ASSIGNMENT NO.1

(PASS I Assembler Code)

Input Files-

1)input.txt

```
        START   100
        MOVER   AREG,B
        ADD     BREG,='6'
        MOVEM   AREG,A
        SUB     CREG,='1'
        LTORG
        ADD     DREG,='5'
A       DS      10
        LTORG
        SUB     AREG,='1'
B       DC      1
C       DC      1
        END
```

2)mot.txt

```
START   AD  01  0
END     AD  02  0
LTORG   AD  05  0
ADD     IS  01  1
SUB     IS  02  1
MULT    IS  03  1
MOVER   IS  04  1
MOVEM   IS  05  1
DS      DL  01  0
DC      DL  02  1
```

3)Prog.java

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.StringTokenizer;
```

```
import java.io.IOException;

//Class for mnemonic contains mnemonic, class, opcode and length
class Tuple {
   String mnemonic, mclass, opcode;
   int length;

   Tuple(String s1, String s2, String s3, String s4) {
      mnemonic = s1;
      mclass = s2;
      opcode = s3;
      length = Integer.parseInt(s4.trim());
   }
}

public class Prog {
   //Data structures for Assembler Pass-I, Hashmaps for Mnemonic and
registers, ArrayLists for Literals and Symbols
        public static HashMap<String, Tuple> map = new HashMap<String,
Tuple>();
   public static HashMap<String, Integer> registers = new HashMap<String,
Integer>();
   public static ArrayList<String> literals = new ArrayList<String>();
   public static ArrayList<String> symbols = new ArrayList<String>();

   //Constructor defining registers
   Prog() {
      registers.put("AREG", 1);
      registers.put("BREG", 2);
      registers.put("CREG", 3);
      registers.put("DREG", 4);
   }

   public static void mapper() {
      try {
         String newSt = "";
         FileInputStream input = new
FileInputStream("/home/student/Downloads/snehal/mot.txt");
         int i = input.read();
         while (i != -1) {
```

```
            newSt += (char) i;
            i = input.read();
         }
         input.close();
         StringTokenizer st = new StringTokenizer(newSt, " ");
         String sst = "";
         while (st.hasMoreTokens()) {
            sst += st.nextToken() + " ";
         }
         sst = sst.toString();
         String[] arr2 = sst.split("\n");
         for (int j = 0; j < arr2.length; j++) {
            map.put(arr2[j].split(" ")[0], new Tuple(arr2[j].split(" ")[0],
arr2[j].split(" ")[1],
                  arr2[j].split(" ")[2], arr2[j].split(" ")[3]));
         }
      } catch (Exception e) {
         System.out.println(e);
      }

   }

   public static String[] inputFileRead() {
      String newSt = "";
      String[] arr2 = {};
      try {
        //Reading the input file
         FileInputStream input = new
FileInputStream("/home/student/Downloads/snehal/input.txt");
         int i = input.read();
         while (i != -1) {
            newSt += (char) i;
            i = input.read();
         }
         input.close();
         //Tokenization line by line
         StringTokenizer st = new StringTokenizer(newSt, " ");
         String sst = "";
         while (st.hasMoreTokens()) {
            sst += st.nextToken() + " ";
```

```
            }
            sst = sst.toString();
            arr2 = sst.split("\n");
        } catch (Exception e) {
            System.out.println("Something went wrong!" + e);
        }
        return arr2;
    }

    public static void intermediateCoder() {
        mapper();
        String[] inputArr = inputFileRead();

        String sst = "";
        String forLiteral = "";
        String forSymbol = "";
        int addressStart = 0, address = 0;
        int addressCounter = 1; //Location counter
        for (int i = 0; i < inputArr.length; i++) {
            inputArr[i] = inputArr[i].trim();
            inputArr[i] = inputArr[i].replaceAll(",",", ");

            Tuple value = map.get(inputArr[i].split(" ")[0]);
            if (value==null){
                value = map.get(inputArr[i].split(" ")[1]);
            }

            String mclass = value.mclass;
            String opcode = value.opcode;
            int length = value.length;
            //For handling Assembler Directives
            if (value.mclass.equalsIgnoreCase("AD")) {
                if(inputArr[i].split(" ").length>1){
                sst = sst + "(" + String.format("%s, %s", mclass, opcode) + ")" + "\t"
+ "("
                        + String.format("C,%s", inputArr[i].split(" ")[1].trim() + ")" +
"\n");
                System.out.println(inputArr[i].split(" ")[1].trim());
                addressStart = Integer.parseInt(inputArr[i].split(" ")[1].trim());
                }
```

```
            else{
               sst = sst + "(" + String.format("%s, %s", mclass, opcode) + ")" +
"\n";
               System.out.println(sst);
            }

         } else if(value.mclass.equalsIgnoreCase("DL")){
            sst = sst + "(" + String.format("%s, %s", mclass, opcode) + ")" + "\t"
+ "("
                  + String.format("%s", inputArr[i].split(" ")[2].trim() + ")" +
"\n");
            System.out.println(sst);

         } else {

            // Literal Case
            address = addressStart;
            if (inputArr[i].split(" ")[2].startsWith("=")) {
               //System.out.println("Here1");
               System.out.println(inputArr[i].split(" ")[2]);
               literals.add(inputArr[i].split(" ")[2].split("=")[1]);
               sst = sst + "(" + String.format("%s, %s", mclass, opcode) + ")" +
"\t" + "(" + String.format("%s",
                     registers.get(inputArr[i].split(" ")[1].split(",")[0].trim()) + ")"
+ "("
                           + String.format("L,%s",
literals.indexOf(inputArr[i].split(" ")[2].split("=")[1]))
                           + ")" + "\n");
               address = addressCounter + address;
               forLiteral = forLiteral + inputArr[i].split(" ")[2].trim() + "\t" +
String.format("%s", address)
                     + "\n";
               addressCounter++;
            }
            // Symbol case
            else {
               System.out.println(inputArr[i].split(" ")[2]);
               symbols.add(inputArr[i].split(" ")[2]);
               address = addressCounter + address;
```

```
                sst = sst + "(" + String.format("%s, %s", mclass, opcode) + ")" +
"\t" + "("
                        + String.format("%s",
                                registers.get(inputArr[i].split(" ")[1].split(",")[0].trim()) +
")" + "("
                                        + String.format("S,%s",
symbols.indexOf(inputArr[i].split(" ")[2])) + ")"
                                        + "\n");
                forSymbol = forSymbol + inputArr[i].split(" ")[2] + "\t" +
String.format("%s", address) + "\t"
                        + String.format("%s", length) + "\n";
                addressCounter++;
            }
        }
      }
      sst = sst.toString();
      forLiteral = forLiteral.toString();
      forSymbol = forSymbol.toString();
      try {
         FileOutputStream output = new FileOutputStream("./out.txt");
         FileOutputStream literalOut = new
FileOutputStream("./literalTable.txt");
         FileOutputStream symbolOut = new
FileOutputStream("./symbolTable.txt");
         output.write(sst.getBytes());
         literalOut.write(forLiteral.getBytes());
         symbolOut.write(forSymbol.getBytes());
         output.close();
         literalOut.close();
         symbolOut.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
   public static void main(String[] args) {
      Prog pg = new Prog();
      intermediateCoder();
   }
}
```

Output Files:-

1)literalTable.txt

='6'	102
='1'	104
='5'	105
='1'	106

2)Out.txt

(AD, 01)	(C,100)
(IS, 04)	(1)(S,0)
(IS, 01)	(2)(L,0)
(IS, 05)	(1)(S,1)
(IS, 02)	(3)(L,1)
(AD, 05)	
(IS, 01)	(4)(L,2)
(DL, 01)	(10)
(AD, 05)	
(IS, 02)	(1)(L,1)
(DL, 02)	(1)
(DL, 02)	(1)
(AD, 02)	

3)symbolTable.txt

B	101	1
A	103	1