DEPARTMENT OF MECHANICAL ENGINEERING
VANDERBILT UNIVERSITY

ME 8353
APPLICATION NOTE ON READING AND WRITING REGISTERS IN THE DSPIC

Much of microcontroller programming is writing to registers (e.g., initializing control registers, sending digital output to port registers, etc.) or reading from registers (e.g., polling status registers or bits, reading digital input from ports or port registers, reading from an encoder register, etc.).

All the registers in a microcontroller, and all the bits within all the registers, are named. Chapter 4 of the data sheet lists all the named registers in the dsPIC, and all the named bits within those registers. You won't need to use the vast majority of these, but you will need some, depending on the functions you use. Fortunately, the compiler incorporates a simple naming structure for writing to and reading from registers. In the naming structure, for a register named REGISTER, for a bit within the register named BIT, or for several bits within the register named BITS (see datasheet for naming), a register can be referred to as REGISTER, a bit within a register as REGISTERbits.BIT, or a set of bits within the register as REGISTERbits.BITS. The development environment (i.e., MPLABX) provides useful support for this structure by automatically providing a pull-down menu for the bit name after you type "REGISTERbits."

Using this structure, a register named REGISTER can be written to directly with the line:

```
REGISTER = value;
```

where value is any numeric representation (e.g., binary or hex) of the number (usually 16-bit) in the register.

A bit named BIT within a register named REGISTER can be written to directly with the line:

```
REGISTERbits.BIT = value;
```

where value is either a 0 or 1.

A segment of n bits named BITS within a register named REGISTER can be written to directly with the line:

```
REGISTERbits.BITS = value;
```

where value is any numeric representation (e.g., binary or hex) of the number of bits represented by BITS.

You can read from a register, a bit in a register, or several bits in a register by simply referring to the name of the register, or register structure. Some variations on this theme, and examples, are given below.

Writing to control registers:

Writing to an entire 16-bit binary word to a register named REGISTERx:

```
REGISTERx = 0b1010111100001101 ;

Example:
TRISA = 0b0000000000000000 ;
/* sets PORTA to all outputs, see section 11 of datasheet */
```

Writing the same data to the same register in hex:

```
REGISTERx = 0xAF0D ;

Example:
TRISA = 0x0000 ;
/* sets PORTA to all outputs, see section 11 of datasheet */
```

Writing data to a single bit named BITy in a control register named REGISTERx:

```
REGISTERxbits.BITy = 1 ;
```

```
Example:
TRISBbits.TRISB10 = 1 ;
/* set PORTB bit 10 to input, see section 11 of datasheet */
```

Writing data to a set of bits (e.g., 3) named BITy in a control register named REGISTERx:

```
REGISTERxbits.BITy = 0b100 ;
```

```
Example:
DFLT1CONbits.QECK = 0b011;
/* set QEI digital filter clock divide bits to 1:16, see QEI reference page 7 */
```

Writing to I/O ports:

Writing data to a single bit named BITy in an I/O port named PORTx:

```
PORTxbits.BITy = 0 ;
```

```
Example:
PORTCbits.RC2 = 1;
/* turn on digital output RC2, see section 11 of datasheet */
```

Writing data to a two (or more) single bits named BITy and BITz to the same I/O port named PORTx:

```
PORTxbits.BITy = 1 ;
asm("nop") ;
PORTxbits.BITz = 0 ;
```

```
/* The asm("nop"), which tells the processor to wait one clock cycle before continuing, is required to
prevent overwriting of data when successively writing to different bits to the same port. */
```

```
Example:
PORTEbits.RE0 = 0 ;
asm("nop") ;
PORTEbits.RE1 = 0 ;
/* turn off digital outputs RE0 and RE1 */
```

Alternate method of writing data to a two (or more) single bits named BITy and BITz to the same I/O port named PORTx:

```
LATxbits.BITy = 1 ;
LATxbits.BITz = 0 ;
```

```
/* When you write to LATx (instead of PORTx), it writes to the same port (i.e., PORTx), but does
not require the asm("nop") statement between successive writes to the same port. */
```

```
Example:
LATFbits.RF0 = 0 ;
LATFbits.RF1 = 1 ;
```

```
/* turn off digital output RF0 and turn on RF1 */
```

Reading from status or information registers:

Reading the value of a single bit named BITy in a status register name REGISTERx:

```
value = REGISTERxbits.BITy ;

/* value should be declared as an integer */

Example:
if (IFS0bits.T1IF == 1) { // timer 1 clocked has ticked
        IFS0bits.T1IF = 0; // reset timer 1
// Perform whatever function is to be performed here
}
```

Reading from an information register name REGISTER:

```
value = REGISTER ;

/* Note: value should be declared as UINT16, assuming REGISTER is 16-bit */

Example:
// Read encoder, see QEI application note page 11
enc_count = POS1CNT ;
```

Reading from an I/O port named PORTx:

```
value = PORTx ;

/* Note: value should be declared as UINT16 */

Example:
value = PORTE ; // read entire 16-bit port
```

Reading the value of a single bit BITy in I/O port named PORTx:

```
value = PORTxbits.BITy ;

Example:
if (PORTBbits.RB5 == 0) {
// if RB5 is low, do something …
}
```