**Documentation for Simple CPU Emulator**

**1. Structure Definition**

**Instruction Structure**

struct Instruction {

string opcode;

int operand1, operand2;

};

* **Purpose**: Represents a single machine instruction.
* **Fields**:
  + opcode: The operation code for the instruction.
  + operand1, operand2: Operands used for the instruction execution.

**2. Memory Management**

**Memory Class**

class Memory {

public:

Memory(int size) : memory(size, 0) {}

int read(int address) {

return memory[address];

}

void write(int address, int value) {

memory[address] = value;

cout << memory[address] << endl;

}

private:

vector<int> memory;

};

* **Purpose**: Simulates a memory unit.
* **Constructor**:
  + Initializes memory with a given size (default value of 0 for all locations).
* **Methods**:
  + read(address): Reads data from a specified memory address.
  + write(address, value): Writes data to a specified memory address.

**3. Reading Instructions**

vector<Instruction> readInstructions(string filename) {

vector<Instruction> instructions;

ifstream file(filename);

string line;

while (getline(file, line)) {

Instruction inst;

istringstream iss(line);

iss >> inst.opcode >> inst.operand1 >> inst.operand2;

instructions.push\_back(inst);

}

return instructions;

}

* **Purpose**: Reads instructions from a file.
* **Input**: File name containing machine code instructions.
* **Output**: A vector of Instruction objects.

**4. Opcode Mapping**

map<string, string> opcodeMap = {

{"LOAD", "00"},

{"ADD", "01"},

{"STORE", "10"},

{"SUB", "11"},

{"AND", "000"},

{"OR", "001"},

{"XOR", "010"},

{"HALT", "1111"}

};

* **Purpose**: Maps human-readable assembly mnemonics to their corresponding machine code opcodes.

**5. Assembler**

void assemble(string filename) {

fstream inputFile(filename);

fstream outputFile("machine\_code.txt");

string line;

while (getline(inputFile, line)) {

istringstream iss(line);

string opcode, operand1, operand2;

iss >> opcode >> operand1 >> operand2;

string machineCode = opcodeMap[opcode] + " " + operand1 + " " + operand2;

outputFile << machineCode << endl;

}

inputFile.close();

outputFile.close();

}

* **Purpose**: Converts assembly instructions into machine code.
* **Input**: File name containing assembly instructions.
* **Output**: Writes machine code to machine\_code.txt.

**6. Main Program**

int main() {

assemble("instructions.txt");

vector<Instruction> instructions = readInstructions("machine\_code.txt");

vector<int> regs(8, 0);

int pc = 0;

Instruction ir;

Memory memory(1024);

while (pc < instructions.size()) {

ir = instructions[pc];

executeInstruction(ir, regs, memory);

pc++;

}

cout << pc;

return 0;

}

* **Steps**:
  1. Calls assemble to convert assembly instructions into machine code.
  2. Reads the generated machine code using readInstructions.
  3. Initializes:
     + **Registers**: 8 registers initialized to 0.
     + **Memory**: Size of 1024.
  4. Executes instructions sequentially using executeInstruction.
  5. Outputs the program counter (pc) after execution.

**7. Example**

**Input (instructions.txt):**

LOAD 1 10

ADD 2 0

STORE 3 0

HALT 0 0

**Output (machine\_code.txt):**

00 1 10

01 2 0

10 3 0

1111 0 0

**Execution Flow:**

1. **LOAD**: Loads value at memory address 10 to register 0.
2. **ADD**: Adds 2 to register 0.
3. **STORE**: Stores result in memory address 3.
4. **HALT**: Stops execution.