# Microprocessor-Based Systems (E155)

**Harris** **Fall 2004**

Lab 5: The Microcontroller Trainer Due: Week of Oct 11

## Requirement

1) *Get acquainted with the MPLAB IDE software.*
2) *Get acquainted with the PIC18F452 and its assembly language.*

## Discussion

You will be using the PIC18F452 for Labs 5, 6, and 7. In this lab you will get acquainted with chip, learn how to use its accompanying software and programmer, and write a simple assembly language program.

**PIC Assembly Language**

To familiarize yourself with the PIC18F452, do the following steps:

a) Read Sections 4.0-4.1 from the PIC datasheet on the PIC's memory organization, and skim Section 20 on the instruction set.

b) Read the "Getting to Know your PIC18F452" in Appendix A of this lab manual and try the examples to test that your PIC is working properly.

c) Write a program that, given five 1-byte numbers, will find the largest number and write it to Port D (to display to the LEDs). Assume that the numbers are in 2's complement.

d) Write a program that will sort 12 numbers. The numbers are given in 2's complement and are stored in data memory addresses $0000-$000B. The sorted results should be stored in the same block of memory with the smallest number in address $0000. The compiler will have generated a sort_xx.lst listing file in the directory that your project is stored in. Look at this listing file and print out the relevant section. You do not need to print out the symbol table, but scroll to the end of the .lst file to see your program's memory usage. Download your compiled code to the PIC and run it.

**Hints**

- Use the ORG directive to set the address of the beginning of your program to some place like 0.
- Use END to finish your code.
- You can reserve variables using the "NAME RES 0x0n" command at the beginning of your program (n is the number of bytes to reserve), or using "NAME EQU 0xnn". If you use RES, MPLAB will reserve variables starting at address 0x0000 in the data memory/file registers. If you want to access the file registers in a different manner (indirect addressing, for example), make sure the memory you are using does not conflict with the memory that MPLAB earmarks. If you use EQU, note that 'nn' is the address where you want the variable stored. You can also use EQU to define constants: "NAME EQU b'nn'" where 'b' is the base of the constant (binary, decimal, etc.) and 'nn' is the value of the constant.
- A negative result on a 2's complement subtraction does not always mean that the subtrahend is smaller—an overflow could have occurred. A 2's complement number 'a' is greater than a 2's complement number 'b' iff the XNOR of the negative and overflow bits is true. You may wish to verify this on your own.
- The latest version of the PIC development environment appears to corrupt projects fairly often, but does not damage the .asm file. If your project gets corrupted, make a new one and include the .asm file again.

**What to Turn In**

Your lab notebook for this lab may be very brief. It should include:

- A listing of your program that finds the greatest number.
- A listing of your program that sorts numbers.
- Explicitly state the test cases you used and the output of the tests on each program. Be sure your tests would convince a skeptic that your algorithm works.
- How many hours did you spend on the lab? This will not count toward your grade.

# Appendix A: Getting to Know your PIC18F452

## Software

We will be using the MPLAB Integrated Development Environment (IDE) software for writing assembly code and programming the PIC. This software includes an assembler, a simulator, and a hardware debugger. The simulator and hardware debugger have a very similar interface, so be sure you know when you are simulating a program and when you are actually executing it on the physical hardware.

Start the MPLAB software using the "MPLAB IDE" shortcut. Create a new project by selecting Project → New. Name your project "leds_xx" and put it in a folder on your Kato account. This will create a "leds_xx.mcp" file in that folder. Next you need to enter some code to run. Select File → New to create a new source file, and enter the following lines of code:

```
; leds.asm
; Written <date> by <your_name>@hmc.edu
; test PIC by turning off LEDs

; Use the 18F452 PIC microprocessor
   LIST  p=18F452
   include "p18f452.inc"

; allocate variables
LEDS     EQU   0x00

; begin main program
   org   0

start
   clrf  TRISD                  ; set PortD to output
   movff LEDS, WREG             ; LED output pattern
   movwf PORTD                  ; turn on LEDs

   end
```

Save the file as "leds_xx.asm" in the same directory. Insert the source file into your project by using Project → Add Files to Project and selecting leds_xx.asm. You can save your changes to the project by selecting Project → Save Project.

Now we are ready to translate the assembly code into machine code. You can either select Project → Build All, or you can use Ctrl+F10. A window will pop up and a couple green bars will scroll, but they will probably be gone before you can read what they

say. [If you are curious, you can play around with the MPASMWIN.EXE program in C:\Program Files\MPLAB IDE\MCHIP_Tools folder.] When the code has finished with assembly, a window labeled Output will appear, and you should see the following messages:

```
Deleting intermediary files... done.
Executing: "C:\Program Files\MPLAB IDE\MCHIP_Tools\MPASMWIN.EXE" /q
   /p18F452 "leds_xx.asm" /l"leds_xx.lst" /e"leds_xx.err"
Loaded H:\E155\lab 5\leds_xx.COD
BUILD SUCCEEDED
```

If you get any warning or error messages, go back and fix your code. When you are working with more complicated assembly language code, error or warning messages will appear in this box. Double-clicking on an error message will take you to the offending line of code.

If for some reason MPLAB can't find its assembler (the paths seem to occasionally get messed up), select Project → Set Language Tool Locations… Under the Microchip MPASM Toolsuite box, select Executables. Then click on MPASM Assembler (mpasmwin.exe). Click the browse button and browse to C:\Program Files\MPLAB IDE\MCHIP_Tools\MPASMWIN.EXE. Similarly, set the paths for mplib.exe and mplink.exe.

When you assembled your leds.asm file, MPLAB created an leds_xx.lst file in the same directory. Open it up with notepad or wordpad. Your assembled code is listed at the top of the file. The first column tells you what addresses each line of your code resides in. The second column gives the machine code. The third column numbers each line of code, and the fourth column gives the assembly code that you entered. Fill out the following table:

| Address | Machine Code | Assembly Command |
|---|---|---|
| 000000 | | |
| 000002 | | |
| 000006 | | |

Why does 'movff' occupy two words in memory? What is the machine code for the 'movff' command? Close the .lst file.

The next section of the list file contains the symbol table. It is not very interesting; it simply defines the internal registers used by the PIC. The last part shows the labels defined by your program and some statistics about the code.

Now we are ready to start debugging. Chose MPLAB SIM from the Debugger → Select Tool menu. This will simulate your code and allow you to debug it before actually running the code on the PIC. Use the Step Into command from the Debugger menu (this command will also step you through your code, one line at a time). Play around with a couple of the commands on this menu, because they will be helpful while debugging complicated assembly code.

While you are simulating, it is helpful to see what is going on going on in the chip's memory. The View menu lists a number of windows to watch the activity. The Program Memory window displays your assembly translated into machine code and stored in the PIC's program memory. Use this if you are ever curious about the instructions the machine is actually running. The File Register window displays the user memory available to you. You can change this user memory simply by typing new numbers into the display. You may find the Special Function Registers window helpful, and the Watch window is convenient for monitoring only the registers you want to look at.

Bring up the Watch and File Registers windows. Double-click on the first memory address in the File Registers window, 0x00, and type in '55'. After you change a value in memory, whether in the File Registers window or Watch window, the value will appear in red. This is helpful for pinpointing the effect of commands on particular registers while you are stepping through assembly code. Next, add TRISD and PORTD as registers in the Watch window. At the top of the window will be two buttons and two pulldown menus. Click on the first pulldown menu and type in "TRISD". Use the ADD SFR button to the left of the menu to display its value. Next, do the same for PORTD and WREG.

Now we are ready to debug some code. Chose Debugger → Reset → Processor Reset or press F6. A green arrow will appear in your code window, pointing to "clrf TRISD". This green arrow tells you where you are in your program. Chose Debugger → Step Into or press F7. The green arrow will move to the next line, and the TRISD value in the Watch window will become '00' and turn red. The next time your press F7, WREG will change to '55'. Finally, the value will be written to PORTD.

Once you have verified that your code works, you are ready to run it on the PIC.

**Hardware**

Apply power to your board from the bench power supply.

First make sure the software knows which PIC you are working with by selecting "PIC18F452" from the Configure → Select Device menu. The box has a list of seven programming tools that are compatible with the PIC chips. Check that MPLAB SIM and

MPLAB ICD 2 both have green lights before selecting OK. Next, go to Configure → Configuration Bits. This sets some of the hardware options on your PIC microprocessor. The clock source is the external clock coming from your jumpers to PIC pin OSC1. PIC pin OSC2 will be configured as a general purpose I/O port bit RA6. Therefore, set the **Oscillator** configuration option to "**EC-OSC2 as RA6**." For the ICD to be able to access the PIC memory and do in circuit debugging, you need to have "**Table Read Protect 00200-01FFF**" listed as "**Disabled**". If you want to disconnect the PIC from the ICD module and have your code run when the PIC is powered independently, you need to turn the Background Debug off (the Table Read Protect setting can remain disabled).

Once these options have been set, switch to running your hardware on the chip itself by selecting "MPLAB ICD2" from the Debugger → Select Tool menu. In the Output window, a new tab labeled MPLAB ICD2 will appear. You should see the following messages:

```
Connecting to MPLAB ICD 2
   ...Connected
Setting Vdd source to MPLAB ICD 2
Target Device PIC18F452 found, revision = b5
   ...Reading ICD Product ID
Running ICD Self Test
   ...Passed
MPLAB ICD 2 Ready
```

If you get a "ICDWarn0020: Invalid target device ID (expected=0x21, read =0x0)" message, you probably forgot to turn on the power supply. Turn on power and choose Debugger → Connect to try again. If you get a "ICD0083: Target not in debug mode, unable to perform operation" message, the ICD is having trouble talking to your PIC. Check that the clock is set to 1 or 2 MHz (not single step) and that the solder joints on the PIC and ICD socket are good. Be sure the configuration bits are set correctly.

You are finally ready to run your code on the PIC. Choose "Program" from the Debugger menu. After you have downloaded your code, you will be able to step through it just as you did with the simulator. View the Program Memory to watch the program counter step through the machine language as well. While you are in debug mode, you can change values in the PIC's data memory by changing the values in file registers. As you step through your program, MPLAB will highlight values in red that it has changed in the last step. Stepping is fairly slow because the ICD sends a large amount of information back and forth after each step. Having many windows open from the View menu slows

stepping even more. If you step to the end of the code, you may get a bunch of spurious "stepping target" messages and have to quit MPLAB.

If you want to run your entire program without stepping through it, choose Run (F9) and your code will run at full speed. You need to Halt the program (F5) before you can reset or re-program the PIC.