Chapter 4
Macro Processors
Chapter 4: Macro Processors

- 4.1 Basic Macro Processors Functions
- 4.2 Machine-Independent Macro Processors Features
- 4.3 Macro Processors Design Options
- 4.4 Implementation Examples
Introduction to Macro Processors

- A *macro instruction* (*macro*) is a notational convenience for the programmer.
  - Allow the programmer to write a **shorthand** version of a program

- A *macro* represents a commonly used group of statements in the source programming language.

- *Expanding* the macros
  - The **macro processor** replaces each macro instruction with the corresponding group of source language statements.
A macro processor
- Essentially involve the substitution of one group of characters or lines for another.
- Normally, it performs no analysis of the text it handles.
- It doesn’t concern the meaning of the involved statements during macro expansion.

The design of a macro processor generally is machine independent.
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, which is not tied to any particular language
C uses a **macro preprocessor** to support language extensions, such as named constants, expressions, and file inclusion.

```c
#define max(a,b) ((a<b)?(a):(b))
#define MACBUF 4
#include <stdio.h>
```
4.1 Basic Macro Processors Functions

- *Macro processor* should processes the
  - Macro definitions
    - Define macro name, group of instructions
  - Macro invocation (macro calls)
    - A body is simply copied or substituted at the point of call
  - Expansion with substitution of parameters
    - *Arguments* are textually substituted for the *parameters*
    - The resulting procedure body is textually substituted for the call
Macro Definition

- Two new assembler directives are used in macro definition:
  - **MACRO**: identify the beginning of a macro definition
  - **MEND**: identify the end of a macro definition

- **label**  **op**  **operands**
  - **name**  **MACRO**  **parameters**
  - :
    - **body**
  - : **MEND**

- **Parameters**: the entries in the operand field identify the **parameters** of the macro instruction
  - We require each parameter begins with ‘&’

- **Body**: the statements that will be generated as the expansion of the macro.

- **Prototype** for the macro:
  - The **macro name** and **parameters** define a pattern or **prototype** for the macro instructions used by the programmer
Fig 4.1: Macro Definition

Macro definition

`COPY START 0 COPY FILE FROM INPUT TO OUTPUT`

`RDUFF MACRO &INDEV, &BUFADR, &RECLTH`

MACRO TO READ RECORD INTO BUFFER

CLEAR X CLEAR A

CLEAR S

+LDT #4096 SET MAXIMUM RECORD LENGTH

TD =X'&INDEV' TEST INPUT DEVICE

JEQ *=3 LOOP UNTIL READY

RD =X'&INDEV' READ CHARACTER INTO REG A

COMPR A S TEST FOR END OF RECORD

JEQ *=11 EXIT LOOP IF EOF

STCH &BUFADR, X STORE CHARACTER IN BUFFER

TIMT T LOOP UNLESS MAXIMUM LENGTH

JLT *=19 HAS BEEN REACHED

STX &RECLTH SAVE RECORD LENGTH

MEND

Macro body contains no label
Fig 4.1: Macro Definition (Cont.)

- Macro definition

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>WRBUFF 0</td>
<td>Macro definition</td>
</tr>
<tr>
<td>105</td>
<td></td>
<td>Macro body contains no label</td>
</tr>
<tr>
<td>110</td>
<td>.</td>
<td>Macro to write record from buffer</td>
</tr>
<tr>
<td>115</td>
<td>.</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>120</td>
<td>CLEAR X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>125</td>
<td>LDT &amp;RECLTH</td>
<td>GET CHARACTER FROM BUFFER</td>
</tr>
<tr>
<td>130</td>
<td>LDCH &amp;BUFADR,X</td>
<td>TEST OUTPUT DEVICE</td>
</tr>
<tr>
<td>135</td>
<td>TD =X'&amp;OUTDEV'</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>140</td>
<td>JEQ *-3</td>
<td>WRITE CHARACTER</td>
</tr>
<tr>
<td>145</td>
<td>WD =X'&amp;OUTDEV'</td>
<td>LOOP UNTIL ALL CHARACTERS</td>
</tr>
<tr>
<td>150</td>
<td>TIXR T</td>
<td>HAVE BEEN WRITTEN</td>
</tr>
<tr>
<td>155</td>
<td>JLT *-14</td>
<td>MEND</td>
</tr>
<tr>
<td>160</td>
<td>MEND</td>
<td>MEND</td>
</tr>
</tbody>
</table>
Macro Invocation

- A **macro invocation statement** (a **macro call**) gives the **name** of the macro instruction being invoked and the **arguments** in expanding the macro.

- Macro Invocation vs. Subroutine Call.
  - Statements of the macro body are expanded **each time** the macro is invoked.
  - Statements of the subroutine appear **only one**, regardless of how many times the subroutine is called.
  - Macro invocation is more efficient than subroutine call, however, the code size is larger
Fig 4.1: Macro Invocation

- Macro invocation

Listing:

165  .    MAIN PROGRAM
170  .
180  FIRST           STL    RETADR     SAVE RETURN ADDRESS
190  CLOOP           RDREFF    FR1, 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  WRREFF    05, BUFFER, LENGTH WRITE OUTPUT RECORD
215  J    CLOOP     LOOP
220  ENDFIL        WRREFF    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 SlC/XE program.
Macro Expansion

- Each macro invocation statement will be expanded into the statements that form the **body** of the macro.

- Arguments from the macro invocation are **substituted** for the parameters in the macro prototype.
  - The arguments and parameters are associated with one another according to their **positions**.
  - The first argument in the macro invocation corresponds to the first parameter in the macro prototype, etc.
Macro Expansion

Source program

WD MACRO
STA  DATA1
STB  DATA2
MEND

Expanded source program

.  
.  
.  
STB  DATA2
STA  DATA1
STA  DATA1
STB  DATA2
.  
.  
.  

Macro definition

Macro Expansion by Macro processor

Macro invocation
Macro Expansion with Parameters Substitution

**Source program**

```plaintext
WD  MACRO &A1,&A2
    STA  &A1
    STB  &A2
    MEND
.
WD  DATA1,DATA2
.
WD  DATA3,DATA4
.
WD  DATA5,DATA6
.
```

**Expanded source program**

```plaintext
.
.
STA  DATA1
STB  DATA2
.
STA  DATA3
STB  DATA4
.
STA  DATA5
STB  DATA6
.
```

**Macro definition**

**Macro Expansion by Macro processor**

**Macro invocation**
Program From Fig. 4.1 with Macros Expanded (fig. 4.2)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>COPY</td>
<td>COPY FILE FROM INPUT TO OUTPUT</td>
</tr>
<tr>
<td>180</td>
<td>FIRST STL</td>
<td>SAVE RETURN ADDRESS</td>
</tr>
<tr>
<td>190</td>
<td>.CLOOP</td>
<td>READ RECORD INTO BUFFER</td>
</tr>
<tr>
<td>190a</td>
<td>CLOOP CLEAR</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>190b</td>
<td>CLEAR</td>
<td>X</td>
</tr>
<tr>
<td>190c</td>
<td>CLEAR</td>
<td>A</td>
</tr>
<tr>
<td>190d</td>
<td>+LDT</td>
<td>S</td>
</tr>
<tr>
<td>190e</td>
<td>TD =X'F1'</td>
<td>SET MAXIMUM RECORD LENGTH</td>
</tr>
<tr>
<td>190f</td>
<td>JEQ *-3</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>190g</td>
<td>RD =X'F1'</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>190h</td>
<td>CMPR A,S</td>
<td>READ CHARACTER INTO REG A</td>
</tr>
<tr>
<td>190i</td>
<td>JEQ *+11</td>
<td>TEST FOR END OF RECORD</td>
</tr>
<tr>
<td>190j</td>
<td>STCH BUFFER,X</td>
<td>EXIT LOOP IF EOR</td>
</tr>
<tr>
<td>190k</td>
<td>TXR T</td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>190l</td>
<td>JLT *-19</td>
<td>LOOP UNLESS MAXIMUM LENGTH</td>
</tr>
<tr>
<td>190m</td>
<td>STX LENGTH</td>
<td>HAS BEEN REACHED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAVE RECORD LENGTH</td>
</tr>
</tbody>
</table>
Program From Fig. 4.1 with Macros Expanded (fig. 4.2)(Cont.)

- Macro expansion

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>195</td>
<td>LDA</td>
<td>LENGTH</td>
<td>TEST FOR END OF FILE</td>
</tr>
<tr>
<td>200</td>
<td>COMP</td>
<td>#0</td>
<td>EXIT IF EOF FOUND</td>
</tr>
<tr>
<td>205</td>
<td>JEQ</td>
<td>ENDFIL</td>
<td>WRITE OUTPUT RECORD</td>
</tr>
<tr>
<td>210</td>
<td>WRBUFB</td>
<td>05, BUFFER, LENGTH</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>210a</td>
<td>CLEAR</td>
<td>X</td>
<td>GET CHARACTER FROM BUFFER</td>
</tr>
<tr>
<td>210b</td>
<td>LUT</td>
<td>LENGTH</td>
<td>TEST OUTPUT DEVICE</td>
</tr>
<tr>
<td>210c</td>
<td>LDCH</td>
<td>BUFFER, X</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>210d</td>
<td>JEQ</td>
<td>*-3</td>
<td>WRITE CHARACTER</td>
</tr>
<tr>
<td>210e</td>
<td>MD</td>
<td>=X’05’</td>
<td>LOOP UNTIL ALL CHARACTERS</td>
</tr>
<tr>
<td>210f</td>
<td>TIXR</td>
<td>T</td>
<td>HAVE BEEN WRITTEN</td>
</tr>
<tr>
<td>210g</td>
<td>JLT</td>
<td>*-14</td>
<td></td>
</tr>
</tbody>
</table>
Program From Fig. 4.1 with Macros Expanded (fig. 4.2)(Cont.)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Instruction 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>215</td>
<td></td>
<td>S</td>
<td>CLOOP</td>
<td>LOOP</td>
</tr>
<tr>
<td>220</td>
<td>.ENDFIL</td>
<td>WRBUFF</td>
<td>05, EOF, THREE</td>
<td>INSERT EOF MARKER</td>
</tr>
<tr>
<td>220a</td>
<td>ENDFIL</td>
<td>CLEAR</td>
<td>X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>220b</td>
<td></td>
<td>LDT</td>
<td>THREE</td>
<td>GET CHARACTER FROM BUFFER</td>
</tr>
<tr>
<td>220c</td>
<td></td>
<td>LDCH</td>
<td>EOF, X</td>
<td>TEST OUTPUT DEVICE</td>
</tr>
<tr>
<td>220d</td>
<td></td>
<td>TD</td>
<td>=X’05’</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>220e</td>
<td></td>
<td>JEQ</td>
<td>*-3</td>
<td>WRITE CHARACTER</td>
</tr>
<tr>
<td>220f</td>
<td></td>
<td>WD</td>
<td>=X’05’</td>
<td>LOOP UNTIL ALL CHARACTERS</td>
</tr>
<tr>
<td>220g</td>
<td></td>
<td>TIXR</td>
<td>T</td>
<td>HAVE BEEN WRITTEN</td>
</tr>
<tr>
<td>220h</td>
<td></td>
<td>JLT</td>
<td>*-14</td>
<td>ØRETTADR</td>
</tr>
<tr>
<td>230</td>
<td></td>
<td>EOF</td>
<td>BYTE</td>
<td>C’EOF’</td>
</tr>
<tr>
<td>235</td>
<td></td>
<td>THREE</td>
<td>WORD</td>
<td>3</td>
</tr>
<tr>
<td>240</td>
<td></td>
<td>RETADR</td>
<td>RESW</td>
<td>1</td>
</tr>
<tr>
<td>245</td>
<td></td>
<td>LENGTH</td>
<td>RESW</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>BUFFER</td>
<td>RESB</td>
<td>4096</td>
</tr>
<tr>
<td>255</td>
<td></td>
<td>END</td>
<td>FIRST</td>
<td>4096-BYTE BUFFER AREA</td>
</tr>
</tbody>
</table>

**Figure 4.2** Program from Fig. 4.1 with macros expanded.
No Label in the Body of Macro

Problem of the label in the body of macro:
- If the same macro is expanded multiple times at different places in the program.
  - There will be duplicate labels, which will be treated as errors by the assembler.

Solutions:
- Simply not to use labels in the body of macro.
- Explicitly use PC-relative addressing instead.
  - For example, in RDBUFF and WRBUFF macros,
    - JEQ \* +11
    - JLT \* -14
  - It is inconvenient and error-prone.
- Other better solution?
  - Mentioned in Section 4.2.2.
4.1.2 Macro Processors Algorithm and Data Structures

- Two-pass macro processor

- One-pass macro processor
Two-pass macro processor

- Two-pass macro processor
  - Pass1: process all *macro definitions*
  - Pass2: expand all *macro invocation* statements

- Problem
  - Does not allow *nested macro definitions*
  - Nested macro definitions
    - The body of a macro contains definitions of other macros
    - Because all macros would *have to be defined during the first pass* before any macro invocations were expanded

- Solution
  - One-pass macro processor
Nested Macros Definition

- **MACROS** (for SIC)
  - contains the definitions of **RDBUFF** and **WRBUFF** written in SIC instructions.

- **MACROX** (for SIC/XE)
  - contains the definitions of **RDBUFF** and **WRBUFF** written in SIC/XE instructions.

- Example 4.3
Macro Definition within a Macro Body
(Figure 4.3(a))

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]
Macro Definition within a Macro Body
(Figure 4.3(b))

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

7                   .    {End of WRBUFF}
8                   .    {End of WRBUFF}
Nested Macros Definition (Cont.)

- A program that is to be run on SIC system could invoke MACROS whereas a program to be run on SIC/XE can invoke MACROX.

- Defining MACROX does not define RDBUFF and WRBUFF.
  - These definitions are processed only when an invocation of MACROX is expanded.
One-pass macro processor

- One-pass macro processor
  - Every macro must be defined before it is called
  - One-pass processor can alternate between macro definition and macro expansion
  - Nested macro definitions are allowed
Three Main Data Structures

- **DEFTAB**
  - A *definition table* used to *store macro definition* including
    - macro prototype
    - macro body
  - Comment lines are omitted.
  - *Positional notation* has been used for the parameters for efficiency in substituting arguments.
    - E.g. the first parameter &INDEV has been converted to ?1 (indicating the first parameter in the prototype)

- **NAMTAB**
  - A *name table* used to *store the macro names*
  - Serves as an index to DEFTAB
    - Pointers to the beginning and the end of the macro definition

- **ARGTAB**
  - A *argument table* used to store the arguments used in the expansion of macro invocation
  - As the macro is expanded, arguments are substituted for the corresponding parameters in the macro body.
Figure 4.4 Contents of macro processor tables for the program in Fig. 4.1: (a) entries in NAMTAB and DEFTAB defining macro RDBUFF, (b) entries in ARGTAB for invocation of RDBUFF on line 190.
One-Pass Macro Processor

- Procedures
  - Macro definition: DEFINE
  - Macro invocation: EXPAND
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE ≠ 'END' do
        begin
            GETLINE
            PROCESSSLINE
        end {while}
end {macro processor}

procedure PROCESSSLINE
begin
    search NAMTAB for OPCODE
    if found then
        EXPAND
    else if OPCODE = 'MACRO' then
        DEFINE
    else write source line to expanded file
end {PROCESSSLINE}

Figure 4.5 Algorithm for a one-pass macro processor.
One-Pass Macro Processor Allows Nested Macro Definition

- Sub-procedure DEFINE should handle the nested macro definition
  - Maintains a counter named **LEVEL**
  - Each time a MACRO directive is read, the value of **LEVEL** is increased by 1
  - Each time an MEND directive is read, the value of **LEVEL** is decreased by 1
Algorithm for one-pass macro processor (Fig. 4.5)

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)