CH- MACROS & MACRO PROCESSOR

Macros are used to provide a program generation facility through macro expansion.
many languages PL/I, C, Ada & C++ provide built-in facilities for writing macros.

-A macro is a unit of specification for program generation through expansion

A macro consists of a name, a set of formal parameters and a body of code.

The use of macro name with a set of actual parameters is replaced by some code generated from its body. This is called macro expansion.

Two kinds of expansion:-

Lexical expansion:- It implies replacement of a character string by another character string during program generation.
It is typically employed to replace occurences of formal parameters by corresponding actual parameters.

2. Semantic expansion:- It implies generation of instructions tailored to the requirements of a specific usage
ex:- generation of type specific instructio for manipulation of byte and word operands.
- semantic expansion is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence & opcodes o instructions.

- Use of a macro name in the mnemon field of an assembly statement leads to its expansion, whereas use of a subroutine name in a call instructio leads to its execution.

- Programs using macros and subrou differ significantly in terms of progra size and execution efficiency.

⇒ MACRO DEFINITION & CALL.

macro definition.

A macro definition is enclosed between a macro header statemen and a macro end statement.
- macro definitions are typically located at the start of a progra

- A macro consists of :-

1. A macro prototype statement
2. One or more model statements
3. Macro preprocessor statements.

- The macro prototype statement declares the name of a macro and the names and kinds of its parameters.
- A model statement is a statement from which an assembly language statement may be generated during macro expansion.
- A preprocessor statement is used to perform auxilliary functions during macro expansion.

- The macro prototype statement syntax:

<macro name> [<formal parameter spec> [,...]]

- <macro name> = appears in mnemonic field of an assembly st.

- <formal parameter spec> = & <parameter name> [<parameter kind>].

Macro Call

- A macro ~~call~~ is called by writing the macro name in the mnemonic field of a assembly statement.

Syntax:-

<macro name> [<actual parameter spec> [,...]]

where an actual parameter typically resembles an operand specification in an assembly language statement.

macro definition:-

```
MACRO
INCR    &MEM_VAL, &INCR_VAL, &REG
MOVER   &REG, &MEM_VAL
ADD     &REG, &INCR_VAL
MOVEM   &REG, &MEM_VAL
MEND
```

↗ 3 parameter for macro.

model statements.

macro header & macro end.

- Parameter kind is not specified for any of the parameters, they are all of the default kind 'positional parameter'

- No preprocessor statements are used in this macro.

→ MACRO EXPANSION.

- A macro call leads to macro expansion. During macro expansion, the macro call statement is replaced by a

sequence of assembly statements.
— macro statements indicated by '+' preceding its label field.

— Two key notions :-

1. Expansion time control flow :- This determines the order in which model statements are visited during macro expansion.

2. Lexical substitution :- Lexical substitution is used to generate an assembly statement from a model statement.

→ Flow of control during expansion.

— The default flow of control during macro expansion is sequential.
— In the absence of preprocessor statements the model statements of a macro are visited sequentially starting with the statement following the macro prototype statement and ending with the statement preceding the MEND statement.

— A preprocessor statement can alter the flow of control during expansion such that some model statements are either never visited during expansion, or are repeatedly visited during expansion.
— Former results in conditional expansion & latter expansion time loop.

- The flow of control during macro expansion is implemented using a macro expansion counter (MEC)

Algo:- outline of macro expansion.

1. MEC := statement number of 1st statement following the prototype statement
2. While statement pointed by MEC is not MEND.

   (a) If a model statement then
   (i) Expand the statement
   (ii) MEC = MEC + 1;

   (b) Else (i.e. a preprocessor st.)
   (i) MEC = new value specified in the statement → i.e. conditional expansion or expansion time loops
3. Exit from macro expansion.

→ Lexical substitution.

- model statement consists of 3 types of strings.

1. An ordinary string, which stands for itself. (retained without substitution).
2. The name of formal parameter which is preceded by the character '&'
3. The name of preprocessor variable which is also preceded by the

replace by formal parameter or preprocessor variables

character '&'

→ (1) Positional parameters

&<parameter name>

eg:- &SAMPLE where SAMPLE is the name of parameter.

<parameter kind> is omitted.

→ Positional association rules:-

1. Find the ordinal position of XYZ in the list of formal parameters in the macro prototype statement.
2. Find the actual parameters spec. occupying the same ordinal position in the list of actual parameters in the macro call st.

- eg:- ordinary string ABC, so value of formal parameter XYZ is ABC.

Ex:- INCR A, B, AREG.

formal parameter	Value
MEM_VAL	A
INCR_VAL	B
REG	AREG.

lexical expansion of the model statements now leads to the code.

```
+    MOVER    AREG, A
+    ADD      AREG, B
+    MOVEM    AREG, A.
```

→ ② Keyword Parameters:

<parameter name> is an ordinary string
<parameter kind> is the string '='

<actual parameter spec> is written as
<formal parameter name> = <ordinary string>

* Keyword association:-

1. Find the actual parameter specification which has the form XYZ = <ordinary string>.
2. Let <ordinary string> in the specificati be the string ABC. Then the value of formal parameter XYZ is ABC.

Ex:- INCR macro rewritten as macro INCR_M using keyword parameters

```
INCR_M    MEM_VAL=A, INCR_VAL=B,
          REG=AREG
....
INCR_M    INCR_VAL=B, REG=AREG,
          MEM_VAL=A.
```

are equivalent.

```
MACRO
INCR_M  &MEM_VAL=, &INCR_VAL=, &
MOVER   &REG, &MEM_VAL
ADD     &REG, &INCR_VAL
MOVEM   &REG, &MEM_VAL.
MEND
```

- A macro definition using keyword parameters.

(3) Default Parameter (assumptio

- A default is a standard specification in the absence of an explicit spec. b the programmer. Default spec of paramete is useful in situations where a parameter has th same value in most calls.

ex:- &REG = AREG is used for all arithmetic operations.

```
INCR_D  MEM_VAL=A, INCR_VAL=B, ~~&REG~~
INCR_D  INCR_VAL=B, MEM_VAL=A
INCR_D  INCR_VAL=B, MEM_VAL=A, REG=BRE
```

overrides the default values for REG with BREG.

- BREG will be used to perform the arithmetic in its expanded code.

```
MACRO
INCR_D  &MEM_VAL=, &INCR_VAL=, &REG=AREG
MOVER   &REG, &MEM_VAL
ADD     &REG, &~~MEM~~INCR_VAL
MOVEM   &REG, &MEM_VAL.
MEND
```

⇒ Nested Macro Calls :

- A model statement in a macro may constitute a call on another macro. Such calls are known as nested macro calls.
- macro containing nested call as outer macro and the called macro as inner macro.
- expansion of nested macro calls follows last-in-first out (LIFO) rule.
- In a structure of nested macro calls expansion of the latest macro call. (i.e. the innermost macro call in the structure) is completed first.

Ex :-

```
+ MOVEM   BREG, TMP
+ MOVER   BREG, X
+ ADD     BREG, Y
+ MOVEM   BREG, X
+ MOVER   BREG, TMP
```

A nested macro call.

```
MACRO
COMPUTE  &FIRST, &SECOND
MOVEM    BREG, TMP
INCR_D   &FIRST, &SECOND, REG=BREG
MOVER    BREG, TMP
MEND
```

COMPUTE X, Y

+ MOVEM BREG, TMP [1]
+ INCR D X, Y
+ MOVER BREG, TMP [5]

+ MOVEM BREG, X [2]
+ ADD BREG, Y [3]
+ MOVEM BREG, X [4]

→ Advanced Macro Facilities

— It supports semantic expansion.

1. Facilities for alteration of flow of control during expansion.
2. Expansion time variables.
3. Attributes of parameters.

* Alteration of flow of control during expansion.

— Two features are provided to facilitate alteration of flow of control during expansion.

1. Expansion time sequencing symbols.
2. Expansion time statements AIF, AGO and ANOP.

— A sequencing symbol (SS) has the syntax

.<ordinary string>

- An SS is defined by putting it in the label field of a statement in the macro body.
- It is used as an operand in an AIF or AGO statement to designate the destination of an expansion time control transfer.
- It never appears in the expanded form of a model statement.

AIF (<expression>) <sequencing symbol>

relational exp. involving ordinary strings, formal parameters & their attributes & expansion time variables.

If true, expansion time control is transferred to SS in its label field.

AGO <sequencing symbol>

- unconditionally transfers expansion time control to the statement containing <SS> in its label field.

<sequencing symbol> ANOP → It has effect of defining the sequencing symbol.

SMVS
Page No.
Date : / /

→ Expansion time variables.

- Expansion time variables (EV's) are variables which can only be used during the expansion of macro calls.
- A local EV is created for use only during a particular macro call.
- A global EV exists across all macro calls situated in a program and can be used in any macro which has a declaration for it.
- Local & global EV's are created through declaration statements:

```
LCL <EV spec.> [,<EV spec>...]
GBL <EV spec.> [,<EV spec>...]
```

&<EV name> which is ordinary string.

- Values of EV's can be manipulated through the preprocessor statement SET.

```
<EV spec.>  SET  <SET-expression>
```

<EV spec.> appears in label field.
SET in mnemonic field.

- A SET statement assigns the value of <SET-expression> to the EV specified in <EV spec.>. The value of an EV can be used in any field of a model statement and in the expression of an AIF statement

Page No.
Date : / /

Ex:-

```
        MACRO
        CONSTANTS
        LCL     &A
&A      SET     1
        DB      &A
&A      SET     &A+1
        DB      &A
        MEND
```

- Local EV - A is created. SET assigns the value '1' to it. DB declares byte constant '1'. second SET assigns value '2' to A and second DB declares constant '2'.

→ Attributes of formal parameters.

<attribute name>' <formal parameter spec>

- type, length, size attributes have names T, L and S.

Ex:-

```
        MACRO
        DCL_CONST &A
        AIF     (L'&A EQ 1).NEXT
        --
.NEXT   --
        --
        MEND.
```

- expansion time control transferred to .NEXT. in label field. only if actual

parameter corresponding to the formal parameter A has the length of 'I'.

→ Conditional expansion.

- It helps in generating assembly code specifically suited to the parameters in a macro call.
- This is achieved by ensuring a model st. is visited only under specific conditions during the expansion of a macro.
- AIF and AGO sts are used.

ex:- macro EVAL to evaluate A − B + C in AREG.

- when 1st 2 parameters of a call are identical, EVAL should generate single MOVER instruction to load the 3rd parameter into AREG.

```
        MACRO
        EVAL    &X, &Y, &Z
        AIF     (&Y EQ &X) .ONLY
        MOVER   AREG, &X
        SUB     AREG, &Y
        ADD     AREG, &Z
        AGO     .OVER
.ONLY   MOVER   AREG, &Z
.OVER   MEND
```

→ Expansion time loops.

- It is often necessary to generate many similar statements during the expansion of a macro.
- This can be achieved by writing similar model statements in the macro.

```
MACRO
CLEAR  &A          ——————— CLEAR B
MOVER  AREG, = '0'
MOVEM  AREG, &A    ——— B
MOVEM  AREG, &A+1  ——— B+1
MOVEM  AREG, &A+2  ——— B+2
MEND
```

- The same effect can be achieved by writing an expansion time loop which visits a model statement, or a set of model statements, repeatedly during macro expansion.

- Expansion time loops can be written using expansion time variables (EV's) and expansion time control transfer statements AIF and AGO.

Ex:-

```
      MACRO
      CLEAR  &X, &N
      LCL    &M        ——— declares M as local EV
&M    SET    0         ——— M initialized to 0
      MOVER  AREG, = '0'
```

(repeated loops).

```
.MORE   MOVEM   AREG, &X+&M
        &M      SET     &M+1        — increment value of M
        AIF     (&M NE N).MORE      till it is equal to N which is 3.
        MEND.
```

expansion of macro call

```
CLEAR   B, 3.
```

— Macro call leads to generation of statements

```
+ MOVER   AREG, ='0'
+ MOVEM   AREG, B
+ MOVEM   AREG, B+1
+ MOVEM   AREG, B+2.
```

→ most expansion time loops can be replaced by execution time loops.

— In above program no need to write so many MOVEM statements, it is possible to write an execution time loop which moves 0 into B, B+1 & B+2.

— An execution time loop leads to more compact assembly programs. such programs would execute slower than programs containing expansion time loops. Thus a macro can be used to trade program size for execution efficiency.

Date : / /

→ Other facilities for expansion time loops:

- ELSE clause in AIF.
- assemblers for M 68000 and Intel 8086 processors provide explicit expansion time looping constructs.

REPT statement

REPT <expression>

st. repeated number of times.

evaluate to a numerical value during macro expansion.

```
        MACRO
        CONST 10
        LCL   &m
&m      SET   1
        REPT  10
        DC    '&m'
&m      SET A &m+1
        ENDM
        MEND.
```

repeated 10 times.

IRP statement

IRP <formal parameter>, <argument list>

(argument list: takes successive values from)

- For each value, the st. betn IRP and END M are expanded once.

```
MACRO
CONSTS  &M, &N, &Z
IRP     &Z, &M, 7, &N
DC      '&Z'.
ENDM
MEND
```

- A macro call CONTS 4,10 leads to declaration of 3 constants with values 4, 7 and 10.

→ Semantic expansion.

- It is the generation of instructions tailored to the requirements of a specific usage.
- It can be achieved by a combination of advanced macro facilities like AIF, AGO & expansion time variables.
- CLEAR macro is an instance of sematic
- no. of MOVEM AREG, ... st. generated by a call on CLEAR is determined by the value of 2nd parameter of CLEAR.
- MACRO EVAL is also an instance of conditional expansion wherein one of two alternative code sequences is generated depending on the peculiarities of actual parameters of a macro call.

ex:- semantic expansion using type attribute

```
MACRO
CREATE_CONST &X, &Y
```

```
        AIF (T' &X EQ B) .BYTE
&Y      DW      25
        AGO     .OVER
.BYTE   ANOP
&Y      DB      25
.OVER   MEND
```

- created a constant '25' with the name given by 2nd parameter. The type of constant matches the type of the 1st parameter.

→ Design of Macro Preprocessor

- The macro preprocessor accepts an assembly program containing definitions and calls and translates it into an assembly program which does not contain any macro definition or calls.

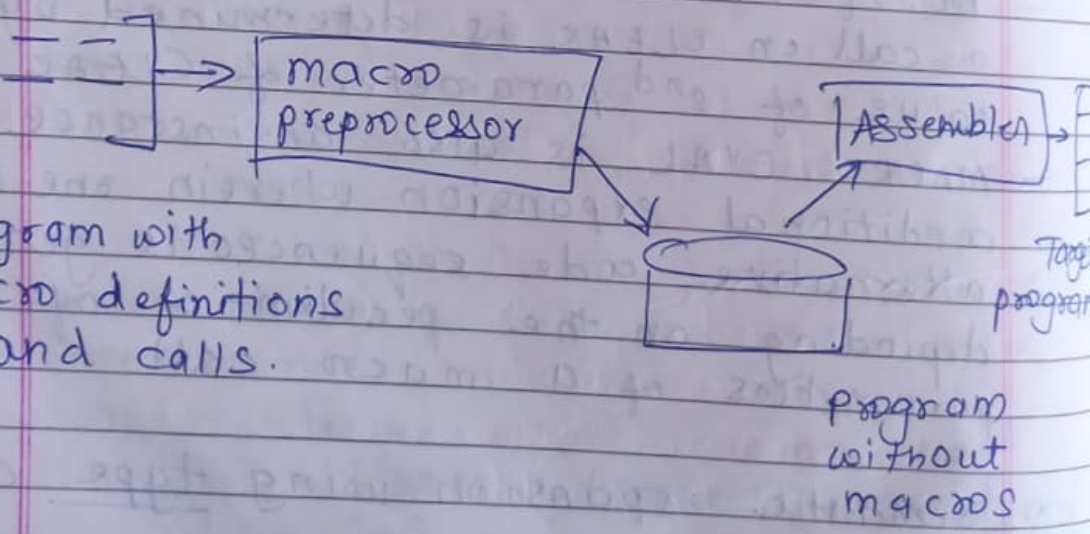


Fig:- A schematic of a macro preprocessor.

- The program form output by the macro preprocessor can now be handed over to an assembler to obtain the target language form of the program.

- Thus the macro preprocessor segregates macro expansion from the process of program assembly.

→ Design Overview

- Tasks involved in macro expansion.

1. Identify macro calls in the program.
2. Determine the values of formal parameters
3. Maintain the values of expansion time variables declared in a macro.
4. Organize expansion time control flow.
5. Determine the values of sequencing symbols.
6. Perform expansion of a model statement.

→ A 4 step procedure is followed to arrive at a design specification for each task:

1. Identify the information necessary to perform a task.
2. Design a suitable data structure to record the information.

3. Determine the processing necessary to obtain the information.
4. Determine the processing necessary to perform the task.

→ Application of this procedure to each of the preprocessor tasks in described.

* Identify macro calls.

- A table called the macro name table (MNT) is designed to hold the names of all macros defined in a program.
- A macro name is entered in this table when a macro definition is processed.
- While processing a statement in the source prg, the preprocessor compares the string found in its mnemonic field with the macro names in MNT. A match indicates that the current statement is a macro call.

* Determine values of formal parameters.

- A table called the actual parameter table (APT) is designed to hold the values of formal parameters during the expansion of a macro call.
- each entry in table is a pair

(<formal parameter name>, <value>) — value of keyword para

- For this, a table called parameter default table (PDT) is used for each macro. This table would be accessible from the MNT entry of a macro and would contain pairs of the form (<formal parameter name>, <default ~~nan~~Value>).
- If a macro call statement does not specify a value for some parameter par, its default value would be copied from PDT to APT.

* Maintain expansion time variables.

- An expansion time variable table (EVT) is maintained. Table contains

  (<EV name>, <value>).

  ↳ accessed when a preprocessor st or a model st under expansion refers to an EV.

* Organize expansion time control flow.

- The body of a macro, i.e. the set of preprocessor st. and model st. in it, is stored in a table called macro definition table (MDT) for use during macro expansion.
- The flow of control during macro expansion determines when a model st. is to be visited for expansion.
- MEC is initialized to the 1st st. of the macro body in the MDT. It is updated

after expanding a model st. or on processing a macro preprocessor st

* Determine values of sequencing symbols

— A sequencing symbol table (SST) is maintained to hold info. like (<sequencing symbol name>, <MDT entry number of the MDT entry which contains the model st. defining the SS

— This entry is made on encountering a st. which contains the SS in its label field. (in case of back reference) or on encountering a reference prior to its defn (in case of forward reference).

* Perform expansion of a model statement

1. MEC points to the MDT entry containing the model st.
2. values of formal parameters & EV's are available in APT and EVT, resp.
3. model st defining a SS can be identified from SST

* Data structures

— Table of macro preprocessor.

SMVS
Page No.
Date : / /

Table	Fields in each entry
macro name table (MNT)	macro name, no. of positional parameters (#PP) no. of keyword para. (#KP) no. of EV's (#EV) MDT pointer (MDTP) KPDTAB pointer (KPDTP) SSTAB pt. (SSTP).
parameter Name Table (PNTAB)	parameter name
EV Name Table (EVNTAB)	EV name
SS name table (SSNTAB)	SS name
Keyword parameter default Table (KPDTAB)	parameter name, default value
macro defn Table (MDT)	Label, opcode, operands
actual parameter table (APTAB)	value
EV Table (EVTAB)	value

વાંચવા-વિચારવા કરતાં વર્તનમાં ઉતારવું વધુ જરૂરી છે.

MDT entry #

SS Table (SSTAB)

EX - 24)

```
    MACRO
    CLEARMEM  &X, &N, &REG=AREG
    LCL       &M
    SET       0
    MOVER     &REG, ='0'
.MORE MOVEM   &REG, &X+&M
    SET       &M+1
    AIF       (&M NE &N) .MORE
    MEND
```

(P,1) (P,2) (P,3)

positional — Keyword

AREG, 10, AREG,

→ CLEARMEM AREAR, 10

used during processing of a macro definition.

PNTAB: X | N | REG

EVNTAB: M

SSNTAB: MORE

MNT

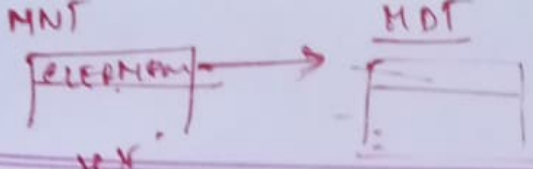
name	#PP	#KP	#EV	MDTP	KPDTP	SSTP
CLEARMEM	2	1	1	25	10	5

KPDTAB

10	REG	AREG

SSTAB

5	28



SMVS
Page No.
Date : / /

pointer

MDT	label	opcode	operand.
25		LCL	(E,1)
26	(E,1)	SET	0
27		MOVER	(P,3) ='0'
28		MOVEM	(P,3),(P,1)+(E,1)
29	(E,1)	SET	(E,1)+1
30		AIF	((E,1) NE (P,2)) (S,1)
31		MEND	

APTAB

AREG
10
AREG

EVTAB

0

Fig:- Data structures of macro preprocessor.

constructed during macro diff processing & used during macro expansion.

Processing of macro definitions

KPDTAB_pointer = 1;
SSTAB_ptr = 1;
MDT_ptr = 1;

Algo:- (Processing of a macro definition)

1. SSNTAB_ptr = 1;
   PNTAB_ptr = 1;

2. Process the macro prototype statement and form the MNT entry.

(a) name = macro name;
(b) For each positional parameter
   (i) Enter parameter name in PNTAB [PNTAB_ptr].
   (ii) PNTAB_ptr = PNTAB_ptr + 1;
   (iii) #PP = #PP + 1;

(c) KPDTP = KPDTAB_ptr;
(d) For each keyword parameter

(i) Enter parameter name & default value (if any), in KPDTAB [KPDTAB_ptr]

(ii) Enter parameter name in PNTAB [PNTAB_ptr]

(iii) KPDTAB_ptr = KPDTAB_ptr + 1;

(iv) PNTAB_ptr = PNTAB_ptr + 1;
(v) #KP = #KP + 1;

(e) MDTP = MDT_ptr;
(f) #EV = 0;
(g) SSTP = SSTP_ptr;

3. While not a MEND statement.
(a) If an LCL statement then
(i) Enter expansion time variable name in EVNTAB.
(ii) #EV = #EV + 1;
(b) If a model statement then
(i) If label field contains a sequencing symbol then
If symbol is present in SSNTAB then
q = entry number is SSNTAB;
else
enter symbol is SSNTAB [SSNTAB_ptr].
q = SSNTAB_ptr;
SSTAB[SSTP + q - 1] = MDT_ptr;

(ii) For a parameter, generate the specification (P, #n)
(iii) For an expansion variable, generate the specification (E, #m)
(iv) Record the IC in MDT (MDT_ptr);
(v) MDT_ptr = MDT_ptr + 1;

(c) If a preprocessor statement then

(i) If a SET statement
search expansion time variable name used in the st. in EVNTAB and generate the spec (E, #m)

(ii) If an AIF and AGO st. then
If sequencing symbol used in the st. is present in SSNTAB then
q = entry no. in SSNTAB;
else
enter symbol in SSNTAB[SSNTAB_ptr]
q = SSNTAB_ptr;
SSNTAB_ptr = SSNTAB_ptr + 1;

Replace the symbol by (S, SSTP+q-1)

(iii) Record the IC in MDT (MDT_ptr)
(iv) MDT_ptr = MDT_ptr + 1;

4. (MEND statement)

If SSNTAB_ptr = 1 (i.e. SSNTAB is empty)
then SSTP = 0;
else
SSTAB_ptr = SSTAB_ptr + SSNTAB_ptr-1

If #KP = 0 then KPDTP = 0;

Macro Expansion.

Data structures.

APTAB	actual parameter table
EVTAB	EV table
MEC	macro expansion counter
APTAB_ptr	APTAB pointer
EVTAB_ptr	EVTAB pointer

- no. of entries in APTAB (#^{e}APTAB) = sum of values in #PP & #KP field of MNT entry of a macro.
- No. of entries in EVTAB (#^{e}EVTAB) is given by the value in #EV field of the MNT.
- APTAB & EVTAB are constructed when a macro call is recognized.
- APTAB_ptr & EVTAB_ptr are set to point at these tables.

Algorithm :- (Macro expansion)

1. Perform initializations for the expansion of a macro.

(a) MEC = MDTP field of the MNT entry;
(b) Create EVTAB with #EV entries & set EVTAB_ptr;
(c) Create APTAB with #PP + #KP entries & set APTAB_ptr.

(d) Copy keyword parameter defaults from the entries KPDTAB [KPDTP] ... KPDTAB [KPDTP + #KP - 1] into APTAB [#PP+1] APTAB [#PP + #KP]

(e) Process positional parameters in the actual parameter list and copy them into APTAB [1] .... APTAB [#PP].

(f) For keyword parameters in the actual parameter list
Search the keyword name in parameter name field of KPDTAB [KPD TP] .... KPDTAB [KP DTP + #KP -1]
Let KPTDAB [q] contain a matching entry. Entry value of the keyword parameter in the call (if any) in APTAB [#PP + q - KPDTP + 1].

2. While statement pointed by MEC is not MEND statement.

(a) If a model statement then

(i) Replace operands of the form (P, #n) and (E, #m) by values in APTAB [n] and EVTAB [m] resp.

(ii) output the generated statement
(iii) MEC = MEC + 1;

(b) If a SET statement with the specification (E,#m) in the label field then

(i) Evaluate the expression in the operand field and set an appropriate value in EVTAB [m].
(ii) MEC = MEC + 1;

(c) If an AGO statement with (S, #S) in operand field then

MEC = SSTAB [SSTP + S - 1];

(d) If an AIF st. with (S, #S) in operand field then
If condition in the AIF st. is true then
MEC = SSTAB [SSTP + S - 1];

3. Exit from macro expansion.

MEC 28

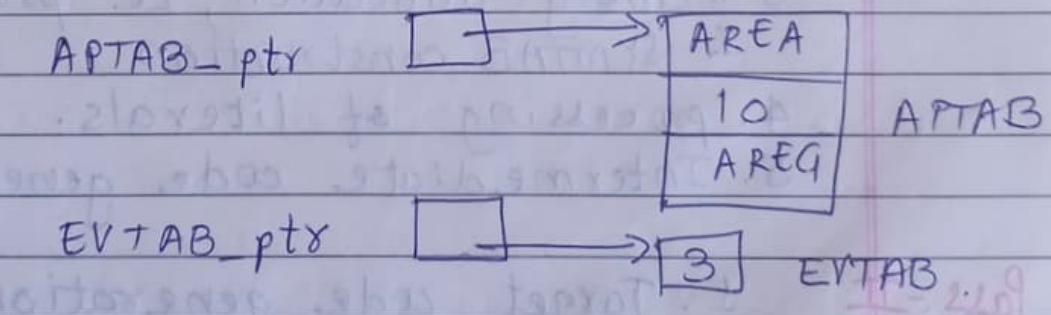


Fig:- DS during macro expansion

* use of stack in macro preprocessor (self study)

Design of macro - Assembler.

Pass-I
1. Macro definition processing
2. SYMTAB construction

Pass-II
1. macro expansion.
2. memory allocation & LC processing
3. Processing of literals.
4. Intermediate code generation.

Pass-III
1. Target code generation.

- Pass-II is large in size since it performs many functions.

→ Pass-I of the assembler with the preprocessor would give us the following two pass structure:

Pass-I
1. Macro definition processing
2. macro expansion
3. memory allocation, LC processing & SYMTAB construction
4. processing of literals.
5. Intermediate code generation.

Pass-II
1. Target code generation.