Macro Processors

Chapter 4

System Software

An introduction to systems programming

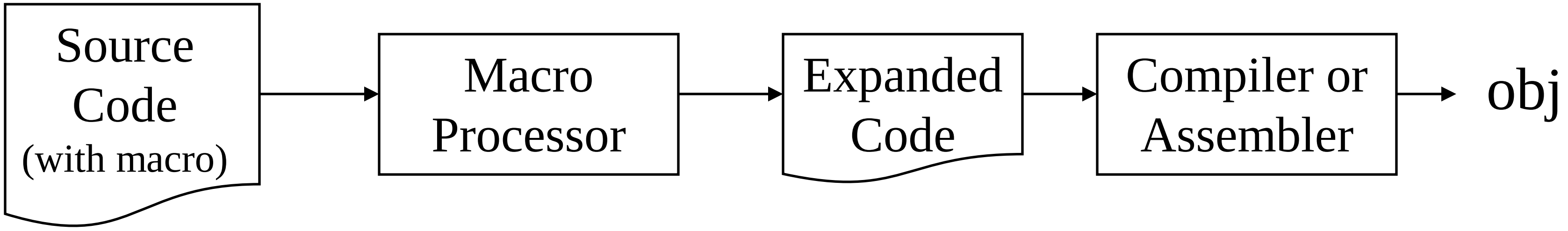
Leland L. Beck

Introduction

- Concept
  - A macro instruction is a notational convenience for the programmer
  - It allows the programmer to write shorthand version of a program (*module programming*)
  - The macro processor replaces each macro invocation with the corresponding sequence of statements (*expanding*)

Macro Processor

- Recognize macro definitions
- Save the macro definition
- Recognize macro calls
- Expand macro calls



Macro Definition

- copy code
- parameter substitution
- conditional macro expansion
- macro instruction defining macros

Copy code -- Example

Source

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

Expanded source

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

Macro vs. Subroutine

- Macro
  - the statement of expansion are generated each time the macro are invoked
- Subroutine
  - the statement in a subroutine appears only once

Parameter Substitution -- Example

Source

```
STRG    MACRO &a1, &a2, &a3
        STA     &a1
        STB     &a2
        STX     &a3
        MEND
   .
STRG    DATA1, DATA2, DATA3
   .
STRG    DATA4, DATA5, DATA6
   .
   .
```

Expanded souce

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

Parameter Substitution

- Dummy arguments
  - Positional argument

```
STRG    DATA1, DATA2, DATA3
GENER  ,,DIRECT,,,,,,3
```

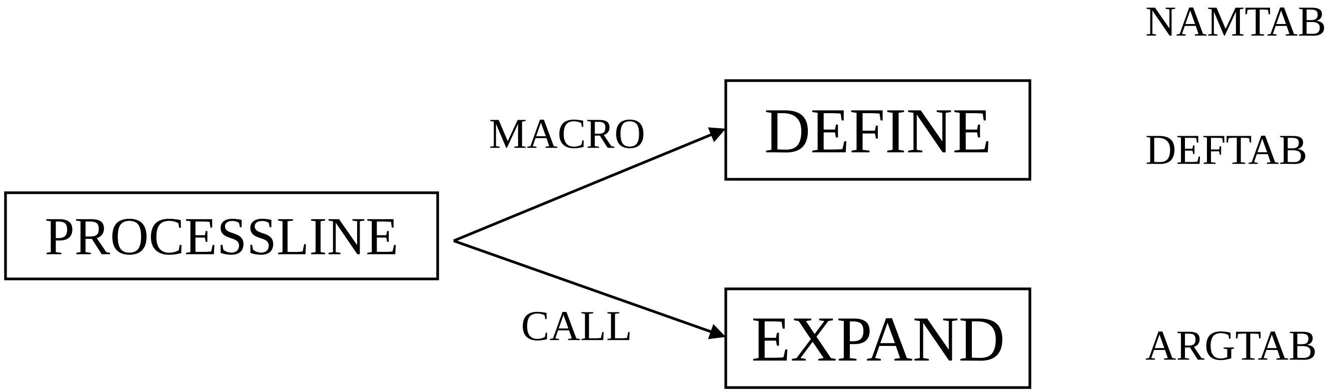
  - Keyword argument

```
STRG    &a3=DATA1, &a2=DATA2, &a1=DATA3
GENER  TYPE=DIRECT, CHANNEL=3
```

- Example: Fig. 4.1, Fig. 4.2
  - Labels are avoided in macro definition

One-Pass Macro Processor

- Prerequisite
  - every macro must be defined before it is called
- Sub-procedures
  - macro definition: DEFINE
  - macro invocation: EXPAND



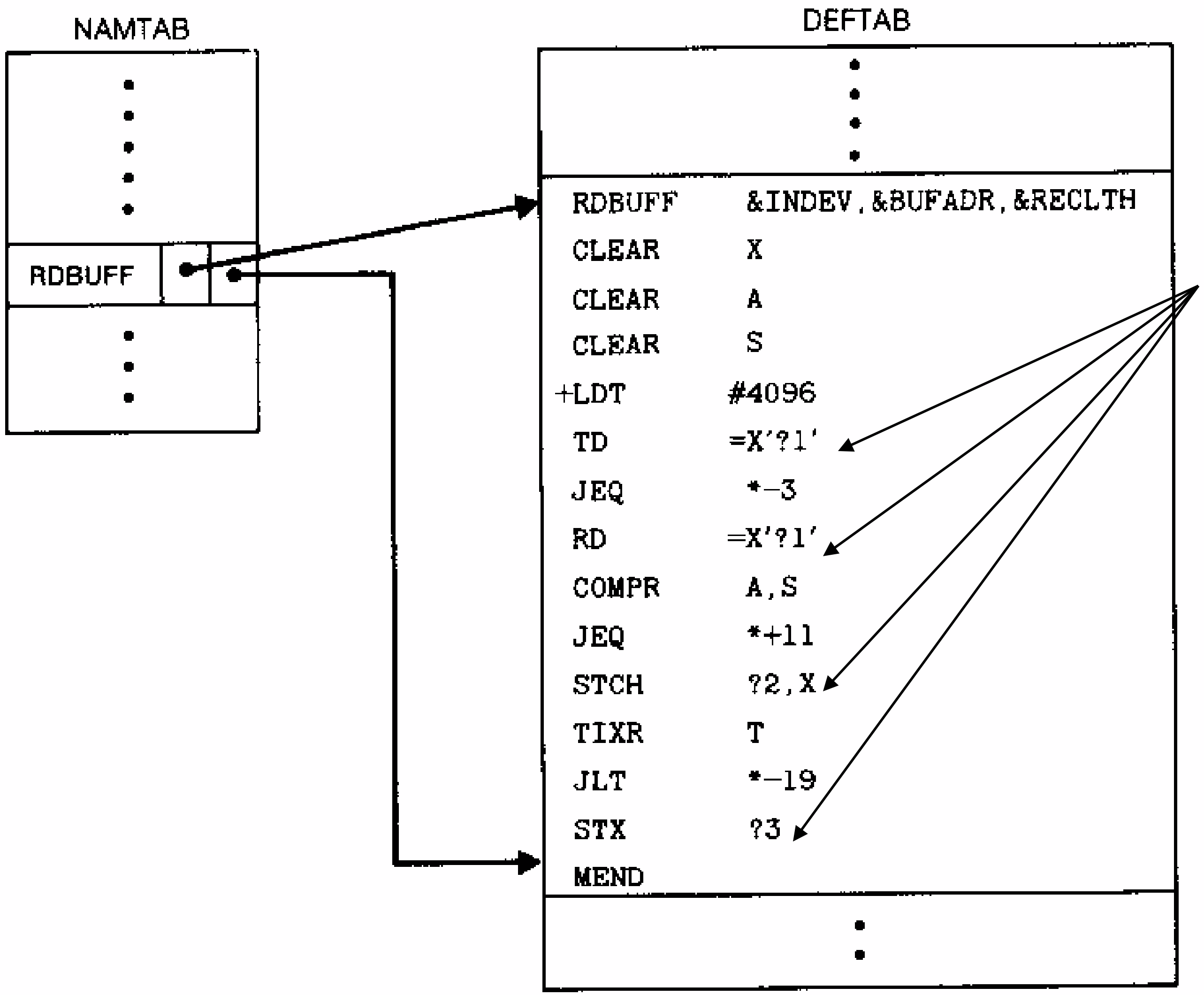
```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE  ≠ 'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
end {macro processor}


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.

Data Structures -- Global Variables

- DEFTAB
- NAMTAB
- ARGTAB

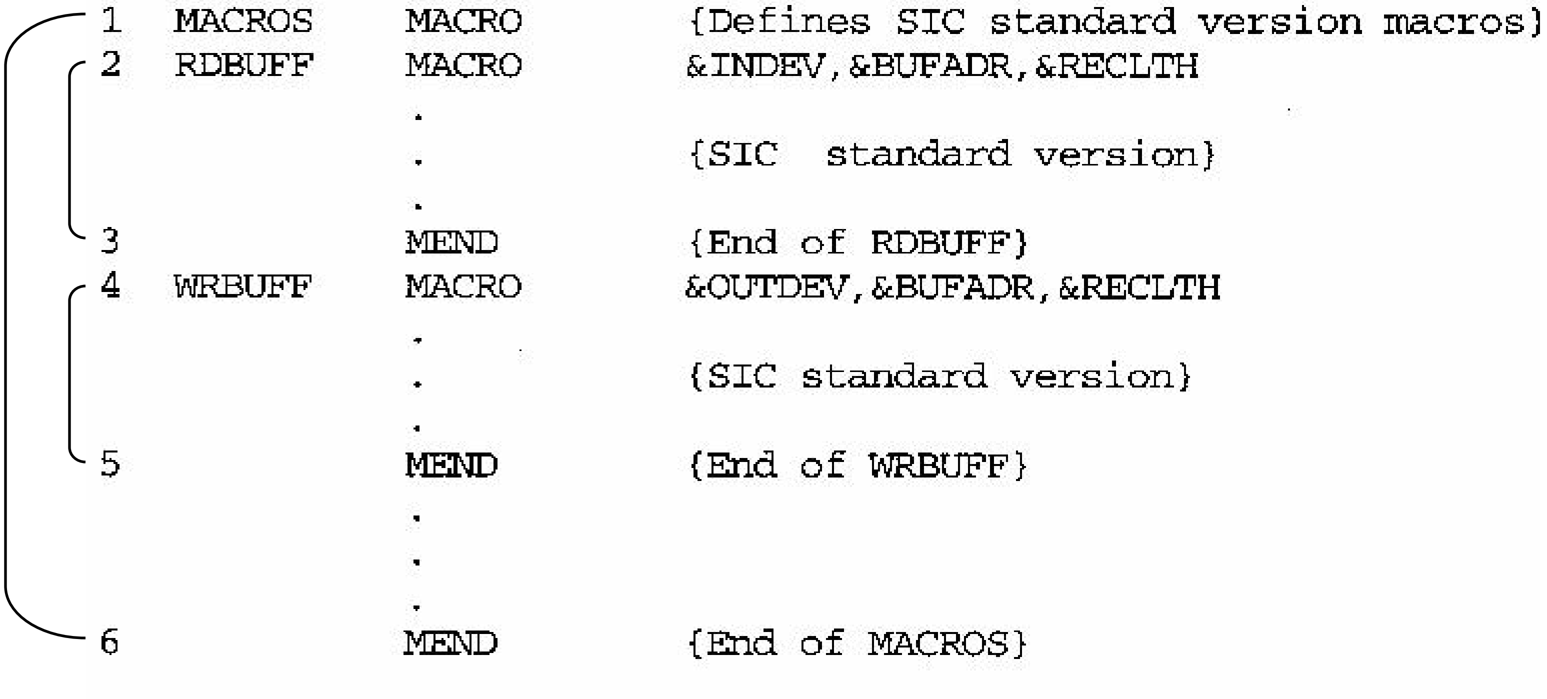


```
 RDBUFF     &INDEV,&BUFADR,&RECLTH
 CLEAR      X
 CLEAR      A
 CLEAR      S
+LDT       #4096
 TD        =X'?1'
 JEQ        *-3
 RD        =X'?1'
 COMPR      A,S
 JEQ        *+11
 STCH       ?2,X
 TIXR       T
 JLT        *-19
 STX        ?3
 MEND
```

EXPANDING

Nested Macros Definition

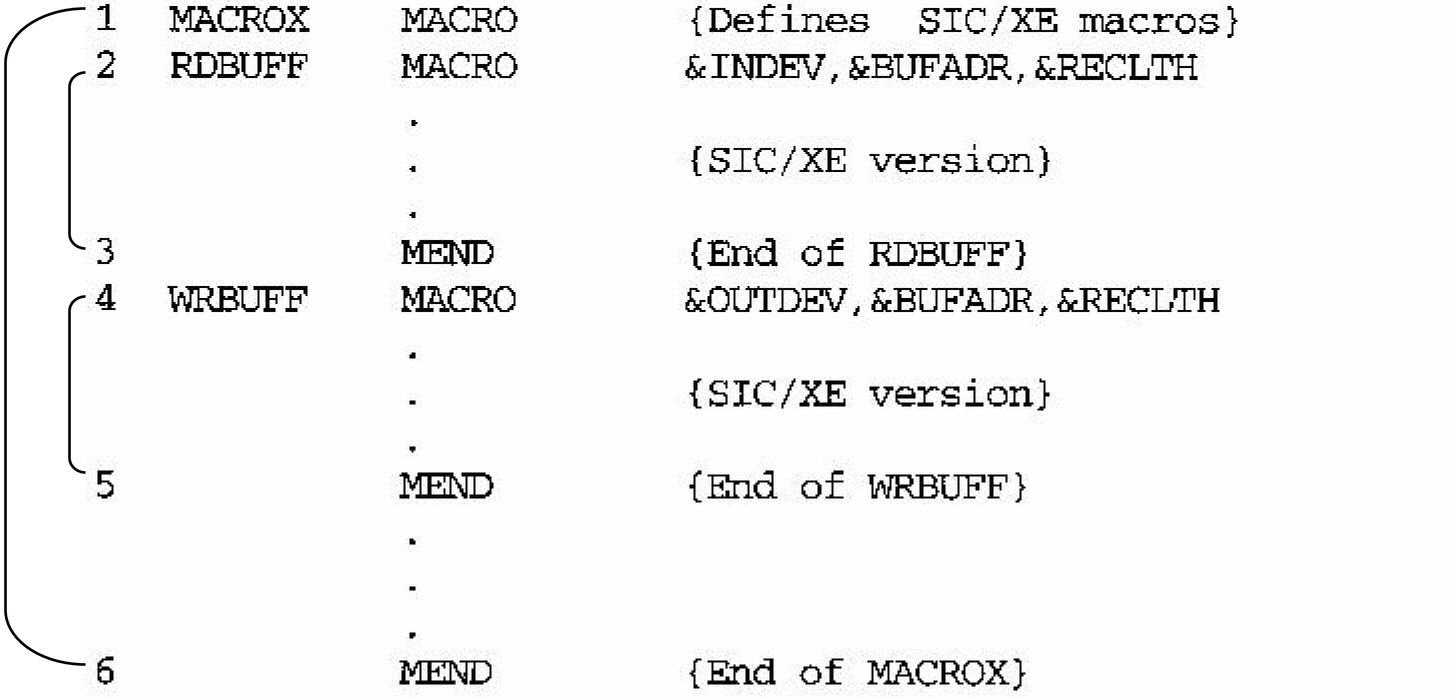
- Macro definition within macros
  - process macro definition during expansion time
- Example 4.3



```
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}
```

(a)

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}
```

One-Pass Macro Processor That Allows Nested Macro Definition

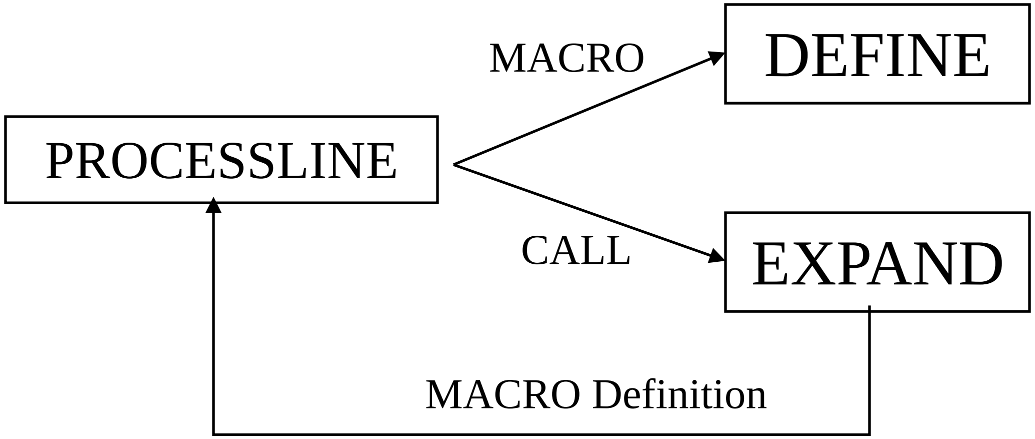
- Sub-procedures
  - macro definition: DEFINE
  - macro invocation: EXPAND
- EXPAND may invoke DEFINE when encounter macro definition

NAMTAB

DEFTAB

ARGTAB

Expanding



MACRO

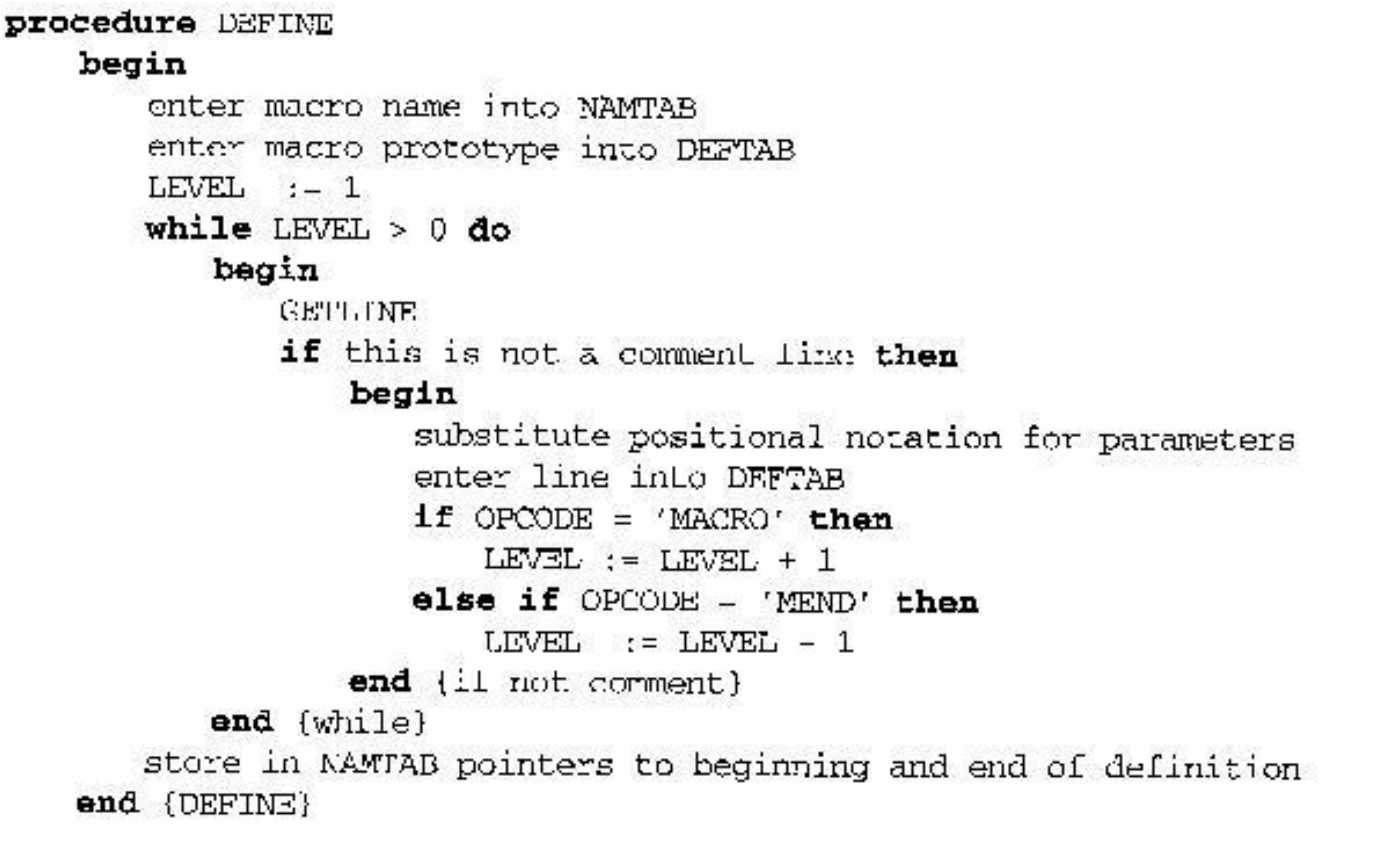
DEFINE

PROCESSLINE

CALL

EXPAND

MACRO Definition



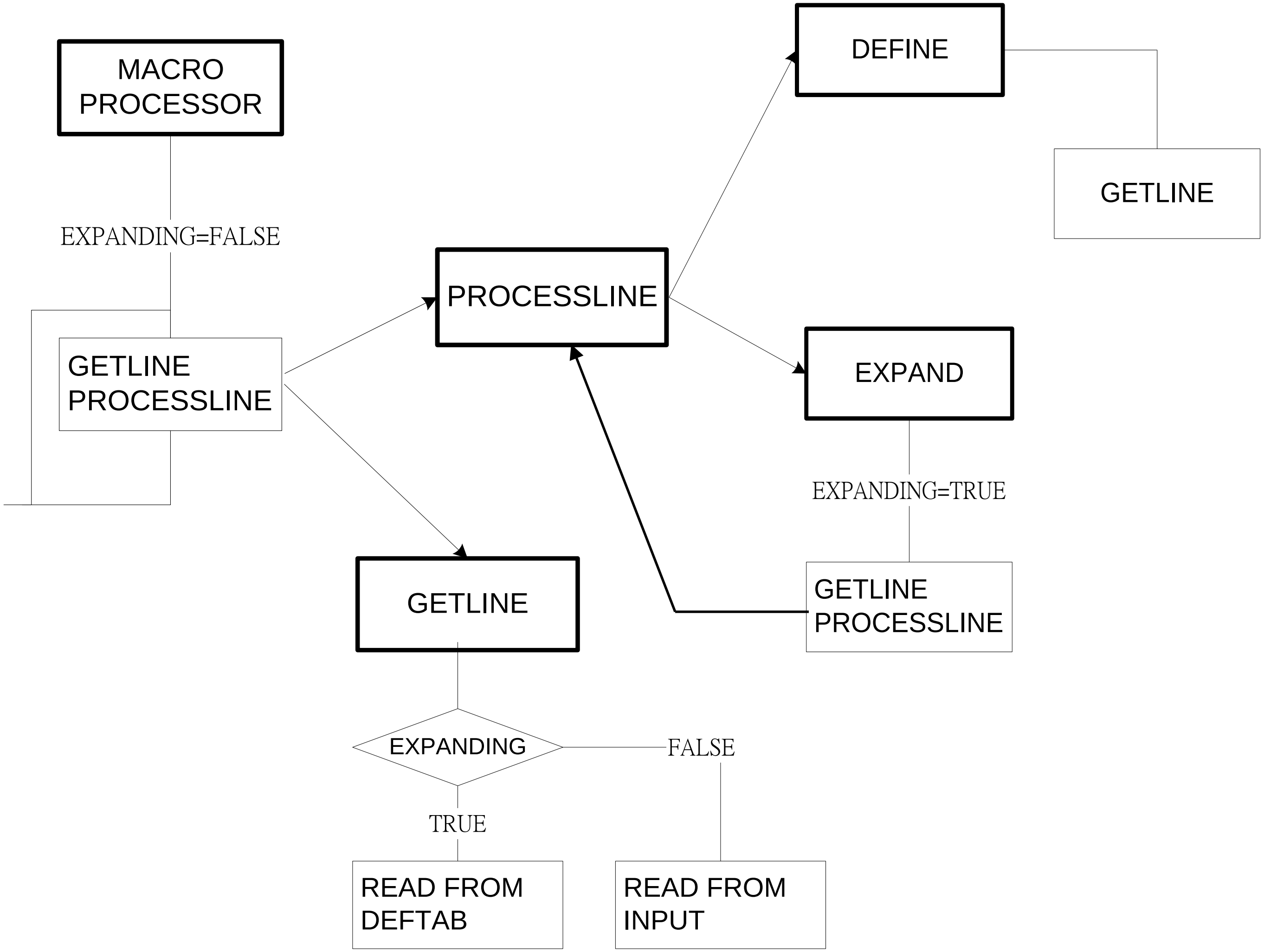
```
procedure DEFINE
    begin
        enter macro name into NAMTAB
        enter macro prototype into DEFTAB
        LEVEL  := 1
        while LEVEL > 0 do
            begin
                GETLINE
                if this is not a comment line then
                    begin
                        substitute 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)*

1-Pass Macro Processor

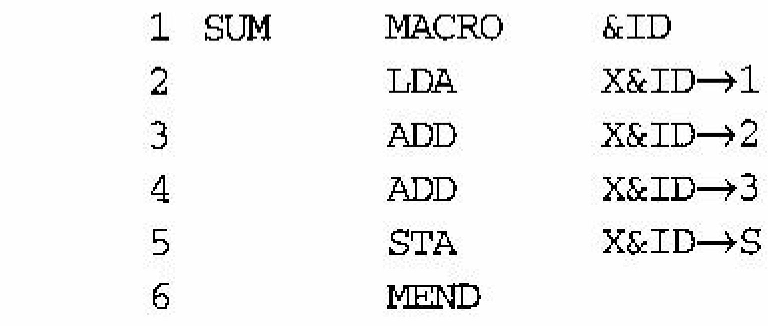


Comparison of Macro Processors Design

- Single pass
  - every macro must be defined before it is called
  - one-pass processor can alternate between macro definition and macro expansion
  - nested macro definitions may be allowed but nested calls are not
- Two pass algorithm
  - Pass1: Recognize macro definitions
  - Pass2: Recognize macro calls
  - nested macro definitions are not allowed

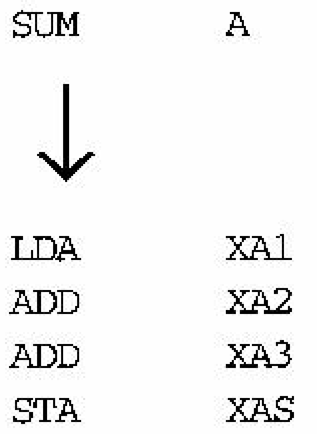
Concatenation of Macro Parameters

- Pre-concatenation
  - » LDA X&ID1
- Post-concatenation
  - » LDA X&ID→1
- Example: Figure 4.6



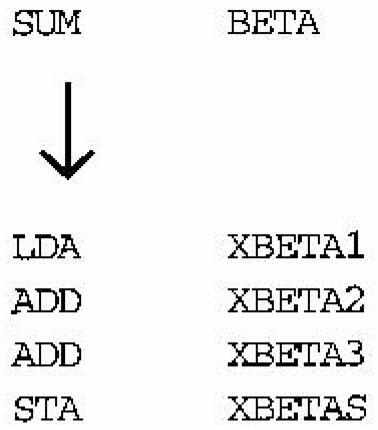
```
1  SUM    MACRO   &ID
2         LDA     X&ID→1
3         ADD     X&ID→2
4         ADD     X&ID→3
5         STA     X&ID→S
6         MEND
```

(a)



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

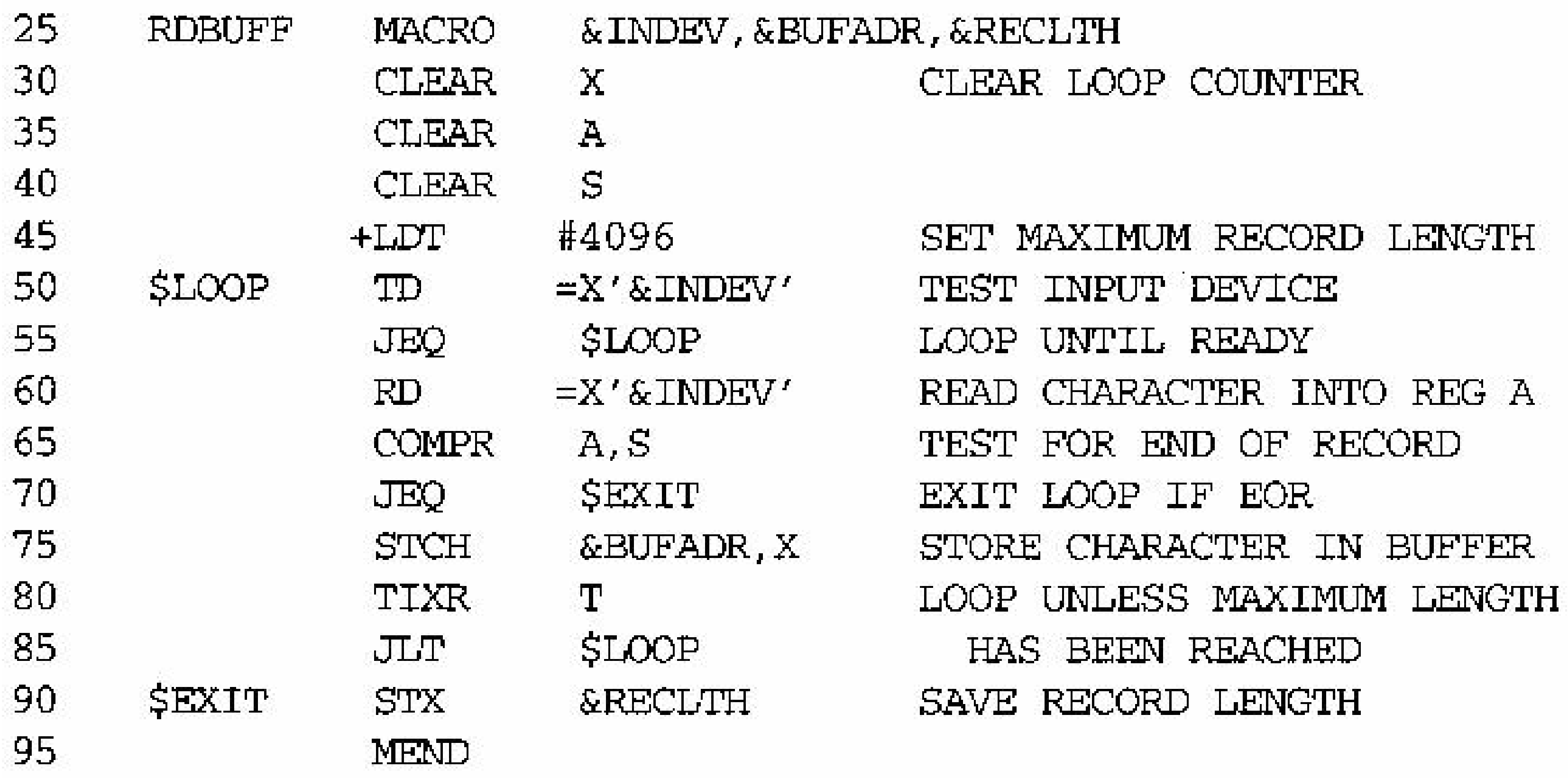
(b)



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

Generation of Unique Labels

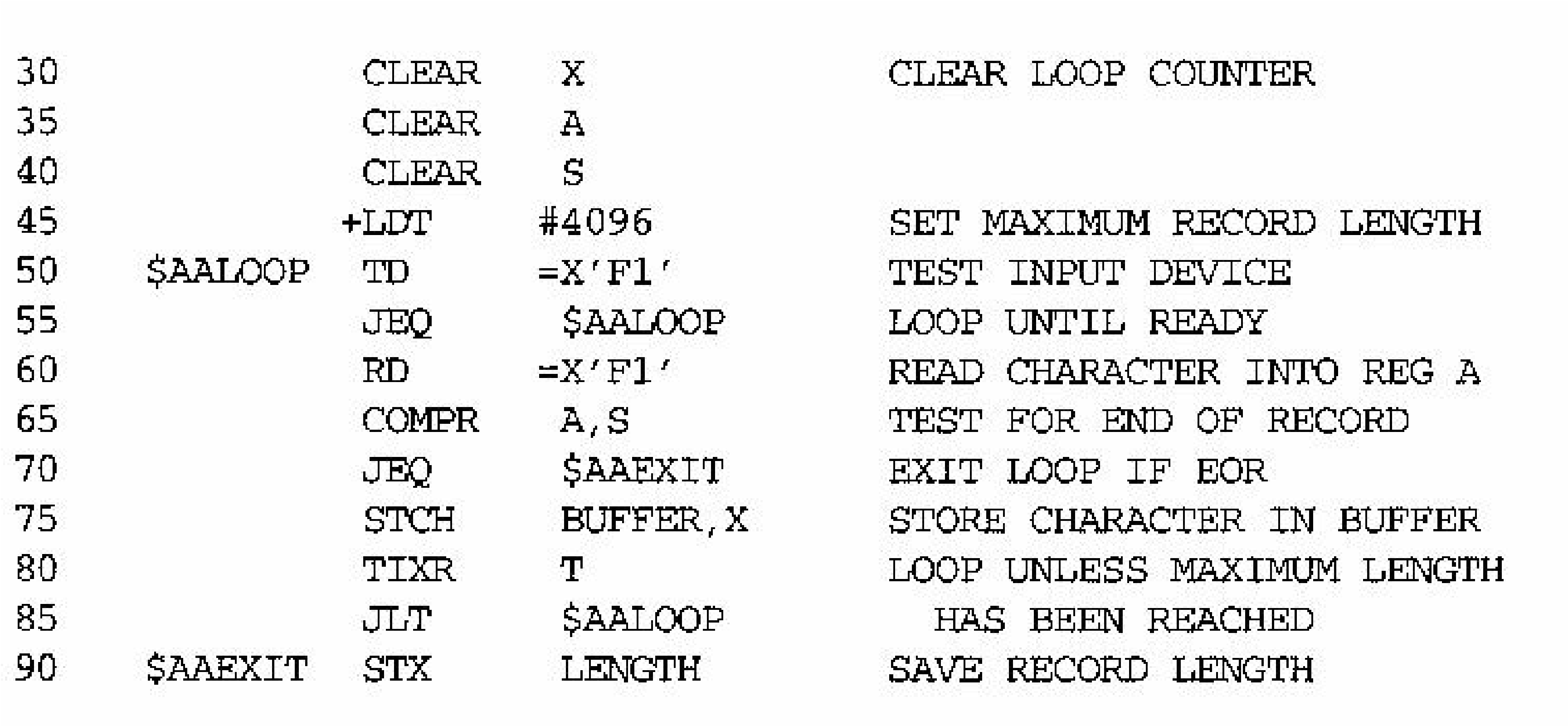
- Example
  - JEQ *-3
  - inconvenient, error-prone, difficult to read
- Example Figure 4.7
  - – $LOOP TD =X’&INDEV’
  - 1st call:
    - – $AALOOP TD =X’F1’
  - 2nd call:
    - – $ABLOOP TD =X’F1’



```
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



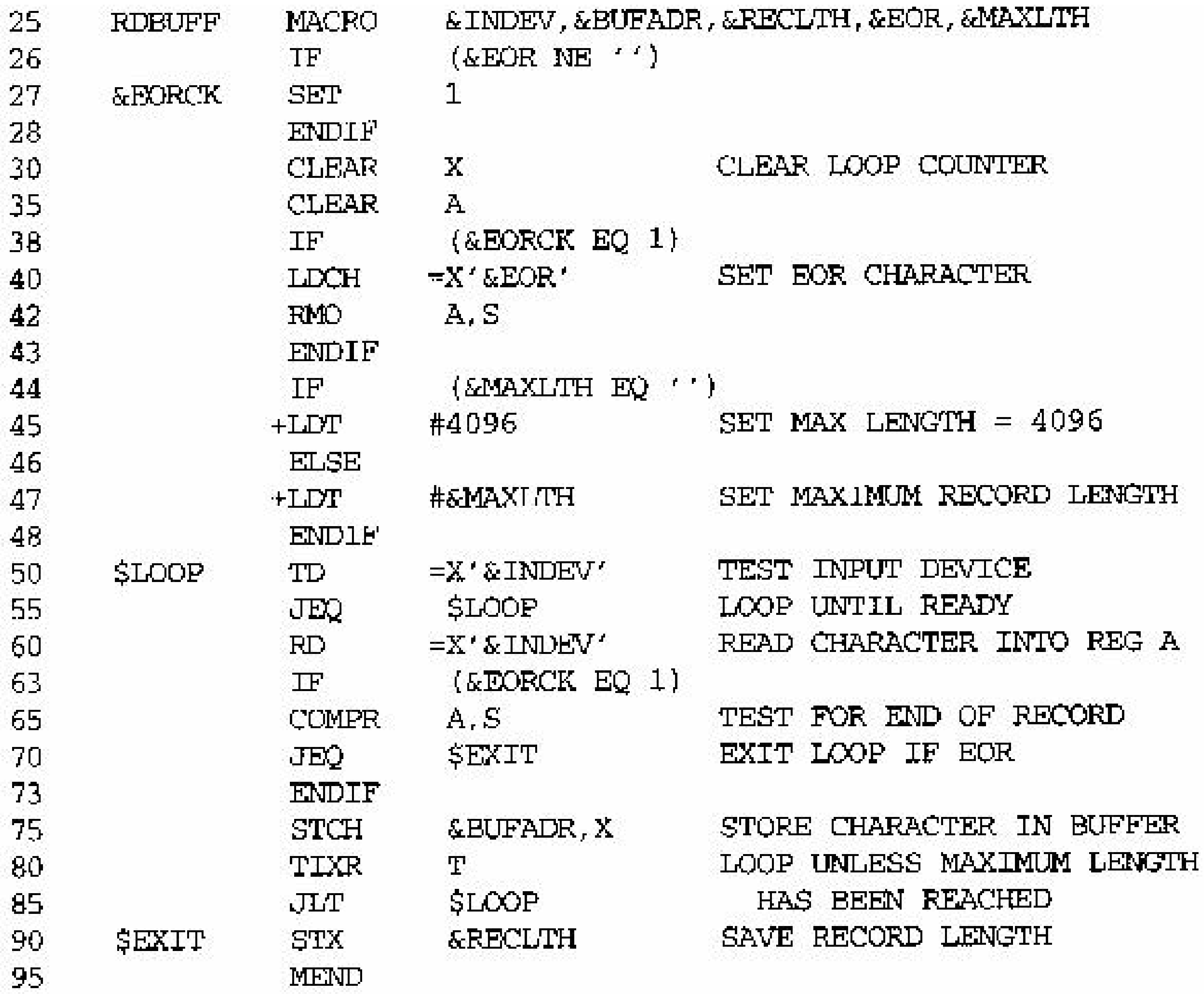
```
30               CLEAR    X              CLEAR LOOP COUNTER
35               CLEAR    A
40               CLEAR    S
45              +LDT     #4096           SET MAXIMUM RECORD LENGTH
50    $AALOOP    TD      =X'F1'          TEST INPUT DEVICE
55               JEQ      $AALOOP        LOOP UNTIL READY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

(b)

Conditional Macro Expansion

- Macro-time conditional statements
  - Example: Figure 4.8
  - *IF-ELSE-ENDIF*

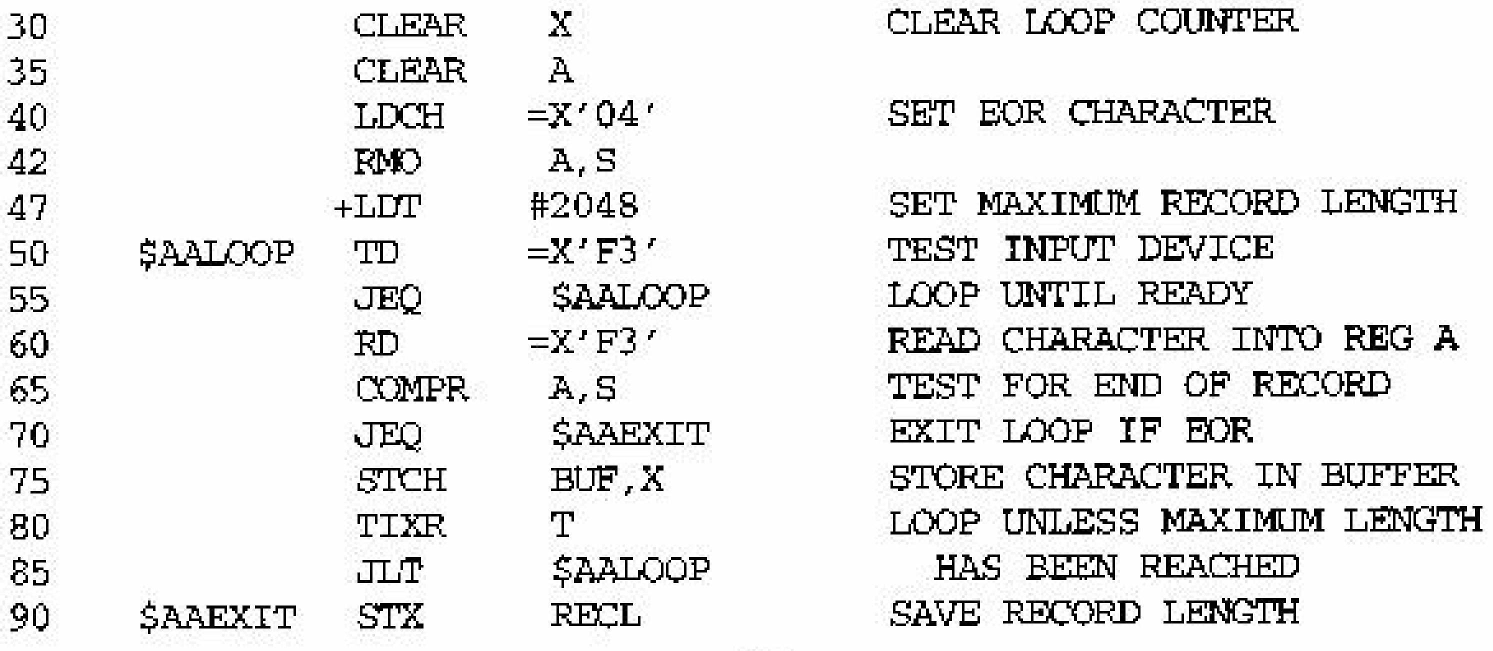
- Macro-time variables
  - any symbol that begins with the character & and that is not a macro parameter
  - macro-time variables are initialized to 0
  - macro-time variables can be changed with their values using SET
    - *&EORCK* SET 1



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

(a)

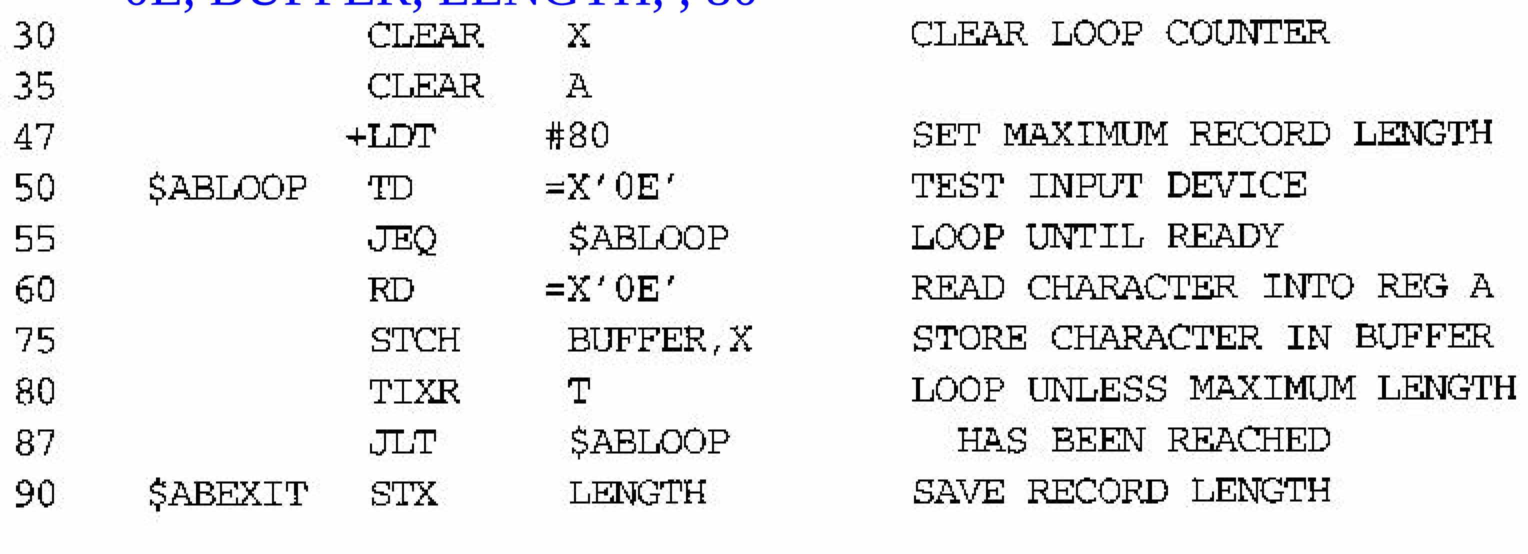
RDBUFF F3. BUF. RECL. 04. 2048



```
30              CLEAR    X             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)

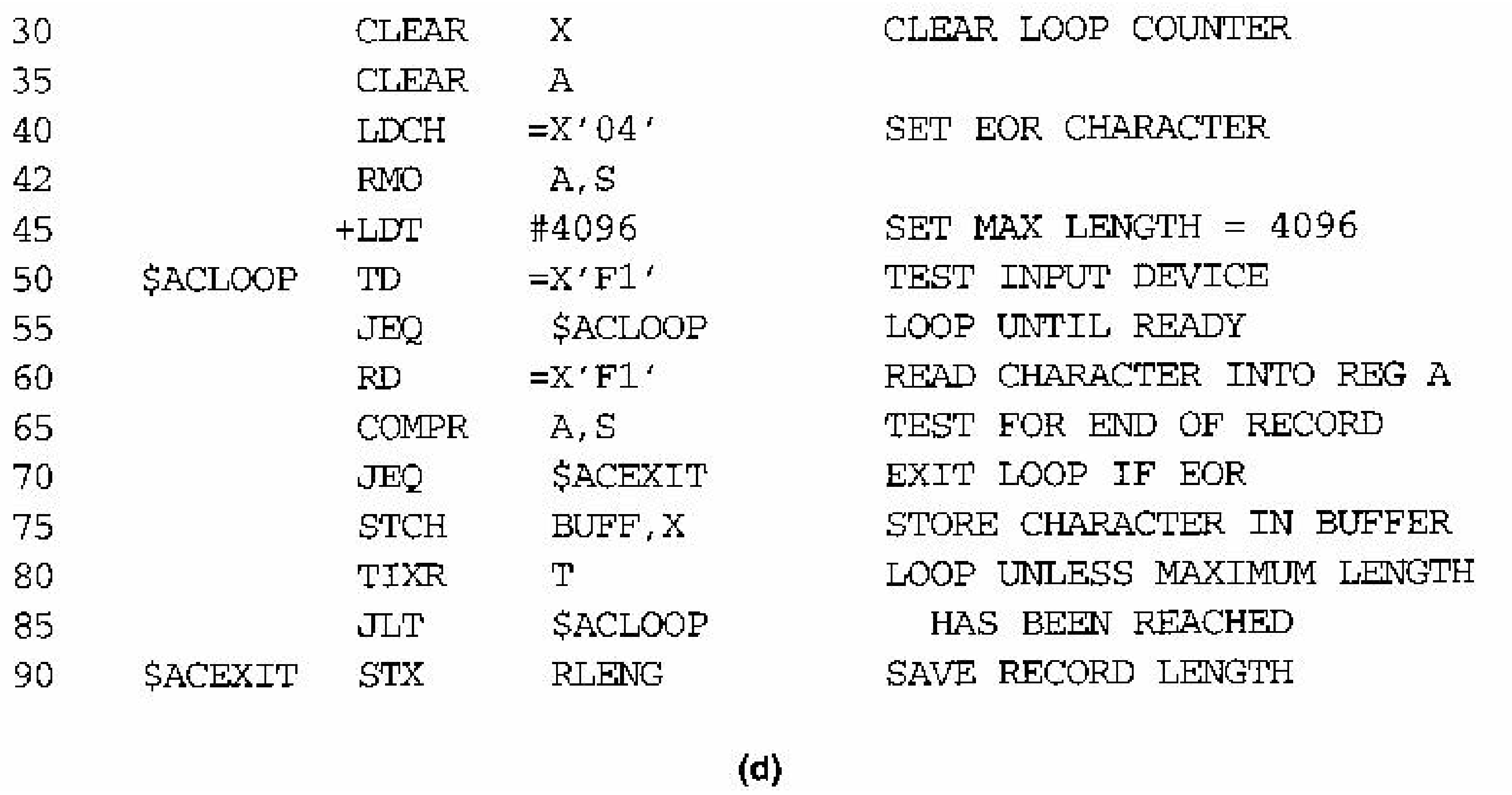
RDBUFF 0E, BUFFER, LENGTH, , 80



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

(c)

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



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

(d)

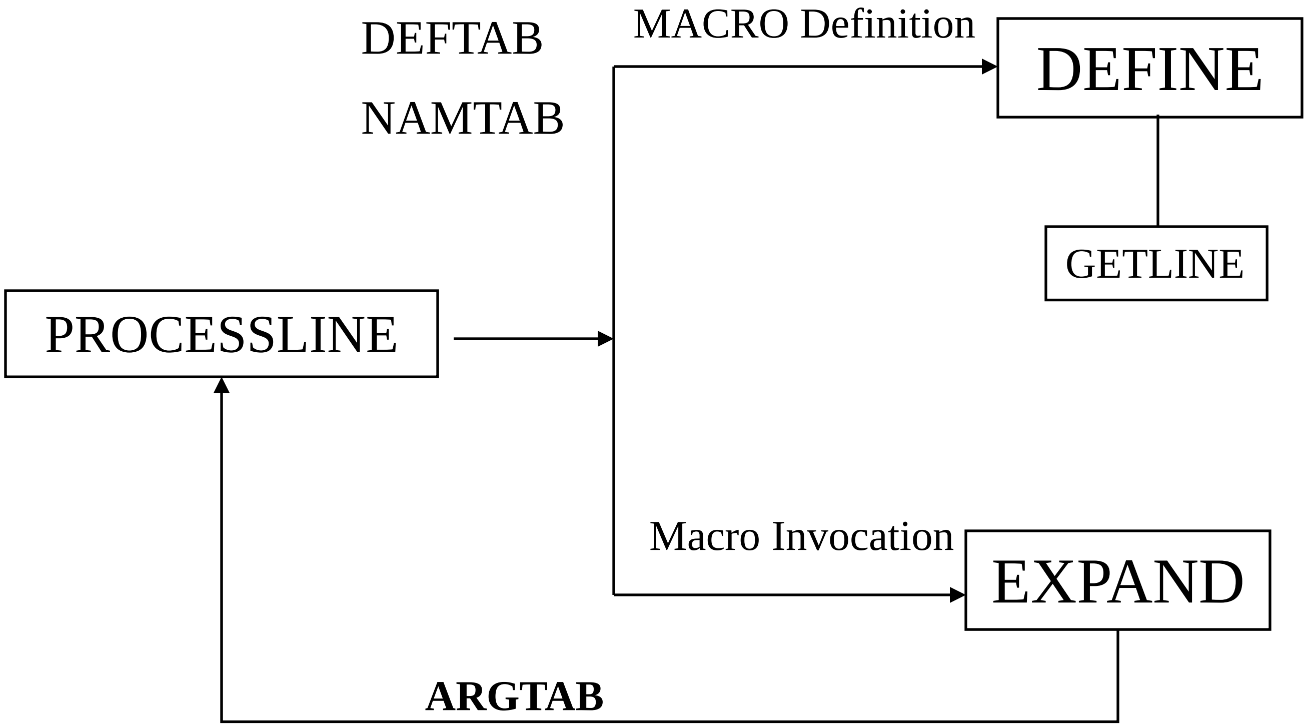
Conditional Macro Expansion (Cont.)

- Macro-time looping statement
  - Example: Figure 4.9
  - WHILE-ENDW
- Macro processor function
  - %NITEMS: THE NUMBER OF MEMBERS IN AN ARGUMENT LIST

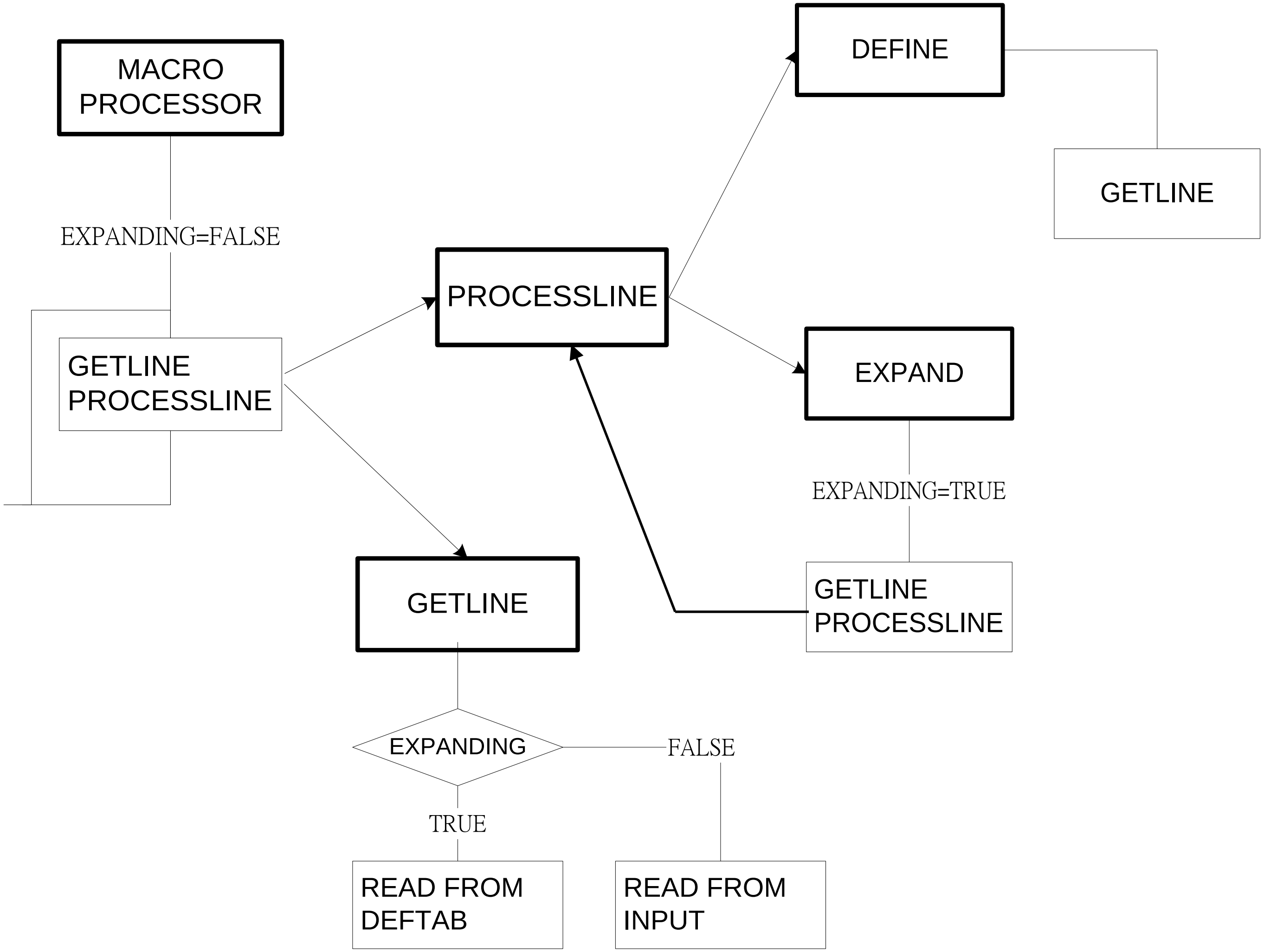
Nested Macro Invocations

- Macro invocations within macros
  - » process macro invocation during expansion time
- Recursive macro expansion
  - » Example: Figure 4.11
  - » Problems:
    - – ARGTAB
    - – EXPANDING
  - » Solution
    - – Recursive call
    - – While loop with stack

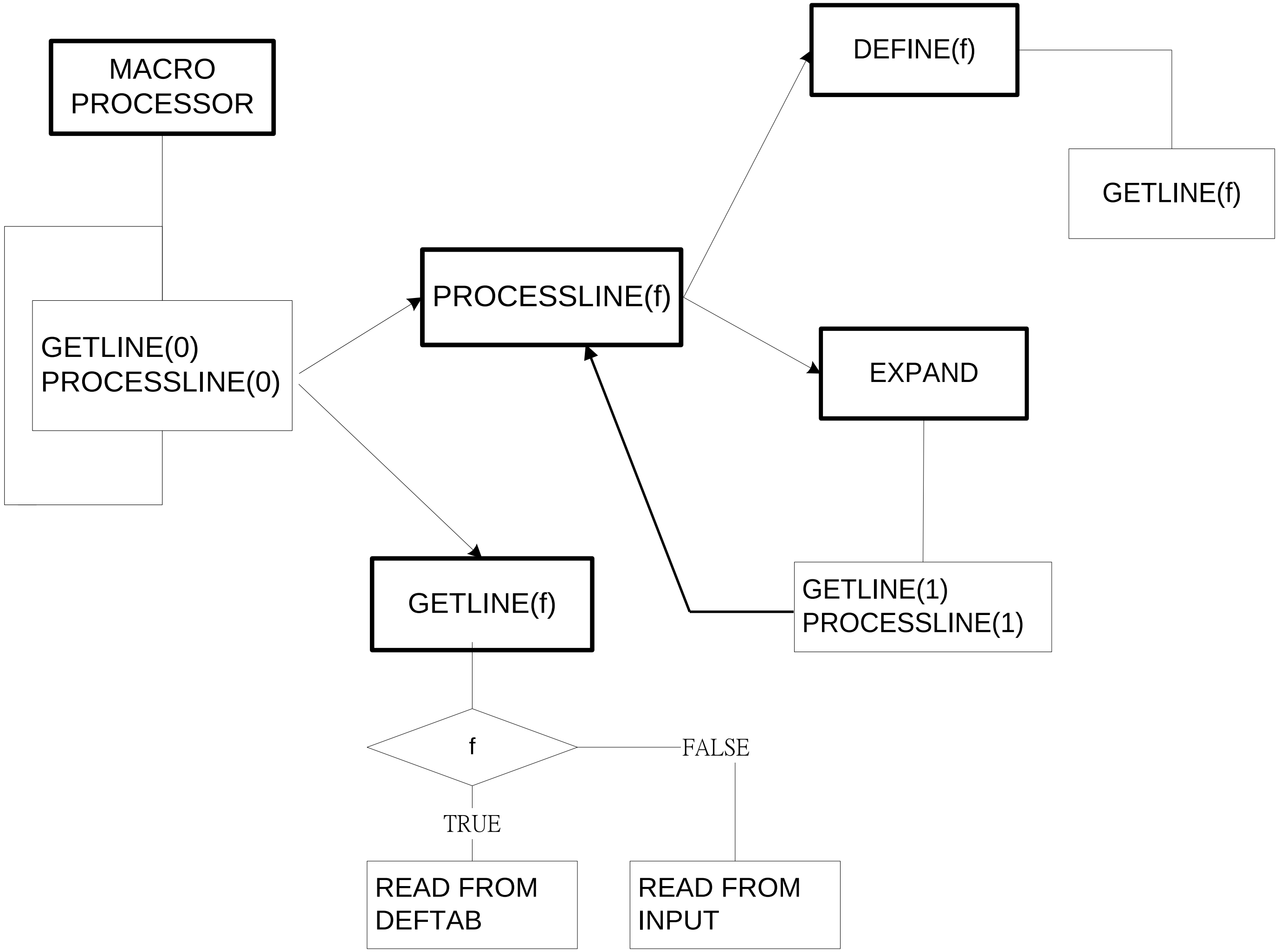
ARGTAB


DEFTAB
NAMTAB
MACRO Definition
DEFINE
GETLINE
PROCESSLINE
Macro Invocation
EXPAND
ARGTAB

1-Pass Macro Processor

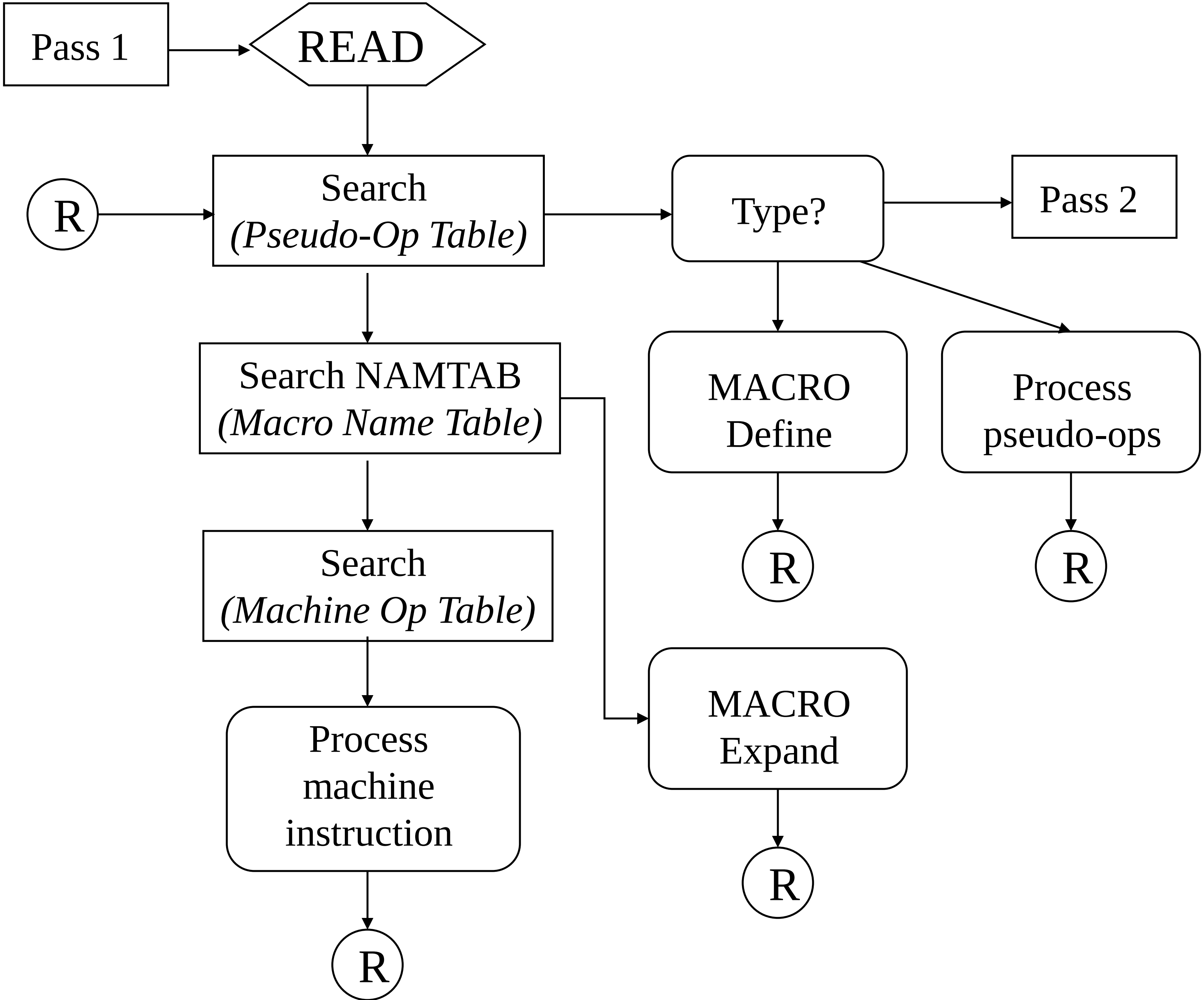


Allowing Nested Macro Invocation



Macro-Assembler

- Advantage
  - reduce 1 pass
  - share same data structure
- Disadvantage
  - more complex



Pass 1

READ

R

Search
(Pseudo-Op Table)

Type?

Pass 2

Search NAMTAB
(Macro Name Table)

MACRO
Define

Process
pseudo-ops

R

R

Search
(Machine Op Table)

MACRO
Expand

Process
machine
instruction

R

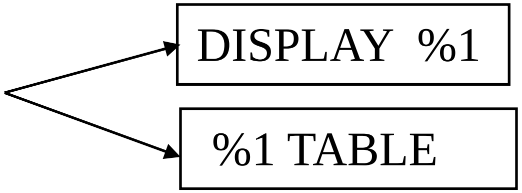
R

General Purpose Macro Processor

- ELENA
  - Software: Practice and Experience, Vol. 14, pp. 519-531, Jun. 1984
- Macro definition
  - header:
    - a sequence of keywords and parameter markers (%)
    - at least one of the first two tokens in a macro header must be a keyword, not a parameter marker
  - body:
    - the character & identifies a local label
    - macro time instruction (.SET, .IF .JUMP, .E)
    - macro time variables or labels (.)

ELENA (cont.)

- Macro invocation
  - There is no single token that constitutes the macro “name”
  - Constructing an index of all macro headers according to the keywords in the first two tokens of the header
  - Example
    - DEFINITION:
      - ADD %1 TO %2
      - ADD %1 TO THE FIRST ELEMENT OF %2
    - INVOCATION:
      - DISPLAY TABLE



DISPLAY %1

%1 TABLE