Macro Processors

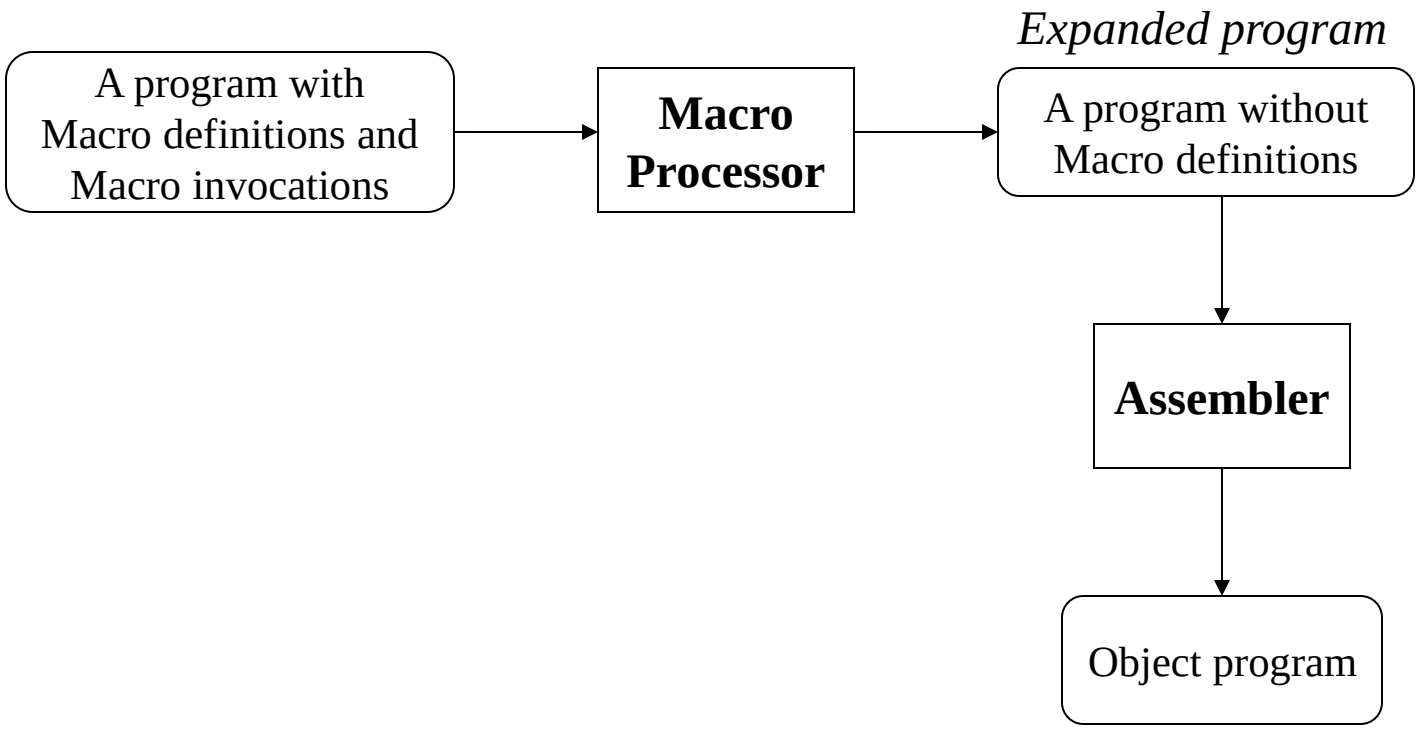
Introduction

- A macro instruction (abbreviated to *macro*) is simply a notational convenience for the programmer.
- A macro represents a commonly used group of statements in the source programming language
- Expanding a macros
  - Replace each macro instruction with the corresponding group of source language statements

Introduction (Cont'd)

- E.g.
  - On SIC/XE requires a sequence of seven instructions to save the contents of all registers
    - Write one statement like SAVERGS
- A macro processor is not directly related to the architecture of the computer on which it is to run
- Macro processors can also be used with high-level programming languages, OS command languages, etc.

Basic Macro Processor Functions



Basic Macro Processor Functions

- Macro Definition
  - Two new assembler directives
    - MACRO
    - MEND
  - A pattern or prototype for the macro instruction
    - Macro name and parameters

Line	Source statement			
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		TD	=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		

```
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         .
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.

Basic Macro Processor Functions

- Macro invocation
  - Often referred to as a *macro call*
    - Need the name of the macro instruction begin invoked and the arguments to be used in expanding the macro
- Expanded program
  - Figure 4.2
  - No macro instruction definitions
  - Each macro invocation statement has been expanded into the statements that form the body of the macro, with the arguments from the macro invocation substituted for the parameters in the prototype

Line	Source statement			
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
1901		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.

Basic Macro Processor Functions

- Macro invocations and subroutine calls are different

- Note also that the macro instructions have been written so that the body of the macro contains no label

Macro Processor Algorithm and Data Structures

- It is easy to design a two-pass macro processor
  - Pass 1:
    - All macro definitions are processed
  - Pass 2:
    - All macro invocation statements are expanded
- However, this macro processor would not allow the body of one macro instruction to contain definitions of other macros

```
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)

```
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.

Macro Processor Algorithm and Data Structures

- Sub-Macro definitions are only processed when an invocation of their Super-Macros are expanded
  - See Figure 4.3: RDBUFF
- A one-pass macro processor that can alternate between macro definition and macro expansions able to handle macros like those in Figure 4.3

Macro Processor Algorithm and Data Structures

- Because of the one-pass structure, the definition of a macro must appear in the source program before any statements that invoke that macro
- Three main data structures involved in an one-pass macro processor
  - DEFTAB, NAMTAB, ARGTAB

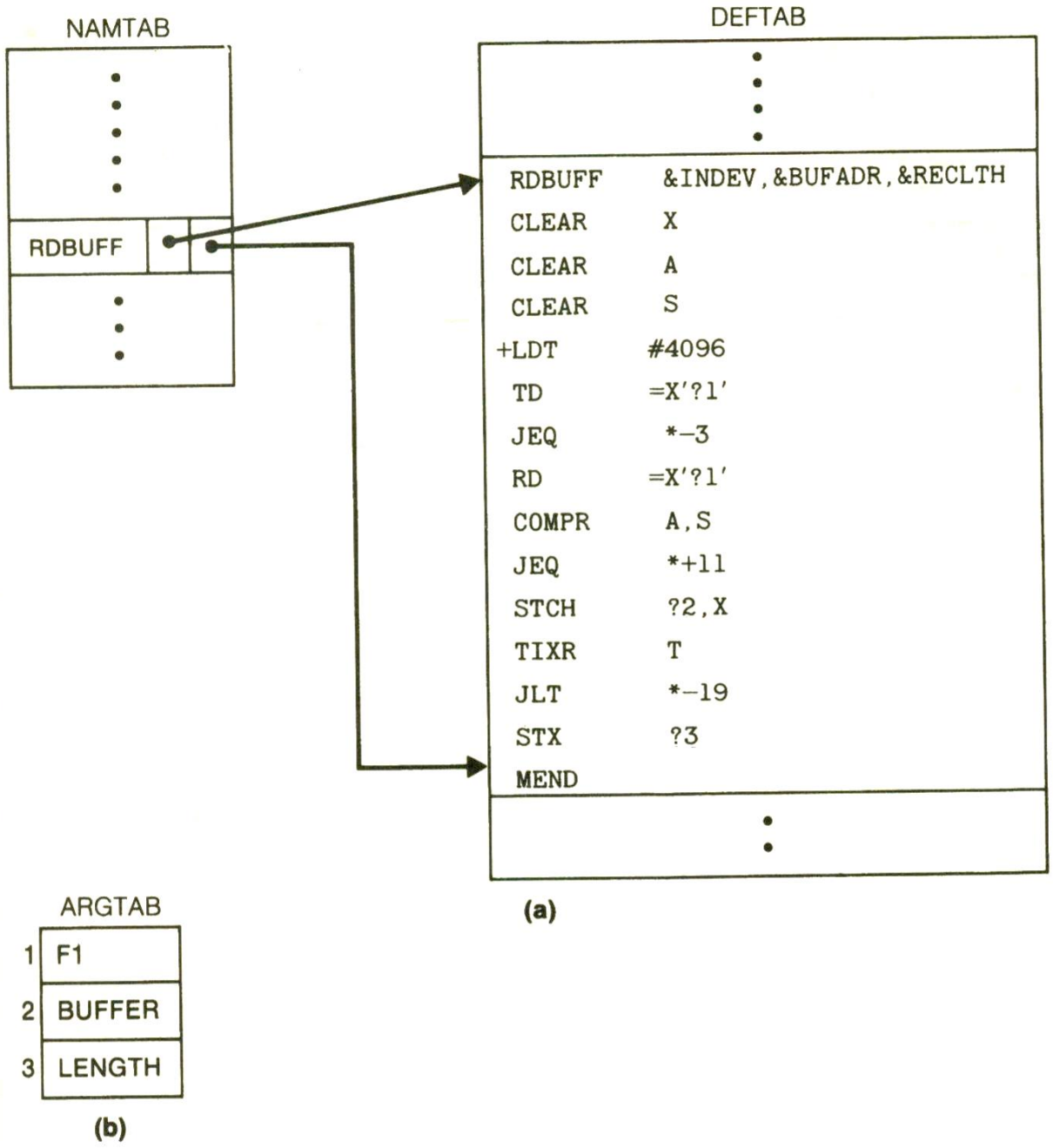


Figure 4.4 Contents of macro processor tables for the program in Fig. 4.1: (a) entries in NAMTAB and DEFTAB defining macro RDBUFF.

```
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.

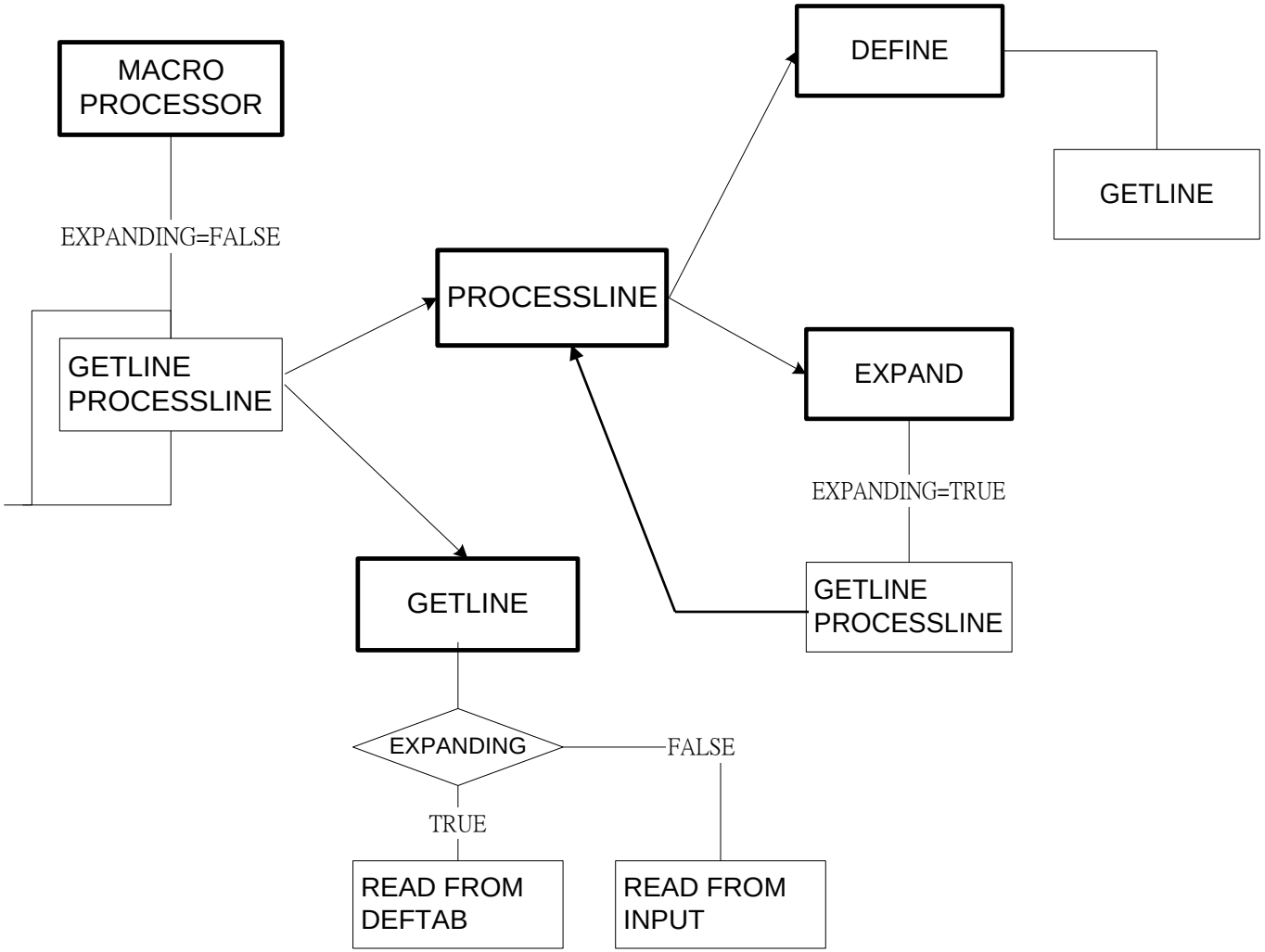
```
procedure DEFINE
    begin
        enter macro name into NAMTAB
        enter macro prototype into DEFTAB
        LEVEL  := 1
        while LEVEL > 0 do
            begin
                GETLINE
                if this is not a comment line then
                    begin
                        substitute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

1-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*)

Machine-Independent Macro Processor Feature

- Concatenation of Macro Parameters
- Generation of Unique Labels
- Conditional Macro Expansion
- Keyword Macro Parameters

Concatenation of Macro Parameters

- Most macro processors allow parameters to be concatenated with other character strings
  - The need of a special catenation operator
    - LDA X&ID1
    - LDA X&ID
  - The catenation operator
    - LDA X&ID→1
- See 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
```

(c)

Figure 4.6 Concatenation of macro parameters.

Generation of Unique Labels

- 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
  - See Figure 4.7

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

(a)

```
.          RDBUFF   F1,BUFFER,LENGTH

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

(b)

Figure 4.7 Generation of unique labels within macro expansion.

Generation of Unique Labels

- Solution:
  - Allowing the creation of special types of labels within macro instructions
  - See Figure 4.7
    - Labels used within he macro body begin with the special character $
  - Programmers are instructed no to use $ in their source programs

Conditional Macro Expansion

- Most macro processors can
  - Modify the sequence of statements generated for a macro expansion,
  - depending on the arguments supplied in the macro invocation
- See Figure 4.8

```
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          +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)

Figure 4.8 (*cont'd*)

Conditional Macro Expansion

- Most macro processors can modify the sequence of statements generated for a macro expansion, depending on the arguments supplied in the macro invocation
- See Figure 4.8
  - Macro processor directive
    - IF, ELSE, ENDIF
    - SET
  - Macro-time variable (set symbol)
- WHILE-ENDW
  - See Figure 4.9

```
25   RDBUFF    MACRO    &INDEV,&BUFADR,&RECLTH,&EOR
27   &EORCT    SET      %NITEMS(&EOR)
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             COMP     =X'0000&EOR[&CTR]'
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 RECORD LENGTH
100            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)

Figure 4.9 Use of macro-time looping statements.

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 statement
  - Consecutive commas is necessary for a null argument

    GENER „DIRECT,,,,,,3

Keyword Macro Parameters

- Keyword parameters
  - Each argument value is written with a keyword that names the corresponding parameter
  - A macro may have a large number of parameters , and only a few of these are given values in a typical invocation

    GENER TYPE=DIRECT, CHANNEL=3

```
25    RDBUFF    MACRO    &INDEV=F1,&BUFADR=,&RECLTH=,&EOR=04,&MAXLTH=4096
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
47             +LDT      #&MAXLTH         SET MAXIMUM RECORD LENGTH
50    $LOOP     TD       =X'&INDEV'       TEST INPUT DEVICE
55              JEQ      $LOOP            LOOP UNTIL READY
60              RD       =X'&INDEV'       READ CHARACTER INTO REG A
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   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*)

Macro Processor Design Options

- Recursive Macro Expansion
  - In Figure 4.3, we presented an example of the definition of on macro instruction by another.
    - We have not dealt with the invocation of one macro by another **(nested macro invocation)**
  - See Figure 4.11

```
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
```

(a)

```
 5    RDCHAR    MACRO    &IN
10    .
15    .         MACRO TO READ CHARACTER INTO REGISTER A
20    .
25              TD       =X'&IN'        TEST INPUT DEVICE
30              JEQ       *-3           LOOP UNTIL READY
35              RD       =X'&IN'        READ CHARACTER
40              MEND
```

(b)

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

(c)

Figure 4.11 Example of nested macro invocation.

Macro Processor Design Options

- Recursive Macro Expansion Applying Algorithm of Fig. 4.5
  - Problem:
    - The processing would proceed normally until line 50, which contains a statement invoking RDCHAR
    - In addition, the argument from the original macro invocation (RDBUFF) would be lost because the values in ARGTAB were overwritten with the arguments from the invocation of RDCHAR
  - Solution:
    - These problems are not difficult to solve if the macro processor is being written in a programming language that allows recursive call

General-Purpose Macro Processors

- Macro processors have been developed for some high-level programming languages
- These special-purpose macro processors are similar in general function and approach; however, the details differ from language to language

General-Purpose Macro Processors

- The advantages of such a general-purpose approach to macro processing are obvious
  - The programmer does not need to learn about a different macro facility for each compiler or assembler language, so much of the time and expense involved in training are eliminated
  - A substantial overall saving in software development cost

General-Purpose Macro Processors

- In spite of the advantages noted, there are still relatively few general-purpose macro processors.
  1. In a typical programming language, there are several situations in which normal macro parameter substitution should not occur
     - E.g. comments should usually be ignored by a macro processor

General-Purpose Macro Processors

2. Another difference between programming languages is related to their facilities for grouping together terms, expressions, or statements
   - E.g. Some languages use keywords such as begin and end for grouping statements. Others use special characters such as { and }.

General-Purpose Macro Processors

3. A more general problem involves the tokens of the programming language
   - E.g. identifiers, constants, operators, and keywords
   - E.g. blanks

General-Purpose Macro Processors

4. Another potential problem with general-purpose macro processors involves the syntax used for macro definitions and macro invocation statements. With most special-purpose macro processors, macro invocations are very similar in form to statements in the source programming language

All these Macros are called **Hygienic macro**.

- Removes problem of "Accidental Capture of Identifiers" (LISP)

The end.