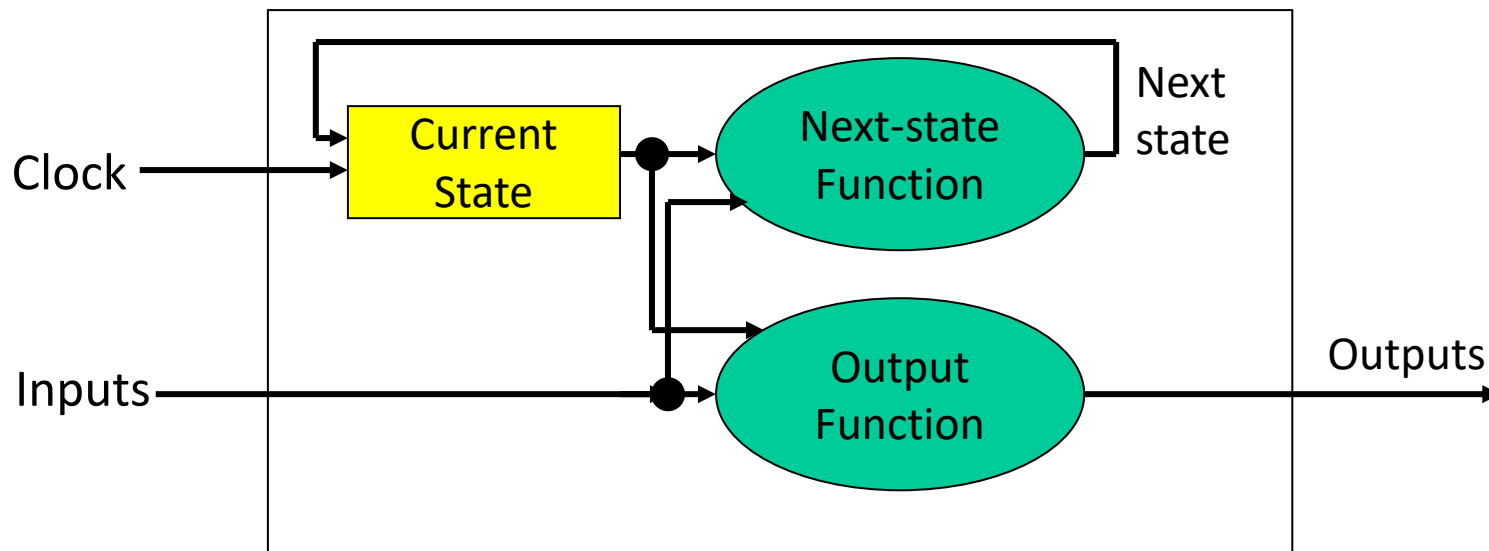
- Terminology:  
Latch: outputs can change any time the clock is high (asserted)  
Flip flop: outputs can change only on a clock edge
- Two D latches in series – ensures that a value is stored only on the falling edge of the clock



# Finite State Machine

---

- A sequential circuit is described by a variation of a truth table – a finite state diagram (hence, the circuit is also called a finite state machine)
- Note that state is updated only on a clock edge



# Basic MIPS Architecture

---

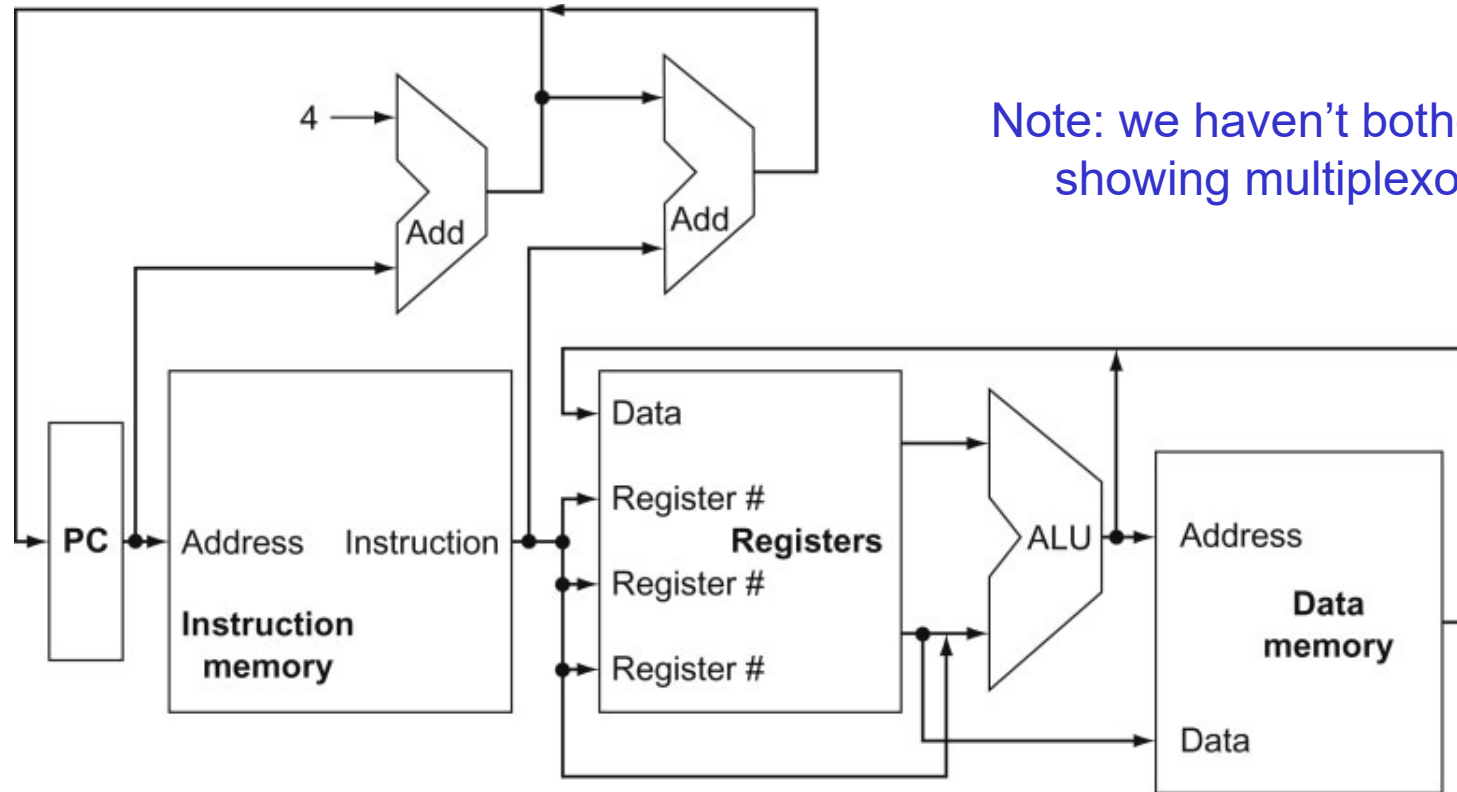
- Now that we understand clocks and storage of states, we'll design a simple CPU that executes:
  - basic math (add, sub, and, or, slt)
  - memory access (lw and sw)
  - branch and jump instructions (beq and j)

# 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

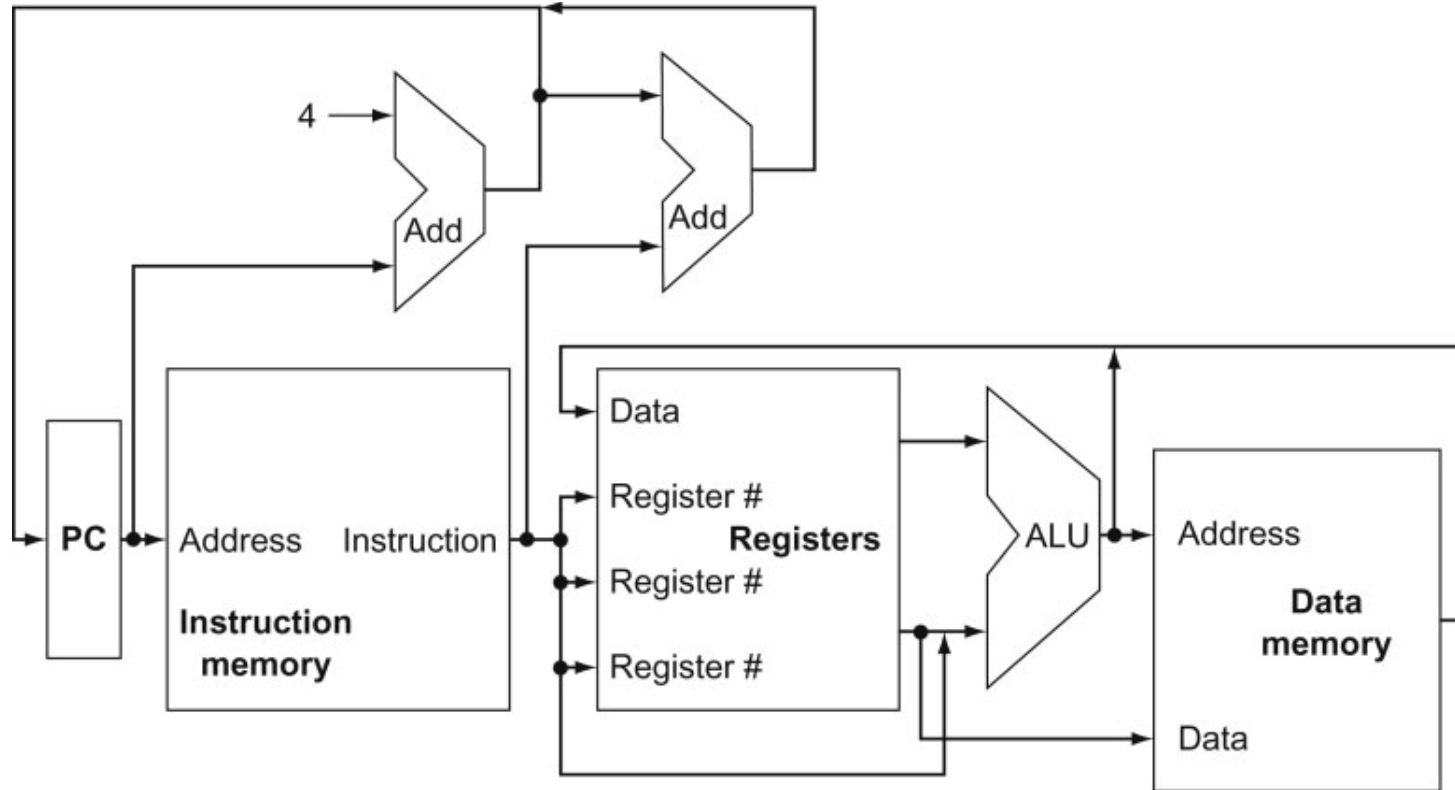


Note: we haven't bothered showing multiplexors

- 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

Source: H&P textbook

# Clocking Methodology



Source: H&P textbook

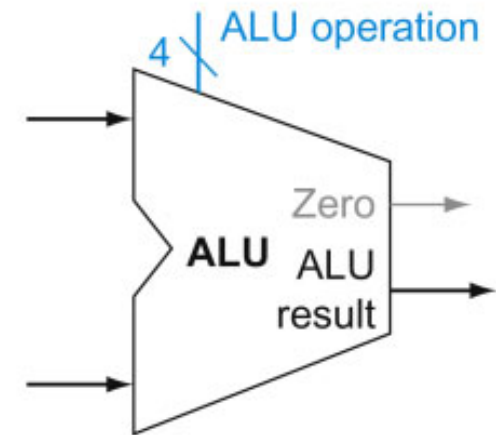
- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?  
Keep in mind that the latched value remains there for an entire cycle

# Implementing R-type Instructions

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



a. Registers

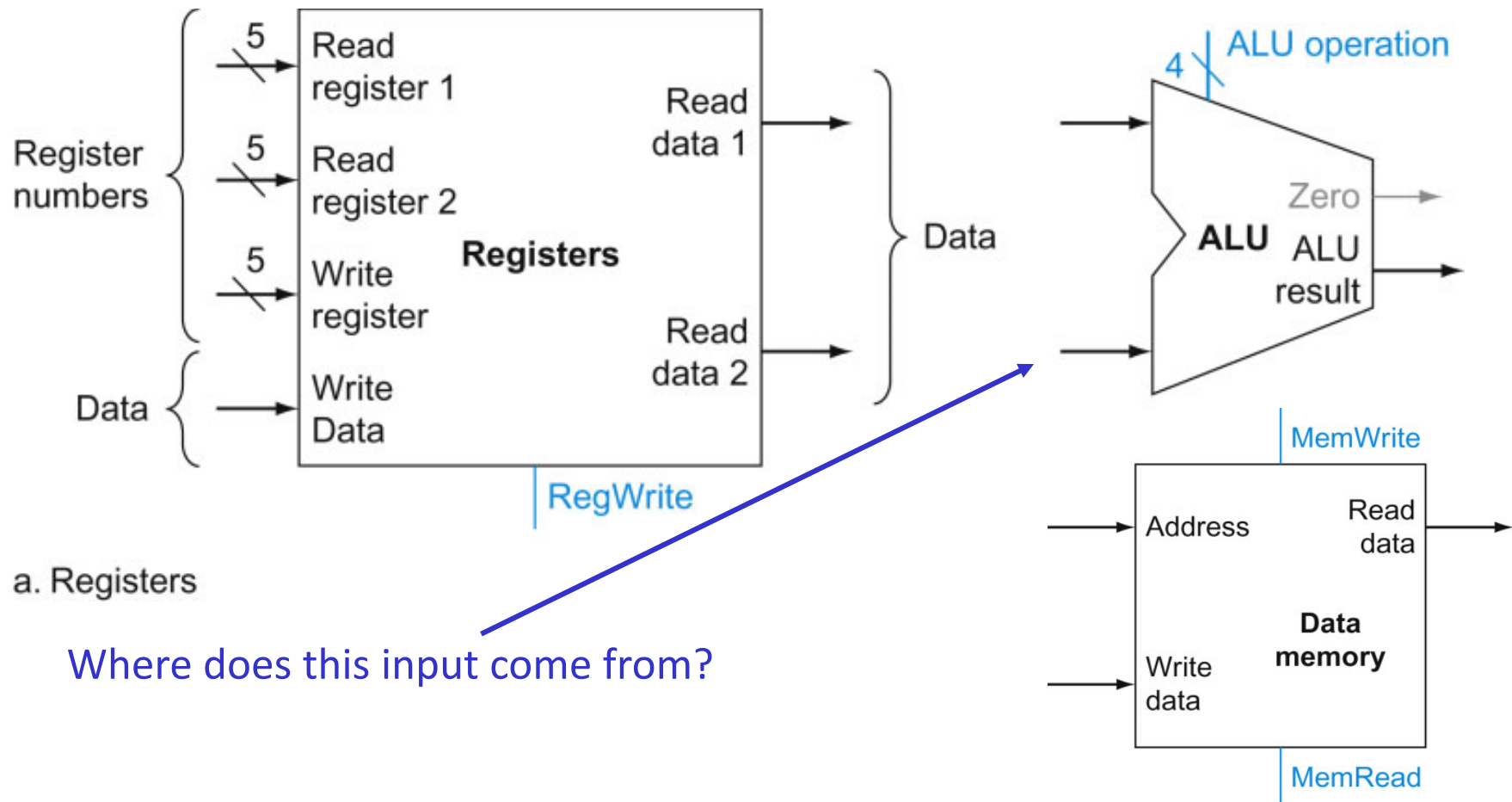


b. ALU

Source: H&P textbook

# Implementing Loads/Stores

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



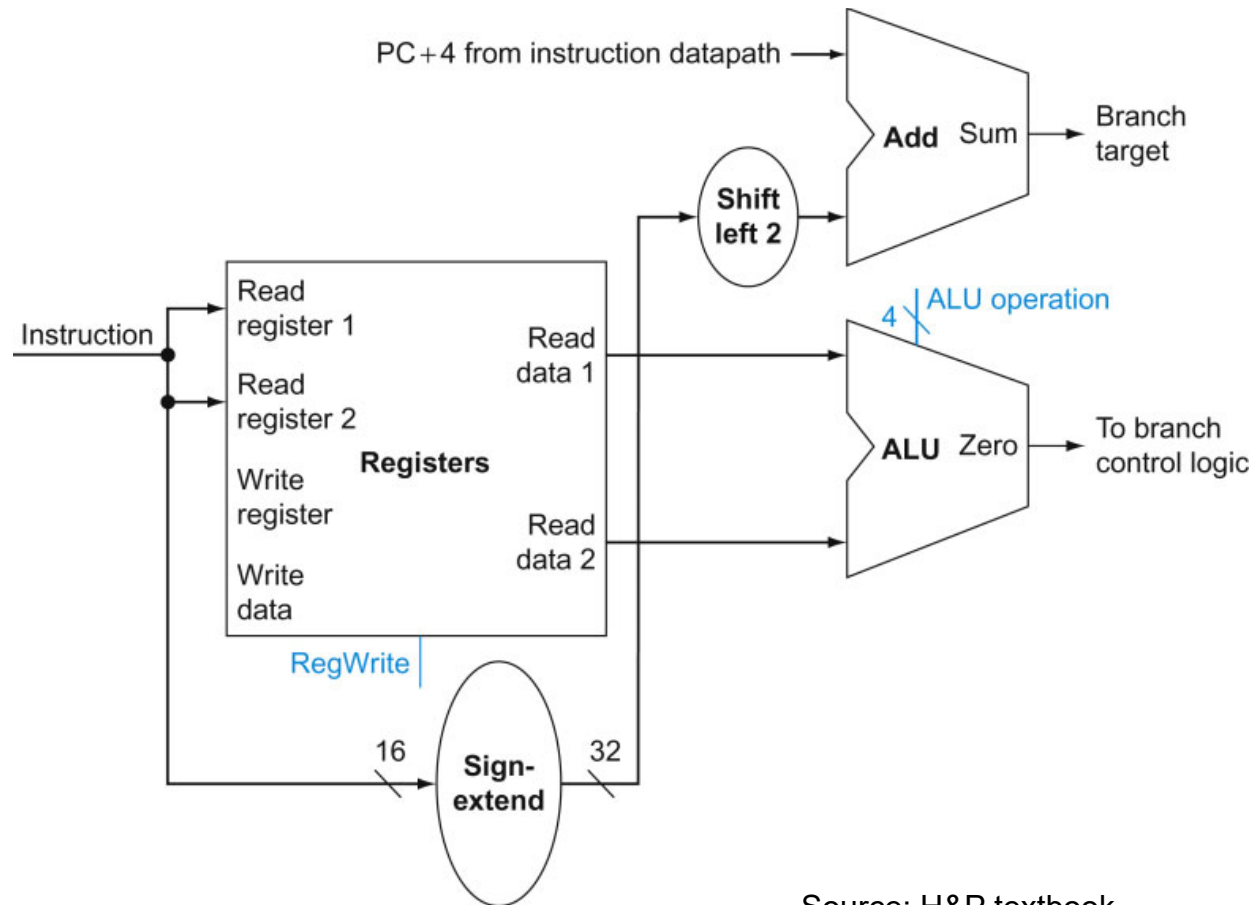
a. Registers

Where does this input come from?

a. Data memory unit Source: H&P textbook

# Implementing J-type Instructions

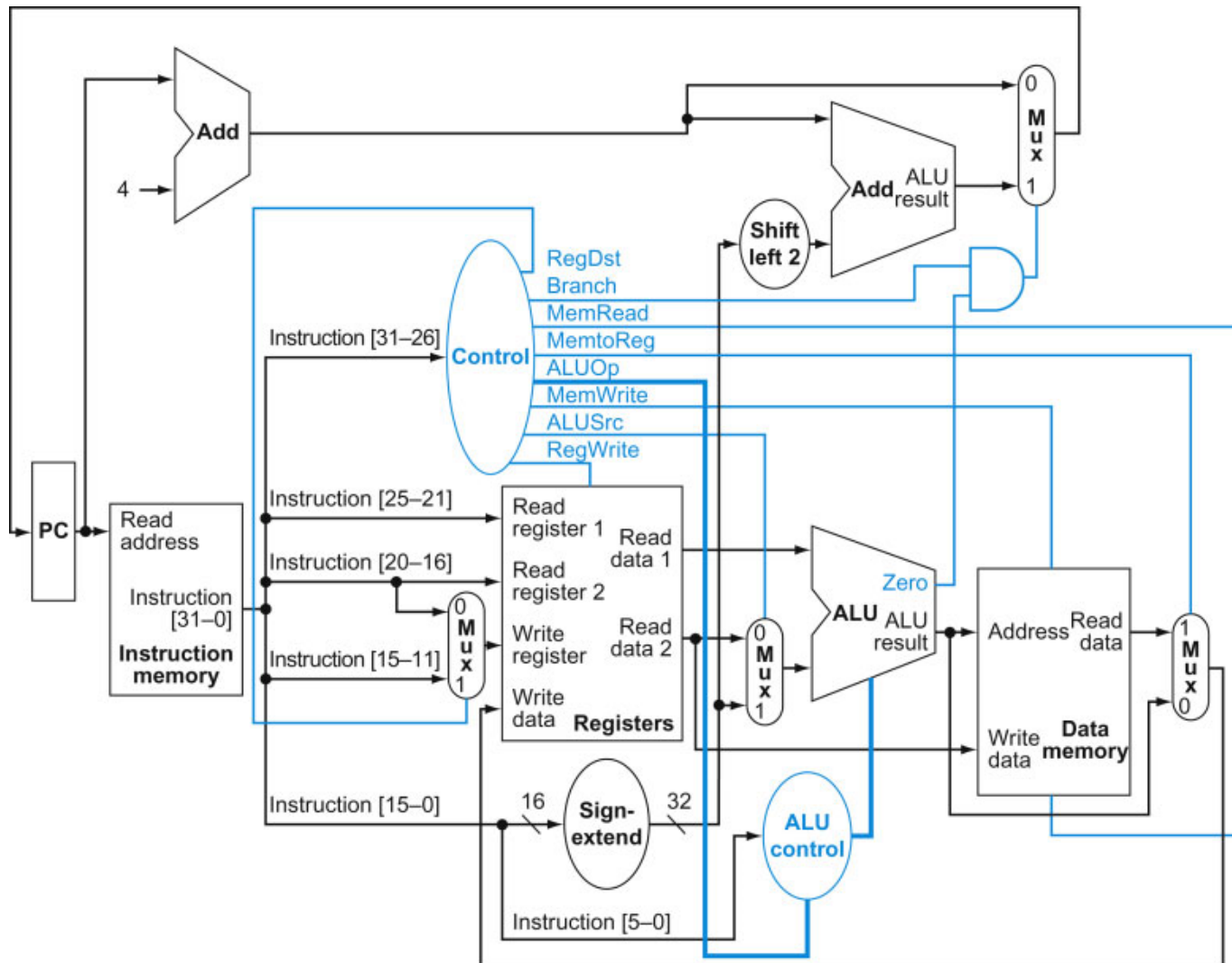
- Instructions of the form `beq $t1, $t2, offset`



Source: H&P textbook

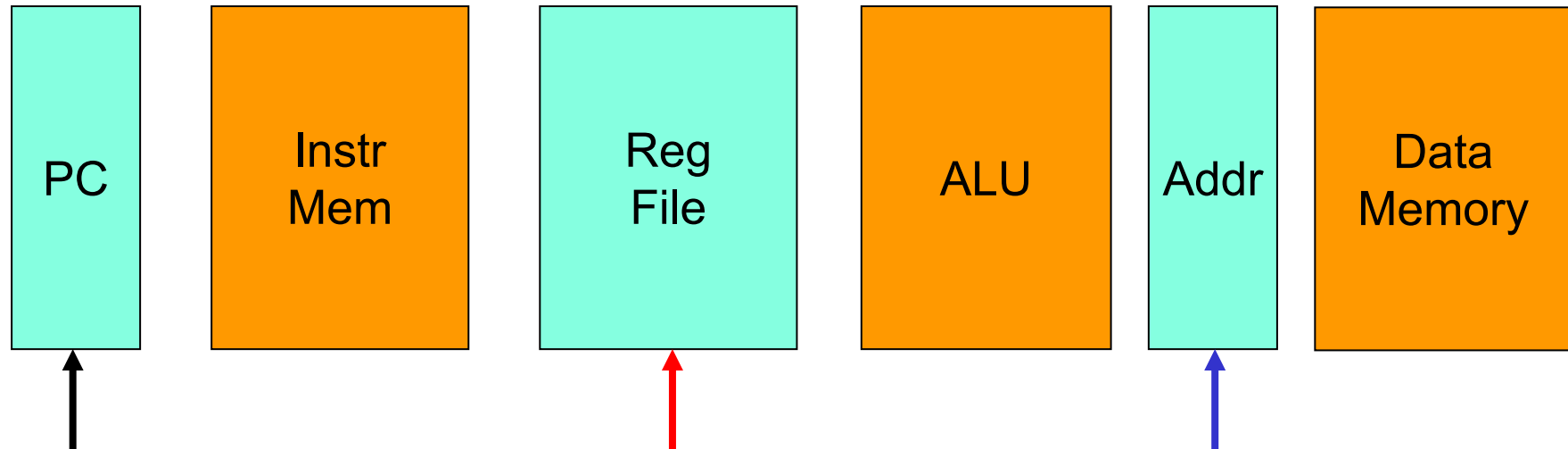






# View from 5,000 Feet



# Latches and Clocks in a Single-Cycle Design

---



- The entire instruction executes in a single cycle
- Green blocks are latches
- At the rising edge, a new PC is recorded 
- At the rising edge, the result of the previous cycle is recorded 
- At the falling edge, the address of LW/SW is recorded so  we can access the data memory in the 2<sup>nd</sup> half of the cycle 

# Multi-Stage Circuit

---

Instead of executing the entire instruction in a single cycle (a single stage), let's break up the execution into multiple stages, each separated by a latch

