

# Lecture 15: Basic CPU Design

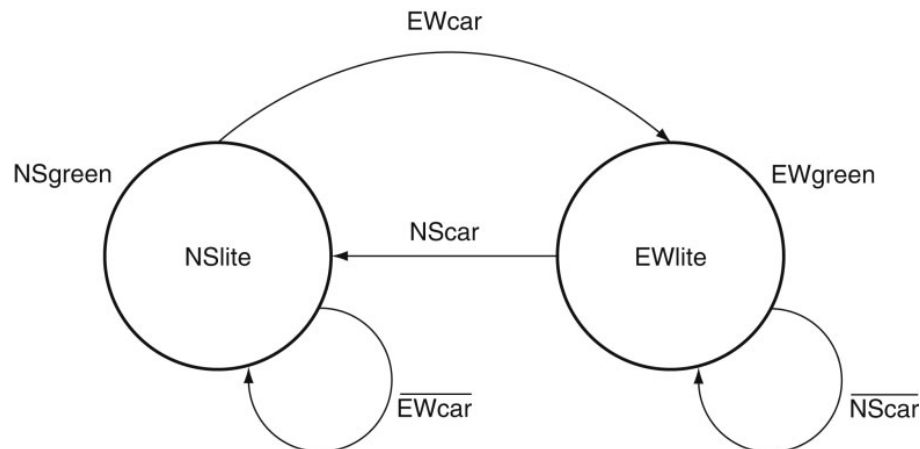
---

- Today's topics:
  - FSM examples
  - Single-cycle CPU
  - Multi-cycle CPU

# State Diagram

State Transition Table:

CurrState	InputEW	InputNS	NextState=Output
N	0	0	N
N	0	1	N
N	1	0	E
N	1	1	E
E	0	0	E
E	0	1	N
E	1	0	E
E	1	1	N



Source: H&P textbook

# Tackling FSM Problems

---

- Three questions worth asking:
  - What are the possible output states? Draw a bubble for each.
  - What are inputs? What values can those inputs take?
  - For each state, what do I do for each possible input value? Draw an arc out of every bubble for every input value.

# Example – Residential Thermostat

---

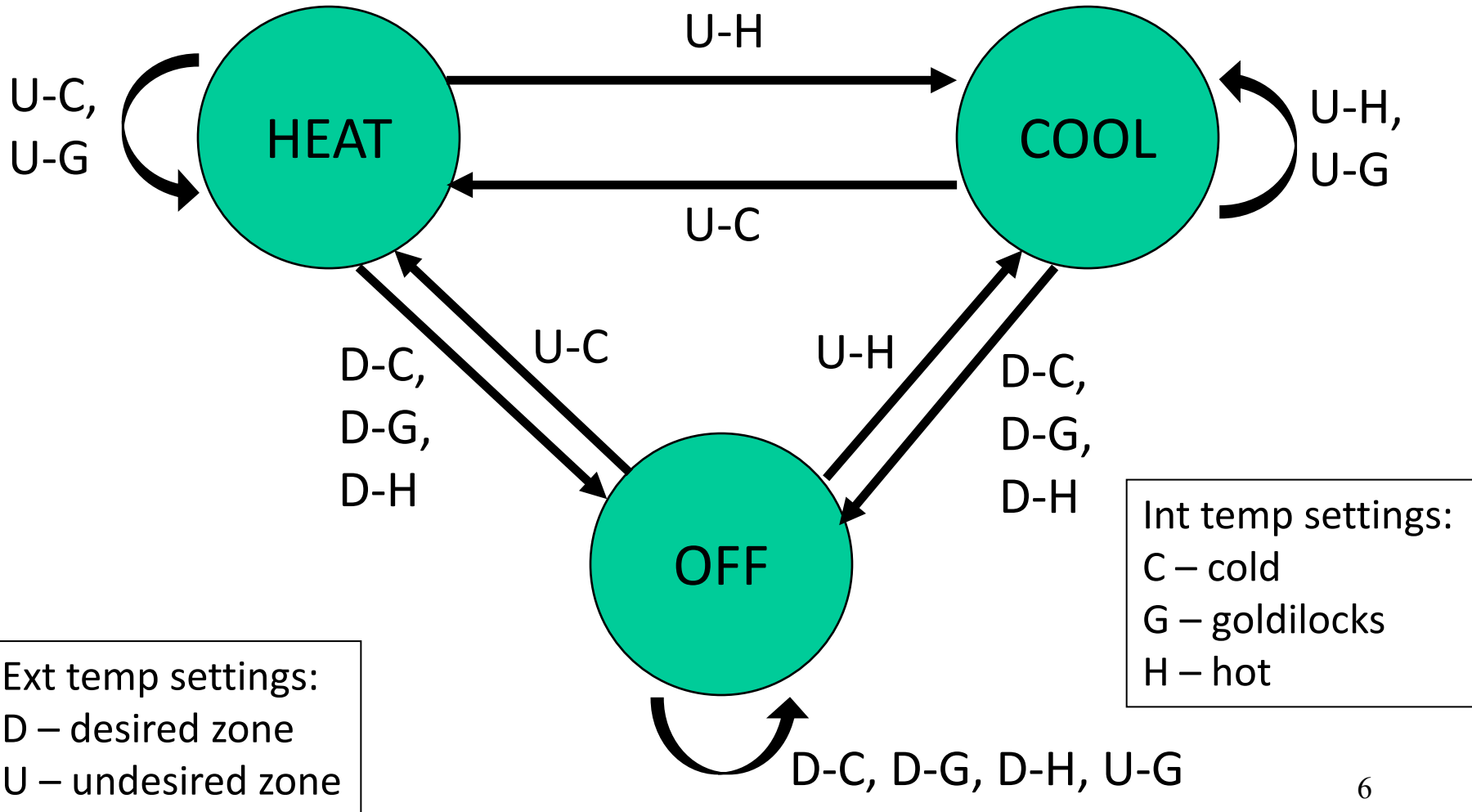
- Two temp sensors: internal and external
- If internal temp is within 1 degree of desired, don't change setting
- If internal temp is  $> 1$  degree higher than desired, turn AC on; if internal temp is  $< 1$  degree lower than desired, turn heater on
- If external temp and desired temp are within 5 degrees, disregard the internal temp, and turn both AC and heater off

# Finite State Machine Table

---

Current State	Input E	Input I	Output State
HEAT	D	C	OFF
HEAT	D	G	OFF
HEAT	D	H	OFF
HEAT	U	C	HEAT
HEAT	U	G	HEAT
HEAT	U	H	COOL
COOL	D	C	OFF
COOL	D	G	OFF
COOL	D	H	OFF
COOL	U	C	HEAT
COOL	U	G	COOL
COOL	U	H	COOL
OFF	D	C	OFF
OFF	D	G	OFF
OFF	D	H	OFF
OFF	U	C	HEAT
OFF	U	G	OFF
OFF	U	H	COOL

# Finite State Diagram



# Latch vs. Flip-Flop

---

- Recall that we want a circuit to have stable inputs for an entire cycle – so I want my new inputs to arrive at the start of a cycle and be fixed for an entire cycle
- A flip-flop provides the above semantics (a door that swings open and shut at the start of a cycle)
- But a flip-flop needs two back-to-back D-latches, i.e., more transistors, delay, power
- You can reduce these overheads with just a single D-latch (a door that is open for half a cycle) as long as you can tolerate stable inputs for just half a cycle

# 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)

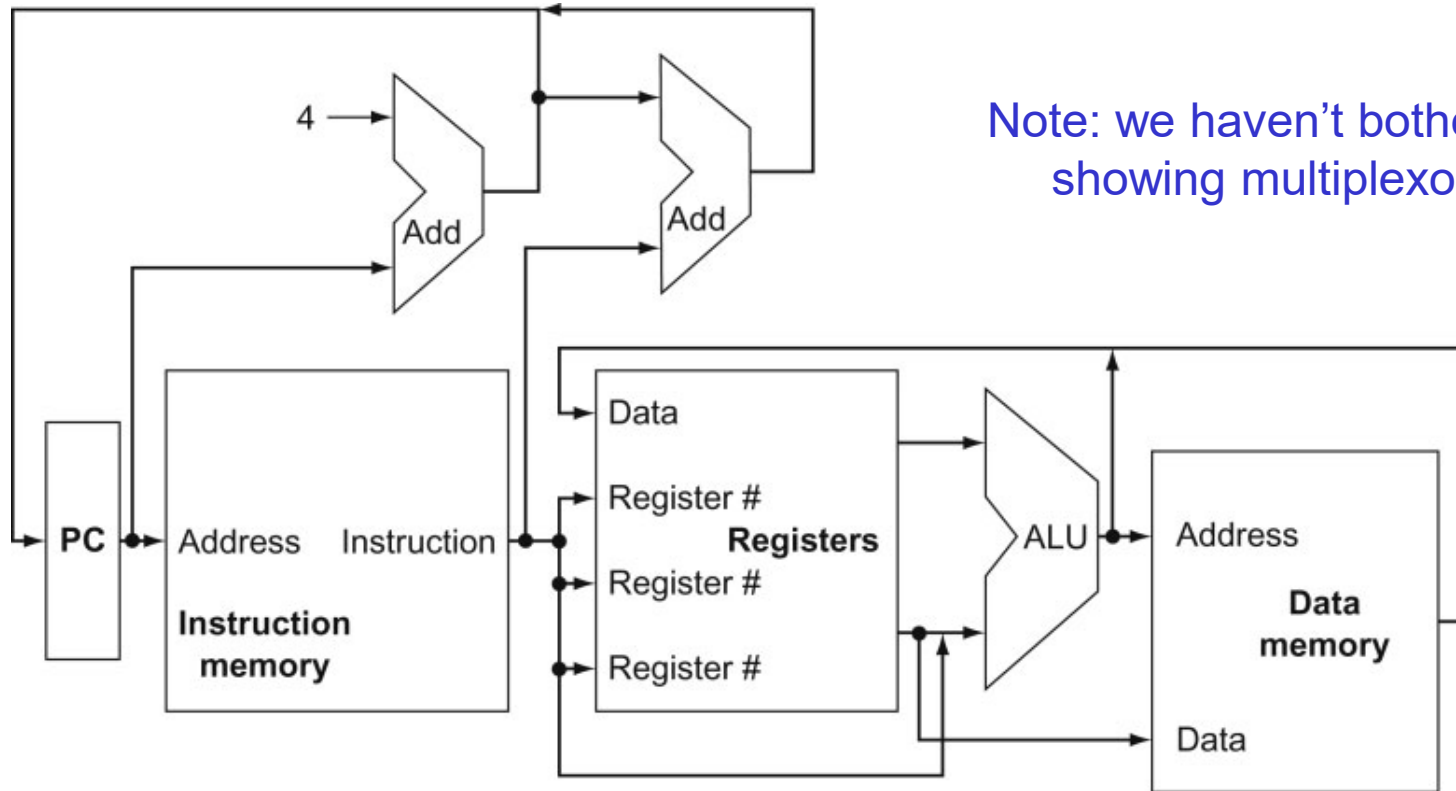


# Implementation Overview

---

- 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

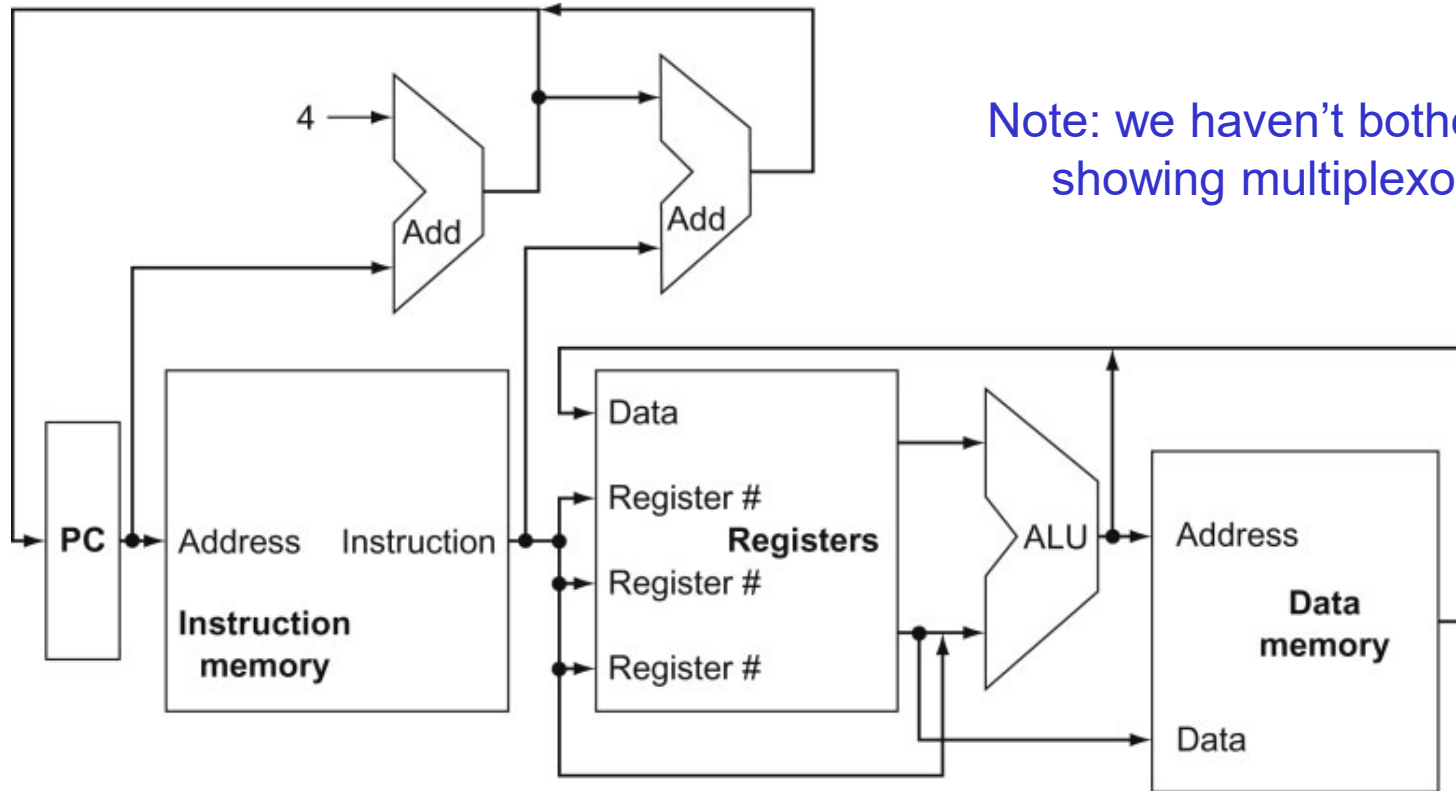


Note: we haven't bothered showing multiplexors

- 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

# View from 30,000 Feet

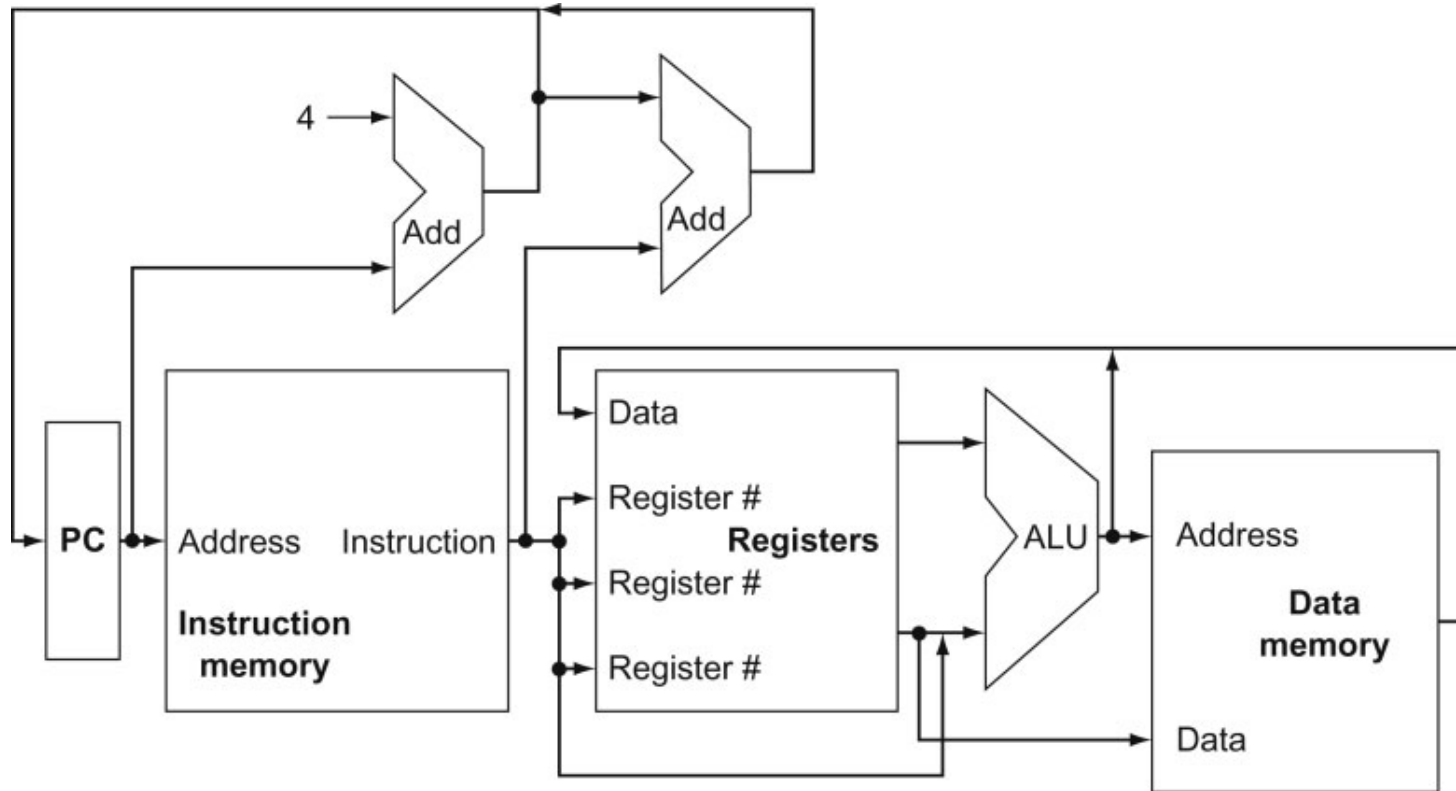


Note: we haven't bothered showing multiplexors

- 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

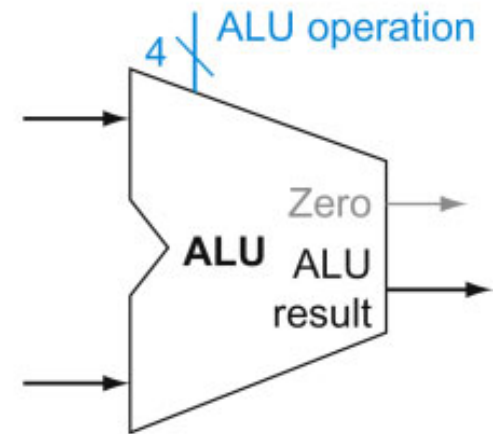
- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?  
Keep in mind that the latched value remains there for an entire cycle

# Implementing R-type Instructions

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



a. Registers

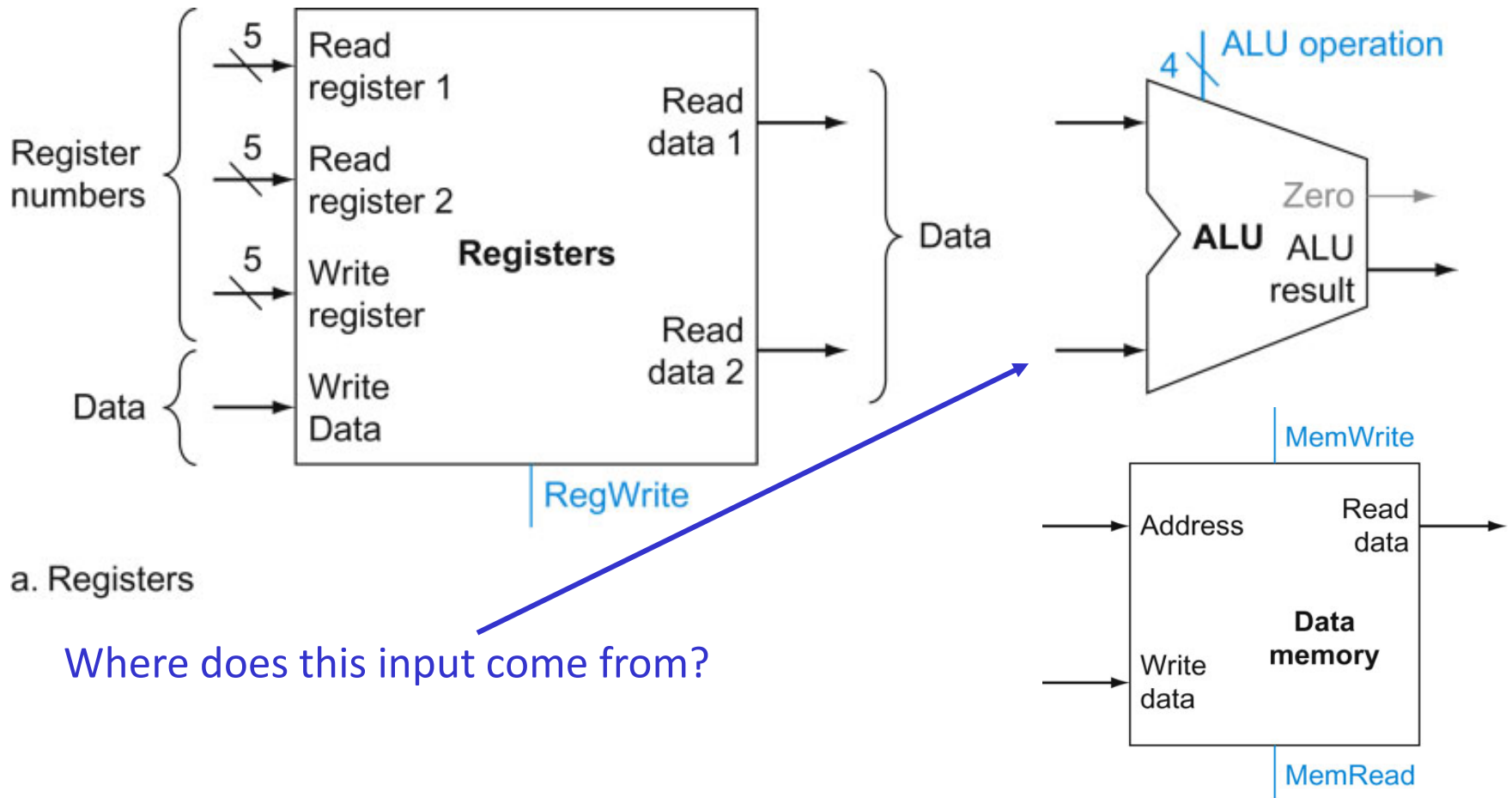


b. ALU

Source: H&P textbook

# Implementing Loads/Stores

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



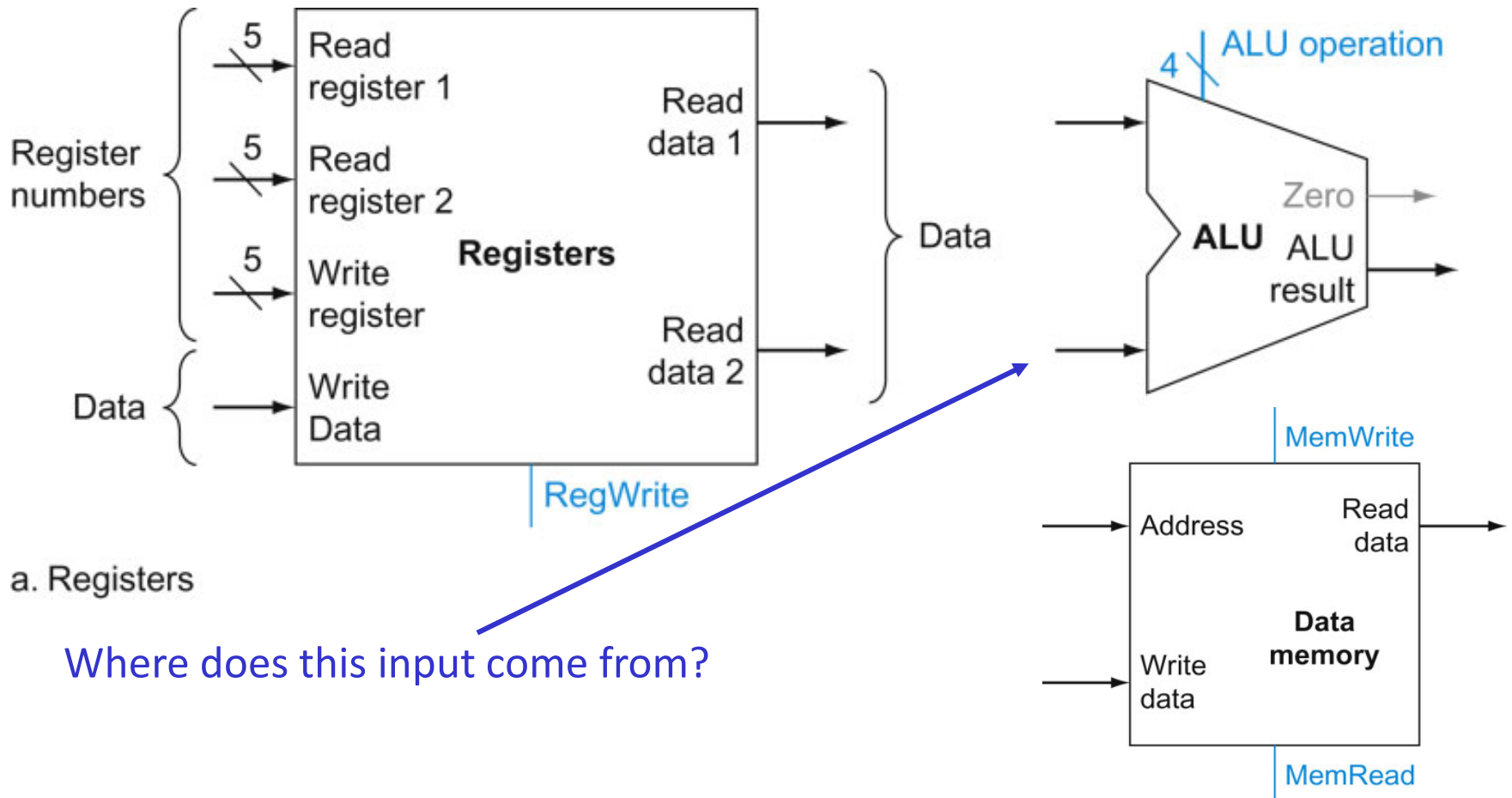
a. Registers

Where does this input come from?

a. Data memory unit Source: H&P textbook

# Implementing Loads/Stores

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



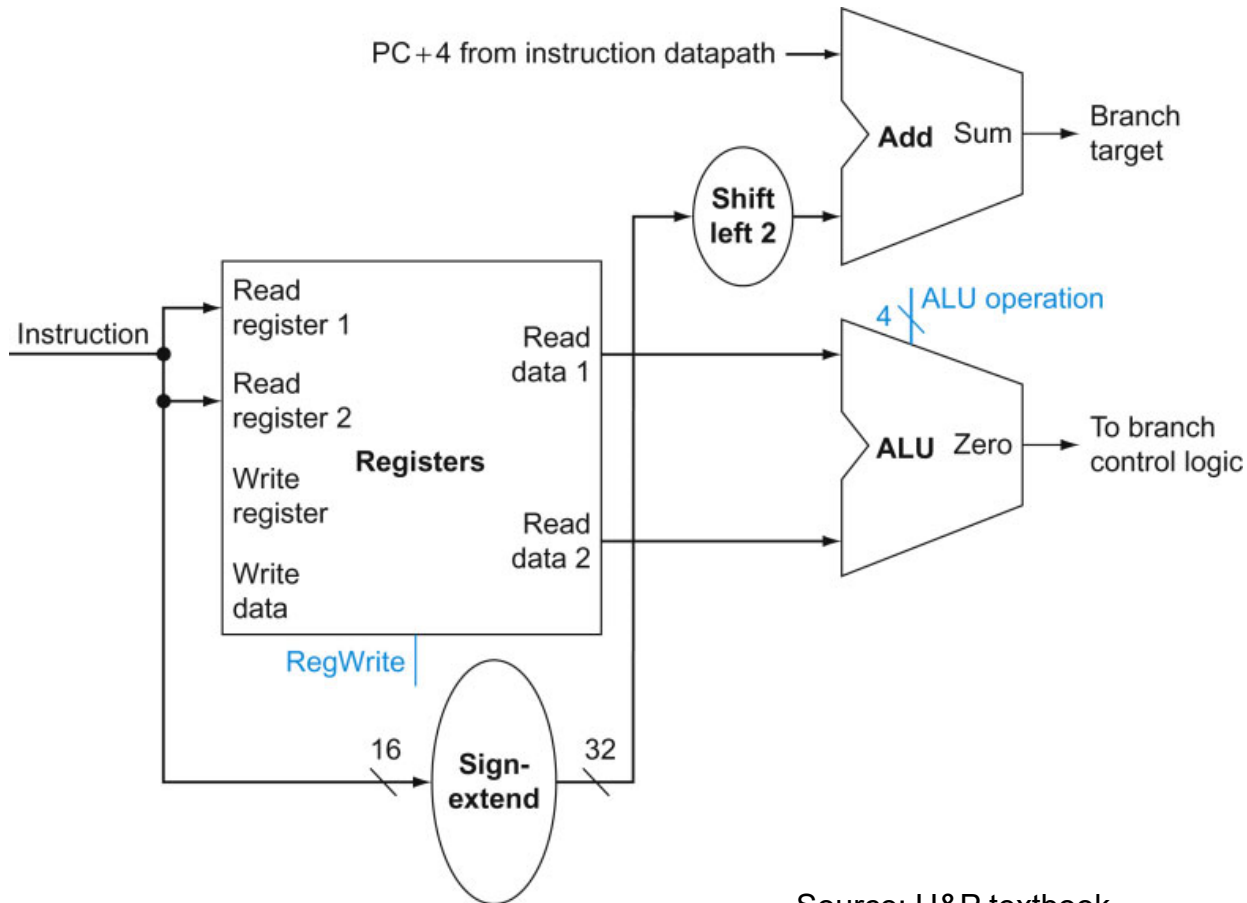
a. Registers

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`

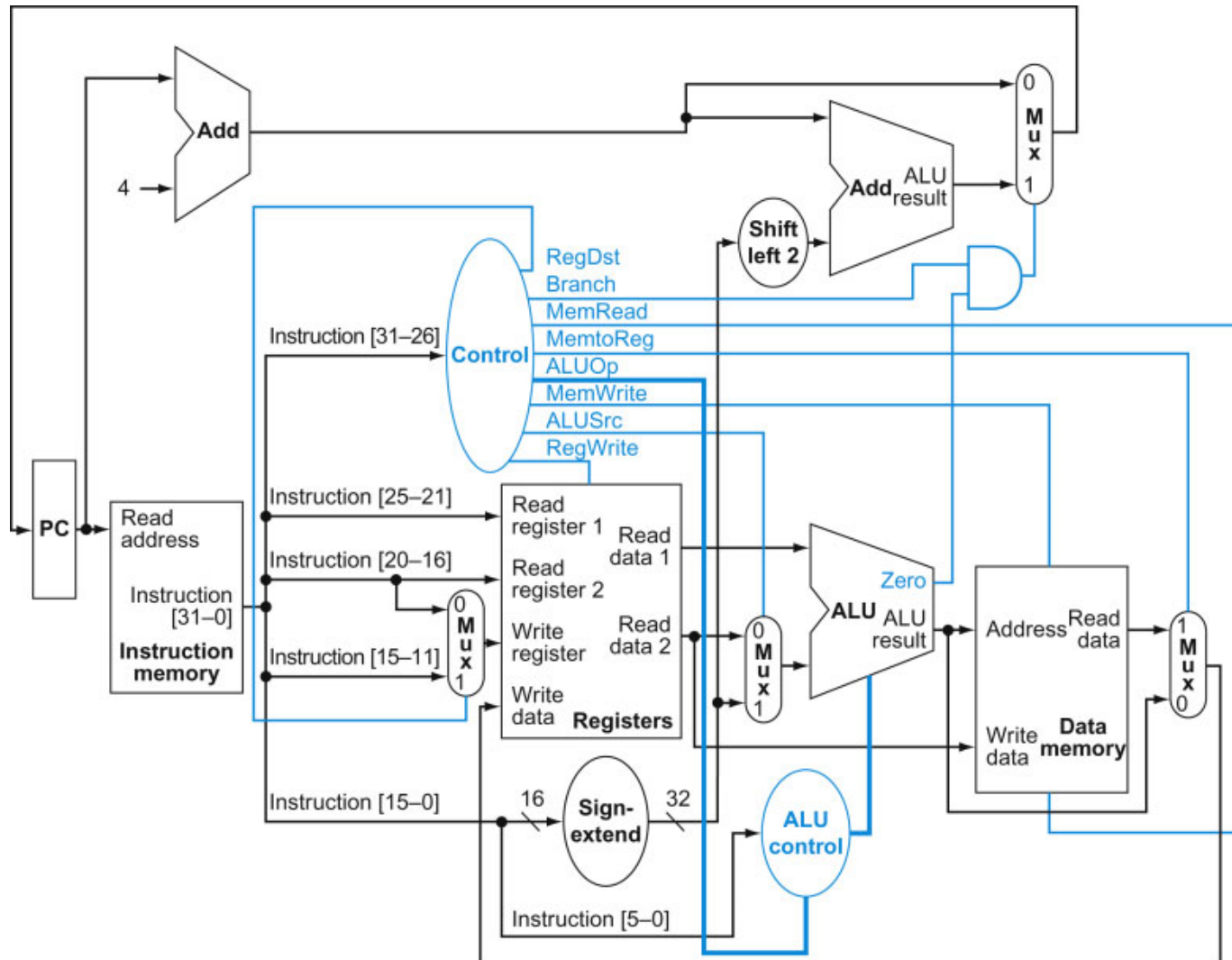


Source: H&P textbook



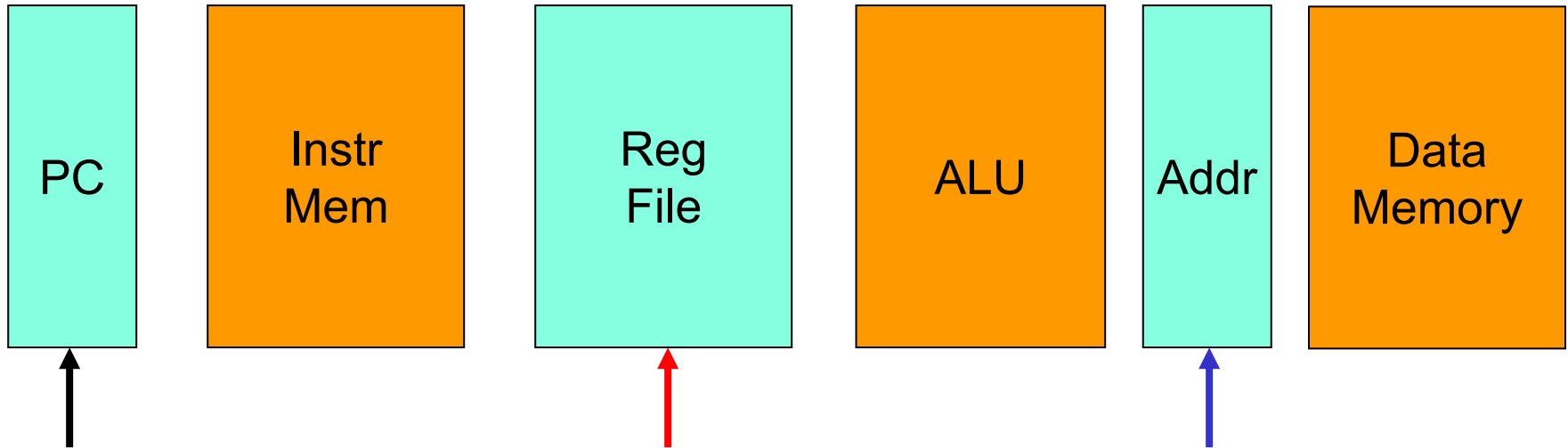





# View from 5,000 Feet



# 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