Chapter 4 Macro Processors
-- Basic Macro Processor Functions
Introduction

- A macro instruction (macro) is a notational convenience for the programmer
  - It allows the programmer to write shorthand version of a program (module programming)
- The macro processor replaces each macro instruction with the corresponding group of source language statements (expanding)
  - Normally, it performs no analysis of the text it handles.
  - It does not concern the meaning of the involved statements during macro expansion.
- The design of a macro processor generally is machine independent!
Basic macro processor functions

- 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
- Prototype for the macro
  - Each parameter begins with ‘&’

```
name   MACRO   parameters
   :
body
   :
MEND
```
- Body: the statements that will be generated as the expansion of the macro.
## Macro Expansion

<table>
<thead>
<tr>
<th>Source</th>
<th>Expanded source</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 MACRO &amp;D1, &amp;D2</td>
<td></td>
</tr>
<tr>
<td>STA &amp;D1</td>
<td>STA DATA1</td>
</tr>
<tr>
<td>STB &amp;D2</td>
<td>STB DATA2</td>
</tr>
<tr>
<td>MEND</td>
<td></td>
</tr>
<tr>
<td>M1 DATA1, DATA2</td>
<td></td>
</tr>
<tr>
<td>M1 DATA4, DATA3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA DATA4</td>
</tr>
<tr>
<td></td>
<td>STB DATA3</td>
</tr>
</tbody>
</table>

*System Programming*
Example of macro definition
Figure 4.1, pp. 178

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 RECORD
90 STX &RECLTH SAVE RECORD LENGTH
95 MEND

System Programming
Macro invocation

- A macro invocation statement (a macro call) gives the name of the macro instruction being invoked and the arguments to be used in expanding the macro.
  - `macro_name p1, p2, ...`

- Difference between macro call and procedure call
  - Macro call: statements of the macro body are expanded each time the macro is invoked.
  - Procedure call: statements of the subroutine appear only one, regardless of how many times the subroutine is called.

- Question
  - How does a programmer decide to use macro calls or procedure calls?
    - From the viewpoint of a programmer
    - From the viewpoint of the CPU
Exchange the values of two variables

```c
void exchange(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

main() {
    int i=1, j=3;
    printf("BEFORE - %d %d\n", i, j);
    exchange(i, j);
    printf("AFTER - %d %d\n", i, j);
}
```

What’s the result?
Pass by Reference

```c
void exchange(int *p1, int *p2) {
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

main() {
    int i=1, j=3;
    printf("BEFORE - %d %d\n", i, j);
    exchange(&i, &j);
    printf("AFTER - %d %d\n", i, j);
}
```

# 12 Lines of Assembly Code

## . Subroutine EXCH

<table>
<thead>
<tr>
<th>EXCH</th>
<th>LDA</th>
<th>@P1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STA</td>
<td>TEMP</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>@P2</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>@P1</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>TEMP</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>@P2</td>
</tr>
<tr>
<td></td>
<td>RSUB</td>
<td></td>
</tr>
</tbody>
</table>

- P1: RESW 1
- P2: RESW 1
- TEMP: RESW 1

## MAIN

<table>
<thead>
<tr>
<th>MAIN</th>
<th>LDA</th>
<th>#1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STA</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>#3</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>J</td>
</tr>
</tbody>
</table>

. Call a subroutine

<table>
<thead>
<tr>
<th></th>
<th>LDA</th>
<th>#I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STA</td>
<td>P1</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>#J</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td>JSUB</td>
<td>EXCH</td>
</tr>
</tbody>
</table>

- I: RESW 1
- J: RESW 1

-END-

*System Programming*
Swap two variables by macro

```c
#define swap(i,j) { int temp; temp=i; i=j; j=temp; }

main() {
    int i=1, j=3;
    printf("BEFORE - %d %d\n", i, j);
    swap(i,j);
    printf("AFTER - %d %d\n", i, j);
}
```

6 Lines of Assembly Code

MAIN       LDA      #1
STA         I
LDA          #3
STA          J

. Invoke a macro

LDA         I
STA          TEMP
LDA          J
STA          I
LDA          TEMP
STA          J

I          RESW      1
J          RESW      1
TEMP        RESW      1
END         MAIN
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 (according to their positions).
  - In the definition of macro: parameter
  - In the macro invocation: argument
- Comment lines within the macro body will be deleted.
- Macro invocation statement itself has been included as a comment line.
- The label on the macro invocation statement has been retained as a label on the first statement generated in the macro expansion.
  - We can use a macro instruction in exactly the same way as an assembler language mnemonic.
Example of macro invocation

Figure 4.1, pp. 178

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

System Programming
Example of macro expansion
Figure 4.2, pp. 179

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>COPY START 0</td>
<td>COPY FILE FROM INPUT TO OUTPUT</td>
</tr>
<tr>
<td>180</td>
<td>FIRST STL RETADR</td>
<td>SAVE RETURN ADDRESS</td>
</tr>
<tr>
<td>190</td>
<td>.CLOOP RDBUFF F1,BUFFER,LENGTH</td>
<td>READ RECORD INTO BUFFER</td>
</tr>
<tr>
<td>190a</td>
<td>CLOOP CLEAR X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>190b</td>
<td>CLEAR A</td>
<td></td>
</tr>
<tr>
<td>190c</td>
<td>CLEAR S</td>
<td></td>
</tr>
<tr>
<td>190d</td>
<td>+LDT #4096</td>
<td>SET MAXIMUM RECORD LENGTH</td>
</tr>
<tr>
<td>190e</td>
<td>TD =X’F1’</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td>190f</td>
<td>JEQ *-3</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>190g</td>
<td>RD =X’F1’</td>
<td>TEST FOR END OF RECORD</td>
</tr>
<tr>
<td>190h</td>
<td>COMPR A, S</td>
<td>TEST FOR END OF RECORD</td>
</tr>
<tr>
<td>190i</td>
<td>JEQ *+11</td>
<td>EXIT LOOP IF EOR</td>
</tr>
<tr>
<td>190j</td>
<td>STCH BUFFER, X</td>
<td>STORE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>190k</td>
<td>TIXR T</td>
<td>LOOP UNLESS MAXIMUM LENGTH</td>
</tr>
<tr>
<td>190l</td>
<td>JLT *-19</td>
<td>HAS BEEN REACHED</td>
</tr>
<tr>
<td>190M</td>
<td>STX LENGTH</td>
<td>SAVE RECORD LENGTH</td>
</tr>
</tbody>
</table>
### Example of macro expansion

**Figure 4.2, pp. 179**

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>195</td>
<td>LDA LENGTH</td>
<td>TEST FOR END OF FILE</td>
</tr>
<tr>
<td>200</td>
<td>COMP #0</td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>JEQ ENDFIL</td>
<td>EXIT IF EOF FOUND</td>
</tr>
<tr>
<td>210</td>
<td>WRBUFF 05,BUFFER,LENGTH</td>
<td>WRITE OUTPUT RECORD</td>
</tr>
<tr>
<td>210a</td>
<td>CLEAR X</td>
<td>CLEAR LOOP COUNTER</td>
</tr>
<tr>
<td>210b</td>
<td>LDT LENGTH</td>
<td></td>
</tr>
<tr>
<td>210c</td>
<td>LDCH BUFFER,X</td>
<td>GET CHARACTER FROM BUFFER</td>
</tr>
<tr>
<td>210d</td>
<td>TD =X’05’</td>
<td>TEST OUTPUT DEVICE</td>
</tr>
<tr>
<td>210e</td>
<td>JEQ #-3</td>
<td>LOOP UNTIL READY</td>
</tr>
<tr>
<td>210f</td>
<td>WD =X’05’</td>
<td>WRITE CHARACTER</td>
</tr>
<tr>
<td>210g</td>
<td>TIXR T</td>
<td>LOOP UNTIL ALL CHARACTERS</td>
</tr>
<tr>
<td>210h</td>
<td>JLT #-14</td>
<td>HAVE BEEN WRITTEN</td>
</tr>
<tr>
<td>215</td>
<td>J CLOOP</td>
<td>LOOP</td>
</tr>
<tr>
<td>220</td>
<td>.ENDFIL WRBUFF 05,EOF,THREE</td>
<td>INSERT EOF MARKER</td>
</tr>
</tbody>
</table>
Example of macro expansion
Figure 4.2, pp. 179

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>220a</td>
<td>ENDFIL</td>
<td>CLEAR X</td>
</tr>
<tr>
<td>220b</td>
<td>LDT</td>
<td>THREE</td>
</tr>
<tr>
<td>220c</td>
<td>LDCH</td>
<td>EOF,X</td>
</tr>
<tr>
<td>220d</td>
<td>TD</td>
<td>=X‘05’</td>
</tr>
<tr>
<td>220e</td>
<td>JEQ</td>
<td>-*3</td>
</tr>
<tr>
<td>220f</td>
<td>WD</td>
<td>=X‘05’</td>
</tr>
<tr>
<td>220g</td>
<td>TIXR</td>
<td>T</td>
</tr>
<tr>
<td>220h</td>
<td>JLT</td>
<td>-*14</td>
</tr>
<tr>
<td>225</td>
<td>J</td>
<td>@RETADR</td>
</tr>
<tr>
<td>230</td>
<td>EOF</td>
<td>BYTE C‘EOF’</td>
</tr>
<tr>
<td>235</td>
<td>THREE</td>
<td>WORD 3</td>
</tr>
<tr>
<td>240</td>
<td>RETADR</td>
<td>RESW 1</td>
</tr>
<tr>
<td>245</td>
<td>LENGTH</td>
<td>RESW 1</td>
</tr>
<tr>
<td>250</td>
<td>BUFFER</td>
<td>RESB 4096</td>
</tr>
<tr>
<td>255</td>
<td>END</td>
<td>FIRST</td>
</tr>
</tbody>
</table>

CLEAR LOOP COUNTER
GET CHARACTER FROM BUFFER
TEST OUTPUT DEVICE
LOOP UNTIL READY
WRITE CHARACTER
LOOP UNTIL ALL CHARACTERS
HAVE BEEN WRITTEN

System Programming

16
No label in the macro body

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:
- Do not use labels in the body of macro.
- Explicitly use PC-relative addressing instead.
  - Ex, in RDBUFF and WRBUFF macros,
    - JEQ  *+11
    - JLT  *-14
  - It is inconvenient and error-prone.
- The way of avoiding such error-prone method will be discussed in Section 4.2.2
Two-pass macro processor

- You may design a two-pass macro processor
  - Pass 1:
    - Process all macro definitions
  - Pass 2:
    - Expand all macro invocation statements

- However, one-pass may be enough
  - Because all macros would have to be defined during the first pass before any macro invocations were expanded.
    - The definition of a macro must appear before any statements that invoke that macro.
  - Moreover, the body of one macro can contain definitions of other macros.
Example of recursive macro definition
Figure 4.3, pp.182

MACROS (for SIC)

Contains the definitions of RDBUFF and WRBUFF written in SIC instructions.

1  MACROS  MACOR  {Defines SIC standard version macros}
2  RDBUFF  MACRO  &INDEV,&BUFADR,&RECLTH
3  MEND
4  WRBUFF  MACRO  &OUTDEV,&BUFADR,&RECLTH
5  MEND  {End of WRBUFF}
6  MEND  {End of MACROS}

System Programming
Example of recursive macro definition
Figure 4.3, pp.182

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

<table>
<thead>
<tr>
<th>Line</th>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MACROX</td>
<td>MACRO {Defines SIC/XE macros}</td>
</tr>
<tr>
<td>2</td>
<td>RDBUFF</td>
<td>MACRO &amp;INDEV,&amp;BUFADR,&amp;RECLTH {SIC/XE version}</td>
</tr>
<tr>
<td>3</td>
<td>MEND</td>
<td>{End of RDBUFF}</td>
</tr>
<tr>
<td>4</td>
<td>WRBUFF</td>
<td>MACRO &amp;OUTDEV,&amp;BUFADR,&amp;RECLTH {SIC/XE version}</td>
</tr>
<tr>
<td>5</td>
<td>MEND</td>
<td>{End of WRBUFF}</td>
</tr>
</tbody>
</table>

*System Programming*
Example of macro definitions

- 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.

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

- A one-pass macro processor that alternate between \textit{macro definition} and \textit{macro expansion} in a recursive way is able to handle recursive macro definition.

- Restriction
  - The definition of a macro must appear in the source program before any statements that invoke that macro.
  - This restriction does not create any real inconvenience.
Data structures for one-pass macro processor

- **DEFTAB** (definition table)
  - Stores the macro definition including *macro prototype* and *macro body*
  - Comment lines are omitted.
  - References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.

- **NAMTAB**
  - Stores macro names
  - Serves as an index to DEFTAB
    - Pointers to the *beginning* and the *end* of the macro definition (DEFTAB)

- **ARGTAB**
  - Stores the arguments of macro invocation according to their positions in the argument list
  - As the macro is expanded, arguments from ARGTAB are substituted for the corresponding parameters in the macro body.
Data structures
Algorithm

**Procedure GETLINE**

If EXPANDING then
get the next line to be processed from DEFTAB
Else
read next line from input file

**MAIN program**
- Iterations of
  - GETLINE
  - PROCESSLINE

**Procedure EXPAND**
Set up the argument values in ARGTAB
Expand a macro invocation statement (like in MAIN procedure)
- Iterations of
  - GETLINE
  - PROCESSLINE

**Procedure PROCESSLINE**
- DEFINE
- EXPAND
- Output source line

**Procedure DEFINE**
Make appropriate entries in DEFTAB and NAMTAB
Algorithm
Figure 4.5, pp. 184

begin \{macro processor\}
  EXPANDINF := FALSE
  while OPCODE \neq \textquoteleft EN D\textquoteright \ do
    begin
      GETLINE
      PROCESSLINE
    end \{while\}
  end \{macro processor\}

Procedure PROCESSLINE
begin
  search MAMTAB for OPCODE
  if found then
    EXPAND
  else if OPCODE = \textquoteleft MACRO\textquoteright then
    DEFINE
  else
    write source line to expanded file
end \{PROCESSOR\}
Algorithm
Figure 4.5, pp. 185

Procedure DEFINE
begin
begin
enter macro name into NAMTAB
enter macro prototype into DEFTAB
LEVEL := 1
while LEVEL > 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}

System Programming
Algorithm
Figure 4.5, pp. 185

Procedure EXPAND
begin
  EXPANDING := TRUE
  get first line of macro definition {prototype} from DEFTAB
  set up arguments from macro invocation in ARGTAB
  while 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}
Handling nested macro definition

- In DEFINE procedure
  - When a macro definition is being entered into DEFTAB, the normal approach is to continue until an MEND directive is reached.
  - This would not work for nested macro definition because the first MEND encountered in the inner macro will terminate the whole macro definition process.
  - To solve this problem, a counter LEVEL is used to keep track of the level of macro definitions.
    - Increase LEVEL by 1 each time a MACRO directive is read.
    - Decrease LEVEL by 1 each time a MEND directive is read.
    - A MEND terminates the whole macro definition process when LEVEL reaches 0.
    - This process is very much like matching left and right parentheses when scanning an arithmetic expression.
Comparison of Macro Processors Design

- One-pass algorithm
  - 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 but nested calls are not

- Two-pass algorithm
  - Pass1: Recognize macro definitions
  - Pass2: Recognize macro calls
  - Nested macro definitions are not allowed