SPCC – Experiment 9 – Pass 1 Assembler

Code :

```
import java.util.*;
import java.io.*;
class Tuple
{       String mnemonic, bin_opcode, type;
        int length;
        Tuple() {}
        Tuple(String s1, String s2, String s3, String s4)
        {       mnemonic = s1;
        bin_opcode = s2;
        length = Integer.parseInt(s3);
        type = s4;          }          }
class SymTuple
{       String symbol, ra;
        int value, length;
        SymTuple(String s1, int i1, int i2, String s2)
        {       symbol = s1;
                value = i1;
                length = i2;
                ra = s2;   }          }
class LitTuple
{       String literal, ra;
        int value, length;
        LitTuple() {}
        LitTuple(String s1, int i1, int i2, String s2)
        {       literal = s1;
                value = i1;
                length = i2;
                ra = s2;   }          }
class PassoneAssembler
{       static int lc;
        static List<Tuple> mot;
        static List<String> pot;
        static List<SymTuple> symtable;
        static List<LitTuple> littable;
        static List<Integer> lclist;
        static Map<Integer, Integer> basetable;

        static PrintWriter out_pass1;
        static int line_no;
        public static void main(String args[]) throws Exception
        {       initializeTables();
                System.out.println("====== PASS 1 ======\n");
                pass1();

static void pass1() throws Exception
{       BufferedReader input = new BufferedReader(new InputStreamReader(new
        FileInputStream("input.txt")));
        out_pass1 = new PrintWriter(new FileWriter("output_pass1.txt"), true);
        PrintWriter out_symtable = new PrintWriter(new FileWriter("out_symtable.txt"),true);
        PrintWriter out_littable = new PrintWriter(new FileWriter("out_littable.txt"),true);
        String s;
        while((s = input.readLine()) != null)
        {       StringTokenizer st = new StringTokenizer(s, " ", false);
                String s_arr[] = new String[st.countTokens()];
```

```
                for(int i=0 ; i < s_arr.length ; i++)
                        s_arr[i] = st.nextToken();
                if(searchPot1(s_arr) == false)
                {       searchMot1(s_arr);
                        out_pass1.println(s);       }
                        lclist.add(lc);       }
                int j;
                String output = new String();
                System.out.println("Symbol Table:");
                System.out.println("Symbol Value Length R/A");
                for(SymTuple i : symtable)
                {       output = i.symbol;
                        for(j=i.symbol.length() ; j < 10 ; j++)
                                output += " ";
                        output += i.value;
                        for(j=new Integer(i.value).toString().length() ; j < 7 ; j++)
                                output += " ";
                        output += i.length + " " + i.ra;
                        System.out.println(output);
                        out_symtable.println(output);       }
                System.out.println("\nLiteral Table:");
                System.out.println("Literal Value Length R/A");
                for(LitTuple i : littable)
                {       output = i.literal;
                        for(j=i.literal.length() ; j < 10 ; j++)
                                output += " ";
                        output += i.value;
                        for(j=new Integer(i.value).toString().length() ; j < 7 ; j++)
                                output += " ";
                        output += i.length + " " + i.ra;
                        System.out.println(output);
                        out_littable.println(output);                   }       }       }
static boolean searchPot1(String[] s)
{       int i = 0;
        int l = 0;
        int potval = 0;
        if(s.length == 3)
                i = 1;
        s = tokenizeOperands(s);
        if(s[i].equalsIgnoreCase("DS") || s[i].equalsIgnoreCase("DC"))
                potval = 1;
        if(s[i].equalsIgnoreCase("EQU"))
                potval = 2;
        if(s[i].equalsIgnoreCase("START"))
                potval = 3;
        if(s[i].equalsIgnoreCase("LTORG"))
                potval = 4;
        if(s[i].equalsIgnoreCase("END"))
                potval = 5;
        switch(potval)
        {       case 1:
                        String x = s[i+1];
                        int index = x.indexOf("F");
                        if(i == 1)
                                symtable.add(new SymTuple(s[0], lc, 4, "R"));
                        if(index != 0)
                        {
                                l = Integer.parseInt(x.substring(0, x.length()-1));
```

```
                                        l *= 4;
                                }
                                else
                                {
                                        for(int j=i+1 ; j<s.length ; j++)
                                                l += 4;
                                }
                                lc += l;
                                return true;
        case 2:
                                if(!s[2].equals("*"))
                                        symtable.add(new SymTuple(s[0], Integer.parseInt(s[2]), 1,"A"));
                                else
                                        symtable.add(new SymTuple(s[0], lc, 1, "R"));
                                return true;
        case 3:
                                symtable.add(new SymTuple(s[0], Integer.parseInt(s[2]), 1, "R"));
                                return true;
        case 4:
                                ltorg(false);
                                return true;
        case 5:
                                ltorg(true);
                                return true;
        }
        return false;
}
static void searchMot1(String[] s)
{       Tuple t = new Tuple();
        int i = 0;
        if(s.length == 3)
                i = 1;
        s = tokenizeOperands(s);
        for(int j=i+1 ; j < s.length ; j++)
        {       if(s[j].startsWith("="))
                        littable.add(new LitTuple(s[j].substring(1, s[j].length()), -1, 4,"R"));        }
        if((i == 1) && (!s[0].equalsIgnoreCase("END")))
                symtable.add(new SymTuple(s[0], lc, 4, "R"));
        for(Tuple x : mot)
                if(s[i].equals(x.mnemonic))
                {       t = x;
                        break;    }
        lc += t.length;     }
static void ltorg(boolean isEnd)
{       Iterator<LitTuple> itr = littable.iterator();
        LitTuple lt = new LitTuple();
        boolean isBroken = false;
        while(itr.hasNext())
        {       lt = itr.next();
                if(lt.value == -1)
                {       isBroken = true;
                        break;    }         }
        if(!isBroken)
                return;
        if(!isEnd)
                while(lc%8 != 0)
                        lc++;
        lt.value = lc;
```

```
            lc += 4;
            while(itr.hasNext())
            {           lt = itr.next();
                                lt.value = lc;
                                lc += 4;   }           }
static String[] tokenizeOperands(String[] s)
{           List<String> temp = new LinkedList<>();
            for(int j=0 ; j<s.length-1 ; j++)
                        temp.add(s[j]);
            StringTokenizer st = new StringTokenizer(s[s.length-1], " ,", false);
            while(st.hasMoreTokens())
                        temp.add(st.nextToken());
            s = temp.toArray(new String[0]);
            return s;               }
static void initializeTables() throws Exception
{           symtable = new LinkedList<>();
            littable = new LinkedList<>();
            lclist = new ArrayList<>();
            basetable = new HashMap<>();
            mot = new LinkedList<>();
            pot = new LinkedList<>();
            String s;
            BufferedReader br;
            br = new BufferedReader(new InputStreamReader(new FileInputStream("mot.txt")));
            while((s = br.readLine()) != null)
            {           StringTokenizer st = new StringTokenizer(s, " ", false);
                        mot.add(new Tuple(st.nextToken(), st.nextToken(), st.nextToken(),
                        st.nextToken()));  }
            br = new BufferedReader(new InputStreamReader(new FileInputStream("pot.txt")));
            while((s = br.readLine()) != null)
                        pot.add(s);
            Collections.sort(pot);        }           }
```

MOT.TXT

```
LA 01h 4 RX
SR 02h 2 RR
L 03h 4 RX
AR 04h 2 RR
A 05h 4 RX
C 06h 4 RX
BNE 07h 4 RX
LR 08h 2 RR
ST 09h 4 RX
BR 15h 2 RR
```

POT.TXT

```
START
END
LTORG
DC
DS
DROP
USING
EQU
```

INPUT.TXT

```
PRGAM2 START 0
USING *,15
LA 15,SETUP
SR TOTAL,TOTAL
AC EQU 2
INDEX EQU 3
TOTAL EQU 4
DATABASE EQU 13
SETUP EQU *
USING SETUP,15
L DATABASE,=A(DATA1)
USING DATAAREA,DATABASE
SR INDEX,INDEX
LOOP L AC,DATA1(INDEX)
AR TOTAL,AC
A AC,=F'5'
ST AC,SAVE(INDEX)
A INDEX,=F'4'
C INDEX,=F'8000'
BNE LOOP
LR 1,TOTAL
BR 14
LTORG
SAVE DS 3F
DATAAREA EQU *
DATA1 DC F'25,26,27'
END
```

OUTPUT:

```
C:\Users\Exam\Desktop\2passcompiler>javac PassoneAssembler.java
C:\Users\Exam\Desktop\2passcompiler>java PassoneAssembler
====== PASS 1 ======
Symbol Table:
```

Symbol	Value	Length	R/A
PRGAM2	0	1	R
AC	2	1	A
INDEX	3	1	A
TOTAL	4	1	A
DATABASE	13	1	A
SETUP	6	1	R
LOOP	12	4	R
SAVE	64	4	R
DATAAREA	76	1	R
DATA1	76	4	R

Literal Table:

Literal	Value	Length	R/A
A(DATA1)	48	4	R
F'5'	52	4	R
F'4'	56	4	R
F'8000'	60	4	R