

# TUT#2 - TUTORIAL SERIES

## PIC PORTS PT 1

This document is copyright and is considered correct at the time of printing / release. The content contained herein is subject to change without notice and no liability is accepted for use of the products contained herein. All product names, trademarks mentioned in this document remain the property of the respective owners.



DUETRONIX

## Table of Contents

|                              |   |
|------------------------------|---|
| Revision Information.....    | 3 |
| Tutorial 2.....              | 4 |
| Using Variables.....         | 4 |
| Read / Write.....            | 4 |
| New C Concept.....           | 6 |
| Summary .....                | 6 |
| Quiz .....                   | 6 |
| Answer .....                 | 6 |
| Disclaimer and Notices ..... | 8 |

## Revision Information

The following table details important information pertaining to the document. For the latest or current version of this material, go to the following site.

|                          |                                      |
|--------------------------|--------------------------------------|
| <b>Document Title</b>    | Tutorial Series #2                   |
| <b>Date last revised</b> | May, 2008                            |
| <b>Revision Detail</b>   | 1.0.0                                |
| <b>Author</b>            | Duetronix Information Resources Dept |

## Tutorial 2

Now that we have looked at getting a PIC fired up and doing something, like toggling an LED. Let's look at some more concept based information surrounding the ports. If you are thinking boring, keep in mind that most issues you will face while working on your projects will be related to bits, bytes and registers not doing what you think that they should be doing.

### Using Variables

We dealt with PORT B in our previous tutorial, so let's build on that. As mentioned, a port is orderly collections of I/O that can be represented by a register 8 bit wide. From this it becomes clear that a port can only represent a number that is 8 bits in size.

In binary that gives us a number 255 or  $2^8 - 1$ . In code, this number is represented by variables as char or character. In practice this type is very useful and used extensively. When we move on to representing ASCII characters, they are all 255 or 8 bits in size.

In code:

```
char myvariable;
```

**Note:** The Mikro C compiler is capable of managing numbers much larger in size than char.

### Read / Write

To write the value of a register to a port, we do the following:

```
PORTB=myvariable;
```

The method of writing the value of a register or variable to its destination is always DESTINATION = SOURCE. To elaborate on the example from above, the destination is the PORT register of port B and the source is a variable containing a value 0 to 255 that will be represented in binary on the port when the contents of the variable are written to it.

Notably, the variable's value remains unchanged. Point in case, if the above statement were written as myvariable = PORTB, the value of myvariable would be overwritten by the value on the port. On the other hand this is correct if you are reading from PORT B into the variable. This is how a PIC obtains data from it's outside world. So it works both ways.

```
PORTB=myvariable; // PORT B all set to outputs | 0X00 - write to port
myvariable=PORTB; // PORTB all set to inputs | 0Xff - read from port
```

Let's illustrate how to make use of the above.

In the next code listing, the variable *myvariable* will be used as a counter, to hold the value of an up or down counter and the result of the operation will be written to PORTB. The name will change to *mycounter* so that its use makes more sense.

```
Char mycounter;

Main()
{
  TRISB=0x00; // all pins output
  PORTB=0x0; // not required but clears register

  While(1)
  {
    Delays(20);
    mycounter = mycounter ++;
    PORTB= mycounter;
  }
}
```

To analyze the above:

- The variable my counter is declared
- The PORTB tris register is configured for output and the port register is cleared
- The PIC uses an infinite loop to increment the counter *mycounter* by one on each passing of the loop
- Finally, the result is displayed on the port.

When you run this code example on your micro, you will notice that the port leds change value in an orderly fashion. Yup, you guessed it. You just created a binary up counter. If you are familiar with discreet logic in the form of logic ICs you will recognize the potential in the micro as a logic gate replacement device, in some ways. For extensive replacement gate arrays are used and not PICs – but you get the idea all the same.

## New C Concept

Incrementing a value can be done in two ways:

1. Mycounter + 1, or
2. Mycounter ++

Both are valid and provide the same result.

**Note:** For details on using 7 segment displays, see Annexure A.

## Summary

- 8 bit PICS have an 8 BIT wide PORTs that can be character type numbers
- CHAR is  $2^8-1$  or 255
- Data / number values cab be read from and written to a PORT

## Quiz

Rewrite the code sample below to create a Binary down counter.

```
Char mycounter;

Main()
{
  TRISB=0x00; // all pins output
  PORTB=0x0; // not required but clears register

  While(1)
  {
    Delays(20);
    mycounter = mycounter ++;
    PORTB= mycounter;
  }
}
```

## Answer

```
Char mycounter;

Main()
{
  TRISB=0x00; // all pins output
  PORTB=0x0; // not required but clears register

  While(1)
```

```
{
    Delays(20);
    mycounter = mycounter --;
    PORTB= mycounter;
}
```

## **Disclaimer and Notices**

This work is ©, 2008 Duetronix, South Africa. The document may be used for personal or educational use but may not be used by any corporate or commercial training institution whatsoever without the expressed permission of Duetronix.

All product names and branding mentioned or referred to in this work remain the copyright and intellectual property of their respective owners.

All rights reserved.