**VISHWAKARMA INSTITUTE OF TECHNOLOGY**

COMPUTER ENGINEERING

**Name: Durgesh Dhurve**

**Division: A**

**Roll Number: 71**

Subject: Operating System (OS) – Course Project

MULTIPROGRAMMING OPERATING SYSTEM (MOS) PROJECT

First Version

ASSUMPTIONS:

* Jobs entered without error in input file
* No physical separation between jobs
* Job outputs separated in output file by 2 blank lines
* Program loaded in memory starting at location 00
* No multiprogramming, load and run one program at a time
* SI interrupt for service request

NOTATION

M: memory; IR: Instruction Register (4 bytes) IR [1, 2]: Bytes 1, 2 of IR/Operation Code

IR [3, 4]: Bytes 3, 4 of IR/Operand Address M[&]: Content of memory location &

IC: Instruction Counter Register (2 bytes) R: General Purpose Register (4 bytes)

C: Toggle (1 byte)

: Loaded/stored/placed into

MOS (MASTER MODE)

SI = 3 (Initialization) Case SI of

1: Read

2: Write

3: Terminate Endcase

READ

IR [4]  0

Read next (data) card from input file in memory locations IR [3,4] through IR [3,4] +9 If M [IR [3,4]] = $END, abort (out-of-data) EXECUTEUSERPROGRAM

WRITE

IR [4]  0

Write one block (10 words of memory) from memory locations IR [3,4] through IR [3,4] + 9 to output file

EXECUTEUSERPROGRAM

TERMINATE

Write 2 blank lines in output file MOS/LOAD

LOAD

m  0

While not e-o-f

Read next (program or control) card from input file in a buffer Control card: $AMJ, end-while

$DTA, MOS/STARTEXECUTION

$END, end-while

Program Card: If m = 100, abort (memory exceeded)

Store buffer in memory locations m through m + 9 m  m + 10

End-While STOP

MOS/STARTEXECUTION IC  00

EXECUTEUSERPROGRAM

EXECUTEUSERPROGRAM (SLAVE MODE)

Loop

IR  M [IC] IC  IC+1

Examine IR[1,2]

LR: R  M [IR[3,4]]

SR: R  M [IR[3,4]]

CR: Compare R and M [IR[3,4]] If equal C  T else C  F

BT: If C = T then IC  IR [3,4] GD: SI = 1

PD: SI = 2

H: SI = 3

End-Examine End-Loop

**Program Code -**

#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;

};

//initiallise everything to 0

void OS::init()

{

    for(int i = 0; i<100; i++)

    {

        for(int j = 0; j<4; j++ )

        {

            M[i][j]=0;

        }

    }

    IR[0] = {0};

    R[0] = {0};

    C = false;

}

//Master Mode

void OS::MOS()

{

    if(SI == 1)     //Read Mode

    {

        for(int i=0;i<=39;i++)

           buffer[i]='\0';

        infile.getline(buffer,40);

        int k = 0;

        int i = IR[2]-48;

        i = i\*10;

        for( int l=0 ; l<10 ;  ++l)

        {

            for(int j = 0 ; j<4; ++j)

            {

                M[i][j] = buffer[k];

                k++;

            }

            if(k == 40)

            {

                break;

            }

            i++;

        }

        for(int i = 0; i<100; i++)

        {

            cout<<"M["<<i<<"]\t";

            for(int j = 0; j<4; j++ )

            {

                cout<<M[i][j];

            }

            cout<<endl;

        }

    }

    else if(SI == 2)        //Write Mode

    {

      for(int i=0;i<=39;i++)

           buffer[i]='\0';

        int k = 0;

        int i = IR[2]-48;

        i = i\*10;

        for( int l=0 ; l<10 ;  ++l)

        {

            for(int j = 0 ; j<4; ++j)

            {

                buffer[k]=M[i][j];

                outfile<<buffer[k];

                k++;

            }

            if(k == 40)

            {

                break;

            }

            i++;

        }

           for(int i = 0; i<100; i++)

        {

            cout<<"M["<<i<<"]\t";

            for(int j = 0; j<4; j++ )

            {

                cout<<M[i][j];

            }

            cout<<endl;

        }

        outfile<<"\n";

    }

    else if(SI == 3)        //Terminate

    {

        outfile<<"\n";

        outfile<<"\n";

    }

}

//Execution

void OS::Execute()

{

    while(true)

    {

        for(int i = 0; i<4; i++)        //Load in register

        {

            IR[i] = M[IC][i];

        }

        IC++;

        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;

        }

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

        {

            int i = IR[2]-48;

            i = i\*10 + ( IR[3]-48);

            for(int j=0;j<=3;j++)

                R[j]=M[i][j];

            //for(int j=0;j<=3;j++)

              // cout<<R[j];

            cout<<endl;

        }

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

        {

            int i = IR[2]-48;

            i = i\*10 +( IR[3]-48) ;

            //cout<<i;

            for(int j=0;j<=3;j++)

                M[i][j]=R[j];

            cout<<endl;

        }

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

        {

            int i = IR[2]-48;

            i = i\*10 + (IR[3] - 48);

            //cout<<i;

            int count=0;

            for(int j=0;j<=3;j++)

                if(M[i][j] == R[j])

                    count++;

            if(count==4)

                C=true;

            //cout<<C;

        }

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

        {

            if(C == true)

            {

                int i = IR[2]-48;

                i = i\*10 + (IR[3] - 48);

                IC = i;

            }

        }

    }

}

//Load Function

void OS::LOAD()

{

    cout<<"Reading Data..."<<endl;

    int x=0;

    do

    {

        for(int i=0;i<=39;i++)      //clear buffer

           buffer[i]='\0';

        infile.getline(buffer,40);

        for(int k=0;k<=39;k++)

           cout<<buffer[k];

        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 = 00;

            Execute();

        }

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

        {

            x=0;

            continue;

        }

        else

        {

            int k = 0;

            for(; 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());         //continues to take input till eof

}

int main()

{

    OS os;

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

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

    if(!os.infile)

    {

        cout<<"Failure"<<endl;

    }

    else

    {

        cout<<"File Exist"<<endl;

    }

    os.LOAD();

    return 0;

}

**Input -**

$AMJ000100030001

GD10PD10H

$DTA

Hello World!

$END0001

$AMJ0003000120004

GD20LR20SR33LR21SR32LR22SR31

LR23SR30PD30H

$DTA

a\_\_\_b\_\_\_c\_\_\_d\_\_\_

$END0002

$AMJ0002000120004

GD20GD30GD40GD50PD20PD30LR20CR30BT11

PD40PD50H

$DTA

VIT

VIIT

NOT

SAME

$END0002

$AMJ0005000170003

GD40GD20GD30PD40LR40CR43BT09PD20PD30

SR40LR41CR42BT14PD20PD30H

$DTA

1\_\_\_0\_\_\_0\_\_\_1\_\_\_

NOT

PALINDROME

$END0005

$AMJ0003000120004

GD20LR20SR35PD30SR43SR47PD40SR51SR55

SR59PD50H

$DTA

\*

$END0002

$AMJ0003000120004

GD20PD20LR20SR21PD20SR22PD20SR23PD20

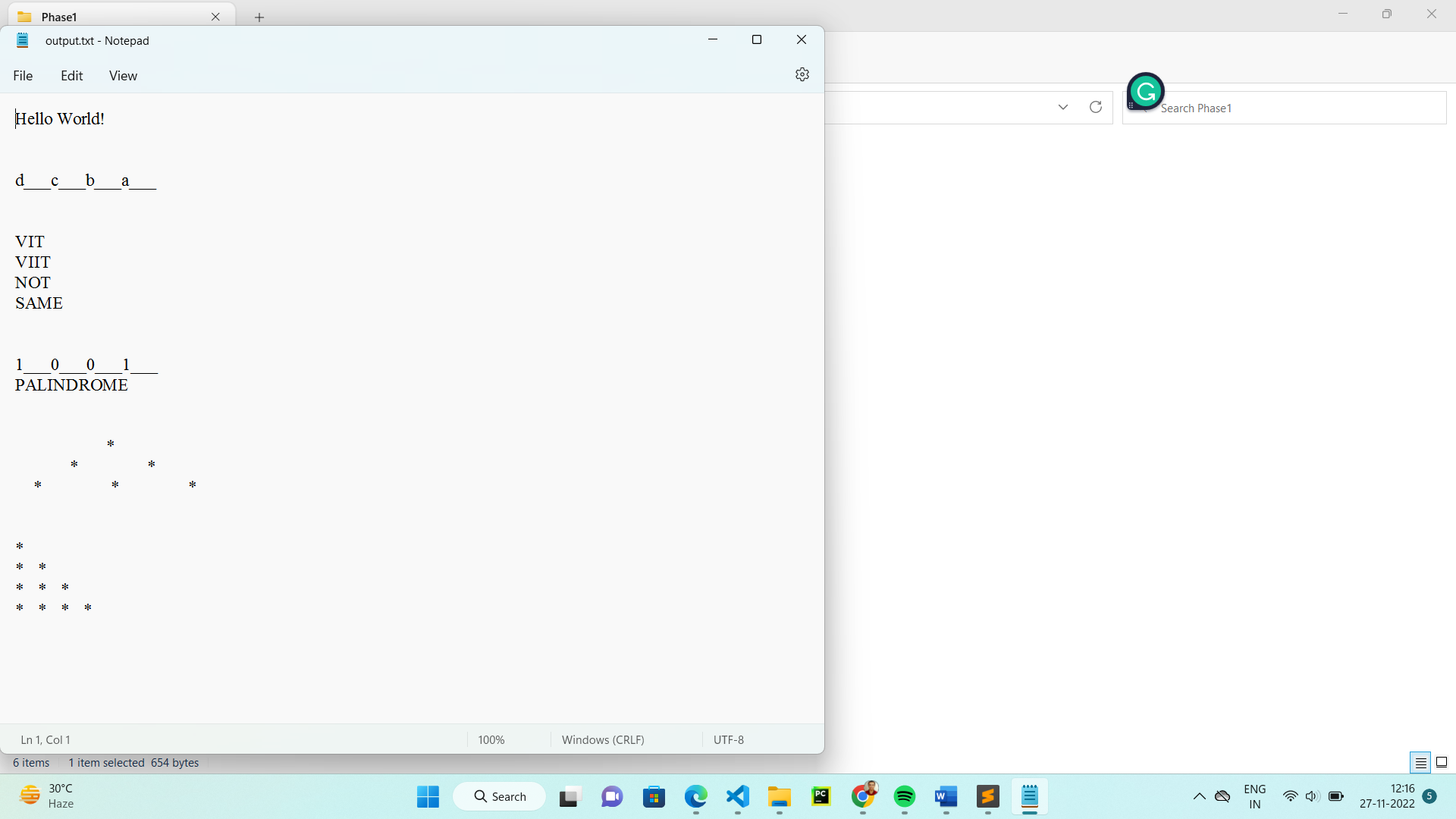
H

$DTA

\*

$END0002

**Output –**



**Terminal –**

