Name: Shubham Chemate

Roll Number: 31118

Subject: LP-1 (Pass One of Two Pass Assembler)

Code:

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

class ImperativeStatement {
    String opcode;
    String info;

    public ImperativeStatement() {
        opcode = "";
        info = "";
    }

    public ImperativeStatement(String a, String b) {
        opcode = a;
        info = b;
    }
}

class AssemblerDirective {
    String opcode;
    String info;

    public AssemblerDirective() {
        opcode = "";
        info = "";
    }

    public AssemblerDirective(String a, String b) {
        opcode = a;
        info = b;
    }
}

class DeclarativeStatement {
    String opcode;
    String info;

    public DeclarativeStatement() {
        opcode = "";
        info = "";
    }

    public DeclarativeStatement(String a, String b) {
        opcode = a;
        info = b;
    }
}

class SymTabEntry {
```

```
    String symbol;
    String addr;

    public SymTabEntry() {
        symbol = "";
        addr = "";
    }

    public SymTabEntry(String a, String b) {
        symbol = a;
        addr = b;
    }
}

class LitTabEntry {
    String literal;
    String addr;

    public LitTabEntry() {
        literal = "";
        addr = "";
    }

    public LitTabEntry(String a, String b) {
        literal = a;
        addr = b;
    }
}

class Pass1 {

    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new FileReader("input.txt"));
        BufferedWriter bw = new BufferedWriter(new FileWriter("output.txt"));

        //Imperative Statements
        ArrayList < ImperativeStatement > imperativeStatements = new ArrayList
<ImperativeStatement> ();
        imperativeStatements.add(new ImperativeStatement("STOP", "00"));
        imperativeStatements.add(new ImperativeStatement("ADD", "01"));
        imperativeStatements.add(new ImperativeStatement("SUB", "02"));
        imperativeStatements.add(new ImperativeStatement("MULT", "03"));
        imperativeStatements.add(new ImperativeStatement("MOVER", "04"));
        imperativeStatements.add(new ImperativeStatement("MOVEM", "05"));
        imperativeStatements.add(new ImperativeStatement("COMP", "06"));
        imperativeStatements.add(new ImperativeStatement("BC", "07"));
        imperativeStatements.add(new ImperativeStatement("DIV", "08"));
        imperativeStatements.add(new ImperativeStatement("READ", "09"));
        imperativeStatements.add(new ImperativeStatement("PRINT", "10"));

        //Assembler Directives
        ArrayList < AssemblerDirective > assemblerDirective = new
        ArrayList < AssemblerDirective > ();
        assemblerDirective.add(new AssemblerDirective("START", "01"));
        assemblerDirective.add(new AssemblerDirective("END", "02"));
        assemblerDirective.add(new AssemblerDirective("ORIGIN", "03"));
        assemblerDirective.add(new AssemblerDirective("EQU", "04"));
        assemblerDirective.add(new AssemblerDirective("LTORG", "05"));
```

```
        //Declarative Statements
        ArrayList < DeclarativeStatement > declarativeStatement = new ArrayList <
DeclarativeStatement > ();
        declarativeStatement.add(new DeclarativeStatement("DC", "01"));
        declarativeStatement.add(new DeclarativeStatement("DS", "02"));

        ArrayList <SymTabEntry> SYMTAB = new ArrayList <SymTabEntry> ();
        ArrayList <LitTabEntry> LITTAB = new ArrayList <LitTabEntry> ();
        ArrayList <Integer> pooltab = new ArrayList <Integer> ();

        int locationPtr = 0;
        int litPtr = 1;
        int symPtr = 1;
        int pooltabPtr = 1;
        String curLine = null;

        boolean firstLine = true;
        while ((curLine=br.readLine()) != null) {

             if (firstLine) {
                bw.write("AD \t 01 \t");
                String s2 = curLine.split(" ")[2];
                bw.write("C \t " + s2 + "\n");
                locationPtr = Integer.parseInt(s2); //assigning the value to
location  pointer
                firstLine = false;
                continue;
             }

             boolean isLocPtrPr = false;
            String statementType = null; // stores the type of opcode
            boolean memFlag = false; // whether the current symbol has  been
assigned a memory address?
            String firstWord = curLine.split(" |\\,")[0];

            //if the firstWord is label;
            if (firstWord.length() != 0)
                for (SymTabEntry entry : SYMTAB)
                    if (firstWord.equals(entry.symbol)) { // first word found in
symbol table
                            entry.addr = String.valueOf(locationPtr); //assign the
value of current location pointer as the memory address of the symbol
                        memFlag = true;
                    }


            if (firstWord.length() != 0 && memFlag == false) {    //if the first
word is a  label and not present in the symbol table
                    SYMTAB.add(new SymTabEntry(firstWord,
String.valueOf(locationPtr)));
                    symPtr++;
            }

            String secondWord = curLine.split(" |\\,")[1];
            // check for imperative statement
            for (ImperativeStatement entry : imperativeStatements) {
                if (secondWord.equals(entry.opcode)) {
                    bw.write("IS\t" + entry.info + "\t");
                    statementType = "imperative";
```

```
                }
            }

            // check for assembler directive
            for (AssemblerDirective entry : assemblerDirective) {
                if (secondWord.equals(entry.opcode)) {
                    bw.write("AD\t" + entry.info + "\t");
                    statementType = "assemblerDirective";
                }
            }

            // check for declarative statement
            for (DeclarativeStatement entry : declarativeStatement) {
                if (secondWord.equals(entry.opcode)) {
                    bw.write("DL\t" + entry.info + "\t");
                    statementType = "declarative";
                }
            }

//            handling LTORG -> Memory Assignment
            if (secondWord.equals("LTORG")) {
                pooltab.add(pooltabPtr);
                for (LitTabEntry entry: LITTAB) {
                    if (entry.addr == "") { // memory address is not assigned to
the literal
                          entry.addr = String.valueOf(locationPtr);
                            locationPtr++;
                            pooltabPtr++;
                            isLocPtrPr = true;
                            bw.write("\nDL\t01\tC\t" + entry.literal);
                        }
                }
            }

//                handling END
            if (secondWord.equals("END")) {
                pooltab.add(pooltabPtr);
                for (LitTabEntry entry : LITTAB) {
                    if (entry.addr == "") { // memory address is not assigned  to
the literal
                          entry.addr = String.valueOf(locationPtr);
                            locationPtr++;
                            pooltabPtr++;
                            isLocPtrPr = true;
                            bw.write("\nDL\t01\tC\t" + entry.literal);
                        }
                }
            }

//          handling ORIGIN
            if (secondWord.equals("ORIGIN")) {

                String expression = curLine.split(" ")[2];

                if (expression.contains("+") || expression.contains("-")) {
                          String op1 = expression.split("\\+|\\-")[0];
                        String op2 = expression.split("\\+|\\-")[1];
                        int opVal1 = 0, opVal2 = 0;
```

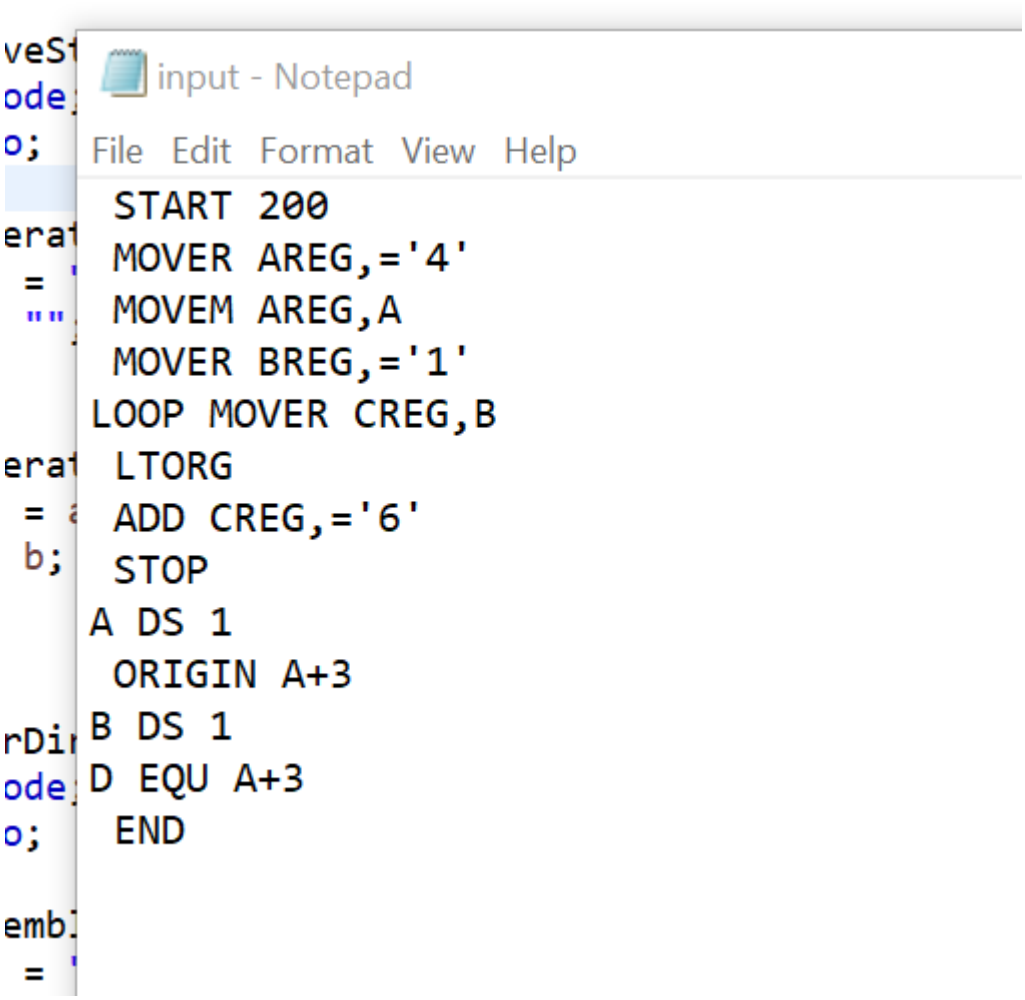
```
                        if (op1.charAt(0) >= '0' && op1.charAt(0) <= '9') // op2
is number
                          opVal1 = Integer.parseInt(op1);
                        else {   // op2 is a symbol
                            for (SymTabEntry entry : SYMTAB)
                                    if (op1.equals(entry.symbol))
                                        opVal1 = Integer.parseInt(entry.addr);
                        }

                        if (op2.charAt(0) >= '0' && op2.charAt(0) <= '9')
                            opVal2 = Integer.parseInt(op2);
                        else {
                            for (SymTabEntry entry : SYMTAB)
                                if (op2.equals(entry.symbol))
                                    opVal2 = Integer.parseInt(entry.addr);
                        }

                        if (expression.contains("+")) locationPtr = opVal1 +
opVal2;
                        else locationPtr = opVal1 - opVal2;

                        isLocPtrPr = true;
                    }
                       else {
                           for (SymTabEntry m: SYMTAB)
                               if (expression.equals(m.symbol)) {
                                   locationPtr = Integer.parseInt(m.addr);
                                   isLocPtrPr = true;
                               }
                       }
                }

            if (secondWord.equals("EQU")) { //assign the memory address of the
first operand to the symbol(first word of the line)
                isLocPtrPr = true;
                String symbol = curLine.split(" ")[0]; //stores the symbol

                for (SymTabEntry entry : SYMTAB)
                    if (symbol.equals(entry.symbol)) {

                        String expression = curLine.split(" ")[2];

                        if (expression.contains("+") || expression.contains("-"))
{

                            String op1 = expression.split("\\+|\\-")[0];
                            String op2 = expression.split("\\+|\\-")[1];

                            int opVal1=0, opVal2=0;

                            if (op1.charAt(0) >= '0' && op1.charAt(0) <= '9') //
op1 is a number
                                 opVal1 = Integer.parseInt(op1);
                            else {      // op1 is a symbol
                                for (SymTabEntry m: SYMTAB)
                                    if (op1.equals(m.symbol))
                                        opVal1 = Integer.parseInt(m.addr);
                            }
```

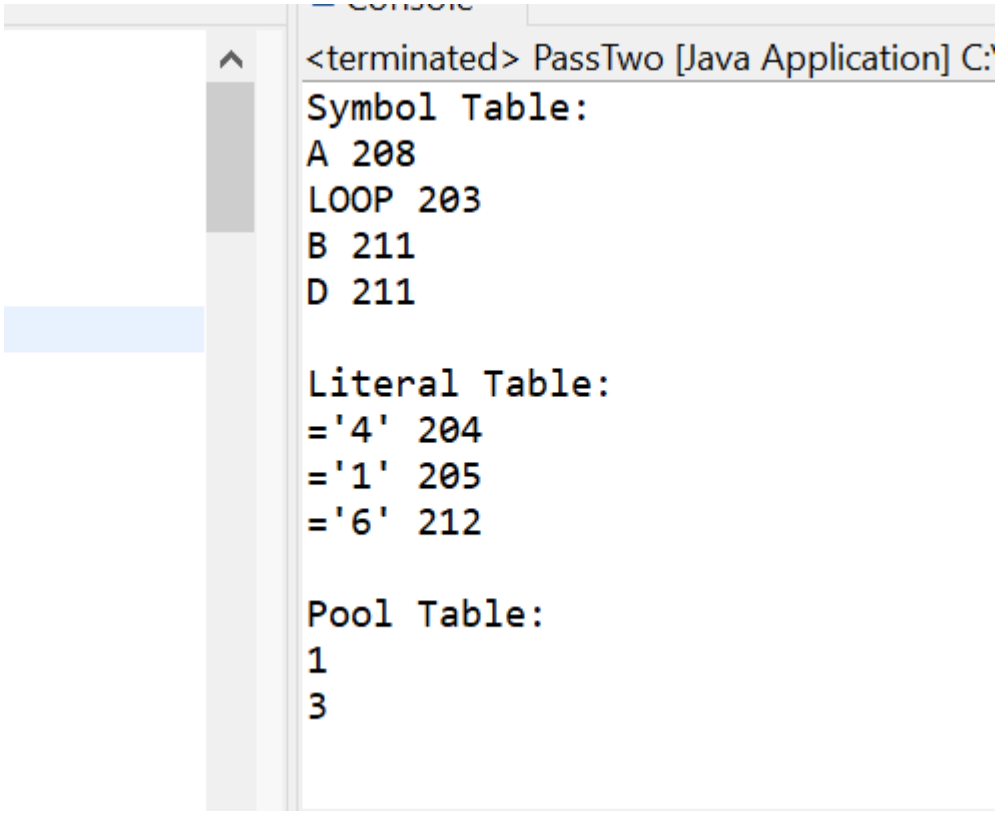
```
                            if (op2.charAt(0) >= '0' && op2.charAt(0) <= '9')
                                opVal2 = Integer.parseInt(op2);
                            else {
                                for (SymTabEntry m : SYMTAB)
                                    if (op2.equals(m.symbol))
                                        opVal2 = Integer.parseInt(m.addr);
                            }

                            if (expression.contains("+"))
                                entry.addr = String.valueOf(opVal1 + opVal2);
                            else entry.addr = String.valueOf(opVal1 - opVal2);
                        }

                        else {
                            for (SymTabEntry m: SYMTAB)
                                if (expression.equals(m.symbol))
                                    entry.addr = m.addr;
                        }
                    }
                }

//              handling DS
                if (secondWord.equals("DS")) {
                    String v = curLine.split(" ")[2];
                    locationPtr += Integer.parseInt(v);
                    isLocPtrPr = true;
                }


//              handling third word of cur line
                if (curLine.split(" |\\,").length > 2) {
                    String thirdWord = curLine.split(" |\\,")[2];

                    if (thirdWord.equals("AREG")) bw.write("1\t");
                    else if (thirdWord.equals("BREG")) bw.write("2\t");
                    else if (thirdWord.equals("CREG")) bw.write("3\t");
                    else if (thirdWord.equals("DREG")) bw.write("4\t");
                    else if (statementType == "declarative") bw.write("C\t" +
thirdWord + "\t");
                    else {
                        if (!thirdWord.contains("+") && !thirdWord.contains("-") &&
!(thirdWord.charAt(0) >= '0' && thirdWord.charAt(0) <= '9')) {
                            boolean isPresent = false;
                            for (SymTabEntry entry: SYMTAB)
                                if (thirdWord.equals(entry.symbol))
                                    isPresent = true;

                            if (!isPresent) {
                                SYMTAB.add(new SymTabEntry(thirdWord, ""));
                                symPtr++;
                            }
                        }
                    }
                }

//              handling fourth word of cur line
                if (curLine.split(" |\\,").length > 3) {
                    String fourthWord = curLine.split(" |\\,")[3]; //second operand
                    if (fourthWord.contains("=")) {// literal
```

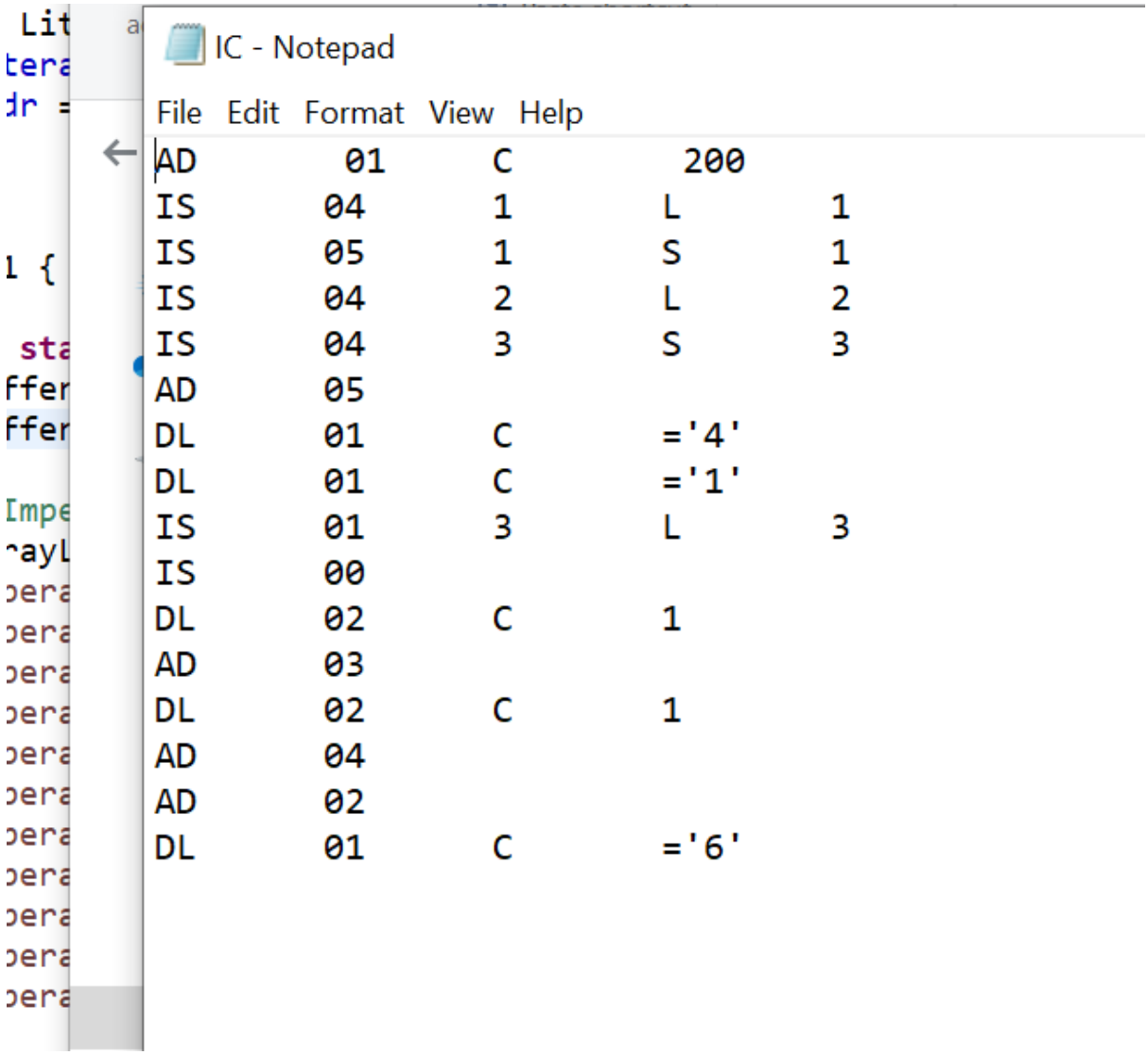
```
                    LITTAB.add(new LitTabEntry(fourthWord, ""));
                    bw.write("L\t" + litPtr + "\t");
                    litPtr++;
                } else { // forward reference symbol
                    if (!fourthWord.contains("+") && !fourthWord.contains("-")) {
                        boolean isPresent = false;
                        int i = 1;
                        for (SymTabEntry m: SYMTAB) {
                            if (fourthWord.equals(m.symbol)) {
                                isPresent = true;
                                bw.write("S\t" + String.valueOf(i) +
                                    "\t");
                            }
                            i++;
                        }
                        if (!isPresent) {
                            SYMTAB.add(new SymTabEntry(fourthWord, ""));
                            bw.write("S\t" + String.valueOf(symPtr) + "\t");
                            symPtr++;
                        }
                    }
                }
            }

            bw.write("\n");
            if (isLocPtrPr == false)
                locationPtr++;
        }

        br.close();
        bw.close();

        bw = new BufferedWriter(new FileWriter("symtab.txt"));
        System.out.println("Symbol Table:");
        for (SymTabEntry entry: SYMTAB) {
            bw.write(entry.symbol + "\t" + entry.addr + "\n");
            System.out.println(entry.symbol + " " + entry.addr);
        }
        bw.close();

        System.out.println("\nLiteral Table:");
        bw = new BufferedWriter(new FileWriter("littab.txt"));
        for (LitTabEntry entry: LITTAB) {
            bw.write(entry.literal + "\t" + entry.addr + "\n");
            System.out.println(entry.literal + " " + entry.addr);
        }
        bw.close();

        System.out.println("\nPool Table:");
        bw = new BufferedWriter(new FileWriter("pooltab.txt"));
        for (Integer item: pooltab) {
            bw.write(item + "\n");
            System.out.println(item);
        }
        bw.close();
    }
}
```

Output:

Assembly Input:




<terminated> PassTwo [Java Application] C:
Symbol Table:
A 208
LOOP 203
B 211
D 211
Literal Table:
='4' 204
='1' 205
='6' 212
Pool Table:
1
3


IC - Notepad
File Edit Format View Help
AD 01 C 200
IS 04 1 L 1
IS 05 1 S 1
IS 04 2 L 2
IS 04 3 S 3
AD 05
DL 01 C ='4'
DL 01 C ='1'
IS 01 3 L 3
IS 00
DL 02 C 1
AD 03
DL 02 C 1
AD 04
AD 02
DL 01 C ='6'