**Experiment No. 04**

**Aim-** Design an implementation of pass I of two pass assembler

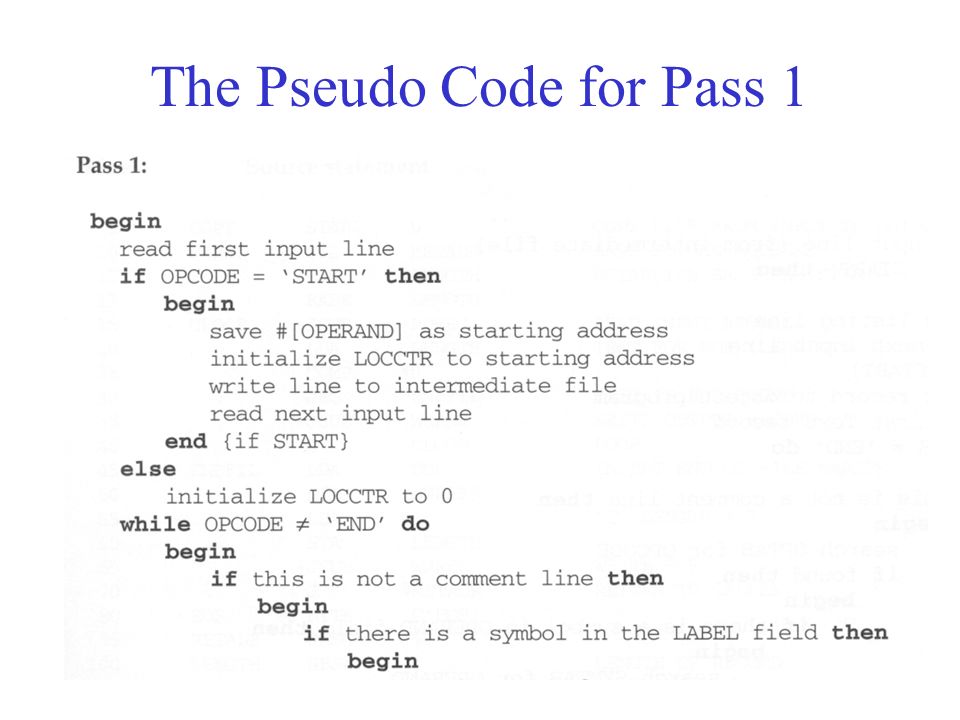
**Requirement-** Java

**Theory -** The one-pass assembler cannot resolve forward references of data symbols. It requires all data symbols to be defined prior to being used. A two-pass assembler solves this dilemma by devoting one pass to exclusively resolve all (data/label) forward references and then generate object code with no hassles in the next pass. If a data symbol depends on another and this another depends on yet another, the assembler resolved this recursively.

It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

Pass-1:

* Define symbols and literals and remember them in symbol table and literal table respectively.
* Keep track of location counter
* Process pseudo-operations
* Assign addresses to all statements in the program.
* Save the values (addresses) assigned to all labels (including label and variable names) for use in Pass 2 (deal with forward references).
* Perform some processing of assembler directives (e.g., BYTE, RESW, these can affect address assignment)

**\**

**Code-**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class pass1

{

    public static void main(String []args)

    {

        BufferedReader reader;

        int lc=0,sti=0,di=0,i,j,li=0;

        int[][] symtab = new int[100][3];

        int[][] littab = new int[100][3];

        String[] sym = new String[100];

        String[] data = new String[100];

        try

        {

            reader = new BufferedReader(new FileReader("prg.txt"));

            String line = reader.readLine();

            String[] words = line.split("\\s+");

            //System.out.println(sym[0]+" "+symtab[0][0]);

            while (!words[1].equals("END"))

            {

                if(!words[0].equals(""))

                {

                    if(words[1].equals("START"))

                    {

                        sym[sti] = words[0];

                        symtab[sti][0] = 0;

                        symtab[sti][1] = 1;

                        symtab[sti][2] = 0;

                        sti++;

                        //System.out.println("in1");

                    }

                    else if(words[1].equals("EQU"))

                    {

                        sym[sti] = words[0];

                        if(words[2].equals("\*")) symtab[sti][0] = lc;

                        else symtab[sti][0] = Integer.parseInt(words[2]);

                        symtab[sti][1] = 1;

                        symtab[sti][2] = 1;

                        sti++;

                        //System.out.println("in2");

                    }

                    else if(words[0].equals("SAVE"))

                    {

                        sym[sti] = words[0];

                        symtab[sti][0] = lc;

                        symtab[sti][1] = 4;

                        symtab[sti][2] = 0;

                        sti++;

                        String[] lcw = words[2].split("F");

                        int ds = Integer.parseInt(lcw[0]);

                        lc+= (ds\*4);

                        //System.out.println("in3");

                    }

                    else

                    {

                        sym[sti] = words[0];

                        symtab[sti][0] = lc;

                        symtab[sti][1] = 4;

                        symtab[sti][2] = 0;

                        sti++;

                        if(words[0].equals("LOOP")) lc+=4;

                        //System.out.println("in4");

                    }

                }

                else if(words[1].equals("USING"))

                {

                    line = reader.readLine();

                    words = line.split("\\s+");

                    //System.out.println("in11");

                    //System.out.println(words[0]);

                    continue;

                }

                else if(words[1].equals("LTORG"))

                {

                    //System.out.println(lc+" "+words[1]);

                    while(lc%8!=0) lc++;

                    for(i=0;i<di;i++)

                    {

                        littab[li][0] = lc;

                        littab[li][1] = 4;

                        littab[li][2] = 0;

                        lc+=4;

                        li++;

                    }

                    //System.out.println("in12");

                }

                else

                {

                    String[] opr = words[2].split(",");

                    //System.out.println(lc+" "+words[1]);

                    //System.out.println(opr[0]+" "+opr[1]);

                    for(i=0;i<opr.length;i++)

                    {

                        if(opr[i].charAt(0) == '=')

                        {

                            data[di] = opr[i];

                            di++;

                        }

                    }

                    if(words[1].charAt(words[1].length()-1) == 'R') lc+=2;

                    else lc+=4;

                    //System.out.println("in13");

                }

                //System.out.println(words[1]);

                line = reader.readLine();

                words = line.split("\\s+");

                //System.out.println(sym[sti-1]+" "+symtab[sti-1][0]+" "+symtab[sti-1][1]+" "+symtab[sti-1][2]);

                //System.out.println(data[di-1]+" "+littab[di-1][0]+" "+littab[di-1][1]+" "+littab[di-1][2]);

                //System.out.println(words[0]);

            }

            reader.close();

            //System.out.println("---------------------------------------------------------");

            //System.out.println("Symbol Table");

            //System.out.println("Symbol    Value   Length  Relocation(0-R;1-A)");

            //System.out.println("---------------------------------------------------------");

            try(OutputStream fw = new FileOutputStream("symboltable.txt"))

            {

                for(i=0;i<sti;i++)

                {

                    //BufferedWriter bw = new BufferedWriter(fw);

                    String content = sym[i]+" "+symtab[i][0]+" "+symtab[i][1]+" "+symtab[i][2]+System.getProperty("line.separator");

                    //System.out.println(content);

                    fw.write(content.getBytes(),0,content.length());

                    //System.out.println(sym[i]+" "+symtab[i][0]+" "+symtab[i][1]+" "+symtab[i][2]);

                }

            }

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

            System.out.println("Check file symboltable.txt");

            //System.out.println("---------------------------------------------------------");

            //System.out.println("Literal Table");

            //System.out.println("Literal   Value   Length  Relocation(0-R;1-A)");

            //System.out.println("---------------------------------------------------------");

            try(OutputStream fw = new FileOutputStream("literaltable.txt"))

            {

                for(i=0;i<di;i++)

                {

                    //BufferedWriter bw = new BufferedWriter(fw);

                    String content = data[i]+" "+littab[i][0]+" "+littab[i][1]+" "+littab[i][2]+System.getProperty("line.separator");

                    //System.out.println(content);

                    fw.write(content.getBytes(),0,content.length());

                    //System.out.println(sym[i]+" "+symtab[i][0]+" "+symtab[i][1]+" "+symtab[i][2]);

                }

            }

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

            System.out.println("Check file literaltable.txt");

        }

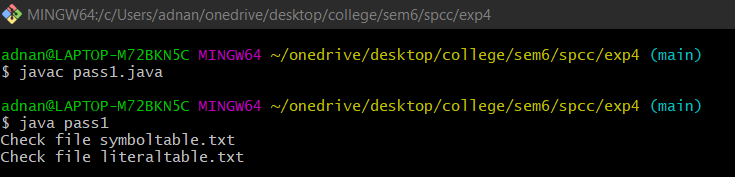
        catch (IOException e) { e.printStackTrace(); }

    }

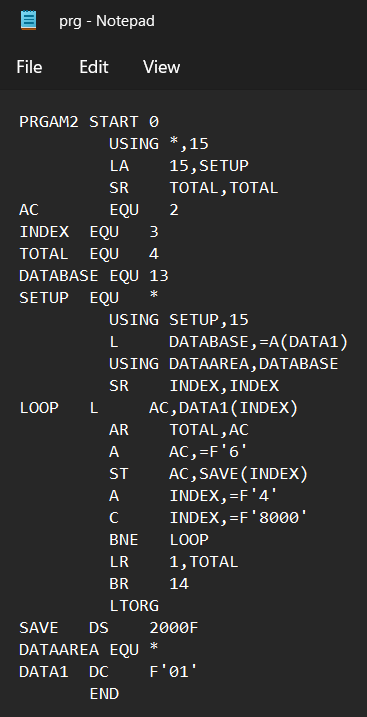
}

**Output-**

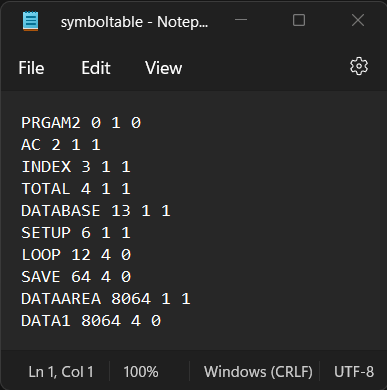
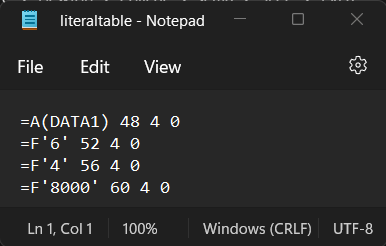
**Execution:**

****

**Input File:**

****

**Symbol Table and Literal Table:**

**Conclusion:** We have successfully implemented Pass 1 of two pass Assembler