Unit 4: Macro Processor Design [8 hrs]

Introduction

- A ***macro instruction*** *(often abbreviated to* ***macro****) is simply a notational convenience* for the programmer.
- A **macro** represents a commonly used group of statements in the source programming language.
- The **macro processor** replaces each macro instruction with the corresponding group of source language statements. This is called ***expanding the macros****.*
- *Thus* ***macro instructions*** allow the programmer to write a shorthand version of a program, and leave the mechanical details to be handled by the macro processor.
- For example, suppose that it is necessary to save the contents of all registers before calling a subprogram.
- On SIC/XE, this would require a sequence of seven instructions (STA, STB, etc.). Using a macro instruction, the programmer could simply write one statement like **SAVEREGS**. This macro instruction would be expanded into the seven assembler language instructions needed to save the register contents.
- A similar macro instruction (perhaps named **LOADREGS**) could be used to reload the register contents after returning from the subprogram.
- The functions of a macro processor essentially involve the substitution of one group of characters or lines for another.
- Except in a few specialized cases, the macro processor performs no analysis of the text it handles.

4.1: Basic Macro Processor Functions

Macro Definition and Expansion

- Figure 4.1 shows an example of a SIC / XE program using macro instructions.
- This program has the same functions and logic as the sample program in Fig. 2.5; however, the numbering scheme used for the source statements has been changed.
- This program defines and uses two macro instructions, **RDBUFF** and **WRBUFF**.
- The functions and logic of the **RDBUFF** macro are similar to those of the **RDREC** subroutine, in Fig. 2.5; likewise, the **WRBUFF** macro is similar to the **WRREC** subroutine.
- The definitions of these macro instructions appear in the source program following the **START** statement.

Use of macros in a SICIXE program (4.1)

Line	Source statement			
5	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
10	RDBUFF	MACRO	&INDEV,&BUFADR,&RECLTH	
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 MAXIMUM RECORD LENGTH
50		TD	=X'&INDEV'	TEST INPUT DEVICE
55		JEQ	*-3	LOOP UNTIL READY
60		RD	=X'&INDEV'	READ CHARACTER INTO REG A
65		COMPR	A,S	TEST FOR END OF RECORD
70		JEQ	*+11	EXIT LOOP IF EOR
75		STCH	&BUFADR,X	STORE CHARACTER IN BUFFER
80		TIXR	T	LOOP UNLESS MAXIMUM LENGTH
85		JLT	*-19	HAS BEEN REACHED
90		STX	&RECLTH	SAVE RECORD LENGTH
95		MEND		

(4.1 Continued)

```
100     WRBUFF  MACRO   &OUTDEV,&BUFADR,&RECLTH
105     .
110     .       MACRO TO WRITE RECORD FROM BUFFER
115     .
120             CLEAR   X               CLEAR LOOP COUNTER
125             LDT     &RECLTH
130             LDCH    &BUFADR,X       GET CHARACTER FROM BUFFER
135             TD      =X'&OUTDEV'     TEST OUTPUT DEVICE
140             JEQ     *-3             LOOP UNTIL READY
145             WD      =X'&OUTDEV'     WRITE CHARACTER
150             TIXR    T               LOOP UNTIL ALL CHARACTERS
155             JLT     *-14              HAVE BEEN WRITTEN
160             MEND
165     .
170     .       MAIN PROGRAM
175     .
180     FIRST   STL     RETADR          SAVE RETURN ADDRESS
190     CLOOP   RDBUFF  F1,BUFFER,LENGTH  READ RECORD INTO BUFFER
195             LDA     LENGTH          TEST FOR END OF FILE
200             COMP    #0
205             JEQ     ENDFIL          EXIT IF EOF FOUND
210             WRBUFF  05,BUFFER,LENGTH  WRITE OUTPUT RECORD
215             J       CLOOP           LOOP
220     ENDFIL  WRBUFF  05,EOF,THREE    INSERT EOF MARKER
225             J       @RETADR
230     EOF     BYTE    C'EOF'
235     THREE   WORD    3
240     RETADR  RESW    1
245     LENGTH  RESW    1               LENGTH OF RECORD
250     BUFFER  RESB    4096            4096-BYTE BUFFER AREA
255             END     FIRST
```

- Two new assembler directives (**MACRO** and **MEND**) are used in macro definitions.
- The first **MACRO** statement (line 10) identifies the beginning of a macro definition.
- The symbol in the label field (**RDBUFF**) is the name of the macro, and the entries in the operand field identify the ***parameters*** *of the macro* instruction.
- In our macro language, each parameter begins with the character **&**, which facilitates the substitution of parameters during macro expansion.
- The **macro name** and **parameters** define a pattern or ***prototype*** *for the* ***macro instructions*** used by the programmer.
- Following the **MACRO** directive are the statements that make up the *body of the* ***macro definition*** *(lines 15 through 90).*
- These are the statements that will be generated as the expansion of the macro.
- The **MEND** assembler directive (line 95) marks the end of the macro definition.
- The definition of the **WRBUFF** macro (lines 100 through 160) follows a similar pattern.

- The main program itself begins on line 180.
- The statement on line 190 is a ***macro invocation statement*** *that gives the name of the macro instruction being* invoked and the *arguments to be used in expanding the macro.*
- *A* ***macro invocation*** **statement** is often referred to as a ***macro call****.*
- *To avoid confusion with the* call statements used for procedures and subroutines, we prefer to use the term ***invocation****.*
- *As we shall see, the processes of* ***macro invocation*** *and* ***subroutine*** **call** are quite different.
- You should compare the logic of the main program in Fig. 4.1 with that of the main program in Fig. 2.5, remembering the similarities in function between **RDBUFF** and **RDREC** and between **WRBUFF** and **WRREC**.

Program from Fig. 4.1 with macros expanded (4.2)

Line	Source statement			
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
210		WRBUFF	05,BUFFER,LENGTH	WRITE OUTPUT RECORD
210a		CLEAR	X	CLEAR LOOP COUNTER
210b		LDT	LENGTH	
210c		LDCH	BUFFER,X	GET CHARACTER FROM BUFFER

(4.2 Continued)

```
210d            TD      =X'05'          TEST OUTPUT DEVICE
210e            JEQ      *-3            LOOP UNTIL READY
210f            WD      =X'05'          WRITE CHARACTER
210g            TIXR     T              LOOP UNTIL ALL CHARACTERS
210h            JLT      *-14             HAVE BEEN WRITTEN
215             J        CLOOP          LOOP
220     .ENDFIL WRBUFF   05,EOF,THREE   INSERT EOF MARKER
220a    ENDFIL  CLEAR    X              CLEAR LOOP COUNTER
220b            LDT      THREE
220c            LDCH     EOF,X          GET CHARACTER FROM BUFFER
220d            TD      =X'05'          TEST OUTPUT DEVICE
220e            JEQ      *-3            LOOP UNTIL READY
220f            WD      =X'05'          WRITE CHARACTER
220g            TIXR     T              LOOP UNTIL ALL CHARACTERS
220h            JLT      *-14             HAVE BEEN WRITTEN
225             J       @RETADR
230     EOF     BYTE     C'EOF'
235     THREE   WORD     3
240     RETADR  RESW     1
245     LENGTH  RESW     1              LENGTH OF RECORD
250     BUFFER  RESB     4096           4096-BYTE BUFFER AREA
255             END      FIRST
```

- Figure 4.2 shows the output that would be generated.
- The macro instruction definitions have been deleted since they are no longer needed 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 substituted for the parameters in the macro prototype.
- The arguments and parameters are associated with one another according to their positions.
- The first argument in the **macro invocation** corresponds to the first parameter in the macro prototype, and so on.
- In expanding the macro invocation on line 190, for example, the argument **F1** is substituted for the parameter **&INDEV** wherever it occurs in the body of the macro.
- Similarly, **BUFFER** is substituted for **&BUFADR**, and **LENGTH** is substituted for **&RECLTH**.

- Lines 190a through 190m show the complete expansion of the macro invocation on line 190.
- The comment lines within the macro body have been deleted, but comments on individual statements have been retained.
- Note that the **macro invocation statement** itself has been included as a comment line. This serves as documentation of the statement written by the programmer.
- The label on the macro invocation statement (**CLOOP**) has been retained as a label on the first statement generated in the macro expansion.
- This allows the programmer to use a macro instruction in exactly the same way as an assembler language mnemonic.
- The macro invocations on lines 210 and 220 are expanded in the same way.
- Note that the two invocations of **WRBUFF** specify different arguments, so they produce different expansions.
- After macro processing, the expanded file (Fig. 4.2) can be used as input to the assembler.
- The **macro invocation statements** will be treated as comments, and the statements generated from the macro expansions will be assembled exactly as though they had been written directly by the programmer.

Macro Processor Data Structures

- 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.
- However, such a two-pass macro processor would not allow the body of one macro instruction to contain definitions of other macros (because all macros would have to be defined during the first pass before any **macro invocations** were expanded).
- Such definitions of macros by other macros can be useful in certain cases.

- Consider, for example, the two macro instruction definitions in Fig. 4.3.
- The body of the first macro (**MACROS**) contains statements that define **RDBUFF**, **WRBUFF**, and other macro instructions for a SIC system (standard version).
- The body of the second macro instruction (**MACROX**) defines these same macros for a SIC/XE system.
- A program that is to be run on a standard SIC system could invoke **MACROS** to define the other utility macro instructions.
- A program for a SIC/XE system could invoke **MACROX** to define these same macros in their XE versions.
- In this way, the same program could run on either a standard SIC machine or a SIC/XE machine (taking advantage of the extended features).
- The only change required would be the invocation of either **MACROS** or **MACROX**.
- It is important to understand that *defining **MACROS*** or **MACROX** does not define **RDBUFF** and the other macro instructions.
- These definitions are processed only when an invocation of **MACROS** or **MACROX** is *expanded.*

Example of the definition of macros within a macro body (4.3 a)

```
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)
```

Example of the definition of macros within a macro body (4.3 b)

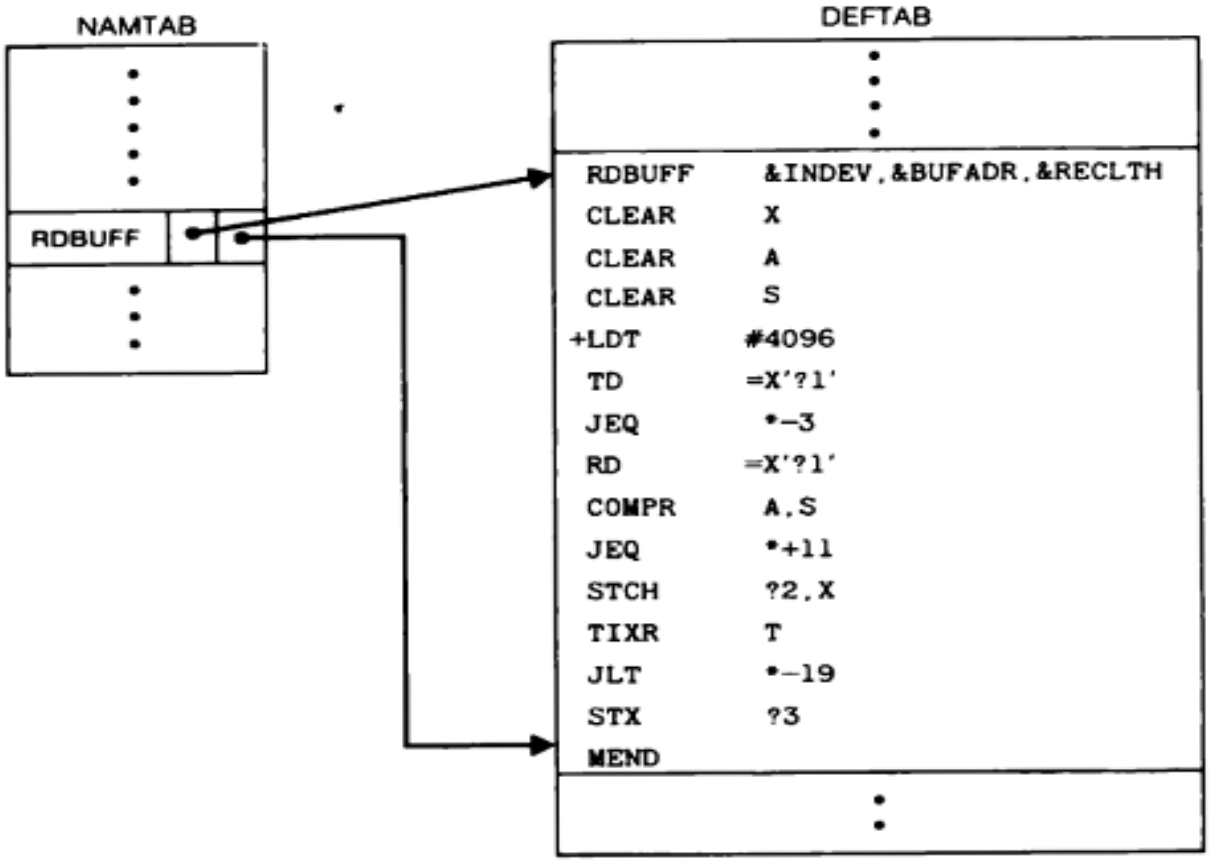
```
1   MACROX    MACRO        {Defines  SIC/XE macros}
2   RDBUFF    MACRO        &INDEV,&BUFADR,&RECLTH
                .
                .          {SIC/XE version}
                .
3             MEND         {End of RDBUFF}
4   WRBUFF    MACRO        &OUTDEV,&BUFADR,&RECLTH
                .
  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

- A one-pass macro processor that can alternate between macro definition and macro expansion is able to handle macros like those in Fig. 4.3.
- In this section we present an algorithm and a set of data structures for such a macro processor.
- Because of the one-pass structure, the definition of a macro must appear in the source program before any statements that invoke that macro.
- This restriction does not create any real inconvenience for the programmer.
- In fact, a macro invocation statement that preceded the definition of the macro would be confusing for anyone reading the program.

- There are **three** main **data structures** involved in our macro processor.
- The macro definitions themselves are stored in a definition table (**DEFTAB**), which contains the macro prototype and the statements that make up the macro body (with a few modifications).
- Comment lines from the macro definition are not entered into **DEFTAB** because they will not be part of the macro expansion.
- References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.
- The macro names are also 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**.
- The third data structure is an argument table (**ARGTAB**), which is used during the expansion of macro invocations.
- When a macro invocation statement is recognized, the arguments are stored in **ARGTAB** according to their position in the argument list.
- As the macro is expanded, arguments from **ARGTAB** are substituted for the corresponding parameters in the macro body.

- Figure 4.4 shows portions of the contents of these tables during the processing of the program in Fig. 4.1.
- Figure 4.4(a) shows the definition of **RDBUFF** stored in **DEFTAB**, with an entry in **NAMTAB** identifying the beginning and end of the definition.
- Note the positional notation that has been used for the parameters: the parameter **&INDEV** has been converted to **? 1** (indicating the first parameter in the prototype), **&BUFADR** has been converted to **?2**, and so on.
- Figure 4.4(b) shows **ARGTAB** as it would appear during expansion of the **RDBUFF** statement on line 190.
- For this invocation, the first argument is **F1**, the second is **BUFFER**, etc.
- This scheme makes substitution of macro arguments much more efficient.
- When the ***?n*** *notation is recognized in a line from* **DEFTAB**, a simple indexing operation supplies the proper argument from **ARGTAB**.

Contents of macro processor tables for the program in Fig. 4.1: (a) entries in NAMTAB and DEFTAB defining macro RDBUFF, (b) entries in ARGTAB for invocation of RDBUFF on line 190 (4.4)



4.2: MACHINE-INDEPENDENT MACRO PROCESSOR FEATURES

Concatenation of Macro Parameters

- Most macro processors allow parameters to be concatenated with other character strings.
- Suppose, for example, that a program contains one series of variables named by the symbols XA1, XA2, XA3, ..., another series named by XB1, XB2, XB3, ..., etc.
- If similar processing is to be performed on each series of variables, the programmer might want to incorporate this processing into 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

  in which the parameter **&ID** is concatenated after the character string **X** and before the character string **1**.
- Closer examination, however, reveals a problem with such a statement.
- The beginning of the macro parameter is identified by the starting symbol **&**; however, the end of the parameter is not marked.
- Thus the operand in the foregoing statement could equally well represent the character string **X** followed by the parameter **&ID1**.
- In this particular case, the macro processor could potentially deduce the meaning that was intended.
- However, if the macro definition contained both **&ID** and **&ID1** as parameters, the situation would be unavoidably ambiguous.

- Most macro processors deal with this problem by providing a special ***concatenation operator***.
- *In the SIC macro language, this operator is the character →.*
- Thus the previous statement would be written as

  LDA X& ID →*1*

  so that the end of the parameter **&ID** is clearly identified.
- The macro processor deletes all occurrences of the concatenation operator immediately after performing parameter substitution, so the character → will not appear in the macro expansion.
- Figure 4.5(a) shows a macro definition that uses the concatenation operator as previously described.
- Figure 4.5(b) and (c) shows macro invocation statements and the corresponding macro expansions.

Concatenation of macro parameters (4.5)

```
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
```

(a)

```
SUM     A
 ↓
LDA     XA1
ADD     XA2
ADD     XA3
STA     XAS
```

(b)

```
SUM     BETA
 ↓
LDA     XBETA1
ADD     XBETA2
ADD     XBETA3
STA     XBETAS
```

(c)

Generation of Unique Labels

- It is in general not possible for the body of a macro instruction to contain labels of the usual kind – This leads to the use of relative addressing at the source statement level.
- Consider, for example, the definition of **WRBUFF** in Fig. 4.1.
- If a label were placed on the **TD** instruction on line 135, this label would be defined twice-once for each invocation of **WRBUFF**.
- This duplicate definition would prevent correct assembly of the resulting expanded program.
- Because it was not possible to place a label on line 135 of this macro definition, the **Jump** instructions on lines 140 and 155 were written using the relative operands ***-3** and ***-14**.
- This sort of relative addressing in a source statement may be acceptable for short jumps such as "**JEQ *-3**."
- However, for longer jumps spanning several instructions, such notation is very **inconvenient**, **error-prone**, and **difficult** to read.
- Many macro processors avoid these problems by allowing the creation of special types of labels within macro instructions.

- Figure 4.6 illustrates one technique for generating unique labels within a macro expansion.
- A definition of the **RDBUFF** macro is shown in Fig. 4.6(a).
- Labels used within the macro body begin with the special character **$.**
- Figure 4.6(b) shows a macro invocation statement and the resulting macro expansion.
- Each symbol beginning with **$** has been modified by replacing **$** with **$AA**.
- More generally, the character **$** will be replaced by **$xx**, where **xx** is a two character alphanumeric counter of the number of macro instructions expanded.
- For the first macro expansion in a program, **xx** will have the value **AA**.
- For succeeding macro expansions, **xx** will be set to **AB**, **AC**, etc. (If only alphabetic and numeric characters are allowed in **xx**, such a two-character counter provides for as many as 1296 macro expansions in a single program.)
- This results in the generation of unique labels for each expansion of a macro instruction.

Generation of unique labels within macro expansion (4.6 a)

```
25    RDBUFF   MACRO   &INDEV,&BUFADR,&RECLTH
30             CLEAR   X              CLEAR LOOP COUNTER
35             CLEAR   A
40             CLEAR   S
45             +LDT    #4096          SET MAXIMUM RECORD LENGTH
50    $LOOP    TD      =X'&INDEV'     TEST INPUT DEVICE
55             JEQ     $LOOP          LOOP UNTIL READY
60             RD      =X'&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 MAXIMUM LENGTH
85             JLT     $LOOP            HAS BEEN REACHED
90    $EXIT    STX     &RECLTH        SAVE RECORD LENGTH
95             MEND
```

Generation of unique labels within macro expansion (4.6 b)

```
        .       RDBUFF  F1,BUFFER,LENGTH


30              CLEAR   X               CLEAR LOOP COUNTER
35              CLEAR   A
40              CLEAR   S
45              +LDT    #4096           SET MAXIMUM RECORD LENGTH
50      $AALOOP TD      =X'F1'          TEST INPUT DEVICE
55              JEQ     $AALOOP         LOOP UNTIL READY
60              RD      =X'F1'          READ CHARACTER INTO REG A
65              COMPR   A,S             TEST FOR END OF RECORD
70              JEQ     $AAEXIT         EXIT LOOP IF EOR
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
```

Conditional Macro Expansion

- In all of our previous examples of macro instructions, each invocation of a particular macro was expanded into the same sequence of statements.
- These statements could be varied by the substitution of parameters, but the form of the statements, and the order in which they appeared, were unchanged.
- A simple macro facility such as this can be a useful tool.
- Most macro processors, however, can also modify the sequence of statements generated for a macro expansion, depending on the arguments supplied in the macro invocation.
- Such a capability adds greatly to the **power** and **flexibility** of a macro language.
- The term ***conditional assembly*** *is commonly used to describe features such* as those discussed in this section.
- However, there are applications of macro processors that are not related to assemblers or assembler language programming.
- For this reason, we prefer to use the term ***conditional macro expansion****.*

- The use of one type of conditional macro expansion statement is illustrated in Fig. 4.7 that shows a definition of a macro **RDBUFF**, the logic and functions of which are similar to those previously discussed.
- However, this definition of **RDBUFF** has two additional parameters: **&EOR**, which specifies a hexadecimal character code that marks the end of a record, and **&MAXLTH**, which specifies the maximum length record that can be read.
- The statements on lines 44 through 48 of this definition illustrate a simple macro-time conditional structure.
- The IF statement evaluates a Boolean expression that is its operand. If the value of this expression is TRUE, the statements following the IF are generated until an ELSE is encountered. Otherwise, these statements are skipped, and the statements following the ELSE are generated. The ENDIF statement terminates the conditional expression that was begun by the IF statement.
- Thus if the parameter **&MAXLTH** is equal to the null string(that is, if the corresponding argument was omitted in the macro invocation statement), the statement on line 45 is generated.
- Otherwise, the statement on line 47 is generated.

- A similar structure appears on lines 26 through 28. In this case, however, the statement controlled by the IF is not a line to be generated into the macro expansion.
- Instead, it is another macro processor directive (**SET**). This **SET** statement assigns the value **1** to **&EORCK**.
- The symbol **&EORCK** is a ***macro-time variable*** *(also often called a* ***set symbol****), which can be used to store working* values during the macro expansion.
- Any symbol that begins with the character **&** and that is not a macro instruction parameter is assumed to be a **macro-time variable**.
- All such variables are initialized to a value of **0**. Thus if there is an argument corresponding to **&EOR** (that is, if **&EOR** is not null), the variable **&EORCK** is set to **1**.
- Otherwise, it retains its default value of **0**. The value of this **macro-time variable** is used in the conditional structures on lines 38 through 43 and 63 through 73.

Use of macro-time conditional statements (4.7 a)

```
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
```

- Figure 4.7(b-d) shows the expansion of three different macro invocation statements that illustrate the operation of the IF statements in Fig. 4.7(a).
- You should carefully work through these examples to be sure you understand how the given macro expansion was obtained from the macro definition and the macro invocation statement.
- The implementation of the conditional macro expansion features just described is relatively simple.
- The macro processor must maintain a symbol table that contains the values of all **macro-time variables** used.
- Entries in this table are made or modified when **SET** statements are processed.
- The table is used to look up the current value of a macro-time variable whenever it is required.

Use of macro-time conditional statements (4.7 b)

```
                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
```

Use of macro-time conditional statements (4.7 c)

```
                RDBUFF   0E,BUFFER,LENGTH,,80


30              CLEAR    X            CLEAR LOOP COUNTER
35              CLEAR    A
47              +LDT     #80          SET MAXIMUM RECORD LENGTH
50    $ABLOOP   TD       =X'0E'       TEST INPUT DEVICE
55              JEQ      $ABLOOP      LOOP UNTIL READY
60              RD       =X'0E'       READ CHARACTER INTO REG A
75              STCH     BUFFER,X     STORE CHARACTER IN BUFFER
80              TIXR     T            LOOP UNLESS MAXIMUM LENGTH
87              JLT      $ABLOOP        HAS BEEN REACHED
90    $ABEXIT   STX      LENGTH       SAVE RECORD LENGTH
```

Use of macro-time conditional statements (4.7 d)

```
                RDBUFF   F1,BUFF,RLENG,04


30              CLEAR    X              CLEAR LOOP COUNTER
35              CLEAR    A
40              LDCH     =X'04'         SET EOR CHARACTER
42              RMO      A,S
45             +LDT      #4096          SET MAX LENGTH = 4096
50    $ACLOOP   TD       =X'F1'         TEST INPUT DEVICE
55              JEQ      $ACLOOP        LOOP UNTIL READY
60              RD       =X'F1'         READ CHARACTER INTO REG A
65              COMPR    A,S            TEST FOR END OF RECORD
70              JEQ      $ACEXIT        EXIT LOOP IF EOR
75              STCH     BUFF,X         STORE CHARACTER IN BUFFER
80              TIXR     T              LOOP UNLESS MAXIMUM LENGTH
85              JLT      $ACLOOP          HAS BEEN REACHED
90    $ACEXIT   STX      RLENG          SAVE RECORD LENGTH
```

Keyword Macro Parameters

- All the macro instruction definitions we have seen thus far used ***positional parameters*** – parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statement.
- With **positional parameters**, the programmer must be careful to specify the arguments in the proper order.
- If an argument is to be omitted, the macro invocation statement must contain a null argument (two consecutive commas) to maintain the correct argument positions.
- **Positional parameters** are quite suitable for most macro instructions.
- However, if a macro has a large number of parameters, and only a few of these are given values in a typical invocation, a different form of parameter specification is more useful.
- Such a macro may occur in a situation in which a large and complex sequence of statements-perhaps even an entire operating system is to be generated from a .macro invocation.
- In such cases, most of the parameters may have acceptable default values; the macro invocation specifies only the changes from the default set of values.

- For example, suppose that a certain macro instruction **GENER** has 10 possible parameters, but in a particular invocation of the macro, only the third and ninth parameters are to be specified.
- If positional parameters were used, the macro invocation statement might look like:

  GENER , ,DIRECT ,,,,,,3
- Using a different form of parameter specification, called ***keyword parameters,*** each argument value is written with a keyword that names the corresponding parameter.
- Arguments may appear in any order.
- If the third parameter in the previous example is named **&TYPE** and the ninth parameter is named **&CHANNEL**, the macro invocation statement would be:

  GENER TYPE=DIRECT,CHANNEL=3
- This statement is obviously much easier to read, and much less error-prone, than the positional version.

- Figure 4.8(a) shows a version of the **RDBUFF** macro definition using keyword parameters.
- In the macro prototype, each parameter name is followed by an equal sign, which identifies a keyword parameter. After the equal sign, a default value is specified for some of the parameters.
- The parameter is assumed to have this default value if its name does not appear in the macro invocation statement.
- Thus the default value for the parameter **&INDEV** is **F1**. There is no default value for the parameter **&BUFADR**.
- Default values can simplify the macro definition in many cases.
- The default value established in Fig. 4.8(a) takes care of this automatically.
- The other parts of Fig. 4.8 contain examples of the expansion of keyword macro invocation statements.
- In Fig. 4.8(b), all the default values are accepted.
- In Fig. 4.8(c), the value of **&INDEV** is specified as **F3**, and the value of **&EOR** is specified as null.
- These values override the corresponding defaults.
- Note that the arguments may appear in any order in the macro invocation statement.

Use of keyword parameters in macro instructions (4.8 a)

```
25    RDBUFF    MACRO    &INDEV=F1,&BUFADR=,&RECLTH=,&EOR=04,&MAXLTH=4096
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
47             +LDT      #&MAXLTH       SET MAXIMUM RECORD LENGTH
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
```

Use of keyword parameters in macro instructions (4.8 b)

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


30                  CLEAR    X              CLEAR LOOP COUNTER
35                  CLEAR    A
40                  LDCH     =X'04'         SET EOR CHARACTER
42                  RMO      A,S
47                 +LDT      #4096          SET MAXIMUM RECORD LENGTH
50      $AALOOP     TD       =X'F1'         TEST INPUT DEVICE
55                  JEQ      $AALOOP        LOOP UNTIL READY
60                  RD       =X'F1'         READ CHARACTER INTO REG A
65                  COMPR    A,S            TEST FOR END OF RECORD
70                  JEQ      $AAEXIT        EXIT LOOP IF EOR
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
```

Use of keyword parameters in macro instructions (4.8 c)

```
        .       RDBUFF  RECLTH=LENGTH,BUFADR=BUFFER,EOR=,INDEV=F3
                .
30              CLEAR   X               CLEAR LOOP COUNTER
35              CLEAR   A
47              +LDT    #4096           SET MAXIMUM RECORD LENGTH
50      $ABLOOP TD      =X'F3'          TEST INPUT DEVICE
55              JEQ     $ABLOOP         LOOP UNTIL READY
60              RD      =X'F3'          READ CHARACTER INTO REG A
75              STCH    BUFFER,X        STORE CHARACTER IN BUFFER
80              TIXR    T               LOOP UNLESS MAXIMUM LENGTH
85              JLT     $ABLOOP           HAS BEEN REACHED
90      $ABEXIT STX     LENGTH          SAVE RECORD LENGTH
```

4.3: MACRO PROCESSOR DESIGN OPTIONS

- **Recursive Macro Expansion**
- **General-Purpose Macro Processors**