```
; PC-PCB Interface
; Program for PIC 16F872 microcontroller (compiles to PICProgram.HEX).
; Programmed December 9, 2011.
; The Host PC should be running HostProgram.vb in Visual Basic.
;
; This test program has a limited function.  Its main loop: (i) toggles an LED on and off every
; three seconds and (ii) every six seconds, it increments a number and sends it to the Host PC.
; From time-to-time, the Host PC will interrupt the PIC and send a four-bit nibble.
; The PIC interprets the nibbles as follows:
;   In response to b'0001'=d'1', the PIC sends b'1110'=d'14' to the Host PC
;   In response to b'0010'=d'2', the PIC sends b'0011'=d'3' to the Host PC
;   In response to b'0011'=d'3', the PIC waits one second and then sends b'0101'=d'5' to the Host PC
;   In response to b'0100'=d'4', the PIC turns on the LED
;   In response to b'0101'=d'5', the PIC turns off the LED
;   In response to b=0110'=d'6', the PIC reports the setting (open/closed) of the switch
;   In response to the other nine combinations of bits, the PIC sends b'1100=d'12' to the Host PC
;
; Register timer0 and its controlling module, Timer0, are used to detect communication timeouts.
; The Timer0 prescalar is set to 256:1, so register timer0 increments every 256 x 400ns = 102.4us.
; Therefore, timer0 will count from 0 to 0 (256 steps) in 256 x 102.4us = 26.2144ms.
; When a communication starts, register CommT0 is set to d'38' and register timer0 is reset to
; zero.  Register CommT0 is decremented at every Timer0 overflow-initiated interrupt and, if and
; only if it reaches zero, a communication timeout is declared.  Therefore, communication with the
; Host PC will time out after 38 x 26.2144 = 996.1472ms, which is close enough to one second for
; our purposes.
;
; Configuration Word
;       Program protection off  ; b<13-12>=1
;       Debugger disabled       ; b<11>=1
;       b<10> is unimplemented  ; b<10>=1, reads as 1
;       WRT enabled             ; b<9>=1
;       EEPROM protection off   ; b<8>=1
;       LVP off                 ; b<7>=0
;       BOR disabled            ; b<6>=0
;       Data protection off     ; b<5-4>=1
;       PUT disabled            ; b<3>=1
;       WDT disabled            ; b<2>=0
;       OSC HS                  ; b<1>=1; b<0>=0
; Crystal frequency = 10MHz, so the instruction cycle time is 400ns.
;
  processor     16F872
  __config      0x3F3A          ; b'0011 1111 0011 1010'
;
; ***********************************************************************************************
; ***********************************************************************************************
; Variable definitions - PIC 16F872 control registers
; ***********************************************************************************************
; ***********************************************************************************************
;
timer0      equ 0x01            ; Timer0 count register
status      equ 0x03            ; status register
carry       equ     0x00        ; carry from MSB has occurred
zero        equ     0x02        ; result of operation is zero
page0       equ     0x05        ; register bank selector low bit
page1       equ     0x06        ; register bank selector high bit
portA       equ 0x05            ; I/O port A
portB       equ 0x06            ; I/O port B
portC       equ 0x07            ; I/O port C
INTCon      equ 0x0B            ; interrupt control register
RB0IF       equ     0x01        ; RB0 interrupt flag
Tmr0IF      equ     0x02        ; Timer0 interrupt flag
RB0IE       equ     0x04        ; RB0 interrupt enable
Tmr0IE      equ     0x05        ; Timer0 interrupt enable
GIE         equ     0x07        ; global interrupt enable
T1CON       equ 0x10            ; controls use of portC<1-0>
CCP1CON     equ 0x17            ; controls use of portC<2>
optionreg   equ 0x81            ; option register
TRISA       equ 0x85            ; portA pin I/O direction
TRISB       equ 0x86            ; portB pin I/O direction
TRISC       equ 0x87            ; portC pin I/O direction
ADCON1      equ 0x9F            ; controls use of portA
f           equ     0x01        ; f and w identify the destination register
w           equ     0x00
;
; ***********************************************************************************************
```

```
; ********************************************************************************************
; Variable definitions - User RAM - Accessible only in bank0
; ********************************************************************************************
; ********************************************************************************************
;
; I/O port mirror registers
portAmirror equ 0x20            ; current contents of portA
SW          equ     0x00        ; switch voltage input - high voltage, or "1", when open
LED         equ     0x01        ; control for test LED - "on" when high
INTOut      equ     0x03        ; interrupt output to the Host PC
portBmirror equ 0x21            ; current contents of portB
INTIn       equ     0x00        ; interrupt input from the Host PC
portCmirror equ 0x22            ; current contents of portC
D0          equ     0x00        ; least significant bit on data bus
D1          equ     0x01
D2          equ     0x02
D3          equ     0x03        ; most significant bit on data bus
; registers for communicating with the Host PC
DataIn      equ 0x23            ; nibble received from the Host PC is DataIn<3-0>
DataOut     equ 0x24            ; nibble to send to the Host PC is DataOut<3-0>
CommTO      equ 0x25            ; counter for one-second communication timeout
; counters for timing subroutines
count1      equ 0x26            ; counter for del100us subroutine
count2      equ 0x27            ; counter for del10ms subroutines
count3      equ 0x28            ; counter for del500ms subroutine
; temporary storage register
temp        equ 0x29            ; temporary storage register
; incrementing number for display on the Host PC screen
number      equ 0x2A
; registers accessible in all banks, for Interrupt Service Routine
_w          equ 0x70            ; temporary storage for w register during interrupts
_status     equ 0x71            ; temporary storage for status register during interrupts
_wRB0       equ 0x72            ; storage for w register during RB0 interrupts
_statusRB0  equ 0x73            ; storage for status register during RB0 interrupts
;
; ********************************************************************************************
; ********************************************************************************************
; On power-up, execution begins at address 0x0000
;
    org 0x0000
    bcf     INTCon,GIE              ; disable global interrupts (until later)
    bcf     INTCon,RB0IE            ; disable RB0 interrupts (until later)
    bcf     INTCon,Tmr0IE           ; disable Timer0 interrupts (until later)
    goto    HardReset               ; goto HardReset on power-up
;
; ********************************************************************************************
; ********************************************************************************************
; Interrupt Service Routine (ISR) starts at address 0x0004
; There are two sources of interrupt: (i) by Timer0 (generated internally by the PIC) and
; (ii) by the Host PC (on a rising edge on pin RB0).  It is important to avoid conflicts
; between the successive interrupts.  The Host PC requests the right to transmit by
; triggering an RB0-type interrupt.  However, to allow communication errors to be detected,
; it is necessary for the program to permit Timer0-type interrupts even while it is
; processing an RB0-type interrupt.
; 1.  Conflict between two Timer0-type interrupts is avoided by: (i) making the duration of
;     a Timer0-type interrupt very short and (ii) by keeping Timer0 interrupts disabled
;     during execution of a Timer0-type interrupt.  Either method would be enough.
;     The only thing a Timer0-type interrupt does is decrement the register CommTO.
;     Since Timer0-type interrupts are 26.2144ms apart, a Timer0-type interrupt should
;     never occur while the preceding Timer0-type interrupt is still running.
; 2.  Conflict between two RB0-type interrupts is avoided by keeping RB0 interrupts
;     disabled during execution of an RB0-type interrupt.
; 3.  Conflict between the two types of interrupt is avoided by providing separate storage
;     for registers w and status.  Timer0-type interrupts store in registers _w and _status.
;     RB0-type interrupts store in _wRB0 and _statusRB0.
; 4.  If the Host PC is programmed correctly, a conflict between successive RB0-type
;     interrupts will never arise.  The handshake procedure by which the two computers
;     exchange data gives both of them confirmation that the communication is proceeding
;     as it should.  The Host PC should be programmed so that it, too, detects communication
;     timeouts.  If it is, then the Host PC will never start a second transmission to the PIC
;     before the preceding one is completed.
;
; ********************************************************************************************
; ********************************************************************************************
;
```

```
    org 0x0004
ISR
    bcf     INTCon,GIE              ; disable global interrupts but ...
                                    ; ... do not clear Timer0 or RB0 interrupt flags
    movwf   _w                      ; save registers w and status without affecting flags
    swapf   status,w
    movwf   _status
    btfsc   INTCon,Tmr0IF           ; INTCon<Tmr0IF> is set if Timer0 caused the interrupt
    goto    Timer0Interrupt
    goto    HostPCInterrupt
Timer0Interrupt
    movf    CommTO,w                ; set the zero flag based on the contents of CommTO
    btfss   status,zero
    decf    CommTO,f                ; decrement register CommTO, but not below zero
    swapf   _status,w               ; restore registers w and status without affecting flags
    movwf   status
    swapf   _w,f
    swapf   _w,w
    bcf     INTCon,Tmr0IF           ; clear the Timer0 interrupt flag
    bsf     INTCon,GIE              ; re-enable global interrupts
    retfie                          ; return from the Timer0 interrupt
HostPCInterrupt
    movf    _status,w               ; save registers _w and _status without affecting flags
    movwf   _statusRB0
    swapf   _w,f
    swapf   _w,w
    movwf   _wRB0
    bcf     INTCon,RB0IE            ; disable RB0 interrupts but ...
    bsf     INTCon,GIE              ; ... enable Timer0 Interrupts
    call    GetHostNibble           ; sets DataIn = 0xFF if reception times out
    call    ExecuteHostCommand      ; decode and execute the command
    swapf   _statusRB0,w            ; restore registers w and status without affecting flags
    movwf   status
    swapf   _wRB0,f
    swapf   _wRB0,w
    bcf     INTCon,RB0IF            ; clear the RB0 interrupt flag
    bsf     INTCon,RB0IE            ; re-enable RB0 interrupts and ...
    bsf     INTCon,GIE              ; ... re-enable global interrupts
    retfie                          ; return from the RB0 interrupt
;
; ********************************************************************************************
; ********************************************************************************************
; Hard reset
; ********************************************************************************************
; ********************************************************************************************
;
HardReset
    bcf     INTCon,GIE              ; disable global interrupts (until later)
    bcf     INTCon,RB0IE            ; disable RB0 interrupts (until later)
    bcf     INTCon,Tmr0IE           ; disable Timer0 interrupts (until later)
    clrf    portA                   ; clear all I/O port latches
    clrf    portB
    clrf    portC
    movlw   0x00                    ; set T1CON=0 to disable Timer1 and ...
    movwf   T1CON                   ; ... release portC<1-0> for digital I/O
    movlw   0x00                    ; set CCP1CON=0 to disable Capture/Compare/PWM and ..
    movwf   CCP1CON                 ; ... release portC<2> for digital I/O
    bsf     status,page0            ; select register bank 1
    bcf     status,page1
    movlw   0x06                    ; set ADCON1<3-1>=b'011' to ...
    movwf   ADCON1                  ; ... configure portA for digital I/O
    movlw   0x01                    ; configure portA<0> for input and portA<5-1> for output
    movwf   TRISA
    movlw   0x01                    ; configure portB<0> for input and portB<7-1> for output
    movwf   TRISB
    movlw   0x0F                    ; to avoid data bus conflict, configure portC<3-0> for input
    movwf   TRISC
    bsf     optionreg,7             ; <7>=1 disables PortB pull-up resistors
    bsf     optionreg,6             ; <6>=1 triggers RB0 interrupts on the rising edge
    bcf     optionreg,5             ; <5>=0 uses the internal clock to increment Timer0
    bcf     optionreg,4             ; <4>=0 increments Timer0 on low-to-high transitions
    bcf     optionreg,3             ; <3>=0 assigns the prescalar to the Timer0 module
    bsf     optionreg,2             ; <2-0>=111 sets the prescalar for Timer0 to 256:1
    bsf     optionreg,1
    bsf     optionreg,0
```

```
    bcf     status,page0            ; re-select register bank 0
    bcf     status,page1
InitializeProgramRegisters
    clrf    portAmirror
    clrf    portBmirror
    clrf    portCmirror
    clrf    number
    call    TurnOffLED
WaitForHostPCReady
    call    del500ms                ; wait one second before testing if the Host PC is ready
    call    del500ms
    movf    portB,w                 ; read the INTIn bit (do not use direct bit read)
    movwf   temp                    ; save in register temp
    btfsc   temp,INTIn              ; if the interrupt line is high, then wait for ...
    goto    WaitForHostPCReady      ; ... the Host PC to complete its initialization
BeginExecution
    bcf     INTCon,Tmr0IE           ; disable Timer0 interrupts
    bcf     INTCon,RB0IF            ; clear RB0 interrupt flag
    bsf     INTCon,RB0IE            ; enable RB0 interrupts
    bsf     INTCon,GIE              ; enable global interrupts
    goto    MainLoop
;
; ********************************************************************************************
; ********************************************************************************************
; Main Program
; ********************************************************************************************
; ********************************************************************************************
;
MainLoop
; The only activity of this test program is to toggle the LED on and off every three
; seconds and, at the same time as the LED is toggled on, to sent an incremented number to
; to the Host PC for display.  Replace this code with whatever you need to operate your
; circuit.
    call    TurnOnLED               ; turn on the LED
    bcf     INTCon,RB0IE            ; disable RB0 interrupts
    incf    number,f                ; increment register number and ...
    movf    number,w
    andlw   0x0F
    movwf   DataOut                 ; ... send it to the Host PC
    call    SendHostNibble
    bcf     INTCon,RB0IF            ; clear the RB0 interrupt flag and ...
    bsf     INTCon,RB0IE            ; ... re-enable RB0 interrupts
    call    del500ms                ; wait three seconds
    call    del500ms
    call    del500ms
    call    del500ms
    call    del500ms
    call    del500ms
    call    TurnOffLED              ; turn off the LED
    call    del500ms                ; wait another three seconds
    call    del500ms
    call    del500ms
    call    del500ms
    call    del500ms
    call    del500ms
    goto    MainLoop
;
; ********************************************************************************************
; Index of subroutines:-
;   GetHostNibble - reads DataIn<3-0> from the Host PC
;   SendHostNibble - sends DataOut<3-0> to the Host PC
;   ExecuteHostCommand - decode and execute a command received from the Host PC
;   StartTOTimer - set up and begin counting a communication timeout
;   StopTOTimer - stop the communication timeout timer
;   ConfigureDataBusForInput
;   ConfigureDataBusForOuput
;   TurnOnLED - turn on the test LED
;   TurnOffLED - turn off the test LED
;   del500ms - looped delay for 503.082ms
;   del10ms - looped delay for 10.0604ms
;   del100us - looped delay for 100.4us
; ********************************************************************************************
;
; ********************************************************************************************
; Subroutine GetHostNibble is called during the Interrupt Service Routine to read one
```

```
; nibble from the Host PC.
; Called with:-
;   interrupt request line (RB0) still high -- the Host PC has not been answered
;   RB0 interrupt flag still high -- it was not cleared by the ISR
;   RB0 interrupts disabled -- they were disabled by the ISR
;   interrupt acknowledge line (RB1) low
;   data bus portC<3-0> configured for input (but, for certainty, re-configure for input)
; Returns with:-
;   nibble received from the Host PC in DataIn<3-0>
;   interrupt request line (RB0) low
;   interrupt acknowledge line (RB1) low
;   data bus portC<3-0> configured for input
;   RB0 interrupt flag still high
;   RB0 interrupts still disabled
; Communication timeouts:-
;   If reception times out, then the procedure:
;   1. explicitly asserts low the interrupt output line to the Host PC,
;   2. stops Timer0 interrupts and
;   3. returns with DataIn = b'11111111' (an invalid command).
;
GetHostNibble
    call    StartTOTimer            ; begin one-second communication timeout test
    call    ConfigureDataBusForInput
    bsf     portAmirror,INTOut      ; acknowledge the Host PC's interrupt
    movf    portAmirror,w
    movwf   portA
GHN1
    movf    CommTO,w                ; check value of register CommTO
    btfsc   status,zero             ; if CommTO counts to zero, then the zero flag is set ...
    goto    CommTimeoutRecv         ; ... and the communication has timed out
    movf    portB,w                 ; read INTIn bit (do not use direct bit read)
    movwf   temp
    btfsc   temp,INTIn              ; wait until the Host PC asserts the interrupt line low
    goto    GHN1
    movf    portC,w                 ; read the data bus
    andlw   0x0F                    ; keep only the low-order nibble
    movwf   DataIn                  ; save the nibble into register DataIn
    bcf     portAmirror,INTOut      ; assert the acknowledge line low
    movf    portAmirror,w
    movwf   portA
    call    StopTOTimer
    return
CommTimeoutRecv
    movlw   0xFF
    movwf   DataIn
    bcf     portAmirror,INTOut      ; assert the acknowledge line low
    movf    portAmirror,w
    movwf   portA
    call    StopTOTimer
    return
;
; ********************************************************************************************
; Subroutine SendHostNibble is the complete procedure to send one nibble to the Host PC.
; This subroutine can be called from either the main program or the ISR.
; Called with:-
;   nibble to send is in register DataOut<3-0>
;   RB0 interrupts disabled (but, for certainty, re-disable RB0 interrupts)
;   Timer0 interrupts enabled
;   interrupt flags unknown
;   interrupt request line (RB0) should be low, but may not be
;   interrupt acknowledge line (RB1) low
;   data bus portC<3-0> configured for input
; Returns with:-
;   RB0 interrupts disabled
;   Timer0 interrupts enabled
;   interrupt flags unknown
;   interrupt request line (RB0) low
;   interrupt acknowledge line (RB1) low
;   data bus portC<3-0> configured for input
;   DataOut<3-0> is unchanged
; Communication timeouts:-
;   If transmission times out, then the procedure:
;   1. explicitly asserts low the interrupt output line to the Host PC,
;   2. stops Timer0 interrupts,
;   3. configures the data bus for input and
```

```
;   4. returns.
;
SendHostNibble
    call    del100us                ; delay 100us to ensure Host PC is finished with ...
                                    ; ... previous communication cycle, if any
    bcf     INTCon,RB0IE            ; disable RB0 interrupts but keep Timer0 enabled
    call    StartTOTimer            ; begin one-second communication timeout test
SHN1
    movf    CommTO,w                ; check value of register CommTO
    btfsc   status,zero             ; if CommTO counts to zero, then the zero flag is set ...
    goto    CommTimeoutTransmit     ; ... and the communication has timed out
    movf    portB,w                 ; read INTIn bit (do not use direct bit read)
    movwf   temp
    btfsc   temp,INTIn              ; do not start until the Host PC is ready (INTIn low)
    goto    SHN1
    bsf     portAmirror,INTOut      ; send interrupt request to the Host PC
    movf    portAmirror,w
    movwf   portA
SHN2
    movf    CommTO,w                ; check value of register CommTO
    btfsc   status,zero             ; if CommTO counts to zero, then the zero flag is set ...
    goto    CommTimeoutTransmit     ; ... and the communication has timed out
    movf    portB,w                 ; read INTIn bit (do not use direct bit read)
    movwf   temp
    btfss   temp,INTIn              ; wait until the Host PC acknowledges the interrupt
    goto    SHN2
    call    ConfigureDataBusForOutput
    movf    portCmirror,w           ; zero-out the low-order nibble of portC
    andlw   0xF0
    movwf   portCmirror
    movf    DataOut,w               ; keep only the low-order nibble of register DataOut
    andlw   0x0F
    iorwf   portCmirror,w           ; or-in the high-order nibble of portC
    movwf   portCmirror             ; save changes in portCmirror
    movwf   portC                   ; present data on the data bus
    call    del100us                ; delay 100us for circuits to settle
    bcf     portAmirror,INTOut      ; assert the interrupt request line low
    movf    portAmirror,w
    movwf   portA                   ; No delay is needed for portA.  The following steps ...
                                    ; ... wait for an acknowledgement from the Host PC.
SHN3
    movf    CommTO,w                ; check value of register CommTO
    btfsc   status,zero             ; if CommTO counts to zero, then the zero flag is set ...
    goto    CommTimeoutTransmit     ; ... and the communication has timed out
    movf    portB,w                 ; read INTIn bit (do not use direct bit read)
    movwf   temp
    btfsc   temp,INTIn              ; wait until the Host PC ackowledges receipt of the data
    goto    SHN3
    call    StopTOTimer
    call    ConfigureDataBusForInput
    return
CommTimeoutTransmit
    bcf     portAmirror,INTOut      ; assert the acknowledge line low
    movf    portAmirror,w
    movwf   portA
    call    StopTOTimer
    call    ConfigureDataBusForInput
    return
;
; ********************************************************************************************
; Subroutine ExecuteHostCommand decodes and executes a command received from the Host PC
;
ExecuteHostCommand
    movf    DataIn,w
    xorlw   0x01
    btfsc   status,zero             ; if DataIn=b'0001', then the zero flag is set and ...
    goto    Command1                ; ... the Host PC has sent Command1
    movf    DataIn,w
    xorlw   0x02
    btfsc   status,zero             ; if DataIn=b'0010', then the zero flag is set and ...
    goto    Command2                ; ... the HostPC has sent Command2
    movf    DataIn,w
    xorlw   0x03
    btfsc   status,zero             ; if DataIn=b'0011', then the zero flag is set and ...
    goto    Command3                ; ... the Host PC has sent Command3
```

```
    movf    DataIn,w
    xorlw   0x04
    btfsc   status,zero             ; if DataIn=b'0100', then the zero flag is set and ...
    goto    Command4                ; ... the HostPC has sent Command4
    movf    DataIn,w
    xorlw   0x05
    btfsc   status,zero             ; if DataIn=b'0101', then the zero flag is set and ...
    goto    Command5                ; ... the Host PC has sent Command5
    movf    DataIn,w
    xorlw   0x06
    btfsc   status,zero             ; if DataIn=b'0110', then the zero flag is set and ...
    goto    Command6                ; ... the Host PC has sent Command6
    movf    DataIn,w
    xorlw   0xFF                    ; if DataIn=b'11111111', then the zero flag is set ...
    btfsc   status,zero             ; ... and the communication timed out
    goto    ReceptionTimedOut
    goto    InvalidCommand          ; the program only recognizes six commands
Command1
    movlw   0x0E                    ; the program sends 0x0E in response to Command1
    movwf   DataOut
    call    SendHostNibble          ; send the reply to the Host PC
    return
Command2
    movlw   0x03                    ; the program sends 0x03 in response to Command2
    movwf   DataOut
    call    SendHostNibble
    return
Command3
    call    del500ms                ; the program waits one second in response to ...
    call    del500ms                ; ... to Command3, then sends 0x05
    movlw   0x05
    movwf   DataOut
    call    SendHostNibble
    return
Command4
    call    TurnOnLED               ; the program turns on the LED in response to Command4
    return
Command5
    call    TurnOffLED              ; the program turns off the LED in response to Command5
    return
Command6
    movf    portA,w                 ; Command6 causes the program to report the switch status
    movwf   temp                    ; read the switch voltage (do not use direct bit read)
    clrf    DataOut                 ; set default value of DataOut=d'0', meaning closed
    btfsc   temp,SW
    incf    DataOut,f               ; if portA<SW> is high, set DataOut=d'1', meaning open
    call    SendHostNibble
    return
;
; ********************************************************************************************
; Procedure RecptionTimedOut
; You must decide what you want your PIC to do if it fails to read command.
; This program simply ignores the failure.
ReceptionTimedOut
    return
;
; ********************************************************************************************
; Procedure InvalidCommand
; You must decide what you want your PIC to do if it receives an invalid command.
; This program simply sends 0x0C to the Host PC.
InvalidCommand
    movlw   0x0C
    movwf   DataOut
    call    SendHostNibble
    return
;
; ********************************************************************************************
; Subroutine StartTOTimer initializes the one-second communication timeout register
;
StartTOTimer
    movlw   0x26                    ; 0x26=d'38' and ...
    movwf   CommTO                  ; ... 38 x 26.2144ms = 996.1ms
    clrf    timer0                  ; reset register timer0
    bcf     INTCon,Tmr0IF           ; clear Timer0 interrupt flag
    bsf     INTCon,Tmr0IE           ; enable Timer0 interrupts
```

```
    return
;
; ********************************************************************************************
; Subroutine StopTOTimer ends the one-second communication timeout register
;
StopTOTimer
    bcf     INTCon,Tmr0IE           ; disable Timer0 interrupts
    bcf     INTCon,Tmr0IF           ; clear Timer0 interrupt flag
    return
;
; ********************************************************************************************
; Subroutine ConfigureDataBusForInput configures portC<3-0> for input
;
ConfigureDataBusForInput
    bsf     status,page0            ; select register bank 1
    bcf     status,page1
    movlw   0x0F
    movwf   TRISC
    bcf     status,page0            ; re-select register bank 0
    bcf     status,page1
    call    del100us                ; delay 100us for circuits to settle
    return
;
; ********************************************************************************************
; Subroutine ConfigureDataBusForOutput configures portC<3-0> for output
;
ConfigureDataBusForOutput
    bsf     status,page0            ; select register bank 1
    bcf     status,page1
    movlw   0x00
    movwf   TRISC
    bcf     status,page0            ; re-select register bank 0
    bcf     status,page1
    call    del100us                ; delay 100us for circuits to settle
    return
;
; ********************************************************************************************
; Subroutine TurnOnLED turns on the test LED
;
TurnOnLED
    bsf     portAmirror,LED
    movf    portAmirror,w
    movwf   portA
    return
;
; ********************************************************************************************
; Subroutine TurnOffLED turns off the test LED
;
TurnOffLED
    bcf     portAmirror,LED
    movf    portAmirror,w
    movwf   portA
    return
;
; ********************************************************************************************
; Subroutine del500ms delays for 503.082ms
;                                   ; Cycle count
del500ms
    movlw   0x32                    ; 1  (Note that 0x32 = d'50'.)
    movwf   count3                  ; 1
del500msa
    call    del10ms                 ; 50 x 25,151 = 1,257,550
    decfsz  count3,f                ; (49 x 1) + (1 x 2) = 51
    goto    del500msa               ; 49 x 2 = 98
    return                          ; 2
;                                   ; CALL adds 2 -- 1,257,705 cycles x 400ns = 503,082,000ns
;
; ********************************************************************************************
; Subroutine del10ms delays for 10.0604ms
;                                   ; Cycle count
del10ms
    movlw   0x63                    ; 1  (Note that 0x63 = d'99'.)
    movwf   count2                  ; 1
del10msa
    call    del100us                ; 99 x 251 = 24,849
```

```
    decfsz  count2,f                ; (98 x 1) + (1 x 2) = 100
    goto    del10msa                ; 98 x 2 = 196
    return                          ; 2
;                                   ; CALL adds 2 -- 25,151 cycles x 400ns = 10,060,400ns
;
; ********************************************************************************************
; Subroutine del100us delays for 100.4us
;                                   ; Cycle count
del100us
    movlw   0x52                    ; 1  (Note that 0x52 = d'82'.)
    movwf   count1                  ; 1
del100usa
    decfsz  count1,f                ; (81 x 1) + (1 x 2) = 83
    goto    del100usa               ; 81 x 2 = 162
    return                          ; 2
;                                   ; CALL adds 2 -- 251 cycles x 400ns = 100,400ns

    END                             ; end assembly
```