Vole machine simulator report

Name	*Id*	*Tasks*
Nagam Mohammed Mostafa	20231188	Collected the code Main UI Part of Alu
Asmaa Sayed Mohammed	20231022	Divided the file into 3 folders bonus
Mohammed abdelrahim	20240820	Cu and CPU and ALU Memory and registers

Object Oriented Programming
CS213

This code simulates a simple Vole machine simulator using C++. It consists of several components that how a CPU works, including memory, registers, an Arithmetic Logic Unit (ALU), and a Control Unit (CU). The CPU can read instructions from a file and execute them.

The classes :

1. Memory Class (MEMORY):
    - Purpose: This class simulates computer memory, which stores data and instructions.
    - Structure: It has an array of 256 memory cells, where each cell can hold a string value (like "00").
    - Methods:
        - Constructor: Initializes each memory cell with a default value of "00".
        - set_memoryvalue: Allows setting a value at a specific address. It checks if the address is valid (0 to 255).
        - get_meoryvalue: Retrieves the value at a specific address, also checking for valid addresses.
        - print: Displays all memory cells and their values.
2. Register Class (REGESTER):
    - Purpose: This class simulates CPU registers, which are small storage locations for quick data access.
    - Structure: It has 16 registers, each initialized to "0".
    - Methods:
        - Constructor: Initializes each register.
        - set_regester: Sets the value of a register at a specific index, checking if the index is valid (0 to 15).
        - get_regvalue: Retrieves the value of a register, again checking if the index is valid.
        - print: Displays all registers and their values.

3. Arithmetic Logic Unit Class (ALU):
    - Purpose: This class performs arithmetic operations, like normal addition and floating-point addition
    - Methods:
        - add: Takes two hexadecimal string inputs, converts them to decimal, adds them, and returns the result as a hexadecimal string.
        - Add the floating point : take to hexadecimal numbers and add them considering the mantissa , exponent and the signs
4. Control Unit Class (CU):
    - Purpose: This class controls the execution of instructions: if the program work step by step or just as whole
    - Structure: It holds instances of MEMORY, REGESTER, and ALU, as well as a program counter (pc) that tracks the current instruction location.
    - Methods:
        - Constructor: Initializes memory, registers, ALU, and sets the program counter to 0.
        - execute: Continuously reads instructions from memory and executes them based on the instruction code.
5. CPU Class:
    - Purpose: This class combines all components into a single CPU unit and allow the execution of all the instructions
    - Methods:
        - read_file: Reads instructions from a specified text file and stores them in memory.
        - store: Helper function that takes a line of instruction, splits it, and saves it in memory.
        - print_memory: Calls the memory's print function to display all memory values.
        - print_regester: Calls the register's print function to display all register values.

    - execute: Starts the instruction execution process by reading instructions from memory and passing them to the Control Unit.

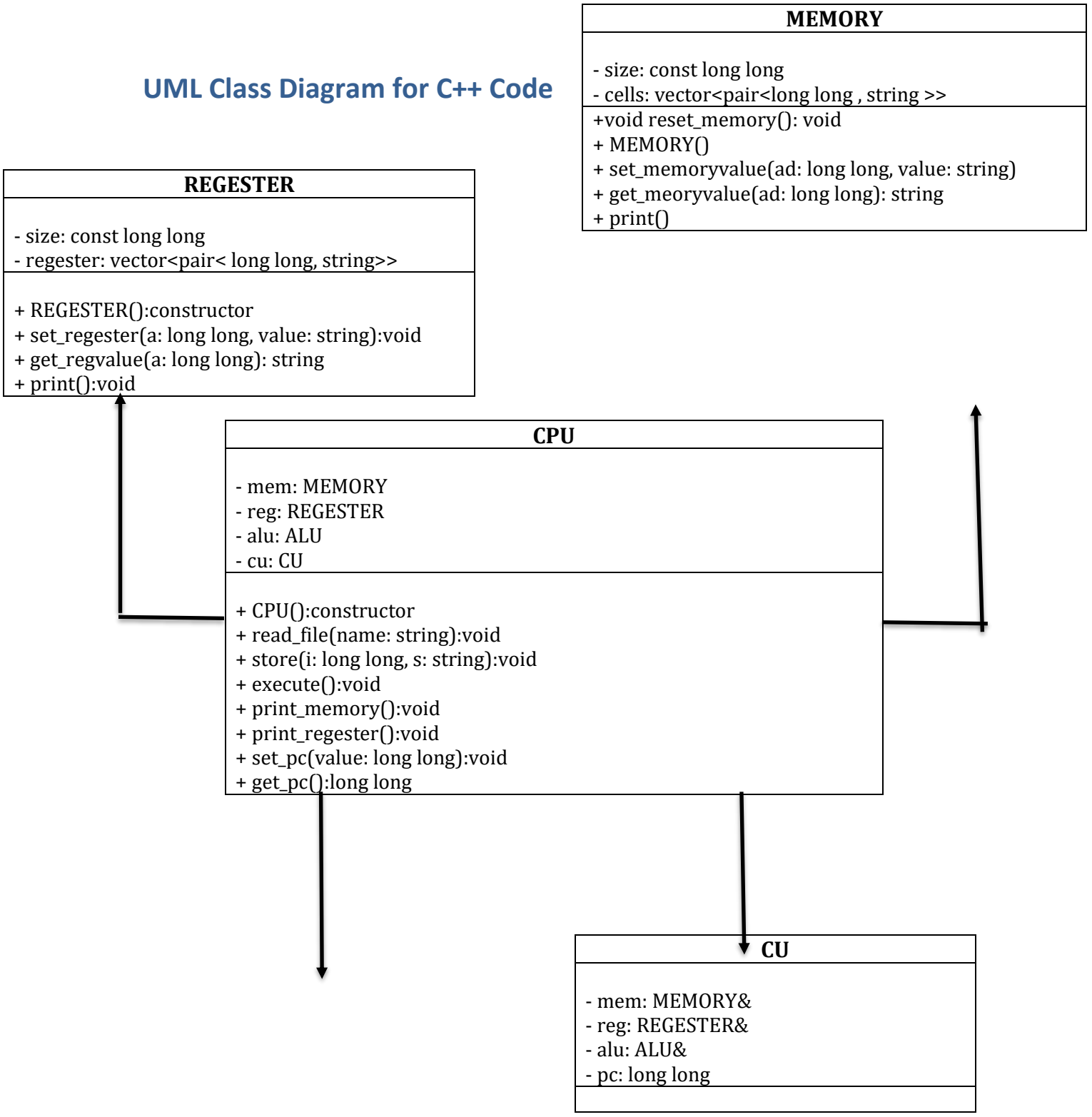
Main Function:

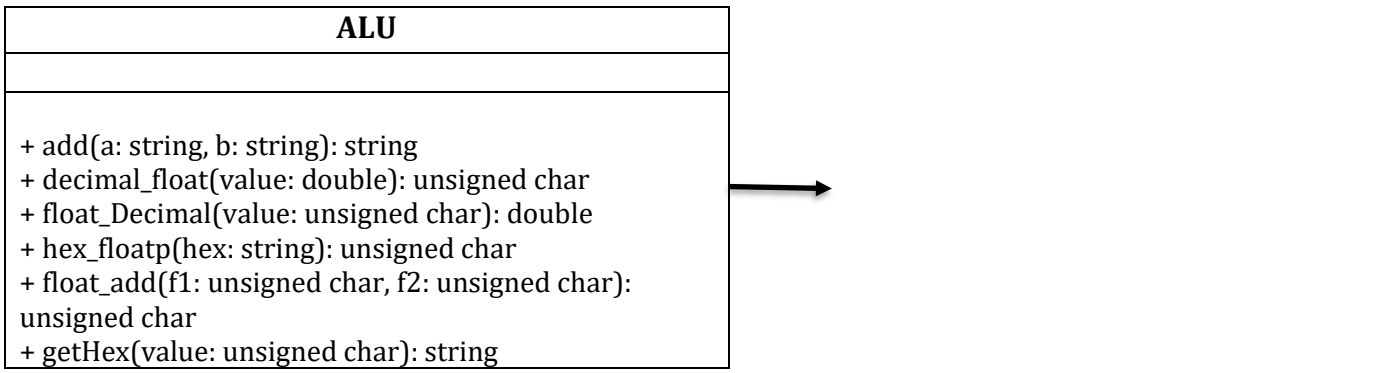
- The main function creates an instance of the CPU, reads a file (which contain the instructions), executes the instructions, and prints the values in the registers.

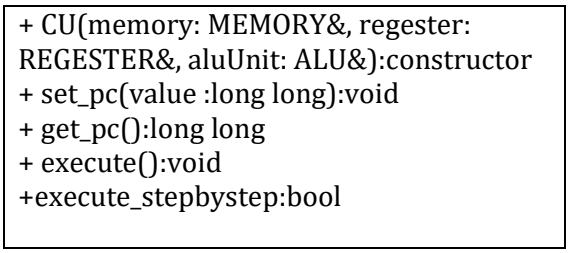
Error Handling:

- The code checks valid addresses when accessing memory and registers, and it also checks the length of instructions to prevent errors or any infinite loops
- Check if the file name is wrong
- If the PC is odd or not

UML Class Diagram for C++ Code




ALU
+ add(a: string, b: string): string
+ decimal_float(value: double): unsigned char
+ float_Decimal(value: unsigned char): double
+ hex_floatp(hex: string): unsigned char
+ float_add(f1: unsigned char, f2: unsigned char): unsigned char
+ getHex(value: unsigned char): string


+ CU(memory: MEMORY&, regester: REGESTER&, aluUnit: ALU&):constructor
+ set_pc(value :long long):void
+ get_pc():long long
+ execute():void
+execute_stepbystep:bool