cations in which only microamps of current are consumed. To further reduce cost and complexity, the microcontrollers contain on-board clock drivers that work with a variety of external frequency-reference components. Quartz crystals are supported, as they are very accurate references. In very small systems wherein cost and size are absolutely paramount concerns, and absolute frequency accuracy is not a concern, less-expensive and smaller frequency references can be used with a PIC microcontroller. One step down from a crystal is a *ceramic resonator,* which functions on a similar principle but with lower accuracy and cost. Finally, if the operating frequency can be allowed to vary more substantially with temperature, voltage, and time, a resistor/capacitor (RC) oscillator, the cheapest option, is supported. Tiny surface mount RC components take up very little circuit board area and cost pennies.

The original 16C5x family incorporates only the most basic of peripherals: power-on-reset, an eight-bit timer/counter, and a *watchdog timer.* A power-on reset circuit ensures that the microcontroller reliably begins operation when power is applied by automatically controlling an internal reset signal. On most microprocessors, reset is purely an external function. A *watchdog* timer can be configured to automatically reset the microcontroller if the system develops an unforeseen fault that causes the software to "crash." The watchdog functions by continuously counting, and software must periodically reset the counter to prevent it from reaching its terminal count value. If this value is reached, the internal reset signal is asserted. Under normal circumstances where software is functioning properly, it resets the watchdog timer with plenty of time to spare. However, if the software crashes, it will presumably not be able to reset the watchdog, and a system reset will soon follow. The watchdog timeout period is configurable from milliseconds to seconds. When using a watchdog, the timeout period is chosen to be long enough so that software can reliably reset the counter to prevent accidental reset, yet short enough to catch a fault and reset the system before serious problems result.

The PIC microcontroller's RISC instruction set obeys the tenets of the general RISC style: accomplish the same task with more simple instructions instead of fewer complex ones. Fewer types of simple instructions require less processing logic within the chip. As an example, there are just two branch instructions: `CALL` and `GOTO`. `CALL` is an unconditional branch-to-subroutine that places the current PC onto the stack. It is the programmer's responsibility to not nest subroutines more than two deep to avoid overflowing the stack. `GOTO` simply loads a new value into the PC. To implement conditional branches, these instructions are paired with one of four instructions that perform an action and then skip the following instruction if a particular result is true. `INCFSZ` and `DECFSZ` increment or decrement a designated register, respectively, and then skip the following instruction if the result is zero. `BTFSC` and `BTFSS` test a specified bit in a register and then skip the following instruction if the bit is 0 or 1, respectively. Using the first pair of instructions, a loop could be written as shown in Fig. 6.8.

Assembly languages commonly offer the programmer a means of representing numeric values with alphanumeric labels for convenience. Here, the loop variable `COUNT` is set to address 0 with an *equate* directive that is recognized and processed by the assembler. `MOVWF` transfers the value in the

```
COUNT           EQU         0           ; define COUNT at address 0

                MOVLW       0x09        ; 9 loop iterations
                MOVW  F     COUNT       ; iteration tracking register
LOOP_START      <loop instructions>     ; body of loop
                DECFSZ      COUNT,1     ; done with loop yet?
                GOTO        LOOP_START  ; non-zero, keep going...
                <more instructions>     ; zero, loop is done...
```

FIGURE 6.8 16C5x assembly language loop.

working register into a particular location in the register file. In this example, the GOTO instruction is executed each time through the loop until COUNT is decremented to 0. (The operand "1" following COUNT in DECFSZ tells the microcontroller to place the decremented result back into COUNT rather than into the working register.) At this point, GOTO is skipped, because the result is 0, causing the microcontroller to continue executing additional instructions outside of the loop.

The second pair of skip instructions, BTFSC and BTFSS, directly supports the common situation in which the microcontroller reads a set of flag bits in a single byte and then takes action based on one of those bits. Such bit-testing instructions are common in microcontrollers by virtue of their intended applications. Some generic microprocessors do not contain bit-testing instructions, requiring software to isolate the bit of interest with a logical *mask* operation. A mask operation works as follows with an AND function, assuming that we want to isolate bit 5 of a byte so as to test its state:

	1	0	1	1	0	1	1	1	Byte to test
	0	0	1	0	0	0	0	0	Mask
AND	0	0	1	0	0	0	0	0	Bit 5 isolated

Here, the mask prevents any bit other than bit 5 from achieving a 1 state in the final result. This masking operation could then be followed with a conditional branch testing whether the overall result was 0 or non-0. In the PIC architecture, and most other microcontrollers, this process is performed directly with bit-test instructions.

Masking also works to set or clear individual bits but, here again, the PIC architecture contains special instructions to optimize this common microcontroller function. Using the above example, bit 5 can be set, regardless of its current state, by ORing the data byte with the same mask.

	1	0	1	1	0	1	1	1	Starting byte
	0	0	1	0	0	0	0	0	Mask
OR	0	0	1	0	0	0	0	0	Result

The mask ensures that only bit 5 is set, regardless of its current state. All other bits propagate through the OR process without being changed. Similarly, an individual bit can be cleared, regardless of its current state, with an inverse AND mask:.

	1	0	1	1	0	1	1	1	Starting byte
	1	1	0	1	1	1	1	1	Mask
AND	1	0	0	1	0	1	1	1	Result

Here, all bits other than bit 5 are ANDed with 1, propagating them through to the result. Bit 5 is ANDed with 0, unconditionally forcing it to a 0. Rather than having to load a mask and then execute a logical instruction, the PIC architecture contains two instructions to clear and set arbitrary bits in a specified register: BCF and BSF, respectively.