CODE:

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

class As1
{
	public static boolean isNum(String str)
	{
		try
		{
			Integer.parseInt(str);
		}
		catch(NumberFormatException e)
		{
			return false;
		}
		return true;
	}

	public static void main(String[] args) throws IOException
	{
		Hashtable<String,String> IS = new Hashtable<String, String>();
		Hashtable<String,String> AD = new Hashtable<String, String>();
		Hashtable<String,String> DL = new Hashtable<String, String>();
		Hashtable<String,String> Reg = new Hashtable<String, String>();
		Hashtable<String,String> BC_Cond = new Hashtable<String, String>();

		//Hashtables
		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");

		DL.put("DC", "01");
		DL.put("DS", "02");

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

		Reg.put("AREG", "1");
		Reg.put("BREG", "2");
		Reg.put("CREG", "3");
		Reg.put("DREG", "4");

		BC_Cond.put("LT", "1");
		BC_Cond.put("LE", "2");
		BC_Cond.put("EQ", "3");
		BC_Cond.put("GT", "4");
		BC_Cond.put("GE", "5");
```

```
            BC_Cond.put("ANY", "6");
            BC_Cond.put("NE", "6");

            ArrayList<String[]> sym_tab = new ArrayList<String[]>();
            ArrayList<String[]> lit_tab = new ArrayList<String[]>();
            ArrayList<Integer> pool_tab = new ArrayList<Integer>();
            int sym_ptr = 1, temp_ptr = 1, lit_ptr = 1;   //Initializing table pointers
            int pool_ptr = 0;
            pool_tab.add(0);
            int linenum = 0;

            //These flags are used to check which instruction is being executed
            boolean[] flags = {false,false,false,false,false,false,false,false,     false,   false};
            //Corresponds to= |start|ltorg|label|  ds |  dc | equ |  bc |sym tab|new line| END|

            File input = new File("input.asm");     //Input file containing Source Code
            input.createNewFile();
            File output = new File("intermediate.asm");   //Output file to contain Intermediate Code
            output.createNewFile();
            File tables = new File("tables.asm");  //File contains symbol and literal tables for use
in Pass 2
            tables.createNewFile();

            //Tokenizer
            Scanner fileReader = new Scanner(input);
            String i_str = "", temp_str[];
            String[] tokens;
            FileWriter fw = new FileWriter("intermediate.asm");
            BufferedWriter bw = new BufferedWriter(fw);
            while(fileReader.hasNextLine())
            {
                  i_str = fileReader.nextLine();
                  flags[8]=true;
                  tokens = i_str.split("[ \\n\\t,]");     //Splits the line into tokens
                  //Assembler Pass I
                  for(String str : tokens)
                  {
                        flags[9]=false;
                        //LABEL
                        if(!str.equals("") && !str.equals("START") && !str.equals("END") &&
str.equals(tokens[0]))    //Checks if token is label
                        {
                              for(String[] str_arr : sym_tab)
                              {
                                    if(str_arr[1].equals(str))//Checks if symbol already in table
                                    {
                                          temp_ptr = Integer.parseInt(str_arr[0])-1;
                                          flags[2] = true;
                                          if(str_arr[2]=="")  //Addresses unaddressed symbol
                                          {
                                                sym_tab.set(temp_ptr, new String[]
{str_arr[0],str_arr[1],""+(linenum-1),"1"});
                                          }
                                    }
                              }
                              if(flags[2] == false)     //Adds new symbol to table
                              {
                                    temp_str = new String[] {""+sym_ptr,str,""+(linenum),"1"};
                                    temp_ptr = sym_ptr++;
                                    sym_tab.add(temp_str);
                                    flags[2] = true;
```

```
						}
					}
					else
					{
						str = str.trim();
					}
					if(str=="")  //Skips blank tokens
						continue;
					//OPCODE
					if(flags[8])
					{
						flags[8]=false;
						if(flags[2] && AD.containsKey(tokens[1]))
						{
							bw.write("\t");
							flags[2]=false;
							continue;
						}
						if(!AD.containsKey(str))  //Checks for Non-Assembler Directives
						{
							bw.write(linenum+")");
						}
						bw.write("\t");
						linenum++;
					}
					if(AD.containsKey(str))   //Checks for Assembler Directives
					{
						bw.write("\t");
						if(str.equals("START")||str.equals("ORIGIN"))
						{
							flags[0]=true;
							bw.write("(AD," + AD.get(str) + ") ");
						}
						else if(str.equals("LTORG")||str.equals("END"))
						{
							if(str.equals("END"))
								flags[9]=true;
							bw.write("(AD,"+AD.get(str)+")");
							for(int i = pool_tab.get(pool_ptr); i < lit_tab.size(); i++)
							{
								bw.write("\n"+linenum+++")\t");
								bw.write("(DL,01) (C,");
								lit_tab.get(i)[2] = Integer.toString(linenum-1);
	bw.write(lit_tab.get(i)[1].substring(lit_tab.get(i)[1].indexOf('\'')+1,lit_tab.get(i)[1].length(
)-1)+") ");
								flags[1]=true;
							}
							flags[1]=false;
							if(pool_tab.get(pool_ptr) != lit_ptr-1)
							{
								pool_tab.add(lit_ptr-1);
								pool_ptr++;
							}
						}
						else if(str.equals("EQU"))
						{
							flags[5]=true;
							bw.write("(AD," + AD.get(str) + ") ");
						}
						else
```

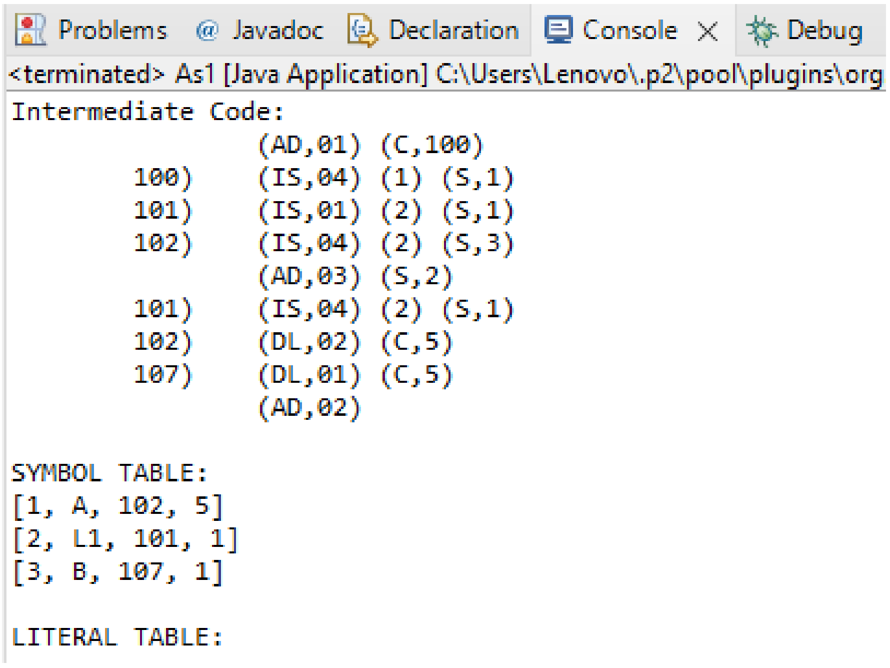
```
                                        bw.write("(AD," + AD.get(str) + ") ");
                        }
                        else if(IS.containsKey(str))     //Checks for Imperative Statements
                        {
                                bw.write("(IS," + IS.get(str) + ") ");
                                if(str.equals("BC"))
                                        flags[6] = true;
                        }
                        else if(DL.containsKey(str))     //Checks for Declaration Statements
                        {
                                bw.write("(DL," + DL.get(str) + ") ");
                                sym_tab.get(temp_ptr)[2]=Integer.toString(linenum-1);
                                if(str.equals("DS"))
                                {
                                        flags[3]=true;
                                }
                                else if(str.equals("DC"))
                                {
                                        flags[4]=true;
                                }
                        }
                        else if(Reg.containsKey(str))    //Checks for register name
                                bw.write("(" + Reg.get(str) + ") ");
                        else if(Pattern.matches("[=]?[']\\d*[']",str))      //Checks for literal
                        {
                                lit_tab.add(new String[]
{""+lit_ptr,"='"+str.substring(str.indexOf('\'')+1,str.lastIndexOf('\''))+"'",""});
                                bw.write("(L,"+(lit_ptr++)+") ");
                        }
                        else if(Pattern.matches("[']*[0-9]+[']*",str))      //Checks for number
                        {
                                bw.write("(C,"+str+") ");
                                if(flags[0] == true)
                                {
                                        linenum = Integer.parseInt(str);
                                        flags[0] = false;
                                }
                                if(flags[3]==true)
                                {
                                        sym_tab.get(temp_ptr)[3]=str;
                                        flags[3]=false;
                                        linenum = linenum + (Integer.parseInt(str)-1);
                                }
                                if(flags[4]==true)
                                {
                                        sym_tab.get(temp_ptr)[3]="1";
                                        flags[4]=false;
                                }
                                if(flags[5]==true)
                                {
                                        sym_tab.get(temp_ptr)[2]=str;
                                        sym_tab.get(temp_ptr)[3]="1";
                                        flags[5] = false;
                                }
                        }
                        else if(flags[2]==false)  //For handling miscelaneous operands
                        {
                                flags[7]=false;
                                if(flags[6]==true)  //Writes OpCode of BC Condition
                                {
                                        bw.write("("+BC_Cond.get(str)+") ");
                        flags[6]=false;
                    }
                    else    //For handling symbols in operand place
                    {
                        for(String[] str_arr : sym_tab)
                        {
                            if(str_arr[1].equals(str))
                            {
                                if(flags[5]==true)  //used when A EQU B
                                {
                                    sym_tab.get(--temp_ptr)[2]=str_arr[2];
                                    sym_tab.get(temp_ptr)[3]=str_arr[3];
                                    bw.write("(S,"+str_arr[0]+") ");
                                }
                                temp_ptr=Integer.parseInt(str_arr[0]);
                                flags[7]=true;
                                if(flags[5]==false)
                                    bw.write("(S,"+temp_ptr+") ");
                                if(flags[0]) //used when ORIGIN A
                                {
                                    linenum = Integer.parseInt(str_arr[2]);
                                    flags[0] = false;
                                }
                            }
                        }
                        if(flags[7]==false) //Used to handle non-label symbols
                        {
                            if(flags[2]==true)
                                sym_tab.add(new String[]
{""+sym_ptr,str,""+(linenum-1),"1"});
                            else
                            {
                                sym_tab.add(new String[] {""+sym_ptr,str,"",""});
                                bw.write("(S,"+sym_ptr+") ");
                            }
                            temp_ptr=sym_ptr;
                            sym_ptr++;
                        }
                    }
                }
                flags[5] = false;
                flags[2] = false;
            }
            if(flags[0])
            {
                bw.write("(C,0) ");
                linenum = 0;
                flags[0]=false;
            }
            if(!flags[9])
                bw.newLine();
        }
        bw.close();
        fileReader.close();

        fileReader = new Scanner(output);
        System.out.println("Intermediate Code:");
        while(fileReader.hasNextLine())
        {
            i_str = fileReader.nextLine();
            if(i_str.charAt(0)!='\t')
```

```
                    System.out.print("\t");
                System.out.println(i_str);
            }
            fileReader.close();

            System.out.println("\nSYMBOL TABLE: ");
            for(String[] arr : sym_tab)
                System.out.println(Arrays.toString(arr));
            System.out.println("\nLITERAL TABLE: ");
            for(String[] arr : lit_tab)
                System.out.println(Arrays.toString(arr));

            //Writing tables to a file
            fw = new FileWriter("tables.asm");
            bw = new BufferedWriter(fw);
            bw.write("[SYMBOL_TABLE]\n");
            for(String[] arr : sym_tab)
            {
                for(String str : arr)
                    bw.write(str+" ");
                bw.write("\n");
            }
            bw.write("[LITERAL_TABLE]\n");
            for(String[] arr : lit_tab)
            {
                for(String str : arr)
                    bw.write(str+" ");
                bw.write("\n");
            }
            bw.close();
            fw.close();
        }
    }
```

INPUT:

```
       START 100
       MOVER AREG, A
L1     ADD BREG, A
       MOVER BREG, B
       ORIGIN L1
       MOVER BREG,A
A      DS 5
B      DC 5
END
```

OUTPUT:



```
<terminated> As1 [Java Application] C:\Users\Lenovo\.p2\pool\plugins\org
Intermediate Code:
                (AD,01) (C,100)
        100)    (IS,04) (1) (S,1)
        101)    (IS,01) (2) (S,1)
        102)    (IS,04) (2) (S,3)
                (AD,03) (S,2)
        101)    (IS,04) (2) (S,1)
        102)    (DL,02) (C,5)
        107)    (DL,01) (C,5)
                (AD,02)

SYMBOL TABLE:
[1, A, 102, 5]
[2, L1, 101, 1]
[3, B, 107, 1]

LITERAL TABLE:
```