

Chapter 4
Macro Processors



Outline

- Basic Macro Processor Functions
- Machine-Independent Macro Processor Features
- Macro Processor Design Options
- Implementation Examples



Basic Macro Processor Functions

Macro and Macro Processor

- 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.
- The *macro processors* do ***macro expansion*** by replacing each macro instruction with the corresponding group of source language statements.
  - Macro instructions allow the programmer to write a shorthand version of a program, and leave the mechanical details to be handled by the macro processor.
  - E.g., Use a macro SAVEREGS to save the contents of all registers on SIC/XE machine, instead of a sequence of seven instructions (STA, STB, etc.).

Macro Processor

- The functions of a macro processor involves ***the substitution of one group of characters or lines for another***.
  - The design and capabilities of a macro processor may be influenced by the ***form*** of the programming language statements.
  - The ***meaning*** of these statements and their translation into machine languages are ***of no concern*** during the macro expansion.
- The design of a macro processor is usually ***machine independent***.
- Macro processors are commonly used in *assemblers*, *high-level programming languages*, and *operating system command languages*.

SIC/XE

Line number	Machine address	Label	Instruction	Operand	Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
133	103C		+LDT	#4096	75101000
135	1040	RLOOP	TD	INPUT	E32019
140	1043		JEQ	RLOOP	332FFA
145	1046		RD	INPUT	DB2013
150	1049		COMPR	A,S	A004
155	104B		JEQ	EXIT	332008
160	104E		STCH	BUFFER,X	57C003
165	1051		TIXR	T	B850
170	1053		JLT	RLOOP	3B2FEA
175	1056	EXIT	STX	LENGTH	134000
180	1059		RSUB		4F0000
185	105C	INPUT	BYTE	X'F1'	F1
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
210	105D	WRREC	CLEAR	X	B410
212	105F		LDT	LENGTH	774000
215	1062	WLOOP	TD	OUTPUT	E32011
220	1065		JEQ	WLOOP	332FFA
225	1068		LDCH	BUFFER,X	53C003
230	106B		WD	OUTPUT	DF2008
235	106E		TIXR	T	B850
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Read records from input device 'F1' and then write to the output device '05'

SIC/XE

```
Line number  Label     Instruction  Operand                Comment
  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
```

Macro definition

Macro name

parameters

Read records from input device ‘F1’ and then write to the output device ’05’

Use macros before macro expansion

Macro invocation

SIC/XE

Comment

Read records from input device ‘F1’ and then write to the output device ’05’

Use macros after macros expansion

```
Line number  Label     Instruction  Operand              Comment
     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
```

Macro Expansion Example

- Two new assembler directives
  - ***MACRO***: the beginning of a macro
  - ***MEND***: the end of a macro
- The macro ***name*** and ***parameters*** define a pattern or prototype for the macro.
  - Macro name is the symbol before the directive MACRO.
  - In SIC/XE, each *parameter* begins with the character ***&***.
    - This facilitates the substitution of parameters during macro expansion.
- A macro invocation will introduce *macro expansion*.
  - In expanding the macro invocation, the arguments are substituted for the parameters.
  - E.g., Line 190,
    - ***F1*** is substituted for ***&INDEV***,
    - ***BUFFER*** for ***&BUFADR***, and
    - ***LENGTH*** for ***&RECLTH***.

Macro Expansion Example (Cont.)

- The macro expansion in this example:
  - The *macro invocation statement* is included as a ***comment line***. (serve as documentation)
  - The *label* on the macro invocation statement (e.g., ***LOOP***) is retained as a label on the first statement generated in the macro expansion.
- After macro processing, the expanded file can be used as input to the assembler.
  - Each macro invocation introduces the generation of macro body.
    - Statements "JEQ *-3" and "JLT *-14" are used to *avoid label duplication*.
  - Statements in a subroutine appear only once, regardless of how many times the subroutine is called.

Macro Processor Algorithm

- Two pass macro processor
  - Pass 1: All macro definitions are processed.
  - Pass 2: All macro invocation statements are expanded.
- Features of a two-pass macro processor
  - Easy to design.
  - Not 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 inside a macro can't be seen unless the outer macro is invoked and also expanded.**
  - Definitions of macros (*nested macros*) by other macros can be useful in some areas.



Example of Macros in a Macro

- A program could run on either SIC or SIC/XE machine by calling the corresponding macros.
- ***Defining*** MACROS or MACROX does not define RDBUFF and the other macros instructions.
  - The definitions are processed when an invocation to them is ***expanded***.

```
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}
```

SIC

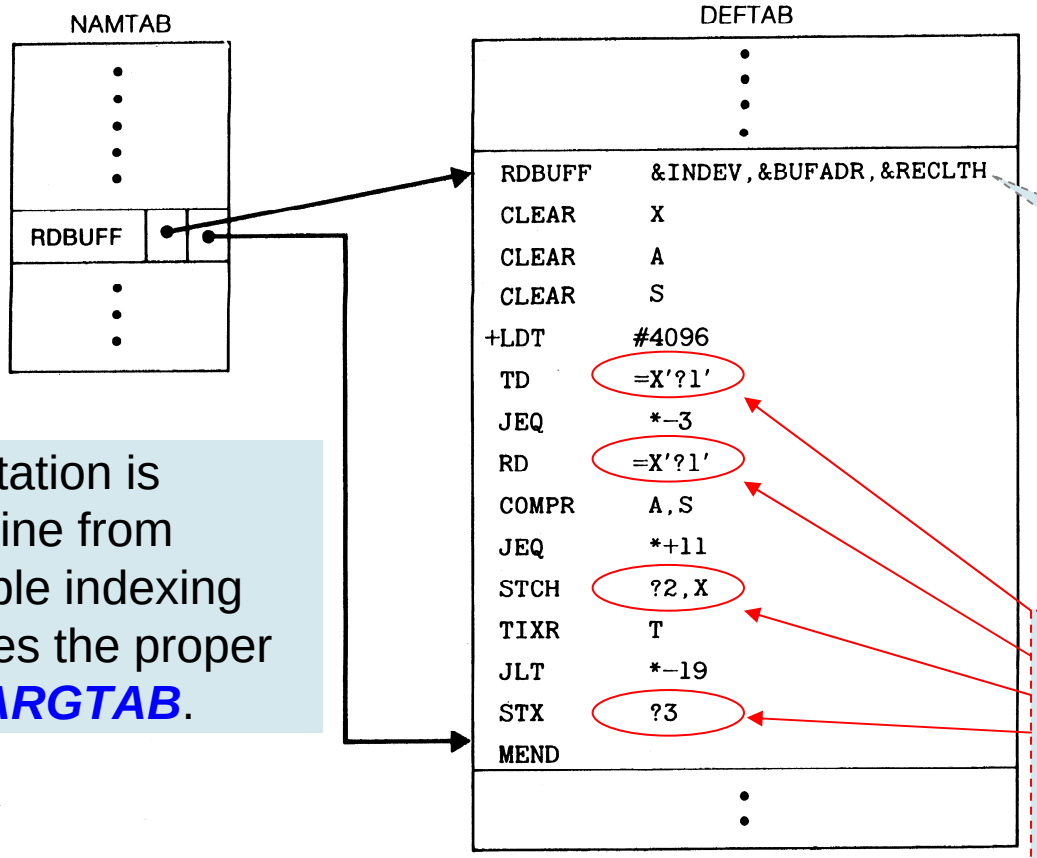
```
1  MACROX   MACRO    {Defines  SIC/XE macros}
2  RDBUFF   MACRO    &INDEV,&BUFADR,&RECLTH
                .
                .    {SIC/XE version}
                .
3           MEND     {End of RDBUFF}
4  WRBUFF   MACRO    &OUTDEV,&BUFADR,&RECLTH
                .
                .    {SIC/XE version}
                .
5           MEND     {End of WRBUFF}
                .
                .
                .
6           MEND     {End of MACROX}
```

SIC/XE

One-Pass Macro Data Structure

- A one-pass algorithm could handle macros in a macro body because it can alternate between ***macro definition*** and ***macro expansion***.
- Because of the one-pass structure, the definition of a macro must appear in the source program *before* any statement that invokes the macro.
- Three data structures involved:
  - Definition table (***DEFTAB***):
    - Macro definitions and body are stored.
    - Comment lines are skipped.
    - References to macro instruction parameters are converted to a *positional notation*.
  - Name table (***NAMTAB***):
    - Macro names with pointers to the beginning and end of the macro in DEFTAB.
  - Argument table (***ARGTAB***):
    - Store *invocation parameters* that are used during the expansion of macro invocation.

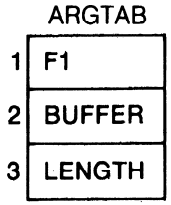
One-Pass Data Structure (Cont.)



When the ***?n*** notation is recognized in a line from ***DEFTAB***, a simple indexing operation supplies the proper argument from ***ARGTAB***.

Definition of RDBUFF stored in DEFTAB.

Positioning notation (to enhance the performance on macro expansion)



Macro invocation (Line 190):
CLOOP RDBUFF **F1**, **BUFFER**, **LENGTH**



One-Pass Macro Algorithm

- Procedure ***DEFINE***:
  - Being called when the beginning of a macro definition is recognized.
  - Make appropriate entries in *DEFTAB* and *NAMTAB*.
- Procedure ***EXPAND***:
  - Being called to set up the argument values in *ARGTAB*.
  - Expand a macro invocation statement.
- Procedure ***GETLINE***:
  - Being called at several points to get a line in the algorithm:
  - The line may come from
    - The input file (EXPANDING = FALSE)
    - The DEFTAB (EXPANDING = TRUE)
- Counter ***LEVEL***:
  - Count the macro level (similar to match left and right parentheses):
    - When a MACRO directive is encountered, LEVEL is advanced by 1.
    - When a MEND directive is encountered, LEVEL is decreased by 1.

Note: Most macro processors allow the definitions of commonly used macro instructions to appear in a standard system library (to make macro uses convenient).



One-Pass Macro Algorithm (Cont.)

```
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}
```

```
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}
```



Machine-Independent Macro Processor Features

Concatenation of Macro Parameters

- Suppose that a macro instruction is name &ID, the body of the macro definition like

  LDA X&ID1

  - &ID is concatenated after the character string X and before 1.
  - The end of the parameter is not marked.

- ***Special concatenation operator***
  - Most macro processors use it to solve the right marker problem.
  - In SIC, this operator is →, so that the above example can be written as:

    LDA X&ID→1



Concatenation of Macro Parameters (Cont.)

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

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

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



Generation of Unique Labels

- If the example program in this chapter include a label on the TD statement (Line 135), this label would be defined twice.
  - The relative addressing in a source statement (e.g., *-3 and *-14) would not be acceptable for long jumps.
  - Long jumps over several instructions are inconvenient, error-prone, and difficult to read.
- Many processor creates ***special labels*** within macros instructions to solve the labeling problem.
  - Each symbol begins with ***$*** is modified by ***$xx***, where ***xx*** is a two-character alphanumeric counter.
    - E.g., first expansion with $AA, and the succeeding is $AB, $AC, etc.

Generation of Unique Labels (Cont.)

```
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
```

Macro

```
               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
```

After macro expansion with the macro invocation:
RDBUF *F1*, *BUFFER*, *LENGTH*

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.
- ***For example*** (listed in the next slides):
  - The definition of the macro **RDBUFF** has two additional parameters:
    - &EOR: Specify a hexadecimal character code that marks the end of a record.
    - &MAXLTH: Specify the maximum length record that can be read.



Conditional Macro Expansion (Cont.)

SET is a macro processor directive.
&EORCK is a *macro-time variable* (also called *set symbol*) that is used to store working values during macro expansion.
- Any symbols that begin with the character ***&*** and that is not a macro instruction parameter is assumed to be a macro-time variable.
- All such variables are initialized to a value as *0*.

```
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
```

If an argument corresponding to *&EOR*, the variable *&EORCK* is set to 1. Otherwise, &EORCK remains 0.

```
      .         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
```

If &MAXLTH equals to the null string, Line 45 is generated. Otherwise Line 47 is generated.

Examining the value of variables is faster than repeating the original test, especially if the test involves complicated expression.



Conditional Macro Expansion (Cont.)

-The macro processor must maintain a symbol table that contains the values of all used macro-time variables.
-The testing of Boolean expressions in IF statements occurs ***at the time macros are expanded***.
-Entries in this table are made or modified when **SET** statements are used.
-When an **IF** statement is encountered, the Boolean expression is evaluated to determine which part of statements should be expanded.

```
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
```

```
        .       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
```



Conditional Macro Expansion (Cont.)

The nested IF structure is not allowed in this example.

```
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
```

```
        .       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
65              COMPR    A,S            TEST FOR END OF RECORD
70              JEQ      $ACEXIT        EXIT LOOP IF EOR
75              STCH     BUFF,X         STORE CHARACTER IN BUFFE
80              TIXR     T              LOOP UNLESS MAXIMUM LENG
85              JLT      $ACLOOP          HAS BEEN REACHED
90    $ACEXIT   STX      RLENG          SAVE RECORD LENGTH
```

Test at run time.

Conditional Macro Expansion (Cont.)

- The macro-time **IF-THEN-ELSE** structure provides a mechanism for either generating or skipping selected statements in the macro body.
- The macro-time *looping* statement **WHILE** specifies that the following lines (until the next **ENDW**) are to be generated repeated as long as a particular condition is true.
- The programmer is also allowed to provide a list corresponding to the same parameter.
  - E.g., (00, 03, 04) corresponds to the parameter &EOR.



Conditional Macro Expansion (Cont.)

%NTIMES is a macro processor function that reuturns the number of members in an argument list. E.g., when &EOR= (00, 03, 04), **%NITEMS(&EOR)** returns 3.
&CTR is used to count the number of times the lines following the WHILE have been generated.

&EORCT = 3

```
 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
```

Initialized to 1

Incremented by 1

Generated 3 times

```
      .       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
```

The nested WHILE structure is not allowed in this example.



Keyword Macro Parameters

- Format of macro parameters:
  - ***Positional parameters***:
    - ***Parameters*** *in the macro prototype* and ***arguments*** *in the macro invocation statement* were associated with each other according to their positions.
  - ***Keyword parameters***:
    - Only parameters that has corresponding arguments in the macro invocation need to be listed. (Others adopt the default values.)
    - Simplify the macro definition in many cases.
    - Good for macros with a large number of parameters.
    - For example: macro GENER has 10 parameters
      - Positional parameter method: `GENER ,,DIRECT,,,,,,3`
      - Keyword parameter method: `GENER TYPE=DIRECT, CHANNEL=3`



Keyword Macro Parameters

Specify a keyword parameter

Default value

```
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
```

```
        .       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
```

```
                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

- The one-pass assembler algorithm introduced in this chapter can't deal with the *invocations of macros within macros*.

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

Parameter	Value
1	F1
2	(unused)
.	.

```
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
```

```
 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
```

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

Procedure **EXPAND** is called during preprocessing.
EXPANDING = TRUE

Procedure **PROCESSLINE** call **EXPAND** again.
EXPANDING = TRUE

EXPANDING = FALSE

Test-and-wait loop to control access

Set EXPANDING = FALSE



Recursive Macro Expansion (Cont.)

- The cause of these difficulties:
  - *The recursive call of the procedure **EXPAND***.
    - When the RDBUFF macro invocation is encountered, ***EXPAND*** is called.
    - Later it calls PROCESSLINE for Line 50, which results in another call to ***EXPAND*** before a return is made from the original call.
  - ***PROCESSLINE** would be called recursively.*
    - From ***main (outermost) loop*** of the macro processor logic
    - From the loop within ***EXPAND***
- If a programming language supports recursive calls (like C), it problem could be solve automatically.
  - Save registers and parameters automatically on each call, and restore them on return.
- If a programming language does not support recursive calls, the looping structure should *save data values on a stack.*



General-Purpose Macro Processors

- General-purpose macro processors are independent to any particular programming language.
  - General-purpose macro processors have higher development cost.
  - The development cost does not need to be repeated.
- The large number of details that needs to be dealt with makes general-purpose macro processors less popular.
  - E.g., Each programming language has its own comments:
    - Pascal and C use special character to mark the start and end of a comment.
    - Ada uses a special character to mark the start of a comment that is automatically terminated at the end of the source line.
    - FORTRAN uses a special symbol to flag an entire line as a comment.
    - Some assembler languages consider characters following the end of the instructions as comments.
    - Some recognize comments according their position in the source line. (COBOL)



General-Purpose Macro Processors (Cont.)

- A general-purpose macro processor may need to take *groupings* into consideration to group ***terms***, ***expressions***, and ***statements***.
  - Some languages use keywords ***begin*** and ***end*** for grouping. (Pascal)
  - Some uses ***{*** and ***}*** (C and Java)
  - Some uses ***(*** and ***)***
- A general problem involves the ***tokens*** of the programming languages.
  - Tokens are identifiers, constants, operators, and keywords.
  - Languages differ substantially on their tokens.
    - Some have multiple-character operators such as ** in FORTRAN and := in Pascal.
      - Macro processors may consider them as two characters.
    - Blanks may be significant and may be not.
- Another program is the ***syntax*** used for macro definition and macro invocation statements.



Macro Processing within Language Translators (Integrated Macro Processor)

- Combining the macro processing functions with the language translator is another design option.
- The simplest method is to combine a ***line-by-line macro processor***.
  - The macro processor reads the source program statements and performs all macro processing functions.
  - The output lines are passed to the language translator as they are generated (one at a time).
    - The macro processor operates like an ***input routine*** for the assembler or complier.
- Advantages of the line-by-line approach:
  - Avoid making an extra pass over the source program.
  - Combine data structures together. E.g., *OPTAB* and *NAMTAB*.
  - Share utility subroutines and functions.
    - E.g., Scanning input lines, table searching, and converting numeric values.
  - Give diagnostic messages related to the source statement containing errors.
- The main form of communication between integrated macro processor and language translator is the *passing of source statements from one to the other*.

Macro Processing within Language Translators (Cont.)

- The integrated macro processor may use the results of translator operations such as scanning symbols and constants.
  - This is useful when the rules vary from one part of the program to another. (e.g., FORTRAN)
- FORTRAN example:

A Loop: DO 100 I = 1,20

DO is a keyword
100 is a statement number

An assignment: DO 100 I = 1

DO100I is an variable

  - The macro processor would be very difficult to distinguish them.

Macro Processing within Language Translators (Cont.)

- An integrated macro processor can support ***macro instructions***.
  - Macro instructions depend on the context in which they occur.
- Disadvantages of integrated macro processors:
  - They must be specially designed and written to work with a particular implementation of an assembler or compiler.
  - The costs of development must be added to the cost of the language translator → Make translators more complicate.
  - The size may be a problem if the translator is to run on a computer with limited memory.



Implementation Examples



MASM Macro Processor

- The macro processor of MASM is integrated with *Pass 1 of the assembler*.
- Macros may be redefined in a program to replace the first one. But this is confusing in practice.

|OP1 – OP2|

```
1   ABSDIF    MACRO     OP1,OP2,SIZE
2             LOCAL     EXIT
3             IFNB      <SIZE>          ;; IF SIZE IS NOT BLANK
4             IFDIF     <SIZE>,<E>      ;;    THEN IT MUST BE E
5             ; ERROR -- SIZE MUST BE E OR BLANK
6             .ERR
7             EXITM
8             ENDIF                     ;; END OF IFDIF
9             ENDIF                     ;; END OF IFNB
10            MOV       SIZE&AX,OP1     ; COMPUTE ABSOLUTE DIFFERENCE
11            SUB       SIZE&AX,OP2     ;; SUBTRACT OP2 FROM OP1
12            JNS       EXIT            ;; EXIT IF RESULT GE 0
13            NEG       SIZE&AX         ;;   OTHERWISE CHANGE SIGN
14    EXIT:
15            ENDM
```

Declare **EXIT** a local label

Direct MASN to terminate

Singal MASN an error

Macro comment

Ordinary assembly comment

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

Indicate using **EAX**

doubleword registers

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

Word registers

```
ABSDIF    P,Q,X
   ↓
; ERROR -- SIZE MUST BE E OR BLANK
```

Generate unique names for labels: ??0000 ~ ??FFFF



MASM Macro Processor (Cont.)

IRP sets S to a sequence of values.

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

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



ANSI C Macro Language

- In ANSI C language, macro definitions and invocations are handled by a preprocessor.
- The preprocessor is not integrated with the compiler.
- For example:

```
#define   NULL    0
#define   EOF     (-1)
```

Avoid the common C error of writing = in place of ==.

```
#define   EQ      ==
```

`while (I EQ 0) …` ⇒ `while (I == 0) …`

Using macros is more efficient than using functions in this case.

```
#define  ABSDIFF(X,Y)   ((X) > (Y) ? (X) – (Y) : (Y) – (X))
```

`ABSDIFF(I+1,J-5)` ⇒ `((I+1) > (J-5) ? (I+1) – (J-5) : (J-5) – (I+1))`

`ABSDIFF(I, 3.14159)` `ABSDIFF('D', 'A')` Support different data types

```
#define  ABSDIFF(X,Y)   X > Y ? X – Y : Y – X
```

Result becomes -14 instead of 2.

`ABSDIFF(3+1, 10-8)` ⇒ `3+1 > 10-8 ? 3+1-10-8 : 10-8-3+1`



ANSI C Macro Language (Cont.)

```
#define  DISPLAY(EXPR)  printf("EXPR = %d\n", EXPR)
```

DISPLAY(I*J+1) ⇒ printf("EXPR = %d\n", I*J+1)

Quoted string won't be expanded

Special "stringizing" operator #

```
#define  DISPLAY(EXPR)  printf(#EXPR "= %d\n", EXPR)
```

DISPLAY(I*J+1) ⇒ printf("I*J+1" "= %d\n", I*J+1) ⇔ printf("I*J+1 = %d\n", I*J+1)

DISPLAY(ABSDIFF(3,8)) ⇒ printf("ABSDIFF(3,8)" "= %d\n", ABSDIFF(3,8))

Macro in macro

⇒ printf("ABSDIFF(3,8)" "= %d\n", ",((3) > (8) ? (3) – (8) : *8) – (3)))

If the body of a macro contains a token that happens to match the name of the macro, the token is not replaced during recanning.

ANSI C Macro Language (Cont.)

- Conditional compilation
  - To make sure that a macro (or other name) is defined at least once:

```
#ifndef     BUFFER_SIZE
#define     BUFFER_SIZE 1024
#endif
```

  - Control debugging statements:

```
#define DEBUG 1
.
.
.
#if DEBUG == 1
printf(...)       /* debugging output */
#endif
```

⟺

```
#ifdef DEBUG
printf(...)       /* debugging output */
#endif
```



ELENA Macro Processor

- The ELENA is a general-purpose macro processor.
  - It is a research tool, not a commercial software product.
- Macro definitions in ELENA are composed of a *header* and a *body*.
  - The header is not required to have any special form.
  - It consists of a sequence of *keywords* and *parameter markers*.
  - Parameter markers are identified by the special character %.
  - At least one of the first two tokens in a macro header must be a *keyword*, not a parameter marker.
  - A *macro invocation* is a *sequence of tokens* that matches the *macro header*.

```
%1 = %2 + %3  ⇒  ALPHA = BETA + GAMMA
Header            Invocation
```

```
ADD %1 TO THE VALUE OF %2  ⇒  ADD 10 TO THE VALUE OF INDEX
Header                        Invocation
```



ELENA Macro Processor (Cont.)

```
%1 := ABSDIFF(%2,%3)
```

Header

(a)

Body for C

```
%1 = (%2) > (%3) ? (%2) - (%3) : (%3) - (%2)
```

(b)

```
Z := ABSDIFF(X,Y)
```

Macro invocation

↓

```
Z = (X) > (Y) ? (X) - (Y) : (Y) - (X)
```

(c)

After macro expansion

```
Z := ABSDIFF(X,Y)
```

Macro invocation

↓

```
        MOV     EAX,X
        SUB     EAX,Y
        JNS     STOR0001
        NEG     EAX
STOR0001 MOV    EAX,Z
```

(e)

After macro expansion

```
        MOV     EAX,%2
        SUB     EAX,%3
        JNS     &STOR
        NEG     EAX
&STOR   MOV     EAX,%1
```

Body for assembly

(d)

&STOR is changed to &STOR0001 because & identify &STOR as a local label within the macro definition.



ELENA Macro Processor (Cont.)

```
ADD %1 TO THE FIRST %2 ELEMENTS OF V
```

Header

Set macro-time variable **.LAA** to 1

(a)

```
        .SET .LAA = 1
.E      V(.LAA) = V(.LAA) + %1
        .SET .LAA = .LAA + 1
        .IF .LAA LE %2 .JUMP .E
```

Body

Increase 1 to **.LAA**

The macro-time instruction .IF causes the macro processor to jump back to the line with the macro-time label .E, if .LAA < %2.

(b)

Macro invocation

```
ADD 5 TO THE FIRST 3 ELEMENTS OF V
```

↓

```
V(1) = V(1) + 5
V(2) = V(2) + 5
V(3) = V(3) + 5
```

After macro expansion

(c)



ELENA Macro Processor (Cont.)

- The ELENA macro processor uses a macro definition table.
- A macro is identified by the sequence of keywords that appear in its header.
  - For example: Two macro headers

```
ADD %1 TO %2
ADD %1 TO THE FIRST ELEMENT OF %2
```

DISPLAY %1 %1 TABLE — Header

DISPLAY TABLE — Invocation

Ambiguous

- ELENA constructs an *index* of all macro headers according to *the keywords in the first two tokens* of the header.
  - Invocation should match at least one of their first tokens.

A SUM B, C (Invocation): Compare all macro headers with the first token is A or the second token is SUM

A = B + 1 — Invocation

%1 = %2 + %3

%1 = %2 + 1

Matched header: select the header with the fewest parameters. Otherwise, select the most recently defined macro.