OPERATING SYSTEM PHASE-1

Made by Group 34

Year: TY Div: B Batch: B2

|  |  |  |
| --- | --- | --- |
| PRN | Name | Roll no. |
| 12011318 | Swapnil Mhoprekar | 67 |
| 12011324 | Tanish Modase | 68 |
| 12010850 | Gautam Mudawadkar | 71 |
| 12011148 | Prathmesh Nagpure | 75 |

CODE

#include <iostream>

#include <fstream>

using namespace std;

class OS

{

private:

    char M[100][4];  // Memory of 10 blocks with 10 words each...1 word = 4 bytes

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

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

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

    int SI;          // Sytem Interrupt

    bool C;          // Toggle or flag (1 byte)

    char buffer[40]; // Buffer of 1 block size i.e. 40 bytes

public:

    void init();

    void LOAD();

    void EXECUTE();

    void MOS();

    fstream infile;

    fstream outfile;

};

//  Initializes the CPU and memory

void OS::init()

{

    std::cout << "Initializing and setting up..." << endl;

    for (int i = 0; i < 100; i++)

        for (int j = 0; j < 4; j++)

            M[i][j] = ' ';

    IR[0] = {' '};

    R[0] = {' '};

    C = false;

}

// Master Mode

void OS::MOS()

{

    // Read Mode

    if (SI == 1)

    {

        for (int i = 0; i < 40; i++)

            buffer[i] = '\0'; // Sets all buffer bytes to NULL

        infile.getline(buffer, 40); // Reads first 40 characters input file line and stores in buffer

        // Sets location var i to starting index of memory block based on IR

        int i = IR[2] - 48;

        i = i \* 10;

        int k = 0;

        // Transfers buffer content to memory block

        for (int l = 0; l < 10; l++)

        {

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

            {

                M[i][j] = buffer[k];

                k++;

            }

            i++;

        }

    }

    // Write Mode

    else if (SI == 2)

    {

        for (int i = 0; i <= 39; i++)

            buffer[i] = '\0';

        int i = IR[2] - 48;

        i = i \* 10;

        int k = 0;

        // Fetching data from memory block to buffer and writing from buffer to output file

        for (int l = 0; l < 10; l++)

        {

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

            {

                buffer[k] = M[i][j];

                outfile << buffer[k];

                k++;

            }

            i++;

        }

        outfile << '\n';

    }

    // Terminate

    else if (SI == 3)

    {

        outfile << "\n";

        outfile << "\n";

    }

}

// Loading all inputs from input file

void OS::LOAD()

{

    cout << "Reading data..." << endl;

    // Initializing memory

    for (int i = 0; i < 100; i++)

        for (int j = 0; j < 4; j++)

            M[i][j] = ' ';

    do

    {

        // Initializing buffer

        for (int i = 0; i <= 39; i++)

            buffer[i] = '\0';

        infile.getline(buffer, 40);

        // Control card cases

        if (buffer[0] == '$' && buffer[1] == 'A' && buffer[2] == 'M' && buffer[3] == 'J')

        {

            init();

        }

        else if (buffer[0] == '$' && buffer[1] == 'D' && buffer[2] == 'T' && buffer[3] == 'A')

        {

            IC = 0;

            EXECUTE();

        }

        else if (buffer[0] == '$' && buffer[1] == 'E' && buffer[2] == 'N' && buffer[3] == 'D')

        {

            continue;

        }

        // Program card case

        else

        {

            int k = 0;

            for (int x = 0; x < 100; x++)

            {

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

                {

                    M[x][j] = buffer[k];

                    k++;

                }

                if (k == 40 || buffer[k] == ' ' || buffer[k] == '\n')

                {

                    break;

                }

            }

        }

    } while (!infile.eof());

}

// Execution of program

void OS::EXECUTE()

{

    std::cout << "Executing programs..." << endl;

    while (true)

    {

        for (int i = 0; i < 4; i++) // Load instruction in IR and increement the pointer/IC to next instruction

            IR[i] = M[IC][i];

        IC++;

        // Examining the first two bytes of IR for opcode and defining the functions for those opcodes

        if (IR[0] == 'L' && IR[1] == 'R') // LR

        {

            int loc = (IR[2] - 48) \* 10 + (IR[3] - 48);

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

                R[i] = M[loc][i];

        }

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

        {

            int loc = (IR[2] - 48) \* 10 + (IR[3] - 48);

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

                M[loc][i] = R[i];

        }

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

        {

            int loc = (IR[2] - 48) \* 10 + (IR[3] - 48);

            bool flag = true; // Equal flag

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

                if (M[loc][i] != R[i])

                    flag = false;

            C = flag;

        }

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

        {

            if (C == true)

            {

                int loc = (IR[2] - 48) \* 10 + (IR[3] - 48);

                IC = loc;

            }

        }

        else if (IR[0] == 'G' && IR[1] == 'D') // GD

        {

            SI = 1;

            MOS();

        }

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

        {

            SI = 2;

            MOS();

        }

        else if (IR[0] == 'H') // H

        {

            SI = 3;

            MOS();

            break;

        }

    }

}

int main()

{

    OS os;

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

    os.outfile.open("output.txt", ios::out);

    if (!os.infile)

    {

        std::cout << "ERROR!!! Input File doesn't exist" << endl;

        return 0;

    }

    else if (!os.outfile)

    {

        std::cout << "ERROR!!! Output File doesn't exist" << endl;

        return 0;

    }

    os.LOAD();

    return 0;

}

INPUT

$AMJ000100030001

GD10PD10H

$DTA

Hello World!

$END0001

$AMJ000200030003

GD10PD10GD20PD20GD30GD40LR10CR20BT11PD40HPD30H

$DTA

VIT

VIIT

ARE THE SAME

ARE NOT THE SAME

$END0002

$AMJ0003000120005

GD10LR10SR10PD10SR10SR11PD10SR10SR11SR12PD10SR10SR11PD10SR10PD10H

$DTA

\*

$END0003

$AMJ0004000600002

GD10PD10GD20GD30LR13CR10BT09PD20HLR12CR11BT14PD20HPD30H

$DTA

1 0 0 1

IS NOT A PALINDROME

IS A PALINDROME

$END0004

$AMJ000500030003

GD10PD10GD20PD20GD30GD40LR10CR20BT11PD40HPD30H

$DTA

10

100

ARE EQUAL

ARE NOT EQUAL

$END0005

$AMJ0006000120003

GD10PD10GD20GD30GD40LR12CR20BT12CR21BT12PD40HPD30H

$DTA

2 5 0

0 5

IS DIVISIBLE BY 5

IS NOT DIVISIBLE BY 5

$END0006

$AMJ0007000180003

GD10PD10GD20GD30GD40LR12CR20BT18CR21BT18CR22BT18CR23BT18CR24BT18PD40HPD30H

$DTA

1 3 7

0 2 4 6 8

IS DIVISIBLE BY 2

IS NOT DIVISIBLE BY 2

$END0007

OUTPUT

Hello World!

CAT

BAT

ARE NOT SAME

\*

\* \*

\* \* \*

\* \*

\*

1 0 0 1

IS A PALINDROME

10

100

ARE NOT EQUAL

250

IS DIVISIBLE BY 5

137

IS NOT DIVISIBLE BY 2