EE 354 **September 1, 2017**
Notes on MACROs

A macro is a subprogram but unlike functions and subroutines, a macro is physically inserted into your program at the point where it is activated. As such, macros are generally faster because they have no call/return statements but they also take up much more memory space because they are inserted into your program every time they are activated.

Macros must be defined before they can be used so they typically go at the top of a program. A macro definition begins with the word MACRO which is an assembler directive . The macro directive declares the name of the macro and lists the formal parameters. The macro definition must end with the directive ENDM for end macro. The code, or text, between the MACRO and ENDM directives is called the macro body.

As an example, a macro named Add10 which adds 10 to a register might look like this:

```
Add10  MACRO x
        mov a, x
        add a, #10
        mov x, a
        ENDM
```

In this macro, Add10 is the macro name and x is a formal parameter which stands for a register. The macro moves the register into the accumulator, adds 10, and moves the accumulator back to the register. To use the macro a main program might look like this:

```
MainSeg SEGMENT CODE
 Add10  MACRO x
        mov a, x
        add a, #10
        mov x, a
        ENDM

 CSEG at 0
    ljmp Start

 RSEG MainSeg
 Start: mov r1, #3
        Add10 r1
        mov r2, #5
        Add10 r2
 Last:  sjmp Last
 END
```

When this program is assembled the assembler generates the code for the macro first and then inserts that code into the main program everywhere that the name of the macro is used.

The main program really looks like this:

```
Start: mov r1, #3
          mov a, r1    ;macro expansion
          add a, #10
          mov r1, a
        mov r2, #5
          mov a, r2    ;macro expansion
          add a, #10
          mov r2, a
 Last:  sjmp Last
```

Macros on the 8051 using the Keil assembler support up to 16 parameters. Parameters can stand for register names or for constants or memory locations.

Occasionally you may want to put a loop in a macro in which case you will need a loop label. This could cause a problem if the macro is used more than once since the label would be repeated. To fix this, macros support a "LOCAL" directive. Any label declared local will be known only within the macro and will be changed by the assembler each time the macro is used.

A macro with a local label might look like this:

```
LoopEx MACRO x
LOCAL LP
       mov r7, #x
    LP:djnz r7, LP
       ENDM
```

In this example, LP is a label which will be changed by the assembler each time the macro is invoked. The Keil assembler allows up to 16 local labels in macros and the LOCAL declaration must immediately follow the MACRO declaration.