3. MACRO AND MACRO PROCESSORS

Macros are used to provide a program generation facility through expansion. Many languages provide built in facilities for writing macro e.g. PL/1, C, Ada and C++. When a language does not support built-in macro facilities then a programmer may achieve an equivalent effect by using generalized preprocessor or software tools like Awk of UNIX

Macro definition: 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.

Difference between macro and subroutines: Use of macro name in the mnemonic field of an assembly statement leads to its expansion whereas use of a subroutine name in a call instruction leads to its execution. Thus the programs using macros and subroutines differ significantly in terms of program size and execution efficiency.

3.1 MACRO DEFINITION AND CALL

Macro Definition

A macro definition is enclosed between a macro header statement and a macro end statement. Macro definitions are typically located at the start of a program. Macro definition consists of:

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

1. Macro prototype statements.

The macro prototype statement declares the name of the macro and the names and kinds of its parameters. The macro prototype statement has the following syntax:

<*macro name*> [<*formal parameter specification*>[…..]]

where <*macro name*> appears in the mnemonic field of assembly Statement and <formal parameter specification> is of the form

&<parameter name> [<parameter kind>]

2. Model statements.

A model statement is a statement from which assembly language statements may be generated during macro expansion.

3. Macro preprocessor statements.

A preprocessor statement is used to perform auxiliary functions during macro expansion.

Macro call

A macro is called by writing the macro name in the mnemonic field of an assembly statement. The macro call has syntax

<macro name> [<actual parameter specification>[,….]]

where actual parameter resembles an operand specification in assembly statement.

Example 1: The definition of macro INCR is given below

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

The prototype statement indicates that three parameters called MEM_VAL, INCR_VAL, and REG are parameters. Since parameter kind isn't specified for any of the parameters, they are all of the default kind 'positional parameters'. Statements with the operation codes MOVER, ADD, MOVEM are model statements. No preprocessor statements are used in this macro.

3.2 MACRO EXPANSION

A macro leads to macro expansion. During macro expansion, the use of a macro name with a set of actual parameters i.e. the macro call statement is replaced by a sequence of assembly code. To differentiate between the original statement of a program and the statements resulting from macro expansion, each expanded statement is marked with a '+' preceding its label field.

There are two kinds of expansions

1. Lexical expansion.
2. Semantic expansion

1. Lexical expansion.

Lexical expansion implies replacement of a character string by another character string during program generation. It replaces the occurrences of formal parameter by corresponding actual parameters.

Example 2: Consider the following macro definition

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

Consider macro call statement on INCR

```
INCR    A, B, AREG
```

The values of formal parameters are:

Formal parameter	value
MEM_VAL	A
INCR_VAL	B
REG	AREG

Lexical expansion of the model statement leads to the code

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

2. Semantic expansion

Semantic expansion implies generation of instructions tailored to the requirements of a specific usage e.g. generation of type specific instruction for manipulation of byte and word operands. It is characterized by the fact that different use of macro can lead to codes which differ in the number, sequence and opcodes of instructions.

Example 3: Consider the macro definition

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

Consider the macro call CLEAR B, 3

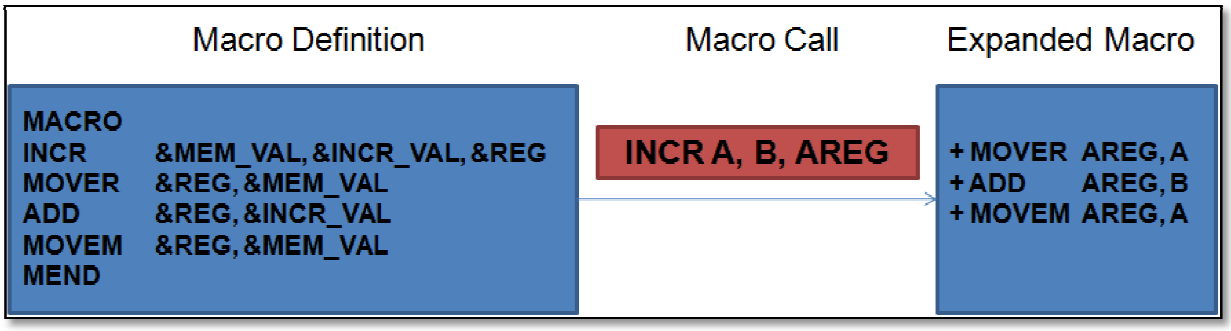
This macro call leads to generation of statements:

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

Two key notions of macro expansion are:

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

Lexical substitution: It 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 macro are visited sequentially starting with the statement following the macro prototype statement and ending with the statement preceding the MEND statement.

Preprocessor statements can alter the flow of control during expansion. So some model statements are either never visited during expansion or are repeatedly visited during expansion.

Never visited → conditional expansion.

Repeatedly visited → expansion time loop.

The flow of control during macro expansion is implemented using macro expansion counter.

Algorithm - Outline of macro expansion

1. MEC: = statement number of first statement following the prototype statements.
2. While statement pointed by MEC is not a MEND statement
   (a)If a model statement then
      (i)Expand the statement
      (ii)MEC: = MEC+1;
   (b) Else (i.e. a preprocessor statement)

(i) MEC: = new value specified in the statement;

3. Exit from macro expansion.

MEC is set to point at the statement following prototype statement. It is incremented by 1 after expanding a model statement. Execution of a preprocessor statement can set MEC to a new value to implement conditional expansion or expansion time loop.

Lexical Substitution

A model statement consists of 3 types of strings:

1. An ordinary string, which stands for itself.
2. The name of formal parameter which is preceded by the character '&'
3. The name of a preprocessor variable, which is also preceded by the character '&'.

During lexical expansion, strings of type 1 are retained without substitution. Strings of type 2 and type 3 are replaced by the values of the formal parameters or preprocessor variables. The rules for determining the value of a formal parameter depend on the kind of parameter.

Types of parameters

1. Positional parameters.
2. Keyword parameters.
3. Default parameters.
4. Macros with mixed parameter list.
5. Other uses of parameters.

1. Positional parameters

A positional formal parameter is written as &<*parameter name*>, e.g. &SAMPLE where SAMPLE is name of parameter. A <*actual parameter specification*> is simply an <*ordinary string*>.

The value of positional parameter XYZ is determined by the rule of positional association as follows:

(i) Find the ordinal position of XYZ in the list of formal parameters in the macro prototype statement.

(ii) Find the actual parameter specification occupying the same ordinal position in the list of actual parameters in the macro call statement. Let this be the ordinary string ABC. Then the value of formal parameter XYZ is ABC.

Example 4: Consider an example

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

In above example, the macro call

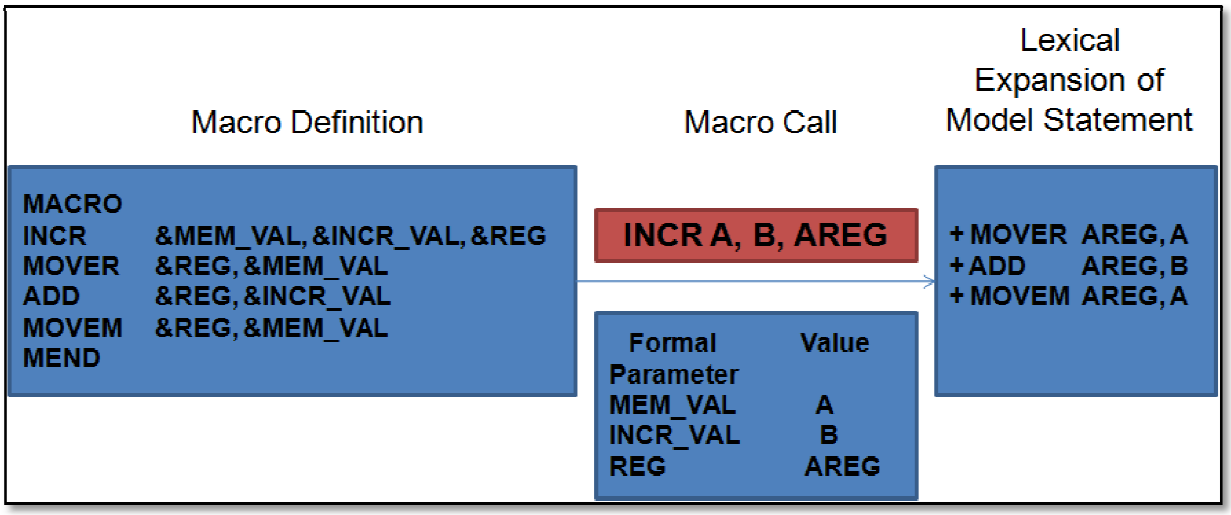
```
INCR    A, B, AREG
```

By rule of positional association, values of the formal parameters are:

Formal parameter	value
MEM_VAL	A
INCR_VAL	B
REG	AREG

Lexical expansion of the model statement leads to the code

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



2. Keyword parameters

A keyword formal parameter is written as '&<parameter name>='. The *<actual parameter specification>* is written as *<formal parameter name>=<ordinary string>*

The value of a formal parameter XYZ is determined by the rule of keyword association as follows:

(i) Find the actual parameter specification which has the form XYZ=<ordinary string>.

(ii) Let <ordinary string> in the specification be the string ABC. Then the value of formal parameter XYZ is ABC.

The ordinal position of the specification XYZ=ABC in the list of actual parameters is immaterial. This is very useful in situations where long lists of parameters have to be used.

Example 5: Consider an example

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

In above example, the following macro calls are equivalent.

```
INCR    MEM_VAL=A, INCR_VAL=B, REG=AREG
```

or

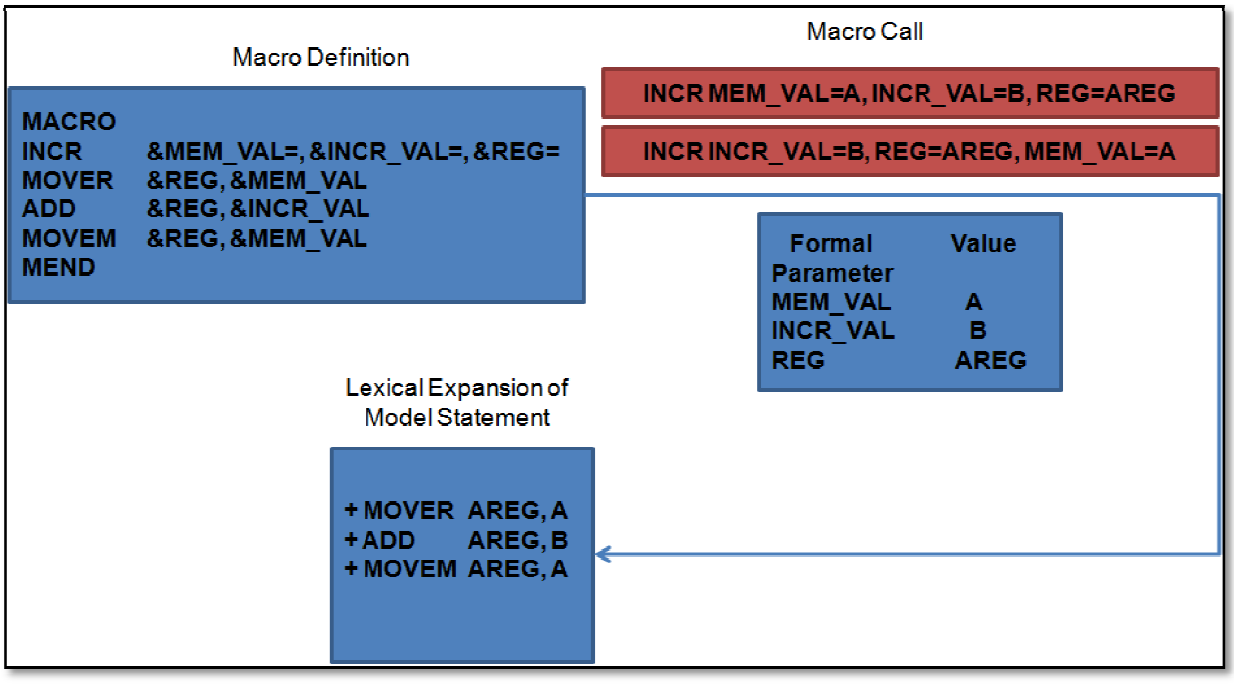
```
INCR    INCR_VAL=B, REG=AREG, MEM_VAL=A
```

By rule of keyword association, values of the formal parameters are:

Formal parameter	value
MEM_VAL	A
INCR_VAL	B

Lexical expansion of model statement leads to the code.

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



3. Default specifications of parameters

A default is a standard assumption in the absence of an explicit specification by the programmer. Default specification of parameters is useful in situations where a parameter has the same value in most calls. When the desired value is different from the default value, the desired value can be specified explicitly in a macro call. This specification overrides the default value of the parameter for the duration of the call.

The syntax for formal parameter specification is as follows:

&<parameter name>[<parameter kind>[<default value>]]

Example 6: Consider an example

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

In above example, the following macro calls are equivalent.

```
INCR        MEM_VAL=A, INCR_VAL=B, REG=BREG
INCR        MEM_VAL=A, INCR_VAL=B
INCR        MEM_VAL=B, MEM_VAL=A
```

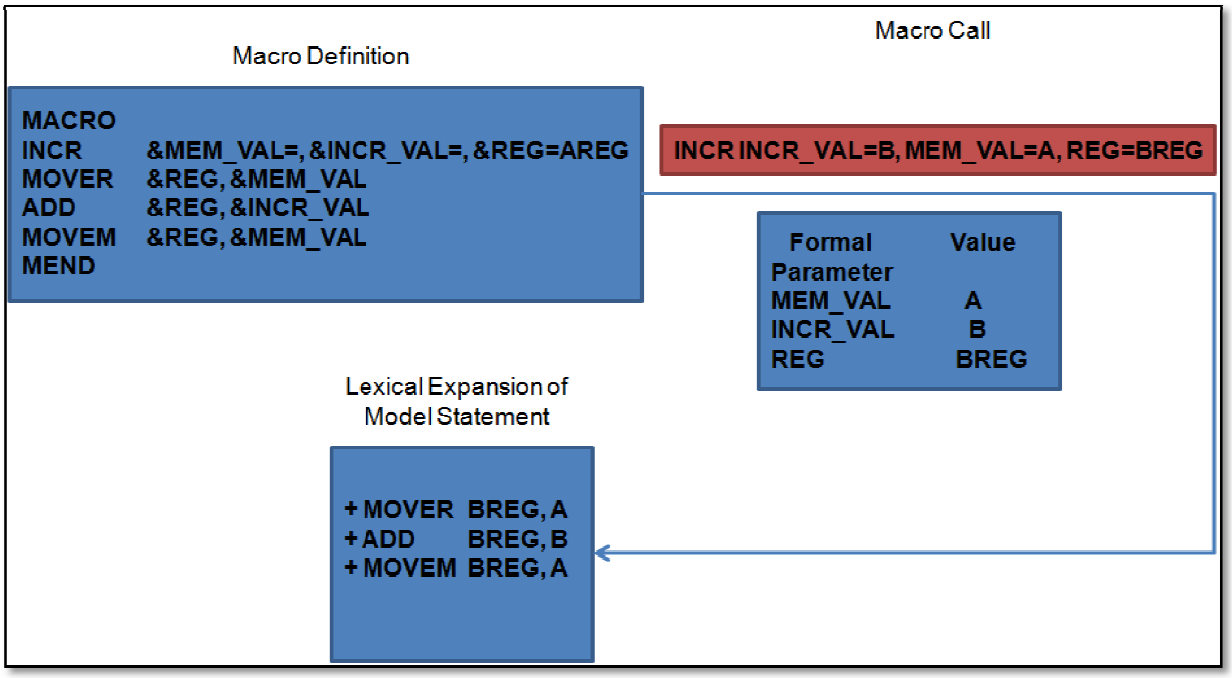
The first call overrides to us a default specification for the parameter BREG.

By rule of default association, values of the formal parameters are:

Formal parameter	value
MEM_VAL	A
INCR_VAL	B
REG	AREG

Lexical expansion of model statement leads to the code.

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



4. Macro with mixed parameter lists

A macro may be defined to use both positional and keyword parameters. In such case, all positional parameters must precede all keyword parameters.

Example 7: Consider an example

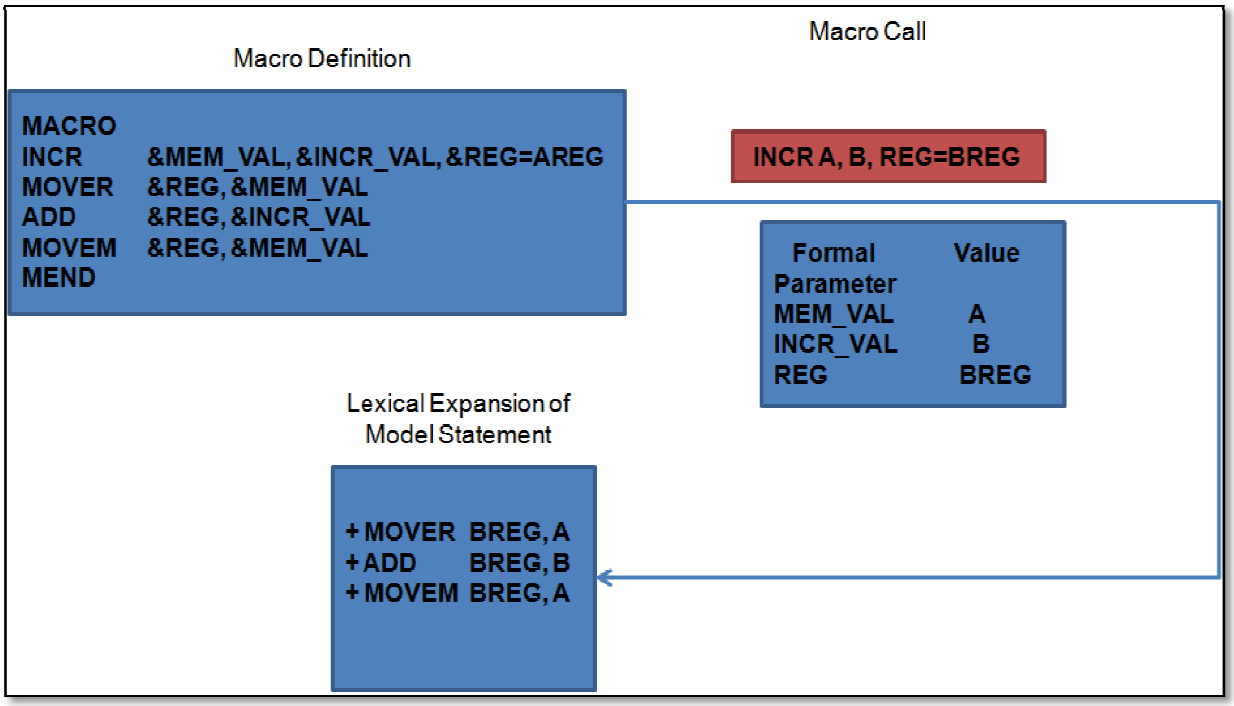
```
MACRO
INCR        &MEM_VAL, &INCR_VAL, &REG=
MOVER       &REG, &MEM_VAL
ADD         &REG, &INCR-VAL
MOVEM       &REG, &MEM_VAL
 MEND
```

In above example, the macro

```
INCR        A, B, REG=BREG
```

Lexical expansion of model statement leads to the code.

```
+    MOVER    BREG, A
+    ADD      BREG, B
+    MOVEM    BREG, A
```



5. Other uses of parameters

The use of parameters is not restricted to operand fields. The formal parameters can also appear in the label and operand fields of model statements.

Example 8: Consider an example

```
        MACRO
        CALC    &X, &Y, &OP=MULT, &LAB=
&LAB    MOVER   AREG, &X
        &OP     AREG, &Y
        MOVEM   AREG, &X
```

MEND

The formal parameters and their values are

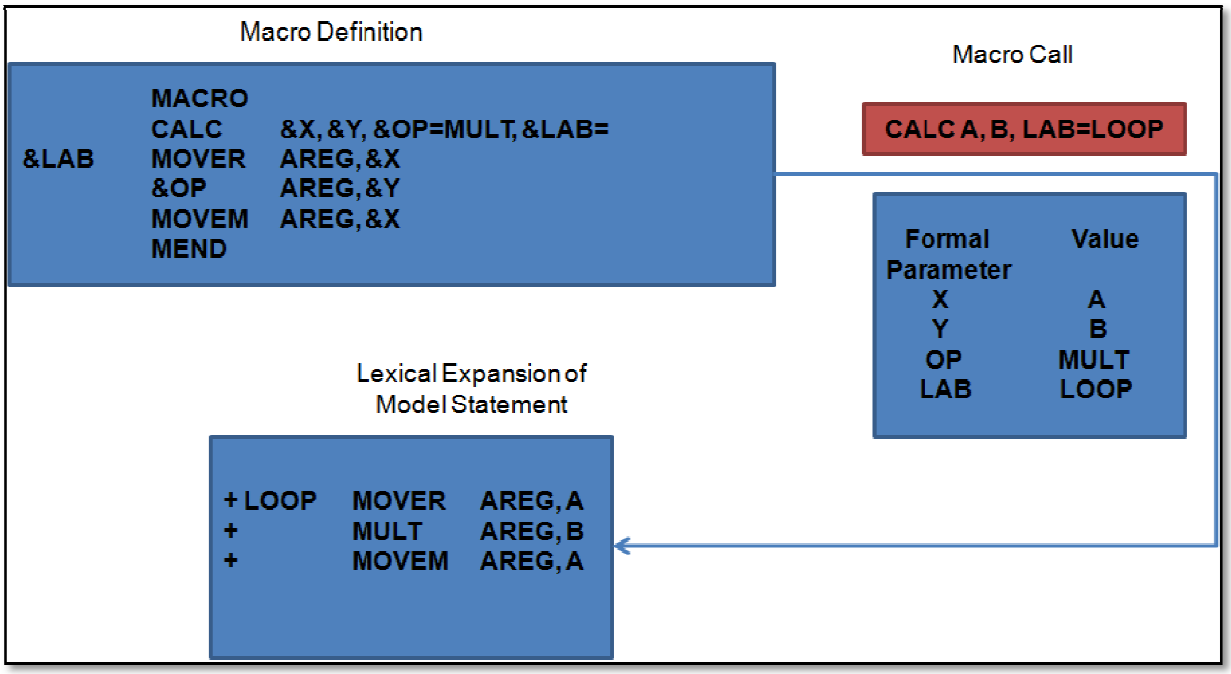
Formal parameter	value
X	A
Y	B
OP	MULT
LAB	LOOP

Consider the macro call

CALC A, B, LAB = LOOP

Lexical expansion of model statement leads to the code.

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



3.3 NESTED MACRO CALLS

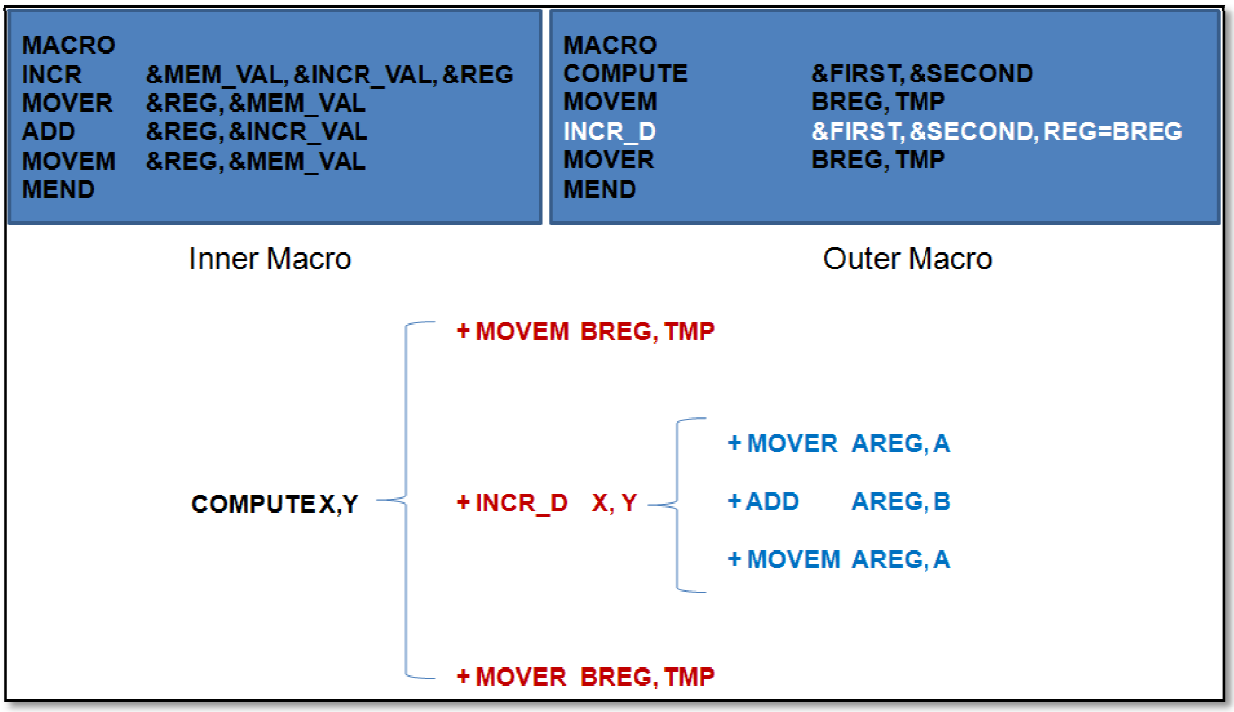
A model statement in a macro may constitute a call on another macro. Such calls are known as nested macro calls. Expansion of nested macro calls follows the last-in-first-out (LIFO) rule. Thus, in a structure of nested macro calls, expansion of the latest macro call (I.e. the innermost macro call in the structure) is completed first.

Example 9: Consider an example

```
        MACRO
        COMPUTE   &FRIST, &SECOND
        MOVEM     BREG, TMP
        INCR      &FIRST, &SECOND, AREG
        MOVER     BREG, TMP
        MEND
```

and macro definition

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



3.4 ADVANCED MACRO FACILITIES

Advanced macro facilities are aimed at supporting semantic expansion. These facilities can be grouped into

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

1. Alteration of flow of control during expansion

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

(i) Expansion time sequencing symbols.

(ii) Expansion time statements AIF, AGO and ANOP.

Expansion time sequencing symbols

A sequencing symbol (SS) as syntax as:

.<*ordinary string*>

As 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 mode statement.

Expansion time statements - AIF, AGO and ANOP

An AIF statement has syntax:

AIF (<*expression*>) <*sequencing symbol*>

where <expression> is a relational expression involving ordinary strings, formal parameters and their attributes, and expansion time variable. If the relational expression evaluates to true, expansion time control is transferred to the statement containing <*sequencing symbol*> in its label field.

An AGO statement has syntax:

AGO<*sequencing symbol*>

and unconditionally transfers expansion time control to the statement containing <*sequencing symbol*> in its label field.

An ANOP statement is written as:

<s*equencing symbol*>ANOP

and simply defines the sequencing symbol.

Example 10:

```
                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
```

2. Expansion time variable

Expansion time variables (EV's) are variables which can only be used during the expansion of macro calls. There are two types of EV's.

(i) ***Local EV:*** A local EV is created for use only during a particular macro call and has following syntax.

LCL <*EV specification*> [, <*EV specification*>…]

(ii) ***Global EV*****:** 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. It has following syntax

GBL <*EV specification*> [, <*EV specification*>..]

<EV specification> has a syntax has &<EV name>, where <EV name> is an ordinary string. Values of EV's can be manipulated through the preprocessor statement SET. A SET statement is written as

<*EV specification*> SET <*SET expression*>

where <*EV specification*> appears in the label field and SET in mnemonic field. A SET statement assigns the value of <*SET expression*> to the EV specified in <*EV specification*>. The value of EV can be used in any field of a model statement and in the expression of an AIF statement.

Example 11:

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

A call on macro CONSTANTS is expanded as follow: The local EV A is created. The first Set statement assigns the value '1' to it. The first DB statement thus declared a

byte constant '1'. The second SET statement assigns the value '2' to A and the second DB statement declares a constant '2'.

3. Attributes of formal parameters

An attributes is written using the syntax:

<*attribute name*>'<*formal parameter specification*>

It represents information about the value of the formal parameter i.e. about corresponding actual parameter. The type, length and size attributes have the names T, L and S.

Example 12:

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

Here expansion time control is transferred to the statement having .NEXT in its label field only if the parameter corresponding to the formal parameter A has the length of '1'.

3.4.1 Conditional expansion

Conditional expansion helps in generating assembly code specifically suited to the parameters in a macro call. This is achieved by ensuring that a model statement is visited only under specific conditions during the expansion of a macro. The AIF and AGO statements are used for this purpose.

Example: Consider a macro EVAL such that a call

```
EVAL A, B, C
```

generates efficient code to evaluate A-B+C in AREG. When the first two parameter of a call are identical then EVAL should generate a single MOVER instruction to load the 3rd parameter into AREG. This is achieved as follows:

Example 13:

```
        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
```

Since the value of a formal parameter is simply the corresponding actual parameter, the AIF statement effectively compares names of the first two actual parameters. If the names are same, expansion time control is transferred to the model statement MOVER AREG, &Z. If not, the MOVE-SUB-ADD sequence is generated and expansion time control is transferred to the statement .OVER MEND which terminates the expansion.

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 statement in the macro.

Example 14: Consider example

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

When called as CLEAR B, the statement puts the value 0 in AREG, while the three MOVEM statements store this value in 3 consecutive bytes with the address B, B+1, B+2.

The same can be achieved by writing an expansion time loop which visits the model statement, or set of model statements, repeatedly during macro expansion. Expansion time loops can be written using EV's and expansion time control transfer statements AIF and AGO.

Example 15:

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

Consider expansion of macro call

```
        CLEAR   B, 3
```

The macro calls leads to generation of the statements

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

3.4.2 Other facilities for expansion time loops

The assemblers provide explicit expansion time looping constructs.

1. REPT statement
2. IRP statement

REPT statement

Syntax: REPT<expression>

<expression> should evaluate to a numerical value during macro expansion. The statements between REPT and ENDM statement would be processed for expansion <expression> number of times.

Example 16: Following example illustrates the use of this facility to declare 10 constants with the value 1, 2, …, 10

```
            MACRO
            CONST10
            LCL     &M
&M          SET     1
            REPT    10
            DC      '&M'
&M          SETA    &M+1
            ENDM
            MEND
```

IRP statement

Syntax: IRP <formal parameter>, <argument list>

The formal parameter mentioned in the statements takes successive values from the argument list. For each value, the statements between the IRP and ENDM statements are expanded once.

Example 17:

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

A macro call CONSTS 4, 10 leads to declaration of 3 constants with the values 4, 7, 10.

3.4.3 Semantic expansion

Semantic expansion is the generation of instructions. It can be achieved by a combination of advanced macro facilities like AIF, AGO statements and expansion time variables.

Example 18:

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

This macro creates a constant '25' with the name given by the 2nd parameter type of the constant matches the type of the first parameter.

Example 19:

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

Example 20:

```
            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
```

3.5 DESIGN OF A MACRO PROPROCESSOR

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

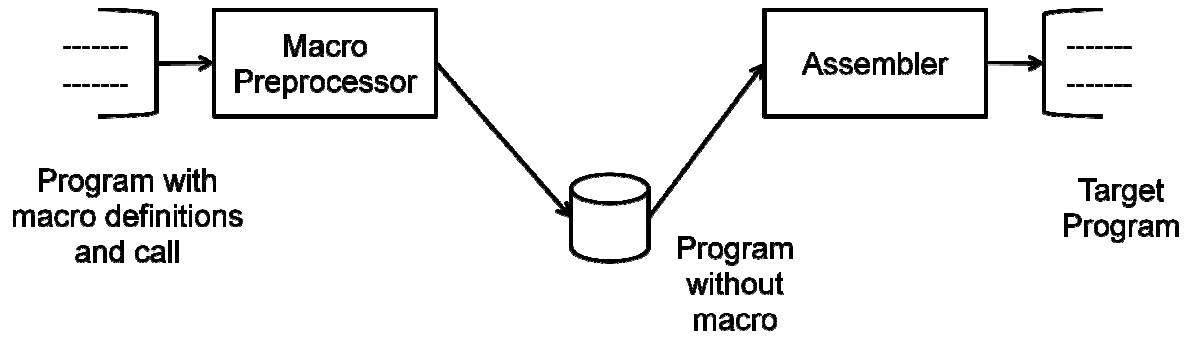


Figure: A Schematic of a macro preprocessor

The macro preprocessor is economical because it can use an existing assembler.

3.5.1 Design Overview

All tasks involved in macro expansion are

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

The following 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 and maintain the information.
4. Determine the processing necessary to perform the task

Application of this procedure to each of the preprocessor tasks is described in the following

Identify macro calls in the program

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

Determine the values of formal parameter.

Actual Parameter Table (APT) is designed to hold the values of formal parameters during expansion of macro call. Each entry in the table is a pair

(<*formal parameter name*>, <*value*>)

Parameter Default Table (PDT) is designed to hold the default values of keyword parameters during expansion of macro call. Each entry in the PDT is a pair

(<formal parameter name>, <default value>)

If a macro call statement does not specify a value for some parameter then its default value would be copied from PDT to APT.

Maintain expansion time variables

Expansion Time Variable's Table (EVT) is maintained for this purpose. Each entry in the table is a pair

(<*EV name*>, <*value*>)

The value field of a pair is accessed when a preprocessor statement or model statement under expansion refers to an EV.

Organize expansion time control flow.

Macro definition Table (MDT) is used to store the body of macro. The flow of control during macro expansion determines when a model statement is to be visited for expansion.

Determine the values of sequencing symbol.

A sequencing Symbol Table (SST) is maintained to hold this information. The table contains pair of the form

(<*sequencing symbol name*>, <*MDT entry #*>)

where < *MDT entry #*> is the number of the MDT entry which contains the model statement defining the sequencing symbol. This entry is made on encountering a statement which contains the sequencing symbol in its label field or on encountering a reference prior to its definition.

Perform expansion of a model statement

This is the trivial task given below:

1. MEC points to the MDT entry containing the model statement.
2. Values of formal parameter and EV's are available in APT and EVT respectively.
3. The model statement defining a sequencing symbol can be identified from SST.

Expansion of model statement is achieved by performing a lexical substitution for the parameters and EV's used in the model statement.

3.5.2 Data Structure

The tables APT, PDT and EVT contains pairs which are searched using the first component of the pair as a key e.g. the formal parameter name is used as the key to obtain its value from APT. This search can be eliminated if the position of an entity within a table is known when its value is to be accessed.

Thus macro expansion can be made more efficient by storing an intermediate code for a statement rather than its source form in MDT table.

E.g. MOVER AREG, ABC

Let the pair (ABC, ALPHA) occupy entry#5 in APT. The search in APT can be avoided if the model statement appears as

MOVER AREG, (P, 5)

in MDT where (P, 5) stands for the words 'parameter#5'

So the information (<formal parameter name>,<value>) in APT has been split into two tables:

PNTAB contains formal parameter names

APTAB contains formal parameter values

Similarly analysis leads to splitting of EVT into EVNTAB and EVTAN and SST into SSNTAB and SSTAB. PDT is replaced by a keyword parameter default table (KPDTAB).

Table of macro preprocessor

Table	**Fields in each enter**
Macro Name Table (MNT)	• Macro name, • Number of positional parameters (#PP), • Number of keyword parameters (#KP),

	• Number of expansion time variable (#EV), • MDT pointer (MDTP), • KPDTAB pointer (KPDTP), • SSTAB pointer (SSTP)
Parameter Name Table (PNTAB)	• Parameter name
EV Name Table (EVNTAB)	• EV name
Keyword Parameter Default Table (KPDTAB)	• Parameter name, • Default value
Macro Definition Table (MDT)	• Label, • Opcode, • Operand
Actual Parameter Table (APTAB)	• Value
EV Table (EVTAB)	• Value
SS Table(SSTAB)	• MDT entry #

- PNTAB and KPDTAB are constructed by processing the prototype statement.
- Entries are added to EVNTAB and SSNTAB as EV declaration and SS definition/references are encountered.
- MDT entries are constructed while processing model statement and preprocessor statement in macro body.
- An entry is added to SSTAB when the definition of a sequencing symbol is encountered.
- APTAB is constructed while processing a macro call.
- EVTAB is constructed at the start of expansion of macro.

Example 21:

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

Consider the macro call

```
CLEAR_MEM   AREA, 10, AREG
```

PNTAB

X
N
REG

EVNTAB

M

SSNTAB

MORE

KPDTAB

Pointer(KPDTP)	**Formal parameter**	**Default value**
10	REG	AREG

MDT

Pointer (MDTP)	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 (S, 2)) (S, 1)
31		MEND	

MNT

Name	#PP	#KP	#EV	MDTP	KPDTP	SSTP
CLEAR_MEM	2	1	1	25	10	5

APTAB

AREA
10
AREG

EVTAB

0

ALGORITHM FOR PROCESSING OF A MACRO DEFINITION

1. **SSNTAB_PTR := 1;**
   PNTAB_PTR := 1;
   KPDTAB_PTR := 1;
   SSTAB_PTR := 1;
   MDT_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 and default value in KPDTP[KPDTP_PTR]
      ii) Enter parameter name in PNTAB[PNTAB_PTR]
      iii) KPTAB_PTR := KPDTP_PTR+1
      iv) PNTAB_PTR := PNTAB_PTR+1
      v) #KP := #KP+1

   e) MDTP := MDT_PTR
   f) #EV := 0
   g) SSTP := SSTAB_PTR

3. **While not a MEND**
   a) If LCL statement then
      i. Enter expansion time variable name in EVNTAB

ii. #EV := #EV+1

b) If a model statement then

i. If a label field contain a sequencing symbol then

If symbol is present in SSNTAB then

q := entry number in SSNTAB

else

enter symbol in SSNTAB[SSNTAB_PTR]

q := SSNTAB_PTR

SSNTAB_PTR := SSNTAB_PTR+1

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 :=MT_PTR+1

4. MEND statement

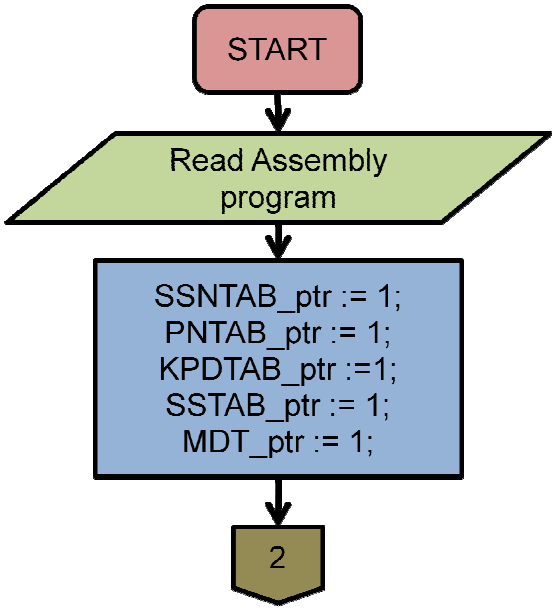
If SSNTAB+PTR=1 then

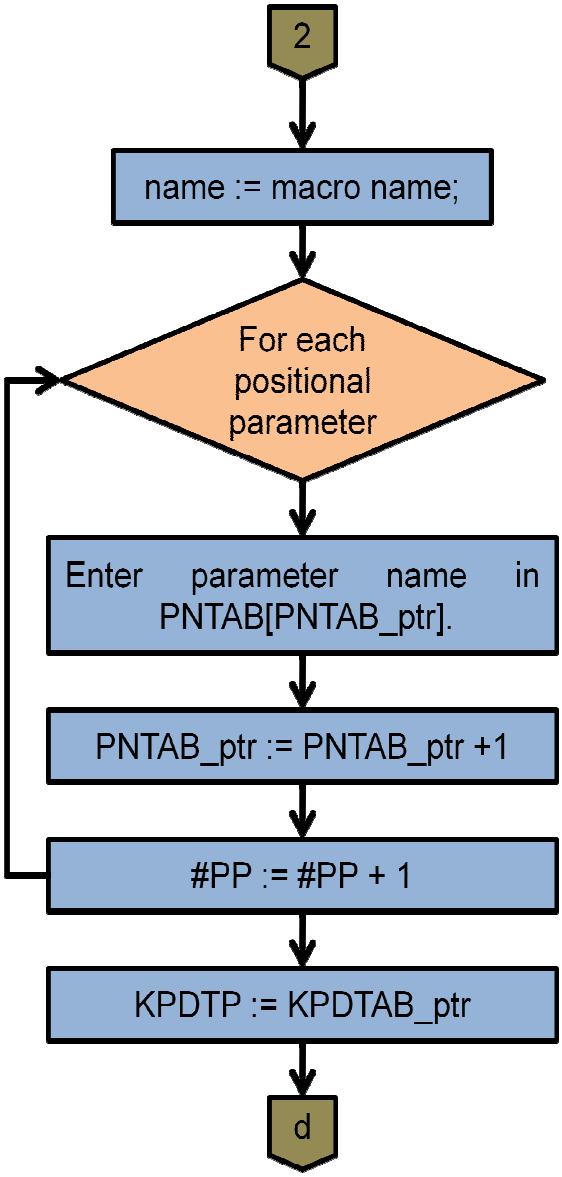
SSTP=0

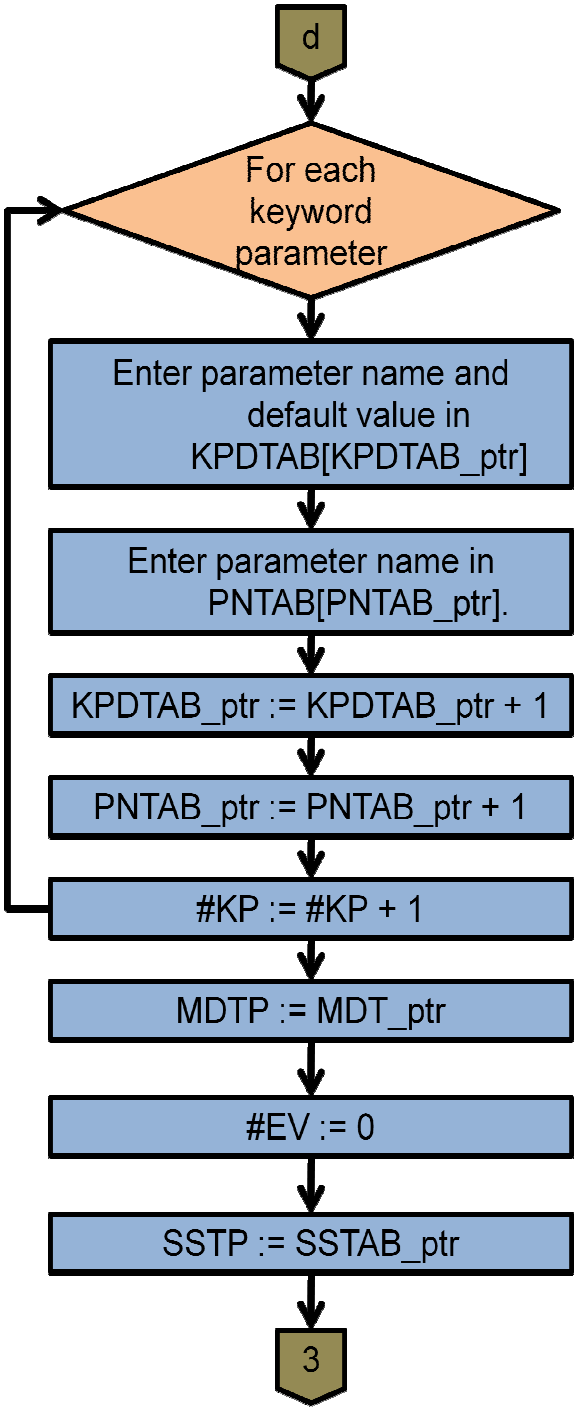
Else

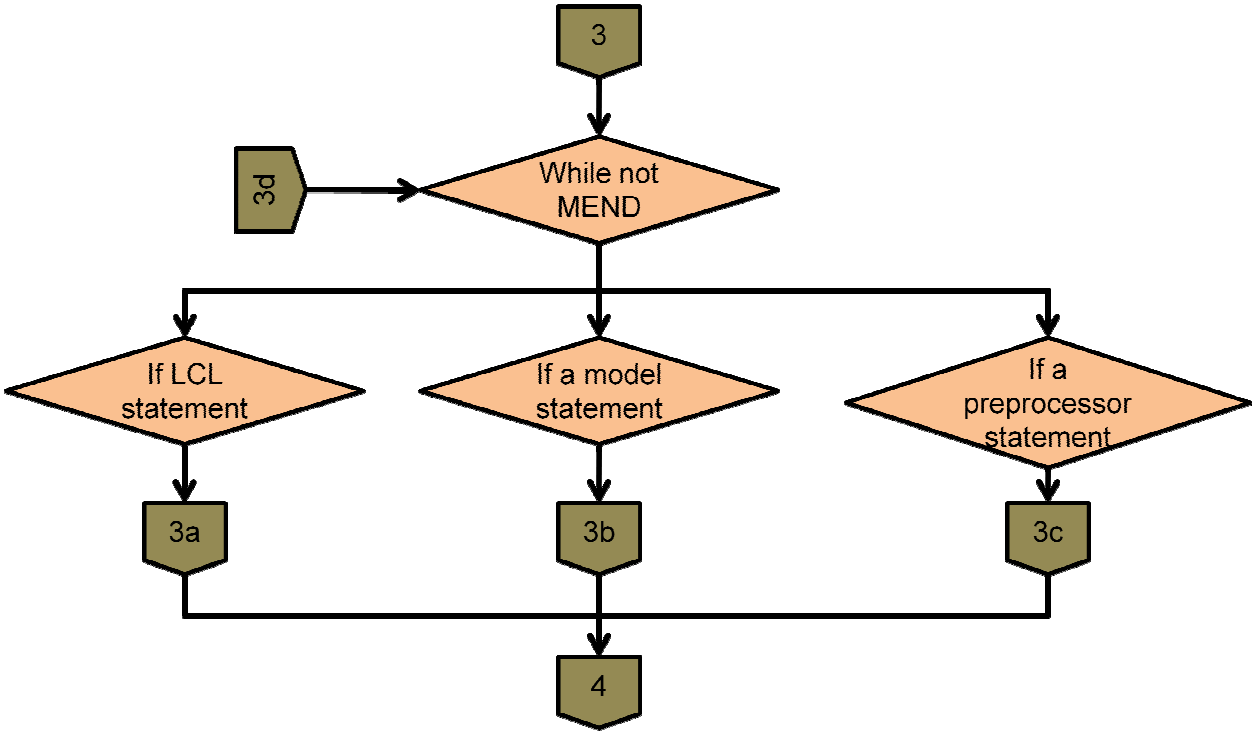
SSTAB_PTR = SSTAB_PTR + SSNTAB_PTR-1

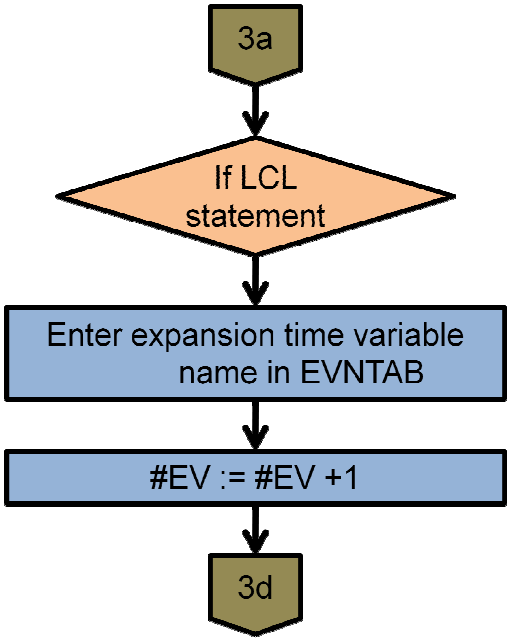
If #KP: = 0 then KPCTP: = 0

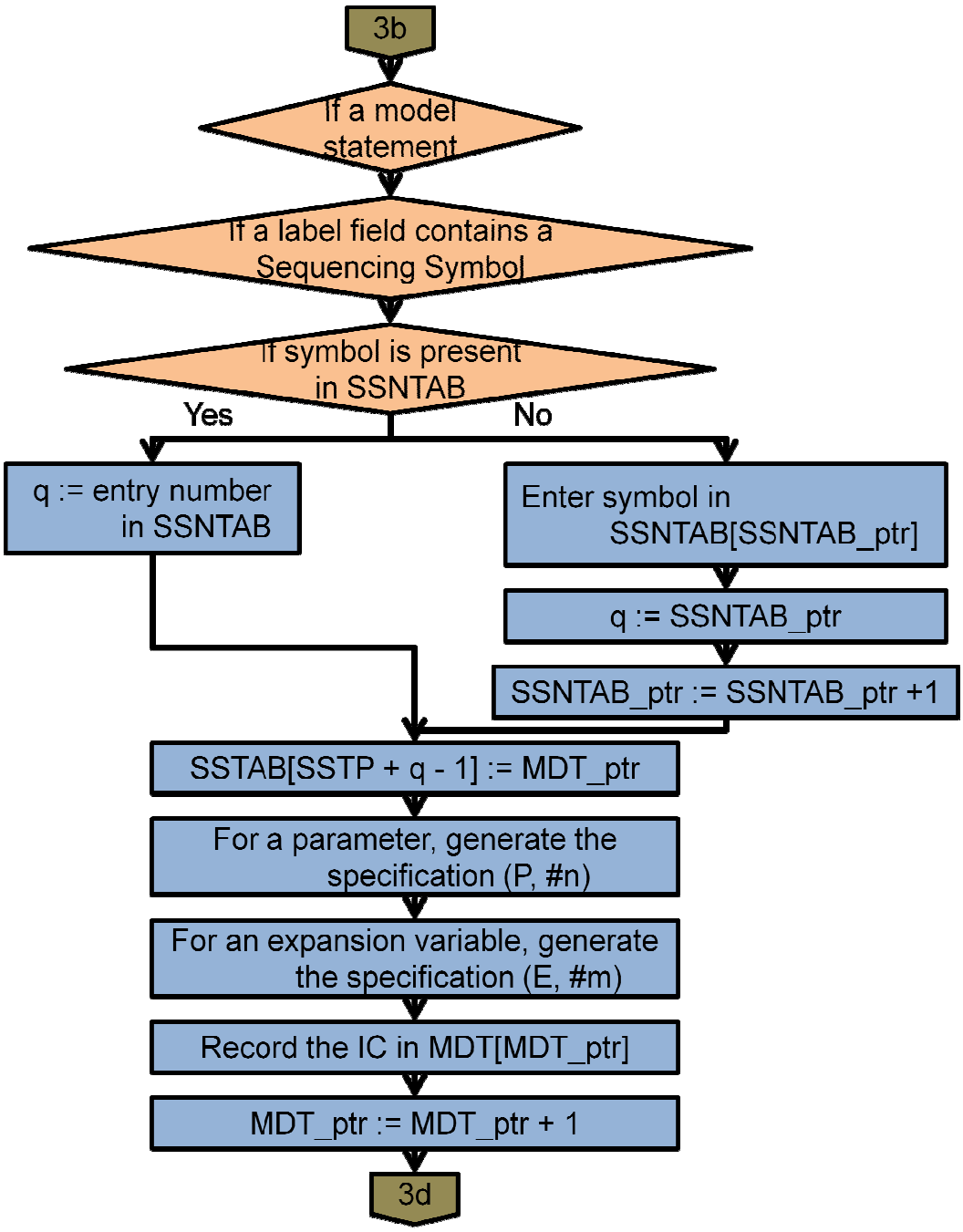

START
Read Assembly program
SSNTAB_ptr := 1;
PNTAB_ptr := 1;
KPDTAB_ptr :=1;
SSTAB_ptr := 1;
MDT_ptr := 1;
2

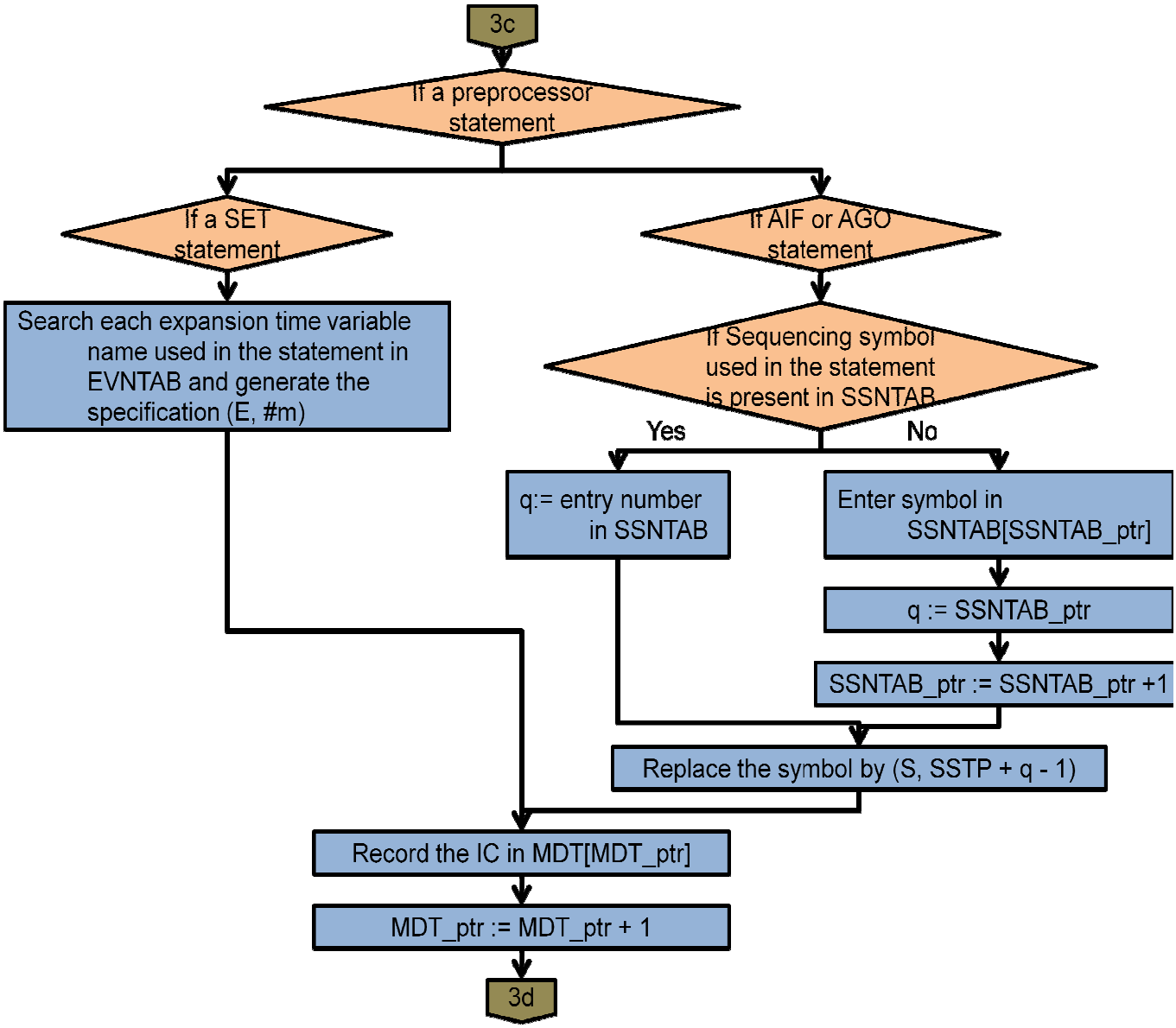

2
name := macro name;
For each positional parameter
Enter parameter name in PNTAB[PNTAB_ptr].
PNTAB_ptr := PNTAB_ptr +1
#PP := #PP + 1
KPDTP := KPDTAB_ptr
d

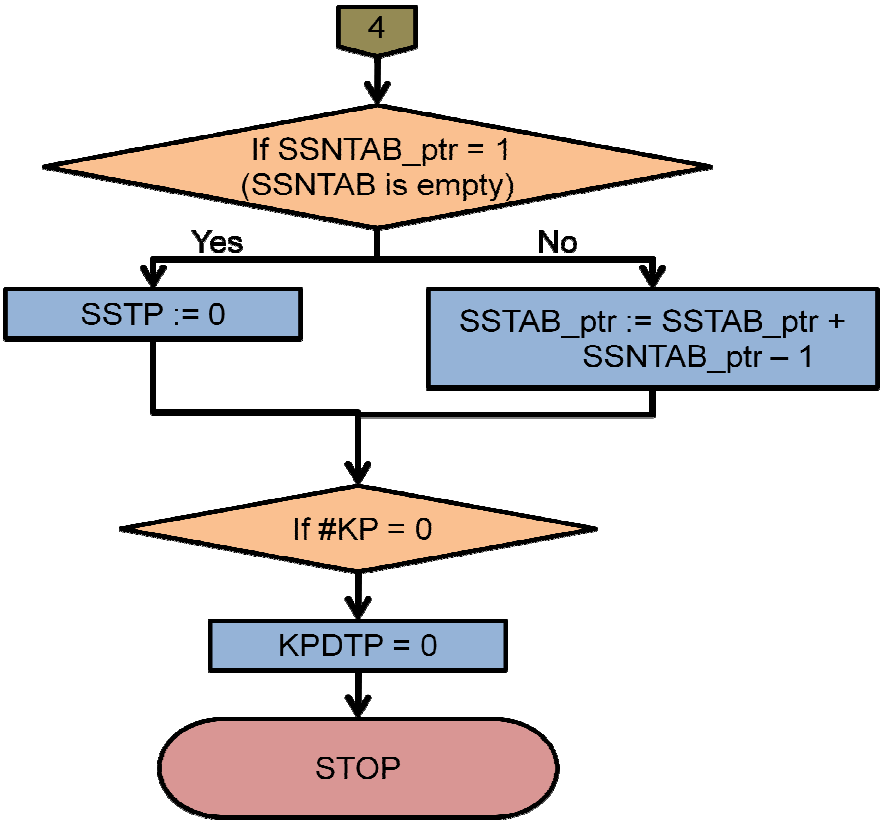

d
For each keyword parameter
Enter parameter name and default value in KPDTAB[KPDTAB_ptr]
Enter parameter name in PNTAB[PNTAB_ptr].
KPDTAB_ptr := KPDTAB_ptr + 1
PNTAB_ptr := PNTAB_ptr + 1
#KP := #KP + 1
MDTP := MDT_ptr
#EV := 0
SSTP := SSTAB_ptr
3


3
3d
While not MEND
If LCL statement
If a model statement
If a preprocessor statement
3a
3b
3c
4


3a
If LCL statement
Enter expansion time variable name in EVNTAB
#EV := #EV +1
3d


3b
If a model statement
If a label field contains a Sequencing Symbol
If symbol is present in SSNTAB
Yes
No
q := entry number in SSNTAB
Enter symbol in SSNTAB[SSNTAB_ptr]
q := SSNTAB_ptr
SSNTAB_ptr := SSNTAB_ptr +1
SSTAB[SSTP + q - 1] := MDT_ptr
For a parameter, generate the specification (P, #n)
For an expansion variable, generate the specification (E, #m)
Record the IC in MDT[MDT_ptr]
MDT_ptr := MDT_ptr + 1
3d


3c
If a preprocessor statement
If a SET statement
If AIF or AGO statement
Search each expansion time variable name used in the statement in EVNTAB and generate the specification (E, #m)
If Sequencing symbol used in the statement is present in SSNTAB
Yes
No
q:= entry number in SSNTAB
Enter symbol in SSNTAB[SSNTAB_ptr]
q := SSNTAB_ptr
SSNTAB_ptr := SSNTAB_ptr +1
Replace the symbol by (S, SSTP + q - 1)
Record the IC in MDT[MDT_ptr]
MDT_ptr := MDT_ptr + 1
3d


4
If SSNTAB_ptr = 1
(SSNTAB is empty)
Yes
No
SSTP := 0
SSTAB_ptr := SSTAB_ptr +
SSNTAB_ptr – 1
If #KP = 0
KPDTP = 0
STOP

ALGORITHM FOR 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 and set EVTAB-PTR
   c) Create APTEB with #PP+#KP entries and set APTAB_PTR
   d) Copt 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 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[KPDTP]….KPDTAB[KPDTP+#KP-1]. Let KPTDAB[q] contain a matching entry. Enter value of the keyword parameter in the call in APTAB[#PP+q-KPDTP+1].

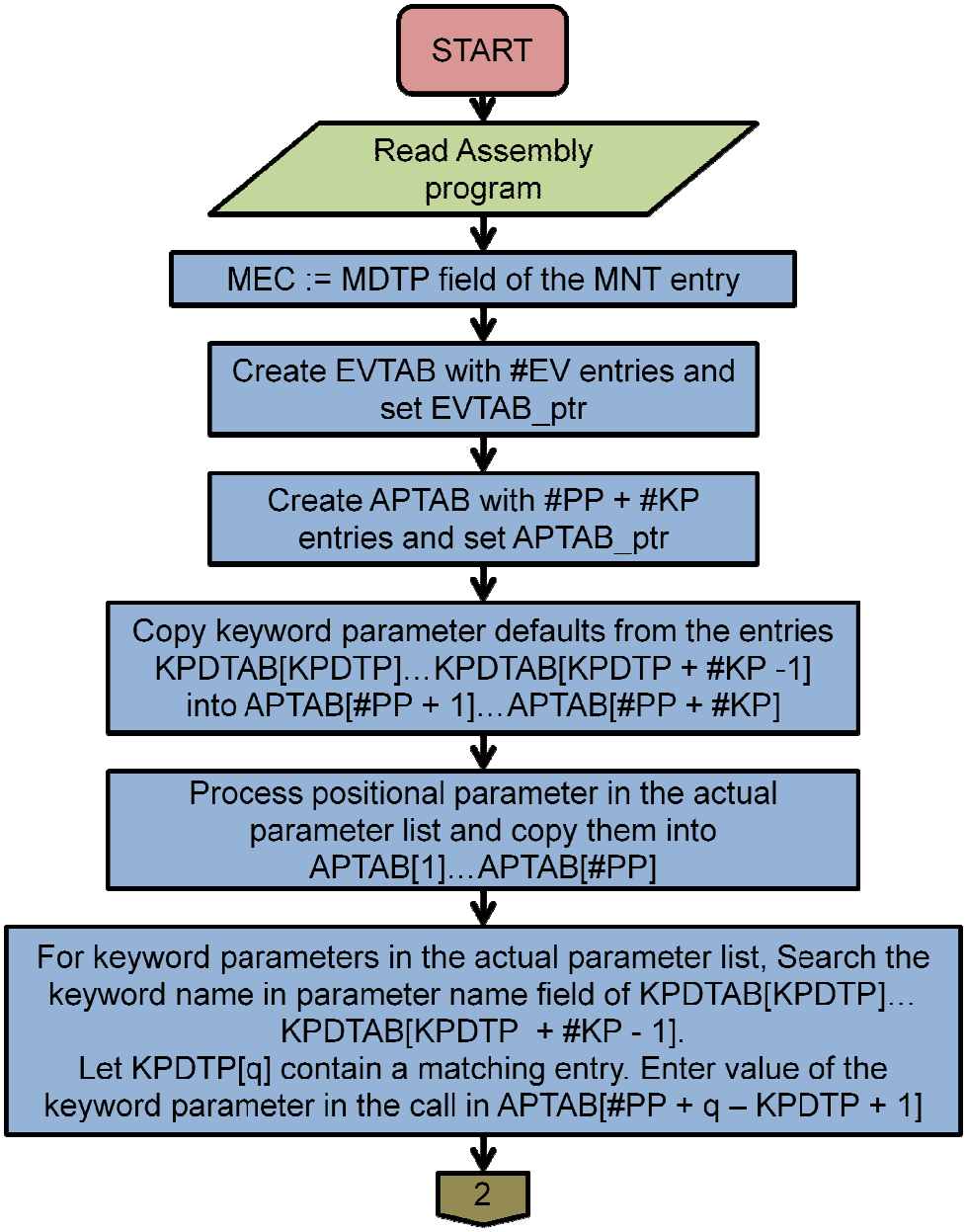
2. **While statement pointed by MEC is not MEND statement**
   a) If model statement then
      i. Replace the operands of the (P, #n) and (E , #m) by values in APTAB[n] and EVTAB[m] respectively.
      ii. Output the general statement.
      iii. MEC := MEC+1

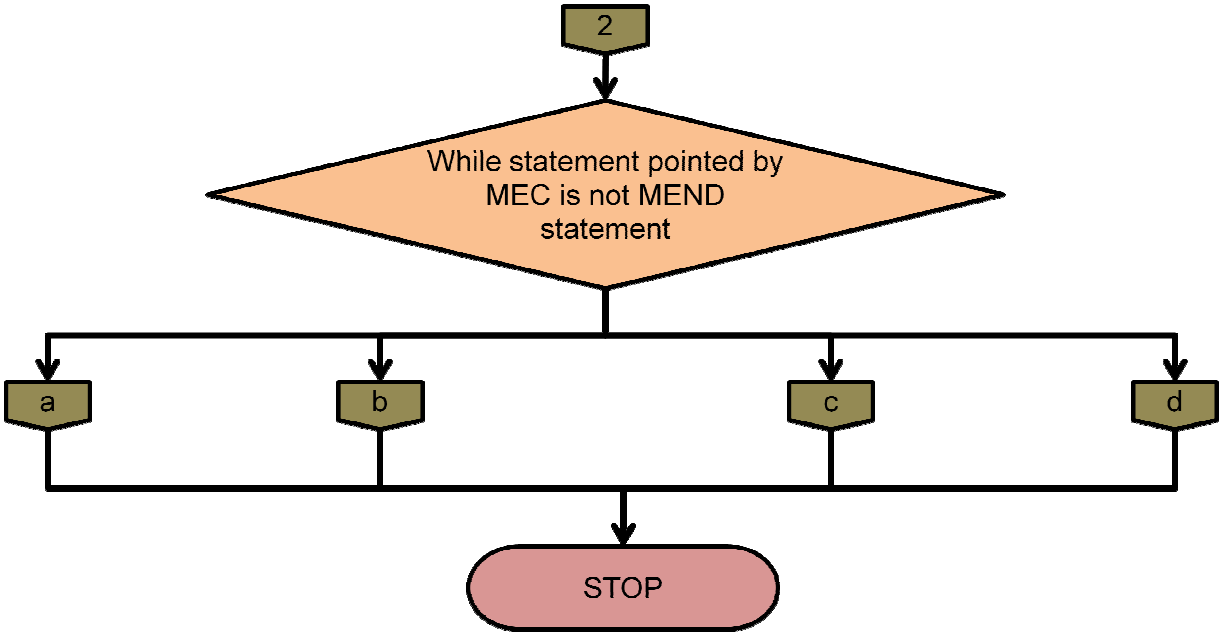
   b) If a SET statement with the specification (E, #m) in the label field then
      i. Evaluate the expression in the operand and set an appropriate value in EVTAB[m].
      ii. MEC:= MEC+1

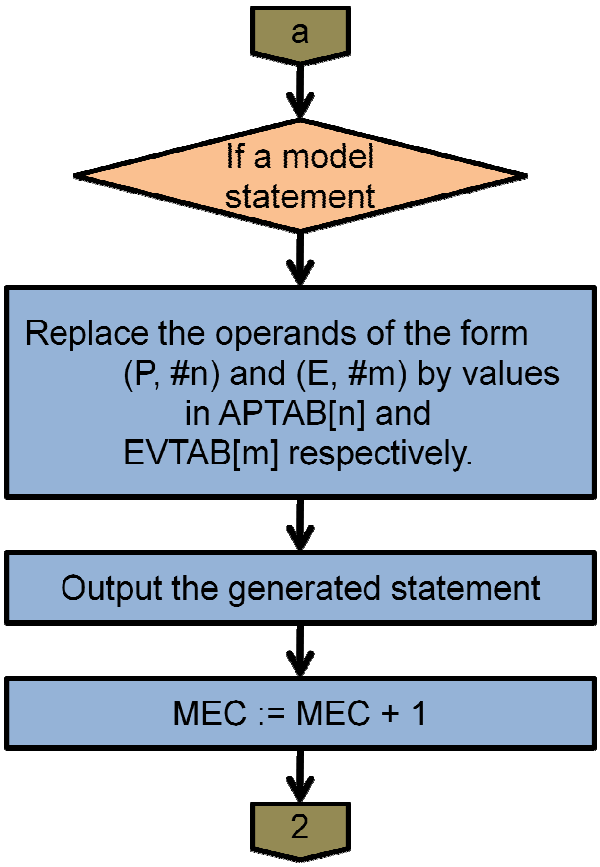
   c) In an AGO statement with (S, #s) in operand field then
      MEC := SSTAB[SSTP+s-1];

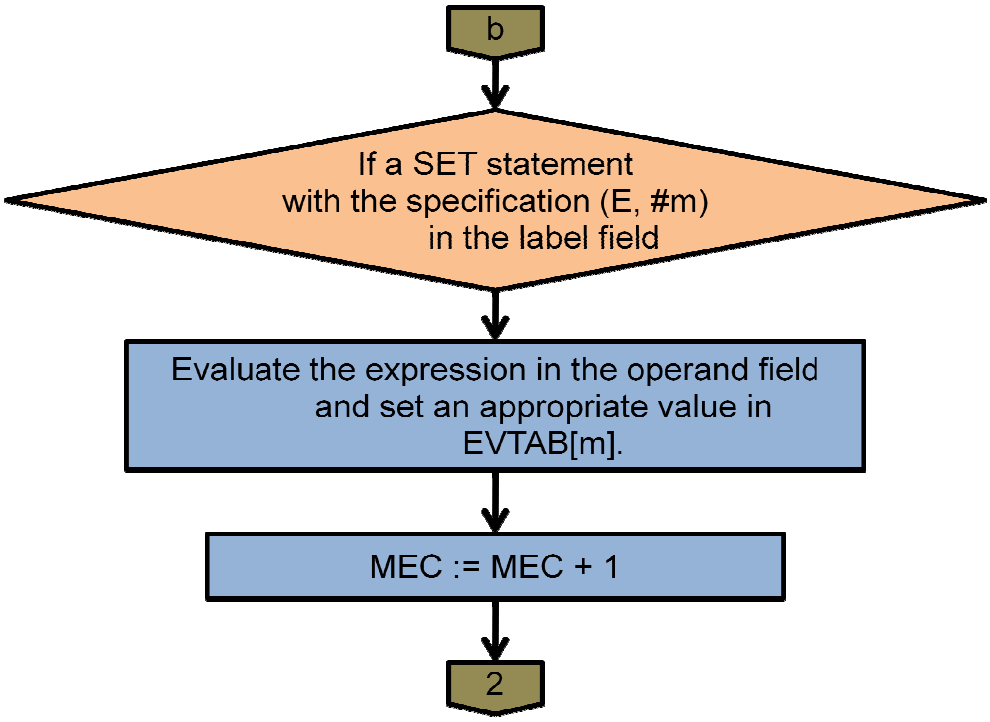
d) If an AIF statement with (S, #s) in operand field then
If condition in the AIF statement is true then
MEC := SSTAB[SSTP+s-1]

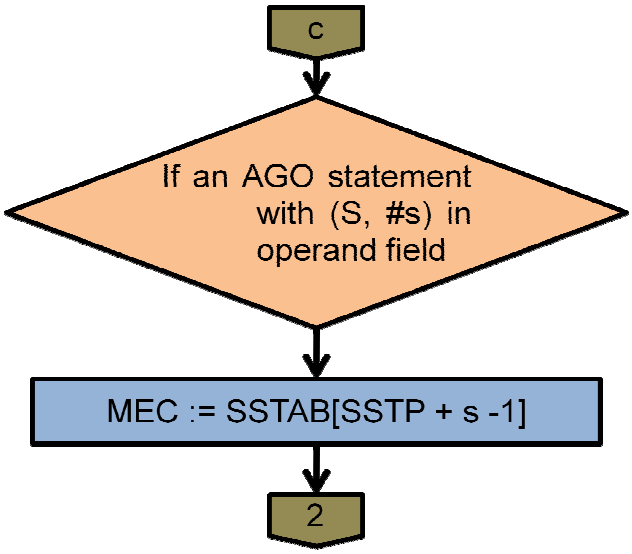
3. **Exit from macro expansion.**

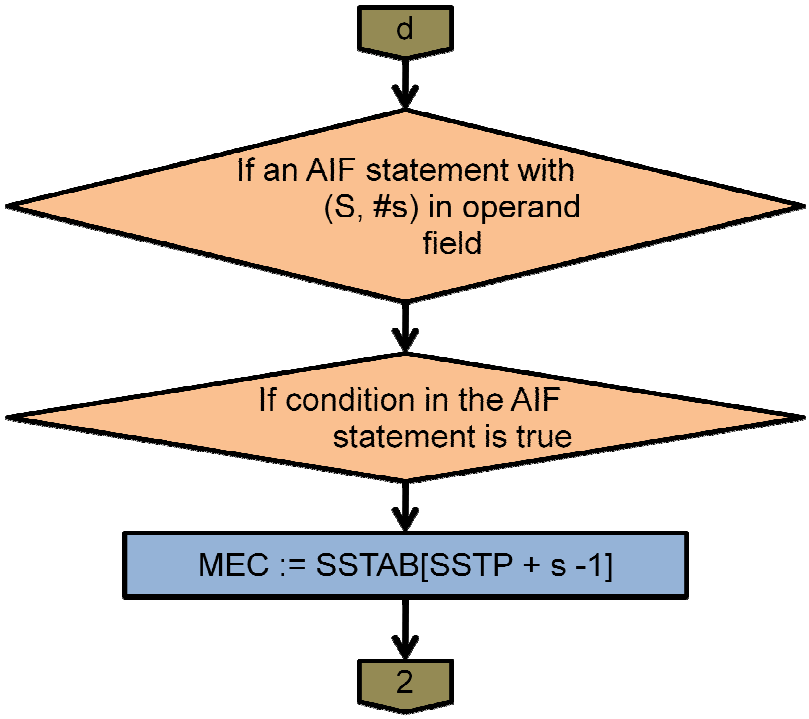

START
Read Assembly program
MEC := MDTP field of the MNT entry
Create EVTAB with #EV entries and set EVTAB_ptr
Create APTAB with #PP + #KP entries and set APTAB_ptr
Copy keyword parameter defaults from the entries KPDTAB[KPDTP]…KPDTAB[KPDTP + #KP -1] into APTAB[#PP + 1]…APTAB[#PP + #KP]
Process positional parameter in the actual parameter list and copy them into APTAB[1]…APTAB[#PP]
For keyword parameters in the actual parameter list, Search the keyword name in parameter name field of KPDTAB[KPDTP]… KPDTAB[KPDTP + #KP - 1].
Let KPDTP[q] contain a matching entry. Enter value of the keyword parameter in the call in APTAB[#PP + q – KPDTP + 1]
2


2
While statement pointed by MEC is not MEND statement
a
b
c
d
STOP


a
If a model statement
Replace the operands of the form (P, #n) and (E, #m) by values in APTAB[n] and EVTAB[m] respectively.
Output the generated statement
MEC := MEC + 1
2


b
If a SET statement
with the specification (E, #m)
in the label field
Evaluate the expression in the operand field
and set an appropriate value in
EVTAB[m].
MEC := MEC + 1
2


c
If an AGO statement with (S, #s) in operand field
MEC := SSTAB[SSTP + s -1]
2



3.5.3 DESIGN OF A MACRO ASSEMBLER

The use of macro preprocessor followed by a conventional assembler is an expensive way of handling macros. Since the number of passes over the source program is large and many functions get duplicated. For example, analysis of a source statement to detect macro calls requires us to process the mnemonic field. A similar function is required in the first pass of the assembler. Similar functions of preprocessor and the assembler can be merged if macros are handled by a macro assembler which performs macro expansion and program assembly simultaneously. This also reduces the number of passes.

Pass structure of a macro assembler

To design the pass structure of a macro assembler we identity the functions of a macro preprocessor and the conventional assembler which can be merged. After merging, the functions can be structured into passes of the macro assembler.

Pass I

1. Macro definition processing.
2. SYMTAB construction.

Pass II

1. Macro expansion.
2. Memory allocation and LC processing.
3. Processing of literals.
4. Intermediate code generation.

Pass III

1. Target code generation.

The pass structure can be simplified if attributes of actual parameters are not to be supported. The macro preprocessor would then be a single pass program. Integrating pass I of the assembler with preprocessor would give us the following two pass structure

Pass I

1. Macro definition processing.
2. Macro expansion.
3. Memory allocation, LC processing and SYMTAB construction.
4. Processing of literals.
5. Intermediate code generation.

Pass II

1. Target code generation.