NAME: TEJAS NALAWADE PRACTICAL NO: 1 ROLL NO: TCOD01

Title of Assignment: Design suitable Data structures and implement Pass-I of a two-pass assembler for pseudo-machine.

Problem Statement: Implement one pass-I of TWO Pass assembler with hypothetical Instruction set using Java language. Instruction set should include all types of assembly language statements such as Imperative, Declarative and Assembler Directive. While designing stress should be given on a) How efficiently Mnemonic opcode could be implemented so as to enable faster retrieval on op-code. b) Implementation of symbol table for faster retrieval.

CODE:

```
import java.io.*;
import java.util.*;

class pass1 {
   public static void main(String args[]) throws NullPointerException, FileNotFoundException {
      String REG[] = {"ax", "bx", "cx", "dx"};
      String IS[] = {"stop", "add", "sub", "mult", "mover", "movem", "comp", "bc", "div", "read"};
      String DL[] = {"ds", "dc"};
      int temp1 = 0;
      int f = 0;
      Obj[] literal_table = new Obj[10];
      Obj[] symb_table = new Obj[10];
      Obj[] optab = new Obj[60];
      Pooltable[] pooltab = new Pooltable[5];
      String line;

      try {
         BufferedReader br = new BufferedReader(new FileReader("sample.txt"));
         BufferedWriter bw = new BufferedWriter(new FileWriter("output.txt"));
         Boolean start = false;
         Boolean end = false, fill_addr = false, ltorg = false;
         int total_symb = 0, total_ltr = 0, optab_cnt = 0, pooltab_cnt = 0, loc = 0, temp, pos;

         while ((line = br.readLine()) != null && !end) {
            String tokens[] = line.split(" ", 4);
            if (loc != 0 && !ltorg) {
               if (f == 1) {
                  ltorg = false;
                  loc = loc + temp1 - 1;
                  bw.write("\n" + String.valueOf(loc));
                  f = 0;
                  loc++;
               } else {
                  bw.write("\n" + String.valueOf(loc));
                  ltorg = false;
                  loc++;
               }
            }
```

```
            ltorg = fill_addr = false;
            for (int k = 0; k < tokens.length; k++) {
               pos = -1;
               if (start == true) {
                  loc = Integer.parseInt(tokens[k]);
                  start = false;
               }
               switch (tokens[k]) {
                  case "start":
                     start = true;
                     pos = 1;
                     bw.write("\t(AD,"+ pos +")");
                     break;
                  case "end":
                     end = true;
                     pos = 2;
                     bw.write("\t(AD," + pos +")\n");
                     for (temp = 0; temp < total_ltr; temp++) {
                        if (literal_table[temp].addr == 0) {
                           literal_table[temp].addr = loc - 1;
                           bw.write("\t(DL,2) \t (C," + literal_table[temp].name +")"+"\n" + loc++);
                        }
                     }
                     break;
                  case "origin":
                     pos = 3;
                     bw.write("\t (AD," + pos +")");
                     pos = search(tokens[++k], symb_table, total_symb);
                     k++;
                     bw.write("\t(C," +(symb_table[pos].addr) +")");
                     loc = symb_table[pos].addr;
                     break;
                  case "ltorg":
                     ltorg = true;
                     pos = 5;
                     bw.write("\t(AD," + pos +")\n");
                     for (temp = 0; temp < total_ltr; temp++) {
                        if (literal_table[temp].addr == 0) {
                           literal_table[temp].addr = loc - 1;
                           bw.write("\t(DL,2) \t (C," + literal_table[temp].name +")" +"\n" +loc++);
                        }
                     }
                     if (pooltab_cnt == 0) {
                        pooltab[pooltab_cnt++] = new Pooltable(0, temp);
                     } else {
                        pooltab[pooltab_cnt] = new Pooltable(pooltab[pooltab_cnt - 1].first + pooltab[pooltab_cnt
- 1].total_literals, total_ltr - pooltab[pooltab_cnt - 1].first - 1);
                        pooltab_cnt++;
                     }
                     break;
                  case "equ":
                     pos = 4;
                     bw.write("\t(AD," + pos +")");
                     String prev_token = tokens[k - 1];
```

```
                int pos1 = search(prev_token, symb_table, total_symb);
                pos = search(tokens[++k], symb_table, total_symb);
                symb_table[pos1].addr = symb_table[pos].addr;
                bw.write("\t(S," + (pos + 1) + ")");
                break;
            }
            if (pos == -1) {
               pos = search(tokens[k], IS);
               if (pos != -1) {
                  bw.write("\t(IS," + (pos) +")");
                  optab[optab_cnt++] = new Obj(tokens[k], pos);
               } else {
                  pos = search(tokens[k], DL); // DC/DS
                  if (pos != -1)
                  {
                     if(pos == 0)
                       { f = 1;}
                      bw.write("\t(DL," + (pos + 1) + ")");
                      optab[optab_cnt++] = new Obj(tokens[k], pos);
                      fill_addr = true;

                  } else if (tokens[k].matches("[a-zA-Z]+:")) { //label
                     pos = search(tokens[k], symb_table, total_symb);
                     if (pos == -1) {
                        symb_table[total_symb++] = new Obj(tokens[k].substring(0, tokens[k].length() - 1), loc
- 1);
                        bw.write("\t(S," + total_symb + ")");
                        pos = total_symb;
                     }
                  }
               }
            }

            if (pos == -1) {
               pos = search(tokens[k], REG);
               if (pos != -1) {
                  bw.write("\t(RG," + (pos + 1) +")"); //register
               } else {
                  if (tokens[k].matches("='(\\d+)'")) { //literal
                     String s = tokens[k].substring(2, 3);
                     literal_table[total_ltr++] = new Obj(s, 0);
                     bw.write("\t(L," + total_ltr + ")");
                  }
                  else if (tokens[k].matches("\\d+") || tokens[k].matches("\\d+H") || tokens[k].matches("\\d+h")
|| tokens[k].matches("\\d+D") || tokens[k].matches("\\d+d")) { //constant
                     bw.write("\t(C," + tokens[k] + ")");
                     temp1 = Integer.parseInt(tokens[k]);
                  }
                  else {
                     pos = search(tokens[k], symb_table, total_symb);
                     if (fill_addr && pos != -1 && symb_table[pos].addr == 0) {
                        symb_table[pos].addr = loc - 1;
                        fill_addr = false;
                     } else if (pos == -1) {
                        symb_table[total_symb++] = new Obj(tokens[k], 0);
```

```
                        bw.write("\t (S," + total_symb + ")");
                    } else {
                        bw.write("\t(S," + pos + ")");
                    }
                }
            }
        }
    }

    pooltab[pooltab_cnt] = new Pooltable(pooltab[pooltab_cnt - 1].first + pooltab[pooltab_cnt -
1].total_literals, total_ltr - pooltab[pooltab_cnt - 1].first - 2);
    pooltab_cnt++;

    System.out.println("\n*LITERAL TABLE*");
    System.out.println("\nIndex\tLITERAL\tADDRESS");
    for (int i = 0; i < total_ltr; i++) {
        if (literal_table[i].addr == 0) {
            literal_table[i].addr = loc++;
        }
        System.out.println((i) + "\t" + literal_table[i].name + "\t" + literal_table[i].addr);
    }

    System.out.println("\n*SYMBOL TABLE*");
    System.out.println("\nSYMBOL\tADDRESS");
    for (int i = 0; i < total_symb; i++) {
        System.out.println(symb_table[i].name + "\t" + symb_table[i].addr);
    }

    System.out.println("\n*POOL TABLE*");
    System.out.println("\nPOOL\tTOTAL LITERALS");
    for (int i = 0; i < pooltab_cnt; i++) {
        System.out.println(pooltab[i].first + "\t" + pooltab[i].total_literals);
    }

    System.out.println("\n*OPTABLE*");
    System.out.println("\nMNEMONIC\tOPCODE");
    for (int i = 0; i < IS.length; i++) {
        System.out.println(IS[i] + "\t\t" + i);
    }

    br.close();
    bw.close();

} catch (Exception e) {
    System.out.println("error while reading the file");
    e.printStackTrace();
}

try {
    BufferedReader br = new BufferedReader(new FileReader("output.txt"));
    System.out.println("\n*Output1.txt\n");
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
```

```
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static int search(String token, String[] list) {
        for (int i = 0; i < list.length; i++) {
            if (token.equalsIgnoreCase(list[i])) {
                return i;
            }
        }
        return -1;
    }

    public static int search(String token, Obj[] list, int cnt) {
        for (int i = 0; i < cnt; i++) {
            if (token.equalsIgnoreCase(list[i].name)) {
                return i;
            }
        }
        return -1;
    }
}
```

OUTPUT:

```
PRACTICAL\CODE\Pass1 on  main [!?] via ☕ v24.0.2
❯ javac Pass1.java

PRACTICAL\CODE\Pass1 on  main [!?] via ☕ v24.0.2
❯ java Pass1.java

*LITERAL TABLE*

Index   LITERAL ADDRESS
0       5       102
1       8       105
2       8       106
3       7       122
4       8       123

*SYMBOL TABLE*

SYMBOL  ADDRESS
up      102
a       109
b       111
c       112
next    102

*POOL TABLE*

POOL    TOTAL LITERALS
0       1
1       2
3       2

*OPTABLE*

MNEMONIC        OPCODE
stop            0
add             1
sub             2
mult            3
mover           4
movem           5
comp            6
bc              7
div             8
read            9
```

*Output1.txt

```
        (AD,1)  (C,100)
100     (IS,4)  (RG,1)  (C,05)
101     (IS,4)  (RG,2)  (C,10)
102     (S,1)   (IS,1)  (RG,1)  (RG,2)
103     (IS,5)   (S,2)  (L,1)
104     (IS,3)  (RG,1)  (S,1)
105      (AD,3) (C,102)
102     (AD,5)
        (DL,2)   (C,5)
103     (IS,5)   (S,3)  (L,2)
104     (IS,5)   (S,4)  (L,3)
105     (AD,5)
        (DL,2)   (C,8)
106     (DL,2)   (C,8)
107     (IS,5)  (S,2)   (L,4)
108     (IS,5)  (S,3)   (L,5)
109     (DL,1)  (C,02)
111     (DL,2)  (C,10)
112     (DL,1)  (C,09)
121      (S,5)  (AD,4)  (S,1)
122     (AD,2)
        (DL,2)   (C,7)
123     (DL,2)   (C,8)
124
```