

```

/*****
/* Program      : INT.C
/* Function     : Interrupt Service Procedures.
/* Author      : John F. Fitter B.E.
/* Platform    : HiTech PIC C v7.72 pl1
/* Target      : PIC16C67
/*
/* Rev No.     Rev date   Test date   Test platform   Description
/* -----
/*    00      6jun98      -----      PIC16C77-ME     Original in HiTech C
/*
/* Copyright © 1998 Eagle Air Australia Pty. Ltd. All rights reserved
*****/

#define _INT_C
#include <commdefs.h>
#include "lcd44780.h"
#include "switch.h"
#include "main.h"
#include "serial.h"
#include "buzzer.h"

#pragma interrupt_level 1
void interrupt_isr() {

    // The dtr line must be asserted to prevent receive buffer overflow while interrupts are
    // being serviced if the host sends data. Since it is possible that the line is already
    // asserted due to the previous data byte processing not yet being complete, it is
    // necessary to restore the line to it's original state after interrupt processing.

    static volatile bit saved_dtr;
    saved_dtr = dtr;
    dtr = B_HIGH;

    // USART receive character interrupt handler - get the char and flag that a char has been
    // received. dtr is left set to prevent reception of more characters. dtr will be reset by
    // mainline code when the character has been processed. The system yield flag is set to
    // ensure rapid response to the received char.

    if(RCIF) {
        c_status.char_is_in = true;
        c_status.ser_rx_err = true;
        c_status.yield = true;
        if(OERR) {
            CREN = false;
            CREN = true;
        }else if(!FERR) c_status.ser_rx_err = false;
        ser_char = RCREG;
    }else {

        // Capture/compare interrupt handler. Timer 1 and tmr_aux constitute a 32 bit timer.

        if(CCPIF) {
            tmr_val.wtval.lw_tval = CCP1R1;
            tmr_val.wtval.hw_tval = tmr_aux;
            c_status.captured = true;
            CCP1IF = false;
        }

        // Timer 1 interrupt handler. Maintains a 16 bit auxillary counter to extend
        // the Timer 1 counter to 32 bits. This interrupt is thrown when timer 1 overflows.

        else if(TMR1IF) {
            ++tmr_aux;
            TMR1IF = false;
        }

        // Backlight interrupt handler - default 1000 Hz for LED backlight.
        // This interrupt occurs at 2kHz. The system yield flag is updated at 1kHz which
        // effectively provides time-sliced multitasking.

        else if(TMR2IF) {
            if(!lcd_bl_ison()){
                set_lcd_bl(false);
                set_bl_offtime();
                if(!--tmr_10ms) {
                    tmr_10ms = 10;
            }
        }
    }
}

```

```
        if(!buz_elaps) --buz_elaps;           // decrement the buzzer timer to zero
        if(!ser_time) --ser_time;           // decrement the serial timer to zero
        if(!auto_rep) --auto_rep;          // decrement autorepeat timer to zero
        if(!--tmr_100ms) {                 // decrement the 100ms timer
            tmr_100ms = 10;                // if zero, set timer to 100ms
            c_status.exp100ms = true;      // and set the 100ms flag
        }
    }
} else {
    set_lcd_bl(true);                      // no, turn it on
    set_bl_ontime();                        // and set the on time
    c_status.yield = true;                 // flag to yield control to system
    clrwdt();                              // good place to tie up the dog
}
TMR2 = 0;
TMR2IF = false;                          // clear the interrupt flag
}
dtr = saved_dtr;                          // restore the status of dtr
}
}
// ***** EOF INT.C *****
```