NAME: TEJAS NALAWADE PRACTICAL NO: 1 ROLL NO: TCOD01

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

**Problem Statement**: Implement one pass-I 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 could be implemented so as to enable faster retrieval on op-code. b) Implementation of symbol table for faster retrieval.

**CODE:**

import java.io.\*;

import java.util.\*;

class pass1 {

    public static void main(String args[]) throws NullPointerException, FileNotFoundException {

        String REG[] = {"ax", "bx", "cx", "dx"};

        String IS[] = {"stop", "add", "sub", "mult", "mover", "movem", "comp", "bc", "div", "read"};

        String DL[] = {"ds", "dc"};

        int temp1 = 0;

        int f = 0;

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

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

        Obj[] optab = new Obj[60];

        Pooltable[] pooltab = new Pooltable[5];

        String line;

        try {

            BufferedReader br = new BufferedReader(new FileReader("sample.txt"));

            BufferedWriter bw = new BufferedWriter(new FileWriter("output.txt"));

            Boolean start = false;

            Boolean end = false, fill\_addr = false, ltorg = false;

            int total\_symb = 0, total\_ltr = 0, optab\_cnt = 0, pooltab\_cnt = 0, loc = 0, temp, pos;

            while ((line = br.readLine()) != null && !end) {

                String tokens[] = line.split(" ", 4);

                if (loc != 0 && !ltorg) {

                    if (f == 1) {

                        ltorg = false;

                        loc = loc + temp1 - 1;

                        bw.write("\n" + String.valueOf(loc));

                        f = 0;

                        loc++;

                    } else {

                        bw.write("\n" + String.valueOf(loc));

                        ltorg = false;

                        loc++;

                    }

                }

                ltorg = fill\_addr = false;

                for (int k = 0; k < tokens.length; k++) {

                    pos = -1;

                    if (start == true) {

                        loc = Integer.parseInt(tokens[k]);

                        start = false;

                    }

                    switch (tokens[k]) {

                        case "start":

                            start = true;

                            pos = 1;

                            bw.write("\t(AD,"+ pos +")");

                            break;

                        case "end":

                            end = true;

                            pos = 2;

                            bw.write("\t(AD," + pos +")\n");

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

                                if (literal\_table[temp].addr == 0) {

                                    literal\_table[temp].addr = loc - 1;

                                    bw.write("\t(DL,2) \t (C," + literal\_table[temp].name +")"+"\n" + loc++);

                                }

                            }

                            break;

                        case "origin":

                            pos = 3;

                            bw.write("\t (AD," + pos +")");

                            pos = search(tokens[++k], symb\_table, total\_symb);

                            k++;

                            bw.write("\t(C," +(symb\_table[pos].addr) +")");

                            loc = symb\_table[pos].addr;

                            break;

                        case "ltorg":

                            ltorg = true;

                            pos = 5;

                            bw.write("\t(AD," + pos +")\n");

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

                                if (literal\_table[temp].addr == 0) {

                                    literal\_table[temp].addr = loc - 1;

                                    bw.write("\t(DL,2) \t (C," + literal\_table[temp].name +")" +"\n" +loc++);

                                }

                            }

                            if (pooltab\_cnt == 0) {

                                pooltab[pooltab\_cnt++] = new Pooltable(0, temp);

                            } else {

                                pooltab[pooltab\_cnt] = new Pooltable(pooltab[pooltab\_cnt - 1].first + pooltab[pooltab\_cnt - 1].total\_literals, total\_ltr - pooltab[pooltab\_cnt - 1].first - 1);

                                pooltab\_cnt++;

                            }

                            break;

                        case "equ":

                            pos = 4;

                            bw.write("\t(AD," + pos +")");

                            String prev\_token = tokens[k - 1];

                            int pos1 = search(prev\_token, symb\_table, total\_symb);

                            pos = search(tokens[++k], symb\_table, total\_symb);

                            symb\_table[pos1].addr = symb\_table[pos].addr;

                            bw.write("\t(S," + (pos + 1) + ")");

                            break;

                    }

                    if (pos == -1) {

                        pos = search(tokens[k], IS);

                        if (pos != -1) {

                            bw.write("\t(IS," + (pos) +")");

                            optab[optab\_cnt++] = new Obj(tokens[k], pos);

                        } else {

                            pos = search(tokens[k], DL); // DC/DS

                            if (pos != -1)

                            {

                                if(pos == 0)

                                   { f = 1;}

                                    bw.write("\t(DL," + (pos + 1) + ")");

                                    optab[optab\_cnt++] = new Obj(tokens[k], pos);

                                    fill\_addr = true;

                            } else if (tokens[k].matches("[a-zA-Z]+:")) { //label

                                pos = search(tokens[k], symb\_table, total\_symb);

                                if (pos == -1) {

                                    symb\_table[total\_symb++] = new Obj(tokens[k].substring(0, tokens[k].length() - 1), loc - 1);

                                    bw.write("\t(S," + total\_symb + ")");

                                    pos = total\_symb;

                                }

                            }

                        }

                    }

                    if (pos == -1) {

                        pos = search(tokens[k], REG);

                        if (pos != -1) {

                            bw.write("\t(RG," + (pos + 1) +")"); //register

                        } else {

                            if (tokens[k].matches("='(\\d+)'")) { //literal

                                String s = tokens[k].substring(2, 3);

                                literal\_table[total\_ltr++] = new Obj(s, 0);

                                bw.write("\t(L," + total\_ltr + ")");

                            }

                            else if (tokens[k].matches("\\d+") || tokens[k].matches("\\d+H") || tokens[k].matches("\\d+h") || tokens[k].matches("\\d+D") || tokens[k].matches("\\d+d")) { //constant

                                bw.write("\t(C," + tokens[k] + ")");

                                temp1 = Integer.parseInt(tokens[k]);

                            }

                            else {

                                pos = search(tokens[k], symb\_table, total\_symb);

                                if (fill\_addr && pos != -1 && symb\_table[pos].addr == 0) {

                                    symb\_table[pos].addr = loc - 1;

                                    fill\_addr = false;

                                } else if (pos == -1) {

                                    symb\_table[total\_symb++] = new Obj(tokens[k], 0);

                                    bw.write("\t (S," + total\_symb + ")");

                                } else {

                                    bw.write("\t(S," + pos + ")");

                                }

                            }

                        }

                    }

                }

            }

            pooltab[pooltab\_cnt] = new Pooltable(pooltab[pooltab\_cnt - 1].first + pooltab[pooltab\_cnt - 1].total\_literals, total\_ltr - pooltab[pooltab\_cnt - 1].first - 2);

            pooltab\_cnt++;

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

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

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

                if (literal\_table[i].addr == 0) {

                    literal\_table[i].addr = loc++;

                }

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

            }

            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\*POOL TABLE\*");

            System.out.println("\nPOOL\tTOTAL LITERALS");

            for (int i = 0; i < pooltab\_cnt; i++) {

                System.out.println(pooltab[i].first + "\t" + pooltab[i].total\_literals);

            }

            System.out.println("\n\*OPTABLE\*");

            System.out.println("\nMNEMONIC\tOPCODE");

            for (int i = 0; i < IS.length; i++) {

                System.out.println(IS[i] + "\t\t" + i);

            }

            br.close();

            bw.close();

        } catch (Exception e) {

            System.out.println("error while reading the file");

            e.printStackTrace();

        }

        try {

            BufferedReader br = new BufferedReader(new FileReader("output.txt"));

            System.out.println("\n\*Output1.txt\n");

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

                System.out.println(line);

            }

            br.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    public static int search(String token, String[] list) {

        for (int i = 0; i < list.length; i++) {

            if (token.equalsIgnoreCase(list[i])) {

                return i;

            }

        }

        return -1;

    }

    public static int search(String token, Obj[] list, int cnt) {

        for (int i = 0; i < cnt; i++) {

            if (token.equalsIgnoreCase(list[i].name)) {

                return i;

            }

        }

        return -1;

    }

}

**OUTPUT:**



