4.2 Machine-Independent Macro Processor Features

- Extensions to the basic macro processor functions
  - Concatenation of Macro Parameters
  - Generation of Unique Labels
  - Conditional Macro Expansion
  - Keyword Macro Parameters
4.2.1 Concatenation of Macro Parameters

- Allow parameters to be concatenated with other character strings
- Suppose the parameter is named `&ID`, the macro body may contain a statement:
  
  ```
  LDA X&ID1
  ```

  - `&ID` is concatenated after the string “X” and before the string “1”.
  
  → LDA XA1 (&ID=A)
  → LDA XB1 (&ID=B)
4.2.1 Concatenation of Macro Parameters

- Problem: ambiguous situation
  - E.g., X&ID1 may mean
    - “X” + &ID + “1”
    - “X” + &ID1
  - This problem occurs because the end of the parameter is not marked.

- Solution:
  - Use a special *concatenation operator* “->” to specify the end of the parameter
  - E.g. LDA X&ID→1

- Example: Figure 4.6
Concatenation of Macro Parameters (Fig. 4.6)

(a)  
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

(b)  
SUM  ↓  A
LDA  ADD  ADD  STA  XAS
XA1  XA2  XA3

(c)  
SUM  ↓  BETA
LDA  ADD  ADD  STA  XBETAS
XBETA1  XBETA2  XBETA3

4.2.2 Generation of Unique Labels

- Labels in the macro body may have *duplicate labels* problem
  - If the macro is invoked multiple times.
  - Use of relative addressing is very inconvenient, error-prone, and difficult to read.
- Example
  - `JEQ *-3`
  - Inconvenient, error-prone, difficult to read
4.2.2 Generation of Unique Labels

- Generating unique labels within macro expansions
  - Labels within the macro body begin with the character $.
  - During macro invocation, $ will be replaced by $xx,
    - xx is a two-character alphanumeric counter of the number of macro instructions expanded.

- Example: Figure 4.7
  - $ LOOP TD =X'&INDEV'
  - 1st call:
    - $A LOOP TD =X'F1'
  - 2nd call:
    - $AB LOOP TD =X'F1'
Generation of unique labels within macro expansion (fig. 4.7)

- Macro definition

```
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    TD =X’&INDEV’ TEST INPUT DEVICE
55    JEQ $LOOP LOOP UNTIL READY
60    RD =X’&INDEV’ READ CHARACTER INTO REG A
65   COMPR A, $ 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 SAVE RECORD LENGTH
95    STX &RECLTH
95    MEND
```
Generation of unique labels within macro expansion (fig. 4.7) (Cont.)

RDBUFF  F1, BUFFER, LENGTH

• Macro expansion

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>CLEAR</td>
<td>X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>35</td>
<td>CLEAR</td>
<td>A</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>40</td>
<td>CLEAR</td>
<td>S</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>45</td>
<td>+LDT</td>
<td>#4096</td>
<td>SET MAXIMUM RECORD LENGTH</td>
</tr>
<tr>
<td>50</td>
<td>TD</td>
<td>=X'F1'</td>
<td>READ CHARACTER INTO REG A</td>
</tr>
<tr>
<td>55</td>
<td>JEQ</td>
<td>$AALOOP</td>
<td>TEST FOR END OF RECORD</td>
</tr>
<tr>
<td>60</td>
<td>RD</td>
<td>=X'F1'</td>
<td>EXIT LOOP IF EOR</td>
</tr>
<tr>
<td>65</td>
<td>COMPR</td>
<td>A, S</td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>70</td>
<td>JEQ</td>
<td>$AAEXIT</td>
<td>LOOP UNLESS MAXIMUM LENGTH HAS BEEN REACHED</td>
</tr>
<tr>
<td>75</td>
<td>STCH</td>
<td>BUFFER, X</td>
<td>SAVE RECORD LENGTH</td>
</tr>
<tr>
<td>80</td>
<td>TIXR</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>JLT</td>
<td>$AALOOP</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>STX</td>
<td>LENGTH</td>
<td></td>
</tr>
</tbody>
</table>
4.2.3 Conditional Macro Expansion

- Arguments in macro invocation can be used to:
  - Substitute the parameters in the macro body.
  - Modify the sequence of statements in macro body for *conditional macro expansion*.
    - This capability adds greatly to the power and flexibility of a macro language.
4.2.3 Conditional Macro Expansion

- **Macro-time conditional statements**
  - Macro processor directives:
    - `IF-ELSE-ENDIF`
    - `SET`
  - Example: Figure 4.8

- **Macro-time variables** (also called a *set symbol*)
  - Be used to store working values during the macro expansion
  - Any symbol that begins with the character `&` and is not a macro parameter
  - Be initialized to 0
  - Be changed with their values using `SET`
    - `&EORCK SET 1`
```
25    RDBUFF   MACRO   &INDEV, &BUFADR, &RECLTH, &EOR, &MAXLTH
26    &EORCK   IF    (&EOR NE '')
27     SET    1
28    ENDIF    CLEAR   X   CLEAR LOOP COUNTER
29    CLEAR   A
30    IF    (&EORCK_EQ 1)
31     LDCH    =X'&EOR'   SET EOR CHARACTER
32    ENDIF
33    IF    (&MAXLTH_EQ '')
34     LDT     #4096   SET MAX LENGTH = 4096
35    ELSE
36    LDT     ?>&MAXLTH   SET MAXIMUM RECORD LENGTH
37    ENDIF
38    $LOOP   TD     =X'&INDEV'   TEST INPUT DEVICE
39     JEQ    $LOOP   LOOP UNTIL READY
40    RD      =X'&INDEV'   READ CHARACTER INTO REG A
41    IF    (&EORCK_EQ 1)
42     COMP    A, S   TEST FOR END OF RECORD
43    ENDIF
44    $EXIT   JEQ    $EXIT   EXIT LOOP IF EOR
45    ENDIF
46    STCH    &BUFADR, X   STORE CHARACTER IN BUFFER
47    TIXR    T   LOOP UNLESS MAXIMUM LENGTH
48    JLT    $LOOP   HAS BEEN REACHED
49    $EXIT   STX    &RECLTH   SAVE RECORD LENGTH
50    MEND
```
Use of Macro-time Conditional Statements (Fig. 4.8) (Cont.)

**RDBUFF F3, BUF, RECL, 04, 2048**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>CLEAR X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>35</td>
<td>CLEAR A</td>
<td>SET EOR CHARACTER</td>
</tr>
<tr>
<td>40</td>
<td>LDCH =X’04’</td>
<td>SET MAXIMUM RECORD LENGTH</td>
</tr>
<tr>
<td>42</td>
<td>RMO A,S</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>47</td>
<td>+LDT #2048</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>50</td>
<td>$AALOOP TD =X’F3’</td>
<td>READ CHARACTER INTO REG A</td>
</tr>
<tr>
<td>55</td>
<td>JEQ $AALOOP</td>
<td>TEST FOR END OF RECORD</td>
</tr>
<tr>
<td>60</td>
<td>RD =X’F3’</td>
<td>EXIT LOOP IF EOR</td>
</tr>
<tr>
<td>65</td>
<td>COMPR A,S</td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>70</td>
<td>JEQ $AAEXIT</td>
<td>LOOP UNLESS MAXIMUM LENGTH</td>
</tr>
<tr>
<td>75</td>
<td>STCH BUF,X</td>
<td>HAS BEEN REACHED</td>
</tr>
<tr>
<td>80</td>
<td>TIXR T</td>
<td>SAVE RECORD LENGTH</td>
</tr>
<tr>
<td>85</td>
<td>JLT $AALOOP</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>$AAEXIT STX RECL</td>
<td></td>
</tr>
</tbody>
</table>
Use of Macro-time Conditional Statements (Fig. 4.8) (Cont.)

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
87          JLT $ABLOOP LOOP UNLESS MAXIMUM LENGTH HAS BEEN REACHED
90        $ABEXIT STX LENGTH SAVE RECORD LENGTH

(c)
Use of Macro-time Conditional Statements (Fig. 4.8) (Cont.)

<table>
<thead>
<tr>
<th>RDBUFF</th>
<th>F1, BUFF, RLENG, 04</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>CLEAR X</td>
</tr>
<tr>
<td>35</td>
<td>CLEAR A</td>
</tr>
<tr>
<td>40</td>
<td>LDCH =X′04′</td>
</tr>
<tr>
<td>42</td>
<td>RMO A, S</td>
</tr>
<tr>
<td>45</td>
<td>+LDT #4096</td>
</tr>
<tr>
<td>50</td>
<td>$ACLOOP TD =X′F1′</td>
</tr>
<tr>
<td>55</td>
<td>JEQ $ACLOOP</td>
</tr>
<tr>
<td>60</td>
<td>RD =X′F1′</td>
</tr>
<tr>
<td>65</td>
<td>COMPR A, S</td>
</tr>
<tr>
<td>70</td>
<td>JEQ $ACEXIT</td>
</tr>
<tr>
<td>75</td>
<td>STCH BUFF, X</td>
</tr>
<tr>
<td>80</td>
<td>TIXR T</td>
</tr>
<tr>
<td>85</td>
<td>JLT $ACLOOP</td>
</tr>
<tr>
<td>90</td>
<td>$ACEXIT STX RLENG</td>
</tr>
</tbody>
</table>

(d)
The testing of Boolean expression in IF statements occurs \textit{at the time macros are expanded}. By the time the program is assembled, all such decisions have been made. There is only one sequence of source statements during program execution.

In contrast, the COMPR instruction tests data values \textit{during program execution}. The sequence of statements that are executed during program execution may be different.
Conditional Macro Expansion (Cont.)

- **Macro-time looping statement**
  - Macro processor directives:
    - **WHILE-ENDW**
  - Example: Figure 4.9

- **Macro processor function**
  - **%NITEMS**: the number of members in an argument list
    - E.g. &EOR=(00,03,04)
      => %NITEMS(&EOR) is 3
    - Specify member in the list: &EOR[1]
Use of Macro-time Looping Statements (Fig. 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)
Use of Macro-time Looping Statements (Fig. 4.9) (Cont.)

```verbatim
RDBUFF F2, BUFFER, LENGTH, (00, 03, 04)
```

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>CLEAR X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>35</td>
<td>CLEAR A</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>+LDT #4096</td>
<td>SET MAX LENGTH = 4096</td>
</tr>
<tr>
<td>50</td>
<td>$AALOOP TD =X’F2’</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>55</td>
<td>JEQ $AALOOP</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>60</td>
<td>RD =X’F2’</td>
<td>READ CHARACTER INTO REG A</td>
</tr>
<tr>
<td>65</td>
<td>COMP =X’000000’</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>JEQ $AAEXIT</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>COMP =X’000003’</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>COMP =X’000004’</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>STCH BUFFER, X</td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>90</td>
<td>$AAEXIT STX LENGTH</td>
<td>SAVE RECORD LENGTH</td>
</tr>
</tbody>
</table>

**Figure 4.9** Use of macro-time looping statements.
4.2.4 Keyword Macro Parameters

- Positional parameters
  - Parameters and arguments are associated according to their positions in the macro prototype and invocation.
  - If an argument is to be omitted, a null argument (two consecutive commas) should be used to maintain the proper order in macro invocation:
    - E.g. RDBUFF 0E, BUFFER, LENGTH, , 80
  - It is not suitable if a macro has a large number of parameters, and only a few of these are given values in a typical invocation.
4.2.4 Keyword Macro Parameters (Cont.)

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

```assembly
25  RDBUFF  MACRO   &INDEV=F1, &BUFADR=, &RECLTH=, &EOR=04, &MAXLTH=4096
26  IF      (&EOR NE ?)
27  &BORCK  SET    1
28  ENDIF
30  CLEAR   X       CLEAR LOOP COUNTER
35  CLEAR   A
38  IF      (&BORCK 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  LF      (&BORCK 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 HAS BEEN REACHED
85  JLT     $LOOP   SAVE RECORD LENGTH
90  $EXIT   STX     &RECLTH
95  MEND
```
Use of keyword parameters in macro instructions (Fig. 4.10) (Cont.)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>CLEAR X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>35</td>
<td>CLEAR A</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>LDCH =X'04'</td>
<td>SET EOR CHARACTER</td>
</tr>
<tr>
<td>42</td>
<td>RMO A,S</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>+LDT #4096</td>
<td>SET MAXIMUM RECORD LENGTH</td>
</tr>
<tr>
<td>50</td>
<td>$AALOOP TD =X'F1'</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>55</td>
<td>JEQ $AALOOP</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>60</td>
<td>RD =X'F1'</td>
<td>READ CHARACTER INTO REG A</td>
</tr>
<tr>
<td>65</td>
<td>COMF A,S</td>
<td>TEST FOR END OF RECORD</td>
</tr>
<tr>
<td>70</td>
<td>JEQ $AAEXIT</td>
<td>EXIT LOOP IF EOR</td>
</tr>
<tr>
<td>75</td>
<td>STCH BUFFER,X</td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>80</td>
<td>TIXR T</td>
<td>LOOP UNLESS MAXIMUM LENGTH HAS BEEN REACHED</td>
</tr>
<tr>
<td>85</td>
<td>JLT $AALOOP</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>$AAEXIT STX LENGTH</td>
<td>SAVE RECORD LENGTH</td>
</tr>
</tbody>
</table>

Figure 4.10 Use of keyword parameters in macro instructions.
Use of keyword parameters in macro instructions (Fig. 4.10) (Cont.)

```
RDEBUFF RECLTH=LENGTH, BUFADR=BUFFER, ECR=, INDEV=23
```

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)
4.3 Macro Processors Design Options

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

- Recursive macro expansion
  - Macro invocations within macros
  - Example: Figure 4.11

- Problems in previous macro processor design:
  - Values in ARGTAB were *overwritten*
    - The procedure EXPAND would be called recursively
    - Thus the invocation arguments in the ARGTAB will be *overwritten*.
  - Recursive call of the procedure EXPANDINGING
    - The Boolean variable EXPANDINGING would be set to FALSE when the “inner” macro expansion is finished
    - That is, the macro process would *forget* that it had been in the middle of expanding an “outer” macro.
Example of nested macro invocation (Fig. 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
```
Example of nested macro invocation (Fig. 4.11) (Cont.)

(b)

RDBUFF BUFFER, LENGTH, F1

(c)

Figure 4.11 Example of nested macro invocation.
4.3.1 Recursive Macro Expansion (Cont.)

- **Solutions:**
  - Write the macro processor in a programming language that allows recursive calls
    - Thus local variables will be retained.
    - Most high-level language have been supported recursive calls
    - The compiler would be sure that previous values of any variables declared within a procedure were saved when the procedure was called recursively
  - Use a *stack* to take care of *pushing and popping local variables* and *return addresses*
4.3.2 General-Purpose Macro processors

- Three examples of actual macro processors:
  - A macro processor designed for use by assembler language programmers
  - Used with a high-level programming language
  - General-purpose macro processor
    - Not tied to any particular language
    - Can be used with a variety of different languages.
General-Purpose Macro processors
(Cont.)

- General-purpose macro processors
  - Advantages
    - Programmers do not need to learn many macro languages.
    - Overall saving in software development cost and software maintenance effort
  - Difficulties:
    - Large number of details must be dealt with in a real programming language
      - Comment identifications ( //, /* */, …)
      - Grouping together terms, expressions, statements (begin_end, { }, …)
      - Tokens (keywords, operators)
    - …
4.3.3 Macro processing within language translators

- Macro processors can be
  - **Preprocessors**
    - Produce an expanded version of the source program, which is then used as input to an assembler or compiler
  - **Line-by-line macro processor**
    - Used as a sort of input routine for the assembler or compiler
      - Read source program
      - Process macro definitions and expand macro invocations
      - Pass output lines to the assembler or compiler
  - **Integrated macro processor**
4.3.3 Macro processing within language translators

- **Preprocessors**

  ![Diagram showing the process of preprocessor]

- **Combining macro processing functions with the language translator itself**

  ![Diagram showing the process of combining macro functions]

Source Code (with `macro`) → Macro Processor → Expanded Code → Compiler / Assembler → Object program
Macro processing within language translators (Cont.)

- Combining macro processing functions with the language translator itself
  - *Line-by-line* macro processor:
  - *Integrated* macro processor

- Advantages: share some data structures, functions
- Disadvantages: more complex
Line-by-Line Macro Processor

- **Benefits**
  - It avoids making an extra pass over the source program.
  - *Data structures* required by the macro processor and the language translator can be combined
    - E.g., OPTAB and NAMTAB)
  - *Utility subroutines* can be used by both macro processor and the language translator.
    - Scanning input lines
    - Searching tables
    - Data format conversion
  - It is easier to give diagnostic messages related to the source statements.
    - i.e., the source statement error can be quickly identified without need to backtrack the source
Integrated Macro Processor

- Integrate a macro processor with a language translator (e.g., compiler)

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

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

- MASM Macro Processor
- ANSI C Macro Language
- The ELENA Macro Processor
4.4.1 MASM Macro Processor

- Macro processor in Microsoft MASM assembler
  - Integrated with pass 1 of the assembler
  - Supports all of the main macro processor functions discussed previously
  - Iteration statement:
    - E.g. IRP S, '<LEFT','DATA','RIGHT'>
Examples of MASM macro and conditional statements (Fig. 4.12)

```
1  ABSDIF  MACRO  OP1, OP2, SIZE
2       LOCAL  EXIT
3       IFNE  <SIZE>,,<F>  ;; IF SIZE IS NOT BLANK
4       IFDIF  <SIZE>,,<F>  ;; THEN IT MUST BE F
5       ;; ERROR -- SIZE MUST BE F OR BLANK
6       ,ERR
7       EXIT
8       ENDFD  ;; END OF IFDIF
9       ENDFI  ;; END OF IFN
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
```

(a)

```
ABSDIF  J, K

↓

MOV  AX, J  ;; COMPUTE ABSOLUTE DIFFERENCE
SUB  AX, AX
JNS  ??0000
NEG  AX

??0000:
```

(b)
Examples of MASM macro and conditional statements (Fig. 4.12) (Cont.)

```
ABSDIF M,N,E

; COMPUTE ABSOLUTE DIFFERENCE
MOV EAX,M
SUB EAX,N
JNS ??0001
NEG EAX

??0001:

(c)

ABSDIF P,Q,X

; ERROR -- SIZE MUST BE E OR BLANK

(d)
```

Figure 4.12  Examples of MASM macro and conditional statements.
Examples of MASM iteration statements (Fig. 4.13)

(a)

<table>
<thead>
<tr>
<th></th>
<th>NODE</th>
<th>MACRO</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>IRP</td>
<td>s,&quot;LEFT&quot;,&quot;DATA&quot;,&quot;RIGHT&quot;</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NAME&amp;s</td>
<td>DW</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>ENDM</td>
<td></td>
<td>; ; END OF IRP</td>
</tr>
<tr>
<td>5</td>
<td>ENDM</td>
<td></td>
<td>; ; END OF MACRO</td>
</tr>
</tbody>
</table>

(b)

`NODE X`

```
XLEFT  DW    0
XDATA  DW    0
XRIGHT DW    0
```

Figure 4.13 Example of MASM iteration statement.
4.4.2 ANSI C Macro Language

- Definitions and invocations of macros are handled by a preprocessor.
  - Simply makes string substitutions, without considering the syntax of the C

- Macro definition:
  - E.g. `#define NULL 0`
  - E.g. `#define AB(X,Y) ( X > Y ? X – Y : Y – X )`

- Macro invocation
  - E.g. `AB(3,4)`
4.4.3 The ELENA Macro Processor

- **ELENA**
  - A research tool, not as a commercial software product.

- **Macro definitions are composed of a header and a body.**
  - **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 (.)
Examples of ELENA macro definition and invocation (Fig. 4.14)

\[
\%1 := \text{ABSDIFF}(\%2, \%3)
\]

(a) Macro definition (header)

\[
\%1 = (\%2 > \%3) \ ? (\%2) - (\%3) : (\%3) - (\%2)
\]

(b) Macro definition (body)

\[
Z := \text{ABSDIFF}(X, Y)
\]

(c) Macro invocation

\[
Z = (X) > (Y) \ ? (X) - (Y) : (Y) - (X)
\]

• Macro expansion
Examples of ELENA macro definition and invocation (Fig. 4.14) (Cont.)

\[
\begin{align*}
\text{MOV} & \quad \text{EAX}, \%2 \\
\text{SUB} & \quad \text{EAX}, \%3 \\
\text{JNS} & \quad \&\text{STOR} \\
\text{NEG} & \quad \text{EAX} \\
\&\text{STOR} & \quad \text{MOV} \quad \text{EAX}, \%1 \\
\end{align*}
\]

\(Z := \text{ABSDIFF}(X, Y)\)

\[
\begin{align*}
\text{MOV} & \quad \text{EAX}, X \\
\text{SUB} & \quad \text{EAX}, Y \\
\text{JNS} & \quad \text{STOR0001} \\
\text{NEG} & \quad \text{EAX} \\
\text{STOR0001} & \quad \text{MOV} \quad \text{EAX}, Z \\
\end{align*}
\]

**Figure 4.14** Examples of ELENA macro definition and invocation.
Examples of ELENA macro-time instructions (Fig. 4.15)

ADD %1 TO THE FIRST %2 ELEMENTS OF V

(a)

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

(b)

ADD 5 TO THE FIRST 3 ELEMENTS OF V

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

(c)

Figure 4.15 Example of ELENA macro-time instructions.
The ELENA Macro Processor (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
  - ELENA selects the header with the fewest parameters if there are two or more matching headers with the same number of parameters, the most recently defined macro is selected.

- **Examples:**
  - **Macro definition:**
    - ADD %1 TO %2
    - ADD %1 TO THE FIRST ELEMENT OF %2
  - **Macro invocation:**
    - DISPLAY TABLE for DISPLAY %1 or %1 TABLE
    - A=B+1 for %1=%2+%3 or %1=%2+1