#include<iostream>

#include<fstream>

using namespace std;

class OS {

private:

char M[100][4]; // Physical Memory

char IR[4]; // Instruction Register (4 bytes)

char R[4]; // General Purpose Register (4 bytes)

int IC; // Instruction Counter Register (2 bytes)

int SI; // Interrupt

bool C; // Toggle (1 byte)

char buffer[40];

public:

void init();

void LOAD();

void Execute();

void MOS();

fstream infile;

fstream outfile;

};

* The code includes necessary libraries (**iostream** for input/output and **fstream** for file handling).
* It declares a class named **OS**.
* The private members of the class include:
  + **M**: a 2D array of characters representing the physical memory.
  + **IR**: an array of characters representing the Instruction Register.
  + **R**: an array of characters representing the General Purpose Register.
  + **IC**: an integer representing the Instruction Counter Register.
  + **SI**: an integer representing an interrupt.
  + **C**: a boolean variable representing a toggle.
  + **buffer**: an array of characters used as a temporary buffer.
* The class has four member functions declared: **init**, **LOAD**, **Execute**, and **MOS**.
* It also declares **infile** and **outfile** as **fstream** objects for reading input from a file and writing output to a file.

void OS::init()

{

// Initialize all elements in M to 0

for (int i = 0; i < 100; i++) {

for (int j = 0; j < 4; j++) {

M[i][j] = 0;

}

}

// Set IR and R arrays to 0

IR[0] = {0};

R[0] = {0};

// Set C to false

C = false;

}

* The **init** function initializes all the private member variables of the **OS** class.
* It sets all elements in **M** (physical memory) to 0.
* It sets the **IR** (Instruction Register) and **R** (General Purpose Register) arrays to 0.
* It sets **C** (toggle) to false.

void OS::MOS()

{

if (SI == 1) {

// Read Mode (GD)

// Read input from the buffer and store it in the physical memory

// based on the values in the Instruction Register (IR)

// ...

}

else if (SI == 2) {

// Write Mode (PD)

// Write the contents of the physical memory to the output file

// based on the values in the Instruction Register (IR)

// ...

}

else if (SI == 3) {

// Terminate

// Perform any necessary actions for termination

// ...

}

}

* The **MOS** function represents the Master Mode of the OS.
* It handles different interrupts based on the value of **SI**.
* If **SI** is 1, it executes the Read Mode (GD) which reads input from the buffer and stores it in the physical memory.
* If **SI** is 2, it executes the Write Mode (PD) which writes the contents of the physical memory to the output file.
* If **SI** is 3, it terminates the execution.

void OS::Execute()

{

while (true) {

// Load the current instruction into the Instruction Register (IR)

for (int i = 0; i < 4; i++) {

IR[i] = M[IC][i];

}

IC++;

// Execute different instructions based on the opcode in IR

if (IR[0] == 'G' && IR[1] == 'D') {

// GD instruction

SI = 1;

MOS();

}

else if (IR[0] == 'P' && IR[1] == 'D') {

// PD instruction

SI = 2;

MOS();

}

else if (IR[0] == 'H') {

// H instruction

SI = 3;

MOS();

break;

}

else if (IR[0] == 'L' && IR[1] == 'R') {

// LR instruction

// Load contents from memory into the General Purpose Register (R)

// ...

}

else if (IR[0] == 'S' && IR[1] == 'R') {

// SR instruction

// Store contents from the General Purpose Register (R) into memory

// ...

}

else if (IR[0] == 'C' && IR[1] == 'R') {

// CR instruction

// Compare contents of memory and General Purpose Register (R)

// Set C (toggle) based on the comparison result

// ...

}

else if (IR[0] == 'B' && IR[1] == 'T') {

// BT instruction

// Branch to a specified instruction based on the value of C (toggle)

// ...

}

}

}

* The **Execute** function represents the execution phase of the OS.
* It executes instructions stored in the physical memory by fetching them into the Instruction Register (IR).
* Inside the while loop, it checks the opcode in IR to determine the type of instruction and performs the corresponding actions.
* If the opcode is 'GD', it sets **SI** to 1 and invokes **MOS** for Read Mode.
* If the opcode is 'PD', it sets **SI** to 2 and invokes **MOS** for Write Mode.
* If the opcode is 'H', it sets **SI** to 3, invokes **MOS** for termination, and breaks the loop.
* The code for the 'LR', 'SR', 'CR', and 'BT' instructions is not provided but should be implemented accordingly.

void OS::LOAD()

{

cout << "Reading Data..." << endl;

int x = 0;

do {

for (int i = 0; i <= 39; i++)

buffer[i] = '\0';

infile.getline(buffer, 40);

// Process different types of input lines

// ...

} while (!infile.eof());

}

* The **LOAD** function reads data from the input file and processes it.
* It enters a do-while loop that continues until the end of the file (**eof**).
* Inside the loop, it reads each line of input into the **buffer** array.
* The function should contain additional code to handle different types of input lines, such as identifying the start and end markers and storing instructions in the physical memory.

int main()

{

OS os;

os.infile.open("input.txt", ios::in);

os.outfile.open("output.txt",