import java.io.\*;

import java.util.\*;

public class Assembler {

private static int LC = 0;

private static int pooltab\_ptr = 1;

private static int littab\_ptr = 1;

private static Map<String, Integer> symbolTable = new HashMap<>();

private static List<String> litTab = new ArrayList<>();

private static List<Integer> poolTab = new ArrayList<>(Collections.singletonList(littab\_ptr));

private static List<String> icList = new ArrayList<>();

private static final Map<String, Integer> optab = new HashMap<>();

public static void main(String[] args) {

initializeOptab();

try {

BufferedReader reader = new BufferedReader(new FileReader("source.asm"));

String line;

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

processLine(line.trim());

}

processEndStatement();

writeTablesAndIC();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void initializeOptab() {

optab.put("ADD", 1);

optab.put("SUB", 2);

optab.put("MOVER", 1);

optab.put("MOVEM", 1);

optab.put("PRINT", 1);

optab.put("STOP", 1);

}

private static void processLine(String line) {

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

if (tokens.length == 0) return;

String instruction = tokens[0].toUpperCase();

switch (instruction) {

case "START":

LC = Integer.parseInt(tokens[1]);

break;

case "DS":

case "DB":

int size = Integer.parseInt(tokens[1]);

LC += size;

icList.add(instruction + " " + size);

break;

case "EQU":

String label = tokens[1];

int address = parseAddress(tokens[2]);

symbolTable.put(label, address);

break;

case "LTORG":

processLiterals();

break;

case "END":

processEndStatement();

break;

case "ORIGIN":

LC = parseAddress(tokens[1]);

break;

case "STOP":

icList.add("STOP");

break;

default:

// Handle IS instructions (MOVER, ADD, etc.)

if (optab.containsKey(instruction)) {

String opcode = instruction;

LC += optab.get(opcode);

icList.add("IS " + opcode + " " + String.join(" ", Arrays.copyOfRange(tokens, 1, tokens.length)));

// Check for literals

for (int i = 1; i < tokens.length; i++) {

String token = tokens[i];

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

String literal = token;

if (!litTab.contains(literal)) {

litTab.add(literal);

littab\_ptr++;

}

} else if (token.contains("+")) {

String operand = token.split("\\+")[0];

if (!symbolTable.containsKey(operand)) {

symbolTable.put(operand, -1);

}

}

}

}

break;

}

}

private static int parseAddress(String address) {

// Handle addresses that might be expressions

try {

return Integer.parseInt(address);

} catch (NumberFormatException e) {

// If it's not a number, handle it as a symbol or expression

// For simplicity, return a placeholder value (-1)

return -1;

}

}

private static void processLiterals() {

for (int i = poolTab.get(pooltab\_ptr - 1); i < litTab.size(); i++) {

String literal = litTab.get(i);

icList.add("LIT " + literal + " " + LC);

LC++;

}

pooltab\_ptr++;

poolTab.add(littab\_ptr);

}

private static void processEndStatement() {

processLiterals();

icList.add("END");

}

private static void writeTablesAndIC() throws IOException {

try (PrintWriter writer = new PrintWriter(new FileWriter("output.txt"))) {

writer.println("Symbol Table:");

for (Map.Entry<String, Integer> entry : symbolTable.entrySet()) {

writer.println(entry.getKey() + " " + entry.getValue());

}

writer.println("\nLiteral Table:");

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

writer.println("L" + i + " " + litTab.get(i));

}

writer.println("\nPool Table:");

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

writer.println("P" + i + " " + poolTab.get(i));

}

writer.println("\nIntermediate Code:");

for (String ic : icList) {

writer.println(ic);

}

}

}

}