Programming PIC Microcontrollers in PicBasic Pro – Lesson 1
Cornerstone Electronics Technology and Robotics II

- **Administration:**
  - Prayer
- **PicBasic Pro Programs Used in This Lesson:**
  - General PicBasic Pro Program Listing: http://www.cornerstonerobotics.org/picbasic.php
  - Lab 1 flicker1 as .pdf: http://www.cornerstonerobotics.org/code/blink1.pdf
  - Lab 1 flicker1 as .pbp: http://www.cornerstonerobotics.org/code/blink1.pbp
  - Lab 1 railroad1 as .pdf: http://www.cornerstonerobotics.org/code/railroad.pdf
  - Lab 1 railroad1 as .pbp: http://www.cornerstonerobotics.org/code/railroad.pbp
- **Computer Programming:**
  - In order to achieve the task at hand, a programmer must write a sequence of instructions and create data structures for the computer to execute.
  - Write a list of detailed instructions for the instructor to make a peanut butter and jelly sandwich, given the starting conditions before you.
- **PIC Microcontrollers Overview:**
  - Using microcontrollers (MCUs):
    - You will need to know how to connect the microcontroller to the hardware.
    - You will need to know how to write and program code into the microcontroller.
  - Levels of Programming Languages:
    - MCUs are programmed in machine language code (binary code) which looks like:

      0000100001001001
      0001100000000011
      0111100000000111

      - Machine language code is the native language for PIC MCUs.
      - PIC machine language code ends with .hex
    - Assembly level code makes programming commands more recognizable; however, it forces the programmer to deal with the MCUs internal structure. Assembly code looks like:

      ```
      movlw   h'07'
      addwf   INDF, w
      btfsc   STATUS,C
      ```

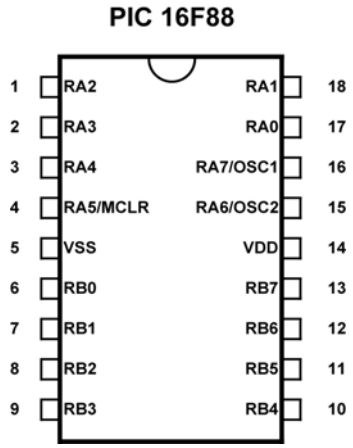
      - Another difficulty with assembly-level code is that each line of machine code must have a line of assembly code written.

        - Assembly code commands are executed at the crystal frequency/4.
        - PIC assembly code ends with .asm
    - High-level language: A programmer needs a programming language that relates to problem solving more than the internal structure of a microcontroller.
        - The most common high-level language for programming MCUs is C.
        - We will start with the language PicBasic Pro, then in time, move on to C. PicBasic Pro code appears like:

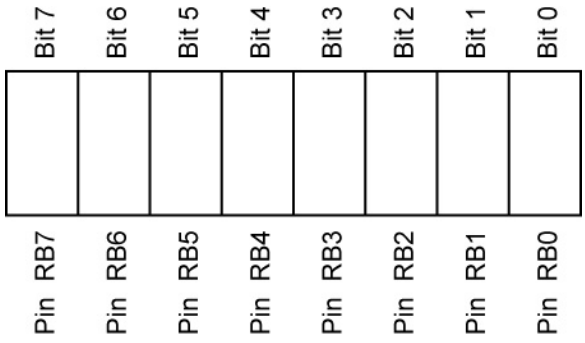
```
For c0 = 1 TO 100      ' Count from 1 to 100
SOUND 1, [75,100]      ' Generate tone on pin 1
Pause 20               ' Delay 20 milliseconds
Next                   ' Return to FOR and add 1 to c0
```

        - PicBasic Pro code ends with the extension .pbp.
- Programming the microcontroller with PicBasic Pro:
    - You will need to communicate with the microcontroller and tell it what instructions you want it to perform. The program language for the PIC microcontrollers uses about 75 words, or instructions, called PicBasic Pro language. Make sure you program using commands that are handled by the PicBasic Pro compiler (PBP), not the PicBasic Compiler. See: http://store.melabs.com/cat/PBP.html
    - You will program on the computer in PicBasic Pro language. You will then compile your program using the PicBasic Pro compiler. The compiler first compiles the program into assembly code (.asm) and then automatically converts the assembly code into the final machine code (.hex) which is then programmed into a microcontroller using the melabs U2 programmer. See: http://melabs.com/products/usbprog.htm
    - Steps in programming a microcontroller will be discussed in more detail later in the class.
- PIC16F88:
    - Cost is about $2.60 each
    - The PIC 16F88 is an 18-pin device that is equipped with two input/output ports, PORTA and PORTB.
    - PORTA has eight input/output (I/O) lines and the PORTB has eight I/O lines available.
        - PORTA I/O lines are labeled RA0, RA1, RA2, RA3, RA4, RA5, RA6, and RA7.
        - PORTB I/O lines are labeled RB0, RB1, RB2, RB3, RB4, RB5, RB6, and RB7.
    - Ordering direct from Microchip: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010243
    - PIC16F88 datasheet: http://ww1.microchip.com/downloads/en/devicedoc/30487c.pdf

- Pin Layout:



- **Programming PIC Microcontrollers:**
  - Outline of a PicBasic program:
    - PicBasic programs generally follow a predetermined format in the order given below.
      - Title
      - Program description
      - Revision history
      - Constants/Defines
      - Variables
      - Initialization
      - Main Code: During each main code cycle:
        - The MCU monitors the status of the input sensors
        - Executes the logic programmed in the main code
        - Changes the state of the output devices
  - Input/Output Port Registers and Input/Output Pins:
    - In a computer, registers are a set of data storage places that are part of a computer processor. A register may hold a computer instruction, a storage address, or any kind of data. In our immediate case, data stored in the I/O port register is 8-bit wide information about I/O pins. See lesson addendum for the PIC16F88 Register File Map.
    - Ports are connected to input/output (I/O) pins in a PIC. PORTB is the I/O port name for the I/O pins associated with PORTB.
    - Typically, an I/O port contains 8 pins. For example, PORTB has 8 pins.



PORTB

- The PIC16F88 has two ports, PORTA and PORTB.
- PORTA has 8 – I/O pins (RA0 – RA7) and PORTB has 8 – I/O pins (RB0 – RB7)
- Pins can be accessed in a number of ways. One way to specify a pin is to use its PORT name and bit number:

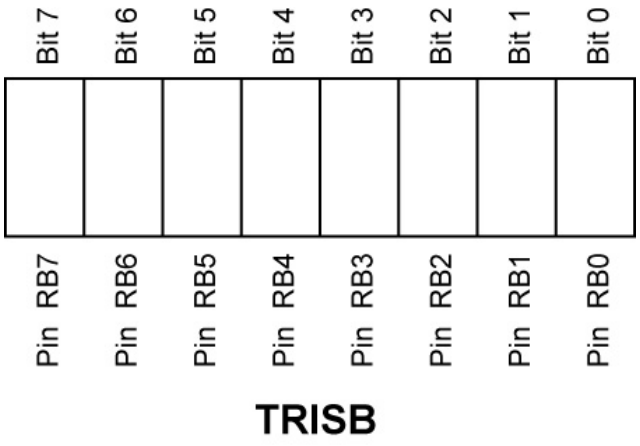
```
PORTB.1 = 1        ‘ Set PORTB, bit 1 to a 1 or HIGH (+5V)
                   ‘ PORTB.1 is pin RB1.

PORTA.3 = 0        ‘ Set PORTA, bit 3 to a 0 or LOW (0V)
                   ‘ PORTA.3 is pin RA3.
```

- Another way to access pins is to change the entire I/O port register.

```
PORTB = %00000000  ‘ Set all PORTB pins to LOW
PORTB = %11111111  ‘ Set all PORTB pins to HIGH
PORTA = %00000001  ‘ Set RA0 HIGH and RA1-RA7 to LOW.
```

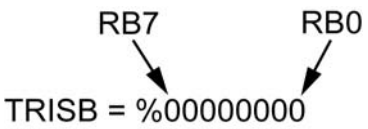
o Tri-State Registers (TRISx Registers):
- TRISx as the data-direction register name for PORTx. For example, TRISB is the TRIS register for PORTB.
- TRISx register is used to set up or initialize a pin as an input or an output. A “1” makes a pin an input and a “0” makes a pin an output. To remember, 1 looks like the I in Input and the 0 looks like the O in Output. Pins can be arranged in any combination of input and output.
- TRISB Register Order:

In graphic form:



TRISB

As written in a program:



- Example TRIS Registers:

```
TRISB = %00001111      ' Set RB0 – RB3 as inputs and
                        RB4 – RB7 as outputs.

TRISA = %00000001      ' Set RA0 as an input and RA1 –
                       RA7 as outputs.
```

- TRISx registers may be written in decimal form as well. TRISA = %00000000 can be written as TRISA = 0 and TRISB = %11111111 is the same as TRISB = 255.
- Individual bit directions may also be set.

```
TRISB.2 = 0            'Set pin PORTB.2 as an output.
```

- **New PicBasic Pro Commands:**
  - **PAUSE:**

    Format:
    PAUSE *Period*
    Explanation:
    Pause the program for *Period* milliseconds. *Period* is 16-bits, so delays can be up to 65,535 milliseconds (a little over a minute). **PAUSE** has the same accuracy as the system clock. **PAUSE** assumes an oscillator frequency of 4MHz. If an oscillator other that 4MHz is used, PBP must be told using a **DEFINE** OSC command. See the section on speed for more information.
    Example:

```
PAUSE 1        ' Delay for 1 millisecond
PAUSE 1000     ' Delay for 1000 milliseconds or 1
                second
```

    See page 112 in the melabs PicBasic Pro Compiler manual:
    http://melabs.com/resources/pbpmanual/

  - **GOTO:**

    Format:
    GOTO *Label*
    Explanation:
    Program execution continues with the statements at *Label*.
    Example:

```
LED1:                  ' Label named LED1

    PORTB.0 = 1        ' Set pin RB0 to HIGH (+5V)

    GOTO LED1          ' Program jumps to LED1
```

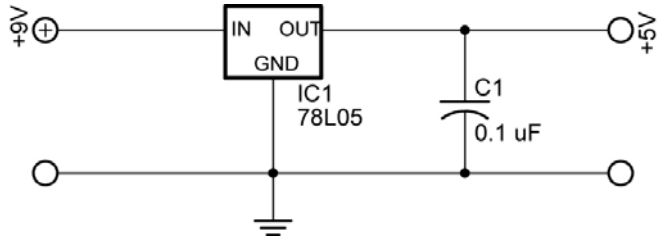
    See about page 146 in the PicBasic Pro Compiler manual:
    http://melabs.com/resources/pbpmanual/

- **END:**

  Format:
  END
  Explanation:
  Stop program execution and enter low power mode. All of the I/O pins remain in their current state. **END** works by executing a Sleep instruction continuously in a loop.
  An **END** or **STOP** or **GOTO** should be placed at the end of every program to keep it from falling off the end of memory and starting over.
  Example:

  END

- **Review of +5V Voltage Regulator Circuit:**
  - The circuit below will create a +5 V voltage source for the PIC microcontroller circuits.



+5 Volt Voltage Regulator Circuit

- **Complete LAB1 - blink1.pbp Program**
- **Procedure to Write, Compile and Download Your Program into the PIC Chip:**
  - Double click on the MicroCode Studio icon on your desktop.
  - Go to Desktop and open the folder with your name.
  - **Make sure you have opened your own folder before proceeding.**
  - Now open blink1.pbp if it is not already on one of the program tabs.
  - Type in your program or program changes.
  - Make sure that the microcontroller on the tool bar matches the microcontroller you are programming.
  - Click on Project on the tool bar.
  - Now double click on Compile and Program or strike F10. This step will automatically save your program and set up the .HEX file to be downloaded into the 16F88 chip.
  - On the melabs Programmer, make sure the 16F88 chip is selected.
  - If the 16F88 is inserted on the breadboard, carefully use the chip extractor tool to remove the chip from the breadboard.
  - Install the 16F88 microcontroller into the ZIF adapter; verify the 16F88 is pointing in the correct direction.
  - Find and click on the Program button to download the program into your chip. The LED will flash several times. Click on the OK button.
  - Remove the chip from the ZIF adapter and insert it into the proper position on your breadboard.

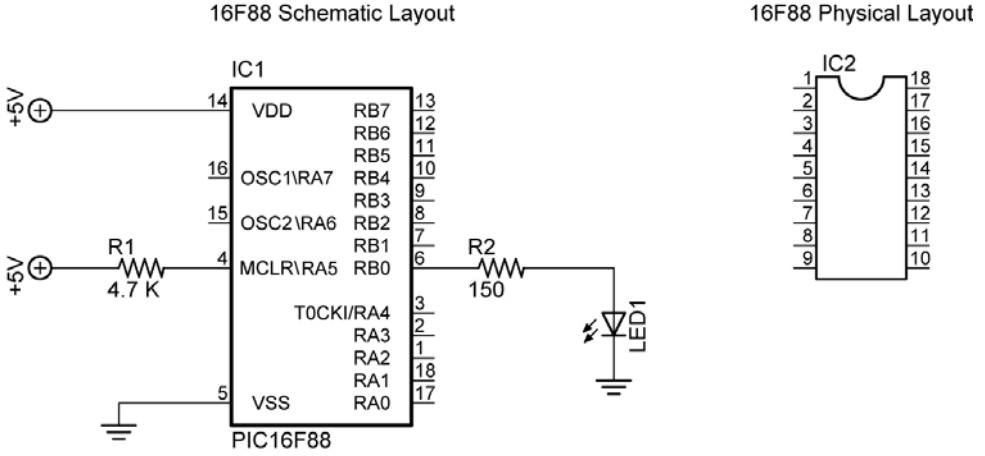
Cornerstone Electronics Technology and Robotics II
PIC Microcontrollers Programming 1 LAB 1 - blink1.pbp Program

- **Purpose:** The purpose of this lab is to acquaint the student on how to:
  - Compile a PicBasic program
  - Download a PicBasic program into a PIC16F88 microcontroller
  - Structure a program using three PicBasic commands
  - Make simple modifications to a PicBasic program
  - Make modifications to the TRIS register

- **Apparatus and Materials:**

  - 1 – Robotic Car by Student
  - 1 – Breadboard with +5V and +9V Power Supplies
  - 1 – 150 Ohm, ½ Watt Resistors
  - 2 – 470 Ohm, ½ Watt Resistors
  - 1 – 1K, ½ Watt Resistor
  - 1 – LED
  - 2 – DC Motors
  - 2 – 2N2222A NPN Transistors
  - 1 – 78L05 Voltage Regulator
  - 1 – 0.1 uF Capacitor

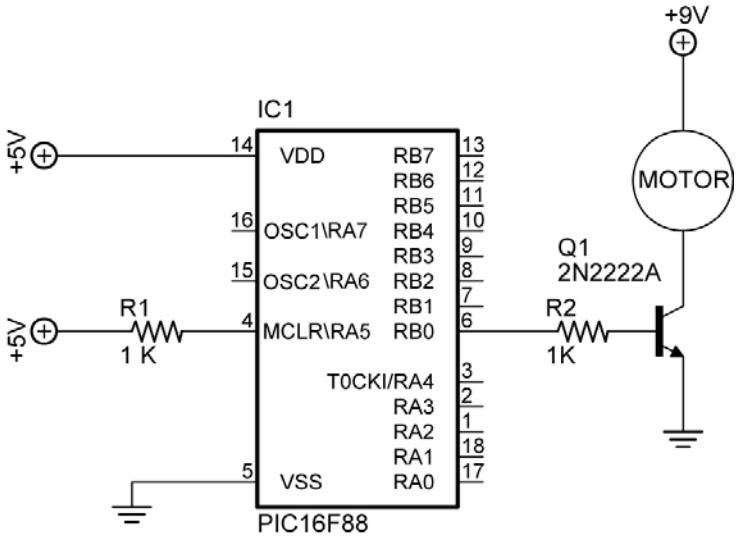
- **Procedure:**
  - Wire the blink1 circuit below on your robotic car's breadboard.



blink1

  - Program **blink1.pbp** into the PIC16F88 following the procedure to write, compile and download your program into the PIC chip at the end of the lesson.
  - Install the 16F88 and test the circuit and program
  - Change the PAUSE values (timing values) and reprogram the chip.

- **Challenges:**
  - Connect the resistor and LED to RB1 and make it blink. Save the program as **blinkrb1**. Remember to change the TRISB register to make RB1 an output.
  - **Railroad Crossing:** Wire a circuit using RB0 and RB1 as outputs and program the chip so that two LEDs alternate flashes like a railroad crossing. Let the flash time be 0.75 seconds. Save the program as **railrd1**.
  - **Drive a Motor:** Design a circuit and program a PIC16F88 which will drive a motor in one direction only. Name the new program **road1**.
    - Use a 9 vdc power source on a separate breadboard to drive the motor.
    - Tie the **grounds** of the +5 vdc and +9 vdc breadboards together, **but NOT the +5V and +9V power sources**. Use two batteries for the power sources; one for the +5 vdc and the other for the +9 vdc bus rows.
    - Step down the +9 vdc to +5 vdc using a 78L05 voltage regulator circuit. Verify the +5 vdc and +9 vdc bus rows with a DMM.
    - You may use notes from prior classes to review NPN transistor switches.
    - Hints:
      - Keep wires away from the 16F88 chip since it will be removed frequently from the circuit.
      - Place the motor on the on the collector side of the NPN transistor. Place a 1K ohm resistor between the 16F88 drive pin and the base of the NPN transistor. See the circuit below:



Transistor Switch as a Motor Driver

- **Drive a Robot:** Now combine this lesson's circuitry and programming to drive your robotic car through the taped course without crossing the inside boundaries of the tape. Revise the program **road1.**
  - You will have to use the process called dead reckoning since the robot is not equipped with any sensors. Wikipedia definition of dead reckoning: Dead reckoning is the process of estimating one's current position based upon a previously determined position, or fix and advancing that position based upon known speed, elapsed time, and course.
  - Hints:
    - Put the following code at the end of the program:

      PORTB.0 = 0 ‘ Set PORTB, bit 1 to a LOW (0V)

      PORTB.1 = 0 ‘ Set PORTB, bit 2 to a LOW (0V)

      PAUSE 1 ‘ Pause 1 millisecond

      This code will stop the robotic car.

PIC16F88 Register File Map

	File Address		File Address		File Address		File Address
Indirect addr. *	00h	Indirect addr. *	80h	Indirect addr. *	100h	Indirect addr. *	180h
TMR0	01h	OPTION-REG	81h	TRM0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCOM	105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
	07h		87h		107h		187h
	08h		88h		108h		188h
	09h		89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h		91h				
T2CON	12h	PR2	92h				
SSPBUF	13h	SSPADD	93h				
SSPCON	14h	SSPSTAT	94h	General Purpose Register 16 Bytes		General Purpose Register 16 Bytes	
CCPR1L	15h		95h				
CCPR1H	16h		96h				
CCP1CON	17h		97h				
RCSTA	18h	TXSTA	98h				
TXREG	19h	SPBRG	99h				
RCREG	1Ah		9Ah				
	1Bh	ANSEL	9Bh				
	1Ch	CMCON	9Ch				
	1Dh	CVRCON	9Dh				
ADRESH	1Eh	ADRESL	9Eh				
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h
			Efh		16Fh		1EFh
		accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Bank0		Bank1		Bank2		Bank3	

Unimplemented data memory locations, read as '0'.

* Not a physical register.