OS PHASE 2:

Group No: SY33

GROUP MEMEBRS:

53. Mayuresh Kaulwar

54. Yash Kawtikwar

57. Samyak Khadse

58. Anurag Khandare

CODE:

#include <bits/stdc++.h>

using namespace std;

ifstream fin;

ofstream fout;

int SI, PI, TI;

bool occupied\_pages[30];

void LOAD();

class memory

{

public:

    char mem[300][4];

    char ch;

    int page\_table\_ptr;

    void reset() // ok

    {

        // reset the memory by replacing every symbol in 2D array by $

        memset(mem, '$', sizeof(char) \* 300 \* 4);

        memset(occupied\_pages, false, sizeof(bool) \* 30);

        page\_table\_ptr = rand() % 30;

        occupied\_pages[page\_table\_ptr] = true;

        page\_table\_ptr \*= 10;

        SI = PI = TI = 0;

    }

    string get\_mem(int pos) // get word form memory

    {

        // get the memory contents of given position

        string temp = "";

        for (int i = 0; i < 4; i++)

            temp += mem[pos][i];

        return temp;

    }

    void set\_mem(string s, int pos) // store word in mem

    {

        // set the memory for the recieved value at postion

        for (int i = 0; i < 4; i++)

            mem[pos][i] = s[i];

    }

    int get\_page\_table\_ptr()

    {

        return page\_table\_ptr;

    }

    int allocate\_page()

    {

        int page\_no = rand() % 30;

        while (occupied\_pages[page\_no] == true)

            page\_no = rand() % 30;

        occupied\_pages[page\_no] = true;

        return page\_no;

    }

    void set\_page\_table(int row\_num, int page\_no)

    {

        ostringstream temp;

        temp << page\_no;

        string table\_entry;

        if (page\_no < 10)

            table\_entry = "$10" + temp.str();

        else

            table\_entry = "$1" + temp.str();

        set\_mem(table\_entry, page\_table\_ptr + row\_num);

    }

    void store\_card(string s, int mem\_cnt)

    {

        // extract the words and call the setmem function

        string word = "";

        int page\_no = allocate\_page();

        set\_page\_table(mem\_cnt, page\_no);

        page\_no \*= 10;

        for (int i = 0; i < s.length(); i += 4)

        {

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

            {

                word += s[i + j];

            }

            set\_mem(word, page\_no);

            page\_no++;

            word = "";

        }

    }

    void print\_mem()

    {

        int flag = 0;

        for (int i = 0; i < 300; i++)

        {

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

            {

                cout << mem[i][j];

            }

            if (flag)

                break;

            cout << endl;

        }

    }

} m\_obj;

class cpu

{

public:

    int fetched\_IC, terminate\_code, TLL, TTL, LLC, TLC, pos;

    bool terminate, fetched\_C;

    string fetched\_IR, operand, opreator, fetched\_R, compare\_string, p\_id;

    char IR[4], R[4], IC[2];

    bool C, run\_mos;

    // set and reset function of all the register

    int s\_to\_i(string operand) // ok

    {

        // return the integer no for the given string

        if (operand[0] >= '0' && operand[0] <= '9' && operand[1] >= '0' && operand[1] <= '9')

            return ((int)operand[0] - 48) \* 10 + ((int)operand[1] - 48);

        return -1;

    }

    void set\_limits(string limits)

    {

        p\_id = limits.substr(0, 4);

        TTL = s\_to\_i(limits.substr(4, 2)) \* 100 + s\_to\_i(limits.substr(6, 2));

        TLL = s\_to\_i(limits.substr(8, 2)) \* 100 + s\_to\_i(limits.substr(10, 2));

        LLC = 0;

        TLC = 0;

        terminate\_code = 0;

    }

    void set\_IC()

    {

        IC[0] = '0';

        IC[1] = '0';

    }

    void set\_IC(int pos) // ok

    {

        IC[1] = ((char)pos % 10) + 48;

        pos = pos / 10;

        IC[0] = ((char)pos % 10) + 48;

    }

    int get\_IC()

    {

        int val;

        val = ((int)IC[0] - 48) \* 10 + ((int)IC[1] - 48);

        cout << "Value of IC: " << val << endl;

        return val;

    }

    void inc\_IC() // ok

    {

        int val;

        val = get\_IC();

        val++;

        set\_IC(val);

    }

    void set\_IR(int IC) // ok

    {

        string returned\_value = "";

        returned\_value = m\_obj.get\_mem(IC);

        for (int i = 0; i < 4; i++)

            IR[i] = returned\_value[i];

    }

    string get\_IR() // ok

    {

        string ret\_IR = "";

        for (int i = 0; i < 4; i++)

            ret\_IR += IR[i];

        cout << "Value of IR: " << ret\_IR << endl;

        ;

        return ret\_IR;

    }

    void set\_R(int pos) // ok

    {

        string returned\_value = "";

        returned\_value = m\_obj.get\_mem(pos);

        for (int i = 0; i < 4; i++)

            R[i] = returned\_value[i];

    }

    string get\_R() // ok

    {

        string ret\_R = "";

        for (int i = 0; i < 4; i++)

        {

            ret\_R += R[i];

        }

        return ret\_R;

    }

    void set\_C(bool value) // ok

    {

        C = value;

    }

    bool get\_C() // ok

    {

        return C;

    }

    int address\_tranlation(int virtual\_add)

    {

        int page = m\_obj.get\_page\_table\_ptr() + (virtual\_add / 10);

        string value\_page = m\_obj.get\_mem(page);

        if (value\_page[1] == '$')

        {

            PI = 3;

            return -1;

        }

        value\_page = value\_page.substr(2, 2);

        return (s\_to\_i(value\_page) \* 10 + (virtual\_add % 10));

    }

    int address\_tranlation(string op)

    {

        if (s\_to\_i(op) == -1)

        {

            PI = 2;

            return -2;

        }

        else

            address\_tranlation(s\_to\_i(op));

    }

    void startexe()

    {

        // fetch decode execute cycle given below

        set\_IC(0);

        terminate = false;

        while (!terminate)

        {

            // fetch ic

            // fetch ir form the location specified by ic

            // get the value of operator and operand

            run\_mos = false;

            fetched\_IC = address\_tranlation(get\_IC());

            inc\_IC();

            set\_IR(fetched\_IC);

            fetched\_IR = get\_IR();

            if ((fetched\_IR.compare("H")) == 3)

                fetched\_IR = "Hrrr";

            opreator = fetched\_IR.substr(0, 2);

            operand = fetched\_IR.substr(2, 2);

            pos = address\_tranlation(s\_to\_i(operand));

            if (address\_tranlation(operand) != -2 || !(operand.compare("rr")))

            {

                if (!(opreator.compare("LR"))) // ok

                {

                    // set the contents of the register R from the given location of memory specified in the operand

                    // cout<<"LR";

                    if (pos == -1)

                    {

                        run\_mos = true;

                    }

                    else

                    {

                        set\_R(pos);

                        TLC++;

                    }

                }

                else if (!(opreator.compare("SR"))) // ok

                {

                    // get the contents of register R

                    // store them at the location of memory specified in the operand

                    // cout<<"SR";

                    // TLC++;

                    fetched\_R = get\_R();

                    if (pos == -1)

                    {

                        run\_mos = true;

                    }

                    else

                    {

                        m\_obj.set\_mem(fetched\_R, pos);

                        TLC++;

                    }

                }

                else if (!(opreator.compare("CR"))) // ok

                {

                    // get the contents of register r

                    // compare with given memory location

                    // if the values of above two matches then set toggle register to true

                    // else set the toggle register to false

                    // cout<<"CR";

                    fetched\_R = get\_R();

                    if (pos == -1)

                    {

                        run\_mos = true;

                    }

                    else

                    {

                        TLC++;

                        compare\_string = m\_obj.get\_mem(pos);

                        if (fetched\_R.compare(compare\_string) == 0)

                            set\_C(true);

                        else

                            set\_C(false);

                    }

                }

                else if (!(opreator.compare("BT"))) // ok

                {

                    // if the value of toggle register is true

                    // we change the value of ic

                    // cout<<"BT";

                    fetched\_C = get\_C();

                    if (fetched\_C)

                    {

                        // int get\_physical\_add=address\_tranlation();

                        set\_IC(s\_to\_i(operand));

                    }

                    TLC++;

                }

                else if (!(opreator.compare("GD"))) // ok

                {

                    if (pos != -1)

                    {

                        TLC++;

                        SI = 1;

                    }

                    run\_mos = true;

                }

                else if (!(opreator.compare("PD"))) // ok

                {

                    if (pos != -1)

                    {

                        TLC++;

                        SI = 2;

                    }

                    run\_mos = true;

                }

                else if (!(opreator.compare("Hr"))) // ok

                {

                    TLC++;

                    fetched\_IR = "H";

                    SI = 3;

                    run\_mos = true;

                }

                else

                {

                    PI = 1;

                    run\_mos = true;

                }

            }

            else

            {

                run\_mos = true;

            }

            if (TLC > TTL)

            {

                TI = 2;

                run\_mos = true;

                terminate = true;

                terminate\_code = 3;

            }

            if (run\_mos)

                MOS();

        }

    }

    void MOS()

    {

        if (TI == 0 && SI == 1)

        {

            // get the instrution of the file in terms of 4 words

            // store it one by one into memory

            // at start store the program card from memory location with unit place 0

            string s;

            pos = (pos / 10) \* 10;

            getline(fin, s);

            if (s.find("$END") != -1)

            {

                terminate = true;

                terminate\_code = 1;

            }

            else

            {

                if (!s.empty() && s[s.size() - 1] == '\r')

                    s.erase(s.size() - 1);

                int len = s.length(), start = 0, i;

                string s1;

                for (i = pos; start < len; i++)

                {

                    if ((len - start) < 4)

                        s1 = s.substr(start, (len - start));

                    else

                        s1 = s.substr(start, 4);

                    start += 4;

                    m\_obj.set\_mem(s1, i);

                }

            }

            SI = 0;

        }

        else if (TI == 2 && SI == 1)

        {

            terminate = true;

            terminate\_code = 3;

        }

        else if ((TI == 0 || TI == 2) && SI == 2)

        {

            LLC++;

            if (LLC > TLL)

            {

                terminate = true;

                terminate\_code = 2;

            }

            // put the data from memory into the file specified

            else

            {

                int pos = address\_tranlation(s\_to\_i(operand)), flag = 0;

                pos = (pos / 10) \* 10;

                string ans = "", temp = "";

                for (int i = pos; i < pos + 10; i++)

                {

                    temp = m\_obj.get\_mem(i);

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

                    {

                        if (temp[j] == '\0' || temp[j] == '$')

                        {

                            break;

                            flag = 1;

                        }

                        ans += temp[j];

                    }

                    if (flag)

                        break;

                }

                fout << ans << endl;

                if (TI == 2)

                {

                    terminate = true;

                    terminate\_code = 3;

                }

            }

            SI = 0;

        }

        else if ((TI == 0 || TI == 2) && SI == 3)

        {

            // make terminate true to end the program

            // as we encountered the halt instruction

            terminate = true;

            terminate\_code = 0;

        }

        else if (TI == 0 && PI == 1)

        {

            terminate\_code = 4;

            terminate = true;

        }

        else if (TI == 0 && PI == 2)

        {

            terminate\_code = 5;

            terminate = true;

        }

        else if (TI == 0 && PI == 3)

        {

            if (!(opreator.compare("GD")) || !(opreator.compare("SR")))

            {

                int page\_no = m\_obj.allocate\_page();

                m\_obj.set\_page\_table((s\_to\_i(operand)) / 10, page\_no);

                set\_IC(get\_IC() - 1);

            }

            else

            {

                terminate\_code = 6;

                terminate = true;

            }

        }

        else if (TI == 2 && PI == 1)

        {

            terminate\_code = 7;

            terminate = true;

        }

        else if (TI == 2 && PI == 2)

        {

            terminate\_code = 8;

            terminate = true;

        }

        else if (TI == 2 && PI == 3)

        {

            terminate\_code = 3;

            terminate = true;

        }

        if (terminate)

        {

            fout << p\_id << " ";

            switch (terminate\_code)

            {

            case 0:

                fout << "NO ERROR\n";

                break;

            case 1:

                fout << "OUT OF DATA\n";

                TLC--;

                break;

            case 2:

                fout << "LINE LIMIT EXCEEDED\n";

                break;

            case 3:

                fout << "TIME LIMIT EXCEEDED\n";

                TLC = TTL;

                break;

            case 4:

                fout << "OPERATION CODE ERROR\n";

                break;

            case 5:

                fout << "OPERAND ERROR\n";

                break;

            case 6:

                fout << "INVALID PAGE FAULT\n";

                break;

            case 7:

                fout << "TIME LIMIT EXCEEDED with OPERATION CODE ERROR\n";

                break;

            case 8:

                fout << "TIME LIMIT EXCEEDED with OPERAND ERROR\n";

                break;

            }

            fout << "job ID " << get\_IC() << "\nIR " << fetched\_IR << "\nTLC " << TLC << "\n LLC " << LLC << "\nValue of IC: " << get\_IC() << "\nValue of IR: " << get\_IR() << endl;

            fout << endl

                 << endl;

            m\_obj.print\_mem();

            cout << "\n";

        }

    }

} exe;

int main()

{

    // open two files one input and one output

    // create memory and cpu object

    fin.open("input2.txt");

    fout.open("Output2.txt");

    LOAD();

    fin.close();

    fout.close();

    return 0;

}

void LOAD()

{

    string s, s1;

    int mem\_cnt = 0;

    while (!(fin.eof())) // ok

    {

        // get the line one by one and check whether it contains as follows

        getline(fin, s);

        if (s.empty())

            break;

        if (s.find("$AMJ") != -1)

        {

            // reset the memory

            m\_obj.reset();

            mem\_cnt = 0;

            exe.set\_limits(s.substr(4, 12));

            continue;

        }

        else if (s.find("$DTA") != -1)

        {

            exe.startexe();

        }

        else if (s.find("$END") != -1)

        {

            // proceed to the next job

            mem\_cnt = 0;

            continue;

        }

        else

        {

            // load the program card as soon as it comes

            m\_obj.store\_card(s, mem\_cnt++);

        }

    }

}