In this chapter we explore the concept of the interrupt and interrupt programming. In Section 11.1 the basics of 8051 interrupts are discussed. In Section 11.2 interrupts belonging to Timers 0 and 1 are discussed. External hardware interrupts are discussed in Section 11.3, while the interrupt related to serial communication is presented in Section 11.4. In Section 11.5, we cover interrupt priority in the 8051/52. Finally, C programming of 8051 interrupts is covered in Section 11.6.

SECTION 11.1: 8051 INTERRUPTS

In this section, first we examine the difference between polling and interrupts and then describe the various interrupts of the 8051.

Interrupts vs. polling

A single microcontroller can serve several devices. There are two ways to do that: interrupts or polling. In the *interrupt* method, whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device. The program associated with the interrupt is called the *interrupt service routine* (ISR) or *interrupt handler*. In *polling*, the microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service. After that, it moves on to monitor the next device until each one is serviced. Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not an efficient use of the microcontroller. The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time, of course); each device can get the attention of the microcontroller based on the priority assigned to it. The polling method cannot assign priority since it checks all devices in a round-robin fashion. More importantly, in the interrupt method the microcontroller can also ignore (mask) a device request for service. This is again not possible with the polling method. The most important reason that the interrupt method is preferable is that the polling method wastes much of the microcontroller's time by polling devices that do not need service. So in order to avoid tying down the microcontroller, interrupts are used. For example, in discussing timers in Chapter 9 we used the instruction "`JNB TF,target`", and waited until the timer rolled over, and while we were waiting we could not do anything else. That is a waste of the microcontroller's time that could have been used to perform some useful tasks. In the case of the timer, if we use the interrupt method, the microcontroller can go about doing other tasks, and when the TF flag is raised the timer will interrupt the microcontroller in whatever it is doing.

Interrupt service routine

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table, shown in Table 11-1.