NAME: - Shaikh Areeba Mohammed Ismail
ROLLNO: - 46
CLASS/BATCH: - TE- B-2

Practical No-1

//Design suitable Data structures and implement Pass-I of a two-pass assembler for pseudomachine. Implementation should consist of a few instructions from each category and few assembler directives. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.

```
import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter; import

java.io.IOException; import

java.util.HashMap;


class symbol {

String      sym;

int addr;

}


class littab {

String lit;     int

addr;

}


public class Pass1 {


   HashMap<String, Integer> OPTAB = new HashMap<String, Integer>();  // mnemonic
table

   HashMap<String, Integer> REGTAB = new HashMap<String, Integer>();  // register table

   HashMap<String, Integer> CONDTAB = new HashMap<String, Integer>(); // condition
table
```

```
    HashMap<String, Integer> ADTAB = new HashMap<String, Integer>();  // assembly
directive table

    int MAX = 20;

    symbol SYMTAB[] = new symbol[MAX];  // Symbol Table
littab LITTAB[] = new littab[MAX];  // literal table

    String buffer;      int lc, litcnt = 0, poolcnt = 0, proc_lit =
0, symcount = 0;

    Pass1() {

        // initialize all tables with their associated codes
initialize_OPTAB();          initialize_REGTAB();
initialize_CONDTAB();          initialize_ADTAB();

        // assign memory in advance for symbols
for (int i = 0; i < MAX; i++) {
SYMTAB[i] = new symbol();
        }

        // assign memory in advance for literals
for (int i = 0; i < MAX; i++) {
            LITTAB[i] = new littab();
        }
    }

    public void initialize_OPTAB() {
        OPTAB.put("STOP", 0);
        OPTAB.put("ADD", 1);
        OPTAB.put("SUB", 2);
        OPTAB.put("MULT", 3);
        OPTAB.put("MOVER", 4);
        OPTAB.put("MOVEM", 5);
        OPTAB.put("COMP", 6);
        OPTAB.put("BC", 7);
        OPTAB.put("DIV", 8);
        OPTAB.put("READ", 9);
        OPTAB.put("PRINT", 10);
    }

    public void initialize_REGTAB() {
REGTAB.put("AREG", 1);
        REGTAB.put("BREG", 2);
        REGTAB.put("CREG", 3);
        REGTAB.put("DREG", 4);
    }

    public void initialize_CONDTAB() {
CONDTAB.put("LT", 1);
        CONDTAB.put("LE", 2);
        CONDTAB.put("EQ", 3);
CONDTAB.put("GT", 4);
        CONDTAB.put("GE", 5);
        CONDTAB.put("ANY", 6);
    }

    public void initialize_ADTAB() {
ADTAB.put("START", 1);
        ADTAB.put("END", 2);
        ADTAB.put("ORIGIN", 3);          ADTAB.put("EQU", 4);
        ADTAB.put("LTORG", 5);
    }
```

```
   public int search_OPTAB(String str) {
if (OPTAB.containsKey(str)) {
return OPTAB.get(str);
      } else
return -1;
   }

   public int search_REGTAB(String str) {
if (REGTAB.containsKey(str)) {
return REGTAB.get(str);
      } else
return -1;
   }

   public int search_CONDTAB(String str) {
if (CONDTAB.containsKey(str)) {
return CONDTAB.get(str);
      } else
return -1;
   }

   public int search_ADTAB(String str) {
if (ADTAB.containsKey(str)) {
return ADTAB.get(str);
      } else
return -1;
   }

   public int search_symbol(String str) {
int i;
```

```
    for (i = 0; i < symcount; i++) {
if (str.equals(SYMTAB[i].sym))
        return i;
    }
return -1;
  }

  void passone() throws IOException {
    int n, i = 0, j = 0, p, k;

    FileReader source_file = new FileReader("input.txt");
    BufferedReader fs = new BufferedReader(source_file);

    FileWriter ic_file = new FileWriter("ic.txt");
    BufferedWriter ft = new BufferedWriter(ic_file);

    while ((buffer = fs.readLine()) != null) {
String[] tokens = buffer.split(" |\\,");              n
= tokens.length; // number of tokens

      switch (n) {

        case 1: // Instruction with length 1

          // STOP
          i = search_OPTAB(tokens[0]);                    if (i == 0) {
ft.write("(IS," + String.format("%02d)", i));// %2d : to write 2 digit number
lc++;                        break;

          }

          // END, LTORG ALLOCATE SPACE FOR LITERALS
i = search_ADTAB(tokens[0]);                    if (i == 2 || i == 5) {
```

```
            for (j = proc_lit; j < litcnt; j++) {
              LITTAB[j].addr = lc++;
            }
proc_lit = litcnt;

            ft.write("(AD," + String.format("%02d)", i));
          }
break;

        case 2: // Instruction with length 2

          // START, ORIGIN ASSIGN NEW LC
i = search_ADTAB(tokens[0]);                    if (i ==
1 || i == 3) {                          lc =
Integer.parseInt(tokens[1]);
            ft.write("(AD," + String.format("%02d)", i) + " (C," + tokens[1] + ")");
break;
          }

          // READ or PRINT A
i = search_OPTAB(tokens[0]);                    if
(i == 9 || i == 10) {                           p =
search_symbol(tokens[1]);                       if
(p == -1) {
              SYMTAB[symcount].sym = tokens[1];

symcount++;
ft.write("(IS," +
String.format("%02d)", i) + "
(S," + String.format("%02d)",
symcount));
            } else {
```

```
                        ft.write("(IS," + String.format("%02d", i) + " (S," +
String.format("%02d", p));
                    }
lc++;
break;
                }

                // NEXT STOP                    i =
search_OPTAB(tokens[1]);                    if (i
== 0) {                        p =
search_symbol(tokens[0]);                        if
(p == -1) {
                        SYMTAB[symcount].sym = tokens[1];
SYMTAB[symcount].addr = lc;                        symcount++;
                    } else {
                        SYMTAB[p].addr = lc;
                    }
lc++;                    }
break;

             case 3: // Instruction with length 3

                i = search_OPTAB(tokens[0]);
if (i >= 1 && i <= 8) {
lc++;

                    if (i == 7)
                        k = search_CONDTAB(tokens[1]); // BC GT,LOOP                        else
                        k = search_REGTAB(tokens[1]); // SUB AREG,='1'

                    if (tokens[2].charAt(0) == '=') // MOVER AREG,='5'
                    {
                      String teemp = tokens[2].substring(2, 3); // extract the literal (5) from the
format ='5'
                      LITTAB[litcnt].lit = teemp;


litcnt++;
                      ft.write("(IS," + String.format("%02d) (", i) + k + ")(L," +
String.format("%02d)", litcnt));


                   } else // MOVER AREG,A
                   {
                      p = search_symbol(tokens[2]);
if (p == -1) {


                         SYMTAB[symcount].sym = tokens[2];
symcount++;


                         ft.write("(IS," + String.format("%02d) (", i) + k + ")(S," +
String.format("%02d)", symcount));
                      } else {
                         ft.write("(IS," + String.format("%02d) (", i) + k + ")(S," +
String.format("%02d)", symcount));


                      }
break;
                   }
                }


                // A DS 2
                if (tokens[1].equals("DS")) {
p = search_symbol(tokens[0]);
if (p == -1) {
                      SYMTAB[symcount].sym = tokens[0];
SYMTAB[symcount].addr = lc;                          symcount++;
```

```
            ft.write("(DL,02) (C," + tokens[2] + ")");

          } else {
            SYMTAB[p].addr = lc;
ft.write("(DL,02) (C," + tokens[2] + ")");

          }
          lc = lc + Integer.parseInt(tokens[2]);
break;
        }

        // ONE DC 1                if
(tokens[1].equals("DC")) {                 p
= search_symbol(tokens[0]);
if (p == -1) {
            SYMTAB[symcount].sym = tokens[0];
SYMTAB[symcount].addr = lc;                  symcount++;

            ft.write("(DL,01) (C," + tokens[2]);

          } else {
            SYMTAB[p].addr = lc;
ft.write("(DL,01) (C," + tokens[2]);                 }
break;
        }

        // check for EQU               i =
search_ADTAB(tokens[1]);              if (i
== 4) {               p =
search_symbol(tokens[0]);             j =
search_symbol(tokens[2]);
```

```
                    if (p == -1 && j != -1) {
                        SYMTAB[symcount].sym = tokens[0];
                        SYMTAB[symcount++].addr = SYMTAB[j].addr;
                    }
                    if (p != -1 && j == -1) {
                        SYMTAB[symcount].sym = tokens[2];
                        SYMTAB[symcount++].addr = SYMTAB[p].addr;
                    }                        if (p != -1
&& j != -1) {                            if
((SYMTAB[j].addr) != 0)
                            SYMTAB[p].addr = SYMTAB[j].addr;
else
                            SYMTAB[j].addr = SYMTAB[j].addr;


                    }
                    lc--; // since lc is incremented after switch and there is no processing for ic
                }


                break;
              case
4:
                i = search_OPTAB(tokens[1]);


                if (i >= 1 && i <= 8) {
p = search_symbol(tokens[0]);
if (p == -1) {
                        SYMTAB[symcount].sym = tokens[0];
SYMTAB[symcount].addr = lc;                              symcount++;
                    } else {
                        SYMTAB[p].addr = lc;
                    }
                if (i ==
7)
                  k = search_CONDTAB(tokens[2]);
else
                  k = search_REGTAB(tokens[2]);

                if (tokens[3].charAt(0) == '=') {
                  String teemp = tokens[3].substring(2, 3); // extract the literal (5) from the
format ='5'

                  LITTAB[litcnt].lit = teemp;
litcnt++;
                  ft.write("(IS," + String.format("%02d) (", i) + k + ")(L," +
String.format("%02d)", litcnt));
                } else {                  p =
search_symbol(tokens[3]);

                  if (p == -1) {
                    SYMTAB[symcount].sym = tokens[3];
symcount++;
                    ft.write("(IS," + String.format("%02d) (", i) + k + ")(S," +
String.format("%02d)", symcount));
                  } else {
                    ft.write("(IS," + String.format("%02d) (", i) + k + ")(S," +
String.format("%02d)", (p + 1)));
                  }
                }
}
break;
        }
ft.write("\n");
      }
ft.close();
```

```
  }

  //
==================================================================
=====================
  void print_littab() {
    for (int i = 0; i < litcnt; i++) {
      System.out.println(LITTAB[i].lit + " \t" + LITTAB[i].addr);
    }
  }

  //
==================================================================
=====================
void print_symtab() {
    for (int i = 0; i < symcount; i++) {
      System.out.println(SYMTAB[i].sym + " \t" + SYMTAB[i].addr);
    }
  }

  //
==================================================================
=====================
  void print_srcfile() throws IOException {
    FileReader source_file = new FileReader("input.txt");
    BufferedReader fs = new BufferedReader(source_file);

    String buffer;
    while ((buffer = fs.readLine()) != null) {
      System.out.println(buffer);
    }

fs.close();
  }
  //
============================================================================
========================

  void print_icfile() throws IOException {
    FileReader source_file = new FileReader("ic.txt");
    BufferedReader fs = new BufferedReader(source_file);


    String buffer;
    while ((buffer = fs.readLine()) != null) {
      System.out.println(buffer);
    }


fs.close();
  }
  public static void main(String[] args) throws IOException {
    Pass1 obj = new Pass1(); // initialize OPTAB, REGTAB, CONDTAB, ADTAB


    obj.passone();


    System.out.println("SOURCE CODE\n");
    obj.print_srcfile();
System.out.println("\n\n*****************************************************");

    System.out.println("\n\nINTERMEDIATE CODE\n");
obj.print_icfile();
    System.out.println("\n\n*****************************************************");
    System.out.println("\n\nSYMBOL TABLE");
    System.out.println("==================");
    System.out.println("Symbol\tAddress");
System.out.println("==================");          obj.print_symtab();
System.out.println("\n\n****************************************************");
System.out.println("\n\nLITERAL TABLE"); System.out.println("=================");
        System.out.println("Literal\tAddress");
        System.out.println("=================");
obj.print_littab();
    }
}
```

Input file-

```
START 200
READ A
LOOP MOVER AREG,A
SUB AREG,='1'
BC GT,LOOP
STOP
LTORG
A DS 1
END
```

Intermediate code-

```
(AD,01) (C,200)
(IS,09) (S,01)
(IS,04) (1)(S,01)
(IS,02) (1)(L,01)
(IS,07) (4)(S,02)
(IS,00)
(AD,05)
(DL,02) (C,1)
(AD,02)
```

Output-

```
gescoe@gescoe-OptiPlex-3010:-/Desktop/TE-46$ javac pass1.java
gescoe@gescoe-OptiPlex-3010:-/Desktop/TE-46$ java pass1 input.txt

SOURCE CODE
START 200
READ A
LOOP MOVER AREG,A
SUB AREG,='1'
BC GT,LOOP
STOP
LTORG
A DS 1
END



**********************************************

INTERMEDIATE CODE
(AD,01) (C,200)
(IS,09) (S,01)
(IS,04) (1)(S,01)
(IS,02) (1)(L,01)
(IS,07) (4)(S,02)
(IS,00)
(AD,05)
(DL,02) (C,1)
(AD,02)
*********************************************
SYMBOL TABLE
================
Symbol      Address
=================
A                  205
LOOP          201
************************************************
LITERAL TABLE
=================
Literal        Address
=================
1                   204
```