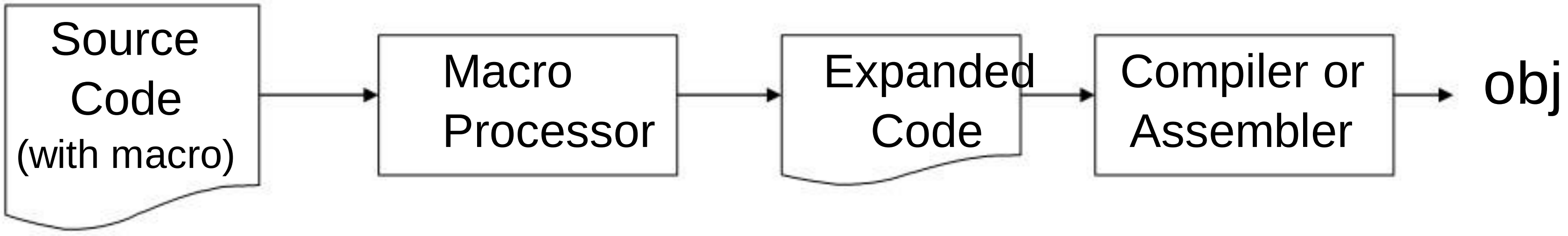
Macro Processors



Source Code (with macro)

Macro Processor

Expanded Code

Compiler or Assembler

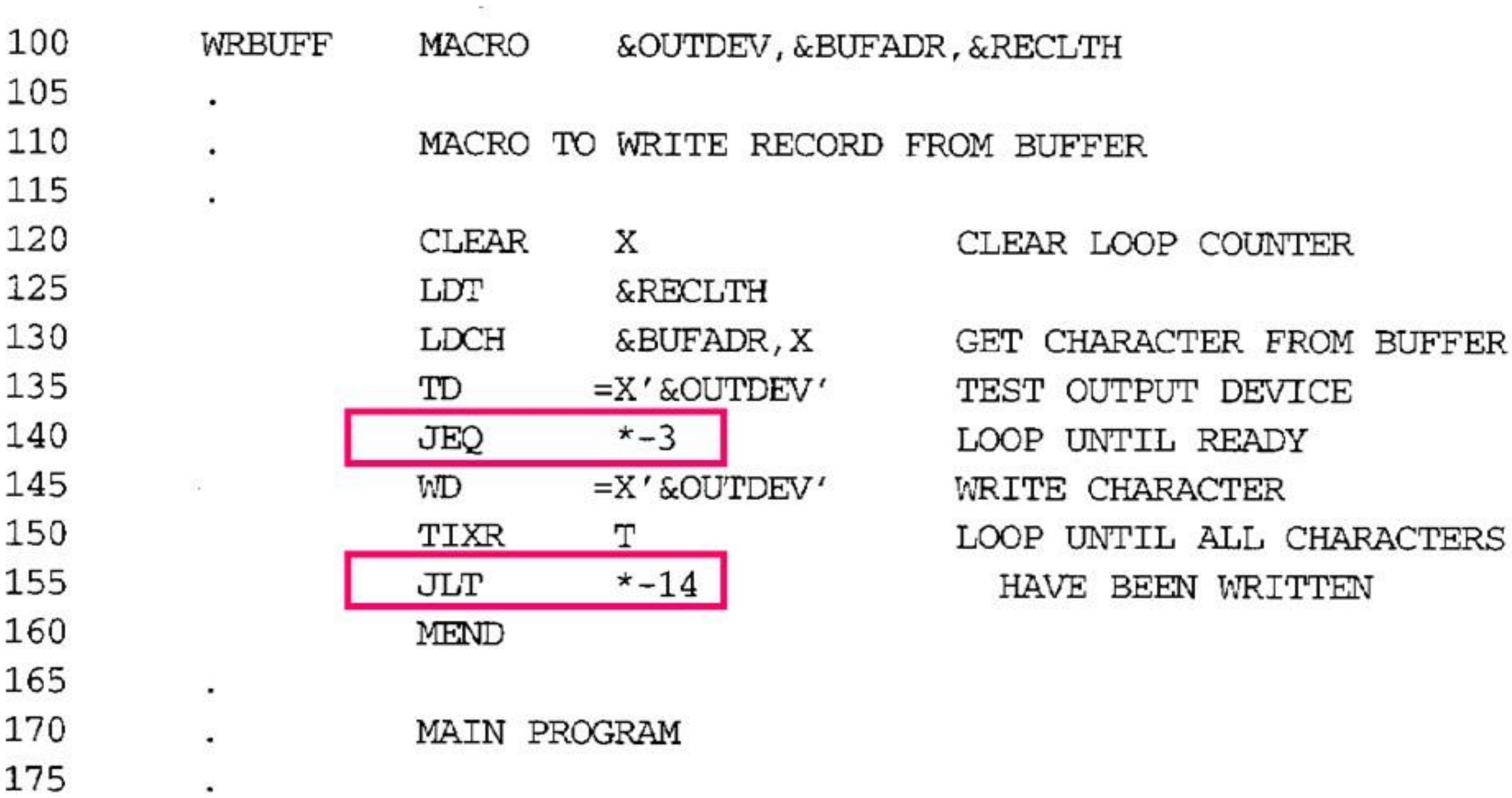
obj

4.1 Basic Macro Processor Functions

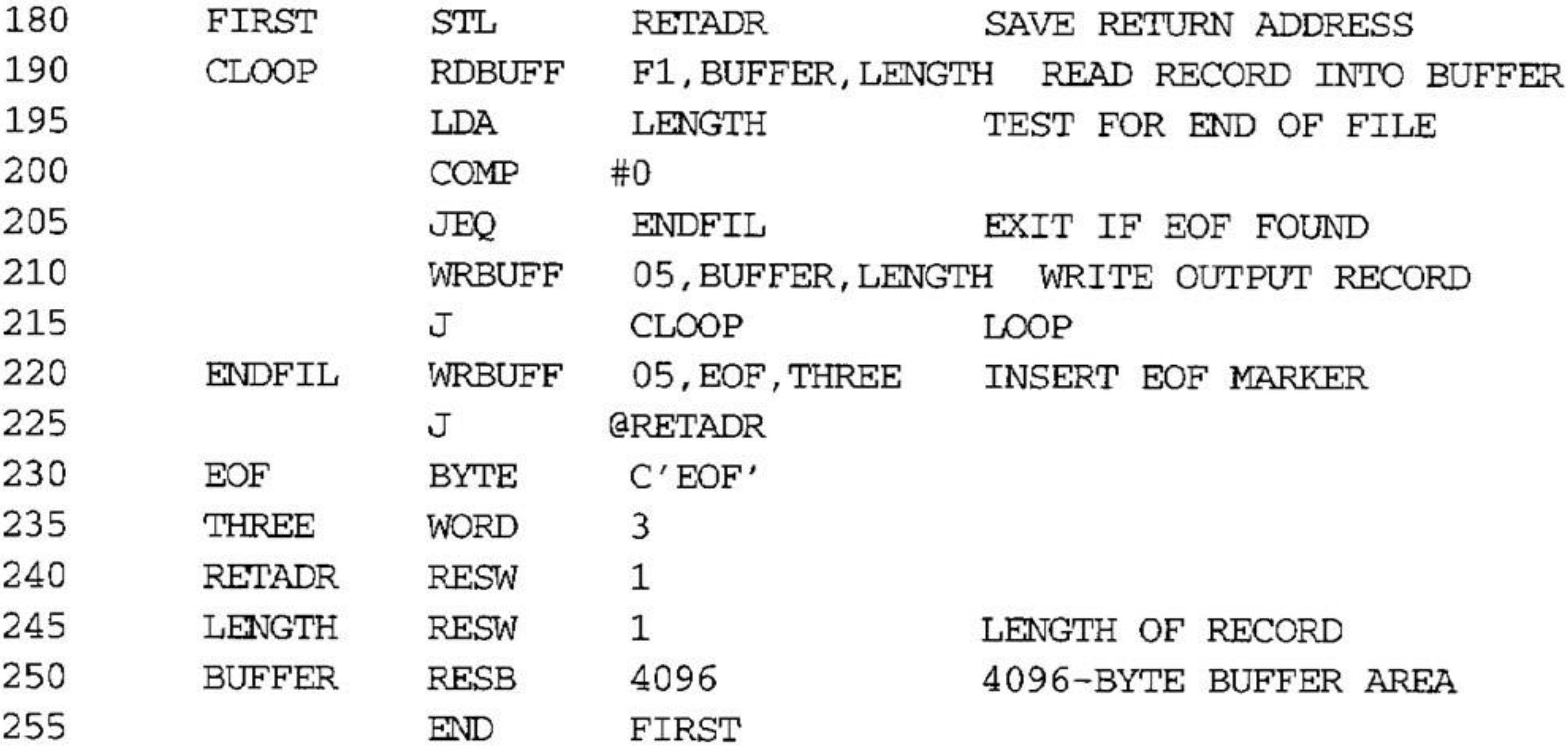
4.1.1 Macro Definition and Expansion

- Fig. 4.1 shows an example of a SIC/XE program using macro instructions.
    - RDBUFF and WRBUFF
    - MACRO and MEND
    - RDBUFF is name
    - *Parameters* of the macro instruction, each parameter begins with the character &.
    - Macro invocation statement and the arguments 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
         JLT      *-19                 HAS BEEN REACHED
         STX      &RECLTH            SAVE RECORD LENGTH
         MEND
```

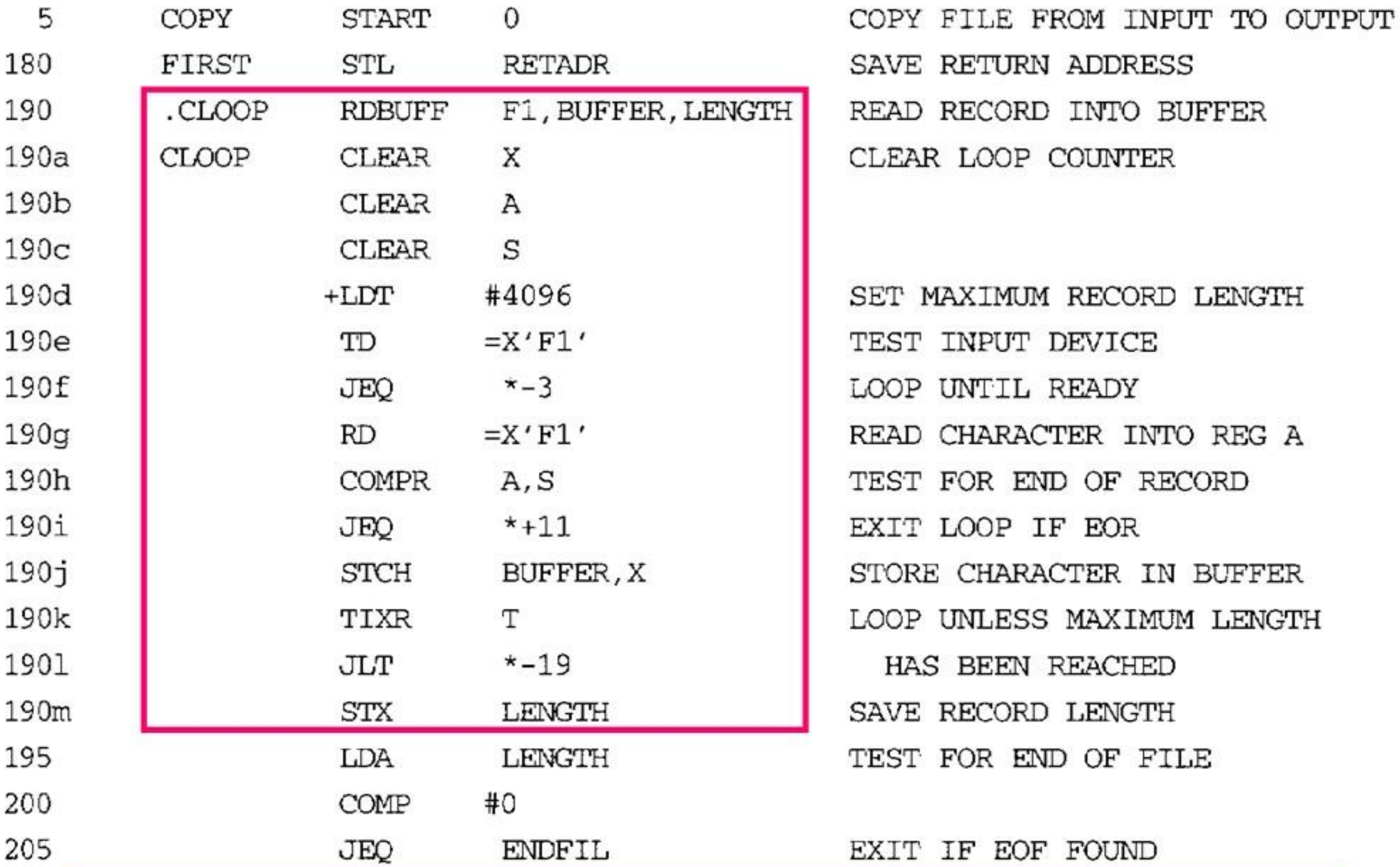


```
100     WRBUFF    MACRO     &OUTDEV,&BUFADR,&RECLTH
105     .
110     .         MACRO TO WRITE RECORD FROM BUFFER
115     .
120               CLEAR     X                    CLEAR LOOP COUNTER
125               LDT       &RECLTH
130               LDCH      &BUFADR,X            GET CHARACTER FROM BUFFER
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     .
```

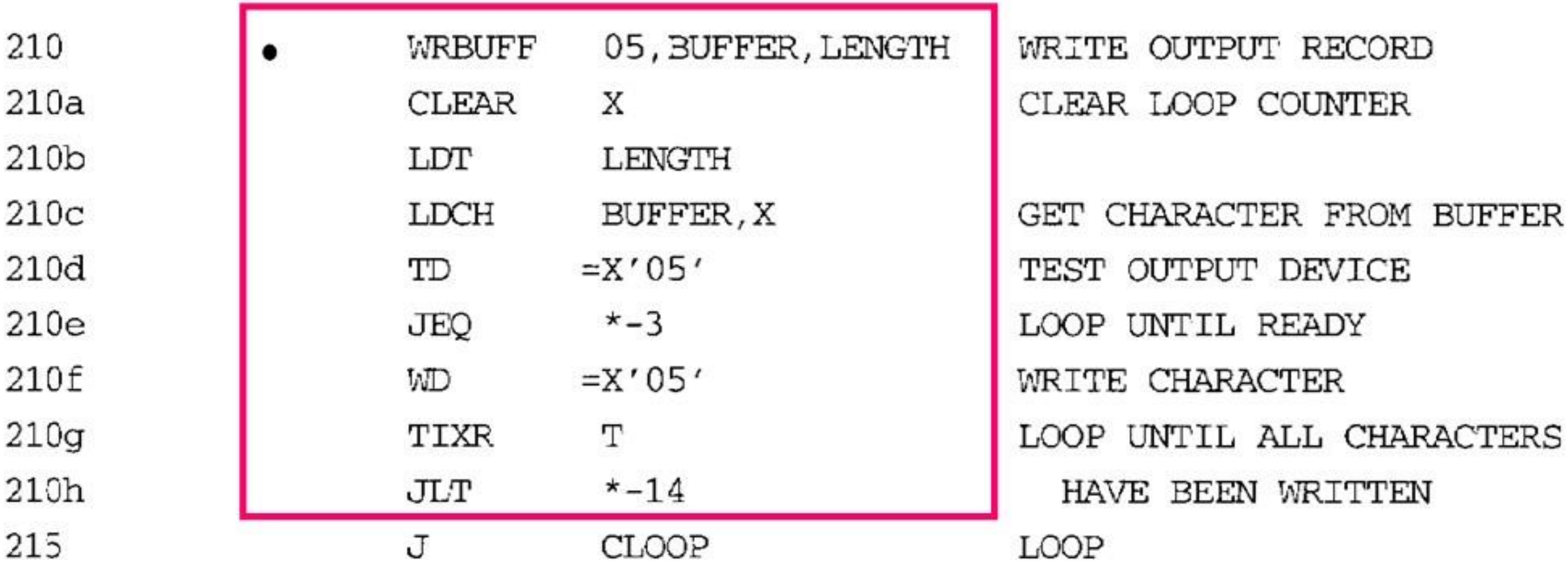


```
FIRST    STL      RETADR            SAVE RETURN ADDRESS
CLOOP    RDBUFF   F1,BUFFER,LENGTH  READ RECORD INTO BUFFER
         LDA      LENGTH            TEST FOR END OF FILE
         COMP    #0
         JEQ      ENDFIL            EXIT IF EOF FOUND
         WRBUFF   05,BUFFER,LENGTH  WRITE OUTPUT RECORD
         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
```

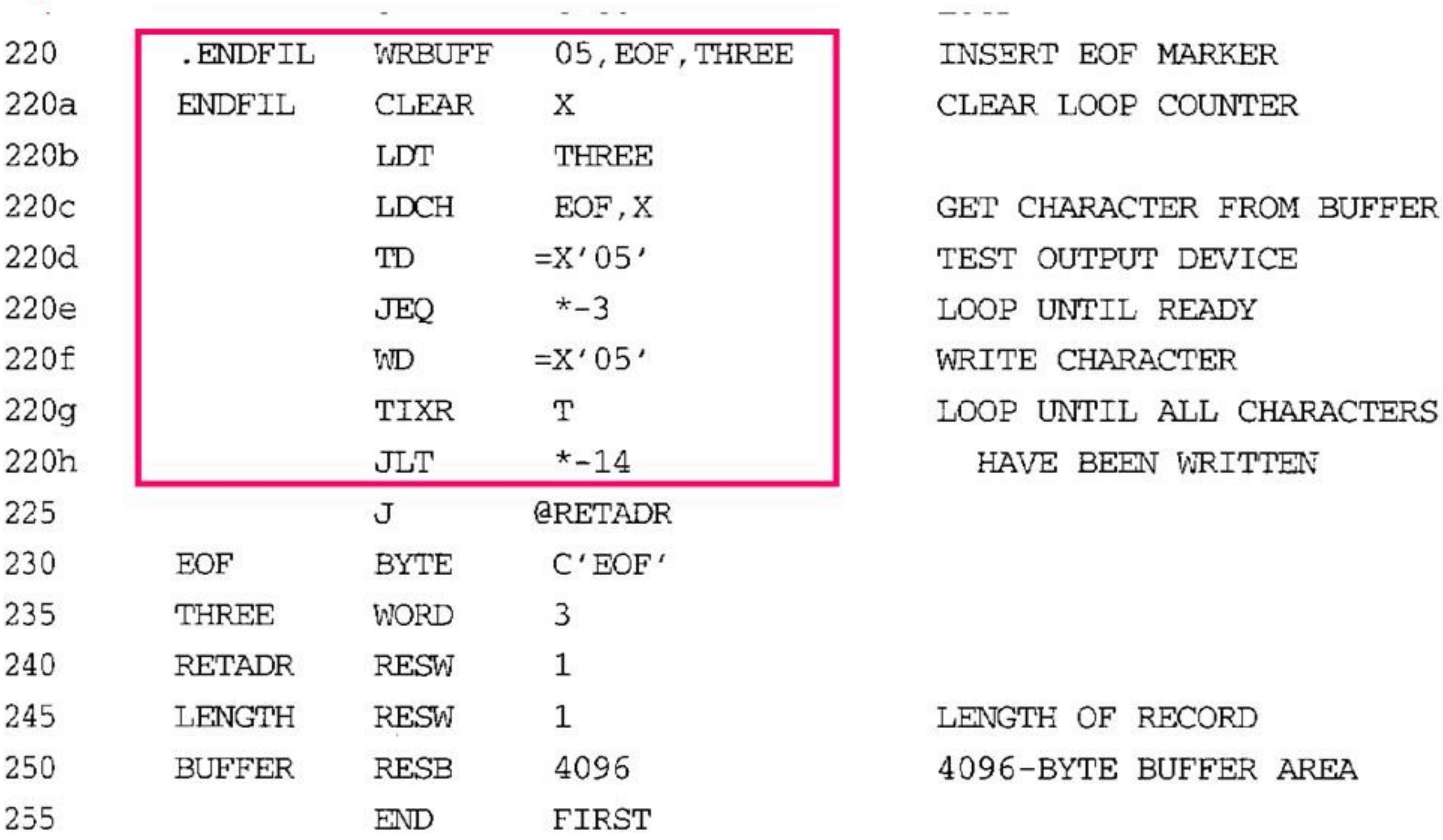
Figure 4.1 Use of macros in a SIC/XE program.



```
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
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 Program from Fig. 4.1 with macros expanded.

Source

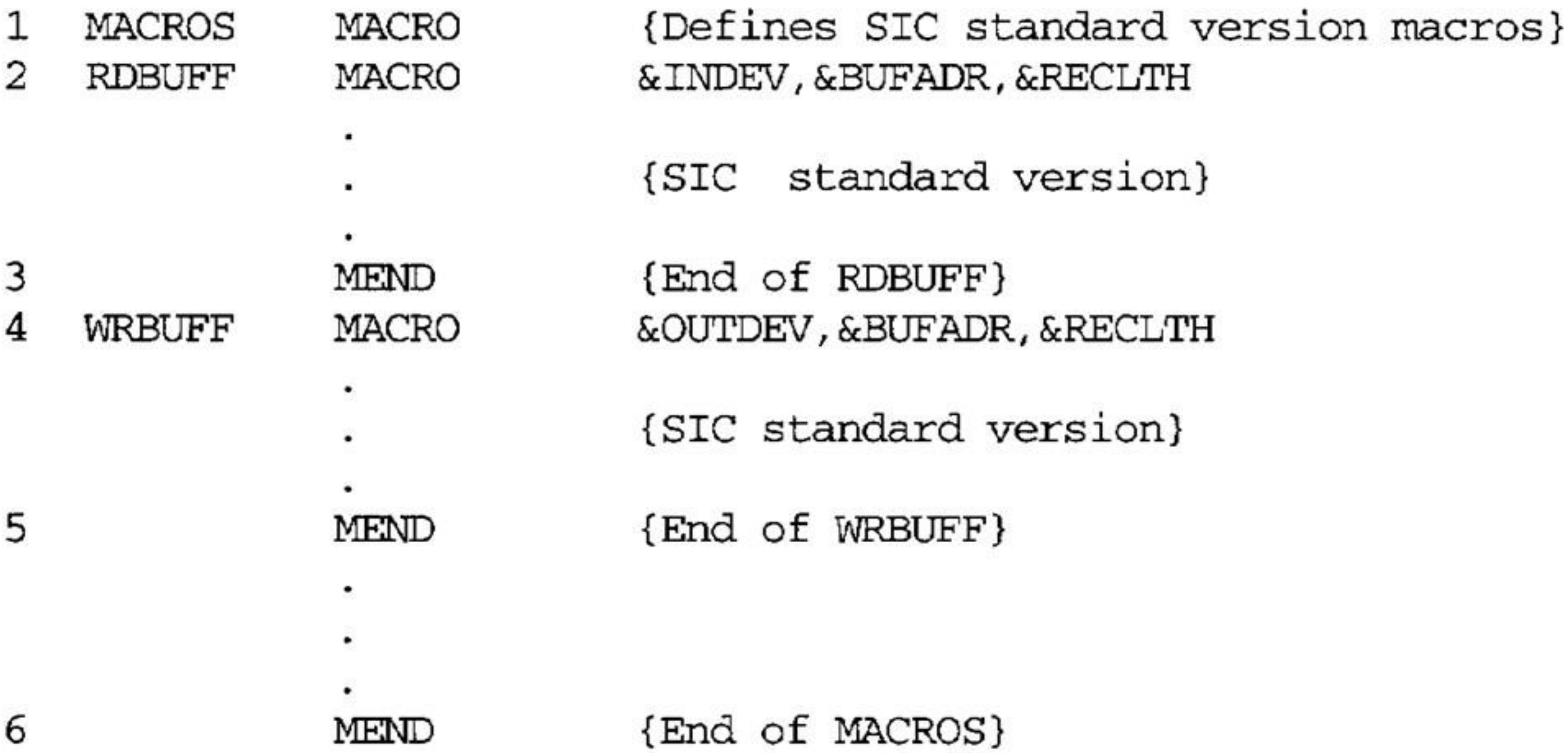
```
STRG    MACRO
       STA    DATA1
       STB    DATA2
       STX    DATA3
       MEND

STRG

STRG
```

Expanded source

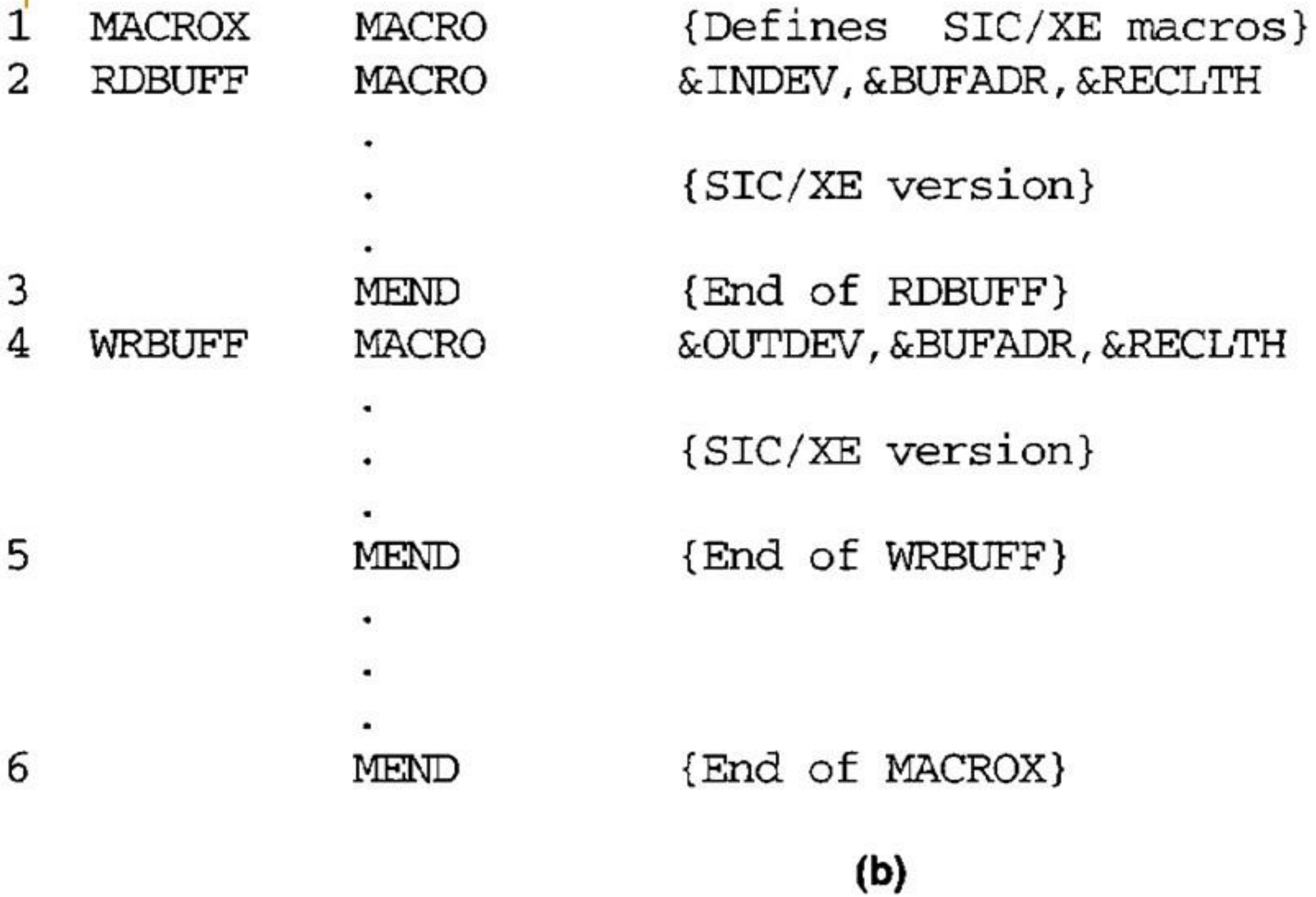
```
   .STRG
        STA    DATA1
      { STB    DATA2
        STX    DATA3
   .STRG
        STA    DATA1
      { STB    DATA2
         STX    DATA3
```

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}
```



```
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.

4.1.2 Macro Processor Algorithm and Data Structures

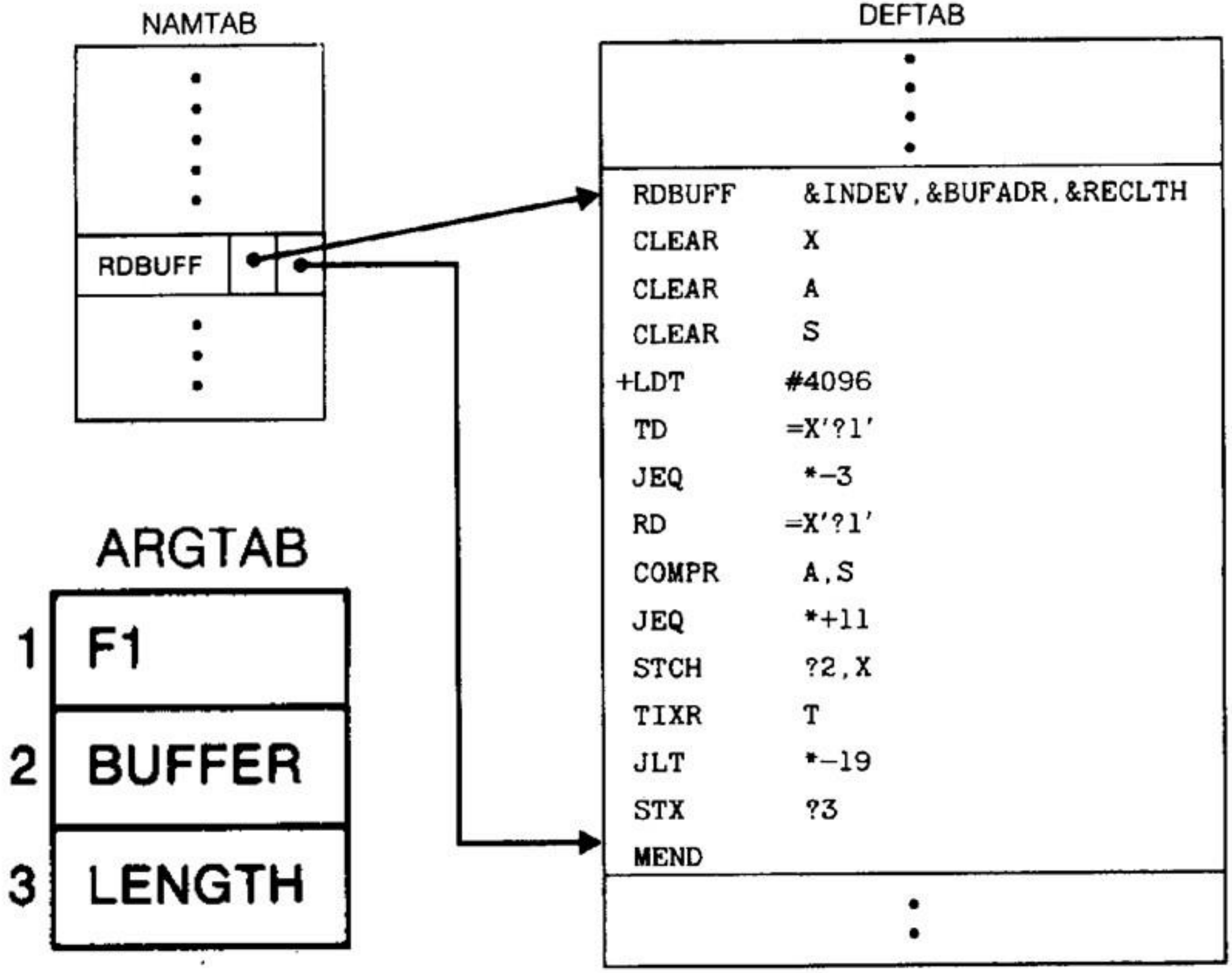
- A one-pass macro processor that can alternate between macro definition and macro expansion.
  - The definition of a macro must appear in the source program before any statements that invoke that macro.
  - Inconvenience of the programmer.
  - Macro definitions are stored in DEFTAB
  - Comment lines 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.



```
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
```

ARGTAB

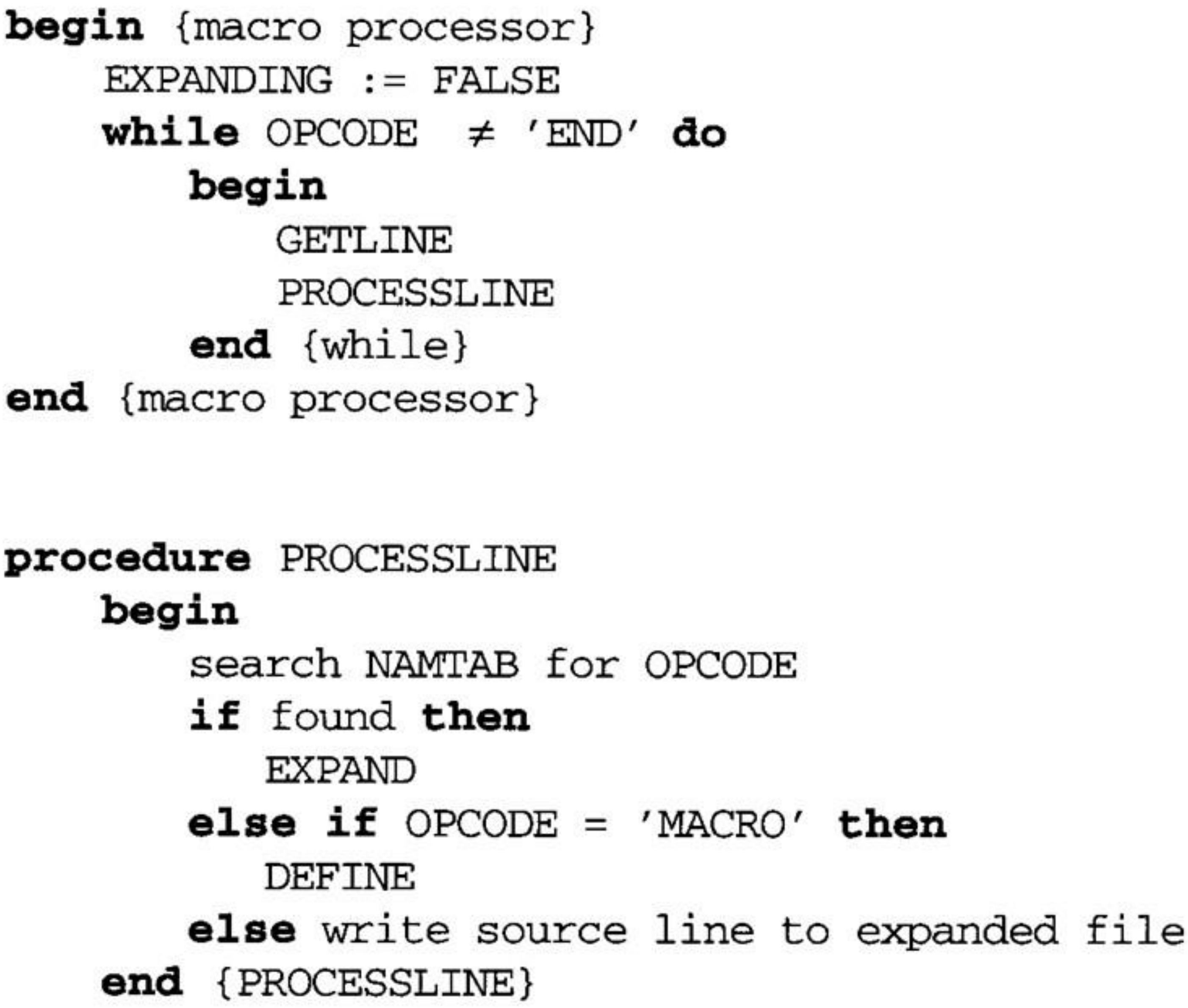
1	F1
2	BUFFER
3	LENGTH

ARGTAB (b)

(a)

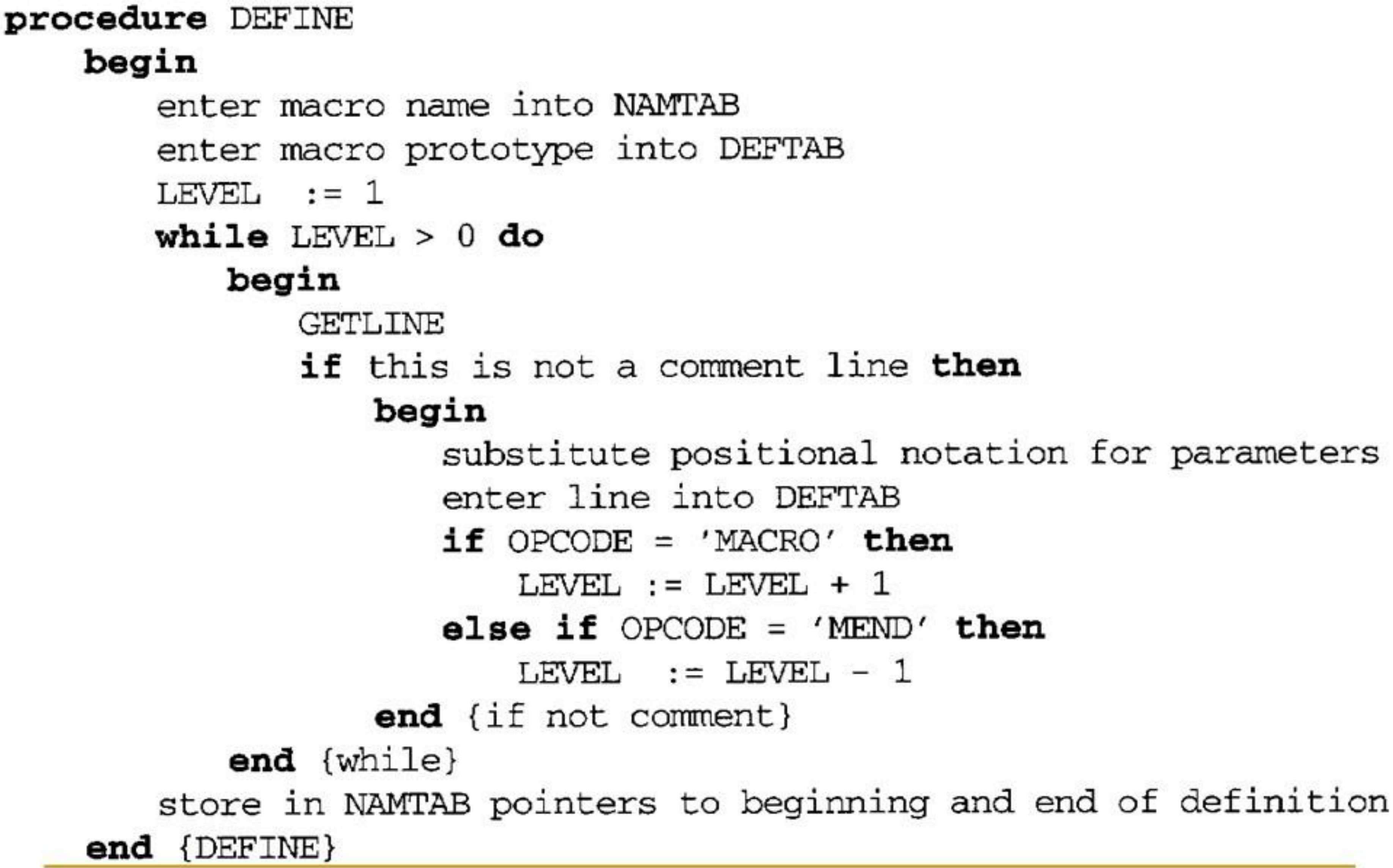
4.1.2 Macro Processor Algorithm and Data Structures

- 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.

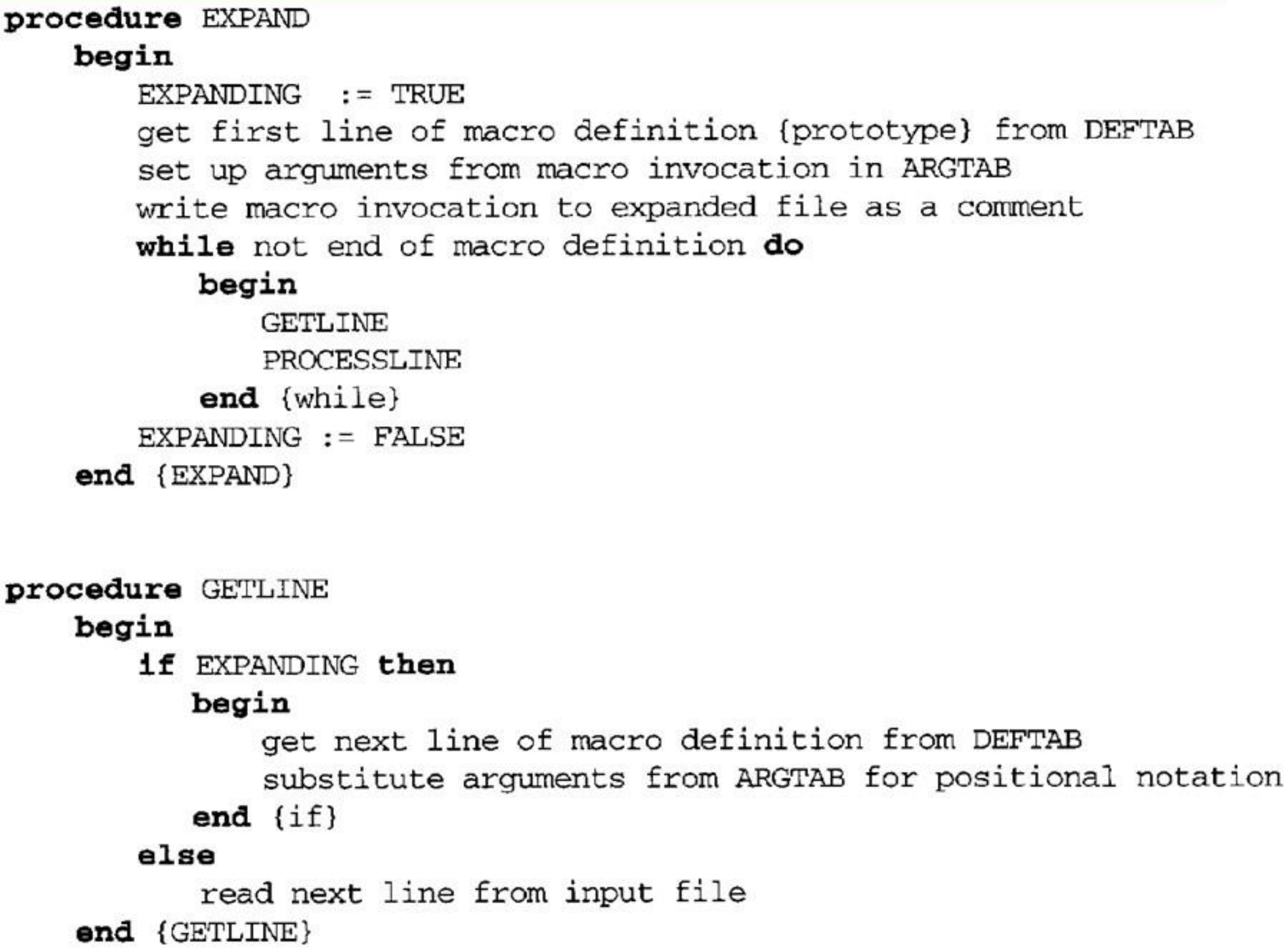


```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE ≠ 'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
end {macro processor}


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}
```

Figure 4.5 *(cont'd)*

4.1.2 Macro Processor Algorithm and Data Structures

④ To solve the problem is Fig. 4.3, our DEFINE procedure maintains a counter named LEVEL.

- ⑨ MACRO directive is read, the value of LEVEL is inc. by 1.
- ⑨ 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

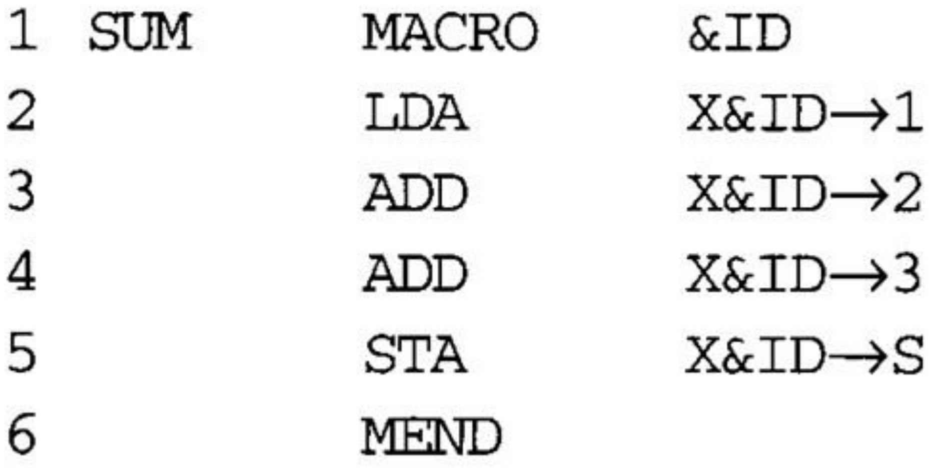
```
SUM     Macro  &ID
         LDA    X&ID1
         LDA    X&ID2
         LDA    X&ID3
         LDA    X&IDS
```

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

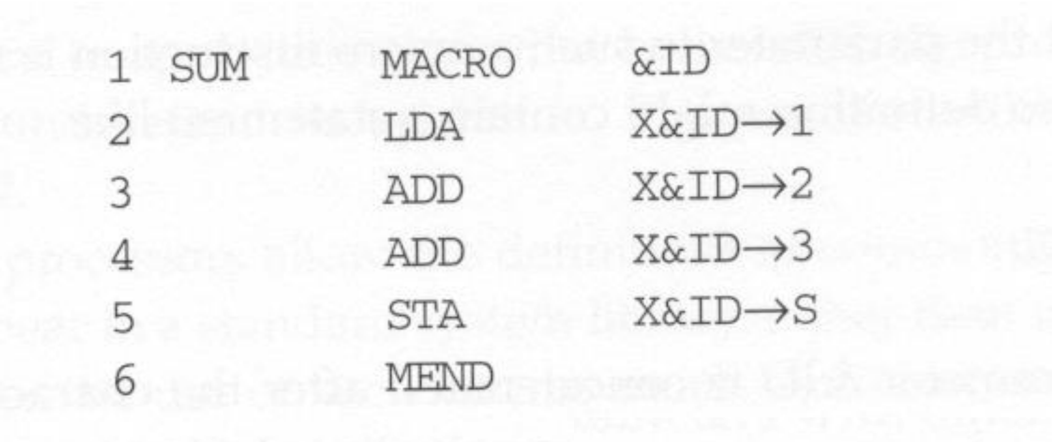
- Most macro processors deal with this problem by providing a special concatenation operator (Fig. 4.6)
  - In SIC or SIC/XE, -> is used



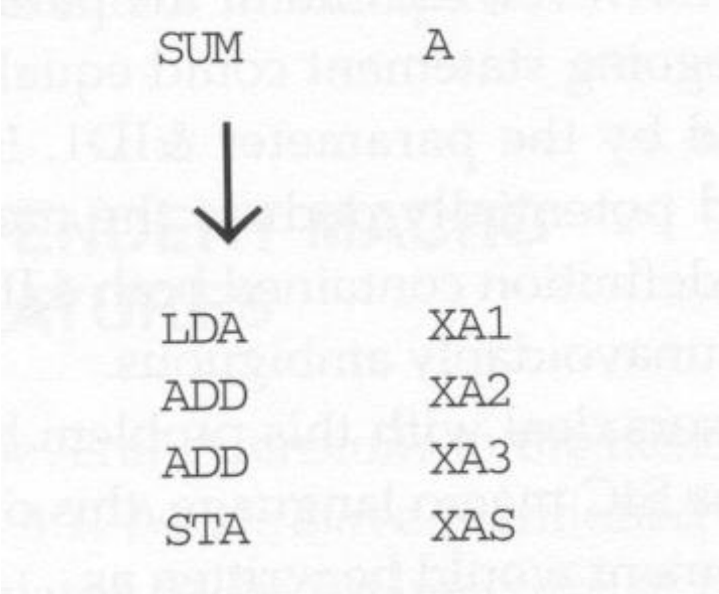
```
SUM      MACRO    &ID
         LDA      X&ID→1
         ADD      X&ID→2
         ADD      X&ID→3
         STA      X&ID→S
         MEND
```

(a)

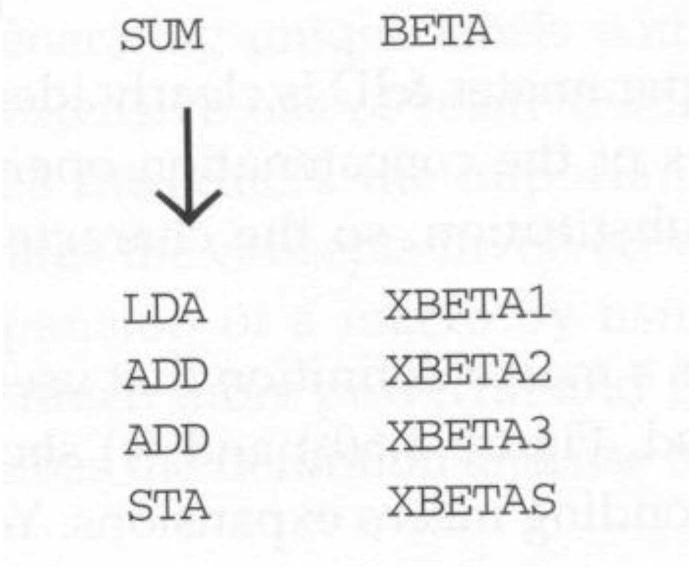
Concatenation Example



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

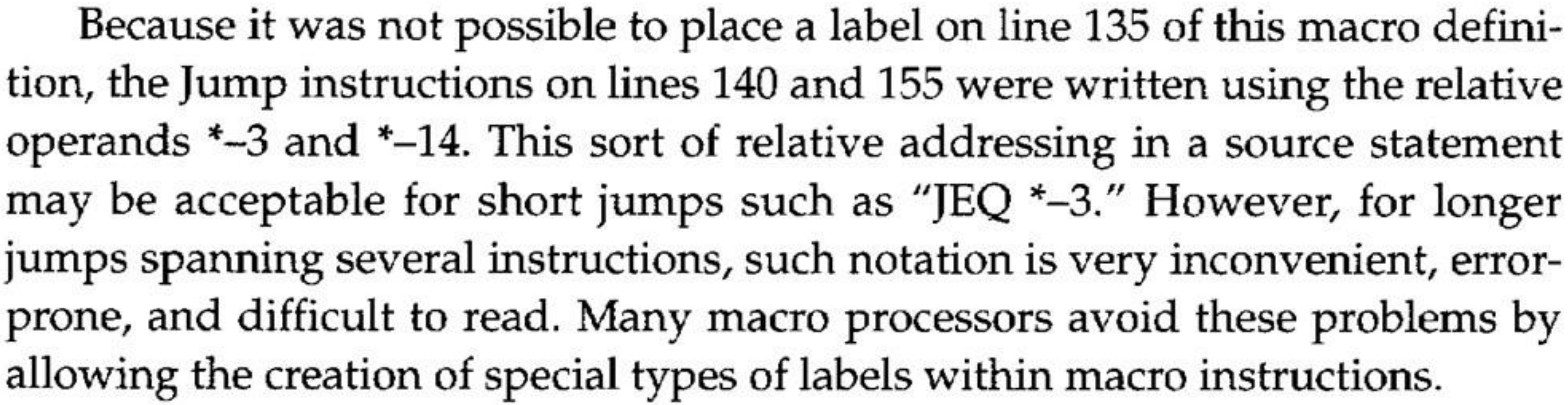


```
SUM      BETA
   ↓
LDA      XBETA1
ADD      XBETA2
ADD      XBETA3
STA      XBETAS
```

4.2.2 Generation of Unique Labels

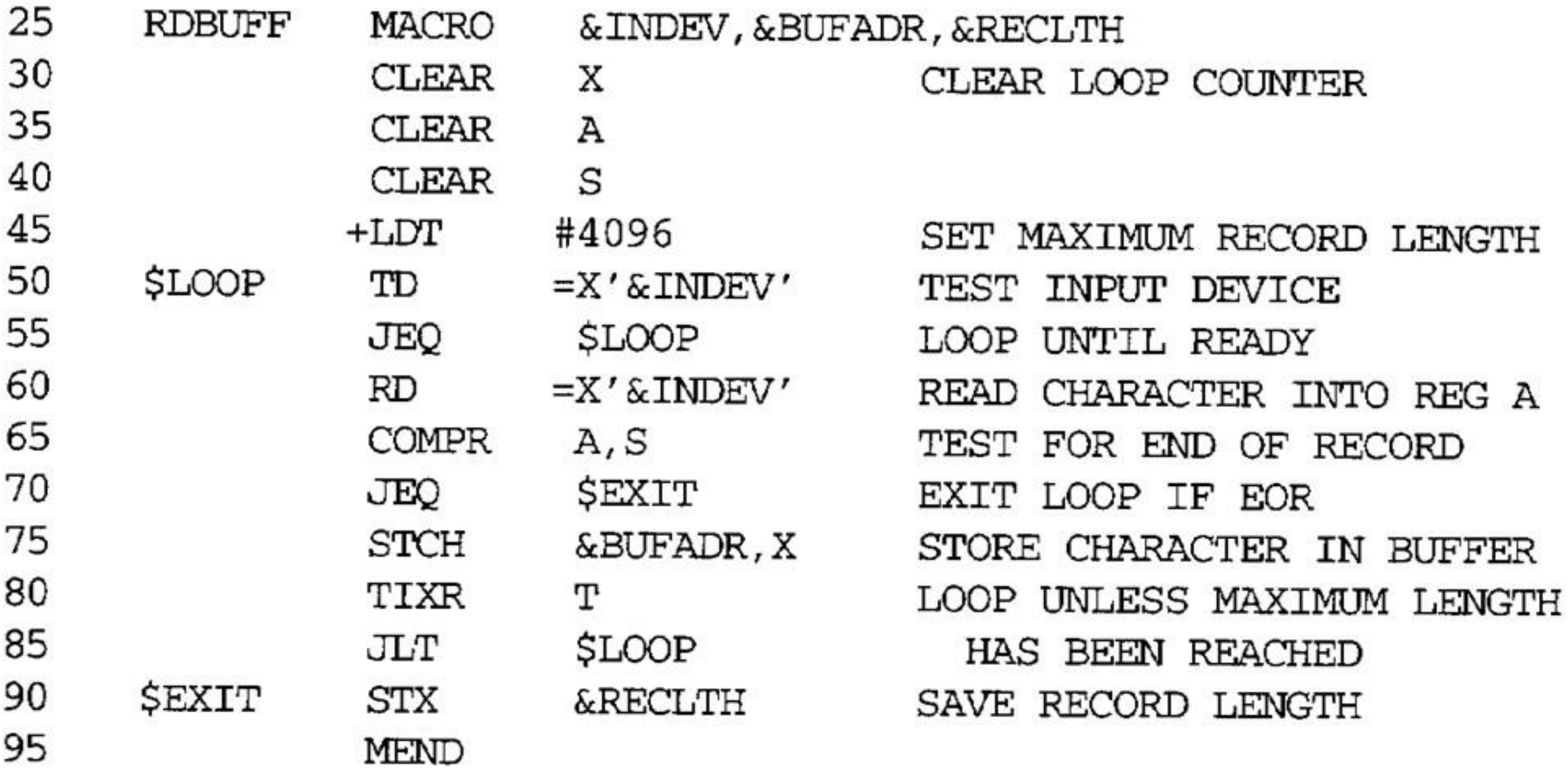
- As we discussed in Section 4.1, it is in general not possible for the body of a macro instruction to contain labels of usual kind.
  - WRBUFF (line 135) is called twice.
  - Fig. 4.7 illustrates one techniques for generating unique labels within a macro expansion.
  - Labels used within the macro body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

4.2.2 Generation of Unique Labels



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



```
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 INTO REG A
         COMPR   A,S             TEST FOR END OF RECORD
         JEQ     $EXIT           EXIT LOOP IF EOR
         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)

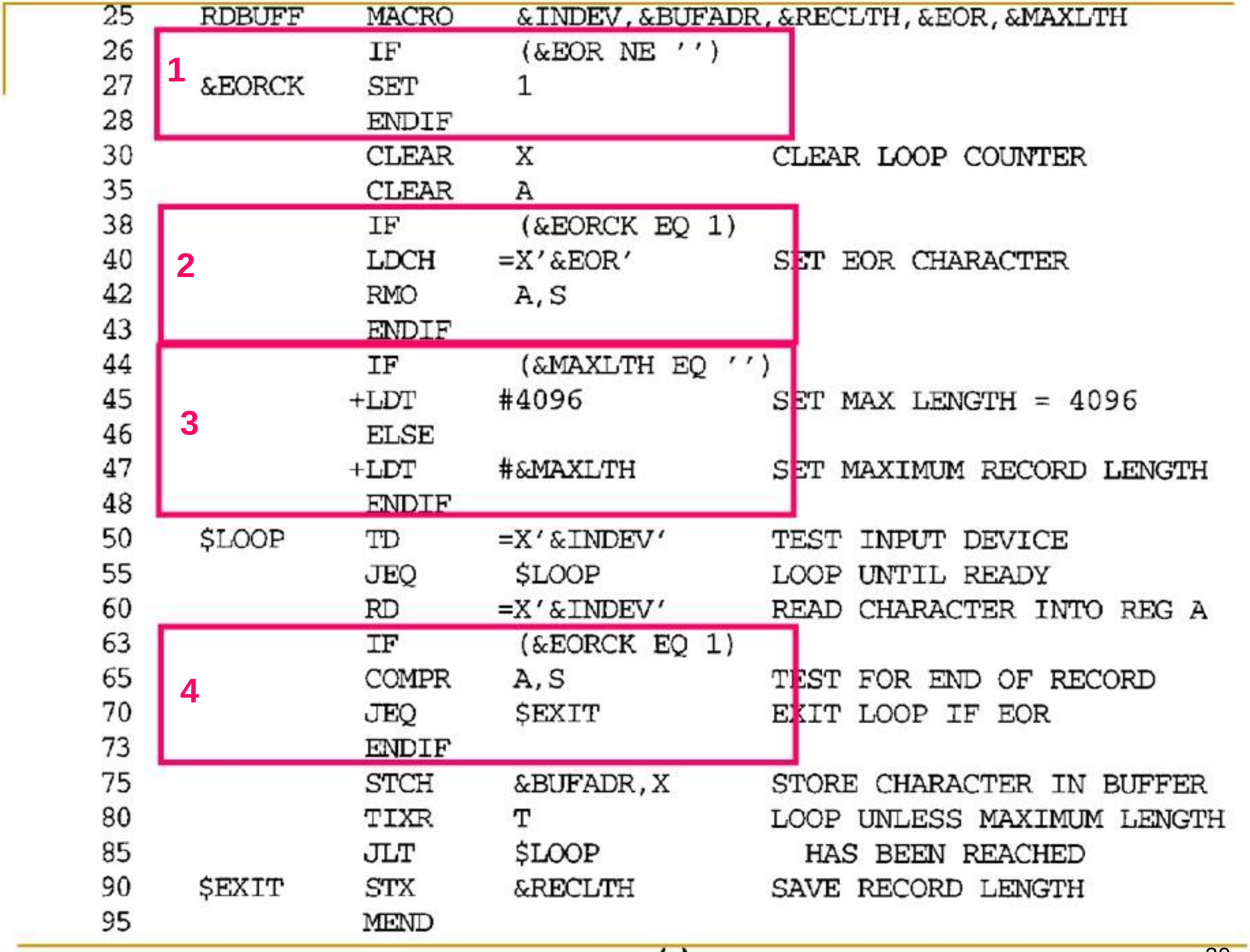
```
.               RDBUFF   F1,BUFFER,LENGTH


                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 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 4.7 Generation of unique labels within macro expansion.

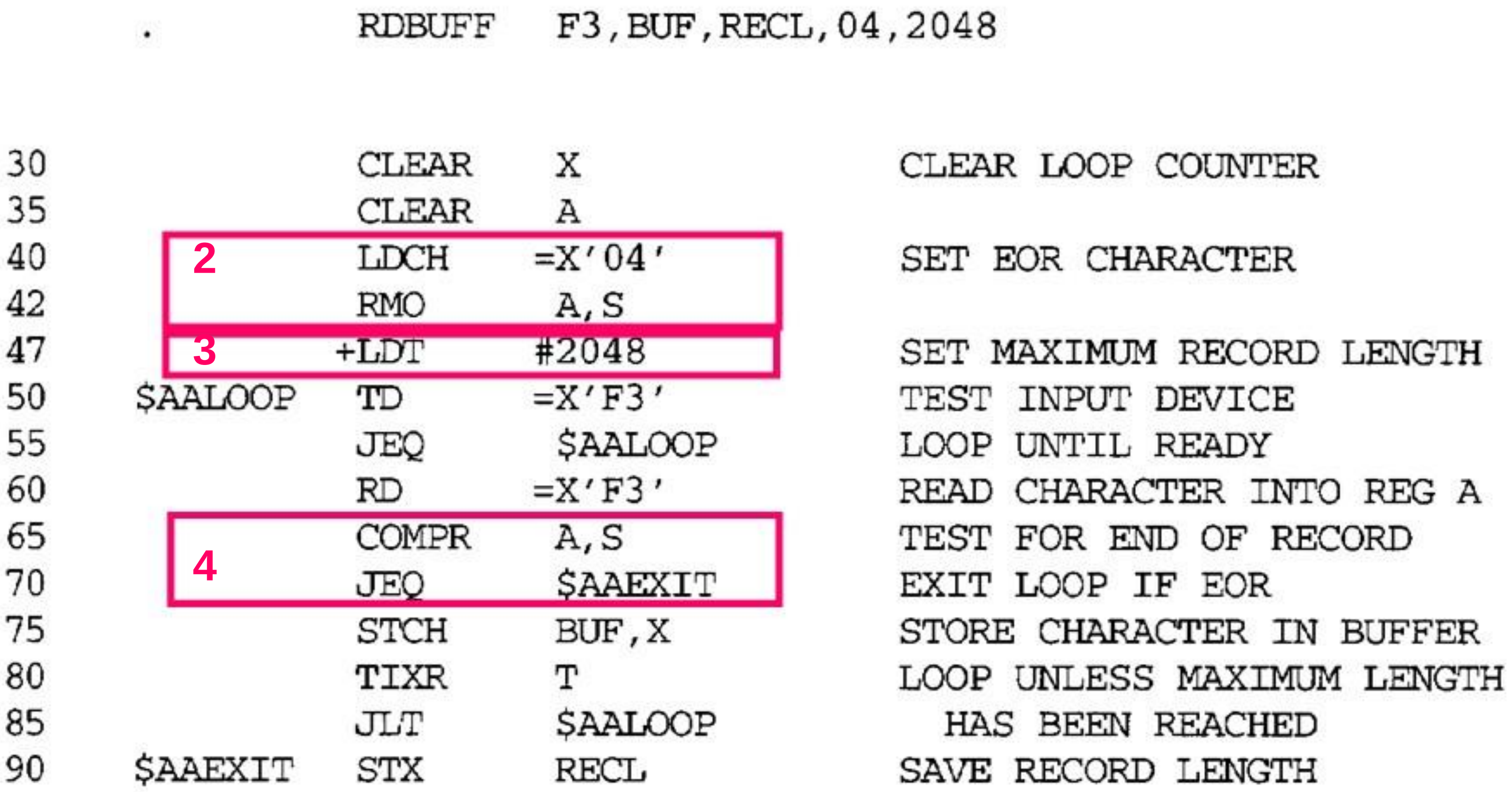
4.2.3 Conditional Macro Expansion

- 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`



```
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
```

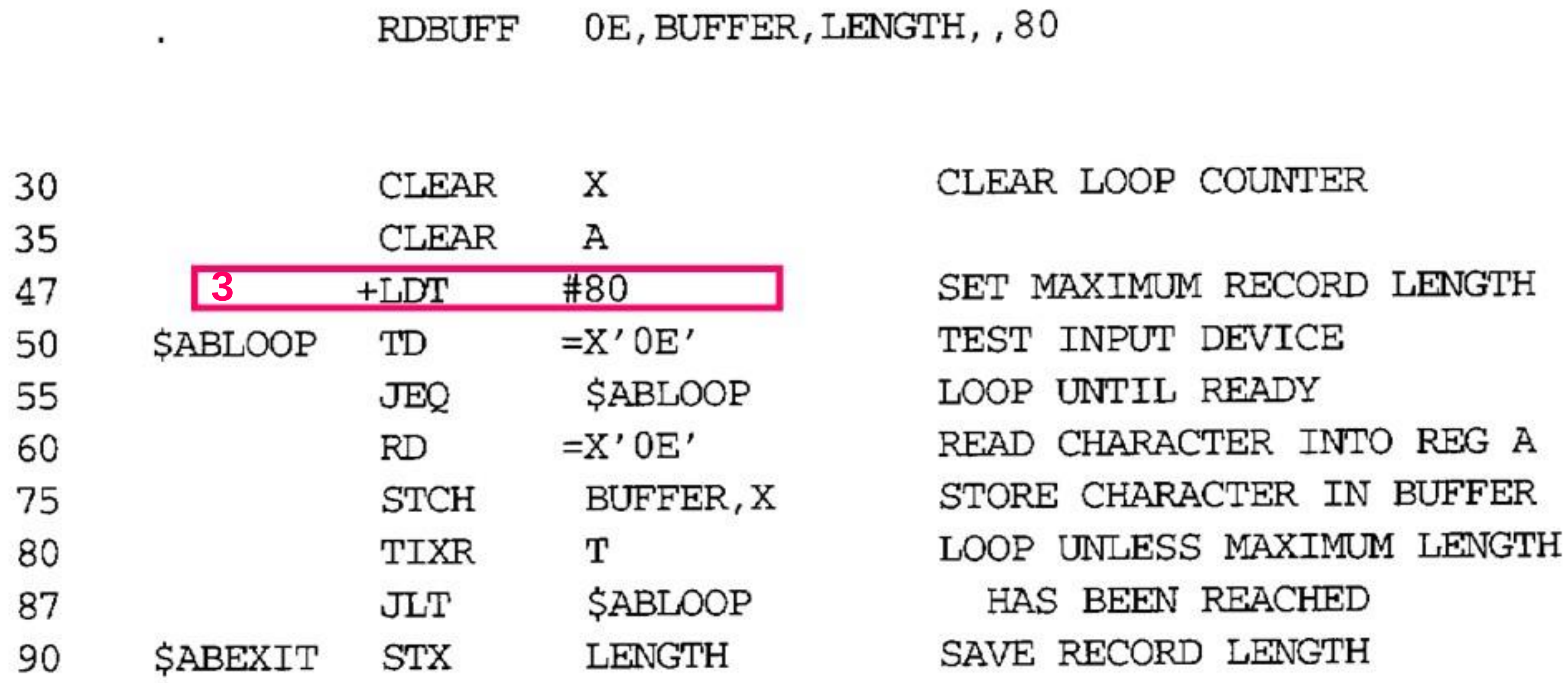
(a)



```
.             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
```

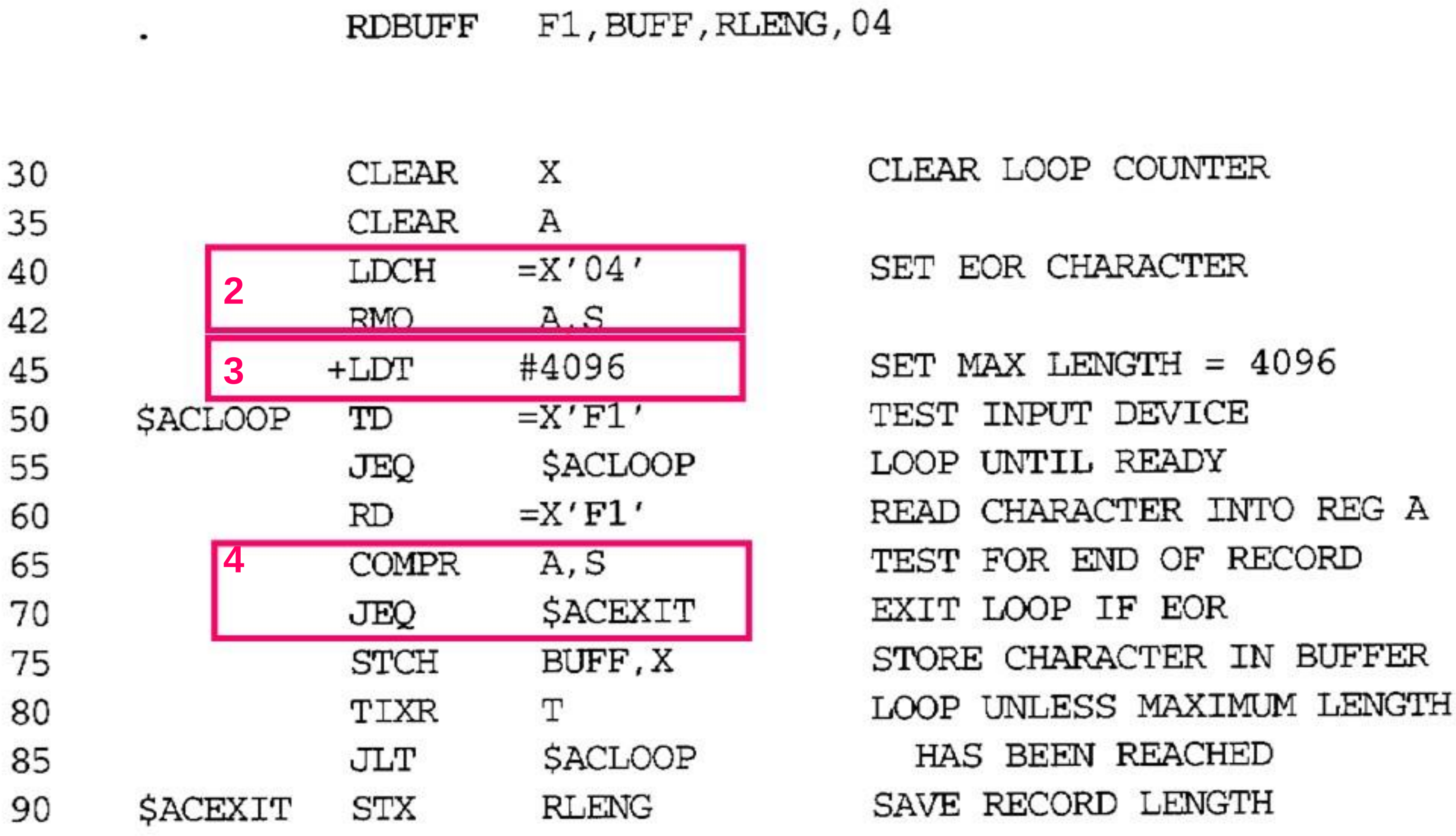
(b)

Figure 4.8 Use of macro-time conditional statements.



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

30                  CLEAR      X                        CLEAR LOOP COUNTER
35                  CLEAR      A
47         3        +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)

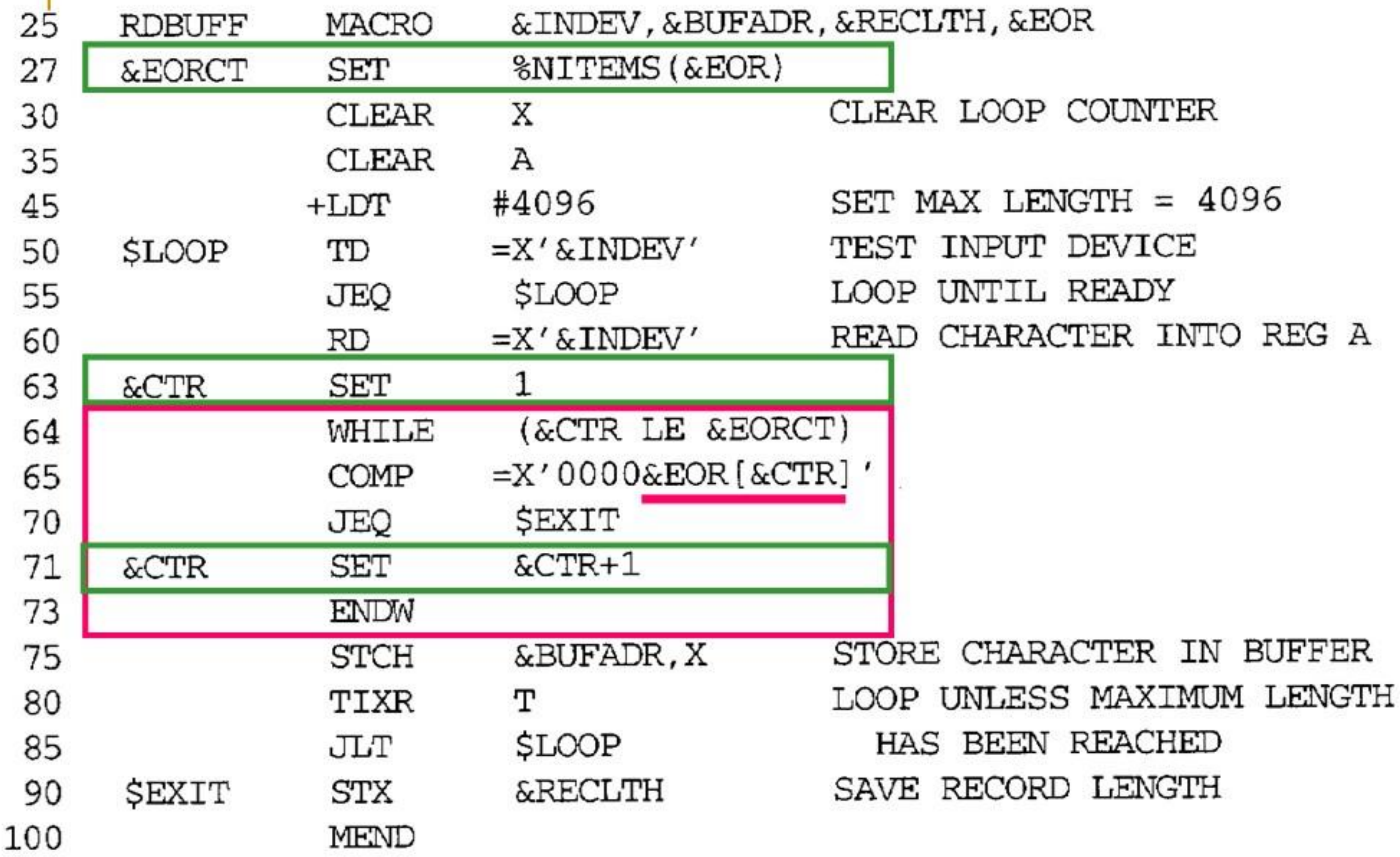


```
.               RDBUFF    F1,BUFF,RLENG,04

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        $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
```

(d)

4.2.3 Conditional Macro Expansion

- 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
&EORCT   SET     %NITEMS(&EOR)
         CLEAR   X                    CLEAR LOOP COUNTER
         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 &EORCT)
         COMP   =X'0000&EOR[&CTR]'
         JEQ     $EXIT
&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
```

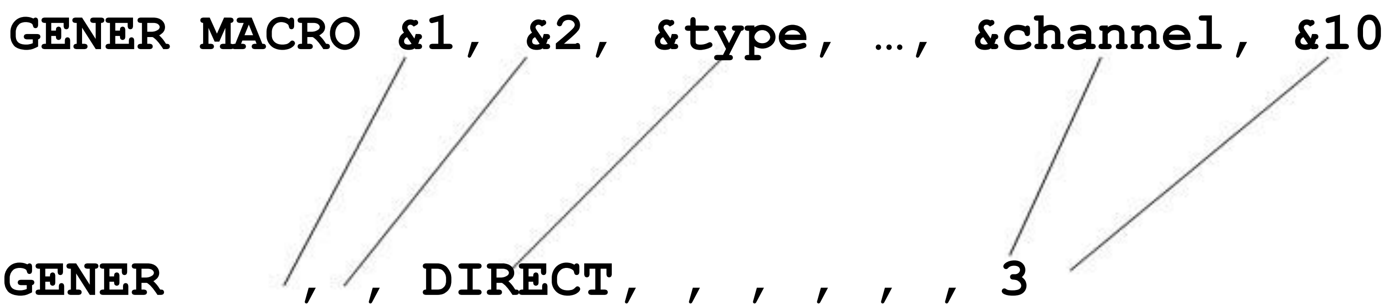
(a)

```
.          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'
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)

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
```

parameter=argument

  - Fig. 4.10 shows a version of the RDBUFF using keyword.

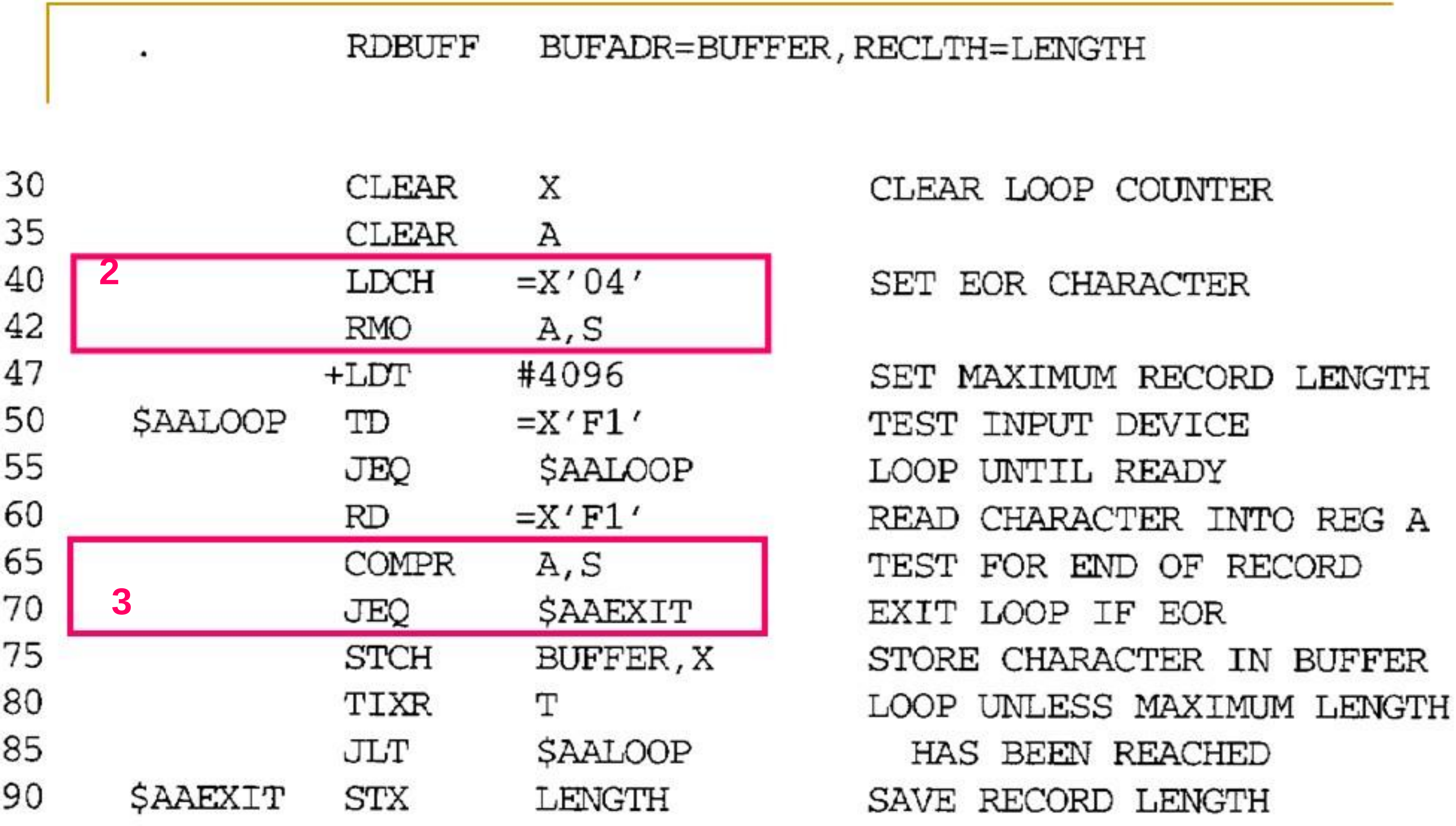
```
RDBUFF   MACRO    &INDEV=F1,&BUFADR=,&RECLTH=,&EOR=04,&MAXLTH=4096
         IF       (&EOR NE '')
&EORCK   SET      1
         ENDIF
         CLEAR    X              CLEAR LOOP COUNTER
         CLEAR    A
```

2

```
         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
```

3

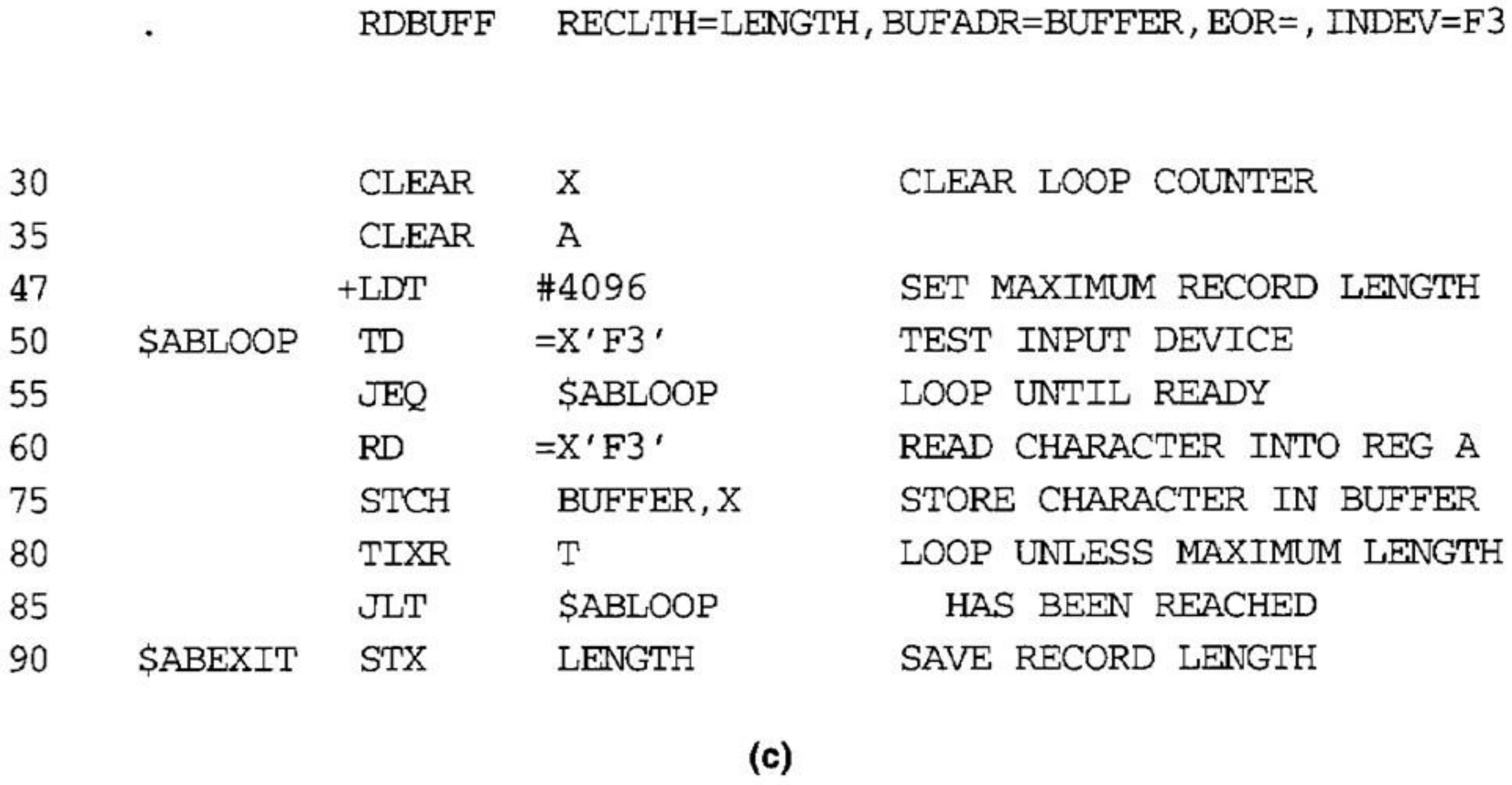
```
         IF       (&EORCK EQ 1)
         COMPR    A,S            TEST FOR END OF RECORD
         JEQ      $EXIT          EXIT LOOP IF EOR
         ENDIF
         STCH     &BUFADR,X      STORE CHARACTER IN BUFFER
         TIXR     T              LOOP UNLESS MAXIMUM LENGTH
         JLT      $LOOP            HAS BEEN REACHED
$EXIT    STX      &RECLTH        SAVE RECORD LENGTH
         MEND
```



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

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
```

(b)

Figure 4.10 Use of keyword parameters in macro instructions.



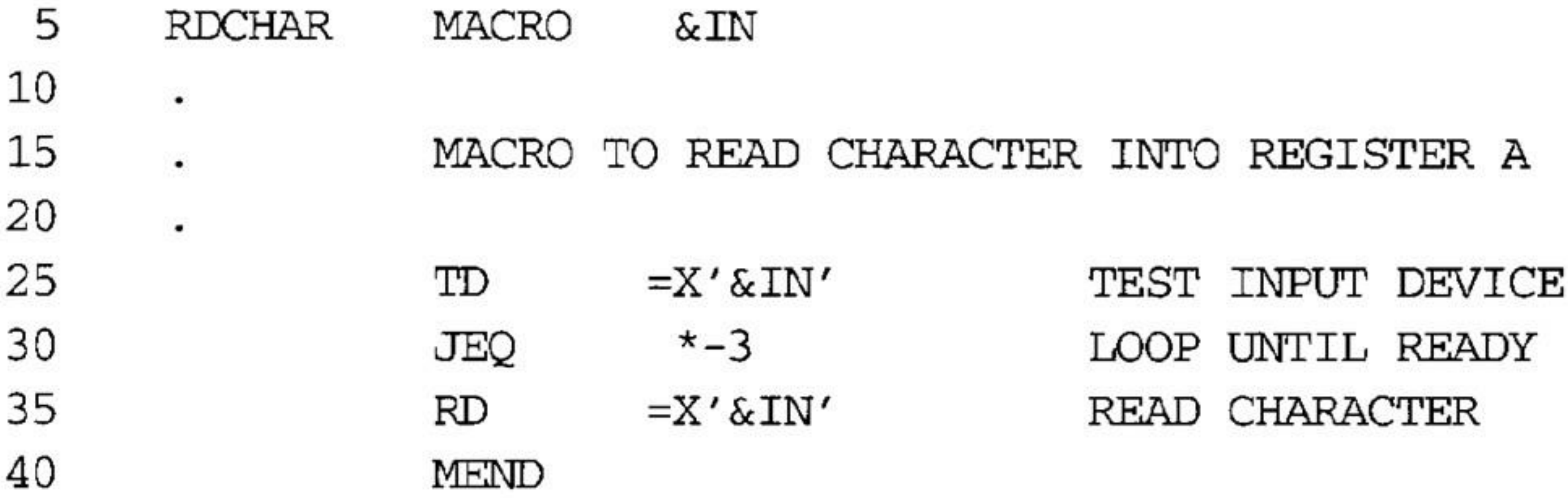
```
   .          RDBUFF   RECLTH=LENGTH,BUFADR=BUFFER,EOR=,INDEV=F3


30            CLEAR     X               CLEAR LOOP COUNTER
35            CLEAR     A
47           +LDT      #4096            SET MAXIMUM RECORD LENGTH
50   $ABLOOP  TD       =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
```

(c)

Figure 4.10 *(cont'd)*

4.3 Macro Processor Design Options
4.3.1 Recursive Macro Expansion

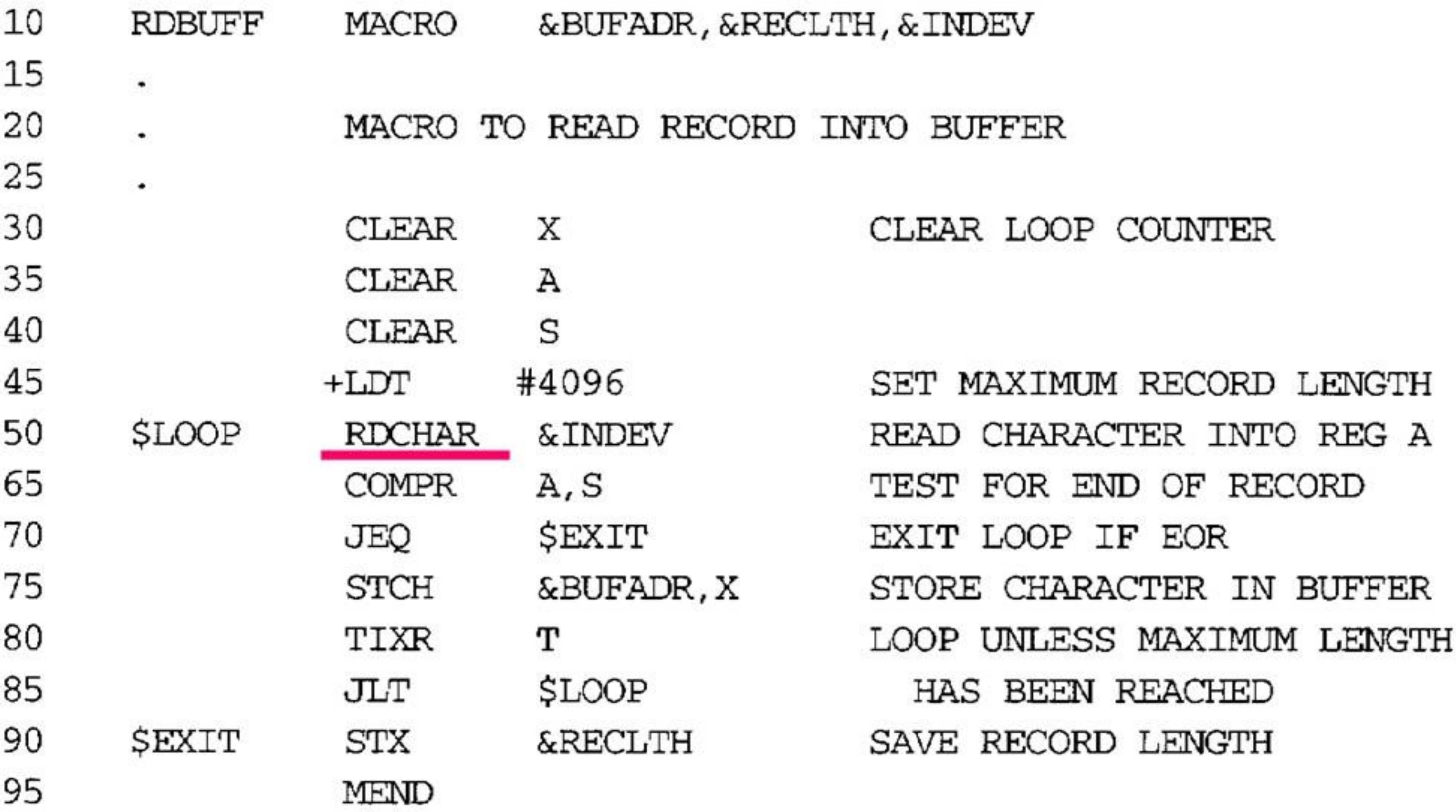
- In Fig. 4.3 we presented an example of the definition of one macro instruction by another.
- Fig. 4.11(a) shows an example - Dealt with the invocation of one macro by another.
- The purpose of RDCHAR Fig. 4.11(b) is to read one character from a specified device into register A, taking care of the necessary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



```
RDCHAR   MACRO    &IN
.
.        MACRO TO READ CHARACTER INTO REGISTER A
.
         TD       =X'&IN'          TEST INPUT DEVICE
         JEQ       *-3             LOOP UNTIL READY
         RD       =X'&IN'          READ CHARACTER
         MEND
```

(b)

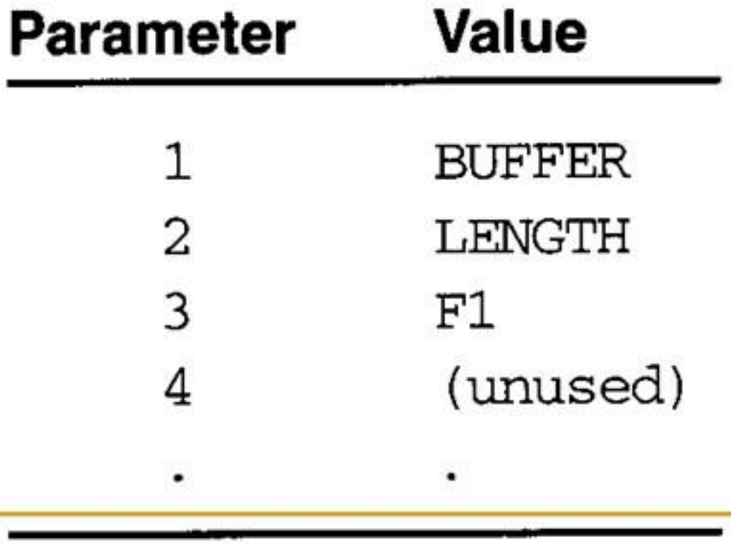
```
         RDBUFF   BUFFER,LENGTH,F1
```



```
RDBUFF   MACRO    &BUFADR,&RECLTH,&INDEV
.
.        MACRO TO READ RECORD INTO BUFFER
.
         CLEAR    X                      CLEAR LOOP COUNTER
         CLEAR    A
         CLEAR    S
        +LDT      #4096                  SET MAXIMUM RECORD LENGTH
$LOOP    RDCHAR   &INDEV                 READ CHARACTER INTO REG A
         COMPR    A,S                    TEST FOR END OF RECORD
         JEQ      $EXIT                  EXIT LOOP IF EOR
         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
```

4.3.1 Recursive Macro Expansion

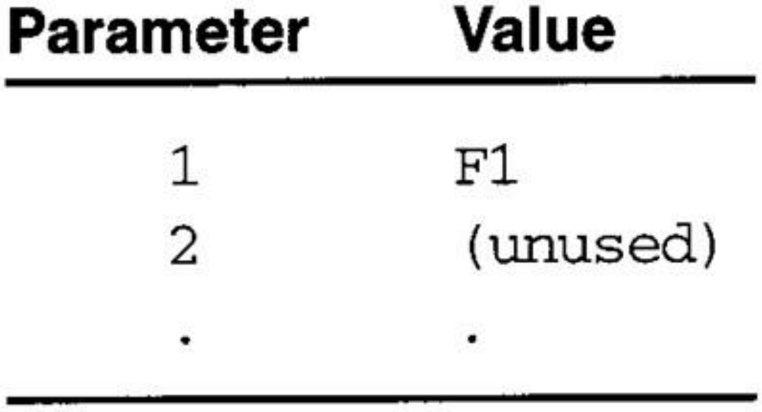
- Fig. 4.11(c), applied to the macro invocation statement
  `RDBUFF  BUFFER, LENGTH, F1`
- The procedure EXPAND would be called when the macro was recognized.
- The arguments from the macro invocation would be entered into ARGTAB as follows:



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

4.3.1 Recursive Macro Expansion

- The Boolean variable EXPANDING would be set to TRUE, and expansion of the macro invocation statement would be 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, ARGTAB would look like

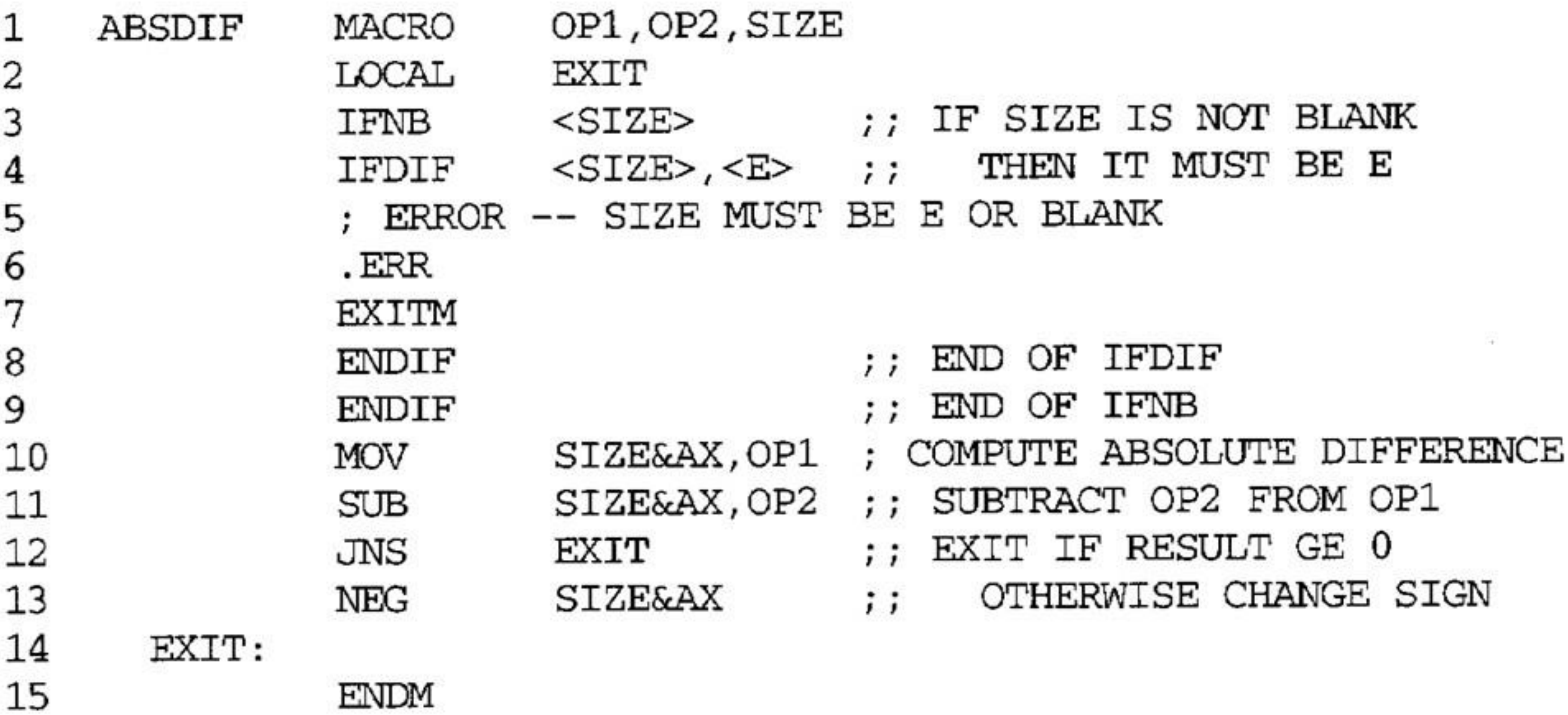


Parameter	Value
1	F1
2	(unused)
.	.

4.3.1 Recursive Macro Expansion

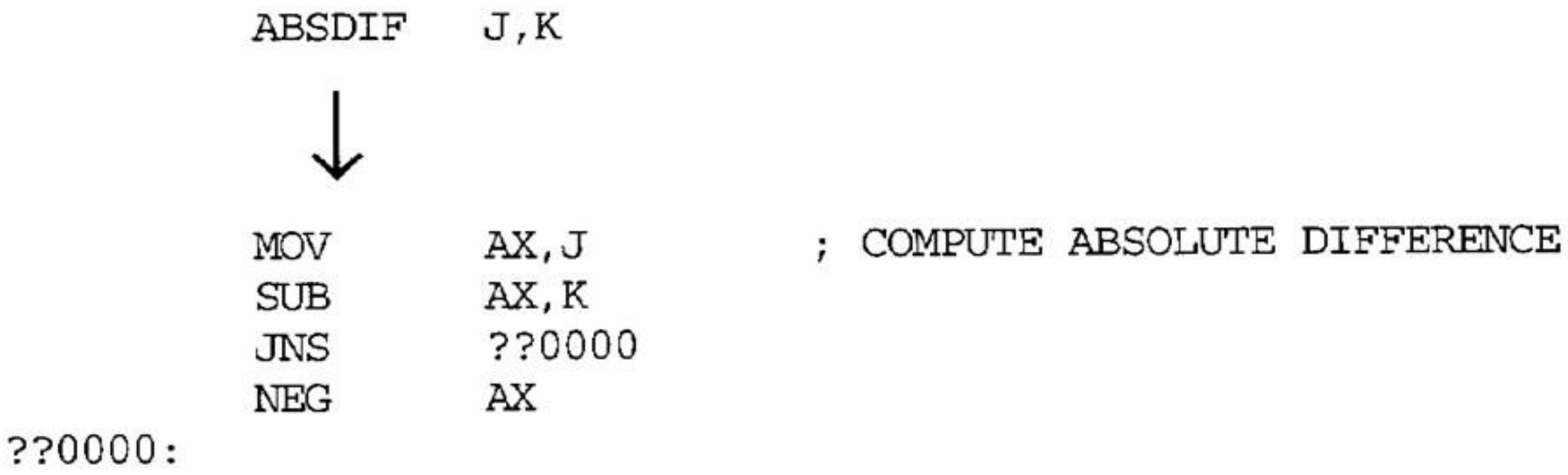
- At the end of this expansion, however, a problem would appear. When the end of the definition of RDCHAR was recognized, EXPANDING would be set to FALSE.
- Thus the macro processor would “forget” that it had been in middle of expanding a macro when it encountered the RDCHAR statement.
- Use a Stack to save ARGTAB.
- Use a counter to identify the expansion.

Pages 214-216, MASM



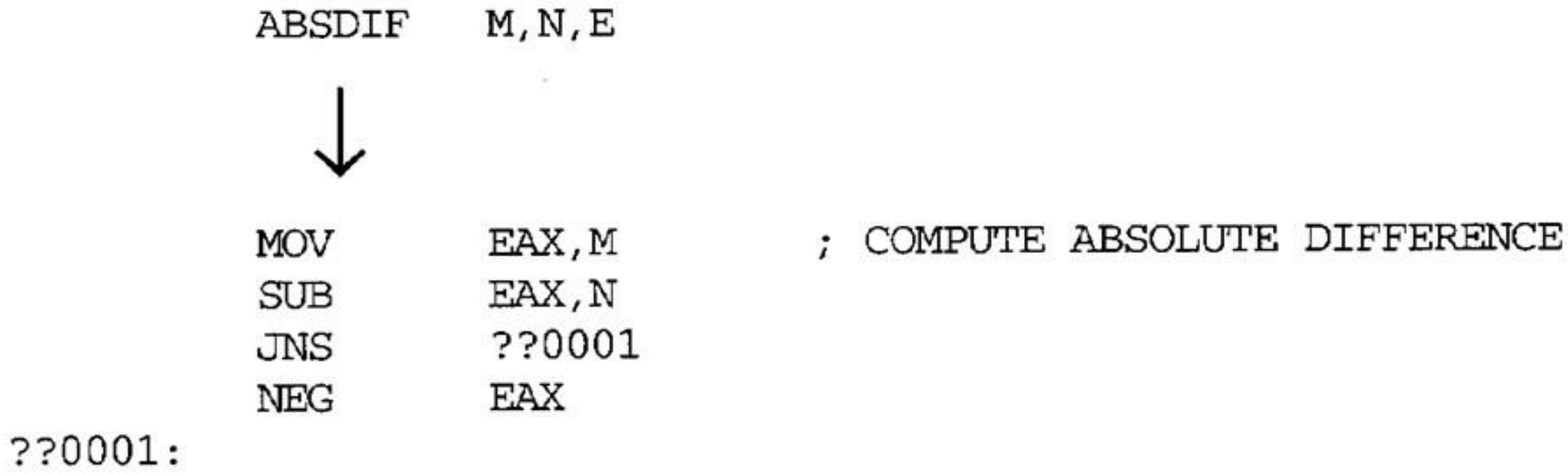
```
ABSDIF   MACRO    OP1,OP2,SIZE
         LOCAL    EXIT
         IFNB     <SIZE>          ;; IF SIZE IS NOT BLANK
         IFDIF    <SIZE>,<E>      ;;   THEN IT MUST BE E
         ; ERROR -- SIZE MUST BE E OR BLANK
         .ERR
         EXITM
         ENDIF                    ;; END OF IFDIF
         ENDIF                    ;; END OF IFNB
         MOV      SIZE&AX,OP1    ; COMPUTE ABSOLUTE DIFFERENCE
         SUB      SIZE&AX,OP2    ;; SUBTRACT OP2 FROM OP1
         JNS      EXIT            ;; EXIT IF RESULT GE 0
         NEG      SIZE&AX         ;;   OTHERWISE CHANGE SIGN
  EXIT:
         ENDM
```

(a)



```
        ABSDIF   J,K
          ↓
        MOV      AX,J        ; COMPUTE ABSOLUTE DIFFERENCE
        SUB      AX,K
        JNS      ??0000
        NEG      AX
??0000:
```

(b)



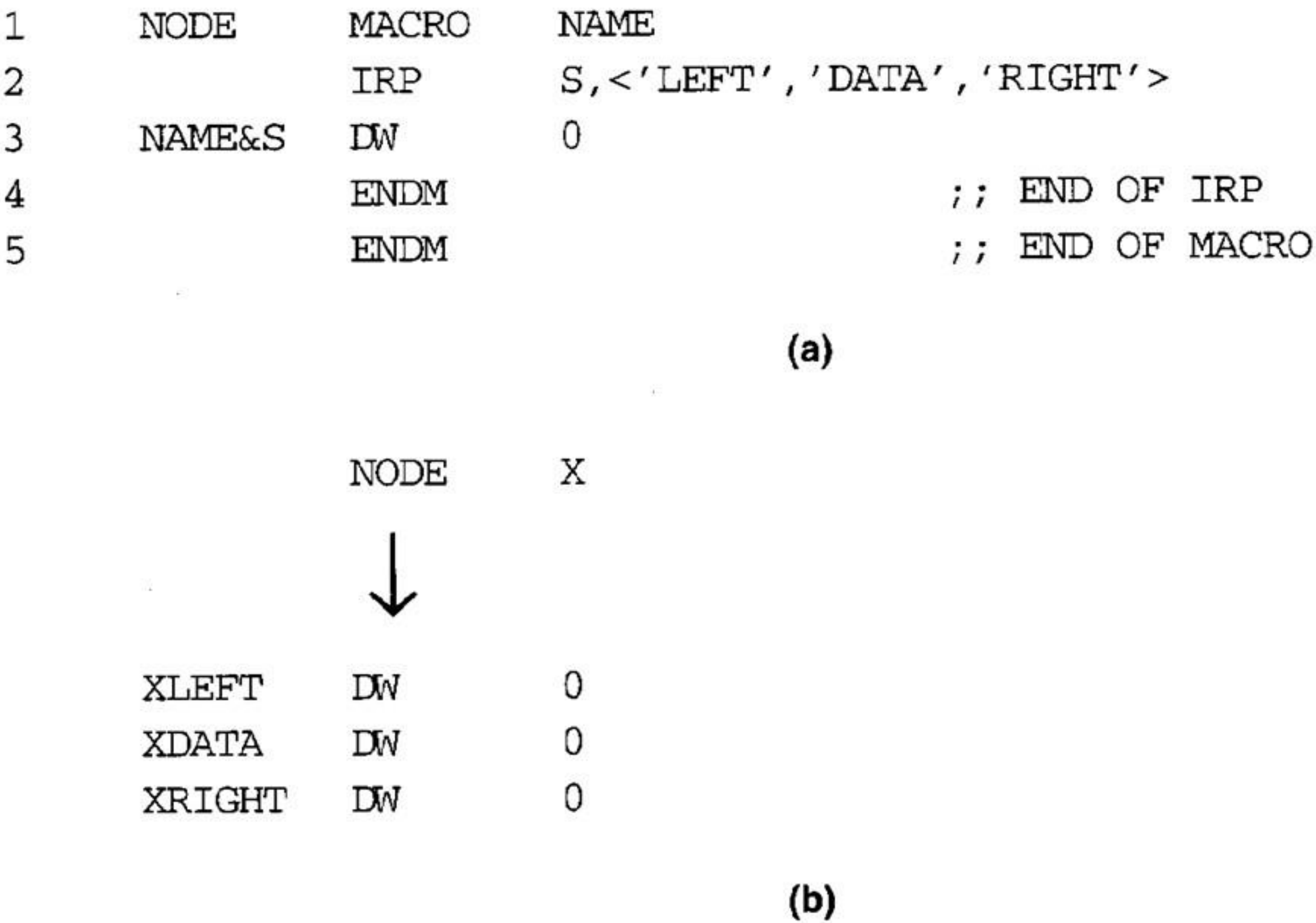
```
        ABSDIF   M,N,E
          ↓
        MOV      EAX,M       ; COMPUTE ABSOLUTE DIFFERENCE
        SUB      EAX,N
        JNS      ??0001
        NEG      EAX
??0001:
```

(c)

```
ABSDIF   P,Q,X

   ↓

; ERROR -- SIZE MUST BE E OR BLANK
```

(d)

Figure 4.12 Examples of MASM macro and conditional statements.



```
NODE      MACRO     NAME
          IRP       S,<'LEFT','DATA','RIGHT'>
NAME&S    DW        0
          ENDM                          ;; END OF IRP
          ENDM                          ;; END OF MACRO
```

(a)

```
          NODE      X
           ↓
XLEFT     DW        0
XDATA     DW        0
XRIGHT    DW        0
```

(b)

Figure 4.13 Example of MASM iteration statement.