VISHWAKARMA UNIVERSITY
Maximising Human Potential

T. Y. B. Tech Computer Engineering

Student Name	Pranav Dambe (Nikam)
SRN No	202201704
Roll No	68
PRN	2280030506
Division	D(D3)
Subject	System Programming
Year	Third Year

Assignment - 2

QUE 1 :

Design suitable data structures and implement Pass-1 of a two-pass assembler for hypothetical machine. Generate Literal table , Pool Table and Intermediate code in Variant I form. Implementation should consider Sample instructions from each category (AD , IS, DL) in MOT .

Operands can be registers / Literals. Jump statements will have address symbols.

Error handling: e.g. Invalid instruction/register etc. Submit a .pdf file containing input ALP, , Litral table , Pool Table, IC and your Souce code in that sequence.

Consider minimum 2 LTORG statements.

OUTPUT :

1. ALP File :

source.asm

```
1    START 100
2    ADD DREG '5'
3    MOV AREG '6'
4    LTORG
5    ORG 200
6    MOV BREG '9'
7    MUL AREG '3'
8    LTORG
9    MOVER 55
10   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment - 2> cd "e:\TY-LAB\SP
 java Assignment_2 }

-----------------------------------

Litral Table:

1.      '5'     105
2.      '6'     106
3.      '9'     209
4.      '3'     210

-----------------------------------

Pool Table:

1.      1
2.      3

-----------------------------------

Intermediate Table:

0       (AD, 20)        (C, 100)
100     (IS, 01)        (R, 4)  (L, 1)
102     (IS, 89)        (R, 1)  (L, 2)
104     (AD, 19)
106     (AD, 17)        (C, 200)
200     (IS, 89)        (R, 2)  (L, 3)
202     (IS, 03)        (R, 1)  (L, 4)
204     (AD, 19)
206     (IS, 04)        (C, 55)
208     (AD, 21)

-----------------------------------
PS E:\TY-LAB\SP - Lab\Assignment - 2>
```

2. ALP File :

source.asm

```
1    START 100
2    ADD DREG '2'
3    MOV AREG '5'
4    LTORG
5    MOV BREG '8'
6    MUL AREG '15'
7    ADD DREG '22'
8    LTORG
9    MOVER 55
10   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment - 2> cd "e:\TY-LAB\SP
 java Assignment_2 }

_____________________________________

Litral Table:

1.      '2'     105
2.      '5'     106
3.      '8'     118
4.      '15'    119
5.      '22'    120

_____________________________________

Pool Table:

1.      1
2.      3

_____________________________________

Intermediate Table:

0       (AD, 20)        (C, 100)
100     (IS, 01)        (R, 4)  (L, 1)
102     (IS, 89)        (R, 1)  (L, 2)
104     (AD, 19)
106     (IS, 89)        (R, 2)  (L, 3)
108     (IS, 03)        (R, 1)  (L, 4)
110     (IS, 01)        (R, 4)  (L, 5)
112     (AD, 19)
115     (IS, 04)        (C, 55)
117     (AD, 21)

_____________________________________
PS E:\TY-LAB\SP - Lab\Assignment - 2>
```

3. ALP File :

source.asm

```
1    START
2    ADD DREG '2'
3    MOV AREG '5'
4    LTORG
5    MOV BREG '8'
6    MOVEM AREG BREG
7    MOV DREG 50
8    LTORG
9    MUL AREG '10'
10   ADD DREG '25'
11   MOVER 55
12   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment - 2> cd "e:\TY-LAB\SP
 java Assignment_2 }

---------------------------------

Litral Table:

1.      '2'     5
2.      '5'     6
3.      '8'     13
4.      '10'    20
5.      '25'    21

---------------------------------

Pool Table:

1.      1
2.      3
3.      4

---------------------------------

Intermediate Table:

0       (AD, 20)
0       (IS, 01)        (R, 4)  (L, 1)
2       (IS, 89)        (R, 1)  (L, 2)
4       (AD, 19)
6       (IS, 89)        (R, 2)  (L, 3)
8       (IS, 05)        (R, 1)  (R, 2)
10      (IS, 89)        (R, 4)  (C, 50)
12      (AD, 19)
13      (IS, 03)        (R, 1)  (L, 4)
15      (IS, 01)        (R, 4)  (L, 5)
17      (IS, 04)        (C, 55)
19      (AD, 21)

---------------------------------
PS E:\TY-LAB\SP - Lab\Assignment - 2>
```

4. ALP File :

source.asm

```
1    ADD DREG '2'
2    MOV AREG '5'
3    LTORG
4    MOV BREG '8'
5    MOVEM AREG BREG
6    MOV DREG 50
7    MUL AREG '10'
8    END
```

```
PS E:\TY-LAB\SP - Lab\Assignment - 2> cd "e:\TY-LAB\SP
 java Assignment_2 }

Error on line no-1: Program should start with 'START'.

PS E:\TY-LAB\SP - Lab\Assignment - 2>
```

5. ALP File :

source.asm

```
1    START
2    ADD DREG '2'
3    XYZ AREG '5'
4    LTORG
5    MOV BREG '8'
6    MOVEM AREG BREG
7    LTORG
8    MUL AREG '10'
9    ADD DREG '25'
10   MOVER 55
11   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment - 2> cd "e:\TY-LAB\SP
 java Assignment_2 }

Error on line no-3: use of invalid mnemonic.

PS E:\TY-LAB\SP - Lab\Assignment - 2>
```

CODE :

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

class customException extends Exception{
    String exception = null;
    public customException(String str){
        exception = str;
    }

    @Override
    public String toString() {
        return exception;
    }
}

public class Assignment_2 {

    ArrayList<String[]> literalTable = new ArrayList<>();
    ArrayList<String[]> intermediateTable = new ArrayList<>();
    ArrayList<String> literalList = new ArrayList<>();
    ArrayList<Integer> poolTable = new ArrayList<>();

    public int locationCounter = 0;
    public int lineNumber = 0;
    public int index = 0;
    public static boolean showTable = true;

    String[][] motTable = {
        // Imperative Statements
        {"ADD",    "01",    "2",    "IS" }, {"SUB",    "02",    "2",    "IS" }, {"MUL",    "03",    "2",    "IS" },
        {"MOVER",  "04",    "2",    "IS" }, {"MOVEM",  "05",    "2",    "IS" }, {"COMP",   "06",    "2",    "IS" },
        {"BC",     "07",    "2",    "IS" }, {"DIV",    "08",    "2",    "IS" }, {"READ",   "09",    "2",    "IS" },
        {"PRINT",  "10",    "2",    "IS" }, {"MOV",    "89",    "2",    "IS" }, {"PUSH",   "50",    "1",    "IS" },
        {"POP",    "58",    "1",    "IS" }, {"JUMP",   "88",    "2",    "IS" },
        // Declarative Statements
        {"DC",     "01",    "1",    "DS" }, {"DS",     "02",    "n",    "DS" },
        // Assembler Directives
        {"ORG",    "-",     "-",    "AD" }, {"EQU",    "-",     "-",    "AD" }, {"LTORG",  "-",     "-",    "AD" },
        {"START",  "-",     "-",    "AD" }, {"END",    "-",     "-",    "AD" }, {"ENDS",   "-",     "-",    "AD" },
    };

    Map<String, String> Registers = new HashMap<>();

    public Assignment_2() {
        Registers.put("AREG", "(R, 1)");
        Registers.put("BREG", "(R, 2)");
        Registers.put("CREG", "(R, 3)");
        Registers.put("DREG", "(R, 4)");
    }

    public void modifyliteralTable(int index, int LC){
        literalTable.get(index-1)[1] = String.valueOf(LC);
    }

    public void processLiteral()
    {
        for(int n =0; n<literalTable.size(); n++)
        {
```

```
        if(literalList.get(0) == literalTable.get(n)[0])
        {
          index = n;
          poolTable.add(index+1);
          break;
        }
    }
    literalList.clear();
    for(int i=index; i< literalTable.size(); i++){
      locationCounter++;
      literalTable.get(i)[1] = String.valueOf(locationCounter);
    }
}

public boolean isMnemonic(String var) {
    return Arrays.stream(motTable).anyMatch(row -> row[0].equals(var.toUpperCase()));
}

public String getOpcode(String var) {
    String OP_ST = null;
    for (int i = 0; i < motTable.length; i++) {
        if (var.toUpperCase().equals(motTable[i][0])) {
            if (motTable[i][3].equals("IS")) {
                OP_ST = "(IS, " + motTable[i][1] + ")";
                locationCounter += Integer.parseInt(motTable[i][2]);
                break;
            } else if (motTable[i][3].equals("AD")) {
                OP_ST = "(AD, " + (i + 1) + ")";
                break;
            }
        }
    }
    return OP_ST;
}

public int getLiteralIndex(String str){
    for(int i=0; i<literalTable.size(); i++){
        if(str.equals(literalTable.get(i)[0])){
          return i+1;
        }
    }
    return -1;
}

public Boolean isLiteral(String str)
{
    String temp = str.substring(1, str.length()-1);
    if(isInteger(temp) && str.equals("'" + temp + "'")){
        return true;
    }
    return false;
}

public String isRegister(String var) {
    return Registers.get(var.toUpperCase());
}

public static boolean isInteger(String str) {
    if (str == null || str.isEmpty()) {
        return false;
    }
```

```
    try {
        Integer.parseInt(str);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}

public void pass1(String fileName) {
    try (BufferedReader fileReader = new BufferedReader(new FileReader(fileName))) {
        String line;
        while ((line = fileReader.readLine()) != null) {
            lineNumber++;
            processLine(line.trim());
            if(!showTable){
              break;
            }
        }
    } catch (IOException e) {
        System.out.println(e);
    }
}

public void processLine(String line) {
    if (line.isEmpty() || line.startsWith(";")) {
        return;
    }

    String[] words = line.split("\\s+");
    try{
        switch (words.length) {
            case 1:
                processSingleToken(words[0]);
                break;
            case 2:
                processTwowords(words);
                break;
            case 3:
                processThreewords(words);
                break;
        }
    }
    catch(Exception e)
    {
        showTable = false;
        System.out.println("\n"+ e +"\n");
    }
}

void processSingleToken(String word) throws customException{
    String opcode = null;
    int prevLC = locationCounter;
    if(isMnemonic(word)){ // case 1 : Mnemonic
        if((lineNumber == 1) && !word.toUpperCase().equals("START")){
            throw new customException("Error on line no-"+lineNumber+": Program should start with 'START'.");
        }

        if (word.toUpperCase().equals("START")) {
            opcode = getOpcode("START");
            locationCounter = 0;
        } else if(word.toUpperCase().equals("LTORG")){
```

```
            opcode = getOpcode(word);
            processLiteral();

        }else if (word.toUpperCase().equals("END")) {
            opcode = getOpcode("END");
            processLiteral();
        }
        intermediateTable.add(new String[] { String.valueOf(prevLC), opcode });
    }else{
      throw new customException("Error on line no-"+lineNumber+": invalid use of Mnemonic.");
    }
  }

  void processTwowords(String[] words) throws customException{
    String opcode = null, operand1 = null;
    int prevLC = locationCounter;

        if((lineNumber == 1) && !words[0].toUpperCase().equals("START")){
            throw new customException("Error on line no-"+lineNumber+": Program should start with 'START'.");
        }
        if(isMnemonic(words[0])) {// case-2 Mnemonic, Operand
            opcode = getOpcode(words[0]);
            if(words[0].toUpperCase().equals("START") && isInteger(words[1]))
            {
                locationCounter = Integer.parseInt(words[1]);
            }
            if(words[0].toUpperCase().equals("ORG") && isInteger(words[1])){
              operand1 = "(C, " + words[1] + ")";
              locationCounter = Integer.parseInt(words[1]);
            }else if(isRegister(words[1]) != null){
             operand1 = isRegister(words[1]);
            }
            else if(isInteger(words[1])){
             operand1 = "(C, " + words[1] + ")";
            }
            else{
                throw new customException("Error on line no-"+lineNumber+": invalid use of characters");
            }
        }
        else{
            throw new customException("Error on line no-"+lineNumber+": invalid use of Mnemonic.");
        }

    intermediateTable.add(new String[] { String.valueOf(prevLC), opcode, operand1 = (operand1 == null)? "" :
operand1});
  }

  void processThreewords(String[] words) throws customException{
    String opcode = null, operand1 = null, operand2 = null;
    int prevLC = locationCounter;

   if (isMnemonic(words[0])) { // case 3  mnemonic operand1 operand2
            if((lineNumber == 1) && !words[0].toUpperCase().equals("START")){
                throw new customException("Error on line no-"+lineNumber+": Program should start with 'START'.");
            }
            opcode = getOpcode(words[0]);

            if(isRegister(words[1]) != null){
                operand1 = isRegister(words[1]);

                if (isRegister(words[2]) != null) {
```

```
                operand2 = isRegister(words[2]);
            } else if (isInteger(words[2])) {
                operand2 = "(C, " + Integer.parseInt(words[2]) + ")";
            } else if(isLiteral(words[2])){
                literalTable.add(new String[] {words[2], "-"});
                literalList.add(words[2]);
                operand2 = "(L, " + getLiteralIndex(words[2]) + ")";
            }else{
                throw new customException("Error on line no-"+lineNumber+": use of invalid symbol.");
            }
          }
          else{
             throw new customException("Error on line no-"+lineNumber+": use of invalid register.");
          }
        }else{
          throw new customException("Error on line no-"+lineNumber+": use of invalid mnemonic.");
        }
        intermediateTable.add(new String[] { String.valueOf(prevLC), opcode, operand1, operand2 = (operand2 ==
null)? "" : operand2 });
      }

  public void printliteralTable() {
    System.out.println("\n_____");
    System.out.println("\nLitral Table:\n");
    for (int i = 0; i < literalTable.size(); i++) {
      System.out.println((i + 1)+"." + "\t" + literalTable.get(i)[0] + "\t" + literalTable.get(i)[1]);
    }
    System.out.println("_____");
  }

  public void printpoolTable() {
    System.out.println("\nPool Table:\n");
    for (int i = 0; i < poolTable.size(); i++) {
      System.out.println((i + 1)+"." + "\t" + poolTable.get(i));
    }
    System.out.println("                                                    ");
  }
  public void printIntermediateTable() {
    System.out.println("\nIntermediate Table:\n");
    for (String[] row : intermediateTable) {
      System.out.println(String.join("\t", row));
    }
    System.out.println("_____");
  }

  public static void main(String[] args) {
    Assignment_2 Assembler = new Assignment_2();
    Assembler.pass1("source.asm");
    if(showTable){
      Assembler.printliteralTable();
      Assembler.printpoolTable();
      Assembler.printIntermediateTable();
    }
  }
}
```