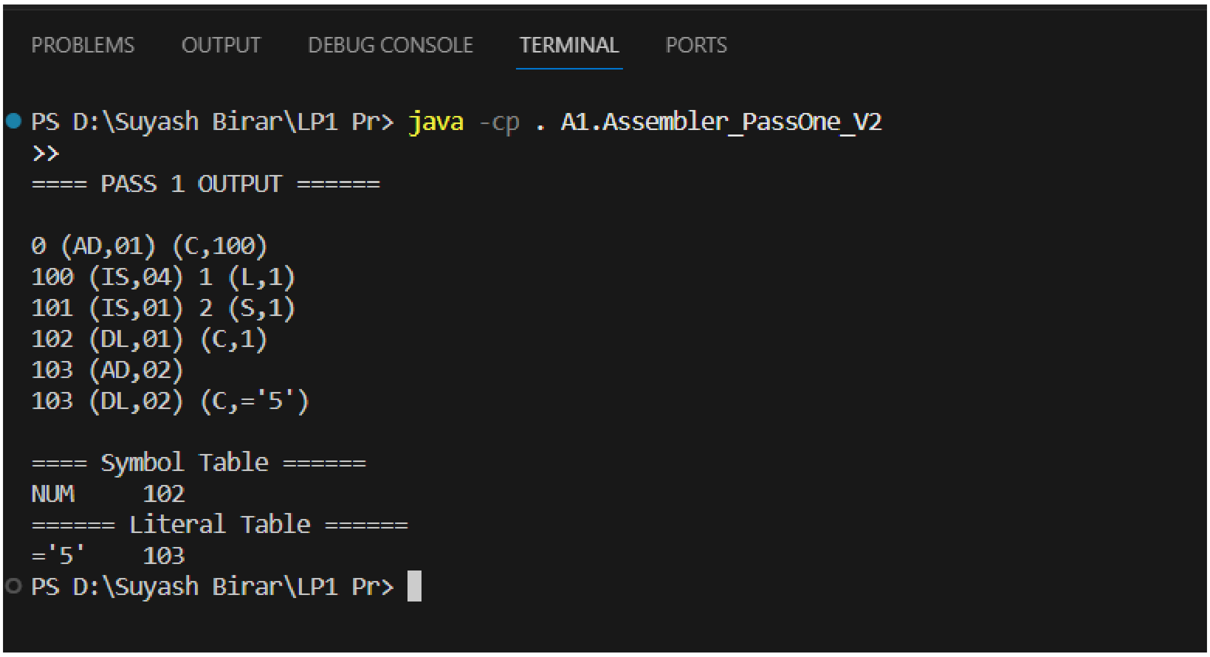
Practical No 01

```
package A1;

import java.io.*;
import java.util.*;

class Tuple {
    String mnemonic, m_class, opcode;
    int length;

    Tuple() {}

    Tuple(String s1, String s2, String s3, String s4) {
        mnemonic = s1;
        m_class = s2;
        opcode = s3;
        length = Integer.parseInt(s4);
    }
}

class SymTuple {
    String symbol, address;
    int length;

    SymTuple(String s1, String s2, int il) {
        symbol = s1;
        address = s2;
        length = il;
    }
}

class LitTuple {
    String literal, address;
    int length;

    LitTuple() {}

    LitTuple(String s1, String s2, int l) {
        literal = s1;
        address = s2;
        length = l;
    }
}

public class Assembler_PassOne_V2 {

    static int lc, iSymTabPtr = 0, iLitTabPtr = 0, iPoolTabPtr = 0;
    static int[] poolTable = new int[10];

    static Map<String, Tuple> MOT;
    static Map<String, SymTuple> symtable;
    static ArrayList<LitTuple> littable;
    static Map<String, String> regAddressTable;

    static PrintWriter out_pass1;

    public static void main(String[] args) throws Exception {
        initializeTables();
        System.out.println("==== PASS 1 OUTPUT ======\n");
        pass1();
    }

    static void pass1() throws Exception {
        BufferedReader input = new BufferedReader(new InputStreamReader(new
FileInputStream("A1/input.txt")));
        out_pass1 = new PrintWriter(new FileWriter("A1/output_pass1.txt"), true);
        PrintWriter out_symtable = new PrintWriter(new FileWriter("A1/symtable.txt"), true);
        PrintWriter out_littable = new PrintWriter(new FileWriter("A1/littable.txt"), true);

        String s;
        lc = 0;

        while ((s = input.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, " ", false);
            String[] s_arr = new String[st.countTokens()];
            for (int i = 0; i < s_arr.length; i++) {
                s_arr[i] = st.nextToken();
            }

            if (s_arr.length == 0) continue;

            int curIndex = 0;

            if (s_arr.length == 3) {
                String label = s_arr[0];
                insertIntoSymTab(label, lc + "");
                curIndex = 1;
            }

            String curToken = s_arr[curIndex];
            Tuple curTuple = MOT.get(curToken);

            if (curTuple == null) continue;

            String intermediateStr = "";

            if (curTuple.m_class.equalsIgnoreCase("IS")) {
                intermediateStr += lc + " (" + curTuple.m_class + "," + curTuple.opcode + ") ";
                intermediateStr += processOperands(s_arr[curIndex + 1]);
                lc += curTuple.length;

            } else if (curTuple.m_class.equalsIgnoreCase("AD")) {
                if (curTuple.mnemonic.equalsIgnoreCase("START")) {
                    intermediateStr += lc + " (" + curTuple.m_class + "," + curTuple.opcode + ") ";
                    lc = Integer.parseInt(s_arr[curIndex + 1]);
                    intermediateStr += "(C," + s_arr[curIndex + 1] + ") ";
                } else if (curTuple.mnemonic.equalsIgnoreCase("LTORG")) {
                    intermediateStr += processLTORG();
                } else if (curTuple.mnemonic.equalsIgnoreCase("END")) {
                    intermediateStr += lc + " (" + curTuple.m_class + "," + curTuple.opcode + ") \n";
                    intermediateStr += processLTORG();
                }
            } else if (curTuple.m_class.equalsIgnoreCase("DL")) {
                intermediateStr += lc + " (" + curTuple.m_class + "," + curTuple.opcode + ") ";
                if (curTuple.mnemonic.equalsIgnoreCase("DS")) {
                    lc += Integer.parseInt(s_arr[curIndex + 1]);
                } else if (curTuple.mnemonic.equalsIgnoreCase("DC")) {
                    lc += curTuple.length;
                }
                intermediateStr += "(C," + s_arr[curIndex + 1] + ") ";
            }

            System.out.println(intermediateStr);
            out_pass1.println(intermediateStr);
        }

        out_pass1.flush();
        out_pass1.close();

        // Print symbol table
        System.out.println("==== Symbol Table ======");
        for (SymTuple tuple : symtable.values()) {
            String tableEntry = tuple.symbol + "\t" + tuple.address;
            out_symtable.println(tableEntry);
            System.out.println(tableEntry);
        }

        out_symtable.flush();
        out_symtable.close();

        // Print literal table
        System.out.println("====== Literal Table ======");
        for (LitTuple litTuple : littable) {
            String tableEntry = litTuple.literal + "\t" + litTuple.address;
            out_littable.println(tableEntry);
            System.out.println(tableEntry);
        }

        out_littable.flush();
        out_littable.close();
    }

    static String processLTORG() {
        String intermediateStr = "";
        for (int i = poolTable[iPoolTabPtr]; i < littable.size(); i++) {
            LitTuple litTuple = littable.get(i);
            litTuple.address = lc + "";
            intermediateStr += lc + " (DL,02) (C," + litTuple.literal + ")\n";
            lc++;
        }
        poolTable[++iPoolTabPtr] = iLitTabPtr;
        return intermediateStr;
    }

    static void insertIntoSymTab(String symbol, String address) {
        if (!symtable.containsKey(symbol)) {
            symtable.put(symbol, new SymTuple(symbol, address, 1));
            iSymTabPtr++;
        } else {
            SymTuple entry = symtable.get(symbol);
            if (entry.address.equals("-")) {
                entry.address = address;
            }
        }
    }

    static void insertIntoLitTab(String literal, String address) {
        littable.add(iLitTabPtr, new LitTuple(literal, address, 1));
        iLitTabPtr++;
    }

    static String processOperands(String operandStr) {
        StringTokenizer st = new StringTokenizer(operandStr, ",", false);
        String result = "";
        while (st.hasMoreTokens()) {
            String op = st.nextToken();
            if (regAddressTable.containsKey(op)) {
                result += regAddressTable.get(op) + " ";
            } else if (op.startsWith("=")) {
                insertIntoLitTab(op, "-");
                result += "(L," + iLitTabPtr + ") ";
            } else {
                insertIntoSymTab(op, "-");
                result += "(S," + iSymTabPtr + ") ";
            }
        }
        return result;
    }

    static void initializeTables() throws Exception {
        symtable = new LinkedHashMap<>();
        littable = new ArrayList<>();
        regAddressTable = new HashMap<>();
        MOT = new HashMap<>();

        BufferedReader br = new BufferedReader(new InputStreamReader(new
FileInputStream("A1/mot.txt")));
```

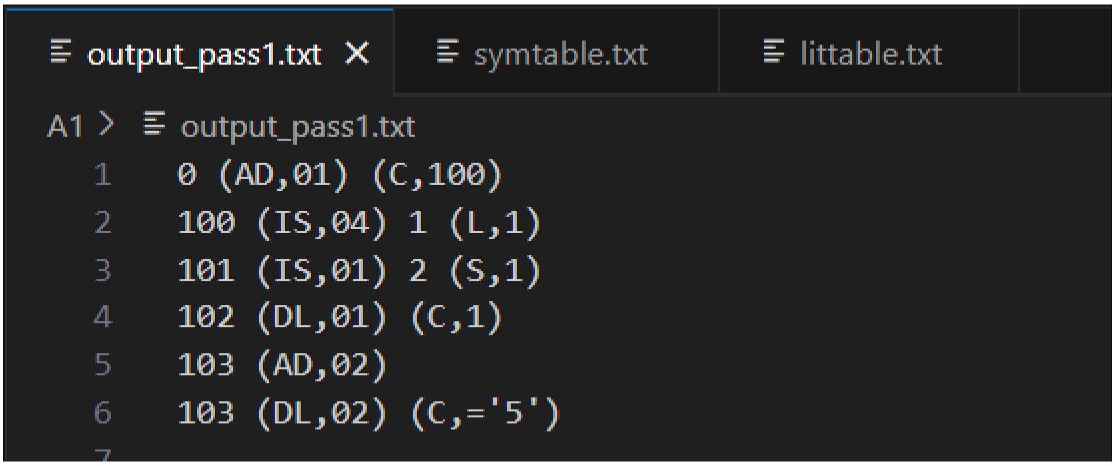
```
        String s;
        while ((s = br.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, " ", false);
            String mnemonic = st.nextToken();
            MOT.put(mnemonic, new Tuple(mnemonic, st.nextToken(), st.nextToken(),
st.nextToken()));
        }
        br.close();

        regAddressTable.put("AREG", "1");
        regAddressTable.put("BREG", "2");
        regAddressTable.put("CREG", "3");
        regAddressTable.put("DREG", "4");

        poolTable[iPoolTabPtr] = iLitTabPtr;
    }
}
```

```
// Input.txt

START 100
MOVER AREG,='5'
ADD BREG,NUM
NUM DS 1
END
```

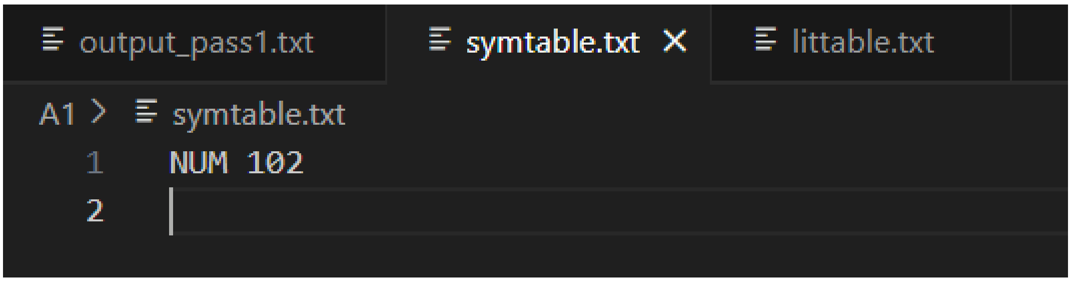
```
//mot.txt

START AD 01 0
END AD 02 0
LTORG AD 03 0
DS DL 01 1
DC DL 02 1
ADD IS 01 1
SUB IS 02 1
MULT IS 03 1
MOVER IS 04 1
MOVEM IS 05 1
```



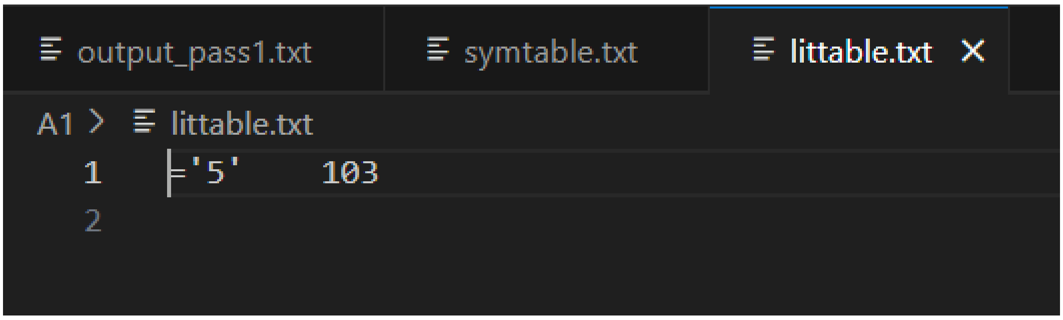
```
PS D:\Suyash Birar\LP1 Pr> java -cp . A1.Assembler_PassOne_V2
>>
==== PASS 1 OUTPUT ======

0 (AD,01) (C,100)
100 (IS,04) 1 (L,1)
101 (IS,01) 2 (S,1)
102 (DL,01) (C,1)
103 (AD,02)
103 (DL,02) (C,='5')

==== Symbol Table ======
NUM     102
====== Literal Table ======
='5'    103
PS D:\Suyash Birar\LP1 Pr>
```



```
0 (AD,01) (C,100)
100 (IS,04) 1 (L,1)
101 (IS,01) 2 (S,1)
102 (DL,01) (C,1)
103 (AD,02)
103 (DL,02) (C,='5')
```



```
NUM 102

```



```
='5'    103

```