System Programming and Operating Systems Lab

ASSIGNMENT 6

Name: Shrirang Mhalgi
Roll No: 322008

Batch: B1

1 Date of Completion:

15/02/2019

2 Aim:

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

3 Objectives:

To implement pass-I of a two-pass assembler.

4 Theory:

The pass-wise grouping of tasks in two pass assembler is given below:

Pass-I

- Seperate the labels, mnemonic op-code and operand fields.
- Determine the storage requirement for every assembly language statement and update the location counter.
- Build the symbol table. Symbol table is used to store eeach label and each variable and its corresponding address.

Pass-II

- Generate machine code.

Following are the data structures used in pass-I of a two-pass assembler:

- Machine operation code table (MOT).
- Symbol table (ST).
- Literal table (LT).
- Pool table (PT).

5 Algorithm:

- 1. loc-cntr := 0; (default value)
  pooltab-ptr := 1;

  POOLTAB [1]:=1;

  littab-ptr := 1;

  2. While next statement is not an END statement

  (a) If label is present then

  this-label := symbol in label field;

  Enter (this-label, loc-cntr) in SYMTAB.

  (b) If an LTORG statement then

  (i) Process literals LITTAB [POOLTAB [pooltab-ptr]]... LITTAB [lit- tab-ptr – 1] to allocate memory and put the address in the address field. Update loc-cntr accordingly.

  (ii) pooltab-ptr := pooltab-ptr + 1;

  (iii) POOLTAB [pooltab-ptr] := littab-ptr;

  (c) If a START or ORIGIN statement then

  loc-cntr := value specified in operand field;

  (d) If an EQU statement then

  (i) this-addr := value of <address spec>;

  (ii) Correct the symtab entry for this-label to (this-label, this-addr).

  (e) If a declaration statement then

  (i) code := code of the declaration statement;

  (ii) size := size of memory area required by DC/DS.

  (iii) loc-cntr := loc-cntr + size;

  (iv) Generate 1C'(DL, code) ……'.

  (f) If an imperative statement then

  (i) code := machine opcode from OPTAB;

  (ii) loc-cntr := loc-cntr + instruction length from OPTAB;

  (iii) If operand is a literal then

  this-literal := literal in operand field;

  LITTAB [littab-ptr] := this-literal;

  littab-ptr := littab-ptr + 1;

  else (i.e. operand is a symbol)

  this-entry := SYMTAB entry number of operand;

  Generate 1C '(IS, code)(S, this-entry)';

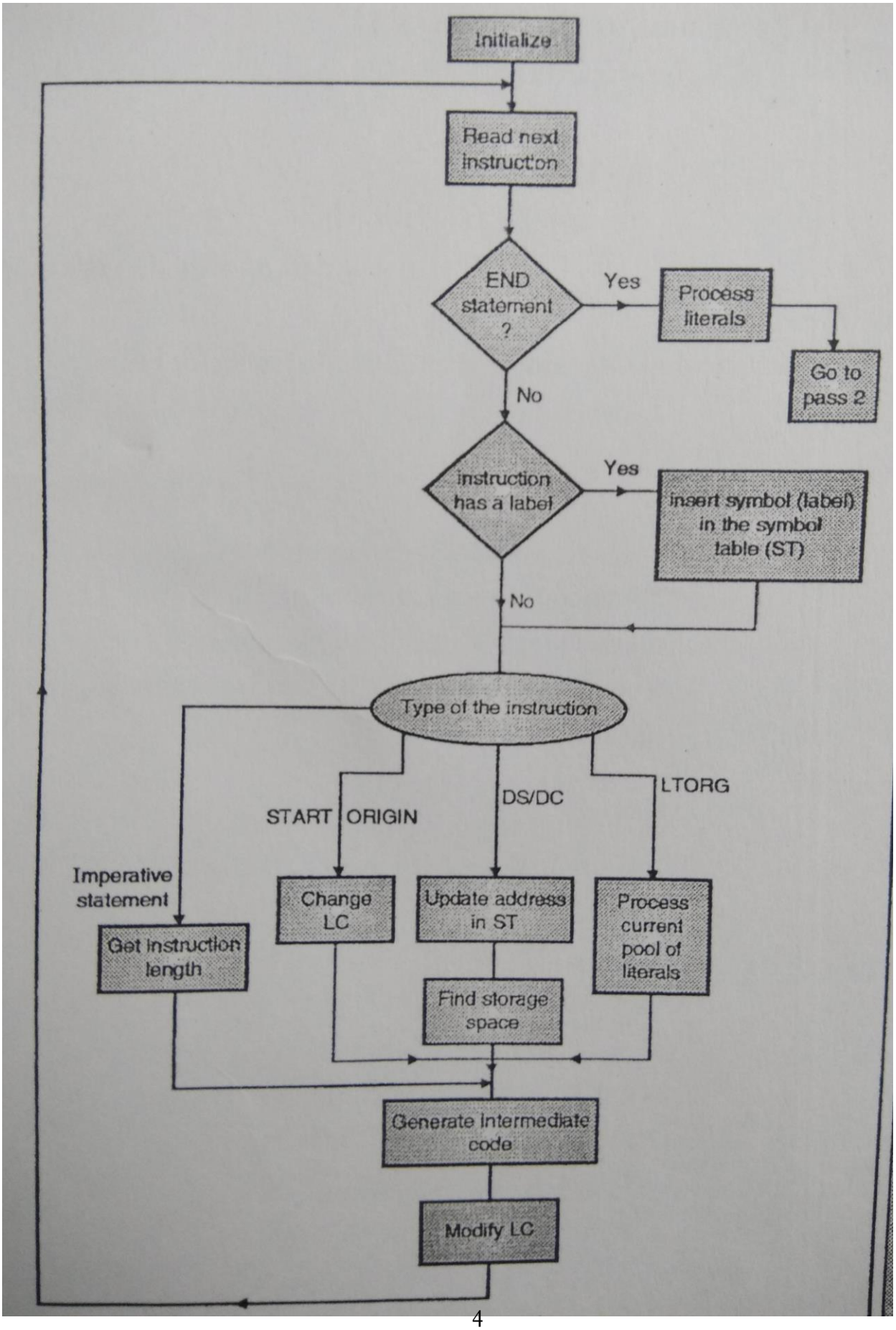
  3. (Processing of END statement)

(a) Perform step 2(b).

(b) Generate 1C'(AD.02)'.

(c) Go to Pass II.

6 Flowchart:


Initialize
Read next instruction
END statement ?
Yes
Process literals
Go to pass 2
No
instruction has a label
Yes
insert symbol (label) in the symbol table (ST)
No
Type of the instruction
START
ORIGIN
DS/DC
LTORG
Imperative statement
Change LC
Update address in ST
Process current pool of literals
Get instruction length
Find storage space
Generate intermediate code
Modify LC

7 Code:

Pass1a.java :

```
import java.io.*;
import java.util.*;
class AD{

    String startclass="IS";
    int startop=1;



    String endclass="IS";
    int endop=2;


    String orgclass="IS";
    int orgop=3;


    String equclass="IS";
    int equop=4;


    String ltorgclass="IS";
    int ltorgop=5;


}
class Dl{

    String dsclass="DL";
    int dspop=1;
    int dslen=0;


    String dcclass="DL";
    int dcop=2;
    int dclen=1;
}
class IS{
    String stpclass="IS";
    int stopop=0;
    int stoplen=1;
```

```
String addclass="IS";
int addop=1;
int addlen=1;


String subclass="IS";
int subop =2;
int  sublen=1;


String multclass="IS";
int multop =3;
int multlen =1;


String moverclass="IS";
int moverop=4;
int moverlen =1;


String movemclass="IS";
int movemop=5;
int movemlen=1;


String compclass="IS";
int compop=6;
int complen =1;


String bcclass="IS";
int bcop =7;
int  bclen =1;


String divclass="IS";
int divop =8;
int  divlen=1;


String readclass="IS";
int readop =9;
int  readlen=1;


String printclass="IS";
```

```
    int printop=10;
    int printlen=1;
}
public class Pass1a {
public static void main(String args[]) throws IOException {
IS is=new IS();
    AD ad=new AD();
    Dl dl=new Dl();
   FileReader fr=new File Reader("abc.txt");
       BufferedReader br=new BufferedReader(fr);
       String s1,s7 = null;
      FileWriter fw=new FileWriter("PassOp.txt",true);
      while((s1=br.readLine())!=null)
       {
          String s2[]=s1.replaceAll("^[,\\s]+", "").split("[,\\s]+");
        String s4[]=s1.split(",");
        for(String w:s4){

        }
        for(String w:s2){


            if(w.equals("START") || w.equals("END")){
               fw.write("(");
               fw.write("AD"+"  ,");
               if(w.equals("START"))
                    fw.write("0"+ad.startop);
               if(w.equals("END"))
                    fw.write("0"+ad.endop);
               fw.write(")");

            }
            if(w.equals("MOVER") || w.equals("MOVEM")){
               fw.write("\n");
               fw.write("(");
               fw.write("IS"+"  ,");
               if(w.equals("MOVER"))
                    fw.write("0"+is.moverop);
               if(w.equals("MOVEM"))
                    fw.write("0"+is.movemop);
               fw.write(")");

            }
            if(w.equals("AREG") || w.equals("BREG")){
               s7=w;
               fw.write(""+w+"");
            }
```

```
                if((w.equals("A")|| w.equals("B")) && (s7.equals("AREG")|| s7.equa
                        fw.write(""+w+"");
                        s7="";
                 }
                if((w.equals("'5'")|| w.equals("'4'")) && (s7.equals("AREG")|| s7.
                        s7="";
                 }
                if(w.equals("'5'") || w.equals("'4'")){
                        fw.write("(");
                    fw.write("L"+"  ,");
                    if(w.equals("'5'")){
                    fw.write("01");
                    }
                    if(w.equals("'4'")){
                    fw.write("02");
                    }
                    fw.write(")");
                }

                if(w.equals("DS")){
                    fw.write("\n");
                    fw.write("(");
                    fw.write("DL"+"  ,");
                    fw.write("01");
                    fw.write(")  ");
                    fw.write("(");
                    fw.write("C,01)");

                }

              }
             fw.write("\n");
          }
          fr.close();
          fw.close();

}

}
```

Contents of the files used:

1. abc.txt (Assembly language code)

```
START
MOVER AREG,A
MOVEM AREG, B
A DS 1
B  DS  1
```

END

8 Output:

Generated symbol table
0 A 204
1 B 205
Generated literal table
0 ’5’ 208
1 ’4’ 209

9 Conclusion:

In this assignment we understood in detail how to implement pass-I of a two-pass assembler and the data structures needed for it.