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

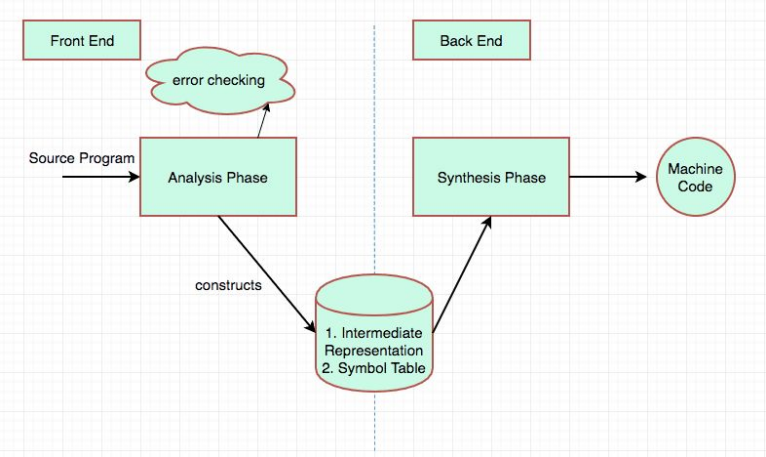
2. Generate data for literals and look for values of symbols

Design specification of an Assember

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, loccntr)* in SYMTAB.

(b) If a START or ORIGIN statement then
loccntr: = value specified in operand field;
(c) If an EQU statement then
(i) this-addr:= value of <*address spec*>;
(ii) Correct the symtab entry for *this-label* to *(this-label, this-addr).*
(d) If a declaration statement then
(i) *code:* = code of the declaration statement;
(ii) *size:* = size of memory area required by DC/DS.
(iii) *loccntr: = loccntr + size;*
(iv) Generate IC '(DL, *code).*
(f) If an imperative statement then
(i) *code:* = machine opcode from OPTAB;
(ii) *loccntr: = loccntr* + instruction length from OPTAB;
(iii) If operand is a symbol then
this-entry: = 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].lengt
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("ggggggggg");
                        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]
ggggggggg
(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 |
--------------------------
*/
```