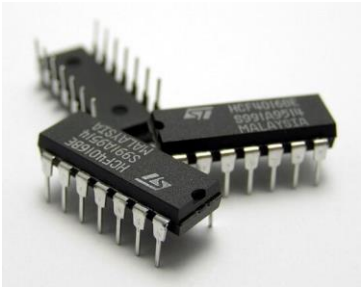
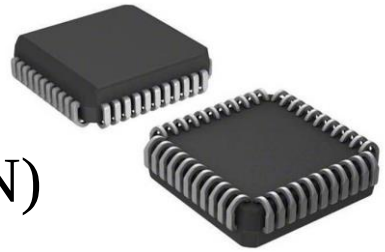
Chapter 8
PICF18 Programmed I/O Using Assembly & C

Chien-Chung Ho (何建忠)

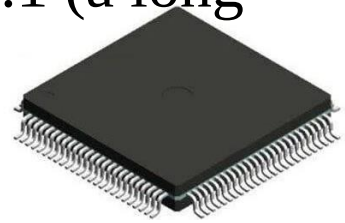
PIC18F Pins and Signals



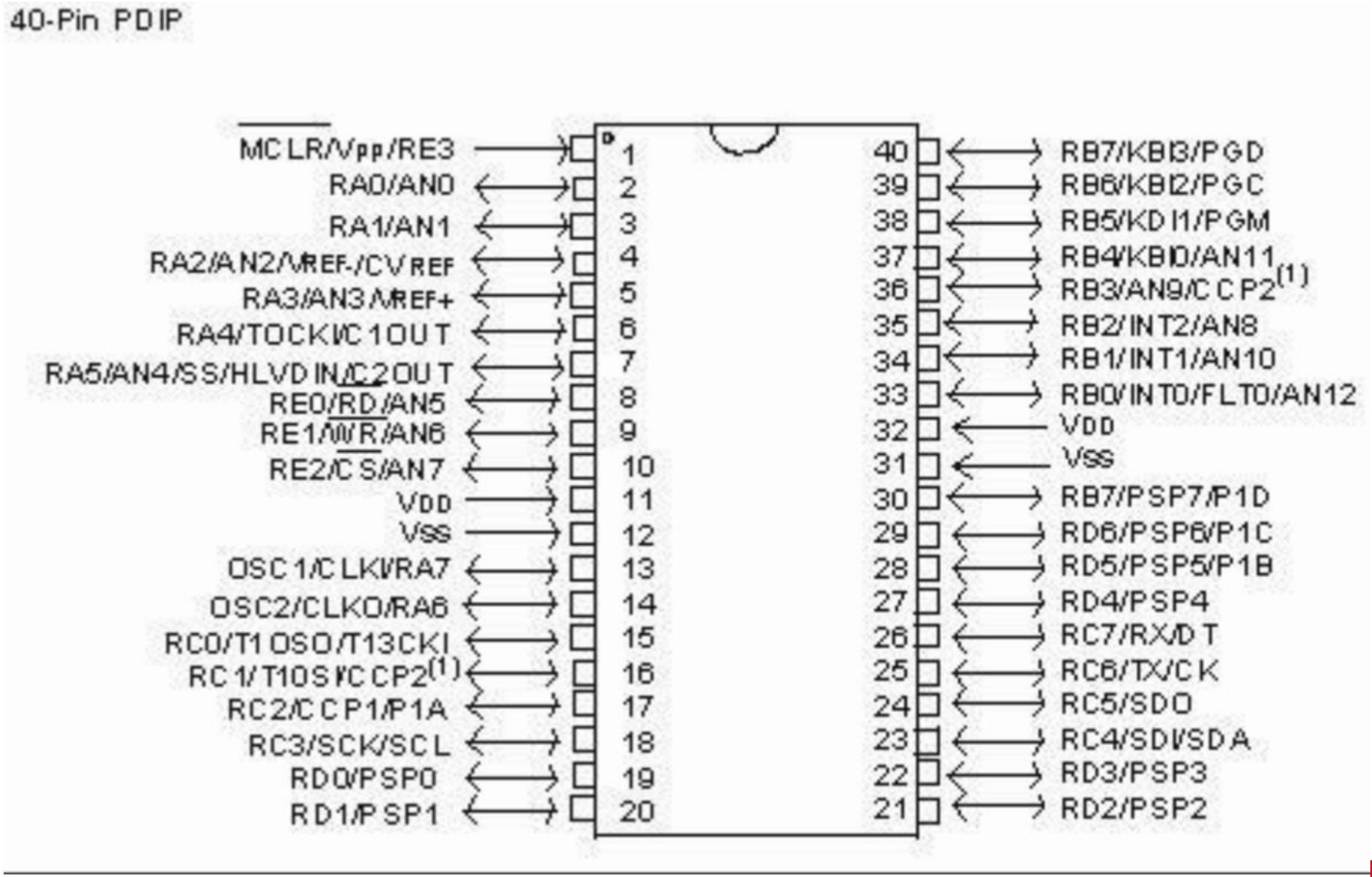
- The PIC18F4321 is contained in three types of packaging:
  - 40-pin Plastic Dual In-line Package (PDIP)
  - 44-pin Quad Flat No-lead plastic package (QFN)
  - 44-pin Thin plastic Quad Flat pack Package (TQFP)



- Figure 8.1 shows the PIC18F4321 pin diagram for a PDIP.
- A brief description of all pins and signals for the PIC18F4321 contained in the 40-pin PDIP is provided in Table 8.1 (a long table).



PIC18F Pins and Signals



Note 1: RB3 is the alternate pin for CCP2 multiplexing.

FIGURE 8.1 PIC18F4321 Pins and Signals

PIC18F Pins and Signals

- There are two VDD (Vcc) pins and two VSS (Ground) pins which are not shared (multiplexed) with other pins. The range of voltages for the VDD pins are from + 4.2V to +5.5V. However, the VDD pins are normally connected to +5V. The VSS pins are connected to ground. The maximum power dissipation for the PIC18F4321 is one watt.
- Note that multiple pins for power and ground are used in order to distribute the power to reduce noise problems at high frequencies.

PIC18F Pins and Signals

- All other 36 pins are multiplexed (shared) with other signals. There are 36 pins assigned to five I/O ports, namely Port A (8-bit, RA0-RA7), Port B (8-bit, RB0-RB7), Port C (8-bit, RC0-RC7), Port D (8-bit, RD0-RD7), and Port E (4-bit, RE0-RE3).
- These pins are multiplexed with other signals such as the clock/oscillator, reset, external interrupt, analog inputs, and CCP (Capture/Compare/Pulse Width Modulation).

Clock

- The PIC18F4321 clock can be generated internally or externally. The internal clock can be obtained by an on-chip oscillator provided by Microchip. The external clock, on the other hand, can be generated by connecting external components such as a crystal and RC oscillator circuit at the OSC1 and OSC2 pins.
- When the internal clock is used, an internal oscillator generates the internal clock. This will eliminate the need for an external oscillator circuit on the OSC1 and/or OSC2 pins.

Clock

- The main disadvantage of the internal oscillator is the ***lack of precision and frequency stability***. The baseline frequency depends on the values of the passive components used in building the internal oscillator circuit, and the tolerances of the values of the passive components are not particularly tight. Also, resistance and capacitance values are affected by ambient temperature. This means that changes in temperature lead to changes in frequency.

Clock

- Since the internal frequency is calibrated at the factory, many applications can tolerate the shortcomings of an internal oscillator. Contemporary microcontrollers such as the PIC18F can provide tolerance of -1.5% to +1.5%. This is accurate enough for serial I/O including RS-232 and USB. Furthermore, the PIC18F uses a hardware circuit called the “Phase-Locked Loop” (PLL) to multiply the internal frequency to a higher frequency.

Clock

- Upon power-on reset, the PIC18F4321 automatically operates at an internal clock frequency of 1-MHz (default). This means that with no crystal oscillator circuit connected to the OSC1 and OSC2 pins, the PIC18F4321 operates from an internal clock frequency of 1-MHz upon hardware reset.

Clock

- Two internal oscillator configurations are available. They are INTIO1 mode and INTIO2 mode. These modes can be selected, for example, using the MPLAB C18 compiler's CONFIG directive. In INTIO1 mode, the OSC2 pin outputs FOSC/4, while OSC1 functions as RA7 (bit 7 of Port A) for digital input and output.
- On the other hand, in INTIO2 mode, OSC1 functions as RA7 and OSC2 functions as RA6 both for digital input and output.

Clock

- There are three clock sources for the PIC18F4321. They are:
  - Primary oscillators, Secondary oscillators, Internal oscillators
- The primary oscillators require external circuits to be connected at the OSC1 and OSC2 pins. The external components such as a crystal and RC oscillator circuit can be connected to the OSC1 and OSC2 pins of the PIC18F4321. This will provide input to the primary oscillators to generate the clock for the PIC18F4321 externally.

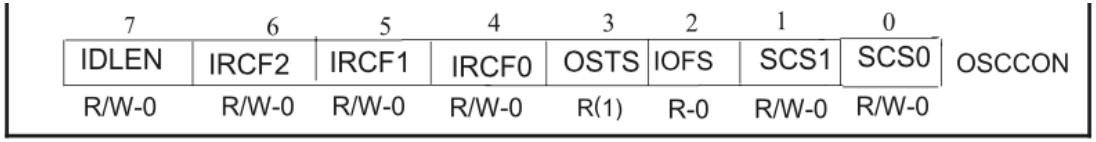
Clock

- The secondary oscillators are those external sources connected to the T1OSI (Timer1 Oscillator Input) and T1OSO (Timer1 Oscillator Output) pins of the PIC18F4321.
- The PIC18F4321 offers the Timer1 (one of the on-chip hardware timers) oscillator as a secondary oscillator.

Clock

- The internal oscillators provide a clock to the PIC18F4321 without connecting any components such as crystal or RC circuits to its clock pins.
- The Primary, Secondary, or Internal oscillator can be selected by programming an internal register called the OSCCON (Oscillator Control) register (Figure 8.2).

Clock



7	6	5	4	3	2	1	0	
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0	OSCCON
R/W-0	R/W-0	R/W-0	R/W-0	R(1)	R-0	R/W-0	R/W-0	

bit 7 **IDLEN:** Idle Enable bit
1 = Device enters an Idle mode when a SLEEP instruction is executed
0 = Device enters SLEEP mode when a SLEEP instruction is executed
bit 6-4 **IRCF2:IRCF0:** Internal Oscillator Frequency Select bits**(5)**
111 = 8 MHz (INTOSC drives clock directly)
110 = 4 MHz
101 = 2 MHz
100 = 1 MHz**(3)**
011 = 500 kHz
010 = 250 kHz
001 = 125 kHz
000 = 31 kHz (from either INTOSC/256 or INTRC directly)**(2)**
bit 3 **OSTS:** Oscillator Start-up Time-out Status bit**(1)**
1 = Oscillator Start-up Timer (OST) time-out has expired; primary oscillator is running
0 = Oscillator Start-up Timer (OST) time-out is running; primary oscillator is not ready
bit 2 **IOFS:** INTOSC Frequency Stable bit
1 = INTOSC frequency is stable
0 = INTOSC frequency is not stable
bit 1-0 **SCS1:SCS0:** System Clock Select bits**(4)**
1x = Internal oscillator block
01 = Secondary (Timer1) oscillator
00 = Primary oscillator
Note 1: Reset state depends on state of the IESO Configuration bit.
2: Source selected by the INTSRC bit (OSCTUNE<7>), see text.
3: Default output frequency of INTOSC on Reset.
4: Modifying the SCSI:SCSO bits will cause an immediate clock source switch.
5: Modifying the IRCF3:IRCF0 bits will cause an immediate clock frequency switch if the internal oscillator is providing the device clocks.
Legend:
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
-n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

FIGURE 8.2 OSCCON (Oscillator Control) Register

Clock

- The OSCCON (Oscillator Control) register controls several aspects of the device clock's operation, both in full power operation and in power-managed modes.
- Note that PIC18F power-managed mode consumes low power (nanowatts), and are suitable for low power applications including smoke detectors, hospital ID tags, and automobile ignition systems.

Clock

- The IDLEN bit controls whether the PIC18F4321 goes into Sleep mode or one of the Idle modes when the SLEEP instruction is executed. The IDLEN bit affects how the oscillator behaves in power managed modes.

Sleep Mode v.s. Idle Mode

Idle Mode

- **Characteristics:**
  - The CPU halts execution and enters an idle state.
  - Peripherals (e.g., timers, ADC) and the clock continue to operate.
  - The system can wake up quickly via an interrupt and resume operations almost immediately.
- **Applications:**
  - Ideal for scenarios where peripherals must remain active, such as:
  - **Temperature Monitoring System:** The ADC periodically measures temperature, while the CPU processes the results only when data is ready.
  - **UART Communication:** The CPU remains idle while waiting for external signals, but the UART must stay active.
- **Why Choose Idle Mode:**
  - Choose Idle Mode if the system requires peripherals to continue functioning and needs a quick response to events.

Sleep Mode

- **Characteristics:**
  - The CPU and peripherals completely stop operation, except for the Watchdog Timer (WDT) or external interrupts, which can wake the system.
  - This is the most power-efficient mode, but waking up requires reinitializing the peripherals.
- **Applications:**
  - Suitable for applications with long periods of inactivity, such as:
  - **Battery-Powered Devices:** For example, wireless sensor nodes that only wake up to transmit data upon timer expiration or an external trigger.
  - **Low-Power Displays:** Devices that shut down entirely during non-operational hours to conserve battery life.
- **Why Choose Sleep Mode:**
  - Choose Idle Mode if the system requires peripherals to continue functioning and needs a quick response to events.

Clock

- Once the internal oscillator is selected (using “config OSC = INTIO2” in assembly or “ #pragma config OSC = INTIO2” in C), the Internal Oscillator Frequency Select bits (IRCF2:IRCF0) in the OSCCON register select the frequency output of the internal oscillator to drive the PIC18F4321 clock. The three bits select one of eight different clock frequencies.

Clock

- The internal oscillator start-up timer is a delay built into the PIC18F4321 to allow the external oscillator to stabilize. However, the internal oscillator will run the PIC18F4321 if it is selected as the main oscillator. The OSTS and IOFS are read-only and are set or reset by the device. The PIC18F4321 has the option to startup running off the internal oscillator until an external oscillator circuit is stabilized. This allows for a faster startup of the microcontroller with external oscillators. The OSTS bit is used to alert the software when the clock source has switched to the external primary oscillator.

Clock

- The System Clock Select bits, SCS1:SCS0, select the clock source. The available clock sources are the primary clock, the secondary clock (Timer1 oscillator), and the internal oscillator. For example, SCS1SCS0 = 1x selects the internal clock.

Clock

- After power-on reset, the PIC18F4321 operates from an on-chip clock frequency of 1-MHz provided by Microchip, and no external crystal or clock circuit is required. This is called the default mode. In the default mode (after reset), the contents of the OSCCON register are 0x40 as follows:
- IDIEN = 0, IRCF2 - IRCF0 = 100 = 1MHz, OSTS = 0, IOFS = 0, SCS1:SCS0 = 00.

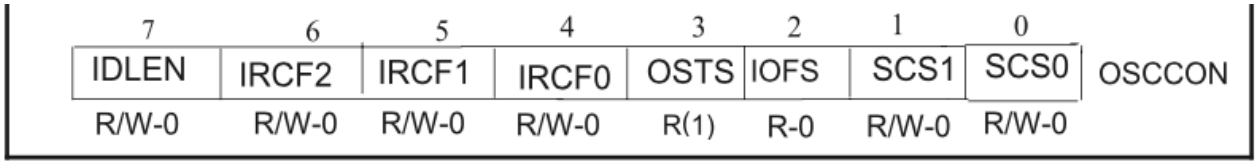
7	6	5	4	3	2	1	0	
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0	OSCCON
R/W-0	R/W-0	R/W-0	R/W-0	R(1)	R-0	R/W-0	R/W-0	

Clock

- However, the internal clock frequency can be changed within the range between 31 kHz and 8 MHz by writing appropriate data into the OSCCON register.
- For example, in order to change the internal frequency from 1-MHz (default mode) to 4-MHz, bits IRCF2 - IRCF0 in the OSCCON register must be 110.

bit 6-4 **IRCF2:IRCF0:** Internal Oscillator Frequency Select bits(5)
111 = 8 MHz (INTOSC drives clock directly)
110 = 4 MHz
101 = 2 MHz
100 = 1 MHz(3)
011 = 500 kHz
010 = 250 kHz
001 = 125 kHz
000 = 31 kHz (from either INTOSC/256 or INTRC directly)(2)

Clock



7	6	5	4	3	2	1	0	
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0	OSCCON
R/W-0	R/W-0	R/W-0	R/W-0	R(1)	R-0	R/W-0	R/W-0	

This can be accomplished by using the following PIC18F assembly code:

```
MOVLW 0x60; Set the internal clock to 4 MHz
MOVWF OSCCON
```

This is equivalent to the following C-code

```
OSCCON = 0x60;          // set the internal crystal to 4 MHz
```

Note that the value of SCS1:SCS0 = 10 or 11 (selecting the internal oscillator in the OSCCON register of Figure 8.2). This is because 'x' in SCS1:SCS0 = 1x (Figure 8.2) is don't care, and can be 0 or 1. Loading OSCCON register with 0x62 or 0x63 will also select the internal oscillator with 4 MHz. The statement config OSC = INTIO2 (for MPLAB PIC18F assembler) and the statement

```
# pragma config OSC = INTIO2
```

(for C18 compiler) will select the internal oscillator in the INTIO2 mode. These statements are sufficient enough to get the processor running using the internal oscillator regardless of the settings of the SCS1:SCS0 bits in the OSCCON register. The directives "config" and "#pragma config" will be covered in details later in this chapter.

Clock

- The PIC18F4321 can be operated in several different oscillator configurations. The user can program the Configuration bits FOSC3:FOSC0 in the CONFIG1H (Configuration Register) to select one of these configurations. Two of these configurations for the primary oscillator are provided below:

Clock

- 1. By connecting a crystal or ceramic oscillator at OSC1 and OSC2 pins, there are several choices for the crystal oscillator. One may consult the “PIC18F data sheet” for details.
- If a crystal oscillator is used, a parallel resonant crystal rather than a series resonant crystal must be used. This is because series resonant crystals do not oscillate upon power-up. Figure 8.3 shows the connection of a crystal oscillator (parallel resonant) to the OSC1 and OSC2 pins of the PIC18F4321.

Clock

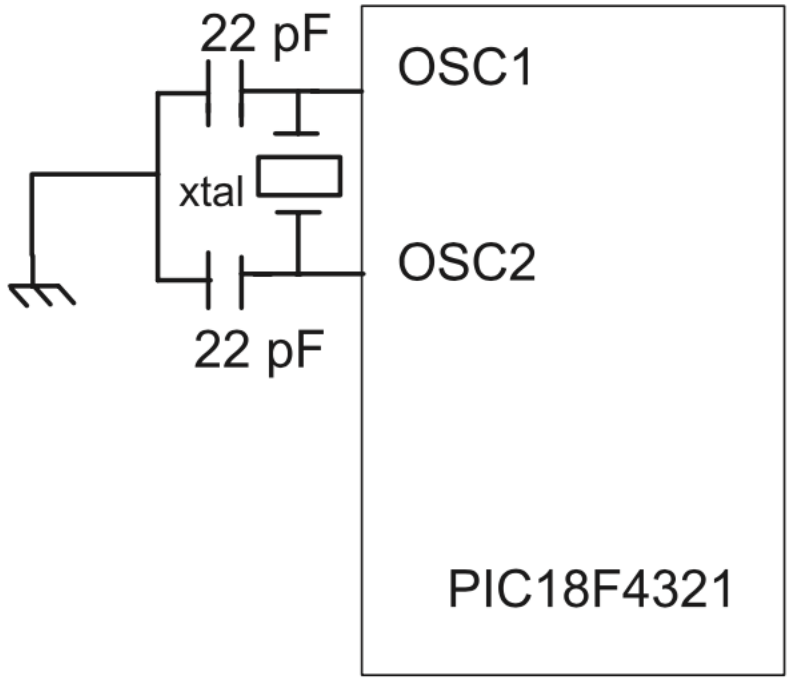


FIGURE 8.3 Typical Crystal Oscillator Circuit

Clock

- 2. By connecting an RC circuit at the OSC1 pin. The oscillator frequency divided by 4 is output at the OSC2 pin. An RC circuit connected to the OSC1 pin is used as the clock source. The RC-based clock is not that accurate compared to the crystal oscillator.
- Using this configuration, the clock input at the OSC1 pin can be used as the CPU clock while the clock output (Fosc/4) at the OSC2 pin can be used for devices requiring Fosc/4 as the input clock.

Clock

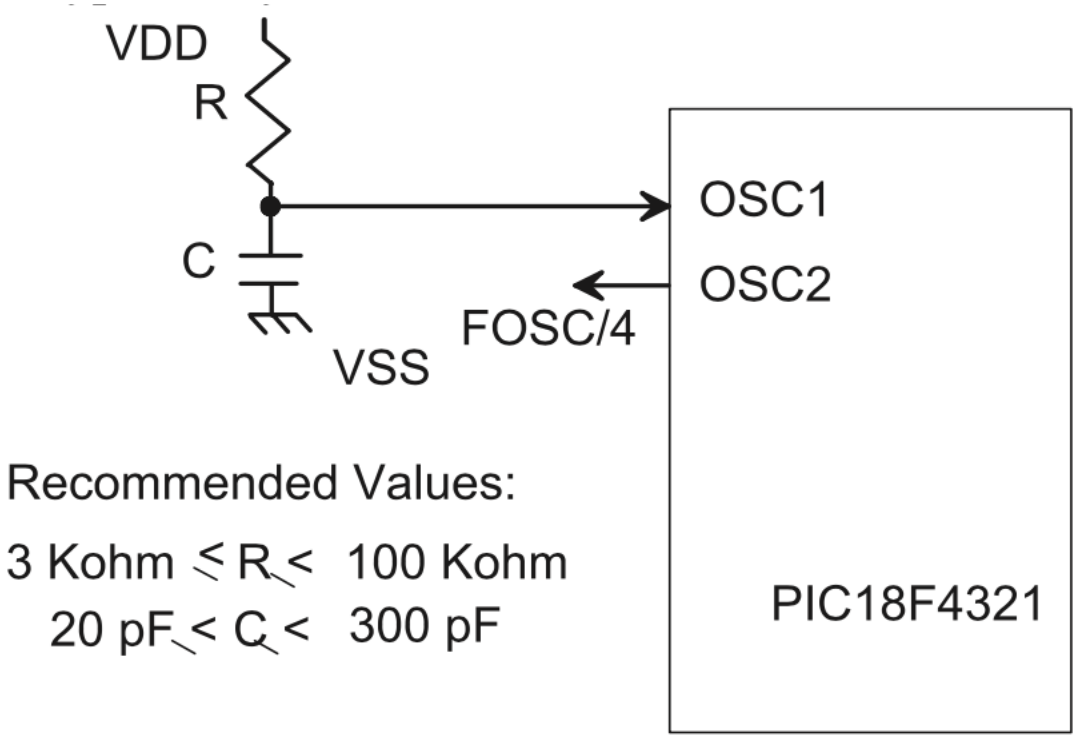


FIGURE 8.4 RC Oscillator

Clock

- In summary, the following suggestions should be followed:
- 1. Use the internal clock with a default value of 1-MHz.
- 2. Select the clock from the source INTIO1 or INTIO2 using the "config" directive for the MPLAB PIC18F assembler or the "#pragma config" directive for the C18 compiler.
- 3. If changing the default frequency is desired, then use bits 6-4 (IRCF2-IRCF0) in the OSCCON register to select the clock frequency within the allowable range.

PIC18F Reset

- Upon initiating a reset, the PIC18F loads ‘0’ into the program counter. Thus, the PIC18F reads the first instruction from the contents of address 0 in the program memory.
- Most registers are unaffected by a reset. However, WREG and STKPTR are cleared to zero. All TRISx (Data Direction Registers) registers are loaded with ones; that is, all I/O ports are configured as inputs.

PIC18F Reset

- The PIC18F4321 can be reset in several different ways. For simplicity, the two most commonly used RESET techniques are Power-on and Manual resets.

PIC18F Reset

- **Power-On Reset (POR)** Upon power-up, a power-on reset pulse is generated internally whenever VDD rises above a certain threshold. This allows the PIC18F chip to start in the initialized state when VDD is adequate for operation.
- When the VDD is connected to power (for example, +5VDC), the resistors in Figure 8.5 with the switch open will provide power-on reset.

PIC18F Reset

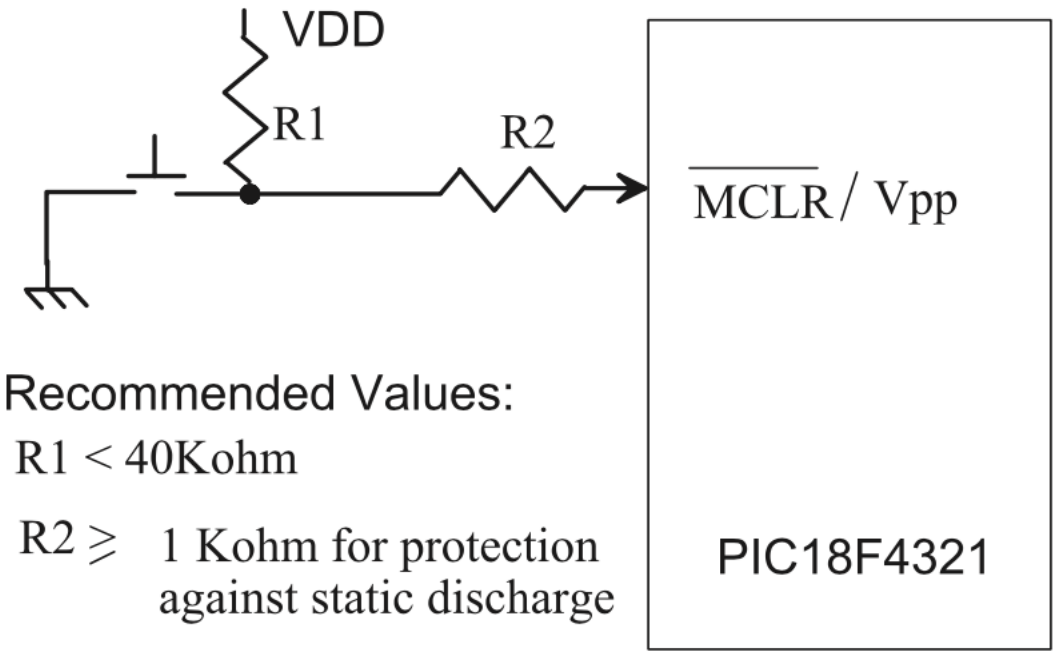
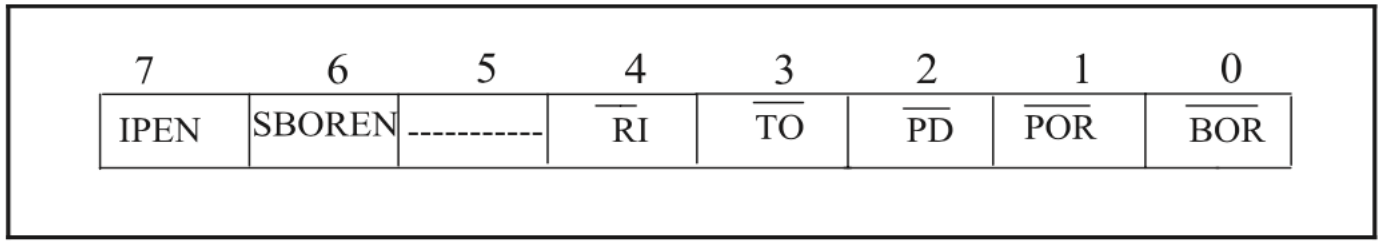


FIGURE 8.5 PIC18F Manual Reset Circuit

PIC18F Reset

- Power-on Reset events are captured by the bit (bit 1 of RCON, Figure 8.6). The state of the bit is cleared to ‘0’ whenever a POR occurs; bit = 1 indicates that a POR has not occurred.



Bit 7 = IPEN (Interrupt Pending, 1 = enable, 0 = disable; to be discussed later) **Bit 6** = SBOREN (BOR software enable bit, 1 = enable, 0 = disable) **Bit 5** = Unimplemented (read as ‘0’) **Bit 4** = $\overline{\text{RI}}$ (RESET instruction bit, 1 = RESET instruction executed, 0 = RESET instruction not executed) **Bit 3** = $\overline{\text{TO}}$ (Watchdog Timeout bit, 1 = upon power-up or execution of CLRWDT or SLEEP instruction, 0 = a watchdog timer timeout occurred) **Bit 2** = $\overline{\text{PD}}$ (Power- Down detection bit, 1 = upon power-up or execution of CLRWDT instruction, 0 = execution of SLEEP instruction. **Bit 1** = A $\overline{\text{POR}}$ (Power-on Reset status bit, 1 = A Power-on reset has not occurred, 0 = A Power-on reset has occurred) **Bit 0** = $\overline{\text{BOR}}$ (Brown-out Reset status bit, 1 = A Brown-out reset has not occurred, 0 = A Brown-out reset has occurred)

FIGURE 8. 6 RCON (RESET CONTROL) Register

PIC18F Reset

- **Manual Reset** The $\overline{\text{MCLR}}$ pin is normally HIGH. Upon activating the push button, the $\overline{\text{MCLR}}$ pin is driven from HIGH to LOW. The internal on-chip circuitry connected to the $\overline{\text{MCLR}}$ pin ensures that the pin is LOW for at least 2 μsec (minimum requirement for reset).

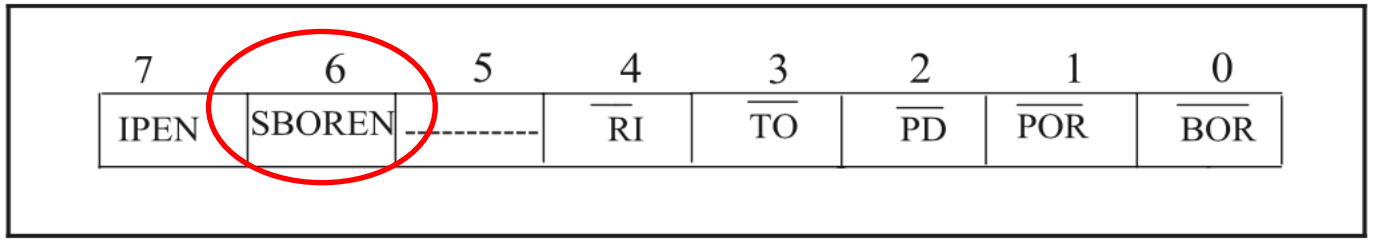
PIC18F Reset

- Some of the other resets include:
  - Brown-out Reset (BOR), Watchdog Timer (WDT) reset (during program execution), RESET Instruction.
- Note that other reset events are tracked through the RCON (Reset Control) register (Figure 8.6). The lower five bits of the register indicate that a specific reset event has occurred.

7	6	5	4	3	2	1	0
IPEN	SBOREN	----------	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

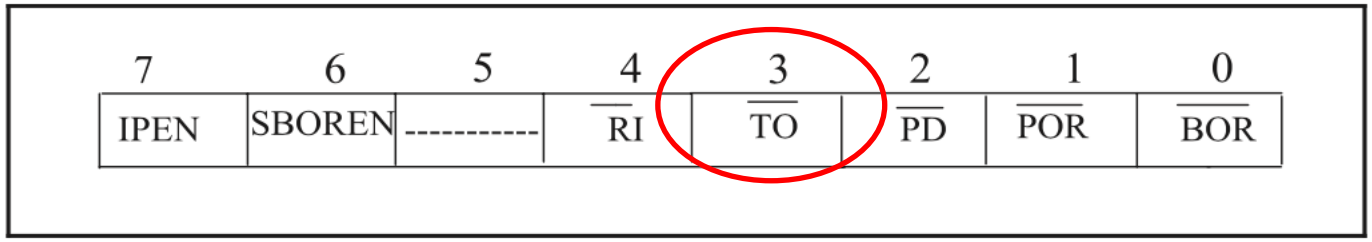
PIC18F Reset

- **Brown-out Reset**. The PIC18F implements a BOR circuit that resets the PIC18F when the power supply voltage drops below a preprogrammed level.
- The idea behind BOR is that the processor will hold in reset when BOR occurs rather than trying to keep running unreliably. Thus, the PIC18F will resume program execution properly when the power goes above brownout threshold level.



PIC18F Reset

- **Watchdog Timer (WDT)** The PIC18F WDT is driven by an internal clock source. When the WDT is enabled via software using WDTCON (Watchdog Timer Control register), the internal clock source is also enabled. If the timer associated with the WDT is enabled and then times out after a specific amount of time during program execution, the PIC18F resets itself automatically and clears the bit in the RCON register.



PIC18F Reset

- **RESET Instruction** This is a software RESET. Upon execution of the RESET instruction, the PIC18F resets all registers and flags that are affected by a power-on Reset, and the bit in the RCON register is cleared to 0.

“pragma config” and “config” directives

- The C18 directive “#pragma config” = MPLAB assembler “config” directive. This directive tells the MPLAB PIC18F assembler that the assembler needs to do something specific to the hardware. Some of these functions include selecting an appropriate oscillator and turning ON or OFF certain features such as WDT and BOR.

“pragma config” and “config” directives

```
include  <P18F4321.INC>
config  OSC  =  INTIO2          ; Select internal oscillator
config  WDT =  OFF              ; Watchdog Timer OFF
config  LVP   =  OFF            ; Low Voltage Programming OFF
config  BOR  =   OFF            ; Brown Out Reset OFF
```

Note that when the $\overline{\text{MCLR}}$ pin is activated for a hardware reset, the PGM is set to HIGH, the LVP (Low Voltage Programming) is enabled, and the PIC18F will enter the “Low Voltage Programming” mode. Since the PGM pin and the pin RB5 (bit 5 of Port B) are multiplexed, disabling the LVP mode frees this pin to be used as RB5 for digital I/O.

“pragma config” and “config” directives

```
#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
```

The program must follow the above statements. This is because the clock source must be available before the program can run.

For simplicity, the PIC18F4321 internal clock in the default mode will be used in this book to illustrate the various hardware aspects of the PIC18F4321.

PIC18F4321 Programmed I/O

- A microcontroller communicates with an external device using programmed I/O via one or more registers called I/O ports. Each bit in the port can be configured individually as either input or output. Each port can be configured as an input or output port by the Data Direction Register (DDR).

PIC18F4321 Programmed I/O

- The PIC18F microcontroller makes an I/O port bit an input by writing a ‘1’ in the corresponding bit in DDR. On the other hand, writing a ‘0’ in a particular bit in DDR will configure the corresponding bit in the port as an output.

PIC18F4321 Programmed I/O

- For example, if an 8-bit DDR in the PIC18F contains 0xCB, the corresponding port is defined as shown in Figure 8.8. In this example, because 0xCB (1100 1011) is stored in the data-direction register, bits 0, 1, 3, 6, and 7 of the port are set up as inputs, and bits 2, 4, and 5 of the port are defined as outputs. The microcontroller can then send output to external devices such as LEDs, connected to bits 2, 4, and 5, through a proper interface.

PIC18F4321 Programmed I/O

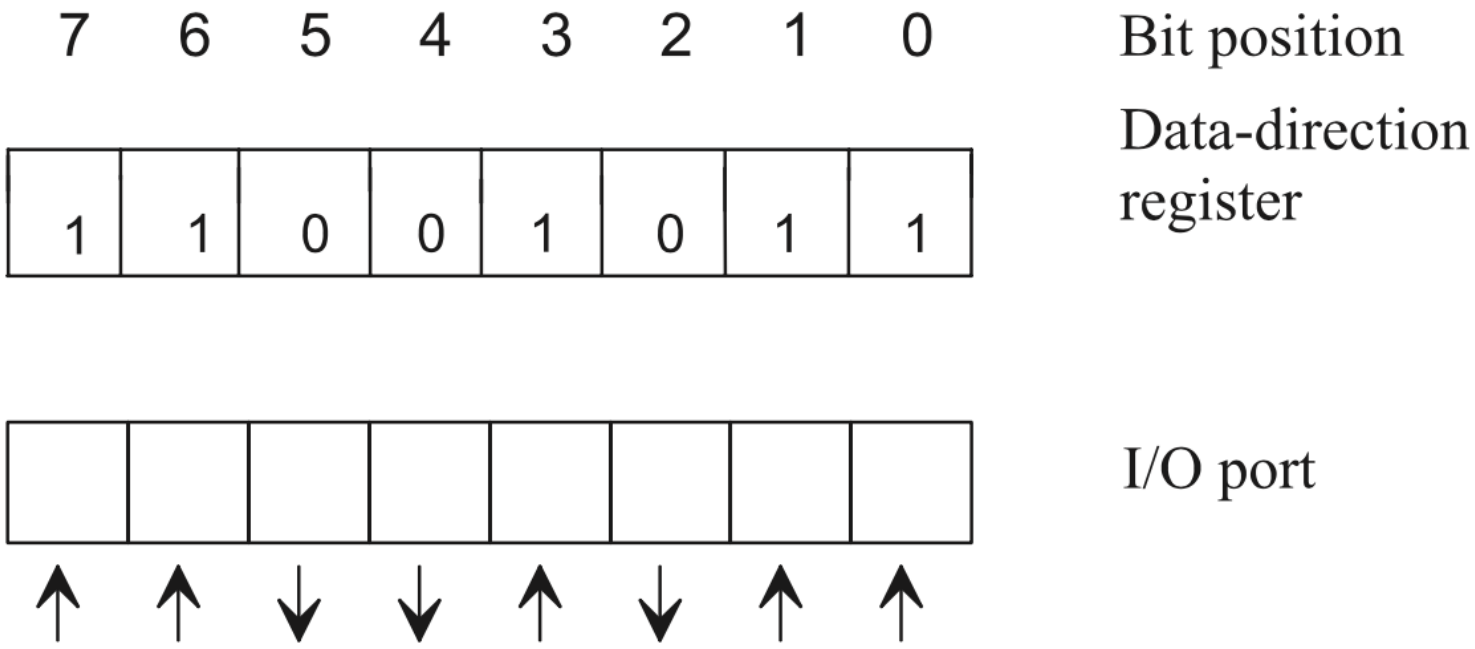


FIGURE 8.8 I/O port with the corresponding data-direction register.

PIC18F4321 Programmed I/O

- I/O ports are addressed using either standard I/O or memory-mapped I/O techniques. Using *Standard I/O* or sometimes called *port I/O*, the CPU outputs an internal signal such as the M/$\overline{IO}$ for memory and I/O units on the microcontroller chip. The CPU outputs a HIGH on M/$\overline{IO}$ to indicate to memory and the I/O that a memory operation is taking place. A LOW output from the CPU to M/$\overline{IO}$ indicates an I/O operation.

PIC18F4321 Programmed I/O

- Execution of an IN or OUT instruction makes the M/$\overline{IO}$ LOW, whereas memory-oriented instructions, such as MOVE, drive the M/$\overline{IO}$ to HIGH.
- In standard I/O, the CPU uses the M/$\overline{IO}$ output signal to distinguish between I/O and memory. Intel microcontrollers such as the 8051 use standard I/O.

PIC18F4321 Programmed I/O

- In *memory-mapped I/O*, the CPU ~~uses an unused address pin to distinguish between memory and I/O~~. The CPU uses a portion of the memory addresses to represent I/O ports. The I/O ports are mapped as part of the microcontroller's main memory addresses which may not exist physically, but are used by the microcontroller's memory-oriented instructions, such as MOVE, to generate the necessary control signals to perform I/O. The PIC18F uses memory-mapped I/O.
  - **There are no "unused address pins"** involved to distinguish between memory and I/O.
  - Instead, certain **ranges of memory addresses** are reserved for I/O devices, while others are used for standard memory.

PIC18F4321 Programmed I/O

- The PIC18F4321 contains five ports namely Port A (8-bit), Port B (8-bit), Port C (8-bit), Port D (8-bit) and Port E (4-bit). **Most of the pins of the PIC18F4321 I/O ports are multiplexed with peripheral I/O pins on the chip.** Note that peripheral I/O pins include signals to support on-chip peripheral I/O such as ADC and hardware timers. In general, **when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.**

PIC18F4321 Programmed I/O

- For simple I/O operation, three latches are associated with each I/O port bit. They are:
  - 1. TRIS (Tristate) latch
  - 2. Input latch
  - 3. Output latch
- Writing a ‘1’ in the TRIS latch will configure the corresponding bit in the port as an input. Writing a ‘0’ at a particular bit in the TRIS latch will configure the corresponding bit as an output.

PIC18F4321 Programmed I/O

- Table 8.2 shows a list of the PIC18F4321 I/O ports along with the associated TRISx registers where ‘x’ is the port name (A, B, C, D, E). Note that these ports and data direction registers are mapped as data memory addresses by Microchip.
- PORT A through PORT E and TRISA through TRISE can be directly used in a program rather than their hex values of **mapped addresses.**

PIC18F4321 Programmed I/O

TABLE 8.2 PIC18F4321 I/O PORTS, TRISx REGISTERS, ALONG WITH ADDRESSES
(Upon RESET, all ports are configured as inputs)

Port Name	Size	Mapped SFR address	Comment
PORTA	8-bit	0xF80	PORTA
TRISA	8-bit	0xF92	Data Direction Register for PORTA
PORTB	8-bit	0xF81	PORTB
TRISB	8-bit	0xF93	Data Direction Register for PORTB
PORTC	8-bit	0xF82	PORTC
TRISC	8-bit	0xF94	Data Direction Register for PORTC
PORTD	8-bit	0xF83	PORTD
TRISD	8-bit	0xF95	Data Direction Register for PORTD
PORTE	3-bit	0xF84	PORTE
TRISE	3-bit	0xF96	Data Direction Register for PORTE

PIC18F4321 Programmed I/O

- In the PIC18F, each bit in a port is assigned with a generic name Rxn where ‘x’ is A for Port A, B for Port B, C for Port C, D for Port D, and E for Port E, and ‘n’ is the bit number. This means that RA0 will indicate bit 0 of Port A. Similarly, in the PIC18F, each bit in a data direction register such as TRISx is assigned with a generic name TRISxn where ‘x’ is A for TRISA, …, and ‘n’ is the bit number. As an example, TRISA0 will indicate bit 0 of the TRISA data direction register.

I/O instructions in PIC18F assembly

- In PIC18F, typical memory-oriented instructions such as MOVWF, MOVF, MOVFF can be used for inputting from or outputting to ports.
- As an example, consider the PIC18F instruction MOVF PORTD, W which will input the contents of PORTD into WREG. The MOVWF PORTC instruction will output the contents of WREG into PORTC.

I/O instructions in PIC18F assembly

- Data can also be outputted from one port to another. For example, MOVFF PORTC, PORTD.
- The instructions such as BSF and BCF can be used to output a ‘0’ or a ‘1’ to the specified bit. For example, the instruction BSF PORTD, RD6 will set bit 6 of PORTD to one; in other words, the PIC18F will output a ‘1’ to bit 6 of PORTD. The instruction BCF PORTC, RC3 will clear bit 3 of PORTC to zero; in other words, the PIC18F4321 will output a ‘0’ to bit 3 of PORTC.

Configuring PIC18F4321 I/O ports

- As mentioned before, writing a ‘1’ at a particular bit position in the TRISx register will make the corresponding bit in the associated port as an input. On the other hand, writing a ‘0’ at a particular bit position in the TRISx register will make the corresponding bit in the associated port as an output.
- Upon reset all TRISx registers are automatically loaded with 1’s, and hence, all ports will be configured as inputs.

Configuring PIC18F4321 I/O ports

Next, in order to illustrate how PIC18F4321 ports are configured using the associated TRISx registers, consider the following PIC18F instruction sequence:

```
MOVLW    0x34    ; Move 0x34 into WREG
MOVWF    TRISD   ; Configure PORT D
```

- In the above instruction sequence, MOVLW loads WREG with 34 (hex), and then moves this data into TRISD (8-bit data direction register for PORTD) containing 34 (hex) (the corresponding port is defined as shown in Figure 8.10).

Configuring PIC18F4321 I/O ports

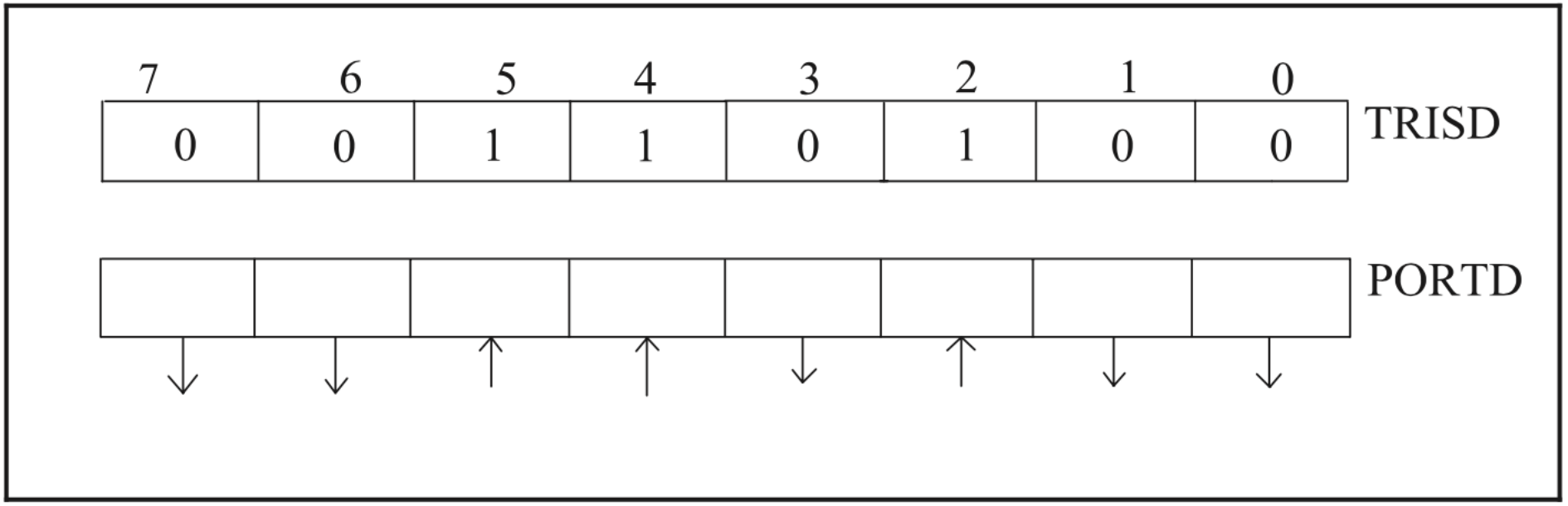


FIGURE 8.10 PORT D along with TRISD

Configuring PIC18F4321 I/O ports

- In this example, bits 0, 1, 3, 6, and 7 of the port are set up as outputs, and bits 2, 4, and 5 of the port are defined as inputs. The microcontroller can then send output to external devices, such as LEDs, connected to bits 0, 1, 3, 6, and 7 through a proper interface. Similarly, the PIC18F4321 can input the status of external devices through bits 2, 4 and 5.

Configuring PIC18F4321 I/O ports

The PIC18F instructions such as SETF and CLRF can be used to configure I/O ports. For example, to configure all bits in PORTC as inputs, and PORTD as outputs, SETF or CLRF instructions can be used as follows:

```
SETF    TRISC   ; Set all bits in TRISC to 1's and configure
                ; configure Port C as an input port.
CLRF    TRISD   ; Clear all bits in TRISD to 0's and configure
                ; configure PORTD as an output port
```

Configuring PIC18F4321 I/O ports

- A specific bit in a port can be configured as an input or as an output using PIC18F bit-oriented instructions such as BSF and BCF. For example, the instruction “BSF TRISD, 7” or “BSF TRISD, TRISD7” will make bit 7 of PORTD an input bit. On the other hand, “BCF TRISC, 1” or “BCF TRISC, TRISC1” will make bit 1 of Port C an output bit.

Configuring PIC18F4321 I/O ports

- Note that configuring PORTA, PORTB, and PORTE is different than configuring PORTC and PORTD. This is because certain bits of PORTA, PORTB, and PORTE are multiplexed with analog inputs. For example, bits 0-3 and bit 5 of PORTA are multiplexed with analog inputs AN0-AN4, bits 0-4 of PORTB are multiplexed with analog inputs AN8-AN12, and bits 0-2 of PORTE are multiplexed with analog inputs AN5-AN7 (Figure 8.1, PIC18F4321 pin diagram).

Configuring PIC18F4321 I/O ports

- When a port bit is multiplexed with an analog input, bits 0-3 of a register called ADCON1 (A/D Control Register 1) must be used to configure the port bit as an input. The other bits in ADCON1 are associated with the A/D converter.
- Figure 8.11 shows the ADCON1 register along with the associated bits for digital I/O.

Configuring PIC18F4321 I/O ports

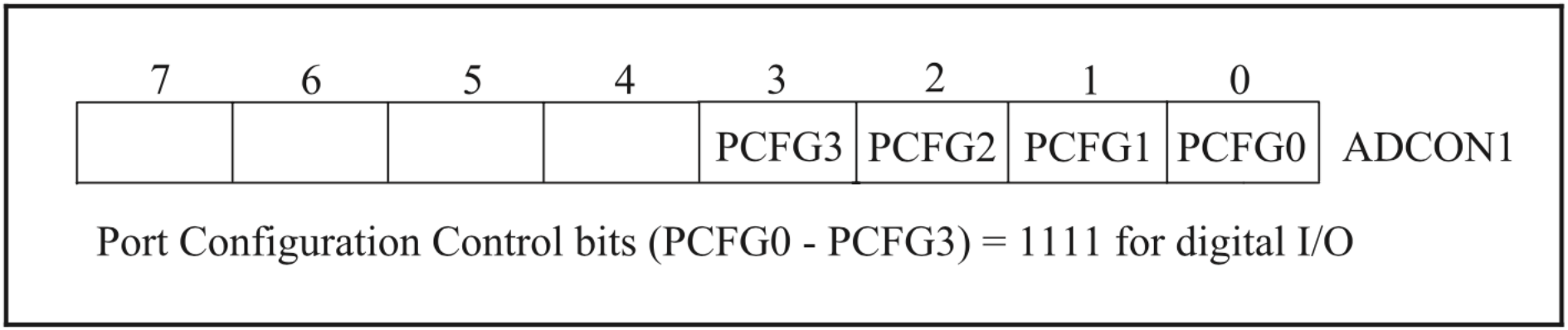


FIGURE 8.11 ADCON1 register for digital I/O

Configuring PIC18F4321 I/O ports

- When bits 0 through 3 of the ADCON1 register are loaded with 1111, the analog inputs (AN0-AN12) multiplexed with the associated bits of PORTA, PORTB, and PORTE (Bits 0, 1, 2) are configured as digital inputs.

Configuring PIC18F4321 I/O ports

For example, the following instruction sequence will configure all 13 port bits multiplexed with AN0 - AN12 as input bits, and TRISx registers are not required:

```
MOVLW     0x0F          ; Move 0xF into WREG
MOVWF     ADCON1        ; Move WREG into ADCON1
```

Next, in order to configure bit 1 of PORTA, all bits of PORTB, and bit 0 of PORTE as outputs, the following instruction sequence can be used:

```
MOVLW     0x0F          ; Move  0xF into WREG
MOVWF     ADCON1        ; Move WREG into ADCON1
BCF       TRISA, TRISA1 ; Configure bit 1 of PORTA as output bit
CLRF      TRISB         ; Configure PORTB as  an output port
BCF       TRISE, TRISE0 ; Configure bit 0 of PORTE as output bit
```

Configuring PIC18F4321 I/O ports using C

The following C language statements will make all bits of Port C and Port D as inputs and outputs:

```
TRISC = 0xFF;          // Configure PORT C as an input port
TRISD = 0 ;            // Configure PORT D as an output port
```

- The MPLAB C18 compiler provides a built-in feature for configuring a port bit. This allows the programmer to address a single bit in a port without changing the other bits in the port. This means that the PIC18F ports are bit addressable.

Configuring PIC18F4321 I/O ports using C

- For example, the C-statement “PORTCbits.RC0;” can be used to access bit 0 of PORTC. Each bit in the TRISx registers can be accessed in the same manner. As an example, the C-statement “TRISCbits.TRISC5 =0;” will configure bit 5 of PORTC as an output bit; a ‘1’ can then be outputted to bit 5 of PORTC using the following C-statement “PORTCbits.RC5 = 1;”.

Configuring PIC18F4321 I/O ports using C

- Furthermore, the C18 directive “#define” can be used to declare a bit in a port. Note that “#define” is a preprocessor directive in C. It is used to define the preprocessor macros for the texts. For example, the name ‘portbit’ can be declared as bit 2 of PORTC as follows:
  - # define portbit PORTCbits.RC2 // Declare a bit (bit 2) of PORTC

Configuring PIC18F4321 I/O ports using C

Next, bit 2 of PORTC can be configured as an output bit with the name 'portbit' using the following C-statements:

```
# define portbit PORTCbits.RC2     // Declare a bit (bit 2) of PORTC
TRISCbits.TRISC2 = 0 ;             // Configure bit 2 of PORTC as an output
```

Now, a '1' can be output to bit 2 of PORT C using the following statement:
portbit = 1;

Similarly, the statement "portbit = 0;" will output a '0' to bit 2 of PORTC
Next, bit 3 of PORTD can be configured as an input by writing a '1' at bit 3 of TRISD as follows:

```
# define switch PORTDbits.RD3      // Declare a bit (bit 3) of PORTD
TRISDbits. TRISD3 = 1;             // Configure bit 3 of PORTD as an input
```

Now, the name 'switch' (bit 3 of PORTD) can be used as an input bit in a C-program.

Note that the names 'portbit' and 'switch' in the above are chosen arbitrarily.

Configuring PIC18F4321 I/O ports using C

- For configuring PORTA, PORTB, and PORTE as outputs, register ADCON1 must be loaded with 0x0F to make PORTA, PORTB, and PORTE digital I/O. Registers TRISA, TRISB, and TRISE can then be loaded with 0x00 to configure these ports as outputs.

Configuring PIC18F4321 I/O ports using C

This can be accomplished by the following C-statements:

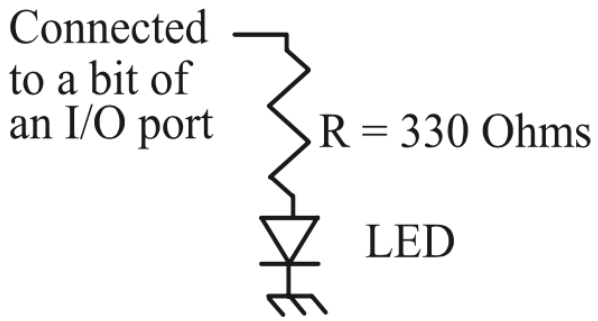
```
ADCON1 = 0x0F; // Configure ports A, B, and E as digital I/O
TRISA = 0;     // PORTA is output port
TRISB = 0;     // PORTB is output port
TRISE = 0;     // PORTE (bits 0-2) is output port
```

Also, to configure bit0 of PORTA as output, bit1 of PORTB as output and bit1 of PORTE as output, the following C-statements can be used:

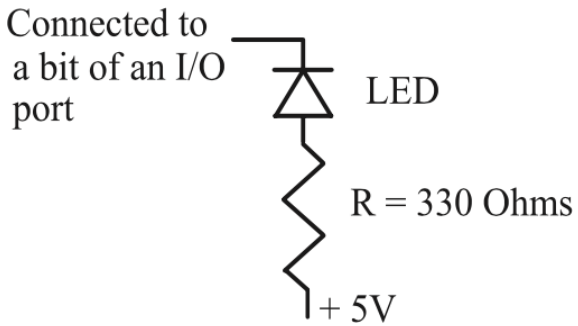
```
ADCON1 = 0x0F;          // Configure ports A, B, and E as digital I/O
TRISAbits.TRISA0 = 0;   // Bit0 of PORTA is output
TRISBbits.TRISB1 = 0;   // Bit1 of PORTB is output
TRISEbits.TRISE1 = 0;   // Bit1 of PORTE is output
```

Interfacing LED's (Light Emitting Diodes) and Seven-segment displays

- The PIC18F sources and sinks adequate currents so that LEDs and seven-segment displays can be interfaced to the PIC18F without buffers (current amplifiers) such as 74HC244. An LED can be connected in two ways. Figures 8.12(a) and (b) show these configurations.



(a) Connecting an LED (cathode groumded) to an I/O port bit



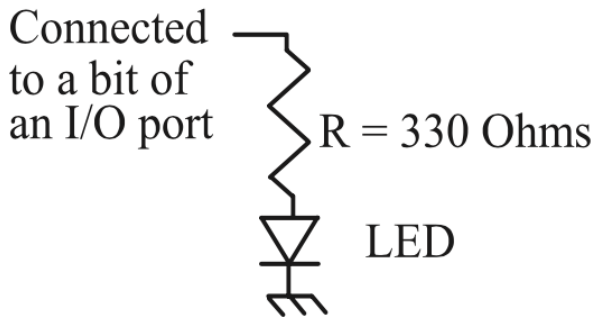
(b) Connecting an LED (anode tied to 5V) to an I/O port bit

FIGURE 8.12 Interfacing LED to PIC18F

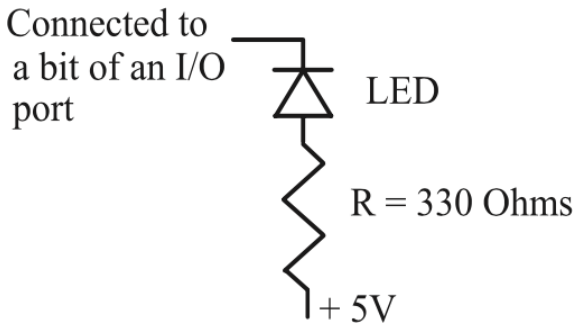
Interfacing LED's (Light Emitting Diodes) and Seven-segment displays

In Figure 8.12(a), the PIC18F will output a HIGH to turn the LED ON; the PIC18F will output a 'LOW' to turn it OFF. In Figure 8.12(b), the PIC18F will output a LOW to turn the LED ON; the PIC18F will output a 'HIGH' to turn it OFF. Also, when an LED is turned on, a typical current of 10 mA flows through the LED with a voltage drop of 1.7V. Hence,

$$R = \frac{5 - 1.7}{10\ \text{mA}} = \frac{5 - 1.7}{10\ \text{mA}} = 330$$



(a) Connecting an LED (cathode groumded) to an I/O port bit



(b) Connecting an LED (anode tied to 5V) to an I/O port bit

FIGURE 8.12 Interfacing LED to PIC18F

Interfacing LED's (Light Emitting Diodes) and Seven-segment displays

- A seven-segment display can be used to display decimal numbers from 0 to 9. The name "seven segment" is based on the fact that there are seven LEDs — one in each segment of the display. Figure 8.13 shows a typical seven-segment display.

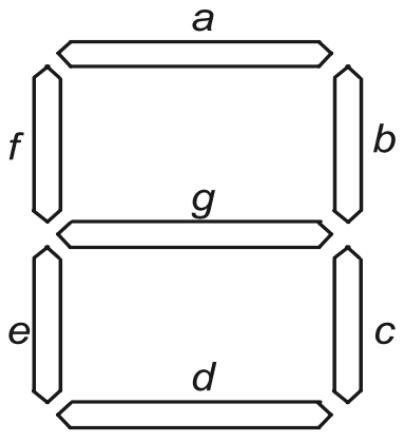


FIGURE 8.13 A seven-segment display

Interfacing LED's (Light Emitting Diodes) and Seven-segment displays

- In Figure 8.13, each segment contains an LED. All decimal numbers from 0 to 9 can be displayed by turning the appropriate segment "ON" or "OFF". There are two types of seven segment displays. These are common cathode (共陰極) and common anode (共陽極). In a common cathode arrangement, the microcontroller can be programmed to send a HIGH to light a segment and a LOW to turn it off. In a common anode configuration, on the other hand, the microcontroller can send a LOW to light a segment and a HIGH to turn it off. In both configurations, R = 330 ohms can be used.

Interfacing LED's (Light Emitting Diodes) and Seven-segment displays

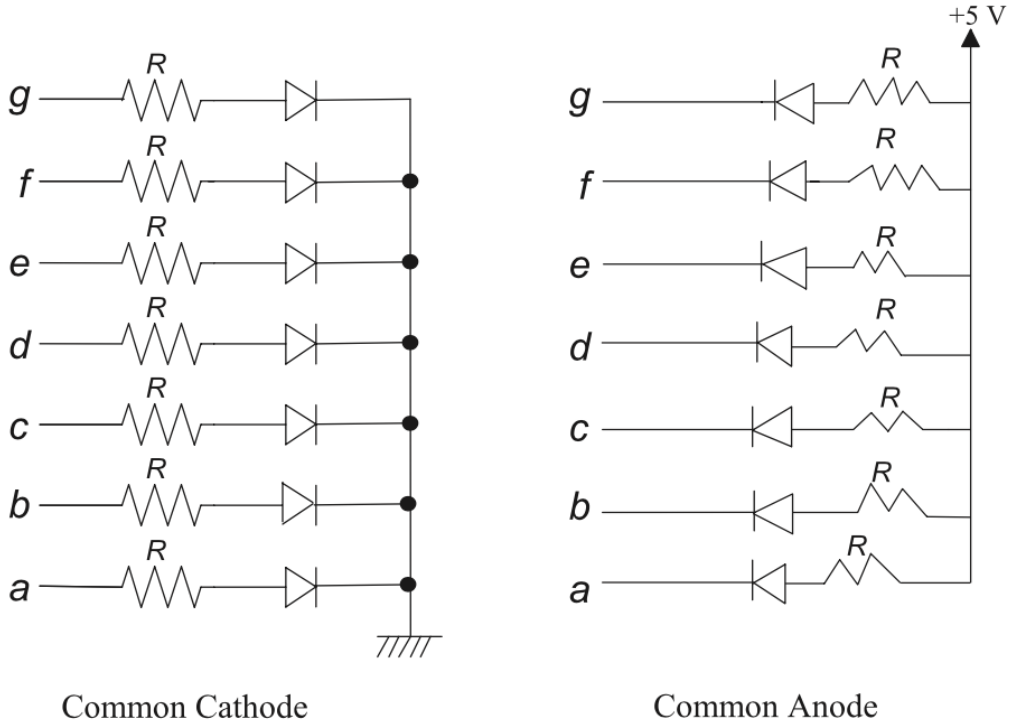


FIGURE 8.14 Seven-segment display configurations

Interfacing LED's (Light Emitting Diodes) and Seven-segment displays

- Figure 8.15 shows a typical interface between the PIC18F4321 and a common cathode seven-segment display via PORT C. Each bit of PORT C is connected to a segment of the seven-segment display via a 330 ohm resistor. A common anode seven-segment display can similarly be interfaced to the PIC18F4321.

Interfacing LED's (Light Emitting Diodes) and Seven-segment displays

```
INCLUDE     <P18F4321.INC>
ORG         0x100
BSF         TRISC, TRISC0  ; Configure bit 0 of Port C as an input
```

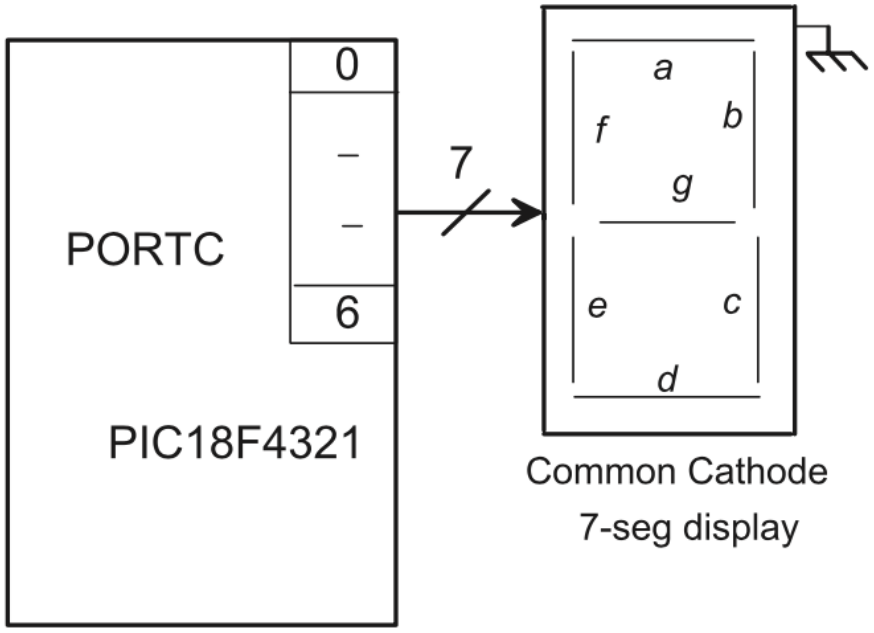


FIGURE 8.15 PIC18F4321 interface to a common cathode seven-segment display via PORT C

Programmed I/O examples

- For the programs to work properly, the following “config” statements should be included in all the programming examples 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
```

Programmed I/O examples

- The file p18f4321.inc is a PIC18F4321-specific file, and primarily contains definitions for the variables used to access the Special Function Registers (SFRs) of the PIC18F4321. For example, PORTA of the PIC18F4321 is mapped as a Special Function Register (SFR) with data memory address 0xF80. The “p18f4321.h” file contains this information. This means that the user can use PORTA in the C-program. The compiler will insert the address 0xF80 during compilation whenever it encounters PORTA in the program.

Programmed I/O examples

- Example 8.1 Write a PIC18F assembly language program at 0x100 to drive an LED connected to bit 0 of Port D based on a switch input to bit 0 of Port C as shown in Figure 8.16. If the switch is opened, turn the LED OFF; turn the LED ON if the switch is closed.

Programmed I/O examples

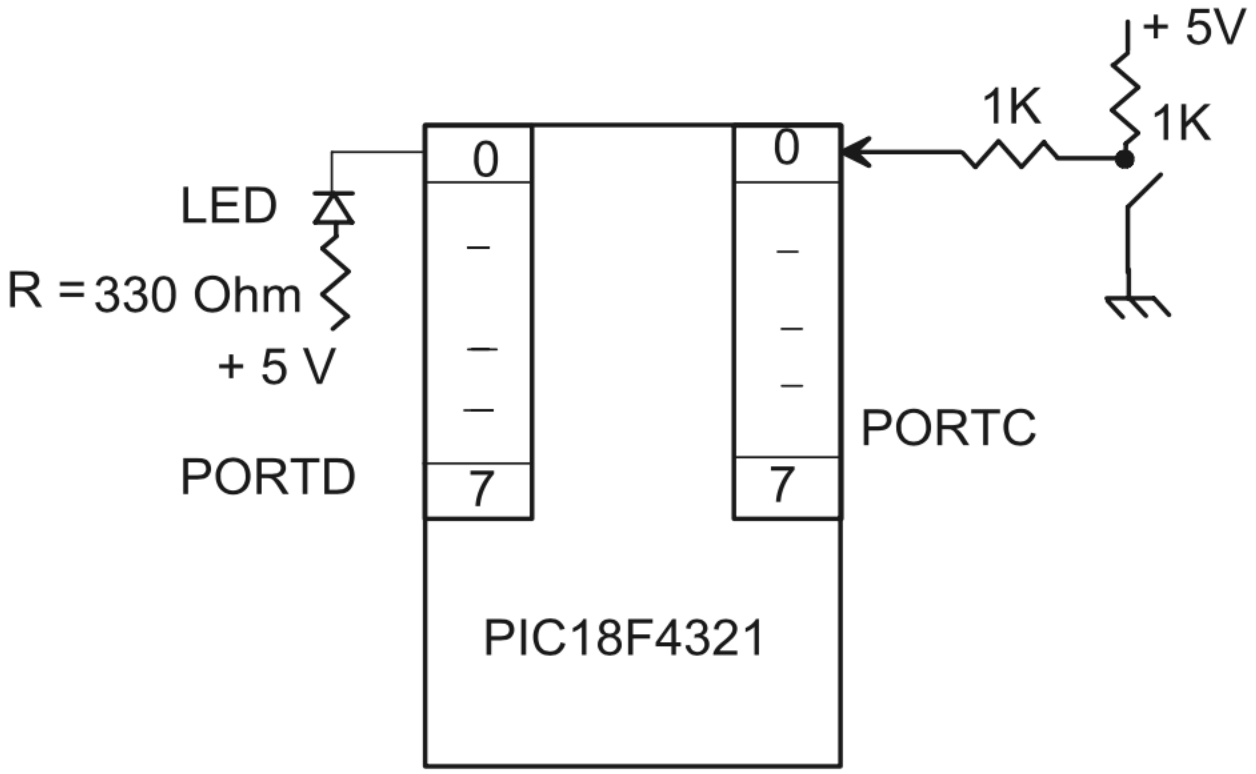


FIGURE 8.16 Figure for Example 8.1

Programmed I/O examples

From Figure 8.16, since the cathode of the LED is connected to bit 0 of Port C, a '0' output from the PIC18F4321 will turn the LED ON, and a '1' will turn it OFF. The PIC18F sinks (for LOW output) adequate current to turn an LED OFF or ON without any buffer such as 74HC244; only a current limiting resistor R = 330 ohm is required.
The 0PIC18F assembly language program is provided below:

```
        INCLUDE     <P18F4321.INC>
        ORG         0x100
        BSF         TRISC, TRISC0  ; Configure bit 0 of Port C as an input
        BCF         TRISD, TRISD0  ; Configure bit 0 of Port D as an output
START   MOVFF       PORTC, PORTD ; Output switch input to LED
        BRA         START          ; Repeat
        END
```

Programmed I/O examples

- In the above, since the switch and the LED data are aligned (both connected at bit 0 of the respective ports), the MOVFF PORTC, PORTD instruction directly inputs the switch input from bit 0 of PORTC and outputs to bit 0 of PORTD. Also, the infinite loop using BRA START will make the LOOP continuous. This means that after execution of the above program once, the LED will be turned ON and OFF automatically as soon as the switch is pressed.

Programmed I/O examples

- Example 8.2 A PIC018F4321 microcontroller is required to drive an LED connected to bit 7 of port C based on two switch inputs connected to bits 6 and 7 of port D. If both switches are equal (either HIGH or LOW), turn the LED ON; otherwise turn it OFF. Assume that a HIGH will turn the LED ON and a LOW will turn it OFF. Write a PIC18F assembly program at 0x200 to accomplish this.

Programmed I/O examples

```
        INCLUDE   <P18F4321.INC>
        ORG       0x200
        BCF       TRISC, TRISC7 ; Configure bit 7 of PORTC as an output
        SETF      TRISD         ; Configure PORTD as an input port
BACK    MOVF      PORTD, W      ; Input PORTD into WREG
        ANDLW     0xC0          ; Retain bits 6 and 7
        BZ        LEDON         ; If both switches are LOW, turn the LED ON
        SUBLW     0xC0          ; If both switches are HIGH, turn the LED ON
        BZ        LEDON
        BCF       PORTC, RC7    ; Turn LED OFF
        BRA       DOWN
LEDON   BSF       PORTC, RC7    ; Turn LED ON
DOWN    BRA       BACK
        END
```

Programmed I/O examples

- Example 8.3 The PIC18F4321 microcontroller shown in Figure 8.17 is required to input a BCD number (0-9) via four switches connected to bits 0 through 3 of PORTA, and then output the BCD digit to a common-anode seven segment display connected to bits 0 through 6 of PORTB. Write a PIC18F at address 0x70 to accomplish this by storing seven-segment codes in data memory starting at address 0x100.

Programmed I/O examples

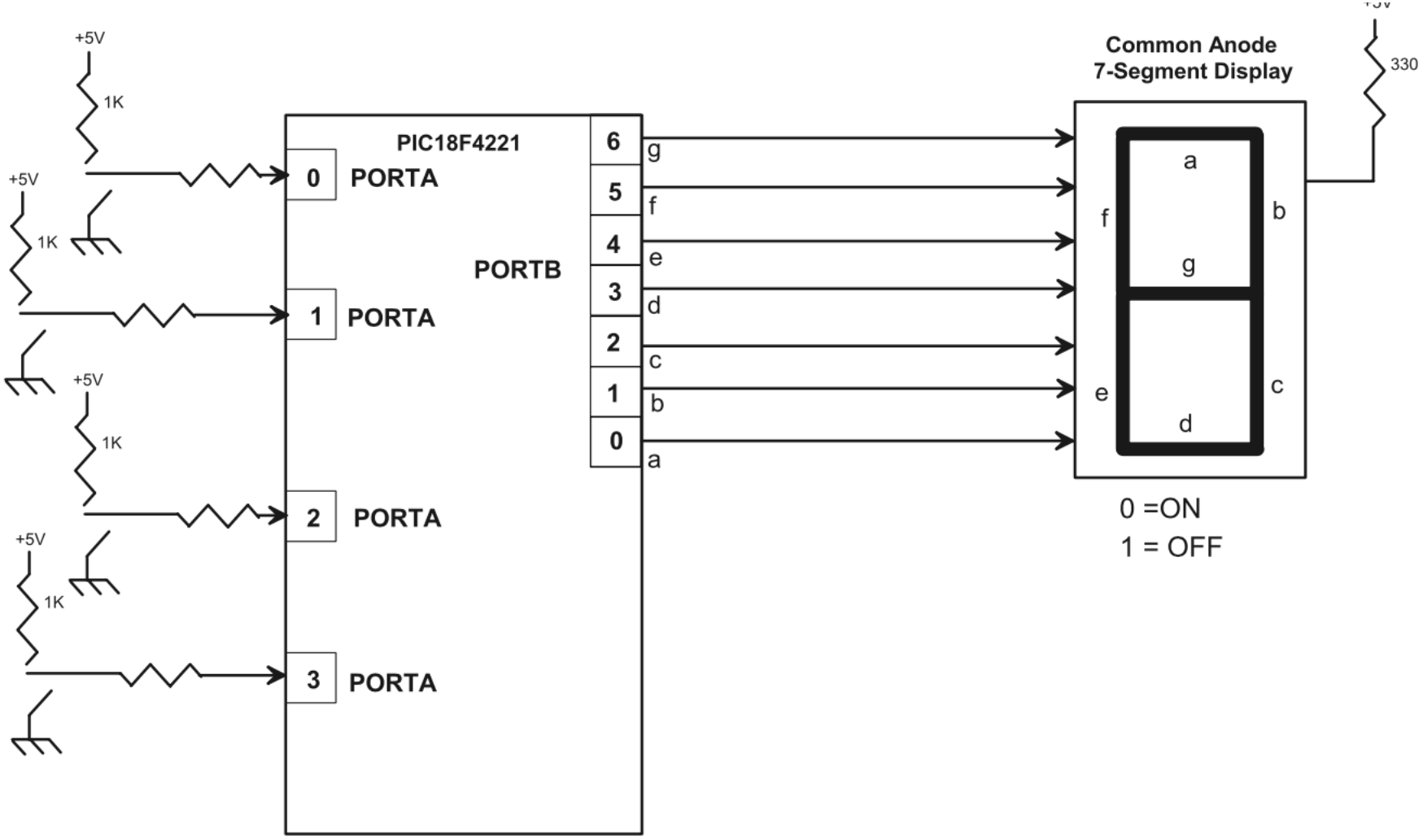
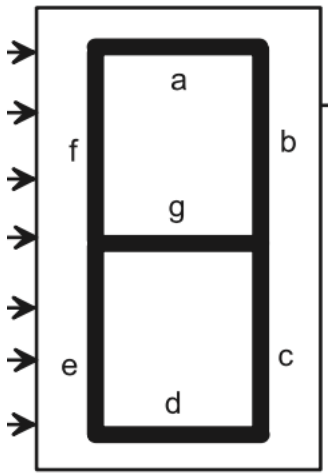


FIGURE 8.17 Figure for Example 8.3

Programmed I/O examples

- To find the proper values for the display, a table (also called "Lookup table") containing the seven segment code for each BCD digit can be obtained from Figure 8.17 as follows:



	g	f	e	d	c	b	a		Hex:
0:	1	0	0	0	0	0	0	=	0x40
1:	1	1	1	1	0	0	1	=	0x79
2:	0	1	0	0	1	0	0	=	0x24
3:	0	1	1	0	0	0	0	=	0x30
4:	0	0	1	1	0	0	1	=	0x19
5:	0	0	1	0	0	1	0	=	0x12
6:	0	0	0	0	0	1	1	=	0x03
7:	1	1	1	1	0	0	0	=	0x78
8:	0	0	0	0	0	0	0	=	0x00
9:	0	0	1	1	0	0	0	=	0x18

Programmed I/O examples

- Note that for a common-anode seven display, a ‘0’ will turn a segment ON and a ‘1’ will turn it OFF. Also, bit 7 of PORTB is assumed to be ‘0’. The above table must be stored in the data memory using MOVLW and MOVWF instructions or using the MPLAB assembler directive DB along with the PIC18F assembly language instruction TBLRD (Table Read) with appropriate addressing mode.

Programmed I/O examples

The PIC18F assembly language program is provided in the following:

```
    INCLUDE <P18F4321.INC>
    ORG     0x70
    MOVLW   0x40                ; Store 7-segment codes
    MOVWF   0x100
    MOVLW   0x79
    MOVWF   0x101
    MOVLW   0x24
    MOVWF   0x102
    MOVLW   0x30
    MOVWF   0x103
    MOVLW   0x19
    MOVWF   0x104
    MOVLW   0x12
```

Programmed I/O examples

```
        MOVWF   0x105
        MOVLW   0x03
        MOVWF   0x106
        MOVLW   0x78
        MOVWF   0x107
        MOVLW   0x00
        MOVWF   0x108
        MOVLW   0x18
        MOVWF   0x109
        MOVLW   0x0F            ; Configure PORTA as input
        MOVWF   ADCON1
        CLRF    TRISB           ; Configure PORTB as output
        LFSR    0, 0x100        ; Initialize FSR0 with the starting
                                ; address of lookup table
LOOP    MOVF    PORTA, W        ; Input switch data to the WREG
        ANDLW   B'00001111'     ; Mask the lower 4-bits
        MOVFF   PLUSW0, PORTB   ; Output BCD data to PORTB
        BRA     LOOP            ; Loop
        END
```

Programmed I/O examples using C

- Typical preprocessor directives are “#include”, “#define”, and “pragma config”. The “#include” preprocessor directive transfers texts (commands, comments, etc.) from another document into the program. For example, consider
  - #include <p18f4321.h>

Programmed I/O examples using C

- The file p18f4321.h is a PIC18F4321-specific header file, and primarily contains definitions for the variables used to access the Special Function Registers (SFRs) of the PIC18F4321. For example, PORTA of the PIC18F4321 is mapped as a Special Function Register (SFR) with data memory address 0xF80. This means that the user can use PORTA in the C-program. The compiler will insert the address 0xF80 during compilation whenever it encounters PORTA in the program.

Programmed I/O examples using C

- The “#define” preprocessor directive can be used to declare constants. For example, consider
  - #define y 4
- The “#define” preprocessor will assign 4 to y. Constants defined with the preprocessor “#define” are textual replacements performed before the program is compiled.

Programmed I/O examples using C

- The “#pragma config” preprocessor directive tells the compiler to perform a specific action before compilation. This directive can be used to turn certain on-chip devices ON or OFF. For example, the statement “#pragma config WDT = OFF “ placed after the “#include” directive will turn the watchdog timer OFF.

Programmed I/O examples using C

In this section, programmed I/O examples using C language are included. As mentioned before, in order for the I/O programs to work properly, the following "# pragma config" statements should be included in all the programming examples after the "#include <p18f4321.h>" statement:

```
#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
```

Programmed I/O examples using C

- The following C-function does not return the result to the main; the reserved word “void” can be used in such a situation. A simple example for the C18 C-compiler for the PIC18F is provided in the following:

```
#include <p18f4321.h>
// Initializations
void CONVERT (void);
void main (void)
{
Statements
CONVERT ( ); // Function Call
}     // end of main
void CONVERT ( )
{ Body of the Function

}
```

Programmed I/O examples using C

- An array of BCD codes for a common-anode seven-segment display can be created using the C18 compiler of the PIC18F as follows:

```
include <pic18f4321>
    void main ( )
     {
unsigned char code [10] = {0x40, 0x79, 0x24, ox30, 0x19, 0x12, 0x03, 0x78,
                           0x00, 0x18};
      ---------
      ---------
      ---------
      }
```

Programmed I/O examples using C

- Suppose that the common-anode seven-segment display is connected to PORTA of the PIC18F4321 microcontroller. Now, in order to output a code for a particular digit, for example digit 1, the following C code can be used:
  - PORTA = code [1];

Programmed I/O examples using C

- Delay routines in C Three ways of obtaining time delays using the PIC18F C18 compiler are:
  - Using PIC18F on-chip timers
  - Using a C-loop
  - Using C18 delay functions such as the function Delay 10KTcyx ( )

PIC18F Delay Routine

- Typical PIC18F software delay routines can be written by loading a “counter” with a value equivalent to the desired delay time, and then decrementing the “counter” in a loop using typically MOVE, DECREMENT, and conditional BRANCH instructions. For example, the following PIC18F instruction sequence can be used for a delay loop:

```
        MOVLW   COUNT
        MOVWF   0x20
DELAY   DECF    0X20, F
        BNZ     DELAY
```

PIC18F Delay Routine

- Note that DECF in the above decrements the register 0x20 by one, and if Z =0, the program branches to DELAY and if Z = 1, the PIC18F executes the next instruction. The initial loop counter value of "COUNT" can be calculated using the machine cycles required to execute the following PIC18F instructions:

MOVLW	(1 cycle)
MOVWF	(1 cycle)
DECF	(1 cycle)
BNZ	(2/1 cycles)

PIC18F Delay Routine

- Note that the BNZ instruction requires two different execution times. BNZ requires 2 cycles when the PIC18F branches if Z = 0. However, the PIC18F goes to the next instruction and does not branch when Z = 1. This means that the DELAY loop will require 2 cycles for “(COUNT-1)” times, and the last iteration will take 1 cycle. The desired delay time can be obtained by loading register 0x20 with the appropriate COUNT value.

PIC18F Delay Routine

Assuming a 1-MHz default crystal frequency, the PIC18F's clock period will be 1μs. Note that the PIC18F divides the crystal frequency by 4. This is equivalent to multiplying the clock period by 4. Hence, each instruction cycle will be 4 microseconds. For a 100 microsecond delay, total cycles $= \frac{100\ micro\,\text{sec}}{4\ micro\,\text{sec}} = 25$. The BNZ in the loop will require 2 cycles for (COUNT - 1) times when Z = 0 and the last iteration will take 1 cycle when no branch is taken (Z = 1). Thus, the total cycles including the MOVLW = Cycles for (MOVLW + MOVWF + (BNZ for Z = 0 + DCF) x (COUNT - 1)) + BNZ (Z = 1) = 1 + 1 + 3 × (COUNT - 1) + 1 = 25. Hence, COUNT = 8.3. Therefore, register 0x20 should be loaded with an integer value of 8 for an approximate delay of 100 microseconds.

PIC18F Delay Routine

Now, in order to obtain delay of one millisecond, the above DELAY loop of 100 microseconds can be used with an external counter. Counter value = $\frac{1\ milli\,\text{sec}}{100\ micro\,\text{secs}} = 10.$

The following instruction sequence will provide an approximate delay of one millisecond:

```
        MOVLW   D'10'
        MOVWF   0x30      ; Initialize counter 0x30 for one millisecond delay
BACK    MOVLW   8
        MOVWF   0x20      ; Initialize counter 0x20 for 100 microsecond delay
DELAY   DECF    0X20, F   ; 100 microsec delay
        BNZ     DELAY
        DECF    0X30, F
        BNZ     BACK
```

PIC18F Delay Routine

- The above instruction sequence will provide an approximate delay of one millisecond. Note that execution times of certain instructions such as “MOVLW D’10’”, “MOVWF 0x30”, “DECF 0x30, F”, and “BNZ BACK” are discarded since their execution times are negligible compared to one millisecond.

PIC18F Delay Routine

Example 7. 10 Assume 1 MHz PIC18F. Consider the following subroutine:

```
DELAY   MOVLW   D'100'
        MOVWF   0x20
LOOP    DECFSZ  0x20, F
        BRA     LOOP
        RETURN
```

(a) Calculate the time delay provided by the above subroutine.

(b) Calculate the counter value to be loaded into data register 0x20 for 1 ms delay.

All instructions in the above except GOTO and BRA are executed in one cycle; GOTO and BRA are executed in two cycles. The size of each instruction except GOTO is one work; the size of GOTO is two words.

PIC18F Delay Routine

(a) Each instruction in the above subroutine is executed in one cycle except for the DECFSZ instruction. DECFSZ is executed in one cycle if it does not skip, and two cycles if it skips (See Appendix D for instruction cycles).

Hence, total instruction cycles = Cycle for MOVLW + Cycle for MOVWF + (100-1) (Cycles for DECFSZ if it does not skip and BRA instructions) + (Cycle for DECFSZ if it skips) + Cycle for RETURN

$= 1+1+99(1+2)+2+1$

$= 302$

Since for the PIC18F, one instruction cycle = 4 clock cycles, total delay = 302 * 4 = 1208 clock cycles. Also, for 1 MHz clock, each clock cycle is 1 usec. Hence, total time delay = 1208 usec.

PIC18F Delay Routine

b) Let n be the counter value. Hence, (1+1 + (n-1)*(1+2)+2+1) * 4 = 1000 usec. Note that 1 ms = 1000 usec. Therefore, n = 332.6666. Therefore, data register 0x20 should be loaded with 333 for an approximate delay of 1 ms.

Programmed I/O examples using C

(a) Using PIC18F on-chip timers: Time delays using the PIC18F hardware timers will be discussed in Chapter 10.

(b) Using a C-loop: Time delays using "for" loop can be obtained as follows:

```
void delay(unsigned int itime)        // 2 ms delay
{
        unsigned int i,j;
        for(i=0; i<itime; i++)
                for(j=0; j<255;j++);
}
```

Programmed I/O examples using C

The above "delay" function in C will provide a time delay of 2 ms using an internal default clock of 1-MHz of the PIC18F4321. In order to determine that the "delay" function will provide a time delay of 3 ms, disassembly of the C-program is given below:

```
DELAY   MOVWF   0x20
LOOP1   MOVLW   D'255'   ; LOOP2 provides 3 ms delay with a count of 255
        MOVWF    0x21
LOOP2   DECFSZ  0X21
        GOTO    LOOP2
        DECFSZ  0x20
        GOTO    LOOP1
        RETURN
```

{[1 + 1 + (1+2)*254 + 2] + 1 + 2} * (count stored in 0x20 – 1) + 2 + 1

[1 + 1 + (1+2)*254 + 2] + 1 + 2 = 769 instruction cycles; 769 * 4 = 3076 usec ≈ 3 ms

Programmed I/O examples using C

- (c) Using C18 delay functions such as the function Delay 10KTCYx( ):
- The C18 compiler contains a function called “Delay 10KTCYx( )” which allows the programmer to specify a time delay in multiples of 10,000 instruction cycles. This “Delay” function is included in the C18 compiler’s header file “delays.h”.. The details of the function name “Delay 10KTCYx( )” can be broken down as follows:

Programmed I/O examples using C

- Function “Delay” is used to provide a fixed time delay.
- “10K” means ten thousand.
- “TCY” means the instruction cycle, Tcy.
- “x” multiplied by the argument (number) specified in the bracket. The number in the bracket must be an 8-bit unsigned integer (0-255). Hence, the time delay created will be the number in brackets multiplied by 10,000 instruction cycles.

Programmed I/O examples using C

As an example, consider writing a C-program to generate a time delay of 10 sec with an internal clock of 4-MHz using the C18 function "Delay10KTCYx( )".

With OSC = 4Mhz, Timer frequency is (4MHz)/4 = 1MHz. Hence, period is 1μsec. For 10 sec delay, Cycles = (10)/(10^{-6}) = 10^{7} = 10,000,000. The C18 function "Delay10KTCYx(250)" will provide delays of 2,500,000 cycles. Performing a loop with the function "Delay10KTCYx(250)" four times will create a delay of 10 sec.

The C-program is provided below:

```
#include <P18F4321.h>
#include <delays.h>     // delay function library
{
unsigned char count;
void main()
    OSCCON = 0x60;  // 4Mhz
    for (count = 0; count < 4; count++)  // Use delay function 4 times
    Delay10KTCYx(250); // Delay in multiples of 10,000 instruction cycles.

}
```

Programmed I/O examples using C

- Example 8.4 Assume PIC18F4321. Suppose that three switches are connected to bits 0-2 of PORTC and an LED to bit 6 of PORTD. If the number of HIGH switches is even, turn the LED on; otherwise, turn the LED off. Write a C language program to accomplish this.

Programmed I/O examples using C

The C language program is shown below:

```
#include <p18f4321.h>
#define    portc0  PORTCbits.RC0
#define    portc1  PORTCbits.RC1
#define    portc2  PORTCbits.RC2
void main (void)
{
      unsigned char mask = 0x07;              // data for masking off upper 5 bits of  PORTC
      unsigned char masked_in;
```

Programmed I/O examples using C

```
unsigned char xor_bit;
TRISC = 0xFF;                          // Configure PORTC as an input port
TRISD = 0;                             // Configure PORTD as an output port
while(1)
{
        masked_in  =  PORTC & mask;          //  Mask input  bits
if (masked_in = = 0)
        PORTD = 0x40;                  // For all low switches (even), turn led on
else
        xor_bit = portc0 ^ portc1 ^ portc2; // xor input bits
if (xor_bit = = 0)                     // for even # of high switches ,
        PORTD = 0x40;                  // turn led on
else
        PORTD = 0;                     // for odd # of high switches, turn led off
}
}
```

Programmed I/O examples using C

- Example 8.5 Assume PIC18F4321. Suppose that it is desired to input a switch connected to bit 4 of PORTC, and then output it to an LED connected at bit 2 of PORTD. Write a C language program to accomplish this.

Programmed I/O examples using C

Solution

The following C code will accomplish this:

```
# include  <P18F4321.h>
# define portc_bitin  PORTCbits.RC4      // Declare a bit (bit 4) of  PORTC
# define portd_bitout  PORTDbits.RD2   // Declare a bit (bit 2) of  PORTD
void main  (void )
{
    TRISCbits.TRISC4 = 1;       // Configure bit 4 of PORTC as an input bit
    TRISDbits.TRISD2 = 0;       // Configure bit 2 of PORTD as an output bit
    while  (1)                          //  halt
        {
    portd_bitout = portc_bitin;   // output switch to LED
        }
}
```

Programmed I/O examples using C

- Example 8.6 The PIC18F4321 microcontroller shown in Figure 8.18 is required to output a BCD digit (0 to 9) to a common-anode seven segment display connected to bits 0 through 6 of PORTB. The PIC18F4321 inputs the BCD number via four switches connected to bits 0 through 3 of PORTA. Write a C language program that will display a BCD digit (0 to 9) on the seven segment display based on the switch inputs. Use a lookup table for the seven-segment codes.

Programmed I/O examples using C

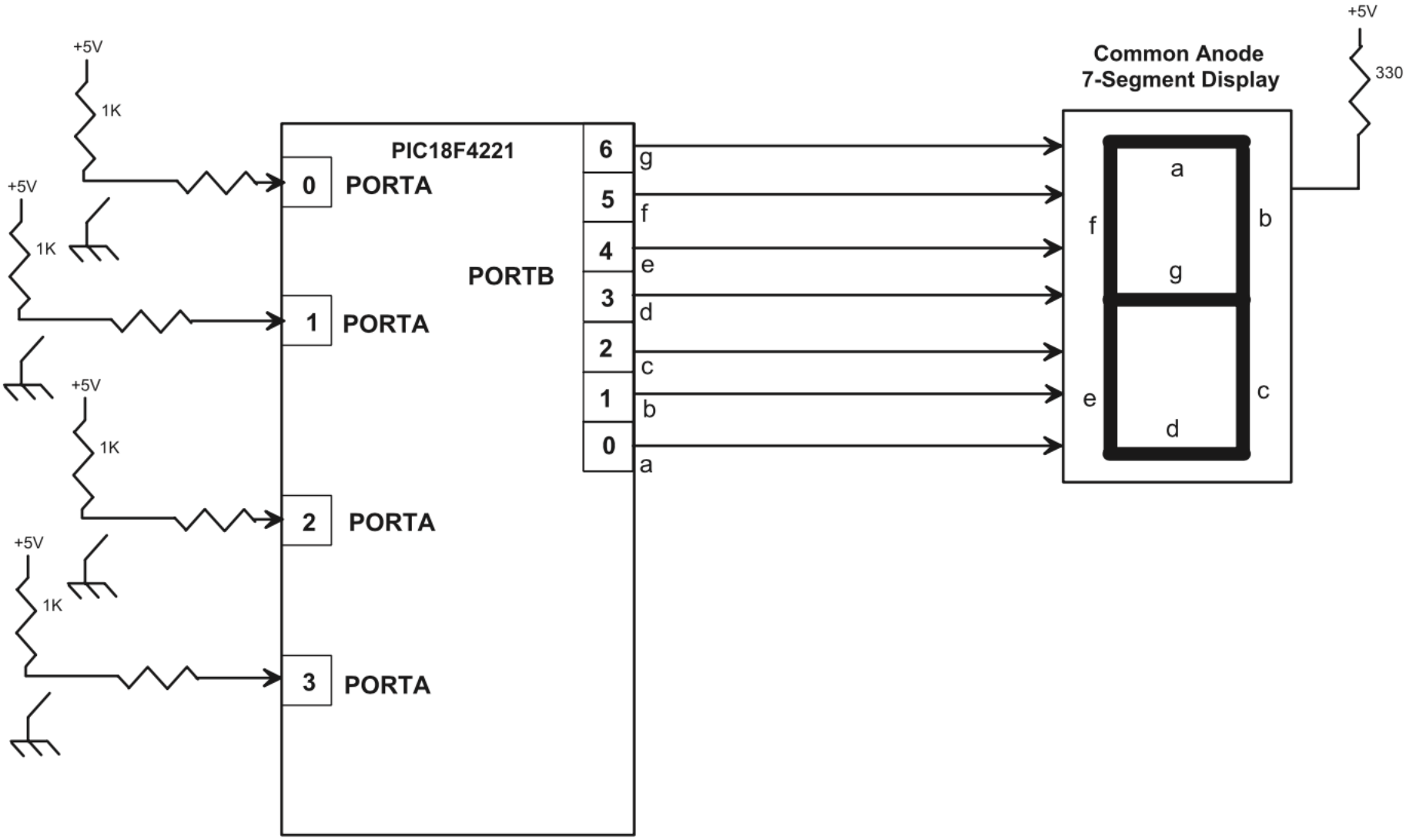


FIGURE 8.18 Figure for Example 8.6

Programmed I/O examples using C

- Solution: To find the proper values for the display, a table (also called “Lookup table”) containing the seven segment code for each BCD digit can be obtained from Figure 8.18 as follows:

	g	f	e	d	c	b	a		Hex:
0:	1	0	0	0	0	0	0	=	0x40
1:	1	1	1	1	0	0	1	=	0x79
2:	0	1	0	0	1	0	0	=	0x24
3:	0	1	1	0	0	0	0	=	0x30
4:	0	0	1	1	0	0	1	=	0x19
5:	0	0	1	0	0	1	0	=	0x12
6:	0	0	0	0	0	1	1	=	0x03
7:	1	1	1	1	0	0	0	=	0x78
8:	0	0	0	0	0	0	0	=	0x00
9:	0	0	1	1	0	0	0	=	0x18

Programmed I/O examples using C

Note that for a common-anode seven segment display, a '0' will turn a segment ON and a '1' will turn it OFF. Also, bit 7 of PORTB is assumed to be '0'.

The C code is provided below:

```
#include <p18f4321.h>
 void main ()
{
unsigned char input;
unsigned char code[10] = {0x40, 0x79, 0x24, 0x30, 0x19, 0x12, 0x03, 0x78, 0x00, 0x18};
ADCON1 = 0x0F; // PORTA is input
TRISB = 0; // PORTB is output
while (1) {
        input = PORTA & 0x0F; // Mask off switch inputs
        PORTB = code [input];  // Output code to 7-segment display via PORTB
        }
}
```

Programmed I/O examples using C

- In the above program, first the PORTB is set as an output port and PORTA is set as an input port. A variable ‘input’ is then declared. The program moves to an infinite ‘while’ loop where it will first take the input from the four switches via PORTA, and mask the first four bits. An unsigned char array code is set up in order to contain the seven-segment code for each decimal digit from 0 through 9. The input is used as the index to the array, and the corresponding 7-segment code is sent to PORTB.