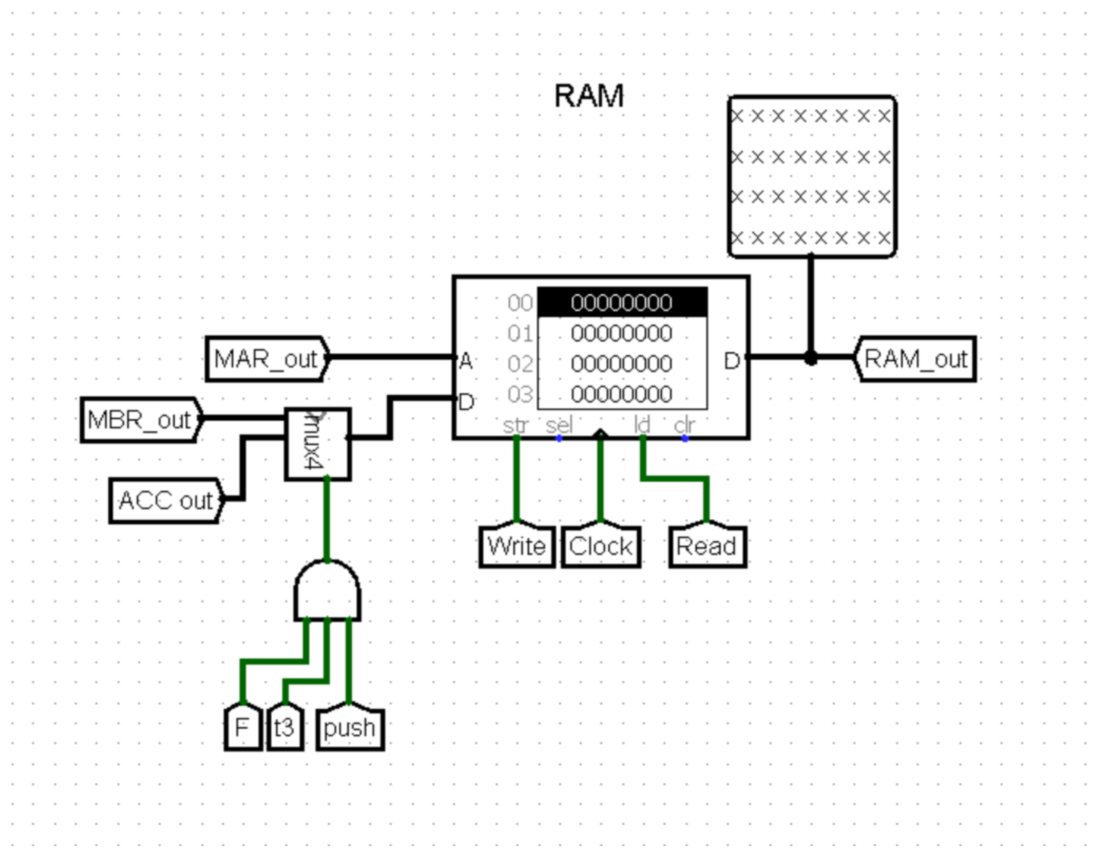
32 Bit CPU System



Name: Tahmid Hossain Chowdhury Mahin

Roll: 2207109

Section: B

Learning Objectives:

Understanding Computer Architecture

Visualizing Complex Systems in logisim

Learning how operations execute in CPU

Introduction:

A Central Processing Unit (CPU) is the main part of a computer that runs programs, does math and logic tasks, and handles data movement. Designing a CPU involves thinking about its structure, commands, speed to make it work well.

A CPU has these parts:

MAR (Memory Address Register): Holds the memory address the CPU needs to find.

MBR (Memory Buffer Register): Keeps data for a short time while moving it between memory and the CPU.

PC (Program Counter): Tracks the address of the next command to run.

AC (Accumulator): Stores results of math and logic tasks temporarily.

IR (Instruction Register): Holds the command the CPU is working on.

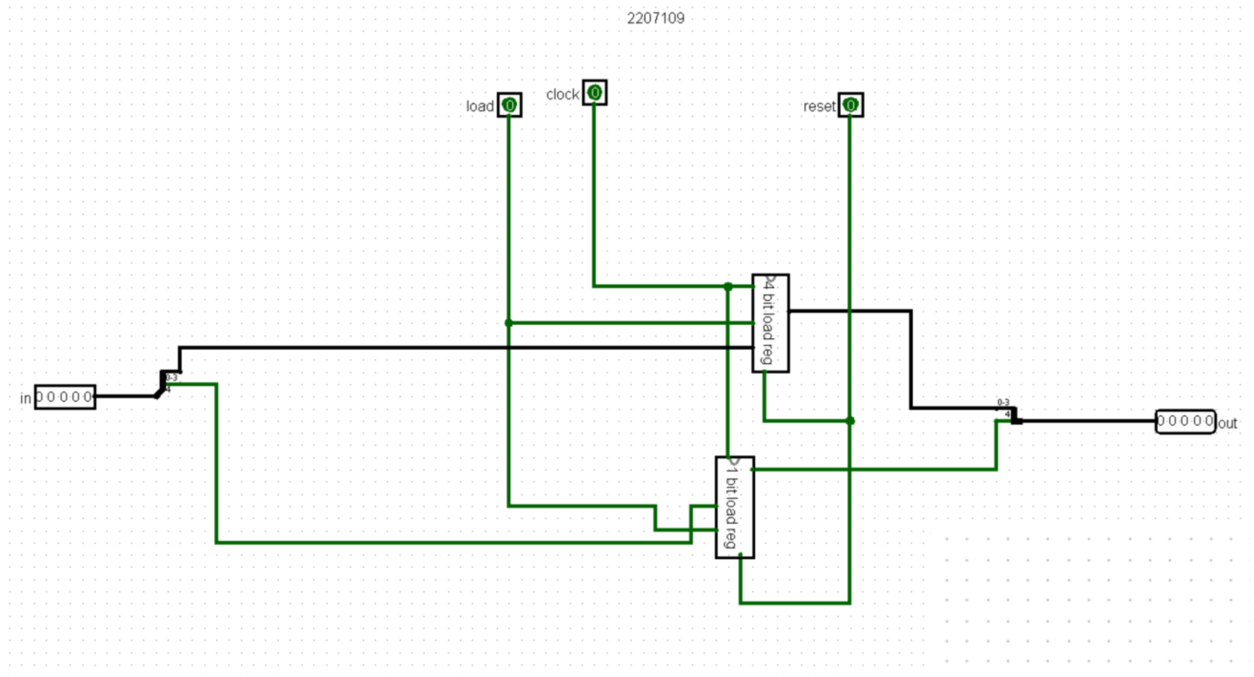
CU (Control Unit): Guides the CPU, controlling how data and commands flow.

Details of the components explained below:

Memory Address Register (MAR):

The Memory Address Register (MAR) is an important part of a computer's Central Processing Unit (CPU). It helps the CPU find data or instructions stored in the computer's memory (RAM). The

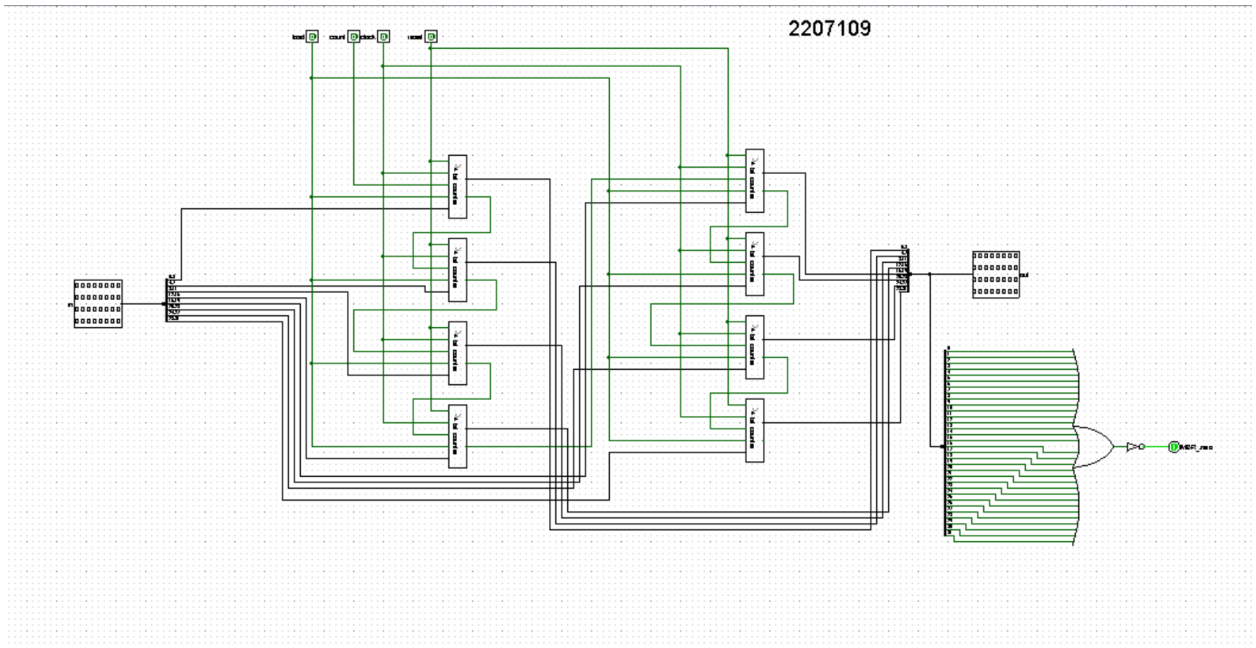
CPU’s Control Unit puts the address into the MAR, which sends it to the memory through the address bus. This helps the CPU fetch instructions from the Program Counter or get data for tasks like calculations, passing it to the Memory Buffer Register.



Memory Buffer Register (MBR):

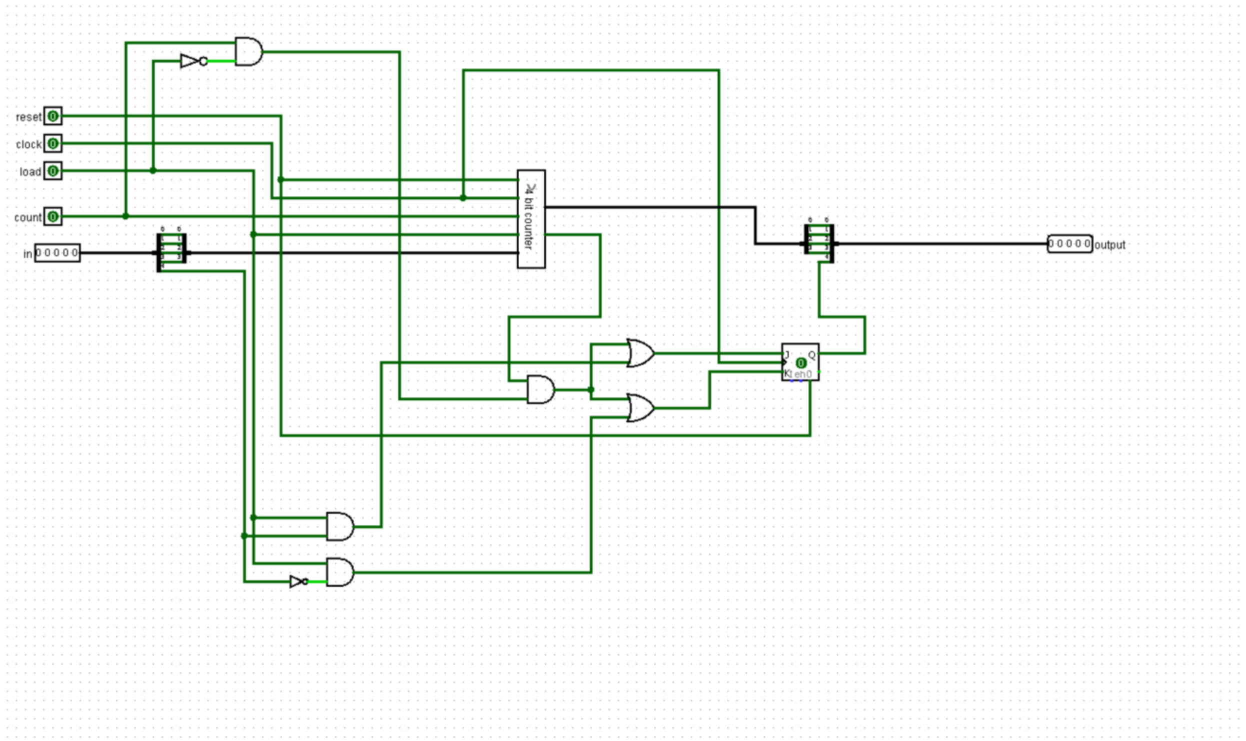
The **Memory Buffer Register (MBR)** temporarily holds data or instructions moving between the CPU and memory (RAM). It stores the content fetched from the memory address in the Memory Address Register (MAR) or data to be saved to memory. During program execution, the MBR passes instructions to the Instruction

Register (IR) or data to the Accumulator (AC). In Logisim, it's shown as a register on the data bus, helping the CPU manage data flow to run programs smoothly.



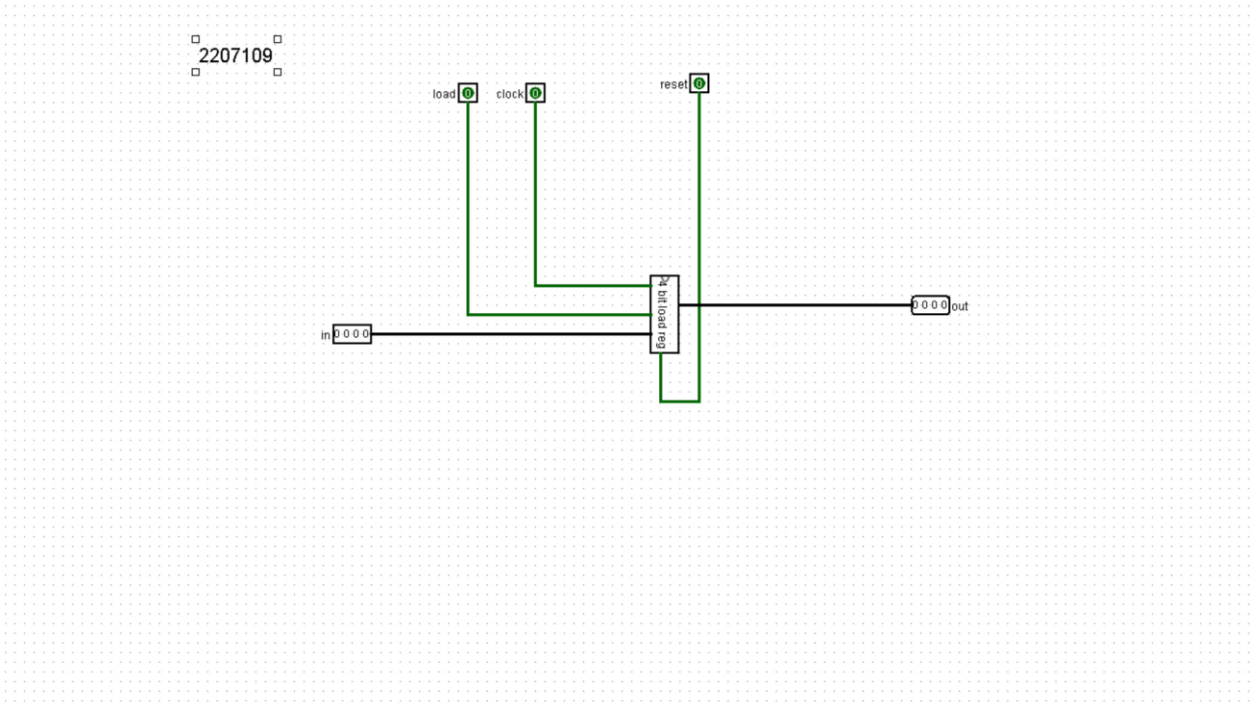
Program Counter (PC):

The Program Counter (PC) is a vital CPU component that keeps track of the memory address of the next instruction to be executed. It acts like a pointer, telling the CPU where to find the next command in memory (RAM). During program execution, the PC sends its stored address to the Memory Address Register (MAR), which fetches the instruction from memory. After the instruction is fetched, the PC automatically increases its value to point to the next instruction.

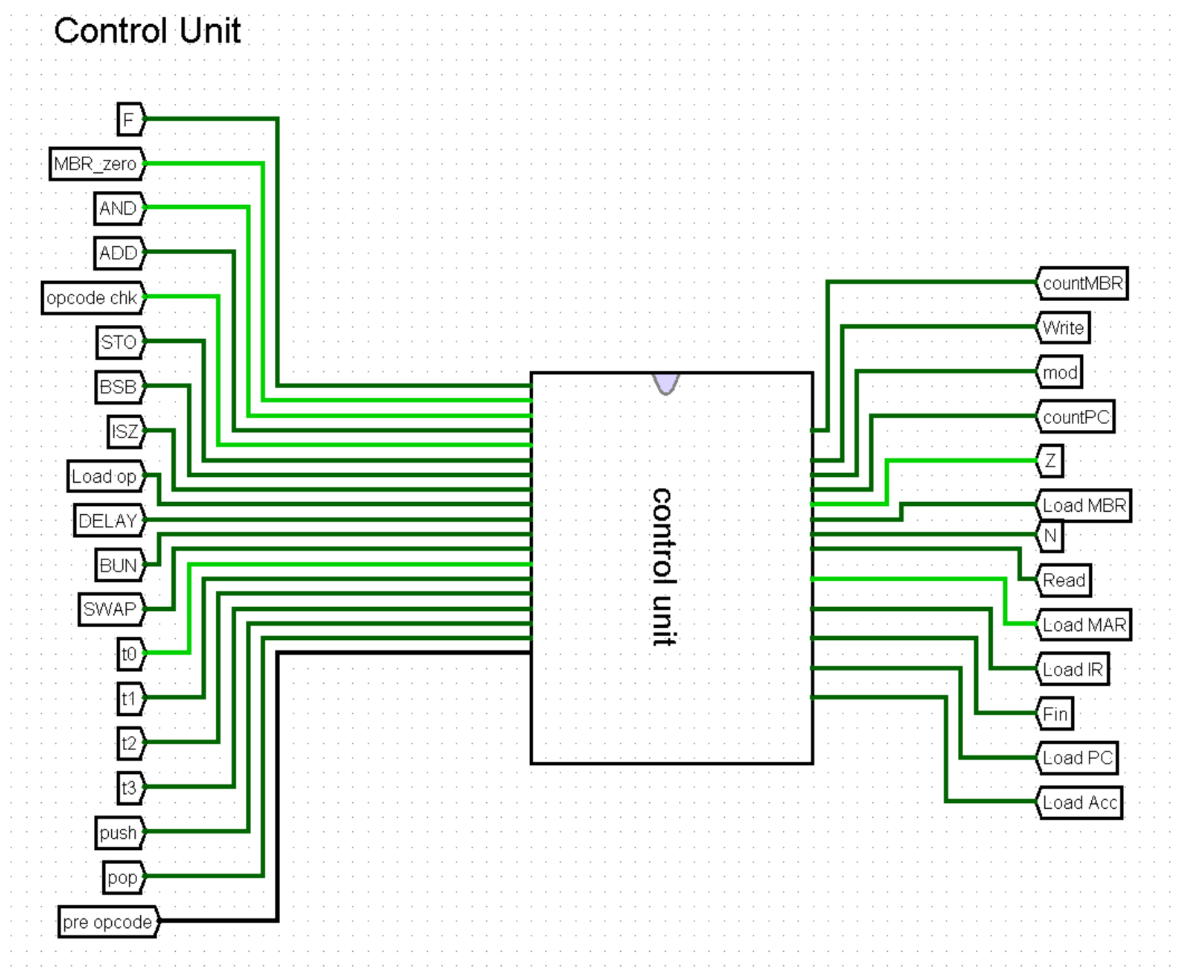


IR (Instruction Register):

The **Instruction Register (IR)** is a CPU component that holds the current instruction being executed. It functions as a temporary storage unit for the instruction fetched from memory. During the fetch phase of the CPU's fetch-decode-execute cycle, the Memory Buffer Register (MBR) transfers the instruction, located at the address specified by the Memory Address Register (MAR), to the IR. The IR then provides the instruction to the Control Unit (CU), which decodes it to determine the CPU's next actions, such as performing calculations or moving data. In Logisim, the IR is represented as a register connected to the MBR and CU, demonstrating how the CPU processes instructions to execute programs accurately.



The **Control Unit (CU)** is a vital CPU component that directs and manages all processor operations to execute program instructions accurately. It oversees the fetch-decode-execute cycle by retrieving the current instruction from the Instruction Register (IR), decoding it to determine the required task, and sending control signals to components like the Arithmetic Logic Unit (ALU), Accumulator (AC), Memory Address Register (MAR), and Memory Buffer Register (MBR) to perform operations such as calculations, data transfers, or memory access. Operating with the CPU’s clock to ensure proper timing, the CU coordinates data flow and synchronizes component actions. In the project, the CU is depicted as a circuit module connected to other CPU parts, illustrating its role in ensuring smooth and precise program execution.



Instructions:

The 32-bit CPU works with a set of instructions that help it do basic math and data tasks, all using 32-bit values. Here's a simple explanation of each instruction for a manual:

- **AND**: Combines the 32-bit value in the Accumulator (AC) with the 32-bit value in the Memory Buffer Register (MBR) using a bitwise AND, and saves the result in the AC.
- **ADD**: Adds the 32-bit value in the AC to the 32-bit value in the MBR, and stores the result in the AC.
- **STORE**: Moves the 32-bit contents of the AC to a specific spot in Main Memory.
- **ISZ** : Moves to the next instruction and skips the one after if the last operation's 32-bit result was zero.
- **BSB (Branch to Sub-routine)**: Saves the Program Counter (PC) value with the BUNinstruction to a memory spot and starts running the next instruction.
- **BUN(Unconditional Jump)**: Goes to a specific 32-bit memory address right away.
- **LOAD**: Takes data from a specific 32-bit memory spot and puts it into the AC.
- **HALT**: Stops the program from running.

Special Instructions:

- **DELAY:** The delay instruction is a special command in the CPU that makes the processor wait for a set time based on a number from memory, with extra time added. It helps control timing in programs and works within the CPU. The Control Unit (CU) manages this process.

- ***What the Delay Instruction Does***

- The delay instruction stops the CPU for a while by:
- **Getting a value**: It takes a 32-bit value from a given memory spot, given by the instruction.
- **Adding Extra Time**: It adds three extra clock pulses for each unit of that value. For example, if the number is 5, it waits for 5 units plus 15 extra pulses (3 × (5-1)), making a total of 12 pulses extra.
- **Pausing the CPU**: The processor stops all work for that calculated time before moving to the next step.
- **Adjusting Timing**: This lets the CPU handle different delay lengths, useful for tasks needing exact timing.

- **MOD: The** modulus instruction is added to 32-bit CPU that calculates the remainder (modulus) when the value in the Accumulator (AC) is divided by the value in the Memory Buffer Register (MBR). It works by repeatedly subtracting the MBR value from the AC until the result is less than the MBR value, giving the modulus. This instruction operates within the CPU's 32-bit architecture using the fetch-decode-execute cycle, managed by the Control Unit (CU).

- *What the Modulus Instruction Does*

- The modulus instruction finds the remainder after division by:
- **Starting Values**: It uses the 32-bit value in the AC as the number to divide and the 32-bit value in the MBR as the divisor.
- **Repeated Subtraction**: It keeps subtracting the MBR value from the AC until the AC becomes smaller than the MBR. The final value left in the AC is the modulus.
- **Result Storage**: The remaining value in the AC is the modulus, which is the leftover after division.

- **SWAP:** The swap instruction is a feature in the CPU that swaps the value in the Accumulator (AC) with a value stored in a specific memory location. This instruction helps the CPU move 32-bit data between the AC and memory, making data handling easier. It works within the CPU's 32-bit system using the fetch-decode-execute cycle, controlled by the Control Unit (CU).

  - *What the Swap Instruction Does*

  - The swap instruction switches data by:
  - **Identifying Values**: It uses the 32-bit value in the AC and a 32-bit value from a chosen memory spot.
  - **Exchanging Data**: It puts the memory value into the AC and moves the original AC value into that memory spot.

- **PUSH TO STACK:** The **push** instruction is a feature in CPU that stores the value in the Accumulator (AC) onto a stack, using the last memory address of the RAM as the stack pointer, specifically for data storage. This instruction helps save 32-bit data onto the stack for later use, such as preserving values during program execution.

- ***What the Push Instruction Does***

- The push instruction saves data by:
- **Storing the Value**: It takes the 32-bit value from the AC and places it into the memory location pointed to by the stack pointer.
- **Updating the Stack Pointer**: After storing, it decreases the stack pointer by one word for the next push.

- **POP TO STACK:** The pop instruction is a feature that retrieves value using stack pointer from the stack into the Accumulator (AC), using the last memory address of the RAM as the stack pointer, specifically for data stored by push. This instruction helps restore 32-bit data saved earlier, such as values preserved during program execution.

    - ***What the Pop Instruction Does***

    - The pop instruction retrieves data by:
    - **Loading the Value**: It takes the 32-bit value from the memory location pointed to by the stack pointer and places it into the AC.

  - **Updating the Stack Pointer**: After retrieving, it increases the stack pointer by one word to point to the next value on the stack.

Discussion:

This project integrates a comprehensive set of components—Memory Address Register (MAR), Memory Buffer Register (MBR), Program Counter (PC), Accumulator (AC), Instruction Register (IR), Control Unit (CU), and Arithmetic Logic Unit (ALU) to execute a variety of 32-bit instructions within a fetch-decode-execute cycle. The CPU supports core operations like AND and ADD for bitwise and arithmetic tasks, alongside data management instructions such as STORE, LOAD, and SWAP, which facilitate efficient data movement between the AC and memory.

The control flow instructions—SKIP, BSB (Branch to Sub-routine), BUN (Unconditional Jump), and HALT—enhances the CPU's ability to manage program execution. To address timing needs, there is DELAY instruction, which fetches a 32-bit value from memory and pauses execution for that value plus three extra clock pulses per unit, offering precise timing control. The MODULUS instruction, which repeatedly subtracts the MBR value from the AC until the remainder is found, provides a practical solution for division-related calculations, expanding the CPU's computational range.

A significant feature of this design is the stack implementation, where I designated the last memory address of the RAM as the initial stack pointer, exclusively for data storage and retrieval.

Conclusion:

We learned that this CPU design effectively combines key components such as the MAR, MBR, PC, AC, IR, CU, and ALU to execute a comprehensive set of 32-bit instructions through a fetch-decode-execute cycle. It supports bitwise and arithmetic operations (AND, ADD), data management instructions (STORE, LOAD, SWAP), and control flow instructions (SKIP, BSB, BUN, HALT) for streamlined program execution. The DELAY instruction ensures precise timing control, while the MODULUS instruction expands computational capabilities for division tasks. We also learned that the stack, utilizing the last RAM address as the initial stack pointer, optimizes data storage and retrieval, resulting in a versatile and efficient CPU architecture.

Reference:

1. Digital Logic and Computer Design
By M. Morris Mano
2. Lab Slides