Code:-

```java
package com.muthadevs;

import java.io.BufferedReader;
import java.io.*;
import java.io.IOException;
import java.util.*;

public class Main {
    public static void main(String[] args) {

        BufferedReader br = null;
        FileReader fr = null;

        FileWriter fw = null;
        BufferedWriter bw = null;

        try {
            String inputfilename = "E:\\pass1_assembler\\INPUT\\Input.asm";
            fr = new FileReader(inputfilename);
            br = new BufferedReader(fr);

            String OUTPUTFILENAME = "E:\\pass1_assembler\\OUTPUT\\IC.txt";
            fw = new FileWriter(OUTPUTFILENAME);
            bw = new BufferedWriter(fw);

            Hashtable<String, String> is = new Hashtable<String, String>();
            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>();
            dl.put("DC", "01");
            dl.put("DS", "02");

            Hashtable<String, String> ad = new Hashtable<String, String>();

            ad.put("START", "01");
            ad.put("END", "02");
            ad.put("ORIGIN", "03");
            ad.put("EQU", "04");
            ad.put("LTORG", "05");

            Hashtable<String, String> symtab = new Hashtable<String, String>();
            Hashtable<String, String> littab = new Hashtable<String, String>();
            ArrayList<Integer> pooltab=new ArrayList<Integer>();

            String sCurrentLine;
            int locptr = 0;
            int litptr = 1;
            int symptr = 1;
            int pooltabptr = 1;

            sCurrentLine = br.readLine();

            String s1 = sCurrentLine.split(" ")[1];
            if (s1.equals("START")) {
                bw.write("AD \t 01 \t");
                String s2 = sCurrentLine.split(" ")[2];
                bw.write("C \t" + s2 + "\n");
                locptr = Integer.parseInt(s2);
            }

            while ((sCurrentLine = br.readLine()) != null) {
                int mind_the_LC = 0;
                String type = null;

                int flag2 = 0;    //checks whether addr is assigned to current symbol

                String s = sCurrentLine.split(" |\\,")[0]; //consider the first word in the
line

                for (Map.Entry m : symtab.entrySet()) {            //allocating addr to
arrived symbols
                if (s.equals(m.getKey())) {
                        m.setValue(locptr);
                        flag2 = 1;
                    }
                }
                if (s.length() != 0 && flag2 == 0) {      //if current string is not " " or addr
is not assigned,
                    //then the current string must be a new symbol.
                    symtab.put(s, String.valueOf(locptr));
                    symptr++;
                }

                int isOpcode = 0;     //checks whether current word is an opcode or not

                s = sCurrentLine.split(" |\\,")[1];          //consider the second word in the
line

                for (Map.Entry m : is.entrySet()) {
                    if (s.equals(m.getKey())) {
                        bw.write("IS\t" + m.getValue() + "\t");       //if match found in
imperative stmt
                        type = "is";
                        isOpcode = 1;
                    }
                }


                for (Map.Entry m : ad.entrySet()) {
                    if (s.equals(m.getKey())) {
                        bw.write("AD\t" + m.getValue() + "\t");       //if match found in
Assembler Directive
                        type = "ad";
                        isOpcode = 1;
                    }
                }
                for (Map.Entry m : dl.entrySet()) {
                    if (s.equals(m.getKey())) {
                        bw.write("DL\t" + m.getValue() + "\t");       //if match found in
declarative stmt
                        type = "dl";
                        isOpcode = 1;
                    }
                }
```

```
if (s.equals("LTORG")) {
    pooltab.add(pooltabptr);
    for (Map.Entry m : littab.entrySet()) {
        if (m.getValue() == "") {           //if addr is not assigned to the literal
            m.setValue(locptr);
            locptr++;
            pooltabptr++;
            mind_the_LC = 1;
            isOpcode = 1;
        }
    }
}


if (s.equals("END")) {
    pooltab.add(pooltabptr);
    for (Map.Entry m : littab.entrySet()) {
        if (m.getValue() == "") {
            m.setValue(locptr);
            locptr++;
            mind_the_LC = 1;
        }
    }
}


if(s.equals("EQU")){
    symtab.put("equ", String.valueOf(locptr));
}


if (sCurrentLine.split(" |\\,").length > 2) {     //if there are 3 words
    s = sCurrentLine.split(" |\\,")[2];            //consider the 3rd word

    //this is our first operand.
    //it must be either a Register/Declaration/Symbol
    if (s.equals("AREG")) {
        bw.write("1\t");
        isOpcode = 1;
    } else if (s.equals("BREG")) {
```

```java
                    bw.write("2\t");
                    isOpcode = 1;
                } else if (s.equals("CREG")) {
                    bw.write("3\t");
                    isOpcode = 1;
                } else if (s.equals("DREG")) {
                    bw.write("4\t");
                    isOpcode = 1;
                } else if (type == "dl") {
                    bw.write("C\t" + s + "\t");
                } else {
                    symtab.put(s, "");          //forward referenced symbol
                }
            }

            if (sCurrentLine.split(" |\\,").length > 3) {     //if there are 4 words

                s = sCurrentLine.split(" |\\,")[3];          //consider 4th word.
                //this is our 2nd operand
                //it is either a literal, or a symbol
                if (s.contains("=")) {
                    littab.put(s, "");
                    bw.write("L\t" + litptr + "\t");
                    isOpcode = 1;
                    litptr++;
                } else {
                    symtab.put(s, "");
                    //
                    bw.write("S\t" + symptr + "\t");
                    symptr++;

                }
            }

            bw.write("\n");       //done with a line.

            if (mind_the_LC == 0)
                locptr++;
        }

        System.out.println("Imperative Statements-------");
        for (Object objectName : is.keySet()) {
            System.out.println(objectName+"\t"+is.get(objectName));
        }

        System.out.println("Assembler Directive-------");
        for (Object objectName : ad.keySet()) {
            System.out.println(objectName+"\t"+is.get(objectName));
        }

        System.out.println("Declarative Statements-------");
        for (Object objectName : dl.keySet()) {
            System.out.println(objectName + "\t" + dl.get(objectName));
        }

        System.out.print("\n---------Symbol Table------ \n");
        String f1 = "E:\\pass1_assembler\\OUTPUT\\SYMTAB.txt";
        FileWriter fw1 = new FileWriter(f1);
        BufferedWriter bw1 = new BufferedWriter(fw1);
        for (Map.Entry m : symtab.entrySet()) {
            bw1.write(m.getKey() + "\t" + m.getValue()+"\n");
            System.out.println(m.getKey() + " " + m.getValue());
        }

        System.out.print("\n---------Literal Table------ \n");
        String f2 = "E:\\pass1_assembler\\OUTPUT\\LITTAB.txt";
        FileWriter fw2 = new FileWriter(f2);
        BufferedWriter bw2 = new BufferedWriter(fw2);
        for (Map.Entry m : littab.entrySet()) {
            bw2.write(m.getKey() + "\t" + m.getValue()+"\n");
            System.out.println(m.getKey() + " " + m.getValue());
        }

        System.out.print("\n---------Pool Table------\n");
        String f3 = "E:\\pass1_assembler\\OUTPUT\\POOLTAB.txt";
        FileWriter fw3 = new FileWriter(f3);
        BufferedWriter bw3 = new BufferedWriter(fw3);
        for (Integer item : pooltab) {
            bw3.write(item+"\n");
            System.out.println(item);
        }

        bw.close();
        bw1.close();
```

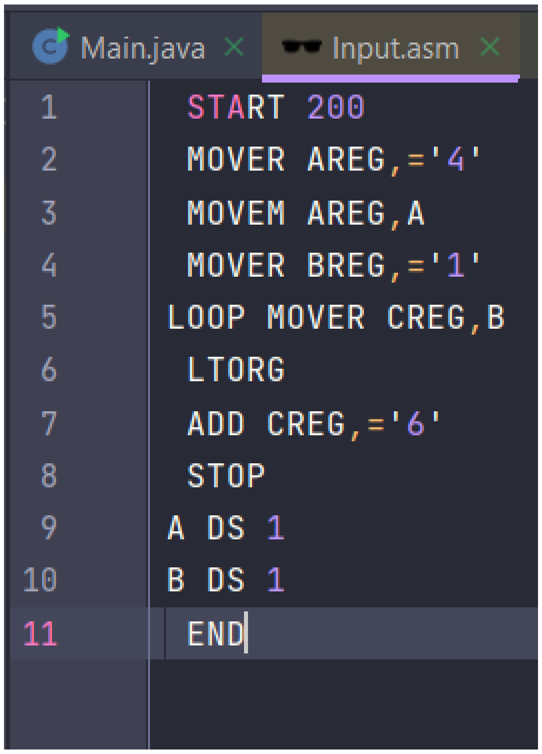
```
            bw2.close();
            bw3.close();

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

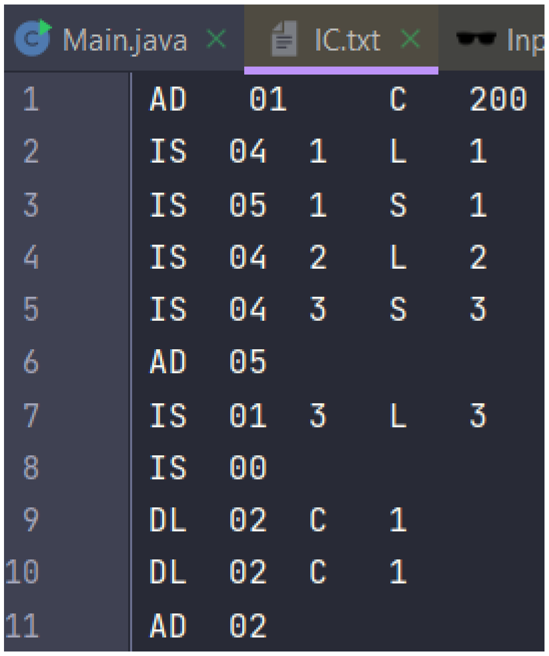
Input file :-



Main.java   Input.asm

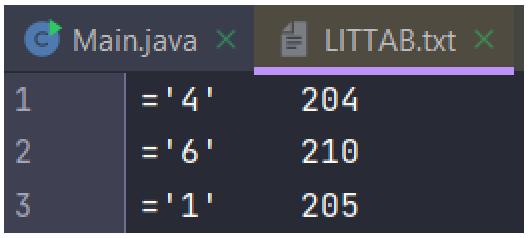
```
  START 200
  MOVER AREG,='4'
  MOVEM AREG,A
  MOVER BREG,='1'
LOOP MOVER CREG,B
  LTORG
  ADD CREG,='6'
  STOP
A DS 1
B DS 1
  END
```

Output File:-



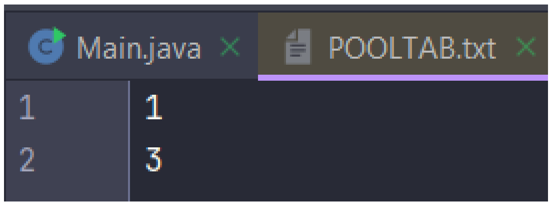
```
AD   01     C   200
IS  04  1   L   1
IS  05  1   S   1
IS  04  2   L   2
IS  04  3   S   3
AD  05
IS  01  3   L   3
IS  00
DL  02  C   1
DL  02  C   1
AD  02
```

1) Intermediate code IC.txt



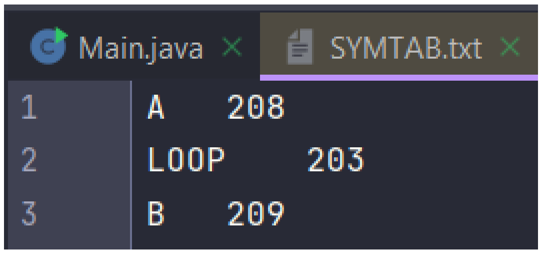
```
='4'    204
='6'    210
='1'    205
```

2) Literal Table LITTABLE.txt



```
1
3
```

3) Pool Table POOLTAB.txt



```
A   208
LOOP    203
B   209
```

4) Symbol Table SYMTAB.txt