國立成功大學
National Cheng Kung University
1931

Introduction to Microcontroller

Chapter 3
Microcontroller Memory and I/O

Chung-Ping Young (楊中平)

Introduction to Microcontroller Memory

- A memory unit is an integral part of any microcontroller, and its primary purpose is to hold instructions and data. The major design goal of on-chip memory inside the microcontroller is to allow it to operate at a speed close to that of the CPU.
- A microcontroller memory system can be divided into two groups: *CPU registers*; *primary or main memory*.

Introduction to Microcontroller Memory

- CPU registers are used to hold temporary results when a computation is in progress. Typical registers inside the CPU include the Accumulator, Program Counter, Stack Pointer, and Status Register.
- Primary or main memory is the storage area in which all programs are executed. The microcontroller can directly access only those items that are stored in main memory. Therefore, all programs must be in the main memory prior to execution.

Introduction to Microcontroller Memory

- The size of the main memory is usually much larger than the number of registers, and its operating speed is slower than that of processor registers.
    - The cost limits a microcontroller architect to include only a few registers in the CPU.

Main Memory

- The main memory (or simply, the memory) stores both instructions and data. For 8-bit microcontrollers, the memory is divided into a number of 8-bit units called *memory words*.
- An 8-bit unit of data is termed a *byte*. Therefore, for an 8-bit microcontroller, *memory word* and *memory byte* mean the same thing. For 16-bit microcontrollers, a word contains two bytes (16 bits).
- A memory word is identified in the memory by an address.

Main Memory

- The PIC18F is an 8-bit microcontroller, and can directly address a maximum of two Megabytes (2^{21}) of *program memory* space.
- The *data memory* address, on the other hand, is 12-bit wide. Hence, the PIC18F can directly address data memory of up to 4kbytes (2^{12}). This provides a maximum of 2^{12} = 4096 bytes of data memory addresses, ranging from 000 to FFF in hexadecimal.

Main Memory

- Large areas of data memory require an efficient addressing scheme to make rapid access to any address possible. For PIC18F, a RAM banking scheme is used.
- This divides the memory space into 16 contiguous banks (bank 0 through 15) of 256 bytes. Depending on the instruction, each location can be addressed directly by its full 12-bit address, or an 8-bit low-order address and a 4-bit Bank Pointer (also called Bank Select Register, BSR).

Main Memory

- There are some advantages of using memory banks. Once the Bank Select Register (4 bits in this case) is initialized with the bank number using instructions, fewer bits (8 bits in this case) are needed for data memory addresses.

Main Memory

- Figure 3.1 shows a simplified data memory layout of the PIC18F. The high 4 bits of an address specify the bank number.
- Consider address 0x105 of segment 1. The high 4 bits, 0001, of this address define the location as in bank 1, and the low 8 bits, 0x05, specify the particular address in bank 1.

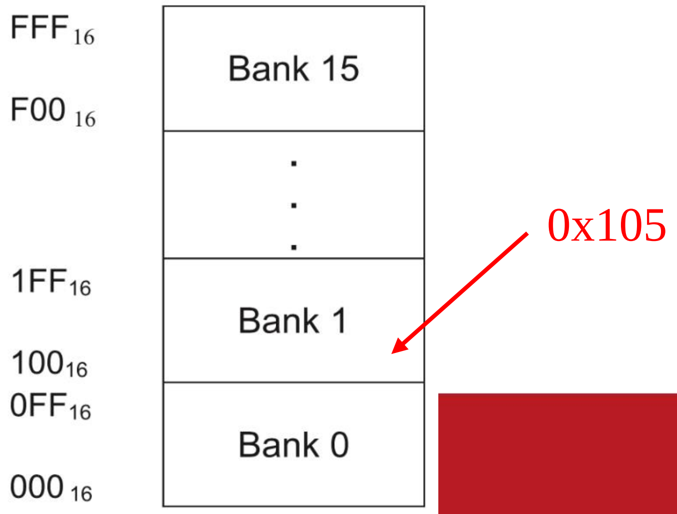


FIGURE 3.1 PIC18F data memory.

National Cheng Kung University

Main Memory

- Memories can be categorized into two main types: read-only memory (ROM) and random-access memory (RAM). ROMs and RAMs are then divided into a number of subcategories.

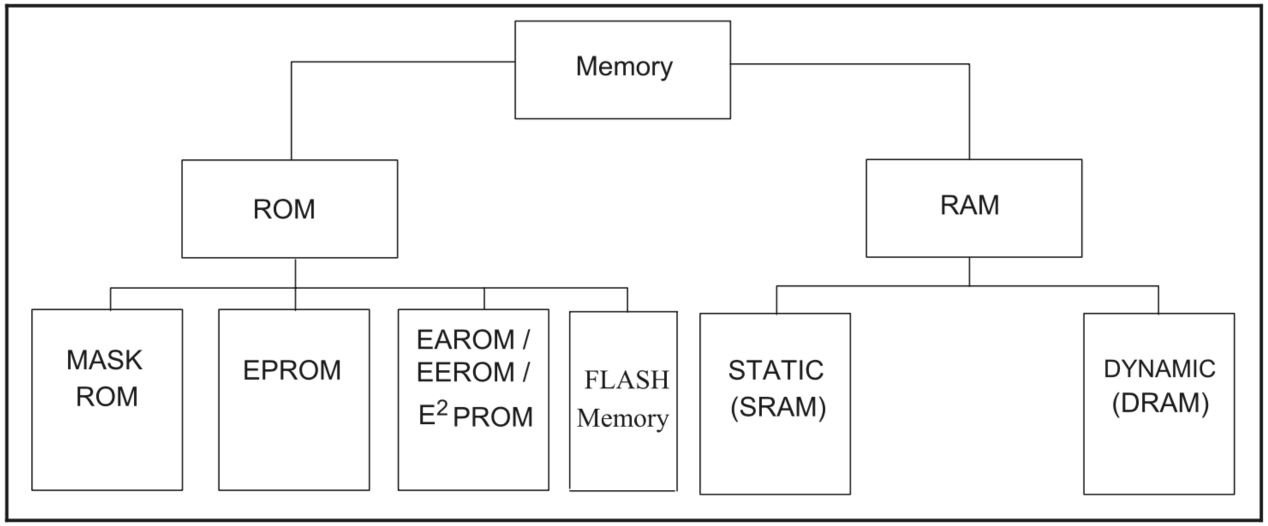


FIGURE 3.2 Summary of available semiconductor memories for microcontroller systems.

National Cheng Kung University 1931

Read-Only Memory

- ROMs (Read-Only memories) can only be read, so it is nonvolatile memory. CMOS technology is used to fabricate ROMs.

Read-Only Memory

- Mask ROMs are programmed by a masking operation performed on a chip during the manufacturing process. The contents of mask ROMs are permanent and cannot be changed by the user.
- EPROMs can be programmed, and their contents can also be altered by using special equipment, called an EPROM programmer.

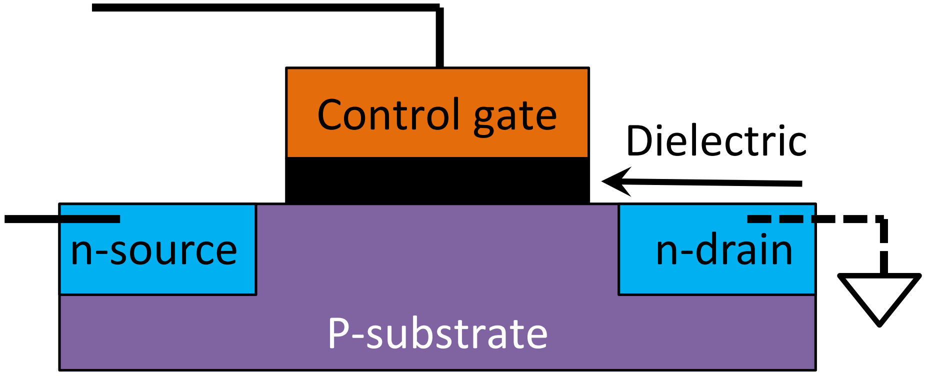
Read-Only Memory

- EEPROMs can be programmed without removing the memory from the ROM’s sockets. These memories are also called read-mostly memories (RMMs), because they have much slower write times than read times. Therefore, these memories are usually suited for operations when mostly reading rather than writing is performed.

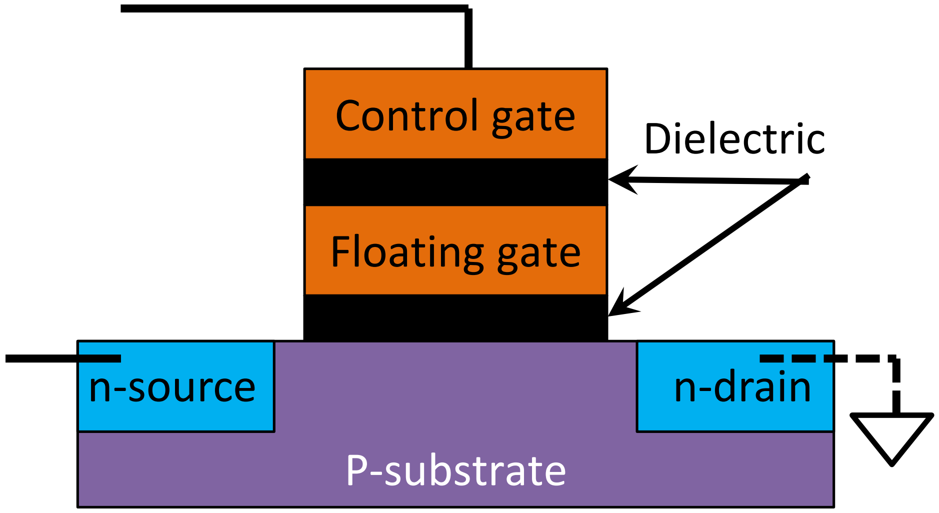
Flash Memory

- Flash memory (nonvolatile) is designed using a combination of EPROM and E2PROM technologies. Flash memory can be reprogrammed electrically while embedded on the board.
- One can change multiple bytes at a time.
- Flash memory is typically used in cellular phones and digital cameras. Note that the PIC18F uses flash memory as its program memory.

The Key Device: Floating-Gate Transistor



MOSFET



Floating-Gate Transistor

MOSFET: metal–oxide–semiconductor field-effect transistor

Random-Access Memory

- There are two types of RAM: static RAM (SRAM), and dynamic RAM (DRAM).
- Static RAM stores data in flip-flops. Therefore, this memory does not need to be refreshed. RAMs are volatile unless backed up by battery. The PIC18F uses SRAM for its data memory.

正反器（英語：Flip-flop, FF）是一種具有兩種穩態的用於儲存的元件，可記錄二進位數位訊號「1」和「0」。

Random-Access Memory

- Dynamic RAM stores data in capacitors (電容器). It can hold data for a few milliseconds. Hence, dynamic RAMs are refreshed typically by using external refresh circuitry. Dynamic RAMs (DRAMs) are used in applications requiring large memory. DRAMs have higher densities than static RAMs (SRAMs).

Random-Access Memory

- Typical examples of DRAMs are the 4464 (64k × 4-bit), 44256 (256k × 4-bit), and 41000 (1M × 1-bit). DRAMs are inexpensive, occupy less space, and dissipate less power than SRAMs. Two enhanced versions of DRAM are EDO DRAM (extended data output DRAM) and SDRAM (synchronous DRAM).

READ and WRITE Timing Diagrams

- To fetch an instruction, when the clock signal goes to HIGH, the CPU places the contents of the program counter on the address bus via address pins A0–A15 on the chip. Note that since each of lines A0–A15 can be either HIGH or LOW, both transitions are shown for the address in Figure 3.3.

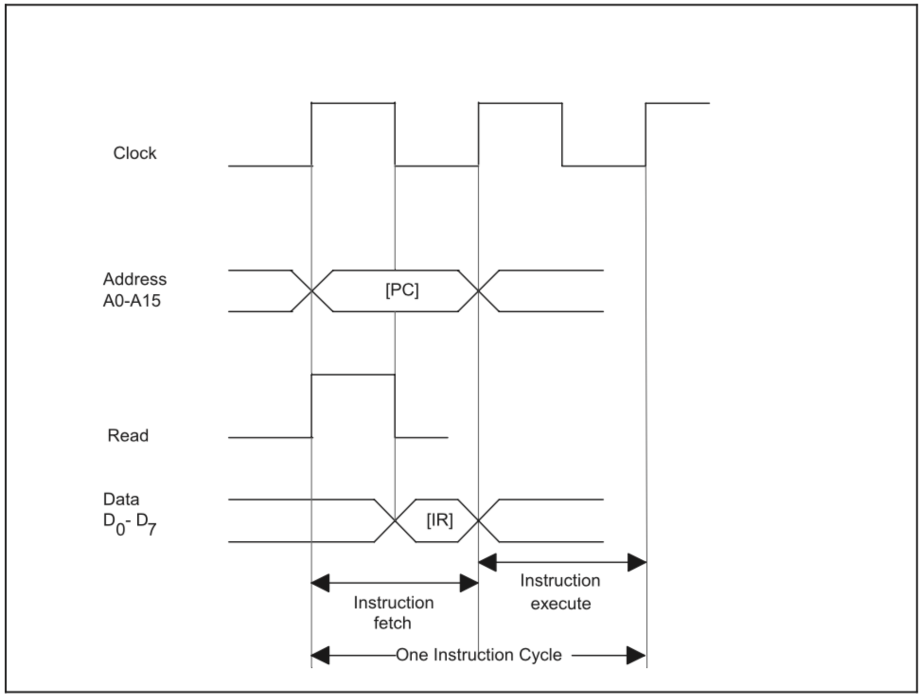


FIGURE 3.3 Typical instruction fetch timing diagram for an 8-bit microprocessor.

READ and WRITE Timing Diagrams

- The instruction fetch is basically a memory READ operation. Therefore, the CPU raises the signal on the READ pin to HIGH. As soon as the clock goes to LOW, the logic external to the CPU gets the contents of the memory location addressed by A0–A15 and places them on the data bus D0–D7. The CPU then takes the data and stores it in the instruction register (IR) so that it gets interpreted as an instruction. This is called *instruction fetch*.

READ and WRITE Timing Diagrams

- Memory READ is basically loading the contents of a memory location of the main ROM/RAM into an internal register of the CPU. The address of the location is provided by the contents of the memory address register (MAR).

READ and WRITE Timing Diagrams

- The READ timing diagram of Figure 3.4

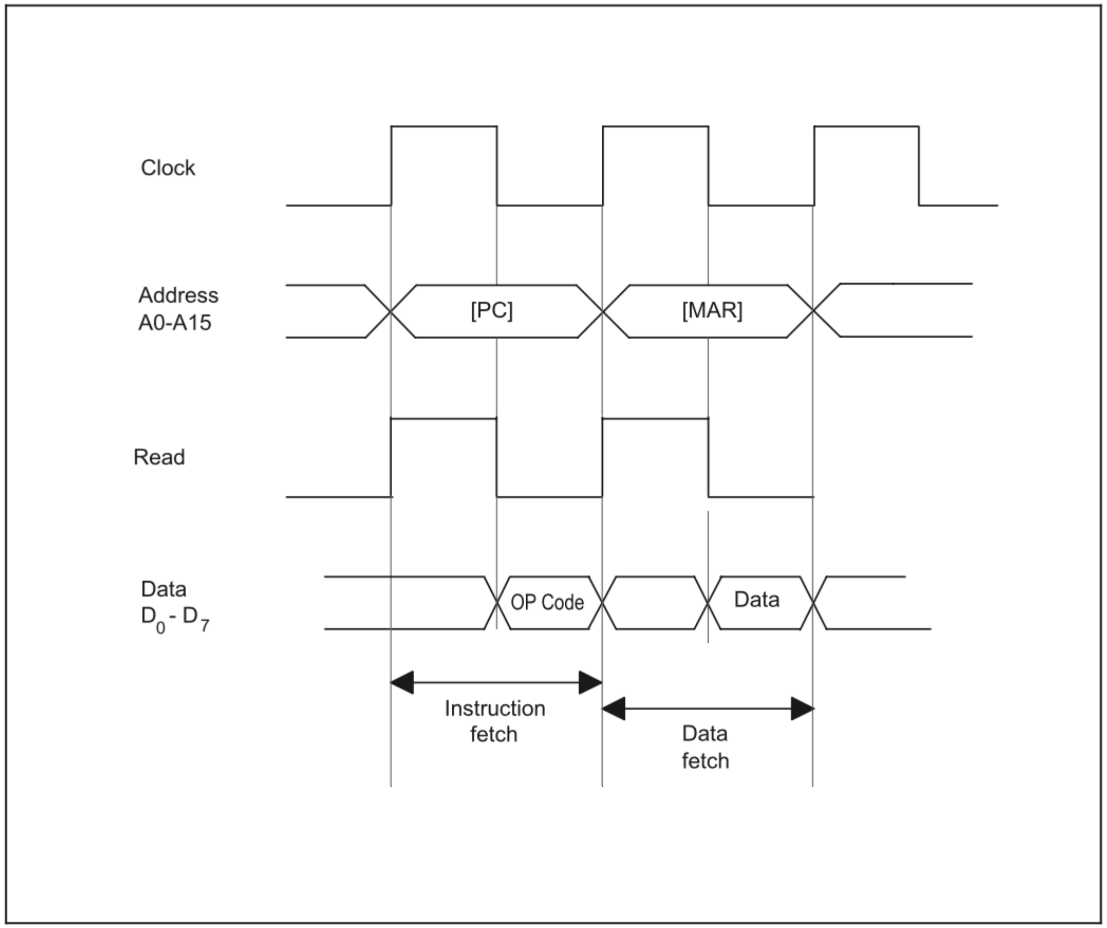


FIGURE 3.4 Typical memory READ timing diagram.

READ and WRITE Timing Diagrams

1. The CPU performs the instruction fetch cycle to READ the op-code.

2. The CPU interprets the op-code as a memory READ operation.

3. When the clock pin signal goes HIGH, the CPU places the contents of the memory address register on the address pins A0–A15 of the chip.

4. At the same time, the CPU raises the READ pin signal to HIGH.

5. The logic external to the CPU gets the contents of the location in the main ROM/RAM addressed by the MAR and places it on the data bus.

6. Finally, the CPU gets this data from the data bus via pins D0 – D7 and stores it in an internal register.

READ and WRITE Timing Diagrams

- Memory WRITE is basically storing the contents of an internal register of the CPU into a memory location of the main RAM. The contents of the memory address register provide the address of the location where data is to be stored.

READ and WRITE Timing Diagrams

- Figure 3.5 shows a typical WRITE timing diagram.

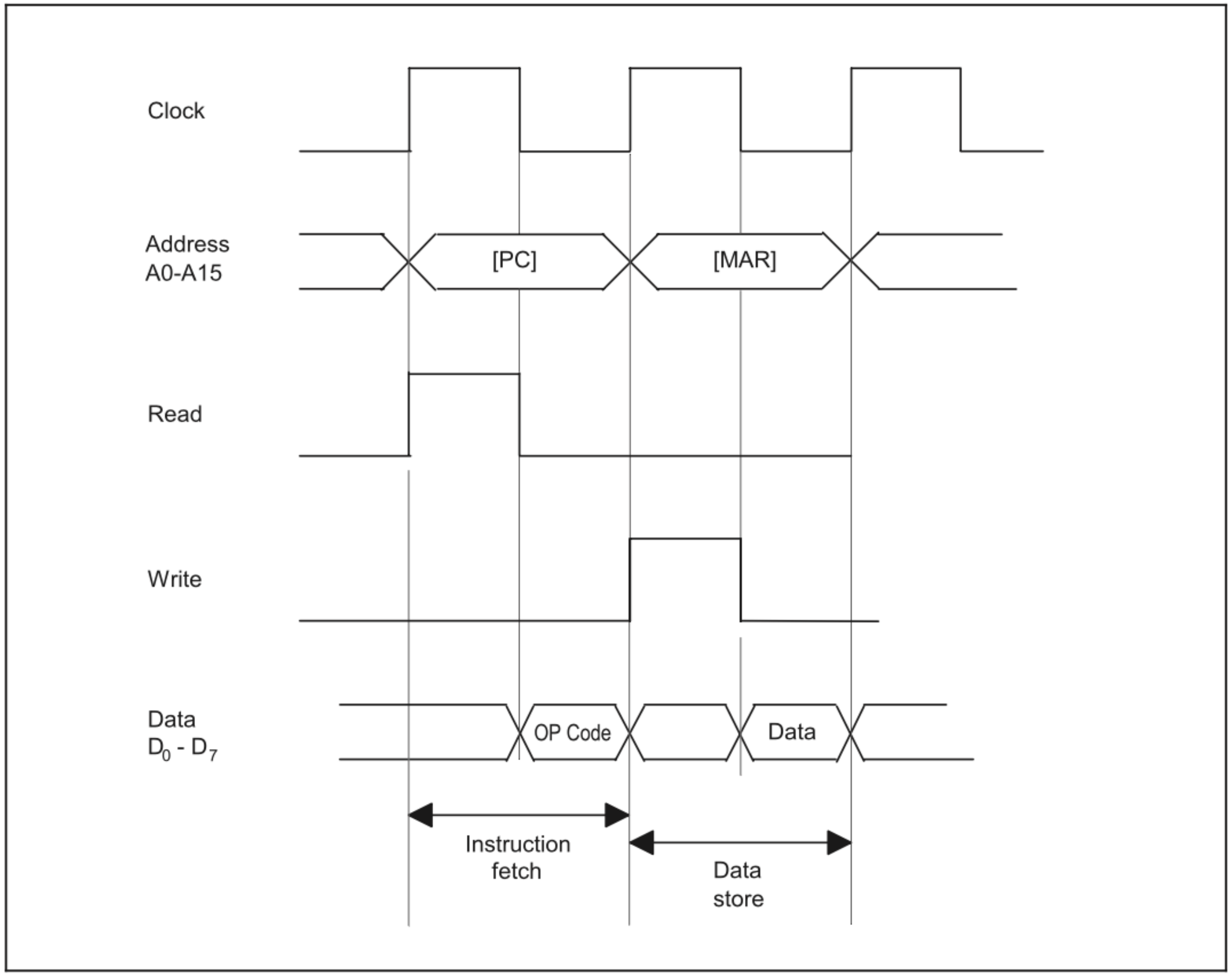


FIGURE 3.5 Typical memory WRITE timing diagram.

READ and WRITE Timing Diagrams

- The CPU fetches the instruction code as before.
- The CPU interprets the instruction code as a memory WRITE instruction and then proceeds to perform the DATA STORE cycle.
- When the clock pin signal goes HIGH, the CPU places the contents of the memory address register on the address pins A0–A15 of the chip.
- At the same time, the CPU raises the WRITE pin signal to HIGH.
- The CPU places data to be stored from the contents of an internal register onto data pins D0–D7.
- The logic external to the CPU stores the data from the register into a RAM location addressed by the memory address register.

Microcontroller Input/Output (I/O)

- There are two ways of transferring data between a microcontroller and I/O devices. These are *programmed I/O*, and *interrupt I/O*.
- Using programmed I/O, the CPU executes a program to perform all data transfers between the CPU and the external device. The main characteristic is that the external device carries out the functions dictated by the program contained in the microcontroller memory.

Microcontroller Input/Output (I/O)

- In interrupt I/O, an external device can force the CPU to stop executing the current program temporarily so that it can execute another program known as an interrupt service routine. This routine satisfies the needs of the external device.
- After completing this program, a return from interrupt instruction can be executed at the end of the service routine to return control at the right place in the main program.

Microcontroller Input/Output (I/O)

- The interrupt procedure is similar in concept to the procedure associated with subroutine CALL and RETURN instructions.
- The subroutine CALL instruction pushes the current contents of the program counter onto the stack. The RETURN instruction placed at the end of the subroutine pops the previously pushed program counter, and returns control to the main program.

Microcontroller Input/Output (I/O)

- The interrupt is initiated externally via hardware or internally via occurrence of events. Once the interrupt is recognized, the microcontroller normally pushes the Program Counter (PC) and the Status Register (SR) onto the stack, and automatically branches to an address predefined by the manufacturer.

Microcontroller Input/Output (I/O)

- The user writes a program called “interrupt service routine” at this address. This program is similar to the subroutine. A “Return from Interrupt” instruction placed by the user at the end of the interrupt service routine will pop the previously pushed PC and SR, and will return control to the main program at the proper location.

Programmed I/O

- A microcontroller communicates with an external device via one or more registers called *I/O ports* using programmed I/O. 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 another register usually called the *Data Direction Register* (*DDR*).

Programmed I/O

- Each bit in the port can be set up as an input or output, normally by writing a 0 or a 1 in the corresponding bit of the DDR.
- The PIC18F microcontroller makes an I/O port bit an input by writing a ‘1’ in the corresponding bit in DDR. Writing a ‘0’ in a particular bit in DDR will configure the corresponding bit in the port as an output.

Programmed I/O

- For example, if an 8-bit DDR in the PIC18F contains 0xCB, the corresponding port is defined as shown in Figure 3.15.
- 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 at bits 2, 4, and 5 through a proper interface.

Programmed I/O

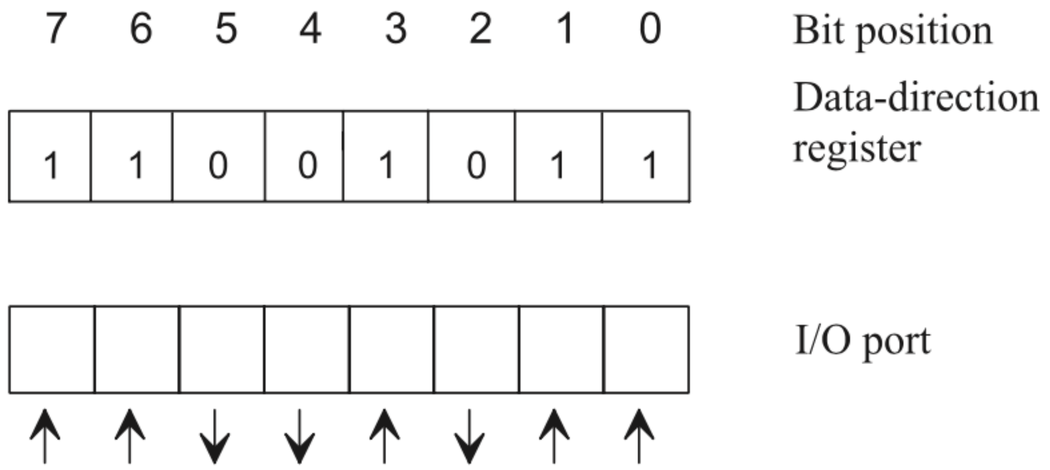


FIGURE 3.15 I/O port with the corresponding data-direction register.

Programmed I/O

- Similarly, the microcontroller can input the status of external devices, such as switches, through bits 0, 1, 3, 6, and 7.
- While receiving input data from an I/O port, however, the microcontroller places a value, probably 0, at the bits configured as outputs and the program must interpret them as “don’t cares.”

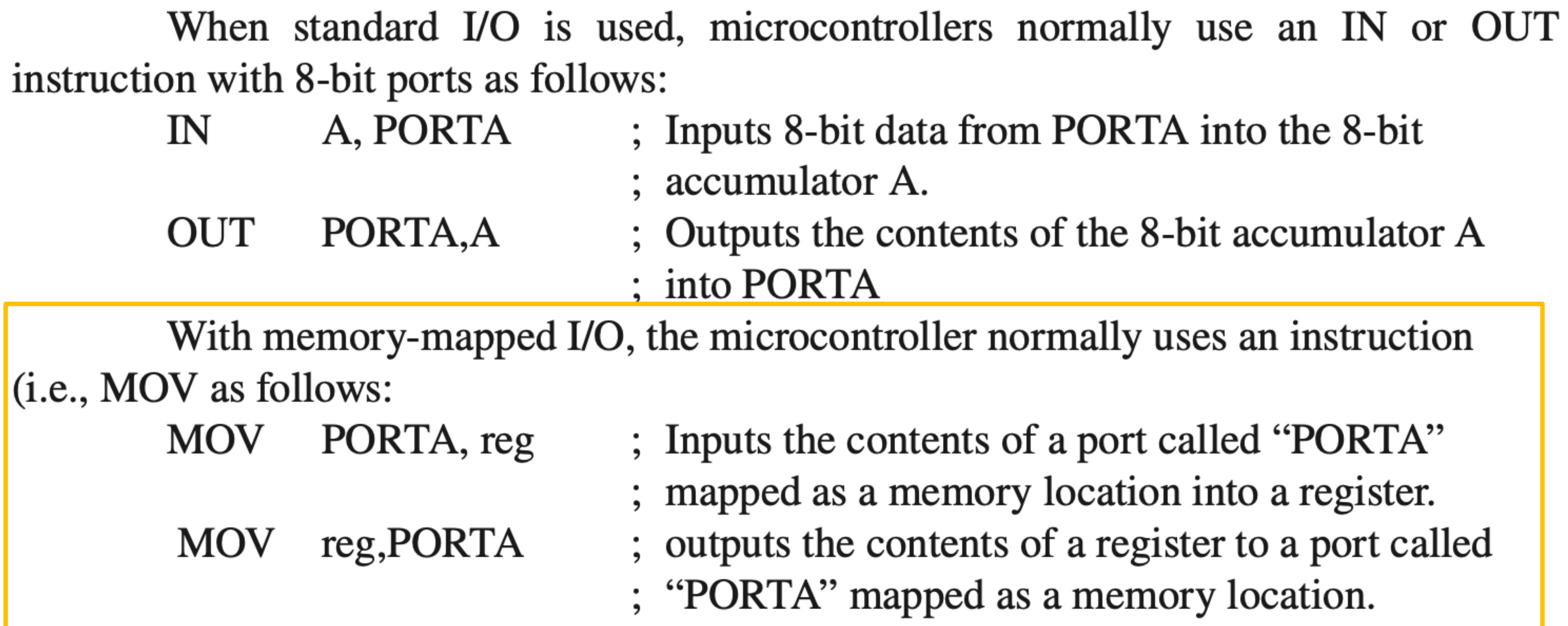
Programmed I/O

- 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 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.
- 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.

Programmed I/O

- Intel microcontrollers such as the 8051 uses standard 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 main memory addresses which may not exist physically, but are used by the memory-oriented instructions, such as MOVE, to generate the necessary control signals to perform I/O. The PIC18F uses memory-mapped I/O.

Programmed I/O



When standard I/O is used, microcontrollers normally use an IN or OUT instruction with 8-bit ports as follows:

```
IN      A, PORTA      ; Inputs 8-bit data from PORTA into the 8-bit
                      ; accumulator A.
OUT     PORTA,A       ; Outputs the contents of the 8-bit accumulator A
                      ; into PORTA
```

With memory-mapped I/O, the microcontroller normally uses an instruction (i.e., MOV as follows:

```
MOV     PORTA, reg    ; Inputs the contents of a port called "PORTA"
                      ; mapped as a memory location into a register.
 MOV    reg,PORTA     ; outputs the contents of a register to a port called
                      ; "PORTA" mapped as a memory location.
```

Uncond. and Cond. Programmed I/O

- There are typically two ways in which programmed I/O can be utilized: *unconditional I/O* and *conditional I/O*.
- The microcontroller can send data to an external device at any time using *unconditional Programmed I/O*. The external device must always be ready for data transfer.
- A typical example is that of a microcontroller outputting a 7-bit code through an I/O port to drive a seven-segment display connected to this port.

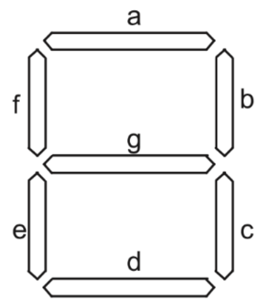


FIGURE 3.14 A seven-segment display

Uncond. and Cond. Programmed I/O

- In conditional Programmed I/O, the microcontroller waits for a particular condition to occur, and then outputs data to an external device based on the condition. *Conditional Programmed I/O* is sometimes called *Polled I/O*.

Uncond. and Cond. Programmed I/O

- Suppose that a comparator is connected to bit 0 of Port C, and an LED is connected to bit 1 of Port D of the PIC18F4321 microcontroller.
- It is desired to turn the LED ON when the comparator output becomes HIGH (Vx > Vy). The microcontroller needs to wait in a loop until the condition "Vx > Vy" occurs. The microcontroller will send a HIGH to bit 1 of Port D as soon as the condition occurs.

Uncond. and Cond. Programmed I/O

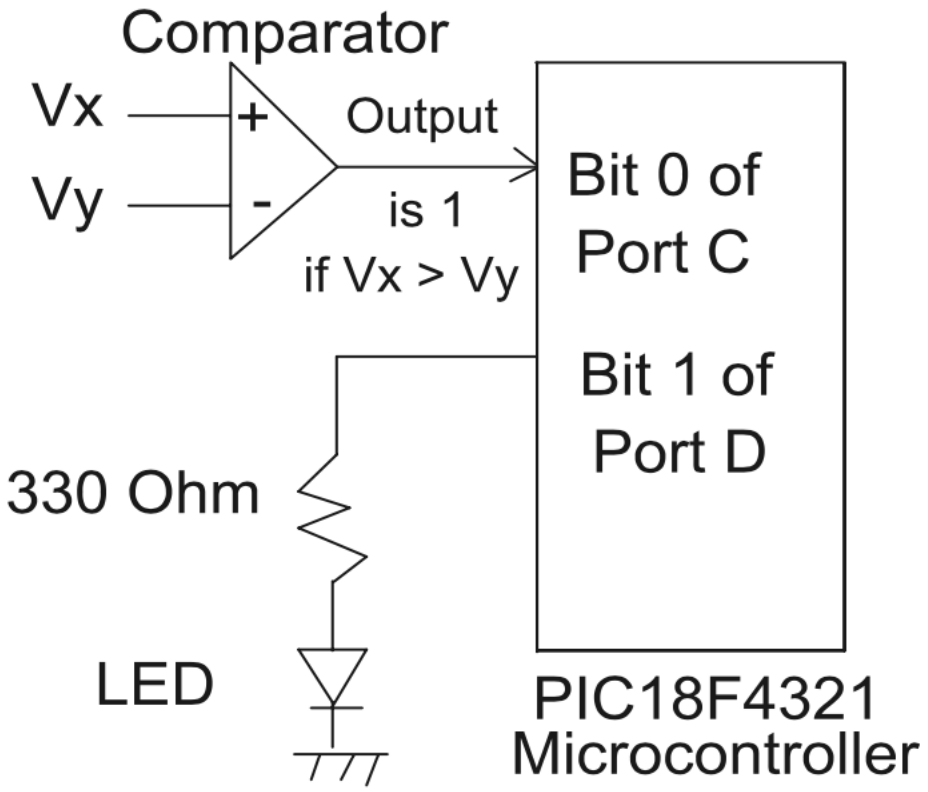


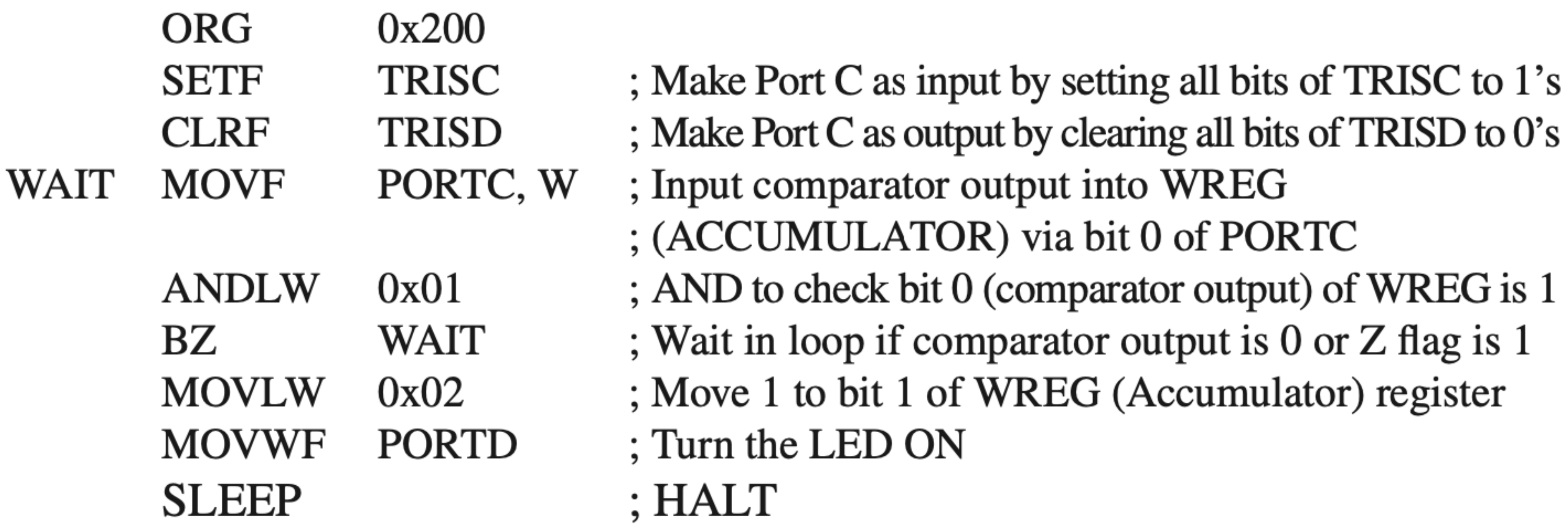
FIGURE 3.16 Example illustrating conditional or polled I/O

Uncond. and Cond. Programmed I/O

- Note that TRISC is the Data Direction Register (DDR) for Port C and TRISD is the Data Direction Register (DDR) for Port D.
- A ‘1’ in a particular position will make the corresponding bit in each of these ports as an input while a ‘0’ will make it an output.

Uncond. and Cond. Programmed I/O

- The following assembly language program starting at address 0x200 for the PIC18F4321 microcontroller



```
        ORG     0x200
        SETF    TRISC       ; Make Port C as input by setting all bits of TRISC to 1's
        CLRF    TRISD       ; Make Port C as output by clearing all bits of TRISD to 0's
WAIT    MOVF    PORTC, W    ; Input comparator output into WREG
                            ; (ACCUMULATOR) via bit 0 of PORTC
        ANDLW   0x01        ; AND to check bit 0 (comparator output) of WREG is 1
        BZ      WAIT        ; Wait in loop if comparator output is 0 or Z flag is 1
        MOVLW   0x02        ; Move 1 to bit 1 of WREG (Accumulator) register
        MOVWF   PORTD       ; Turn the LED ON
        SLEEP               ; HALT
```

ANDLW	**AND Literal with W**
BZ	**Branch if Zero**

Uncond. and Cond. Programmed I/O

- The “SETF TRISC” sets all bits in TRISC (DDR for PORTC) to 1’s and thus, configures bit 0 of PORTC as an input bit.
- The “CLRF TRISD” clears all bits in TRISD (DDR for PORTD) to 0’s and configures bit 1 of PORTD as an output bit.
- The “MOVF PORTC, W” instruction moves (inputs) the contents of PORTC into the WREG register. Thus, the comparator output connected to bit 0 of PORTC is inputted into bit 0 of the WREG register.

Uncond. and Cond. Programmed I/O

- The “ANDLW 0x01 logically ANDs the contents of WREG with 0x01, and stores the result in WREG. The contents of WREG will be zero (Z=1) if the comparator output at bit 0 of PORTC is 0; the contents of WREG will be one (Z = 0) if the comparator output is 1.

Uncond. and Cond. Programmed I/O

- The “BZ WAIT” instruction checks the Z flag. If the Z flag is 1 (comparator output is 0), the program branches back to WAIT , and stays in the loop. As soon as the comparator output is 1 (Z = 0), the “MOVLW 0x02” moves 0x02 into WREG, and thus, the bit 1 of WREG is a ‘1’.
- The MOVWF PORTD” instruction moves the contents of WREG to PORTD. Thus, a ‘1’ is outputted to bit 1 of PORTD, and the LED is turned ON.

Uncond. and Cond. Programmed I/O

- Note that in the program, the PIC18F4321 has to wait in a loop indefinitely for the comparator output to become one (Vx > Vy). This is called “Conditional” or “Polled I/O”, and is obviously inefficient because of the wait loop.

Interrupt I/O

- A disadvantage of conditional programmed I/O is that the CPU needs to check the status bit (output of the Comparator) by waiting in a loop.
- This type of I/O transfer is dependent on the occurrence of the external condition. This waiting may slow down the CPU’s ability to process other data.

Interrupt I/O

- Interrupt I/O is a device-initiated I/O transfer. The external device is connected to a pin called the interrupt (INT) pin on the microcontroller chip. When the device needs an I/O transfer with the microcontroller, it activates its interrupt pin.
- The microcontroller usually completes the current instruction and saves the contents of the current program counter and the status register onto the stack.

Interrupt I/O

- The microcontroller then loads an address into the program counter automatically to branch to a subroutine-like program called the *interrupt service routine*. This program is written by the user.
- The last instruction of the service routine is a RETURN. The RETURN from interrupt instruction typically restores the program counter and the status register with the information saved in the stack before going to the service routine.

Interrupt I/O

- Figure 3.17 provides a simple example for illustrating the concept of interrupt I/O. This is the same example used to illustrate polled I/O of Figure 3.16 except that the comparator output is connected to the microcontroller's interrupt (INT) pin instead of bit 0 of Port C.

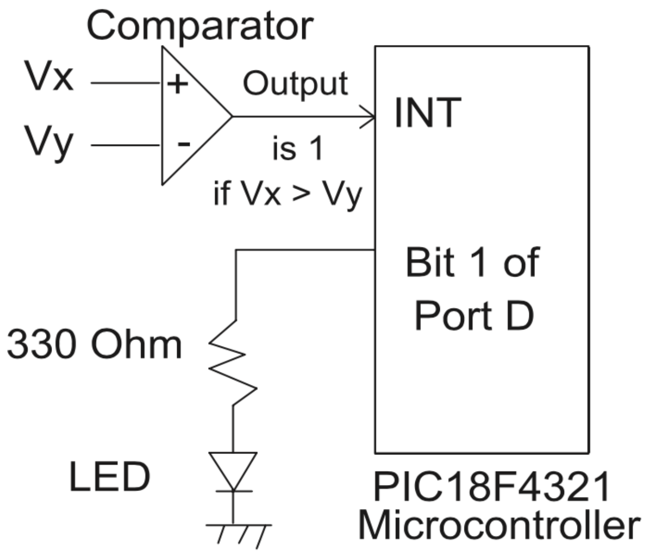


FIGURE 3.17 Example illustrating interrupt I/O

Interrupt I/O

Assume that the PIC18F 4321 microcontroller is executing the following main program:

```
          ORG       0x200
          SETF      TRISC    ; Make Port C as input by setting all bits of TRISC to 1's
          CLRF      TRISD    ; Make Port C as output by clearing all bits of TRISD to 0's
          MOVLW     0x15
          MOVWF     STKPTR   ; Initialize STKPTR to 0x15
          MOVLW     3        ;  Move  3 into WREG register
BEGIN     MOVWF     0x30     ; Move WREG into 0x30
          -
          -
```

Interrupt I/O

- Since interrupt I/O uses stack to save the return address, the stack pointer should be initialized in the main program. The PIC18F4321 then continues with execution of the “MOVLW 3” instruction.
- Suppose that during execution of the “MOVLW 3” instruction, the output of the comparator becomes HIGH, indicating that Vx is greater than Vy. This drives the INT signal to HIGH, interrupting the microcontroller

Interrupt I/O

- The microcontroller completes execution of the current instruction, “MOVLW 3”. It then saves the current contents of the program counter (address BEGIN) automatically and executes a subroutine-like program called the service routine. This program is usually written by the user.
- The microcontroller manufacturer normally specifies the starting address of the service routine. This address is 0x000008 in the PIC18F.

Interrupt I/O

- The user writes a service routine at this address to turn the LED ON, and then returns to the main program as follows:

```
ORG       0x000008    ; Starting address of the service routine
MOVLW     0x02        ; Move 1 to bit 1 of WREG (Accumulator) register
MOVWF     PORTD       ; Turn the LED ON
RETFIE                ; Restore PC and SR, and return from interrupt
```

Interrupt I/O

- Using the MOVLW and MOVWF instructions, the microcontroller turns the LED ON. The return instruction RETFIE, at the end of the service routine, loads the address BEGIN from the stack into the program counter. The microcontroller executes the “MOVWF 0x30” instruction at the address BEGIN and continues with the main program.

Interrupt I/O

- **Input Types.** There are typically two types of interrupts: external interrupts and internal interrupts.
- External interrupts are initiated through a microcontroller’s interrupt pins by external devices. External interrupts can be divided further into two types: maskable and nonmaskable.

Interrupt I/O

- The nonmaskable interrupt cannot be enabled or disabled by instructions, whereas a microcontroller’s instruction set typically contains instructions to enable or disable maskable interrupt. A nonmaskable interrupt has a higher priority than a maskable interrupt.
- If maskable and nonmaskable interrupts are activated at the same time, the processor will service the nonmaskable interrupt first.

Interrupt I/O

- A nonmaskable interrupt is typically used as a power failure interrupt. Few milliseconds before power off, the power-failure-sensing circuitry can interrupt the processor. The interrupt service routine can be written to store critical data in nonvolatile memory such as battery-backed CMOS RAM, and the interrupted program can continue without any loss of data when the power returns.
- The PIC18F does not have any nonmaskable interrupts.

Interrupt I/O

- Internal interrupts are maskable, and can be disabled by instructions. They are activated internally by conditions such as completion of A/D conversion, Timer interrupt, or interrupt due to serial I/O.
- Some microcontrollers include software interrupt instructions. When one of these instructions is executed, the microcontroller is interrupted and serviced similarly to external or internal interrupts.

Interrupt I/O

- The PIC18F provides external maskable interrupts only. The PIC18F does not have any external nonmaskable interrupts.
- However, the PIC18F provides internal interrupts. The internal interrupts are activated internally by conditions such as timer interrupts, completion of analog to digital conversion, and serial I/O.

Interrupt I/O

- **Interrupt Address Vector.** The technique used to find the starting address of the service routine (commonly known as the interrupt address vector) varies from one processor to another. The microcontroller manufacturers typically define the fixed starting address for each interrupt.

Interrupt I/O

- **Saving the Microcontroller Registers.** When a microcontroller is interrupted, it normally saves the program counter (PC) and the status register (SR) onto the stack.
- The user should know the specific registers the microcontroller saves prior to executing the service routine. This will allow the user to use the appropriate return instruction at the end of the service routine to restore the original conditions upon return to the main program.