Understanding standard numbering systems, the binary system and others.

Intended Audience: Individuals that have little to no experience with computers, the binary numbering system and other numbering systems besides base 10.

Base 10 numbering system

The easiest way to understand various numbering systems, if you are not familiar with them, is to realize you already are familiar with the base 10 numbering system. All information that is conveyed between one individual and another is conveyed using a standard set of symbols. We will be speaking about written communication in this context. For example the system being used right now implies the use of 26 letters and various punctuation symbols. We are all familiar with it as it is written English.

The numeric system can also be used to convey not only magnitudes, amounts, etc. but also information. The base 10 numbering system we are all so familiar involves 10 symbols:

{0,1,2,3,4,5,6,7,8,9}

With these symbols and the positions of the symbols within a given number, we gain meaning. For example if I state I have 285 Gallons of water. You know exactly what I mean because you know the base 10 numbering system. But what does 285 really mean? In the numbering systems we will be describing the position of the symbol determines its magnitude, the further to the left the symbol is the greater its magnitude. For example in the above 285, the 8 has 10 times more magnitude that the 5, as it is 8x10 or 80. Let's now express this formally:

2 x 10^2 + 8 x 10^1 + 5 x 10^0 = 285

In this form the '^' symbol simply mean to take the number to the left of the symbol and multiply itself by the number of times on the right of the symbol, thus 10^2 is nothing more than 10x10 or 100.

Thus the first term is simply 2 x 10^2 or 2 x 10 x 10 or 200.

The second term is simply 8 x 10^1 or 8 x 10 = 80

The last term is 5 x 10^0 or 5 x 1 = 5

Then we simply sum these three terms 200+80+5 to get the number 285.

If you were paying close attention there something probably caught your eye. 10^2 makes sense: Multiply 10 by itself two times, and 10^1 is multiply 10 by itself 1 time, the result is simple 10. But what about:

10^0, I stated that it is 1. To many people this makes no sense, how can you multiply a number by itself zero times and end up with one?

I know that might not be very intuitive or even seem logical. But let me prove that 10^0 actually is one via a simple proof using what we have learned:

Notice: 10^2 is 10 times greater than 10^1. But generalize it further:

10^n is ten times greater and 10^(n-1). But notice in positional notation that is what we really have, every time we move one digit to the right we divide by 10:

10^n/10 = 10^(n-1).

Now for the proof that 10^0 must be one, let's look at a much larger number, except we will make all the digits 1 this will allow us to drop out the 1's because anything multiplied by one is simply itself so take the number:

11,111

Or Eleven Thousand, One hundred and eleven, this is the same as:

10^4 + 10^3 + 10^2 + 10^1 + 10^0

So 10 multiplied by itself 4 times is 10,000 and from what we have already learned the next digit to the right will be 10 times smaller or 1,000 (10^3), divide by 10 again you get the next digit 100, divide by 10 again you get the next digit 10, but what happens next? From 10^1 to 10^0. You just do the same thing, divide by 10, but 10 divided by 10 is 1! Thus by inference I have just proven to you the crazy notion that if you multiply 10 by itself zero times you get 1!

Any numbering system

In fact it just so happens that if you multiply ANY number by itself zero times you will get one. Let's consider an x based numbering system where x represents any positive integer greater than 1, the following table shows the positional notation for such a system:

x^4	x^3	x^2	x^1	x^0

Here we have replaced the ^ for taking x to a power with the more common notation of using a superscript that represents the number of times x is to be multiplied by itself.

Generalizing from the base 10 numbering system, a number in this system is simply the sum of the digits as before, as long as we normalize and use 1 for each digit:

$x^4+x^3+x^2+x^1+x^0$

Just like before, x^2 is just x times x, and x^1 is just x or x times less than x^2. And the formula that applied to base 10 applies here to:

$x^n/x=x^{(n-1)}$

Applying the formula to x^1/x gives us x^0, but x^1 is simply x so this simplifies to x/x which is thus one. We have just proved via inference that any number x, taken to the power of 0 is one.

Base 10 Numbering System with Fractions

So far we have only considered whole numbers, but we can expand the system to handle numbers with fractions. An example is 12.52 pounds of gold, which I am sure you would like to own. Intuitively we know this implies just a little more than about 12 and ½ pounds of gold. Here the dot (.) separates the whole number portion from the fractional portion, other counties besides the US also tend to use a comma (,) as the separator. All that matters is that a symbol everyone agrees upon is used to separate the whole number part from the fractional part. We can formalize the system as follows:

10^2	10^1	10^0	10^{-1}	10^{-2}

So what does 10^{-1} mean? Just use the formula from before:

10^n/10 = 10^(n-1)

Here the digit to the left of 10^{-1} is 10 time larger than it or:

10^0/10=10^-1

But 10^0 is one, as we have already proven and thus 10^-1 is 1/10 or simply one tenth.

The same can be applied to 10^{-2} it is simply (1/10)/10 or 1/100 or one hundredth

And thus the number 12.52 pounds can be expressed as:

$1x10^1 + 2x10^0 + 5x10^{-1} + 2x10^{-2} = 12.52$

Any Numbering System with Fractions

Any numbering system with fractions can be formalized the same way as base 10:

x^2	x^1	x^0	x^{-1}	x^{-2}

So what does x^{-1} mean? We apply the formula as before:

$x^n/x=x^{(n-1)}$

Or x^0/x because x^0 is one it implies $x^{-1}=1/x$

And x^{-2} would be x^{-1}/x or $(1/x)/x=1/(x*x)$ or $1/x^2$

And so on for other fractional parts.

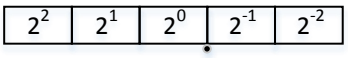
The Binary Numbering System

The above generalization to any numbering system is a bit abstract. Let's apply it to a common numbering system that is used in computers, micro-controllers and digital circuits. Computers, micro-controllers and other digital circuits are circuits that have an on/off nature. That is the sub components of these circuits can take on any of two values 'on' or 'off'. Sort of like a light switch. The devices especially computers, contain millions of these on/off circuits that work together to express numbers and or symbols to a user. We will not go into the details of why the binary numbering system and digital systems in general use on/off sub-components, there is good reasons. An entire article could be devoted to just that subject.

Because there is only two states, the Binary numbering system base is 2 and contains only two symbols:

{1, 0}

Formalized it looks as follows:



Let's take an example, remember we only have two symbols to work with:

101.1_2

Notice the subscript at the bottom of the number, this is typical when we want the reader to understand that the number being represented is in a different base other than 10. Using the rules we already know we can 'convert' this number to base 10. The only reason I state we can convert it to base 10 is simply because we are used to base 10 numbers. For someone who works in the binary numbering system all the time, they may feel comfortable with this number being expressed this way and even have an implicit idea just by looking at it how large of a number it is.

Applying the conversion rules we get:

$1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} = 1x2x2 + 0x2 + 1x1 + 1*(1/2) = 4 + 0 + 1 + 0.5$ or:

5.5

In the base 10 numbering system or: $101.1_2=5.5_{10}$

Groupings within Numbering Systems

When expressing very large numbers, it is convenient to place them in groups, typically with a separator symbol to make them easier to read, for example:

1985286

Represents 1 Million 985 Thousand, two hundred and eighty six. If you are like most people you probably in your mind placed this number into groups of three and mentally placed a comma between each group which is what we are more used to seeing and makes it easier to read:

1,985,286

This grouping idea is common among other numbering systems as well, the groups above represent 1000's. Let's look at the common groupings for computers and digital systems.

For base 2 the smallest common group is called a bit, it is a single digit in any number, it can be either 0 or 1. That is each digit in a binary number is called a bit, thus the following number has 3 bits:

110_2

The next less commonly used (but still used off and on in the Electronics industry is the nibble) a nibble is four bits, the following number contains two nibbles:

10101100_2

The range of a nibble is 0000 to 1111 if you do the math that is 0-15. That is a nibble has 16 possible values.

One of the most commonly used groupings for digital systems is the byte. This is common to the computer industry. A computer usually works with bytes of information. At least the low end micro-controllers in the industry do. These devices are referred to as 8 bit micro-controllers. (A micro-controller is like a low end not very powerful computer, but I don't want to go off on a tangent on that). Here is a definition of a micro-controller from the internet:

"A **microcontroller** is a computer present in a single integrated circuit which is dedicated to perform one task and execute one specific application. It contains memory, programmable input/output peripherals as well a processor."

An 8 Bit micro-controller operates on bytes of information. A byte being 8 bits and has the range:

0000_0000->1111_1111

Note the underscore notation, this makes it easier to read, it is common to group nibbles with underscores in computer notation. If you do the math you will get 0 to 255 or 256 possible combinations.

We keep referring to things with x numbers of bits in them in binary, there is a simple formula you can use to find out how many combinations exist for any given number of bits:

Number of combinations = 2^n where n is the number of bits, thus 2^9 is the number of combinations you can have if you have nine bits, it happens to be 512 combinations. Note that zero is included in the number of combinations, thus the largest number you can represent with n bits is one less than that or 2^n-1. Thus 511 would be the largest number you could represent with 9 bits.

The next division for computers and micro-controllers is called a word. A word is 2 bytes or 16 bits.

These are mid-level micro-controllers, more powerful than the 8 bit ones, but not as powerful as the one's we typically use for laptops and PC's. A word has 2^{16} possible combinations or 65,536 combinations.

The most common PC's used in the 1990's up until around 2000 was 32 bit PC's. A 32 bit number is referred to as a dword and has a range of 2^{32} or 4,294,967,296 combinations. That is a lot of combinations and can represent some rather large numbers.

The most modern PC's on the market now (as of 2017) are almost all 64 bit. A 64 bit number is typically called a long or a ddword and has a range of 2^{64} or 18,446,744,073,709,551,616 combinations! This is a number so large it is hard to describe it, the approx. population of the earth is around 7 Billion people this number is 2,635,249,153 or over 2 Billion times larger than the population of the earth as of 2017.

The Hexadecimal Numbering System

With computers being so advanced, dealing with binary numbers in the raw can get unwieldy to write out as it requires 64 bits to display it, thus the computer industry, and the programmers that program these machines decided to use a similar numbering system that easily translates to the binary numbering system. That is the hexadecimal system. In the hexadecimal system there is 16 digits, because 0-9 is only 10, we needed 6 more, the choice was the first 6 letters of the alphabet, and thus the hexadecimal system uses the following in order of magnitude (left to right):

{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

With 0 representing zero and F representing 15.

Here is a hexadecimal number and its conversion to base 10 as an example:

$2FA_{16}=762_{10}$

Commonly a hexadecimal number in programming terms will be prefixed with 0x to indicate it is a hexadecimal number instead of the subscript thus the above could have been written:

0x2FA=762

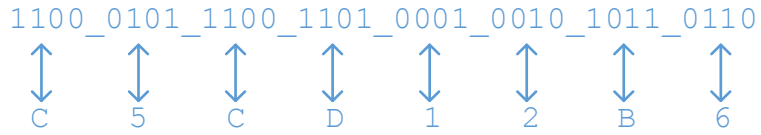
Notice I left off the base on the right hand number, when that is done it is understood to be base 10.

Hexadecimal and Binary a sweet combination for Computer programmers

To understand why computer designers and programmers like hexadecimal so much consider the following, a nibble as we have learned is 4 bits and represents 0-15 combinations but so does a hexadecimal digit, thus we can group binary numbers into groups of 4 and each group represents a hexadecimal digit, the following table shows the relationship:

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Using the above table we can easily go back and forth between hexadecimal and binary by grouping the bits into groups of 4 as shown here:



The arrows indicate you can go either direction with this technique, thus the number above is:

0xC5CD12B6

With a 4:1 compaction ratio, you can see why it is a lot easier to write out binary numbers in hexadecimal format, and with a little practice you can learn the table and do it in your head.

Representing English Readable Information using Computer Bytes

So far we have been using numbering systems strictly to represent numbers, but computers must store information. Often in different forms, one form involves storing English Equivalent Symbols as specific binary Bytes. The following is often referred to as the ASCII Table (American Standard Code for Information Interchange). There is also more advanced codes called Unicode that is used to store information on multiple languages and uses words for the letters of the languages of the world.

Here is the ASCII Table:

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Some of the translations I will not discuss here (the special function ones: 0->31_{10}).

So if the computer has the following list of bytes in memory it would print a message based on them:

0x47 0x6F 0x64 0x20 0x69 0x73 0x20 0x67 0x72 0x65 0x61 0x74 0x21

↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕

G o d space i s space g r e a t !

Which of course would print out “God is great!” If you disagree with this one or think He does not exist, please stop reading this article, you have more important things to worry about like your eternal destiny!

The above message took 13 Bytes to encode. Thus we can use the computer information as bytes to represent the English language.

The most commonly used numbering system in the World

You might think with all the computers around that the most commonly used numbering system in the world is binary (base 2). You would be wrong it is actually base 4. In fact your body is using it right now. The human body contains roughly 30 Trillion Cells and all of them are actively using this numbering system to create proteins (molecular machines) to perform thousands of tasks and it is doing it every second you are alive. The code is stored on DNA, a very long strand of symbols that contains all the instructions necessary for each cell to function properly. Being there is roughly 7 Billion people on the planet then the rate of usage of this code just for the people (not including all the other living things on the planet) is an astonishing large number: 30 Trillion x 7 Billion or 210,000,000,000,000,000,000,000 cells are actively using it every second of every day. I have no way to describe to you a number of that size. In fact not just humans but ALL life from plants to every animal on the planet uses DNA within its cells to survive. The DNA stores the instructions to build the necessary proteins for all life on earth. It is a code and like all codes it was created by an intelligence, in this case God himself.

In molecular Biology the symbols for the codes are called nucleotides. There is 4 of them (thus base 4). And we have arbitrarily given letters to represent them, the 4 code letters of DNA is:

{A, C, T, G}

Because this numbering system is used primarily to store information we will not discuss the usage of it for numerical amounts, because it is not used that way.

Interestingly enough, just like we have groupings for base 10 (thousands) and base 2 (nibbles, bits, bytes, words, dwords and ddwords). God has chosen a grouping himself also. The symbols are grouped into three letter pairs called codons, thus the following is three codons:

ACT, TAC, CGT

What do the codons represent? There is 20 amino acids used by all life on earth, these codons map to those 20 amino acids and two additional code (STOP) codons which is like a period at the end of a sentence.

Being that is it base 4 and from things we have already learned there is 4^3 possible combinations of these codons or 64 codons.

Now I said they map to 20 amino acids, well there is more codons than amino acids so some of the codons actually code for the same amino acids, just like the ASCII table for computers here is the table for the codons used for the 20 amino acids:

FIRST LETTER (base)	SECOND LETTER (base) A		SECOND LETTER (base) U		SECOND LETTER (base) C		SECOND LETTER (base) G		THIRD LETTER (base)
A	AAA	Lysine	AUA	Isoleucine	ACA	Threonine	AGA	Arginine	A
	AAU	Asparagine	AUU	Isoleucine	ACU	Threonine	AGU	Serine	U
	AAC	Asparagine	AUC	Isoleucine	ACC	Threonine	AGC	Serine	C
	AAG	Lysine	AUG	Initiation Codon; Methionine	ACG	Threonine	AGG	Arginine	G
U	UAA	Stop Codon	UUA	Leucine	UCA	Serine	UGA	Stop Codon	A
	UAU	Tyrosine	UUU	Phenylalanine	UCU	Serine	UGU	Cysteine	U
	UAC	Tyrosine	UUC	Phenylalanine	UCC	Serine	UGC	Cysteine	C
	UAG	Stop Codon	UUG	Leucine	UCG	Serine	UGG	Tryptophan	G
C	CAA	Glutamine	CUA	Leucine	CCA	Proline	CGA	Arginine	A
	CAU	Histidine	CUU	Leucine	CCU	Proline	CGU	Arginine	U
	CAC	Histidine	CUC	Leucine	CCC	Proline	CGC	Arginine	C
	CAG	Glutamine	CUG	Leucine	CCG	Proline	CGG	Arginine	G
G	GAA	Glutamic Acid	GUA	Valine	GCA	Alanine	GGA	Glycine	A
	GAU	Aspartic Acid	GUU	Valine	GCU	Alanine	GGU	Glycine	U
	GAC	Aspartic Acid	GUC	Valine	GCC	Alanine	GGC	Glycine	C
	GAG	Glutamic Acid	GUG	Valine	GCG	Alanine	GGG	Glycine	G

Just like the ASCII table, you can use this table to decode any sequence of codons into its associated amino acid sequence, take the following sequence for example:

AAG, CAU, CAG, CUC, UAA

Using the table above this decodes to:

Lysine, Histidine, Glutamine, Leucine, STOP

The cell connects these amino acids together after reading the DNA (A complex process I am not going to go into here), even the connecting them together is an unbelievably complex process. The string of amino acids is then folded up into a 3 dimensional shape to form a molecular machine to carry out some necessary function in the cell.

For Cells, the next higher level above the codon is called a gene. A Gene is a sequence of codons (often hundreds and even thousands of codons long that ends with the STOP codon. The stop codon tells the cell where the gene ends, each gene thus encodes for a specific protein (molecular machine). A lot of the time these molecular machines are composed of hundreds of 'parts' each part being a protein encoded by the DNA. Then the cell assembles the parts to form higher level more complex machines that use multiple protein structures for their operation. For more information about this subject visit the internet you can read up on and see articles on molecular biology. Be careful because a lot of books have gone with the assumption that somehow this complex code happened by some freak accident. Being this is not observable, provable, or testable in the lab, the statement that it could have happened

by accident is simply an unscientific statement. When you see organization and a complex code, let common sense drive you, an intelligence is behind it!

Summary:

The numbering systems described here involve positional notation. The base of the numbering system taken to a given power determines the magnitude of each digit forming such numbering systems. Numbering systems can also be and often are used to encode information, be it codes humans create such as the ASCII table or much more elaborate codes like the DNA coding sequence.

If you have any questions or comments concerning this article. Please feel free to contact me:

Dennis Bingaman

ddbtech@hughes.net