Name: Shardul Vikram Dharmadhikari
Roll no: 8016

ASSIGNMENT No.1

Problem Statement:

Design suitable data structures and implement pass-I of a two-pass assembler for pseudo-machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives.

1.1 Prerequisite:

Basic concepts of Assembler pass 1& pass2(syntax analyzer)

1.2 Learning Objectives:

- To understand data structures to be used in pass I of an assembler.
- To implement pass I of an assembler

1.3 . Theory Concepts:

A language translator bridges an execution gap to machine language of computer system. An assembler is a language translator whose source language is assembly language.
An Assembler is a program that accepts as input an Assembly language program and converts it into machine language.

- **TWO PASS TRANSLATION SCHEME:**

In a 2-pass assembler, the first pass constructs an intermediate representation of the source program for use by the second pass. This representation consists of two main components - data structures like Symbol table, Literal table and processed form of the source program called as intermediate code(IC). This intermediate code is represented by the syntax of Variant –I. Forward reference of a program entity is a reference to the entity, which precedes its definition in the program. While processing a statement containing a forward reference, language processor does not posses all relevant information concerning referenced entity. This creates difficulties in synthesizing the equivalent target statements. This problem can be solved by postponing the generation of target code until more information concerning the entity is available. This also reduces memory requirements of LP and simplifies its organization. This leads to multi-pass model of language processing.

- **DATA STRUCTURES OF A TWO PASS ASSEMBLER:**

Data Structure of Assembler:

a) Operation code table (OPTAB) :This is used for storing mnemonic, operation code and class of instruction
Structure of OPTAB is as follows
b) Data structure updated during translation: Also called as translation time data structure. They are

I. SYMBOL TABLE (SYMTAB) : Ii contains entries such as symbol,it's address and value.

II. LITERAL TABLE (LITTAB) : it contains entries such as literal and it's value.

III . POOL TABLE (POOLTAB): Contains literal number of the starting literal of each literal pool. 1

1.4 Design of a Two Pass Assembler: -

Tasks performed by the passes of two-pass assembler are as follows: **Pass I: -**

Separate the symbol, mnemonic opcode and operand fields.
Determine the storage-required foe every assembly language statement and update the location counter.
Build the symbol table and the literal table.
Construct the intermediate code for every assembly language statement.

Pass II: -

Synthesize the target code by processing the intermediate code generated during pass1
INTERMEDIATE CODE REPRESENTATION
The intermediate code consists of a set of IC units, each IC unit consisting of the following three fields
1. Address
2. Representation of the mnemonic opcode
Representation of operands
Where statement class can be one of IS,DL and AD standing for imperative statement, declaration statement and assembler directive , respectively.
8. Algorithms(procedure) :

PASS 1

- Initialize location counter, entries of all tables as zero.
- Read statements from input file one by one.
- While next statement is not END statement

I. Tokenize or separate out input statement as label,numonic,operand1,operand2 II. If label is present insert label into symbol table.

III. If the statement is LTORG statement processes it by making it's entry into literal table, pool table and allocate memory.
IV. If statement is START or ORIGEN Process location counter accordingly.
V. If an EQU statement, assign value to symbol by correcting entry in symbol table.
VI. For declarative statement update code, size and location counter.
VII. Generate intermediate code.
VIII. Pass this intermediate code to pass -2.

1.5 Conclusion:

Thus, I have studied visual programming and implemented dynamic link library application for arithmetic operation

1.6 Code:

PassOne.java:-

```
package com.company.PassOne;

import com.company.support.InstanceTable;
import com.company.support.TableRow;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;

public class PassOne {
    private int locCounter;
    private int literalTable_pnt;
    private int poolTable_pnt;
    private int symbolIndex;
    private int literalIndex;

    private final LinkedHashMap<String, TableRow> symbolTable;
    private final ArrayList<TableRow> literalTable;
    private final ArrayList<Integer> poolTable;
    private final StringBuilder sb;

    public PassOne() {
        this.locCounter = 0;
        this.literalTable_pnt = 0;
        this.poolTable_pnt = 0;
        this.symbolIndex = 0;
        this.literalIndex = 0;

        symbolTable = new LinkedHashMap<>();
        literalTable = new ArrayList<>();
        poolTable = new ArrayList<>();
        poolTable.add(0);
        sb = new StringBuilder();
    }

    public void parseFile(String s) throws IOException {
        String line;
        BufferedReader br = new BufferedReader(new FileReader(s));
        InstanceTable lookUp = new InstanceTable();

        while((line = br.readLine()) != null) {
            line = line.toUpperCase();
            String[] parts =line.split("\\s+");
```

```
            if(!parts[0].isEmpty()) {
                if(symbolTable.containsKey(parts[0])) {
                    symbolTable.put(parts[0], new TableRow(parts[0],
locCounter, symbolTable.get(parts[0]).getIndex()));
                }else {
                    symbolTable.put(parts[0],new TableRow(parts[0],
locCounter, ++symbolIndex));
                }

sb.append("(S,").append(symbolTable.get(parts[0]).getIndex()).append(") ");
            }
            if(parts[1].equals("START")) {
                locCounter = expr(parts[2]);
                sb.append(lookUp.getCode(parts[1]));
                sb.append("(C,").append(locCounter).append(") ");
            }
            if(parts[1].equals("ORIGIN")) {
                locCounter = expr(parts[2]);
                sb.append(lookUp.getCode(parts[1]));
                sb.append("(C,").append(locCounter).append(") ");
            }
            if(parts[1].equals("LTORG")) {
                int ptr = poolTable.get(poolTable_pnt);
                for(int j = ptr; j<literalTable_pnt; j++) {

sb.append(lookUp.getCode(parts[1])).append(lookUp.getCode("DC")).append("(C
,").append(literalTable.get(j).getSymbol()).append(")");
                    if(j != literalTable_pnt-1) {
                        sb.append("\n");
                    }
                    literalTable.get(j).setAddress(locCounter);
                    ++locCounter;
                }
                poolTable_pnt++;
                poolTable.add(literalTable_pnt);
            }
            if(parts[1].equals("EQU"))
            {
                int loc=expr(parts[2]);
                //below If conditions are optional as no IC is generated
for them
                if(parts[2].contains("+"))
                {
                    String[] splits =parts[2].split("\\+");

sb.append(lookUp.getCode(parts[1])).append("(S,").append(symbolTable.get(sp
lits[0]).getIndex()).append(")+").append(splits[1]);
                }
                else if(parts[2].contains("-"))
                {
                    String[] splits =parts[2].split("\\-");

sb.append(lookUp.getCode(parts[1])).append("(S,").append(symbolTable.get(sp
lits[0]).getIndex()).append(")-").append(splits[1]);
                }
                else
                {

sb.append(lookUp.getCode(parts[1])).append("(C,").append(parts[2]);
                }
```

```
                if(symbolTable.containsKey(parts[0]))
                    symbolTable.put(parts[0], new
TableRow(parts[0],loc,symbolTable.get(parts[0]).getIndex())) ;
                else
                    symbolTable.put(parts[0], new
TableRow(parts[0],loc,++symbolIndex));
                ++locCounter;
            }
            if(parts[1].equals("DS")) {

sb.append(lookUp.getCode(parts[1])).append("(C,").append(parts[2]).append("
) ");
                locCounter = locCounter + Integer.parseInt(parts[2]);
            }
            if(parts[1].equals("DC")) {
                ++locCounter;
                int constant = Integer.parseInt(parts[2].replace("'",""));

sb.append(lookUp.getCode(parts[1])).append("(C,").append(constant).append("
) ");
            }
            if(lookUp.getType(parts[1]).equals("IS")) {
                sb.append(lookUp.getCode(parts[1]));
                int j = 2;
                while(j < parts.length) {
                    parts[j]=parts[j].replace(",", "");
                    if(lookUp.getType(parts[j]).equals("RG")) {
                        sb.append(lookUp.getCode(parts[j]));
                    }
                    else {
                        if(parts[j].contains("=")) {
                            parts[j] =
parts[j].replace("=","").replace("'","");
                            literalTable.add(new TableRow(parts[j],-
1,++literalIndex));
                            ++literalTable_pnt;
                            sb.append("(L,").append(literalIndex).append(")
");
                        }else if(symbolTable.containsKey(parts[j])) {
                            int idx=symbolTable.get(parts[j]).getIndex();
                            sb.append("(S,").append(idx).append(") ");
                        }
                        else {
                            symbolTable.put(parts[j],new
TableRow(parts[j],-1,++symbolIndex));
                            int idx=symbolTable.get(parts[j]).getIndex();
                            sb.append("(S,").append(idx).append(") ");
                        }
                    }
                    j++;
                }
                ++locCounter;
            }
            if(parts[1].equals("END")) {
                int ptr = poolTable.get(poolTable_pnt);
                if(ptr != literalTable_pnt) {
                    for(int j = ptr; j<literalTable_pnt; j++) {

sb.append(lookUp.getCode("LTORG")).append(lookUp.getCode("DC")).append("(C,
").append(literalTable.get(j).getSymbol()).append(")\n");
                        literalTable.get(j).setAddress(locCounter);
```

```
                        ++locCounter;
                    }
                    poolTable_pnt++;
                    poolTable.add(literalTable_pnt);
                }
                sb.append(lookUp.getCode(parts[1]));
            }
            sb.append("\n");
        }
    }

    public String getIC() {
        return sb.toString();
    }

    public String getSymbolTable() {
        StringBuilder temp = new StringBuilder();
        temp.append("\n**********************SYMBOL
TABLE**********************\n");
        temp.append("Index\tSymbol\tAddress\n");
        for(Map.Entry<String,TableRow> mapElement : symbolTable.entrySet())
{
            TableRow tableRow = mapElement.getValue();
            String symbol = tableRow.getSymbol();
            int address = tableRow.getAddress();
            int index = tableRow.getIndex();

temp.append(index).append("\t\t").append(symbol).append("\t\t").append(addr
ess).append("\n");
        }
        return temp.toString();
    }

    public String getLiteralTable() {
        StringBuilder temp = new StringBuilder();
        temp.append("\n**********************LITERAL
TABLE**********************\n");
        temp.append("Index\tSymbol\tAddress\n");
        for(TableRow tableRow : literalTable) {
            String symbol = tableRow.getSymbol();
            int address = tableRow.getAddress();
            int index = tableRow.getIndex();

temp.append(index).append("\t\t").append(symbol).append("\t\t").append(addr
ess).append("\n");
        }
        return temp.toString();
    }

    public String getPoolTable() {
        StringBuilder temp = new StringBuilder();
        temp.append("\n**********************POOL
TABLE**********************\n");
        temp.append("Index\n");
        for(Integer i : poolTable) {
            temp.append(i).append("\n");
        }
        return temp.toString();
    }

    private int expr(String str)
```

```
    {
        int temp;
        if(str.contains("+"))
        {
            String[] splits =str.split("\\+");
            temp =
symbolTable.get(splits[0]).getAddress()+Integer.parseInt(splits[1]);
        }
        else if(str.contains("-"))
        {
            String[] splits =str.split("\\-");
            temp = symbolTable.get(splits[0]).getAddress()-
(Integer.parseInt(splits[1]));
        }else
        {
            temp=Integer.parseInt(str);
        }
        return temp;
    }
}
```

TableRow.java:-

```
package com.company.support;

public class TableRow {
    private final String symbol;
    private int address;
    private final int index;

    public TableRow(String symbol, int address, int index) {
        this.symbol = symbol;
        this.address = address;
        this.index = index;
    }

    public String getSymbol() {
        return symbol;
    }

    public int getAddress() {
        return address;
    }

    public void setAddress(int address) {
        this.address = address;
    }

    public int getIndex() {
        return index;
    }

}
```

InstanceTable.java:-

```
package com.company.support;

import java.util.HashMap;

public class InstanceTable {
```

```
private static HashMap<String,String> AD;
private static HashMap<String,String> RG;
private static HashMap<String,String> IS;
private static HashMap<String,String> CC;
private static HashMap<String,String> DL;

public InstanceTable() {
    //Initialize HashMaps
    AD = new HashMap<>();
    RG = new HashMap<>();
    IS = new HashMap<>();
    CC = new HashMap<>();
    DL = new HashMap<>();

    //Enter data in the classes
    IS.put("STOP","01");
    IS.put("ADD","02");
    IS.put("SUB","03");
    IS.put("MUL","04");
    IS.put("MOVER","05");
    IS.put("MOVEM","06");
    IS.put("COMB","07");
    IS.put("BC","08");
    IS.put("DIV","09");
    IS.put("READ","10");
    IS.put("PRINT","11");
    IS.put("STORE","12");

    AD.put("START","01");
    AD.put("END","02");
    AD.put("ORIGIN","03");
    AD.put("EQU","04");
    AD.put("LTORG","05");

    DL.put("DS","01");
    DL.put("DC","02");

    RG.put("AREG","01");
    RG.put("BREG","02");
    RG.put("CREG","03");
    RG.put("DREG","04");

    CC.put("EQ","01");
    CC.put("LT","02");
    CC.put("GT","03");
    CC.put("LE","04");
    CC.put("GE","05");
    CC.put("NE","06");
```

```
    }

    public String getType(String s) {
        s = s.toUpperCase();

        if(IS.containsKey(s)) {
            return "IS";
        }else if(AD.containsKey(s)) {
            return "AD";
        }else if(DL.containsKey(s)) {
            return "DL";
        }else if(RG.containsKey(s)) {
            return "RG";
        }else if(CC.containsKey(s)) {
            return "CC";
        }else {
            return "";
        }
    }

    public String getCode(String s) {
        s = s.toUpperCase();

        if(IS.containsKey(s)) {
            return "(IS,"+IS.get(s)+") ";
        }else if(AD.containsKey(s)) {
            return "(AD,"+AD.get(s)+") ";
        }else if(DL.containsKey(s)) {
            return "(DL,"+DL.get(s)+") ";
        }else if(RG.containsKey(s)) {
            return "(RG,"+RG.get(s)+") ";
        }else if(CC.containsKey(s)){
            return "(RG,"+CC.get(s)+") ";
        }else {
            return "-01";
        }
    }
}
```

Main.java:-

```
package com.company;

import com.company.PassOne.PassOne;

import java.io.IOException;

public class Main {
```

```
  public static void main(String[] args) throws IOException {
    PassOne passOne = new PassOne();
    passOne.parseFile("input2.asm");

    System.out.println(passOne.getIC());
    System.out.println(passOne.getSymbolTable());
    System.out.println(passOne.getLiteralTable());
    System.out.println(passOne.getPoolTable());
  }

}
```

INPUT CODE-

```
START 200
    MOVER AREG, A
L1  ADD BREG, B
    PRINT B
    READ ='5'
    PRINT ='15'
    LTORG
    MUL CREG, X
    SUB BREG, ='5'
    DIV CREG LP
LP  STORE X
    ORIGIN L1+30
A   DS 3
    LTORG
    MOVER BREG, X
X   DC 2
B   DS 2
    END
```

OUTPUT-

**********IC CODE**********
(AD,01) (C,200)
(IS,05) (RG,01) (S,1)
(S,2) (IS,02) (RG,02) (S,3)
(IS,11) (S,3)
(IS,10) (L,1)
(IS,11) (L,2)
(AD,05) (DL,02) (C,5)
(AD,05) (DL,02) (C,15)
(IS,04) (RG,03) (S,4)
(IS,03) (RG,02) (L,3)
(IS,09) (RG,03) (S,5)
(S,5) (IS,12) (S,4)
(AD,03) (C,231)
(S,1) (DL,01) (C,3)

(AD,05) (DL,02) (C,5)
(IS,05) (RG,02) (S,4)
(S,4) (DL,02) (C,2)
(S,3) (DL,01) (C,2)
(AD,02)

**********SYMBOL TABLE**********

Index	Symbol	Address
1	A	231
2	L1	201
3	B	237
4	X	236
5	LP	210

**********LITERAL TABLE**********

Index	Symbol	Address
1	5	205
2	15	206
3	5	234

**********POOL TABLE**********

Index
0
2
3

Process finished with exit code 0