NAME: TEJAS NALAWADE PRACTICAL NO: 2 ROLL NO: TCOD01

**Title of Assignment**: Design suitable Data structures and implement Pass-II of a two-pass assembler for pseudo-machine.

**Problem Statement**: Implement pass-II of TWO Pass assembler with hypothetical Instruction set using Java language. Instruction set should include all types of assembly language statements such as Imperative, Declarative and Assembler Directive. While designing stress should be given on

a) How efficiently Mnemonic opcode table could be implemented so as to enable faster retrieval on op code. b) Implementation of symbol table, pool tables for faster retrieval.

**CODE:**

import java.io.\*;

import java.util.Scanner;

public class Pass2 {

    static Obj[] symb\_table = new Obj[10];

    static Obj[] literal\_table = new Obj[10];

    static int symb\_found = 0;

    public static void main(String[] args) throws IOException {

        Scanner sc = new Scanner(System.in);

        System.out.println("ENTER TOTAL NUMBER OF SYMBOLS: ");

        int total\_symb = sc.nextInt();

        for (int i = 0; i < total\_symb; i++) {

            symb\_table[i] = new Obj("", 0);

            System.out.println("ENTER SYMBOL NAME: ");

            symb\_table[i].name = sc.next();

            System.out.println("ENTER SYMBOL ADDRESS: ");

            symb\_table[i].addr = sc.nextInt();

        }

        System.out.println("ENTER TOTAL NUMBER OF LITRALS: ");

        int total\_ltr = sc.nextInt();

        for (int i = 0; i < total\_ltr; i++) {

            literal\_table[i] = new Obj("", 0);

            System.out.println("ENTER LITERAL NAME: ");

            literal\_table[i].name = sc.next();

            System.out.println("ENTER LITERAL ADDRESS: ");

            literal\_table[i].addr = sc.nextInt();

        }

        System.out.println("\n\*\*\*\*SYMBOL TABLE\*\*\*\*");

        System.out.println("\nSYMBOL\tADDRESS");

        for (int i = 0; i < total\_symb; i++) {

            System.out.println(symb\_table[i].name + "\t" + symb\_table[i].addr);

        }

        System.out.println("\n\*\*\*\*\*\*LITERAL TABLE\*\*\*\*\*\*");

        System.out.println("\nIndex\tLITERAL\tADDRESS");

        for (int i = 0; i < total\_ltr; i++) {

            System.out.println((i + 1) + "\t" + literal\_table[i].name + "\t" + literal\_table[i].addr);

        }

        BufferedReader br2 = new BufferedReader(new FileReader("Output.txt"));

        String line;

        boolean symbol\_error = false, undef\_mnemonic = false;

        System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT FILE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

        lab:

        while ((line = br2.readLine()) != null) {

            String[] token\_list = line.split("\\s+",5);

            symbol\_error = false;

            undef\_mnemonic = false;

            labl:

            for (String token : token\_list) {

                if (token.isEmpty()) {

                    continue;

                }

                if (token.matches("[0-9]+")) {

                    System.out.print("\n" + token);

                } else if (token.startsWith("(") && token.endsWith(")")) {

                    String content = token.substring(1, token.length() - 1);

                    String[] parts = content.split(",");

                    if (parts.length == 2) {

                        String letters = parts[0].trim();

                        int num = Integer.parseInt(parts[1].trim());

switch (letters.toUpperCase()) {

    case "S":

        if (num > 0 && num <= total\_symb && symb\_table[num - 1].addr != 0) {

            System.out.print("\t" + symb\_table[num - 1].addr);

        } else {

            System.out.print("\t---");

            symbol\_error = true;

        }

        break;

    case "L":

        if (num > 0 && num <= total\_ltr) {

            System.out.print("\t" + literal\_table[num - 1].addr);

        } else {

             System.out.print("\t---");

             symbol\_error = true;

        }

        break;

    case "AD":

        System.out.print("\n");

        continue labl;

    case "DL":

        switch (num) {

            case 1:

                System.out.print("\n");

                continue labl;

            case 2:

                System.out.print("\t 00 \t 00");

                break;

        }

        break;

    case "C":

        System.out.print(String.format("\t%03d", num));

        break;

    default:

        System.out.print(String.format("\t%03d", num));

        break;

            }

            }

        }

    }

}

        System.out.println();

        if (symbol\_error) {

            System.out.print("\n\n\*\*\*\*\*\*\*\*\*\*\*\*SYMBOL IS NOT DEFINED\*\*\*\*\*");

        }

        if (undef\_mnemonic) {

            System.out.print("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INVALID MNEMONIC\*\*\*\*\*\*");

        }

        int[] flag = new int[total\_symb];

        for (int i = 0; i < total\_symb; i++) {

            symb\_found = 0;

            for (int j = 0; j < total\_symb; j++) {

                if (symb\_table[i].name.equalsIgnoreCase(symb\_table[j].name) && flag[j] == 0) {

                    symb\_found++;

                    if(symb\_found > 1) flag[j] = 1;

                }

            }

            if (symb\_found > 1) {

                System.out.print("\n\n\*\*\*\*\*\*\*" + symb\_table[i].name + "\" IS DUPLICATE SYMBOL");

            }

        }

        br2.close();

        sc.close();

    }

}

class Obj {

    String name;

    int addr;

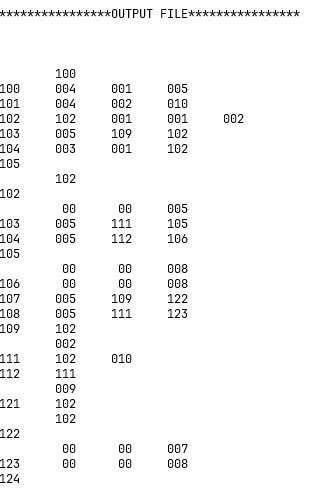
    Obj(String nm, int address) {

        this.name = nm;

        this.addr = address;

    }

}

**OUTPUT:**

