

Programming The PIC Micro Controller

Computer Organization
CS 140

The Lab - Overview

The purpose of this lab is to program a microprocessor. Actually, it's more about learning the whole programming process. You'll be doing only a tiny bit of programming.

There is nothing you have to hand in at the end of this lab, but you should be familiar with the programs that are provided by Microchip as exercises.

You could download and run ALL 12 – but that's probably overkill. I would recommend that you execute and understand:

- 01_Hello_World
- 02_Blink
- 03_Rotate
- 04_A2D
- 05_Variable_Rotate
- 06_Debounce
- 07_Reversible
- 09_Timer0
- 10_Interrupt

Know how to debug the code using MPLAB. The debugger is a wonder to behold – it allows you to do things like set breakpoints, single step, look at memory locations, etc. **YOU SHOULD KNOW HOW TO DO THESE OPERATIONS!**

Take one of these programs and change something. It could be a delay time, it could be something bigger – your choice.

The Lab - Overview

This write-up describes the Microchip “Low Pin Count Demo Board”, how to write assembler code, the process of getting code into the Demo Board and running that code.

This document contains the following:

1. A description of the tools and references you have available.
2. A description of the demo Board in words and pictures.
3. The Development Process.
4. How to write an assembly program.
5. Configuring the Development IDE (MPLAB)
6. Getting to perfect code.
7. Getting the Demo Board working.
8. Your deliverables.

Tools

So, what tools do you have available for this project?

1. Your brain – always the most important.
2. A PC running Windows – any Windows version works.
3. Editors (Windows types) as well as VI the “God of editors”.
4. A browser – you can use e-mail to get your files back and forth from the lab machines. SSH also works.
5. MPLAB IDE – MPLAB is a software product made by MicroChip, the manufacturers of the PIC processor. The IDE is an Integrated Development Environment that will make your life a lot easier. You can download it in the lab or at home by going to: [MPLAB Download](#)
6. Electronics (Pickit2) to download Hex code into the Demo Board. Also a debugger.
7. A C compiler. MinGW works well in this environment – but not needed this first week.

References

Where's a good source for more information – this document certainly only gives you an overview.

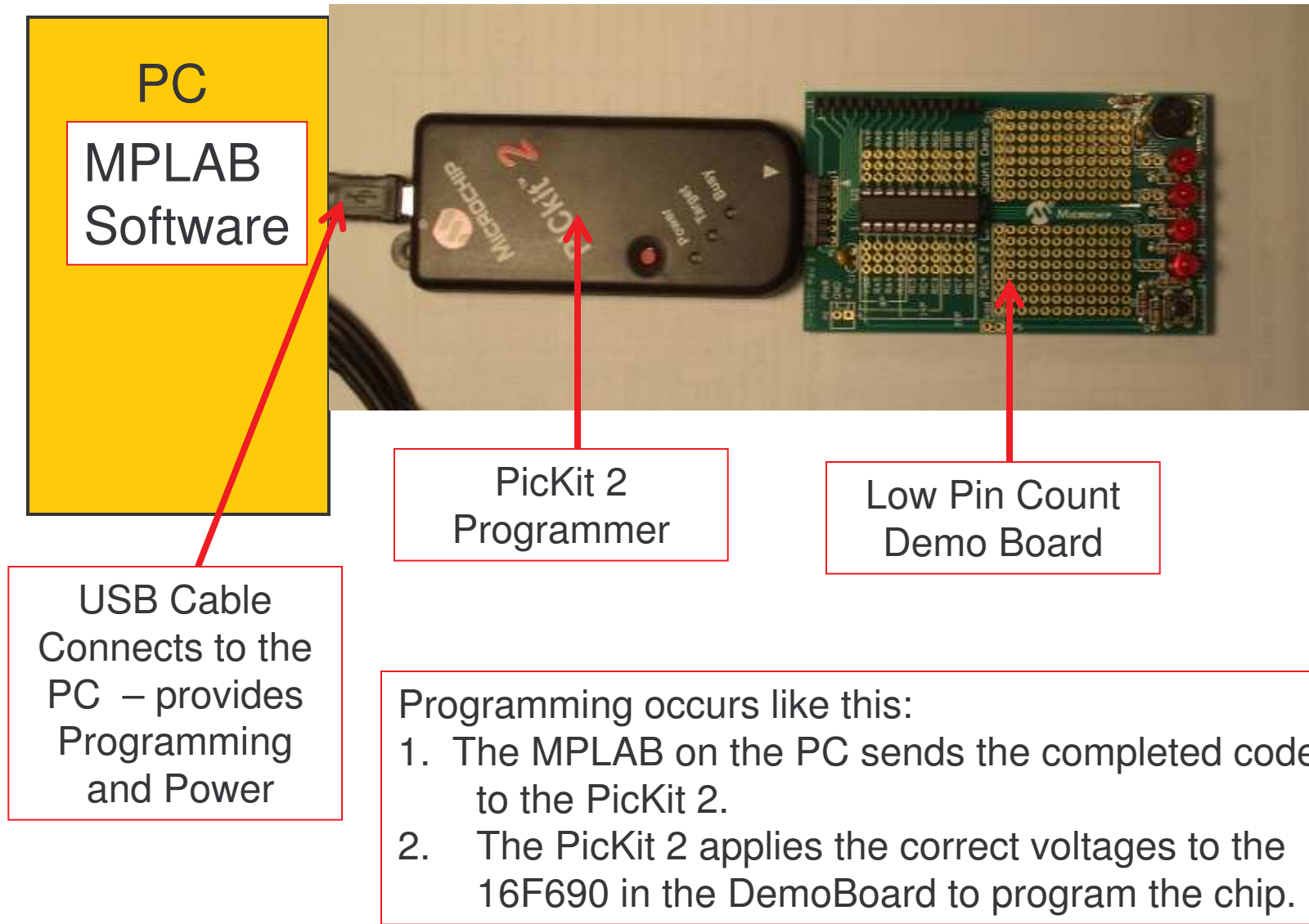
1. Documentation for the Demo Board, [Demo Board](#). It also includes an overview of the programs you will be executing. You got this on paper.
2. The code for the programs you will be running (so you don't have to type them in is at [Code](#).
3. The documentation for the chip we're using here, the [16F690](#). I have handed out a paper copy of a portion of this document.
4. There's an excellent [MPLAB Getting Started](#) that you will find useful.
5. To better understand the MPLAB environment, there's great documentation at [MPLAB IDE](#). (But it's lots of pages.)
6. Bates, Chapters 6 – 8 provides excellent information. That write up is for the PIC 16F84A, but it's very applicable here.

I recommend you look at these materials in approximately the order given above.

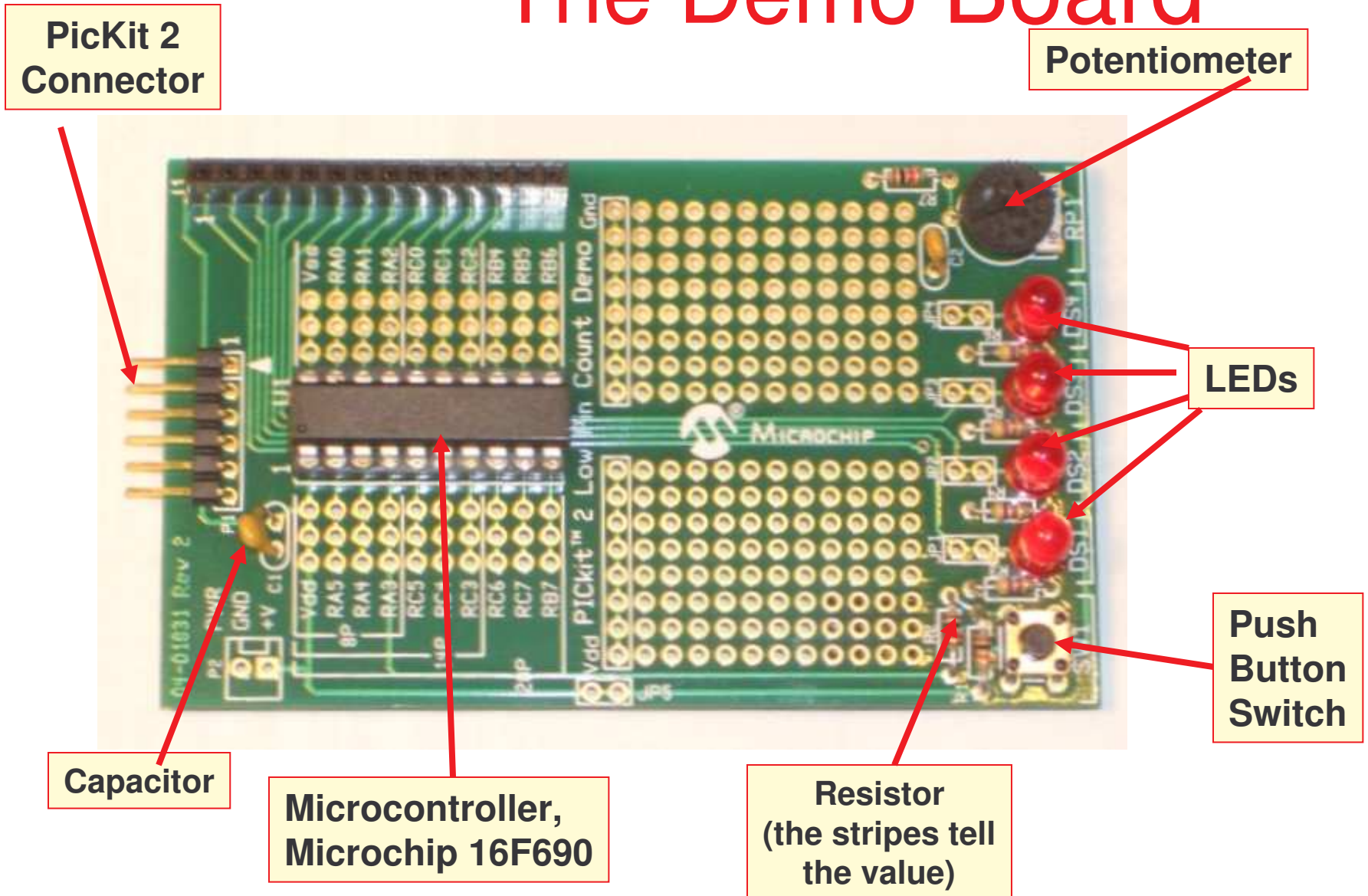
You might want to try this – a video of how to use MPLAB.

http://techtrain.microchip.com/media/websem/IntroToMplab_033004.wmv

The Big Picture

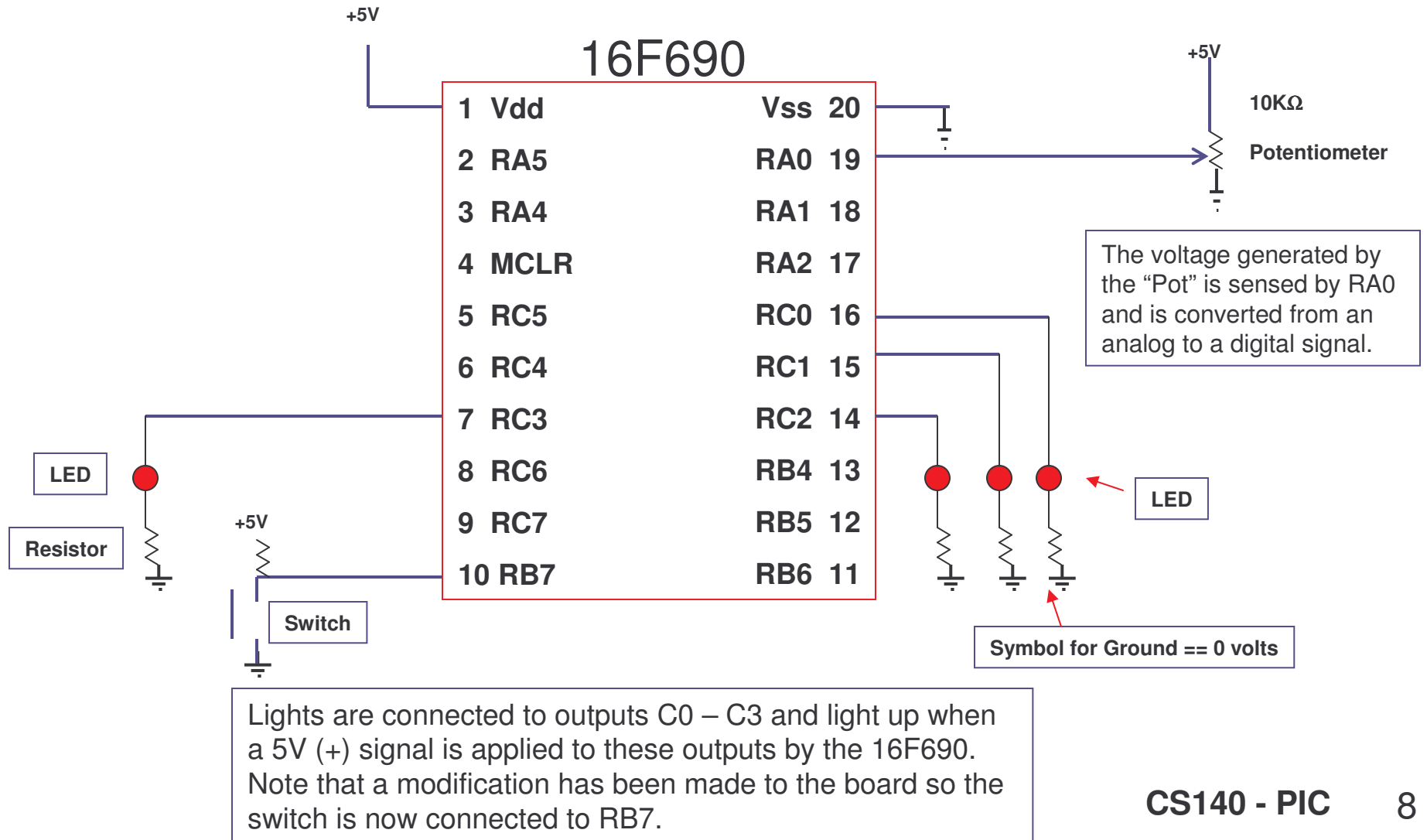


The Demo Board



The Demo Board

This is another way of drawing the **schematic** on Page 37 of the Demo Board Spec.



The Development Process

1. Write assembly code – at home or in the lab. Remember, you can download the MPLAB on your home Windows machine.
2. Transfer code to the development PC – via email or ssh.
3. Load your code into the MPLAB.
4. Configure MPLAB.
5. Assemble the code – repeat until free of assembly errors.
6. Load code into the MP Simulator (MPSIM)
7. Debug your code using MPSIM – go back to 5 as necessary.
8. Load hex code (produced by the assembler) into the DemoBoard.
9. Run the code.
10. Smile.

A great way to see how this is all put together is to run the tutorial in [MPLAB Getting Started.pdf](#)

Writing Assembly Code

1. We will have talked about this in class and lab. How to write code, what the instruction set looks like, what the code means in terms of the hardware, etc.
2. Your first assignment starts by simply taking code that's already written and getting it going on the DemoBoard. Doing all the development steps is complicated enough without worrying about coding intricacies.
3. By the time you complete this, you'll be a pro at using MPLAB and running code.

All the code for these exercises can be found [HERE](#).

Writing Assembly Code

```
; *****  
; PICkit 2 Lesson 3 - "Rotate"  
;  
; *****  
; * See Low Pin Count Demo Board User's Guide for Lesson Information*  
; *****  
; * You will be able to debug using the MPLAB simulator, but you *  
; * will not be able to debug on the Microcontroller itself.  
; *****  
  
#include <p16F690.inc>  
; The __config sets the bits described on pg 195 of the 16F690 manual.  
    __config (_INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF & _MCLRE_OFF &  
_CP_OFF & _BOR_OFF & _IESO_OFF & _FCMEN_OFF)  
  
    cblock 0x20  
Delay1          ; Assign an address to label Delay1  
Delay2  
Display        ; define a variable to hold the diplay  
    endc
```

This is the example code for Program 3. Let's talk about it. Note that the text in the Demo Board writeup explains all.

Writing Assembly Code

```
    org 0
Start:
    bsf     STATUS,RP0        ; select Register Page 1
    clrf   TRISC              ; make IO PortC all output
    bcf     STATUS,RP0        ; back to Register Page 0
    movlw  0x08
    movwf  Display

MainLoop:
    movf   Display,w          ; Copy the display to the LEDs
    movwf  PORTC

OndelayLoop:
    decfsz Delay1,f           ; Waste time.
    goto   OndelayLoop        ; The Inner loop takes 3 instructions
                                        ; per loop * 256 loopss = 768 instructions
    decfsz Delay2,f           ; The outer loop takes an additional
                                        ; 3 instructions per lap * 256 loops
    goto   OndelayLoop        ; (768+3) * 256 = 197376 instructions
                                        ; divided by 1M instr./second = 0.197 sec.
                                        ; call it a two-tenths of a second.
    bcf     STATUS,C          ; ensure the carry bit is clear
    rrf    Display,f
    btfsc  STATUS,C           ; Did the bit rotate into the carry?
    bsf    Display,3          ; yes, put it into bit 3.
    goto   MainLoop
end
```

Getting To Perfect Code

Wow – perfect code is quite a challenge. It requires, typing perfectly, debugging and inspection of results to make sure you have what you want. Here's the series of steps I followed once I had written the code in Rotate.asm.

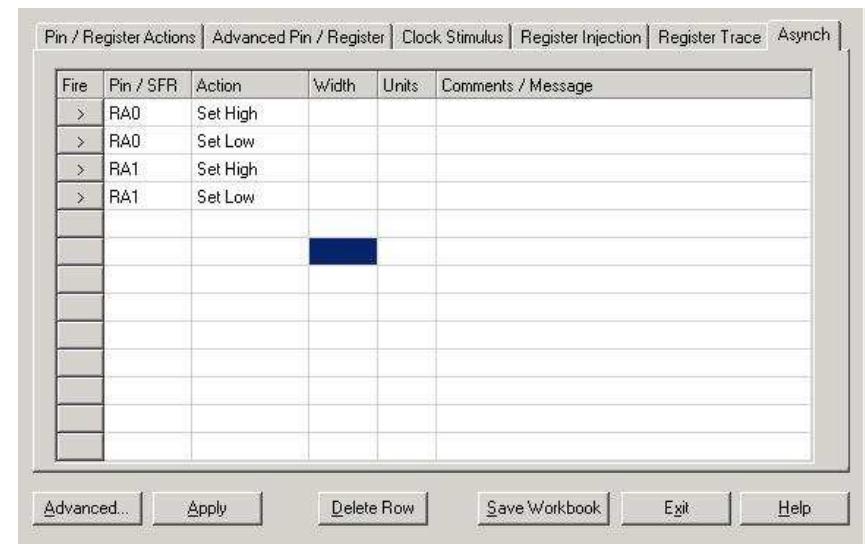
Once you've started up MPLAB, here are the steps to follow:

1. Configure → Select Device → Device = PIC16F690 // This tells MPLAB the name of the chip being used
2. File → Open → "Rotate.asm" // Bring into MPLAB the file from wherever you've put it.
3. Debugger → Select Tool → MPLAB SIM // You're going to be using the Simulator to check out the code. This is because the 16F690 doesn't have debugging facilities on the chip. So you have to get it right before you download it onto the chip – there's no way of knowing on the chip what's wrong.
4. Project → Quickbuild // This invokes the assembler to process your code
5. If Assembly errors occur, fix the problems and return to step 4.
 - a) A common message is the one shown here. Ignore it.

Message[302] X:\LABS\Prog01\Rotate.ASM 43 : Register in operand not in bank 0. Ensure that bank bits are correct.
 - b) Like any compiler/assembler, MPLAB does its best to interpret the error it found and give you hints on how to fix it. But, it's only human and of course is not perfect. Work with it the best you can.

The MPLAB Getting Started document, Section 2.12, does a great job of explaining the debugger MPSIM.

6. Help --> Topic --> MPLAB SIM --> Using Stimulus // Amazing concept – do a bit of reading to better understand what you're doing!
7. Debugger --> Stimulus --> New Workbook --> Asynch // This sets up the mechanism that allows you to simulate the pushing of the buttons on the Board. Thus the simulator behaves like it's the real processor.



Getting To Perfect Code

8. View --> File Registers // This lets you observe what all the variables are doing at any time.

Delay1 is at location 0x020

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00	--	00	0E	1B	00	03	01	--	00	00	00	00	00	00	00	00	-.....-
10	44	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	D.....
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	-----
60	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	-----
70	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	-----

Hex Symbolic

PORTA is at location 0x005

PORTB is at location 0x006

9. Debugger --> Animate // Here the magic happens! You can see the code proceeding in a single step fashion. You can push the buttons in the stimulus window and watch how your code behaves.
10. Try other features. // What happens with other windows that you can control from the view button.

Getting The Board Working

11. Programmer --> Select Device --> PicKit 2 // OK – now you’re done debugging. You think that the code is correct. This starts the process of putting the code onto the 16F690. MPLAB will try to download Code into the PicKit2 which can take a minute. Errors typically occur because something isn’t plugged in correctly. If all is OK, and you are patient, you will eventually get a message saying:

```
Initializing PICkit 2 version 0.0.3.12
Found PICkit 2 - Operating System Version 2.20.0
Target power not detected - Powering from PICkit 2
PIC16F690 found (Rev 0x5)
PICkit 2 Ready
```

12. Programmer --> Program // This causes the MPLAB to download the hex code produced by the Assembler down to the 16F690. It should eventually print out lots of stuff including “Programming Succeeded”.

```
Programming Target (1/4/2008 3:45:26 PM)
Erasing Target
Programming Program Memory (0x0 - 0xF)
Verifying Program Memory (0x0 - 0xF)
Programming Configuration Memory
Verifying Configuration Memory
```

13. Programmer → Release from reset // And it should work! Lights blink, cheers from the crowd.

Your Deliverables

1. There is nothing you have to hand in at the end of this lab, but you should be familiar with the programs that are provided by Microchip as exercises.
2. You could download and run ALL 12 – but that's probably overkill. I would recommend that you execute:

- 01_Hello_World
- 02_Blink
- 03_Rotate
- 04_A2D
- 05_Variable_Rotate
- 06_Debounce
- 07_Reversible
- 09_Timer0
- 10_Interrupt

3. Know how to debug the code using MPLAB. The debugger is a wonder to behold – it allows you to do things like set breakpoints, single step, look at memory locations, etc. **YOU SHOULD KNOW HOW TO DO THIS!**
4. Take one of these programs and change something. It could be a delay time, it could be something bigger – your choice. It would be nice to read the 16F690 spec and teach the chip to do some new function – can you change the system clock for instance?
5. Smile.