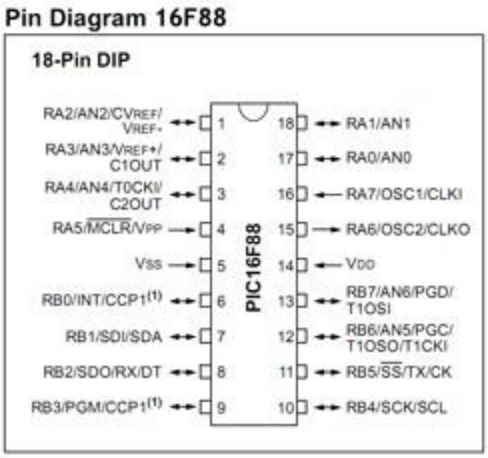
16F88 PIC Experimenter's Board

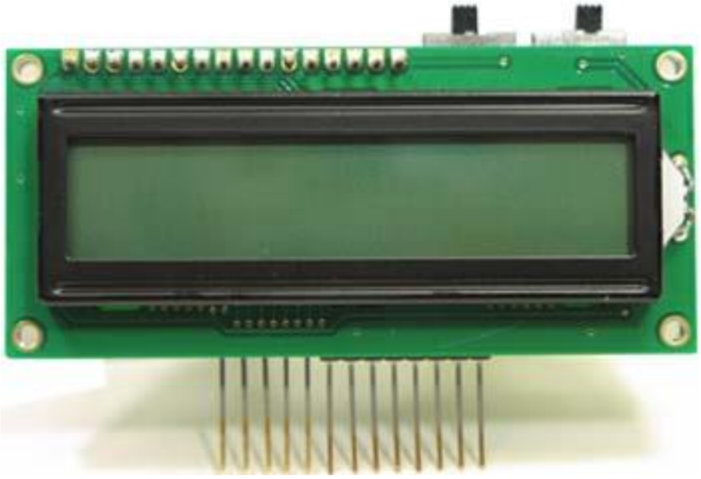
Introduction

The 16F88 PIC Experimenter's Board is a complete prototyping platform for the 16F88 18-pin PIC microcontroller. This article is one the use of the 16F88 PIC Experimenter's Board. It assumes you already know how to write PICBasic programs, compile the program to a hex file, and upload the hex file (firmware) into the 16F88 PIC Microcontroller. For beginner's needing tutorials on writing PICBasic Programs, compiling and uploading firmware, go here.



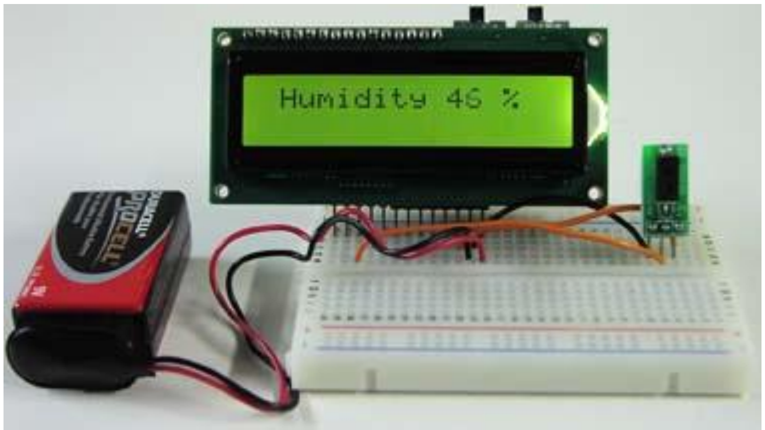
Main 16F88 Features:

Internal Oscillator -plus 3 Xtal Modes up to 20 MHz.
10 Bit, 7 Channel Analog-to-Digital Converter
2 Comparators
Asynchronous Receiver Transmitter
7168 Bytes memory
368 SRAM
256 EEPROM - Data Retention > 40 Years
100,000 erase/write cycles
Operating voltages 2.0V to 5.5V



16F88 PIC Experimenter's board

The 16F88 PIC Experimenter's board is designed to be plugged into a solderless bread board connection for quick and easy access to Port A (RA5, RA6, RA7) and Port B (RB0, RB1, RB2, RB4, RB5, RB6, RB7) I/O lines. It includes an integrated 16x2 LCD display with backlight.



PINS or I/O Lines

When discussing connections to the 16F88 I use two terms interchangeably, pins or I/O lines (Input/Output lines). I imagine one could argue the nomenclature, technically pins are on the microcontroller and the I/O lines are the connections to the pins. Beyond this, Input/Output means we can configure the I/O line to be either and input or output line. While the I/O line can not be both input and output simultaneously, the program can switch the lines state from input to output and vice versa. This switching can happen so fast that it may give the impression that it is functioning as both.

Use

The board can be powered by either a 9V battery or a AC/DC transformer. The power switch in the upper right turns the board on and off. The LCD backlight switch is next to the power switch, located on the right hand side above the LCD. The prototyping area is as large as the solderless breadboard you plug the 16F88 experimenter's board into. For those who have not dabbled in electronics we will describe the usage of the solderless bread board and wire a simple LED project. See figure 1.

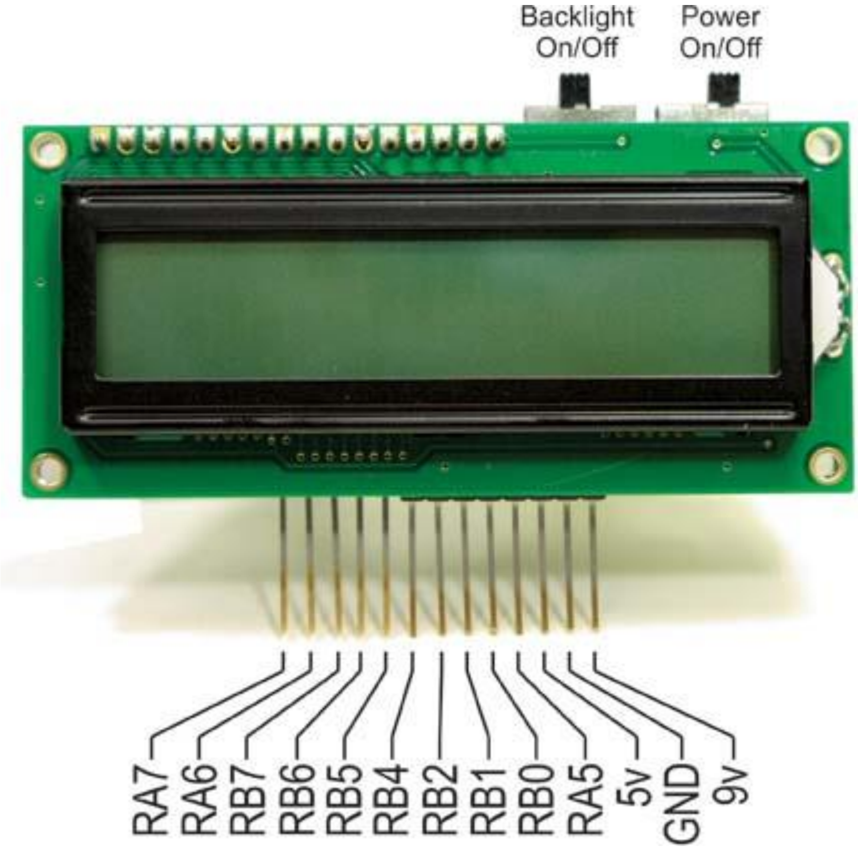
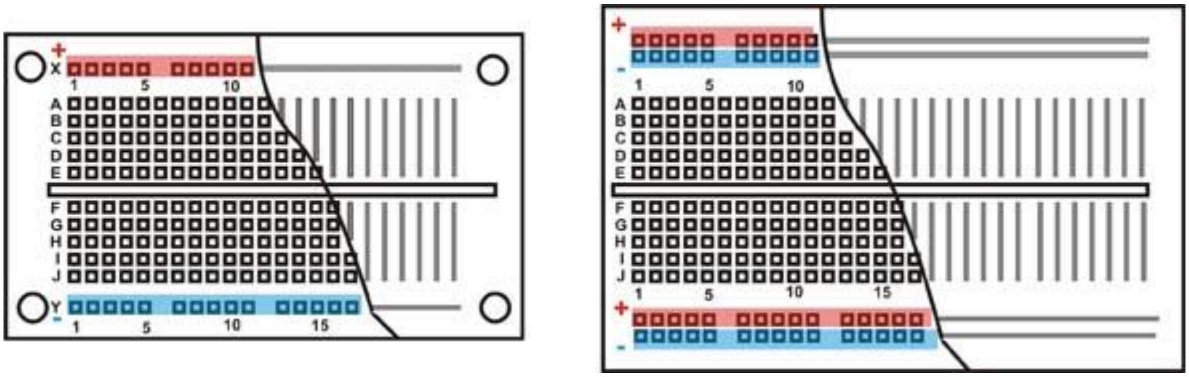


Figure 1

The design and function to solderless breadboards is shown in Figure 2.

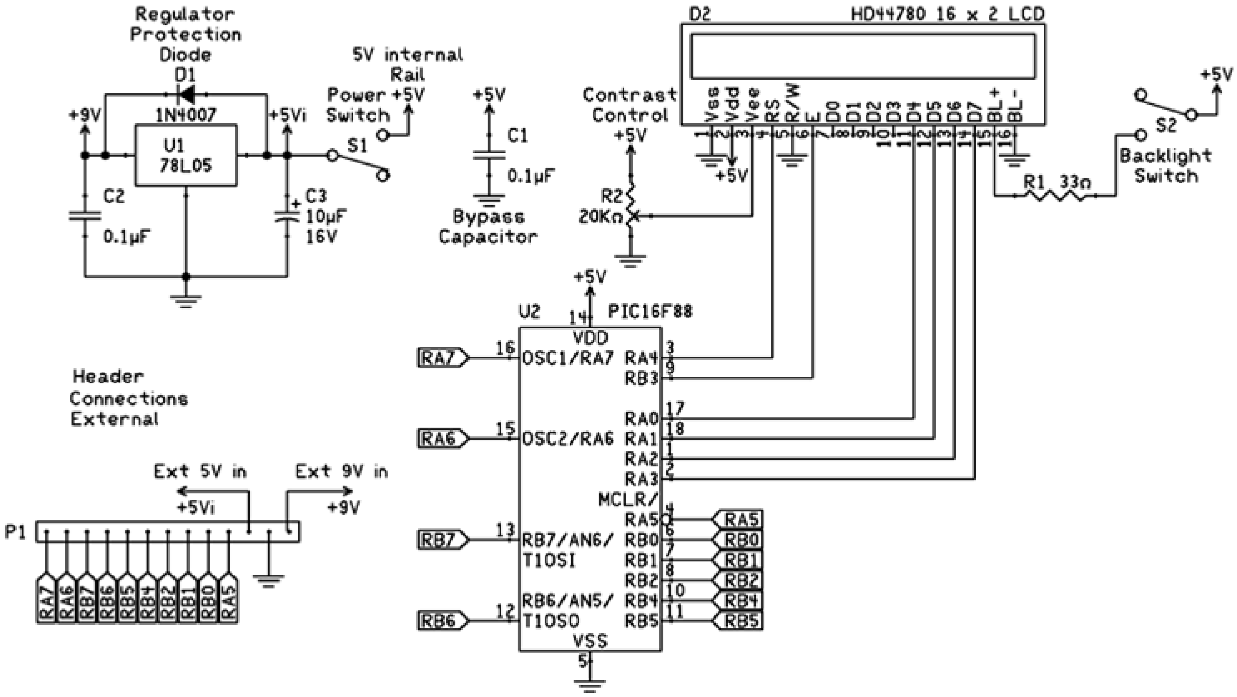


You can breadboard (assemble and connect) electronic components and electronic circuits into the prototyping area without soldering. The prototyping area is reusable; you can change, modify or remove circuit components at any time. This makes it easy to correct any wiring errors. A cutaway of two common solderless breadboards are shown above.

The square holes shown in the area are sockets. When a wire or pin is inserted into a hole, it makes electrical contact with the underlying metal strip. The holes are spaced so that integrated circuits and many other components can be plugged right in.

The internal electrical connection structure of the solderless breadboard is shown in the cutaway.

Simple Experiment:
We shall wire a simple experiment to illustrate the usage of the experimenter's board for prototyping. Before we do so Let's look at a schematic of the circuit on the experimenter's board.



This schematic is comprehensive but not complete. In the full schematic there are options for external crystals on pins 15 and 16, as well as 12 and 13. We are not using crystals in our circuit(s) just yet. Instead we are using the internal oscillator of the 16F88.

We can simplify our schematic even further. It's not necessary to continually show the PIC 16F88, power supply and LCD module. Instead I will only show the header pinout to the experimenter's board along with the rest of the circuit. The schematic below is to blink an LED. The following is a small PBP program to blink an LED on PIN RB1.

PIC 16F88 Configuration Register

The 16F88 has two 14-bit configuration registers at memory locations 2007H and 2008H. The bits in the PIC's configuration register enable or disable features in the microcontroller. Features such as Code Protection, Watchdog timer, Oscillator, etc.

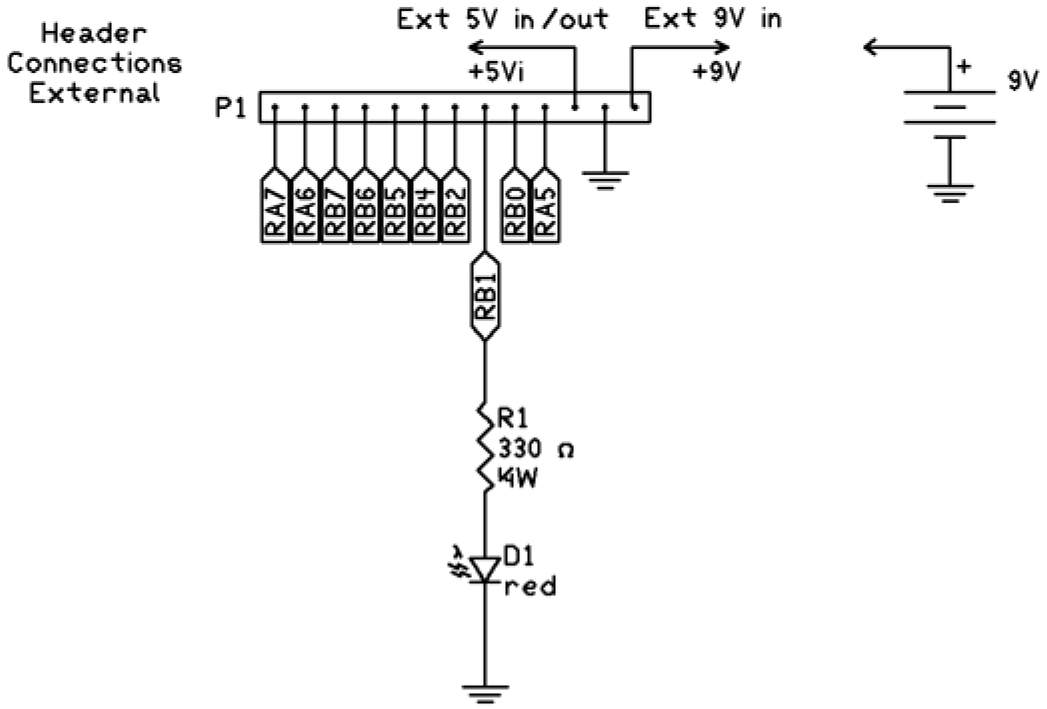
Using PBP compiler and MicroCode Studio IDE allows you to set the configuration switches inside the program. We only need to set a few bit registers and leave the rest at their default status. Add the following three lines to the top of your program to set the configuration switches.

#CONFIG
__config _CONFIG1,_INTRC_IO & _WDT_OFF & _PWRTE_ON & _MCLR_OFF & _LVP_OFF & _CP_OFF
#ENDCONFIG

These settings are applicatable for most of our programming needs at this moment. For more information on this Configuration Setup go here.

```
'BLINK PBP Program 1.1
#CONFIG
__config _CONFIG1,_INTRC_IO & _WDT_OFF & _PWRTE_ON & _MCLR_OFF & _LVP_OFF & _CP_OFF
#ENDCONFIG
osccon = %01110100        'Set-up internal oscillator at 8 MHz
start:                    'Start of routine
High PortB.1              'Turn on LED
Pause 500                 'Wait 1/2 second
Low PortB.1               'Turn off LED
Pause 500                 'Wait 1/2 second
Goto start                'Goto Start and repeat
```

Download PBP 1.1 Source   Download PBP 1.1 Hex File

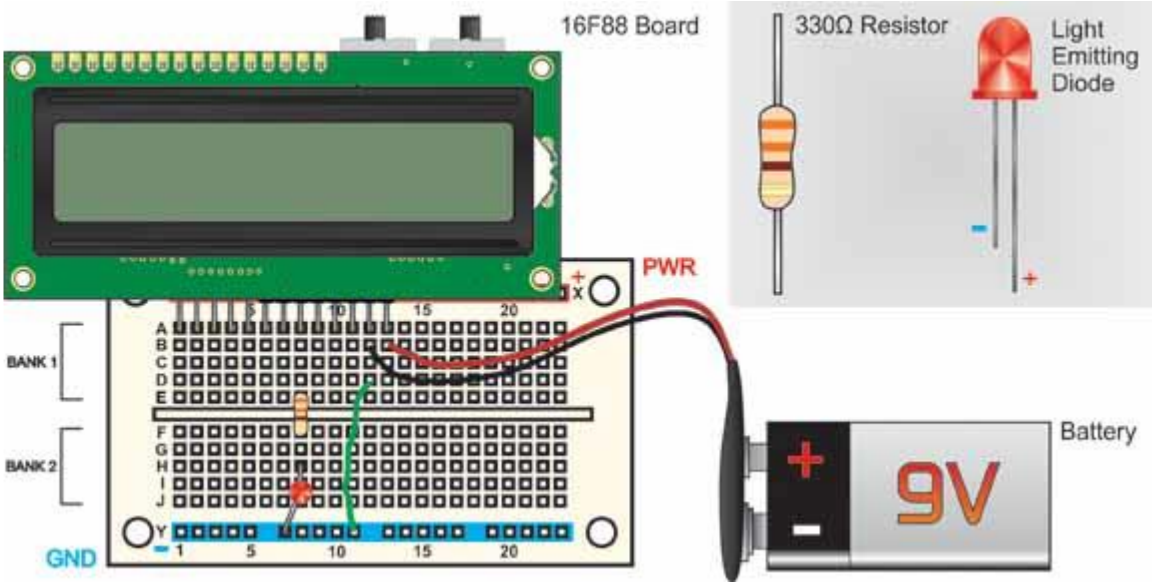


Aside from a programmed 16F88 we only need a 9V battery and two other components; a 330-ohm ¼ watt resistor and a sub-miniature LED. All the other components needed to make the 16F88 work are already hard wired on the PIC Experimenters Board.

The LED has two terminals, one longer than the other. On the schematic it is shown as a diode. The longer terminal on the LED is positive, shown in the legend of figure below.

To wire this circuit, Plug the 16F88 project board into the solderless breadboard. Connect a 9 volt battery to the 9V terminal and ground termin repsectively. Connect a wire from the ground of the 16F88 project board to the ground strip of the solderless breadboard. Shown as the green wire in the drawing. Next connect one lead of the ¼ watt resistor into one of the RB1 sockets. Connect the other lead of the ¼ watt resistor into a socket in Bank 2.

Take the positive lead of the LED and plug it into a socket in the same column as the one containing the resistor lead. Connect the opposite lead of the LED and plug it into one of the ground sockets at the bottom, see figure below.



Plug the programmed 16F88 microcontroller into the 18 pin socket on the PIC Experimenters Board and turn on the power. The LED should begin blinking on and off. This on-off cycle (blinking) continually repeats.

PIC Experimenter's Board Manual

Built-in LCD

We have an LCD module build on our experimentors board that allows our microcontroller to communicate to us using alpha-numeric text. To use the built-in LCD, we need to add a few lines of additional code.

```
ANSEL = %01100000            'Set RA0-RA4 as digital I/O for LCD leave RA5 and RA6 A/D

pause 1000

LCDOUT 254,1, "LED is Blinking"
```

The first line ANSEL = %01100000 sets the pins (or I/O lines) RA0 to RA4 to digital I/O lines. These lines are dedicated to making the LCD function and are not accessible to us for other uses.

The next line Pause 1000, stops the program execution for one second. This is to allow the LCD to complete its own power up initiation and be available for use.

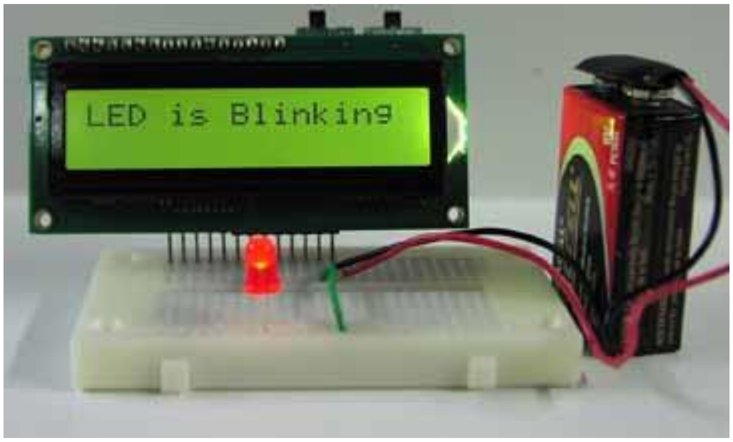
The next line LCDOUT 254,1, "LED is Blinking", clears the LCD screen, positions the cursor on line 1 position 1 then prints the text "LED is blinking" on the LCD. The LCDOUT command structure is discussed below.

Updated Blink Program with LCD Message

Here is the updated PBP program with the three addition lines:

```
'BLINK with LCD Message PBP Program 1.2
#CONFIG
__config _CONFIG1,_INTRC_IO & _WDT_OFF & _PWRTE_ON & _MCLR_OFF & _LVP_OFF & _CP_OFF
#ENDCONFIG
osccon = %01110100        'Set-up internal oscillator at 8 MHz
ANSEL = %01100000         'Set RA0-RA4 as digital I/O for LCD
Pause 1000
LCDOUT 254,1, "LED is Blinking"
start:                    'Start of routine
High PortB.1               'Turn on LED
Pause 500                 'Wait 1/2 second
Low PortB.1                'Turn off LED
Pause 500                 'Wait 1/2 second
Goto start                'Goto Start and repeat
```

Download PBP 1.2 Source   Download PBP 1.2 Hex File



LCDOUT command:

The LCD Module has two operational modes: text and instruction. Both text and commands are access using the same PBP command "LCDOUT". Text and commands may be combined on the same LCDOUT program line, as is the case with our LCDOUT command. ASCII text are called strings. To print the string “Images” use the Command (LCDOUT “Images”) and the text "Images" will appear on the LCD. The string text to be printed is surrounded by double quotation marks "". These double quotation marks are not printed on the LCD.

To input instructions to the LCD module, you must prefix the instruction with ASCII 254 ($FE). The byte following prefix is seen and treated as a instruction code. Every instruction code must be sent with its own 254 prefix. The clear-screen instruction is ASCII 1.

To clear the LCD screen and place the cursor on the first line first position use the command LCDOUT 254,1

You can also use the hex numbers to send commands. The hex equivalent to decimal number 254 is $FE. So to clear the screen you can also use the command LCDOUT $FE,1.

LCD Instruction Codes

Instruction
Clear Screen
Home position (move cursor top left of display)
Move cursor one character position left
Move cursor one character position right
Scroll display one character position left
Scroll display one character position right
Set cursor porition (DDRAM address)
Set point in character-generator (CG) Ram

Code (Decimal)
1
2
16

20
24
28
128+addr
64+addr

Character Display and off screen characters

The LCD displays the first 16 characters on each line. However, each LCD line can hold 40 characters. When you print past the 16 visible characters, the next 24 characters are in an off-screen memory area. This LCD module has 80 bytes of character memory, arranged appropriately for a 2x40 screen. While the off-screen text can't be seen on the screen, one could use scroll instructions to scroll and reveal the off screen character text.

Back Light and Contrast Control

Your LCD has a back light that is turned on or off using the back light switch. A potentiometer on the back of the board adjusts contrast of the LCD. You can adjust the contrast control by hand or your optimum viewing.

More information on the LCD is available in the Serial LCD Module manual.

Timing

Before moving on to Binary numbers, lets take a quick look at timing. The PIC microcontroller is accurate in regard to timing. If you built the LED blinker you may have noticed that the LED is turning on-off faster than the program cycle calls for. The program cycle states the LED should turn on for 1/2 second then off 1/2 second where the process repeats. While we set the internal oscillator speed to 8 MHz the PBP compiler does not know the oscillator speed we have set. We have to tell it. While our LED blinking program is a trival timing issue, other processes like serial communication require precise timing to be reliable. We can tell the PBP compiler the speed of the oscillator by adding one line to our program.

```
Define OSC 8        'let pbp know about oscillator value
```

Updated Blink Program with LCD Message and PBP Set Oscillator

```
'BLINK with LCD Message and PBP Set Oscillator Program 1.3
#CONFIG
__config _CONFIG1,_INTRC_IO & _WDT_OFF & _PWRTE_ON & _MCLR_OFF & _LVP_OFF & _CP_OFF
#ENDCONFIG
osccon = %01110100        'Set-up internal oscillator at 8 MHz
Define OSC 8              'let pbp know about oscillator value
ANSEL = %01100000         'Set RA0-RA4 as digital I/O for LCD
Pause 1000
LCDOUT 254,1, "LED is Blinking"
start:                    'Start of routine
High PortB.1              'Turn on LED
```

```
Pause 500               'Wait 1/2 second
Low PortB.1              'Turn off LED
Pause 500               'Wait 1/2 second
Goto start               'Goto Start and repeat
```

Download PBP 1.3 Source Download PBP 1.3 Hex File

Run this program and the LED timing cycle should be accurate.

PIC 16F88 Experimenter's Board Manual

Experiments with Binary and the PIC Microcontroller

In the Beginning there is Binary

The term binary means "based on two" as in two numbers, 0 and 1. It's also like an electrical switch that has two values on (1) and off (0).

The term bit is an acronym that stands for the term Binary digit. A bit or binary digit can have two values, either 0 or 1. A byte is a digital expression (number) containing 8 bits.

Binary is important to computers and microcontrollers. The bit values of (0) and (1) are the only things a computer can read. Actually the computer or microcontroller can't really read, but it can sense voltage values. So a bit that is on (1), is represented by a positive voltage. Consequentially a bit (0) is off (0) and is represented as no voltage.

A single bit by itself is of little value, but start putting them together to make Bytes (8-bits), Words(16-bits) ((32 bits) (64 bits) (128) and so on) and we can make the computers perform mathematics, create word processors, spreadsheets , create a cyberspace (internet) etc. All these amazing things based on a bit.

Reading and Writing

To read or write to microcontroller pin (I/O line) or a port register requires understanding a little binary. When we read the staus of an I/O line or pin on our microcontroller we are seeing if there is a binary 1 or binary 0 present of the line. When we write to any pin or I/O line we are outputting either a binary 1 or binary 0. When reading or writing to a port, we can influence all the pins or I/O lines available on that port at one time. While we may typically use standard decimal numbers when writing to or reading a port status, it is the binary equivalent of that number that the microcontroller uses. Therefore when writing to a port you can use decimal (or hexadecimal) numbers but only if you can understand the binary number equivalen that the PIC Microcontroller uses.

Setting Individual Pins to be either Inputs or Output Pins

PBP compiler has commands that allows you to read and write to indivial pins or complete ports at a time. The command INPUT will cause the identified pin (following the command) an input pin. So for example INPUT PORTA.0 will make PORTA, pin 0 an input. To turn this pin into an output you can use the PBP command OUTPUT. The command OUTPUT will cause the identified pin (following the command) an output pin. So for example OUTPUT PORTA.0 will make PORTA, pin 0 an output. The advantange of using these commands is that it also sets the Data Direction Register (DDR) or TRISA at the same time. See explaination and access of TRIS registers below. The disadvantage is that in program execution time is slower than if you set the TRISA yourself. Typically the program execution time is so quick, the slight decrease in time is not noticable.

What are the PORTA, PORTB, TRISA and TRISB Registers?

The 16F88 has two 8-bit bi-directional ports; PortA and PortB. Each bit of these ports correspond to a PIN on the 16F88 microcontroller. Each of these Ports has a Data Direction Register TRISA for PortA and TRISB for PortB. The TRIS bit determines whether the corresponding bit on the port is either an input or output pin. Setting the TRISA bit (= 1) will make the corresponding PORTA pin an input. Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output.

Accessing the Ports and Registers

The 16F88 uses 8-bit registers for PORTA, PORTB, TRISA and TRISB registers, so we only need to concern ourselves with small 8 bit numbers and their decimal equivalent right now. Accessing a port or register on the PIC microcontroller allows you to set eight I/O lines or pins at once. Remember an 8 bit number is called a byte. An 8-bit number can represent any decimal value between 0 and 255. When we write a decimal number into a register, the PIC microcontroller can only see the binary equivalent of that decimal number (byte) we wrote to the register. For us to understand what's happening inside the register we need to be able to look at the binary equivalents of the decimal (byte) number also. Once we can do this, our ability to effectively and elegantly program the PIC microcontrollers is enhanced.

Examine the binary number table below, it shows all the decimal and binary number equivalents for numbers 0 through 32. Using this information, the binary numbers from 32 to 255 can be extrapolated.

Each decimal number on the left side of the equal sign has its binary equivalent on the right side. So where we see a decimal number, the microcontroller will see the same number as a series of eight bits. (bits -- eight bits to a byte)

Binary Number Table

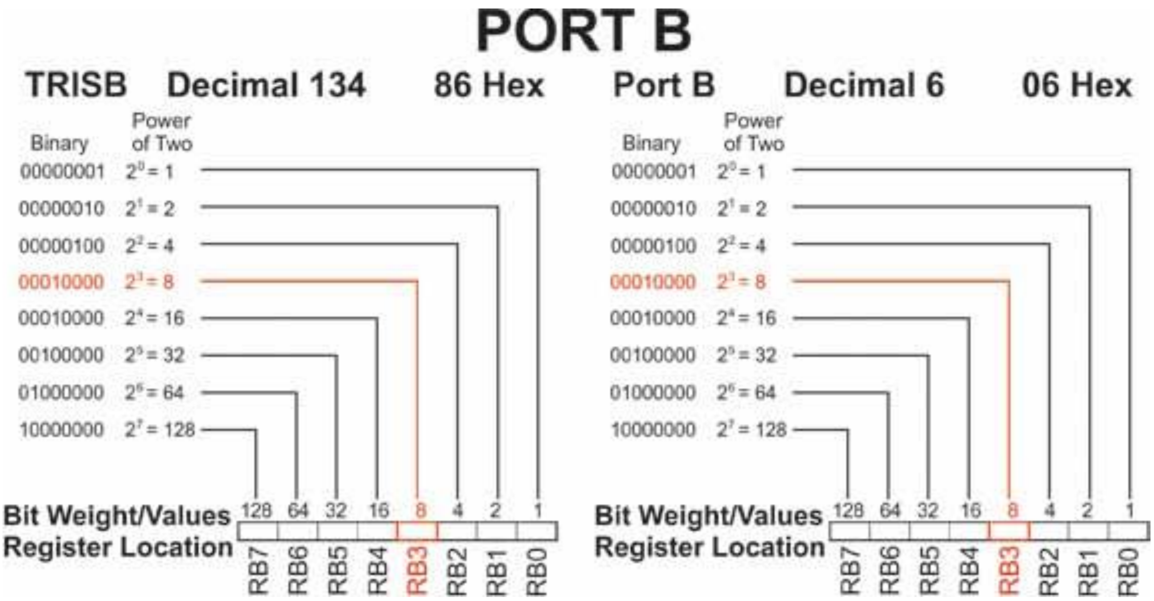
0 = 00000000
1 = 00000001
2 = 00000010
3 = 00000011
4 = 00000100
5 = 00000101
6 = 00000110
7 = 00000111
8 = 00001000
9 = 00001001
10 = 00001010
11 = 00001011

12 = 00001100
13 = 00001101
14 = 00001110
15 = 00001111

16 = 00010000
17 = 00010001
18 = 00010010
19 = 00010011
20 = 00010100
21 = 00010101
22 = 00010110
23 = 00010111
24 = 00011000
25 = 00011001
26 = 00011010
27 = 00011011
28 = 00011100
29 = 00011101
30 = 00011110
31 = 00011111

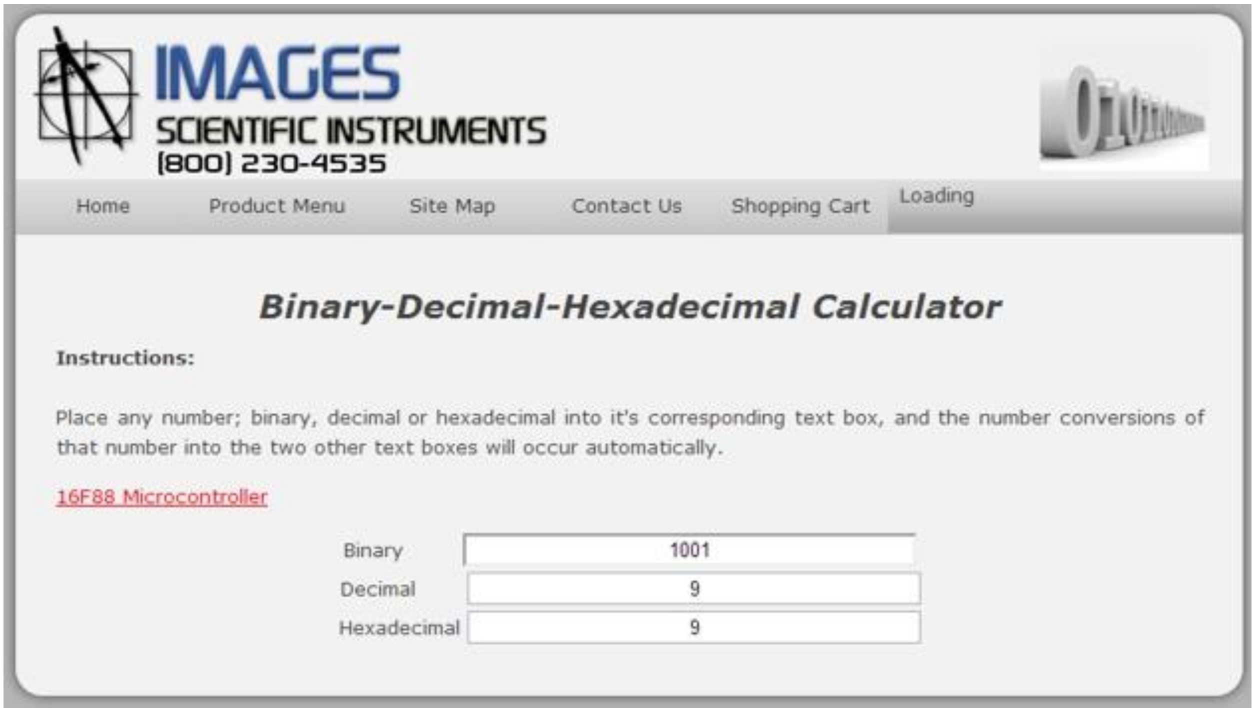
32 = 00100000
.
.
.
64 = 01000000
.
.
.
128 = 10000000
.
.
.
255 = 11111111

Figure 1 (below) shows the relationship between a binary number and the two PIC microcontroller registers that control Port-B. Notice each register has eight open positions. This register can hold an 8-bit (1 byte) number.



Binary-Decimal-Hexadecimal Calculator

The following link brings you to a binary-decimal-calculator page. Binary-Decimal-Hexadecimal Calculator Page. Place any number; binary, decimal or hexadecimal into it's corresponding text box, and the number conversions of that number into the two other text boxes will occur automatically.



Binary-Decimal-Hexadecimal Calculator

Instructions:

Place any number; binary, decimal or hexadecimal into it's corresponding text box, and the number conversions of that number into the two other text boxes will occur automatically.

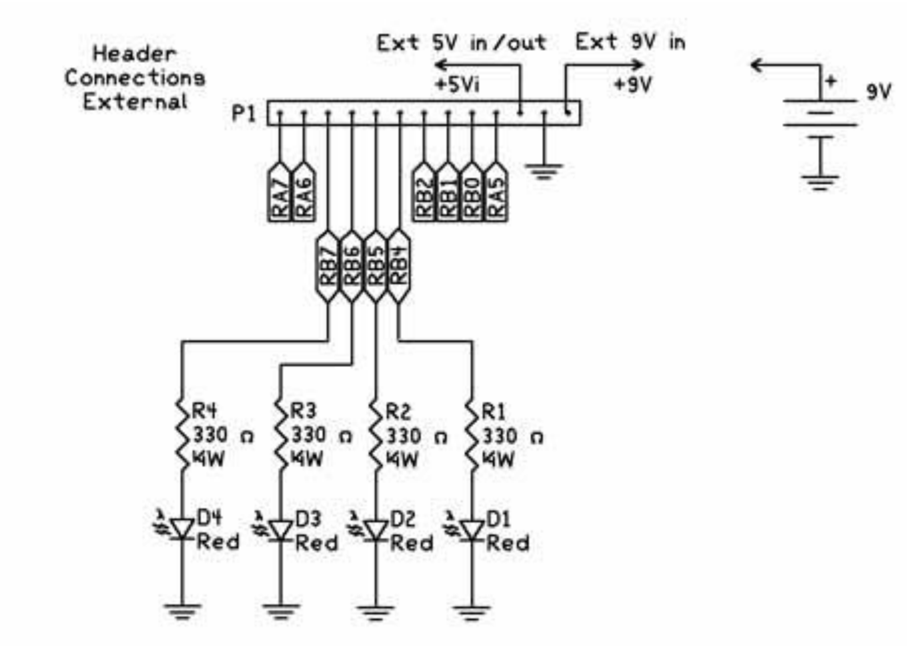
16F88 Microcontroller

Binary	1001
Decimal	9
Hexadecimal	9

Counting in Binary

To reinforce the binary number system lets write a program that counts from 1 to 16 and displays the decimal number in binary using four LED's. If we look at figure 1 we see RB3 is highlighted in red. It is highlighted in red because that pin I/O line has been assigned to the LCD display. Therefore we ought to leave that pin/line alone. Our counting program from 1 to 16 would typically use the pins I/O lines RB0 to RB3. Since we can not use RB3, what we will do in this program is shift the number four places to the left to RB4 to RB7 and display the LED's

Below is the schematic for the Binary counting program.



'Binary Counting Program

```
osccon = %01110100      'Set-up internal oscillator at 8 Mhz
define OSC 8            'define the oscillator frequency
ANSEL = %01100000       'Set RA0-RA4 as digital I/O for LCD leave RA5 and RA6 A/D
trisb = 0               ' Set Port B to outputs
pause 1000              'Initalize time for LCD
j var byte              'Create variable j
i var byte              'Create variable i
start:                  'Label for start of program
For J = 0 to 15         'Start counting loop
i = j << 4              'shift number 4 places to right
```

```
portb = i                   'display binary number using LEDs on RB4 to RB7
pause 500                   'Delay to give human time to see LED binary number
next j                      'next number
goto start                  'Start from the beginning again
```

Our Binary counting program works pretty well. Notice that the "j" variable is between 0 and 15. This is because computers start counting at zero, while we humans starting counting at one. One thing missing from out program is using the LCD Display to provide information on program executuion. With a few more lines we can add an LCD routine that shows the decimal equivalent of the binary number shown on the LEDs. If the program runs too fast for you increase the Pause 500 line to Pause 1000.

'Binary Counting Program with LCD

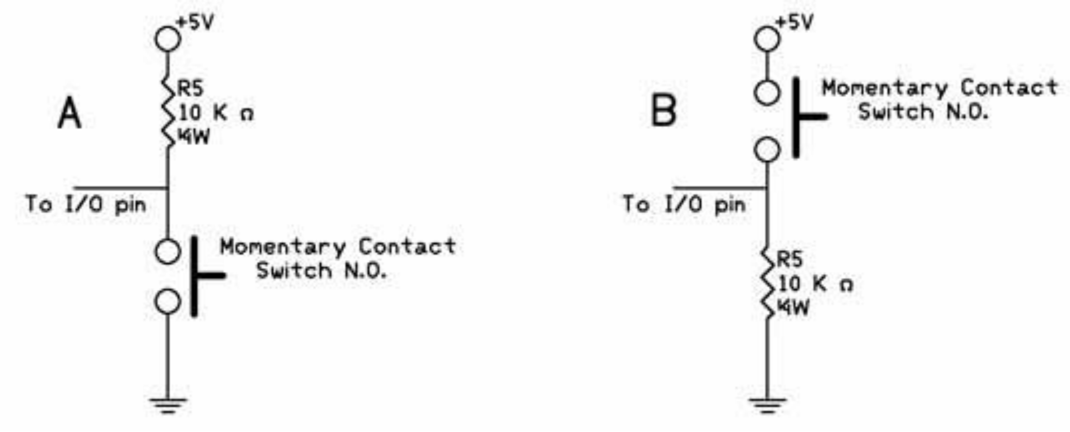
```
osccon = %01110100                  'Set-up internal oscillator at 8 Mhz
define OSC 8                        'define the oscillator frequency
ANSEL = %01100000                   'Set RA0-RA4 as digital I/O for LCD leave RA5 and RA6 A/D
trisb = 0                           ' Set Port B to outputs
pause 1000                          'Initalize time for LCD
LCDOUT 254,1, "Binary Counting"     'Print on LCD line 1
j var byte                          'Create variable j
i var byte                          'Create variable i
start:                              'Label for start of program
For J = 0 to 15                     'Start counting loop
LCDOUT 254,192, "                " 'Clear printing on LCD line 2
LCDOUT 254,192, "Binary LED # ", #j     'Print on LCD line 2

i = j << 4                              'shift number 4 places to right

portb = i                               'display binary number using LEDs on RB4 to RB7

pause 500                               'Delay to give human time to see LED binary number

next j                                  'next number

goto start                              'Start from the beginning again
```

PIC Experimenter's Board Manual

Input

The ability of our microcontroller to read the electrical status of its pin(s) allows the microcontroller to see the outside world. The line (pin) status may represent a switch, sensor, or electrical information from another circuit or computer. And as we'll see later can read an analog voltage on a pin using the 16F88 microcontroller's built in Analog/Digital (A/D) converter.

If a +5 TTL voltage is present on a pin (I/O) line and we looked at the line we would get a binary "1". If zero voltage or ground is present on a pin (I/O) line and we looked at the line we would get a binary "0". The figure belows shows two simple switch configurations that can place either a binary "1" at a pin (switch A) or a binary "0" (switch B). The status of the line reverses if one presses down on the momentary switch.



Lets take another look at the switch schematic in figure 4 before we start using the switch in a program to visualize how the switches affect the I/O pin electrically.

The switch labeled "A" connects the I/O pin to a +5V power supply through a 10,000 ohm resistor. With the switch open, the electrical status of the I/O pin is kept high (binary 1). When the switch is closed, the I/O pin connects to ground, and the status of the I/O pin is brought low (binary 0).

The switch labeled "B" has an electrical function opposite the switch labeled A. In this case, when the switch open the I/O pin is connected to ground keeping the I/O pin low (binary 0). When the switch is close the I/O pin is brought high (binary 1).

In place of a switch, we can substitute an electrical signal, high or low, that can also be read usig the Button command.

Typically checking the switch status is accomplished inside a program loop, where the program is looking for a change of state (switch closure). When the state of the I/O pin (line) matches the state defined in the commandr, the program execution jumps out of the loop to the "Label" portion of the program.

We can use standard PBP commands to read the status. Let's say our switches are connect to RB0. In the case of the A switch, the line is normally high (binary 1), so to detect a switch press we want to see a binary "0".

```
lp:
if portb.0 = 0 goto routine2
goto lp
routine2:
```

In the program snippet above, the line "if portb.0 = 0 then goto routine2" checks the status of pin RB.0. In the case where RB0 = 1 the routine jumps back to start and keeps re-checking the statud until it changes. The program can be easily modified to check for a binary "1", for the use with switch B, see below.

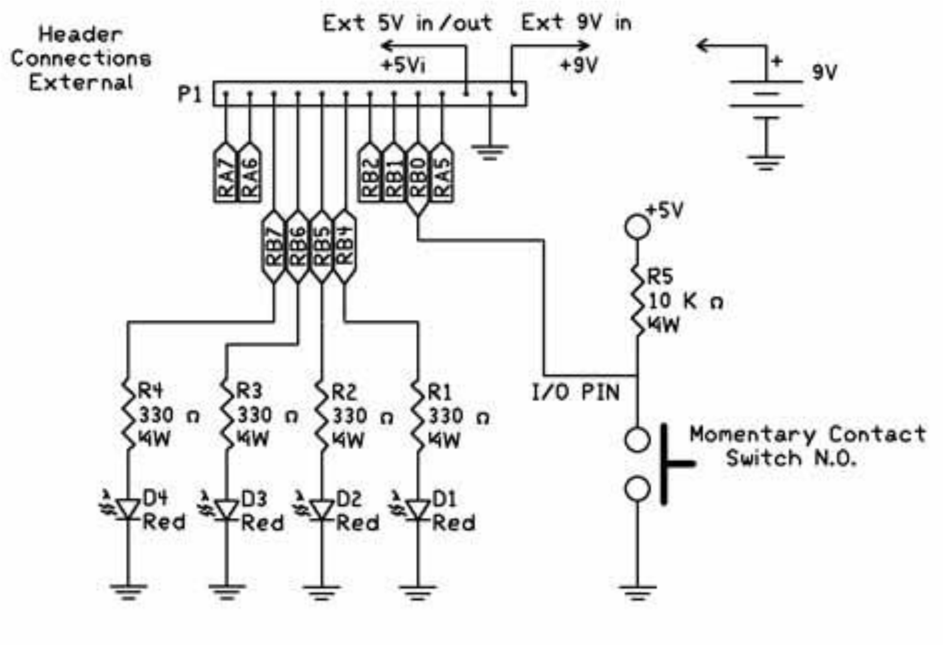
```
lp:
if portb.0 = 1 goto routine2
goto lp
routine2:
```

Reading an Input Line

Now its time to use a momentary contact button to our application. By adding a switch we can have the program stop or start at the press of a button. The image below is a minature momentary contact push button switch Normally Open (N.O.)

This particular momentary contact switch has terminals that are for soldering directly to a pc board, but also plug right into our solderless bread board making connection into our circuit easy.

For this example we will have the program stop counting while the switch is pressed and resume where it left off when the switch is released. The first thing we need to do is add our switch to the circuit. The schematic follows.



The next thing we need to do is set PortB.0 line to an input. The TRISB = 0 command set all Port B pin/lines to outputs. We can use the INPUT command and add a line in our program INPUT PORTB.0 which will change this line to an input. Alternatively we can use the TRISB = 1 command to set most of Port B lines to outputs with the except of PortB.0 which is set to an input. We will be going over using 8-bit ports and resgisters a little later.

```
osccon = %01110100                  'Set-up internal oscillator at 8 Mhz

define OSC 8                        'define the oscillator frequency

ANSEL = %01100000                   'Set RA0-RA4 as digital I/O for LCD leave RA5 and RA6 A/D

trisb = 0                           ' Set Port B to outputs

input portb.0                       ' Set Port B.0 to input

pause 1000                          'Initalize time for LCD

LCDOUT 254,1, "Binary Counting"     'Print on LCD line 1

j var byte                          'Create variable j

i var byte                          'Create variable i

start:                              'Label for start of program

For J = 0 to 15                     'Start counting loop

lp:                                 'Start of loop routine to read switch
```

```
if portb.0 = 1 then routine2                 'If PB.0 = 1 (high) exit loop
goto lp                                      ' If PB.0 = 0 (low) stay inside loop, stop counting
routine2:                                    'If PB.0 = 1 (high) exit loop here
LCDOUT 254,192, "                  "         'Clear printing on LCD line 2
LCDOUT 254,192, "Binary LED # ", #j          'Print on LCD line 2
i = j << 4                                   'shift number 4 places to right
portb = i                                    'display binary number using LEDs on RB4 to RB7
pause 500                                    'Delay to give human time to see LED binary number
next j                                       'next number
goto start                                   'Start from the beginning again
```

The Button Command

The PBP Compiler comes equipped with a simple command to read the electrical status of a pin called the Button command. The Button command while useful has a few limitations. One is that you can not read multiple port pin inputs at once, only one pin at a time. We will overcome these button command limitations when using the peek command. But for the time being lets use and understand the button command.

As the name implies the Button command is made to read the status of an electrical "button" switch connected to a Port pin. We can use the same two basic switch schematics, labeled A and B, connected to a I/O pin. The Button Command structure is as follows:

Button *Pin, Down, Delay, Rate, BVar, Action, Label*

Pin	Pin number (0 – 15), or pin name (e.g Portb.0)
Down	State of pin when button is pressed (0 / 1)
Delay	Cycle count before auto repeat starts (0-255). If 0, no debounce or auto-repeat is performed. If 255, debounce, but no auto-repeat,is performed.
Rate	Auto-Repeat Rate (0-255).
Var	Byte sized variable used for delay/repeat countdown.Should be initialized to 0 prior to use.
Action	State of button to perform GOTO (0 if not pressed, 1 if pressed).

Label	Execution resumes at this label if Action is true.

Button Example

If we wanted to read the status of a switch off I/O PortB.0, here is a command we will use in the next program.
Button 0, 0,254,0,B1,1,loop

The next program is similar to the previous program, in as much that it performs a binary counting and stops when the button is depressed. This program uses the button command.

The program contains two loops as before, the first loop counts to 15 and the current number's binary equivalent is reflected by the lit LED's connected to port B. The loop continues to count as long as the switch SW1 remains open.

When SW1 is closed the Button command stays inside a smaller internal non-counting loop "lp" where the program remains until the switch SW1 is re-opened. You can switch back and forth between counting and non-counting states.

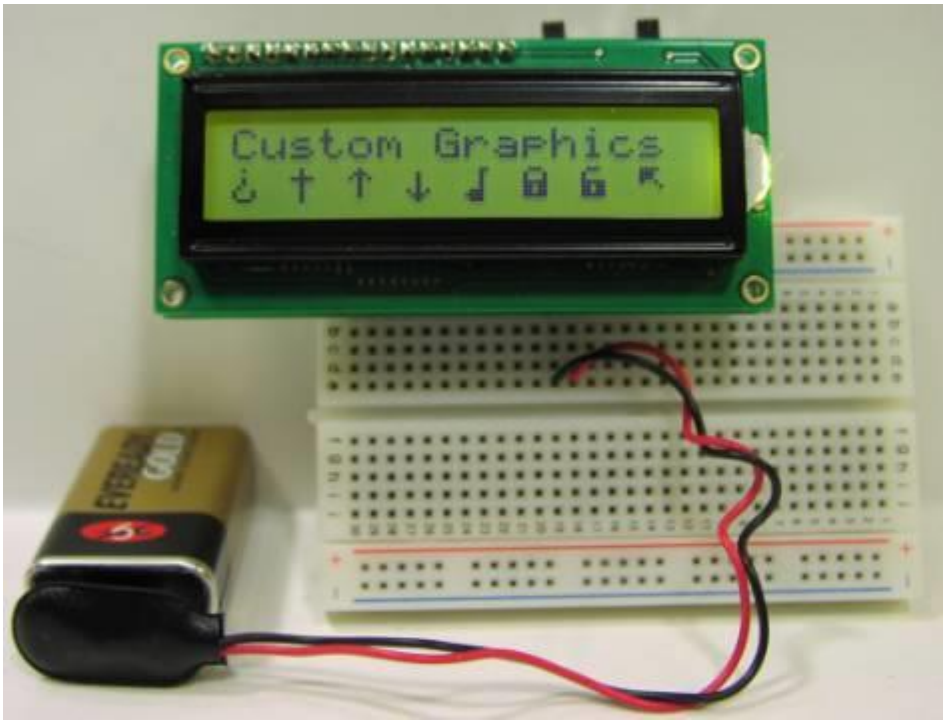
We use the same scehematic as before.

```
osccon = %01110100                  'Set-up internal oscillator at 8 Mhz
define OSC 8                        'define the oscillator frequency
ANSEL = %01100000                   'Set RA0-RA4 as digital I/O for LCD leave RA5 and RA6 A/D
trisb = 0                           ' Set Port B to outputs
input portb.0                       ' Set Port B.0 to input
pause 1000                          'Initalize time for LCD
LCDOUT 254,1, "Binary Counting"     'Print on LCD line 1
j var byte                          'Create variable j
i var byte                          'Create variable i
B1 var byte                         'Create variable B1 for Button Command
start:                              'Label for start of program
For J = 0 to 15                     'Start counting loop
lp:                                 'Start of loop routine to read switch
```

```
Button Portb.0, 1,254,0,B1,1,routine2      'If PB.0 = 1 (high) exit loop

goto lp                                    ' If PB.0 = 0 (low) stay inside loop, stop counting

routine2:                                  'If PB.0 = 1 (high) exit loop here

LCDOUT 254,192, "                "         'Clear printing on LCD line 2

LCDOUT 254,192, "Binary LED # ", #j        'Print on LCD line 2

i = j << 4                                 'shift number 4 places to right

portb = i                                  'display binary number using LEDs on RB4 to RB7

pause 500                                  'Delay to give human time to see LED binary number

next j                                     'next number

goto start                                 'Start from the beginning again
```

When the program is ran, it begins counting. When the switch is closed all the LEDs will turn off and it stops counting. Open the switch and the counting resumes starting from where it stopped.

PIC Experimenter's Board Manual

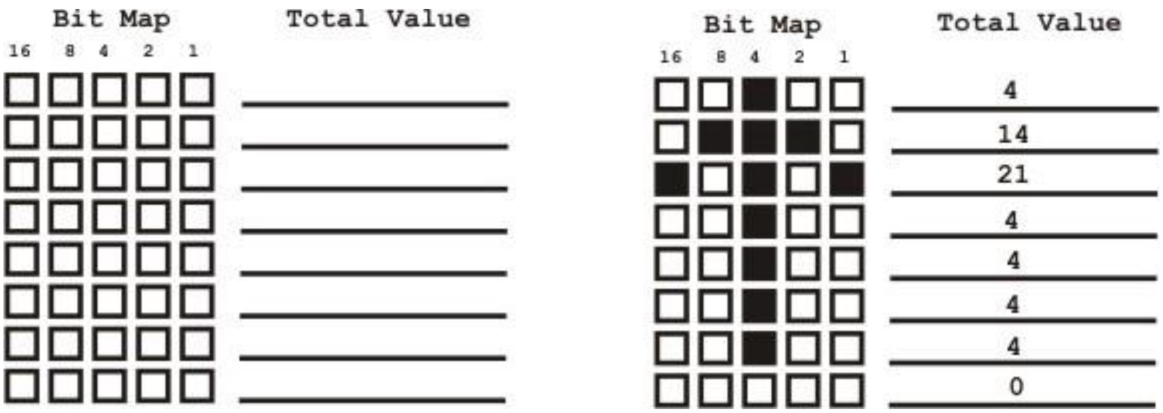


Creating Custom LCD Characters

This is an advanced topic. Feel free to skip this section completely and move on to chapter 4 that is the begining of the Project chapters. If you ever have a need for custom characters you can revist this section.

The LCD has eight unused characters in RAM (Random Access Memory) that the user may define and use. Each character is defined by a 5x8-pixel pattern. The standard character generator of the HD44780 holds the 5x8-pixel patterns for its standard ASCII characters in ROM (Read Only Memory) and cannot be changed.

Each custom character is defined by an 5x8-pixel pattern. The pattern consists of five pixels across each row and eight rows high. Each 5-pixel row is represented by one byte. See the illustration below.



Each box in the bit map has a value listed at the top of its column. When the box is colored in the value is added to all the colored boxes across each row and placed in the total value column to the right. Those total values are the byte values you will use to define your custom characters. Each character requires eight bytes of information to be defined. An up pointing arrow character is shown for example.

A worksheet is provided to help you to define the eight custom characters. You do not need to define all eight characters; you can define just one character if that is all you need.

Writing Custom Characters to CG RAM

Program examples follow this general outline.

1) Set pointer to beginning of CG RAM (memory location 64) on LCD display

LCDOUT 254,64

2) Send bytes representing bit patterns into CG RAM. The LCD controller automatically increments CG RAM addresses, as bytes are transmitted.

3) Leave CG RAM by issuing a control command such as clear LCD screen.

LCDOUT 254,1 'clear screen

4) The custom characters are the first 8 character positions in the LCD CG memory (memory locations 0 to 7). To print custom characters, use the character position. The following commands

prints the first two customer characters.

```
LCDOUT 0  'Print 1st Custom Char.

LCDOUT 1  'Print 2nd Custom Char.
```

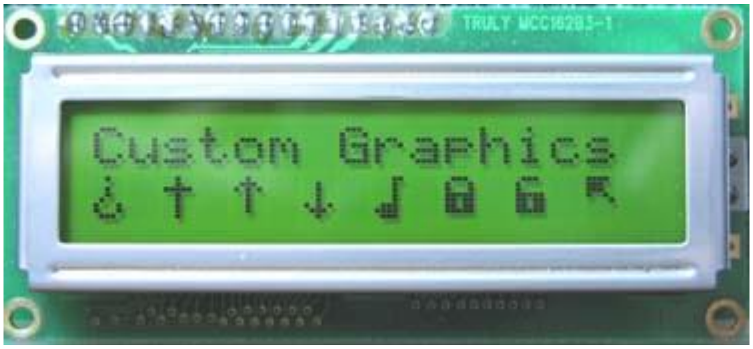
Program Listing:

```
'LCD Custom Characters

#CONFIG
  __config _CONFIG1,_INTRC_IO & _WDT_OFF &  _PWRTE_ON &  _MCLR_OFF & _LVP_OFF &
_CP_OFF
#ENDCONFIG

osccon = %01110100    'Set-up internal oscillator at 8 Mhz
define OSC 8               'define the oscillator frequency
ANSEL = %01100000     'Set RA0-RA4 as digital I/O for LCD leave RA5 and RA6 A/D

pause 1000

cnt VAR BYTE
char VAR BYTE

Data 4,0,4,4,8,17,17,14     'upside down question mark
Data 4,4,31,4,4,4,4,4       'cross
Data 4,14,21,4,4,4,4,0      'up arrow
Data 0,4,4,4,4,21,14,4      'Down arrow
Data 3,2,2,2,2,14,30,14      'music note
Data 14,17,17,31,27,27,31,0 'locked
Data 14,16,16,31,27,27,31,0 'unlocked
Data 30,28,28,18,1,0,0,0    'microsoft icon

lcdout 254,1
pause 1
lcdout 254,64
Pause 1
For cnt= 0 TO 63
Read cnt, char
lcdout char
Pause 1
Next

lcdout 254,1  'clear screen
Pause 4
lcdout 254,1,"Custom Graphics"

'Move to second line
lcdout 254,192 'move to line 2
Pause 4
```

```
For cnt= 0 TO 7
lcdout cnt," " 'Print character then space
Next
end
```

PIC Experimenter's Board Manual

Program Output



Program Function:

The program uses the 16f88's on chip EEPROM to store the custom character's bit maps before uploading them to the LCD's CG memory..

Animation

Basic animations may be created by rapidly displaying successive custom characters in the same LCD screen position.

Animation using external memory

You can print more than eight custom characters using external memory and uploading new character patterns to the LCD. If a custom character is already printed on the LCD display, changing its bit map pattern in CG RAM changes its display on the LCD screen. This technique may be used to created animations as well as creating more than eight custom characters.

This concludes the preliminary introduction to the PIC 16F88 microcontroller and PCB. Each of the following chapters will now be independent projects. Each project has something to teach.

As you go through the upcoming projects in our book if you run into difficulty we will try to help you along. Here is the URL for our 16F88 PIC microcontroller support page.

http://www.imagesco.com/microcontroller/micro-support.php