MODULE 5

MACRO PROCESSOR

A *Macro* represents a commonly used group of statements in the source programming language.

- A macro instruction (macro) is a notational convenience for the programmer
  - It allows the programmer to write shorthand version of a program (module programming)
- The macro processor replaces each macro instruction with the corresponding group of source language statements (*expanding*)
  - Normally, it performs no analysis of the text it handles.
  - It does not concern the meaning of the involved statements during macro expansion.
- The design of a macro processor generally is *machine independent!*
- Two new assembler directives are used in macro definition
  - **MACRO:** identify the beginning of a macro definition
  - **MEND:** identify the end of a macro definition
- Prototype for the macro
  - Each parameter begins with '&'

    -
      ```
      name   MACRO      parameters
             :
             body
             :
      MEND
      ```
  - Body: the statements that will be generated as the expansion of the macro.

5.1 Basic Macro Processor Functions:

- Macro Definition and Expansion
- Macro Processor Algorithms and Data structures

5.1.1 Macro Definition and Expansion:

- Consider the example of an SIC/XE program using macro instructions. This program defines and uses two macro instructions , RDBUFF and WRBUFF.

- The functions and logic of RDBUFF macro are similar to RDREC subroutine.

```
COPY      START    0               COPY FILE FROM INPUT TO OUTPUT
RDBUFF    MACRO    &INDEV,&BUFADR,&RECLTH
.
.         MACRO TO READ RECORD INTO BUFFER
.
          CLEAR    X               CLEAR LOOP COUNTER
          CLEAR    A
          CLEAR    S
         +LDT     #4096            SET MAXIMUM RECORD LENGTH
          TD      =X'&INDEV'       TEST INPUT DEVICE
          JEQ      *-3             LOOP UNTIL READY
          RD      =X'&INDEV'       READ CHARACTER INTO REG A
          COMPR    A,S             TEST FOR END OF RECORD
          JEQ      *+11            EXIT LOOP IF EOR
          STCH     &BUFADR,X       STORE CHARACTER IN BUFFER
          TIXR     T               LOOP UNLESS MAXIMUM LENGTH
          JLT      *-19              HAS BEEN REACHED
          STX      &RECLTH         SAVE RECORD LENGTH
          MEND
```

```
WRBUFF    MACRO    &OUTDEV,&BUFADR,&RECLTH
.
.         MACRO TO WRITE RECORD FROM BUFFER
.
          CLEAR    X               CLEAR LOOP COUNTER
          LDT      &RECLTH
          LDCH     &BUFADR,X       GET CHARACTER FROM BUFFER
          TD      =X'&OUTDEV'      TEST OUTPUT DEVICE
          JEQ      *-3             LOOP UNTIL READY
          WD      =X'&OUTDEV'      WRITE CHARACTER
          TIXR     T               LOOP UNTIL ALL CHARACTERS
          JLT      *-14              HAVE BEEN WRITTEN
          MEND
.
.         MAIN PROGRAM
.


FIRST     STL      RETADR          SAVE RETURN ADDRESS
CLOOP     RDBUFF   F1,BUFFER,LENGTH  READ RECORD INTO BUFFER
          LDA      LENGTH          TEST FOR END OF FILE
          COMP    #0
          JEQ      ENDFIL          EXIT IF EOF FOUND
          WRBUFF   05,BUFFER,LENGTH  WRITE OUTPUT RECORD
          J        CLOOP           LOOP
ENDFIL    WRBUFF   05,EOF,THREE    INSERT EOF MARKER
          J       @RETADR
EOF       BYTE     C'EOF'
THREE     WORD     3
RETADR    RESW     1
LENGTH    RESW     1               LENGTH OF RECORD
BUFFER    RESB     4096            4096-BYTE BUFFER AREA
          END      FIRST
```

Figure 4.1 Use of macros in a SIC/XE program.

- Two new assembler directives (Macro and MEND) are used in macro definitions. The keyword macro identifies the beginning of the macro definition. The symbol in the label field (RDBUFF) is the name of the macro and entries in the operand field identify the parameters of the macro. Each parameter begins with the character & which helps in the substitution of parameters during macro expansion. Following the macro directive are the statements that make up the body of the macro definition. These are the statements that will be generated as the expansion of the macro. The MEND directive marks the end of the macro.
- Macro invocation or call is written in the main program. In macro invocation the name of the macro is followed by the arguments. Output of the macroprocessor is the expanded program.

```
  5    COPY      START    0                  COPY FILE FROM INPUT TO OUTPUT
180    FIRST     STL      RETADR             SAVE RETURN ADDRESS
190    .CLOOP    RDBUFF   F1,BUFFER,LENGTH   READ RECORD INTO BUFFER
190a   CLOOP     CLEAR    X                  CLEAR LOOP COUNTER
190b             CLEAR    A
190c             CLEAR    S
190d            +LDT     #4096               SET MAXIMUM RECORD LENGTH
190e             TD      =X'F1'              TEST INPUT DEVICE
190f             JEQ      *-3                LOOP UNTIL READY
190g             RD      =X'F1'              READ CHARACTER INTO REG A
190h             COMPR    A,S                TEST FOR END OF RECORD
190i             JEQ      *+11               EXIT LOOP IF EOR
190j             STCH     BUFFER,X           STORE CHARACTER IN BUFFER
190k             TIXR     T                  LOOP UNLESS MAXIMUM LENGTH
190l             JLT      *-19                 HAS BEEN REACHED
190m             STX      LENGTH             SAVE RECORD LENGTH
195              LDA      LENGTH             TEST FOR END OF FILE
200              COMP    #0
205              JEQ      ENDFIL             EXIT IF EOF FOUND
```

Expanded Program

- Another simple example is given below:
- Program with macro

```
EX1       MACRO     &A,&B

          LDA       &A

          STA       &B

          MEND


SAMPLE    START     1000

          EX1       N1,N2

N1        RESW      1

N2        RESW      1


          END
```

[Type text]

Expanded program

```
SAMPLE      START           1000

.            EX1            N1,N2
             LDA            N1
             STA            N2
N1          RESW            1
N2          RESW            1
```

Macro expansion

- Macro definition statements have been deleted since they are no longer required after the macros are expanded. Each macro invocation statement has been expanded into the statements that form the body of the macro with the arguments from the macro invocation is substituted for the parameters in the macro definition. Macro invocation statement is included as a comment line in the expanded program.

- After macroprocessing the expanded file can be used as input to the assembler.

- Differences between macro and subroutine: The statements that form the expansion of a macro are generated and (assembled) each time the macro is invoked. Statements in a subroutine appear only once, regardless of how many time the subroutine is called.

5.1.2 Macro Processor Algorithm and Data Structure:

- It is easy to design a two pass macro processor in which all macro definitions are processed during the first pass and all macro invocation statements are expanded during the second pass.

- But such a two pass macro processor would not allow the body of one macro instruction to contain definitions of other macros.

```
1   MACROS     MACRO       {Defines SIC standard version macros}
2   RDBUFF     MACRO       &INDEV,&BUFADR,&RECLTH
               .
               .           {SIC  standard version}
               .
3              MEND        {End of RDBUFF}
4   WRBUFF     MACRO       &OUTDEV,&BUFADR,&RECLTH
               .
               .           {SIC standard version}
               .
5              MEND        {End of WRBUFF}
               .
               .
               .
6              MEND        {End of MACROS}
```

```
1   MACROX     MACRO       {Defines  SIC/XE macros}
2   RDBUFF     MACRO       &INDEV,&BUFADR,&RECLTH
               .
               .           {SIC/XE version}
               .
3              MEND        {End of RDBUFF}
4   WRBUFF     MACRO       &OUTDEV,&BUFADR,&RECLTH
               .
               .           {SIC/XE version}
               .
5              MEND        {End of WRBUFF}
               .
               .
               .
6              MEND        {End of MACROX}
```

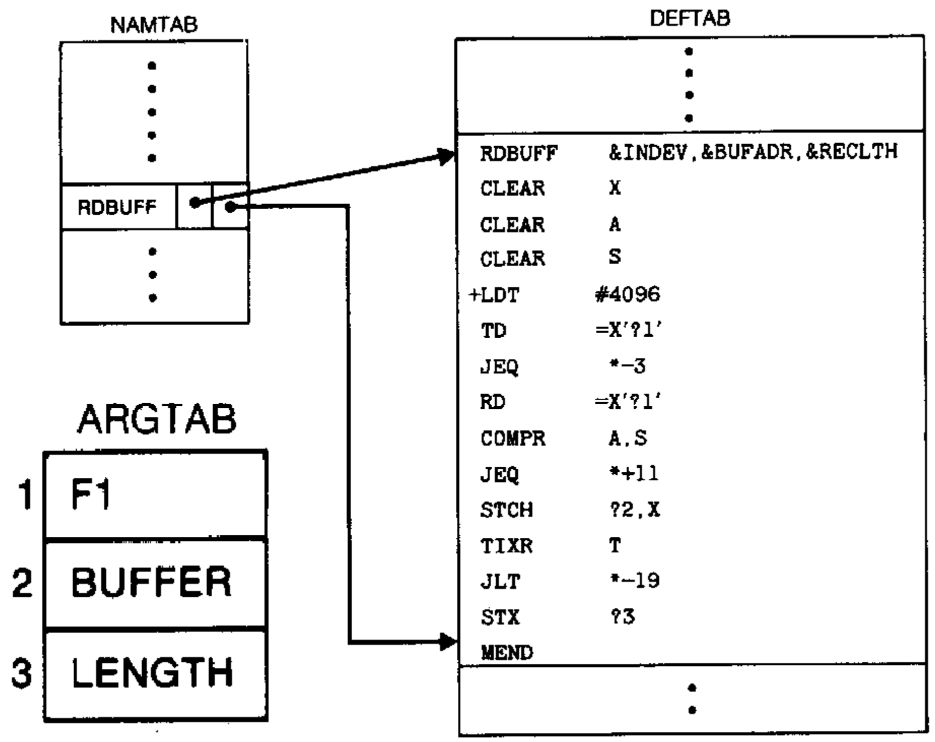
(b)

Figure 4.3 Example of the definition of macros within a macro body.

- Here defining MACROS does not define RDBUFF and WRBUFF. These definitions are processed only when an invocation of MACROS is expanded.
- A one pass macro processor that can alternate between macro definition and macro expansion is able to handle these type of macros.
- There are 3 main data structures:-
  - DEFTAB- The macro definitions are stored in a definition table(DEFTAB) which contain the

macro definition and the statements that form the macro body. References to the macro instruction parameters are converted to positional notation.

- NAMTAB- Macro names are entered into NAMTAB, which serves as an index to DEFTAB. For each macro instruction defined , NAMTAB contains pointers to the beginning and end of the definition in DEFTAB.
- ARGTAB- is used during the expansion of the macro invocation. When a macro invocation statement is recognized the arguments are stored in argument table. As the macro is expanded arguments from ARGTAB are substituted for the corresponding parameters in the macro body.
- Eg



```
RDBUFF    &INDEV,&BUFADR,&RECLTH
 CLEAR    X
 CLEAR    A
 CLEAR    S
+LDT      #4096
 TD       =X'?1'
 JEQ      *-3
 RD       =X'?1'
 COMPR    A,S
 JEQ      *+11
 STCH     ?2,X
 TIXR     T
 JLT      *-19
 STX      ?3
 MEND
```

Macro processor algorithm

```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE ≠ 'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
end {macro processor}


procedure PROCESSLINE
    begin
        search NAMTAB for OPCODE
        if found then
           EXPAND
        else if OPCODE = 'MACRO' then
           DEFINE
        else write source line to expanded file
    end {PROCESSLINE}
```

Figure 4.5 Algorithm for a one-pass macro processor.

```
procedure DEFINE
    begin
        enter macro name into NAMTAB
        enter macro prototype into DEFTAB
        LEVEL  := 1
        while LEVEL > 0 do
            begin
                GETLINE
                if this is not a comment line then
                    begin
                        substitute positional notation for parameters
                        enter line into DEFTAB
                        if OPCODE = 'MACRO' then
                            LEVEL := LEVEL + 1
                        else if OPCODE = 'MEND' then
                            LEVEL  := LEVEL - 1
                    end {if not comment}
            end {while}
        store in NAMTAB pointers to beginning and end of definition
    end {DEFINE}
```

```
procedure EXPAND
    begin
        EXPANDING  := TRUE
        get first line of macro definition {prototype} from DEFTAB
        set up arguments from macro invocation in ARGTAB
        write macro invocation to expanded file as a comment
        while not end of macro definition do
            begin
                GETLINE
                PROCESSLINE
            end {while}
        EXPANDING := FALSE
    end {EXPAND}


procedure GETLINE
    begin
        if EXPANDING then
            begin
                get next line of macro definition from DEFTAB
                substitute arguments from ARGTAB for positional notation
            end {if}
        else
            read next line from input file
    end {GETLINE}
```

Figure 4.5 (*cont'd*)

- Procedure DEFINE which is called when the beginning of a macro definition is recognized makes the appropriate entries in DEFTAB and NAMTAB.
- EXPAND is called to set up the argument values in ARGTAB and expand a *Macro Invocation* statement.
- Procedure GETLINE is called to get the next line to be processed either from the DEFTAB or from the input file .
- Handling of macro definition within macro:- When a macro definition is encountered it is entered in the DEFTAB. The normal approach is to continue entering till MEND is encountered. If there is a program having a Macro defined within another Macro.While defining in the DEFTAB the very first MEND is taken as the end of the Macro definition. This does not complete the definition as there is another outer Macro which completes the definition of Macro as a whole. Therefore the DEFINE procedure keeps a counter variable LEVEL.Every time a Macro directive is encountered this counter is incremented by 1. The moment the innermost Macro ends indicated by the directive MEND it starts decreasing the value of the counter variable by one. The last MEND should make the counter value set to zero. So when LEVEL becomes zero, the MEND corresponds to the original MACRO directive.

5.3Machine-independent Macro-Processor Features.

The design of macro processor doesn’t depend on the architecture of the machine. We will be studying some extended feature for this macro processor. These features are:

- Concatenation of Macro Parameters
- Generation of unique labels
- Conditional Macro Expansion
- Keyword Macro Parameters

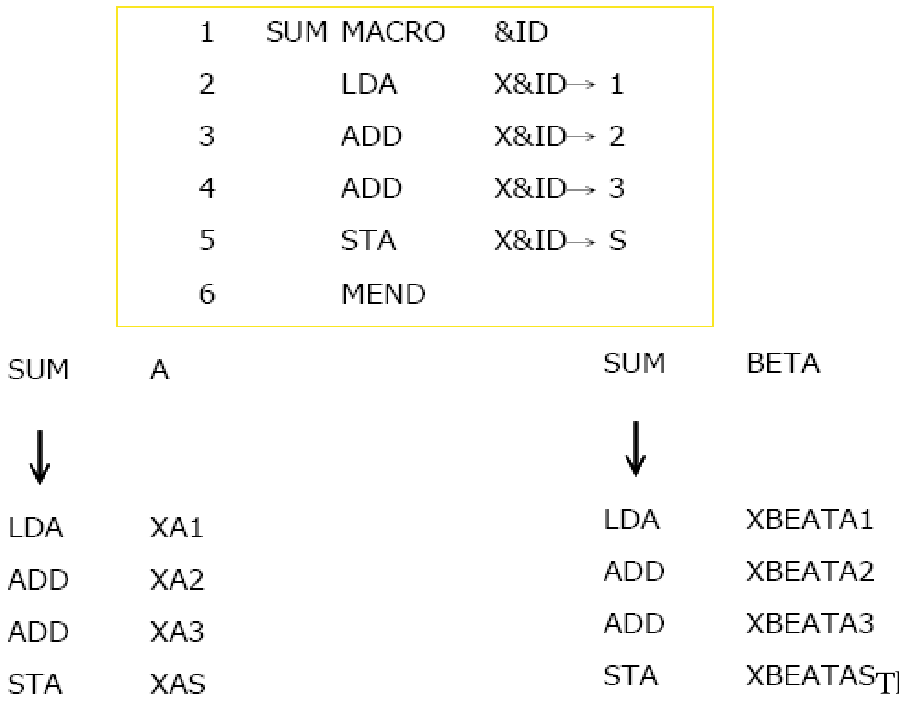
5.3.1Concatenation of Macro parameters:

- Most macro processor allows parameters to be concatenated with other character strings. Suppose that a program contains a series of variables named by the symbols XA1, XA2, XA3,…, another series of variables named XB1, XB2, XB3,…, etc. If similar processing is to be performed on each series of labels, the programmer might put this as a macro instruction.
- The parameter to such a macro instruction could specify the series of variables to be operated on (A, B, etc.). The macro processor would use this parameter to construct the symbols required in the macro expansion (XA1, XB1, etc.).

- Suppose that the parameter to such a macro instruction is named &ID. The body of the macro definition might contain a statement like
    - LDA X&ID1
- & is the starting character of the macro instruction; but the end of the parameter is not marked. So in the case of &ID1, the macro processor could deduce the meaning that was intended.

- If the macro definition contains &ID and &ID1 as parameters, the situation would be unavoidably ambiguous.
- Most of the macro processors deal with this problem by providing a special **concatenation operator.** In the SIC macro language, this operator is the character . Thus the statement LDA X&ID1 can be written as

```
LDA        X&ID[illegible]
```



```
1    SUM  MACRO    &ID
2         LDA      X&ID→ 1
3         ADD      X&ID→ 2
4         ADD      X&ID→ 3
5         STA      X&ID→ S
6         MEND
```

```
SUM    A                         SUM    BETA
  ↓                                ↓
LDA    XA1                       LDA    XBEATA1
ADD    XA2                       ADD    XBEATA2
ADD    XA3                       ADD    XBEATA3
STA    XAS                       STA    XBEATAS
```

- The above figure shows a macro definition that uses the concatenation operator as previously described. The statement SUM A and SUM BETA shows the invocation statements and the corresponding macro expansion.

5.3.2Generation of Unique Labels

- it is not possible to use labels for the instructions in the macro definition, since every expansion of macro would include the label repeatedly which is not allowed by the assembler.
- We can use the technique of generating unique labels for every macro invocation and expansion.
- During macro expansion each $ will be replaced with $XX, where xx is a two- character alphanumeric counter of the number of macro instructions expansion.

For example,

XX = AA, AB, AC…

This allows 1296 macro expansions in a single program.

The following program shows the macro definition with labels to the instruction.

```
RDBUFF   MACRO    &INDEV, &BUFADR, &RECLTH
         CLEAR    X                    CLEAR LOOP COUNTER
         CLEAR    A
         CLEAR    S
        +LDT      #4096                SET MAXIMUM RECORD LENGTH
$LOOP    TD       =X'&INDEV'           TEST INPUT DEVICE
         JEQ      $LOOP                LOOP UNTIL READY
         RD       =X'&INDEV'           READ CHARACTER INTI REG A
         COMPR    A, S                 TEST FOR END OF RECORD
         JEQ      $EXIT                EXIT LOOP IF EOR
         STCH     &BUFADR, X           STORE CHARACTER IN BUFFER
         TIXR     $LOOP                HAS BEEN REACHED
$EXIT    STX      &RECLTH              SAVE RECORD LENGTH
         MEND
```

The following figure shows the macro invocation and expansion first time.

```
.          RDBUFF   F1, BUFFER, LENGTH


           CLEAR    X                  CLEAR LOOP COUNTER
           CLEAR    A
           CLEAR    S
          +LDT      #4096              SET MAXIMUM RECORD LENGTH
$AALOOP    TD       =X'F1'             TEST INPUT DEVICE
           JEQ      $AALOOP            LOOP UNTIL READY
           RD       =X'F1'             READ CHARACTER INTI REG A
           COMPR    A, S               TEST FOR END OF RECORD
           JEQ      $AAEXIT            EXIT LOOP IF EOR
           STCH     BUFFER, X          STORE CHARACTER IN BUFFER
           TIXR     T                  LOOP UNLESS MAXIMUM LENGTH
           JLT      $AALOOP            HAS BEEN REACHED
$AAEXIT    STX      LENGTH             SAVE RECORD LENGTH
```

- If the macro is invoked second time the labels may be expanded as $ABLOOP $ABEXIT.

5.3.3Conditional Macro Expansion

- o IF ELSE
- o WHILE loop

- We can modify the sequence of statements generated for a macro expansion depending on conditions.

IF ELSE ENDIF structure

- Consider the following example.

```
25   RDBUFF   MACRO    &INDEV,&BUFADR,&RECLTH,&EOR,&MAXLTH
26            IF       (&EOR NE '')
27   &EORCK   SET      1
28            ENDIF
30            CLEAR    X                CLEAR LOOP COUNTER
35            CLEAR    A
38            IF       (&EORCK EQ 1)
40            LDCH    =X'&EOR'          SET EOR CHARACTER
42            RMO      A,S
43            ENDIF
44            IF       (&MAXLTH EQ '')
45           +LDT     #4096             SET MAX LENGTH = 4096
46            ELSE
47           +LDT     #&MAXLTH          SET MAXIMUM RECORD LENGTH
48            ENDIF
50   $LOOP    TD      =X'&INDEV'        TEST INPUT DEVICE
55            JEQ      $LOOP            LOOP UNTIL READY
60            RD      =X'&INDEV'        READ CHARACTER INTO REG A
63            IF       (&EORCK EQ 1)
65            COMPR    A,S              TEST FOR END OF RECORD
70            JEQ      $EXIT            EXIT LOOP IF EOR
73            ENDIF
75            STCH     &BUFADR,X        STORE CHARACTER IN BUFFER
80            TIXR     T                LOOP UNLESS MAXIMUM LENGTH
85            JLT      $LOOP              HAS BEEN REACHED
90   $EXIT    STX      &RECLTH          SAVE RECORD LENGTH
95            MEND
```

(a)

```
     .        RDBUFF   F3,BUF,RECL,04,2048


30            CLEAR    X                CLEAR LOOP COUNTER
35            CLEAR    A
40            LDCH    =X'04'            SET EOR CHARACTER
42            RMO      A,S
47           +LDT     #2048             SET MAXIMUM RECORD LENGTH
50   $AALOOP  TD      =X'F3'            TEST INPUT DEVICE
55            JEQ      $AALOOP          LOOP UNTIL READY
60            RD      =X'F3'            READ CHARACTER INTO REG A
65            COMPR    A,S              TEST FOR END OF RECORD
70            JEQ      $AAEXIT          EXIT LOOP IF EOR
75            STCH     BUF,X            STORE CHARACTER IN BUFFER
80            TIXR     T                LOOP UNLESS MAXIMUM LENGTH
85            JLT      $AALOOP            HAS BEEN REACHED
90   $AAEXIT  STX      RECL             SAVE RECORD LENGTH
```

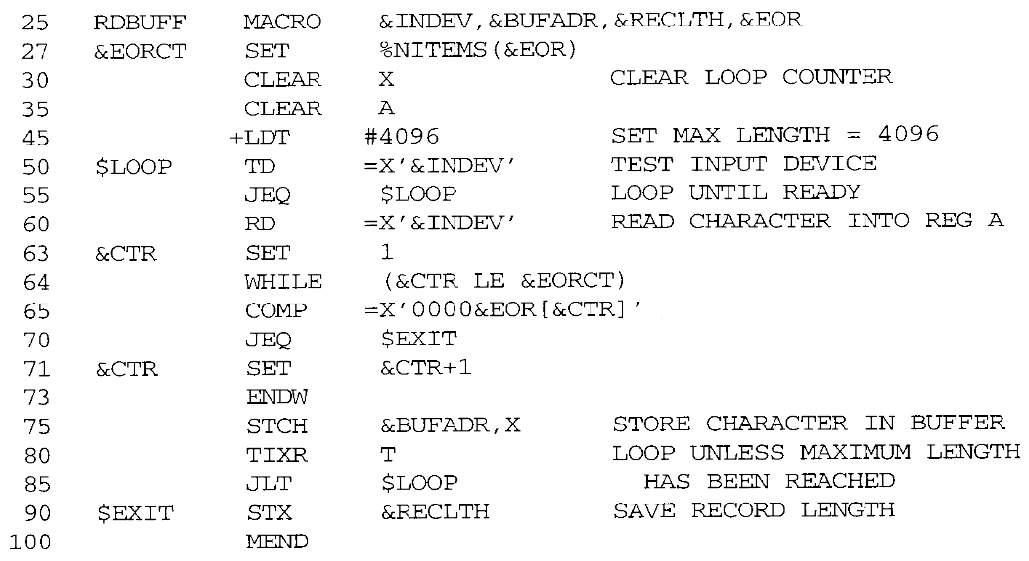
(b)

Figure 4.8 Use of macro-time conditional statements.

- Here the definition of RDBUFF has two additional parameters. &EOR(end of record) &MAXLTH(maximum length of the record that can be read)
- The macro processor directive SET – The statement assigns a value 1 to &EORCK and &EORCK is known as macrotime variable. A **macrotime variable** is used to store working values during the macro expansion. Any symbol that begins with & and that is not a macro instruction parameter is assumed to be a macro time variable. All such variables are initialized to a value 0.
- Implementation of Conditional macro expansion- Macro processor maintains a symbol table that contains the values of all macrotime variables used. Entries in this table are made when SET statements are processed. The table is used to look up the current value of the variable.
- Testing of Boolean expression in IF statement occurs at the time macros are expanded. By the time the program is assembled all such decisions are made and conditional macro instruction directives are removed.
- IF statements are different from COMPR which test data values during program expansion.

Looping-WHILE

- Consider the following example.



```
 25   RDBUFF    MACRO    &INDEV,&BUFADR,&RECLTH,&EOR
 27   &EORCT    SET      %NITEMS(&EOR)
 30             CLEAR    X                CLEAR LOOP COUNTER
 35             CLEAR    A
 45            +LDT      #4096            SET MAX LENGTH = 4096
 50   $LOOP     TD       =X'&INDEV'       TEST INPUT DEVICE
 55             JEQ      $LOOP            LOOP UNTIL READY
 60             RD       =X'&INDEV'       READ CHARACTER INTO REG A
 63   &CTR      SET      1
 64             WHILE    (&CTR LE &EORCT)
 65             COMP     =X'0000&EOR[&CTR]'
 70             JEQ      $EXIT
 71   &CTR      SET      &CTR+1
 73             ENDW
 75             STCH     &BUFADR,X        STORE CHARACTER IN BUFFER
 80             TIXR     T                LOOP UNLESS MAXIMUM LENGTH
 85             JLT      $LOOP              HAS BEEN REACHED
 90   $EXIT     STX      &RECLTH          SAVE RECORD LENGTH
100             MEND
```

(a)

```
      .          RDBUFF   F2,BUFFER,LENGTH,(00,03,04)


30               CLEAR    X               CLEAR LOOP COUNTER
35               CLEAR    A
45              +LDT     #4096            SET MAX LENGTH = 4096
50    $AALOOP    TD      =X'F2'           TEST INPUT DEVICE
55               JEQ      $AALOOP         LOOP UNTIL READY
60               RD      =X'F2'           READ CHARACTER INTO REG A
65               COMP    =X'000000'
70               JEQ      $AAEXIT
65               COMP    =X'000003'
70               JEQ      $AAEXIT
65               COMP    =X'000004'
70               JEQ      $AAEXIT
75               STCH     BUFFER,X        STORE CHARACTER IN BUFFER
80               TIXR     T               LOOP UNLESS MAXIMUM LENGTH
85               JLT      $AALOOP           HAS BEEN REACHED
90    $AAEXIT    STX      LENGTH          SAVE RECORD LENGTH
```

(b)

- Here the programmer can specify a list of end of record characters.
- In the macro invocation statement there is a list(00,03,04) corresponding to the parameter &EOR. Any one of these characters is to be considered as end of record.
- The WHILE statement specifies that the following lines until the next ENDW are to be generated repeatedly as long as the condition is true.
- The testing of these condition and the looping are done while the macro is being expanded.The conditions do not contain any runtime values.
- %NITEMS is a macroprocessor function that returns as its value the number of members in an argument list. Here it has the value 3. The value of &CTR is used as a subscript to select the proper member of the list for each iteration of the loop. &EOR[&CTR] takes the values 00,03,04 .
- Implementation- When a WHILE statement is encountered during a macro expansion the specified Boolean expression is evaluated , if the value is false the macroprocessor skips ahead in DEFTAB until it finds the ENDW and then resumes normal macro expansion(not at run time).

5.3.4Keyword Macro Parameters

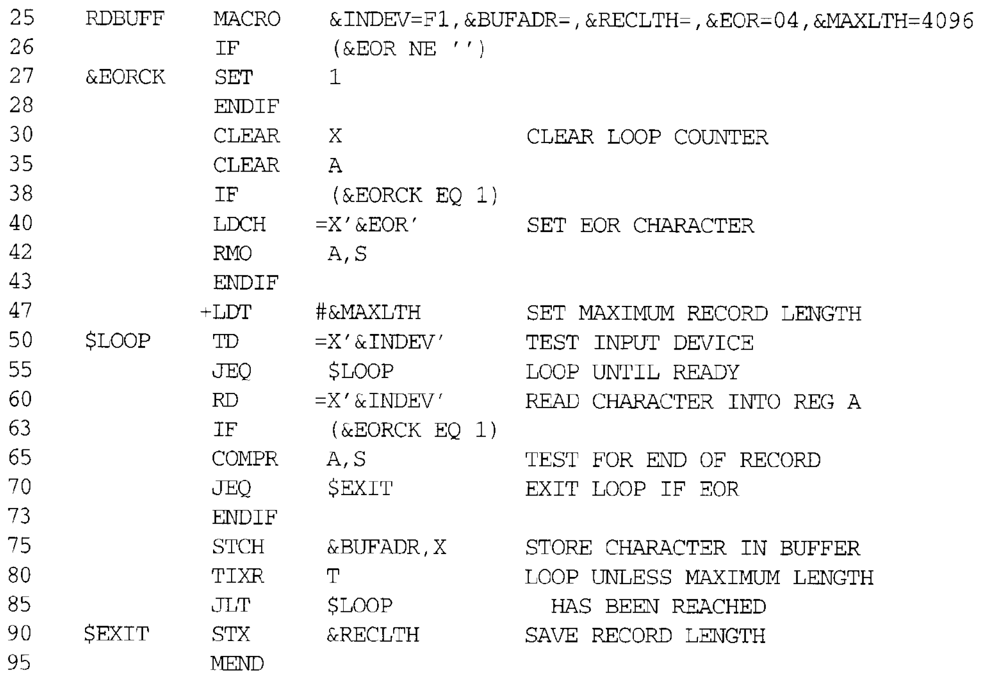
- All the macro instruction definitions used positional parameters. Parameters and arguments are matched according to their positions in the macro prototype and the macro invocation statement.

- The programmer needs to be careful while specifying the arguments. If an argument is to be omitted the macro invocation statement must contain a null argument mentioned with two commas.

- Positional parameters are suitable for the macro invocation. But if the macro invocation has large number of parameters, and if only few of the values need to be used in a typical invocation, a different type of parameter specification is required.
- Eg: Consider the macro GENER which has 10 parameters, but in a particular invocation of a macro only the third and nineth parameters are to be specified. If positional parameters are used the macro invocation will look like

  GENER , , DIRECT, , , , , , 3,
- But using keyword parameters this problem can be solved. We can write

  GENER TYPE=DIRECT, CHANNEL=3



```
RDBUFF   MACRO   &INDEV=F1,&BUFADR=,&RECLTH=,&EOR=04,&MAXLTH=4096
         IF      (&EOR NE '')
&EORCK   SET     1
         ENDIF
         CLEAR   X               CLEAR LOOP COUNTER
         CLEAR   A
         IF      (&EORCK EQ 1)
         LDCH    =X'&EOR'        SET EOR CHARACTER
         RMO     A,S
         ENDIF
        +LDT     #&MAXLTH        SET MAXIMUM RECORD LENGTH
$LOOP    TD      =X'&INDEV'      TEST INPUT DEVICE
         JEQ     $LOOP           LOOP UNTIL READY
         RD      =X'&INDEV'      READ CHARACTER INTO REG A
         IF      (&EORCK EQ 1)
         COMPR   A,S             TEST FOR END OF RECORD
         JEQ     $EXIT           EXIT LOOP IF EOR
         ENDIF
         STCH    &BUFADR,X       STORE CHARACTER IN BUFFER
         TIXR    T               LOOP UNLESS MAXIMUM LENGTH
         JLT     $LOOP             HAS BEEN REACHED
$EXIT    STX     &RECLTH         SAVE RECORD LENGTH
         MEND
```

```
.           RDBUFF   BUFADR=BUFFER,RECLTH=LENGTH

            CLEAR    X                 CLEAR LOOP COUNTER
            CLEAR    A
            LDCH    =X'04'             SET EOR CHARACTER
            RMO      A,S
           +LDT     #4096              SET MAXIMUM RECORD LENGTH
$AALOOP     TD      =X'F1'             TEST INPUT DEVICE
            JEQ      $AALOOP           LOOP UNTIL READY
            RD      =X'F1'             READ CHARACTER INTO REG A
            COMPR    A,S               TEST FOR END OF RECORD
            JEQ      $AAEXIT           EXIT LOOP IF EOR
            STCH     BUFFER,X          STORE CHARACTER IN BUFFER
            TIXR     T                 LOOP UNLESS MAXIMUM LENGTH
            JLT      $AALOOP             HAS BEEN REACHED
$AAEXIT     STX      LENGTH            SAVE RECORD LENGTH
```

(b)

Figure 4.10 Use of keyword parameters in macro instructions.

Keyword parameters

- Each argument value is written with a keyword that names the corresponding parameter.
- Arguments may appear in any order.
- Null arguments no longer need to be used.
- It is easier to read and much less error-prone than the positional method.

5.4 Macro Processor Design Options

5.4.1Recursive Macro Expansion

- We have seen an example of the *definition* of one macro instruction by another. But we have not dealt with the *invocation* of one macro by another. The following example shows the invocation of one macro by another macro.

```
10    RDBUFF   MACRO    &BUFADR, &RECLTH, &INDEV
15    .
20    .        MACRO TO READ RECORD INTO BUFFER
25    .
30             CLEAR    X                 CLEAR LOOP COUNTER
35             CLEAR    A
40             CLEAR    S
45            +LDT      #4096             SET MAXIMUN RECORD LENGTH
50    $LOOP    RDCHAR   &INDEV            READ CHARACTER INTO REG A
65             COMPR    A, S              TEST FOR END OF RECORD
70             JEQ      &EXIT             EXIT LOOP IF EOR
75             STCH     &BUFADR, X        STORE CHARACTER IN BUFFER
80             TIXR     T                 LOOP UNLESS MAXIMUN LENGTH
85             JLT      $LOOP             HAS BEEN REACHED
90    $EXIT    STX      &RECLTH           SAVE RECORD LENGTH
95             MEND
```

```
5    RDCHAR        MACRO    &IN
10   .
15   .     MACROTO READ CHARACTER INTO REGISTER A
20   .
25                 TD       =X'&IN'                 TEST INPUT DEVICE
30                 JEQ        *-3                   LOOP UNTIL READY
35                 RD       =X'&IN'                 READ CHARACTER
40                 MEND
```

Problem of Recursive Expansion

- Previous macro processor design cannot handle such kind of recursive macro invocation and expansion
    - The procedure EXPAND would be called recursively, thus the invocation arguments in the ARGTAB will be overwritten.
    - The Boolean variable EXPANDING would be set to FALSE when the “inner” macro expansion is finished, *i.e.*, the macro process would forget that it had been in the middle of expanding an “outer” macro.

The procedure EXPAND would be called when the macro was recognized. The arguments from the macro invocation would be entered into ARGTAB as follows:

Parameter	Value
1	BUFFER
2	LENGTH
3	F1
4	(unused)
-	-

The Boolean variable EXPANDING would be set to TRUE, and expansion of the macro invocation statement would begin. The processing would proceed normally until statement invoking RDCHAR is processed. This time, ARGTAB would look like

Parameter	Value
1	F1

2	(Unused)
--	--

At the expansion, when the end of RDCHAR is recognized, EXPANDING would be set to FALSE. Thus the macro processor would ‘forget’ that it had been in the middle of expanding a macro when it encountered the RDCHAR statement. In addition, the arguments from the original macro invocation (RDBUFF) would be lost because the value in ARGTAB was overwritten with the arguments from the invocation of RDCHAR.

- Solutions
  - Write the macro processor in a programming language that allows recursive calls, thus local variables will be retained.
  - If you are writing in a language without recursion support, use a stack to take care of pushing and popping local variables and return addresses.

5.4.2General-Purpose Macro Processors

- Macro processors that do not dependent on any particular programming language, but can be used with a variety of different languages
- **Pros**
  - Programmers do not need to learn many macro languages.
  - Although its development costs are somewhat greater than those for a language specific macro processor, this expense does not need to be repeated for each language, thus save substantial overall cost.
- **Cons**
  - Large number of details must be dealt with in a real programming language
    - Situations in which normal macro parameter substitution should not occur, e.g., comments.
    - Facilities for grouping together terms, expressions, or statements. Eg: some languages use begin and end . Some use { and }
    - Tokens, e.g., identifiers, constants, operators, keywords
    - Syntax used for macro definition and macro invocation statement is different.

5.4.3Macro Processing within Language Translators

- The macro processors we discussed are called “Preprocessors”.
  - Process macro definitions
  - Expand macro invocations
  - Produce an expanded version of the source program, which is then used as input to an assembler or compiler
- You may also combine the macro processing functions with the language translator:
  - Line-by-line macro processor
  - Integrated macro processor

Line-by-Line Macro Processor

- Used as a sort of input routine for the assembler or compiler
  - Read source program
  - Process macro definitions and expand macro invocations
  - Pass output lines to the assembler or compiler
- Benefits
  - Avoid making an extra pass over the source program.
  - Data structures required by the macro processor and the language translator can be combined (e.g., OPTAB and NAMTAB)
  - Utility subroutines can be used by both macro processor and the language translator.
    - Scanning input lines
    - Searching tables
    - Data format conversion
  - It is easier to give diagnostic messages related to the source statements

Integrated Macro Processor

- An integrated macro processor can potentially make use of any information about the source program that is extracted by the language translator.
  - Ex (blanks are not significant in FORTRAN)
    - DO 100 I = 1,20

        - a DO statement
    - DO 100 I = 1
        - An assignment statement
        - DO100I: variable (blanks are not significant in FORTRAN)
- An integrated macro processor can support macro instructions that depend upon the context in which they occur.

- Disadvantages- They must be specially designed and written to work with a particular implementation of an assembler or compiler.. Cost of development is high.