Read Modify Write Problem

PIC16Fxxx family of MCUs

The Microchip mid-range PIC microcontrollers use a sequence known as **Read-Modify-Write** (RMW) when changing an output state (1 or 0) on a pin. This can cause unexpected behavior under certain circumstances.

When your program changes the state on a specific pin, for example RB0 in PORTB, the PIC microcontroller first **READs** all 8 bits of the PORTB register which represents the states of all 8 pins in PORTB (RB7-RB0). The PIC microcontroller then stores this data in the MCU. The bit associated with RB that you've commanded to **MODIFY** is changed, and then the PIC microcontroller **WRITEs** all 8 bits (RB7-RB0) back to the PORTB register.

During the first reading of the PORT register, you will be reading the actual state of the physical pin.
The problem arises when an output pin is loaded in such a way that its logic state is affected by the load. Instances of such loads are LEDs without current-limiting resistors or loads with high capacitance or inductance.

For example, if a capacitor is attached between pin and ground, it will take a short while to charge when the pin is set to 1.
On the other hand, if the capacitor is discharged, it acts like a short circuit, forcing the pin to '0' state, and, therefore, a read of the PORT register will return 0, even though we wrote a 1 to it.

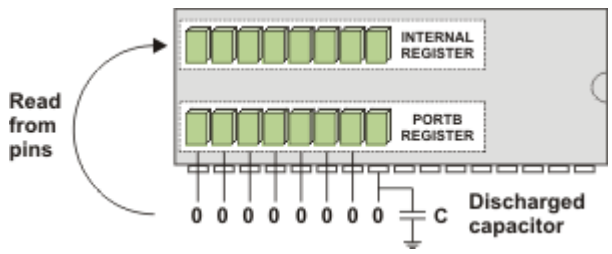
Lets analyze the following example :

```
PORTB.B0 = 1;
PORTB.B1 = 1;
```

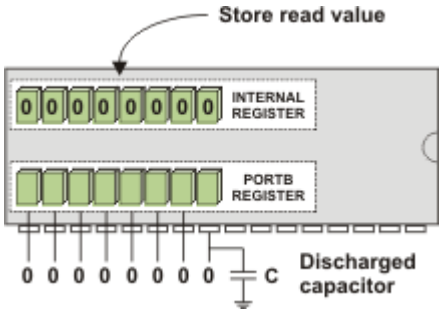
Assume that the PORTB is initially set to zero, and that all pins are set to output. Let's say we connect a discharged capacitor to RB0 pin.

The first line, `PORTB.B0 = 1;` will be decoded like in this way :

READ PORTB is **read** :

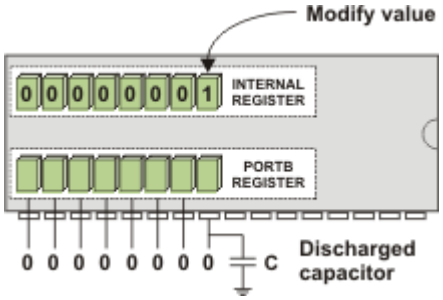


STORE Data is stored inside a temporary internal register in the PIC :

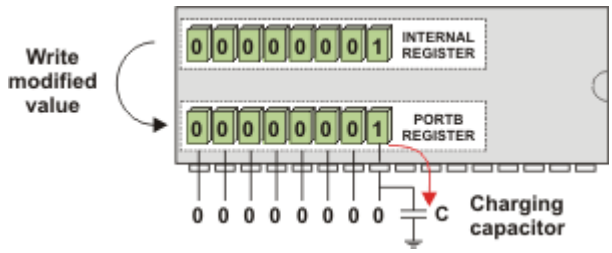


Actual voltage levels on MCU pins are relevant.

MODIFY Data is **modified** to set the RB0 bit :

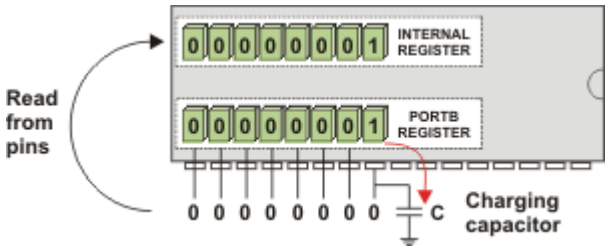


WRITE PORTB is **written** with the modified data. The output driver for RB0 turns on, and the capacitor starts to charge :

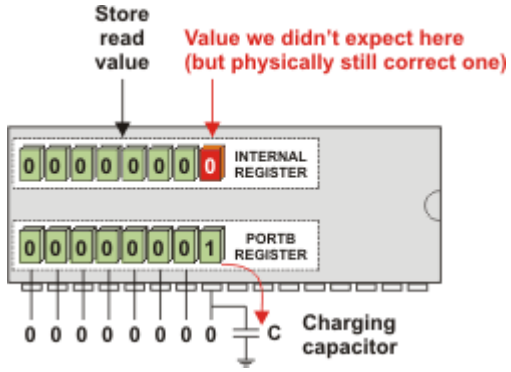


The second line, `PORTB.B1 = 1;` will be decoded like in this way :

READ PORTB is **read** :

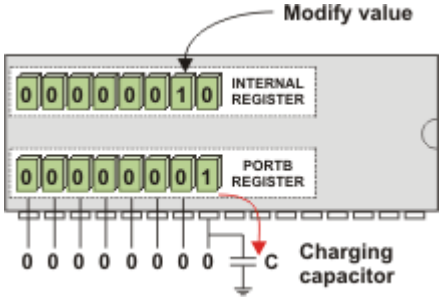


STORE Because the capacitor is still charging, the voltage at RB0 is still low and reads as a '0' (since we are reading from the pins directly, not from the PORTB register) :

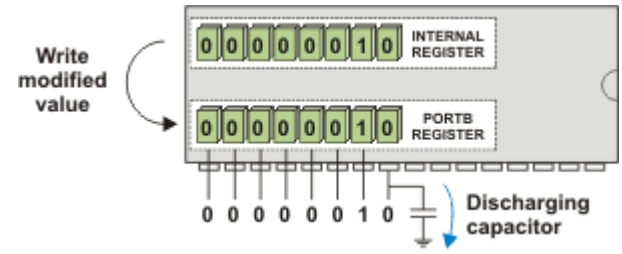


Actual voltage levels on MCU pins are relevant.

MODIFY Data is **modified** to set the bit :



WRITE PORTB is **written** with the new data. The output driver for RB1 turns on, **but the driver for RB0 turns back off** :



To correct the problem in the code, insert a delay after each `PORTB.Bx = 1` line, or modify the entire PORTB register in a single line `PORTB = 0b00000011`.

PIC18Fxxxx family of MCUs

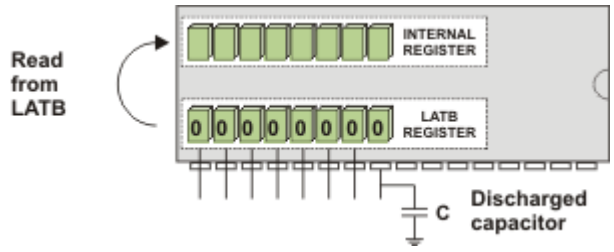
On a PIC18Fxxxx device, this problem can be avoided by using LATx register when writing to ports, rather than using PORTx registers.
Writing to a LATx register is equivalent to writing to a PORTx register, **but readings from LATx registers return the data value held in the port latch, regardless of the state of the actual pin**.

For example, lets analyze the following example :

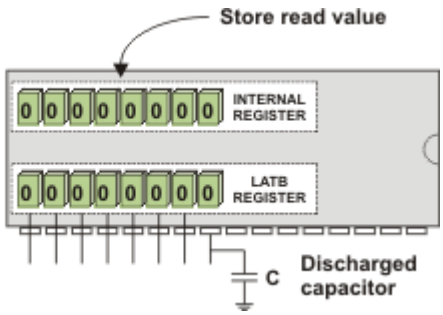
```
LATB.B0 = 1;
LATB.B1 = 1;
```

The first line, `LATB.B0 = 1;` will be decoded like in this way :

READ LATB is **read** :

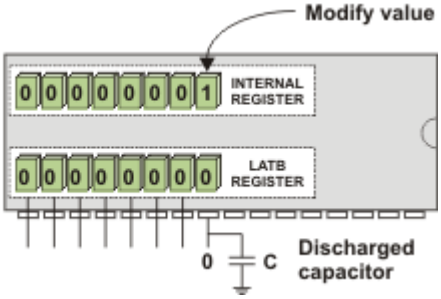


STORE Data is stored inside a temporary internal register in the PIC :

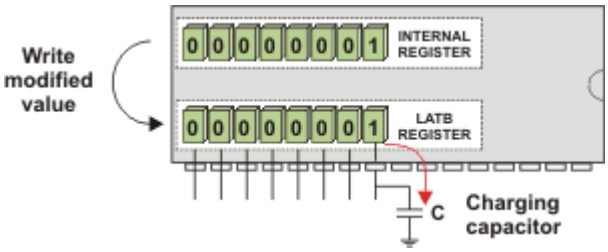


Actual voltage levels on MCU pins are no longer relevant when using LATx for output.

MODIFY Data is **modified** to set the RB0 bit :



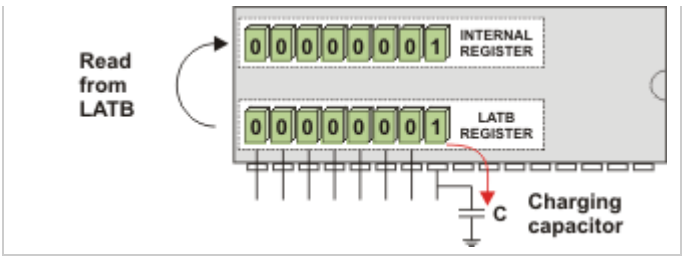
WRITE LATB is **written** with the modified data. The output driver for RB0 turns on, and the capacitor starts to charge :

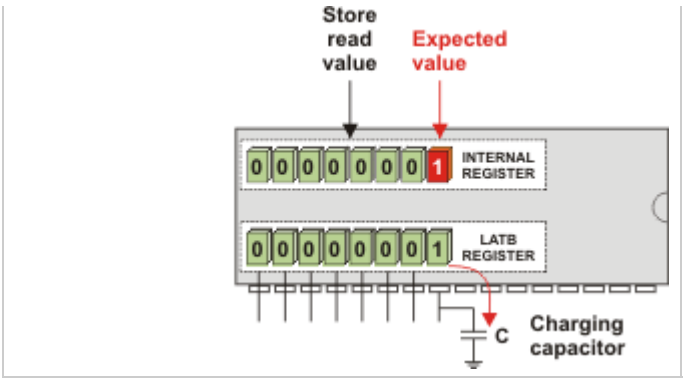


The second line, `LATB.B1 = 1;` will be decoded like in this way :

READ LATB is **read** :

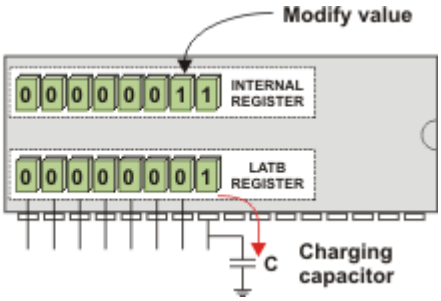
STORE Since the voltage levels on MCU pins are no longer relevant, we get the expected value :



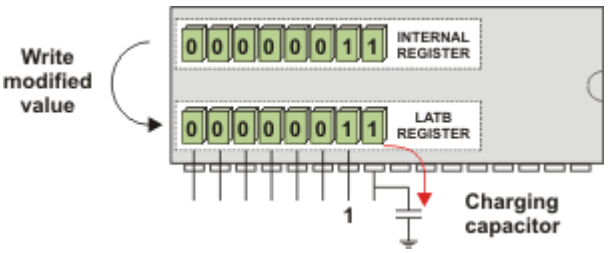


Actual voltage levels on MCU pins are no longer relevant when using LATx for output.

MODIFY	Data is **modified** to set the bit :



WRITE	LATB is **written** with the new data. The output driver for RB1 turns on, and the output driver for RB0 remains turned on :



When to use LATx instead of PORTx

Depending on your hardware, one may experience unpredictable behavior when using PORTx bits for driving output.
Displays (GLCD, LCD), chip select pins in SPI interfaces and other cases when you need fast and reliable output, **LATx should be used instead of PORTx.**