Name- Vasu Kalariya
Roll - PE29
Sub - SSC

Lab Assignment - 1

Title: Design of Pass 1 of a Pass Assembler

Aim: Design suitable data structures & implement pass 1 of a pass assembler pseda machine.

Objective: Design suitable data structure & implement pass 1 of a pass assembler pseda machine. Subset should consist of a few instruction from each category in few assembler directive

Theory:

→ Assembler: It translate assembly ~~lan~~ language program to binary language. Input for assembler is language generated by compiler. For Pass 1 assembler defines symbols & literals & save them in symbolic opcode table. It assigns machine address to symbolic labels.
It performs assembler service required by pseda operations & saves files for future use

→ Design specification of Assembler

⇒ Analysis Phase
It seperates label, memonic, opcode & operand from instruction statement.
If label is present, it made entry in symbol table thus builds symbol table
Performs LC processing & constructs IC.

⇒ Synthesis Phase.
It obtains machine code corresponding to memonics from opcode table

It obtains address of memory operand from symbol table
Synthesizes a machine instruction

→ Algorithm for Pass I

1 loccntr = 0 (default value)
2 while next statement is not an END statement
(a) If lable is present then
this label = symbol in label field
Enter (this-label, loccntr) in SYMTAB.
(b) If a START or ORIGIN statement then
loccntr = value specified in operand field
(c) 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)
(d) If a declaration statement then
(i) code = code of the declaration statement.
(ii) size = size of memory area required by DC/DS
(iii) loccntr = loccntr + size.
(iv) Generate IC '(DL, code)
(e) If an imperative statement then
(i) code = machine opcode from OPTAB
(ii) loccntr = loccntr + instruction length from OPTAB
(iii) If operand is a symbol then.
this-entry = SYMTAB entry number of operand
Generate IC (IS, code) (S, this-entry)

3 Processing of END statement
(b) Generate IC
(c) Go to Pass II.

Input: ALP1 intermediate code generated by Pass 1

Output:

OPTAB

Mnemonic	OP-code
Start	01, AD
MOVER	01, IS
SUB	02, IS
MOVER	04, IS
ORIGIN	03, AD
MOVER	04, IS
DS	01, DL
DC	02, DL
END	02, AD

Symbol Table

Sym-id	Sym-name	Sym-addr	length
1	A1	301	3
2	LOOP	401	1
3	B1	304	1

Intermidiate from (After Pass 1) / Final output.

Add. (LC value)	Opcode	Operand 1	Operand 2
	(AD,01)		(C,400)
400	(IS,04)	1	(S,01)
401	(IS,02)	2	(S,01)
402	(IS,04)	2	(S,03)
	(AD,03)		(C,300)
300	(IS,04)	2	(S,01)
301	(DL,02)		(C,3)
304	(DL,01)		(C,3)
305	(AD,02)		

Conclusion: The function of Pass I in assembler are ~~schedu~~ studied along with errors coming in each pass.

Platform: Linux (JAVA)

File - F:\T9\SSC\Assi1\src\Pass1.java

```java
/*
Name : Vasu Kalariya
Roll : PE29
 */

import java.util.*;
import java.io.*;
public class Pass1 {
    public static void main(String[] args) {

        BufferedReader br = null;
        FileReader fr = null;

        FileWriter fw = null;
        BufferedWriter bw = null;

        try {
            String inputfilename = "F:\\T9\\SSC\\Assi1\\src\\input.txt";        //Input File
            fr = new FileReader(inputfilename);
            br = new BufferedReader(fr);

            String OUTPUTFILENAME = "F:\\T9\\SSC\\Assi1\\src\\output.txt";      //Output File
            fw = new FileWriter(OUTPUTFILENAME);
            bw = new BufferedWriter(fw);

            Hashtable<String, String> is = new Hashtable<String, String>();     //For Imperative
            is.put("STOP", "00");
            is.put("ADD", "01");
            is.put("SUB", "02");
            is.put("MULT", "03");
            is.put("MOVER", "04");
            is.put("MOVEM", "05");
            is.put("COMP", "06");
            is.put("BC", "07");
            is.put("DIV", "08");
            is.put("READ", "09");
            is.put("PRINT", "10");

            Hashtable<String, String> dl = new Hashtable<String, String>();     //For Declarative
            dl.put("DC", "01");
```

```
41              dl.put("DS", "02");
42
43              Hashtable<String, String> ad = new
   Hashtable<String, String>();     // For Assembler
   Directive
44
45              ad.put("START", "01");
46              ad.put("END", "02");
47              ad.put("ORIGIN", "03");
48              ad.put("EQU", "04");
49              ad.put("LTORG", "05");
50
51              Hashtable<String,String> rt = new
   Hashtable<>();
52              rt.put("AREG","1");
53              rt.put("BREG","2");
54              rt.put("CREG","3");
55              rt.put("DREG","4");
56
57              Hashtable<String, String> symtab = new
   Hashtable<String, String>();     // SYMTAB for
   Symbols
58              Hashtable<String, Integer> symtab_ptr =
   new Hashtable<>();
59              Hashtable<String, String> littab = new
   Hashtable<String, String>();     // LITTAB for
   Literals
60              ArrayList<Integer> pooltab=new ArrayList<
   Integer>();                      // PoolTab
61
62              String sCurrentLine;
63              int locptr = 0;
64              int litptr = 1;
65              int symptr = 1;
66              int pooltabptr = 1;
67
68              sCurrentLine = br.readLine();
69
70              String s1 = sCurrentLine.split(" ")[1];
71              String s2 = sCurrentLine.split(" ")[2];
72              if (s1.equals("START"
   )) {                                    // Initial
   Start check
73                  bw.write("\t (AD,01)\t");
```

```
74                  s2 = sCurrentLine.split(" ")[2];
75                  bw.write("(C," + s2 + ")\n");
76                  locptr = Integer.parseInt(s2
   );                              // Storing the address
    in pointer
77              }
78
79              while ((sCurrentLine = br.readLine
   ()) != null) {                  // untill reach end of
    file
80                  int mind_the_LC = 0;
81                  String type = null;
82
83                  int flag2 = 0
   ;                                    //checks whether
   addr is assigned to current symbol
84
85                  s1 = sCurrentLine.split(" |\\,")[0];
   // reading first word from string
86
87                  for (Map.Entry m : symtab.entrySet
   ()) {            // allocating addr to symbols listed
88                      if (s1.equals(m.getKey())) {
89                          m.setValue(locptr);
90                          flag2 = 1;
91                      }
92                  }
93                  if (s1.length() != 0 && flag2 == 0
   ) {     //if current string addr is not assigned,
94                      symtab.put(s1, String.valueOf(
   locptr));
95                      symptr++;
96                  }
97
98                  int isOpcode = 0
   ;                                 //checks whether current
    word is an opcode or not
99
100                 s1 = sCurrentLine.split(" |\\,")[1
   ];           //second word from string
101
102                 // Cheacking in different table and
   values
103
```

```
104                 for (Map.Entry m : is.entrySet()) {
105                     if (s1.equals(m.getKey())) {
106                         bw.write(locptr + "  (IS,"
     + m.getValue() + ")\t");       //if match found in
    imperative stmt
107                         type = "is";
108                         isOpcode = 1;
109                     }
110                 }
111 
112                 for (Map.Entry m : ad.entrySet()) {
113                     if (s1.equals(m.getKey())) {
114                         if (s1.equals("ORIGIN")){
115                             bw.write("\t (AD," + m.
    getValue() + ")\t");         // Origin condition
    check for reset address
116                             type = "ad";
117                             isOpcode = 1;
118                         }
119                         else{
120                             bw.write(locptr + "   (
    AD," + m.getValue() + ")\t");        //if match
    found in Assembler Directive
121                             type = "ad";
122                             isOpcode = 1;
123                         }
124                     }
125                 }
126                 for (Map.Entry m : dl.entrySet()) {
127                     if (s1.equals(m.getKey())) {
128                         bw.write(locptr+ "   (DL,"
     + m.getValue() + ")\t");       //if match found in
    declarative stmt
129                         type = "dl";
130                         isOpcode = 1;
131                     }
132                 }
133 
134 
135                 if (s1.equals("LTORG")) {
136                     pooltab.add(pooltabptr);
137                     for (Map.Entry m : littab.
    entrySet()) {
138                         if (m.getValue() == ""
```

```
138 ) {                 //if addr is not assigned to the
    literal
139                             m.setValue(locptr);
140                             locptr++;
141                             pooltabptr++;
142                             mind_the_LC = 1;
143                             isOpcode = 1;
144                         }
145                     }
146                 }
147
148
149                 if (s1.equals("END")) {
150                     pooltab.add(pooltabptr);
151                     for (Map.Entry m : littab.
    entrySet()) {
152                         if (m.getValue() == "") {
153                             m.setValue(locptr);
154                             locptr++;
155                             mind_the_LC = 1;
156                         }
157                     }
158                 }
159
160
161                 if(s1.equals("EQU")){
162                     symtab.put("equ", String.valueOf
    (locptr));
163                 }
164
165
166                 if (sCurrentLine.split(" |\\,").
    length > 2) {       //if there are 3 words handel
    index out of bound
167                     s2 = sCurrentLine.split(" |\\,"
    )[2];               //reading the 3rd word
168
169                     if (rt.containsKey(s2)){
170                         bw.write(rt.get(s2)+"\t");
171                         isOpcode = 1;
172                     }
173                     else if (type == "dl") {
174                         bw.write("(C," + s2 + ")\t"
    );
```

```
175                     }
176                     else if (s1.equals("ORIGIN")){
177                         bw.write("(C," + s2 + ")\t"
    );
178                         locptr = (Integer.parseInt(
    s2) - 1);
179                     }
180                     else {
181                         symtab.put(s2, "");
182                     }
183                 }
184 
185 
186                 if (sCurrentLine.split(" |\\,").
    length > 3) {        //if there are 4 words
187 
188                     String s3 = sCurrentLine.split(
    " |\\,")[3];            //reading 4th word.
189 
190                     if (s3.contains("="
    )) {                                //it is either
    a literal, or a symbol
191                         littab.put(s3, "");
192                         bw.write("(L," + litptr + ")
    \t");
193                         isOpcode = 1;
194                         litptr++;
195                     }
196                     else {
197                         if (symtab_ptr.containsKey(
    s3)){
198                             bw.write("(S," +
    symtab_ptr.get(s3) + ")\t");
199                         }
200                         else {
201                             symtab.put(s3, "");
202                             symtab_ptr.put(s3,symptr
    );
203                             bw.write("(S," + symptr
     + ")\t");
204                             symptr++;
205                         }
206                     }
207                 }
```

```
208
209                 bw.write("\n");     //done with a
    line.
210
211                 if (mind_the_LC == 0){
212                     if (s1.equals("DS")) {
213                         locptr += Integer.parseInt(
    s2);
214                     }
215                     else {
216                         locptr++;
217                     }
218                 }
219
220             }
221
222             String f1 = "F:\\T9\\SSC\\Assi1\\src\\
    SYMTAB.txt";
223             FileWriter fw1 = new FileWriter(f1);
224             BufferedWriter bw1 = new BufferedWriter(
    fw1);
225             for (Map.Entry m : symtab.entrySet()) {
226                 bw1.write(m.getKey() + "\t" + m.
    getValue()+"\n");
227                 System.out.println(m.getKey() + " "
     + m.getValue());
228             }
229
230             String f2 = "F:\\T9\\SSC\\Assi1\\src\\
    LITTAB.txt";
231             FileWriter fw2 = new FileWriter(f2);
232             BufferedWriter bw2 = new BufferedWriter(
    fw2);
233             for (Map.Entry m : littab.entrySet()) {
234                 bw2.write(m.getKey() + "\t" + m.
    getValue()+"\n");
235                 System.out.println(m.getKey() + " "
     + m.getValue());
236             }
237
238             String f3 = "F:\\T9\\SSC\\Assi1\\src\\
    POOLTAB.txt";
239             FileWriter fw3 = new FileWriter(f3);
240             BufferedWriter bw3 = new BufferedWriter(
```

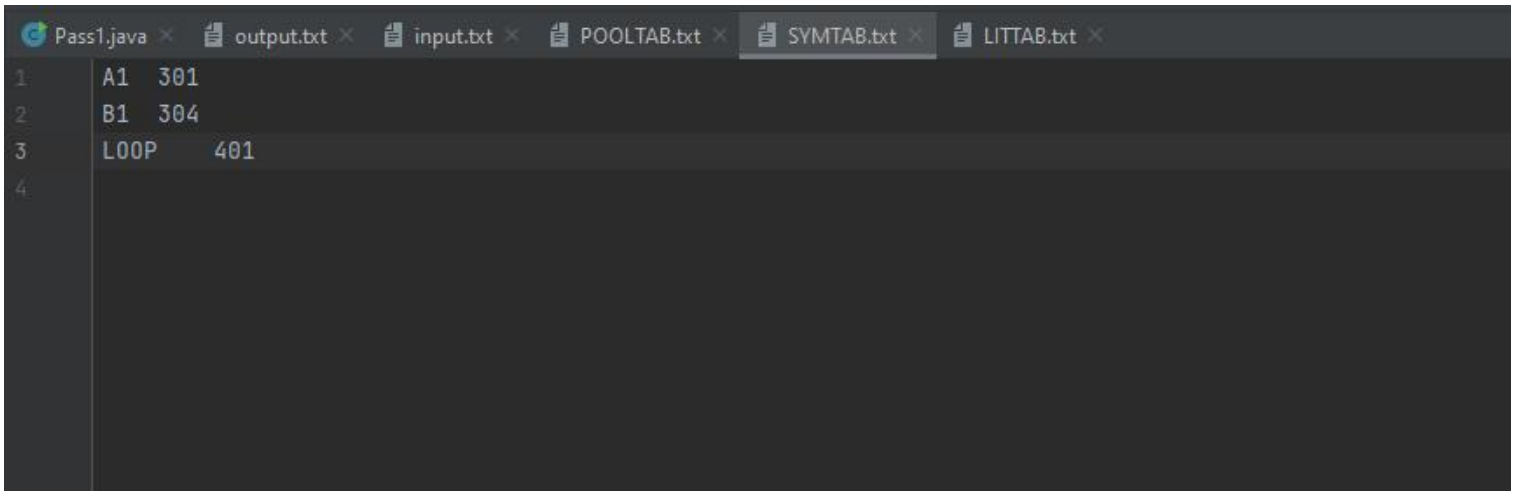
```
240 fw3);
241             for (Integer item : pooltab) {
242                 bw3.write(item+"\n");
243                 System.out.println(item);
244             }
245 
246             bw.close();
247             bw1.close();
248             bw2.close();
249             bw3.close();
250 
251         } catch (IOException e) {
252             e.printStackTrace();
253         }
254 
255     }
256 }
```

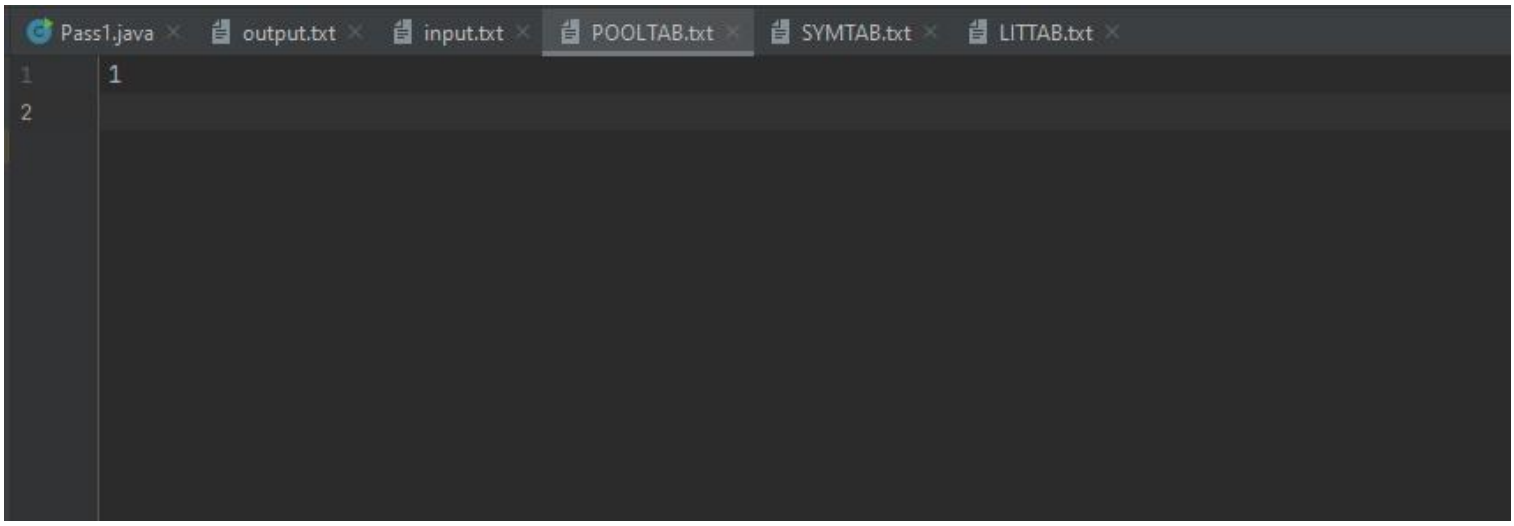
Pass1.java | output.txt | input.txt | POOLTAB.txt | SYMTAB.txt | LITTAB.txt

```
 START 400
 MOVER AREG,A1
LOOP SUB BREG,A1
 MOVER BREG,B1
 ORIGIN 300
 MOVER BREG,A1
A1 DS 3
B1 DC 3
 END
```

Pass1.java | output.txt | input.txt | POOLTAB.txt | SYMTAB.txt | LITTAB.txt

```
     (AD,01)    (C,400)
400  (IS,04)    1   (S,1)
401  (IS,02)    2   (S,1)
402  (IS,04)    2   (S,3)
     (AD,03)    (C,300)
300  (IS,04)    2   (S,1)
301   (DL,02)   (C,3)
304   (DL,01)   (C,3)
305   (AD,02)
```


Pass1.java
output.txt
input.txt
POOLTAB.txt
SYMTAB.txt
LITTAB.txt
A1 301
B1 304
LOOP 401


Pass1.java
output.txt
input.txt
POOLTAB.txt
SYMTAB.txt
LITTAB.txt
1
1
2