Ritom Gupta, PA-25, Panel 1, Batch B2

School of CET

System Software and Compiler lab

Assignment No.2

TY BTech CSE

Assignment Title: Design of Pass 2 of Two Pass Assembler.

Aim: Design suitable data structure & implement pass 2 of Two Pass Assembler pseudo machine.

Objective: Design suitable data structure & implement pass 2 of Two Pass Assembler pseudo machine. Subset should consist of a few instructions from each category & few assembler directive.

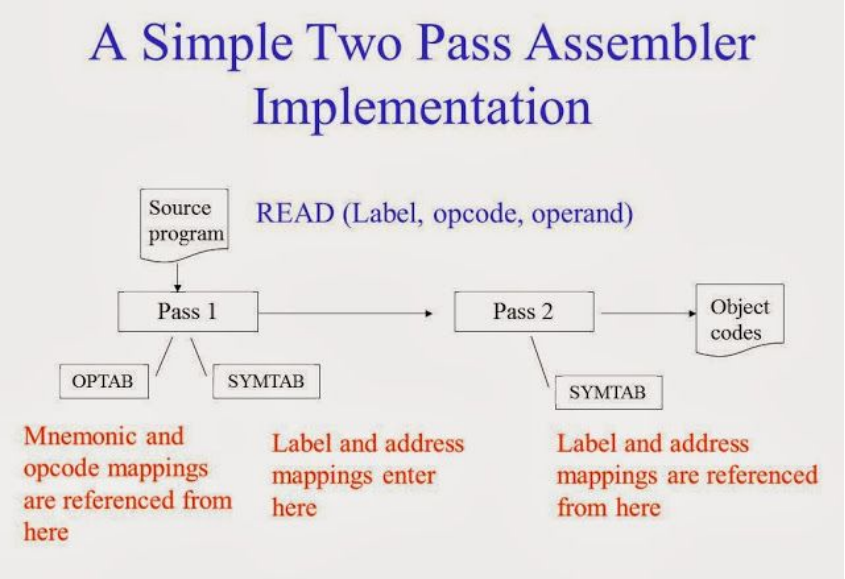
Theory:

Design of a Two Pass Assembler:

Two-pass assembler: Assemblers typically make two or more passes through a source program in order to resolve forward references in a program. A forward reference is defined as a type of instruction in the code segment that is referencing the label of an instruction, but the assembler has not yet encountered the definition of that instruction.

Pass 1: Assembler reads the entire source program and constructs a symbol table of names and labels used in the program, that is, name of data fields and programs labels and their relative location (offset) within the segment.
Pass 1 determines the amount of code to be generated for each instruction.

Pass 2: The assembler uses the symbol table that it constructed in Pass 1. Now it knows the length and relative of each data field and instruction, it can complete the object code for each instruction. It produces .OBJ (Object file), .LST (list file) and cross reference (.CRF) files.



Algorithm for Pass II

1. Code_area_address: = address of code area;
   loccntr: = 0;

2. While next statement is not an END statement
(a) Clear *machine_code_buffer;*
(b) If a START or ORIGIN statement then
   (i) *loccntr:* = value specified in operand field;
   (ii) *size:* = 0;
(c) If a declaration statement
   (i) If a DC statement then
      Assemble the constant in *machine_code_buffer.*
   (ii) s*ize:* = size of memory area required by DC/DS;
(d) 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 i*= 0 then
(i) Move contents of *machine_code_buffer* to the address *code_area_address+loccntr;*
(ii) *loccntr: = loccntr + size;*

3. Write *code area* into output file.

Ritom Gupta, PA-25, Panel 1, Batch B2

Input: Symbol table and Intermediate code generated by Pass I.

Output:

1. Final Output (After Pass II)

Address (LC value)	Op-code	Operand 1 (Value/Address)	Operand 2 (Value/Address)

Conclusion: The function of Pass II in an assembler are studied.

Platform: Linux (JAVA)

```
1
2  import java.io.*;
3  import java.util.*;
4
5  class Operator
6  {
7    String name;
8    String cls;
9    int opcode;
10   Operator(String a, String c, int op)
11   {
12     this.name = a;
13     this.cls = c;
14     this.opcode = op;
15   }
16 }
17
18 class Register
19 {
20   String name;
21   int no;
22   Register(String a, int op)
23   {
24     this.name = a;
25     this.no = op;
26   }
27 }
28
29
30 class Condition
31 {
32   String name;
33   int no;
34   Condition(String a, int op)
35   {
36     this.name = a;
37     this.no = op;
38   }
39 }
40
41
42 class Symbol
43 {
44   String name;
45   int addr;
46   int length;
47   Symbol(String a, int op, int len)
48   {
49     this.name = a;
50     this.addr = op;
51     this.length = len;
52   }
53 }
54
55
56
57 public class assembler_pass2
58 {
59   public static void main(String[] args) throws IOException
60   {
```

```
61      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
62
63      BufferedReader objReader = new BufferedReader(new
FileReader("pass1.txt"));
64      String strCurrentLine;
65
66      //Making the symbol tables
67      ArrayList<Symbol> symboltable = new ArrayList<Symbol>(25);
68
69      //Accepting the symbol table
70      int cont = 1;
71      while(cont == 1)
72      {
73      System.out.println("Enter name: ");
74      String s = br.readLine();
75      System.out.println("Enter address: ");
76      int ad = Integer.parseInt(br.readLine());
77      System.out.println("Enter size: ");
78      int size = Integer.parseInt(br.readLine());
79
80      //Adding the symbol
81
82      symboltable.add(new Symbol(s,ad,size));
83
84      System.out.println("Add another? ");
85      cont = Integer.parseInt(br.readLine());
86
87      }
88
89      //Displaying the symbol table
90      Iterator<Symbol> display = symboltable.iterator();
91      Symbol x;
92      while(display.hasNext())
93      {
94        x = display.next();
95        System.out.println(x.name + " " + x.addr + " " + x.length);
96      }
97
98      //Creating the output files
99      FileWriter fw = new FileWriter("pass2.txt");
100     BufferedWriter write = new BufferedWriter(fw);
101     int first = 1;
102     while ((strCurrentLine = objReader.readLine()) != null)
103     {
104       if(first == 1)
105       {
106         first = 0;
107       }
108       else
109       {
110         String[] splited = strCurrentLine.split("\\s+");
111         int len = splited[1].length();
112         String[] inst = splited[1].substring(1,len-1).split(",");
113         if(inst[0].equals("AD") && (inst[1].equals("1") ||
inst[1].equals("2")))
114         {
115           //skip
116         }
117         else
118         {
```

```
          String pass2line = "";
          pass2line = pass2line+splited[0] + " + ";
          if(inst[0].equals("IS"))
          {
            if(inst[1].equals(0))
            {
              pass2line = pass2line +"00 0 000" + " ";
            }
            else
            {
              pass2line = pass2line + inst[1] + " ";
              int len2 = splited[2].length();
              String[] arg = splited[2].substring(1,len2-1).split(",");
              if(arg[0].equals("S"))
              {
                //Fetching from symbol table
                Symbol s = symboltable.get(Integer.parseInt(arg[1]));
                pass2line = pass2line + s.addr + " ";
              }
              else
              {
                // Register or condition
                pass2line = pass2line+ arg[0] + " ";
              }

              len2 = splited[3].length();
              arg = splited[3].substring(1,len2-1).split(",");
              if(arg[0].equals("S"))
              {
                //Fetching from symbol table
                int index = Integer.parseInt(arg[1])-1;
                Symbol s = symboltable.get(index);
                pass2line = pass2line + s.addr + " ";
              }
              else
              {
                // Register or condition
                pass2line = pass2line+ arg[0] + " ";
              }
            }


          }

          else if(inst[0].equals("DL") && inst[1].equals("1"))
          {
            pass2line = pass2line + "00 0 ";
            int len3 = splited[2].length();
            String[] arg = splited[2].substring(1,len3-1).split(",");
            pass2line = pass2line + arg[1] + " ";
          }
          write.write(pass2line);
          write.newLine();
        }
      }

    }

    write.close();

```

```
179
180
181   }
182
183 }
184
185 /*
186 //-------------------------------------------------------------------------
    --------
187
188  OUTPUT -
189
190 100) +
191 103) + 4 1 108
192 104) + 7 2 103
193 105) + 5 1 106
194 106) + 00 0 2
195 107) + 1 2 100
196 108) + 00 0 1
197 109) + 00 0 3
198
199 //-------------------------------------------------------------------------
    --------
200
201 INPUT ( PASS1 CODE)
202
203     (AD,1) (C,100)
204 100) (DL,2) (C,3)
205 103) (IS,4) (1) (S,3)
206 104) (IS,7) (2) (S,2)
207 105) (IS,5) (1) (S,4)
208 106) (DL,1) (C,2)
209 107) (IS,1) (2) (S,1)
210 108) (DL,1) (C,1)
211 109) (DL,1) (C,3)
212 110) (AD,2)
213
214 //-------------------------------------------------------------------------
    --------
215 */
216
217
```