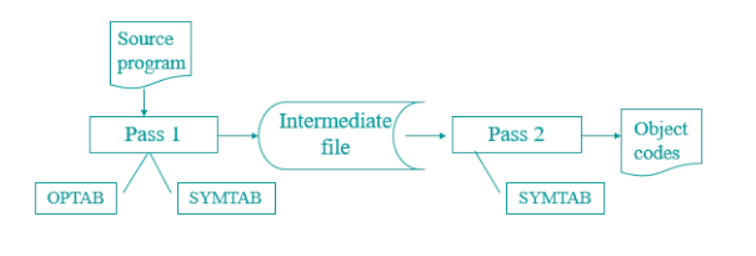
System Software (CS306)

Assignment

**U20cs135**

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



PASS-1

**Code:**

*// 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**

**Code:**

*// 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/0**

|  |  |
| --- | --- |
| **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 |

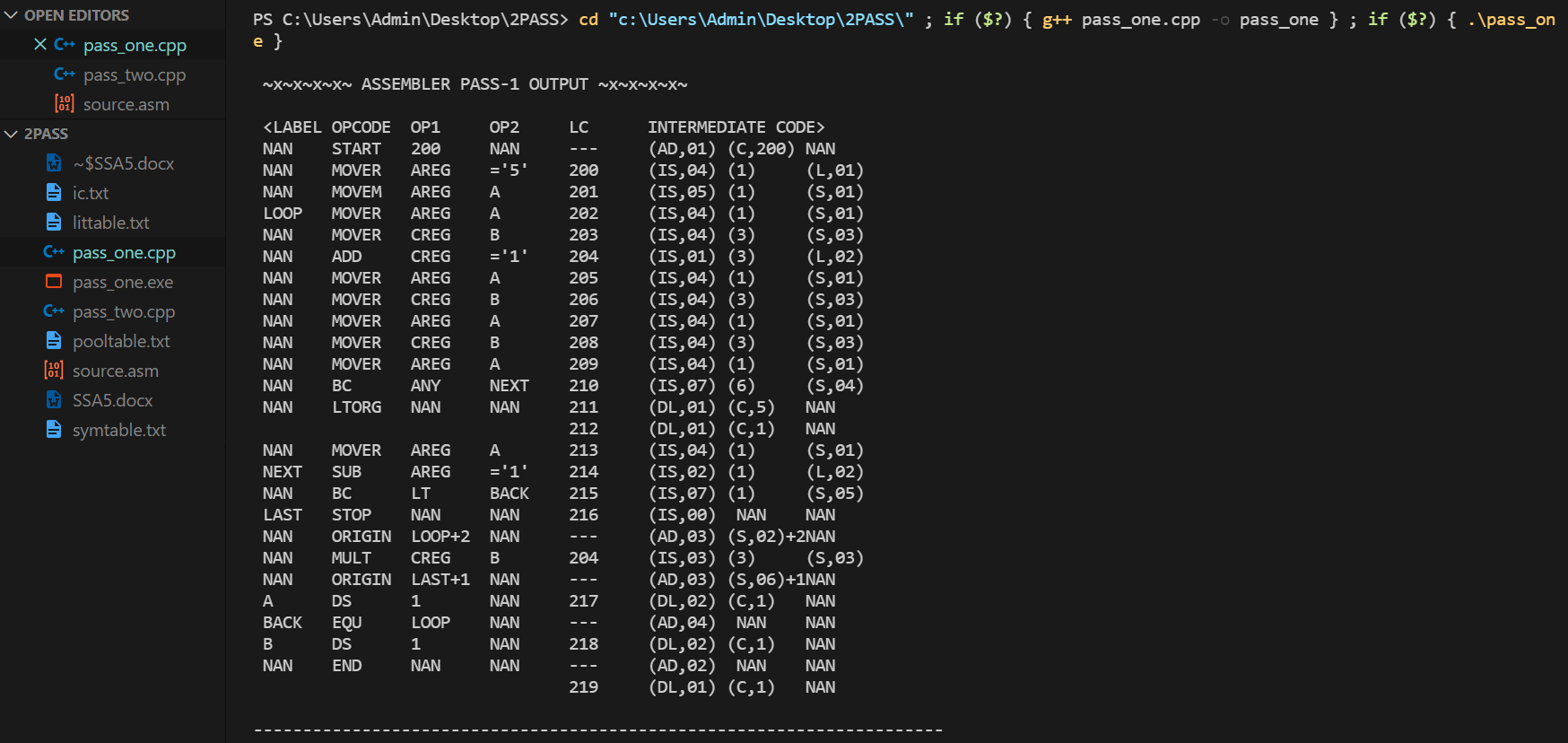
**PASS-2 I/0**

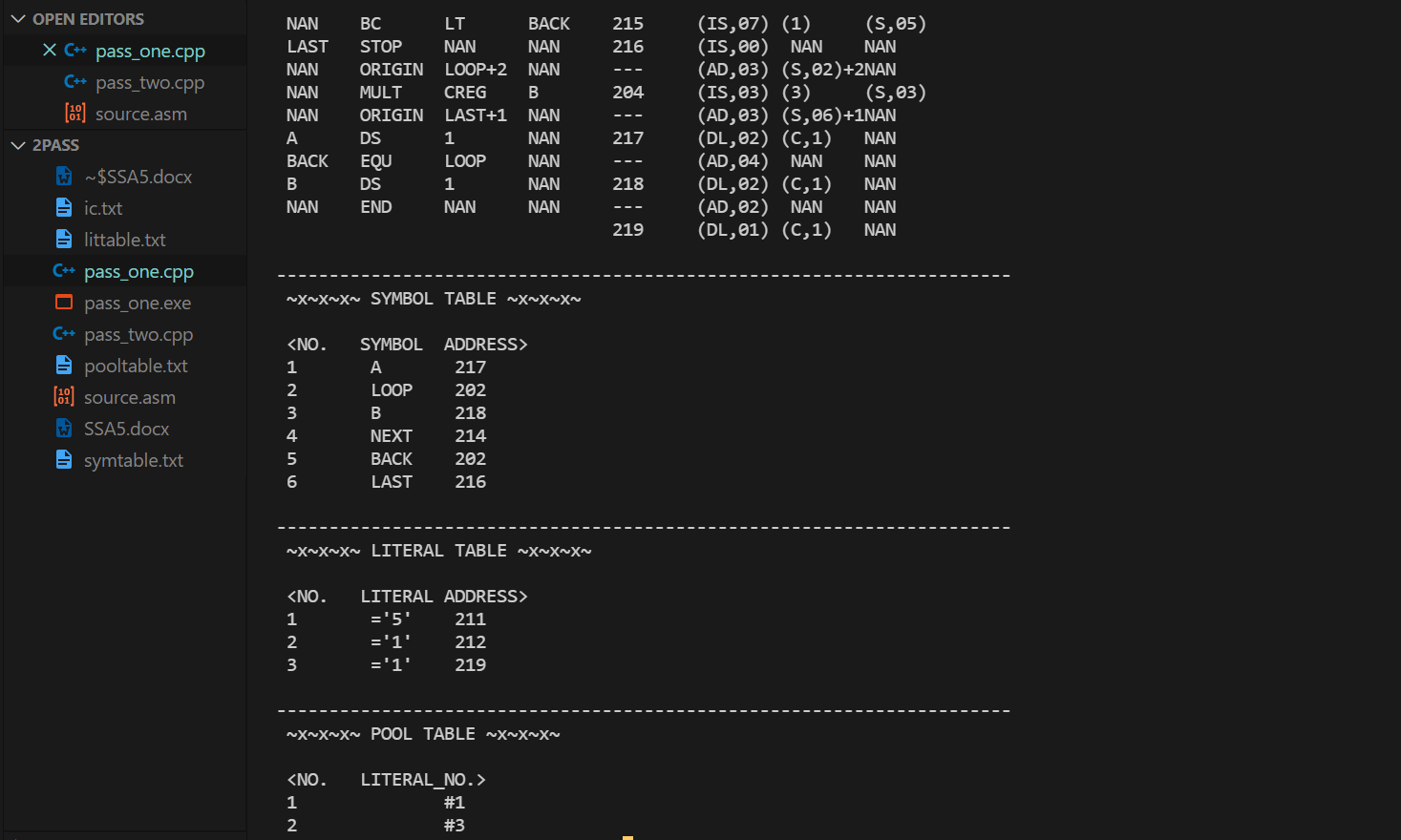
|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **ic.txt containing intermediate code** | **machine\_code.txt** containing machine code. |
| **littable.txt containing literal table** |  |
| **symtable.txt containing symbol table** |  |

**How to execute?**

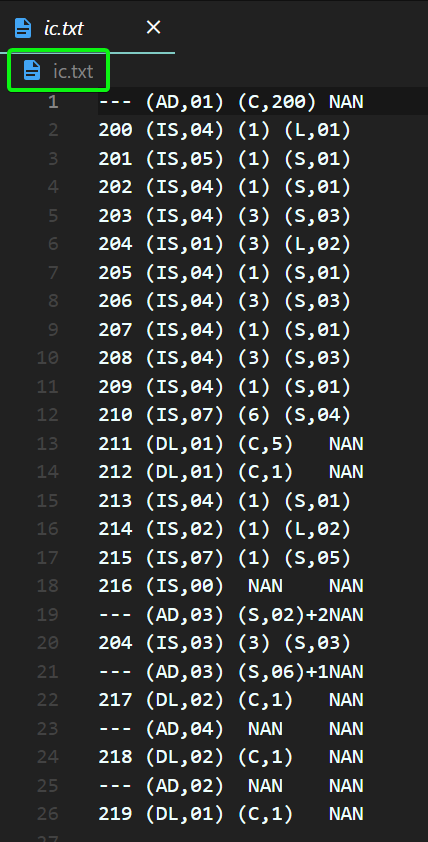
1. Compile and execute **pass\_one.cpp** source code by providing **source.asm** as input (save it in the same folder as pass1.cpp).
2. The output of this file will be shown on terminal as well as saved in the files name “**littable.txt**”, “**symtable.txt**”, “**ic.txt**”, “**pooltable.txt**”.
3. Now, 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**

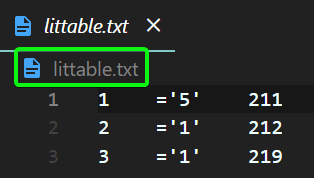




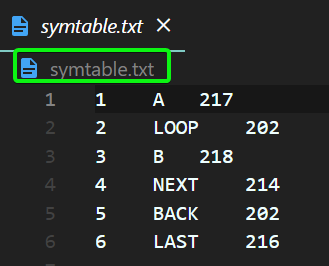
**ic.txt**



**littable.txt**



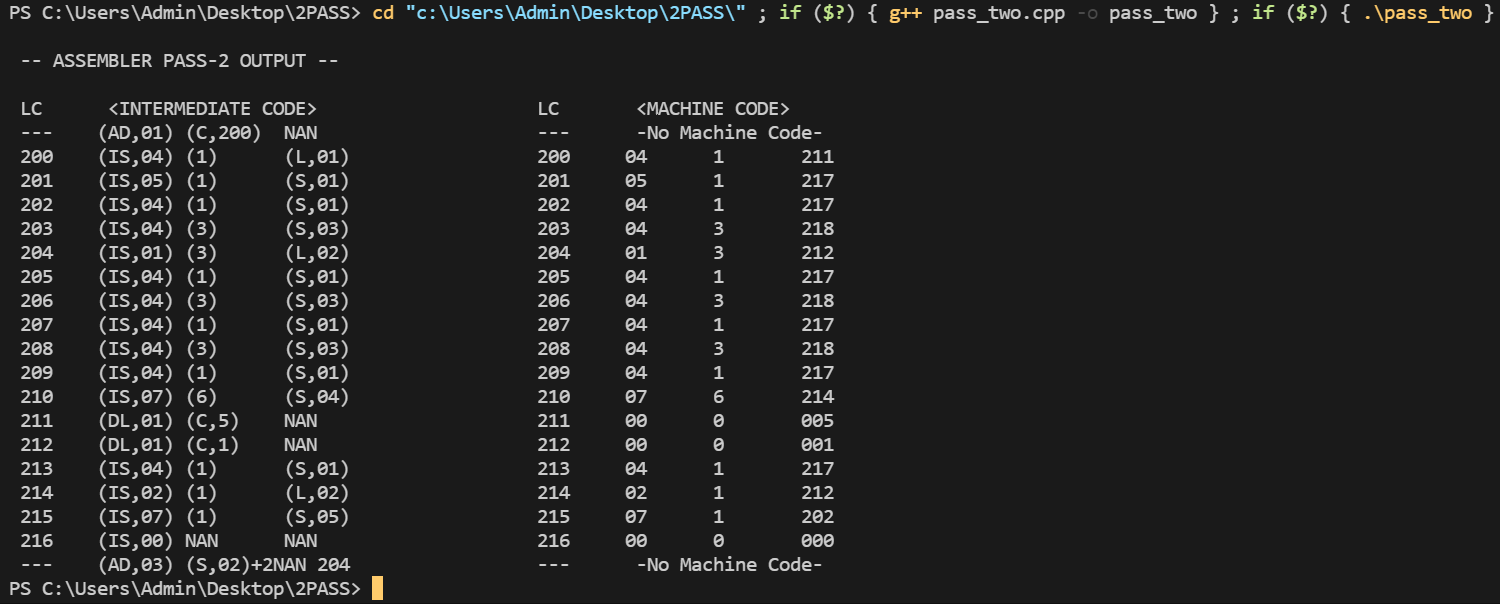
**symtable.txt**



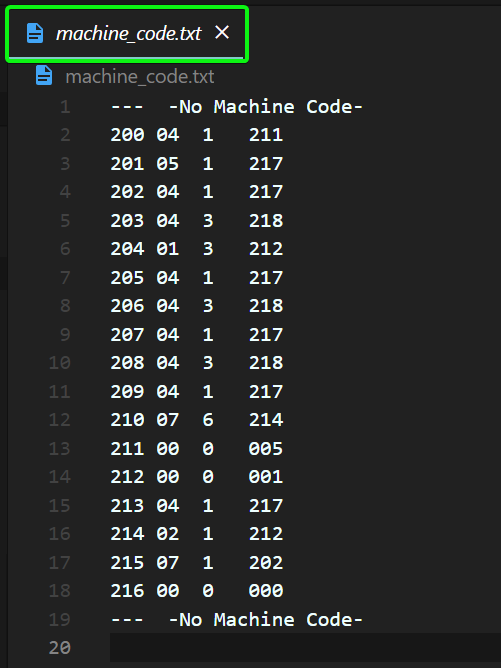
**pooltable.txt**



**After Executing PASS-2**



**machine\_code.txt**



**SUBMITTED BY**: U20cs135

Shivam Mishra