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 - 1

QUE 1 :

Design suitable data structures and implement Pass-1 of a two-pass assembler for hypothetical machine. Generate symbol table and Intermediate code file. Implementation should consider sample instructions from each category (AD , IS, DL) in MOT .

Operands can be registers / symbols / constants.
Error handling: e.g. Invalid instruction/register, undefined symbol etc.

Submit a single .doc /.pdf file containing input ALP, Symbol table , IC, and source code in this sequence

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_1 {

    ArrayList<String[]> symbolTable = new ArrayList<>();
    ArrayList<String[]> intermediateTable = new ArrayList<>();
    ArrayList<String> symbolList = new ArrayList<>();

    public int locationCounter = 0;
    public int lineNumber = 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",     "1",     "1",    "DS" }, {"DS",     "2",     "n",    "DS" },
        // Assembler Directives
        {"ORG",    "-",     "-",    "AD" }, {"EQU",    "-",     "-",    "AD" }, {"LTORG",   "-",     "-",    "AD" },
        {"START",   "-",     "-",    "AD" }, {"END",    "-",     "-",    "AD" }, {"ENDS",    "-",     "-",    "AD" },
    };

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

    public Assignment_1() {
        Registers.put("AREG", "(R, 1)");
        Registers.put("BREG", "(R, 2)");
        Registers.put("CREG", "(R, 3)");
        Registers.put("DREG", "(R, 4)");
    }

    public void addSymbolTable(String symbol, int LC) {
        if (symbol.endsWith(":")) {
            symbol = symbol.substring(0, symbol.length() - 1);
        }
        symbolTable.add(new String[] { symbol.toUpperCase(), String.valueOf(LC) });
    }

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

```
public boolean alreadyExists(String symbol) {
    boolean isExists = false;
    for (int k = 0; k < symbolTable.size(); k++) {
        if (symbol.toUpperCase().equals(symbolTable.get(k)[0])) {
            isExists = true;
            break;
        }
    }
    return isExists;
}

public boolean isLable(String var) {
    boolean result = false;
    for (int i = 0; i < motTable.length; i++) {
        if (!var.toUpperCase().equals(motTable[i][0]) && var.endsWith(":")) {
            result = true;
        }
        if (var.toUpperCase().equals(motTable[i][0])) {
            result = false;
        }
    }
    return result;
}

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

public int getSymbolIndex(String symbol) {
    for (int i = 0; i < symbolTable.size(); i++) {
        if (symbol.toUpperCase().equals(symbolTable.get(i)[0])) {
            return i + 1;
        }
    }
    return -1;
}

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 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;
            case 4:
                processFourwords(words);
                break;
        }
    }
    catch(Exception e)
    {
        showTable = false;
        System.out.println("\n"+ e +"\n");
    }
}

void processSingleToken(String word) throws customException{
    String opcode = null;
    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("END")) {
            if(symbolList.size() > 0)
            {
                throw new customException("Error: symbols not declared:" + symbolList);
            }
            opcode = getOpcode("END");
        }
        intermediateTable.add(new String[] { String.valueOf(locationCounter), opcode });
    }else{
      throw new customException("Error on line no-"+lineNumber+": invalid use of Mnemonic.");
    }
}

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

    if (isLable(words[0]) && isMnemonic(words[1])) { // case-1 label, Mnemonic

        if((lineNumber == 1) && !words[1].toUpperCase().equals("START")){
            throw new customException("Error on line no-"+lineNumber+": Program should start with 'START'.");
        }

        label = words[0].substring(0, words[0].length() - 1);
        if (!alreadyExists(label)) {
            addSymbolTable(label, locationCounter);
        }else{
            throw new customException("Error on line no-"+lineNumber+": Label already exist.");
        }
        opcode = getOpcode(words[1]);
    } else {

        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 (isRegister(words[1]) != null) {
                operand1 = isRegister(words[1]);
            } else if (isInteger(words[1])) {
                if(words[0].toUpperCase().equals("START"))
                {
                    locationCounter += Integer.parseInt(words[1]);
                }
                operand1 = "(C, " + Integer.parseInt(words[1]) + ")";
            } else {
                if (!alreadyExists(words[1])) {
                    addSymbolTable(words[1], locationCounter);
                    symbolList.add(words[1]);
                }
                operand1 = "(S, " + getSymbolIndex(words[1]) + ")";
            }
        }
        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, label = null, operand1 = null, operand2 = null;
        int prevLC = locationCounter;

        if (isLable(words[0])) { // case-1 : label mnemonic operand1
            label = words[0].substring(0, words[0].length() - 1);

            if (!alreadyExists(label)) {
                addSymbolTable(label, locationCounter);
            }else{
                throw new customException("Error on line no-"+lineNumber+": Label already exist.");
            }

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

                if (isRegister(words[2]) != null) {
                    operand1 = words[2];
                } else if (isInteger(words[2])) {
                    operand1 = "(C, " + Integer.parseInt(words[2]) + ")";
                } else {
                    if (!alreadyExists(words[2])) {
                        addSymbolTable(words[2], locationCounter);
                        symbolList.add(words[2]);
                    }
                    operand1 = "(S, " + getSymbolIndex(words[2]) + ")";
                }
            }else{
                throw new customException("Error on line no-"+lineNumber+": invalid use mnemonic.");
            }

        } else if (!isLable(words[0]) && isMnemonic(words[1]) && isInteger(words[2])) { // case-2  symbol
mnemonic constant

            if((lineNumber == 1) && !words[1].toUpperCase().equals("START")){
                throw new customException("Error on line no-"+lineNumber+": Program should start with 'START'.");
            }

            if (alreadyExists(words[0])) {
                modifySymbolTable(getSymbolIndex(words[0]), locationCounter);
                symbolList.remove(words[0]);
            }else{
              addSymbolTable(words[0], locationCounter);
              symbolList.add(words[0]);
            }
            if (words[1].toUpperCase().equals("DC")) {
                opcode = "(DL, 1)";
                operand1 = "(C, " + Integer.parseInt(words[2]) + ")";
                locationCounter += 1;
            }
            if (words[1].toUpperCase().equals("DS")) {
                opcode = "(DL, 2)";
                operand1 = "(C, " + Integer.parseInt(words[2]) + ")";
                locationCounter += Integer.parseInt(words[2]);
```

```
        }

    } else {
        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 (!isInteger(words[2])) {
                    if (!alreadyExists(words[2])) {
                        addSymbolTable(words[2], locationCounter);
                        symbolList.add(words[2]);
                    }
                    operand2 = "(S, " + getSymbolIndex(words[2]) + ")";
                }
                else{
                    throw new customException("Error on line no-"+lineNumber+": found invalid symbol-" + words[2]);
                }
            }
            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 });
  }

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

    if (isLable(words[0])) { // case 1: label mnemonic operand1 operand2

        label = words[0].substring(0, words[0].length() - 1);
        if (!alreadyExists(label)) {
            addSymbolTable(label, locationCounter);
        }
        else{
            throw new customException("Error on line no-"+lineNumber+": Label already exist.");
        }

        if(isMnemonic(words[1])){
            opcode = getOpcode(words[1]);

            if(isRegister(words[2]) != null)
            {
                operand1 = isRegister(words[2]);
                if (isRegister(words[3]) != null) {
                    operand2 = isRegister(words[3]);
                } else if (isInteger(words[3])) {
```

```
                operand2 = "(C, " + Integer.parseInt(words[3]) + ")";
            } else if (!isInteger(words[3])) {
                if(!alreadyExists(words[3]))
                {
                    addSymbolTable(words[3], locationCounter);
                    symbolList.add(words[3]);
                }
                operand2 = "(S, " + getSymbolIndex(words[3]) + ")";
            }
            else{
                throw new customException("Error on line no-"+lineNumber+": found invalid symbol-" + words[3]);
            }
        }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.");
    }
}
else{
    throw new customException("Error on line no-"+lineNumber+": use of invalid Label.");
}
intermediateTable.add(new String[] { String.valueOf(prevLC), opcode, operand1, operand2 });
}

public void printSymbolTable() {
    System.out.println("\n_____");
    System.out.println("\nSymbol Table:\n");
    for (int i = 0; i < symbolTable.size(); i++) {
        System.out.println((i + 1) + "\t" + symbolTable.get(i)[0] + "\t" + symbolTable.get(i)[1]);
    }
    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  1 Assembler = new Assignment  1();
    Assembler.pass1("source.asm");
    if(showTable){
        Assembler.printSymbolTable();
        Assembler.printIntermediateTable();
    }
}
}
```

OUTPUT :

1. ALP File :

source.asm

```
1    START 200
2    mov AREG CREG
3    LABEL: MOVER AREG 20
4    mov DREG X
5    MOVEM AREG ALPHA
6    MOV BREG 30
7    MOVER AREG 20
8    JUMP LABEL
9    X DS 10
10   ALPHA DC 5
11   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment Codes> cd "e:\TY-LAB\SP
) { java Assignment_1 }

_____________________________

Symbol Table:

1       LABEL   202
2       X       214
3       ALPHA   224

_____________________________

Intermediate Table:

0       (AD, 20)        (C, 200)
200     (IS, 89)        (R, 1)  (R, 3)
202     (IS, 04)        (R, 1)  (C, 20)
204     (IS, 89)        (R, 4)  (S, 2)
206     (IS, 05)        (R, 1)  (S, 3)
208     (IS, 89)        (R, 2)  (C, 30)
210     (IS, 04)        (R, 1)  (C, 20)
212     (IS, 88)        (S, 1)
214     (DL, 02)        (C, 10)
224     (DL, 01)        (C, 5)
225     (AD, 21)

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

2. ALP File :

source.asm

```
1    START 500
2    mov AREG CREG
3    LABEL: MOVER AREG 20
4    mov DREG X
5    MOVEM AREG ALPHA
6    NEXT: MOVER DREG CREG
7    MOV BREG 30
8    MOVER AREG 20
9    JUMP LABEL
10   ADD CREG Y
11   X DS 10
12   ALPHA DC 5
13   MOV DREG 22
14   JUMP NEXT
15   Y DC 15
16   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment Codes> cd "e:\TY-LAB\SP
Assignment_1 }

_____________________________

Symbol Table:

1       LABEL   502
2       X       518
3       ALPHA   528
4       NEXT    508
5       Y       533

_____________________________

Intermediate Table:

0       (AD, 20)        (C, 500)
500     (IS, 89)        (R, 1)  (R, 3)
502     (IS, 04)        (R, 1)  (C, 20)
504     (IS, 89)        (R, 4)  (S, 2)
506     (IS, 05)        (R, 1)  (S, 3)
508     (IS, 04)        (R, 4)  (R, 3)
510     (IS, 89)        (R, 2)  (C, 30)
512     (IS, 04)        (R, 1)  (C, 20)
514     (IS, 88)        (S, 1)
516     (IS, 01)        (R, 3)  (S, 5)
518     (DL, 02)        (C, 10)
528     (DL, 01)        (C, 5)
529     (IS, 89)        (R, 4)  (C, 22)
531     (IS, 88)        (S, 4)
533     (DL, 01)        (C, 15)
534     (AD, 21)

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

3. ALP File :

source.asm

```
START 500
mov AREG CREG
LABEL: MOVER AREG 20
mov DREG X
MOVEM AREG ALPHA
MOVER DREG NUM
MOV BREG 30
MOVER AREG Z
JUMP LABEL
ADD CREG Y
X DS 10
ALPHA DC 5
END
```

```
PS E:\TY-LAB\SP - Lab\Assignment Codes> cd "e:\TY-LAB\SP
Assignment_1 }

Error: symbols are not declared:[NUM, Z, Y]

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

4. ALP File :

source.asm

```
mov AREG CREG
LABEL: MOVER AREG 20
mov DREG X
MOVEM AREG ALPHA
MOVER DREG NUM
MOV BREG 30
MOVER AREG Z
JUMP LABEL
ADD CREG Y
X DS 10
ALPHA DC 5
END
```

```
PS E:\TY-LAB\SP - Lab\Assignment Codes> cd "e:\TY-LAB\SP
) { java Assignment_1 }

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

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

5. ALP File :

source.asm

```
1    START 300
2    mov AREG CREG
3    LABEL: MOVER AREG 20
4    mov DREG X
5    MOVEM AREG ALPHA
6    MOVER DREG NUM
7    LABEL: MOV BREG 30
8    MOVER AREG Z
9    JUMP LABEL
10   ADD CREG Y
11   X DS 10
12   ALPHA DC 5
13   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment Codes> cd "e:\TY-LAB\SP
) { java Assignment_1 }

Error on line no-7: Label already exists.

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

6. ALP File :

source.asm

```
1    START 300
2    mov AREG CREG
3    LABEL: MOVER AREG 20
4    mov DREG X
5    MOVEM AREG ALPHA
6    XYZ DREG NUM
7    MOV BREG 30
8    JUMP LABEL
9    ADD CREG Y
10   X DS 10
11   ALPHA DC 5
12   Y DC 22
13   END
```

```
PS E:\TY-LAB\SP - Lab\Assignment Codes> cd "e:\TY-LAB\SP
) { java Assignment_1 }

Error on line no-6: use of invalid mnemonic.

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