Chapter 4
Macro Processors

Source Code (with macro) → Macro Processor → Expanded Code → Compiler or Assembler → obj
4.1 Basic Macro Processor Functions

4.1.1 Macro Definition and Expansion

Fig. 4.1 shows an example of a SIC/XE program using macro instructions.

- RDBUFF and WRBUFF
- MACRO and MEND
- RDBUFF is name
- Parameters (參數) of the macro instruction, each parameter begins with the character &.
- Macro invocation (引用) statement and the arguments (引|數) to be used in expanding the macro.

Fig. 4.2 shows the output that would be generated.
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 HAS BEEN REACHED
JLT *-19
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
.
.
Figure 4.1 Use of macros in a SIC/XE program.
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Instruction 3</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>COPY</td>
<td>START</td>
<td>0</td>
<td>COPY FILE FROM INPUT TO OUTPUT</td>
</tr>
<tr>
<td>180</td>
<td>FIRST</td>
<td>ST1</td>
<td>RETADR</td>
<td>SAVE RETURN ADDRESS</td>
</tr>
<tr>
<td>190</td>
<td>.CLOOP</td>
<td>RDBUFF</td>
<td>F1, BUFFER, LENGTH</td>
<td>READ RECORD INTO BUFFER</td>
</tr>
<tr>
<td>190a</td>
<td>CLOOP</td>
<td>CLEAR</td>
<td>X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>190b</td>
<td></td>
<td>CLEAR</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>190c</td>
<td></td>
<td>CLEAR</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>190d</td>
<td>+LDT</td>
<td>#4096</td>
<td></td>
<td>SET MAXIMUM RECORD LENGTH</td>
</tr>
<tr>
<td>190e</td>
<td>TD</td>
<td>=X‘F1’</td>
<td></td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>190f</td>
<td>JEQ</td>
<td>*-3</td>
<td></td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>190g</td>
<td>RD</td>
<td>=X‘F1’</td>
<td></td>
<td>READ CHARACTER INTO REG A</td>
</tr>
<tr>
<td>190h</td>
<td>COMPR</td>
<td>A, S</td>
<td></td>
<td>TEST FOR END OF RECORD</td>
</tr>
<tr>
<td>190i</td>
<td>JEQ</td>
<td>*+11</td>
<td></td>
<td>EXIT LOOP IF EOR</td>
</tr>
<tr>
<td>190j</td>
<td>STCH</td>
<td>BUFFER, X</td>
<td></td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>190k</td>
<td>TIXR</td>
<td>T</td>
<td></td>
<td>LOOP UNLESS MAXIMUM LENGTH HAS BEEN REACHED</td>
</tr>
<tr>
<td>190l</td>
<td>JLT</td>
<td>*-19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>190m</td>
<td>STIX</td>
<td>LENGTH</td>
<td></td>
<td>SAVE RECORD LENGTH</td>
</tr>
<tr>
<td>195</td>
<td>LDA</td>
<td>LENGTH</td>
<td></td>
<td>TEST FOR END OF FILE</td>
</tr>
<tr>
<td>200</td>
<td>COMP</td>
<td>#0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>JEQ</td>
<td>ENDFIL</td>
<td></td>
<td>EXIT IF EOF FOUND</td>
</tr>
</tbody>
</table>
210          WRBUFF  05,BUFFER,LENGTH
210a         CLEAR  X
210b         LDT    LENGTH
210c         LDCH   BUFFER,X
210d         TD      =X’05’
210e         JEQ    *=3
210f         WD      =X’05’
210g         TIXR   T
210h         JLT    *=14
215          J      CLOOP
Figure 4.2  Program from Fig. 4.1 with macros expanded.
Source
STRG MACRO
  STA DATA1
  STB DATA2
  STX DATA3
MEND
.

Expanded source
.
.
.

  .

  .

  .

  .

  .

  .

  .

  .

  .

  .

  .

  .
4.1.2 Macro Processor Algorithm and Data Structures

- Two-pass macro processor
  - All macro definitions are processed during the first pass.
  - All macro invocation statements are expanded during the second pass.
  - Two-pass macro processor would not allow the body of one macro instruction to contain definitions of other macros.

- Such definitions of macros by other macros Fig. 4.3
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}
MACROX MACRO {Defines SIC/XE macros}
RDBUFF MACRO &INDEV,&BUFADR,&RECLTH
               {SIC/XE version}
.
.
MEND {End of RDBUFF}
.
.
WRBUFF MACRO &OUTDEV,&BUFADR,&RECLTH
               {SIC/XE version}
.
.
MEND {End of WRBUFF}
.
.
MEND {End of MACROX}

Figure 4.3 Example of the definition of macros within a macro body.
A one-pass macro processor that can alternate between macro definition and macro expansion.

- The definition of a macro must appear in the source program before any statements that invoke that macro.
- Inconvenience of the programmer.
- Macro definitions are stored in DEFTAB
- Comment lines are not entered the DEFTAB.
4.1.2 Macro Processor Algorithm and Data Structures

- The **macro names** are entered into **NAMTAB**, **NAMTAB** contains **two pointers** to the beginning and the end of the definition in **DEFTAB**.

- The third data structure is an argument table **ARGTAB**, which is used during the expansion of macro invocations.

- The arguments are stored in **ARGTAB** **according to their position** in the argument list.
4.1.2 Macro Processor Algorithm and Data Structures

Fig. 4.4 shows positions of the contents of these tables during the processing.

- Parameter &INDEV -> Argument ?1
- Parameter &BUFADR -> Argument ?2
- When the ?n notation is recognized in a line form DEFTAB, a simple indexing operation supplies the proper argument form ARGTAB.
4.1.2 Macro Processor Algorithm and Data Structures

- The macro processor algorithm itself is presented in Fig. 4.5.
  - The procedure PROCESSING
  - The procedure DEFINE
    - Called when the beginning of a macro definition is recognized, makes the appropriate entries in DEFTAB and NAMTAB.
  - The procedure EXPAND
    - Called to set up the argument values in ARGTAB and expand a macro invocation statement.
  - The procedure GETLINE
    - Called at several points in the algorithm, gets the next line to be processed.
  - EXPANDING is set to TRUE or FALSE.
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}
4.1.2 Macro Processor Algorithm and Data Structures

To solve the problem in Fig. 4.3, our DEFINE procedure maintains a counter named LEVEL.

- **MACRO** directive is read, the value of LEVEL is inc. by 1.
- **MEND** directive is read, the value of LEVEL is dec. by 1.
4.2 Machine-Independent Macro Processor Features

4.2.1 Concatenation of Macro Parameters

- Most macro processors allow parameters to be concatenated with other character strings.
  - A program contains one series of variables named by the symbols XA1, XA2, XA3, ..., another series named by XB1, XB2, XB3, ..., etc.
  - The body of the macro definition might contain a statement like

```
SUM Macro &ID1
LDA X&ID1
LDA X&ID2
LDA X&ID3
LDA X&IDS
```
4.2.1 Concatenation of Macro Parameters

- The beginning of the macro parameter is identified by the starting symbol &; however, the end of the parameter is not marked.
- The problem is that the end of the parameter is not marked. Thus X&ID1 may mean “X” + ID + “1” or “X” + ID1.
- In which the parameter &ID is concatenated after the character string X and before the character string 1.
4.2.1 Concatenation of Macro Parameters

- Most macro processors deal with this problem by providing a special concatenation operator (Fig. 4.6).
  - In SIC or SIC/XE, -> is used

```
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)
```
4.2.2 Generation of Unique Labels

- As we discussed in Section 4.1, it is in general not possible for the body of a macro instruction to contain labels of usual kind.
  - WRBUFF (line 135) is called twice.
  - Fig. 4.7 illustrates one technique for generating unique labels within a macro expansion.
  - Labels used within the macro body begin with the special character $.
  - Each symbol beginning with $ has been modified by replacing $ with $AA.
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.
4.2.2 Generation of Unique Labels

25 RBUFF 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

(a)
. 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

(b)

Figure 4.7 Generation of unique labels within macro expansion.
4.2.3 Conditional Macro Expansion

The use of one type of conditional macro expansion statement is illustrated in Fig. 4.8.

- The definition of RDBUFF has two additional parameters: &EOR and &MAXLTH.
- Macro processor directive SET
- This SET statement assigns the value 1 to &EORCK.
- The symbol &EORCK is a macro time variables, which can be used to store working values during the macro expansion.

- RDBUFF F3,BUF,RECL,04,2048
- RDBUFF 0E,BUFFER,LENGTH,,80
- RDBUFF F1,BUFF,RLENG,04
RDBUFF MACRO &INDEV, &BUFADR, &RECLTH, &EOR, &MAXLTH
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
IF (&MAXLTH EQ ' ')
  +LDT #4096 SET MAX LENGTH = 4096
ELSE
  +LDT #&MAXLTH SET MAXIMUM RECORD LENGTH
ENDIF
$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

(a)
. RDBUFF F3,BUF,RECL,04,2048

30 CLEAR X
35 CLEAR A
40 LDCH =X'04'
42 RMO A,S
47 +LDT #2048
50 $AALOOP TD =X'F3'
55 JEQ $AALOOP
60 RD =X'F3'
65 COMPR A,S
70 JEQ $AAEXIT
75 STCH BUF,X
80 TIXR T
85 JLT $AALOOP
90 $AAEXIT STX RECL

CLEAR LOOP COUNTER
SET EOR CHARACTER
SET MAXIMUM RECORD LENGTH
TEST INPUT DEVICE
LOOP UNTIL READY
READ CHARACTER INTO REG A
TEST FOR END OF RECORD
EXIT LOOP IF EOR
STORE CHARACTER IN BUFFER
LOOP UNLESS MAXIMUM LENGTH HAS BEEN REACHED
SAVE RECORD LENGTH

Figure 4.8 Use of macro-time conditional statements.
. 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

(c)
. 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 $ACEEXIT 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 $ACEEXIT STX RLENG SAVE RECORD LENGTH
A different type of conditional macro expansion statement is illustrated in Fig. 4.9.

- There is a list (00, 03, 04) corresponding to &EOR.
- %NITEMS is a macro processor function that returns as its value the number of members in an argument list.
- %NITEMS(&EOR) is equal to 3.
- &CTR is used to count the number of times the lines following the WHILE statement have been generated.
- Thus on the first iteration the expression &EOR[&CTR] on line 65 has the value 00 = &EOR[1]; on the second iteration it has the value 03, and so on.
- How to implement nesting WHILE structures?
RDBUFF MACRO &INDEV, &BUFADR, &RECLTH, &EOR

&EOCT SET %NITEMS (&EOR) ; CLEAR LOOP COUNTER

CLEAR X
CLEAR A

+LDT #4096 ; SET MAX LENGTH = 4096

$LOOP TD =X’&INDEV’ ; TEST INPUT DEVICE
JEQ $LOOP ; LOOP UNTIL READY
RD =X’&INDEV’ ; READ CHARACTER INTO REG A

&CTR SET 1
WHILE (&CTR LE &EOCT)
COMP =X’0000’&EOR[&CTR]’
JEQ $EXIT ; HAS BEEN REACHED

&CTR SET &CTR+1
ENDW

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

(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' READ CHARACTER INTO REG A
67 JEQ $AAEXIT
69 COMP =X'000003' STORE CHARACTER IN BUFFER
70 JEQ $AAEXIT
72 COMP =X'000004' LOOP UNLESS MAXIMUM LENGTH
74 JEQ $AAEXIT HAS BEEN REACHED
75 STCH BUFFER,X
77 TIXR T
80 JLT $AALOOP SAVE RECORD LENGTH
85 $AAEXIT STX LENGTH

(b)
4.2.4 Keyword Macro Parameters

- **Positional parameters**
  - Parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statements.
  - A certain macro instruction GENER has 10 possible parameters.
    
    GENER MACRO &1, &2, &type, ..., &channel, &10

    GENER , , DIRECT, , , , , , 3
4.2.4 Keyword Macro Parameters

Keyword parameters

- Each argument value is written with a keyword that names the corresponding parameter.
- Arguments may appear in any order.

```
GENER , , DIRECT, , , , , , 3
GENER TYPE=DIRECT, CHANNEL=3
GENER CHANNEL=3, TYPE=DIRECT
```

- Fig. 4.10 shows a version of the RDBUFF using keyword.
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
Figure 4.10 Use of keyword parameters in macro instructions.
.  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    $ABLOOP  TD  =$'F3'  TEST INPUT DEVICE
55    JEQ  $ABLOOP  LOOP UNTIL READY
60    RD  =$'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

(c)

Figure 4.10 (cont’d)
4.3 Macro Processor Design Options
4.3.1 Recursive Macro Expansion

- In Fig. 4.3 we presented an example of the definition of one macro instruction by another.
- Fig. 4.11(a) shows an example - Dealt with the invocation of one macro by another.
- The purpose of RDCHAR Fig. 4.11(b) is to read one character from a specified device into register A, taking care of the necessary test-and-wait loop.
5   RDCHAR MACRO &IN
10   .
15   . MACRO TO 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

(b)

RDBUFF BUFFER, LENGTH, F1
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 MAXIMUM RECORD LENGTH
50 $LOOP RDCHAR &INDEV READ CHARACTER INTO REG A
55 COMPR A,S TEST FOR END OF RECORD
60 JEQ $EXIT EXIT LOOP IF EOR
65 STCH &BUFADR,X STORE CHARACTER IN BUFFER
70 TIXR T LOOP UNLESS MAXIMUM LENGTH
75 JLT $LOOP HAS BEEN REACHED
80 $EXIT STX &RECLTH SAVE RECORD LENGTH
85 MEND
4.3.1 Recursive Macro Expansion

- Fig. 4.11(c), applied to the macro invocation statement
  `RDBUFF BUFFER, LENGTH, F1`
- The procedure EXPAND would be called when the macro was recognized.
- The arguments from the macro invocation would be entered into ARGTAB as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BUFFER</td>
</tr>
<tr>
<td>2</td>
<td>LENGTH</td>
</tr>
<tr>
<td>3</td>
<td>F1</td>
</tr>
<tr>
<td>4</td>
<td>(unused)</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
4.3.1 Recursive Macro Expansion

- The Boolean variable EXPANDING would be set to TRUE, and expansion of the macro invocation statement would begin.
- The processing would proceed normally until line 50, which contains a statement invoking RDCHAR. At that point, PROCESSLINE would call EXPAND again.
- This time, ARGTABLE would look like

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F1</td>
</tr>
<tr>
<td>2</td>
<td>(unused)</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
4.3.1 Recursive Macro Expansion

- At the end of this expansion, however, a problem would appear. When the end of the definition of RDCHAR was recognized, EXPANDING would be set to FALSE.

- Thus the macro processor would “forget” that it had been in middle of expanding a macro when it encountered the RDCHAR statement.

- Use a Stack to save ARGTAB.

- Use a counter to identify the expansion.
48

Pages 208-209, MASM

1       ABSDIF     MACRO     OP1,OP2,SIZE
2       LOCAL     EXIT
3       IFNB     <SIZE> ; ; IF SIZE IS NOT BLANK
4       IFDIF     <SIZE>,<E> ; ; THEN IT MUST BE E
5       ; ERROR -- SIZE MUST BE E OR BLANK
6       .ERR
7       .EXITM
8       ENDF       ; ; END OF IFDIF
9       ENDF       ; ; END OF IFNB
10      MOV SIZE&AX,OP1 ; ; COMPUTE ABSOLUTE DIFFERENCE
11      SUB SIZE&AX,OP2 ; ; SUBTRACT OP2 FROM OP1
12      JNS EXIT ; ; EXIT IF RESULT GE 0
13      NEG SIZE&AX ; ; OTHERWISE CHANGE SIGN
14      EXIT:
15      ENDM

(a)
ABSDIF    J, K

MOV       AX, J       ; COMPUTE ABSOLUTE DIFFERENCE
SUB       AX, K
JNS       ??0000
NEG       AX

??0000:

(b)

ABSDIF    M, N, E

MOV       EAX, M       ; COMPUTE ABSOLUTE DIFFERENCE
SUB       EAX, N
JNS       ??0001
NEG       EAX

??0001:

(c)
ABSDIF P, Q, X

; ERROR -- SIZE MUST BE E OR BLANK

(d)

Figure 4.12 Examples of MASM macro and conditional statements.
Figure 4.13   Example of MASM iteration statement.