

# ASSIGNMENT 6

\* Title : Macro Processor.

\* Problem Statement : Study assignment for macro processor

\* Objective :

- i) To study basic functioning of Macro Processor.
- ii) To study working of nested macro.

\* Theory :

• What is macro processor :

A general-purpose macro processor is a macro processor that is not tied to or integrated with a particular language or piece of software.

A macro processor is a program that copies of stream of text from one place to another, making a systematic set of replacements as it does so.

Macro definition

The #define directive specifies a macro identifier and a replacement list and terminates with a new-line character. The replacement list, a sequence of

preprocessing tokens is substituted for every subsequent occurrence of that macro identifier.

Syntax: #define identifier replacement-list  
or  
#define identifier (identifier-list (opt))  
replacement-list newline.

#### • Macro call:

A macro call consists of a name optionally followed by an actual-parameter list.

If the macro has no formal parameter list, its call must have no actual parameter list.

#### \* Parameters-passing

A parameter can be either a simple string or a quoted string. It can be passed by using the standard method of putting variables into char and profile pools.

You can enter a parameters along with an edit macro name as primary command by using the MACRO command.

It is an error to enter parameter values for a macro without parameter variables. If you make this mistake, the editor display an error message.



## Nested Macro definition

Nested macro definition are those cases where the definition of one macro is contained within the def<sup>n</sup> of another.

Example:

```

MACRO
    MAC-X    &par1, &par2
    MOVER    REG1, REG2 &par1
    - MACRO
        MAC-Y    &par2, REG = REG-3
        ADD.    REG, &par2
    MEND
PRINT &par1
MEND
    
```

If a macro call is seen throughout the expansion of a macro, the assembler starts immediately with the expansion of the called macro.

For this, its expanded body lines are simply inserted into the expanded macro body of the calling macro, until the called macro is completely expanded. Then the expansion of the calling macro is continued with the body line following the nested macro call.

Example :

```

INSIDE MACRO
    SUBB A, R3
ENDM
    
```

OUTSIDE MACRO

MOV A, #42

INSIDE

MOV R7, A

ENDM

In the body of macro OUTSIDE, the macro INSIDE is called. If OUTSIDE is called (and the list mode is set to \$GENONLY), one gets something like the following expansion

Line I	Addr.	Code	Source
15 + 1	0000	74 2A	MOV A, #42
17 + 2	0002	9B	SUBB A, R3
18 + 1	0003	FF	MOV R7, A

Since macro calls can be nested to any depth (while there is free memory), the macro expansion level is shown in the I-column of the list file.

#### \* Parameter passing methods in macro processor

- i) Positional parameters
- ii) Keyword parameters
- iii) Specifying default values of parameter
- iv) Macros with mixed parameter list



## A) Positional Parameters

A positional formal parameter is written as  $\langle \text{parameter name} \rangle$ . e.g.  $\&\text{SAMPLE}$  SAMPLE is the name of a parameter. In a call on a macro using positional parameters the  $\langle \text{actual parameter specification} \rangle$  is an ordinary string.

## B) Keyword Parameters

For keyword parameters, the specification  $\langle \text{parameter kind} \rangle$  is the string  $" = "$ . The  $\langle \text{actual param specification} \rangle$  is written as ~~as~~  $\langle \text{formal parameter name} \rangle = \langle \text{ordinary string} \rangle$ .

## C) Specifying default values of parameter

If a parameter has the same value in most calls on a macro, this value can be specified as its default value in the macro definition itself.

If a macro call does not explicitly specify the value of the parameters, the preprocessor uses its default value; otherwise ~~and~~ uses the value specified in the macro call.

The extended syntax is:

$\&\langle \text{param name} \rangle [ \langle \text{param kind} \rangle [ \langle \text{default value} \rangle ] ]$

## d) Macro with mixed parameter lists:

A macro definition may ~~use~~ use both positional

and keyword parameters. In such a case, all positional params must precede all keyword params in a macro call.

## \* Data structures for Macro processor

### i) Pass-I data structures :

- The input macro source deck.
- The output macro source deck copy for use by pass 2.
- The Macro Definition Table (MDT), used to store the body of the macro definitions.
- The Macro Name Table (MNT), used to store the names of defined macros.
- The Macro Definition Table Counter (MDTC), used to indicate the next available entry in the MDT.
- The Macro Name Table Counter (MNTC) used to indicate the next available entry in the MNT.
- Argument List Array (ALA) used to substitute index markers for dummy arguments before storing a macro definition.

### ii) Pass-2 data structures :

- Copy of the input macro source deck
- The o/p expanded source code to be used as input to assembler.
- MDT by pass 1



- MNT by pass 1
- MOTE used to indicate the next line of text to be used during macro expansion.
- ALA used to substitute macro call arguments for the index markers in the stored macro definition.

#### \* Pass structure of macro processor

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

So, there can be chances of macro call before its definition which gives rise to the forward reference problems of macros.,  
due to which macro is divided into 2 passes:-

i) Pass-1 : Recognize macro definition, save macro definition.

ii) Pass-2 : Recognize macro call, perform macro expansion.

# \* EXAMPLE 1

1. READ A
2. READ B
3. MACRO
4. ADD2 X, Y, Z
5. LOAD X
6. ADD Y
7. STORE Z
8. MEND
9. MACRO
10. MULT L, X, Y, Z
11. LOAD @0
12. STORE Z
13. L ADD2 X, Z, Z
14. LOAD Y
15. SUB @1
16. STORE Y
17. JPOSE L
18. MEND.
19. MULT L1, A, B, C
20. ENDP
21. A DEFW
22. B DEFW
22. C DEFW
24. END



## MDT

LOAD (P, #1)

ADD (P, #2)

STORE (P, #3)

MEND

LOAD @0

STORE (P, #4)

(R, #1) ADD2 (P, #2) (P, #4) (P, #4)

LOAD (P, #3)

SUB @1

STORE (P, #3)

JPOSE (P, #1)

MEND

## MNT

Name	#R	#P	#MTOC	#HPP
------	----	----	-------	------

ADD2	0	3	1	0
------	---	---	---	---

MULT	0	4	5	0
------	---	---	---	---

ALA1

#1	X
----	---

#2	Y
----	---

#3	Z
----	---

ALA2

#1	L
----	---

#2	X
----	---

#3	Y
----	---

#4	Z
----	---

## Expanded Code

+ LOAD @0

+ STORE C

+ LOAD A

+ ADD C

+ STORE C

+ LOAD B

+ SUB @1

+ STORE B

+ JPOSE L1

ENDP

END

## Example 2

START

SR 2, 2

L 1, DATA1

MACRO

ADD -M &ARG1

L 1, &ARG1

A 1, =F '10'

SR 3, 3

ST 1, &ARG1

MEND

AR 2, 2

MACRO

ADD -S &A1, &A2, &A3

ADD -M &A1

ADD -M &A2

ADD -M &A3

MEND

ADD -M DATA1

ADD -S X1, X2, X3

ADD -S X2, X1, X3

DATA1 DC, F '20'

X1 DC F '25'

X2 DC F '30'

X3 DC F '35'



### MDT

1. L 1, (P, #1)  
2. A 1, =F'10'  
3. SR 3, 3  
4. ST 1, (P, #1)  
5. MEND  
6. ADD-M (P, #1)  
7. ADD-M (P, #2)  
8. ADD-M (P, #3)  
9. MEND

### MNT

Name	HR	#P	#MIDC	#KPTAB
ADD-M	0	1	1	0
ADD-S	0	3	6	0

### ALA1

#1 & ARG1

### ALA2

#1 & A1  
#2 & A2  
#3 & A3

### Expansion:-

+ L, 1, DATA1

+ A 1, =F'10'

+ SR 3, 3

+ ST 1, DATA1

+ L 1, X1

+ A, 1, =F'10'

+ SR 3, 3

+ ST 1, X1

+ L, 1, X1

+ L, 1, X2

+ A, 1, =F'10'

+ SR 3, 3

+ ST 1, X2

+ L, 1, X3

+ A, 1, =F'10'

+ SR 3, 3

+ L, 1, X2

+ A, 1, =F'10'

+ SR 3, 3

+ ST, 1, X2

+ L, 1, X1

+ A, 1, =F'10'

+ SR, 3, 3

+ ST, 1, X1

+ L, 1, X3

+ A, 1, =F'10'

+ SR, 3, 3

+ ST 1, X3

### Example 3

MAIRO

MACI

MOVER AREG, M

ADD AREG, M

MOVEM CREG, M

MEND

MACRO

EVAL &X, &Y, &Z

MOVER AREG, &X

SUB AREG, &Y

ADD AREG, &Z

MEND

MACRO

CALC &X, &Y, &OP=MULT, &LAB

&LAB MOVER AREG, &X

&OP AREG, &Y

MOVEM AREG, &X

MEND.

START

MOVEM AREG, B

EVAL A, B, C

ADD AREG, N

MOVEM AREG, N

CALC P, Q, LAB=LOOP

MOVEM AREG, N

MACI

CALC P, Q, OP=DIV, LAB=NEXT

M DS 1

A DS 5

P DS 1 Q DS 1

END.



MDT

- 1) MOVER AREG, M
- 2) ADD BREG, M
- 3) MOVEM CREG, M
- 4) MEND
- 5) MOVER AREG, (P, #1)
- 6) SUB AREG, (P, #2)
- 7) ADD AREG, (P, #3)
- 8) MOVER AREG, (P, #3)
- 9) MEND
- 10) (P, #4) MOVER AREG (P, #1)
- 11) (P, #3) AREG (P, #2)
- 12) MOVEM AREG (P, #1)
- 13) MEND

MNT

Name	HR	#P	#MTC	#KPTAR
MACI	0	0	1	0
EVAL	0	3	5	0
CALC	2	4	10	1

ALA 2ALA 3

#1	8X	#1	2
#2	8Y	#2	Y
#3	8Z	#3	8OP
		#3	8LAB

Expanded:START

- + MOVEM AREG, B
- + MOVER AREG, A
- + ADD AREG, B
- + SUB AREG, C
- + ADD AREG, N
- + MOVER AREG, N
- + MOVER AREG, P
- + MULT AREG, Q
- + MOVEM AREG, P
- + NEXT MOVER AREG, P
- + DIV AREG, Q
- + MOVEM AREG, P

### \* Conclusion:

Hence, macro processor has been thoroughly studied and understood and macro examples have been correctly solved.

~~Q.~~