Practical No 01

Name : Kiran Dadarao Janjal
Roll No : 13216
Batch : B1
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.*

Pass 1
Input Code
Input.asm

```
START 100
MOVER AREG, =5
ADD BREG, =3
MOVEM AREG, ALPHA
ALPHA DS 1
LTORG
MOVER CREG, ALPHA
END
```

Java Program :

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

public class Pass1 {
    static int LC = 0; // Location Counter
    static Map<String, Integer> SYMTAB = new LinkedHashMap<>();
    static List<String> LITTAB = new ArrayList<>();
    static Map<String, Integer> LITADDR = new LinkedHashMap<>();
    static List<Integer> POOLTAB = new ArrayList<>();
    static Map<String, String[]> OPTAB = new HashMap<>();
    static List<String> intermediate = new ArrayList<>();
    public static void main(String[] args) throws IOException {
        initOPTAB();

        BufferedReader br = new BufferedReader(new FileReader("input.asm"));
        POOLTAB.add(1);

        String line;
        while ((line = br.readLine()) != null) {
            process(line.trim());
        }
        br.close();

        // Write Intermediate Code
        try (BufferedWriter bw = new BufferedWriter(new FileWriter("intermediate.txt"))) {
            for (String ic : intermediate) {
                bw.write(ic + "\n");
            }
        }

        // Write Symbol Table
        try (BufferedWriter bw = new BufferedWriter(new FileWriter("symtab.txt"))) {
            for (Map.Entry<String, Integer> e : SYMTAB.entrySet()) {
                bw.write(e.getKey() + "\t" + e.getValue() + "\n");
            }
        }

        // Write Literal Table
        try (BufferedWriter bw = new BufferedWriter(new FileWriter("littab.txt"))) {
            int index = 1;
            for (String lit : LITTAB) {
                bw.write(index++ + "\t" + lit + "\t" + LITADDR.get(lit) + "\n");
            }
        }

        // Write Pool Table
        try (BufferedWriter bw = new BufferedWriter(new FileWriter("pooltab.txt"))) {
            for (int p : POOLTAB) {
                bw.write(p + "\n");
            }
        }

        System.out.println("PASS-1 complete. Files generated: intermediate.txt, symtab.txt, littab.txt,
pooltab.txt");
    }

    static void initOPTAB() {
        OPTAB.put("START", new String[] { "AD", "01" });
        OPTAB.put("END", new String[] { "AD", "02" });
        OPTAB.put("LTORG", new String[] { "AD", "03" });
        OPTAB.put("DS", new String[] { "DL", "01" });
        OPTAB.put("DC", new String[] { "DL", "02" });
        OPTAB.put("MOVER", new String[] { "IS", "04" });
        OPTAB.put("MOVEM", new String[] { "IS", "05" });
        OPTAB.put("ADD", new String[] { "IS", "01" });
        OPTAB.put("SUB", new String[] { "IS", "02" });
    }

    static void process(String line) {
        if (line.isEmpty())
            return;
        String[] parts = line.split("\\s+");
        int idx = 0;
        String label = null;

        if (!OPTAB.containsKey(parts[idx])) {
            label = parts[idx++];
        }

        String opcode = parts[idx++];
        String[] code = OPTAB.get(opcode);

        switch (opcode) {
            case "START":
                LC = Integer.parseInt(parts[idx]);
                intermediate.add("(AD,01) (C," + LC + ")");
                break;

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

            case "LTORG":
                assignLiterals();
                intermediate.add("(AD,03)");
                break;

            case "DS":
            case "DC":
                if (label != null)
                    SYMTAB.put(label, LC);
                int size = Integer.parseInt(parts[idx]);
                intermediate.add("(DL," + code[1] + ") (C," + size + ")");
                LC += size;
                break;
            default:
                if (label != null)
                    SYMTAB.put(label, LC);
                String reg = parts[idx++];
                String operand = parts[idx];

                if (operand.startsWith("=")) {
                    if (!LITTAB.contains(operand)) {
                        LITTAB.add(operand);
                        LITADDR.put(operand, null);
                    }
                    intermediate.add(
                            "(IS," + code[1] + ") " + parseReg(reg) + " (L," + (LITTAB.indexOf(operand) + 1) + ")");
                } else {
                    if (!SYMTAB.containsKey(operand))
                        SYMTAB.put(operand, null);
                    intermediate.add("(IS," + code[1] + ") " + parseReg(reg) + " (S," + operand + ")");
                }
                LC++;
                break;
        }
    }

    static void assignLiterals() {
        int startIndex = POOLTAB.get(POOLTAB.size() - 1) - 1;
        for (int i = startIndex; i < LITTAB.size(); i++) {
            if (LITADDR.get(LITTAB.get(i)) == null) {
                LITADDR.put(LITTAB.get(i), LC);
                LC++;
            }
        }
        POOLTAB.add(LITTAB.size() + 1);
    }

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

Intermediate.txt

```
(AD,01) (C,100)
(IS,04) 0 (L,1)
(IS,01) 0 (L,2)
(IS,05) 0 (S,ALPHA)
(DL,01) (C,1)
(AD,03)
(IS,04) 0 (S,ALPHA)
(AD,02)
```

Symbo.txt

```
ALPHA 103
```

Literal.txt

```
1    =5    104
2    =3    105
```

Pooltab.txt

```
1
3
3
```

Pass 2 :

Java Program :

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

public class Pass2 {
    public static void main(String[] args) throws IOException {
        Map<String, Integer> SYMTAB = new HashMap<>();
        Map<Integer, String> LITTAB = new HashMap<>();

        // Load Symbol Table
        try (BufferedReader br = new BufferedReader(new FileReader("symtab.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] p = line.split("\\s+");
        SYMTAB.put(p[0], Integer.parseInt(p[1]));
    }
}

// Load Literal Table
try (BufferedReader br = new BufferedReader(new FileReader("littab.txt"))) {
    String line;
    while ((line = br.readLine()) != null) {
        String[] p = line.split("\\s+");
        int index = Integer.parseInt(p[0]);
        String val = p[1].replace("=", "").replace("'", "");
        LITTAB.put(index, val);
    }
}

BufferedReader ic = new BufferedReader(new FileReader("intermediate.txt"));
BufferedWriter mc = new BufferedWriter(new FileWriter("machinecode.txt"));

String line;
while ((line = ic.readLine()) != null) {
    if (line.startsWith("(AD") || line.startsWith("(DL,01)"))
        continue;

    if (line.startsWith("(DL,02)")) {
        String value = line.substring(line.indexOf("(C,") + 3, line.indexOf(")"));
        mc.write("00 0 " + value + "\n");
    } else if (line.startsWith("(IS")) {
        String[] parts = line.split("\\s+");
        String opcode = parts[0].substring(parts[0].indexOf(",") + 1, parts[0].indexOf(")"));
        String reg = parts[1];
        String operand = parts[2];

        if (operand.startsWith("(S,")) {
            String sym = operand.substring(3, operand.length() - 1);
            mc.write(opcode + " " + reg + " " + SYMTAB.get(sym) + "\n");
        } else if (operand.startsWith("(L,")) {
            int litIndex = Integer.parseInt(operand.substring(3, operand.length() - 1));
            mc.write(opcode + " " + reg + " " + LITTAB.get(litIndex) + "\n");
        }
    }
}

ic.close();
mc.close();
```

```
        System.out.println("PASS-2 complete. File generated: machinecode.txt");
    }
}
```

Machine Code :

```
04 0 5
01 0 3
05 0 103
04 0 103
```