The University of Texas at Arlington

Lecture 5
PIC I/O

CSE@UTA

CSE 3442/5442

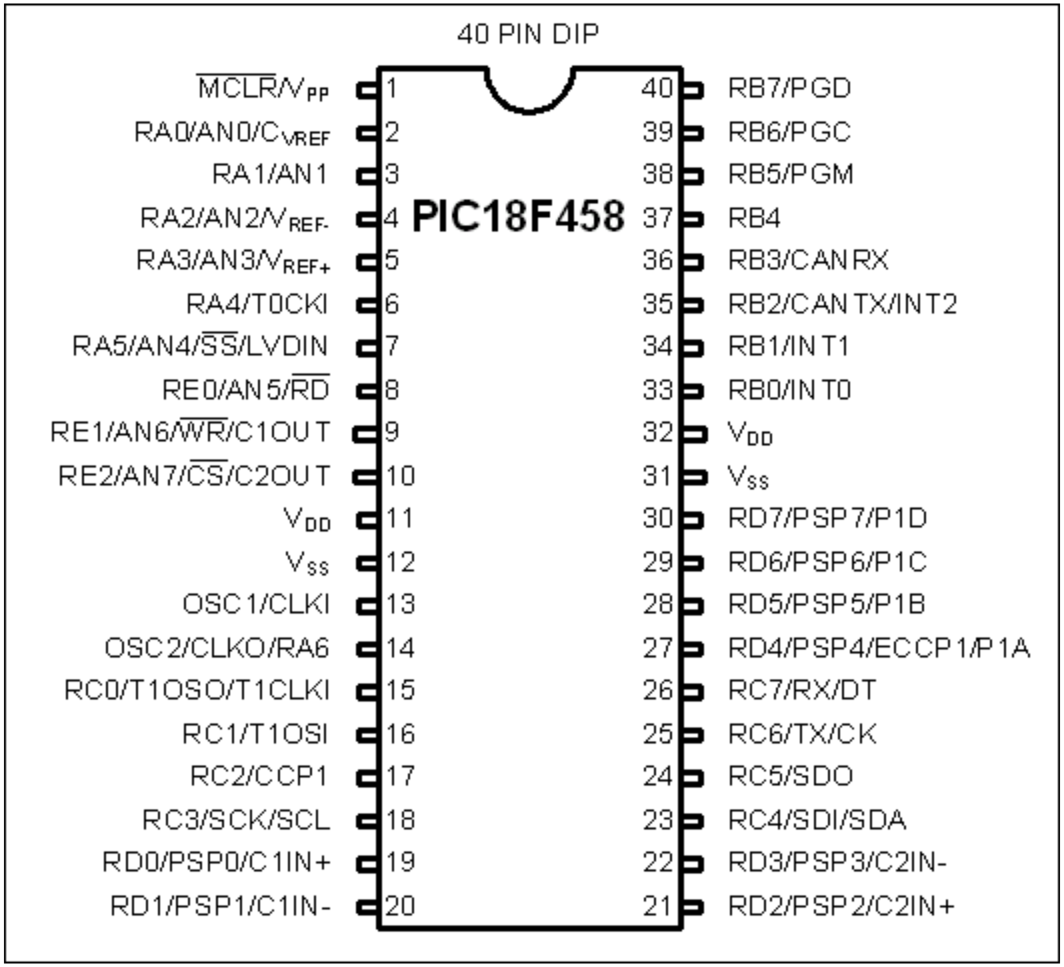
Embedded Systems I

Based heavily on slides by Dr. Roger Walker

Chapter 4 – PIC I/O PORT PROGRAMMING

- Ports are not only used for simple I/O, but also can be used other functions such as ADC, timers, interrupts, and serial communication pins. The following figure (Figure 4-1) shows the alternate functions for the PIC18F458 pins.

Figure 4-1 PICF458 Pin Diagram



PIC18F458/452 (40 Pins) has 5 ports, other Family Members Can Have More or Less Number

Table 4-1: Number of Ports in PIC18 Family Members

Pins	18-pin	28-pin	40-pin	64-pin	80-pin
Chip	PIC18F1220	PIC18F2220	PIC18F458	PIC18F6525	PIC18F8525
Port A	X	X	X	X	X
Port B	X	X	X	X	X
Port C		X	X	X	X
Port D			X	X	X
Port E			X	X	X
Port F				X	X
Port G				X	X
Port H				X	X
Port J				X	X
Port K					X
Port L					X

Note: X indicates that the port is available.

Number of Individual Port Pins

- For example, for the PIC18F458, Port A has 7 pins; Ports B, C, and D each have 8 pins; and Port E has only 3 pins.
- Each port has three SFRs associated with it. -- PORTx, TRISx (TRIState), and LATx (LATch).

Using PIC18F458 A-E Ports for Input/Output

- Each of the Ports A-E in the PIC18F458 can be used for input or output. The TRISx SFR is used solely for the purpose of making a given port an input or output port. To make a port an output, write Os to the TRISx register. Or, to output data to any of the pins of the Port B, first put Os into the TRISB register to make it an output port. Then send the data to the Port B SFR itself.

Addresses of SFR, PORTx, TRISx (TRIState), and LATx (LATch).

- See Table 4-2
- PORTA F80H
- PORTB F81H
-
-
-
- TRISA F92H
-

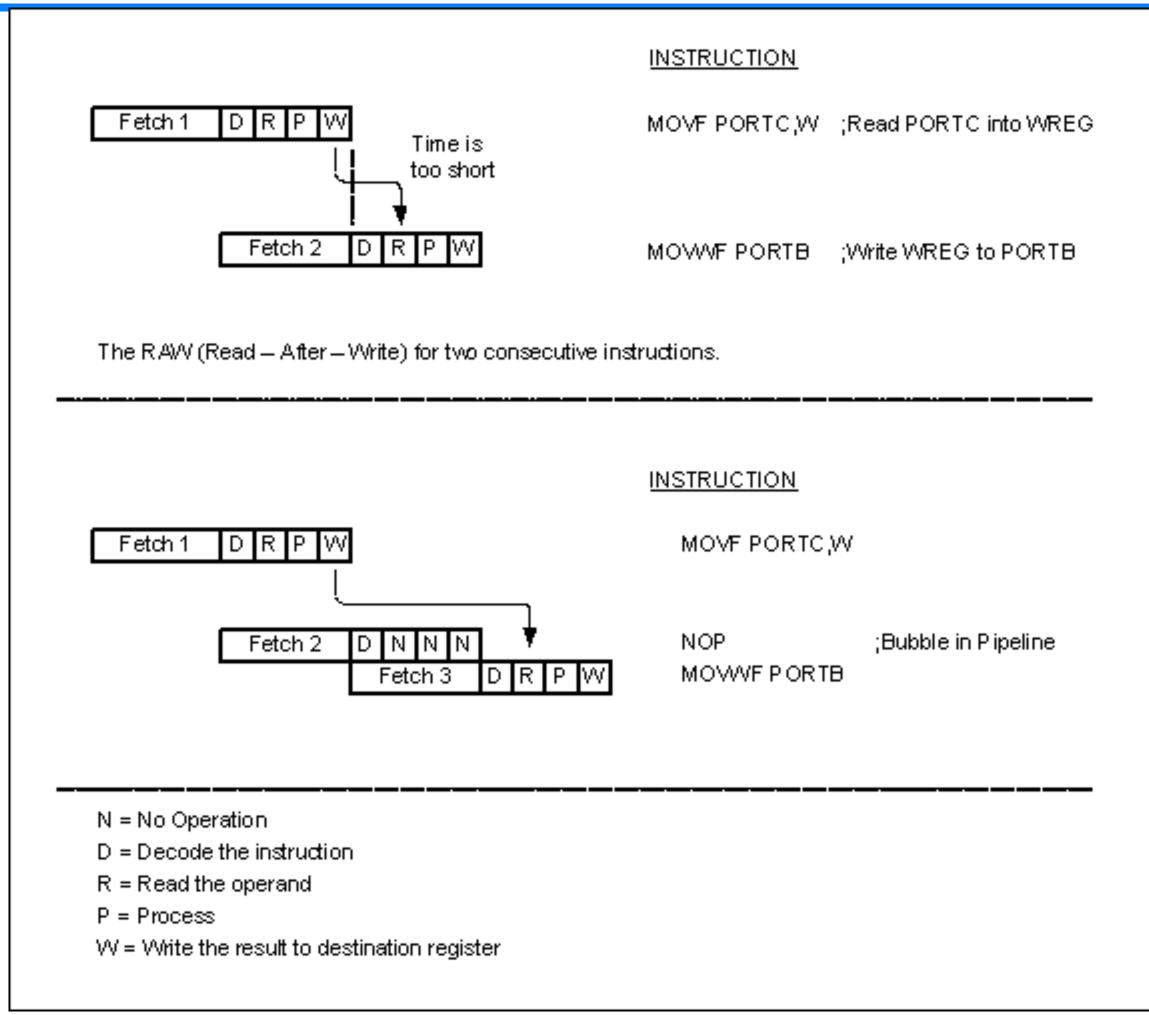
Example Using Port A as Input

In order to make all the bits of Port A an input, TRISA must be programmed by writing 1 to all the bits. In the code below, Port A is configured first as an input port by writing all 1 s to register TRISA, and then data is received from Port A and saved in some RAM location of the file registers:

```
MYREG     EQU 0X20    ;Program location (RAM)
MOVLW     B'11111111'       ;All 1's to WREG
MOVWF     TRISA  ;Port A as input port (1 for In)
MOVF  PORTA,W ;move from filereg of Port A to WREG
MOVWF  MYREG   ;save in fileReg of MYREG
```

Read After Write

- Need to add NOP between read from port and write to port, or use 4 byte MOVFF instruction



Register bit manipulation

- Bit set flag BSF filereg, bit
- Bit clear flag BCF filereg, bit
- Bit toggle flag BTF filereg, bit
- Bit test filereg skip next instruction if clear BTFSC filereg, bit
- Bit test filereg skip next instruction if set BTFSS filereg, bit
- Work for all file registers but especially helpful for PortA(RA0-RA5), PortB, PortC, PortD, and PortE(RE0-RE2)

Read-Modify-Write

- Any instruction which performs a write operation actually does a read followed by a write operation. The BCF and BSF instructions, for example, read the register into the CPU, execute the bit operation, and write the result back to the register. Caution must be used when these instructions are applied to a port with both inputs and outputs defined.
- For example, a BSF operation on bit5 of PORTB will cause all eight bits of PORTB to be read into the CPU. Then the BSF operation takes place on bit5 and PORTB is written to the output latches. If another bit of PORTB is used as a bi-directional I/O pin (e.g., bit0) and it is defined as an input at this time, the input signal present on the pin itself would be read into the CPU and rewritten to the data latch of this particular pin, overwriting the previous content. As long as the pin stays in the input mode, no problem occurs. However, if bit0 is switched to an output, the content of the data latch may now be unknown.
- The LATx registers could/should be used in this case.

CSE@UTA

PORT vs LATCH

- The differences between the PORT and LAT registers can be summarized as follows:
  - **A write to the PORTx register writes the data value to the port latch.**
  - A write to the LATx register writes the data value to the port latch.
  - **A read of the PORTx register reads the data value on the I/O pin.**
  - A read of the LATx register reads the data value held in the port latch.

Fan-out

- Current can flow in (pin at 0 level) and out (pin at 1 level) of port pins.
- This current is limited by the design of the IC.
- For PIC18 pins can drain a total of 8.5mA and source a total of 3mA.
- Fan-out is really the number of logic gates a pin can drive but is closely connected to the total current of pins. (total current / current drained by the input of the next logic gates)

Working with I/O Ports Using C

```
#include <p18F452.h>
void main(void)
{
    unsigned char mybyte;
    TRISC = 0b11111111;  //PORTC is input
    TRISB = 0b00000000;  //PORTB is output
    TRISD = 0b00000000;  //PORTD is output
    while(1)
    {
        mybyte = PORTC;   // load the value of PORTC
        if(mybyte < 100)
                PORTB = mybyte;  //send it to PORTB is it is less than 100
        else
                PORTD = mybyte; //otherwise, send to PORTD
    }
}

//don't forget linker script and library settings in MPLAB!
```

Example 7-11

Working with I/O ports using C

- Lab example:
  - Read the numbers using the example shown in the previous slide.
  - Do arithmetic manipulation on the input
  - Convert the output into decimal value to display.

Example 7-14 pp 263

Example 7-14

A door sensor is connected to the RB1 pin, and a buzzer is connected to RC7. Write a C18 program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz frequency to it.

Solution:

```
#include <P18F458.h>
void MSDelay(unsigned int);
#define Dsensor PORTBbits.RB1
#define buzzer PORTCbits.RC7
void main(void)
  {
    TRISBbits.TRISB1 = 1;          //PORTB.1 as an input
    TRISCbits.TRISC7 = 0;          //make PORTC.7 an output

    while(Dsensor == 1)
      {
        buzzer = 0;
        MSDelay(200);
        buzzer = 1;
        MSDelay(200);
      }
    while(1);                  //stay here forever
  }

void MSDelay(unsigned int itime)
  {
    unsigned int i;
    unsigned char j;
    for(i=0;i<itime;i++)
      for(j=0;j<165;j++);
  }
```

Logic Analyzer for Ex. 7-14

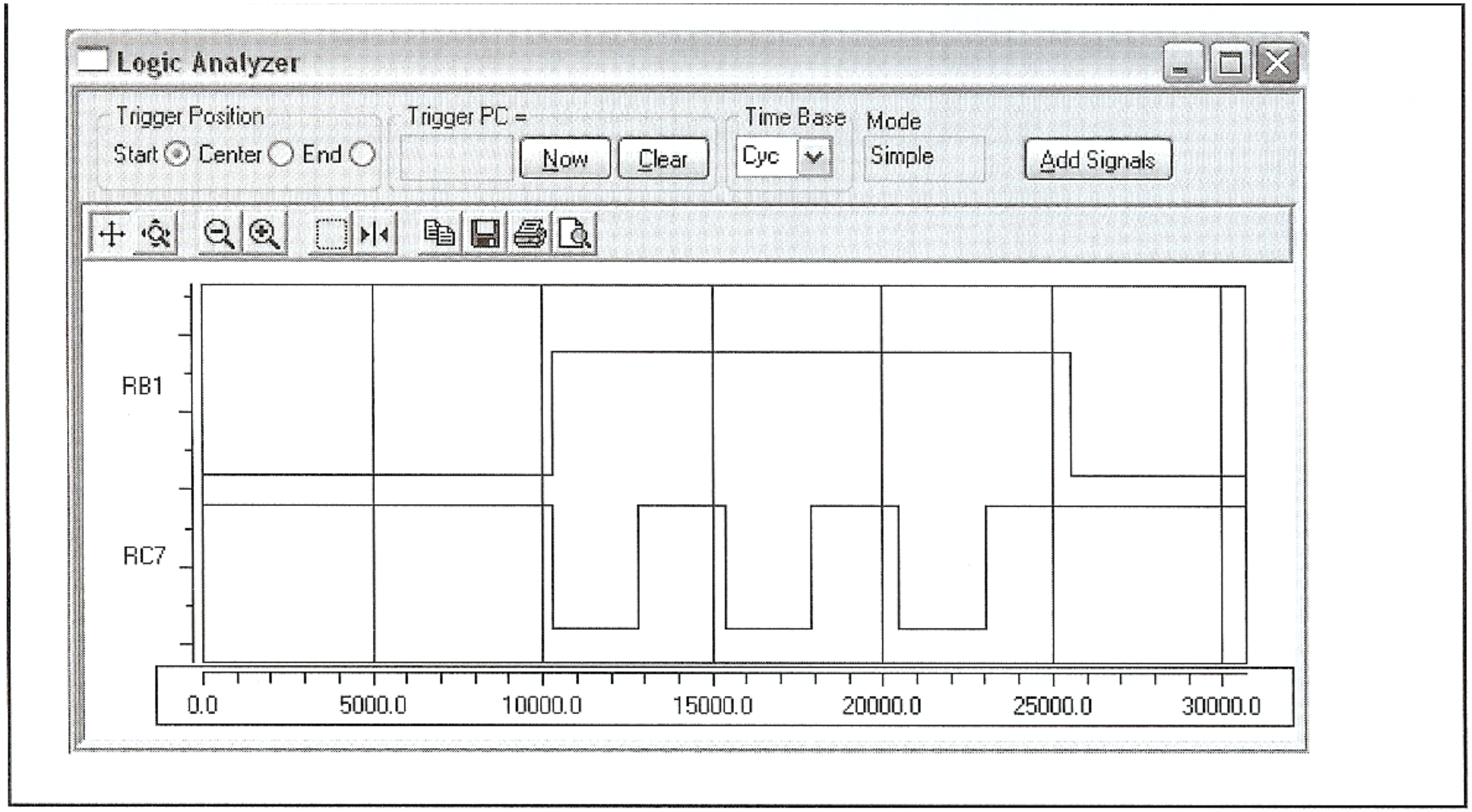


Figure 7-6. MPLAB Logic Analyzer for Example 7-14