CHAPTER 9

8051 TIMER PROGRAMMING IN ASSEMBLY AND C

OBJECTIVES

Upon completion of this chapter, you will be able to:

- **>> List the timers of the 8051 and their associated registers**
- **>> Describe the various modes of the 8051 timers**
- **>> Program the 8051 timers in Assembly and C to generate time delays**
- **>> Program the 8051 counters in Assembly and C as event counters**

The 8051 has two timers/counters. They can be used either as timers to generate a time delay or as counters to count events happening outside the microcontroller. In Section 9.1 we see how these timers are used to generate time delays. In Section 9.2 we show how they are used as event counters. In Section 9.3 we use C language to program the 8051 timers.

SECTION 9.1: PROGRAMMING 8051 TIMERS

The 8051 has two timers: Timer 0 and Timer 1. They can be used either as timers or as event counters. In this section we first discuss the timers' registers and then show how to program the timers to generate time delays.

Basic registers of the timer

Both Timer 0 and Timer 1 are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte. Each timer is discussed separately.

Timer 0 registers

The 16-bit register of Timer 0 is accessed as low byte and high byte. The low byte register is called TL0 (Timer 0 low byte) and the high byte register is referred to as TH0 (Timer 0 high byte). These registers can be accessed like any other register, such as A, B, R0, R1, R2, etc. For example, the instruction "`MOV TL0,#4FH`" moves the value 4FH into TL0, the low byte of Timer 0. These registers can also be read like any other register. For example, "`MOV R5,TH0`" saves TH0 (high byte of Timer 0) in R5.

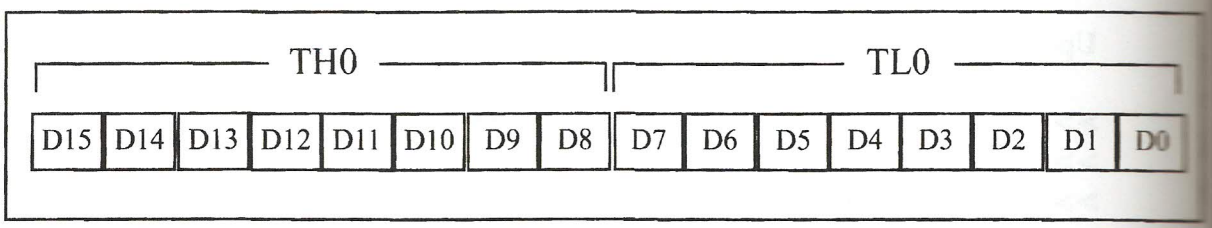


Figure 9-1. Timer 0 Registers

Timer 1 registers

Timer 1 is also 16 bits, and its 16-bit register is split into two bytes, referred to as TL1 (Timer 1 low byte) and TH1 (Timer 1 high byte). These registers are accessible in the same way as the registers of Timer 0.

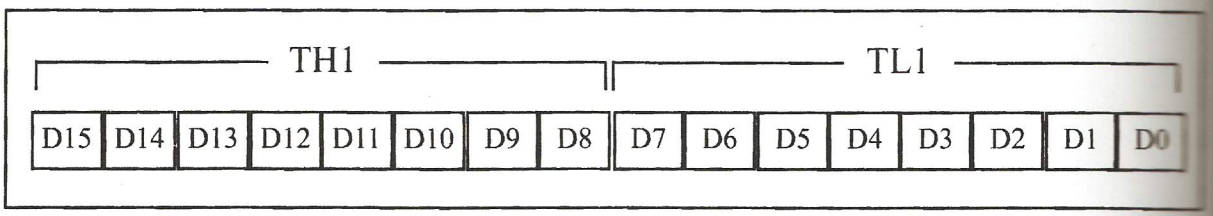


Figure 9-2. Timer 1 Registers

TMOD (timer mode) register

Both timers 0 and 1 use the same register, called TMOD, to set the various timer operation modes. TMOD is an 8-bit register in which the lower 4 bits are set aside for Timer 0 and the upper 4 bits for Timer 1. In each case, the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation. These options are discussed next.

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

GATE Gating control when set. The timer/counter is enabled only while the INTx pin is high and the TRx control pin is set. When cleared, the timer is enabled whenever the TRx control bit is set.

C/T Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

M1 Mode bit 1

M0 Mode bit 0

M1	M0	Mode	Operating Mode
0	0	0	13-bit timer mode 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	16-bit timer mode 16-bit timer/counters THx and TLx are cascaded; there is no prescaler
1	0	2	8-bit auto reload 8-bit auto reload timer/counter; THx holds a value that is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

Figure 9-3. TMOD Register

M1, M0

M0 and M1 select the timer mode. As shown in Figure 9-3, there are three modes: 0, 1, and 2. Mode 0 is a 13-bit timer, mode 1 is a 16-bit timer, and mode 2 is an 8-bit timer. We will concentrate on modes 1 and 2 since they are the ones used most widely. We will soon describe the characteristics of these modes, after describing the rest of the TMOD register.

C/T (clock/timer)

This bit in the TMOD register is used to decide whether the timer is used as a delay generator or an event counter. If C/T = 0, it is used as a timer for time delay generation. The clock source for the time delay is the crystal frequency of the 8051. This section is concerned with this choice. The timer's use as an event counter is discussed in the next section.

Example 9-1

Indicate which mode and which timer are selected for each of the following.
(a) `MOV TMOD,#01H` (b) `MOV TMOD,#20H` (c) `MOV TMOD,#12H`

Solution:

We convert the values from hex to binary. From Figure 9-3 we have:

(a) TMOD = 00000001, mode 1 of Timer 0 is selected.
(b) TMOD = 00100000, mode 2 of Timer 1 is selected.
(c) TMOD = 00010010, mode 2 of Timer 0, and mode 1 of Timer 1 are selected.

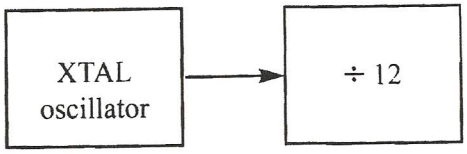
Clock source for timer

As you know, every timer needs a clock pulse to tick. What is the source of the clock pulse for the 8051 timers? If C/T = 0, the crystal frequency attached to the 8051 is the source of the clock for the timer. This means that the size of the crystal frequency attached to the 8051 also decides the speed at which the 8051 timer ticks. The frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051. See Example 9-2.

Example 9-2

Find the timer's clock frequency and its period for various 8051-based systems, with the following crystal frequencies.
(a) 12 MHz
(b) 16 MHz
(c) 11.0592 MHz

Solution:



(a) 1/12 × 12 MHz = 1 MHz and T = 1/1 MHz = 1 µs

(b) 1/12 × 16 MHz = 1.333 MHz and T = 1/1.333 MHz = .75 µs

(c) 1/12 × 11.0592 MHz = 921.6 kHz;
T = 1/921.6 kHz = 1.085 µs

NOTE THAT 8051 TIMERS USE 1/12 OF XTAL FREQUENCY, REGARDLESS OF MACHINE CYCLE TIME.

Although various 8051-based systems have an XTAL frequency of 10 MHz to 40 MHz, we will concentrate on the XTAL frequency of 11.0592 MHz. The reason behind such an odd number has to do with the baud rate for serial communication of the 8051. XTAL = 11.0592 MHz allows the 8051 system to communicate with the IBM PC with no errors, as we will see in Chapter 10.

GATE

The other bit of the TMOD register is the GATE bit. Notice in the TMOD register of Figure 9-3 that both Timers 0 and 1 have the GATE bit. What is its purpose? Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The timers in the 8051 have both. The start and stop of the timer are controlled by way of software by the TR (timer start) bits TR0 and TR1. This is achieved by the instructions "`SETB TR1`" and "`CLR TR1`" for Timer 1, and "`SETB TR0`" and "`CLR TR0`" for Timer 0. The SETB instruction starts it, and it is stopped by the CLR instruction. These instructions start and stop the timers as long as GATE = 0 in the TMOD register. The hardware way of starting and stopping the timer by an external source is achieved by making GATE = 1 in the TMOD register. However, to avoid further confusion for now, we will make GATE = 0, meaning that no external hardware is needed to start and stop the timers. In using software to start and stop the timer where GATE = 0, all we need are the instructions "`SETB TRx`" and "`CLR TRx`". The use of external hardware to stop or start the timer is discussed in Chapter 11 when interrupts are discussed.

Example 9-3

Find the value for TMOD if we want to program Timer 0 in mode 2, use 8051 XTAL for the clock source, and use instructions to start and stop the timer.

Solution:

TMOD= 0000 0010 Timer 0, mode 2,
C/T = 0 to use XTAL clock source, and
gate = 0 to use internal (software)
start and stop method.

Now that we have this basic understanding of the role of the TMOD register, we will look at the timer's modes and how they are programmed to create a time delay. Because modes 1 and 2 are so widely used, we describe each of them in detail.

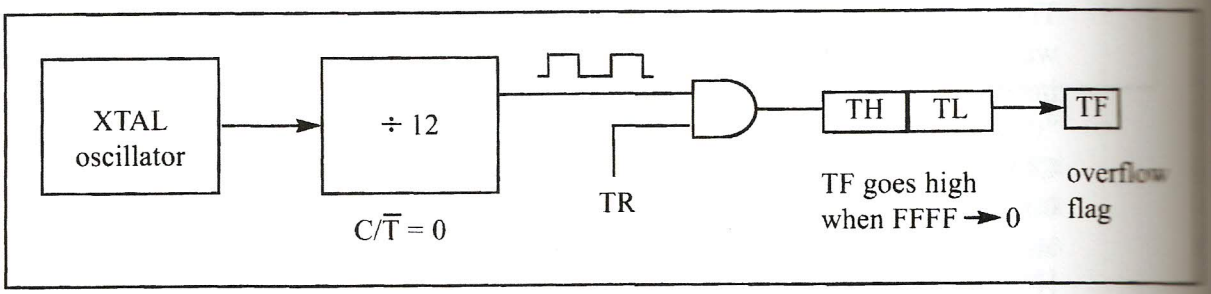
Mode 1 programming

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer; therefore, it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH.
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by "`SETB TR0`" for Timer 0 and "`SETB TR1`" for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its

limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions "`CLR TR0`" or "`CLR TR1`", for Timer 0 and Timer 1, respectively. Again, it must be noted that each timer has its own timer flag: TF0 for Timer 0, and TF1 for Timer 1.

4. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value, and TF must be reset to 0.



Steps to program in mode 1

To generate a time delay, using the timer's mode 1, the following steps are taken. To clarify these steps, see Example 9-4.

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used and which timer mode (0 or 1) is selected.
2. Load registers TL and TH with initial count values.
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the "`JNB TFx, target`" instruction to see if it is raised. Get out of the loop when TF becomes high.
5. Stop the timer.
6. Clear the TF flag for the next round.
7. Go back to Step 2 to load TH and TL again.

To calculate the exact time delay and the square wave frequency generated on pin P1.5, we need to know the XTAL frequency. See Example 9-5.

From Example 9-6 we can develop a formula for delay calculations using mode 1 (16-bit) of the timer for a crystal frequency of XTAL = 11.0592 MHz. This is given in Figure 9-4. The scientific calculator in the Accessories directory of Microsoft Windows can help you to find the TH, TL values. This calculator supports decimal, hex, and binary calculations.

(a) in hex	**(b) in decimal**
(FFFF - YYXX + 1) × 1.085 µs where YYXX are TH, TL initial values respectively. Notice that values YYXX are in hex.	Convert YYXX values of the TH,TL register to decimal to get a NNNNN decimal number, then(65536 - NNNNN) × 1.085 µs

Figure 9-4. Timer Delay Calculation for XTAL = 11.0592 MHz

Example 9-4

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program.

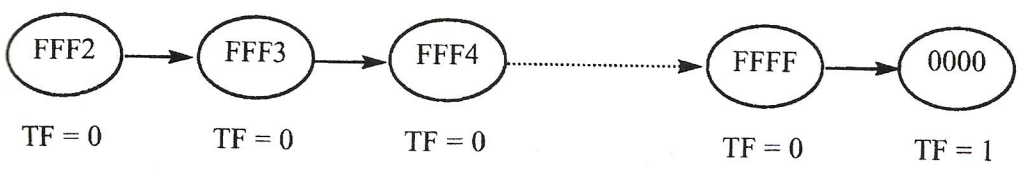
```
            MOV    TMOD,#01          ;Timer 0, mode 1(16-bit mode)
HERE:       MOV    TL0,#0F2H         ;TL0 = F2H, the Low byte
            MOV    TH0,#0FFH         ;TH0 = FFH, the High byte
            CPL    P1.5              ;toggle P1.5
            ACALL  DELAY
            SJMP   HERE              ;load TH, TL again
;----------delay using Timer 0
DELAY:
            SETB   TR0               ;start Timer 0
AGAIN:      JNB    TF0,AGAIN         ;monitor Timer 0 flag until
                                     ;it rolls over
            CLR    TR0               ;stop Timer 0
            CLR    TF0               ;clear Timer 0 flag
            RET
```

Solution:

In the above program notice the following steps.

1. TMOD is loaded.
2. FFF2H is loaded into TH0 - TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, Timer 0 is started by the "`SETB TR0`" instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0 = 1). At that point, the JNB instruction falls through.
7. Timer 0 is stopped by the instruction "`CLR TR0`". The DELAY subroutine ends, and the process is repeated.

Notice that to repeat the process, we must reload the TL and TH registers and start the timer again.



Example 9-5

In Example 9-4, calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume that XTAL = 11.0592 MHz.

Solution:

The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency. As a result, each clock has a period of T = 1 / 921.6 kHz = 1.085 μs. In other words, Timer 0 counts up each 1.085 μs resulting in delay = number of counts × 1.085 μs.
The number of counts for the rollover is FFFFH – FFF2H = 0DH (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raises the TF flag. This gives 14 × 1.085 μs = 15.19 μs for half the pulse. For the entire period T = 2 × 15.19 μs = 30.38 μs gives us the time delay generated by the timer.

Example 9-6

In Example 9-5, calculate the frequency of the square wave generated on pin P1.5.

Solution:

In the time delay calculation of Example 9-5, we did not include the overhead due to instructions in the loop. To get a more accurate timing, we need to add clock cycles due to the instructions in the loop. To do that, we use the machine cycles from Table A-1 in Appendix A, as shown below.

```
                                          Cycles
HERE:     MOV   TL0,#0F2H                   2
          MOV   TH0,#0FFH                   2
          CPL   P1.5                        1
          ACALL DELAY                       2
          SJMP  HERE                        2
;-----------delay using Timer 0
DELAY:
          SETB  TR0                         1
AGAIN:    JNB   TF0,AGAIN                   14
          CLR   TR0                         1
          CLR   TF0                         1
          RET                               2
                                Total       28
```

T = 2 × 28 × 1.085 μs = 60.76 μs and F = 16458.2 Hz.

NOTE THAT 8051 TIMERS USE 1/12 OF XTAL FREQUENCY, REGARDLESS OF MACHINE CYCLE TIME.

Example 9-7

Find the delay generated by Timer 0 in the following code, using both of the methods of Figure 9-4. Do not include the overhead due to instructions.

```
        CLR   P2.3          ;clear P2.3
        MOV   TMOD,#01      ;Timer 0, mode 1(16-bit mode)
HERE:   MOV   TL0,#3EH      ;TL0 = 3EH, Low byte
        MOV   TH0,#0B8H     ;TH0 = B8H, High byte
        SETB  P2.3          ;SET high P2.3
        SETB  TR0           ;start Timer 0
AGAIN:  JNB   TF0,AGAIN     ;monitor Timer 0 flag
        CLR   TR0           ;stop Timer 0
        CLR   TF0           ;clear Timer 0 flag for
                            ;next round
        CLR   P2.3
```

Solution:

(a) (FFFF - B83E + 1) = 47C2H = 18370 in decimal and 18370 × 1.085 μs = 19.93145 ms.

(b) Since TH - TL = B83EH = 47166 (in decimal) we have 65536 - 47166 = 18370. This means that the timer counts from B83EH to FFFFH. This plus rolling over to 0 goes through a total of 18370 clock cycles, where each clock is 1.085 μs in duration. Therefore, we have 18370 × 1.085 μs = 19.93145 ms as the width of the pulse.

Example 9-8

Modify TL and TH in Example 9-7 to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop.

Solution:

To get the largest delay we make TL and TH both 0. This will count up from 0000 to FFFFH and then roll over to zero.

```
        CLR   P2.3          ;clear P2.3
        MOV   TMOD,#01      ;Timer 0, mode 1(16-bit mode)
HERE:   MOV   TL0,#0        ;TL0 = 0, Low byte
        MOV   TH0,#0        ;TH0 = 0, High byte
        SETB  P2.3          ;SET P2.3 high
        SETB  TR0           ;start Timer 0
AGAIN:  JNB   TF0,AGAIN     ;monitor Timer 0 flag
        CLR   TR0           ;stop Timer 0
        CLR   TF0           ;clear Timer 0 flag
        CLR   P2.3
```

Making TH and TL both zero means that the timer will count from 0000 to FFFFH, and then roll over to raise the TF flag. As a result, it goes through a total of 65536 states. Therefore, we have delay = (65536 - 0) × 1.085 μs = 71.1065 ms.

Example 9-9

The following program generates a square wave on pin P1.5 continuously using Timer 1 for a time delay. Find the frequency of the square wave if XTAL = 11.0592 MHz. In your calculation do not include the overhead due to instructions in the loop.

```
        MOV   TMOD,#10H       ;Timer 1, mode 1(16-bit)
AGAIN:  MOV   TL1,#34H        ;TL1 = 34H, Low byte
        MOV   TH1,#76H        ;TH1 = 76H, High byte
                              ;(7634H = timer value)
        SETB  TR1             ;start Timer 1
BACK:   JNB   TF1,BACK        ;stay until timer rolls over
        CLR   TR1             ;stop Timer 1
        CPL   P1.5            ;comp. P1.5 to get hi, lo
        CLR   TF1             ;clear Timer 1 flag
        SJMP  AGAIN           ;reload timer since Mode 1
                              ;is not auto-reload
```

Solution:

In the above program notice the target of SJMP. In mode 1, the program must reload the TH, TL register every time if we want to have a continuous wave. Now the calculation. Since FFFFH - 7634H = 89CBH + 1 = 89CCH and 89CCH = 35276 clock count. 35276 × 1.085 μs = 38.274 ms for half of the square wave. The entire square wave length is 38.274 × 2 = 76.548 ms and has a frequency = 13.064 Hz.

Also notice that the high and low portions of the square wave pulse are equal. In the above calculation, the overhead due to all the instructions in the loop is not included.

In Examples 9-7 and 9-8, we did not reload TH and TL since it was a single pulse. Look at Example 9-9 to see how the reloading works in mode 1.

Finding values to be loaded into the timer

Assuming that we know the amount of timer delay we need, the question is how to find the values needed for the TH, TL registers. To calculate the values to be loaded into the TL and TH registers look at Example 9-10 where we use crystal frequency of 11.0592 MHz for the 8051 system.

Assuming XTAL = 11.0592 MHz from Example 9-10 we can use the following steps for finding the TH, TL registers' values.

1. Divide the desired time delay by 1.085 μs.
2. Perform 65536 – *n*, where *n* is the decimal value we got in Step 1.
3. Convert the result of Step 2 to hex, where *yyxx* is the initial hex value to be loaded into the timer's registers.
4. Set TL = *xx* and TH = *yy*.

Example 9-10

Assume that XTAL = 11.0592 MHz. What value do we need to load into the timer's registers if we want to have a time delay of 5 ms (milliseconds)? Show the program for Timer 0 to create a pulse width of 5 ms on P2.3.

Solution:

Since XTAL = 11.0592 MHz, the counter counts up every 1.085 μs. This means that out of many 1.085 μs intervals we must make a 5 ms pulse. To get that, we divide one by the other. We need 5 ms / 1.085 μs = 4608 clocks. To achieve that we need to load into TL and TH the value 65536 - 4608 = 60928 = EE00H. Therefore, we have TH = EE and TL = 00.

```
        CLR   P2.3          ;clear P2.3
        MOV   TMOD,#01      ;Timer 0, mode 1 (16-bit mode)
HERE:   MOV   TL0,#0        ;TL0 = 0, Low byte
        MOV   TH0,#0EEH     ;TH0 = EE( hex), High byte
        SETB  P2.3          ;SET P2.3 high
        SETB  TR0           ;start Timer 0
AGAIN:  JNB   TF0,AGAIN     ;monitor Timer 0 flag
                            ;until it rolls over
        CLR   P2.3          ;clear P2.3
        CLR   TR0           ;stop Timer 0
        CLR   TF0           ;clear Timer 0 flag
```

Example 9-11

Assuming that XTAL = 11.0592 MHz, write a program to generate a square wave of 2 kHz frequency on pin P1.5.

Solution:

This is similar to Example 9-10, except that we must toggle the bit to generate the square wave. Look at the following steps.

(a) T = 1 / f = 1 / 2 kHz = 500 μs the period of the square wave.

(b) 1/2 of it for the high and low portions of the pulse is 250 μs.

(c) 250 μs / 1.085 μs = 230 and 65536 - 230 = 65306, which in hex is FF1AH.

(d) TL = 1AH and TH = FFH, all in hex. The program is as follows.

```
        MOV   TMOD,#10H     ;Timer 1, mode 1(16-bit)
AGAIN:  MOV   TL1,#1AH      ;TL1=1AH, Low byte
        MOV   TH1,#0FFH     ;TH1=FFH, High byte
        SETB  TR1           ;start Timer 1
BACK:   JNB   TF1,BACK      ;stay until timer rolls over
        CLR   TR1           ;stop Timer 1
        CPL   P1.5          ;complement P1.5 to get hi, lo
        CLR   TF1           ;clear Timer 1 flag
        SJMP  AGAIN         ;reload timer since mode 1
                            ;is not auto-reload
```

Example 9-12

Assuming XTAL = 11.0592 MHz, write a program to generate a square wave of 50 Hz frequency on pin P2.3.

Solution:

Look at the following steps.
(a) T = 1 / 50 Hz = 20 ms, the period of the square wave.
(b) 1/2 of it for the high and low portions of the pulse = 10 ms
(c) 10 ms / 1.085 µs = 9216 and 65536 - 9216 = 56320 in decimal, and in hex it is DC00H.
(d) TL = 00 and TH = DC (hex)

The program follows.

```
        MOV   TMOD,#10H      ;Timer 1, mode 1 (16-bit)
AGAIN:  MOV   TL1,#00        ;TL1 = 00, Low byte
        MOV   TH1,#0DCH      ;TH1 = DCH, High byte
        SETB  TR1            ;start Timer 1
BACK:   JNB   TF1,BACK       ;stay until timer rolls over
        CLR   TR1            ;stop Timer 1
        CPL   P2.3           ;comp. P2.3 to get hi, lo
        CLR   TF1            ;clear Timer 1 flag
        SJMP  AGAIN          ;reload timer since mode 1
                             ;is not auto-reload
```

Generating a large time delay

As we have seen in the examples so far, the size of the time delay depends on two factors, (a) the crystal frequency, and (b) the timer's 16-bit register in mode 1. Both of these factors are beyond the control of the 8051 programmer. We saw earlier that the largest time delay is achieved by making both TH and TL zero. What if that is not enough? Example 9-13 shows how to achieve large time delays.

Using Windows calculator to find TH, TL

The scientific calculator in Microsoft Windows is a handy and easy-to-use tool to find the TH, TL values. Assume that we would like to find the TH, TL values for a time delay that uses 35,000 clocks of 1.085 µs. The following steps show the calculation.

1. Bring up the scientific calculator in MS Windows and select decimal.
2. Enter 35,000.
3. Select hex. This converts 35,000 to hex, which is 88B8H.
4. Select +/- to give -35000 decimal (7748H).
5. The lowest two digits (48) of this hex value are for TL and the next two (77) are for TH. We ignore all the Fs on the left since our number is 16-bit data.

Example 9-13

Examine the following program and find the time delay in seconds. Exclude the overhead due to the instructions in the loop.

```
        MOV   TMOD,#10H       ;Timer 1, mode 1(16-bit)
        MOV   R3,#200         ;counter for multiple delay
AGAIN:  MOV   TL1,#08H        ;TL1 = 08, Low byte
        MOV   TH1,#01H        ;TH1 = 01, High byte
        SETB  TR1             ;start Timer 1
BACK:   JNB   TF1,BACK        ;stay until timer rolls over
        CLR   TR1             ;stop Timer 1
        CLR   TF1             ;clear Timer 1 flag
        DJNZ  R3,AGAIN        ;if R3 not zero then
                              ;reload timer
```

Solution:

TH - TL = 0108H = 264 in decimal and 65536 - 264 = 65272. Now 65272 × 1.085 μs = 70.820 ms, and for 200 of them we have 200 × 70.820 ms = 14.164024 seconds.

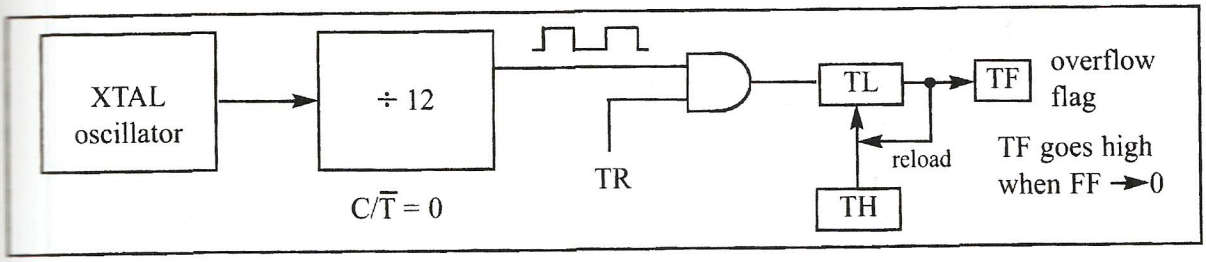
Mode 0

Mode 0 is exactly like mode 1 except that it is a 13-bit timer instead of 16-bit. The 13-bit counter can hold values between 0000 to 1FFFH in TH – TL. Therefore, when the timer reaches its maximum of 1FFH, it rolls over to 0000, and TF is raised.

Mode 2 programming

The following are the characteristics and operations of mode 2.

1. It is an 8-bit timer; therefore, it allows only values of 00 to FFH to be loaded into the timer's register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started. This is done by the instruction "`SETB TR0`" for Timer 0 and "`SETB TR1`" for Timer 1. This is just like mode 1.
3. After the timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TF (timer flag). If we are using Timer 0, TF0 goes high; if we are using Timer 1, TF1 is raised.



4. When the TL register rolls from FFH to 0 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL.

It must be emphasized that mode 2 is an 8-bit timer. However, it has an auto-reloading capability. In auto-reload, TH is loaded with the initial count and a copy of it is given to TL. This reloading leaves TH unchanged, still holding a copy of the original value. This mode has many applications, including setting the baud rate in serial communication, as we will see in Chapter 10.

Steps to program in mode 2

To generate a time delay using the timer's mode 2, take the following steps.

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used, and select the timer mode (mode 2).
2. Load the TH registers with the initial count value.
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the "`JNB TFx, target`" instruction to see whether it is raised. Get out of the loop when TF goes high.
5. Clear the TF flag.
6. Go back to Step 4, since mode 2 is auto-reload.

Example 9-14 illustrates these points. To achieve a larger delay, we can use multiple registers as shown in Example 9-15.

Example 9-14

Assuming that XTAL = 11.0592 MHz, find (a) the frequency of the square wave generated on pin P1.0 in the following program, and (b) the smallest frequency achievable in this program, and the TH value to do that.

```
        MOV   TMOD,#20H      ;T1/mode 2/8-bit/auto-reload
        MOV   TH1,#5         ;TH1 = 5
        SETB  TR1            ;start Timer 1
BACK:   JNB   TF1,BACK       ;stay until timer rolls over
        CPL   P1.0           ;comp. P1.0 to get hi, lo
        CLR   TF1            ;clear Timer 1 flag
        SJMP  BACK           ;mode 2 is auto-reload
```

Solution:

(a) First notice the target address of SJMP. In mode 2 we do not need to reload TH since it is auto-reload. Now (256 - 05) × 1.085 μs = 251 × 1.085 μs = 272.33 μs is the high portion of the pulse. Since it is a 50% duty cycle square wave, the period T is twice that; as a result T = 2 × 272.33 μs = 544.67 μs and the frequency = 1.83597 kHz.

(b) To get the smallest frequency, we need the largest T and that is achieved when TH = 00. In that case, we have T = 2 × 256 × 1.085 μs = 555.52 μs and the frequency = 1.8 kHz.

Example 9-15

Find the frequency of a square wave generated on pin P1.0.

Solution:

```
        MOV   TMOD,#2H          ;Timer 0, mode 2
                                ;(8-bit, auto-reload)
        MOV   TH0,#0            ;TH0=0
AGAIN:  MOV   R5,#250           ;count for multiple delay
        ACALL DELAY
        CPL   P1.0              ;toggle P1.0
        SJMP  AGAIN             ;repeat
DELAY:  SETB  TR0               ;start Timer 0
BACK:   JNB   TF0,BACK          ;stay until timer rolls over
        CLR   TR0               ;stop Timer 0
        CLR   TF0               ;clear TF for next round
        DJNZ  R5,DELAY
        RET
```

T = 2 (250 × 256 × 1.085 μs) = 138.88 ms, and frequency = 72 Hz.

Example 9-16

Assuming that we are programming the timers for mode 2, find the value (in hex) loaded into TH for each of the following cases.

(a) `MOV TH1,#-200`
(b) `MOV TH0,#-60`
(c) `MOV TH1,#-3`
(d) `MOV TH1,#-12`
(e) `MOV TH0,#-48`

Solution:

You can use the Windows scientific calculator to verify the results provided by the assembler. In Windows calculator, select decimal and enter 200. Then select hex, then +/- to get the TH value. Remember that we only use the right two digits and ignore the rest since our data is an 8-bit data. The following is what we get.

Decimal	*2's complement (TH value)*
-200	38H
-60	C4H
-3	FDH
-12	F4H
-48	D0H

Assemblers and negative values

Since the timer is 8-bit in mode 2, we can let the assembler calculate the value for TH. For example, in "`MOV TH1,#-100`", the assembler will calculate the -100 = 9C, and makes THl = 9C in hex. This makes our job easier.

Example 9-17

Find (a) the frequency of the square wave generated in the following code, and (b) the duty cycle of this wave.

```
            MOV   TMOD,#2H          ;Timer 0, mode 2
                                    ;(8-bit, auto-reload)
            MOV   TH0,#-150         ;TH0 = 6AH = 2's comp of -150
AGAIN:      SETB  P1.3              ;P1.3 = 1
            ACALL DELAY
            ACALL DELAY
            CLR   P1.3              ;P1.3 = 0
            ACALL DELAY
            SJMP  AGAIN
DELAY:
            SETB  TR0               ;start Timer 0
BACK:       JNB   TF0,BACK          ;stay until timer rolls over
            CLR   TR0               ;stop Timer 0
            CLR   TF0               ;clear TF for next round
            RET
```

Solution:

For the TH value in mode 2, the conversion is done by the assembler as long as we enter a negative number. This also makes the calculation easy. Since we are using 150 clocks, we have time for the DELAY subroutine = 150 × 1.085 μs = 162 μs. The high portion of the pulse is twice that of the low portion (66% duty cycle). Therefore, we have: T = high portion + low portion = 325.5 μs + 162.25 μs = 488.25 μs and frequency = 2.048 kHz.

Notice that in many of the time delay calculations we have ignored the clocks caused by the overhead instructions in the loop. To get a more accurate time delay, and hence frequency, you need to include them. If you use a digital scope and you don't get exactly the same frequency as the one we have calculated, it is because of the overhead associated with those instructions.

In this section, we used the 8051 timer for time delay generation. However, a more powerful and creative use of these timers is to use them as event counters. We discuss this use of the counter next.

Review Questions

1. How many timers do we have in the 8051?
2. Each timer has _____ registers that are ___ bits wide.
3. TMOD register is a(n) ___-bit register.
4. True or false. The TMOD register is a bit-addressable register.
5. Indicate the selection made in the instruction "`MOV TMOD,#20H`".
6. In mode 1, the counter rolls over when it goes from ____ to ____.
7. In mode 2, the counter rolls over when it goes from ____ to ____.
8. In the instruction "`MOV TH1,#-200`", find the hex value for the TH register.
9. To get a 2-ms delay, what number should be loaded into TH, TL using mode 1? Assume that XTAL = 11.0592 MHz.
10. To get a 100-µs delay, what number should be loaded into the TH register using mode 2? Assume XTAL = 11.0592 MHz.

SECTION 9.2: COUNTER PROGRAMMING

In the last section we used the timer/counter of the 8051 to generate time delays. These timers can also be used as counters counting events happening outside the 8051. The use of the timer/counter as an event counter is covered in this section. As far as the use of a timer as an event counter is concerned, everything that we have talked about in programming the timer in the last section also applies to programming it as a counter, except the source of the frequency. When the timer/counter is used as a timer, the 8051's crystal is used as the source of the frequency. When it is used as a counter, however, it is a pulse outside the 8051 that increments the TH, TL registers. In counter mode, notice that the TMOD and TH, TL registers are the same as for the timer discussed in the last section; they even have the same names. The timer's modes are the same as well.

C/T bit in TMOD register

Recall from the last section that the C/T bit in the TMOD register decides the source of the clock for the timer. If C/T = 0, the timer gets pulses from the crystal. In contrast, when C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051. Therefore, when C/T = 1, the counter counts up as pulses are fed from pins 14 and 15. These pins are called T0 (Timer 0 input) and T1 (Timer 1 input). Notice that these two pins belong to port 3. In the case of Timer 0, when C/T = 1, pin P3.4 provides the clock pulse and the counter counts up for each clock pulse coming from that pin. Similarly, for Timer 1, when C/T = 1 each clock pulse coming in from pin P3.5 makes the counter count up.

Table 9-1: Port 3 Pins Used For Timers 0 and 1

Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/Counter 0 external input
15	P3.5	T1	Timer/Counter 1 external input

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

Example 9-18

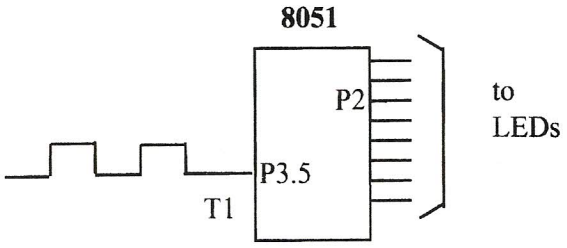
Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2.

Solution:

```
        MOV   TMOD,#01100000B     ;counter 1, mode 2,C/T=1
                                  ;external pulses
        MOV   TH1,#0              ;clear TH1
        SETB  P3.5                ;make T1 input
AGAIN:  SETB  TR1                 ;start the counter
BACK:   MOV   A,TL1               ;get copy of count TL1
        MOV   P2,A                ;display it on port 2
        JNB   TF1,BACK            ;keep doing it if TF=0
        CLR   TR1                 ;stop the counter 1
        CLR   TF1                 ;make TF=0
        SJMP  AGAIN               ;keep doing it
```

Notice in the above program the role of the instruction "SETB P3.5". Since ports are set up for output when the 8051 is powered up, we make P3.5 an input port by making it high. In other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.

P2 is connected to 8 LEDs and input T1 to pulse.



In Example 9-18, we use Timer 1 as an event counter where it counts up as clock pulses are fed into pin 3.5. These clock pulses could represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses.

In Example 9-18, the TL data was displayed in binary. In Example 9-19, the TL registers are converted to ASCII to be displayed on an LCD.

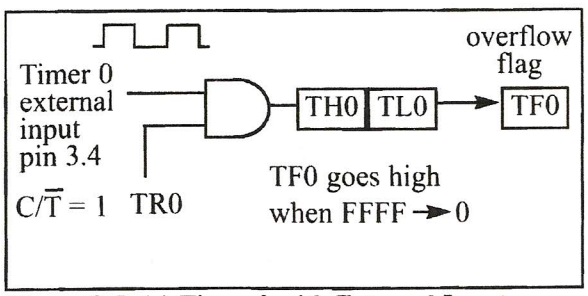
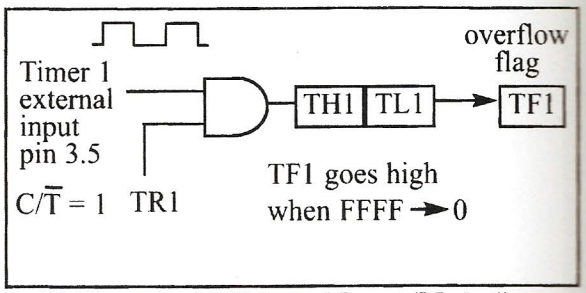


Figure 9-5. (a) Timer 0 with External Input (Mode 1)



(b) Timer 1 with External Input (Mode 1)

Example 9-19

Assume that a 1-Hz frequency pulse is connected to input pin 3.4. Write a program to display counter 0 on an LCD. Set the initial value of TH0 to -60.

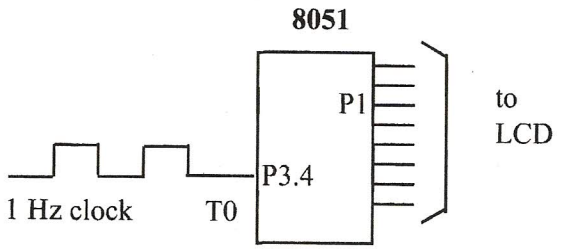
Solution:

To display the TL count on an LCD, we must convert 8-bit binary data to ASCII. See Chapter 6 for data conversion.

```
            ACALL  LCD_SET_UP       ;initialize the LCD
            MOV    TMOD,#00000110B  ;counter 0,mode 2,C/T=1
            MOV    TH0,#-60         ;counting 60 pulses
            SETB   P3.4             ;make T0 as input
AGAIN:      SETB   TR0              ;starts the counter
BACK:       MOV    A,TL0            ;get copy of count TL0
            ACALL  CONV             ;convert in R2, R3, R4
            ACALL  DISPLAY          ;display on LCD
            JNB    TF0,BACK         ;loop if TF0=0
            CLR    TR0              ;stop the counter 0
            CLR    TF0              ;make TF0=0
            SJMP   AGAIN            ;keep doing it

;converting 8-bit binary to ASCII
;upon return, R4, R3, R2 have ASCII data (R2 has LSD)

CONV:       MOV   B,#10             ;divide by 10
            DIV   AB
            MOV   R2,B              ;save low digit
            MOV   B,#10             ;divide by 10 once more
            DIV   AB
            ORL   A,#30H            ;make it ASCII
            MOV   R4,A              ;save MSD
            MOV   A,B
            ORL   A,#30H            ;make 2nd digit an ASCII
            MOV   R3,A              ;save it
            MOV   A,R2
            ORL   A,#30H            ;make 3rd digit an ASCII
            MOV   R2,A              ;save the ASCII
            RET
```



By using 60 Hz we can generate seconds, minutes, hours.

Note that on the first round, it starts from 0, since on RESET, TL0 = 0.
To solve this problem, load TL0 with -60 at the beginning of the program.

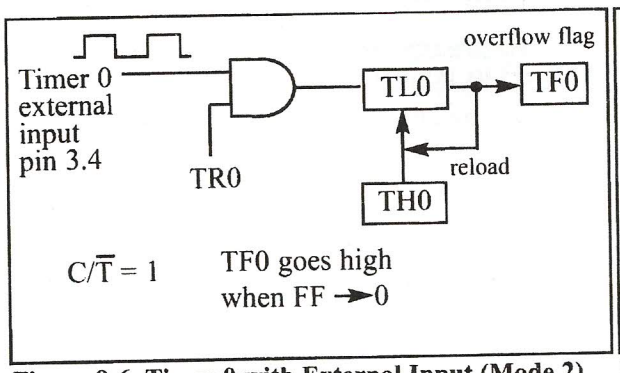


Figure 9-6. Timer 0 with External Input (Mode 2)

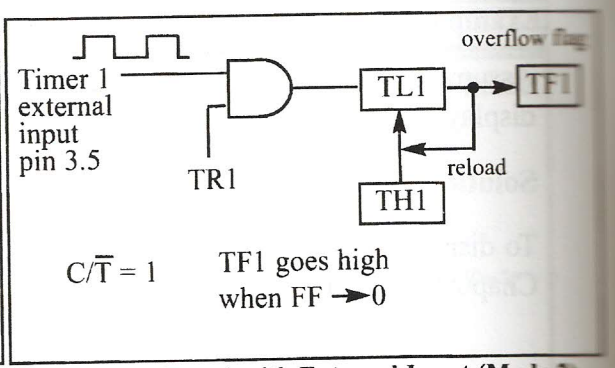


Figure 9-7. Timer 1 with External Input (Mode 2)

As another example of the application of the timer with C/T = 1, we can feed an external square wave of 60 Hz frequency into the timer. The program will generate the second, the minute, and the hour out of this input frequency and display the result on an LCD. This will be a nice digital clock, but not a very accurate one.

Before we finish this chapter, we need to state two important points.

1. You might think that the use of the instruction "`JNB TFx,target`" to monitor the raising of the TFx flag is a waste of the microcontroller's time. You are right. There is a solution to this: the use of interrupts. By using interrupts we can go about doing other things with the microcontroller. When the TF flag is raised it will inform us. This important and powerful feature of the 8051 is discussed in Chapter 11.
2. You might wonder to what register TR0 and TR1 belong. They belong to a register called TCON, which is discussed next.

Table 9-2: Equivalent Instructions for the Timer Control Register (TCON)

For Timer 0			
	SETB TR0	=	SETB TCON.4
	CLR TR0	=	CLR TCON.4
	SETB TF0	=	SETB TCON.5
	CLR TF0	=	CLR TCON.5
For Timer 1			
	SETB TR1	=	SETB TCON.6
	CLR TR1	=	CLR TCON.6
	SETB TF1	=	SETB TCON.7
	CLR TF1	=	CLR TCON.7

TCON: Timer/Counter Control Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TCON register

In the examples so far we have seen the use of the TR0 and TR1 flags to turn on or off the timers. These bits are part of a register called TCON (timer control). This register is an 8-bit register. As shown in Table 9-2, the upper four bits are used to store the TF and TR bits of both Timer 0 and Timer 1. The lower four bits are set aside for controlling the interrupt bits, which will be discussed in Chapter 11. We must notice that the TCON register is a bit-addressable register. Instead of using instructions such as "`SETB TR1`" and "`CLR TR1`", we could use "`SETB TCON.6`" and "`CLR TCON.6`", respectively. Table 9-2 shows replacements of some of the instructions we have seen so far.

The case of GATE = 1 in TMOD

Before we finish this section we need to discuss another case of the GATE bit in the TMOD register. All discussion so far has assumed that GATE = 0. When GATE = 0, the timer is started with instructions "`SETB TR0`" and "`SETB TR1`", for Timers 0 and 1, respectively. What happens if the GATE bit in TMOD is set to 1? As can be seen in Figures 9-8 and 9-9, if GATE = 1, the start and stop of the timer are done externally through pins P3.2 and P3.3 for Timers 0 and 1, respectively. This is in spite of the fact that TRx is turned on by the "`SETB TRx`" instruction. This allows us to start or stop the timer externally at any time via a simple switch. This hardware way of controlling the stop and start of the timer can have many applications. For example, assume that an 8051 system is used in a product to sound an alarm every second using Timer 0, perhaps in addition to many other things. Timer 0 is turned on by the software method of using the "`SETB TR0`" instruction and is beyond the control of the user of that product. However, a switch connected to pin P3.2 can be used to turn on and off the timer, thereby shutting down the alarm.

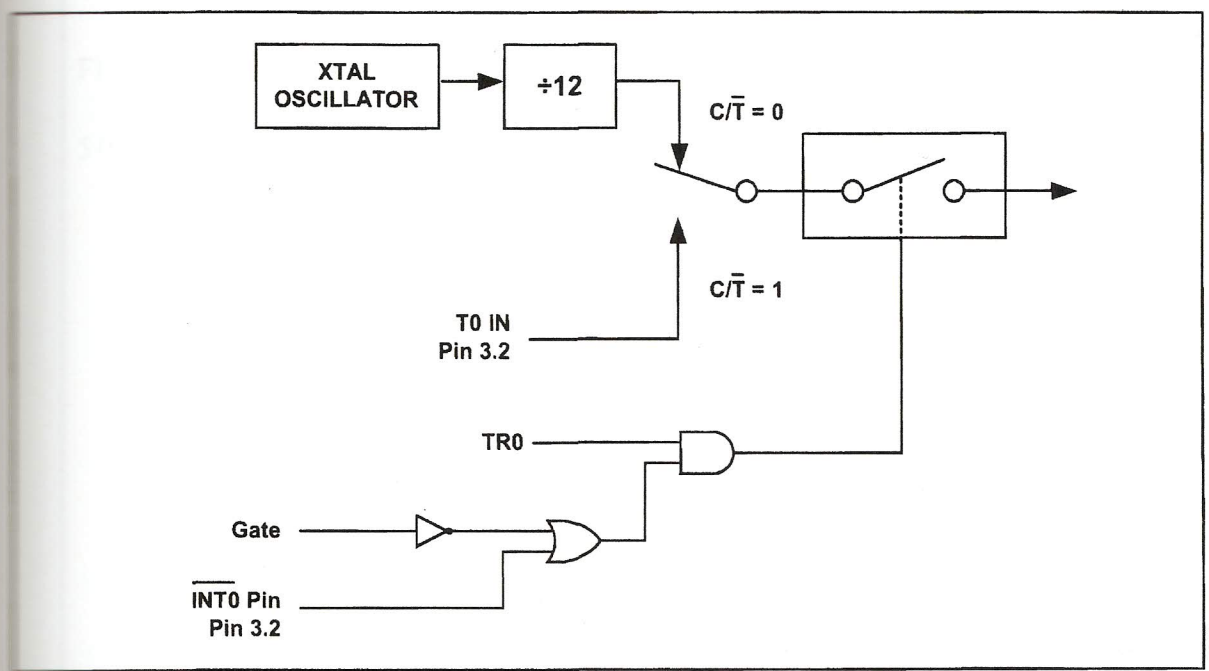


Figure 9-8. Timer/Counter 0

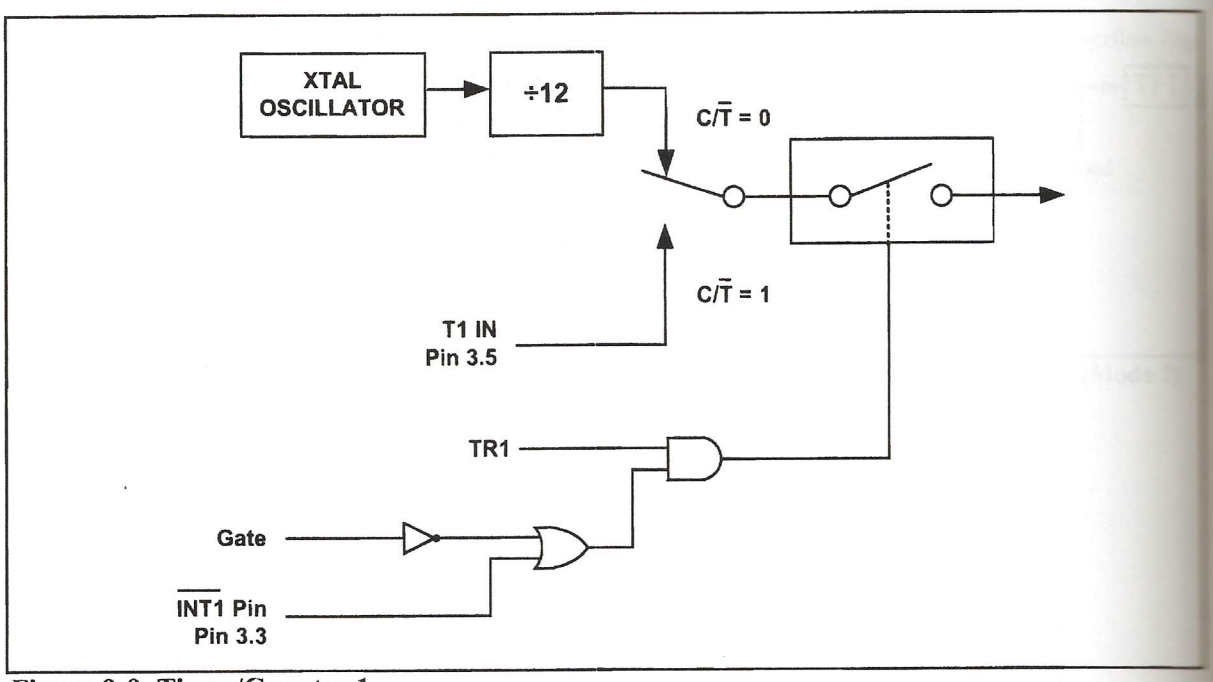


Figure 9-9. Timer/Counter 1

Review Questions

1. Who provides the clock pulses to 8051 timers if C/T = 0?
2. Who provides the clock pulses to 8051 timers if C/T = 1?
3. Does the discussion in Section 9.1 apply to timers if C/T = 1?
4. What must be done to allow P3.4 to be used as an input for T1, and why?
5. What is the equivalent of the following instruction? "`SETB TCON.6`"

SECTION 9.3: PROGRAMMING TIMERS 0 AND 1 IN 8051 C

In Chapter 7 we showed some examples of C programming for the 8051. In this section we study C programming for the 8051 timers. As we saw in the examples in Chapter 7, the general-purpose registers of the 8051, such as R0 - R7, A, and B, are under the control of the C compiler and are not accessed directly by C statements. In the case of SFRs, the entire RAM space of 80 - FFH is accessible directly using 8051 C statements. As an example of accessing the SFRs directly, we saw how to access ports P0 - P3 in Chapter 7. Next, we discuss how to access the 8051 timers directly using C statements.

Accessing timer registers in C

In 8051 C we can access the timer registers TH, TL, and TMOD directly using the reg51.h header file. This is shown in Example 9-20. Example 9-20 also shows how to access the TR and TF bits.

Example 9-20

Write a 8051 C program to toggle all the bits of port P1 continuously with some delay in between. Use Timer 0, 16-bit mode to generate the delay.

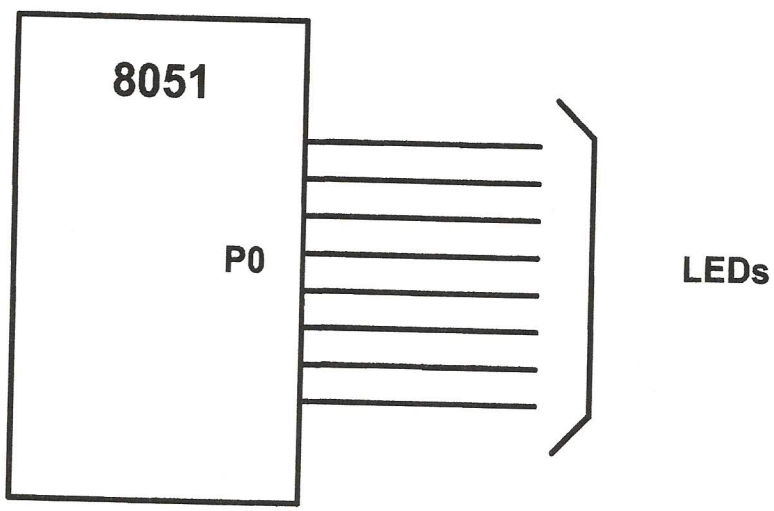
Solution:

```
#include <reg51.h>
void T0Delay(void);
void main(void)
  {
   while(1)                 //repeat forever
    {
      P1=0x55;              //toggle all bits of P1
      T0Delay();            //delay size unknown
      P1=0xAA;              //toggle all bits of P1
      T0Delay();
    }
  }

void T0Delay()
  {
    TMOD=0x01;              //Timer 0, Mode 1
    TL0=0x00;               //load TL0
    TH0=0x35;               //load TH0
    TR0=1;                  //turn on T0
    while(TF0==0);          //wait for TF0 to roll over
    TR0=0;                  //turn off T0
    TF0=0;                  //clear TF0
  }
```

FFFFH – 3500H = CAFFH = 51967 + 1 = 51968

51968 × 1.085 μs = 56.384 ms is the approximate delay.



Calculating delay length using timers

As we mentioned in Chapter 7, the delay length depends on three factors: (a) the crystal frequency, (b) the number of clocks per machine cycle, and (c) the C compiler. The original 8051 used 1/12 of the crystal oscillator frequency as one machine cycle. In other words, each machine cycle is equal to 12 clocks periods of the crystal frequency connected to the X1 - X2 pins. The time it takes for the 8051 to execute an instruction is one or more machine cycles, as shown in Appendix A. To speed up the 8051, many recent versions of the 8051 have reduced the number of clocks per machine cycle from 12 to four, or even one. For example, the AT89C51/52 uses 12, while the DS5000 uses 4 clocks, and the DS89C4x0 uses only one clock per machine cycle. As we mentioned earlier in this chapter, the 8051 timers also use the crystal frequency as the clock source. The frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051, regardless of the 8051 version. In other words, for the AT89C51/52, DS5000, or DS89C4x0 the duration of the time to execute an instruction varies, but they all use 1/12th of the crystal's oscillator frequency for the clock source to the timers. This is done in order to maintain compatibility with the original 8051 since many designers use timers to create time delay. This is an important point and needs to be emphasized. The C compiler is a factor in the delay size since various 8051 C compilers generate different hex code sizes. This explains why the timer delay duration is unknown for Example 9-20 since none of the other factors mentioned is specified.

Delay duration for the AT89C51/52 and DS89C4x0 chips

As we stated before, there is a major difference between the AT89C51 and DS89C4x0 chips in term of the time it takes to execute a single instruction. Although the DS89C4x0 executes instructions 12 times faster than the AT89C51 chip, they both still use Osc/12 clock for their timers. The faster execution time for the instructions will have an impact on your delay length. To verify this very important point, compare parts (a) and (b) of Example 9-21 since they have been tested on these two chips with the same speed and C compiler.

Timers 0 and 1 delay using mode 1 (16-bit non auto-reload)

Examples 9-21 and 9-22 show 8051 C programming of the timers 0 and 1 in mode 1 (16-bit non-auto reload). Examine them to get familiar with the syntax.

Timers 0 and 1 delay using mode 2 (8-bit auto-reload)

Examples 9-23 through 9-25 shows 8051 C programming of timers 0 and 1 in mode 2 (8-bit auto-reload). Study these examples to get familiar with the syntax.

Example 9-21

Write an 8051 C program to toggle only bit P1.5 continuously every 50 ms. Use Timer 0, mode 1 (16-bit) to create the delay. Test the program (a) on the AT89C51 and (b) on the DS89C420.

Solution:

```
#include <reg51.h>
void T0M1Delay(void);
sbit mybit=P1^5;
void main(void)
  {
   while(1)
    {
      mybit=~mybit;   //toggle P1.5
      T0M1Delay();    //Timer 0, mode 1(16-bit)
    }
  }
```

(a) Tested for AT89C51, XTAL=11.0592 MHz, using the Proview32 compiler

```
void T0M1Delay(void)
  {
    TMOD=0x01;          //Timer 0, mode 1(16-bit)
    TL0=0xFD;           //load TL0
    TH0=0x4B;           //load TH0
    TR0=1;              //turn on T0
    while(TF0==0);      //wait for TF0 to roll over
    TR0=0;              //turn off T0
    TF0=0;              //clear TF0
  }
```

(b) Tested for DS89C420, XTAL=11.0592 MHz, using the Proview32 compiler

```
void T0M1Delay(void)
  {
    TMOD=0x01;          //Timer 0, mode 1(16-bit)
    TL0=0xFD;           //load TL0
    TH0=0x4B;           //load TH0
    TR0=1;              //turn on T0
    while(TF0==0);      //wait for TF0 to roll over
    TR0=0;              //turn off T0
    TF0=0;              //clear TF0
  }
```

FFFFH – 4BFDH = B402H = 46082 + 1 = 46083

Timer delay = 46083 × 1.085 μs = 50 ms

Example 9-22

Write an 8051 C program to toggle all bits of P2 continuously every 500 ms. Use Timer 1, mode 1 to create the delay.

Solution:

//tested for DS89C420, XTAL = 11.0592 MHz, using the Proview32 compiler

```
#include <reg51.h>
void T1M1Delay(void);
void main(void)
  {
    unsigned char x;
    P2=0x55;
    while(1)
      {
        P2=~P2;          //toggle all bits of P2
        for(x=0;x<20;x++)
          T1M1Delay();
      }
  }

void T1M1Delay(void)
  {
    TMOD=0x10;           //Timer 1, mode 1(16-bit)
    TL1=0xFE;            //load TL1
    TH1=0xA5;            //load TH1
    TR1=1;               //turn on T1
    while(TF1==0);       //wait for TF1 to roll over
    TR1=0;               //turn off T1
    TF1=0;               //clear TF1
  }
```

A5FEH = 42494 in decimal

65536 – 42494 = 23042

23042 × 1.085 μs = 25 ms and 20 × 25 ms = 500 ms

NOTE THAT 8051 TIMERS USE 1/12 OF XTAL FREQUENCY, REGARDLESS OF MACHINE CYCLE TIME.

Example 9-23

Write an 8051 C program to toggle only pin P1.5 continuously every 250 ms. Use Timer 0, mode 2 (8-bit auto-reload) to create the delay.

Solution:

//tested for DS89C420, XTAL = 11.0592 MHz, using the Proview32 compiler

```
#include <reg51.h>
void T0M2Delay(void);
sbit mybit=P1^5;
void main(void)
  {
    unsigned char x, y;
    while(1)
      {
        mybit=~mybit;              //toggle P1.5
        for(x=0;x<250;x++)         //due to for loop overhead
          for(y=0;y<36;y++)        //we put 36 and not 40
            T0M2Delay();
      }
  }

void T0M2Delay(void)
  {
    TMOD=0x02;                 //Timer 0, mode 2(8-bit auto-reload)
    TH0=-23;                   //load TH0(auto-reload value)
    TR0=1;                     //turn on T0
    while(TF0==0);             //wait for TF0 to roll over
    TR0=0;                     //turn off T0
    TF0=0;                     //clear TF0
  }
```

256 – 23 = 233

23 × 1.085 μs = 25 μs

25 μs × 250 × 40 = 250 ms by calculation.

However, the scope output does not give us this result. This is due to overhead of the for loop in C. To correct this problem, we put 36 instead of 40.

Example 9-24

Write an 8051 C program to create a frequency of 2500 Hz on pin P2.7. Use Timer 1, mode 2 to create the delay.

Solution:

//tested for DS89C420, XTAL = 11.0592 MHz, using the Proview32 compiler

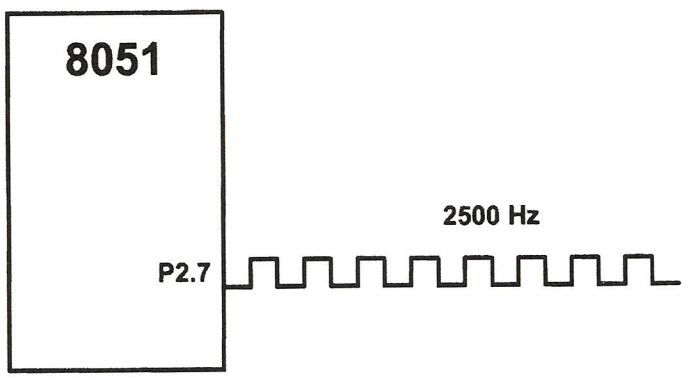
```
#include <reg51.h>
void T1M2Delay(void);
sbit mybit=P2^7;
void main(void)
  {
    unsigned char x;
    while(1)
      {
        mybit=~mybit;        //toggle P2.7
          T1M2Delay();
      }
  }

void T1M2Delay(void)
  {
    TMOD=0x20;               //Timer 1, mode 2(8-bit auto-reload)
    TH1=-184;                //load TH1(auto-reload value)
    TR1=1;                   //turn on T1
    while(TF1==0);           //wait for TF1 to roll over
    TR1=0;                   //turn off T1
    TF1=0;                   //clear TF1
  }
```

1 / 2500 Hz = 400 µs

400 µs / 2 = 200 µs

200 µs / 1.085 µs = 184



Example 9-25

A switch is connected to pin P1.2. Write an 8051 C program to monitor SW and create the following frequencies on pin P1.7:
SW=0: 500 Hz
SW=1: 750 Hz
Use Timer 0, mode 1 for both of them.

Solution:

```
//tested for AT89C51/52, XTAL = 11.0592 MHz, using the Proview32 compiler
#include <reg51.h>
sbit mybit=P1^5;
sbit SW=P1^7;
void T0M1Delay(unsiged char);
void main(void)
  {
    SW=1;                    //make P1.7 an input
    while(1)
      {
        mybit=~mybit;        //toggle P1.5
        if(SW==0)            //check switch
          T0M1Delay(0);
        else
          T0M1Delay(1);
      }
  }
void T0M1Delay(unsigned char c)
  {
    TMOD=0x01;
    if(c==0)
      {
        TL0=0x67;            //FC67
        TH0=0xFC;
      }
    else
      {
        TL0=0x9A;            //FD9A
        TH0=0xFD;
      }
    TR0=1;
    while(TF0==0);
    TR0=0;
    TF0=0;
  }
```

FC67H = 64615
65536 – 64615 = 921
921 × 1.085 μs = 999.285 μs
1 / (999.285 μs × 2) = 500 Hz

C Programming of timers 0 and 1 as counters

In Section 9.2 we showed how to use timers 0 and 1 as event counters. A timer can be used as a counter if we provide pulses from outside the chip instead of using the frequency of the crystal oscillator as the clock source. By feeding pulses to the T0 (P3.4) and T1 (P3.5) pins, we turn Timer 0 and Timer 1 into counter 0 and counter 1, respectively. Study the next few examples to see how timers 0 and 1 are programmed as counters using the C language.

Example 9-26

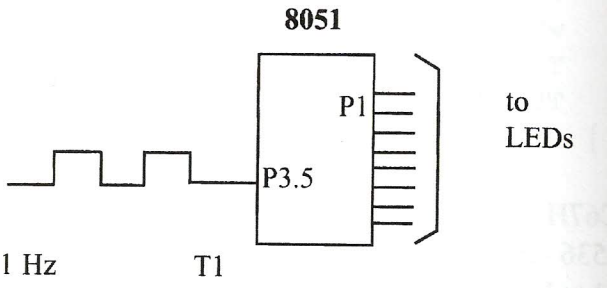
Assume that a 1-Hz external clock is being fed into pin T1 (P3.5). Write a C program for counter 1 in mode 2 (8-bit auto reload) to count up and display the state of the TL1 count on P1. Start the count at 0H.

Solution:

```
#include <reg51.h>
sbit T1 = P3^5;
void main(void)
  {
    T1=1;                    //make T1 an input
    TMOD=0x60;               //
    TH1=0;                   //set count to 0

    while(1)                 //repeat forever
      {
       do
        {
          TR1=1;             //start timer
          P1=TL1;            //place value on pins
        }
       while(TF1==0);        //wait here
       TR1=0;                //stop timer
       TF1=0;                //clear flag
      }
  }
```

P1 is connected to 8 LEDs.
T1 (P3.5) is connected to a 1-Hz external clock.



Example 9-27

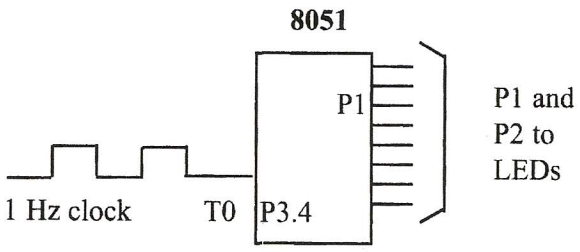
Assume that a 1-Hz external clock is being fed into pin T0 (P3.4). Write a C program for counter 0 in mode 1 (16-bit) to count the pulses and display the TH0 and TL0 registers on P2 and P1, respectively.

Solution:

```
#include <reg51.h>

void main(void)
  {
    T0=1;                    //make T0 an input
    TMOD=0x05;               //
    TL0=0;                   //set count to 0
    TH0=0;                   //set count to 0

    while(1)                 //repeat forever
      {
       do
        {
          TR0=1;             //start timer
          P1=TL0;            //place value on pins
          P2=TH0;            //
        }
       while(TF0==0);        //wait here
       TR0=0;                //stop timer
       TF0=0;
      }
  }
```



Example 9-28

Assume that a 2-Hz external clock is being fed into pin T1 (P3.5). Write a C program for counter 0 in mode 2 (8-bit auto reload) to display the count in ASCII. The 8-bit binary count must be converted to ASCII. Display the ASCII digits (in binary) on P0, P1, and P2 where P0 has the least significant digit. Set the initial value of TH0 to 0.

Solution:

To display the TL1 count we must convert 8-bit binary data to ASCII. See Chapter 7 for data conversion. The ASCII values will be shown in binary. For example, '9' will show as 00111001 on ports.

```
#include <reg51.h>
void BinToASCII(unsigned char);
void main()
  {
    unsigned char value;
    T1=1;
    TMOD=0x06;
    TH0=0;

    while(1)
      {
       do
        {
          TR0=1;
          value=TL0;
          BinToASCII(value);
        }
       while(TF0==0);
       TR0=0;
       TF0=0;
      }
  }

void BinToASCII(unsigned char value)        //see Chapter 7
  {
    unsigned char x,d1,d2,d3;
    x = value / 10;
    d1 = value % 10
    d2 = x % 10;
    d3 = x / 10
    P0 = 30 | d1;
    P1 = 30 | d2;
    P2 = 30 | d3
  }
```

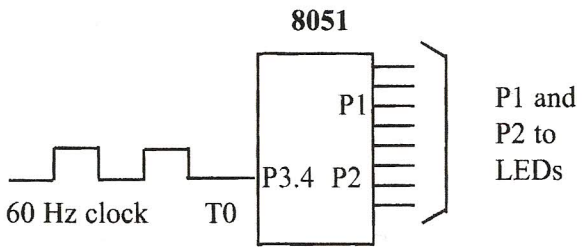
Example 9-29

Assume that a 60-Hz external clock is being fed into pin T0 (P3.4). Write a C program for counter 0 in mode 2 (8-bit auto-reload) to display the seconds and minutes on P1 and P2, respectively.

Solution:

```
#include <reg51.h>
void ToTime(unsigned char);
void main()
  {
    unsigned char val;
    T0=1;
    TMOD=0x06;                  //T0, mode 2, counter
    TH0=-60;                    //sec = 60 pulses
    while(1)
      {
       do
        {
          TR0=1;
          sec=TL0;
          ToTime(val);
        }
       while(TF0==0);
       TR0=0;
       TF0=0;
      }
  }

void ToTime(unsigned char val)
  {
    unsigned char sec, min;
    min = value / 60;
    sec = value % 60;
    P1 = sec;
    P2 = min;
  }
```



By using 60 Hz, we can generate seconds, minutes, hours.

For Examples of Timer 2, see the www.MicroDigitalEd.com Web site.

Review Questions

1. Who provides the clock pulses to 8051 timers if C/T = 0?
2. Indicate the selection made in the statement "`TMOD = 0x20`".
3. In mode 1, the counter rolls over when it goes from ____ to ____.
4. In mode 2, the counter rolls over when it goes from ____ to ____.
5. In the statement "`TH1 = -200`", find the hex value for the TH register.
6. TF0 and TF1 are part of register ____.
7. In Question 6, is the register bit-addressable?
8. Show how to monitor the TF1 flag for high in 8051 C.

SUMMARY

The 8051 has two timers/counters. When used as timers they can generate time delays. When used as counters they can serve as event counters. This chapter showed how to program the timers/counters for various modes.

The two timers are accessed as two 8-bit registers: TL0 and TH0 for Timer 0, and TL1 and TH1 for Timer 1. Both timers use the TMOD register to set timer operation modes. The lower 4 bits of TMOD are used for Timer 0 and the upper 4 bits are used for Timer 1.

There are different modes that can be used for each timer. Mode 0 sets the timer as a 13-bit timer, mode 1 sets it as a 16-bit timer, and mode 2 sets it as an 8-bit timer.

When the timer/counter is used as a timer, the 8051's crystal is used as the source of the frequency; when it is used as a counter, however, it is a pulse outside the 8051 that increments the TH, TL registers.

PROBLEMS

SECTION 9.1: PROGRAMMING 8051 TIMERS

1. How many timers do we have in the 8051?
2. The timers of the 8051 are ____-bit and are designated as _____ and _____.
3. The registers of Timer 0 are accessed as _____ and _____.

4. The registers of Timer 1 are accessed as _____ and _____.
5. In Questions 3 and 4, are the registers bit-addressable?
6. The TMOD register is a(n) ___-bit register.
7. What is the job of the TMOD register?
8. True or false. TMOD is a bit-addressable register.
9. Find the TMOD value for both Timer 0 and Timer 1, mode 2, software start / stop (gate = 0), with the clock coming from the 8051's crystal.
10. Find the frequency and period used by the timer if the crystal attached to the 8051 has the following values.
    (a) XTAL = 11.0592 MHz (b) XTAL = 20 MHz
    (c) XTAL = 24 MHz (d) XTAL = 30 MHz
11. Indicate the size of the timer for each of the following modes.
    (a) mode 0 (b) mode 1 (c) mode 2
12. Indicate the rollover value (in hex and decimal) of the timer for each of the following modes.
    (a) mode 0 (b) mode 1 (c) mode 2
13. Indicate when the TF1 flag is raised for each of the following modes.
    (a) mode 0 (b) mode 1 (c) mode 2
14. True or false. Both Timer 0 and Timer 1 have their own TF.
15. True or false. Both Timer 0 and Timer 1 have their own timer start (TR).
16. Assuming XTAL = 11.0592 MHz, indicate when the TF0 flag is raised for the following program.

```
MOV  TMOD,#01
MOV  TL0,#12H
MOV  TH0,#1CH
SETB TR0
```

17. Assuming that XTAL = 16 MHz, indicate when the TF0 flag is raised for the following program.

```
MOV  TMOD,#01
MOV  TL0,#12H
MOV  TH0,#1CH
SETB TR0
```

18. Assuming that XTAL = 11.0592 MHz, indicate when the TF0 flag is raised for the following program.

```
MOV  TMOD,#01
MOV  TL0,#10H
MOV  TH0,#0F2H
SETB TR0
```

19. Assuming that XTAL = 20 MHz, indicate when the TF0 flag is raised for the following program.

```
MOV  TMOD,#01
MOV  TL0,#12H
MOV  TH0,#1CH
SETB TR0
```

20. Assume that XTAL = 11.0592 MHz. Find the TH1,TL1 value to generate a time delay of 2 ms. Timer 1 is programmed in mode 1.

21. Assume that XTAL = 16 MHz. Find the TH1,TL1 value to generate a time delay of 5 ms. Timer 1 is programmed in mode 1.
22. Assuming that XTAL = 11.0592 MHz, program Timer 0 to generate a time delay of 2.5 ms.
23. Assuming that XTAL = 11.0592 MHz, program Timer 1 to generate a time delay of 0.2 ms.
24. Assuming that XTAL = 20 MHz, program Timer 1 to generate a time delay of 100 ms.
25. Assuming that XTAL = 11.0592 MHz, and we are generating a square wave on pin P1.2, find the lowest square wave frequency that we can generate using mode 1.
26. Assuming that XTAL = 11.0592 MHz, and we are generating a square wave on pin P1.2, find the highest square wave frequency that we can generate using mode 1.
27. Assuming that XTAL = 16 MHz, and we are generating a square wave on pin P1.2, find the lowest square wave frequency that we can generate using mode 1.
28. Assuming that XTAL = 16 MHz, and we are generating a square wave on pin P1.2, find the highest square wave frequency that we can generate using mode 1.
29. In mode 2 assuming that TH1 = F1H, indicate which states timer 2 goes through until TF1 is raised. How many states is that?
30. Program Timer 1 to generate a square wave of 1 kHz. Assume that XTAL = 11.0592 MHz.
31. Program Timer 0 to generate a square wave of 3 kHz. Assume that XTAL = 11.0592 MHz.
32. Program Timer 0 to generate a square wave of 0.5 kHz. Assume that XTAL = 20 MHz.
33. Program Timer 1 to generate a square wave of 10 kHz. Assume that XTAL = 20 MHz.
34. Assuming that XTAL = 11.0592 MHz, show a program to generate a 1-second time delay. Use any timer you want.
35. Assuming that XTAL = 16 MHz, show a program to generate a 0.25-second time delay. Use any timer you want.
36. Assuming that XTAL = 11.0592 MHz and that we are generating a square wave on pin P1.3, find the lowest square wave frequency that we can generate using mode 2.
37. Assuming that XTAL = 11.0592 MHz and that we are generating a square wave on pin P1.3, find the highest square wave frequency that we can generate using mode 2.
38. Assuming that XTAL = 16 MHz and that we are generating a square wave on pin P1.3, find the lowest square wave frequency that we can generate using mode 2.
39. Assuming that XTAL = 16 MHz and that we are generating a square wave on pin P1.3, find the highest square wave frequency that we can generate using mode 2.
40. Find the value (in hex) loaded into TH in each of the following.

(a) MOV TH0,#-12 (b) MOV TH0,#-22
(c) MOV TH0,#-34 (d) MOV TH0,#-92
(e) MOV TH1,#-120 (f) MOV TH1,#-104
(g) MOV TH1,#-222 (h) MOV TH1,#-67

41. In Problem 40, indicate by what number the machine cycle frequency of 921.6 kHz (XTAL = 11.0592 MHz) is divided.
42. In Problem 41, find the time delay for each case from the time the timer starts to the time the TF flag is raised.

SECTION 9.2: COUNTER PROGRAMMING

43. To use the timer as an event counter we must set the C/T bit in the TMOD register to _____ (low, high).
44. Can we use both of the timers as event counters?
45. For counter 0, which pin is used to input clocks?
46. For counter 1, which pin is used to input clocks?
47. Program Timer 1 to be an event counter. Use mode 1 and display the binary count on P1 and P2 continuously. Set the initial count to 20,000.
48. Program Timer 0 to be an event counter. Use mode 2 and display the binary count on P2 continuously. Set the initial count to 20.
49. Program Timer 1 to be an event counter. Use mode 2 and display the decimal count on P2, P1, and P0 continuously. Set the initial count to 99.
50. The TCON register is a(n) _____-bit register.
51. True or false. The TCON register is not a bit-addressable register.
52. Give another instruction to perform the action of "SETB TR0".

SECTION 9.3: PROGRAMMING TIMERS 0 AND 1 IN 8051 C

53. Program Timer 0 in C to generate a square wave of 3 kHz. Assume that XTAL = 11.0592 MHz.
54. Program Timer 1 in C to generate a square wave of 3 kHz. Assume that XTAL = 11.0592 MHz.
55. Program Timer 0 in C to generate a square wave of 0.5 kHz. Assume that XTAL = 11.0592 MHz.
56. Program Timer 1 in C to generate a square wave of 0.5 kHz. Assume that XTAL = 11.0592 MHz.
57. Program Timer 1 in C to be an event counter. Use mode 1 and display the binary count on P1 and P2 continuously. Set the initial count to 20,000.
58. Program Timer 0 in C to be an event counter. Use mode 2 and display the binary count on P2 continuously. Set the initial count to 20.

ANSWERS TO REVIEW QUESTIONS

SECTION 9.1: PROGRAMMING 8051 TIMERS

1. Two
2. 2, 8
3. 8
4. False
5. 0010 0000 indicates Timer 1, mode 2, software start and stop, and using XTAL for frequency.
6. FFFFH to 0000
7. FFH to 00
8. –200 is 38H; therefore, TH1 = 38H
9. 2 ms/1.085 ms = 1843 = 0733H where TH = 07H and TL = 33H
10. 100 ms/1.085 ms = 92 or 5CH; therefore, TH = 5CH

SECTION 9.2: COUNTER PROGRAMMING

1. The crystal attached to the 8051
2. The clock source for the timers comes from pins T0 and T1.
3. Yes
4. We must use the instruction "SETB P3.4" to configure the T1 pin as input, which allows the clocks to come from an external source. This is because all ports are configured as output upon reset.
5. SETB TR1

SECTION 9.3: PROGRAMMING TIMERS 0 AND 1 IN 8051 C

1. The crystal attached to the 8051
2. Timer 2, mode 2, 8-bit auto reload
3. FFFFH to 0
4. FFH to 0
5. 38H
6. TMOD
7. Yes
8. while (TF1==0);