Chapter 4: Macro Processor

Overview

Basic Functions

Features (Machine Independent)

Design Options

Overview[1]

Definition of Macro

- Webster defines the word macro (derived from the Greek *μαϰϱοσ*) as meaning long, great, excessive or large.
- The word is used as a prefix in many compound technical terms, e.g., Macroeconomics, and Macrograph.
- We will see that a single macro directive can result in many source lines being generated, which justifies the use of the word *macro* in assemblers.

Overview[2]

Macro Vs Subroutine

Macro

- Section of code that the programmer writes (defines) once, and then can use many times.
- Completely handled by the assembler/macro processor, at assembly/macro processing time.
- Duplicated as many times as necessary.

Subroutine

- Section of the program that is written once, and can be used many times by simply calling it from any point in the program.
- Completely handled by the hardware, at run time.
- Stored in memory once (just one copy)

Overview[3]

- In assembly language programming it is often that some set or block of statements get repeated every now.
- In this context the programmer uses the concept of *macro instructions* (often called as *macro*) where a single line abbreviation is used for a set of line.
- For every occurrence of that single line the whole block of statements gets expanded in the main source code.
- This gives a high level feature to assembly language that makes it more convenient for the user to write code easily.

Overview[4]

- A macro instruction (*macro*) is simply a notational convenience for the programmer. It allows the programmer to write shorthand version of a program (module programming).
- A macro represents a commonly used *group* of statements in the source program.
- The *macro processor* replaces each macro instruction with the corresponding group of source statements.
    - This operation is called "*expanding the macro*"
- Using macros allows a programmer to write a shorthand version of a program.
- *For example*, before calling a subroutine, the contents of all registers may need to be stored. This routine work can be done using a macro.

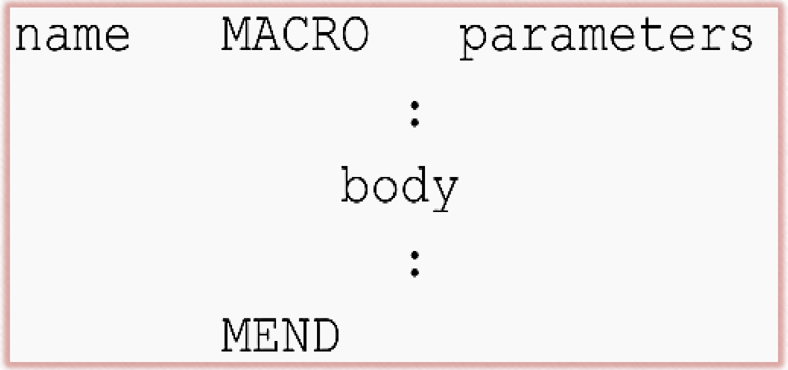
Overview[5]

- The functions of a macro processor essentially involve the substitution of one group of lines for another. Normally, the processor performs no analysis of the text it handles.
- The meaning of these statements are of no concern during macro expansion. Therefore, the design of a macro processor generally is *machine independent*.
- Macros *mostly* are used in assembler language programming. However, it can also be used in high-level programming languages such as C or C++.

Overview[6]

- The macro definition consists of the following parts:

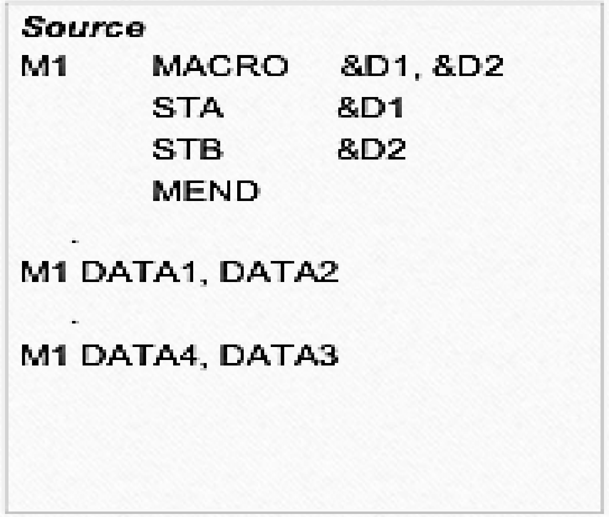
1.	Macro name	[ ]
2.	Start of definition	*MACRO*
3.	Sequence of statements	- - - - - - - - - - - - - - - -
4.	End of definition	*MEND*



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

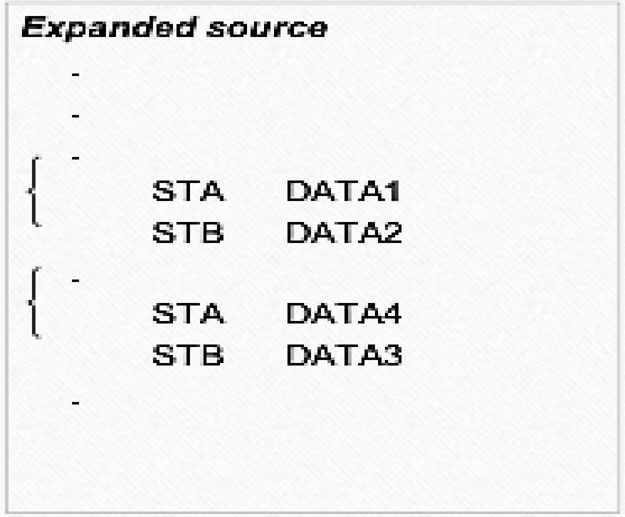
- *Once the macro is defined then the name of macro instruction now acts as a mnemonic in assembly language that is equivalent to sequence of the statements.*

Overview[7]



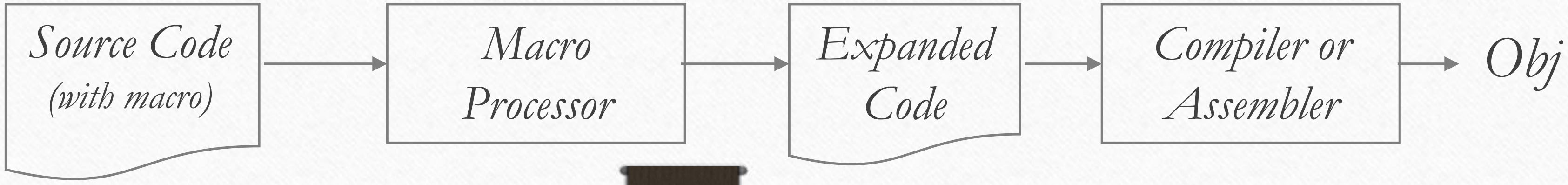
Source

```
M1      MACRO     &D1, &D2
        STA       &D1
        STB       &D2
        MEND
   .
M1 DATA1, DATA2
   .
M1 DATA4, DATA3
```



Expanded source

```
   .
   .
{  .
        STA     DATA1
        STB     DATA2
{  .
        STA     DATA4
        STB     DATA3
   .
```

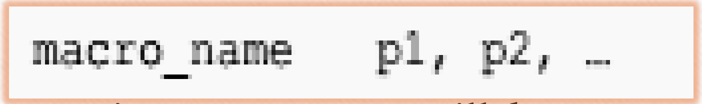


Basic Functions[1]

- Macro definition
  - The two directive *MACRO* and *MEND* are used in macro definition.
  - The macro’s name appears before the MACRO directive.
  - The macro’s parameters appear after the MACRO directive.
  - Each parameter begins with ‘&’
  - Between MACRO and MEND is the body of the macro.
    - These are the statements that will be generated as the expansion of the macro definition.

Basic Functions[2]

- Macro expansion (or invocation)
  - Give the name of the macro to be expanded and the arguments to be used in expanding the macro.



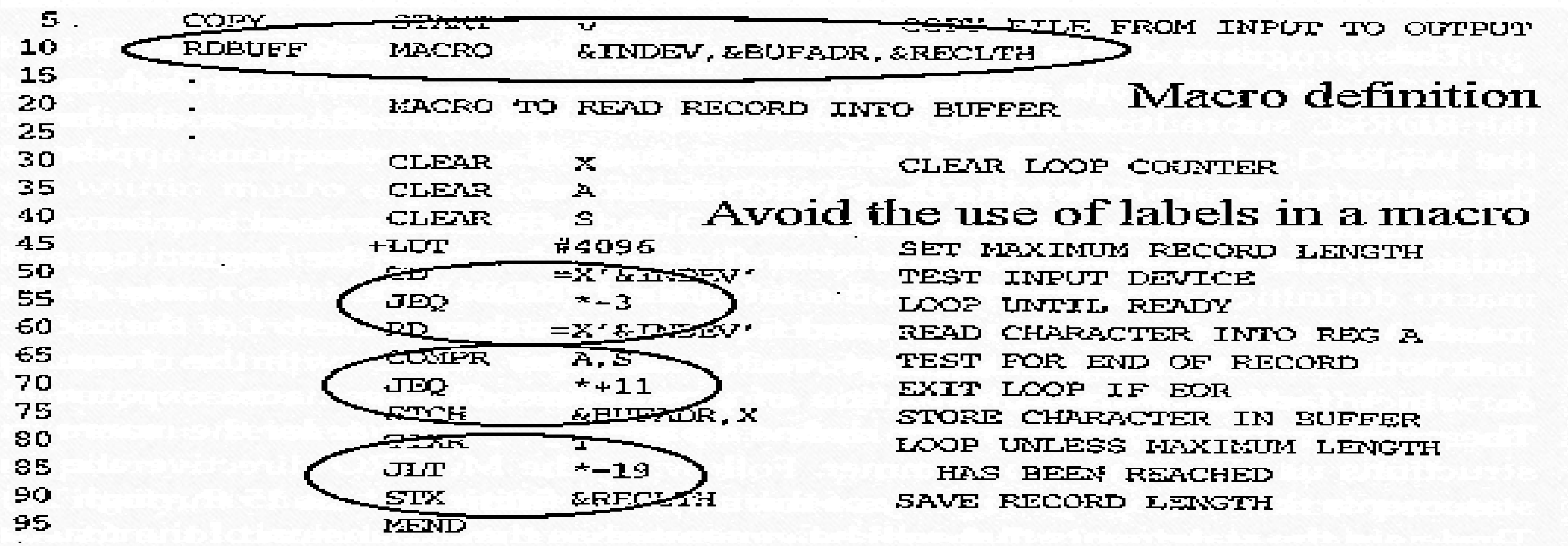
```
macro_name   p1, p2, …
```

  - Each macro invocation statement will be expanded into the statements that form the body of the macro, with 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 corresponds to the first parameter, and so on.

Retain Labels

- The label on the macro invocation statement 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.

Macro Program Example[1]

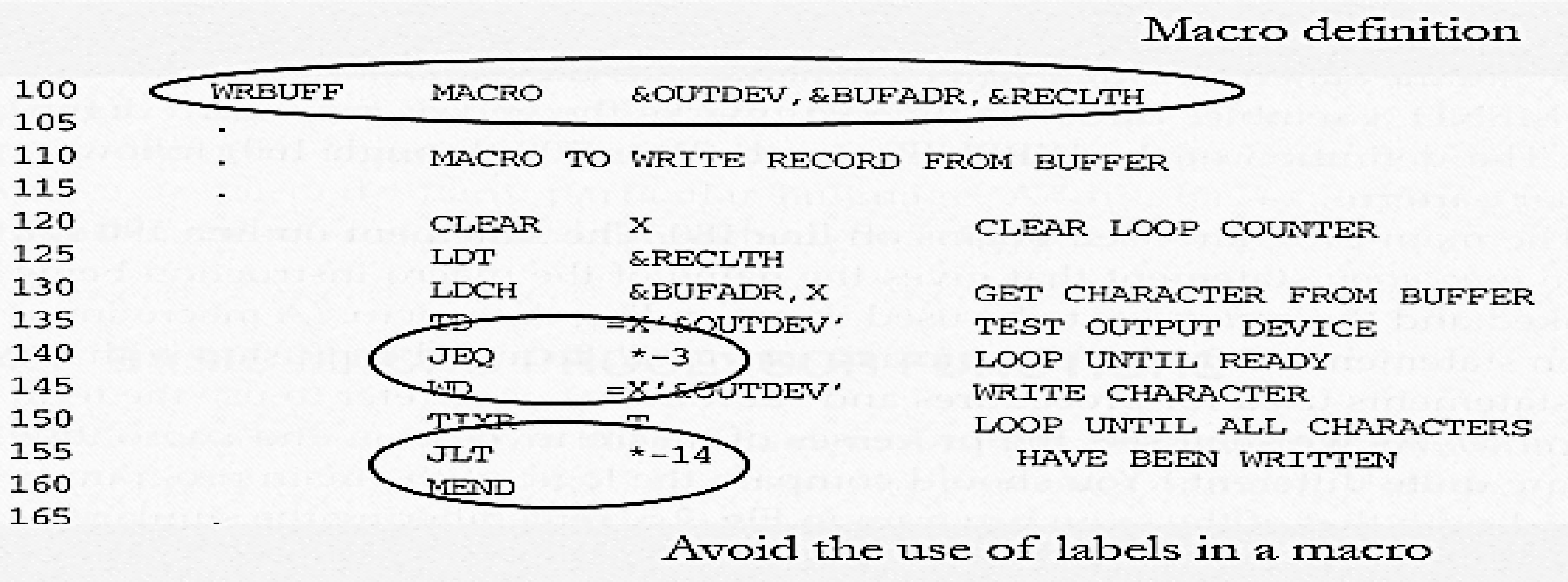


```
 5     COPY      START    0                   COPY FILE FROM INPUT TO OUTPUT
10     RDBUFF    MACRO    &INDEV,&BUFADR,&RECLTH
15     .
20     .                  MACRO TO READ RECORD INTO BUFFER
25     .
30               CLEAR    X                   CLEAR LOOP COUNTER
35               CLEAR    A
40               CLEAR    S
45              +LDT      #4096               SET MAXIMUM RECORD LENGTH
50               RD       =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
```

Macro definition

Avoid the use of labels in a macro

Macro Program Example[2]

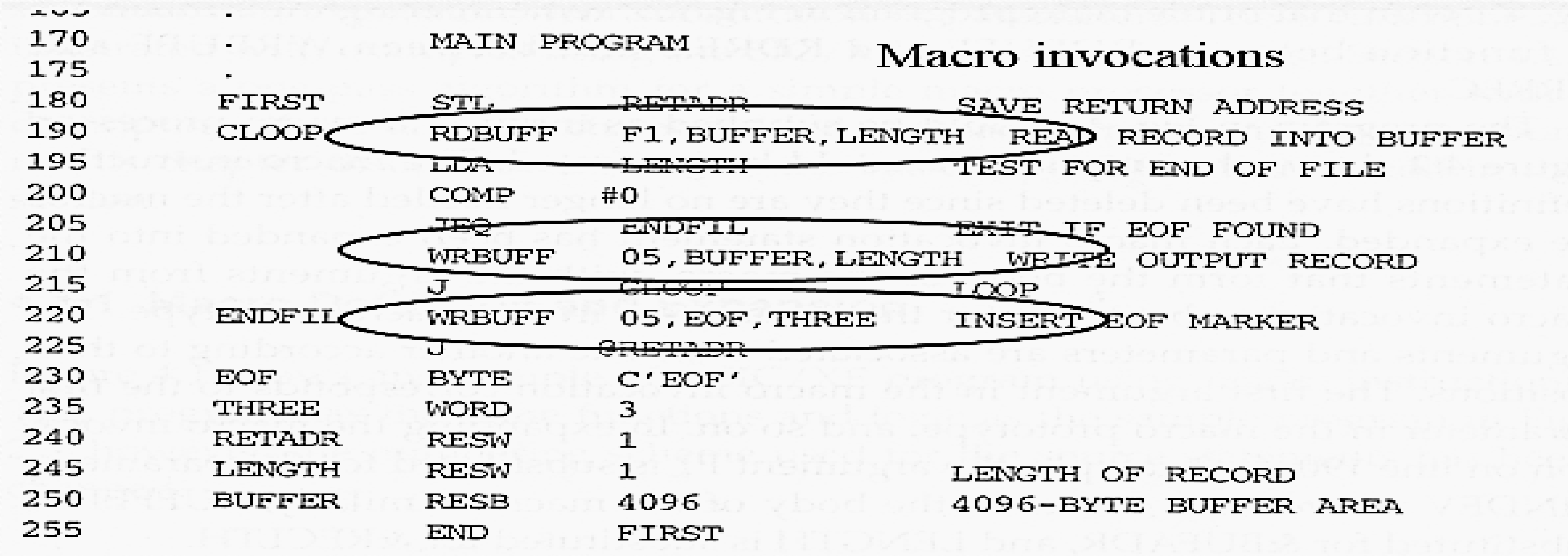


Macro definition

```
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
165    .
```

Avoid the use of labels in a macro

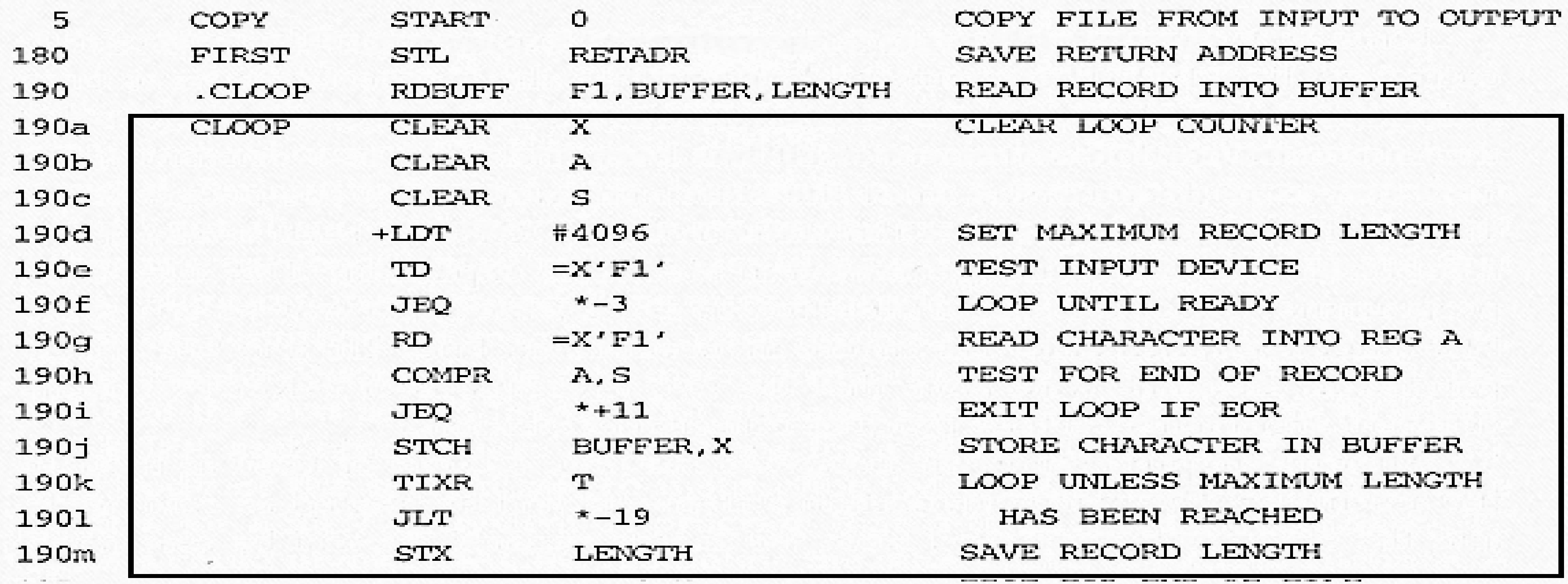
Macro Program Example[3]



Macro invocations

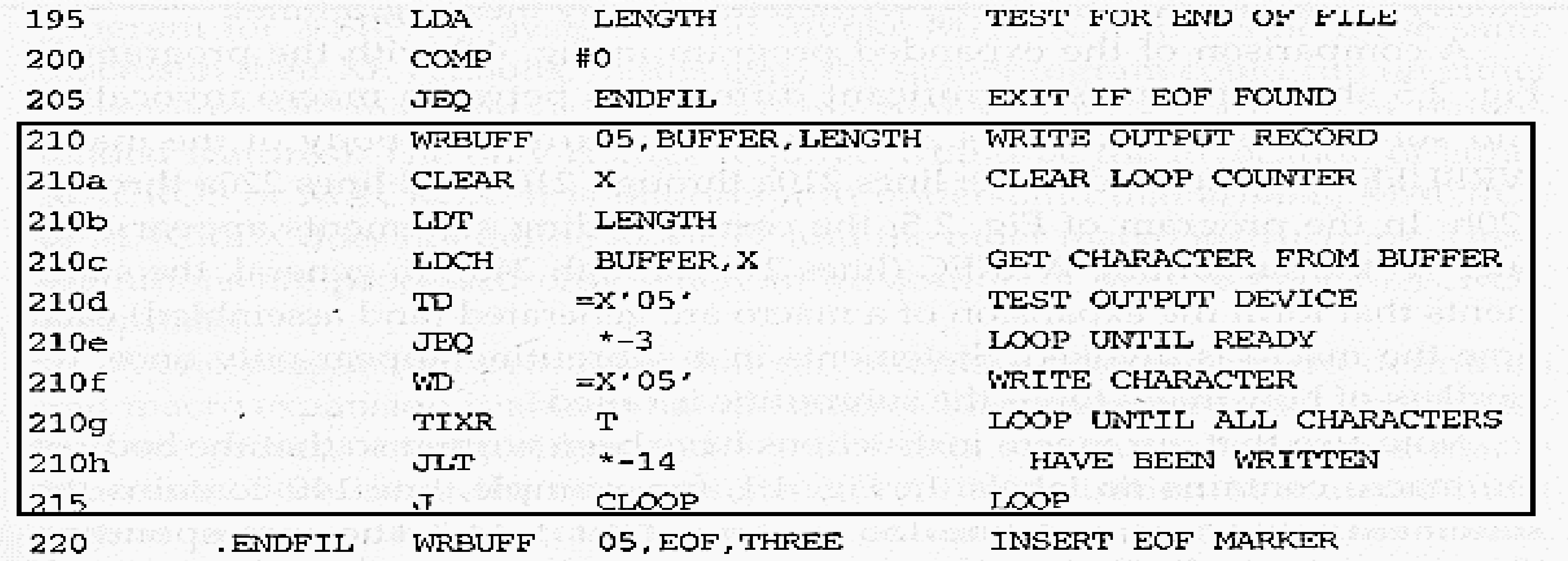
```
165        .
170        .              MAIN PROGRAM
175        .
180        FIRST          STL        RETADR               SAVE RETURN ADDRESS
190        CLOOP          RDBUFF     F1,BUFFER,LENGTH     READ RECORD INTO BUFFER
195                       LDA        LENGTH               TEST FOR END OF 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
```

Expanded Macro Example[4]



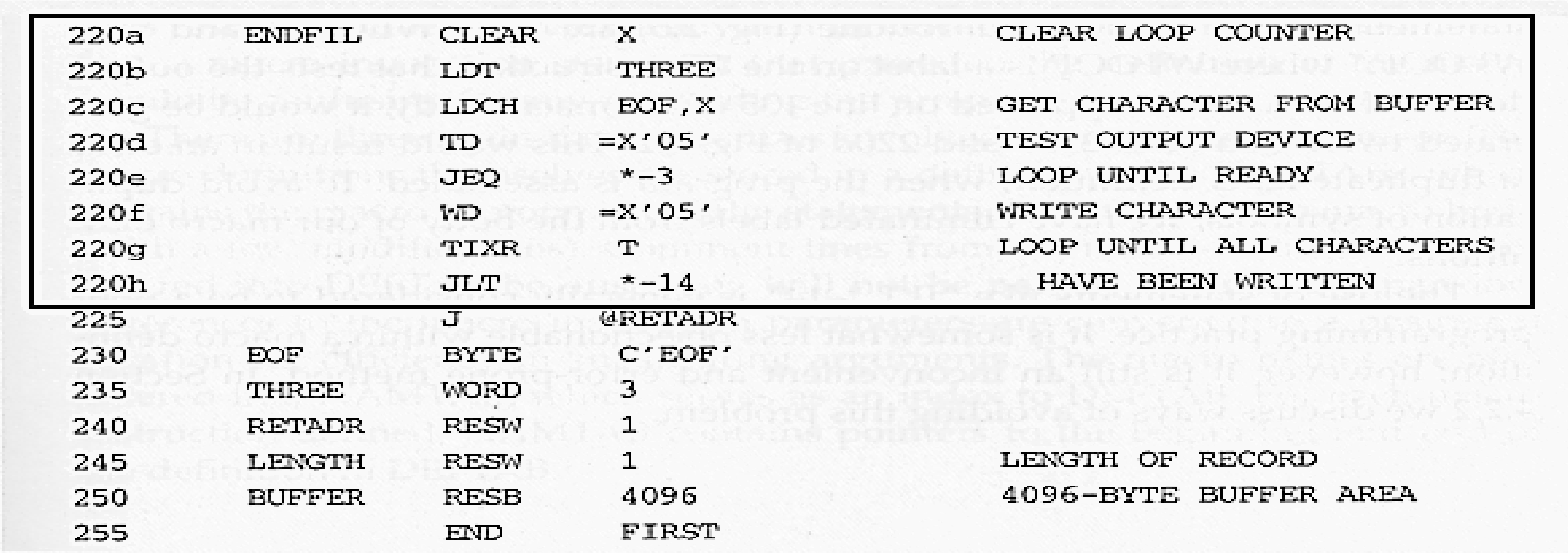
```
  5     COPY     START    0                   COPY FILE FROM INPUT TO OUTPUT
180     FIRST    STL      RETADR              SAVE RETURN ADDRESS
190     .CLOOP   RDBUFF   F1,BUFFER,LENGTH    READ RECORD INTO BUFFER
190a    CLOOP    CLEAR    X                   CLEAR LOOP COUNTER
190b             CLEAR    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
```

Expanded Macro Example[5]



```
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
```

Expanded Macro Example[6]



```
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
```

Labels in Macro Body

Problem

- 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.

Solution

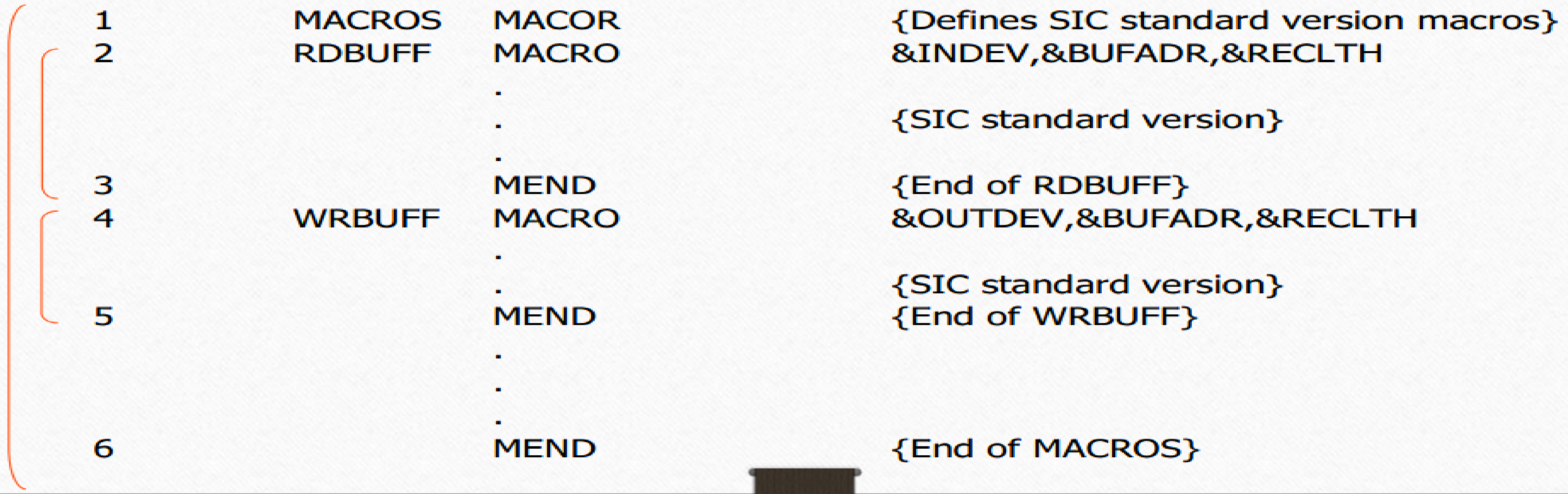
- Do not use labels in the body of macro.
- Explicitly use PC-relative addressing:
  - Ex, In RDBUFF and WRBUFF macros, many program-counter relative addressing instructions are used to avoid the uses of labels in a macro.
    - JEQ * + 11
    - JLT * - 14
  - *It is inconvenient and error-prone.*
- Later on, we will present a method which allows a programmer to use labels in a macro definition.

Two-Pass Macro Processor

- Like an assembler or a loader, we can design a two-pass macro processor in which:
  - *First pass:* process all macro definitions, and
  - *Second pass:* expand all macro invocation statements.
- However, such a macro processor cannot 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.

Macros(for SIC)

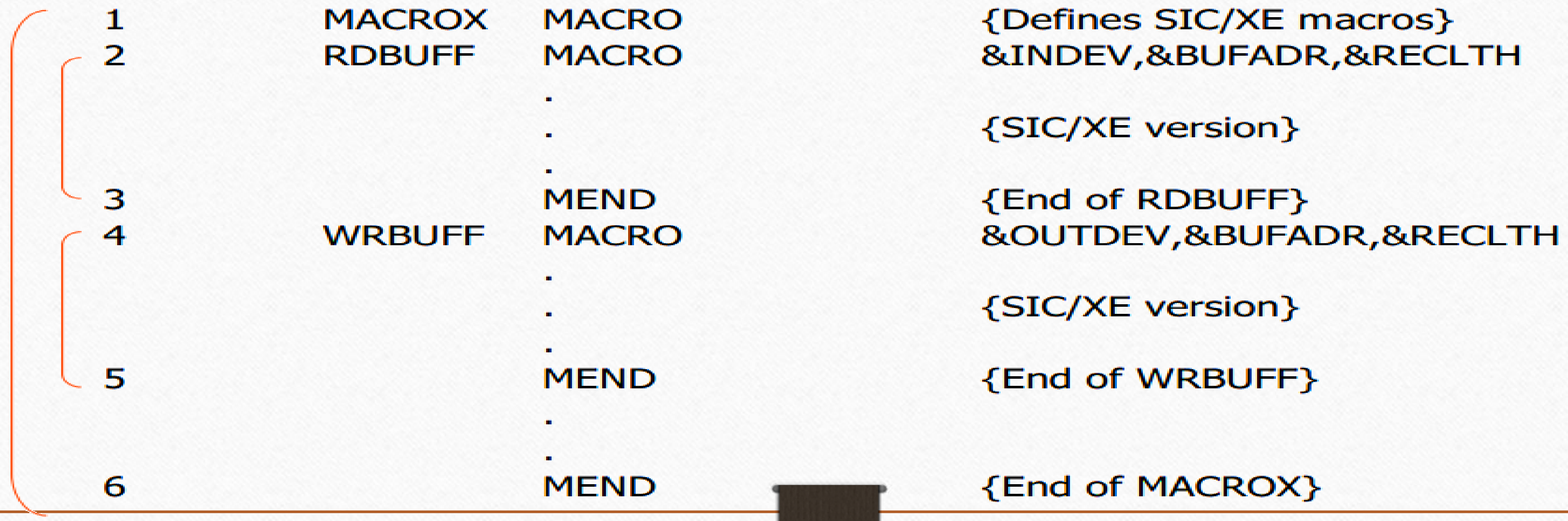
Contains the definitions of *RDBUFF* and *WRBUFF* written in SIC *instructions*.



```
1     MACROS    MACOR       {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(for SIC/XE)

Contains the definitions of *RDBUFF* and *WRBUFF* written in *SIC/XE instructions.*



```
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}
```

Macro Containing Macro Example

- MACROS contains the definitions of RDBUFF and WRBUFF which are written in SIC instructions.
- MACROX contains the definitions of RDBUFF and WRBUFF which are written in SIC/XE instructions.
- 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 MACROS or MACROX does not define RDBUFF and WRBUFF. These definitions are processed only when an invocation of MACROS or MACROX is expanded.

One-Pass Macro Processor

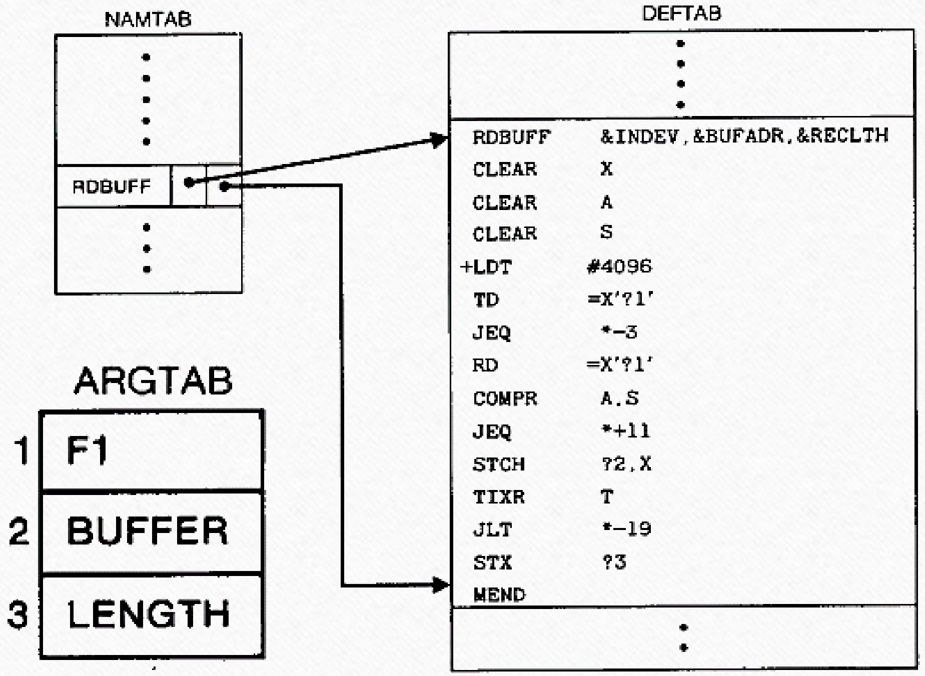
- A one-pass macro processor that alternate between macro definition and macro expansion is able to handle “macro in macro”.
- However, because of the one-pass structure, the definition of a macro must appear in the source program before any statements that invoke that macro.
    - This restriction is reasonable (does not create any real inconvenience).

Data Structures-- *Global Variables*

- Three main data structures involved in an one-pass macro processor:
  - *DEFTAB*
    - Stores the macro definition including *macro prototype* and *macro body.*
    - Comment lines are omitted.
    - References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.
  - *NAMTAB*
    - Store macro names, which serves an index to DEFTAB contain pointers to the beginning and end of the definition
  - *ARGTAB*
    - Used during the expansion of macro invocations.
    - When a macro invocation statement is encountered, the arguments are stored in this table according to their position in the argument list.

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.



```
 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

Algorithm

- Procedure DEFINE
  - Called when the beginning of a macro definition is recognized. Make appropriate entries in DEFTAB and NAMTAB.
- Procedure EXPAND
  - Called to set up the argument values in ARGTAB and expand a macro invocation statement
- Procedure GETLINE
  - Get the next line to be processed

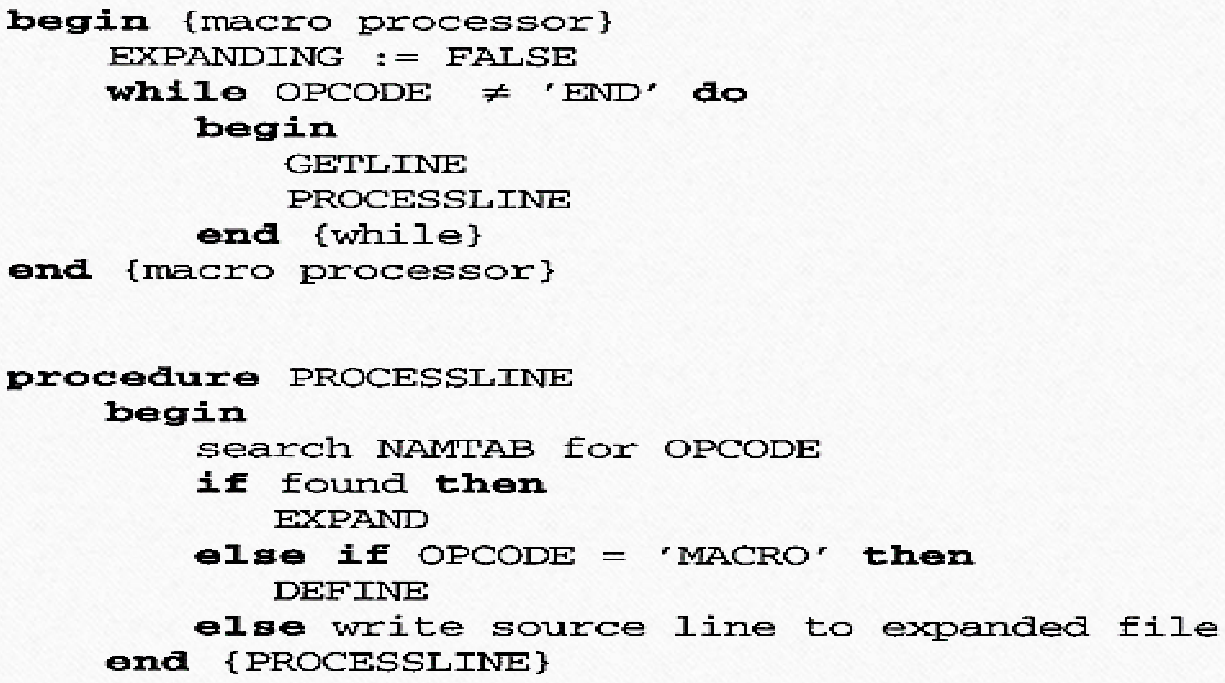
Handle Macro in Macro(Nested Macro)

- When a macro definition is being entered into DEFTAB, the normal approach is to continue until an MEND directive is reached.
- This will not work for "*macro in macro*" because the MEND first encountered (for the inner macro) will terminate the whole macro definition process.
- *To solve this problem*, a counter LEVEL is used to keep track of the level of macro definitions.
  - Increase LEVEL by 1 each time a MACRO directive is read and decrease LEVEL by 1 each time a MEND directive is read.
  - A MEND terminates the whole macro definition process when LEVEL reaches 0.
  - This is very much like matching left and right parentheses when scanning an arithmetic expression.

Nested Macro Definition Example

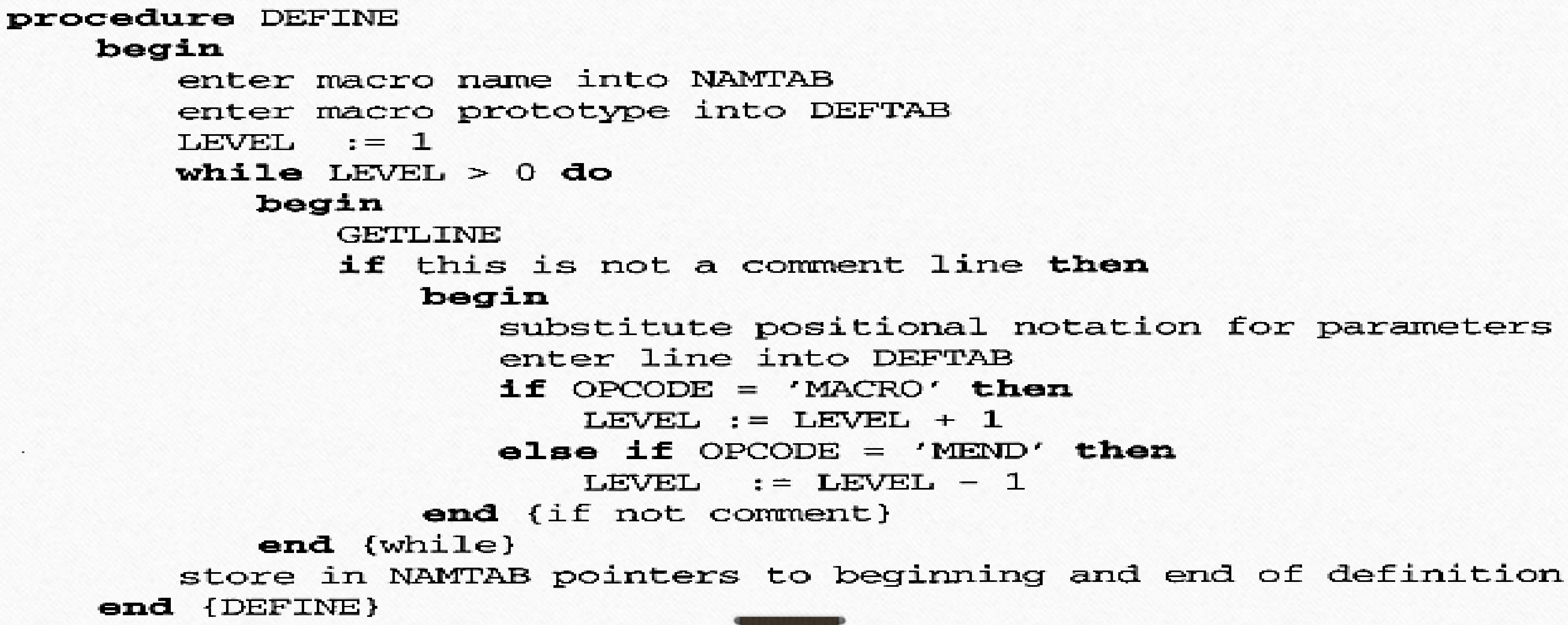
```
TEST                START               2000h
MACROS              MACRO
CELTOFER MACRO                &CEL      &FER
                    LDA                 &CEL
                    MULT                NINE
                    DIV                 FIVE
                    ADD                 THIRTYTWO
                    STA                 &FER
                    MEND
                    MEND

MACROF              MACRO
CELTOFER MACRO                &CEL      &FER
                    LDAF                &CEL
                    MULTF               NINE
                    DIVF                FIVE
                    ADDF                THIRTYTWO
                    STAF                &FER
                    MEND
                    MEND
```

Algorithm for One Pass Macro Processor[1]



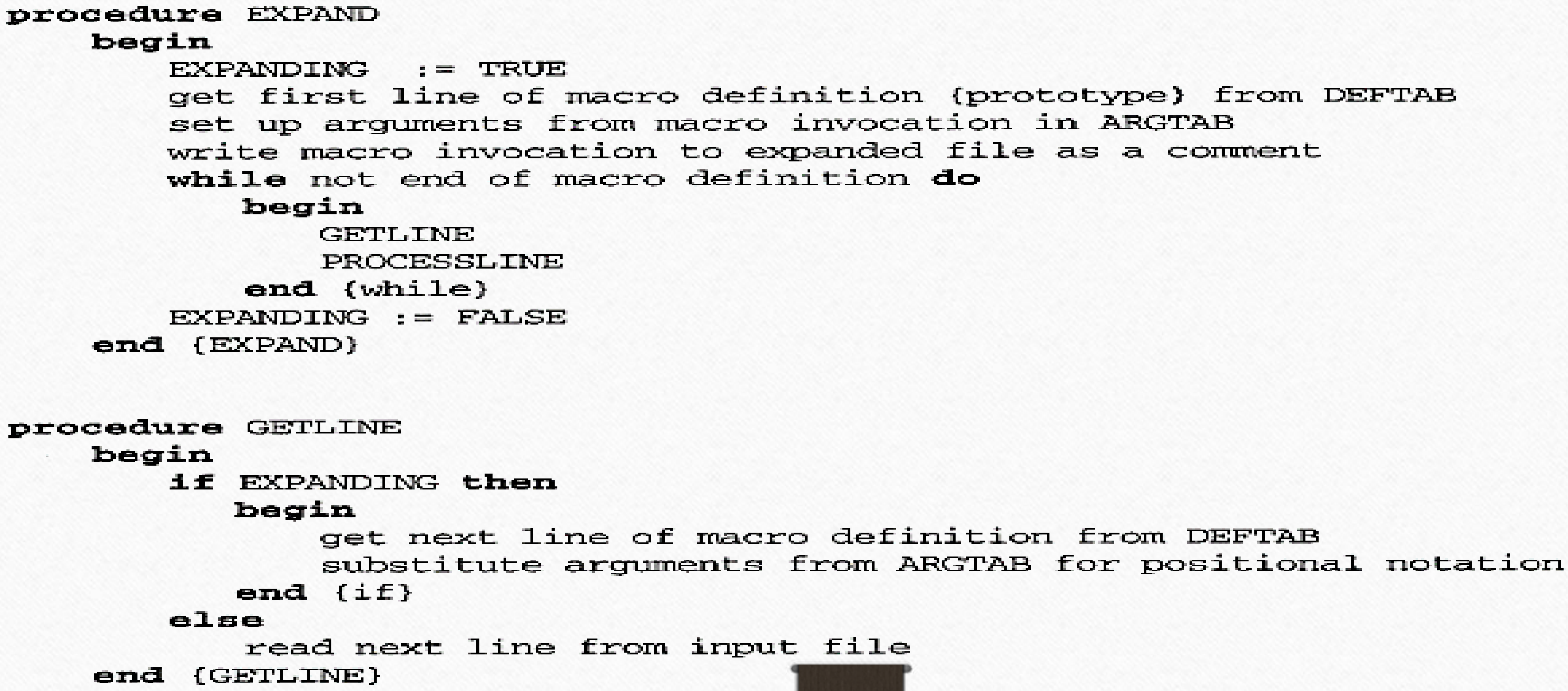
```
begin {macro processor}
     EXPANDING := FALSE
     while OPCODE ≠ 'END' do
          begin
               GETLINE
               PROCESSLINE
          end {while}
end {macro processor}


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}
```

Algorithm for One Pass Macro Processor[2]



```
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}
```

Algorithm for One Pass Macro Processor[3]



```
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}
```

Two-Pass **Vs** *One-Pass* **Macro Processor**

Two Pass Macro Processor

- Passes:
  - Pass1: Recognize macro definitions
  - Pass2: Recognize macro calls
- Nested macro definitions are not allowed.

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 but nested calls are not

Machine Independent Features

Extensions to the basic macro processor functions

Concatenation of Macro Parameters

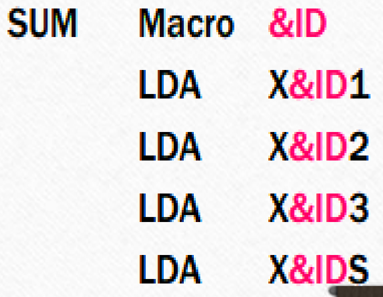
Generation of Unique Labels

Conditional Macro Expansion

Keyword Macro Parameters

Concatenation of Macro Parameters[1]

- Most macro processors allow parameters to be concatenated with other character stings.
- 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
```

Concatenation of Macro Parameters[2]

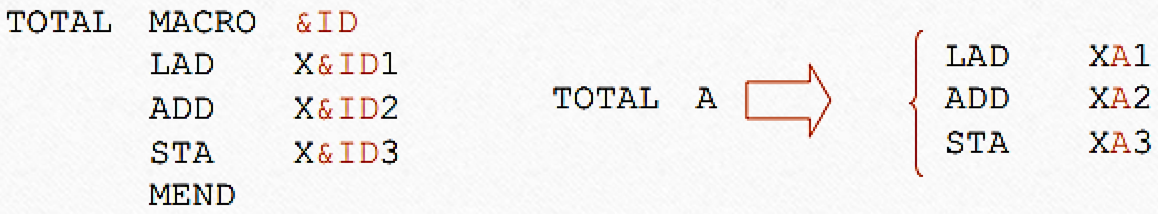
- Used when a program contains a set of series of variables.
- Suppose the parameter is named &ID, the macro body may contain a statement:
- LDA X&ID1
- &ID is concatenated after the string "X" and before the string "1".

  LDA XA1 (&ID=A)

  LDA XB1 (&ID=B)

Concatenation of Macro Parameters[3]

- Example



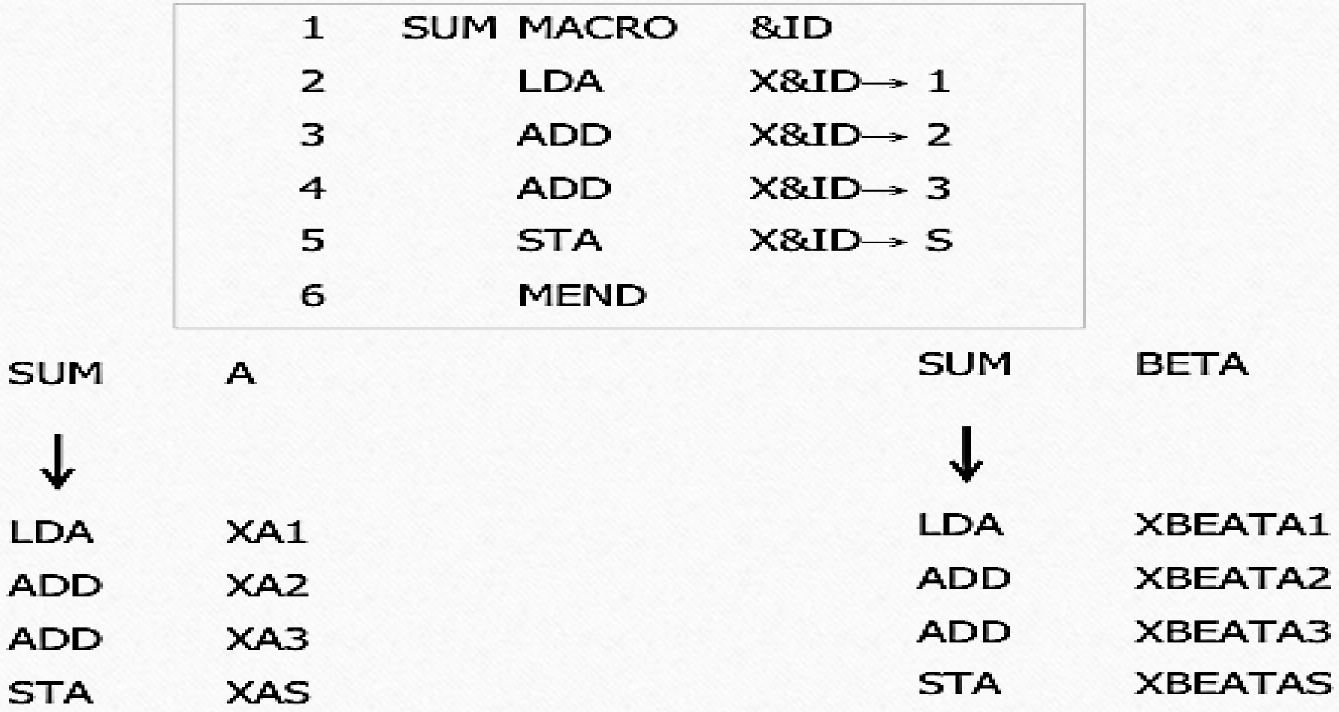
```
TOTAL  MACRO  &ID
       LAD    X&ID1
       ADD    X&ID2
       STA    X&ID3
       MEND
```

```
TOTAL  A
```

```
LAD    XA1
ADD    XA2
STA    XA3
```

- Problem
  - Ambiguous Situation
  - The problem is that the end of the parameter is not marked. Thus X**&ID**1 may mean “X” + ID + “1” or “X” + ID1.
- Solution
  - To avoid this ambiguity, a special concatenation operator **->** is used to specify the end of the parameter.
    - The new form becomes X&ID->1.
    - Of course, **->** will not appear in the macro expansion.

Concatenation of Macro Parameters[4]



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

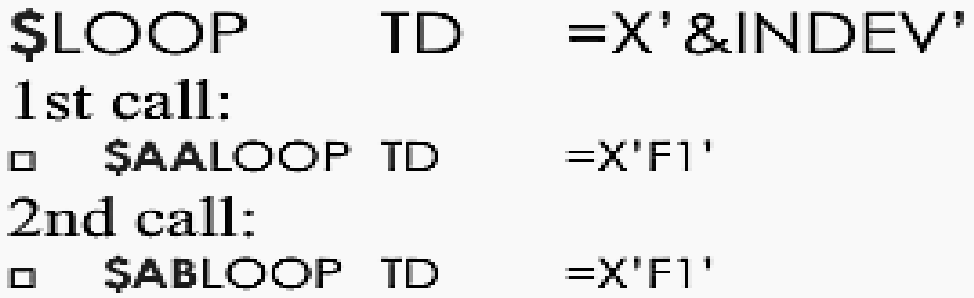
```
SUM       A                         SUM       BETA

↓                                   ↓

LDA       XA1                       LDA       XBEATA1
ADD       XA2                       ADD       XBEATA2
ADD       XA3                       ADD       XBEATA3
STA       XAS                       STA       XBEATAS
```

Generation of Unique Labels[1]

- It is in general not possible for the body of a macro instruction to contain labels of the usual kind.
  - Leading to the use of relative addressing at the source statement level
    - Only be acceptable for short jumps
- Solution:
  - Allowing the creation of special types of labels within macro instructions
- Previously we see that, without special processing, if labels are used in macro definition, we may encounter the "*duplicate labels*" problem if a macro is invocated multiple time.

Generation of Unique Labels[2]

- To generate unique labels for each macro invocation, when writing macro definition, we must begin a label with *$*.
- During macro expansion, the *$* will be replaced with *$xx*, where xx is a two-character alphanumeric counter of the number of macro instructions expanded.
  - XX will start from AA, AB, AC,…..



```
$LOOP      TD      =X'&INDEV'
```

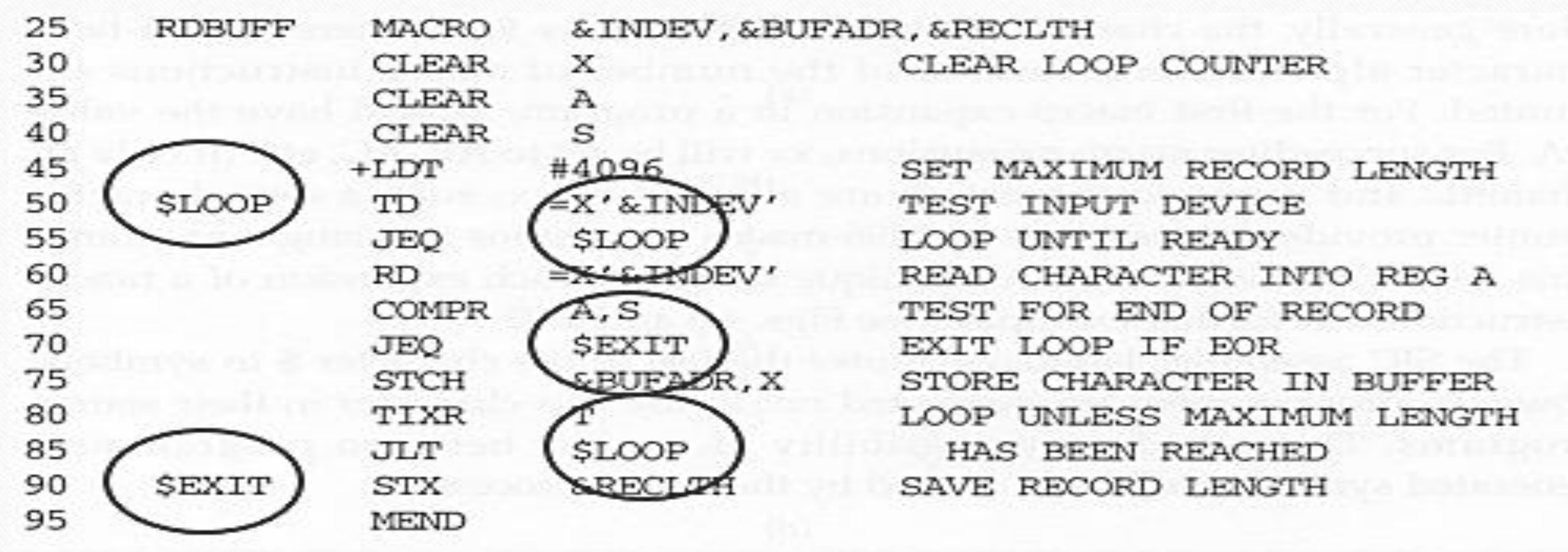
1st call:

```
□    $AALOOP  TD      =X'F1'
```

2nd call:

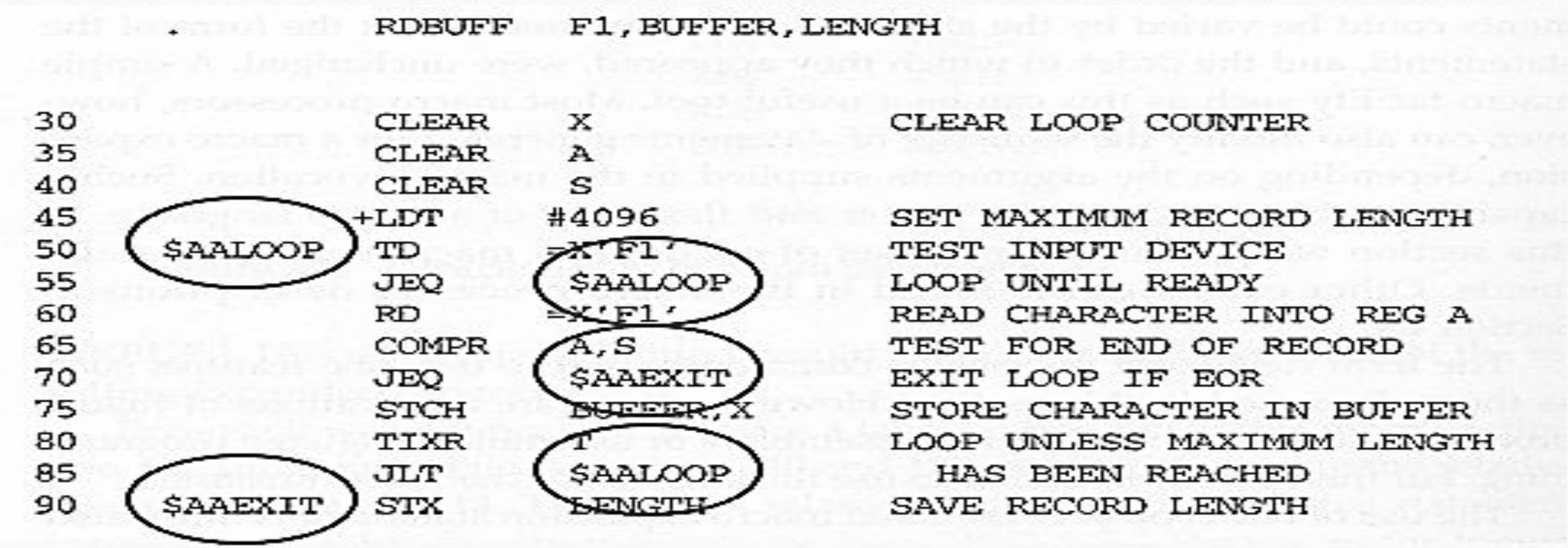
```
□    $ABLOOP  TD      =X'F1'
```

Generation of Unique Labels[3]



```
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
```

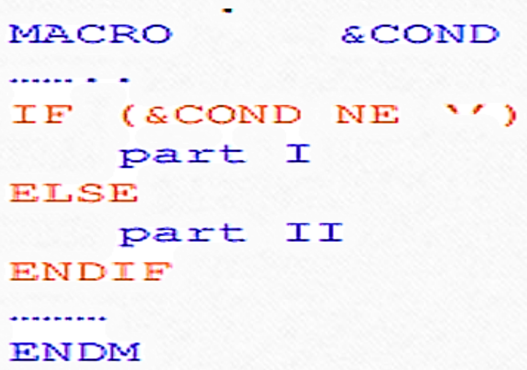
Generation of Unique Labels[4]



```
.                  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[1]

- Most macro processors can modify the sequence of statements generated for a macro expansion, depending on the arguments supplied in the macro invocation.



```
MACRO        &COND
...... . .
IF (&COND NE '')
    part I
ELSE
    part II
ENDIF
.........
ENDM
```

- Part I is expanded if condition part is true, otherwise part II is expanded
- Compare operator: NE, EQ, LE, GT

Conditional Macro Expansion[2]

- So far, when a macro instruction is invoked, the same sequence of statements are used to expand the macro.
- Here, we allow conditional assembly to be used.
  - Depending on the arguments supplied in the macro invocation, the sequence of statements generated for a macro expansion can be modified.
- Conditional macro expansion can be very useful.
- It can generate code that is suitable for a particular application.

Conditional Macro Expansion[3]

- In the following example, the values of &EOR and &MAXLTH parameters are used to determine which parts of a macro definition need to be generated.
- There are some *macro-time control structures* introduced for doing conditional macro expansion:
  - IF- ELSE-ENDIF
  - WHILE-ENDW
- *Macro-time variables(also called a set symbol)* can also be used to store values that are used by these macro-time control structures.
  - Used to store the Boolean expression evaluation result
  - A variable that starts with & but not defined in the parameter list is treated as a macro-time variable.

Conditional Macro Expansion[4]

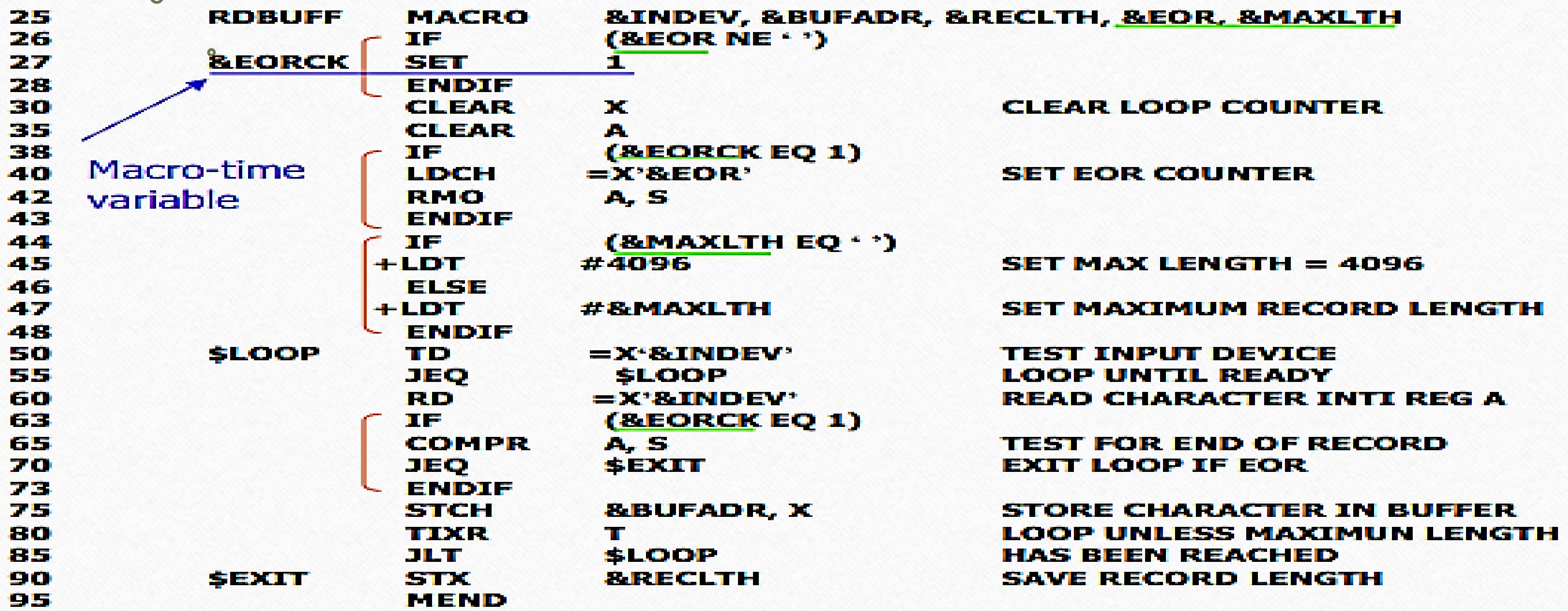
Macro-time Variables[4.1]

- *Macro-time conditional statements*
  - Macro processor directives:
    - *IF-ELSE-ENDIF*
    - *SET*
- *Macro-time variables* (also called a *set symbol*)
  - Begins with "&" but is not a macro instruction parameter ➔ any symbol that begins with the character & and is *not a macro parameter*
  - Be used to store working values during the macro expansion:
    - *Store the evaluation result of Boolean expression*
    - *Control the macro-time conditional structures*
  - Be initialized to 0
  - Be changed with their values using SET directives
    - &EORCK SET 1

Macro-time variable

Conditional Macro Expansion[5]

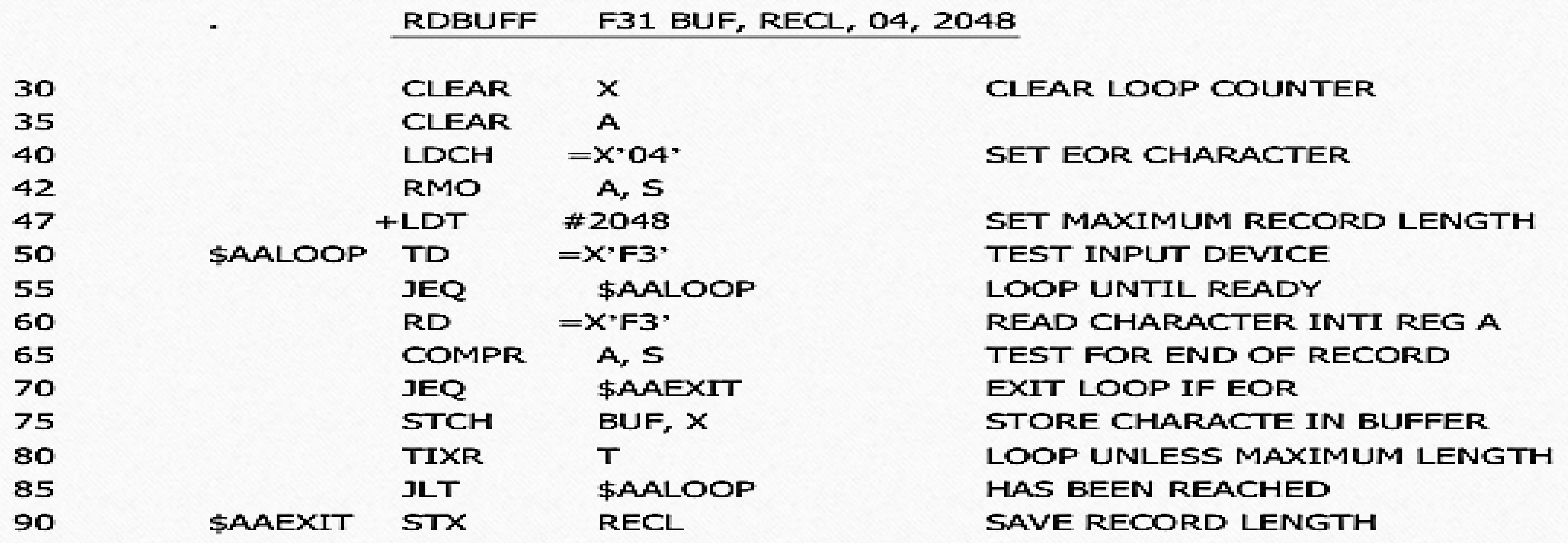
Macro-time Variables[4.2]



```
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 COUNTER
         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 INTI 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 MAXIMUN LENGTH
         JLT      $LOOP                  HAS BEEN REACHED
$EXIT    STX      &RECLTH                SAVE RECORD LENGTH
         MEND
```

Conditional Macro Expansion[6]

Macro-time Variables[4.3]

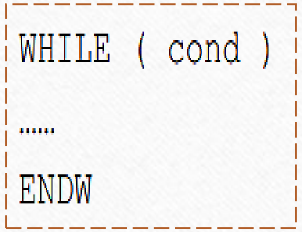


```
          .           RDBUFF     F31 BUF, RECL, 04, 2048

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 INTI REG A
65                    COMPR      A, S                       TEST FOR END OF RECORD
70                    JEQ        $AAEXIT                    EXIT LOOP IF EOR
75                    STCH       BUF, X                     STORE CHARACTE IN BUFFER
80                    TIXR       T                          LOOP UNLESS MAXIMUM LENGTH
85                    JLT        $AALOOP                    HAS BEEN REACHED
90        $AAEXIT     STX        RECL                       SAVE RECORD LENGTH
```

Conditional Macro Expansion[7]

Macro-time Looping Statements[4.1]

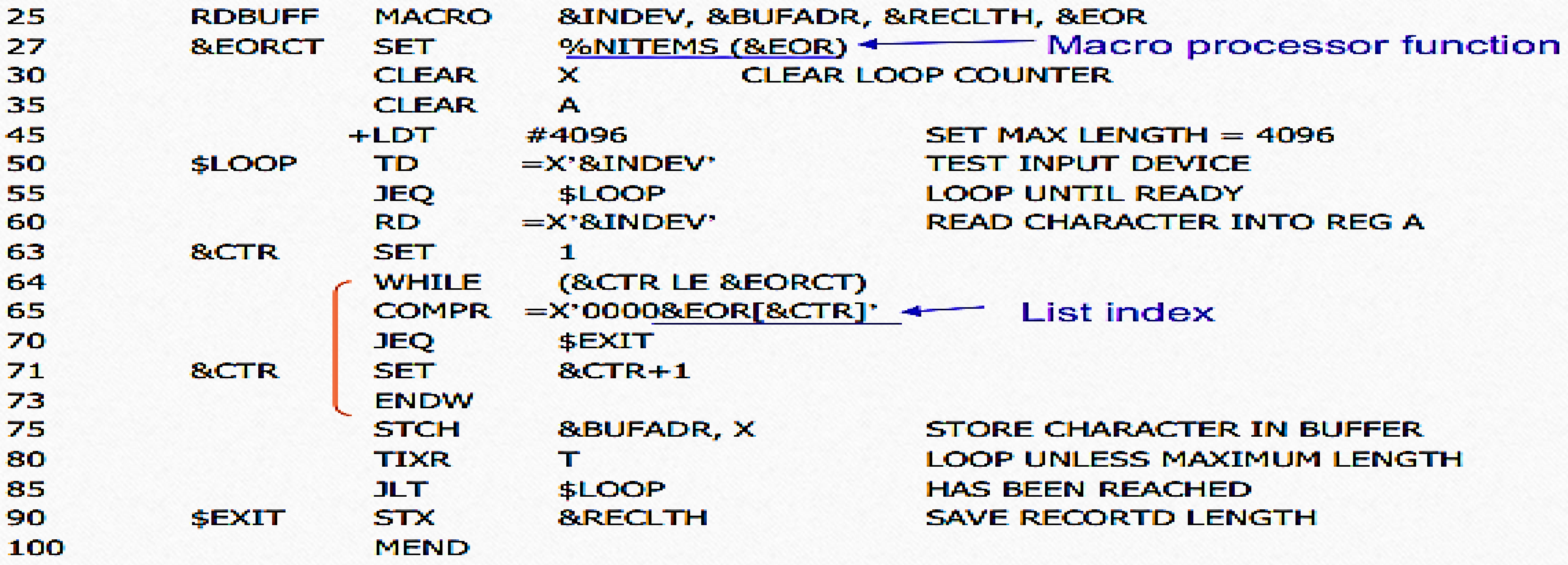
- Macro processor function
  - %NITEMS: is a macro processor function that returns as its value the number of members in an argument list.
  - The execution of testing of IF/WHILE, SET,
- %NITEMS() occurs at macro expansion time



```
WHILE ( cond )
......
ENDW
```

Conditional Macro Expansion[8]

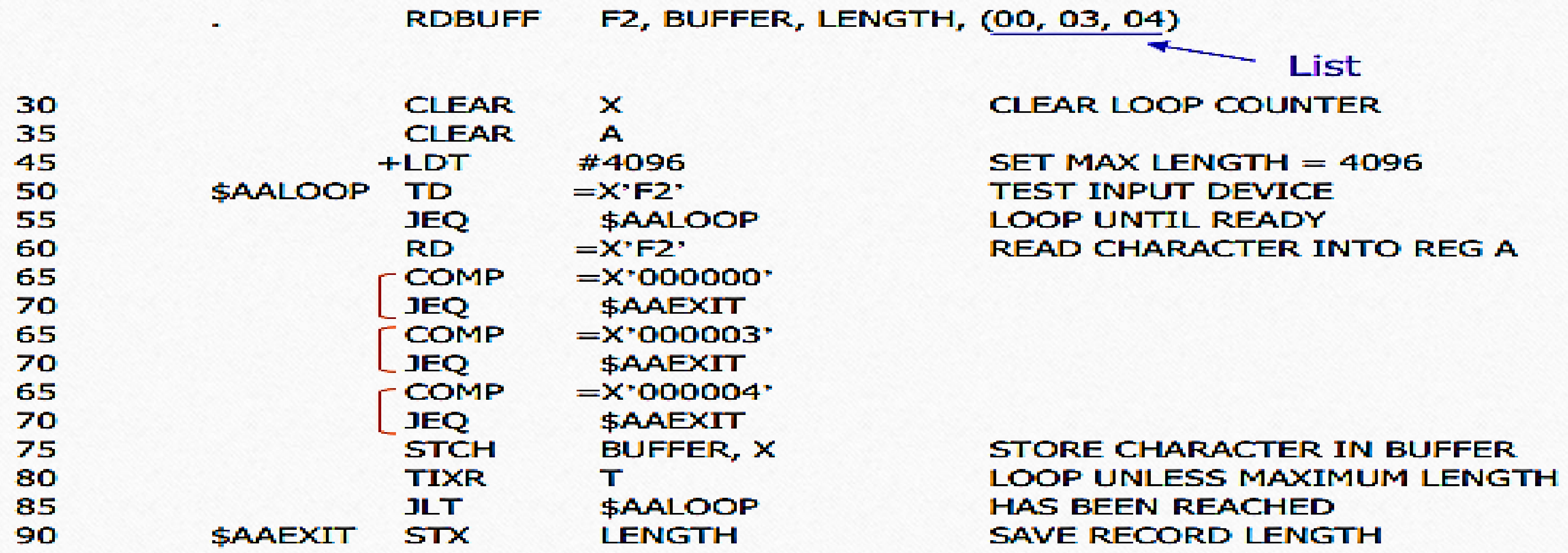
Macro-time Looping Statements[4.2]



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

Conditional Macro Expansion[9]

Macro-time Looping Statements



```
     .           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
```

Conditional Macro Expansion[10]

Conditional Macro Expansion Vs Conditional Jump Instructions

- The testing of Boolean expression in IF statements occurs at the time macros are expanded.
- By the time the program is assembled, all such decisions have been made.
- There is only one sequence of source statements during program execution.
- In contrast, the COMPR instruction test data values during program execution. The sequence of statements that are executed during program execution may be different in different program executions.

Conditional Macro Expansion[11]

Implementation[11.1]

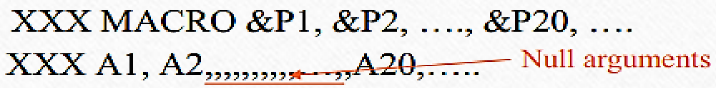
- The macro processor must maintain a symbol table:
  - This table contains the values of *all macro-time variables* used.
  - Entries in this table are made or modified when SET statements are processed.
  - This table is used to look up the current value of a macro-time variable whenever it is required.
- When an IF statement is encountered during the expansion of a macro, the specified Boolean expression is evaluated.
  - *If the value of this expression is TRUE*, the macro processor continues to process until it encounters the next ELSE or ENDIF.
    - If ELSE is encountered, then skips to ENDIF
  - *Otherwise*, the assembler skips to ELSE and continues to process until it reaches ENDIF.

Conditional Macro Expansion[12]

Implementation[11.2]

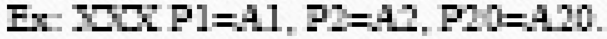
- When a WHILE statement is encountered during the expansion of a macro, the specified Boolean expression is evaluated.
- *If the value of this expression is TRUE:*
  - The macro processor continues to process lines from DEFTAB until it encounters the next ENDW statement.
  - When ENDW is encountered, the macro processor returns to the preceding WHILE, re-evaluates the Boolean expression, and takes action based on the new value.
- *Otherwise:*
  - The macro processor skips ahead in DEFTAB until it finds the next ENDW statement and then resumes normal macro expansion.

Keyword Macro Parameters[1]

- So far, all macro instructions use positional parameters.
  - If an argument is to be omitted, the macro invocation statement must contain a null argument to maintain the correct argument positions.
  - E.g., 

    XXX MACRO &P1, &P2, …., &P20, ….

    XXX A1, A2,,,,,,,,,,,…,,A20,…..

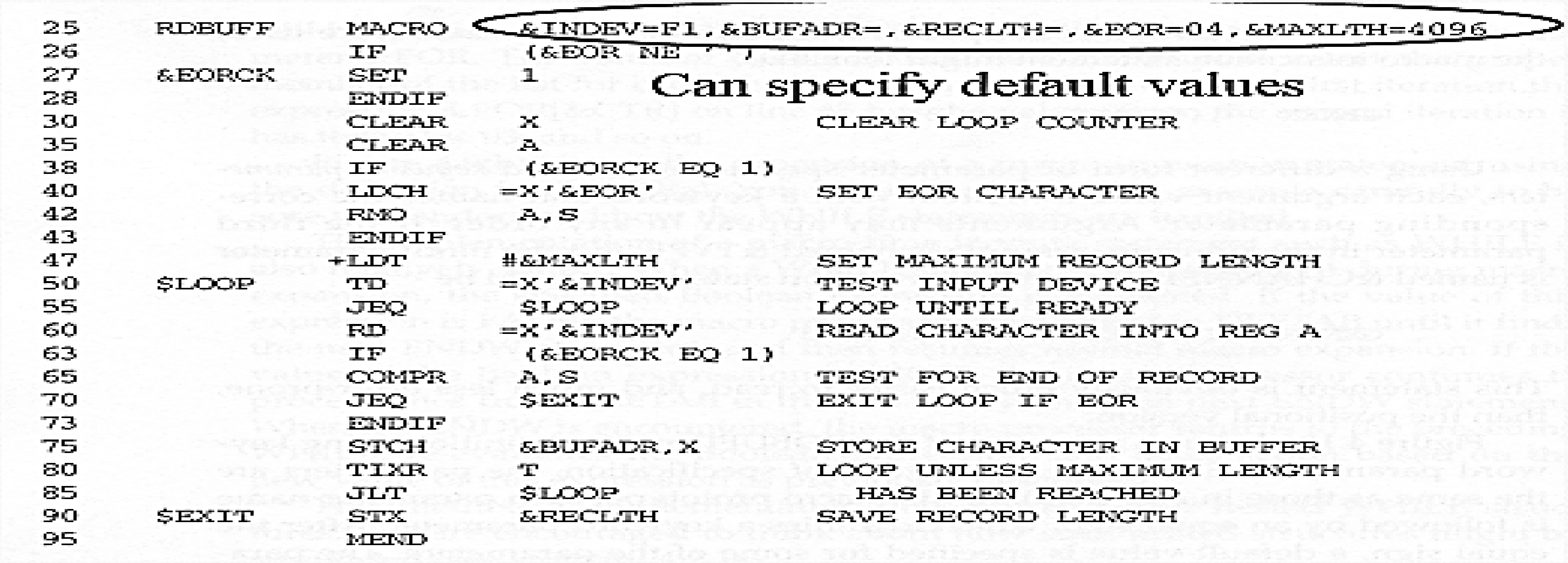
    Null arguments
- If keyword parameters are used, each argument value is written with a keyword that names the corresponding parameters.
  - Arguments thus can appear in any order.
  - Null arguments no longer need to be used.
  - 

    Ex: XXX P1=A1, P2=A2, P20=A20.
- Keyword parameter method can make a program easier to read than the positional method.

Keyword Macro Parameters[2]

- *Keyword parameters*
  - Each argument value is written with a keyword that names the corresponding parameter.
  - Arguments may appear in any order.
    - Null arguments no longer need to be used.
  - E.g. GENER TYPE=DIRECT, CHANNEL=3
  - It is easier to read and much less error-prone than the positional method. E.g. Fig. 4.10

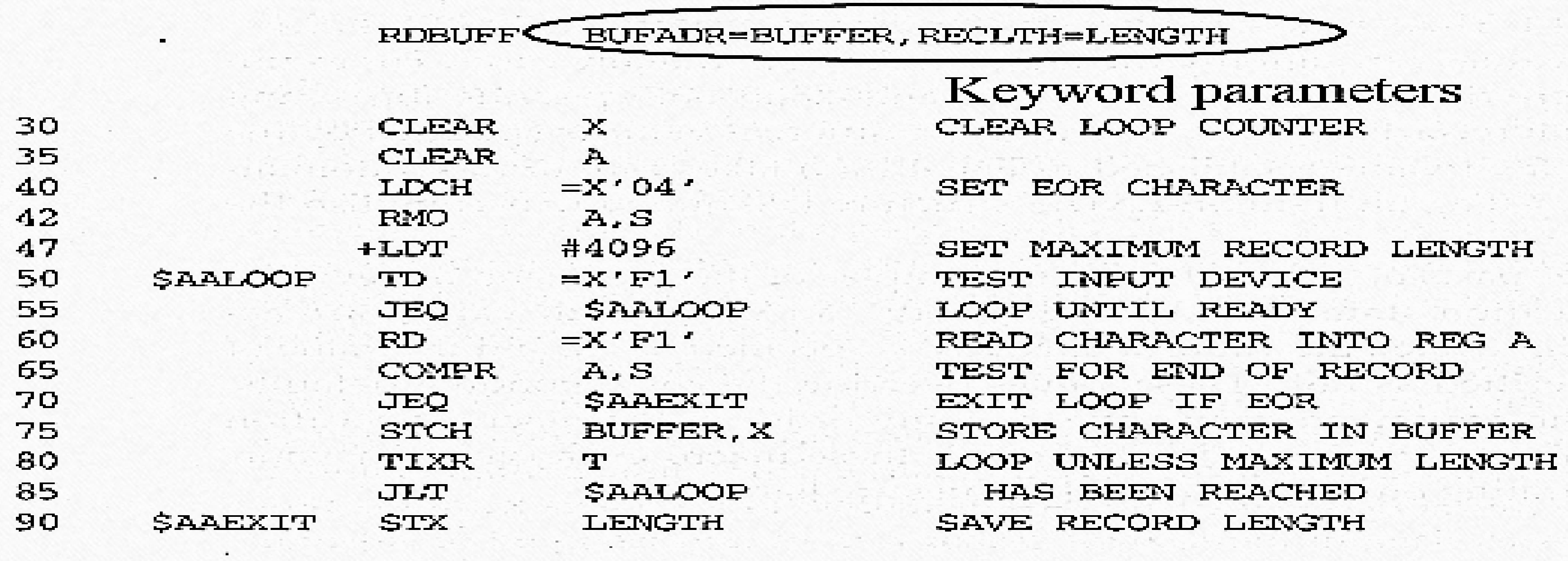
Keyword Macro Parameters[3]



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

Can specify default values

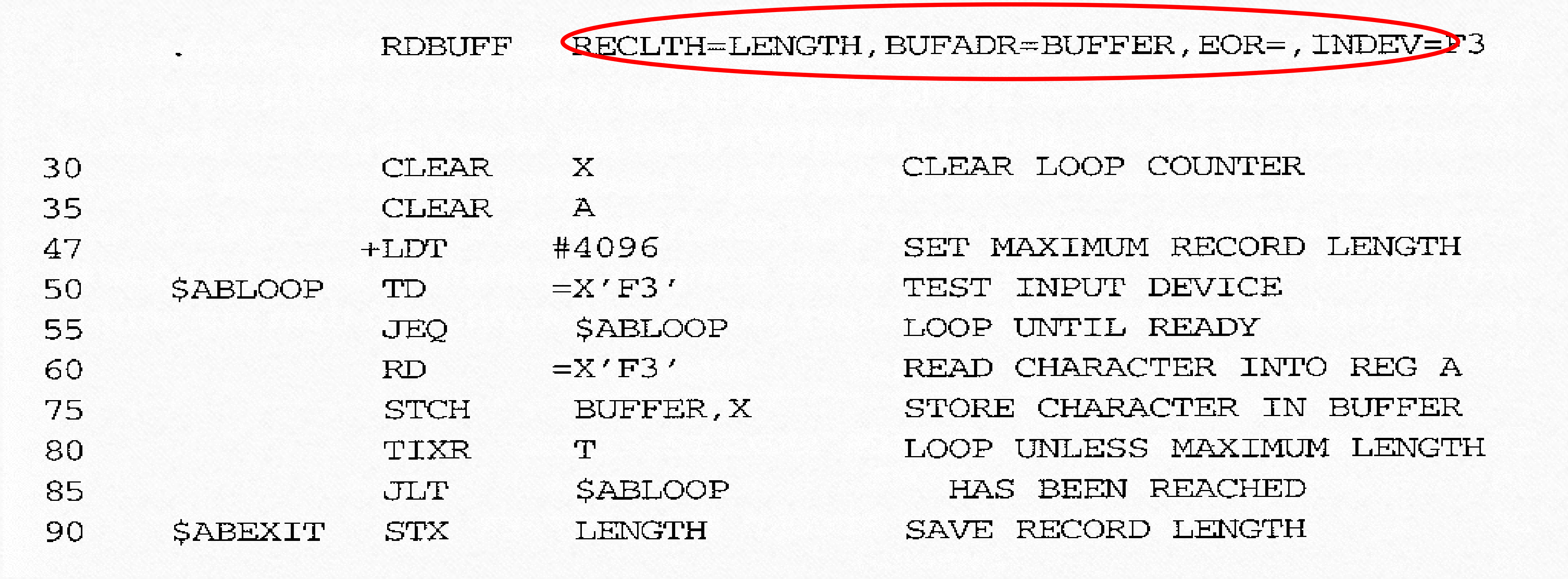
Keyword Macro Parameters[4]



```
          .          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
             .
```

Keyword parameters

Keyword Macro Parameters[5]



```
 .              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
```

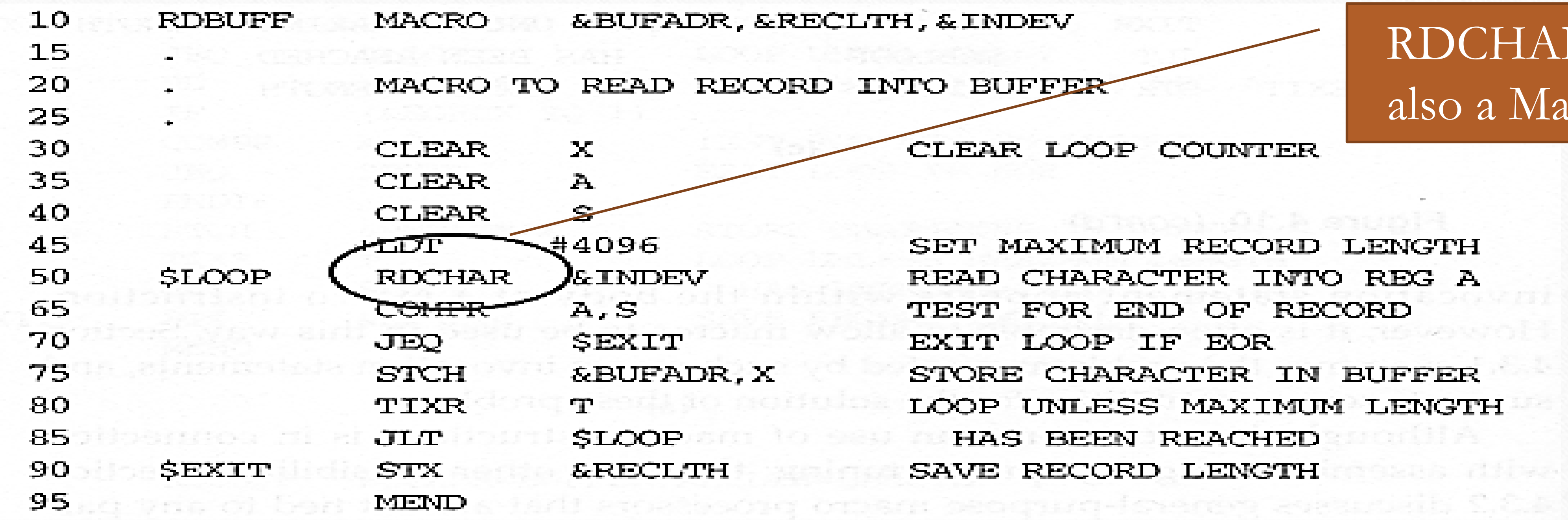
Macro Processor Design Options

Recursive macro expansion
General-purpose macro processors
Macro processing within language translators

Recursive Macro Expansion[1]

- If we want to allow a macro to be invoked in a macro definition, the already presented macro processor implementation cannot be used.
- This is because the EXPAND routine is recursively called but the variable used by it (e.g., EXPANDING) *is not saved across these calls*.
- It is easy to solve this problem if we use a programming language that support recursive functions. (e.g., C or C++).

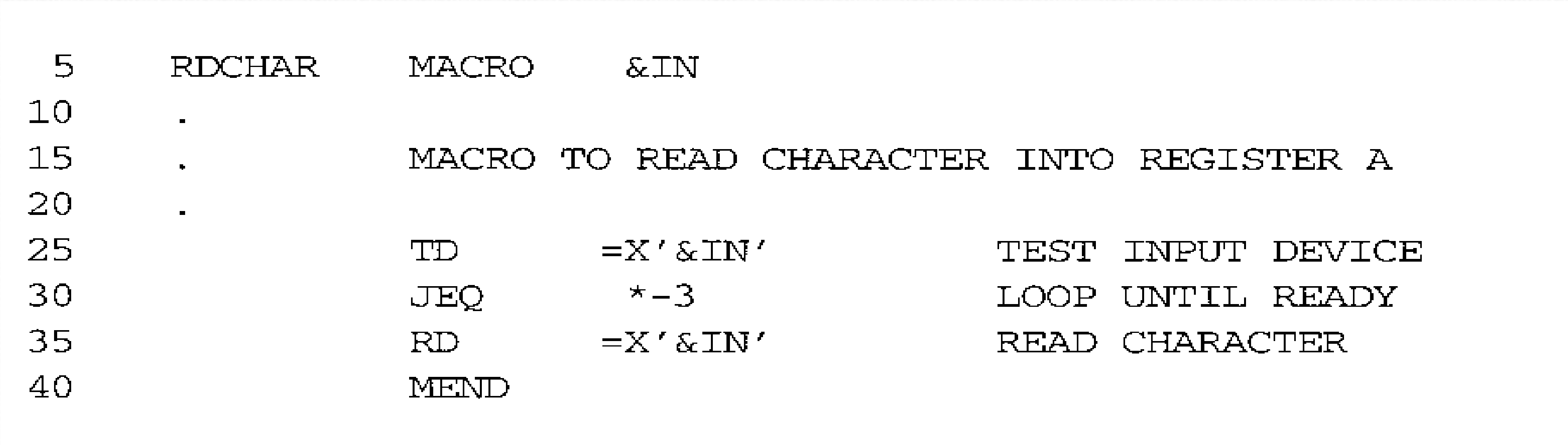
Recursive Macro Expansion[2]



```
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
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
```

RDCHAR is also a Macro

Recursive Macro Expansion[3]



```
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
```

- *For easy implementation, we require that RDCHAR macro be defined before it is used in RDBUFF macro.*
- *This requirement is very reasonable.*

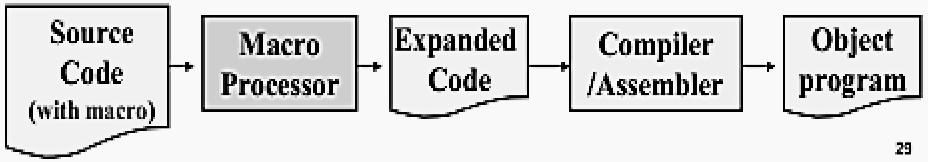
Recursive Macro Expansion[4]

Solutions

- Write the macro processor in a programming language that allows recursive calls. Thus, local variables will be retained.
  - Most high-level language have been supported recursive calls
    - *The compiler would be sure that previous values of any variables declared within a procedure were saved when the procedure was called recursively*
- If you are writing in a language without recursion support, Use a *stack* to take care of *pushing and popping local variables* and *return addresses*

General-Purpose Macro Processors[1]

- Macro processors that *do not dependent* on any particular programming language, but can be *used with a variety* of different languages.
  - Not tied to any particular language
  - Can be used with a variety of different languages.

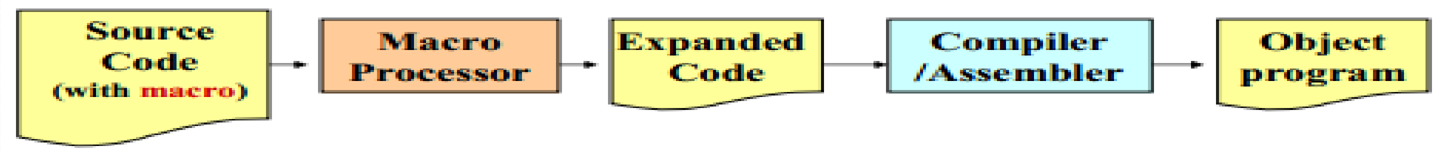


General-Purpose Macro Processors[2]

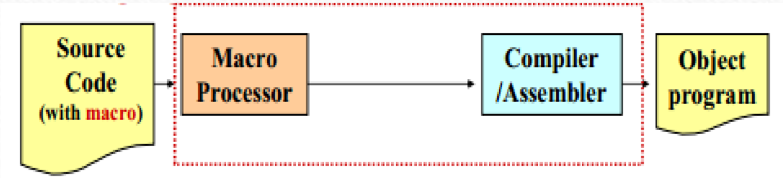
- Advantages
  - Programmers do not need to learn many macro languages.
  - Overall saving in software development cost and software maintenance effort
- Difficulties:
  - Large number of details must be dealt with in a real programming language
    - *Comment identifications (//, /* */, …)*
    - *Grouping together terms, expressions, statements (begin_end, { }, …)*
    - *Tokens (keywords, operators)*
    - *Syntax had better be consistent with the source programming language*

Macro Processing within Language Translators

- The macro processors we discussed are called “Preprocessors”.
  - Process macro definitions
  - Expand macro invocations
  - Produce an expanded version of the source program, which is then used as input to an assembler or compiler.



- You may also combine the macro processing functions with the language translator:
  - Line-by-line macro processor
  - Integrated macro processor



Line-by-Line Macro Processor

- Used as a sort of input routine for the assembler or compiler
  - Read source program
  - Process macro definitions and expand macro invocations
  - Pass output lines to the assembler or compiler
- Benefits
  - It avoids making an extra pass over the source program.
  - Data structures required by the macro processor and the language translator can be combined
    - E.g., OPTAB and NAMTAB)
  - Utility subroutines can be used by both macro processor and the language translator.
    - Scanning input lines ➔ Searching tables ➔ Data format conversion
  - It is easier to give diagnostic messages related to the source statements (i.e., the source statement error can be quickly identified without need to backtrack the source)

Integrated Macro Processor

- An integrated macro processor can potentially make use of any information about the source program that is extracted by the language translator.
- Benefits:
  - An integrated macro processor can support macro instructions that depend upon the context in which they occur.
  - Since the Macro Processor may recognize the meaning of source language

Drawbacks of Line-by-line/Integrated

- They must be specially designed and written
  - To work with a particular implementation of an assembler or compiler.
- The costs of macro processor development is added to the costs of the language translator
  - Which results in a more expensive software.
- The assembler or compiler will be considerably larger and more complex.

Chapter 4}

Chapter 5{