Chapter 4
Macro Processors

Chapter 4: Macro Processors

- 4.1 Basic Macro Processors Functions
- 4.2 Machine-Independent Macro Processors Features
- 4.3 Macro Processors Design Options
- 4.4 Implementation Examples

Introduction to Macro Processors

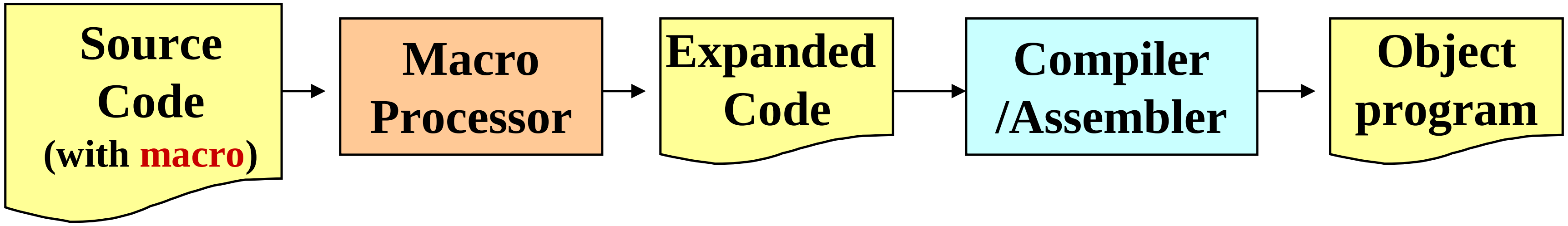
- A *macro instruction* (*macro*) is a notational convenience for the programmer.
  - Allow the programmer to write a shorthand version of a program
- A *macro* represents a commonly used group of statements in the source programming language.
- *Expanding* the macros
  - The macro processor replaces each macro instruction with the corresponding group of source language statements.

Introduction to Macro Processors (Cont.)

- A macro processor
  - Essentially involve the substitution of one group of characters or lines for another.
  - Normally, it performs no analysis of the text it handles.
  - It doesn't concern the meaning of the involved statements during macro expansion
- The design of a macro processor generally is machine independent.

Introduction to Macro Processors (Cont.)

- Three examples of actual macro processors:
  - A macro processor designed for use by assembler language programmers
  - Used with a high-level programming language
  - General-purpose macro processor, which is not tied to any particular language



Introduction to Macro Processors (Cont.)

- C uses a macro preprocessor to support language extensions, such as named constants, expressions, and file inclusion.

```
#define max(a,b) ((a<b)?(a):(b))
#define MACBUF    4
#include <stdio.h>
```

4.1 Basic Macro Processors Functions

- *Macro processor* should processes the
  - **Macro definitions**
    - Define macro name, group of instructions
  - **Macro invocation (macro calls)**
    - A body is simply copied or substituted at the point of call
  - Expansion with substitution of parameters
    - ***Arguments*** are textually substituted for the ***parameters***
    - The resulting procedure body is textually substituted for the call

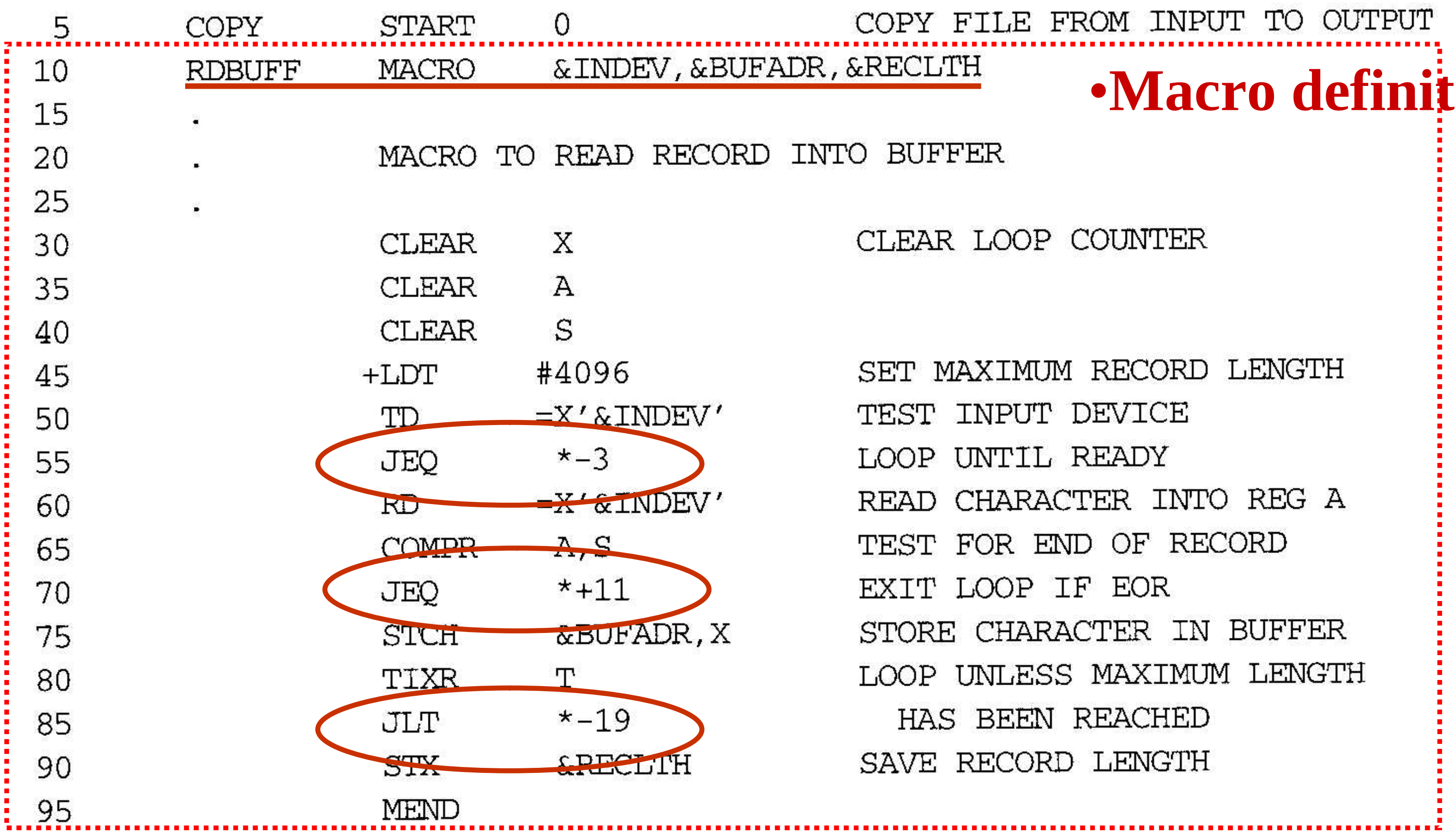
Macro Definition

- Two new assembler directives are used in macro definition:
  - **MACRO**: identify the beginning of a macro definition
  - **MEND**: identify the end of a macro definition
- label op operands

```
name   MACRO   parameters
         :
        body
         :
        MEND
```

- Parameters: the entries in the operand field identify the parameters of the macro instruction
  - We require each parameter begins with ‘&’
- Body: the statements that will be generated as the expansion of the macro.
- Prototype for the macro:
  - The *macro name* and *parameters* define a pattern or *prototype* for the macro instructions used by the programmer

Fig 4.1: Macro Definition



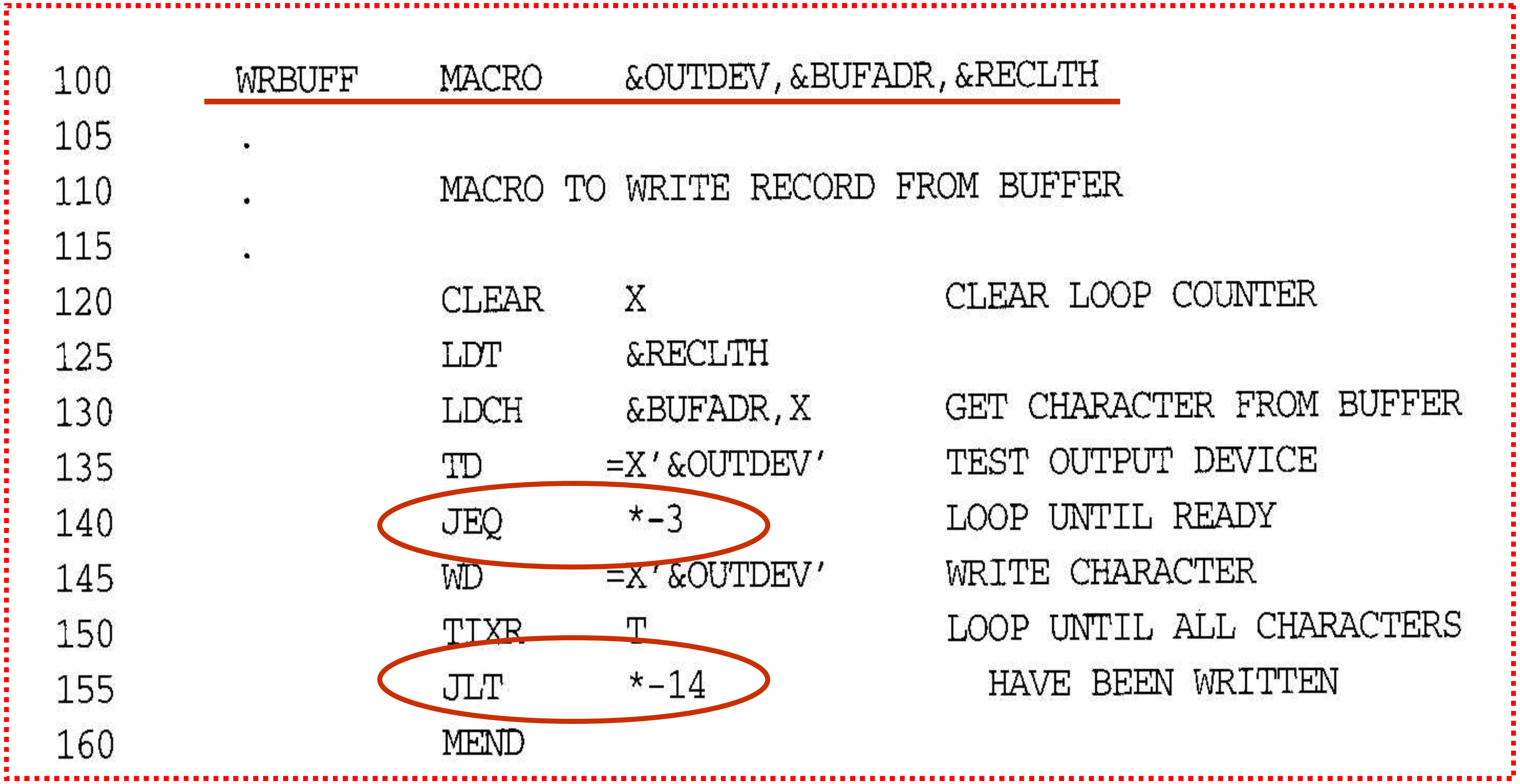
```
COPY      START   0                  COPY FILE FROM INPUT TO OUTPUT
RDBUFF    MACRO   &INDEV,&BUFADR,&RECLTH
.
.         MACRO TO READ RECORD INTO BUFFER
.
          CLEAR   X                  CLEAR LOOP COUNTER
          CLEAR   A
          CLEAR   S
         +LDT     #4096              SET MAXIMUM 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
```

- **Macro definition**

Macro body contains no label

Fig 4.1: Macro Definition (Cont.)

•**Macro definition**



```
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
```

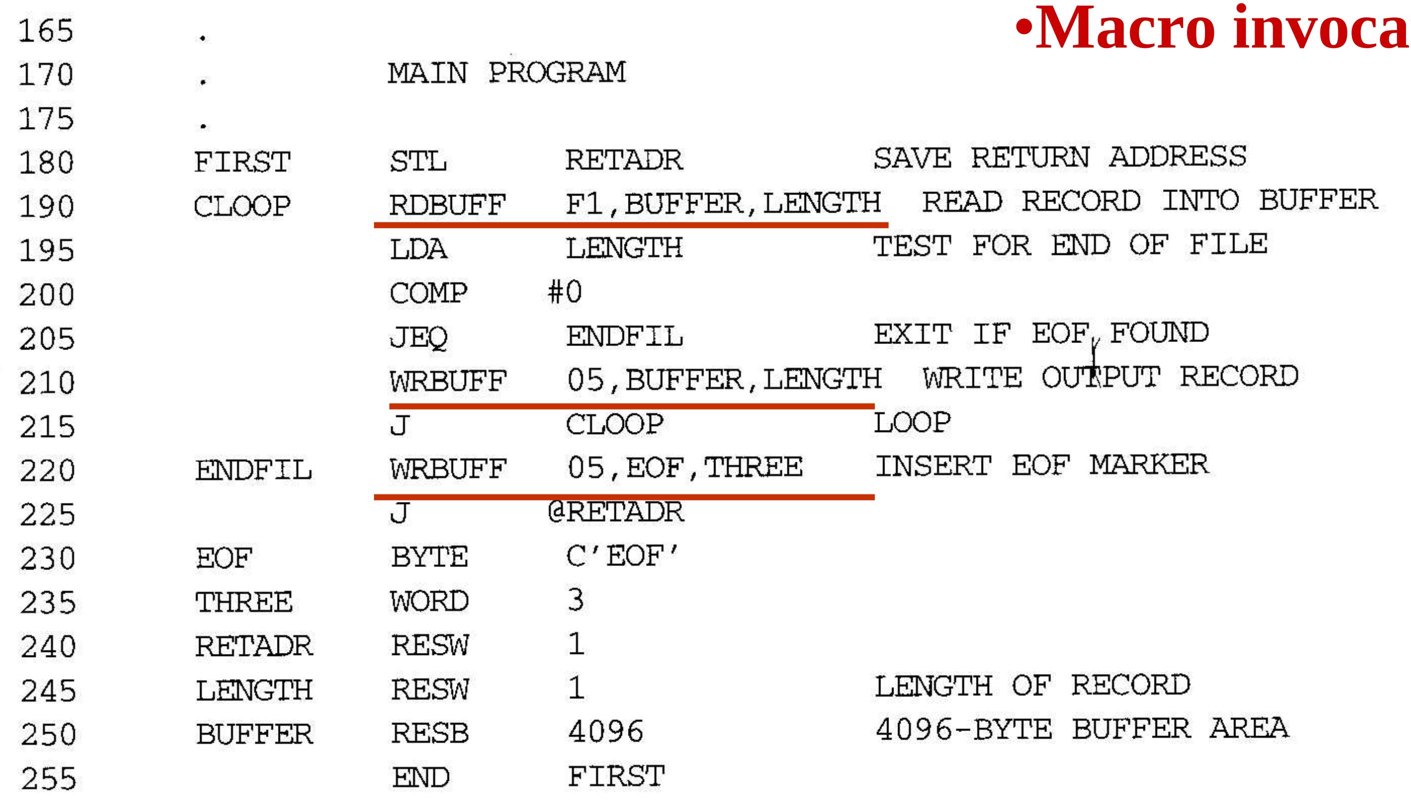
Macro body contains no label

Macro Invocation

- A ***macro invocation statement*** (a ***macro call***) gives the **name** of the macro instruction being invoked and the **arguments** in expanding the macro.
- Macro Invocation vs. Subroutine Call.
  - Statements of the macro body are expanded each time the macro is invoked.
  - Statements of the subroutine appear only one, regardless of how many times the subroutine is called.
  - Macro invocation is more efficient than subroutine call, however, the code size is larger

Fig 4.1: Macro Invocation

•**Macro invocation**



```
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
```

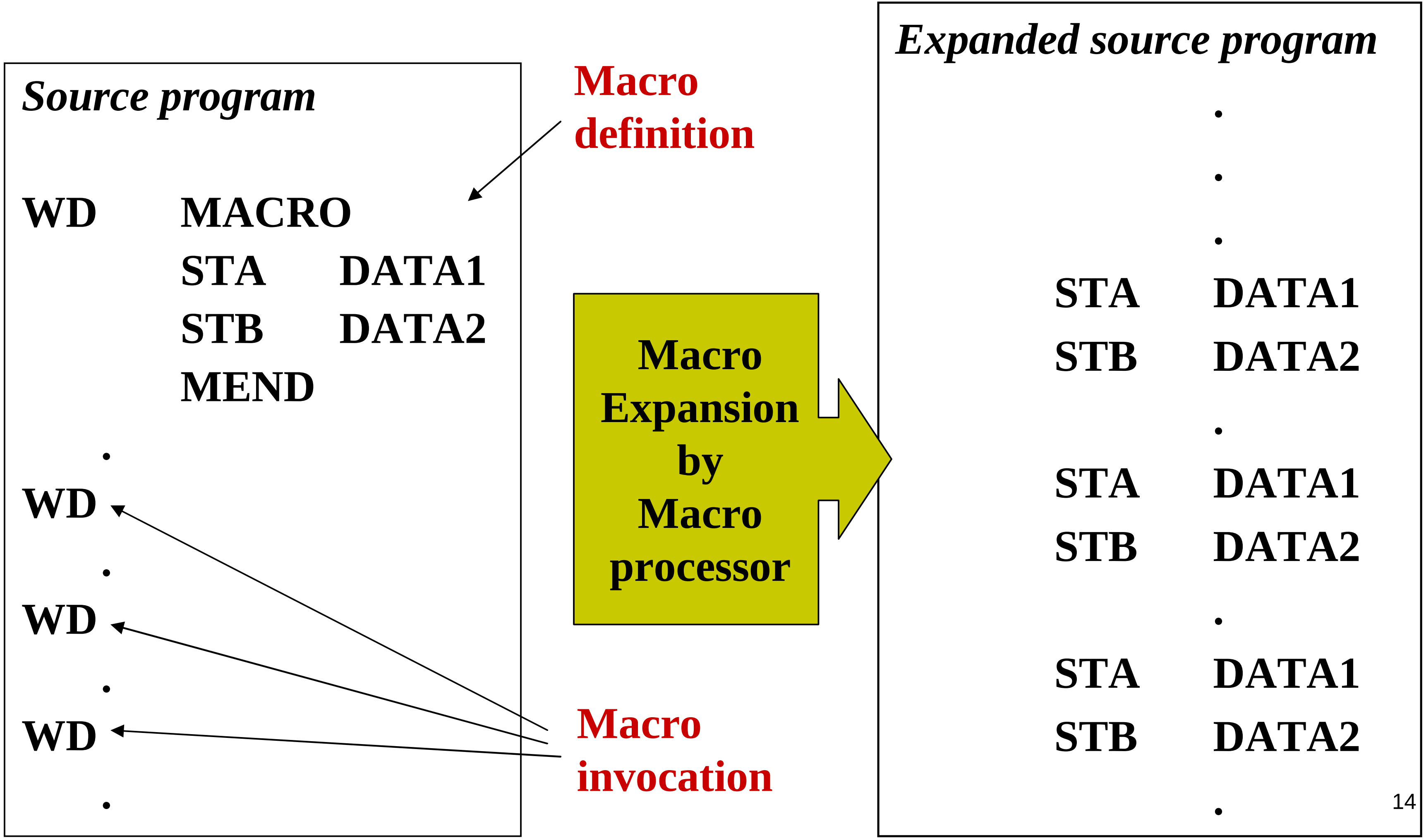
Figure 4.1 Use of macros in a SIC/XE program.

Macro Expansion

- Each macro invocation statement will be expanded into the statements that form the **body** of the macro.

- Arguments from the macro invocation are 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, etc.

Macro Expansion



Source program

```
WD      MACRO
        STA     DATA1
        STB     DATA2
        MEND
     .
WD
     .
WD
     .
WD
     .
```

Macro definition

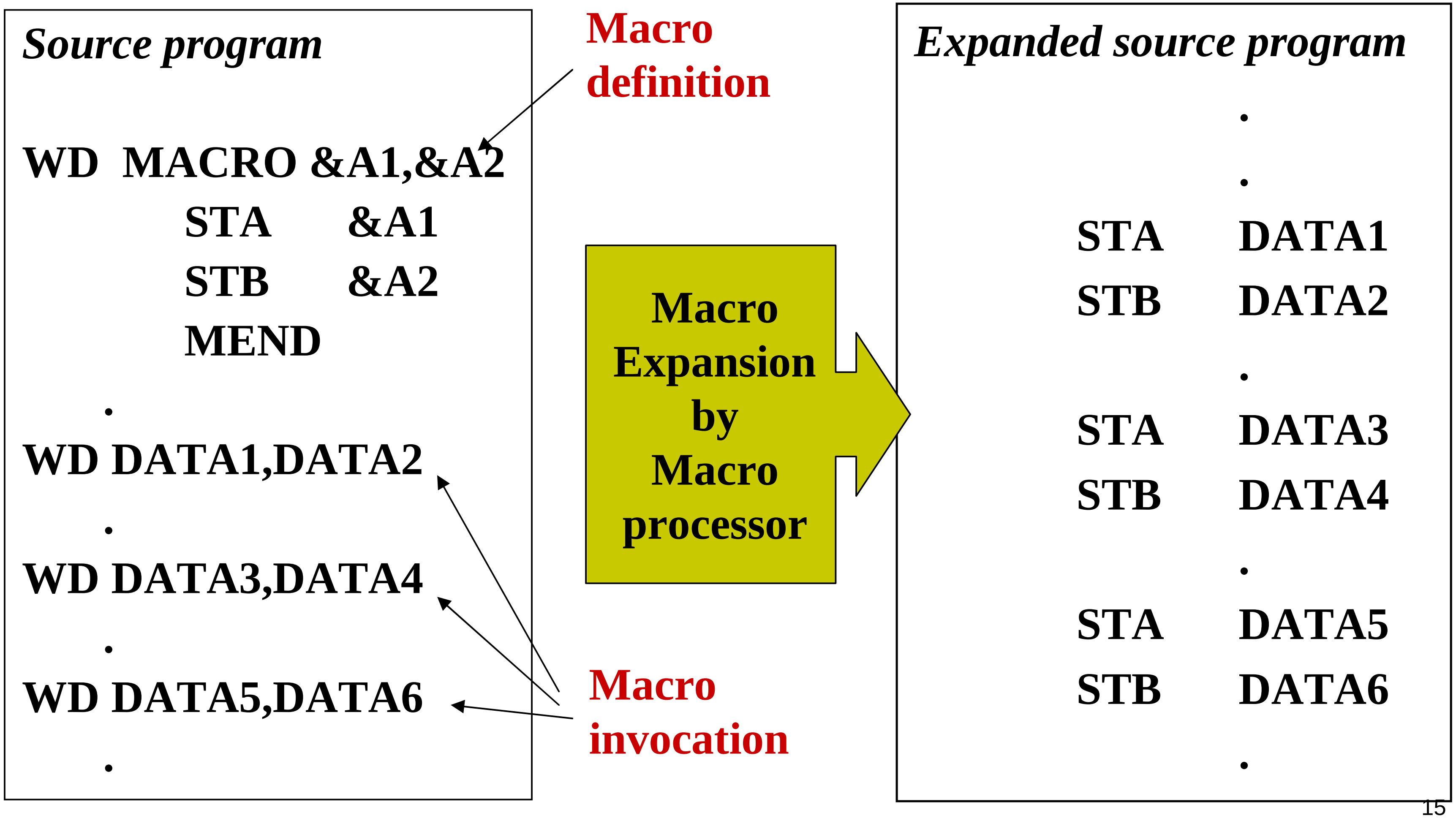
Macro Expansion by Macro processor

Macro invocation

Expanded source program

```
          .
          .
          .
STA       DATA1
STB       DATA2
          .
STA       DATA1
STB       DATA2
          .
STA       DATA1
STB       DATA2
          .
```

Macro Expansion with Parameters Substitution



Source program

```
WD  MACRO &A1,&A2
        STA     &A1
        STB     &A2
        MEND
     .
WD DATA1,DATA2
     .
WD DATA3,DATA4
     .
WD DATA5,DATA6
     .
```

Macro definition

Macro Expansion by Macro processor

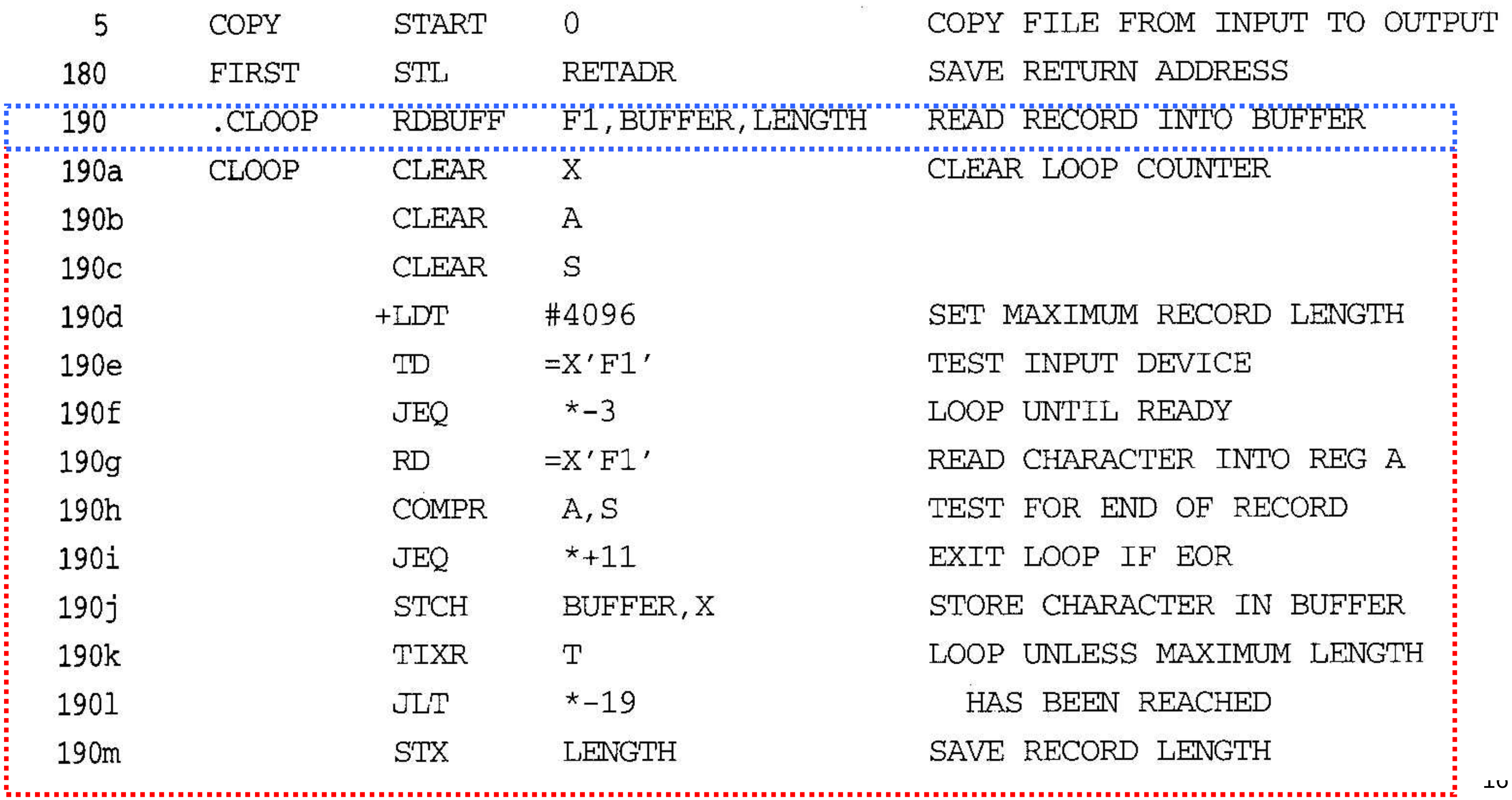
Macro invocation

Expanded source program

```
          .
          .
STA     DATA1
STB     DATA2
          .
STA     DATA3
STB     DATA4
          .
STA     DATA5
STB     DATA6
          .
```

Program From Fig. 4.1 with Macros Expanded (fig. 4.2)

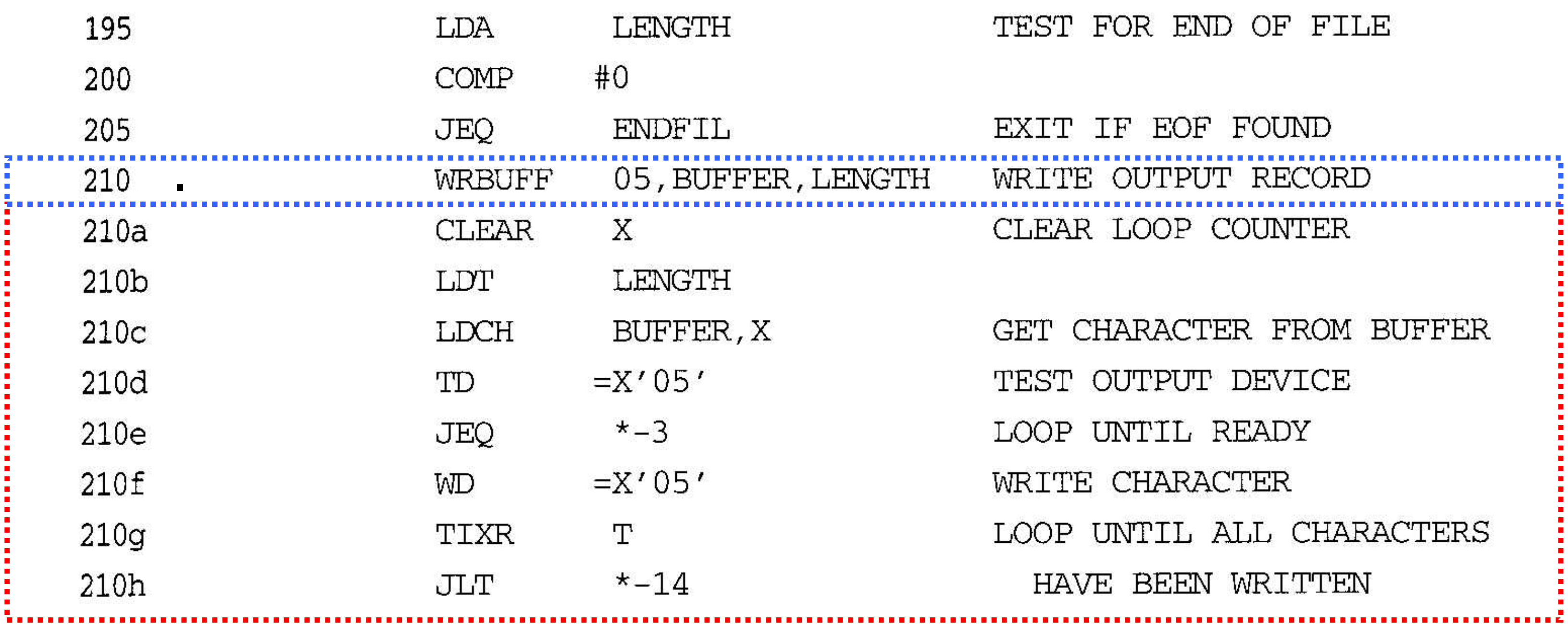
•Macro expansion



```
  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
```

Program From Fig. 4.1 with Macros Expanded (fig. 4.2)(Cont.)

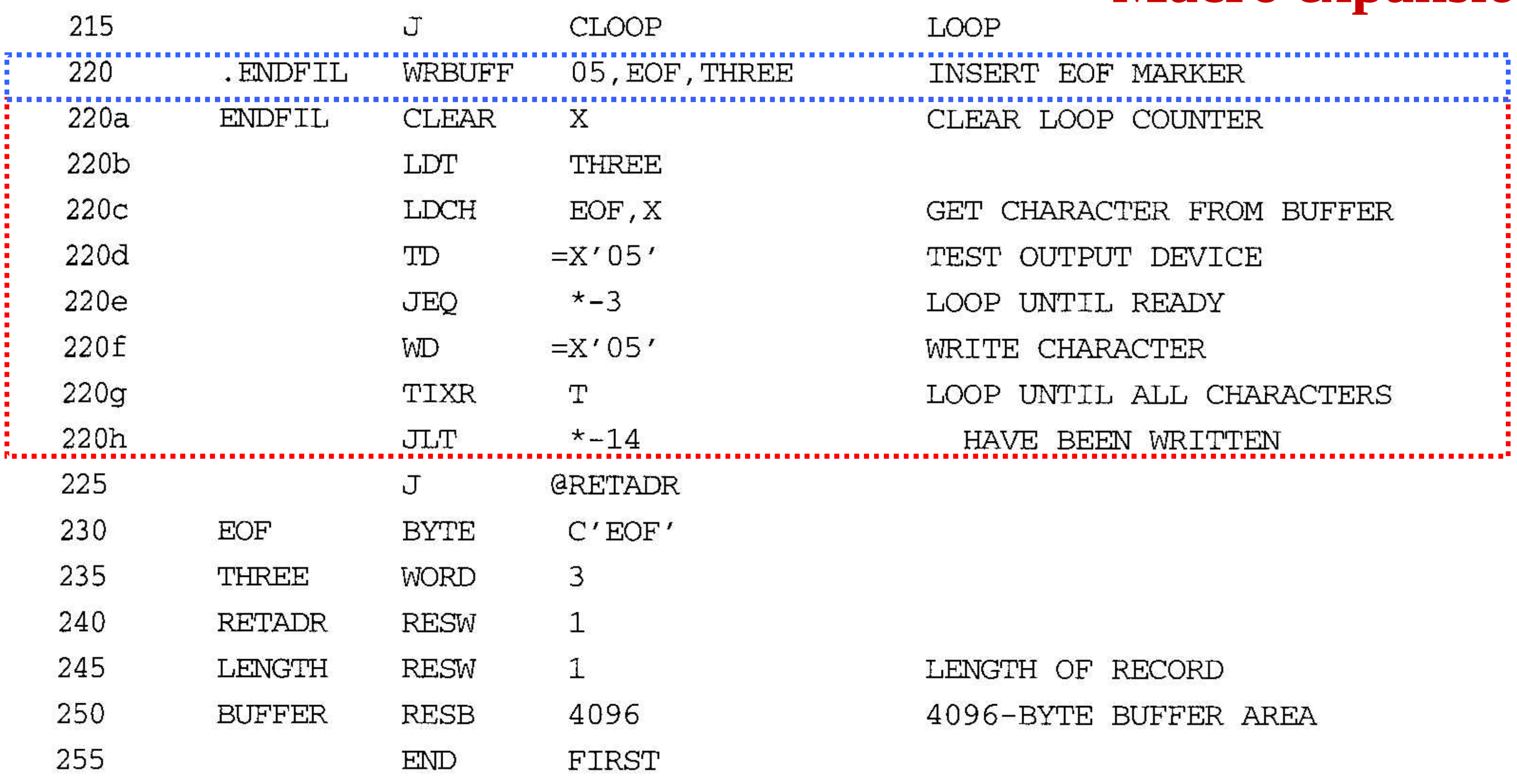
•Macro expansion



```
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
```

Program From Fig. 4.1 with Macros Expanded (fig. 4.2)(Cont.)

•Macro expansion



```
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.

No Label in the Body of Macro

- Problem of the label in the body of macro:
  - If the same macro is expanded multiple times at different places in the program.
    - There will be **duplicate** labels, which will be treated as errors by the assembler,
- Solutions:
  - Simply not to use labels in the body of macro.
  - Explicitly use PC-relative addressing instead.
    - For example, in RDBUFF and WRBUFF macros,

      ```
      JEQ          * +11
      JLT      *-14
      ```
    - It is inconvenient and error-prone.
  - Other better solution?
    - Mentioned in Section 4.2.2.

4.1.2 Macro Processors Algorithm and Data Structures

- Two-pass macro processor
- One-pass macro processor

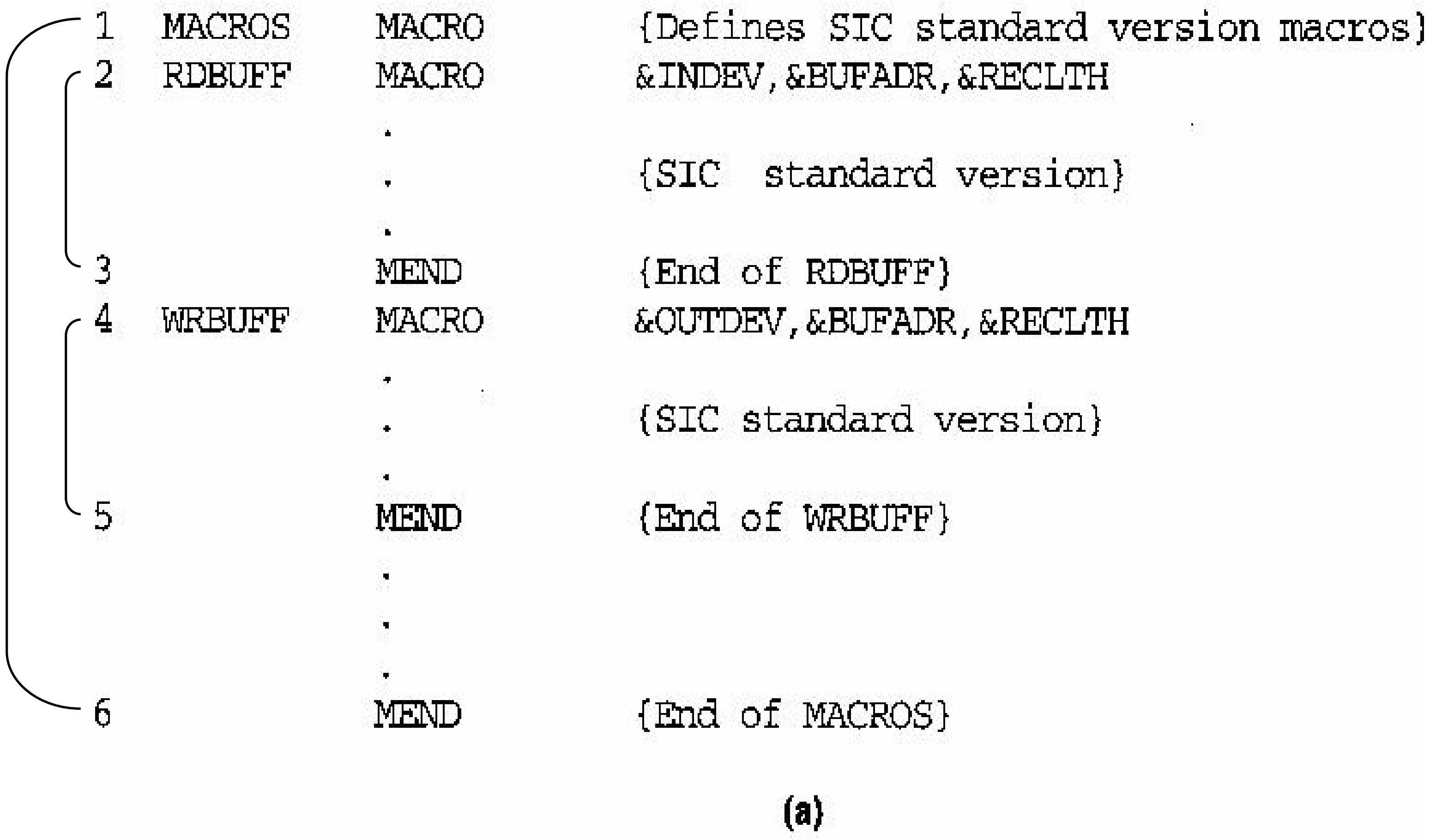
Two-pass macro processor

- Two-pass macro processor
  - Pass1: process all *macro definitions*
  - Pass2: expand all *macro invocation* statements
- Problem
  - Does not allow *nested macro definitions*
  - Nested macro definitions
    - The body of a macro contains definitions of other macros
  - Because all macros would *have to be defined during the first pass* before any macro invocations were expanded
- Solution
  - One-pass macro processor

Nested Macros Definition

- MACROS (for SIC)
  - contains the definitions of RDBUFF and WRBUFF written in SIC instructions.
- MACROX (for SIC/XE)
  - contains the definitions of RDBUFF and WRBUFF written in SIC/XE instructions.
- Example 4.3

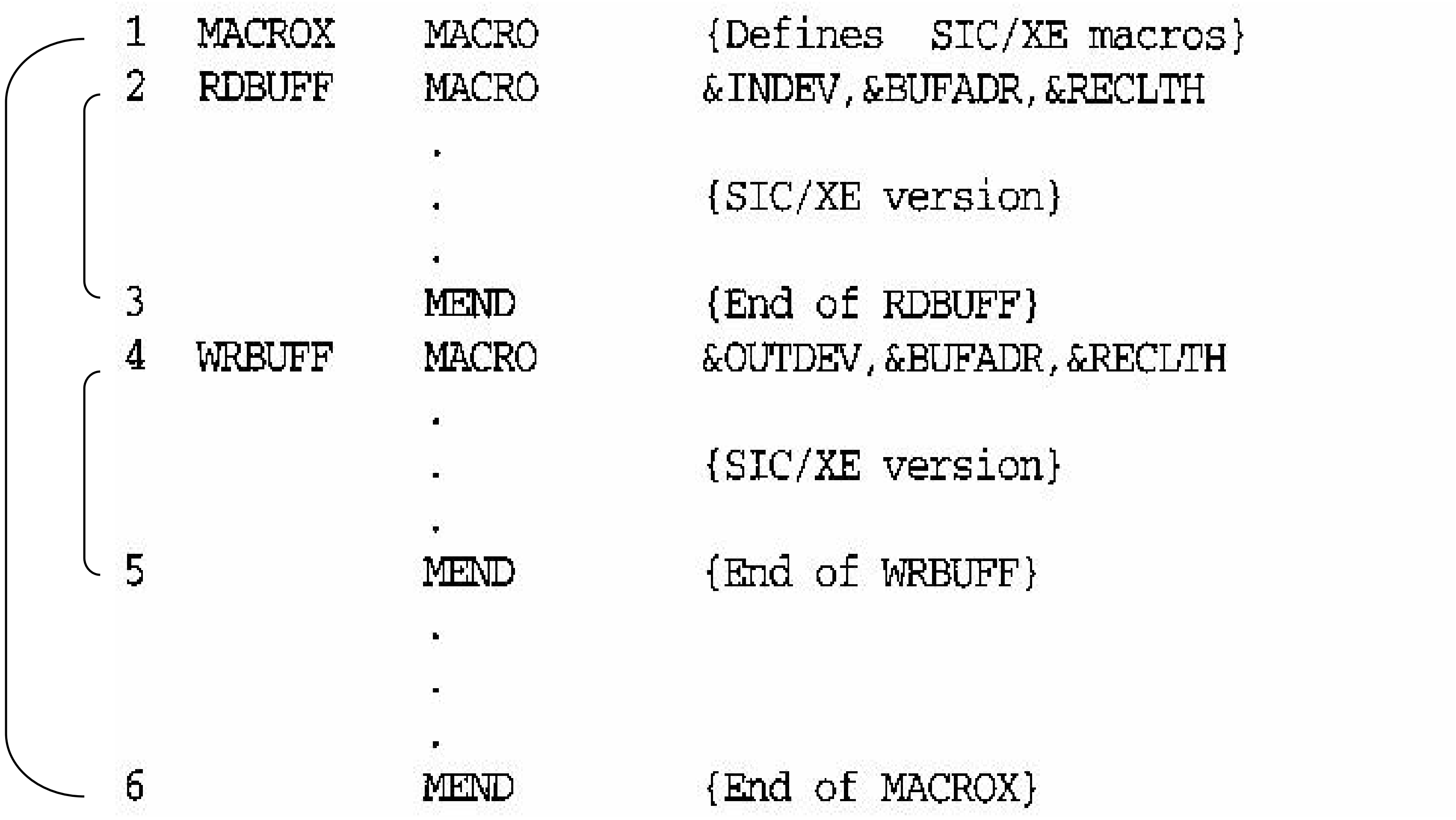
Macro Definition within a Macro Body (Figure 4.3(a))



```
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}
```

(a)

Macro Definition within a Macro Body (Figure 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}
```

Nested Macros Definition (Cont.)

- A program that is to be run on SIC system could invoke MACROS whereas a program to be run on SIC/XE can invoke MACROX.

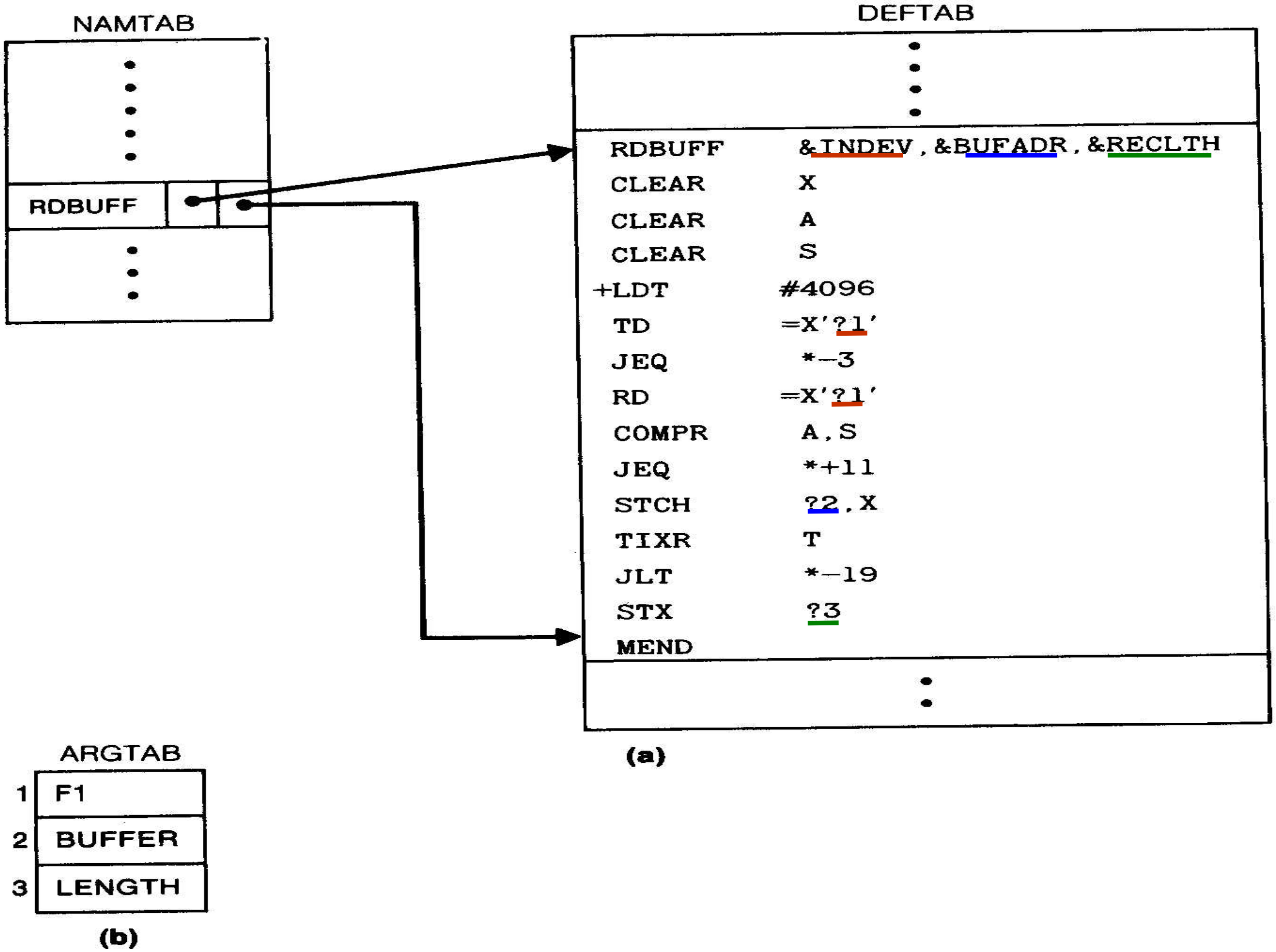
- Defining MACROX **does not** define RDBUFF and WRBUFF.
  - These definitions are processed only when an invocation of MACROX is expanded.

One-pass macro processor

- One-pass macro processor
  - *Every macro must be defined before it is called*
  - One-pass processor can alternate between ***macro definition*** and ***macro expansion***
  - Nested macro definitions are allowed

Three Main Data Structures

- **DEFTAB**
  - A ***definition table*** used to *store macro definition* including
    - macro prototype
    - macro body
  - Comment lines are omitted.
  - *Positional notation* has been used for the parameters for efficiency in substituting arguments.
    - E.g. the first parameter &INDEV has been converted to ?1 (indicating the first parameter in the prototype)
- **NAMTAB**
  - A ***name table*** used to *store the macro names*
  - Serves as an index to DEFTAB
    - Pointers to the beginning and the end of the macro definition
- **ARGTAB**
  - A ***argument table*** used to store the arguments used in the expansion of macro invocation
  - As the macro is expanded, arguments are substituted for the corresponding parameters in the macro body.



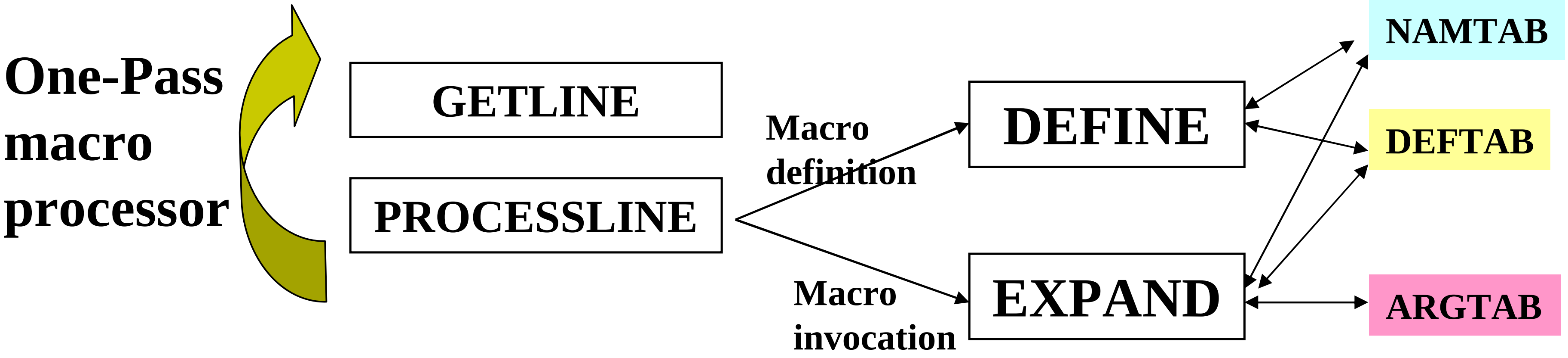
```
RDBUFF    &INDEV,&BUFADR,&RECLTH
CLEAR     X
CLEAR     A
CLEAR     S
+LDT      #4096
TD        =X'?1'
JEQ       *-3
RD        =X'?1'
COMPR     A,S
JEQ       *+11
STCH      ?2,X
TIXR      T
JLT       *-19
STX       ?3
MEND
```

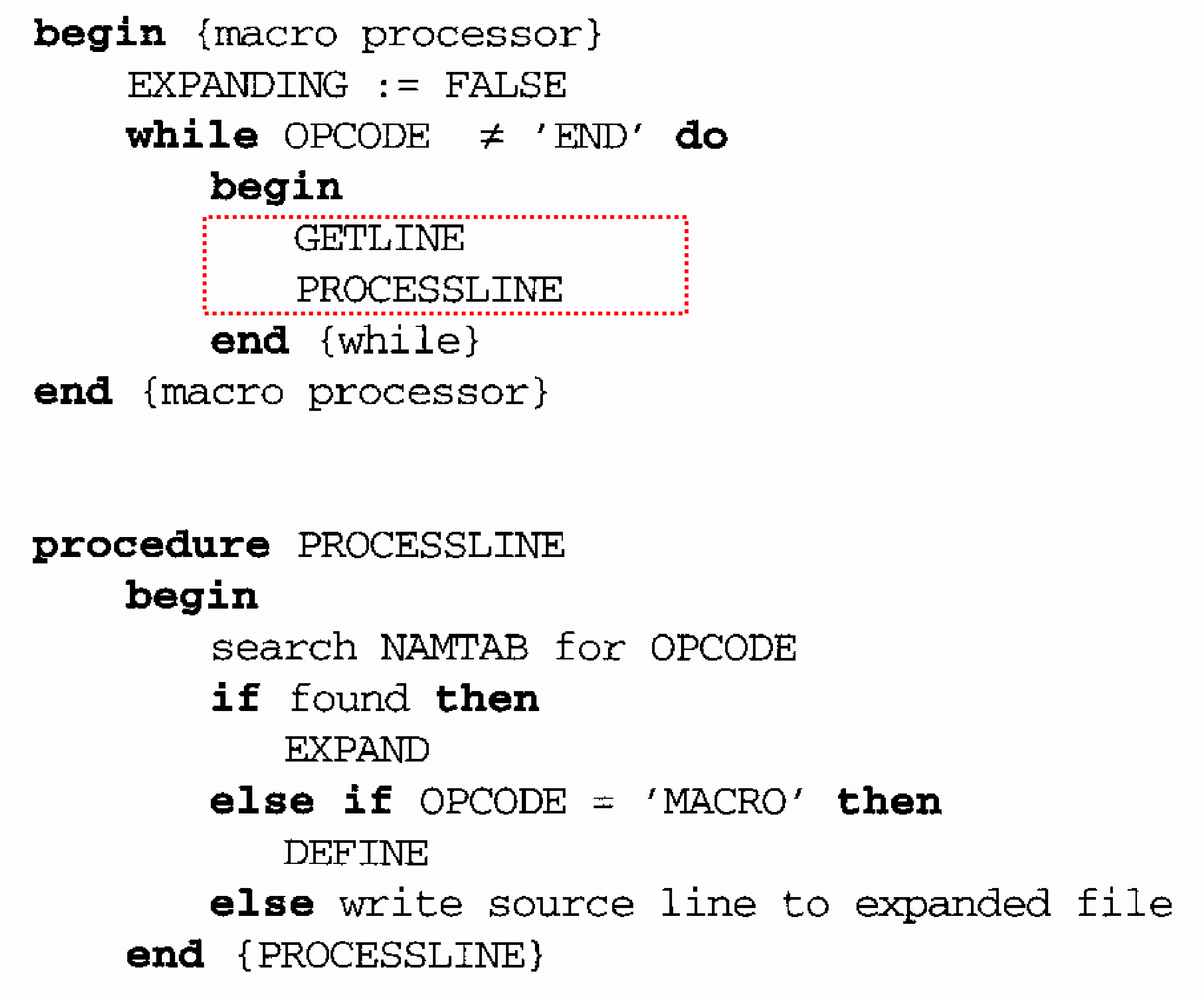
1	F1
2	BUFFER
3	LENGTH

Figure 4.4 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.

One-Pass Macro Processor

- Procedures
  - Macro definition: DEFINE
  - Macro invocation: EXPAND





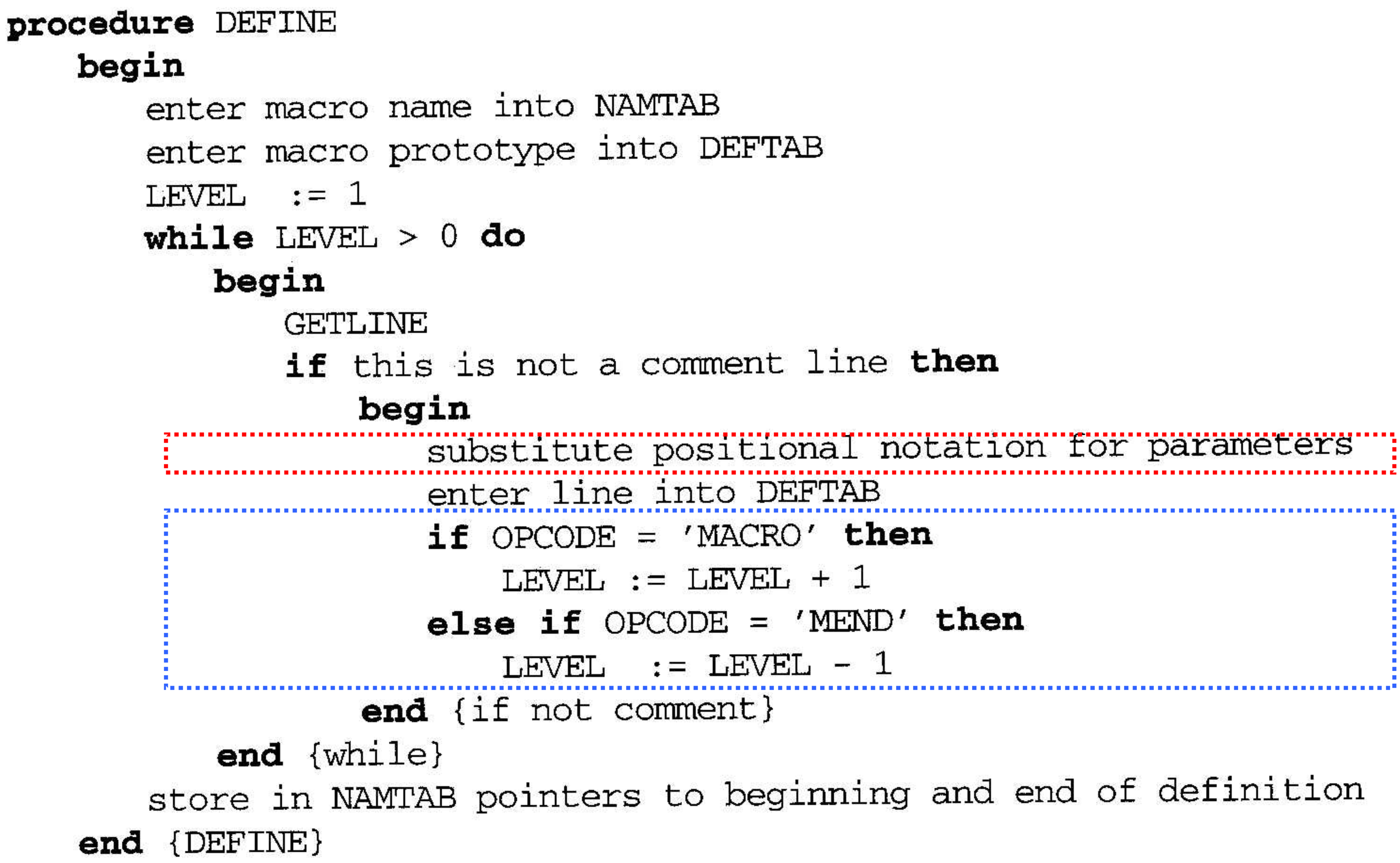
```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE ≠ 'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
end {macro processor}


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.

One-Pass Macro Processor Allows Nested Macro Definition

- Sub-procedure DEFINE should handle the nested macro definition
  - Maintains a counter named **LEVEL**
  - Each time a MACRO directive is read, the value of LEVEL is increased by 1
  - Each time an MEND directive is read, the value of LEVEL is decreased by 1

Algorithm for one-pass macro processor (Fig. 4.5)



```
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)*