Macros & Macro Processors (continued)

Ref: Chapter 4 of [Beck].

Nested macro calls:

Example:

```
ADDVAL   MACRO    &X
         LDA      DEPOT
         ADD      &X
         STA      DEPOT
         MEND

ADD3     MACRO    &P,&Q,&R
         ADDVAL   &P
         ADDVAL   &Q
         ADDVAL   &R
         MEND
```

Reading assignment: Figure 4.11 on page 200 of [Beck].

- Nested calls much more common than nested definitions.
- The nesting level can be larger than 1.
- Macro processor that handles nested calls can be implemented using recursion.
  (See Handout 10.1).

Parameter concatenation:

Example:

```
SUM      MACRO    &P,&Q,&R
         CLEAR    A
         LDA      &P
         ADD      &Q
         ADD      &R
         MEND
```

- Consider the following calls to the above macro:

```
SUM      XA1,XA2,XA3
SUM      XB1,XB2,XB3
SUM      XC1,XC2,XC3
```

- Parameters in calls differ only by one character.
- Calls can be simplified if parameters can be "parts" of labels.

Example: The SUM macro can be redefined as follows (using notation from [Beck]).

```
SUM     MACRO   &L
        CLEAR   A
        LDA     X&L->1
        ADD     X&L->2
        ADD     X&L->3
        MEND
```

- The symbols '&' and '->' mark the extent of the dummy parameter.
- Advantage: The calls can now be simplified:

```
SUM     A
SUM     B
SUM     C
```


- Disadvantage: Program is harder to understand.

Generation of unique labels:

- Labels are generally avoided in macro bodies.
- LC-relative addressing used to avoid using labels.
- Use of LC-relative addressing makes modifications cumbersome.
- This difficulty can be overcome with help from the macro processor.
- Use special labels in macros (e.g. labels starting with '$').

Example: See Handout 10.2.

- In each call, the macro processor generates a unique prefix to be added to each label.
- Note: In SIC/XE programs, avoid using labels starting with '$', except for macro bodies.

Conditional macro expansion:

- Recall: Conditional macro facility in C.
- Sometimes called "conditional assembly".
- SIC/XE assembly language provides the following forms of conditional expansion.

```
IF      condition
  .
  .
ENDIF

IF      condition
  .
  .
ELSE
  .
  .
ENDIF
```

- `IF`, `ELSE`, `ENDIF`: New pseudo-ops.
- *condition*: Boolean condition that can be evaluated at the time of macro expansion.

Examples: See Handout 10.3.

Note: To check whether or not an actual parameter is specified, compare dummy parameter with the null character.

Example:

```
MAC       MACRO    &X,&Y,&Z
          IF       (&Y  EQ  '')
          .
          .
          ENDIF
          .
          .
          MEND
```

When the above macro is invoked as

```
MAC   L1,,L3
```

the lines between `IF` and `ENDIF` will be included in the expansion.

Macro-time variables:

- Any symbol whose name begins with & and which is not a dummy parameter is macro-time variable.
- All such variables are initialized to 0.
- The SET pseudo-op can be used to assign a value to such a variable. (The operand of the pseudo-op may be another expression that can be evaluated at macro expansion time.)
- Typical use: To remember the result of an expression evaluated by the macro processor.

Example: See Handout 10.4.

Implementing conditional expansion:

- No major changes needed to read in a macro body.
- In the expansion phase, Boolean expressions need to be evaluated.
- Nested IF statements can be handled using a stack.

- Macro-time variables can be stored along with their values in a symbol table. (Entries in the table are modified when SET pseudo-ops are processed.)

Keyword macro parameters:

- Alternative to specifying parameters by position.

Example: Consider the macro DOIO.

```
MACRO   DOIO   &DEV,&BUF,&CODE
          .
          .
        MEND
```

The macro can be invoked as follows:

```
      DOIO    A7,LOC,R
```

It can also be invoked as follows:

```
     DOIO    DEV=A7,BUF=LOC,CODE=R
```

- The second form is convenient if it is hard to remember the order of the parameters.
- The second form is also useful when the macro has a large number of parameters and each call needs to specifies only a few. (The other parameters are assumed to be omitted.)
- Implementing keyword parameters is straightforward: macro processor can set up the argument table from the call.