**Input:**

NAN START 200 NAN

NAN MOVER AREG ='5'

NAN MOVEM AREG A

LOOP MOVER AREG A

NAN MOVER CREG B

NAN ADD CREG ='1'

NAN MOVER AREG A

NAN MOVER CREG B

NAN MOVER AREG A

NAN MOVER CREG B

NAN MOVER AREG A

NAN BC ANY NEXT

NAN LTORG NAN NAN

NAN MOVER AREG A

NEXT SUB AREG ='1'

NAN BC LT BACK

LAST STOP NAN NAN

NAN ORIGIN LOOP+2 NAN

NAN MULT CREG B

NAN ORIGIN LAST+1 NAN

A DS 1 NAN

BACK EQU LOOP NAN

B DS 1 NAN

NAN END NAN NAN

// Two-Pass Assembler: Pass-1 Code

**Code:**

#include <bits/stdc++.h>

using namespace std;

// To store mnemonics of the opcodes

struct OPtab

{

    string opcode;

    string mclass;

    string mnemonic;

};

// Hard-coding the class and mnemonic for respective opcode

struct OPtab optab[18] = {

        {"STOP",   "IS", "00"},

        {"ADD",    "IS", "01"},

        {"SUB",    "IS", "02"},

        {"MULT",   "IS", "03"},

        {"MOVER",  "IS", "04"},

        {"MOVEM",  "IS", "05"},

        {"COMP",   "IS", "06"},

        {"BC",     "IS", "07"},

        {"DIV",    "IS", "08"},

        {"READ",   "IS", "09"},

        {"PRINT",  "IS", "10"},

        {"START",  "AD", "01"},

        {"END",    "AD", "02"},

        {"ORIGIN", "AD", "03"},

        {"EQU",    "AD", "04"},

        {"LTORG",  "AD", "05"},

        {"DC",     "DL", "01"},

        {"DS",     "DL", "02"}

};

// Function to fetch the opcode entry

int getOP(string s)

{

    for(int i = 0; i < 18; ++i)

    {

        if(optab[i].opcode == s)

            return i;

    }

    return -1;

}

// Function to fetch the register code

int getRegID(string s)

{

    if(s == "AREG")

        return 1;

    else if(s == "BREG")

        return 2;

    else if(s == "CREG")

            return 3;

    else if(s == "DREG")

            return 4;

    else

        return (-1);

}

// Function to fetch conditional code

int getConditionCode(string s)

{

    if(s == "LT")

        return 1;

    else if(s == "LE")

        return 2;

    else if(s == "EQ")

            return 3;

    else if(s == "GT")

            return 4;

    else if(s == "GE")

            return 5;

    else if(s == "ANY")

            return 6;

    else

        return (-1);

}

// To store Symbol Table output

struct symTable

{

    int no;

    string sname;

    string addr;

};

struct symTable ST[10];

// Function to check presence of a particular 'symbol'

bool presentST(string s)

{

    for(int i = 0; i < 10; ++i)

    {

        if(ST[i].sname == s)

            return true;

    }

    return false;

}

// Function to fetch the symbol entry

int getSymID(string s)

{

    for(int i = 0; i < 10; ++i)

    {

        if(ST[i].sname == s)

            return i;

    }

    return (-1);

}

// To store Literal Table output

struct litTable

{

    int no;

    string lname;

    string addr;

};

struct litTable LT[10];

// Function to check presence of a particular 'literal'

bool presentLT(string s)

{

    for(int i = 0; i < 10; ++i)

    {

        if(LT[i].lname == s)

            return true;

    }

    return false;

}

// Function to fetch the literal entry

int getLitID(string s)

{

    for(int i = 0; i < 10; ++i)

    {

        if(LT[i].lname == s)

            return i;

    }

    return (-1);

}

// To store Pool Table output

struct poolTable

{

    int no;

    string lno;

};

struct poolTable PT[10];

int main()

{

    ifstream fin;

    // input assembly code file

    // empty space (eg. no operand2 / no label) is denoted by "NAN"

    fin.open("input.txt");

    ofstream ic, st, lt;

    // Saving the output of pass1 into pass2 source code directory. Since it will be the input for pass2.cpp

    // The paths may change accordingly

    const char \*path1 = "D:\\EclipseProjects\\LP1\_Pass2\\ic.txt"; // empty space is filled with "NAN"

    const char \*path2 = "D:\\EclipseProjects\\LP1\_Pass2\\symtable.txt";

    const char \*path3 = "D:\\EclipseProjects\\LP1\_Pass2\\littable.txt";

    ic.open(path1); st.open(path2); lt.open(path3);

    string label, opcode, op1, op2;

    int scnt = 0, lcnt = 0, nlcnt = 0, pcnt = 0, LC = 0;

    cout << "\n -- ASSEMBLER PASS-1 OUTPUT --" << endl;

    cout << "\n <LABEL\tOPCODE\tOP1\tOP2\tLC\tINTERMEDIATE CODE>" << endl;

    while(!fin.eof())

    {

        fin >> label >> opcode >> op1 >> op2; // reading the assembly code line by line

        int id; string IC, lc; // lc - LC processing, IC - Intermediate code

        id = getOP(opcode); // fetch the opcode entry

        IC = "(" + optab[id].mclass + "," + optab[id].mnemonic + ") ";

        // Individual cases for Assembly Directives (AD) - START, END, ORIGIN, EQU, LTORG

        // no LC processing for AD so lc = "---"

        if(opcode == "START")

        {

            lc = "---";

            if(op1 != "NAN")

            {

                LC = stoi(op1);

                IC += "(C," + op1 + ")  NAN";

            }

        }

        if(opcode == "EQU")

        {

            lc = "---";

            IC += " NAN     NAN";

            if(presentST(label))

            {

                ST[getSymID(label)].addr = ST[getSymID(op1)].addr;

            }

            else

            {

                ST[scnt].no = scnt + 1;

                ST[scnt].sname = label;

                ST[scnt].addr = ST[getSymID(op1)].addr;

                scnt++;

            }

        }

        else if(label != "NAN")

        {

            if(presentST(label))

            {

                ST[getSymID(label)].addr = to\_string(LC);

            }

            else

            {

                ST[scnt].no = scnt + 1;

                ST[scnt].sname = label;

                ST[scnt].addr = to\_string(LC);

                scnt++;

            }

        }

        if(opcode == "ORIGIN")

        {

            string token1, token2; char op;

            stringstream ss(op1);

            size\_t found = op1.find('+');

            if(found != string::npos)

                op = '+';

            else

                op = '-';

            getline(ss, token1, op); getline(ss, token2, op);

            lc = "---";

            if(op == '+')

            {

                LC = stoi(ST[getSymID(token1)].addr) + stoi(token2);

                IC += "(S,0" + to\_string(ST[getSymID(token1)].no) + ")+" + token2 + " NAN";

            }

            else

            {

                LC = stoi(ST[getSymID(token1)].addr) - stoi(token2);

                IC += "(S,0" + to\_string(ST[getSymID(token1)].no) + ")-" + token2 + " NAN";

            }

        }

        if(opcode == "LTORG")

        {

            cout << "  " << label << "\t" << opcode << "\t" << op1 << "\t" << op2 << "\t";

            for(int i = lcnt - nlcnt; i < lcnt; ++i)

            {

                lc = to\_string(LC);

                IC = "(DL,01) (C,";

                string c(1, LT[i].lname[2]);

                IC += c + ")    NAN";

                LT[i].addr = to\_string(LC);

                LC++;

                if(i < lcnt - 1)

                    cout << lc << "\t" << IC << "\n\t\t\t\t";

                else

                    cout << lc << "\t" << IC << endl;

                ic << lc << "\t" << IC << endl;

            }

            // managing pool table in LTORG

            PT[pcnt].lno = "#" + to\_string(LT[lcnt - nlcnt].no); PT[pcnt].no = pcnt + 1; pcnt++;

            nlcnt = 0;

            continue;

        }

        if(opcode == "END")

        {

            lc = "---";

            IC += " NAN     NAN";

            cout << "  " << label << "\t" << opcode << "\t" << op1 << "\t" << op2 << "\t" << lc << "\t" << IC << endl;

            ic << lc << "\t" << IC << endl;

            if(nlcnt)

            {

                for(int i = lcnt - nlcnt; i < lcnt; ++i)

                {

                    lc = to\_string(LC);

                    IC = "(DL,01) (C,";

                    string c(1, LT[i].lname[2]);

                    IC += c + ")    NAN";

                    LT[i].addr = to\_string(LC);

                    LC++;

                    cout << "\t\t\t\t" << lc << "\t" << IC << endl;

                    ic << lc << "\t" << IC << endl;

                }

                // managing pool table after END (if any literals are left)

                PT[pcnt].lno = "#" + to\_string(LT[lcnt - nlcnt].no); PT[pcnt].no = pcnt + 1; pcnt++;

            }

            break;

        }

        // Declarative Statements (DL)

        if(opcode == "DC" || opcode == "DS")

        {

            lc = to\_string(LC);

            if(opcode == "DS")

            {

                IC += "(C," + op1 + ")    NAN";

                LC += stoi(op1);

            }

            else

            {

                string c(1, op1[1]);

                IC += "(C," + c + ")";

                LC++;

            }

        }

        // if not AD or DL then, Imperative Statements (IS)

        if(opcode != "START" && opcode != "END" && opcode != "ORIGIN" && opcode != "EQU" && opcode != "LTORG" && opcode != "DC" && opcode != "DS")

        {

            if(op2 == "NAN")

            {

                if(op1 == "NAN")

                {

                    lc = to\_string(LC);

                    LC++;

                    IC += " NAN     NAN";

                }

                else

                {

                    if(presentST(op1))

                    {

                        IC += "(S,0" + to\_string(ST[getSymID(op1)].no) + ")";

                        lc = to\_string(LC);

                        LC++;

                    }

                    else

                    {

                        ST[scnt].no = scnt + 1;

                        ST[scnt].sname = op1;

                        scnt++;

                        IC += "(S,0" + to\_string(ST[getSymID(op1)].no) + ")";

                        lc = to\_string(LC);

                        LC++;

                    }

                }

            }

            else

            {

                if(opcode == "BC")

                    IC += "(" + to\_string(getConditionCode(op1)) + ")     ";

                else

                    IC += "(" + to\_string(getRegID(op1)) + ")     ";

                if(op2[0] == '=') // operand2 is a literal

                {

                    LT[lcnt].no = lcnt + 1;

                    LT[lcnt].lname = op2;

                    lcnt++; nlcnt++;

                    IC += "(L,0" + to\_string(LT[getLitID(op2)].no) + ")";

                }

                else // operand2 is a symbol

                {

                    if(presentST(op2))

                    {

                        IC += "(S,0" + to\_string(ST[getSymID(op2)].no) + ")";

                    }

                    else

                    {

                        ST[scnt].no = scnt + 1;

                        ST[scnt].sname = op2;

                        scnt++;

                        IC += "(S,0" + to\_string(ST[getSymID(op2)].no) + ")";

                    }

                }

                lc = to\_string(LC);

                LC++;

            }

        }

        // console output

        cout << "  " << label << "\t" << opcode << "\t" << op1 << "\t" << op2 << "\t" << lc << "\t" << IC << endl;

        ic << lc << "\t" << IC << endl;

    }

    cout << "\n----------------------------------------------------------------" << endl;

    cout << " -- SYMBOL TABLE --" << endl;

    cout << "\n <NO.\tSYMBOL\tADDRESS>" << endl;

    for(int i = 0; i < scnt; ++i)

    {

        cout << "  " << ST[i].no << "\t " << ST[i].sname << "\t  " << ST[i].addr << endl;

        st << ST[i].no << "\t " << ST[i].sname << "\t  " << ST[i].addr << endl;

    }

    cout << "\n----------------------------------------------------------------" << endl;

    cout << " -- LITERAL TABLE --" << endl;

    cout << "\n <NO.\tLITERAL\tADDRESS>" << endl;

    for(int i = 0; i < lcnt; ++i)

    {

        cout << "  " << LT[i].no << "\t " << LT[i].lname << "\t  " << LT[i].addr << endl;

        lt << LT[i].no << "\t " << LT[i].lname << "\t  " << LT[i].addr << endl;

    }

    cout << "\n----------------------------------------------------------------" << endl;

    cout << " -- POOL TABLE --" << endl;

    cout << "\n <NO.\tLITERAL\_NO.>" << endl;

    for(int i = 0; i < pcnt; ++i)

    {

        cout << "  " << PT[i].no << "\t   " << PT[i].lno << endl;

    }

    return 0;

}

**Literals:**

1 ='5' 211

2 ='1' 212

3 ='1' 219

**Symbols:**

1 A 217

2 LOOP 202

3 B 218

4 NEXT 214

5 BACK 202

6 LAST 216

**Output:**

--- -No Machine Code-

200 04 1 211

201 05 1 217

202 04 1 217

203 04 3 218

204 01 3 212

205 04 1 217

206 04 3 218

207 04 1 217

208 04 3 218

209 04 1 217

210 07 6 214

211 00 0 005

212 00 0 001

213 04 1 217

214 02 1 212

215 07 1 202

216 00 0 000

--- -No Machine Code-

204 03 3 218

--- -No Machine Code-

217 -No Machine Code-

--- -No Machine Code-

218 -No Machine Code-

--- -No Machine Code-

219 00 0 001