Base Conversion

written by Cathy Saxton

1. Base 10

In base 10, the digits, from right to left, specify the 1's, 10's, 100's, 1000's, etc.
These are powers of 10 (10^x): $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$, etc.
Each digit in a base 10 number can have a value from 0 to 9.
Base 10 numbers are commonly referred to as "decimal" numbers.

2. Notation

As we work with numbers, the following notation will be used. The top row of the table shows the value for each place in the number. The second row shows the actual digits in the number.
253 is represented by:

100's	10's	1's	← *places*
2	5	3	← *digits*

3. General Base Definitions

Base *n* means that each place (x) in the number represents n^x. (n^0, n^1, n^2, n^3, etc.)
Valid digits in base *n* are 0 to *n*-1.
When working with numbers in different bases, a subscript is usually used to indicate the base
(e.g. 253_{10} means 253 in base 10).
We can represent any number in any base. The number will usually look different when expressed in another base, but its *value* is the same. When a number is stored in the computer's memory, it has a specific value; we can show it using base 10, base 2, base 16, etc., but the value never changes -- only the representation changes.

4. Base 2: Binary

Each digit in a base 2 number can have a value of 0 or 1.
Places are 1's, 2's, 4's, 8's, etc. ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, etc.)
Computers store numbers in binary, which is the common name for base 2 numbers. Each binary digit corresponds to one bit of memory. There are 8 bits (or 8 binary digits) in a byte. The computer stores values in memory by turning these bits on (1) or off (0) to correspond to the binary representation of the number.

4.1. Converting binary to decimal

Converting from binary (base 2) to decimal (base 10) requires doing a little math with the digits in each place. Here is an example for converting a binary value of 1001 to its decimal equivalent.

$1001_2 = ?_{10}$

Remember that the places are 1's, 2's, 4's, 8's, etc:

(2^3)	(2^2)	(2^1)	(2^0)
8's	**4**'s	**2**'s	**1**'s
1	0	0	1

We make the following calculation to get the equivalent decimal number:

$1001_2 = (1 * \mathbf{8}) + (0 * \mathbf{4}) + (0 * \mathbf{2}) + (1 * \mathbf{1}) = 8 + 0 + 0 + 1 = \boxed{9_{10}}$

Here's another example:
$11010_2 = ?_{10}$

16's	**8**'s	**4**'s	**2**'s	**1**'s
1	1	0	1	0

$11010_2 = (1 * \mathbf{16}) + (1 * \mathbf{8}) + (0 * \mathbf{4}) + (1 * \mathbf{2}) + (0 * \mathbf{1}) = 16 + 8 + 0 + 2 + 0 = \boxed{26_{10}}$

4.2. Converting decimal to binary

Converting from decimal to binary is a little trickier. Let's consider the case of converting a decimal 5 to its binary representation.

$5_{10} = ?_2$

We'll calculate the appropriate digit for each place in the resulting binary number, working from the left to the right. First, we determine the largest power of 2 that's less than or equal to the number. In this case, 8 (2^3) is too large, so we start with the 4 (2^2). Therefore, we know the result will contain digits in three places -- 4's, 2's and 1's:

4's	**2**'s	**1**'s
?	?	?

We need to figure out which digits should be 1's, and which should be 0's (since these are the only legal values for a binary digit). We can see that putting a 1 in the 4's place will help:

4's	**2**'s	**1**'s
1	?	?

This accounts for 4 of the 5 "units" that we need to represent. That means that we can't have any 2's (or the total would become larger than our goal of 5). So, we put a 0 in the 2's place:

4's	**2**'s	**1**'s
1	0	?

That leaves the 1's place to fill, and we see that putting a 1 there will give us the desired result:

4's	**2**'s	**1**'s
1	0	1

← *this is the equivalent binary value*

$5_{10} = (1 * \mathbf{4}) + (0 * \mathbf{2}) + (1 * \mathbf{1}) = \boxed{101_2}$

Let's work though another example:
$22_{10} = ?_2$

In this example, we need to start with the 16's place (2^4); the next place (2^5) is the 32's, which is too large:

16's	**8**'s	**4**'s	**2**'s	**1**'s
?	?	?	?	?

Putting a 1 in the 16's place means that we've accounted for 16 of 22 "units". This leaves 22 - 16 = 6 remaining:

16's	**8**'s	**4**'s	**2**'s	**1**'s
1	?	?	?	?

total so far: 16 remaining: 6

We can't use any 8's since we only have 6 units remaining. (You can also observe that 16 + 8 = 24, which is too large.) So, we put a 0 in the 8's place:

16's	**8**'s	**4**'s	**2**'s	**1**'s
1	0	?	?	?

total so far: 16 remaining: 6

We now see whether we can use a 4. Since 4 is less than the remaining 6, the 4 will be helpful, so we put a 1 in the 4's place. That means that we now have remaining: 6 (our previous value) - 4 (what we just accounted for) = 2.

16's	**8**'s	**4**'s	**2**'s	**1**'s
1	0	1	?	?

total so far: 16 + 4 = 20 remaining: 2

Next we look at the 2's. Since that's exactly what we still need, we put a 1 in the 2's place. That leaves us with 0 remaining. Since we don't need anything more, we put a 0 in the 1's place:

16's	**8**'s	**4**'s	**2**'s	**1**'s
1	0	1	1	0

← *the binary representation for* 22_{10}

$22_{10} = (1 * \mathbf{16}) + (0 * \mathbf{8}) + (1 * \mathbf{4}) + (1 * \mathbf{2}) + (0 * \mathbf{1}) = \boxed{10110_2}$

5. Base 16: Hexadecimal

As we look at a numbers stored by the computer, it quickly gets cumbersome to deal with long sequences of 0's and 1's in the binary representation. Programmers often use base 16 since it provides a convenient way to express numbers and is easily converted to and from base 2.

For a base 16 number, the places are 1's, 16's, 256's, etc. ($16^0 = 1$, $16^1 = 16$, $16^2 = 256$).
Each digit can hold a value from 0 to 15. In order to have single characters for representing 10, 11, 12, 13, 14, and 15, we use A, B, C, D, E, and F, respectively. Thus, the valid values for a base 16 digit are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
Base 16 numbers are commonly referred to as "hexadecimal" or "hex" numbers. In C/C++ programming, hex numbers often use the prefix "0x" to indicate hex numbers, e.g. 0x3AC.

5.1. Converting hex to decimal

Converting from hex (base 16) to decimal (base 10) is similar to converting binary to decimal. Here is an example for converting a hex value of 45 to its decimal representation.

$45_{16} = ?_{10}$

Remember that the places are 1's and 16's:

(16^1)	(16^0)
16's	**1**'s
4	5

We can make the following calculation to get the equivalent decimal number:

$45_{16} = (4 * \mathbf{16}) + (5 * \mathbf{1}) = 64 + 5 = \boxed{69_{10}}$

Here's another example.
$3AC_{16} = ?_{10}$

(16^2)	(16^1)	(16^0)
256's	**16**'s	**1**'s
3	A (=10)	C (=12)

$3AC_{16} = (3 * \mathbf{256}) + (10 * \mathbf{16}) + (12 * \mathbf{1}) = 768 + 160 + 12 = \boxed{940_{10}}$

5.2. Converting decimal to hex

To convert from decimal to hex, we use the same technique as we did above when converting from decimal to binary. Let's consider the case of converting a decimal 73 to its hex equivalent.

$73_{10} = ?_{16}$

We'll calculate the appropriate digit for each place in the resulting hex number, working from the left to the right. First, we determine the largest power of 16 that's less than or equal to the number. In this case, 256 (16^2) is too large, so we'll start with the 16's place. So, we know the result will contain digits in two places -- 16's and 1's:

16's	**1**'s
?	?

73 / **16** = 4 with a remainder of 9. So, there are four 16's in 73. Now we know part of the result:

16's	**1**'s
4	?

So, we've accounted for 4 * **16** = 64 of the 73. That leaves the remainder, 9, which we take care of by putting it in the 1's place:

16's	**1**'s
4	9

$73_{10} = (4 * \mathbf{16}) + (9 * \mathbf{1}) = \boxed{49_{16}}$

Let's work though another example:
$940_{10} = ?_{16}$

In this case, we start with 256 (16^2). The next place is $16^3 = 4096$, which is larger than our value. The resulting hex number will have 3 digits:

256's	**16**'s	**1**'s
?	?	?

Calculate the digit for the 256's place: 940 / **256** = 3 with a remainder of 172. So, we'll put a 3 in the 256's place.

256's	**16**'s	**1**'s
3	?	?

Calculate how much is left over: we've accounted for 3 * **256** = 768; that leaves 940 - 768 = 172.
Calculate the digit for the 16's place: 172 / **16** = 10 with a remainder of 12. So, we need ten 16's.
Recall that 10 is represented with A.

256's	**16**'s	**1**'s
3	A	?

The remainder, 12, tells us how much we still need to take care of. We put that value in the 1's place, using C to represent 12.

$940_{10} = (3 * \mathbf{256}) + (10 * \mathbf{16}) + (12 * \mathbf{1}) = \boxed{3AC_{16}}$

6. Converting Between Binary and Hexadecimal

Converting between base 2 and base 16 is much simpler than converting either to base 10.

The minimum value that can be expressed with four binary digits is 0000, or 0_{10}.
The maximum value that can be expressed with four binary digits is 1111, or 15_{10} (8 + 4 + 2 + 1).
So, we see that four binary digits can be used to represent the numbers 0-15, which is exactly the range represented by a single hex digit. Here are conversions from binary to hex:

$0000_2 = 0_{16}$	$0100_2 = 4_{16}$	$1000_2 = 8_{16}$	$1100_2 = C_{16}$
$0001_2 = 1_{16}$	$0101_2 = 5_{16}$	$1001_2 = 9_{16}$	$1101_2 = D_{16}$
$0010_2 = 2_{16}$	$0110_2 = 6_{16}$	$1010_2 = A_{16}$	$1110_2 = E_{16}$
$0011_2 = 3_{16}$	$0111_2 = 7_{16}$	$1011_2 = B_{16}$	$1111_2 = F_{16}$

Notice that every value that can be expressed with four binary digits can be expressed with a single hex digit, and every value that can be expressed with a single hex digit can be expressed with four binary digits.

Every hex digit can be represented with four binary digits.
Every four binary digits can be converted to one hex digit.

To convert from binary to hex, separate the binary number into groups of four digits *starting on the right*. Each group can be converted to a hex value.

For example, 10010010 can be visualized as two 4-digit groups: 1001 0010; each group is converted to a hex value, and we get a result of 92:

1	0	0	1	0	0	1	0
9				2			

$10010010_2 = 92_{16}$

For binary numbers like 100110 that don't divide evenly into 4-digit groups, add 0's to the left as necessary: $\underline{00}10\ 0110_2 = 26_{16}$.

Converting hex to binary is just the reverse; be careful to write *four* binary digits for each hex digit -- that means starting with a zero for values less than 8.
$B4_{16} = 1011\ \underline{0}100_2$
You don't need the zeros at the beginning of the binary number:
$6F_{16} = \_110\ 1111_2$

Memory is just a continuous span of 1's and 0's. Modern computers typically like to retrieve 32 bits at a time. While we often use all 32 bits to represent a single value, it is also common to use that 32-bit space to store 2 or 4 values. Consider a 32-bit number such as:

00000000010001011010000000100011

The following table shows how we can interpret that memory -- as a single 32-bit value, two 16-bit values, or four 8-bit values:

	Binary Representation	**Hex**	**Decimal**
One Value:	00000000010001011010000000100011	0045A023	4562979
Two Values:	0000000001000101, 1010000000100011	0045, A023	69, 40995
Four Values:	00000000, 01000101, 10100000, 00100011	00, 45, A0, 23	0, 69, 160, 35

Notice that by using the hex representation for the 32-bit value, you can easily determine the 2 or 4 values that are held in the 32 bits of memory.

Ultimately, the computer uses binary to store numbers. Unfortunately, representing numbers in binary is very verbose and hard to read. Using a base that's a power of 2 retains the binary nature of computer math, but yields more compact and readable numbers. Base 4 isn't much of an improvement. Base 8 is good, but represents binary digits in groups of three, while computers generally use groups of eight. Base 256 would require too many symbols. Base 16 represents binary digits in convenient groups of four, and only requires 16 symbols. That is why hex is commonly used when programming.

7. Other Interesting Facts

When we put a 0 at the end of a base 10 number, it has the effect of multiplying the value by 10; e.g. adding a 0 after 37 gives us 370, or 37 * 10. The same works for other bases, with the multiplier being the base number; e.g. 101_2 (5_{10}) becomes 1010_2 (10_{10} = 5 [the original value] * 2 [the base]).

Dropping the last digit does an integer divide by the base; e.g. 368 / 10 = 36.

Base 8's common name is octal. In C or C++, if you start an integer with a 0, it will be interpreted as an octal value. Conversion from binary to octal is like converting to hex, but using groups of *three* binary digits; e.g. 110 $101_2 = 65_8$.

When converting between hex and octal, the easiest way is to start by representing the value in binary!

8. Summary

base	**common name**	**places**	**valid digit values**
10	decimal	1, 10, 100, 1000, etc.	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
2	binary	1, 2, 4, 8, 16, 32, etc.	0, 1
16	hexadecimal, hex	1, 16, 256, 4096, etc.	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F