Tutorial 7 – Flashing an LED

Objectives:

- Configure the SFRs
- Use basic assembly language instructions (*movlw, movwf, clrf, goto*):
  - To write to *PORTS* to turn an LED on and off, repeatedly
- Simulate code using *MPLAB SIM* before flashing code to PIC
  - Adding the *Stopwatch* window and breakpoints to verify the timing of a delay
- Program your PIC using the *PICKit3*

Procedure:

1. Create new project:
   a. Create a new project and workspace called *asmLED_Flash* in a folder with the same name
   b. Open *asmTemplate.asm* from the Tutorial #1 and re-save it as *asmLEDFlash_A.asm*.
   c. Add the file to the project's source code
2. The Delay header file:
   a. After the ***INCLUDE "p16f684.inc"*** add: ***INCLUDE "asmDelay.inc"*** on the next line. Locate and open the file. This file contains code that allows the programmer to specify a delay in microseconds. For example - ***Dlay 1500000*** – represents 1.5 seconds.
   b. To allow us to use it in multiple programs, we need to save it with the other header files in the following path: ***C:\Program Files\Microchip\MPASM Suite*****.**
   If this is not possible for you, save it in the *asmLED_Flash* folder and add it to *HEADER* within the MPLAB program.
   c. Close the file after you've saved it in the new location.
3. Modify Code:
   a. Modify the code to match what is shown on *asmLEDFlash_A.asm* on the next page.
   b. Build all (Cntrl + F10) and rearrange the windows as shown below.

Configuring SFRs:

- Before we can begin to use the PIC to send out digital 1s and 0s, we need to first configure some of the SFRs (Special Function Registers).
  - turn off comparators
  - switch all ports from analog to digital
  - Teach each ports to be inputs and/or outputs
- CMCON0:
  - This code is simply disabling the internal comparators so that it doesn't interfere with your program.
- ANSEL:
  - This SFR is in Bank 1. Therefore, to access it, you must move to Bank 1 (set RP0 (bit 5) in your STATUS Register). This is called BANK SWITCHING.
  - This PIC has 8 analog channels which share with the digital ports. The problem is, the PIC defaults to analog. Thus, you must specify that you're using digital from the start. Hence, the code.
- TRISC:
  - All 12 ports in the P16F684 are bi-directional (with the exception of RA3 which can never be an output)
  - To teach a port to be an input, you set (1) that bit in the TRIS reg
  - To teach a port to be an output, you clear (0) that bit in the TRIS reg

- In our example, by clearing the TRISC reg, you're making all the ports in PORTC (RC0:RC5) outputs.

• STATUS:

- We must return to Bank 0 so that we can write to PORTC. To accomplish this we clear (0) RP0. This is called BANK SWITCHING!

```
title "asmLEDFlash_A"
;
; Write to PORTC (RC0) with a delay which makes one LED flash
; Use bsf, bcf instructions
;
; Hardware Notes:
; -P16F684 running at 4MHz
; -Output:
; - RC0: one LED through a 680 resistor
;
; author
; date
;-------------------------------------------------------------------------------------------
; Setup
LIST r=dec
INCLUDE "p16f684.inc"
INCLUDE "asmDelay.inc" ;new header file for the delay
__CONFIG _FCMEN_OFF & _IESO_OFF & _BOD_OFF & _CPD_OFF & _CP_OFF & _MCLRE_ON & _PWRTE_ON &
_WDT_OFF & _INTOSCIO ; all on one line
; Variables
CBLOCK 0x020
ENDC
;-------------------------------------------------------------------------------------------
PAGE
;Mainline of asmLEDFlash_A
org 0
movlw 7             ; Turn off Comparators
movwf CMCON0        ; The decimal value '7' is '00000111' in binary. When this value is moved to
                    ; CMCON0, the last three bits turn off the comparators.

bsf STATUS, RP0     ; Switching to BANK 1
clrf ANSEL ^ 0x080  ; Clearing ANSEL makes all ports digital

clrf TRISC ^ 0X080  ; Teach all of PORT C to be outputs
bcf STATUS, RP0     ; Switching to BANK 0

again               ; Loop continuously turning LED ON and OFF
nop                 ; No Operation
bsf PORTC,0         ; Turn ON LED
Dlay 2000000        ; Delay of 2 seconds
nop
bcf PORTC,0         ; Turn OFF LED
Dlay 2000000
goto again
end
```

4. Setup *MPLAB SIM*
    a. Open *MPLAB SIM*
    b. Open the *Watch* window. Move it to the bottom right-hand corner of the screen.
    c. Add the following SFRs to the *Watch* window:
        • STATUS
        • CMCON0
        • ANSEL
        • TRISC
        • WREG
        • PORTC
5. Setup the *Stopwatch* window:
    a. Select *Debugger>Stopwatch*
    b. Select *Debugger>Settings…*
    c. Change the *Processor Frequency* from 20 to 4 MHz, which is what the P16F684's internal oscillator defaults at. Click *OK.*
    d. Move the *Stopwatch* window to the right center side of the screen.
6. Verify values in SFRs:
    a. Click on the *"Step Into"* button for *MPLAB SIM* and verify the contents of the SFRs in the *Watch* window
    b. Stop when you get to the first ***nop*** instruction.
    c. You should notice that ***RC0*** in PORTC (i.e bit 0 in PORTC) has been "set". In other words, ***RC0*** is outputting or 'writing' a HIGH, which turns the LED ON.

7. Add *Breakpoints:*
    a. Breakpoints allow us to "Run" from one breakpoint to another, rather than stepping one instruction at a time. Due to the fact that we now have delays in our program which represent millions of instructions, it might take you some time to step through that!!!
    b. To add a breakpoint, double-click next to the ***nop*** instructions, which stand for "no instruction", and which offers another way to insert breakpoints.
    c. A circled **B** will appear signifying a breakpoint (to remove a breakpoint, simply double-click again)

The *Stopwatch* window

- The *P16F684* contains an internal clock that is configured to operate at 4MHz , or 4,000,000 clock cycles/second which is why you changed this in Step 5 above. All PICs take 4 clock cycles to complete 1 instruction cycle, which equates to 1,000,000 instructions/sec if you're using a 4MHz clock speed (4,000,000/4). Creating a delay is simply a matter of keeping the processor busy looping nothing for a specific number of instructions – precisely 1,000,000 instructions per second.
- Take note that the *Stopwatch* Window has counted 6 instructions (or 6 microseconds). This should make sense as you have just simulated 6 lines of code. Most instructions in assembler utilize one instruction cycle, whereas program branches (i.e. *goto*, subroutine *calls*) require two instruction cycles. This is why assembler is considered to be far more efficient a program than high-level languages like 'C' or 'Basic', which require considerably more instruction cycles/line of code, and are thus slower to process, and require more program memory.

8. Simulate LED Flashing with *MPLAB SIM*:
    a. Recall the purpose of this program: turn an LED ON, then turn it OFF; repeatedly.
    b. Click on the *Zero* button on the *Stopwatch* window to reset the count back to ***0***.
    c. With your green PC pointer at the ***nop*** line, click on the "Run" command in the simulator toolbar. **You should observe two crucial pieces of information:**
        • The *Time* in the *Stopwatch* now shows ***2.000002*** seconds (the ***.000002*** represents the two instructions (***nop*** and ***bsf PORTC, 0***) executed in addition to the ***2s*** delay
        • The *Watch* window shows ***PORTC*** outputting a ***'000001`(RC0 is ON)*** which is exactly what you want, and represents the turning ON of the LED for those 2 seconds.
    d. Using MPLAB SIM to verify data flowing through registers BEFORE programming the PIC saves time in the long run because you're instantly aware that any problems must be hardware related (since the software has been verified).
9. Simulate and Remainder of Program
    a. Click on the *Zero* button on the *Stopwatch* window to reset the count back to ***0***.
    b. Using *MPLAB SIM*, click "Run" to the next breakpoint. Note the following:
        • *RC0* (bit 0 in PORT0) is now 0, which represents the turning OFF of the LED.
        • The *Time* in the *Stopwatch* now shows ***2.000004*** seconds (the ***.000004*** represents two instructions that require 1 instruction cycle (***nop*** and ***bsf PORTC, 0***) plus 1 instruction, ***goto***, requiring two instruction cycles (2+2=4 instruction cycles), plus the ***2s*** delay
        • The *Watch* window shows ***PORTC*** outputting a ***'000000`(RC0 is OFF)*** which is exactly what you want, and represents the turning OFF of the LED for those 2 seconds.

10. Setting up the *PICKit3:*
    a. The driver on the *Starter Kit CD* that comes with the *PICKit3* must be installed before you can use this programmer. It comes with the program, but occasionally it needs to be separately installed.
    b. Plug in the *PICKit3* to your PC's USB port.
    c. Connect it properly to your breadboard (the arrow on the programmer signifies the V_{PP} pin or "programming pin", which is wired to ***RA3*** (pin 4) of your PIC.
    d. Turn on your breadboard power supply .
    e. Select *Project>Select Programmer>PICKit3*. The output window will show *PICKit3* connected.
11. Program the PIC:
    a. *Project>Program* will allow the computer to program the device. The computer will give a notification informing about voltage warning, make sure that the microcontroller uses the same voltage as the selected one from the computer.
    b. Your should receive a message like the one below.
    c. Turn on the 5V battery pack, and watch your LED flash! If it's connected to a voltage regulator with a 9 volt battery, connect the battery and it will work!
    If there are any errors, refer to the information page!
    d. Can you make it flash faster??

Conclusion

1. Look below at the mainline for *asmFlashLED_B* (version B).

a. What has changed?

b. When would you use method A to write to PORTC?

c. When would you use method B to write to PORTC?

```
again ; loop continously turning LED ON and OFF
nop
movlw b'000001' ; turn ON LED
movwf PORTC
Dlay 2000000
nop ; turn OFF LED
movlw b'000000'
movwf PORTC
Dlay 2000000
goto again
end
```

2. How many ports are there on the P16F684?

3. Are all ports bi-directional?

4. What is the first bit# on all registers in this PIC? The last bit#?

5. What three things must you do before you write or read from any ports? Why?

6. How do you use bank switching and why is it necessary?

7. Why is MPLAB SIM such an important tool?

8. What does the IDE in MPLAB IDE stand for? What does it mean?

9. Why, when using the *Stopwatch* window, do we have to change the processor frequency from 20MHz to 4MHz?

10. How many clock cycles does 1 instruction cycle represent?

11. How many instruction cycles equals:

a. one tenth of a second?

b. one thousandth?