School of CET

System Software and Compiler lab

Assignment No.3

TY BTech CSE

Assignment Title: Design of Pass 1 of Two Pass Macroprocessor.

Aim: Design suitable data structure & implement pass 1 of Two Pass Macroprocessor.

Objective: Design suitable data structure & implement pass 1 of Two Pass Macroprocessor. Input should consist of a one macro definition and one macro call and few assembly language instructions.

Theory:

Write about

1. Description about the macroprocessor.
2. Data structures required for 2 pass macroprocessor.
3. Flowchart for Pass I.

Macroprocessor

- Macro represents a group of commonly used statements in the source programming language.
- Macro Processor replace each macro instruction with the corresponding group of source language statements. This is known as expansion of macros.
- Using Macro instructions programmer can leave the mechanical details to be handled by the macro processor.
- Macro Processor designs are not directly related to the computer architecture on which it runs.

- Macro Processor involves definition, invocation and expansion.

Forward reference Problem

The assembler specifies that the macro definition should occur anywhere in the program .

So there can be chances of macro call before it's definition witch gives rise to the forwards reference problem

Due to which macro is divided into two passes

1. PASS 1-Recognize macro definition, save macro definition
1. PASS 2-Recognize macro call perform macro expansion

Databases required for pass 2

In pass2 we perform recognize macro call and perform macro expansion

1.COPY FILE

It is a file it contains the output given from PASS1

2.MNT

It is used for recognizing macro name

3.MDT

It is used to perform macro EXPANSION

4.MDTP

It is used to point to the index of MDT .

The starting index is given by MNT

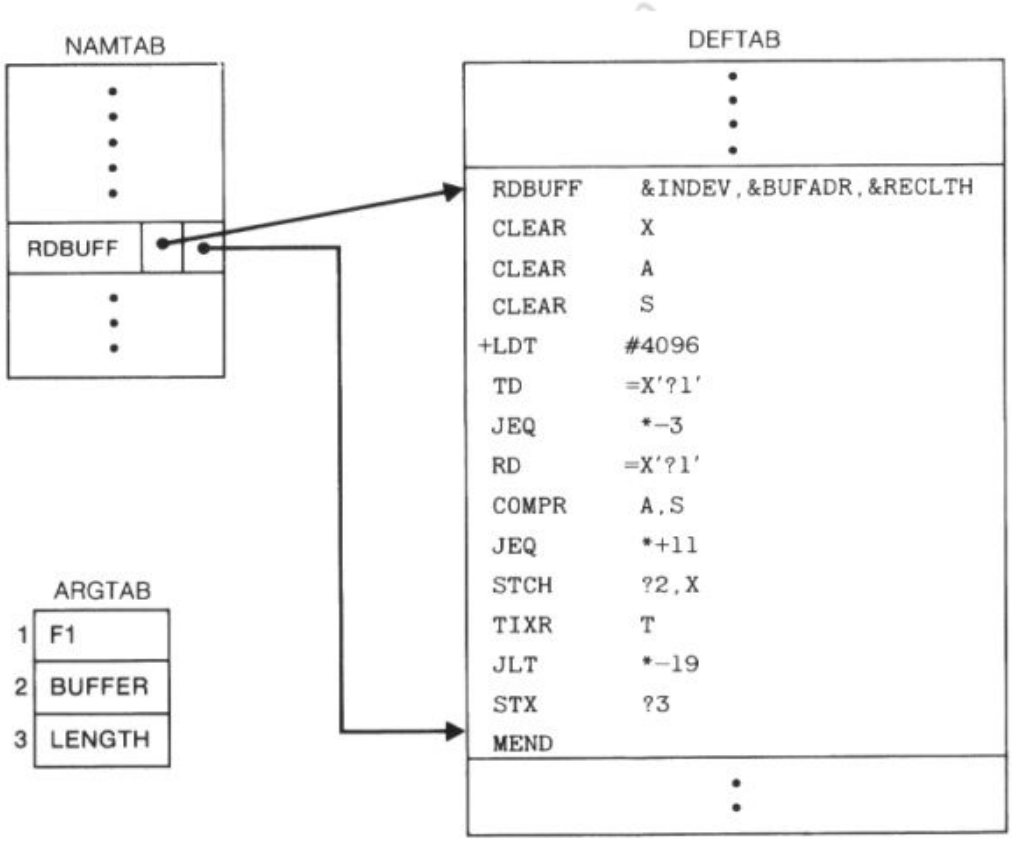
5.ALA

It is used to replace the index notation by it actual value

6.ESC

It is used to contain the expanded macro call which is given to the assembler for further processing

Data structures for 2 pass macroprocessor



Data structures required for macro definition processing –

- Macro Name Table [MNT] – Fields-Name of Macro, #pp (no of positional parameters), # kp(no of keyword parameters), , MDTP (Macro Definition Table Pointer), Keyword Parameters Default Table Position (KPDTP),
- Parameter Name Table [PNTAB] –Fields – Parameter Name
- Keyword parameter Default Table [KPDTAB] –Fields – Parameter Name, Default value
- Macro Definition Table [MDT] –Model Statement are stored in the intermediate code from as: Opcode, Operands.

Algorithm for Pass 1:

Step 1: /* Initialization of counters for MDT and MNT * /

Step 2: Read Next Instruction (and divide it into it's various field as label,
mnemonic (opcode arguments).

Step 3: / * Check for macro definition start * /
if opcode = MACRO goto Step 5
else /* this is not macro definition * /
go to step 4.

Step 4: (a) Write copy of instruction to output of Pass-I
(b) Check whether opcode = END or not
(c) if OPCODE "* END goto Step 2
(d) if OPCODE = END goto Pass-2 i.e. End of this algorithm for Pass- I.

Step 5: /* Start of macro definition is identified. Now Pass-I will process
contents of macro definition after pseudo op MACRO to MEND * /
(a) Read Next Instruction.
(*definitely this is macro name instruction therefore as a processing of this instruction an entry will be made in MNT, ALA will be prepared for this macro, this macro name instruction will be entered in MDT * /
(b) Enter <macro-name, MDTC> into MNT at MNTC
/* current available rows in MDT and MNT are MDTC and MNTC, so macro name and it's starting MDT index i.e. current value of

MDTC is entered in MNT at available row i.e. MNTC * /

(c) MNTC ~ MNTC + 1 /* To point next available row in MNT * /

(d) Prepare Argument List Array

/* ALA is partially constructed by Pass-I to assign universal integer index to dummy arguments * /

(e) Enter macroname instruction in MDT at MDTC.

(1) MDTC ~ MDTC + 1.

/* In step 5, macro name instruction (instruction just after MACRO pseudo op in macro definition and on this instruction name of the macro and corresponding dummy arguments are specified) is processed * /

Step 6: /* Process other instructions in macro definition inducing MEND Instruction * /

(a) Read next card

(b) Substitute Index notations for dummy-arguments.

(c) Enter this instruction (where dummy arguments are replaced by integer indices) into MDT.

(d) MDTC ~ MDTC + 1

(e) if OPCODE of this instruction is MEND then goto Step 2.
else goto Step 6 a.

The data structures associated with Macro Processor:

Input: Assembly Language Program.

Output:

1. Program without Macro Definition (Pass-I)

2. Macro Definition Table (MDT)

Index	MDT- Instruction

3. Macro Name Table (MNT)

Index	Macro Name	MDT- Index

4. Argument List Array (ALA).

Index	Dummy Argument

Conclusion: The function of Pass 1 in a Macro Processor studied.

Platform: Linux (Java)

Conclusion: The function of Pass 1 in assembler is studied along with errors coming in each pass.

Platform: Linux (JAVA)

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

class MntTuple {
  String name;
  int index;

  MntTuple(String s, int i) {
    name = s;
    index = i;
  }

  public String toString() {
    return("[" + name + ", " + index + "]");
  }
}

class MacroProcessor{
  static List<MntTuple> mnt;
  static List<String> mdt;
  static int mntc;
  static int mdtc;
  static int mdtp;
  static BufferedReader input;
  static List<List <String>> ala;
  static Map<String, Integer> ala_macro_binding;

  public static void main(String args[]) throws Exception {
    initializeTables();
    System.out.println("===== PASS 1 =====\n");
    pass1();
  }

  static void pass1() throws Exception {
    String s = new String();
    input = new BufferedReader(new InputStreamReader(new
FileInputStream("input.txt")));
    PrintWriter output = new PrintWriter(new
FileOutputStream("output_pass1.txt"), true);
    while((s = input.readLine()) != null) {
      if(s.equalsIgnoreCase("MACRO")) {
        processMacroDefinition();
      } else {
        output.println(s);
      }
    }
    System.out.println("ALA:");
    showAla(1);
    System.out.println("\nMNT:");
    showMnt();
    System.out.println("\nMDT:");
    showMdt();
  }

  static void processMacroDefinition() throws Exception {
    String s = input.readLine();
    String macro_name = s.substring(0, s.indexOf(" "));
    mnt.add(new MntTuple(macro_name, mdtc));
    mntc++;
    pass1Ala(s);
```

```
    StringTokenizer st = new StringTokenizer(s, " ,", false);
    String x = st.nextToken();
    for(int i=x.length() ; i<12 ; i++) {
      x += " ";
    }
    String token = new String();
    int index;
    token = st.nextToken();
    x += token;
    while(st.hasMoreTokens()) {
      token = st.nextToken();
      x += "," + token;
    }
    mdt.add(x);
    mdtc++;
    addIntoMdt(ala.size()-1);
  }

  static void pass1Ala(String s) {
    StringTokenizer st = new StringTokenizer(s, " ,", false);
    String macro_name = st.nextToken();
    List<String> l = new ArrayList<>();
    int index;
    while(st.hasMoreTokens()) {
      String x = st.nextToken();
      if((index = x.indexOf("=")) != -1) {
        x = x.substring(0, index);
      }
      l.add(x);
    }
    ala.add(l);
    ala_macro_binding.put(macro_name, ala_macro_binding.size());
  }

  static void addIntoMdt(int ala_number) throws Exception {
    String temp = new String();
    String s = new String();
    List l = ala.get(ala_number);
    boolean isFirst;
    while(!s.equalsIgnoreCase("MEND")) {
      isFirst = true;
      s = input.readLine();
      String line = new String();
      StringTokenizer st = new StringTokenizer(s, " ,", false);
      temp = st.nextToken();
      for(int i=temp.length() ; i<12 ; i++) {
        temp += " ";
      }
      line += temp;
      while(st.hasMoreTokens()) {
        temp = st.nextToken();
        if(temp.startsWith("&")) {
          int x = l.indexOf(temp);
          temp = ",#" + x;
          isFirst = false;
        } else if(!isFirst) {
          temp = "," + temp;
        }
        line += temp;
      }
```

```
      mdt.add(line);
      mdtc++;
    }
  }

  static void showAla(int pass) throws Exception {
    PrintWriter out = new PrintWriter(new FileOutputStream("out_ala_pass" + pass + ".txt"), true);
    for(List l : ala) {
      System.out.println(l);
      out.println(l);
    }
  }

  static void showMnt() throws Exception {
    PrintWriter out = new PrintWriter(new FileOutputStream("out_mnt.txt"), true);
    for(MntTuple l : mnt) {
      System.out.println(l);
      out.println(l);
    }
  }

  static void showMdt() throws Exception {
    PrintWriter out = new PrintWriter(new FileOutputStream("out_mdt.txt"), true);
    for(String l : mdt) {
      System.out.println(l);
      out.println(l);
    }
  }
  static void initializeTables() {
    mnt = new LinkedList<>();
    mdt = new ArrayList<>();
    ala = new LinkedList<>();
    mntc = 0;
    mdtc = 0;
    ala_macro_binding = new HashMap<>();
  }
}

```