SECTION I

An Introduction to PIC Microcontrollers

CHAPTER 1

The PIC Microcontroller Family

The PIC microcontroller family is manufactured by Microchip Technology Inc. Currently they are one of the most popular microcontrollers, used in many commercial and industrial applications. Over 120 million devices are sold each year.

The PIC microcontroller architecture is based on a modified Harvard RISC (Reduced Instruction Set Computer) instruction set with dual-bus architecture, providing fast and flexible design with an easy migration path from only 6 pins to 80 pins, and from 384 bytes to 128 kbytes of program memory.

PIC microcontrollers are available with many different specifications depending on:

- Memory Type
  - Flash
  - OTP (One-time-programmable)
  - ROM (Read-only-memory)
  - ROMless
- Input–Output (I/O) Pin Count
  - 4–18 pins
  - 20–28 pins
  - 32–44 pins
  - 45 and above pins
- Memory Size
  - 0.5–1 K
  - 2–4 K
  - 8–16 K
  - 24–32 K
  - 48–64 K
  - 96–128 K
- Special Features
  - CAN
  - USB

  - LCD
  - Motor Control
  - Radio Frequency

Although there are many models of PIC microcontrollers, the nice thing is that they are upward compatible with each other and a program developed for one model can very easily, in many cases with no modifications, be run on other models of the family. The basic assembler instruction set of PIC microcontrollers consists of only 33 instructions and most of the family members (except the newly developed devices) use the same instruction set. This is why a program developed for one model can run on another model with similar architecture without any changes.

All PIC microcontrollers offer the following features:

- RISC instruction set with only a handful of instructions to learn
- Digital I/O ports
- On-chip timer with 8-bit prescaler
- Power-on reset
- Watchdog timer
- Power-saving SLEEP mode
- High source and sink current
- Direct, indirect, and relative addressing modes
- External clock interface
- RAM data memory
- EPROM or Flash program memory

Some devices offer the following additional features:

- Analog input channels
- Analog comparators
- Additional timer circuits
- EEPROM data memory
- External and internal interrupts
- Internal oscillator

- Pulse-width modulated (PWM) output
- USART serial interface

Some even more complex devices in the family offer the following additional features:

- CAN bus interface
- I^2C bus interface
- SPI bus interface
- Direct LCD interface
- USB interface
- Motor control

Although there are several hundred models of PIC microcontrollers, choosing a microcontroller for an application is not a difficult task and requires taking into account these factors:

- Number of I/O pins required
- Required peripherals (e.g., USART, USB)
- The minimum size of program memory
- The minimum size of RAM
- Whether or not EEPROM nonvolatile data memory is required
- Speed
- Physical size
- Cost

The important point to remember is that there could be many models that satisfy all of these requirements. You should always try to find the model that satisfies your minimum requirements and the one that does not offer more than you may need. For example, if you require a microcontroller with only 8 I/O pins and if there are two identical microcontrollers, one with 8 and the other one with 16 I/O pins, you should select the one with 8 I/O pins.

Although there are several hundred models of PIC microcontrollers, the family can be broken down into three main groups, which are:

- 12-bit instruction word (e.g., 12C5XX, 16C5X) (also referred to in this book as the 12 Series and the 16C5X Series)

- 14-bit instruction word (e.g., 16F8X, 16F87X) (also referred to in this book as the 16 Series)
- 16-bit instruction word (e.g., 17C7XX, 18C2XX) (also referred to in this book as the 17 Series and the 18 Series).

All three groups share the same RISC architecture and the same instruction set, with a few additional instructions available for the 14-bit models, and many more instructions available for the 16-bit models. Instructions occupy only one word in memory, thus increasing the code efficiency and reducing the required program memory. Instructions and data are transferred on separate buses, so the overall system performance is increased.

The features of some microcontrollers in each group are given in the following sections.

1.1 12-bit Instruction Word

Table 1.1 lists some of the devices in this group. These devices have a very simple architecture; however, as the prices of 14-bit devices have declined, it is rarely necessary to use a 12-bit device these days, except for the smaller physical size.

Table 1.1: Some 12-bit PIC Microcontrollers

Microcontroller	Program Memory	Data RAM	Max Speed (MHz)	I/O Ports	A/D Converter
12C508	512 × 12	25	4	6	–
16C54	384 × 12	25	20	12	–
16C57	2048 × 12	72	20	20	–
16C505	1024 × 12	41	4	12	–
16C58A	2048 × 12	73	20	12	–

PIC12C508: This is a low-cost, 8-pin device with 512 × 12 EPROM program memory, and 25 bytes of RAM data memory. The device can operate at up to 4-MHz clock input and the instruction set consists of only 33 instructions. The device features six I/O ports, 8-bit timer, power-on reset, watchdog timer, and internal 4-MHz oscillator capability. One of the major disadvantages of this microcontroller is that the program memory is EPROM-based and it cannot be erased or programmed using the standard programming devices. The program memory has to be erased using an EPROM eraser device (an ultraviolet light source).

The “F” version of this family (e.g., PIC12F508) is based on flash program memory, which can be erased and reprogrammed using the standard PIC programmer devices. Similarly, the “CE” version of the family (e.g., PIC12CE518) offers an additional 16-byte nonvolatile EEPROM data memory.

Figure 1.1 shows the pin configuration of the PIC12F508 microcontroller.

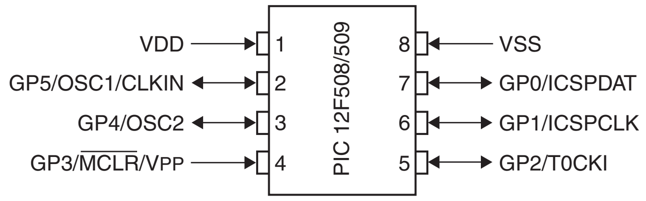

Figure 1.1: PIC12F508 Microcontroller

PIC16C5X: This is one of the earliest PIC microcontrollers. The device is 18-pin with a 384 × 12 EPROM program memory, 25 bytes of RAM data memory, 12 I/O ports, a timer, and a watchdog. Some other members in the family, such as PIC16C56, have the same architecture but more program memory (1024 × 12). PIC16C58A has more program memory (2048 × 12) and also more data memory (73 bytes of RAM). Figure 1.2 shows the pin configuration of the PIC16C56 microcontroller.

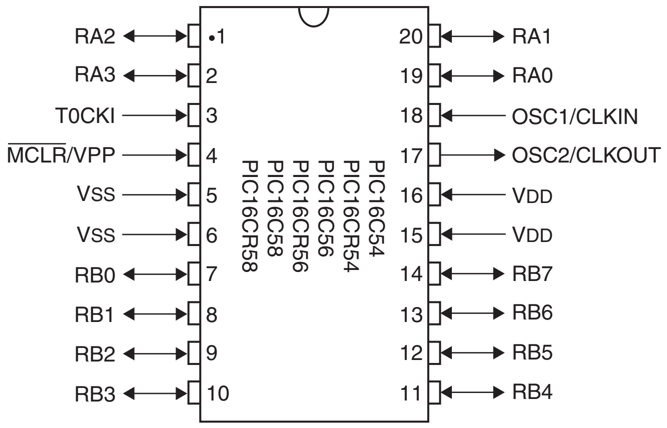

Figure 1.2: PIC16C56 Microcontroller

1.2 14-bit Instruction Word

This is a big family that includes many models of PIC microcontrollers. Most of the devices in this family can operate at up to a 20-MHz clock rate. The instruction set consists of 35 instructions. These devices offer advanced features such as internal and external interrupt sources. Table 1.2 lists some of the microcontrollers in this group.

PIC16C554: This microcontroller has similar architecture to the PIC16C54 but the instructions are 14 bits wide. The program memory is 512 × 14 and the data memory is 80 bytes of RAM. There are 13 I/O pins and each pin can source or sink 25 mA of current. Additionally, the device contains a timer and a watchdog.

Table 1.2: Some 14-bit Microcontrollers

Microcontroller	Program Memory	Data RAM	Max Speed (MHz)	I/O Ports	A/D Converter
16C554	512 × 14	80	20	13	–
16C64	2048 × 14	128	20	33	–
16F84	1024 × 14	36	10	13	–
16F627	1024 × 14	224	20	16	–
16F628	2048 × 14	224	20	16	–
16F676	1024 × 14	64	20	12	8
16F73	4096 × 14	192	20	22	5
16F876	8192 × 14	368	20	22	5
16F877	8192 × 14	368	20	33	8

PIC16F84: This has been one of the most popular PIC microcontrollers for a very long time. This is an 18-pin device and it offers 1024 × 14 flash program memory, 36 bytes of data RAM, 64 bytes of nonvolatile EEPROM data memory, 13 I/O pins, a timer, a watchdog, and internal and external interrupt sources. The timer is 8 bits wide but can be programmed to generate internal interrupts for timing purposes. PIC16F84 can be operated from a crystal or a resonator for accurate timing. A resistor-capacitor can also be used as a timing device for applications where accurate timing is not required. Figure 1.3 shows the pin configuration of this microcontroller. The pin descriptions are given in Table 1.3.

PIC16F877: This microcontroller is a 40-pin device and is one of the popular microcontrollers used in complex applications. The device offers 8192 × 14 flash program memory,

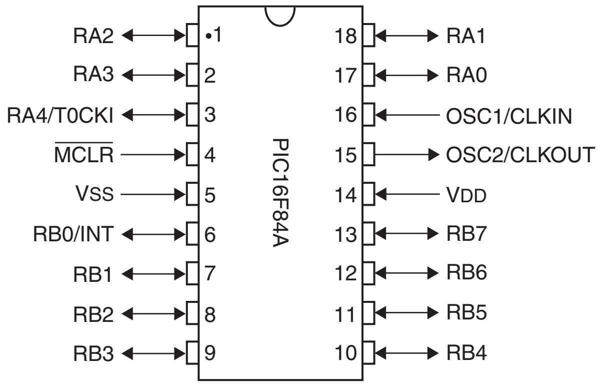

Figure 1.3: PIC16F84 Microcontroller Pin Configuration

Table 1.3: PIC16F84 Microcontroller Pin Descriptions

Pin	Description	Pin	Description
1	RA2—PORTA bit 2	10	RB4—PORTB bit 4
2	RA3—PORTA bit 3	11	RB5—PORTB bit 5
3	RA4/T0CK1—PORTA bit 4/Counter clk	12	RB6—PORTB bit 6
4	MCLR—Master clear	13	RB7—PORTB bit 7
5	Vss—Gnd	14	Vdd—+V supply
6	RB0/INT—PORTB bit 0	15	OSC2
7	RB1—PORTB bit 1	16	OSC1
8	RB2—PORTB bit 2	17	RA0—PORTA bit 0
9	RB3—PORTB bit 3	18	RA1—PORTA bit 1

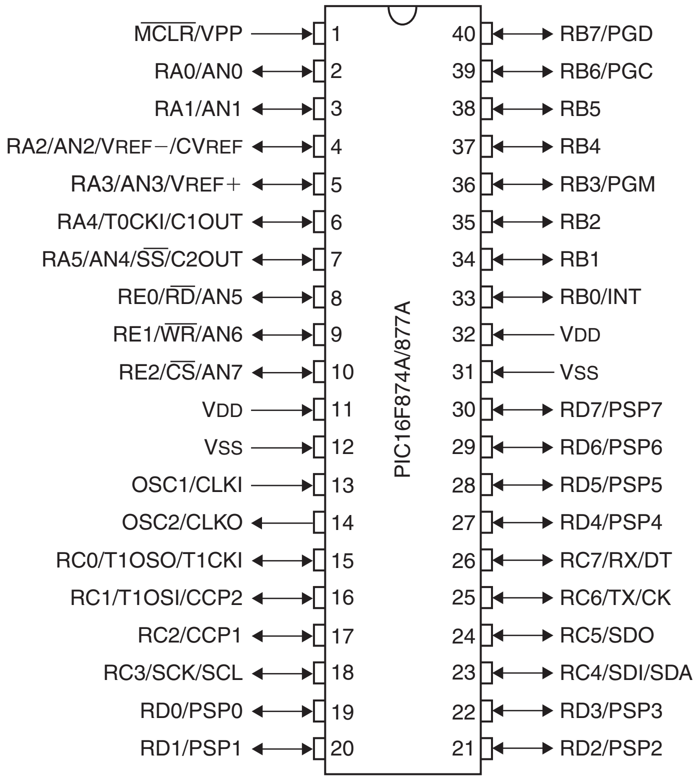

Figure 1.4: PIC16F877 Microcontroller Pin Configuration

368 bytes of RAM, 256 bytes of nonvolatile EEPROM memory, 33 I/O pins, 8 multiplexed A/D converters with 10 bits of resolution, PWM generator, three timers, an analog capture and comparator circuit, USART, and internal and external interrupt facilities. Figure 1.4 shows the pin configuration of this microcontroller.

PIC16F627: This is an 18-pin microcontroller with 1024 × 14 flash program memory. The device offers 224 bytes of RAM, 128 bytes of nonvolatile EEPROM memory, 16 I/O pins, two 8-bit timers, one 16-bit timer, a watchdog, and comparator circuits. This microcontroller is similar to PIC16F84, but offers more I/O pins, more program memory, and a lot more RAM. In addition, PIC16F627 is more suited to applications that require more than one timer. Figure 1.5 shows the pin configuration of this microcontroller.

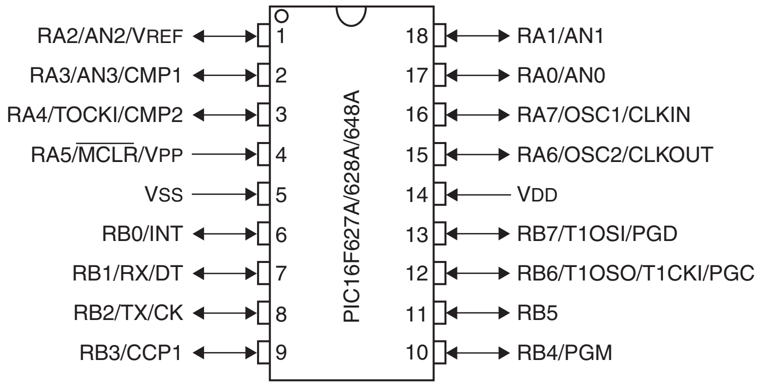

Figure 1.5: PIC16F627 Microcontroller Pin Configuration

PIC16F676: Figure 1.6 shows a 14-pin microcontroller that is becoming very popular. The device offers 1024 × 14 flash program memory, 64 bytes of RAM, 12 I/O pins, 128 bytes of EEPROM, 8 multiplexed A/D converters, each with 10-bit resolution, one 8-bit timer, one 16-bit timer, and a watchdog. One of the advantages of this microcontroller is the built-in A/D converter.

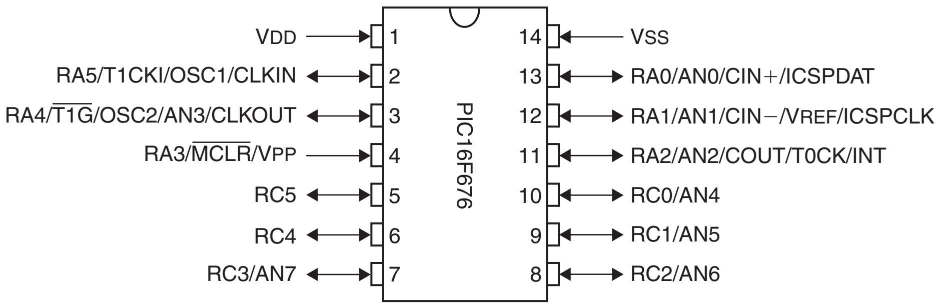

Figure 1.6: PIC16F676 Microcontroller Pin Configurationm

PIC16F73: This is a powerful 28-pin microcontroller with 4096 14 flash program memory, 192 bytes of RAM, 22 I/O pins, five multiplexed 8-bit A/D converters, two 8-bit timers, one 16-bit timer, watchdog, USART, and I^{2}C bus compatibility. This device combines A/D converter, digital I/O, and serial I/O capability in a 28-pin medium size package. Figure 1.7 shows the pin configuration of this microcontroller.

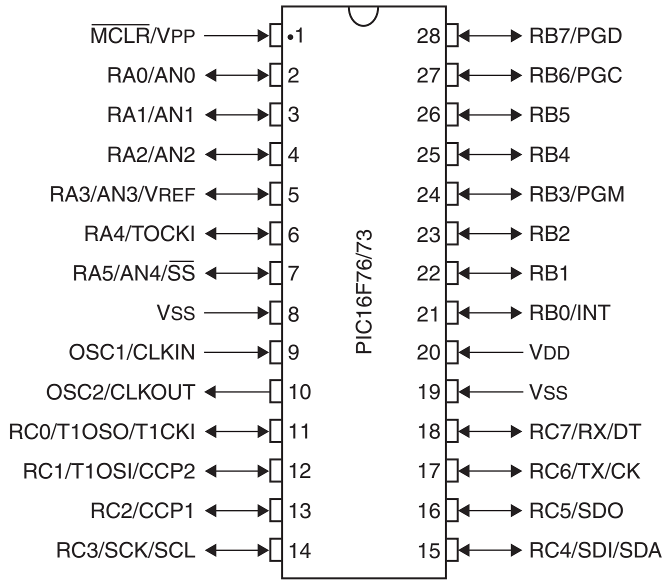

Figure 1.7: PIC16F73 Microcontroller Pin Configuration

1.3 16-bit Instruction Word

The 16-bit microcontrollers are at the high end of the PIC microcontroller family. Most of the devices in this group can operate at up to 40 MHz, have 33 I/O pins, and three timers. They have 23 instructions in addition to the 35 instructions found on the 14-bit microcontrollers. Table 1.4 lists some of the devices in this family.

All memory for the PIC microcontroller family is internal and it is usually not very easy to extend this memory externally. No special hardware or software features are provided for

Table 1.4: Some 16-bit PIC Microcontrollers

Microcontroller	Program Memory	Data RAM	Max Speed (MHz)	I/O Ports	A/D Converter
17C43	4096 × 16	454	33	33	–
17C752	8192 × 16	678	33	50	12
18C242	8192 × 16	512	40	23	5
18C252	16384 × 16	1536	40	23	5
18F4520	32768 × 16	1536	40	36	13

extending either the program memory or the data memory. The program memory is usually sufficient for small to medium size projects. But the data memory is generally small and may not be enough for medium to large projects unless a bigger and more expensive member of the family is chosen. For some large projects even this may not be enough and the designer may have to sacrifice the I/O ports to interface an external data memory, or to choose a microcontroller from a different manufacturer.

1.4 Inside a PIC Microcontroller

Although there are many models of microcontrollers in the PIC family, they all share some common features, such as program memory, data memory, I/O ports, and timers. Some devices have additional features such as A/D converters, USARTs and so on. Because of these common features, we can look at these attributes and cover the operation of most devices in the family.

1.4.1 Program Memory (Flash)

The program memory is where your program resides. In early microprocessors and microcontrollers the program memory was EPROM, which meant that it had to be erased using UV light before it could be reprogrammed. Most PIC microcontrollers nowadays are based on flash technology, where the memory chip can be erased or reprogrammed using a programmer device. Most PIC microcontrollers can also be programmed without removing them from their circuits. This process (called in-circuit serial programming, or ISP) speeds up the development cycle and lowers the development costs. Although the program memory is mainly used to store a program, there is no reason why it cannot be used to store constant data used in programs.

PIC microcontrollers can have program memories from 0.5 to over 16 K. A PicBasic program can have several pages of code and still fit inside 1 K of program memory. The width of a 14-bit program memory is actually 14 bits. It is interesting to note that PICs are known as 8-bit microcontrollers. This is actually true as far as the width of the data memory is concerned, which is 8 bits wide. Microchip calls the 14 bits a *word*, even though a *word* is actually 16 bits wide.

When power is applied to the microcontroller or when the MCLR input is lowered to logic 0, execution starts from the Reset Vector, which is the first word of the program memory. Thus, the first instruction executed after a reset is the one located at address 0 of the program memory. When the program is written in assembler language, the programmer has to use special instructions (called ORG) so that the first executable instruction is loaded into address 0 of the program memory. High-level languages such as PicBasic or PicBasic Pro compile your program such that the first executable statement in your program is loaded into the first location of the program memory.

1.4.2 Data Memory (RAM)

The data memory is used to store all of your program variables. This is a RAM, which means that all the data is lost when power is removed. The data memory is 8 bits wide and this is why the PIC microcontrollers are called 8-bit microcontrollers.

The data memory in a PIC microcontroller consists of banks, with some models having only two banks, some models four banks, and so on. A required bank of the data memory can be selected under program control.

1.4.3 Register File Map and Special Function Registers

Register File Map (RFM) is a layout of all the registers available in a microcontroller and this is extremely useful when programming the device, especially when using assembler language. The RFM is divided into two parts: the *Special Function Registers* (SFR), and the *General Purpose Registers* (GPR). For example, on a PIC16F84 microcontroller there are 68 GPR registers and these are used to store temporary data.

SFR is a collection of registers used by the microcontroller to control the internal operations of the device. Depending upon the complexity of the devices, the number of registers in the SFR varies. It is important that the programmer understands the functions of the SFR registers fully since they are used both in assembly language and in high-level languages.

Depending on the model of PIC microcontroller used, there could be other registers. For example, writing and reading from the EEPROM are controlled by SFR registers EECON1, EECON2, EEADR, and EEDATA. Fortunately, PicBasic and PicBasic Pro compilers provide simple high-level instructions for writing to and reading from the EEPROM and thus you do not need to know how to load these registers if you are programming in these languages.

Some of the important SFR registers that you may need to configure while programming using a high-level language are

- OPTION register
- I/O registers
- Timer registers
- INTCON register
- A/D converter registers

The functions and the bit definitions of these registers are described in detail in the following sections.

1.4.3.1 OPTION Register

This register is used to set up various internal features of the microcontroller and is named as OPTION_REG. This is a readable and writable register containing various control bits

7	6	5	4	3	2	1	0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Bit 7: PORTB Pull-up Enable
- 1: PORTB pull-ups disabled
- 0: PORTB pull-ups enabled

Bit 6: INT Interrupt Edge Detect
- 1: Interrupt on rising edge of INT input
- 0: Interrupt on falling edge of INT input

Bit 5: TMR0 Clock Source
- 1: T0CK1 pulse
- 0: Internal oscillator

Bit 4: TMR0 Source Edge Select
- 1: Increment on HIGH to LOW of T0CK1
- 0: Increment on LOW to HIGH of T0CK1

Bit 3: Prescaler Assignment
- 1: Prescaler assigned to Watchdog Timer
- 0: Prescaler assigned to TMR0

Bit 2-0: Prescaler Rate

000	1:2
001	1:4
010	1:8
011	1:16
100	1:32
101	1:64
110	1:128
111	1:256

Figure 1.8: OPTION_REG Bit Definitions

to configure the on-chip timer and the watchdog timer. This register is at address 81 (hexadecimal) of the microcontroller and its bit definitions are given in Figure 1.8. The OPTION REG register is also used to control the external interrupt pin RB0. This pin can be set up to generate an interrupt—for example, when it changes from logic 0 to logic 1. The microcontroller then suspends the main program execution and jumps to the interrupt service routine (ISR) to service the interrupt. Upon return from the interrupt, normal processing resumes.

For example, to configure the INT pin so that external interrupts are accepted on the rising edge of the INT pin, the following bit pattern should be loaded into the OPTION_REG:

X1XXXXXX

where X is a don't care bit and can be a 0 or a 1. We shall see later how to configure various bits of this register.

1.4.3.2 I/O Registers

These registers are used for the I/O control. Every I/O port in the PIC microcontroller has two registers: *port data register* and *port direction control register*.

Port data register has the same name as the port it controls. For example, the PIC16F84 microcontroller has two port data registers, PORTA and PORTB. A PIC16F877 microcontroller has five port data registers, PORTA, PORTB, PORTC, PORTD, and PORTE. Eight bits of data can be sent to any port, or 8 bits of data can be read from the ports. It is also possible to read or write to individual port pins. For example, any bit of a given port can be set or cleared, or data can be read from one or more port pins at the same time.

Ports in a PIC microcontroller are bidirectional. Thus, each pin of a port can be used as an input or an output pin. Port direction control register configures the port pins as either inputs or outputs. This register is called the TRIS register and every port has a TRIS register named after its port name. For example, TRISA is the direction control register for PORTA. Similarly, TRISB is the direction control register for PORTB and so on.

Setting a bit in the TRIS register makes the corresponding port register pin an input. Clearing a bit in the TRIS register makes the corresponding port pin an output. For example, to make bits 0 and 1 of PORTB input and the other bits output, we have to load the TRISB register with the bit pattern.

    00000011

Figure 1.9 shows the TRISB register and the direction of PORTB pins.

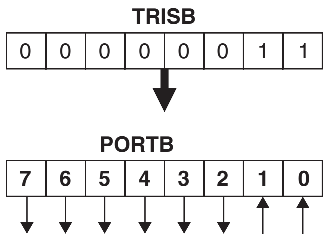

Figure 1.9: TRISB and PORTB Direction

Note that port data register and port direction control registers can be accessed directly using the PicBasic Pro compiler. For example, as we shall see later, TRISB register can be set to 3 and data can be read from PORTB into a variable named CNT by the PicBasic Pro instructions:

```
TRISB = 3
CNT = PORTB
```

The PicBasic compiler has no direct register control instructions and, as we shall see later, we have to use the PEEK and POKE instructions. PEEK is used to read data from a register and POKE is used to send data to a register.

When we use the PEEK and POKE instructions, we have to specify the register address of the register we wish to access. The register addresses of port registers are (the "$" character specifies that the number is in hexadecimal format):

Ports	**Address (Hexadecimal)**
PORTA	$05
PORTB	$06
PORTC	$07
PORTD	$08
PORTE	$09
TRISA	$85
TRISB	$86
TRISC	$87
TRISD	$88
TRISE	$89

Thus, for the above example, the required PicBasic instructions will be

```
POKE $86, 3
PEEK $06, CNT
```

1.4.3.3 Timer Registers

Depending on the model used, some PIC microcontrollers have only one timer, and some may have up to three timers. In this section we shall look at the PIC16F84 microcontroller, which has only one timer. The extension to several timers is similar and we shall see in the projects section how to use more than one timer.

The timer in the PIC16F84 microcontroller is an 8-bit register (called TMR0), which can be used as a timer or a counter. When used as a counter, the register increments each time a clock pulse is applied to pin T0CK1 of the microcontroller. When used as a timer, the register increments at a rate determined by the system clock frequency and a prescaler selected by register OPTION_REG. Prescaler rates vary from 1:2 to 1:256. For example, when using a 4-MHz clock, the basic instruction cycle is 1 μs (the clock is internally divided by four). If we select a prescaler rate of 1:16, the counter will be incremented at every 16 μs.

The TMR0 register has address 01 in the RAM.

A timer interrupt is generated when the timer overflows from 255 to 0. This interrupt can be enabled or disabled by our program. Thus, for example, if we need to generate interrupts at intervals of 200 μs using a 4-MHz clock, we can select a prescaler value of 1:4 and enable timer interrupts. The timer clock rate is then 4 μs. For a time-out of 200 μs, we have to send 50 clocks to the timer. Thus, the TMR0 register should be loaded with 256 – 50 = 206—i.e., a count of 50 before an overflow occurs.

The watchdog timer's oscillator is independent from the CPU clock and the time-out is 18 ms. To prevent a time-out condition the watchdog must be reset periodically via software. If the watchdog timer is not reset before it times out, the microprocessor will be forced to jump to the reset address. The prescaler can be used to extend the time-out period and valid rates are 1, 2, 4, 8, 16, 32, 64, and 128. For example, when set to 128, the time-out period is about 2 s ($18 \times 128 = 2304$ ms). The watchdog timer can be disabled during programming of the device if it is not used.

Since the timer is a very important part of the PIC microcontrollers, more detailed information is given on its operation below.

1.4.3.4 TMR0 and Watchdog

TMR0 and a watchdog are found in nearly all PIC microcontrollers. Figure 1.10 shows the functional diagram of TMR0 and the watchdog circuit. The operation of the watchdog circuit is as described earlier and only the TMR0 circuit is described in this section.

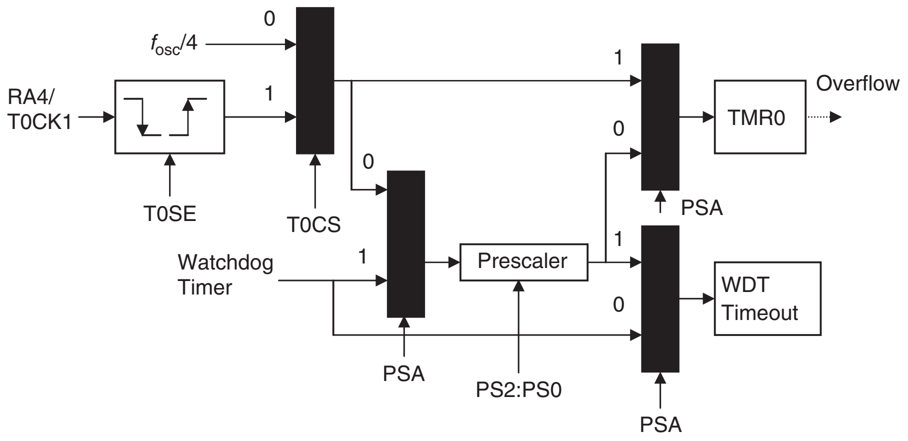

Figure 1.10: TMR0 and Watchdog Circuit

The source of input for TMR0 is selected by bit T0CS of OPTION_REG and it can be either from the microcontroller oscillator f_{osc} divided by 4, or it can be an external clock applied to the RA4/T0CK1 input. Here, we will only look at using the internal oscillator. If a 4-MHz crystal is used, the internal oscillator frequency is $f_{osc}/4 = 1$ MHz, which corresponds to a period of T $= 1/f = 10^{-6}$, or 1 μs. TMR0 is then selected as the source for the prescaler by clearing the PSA bit of OPTION_REG. The required prescaler value is selected by bits PS0 to PS2 as shown in Fig. 1.8. Bit PSA should then be cleared to 0 to select the prescaler for the timer. All the bits are configured now and the TMR0 register increments each time a pulse is applied by the internal oscillator. TMR0 register is 8 bits wide and it counts up to 255, then creates an overflow condition, and continues counting from 0. When TMR0 changes from 255 to 0 it generates a timer interrupt if timer interrupts and global interrupts are enabled (see INTCON register; an interrupt will be generated if GIE and TMR0 bits of INTCON are both set to 1). See Section 1.4.6 on Interrupts for more information.

By loading a value into the TMR0 register we can control the count until an overflow occurs. The formula given below can be used to calculate the time it will take for the timer to overflow (or to generate an interrupt) given the oscillator period, value loaded into the timer and the prescaler value.

$$\text{Overflow time} = 4 \times T_{\underline{\text{OSC}}} \times \text{Prescaler} \times (256 - \text{TMR0}) \qquad (1.1)$$

where

Overflow time is in μs
$T_{\underline{\text{OSC}}}$ is the oscillator period in μs
Prescaler is the prescaler value chosen using OPTION_REG
TMR0 is the value loaded into TMR0 register

For example, assume that we are using a 4-MHz crystal, and the prescaler is chosen as 1:8 by setting bits PS2:PS0 to "010". Also assume that the value loaded into the timer register TMR0 is decimal 100. The overflow time is then given by

$$4\,\text{MHz clock has a period, } T = 1/f = 0.25\,\mu\text{s}$$

Using the above formula,

$$\text{Overflow time} = 4 \times 0.25 \times 8 \times (256 - 100) = 1248\,\mu\text{s}.$$

Thus, the timer will overflow after 1.248 μs and a timer interrupt will be generated if the timer interrupt and global interrupts are enabled.

What we normally need is to know what value to load into the TMR0 register for a required Overflow time. This can be calculated by modifying Eq. (1.1) as

$$\text{TMR0} = 256 - (\text{Overflow time})/(4 \times T_{\underline{\text{OSC}}} \times \text{Prescaler}) \qquad (1.2)$$

For example, suppose that we want an interrupt to be generated after 500 μs and the clock and the prescaler values are as before. The value to be loaded into the TMR0 register can be calculated using Eq. (1.2) as

$$\text{TMR0} = 256 - 500/(4 \times 0.25 \times 8) = 193.5$$

The nearest number we can load into the TMR0 register is 193.

Table 1.5 gives the values that should be loaded into the TMR0 register for different Overflow times. In this table a 4-MHz crystal is assumed and the table gives the result as the prescaler value is changed from 2 to 256.

Table 1.5: Required TMR0 Values for Different Overflow Times

Time to Overflow (μs)	Prescaler							
	2	4	8	16	32	64	128	256
100	206	231	243	250	253	254	–	–
200	156	206	231	243	250	253	254	–
300	106	181	218	237	246	251	253	255
400	56	156	206	231	243	250	253	254
500	6	131	193	224	240	248	252	254
600	–	106	181	218	237	16	251	253
700	–	81	168	212	234	245	250	253
800	–	56	156	206	231	243	250	253
1,000	–	6	131	193	225	240	248	252
5,000	–	–	–	–	100	178	77	236
10,000	–	–	–	–	–	100	178	217
20,000	–	–	–	–	–	–	100	178
30,000	–	–	–	–	–	–	–	139
40,000	–	–	–	–	–	–	–	100
50,000	–	–	–	–	–	–	–	60
60,000	–	–	–	–	–	–	–	21

1.4.3.5 TMR1

Although TMR0 is the basic timer found in nearly all PIC microcontrollers, some devices have several timers, such as TMR0, TMR1, and TMR2. Additional timers give added functionality to a microcontroller. In this section the operation of TMR1 will be described in detail.

TMR1 is a 16-bit timer, consisting of two 8-bit registers TMR1H and TMR1L. As shown in Fig. 1.11, a prescaler is used with TMR1 and the available prescaler values are only 1, 2, 4, and 8.

Register T1CON controls the operation of TMR1. The bit definition of this register is shown in Fig. 1.12. TMR1 can operate either as a timer or as a counter, selected by bit TMR1CS of T1CON. When operated in timer mode, TMR1 increments every oscillator frequency $f_{osc}/4$.

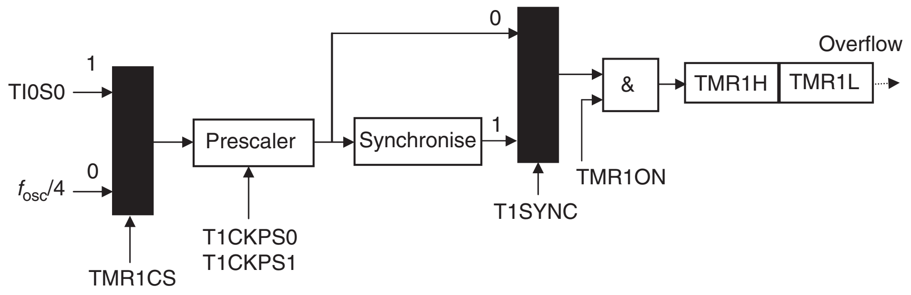

Figure 1.11: TMR1 Structure

7	6	5	4	3	2	1	0
–	–	TICKPS1	TICKPS0	TIOSCEN	TISYNC	TMR1CS	TMR1ON

Bit 7: Unused

Bit 6: Unused

Bit 5–4: 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: Timer1 Oscillator Enable Bit
1: Oscillator is enabled
0: Oscillator is disabled

Bit 2: 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 internal clock

Bit 1: Timer1 Clock Source Select Bit
1: External clock from pin TIOSO (on rising edge)
0: Internal clock ($f_{osc}/4$)

Bit 0: Timer1 On Bit
1: Enable Timer1
0: Stops Timer1

Figure 1.12: T1CON Bit Definitions

TMR1 can be enabled or disabled by setting or clearing control bit TMR1ON. TMR1 can count from 0 to 65,535 and it generates an overflow when changing from 65,535 to 0. A timer interrupt is generated if the TMR1 interrupt enable bit TMR1IE is enabled and also the global interrupts are enabled by register INTCON.

When TMR1 is operated in counter mode, it increments on every rising edge (from logic 0 to logic 1) of the clock input.

1.4.3.6 TMR2

TMR2 is an 8-bit timer with a prescaler and a postscaler and it has an 8-bit period register PR2. This timer is controlled by register T2CON whose bit definitions are given in Fig. 1.13. The prescaler options are 1, 4, and 16 only and are selected by T2CKPS1 and T2CKPS0 bits of T2CON. TMR2 increments from 0, until it matches PR2, and then resets to 0 on the next cycle. Then the cycle is repeated. TMR2 can be shut off by clearing TMR2ON of T2CON register to minimize power consumption.

7	6	5	4	3	2	1	0
–	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Bit 7: Unused

Bit 6-3: Timer2 Output Postscale Select Bits

0000	1:1 Postscale
0001	1:2 Postscale
0010	1:3 Postscale
....	
....	
1111	1:16 Postscale

Bit 2: Timer2 On Bit
- 1: Timer2 is On
- 0: Timer2 is Off

Bit 1-0: Timer2 Clock Prescale Select Bits

00	Prescaler is 1
01	Prescaler is 4
10	Prescaler is 16
11	Prescaler is 16

Figure 1.13: T2CON Bit Definitions

1.4.3.7 INTCON Register

This is the interrupt control register. This register is at address 0 and 8B (hexadecimal) of the microcontroller RAM and the bit definitions are given in Fig. 1.14. For example, to enable interrupts so that external interrupts from pin INT (RB0) can be accepted on a PIC16F84, the following bit pattern should be loaded into register INTCON:

1XX1XXXX

Similarly, to enable timer interrupts, bit 5 of INTCON must be set to 1.

7	6	5	4	3	2	1	0
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Bit 7: Global Interrupt Enable
    1: Enable all un-masked interrupts
    0: Disable all interrupts

Bit 6: EE Write Complete Interrupt
    1: Enable EE write complete interrupt
    0: Disable EE write complete interrupt

Bit 5: TMR0 Overflow Interrupt
    1: Enable TMR0 interrupt
    0: Disable TMR0 interrupt

Bit 4: INT External Interrupt
    1: Enable INT External Interrupt
    0: Disable INT External Interrupt

Bit 3: RB Port Change Interrupt
    1: Enable RB port change interrupt
    0: Disable RB port change interrupt

Bit 2: TMR0 Overflow Interrupt Flag
    1: TMR0 has overflowed
    0: TMR0 did not overflow

Bit 1: INT Interrupt Flag
    1: INT interrupt occurred
    0: INT interrupt did not occur

Bit 0: RB Port Change Interrupt Flag
    1: One or more of RB4-RB7 pins changed state
    0: None of RB4-RB7 changed state

Figure 1.14: INTCON Register Bit Definitions

1.4.3.8 A/D Converter Registers

The A/D converter is used to interface analog signals to the microcontroller. The A/D converts analog signals (e.g., voltage) into digital form so that they can be connected to a computer. A/D converter registers are used to control the A/D converter ports. On most PIC microcontrollers equipped with A/D, PORTA pins are used for analog input and these port pins are shared between digital and analog functions.

PIC16F876 includes five A/D converters. Similarly, PIC16F877 includes eight A/D converters. There is actually only one A/D converter, as shown in Fig. 1.15, and the inputs are multiplexed and they share the same converter. The width of the A/D converter can be 8 bits or 10 bits. Both PIC16F876 and PIC16F877 have 10-bit converters. PIC16F73 has 8-bit converters. An A/D converter requires a reference voltage to operate. This reference voltage is

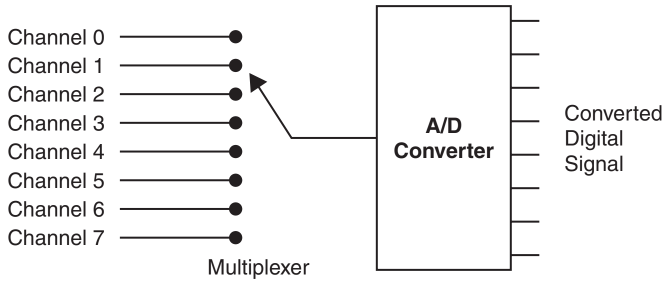

Figure 1.15: Multiplexed A/D Structure

chosen by programming the A/D converter registers and is typically 5 V. Thus, if we are using a 10-bit converter (1024 quantization levels), the resolution of our converter will be 5/1024 = 0.00488 V, or 4.88 mV; i.e., we can measure analog voltages with a resolution of 4.88 mV. For example, if the measured analog input voltage is 4.88 mV we get the 10-bit digital number "0000000001"; if the analog input voltage is $2 \times 4.88 = 9.76$ mV, the 10-bit converted number will be "0000000010"; if the analog input voltage is $3 \times 4.88 = 14.64$ mV, the converted number will be "0000000011"; and so on.

In a similar way, if the reference voltage is 5 V and we are using an 8-bit converter (256 quantization levels), the resolution of the converter will be 5/256 = 19.53 mV. For example, if the measured input voltage is 19.53 mV we get the 8-bit number "00000001"; if the analog input voltage is $2 \times 19.53 = 39.06$ mV we get the 8-bit number "00000010"; and so on.

The A/D converter is controlled by registers ADCON0 and ADCON1. The bit pattern of ADCON0 is shown in Fig. 1.16. ADCON0 is split into four parts; the first part consists of the highest two bits ADCS1 and ADCS0 and they are used to select the conversion clock. The internal RC oscillator or the external clock can be selected as the conversion clock as in the following table:

00	External clock/2
01	External clock/8
10	External clock/32
11	Internal RC clock

The second part of ADCON0 consists of the three bits CHS2, CHS1, and CHS0. These are the channel select bits, and they select which input pin is routed to the A/D converter. The selection is as follows:

CHS2:CHS1:CHS0

000	Channel 0
001	Channel 1
010	Channel 2

7	6	5	4	3	2	1	0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	–	ADON

Bit 7-6: A/D Converter Clock Select

00	$f_{osc}/2$
01	$f_{osc}/8$
10	$f_{osc}/32$
11	Internal RC oscillator

Bit 5-3: A/D Channel Select

000	Channel 0
001	Channel 1
010	Channel 2
011	Channel 3
100	Channel 4
101	Channel 5
110	Channel 6
111	Channel 7

Bit 2: GO/DONE Bit
- 1: Start conversion
- 0: A/D conversion is complete

Bit 1: Not used

Bit 0: ADON Bit
- 1: Turn ON A/D circuit
- 0: Turn OFF A/D circuit

Figure 1.16: ADCON0 Bit Definitions

011	Channel 3
100	Channel 4
101	Channel 5
110	Channel 6
111	Channel 7

The third part of ADCON0 is the single GO/DONE bit. This bit has two functions: first, by setting the bit it starts the A/D conversion. Second, the bit is cleared when the conversion is complete and this bit can be checked to see whether or not the conversion is complete.

The fourth part of ADCON0 is also a single bit, ADON, which is set to turn on the A/D converter circuitry.

ADRESH and ADRESL are the A/D converter result registers. ADRESL is the low byte and ADRESH is the upper 2 bits (if a 10-bit converter is used). We shall see how to configure the result of the conversion later.

ADCON1 is the second A/D control register. This register controls the format of converted data and mode of the PORTA inputs. The bit format of this register is shown in Fig. 1.17.

Bit 7 is called ADFM and when this bit is 0 the result of the A/D conversion is left justified; when it is 1, the result of the A/D conversion is right justified. If we have an 8-bit converter, we can clear ADFM and just read ADRESH to get the 8-bit converted data. If we have a 10-bit converter, we can set ADFM to 1 and the 8 bits of the result will be in ADRESL, and 2 bits of the result will be in the lower bit positions of ADRESH. The remaining 6 positions of ADRESH (bit 2 to bit 7) will be cleared to zero.

Bits PCFG0-3 control the mode of PORTA pins. As seen in Fig. 1.17, a PORTA pin can be programmed to be a digital pin or an analog pin. For example, if we set PCFG0-3 to "0110" then all PORTA pins will be digital I/O pins. PCFG0-3 bits can also be used to define the reference voltage for the A/D converter. As we shall see in the projects section of the book, the reference voltage Vref is usually set to be equal to the supply voltage (Vdd), and Vref is set to be equal to Vss. This causes the A/D reference voltage to be +5 V.

1.4.4 Oscillator Circuits

An oscillator circuit is used to provide a microcontroller with a clock. A clock is needed so that the microcontroller can execute a program.

7	6	5	4	3	2	1	0
ADFM	–	–	–	PCFG3	PCFG2	PCFG1	PCFG0

Bit 7: A/D Converter Result Format Select
  1: A/D converter output is right justified
  0: A/D converter output is left justified

Bit 6: Not used

Bit 5: Not used

Bit 4: Not used

Bit 3-0: Port Assignment and Reference Voltage Selection
  (see Figure 1.17)

PIC microcontrollers have built-in oscillator circuits and this oscillator can be operated in one of five modes.

- LP—Low-power crystal
- XT—Crystal/resonator
- HS—High-speed crystal/resonator
- RC resistor–capacitor
- No external components (only on some PIC microcontrollers).

PCFG3-PCFG0	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	Vref+	Vref−
0000	A	A	A	A	A	A	A	A	Vdd	Vss
0001	A	A	A	A	Vref+	A	A	A	RA3	Vss
0010	D	D	D	A	A	A	A	A	Vdd	Vss
0011	D	D	D	A	Vref+	A	A	A	RA3	Vss
0100	D	D	D	D	A	D	A	A	Vdd	Vss
0101	D	D	D	D	Vref+	D	A	A	RA3	Vss
0110	D	D	D	D	D	D	D	D	Vdd	Vss
0111	D	D	D	D	D	D	D	D	Vdd	Vss
1000	A	A	A	A	Vref+	Vref−	A	A	RA3	RA2
1001	D	D	A	A	A	A	A	A	Vdd	Vss
1010	D	D	A	A	Vref+	A	A	A	RA3	Vss
1011	D	D	A	A	Vref+	Vref−	A	A	RA3	RA2
1100	D	D	D	A	Vref+	Vref−	A	A	RA3	RA2
1101	D	D	D	D	Vref+	Vref−	A	A	RA3	RA2
1110	D	D	D	D	D	D	D	A	Vdd	Vss
1111	D	D	D	D	Vref+	Vref	D	A	RA3	RA2

Figure 1.17: ADCON1 Bit Definitions

In LP, XT, or HS modes, an external oscillator can be connected to the OSC1 input as shown in Fig. 1.18. This can be a crystal-based oscillator, or simple logic gates can be used to design an oscillator circuit.

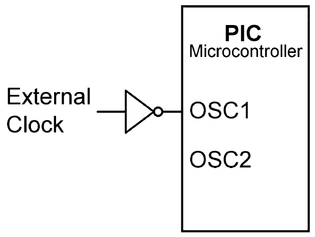

Figure 1.18: Using an External Oscillator

1.4.4.1 Crystal Operation

As shown in Fig. 1.19, in this mode of operation an external crystal and two capacitors are connected to the OSC1 and OSC2 inputs of the microcontroller. The capacitors should be chosen as in Table 1.6. For example, with a crystal frequency of 4 MHz, two 22-pF capacitors can be used.

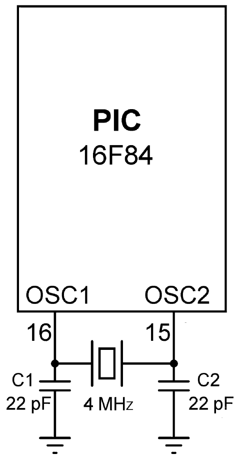

Figure 1.19: Crystal Oscillator Circuit

Table 1.6: Capacitor Selection for Crystal Operation

Mode	Frequency	C1, C2
LP	32 kHz	68–100 pF
LP	200 kHz	15–33 pF
XT	100 kHz	100–150 pF
XT	2 MHz	15–33 pF
XT	4 MHz	15–33 pF
HS	4 MHz	15–33 pF
HS	10 MHz	15–33 pF

1.4.4.2 Resonator Operation

Resonators are available from 4 to about 8 MHz. They are not as accurate as crystal-based oscillators. Resonators are usually 3-pin devices and the two pins at either side are connected to OSC1 and OSC2 inputs of the microcontroller. The middle pin is connected to the ground. Figure 1.20 shows how a resonator can be used in a PIC microcontroller circuit.

1.4.4.3 RC Oscillator

For applications where the timing accuracy is not important, we can connect an external resistor and a capacitor to the OSC1 input of the microcontroller as in Fig. 1.21. The oscillator frequency depends upon the values of the resistor and capacitor (see Table 1.7), the supply voltage, and to the temperature. For most applications, using a 5 K resistor with a 20-pF capacitor gives about 4 MHz and this may be acceptable in non-time-critical applications.

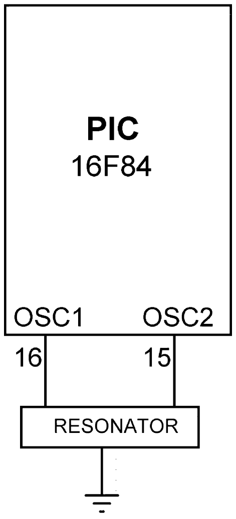

Figure 1.20: Resonator Oscillator Circuit

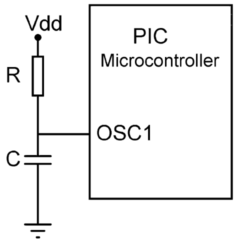

Figure 1.21: RC Oscillator Circuit

Table 1.7: RC Oscillator Component Selection

C	R	Frequency
20 pF	5 K	4.61 MHz
	10 K	2.66 MHz
	100 K	311 kHz
100 pF	5 K	1.34 MHz
	10 K	756 kHz
	100 K	82.8 kHz
300 pF	5 K	428 kHz
	10 K	243 kHz
	100 K	26.2 kHz

1.4.4.4 Internal Oscillator

Some PIC microcontrollers (e.g., PIC12C672 and PIC16F628) have built-in oscillator circuits and they do not require any external timing components. The built-in oscillator is usually set to operate at 4 MHz and is selected during the programming of the device. For example, the PIC16F62X series of PIC microcontrollers can be operated with an internal resistor–capacitor-based 4-MHz oscillator (called mode INTRC). Additionally, a single resistor can be connected to pin RA7 of the microcontroller to create a variable oscillator frequency (called ER mode). For example, in the PIC16F62X microcontroller OSC1 and OSC2 pins are shared with the RA7 and RA6 pins, respectively. The internal oscillator frequency can be set by connecting a resistor to pin RA7 as shown in Fig. 1.22. Depending on the value of this resistance, the oscillator frequency can be selected from 200 kHz to 10.4 MHz (see Table 1.8). When used in this mode, pin RA7 is not available as a digital I/O pin.

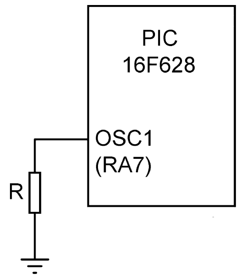

Figure 1.22: Changing the Internal Oscillator Frequency

Table 1.8: Resistor Value for the Internal Oscillator

Resistance	Frequency
0	10.4 MHz
1 K	10.0 MHz
10 K	7.4 MHz
20 K	5.3 MHz
47 K	3 MHz
100 K	1.6 MHz
220 K	800 kHz
470 K	300 kHz
1 M	200 kHz

The internal oscillator frequency of some microcontrollers (such as PIC16F630) can be calibrated so that more accurate timing pulses can be generated in time-critical applications (in serial communications, for example). In these microcontrollers an oscillator register called OSCCAL is used for the calibration of the oscillator frequency. A factory-calibrated oscillator constant is loaded into the last location of the memory. By copying this constant value into the oscillator register, we can have a more accurate 4-MHz clock frequency for our microcontroller. It is also possible to modify the OSCCAL register values in order to fine-tune the oscillator frequency.

As a practical example, the following PicBasic and PicBasic Pro statements can be used to copy the oscillator calibration constant from the last memory location into the OSCCAL register. These commands must be declared at the beginning of the programs.

```
DEFINE OSCCAL_1 K 1      For 1 K core-size microcontrollers
DEFINE OSCCAL_2 K 1      For 2 K core-size microcontrollers
```

Note that the oscillator constant can be erased during the erasing of the program memory. You should make a note of the value at the last location of the program memory before erasing the memory. If this value is known it can be loaded directly into the OSCCAL register at the beginning of our programs, as shown below (here it is assumed that the constant is $24).

```
OSCCAL = $24
```

1.4.5 Reset Circuit

Reset is used to put the microcontroller into a known state. Normally when a PIC microcontroller is reset, execution starts from address 0 of the program memory. This is where the first executable user program resides. The reset action also initializes various SFR registers inside the microcontroller.

PIC microcontrollers can be reset when one of the following conditions occur:

- Reset during power on (POR – Power On Reset)
- Reset by lowering MCLR input to logic 0
- Reset when the watchdog overflows.

As shown in Fig. 1.23, a PIC microcontroller is normally reset when power is applied to the chip and when the MCLR input is tied to the supply voltage through a 4.7 K resistor.

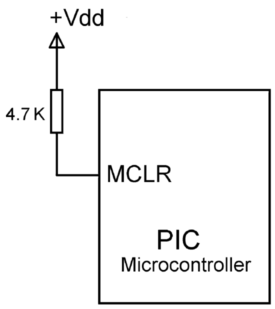

Figure 1.23: Using the Power on Reset

There are many applications where we want to reset the microcontroller, for instance by pressing an external button. The simplest circuit to achieve an external reset is shown in Fig. 1.24. In this circuit, the MCLR input is normally at logic 1 and the microcontroller is operating normally. When the reset button is pressed, this pin goes to logic 0 and the microcontroller is reset. When the reset button is released, the microcontroller starts executing from address 0 of the program memory.

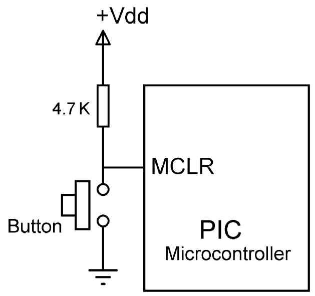

Figure 1.24: Using an External Reset Button

1.4.6 Interrupts

Interrupts are an important feature of all microcontrollers. An interrupt can either occur asynchronously or synchronously. Asynchronous interrupts are usually external events which interrupt the microcontroller and request service. For example, pin INT (RB0) of a PIC16F84 microcontroller is the external interrupt pin and this pin can be used to interrupt the microcontroller asynchronously; i.e., the interrupt can occur at any time independent of the program being executed inside the microcontroller. Synchronous interrupts are usually timer interrupts, such as the timer overflow generating an interrupt.

Depending on the model used, different PIC microcontrollers may have a different number of interrupt sources. For example, the PIC16F84 microcontroller has the following four sources of interrupts:

- External interrupt from INT (RB0) pin
- TMR0 interrupt caused by timer overflow
- External interrupt when the state of RB4, RB5, RB6, or RB7 pins change
- Termination of writing data to the EEPROM.

Interrupts are enabled and disabled by the INTCON register. Each interrupt source has two bits to control it. One enables interrupts, and the other one detects when an interrupt occurs. There is a common bit called GIE (see INTCON register bit definitions) which can be used to disable all sources of interrupts.

The INTCON control bits of various interrupt sources are

Interrupt Source	**Enabled by**	**Completion Status**
External interrupt from INT	INTE = 1	INTF = 1
TMR0 interrupt	T0IE = 1	T0IF = 1
RB4–RB7 state change	RBIE = 1	RBIF = 1
EEPROM write complete	EEIE = 1	–

Whenever an interrupt occurs, the microcontroller jumps to the ISR. On low-end microcontrollers (e.g., PIC16F84 or PIC16F628) all interrupt sources use address 4 in program memory as the start of the ISR. Because all interrupts use the same ISR address, we have to check the interrupt completion status to detect which interrupt has occurred when multiple interrupts are enabled.

The completion status has to be cleared to zero if we want the same interrupt source to be able to interrupt again.

Assuming that we wish to use the external interrupt (INT) input, and interrupts should be accepted on the low to high transition of the INT pin, the steps before and after an interrupt are summarized below.

- Set the direction of the external interrupt to be on rising edge by setting INTEDG = 1 in register OPTION_REG.
- Enable INT interrupts by setting INTE = 1 in register INTCON.
- Enable global interrupts by setting GIE = 1 in register INTCON.
- Carry out normal processing. When interrupt occurs, program will jump to the ISR.
- Carry out the required tasks in the ISR routine.
- At the end of the ISR, re-enable the INT interrupts by clearing INTF = 0.

1.4.7 The Configuration Word

PIC microcontrollers have a special register called the *Configuration Word*. This is a 14-bit register and is mapped in program memory 2007 (hexadecimal). This address is beyond the user program-memory space and cannot be directly accessed in a program. This register can be accessed during the programming of the microcontroller.

The configuration word stores the following information about a PIC microcontroller:

- Code protection bits: These bits are used to protect blocks of memory so that they cannot be read.
- Power-on timer enable bit.
- Watchdog (WDT) timer enable bit.

- Oscillator selection bits: The oscillator can be selected as XT, HS, LP, RC, or internal (if supported by the microcontroller).

For example, in a typical application we can have the following configuration word selection during the programming of the microcontroller:

- Code protection OFF
- XT oscillator selection
- WDT disabled
- Power-up timer enables.

1.4.8 I/O Interface

A PIC microcontroller port can source and sink 25 mA of current. When sourcing current, the current is flowing out of the port pin, and when sinking current, the current is flowing into the pin. When the pin is sourcing current, one pin of the load is connected to the microcontroller port and the other pin to the ground (see Fig. 1.25a). The load is then energized when the port output is at logic 1. When the pin is sinking current, one pin of the load is connected to the supply voltage and the other pin to the output of the port (see Fig. 1.25b). The load is then energized when the port output is at logic 0.

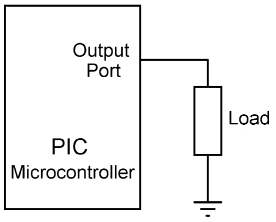

Figure 1.25a: Current Sourcing

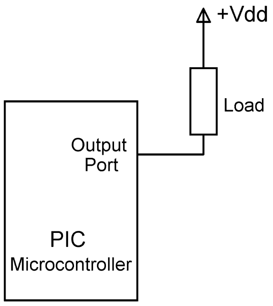

Figure 1.25b: Current Sinking

Some useful interface circuits are given in this section.

1.4.8.1 LED Interface

LEDs come in many different sizes, shapes, and colors. The brightness of an LED depends on the current through the device. Some small LEDs operate with only a few milliamperes of current, while standard size LEDs consume about 10 mA of current for normal brightness. Some very bright LEDs consume 15–20 mA of current. The voltage drop across an LED is about 2 V, but the voltage at the output of a microcontroller port is about 5 V when the port is at logic 1 level. As a result of this, it is not possible to connect an LED directly to a microcontroller output port. What is required is a resistor to limit the current in the circuit.

If the output voltage of the port is 5 V and the voltage drop across the LED is 2 V, we need to drop 3 V across the resistor. If we assume that the current through the LED is 10 mA, we can calculate the value of the required resistor as

$$R = \frac{5 - 2\,\text{V}}{10\,\text{mA}} = \frac{3\,\text{V}}{10\,\text{mA}} = 0.3\,\text{K}$$

The nearest physical resistor we can use is 330 Ω. Figure 1.26 shows how an LED can be connected to an output port pin in current source mode. In this circuit the LED will be ON when the port output is set to logic 1. Similarly, Fig. 1.27 shows how an LED can be connected to an output port pin in current sink mode. In this circuit the LED will be ON when the port output is at logic 0.

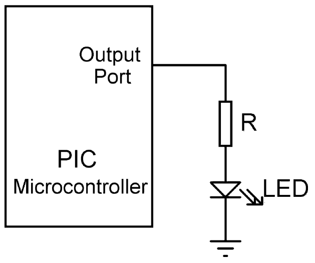

Figure 1.26: Connecting an LED in Current Source Mode

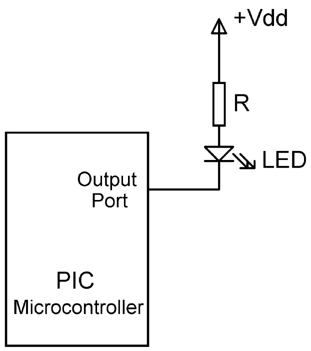

Figure 1.27: Connecting an LED in Current Sink Mode

1.4.8.2 Higher Current Load Interface

The circuits given in Figs 1.26 and 1.27 work fine for an LED, or for any other device whose current requirement is less than 25 mA. What do we do if we wish to operate a load with a higher current rating (e.g., a 12 V filament lamp)? The answer is that we have to use a switching device, such as a transistor or a relay.

Figure 1.28 shows how we can drive a small lamp from our port pin using a bipolar transistor. In this circuit, when the port output pin is at logic 1, current flows through the resistor and turns the transistor ON, effectively connecting the bottom end of the lamp to ground. It is important to realize that the positive supply to the lamp is not related to the PIC supply voltage and while the PIC is operating from 5 V, the lamp can be operated from a 12 V supply.The current capability depends upon the type of transistor used and several hundred milliamperes can be achieved with any type of small npn transistors. For higher currents, bipolar power transistors, or preferably MOSFET transistors, can be used.

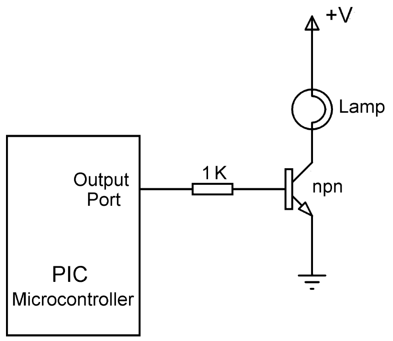

Figure 1.28: Driving a Lamp Using a Transistor

1.4.8.3 Relay Interface

When we want to switch inductive loads such as relays we have to use a diode in the circuit to prevent the transistor from being damaged (see Fig. 1.29). An inductive load can generate a

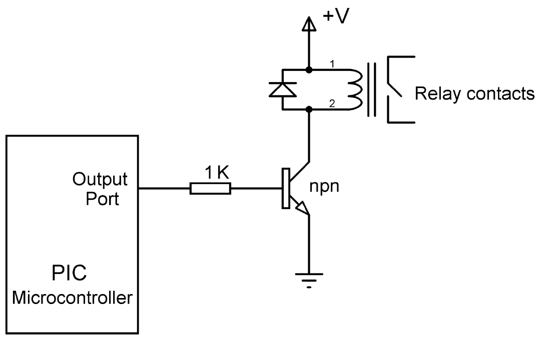

Figure 1.29: Driving an Inductive Load (e.g., a Relay)

back EMF which could easily damage a transistor. By connecting a diode in reverse bias mode this back EMF is dissipated without damaging the transistor.

Since we can drive a relay, we can connect any load to the relay outputs as long as we do not exceed the contact ratings of the relay. Figure 1.30 shows how a mains lamp can be operated from the microcontroller output port using a relay. The relay could also be operated using a MOSFET power transistor. In this circuit the mains lamp will turn ON when the output port of the microcontroller is a logic 1.

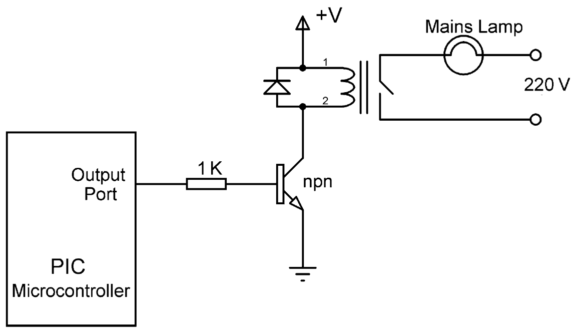

Figure 1.30: Driving a Mains Bulb Using a Relay

1.4.8.4 Button Input

One of the most common types of inputs is a button (a push-button switch) input where the user can change the state of an input pin by pressing a button. Basically, button input can use two different methods: active low and active high. As shown in Fig. 1.31, in active low implementation, the microcontroller input pin is connected to the supply voltage using a resistor (this is also called a pull-up resistor) and the button is connected between the port pin and ground. Normally the microcontroller input is pulled to logic 1 by the resistor. When the button is pressed, the input is forced to ground potential, which is logic 0. The change of state in the input pin can be determined by a program.

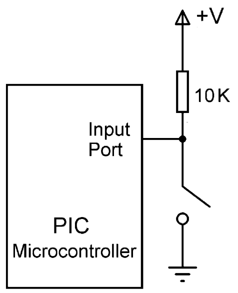

Figure 1.31: Active Low-Button Input

Some ports in PIC microcontrollers have internal pull-up resistors (e.g., PORTB) and these resistors can be enabled by clearing bit 7 (RBPU) of register INTCON to zero. When one of these port pins is used for button input, there is no need to use an external pull-up resistor and the button can simply be connected between the port pin and ground.

A button can also be connected in active high mode as shown in Fig. 1.32. In this configuration the button is connected between the supply voltage and the port pin. A resistor (this is also called a pull-down resistor) is connected between the port pin and ground. Normally, the port pin is at logic 0. When the button is pressed the port pin goes to the supply voltage, which is logic 1.

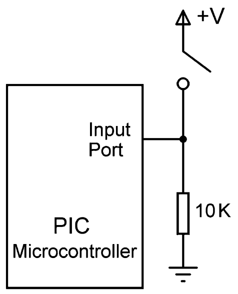

Figure 1.32: Active High-Button Input

One of the problems with mechanical switches is that, when a switch closes, its metal parts compress and relax and this causes the switch to open and close several times quickly. The problem is that the microcontroller can read the switch so fast that it can see the switch open and close during the bouncing of the metal parts and this can cause a wrong switch state to be read by the microcontroller. One way to eliminate this switch-bouncing problem is to delay reading the input after the switch state changes. For example, when we detect the switch is pressed, we may wait about 10 ms before we read the state of the switch.

In Figs 1.31 and 1.32, we have seen how simple buttons can be connected to a microcontroller port. It is also possible to connect a switching transistor to an input pin, the output of another IC, or simply the output of another PIC port pin. Figure 1.33 shows how a switching transistor can be connected as an input. In this circuit the transistor acts like an inverting switch. When the transistor input voltage is 0 V, the transistor is in OFF state and the port pin is at logic 1 level. When the transistor input voltage is 5 V the transistor turns ON and its collector-emitter voltage drops to 0 V, making the port pin logic 0. One nice thing about this circuit is that the transistor input voltage does not need to be 5 V to turn the transistor ON; it could easily be 9 or 12 V.

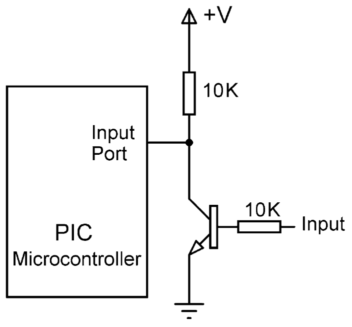

Figure 1.33: Transistor Input

The input ports of PIC microcontrollers are protected by internal diodes for over-voltage and under-voltage. Thus, the voltage on a pin can exceed the supply voltage, or it can go below the ground voltage, without causing any harm to the microcontroller. The RS232 serial communication lines operate with 12 V and we can usually connect these lines directly to the input ports using resistors without damaging the microcontroller.