Instruction Execution

This documentation outlines the development and implementation of the **Fetch-Decode-Execute Cycle** using Python for a basic CPU simulation. The goal of Week 4 was to create a simplified CPU model capable of executing basic instructions through a structured cycle.

Objective

- Develop a basic CPU model that can fetch instructions from memory, decode them, and execute using an ALU and registers.
- Test the cycle with a set of sample instructions to ensure correct operation.

Fetch-Decode-Execute Cycle: Overview

The cycle is divided into three main stages:

1. **Fetch**: Retrieve the next instruction from memory.
2. **Decode**: Break down the instruction into its components (opcode and operands).
3. **Execute**: Perform the operation using the ALU and registers.

Algorithm for Instruction Execution

Step 1: Initialization

1. Create a **memory** list containing a set of instructions.
2. Initialize the **Program Counter (PC)** to 0.
3. Create a **register file** (e.g., R1, R2) and set initial values to 0.
4. Instantiate the ALU (Arithmetic Logic Unit) for performing operations.

Step 2: Fetch Stage

1. Retrieve the instruction from memory at the address pointed to by the PC.
2. Store the fetched instruction in the **Instruction Register (IR)**.
3. Increment the PC to point to the next instruction.

Step 3: Decode Stage

1. Split the fetched instruction into:
    - **Opcode**: Specifies the operation (e.g., LOAD, ADD, STORE).
    - **Operands**: Specifies registers and/or immediate values.
2. Identify the type of operation (e.g., arithmetic, load, store).

Step 4: Execute Stage

1. Based on the opcode, perform the following operations:
    - **LOAD**: Move an immediate value into a register.
    - **ADD**: Add values from two registers and store the result in a target register.
    - **STORE**: Output the value of a register to a memory location.
2. Update the register file with the result or print output for a store operation.

Complete Algorithm

1. Initialize memory with a list of instructions.
2. Set Program Counter (PC) to 0.
3. Initialize registers R1 and R2 to 0.
4. Repeat until all instructions are executed:
   a. Fetch the instruction at memory[PC].
   b. Increment PC.
   c. Decode the instructions to extract opcode and operands.
   d. Perform the operation based on the opcode:
      i. If LOAD, move the value to the specified register.
      ii. If ADD, add values from two registers and update the target register.
      iii. If STORE, output the result.

Conclusion

The Fetch-Decode-Execute cycle was successfully implemented in Python. The project demonstrated how a basic CPU processes instructions sequentially, decodes them, and executes arithmetic or memory operations. This simulation helped reinforce concepts of CPU operations and data handling through a structured approach.