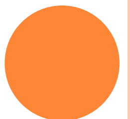
Macros and Macro Processors

By: Bhargavi H. Goswami
Assistant Professor
MCA Faculty
Sunshine Group of Institutions
Email: bhargavigoswami@gmail.com
Mob: +91 8140099018

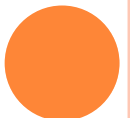
INTRODUCTION

- Macros are used to provide a program generation facility through macro expansion.
- Many languages provide build-in facilities for writing macros like PL/I, C, Ada AND C++.
- Assembly languages also provide such facilities.
- When a language does not support build-in facilities for writing macros what is to be done?
- A programmer may achieve an equivalent effect by using generalized preprocessors or software tools like Awk of Unix.



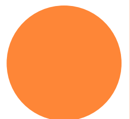
A MACRO

- **Def: 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 a 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 can be identified.



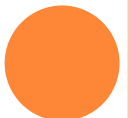
Classification of macros:

- Lexical expansion:
  - Lexical expansion implies replacement of a character string by another character string during program generation.
  - Lexical expansion is to replace occurrences of formal parameters by corresponding actual parameters.
- Semantic expansion:
  - Semantic expansion implies generation of instructions tailored to the requirements of a specific usage.
  - Semantic expansion is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.
  - Eg: Generation of type specific instructions for manipulation of byte and word operands.



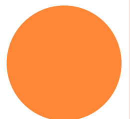
EXAMPLE

- The following sequence of instructions is used to increment the value in a memory word by a constant.
  - 1. Move the value from the memory word into a machine-register.
  - 2. Increment the value in the machine register.
  - 3. Move the new value into the memory word.
- Since the instruction sequence MOVE-ADD-MOVE may be used a number of times in a program, it is convenient to define a macro named INCR.
- Using Lexical expansion the macro call INCR A,B,AREG can lead to the generation of a MOVE-ADD-MOVE instruction sequence to increment A by the value of B using AREG to perform the arithmetic.
- Use of Semantic expansion can enable the instruction sequence to be adapted to the types of A and B.
- For example an INC instruction could be generated if A is a byte operand and B has the value '1'.



How does macro differ from subroutine?

- Macros differ from subroutines in one fundamental respect.
- Use of a macro name in the mnemonic field of an assembly statement leads to its expansion,
- whereas use of subroutine name in a call instruction leads to its execution.
- So there is difference in
  - Size
  - Execution Efficiency
- Macros can be said to trade program size for execution efficiency.
- More difference would be discussed at the time of discussion of macro expansion.

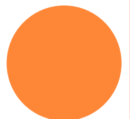


MACRO DEFINITION AND CALL

- 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.
- A macro definition consists of.
  - A macro prototype statement
  - One or more model statements
  - Macro preprocessor statements
- The macro prototype statement declares the name of a macro and the names and kinds of its parameters.
- It has the following syntax

  <macro name> [< formal parameter spec > [,..]]
- Where <macro name> appears in the mnemonic field of an assembly statement and
- < formal parameter spec> is of the form
- &<parameter name> [<parameter kind>]

- Open your book and see example 5.2 on pg 133.

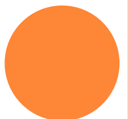


Macro Call

- A macro is called by writing the macro name in the mnemonic field.
- Macro call has the following syntax.

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

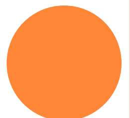
- Where an actual parameter resembles an operand specification in an assembly language statement.



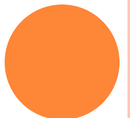
EXAMPLE

- MACRO and MEND are the macro header and macro end statements.
- The prototype statement indicates that three parameters called
  - MEM_VAL,
  - INCR_VAL and
  - REG exists

  for the macro.
- Since parameter kind is not specified for any of the parameters, they are all of the default kind 'positional parameter'.
- Statements with the operation codes MOVER, ADD and MOVEM are model statements.
- No preprocessor statements are used in this macro.

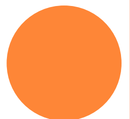


```
MACRO
INCR      &MEM_VAL, &INCR_VAL, &REG
MOVER     &REG, &MEM_VAL
ADD       &REG, &INCR_VAL
MOVEM     &REG, &MEM_VAL
MEND
```



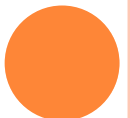
MACRO EXPANSION

- Macro call leads to macro expansion.
- During macro expansion, the macro call statement is replaced by a sequence of assembly statements.
- How to differentiate between 'the original statements of a program' and 'the statements resulting from macro expansion' ?
- Ans: Each expanded statement is marked with a '+' preceding its label field.
- Two key notions concerning macro expansion are
  - A. Expansion time control flow : This determines the order in which model statements are visited during macro expansion.
  - B. Lexical substitution: Lexical substitution is used to generate an assembly statement from a model statement.



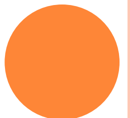
A. Expansion time control flow

- 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.
- What can alter the flow of control during expansion?
- A preprocessor statement can alter the flow of control during expansion such that
  - Conditional Expansion: some model statements are either never visited during expansion, or
  - Expansion Time Loops: are repeatedly visited during expansion.
- The flow of control during macro expansion is implemented using a macro expansion counter (MEC)



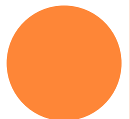
ALGOTIRHM (MACRO EXPANSION)

- 1. MEC:=statement number of first statement following the prototype stmt.
- 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.



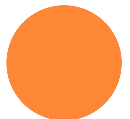
B. Lexical Substitution

- A model statement consists of 3 types of strings.
  - An ordinary string, which stands for itself.
  - The name of a formal parameter which is preceded by the character ‘&’.
  - The name of a preprocessor variable, which is also preceded by the character ‘&’.
- During lexical expansion, strings of type 1 are retained without substitution.
- String of types 2 and 3 are replaced by the ‘values’ of the formal parameters or preprocessor variables.
- Rules for determining the value of a formal parameter depends on the kind of parameter:
  - Positional Parameter
  - Keyword Parameter
  - Default specification of parameters
  - Macros with mixed parameter lists
  - Other uses of parameter



Positional Parameters

- A positional formal parameter is written as &<parameter name>,
- e.g. &SAMPLE
- where SAMPLE is the name of parameter.
- <parameter kind> of syntax rule is omitted.
- The value of a positional formal parameter XYZ is determined by the rule of positional association as follows:
- Find the ordinal position of XYZ in the list of formal parameters in the macro prototype statement.
- Find the actual parameter specification occupying the same ordinal position in the list of actual parameters in the macro call statement.

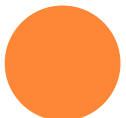


EXAMPLE

- Consider the call:

  INCR A,B,AREG

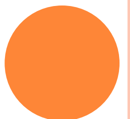
- On macro INCR, following rule of positional association, values of formal parameters are:
- Formal parameter value
- MEM_VAL A
- INCR_VAL B
- REG AREG
- Lexical expansion of the model statements now leads to the code
- \+ MOVER AREG,A
- \+ ADD AREG,B
- \+ MOVEM AREG,A



Keyword Parameter

- For keyword parameter,
  - <parameter name> is an ordinary string and
  - <parameter kind> is the string '='

  in syntax rule.
- The <actual parameter spec> is written as <formal parameter name>=<ordinary string>.
- Note that 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.
- Let us see example for it.



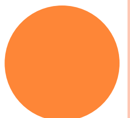
EXAMPLE:

- Following are macro call statement:

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

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

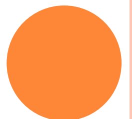
- Both are equivalent.
- Following is macro definition using keyword parameter:
- MACRO
- INCR_M &MEM_VAL=, &INCR_VAL=,®=
- MOVER ®, &MEM_VAL
- ADD   ®, &INCR_VAL
- MOVEM ®,&MEM_VAL
- MEND



DEFAULT SPECIFICATIONS OF PARAMETERS

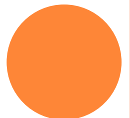
- A default value is a standard assumption in the absence of an explicit specification by the programmer.
- Default specification of parameters is useful in situations where a parameter has the same value in most calls.
- When the desired value is different from the default value, the desired value can be specified explicitly in a macro call.
- The syntax for formal parameter specification, as follows:

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



EXAMPLE

- The macro can be redefined to use a default specification for the parameter REG
- INCR_D MEM_VAL=A, INCR_VAL=B
- INCR_D INCR_VAL=B, MEM_VAL=A
- INCR_D INCR_VAL=B, MEM_VAL=A, REG=BREG
- First two calls are equivalent but third call overrides the default value for REG with the value BREG in next example. Have a look.
- MACRO
- INCR_D &MEM_VAL=, &INCR_VAL=, ®=AREG
- MOVER ®, &MEM_VAL
- ADD ®, &INCR_VAL
- MOVEM ®, &MEM_VAL
- MEND

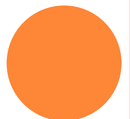


Macros with mixed parameter lists

- A macro may be defined to use both positional and keyword parameters.
- In such a case, all positional parameters must precede all keyword parameters.
- example in the macro call

SUMUP A, B, G=20, H=X

- A, B are positional parameters while G, H are keyword parameters.
- Correspondence between actual and formal parameters is established by applying the rules governing positional and keyword parameters separately.



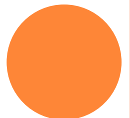
OTHER USES OF PARAMETERS

- The model statements have used formal parameters only in operand field.
- However, use of parameters is not restricted to these fields.
- Formal parameters can also appear in the label and opcode fields of model statements.
- Example:

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

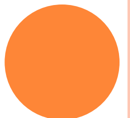
- Expansion of the call CALC A, B, LAB=LOOP leads to the following code:

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



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 the nested call is the outer macro and,
- Macro called is inner macro.
- They follow LIFO rule.
- Thus, in structure of nested macro calls, expansion of latest macro call (i.e inner macro) is completed first.

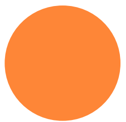


EXAMPLE:

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

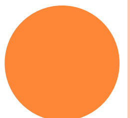
- MACRO
- COMPUTE &FIRST, &SECOND
- MOVEM BREG, TMP
- INCR_D &FIRST, &SECOND, REG=BREG
- MOVER BREG, TMP
- MEND

- COMPUTE X,Y:
  - + MOVEM BREG, TMP [1]
  - + INCR_D X,Y
    - + MOVER BREG,X [2]
    - + ADD BREG,Y [3]
    - + MOVEM BREG,X [4]
  - + MOVER BREG,TMP [5]



Advanced Macro Facilities

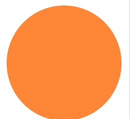
- Advanced macro facilities are aimed to supporting semantic expansion.
- Used for:
  - performing conditional expansion of model statements and
  - in writing expansion time loops.
- These facilities can be grouped into following.
  - 1. Facilities for alteration of flow of control during expansion.
  - 2. Expansion time variables.
  - 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.
- 1. Expansion time sequencing symbol
- 2. Expansion time statements
  - AIF,
  - AGO and
  - ANOP.
- A sequencing symbol (SS) has the syntax

  .<ordinary string>
- As SS is defined by putting it in the label field of 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.

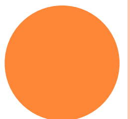


AIF STATEMENT

- An AIF statement has the syntax

  AIF (<expression>) <sequencing symbol>

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

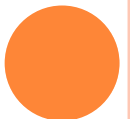


An AGO statement

- An AGO statement has the syntax

    AGO <sequencing symbol>

- Unconditionally transfers expansion time control to the statement containing <sequencing symbol> in its label field.

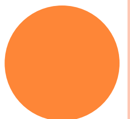


AN ANOP STATEMENT

- An ANOP statement is written as

  <sequencing symbol>   ANOP

- And simply has the effect of defining the sequencing symbol.

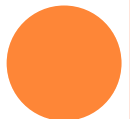


2. Expansion Time Variable

- 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 and global EV's are created through declaration statements with the following syntax:

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

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

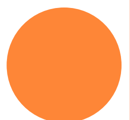


- <EV specification> has the syntax

  &<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

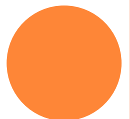
  <EV specification> SET <SET-expression>
  - Where <EV specification> appears in the label field and
  - SET in the mnemonic field.
- A SET statement assigns the value of <SET-expression> to the EV specified in <EV specification>.
- The value of an EV can be used in any field of a model statement, and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



EXAMPLE

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

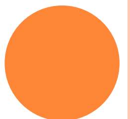
- A call on macro CONSTANTS is expanded as follows.
- The local EV A is created.
- The first SET statement assigns the value ‘1’ to it.
- The first DB statement thus declares a byte constant ‘1’.
- The second SET statement assigns the value ‘2’ to A
- And the second DB statement declares a constant ‘2’.



3. Attributes of formal parameters

- An attribute is written using the syntax

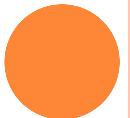
  <attribute name>' <formal parameter spec>
- And represents information about the value of the formal parameter,
- i.e. about the corresponding actual parameter.
- The type, length and size attributes have the names T,L and S



EXAMPLE

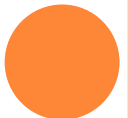
- Here expansion time control is transferred to the statement having .NEXT in its label field only if the actual parameter corresponding to the formal parameter A has the length of ‘1’.

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



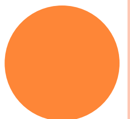
Conditional Expansion

- While writing a general purpose macro it is important to ensure execution efficiency of its generated code.
- This is achieved by ensuring that a model statement is visited only under specific conditions during the expansion of a macro.
- How to do that?
- Ans: The AIF and AGO statements are used for this purpose.
- Let us take example which would clear our doubts for the same.

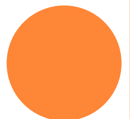


Example: A-B+C

```
                ONLY        ANOP
                OVER        ANOP
                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
```

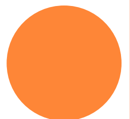


- It is required to develop a macro EVAL such that a call EVAL A,B,C generates efficient code to evaluate A-B+C in AREG.
- When the first two parameters of a call are identical, EVAL should generate single MOVER instruction to load 3rd parameter into AREG.
- As formal parameter is corresponding to actual parameter, AIF statement effectively compares names of first two actual parameters.
- If condition is true, expansion time control is transferred to model statement MOVER AREG, &Z.
- If false, MOVE-SUB-ADD sequence is generated and expansion time control is transferred to statement .OVER MEND which terminates expansion.
- Thus, efficient code is generated under all conditions.



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:
- 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 MOVER statement puts the value '0' in AREG, while the three MOVEM statements store this value in 3 consecutive bytes with the addresses B, B+1 and B+2.

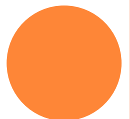


- Alternatively, the same effect can be achieved by writing an expansion time loop which visits a model statement, or a set of model statement repeatedly during macro expansion.
- Expansion time loops can be written using expansion time variables (EV's) and expansion time control transfer statements AIF and AGO.
- Consider expansion of the macro call

  CLEAR B, 3

- Example

```
              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
```

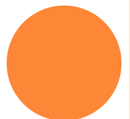


Other Facilities for Expansion Time Loops

- The assembler for M 68000 and Intel 8088 processors provide explicit expansion time looping constructs.
- <expression> should evaluate to a numerical value during macro expansion.
- The **REPT** statement

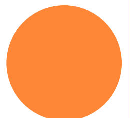
  REPT <expression>
- Statements between **REPT** and an ENDM statement would be processed for expansion <expression> number of times.
- Following example use REPT to declare 10 constant with the value 1,2,…10.

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



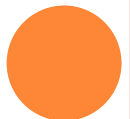
The IRP Statement

- IRP <formal parameter> <argument list>
- The formal parameter mentioned in the statement takes successive values from the argument list.
- For each value, the statements between the IRP and ENDM statements are expanded once.
- 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 value 4, 7 and 10.



Semantic Expansion

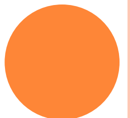
- Semantic expansion is the generation of instructions to the requirements of a specific usage.
- It can be achieved by a combination of advanced macro facilities like AIF, AGO statements and expansion time variables.
- The CLEAR example is an instance of semantic expansion. In this example the number of MOVEM AREG,….. statement generated by a call on CLEAR is determined by the value of the second parameter of CLEAR.
- Following example is another instance of conditional expansion wherein one of two alternative code sequences is generated depending on actual parameters of a macro call.



Example

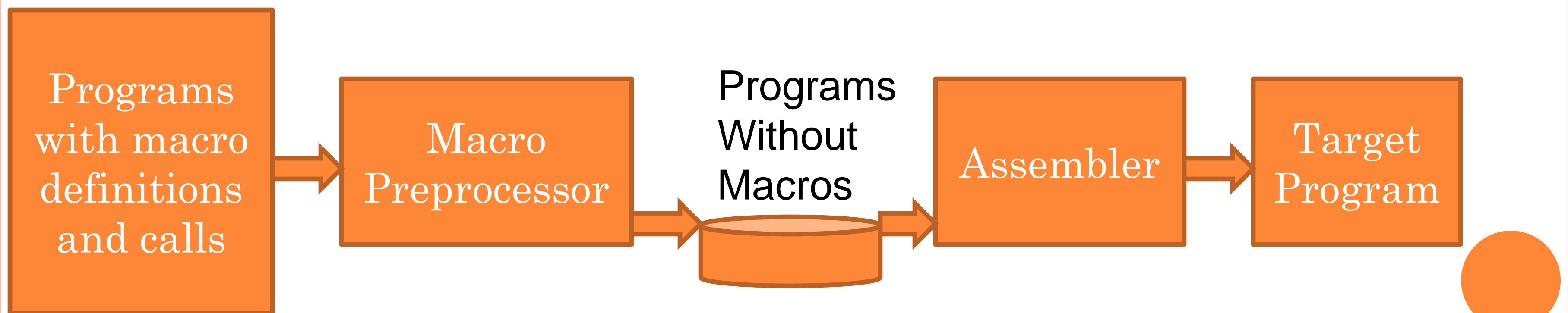
- This macro creates a constant '25' with the name given by the 2nd parameter.
- The type of the constant matches the type of the first parameter.

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



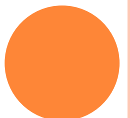
Design of a Macro Preprocessor

- The macro preprocessor accepts an assembly program containing definitions and calls and translates it into an assembly program which does not contain any macro definitions and calls.
- The program form output by the macro preprocessor can be handed over to an assembler to obtain the target program.



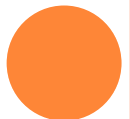
Design Overview

- We begin the design by listing all 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.
- Following 4 step procedure is followed to arrive at a design specification for each task.
  - Identify the information necessary to perform a task.
  - Design a suitable data structure to record the information.
  - Determine the processing necessary to obtain the information.
  - Determine the processing necessary to perform the task.



1. Identify Macro Calls

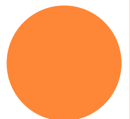
- A table called the Macro Name Table (MNT) is designed to hold the names of all macros defined in a program.
- A macro name is entered in this table when macro definition is processed.
- While processing a statement in the source program, the preprocessor compares the string found in its mnemonic field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



2. Determine the 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 the table is a pair (<formal parameter name>, <value>).
- Two items of information are needed to construct this table, names of formal parameters and default values of keyword parameters.
- A table called the Parameter Default Table (PDT) is used for each macro.
- It would contain pairs of the form

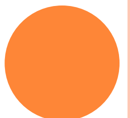
  (<formal parameter name>, <default value>)
- If a macro call statement does not specify a value for some parameter par, its default value would be copied from PDT to APT.



3. Maintain expansion time variables

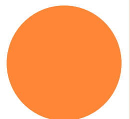
- An Expansion time Variable Table (EVT) is maintained for this purpose.
- The table contains pairs of the form

  (<EV name>, <value>).
- The value field of a pair is accessed when a preprocessor statement or a model statement under expansion refers to an EV.



4. Organize expansion time control flow

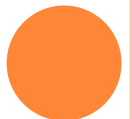
- The body of a macro, i.e. the set of preprocessor statements and model statements in it, is stored in a table called the Macro Definition Table (MDT) for use during macro expansion.
- The flow of control during macro expansion determines when a model statement is to be visited for expansion.
- For this purpose MEC (Macro Expansion Counter) is initialized to the first statement of the macro body in the MDT.
- It is updated after expanding a model statement of on processing a macro preprocessor statement.



5. Determine values of sequencing symbols

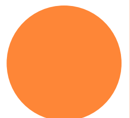
- A Sequencing Symbols Table (SST) is maintained to hold this information.
- The table contains pairs of the form

  (<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 (for back reference to symbol) or on encountering a reference prior to its definition(forward reference).



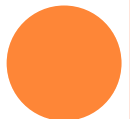
6. Perform expansion of a model statement

- This is trivial task given the following:
  - 1. MEC points to the MDT entry containing the model statement.
  - 2. Values of formal parameters 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 a model statement is achieved by performing a lexical substitution for the parameters and EV's used in the model statement.

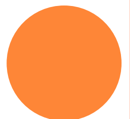


DATA STRUCTURE

- The tables APT, PDT and EVT contain pairs which are searched using the first component of the pair as a key.
- For example 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 the table is known when its value is to be accessed.
- In context of APT, the value of a formal parameter ABC is needed while expanding a model statement using it.
- 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 the MDT, where (P,5) stands for the words "parameter #5".
- Thus macro expansion can be made more efficient by storing an intermediate code for a statement in the MDT.

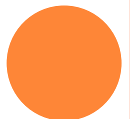


- All the parameter names could be replace by pairs of the form (P,n) in model statements and preprocessor statements stored in MDT.
- The information (P,5) appearing in a model statement is sufficient to access the value of formal parameter ABC. Hence APT containing (<formal parameter name> , <value>) is replace by another table called APTAB which only contains <value>'s.
- To implement this, ordinal numbers are assigned to all parameters of a macro.
- A table named Parameter Name Table (PNTAB) is used for this purpose. PNTAB is used while processing the definition of a macro.
- Parameter names are entered in PNTAB in the same order in which they appear in the prototype statement.

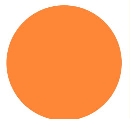


- Its entry number is used to replace the parameter name in the model and preprocessor statements of the macro while storing it in the MDT.
- This implements the requirement that the statement MOVER AREG, &ABC should appear as MOVER

  AREG, (P,5) in MDT.
- In effect, the information (<formal parameter name>,<value>) in APT has been split into two table
  - PNTAB which contains formal parameter names.
  - APTAB which contains formal parameter values.
- PNTAB is used while processing a macro definition while APTAB is used during macro expansion.

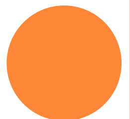


- Similar Analysis leads to splitting
  - EVT into EVNTAB and EVTAB.
  - SST into SSNTAB and SSTAB.
- EV names are entered into EVNTAB while processing EV declaration statements.
- SS names are entered in SSNTAB while processing an SS reference or definition, whichever occurs earlier.
- Entries only need to exist for default parameter, therefore we replace the parameter default table (PDT) by a keyword parameter default table (KPDTAB).
- We store the number of positional parameters of macro in a new field of the MNT entry.
- MNT has entries for all macros defined in a program.
- Each MNT entry contains three pointers MDTP, KPDTP and SSTP, which are pointers to MDT, KPDTAB and SSNTAB.
- Instead of creating different MDT's for different macros, we can create a single MDT and use different sections of this table for different macros.



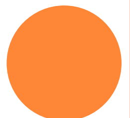
Tables are constructed for macro preprocessor.

Table	Fields
MNT (Macro Name Table)	Macro Name
	Number of Positional Parameter (#PP)
	Number of keyword parameter (#KP)
	Number of Expansion Time Variable (#EV)
	MDT pointer (MDTP)
	KPDTAB pointer (KPDTABP)
	SSTAB pointer (SSTP)



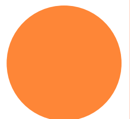
(CONTI.....) TABLES ARE CONSTRUCTED FOR MACRO PREPROCESSOR.

Tables	Fields
PNTAB (Parameter Name Table)	Parameter name
EVNTAB (EV Name Table)	EV Name
SSNTAB (SS Name Table)	SS Name
KPDTAB (Keyword Parameter Default Table)	Parameter name, default value
MDT (Macro Definition Table)	Label, Opcode, Operands Value
APTAB (Actual Parameter Table)	Value
EVTAB (EV Table)	Value
SSTAB (SS Table)	MDT entry #



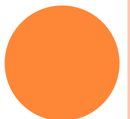
Construction and use of the macro preprocessor data structures can be summarized as follows.

- PNTAB and KPDTAB are constructed by processing the prototype statement.
- Entries are added to EVNTAB and SSNTAB as EV declarations and SS definitions/references are encountered.
- MDT are constructed while processing the model statements and preprocessor statements in the 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 a macro.
- See Pg.151, Fig 5.8.



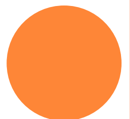
Processing of Macro Definitions

- The following initializations are performed before initiating the processing of macro definitions in a program
- KPDTAB_pointer:=1;
- SSTAB_ptr:=1;
- MDT_ptr:=1;
- Now let us see the algorithm which is invoked for every macro definition.

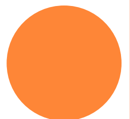


ALGORITHM (PROCESSING OF A MACRO DEFINITION)

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 and 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:=SSTAB_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 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:=MDT_ptr + 1;



c. If a preprocessor statement then

i. If a SET statement

Search each expansion time variable name used in the statement in EVNTAB and generate the spec (E,#m).

ii. If an AIF or AGO statement then

If sequencing symbol used in the statement is present in SSNTAB

Then

q:=entry number in SSNTAB;

else

Enter symbol in SSNTAB[SSNTAB_ptr].

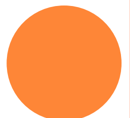
q:=SSNTAB_ptr;

SSNTAB_ptr:=SSNTAB_ptr+1;

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)

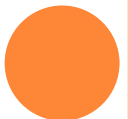
then

SSTP:=0;

Else

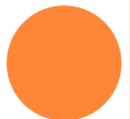
SSTAB_ptr:=SSTAB_ptr+SSNTAB_ptr-1;

If #KP=0 then KPDTP=0;



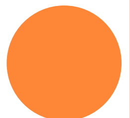
Macro Expansion

- We use the following data structure to perform macro expansion
  - APTAB Actual Parameter Table
  - EVTAB EV Table
  - MEC Macro expansion counter
  - APTAB_ptr APTAB pointer
  - EVTAB_ptr EVTAB pointer
- The number of entry in APTAB equals the sum of values in the #PP and #KP fields of the MNT entry of
- macro.
- Number of entries in EVTAB is given by the value in #EV field of the MNT.
- APTAB and EVTAB are constructed when a macro call is recognized.
- APTAB_ptr and EVTAB_ptr are set to point at these tables.
- MEC always pointers to the next statement to be expanded.
- For data structure, pl see Fig. 5.9 which explains Data Structure.



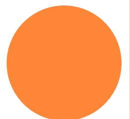
Algorithm (Macro Expansion)

1. Perform initializations for the expansion of a macro.
   a. MEC:=MDTP field of the MNT entry.
   b. Create EVTAB with #EV entries and set EVTAB_ptr.
   c. Create APTAB with #PP+#KP entries and 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[KPDTP]…KPDTAB[KPDTP+#KP-1].
      Let KPDTAB[q] contain a matching entry.
      Enter value of the keyword parameter in the call (if any) in APTAB[#PP+q-KPDTP+1].



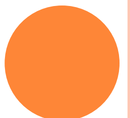
2. While statement pointed by MEC is not MEND statement
   a. If a model statement then
      i. Replace operands of the form (P,#n) and (E,#m) by values in APTAB[n] and EVTAB[m] respectively.
      ii. Output the generated statement.
      iii. MEC:=MEC+1;
   b. If a SET statement with the specification (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 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 the macro expansion.

- See Fig 5.9 on page 154 explaining DS for Macro Expansion.



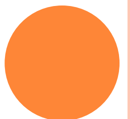
Nested Macro Calls

- Macro calls appearing in the source program have been expanded but statements resulting from the expansion may themselves contain macro calls.
- The macro expansion can be applied until we get the code form which does not contain any macro call statement.
- Such expansion requires a number of passes of macro expansion.
- To increase the efficiency, another alternative would be to examine each statement generated during macro expansion to see if it is itself a macro call.
- If so, provision can be made to expand this call before continuing with the expansion of the parent macro call.
- This avoids multiple passes of macro expansion.



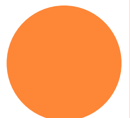
CONSIDER THE SITUATION

- Consider COMPUTE macro gives raise to the INCR_D macro calling statement which requires expansion of the INCR_D macro calling statement.
- These model statements will be expanded using the expansion time data structure MEC, APTAB, EVTAB, APTAB_ptr and EVTAB_ptr for inner macro and for outer macro these data structure should be restored with its original value.

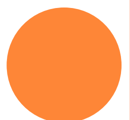


Required Provision

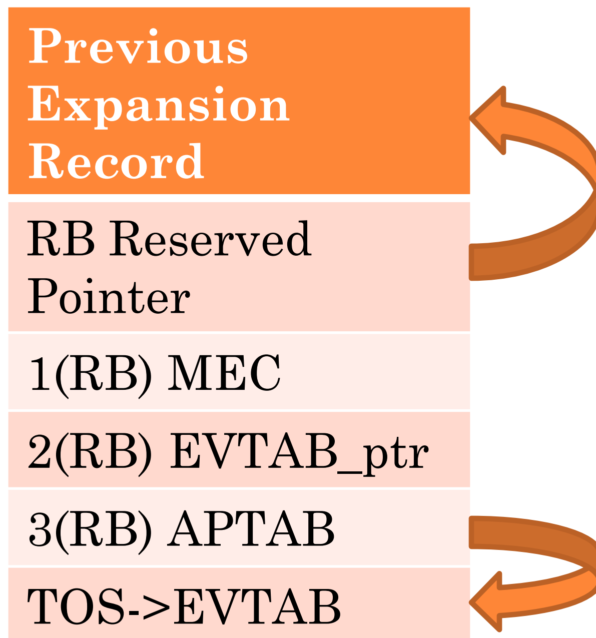
- Thus two provisions are required to implement the expansion of nested macro calls.
  - 1. Each macro under expansion must have its own set of data structure viz. MEC, APTAB, EVTAB, APTAB_ptr and EVTAB_ptr.
  - 2. An expansion nesting counter (Nest_cntr) is maintained to count the number of nested macro calls.
  - Nest_cntr is incremented when macro call is recognized and decremented when a MEND statement is encountered.
  - Thus Nest_cntr > 1 indicates that a nested macro call is under expansion, while Nest_cntr=0 implies that macro expansion is not in progress currently.



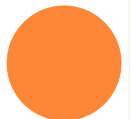
- The first provision can be implemented by creating many copies of the expansion time data structure.
- These can be stored in the form of an array. For example, we can have an array called APTAB_ARRAY, each element of which is an APTAB. For the innermost macro call would be given by APTAB_ARRAY[Nest_cntr].
- However it is expensive in terms of memory requirement.
- Since macro calls are expanded in a LIFO manner, a practical solution is to use a stack to accommodate the expansion time data structure.
- The stack consists of expansion records, each expansion record accommodating one set of expansion time data structure.
- The expansion record at the top of the stack corresponds to the macro call currently being expanded.
- When a nested macro call is recognized, a new expansion record is pushed on the stack to hold the data structure for the call.
- At MEND, an expansion record is popped off the stack.
- Use of stack for macro preprocessor data structure.



Data Structure for Nested Macro



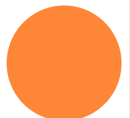
Data Structure	Address
Reserved Pointer	0(RB)
MEC	1(RB)
EVTAB_ptr	2(RB)
APTAB	3(RB) to entry of APTAB + 2(RB)
EVTAB	Contents of EVTAB_ptr



ACTIONS AT START AND END OF MACRO EXPANSION

No.	Statement
1.	TOS := TOS + 1;
2.	TOS* := RB;
3.	RB := TOS;
4.	1(RB) := MDTP entry of MNT;
5.	2(RB) := RB + 3 + #e of APTAB;
6.	TOS := TOS + #e of APTAB + #e of EVTAB + 2;

No.	Statement
1.	TOS := RB-1;
2.	RB := RB*;



See example 5.17 on page 158

Last topic: Design of Macro Assembler

- We have already discussed from the book.
- Do it from the book.
- Pg 158 to pg 160.

- And we end the chapter here.
- Thank You.