Assembler Assignment

U20CS005
BANSI MARAKANA

Pass1.h File:

```
#include <bits/stdc++.h>
using namespace std;

struct OPtab
{
    string opcode;
    string mclass;
    string mnemonic;
};

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"}
};

int getOP(string s)
{
    for(int i = 0; i < 18; ++i)
        if(optab[i].opcode == s)
```

```
            return i;
    return -1;
}

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;
}

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;
}

//Symbol Table
struct symTable
{
    int no;
```

```
    string sname;
    string addr;
};
struct symTable ST[10];
bool presentST(string s)
{
    for(int i = 0; i < 10; ++i)
        if(ST[i].sname == s)
            return true;
    return false;
}
int getSymID(string s)
{
    for(int i = 0; i < 10; ++i)
        if(ST[i].sname == s)
            return i;
    return -1;
}

//Literal Table
struct litTable
{
    int no;
    string lname;
    string addr;
};
struct litTable LT[10];
bool presentLT(string s)
{
    for(int i = 0; i < 10; ++i)
        if(LT[i].lname == s)
            return true;
    return false;
}
int getLitID(string s)
{
    for(int i = 0; i < 10; ++i)
        if(LT[i].lname == s)
            return i;
    return -1;
```

```
}

//Pool Table
struct poolTable
{
    int no;
    string lno;
};
struct poolTable PT[10];


int pass_1()
{
    ifstream fin;
    fin.open("input.txt");
    ofstream ic, st, lt;
    ic.open("ic.txt"); st.open("symtab.txt"); lt.open("ltorg.txt");
    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;
        int id; string IC, lc;
        id = getOP(opcode);
        IC = "(" + optab[id].mclass + "," + optab[id].mnemonic + ") ";
        if(opcode == "START")
        {
            lc = "---";
            if(op1 != "----")
            {
                LC = stoi(op1);
                IC += "(C," + op1 + ")  ----";
            }
        }
```

```
        if(opcode == "EQU")
        {
            lc = "---";
            IC += " ----     ----";
            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 != "----")
        {
            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 + " ----";
            }
            else
            {
                LC = stoi(ST[getSymID(token1)].addr) - stoi(token2);
                IC += "(S,0" + to_string(ST[getSymID(token1)].no) + ")-" +
token2 + " ----";
            }
        }

        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 + ")    ----";
                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;
            }
            PT[pcnt].lno = "#" + to_string(LT[lcnt - nlcnt].no);
PT[pcnt].no = pcnt + 1; pcnt++;
            nlcnt = 0;
            continue;
        }

        if(opcode == "END")
        {
            lc = "---";
```

```
            IC += " ----     ----";
            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 + ")    ----";
                    LT[i].addr = to_string(LC);
                    LC++;
                    cout << "\t\t\t\t" << lc << "\t" << IC << endl;
                    ic << lc << "\t" << IC << endl;
                }
                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 + ")    ----";
                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 == "----")
            {
                if(op1 == "----")
                {
                    lc = to_string(LC);
                    LC++;
                    IC += " ----     ----";
                }
                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++;
            }
        }
        cout << "  " << label << "\t" << opcode << "\t" << op1 << "\t" <<
op2 << "\t" << lc << "\t" << IC << endl;
        ic << lc << "\t" << IC << endl;
    }
    cout << "\n ----------------------------------- SYMBOL  TABLE
-----------------------------------" << endl;
    cout << "\n\t\t\t\t <NO.\tSYMBOL\tADDRESS>" << endl;
    for(int i = 0; i < scnt; ++i)
    {
        cout << "\t\t\t\t  " << 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 ----------------------------------- LITERAL TABLE
-----------------------------------" << endl;
    cout << "\n\t\t\t\t\t <NO.\tLITERAL\tADDRESS>" << endl;
    for(int i = 0; i < lcnt; ++i)
    {
        cout << "\t\t\t\t  " << 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 ------------------------------------ POOL  TABLE
------------------------------------" << endl;
    cout << "\n\t\t\t\t\t   <NO.\tLITERAL_NO.>" << endl;
    for(int i = 0; i < pcnt; ++i)
        cout << "\t\t\t\t    " << PT[i].no << "\t   " << PT[i].lno <<
endl;
    return 0;
}
```

```
#include <bits/stdc++.h>
#include "pass1.h"
using namespace std;

string table(ifstream &fin, string n)
{
    string no, name, addr;
    while(fin >> no >> name >> addr)
    {
        if(no == n)
        {
            fin.seekg(0, ios::beg);
            return addr;
        }
    }
    fin.seekg(0, ios::beg);
    return "----";
```

```
}

int main()
{
    int l=pass_1();
    ifstream ic, st, lt;
    ic.open("ic.txt"); st.open("symtable.txt"); lt.open("littable.txt");
    ofstream mc;
    mc.open("machine_code.txt");

    string lc, ic1, ic2, ic3;
    cout << "\n ------------------------------- ASSEMBLER PASS-2 OUTPUT
-------------------------------" << endl;
    cout << "\n LC\t <INTERMEDIATE CODE>\t\t\tLC\t <MACHINE CODE>" <<
endl;

    while(ic >> lc >> ic1 >> ic2 >> ic3)
    {
        string MC;
        if(ic1.substr(1, 2) == "AD" || (ic1.substr(1, 2) == "DL" &&
ic1.substr(4, 2) == "02"))
            MC = " -Machine-code not generated-";
        else if(ic1.substr(1, 2) == "DL" && ic1.substr(4, 2) == "01")
            MC = "00\t0\t00" + ic2.substr(3, 1);
        else
        {
            if(ic1 == "(IS,00)")
                MC = ic1.substr(4, 2) + "\t0\t000";
            else if(ic2.substr(1, 1) == "S")
                MC = ic1.substr(4, 2) + "\t0\t" + table(st, ic2.substr(4,
1));
            else
            {
                if(ic3.substr(1, 1) == "S")
                    MC = ic1.substr(4, 2) + "\t" + ic2.substr(1, 1) + "\t"
+ table(st, ic3.substr(4, 1));
                else
                    MC = ic1.substr(4, 2) + "\t" + ic2.substr(1, 1) + "\t"
+ table(lt, ic3.substr(4, 1));
            }
```

```
        }
        if(ic1 == "(AD,03)")
        {
            cout << " " << lc << "\t" << ic1 << "\t" << ic2 << " " << ic3
<< "\t\t\t" << lc << "\t" << MC << endl;
            mc << lc << "\t" << MC << endl;
            continue;
        }
        cout << " " << lc << "\t" << ic1 << "\t" << ic2 << "\t " << ic3 <<
"\t\t\t" << lc << "\t" << MC << endl;
        mc << lc << "\t" << MC << endl;
    }
    return 0;
}
```

```
------------------------------ ASSEMBLER PASS-1 OUTPUT ------------------------------

<LABEL OPCODE  OP1      OP2      LC      INTERMEDIATE CODE>
 ----  START   200      ----     ---     (AD,01) (C,200)  ----
 ----  MOVER   DREG     ='300'   200     (IS,04) (4)     (L,01)
 ----  MOVEM   DREG     M        201     (IS,05) (4)     (S,01)
 ----  LTORG   ----     ----     202     (DL,01) (C,3)    ----
 ----  MOVER   CREG     ='10'    203     (IS,04) (3)     (L,02)
 ----  MOVEM   DREG     D        204     (IS,05) (4)     (S,02)
 SOLVE MOVER   AREG     M        205     (IS,04) (1)     (S,01)
 ----  MOVER   CREG     G        206     (IS,04) (3)     (S,04)
 ----  ADD     CREG     ='100'   207     (IS,01) (3)     (L,03)
 ----  MOVER   AREG     M        208     (IS,04) (1)     (S,01)
 ----  MOVER   CREG     G        209     (IS,04) (3)     (S,04)
 ----  MOVER   AREG     M        210     (IS,04) (1)     (S,01)
 ----  MOVER   CREG     G        211     (IS,04) (3)     (S,04)
 ----  MOVER   AREG     M        212     (IS,04) (1)     (S,01)
 ----  BC      ANY      NEXT     213     (IS,07) (6)     (S,05)
 ----  LTORG   ----     ----     214     (DL,01) (C,1)    ----
                                 215     (DL,01) (C,1)    ----
 ----  MOVER   AREG     M        216     (IS,04) (1)     (S,01)
 NEXT  SUB     AREG     ='150'   217     (IS,02) (1)     (L,04)
 ----  BC      LT       BACK     218     (IS,07) (1)     (S,06)
 NOPE  STOP    ----     ----     219     (IS,00)  ----     ----
 ----  ORIGIN  SOLVE+12         ----    ---     (AD,03) (S,03)+12 ----
 ----  MULT    CREG     G        217     (IS,03) (3)     (S,04)
 ----  ORIGIN  NOPE+15 ----     ---     (AD,03) (S,07)+15 ----
 M     DS      1        ----     234     (DL,02) (C,1)    ----
 BACK  EQU     SOLVE    ----     ---     (AD,04)  ----     ----
 G     DS      1        ----     235     (DL,02) (C,1)    ----
 D     DS      1        ----     236     (DL,02) (C,1)    ----
 ----  END     ----     ----     ---     (AD,02)  ----     ----
                                 237     (DL,01) (C,1)    ----

----------------------------------- SYMBOL  TABLE -----------------------------------

                              <NO.   SYMBOL  ADDRESS>
                               1      M        234
                               2      D        236
                               3      SOLVE    205
                               4      G        235
                               5      NEXT     217
                               6      BACK     205
                               7      NOPE     219
```

```
---------------------------------- LITERAL TABLE ----------------------------------

                              <NO.   LITERAL ADDRESS>
                               1      ='300'   202
                               2      ='10'    214
                               3      ='100'   215
                               4      ='150'   237

---------------------------------- POOL  TABLE ----------------------------------

                                <NO. LITERAL_NO.>
                                 1      #1
                                 2      #2
                                 3      #4

------------------------------ ASSEMBLER PASS-2 OUTPUT ------------------------------

LC      <INTERMEDIATE CODE>                 LC       <MACHINE CODE>
---    (AD,01) (C,200)  ----                ---      -Machine-code not generated-
200    (IS,04) (4)      (L,01)              200     04      4       ----
201    (IS,05) (4)      (S,01)              201     05      4       ----
202    (DL,01) (C,3)    ----                202     00      0       003
203    (IS,04) (3)      (L,02)              203     04      3       ----
204    (IS,05) (4)      (S,02)              204     05      4       ----
205    (IS,04) (1)      (S,01)              205     04      1       ----
206    (IS,04) (3)      (S,04)              206     04      3       ----
207    (IS,01) (3)      (L,03)              207     01      3       ----
208    (IS,04) (1)      (S,01)              208     04      1       ----
209    (IS,04) (3)      (S,04)              209     04      3       ----
210    (IS,04) (1)      (S,01)              210     04      1       ----
211    (IS,04) (3)      (S,04)              211     04      3       ----
212    (IS,04) (1)      (S,01)              212     04      1       ----
213    (IS,07) (6)      (S,05)              213     07      6       ----
214    (DL,01) (C,1)    ----                214     00      0       001
215    (DL,01) (C,1)    ----                215     00      0       001
216    (IS,04) (1)      (S,01)              216     04      1       ----
217    (IS,02) (1)      (L,04)              217     02      1       ----
218    (IS,07) (1)      (S,06)              218     07      1       ----
219    (IS,00) ----     ----                219     00      0       000
---    (AD,03) (S,03)+12 ----               ---      -Machine-code not generated-
217    (IS,03) (3)      (S,04)              217     03      3       ----
---    (AD,03) (S,07)+15 ----               ---      -Machine-code not generated-
234    (DL,02) (C,1)    ----                234      -Machine-code not generated-
---    (AD,04) ----     ----                ---      -Machine-code not generated-
235    (DL,02) (C,1)    ----                235      -Machine-code not generated-
236    (DL,02) (C,1)    ----                236      -Machine-code not generated-
---    (AD,02) ----     ----                ---      -Machine-code not generated-
237    (DL,01) (C,1)    ----                237     00      0       001
PS D:\BANSI MARAKANA\SS>
```