Introduction

Three types of motors are applicable to small projects: DC brushed motors, stepper motors, and servo motors. DC brushed motors simply rotate in a direction dependent on the flow of current. Servo motors move to an angular location dependent on the duty cycle of a pulse width modulated (PWM) input signal. Stepper motors rotate in discreet steps. Motors require large currents to operate, exceeding the drive of most digital logic; therefore, H-bridges are used to supply the high currents under the control of digital signals. Motor design and torque/speed considerations are beyond the scope of this documentation.

Pulse Width Modulation

Torque of most electrical motors is dependent on the power (P = I * V) being used to drive the motor. The easiest solution to vary the power of a digital signal is by using a method called Pulse Width Modulation (PWM). A pulse width modulated signal is a square wave, which, when sufficiently fast, creates an effective average voltage on the line. The ratio of high pulse length to period of the signal is called the duty cycle. By varying the duty cycle you can vary the average voltage. Hence, to give the motor more power, you should increase the duty cycle, as shown in Figure 1.

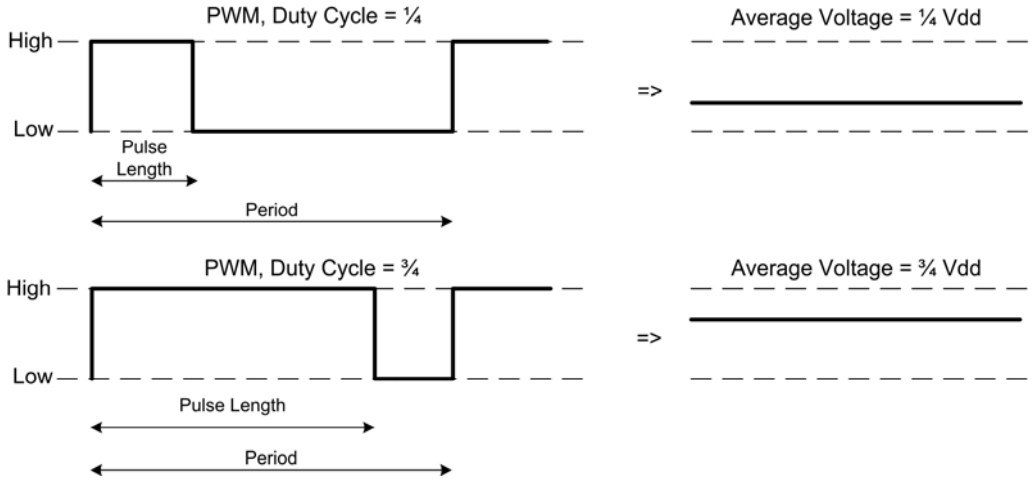


Figure 1: Pulse Width Modulation

The PIC18F452 includes a built-in PWM module that supports a period up to 819.2 µs (see page 122 of the PIC datasheet for how this is calculated) using a 20 MHz oscillator (409.6 µs with a 40 MHz oscillator). Sometimes the period for a PWM you wish to generate is too long for the PWM module. To work around this limitation you can use the PIC18F452's interrupt system to generate the PWM signal. An interrupt request handler (a procedure executed when an interrupt occurs) is called when a timer overflows, giving a predictable delay between calls to the handler. The interrupt request handler would then compare a "count" with a maximum period length variable and a high pulse length variable. A pin would be driven high when count is less than the high pulse

length variable and the count is reset when it exceeds the maximum period length. Both methods free the majority of processor time for other tasks. The PWM module drives a single pin, PORTC<2>. With the interrupt driven method you can drive any pin, as well as do more complex tasks beyond generating a simple PWM signal. Examples of both methods are given in the sample code section.

H-Bridges

The PIC output drivers are only rated for 25 mA, and most other digital chips provide less than 20mA. Motors draw more current than a PIC can provide. One exception is the servo, which has separate control and power inputs, so current draw is not an issue. Also, motors may operate at different voltage levels than a PIC. A common drive circuit, called an H-bridge, can be used to provide higher current and isolate voltage levels. An H-bridge consists of four switches/transistors/relays/etc: two switches go to VCC and two go to ground, as shown in Figure 2. By closing switches 1 and 4, current flows through the motor from left to right in the figure, hence the motor will turn one direction. By closing switches 2 and 3, the motor will turn the other direction. To apply braking to the motor, by induction, close switches 2 and 4. To allow the motor shaft to rotate freely, do not close any switches. The voltage VCC of the h-bridge may be larger than the logic levels of the control device you are using.

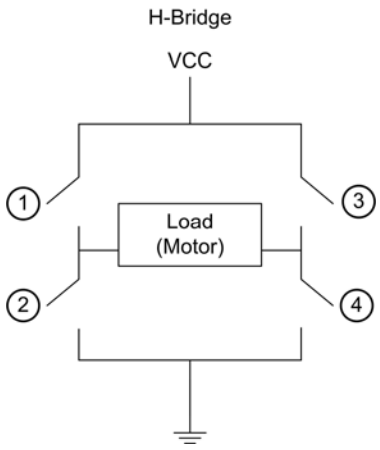


Figure 2: H-Bridge

H-bridges are often packaged in an IC form. Some come in half-h driver form, which is one side of an h-bridge (i.e. will drive a lead high, low, or high-impedance). The SN754410NE has four half-h drivers, which can be paired, so it can drive two motors with two separate H-bridges. It is rated for 1.1A continuous output at up to 36V. Although the input pins are specified for 4.5V control, they are also specified for TTL levels, and so appear to work at 3.3V. The SN754410NE has four input pins and two enable pins. When an enable pin is high, the two corresponding drivers are high or low depending on their input pins (when input pin is high, driver is high. Otherwise driver is low). When the enable pin is low, both drivers controlled by the enable pin are high-impedance.

DC Brushed Motors

Many motors use coils at fixed symmetrical angles to attract magnets on a motor shaft to different positions. When current is applied to the coils in the correct sequence, rotation occurs. A DC brushed motor automatically supplies current to the correct coils, often by using spring-loaded "brushes" which touch a contact, on the shaft, connected to a coil. As the shaft rotates, the contact the brush touches changes, keeping the shaft rotating without the need for more complex control circuitry. All that is needed to drive a brushed motor is current flow through the leads of the motor, as shown in Figure 3. Brushed motors are ideal for situations where high-speed/torque rotation is needed. However, the motor does not directly indicate velocity or shaft position. (A device called a *shaft encoder* may be used to optically determine the amount of rotation of the motor shaft, to determine the shaft position or velocity, but it is beyond the scope of this documentation.)

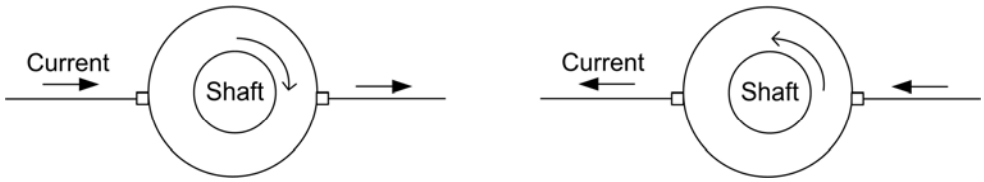


Figure 3: Driving a DC Brushed Motor

The speed/torque of the motor is determined by the electrical power (I*V) supplied and the direction of rotation is determined by the direction that current flows through the motor. To control the speed using the PIC, pulse width modulation is used. One motor lead is connected to PORTC<2>, which is the output for the PIC's built-in PWM generation module. The duty cycle determines the average voltage delivered to the motor and hence the speed of the motor. If speed control is not needed, or interrupt-based PWM is used, you do not need to use the PIC's built-in PWM and another pin may be used to control the motor.

To control directions, an arbitrary port, in the example code PORTC<1>, is connected to the other lead of the motor. Normally PORTC<1> is low, causing current to flow through the motor from PORTC<2> (when high) to PORTC<1>. When PORTC<1> goes high, current instead flows from PORTC<1> to PORTC<2> (when low), effectively reversing the direction of the shaft rotation.

Since the RS-550PF motor draws up to 29.1 A at 5VDC to 12VDC, more current than the PIC can provide, it is connected to a quadruple half-h driver, as shown in Schematic 1 on page 7 in the example schematic section. It is often desirable to give the motor more current than a single H-bridge can provide, so you may use multiple H-bridges attached to the same lead, or, of course, use a more powerful H-bridge.

Stepper Motors

Stepper motors have coils, connected directly to the external motor leads, which align the motor shaft to discreet positions or "steps". To rotate the motor shaft in a direction,

current must be sent through the motor leads in proper sequence. The stepper motor demonstrated in this paper is a bipolar stepper motor, which has two coils, perpendicular to each other, and multiple permanent magnets on the shaft. The Airpax LB82773-M1 stepper motor rotates 7.5 degrees per step. Stepper motors are ideal when precise control over amount of rotation is needed, but provide less speed and torque than a DC brushed motor would.

The sequence in which the motor leads are driven is called the stepping sequence. There are different stepping sequences depending on different factors such as power consumption, torque, and granularity of angle. For example, with a half-step stepping sequence, two leads could be driven simultaneously, which provides more torque than a simple one-phase sequence, which is discussed below, but requires more power to operate. The speed of rotation is determined by the delay between changing steps in the sequence. When starting at maximum speed the shaft will not rotate into position before the sequence progresses a step, causing the shaft to jitter but not rotate, therefore you should not start the sequence at maximum speed but instead gradually increase the stepping speed. To reverse direction of rotation just reverse the sequence. For this documentation we implement a simple one-phase (only one coil has current at a time) sequence. The stepping sequence as implemented in the sample code is shown in Table 1. As power is switched from one coil to another coil the motor moves a step. The process repeats causing continuous rotation in the direction indicated.

← Counter-Clockwise				Clockwise →
	1	2	3	4
Grey	ON	OFF	OFF	OFF
Black	OFF	OFF	ON	OFF
Red	OFF	ON	OFF	OFF
Yellow	OFF	OFF	OFF	ON

Table 1: Sample Airpax LB82773-M1 Stepping Sequence

Controlling a stepper motor using the 18F452 PIC is achieved using interrupts. First an initial bit-pattern is loaded onto PORTD. After a specific delay, which in the sample code is determined by the *speed* register, PORTD is rotated right or left depending on the direction desired of the motor shaft. For more complex stepping sequences, as discussed earlier, either modify the bit pattern on PORTD or implement an algorithm to determine which leads should be driven.

Servos

Servos are motor devices which can be positioned to a specific absolute angle. The servo can be viewed as a DC motor with a built-in controller. Internally servos use a device such as a potentiometer to measure the angle of the shaft, by using the voltage from the potentiometer. The servo's circuitry internally generates a signal from the voltage of the potentiometer, and then compares it to the input signal and matches the two by moving the DC motor accordingly. The speed at which the servo moves to the new position is proportional to the distance it needs to travel, so as the servo becomes closer to the target angle it will gradually slow. Servos are similar to stepper motors in that a specific amount of rotation can be achieved, but servos allow you to control an absolute angular

position instead of a relative amount of angular rotation. Additionally, servos often have finer granularity of amount of rotation than stepper motors, but are limited to a maximum angle of rotation, often not exceeding 360 degrees. Servos are ideal when absolute positioning of a shaft is desired.

The HS-322HD servo has three pins: VCC (Red), GND (Black), and SIGNAL (Yellow). Typically servos require a PWM signal with a 20ms period of a pulse length between 0.9ms and 2.1ms. 0.9ms corresponds to zero angle and 2.1ms corresponds to the maximum angle, as shown in Figure 4. Therefore, middle position is 1.5ms (the average of the pulse lengths). The HS-322HD has a maximum angle of 180 degrees. Servos only move a finite angular amount per cycle of the signal, so multiple cycles must be sent before the servo arrives at the correct angle. The number of cycles needed is dependent on the distance it must move. The servo will resist change away from the designated angle as long as signal is applied. The servo draws its power from VCC and the signal needs no more than 20 mA of current while running at 3VDC to 5VDC. Hence, it can be driven directly by the PIC without the need of an H-bridge. The voltage of the signal appears to not affect the angular position of the servo, as long as the voltage is within the 3VDC to 5VDC range. The HS-322HD specifications define VCC to be between 4.8VDC and 6VDC (though the servo appears to work at 3.3VDC). In the sample code the PWM is generated using interrupts, therefore the signal input on the servo can be connected to any arbitrary output pin on the PIC.

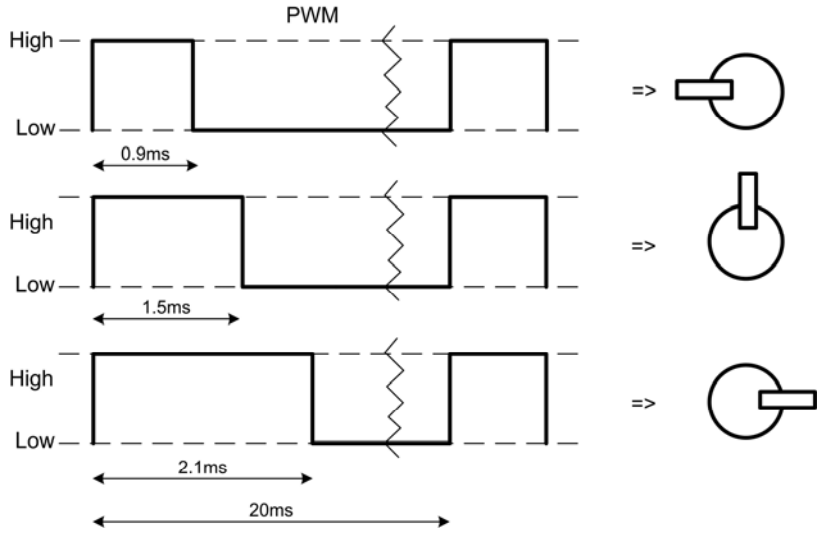


Figure 4: Controlling a HS-322HD Servo with PWM

Specifications

PIC18CXX2 Data Sheet
http://ww1.microchip.com/downloads/en/DeviceDoc/39026c.pdf

Mabuchi RS-550PF VDC Motor
http://cvhsrobotics.org/files/2003/mabuchi.pdf

Hitec HS-322HD Servo Motor
http://www.hitecrcd.com/Servos/spec_sheets/HS322HD.pdf

Airpax LB82773-M1 Stepper Motor
http://www.allelectronics.com/pdf/32.pdf

Texas Instruments SN754410NE Quadruple Half-H Driver
http://www-s.ti.com/sc/ds/sn754410.pdf

Supplier

Part	Vendor	Part #	Price
Mabuchi RS-550PF	All Electronics	DCM-104	$3.50
Airpax LB82773-M1	All Electronics	SMT-75	$2.75
SN754410NE H-Bridge	Digi-Key	296-9911-5-ND	$1.88
Hitec HS-322HD Servo	Hitec RCD USA	HS-322HD	$11.49

www.allelectronics.com
www.digikey.com
www.hitecrcd.com

Additional Resources

PICmicro® DC Motor Control Tips ‘n Tricks
http://ww1.microchip.com/downloads/en/DeviceDoc/41233A.pdf

Brushed DC Motor Fundamentals
http://ww1.microchip.com/downloads/en/AppNotes/00905a.pdf

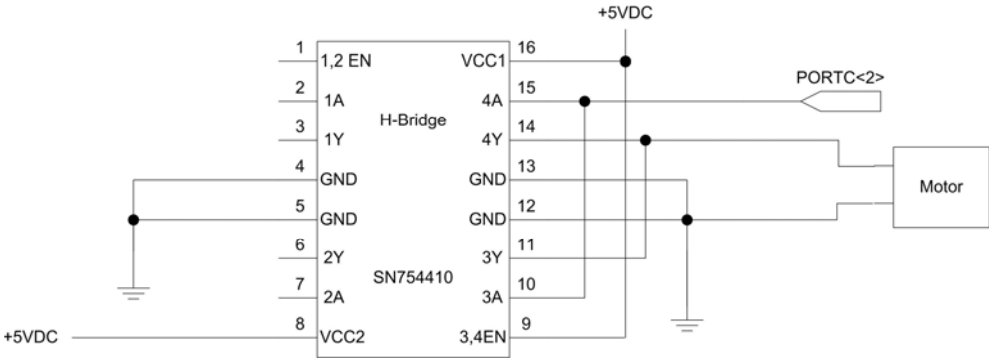
Stepper Motor Theory – Haydon Switch and Instrument
http://www.hsimotors.com/technical-data/theory.htm

Jones on Stepping Motor
http://www.cs.uiowa.edu/~jones/step/

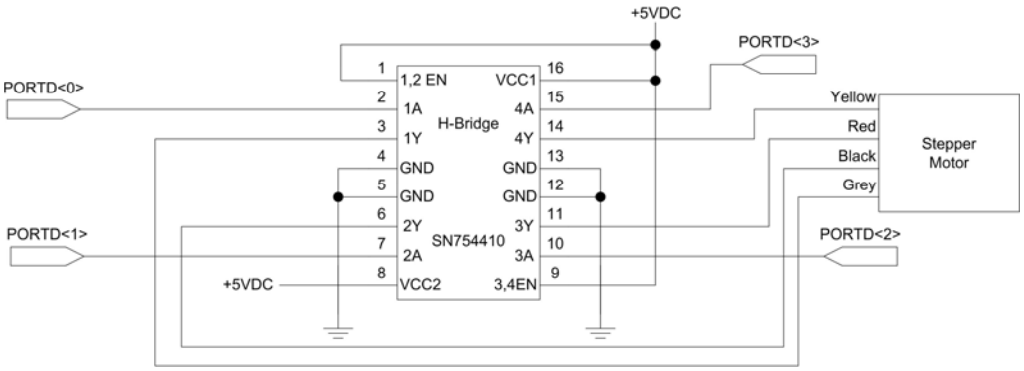
What’s a servo: A quick tutorial
http://www.seattlerobotics.org/guide/servos.html

Schematics

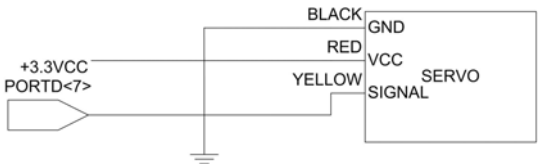
Schematic 1: Brushed DC Motor



Schematic 2: Stepper Motor



Schematic 3: Servo



Sample Code

Listing 1: brushed.asm

```
; brushed.asm:  Demonstrates how to drive a dc brushed motor.
; 2005 Nathaniel Pinckney <npinckney@hmc.edu>
; Takes advantage of 18F452's builtin PWM module.  Assumes 20Mhz clock.
;
; PORTC<1> is always 0 or 1, which will set direction by flipping polarity.
; PORTC<2> is controlled by PWM module.

      LIST p=18F452
      include "p18f452.inc"

       org 0x0

start:
       clrf TRISC            ; PORTC as output

       call init_pwm
       movlw b'01100100'     ; Speed.  MSB is direction.
       btfss WREG,7          ; Check direction.
       bcf PORTC,1           ; Set direction.
       btfsc WREG,7
       bsf PORTC,1
       btfsc WREG,7          ; Also, if opposite direction,
       negf WREG             ;  negate speed so pulse length is the same
       rlcf WREG             ; Multiply by 2, since speed is really on 7-bits.
       movwf CCPR1L          ; Move into PWM module's pulse length reg.
       goto $                ; $ is current address. Loop endlessly.
                             ; (Your code would replace goto loop.)

; Initializes the PWM module for DC Brushed motor control.
init_pwm:
       setf PR2              ; Give us a period of (255+1)*4 cycles.
       clrf CCPR1L           ; MSB of duty cycle, initially 0.
       movlw b'00000100'     ; 0 postscale, PWM on, 0 prescale
       movwf T2CON
       movlw b'00001100'     ; LSB of duty cycle, PWM Mode
       movwf CCP1CON
       return

end
```

Listing 2: stepper.asm

```
; stepper.asm:  Control a stepper motor.
; 2005 Nathaniel Pinckney <npinckney@hmc.edu>
; Assumes 20Mhz clock.
; Overview:
;   * Configure interrupt on TMR0 overflow. This slows us down some.
;   * Configure and enable TMR0.
;   * Move a bit pattern of wires activated to PORTD.  To cycle through
;     wires in sequence all we must do is rotate PORTD.
;   * Interrupt handler counts number of overflows in 'count' register and
;     compares to 'speed' register.  If count > speed then rotate
;     our PORTD which will activate correct wires in sequence.  Also,
;     count is then reset.
;   * Speed's MSB is the direction.  Rest of the bits are multiplied by 2
;     to determine delay.

      LIST p=18F452
      include "p18f452.inc"

; Registers
W_TEMP         equ 0x01       ; Save state, used in ISR.
STATUS_TEMP    equ 0x02       ; Save state, used in ISR.
count          equ 0x03       ; Keeps track for timing/speed.
speed          equ 0x04       ; Speed/delay of stepper motor.

       org 0x0                ; Reset vector.
       goto start
       org 0x08               ; Interrupt vector.
       goto TMR0_ISR          ; goto TMR0 ISR (Interrupt Service Request)

start:
       clrf TRISD             ; PORTD is output.
       movlw b'11100000'      ; Set our speed/delay.  MSB is direction.
       movwf speed

       movlw  b'10100000'     ; Enable all interrupts, and TMR0 overflow as
                              ; an interrupt.
       movwf  INTCON
       movlw  b'11000000'     ; Setup TMR0, 8-bit mode, no prescaler.
       movwf  T0CON

       movlw  b'00010001'     ; Stepping sequence pattern.
       movwf  PORTD

       goto       $           ; $ is current address. Loop endlessly.
                              ; (Your code would replace goto loop.)

TMR0_ISR:                             ; TMR0 ISR Handler.
       movwf  W_TEMP                  ; Save WREG
       movff  STATUS,STATUS_TEMP      ; Save Status
       incf   count                   ; Increment count.
       movf   speed,w
       rlcf   WREG                    ; Shift speed by 1 (multiply by two)
       cpfsgt count                   ; Check if we hit our upper-bound of count.
       goto   _then
       clrf   count                   ; Reset count
       btfsc  speed,7                 ; Rotate our output in direction
       rlncf  PORTD                   ;   determined by MSB of speed.
       btfss  speed,7
       rrncf  PORTD

_then:
       bcf    INTCON,TMR0IF           ; We handled the interrupt.
       movf   W_TEMP,w                ; Restore WREG
       movff STATUS_TEMP,STATUS       ; Restore Status
       retfie                         ; Return from interrupt handler.

end
```

Listing 3: servo.asm

```
; servo.asm:  Control a servo.
; 2005 Nathaniel Pinckney <npinckney@hmc.edu>
; Assume 20Mhz clock.
; Overview:
;   * Configure interrupt to trigger on TMR0 overflow.
;     Interrupt will trigger approximately every 0.1ms.
;   * Configure TMR0.
;   * On interrupt check if pwm >= pwm_count, if true turn PORTD<7> on,
;     else turn PORTD<7> off.
;   * Check if we reach 20ms by comparing pwm_count to 200.
;     If so reset to 0.

      LIST p=18F452
      include "p18f452.inc"

; Constants to test with.
ANGLE          equ .15        ; This is really pulse length in 0.1 ms, which
                              ; translates to angle. 15 (1.5 ms) is center position,
                              ; 9 (0.9 ms) is ~0 degrees, and 21 (2.1 ms) is ~180
                              ; degrees.
; Registers
W_TEMP         equ 0x01       ; Save state, used in ISR.
pwm_count      equ 0x02       ; Keeps track if TMR0 overflows, for pwm generation.
pwm            equ 0x03       ; Pulse length of pwm.

       org 0x0                ; Reset vector.
       goto start
       org 0x08               ; Interrupt vector.
       goto TMR0_ISR          ; (goto TMR0 ISR)

start:
       clrf TRISD             ; PORTD is output.
       movlw ANGLE            ; Pulse length (/ a period of 200)
       movwf pwm

       movlw  b'10100000'     ; Enable all interrupts, and TMR0 overflow as an
                                                      ; interrupt.
       movwf  INTCON
       movlw  b'11000000'     ; Setup TMR0, 8-bit mode, no prescaler.
       movwf  T0CON

       goto       $           ; $ is current address. Loop endlessly.
                              ; (Your code would replace goto loop.)


TMR0_ISR:                     ; TMR0 ISR (Interrupt Service Request) Handler.
       movwf  W_TEMP          ; Save WREG

       movf   pwm_count,w
       cpfslt pwm             ; Compare pwm to pwm_count.
       bsf    PORTD,7         ; >= PORTD<7> = 1
       cpfsgt pwm
       bcf    PORTD,7         ; <= PORTD<7> = 0

       incfsz pwm_count       ; infsz does not modify STATUS (unlike incf)
       movlw  .200            ; and since we don't save STATUS...
       cpfslt pwm_count       ; 200 is selected because TMR0 overflows
       clrf   pwm_count       ;  once every 0.1ms, and we want a 20ms period.

       bcf    INTCON,TMR0IF   ; We handled the interrupt.
       movf   W_TEMP,w        ; Restore WREG
       retfie                 ; Return from interrupt handler.

end
```