Number Systems

Using and Converting Between Decimal, Binary, Octal and Hexadecimal Number Systems

In everyday life, we humans most often count using ‘decimal’ or ‘base-10’ numbers. In computer science, it is often more convenient to use different number systems, especially ‘base-2’ (binary), ‘base-8’ (octal) and ‘base-16’ (hexadecimal). We don’t use octal all that much, but include it here for completeness. There are many techniques for converting to and from different number systems and the one explained here is one that the instructor has found to be both easy and in common use in the computer industry.

The number systems discussed here are ‘positional’. This means that we have a certain group of symbols and the position that a specific symbol has dictates its numeric value. For this document, we will only consider integer values – no fractions or values after a, for example, decimal point, will be used.

DECIMAL (Base-10)

The decimal number system is one that is common in everyday usage. In this system, we have ten symbols – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 – and we form larger numbers by grouping these symbols. In decimal, we call these ten symbols decimal digits. When we see the number 452, we interpret this as ‘four hundred fifty two’. The number 245 is interpreted as ‘two hundred forty five’. We can see that the value of the number is dependent on how the digits are interpreted based on position. The ‘2’ was interpreted as ‘two’ in the first example and as ‘two hundred’ in the second example.

While most people take for granted the mathematics behind this decimal notation, we will discuss it in some detail as the underlying mathematics are important when converting to and from other number systems. Remember that we are only dealing with integers.

So just what does the number ‘452’ actually mean from a mathematical perspective? To understand that, we need to be a little bit more formal about this whole ‘positional’ numbers thing that we mentioned earlier.

When we were younger, it is possible that the number 452 would have been explained in a manner such as this: There are two ones, 5 tens, and 4 hundreds all added together to get four hundred fifty two. What that tells us is that, beginning from the right, there is a ones place, a tens place and a hundreds place. But there is a mathematical way of looking at this that is really useful (at times).

Let’s number the positions beginning from the right with position zero – in computer science, we seem to always want to start numbering from zero. We say that for the number 452 that the ‘2’ occupies position zero, the ‘5’ occupies position one, and the ‘4’ occupies position two. Because we are using decimal, which is also known as base-10, we know that, for example, the ‘4’ in 452 occupies position 2 and that position 2 corresponds to the number of ‘hundreds’. Please note that one hundred can be written as 10^2 as well. Interestingly, since we are using base-10 counting, we find that the value of the number is found by taking the symbol, multiplying it by the base raised to the number of the position. A bit confusing, but let’s do a few examples.

Example 1

The number 452, when written out using our positional notation becomes:

$$(2 \times 10^0) + (5 \times 10^1) + (4 \times 10^2) = (2 \times 1) + (5 \times 10) + (4 \times 100) = 2 + 50 + 400 = 452$$

Example 2

The number 2048, when written out using our positional notation becomes:

$$(8 \times 10^0) + (4 \times 10^1) + (0 \times 10^2) + (2 \times 10^3) =$$

$$(8 \times 1) + (4 \times 10) + (0 \times 100) + (2 \times 1000) = 8 + 40 + 0 + 2000 = 2048$$

In example 2, notice that the position of the zero symbol had specific meaning and could not be left out, even though when we add all the numbers together, adding zero to the other numbers did not change the result. The zero was needed because it occupies a ‘position’ and that position affects the values all the numbers to the left of the zero in the final, symbolic form. If we had left out the ‘0’ the number would have been written as ‘248’ and it is left as an exercise to the students to see that this number, in positional notation, is much different from 2048.

A quick note on the term ‘base-10’ should be made at this point. Did you see how we took the value of the position and made it the exponent for the number ten? Ten is our ‘base’ in this case, so *the symbols are said to be multiplied by the bases raised to the position*. This is important, so make sure to ask questions about any part that is not clear.

So this is our review of the decimal number system for integers. Hopefully it is very familiar and perhaps even mundane. However, it is important that we understand this before we move on to the other number systems.

BINARY (Base-2)

The binary number system is the one used by computers. In this system, we have just two symbols – 0 and 1 – and, like decimal, we form larger numbers by grouping these symbols. Since we know that the position of the symbol denotes the symbol value (0 or 1 for binary) times the base raised to the number of the position, we find that for binary, we really want to know our powers of 2. Seriously, we really want to know our powers of 2. We can always get the next value for a power of 2 by multiplying the previous value by 2, but knowing a few of them from memory is very valuable.

Here is a short table of the values of 2 with an exponent ranging from 0 to 10. Note that the exponent is just the position of the symbol when counting from the right.

Position	**Binary Representation**	**Decimal Value**
0	1	1
1	10	2
2	100	4
3	1000	8
4	10000	16
5	100000	32
6	1000000	64
7	10000000	128
8	100000000	256
9	1000000000	512
10	10000000000	1024

We can see that, for example, the decimal number 4 has a binary representation of 100 which, using our positional notation, becomes $(0 \times 2^0) + (0 \times 2^1) + (1 \times 2^2) = 0 + 0 + 4 = 4$.

Now we will work several examples, converting decimal numbers to binary numbers and binary numbers to decimal numbers. The process (algorithm) that we will use is much simpler than for the general case and only works for integers and the bases mentioned here. But that is okay, as we will only be converting integers for now.

Example 3

The decimal number 2048 written in binary is 100000000000. How do I know this? Well, here is the trick, oops, algorithm.

The first thing that we way to do is subtract the largest power of two that is less than, or equal to, our decimal number. Our table has values up to decimal 1024, so let’s start there.

2048 - 1024 = 1024 **huh?**

We can see that 1024 subtracted from 2048 gave us back 1024, which is a power of 2! This means that 2048 is twice 1024. In fact, it is 2^{11} exactly! We can see this by multiplying the value in position 10 (1024) by 2, which moves us to position 11.

Thus, 2048 is represented in binary as a ‘1’ followed by 10 ‘0’s.

Example 4

Let’s convert the decimal number 53 to binary. We will use the same algorithm as before, repeating it until we have the number fully converted to binary.

The largest power of two that is still less than (or equal to) 53 is 32. From our table, we know that decimal 32 is binary 100000. 53-32 = 21, so let’s repeat the process with the largest power of 2 less than 21, which we can see is 16. From our table, we know that decimal 16 is binary 10000. 21-16 = 5. Repeating, we see that 4 is the largest power of 2 less than 5 and decimal 4 is binary 100. 5-4 = 1. We see immediately that the number 1 is a power of two, namely two raised to the zero power, and its binary representation is 1. Now, let’s add all these binary numbers.

32	100000
+16	010000
+4	000100
+1	000001
= 53	110101

Example 5

Let’s verify that decimal 110101 really is 53. To do this, we just reverse our algorithm.

Looking at our table, we see that we have 1 of position zero, 0 of position 1, 1 of position 2, 0 of position 3, 1 of position 4, and 1 of position 5. This becomes:

$(1 \text{ x } 2^{0)} + (0 \text{ x } 2^1) + (1 \text{ x } 2^2) + (0 \text{ x } 2^3) + (1 \text{ x } 2^4) + (1 \text{ x } 2^5) =$
$(1 \text{ x } 1) + (0 \text{ x } 2) + (1 \text{ x } 4) + (0 \text{ x } 8) + (1 \text{ x } 16) + (1 \text{ x } 32) =$
1+ 0 + 4 + 0 +16 + 32 =
53.

We really want to know how to convert from decimal to binary and back again. For our purposes, this is a useful way (well, it’s the way that I use) to convert between all the different number bases discussed here.

A Little About Bits and Bytes

While computers use binary, it is really inconvenient to just look at data 1 bit (binary digit) at a time. As such, we have a couple of groupings that we will use in this class. Please note that there are many more groupings possible, but you will not need to know them for this class.

So, a 'bit' is a binary digit. It can be a '0' or a '1'. That's it.

But sometimes, we want to group bits for ease of the humans reading the value or for operations inside the computer. The most common grouping is called a 'byte'. A byte consists of eight bits and the positions of the binary digits are numbered 0 through 7 within the byte. This means that a byte may have a value ranging from 0 to 255. We can see the 255 by recognizing that the largest binary value in a byte is 1111 1111 (we often write binary values in groups of 4 bits). This value in decimal is 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255. It is handy to note that the maximum value of a byte is one less than the value of the position just bigger than the byte. Since the largest position in a byte is '7', if we take 2^8 (the next position) and subtract one, we get 255, which is the maximum value of a byte. All bytes are 8 binary digits long.

We sometimes want to consider a 4 bit number. For historical reasons (and I've been told it is based on a joke), we call a 4 bit value a 'nibble'. Use of nibble values becomes important when we consider hexadecimal numbers below.

Sometimes we want a larger grouping than just a byte, so we have a 'word' which is 2 bytes long. That is, 16 binary digits. There are other groupings used in computer science, but they are all multiples of a byte, so we'll leave them out for now.

So, a bit is one binary digit. A nibble is 4 bits. A byte is eight bits. A word is two bytes or 16 bits.

Now let's get back to our number systems.

OCTAL (Base-8)

Octal is a number system that isn't used a great deal anymore, except in certain specialized applications, such as one of the computers in a car. But since we traditionally learn octal and the conversion is straightforward, we'll look at octal before we get to hexadecimal.

In this system, we have eight symbols – 0, 1, 2, 3, 4, 5, 6, 7 – and we form larger numbers by grouping these symbols just as we did in both decimal and binary. Converting directly between decimal and octal can be difficult without a calculator, so we are going to use a different method, or algorithm, that I hope will make things easier.

It helps at this point to know that 8 is a power of 2, in fact it is 2^3. This means that we need 3 binary digits to represent one octal digit. We can see that the binary number '111' is (1×2^0) +

$(1 \times 2^1) + (1 \times 2^2) = 1 + 2 + 4 = 7$, which is the largest octal value for a single symbol position. We can see that converting from binary to octal is very straightforward.

Our algorithm for converting a number to octal from decimal will have two steps: 1) convert the decimal number to its binary representation; and 2) convert the binary representation to octal representation. Our algorithm for converting from octal to decimal will likewise have two steps: 1) convert the octal number to its binary representation; and 2) convert the binary representation to its decimal representation.

Example 6

Let's convert the decimal number 53 to octal.

Step 1: convert the decimal representation to binary. We already did this in example 4, so we will use that algorithm and get 110101.

Step 2: convert the binary representation to octal representation. We know that to do this, we group the bits in groups of three (from the right) and convert these grouped numbers to octal.

So we group '110101' as '110' and '101'. We see that the first grouping is '6' and the second grouping is '5', which gives us 65 as the octal representation of the decimal number 53.

Example 7

Let's convert the octal number 122 to decimal.

Step 1: convert the octal number to binary representation. We know that each octal digit takes 3 binary digits. So for 122 octal, the '1' becomes '001', and each of the '2's become '010'. So 122 octal is now 001010010.

Step 2: convert the binary representation to decimal. Once again, we will just add the powers of 2 (remember our table?) and we get (from the right):

0 + 2 + 0 + 0 + 16 + 0 + 64 + 0 = 82 decimal.

HEXADECIMAL (Base-16)

Hexadecimal (usually called just 'hex') is a number system that has frequent use in computer science. It is very useful to be able to quickly convert from hex to decimal.

In this system, we have sixteen symbols – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F – and we form larger numbers by grouping these symbols just as we did in decimal, binary and octal. Notice that we started using alphabetic characters to represent a hex number. This is because we use a positional notation and since we have 16 possible values for a hex number (0 – 15), we need 16 single symbols to use. This can be confusing at first, but the more you use this notation, the more comfortable you will be with hex numbers. So the decimal number '10' is represented in hex as 'A' and the decimal number '15' is represented in hex as 'F'. The others should be obvious (I hope!). We can also see that the hexadecimal values for a byte range from 0 to FF. Please note that hex values are represented with the tag ‘0x’ prepended, so the value ‘FF’ is written as ‘0xFF’ or perhaps 0XFF.

We know (remember the table, above?) that 16 is 2^4, which means that each hex number has 4 binary digits. So one hex number is four bits long. We can see that the binary number ‘1111’ is $(1 \times 2^0) + (1 \times 2^1) + (1 \times 2^2) + (1 \times 2^3) = 1 + 2 + 4 + 8 = F$, which is the largest hex value for a single symbol position. We can see that converting from binary to hexadecimal is very straightforward.

Our algorithm for converting a number to hex from decimal will have two steps: 1) convert the decimal number to its binary representation; and 2) convert the binary representation to hex representation. Our algorithm for converting from hex to decimal will likewise have two steps: 1) convert the hex number to its binary representation; and 2) convert the binary representation to its decimal representation.

Example 8

Let’s convert the decimal number 53 to hex.

Step 1: convert the decimal representation to binary. We already did this in example 4, so we will use that algorithm and get 110101.

Step 2: convert the binary representation to hex representation. We know that to do this, we group the bits in groups of four (from the right) and convert these grouped numbers to hex.

So we group ‘110101’ as ‘11’ and ‘0101’. Now, the first grouping only has 2 binary digits, but that’s okay because we can always prepend a couple of zeroes without changing the value of the number. So we now have two 4-bit groupings: ‘0011’ and ‘0101’. We see that the first grouping is ‘3’ and the second grouping is ‘5’ which gives us 0x35 as the hex representation of the decimal number 53.

Example 9

Let’s convert the hex number 0xF7 to decimal.

Step 1: convert the hex number to binary representation. We know that each hex digit takes 4 binary digits. So for 0xF7, the ‘F’ becomes ‘1111’ and the ‘7’ becomes ‘0111’. So 0xF7 becomes 11110111. For ease of reading, we usually group longer binary strings as nibbles (see above) with a space in between each nibble, so 11110111 is usually written as 1111 0111. At this point, it is convenient to note that a single hex symbol occupies a single nibble and so each byte contains two hex digits.

Step 2: convert the binary representation to decimal. Once again, we will just add the powers of 2 (remember our table?) and we get (from the right):

1 + 2 + 4 + 0 + 16 + 32 + 64 + 128 = 247 decimal.

To Sum Up

Our method for converting between all these different number systems requires us to use the binary representation. For conversion to and from both octal and hexadecimal, the binary representation is an intermediate representation and then we move on to our final representation. I've found this technique to be easy to remember and easy to use in over 35 years of computer programming and I hope that you do as well.

You will be required to convert numbers among these representations, but if you find a technique that you like better, you are free to use it. Well, except for using a calculator.