

## Clarifications

### Right Shift (>>) on Signed operands

The book uses the HI-TECH PICC18 compiler convention that a right shift on a signed quantity preserves the sign bit. Thus, the new value of  $k$  in the code below is 0xC0.

```
signed char k;  
  
k = 0x80;  
k = k >> 1;           ;PICC18 preserves sign bit, new value is 0xC0
```

However, it should be mentioned that ANSI C states that this operation is *compiler dependent*. For example, the Microchip MCC18 compiler always shifts in a '0' for a right shift operation regardless of whether it is a signed or unsigned operand. The message is that you should not write code that depends on sign preservation of signed quantities using a right shift operation.

### Use of *pcrlf()* function in C code examples

The C code examples use a function named *pcrlf()* to output a carriage return (0x0D), line feed (0x0A) to the console. I did this because I was unsure as to what terminal programs a reader might use with the code, and how the terminal program might interpret a carriage return, line feed (i.e, a line feed might also perform a carriage return in the terminal program). Using the *pcrlf()* function makes it easy to alter the behavior of how a new line/carriage return is produced. However, if Hyperterm is always used, then one can just use "\r\n" embedded in the string to produce a carriage return, line feed which might be preferable for some readers. Thus, the following two *printf()* statements accomplish the same thing in Hyperterm.

```
printf("Hello World"); pcrlf();  
printf("Hello World\r\n");
```

The advantage of "\r\n" embedded in the string is that less program code space is used.

### Use of *printf()* for formatted I/O

The C code examples use *printf()* for formatted I/O to make the examples easy to understand, portable between compilers, and to provide flexibility in producing output for debug purposes. However, be aware that *printf()* is a complex function and that code size is greatly increased when the *printf()* function is included. Most compilers provide compiler-dependent string I/O functions that offer less capability than *printf()*, but require a fraction of the code space that *printf()* requires.

## Modifying the LCD code in Figure 8.25

Be careful when modifying the `#define` statements of the LCD code in Figure 8.25 to change the port assignments. The code in `lcd_write()` was written assuming that different ports would be used for the control signals and data signals. If you use the same port (perhaps PORTB) for both the data signals and the control signals, then you will have to make modifications to the `lcd_write()` function as well to ensure that the control signal values are not disturbed when you write data to the port.

## Floating Point Format for the PICC18, MCC18 Compilers

In Chapter 7, the IEEE 754 floating point format is discussed which is 32-bit single precision and 64-bit double precision. In most C compilers, the 32-bit format is used for a `float` type and the 64-bit format is used for a `double` type. However, the PICC18 compiler uses a truncated 24-bit format for both `float` and `double` types by default. The IEEE 754 32-bit format is used for `double` only if a compiler option is used (`-D24` or `--double=32` depending on the compiler version). The Microchip MCC18 compiler (V3.02 at this writing) uses the IEEE 754 32-bit format for both `float` and `double` types.

## Errata

Chapter 1:

Sample problem, pg. 23 (correction in BOLDFACE,  $10^6$  replaced with  $10^{-6}$ )  
 “via:  $5.21e10-5s = 5.21e10^{-5}s * 1 \text{ us}/1e10^{-6}s = 52.1 \mu s$ ”

Chapter 2:

Page 36 (typo), First paragraph, line 6, replace “requires **six** DFFs” with “requires **seven** DFFs”. Table 2.1 has the correct number of seven bits for one-hot encoding.

Page 43. First paragraph, first line, replace “label `start`” with “label `local`”,  
 “we see that the value for label `local` is the memory address 0x4”.

Chapter 4 (typo):

Pg. 102, Problem 18, 3<sup>rd</sup> line

“..tests the value of the LSb of `jwhile{}` loop structure.” should read as only:  
 “..tests the value of the LSb of `j`.”

Chapter 5(typo)

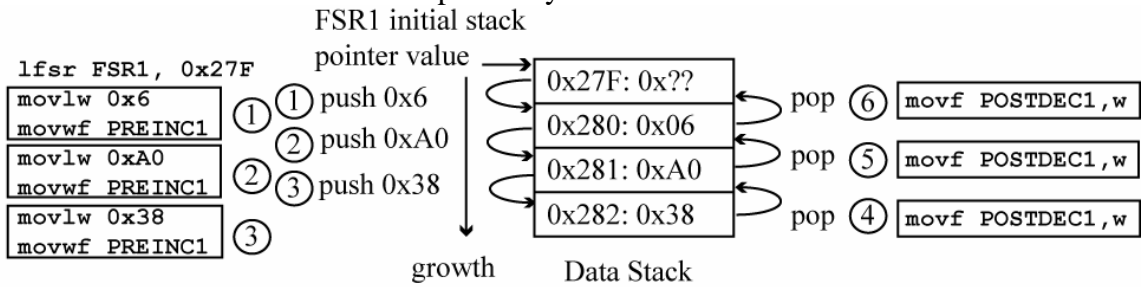
Pg. 125, Figure 5.25, in the assembly code, replace “`subwfb j+w,w`” with

“**subwfb j+1,w**”

Chapter 6:

Page 154, first word, first paragraph: Replace “**PLUWn**” with “**PLUSWn**”.

Page 163, Figure 6.20. All occurrences of **PREINC** should be replaced by **PREINC1**, and all occurrences of **POSTDEC** be replaced by **POSTDEC1** as shown below.



page 173, problem 18 (cut/paste error)  
 The **putch** function should read as:

```
putch (unsigned char c)
{
    // not shown
}
```

Chapter 7:

Pg 179, Figure 7.4. The leftmost HA on the last row should be a FA as shown below.



Figure 13.13 (Compare Mode): The '1' and '0' inputs on the two muxes are reversed.  
 The corrected figure is shown below:

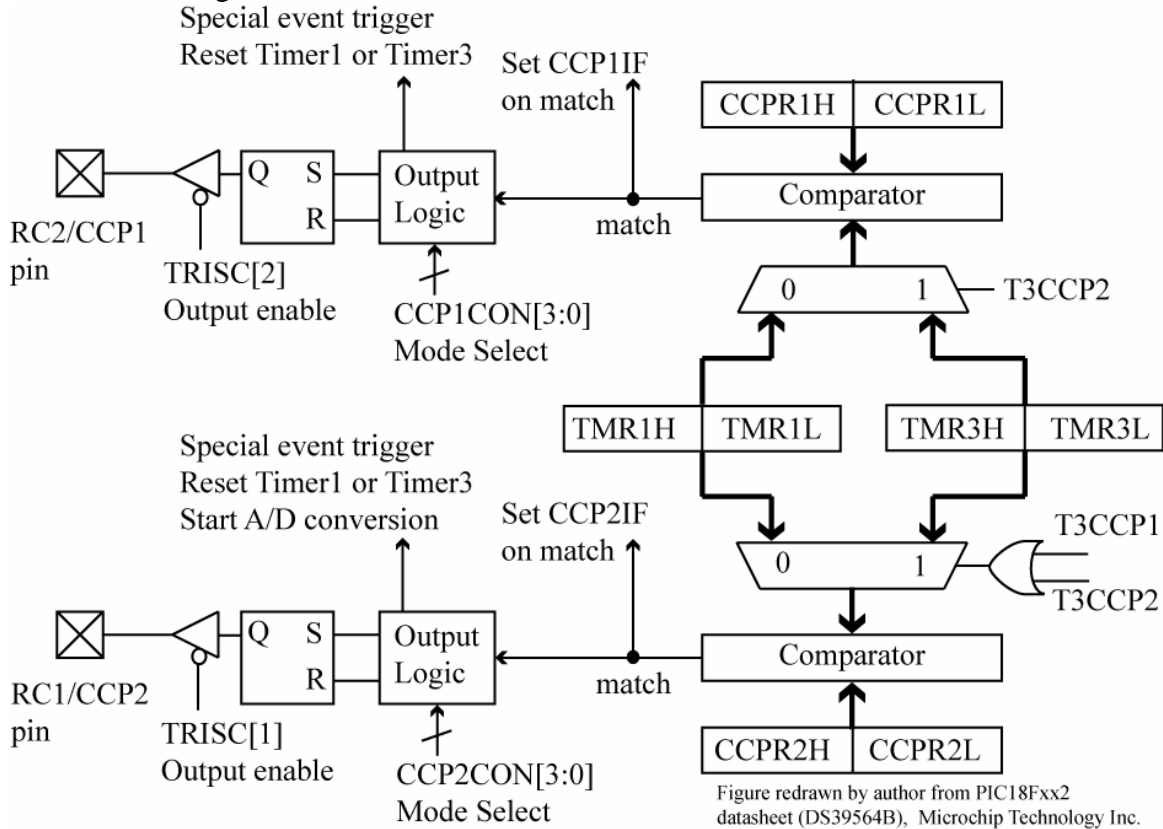


Figure redrawn by author from PIC18Fxx2  
 datasheet (DS39564B), Microchip Technology Inc.

Chapter 15:

Table 15.1, pg 515, formatting problem.

The entry "TBLWT(\*/\*+/\*-/\*+\*)" as the 2<sup>nd</sup> to last entry in Column 1 (*Feature*) should be in Column 3 (*PIC18*).