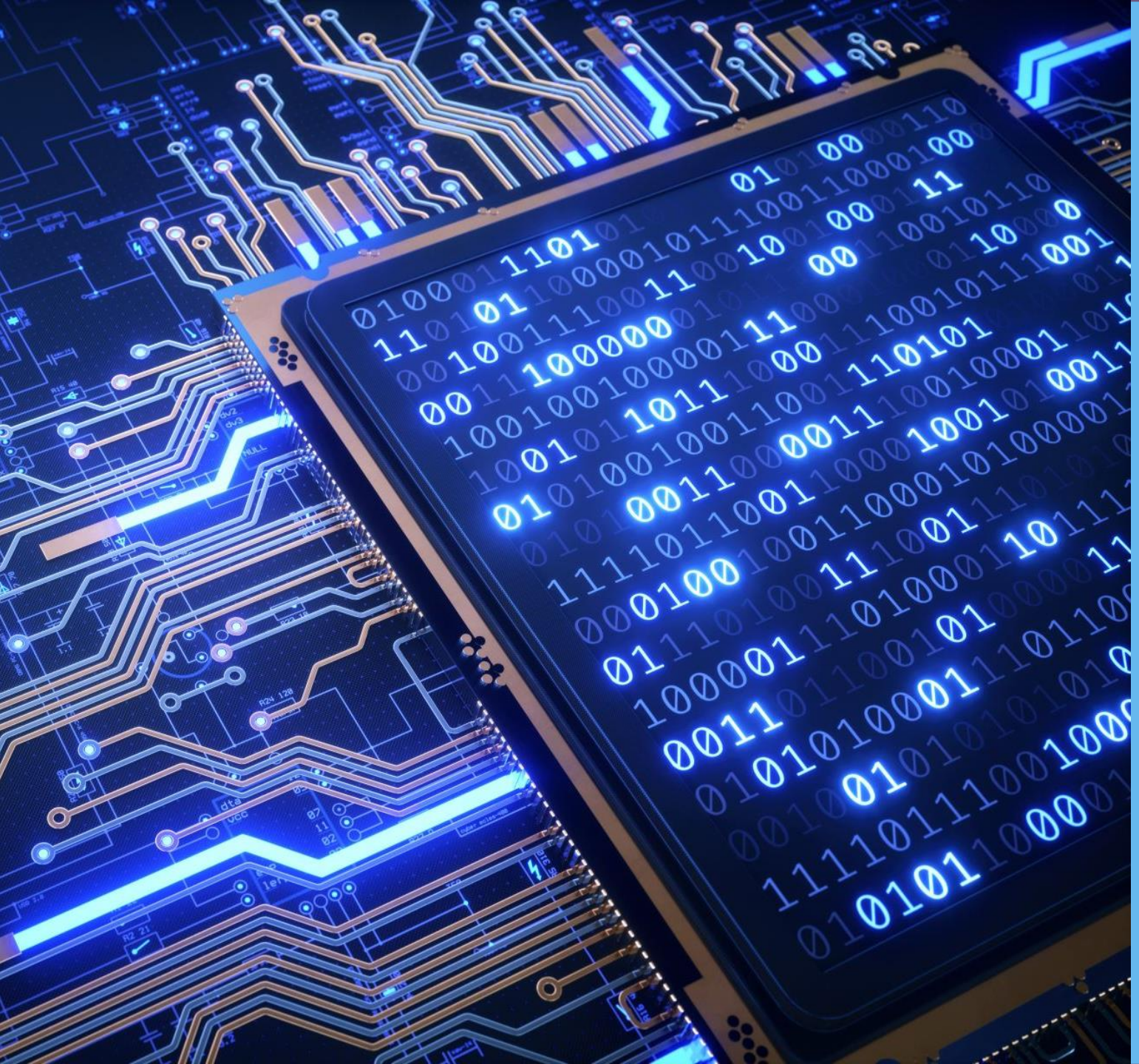


Computer Systems

# Central Processing Unit

CS51 – Fall 2025

# The Central Processing Unit (CPU)

- The CPU is the “brain” of the computer. Its job is to run programs, like Word, Chrome, etc.
- Programs are made from **instructions** that tell the computer what to do.
- Some examples of instructions might be to perform mathematical and logical operations (e.g., add) or read/write in memory.
- At this level, we will stop caring about the physical components that the CPU is built out of and focus more broadly on the computer organization or **microarchitecture**, focusing on how instructions are executed, data are handled, and operations are performed by the CPU.
- We know that data, like numbers, characters, etc are stored in memory. Programs are stored in memory, too. We will specifically assume that the instructions are stored in a block of contiguous words in memory.

# Words and processors

- Processors handle data in chunks of bits called **words.**
  - The size of the word depends on the architecture of the processor. Today, for most computers, the size of a word is 32 or 64-bits/4 or 8 bytes.
- Next, we will see a simplified model of how the CPU reads a program from the memory and reads/writes in memory.

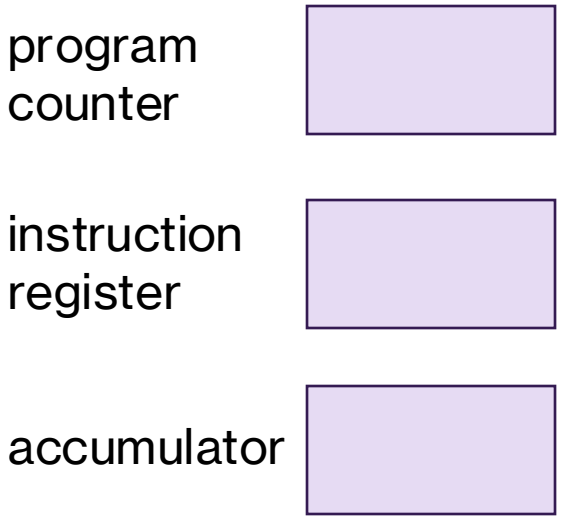
# Memory

- Let's assume our (tiny!) RAM consists of 8 words, each having 8 bits.
- To simplify things later, we will assume that addresses are 4-bits.
- Our RAM will hold both the instructions of a small program and the results of the program as it's executed.

| Address | Data |
|---|---|
| 0000 | 0001 0110 |
| 0001 | 1011 0111 |
| 0010 | 0100 0110 |
| 0011 | 1101 0001 |
| 0100 | 0000 0000 |
| 0101 | 0000 0000 |
| 0110 | 0000 0001 |
| 0111 | 0000 0001 |

# Registers

- Let's also assume that we have 1 register, the **accumulator**, which can hold one word, i.e. 8 bits.
- And two special registers:
  - the **program counter**, which has the address of the next instruction, and
  - the **instruction register**, which stores the instruction currently being executed.
- The program counter and instruction register are part of the control unit of the CPU while the accumulator is part of the arithmetic logic unit.



# Instruction table

- Rather than working with binary numbers for the instructions, we will assume that an instruction can be specified by a 4-bit opcode (operation code) followed by a 4-bit address (addr). For example, for our program, the four instructions we are interested in are:

| opcode | Instruction | Description |
|---|---|---|
| 0001 | LOAD addr | Load into accumulator the contents of the RAM at addr |
| 1011 | ADD addr | Add to accumulator the value stored in RAM at addr |
| 0100 | STORE addr | Store the contents of accumulator into RAM at address addr |
| 1101 | JUMP addr | Update program counter with the specified addr |

- Such a low-level programming language is known as **assembly**. An **assembler** is a special program that reads text-based instructions like the ones above and translates them into the binary **machine code** (i.e. binary) that is truly the only language that computers understand.

# Our (more) readable program

| opcode | Instruction |
|---|---|
| 0001 | LOAD addr |
| 1011 | ADD addr |
| 0100 | STORE addr |
| 1101 | JUMP addr |

- Using the instruction table we just saw, the RAM contents are more human-readable.
- For simplicity, let's also use decimal numbers instead of binary addresses

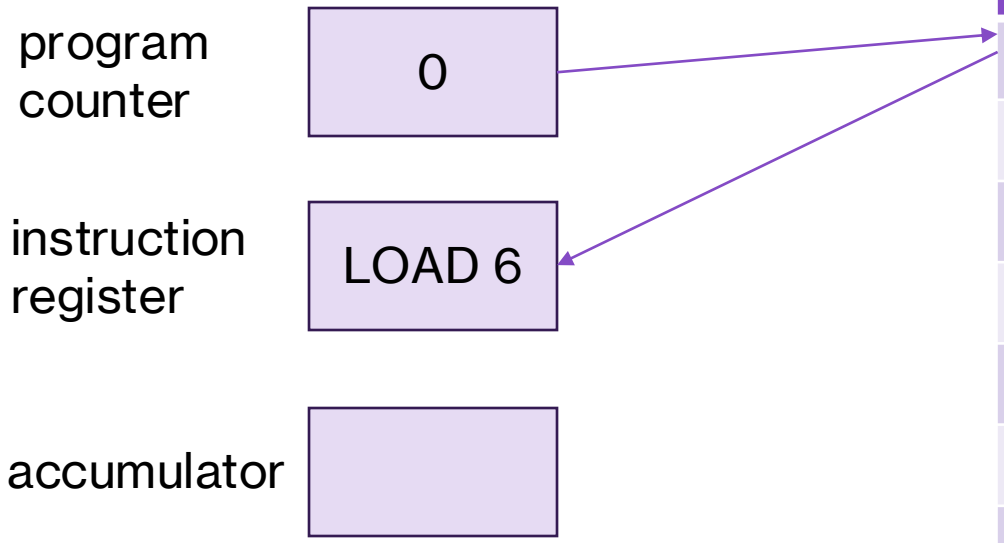
| Address | Data |
|---|---|
| 0000 | 0001 0110 |
| 0001 | 1011 0111 |
| 0010 | 0100 0110 |
| 0011 | 1101 0001 |
| 0100 | 0000 0000 |
| 0101 | 0000 0000 |
| 0110 | 0000 0001 |
| 0111 | 0000 0001 |

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Clock and fetch-decode-execute cycle

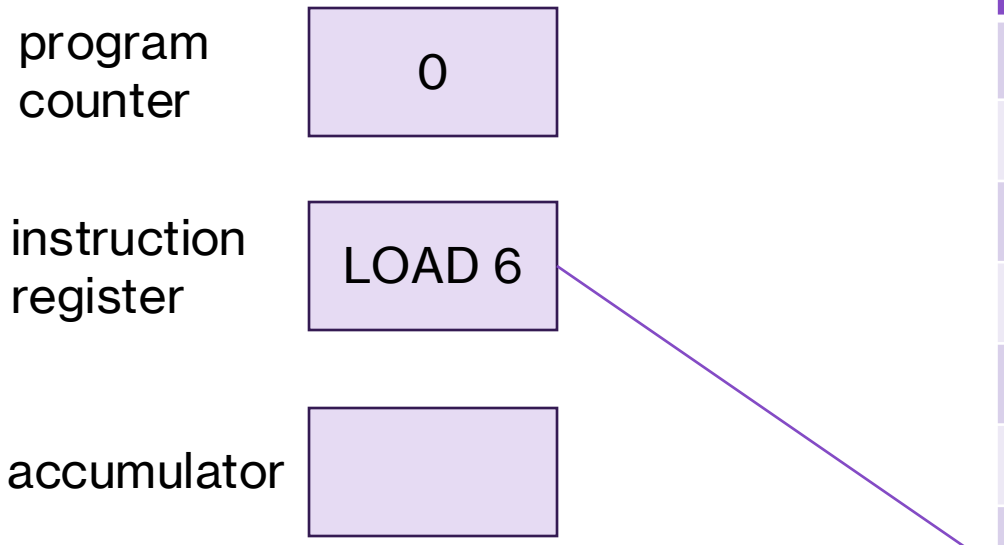
- Our CPU has a clock that acts as the “orchestra conductor.” All components are synchronized with the clock.
- With every tick of the clock, the CPU will go through a step of the **fetch-decode-execute cycle** or **instruction cycle** to execute the instructions of a program.
- The instruction cycle repeats continuously the following steps:
  - **Fetch**:
    - The CPU uses the program counter to locate the next instruction from RAM and put it into the instruction register.
  - **Decode**:
    - The CPU reads the fetched instruction from the instruction register and determines what operation needs to be performed.
  - **Execute**:
    - The CPU performs the operation specified by the instruction, which might involve accessing data, performing calculations, or controlling other parts of the computer.

# Executing our program - Fetch



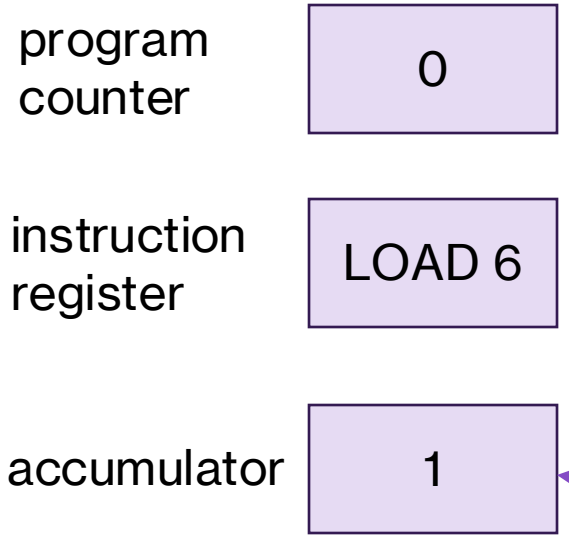

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Decode



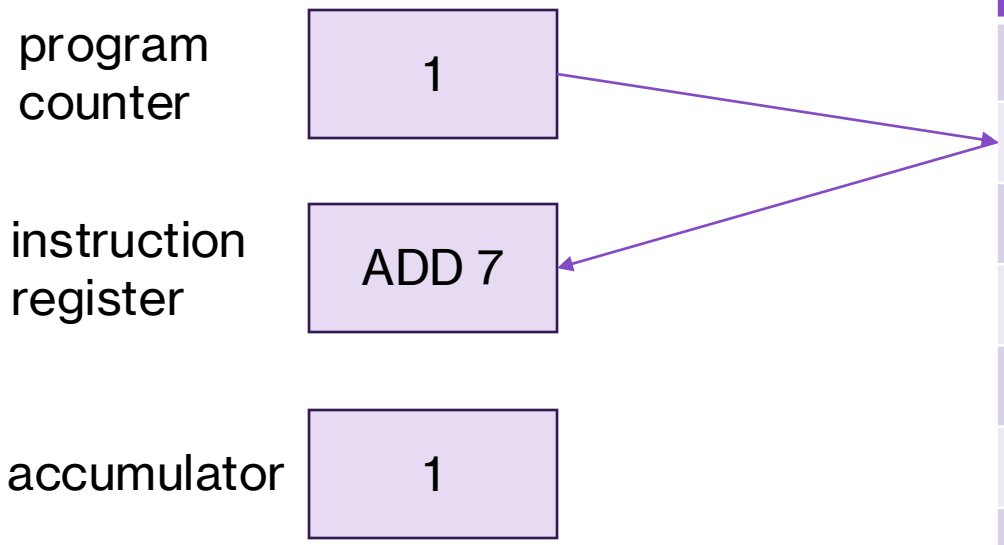

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Execute



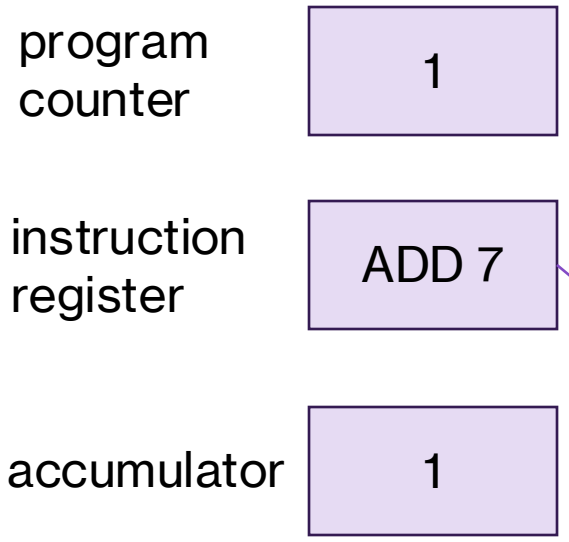

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Fetch



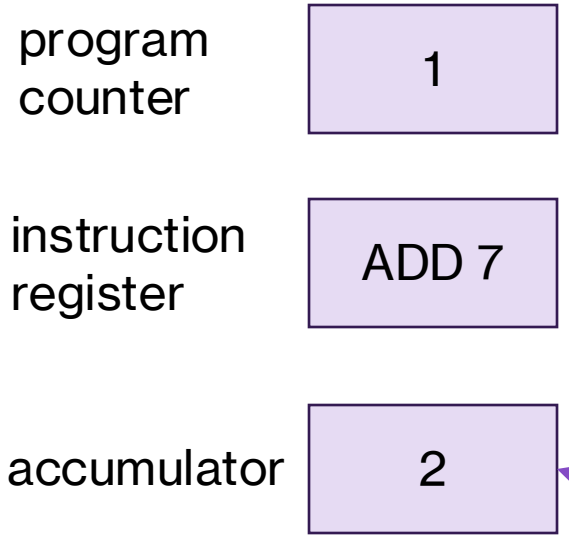

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Decode



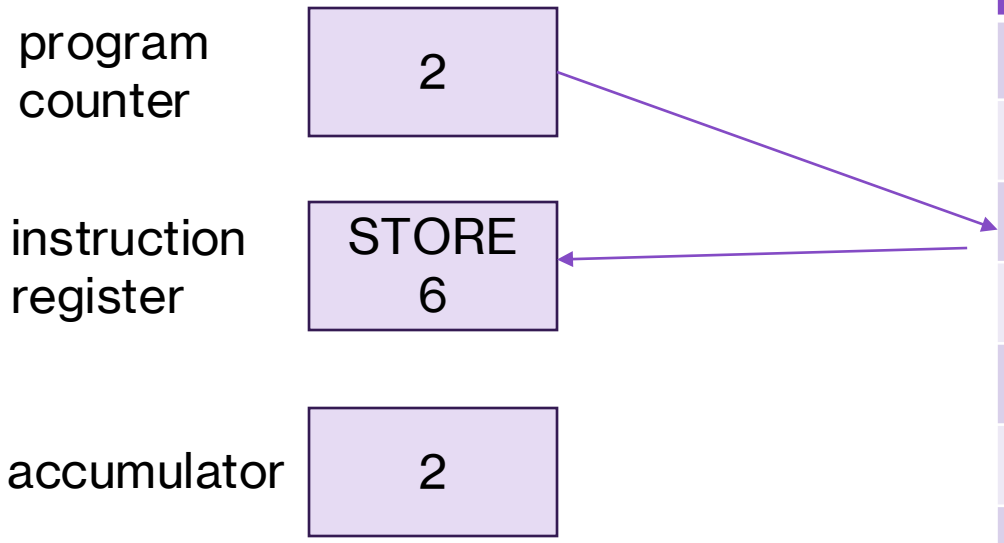

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Execute



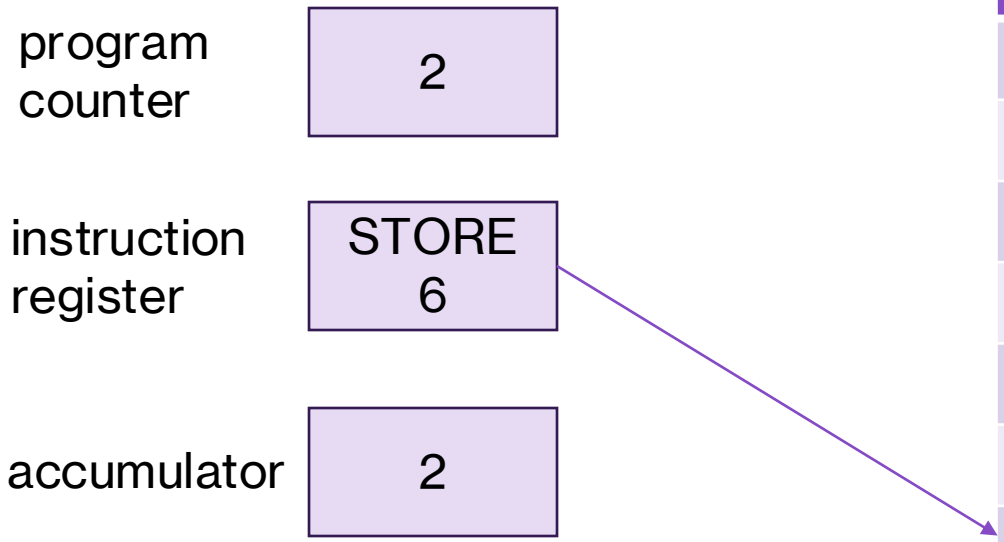

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Fetch



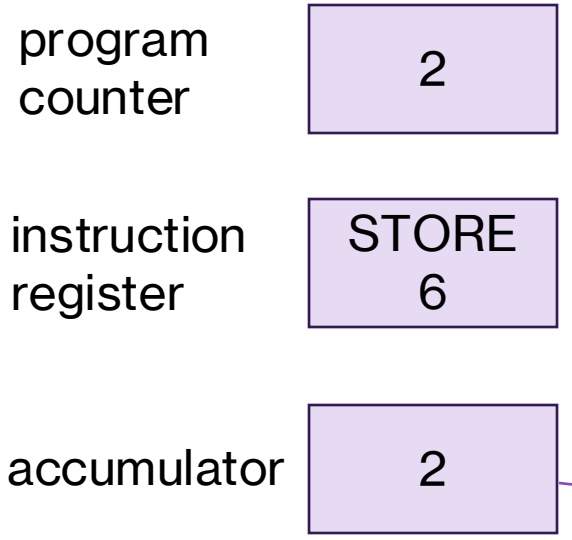

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Decode



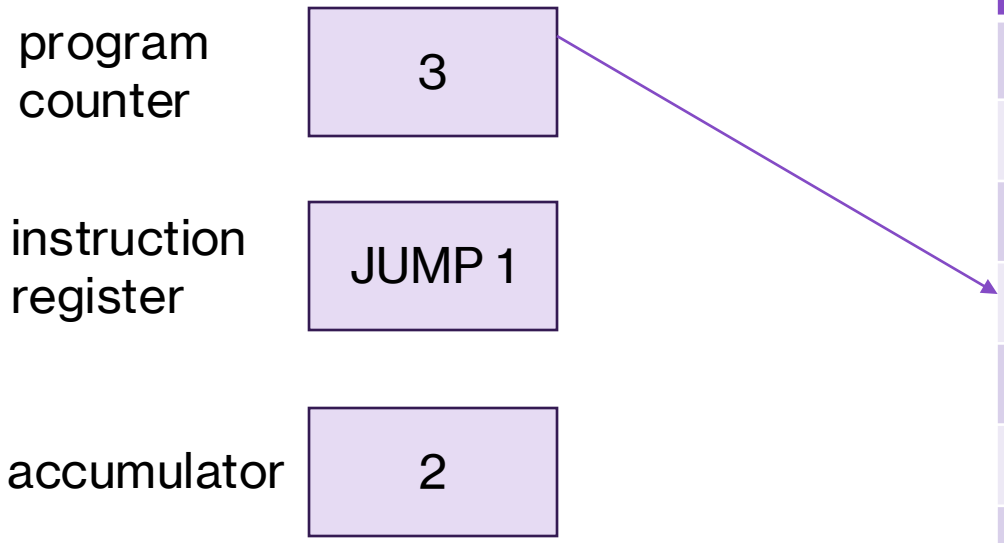

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |

# Executing our program - Execute



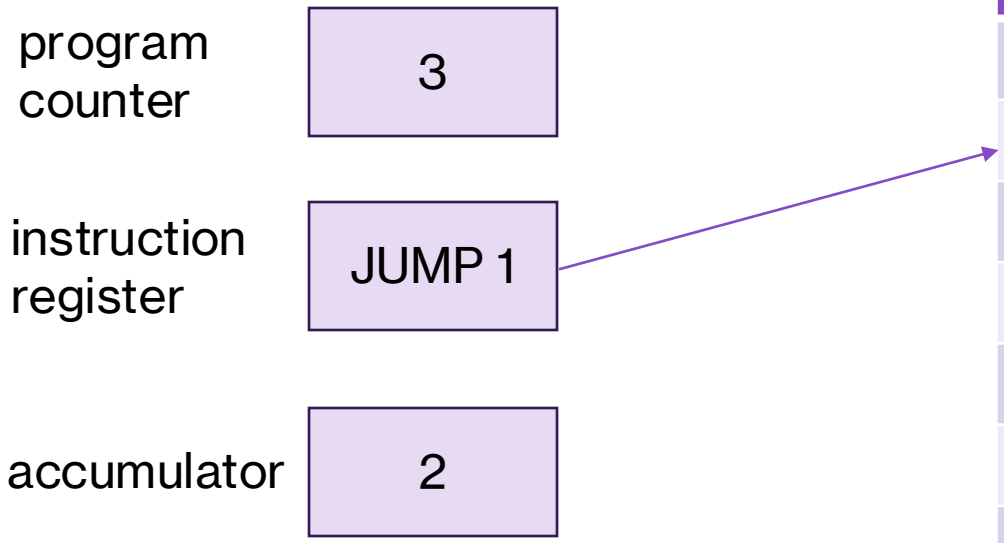

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Fetch



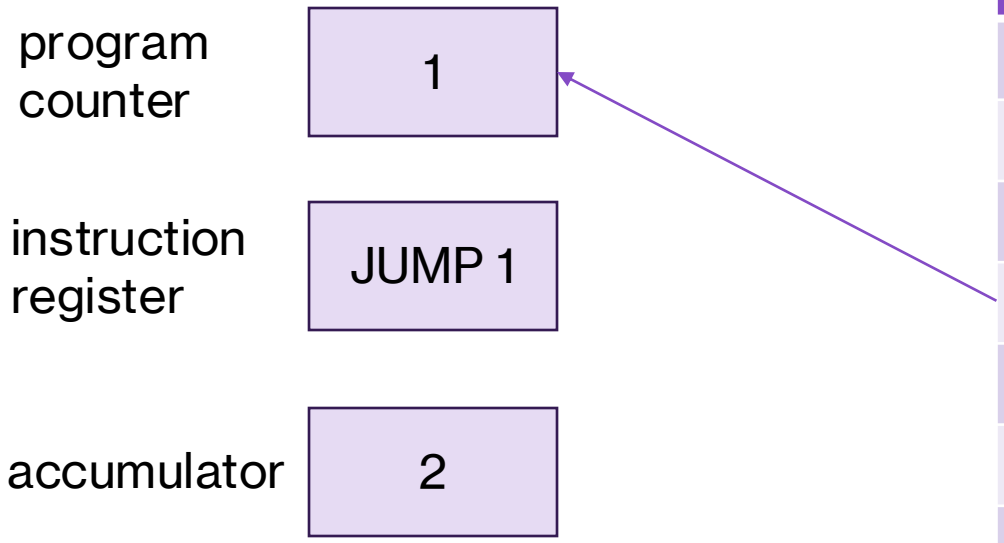

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Decode



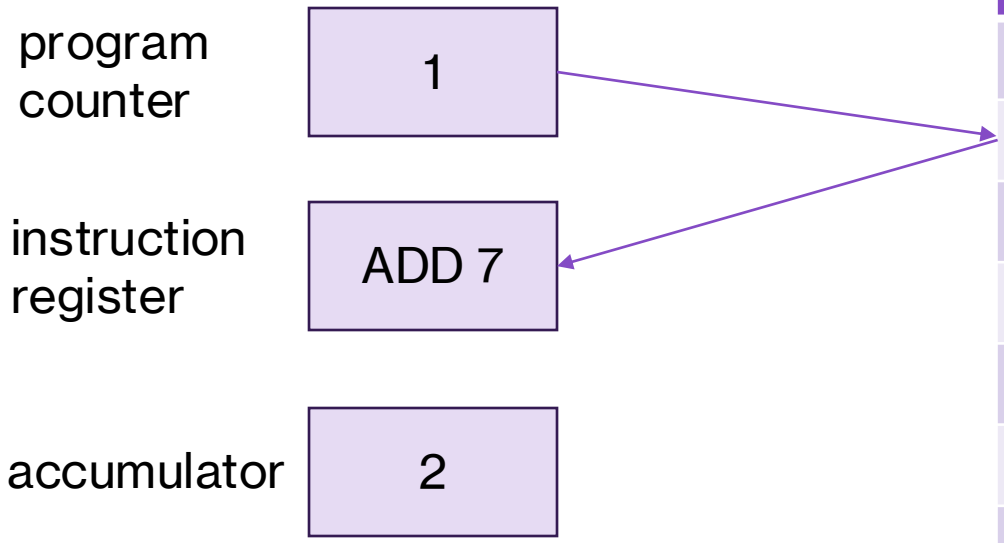

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Execute



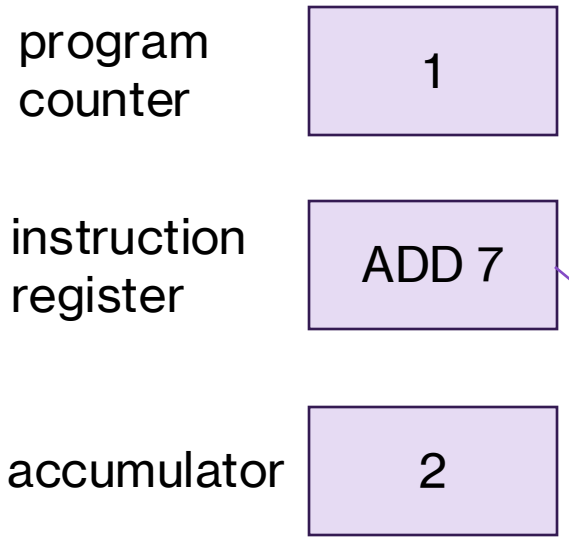

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Fetch



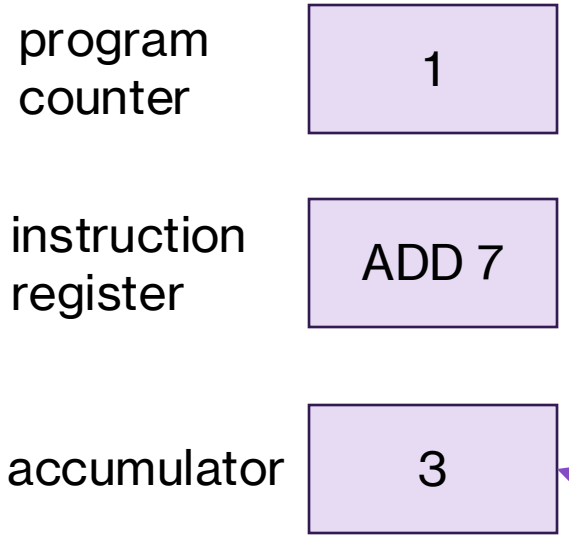

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Decode



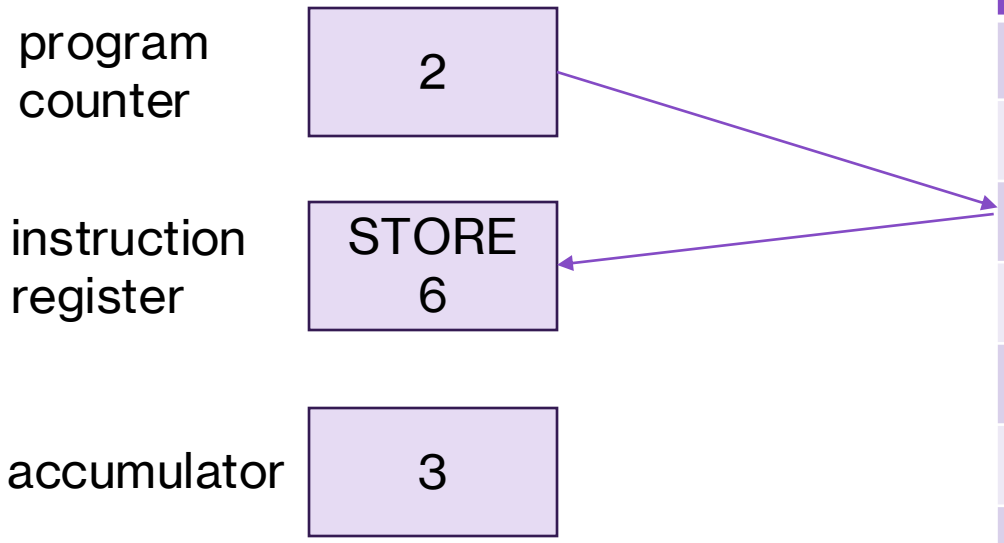

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Execute



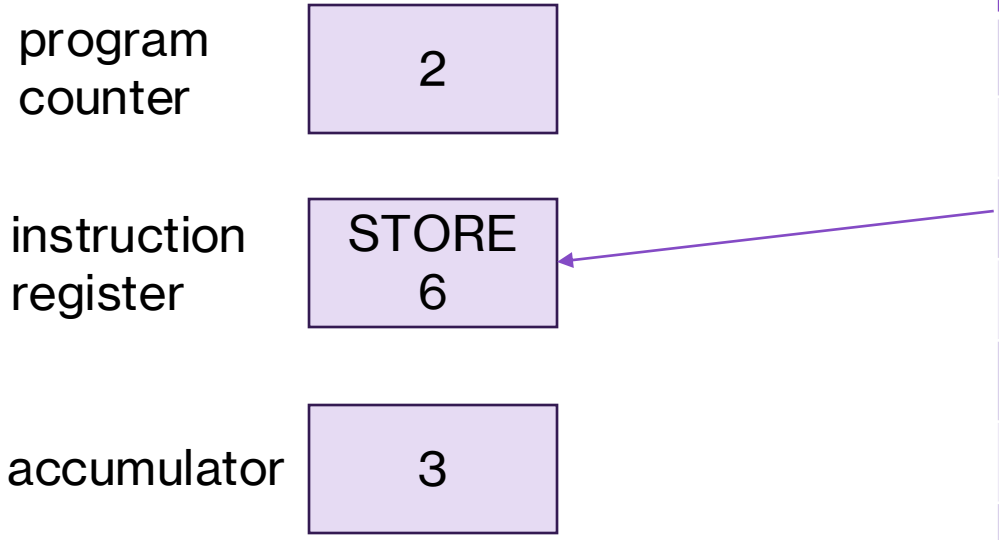

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Fetch



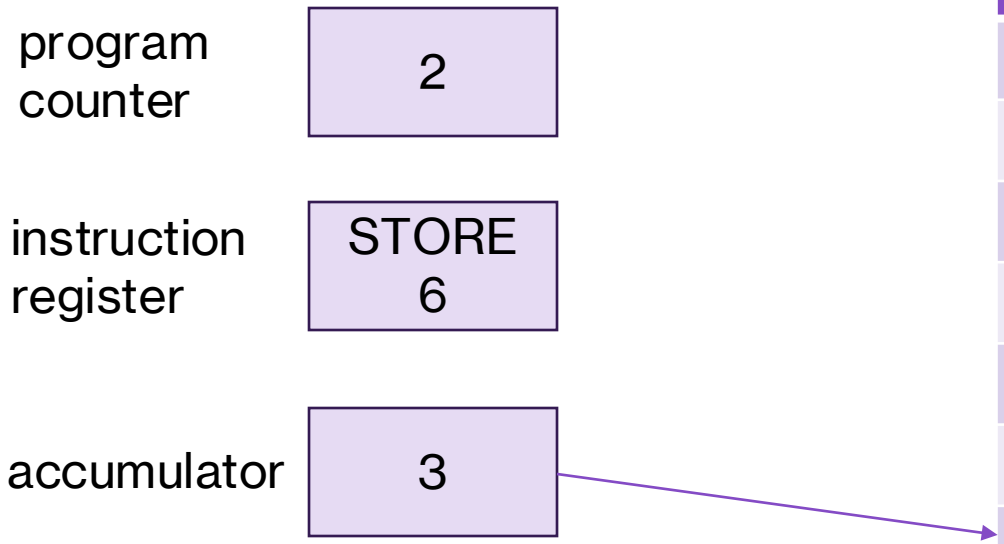

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Decode



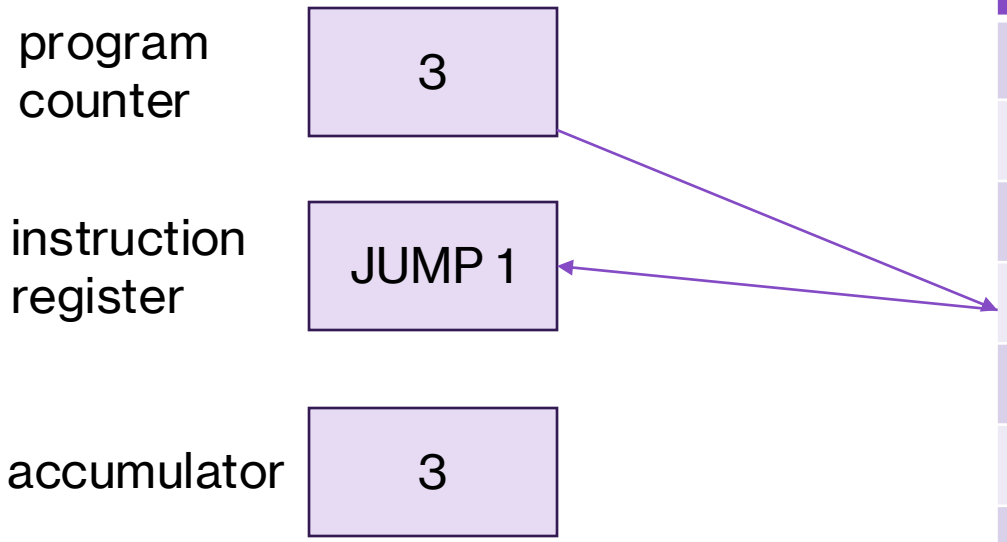

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 1 |

# Executing our program - Execute



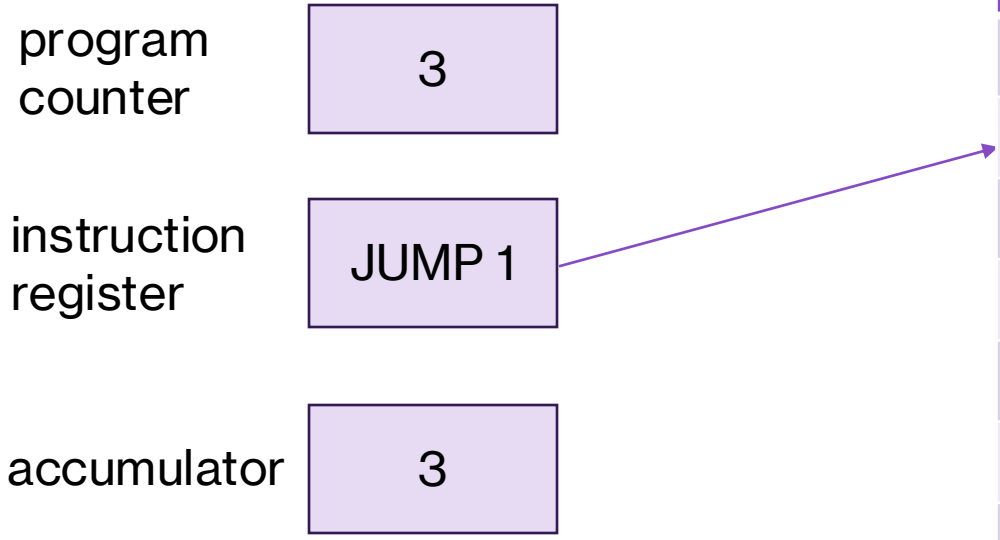

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 3 |
| 7 | 1 |

# Executing our program - Fetch



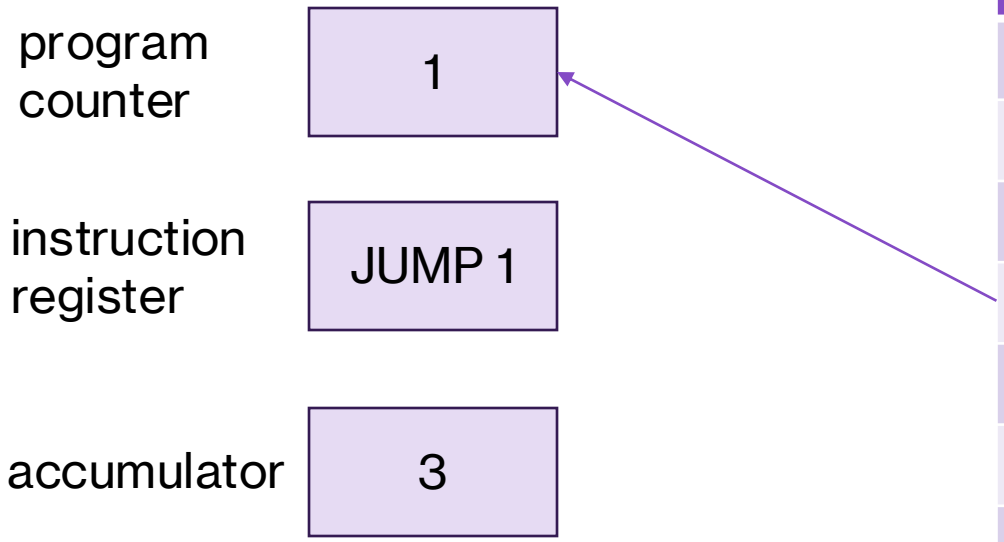

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 3 |
| 7 | 1 |

# Executing our program - Decode



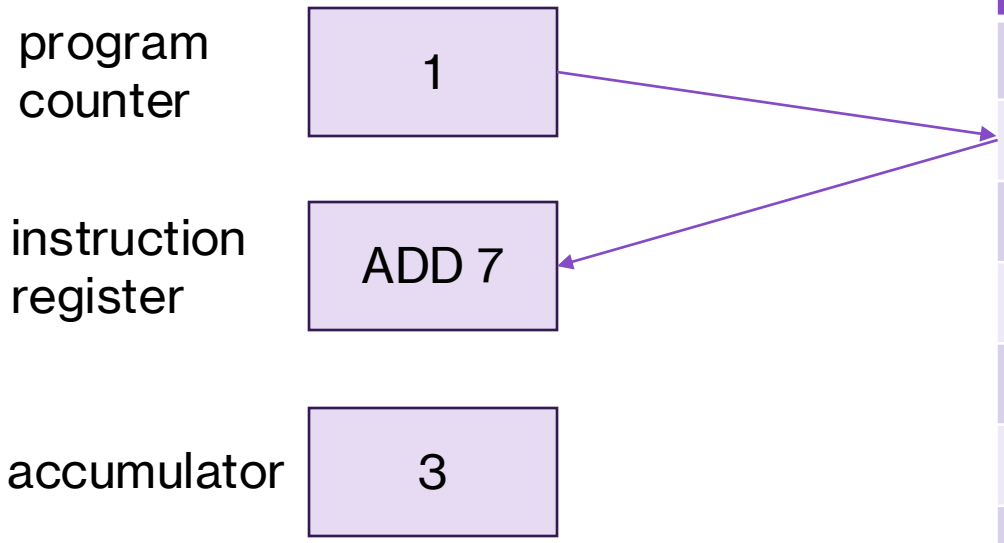

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 3 |
| 7 | 1 |

# Executing our program - Execute



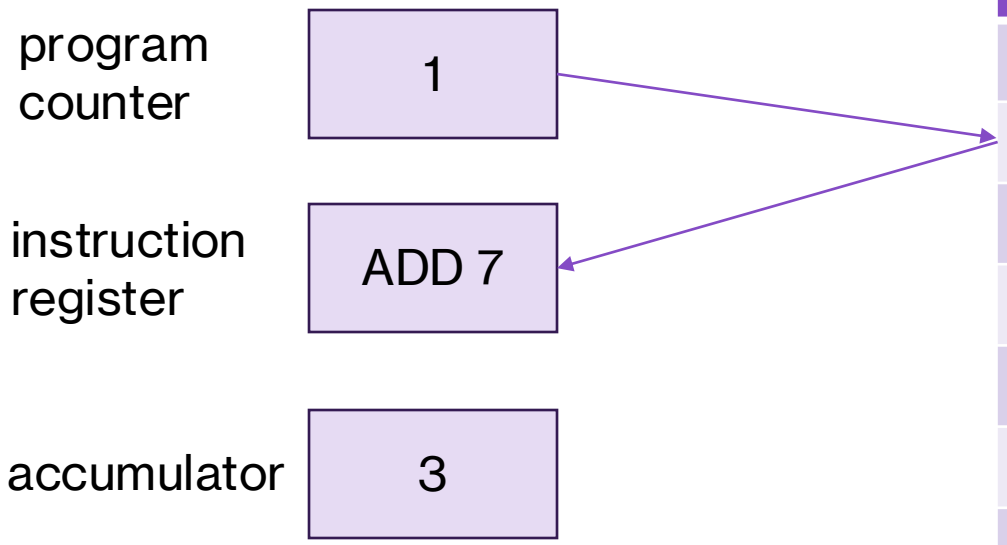

| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 3 |
| 7 | 1 |

# Executing our program - Fetch




| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 3 |
| 7 | 1 |

# What does this program do??




| Address | Data |
|---|---|
| 0 | LOAD 6 |
| 1 | ADD 7 |
| 2 | STORE 6 |
| 3 | JUMP 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 3 |
| 7 | 1 |

# Our program

- It starts with the value 1 at address 6 and it continuously increments it by 1.
  - The increment happens on every ninth clock cycle.

# From 0s and 1s to high-level programming

- Working with 0s and 1s (machine code) or even assembly code is rather hard. This is where high-level programming languages come to play.
- The program we saw would be similar in function to the following Python code. But code in high-level programming languages like Python is much more readable and quickly conveys its functionality:

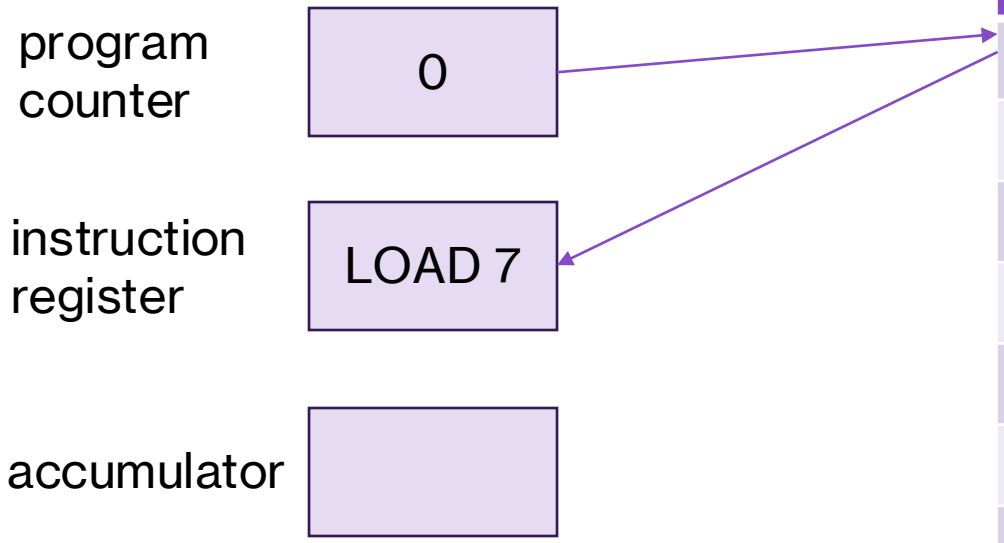
| Machine Code | Assembly Program | High-Level Language (e.g., Python) |
|---|---|---|
| 0001 0110<br>1011 0111<br>0100 0110<br>1101 0001<br>0000 0000<br>0000 0000<br>0000 0001<br>0000 0001 | LOAD 6<br>ADD 7<br>STORE 6<br>JUMP 1<br>0<br>0<br>1<br>1 | x = 1<br>while (True):<br>  x = x + 1 |

# Practice Time

- Assume you are given the following program. What does it do?

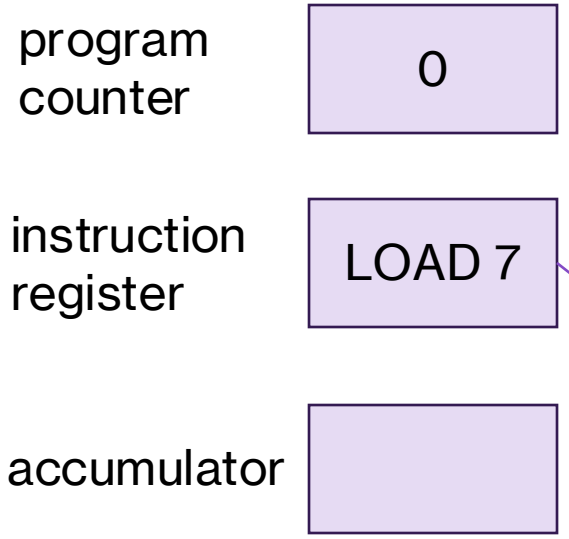
| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Fetch



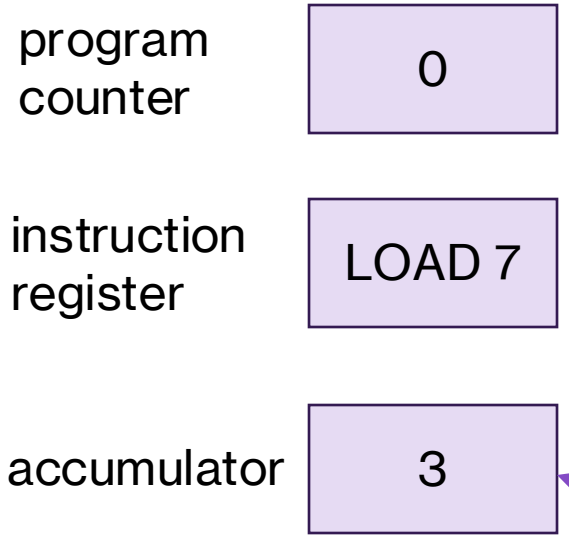

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Decode



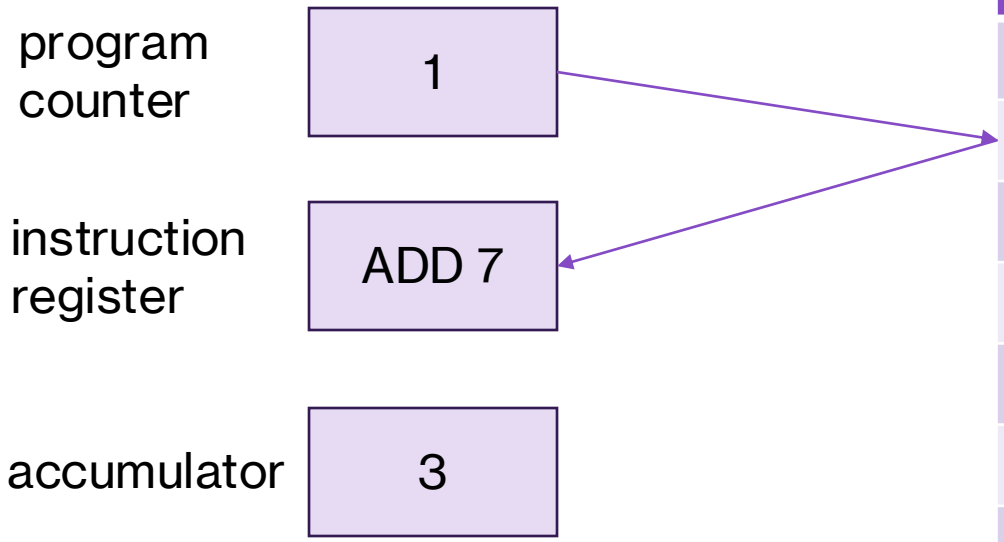

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Execute



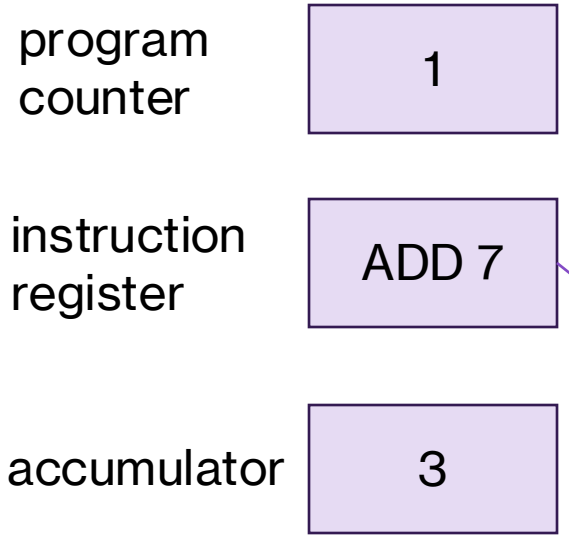

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Fetch



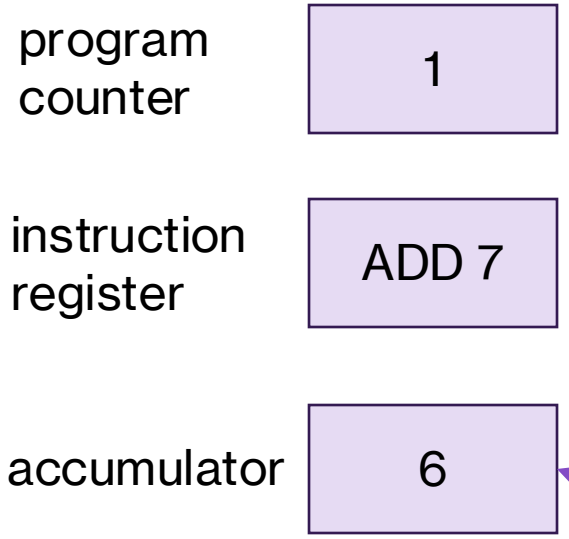

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Decode



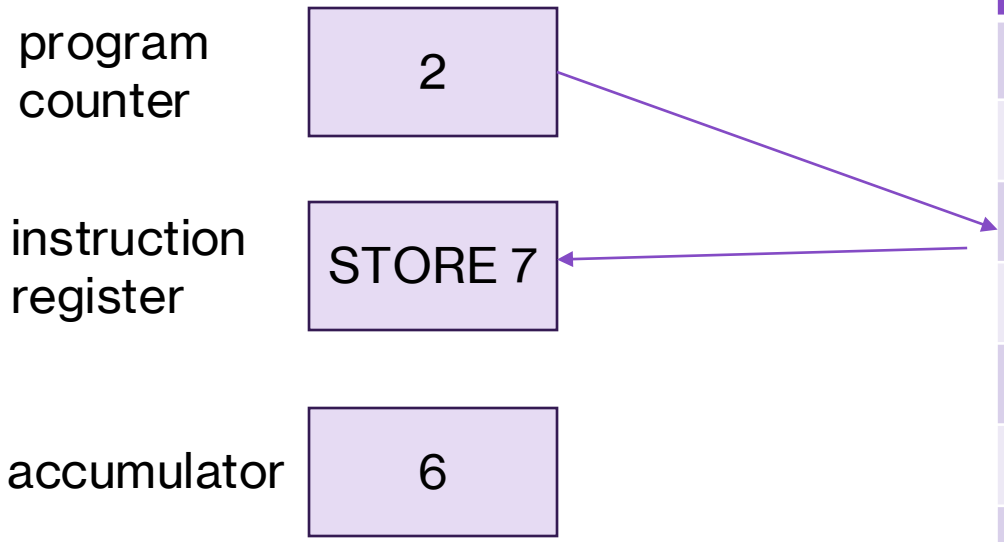

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Execute



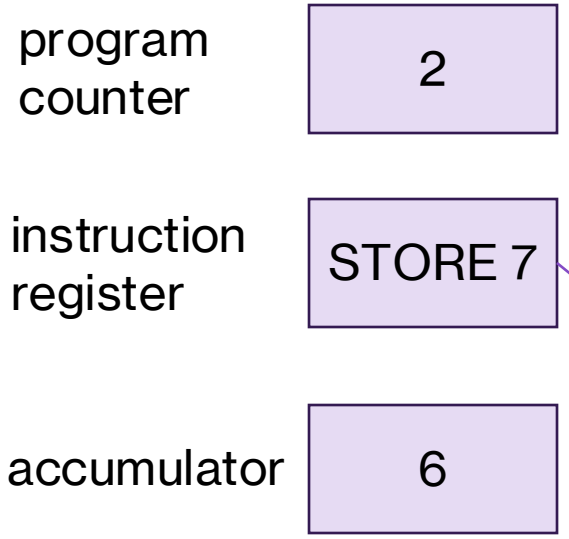

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Fetch



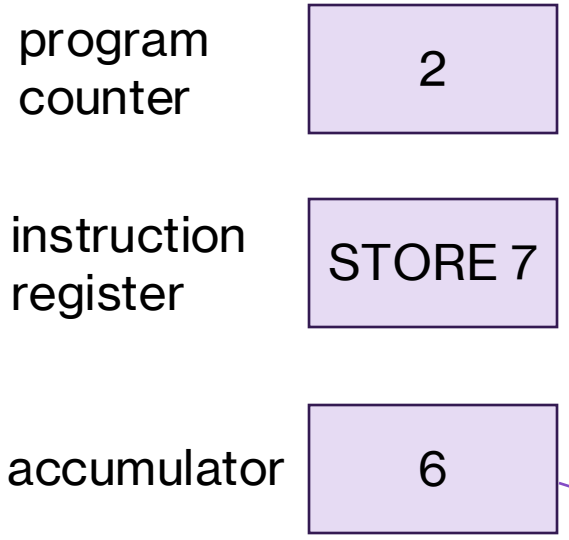

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Decode



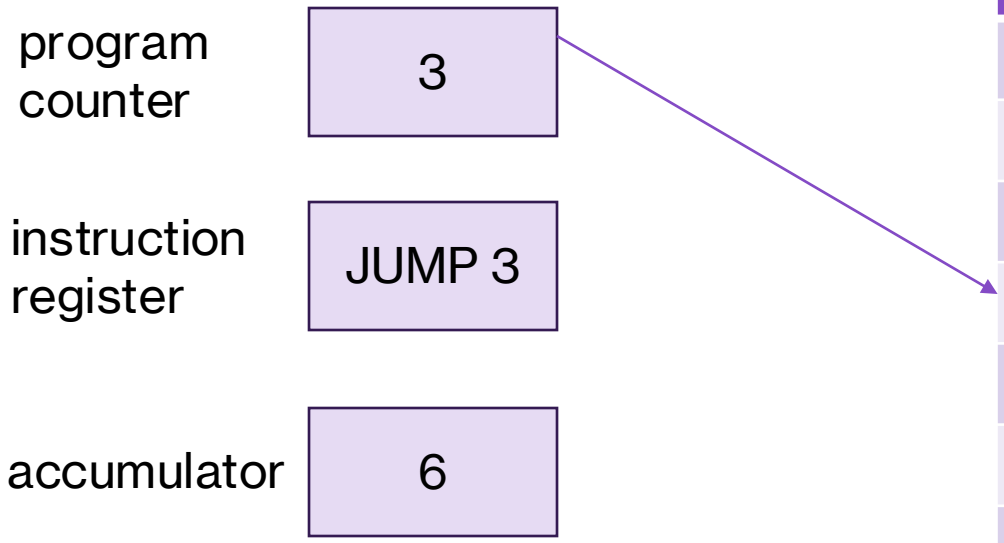

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |

# Executing our program - Execute



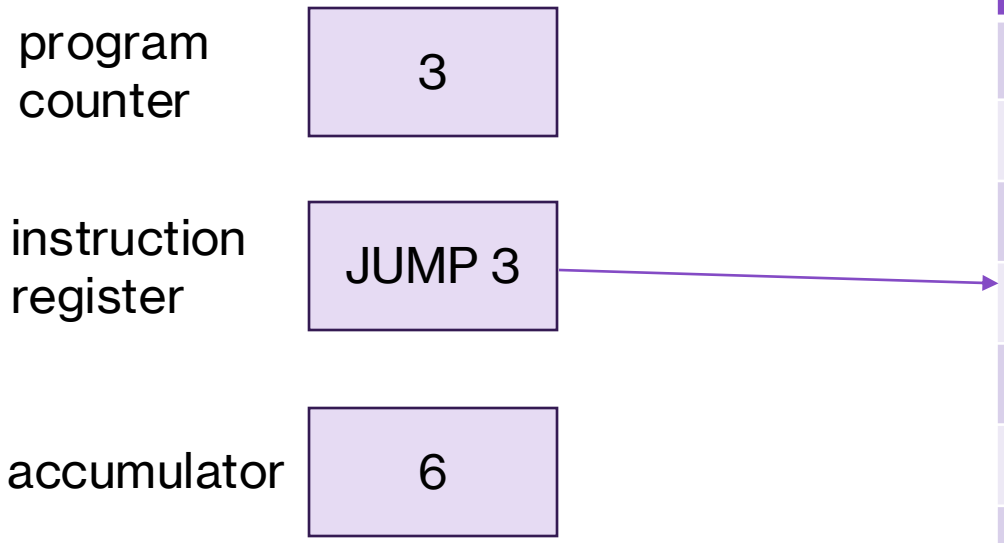

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 6 |

# Executing our program - Fetch



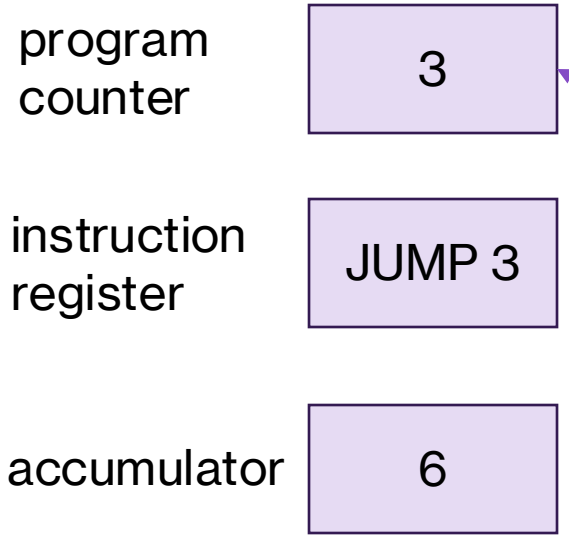

| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 6 |

# Executing our program - Decode




| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 6 |

# Executing our program - Execute




| Address | Data |
|---|---|
| 0 | LOAD 7 |
| 1 | ADD 7 |
| 2 | STORE 7 |
| 3 | JUMP 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 6 |

# Answer

- It doubles the value at address 7 once (i.e. it doubles 3 to 6), then loops infinitely.

# In practice

- In practice, there are a few more registers and more instructions supported, e.g., to halt the execution of the program, call and return from subroutines (functions), jump when certain conditions are met to reflect control flow, and streamline standard logical and arithmetic operations.