17.383 Microprocessors A
Lab 5
Fall 2010

# Assembly Language Introduction Using the PIC16F684 Microcontroller

## Objectives

1) To use MPLAB IDE software and the PICkit ™ 1 demonstrate the following:

    a. Develop Assembly language code such that an LED on the PICkit ™ 1 flashes
    b. Develop Assembly language code such that two LEDs on the PICkit ™ 1 flash
    c. Develop Assembly language code for digital input

## Materials

- ☐ 1 Computer (PC) with an available USB Port
- ☐ 1 MPLAB IDE and HI-Tech PICC Lite software installed on PC
- ☐ 1 PICkit ™ 1 with USB cable
- ☐ 1 PIC16F684 – supplied with PICkit ™ 1 Starter Kit
- ☐ 1 Oscilloscope

## WARNINGS AND PRECAUTIONS

1) Never remove the PIC16F684 from an energized circuit

2) Do not construct circuits while energized

3) Follow electrical safety precautions

## Source File Locations

1) HI-TECH Universal Toolsuite

   - HI-TECH ANSI C Compiler

     *C:\Program Files\HI-TECH Software\9.70\bin\picc.exe*

2) Microchip MPASM Toolsuite

   - MPASM Assembler (mpasmwin.exe)

     C:\Program Files\Microchip\MPASM Suite\mpasmwin.exe

   - MPLINK Object Linker (mplink.exe)

     C:\Program Files\Microchip\MPASM Suite\mplink.exe

   - MPLIB Librarian (mplib.exe)

     C:\Program Files\Microchip\MPASM Suite\mplib.exe

## Background Information

Thus far we have been developing code written in a high level language, C. We have performed various tasks utilizing the C language including flashing LEDs, monitoring for switch actions, analog to digital conversions, etc.

We will explore assembly language. The differences between C and assembly are in the basic statements and how they are used to create applications. The flow of the code is the same. Assembly language is no more difficult than C. Assembly code tends to be longer due to the need to do smaller tasks. C commands do many steps at a time. Assembly commands will do one small step at a time.

Assembly language utilizes an assembler, which converts an assembly language program into a file that will be utilized by the programmer to produce the actual code that will be "burned" into the device. The assembler is analogous to the compiler that converts C into the same type code. The assembler we will use is the MPASM, which is built into the MPLAB IDE. You should recall that PICC, which was what we used for C programs, was not built into MPLAB IDE.

The MPASM assembler is a full-featured, universal macro assembler for all PICmicro MCUs. The MPASM assembler features include:

- Integration into MPLAB IDE project
- User defined macros to streamline assembly code
- Conditional assembly for multi-purpose source files
- Directives that allow complete control over the assembly process

This lab introduces the assembly language by utilizing some of the past labs written in C.

## Pre-Lab Preparation

- Examine the provided code and understand how the commands are utilized
- Develop the required code as outlined in the procedural steps

## Procedure

### Experiment 1. CREATING MPASM PROJECT

**a.** In this portion of the Lab we will write code in assembly language to flash D0 on the PICkit 1 Starter Kit.

**b.** Start MPLAB IDE

**c.** Within MPLAB IDE, create a new file by typing in the code provided in Figure 1.

**IMPORTANT – to understand how the code is constructed, which columns it needs to be placed in, etc, you will retype the code. Do not copy and paste.** For convenience, the Code is also located on the Course Web page as Flash_D0.txt

**d.** Save the file as Lab_5A.asm

**e.** Create a new project. Call the project Lab_Five and put it in the following directory:

C:\Student Data Area Drive C

**f.** Set the Language Toolsuite to:

Microchip MPASM Toolsuite

**g.** Add Lab_5A.asm as the Source File

**h.** Build the project. You should receive a Build Succeeded message. If not, go back and determine the cause of the error(s) by looking at the error messages

> **NOTE:** You **will receive** the following warning
>
> Message[302] C:\STUDENT DATA AREA DRIVE C\FLASH_D0.ASM 48 : Register in operand not in bank 0. Ensure that bank bits are correct.
>
> This message is just a warning and is not a problem. The following is the definition of the message (From MPASM Assembler Guide):
>
> > **302 Register in operand not in bank 0. Ensure that bank bits are correct.**
> >
> > This is a commonly seen reminder message to tell you that a variable that is being accessed in not in bank 0. This message was added to remind you to check your code, particularly code in banks other than 0. Review the section on banksel (**Section 4.7 "banksel – Generate Bank Selecting Code"**) and bankisel (**Section 4.6 "bankisel – Generate Indirect Bank Selecting Code (PIC12/16 MCUs)"**) and ensure that your code uses bank bits whenever changing from ANY bank to ANY other bank (including bank 0).
> >
> > Since the assembler or linker can't tell which path your code will take, you will always get this message for any variable not in bank 0. You can use the errorlevel command to turn this and other messages on and off, but be careful as you may not spot a banking problem with this message turned off. For more about errorlevel, see **Section 4.29 "errorlevel – Set Message Level"**.
> >
> > A similar message is 306 for paging.

**i.** Use the animate feature of the simulator. Watch the Code execute paying particular attention to the delay routines.

**j. QUESTION:** What is the actual delay time?

**k.** After you verify that the code is executing properly. Connect the PICkit to the computer and program the device.

**l.** The LED will appear as though it is always on due to the short delay that we have utilized.

**m.** **OPTIONAL**: To verify that the LED is in fact turning on and off, connect an oscilloscope to the circuit.

**n.** When the software and hardware are operating properly, notify the instructor so it can be observed.

**o.** Insert a copy of your code, Lab_5A.asm at the end of the lab report in the space provided as well as email a copy.

## Experiment 2. DELAY ROUTINE USING TMR0

**a.** The delay routine in Experiment 1 does not produce a sufficient amount delay so that visually we can see that the LED actually turns on and shuts off. In this section we will add a delay routine that is written around the use of the TMR0 timer of the PIC16F684.

**b.** Replace the current delay code in Lab_5A with the Code shown in Figure 2. Save this new Code as Lab_5B.asm.

**c.** Use the animate feature of the simulator. Watch the Code execute paying particular attention to the delay routines.

**d.** **QUESTION:** Using the features of the simulator, what is the actual delay time?

**e.** After you verify that the code is executing properly. Connect the PICkit to the computer and program the device.

**f.** The LED will now be flashing.

**g.** When the software and hardware are operating properly, notify the instructor so it can be observed.

**h.** Insert a copy of your code, Lab_5B.asm at the end of the lab report in the space provided as well as email a copy.

## Experiment 3. FLASHING TWO LEDs

**a.** Alter the assembly language code developed in Experiments 1 and 2 such that D3 and D7 flash (i.e. D3 will come on, go off, D7 comes on, goes off, and then repeat the sequence). You may need to add more delay to you code in order to see the alternating lights. Ensure that you include the header information as shown in Figure 3. Save your code as Lab_5C.asm

**b.** When the software and hardware are operating properly, notify the instructor so it can be observed.

**c.** Insert a copy of your code, Lab_5C.asm at the end of the lab report in the space provided as well as email a copy.

## Experiment 4. ASSEMBLY LANGUAGE AND DIGITAL SWITCH INTERFACE

**a.** Review the schematic in Figure 4 and determine how the PICKIT1 Starter kit pushbutton switch (SW1) is connected to the PIC16F684.

**b.** Create assembly language code that will perform as follows:

1. When the switch is off, all of the LEDs will be off.

2. When the switch is on, D5 LED will be on. Schematic for the LEDs is shown in Figure 5.

3. When the switch is release, D5 LED will go out.

4. The code will contain the following:

    a. Write the code in assembly language.
    b. The header information as shown in Figure 3
    c. Include the “p16f684.inc” file
    d. Include the proper configuration word at the proper location
    e. Initialize PORTA to zero
    f. Turn off the comparators
    g. Turn off ADCs

**c.** Save the program as “c:\ Student Data Area Drive C\Lab_5D.asm”

**d.** Burn the code into your PIC16F684 microcontroller and verify that your hardware is functioning as planned.

**e.** When the software and hardware are operating properly, notify the instructor so it can be observed.

**f.** Insert a copy of your code, Lab_5D.asm at the end of the lab report in the space provided as well as email a copy.

## Summary:

This lab provided an introduction to assembly language as well as an introduction to the Timer 0 feature of the PIC 16F684.

## Questions:

1. Answer the questions throughout the lab.

2. Ensure that you have recorded all the data requested during the lab as well as provide the requested printouts.

## Figure 1 – Flash_D0 Assembly Code

```
 title “Flash_D0.asm – Flash D0 LED”

; ----------------------------------
; Program File Name: Lab_XA.c
;
; Program Title:     xxxx
;
; Microprocessors A 17.383
;
; xxxxxxxx - Put in Semester (i.e. Fall 2010) here
;
; xxxxxxxx - Put in your name here
;
; xx/xx/xx - Put date here
;
; ----------------------------------

 list r=dec

#include "p16f684.inc"

 __CONFIG _FCMEN_OFF & _IESO_OFF & _BOD_OFF & _CPD_OFF & _CP_OFF & _MCLRE_OFF &
_PWRTE_ON & _WDT_OFF & _INTOSCIO

; ----------------------------------
; Defining constants or variables in data memory space

 Cblock 0x20                            ; start of GPR

count                                   ;

 endc                                   ; This directive must be
                                        ; supplied to terminate the
                                        ; cblock list
;
; ----- main program --------------

             org    0x000               ; Hex address 0x000, the first
                                        ; program memory location
;
start  BCF    STATUS,RP0                ; Select Bank 0
;
             CLRF   PORTA               ; Initialize PORTA (to all zeros)
;
             MOVLW  7                   ; Load w with 7
             MOVWF  CMCON0              ; Load CMCON0 with 7
                                        ; Turns off comparators
;
             BSF    STATUS,RP0          ; Select Bank 1
;
             CLRF   ANSEL               ; Shut off ADC (digital I/O)
;
             MOVLW  b'001111'           ; Load w – RA4 and RA5 outputs
             MOVWF  TRISA               ; copy w to TRIS PORTA
;
             BCF    STATUS,RP0          ; Select Bank 0
;
Loop         BSF    PORTA,4             ; Make RA4 high -- D0 ON
;
             CALL   Delay               ; Goto the delay routine
;
             BCF    PORTA,4             ; Make RA4 low -- D0 OFF
;
             CALL   Delay               ; Goto the delay routine
;
```

```
             GOTO   Loop               ; Do it again
;
; --- Start of delay routine ----------------------------
;Delay       MOVLW  10                 ; Decimal 10
;            MOVWF  count              ; Initialize counter to 10
;Repeat      DECFSZ count,f            ; Decrement counter
;            GOTO   Repeat             ; If counter <> 0
;            RETURN                    ; If counter = 0 (end delay)
;--- End of delay routine ---------------------------------
;
 end
```

## Figure 2 – TMR0 Delay Routine

```
; --- New Delay Routine using TMR0 ---------------------------
Delay  CLRF   TMR0                ; Clear Timer0 register, start counting
             CLRF   INTCON        ; Disable interrupts and clear T0IF
             BSF    STATUS, RP0   ; Bank1
             MOVLW  0xC7          ; PortB pull-ups are disabled,
             MOVWF  OPTION_REG    ; Interrupt on rising edge of RB0
                                  ; Timer0 increment from internal clock
                                  ; with a prescaler of 1:256
             BCF    STATUS, RP0   ; Bank0
             BSF    INTCON, T0IE  ; Enable TMR0 interrupt

again  btfss  INTCON,2             ; Bit 2 set?
             goto   again         ; No, bit is clear, goto again
             return

; --- End of new Delay routine -------------------------------
```

## Figure 3 Required Header Code

```
; ---------------------------------
; Program File Name: Lab_XA.c
;
; Program Title:     xxxx
;
; Microprocessors A 17.383
;
; xxxxxxxx - Put in Semester (i.e. Fall 2010) here
;
; xxxxxxxx - Put in your name here
;
; xx/xx/xx - Put date here
;
; ---------------------------------
```

**Figure 4 – PICkit 1 Starter Kit - Switch (SW1) Schematic**

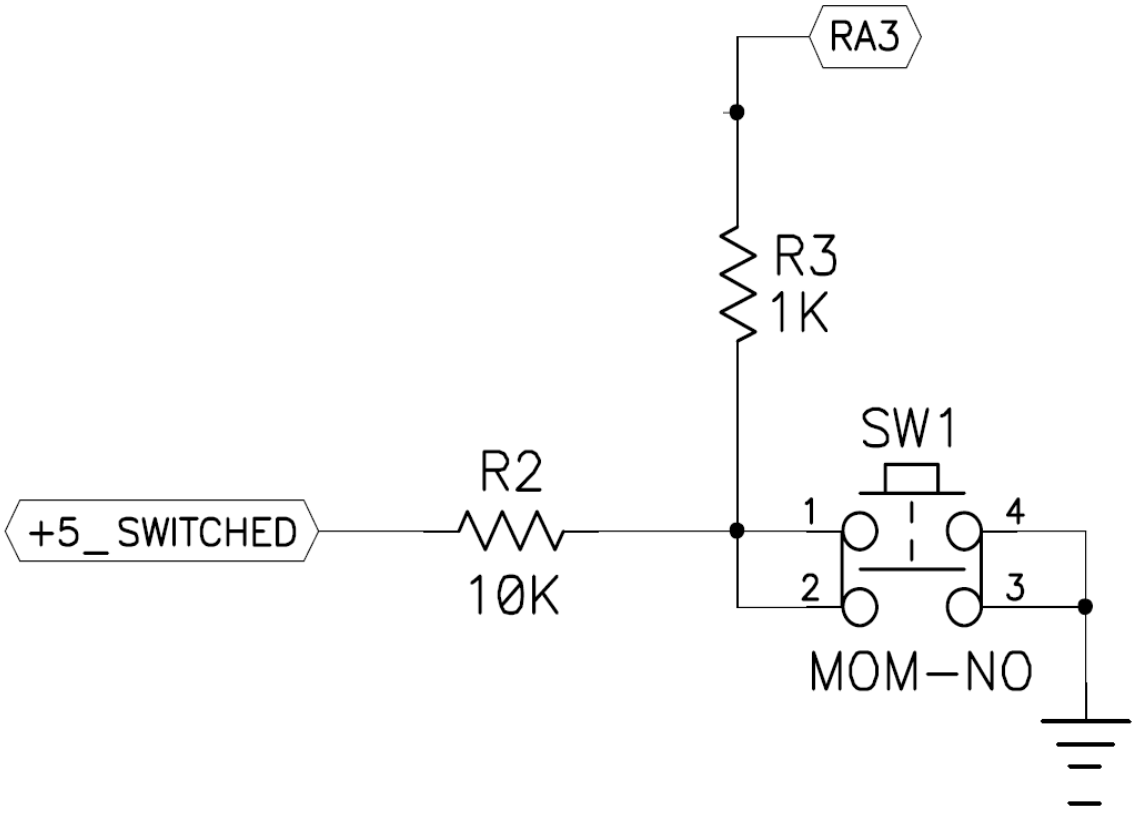

**Figure 5 - PICkit 1 Starter Kit LED Layout**