Problem Statement: Implement one pass-I of TWO Pass assembler with hypothetical Instruction set using Java language. Instruction set should include all types of assembly language statements such as Imperative, Declarative and Assembler Directive. While designing stress should be given on a) How efficiently Mnemonic opcode could be implemented so as to enable faster retrieval on op-code. b) Implementation of symbol table for faster retrieval.

```
import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.FileWriter;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.StringTokenizer;


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 i1) {
        symbol = s1;
        address = s2;
        length = i1;
    }
}

class LitTuple {
    String literal, address;
    int length;

    LitTuple() {}

    LitTuple(String s1, String s2, int i1) {
        literal = s1;
        address = s2;
        length = i1;
    }
}

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;
    static int line_no;

    static String processLTORG() {
        LitTuple litTuple;
        StringBuilder intermediateStr = new StringBuilder();
        for (int i = poolTable[iPoolTabPtr - 1]; i < littable.size(); i++) {
            litTuple = littable.get(i);
            litTuple.address = lc + "";
            intermediateStr.append(lc).append(" (DL,02)  (C,")
                    .append(litTuple.literal).append(") \n");
            lc++;
        }
        poolTable[iPoolTabPtr] = iLitTabPtr;
        iPoolTabPtr++;
        return intermediateStr.toString();
    }

    static void pass1() throws Exception {
        BufferedReader input = new BufferedReader(new InputStreamReader(new
FileInputStream("C:/Users/malus/Sakshi50/Workshop/input.txt")));
        out_pass1 = new PrintWriter(new FileWriter("output_pass1.txt"), true);
        PrintWriter out_symtable = new PrintWriter(new FileWriter("Symtable.txt"), true);
        PrintWriter out_littable = new PrintWriter(new FileWriter("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);
StringBuilder intermediateStr = new StringBuilder();
if (curTuple.m_class.equalsIgnoreCase("IS")) {
    intermediateStr.append(lc).append(" (")
            .append(curTuple.m_class).append(",")
            .append(curTuple.opcode).append(" ) ");
    lc += curTuple.length;
    intermediateStr.append(processOperands(s_arr[curIndex + 1]));
} else if (curTuple.m_class.equalsIgnoreCase("AD")) {
    if (curTuple.mnemonic.equalsIgnoreCase("START")) {
        intermediateStr.append(lc).append(" (")
                .append(curTuple.m_class).append(",")
                .append(curTuple.opcode).append(") ");
        lc = Integer.parseInt(s_arr[curIndex + 1]);
        intermediateStr.append("(C,").append(s_arr[curIndex + 1]).append(") ");
    } else if (curTuple.mnemonic.equalsIgnoreCase("LTORG")) {
        intermediateStr.append(processLTORG());
    } else if (curTuple.mnemonic.equalsIgnoreCase("END")) {
        intermediateStr.append(lc).append(" (")
                .append(curTuple.m_class).append(",")
                .append(curTuple.opcode).append(")  \n");
            intermediateStr.append(processLTORG());
        }
    } else if (curTuple.m_class.equalsIgnoreCase("DL")) {
        intermediateStr.append(lc).append(" (")
                .append(curTuple.m_class).append(",")
                .append(curTuple.opcode).append(") ");
        if (curTuple.mnemonic.equalsIgnoreCase("DS")) {
            lc += Integer.parseInt(s_arr[curIndex + 1]);
        } else if (curTuple.mnemonic.equalsIgnoreCase("DC")) {
            lc += curTuple.length;
        }
        intermediateStr.append("(C,").append(s_arr[curIndex + 1]).append(") ");
    }
    System.out.println(intermediateStr);

    out_pass1.println(intermediateStr);
}
out_pass1.flush();
out_pass1.close();

System.out.println("====== Symbol Table ======");
SymTuple tuple;
for (Map.Entry<String, SymTuple> entry : symtable.entrySet()) {
    tuple = entry.getValue();
    String tableEntry = tuple.symbol + "\t" + tuple.address;
    out_symtable.println(tableEntry);
    System.out.println(tableEntry);
}
out_symtable.flush();
```

```
        out_symtable.close();

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

    static String processOperands(String operands) {
        StringTokenizer st = new StringTokenizer(operands, ",", false);
        String[] s_arr = new String[st.countTokens()];
        for (int i = 0; i < s_arr.length; i++) {
            s_arr[i] = st.nextToken();
        }
        StringBuilder intermediateStr = new StringBuilder();
        for (String curToken : s_arr) {
            if (curToken.startsWith("=")) {
                StringTokenizer str = new StringTokenizer(curToken, "'", false);
                String[] tokens = new String[str.countTokens()];
                for (int j = 0; j < tokens.length; j++) {
                    tokens[j] = str.nextToken();
                }
                String literal = tokens[1];
                insertIntoTab(literal, "");
                intermediateStr.append("(L,").append(iLitTabPtr - 1).append(")");
        } else if (regAddressTable.containsKey(curToken)) {
            intermediateStr.append("(RG,").append(regAddressTable.get(curToken)).append(") ");
        } else {
            insertIntoSymTab(curToken, "");
            intermediateStr.append("(S,").append(iSymTabPtr - 1).append(")");
        }
    }
    return intermediateStr.toString();
}

static void insertIntoSymTab(String symbol, String address) {
    if (symtable.containsKey(symbol)) {
        // If the symbol already exists, update the address only if the address is empty
        SymTuple s = symtable.get(symbol);
        if (s.address.equals("") && !address.equals("")) {
            s.address = address;
        }
    } else {
        // If the symbol does not exist, insert it with the provided address
        symtable.put(symbol, new SymTuple(symbol, address.equals("") ? lc + "" : address, 1));
    }
    iSymTabPtr++;
}




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

```
    static void initializeTables() throws Exception {
        symtable = new LinkedHashMap<>();
        littable = new ArrayList<>();
        regAddressTable = new HashMap<>();
        MOT = new HashMap<>();
        String s, mnemonic;
        BufferedReader br;
        br = new BufferedReader(new InputStreamReader(new
FileInputStream("C:\\Users\\malus\\Sakshi50\\Workshop\\MOT.txt")));
        while ((s = br.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, " ", false);
            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;
        iPoolTabPtr++;
    }

    public static void main(String[] args) throws Exception {
        System.out.println("Sakshi Malusare 22150");
        initializeTables();
        System.out.println("====== Pass 1 OUTPUT =====\n");
        pass1();
    }
}
```

MOT.TXT:

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

INPUT.TXT:

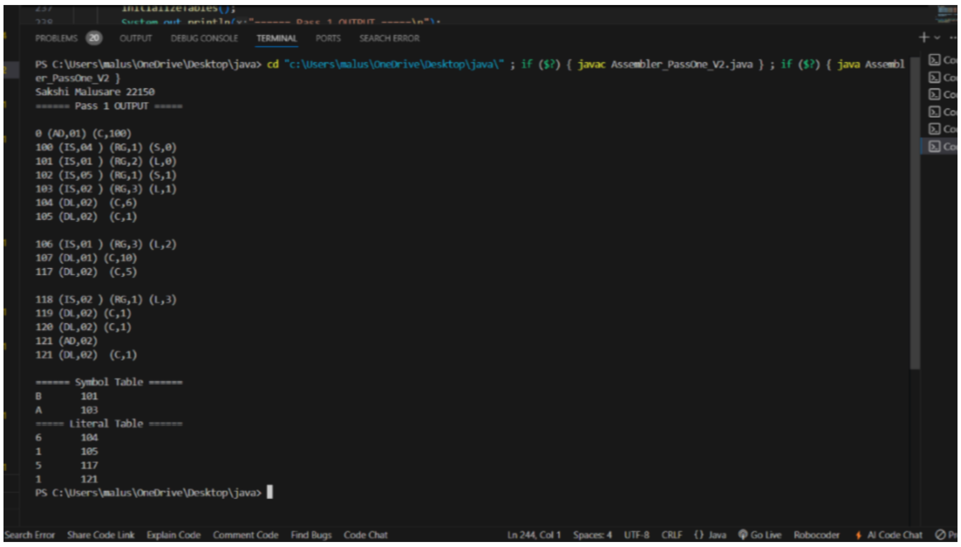
```
START    100
MOVER    AREG,B
ADD      BREG,='6'
MOVEM    AREG,A
SUB      CREG,='1'
LTORG
ADD      DREG,='5'
DS       10
LTORG
SUB      AREG,='1'
```

```
DC      1

DC      1

END
```

OUTPUT:

OUTPUT_PASS1.TXT:

```
0 (AD,01) (C,100)

100 (IS,04 ) (RG,1) (S,0)

101 (IS,01 ) (RG,2) (L,0)

102 (IS,05 ) (RG,1) (S,1)

103 (IS,02 ) (RG,3) (L,1)

104 (DL,02)  (C,6)

105 (DL,02)  (C,1)


106 (IS,01 ) (RG,3) (L,2)

107 (DL,01) (C,10)

117 (DL,02)  (C,5)


118 (IS,02 ) (RG,1) (L,3)

119 (DL,02) (C,1)

120 (DL,02) (C,1)

121 (AD,02)

121 (DL,02)  (C,1)
```



```
PS C:\Users\malus\OneDrive\Desktop\java> cd "c:\Users\malus\OneDrive\Desktop\java\" ; if ($?) { javac Assembler_PassOne_V2.java } ; if ($?) { java Assembl
er_PassOne_V2 }
Sakshi Malusare 22150
====== Pass 1 OUTPUT =====

0 (AD,01) (C,100)
100 (IS,04 ) (RG,1) (S,0)
101 (IS,01 ) (RG,2) (L,0)
102 (IS,05 ) (RG,1) (S,1)
103 (IS,02 ) (RG,3) (L,1)
104 (DL,02)  (C,6)
105 (DL,02)  (C,1)

106 (IS,01 ) (RG,3) (L,2)
107 (DL,01) (C,10)
117 (DL,02)  (C,5)

118 (IS,02 ) (RG,1) (L,3)
119 (DL,02) (C,1)
120 (DL,02) (C,1)
121 (AD,02)
121 (DL,02)  (C,1)

====== Symbol Table ======
B       101
A       103
===== Literal Table ======
6       104
1       105
5       117
1       121
PS C:\Users\malus\OneDrive\Desktop\java> 
```