



Dr. Vishwanath Karad  
**MIT WORLD PEACE**  
**UNIVERSITY** | PUNE  
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

## School of CET

### System Software and Compiler lab

#### Assignment No.1

#### TY BTech CSE

**Assignment Title:** Design of Pass 1 of Two Pass Assembler.

**Aim:** Design suitable data structure & implement pass 1 of Two Pass Assembler pseudo machine.

**Objective:** Design suitable data structure & implement pass 1 of Two Pass Assembler pseudo machine. Subset should consist of a few instructions from each category & few assembler directive.

#### Theory:

##### Assembler

The assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.

It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

- **Pass-1:**
  1. Define symbols and literals and remember them in symbol table and literal table respectively.
  2. Keep track of location counter
  3. Process pseudo-operations
- **Pass-2:**
  1. Generate object code by converting symbolic op-code into respective numeric op-code

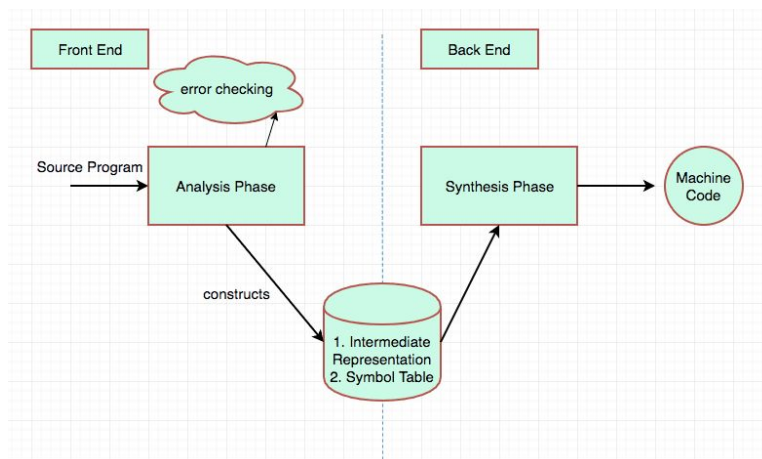
2. Generate data for literals and look for values of symbols

### Design specification of an Assembler

Analysis phase: reads the source program and splits it into multiple tokens and constructs the intermediate representation of the source program. And also checks and indicates the syntax and semantic errors of a source program. It collects information about the source program and prepares the symbol table. Symbol table will be used all over the compilation process. This is also called as the front end of a compiler.

Synthesis phase: It will get the analysis phase input(intermediate representation and symbol table) and produces the targeted **machine level code**.

This is also called as the **back end** of a compiler.



### Algorithm

#### Algorithm for Pass I

1. loccntr: =0; (default value)
2. While next statement is not an END statement
  - (a) If label is present then

*this-label*: = symbol in label field;  
Enter (*this-label*, *locctr*) in SYMTAB.

- (b) If a START or ORIGIN statement then  
 $locntr := \text{value specified in operand field};$
- (c) If an EQU statement then
  - (i)  $this-addr := \text{value of } \langle address\ spec \rangle;$
  - (ii) Correct the symtab entry for  $this-label$  to  $(this-label, this-addr)$ .
- (d) If a declaration statement then
  - (i)  $code := \text{code of the declaration statement};$
  - (ii)  $size := \text{size of memory area required by DC/DS}.$
  - (iii)  $locntr := locntr + size;$
  - (iv) Generate IC '(DL, code)'.
- (f) If an imperative statement then
  - (i)  $code := \text{machine opcode from OPTAB};$
  - (ii)  $locntr := locntr + \text{instruction length from OPTAB};$
  - (iii) If operand is a symbol then  
 $this-entry := \text{SYMTAB entry number of operands};$   
Generate IC '(IS, code) (S, this-entry)';

### 3. (Processing of END statement)

- (b) Generate IC
- (c) Go to Pass II.

### Listing & Error Handling:

**Input:** Assembly Language Program. /Intermediate code generated by PASS I

#### Output:

##### 1. Mnemonic Table

Mnemonic	Op-code	Length

##### 2. Symbol Table

Symbols/ Labels	Address

##### 3. Intermediate Form (After Pass I)/Final Output

Address (LC value)	Op-code	Operand 1 (Temp)	Operand 2 (Temp)

**Conclusion:** The function of Pass I in assembler are studied along with errors coming in each pass.

**Platform:** Linux (JAVA)

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

class Condition {
    String name;
    int no;

    Condition(String a, int op) {
        this.name = a;
        this.no = op;
    }
}

class Register {
    String name;
    int mc_code;

    Register(String a, int op) {
        this.name = a;
        this.mc_code = op;
    }
}

class Operator {
    String name;
    String cls;
    int opcode;

    Operator(String a, String c, int op) {
        this.name = a;
        this.cls = c;
        this.opcode = op;
    }
}
```

```
    }  
}  
  
class Symbol {  
    String name;  
    int addr;  
    int length;  
  
    Symbol(String a, int op, int len) {  
        this.name = a;  
        this.addr = op;  
        this.length = len;  
    }  
}  
  
class Tuple<X, Y> {  
    public X x;  
    public Y y;  
  
    public Tuple(X x, Y y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
public class assembler_rg {  
  
    // static ArrayList<Symbol> symbolTable = new  
ArrayList<Symbol>(25);  
    static ArrayList<String> intermed_code = new  
ArrayList<String>(25);  
    static ArrayList<Tuple<String, String>> literalTable = new  
ArrayList<Tuple<String, String>>(25);  
    static ArrayList<Tuple<String, String>> symbolTable = new  
ArrayList<Tuple<String, String>>(25);  
    static ArrayList<Integer> poolTable = new  
ArrayList<Integer>(25);  
}
```

```
        static      ArrayList<Operator>      aList      =      new
ArrayList<Operator>(25);

        static      ArrayList<Register>      rList      =      new
ArrayList<Register>(25);

        static      ArrayList<Condition>      cList      =      new
ArrayList<Condition>(25);

static void def_aList() {
    aList.add(new Operator("STOP", "IS", 0));
    aList.add(new Operator("ADD", "IS", 1));
    aList.add(new Operator("SUB", "IS", 2));
    aList.add(new Operator("MULT", "IS", 3));
    aList.add(new Operator("MOVER", "IS", 4));
    aList.add(new Operator("MOVEM", "IS", 5));
    aList.add(new Operator("COMP", "IS", 6));
    aList.add(new Operator("BC", "IS", 7));
    aList.add(new Operator("DIV", "IS", 8));
    aList.add(new Operator("READ", "IS", 9));
    aList.add(new Operator("PRINT", "IS", 10));
    aList.add(new Operator("DC", "DL", 2));
    aList.add(new Operator("DS", "DL", 1));
    aList.add(new Operator("START", "AD", 1));
    aList.add(new Operator("END", "AD", 2));
    aList.add(new Operator("ORIGIN", "AD", 3));
    aList.add(new Operator("EQU", "AD", 4));
    aList.add(new Operator("LTORG", "AD", 5));
}

static void def_rlist() {
    rList.add(new Register("AREG", 1));
    rList.add(new Register("BREG", 2));
    rList.add(new Register("CREG", 3));
    rList.add(new Register("DREG", 4));
}

static void def_clist() {
```

```
cList.add(new Condition("LT", 1));
cList.add(new Condition("LE", 2));
cList.add(new Condition("EQ", 3));
cList.add(new Condition("GT", 4));
cList.add(new Condition("GE", 5));
cList.add(new Condition("ANY", 6));
}

// static void def_printsymbolTable()
// {
// for(int i=0;i<symbolTable.size();i++)
// {
// Symbol e=symbolTable.get(i);
// System.out.println(e.name+" "+e.addr+" "+e.length);
// }
// }

static void def_printIntermediateCode() {
    for (int i = 0; i < intermed_code.size(); i++) {
        System.out.println(intermed_code.get(i));
    }
}

static void def_printLiteralTable() {
    System.out.println("-----LITERAL TABLE-----");
    for (int i = 0; i < literalTable.size(); i++) {
        System.out.println("| [" + literalTable.get(i).x +
", " + literalTable.get(i).y + "]"");
    }
    System.out.println("-----");
}

static void def_printSymbolTable() {
    System.out.println("-----SYMBOL TABLE-----");
    for (int i = 0; i < symbolTable.size(); i++) {
        System.out.println("| [" + symbolTable.get(i).x + ", "
+ symbolTable.get(i).y + "]"");
    }
}
```

```
        System.out.println("-----");
    }

    static void def_printPoolTable()
    {
        System.out.println("-----POOL TABLE-----");
        for (int i = 0; i < poolTable.size(); i++) {
            System.out.println("| "+poolTable.get(i)+ " |");
        }
        System.out.println("-----");
    }

    static boolean intparseable(String s) {
        try {
            Integer.parseInt(s);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

    static boolean InAList(String s) {
        for (int i = 0; i < aList.size(); i++) {
            if (s.equals(aList.get(i).name)) {
                return true;
            }
        }
        return false;
    }

    public static void main(String args[]) throws IOException {
        /// Making the opp list
        def_aList();
        def_rlist();
        int line = 0;
        BufferedReader br = new BufferedReader(new
        FileReader("assembly_rg.txt"));
```



```
String str;
int line_number = 0;
while ((str = br.readLine()) != null) {

    String[] splitted = str.split("\\s+");

    System.out.println("#####\n");
    if (line_number == 0) {
        line_number = Integer.parseInt(splitted[1]) - 1;
        System.out.println(Arrays.toString(splitted));
    } else {

        System.out.println((line_number+1) + ": " +
Arrays.toString(splitted));
    }

    if (!splitted[0].equals("-")) // if not starts with -
    {

        if (splitted.length >= 2) {
            if (splitted.length == 3) {
                // MOVEM AREG,X
                System.out.println("splitted[2]:" +
splitted[2]);

                if (InAList(splitted[0])) {
                    if (splitted[2].contains("=")) // put
into literal table
                    {
                        Tuple<String, String> lt = new
Tuple<String, String>(splitted[2], "Address not given");
                        literalTable.add(lt);
                        def_printLiteralTable();
                    }

                    else if (!splitted[2].contains("+"))
{// put into symbol table
```

```

        Tuple<String, String> st = new
Tuple<String, String>(splitted[2], "Address not given");
        symbolTable.add(st);
        def_printSymbolTable();
    }
    } else {
        // X DS 1
        for (int k = 0; k <
symbolTable.size(); k++) {
            Tuple<String, String> s =
symbolTable.get(k);
            if (s.x.equals(splitted[0])) {
                intermed_code.add("(S," + k +
")");
                s.y=(line_number+1)+"";
                break;
            }
        }
        String[] newArray =
Arrays.copyOfRange(splitted, 1, splitted.length);
        splitted = newArray;
    }
}
}
if (splitted.length == 4) {
    Tuple<String, String> s = new
Tuple<String, String>(splitted[0], line_number + "");
    symbolTable.add(s);
    intermed_code.add("(S," +
(symbolTable.size() - 1) + ")");
    // for(int k=0;k<symbolTable.size();k++)
    // {

```

```

// Tuple<String,String>
s=symbolTable.get(k);

// if(s.x.equals(splitted[2]))
// {
//   intermed_code.add("(S,"+k+"");
//   break;
// }

// }

if (splitted[3].contains("="))// put into
literal table
{
    Tuple<String, String> lt = new
Tuple<String, String>(splitted[3], "Address not given");
    literalTable.add(lt);
    def_printLiteralTable();
} else { // put into symbol table

    Tuple<String, String> st = new
Tuple<String, String>(splitted[3], "Address not given");
    symbolTable.add(st);
    def_printSymbolTable();
}

String[] newArray =
Arrays.copyOfRange(splitted, 1, splitted.length);
splitted = newArray;
def_printSymbolTable();

}

Operator x = null;
Iterator<Operator> i = aList.iterator();
while (i.hasNext()) {
    x = i.next();
    if (x.name.equals(splitted[0])) {

```

```

// System.out.println(x.name+"
"+x.cls+" "+x.opcode);

        intermed_code.add("(" + x.cls + "," +
x.opcode + ")");

    }

}

    if (intparseable(splitted[1])) {
        intermed_code.add(new String("(" + "C" +
", " + Integer.parseInt(splitted[1]) + ")"));
    } else if (splitted[1].contains("REG")) {

        Register r = null; // name, mc_code
        Iterator<Register> j = rList.iterator();
        while (j.hasNext()) {
            r = j.next();
            // System.out.println(r.name);

//
System.out.println("oo"+splitted[1].substring(0,splitted[1].length
h()-1));

            if
(r.name.equals(splitted[1].substring(0, splitted[1].length() -
1))) {

                intermed_code.add("(RG," +
r.mc_code + ")");

            }

        }

    }

    if (splitted.length == 3 &&
splitted[2].contains("=")) {
        for (int k = 0; k < literalTable.size();
k++) {

            Tuple<String, String> s =
literalTable.get(k);

            if (s.x.equals(splitted[2])) {

```

```
intermed_code.add("(L," + k +
");");
break;
}
}
} else if (splitted.length == 3 &&
!splitted[2].contains("+")) {
for (int k = 0; k < symbolTable.size();
k++) {
Tuple<String, String> s =
symbolTable.get(k);
if (s.x.equals(splitted[2])) {
intermed_code.add("(S," + k +
");");
break;
}
}
}
if (splitted.length == 3 &&
splitted[2].contains("+")) // ORIGIN L1 +3
{
int indicated_line_number = line_number;
for (int gg = 0; gg < symbolTable.size();
gg++) {
Tuple<String, String> s =
symbolTable.get(gg);
if (s.x.equals(splitted[1])) {
indicated_line_number =
Integer.parseInt(s.y);
}
}
line_number = indicated_line_number +
Integer.parseInt(splitted[2].substring(1)) - 1;
intermed_code.add("(C," + (line_number +
1) + ")");
```

```
    }

    }
    if (splitted.length == 1) // LTORG
    {
        if(splitted[0].equals("LTORG"))
        {
            int gg = line_number;
            for (int i = 0; i < literalTable.size();
i++) {

                literalTable.get(i).y = (gg++) + "";

                Operator x = null;
                Iterator<Operator> hh =
aList.iterator();

                while (hh.hasNext()) {
                    x = hh.next();
                    if (x.name.equals(splitted[0])) {
                        // System.out.println(x.name+"
"+x.cls+" "+x.opcode);

                        intermed_code.add("(" + x.cls
+ "," + x.opcode + ")");

                    }

                }

                intermed_code.add("(DL,2)");

                intermed_code.add("(C,"+literalTable.get(i).x.substring(2,3)+"")
;

            }

            poolTable.add(0); //ideally wherever i
begins for literal table;

            def_printLiteralTable();
            def_printPoolTable();

        }
    }
```

```
        if (splitted[0].equals("END"))
        {
            System.out.println("gggggggggg");
            Operator x = null;
            Iterator<Operator> hh =
aList.iterator();
            while (hh.hasNext()) {
                x = hh.next();
                if (x.name.equals(splitted[0])) {
                    // System.out.println(x.name+"
"+x.cls+" "+x.opcode);
                    intermed_code.add("(" + x.cls
+ "," + x.opcode + ")");
                }
            }
        }
    }

    def_printIntermediateCode();
    def_printLiteralTable();
    def_printSymbolTable();
    def_printPoolTable();
    line_number++;
}
// def_printsymbolTable();

br.close();
}
}
```

```
/*
START 200
MOVER AREG, ='5'
MOVEM AREG, X
L1 MOVER BREG, ='2'
ORIGIN L1 +3
LTORG
X DS 1
END

#####

[START, 200]
(AD,1)
(C,200)
-----LITERAL TABLE-----
-----
-----SYMBOL TABLE-----
-----
-----POOL TABLE-----
-----
#####

201: [MOVER, AREG,, ='5']
splitted[2]:='5'
-----LITERAL TABLE-----
| [[='5',Address not given]]
-----
(AD,1)
(C,200)
(IS,4)
(RG,1)
(L,0)
-----LITERAL TABLE-----
| [[='5',Address not given]]
-----
-----SYMBOL TABLE-----
```



```
-----  
----POOL TABLE-----  
-----  
#####  
  
202: [MOVEM, AREG,, X]  
splitted[2]:X  
----SYMBOL TABLE-----  
| [[X,Address not given]]  
-----  
  
(AD,1)  
(C,200)  
(IS,4)  
(RG,1)  
(L,0)  
(IS,5)  
(RG,1)  
(S,0)  
  
----LITERAL TABLE-----  
| [[='5',Address not given]]  
-----  
  
----SYMBOL TABLE-----  
| [[X,Address not given]]  
-----  
  
----POOL TABLE-----  
-----  
#####  
  
203: [L1, MOVER, BREG,, ='2']  
----LITERAL TABLE-----  
| [[='5',Address not given]]  
| [[='2',Address not given]]  
-----  
  
----SYMBOL TABLE-----  
| [[X,Address not given]]  
| [[L1,202]]  
-----
```

```
(AD,1)
(C,200)
(IS,4)
(RG,1)
(L,0)
(IS,5)
(RG,1)
(S,0)
(S,1)
(IS,4)
(RG,2)
(L,1)
-----LITERAL TABLE-----
| [['5',Address not given]]
| [['2',Address not given]]
|
-----SYMBOL TABLE-----
| [[X,Address not given]]
| [[L1,202]]
|
-----POOL TABLE-----
|
#####

204: [ORIGIN, L1, +3]
splitted[2]:+3
(AD,1)
(C,200)
(IS,4)
(RG,1)
(L,0)
(IS,5)
(RG,1)
(S,0)
(S,1)
(IS,4)
(RG,2)
```

```
(L,1)
(AD,3)
(C,205)
-----LITERAL TABLE-----
| [['5',Address not given]]
| [['2',Address not given]]
|
-----SYMBOL TABLE-----
| [[X,Address not given]]
| [[L1,202]]
|
-----POOL TABLE-----
|
#####

206: [LTORG]
-----LITERAL TABLE-----
| [['5',205]]
| [['2',206]]
|
-----POOL TABLE-----
| 0 |
|
(AD,1)
(C,200)
(IS,4)
(RG,1)
(L,0)
(IS,5)
(RG,1)
(S,0)
(S,1)
(IS,4)
(RG,2)
(L,1)
(AD,3)
(C,205)
```

```
(AD,5)
(DL,2)
(C,5)
(AD,5)
(DL,2)
(C,2)
-----LITERAL TABLE-----
|  [['5',205]]
|  [['2',206]]
-----
-----SYMBOL TABLE-----
|  [[X,Address not given]]
|  [[L1,202]]
-----
-----POOL TABLE-----
|  0  |
-----
#####

207: [X, DS, 1]
splitted[2]:1
(AD,1)
(C,200)
(IS,4)
(RG,1)
(L,0)
(IS,5)
(RG,1)
(S,0)
(S,1)
(IS,4)
(RG,2)
(L,1)
(AD,3)
(C,205)
(AD,5)
(DL,2)
```

```
(C,5)
(AD,5)
(DL,2)
(C,2)
(S,0)
(DL,1)
(C,1)
-----LITERAL TABLE-----
|  [['5',205]]
|  [['2',206]]
-----
-----SYMBOL TABLE-----
|  [[X,207]]
|  [[L1,202]]
-----
-----POOL TABLE-----
|  0  |
-----
#####

208:  [END]
ggggggggggg
(AD,1)
(C,200)
(IS,4)
(RG,1)
(L,0)
(IS,5)
(RG,1)
(S,0)
(S,1)
(IS,4)
(RG,2)
(L,1)
(AD,3)
(C,205)
(AD,5)
```

```
(DL,2)
(C,5)
(AD,5)
(DL,2)
(C,2)
(S,0)
(DL,1)
(C,1)
(AD,2)
-----LITERAL TABLE-----
|  [['5',205]]
|  [['2',206]]
-----
-----SYMBOL TABLE-----
|  [['X',207]]
|  [['L1',202]]
-----
-----POOL TABLE-----
|  0  |
-----
*/
```

```

1 import java.io.*;
2 import java.util.*;
3
4 class Condition {
5     String name;
6     int no;
7
8     Condition(String a, int op) {
9         this.name = a;
10        this.no = op;
11    }
12 }
13
14 class Register {
15     String name;
16     int mc_code;
17
18     Register(String a, int op) {
19         this.name = a;
20         this.mc_code = op;
21     }
22 }
23
24 class Operator {
25     String name;
26     String cls;
27     int opcode;
28
29     Operator(String a, String c, int op) {
30         this.name = a;
31         this.cls = c;
32         this.opcode = op;
33     }
34 }
35
36 class Symbol {
37     String name;
38     int addr;
39     int length;
40
41     Symbol(String a, int op, int len) {
42         this.name = a;
43         this.addr = op;
44         this.length = len;
45     }
46 }
47
48 class Tuple<X, Y> {
49     public X x;
50     public Y y;
51
52     public Tuple(X x, Y y) {
53         this.x = x;
54         this.y = y;
55     }
56 }
57
58 public class assembler_rg {
59
60     // static ArrayList<Symbol> symbolTable = new ArrayList<Symbol>(25);

```

```

61     static ArrayList<String> intermed_code = new ArrayList<String>(25);
62     static ArrayList<Tuple<String, String>> literalTable = new
ArrayList<Tuple<String, String>>(25);
63     static ArrayList<Tuple<String, String>> symbolTable = new
ArrayList<Tuple<String, String>>(25);
64     static ArrayList<Integer> poolTable = new ArrayList<Integer>(25);
65
66     static ArrayList<Operator> aList = new ArrayList<Operator>(25);
67     static ArrayList<Register> rList = new ArrayList<Register>(25);
68     static ArrayList<Condition> cList = new ArrayList<Condition>(25);
69
70     static void def_aList() {
71         aList.add(new Operator("STOP", "IS", 0));
72         aList.add(new Operator("ADD", "IS", 1));
73         aList.add(new Operator("SUB", "IS", 2));
74         aList.add(new Operator("MULT", "IS", 3));
75         aList.add(new Operator("MOVER", "IS", 4));
76         aList.add(new Operator("MOVEM", "IS", 5));
77         aList.add(new Operator("COMP", "IS", 6));
78         aList.add(new Operator("BC", "IS", 7));
79         aList.add(new Operator("DIV", "IS", 8));
80         aList.add(new Operator("READ", "IS", 9));
81         aList.add(new Operator("PRINT", "IS", 10));
82         aList.add(new Operator("DC", "DL", 2));
83         aList.add(new Operator("DS", "DL", 1));
84         aList.add(new Operator("START", "AD", 1));
85         aList.add(new Operator("END", "AD", 2));
86         aList.add(new Operator("ORIGIN", "AD", 3));
87         aList.add(new Operator("EQU", "AD", 4));
88         aList.add(new Operator("LTORG", "AD", 5));
89     }
90
91     static void def_rlist() {
92         rList.add(new Register("AREG", 1));
93         rList.add(new Register("BREG", 2));
94         rList.add(new Register("CREG", 3));
95         rList.add(new Register("DREG", 4));
96     }
97
98     static void def_clist() {
99         cList.add(new Condition("LT", 1));
100        cList.add(new Condition("LE", 2));
101        cList.add(new Condition("EQ", 3));
102        cList.add(new Condition("GT", 4));
103        cList.add(new Condition("GE", 5));
104        cList.add(new Condition("ANY", 6));
105    }
106
107    // static void def_printsymbolTable()
108    // {
109    //     for(int i=0;i<symbolTable.size();i++)
110    //     {
111    //         Symbol e=symbolTable.get(i);
112    //         System.out.println(e.name+" "+e.addr+" "+e.length);
113    //     }
114    // }
115    static void def_printIntermediateCode() {
116        for (int i = 0; i < intermed_code.size(); i++) {
117            System.out.println(intermed_code.get(i));
118        }

```



```

119     }
120
121     static void def_printLiteralTable() {
122         System.out.println("-----LITERAL TABLE-----");
123         for (int i = 0; i < literalTable.size(); i++) {
124             System.out.println("| [" + literalTable.get(i).x + "," +
literalTable.get(i).y + "]]");
125         }
126         System.out.println("-----");
127     }
128
129     static void def_printSymbolTable() {
130         System.out.println("-----SYMBOL TABLE-----");
131         for (int i = 0; i < symbolTable.size(); i++) {
132             System.out.println("| [" + symbolTable.get(i).x + "," +
symbolTable.get(i).y + "]]");
133         }
134         System.out.println("-----");
135     }
136     static void def_printPoolTable()
137     {
138         System.out.println("-----POOL TABLE-----");
139         for (int i = 0; i < poolTable.size(); i++) {
140             System.out.println("| "+poolTable.get(i)+ " |");
141         }
142         System.out.println("-----");
143     }
144
145
146     static boolean intparseable(String s) {
147         try {
148             Integer.parseInt(s);
149             return true;
150         } catch (NumberFormatException e) {
151             return false;
152         }
153     }
154
155     static boolean InAList(String s) {
156         for (int i = 0; i < aList.size(); i++) {
157             if (s.equals(aList.get(i).name)) {
158                 return true;
159             }
160         }
161         return false;
162     }
163
164     public static void main(String args[]) throws IOException {
165         /// Making the opp list
166         def_aList();
167         def_rlist();
168         int line = 0;
169         BufferedReader br = new BufferedReader(new
FileReader("assembly_rg.txt"));
170         String str;
171         int line_number = 0;
172         while ((str = br.readLine()) != null) {
173
174             String[] splitted = str.split("\\s+");
175

```

```

176 System.out.println("#####\n");
177 if (line_number == 0) {
178     line_number = Integer.parseInt(splitted[1]) - 1;
179     System.out.println(Arrays.toString(splitted));
180 } else {
181
182     System.out.println((line_number+1) + ": " +
Arrays.toString(splitted));
183 }
184
185 if (!splitted[0].equals("-"))// if not starts with -
186 {
187
188     if (splitted.length >= 2) {
189         if (splitted.length == 3) {
190             // MOVEM AREG,X
191             System.out.println("splitted[2]:" + splitted[2]);
192
193             if (InAList(splitted[0])) {
194                 if (splitted[2].contains("="))// put into literal
table
195                 {
196                     Tuple<String, String> lt = new Tuple<String,
String>(splitted[2], "Address not given");
197                     literalTable.add(lt);
198                     def_printLiteralTable();
199                 }
200
201                 else if (!splitted[2].contains("+")){// put into
symbol table
202
203                     Tuple<String, String> st = new Tuple<String,
String>(splitted[2], "Address not given");
204                     symbolTable.add(st);
205                     def_printSymbolTable();
206                 }
207             } else {
208                 // X DS 1
209                 for (int k = 0; k < symbolTable.size(); k++) {
210                     Tuple<String, String> s = symbolTable.get(k);
211                     if (s.x.equals(splitted[0])) {
212                         intermed_code.add("(S," + k + ")");
213                         s.y=(line_number+1)+"";
214                         break;
215                     }
216                 }
217             }
218             String[] newArray = Arrays.copyOfRange(splitted,
1, splitted.length);
219             splitted = newArray;
220
221
222
223         }
224
225     }
226     if (splitted.length == 4) {
227         Tuple<String, String> s = new Tuple<String, String>
(splitted[0], line_number + "");
228         symbolTable.add(s);

```

```

229         intermed_code.add("(S," + (symbolTable.size() - 1) +
")");
230         // for(int k=0;k<symbolTable.size();k++)
231         // {
232         // Tuple<String,String> s=symbolTable.get(k);
233         // if(s.x.equals(splitted[2]))
234         // {
235         // intermed_code.add("(S,"+k+")");
236         // break;
237         // }
238
239         // }
240
241         if (splitted[3].contains("="))// put into literal
table
242         {
243             Tuple<String, String> lt = new Tuple<String,
String>(splitted[3], "Address not given");
244             literalTable.add(lt);
245             def_printLiteralTable();
246         } else { // put into symbol table
247
248             Tuple<String, String> st = new Tuple<String,
String>(splitted[3], "Address not given");
249             symbolTable.add(st);
250             def_printSymbolTable();
251         }
252         String[] newArray = Arrays.copyOfRange(splitted, 1,
splitted.length);
253         splitted = newArray;
254         def_printSymbolTable();
255
256     }
257
258     Operator x = null;
259     Iterator<Operator> i = aList.iterator();
260     while (i.hasNext()) {
261         x = i.next();
262         if (x.name.equals(splitted[0])) {
263             // System.out.println(x.name+" "+x.cls+"
"+x.opcode);
264             intermed_code.add("(" + x.cls + "," + x.opcode +
")");
265         }
266     }
267
268     if (Integer.parseInt(splitted[1])) {
269         intermed_code.add(new String("(" + "C" + "," +
Integer.parseInt(splitted[1]) + ")"));
270     } else if (splitted[1].contains("REG")) {
271
272         Register r = null; // name,mc_code
273         Iterator<Register> j = rList.iterator();
274         while (j.hasNext()) {
275             r = j.next();
276             // System.out.println(r.name);
277             //
278             System.out.println("oo"+splitted[1].substring(0,splitted[1].length()-1));

```

```

279         if (r.name.equals(splitted[1].substring(0,
splitted[1].length() - 1))) {
280             intermed_code.add("(RG," + r.mc_code + ")");
281         }
282     }
283 }
284 if (splitted.length == 3 && splitted[2].contains("=")) {
285     for (int k = 0; k < literalTable.size(); k++) {
286         Tuple<String, String> s = literalTable.get(k);
287         if (s.x.equals(splitted[2])) {
288             intermed_code.add("(L," + k + ")");
289             break;
290         }
291     }
292 }
293 } else if (splitted.length == 3 &&
!splitted[2].contains("+")) {
294     for (int k = 0; k < symbolTable.size(); k++) {
295         Tuple<String, String> s = symbolTable.get(k);
296         if (s.x.equals(splitted[2])) {
297             intermed_code.add("(S," + k + ")");
298             break;
299         }
300     }
301 }
302 }
303 if (splitted.length == 3 && splitted[2].contains("+"))//
ORIGIN L1 +3
304 {
305     int indicated_line_number = line_number;
306     for (int gg = 0; gg < symbolTable.size(); gg++) {
307         Tuple<String, String> s = symbolTable.get(gg);
308         if (s.x.equals(splitted[1])) {
309             indicated_line_number =
Integer.parseInt(s.y);
310         }
311     }
312     line_number = indicated_line_number +
Integer.parseInt(splitted[2].substring(1)) - 1;
313     intermed_code.add("(C," + (line_number + 1) + ")");
314 }
315 }
316 if (splitted.length == 1)// LTORG
317 {
318     if(splitted[0].equals("LTORG"))
319     {
320         int gg = line_number;
321         for (int i = 0; i < literalTable.size(); i++) {
322             literalTable.get(i).y = (gg++) + "";
323         }
324         Operator x = null;
325         Iterator<Operator> hh = aList.iterator();
326         while (hh.hasNext()) {
327             x = hh.next();
328             if (x.name.equals(splitted[0])) {
329                 // System.out.println(x.name+" "+x.cls+"
"+x.opcode);
330                 intermed_code.add("(" + x.cls + "," +
x.opcode + ")");

```

```

332         }
333
334     }
335     intermed_code.add("(DL,2)");
336     intermed_code.add("
(C, "+literalTable.get(i).x.substring(2,3)+"");
337
338
339     }
340     poolTable.add(0); //ideally wherever i begins for
literal table;
341     def_printLiteralTable();
342     def_printPoolTable();
343 }
344 if(splitted[0].equals("END"))
345 {
346     System.out.println("gggggggggg");
347     Operator x = null;
348     Iterator<Operator> hh = aList.iterator();
349     while (hh.hasNext()) {
350         x = hh.next();
351         if (x.name.equals(splitted[0])) {
352             // System.out.println(x.name+" "+x.cls+"
"+x.opcode);
353             intermed_code.add("(" + x.cls + ", " +
x.opcode + ")");
354         }
355     }
356 }
357 }
358
359
360
361     }
362
363     }
364     def_printIntermediateCode();
365     def_printLiteralTable();
366     def_printSymbolTable();
367     def_printPoolTable();
368     line_number++;
369
370 }
371 // def_printsymbolTable();
372
373 br.close();
374
375 }
376
377 }
378 /*
379 START 200
380 MOVER AREG, ='5'
381 MOVEM AREG, X
382 L1 MOVER BREG, ='2'
383 ORIGIN L1 +3
384 LTORG
385 X DS 1
386 END
387

```

```

388 #####
389
390 [START, 200]
391 (AD,1)
392 (C,200)
393 -----LITERAL TABLE-----
394 -----
395 -----SYMBOL TABLE-----
396 -----
397 -----POOL TABLE-----
398 -----
399 #####
400
401 201: [MOVER, AREG,, ='5']
402 splitted[2]:='5'
403 -----LITERAL TABLE-----
404 | [[='5',Address not given]]
405 -----
406 (AD,1)
407 (C,200)
408 (IS,4)
409 (RG,1)
410 (L,0)
411 -----LITERAL TABLE-----
412 | [[='5',Address not given]]
413 -----
414 -----SYMBOL TABLE-----
415 -----
416 -----POOL TABLE-----
417 -----
418 #####
419
420 202: [MOVEM, AREG,, X]
421 splitted[2]:X
422 -----SYMBOL TABLE-----
423 | [[X,Address not given]]
424 -----
425 (AD,1)
426 (C,200)
427 (IS,4)
428 (RG,1)
429 (L,0)
430 (IS,5)
431 (RG,1)
432 (S,0)
433 -----LITERAL TABLE-----
434 | [[='5',Address not given]]
435 -----
436 -----SYMBOL TABLE-----
437 | [[X,Address not given]]
438 -----
439 -----POOL TABLE-----
440 -----
441 #####
442
443 203: [L1, MOVER, BREG,, ='2']
444 -----LITERAL TABLE-----
445 | [[='5',Address not given]]
446 | [[='2',Address not given]]
447 -----

```

```

448 -----SYMBOL TABLE-----
449 | [[X,Address not given]]
450 | [[L1,202]]
451 -----
452 (AD,1)
453 (C,200)
454 (IS,4)
455 (RG,1)
456 (L,0)
457 (IS,5)
458 (RG,1)
459 (S,0)
460 (S,1)
461 (IS,4)
462 (RG,2)
463 (L,1)
464 -----LITERAL TABLE-----
465 | [[='5',Address not given]]
466 | [[='2',Address not given]]
467 -----
468 -----SYMBOL TABLE-----
469 | [[X,Address not given]]
470 | [[L1,202]]
471 -----
472 -----POOL TABLE-----
473 -----
474 #####
475
476 204: [ORIGIN, L1, +3]
477 splitted[2]:+3
478 (AD,1)
479 (C,200)
480 (IS,4)
481 (RG,1)
482 (L,0)
483 (IS,5)
484 (RG,1)
485 (S,0)
486 (S,1)
487 (IS,4)
488 (RG,2)
489 (L,1)
490 (AD,3)
491 (C,205)
492 -----LITERAL TABLE-----
493 | [[='5',Address not given]]
494 | [[='2',Address not given]]
495 -----
496 -----SYMBOL TABLE-----
497 | [[X,Address not given]]
498 | [[L1,202]]
499 -----
500 -----POOL TABLE-----
501 -----
502 #####
503
504 206: [LTORG]
505 -----LITERAL TABLE-----
506 | [[='5',205]]
507 | [[='2',206]]

```

```

508 -----
509 -----POOL TABLE-----
510 | 0 |
511 -----
512 (AD,1)
513 (C,200)
514 (IS,4)
515 (RG,1)
516 (L,0)
517 (IS,5)
518 (RG,1)
519 (S,0)
520 (S,1)
521 (IS,4)
522 (RG,2)
523 (L,1)
524 (AD,3)
525 (C,205)
526 (AD,5)
527 (DL,2)
528 (C,5)
529 (AD,5)
530 (DL,2)
531 (C,2)
532 -----LITERAL TABLE-----
533 | [[='5',205]]
534 | [[='2',206]]
535 -----
536 -----SYMBOL TABLE-----
537 | [[X,Address not given]]
538 | [[L1,202]]
539 -----
540 -----POOL TABLE-----
541 | 0 |
542 -----
543 #####
544
545 207: [X, DS, 1]
546 splitted[2]:1
547 (AD,1)
548 (C,200)
549 (IS,4)
550 (RG,1)
551 (L,0)
552 (IS,5)
553 (RG,1)
554 (S,0)
555 (S,1)
556 (IS,4)
557 (RG,2)
558 (L,1)
559 (AD,3)
560 (C,205)
561 (AD,5)
562 (DL,2)
563 (C,5)
564 (AD,5)
565 (DL,2)
566 (C,2)
567 (S,0)

```



```
568 (DL,1)
569 (C,1)
570 -----LITERAL TABLE-----
571 | [[='5',205]]
572 | [[='2',206]]
573 -----
574 -----SYMBOL TABLE-----
575 | [[X,207]]
576 | [[L1,202]]
577 -----
578 -----POOL TABLE-----
579 | 0 |
580 -----
581 #####
582
583 208: [END]
584 ggggggggggg
585 (AD,1)
586 (C,200)
587 (IS,4)
588 (RG,1)
589 (L,0)
590 (IS,5)
591 (RG,1)
592 (S,0)
593 (S,1)
594 (IS,4)
595 (RG,2)
596 (L,1)
597 (AD,3)
598 (C,205)
599 (AD,5)
600 (DL,2)
601 (C,5)
602 (AD,5)
603 (DL,2)
604 (C,2)
605 (S,0)
606 (DL,1)
607 (C,1)
608 (AD,2)
609 -----LITERAL TABLE-----
610 | [[='5',205]]
611 | [[='2',206]]
612 -----
613 -----SYMBOL TABLE-----
614 | [[X,207]]
615 | [[L1,202]]
616 -----
617 -----POOL TABLE-----
618 | 0 |
619 -----
620 */
```