```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; lab01.asm
;
; Revised: 1/2/2012
; Revised: 1/15/2013
;
; Dan Simon
; Rick Rarick
;
; Cleveland State University
;
; This program blinks the LED that is connected to the RD2 pin
; every half second, assuming a 3.6864 MHz oscillator frequency.
; This illustrates how precise timing intervals can be
; generated using instruction counting. This is a "brute force"
; method for generating timing intervals. Later we will use one
; of the PIC's built-in timers and an interupt to achieve the
; same result.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; MPLAB IDE usage note:
;
; Occasionally, MPLAB can act kind of buggy. The software usually
; works fine, but sometimes you will get strange errors, or MPLAB
; will stop working, so you will have to shut down MPLAB and restart it,
; or you may even have to reboot your PC. I think your experience
; with MPLAB will be positive – just be aware that it is not
; flawless. It’s like most old (and a lot of new) Windows
; software – if it starts acting strange, reboot.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Assembler Directives
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Directives are assembler commands that appear in the source code
; and tell the assembler HOW to assemble the code. They are used to
; control the input, output, and data allocation of
; the assembler. Refer to Chapter 4 of the MPASM Assembler User's
; Guide for more information. Directives are not case-sensitive.

    list    p = 16F877

; The 'list' directive instructs the assembler to create a list file
; which contains detailed disassembly information. The 'p' option of
; the 'list' directive configures the assembler for the correct
; processor type. The MPLAB IDE automatically creates this file by
; default, but putting the directive in the source code insures that
; the assembler is configured for the correct processor in other
; environments. In the  MPLAB IDE, the processor type is set in the
```

```
; menu under Configure/Device.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    #include    <P16f877.inc>

; The #include directive tells the assembler to include an additional
; source file in the current source file. The P16F877.inc file is the
; processor-specific definition file for the 16F877 processor. It
; must be included in the source code to resolve the address of
; mnemonic symbols such as PORTA. You can view a copy of this file
; on the course website in the Resources/PIC16F877_Datasheet folder.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

config_1    EQU     _FOSC_XT   & _WDT_OFF  & _PWRTE_OFF & _CP_OFF

config_2    EQU     _BODEN_OFF & _LVP_OFF  & _CPD_OFF   & _WRT_ON

    __config    config_1 & config_2

; The '__config' directive (double underscore) instructs the
; assembler to set certain processor configuration bits in the
; configuration register located at 2007h in program memory (See page
; 119 in the PIC datasheet (DS) and page 64 in the MPLAB User guide ).
; In the [Configure -> Configuration bits] menu, the checkbox for
; "Configuration bits set in code" should be checked. Do not change
; settings in this window unless you understand what the settings
; control.
;
; The configuration bits are used to select various device
; configurations such as enabling/disabling the watchdog timer. The
; 14-bit configuration word is determined by AND-ing the appropriate
; mnemonic symbols defined in the P16F877.inc file. The mnemonics
; in the above __config directive determine the following
; configurations (with single underscores):
;
; _FOSC_XT   ->  XT (Crystal) Oscillator selected
; _WDT_OFF   ->  Watchdog timer disabled
; _PWRTE_OFF ->  Power-up timer disabled
; _CP_OFF    ->  Program Memory (Flash EEPROM) protection off
; _BOREN_OFF ->  Brown-out Reset disabled
; _LVP_OFF   ->  Low voltage programming disabled
; _CPD_OFF   ->  Data Memory (EEEPROM) Code Protection disabled
; _WRT_ON    ->  Program Memory (Flash EEPROM) may be written to by EECON

; __config = 3739h = 11 0111 0011 1001 for the above configuration

; See also Page 27-7 in the Mid-Range Family Reference Manual for
; more symbol definitions.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
; cblock - General purpose data memory (RAM) allocations
;
; The cblock ( "constant block" ) and endc directives are used to reserve
; or allocate a block of general purpose data memory (page 13 in the DS)
; for user-defined variables beginning at a specifed address. Each
; variable is aassigned an adress (constant).

    cblock 0x20     ; Variable assignments start at RAM address 20h.
                    ; This address is in Bank0, the default bank.

        Count       ; Address in Data Memory - 0x20
        CountInner  ; Address in Data Memory - 0x21
        CountOuter  ; Address in Data Memory - 0x22

    endc

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Start of executable code
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    org     0x0000      ; 0000h is called the reset address or reset
                        ; vector. Upon reset, the program counter
                        ; (PC) jumps to the first address (0x0000)
                        ; in program memory and begins executing the
                        ; instructions that follow.

                        ; The org (origin) directive specifies the
                        ; starting address in program memory for any
                        ; instructions following the directive. It is
                        ; used when instructions need to be placed at
                        ; a particular address in program memory, such
                        ; as at the beginning of the program or after
                        ; an interrupt.

    nop                 ; This is the first instruction after a reset
                        ; operation, and it is located at 0000h in program
                        ; memory. The nop (no operation) instruction tells
                        ; the processor to do nothing and advance the PC to
                        ; the next instruction at 0001h. The nop instruction
                        ; is used as a  short delay (about 1 us for a 4 MHz
                        ; oscillator) to allow electronic tasks
                        ; in the processor to complete after a reset before
                        ; executing the next instruction. It is also used
                        ; for backward compatibility with older in-circuit
                        ; debuggers.

    goto    INIT        ; The 'goto' intruction forces the Program Counter
                        ; to skip the following instructions and jump to the
                        ; address in program memory with the label
                        ; 'INIT' below.
```

```
    org     0x0004      ; 0004h is the interrupt address. When the
                        ; processor receives an interrupt request,
                        ; the program counter jumps to this address
                        ; in program memory, and continues executing
                        ; instructions from that address onward. This
                        ; program does not use interrupts. We will
                        ; discuss interrupts later.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Program Initialization
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This section of code performs initialization of variables and
; registers used in the program. This can also be done in a
; subroutine if there are a large number of initializations.
; We put RD2 (the second bit in PORTD, p. 35 in DS)) in the
; output state by by setting the second bit of the tristate
; buffer register, TRISD<2>, to 0. Then we set the output of
; RD2 (PORTD<2> to 0.

INIT

    bcf         STATUS, RP1 ; Set bit 6 (RP1) equal to 0 and bit 5
                            ; (RP0) in the STATUS register to 1. We
                            ; use the following notation to indicate
                            ; this: STATUS<RP1:RP0> = 01.

    bsf         STATUS, RP0 ; These two instructions configure the
                            ; compiler to use Bank 1 in data memory
                            ; where the TRISD register is located
                            ; (see p. 18, DS).

    bcf         TRISD, 2    ; Configure the tristate buffer at pin
                            ; RD2 as an
                           output.


    bcf         STATUS, RP0 ; This returns STATUS<RP1:RP0> to 00,
                            ; so we are back in Bank 0.

    bcf         PORTD, 2    ; Initialize the output of RD2 to 0 V.
                            ; You can also use 'PORTD, RD2'.

; 'INIT' is called a label and is used to name an address in program
; memory. You can see this by opening the menu View/Pogram Memory
; and clicking on the 'Symbolic' tab.

; Labels usually start in column 1. They may be followed by a colon,
; space, tab, or an 'end of line' character. Labels must begin with
; an alpha character or an underscore and may contain alphanumeric
; characters, the underscore, and the question mark.
```

```
; Labels may not begin with two leading underscores, e.g., __config
; or begin with a leading underscore and number, e.g., _2NDLOOP
; or be an assembler reserved word (see “Reserved Words" in the MPASM
; User's Guide).

; Labels may be up to 32 characters long. By default
; they are case sensitive.

;----------------------------

; 'bcf' is the 'bit clear f' instruction. To 'clear' a bit
; means to force the bit to a value of logic '0'. To 'set'
; a bit means to force the bit to a value of logic '1'.
; The symbole 'f' is short for 'file'. The word 'file' is
; sometimes used in place of the word 'register'. This
; stems from the fact that in computer architecture a 'register
; file' is an array of registers, so an array containing only
; one register is also a register file, which can be abbreviated to
; 'file'.

; STATUS<RP1:RP0> Register Bank Select bits (used for direct
; addressing)

; 00 = Bank 0 (00h - 7Fh)
; 01 = Bank 1 (80h - FFh)
; 10 = Bank 2 (100h - 17Fh)
; 11 = Bank 3 (180h - 1FFh)

; The 'banksel' directive is another way to select a data memory
; bank. It tells the assembler to switch to the bank in which its
; argument is located. So in the code above,
;
; banksel   PORTD
;
; the assembler determines that PORTD is in Bank 0 and clears the
; STATUS RP1 and RP0 bits, that is, it changes both to 0. This
; eliminates the need to look up what bank PORTD is in.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Main Program Loop
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Toggle the LED on and off every half second
; for a 4 MHz (nominal) oscillator.

MAIN

        bsf     PORTD, 2        ; pin RD2 = high = led on

        call    DELAY_500ms
```

```
        bcf     PORTD, 2        ; pin RD2 = low = led off

        call    DELAY_500ms

        goto    MAIN

; 'call DELAY_500ms' causes the program counter to jump to the
; 'DELAY_500ms' label in program memory. The PC executes the
; instructions following the label until it reaches a 'return'
; instruction at which point the PC jumps back to the address
; following the 'call' instruction. The DELAY_500ms label
; indicates the beginning of a subroutine which inserts a
; delay of 500 milliseconds into the execution of code before
; the subroutine is exited.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; End of Main Program Loop
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; DELAY_500ms Subroutine
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; A precise timing interval can be generated using “brute force”
; instruction counting.

DELAY_500ms

    movlw   d'50'           ; Move the "literal" value d'50'
    movwf   Count           ; into the W register. Then move
                            ; the contents of W into f which
    Loop                    ; is the 'Count' register.

    call    DELAY_10ms      ; Call DELAY_10ms 50 times = 500 ms

    decfsz  Count, F        ; 1 instruction
    goto    Loop            ; 2 instructions

    return                  ; Return the PC to the address following
                            ; the address of the 'call' instruction.

    ; The 'decfsz' instruction is a common method for creating
    ; loops in MPASM.

    ; decfsz    f, d <--> decrement f, skip if zero, put result in d.

    ; This means: Decrement the contents of register 'f'. If the
    ; decremented result is 0, skip the next instruction. Otherwise,
    ; execute the next instruction. If d = 0, put the decremented
    ; result in the working register W, otherwise, if d = 1, put the
```

```
    ; result in the 'f' register.
    ;
    ; Instead of using 1 and 0 for 'd', you can use the mnemonics
    ; F and W. So you can write either
    ;
    ; decfsz    Count, W
    ;
    ; or
    ;
    ; decfsz    Count, 0
    ;
    ; This means: Decrement the contents of the register 'Count'.
    ; If the result is zero, skip the next instruction. Put the
    ; decremented result in the working register W, and leave the
    ; contents of 'Count' unchanged.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; End of DELAY_500ms subroutine
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DELAY_10ms

    movlw   d'10'
    movwf   CountOuter

    OuterLoop

        movlw   d'230'                  ; 1 instruction
        movwf   CountInner              ; 1 instruction

        ;----------------------------------------

        InnerLoop

            nop                         ; 1 instruction
            decfsz  CountInner, F       ; 1 instruction
            goto    InnerLoop           ; 2 instructions

            ; Approximate calculation: The InnerLoop code
            ; will execute 230 times for a total of
            ; 4 * 230 = 920 instructions.

            ; Else exit InnerLoop

        ;----------------------------------------

        decfsz  CountOuter, F       ; 1 instruction
        goto    OuterLoop           ; 2 instructions

        ; The OuterLoop code will execute 10 times for a total
        ; of 10 * ( 2 + 920 + 3 ) = 9250 instructions (approximately).
```

```
        ; This gives a delay of 9250 * 1.0851 = 10.037 ms. (See below)

        ; Else exit OuterLoop

    return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; End of DELAY_10ms subroutine
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    end     ; All programs must end with the 'end' directive. Note:
            ; The program continues to run after the 'end' directive!

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; For an oscillator frequency of 3.6864 * 10^6 Hz, the oscillator
; period (or clock cycle period) is Tosc = 1 / 3.6864 * 10^(-6) =
; 0.27127 us. Each instruction requires 4 oscillator periods, so the
; instruction cycle period is Tcy = 4 * 0.27127 us = 1.0851 us.
; A 10 ms delay will require 10 ms / 1.0851 us = 9217 instructions.
; See page 5-4 in the PIC Mid-Range Reference Manual (MRM) for more
; information on instruction cycles.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Summary of case-sensitivity:

; Directives   - No
; Instructions - No
; Labels       - Yes
; Variables    - Yes
; Special Function Registers - Yes
```