Practical No 01

Name : Bhakti Gadilkar
Rollno: 13163
Batch: A3
System Programming & Operating System Lab

Problem Statement : Design suitable Data structures and implement Pass-I and Pass-II of a two- pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.

Java Program :

```
import java.io.*;
import java.util.*;

public class TwoPassAssemblerPass1 {
   static int locationCounter = 0;
   static Map<String, Integer> symbolTable = new LinkedHashMap<>();
   static List<String> intermediateCode = new ArrayList<>();
   static Map<String, String[]> opcodeTable = new HashMap<>();

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

      BufferedReader reader = new BufferedReader(new FileReader("input.asm"));
      BufferedWriter icWriter = new BufferedWriter(new FileWriter("intermediate.txt"));
      BufferedWriter symWriter = new BufferedWriter(new FileWriter("symbol.txt"));

      String line;
      while ((line = reader.readLine()) != null) {
         processLine(line.trim());
      }
      reader.close();

      // Write Intermediate Code
      for (String code : intermediateCode) {
         icWriter.write(code + "\n");
      }

      // Write Symbol Table
      for (Map.Entry<String, Integer> entry : symbolTable.entrySet()) {
         symWriter.write(entry.getKey() + " " + entry.getValue() + "\n");
      }
```

```
        icWriter.close();
        symWriter.close();
        System.out.println("Pass 1 complete. Files: intermediate.txt, symbol.txt");
    }

    static void initOpcodeTable() {
        opcodeTable.put("START", new String[]{"AD", "01"});
        opcodeTable.put("END", new String[]{"AD", "02"});
        opcodeTable.put("LTORG", new String[]{"AD", "03"});
        opcodeTable.put("DS", new String[]{"DL", "01"});
        opcodeTable.put("DC", new String[]{"DL", "02"});
        opcodeTable.put("MOVER", new String[]{"IS", "01"});
        opcodeTable.put("MOVEM", new String[]{"IS", "02"});
        opcodeTable.put("ADD", new String[]{"IS", "03"});
        opcodeTable.put("SUB", new String[]{"IS", "04"});
    }

    static void processLine(String line) {
        if (line.isEmpty()) return;

        String[] tokens = line.split("\\s+");
        int index = 0;
        String label = null;

        // Check if label exists
        if (!opcodeTable.containsKey(tokens[index])) {
            label = tokens[index++];
        }

        String opcode = tokens[index++];
        String[] code = opcodeTable.get(opcode);

        switch (opcode) {
            case "START":
                locationCounter = Integer.parseInt(tokens[index]);
                intermediateCode.add("(AD,01) (C," + locationCounter + ")");
                break;

            case "END":
                intermediateCode.add("(AD,02)");
                break;

            case "DS":
            case "DC":
                int size = Integer.parseInt(tokens[index]);
                if (label != null) {
                    symbolTable.put(label, locationCounter);
                }
```

```
                intermediateCode.add("(DL," + code[1] + ") (C," + size + ")");
                locationCounter += size;
                break;

            default:
                String reg = tokens[index++];
                String operand = tokens[index];
                if (label != null) {
                    symbolTable.put(label, locationCounter);
                }
                if (!symbolTable.containsKey(operand)) {
                    symbolTable.put(operand, null); // forward reference
                }
                intermediateCode.add("(IS," + code[1] + ") " + parseRegister(reg) + " (S," + operand + ")");
                locationCounter++;
                break;
        }
    }

    static String parseRegister(String reg) {
        return switch (reg) {
            case "AREG" -> "1";
            case "BREG" -> "2";
            case "CREG" -> "3";
            case "DREG" -> "4";
            default -> "0";
        };
    }
}
```

Input.asm

```
START 100
MOVER AREG A
ADD BREG B
MOVEM AREG C
A DS 1
B DC 5
C DS 1
END
```

Intermediate Code :

```
(AD,01) (C,100)
(IS,01) 1 (S,A)
(IS,03) 2 (S,B)
(IS,02) 1 (S,C)
```

(DL,01) (C,1)
(DL,02) (C,5)
(DL,01) (C,1)
(AD,02)

Machine Code :

machine code -

01 1 103
03 2 104
02 1 109
-
-
-
- symbol table A 103
B 104
C 109

Pass 2

```
import java.io.*;
import java.util.*;

public class Pass2Assembler {
   static Map<String, Integer> symbolTable = new HashMap<>();

   public static void main(String[] args) throws IOException {
      loadSymbolTable("symbol.txt");
      BufferedReader reader = new BufferedReader(new FileReader("intermediate.txt"));
      BufferedWriter writer = new BufferedWriter(new FileWriter("machinecode.txt"));

      String line;
      while ((line = reader.readLine()) != null) {
         if (line.startsWith("(IS")) {
            String[] parts = line.split("\\s+");
            String opcode = parts[0].replaceAll("[()]", "").split(",")[1];
            String reg = parts[1];
            String symRef = parts[2].replaceAll("[()]", "");
            String symbol = symRef.split(",")[1];
            int addr = symbolTable.getOrDefault(symbol, 0);
            writer.write(opcode + " " + reg + " " + addr + "\n");
         } else {
            writer.write("-\n"); // For AD, DL lines
         }
      }

      reader.close();
```

```
        writer.close();
        System.out.println("Pass 2 complete. Machine code generated.");
    }

    static void loadSymbolTable(String filename) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(filename));
        String line;
        while ((line = br.readLine()) != null) {
            String[] parts = line.split("\\s+");
            symbolTable.put(parts[0], Integer.parseInt(parts[1]));
        }
        br.close();
    }
}
```