Peitam Mogal TE(B)-17

Assignment No-2

Pass-2 Assembler

Aim:- To design data structure for Pass-2 Assembler.

Problem Statement: Implement Pass-II of two pass assembler for pseudo-machine in Java using object oriented features. The output of assignment-1 (intermediate file and symbol table) should be input of this assignment.

Theory:

Two-pass assembler:

The two pass assembler two passes over the source program. In the first pass, it reads the entire source program, looking only for label definations. All the labels are collected, assigned address, and placed in the symbol table in the pass, no instructions as assembled and at the end the symbol table should contain all the labels defined in the program. To assign address or labels, the assembles maintain a Location Counter (LC). In the second pass the instructions are again read and the assembled using the symbol table.

Basically the assembler goes through the program one line at a time. and generates machine code for that instruction. Then the assembler proceed to the next instruction. In this way, the entire machine code progoam is created.

Difference between One pass and two Pass assembler:-

A one pass assembler passes over the source file exactly once, in the same pass collecting the labels resolving future references and doing the actual assembly

The difficult part is to resolve future label references (the problem of forward referencing) and assemble code in one pass. The one pass assembler prepare an intermediate file, which is used as input by the two pass assembler.

A two-pass assembler performs two sequential scans over the source code:

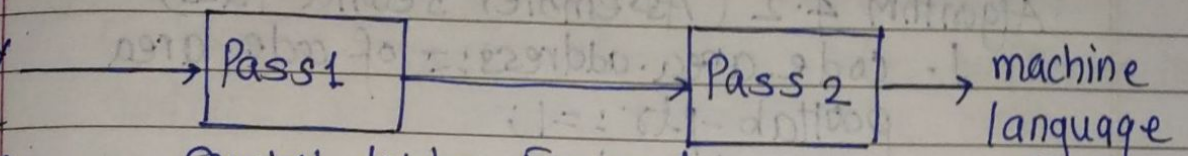
Pass I : symbols and literals are defined

Pass II : Object program is generated.

Parsing : moving in program lines to pull out op-codes and operands.

Data structures :

- Location Counter (LC): Points to the next location where the code will be placed.
- Op-code translation table: contains symbolic instruction's their lengths and their op-codes (or subroutine to use for translation)
- Symbol table (ST): Contains labels and their values
- String storage buffer (SSB): Contains ASCII character for the strings.
- Forward referena table (FRT): Contains pointer to the string in SSB and offset where it's value will be inserted in the object code



Symbol table forward references table String storage buffer Partially configured object file.

Algorithm:-

```
begin
    if starting address is given
        LOCCTR = starting address;
else
    LOCCTR = 0;
while OPCODE != END do ;; or EOF
    begin
    read a line from the code
    if there is a label
        if this label is in SYMTAB, then error
        else insert (label, LOCCTR) into SYMTAB
    Search OPTAB for the op code
    if found
        LOCCTR += N    ;; N is the length of this
                       instruction (4 for MIPS)
else if this is a assembly directive
        update LOCCTR as directed
    else error
    write line to intermediate file
    end
    program size = LOCCTR - starting address;
end
```

Algorithm 4.2 (Assembler Second Pass)

1. code_area_address := of code area
   pooltab_ptr := 1;
   loc_cntr := 0;

2. While next statement is not an END statement.
   (a) clear machine_code_buffer,
   (b) if an LTORG statement
      (i) Process literals in LITTAB (POOLTAB)
         [Pooltab_ptr] ... LITTAB

         [POOTLAB [pooltab_ptr+1]]-1 similar to processing of constants in DC statement.
         i.e assemble for literals in machine_code_buffer
      (ii) size := size of memory area required for literals;
      (iii) pooltab_ptr := pooltab_ptr+1;
   (c) If a START or ORIGIN statement then
      (i) loc_cntr := value specified in operand field;
      (ii) size := 0;
   (d) If a declaration statement
      (i) If a DC statement then
         Assemble the constant in machine_code_buffer.
      (ii) size := size of memory area required by DC/DS;
   (e) If an imperative statement
      (i) Get operand address from SYMTAB or LITTAB.
      (ii) Assemble instruction in machine_code_buffer.
      (iii) size := size of instruction;

(f) If size ≠ 0 then
(i) Move contents of machine_code_buffer to the address code area_address + loc_cntr.
(ii) loc_cntr := loc_cntr + size;

3. (Processing of END Statement)
(a) Perform steps 2(b) and 2(f)
(b) Write code_area into output file.

Input:
IC.txt

AD	01	C	200	
IS	04	1	L	1
IS	05	1	S	1
IS	04	2	L	2
IS	04	3	S	3
AD	05			
IS	01	3	L	3
IS	00			
DL	02	C	1	
DL	02	C	1	
AD	02			

LITTAB.txt:

='4'	204
='6'	210
='1'	205

SYMTAB.txt

A	208
LOOP	203
B	209

POOLTAB.txt

1
3

Expected Output:

200	04	1	204
201	05	1	208
202	04	2	210
203	04	3	209
204	00	0	004
205	00	0	006
206	01	3	205
207	00	0	000
208			
209			
210	00	0	001

Conclusion:
Thus we have generated machine code for the source program.

Name: Pritam Gotiram Mogal **DivRoll**: TE -B-17

Assignment No. 02 [Pass 2 Assembler]

Problem Satement: Implement Pass-II of two pass assembler for pseudo-machine in Java using object oriented features. The output of assignment-1 (intermediate file and symbol table) should be input for this assignment.

---

1. Pass 2 Program:

```
import java.io.BufferedReader; import
java.io.BufferedWriter; import
java.io.FileReader; import java.io.FileWriter;
import java.io.IOException; import
java.lang.reflect.Array; import
java.util.ArrayList; import
java.util.Hashtable; import java.util.Map;
public class Pass2 { public static void
main(String[] args) { try {

                    //1. Read Intermediate code file
                      String f ="/home/sagar-ravan/Desktop/IC_new.txt";
                      FileReader fw =new FileReader(f);
                      BufferedReader IC_file=new BufferedReader(fw);

                      //2.Read Symbol table file
                      String f1="/home/sagar-ravan/Desktop/SYMTAB.txt";
                      FileReader fs=new FileReader(f1);
                      BufferedReader symtab_file=new BufferedReader(fs);
                    symtab_file.mark(500);

                      //3.Read Literal table file
                      String f2="/home/sagar-ravan/Desktop/LITTAB.txt";
                      FileReader fl=new FileReader(f2);
                      BufferedReader littab_file=new BufferedReader(fl);
                    littab_file.mark(500);


                      //4.create  littab array and hashtable for symbol table


                    String littab[][]=new String[10][2] ;

                    Hashtable<String, String> symtab = new Hashtable<String,
String>();
                    String str;
                    int z=0;
                    //5.Read LITTAB.txt
                    while ((str = littab_file.readLine()) != null) {

                          littab[z][0]=str.split("\\s+")[0]; //first word
                          littab[z][1]=str.split("\\s+")[1]; //second word z++;
                    }
                    //6.Read SYMTAB.txt
```

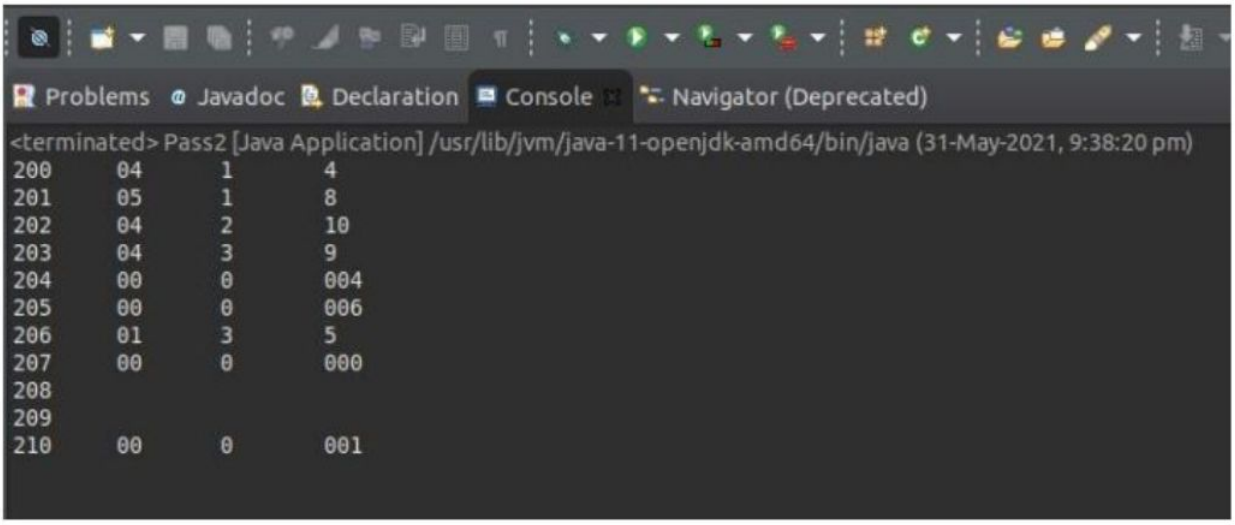
```
                while ((str = symtab_file.readLine()) != null) {
                symtab.put(str.split("\\s+")[0], str.split("\\s+")[1]); }
          //7.Read POOLTAB.txt
                String f3 = "/home/sagar-ravan/Desktop/POOLTAB.txt";
                FileReader fw3 = new FileReader(f3);
                BufferedReader pooltab_file = new BufferedReader(fw3);



                ArrayList<Integer> pooltab = new ArrayList<Integer>();
                String t;
                while ((t = pooltab_file.readLine()) != null) {
                      pooltab.add(Integer.parseInt(t));
                }

                int pooltabptr = 1;
                int temp1 = pooltab.get(0);    //dry run
                int temp2 = pooltab.get(1);

                //7.Read IC.txt
                String sCurrentLine;
                sCurrentLine = IC_file.readLine();
                int locptr=0;
                //locptr=Integer.parseInt(sCurrentLine.split("\\s+")[3]);
                locptr=Integer.parseInt(sCurrentLine.split("\t")[3]);
                while ((sCurrentLine = IC_file.readLine()) != null) {


                       System.out.print(locptr+"\t");


                      String s0 = sCurrentLine.split("\t")[0]; //contains
statement type

                      String s1 = sCurrentLine.split("\t")[1]; //contains
statement code

                      if (s0.equals("IS")) {

                            System.out.print(s1+"\t"); if
                            (sCurrentLine.split("\t").length == 5) {


                                  System.out.print(sCurrentLine.split("\t")[2]
+  "\t");
                                  //7.2 if third character is L
                                  if (sCurrentLine.split("\t")[3].equals("L"))
{ int add =
Integer.parseInt(sCurrentLine.split("\t")[4]);


      //machine_code_file.write(littab[add-1][1]);
                                        System.out.print(littab[add-1][1]);

                                    }
                                   //7.3 or if third character is S
                                  if (sCurrentLine.split("\t")[3].equals("S"))
{ int add1 =
Integer.parseInt(sCurrentLine.split("\t")[4]);

                                        //search for the 4th word in symbol
```

```
table int i = 1; String l1;
                                        for (Map.Entry m : symtab.entrySet())
{
                                  if (i == add1) {

                                                        System.out.print((String)
m.getValue());
                                                    }
                                                    i++;
                                            }

                                            }
                                    }  else  {

                                System.out.print("0\t000");
                                    }
                            }



                            //DRY RUN is a must

                            if (s0.equals("AD")) {
                                  littab_file.reset();
                                  if (s1.equals("05")) {  //if it is
LTORG int j = 1; while (j < temp1) { littab_file.readLine();
                                            }
                                 while (temp1 < temp2) {

                                            System.out.print("00\t0\t00" +
littab_file.readLine().split("'")[1]);
                                            if(temp1<(temp2-1)){
                                                  locptr++;

                                                  System.out.println();

                                                  System.out.print(locptr+"\t");
                                            }
                                            temp1++;
                                      } temp1 =
                                      temp2;
                                      pooltabptr++;
                                      if (pooltabptr < pooltab.size()) {
                                      temp2 = pooltab.get(pooltabptr); }
                                } int j =
                                1;
                                if (s1.equals("02")) {  //if it is
"END" stmt
                                     String s;
                                      while ((s = littab_file.readLine()) != null)
{
                                   if (j >= temp1)

                                                  System.out.print("00\t0\t00" +
s.split("'")[1]); j++;
                                            }
                                }
                            }



                             if(s0.equals("DL")&&s1.equals("01")){      //if it
is DC stmt
```

```
                                System.out.print("00\t0\
t00"+sCurrentLine.split("'")[1]);

                            }



                            locptr++;

                            System.out.println();
                        }
                        IC_file.close();
                        symtab_file.close();
                        littab_file.close();
                        pooltab_file.close();

                    }  catch (IOException e)  {
                        e.printStackTrace();
                    }

            }
```

PASS 2 - ASSEMBLER OUTPUT:

PASS- 2 OUTPUT:



IC_New.txt

IC_new.txt

```
1 AD      01      C       200
2 IS      04      1       L       1
3 IS      05      1       S       1
4 IS      04      2       L       2
5 IS      04      3       S       3
6 AD      05
7 IS      01      3       L       3
8 IS      00
9 DL      02      C       1
10 DL     02      C       1
11 AD     02
```

Input.txt

SYMTAB.txt

```
1 A       8
2 LOOP    3
3 B       9
```

LITTAB.txt

LITTAB.txt

```
1 ='4'    4
2 ='6'    10
3 ='1'    5
```

POOLTAB.txt

POOLTAB.txt

```
1 1
2 3
```

SYMTAB.txt

Input.txt

```
1 START 200
2  MOVER AREG,='4'
3  MOVEM AREG,A
4  MOVER BREG,='1'
5 LOOP MOVER CREG,B
6  LTORG
7  ADD CREG,='6'
8  STOP
9 A DS 1
10 B DS 1
11  END
```