package ketan;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.FileWriter;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.StringTokenizer;

class Tuple {

    String mnemonic, m\_class, opcode;

    int length;

    Tuple() {}

    Tuple(String s1, String s2, String s3, String s4) {

        mnemonic = s1;

        m\_class = s2;

        opcode = s3;

        length = Integer.parseInt(s4);

    }

}

class SymTuple {

    String symbol, address;

    int length;

    SymTuple(String s1, String s2, int i1) {

        symbol = s1;

        address = s2;

        length = i1;

    }

}

class LitTuple {

    String literal, address;

    int length;

    LitTuple() {}

    LitTuple(String s1, String s2, int i1) {

        literal = s1;

        address = s2;

        length = i1;

    }

}

public class Assembler\_PassOne {

    static int lc, iSymTabPtr = 0, iLitTabPtr = 0, iPoolTabPtr = 0;

    static int poolTable[] = new int[10];

    static Map<String, Tuple> MOT;

    static Map<String, SymTuple> symtable;

    static ArrayList<LitTuple> littable;

    static Map<String, String> regAddressTable;

    static PrintWriter out\_pass2;

    static PrintWriter out\_pass1;

    static int line\_no;

    static String processLTORG() {

        LitTuple litTuple;

        String intermediateStr = "";

        for (int i = poolTable[iPoolTabPtr - 1]; i < littable.size(); i++) {

            litTuple = littable.get(i);

            litTuple.address = lc + "";

            intermediateStr += lc + " (DL,02)  (C," + litTuple.literal + ") \n";

            lc++;

        }

        poolTable[iPoolTabPtr] = iLitTabPtr;

        iPoolTabPtr++;

        return intermediateStr;

    }

    static void Pass1() throws Exception {

        BufferedReader input = new BufferedReader(new InputStreamReader(new FileInputStream("src/lp\_practical1/input.txt")));

        out\_pass1 = new PrintWriter(new FileWriter("src/lp\_practical1/outputpass1.txt"), true);

        PrintWriter out\_symtable = new PrintWriter(new FileWriter("src/lp\_practical1/symtable.txt"), true);

        PrintWriter out\_littable = new PrintWriter(new FileWriter("src/lp\_practical1/littable.txt"), true);

        out\_pass1.println("Ketan");

        String s;

        lc = 0;

        while ((s = input.readLine()) != null) {

            StringTokenizer st = new StringTokenizer(s, " ", false);

            String s\_arr[] = new String[st.countTokens()];

            for (int i = 0; i < s\_arr.length; i++) {

                s\_arr[i] = st.nextToken();

            }

            if (s\_arr.length == 0) {

                continue;

            }

            int curIndex = 0;

            if (s\_arr.length == 3) {

                String label = s\_arr[0];

                insertIntoSymTab(label, lc + "");

                curIndex = 1;

            }

            String curToken = s\_arr[curIndex];

            Tuple curTuple = MOT.get(curToken);

            String intermediateStr = "";

            // Null check for curTuple

            if (curTuple != null) {

                if (curTuple.m\_class.equalsIgnoreCase("IS")) {

                    intermediateStr += lc + " (" + curTuple.m\_class + "," + curTuple.opcode + " ) ";

                    lc += curTuple.length;

                    intermediateStr += processOperands(s\_arr[curIndex + 1]);

                } else if (curTuple.m\_class.equalsIgnoreCase("AD")) {

                    if (curTuple.mnemonic.equalsIgnoreCase("START")) {

                        intermediateStr += lc + " (" + curTuple.m\_class + "," + curTuple.opcode + ") ";

                        lc = Integer.parseInt(s\_arr[curIndex + 1]);

                        intermediateStr += "(C," + s\_arr[curIndex + 1] + ") ";

                    } else if (curTuple.mnemonic.equalsIgnoreCase("LTORG")) {

                        intermediateStr += processLTORG();

                    } else if (curTuple.mnemonic.equalsIgnoreCase("END")) {

                        intermediateStr += lc + " (" + curTuple.m\_class + "," + curTuple.opcode + ")  \n";

                        intermediateStr += processLTORG();

                    }

                } else if (curTuple.m\_class.equalsIgnoreCase("DL")) {

                    intermediateStr += lc + " (" + curTuple.m\_class + "," + curTuple.opcode + ") ";

                    if (curTuple.mnemonic.equalsIgnoreCase("DS")) {

                        lc += Integer.parseInt(s\_arr[curIndex + 1]);

                    } else if (curTuple.mnemonic.equalsIgnoreCase("DC")) {

                        lc += curTuple.length;

                    }

                    intermediateStr += "(C," + s\_arr[curIndex + 1] + ") ";

                }

            } else {

                // Log and handle null curTuple

                System.out.println("Unknown mnemonic: " + curToken);

                continue; // Skip processing for this token

            }

            System.out.println(intermediateStr);

            out\_pass1.println(intermediateStr);

        }

        out\_pass1.flush();

        out\_pass1.close();

        // Symbol Table Output

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

        SymTuple tuple;

        Iterator<SymTuple> it = symtable.values().iterator();

        String tableEntry;

        while (it.hasNext()) {

            tuple = it.next();

            tableEntry = tuple.symbol + "\t" + tuple.address;

            out\_symtable.println(tableEntry);

            System.out.println(tableEntry);

        }

        out\_symtable.flush();

        out\_symtable.close();

        // Literal Table Output

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

        LitTuple litTuple;

        tableEntry = "";

        for (int i = 0; i < littable.size(); i++) {

            litTuple = littable.get(i);

            tableEntry = litTuple.literal + "\t" + litTuple.address;

            out\_littable.println(tableEntry);

            System.out.println(tableEntry);

        }

        out\_littable.flush();

        out\_littable.close();

    }

    static String processOperands(String operands) {

        StringTokenizer st = new StringTokenizer(operands, ",", false);

        String s\_arr[] = new String[st.countTokens()];

        for (int i = 0; i < s\_arr.length; i++) {

            s\_arr[i] = st.nextToken();

        }

        String intermediateStr = "", curToken;

        for (int i = 0; i < s\_arr.length; i++) {

            curToken = s\_arr[i];

            if (curToken.startsWith("=")) {

                StringTokenizer str = new StringTokenizer(curToken, "'", false);

                String tokens[] = new String[str.countTokens()];

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

                    tokens[j] = str.nextToken();

                }

                String literal = tokens[1];

                insertIntoTab(literal, "");

                intermediateStr += "(L," + (iLitTabPtr - 1) + ")";

            } else if (regAddressTable.containsKey(curToken)) {

                intermediateStr += "(RG," + regAddressTable.get(curToken) + ") ";

            } else {

                insertIntoSymTab(curToken, "");

                intermediateStr += "(S," + (iSymTabPtr - 1) + ")";

            }

        }

        return intermediateStr;

    }

    static void insertIntoSymTab(String symbol, String address) {

        if (symtable.containsKey(symbol)) {

            SymTuple s = symtable.get(symbol);

            s.address = address;

        } else {

            symtable.put(symbol, new SymTuple(symbol, address, 1));

        }

        iSymTabPtr++;

    }

    static void insertIntoTab(String literal, String address) {

        littable.add(iLitTabPtr, new LitTuple(literal, address, 1));

        iLitTabPtr++;

    }

    static void initializeTables() throws Exception {

        symtable = new LinkedHashMap<>();

        littable = new ArrayList<>();

        regAddressTable = new HashMap<>();

        MOT = new HashMap<>();

        String s, mnemonic;

        BufferedReader br;

        br = new BufferedReader(new InputStreamReader(new FileInputStream("src/lp\_practical1/MOT.txt")));

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

            StringTokenizer st = new StringTokenizer(s, " ", false);

            mnemonic = st.nextToken();

            MOT.put(mnemonic, new Tuple(mnemonic, st.nextToken(), st.nextToken(), st.nextToken()));

        }

        br.close();

        regAddressTable.put("AREG", "1");

        regAddressTable.put("BREG", "2");

        regAddressTable.put("CREG", "3");

        regAddressTable.put("DREG", "4");

        poolTable[iPoolTabPtr] = iLitTabPtr;

        iPoolTabPtr++;

    }

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

        System.out.println("Name: Ketan Devraj\nRoll No. 22122\n");

        initializeTables();

        Pass1();

    }

}

**INPUT.txt**

START 100

MOVER AREG,B

ADD BREG,='6'

MOVEM AREG,A

SUB CREG,='1'

LTORG

ADD DREG,='5'

A DS 10

LTORG

SUB AREG,='1'

B DC 1

C DC 1

END

**LIT TABLE**

6 104

1 105

5 117

1 119

**MOT**

START AD 01 0

END AD 02 0

LTORG AD 05 0

ADD IS 01 1

SUB IS 02 1

MULT IS 03 1

MOVER IS 04 1

MOVEM IS 05 1

DC DL 01 0

DS DL 02 1

**SYM TABLE**

B 119

A 107

C 119

**OUTPUT**



