Chapter 10
Timers and Analog Interface

Chien-Chung Ho (何建忠)

PIC18F Timers

- The PIC18F4321 microcontroller includes four timers, namely, Timer0 (8-bit or 16-bit), Timer1 (16-bit), Timer2 (8-bit), and Timer3 (16-bit). These timers can be used to generate time delays using on-chip hardware. Because these timers are on-chip hardware devices, they accurately keep time in the background while other functions are performed by the PIC18F4321.

PIC18F Timers

- The basic hardware inside each of these timers is a register that can be incremented or decremented at the rising or falling edge of a clock. The register can be loaded with a count for a specific time delay. *Time delay* is computed by subtracting the initial starting count from the final count in the register, and then multiplying the subtraction result by the clock frequency.

PIC18F Timers

- These timers can also be used as *event counters*. Note that an event counter is basically a register with the clock replaced by an event such as a switch. The counter is incremented or decremented whenever the switch is activated. Thus, the number of times the switch is activated (occurrence of the event) can be determined.

PIC18F Timers

- Timer0 can operate as an 8-bit timer or a 16-bit timer. Timer0 utilizes the internal clock when used as a timer, and an external clock when used as a counter.
- Timer1 can only be used as a 16-bit timer or a 16-bit counter; it cannot be used as an 8-bit timer or 8-bit counter.

PIC18F Timers

- Like Timer0, Timer1 uses the internal clock when used as a timer, and the external clock when used as a counter.
- Timer2 can be used as an 8-bit timer using the internal clock. It cannot be used as a counter.
- Finally, Timer3 can operate as a 16-bit timer or a 16-bit counter. It cannot be used as an 8-bit timer or an 8-bit counter.

PIC18F Timers

- Timer1, Timer2, and Timer3 are used for the CCP (Capture/Compare/Pulse Width Modulation) operation.
- For example, the PIC18F4321 utilizes Timer1 or Timer3 for the capture and compare modes, and Timer2 for the PWM mode.
- Note that Timer0 is not used by the CCP module.

PIC18F Timers

- In summary, the timer is basically a binary counter (Timer register) which can be programmed to count clock pulses . Once the timer register reaches the maximum value (0xFF for 8-bit, 0xFFFF for 16-bit), it will roll back to 0 while setting up an overflow flag to one at the next cycle, and also generate an interrupt if enabled.

PIC18F Timers

- **Polling Timer flags**. Timer0 counts up to a maximum value of 0xFF when it operates as an 8-bit timer and then overflows to 0. Timer0/1/3 counts up to a maximum value of 0xFFFF while operating as a 16-bit timer, and then overflows to 0.
- Timer2 contains an 8-bit timer register (TMR2) and an 8-bit period register (PR2). Timer2 overflows to 0 for TMR2-to-PR2 match. The overflow is indicated automatically by the PIC18F4321 by setting a flag bit (TMRxIF bit where x = 0, 1, 2, 3) associated with each of these timers. This flag bit can be polled in software to provide time-related functions. All four timers include the flag bit.

PIC18F Timers

TABLE 10.1 PIC18F4321 Timers with basic features

Timer	Size	Register (Enable and Flag bits)	Counter register
Timer0	8-bit or 16-bit	TMR0IE, TMR0IF in INTCON	TMR0L for 8-bit TMR0H:TMR0L for 16-bit
Timer1	16-bit	TMR1IE in PIE1 TMR1IF in PIR1	TMR1H:TMR1L
Timer2	8-bit	TMR2IE in PIE1 TMR2IF in PIR1	TMR2
Timer3	16-bit	TMR3IE in PIE2 TMR3IF in PIR2	TMR3H:TMR3L

PIC18F Timers

- **Timer interrupts**. The problem with polling the timer flag bit is that the PIC18F4321 has to wait in a loop checking for the TMRxIF bit to be HIGH. In this case, the PIC18F4321 will not be able to do any other task. Hence, timer interrupts can be used.

PIC18F Timers

- Since timers are considered as peripheral devices, a single peripheral interrupt (PEIE) bit is provided in the INTCON register for all four timers.
- Each timer (TMRx where x = 0,1,2,3) is also provided with an interrupt enable bit (TMRxIE where x = 0,1,2,3) and an interrupt flag bit (TMRxIF where x = 0,1,2,3).

PIC18F Timers

- In order for obtaining timer-driven interrupts with any of the four timers, initializations of certain registers are necessary. Consider Figure 10.1 for Timer0. The following bits associated with Timer0 can be initialized via programming as follows: TMR0IF = 0, TMR0IE = 1, PEIE = 1, GIE = 1. This can be accomplished as follows

```
BCF     INTCON, TMR0IF          INTCONbits.TMR0IF = 0;
BSF     INTCON, TMR0IE          INTCONbits.TMR0IE = 1;
BSF     INTCON, PEIE            INTCONbits.PEIE = 1;
BSF     INTCON, GIE             INTCONbits.GIE = 1;
```

PIC18F Timers

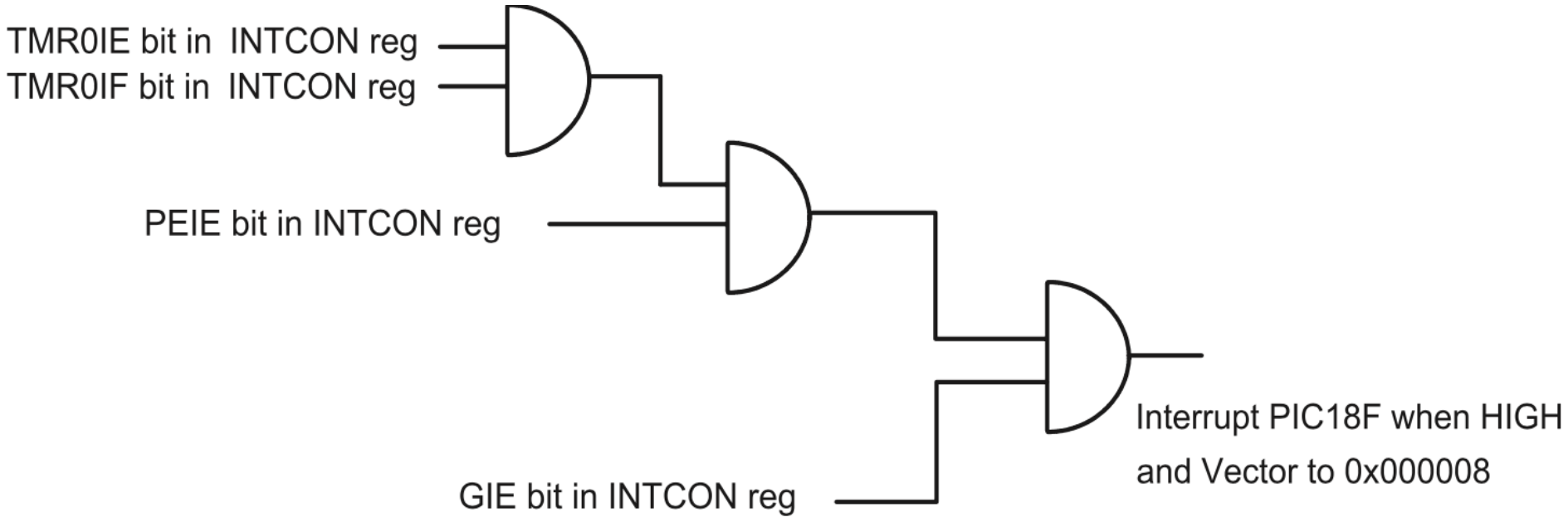


FIGURE 10.1 Timer0 Interrupt

PIC18F Timers

- As soon as Timer0 interrupts the PIC18F4321, the microcontroller completes execution of the current instruction and loads PC with address 0x000008 (assuming default mode). A service routine can be written at this address to perform timer-related functions. Note that interrupt-driven timers using Timer1 through Timer3 work in the same manner.

PIC18F Timers

- **Prescaler (**預除器**)** and Postscaler.
  - The prescaler option divides the PIC18F's clock by an integer, and reduces the clock period. This reduction in clock period provides the timer with higher time delays.
  - For example, consider internal clock of 4 MHz. Since the PIC18F timer works with a divide-by-4 crystal, the timer frequency = (4 MHz/4) = 1 MHz. Using a prescaler value of 1:4, Timer clock frequency = (1MHz/4) = 0.25 MHz = 250 kHz. Hence, a lower clock frequency will increase the clock period providing a higher time delay.
  - Note that prescaler means that the timer clock frequency is divided before the frequency is used by the timer. In the PIC18F4321, **Timer0, Timer1, and Timer3 are provided with prescalers.** They do not have postscaler option.

PIC18F Timers

- Prescaler and **Postscaler (**後除器**)**.
  - The postscaler also divides the clock by an integer after the timer is used. This clock is then used to slow down setting of the timer interrupt flag bit.
  - In the PIC18F4321, the Timer2 is provided with both prescaler and postscaler options. As will be shown later, Timer2 prescaler option can be used to obtain more time delays while the Timer2 postscaler option can be programmed to delay TMR2IF (Timer Interrupt Flag) generation.

Timer0

- The Timer0 can operate as a timer or as a counter in 8-bit or 16-bit mode. The Timer0 uses the internal clock when used as a timer, and external clock (T0CK1) when used as a counter.

Timer0

- **Timer0 as a timer**. The Timer0 can be used as a timer by setting the TMR0ON (bit 7 of T0CON register of Figure 10.2) to 1.
- After the Timer0 is started, **it counts up by incrementing the contents of the register** (TMR0L for 8-bit timer mode or TMR0H:TMR0L for 16-bit timer mode) **by 1 at each instruction cycle**.
  - The TMR0L counts up until the TMR0L reaches **0xFF** in the 8-bit mode. The TMR0IF (interrupt on overflow) flag bit in the INTCON register is set to 1 when the TMR0L rolls over from 0xFF to 0x00. In the 8-bit mode, only the TMR0L register is used; the TMR0H register is not used, and contains a value of 0.
  - In the 16-bit, after the Timer0 is started, the TMR0H:TMR0L register pair counts up until the TMR0H:TMR0L reaches **0xFFFF** in the 16-bit timer mode.
  - The timer **can be stopped** in either 8-bit or 16-bit mode **by clearing the TMR0ON to 0.**

Timer0

7	6	5	4	3	2	1	0	
TMR0ON	T08BIT	T0CS	T0SE	PSA	TOPS2	TOPS1	TOPS0	T0CON

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = External clock connected to RA4/T0CKI pin (pin 6; Timer0 external clock input)
0 = Internal clock from crystal oscillator (divide by 4; crystal frequency can vary from 4-MHz to 25 MHz)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

FIGURE 10. 2 T0CON (TIMER0 Control) Register

Timer0

- **Timer0 as a counter**. The Timer0 can be configured as a counter by setting the T0CS bit (bit 5 in the T0CON register of Figure 10.2) to 1.
- This will enable the PIC18F4321 to use an external clock connected to the T0CK1 pin. The T0SE bit (bit 4 of the T0CON register) can then be cleared to 0 to increment the Timer0 register to increment on the rising edge of the clock or set to one to increment the Timer0 register on the falling edge of the clock.

Timer0

- **Timer0 block diagrams**. In Figure 10.3 TOSE (bit 4 of T0CON register of Figure 10.2) is Exclusive-ORed with the T0CK1 clock pin of the PIC18F4321. The output of the Exclusive-OR gate is selected by T0CS (bit 5 of T0CON of Figure 10.2). When T0CS is 0, the internal oscillator (Fosc/4) is used (T0CS=0), and the Timer0 operates as a timer; otherwise, the external clock is selected (T0CS=1) and the Timer0 operates as a counter.

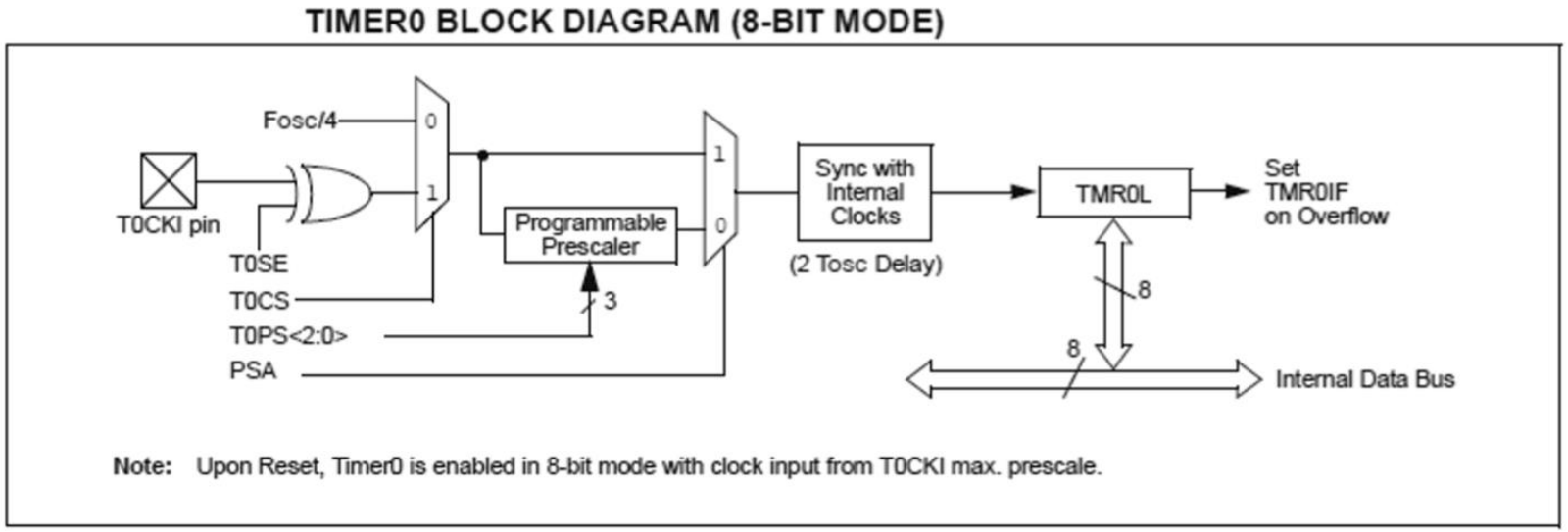


FIGURE 10. 3 TIMER0 Block Diagram (8-bit Mode)

Timer0

- The Timer0 bypasses the prescaler if PSA bit (bit 3 of T0CON register of Figure 10.2) is 1; otherwise, the prescaler is selected (PSA = 0), and is specified by TOPS2:TOPS0 (T0CON of Figure 10.2). The next block provides a two cycle (Tosc) delay. This is because when control data is written to T0CON register, the increment operation is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the timer register called TMR0.

Timer0

- An interrupt on overflow indicated by TMR0IF (bit 2 of INTCON register in Figure 9.7) is set to one if the TMR0L rolls over from 0xFF to 0x00.
- The block diagram for the Timer0 16-bit mode of Figure 10.4 can similarly be explained.

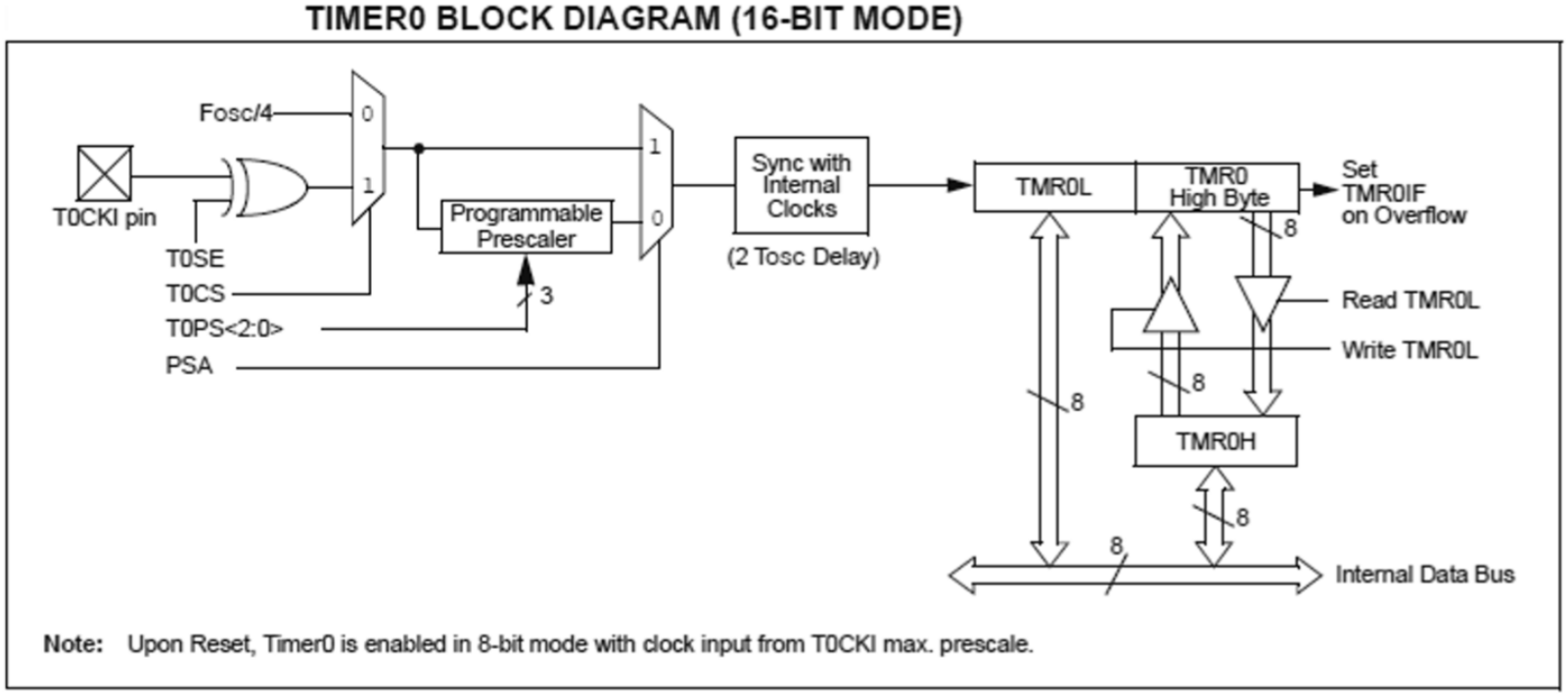


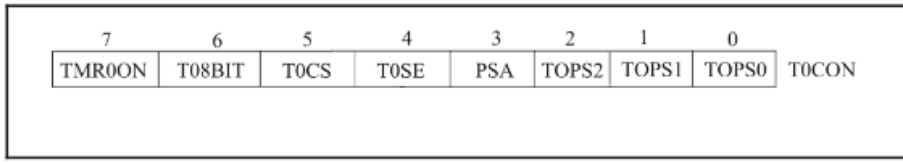
FIGURE 10. 4 TIMER0 Block Diagram (16-bit Mode)

Timer0

- **Timer0 Read/Write in 16-Bit Mode**. Two 8-bit registers (TMR0H and TMR0L) are used to hold the 16-bit value in this mode. The 8-bit high byte TMR0H is latched (buffered). It is not the actual high byte of Timer0 in 16-bit mode and is not directly readable nor writable. Since **TMR0H is not the actual high byte register of Timer0, one should initialize (write to) TMR0H before TMR0L to avoid any errors.** Note that the upper 8-bit value of the timer is stored in the latched register, and loaded into actual TMR0H when TMR0L is loaded.
- Similarly, a write to the high byte of Timer0 must also take place through the TMR0H Buffer register. The high byte is updated with the contents of TMR0H when a write occurs to TMR0L. This allows all 16 bits of Timer0 to be updated at once.

Timer0

- **Prescaler** The value of the Prescaler is set by the PSA (bit 3 of T0CON), and T0PS2:T0PS0 bits (bits 0 through 3 of T0CON) which determine the prescaler assignment and prescale ratio. Clearing the PSA bit assigns the prescaler to the Timer0. When it is assigned, prescale values from 1:2 through 1:256 in power-of-2 increments are selectable.



bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = External clock connected to RA4/T0CKI pin (pin 6; Timer0 external clock i
0 = Internal clock from crystal oscillator (divide by 4; crystal frequency can vary
4-MHz to 25 MHz)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler outpu

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

FIGURE 10. 2 T0CON (TIMER0 Control) Register

Timer0

- When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., in assembly, CLRF TMR0, MOVWF TMR0, BSF TMR0) will clear the prescaler count. The prescaler assignment is fully under software control and can be changed “on-the-fly” during program execution.

Timer0

- **Timer0 Interrupt and Timer0 Flag bits**. The INTCON register shows the TMR0IE and TMR0IF bits. The TMR0 interrupt can be generated internally (if enabled by setting TMR0IE to 1) using "INTCONbits.TMR0IE = 1;" in C. This is used for interrupt driven Timer0.

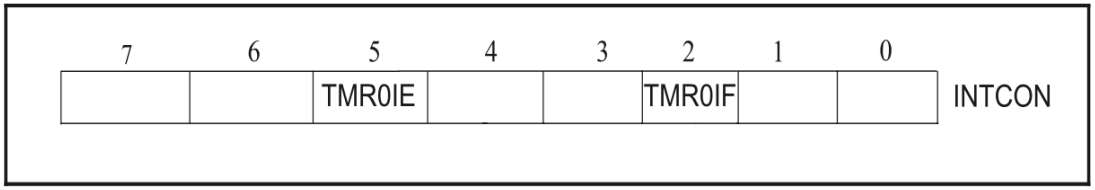
(a) **INTCON Register**

7	6	5	4	3	2	1	0	
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0I F	INT0I F	RBIF	INTCON

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0: 1 = Enables all unmasked interrupts 0 = Disables all interrupts
When IPEN = 1: 1 = Enables all high priority interrupts 0 = Disables all interrupts
bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0: 1 = Enables all unmasked peripheral interrupts 0 = Disables all peripheral interrupts
When IPEN = 1: 1 = Enables all low priority peripheral interrupts 0 = Disables all low priority peripheral interrupts
bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit,
1 = Enables the TMR0 overflow interrupt 0 = Disables the TMR0 overflow interrupt
bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt, 0 = Disables the INT0 external interrupt
bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt, 0 = Disables the RB port change interrupt
bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software), 0 = TMR0 register did not overflow
bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software), 0 = The INT0 external interrupt did not occur
bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Timer0

- When the TMR0 register overflows from 0xFF to 0x00 in 8-bit mode, or from 0xFFFF to 0x0000 in 16-bit mode, this overflow sets the TMR0IF flag bit in the INTCON register to 1 (Figure 10.5).



Bit 5 **TMR0IE** (TMR0 Overflow Interrupt Enable bit): 1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt.

Bit 2 **TMR0IF** (TMR0 Overflow Interrupt Flag bit): 1 = TMR0 register has overflowed (must be cleared in software),
0 = TMR0 register did not overflow

FIGURE 10. 5 INTCON Register with the TMR0IE and TMR0IF bits

Timer0

- For interrupt driven Timer0, the following C-code will clear TMR0IF to 0, set PEIE to 1, TMR0IE to 1, GIE to 1, and then start Timer0:

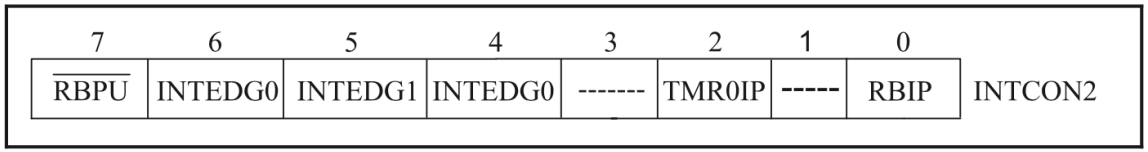
```
INTCONbits.TMR0IF = 0;
INTCONbits.PEIE = 1;          // enable peripheral interrupt
INTCONbits.TMR0IE = 1;        // Set TMR0IE to 1
INTCONbits.GIE = 1;           // Enable global interrupt
T0CONbits.TMR0ON = 1;         // Start TMR0
```

Timer0

- Once TMR0 overflows, the PIC18F4321 sets the TMR0IF bit to one in the INTCON register. The PIC18F4321 is then interrupted. The microcontroller completes execution of the current instruction, saves the program counter (PC) internally, and automatically loads the PC with 0x00008 where a service routine can be written.

Timer0

- Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit, TMR0IP (bit 2 of INTCON2, Figure 9.9). The PIC18F4321 can use the Timer0 interrupt as a high priority interrupt by setting TMR0IP to 1.



bit 7 **RBPU:** PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values
bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit, 1 = Interrupt on rising edge, 0 = Interrupt on falling edge
bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit, 1 = Interrupt on rising edge, 0 = Interrupt on falling edge
bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit, 1 = Interrupt on rising edge, 0 = Interrupt on falling edge
bit 3 **Unimplemented:** Read as '0'
bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit, 1 = High priority, 0 = Low priority
bit 1 **Unimplemented:** Read as '0'
bit 0 **RBIP:** RB Port Change Interrupt Priority bit, 1 = High priority, 0 = Low priority

FIGURE 9. 9 INTCON2 Register

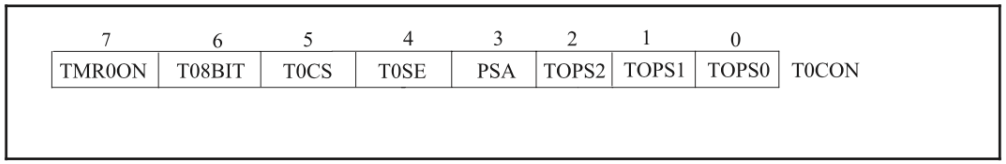
Timer0

- The PIC18F4321 can use the timer interrupt as a low priority interrupt (vector address 0x000018) by clearing TMR0IP to 0 using the Code: INTCON2bits.TMR0IP = 0.
- The TMR0IF bit must be cleared via programming in the Interrupt Service Routine. Note that the PIC18F instruction “BCF INTCON, TMR0IF” will clear the timer interrupt flag bit.

Timer0

Example 10.1 Assuming 4 MHz crystal oscillator, calculate the time delay for the following PIC18F instruction sequence:

```
MOVLW   0xD4
MOVWF   T0CON     ; Initialize T0CON with 0xD4
MOVLW   0x80      ; Load 8-bit timer with count 0x80
MOVWF   TMR0L
```



bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = External clock connected to RA4/T0CKI pin (pin 6; Timer0 external clock input)
0 = Internal clock from crystal oscillator (divide by 4; crystal frequency can vary from 4-MHz to 25 MHz)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

FIGURE 10. 2 T0CON (TIMER0 Control) Register

Timer0

- Solution. The above PIC18F instruction sequence loads 0xD4 into the T0CON register. Note that 0xD4 = 11010100. Hence, from Figure 10.2, the T0CON register can be drawn with the binary data as shown in Figure 10.6. TMR0ON = 1 meaning TIMER0 is ON, T08BIT = 1 meaning 8-bit timer, T0CS = 0 meaning internal instruction clock, PSA = 0 meaning prescaler enabled, TOPS2 TOPS1 TOPS0 = 100 meaning 1:32 prescale value.

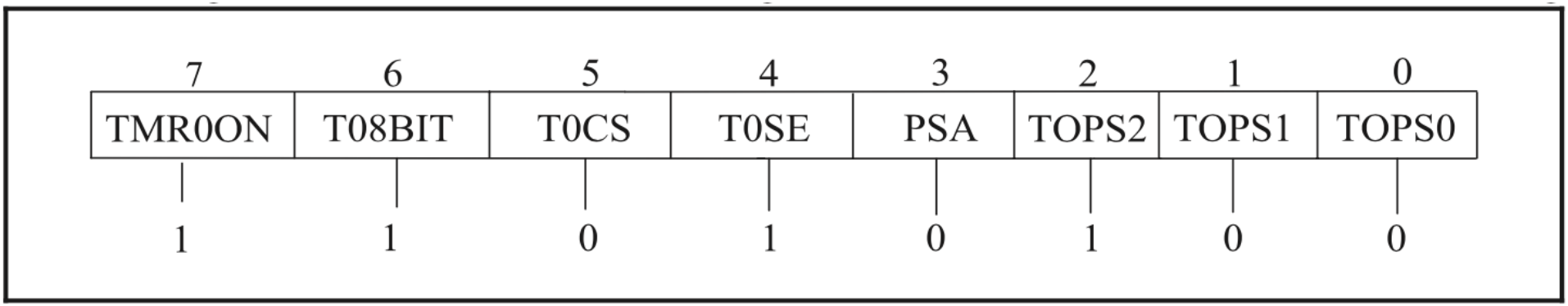


FIGURE 10.6 T0CON register with binary data 11010100

Timer0

- Clock Period = 1/(4 MHz) = 0.25, Instruction cycle clock period = 4 x 0.25 μsec = 1 μsec. Since the prescaler multiplies the Instruction cycle clock period by the prescaler value, Time Delay = (Instruction cycle clock perod) x (Prescaler value) x (Counter value) = (1 μsec) x (32) x (128) = 4096 μsec = 4.096 ms.
- Note that, in the above, Counter value = 0x80 = 128 in decimal. This value determines the desired time delay. Also, the last two instructions MOVLW and MOVWF account for the two instruction cycles, during which the increment operation is inhibited before writing to the TMR0L register.

Timer0

- Example 10.2 Using Timer0 in 16-bit mode, write a PIC18F assembly language program to obtain a time delay of 1 ms. Assume an 8-MHz crystal, and a prescale value of 1:128. As mentioned in the last section, TMR0H must be initialized prior to TMR0L to avoid any errors since the contents of TMR0 are changed after Timer0 is started.

Timer0

- Solution. Since the timer works with a divide by 4 crystal, the timer frequency = (8MHz)/4 = 2 MHz.
- Instruction cycle clock period = (1/2 MHz) = 0.5 μsec.
- The bits in register T0CON are as follows:
  - TMR0ON(bit 7) = 0, T08BIT (bit 6) = 0, T0CS (bit 5) = 0, PSA (bit 3) = 0,
  - TOPS2 TOPS1 TOPS0 = 110 for a prescale value of 1: 128. Hence, the T0CON register will be initialized with 0x06.

Timer0

- **Time delay = Instruction cycle x Prescale value x Count**
- Hence, Count = (1 ms) / (0. 5 μsec x 128) = 15.625 which can be approximated to an integer value of 16 (0x0010). The timer counts up from an initialized value to 0xFFFF, and then rolls over (increments) to 0x0000. The number of counts for rollover is (0xFFFF - 0x0010) = 0xFFEF.

Timer0

- Note that an extra cycle is needed for the roll over from 0xFFFF to 0x0000, and the TMR0IF flag is then set to 1. Because of this extra cycle, the total number of counts for roll over = 0xFFEF + 1 = 0xFFF0.
- The following PIC18F assembly language program will provide a time delay of 1 ms:

Timer0

```
        INCLUDE   <P18F4321.INC>
        MOVLW     0x70                ;8MHz internal clock
        MOVWF     OSCCON
        MOVLW     0x06                ; Initialize T0CON
        MOVWF     T0CON
        MOVLW     0xFF                ; Initialize TMR0H first with 0xFF
        MOVWF     TMR0H
        MOVLW     0xF0                ; Initialize TMR0L next
        MOVWF     TMR0L
        BCF       INTCON, TMR0IF      ; Clear Timer0 flag bit
        BSF       T0CON, TMR0ON       ; Start Timer0
BACK    BTFSS     INTCON, TMR0IF      ; Check Timer0 flag bit for 1
        GOTO      BACK                ; Wait in loop
        BCF       T0CON, TMR0ON       ; Stop Timer0
FINISH  BRA       FINISH              ; Halt
        END
```

Timer0

- Example 10.3 Using Timer0 in 16-bit mode, write a C language program to obtain a time delay of 1 ms. Assume an 8-MHz crystal, and a prescale value of 1:128. As mentioned in the last section, one should initialize TMR0H before TMR0L to avoid any errors.

Timer0

Since the timer works with a divide by 4 crystal, the timer frequency = (8MHz)/4 = 2 MHz. Instruction cycle clock period = (1/2 MHz) = 0.5 μsec.

The bits in register T0CON of Figure 10.2 are as follows:
TMR0ON(bit 7) = 0, T08BIT (bit 6) = 0, T0CS (bit 5) = 0, Since internal clock is used, T0SE = x (don't care), assumed 0 here. Hence, T0SE (bit 4) = 0, PSA (bit 3) = 0, TOPS2 TOPS1 TOPS0 = 110 for a prescale value of 1: 128. Hence, the T0CON register will be initialized with 0x06.

Time delay = Instruction cycle x Prescale value x Count
Hence, Count = (1 ms) / (0.5 μsec x 128) = 15.625 which can be approximated to an integer value of 16 (0x0010). The timer counts up from an initialized value to 0xFFFF, and then rolls over (increments) to 0000H. The number of counts for rollover is (0xFFFF - 0x0010) = 0xFFEF.

Timer0

The following C language program will provide a time delay of 1 ms:

```
#include<p18f4321.h>
void main(void)
        {
            OSCCON=0x70;                        // 8MHz
            T0CON=0x06;                         // Initialize T0CON
            TMR0H=0xFF;                         // Initialize TMR0H first with 0xFF
            TMR0L=0xF0;                         // Initialize TMR0L next
            INTCONbits.TMR0IF=0;                // Clear Timer0 flag bit
            T0CONbits.TMR0ON=1;                 // Start Timer0
            while(INTCONbits.TMR0IF==0);// Wait for Timer0 flag bit to be 1
            T0CONbits.TMR0ON=0;                 // Stop Timer0
            While(1);                           // Halt
        }
```

Timer0

Example 10.4 Using Timer0 in 16-bit mode, write a C language program to toggle an LED connected to bit 0 of PORTC after 10 seconds. Assume a 4-MHz crystal, a leading edge clock, and a prescale value of 1:256. Assume a '0' will turn the LED OFF while a '1' will turn it ON.

(a) Use polled I/O

(b) Use interrupt I/O

Timer0

- (a) Since the timer works with a divide by 4 crystal, the timer frequency = (4MHz)/4 = 1 MHz.
- Instruction cycle clock period = (1/1 MHz) = 1 μsec.
- The bits in register T0CON are as follows:
  - TMR0ON(bit 7) = 0, T08BIT (bit 6) = 0, T0CS (bit 5) = 0, T0SE (bit 4) = 0, PSA (bit 3) = 0, and TOPS2 TOPS1 TOPS0 = 111 for a prescale value of 1: 256. Hence, the T0CON register will be initialized with 0x07.

Timer0

- Time delay = Instruction cycle x Prescale value x Count
- Hence, Count = (10 second) / (1 μsec x 256) = 39062.5. For a 16-bit, total counts in decimal will be 65,535. Hence, Timer0 needs to count up from an initialized value of 65535- 39062.5 = 26472.5 which can be approximated to 26472 in decimal or 6768 in hex.

Timer0

- Note that an extra cycle is needed for the rollover from 0xFFFF to 0x0000, and the TMR0IF flag is then set to 1. Because of this extra cycle, the number of counts for the rollover is 0x6769 (0x6768 + 1) cycles which means that the total number of counts for rollover will be 0x6769. Hence, TMR0H = 67, TMR0L = 69.

Timer0

```
//    C-program using polled I/O:
#include <P18F4321.h>
unsigned char count;
void T0Delay();
void main()
{
      OSCCON = 0x60;                  // 4MHz
      TRISC = 0x00;                   // Port C output
      PORTCbits.RC0 = 0;              // turn LED OFF

T0Delay();                            // Use Timer0
      PORTCbits.RC0 = 1;              // turn LED ON
}
void T0Delay()
{
T0CON = 0x07;                         // 16-bit, 1:256 prescale, internal clock
      TMR0H = 0x67;
      TMR0L = 0x69;
      INTCONbits.TMR0IF = 0;          // clear timer flag
      T0CONbits.TMR0ON = 1;           // start Timer0
      while (!INTCONbits.TMR0IF);     // polling, wait until timer finishes counting
      T0CONbits.TMR0ON = 0;           // Stop Timer0
}
```

Timer0

- (b) For interrupt I/O, we will have to loop the complete program. Although all other instructions in the loop will be repeated, execution times of these instructions are negligible compared to 10 seconds, and can be discarded.

Timer0

```
// using C
#include <P18F4321.h>
unsigned char count;
void LEDON();
#pragma code T0ISR=0x08
void T0ISR()
{
      _asm
      GOTO LEDON
      _endasm
}
#pragma code

void main()
{
      OSCCON = 0x60;              // 4MHz
      TRISC = 0x00;               // Port C output
      PORTCbits.RC0 = 0;          // LED off initially
      INTCONbits.PEIE = 1;        // enable peripheral interrupt
      INTCONbits.GIE = 1;         // enable global interrupt
      T0CON = 0x07;               // 16-bit, 1:256 prescale, internal clock
      TMR0H = 0x67;
      TMR0L = 0x69;
      INTCONbits.TMR0IE = 1;      // enable Timer0 interrupt
      INTCONbits.TMR0IF = 0;      // clear Timer0 interrupt
      T0CONbits.TMR0ON = 1;       // start Timer0
      while (1);                  // wait until interrupt occurs
}
```

Timer0

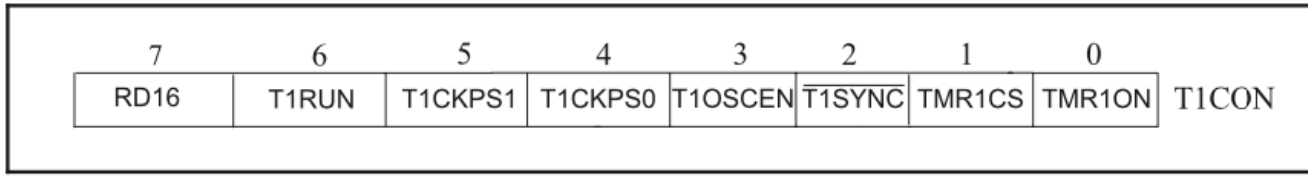
```
#pragma interrupt LEDON
void LEDON()
{
    PORTCbits.RC0 = 1;          // LED on
    INTCONbits.TMR0IF = 0;      // clear Timer0 flag
}
```

Timer1

- The Timer1 can be used as a 16-bit timer or a counter. It consists of two 8-bit registers, namely TMR1H and TMR1L. The TMR1IF flag in the PIR1 register goes to HIGH when the Timer1 overflows from 0xFFFF to 0x0000. An extra cycle is required when the Timer1 rolls over from 0xFFFF to 0x0000.

Timer1

- Timer1 is controlled through the **T1CON Control** register. It also contains the Timer1 Oscillator Enable bit (T1OSCEN). Timer1 can be enabled or disabled by setting or clearing the TMR1ON (bit 0 of T1CON) control bit.



7	6	5	4	3	2	1	0	
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	T1CON

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of TImer1 in one 16-bit operation
0 = Enables register read/write of Timer1 in two 8-bit operations

bit 6 **T1RUN:** Timer1 System Clock Status bit
1 = Device clock is derived from Timer1 oscillator
0 = Device clock is derived from another source

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut off

bit 2 **T1SYNC:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

FIGURE 10.7 T1CON (Timer1 Control) Register

Timer1

- **Timer1 Operation**
- Timer1 can operate as a timer, a synchronous counter, or an asynchronous counter. The operating mode is determined by the clock select bit, TMR1CS.
  - When TMR1CS is cleared to 0, Timer1 operates as a timer using the internal clock, and increments on every internal instruction cycle (Fosc/4).
  - When TMR1CS bit is set to 1, Timer1 increments on every rising edge of the Timer1 external clock input or the Timer1 oscillator, if enabled.
    - Note that the on-chip crystal oscillator circuit can be enabled by setting the Timer1 Oscillator Enable bit, T1OSCEN. The oscillator is a low-power circuit rated for 32 kHz crystals.
  - Finally, the Timer1 is enabled by setting the TMR1ON to 1.

Timer1

- **Timer1 interrupts** The TMR1 register pair (TMR1H:TMR1L) increments from 0x0000 to 0xFFFF and rolls over to 0x0000. The Timer1 interrupt, if enabled, is generated on overflow which is latched in an interrupt flag bit, TMR1IF (bit 0 of PIR1), shown in Figure 10.8. The TMR1 overflow interrupt bit can be enabled or disabled by setting or clearing the Timer1 Interrupt Enable bit, TMR1IE (bit 0 of PIE1), shown in Figure 10.9.

Timer1

7	6	5	4	3	2	1	0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

bit 7 **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred
Note 1: This bit is unimplemented on 28-pin devices and will read as '0'

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = An A/D conversion is not complete

bit 5 **RCIF:** EUSART Receive Interrupt Flag bit
1 = The EUSART receive buffer, RCREG is full (cleared when RCREG is read)
0 = The EUSART receive buffer is empty

bit 4 **TXIF:** EUSART Transmit Interrupt Flag bit
1 = The EUSART transmit buffer, TXREG is empty (cleared when RCREG is read)
0 = The EUSART transmit buffer is full

bit 3 **SSPIF:** Master Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete (must be cleared in software)
0 = Waiting to transmit/receive

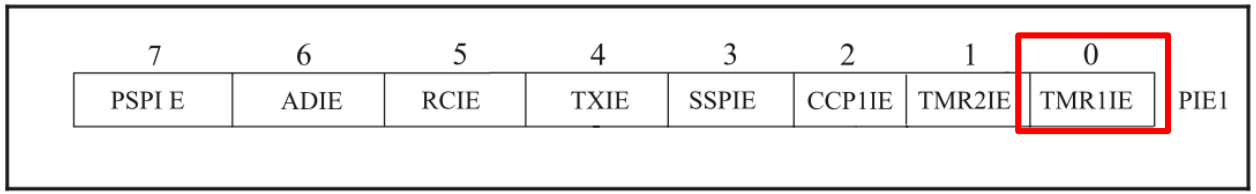
bit 2 **CCP1IF:** CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

bit 1 **TMR2IF:** TMR2-to-PR2 Match Interrupt Flag bit
1 = A TMR2-toPR2 match occurred (must be cleared in software)
0 = No TMR2-toPR2 match occurred

bit 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit,
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

FIGURE 10.8 PIR1 (Peripheral Interrupt Request) Register1

Timer1



7	6	5	4	3	2	1	0	
PSPI E	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	PIE1

bit 7 **PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RCIE:** EUSART Receive Interrupt Enable bit
1 = Enables the EUSART receive interrupt
0 = Disables the EUSART receive interrupt

bit 4 **TXIE:** EUSART Transmit Interrupt Enable bit
1 = Enables the EUSART transmit interrupt
0 = Disables the EUSART transmit interrupt

bit 3 **SSPIE:** Master Synchronous Serial Port Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE:** TMR2-to-PR2 Match Interrupt Enable bit
1 = Enables the TMR2-to-PR2 match interrupt
0 = Disables the TMR2-to-PR2 match interrupt

bit 0 **TMR1IE:** TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

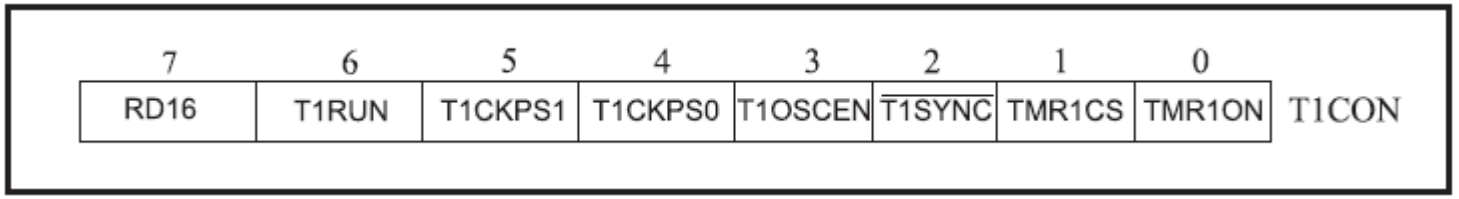
FIGURE 10.9 PIE1 (Peripheral Interrupt Enable) Register 1

Timer1

- Example 10. 5 Write a PIC18F assembly language program to provide a delay of 1 ms using Timer1 with an internal clock of 4 MHz. Use the 16-bit mode of Timer1 and the prescaler value of 1:4.

For a 4 MHz clock, each instruction cycle = 4 x (1/4 MHz) = 1 μsec
Total instruction cycles for 1 ms delay = (1 x 10^{-3} /10^{-6}) = 1000
With the prescaler value of 1:4, instruction cycles = 1000 / 4 = 250
Number of Counts for rollover = 65535_{10} - 250_{10} = 65285_{10} = 0xFF05
An extra cycle is required for rollover from 0xFFFF to 0x0000 which sets the TMR1IF to 1.
Hence, the total number of counts = 0xFF05 + 1 = 0xFF06
Therefore, TMR1H must be loaded with 0xFF, and TMR1L with 0x06.

Timer1



The PIC18F assembly language program for one ms delay is provided below:

```
        INCLUDE   <P18F4321.INC>
        MOVLW     0x60              ; 4MHz
        MOVWF     OSCCON
        MOVLW     0xA0              ; 16-bit mode, 1:4 prescaler, Timer1 OFF
        MOVWF     T1CON             ; Load into T1CON register
        MOVLW     0xFF              ; Initialize TMR1H with 0xFF
        MOVWF     TMR1H
        MOVLW     0x06              ; Initialize TMR1L with 0x06
        MOVWF     TMR1L
        BCF       PIR1, TMR1IF      ; Clear Timer1 overflow flag in PIR1
        BSF       T1CON, TMR1ON     ; Turn Timer1 ON
BACK    BTFSS     PIR1, TMR1IF      ; If TMR1IF=1, skip next instruction to halt
        GOTO      BACK
HERE    BRA       HERE              ; Halt
```

Note that the external loop can be used with the above 1 ms delay routine as the inner loop to obtain higher time delays.

Timer1

Example 10.6 Using Timer1, and write a C language program to turn on an LED connected to bit 0 of PORTC after 10 seconds. Assume a 4-MHz crystal, and a prescale value of 1:8. Assume a '0' will turn the LED OFF while a '1' will turn it ON.

(a) Use polled I/O

(b) Use interrupt I/O

Timer1

- In order to implement an I/O program using C such as Example 10.6 in the laboratory, the following configuration commands should be inserted in the C-program using the directive “#pragma config” after #include <p18f4321.h> as follows:

```
#include <p18f4321.h>
#pragma config OSC = INTIO2 // Select internal oscillator
#pragma config WDT = OFF    // Watch Dog Timer OFF
#pragma config LVP = OFF    // Low Voltage Programming OFF
#pragma config BOR = OFF    // Brown Out Reset OFF
```

Timer2

- Timer2 contains an 8-bit timer register and 8-bit period register (TMR2 and PR2). The Timer2 can only be programmed as a timer (not as a counter) with prescale values of 1:1, 1:4, and 1:16, and postscale values of 1:1 through 1:16.
- The module is controlled through the T2CON register shown in Figure 10.10. The T2CON register enables or disables the timer and configures the prescaler and postscaler. Timer2 can be shut off by clearing the control bit, TMR2ON (bit 2 of T2CON), to minimize power consumption.

Timer1

(a) Since the timer works with a divide by 4 crystal, the timer frequency = (4MHz)/4 = 1 MHz.

Instruction cycle clock period = (1/1 MHz) = 1 μs .

T1CON register will be initialized with 0x30 for an internal clock and 1:8 prescale. Time delay = Instruction cycle x Prescale value x (Count+1). Since Timer1 has a maximum prescaler of 8, maximum time delay=(1μsec)x(8)x(65536) = 0.5 seconds (approximately). Hence, an external counter is required. External counter value = (10sec/0.5)=20

```
// C-program for Timer1 polled I/O
#include <P18F4321.h>
unsigned char count;
void T1Delay();

void main()
{
  OSCCON = 0x60;                    // 4MHz
   TRISC = 0x00;                    // Port C output
   PORTCbits.RC0 = 0;               // Turn LED OFF
   T1Delay();                       // Use Timer1
   PORTCbits.RC0 = 1;               // Turn LED ON
}
void T1Delay()
{
T1CON = 0x30;                       // 16-bit, 1:8 prescale, internal clock
   for (count = 0; count < 20; count++)
   {
      TMR1H = 0;
```

```
        TMR1L = 0;
        PIR1bits.TMR1IF = 0;          // clear timer flag
        T1CONbits.TMR1ON = 1;         // start Timer0
        while (!PIR1bits.TMR1IF);     // polling, wait until timer finishes counting
        T1CONbits.TMR1ON = 0;         // Stop Timer0
    }
}
```

Timer1

(b) For interrupt I/O, we will have to loop the complete program. Although all other instructions in the loop will be repeated, execution times of these instructions are negligible compared to 10 seconds, and can be discarded.

Timer1

```
unsigned char count = 0;            // loop count
void LEDON();

#pragma code T1ISR=0x08
void T1ISR()
{
   _asm
   GOTO LEDON
   _endasm
}
#pragma code

void main()
{
  OSCCON = 0x60;                    // 4MHz
   TRISC = 0x00;                    // Port C output
   PORTCbits.RC0 = 0;               // LED off initially

   INTCONbits.PEIE = 1;             // enable peripheral interrupt
   INTCONbits.GIE = 1;              // enable global interrupt
   T1CON = 0x30;                    // 16-bit, 1:8 prescale, internal clock
   TMR1H = 0;
   TMR1L = 0;
   PIE1bits.TMR1IE = 1;             // enable Timer1 interrupt enable
   PIR1bits.TMR1IF = 0;             // clear Timer1 flag
   T1CONbits.TMR1ON = 1;            // start Timer1
   while (1);                       // wait until interrupt occurs
}
```

```
#pragma interrupt LEDON
void LEDON()
{
   if (count == 19)                 // loop count from 0 to 19
      PORTCbits.RC0 = 1;            // LED on
   else
   {
      count++;
      PIR1bits.TMR1IF = 0;          // clear Timer1 flag
   }
}
```

Timer1

- In order to implement an I/O program in PIC18F assembly such as Example 10.5 in the laboratory, the following configuration commands should be inserted using the assembler directive "config" in the PIC18F assembly language program after the "include <p18f4321.inc>" statement as follows:

```
include  <p18f4321.inc>
config  OSC  =  INTIO2;  // Select internal oscillator
config  WDT =  OFF;      // Watch Dog Timer OFF
config  LVP   =  OFF;    // Low Voltage Programming OFF
config  BOR  =   OFF;    // Brown Out Reset OFF
```

Timer2

7	6	5	4	3	2	1	0	
----------	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0	T2CON

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS3:T2OUTPS0:** Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0:** Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

FIGURE 10.10 T2CON (Timer2 Control) Register

Timer2

- **Timer2 Operation**
- In normal operation, the PR2 register is initialized to a specific value, and the **8-bit timer register (TMR2)** is incremented from 0x00 on each internal clock (FOSC/4).
- A 4-bit counter/prescaler on the clock input gives direct input, divide- by-4, and divide-by-16 prescale options. These are selected by the prescaler control bits, **T2CKPS1:T2CKPS0 (bits 1, 0 of T2CON)**.
- The value of TMR2 is compared with the 8-bit period register, PR2, on each clock cycle. When the two values match, the Timer2 outputs a HIGH on the TMR2IF flag in the PIR1 register, and also resets the value of TMR2 to 0x00 on the next cycle.
- The output frequency is divided by a counter/postscale value (1:1 to 1:16) as specified in the T2CON register. Note that the interrupt is generated and the TMR2IF flag bit in the PIR1 register is set to 1 indicating, the match between TMR2 and PR2 registers. The TMR2IF bit must be cleared to 0 using software.

Timer2

- **Timer2 Interrupt** In order for the PIC18F4321 to generate a Timer2 interrupt, the following initializations are necessary:
  1. Set PEIE and GIE bits in the INTCON register to one.
  2. Set TMR2IE bit in the PIE1 register to one.
  3. Clear TMR2IF in the PIR1 register to 0.
  4. Clear TMR2 register to 0.
  5. Initialize PR2 and T2CON registers with appropriate data.
  6. Turn the Timer2 ON by programming the T2CON register.
  7. After the Timer2 is started, TMR2 is incremented every cycle. As soon as the TMR2 value matches PR2, the PIC18F sets the TMR2IF in the PIR1 register to one, and then clears TMR2 to 0 in the next cycle. Since TMR2IE is one, the PIC18F4321 completes execution of the current instruction, and then branches to address 0x08 where the service routine is executed.

Timer2

- Example 10.7 Write a PIC18F assembly language program using Timer2 to turn ON an LED connected at bit 0 of PORT D after 10 sec. Assume an internal clock 4 MHz, a prescaler value of 1:16, and a postscaler value of 1:16.

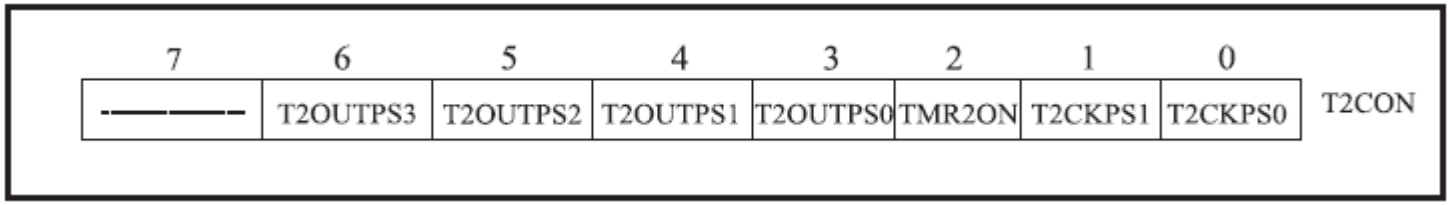
For a 4 MHz clock, each instruction cycle = 4 x 1/(4MHz) = 1 μsec. TMR2 is incremented every 1 μsec. When the TMR2 value matches with the value in PR2, the value in TMR2 is cleared to 0 in one instruction cycle. Since the PR2 is 8-bit wide, we can have a maximum counter value of 255. Let us calculate the delay with this PR2 value.

Delay = (Instruction cycle) x (Prescale value) x (Postscale value) x (Counter value + 1)
= (1 μsec) x (16) x (16) (255 + 1)
= 65.536 ms

Note that in the above, one is added to the Counter value since an additional clock is needed when it rolls over from 0xFF to 0x00, and sets the TMR2IF to 1.

External counter value for a 10 sec delay using 65.536 ms as the inner loop = (10 sec)/(65.536 ms) which is approximately 153 in decimal.

Timer2



7	6	5	4	3	2	1	0	
———	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0	T2CON

```
            INCLUDE   <P18F4321.INC>
EXT_CNT     EQU       0x50
            MOVLW     0x60              ; 4MHz
            MOVWF     OSCCON
            BCF       TRISD, TRISD0     ; Configure bit 0 of PORT D as an output
            BCF       PORTD,  RD0       ; Turn LED OFF
            MOVLW     0x7A              ; 1:16 prescaler, 1:16 postscaler Timer1 off
            MOVWF     T2CON             ; Load into T2CON register
            MOVLW     0x00              ; Initialize TMR2 with 0x00
            MOVWF     TMR2
            MOVLW     D'153'            ; Initialize EXT_CNT with 153
            MOVWF     EXT_CNT
LOOP        MOVLW     D'255'            ; Load PR2 with 255
            MOVWF     PR2
            BCF       PIR1, TMR2IF      ; Clear Timer2 interrupt flag in PIR1
            BSF       T2CON, TMR2ON     ; Set TMR2ON bit in T2CON to start timer
BACK        BTFSS     PIR1, TMR2IF      ; If TMR2IF=1, skip next instruction
            GOTO      BACK
            DECF      EXT_CNT
            BNZ       LOOP
            BSF       PORTD, RD0        ; Turn LED ON
            BCF       T2CON, TMR2ON     ; Turn off Timer2
FINISH      GOTO      FINISH            ; Halt
            END
```

Timer2

- In the above program, the execution times associated with some of the instructions such as MOVLW D’153’, MOVWF EXT_CNT, DECF EXT_CNT, and BNZ LOOP are discarded. These execution times are very small compared to a 10 sec delay.

Timer2

- Example 10.8 Using Timer2, write a C language program to turn on an LED connected to bit 0 of PORTC after 10 seconds. Assume a 4-MHz crystal.
  - (a) Use polled I/O
  - (b) Use interrupt I/O

(a) Polled I/O

Since the timer works with a divide by 4 crystal, timer frequency = (4MHz)/4 = 1 MHz. Instruction cycle clock period = 1/(1 MHz) = 1 μsec. T2CON register will be initialized with 0x7A for an internal clock and 1:16 prescale , 1:16 postscale.

Maximum Time delay = Instruction cycle x Prescale value x Postscale value x (Count+1)
= (1 μsec) x (16) x (16) x (255 +1)
= 0.06 (approximately)
External counter = (10sec) / (0.06) = 167 (approximately)

Timer2

```
// Timer2 polled I/O C-program
#include <P18F4321.h>
unsigned char count;
void T2Delay();

void main()
{
   OSCCON = 0x60;                    // 4MHz
   TRISC = 0x00;                     // Port C output
    PORTCbits.RC0 = 0;               // Turn LED OFF
   T2Delay();                        // Use Timer2
   PORTCbits.RC0 = 1;                // Turn  LED ON
}
                    void T2Delay()
                    { T2CON = 0x7A;                          // 1:16 prescale, 1:16 postscale
                       for (count = 0; count < 167; count++)
                       {
                          TMR2 = 0;
                          PR2 = 0xFF;
                          PIR1bits.TMR2IF = 0;               // clear Timer2 flag
                          T2CONbits.TMR2ON = 1;              // start Timer0
                          while (!PIR1bits.TMR2IF);          // polling, wait until timer finishes counting
                          T2CONbits.TMR2ON = 0;              // Stop Timer0
                       }
                    }
```

Timer2

- (b) For interrupt I/O, we will have to loop the complete program. Although all other instructions in the loop will be repeated, execution times of these instructions are negligible compared to 10 seconds, and can be discarded.

Timer2

```
// C-program
#include <P18F4321.h>
unsigned char count = 0;            // loop count
void LEDON();
#pragma code T2ISR=0x08
void T2ISR()
{
   _asm
   GOTO LEDON
   _endasm
}
#pragma code
void main()
{
  OSCCON = 0x60;                    // 4MHz
   TRISC = 0x00;                    // Port C output
   PORTCbits.RC0 = 0;               // LED off initially
   INTCONbits.PEIE = 1;             // enable peripheral interrupt
   INTCONbits.GIE = 1;               // enable global interrupt
   T2CON = 0x7A;                    // 16-bit, 1:8 prescale, internal clock
   TMR2 = 0;
   PR2 = 0xFF;
   PIE1bits.TMR2IE = 1;             // enable Timer2 interrupt
   PIR1bits.TMR2IF = 0;             // clear Timer2 flag
   T2CONbits.TMR2ON = 1;            // start Timer2
   while (1);                       // stay here wait for interrupt occurs then loop
}
```

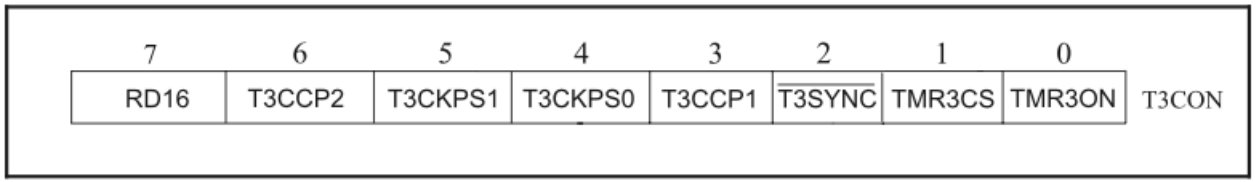
Timer2

```
#pragma interrupt LEDON
void LEDON()
{
   if (count == 166)                    // loop count = 167
    PORTCbits.RC0 = 1;                 // LED on
 else
 {
    count++;
    PIR1bits.TMR2IF = 0;               // clear Timer2 flag
 }
}
```

Timer3

- Timer3 can be used as a 16-bit timer or a 16-bit counter. Although Timer3 consists of two 8-bit registers, namely TMR3H (high byte) and TMR3L (low byte), it can only be programmed in 16-bit mode.
- The Timer3 module is controlled through the T3CON register (Figure 10.11). Some of the bits of the T3CON register are associated with the CCP module.

Timer3



bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP2:T3CCP1:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the capture/compare clock source for the CCP modules
01 = Timer3 is the capture/compare clock source for CCP2;
Timer1 is the capture/compare clock source for CCP1
00 = Timer1 is the capture/compare clock source for the CCP modules

bit 5-4 **T3CKPS1:T3CKPS0**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 2 **T3SYNC:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the device clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T13CKI (on the rising edge after the first falling edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

FIGURE 10.11 T3CON (Timer3 Control) Register

Timer3

- **Timer3 Operation** Timer3 can operate in one of three modes, namely timer, synchronous counter, and asynchronous counter. The operating mode is determined by the clock select bit, TMR3CS (bit 1 of T3CON, Figure 10.11). When TMR3CS is cleared to 0, Timer3 increments on every internal instruction cycle (FOSC/4). When the bit is set to 1, Timer3 increments on every rising edge of the Timer1 external clock input or the Timer1 oscillator, if enabled.

Timer3

- **Timer3 Interrupt** The TMR3 register pair (TMR3H:TMR3L) increments from 0x0000 to 0xFFFF and overflows to 0x0000. The Timer3 interrupt, if enabled, generates an overflow and is latched in an interrupt flag bit, TMR3IF (bit 1 of PIR2, Figure 10.12). This interrupt can be enabled or disabled by setting or clearing the Timer3 Interrupt Enable bit, TMR3IE (bit 1 of PIE2, Figure 10.13).

Timer3

7	6	5	4	3	2	1	0	
OSCFIF	CMIF	----------	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF	PIR2

bit 7 **OSCFIF:** Oscillator Fail Interrupt Flag bit
1 = Device oscillator failed, clock input has changed to INTOSC (must be cleared in software)
0 = Device clock operating

bit 6 **CMIF:** Comparator Interrupt Flag bit
1 = Comparator input has changed (must be cleared in software)
0 = Comparator input has not changed
bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIF:** Data EEPROM/Flash Write Operation Interrupt Flag bit
1 = The write operation is complete (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **BCLIF:** Bus Collision Interrupt Flag bit
1 = A bus collision occurred (must be cleared in software)
0 = No bus collision occurred

bit 2 **HLVDIF:** High/Low-Voltage Detect Interrupt Flag bit
1 = A high/low-voltage condition occurred; direction determined by VDIRMAG bit (HLVDCON<7>)
0 = A high/low-voltage condition has not occurred

bit 1 **TMR3IF:** TMR3 Overflow Interrupt Flag bit
1 = TMR3 register overflowed (must be cleared in software)
0 = TMR3 register did not overflow

bit 0 **CCP2IF:** CCP2 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

FIGURE 10.12 PIR2 (Peripheral Interrupt Request) Register 2

Timer3

7	6	5	4	3	2	1	0	
OSCFIE	CMIE	----------	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE	PIE2

bit 7 **OSCFIE:** Oscillator Fail Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 6 **CMIE:** Comparator Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIE:** Data EEPROM/Flash Write Operation Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 3 **BCLIE:** Bus Collision Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 2 **HLVDIE:** High/Low-Voltage Detect Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 1 **TMR3IE:** TMR3 Overflow Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 0 **CCP2IE:** CCP2 Interrupt Enable bit
1 = Enabled
0 = Disabled

FIGURE 10.13 PIE2 (Peripheral Interrupt Enable) Register 2

Timer3

- Example 10.9 Using Timer3, write a C language program to turn on an LED connected to bit 0 of PORTC after 10 seconds. Assume a 4-MHz crystal, and a prescale value of 1:8. Assume a '0' will turn the LED OFF while a '1' will turn it ON.
  - (a) Use polled I/O
  - (b) Use interrupt I/O

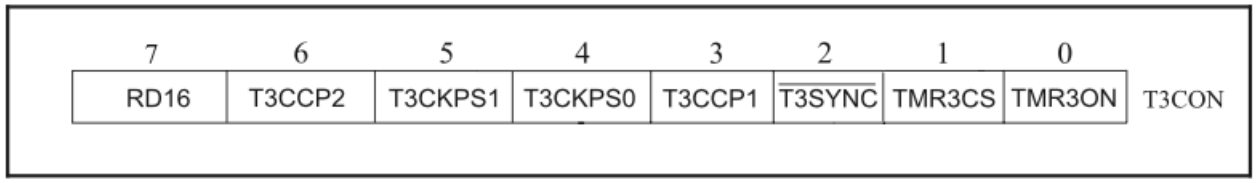
(a) Using polled I/O Since the timer works with divide by 4 crystal, timer frequency = (4MHz)/4 = 1 MHz. Instruction cycle clock period = (1/1 MHz) = 1 μsec. T3CON register will be initialized with 0x30 for an internal clock, a 16-bit timer, and a 1:8 prescale.

Maximum Time delay = Instruction cycle x Prescale value x Postscale value x (Count +1)

= (1 μsec) x (8) x (65535 +1)

= 0.5 (approximately)

Hence, External counter = (10s) / (0.5) = 20



```
// Timer3 Polled I/O
#include <P18F4321.h>
unsigned char count;
void T3Delay();

void main()
{
   OSCCON = 0x60;                    // 4MHz
   TRISC = 0x00;                     // Port C output
   PORTCbits.RC0 = 0;                // Turn LED OFF
  T3Delay();                         // Use Timer3
   PORTCbits.RC0 = 1;                // Turn  LED ON
}

void T3Delay()
{ T3CON = 0x30;                      // 16-bit, 1:8 prescale, internal clock
   for (count = 0; count < 20; count++)
   {
      TMR3H = 0;
      TMR3L = 0;
      PIR2bits.TMR3IF = 0;           // clear Timer3 flag

      T3CONbits.TMR3ON = 1;          // start Timer3
      while (!PIR2bits.TMR3IF);      // polling, wait until timer finishes counting
      T3CONbits.TMR3ON = 0;          // Stop Timer3
   }
}
```

Timer3

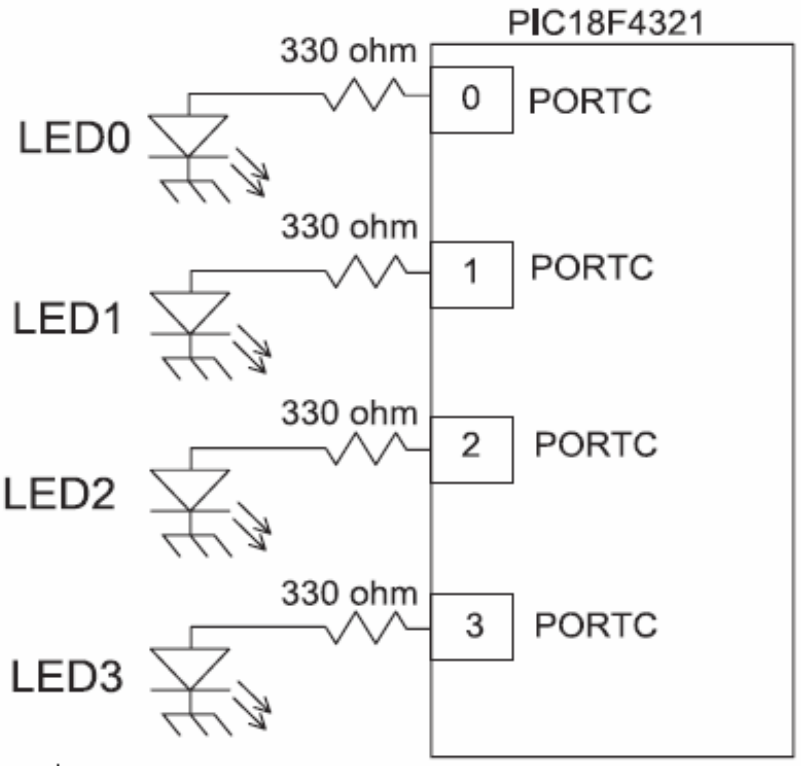
- (b) For interrupt I/O, we will have to loop the complete program. Although all other instructions in the loop will be repeated, execution times of these instructions are negligible compared to 10 seconds, and can be discarded.

Timer3

```
// C-program
#include <P18F4321.h>
unsigned char count = 0;          // loop count
void LEDON();

#pragma code T3ISR=0x08
void T3ISR()
{
   _asm
   GOTO LEDON
   _endasm
}
#pragma code

void main()
{
  OSCCON = 0x60;                  // 4MHz
   TRISC = 0x00;                  // Port C output
   PORTCbits.RC0 = 0;             // LED off initially

   INTCONbits.PEIE = 1;           // enable peripheral interrupt
   INTCONbits.GIE = 1;            // enable global interrupt
   T3CON = 0x30;                  // 16-bit, 1:8 prescale, internal clock
   TMR3H = 0;
   TMR3L = 0;
   PIE2bits.TMR3IE = 1;           // enable Timer3 interrupt
   PIR2bits.TMR3IF = 0;           // clear Timer3 interrupt
   T3CONbits.TMR3ON = 1;          // start Timer3
   while (1);                     // stay here, wait until interrupt occurs
}

#pragma interrupt LEDON
void LEDON()
{
   if (count == 19)               // loop count 20
      PORTCbits.RC0 = 1;          // LED on
   else
   {
      count++;
      PIR2bits.TMR3IF = 0;        // clear Timer3 interrupt
   }
```

Timer3

- Example 10.10. Four LEDs are connected at bits 0-3 of PORTC of PIC18F4321. Write a C-program to flash the LEDs ON or OFF based on time delay provided by Timer3 by polling the TMR3IF flag. Use maximum delay provided by Timer3, and a TMR3 prescaler of 1:4. Use 1MHz default clock.



Timer3

Solution

The Program is required to do the following:

1. Configure PORTC as an output, outputs 0's to LEDs to turn them OFF.
2. Configure T3CON for one 16-bit operation, 1:4 prescaler, and then turn Timer3 OFF.
3. Initialize TIMR3H:TMR3L to 0:0.
4. Clear TMR3IF to 0
5. Start Timer3, and then wait in a 'while' loop for the TMR3IF to become HIGH.
6. As soon as Timer3 counts up to 0xFFFF (providing maximum time delay), and overflows to 0x0000, the TMR3IF becomes 1.
7. Stop Timer3.
8. Re-initialize TMR3H:TMR3L to 0:0 for the next round.
9. Invert PORTC so that all four LEDs will be turned ON the first time through the loop.
10. Clear TMR3IF to 0.
11. The process continues.

```
#include <p18f4321.h>
void main (void)
{
      TRISC   = 0x00;                          // PortC is output
      PORTC = 0x00;
      T3CON = 0xA0;                            // Timer3 OFF with prescaler of 4
     TMR3H = 0x00;                             // TMR3H initialized to 0
     TMR3L = 0x00;                             // TMR3L initialized to 0
     PIR2bits.TMR3IF=0;                        // Clear TMR3IF to 0
     while(1)
     {
            T3CONbits.TMR3ON=1;                // Turn on TMR3
            while(PIR2bits.TMR3IF= =0);        // Wait for TMR3IF to be 1
            T3CONbits.TMR3ON=0;                // Stop TMR3
            TMR3H=0x00;                        // Re-initialize TMR3H to 0
            TMR3L=0x00;                        // Re-initialize TMR3L to 0
            PORTC= ~PORTC;                     // Invert PORTC output
            PIR2bits.TMR3IF=0;                 // Clear TMR3IF to 0
     }
}
```

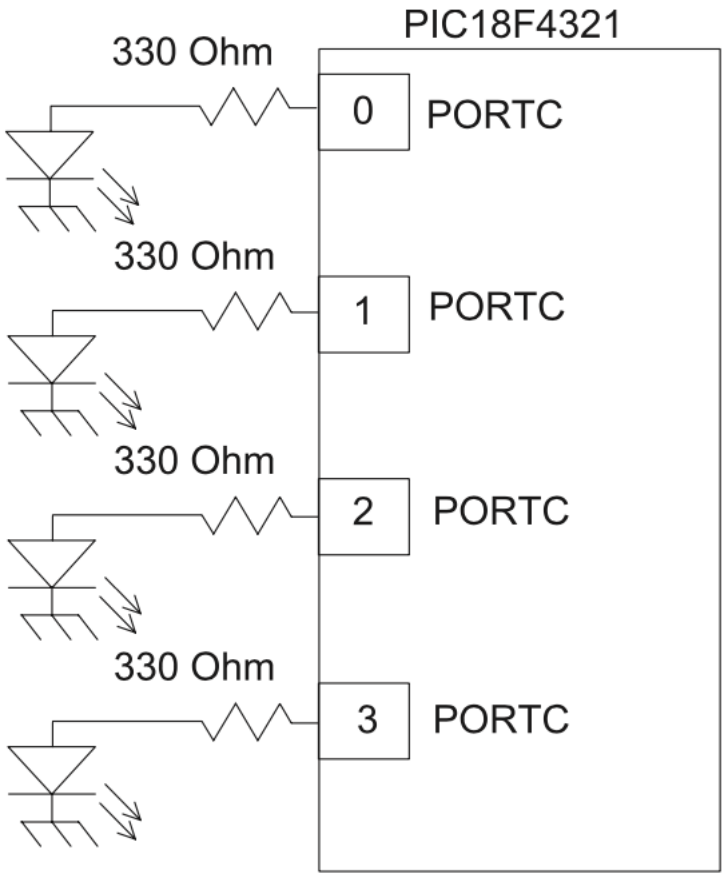
7	6	5	4	3	2	1	0	
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON	T3CON

Timer3

Example 10.11 Four LEDs are connected at bits 0-3 of PORTC of the PIC18F4321 in Figure 10.15. Use 1MHz default clock. Write a C-program to do the following:

1. Turn all LEDs OFF.
2. Turn LED0 ON using time delay provided by interrupt-driven Timer0. Use maximum time delay with a prescaler value of 1:4.
3. Turn LED1 ON using time delay provided by interrupt-driven Timer1. Use maximum time delay with a prescaler value of 1:4.
4. Turn LED2 ON using time delay provided by interrupt-driven Timer2. Use maximum value for PR2 with a prescaler value of 1:4 and a postscaler value of 1:16.
5. Turn LED3 ON using time delay provided by interrupt-driven Timer3. Use maximum time delay with a prescaler value of 1:4.
6. Repeat the process.

Timer3


PIC18F4321
330 Ohm
0
PORTC
330 Ohm
1
PORTC
330 Ohm
2
PORTC
330 Ohm
3
PORTC

Timer3

```
#include <p18f4321.h>
void TMR0_ISR (void);
void TMR1_ISR (void);
void TMR2_ISR (void);
void TMR3_ISR (void);
#pragma interrupt check_int
void check_int(void)                          // Check whitch interrupt flag is triggered
{
        if ( INTCONbits.TMR0IF = = 1)
         TMR0_ISR( );
        if ( PIR1bits.TMR1IF = =1)
         TMR1_ISR ( );
        if ( PIR1bits.TMR2IF = =1)
         TMR2_ISR ( );
        if ( PIR1bits.TMR3IF = =1)
         TMR3_ISR ( );
}
```

Timer3

```
#pragma code TMR_Int = 0x08          // At interrupt, code jumps here
void TMR_Int (void)
{
        _asm                         // Using assembly language
        GOTO check_int
        _endasm
}
#pragma code                         // End of code
```

```
void main (void)
{
        TRISC = 0x00;                   // PortC is output
        PORTC = 0x00;                   // All LEDs OFF
        T0CON = 0x21;                   // Timer0 OFF, 8-bit with prescaler of 4
        TMR0L = 0x00;                   // Value placed in lower 8 bits of TMR0
        INTCONbits.TMR0IF = 0;          // Clear TMR0 interrupt flag
        INTCONbits.TMR0IE = 1;          // Enable TMR0 interrupt
        T1CON = 0xA0;                   // Timer1 OFF, 16-bit with prescaler of 4
        TMR1H = 0x00;                   // TMR1H = 0
        TMR1L = 0x00;                   // TMR1L = 0
        PIR1bits.TMR1IF = 0;            // Clear TMR1 interrupt flag
        PIE1bits.TMR1IE = 0;            // Enable TMR interrupt
        T2CON = 0x 79;                  // Timer2 OFF, prescaler of 4 and postscaler of 16
        TMR2 = 0x00;                    // Value placed in Timer2
        PR2 = 0x FF;                    // Value to compare with Timer2
        PIR1bits.TMR2IF = 0;            // Clear TMR2 interrupt flag
        PIE1bits.TMR2IE = 1;            // Enable TMR2 interrupt
        T3CON = 0xA0;                   // Timer3 OFF, 16-bit with prescaler of 4
        TMR3H = 0x00;                   // TMR3H = 0;
        TMR3L = 0x00;                   // TMR3L = 0;
        PIR2bits.TMR3IF = 0;            // Clear TMR3 interrupt flag
        PIE2bits.TMR3IE = 1;            // Enable TMR3 interrupt

        INTCONbits.GIE = 1;             // Enable global interrupts
        INTCONbits.PEIE = 1;            // Enable peripheral interrupts
```

Timer3

```
        T0CONbits. TMR0ON = 1;      // Turn on TMR0
        while(1);                   // Do nothing while timers are counting
}
void TMR0_ISR (void)
{
        T0CONbits.TMR0ON = 0;       // Turn off TMR0
        PORTC = 0x01;               // Light ON LED connected to PORTC bit 0
        TMR0H = 0x00;               // TMR0H = 0
        TMR0L = 0x00;               // TMR0L = 0
        INTCONbits.TMR0IF = 0;      // Clear TMR0 interrupt flag
        T1CONbits.TMR1ON = 1;       // Turn on TMR1
}
void TMR1_ISR (void)
{
        T1CONbits.TMR1ON = 0;       // Turn off TMR1
        PORTC = 0x02;               // Light LED connected to PORTC bit 1
        TMR1H = 0x00;               // TMR1H = 0
        TMR1L = 0x00;               // TMR1L = 0
        PIR1bits.TMR1IF = 0;        // Clear TMR1 interrupt flag
        T2CONbits.TMR2ON = 1;       // Turn on TMR2
}
```

Timer3

```
void TMR2_ISR (void)
{
        T2CONbits.TMR2ON = 0;       // Turn off TMR2
        PORTC = 0x04;               // Light  LED connected to PORTC bit 2
        TMR2 = 0x00;                // Value placed in Timer2
        PIR1bits.TMR2IF = 0;        // Clear TMR2 interrupt flag
        T3CONbits.TMR3ON = 1;       // Turn on TMR3
}
void TMR3_ISR (void)
{
        T3CONbits.TMR3ON = 0;       // Turn off TMR3
        PORTC = 0x08;               // Light LED connected to PORTC bit 3
        TMR3H = 0x00;               // TMR3H = 0;
        TMR3L = 0x00;               // TMR3L = 0;
        PIR2bits.TMR3IF = 0;        // Clear TMR3 interrupt flag
        T0CONbits.TMR0ON =1;        // Turn on TMR0
}
```

Analog Interface

- Analog interface is provided as an on-chip feature with a microcontroller. This interface typically includes on-chip A/D (Analog to Digital) and D/A (Digital to Analog) converters. This facilitates interfacing of analog signals such as temperature, flow, and pressure with the microcontroller. **The ADC (also called A/D converter) converts an analog voltage to a binary number. The DAC (also called D/A converter), on the other hand, converts a binary number into an analog voltage or current**.
- Separate A/D and D/A converter chips are commercially available. These chips are typically used in microprocessor-based applications. The PIC18F4321 includes an on-chip A/D. However, an external D/A chip needs to be interfaced if digital-to-analog conversion is desired.
- The PIC18F on-chip ADC provides outputs of 8-bit or 10-bit. Typical ADC characteristics include **Resolution**, **Reference voltage**, **ADC channels**, **Acquisition time**, and **ADC clock**.

PIC18F on-chip ADC (A/D Converter)

- **Resolution** is one of the most important characteristics of ADCs. The resolution specifies how accurately the ADC converts the analog voltage. For example, if an analog voltage (also called 'Reference Voltage') is between 0 and 5 volt DC, then an 8-bit ADC will convert the input voltage into 8-bit binary number, and can measure it accurately up to 19mV ($5/2^8 = 5/(256) = 19$ mV). This means that an 8-bit ADC cannot distinguish the difference between 1 mV and 18 mV.
- A 10-bit ADC, on the other hand, can convert an analog voltage from 0 to 5V DC into a 10-bit binary number with an accuracy of 4.8 mV ($5/2^{10} = 5/(1024) = 4.8$ mV). Hence, the 10-bit ADC provides higher resolution.
- The PIC18F on-chip ADC is 8-bit or 10-bit, and can be selected via programming.

PIC18F on-chip ADC (A/D Converter)

- **Reference voltage** specifies the minimum and maximum voltages of the analog input voltage. Two reference voltage inputs, Vref+ (pin5 on the PIC18F4321) and Vref- (pin 4) are included. Vref+ is connected to the maximum voltage while Vref- is connected to the minimum voltage. The values of Vref+ and Vref- are selected by programming a register in the PIC18F. Common values of Vref+ and Vref- are +5VDC and 0V. When 8-bit ADC is selected via programming, 5V will be converted to 255 in decimal (FFH) and 0V to 0.

PIC18F on-chip ADC (A/D Converter)

- **ADC channels** The ADC hardware module is connected to several channels via a multiplexer. The multiplexer allows multiple analog inputs to be connected to ADC for conversion. Each channel can be connected to an analog voltage to be converted into binary by the ADC. The PIC18F includes 13 channels (AN0 through AN12).

PIC18F on-chip ADC (A/D Converter)

- **Acquisition time** When an ADC channel is selected, the voltage at this channel is used to charge an internal capacitor. It takes a certain time for the capacitor to get completely charged to the voltage at the selected channel. This charging time is called "**acquisition time**". The PIC18F ADC acquisition time can be programmed by the user. Once the acquisition time is reached, the PIC18F disconnects the selected channel from the source, and then starts the conversion.

PIC18F on-chip ADC (A/D Converter)

- The **ADC clock** that is used to perform the conversion is called the period of the ADC clock (T_{AD}). This is the same as the time required to convert one bit.
- The PIC18F4321 contains an on-chip Analog-to-Digital converter (ADC) module with 13 channels (AN0-AN12). An analog input can be selected as an input on one of these 13 channels, and can be converted to a corresponding 8-bit or 10-bit digital number.

PIC18F on-chip ADC (A/D Converter)

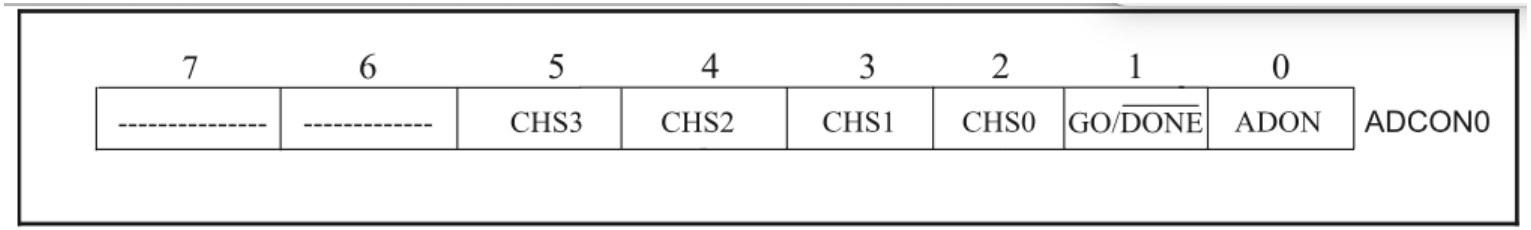
- Three 8-bit control registers (ADCON0 through ADCON2) along with two 8-bit data registers (ADRESH, ADRESL) are used in the ADC conversion process.
- The register names ADCON0, ADCON1, ADCON2, ADRESH, and ADRESL can be used while writing a PIC18F assembly or a C program using the MPLAB assembler or C18 compiler.

PIC18F on-chip ADC (A/D Converter)

- **ADCON0 register** The ADCON0 register can be loaded with data to select an analog input channel, start the conversion, check whether the ADC conversion is complete, and enable or disable the ADC module.

PIC18F on-chip ADC (A/D Converter)

- The ADCON0 register can be programmed to select one of 13 channels using bits CHS3 through CHS0 (bits 5 through 2). The conversion can be started by setting the GO/$\overline{\text{DONE}}$ (bit 1) to 1. Once the conversion is completed, this bit is automatically cleared to 0. For example, writing 01H into ADCON0 will select channel 0 (CHS3 CHS2 CHS1 CHS0 = 0000), make ADC idle (GO/$\overline{\text{DONE}}$ = 0), and then enable the ADC module.



bit 7-6 **Unimplemented:** Read as '0'
bit 5-2 **CHS3:CHS0:** Analog Channel Select bits
0000 = Channel 0 (AN0)
0001 = Channel 1 (AN1)
0010 = Channel 2 (AN2)
0011 = Channel 3 (AN3)
0100 = Channel 4 (AN4)
0101 = Channel 5 (AN5)
0110 = Channel 6 (AN6)
0111 = Channel 7 (AN7)
1000 = Channel 8 (AN8)
1001 = Channel 9 (AN9)
1010 = Channel 10 (AN10)
1011 = Channel 11 (AN11)
1100 = Channel 12 (AN12)
1101 = Unimplemented
1110 = Unimplemented
1111 = Unimplemented

bit 1 **GO/ $\overline{\text{DONE}}$:** ADC Conversion Status bit
1 = ADC conversion in progress
0 = ADC Idle

bit 0 **: ADON :** ADC On bit
1 = ADC module is enabled
0 = ADC module is disabled

FIGURE 10.16 ADCON0 (ADC Control Register0)

PIC18F on-chip ADC (A/D Converter)

```
MOVLW     01H          ; Load 01H into ADCON0
MOVWF     ADCON0
```

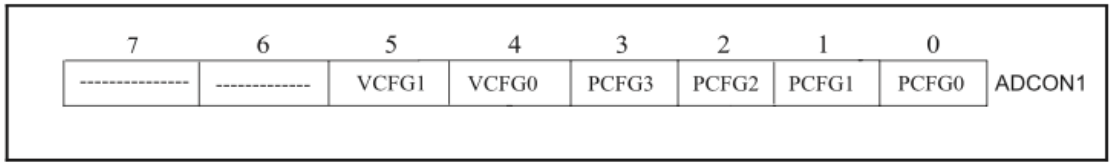
The following C statement will also accomplish this:

```
ACON0 = 01H;
```

After initializations of ADCON1 and ADCON2 (to be discussed later), the ADC can be started by setting the "GO/ $\overline{\text{DONE}}$" bit (bit 1 in ADCON0) to one. During the conversion process, GO/ $\overline{\text{DONE}}$ = 1. Upon ADC completion, this bit is cleared to 0. Both polled and interrupt I/O can be used to obtain the converted 8-bit or 10-bit binary value into ADRESH:ADRESL register pair (to be discussed later).

PIC18F on-chip ADC (A/D Converter)

- **ADCON1 register** The ADCON1 register can be initialized to specify reference voltages, and configure AN0 through AN12 pins as analog inputs or digital I/O.
- The ADCON1 register configures the functions of the port pins as Analog (A) input or Digital (D) I/O. The table shown in Figure 10.17 shows how the port bits are defined as analog or digital signals by programming the PCFG3 through PCFG0 (bits 3 through 0) of the ADCON1 register.
- Note that in the PIC18F4321, AN0 through AN12 signals are multiplexed (shared) with other signals such as digital I/O.



bit 7-6 **Unimplemented:** Read as '0'
bit 5 **VCFG1:** Voltage Reference Configuration bit (VREF- source)
1 = VREF- (AN2)
0 = VSS
bit 4 **VCFG0:** Voltage Reference Configuration bit (VREF+ source)
1 = VREF+ (AN3)
0 = VDD
bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

bit 7-6 **Unimplemented**: Read as '0'
bit 5 **VCFG1:** Voltage Regerence Configuration bit (VREF-source)
1 = VREF- (AN2)
0 = VSS
bit 4 **VCFG0:** Voltage Reference Configuration bit (VREF + source)
1 = VREF + (AN3)
0 = VDD

bit 3-0 (decimal) in Column 1: **PCFG3:PCFG0:**A/D Port Configuration Control bits

	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0	A	A	A	A	A	A	A	A	A	A	A	A	A
1	A	A	A	A	A	A	A	A	A	A	A	A	A
2	A	A	A	A	A	A	A	A	A	A	A	A	A
3	D	A	A	A	A	A	A	A	A	A	A	A	A
4	D	D	A	A	A	A	A	A	A	A	A	A	A
5	D	D	D	A	A	A	A	A	A	A	A	A	A
6	D	D	D	D	A	A	A	A	A	A	A	A	A
7	D	D	D	D	D	A	A	A	A	A	A	A	A
8	D	D	D	D	D	D	A	A	A	A	A	A	A
9	D	D	D	D	D	D	D	A	A	A	A	A	A
10	D	D	D	D	D	D	D	D	A	A	A	A	A
11	D	D	D	D	D	D	D	D	D	A	A	A	A
12	D	D	D	D	D	D	D	D	D	D	A	A	A
13	D	D	D	D	D	D	D	D	D	D	D	A	A
14	D	D	D	D	D	D	D	D	D	D	D	D	A
15	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

FIGURE 10.17 ADCON1 (A/D Control Register 1)

PIC18F on-chip ADC (A/D Converter)

- The following PIC18F assembly language instruction sequence can be used to select AN0 and AN1 as analog input, and V_{DD} and V_{SS} as reference voltages:

```
MOVLW     0x0D;      Move 0DH to WREG
MOVWF     ADCON1;    Move WREG to ADCON1
```

This is equivalent to the C statement:

```
ADCON1 = 0x0D;
```

PIC18F on-chip ADC (A/D Converter)

- From Figure 10.17, initializing ADCON1 with 0DH will mean that reference voltages are selected as follows: Vref+ = V_{DD} and Vref- = V_{SS} since bits 4 and 5 of the ADCON1 register are 00. If VDD and VSS are connected to 5V and ground, then Vref+ and Vref- will be 5V and 0V respectively. Also, since bits 3 through 0 are 0xD (13 decimal), pins AN0 and AN1 of the PIC18F4321 are configured as analog inputs and hence, analog inputs can be connected to these pins. On the other hand, pins AN2 through AN12 of the PIC18F4321 are configured as digital I/O. This can be verified using the table in Figure 10.17.

PIC18F on-chip ADC (A/D Converter)

- **ADCON2** The ADCON2 register, shown in Figure 10.18, configures the ADC clock source, sets up acquisition time, and specifies justification. After conversion, the binary output of the ADC is placed in a 16-bit register (two 8-bit register pair) ADRESH:ADRESL.

7	6	5	4	3	2	1	0	
ADFM	----------	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0	ADCON2

bit 7 **ADFM:** ADC Result Format Select bit
1 = Right justified; 10-bit result in low two bits of ADRESH and in all eight bits of ADRESL
0 = Left justified; 8-bit result in ADRESH, and the contents of ADRESL are ignored. Used for 8-bit conversion.
bit 6 **Unimplemented:** Read as '0'
bit 5-3 **ACQT2:ACQT0:** ADC Acquisition Time Select bits
111 = 20 TAD
110 = 16 TAD
101 = 12 TAD
100 = 8 TAD
011 = 6 TAD
010 = 4 TAD
001 = 2 TAD
000 = 0 TAD(1)
bit 2-0 **ADCS2:ADCS0:** A/D Conversion Clock Select bits
111 = FRC (clock derived from A/D RC oscillator)(1)
110 = FOSC/64
101 = FOSC/16
100 = FOSC/4
011 = FRC (clock derived from A/D RC oscillator)(1)
010 = FOSC/32
001 = FOSC/8
000 = FOSC/2
Note 1: If the ADC FRC clock source is selected, a delay of one TCY (instruction cycle) is added before the ADC clock starts. This allows the SLEEP instruction to be executed before starting a conversion.

FIGURE 10.18 ADCON2 (ADC Control Register 2)

PIC18F on-chip ADC (A/D Converter)

- The ADFM bit (bit 7) in the ADCON2 register is used to specify whether the converted result is “Left Justified” or “Right Justified”. By clearing the ADFM bit in ADCON2 register, the converted result is interpreted as “Left Justified”. The left justified format provides an 8-bit result in the ADRESH register; the contents of ADRESL are discarded.

PIC18F on-chip ADC (A/D Converter)

- On the other hand, the converted result can be interpreted as "Right Justified" by setting the ADFM bit in the ADCON2 register to one. The right justified format provides the 10-bit converted result in ADRESH:ADRESL with the upper two bits in bits 1, 0 of the ADRESH and lower eight bits in ADRESL; the upper six bits of ADRESH are discarded.

PIC18F on-chip ADC (A/D Converter)

- Bits 3 through 5 of the ADCON2 register are used to select ADC acquisition time. This can be selected by multiplying T_{AD} by 2, 4, 6, 8, 12, 16, and 20. According to the PIC18F4321 data sheet, a minimum value for acquisition time (T_{acq}) of 2.4 μsec is recommended.

PIC18F on-chip ADC (A/D Converter)

- Bits 0 through 2 of the ADCON2 register are used to specify the ADC clock. In the PIC18F, the ADC requires 11 T_{AD} to perform a 10-bit conversion. This can be obtained from the CPU clock (F_{OSC}) after dividing it by 2, 4, 8, 16, 32, 64.

PIC18F on-chip ADC (A/D Converter)

According to Microchip PIC18F4321 data sheet, for correct A/D conversion, T_{AD} must be as short as possible, but greater than the minimum requirement of 0.7 microseconds.

As an example, if a 1-MHz default clock is used on the PIC18F4321, T_{OSC}= 1/ (1MHz) = 1 microsecond. Using a prescaler of 2,

T_{AD} = 2 x T_{OSC}= 2 x 1 μsec = 2 μsec which is more than the minimum requirement of 0.7 microsecond. Note that dividing the frequency F_{osc} by a constant is equivalent to multiplying the period T_{osc} by the same constant.

The division factor can be calculated from the above example for other frequencies. Next, acquisition time (T_{acq}) will be calculated. Acquisition time can be selected in the PIC18F via programming to one of the following values: $2T_{AD}$, $4T_{AD}$, $6T_{AD}$, $8T_{AD}$, $12T_{AD}$, $16T_{AD}$, and $20T_{AD}$. If $2T_{AD}$ is used , then T_{acq} = 2 x T_{AD}= 2 x 2 μsec = 4 μsec which is more than the minimum value of 2.4 μsec. Hence, the timing requirements T_{AD} and T_{acq} are satisfied.

PIC18F on-chip ADC (A/D Converter)

- Figure 10.19 shows a simplified block diagram of the PIC18F4321 on-chip A/D converter. The figure shows how 13 channels (AN0 through AN12) are connected to the PIC18F4321 through a 13x1 multiplexer. One of these 13 channels are selected via programming bits CHS3 through CHS0 of the ADCON0 register. For example, writing 0100 at bits CHS3 through CHS0 of the ADCON0 register will connect the analog voltage connected to AN4 to the PIC18F4321 on-chip ADC for conversion. Also, programming bits VCFG1 and VCFG0 of the ADCON1 register will select the reference voltages.

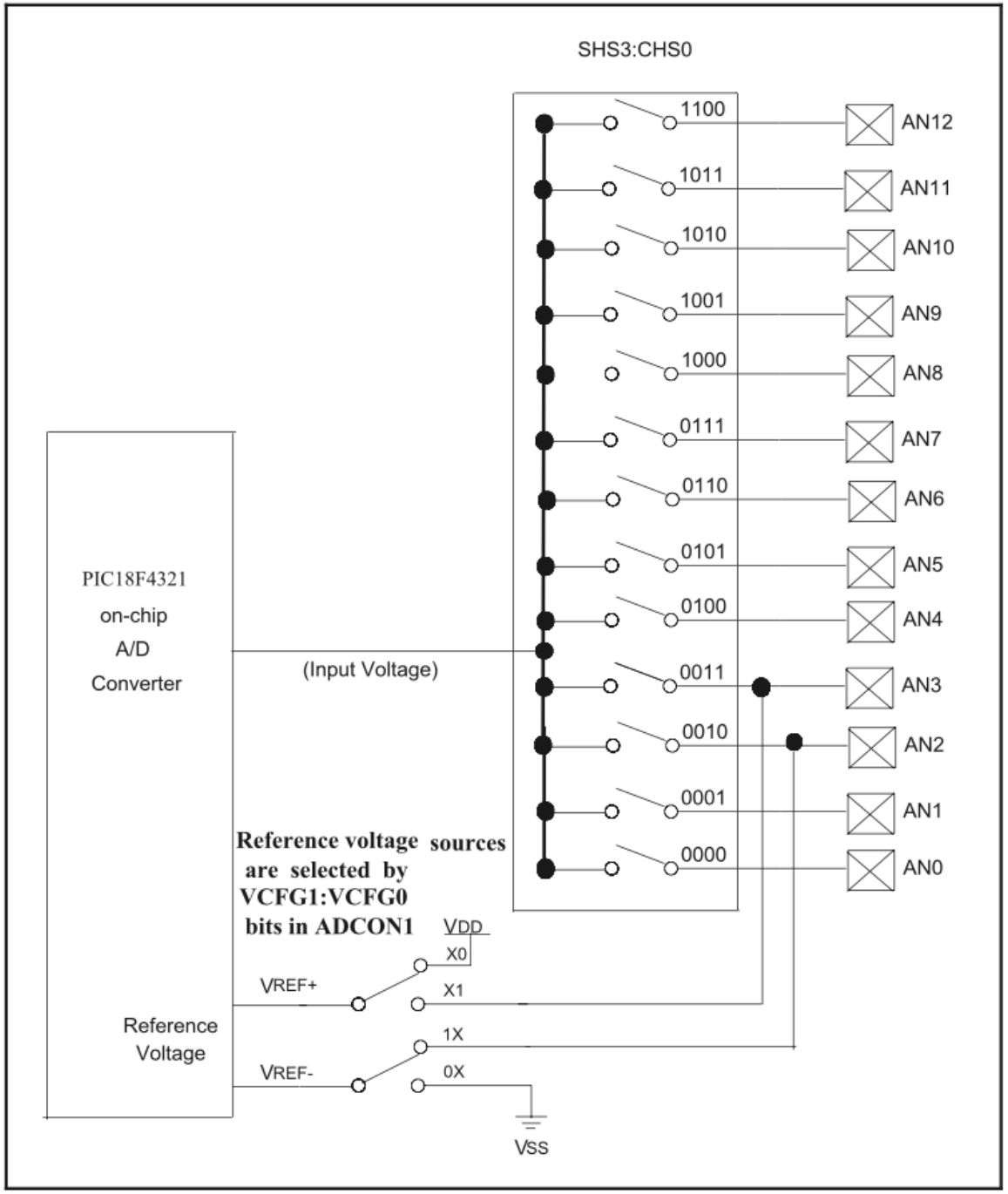


FIGURE 10.19 Block diagram of the PIC18F4321 A/D

PIC18F on-chip ADC (A/D Converter)

Polled vs. Interrupt-driven ADC The A/D conversion can be performed using polled I/O or interrupt I/O. Using polled I/O, the GO/ DONE bit in the ADCON0 register can be polled in a loop in order to check for completion of the ADC conversion. The following PIC18F assembly language instruction sequence can be used to accomplish this:

```
                BSF     ADCON0, GO;   Start ADC conversion
CONV            BTFSC   ADCON0, GO;   Wait until ADC conversion is done
                BRA     CONV
                ------
                ------
```

The above PIC18F is equivalent to the following C-code:

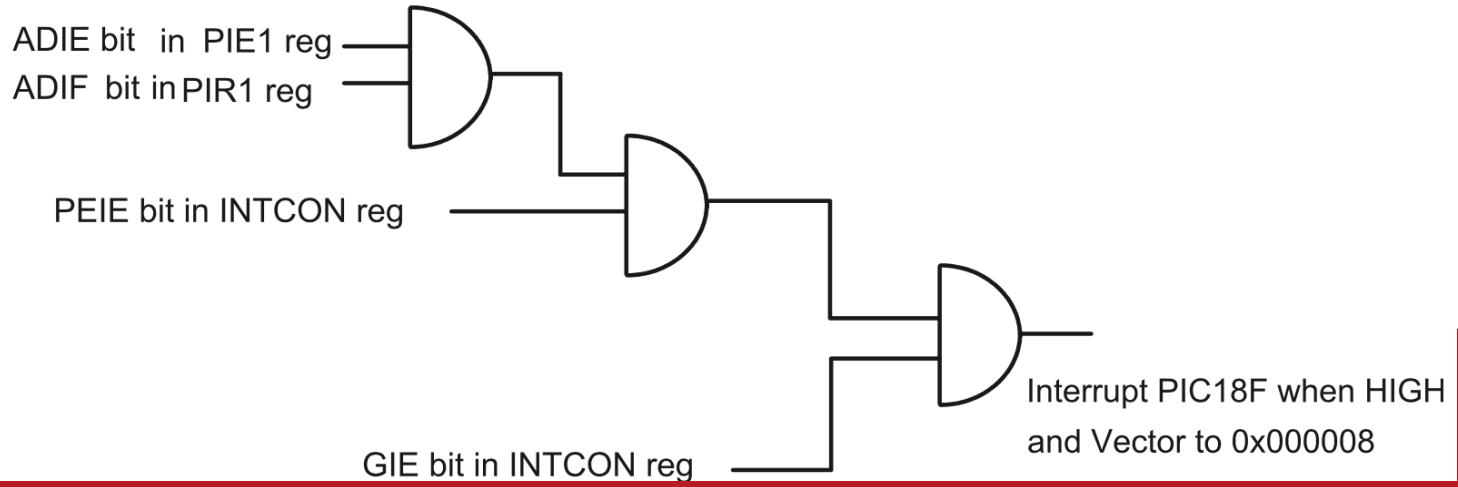
```
ADCON0bits.GO = 1;                  // Start ADC
while (ADCON0bits.DONE ==1); // Wait until ADC conversion is DONE
-----
-----
```

PIC18F on-chip ADC (A/D Converter)

- Note that in the above C-code, “GO” in the statement, “ADCON0bits.GO = 1;” is used to start the ADC conversion while “DONE” in the statement, “while (ADCON0bits. DONE ==1);” rather than “ while (ADCON0bits.GO ==1);” is used for polling the GO/ $\overline{\text{DONE}}$ bit (bit 1 of ADCON0).
- Both statements will work with the C18 compiler since they have the same bit number in ADCON0. The “while (ADCON0bits.DONE ==1);” is used for polling in order to indicate that “wait until ADC conversion is DONE”. This is used for better readability.

PIC18F on-chip ADC (A/D Converter)

- For interrupt I/O, from Figure 10.20:
  1. Set the ADIE bit in the PIE1 register to one. This will enable the ADC interrupt.
  2. Clear the ADIF bit in the PIR1 register to zero. Note that the ADIF bit is automatically set to one by the processor after interrupt occurs upon completion of the A/D conversion..
  3. Set the PEIE bit in the INTCON register to one. This will enable the peripheral interrupt. Note that ADC is considered as a peripheral device.
  4. Set the GIE bit in the INTCON register to one.
  5. Start the A/D conversion by setting the GO/DONE bit in the ADCON0 register to one.
  6. Wait for interrupt to occur, indicating completion of A/D conversion.



PIC18F on-chip ADC (A/D Converter)

The above steps can be accomplished using the following PIC18F assembly language instruction sequence:

```
        BSF PIE1 , ADIE         ; Enable ADC interrupt
        BCF PIR1, ADIF          ; Clear ADC interrupt flag bit
        BSF INTCON, PEIE        ; Enable PIC18F interrupts
        BSF INTCON, GIE         ; Enable global interrupt
        BSF ADCON0, GO          ; Start ADC
WAIT    BRA WAIT                ; Wait for conversion to complete
```

This is equivalent to the following C-language statements:

```
PIE1bits.ADIE = 1;          // Set ADIE bit in PIR1 register to one
PIR1bits.ADIF = 0;          // Clear ADIF bit in PIR1 register to zero
INTCONbits.PEIE = 1;        // Enable peripheral interrupt
INTCONbits.GIE  = 1;        // Enable global interrupt
ADCON0bits.GO = 1;          // Start A/D conversion
while (1);                  // Wait for conversion to complete
```

PIC18F on-chip ADC (A/D Converter)

- As soon as A/D conversion is completed, the ADIF bit in the PIR1 register is automatically set to one. The PIC18F4321 completes execution of the current instruction, and then branches to address 0x08 (high priority interrupt address vector upon power-on reset). Note that the low priority can be assigned to ADC interrupt by clearing the ADIP (ADC interrupt priority bit) in the IPR1 register to zero. In that case, the interrupt address vector will be 0x018.

PIC18F on-chip ADC (A/D Converter)

- **Converting the 8-bit contents of ADRESH into Voltage**
- Note that the maximum decimal value that can be accommodated in 8 bits of ADRESH is 255_{10} (FF_{16}). Hence, the maximum voltage of 5V will be equivalent to 255_{10}. This means that 1 volt = 51 (decimal); this factor of 51 will be used to convert the 8-bit ADC output in the ADRESH into analog voltage.

PIC18F on-chip ADC (A/D Converter)

- Suppose that the decimal equivalent of the 8-bit ADC output in ADRESH register is X, the integer part of the voltage is Y (after converting to Voltage), and the fractional part (after converting to Voltage) is Z.
  - i.e., X = Y.Z

Therefore, voltage equivalent of $Y = 5 \times (X/255)$

$= X/51$

$=$ Quotient + Remainder

Note that Quotient is the integer part, Y of the voltage. The fractional part Z can be calculated from the Remainder as follows: $Z =$ (Remainder/51) x 10 which is approximately (Remainder/5). This will provide the result up to one digit after the decimal point.

For example, suppose that the decimal equivalent of the 8-bit output of A/D is 200. $Y = 200/51 = 3$ with Remainder 47. Hence, $Z = (47/5) = 9$ with remainder 2. This will provide a voltage of 3.9. However, if more fractional digits are desired, they can be obtained as follows. For example, the second fractional digit can be obtained from (2/51) x 100 = 4 (approximately). The final result will be 3.94.

PIC18F on-chip ADC (A/D Converter)

Example 10.12 A PIC18F4321 microcontroller shown in Figure 10.21 with 1MHz internal clock is used to implement a voltmeter to measure voltage in the range 0 to 5 V and display the result in two decimal digits: one integer part and one fractional part. Using both polled I/O and interrupt I/O,

(a) Write a PIC18F assembly language program to accomplish this.

(b) Write a C language program to accomplish this.

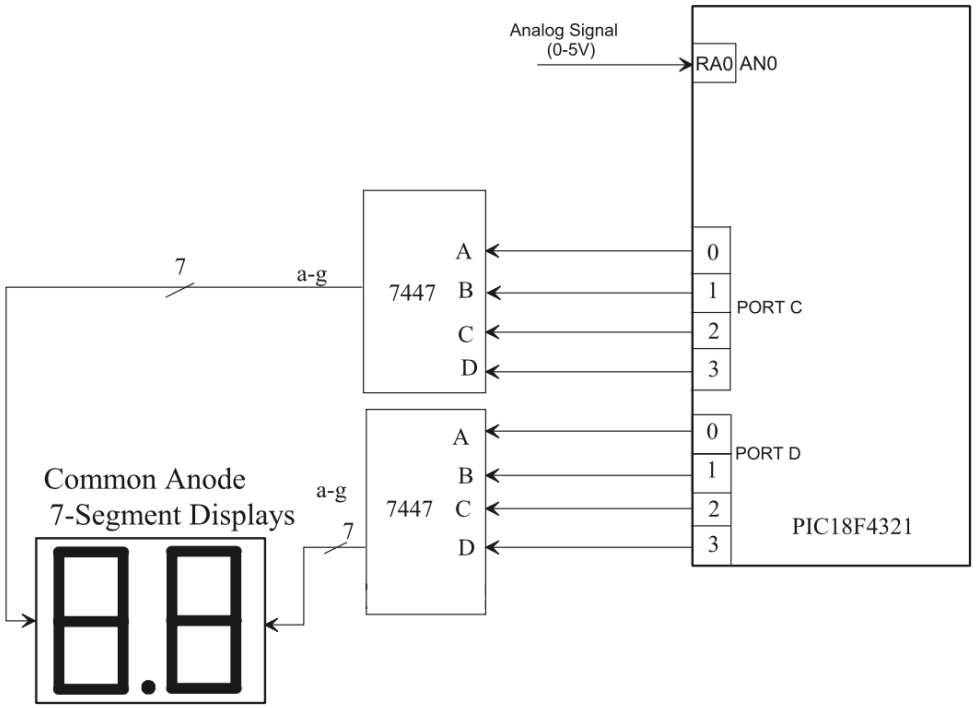


FIGURE 10.21 Figure for Example 10.12

PIC18F on-chip ADC (A/D Converter)

- Three registers ADCON0-ADCON2 need to be configured. In **ADCON0**, analog input AN0 is designated as the analog signal to be converted. Hence, CHS3-CHS0 bits (bits 5-2) are programmed as 0000 to select channel 0 (AN0). The ADCON0 register is also used to enable the A/D, start the A/D, and then check the “End of conversion” bit. In the PIC18F assembly language program provided below, the ADCON0 is loaded with 0x01 which will select AN0, and enable A/D.

PIC18F on-chip ADC (A/D Converter)

- The reference voltages (0V and 5V) are chosen by programming the ADCON1 register. In this example, V_{DD} (by clearing bit 4 of ADCON1 to 0), and V_{SS} (by clearing bit 5 of ADCON1 to 0) will be used. Note that V_{DD} and V_{SS} already connected to the PIC18F4321. The ADCON1 register is also used to configure **AN0 (bit 0 of PORT A) as an analog input by writing 1101** (13 decimal in Figure 10.17) at PCFG3-PCFG0 (bits 3-0 of ADCON1). Note that there are several choices to configure AN0 as an analog input. In the program, the ADCON1 is loaded with 0x0D which will select V_{SS} and V_{DD} as reference voltage sources, and AN0 as analog input.

bit 3-0 (decimal) in Column 1: **PCFG3:PCFG0:**A/D Port Configuration Control bits

	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0	A	A	A	A	A	A	A	A	A	A	A	A	A
1	A	A	A	A	A	A	A	A	A	A	A	A	A
2	A	A	A	A	A	A	A	A	A	A	A	A	A
3	D	A	A	A	A	A	A	A	A	A	A	A	A
4	D	D	A	A	A	A	A	A	A	A	A	A	A
5	D	D	D	A	A	A	A	A	A	A	A	A	A
6	D	D	D	D	A	A	A	A	A	A	A	A	A
7	D	D	D	D	D	A	A	A	A	A	A	A	A
8	D	D	D	D	D	D	A	A	A	A	A	A	A
9	D	D	D	D	D	D	D	A	A	A	A	A	A
10	D	D	D	D	D	D	D	D	A	A	A	A	A
11	D	D	D	D	D	D	D	D	D	A	A	A	A
12	D	D	D	D	D	D	D	D	D	D	A	A	A
13	D	D	D	D	D	D	D	D	D	D	D	A	A
14	D	D	D	D	D	D	D	D	D	D	D	D	A
15	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

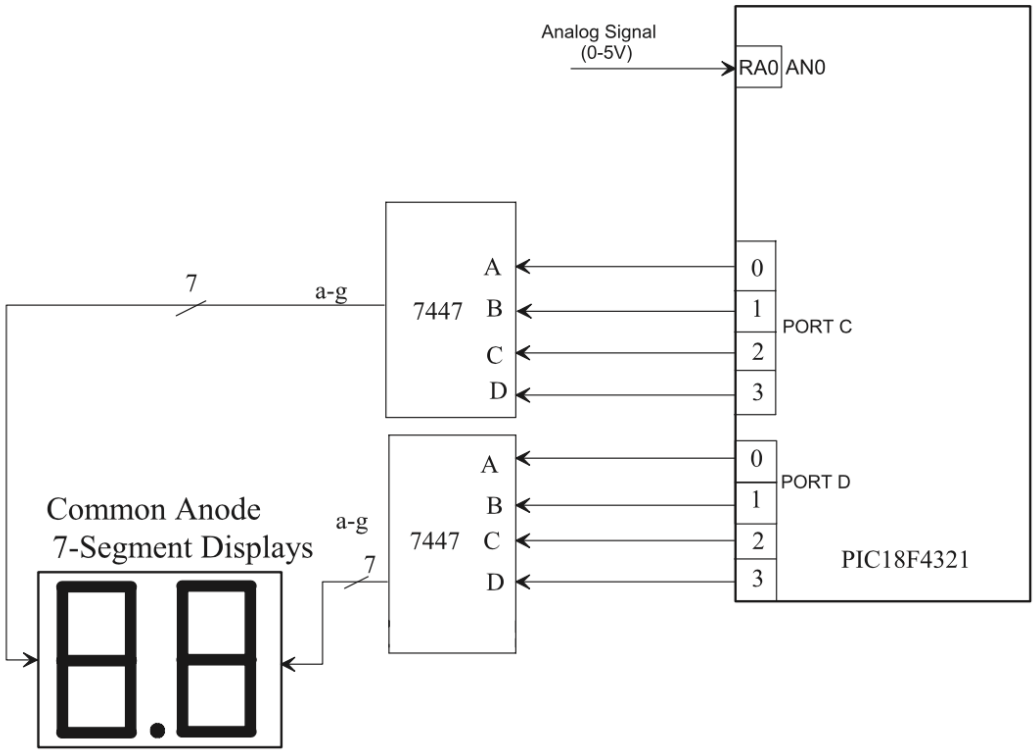
FIGURE 10.17 ADCON1 (A/D Control Register 1)

PIC18F on-chip ADC (A/D Converter)

- In the program, the ADCON2 is loaded with 0x08 which will provide the 8-bit result left justified, select 2 T_{AD}, and select Fosc/2.
- The ADCON2 is used to set up the acquisition time, conversion clock, and also if the result is to be left or right justified. In this example, an 8-bit result is assumed. The A/D result is configured as left justified, and therefore, the 8-bit register ADRESH will contain the result. The contents of ADRESL are ignored. The contents of ADRESH are divided by 51 to obtain the integer part of the voltage. The fractional part of the voltage can be obtained by dividing the remainder by 5.

PIC18F on-chip ADC (A/D Converter)

- The final result will be in decimal with one integer digit (Integer part) followed by one digit after decimal point (Fractional part). Both integer and fractional parts of the result will be outputted to two 7447's (BCD to 7-segment decoder) in order to display them on two seven-segment displays arranged in a row as shown in Figure 10.21.



PIC18F on-chip ADC (A/D Converter)

```
                INCLUDE   <P18F4321.INC>
D0              EQU       0x30          ; contains data for right (fractional) 7-seg
D1              EQU       0x31          ; contains data for left (integer) 7-seg
ADCONRESULT     EQU       0x34          ; contains 8-bit A/D result
                ORG       0x100         ; starting address of the program
                MOVLW     0x10          ; Initialize STKPTR to 0x10 (arbitrary value)
                MOVWF     STKPTR        ; since subroutines are used
                CLRF      TRISC         ; Configure PortC  as output
                CLRF      TRISD         ; Configure PortD  as output
                SETF      TRISA         ; Configure PortA  as input
                MOVLW     0x01
                MOVWF     ADCON0        ; Select AN0 for input and enable ADC
                MOVLW      0x0D
                MOVWF     ADCON1        ; Select VDD and VSS as reference
                                        ; voltages and AN0 as analog input.
                MOVLW     0x08
                MOVWF     ADCON2        ; Select Left justified, 2TAD and Fosc/2
```

PIC18F on-chip ADC (A/D Converter)

```
START   BSF     ADCON0, GO          ; Start A/D conversion
INCONV  BTFSS   ADCON0, DONE        ; Wait until A/D conversion is done
        BRA     INCONV
        MOVFF   ADRESH,ADCONRESULT  ; Move ADRESH of result
                                    ; into ADCONRESULT register
        CALL    DIVIDE              ; Call the divide subroutine
        CALL    DISPLAY             ; Call display subroutine
        BRA     START
```

PIC18F on-chip ADC (A/D Converter)

```
DIVIDE      CLRF    D0              ; Clears  D0
            CLRF    D1              ; Clears D1
            MOVLW   D'51'           ; #1 Load 51 into WREG
EVEN        CPFSEQ  ADCONRESULT     ; #2
            BRA     QUOTIENT        ; #3
            INCF    D1, F           ; #4
            SUBWF   ADCONRESULT, F  ; #5
QUOTIENT    CPFSGT  ADCONRESULT     ; #6 Checks if ADCONRESULT
                                    ; still greater than 51
            BRA     DECIMAL         ; #7
            INCF    D1, F           ; #8 increment D1 for each time
                                    ; ADCONRESULT is greater than 51
            SUBWF   ADCONRESULT, F  ; #9 Subtract 51 from
                                    ; ADCONRESULT
```

PIC18F on-chip ADC (A/D Converter)

```
                BRA     EVEN            ; #10
DECIMAL         MOVLW   0x05            ; #11
REMAINDER       CPFSGT  ADCONRESULT     ; #12 Checks if ADCONRESULT
                                        ; greater than 5
                BRA     DIVDONE         ; #13
                INCF    D0, F           ; #14 Increment D0 each
                SUBWF   ADCONRESULT, F  ; #15 Subtract 5
                                        ; from ADCONRESULT
                BRA     REMAINDER
DIVDONE         RETURN                  ; #16
DISPLAY         MOVFF   D1, PORTC       ; #17 Output D1  on integer 7-seg

                MOVFF   D0, PORTD       ; #18 Output D0  on fractional 7-seg
                RETURN
                END
```

PIC18F on-chip ADC (A/D Converter)

- Since the PIC18F does not have any unsigned division instruction, a subroutine called DIVIDE is written to perform unsigned division using repeated subtraction. In the DIVIDE subroutine, the output of the A/D contained in the ADCONRESULT register is subtracted by 51. Each time the subtraction result is greater than 51, the contents of register D1 (address 0x31) is incremented by one, which would yield the integer part of the answer.

PIC18F on-chip ADC (A/D Converter)

- Once the contents of the ADCONRESULT reaches a value below 51, the remainder part of the answer is determined. This is done by subtracting the number in ADCONRESULT subtracted by 5. Each time the subtraction result is greater than 5, register D0 (address 0x30) is incremented by one. Finally, the integer value is placed in D1 and the remainder part is placed in D0. Now the only task left is to display the result on the 7-segment display.

PIC18F on-chip ADC (A/D Converter)

- Line#1 moves 51 (decimal) into WREG.
- The CPFSEQ at Line#2 compares the A/D's 8-bit result in ADCONRESULT with 51 for equality. Suppose that the analog input voltage at AN0 is one volt, which is 51 in decimal. Since [WREG] = [ADCONRESULT] = 51, the program branches to line #4, and increments [D1] by 1, storing 1 in D1.
- The SUBWF ADCONRESULT, F at Line #5 subtracts [WREG] from [ADCONRESULT] and stores the result in ADCONRESULT.

PIC18F on-chip ADC (A/D Converter)

- Since the subtraction result is 0 in this case, ‘0’ is stored in ADCONRESULT. The CPFSGT ADCONRESULT instruction at Line #6 compares [WREG] with [ADCONRESULT] to check whether [ADCONRESULT] > [WREG].
- Since [WREG] = 51 and [ADCONRESULT] = 0, the program executes BRA DECIMAL at Line #7, and branches to label DECIMAL at Line #11 where 5 is moved into WREG.

PIC18F on-chip ADC (A/D Converter)

- The CPFSGT ADCONRESULT at Line #12 is then executed to check whether [ADCONRESULT] > [WREG]. Since [WREG] = 5 and [ADCONRESULT] = 0, the program executes BRA DIVDONE at Line #13, and branches to label DIVDONE at Line #16 where the RETURN instruction is executed. The program returns to the “CALL DISPLAY”-- one instruction after “CALL DIVIDE”.
- The program pushes the address of the next instruction “BRA START” onto the hardware stack, and executes the subroutine called DISPLAY (Line #17). The instruction “MOVFF D1, PORTC” at Line #17 outputs [D1] = 0x01 to PORTC. Hence, ‘1’ is displayed on the integer display.

PIC18F on-chip ADC (A/D Converter)

- Note that the BCD number ‘1’ of the integer part is contained in low four bits of D1 which are output to the DCBA inputs of the integer 7447 to display a ‘1’ on the left (integer) 7-segment display. The instruction “MOVFF D0, PORTD” outputs [D0] = 0x00 to PORTD, and a ‘0’ is displayed on the right (fractional) display. Outputting to integer and fractional displays using instructions in sequence are executed so fast by the PIC18F that the displays appear to human eyes at the same time. Finally, “1.0” indicates that 1.0 volt is displayed on the two seven-segment displays.

PIC18F on-chip ADC (A/D Converter)

- Suppose that the decimal value contained in the ADCONRESULT (A/D converter's output) is 200 (decimal) which is equivalent to 3.9 volts. Line#1 moves 51 (decimal) into WREG. The CPFSEQ at Line#2 compares the A/D's 8-bit result in ADCONRESULT with 51 for equality. Since [WREG] = 51, and [ADCONRESULT] = 200, the program executes the instruction "BRA QUOTIENT" at line #3, and branches to Line #6. The CPFSGT ADCONRESULT instruction at Line #6 compares [WREG] with [ADCONRESULT] to check whether [ADCONRESULT] > [WREG].

PIC18F on-chip ADC (A/D Converter)

- Since [WREG] = 51 and [ADCONRESULT] = 200, the program branches to Line #8, and increments [D1] by 1, and stores the result in D1. The instruction "SUBWF ADCONRESULT, F" at Line #9 is then executed, and [WREG] is subtracted from [ADCONRESULT], and the result is stored in ADCONRESULT. Since [WREG] = 51 and [ADCONRESULT] = 200, the subtraction result 149 will be stored in ADCONRESULT. The instruction "BRA EVEN" at Line #10 is executed next. The program branches to label EVEN at line #2 to execute the instruction "CPFSEQ ADCONRESULT", and the loop is repeated until the result of subtraction in ADCONRESULT is less than 51.

PIC18F on-chip ADC (A/D Converter)

- This will happen in this case when [D1] = 3, and [ADCONRESULT] = 47 = subtraction result of "[ADCONESULT]- [WREG]" after going through the loop 3 times. As soon as [ADCONRESULT] < [WREG], the instruction "BRA DECIMAL' at Line #7 is executed where the fractional part '9' is determined in the same manner as the last example. The rest of the logic is very similar to the last example. Finally, "3.9" will be displayed on the two seven-segment displays.

Using interrupt I/O, the PIC18F assembly language program for the voltmeter

```
                INCLUDE <P18F4321.INC>
                D0 EQU 0x30
                D1 EQU 0x31
ADCONRESULT     EQU 0x34
                ORG 0x00; Reset
                BRA MAIN_PROG     ; Main Program
                ORG 0x100
MAIN_PROG       MOVLW 0x10        ; Initialize STKPTR
                MOVWF STKPTR      ; Move 10H to STKPTR
                CLRF TRISC        ; Configure PortC to be output
                CLRF TRISD        ; Configure PortD to be output
                MOVLW 0x01        ; Move 01H to WREG
                MOVWF ADCON0      ; Select AN0 for input and enable A/D
                MOVLW 0x0D        ; Set Vdd and Vss as reference voltages
                MOVWF ADCON1      ; and AN0 as analog input
                MOVLW 0x08        ; Select Left justify, 2TAD, FOSC/2
                MOVWF ADCON2
                BSF PIE1 , ADIE   ; Enable ADC interrupt
                BCF PIR1, ADIF    ; Clear ADC interrupt flag bit
                BSF INTCON, PEIE  ; Enable PIC18F interrupts
                BSF INTCON, GIE
                BSF ADCON0, GO    ; Start ADC
```

PIC18F on-chip ADC (A/D Converter)

```
WAIT        BRA WAIT              ; Wait for conversion to be complete
            BRA MAIN_PROG         ; Return to Main Program
                                  ; Interrupt Service Routine
            ORG 0x0008            ; Interrupt address vector
            BCF PIR1,ADIF         ; Clear ADC interrupt flag bit
            MOVFF ADRESH, ADCONRESULT      ; Store ADRESH
            CALL DIVIDE           ; Calls divide subroutine
            CALL  DISPLAY         ; Calls display subroutine
            RETFIE                ; Enable interrupt and return to main
DIVIDE      CLRF D0               ; Clears D0
            CLRF D1               ; Clears D1
            MOVLW D'51'           ; Load 51 into WREG
EVEN        CPFSEQ ADCONRESULT; Compare with 51 for equality
            BRA QUOTIENT          ; Branch to QUOTIENT
            INCF D1, F            ; Increment [D1] by 1, store in D1
            SUBWF ADCONRESULT, F      ; Subtract [WREG] from
                                      ; [ADCONRESULT],
                                  ; store in ADCONRESULT
```

PIC18F on-chip ADC (A/D Converter)

```
QUOTIENT CPFSGT ADCONRESULT   ; Checks if ADCONRESULT is still greater than 51
                BRA DECIMAL   ; Branch to DECIMAL
                INCF D1, F    ; Increment D1 for each time
                              ; ADCONRESULT is greater than 51
                SUBWF ADCONRESULT, F   ; Subtract 51 from ADCONRESULT
                BRA EVEN      ; Branch to EVEN
DECIMAL         MOVLW 0x05    ; Move 5 into WREG
REMAINDER CPFSGT ADCONRESULT  ; Checks if ADCONRESULT is greater
                              ; than 5
                BRA DIVDONE   ; Branch to DIVDONE
                INCF D0, F    ; Increment D0
                SUBWF ADCONRESULT, F   ; Subtract 5 from ADCONRESULT
                BRA REMAINDER ; Branch to REMAINDER
DIVDONE         RETURN        ; Execute RETURN instruction
DISPLAY         MOVFF D1, PORTC  ; Output D1 on integer 7-seg
                MOVFF D0, PORTD  ; Output D0 on fractional 7-seg
                RETURN
                END
```

PIC18F on-chip ADC (A/D Converter)

Using polled I/O, the C- program for the Voltmeter is provided in the following:

```
#include <P18F4321.h>
unsigned int result; //Initialize variable
void main ( )
{
TRISC = 0;                          // Configure  Port C as output
TRISD = 0;                          // Configure PORT D as output
ADCON0 = 0x01;                      // Select   channel 0 for AN0 and enable ADC module
ADCON1 = 0x00;                      // Select reference voltages  0V and 5V, enable AN0
ADCON2 = 0x08;                      // Left justified, 8-bit result in ADRESH, 2TAD, FOSC/2
ADCON0bits.GO = 1;                  // Start the ADC
while(ADCON0bits.DONE == 1); // Wait until conversion is complete
result = ADRESH;                    // Move the converted  8-bit data into result
PORTC = result/51;
PORTD = (result%51)/5;
ADCON0bits.GO = 1;                  // Start A/D converter again
}
```

PIC18F on-chip ADC (A/D Converter)

Using interrupt I/O, the C- program for the Voltmeter is provided in the following :

```
#include <p18f4321.h>
unsigned int result;
void convert (void);
#pragma code ADCINT=0x08  // At interrupt, code jumps here
void ADCINT (void)
{ _asm

GOTO  convert
_endasm
}
#pragma code // End code
```

PIC18F on-chip ADC (A/D Converter)

```
void main()
{
TRISC = 0;                  // Configure Port C as output
TRISD = 0;                  // Configure Port D as output
PIE1bits.ADIE = 1;          // enable ADC interrupt
PIR1bits.ADIF = 0;          // clear ADC interrupt flag
INTCONbits.PEIE = 1;        // enable peripheral interrupt
INTCONbits.GIE = 1;         // enable global interrupt
ADCON0 = 0x01;              // Select   channel 0 for AN0 and enable ADC module
ADCON1 = 0x00;              // Select reference voltages  0V and 5V, enable AN0
ADCON2 = 0x08;              // Left justified, 8-bit result in ADRESH, 2TAD, FOSC/2
```

PIC18F on-chip ADC (A/D Converter)

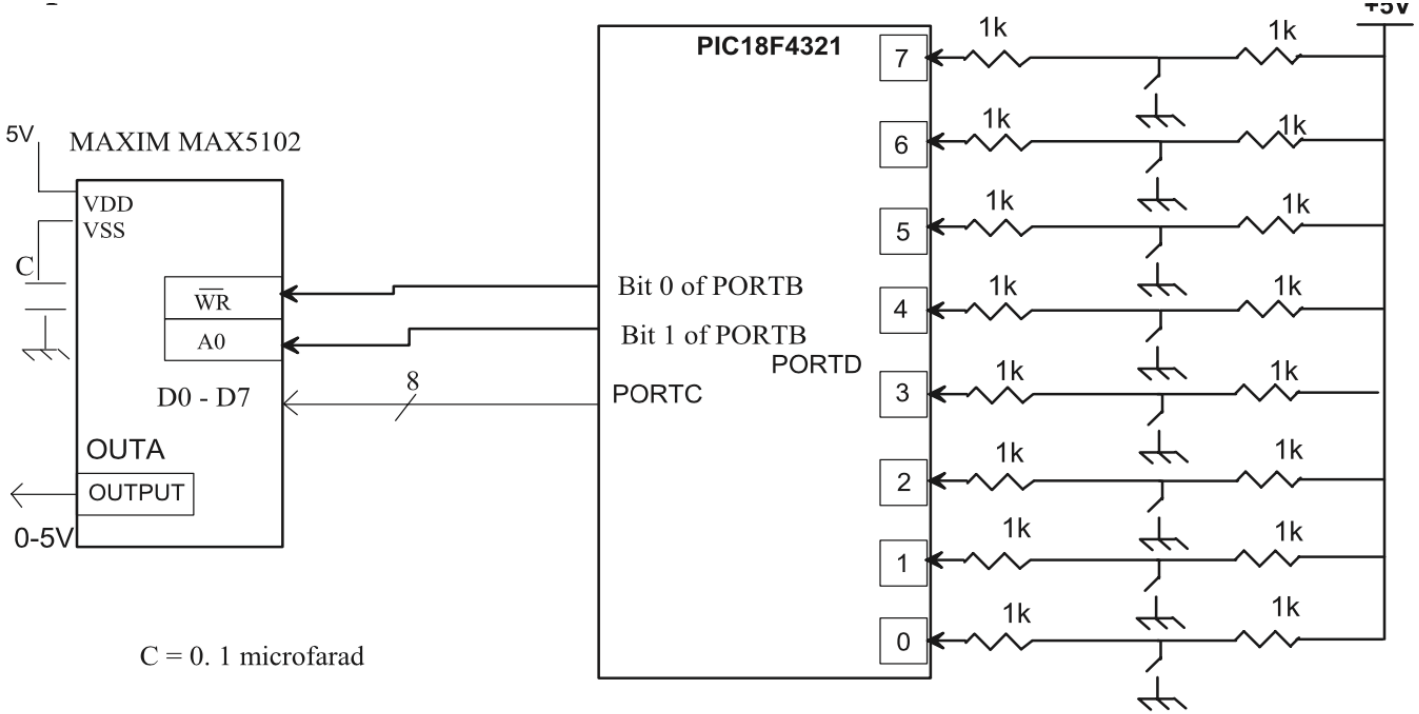
```
ADCON0bits.GO = 1;          // Start A/D conversion
while(1);                   // Wait for interrupt
}
#pragma interrupt convert
void convert (void)
{
PIR1bits.ADIF = 0;          // Clear ADIF to 0
result = ADRESH;
PORTC = result/51;
PORTD = (result%51)/5;
ADCON0bits.GO = 1;          // Start A/D conversion again
}
```

Interfacing an external D/A Converter

- As discussed before, most microcontrollers such as the PIC18F4321 do not have any on-chip D/A converter (or sometimes called DAC). Hence, an external D/A converter chip is interfaced to the PIC18F4321 to accomplish this function.
- In order to illustrate the basic concepts associated with interfacing a D/A coverter to the PIC18F, a typical D/A converter such as the Maxim MAX5102 is interfaced to the PIC18F4321.

Interfacing an external D/A Converter

- Example 10.13: Assume the block diagram of Figure 10.22. Write a C program that will input eight switches via PORTD of the PIC18F4321, and output the byte to D0-D7 input pins of the MAX5102 D/A converter. The microcontroller will send appropriate signals to the WR and A0 pins so that the D/A converter will convert the input byte to an analog voltage between 0V and 5V, and output the converted voltage on its OUTA pin. Use 1-MHz internal clock.



Interfacing an external D/A Converter

- Solution
  1. Configure PORTB and PORTC as outputs, and PORTD as an input.
  2. Output a LOW to A0 Pin of the D/A via bit 1 of PORTB to select OUTA.
  3. Output a LOW to WR pin of the D/A via bit 0 of PORTB.
  4. Input the switches via PORTD, and output to PORTC.
  5. Output a HIGH to WR pin of the D/A via bit 0 of PORTB to latch an 8-bit input data for converting to analog voltage. No delay is needed since the program will be written to input one byte of data from the switches.

Interfacing an external D/A Converter

The C language program is provided below:

```
#include <p18f4321.h>
void main (void)
{
        TRISB=0x00;          // Configure PORTB as output
        TRISC=0x00;          // Configure PORTC as output
        TRISD=0xFF;          // Configure PORTD as input
        PORTBbits.RB1=0;     // Clear A0 to 0 to select OUTA
        PORTBbits.RB0=0;     // Output LOW on  bit 0 of PORTB
        PORTC=PORTD;         // Input switches, output to PORTD
        PORTBbits.RB0=1;     // Latch data for conversion
        while(1);            // Halt
}
```