Understanding Data

by Martin Pace

Understanding the Binary Number System (Base 2)

The Binary Number System is a *base 2 number system* consisting of 0's and 1's. For computer systems, 0 represents *no voltage*, whereas 1 represents *voltage*. Often, 0 may also represent *false*, while 1 may represent *true*.

0 0 0 0 0 0 0 0

Shown above is an 8-digit sequence of binary numbers. In reference to data, each binary number is commonly referred to as a **bit**. Eight bits is equal to a single byte. A **byte** is the single most common base unit of data. As such, all data is inherently a collection of bytes. When collections of bytes grow in size, there are other denominations of bytes such as kilobyte, megabyte, gigabyte, and terabyte. These aren't all too important as they are simply alternative conversions and denominations to simplify the display of the size of information stored on a computer, or to simplify things when we are talking about amounts of data. Binary @ Wikipedia.org.

Understanding the Decimal Number System (Base 10)

As mentioned above, the binary number system is a *base 2 number system*. More familiar to us humans is the *base 10 number system* (or *decimal number system*) which we learn from our earliest years as children. For a convenient reminder, review the following example of the base 10 number system. The system begins at 0, increments to 9, then starts over at 0, but the next 0 value is actually 10, which increments to 19 before starting over at 20. Because the *base 10 number system* can reach extraordinarily high values and display them as such, a set of 10 is incremented in the next integer slot to the left, which is why 0 increments to 10, then to 20, to 30, etc. Decimal @ Wikipedia.org.

0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
…

From Binary to Decimal

Converting from the base 2 number system to the base 10 number system (from Binary to Decimal) is relatively simple, although it may appear difficult at first glance. When performing such a conversion, we are simply calculating an integer value from a binary number.

To do so, let's first look at binary values from a different perspective. Each bit (or single binary value) placeholder is an exponent that a base can be multiplied by.

$$base^{exponent}$$

Of course as you may already know, the process of using an exponent is generally called "raising (base) to a power of (exponent)."

In a conversion from *binary* to *decimal*, the base will always be 2, and the exponent will be the digit placeholder value (incrementing from right to left as shown below).

Binary Values	0	0	0	0	0	0	0	0
Placeholder Values (n)	7	6	5	4	3	2	1	0

Formula

$$Decimal = 2^0 + 2^1 + \ldots + 2^n$$

Example

Binary Number: 00000101

Refer to the above placeholder table. The formula is only applicable to bits that are equal to 1 (0-value bits are ignored during calculation). As can be seen, the above binary number has 1-value bits in the 0 placeholder and the 2 placeholder. Let's construct the equation:

$$Decimal = 2^0 + 2^2$$

Reminder // Any non-zero value raised to the power of 0, will always evaluate to 1.

$$Decimal = 1 + 4$$
$$Decimal = 5$$

Therefore,

00000101 (*Binary*) = 5 (*Decimal*)

You could trim unnecessary preceding zeros from a binary representation as well,

0101 (*Binary*) = 5 (*Decimal*)
101 (*Binary*) = 5 (*Decimal*)

Looking at the conversion from yet another perspective may be easier – it boils down to personal preference.

We could go ahead and apply the above formula to the placeholder values:

Binary Values	0	0	0	0	0	0	0	0
Placeholder Values (n)	7	6	5	4	3	2	1	0
Post-Conversion Values	128	64	32	16	8	4	2	1

Now it's just a matter of adding up the post-conversion values for each 1-value bit. Let's revisit our example binary number from above.

Example

Binary Number: 00000101

Refer to the above post-conversion table. We already know that placeholders 0 and 2 have 1-value bits. Looking at the post-conversion table, the post-conversions for those placeholders are 1 and 4, respectively. Therefore, it's simply a matter of adding those two post-conversion values.

$$Decimal = 1 + 4$$
$$Decimal = 5$$

Therefore,

00000101 (*Binary*) = 5 (*Decimal*)

Now you shouldn't have any problem understanding the T-Shirts that say "There are 10 types of people in this world, those who understand binary and those who don't."

Understanding the Hexadecimal Number System (Base 16)

Due to the fact that binary numbers expand so rapidly in size as the number increases, readability becomes increasingly difficult. Therefore, using the *hexadecimal number system* which has a *base of 16* makes life a little easier when working with data. Thus, it is primarily used as a human friendly representation of binary data. Hexadecimal @ Wikipedia.org.

Values in the Hexadecimal Number System															
Hex Values	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal Values	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Due to its convenience, hexadecimal byte data is what you will find displaying data packets and file data.

Hex -> Binary, Binary -> Decimal, Decimal -> Hex

Each digit of a hexadecimal value (hex value for short) is equivalent to half of a byte. So let's look at a byte:

2E

The above hex value contains two digits: 2 and E. Each digit is half of a byte, and since a byte is the equivalent of eight bits, each digit of a byte would therefore be equal to four bits.

2E
2 E
0010 1110

Taking those binary values, let's convert them to decimal:

0010 1110
2 14

Now that we have the decimal values of the binary numbers, the hexadecimal conversion is the next step. Simply swap decimal values 2 and 14 for their respective hex values, shown in the table above. 2 is going to be 2, and 14 is going to be E. Simply place 2 next to E, for 2E, and you've successfully converted a byte to bits and back to a byte, and also a hex value to binary, then to decimal, then back to hex.

2 E
0x2E
(Common Notation of Hex Values)

Of course, given the hex table above, converting to binary was unnecessary for this example. However, now you should understand the conversions between hex, binary, and decimal.

Note

If using Windows, you may use the Windows Calculator to perform these conversions so long as you set it to its scientific mode. Additionally, there are calculators on the internet and downloadable conversion tools such as my simple ByteMe application written in C# using the .NET Framework.

ASCII/Dec/Hex/Oct/Bin Calculator at CodeCall.net
ByteMe at martinpace.com

Some conversions are beyond the scope of this document. Simply search the internet for floating point and/or negative number conversions across these bases.

What about the Octal Number System (Base 8) and others?

The *octal number system* is beyond the scope of this document, as are other *base x number systems* such as *base 32* and *base 64*. These auxiliary number systems are indeed important to data. However, octal numbers are often used in special case scenarios and I don't feel they are pertinent to this document. Please use Wikipedia.org to read about the octal number system and others.