Input:-

```
START 100
A DC '5'
B DC '10'
MOVER AREG, A
ADD AREG, ='15'
MOVEM AREG, B
LTORG
PRINT A
STOP
END
```

Ass1.java:-

```
// Save this file as Ass1.java

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

// Mnemonic Table Entry
class MnemonicTable {
    public String mnemonic;
    public String opcode;
    public int num;

    public MnemonicTable(String mnemonic, String opcode, int num) {
        this.mnemonic = mnemonic;
        this.opcode = opcode;
        this.num = num;
    }
}

// Main Assembler Class
public class Ass1 {

    Map<String, MnemonicTable> is = new Hashtable<>();
    ArrayList<String> symtab = new ArrayList<>();
    ArrayList<Integer> symaddr = new ArrayList<>();
    ArrayList<String> littab = new ArrayList<>();
    ArrayList<Integer> litaddr = new ArrayList<>();
    ArrayList<Integer> pooltab = new ArrayList<>();
    int LC = 0;

    // Create Instruction Set Table
    public void createIS() {
        is.put("STOP", new MnemonicTable("STOP", "00", 0));
        is.put("ADD", new MnemonicTable("ADD", "01", 0));
        is.put("SUB", new MnemonicTable("SUB", "02", 0));
        is.put("MULT", new MnemonicTable("MULT", "03", 0));
        is.put("MOVER", new MnemonicTable("MOVER", "04", 0));
        is.put("MOVEM", new MnemonicTable("MOVEM", "05", 0));
        is.put("COMP", new MnemonicTable("COMP", "06", 0));
        is.put("BC", new MnemonicTable("BC", "07", 0));
        is.put("DIV", new MnemonicTable("DIV", "08", 0));
        is.put("READ", new MnemonicTable("READ", "09", 0));
        is.put("PRINT", new MnemonicTable("PRINT", "10", 0));
    }

    // Pass I → Generate Intermediate Code
    public void generateIC(String filename) throws Exception {
        BufferedWriter wr = new BufferedWriter(new FileWriter("ic.txt"));
        BufferedReader br = new BufferedReader(new FileReader(filename));
```

```
        String line = " ";
        pooltab.add(0, 0);

        wr.write("---------------------\n Intermediate
Code\n---------------------\n");

        while ((line = br.readLine()) != null) {
            String[] split = line.split("\\s+");

            // Symbol Handling
            if (split[0].length() > 0 && !split[0].equals("START")) {
                if (!symtab.contains(split[0])) {
                    symtab.add(split[0]);
                    symaddr.add(LC);
                } else {
                    int index = symtab.indexOf(split[0]);
                    symaddr.set(index, LC);
                }
            }

            // Directive Handling
            if (split.length > 1) {
                if (split[1].equals("START")) {
                    LC = Integer.parseInt(split[2]);
                    wr.write("(AD,01)(C," + split[2] + ") \n");
                } else if (split[1].equals("ORIGIN")) {
                    LC = getAddress(split[2]);
                } else if (split[1].equals("EQU")) {
                    int addr = getAddress(split[2]);
                    if (!symtab.contains(split[0])) {
                        symtab.add(split[0]);
                        symaddr.add(addr);
                    } else {
                        int index = symtab.indexOf(split[0]);
                        symaddr.set(index, addr);
                    }
                } else if (split[1].equals("LTORG") || split[1].equals("END")) {
                    for (int i = pooltab.get(pooltab.size() - 1); i <
littab.size(); i++) {
                        if (litaddr.get(i) == 0) {
                            litaddr.set(i, LC);
                            LC++;
                        }
                    }
                    if (!split[1].equals("END")) {
                        pooltab.add(littab.size());
                        wr.write("\n(AD,05)\n");
                    } else {
                        wr.write("(AD,04)\n");
                    }
                } else if (split[1].contains("DS")) {
                    LC += Integer.parseInt(split[2]);
                    wr.write("(DL,01) (C," + split[2] + ") \n");
                } else if (split[1].equals("DC")) {
                    LC++;
                    wr.write("\n(DL,02) (C," + split[2].replace("'", "") + ")
\n");
                } else if (is.containsKey(split[1])) {
                    wr.write("(IS," + is.get(split[1]).opcode + ") ");
                    if (split.length > 2) {
                        String reg = split[2].replace(",", "");
                        if (reg.equals("AREG")) wr.write("(1) ");
                        else if (reg.equals("BREG")) wr.write("(2) ");
                        else if (reg.equals("CREG")) wr.write("(3) ");
```

```
                        else if (reg.equals("DREG")) wr.write("(4) ");
                        else {
                            if (!symtab.contains(reg)) {
                                symtab.add(reg);
                                symaddr.add(0);
                            }
                            wr.write("(S," + symtab.indexOf(reg) + ") ");
                        }
                    }
                    if (split.length > 3) {
                        if (split[3].contains("=")) {
                            String norm = split[3].replace("=", "").replace("'",
"");
                            if (!littab.contains(norm)) {
                                littab.add(norm);
                                litaddr.add(0);
                            }
                            wr.write("(L," + littab.indexOf(norm) + ")");
                        } else {
                            if (!symtab.contains(split[3])) {
                                symtab.add(split[3]);
                                symaddr.add(0);
                            }
                            wr.write("(S," + symtab.indexOf(split[3]) + ")");
                        }
                    }
                    wr.write("\n");
                    LC++;
                }
            }
        }

        wr.flush();
        br.close();
        wr.close();

        // Symbol Table
        BufferedWriter br1 = new BufferedWriter(new FileWriter("sym.txt"));
        br1.write("-------------------\n Symbol Table\n-------------------
\nSymbol Address\n");
        for (int i = 0; i < symtab.size(); i++) {
            br1.write(" " + symtab.get(i) + " " + symaddr.get(i) + "\n");
        }
        br1.flush();
        br1.close();

        // Literal Table
        BufferedWriter br2 = new BufferedWriter(new FileWriter("lit.txt"));
        br2.write("-----------------------\n Literal
Table\n-----------------------\nLiteral Address\n");
        for (int i = 0; i < littab.size(); i++) {
            br2.write("='" + littab.get(i) + "' " + litaddr.get(i) + "\n");
        }
        br2.flush();
        br2.close();

        // Pool Table
        BufferedWriter br3 = new BufferedWriter(new FileWriter("pool.txt"));
        br3.write("----------------------------\n Pool
Table\n----------------------------\nPool Index Literal Index\n");
        for (int i = 0; i < pooltab.size(); i++) {
            br3.write(" " + i + " " + pooltab.get(i) + "\n");
        }
        br3.flush();
```

```
        br3.close();
    }

    // For ORIGIN / EQU
    private int getAddress(String string) {
        int temp = 0;
        if (string.contains("+")) {
            String[] sp = string.split("\\+");
            int ad = symaddr.get(symtab.indexOf(sp[0]));
            temp = ad + Integer.parseInt(sp[1]);
        } else if (string.contains("-")) {
            String[] sp = string.split("\\-");
            int ad = symaddr.get(symtab.indexOf(sp[0]));
            temp = ad - Integer.parseInt(sp[1]);
        } else {
            temp = symaddr.get(symtab.indexOf(string));
        }
        return temp;
    }

    // Pass II → Generate Machine Code
    public void generateMachineCode() throws Exception {
        BufferedReader ic = new BufferedReader(new FileReader("ic.txt"));
        BufferedWriter mc = new BufferedWriter(new FileWriter("machine.txt"));
        System.out.println("\n=== PASS 2 OUTPUT ===");
        mc.write("=== MACHINE CODE ===\n");

        String line;
        while ((line = ic.readLine()) != null) {
            if (!line.contains("IS")) continue;

            String[] parts = line.trim().split("\\s+");
            int opcode = Integer.parseInt(parts[0].replaceAll("[^0-9]", ""));
            String reg = "0", operand = "000";

            if (parts.length > 1 && parts[1].contains("(")) {
                reg = parts[1].replaceAll("[^0-9]", "");
            }
            if (parts.length > 2) {
                String op = parts[2];
                if (op.contains("S")) {
                    int idx = Integer.parseInt(op.replaceAll("[^0-9]", ""));
                    operand = String.format("%03d", symaddr.get(idx));
                } else if (op.contains("L")) {
                    int idx = Integer.parseInt(op.replaceAll("[^0-9]", ""));
                    operand = String.format("%03d", litaddr.get(idx));
                } else {
                    operand = String.format("%03d",
Integer.parseInt(op.replaceAll("[^0-9]", "")));
                }
            }

            String code = String.format("%02d %s %s", opcode, reg, operand);
            System.out.println(code);
            mc.write(code + "\n");
        }

        ic.close();
        mc.close();
    }

    // Main
    public static void main(String[] args) throws Exception {
        Ass1 p = new Ass1();
```

```
        p.createIS();
        p.generateIC("input9.asm");   // use input9.asm as input
        p.generateMachineCode();
    }
}

swaraj@swaraj-VirtualBox:~/LP-1$ javac Ass1.java
swaraj@swaraj-VirtualBox:~/LP-1$ java Ass1

Output :-
------------------- Symbol Table -------------------
Symbol Address
A 100
B 101
----------------------- Literal Table -----------------------
Literal Address
='15' 102
---------------------------- Pool Table ----------------------------
Pool Index Literal Index
0 0
--------------------- Intermediate Code ---------------------
(AD,01)(C,100)
(DL,02)(C,5)
(DL,02)(C,10)
(IS,04)(1)(S,0)
(IS,01)(1)(L,0)
(IS,05)(1)(S,1)
(AD,05)
(IS,10)(S,0)
(IS,00)
(AD,04)
-----------------Machine Code----------------
04 1 100
01 1 102
05 1 101
10 0 100
00 0 000
```