Understanding Binary

My Binary Finger Counting page and binary tutorial have now been on the web for 9 years, almost as long as the web has been around! It's fun to see all the other Binary Finger Counting pages online (I even saw a T-Shirt! Cool!), but I hope you find this one to be as good as the rest.

I learned how to interpret binary from reading Michael Crichton's ***Andromeda Strain*** in college. Now, you gotta understand that for an artistic, creative-type like myself, suddenly understanding how binary works was a big deal, worthy of running out of the bathroom stall where I was reading the book and yelling *Eureka!* Of course, I immediately got a pen from my dorm room and went back and wrote the whole process on the bathroom stall walls for future reference.

It was shortly thereafter while discussing with friends how ancient cave artists allegedly kept track of the number of animals that they had killed by counting on their fingers that I learned Binary Finger Counting. Programmer extraordinaire/good friend ***Jim Reneau*** pointed out to me that if caveman had only known binary, he could have counted **1023 animals** instead of limiting himself to a mere 10 using 10 fingers. He then proceeded to demonstrate to me Binary Finger Counting.

Binary is the language of computers. Everything you type, input, output, send, retrieve, draw, paint, or place blame on when something doesn't work is, in the end, converted to the computer's native language- **binary**. But just how does this whole "on/off", "1/0", "hi/lo" thing work?

Binary is called a ***Base 2*** numbering system (the "bi" in the word binary was a dead giveaway). ***Base 2*** allows us to represent numbers from our ***Base 10*** system (called the decimal system - "dec" meaning 10) using only 2 digits - **1 and 0** - in various combinations.

Example of a typical binary number: 10001010
8-bit binary number representing the decimal number 138.

To the computer, **binary** digits aren't really **1s and 0s**. They're actually electrical impulses in either a (mostly) **on** state or a (mostly) **off** state. Just like a switch - either **on or off**. Since you only have 2 possible switch combinations or electrical possibilities, the computer only needs various combinations of 2 digits to represent numbers, letters, pixels, etc. These 2 digits, for our sake are visually represented by **1s and 0s**.

Bits, Bytes and Words

Binary numbers can be from 1 digit to infinity. But for our uses, there aren't too many numbers that we can't live without that can't be represented by 32-bit or 64-bit **binary** numbers. Let's start with the basics.

A single binary **1** or a single binary **0** is called a `bit`, which is short for "**b**inary dig**it**". A single bit by itself isn't of much use to the casual user, but can do wonders in the hands of a programmer working at the system level.

Take 4 of these **bits** and slap them together and they now form what's called a **nibble** (though this term isn't used very often). A nibble can represent the decimal values **0 to 15** (16 distinct values).

1	0	1	0

4-bit (Nibble) Sample

Take **8 bits** and put them together and you have one of the mostly commonly used computer terms in existence - the **byte**. A single **byte** can represent the decimal values **0 to 255** (256 distinct values) and since every possible character you can type on an English keyboard is represented by a number less than 128 to the computer (called ASCII codes) , a single letter of the alphabet takes 1 **byte** to represent internally (technically, you can represent all the letters of the alphabet using only 7-bits of a byte, but we won't get into that). When we speak of how much ram a computer has, we say it has 256-megabytes of ram, meaning 256 million **bytes** (most people nowadays just say 256 megs of ram and leave off the **byte** word).

1	0	1	1	1	0	1	0

8-bit (1 Byte) Sample

Place a couple of bytes together to represent a single value and you have a 16-bit **word** (2 bytes = 16-bits). A 16-bit word can represent the values **0 to 65535** (65536 distinct values). In the old days 16-bit **words** were used to form the addresses of 8-bit computers such as the Commodore 64, the Atari 800, and the Apple IIs, to name a few. These 16-bit **words** which were big enough to store an address for a 64k computer consisted of 2 bytes, a high byte and a low byte.

1 BYTE								1 BYTE							
1	0	0	0	1	0	1	0	1	0	0	0	1	0	1	0

16-bit (2 Byte) Sample

32-bit words are 4 bytes in length. They can represent a value from 0 to 4,294,967,295 (4,294,967,296 distinct values).

1 BYTE								1 BYTE								1 BYTE								1 BYTE							
1	0	0	1	1	1	0	1	1	0	0	0	0	1	1	1	0	1	1	0	0	1	1	1	0	1	0	0	0	1	1	1

32-bit (4 Byte) Sample

Pretty big number, but bigger still is the 64-bit word, which is, for you math-deprived people, 8 bytes in length (8 bytes times 8 bits per byte).

1 BYTE								1 BYTE								1 BYTE								1 BYTE								1 BYTE								1 BYTE								1 BYTE								1 BYTE						
1	0	1	1	0	0	0	1	1	1	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0	1	1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	1	1	0	0	0	11

64-bit (8 Byte) Sample

> This whopping monstrosity can represent a number from 0 to 1.844 E+19 if I understand my calculator correctly.

Okay, enough about bits, bytes and words. Let's figure out how to read a lowly 8-bit binary number.

How to Read a Binary Number

As stated previously, a byte consists of 8 bits, each bit having the possible value of either a 1 or a 0. Now when deciphering binary numbers, don't think of the 1 or 0 as an actual value itself, but a flag to determine whether it has any importance in the calculation of the final result. Lost? Let's look at an example:

Example binary number: 10001010
Binary representation of decimal 138.

Any time you're going to interpret a binary number, set something up on a piece of paper that looks like this-

128	64	32	16	8	4	2	1
1	**0**	**0**	**0**	**1**	**0**	**1**	**0**

Now, look at those numbers above the boxes with the red 1s and 0s. Those are decimal numbers representing powers of 2. Starting from the left and going to the right they are 2 to the 7th power (2^7), 2 to the 6th power (2^6), 2 to the 5th power (2^5), 2 to the 4th power (2^4), 2 to the 3rd power (2^3), 2 to the 2nd power (2^2), 2 to the 1st power (2^1) and 2 to the 0th power (2^0):

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

It just so happens that with powers of 2, each successive number is double the one before it. In other words if 2^0 is equal to 1 then 2^1 = 2 and 2^2 = 4 and 2^3 = 8 and so on. Here are the values of the powers of 2 going from 2^7 down to 2^0:

128 64 32 16 8 4 2 1

These are the values that are above the boxes. Now the actual binary number itself consists of 1s and 0s in the blue boxes which somehow magically represents the decimal number 138. How do we get 138 from 10001010? In the binary number when you see a 1, multiply that 1 times the value that is directly over it. Where you see a 0 in the box, just ignore it. So looking at our graphic again,

128	64	32	16	8	4	2	1
1	**0**	**0**	**0**	**1**	**0**	**1**	**0**

we see that there is a 1 under the 128, a 1 under the 8, and a 1 under the 2. If we add only those numbers which have a binary 1 in the box under them, we come up with 128+8+2 which equals 138.

Here's another example:

128	64	32	16	8	4	2	1
1	**1**	**1**	**0**	**0**	**1**	**1**	**0**

Thus, binary **11100110** is equal to 128+64+32+4+2 which is decimal **230**

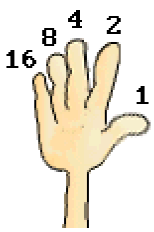
And another one:

128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	1

Thus, binary **10000001** is equal to 128+1 which is decimal **129**.

Hope this makes more sense than when you started. Be sure to check out the section on Binary Finger Counting to learn how to count to 31 on one hand. See ya!

Finger Counting

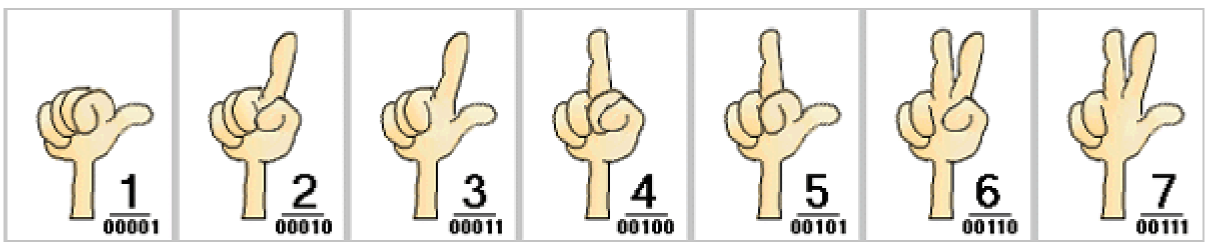


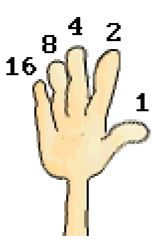
Binary Finger Counting: 1 to 7

The thumb represents the value 1, the index is double that, so it's 2, and the middle is double the index, so it's 4, etc. This is key to understanding and fluently counting in binary on your fingers.

To make the number 3 for instance, you have to combine a 2 finger and a 1 finger (2+1=3).

For a Shockwave version of this (168k), click **HERE**

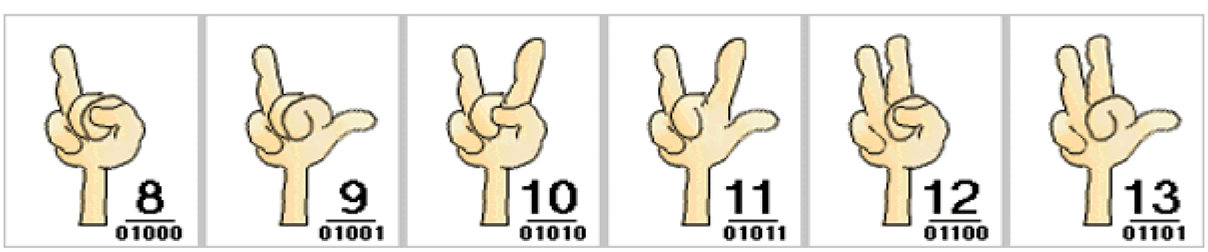




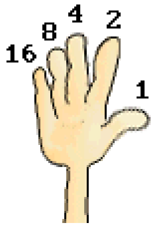
Binary Finger Counting: 8 to 13

The thumb represents the value 1, the index is double that, so it's 2, and the middle is double the index, so it's 4, etc. This is key to understanding and fluently counting in binary on your fingers.

To make the number 11 for instance, you have to combine an 8 finger, a 2 finger and a 1 finger (8+2+1=11).

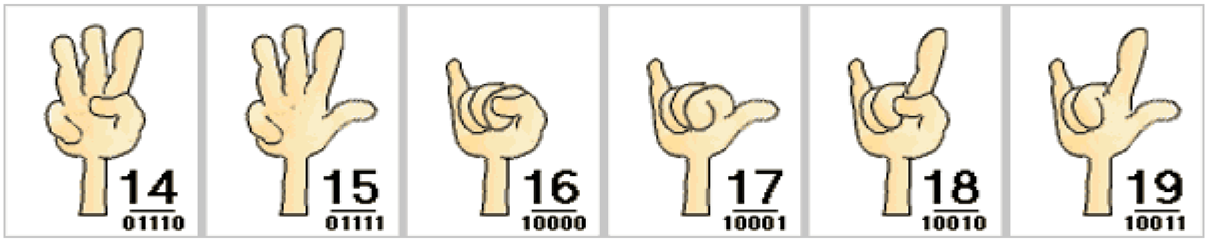


Binary Finger Counting: 14 to 19

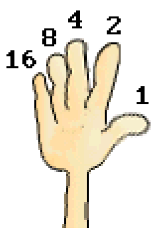


The thumb represents the value 1, the index is double that, so it's 2, and the middle is double the index, so it's 4, etc. This is key to understanding and fluently counting in binary on your fingers.

To make the number 18 for instance, you have to combine a 16 and a 2 finger (16+2=18).

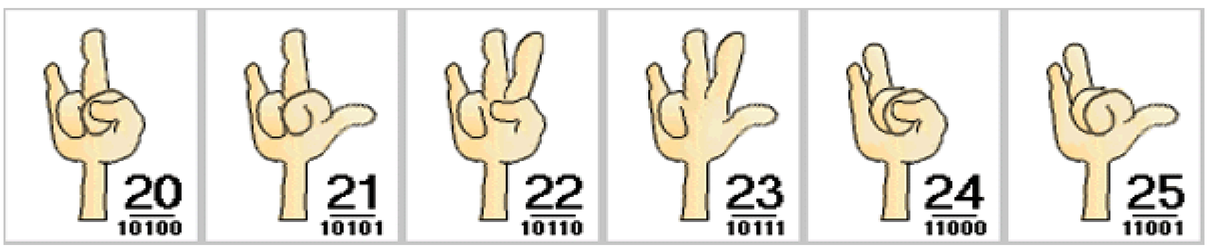


Binary Finger Counting: 20 to 25

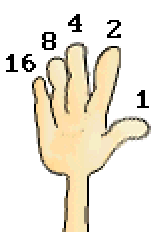


The thumb represents the value 1, the index is double that, so it's 2, and the middle is double the index, so it's 4, etc. This is key to understanding and fluently counting in binary on your fingers.

To make the number 23 for instance, you have to combine the 16 finger, the 4 finger, the 2 finger and the 1 finger (16+4+2+1=23).

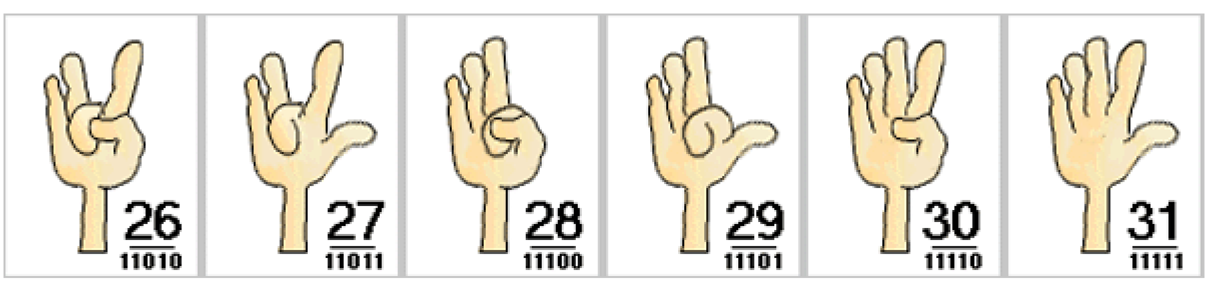


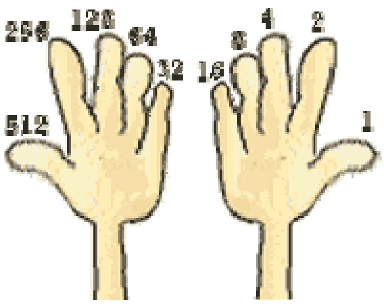
Binary Finger Counting: 26 to 31



The thumb represents the value 1, the index is double that, so it's 2, and the middle is double the index, so it's 4, etc. This is key to understanding and fluently counting in binary on your fingers.

To make the number 28 for instance, you have to combine the 16 finger, the 8 finger, and the 4 finger (16+8+4=28).



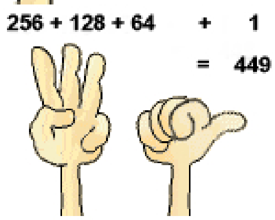


So How Do We Get 1023?
Take a look at this illustration-

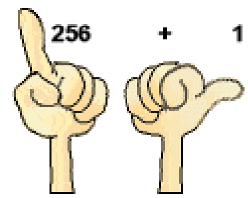
Once you've used all 5 fingers on the right hand, start with the pinky of the left hand. The pinky would be double the pinky of the right hand, so its value would be 32.

For a Shockwave version of this (168k), click **HERE**

So to represent the number 449 on your fingers:



And to represent the number 257:



Understanding Hexadecimal

Ah, hexadecimal. If you've ever worked with colors in web page design, you've proably seen something like '<body bgcolor="#A09CF3"> or something to that effect. Somehow, that 6 digit hexadecimal number is equal to a lavender or light purple color. What on earth does 'A09CF3' mean? Before we explain that, let's look at what hexadecimal (hex) is.

Our decimal system, as mentioned in my **Binary Tutorial**, is a base-10 system, meaning we can count to any number in the universe using only 10 symbols or digits, 0 thru 9.

For the computer, a 10-based system probably isn't the most efficient system, so the computer uses binary (in reality, it uses microscopic switches which are either on or off, but we represent them using the digits '1' and '0'). Unfortunately, binary isn't very efficient for humans, so to sort of find a happy middle ground, programmers came up with hexadecimal.

Hexadecimal is a base-16 number format (hex=6, decimal=10). This means that instead of having only the digits from '0' to '9' to work with like our familiar decimal, or '1' and '0' like binary, we have the digits '0' to '15'. It also means that we are using the powers of 16, instead of the powers of 2 like in binary.

Digits '0' thru '9' are the same as our decimal system, but how can 10 thru 15 be digits? Well, since there are no 10 thru 15 symbols, we have to invent some.

To count beyond 9 in hexadecimal, we use A thru F of the alphabet:

DECIMAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEXADECIMAL	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

As soon as you count over 9 in hex, new digits take over. A=10, B=11, C=12, D=13, E=14 and F=15. Alright, so we have 6 new digits we never saw before. What can we do with them? Let's look at some samples-

$0F **(Pronounced "OH EFF" or "ZERO EFF")**

Note the use of the $ to show hex. You'll also see a hex number like this:

0x0F

In fact, the '0x' in front of a hex number is more current than the '$'. The '$' is sort of old school. Whether you use a '$' or a '0x', it tells you that you're working with a hex number, not a decimal one.Why use a special symbol to denote hex? Isn't it obvious that '0F' isn't an ordinary decimal number? Sure, but what if you had this number:

5396

Hmmm. How do you know whether that is a decimal 5396 or a hex 5396? See, not all hex numbers will have an 'A' thru 'F' in them. Many do, but not all. But as soon as we put a '$' or a '0x' in front of that number

$5396 or 0x5396

We know it's a hexadecimal number. Look, I'm already tired of typing it both ways, so I'm going to keep using the '$' to denote hex instead of showing you both methods from now on. If you feel more comfortable with the '0x', feel free to substitute that for the rest of the tutorial.

Here's another:

$A0FF **(Pronounced "A OH EFF EFF" or "A ZERO EFF EFF")**

And here's some others (yes, they're legitimate hex numbers):

(Numbers in parentheses are the decimal versions)

$BEAD (48,813)	**$DEAD** (57,005)	**$FEED** (65,261)	**$DEAF** (57,007)	**$FADE** (64,222)	**$BABE** (47,806)
$ABBA (43,962)	**$DEED** (57,069)	**$FACE** (64,206)	**$BEEF** (48,879)	**$CAD** (3,245)	**$ADD** (2,781)
$CAFE (51,966)	**$CAB** (3,243)	**$BAD** (2,989)	**$ACE** (2,766)	**$FAD** (4,013)	**$BED** (3,053)

WHERE'S THE HEX??!?

Now, you may be wondering, "Hey, it's all well and good that you're *showing* us hex examples, but how do we go about translating those hex numbers into decimal?"

Well, I'll tell you... (and to keep things simple, we'll stick to smaller numbers for now):

Let's look at a typical small decimal number.

235

Now, in decimal, the above number is equivalent to:

2 x 100 + 3 x 10 + 5 or 200 + 30 + 5

Easy enough. Now look at a slightly larger decimal number:

1,236

This is equivalent to:

1 x 1000 + 2 x 100 + 3 x 10 + 6 or 1,000 + 200 + 30 + 6

Well, hex numbers follow a similar pattern where each digit is in a "place". But instead of 10's place or 100's place or 1000's place, hex uses the following progression (from right to left):

4096's Place	256's Place	16's Place	1's Place

Then, to translate a hex number such as **$A0FF**, you set up a chart like this:

4096	**256**	**16**	**1**
A	**0**	**F**	**F**

Remember that hex 'A' actually is equal to decimal 10, and hex 'F' is actually decimal 15 (look at the chart above if you just felt your brain drop into your underwear and are totally lost. I don't know where that image came from...). Here's our revised chart:

4096	**256**	**16**	**1**
10	**0**	**15**	**15**

Now multiply 10 times 4096, then 0 times 256, then 15 times 16, then 15 times 1. Then add the results (10 times 4096=40960 + 0 times 256=0 + 15 times 16=240 + 15 times 1=15 which comes out to decimal 41215.

SAY WHAT?!?!?

Still with me? No? Here's a smaller example for the 'larger example impaired':

2 Digit Hex Number: $B9

16's Place	1's Place
B	**9**

Remember that hex 'B' = 11 and hex '9' is still just 9:

16's Place	1's Place
11	**9**

Or

16	**1**
11	**9**

Since 'B' = 11, you multiply 11 times 16 which equals 176, then add a 9 times 1.

The final translation of $B9 to decimal would thus be 11 times 16 + 9 times 1=decimal 185.

Another 4 Digit Hex Number: $FEBC

4096	256	16	1
F	E	B	C

Remember that hex 'F' = 15, hex 'E' = 14, hex 'B'=11, and hex 'C' = 12:

4096	256	16	1
15	14	11	12

So multiply 15 times 4096, 14 times 256, 11 times 16, and 12 times 1 and add the results-

15 x 4096 =61440 + 14 x 256=3584 + 11 x 16=176 + 12 x 1 = 65212.

WHAT ABOUT REALLY LARGE HEX NUMBERS?!?!?

To translate (or convert as the big boys say) large hex numbers, you need to know more powers of 16. We've already seen the first four powers of 16: 4096 (16^3), 256 (16^2), 16 (16^1), and 1 (16^0). Whipping out my calculator (you think I studied multiplications tables in school larger than 12?!?!?), I find that after 4096 (16^3), the powers of 16 are:

16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0
268,435,456	16,777,216	1,048,576	65536	4096	256	16	1

Whew! Big numbers!!! So if we had a hex number like:

$FE973BDC

We could convert it by setting up the table like this:

268,435,456	16,777,216	1,048,576	65536	4096	256	16	1
F	E	9	7	3	B	D	C

This is going to be a big number. You'll definitely want your calculator for this one. Here goes:

Convert the hex digits to decimal digits using the chart at the top:

268,435,456	16,777,216	1,048,576	65536	4096	256	16	1
15	14	9	7	3	11	13	12

Multiply the decimal value in the bottom box by the powers of 16 values in the top box:

15 x 268,435,456 + 14 x 16,777,216 + 9 x 1,048,576 + 7 x 65,536 + 3 x 4,096 + 11 x 256 + 13 x 16 + 12 x 1 =

4,026,531,840 + 234,881,024 + 9,437,184 + 458,752 + 12,288 + 2,816 + 208 + 12 =4,271,324,124.

Ow! That hurts my brain. I'm resting for awhile...

HEXADECIMAL AND BINARY'S RELATIONSHIP

Alright, I'm back.

There is a wonderful (if you're a geek like me) relationship between hexadecimal and binary that might not be readily apparent. In fact, they're so closely tied together, that many programmers learn both equally well (especially assembler programmers). Let's take a look.

Here's a binary number (again, review my **Binary Tutorial** if you're a bit rusty. Get it? A 'bit' rusty. I kill me):

#10111101

Notice the pound sign to signify that it is a binary number so we don't confuse decimal 10,111,101 with binary 10111101.

So, based on our tutorial, we set up something like this first:

128	64	32	16	8	4	2	1
1	0	1	1	1	1	0	1

Now, add together all the numbers in the top boxes that have a 1 below them and ignore the numbers with zeroes below them.

128+32+16+8+4+1=189 in decimal.

Now if you split that little 8-bit binary number into 2 sets of 4 bits, let's call them the leftmost 4 bits and the rightmost 4 bits, we get

8	4	2	1		8	4	2	1
1	0	1	1		1	1	0	1

What the?!? Notice, we dumped the 128, 64, 32 and 16, because we're now working with 2 sets of 4 bit numbers instead of 1 big 8-bit number.

Wow that makes it so much easier...

To convert our binary number to hex, figure out what the leftmost 4 bits is equal to, and the rightmost 4 bits.

LEFTMOST	1011	=8+2+1	=11	=$B
RIGHTMOST	1101	=8+4+1	=13	=$D

And since 11 decimal = $B hex and 13 decimal = $D hex, the final hexidecimal conversion of binary #10111101 = $BD.

For larger binary numbers, it still works. Take this 16 bit number:

#1011010010100111

Break it up into four 4-bit sets:

8	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**
1	0	1	1	0	1	0	0	1	0	1	0	0	1	1	1

And add up the numbers which have a 1 below them, but be sure to keep it in 4 different sets:

8+0+2+1	0+4+0+0	8+0+2+0	0+4+2+1
11	4	10	7
B	4	A	7

So #1011010010100111 is equal to $B4A7 hex and 46247 decimal.

Here's a 32-bit binary number to convert to hex:

#11010100010100111111011100010101

Break it up into eight 4-bit chunks...

8	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**	**8**	**4**	**2**	**1**
1	1	0	1	0	1	0	0	0	1	0	1	0	0	1	1	1	1	1	1	0	1	1	1	0	0	0	1	0	1	0	1

Add up all numbers that have a 1 below them:

8+4+ 0+1	0+4+ 0+0	0+4+ 0+1	0+0+ 2+1	8+4+ 2+1	0+4+ 2+1	0+0+ 0+1	0+4+ 0+1
13	4	5	3	15	7	1	5
D	4	5	3	F	7	1	5

So binary 11010100010100111111011100010101 is equal to $D453F715 hex. Cool.

Here are the numbers 0 to 255 in decimal, hex and binary.

DEC	HEX	BIN	DEC	HEX	BIN	DEC	HEX	BIN	DEC	HEX	BIN
0	0	00000000	1	1	00000001	2	2	00000010	3	3	00000011
4	4	00000100	5	5	00000101	6	6	00000110	7	7	00000111
8	8	00001000	9	9	00001001	10	A	00001010	11	B	00001011
12	C	00001100	13	D	00001101	14	E	00001110	15	F	00001111
16	10	00010000	17	11	00010001	18	12	00010010	19	13	00010011
20	14	00010100	21	15	00010101	22	16	00010110	23	17	00010111
24	18	00011000	25	19	00011001	26	1A	00011010	27	1B	00011011
28	1C	00011100	29	1D	00011101	30	1E	00011110	31	1F	00011111
32	20	00100000	33	21	00100001	34	22	00100010	35	23	00100011
36	24	00100100	37	25	00100101	38	26	00100110	39	27	00100111
40	28	00101000	41	29	00101001	42	2A	00101010	43	2B	00101011
44	2C	00101100	45	2D	00101101	46	2E	00101110	47	2F	00101111
48	30	00110000	49	31	00110001	50	32	00110010	51	33	00110011
52	34	00110100	53	35	00110101	54	36	00110110	55	37	00110111
56	38	00111000	57	39	00111001	58	3A	00111010	59	3B	00111011
60	3C	00111100	61	3D	00111101	62	3E	00111110	63	3F	00111111
64	40	01000000	65	41	01000001	66	42	01000010	67	43	01000011
68	44	01000100	69	45	01000101	70	46	01000110	71	47	01000111
72	48	01001000	73	49	01001001	74	4A	01001010	75	4B	01001011
76	4C	01001100	77	4D	01001101	78	4E	01001110	79	4F	01001111
80	50	01010000	81	51	01010001	82	52	01010010	83	53	01010011
84	54	01010100	85	55	01010101	86	56	01010110	87	57	01010111
88	58	01011000	89	59	01011001	90	5A	01011010	91	5B	01011011
92	5C	01011100	93	5D	01011101	94	5E	01011110	95	5F	01011111
96	60	01100000	97	61	01100001	98	62	01100010	99	63	01100011
100	64	01100100	101	65	01100101	102	66	01100110	103	67	01100111
104	68	01101000	105	69	01101001	106	6A	01101010	107	6B	01101011
108	6C	01101100	109	6D	01101101	110	6E	01101110	111	6F	01101111
112	70	01110000	113	71	01110001	114	72	01110010	115	73	01110011
116	74	01110100	117	75	01110101	118	76	01110110	119	77	01110111
120	78	01111000	121	79	01111001	122	7A	01111010	123	7B	01111011
124	7C	01111100	125	7D	01111101	126	7E	01111110	127	7F	01111111
128	80	10000000	129	81	10000001	130	82	10000010	131	83	10000011

132	84	10000100	133	85	10000101	134	86	10000110	135	87	10000111
136	88	10001000	137	89	10001001	138	8A	10001010	139	8B	10001011
140	8C	10001100	141	8D	10001101	142	8E	10001110	143	8F	10001111
144	90	10010000	145	91	10010001	146	92	10010010	147	93	10010011
148	94	10010100	149	95	10010101	150	96	10010110	151	97	10010111
152	98	10011000	153	99	10011001	154	9A	10011010	155	9B	10011011
156	9C	10011100	157	9D	10011101	158	9E	10011110	159	9F	10011111
160	A0	10100000	161	A1	10100001	162	A2	10100010	163	A3	10100011
164	A4	10100100	165	A5	10100101	166	A6	10100110	167	A7	10100111
168	A8	10101000	169	A9	10101001	170	AA	10101010	171	AB	10101011
172	AC	10101100	173	AD	10101101	174	AE	10101110	175	AF	10101111
176	B0	10110000	177	B1	10110001	178	B2	10110010	179	B3	10110011
180	B4	10110100	181	B5	10110101	182	B6	10110110	183	B7	10110111
184	B8	10111000	185	B9	10111001	186	BA	10111010	187	BB	10111011
188	BC	10111100	189	BD	10111101	190	BE	10111110	191	BF	10111111
192	C0	11000000	193	C1	11000001	194	C2	11000010	195	C3	11000011
196	C4	11000100	197	C5	11000101	198	C6	11000110	199	C7	11000111
200	C8	11001000	201	C9	11001001	202	CA	11001010	203	CB	11001011
204	CC	11001100	205	CD	11001101	206	CE	11001110	207	CF	11001111
208	D0	11010000	209	D1	11010001	210	D2	11010010	211	D3	11010011
212	D4	11010100	213	D5	11010101	214	D6	11010110	215	D7	11010111
216	D8	11011000	217	D9	11011001	218	DA	11011010	219	DB	11011011
220	DC	11011100	221	DD	11011101	222	DE	11011110	223	DF	11011111
224	E0	11100000	225	E1	11100001	226	E2	11100010	227	E3	11100011
228	E4	11100100	229	E5	11100101	230	E6	11100110	231	E7	11100111
232	E8	11101000	233	E9	11101001	234	EA	11101010	235	EB	11101011
236	EC	11101100	237	ED	11101101	238	EE	11101110	239	EF	11101111
240	F0	11110000	241	F1	11110001	242	F2	11110010	243	F3	11110011
244	F4	11110100	245	F5	11110101	246	F6	11110110	247	F7	11110111
248	F8	11111000	249	F9	11111001	250	FA	11111010	251	FB	11111011
252	FC	11111100	253	FD	11111101	254	FE	11111110	255	FF	11111111