# Basic CPU Components

This week's project simulates the basic operations of a CPU, allowing users to interact with the system through load, execute, write, read, and exit commands. The algorithm follows a sequence of operations using an arithmetic logic unit (ALU) and a set of general-purpose registers to perform basic arithmetic and logical instructions.

## Algorithm:

### 1. Initialization:

- Initialize a set of 4 registers (R0 to R3), each set to 0.
- Initialize the program counter (PC) to 0.
- Initialize an instruction register to store the current instruction.

### 2. Input and Actions:

- **Write Action**:
    - Ask the user to select a register (R0 to R3) and input a value to store in the chosen register.
    - Store the input value in the selected register.
- **Read Action**:
    - Ask the user to select a register (R0 to R3).
    - Output the current value stored in the selected register.
- **Load Instruction**:
    - Ask the user to input an instruction (e.g., ADD 0 1 2).
    - The instruction is parsed to identify the operation (ADD, SUB, AND, OR, NOT) and the registers involved.
    - Store the instruction in the instruction register.
- **Execute Instruction**:
    - Fetch the instruction from the instruction register.
    - Depending on the operation, execute the corresponding arithmetic or logical operation on the specified registers using the ALU.
    - Store the result in the specified register.
    - Increment the program counter (PC) to move to the next instruction.

### 3. Operations:

- **Arithmetic Operations**:
    - ADD: Add the values in two registers and store the result in another register.
    - SUB: Subtract the value of one register from another and store the result.

    - AND: Perform a bitwise AND operation on two registers.
    - OR: Perform a bitwise OR operation on two registers.
    - NOT: Perform a bitwise NOT operation on a register and store the result.

**4. Error Handling:**

- Ensure valid register numbers (between 0 and 3).
- Validate the format of the input instruction.
- Check for invalid operations and provide feedback to the user.

**5. Program Flow:**

- The program continuously prompts the user for actions until they choose to exit.
- Each operation (load, execute, write, read) is handled in a loop.
- The program terminates when the user selects the "exit" option.

## Step-by-Step Algorithm:

1. **Initialize Registers**:
    - Set all registers (R0 to R3) to 0.
    - Set the program counter (PC) to 0.
2. **Repeat until Exit**:
    - **Display options**: Provide the user with the following options:
        - load: Load an instruction into the instruction register.
        - execute: Execute the loaded instruction.
        - write: Store a value in a register.
        - read: Display the value of a register.
        - exit: Terminate the program.
3. **Write Action**:
    - Input the register number (0-3).
    - Input a value to store in the selected register.
    - Store the value in the chosen register.
4. **Read Action**:
    - Input the register number (0-3).
    - Output the value stored in the selected register.
5. **Load Instruction**:

   - Input an instruction (e.g., ADD 0 1 2).
     - Parse the instruction to identify the operation (ADD, SUB, etc.) and the involved registers.
   - Store the parsed instruction in the instruction register.
6. **Execute Instruction**:
   - Fetch the instruction from the instruction register.
   - Depending on the operation in the instruction:
     - **ADD**: Add values in the two specified registers and store the result in the first register.
     - **SUB**: Subtract the second register’s value from the first and store the result in the first register.
     - **AND**: Perform bitwise AND on the two specified registers and store the result in the first register.
     - **OR**: Perform bitwise OR on the two specified registers and store the result in the first register.
     - **NOT**: Perform bitwise NOT on the second register and store the result in the first register.
   - Increment the program counter (PC) by 1.
7. **Error Handling**:
   - Ensure that only valid registers (0-3) are selected.
   - Check for valid instruction formats and provide feedback for incorrect input.
8. **Exit**:
   - Exit the loop when the user chooses the exit option.

## Sample Input and Output:

**Input:**

**Choose action (load, execute, write, read, exit):** write
**Enter register number (0-3):** 2
**Enter value to store:** 15

**Choose action (load, execute, write, read, exit):** write
**Enter register number (0-3):** 1
**Enter value to store:** 10

**Choose action (load, execute, write, read, exit):** load
**Enter instruction (e.g., ADD 0 1 2):** ADD 0 1 2

**Choose action (load, execute, write, read, exit):** execute

**Output:**

```
Stored 15 in R2
Stored 10 in R1
Loaded instruction: ADD 0 1 2
Executing instruction at PC=0: ADD 0 1 2
Stored 25 in R0
```

## Conclusion:

The algorithm effectively simulates the functioning of a CPU by allowing users to interact with it using simple arithmetic and logical operations. By storing values in registers, executing instructions, and providing feedback on the program state (through reading registers and executing commands), this simulation provides an interactive environment for learning CPU operations.

This project introduces basic concepts such as the fetch-execute cycle, register management, and error handling, which are foundational in understanding how CPUs execute instructions and manage data.