SPOSL A1

Title: Pass I of Two Pass Assembler

Problem Statement:
Design a suitable data structures & implement pass-I of a two pass assembler for pseudo machine in Java using object oriented feature.

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

Outcomes:
- Parse & tokenize the pseudo assembly source code.
- generate intermidiate code
- designing of symbol table, literal table & pool table

Software & Hardware Req.
Java SE 11, IDE
Ubuntu 20.04
Processor: intel i5, 64bit
i/o devices

Scanned with CamScanner

Page No.
Date: / /

Theory:
Assembler is a program which converts assembly language instructions into machine language form. A two pass assembler takes two scans of source code to produce the machine code from assembly language program.

Pass I tasks:
- Assign addresses to statements in program.
- Save values assigned to all labels for use in pass II
- Perform processing of assembler directives.

Description using set theory:

S = { I, O, T, D, Succ, fail}
I : Input O: Output T: Type (I | II)
D: Data Structure

O = { Sf, Mf}
Sf: Source code file
Mf: Mnemonic code file

O = { St, Lt, Ic}
St : Symbol table
Lt : Literal table
Ic : Intermidiate code file.

Scanned with CamScanner

Page No.
Date: / /

St = {N, A}
N: Name of Symbol.
A: Address of Symbol.

Lt = {N, A}
N: Name of literal
A: Address of literal

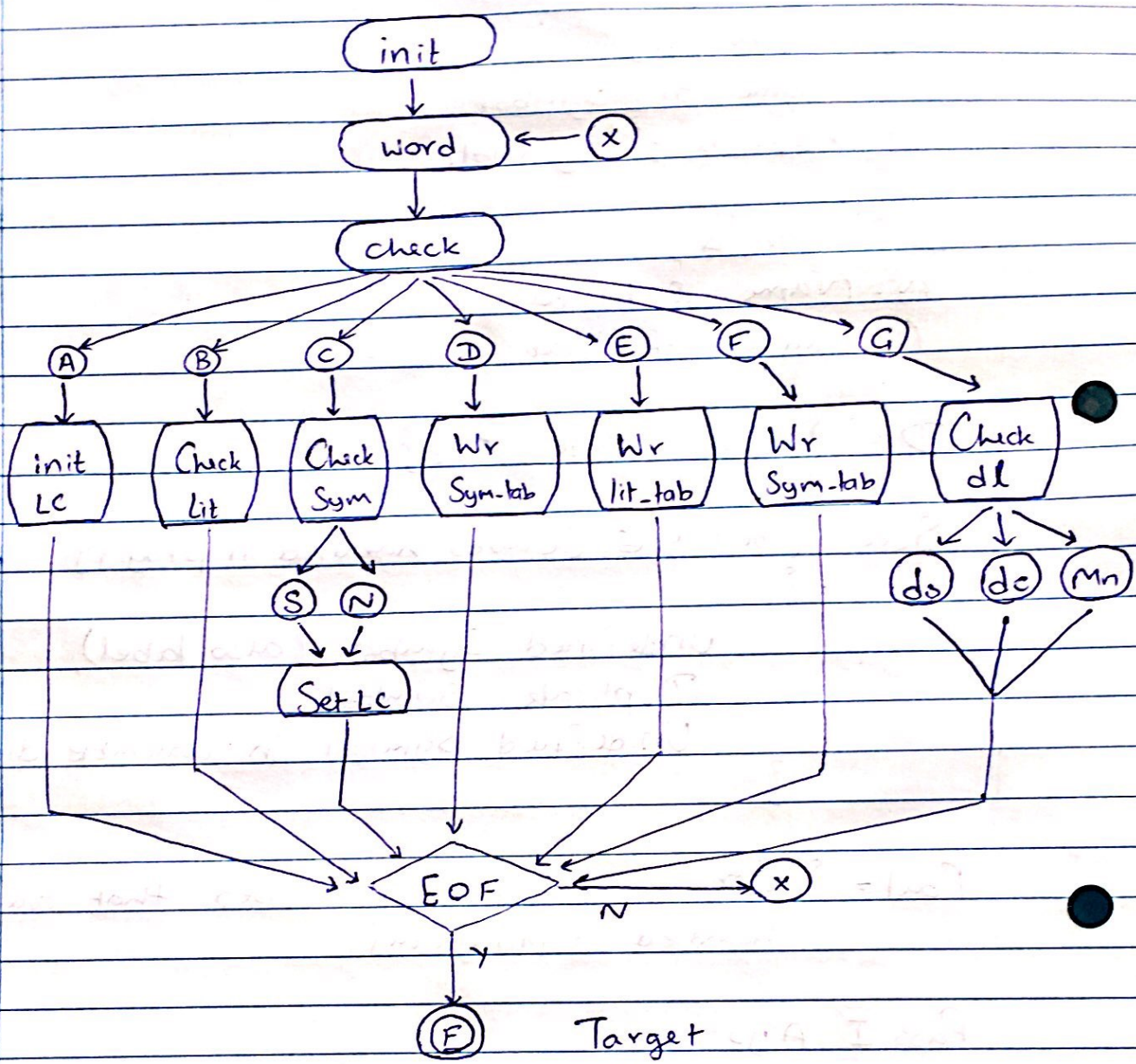
D = {Map, Array, File}

Succ = {x | x ∈ {cases handled in prog.}}
= {
undefined Symbol (also label),
Duplicate Symbol,
Undefined Symbol in assemble directives,
}

Fail = {x | x is a set of all cases that are not handled in program}

Pass I Algo:
1) Initialize location Counter
2) Read input file line by line
3) while next statement != NULL Repeat
4) tokenize line.
5) If label is present insert into Symbol table
6) If mnemonic is LOTRG make it's entry in all 2 tables
7) If mnemonic is START process LC accordingly
8) If mnemonic is EQU correct entry in symbol table
9) For declarative statement update code, size & location counter.

Scanned with CamScanner



- Test Cases

Description	Expected o/p	Result
1) Input all valid mnemonics	Replace all mnemonics with correct opcodes	Success.
2) Input the inst. & operands in valid format	Generate valid intermidiate code for Pass II	Success

Scanned with CamScanner

Page No.
Date: / /

Conclusion:

In this assignment we have implemented pass I of two pass assembler for generating intermidiate code & data structures, symbol table & pool table using Java Collection framework.

Scanned with CamScanner

```java
package Assignment_A01.src;

import java.io.*;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;

public class Assembler {
private final Map<String, Integer> symbolTable;
private final Map<String, Integer> literalTable;
private final ArrayList<Integer> poolTable;
private String file;
private String code;
private int locationCounter;
private int poolPointer;

public Assembler(){
this(null);
}

public Assembler(String file) {
this.file = file;
symbolTable = new LinkedHashMap<>();
literalTable = new LinkedHashMap<>();
poolTable = new ArrayList<>();
poolPointer = 0;
code = "";
}

public Map<String, Integer> getSymbolTable() {
return symbolTable;
}

public Map<String, Integer> getLiteralTable() {
return literalTable;
}

public ArrayList<Integer> getPoolTable() {
return poolTable;
}

public String getCode() {
return code;
}

public void setFile(String file) {
this.file = file;
}

private void initializeLocationCounter() throws Exception {
BufferedReader reader = new BufferedReader(new FileReader(file));
String line = reader.readLine();
String[] tokens = line.split("\\s+");
locationCounter = Integer.parseInt(tokens[2]);
reader.close();
```

```
}

private void interpret(String line) throws Exception {
String[] tokens = line.split("\\s+");
String label = tokens[0];
String instruction = tokens[1].toUpperCase();
String instructionType = InstructionTable.getInstructionType(instruction);

if(!label.isBlank()){
symbolTable.put(label,locationCounter);
}

switch (instructionType){
case "AD":
if(instruction.equals("START")){
code = code + String.format("(AD,01)\t(C,%s)\n",locationCounter);
}
else if(instruction.equals("END")){
poolTable.add(poolPointer+1);
updateLiteralTable(true);
code = code + "(AD,02)\n";
}
else if(instruction.equals("ORIGIN")){
String expression = tokens[2];
if(expression.contains("+")){
String[] parts = expression.split("\\+");
code = code + String.format("(AD,03)\t(S,%02d)+%s\n",getTableIndex(parts[0],symbolTable),parts[1]);
}
else if(expression.contains("-")){
String[] parts = expression.split("-");
code = code + String.format("(AD,03)\t(S,%02d)-%s\n",getTableIndex(parts[0],symbolTable),parts[1]);
}
else {
try{
Integer.parseInt(expression);
code = code + String.format("(AD,03)\t(C,%s)\n",expression);
} catch (NumberFormatException e){
code = code + String.format("(AD,03)\t(S,%02d)\n",getTableIndex(expression,symbolTable));
}
}
locationCounter = evaluate(expression);
}
else if(instruction.equals("EQU")){
String expression = tokens[2];
if(expression.contains("+")){
String[] parts = expression.split("\\+");
code = code + String.format("(AD,04)\t(S,%02d)+%s\n",getTableIndex(parts[0],symbolTable),parts[1]);
}
else if(expression.contains("-")){
String[] parts = expression.split("-");
code = code + String.format("(AD,04)\t(S,%02d)-%s\n",getTableIndex(parts[0],symbolTable),parts[1]);
}
else {
try{
Integer.parseInt(expression);
code = code + String.format("(AD,04)\t(C,%s)\n",expression);
} catch (NumberFormatException e){
```

```
code = code + String.format("(AD,04)\t(S,%02d)\n",getTableIndex(expression,symbolTable));
}
}
symbolTable.put(label,evaluate(expression));
}
else if(instruction.equals("LTORG")){
poolTable.add(poolPointer+1);
updateLiteralTable(false);
}
break;
case "DL":
code = code + String.format("(DL,%02d)\t",InstructionTable.getOpCode(instruction));
if(instruction.equals("DC")){
int constant = Integer.parseInt(tokens[2].replace("'",""));
code = code + String.format("(C,%s)\n",constant);
locationCounter++;
}
else if(instruction.equals("DS")){
int size = Integer.parseInt(tokens[2]);
code = code + String.format("(C,%s)\n",size);
locationCounter = locationCounter+size;
}
break;
case "IS":
code = code + String.format("(IS,%02d)\t",InstructionTable.getOpCode(instruction));
for(int i=2; i<tokens.length; i++){
tokens[i] = tokens[i].replace("'","").replace(",","");
if(InstructionTable.getInstructionType(tokens[i]).equals("RG")){
code = code + String.format("(RG,%02d)\t",InstructionTable.getOpCode(tokens[i]));
}else if(InstructionTable.getInstructionType(tokens[i]).equals("CC")){
code = code + String.format("(CC,%02d)\t",InstructionTable.getOpCode(tokens[i]));
} else {
if(tokens[i].contains("=")){
tokens[i] = tokens[i].replace("=","");
literalTable.put(tokens[i],-1);
code = code + String.format("(L,%02d)\t",getTableIndex(tokens[i],literalTable));
}
else if(symbolTable.containsKey(tokens[i])){
code = code + String.format("(S,%02d)\t",getTableIndex(tokens[i],symbolTable));
}
else {
symbolTable.put(tokens[i],-1);
code = code + String.format("(S,%02d)\t",getTableIndex(tokens[i],symbolTable));
}
}
}
code = code + "\n";
locationCounter++;
break;
default:
throw new Exception(instruction+":invalid instruction type");
}
}

public void passOne() throws Exception {
if(file == null)
throw new FileNotFoundException("no input file");
```

```
initializeLocationCounter();

String line;
BufferedReader bufferedReader = new BufferedReader(new FileReader(file));

while ((line=bufferedReader.readLine())!=null){
interpret(line);
}
bufferedReader.close();
generateOutput();
}

private void generateOutput() throws Exception{
BufferedWriter bufferedWriter = null;
int index = 0;

bufferedWriter = new BufferedWriter(new FileWriter("Assignment_A01/lib/INTERMEDIATE_CODE.txt"));
bufferedWriter.write(code);
bufferedWriter.close();

index = 1;
bufferedWriter = new BufferedWriter(new FileWriter("Assignment_A01/lib/SYMBOL_TABLE.txt"));
for(String key: symbolTable.keySet()) {
bufferedWriter.write(index+"\t"+key+"\t"+symbolTable.get(key)+"\n");
index++;
}
bufferedWriter.close();

index = 1;
bufferedWriter = new BufferedWriter(new FileWriter("Assignment_A01/lib/LITERAL_TABLE.txt"));
for(String key: literalTable.keySet()){
bufferedWriter.write(index+"\t"+key+"\t"+literalTable.get(key)+"\n");
index++;
}
bufferedWriter.close();

index = 1;
bufferedWriter = new BufferedWriter(new FileWriter("Assignment_A01/lib/POOL_TABLE.txt"));
for(Integer pointer: poolTable) {
bufferedWriter.write(index+"\t#"+pointer+"\n");
index++;
}
bufferedWriter.close();

}

private void updateLiteralTable(boolean end){
int index = 0;
for(String literal : literalTable.keySet()){
if(poolPointer == index){
literalTable.put(literal, locationCounter);
if(!end)
code = code + String.format("(AD,05)\t(DL,02)\t(C,%s)\n",literal);
else
```

```
code = code + String.format("(DL,02)\t(C,%s)\n",literal);
poolPointer++;
locationCounter++;
}
index++;
}
}



private int evaluate(String expression){
if(expression.contains("+")){
String[] tokens = expression.split("\\+");
return symbolTable.get(tokens[0]) + Integer.parseInt(tokens[1]);
}
else if(expression.contains("-")){
String[] tokens = expression.split("-");
return symbolTable.get(tokens[0]) - Integer.parseInt(tokens[1]);
}
else {
try{
return Integer.parseInt(expression);
} catch (NumberFormatException e){
return symbolTable.get(expression);
}
}
}

private int getTableIndex(String entry, Map<String, Integer> table){
int index = 0;
for(String key : table.keySet()){
if(key.equals(entry))
return index+1;
index++;
}
return -1;
}

}


package Assignment_A01.src;

import java.util.HashMap;

public class InstructionTable {
public static HashMap<String, Integer> AD, RG, IS, CC, DL;

static {
AD = new HashMap<>();
RG = new HashMap<>();
IS = new HashMap<>();
CC = new HashMap<>();
DL = new HashMap<>();

DL.put("DC", 1);
```

```
DL.put("DS", 2);

IS.put("STOP",0);
IS.put("ADD",1);
IS.put("SUB",2);
IS.put("MULT",3);
IS.put("MOVER",4);
IS.put("MOVEM",5);
IS.put("COMP",6);
IS.put("BC",7);
IS.put("DIV",8);
IS.put("READ",9);
IS.put("PRINT",10);

CC.put("LT",1);
CC.put("LE",2);
CC.put("EQ",3);
CC.put("GT",4);
CC.put("GE",5);
CC.put("ANY",6);

AD.put("START",1);
AD.put("END",2);
AD.put("ORIGIN",3);
AD.put("EQU",4);
AD.put("LTORG",5);

RG.put("AREG",1);
RG.put("BREG",2);
RG.put("CREG",3);
RG.put("DREG",4);
}

public static String getInstructionType(String instruction) {
instruction = instruction.toUpperCase();
if(AD.containsKey(instruction)) return "AD";
if(RG.containsKey(instruction)) return "RG";
if(IS.containsKey(instruction)) return "IS";
if(CC.containsKey(instruction)) return "CC";
if(DL.containsKey(instruction))return "DL";
return "NULL";
}

public static int getOpCode(String instruction) {
instruction = instruction.toUpperCase();
if(AD.containsKey(instruction)) return AD.get(instruction);
if(RG.containsKey(instruction)) return RG.get(instruction);
if(IS.containsKey(instruction)) return IS.get(instruction);
if(CC.containsKey(instruction)) return CC.get(instruction);
if(DL.containsKey(instruction)) return DL.get(instruction);
return -1;
}

}
```

```java
package Assignment_A01.src;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Scanner;

public class Main {
private static final Scanner scanner = new Scanner(System.in);

public static void main(String[] args) {
System.out.println("START");
String line;
BufferedReader bufferedReader = null;
Assembler assembler = new Assembler();

System.out.print("File Path: ");
String file = scanner.nextLine();
assembler.setFile(file);

try {
assembler.passOne();

bufferedReader = new BufferedReader(new FileReader("Assignment_A01/lib/INTERMEDIATE_CODE.txt"));
System.out.println("\nIntermediate Code:");
while ((line=bufferedReader.readLine())!=null)
System.out.println(line);
bufferedReader.close();

bufferedReader = new BufferedReader(new FileReader("Assignment_A01/lib/SYMBOL_TABLE.txt"));
System.out.println("\nSymbol Table:");
while ((line=bufferedReader.readLine())!=null)
System.out.println(line);
bufferedReader.close();

bufferedReader = new BufferedReader(new FileReader("Assignment_A01/lib/LITERAL_TABLE.txt"));
System.out.println("\nLiteral Table:");
while ((line=bufferedReader.readLine())!=null)
System.out.println(line);
bufferedReader.close();

bufferedReader = new BufferedReader(new FileReader("Assignment_A01/lib/POOL_TABLE.txt"));
System.out.println("\nPool Table:");
while ((line=bufferedReader.readLine())!=null)
System.out.println(line);
bufferedReader.close();

} catch (Exception e) {
e.printStackTrace();
}
}
}
```

OUTPUT :

input1.asm

```
        START       200
A       DS      2
B       ADD AREG    =10
        LTORG
        MULT BREG   A
        END
```