# EXPERIMENT NO : 02

1. **Title: Design suitable data structures and implement Pass-I and Pass-II of a two-pass macroprocessor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II**

## PART – 1

**1.** Title:

Design suitable data structures and implement pass-I of a two-pass macro-processor using OOPfeatures in Java

**2.** Objectives :

- To understand Data structure of Pass-1 macroprocessor
- To understand Pass-1 macroprocessor concept
- To understand macro facility.

**3.** Problem Statement :

Design suitable data structures and implement pass-I of a two-pass macro-processor using OOPfeatures in Java.

**4.** Outcomes:

After completion of this assignment students will be able to:

- Implemented Pass – 1 macroprocessor
- Implemented MNT, MDT table.
- Understood concept Pass-1 macroprocessor.

**5.** Software Requirements:

Latest jdk., Eclipse

**6.** Hardware Requirement:

- M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD

**7.**

7. Theory Concepts:

**Macroprocessor**

Macro pre-processor takes a source program containing macro definitions and macro calls and translates into an assembly language program without any macro definitions or calls. This program can now be handed over to a conventional assembler to obtain the target language (as shown in Fig. 2.5.1).


(S3.6)**Fig. 2.5.1 : A scheme for a macro pre-processor**

**Macro** :

Macro allows a sequence of source language code to be defined once & then referred to by name each time it is referred. Each time this name occurs in a program , the sequence of codes is substituted at that point.

Macro has following parts:-

(1) Name of macro

(2) Parameters in macro

(3) M

acro Definition

Parameters are

optional.

How To Define a Macro :-

Macro can be formatted in following order :-

| Start of definition | MACRO |
|---|---|
| macro name | mymacro |
| macro body | ADD AREG, X<br>ADD BREG, X |
| End of macro definition | MEND |

'MACRO' pseudo-op is the first line of definition & identifies the following line as macroinstruction name.

Following the name line is sequence of instructions being abbreviated the instructions

comprising the 'MACRO' instruction.

The definition is terminated by a line with MEND pseudo-op.

### Example of Macro:-

1. Macro without parameters

```
MACRO
        Mymacro
        ADD AREG,X
        ADD BREG,X
        MEND
```

Macro with parameters

```
MACRO
    addmacro  &A
    ADD AREG,&A
    ADD BREG,&A
    MEND
```

The macro parameters (Formal Parameters) are initialized with '&' . used as it is in

operation..Formal Parameters are those which are in definition of macro.

Whereas while calling macros we use Actual Parameters.

### How To Call a Macro?

A macro is called by writing macro name with actual parameters in an AssemblyProgram.

Macro call leads to Macro Expansion.

Syntax: <macro-name> [<list of parameters>]

Example:- for above definitions of macro…

(1) mymcro

(2) addmacro X

## Macro Expansion:-

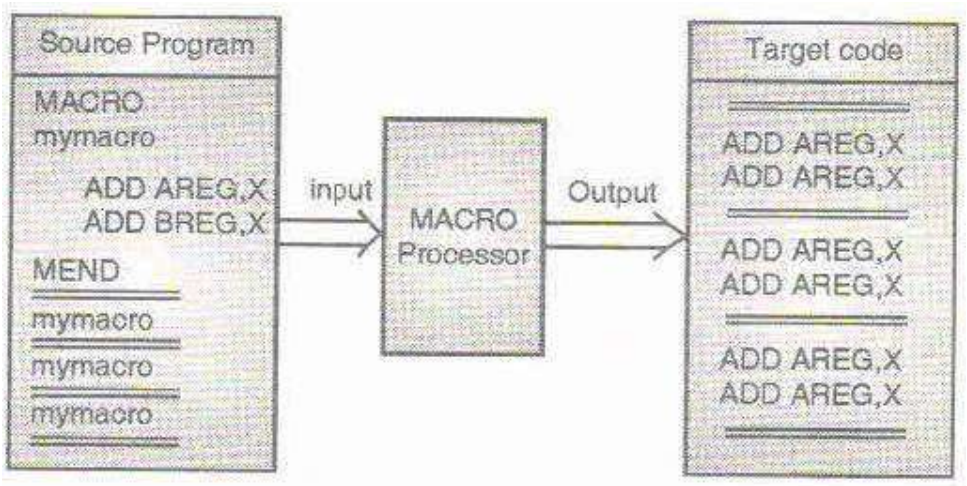
Each Call to macro is replaced by its body.

During Replacement, actual parameter is used in place of formal parameter.

- During Macro expansion, each statement forming the body of macro is picked up one byeone sequentially.
- Each Statement inside the macro may have:

  (1) An ordinary string, which is copied as it is during expansion.

  (2) The name of a formal parameter which is proceeded by character '&'.
- During macro expansion an ordinary string is retained without any modification. FormalParameters(Strings starting with &) is replaced by the actual parameter value.




(S3.2)**Fig. 2.1.2 : Macro expansion**

## Macro with Keyword Parameters :-

These are the methods to call macros with formal parameters. These formal parameters are oftwo types

**(1)** Positional Parameters :

Initiated with '&'.

Ex:- mymacro &X

**(2)** Keyword Parameters :

Initiated with '&' . but has some default value.

During a call to macro , a keyword parameter is specified by its name.Ex:- mymacro &X=A

## Nested Macro Calls :-

Nested Macro Calls are just like nested function calls in our normal calls. Only the transfer ofcontrol from one macro to other Is done.

Consider this example :-

```
MACRO
   Innermacro
        ADD AREG,X
MEND

MACRO
   outermacro
        innermacro
        ADD AREG,Y
MEND

outermacro
```

In this example, firstly the MACRO outermacro gets executed & then innermacro.

So Output will be Adding X & Y values in AREG register.

## Algorithm:

Scan all MACRO definition one by one.

(a) Enter its name in macro name table (MNT).

(b) Store the entire macro definition in the macro definition table (MDT).

(c) Add the information in the MNT indicates where definition of macro can be found inMDT.

(d) Prepare argument list array (ALA).

## Data Structures of Two Pass Macros:

1] Macro Name Table Pointer (MNTP) :

2] Macro Definition Table Pointer (MDTP) :

3] Macro Name Table :

- macro number(i.e pointer referenced from MNTP)
- Name of macros
- MDTP (i.e points to

start position to MDT)4]

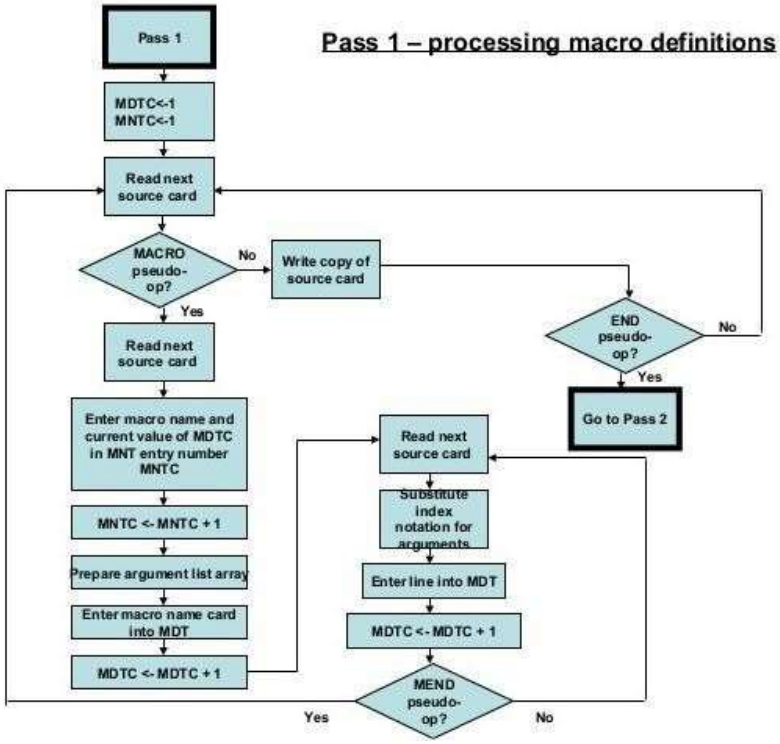
Macro Definition Table :

- Location Counter(where MDTP points to start position of macro)
- Opcode
- Rest (i.e it will contain the other part than

opcodes used in macro).5] Argument List Array :

- Index given to parameter
- Name of parameter

**Flowchart : .**




**PASS 1 FLOWCHART**

## 8. Conclusion :

Thus , I have implemented Pass-1 macroprocessor by producing MNT and MDT table.

## PART – 2

**1.** Title:

Write a Java program for pass-II of a two-pass macro-processor. The output of assignment-3 (MNT,MDT and file without any macro definitions) should be input for this assignment

**2.** Objectives :

- To understand Data structure Pass-2 macroprocessor
- To understand Pass-1 & Pass-2 macroprocessor concept
- To understand Advanced macro facility

**3.** Problem Statement :

Write a Java program for pass-II of a two-pass macro-processor. The output of assignment-3 (MNT,MDT and file without any macro definitions) should be input for this assignment

**4.** Outcomes:

After completion of this assignment students will be able to:

- Implemented Pass – 2 macroprocessor
- Implemented machine code from MDT and MNT table.
- Understood concept Pass-2 macroprocessor.

**5.** Software Requirements:

Latest jdk., Eclipse

**6.** Hardware Requirement:

- M/C Lenovo Think center M700 Ci3,6100,6th Gen. H81, 4GB RAM ,500GB HDD

**7.** Theory

Concepts:

Advanced

Macro

Facilities:

(1) AIF

(2) AGO

(3) Sequential Symbol

(4) Expansion time variable

### (1) AIF

Use the AIF instruction to branch according to the results of a condition test. You can thus alter the sequence in which source program statements or macro definition statements are processed by the assembler.

The AIF instruction also provides loop control for conditional assembly processing, which lets you control the sequence of statements to be generated.

It also lets you check for error conditions and thereby to branch to the appropriate MNOTE instruction to issue an error message.

### (2) AGO

The AGO instruction branches unconditionally. You can thus alter the sequence in which your assembler language statements are processed. This provides you with final exits from conditional assembly loops.

### 3) Sequence Symbols

You can use a sequence symbol in the name field of a statement to branch to that statement during conditional assembly processing, thus altering the sequence in which the assembler processes your conditional assembly and macro instructions. You can select the model statements from which the assembler generates assembler language statements for processing at assembly time.

A sequence symbol consists of a period (.) followed by an alphabetic character, followed by 0 to 61 alphanumeric characters.

Examples:

.BRANCHING_LABEL#1

.A

Sequence symbols can be specified in the name field of assembler language statements and model statements; however, sequence symbols must not be used as name entries in the following assembler instructions:

| | | | |
|---|---|---|---|
| ALIAS | EQU | OPSYN | SETC |
| AREAD | ICTL | SETA | SETAF |
| CATTR | LOCTR | SETB | SETCF |
| DXD | | | |

Also, sequence symbols cannot be used as name entries in macro prototype instructions, or in anyinstruction that already contains an ordinary or a variable symbol in the name field.

Sequence symbols can be specified in the operand field of an AIF or AGO instruction to branch to astatement with the same sequence symbol as a label

**4)** Expansion Time Variables

**Note :- write theory from book or notes.**

- Data Structures of Two Pass Macros:

1] Input Source Program for pass- II . It is
produced by pass – I .2] Macro Definition
Table : (MDT) produced by pass - I

- Location Counter(where MDTP points to start position of macro)
- Opcode
- Rest (i.e it will contain the other part than

opcodes used in macro).3] Macro Name Table :
(MNT) produced by pass - I

- macro number(i.e pointer referenced from MNTP)
- Name of macros
- MDTP (i.e points to start position to MDT)
-

4] MNTP (macro name table pointer) gives

number of entries in MNT.5] Argument List

Array :

- Index given to parameter
- Name of parameter

  Which gives association between integer indices & actual parameters.

6] Source Program with macro-calls expanded. This is output of pass- II.

7] MDTP (macro definition table pointer) gives the address of macro definition in macro definitiontable.

## Algorithm:

Take Input from Pass - I

Examine all statements in the assembly source program to detect macro calls. For EachMacro call:

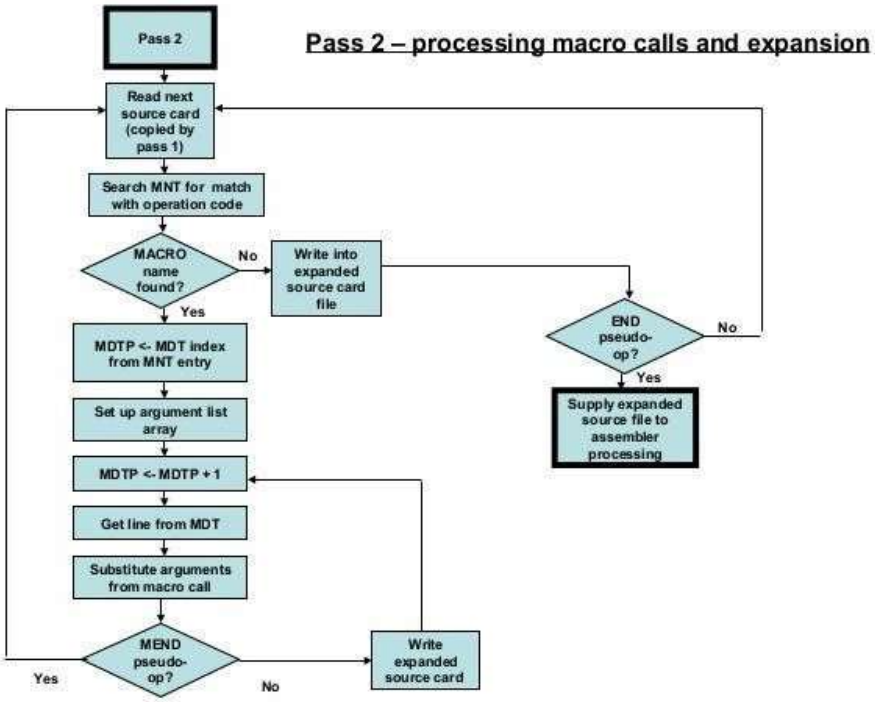
(a) Locate the macro name in MNT.

(b) Establish correspondence between formal parameters & actual parameters.

(c) Obtain information from MNT regarding position of the macro definition in MDT.

(d) Expand the macro call by picking up model statements from MDT.

**Flowchart : .**




**PASS 2 FLOWCHART**

## 8. Conclusion :

Thus , I have implemented Pass-2 macroprocessor by taking input as output of assignment-3 (i.e.MDT and MNT table)