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.

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps.

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
2. It also saves the current status of all the interrupts internally (i.e., not on the stack).
3. It jumps to a fixed location in memory called the interrupt vector table that holds the address of the interrupt service routine.
4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI (return from interrupt).
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC. Then it starts to execute from that address.

Notice from Step 5 the critical role of the stack. For this reason, we must be careful in manipulating the stack contents in the ISR. Specifically, in the ISR, just as in any CALL subroutine, the number of pushes and pops must be equal.

Six interrupts in the 8051

In reality, only five interrupts are available to the user in the 8051, but many manufacturers' data sheets state that there are six interrupts since they include reset. The six interrupts in the 8051 are allocated as follows.

1. Reset. When the reset pin is activated, the 8051 jumps to address location 0000. This is the power-up reset discussed in Chapter 4.
2. Two interrupts are set aside for the timers: one for Timer 0 and one for Timer 1. Memory locations 000BH and 001BH in the interrupt vector table belong to Timer 0 and Timer 1, respectively.
3. Two interrupts are set aside for hardware external hardware interrupts. Pin numbers 12 (P3.2) and 13 (P3.3) in port 3 are for the external hardware interrupts INT0 and INT1, respectively. These external interrupts are also referred to as EX1 and EX2. Memory locations 0003H and 0013H in the interrupt vector table are assigned to INT0 and INT1, respectively.
4. Serial communication has a single interrupt that belongs to both receive and transmit. The interrupt vector table location 0023H belongs to this interrupt.

Notice in Table 11-1 that a limited number of bytes is set aside for each interrupt. For example, a total of 8 bytes from location 0003 to 0000A is set aside for INT0, external hardware interrupt 0. Similarly, a total of 8 bytes from location 000BH to 0012H is reserved for TF0, Timer 0 interrupt. If the service routine for a given interrupt is short enough to fit in the memory space allocated to it, it is placed in the vector table; otherwise, an LJMP instruction is placed in the vector table to point to the address of the ISR. In that case, the rest of the bytes allocated to that interrupt are unused. In the next three sections we will see many examples of interrupt programming that clarify these concepts.

From Table 11-1, also notice that only three bytes of ROM space are assigned to the reset pin. They are ROM address locations 0, 1, and 2. Address location 3 belongs to external hardware interrupt 0. For this reason, in our program we put the LJMP as the first instruction and redirect the processor away from the interrupt vector table, as shown in Figure 11-1. In the next section we will see how this works in the context of some examples.

Table 11-1: Interrupt Vector Table for the 8051

Interrupt	ROM Location (Hex)	Pin	Flag Clearing
Reset	0000	9	Auto
External hardware interrupt 0 (INT0)	0003	P3.2 (12)	Auto
Timer 0 interrupt (TF0)	000B		Auto
External hardware interrupt 1 (INT1)	0013	P3.3 (13)	Auto
Timer 1 interrupt (TF1)	001B		Auto
Serial COM interrupt (RI and TI)	0023		Programmer clears it.

```
          ORG  0     ;wake-up ROM reset location
          LJMP MAIN ;bypass interrupt vector table

;---- the wake-up program
          ORG  30H
MAIN:
          ....
          END
```

Figure 11-1. Redirecting the 8051 from the Interrupt Vector Table at Power-up

Enabling and disabling an interrupt

Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated. The interrupts must be enabled by software in order for the microcontroller to respond to them. There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts. Figure 11-2 shows the IE register. Note that IE is a bit-addressable register.

From Figure 11-2 notice that bit D7 in the IE register is called EA (enable all). This must be set to 1 in order for the rest of the register to take effect. D6 is unused. D5 is used by the 8052. The D4 bit is for the serial interrupt, and so on.

Steps in enabling an interrupt

To enable an interrupt, we take the following steps:

1. Bit D7 of the IE register (EA) must be set to high to allow the rest of register to take effect.
2. If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high. If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is high.

To understand this important point look at Example 11-1.

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
--	IE.6	Not implemented, reserved for future use.*
ET2	IE.5	Enables or disables Timer 2 overflow or capture interrupt (8052 only).
ES	IE.4	Enables or disables the serial port interrupt.
ET1	IE.3	Enables or disables Timer 1 overflow interrupt.
EX1	IE.2	Enables or disables external interrupt 1.
ET0	IE.1	Enables or disables Timer 0 overflow interrupt.
EX0	IE.0	Enables or disables external interrupt 0.

*User software should not write 1s to reserved bits. These bits may be used in future flash microcontrollers to invoke new features.

Figure 11-2. IE (Interrupt Enable) Register

Example 11-1

Show the instructions to (a) enable the serial interrupt, Timer 0 interrupt, and external hardware interrupt 1 (EX1), and (b) disable (mask) the Timer 0 interrupt, then (c) show how to disable all the interrupts with a single instruction.

Solution:

```
(a)   MOV IE,#10010110B   ;enable serial, Timer 0, EX1
```

Since IE is a bit-addressable register, we can use the following instructions to access individual bits of the register.

```
(b)   CLR IE.1        ;mask(disable) Timer 0 interrupt only
(c)   CLR IE.7        ;disable all interrupts
```

Another way to perform the "MOV IE,#10010110B" instruction is by using single-bit instructions as shown below.

```
SETB IE.7       ;EA=1, Global enable
SETB IE.4       ;enable serial interrupt
SETB IE.1       ;enable Timer 0 interrupt
SETB IE.2       ;enable EX1
```

Review Questions

1. Of the interrupt and polling methods, which one avoids tying down the microcontroller?
2. Besides reset, how many interrupts do we have in the 8051?
3. In the 8051, what memory area is assigned to the interrupt vector table? Can the programmer change the memory space assigned to the table?
4. What are the contents of register IE upon reset, and what do these contents mean?
5. Show the instruction to enable the EX0 and Timer 0 interrupts.
6. Which pin of the 8051 is assigned to the external hardware interrupt INT1?
7. What address in the interrupt vector table is assigned to the INT1 and Timer 1 interrupts?

SECTION 11.2: PROGRAMMING TIMER INTERRUPTS

In Chapter 9 we discussed how to use Timer 0 and Timer 1 with the polling method. In this section we use interrupts to program the 8051 timers. Please review Chapter 9 before you study this section.

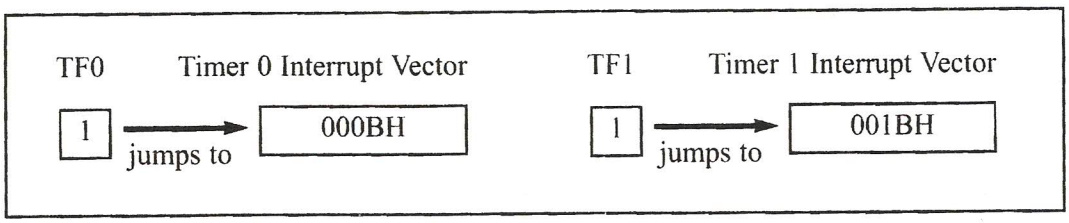


Figure 11-3. TF Interrupt

Roll-over timer flag and interrupt

In Chapter 9 we stated that the timer flag (TF) is raised when the timer rolls over. In that chapter, we also showed how to monitor TF with the instruction "`JNB TF, target`". In polling TF, we have to wait until the TF is raised. The problem with this method is that the microcontroller is tied down while waiting for TF to be raised, and cannot do any thing else. Using interrupts solves this problem and avoids tying down the controller. If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised, and the microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the ISR. In this way, the microcontroller can do other things until it is notified that the timer has rolled over. See Figure 11-3 and Example 11-2.

Notice the following points about the program in Example 11-2.

1. We must avoid using the memory space allocated to the interrupt vector table. Therefore, we place all the initialization codes in memory starting at 30H. The LJMP instruction is the first instruction that the 8051 executes when it is powered up. LJMP redirects the controller away from the interrupt vector table.

2. The ISR for Timer 0 is located starting at memory location 000BH since it is small enough to fit the address space allocated to this interrupt.
3. We enabled the Timer 0 interrupt with "MOV IE,#10000010B" in MAIN.
4. While the P0 data is brought in and issued to P1 continuously, whenever Timer 0 is rolled over, the TF0 flag is raised, and the microcontroller gets out of the "BACK" loop and goes to 0000BH to execute the ISR associated with Timer 0.
5. In the ISR for Timer 0, notice that there is no need for a "CLR TF0" instruction before the RETI instruction. This is because the 8051 clears the TF flag internally upon jumping to the interrupt vector table.

Example 11-2

Write a program that continuously gets 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 μs period on pin P2.1. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

Solution:

We will use Timer 0 in mode 2 (auto-reload). TH0 = 100/1.085 μs = 92.

```
;---Upon wake-up go to main, avoid using memory space ;allocat-
ed to Interrupt Vector Table
          ORG   0000H
          LJMP  MAIN          ;bypass interrupt vector table
;
;---ISR for Timer 0 to generate square wave
          ORG   000BH         ;Timer 0 interrupt vector table
          CPL   P2.1          ;toggle P2.1 pin
          RETI                ;return from ISR
;
;---The main program for initialization
          ORG   0030H         ;after vector table space
MAIN:     MOV   TMOD,#02H     ;Timer 0, mode 2(auto-reload)
          MOV   P0,#0FFH      ;make P0 an input port
          MOV   TH0,#-92      ;TH0=A4H for -92
          MOV   IE,#82H       ;IE=10000010(bin) enable Timer 0
          SETB  TR0           ;Start Timer 0
BACK:     MOV   A,P0          ;get data from P0
          MOV   P1,A          ;issue it to P1
          SJMP  BACK          ;keep doing it
                              ;loop unless interrupted by TF0
          END
```

In Example 11-2, the interrupt service routine was short enough that it could be placed in memory locations allocated to the Timer 0 interrupt. However, that is not always the case. See Example 11-3.

Example 11-3

Rewrite Example 11-2 to create a square wave that has a high portion of 1085 μs and a low portion of 15 μs. Assume XTAL = 11.0592 MHz. Use Timer 1.

Solution:

Since 1085 μs is 1000 × 1.085 we need to use mode 1 of Timer 1.

```
;--Upon wake-up go to main, avoid using memory space
;--allocated to Interrupt Vector Table
          ORG   0000H
          LJMP  MAIN        ;bypass interrupt vector table
;
;--ISR for Timer 1 to generate square wave
          ORG   001BH       ;Timer 1 interrupt vector table
          LJMP  ISR_T1      ;jump to ISR
;
;--The main program for initialization
          ORG   0030H       ;after vector table
MAIN:     MOV   TMOD,#10H   ;Timer 1, mode 1
          MOV   P0,#0FFH    ;make P0 an input port
          MOV   TL1,#018H   ;TL1=18 the Low byte of -1000
          MOV   TH1,#0FCH   ;TH1=FC the High byte of -1000
          MOV   IE,#88H     ;IE=10001000 enable Timer 1 int.
          SETB  TR1         ;start Timer 1
BACK:     MOV   A,P0        ;get data from P0
          MOV   P1,A        ;issue it to P1
          SJMP  BACK        ;keep doing it
;
;--Timer 1 ISR. Must be reloaded since not auto-reload
ISR_T1:   CLR   TR1         ;stop Timer 1
          CLR   P2.1        ;P2.1=0, start of low portion
          MOV   R2,#4       ;                              2 MC
HERE:     DJNZ  R2,HERE     ;4x2 machine cycle(MC)         8 MC
          MOV   TL1,#18H    ;load T1 Low byte value        2 MC
          MOV   TH1,#0FCH   ;load T1 High byte value       2 MC
          SETB  TR1         ;starts Timer 1                1 MC
          SETB  P2.1        ;P2.1=1, back to high          1 MC
          RETI              ;return to main
          END
```

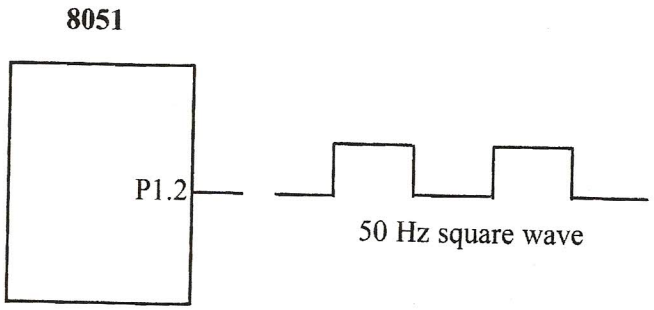
Notice that the low portion of the pulse is created by the 14 MC (machine cycles) where each MC = 1.085 μs and 14 × 1.085 μs = 15.19 μs.

Example 11-4

Write a program to generate a square wave of 50 Hz frequency on pin P1.2. This is similar to Example 9-12 except that it uses an interrupt for Timer 0. Assume that XTAL = 11.0592 MHz.

Solution:

```
            ORG   0
            LJMP  MAIN
            ORG   000BH              ;ISR for Timer 0
            CPL   P1.2               ;complement P1.2
            MOV   TL0,#00            ;reload timer values
            MOV   TH0,#0DCH
            RETI                     ;return from interrupt
            ORG   30H                ;starting location for prog.
;------main program for initialization
MAIN:       MOV   TMOD,#00000001B    ;Timer 0, Mode 1
            MOV   TL0,#00
            MOV   TH0,#0DCH
            MOV   IE,#82H            ;enable Timer 0 interrupt
            SETB  TR0                ;start timer
HERE:       SJMP  HERE               ;stay here until interrupted
            END
```



Review Questions

1. True or false. There is only a single interrupt in the interrupt vector table assigned to both Timer 0 and Timer 1.
2. What address in the interrupt vector table is assigned to Timer 0?
3. Which bit of IE belongs to the timer interrupt? Show how both are enabled.
4. Assume that Timer 1 is programmed in mode 2, TH1 = F5H, and the IE bit for Timer 1 is enabled. Explain how the interrupt for the timer works.
5. True or false. The last two instructions of the ISR for Timer 0 are:

```
CLR  TF0
RETI
```

SECTION 11.3: PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

The 8051 has two external hardware interrupts. Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts. Upon activation of these pins, the 8051 gets interrupted in whatever it is doing and jumps to the vector table to perform the interrupt service routine. In this section we study these two external hardware interrupts of the 8051 with some examples.

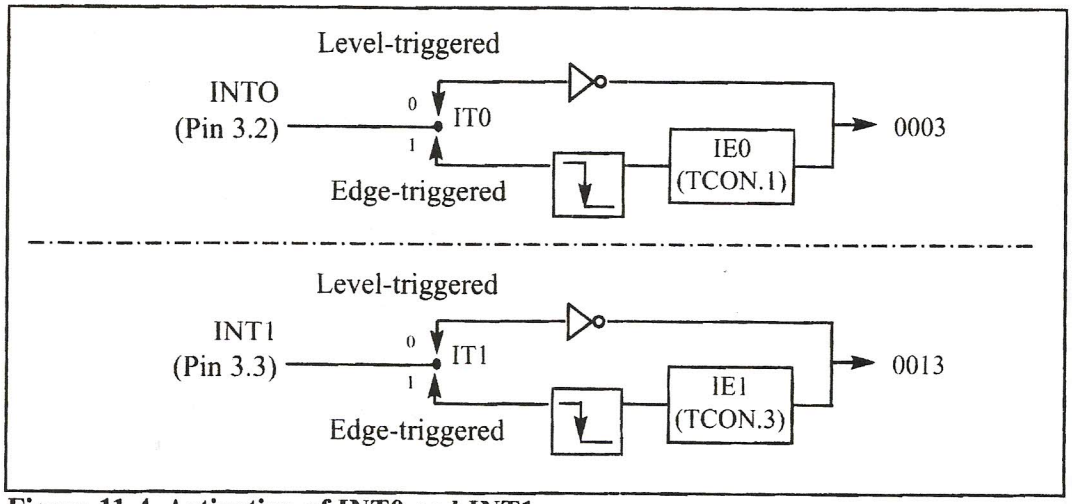


Figure 11-4. Activation of INT0 and INT1

External interrupts INT0 and INT1

There are only two external hardware interrupts in the 8051: INT0 and INT1. They are located on pins P3.2 and P3.3 of port 3, respectively. The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1, respectively. As mentioned in Section 11.1, they are enabled and disabled using the IE register. How are they activated? There are two types of activation for the external hardware interrupts: (1) level triggered, and (2) edge triggered. Let's look at each one. First, we see how the level-triggered interrupt works.

Level-triggered interrupt

In the level-triggered mode, INT0 and INT1 pins are normally high (just like all I/O port pins) and if a low-level signal is applied to them, it triggers the interrupt. Then the microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt. This is called a *level-triggered* or *level-activated* interrupt and is the default mode upon reset of the 8051. The low-level signal at the INT pin must be removed before the execution of the last instruction of the interrupt service routine, RETI; otherwise, another interrupt will be generated. In other words, if the low-level interrupt signal is not removed before the ISR is finished it is interpreted as another interrupt and the 8051 jumps to the vector table to execute the ISR again. Look at Example 11-5.

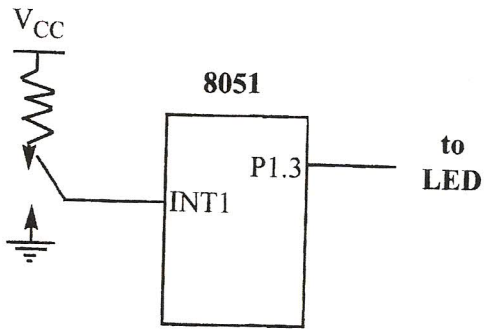
Example 11-5

Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.

Solution:

```
            ORG   0000H
            LJMP  MAIN              ;bypass interrupt vector table
;--ISR for hardware interrupt INT1 to turn on the LED
            ORG   0013H             ;INT1 ISR
            SETB  P1.3              ;turn on LED
            MOV   R3,#255           ;load counter
BACK:       DJNZ  R3,BACK           ;keep LED on for a while
            CLR   P1.3              ;turn off the LED
            RETI                    ;return from ISR
;--MAIN program for initialization
            ORG   30H
MAIN:       MOV   IE,#10000100B     ;enable external INT1
HERE:       SJMP  HERE              ;stay here until interrupted
            END
```

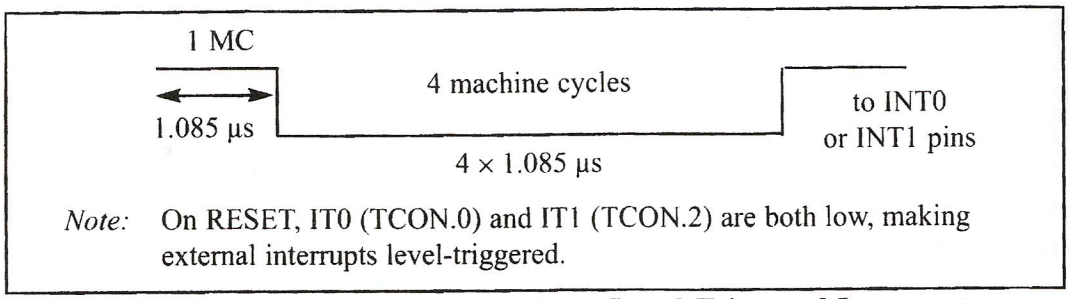
Pressing the switch will turn the LED on. If it is kept activated, the LED stays on.



In this program, the microcontroller is looping continuously in the HERE loop. Whenever the switch on INT1 (pin P3.3) is activated, the microcontroller gets out of the loop and jumps to vector location 0013H. The ISR for INT1 turns on the LED, keeps it on for a while, and turns it off before it returns. If by the time it executes the RETI instruction, the INT1 pin is still low, the microcontroller initiates the interrupt again. Therefore, to end this problem, the INT1 pin must be brought back to high by the time RETI is executed.

Sampling the low level-triggered interrupt

Pins P3.2 and P3.3 are used for normal I/O unless the INT0 and INT1 bits in the IE registers are enabled. After the hardware interrupts in the IE register are enabled, the controller keeps sampling the INT*n* pin for a low-level signal once each machine cycle. According to one manufacturer's data sheet "the pin must be held in a low state until the start of the execution of ISR. If the INT*n* pin is brought back to a logic high before the start of the execution of ISR there will be no interrupt." However, upon activation of the interrupt due to the low level, it must be brought back to high before the execution of RETI. Again, according to one manufacturer's data sheet, "If the INT*n* pin is left at a logic low after the RETI instruction of the ISR, another interrupt will be activated after one instruction is executed." Therefore, to ensure the activation of the hardware interrupt at the INT*n* pin, make sure that the duration of the low-level signal is around 4 machine cycles, but no more. This is due to the fact that the level-triggered interrupt is not latched. Thus the pin must be held in a low state until the start of the ISR execution.



Note: On RESET, IT0 (TCON.0) and IT1 (TCON.2) are both low, making external interrupts level-triggered.

Figure 11-5. Minimum Duration of the Low Level-Triggered Interrupt (XTAL = 11.0592 MHz)

Edge-triggered interrupts

As stated before, upon reset the 8051 makes INT0 and INT1 low-level triggered interrupts. To make them edge-triggered interrupts, we must program the bits of the TCON register. The TCON register holds, among other bits, the IT0 and IT1 flag bits that determine level- or edge-triggered mode of the hardware interrupts. IT0 and IT1 are bits D0 and D2 of the TCON register, respectively. They are also referred to as TCON.0 and TCON.2 since the TCON register is bit-addressable. Upon reset, TCON.0 (IT0) and TCON.2 (IT1) are both 0s, meaning that the external hardware interrupts of INT0 and INT1 pins are low-level triggered. By making the TCON.0 and TCON.2 bits high with instructions such as "`SETB TCON.0`" and "`SETB TCON.2`", the external hardware interrupts of INT0 and INT1 become edge-triggered. For example, the instruction "`SETB CON.2`" makes INT1 what is called an *edge-triggered interrupt*, in which, when a high-to-low signal is applied to pin P3.3, in this case, the controller will be interrupted and forced to jump to location 0013H in the vector table to service the ISR (assuming that the interrupt bit is enabled in the IE register).

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off.
IE1	TCON.3	External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.
IE0	TCON.1	External interrupt 0 edge flag. Set by CPU when external interrupt (H-to-L transition) edge is detected. Cleared by CPU when interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.

Figure 11-6. TCON (Timer/Counter) Register (Bit-addressable)

Look at Example 11-6. Notice that the only difference between this program and the program in Example 11-5 is in the first line of MAIN where the instruction "`SETB TCON.2`" makes INT1 an edge-triggered interrupt. When the falling edge of the signal is applied to pin INT1, the LED will be turned on momentarily. The LED's on-state duration depends on the time delay inside the ISR for INT1. To turn on the LED again, another high-to-low pulse must be applied to pin 3.3. This is the opposite of Example 11-5. In Example 11-5, due to the level-triggered nature of the interrupt, as long as INT1 is kept at a low level, the LED is kept in the on state. But in this example, to turn on the LED again, the INT1 pulse must be brought back high and then forced low to create a falling edge to activate the interrupt.

Example 11-6

Assuming that pin 3.3 (INT1) is connected to a pulse generator, write a program in which the falling edge of the pulse will send a high to P1.3, which is connected to an LED (or buzzer). In other words, the LED is turned on and off at the same rate as the pulses are applied to the INT1 pin. This is an edge-triggered version of Example 11-5.

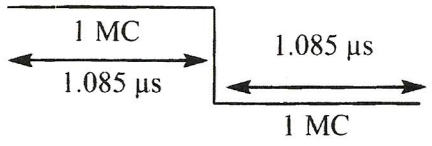
Solution:

```
      ORG   0000H
      LJMP  MAIN
;--ISR for hardware interrupt INT1 to turn on the LED
      ORG   0013H             ;INT1 ISR
      SETB  P1.3              ;turn on the LED
      MOV   R3,#255
BACK: DJNZ  R3,BACK           ;keep the LED on for a while
      CLR   P1.3              ;turn off the LED
      RETI                    ;return from ISR
;--MAIN program for initialization
      ORG   30H
MAIN: SETB  TCON.2            ;make INT1 edge-trigger interrupt
      MOV   IE,#10000100B     ;enable External INT1
HERE: SJMP  HERE              ;stay here until interrupted
      END
```

Sampling the edge-triggered interrupt

Before ending this section, we need to answer the question of how often the edge-triggered interrupt is sampled. In edge-triggered interrupts, the external source must be held high for at least one machine cycle, and then held low for at least one machine cycle to ensure that the transition is seen by the microcontroller.

Minimum pulse duration to detect edge-triggered interrupts.
XTAL = 11.0592 MHz



The falling edge is latched by the 8051 and is held by the TCON register. The TCON.1 and TCON.3 bits hold the latched falling edge of pins INT0 and INT1, respectively. TCON.1 and TCON.3 are also called IE0 and IE1, respectively, as shown in Figure 11-6. They function as interrupt-in-service flags. When an interrupt-in-service flag is raised, it indicates to the external world that the interrupt is being serviced and no new interrupt on this INT*n* pin will be responded to until this service is finished. This is just like the busy signal you get if calling a telephone number that is in use. Regarding the IT0 and IT1 bits in the TCON register, the following two points must be emphasized.

1. The first point is that when the ISRs are finished (that is, upon execution of instruction RETI), these bits (TCON.1 and TCON.3) are cleared, indicating that the interrupt is finished and the 8051 is ready to respond to another interrupt on that pin. For another interrupt to be recognized, the pin must go back to a logic high state and be brought back low to be considered an edge-triggered interrupt.
2. The second point is that while the interrupt service routine is being executed, the INT*n* pin is ignored, no matter how many times it makes a high-to-low transition. In reality one of the functions of the RETI instruction is to clear the corresponding bit in the TCON register (TCON.1 or TCON.3). This informs us that the service routine is no longer in progress and has finished being serviced. For this reason, TCON.1 and TCON.3 in the TCON register are called interrupt-in-service flags. The interrupt-in-service flag goes high whenever a falling edge is detected at the INT pin, and stays high during the entire execution of the ISR. It is only cleared by RETI, the last instruction of the ISR. Because of this, there is no need for an instruction such as "`CLR TCON.1`" (or "`CLR TCON.3`" for INT1) before the RETI in the ISR associated with the hardware interrupt INT0. As we will see in the next section, this is not the case for the serial interrupt.

Example 11-7

What is the difference between the RET and RETI instructions? Explain why we cannot use RET instead of RETI as the last instruction of an ISR.

Solution:

Both perform the same actions of popping off the top two bytes of the stack into the program counter, and making the 8051 return to where it left off. However, RETI also performs an additional task of clearing the interrupt-in-service flag, indicating that the servicing of the interrupt is over and the 8051 now can accept a new interrupt on that pin. If you use RET instead of RETI as the last instruction of the interrupt service routine, you simply block any new interrupt on that pin after the first interrupt, since the pin status would indicate that the interrupt is still being serviced. In the cases of TF0, TF1, TCON.1, and TCON.3, they are cleared by the execution of RETI.

More about the TCON register

Next we look at the TCON register more closely to understand its role in handling interrupts. Figure 11-6 shows the bits of the TCON register.

IT0 and IT1

TCON.0 and TCON.2 are referred to as IT0 and IT1, respectively. These two bits set the low-level or edge-triggered modes of the external hardware interrupts of the INT0 and INT1 pins. They are both 0 upon reset, which makes them low-level triggered. The programmer can make either of them high to make the external hardware interrupt edge-triggered. In a given system based on the 8051, once they are set to 0 or 1 they will not be altered again since the designer has fixed the interrupt as either edge- or level-triggered.

IE0 and IE1

TCON.1 and TCON.3 are referred to as IE0 and IE1, respectively. These bits are used by the 8051 to keep track of the edge-triggered interrupt only. In other words, if the IT0 and IT1 are 0, meaning that the hardware interrupts are low-level triggered, IE0 and IE1 are not used at all. The IE0 and IE1 bits are used by the 8051 only to latch the high-to-low edge transition on the INT0 and INT1 pins. Upon the edge transition pulse on the INT0 (or INT1) pin, the 8051 marks (sets high) the IE*x* bit in the TCON register, jumps to the vector in the interrupt vector table, and starts to execute the ISR. While it is executing the ISR, no H-to-L pulse transition on the INT0 (or INT1) is recognized, thereby preventing any interrupt inside the interrupt. Only the execution of the RETI instruction at the end of the ISR will clear the IE*x* bit, indicating that a new H-to-L pulse will activate the interrupt again. From this discussion we can see that the IE0 and IE1 bits are used internally by the 8051 to indicate whether or not an interrupt is in use. In other words, the programmer is not concerned with these bits since they are solely for internal use.

TR0 and TR1

These are the D4 (TCON.4) and D6 (TCON.6) bits of the TCON register. We were introduced to these bits in Chapter 9. They are used to start or stop timers 0 and 1, respectively. Although we have used syntax such as "`SETB TRx`" and "`CLR Trx`", we could have used instructions such as "`SETB TCON.4`" and "`CLR TCON.4`" since TCON is a bit-addressable register.

TF0 and TF1

These are the D5 (TCON.5) and D7 (TCON.7) bits of the TCON register. We were introduced to these bits in Chapter 9. They are used by timers 0 and 1, respectively, to indicate if the timer has rolled over. Although we have used the syntax "`JNB TFx,target`" and "`CLR Trx`", we could have used instructions such as "`JNB TCON.5,target`" and "`CLR TCON.5`" since TCON is bit-addressable.

Review Questions

1. True or false. There is a single interrupt in the interrupt vector table assigned to both external hardware interrupts IT0 and IT1.
2. What address in the interrupt vector table is assigned to INT0 and INT1? How about the pin numbers on port 3?
3. Which bit of IE belongs to the external hardware interrupts? Show how both are enabled.
4. Assume that the IE bit for the external hardware interrupt EX1 is enabled and is active low. Explain how this interrupt works when it is activated.
5. True or false. Upon reset, the external hardware interrupt is low-level triggered.
6. In Question 5, how do we make sure that a single interrupt is not recognized as multiple interrupts?
7. True or false. The last two instructions of the ISR for INT0 are:

```
CLR  TCON.1
RETI
```

8. Explain the role that each of the two bits TCON.0 and TCON.2 play in the execution of external interrupt 0.

SECTION 11.4: PROGRAMMING THE SERIAL COMMUNICATION INTERRUPT

In Chapter 10 we studied the serial communication of the 8051. All examples in that chapter used the polling method. In this section we explore interrupt-based serial communication, which allows the 8051 to do many things, in addition to sending and receiving data from the serial communication port.

RI and TI flags and interrupts

As you may recall from Chapter 10, TI (transfer interrupt) is raised when the last bit of the framed data, the stop bit, is transferred, indicating that the SBUF register is ready to transfer the next byte. RI (received interrupt), is raised when the entire frame of data, including the stop bit, is received. In other words, when the SBUF register has a byte, RI is raised to indicate that the received byte needs to be picked up before it is lost (overrun) by new incoming serial data. As far as serial communication is concerned, all the above concepts apply equally when using either polling or an interrupt. The only difference is in how the serial communication needs are served. In the polling method, we wait for the flag (TI or RI) to be raised; while we wait we cannot do anything else. In the interrupt method, we are notified when the 8051 has received a byte, or is ready to send the next byte; we can do other things while the serial communication needs are served.

In the 8051 only one interrupt is set aside for serial communication. This interrupt is used to both send and receive data. If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory address location 0023H to execute the ISR. In that ISR we must examine the TI and RI flags to see which one caused the interrupt and respond accordingly. See Example 11-8.

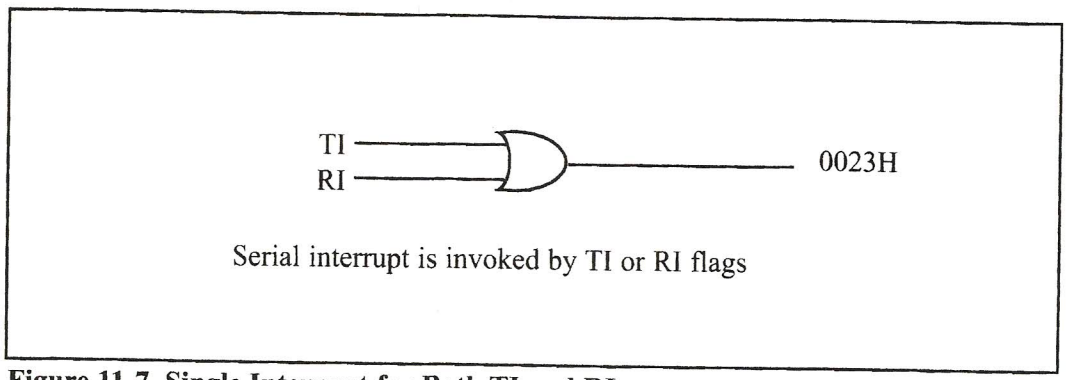


Figure 11-7. Single Interrupt for Both TI and RI

Use of serial COM in the 8051

In the vast majority of applications, the serial interrupt is used mainly for receiving data and is never used for sending data serially. This is like receiving a telephone call, where we need a ring to be notified. If we need to make a phone call there are other ways to remind ourselves and so no need for ringing. In receiving the phone call, however, we must respond immediately no matter what we are doing or we will miss the call. Similarly, we use the serial interrupt to receive incoming data so that it is not lost. Look at Example 11-9.

Example 11-8

Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

Solution:

```
            ORG   0
            LJMP  MAIN
            ORG   23H
            LJMP  SERIAL            ;jump to serial interrupt ISR
            ORG   30H
MAIN:       MOV   P1,#0FFH          ;make P1 an input port
            MOV   TMOD,#20H         ;timer 1, mode 2(auto-reload)
            MOV   TH1,#0FDH         ;9600 baud rate
            MOV   SCON,#50H         ;8-bit, 1 stop, REN enabled
            MOV   IE,#10010000B     ;enable serial interrupt
            SETB  TR1               ;start timer 1
BACK:       MOV   A,P1              ;read data from port 1
            MOV   SBUF,A            ;give a copy to SBUF
            MOV   P2,A              ;send it to P2
            SJMP  BACK              ;stay in loop indefinitely
;
;-----------------Serial Port ISR
            ORG   100H
SERIAL:     JB    TI,TRANS          ;jump if TI is high
            MOV   A,SBUF            ;otherwise due to receive
            CLR   RI                ;clear RI since CPU does not
            RETI                    ;return from ISR
TRANS:      CLR   TI                ;clear TI since CPU does not
            RETI                    ;return from ISR
            END
```

In the above program notice the role of TI and RI. The moment a byte is written into SBUF it is framed and transferred serially. As a result, when the last bit (stop bit) is transferred the TI is raised, which causes the serial interrupt to be invoked since the corresponding bit in the IE register is high. In the serial ISR, we check for both TI and RI since both could have invoked the interrupt. In other words, there is only one interrupt for both transmit and receive.

Clearing RI and TI before the RETI instruction

Notice in Example 11-9 that the last instruction before the RETI is the clearing of the RI or TI flags. This is necessary since there is only one interrupt for both receive and transmit, and the 8051 does not know who generated it; therefore, it is the job of the ISR to clear the flag. Contrast this with the external and timer interrupts where it is the job of the 8051 to clear the interrupt flags. By contrast,

Example 11-9

Write a program in which the 8051 gets data from P1 and sends it to P2 continuously while incoming data from the serial port is sent to P0. Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

Solution:

```
            ORG   0
            LJMP  MAIN
            ORG   23H
            LJMP  SERIAL          ;jump to serial ISR
            ORG   30H
MAIN:       MOV   P1,#0FFH        ;make P1 an input port
            MOV   TMOD,#20H       ;timer 1, mode 2(auto-reload)
            MOV   TH1,#0FDH       ;9600 baud rate
            MOV   SCON,#50H       ;8-bit,1 stop, REN enabled
            MOV   IE,#10010000B   ;enable serial interrupt
            SETB  TR1             ;start Timer 1
BACK:       MOV   A,P1            ;read data from port 1
            MOV   P2,A            ;send it to P2
            SJMP  BACK            ;stay in loop indefinitely
;-----------------SERIAL PORT ISR
            ORG   100H
SERIAL:     JB    TI,TRANS        ;jump if TI is high
            MOV   A,SBUF          ;otherwise due to receive
            MOV   P0,A            ;send incoming data to P0
            CLR   RI              ;clear RI since CPU doesn't
            RETI                  ;return from ISR
TRANS:      CLR   TI              ;clear TI since CPU doesn't
            RETI                  ;return from ISR
            END
```

in serial communication the RI (or TI) must be cleared by the programmer using software instructions such as "CLR TI" and "CLR RI" in the ISR. See Example 11-10. Notice that the last two instructions of the ISR are clearing the flag, followed by RETI.

Before finishing this section notice the list of all interrupt flags given in Table 11-2. While the TCON register holds four of the interrupt flags, in the 8051 the SCON register has the RI and TI flags.

Table 11-2: Interrupt Flag Bits for the 8051/52

Interrupt	Flag	SFR Register Bit
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 0	TF0	TCON.5
Timer 1	TF1	TCON.7
Serial port	TI	SCON.1
Timer 2	TF2	T2CON.7 (AT89C52)
Timer 2	EXF2	T2CON.6 (AT89C52)

Example 11-10

Write a program using interrupts to do the following:
(a) Receive data serially and send it to P0,
(b) Have port P1 read and transmitted serially, and a copy given to P2,
(c) Make Timer 0 generate a square wave of 5 kHz frequency on P0.1.
Assume that XTAL = 11.0592 MHz. Set the baud rate at 4800.

Solution:

```
            ORG   0
            LJMP  MAIN
            ORG   000BH             ;ISR for Timer 0
            CPL   P0.1              ;toggle P0.1
            RETI                    ;return from ISR
            ORG   23H
            LJMP  SERIAL            ;jump to serial int. ISR
            ORG   30H
MAIN:       MOV   P1,#0FFH          ;make P1 an input port
            MOV   TMOD,#22H         ;timer 0&1,mode 2, auto-reload
            MOV   TH1,#0F6H         ;4800 baud rate
            MOV   SCON,#50H         ;8-bit, 1 stop, REN enabled
            MOV   TH0,#-92          ;for 5 KHz wave
            MOV   IE,#10010010B     ;enable serial, timer 0 int.
            SETB  TR1               ;start timer 1
            SETB  TR0               ;start timer 0
BACK:       MOV   A,P1              ;read data from port 1
            MOV   SBUF,A            ;give a copy to SBUF
            MOV   P2,A              ;write it to P2
            SJMP  BACK              ;stay in loop indefinitely

;-----------------SERIAL PORT ISR
            ORG   100H
SERIAL:     JB    TI,TRANS          ;jump if TI is high
            MOV   A,SBUF            ;otherwise due to received
            MOV   P0,A              ;send serial data to P0
            CLR   RI                ;clear RI since CPU does not
            RETI                    ;return from ISR
TRANS:      CLR   TI                ;clear TI since CPU does not
            RETI                    ;return from ISR
            END
```

Review Questions

1. True or false. There is a single interrupt in the interrupt vector table assigned to both the TI and RI interrupts.
2. What address in the interrupt vector table is assigned to the serial interrupt?
3. Which bit of the IE register belongs to the serial interrupt? Show how it is enabled.
4. Assume that the IE bit for the serial interrupt is enabled. Explain how this interrupt gets activated and also explain its actions upon activation.

5. True or false. Upon reset, the serial interrupt is active and ready to go.
6. True or false. The last two instructions of the ISR for the receive interrupt are:

```
CLR  RI
RETI
```

7. Answer Question 6 for the send interrupt.

SECTION 11.5: INTERRUPT PRIORITY IN THE 8051/52

The next topic that we must deal with is what happens if two interrupts are activated at the same time? Which of these two interrupts is responded to first? Interrupt priority is the main topic of discussion in this section.

Interrupt priority upon reset

When the 8051 is powered up, the priorities are assigned according to Table 11-3. From Table 11-3 we see, for example, that if external hardware interrupts 0 and 1 are activated at the same time, external interrupt 0 (INT0) is responded to first. Only after INT0 has been serviced is INT1 serviced, since INT1 has the lower priority. In reality, the priority scheme in the table is nothing but an internal polling sequence in which the 8051 polls the interrupts in the sequence listed in Table 11-3, and responds accordingly.

Table 11-3: 8051/52 Interrupt Priority Upon Reset

Highest to Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)
Timer 2 (8052 only)	**TF2**

Example 11-11

Discuss what happens if interrupts INT0, TF0, and INT1 are activated at the same time. Assume priority levels were set by the power-up reset and that the external hardware interrupts are edge-triggered.

Solution:

If these three interrupts are activated at the same time, they are latched and kept internally. Then the 8051 checks all five interrupts according to the sequence listed in Table 11-3. If any is activated, it services it in sequence. Therefore, when the above three interrupts are activated, IE0 (external interrupt 0) is serviced first, then Timer 0 (TF0), and finally IE1 (external interrupt 1).

D7							D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

Priority bit = 1 assigns high priority. Priority bit = 0 assigns low priority.

--	IP.7	Reserved
--	IP.6	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only)
PS	IP.4	Serial port interrupt priority bit
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

User software should never write 1s to unimplemented bits, since they may be used in future products.

Figure 11-8. Interrupt Priority Register (Bit-addressable)

Setting interrupt priority with the IP register

We can alter the sequence of Table 11-3 by assigning a higher priority to any one of the interrupts. This is done by programming a register called IP (interrupt priority). Figure 11-8 shows the bits of the IP register. Upon power-up reset, the IP register contains all 0s, making the priority sequence based on Table 11-3. To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high. Look at Example 11-12.

Example 11-12

(a) Program the IP register to assign the highest priority to INT1 (external interrupt 1), then (b) discuss what happens if INT0, INT1, and TF0 are activated at the same time. Assume that the interrupts are both edge-triggered.

Solution:

(a)
```
MOV IP,#00000100B ;IP.2=1 to assign INT1 higher priority
```
The instruction "`SETB IP.2`" also will do the same thing as the above line since IP is bit-addressable.

(b) The instruction in Step (a) assigned a higher priority to INT1 than the others; therefore, when INT0, INT1, and TF0 interrupts are activated at the same time, the 8051 services INT1 first, then it services INT0, then TF0. This is due to the fact that INT1 has a higher priority than the other two because of the instruction in Step (a). The instruction in Step (a) makes both the INT0 and TF0 bits in the IP register 0. As a result, the sequence in Table 11-3 is followed, which gives a higher priority to INT0 over TF0.

Example 11-13

Assume that after reset, the interrupt priority is set by the instruction "`MOV IP, #00001100B`". Discuss the sequence in which the interrupts are serviced.

Solution:

The instruction "`MOV IP,#00001100B`" (B is for binary) sets the external interrupt 1 (INT1) and Timer 1 (TF1) to a higher priority level compared with the rest of the interrupts. However, since they are polled according to Table 11-3, they will have the following priority.

Highest Priority	External Interrupt 1	(INT1)
	Timer Interrupt 1	(TF1)
	External Interrupt 0	(INT0)
	Timer Interrupt 0	(TF0)
Lowest Priority	Serial Communication	(RI + TI)

Another point that needs to be clarified is the interrupt priority when two or more interrupt bits in the IP register are set to high. In this case, while these interrupts have a higher priority than others, they are serviced according to the sequence of Table 11-3. See Example 11-13.

Interrupt inside an interrupt

What happens if the 8051 is executing an ISR belonging to an interrupt and another interrupt is activated? In such cases, a high-priority interrupt can interrupt a low-priority interrupt. This is an interrupt inside an interrupt. In the 8051 a low-priority interrupt can be interrupted by a higher-priority interrupt, but not by another low-priority interrupt. Although all the interrupts are latched and kept internally, no low-priority interrupt can get the immediate attention of the CPU until the 8051 has finished servicing the high-priority interrupts.

Triggering the interrupt by software

There are times when we need to test an ISR by way of simulation. This can be done with simple instructions to set the interrupts high and thereby cause the 8051 to jump to the interrupt vector table. For example, if the IE bit for Timer 1 is set, an instruction such as "`SETB TF1`" will interrupt the 8051 in whatever it is doing and force it to jump to the interrupt vector table. In other words, we do not need to wait for Timer 1 to roll over to have an interrupt. We can cause an interrupt with an instruction that raises the interrupt flag.

Review Questions

1. True or false. Upon reset, all interrupts have the same priority.
2. What register keeps track of interrupt priority in the 8051? Is it a bit-addressable register?
3. Which bit of IP belongs to the serial interrupt priority? Show how to assign it the highest priority.
4. Assume that the IP register contains all 0s. Explain what happens if both INT0 and INT1 are activated at the same time.
5. Explain what happens if a higher-priority interrupt is activated while the 8051 is serving a lower-priority interrupt (that is, executing a lower-priority ISR).

SECTION 11.6: INTERRUPT PROGRAMMING IN C

So far all the programs in this chapter have been written in Assembly. In this section we show how to program the 8051/52's interrupts in 8051 C language. In reading this section, it is assumed that you already know the material in the first two sections of this chapter.

8051 C interrupt numbers

The 8051 C compilers have extensive support for the 8051 interrupts with two major features as follows:

1. They assign a unique number to each of the 8051 interrupts, as shown in Table 11-4.
2. It can also assign a register bank to an ISR. This avoids code overhead due to the pushes and pops of the R0 - R7 registers.

Table 11-4: 8051/52 Interrupt Numbers in C

Interrupt	Name	Numbers used by 8051 C
External Interrupt 0	(INT0)	0
Timer Interrupt 0	(TF0)	1
External Interrupt 1	(INT1)	2
Timer Interrupt 1	(TF1)	3
Serial Communication	(RI + TI)	4
Timer 2 (8052 only)	(TF2)	5

Example 11-14 shows how a simple interrupt is written in 8051 C.

Example 11-14

Write a C program that continuously gets a single bit of data from P1.7 and sends it to P1.0, while simultaneously creating a square wave of 200 μs period on pin P2.5. Use timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

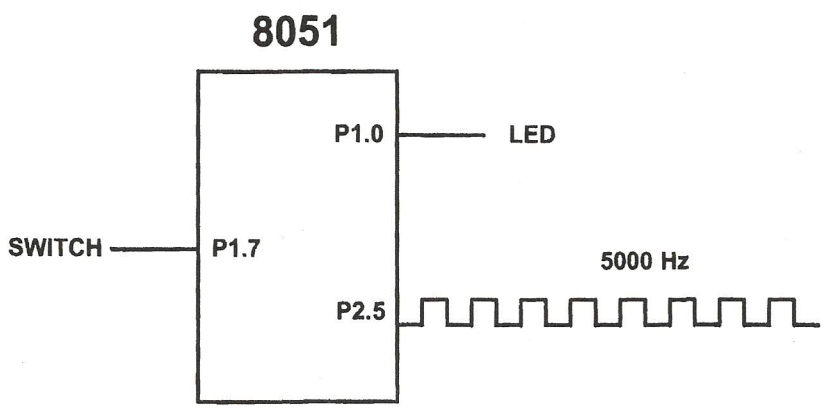
Solution:

We will use timer 0 in mode 2 (auto-reload). One half of the period is 100 μs.
100 /1. 085 μs = 92, and TH0 = 256 – 92 = 164 or A4H

```
#include <reg51.h>

sbit SW     = P1^7;
sbit IND    = P1^0;
sbit WAVE   = P2^5;

void timer0(void) interrupt 1
  {
    WAVE = ~WAVE;     //toggle pin
  }

void main()
  {
    SW = 1;           //make switch input
    TMOD = 0x02;
    TH0 = 0xA4;       //TH0 = -92
    IE = 0x82;        //enable interrupts for timer 0
    while(1)
      {
        IND = SW;     //send switch to LED
      }
  }
```

200 μs / 2 = 100 μs

100 μs / 1.085 μs = 92



Example 11-15

Write a C program that continuously gets a single bit of data from P1.7 and sends it to P1.0 in the main, while simultaneously (a) creating a square wave of 200 μs period on pin P2.5, and (b) sending letter 'A' to the serial port. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz. Use the 9600 baud rate.

Solution:

We will use Timer 0 in mode 2 (auto-reload). TH0 = 100/1.085 μs = –92, which is A4H

```
#include <reg51.h>

sbit SW    = P1^7;
sbit IND   = P1^0;
sbit WAVE  = P2^5;

void timer0(void) interrupt 1
  {
    WAVE = ~WAVE;     //toggle pin
  }

void serial0() interrupt 4
  {
    if(TI == 1)
      {
        SBUF = 'A';  //send A to serial port
        TI = 0;      //clear interrupt
      }
    else
      {
        RI = 0;      //clear interrupt
      }
  }

void main()
  {
    SW = 1;          //make switch input
    TH1 = -3;        //9600 baud
    TMOD = 0x22;     //mode 2 for both timers
    TH0 = 0xA4;      //-92=A4H for timer 0
    SCON = 0x50;
    TR0 = 1;
    TR1 = 1;              //start timer
    IE = 0x92;       //enable interrupt for T0
    while(1)         //stay here
      {
        IND = SW;    //send switch to LED
      }
  }
```

Example 11-16

Write a C program using interrupts to do the following:
(a) Receive data serially and send it to P0,
(b) Read port P1, transmit data serially, and give a copy to P2,
(c) Make timer 0 generate a square wave of 5 kHz frequency on P0.1.
Assume that XTAL = 11.0592 MHz. Set the baud rate at 4800.

Solution:

```
#include <reg51.h>
sbit WAVE = P0^1;

void timer0() interrupt 1
  {
    WAVE = ~WAVE;           //toggle pin
  }

void serial0() interrupt 4
  {
    if(TI == 1)
      {
        TI = 0;             //clear interrupt
      }
    else
      {
        P0 = SBUF;          //put value on pins
        RI = 0;             //clear interrupt
      }
  }

void main()
  {
    unsigned char x;
    P1 = 0xFF;              //make P1 an input
    TMOD = 0x22;
    TH1 = 0xF6;             //4800 baud rate
    SCON = 0x50;
    TH0 = 0xA4;             //5 kHz has T = 200 µs
    IE = 0x92;              //enable interrupts
    TR1 = 1;                //start timer 1
    TR0 = 1;                //start timer 0
    while(1)
      {
        x = P1;             //read value from pins
        SBUF = x;           //put value in buffer
        P2 = x;             //write value to pins
      }
  }
```

Example 11-17

Write a C program using interrupts to do the following:
(a) Generate a 10000 Hz frequency on P2.1 using T0 8-bit auto-reload,
(b) Use timer 1 as an event counter to count up a 1-Hz pulse and display it on P0. The pulse is connected to EX1.

Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

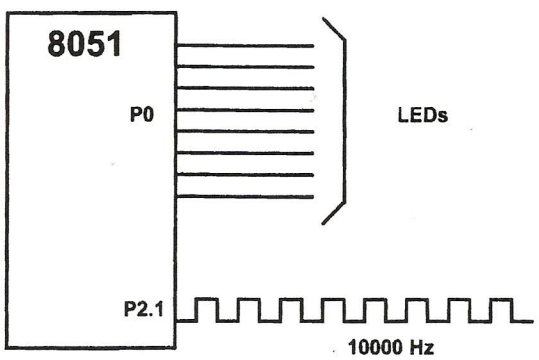
Solution:

```
#include <reg51.h>

sbit WAVE = P2^1;
unsigned char cnt;

void timer0() interrupt 1
  {
    WAVE = ~WAVE;           //toggle pin
  }
void timer1() interrupt 3
  {
    cnt++;                  //increment counter
    P0 = cnt;               //display value on pins
  }

void main()
  {
    cnt = 0;                //set counter to zero
    TMOD = 0x42;
    TH0 = 0x-46;            //10000 Hz
    IE = 0x86;              //enable interrupts
    TR0 = 1;                //start timer 0
    TR1 = 1;                //start timer 1
    while(1);               //wait until interrupted
  }
```

1 / 10000 Hz = 100 μs

100 μs / 2 = 50 μs

50 μs / 1.085 μs = 46



SUMMARY

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service. Every interrupt has a program associated with it called the ISR, or interrupt service routine. The 8051 has 6 interrupts, 5 of which are user-accessible. The interrupts are for reset: two for the timers, two for external hardware interrupts, and a serial communication interrupt. The 8052 has an additional interrupt for Timer 2.

The 8051 can be programmed to enable or disable an interrupt, and the interrupt priority can be altered. This chapter showed how to program 8051/52 interrupts in both Assembly and C languages.

PROBLEMS

SECTION 11.1: 8051 INTERRUPTS

1. Which technique, interrupt or polling, avoids tying down the microcontroller?
2. Including reset, how many interrupts does the 8051 have?
3. In the 8051 what memory area is assigned to the interrupt vector table?
4. True or false. The 8051 programmer cannot change the memory space assigned to the interrupt vector table.
5. What memory address in the interrupt vector table is assigned to INT0?
6. What memory address in the interrupt vector table is assigned to INT1?
7. What memory address in the interrupt vector table is assigned to Timer 0?
8. What memory address in the interrupt vector table is assigned to Timer 1?
9. What memory address in the interrupt vector table is assigned to the serial COM interrupt?
10. Why do we put an LJMP instruction at address 0?
11. What are the contents of the IE register upon reset, and what do these values mean?
12. Show the instruction to enable the EX1 and Timer 1 interrupts.
13. Show the instruction to enable every interrupt of the 8051.
14. Which pin of the 8051 is assigned to the external hardware interrupts INT0 and INT1?
15. How many bytes of address space in the interrupt vector table are assigned to the INT0 and INT1 interrupts?
16. How many bytes of address space in the interrupt vector table are assigned to the Timer 0 and Timer 1 interrupts?
17. To put the entire interrupt service routine in the interrupt vector table, it must be no more than _____ bytes in size.
18. True or false. The IE register is not a bit-addressable register.
19. With a single instruction, show how to disable all the interrupts.
20. With a single instruction, show how to disable the EX1 interrupt.
21. True or false. Upon reset, all interrupts are enabled by the 8051.
22. In the 8051, how many bytes of ROM space are assigned to the reset interrupt, and why?

SECTION 11.2: PROGRAMMING TIMER INTERRUPTS

23. True or false. For both Timer 0 and Timer 1, there is an interrupt assigned to it in the interrupt vector table.
24. What address in the interrupt vector table is assigned to Timer 1?
25. Which bit of IE belongs to the Timer 0 interrupt? Show how it is enabled.
26. Which bit of IE belongs to the Timer 1 interrupt? Show how it is enabled.
27. Assume that Timer 0 is programmed in mode 2, TH1 = F0H, and the IE bit for Timer 0 is enabled. Explain how the interrupt for the timer works.
28. True or false. The last two instructions of the ISR for Timer 1 are:

```
CLR  TF1
RETI
```

29. Assume that Timer 1 is programmed for mode 1, TH0 = FFH, TL1 = F8H, and the IE bit for Timer 1 is enabled. Explain how the interrupt is activated.
30. If Timer 1 is programmed for interrupts in mode 2, explain when the interrupt is activated.
31. Write a program to create a square wave of T = 160 ms on pin P2.2 while at the same time the 8051 is sending out 55H and AAH to P1 continuously.
32. Write a program in which every 2 seconds, the LED connected to P2.7 is turned on and off four times, while at the same time the 8051 is getting data from P1 and sending it to P0 continuously. Make sure the on and off states are 50 ms in duration.

SECTION 11.3: PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

33. True or false. A single interrupt is assigned to each of the external hardware interrupts EX0 and EX1.
34. What address in the interrupt vector table is assigned to INT0 and INT1? How about the pin numbers on port 3?
35. Which bit of IE belongs to the EX0 interrupt? Show how it is enabled.
36. Which bit of IE belongs to the EX1 interrupt? Show how it is enabled.
37. Show how to enable both external hardware interrupts.
38. Assume that the IE bit for external hardware interrupt EX0 is enabled and is low-level triggered. Explain how this interrupt works when it is activated. How can we make sure that a single interrupt is not interpreted as multiple interrupts?
39. True or false. Upon reset, the external hardware interrupt is edge-triggered.
40. In Question 39, how do we make sure that a single interrupt is not recognized as multiple interrupts?
41. Which bits of TCON belong to EX0?
42. Which bits of TCON belong to EX1?
43. True or false. The last two instructions of the ISR for INT1 are:

```
CLR  TCON.3
RETI
```

44. Explain the role of TCON.0 and TCON.2 in the execution of external interrupt 0.
45. Explain the role of TCON.1 and TCON.3 in the execution of external interrupt 1.
46. Assume that the IE bit for external hardware interrupt EX1 is enabled and is edge-triggered. Explain how this interrupt works when it is activated. How can

we make sure that a single interrupt is not interpreted as multiple interrupts?
47. Write a program using interrupts to get data from P1 and send it to P2 while Timer 0 is generating a square wave of 3 kHz.
48. Write a program using interrupts to get data from P1 and send it to P2 while Timer 1 is turning on and off the LED connected to P0.4 every second.
49. Explain the difference between the low-level and edge-triggered interrupts.
50. How do we make the hardware interrupt edge-triggered?
51. Which interrupts are latched, low-level or edge-triggered?
52. Which register keeps the latched interrupt for INT0 and INT1?

SECTION 11.4: PROGRAMMING THE SERIAL COMMUNICATION INTERRUPT

53. True or false. There are two interrupts assigned to interrupts TI and RI.
54. What address in the interrupt vector table is assigned to the serial interrupt? How many bytes are assigned to it?
55. Which bit of the IE register belongs to the serial interrupt? Show how it is enabled.
56. Assume that the IE bit for the serial interrupt is enabled. Explain how this interrupt gets activated and also explain its working upon activation.
57. True or false. Upon reset, the serial interrupt is blocked.
58. True or false. The last two instructions of the ISR for the receive interrupt are:
```
CLR  TI
RETI
```
59. Answer Question 58 for the receive interrupt.
60. Assuming that the interrupt bit in the IE register is enabled, when TI is raised, what happens subsequently?
61. Assuming that the interrupt bit in the IE register is enabled, when RI is raised, what happens subsequently?
62. Write a program using interrupts to get data serially and send it to P2 while at the same time Timer 0 is generating a square wave of 5 kHz.
63. Write a program using interrupts to get data serially and send it to P2 while Timer 0 is turning the LED connected to P1.6 on and off every second.

SECTION 11.5: INTERRUPT PRIORITY IN THE 8051/52

64. True or false. Upon reset, EX1 has the highest priority.
65. What register keeps track of interrupt priority in the 8051? Explain its role.
66. Which bit of IP belongs to the EX2 interrupt priority? Show how to assign it the highest priority.
67. Which bit of IP belongs to the Timer 1 interrupt priority? Show how to assign it the highest priority.
68. Which bit of IP belongs to the EX1 interrupt priority? Show how to assign it the highest priority.
69. Assume that the IP register has all 0s. Explain what happens if both INT0 and INT1 are activated at the same time.
70. Assume that the IP register has all 0s. Explain what happens if both TF0 and TF1 are activated at the same time.

71. If both TF0 and TF1 in the IP are set to high, what happens if both are activated at the same time?
72. If both INT0 and INT1 in the IP are set to high, what happens if both are activated at the same time?
73. Explain what happens if a low-priority interrupt is activated while the 8051 is serving a higher-priority interrupt.

ANSWERS TO REVIEW QUESTIONS

SECTION 11.1: 8051 INTERRUPTS

1. Interrupts
2. 5
3. Address locations 0000 to 25H. No. They are set when the processor is designed.
4. All 0s means that all interrupts are masked, and as a result no interrupts will be responded to by the 8051.
5. `MOV IE,#10000011B`
6. P3.3, which is pin 13 on the 40-pin DIP package
7. 0013H for INT1 and 001BH for Timer 1

SECTION 11.2: PROGRAMMING TIMER INTERRUPTS

1. False. There is an interrupt for each of the timers, Timer 0 and Timer 1.
2. 000BH
3. Bits D1 and D3 and "`MOV IE,#10001010B`" will enable both of the timer interrupts.
4. After Timer 1 is started with instruction "`SETB TR1`", the timer will count up from F5H to FFH on its own while the 8051 is executing other tasks. Upon rolling over from FFH to 00, the TF1 flag is raised, which will interrupt the 8051 in whatever it is doing and force it to jump to memory location 001BH to execute the ISR belonging to this interrupt.
5. False. There is no need for "`CLR TF0`" since the RETI instruction does that for us.

SECTION 11.3: PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

1. False. There is an interrupt for each of the external hardware interrupts of INT0 and INT1.
2. 0003H and 0013H. The pins numbered 12 (P3.2) and 13 (P3.3) on the DIP package.
3. Bits D0 and D2 and "`MOV IE,#10000101B`" will enable both of the external hardware interrupts.
4. Upon application of a low pulse (4 machine cycles wide) to pin P3.3, the 8051 is interrupted in whatever it is doing and jumps to ROM location 0013H to execute the ISR.
5. True
6. Make sure that the low pulse applied to pin INT1 is no wider than 4 machine cycles. Or, make sure that the INT1 pin is brought back to high by the time the 8051 executes the RETI instruction in the ISR.
7. False. There is no need for the "`CLR TCON.0`" since the RETI instruction does that for us.
8. TCON.0 is set to high to make INT0 an edge-triggered interrupt. If INT0 is edge-triggered (that is, TCON.0 is set), whenever a high-to-low pulse is applied to the INT0 pin it is captured (latched) and kept by the TCON.2 bit by making TCON.2 high. While the ISR for INT0 is being serviced, TCON.2 stays high no matter how many times an H-to-L pulse is applied to pin INT0. Upon the execution of the last instruction of the ISR, which is RETI, the TCON.2 bit is cleared, indicating that the INT0 pin can respond to another interrupt.