PROGRAM CODE:

```
import java.util.*;
class chinuAssembler {
   static class Symbol {
      String name;
      int addr;
      Symbol(String x, int y) { name = x; addr = y; }
   }

   static class Literal {
      String value;
      int addr;
      Literal(String v, int a) { value = v; addr = a; }
   }

   static class Instruction {
      String type;
      String opcode;
      String operand1;
      String operand2;
      Instruction(String type, String opcode, String operand1, String operand2) {
         this.type = type; this.opcode = opcode; this.operand1 = operand1; this.operand2 = operand2;
      }
      public String toString() {
         return "(" + type + "," + opcode + ") " +
               (operand1 != null ? operand1 + " " : "") +
               (operand2 != null ? operand2 : "");
      }
   }

   static class OptabEntry {
      String mnemonic, opcode, type; // IS / AD / DL
      OptabEntry(String mnemonic, String opcode, String type) {
         this.mnemonic = mnemonic; this.opcode = opcode; this.type = type;
      }
   }

   static List<OptabEntry> OPTAB = new ArrayList<>();
   static List<Symbol>     SYMTAB = new ArrayList<>();
   static List<Literal>    LITTAB = new ArrayList<>();
   static List<Instruction> IR = new ArrayList<>();
   static int LC = 0;
   static int START_LC = 0;

   static Map<String,String> registerMap = new HashMap<>();
   static {
      registerMap.put("AREG","1");
      registerMap.put("BREG","2");
      registerMap.put("CREG","3");
   }
```

```
static void initializeOptab() {
   OPTAB.add(new OptabEntry("MOVER","01","IS"));
   OPTAB.add(new OptabEntry("ADD",  "02","IS"));
   OPTAB.add(new OptabEntry("STORE","03","IS"));
   OPTAB.add(new OptabEntry("READ", "04","IS"));
   OPTAB.add(new OptabEntry("PRINT","05","IS"));
   OPTAB.add(new OptabEntry("HALT", "06","IS"));

   OPTAB.add(new OptabEntry("START","01","AD"));
   OPTAB.add(new OptabEntry("END",  "02","AD"));

   OPTAB.add(new OptabEntry("DS","01","DL"));
   OPTAB.add(new OptabEntry("DC","02","DL"));
}

public static void main(String[] args) {
   initializeOptab();

   String[] code = {
      "START 100",
      "READ A",
      "READ B",
      "MOVER AREG, A",
      "ADD AREG, B",
      "ADD AREG, =1",
      "STORE AREG, SUM",
      "PRINT SUM",
      "HALT",
      "A DS 1",
      "B DS 1",
      "SUM DS 1",
      "END"
   };

   pass1(code);
   resolveAddresses();

   // Print tables
   System.out.println("SYMTAB:");
   for (Symbol s : SYMTAB) System.out.println(s.name + " -> " + s.addr);

   System.out.println("\nLITTAB:");
   if (LITTAB.isEmpty()) System.out.println("(empty)");
   else for (Literal l : LITTAB) System.out.println(l.value + " -> " + l.addr);

   System.out.println("\nINTERMEDIATE CODE (with addresses):");
   for (Instruction ic : IR) System.out.println(ic);

   System.out.println("\nTARGET CODE TABLE:");
   pass2();
```

```
}

// ---------- PASS 1 ----------
static void pass1(String[] code) {
   for (String raw : code) {
      String line = raw.trim().replaceAll("\\s+", " "); // normalize spaces
      if (line.isEmpty()) continue;

      String[] tokens = line.split(" ");
      String label = null, opcode, op1 = null, op2 = null;

      if (tokens[0].equals("START")) {
         LC = Integer.parseInt(tokens[1]);
         START_LC = LC;
         IR.add(new Instruction("AD","01","C,"+tokens[1],null));
         continue;
      }
      if (tokens[0].equals("END")) {
         // assign literal addresses at the end of program
         for (Literal l : LITTAB) if (l.addr == -1) l.addr = LC++;
         IR.add(new Instruction("AD","02",null,null));
         break;
      }

      // determine if first token is a mnemonic or a label
      if (isMnemonic(tokens[0])) {
         opcode = tokens[0];
         String after = substringAfterOpcode(line, opcode, /*labelLen*/0);
         String[] ops = splitOperands(after);
         if (ops.length > 0) op1 = ops[0];
         if (ops.length > 1) op2 = ops[1];
      } else {
         // labeled instruction / declaration
         label = tokens[0];
         if (!symbolExists(label)) SYMTAB.add(new Symbol(label, -1));
         opcode = tokens[1];

         int labelLen = label.length();
         String after = substringAfterOpcode(line, opcode, labelLen);
         String[] ops = splitOperands(after);
         if (ops.length > 0) op1 = ops[0];
         if (ops.length > 1) op2 = ops[1];
      }

      OptabEntry entry = getOptabEntry(opcode);
      if (entry == null) {
         System.out.println("Invalid Opcode: " + opcode);
         continue;
      }

      if (entry.type.equals("DL")) {
```

```
            // define label address for DS/DC
            if (label != null) updateSymbolAddress(label, LC);
            // record IR with constant
            if (op1 == null) op1 = "0";
            IR.add(new Instruction("DL", entry.opcode, "C," + op1, null));
            if (opcode.equals("DS")) LC += Integer.parseInt(op1);
            else LC++;
          } else {
            // normal instruction
            if (label != null) updateSymbolAddress(label, LC);

            // collect symbol/literal references for SYMTAB/LITTAB
            if (op1 != null && !registerMap.containsKey(op1) && !op1.startsWith("=") &&
!symbolExists(op1))
               SYMTAB.add(new Symbol(op1, -1));
            if (op2 != null) {
               if (op2.startsWith("=")) {
                  if (!literalExists(op2)) LITTAB.add(new Literal(op2, -1));
               } else if (!symbolExists(op2)) {
                  SYMTAB.add(new Symbol(op2, -1));
               }
            }

            // encode operands in IR with R/S/L tags
            String irOp1 = null, irOp2 = null;
            if (op1 != null) {
               if (registerMap.containsKey(op1)) irOp1 = "R," + registerMap.get(op1);
               else if (op1.startsWith("="))     irOp1 = "L," + op1;
               else                              irOp1 = "S," + op1;
            }
            if (op2 != null) {
               if (op2.startsWith("="))          irOp2 = "L," + op2;
               else if (registerMap.containsKey(op2)) irOp2 = "R," + registerMap.get(op2); // just in case
               else                              irOp2 = "S," + op2;
            }

            IR.add(new Instruction(entry.type, entry.opcode, irOp1, irOp2));
            LC++;
          }
        }
      }

      // Return the substring after the opcode (handles "AREG, A" correctly)
      static String substringAfterOpcode(String line, String opcode, int labelLen) {
        int start = (labelLen > 0) ? line.indexOf(opcode, labelLen) : line.indexOf(opcode);
        if (start < 0) return "";
        String after = line.substring(start + opcode.length()).trim();
        return after;
      }

      // Split operand string by comma, trimming spaces; returns 0, 1, or 2 operands.
```

```
static String[] splitOperands(String after) {
   if (after.isEmpty()) return new String[0];
   String[] parts = after.split(",", 2);
   String a = parts[0].trim();
   if (parts.length == 1) {
      return a.isEmpty() ? new String[0] : new String[]{a};
   }
   String b = parts[1].trim();
   if (a.isEmpty()) return b.isEmpty()? new String[0] : new String[]{b};
   return b.isEmpty()? new String[]{a} : new String[]{a, b};
}

// ---------- Resolve addresses in IR ----------
static void resolveAddresses() {
   for (Instruction ic : IR) {
      if (ic.operand1 != null) {
         if (ic.operand1.startsWith("S,")) {
            String sym = ic.operand1.substring(2);
            ic.operand1 = "S," + getSymbolAddress(sym);
         } else if (ic.operand1.startsWith("L,")) {
            String lit = ic.operand1.substring(2);
            ic.operand1 = "L," + getLiteralAddress(lit);
         }
      }
      if (ic.operand2 != null) {
         if (ic.operand2.startsWith("S,")) {
            String sym = ic.operand2.substring(2);
            ic.operand2 = "S," + getSymbolAddress(sym);
         } else if (ic.operand2.startsWith("L,")) {
            String lit = ic.operand2.substring(2);
            ic.operand2 = "L," + getLiteralAddress(lit);
         }
      }
   }
}

// ---------- PASS 2: generate target code table ----------
static void pass2() {
   int loc = START_LC;
   System.out.printf("%-5s %-7s %-5s %-5s%n", "LC", "OPCODE", "REG", "ADDR");
   for (Instruction ic : IR) {
      if (ic.type.equals("AD")) continue; // no code for AD

      if (ic.type.equals("DL")) {
         // DC emits value, DS reserves space (0)
         if ("02".equals(ic.opcode)) { // DC
            String val = ic.operand1.split(",", 2)[1];
            System.out.printf("%-5d %-7s %-5s %-5s%n", loc, "-", "-", val);
         } else { // DS
            System.out.printf("%-5d %-7s %-5s %-5s%n", loc, "-", "-", "0");
         }
```

```
            loc++;
            continue;
         }

         if (ic.type.equals("IS")) {
            String reg = "0";
            String addr = "0";

            if (ic.operand1 != null) {
               if (ic.operand1.startsWith("R,")) reg = ic.operand1.split(",",2)[1];
               else if (ic.operand1.startsWith("S,") || ic.operand1.startsWith("L,")) addr =
ic.operand1.split(",",2)[1];
            }
            if (ic.operand2 != null) {
               if (ic.operand2.startsWith("S,") || ic.operand2.startsWith("L,")) addr =
ic.operand2.split(",",2)[1];
               else if (ic.operand2.startsWith("R,")) reg = ic.operand2.split(",",2)[1];
            }

            System.out.printf("%-5d %-7s %-5s %-5s%n", loc, ic.opcode, reg, addr);
            loc++;
         }
      }
   }

   // ---------- helpers ----------
   static boolean isMnemonic(String word) {
      for (OptabEntry e : OPTAB) if (e.mnemonic.equals(word)) return true;
      return false;
   }

   static boolean symbolExists(String symbol) {
      for (Symbol s : SYMTAB) if (s.name.equals(symbol)) return true;
      return false;
   }

   static boolean literalExists(String lit) {
      for (Literal l : LITTAB) if (l.value.equals(lit)) return true;
      return false;
   }

   static OptabEntry getOptabEntry(String mnemonic) {
      for (OptabEntry e : OPTAB) if (e.mnemonic.equals(mnemonic)) return e;
      return null;
   }

   static int getSymbolAddress(String symbol) {
      for (Symbol s : SYMTAB) if (s.name.equals(symbol)) return s.addr;
      return -1;
   }
```

```
    static int getLiteralAddress(String lit) {
        for (Literal l : LITTAB) if (l.value.equals(lit)) return l.addr;
        return -1;
    }

    static void updateSymbolAddress(String symbol, int addr) {
        for (Symbol s : SYMTAB) if (s.name.equals(symbol)) { s.addr = addr; return; }
    }
}
```

OUTPUT:

```
SYMTAB:
A -> 108
B -> 109
SUM -> 110

LITTAB:
=1 -> 111

INTERMEDIATE CODE (with addresses):
(AD,01) C,100
(IS,04) S,108
(IS,04) S,109
(IS,01) R,1 S,108
(IS,02) R,1 S,109
(IS,02) R,1 L,111
(IS,03) R,1 S,110
(IS,05) S,110
(IS,06)
(DL,01) C,1
(DL,01) C,1
(DL,01) C,1
(AD,02)

TARGET CODE TABLE:
LC    OPCODE  REG   ADDR
100   04      0     108
101   04      0     109
102   01      1     108
103   02      1     109
104   02      1     111
105   03      1     110
106   05      0     110
107   06      0     0
108   -       -     0
109   -       -     0
110   -       -     0
```