



# Binary 2 Decimal

## By Neo Avestaran

## Introduction

My aim with this project is to help people learn about binary and to be able to quickly convert binary to decimal through muscle memory. I have tried to make the device and this information as easy to use and understand as well as including all the necessary information needed to learn from my project.

I have opted for the PIC16F84A Microcontroller with an external clock connected to a 16 x 2 Alphanumeric LCD Character Display. The project has been built on a breadboard comprising of discrete components such as a bicolour LED, various resistors, capacitors, several push to make switches, a 4Mhz crystal and an off the shelf inline 3.3/5.0v power supply module.

I decided to use a breadboard because it meant that minimal soldering was required, making it easier to prototype changes. Also, I decided to use links instead of wires because it allows the connections to be easily run under components instead of over them meaning the connections are less exposed and it gives it a much neater appearance.

Something I could improve on would be to try and cut-down on the number of components that are required by for instance using instead the PIC16F627A which has an internal clock.



## A Brief History of Binary

A binary number is any number that can be displayed in a base-2 numerical system. Base-10 (decimal) uses 10 different symbols to represent digits, base-2 uses only 2 symbols to represent digits. Around 1700 Gottfried Leibniz developed a base-2 numerical system that only used 1's and 0's which goes up in powers of 2.

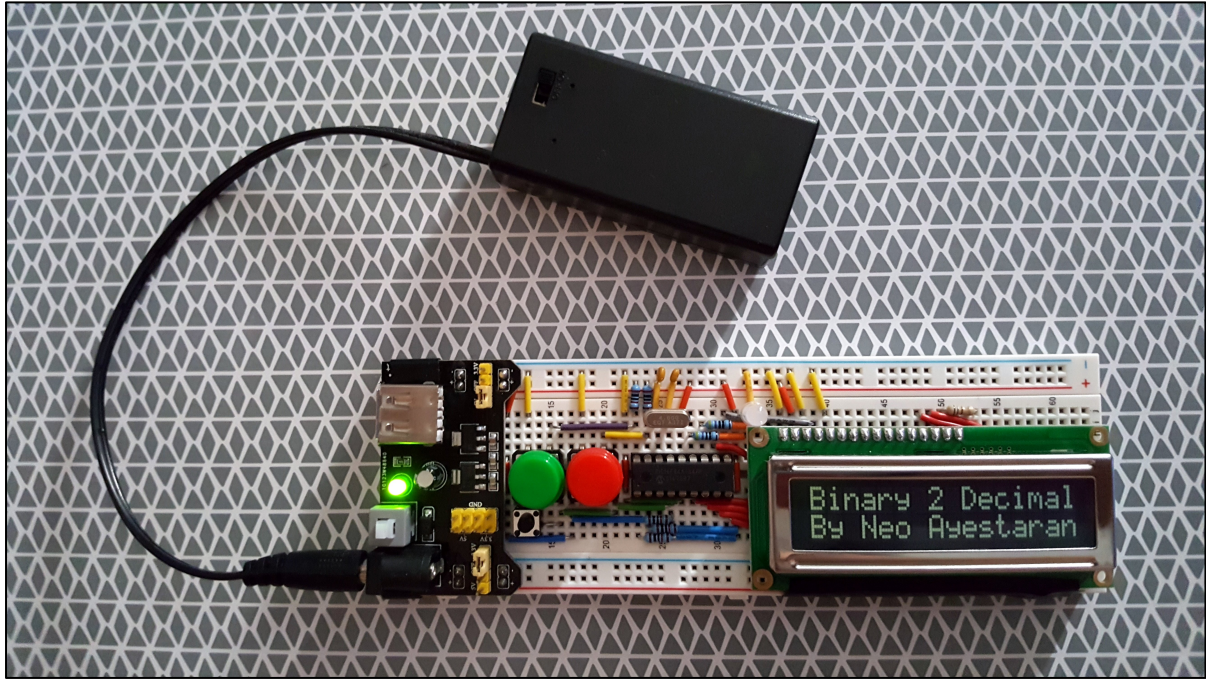
Each digit in a binary sequence is referred to as a 'bit' and doesn't necessarily need to be visually represented with 1's and 0's. Any two symbols can be used by it is preferable to be represented as something that can be easily identified as something similar to an (on or off), (yes or no), (a tick or a cross), (3.3/5v or 0v).

### Examples

000001 =  $2^0 = 1$   
000010 =  $2^1 = 2$   
000100 =  $2^2 = 4$   
001000 =  $2^3 = 8$   
010000 =  $2^4 = 16$   
100000 =  $2^5 = 32$

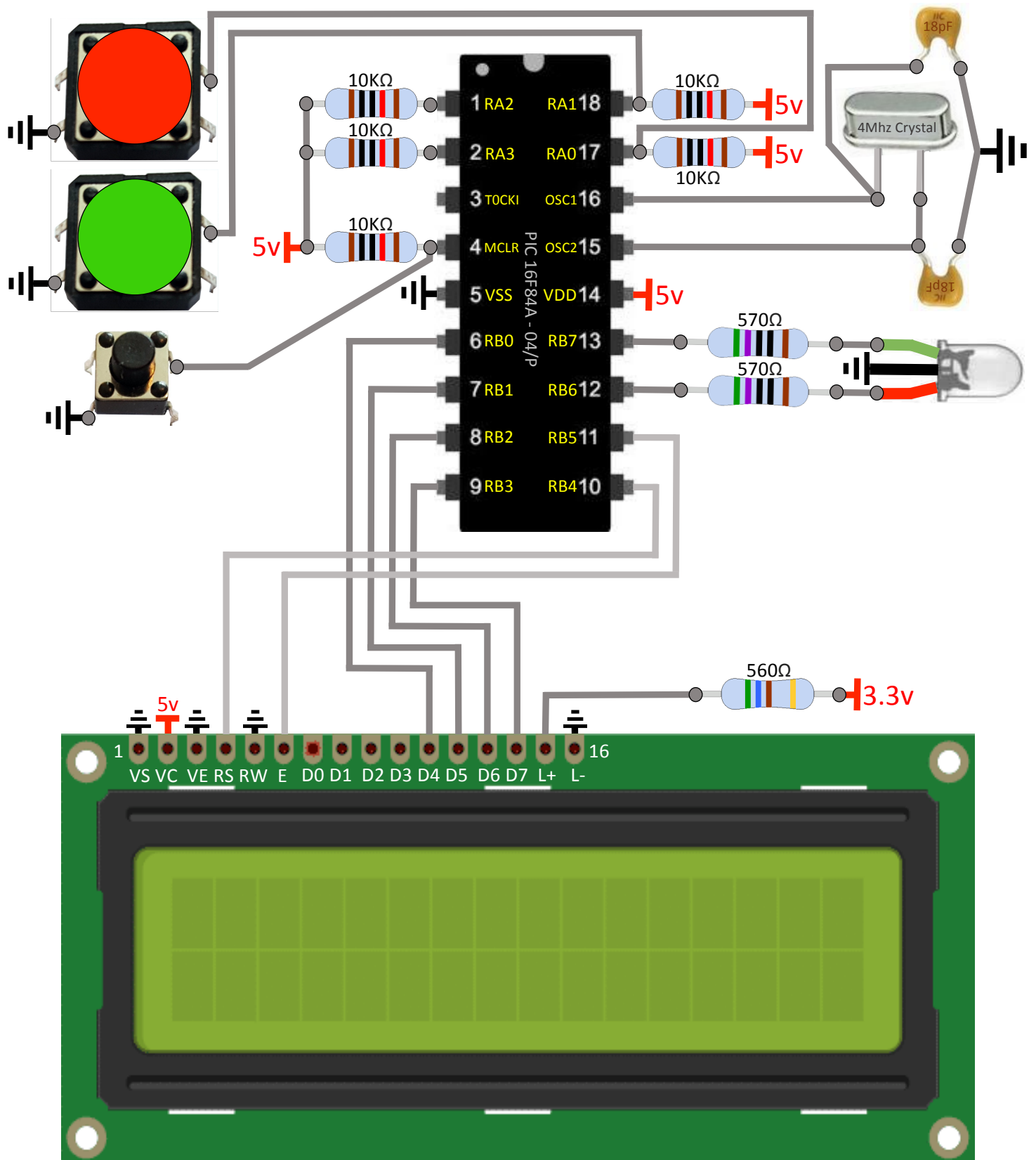
001101 = 13  
111111 = 63  
000000 = 0  
011011 = 27  
110010 = 50  
111001 = 57  
110011 = 51  
011001 = 25

## Parts List

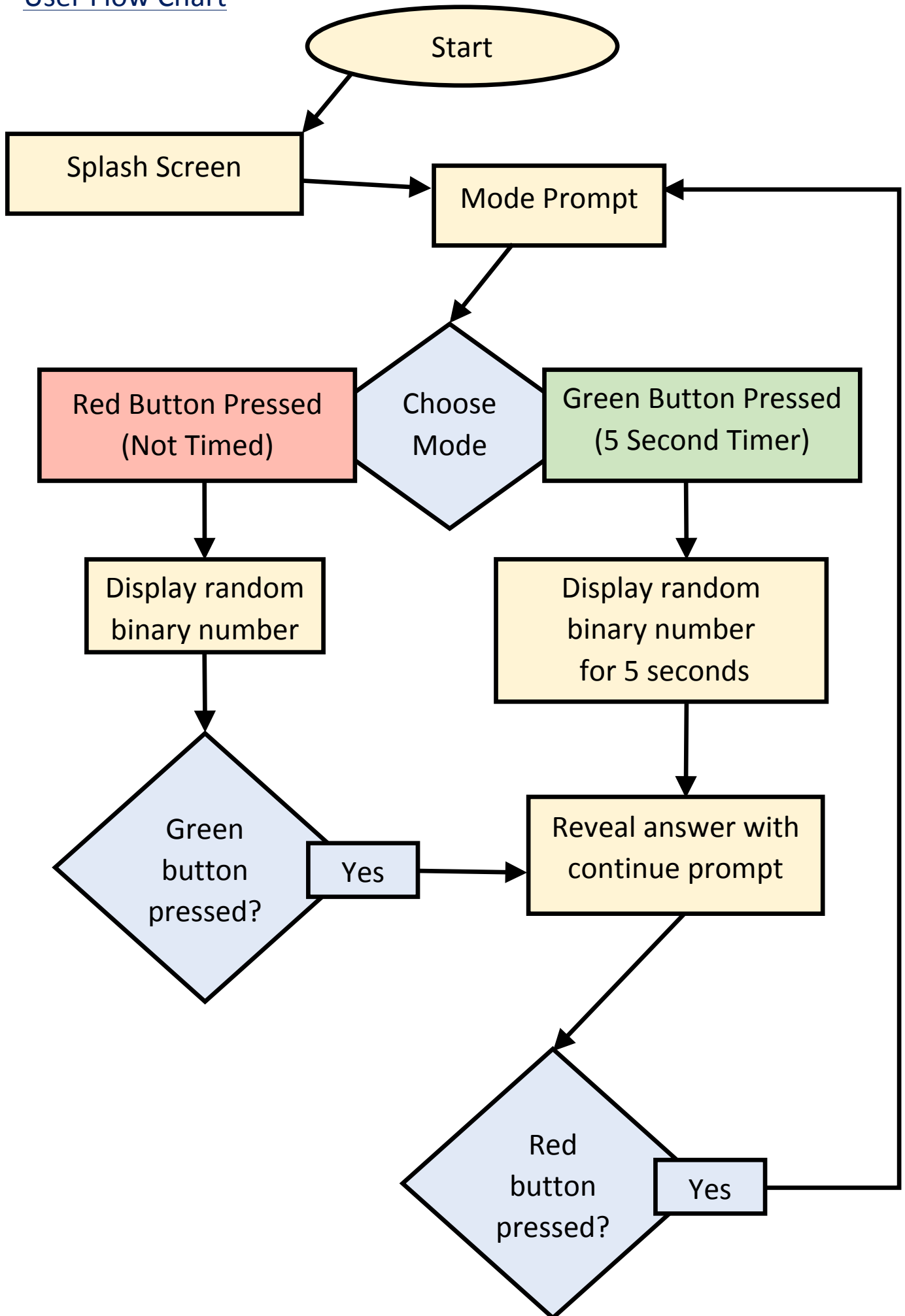


- 1 x 830 Point Breadboard
- 1 x Inline Breadboard Power Supply Module 3.3/5.0v
- 1 x 9V PP3 Enclosure with ON/OFF Switch and 2.1mm DC Plug
- 1 x PIC16F84A-04/P Microcontroller
- 1 x 18 Way, Turned Pin IC Dip Socket
- 1x 162C-3LP 16 x 2 Alphanumeric LCD Character Display
- 3 x Push to make Switches Micro/Red/Green
- 1 x 4Mhz Crystal
- 1 x Bicolour Green/Red LED - 5mm
- 2 x 18pF Multilayer Ceramic Capacitors - 50V dc  $\pm 5\%$
- 5 x 10k ohm resistors - 0.25w 5%
- 2 x 570 ohm resistors - 0.25w 5%
- 1 x 560 ohm resistor - 0.25w 5%
- Assorted coloured insulated links
- OshonSoft PIC Simulator IDE v7.43
- Microchip MPLAB IDE v8.91
- MiniPro 6.50

## Circuit Diagram



## User Flow Chart




## Instructions

**Step 1:** Turn the unit on by pressing the grey button, a splash screen will appear for 5 seconds.




Binary 2 Decimal  
By Neo Avestaran

**Step 2:** Select the desired mode, press the green button for a 5 second timer or the red button for no timer.



5sec Timed Mode?  
Green=Yes/Red=No

Convert 11101001  
Green for answer



Convert 01011111  
Answer in 5

**Step 3:**

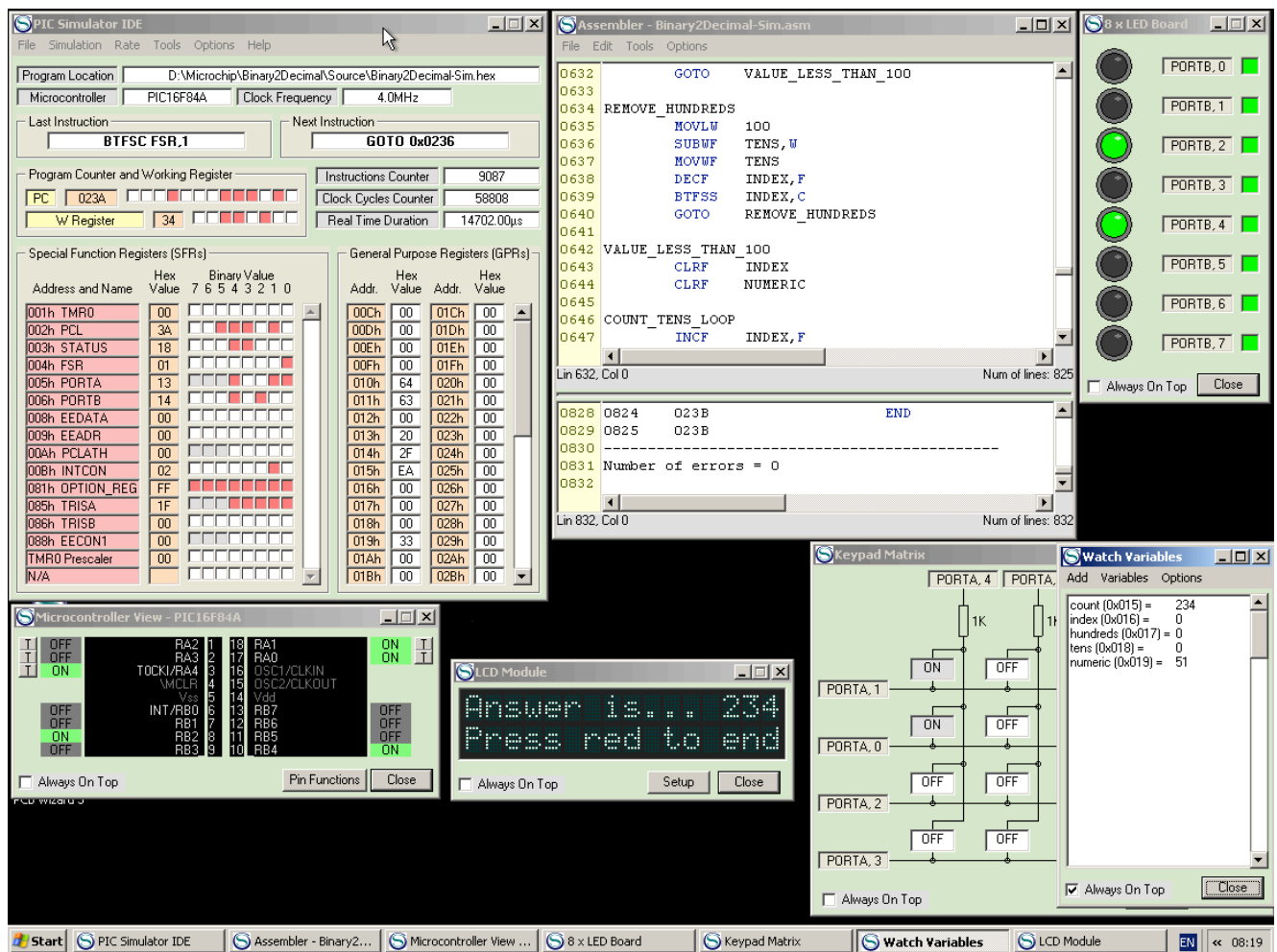
(**Green**): A randomly generated binary number will appear and a 5 second countdown will start, once the countdown ends the answer will be revealed. (**Red**): A randomly generated binary number will appear along with a prompt to press the green button when you are ready to reveal the answer.

**Step 4:** (both): The answer will appear as well as a prompt to press the red button to return to the mode selection screen.



Answer is 233  
Press red to end

## Simulator, Compiler & Flash Programming Software



In order to reduce the amount of testing time required when developing code for the microcontroller, I decided to enlist the help from an off the shelf PIC simulator. The IDE allows one to continually develop a prototype without the interruption of having to keep flashing the code to a physical test bench. The software environment comprises of multiple windows that mimic hardware modules. I used MPLAB IDE in order to compile the existing assembly source code into hex, which is required to program the microcontroller. In order to flash the PIC16F84A I used the MiniPro TL866CS Universal Programmer with accompanying software.



## Troubleshooting

One of the problems I faced was how to generate a non pseudo-random number in assembly code in order for the game to not present to the user with the same repetitive binary value. The solution was to use a hybrid approach, by providing randomness harvested from a natural source. This was achieved by having a value continuously loop from 0 to 255 in the background until the user selected the required game mode. Once a mode had been chosen the loop would be interrupted and the current value would be displayed in binary. When the user presses the red button to go back to the mode select screen, the next binary value to display will re-seed and the incrementing cycle will start over again.

A good way to visualise this is to imagine a wheel with 256 sections numbered 0 to 255 with an arrow at the top. The wheel will keep spinning until a button is pressed, at this point it will stop on whichever number the pointer is on. When another button is pressed the wheel will begin spinning again.

The idea is that because the value increments so quickly even if the user could see the value changing they would not be able to reliably choose the same number in a row multiple times because of the shear speed of the cycling process.