Implement the instruction fetching mechanism

The instruction fetching mechanism is a critical part of a computer's Central Processing Unit (CPU). It retrieves the next instruction to be executed from memory. Here’s an explanation of its implementation:

Components of Instruction Fetching Mechanism

1. Program Counter (PC):

Holds the address of the next instruction to be executed.

Initially points to the memory address of the first instruction.

2. Instruction Register (IR):

Temporarily holds the fetched instruction.

3. Memory Unit:

Stores the program instructions and data.

4. Control Unit:

Orchestrates the fetching, decoding, and execution of instructions.

5. System Bus:

Facilitates communication between CPU and memory.

Steps in Instruction Fetching

1. Fetch Address from PC:

The CPU reads the address stored in the PC.

2. Memory Read:

The address in the PC is sent to the memory unit via the address bus.

The memory unit retrieves the instruction stored at the specified address and places it on the data bus.

3. Load into IR:

The fetched instruction from the memory is loaded into the Instruction Register (IR).

4. Increment PC:

The PC is incremented to point to the next instruction. This can be:

A fixed increment (e.g., +1 for sequential instructions).

A modified value (for branch or jump instructions).

5. Decode and Execute:

The Control Unit decodes the instruction in the IR and signals the necessary components to execute it.

Hardware Implementation Example

Sequential Logic:

The PC is implemented as a register with an incrementer to update its value after each fetch cycle.

Clock Cycle:

Each fetch operation is synchronized with the system clock.

Pseudocode for Instruction Fetching

START:

1. PC → Address Bus // Place PC on address bus
2. Memory[PC] → IR // Fetch instruction from memory into IR
3. PC ← PC + 1 // Increment PC
4. Decode and Execute(IR) // Decode and execute instruction
5. Repeat

END

This basic mechanism ensures the CPU can continuously fetch and execute instructions in a loop, forming the foundation of all computer operations.

Here's a C++ implementation of the instruction-fetching mechanism as part of a simple virtual CPU emulator:

C++ Code: Virtual CPU Emulator

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <unordered_map>
```

```
class VirtualCPU {
private:
    int PC;
    std::string IR;
    std::vector<std::string> memory;
    std::unordered_map<int, int> registers;
    bool running;

public:
    VirtualCPU(const std::vector<std::string>& program)
        : PC(0), memory(program), running(true) {
        // Initialize registers (R0 to R7) to 0
        for (int i = 0; i < 8; ++i) {
            registers[i] = 0;
        }
    }

    void fetch() {
        if (PC < memory.size()) {
            IR = memory[PC]; // Fetch instruction at the current PC
            std::cout << "Fetched Instruction: " << IR << "\n";
            ++PC;
        } else {
            running = false; // Stop if PC goes out of bounds
        }
    }

    void decodeAndExecute() {
```

```
    if (IR.empty()) return;

    std::istringstream iss(IR);
    std::string opcode;
    iss >> opcode;

    if (opcode == "LOAD") {
        int reg, value;
        iss >> reg >> value;
        registers[reg] = value;
    } else if (opcode == "ADD") {
        int reg1, reg2, reg3;
        iss >> reg1 >> reg2 >> reg3;
        registers[reg1] = registers[reg2] + registers[reg3];
    } else if (opcode == "SUB") {
        int reg1, reg2, reg3;
        iss >> reg1 >> reg2 >> reg3;
        registers[reg1] = registers[reg2] - registers[reg3];
    } else if (opcode == "HALT") {
        running = false;
    } else {
        std::cout << "Unknown instruction: " << opcode << "\n";
    }
}

void run() {
    while (running) {
        fetch();
        decodeAndExecute();
```

```
            displayRegisters();
        }
    }

    void displayRegisters() const {
        std::cout << "Registers: ";
        for (int i = 0; i < 8; ++i) {
            std::cout << "R" << i << "=" << registers.at(i) << " ";
        }
        std::cout << "\n";
    }
};

int main() {
    // Example instruction set
    std::vector<std::string> program = {
        "LOAD 0 5",
        "LOAD 1 10",
        "ADD 2 0 1",
        "SUB 3 1 0",
        "HALT"
    };
    VirtualCPU cpu(program);
    cpu.run();

    return 0;
}
```