EXPERIMENT NO : 1

**Problem Statement:** Design suitable Data structures and implement Pass-I and Pass- II of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.

# PASS 1 Assembler: Source Code:

package com.spos; import java.util.Scanner;

import java.util.StringTokenizer; import java.io.\*;

public class PASSoneassembler {

static Scanner in = new Scanner(System.in); static String is[] = {

"STOP","ADD","SUB","MULT","MOVER","MOVEM","COMP","BC","DIV","READ ","PRINT"};

static String ad[] = { "START","END","ORIGIN","EQU","LTORG"};

static String dl[] = { "DC" , "DS" };

static String cc[] = { "LT","LE","EQ","GT","GE","ANY"};

static int symcounter =0 ; static int litcounter =0 ;

static String sym[][] = new String[100][2]; static String lit[][] = new String[100][2]; static String ptab[][] = new String[100][2];

public static void main(String args[]) throws Exception{

int locate=0; int litcount = 0;

File file = new File("input\_assembler.asm"); File file1 = new File("output.asm");

BufferedReader reader = new BufferedReader(new FileReader(file)); BufferedWriter writer= new BufferedWriter(new FileWriter(file1));

String st;

String y,prev=null; int stp = 0;

String ans; int k=0;

String buffer = "";

while((st = reader.readLine())!=null){ int isflag=0;

k++;

StringTokenizer splitted = new StringTokenizer(st); ans="";

while(splitted.hasMoreTokens()){ y = splitted.nextToken();

if(y.equals("START")){

locate = Integer.parseInt(splitted.nextToken()); ans="(AD,01)(C,"+locate+")";

break;

}

else{

if(searchis(y)){ if(y.equals("STOP")){

stp=1;

}

ans+="(IS,"+Integer.toString(indexis(y))+")"; isflag=1;

locate+=1;

}

else if(searchad(y)){

if(y.equals("LTORG")){ locate+=litcount;

ans="(AD,05)\n"; while(litcount>0){

lit[litcounter-litcount][1] = Integer.toString(locate-litcount); int len=lit[litcounter-litcount][0].length();

String temp = lit[litcounter-litcount][0].substring(2,len-1); ans+="(DL,01)(C,"+temp+")";

litcount--; if(litcount!=0)

ans+="\n";

}

}

if(y.equals("ORIGIN")){ y=splitted.nextToken(); String[] words = y.split("\\+");

int location = Integer.parseInt(sym[indexsym(words[0])][1]); locate=location+Integer.parseInt(words[1]);

ans="(AD,03)(S,"+Integer.toString(indexsym(words[0])+1)+")+"+words[1];

}

if(y.equals("END") && litcount!=0){ locate+=litcount;

ans="(AD,02)\n"; while(litcount>0){

lit[litcounter-litcount][1] = Integer.toString(locate-litcount); int len=lit[litcounter-litcount][0].length();

String temp = lit[litcounter-litcount][0].substring(2,len-1); ans+="(DL,01)(C,"+temp+")\n";

litcount--;

if(litcount!=0) ans+="\n";

}

}

if(y.equals("EQU")){

int temp = indexsym(splitted.nextToken()); y=prev;

sym[indexsym(y)][1]= sym[temp][1]; ans="";

}

}

else if(searchdl(y)){ if(y.equals("DC")){

ans=""; ans+="(DL,1)(C,"+splitted.nextToken()+")";

}

if(y.equals("DS")){ ans="";

ans+="(DL,2)(C,"+splitted.nextToken()+")";

}

locate+=1;

}

else{

prev=y;

char[] x=y.toCharArray(); if(x[0]=='='){

int z=litcounter; ans+="(L,"+(z+1)+")";

lit[litcounter++][0]=y; litcount++;

}

else if(y.equals("AREG")){ ans+="(1)";

}

else if(y.equals("BREG")){ ans+="(2)";

}

else if(y.equals("CREG")){ ans+="(3)";

}

else if(y.equals("DREG")){ ans+="(4)";

}

else if(searchcc(y)){ ans+="("+Integer.toString(indexcc(y)+1)+")";

}

else{

if(!searchsym(y) && isflag==0 && stp==0){ sym[symcounter][0] = y;

sym[symcounter++][1] = Integer.toString(locate);

ans+="(S,"+Integer.toString(indexsym(y)+1)+")"; if(splitted.hasMoreTokens())

ans="";

}

else if(!searchsym(y) && isflag==1 && stp==0 ){//if instruction has passed on the line then only add the symbol not the address

sym[symcounter++][0] = y; ans+="(S,"+Integer.toString(indexsym(y)+1)+")";

}

else if(searchsym(y) && isflag==0){ sym[indexsym(y)][1]= Integer.toString(locate); ans+="(S,"+Integer.toString(indexsym(y)+1)+")"; if(splitted.hasMoreTokens())

ans=""; prev=y;

}

else{

if(!splitted.hasMoreTokens()) ans+="(S,"+Integer.toString(indexsym(y)+1)+")";

continue;

}

}

}

}

}

ans=ans+"\n"; buffer+=ans;

}

System.out.println(buffer); System.out.println("Symbol Table : "); for(int i=0;i<symcounter;i++){

System.out.print(sym[i][0]+"\t"); System.out.println(sym[i][1]);

}

System.out.println("Literal Table : "); for(int i=0;i<litcounter;i++){

System.out.print(lit[i][0]+"\t");

System.out.println(lit[i][1]);

}

writer.write(buffer); reader.close(); writer.close();

}

public static boolean searchis(String s){ boolean flag = false;

int i=0; while(i<11){

if(is[i].equals(s)){ flag=true; break;

} i++;

}

return flag;

}

public static boolean searchad(String s){ boolean flag = false;

int i=0; while(i<5){

if(ad[i].equals(s)){ flag=true;

break;

} i++;

}

return flag;

}

public static boolean searchdl(String s){ boolean flag = false;

int i=0; while(i<2){

if(dl[i].equals(s)){ flag=true; break;

} i++;

}

return flag;

}

public static boolean searchsym(String s){

boolean flag = s.equals("BREG") || s.equals("AREG") || s.equals("CREG") || s.equals("DREG") || s.equals(",") ||s.equals("LE") || s.equals("LT") ||s.equals("ANY")

||s.equals("EQ") ||s.equals("GT") ||s.equals("GE"); int i=0;

while(i<symcounter ){ if(sym[i][0].equals(s)){

flag=true; break;

} i++;

}

return flag;

}

public static boolean searchcc(String s){ boolean flag = false;

int i=0; while(i<6){

if(cc[i].equals(s)){ flag=true; break;

} i++;

}

return flag;

}

public static int indexsym(String s){ int c = 0;

int i=0; while(i<symcounter){

if(sym[i][0].equals(s)){ c=i;

break;

} i++;

}

return i;

}

public static int indexlit(String s){ int c = 0;

int i=0; while(i<litcounter){

if(lit[i][0].equals(s)){ c=i;

break;

} i++;

}

return i;

}

public static int indexis(String s){ int i=0;

while(i<11){ if(is[i].equals(s)){

break;

} i++;

}

return i;

}

public static int indexad(String s){ int i=0;

while(i<5){ if(ad[i].equals(s)){

break;

} i++;

}

return i;

}

public static int indexdl(String s){ int i=0;

while(i<2){ if(dl[i].equals(s)){

break;

} i++;

}

return i;

}

public static int indexcc(String s){ int i=0;

while(i<6){ if(cc[i].equals(s)){

break;

} i++;

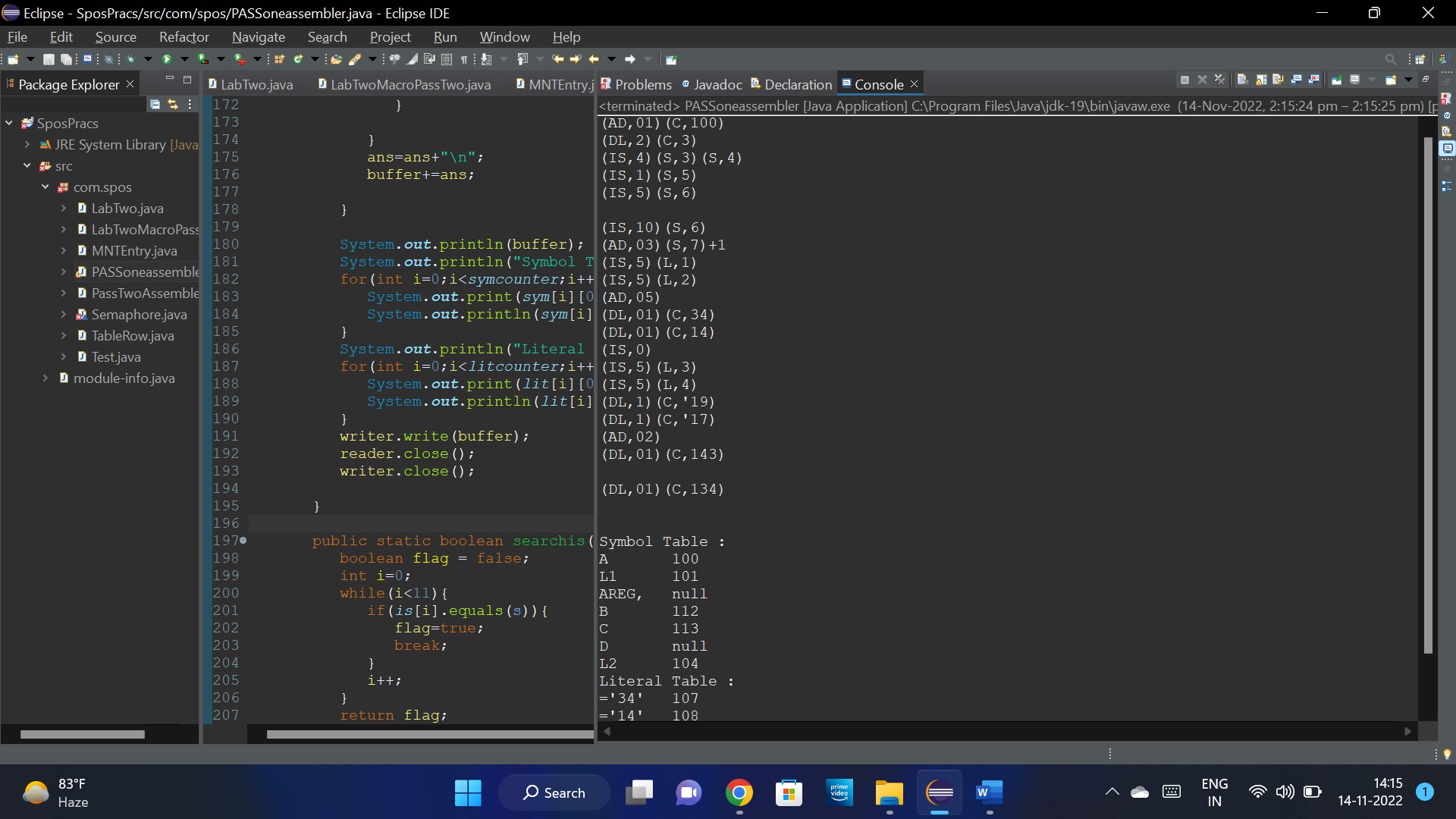
}

return i;

}

}

**OUTPUT:**



# PASS 2 Assembler : Source Code:

package com.spos;

import java.io.BufferedReader; import java.io.BufferedWriter; import java.io.FileReader; import java.io.FileWriter; import java.util.ArrayList;

public class PassTwoAssembler { ArrayList<TableRow> SYMTAB,LITTAB;

public PassTwoAssembler()

{

SYMTAB=new ArrayList<>(); LITTAB=new ArrayList<>();

}

public static void main(String[] args) {

PassTwoAssembler pass2=new PassTwoAssembler();

try {

pass2.generateCode("IC.txt");

} catch (Exception e) {

// **TODO** Auto-generated catch block e.printStackTrace();

}

}

public void readtables()

{

BufferedReader br; String line;

try

{

br=new BufferedReader(new FileReader("SYMTAB.txt")); while((line=br.readLine())!=null)

{

String parts[]=line.split("\\s+"); SYMTAB.add(new TableRow(parts[1],

Integer.*parseInt*(parts[2]),Integer.*parseInt*(parts[0]) ));

}

br.close();

br=new BufferedReader(new FileReader("LITTAB.txt")); while((line=br.readLine())!=null)

{

String parts[]=line.split("\\s+");

LITTAB.add(new TableRow(parts[1], Integer.*parseInt*(parts[2]),Integer.*parseInt*(parts[0])));

}

br.close();

}

catch (Exception e) {

System.***out***.println(e.getMessage());

}

}

public void generateCode(String filename) throws Exception

{

readtables();

BufferedReader br=new BufferedReader(new FileReader(filename));

BufferedWriter bw=new BufferedWriter(new FileWriter("PASS2.txt")); String line,code;

while((line=br.readLine())!=null)

{

String parts[]=line.split("\\s+"); if(parts[0].contains("AD")||parts[0].contains("DL,02"))

{

bw.write("\n"); System.***out***.println("\n"); continue;

}

else if(parts.length==2)

{

if(parts[0].contains("DL")) //DC INSTR

{

parts[0]=parts[0].replaceAll("[^0-9]", ""); if(Integer.*parseInt*(parts[0])==1)

{

int constant=Integer.*parseInt*(parts[1].replaceAll("[^0-9]", "")); code="00\t0\t"+String.*format*("%03d", constant)+"\n"; bw.write(code);

System.***out***.println(code);

}

}

else if(parts[0].contains("IS"))

{

int opcode=Integer.*parseInt*(parts[0].replaceAll("[^0-9]", "")); if(opcode==10)

{

if(parts[1].contains("S"))

{

9]", ""));

int symIndex=Integer.*parseInt*(parts[1].replaceAll("[^0-

code=String.*format*("%02d",

opcode)+"\t0\t"+String.*format*("%03d", SYMTAB.get(symIndex-1).getAddress())+"\n";

bw.write(code); System.***out***.println(code);

}

else if(parts[1].contains("L"))

{

9]", ""));

int symIndex=Integer.*parseInt*(parts[1].replaceAll("[^0-

code=String.*format*("%02d",

opcode)+"\t0\t"+String.*format*("%03d", LITTAB.get(symIndex-1).getAddress())+"\n";

bw.write(code); System.***out***.println(code);

}

}

}

}

else if(parts.length==1 && parts[0].contains("IS"))

{

int opcode=Integer.*parseInt*(parts[0].replaceAll("[^0-9]", "")); code=String.*format*("%02d", opcode)+"\t0\t"+String.*format*("%03d", 0)+"\n"; bw.write(code);

System.***out***.println(code);

}

else if(parts[0].contains("IS") && parts.length==3) //All OTHER IS INSTR

{

int opcode= Integer.*parseInt*(parts[0].replaceAll("[^0-9]", "")); int regcode=Integer.*parseInt*(parts[1]);

if(parts[2].contains("S"))

{

int symIndex=Integer.*parseInt*(parts[2].replaceAll("[^0-9]", "")); code=String.*format*("%02d",

opcode)+"\t"+regcode+"\t"+String.*format*("%03d", SYMTAB.get(symIndex-1).getAddress())+"\n"; bw.write(code);

System.***out***.println(code);

}

else if(parts[2].contains("L"))

{

int symIndex=Integer.*parseInt*(parts[2].replaceAll("[^0-9]", "")); code=String.*format*("%02d",

opcode)+"\t"+regcode+"\t"+String.*format*("%03d", LITTAB.get(symIndex-1).getAddress())+"\n";

bw.write(code); System.***out***.println(code);

}

}

}

bw.close();

br.close();

}

}

**OUTPUT:**

