# Lab Assignment - 5


## U21CS089

## Garvit Shah


1) Write a C Program to Implement **Two Pass Assembler**.

PASS-1

```cpp
// PASS 1 Of Two-Pass Assembler

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

// [U19CS012] BHAGYA VINOD RANA

// To store mnemonics of the opcodes
// Operational Table.
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);

// Function to fetch the register code
int getRegID(string s);

// Function to fetch conditional code
int getConditionCode(string s);

// 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);

// Function to fetch the symbol entry
int getSymID(string s);

// 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);

// Function to fetch the literal entry
int getLitID(string s);

// 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("source.asm");

    ofstream ic, st, lt, pt;

    // Saving the output of pass1 into pass2 source code directory.
    // Since it will be the input for pass2.cpp
    // The paths may change accordingly

    ic.open("ic.txt");
    st.open("symtable.txt");
    lt.open("littable.txt");
    pt.open("pooltable.txt");

    string label, opcode, op1, op2;

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

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

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

    while (!fin.eof())
    {
        // reading the assembly code line by line
        fin >> label >> opcode >> op1 >> op2;

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

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

        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 << " ~x~x~x~ SYMBOL TABLE ~x~x~x~" << 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 << " ~x~x~x~ LITERAL TABLE ~x~x~x~" << 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 << " ~x~x~x~ POOL TABLE ~x~x~x~" << endl;
    cout << "\n <NO.\tLITERAL_NO.>" << endl;
    for (int i = 0; i < pcnt; ++i)
    {
        cout << " " << PT[i].no << "\t  " << PT[i].lno << endl;
        pt << PT[i].no << "\t   " << PT[i].lno << endl;
    }

    return 0;
}

// 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;
    }
}

// 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;
}

// 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;
}
```

PASS - 2

```cpp
// PASS 2 Of Two-Pass Assembler

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

// [U19CS012] BHAGYA VINOD RANA

// Function to fetch symbol/literal address from symbol_table or literal_table
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 "NAN";
}

int main()
{

    ifstream ic, st, lt;
    // pass1 output files as input to pass2
    ic.open("ic.txt");
    st.open("symtable.txt");
    lt.open("littable.txt");
    // generate file output of machine code
    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;

    // reading input file line by line
    while (ic >> lc >> ic1 >> ic2 >> ic3)
    {
        // machine code
        string MC;

        // no machine code for AD and DL,02 i.e. DS opcodes
```

```
        if (ic1.substr(1, 2) == "AD" || (ic1.substr(1, 2) == "DL" && ic1.substr(4, 2)
== "02"))
        {

            MC = " -No Machine Code-";
        }
        // if opcode is DL i.e. DL,01 then display constant value at the place of
memory operand
        else if (ic1.substr(1, 2) == "DL" && ic1.substr(4, 2) == "01")
        {
            MC = "00\t0\t00" + ic2.substr(3, 1);
        }
        else
        {
            // IS opcode
            if (ic1 == "(IS,00)")
            { // specifically for STOP
                MC = ic1.substr(4, 2) + "\t0\t000";
            }
            else if (ic2.substr(1, 1) == "S")
            { // if opcode in pass1 was ORIGIN
                MC = ic1.substr(4, 2) + "\t0\t" + table(st, ic2.substr(4, 1));
            }
            else
            {
                if (ic3.substr(1, 1) == "S")
                    // for symbols
                    MC = ic1.substr(4, 2) + "\t" + ic2.substr(1, 1) + "\t" +
table(st, ic3.substr(4, 1));
                else
                    // for literals
                    MC = ic1.substr(4, 2) + "\t" + ic2.substr(1, 1) + "\t" +
table(lt, ic3.substr(4, 1));
            }
        }
        if (ic1 == "(AD,03)")
        {
            // just for console output display format
            cout << " " << lc << "\t" << ic1 << "\t" << ic2 << " " << ic3 << "\t\t\t"
<< lc << "\t" << MC << endl;
            mc << lc << "\t" << MC << endl;
            continue;
        }
        // console output
        cout << " " << lc << "\t" << ic1 << "\t" << ic2 << "\t " << ic3 << "\t\t\t"
<< lc
             << "\t" << MC << endl;
        mc << lc << "\t" << MC << endl;
    }
    return 0;
```

```
}
```

## PASS-1 I/O

| INPUT | OUTPUT |
|---|---|
| **source.asm->**assembly language code | **ic.txt** containing intermediate code |
| **Prebuilt** OPTAB | **littable.txt** containing literal table |
| | **symtable.txt** containing symbol table |
| | **pooltable.txt** containing pool table |

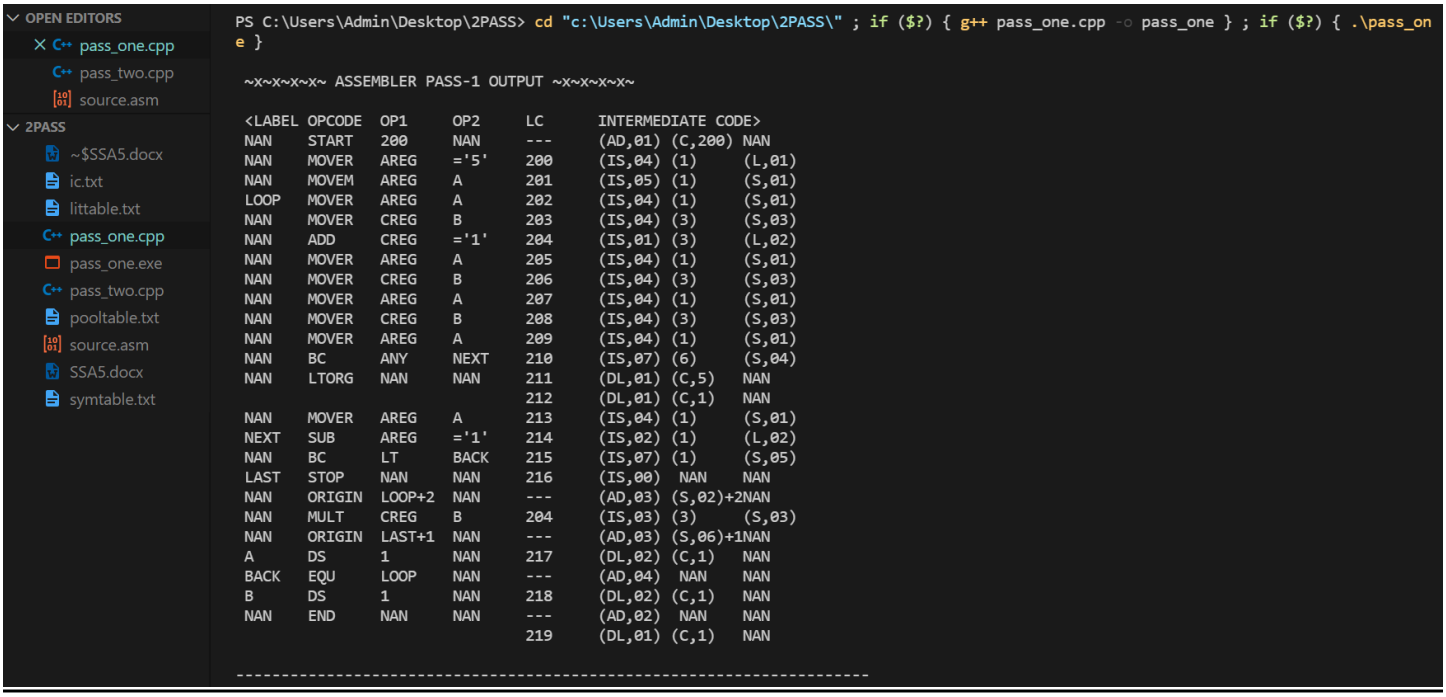
## PASS-2 I/O

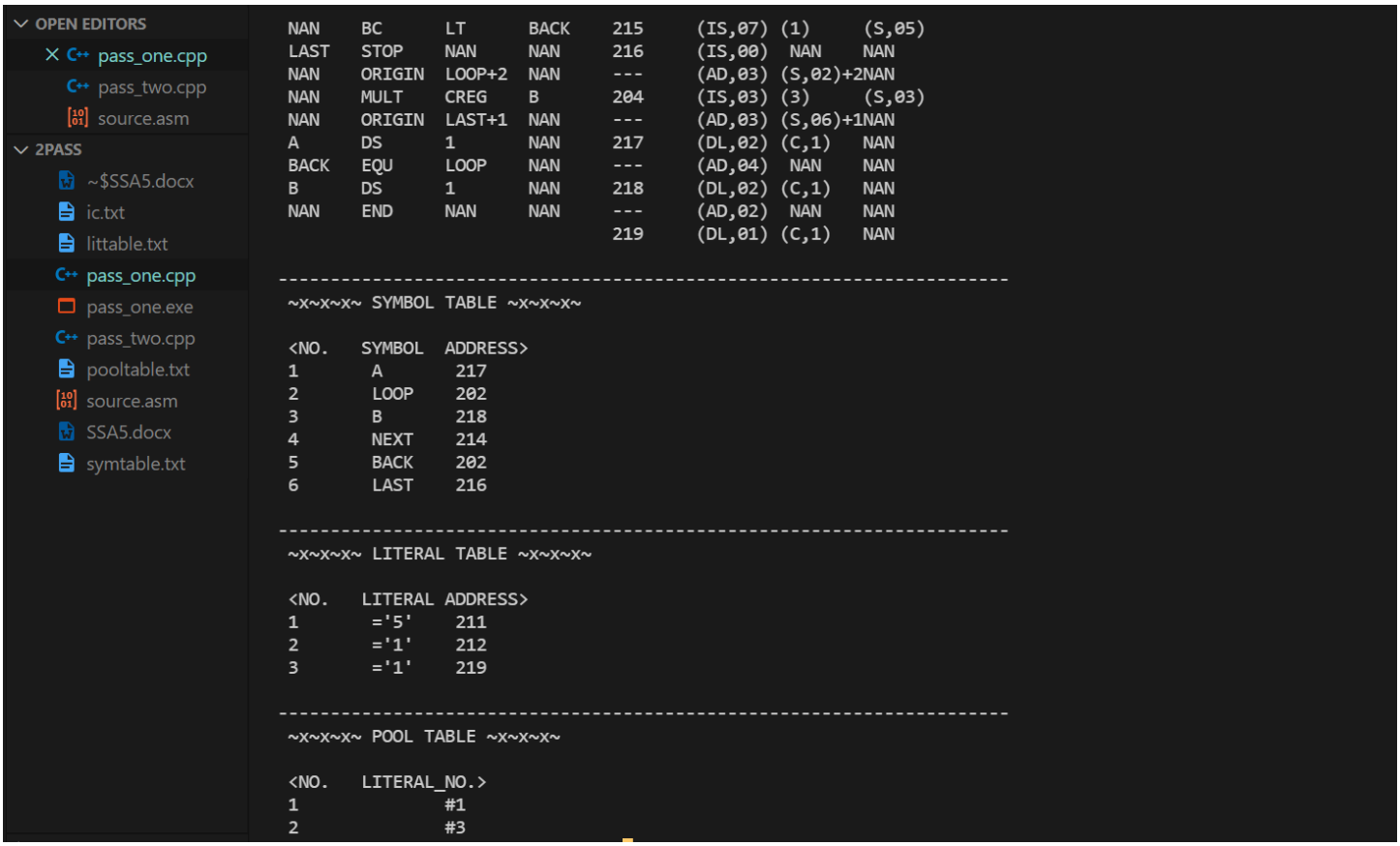
| INPUT | OUTPUT |
|---|---|
| ic.txt **containing intermediate code** | **machine_code.txt** containing machine code. |
| littable.txt **containing literal table** | |
| symtable.txt **containing symbol table** | |

## How to execute?

1. Compile and execute **pass_one.cpp** source code by providing **source.asm** as input (save it in the same folder as pass1.cpp).
2. The output of this file will be shown on terminal as well as saved in the files name "**littable.txt**", "**symtable.txt**", "**ic.txt**", "**pooltable.txt**".
3. Now, compile and execute **pass_two.cpp** source code. It will take ic.txt, littable.txt, symtable.txt as an **input**.
4. The output will be saved in "**machine_code.txt**" file.

# After Executing PASS-1




OPEN EDITORS
pass_one.cpp
pass_two.cpp
source.asm
2PASS
~$SSA5.docx
ic.txt
littable.txt
pass_one.cpp
pass_one.exe
pass_two.cpp
pooltable.txt
source.asm
SSA5.docx
symtable.txt
NAN BC LT BACK 215 (IS,07) (1) (S,05)
LAST STOP NAN NAN 216 (IS,00) NAN NAN
NAN ORIGIN LOOP+2 NAN --- (AD,03) (S,02)+2NAN
NAN MULT CREG B 204 (IS,03) (3) (S,03)
NAN ORIGIN LAST+1 NAN --- (AD,03) (S,06)+1NAN
A DS 1 NAN 217 (DL,02) (C,1) NAN
BACK EQU LOOP NAN --- (AD,04) NAN NAN
B DS 1 NAN 218 (DL,02) (C,1) NAN
NAN END NAN NAN --- (AD,02) NAN NAN
219 (DL,01) (C,1) NAN
~X~X~X~ SYMBOL TABLE ~X~X~X~
<NO. SYMBOL ADDRESS>
1 A 217
2 LOOP 202
3 B 218
4 NEXT 214
5 BACK 202
6 LAST 216
~X~X~X~ LITERAL TABLE ~X~X~X~
<NO. LITERAL ADDRESS>
1 ='5' 211
2 ='1' 212
3 ='1' 219
~X~X~X~ POOL TABLE ~X~X~X~
<NO. LITERAL_NO.>
1 #1
2 #3

**ic.txt**

ic.txt

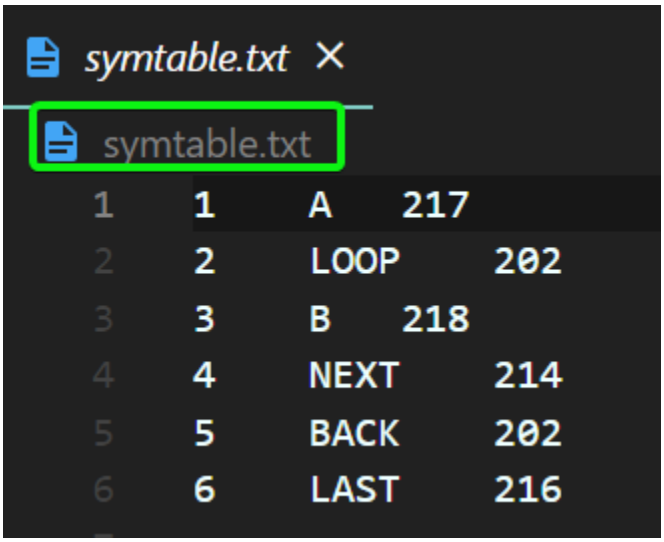
```
1   --- (AD,01) (C,200) NAN
2   200 (IS,04) (1) (L,01)
3   201 (IS,05) (1) (S,01)
4   202 (IS,04) (1) (S,01)
5   203 (IS,04) (3) (S,03)
6   204 (IS,01) (3) (L,02)
7   205 (IS,04) (1) (S,01)
8   206 (IS,04) (3) (S,03)
9   207 (IS,04) (1) (S,01)
10  208 (IS,04) (3) (S,03)
11  209 (IS,04) (1) (S,01)
12  210 (IS,07) (6) (S,04)
13  211 (DL,01) (C,5)   NAN
14  212 (DL,01) (C,1)   NAN
15  213 (IS,04) (1) (S,01)
16  214 (IS,02) (1) (L,02)
17  215 (IS,07) (1) (S,05)
18  216 (IS,00)  NAN    NAN
19  --- (AD,03) (S,02)+2NAN
20  204 (IS,03) (3) (S,03)
21  --- (AD,03) (S,06)+1NAN
22  217 (DL,02) (C,1)   NAN
23  --- (AD,04)  NAN    NAN
24  218 (DL,02) (C,1)   NAN
25  --- (AD,02)  NAN    NAN
26  219 (DL,01) (C,1)   NAN
```

littable.txt

littable.txt

```
1   1    ='5'    211
2   2    ='1'    212
3   3    ='1'    219
```

## symtable.txt




## pooltable.txt


## After Executing PASS-2

```
PS C:\Users\Admin\Desktop\2PASS> cd "c:\Users\Admin\Desktop\2PASS\" ; if ($?) { g++ pass_two.cpp -o pass_two } ; if ($?) { .\pass_two }

 -- ASSEMBLER PASS-2 OUTPUT --

 LC      <INTERMEDIATE CODE>                LC       <MACHINE CODE>
 ---    (AD,01) (C,200)  NAN                ---      -No Machine Code-
 200    (IS,04) (1)      (L,01)             200     04      1       211
 201    (IS,05) (1)      (S,01)             201     05      1       217
 202    (IS,04) (1)      (S,01)             202     04      1       217
 203    (IS,04) (3)      (S,03)             203     04      3       218
 204    (IS,01) (3)      (L,02)             204     01      3       212
 205    (IS,04) (1)      (S,01)             205     04      1       217
 206    (IS,04) (3)      (S,03)             206     04      3       218
 207    (IS,04) (1)      (S,01)             207     04      1       217
 208    (IS,04) (3)      (S,03)             208     04      3       218
 209    (IS,04) (1)      (S,01)             209     04      1       217
 210    (IS,07) (6)      (S,04)             210     07      6       214
 211    (DL,01) (C,5)    NAN                211     00      0       005
 212    (DL,01) (C,1)    NAN                212     00      0       001
 213    (IS,04) (1)      (S,01)             213     04      1       217
 214    (IS,02) (1)      (L,02)             214     02      1       212
 215    (IS,07) (1)      (S,05)             215     07      1       202
 216    (IS,00) NAN      NAN                216     00      0       000
 ---    (AD,03) (S,02)+2NAN 204             ---      -No Machine Code-
PS C:\Users\Admin\Desktop\2PASS>
```

**machine_code.txt**