Decimal to Binary Conversion:

We commonly work in decimal, but the computer uses binary to represent and store a number. In binary a number is represented as a sequence of bits (BInary digiTS) where each bit represents a power of two. As a matter of reference, we first consider how a decimal number actually works.

$$253_{10} = 2 \times 10^2 + 5 \times 10^1 + 3 \times 10^0$$

In a similar fashion, a binary number uses each bit times a power of two. So if we were to convert a binary number to decimal, we can simply write out the various bits times the various powers of two and add them up. This looks something like this

Binary	As powers of 2	As decimal.
11111101_2 =	1×2^7	1 x 128
	$+ 1 \times 2^6$	+ 1 x 64
	$+ 1 \times 2^5$	+ 1 x 32
	$+ 1 \times 2^4$	+ 1 x 16
	$+ 1 \times 2^3$	+ 1 x 8
	$+ 1 \times 2^2$	+ 1 x 4
	$+ 0 \times 2^1$	+ 0 x 2
	$+ 1 \times 2^0$	$\underline{+ 1 \times 1}$
		= 253

This is the process for converting from binary to decimal. However we will also need to convert a decimal number into binary. This operation is a little more involved, but if we simply write out the binary in a generic format we will see a process that works.

$$155_{10} = b_n \times 2^n + b_{n-1} \times 2^{n-1} + ... + b_0 \times 2^0$$

So if we were to divide both sides be 2, we have

$$77.5_{10} = b_n 2^{n-1} + b_{n-1} 2^{n-2} + ... + b_1 2^0 + b_0 2^{-1}$$
$$b_n 2^{n-1} + b_{n-1} 2^{n-2} + ... + b_1 \times 1 + b_0 \times 0.5$$

Now the lowest bit b_0 is multiplied by 2^{-1} or 0.5 and thus shows up as the fractional part 0.5. So we can now say that b_0 is 1. We continue the process by taking the integer part from after the division result and repeating the process. It commonly gets written out as follows.

```
155/2  = 77.5  =>  1
77/2   = 38.5  =>  1
38/2   = 19.0  =>  0
19/2   =  9.5  =>  1
9/2    =  4.5  =>  1
4/2    =  2.0  =>  0
2/2    =  1.0  =>  0
1/2    =  0.5  =>  1
0/2    =  0.0  =>  0
```

Once the number being divided by 2 reaches zero, we are done. So we have $155_{10} = 0\ 1001\ 1011_2$

Note that the binary number has been segmented out in groups of four bits, this was done simply for readability.

Another approach to this is more heuristic and a little more of a short hand, and that is to simply start counting up by power of two's until you have a number greater than the decimal number you are converting. Then start with a zero in the first bit, then move back by one power of two and if this power of two is less than the remainder, set the bit to one, and reduce the remainder by this power of two. If the power of two is greater than the remainder, set the bit to zero and do not change the remainder. This is repeated until the power of two is reduced to 1, when the bit equals the remainder.

Consider the decimal number 55_{10}

```
1 < 55           // Since 55 is greater than 1, move to next power of 2
2 < 55           // Since 55 is greater than 2, move to next power of 2
4 < 55           // Since 55 is greater than 4, move to next power of 2
8 < 55           // Since 55 is greater than 8, move to next power of 2
16 < 55          // Since 55 is greater than 16, move to next power of 2
32 < 55          // Since 55 is greater than 32, move to next power of 2
64 > 55     => 0  // Since 64 is greater than 55, set "top" bit to zero, and move back to 32
55-32 = 23  => 1  // Subtract 32 from 55, if result is positive, set bit to one, else it is zero
23 -16 = 7  => 1  // Subtract 16 from remainder from previous line,
                  // and if result is positive, set bit to one, else it is zero
7 - 8 = -1  => 0  // Subtract 8 from remainder from previous line, and if result is positive,
                  // set bit to one, else it is zero and remainder is ignored.
7 - 4 = 3   => 1  // Continues to end.
3 - 2 = 1   => 1
1 - 1 = 0   => 1  // Once power of two is 1 we are done.
```

Thus in the end $55_{10} = 0110111_2 = 32 + 16 + 4 + 2 + 1$

There is another radix that will be useful to us and that is known as Hexadecimal. This is a base 16 numbering system and a number is of the form

$$h_3h_2h_1h_0 = h_3 * 16^3 + h_2 * 16^2 + h_1 * 16^1 + h_0 * 16^0$$

Now each hex digit can have a range of 0 to 15 (one less than the radix). Thus they will need to extra symbols, which is accomplished by adding in the letters A, B, C, D, E and F. In this case 0, through nine are the same as before and A = 10, B = 11, C = 12, D = 13, E = 14 and F = 15.

Now I am sure your first question is Why? Well the real use of hexadecimal is in the display of binary numbers, since each hex digit will match up with a group of four bits in the binary number. As we can see in the next line, if we group the binary bits in groups of four.

$$155_{10} = b_8 \times 2^8 + b_7 \times 2^7 + b_6 \times 2^6 + b_5 \times 2^5 + b_4 \times 2^4 + b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

$$\begin{aligned}155_{10} = {} & (b_8 \times 2^0) \times 2^8 \\ & + (b_7 \times 2^3 + b_6 \times 2^2 + b_5 \times 2^1 + b_4 \times 2^0) \times 2^4 \\ & + (b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0) \times 2^0\end{aligned}$$

$$\begin{aligned} 155_{10} &= (\, b_8 \text{ x } 2^0 \,) \text{ x } 16^2 \\ &\quad + (\, b_7 \text{ x } 2^3 + b_6 \text{ x } 2^2 + b_5 \text{ x } 2^1 + b_4 \text{ x } 2^0 \,) \text{ x } 16^1 \\ &\quad + (\, b_3 \text{ x } 2^3 + b_2 \text{ x } 2^2 + b_1 \text{ x } 2^1 + b_0 \text{ x } 2^0 \,) \text{ x } 16^0 \end{aligned}$$

$$\begin{aligned} 155_{10} &= 0\ 1001\ 1011_2 \\ &= 0 \quad 9 \quad\ 11 \\ &= 0 \quad 9 \quad\ \ B_{16} \end{aligned}$$

Arbitrary Numbering System (American Standard Coding for Information Interchange or ASCII)

In order to represent letters and other characters on a computer a standard coding was established called ASCII. A complete table of the character to letter relationship can be found on the web at http://www.asciitable.com/

The standard was established to allow for simple relationships between certain characters. Examples of this can be seen by comparing binary codes for the upper and lower case letters as shown below.

'A' => 0x41 or 0100 0001_2 'a' => 0x61 or 0110 0001_2 Note that the difference between
'B' => 0x42 or 0100 0010_2 'b' => 0x62 or 0110 0010_2 upper and lower case is a single bit.
...
'Z' => 0x5A or 0101 1010_2 'z' => 0x7A or 0111 1010_2

Also of note are the decimal digits

'0' => 0x30 or 0011 0000_2 Thus the last hex digit or last four bits is the
'1' => 0x31 or 0011 0001_2 same as the digits value.
...
'9' => 0x39

Adding Binary Numbers:

Adding numbers together is an important operation, and necessary to make a computer functional. So lets consider how an addition would progress in binary. Just like a decimal addition, binary addition progress from the lowest bit to the upper bit. Thus the n'th bit addition will have a carry in (cin) coming from the bit below, and the bits from each number (a_n b_n). The results are an output bit (d_n) and a carry out (cout) that will go to the next bit. This is shown in the following

$$\begin{array}{r} a_n^{\ \ \text{cin}} \\ \underline{+\ b_n} \\ {}^{\text{cout}}\ d_n \end{array}$$

Since each bit is binary, we can write out all the possible combinations for cin, a_n, and b_n, as shown here.

$$\begin{array}{cccccccc} 0^{0} & 0^{0} & 1^{0} & 1^{0} & 0^{1} & 0^{1} & 1^{1} & 1^{1} \\ \underline{+0} & \underline{+1} & \underline{+0} & \underline{+1} & \underline{+0} & \underline{+1} & \underline{+0} & \underline{+1} \\ {}^{0}\,0 & {}^{0}\,1 & {}^{0}\,1 & {}^{1}\,0 & {}^{0}\,1 & {}^{1}\,0 & {}^{1}\,0 & {}^{1}\,1 \end{array}$$

So we could add two number together as

$$\begin{array}{rcr} \text{binary} & \qquad & \text{decimal} \\ 0^1\;\;1^1\;\;1^1\;\;1^0 & & 7 \\ \underline{+0\;\;\;1\;\;\;0\;\;\;1} & & \underline{+\;5} \\ {}^{\underline{0}}\,1\;\;\;1\;\;\;0\;\;\;0 & & 12 \end{array}$$

A similar process can be done for a subtraction. A natural form for negative numbers comes about by doing a subtraction that results in a negative number. We start with an example of a decimal number. Consider 33 - 51

$$\begin{array}{r} {}^{1}0\;\;{}^{1}0\;\;{}^{1}3\;\;{}^{0}3 \\ \underline{-0\;\;\;0\;\;\;5\;\;\;1} \\ 9\;\;\;9\;\;\;8\;\;\;2 \end{array}$$

So in the decimal case, an infinite string of 9's is a 9's complement negative representation. Now in a binary case, we have the following

$$\begin{array}{rcr} \text{binary} & \qquad & \text{decimal} \\ {}^{1}0010\;0011 & & 35 \\ \underline{-\;0011\;0110} & & \underline{-\;54} \\ 1101\;1101 & & -19 \end{array}$$

So in a binary case, an infinite string of ones can represent a negative number. Now since a computer has a finite number of bits for any number, it is stated that the top bit in any number is the sign bit, and if it is one it is a negative number. An example of how this can work in demonstrated here with a simple eight bit example.

$$\begin{array}{rcr} \text{binary} & \qquad & \text{decimal} \\ 0000\;1100^{\,0} & & 12 \\ \underline{+\;1111\;1100} & & \underline{-4} \\ {}^{1}\;0000\;1000 & & 8 \end{array}$$

A number can be changed from a positive to negative number by doing what is known as the two's complement operation. In this operation, we first invert every bit in the number and then add 1 to the results. Example

$$\begin{array}{rcr} 0000\;0100 & \qquad & 4 \\ 1111\;1011 & & \text{inverted} \\ \underline{+\;0000\;0001} & & \text{adding 1} \\ 1111\;1100 & & \text{-4 as above} \end{array}$$

So what does this mean when we are programming our Arduino in the lab. A common number type is the integer, declared as int, which is a 16 bit binary number. Being a 16 bit signed number, the range of values for an int is -2^{16} to $2^{16}-1$ (-32768 to 32767). It should be noted that there is an unsigned integer (unsigned int), which ignores the sign bit and thus has a range of 0 to $2^{32}-1$ (0 to 65535).

A computer example is posted on the course website.

Examples of Video that can help

https://www.youtube.com/watch?v=9W67I2zzAfo

https://www.youtube.com/watch?v=Hof95YlLQk0

https://www.youtube.com/watch?v=ZLA0Ahymiv8