**Name:** Sanika Kendre

**Class:** TE Computer A

**Batch:** C

**Roll No.:**21CO059

**Problem Statement:**

**Design suitable Data structures and implement Pass-I of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives.**

**Program:**

import java.io.\*;

import java.util.ArrayList;

import java.util.Hashtable;

class MnemonicTable {

//m\_class specifies class of the mnemonic such as IS, DL, or AD

String mnemonic, m\_class, opcode;

int length;

MnemonicTable() {}

MnemonicTable(String s1, String s2, int s3) {

mnemonic = s1;

opcode = s2;

length = s3;

}

}

public class PassOne {

Hashtable<String,MnemonicTable> is=new Hashtable<>();

ArrayList<String>symtab=new ArrayList<>();

ArrayList<Integer> symaddr=new ArrayList<>();

ArrayList<String>littab=new ArrayList<>();

ArrayList<Integer> litaddr=new ArrayList<>();

ArrayList<Integer>pooltab=new ArrayList<>();

int LC=0;

public void createIS() {

MnemonicTable m=new MnemonicTable("STOP","00", 0);

is.put("STOP",m);

m=new MnemonicTable("ADD","01", 0);

is.put("ADD",m);

m=new MnemonicTable("SUB","02", 0);

is.put("SUB",m);

m=new MnemonicTable("MULT","03", 0);

is.put("MULT",m);

m=new MnemonicTable("MOVER","04", 0);

is.put("MOVER",m);

m=new MnemonicTable("MOVEM","05", 0);

is.put("MOVEM",m);

m=new MnemonicTable("COMP","06", 0);

is.put("COMP",m);

m=new MnemonicTable("BC","07", 0);

is.put("BC",m);

m=new MnemonicTable("DIV","08", 0);

is.put("DIV",m);

m=new MnemonicTable("READ","09", 0);

is.put("READ",m);

m=new MnemonicTable("PRINT","10", 0);

is.put("PRINT",m);

}

public void generateIC() throws Exception {

BufferedWriter wr=new BufferedWriter(new FileWriter("intercode.txt"));

BufferedReader br=new BufferedReader(new FileReader("input1.txt"));

String line=" ";

pooltab.add(0, 0);

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

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

if(split[0].equals("START")) {

LC=Integer.parseInt(split[1]);

wr.write("(AD,01)(C,"+split[1]+") \n");

}

else if(split[0].equals("ORIGIN")) {

if(split[2].contains("+") || split[2].contains("-")) {

LC=getAddress(split[2]);

wr.write("(AD,03)(C,"+split[1]+") \n");

}

else {

LC=symaddr.get(symtab.indexOf(split[2]));

}

}

else if(split[0].equals("EQU")) {

int addr=0;

if(split[2].contains("+") || split[2].contains("-")) {

addr=getAddress(split[2]);

wr.write("(AD,04)(C,"+split[1]+") \n");

}

else {

addr=symaddr.get(symtab.indexOf(split[2]));

}

}

else if(split[0].equals("LTORG") || split[0].equals("END")) {

if(litaddr.contains(0)) {

for(int i=pooltab.get(pooltab.size()-1);i<littab.size();i++) {

if(litaddr.get(i)==0) {

litaddr.remove(i);

litaddr.add(i, LC);

LC++;

}

}

if(!split[0].equals("END")) {

pooltab.add(littab.size());

wr.write("(AD,05) \n");

}

else

wr.write("(AD,02) \n");

}

}

else if (split[0].contains("END"))

{

wr.write("(AD,02) \n");

}

else if(split[1].contains("DS")) {

LC+=Integer.parseInt(split[2]);

if(symtab.contains(split[0]))

{symaddr.set(0,LC);}

wr.write("(DL,01) (C,"+split[2]+") \n");

}

else if(split[1].equals("DC")) {

LC++;

if(symtab.contains(split[0]))

{symaddr.set(1,LC);}

wr.write("(DL,02) (C,"+split[2].replace("'",

"").replace("'", "")+") \n");

}

else if(is.containsKey(split[0])) {

// System.out.println(split[0]);

wr.write("(IS,"+is.get(split[0]).opcode+") ");

if(split.length>=2 && split[1]!=null) {

String[] reg=split[1].split(",");

System.out.println(reg[1]);

if(reg[0].equals("AREG")) {

wr.write("(RG,1) ");

}

else if(reg[0].equals("BREG")) {

wr.write("(RG,2) ");

}

else if(reg[0].equals("CREG")) {

wr.write("(RG,3) ");

}

else if(reg[0].equals("DREG")) {

wr.write("(RG,4) ");

}

else {

}

if(reg.length>=1) {

if(reg[1].contains("="))

{

// String[] norm=reg[1].split("=");

String norm=reg[1].substring(2,3);

if(!littab.contains(norm)) {

littab.add(norm);

litaddr.add(0);

wr.write("(L,"+littab.indexOf(norm)+") \n");

}

else {

wr.write("(L,"+littab.indexOf(norm)+") \n");

}

}

else{

symtab.add(reg[1]);

symaddr.add(0);

wr.write("(S,"+symtab.indexOf(reg[1])+") \n");

}

}

}

}

LC++;

}

wr.flush();

wr.close();

br.close();

BufferedWriter br1=new BufferedWriter(new FileWriter("sym.txt"));

BufferedWriter br2=new BufferedWriter(new FileWriter("lit.txt"));

BufferedWriter br3=new BufferedWriter(new FileWriter("pool.txt"));

//System.out.println("symtab");

for(int i=0;i<symtab.size();i++)

br1.write(symtab.get(i)+" "+symaddr.get(i)+"\n");

//System.out.println("littab");

for(int i=0;i<littab.size();i++)

br2.write(littab.get(i)+" "+litaddr.get(i)+"\n");

for(int i=0;i<pooltab.size();i++)

br3.write(pooltab.get(i)+"\n");

br1.flush();

br2.flush();

br3.flush();

br1.close();

br2.close();

br3.close();

}

private int getAddress(String string) {

int temp=0;

if(string.contains("+")) {

String sp[]=string.split("\\+");

int ad=symaddr.get(symtab.indexOf(sp[0]));

temp=ad+Integer.parseInt(sp[1]);

}

else if(string.contains("-")) {

String sp[]=string.split("\\-");

int ad=symaddr.get(symtab.indexOf(sp[0]));

temp=ad-Integer.parseInt(sp[1]);

}

return temp;

}

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

PassOne p=new PassOne();

p.createIS();

p.generateIC();

}

}

**Output:**

**input1.txt:**

START 100

MOVER AREG,B

ADD BREG,='6'

MOVEM AREG,A

SUB CREG,='1'

LTORG

ADD DREG,='5'

A DS 10

LTORG

B DC 1

END

**intercode.txt:**

(AD,01)(C,100)

(IS,04) (RG,1) (S,0)

(IS,01) (RG,2) (L,0)

(IS,05) (RG,1) (S,1)

(IS,02) (RG,3) (L,1)

(AD,05)

(IS,01) (RG,4) (L,2)

(DL,01) (C,10)

(AD,05)

(DL,02) (C,1)

(AD,02)

**lit.txt**

6 105

1 106

5 120

**sym.txt**

B 119

A 123

**pool.txt**

0

2

3