Senior Research Project
PIC Chip Lab
Developing a PIC Based Application

John Cleveland
PHYS 451-452

Advisor: Dr. Kossler

Abstract

A lab for the Electronics Course is developed. This lab uses a PIC chip and demonstrates the use of microcontrollers. A sample program is included and the means to program and use the chip are described.

Introduction

There is a need to modernize the Physics Department's Electronics Lab (PHYS 252-351). As an attempt to do this, the study of microcontrollers was considered. The present course uses a Motorola 68008 cpu and peripheral memory, address registers, and I / O controller. Modern microcontrollers have all of this built in. While the separate component route allows the details of a computer to be understood., such a system is complex and difficult to program and interface in comparison to microcontrollers. Further, microcontrollers have become ubiquitous and need to be understood by electronics students.

Microchip Corporation manufactures a microcontroller called a PIC chip. PIC chips are characteristic of typical microcontrollers in terms of price and ease of use. The Basic Stamp is a microcontroller manufactured by Parallax which is easier to program but is more expensive than the PIC chip. However PIC chips have several advantages over other microcontroller for several reasons. First, Microchip Corporation offers extensive help and support for their PIC chips through their internet site. The Microchip Corporation website is a valuable resource for working through problems should they arise. Also, there is extensive information on the internet devoted to PIC chips so it is always easy to find help, solutions to problems, and PIC related products.

The following paper presents a lab write up to be used in the Electronics course of the physics curriculum. This lab introduces a programmable microcontroller called a PIC chip to the physics student. The projects in the lab were adapted from two texts on PIC chips: Easy PIC'n, by David Benson and PIC: Your Personal Introductory Course, by John Morton. Both of these texts offer excellent introductory material on the PIC chip and many simple projects. The lab takes the student step by step through the process of developing a PIC based application. The purpose of this lab is for the student to understand the process behind developing a real-world application using programmable microcontrollers.

The PIC chip
An Instructive Lab

Introduction

Welcome to the world of PIC chips. PIC stands for **P**eripheral **I**nterface **C**ontroller. It is a microcontroller manufactured by Microchip Corporation. It's not a microcontroller like the Pentium processor; a PIC chip doesn't handle large amounts of data. This chip was originally designed as a means of interacting with the outside world, hence the term "Peripheral Interface" which is just a concise way of saying "interaction with the outside world."

Microchip Corporation makes dozens of PIC chips. While they all have slightly different features, capabilities, and characteristics, their basic structure is nevertheless fairly similar. Once you learn how to work with one PIC chip, working with another will be a natural extension.

So why study PIC chips? First, PIC chips are on the cutting edge of technology in the electronics industry. Microchip Corporation is coming out with new PIC chips all the time to extend the capabilities and ease of use of these devices. Second, the wave of the future in electronics is circuits built around programmable microcontrollers, such as a PIC chip. Just ten years ago, it would have taken hundreds of wires, resistors, capacitors, and transistors, all put together in a confusing array on a breadboard to accomplish the same thing that a little PIC chip can do today. With the proper know-how a PIC can be used in thousands of practical applications, from TV remote controls to digital clocks to credit card swipers. In virtually every electronic device today you can bet that a microcontroller is somewhere in the circuitry, perhaps not a PIC chip, but something similar.

After this lab you will be able to see microcontroller solutions in a variety of applications at school, in the everyday world around you, and possibly on the job.

Overview of the Project

Here are the basic steps one uses to construct a PIC based application:

Step 1: One needs both a PC running Windows and a PIC programmer, which is a circuit that connects to the computer via a COM port and is used to transfer the PIC code from the computer into the program memory of the chip. You have been supplied with a constructed programmer for today's lab.

Step 2: Choose the particular PIC chip you want to use for your project. This has already been decided for you; you will be using the PIC16F84. This chip can be electronically programmed and reprogrammed as many times as you want. Other microcontrollers must be exposed to UV light to erase the old program, or they may only be programmed once. Reprogramming the PIC16F84 is simply a matter of placing the chip into the programmer and hitting 'Program' in MPLAB (more on this shortly). Thus, this particular chip is extremely useful for educational purposes.

Step 3: Write the program. This depends on what you want your PIC chip to do. The PIC programming language consists of 35 instructions so it's not too complicated. You'll be using MPLAB software to write the program. MPLAB is provided free by Microchip Corp. (they figure if they give you the software for free you'll get interesting in this stuff and start to buy the PIC chips themselves as well as other PIC related products they sell). MPLAB contains a text editor, with which you'll write the program in PIC assembly language, an assembler, which translates your program into binary code (which is what the PIC chip actually understands), and a simulator, which helps you debug your code.

Step 4: Assemble the program. This is the easiest step. You just use the MPASM (**M**icrochi**p** **As**se**m**bler) to assemble the program. This is simply a matter of clicking on a few buttons in the software. As explained above, MPASM is part of the MPLAB package.

Step 5: 'Fuse' the PIC. This is just a matter of plugging your programmer into the computer and hitting 'Program' in MPLAB.

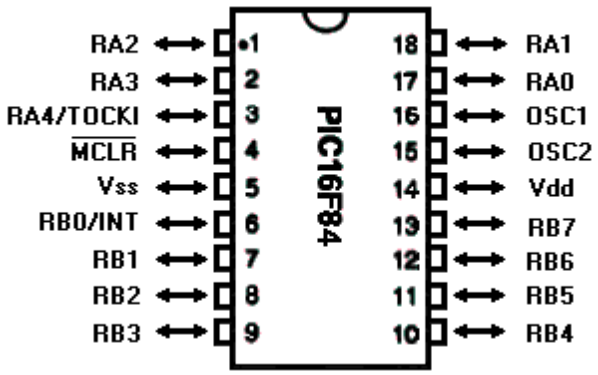
Step 6: Build your circuit with which the PIC chip operates. This circuit is built to interact with the program that you placed in the PIC's memory. We'll be building a few simple PIC circuits in this lab to turn on several lights.

Step 7: Put the programmed chip into the circuit and watch it work.

Architecture of a PIC chip

Pin Layout

Shown below is the pin layout of the chip you'll be using for today's lab:

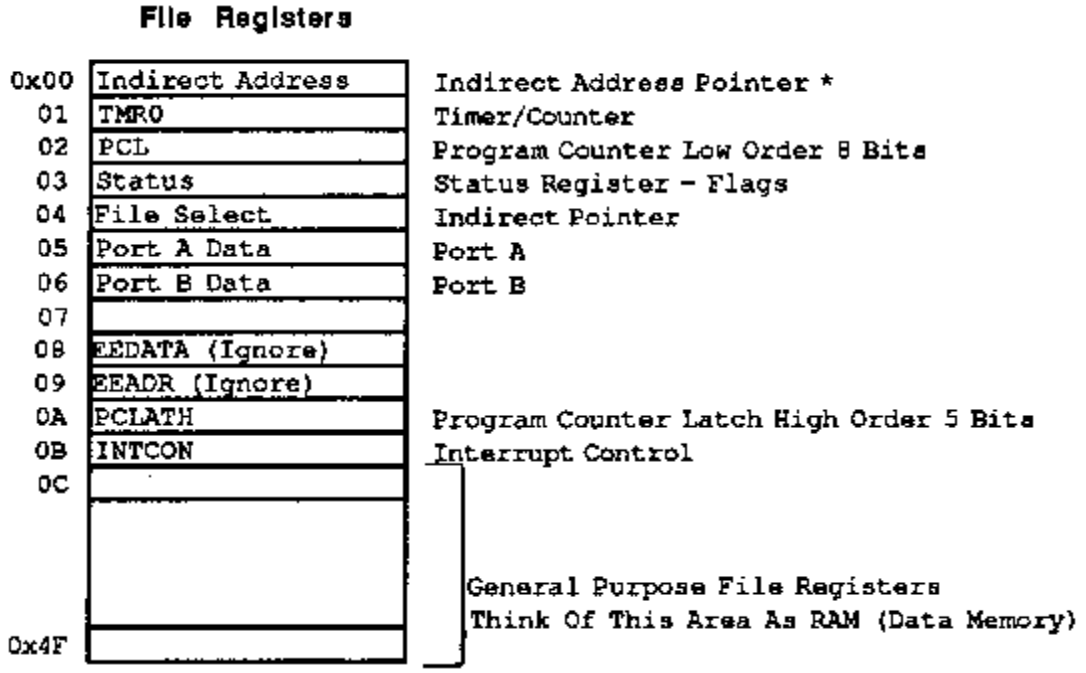


Note that there are 18 pins. The pins labeled RA0 (pin 13), RA1 (pin 18), RA2 (pin 1), RA3 (pin 2), and RB0 – RB7 (pins 6 – 13) are input/output pins. This means that these pins can be used to either read in a signal from the outside world or send out a signal to the outside world. The PIC project builder (you) determines which pins will be used as inputs and which pins will be used as outputs based on the particular application (you will learn how to do this shortly). The MCLR pin (which stands for Master Clear) is used to bring the chip into a known initial state. The pin Vdd is the pin used to power the chip so it is connected to +5 VDC and Vcc is the ground pin.

Internal Structure
File Registers

The next thing to look at is the internal architecture. The internal structure of the PIC chip is composed of two buses: the **file registers** and the **program memory**.

Here's a diagram of the file registers:



Think of the file register system as a large filing cabinet. Each drawer in the filing cabinet represents a particular file register. Each file register in the filing cabinet holds a number that is eight bits long, or 8-bit wide "word." For example, let's say the number 00000111 is in register 06, Port B. The significance of this number is **that each particular bit of this 8-bit number has a certain function**. Each bit in Port B corresponds to one of the pins RB0 – RB7 on the chip. A binary "word" is read from right to left, so in out example binary word, 00000111, the three ones (the first three bits of the 8-bit binary number) correspond to RB0, RB1 and RB2 and the 0's correspond to RB3 – RB7. If a bit is 1, then the corresponding pin is high, or +5VDC. If a bit is 0, then the corresponding pin is low 0 V. Thus, if the number 00000111 is in Port B, then pins 6, 7, and 8 would be high and pins 9 – 13 would all be low.

In addition to the file registers listed in the diagram there is the working register which is used to transfer data from one register to another. Think of the working register as a pair of hands that takes a number out of one file register and puts it into another file register. You will use the working register later in the lab. The two file registers with which we will be working in today's lab are Port A (the first four of its 8 bits correspond to the input/output pins RA0 – RA3), and Port B, discussed above. Don't worry about all the other file registers, they are beyond the scope of a three-hour lab. Just be aware that a PIC chip has many file registers and understand the concept of each bit in a file register having a particular function.

Program Memory

In addition, there is the program memory. Each memory block is 14 bits wide and there are 1 k of them (in cpu terms 1k is 1024, not 1000) in this particular chip. Other PIC chips have different amounts of program memory. 1k memory is enough memory to write fairly lengthy and sophisticated programs. Here is a diagram of the structure of the memory:

The notation 0x signifies hexadecimal, or just "hex" if you want to sound like you know what you're talking about. Hexadecimal is a convenient numbering system to use with computers, microcontrollers, or any sort of programming memory. Thus, it is extremely helpful to understand hex notation.

Question 1:

Write the decimal number 4238 in hex. What is 3FF in decimal notation? In addition, what is 3FF in binary representation.

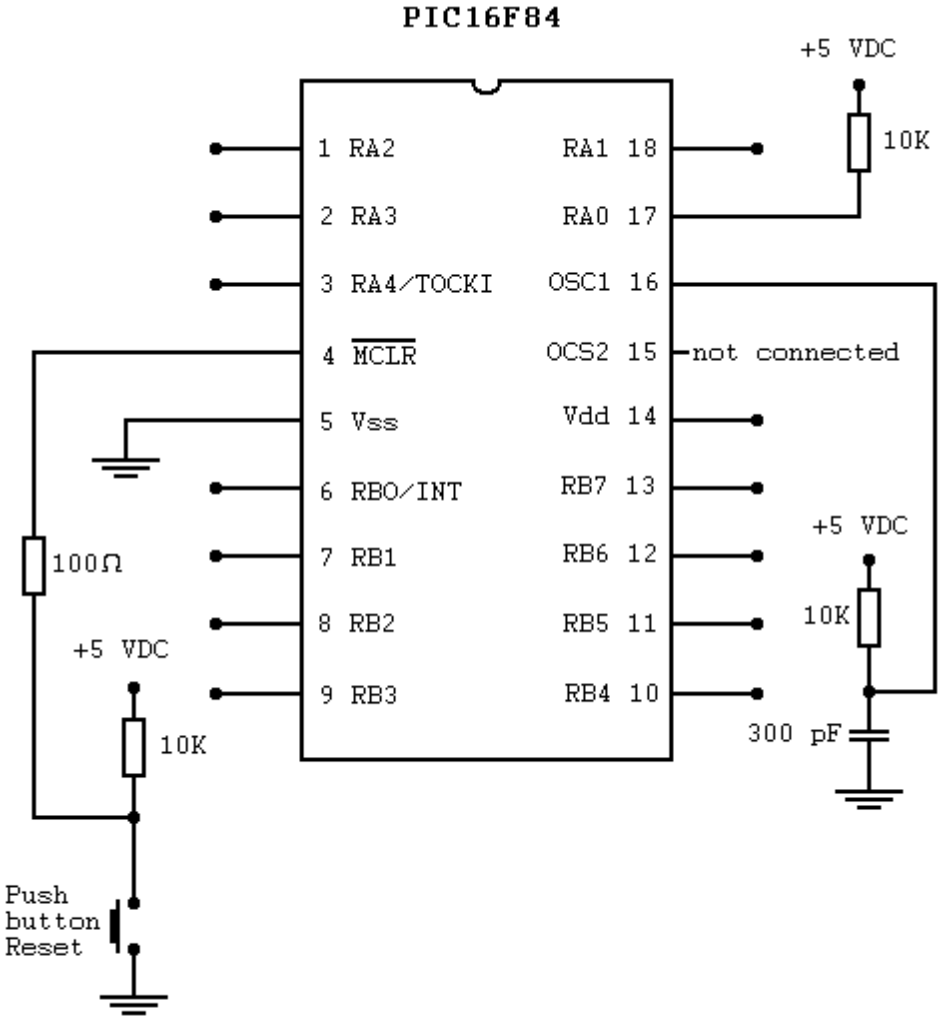
Question 2:

How many bits are in a block of program memory in a regular desktop computer?

Let's get started.

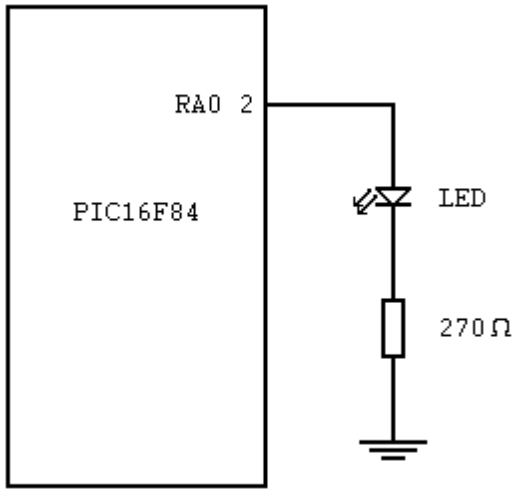
Initial Circuit

Build the circuit below on the breadboard. This is the basic circuit you will use for the three projects that we will build today. Note the way RA0 is wired (pin 17). All port lines (RA0 – RA3 and RB0 – RB7) should be wired in the same manner as RA0. Once this circuit is wired it will only be a matter of changing several wires for the projects in today's lab.



Project #1: Turn on an LED

We will start with a simple project in order to understand the fundamentals of PIC chips. This project uses the PIC chip to turn on an LED and leave it on. Now that you have built the basic circuit, the circuit for this project only requires you to change one wire. Attach an LED to RA0; see the schematic below:



The Program

Now that the circuit is built we can write the program. We'll use the MPLAB software provided by Microchip Corp. to write the program. Start by opening MPLAB. Go to My Computer > C: drive > PIC Lab > MPLAB. Once inside the MPLAB folder double click on MPLAB.exe. Once MPLAB opens go to Project > Open Project. When the dialog box comes up, open the project **led_on.pjt**. If no next file comes up after you open **led_on.pjt** then go to File > Open and click on **led_on.asm**. A text editor screen should now be open with c:\PIC Lab\MPLAB\led_on.asm in the title bar. Some information has already been written in this file. At the top where it says 'written by: ' and 'date: ' type in your name and the day's date.

This is where you will type your PIC source code. MPLAB assembler will then take this source code and translate it into binary code that will be fused into the PIC program memory using the PIC programmer.

Notice the semicolons (;). The assembler ignores everything after a semicolon. It is the same thing as // in C++. This enable the person writing the program (you) to add comments to a program.

Notice the lines that say

```
list        p=16f84
radix       hex
```

These two lines tell the assembler which type of PIC is being used, in this case the PIC16F84, and that you will be using hex notation in your program.

The next section is labeled **;Declarations**. This is where you will begin writing code. The assembler understands the file register Port A as simply 05 because this register has hexadecimal address 05 (refer back to the file register diagram if you don't' remember). Instead of writing 05 every time we want to refer to Port A, we would rather refer to this register with a label which we have chose, for example, 'porta' (we could just have easily chose 'pa' or 'portA'; this is simply a mater of preference). Referring to register Port A with 'porta' is much easier that remembering Port A is register number 05. We can attach a label to any file register with the **equ** instruction. On the line under **;Declarations** type

porta equ 05

(porta, tab, equ, tab, 05) We have just labeled file register 05 'porta'. Now every time we refer to this file register in the program we can just say 'porta'. Next we have the lines

org 0x000

This line tells the PIC to start executing instructions from the first address in program memory (of course!). Now let's go to the section labeled **;Program Start**. Place the cursor on the next line. Here we will write a function that initializes the PIC. Type the word **Init** and hit the tab key. Now we are want to tell the PIC chip which pins are input pins and which pins are output pins. We do that by loading a particular binary number into **porta**. A 1 designates an input, and a 0 designates an output

1 = 1nput
0 = 0utput

Remember that the first bit of the number in **porta** corresponds to RA0, the second bit of the number in porta corresponds to RA1, etc. We need an eight-bit number and since there are no RA5 – RA7 pins we make the last three bits 0 by default.

If an input/output pin is not being used we make it an output by default. Thus, since the only pin we are using for this project is RA0, and this is an output pin, we'll want the number 00000000 to be stored in porta (the first 0 for the output pin RA0, the next four 0's for pins RA1 – RA4 that are not being used, and 0's for the last three bits that never get used). We tell the PIC chip this information by loading 00000000 into a file register called the working register (the working register is used to load a binary number into a file register). We do this with the instruction **movlw**, which means move the literal (the number) into the working register. Thus you should have

Init movlw 00000000

Next we call upon the instruction **tris** which is the instruction tells the PIC chip to access the input / output pins. We take the number in the working register, **00000000**, and tell the PIC that we want all outputs. Thus type

tris porta

Initialization is complete.

Now place the cursor on the line that says **Main.** Look at your circuit and notice that the LED is connected to pin 17 on the PIC chip, which corresponds to RA0. Also, remember that RA0 corresponds to the first bit of file register **porta.** In order to turn on the LED we have to have some current going through it, which means making pin 17 high (+5VDC). Thus we need some way to make bit 0 or porta high. When a number in a register is 1, then the corresponding pin is high. When a number in a file register is 0, then the corresponding pin is low. Thus, in order to make RA0 high we need to load the number 00000001 into porta. Again we do this with the working register. Thus type

```
Main   movlw        0x01          ;move 00000001 into working register
```

Where 0x01 is hex notation (it is the same thing as 00000001 in binary notation). Now we take the number in the working register and place it into porta. Thus on the next line type

```
       movwf        porta         ;turn on LED
```

Now we want to keep the LED on. So the next instruction should make the program go back to the beginning of Main. You can do this with the **goto** instruction, which makes the program go to some particular place in the program. Thus, on the next line, type

```
       goto    Main
```

The program will just keep looping around and around, keeping the LED on. On the next line type

```
       end
```

This will tell the assembler when to stop assembling your code. If all is well you should be able to assemble your code.

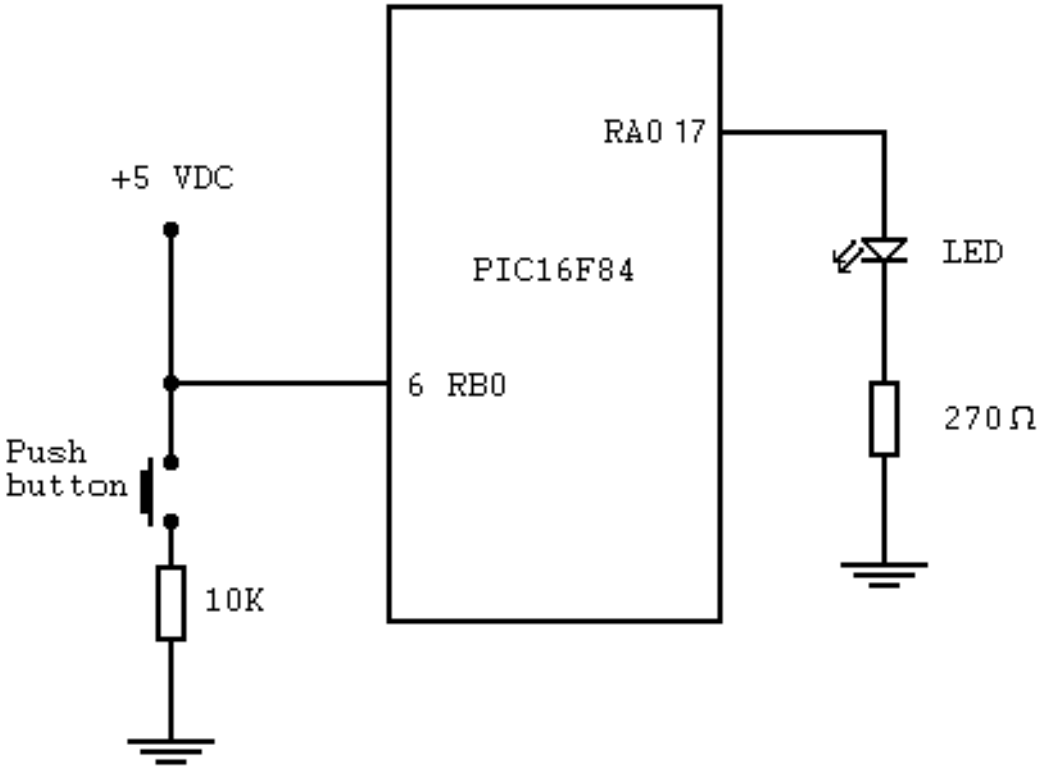
Assembling

In order to assemble your code go to Project > Build Project. This will assemble your code and tell you of any errors if you have any. If you do get an error message, simply double-click on the line where the error message is written and MPLAB will take you to the line in your program that has the error. Ignore the warning message due to the **tris** instruction. If you can't figure out how to fix your errors, take a look at Appendix A.

To fuse the PIC with the program go to Picstart Plus > Enable Programmer. When the dialog box pops up click on the button at the bottom of the screen which says Program. It will take a few seconds. Once the dialog box says Success, you may remove the chip from the programmer and place it into the circuit.

Project #2: Push Button to turn on LED

This project will incorporate a PIC chip to use a push button to turn on an LED. Here is the schematic:



The Program

We will use the last project as a template for this one. Close the Picstart Plus dialog box. Go to Project > Save As and type in button.pjt. Now go to File > Save As and type in button.asm. Now we are ready to start writing code.

We will now be using another file register in this project: portb. RB0, pin 6 on the chip, is connected to the button. Thus it should be designated as an input pin. You need to set up portb input / output pins. On the two lines after **tris porta**, type the appropriate set up for portb.

When the push button is not pressed, what number is in portb? Remember RB0 corresponds to the first bit of portb. When the button is not pressed, then pin 6 is high. When the button is pressed, then pin 6 goes low.

Thus, if the button is not pressed, then the number 0000001 is in portb. When the button is not pressed we want the LED to be on, thus RA0 should be high which means the number 00000001 should be in porta. When the button gets pressed then the number 00000000 is now in portb (because RB0 is now low) and we want to turn off the LED which means putting 00000000 into porta. Therefore we need to move whatever number is in portb into porta. We have already learned how to move numbers into and out of file registers. First we move the number that is in portb into the working register. Go to the line that says Main and delete the instruction. Replace it with the following

```
Main   movfw        portb
```

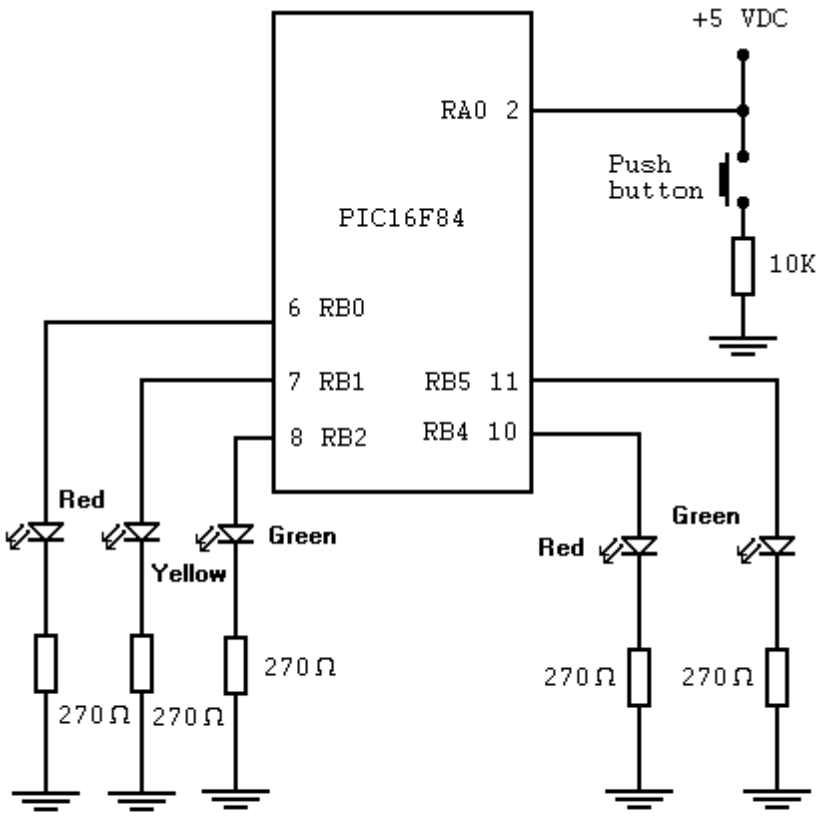
This moves the number in the file register portb into the working register. Now we want to move the number in the working register into porta. On the next line erase the instruction there and replace it with the following

```
       movwf        porta
```

We want to continuously test to see if the button is pressed, so again make sure the instruction **goto Main** remains in the program. Also make sure the instruction **end** is still in your program. The complete program is in Appendix B. Verify that your program matches the one in Appendix B. Assemble your program and fuse the program into the PIC. Place the PIC in the circuit and see if the LED tums on and off as you press the button.

Project #3: Pedestrian Crosswalk

This project uses a PIC chip to simulate a pedestrian crosswalk switch. The code for this project is beyond the scope of this lab, so it has already been written for you. Go to Project > Open Project in MPLAB and open traffic.pjt. Once opened to Project > Build All to assemble the program. Then program the chip using the programmer as before. Next build the circuit below and place the programmed chip into the circuit. The complete code for this project is in Appendix C.



Try to figure out what this circuit does by pressing the button.

While we have only scratched the surface of PIC chips, you should now have a feel of how to work with programmable microcontrollers. If you are interested in learning more there are numerous resources on the web to help the beginner get involved with PIC chips.

Appendix A

```
;*****************************************************************
;  Author:      John Cleveland
;  Date  :      April 20, 2000
;  File Name:   mod_led.asm
;  PIC type:    PIC16F84
;  Description: This is the model program for the first project in the
;               PIC Lab.  This program uses the PIC16F84 to turn on an
;               LED at port RA0.
;*****************************************************************
;
      list  p=16f84             ;tells which PIC to be used: PIC16F84
      radix hex                 ;to use hex notation
;
;*****************************************************************
;Declarations
porta equ   0x05                ;now file register 05 labeled porta
;
      org   0x000               ;start executing instructions from
;                                first address
;
Init  movlw 0x00                ;set up I/O for porta: RA0 as output,
      tris  porta               ;RA1 - RA3 not connected
;
Main  movlw 0x01
      movwf porta               ;turn on LED
      goto  Main                ;keep LED on
      end
```

Appendix B

```
;*****************************************************************
;  Author:      John Cleveland
;  Date  :      April 20, 2000
;  File Name:   mod_but.asm
;  PIC type:    PIC16F84
;  Description: This is the model program for the second project in the
;               PIC Lab.  This program uses the PIC16F84 to turn on an
;               LED at port pin RA0 when a push button is pressed.
;*****************************************************************
;
      list  p=16f84             ;tells which PIC to be used: PIC16F84
      radix hex                 ;to use hex notation
;
;*****************************************************************
;Declarations
porta equ   0x05                ;now file register 05 labeled porta
portb equ   0x06                ;not file register 06 labeled portb
;
      org   0x000               ;start executing instructions from
;                                first address
;
Init  movlw 0x00                ;set up I/O for porta: RA0 as output,
      tris  porta               ;RA1 - RA3 not connected
      movlw 0xff                ;set up I/O for portb: RB0 as input,
      tris  portb               ;RB1 - RB7 not connected
;
Main  movfw portb               ;read portb, put result in working
;                                register
      movwf porta               ;turn on LED
      goto  Main                ;keep sensing when button is pressed
      end
```

Appendix C

```
;*****************************************************************
;  Author:      John Cleveland
;  Date  :      April 20, 2000
;  File Name:   traffic.asm
;  PIC type:    PIC16F84
;  Description: This is the model program for the third project in the
;               PIC Lab.  This program uses the PIC16F84 to simulate a
;               pedestrian crosswalk by turning on and off a motorist
;               traffic light when the button is pushed.
;*****************************************************************
;
      list  p=16f84             ;tells which PIC to be used: PIC16F84
      radix hex                 ;to use hex notation
;
;*****************************************************************
;Declarations
porta equ   0x05                ;now file register 05 labeled porta
portb equ   0x06                ;not file register 06 labeled portb
ncount      equ   0x0c          ;holds number for delay routine
mcount      equ   0x0d          ;holds number for delay routine
count5      equ   0x0e          ;for flashing routine
;
      org   0x000               ;start executing instructions from
;                                first address
;
Init  movlw 0x01                ;set up I/O for porta: RA0 as button,
      tris  porta               ;RA1 - RA3 not connected
      movlw 0x00                ;set up I/O for portb: RB0-RB4
      tris  portb               ;as inputs, RB3, RB6-7 not connected
      goto  Main
;
;*****************************************************************
;Subroutines
delay movlw 0xff                ;creates a 1/3 second delay
      movwf mcount              ;load 256 into mcount
loadn movlw 0xff                ;load 256 into ncount
      movwf ncount
decn  decfsz      ncount, f     ;decrement number in ncount
      goto decn
      decfsz      mcount, f     ;decrement number in mcount
      goto  loadn
      return
;
;
;
Main  movlw 00100100            ;motorist green LED on, pedestrian red
;                                LED on, all others off
      movwf portb
;
ButtonTest  btfss porta, 0      ;is pedestrian button pressed?
      goto  ButtonTest          ;if no, go back and test again
;
      movlw 00000010
      movwf porta               ;turn on motorist yellow light
      movlw 00000000
      movwf portb               ;turn off motorist green light
      call delay
      call delay
```

```
      call delay
      call delay
      call delay
      call delay                ;waits for two seconds
      movlw 00010001
      movwf portb               ;motorist red on, ped green on, red off
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay
      call delay                ;wait for five seconds
      movlw 00010010
      movwf portb               ;turn on motorist yellow, red off
      movlw 0x05
      movwf count5              ;load count5 with the number 5
Flash movlw 00010010            ;this routine flashes the ped green LED
      movwf portb
      call delay
      call delay                ;keep ped green on for half second
      movlw 00000010
      movwf portb
      call delay
      call delay                ;turn green off for half second
      decfszcount5              ;makes ped green LED blink on and off
;                                five times
      goto  Flash
      goto  Main                ;loop back to main to sense when button
;                                is pressed
      end
```

Appendix D

The following is an explanation of the instructions used in this lab. This is not the complete instruction set for the PIC chip. There are 35 instructions in the complete set.

call subroutine calls a subroutine

decfsz fileRegister decrements the number in the file register and puts the result back into the same file register

goto k go to a specified address or label

movlw k loads the working register with a number

movfw fileRegister copies contents of selected register into working register

movwf fileRegister copies contents of working register into selected file register

return returns from a subroutine

tris fileRegister working register bit pattern determines port line input vs. output

Discussion and Conclusions

The preceding lab would fit naturally at the end of the current course. It would take the place of the applications lab sessions (Lab 23 of Horowitz and Hill). It is likely that after a first trial, modification in the proposed lab would develop. It is hoped that through this lab the student will gain a sufficient understanding of microcontrollers to be able to directly use them in their future research or career paths.

Works Cited

Benson, David. Easy PIC'n A Beginner's Guide to Using PIC 16/17 Microcontrollers. Square 1 Electronics, Kelseyville, CA, 1997.

Horowitz, Paul and Hill, Winfield. The Art of Electronics. Cambridge University Press, 2nd Edition, 1989.

Morton, John. PIC: Your Personal Introductory Source. Reed Educational and Professional Publishing Ltd, Woburn, MA, 1998.