# Binary Numbers – Base 2

In the [Decima](https://en.wikipedia.org/wiki/Decimal)​ [l](https://en.wikipedia.org/wiki/Decimal) number system(base 10) we represent numbers with the digits 0,1,2,3,4,5,6,7,8 and 9, in​ Binary (base 2) we represent numbers using ,only, the digits 0 and 1 The 0 and 1 can represent the state of a [‘switch](https://plus.maths.org/content/snakes-and-adders)​ [’](https://plus.maths.org/content/snakes-and-adders) ​ or memory address within a computer’s memory or CPU; they could also represent the True and False values used for [Boolea](https://en.wikipedia.org/wiki/Boolean_data_type)​ [n](https://en.wikipedia.org/wiki/Boolean_data_type) ​ (logical) operations.

The common number systems are used in computer science is:

* Base 2 (binary)
* Base 8 (octal)
* Base 10 (denary)
* Base 16 (Hexadecimal)

You probably will not encounter binary much these days, but it's useful to understand that this is how computers work internally, so you can understand the basic concept of how computers communicate. The fact that computers use binary is why everything is a multiple of 2 - why computers come with 8Mb, 16Mb, 32Mb, 64Mb, etc., of memory, rather than 10Mb, 20Mb, 30Mb, etc., and also why there are 1024 bytes in a Kilobyte (1024 = 210​ ​) rather than [100](https://physics.nist.gov/cuu/Units/binary.html)​ [0](https://physics.nist.gov/cuu/Units/binary.html) bytes.​

There are some other binary-related terms you'll need to know. Firstly, a *bit*​ ​ is a *binary*​ *digit*​ - i.e. a single occurrence of 0 or 1. This is the smallest unit of storage you can have inside a computer. Groups of 8 bits are called bytes. A byte can be used to represent a number, or a colour, or a character (e.g. using [ASCI](https://en.wikipedia.org/wiki/ASCII)​ [I)](https://en.wikipedia.org/wiki/ASCII)​ . You may also hear the term nibble, which is 4 bits. Finally, a word is the largest number of bits that a processor can handle in one go - for example, when we say that new computers have 64-bit processors, we mean that the word length is 64-bits, or 8 bytes.

The largest value that you can store using a particular number of bits can be determined quite easily. Using n bits, the largest value you can store is 2n​ ​ - 1, and the number of different values you can store is 2n​ ​ (from 1 to 2n​ ​ - 1, and then 0 as well). So using 8 bits, the largest number you can store is 28​ ​ - 1 = 255, and the number of possible values is 28​ ​ = 256 (i.e. 0 - 255). A 32-bit computer can therefore handle values up to 4,194,967,296 in one clock cycle of the CPU - it can obviously cope with larger numbers, but it would need to split them up first.

# Converting between Binary and Decimal

Conversion between the two number systems is simple, involving only addition, subtraction, and rarely multiplication or division, all based on simple rules. If we use units, tens, hundreds, thousands and so on (powers of 10) to represent decimal numbers we use a similar system for binary. Look at the table below:

LSB = Least significant bit

MSB = Most significant bit

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | MSB |  | 8 bits – 1 byte (binary digits 1 or 0) | | | |  | LSB |
| Decimal  Equivalent | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Binary | 1 or 0 | 1 or 0 | 1 or 0 | 1 or 0 | 1 or 0 | 1 or 0 | 1 or 0 | 1 or 0 |

The top row shows the corresponding decimal values to the possible binary digits in the bottom row.

These are arranged as powers of 2 so if we continue this to cover more than 8 bits we would go on to 256 then 512 and so on. Each successive number doubles as you move to the left . E.g. A 9th​ ​ bit would represented by a doubling of the 8th​ ​ bit – 128 x 2 = 256

To convert from binary to decimal we place our actual binary digits in the bottom row, then we can add together the corresponding decimal values and the result is our binary number converted to decimal; see the example below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 8 bits – 1 byte (binary digits 1 or 0) | | | |  |  |  |
| 128 | 64 | 32 | 16 | 8 |  | 4 | 2 | 1 |
| 1 | 0 | 1 | 0 | 0 |  | 1 | 0 | 1 |

.˙. 10100101 = 128 + 32 + 4 + 1 = 165

To perform the conversion from decimal to binary we need to start by subtracting the largest of the corresponding decimal values that we can, then mark this bit as a 1, then find the next largest value that can be subtracted from the remainder and subtract this and mark the corresponding value as 1 again, and continue this process until there is no remainder. So, for example:

243 - 128 = 115, 115 - 64 = 51, 51 - 32 = 19, 19 - 16 = 3, 3 - 2 = 1, 1 - 1 = 0 Which gives you the binary code below 11110011 representing 243

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 8 bits (binary digits 1 or 0) | |  |  |  |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

# Binary Addition

Binary addition works in the same way as decimal addition, but there are only 2 digits in base 2 so when a value is greater than 1 we must carry a bit. This gives the following possible combinations:

* 0 + 0 = 0
* 0 + 1 = 1
* 1 + 0 = 1
* 1 + 1 = 10 ( 0 down carry 1)
* 1 + 1 + 1 = 11 (1 down carry 1)

In many ways this is simpler than working with decimal numbers.

Take a look at the example below to see how this works in practise:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| top row: |  |  | 1 | 1 | 0 | 1 | 0 | 1 |
| bottom row: |  | + | 1 | 0 | 1 | 1 | 1 | 0 |
| carry bits: |  | 1 | 1 | 1 | 1 |  |  |  |
| result: |  | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

It should be noted that things become more complicated because all other arithmetic in binary is based on addition (division, multiplication and subtraction all use addition and bit shifting).

# Bit Shifting

Bit shifting is the process of moving bits to the left or right as a number is multiplied or divided respectively by a power of 2. So if we multiply a binary number by 2 we move it 1 place to the left and append a 0 to the end (LSB) if necessary. If we multiply a binary number by 8 we move it 3 places to the left (8 is 2^3 ( 2 to the power 3)); and so on. Look at the example below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***multiply by*** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0010 (2^1) | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0100 (2^2) | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1000 (2^3) | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

The same principal can be applied to division but this time the bits are shifted to the right as in the example below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***divide by*** | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0010 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0100 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1000 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

Multiplication and division using more complex numbers requires us to then use bit shifting and addition or subtraction (subtraction being addition with 2’s compliment, but more on this later).

So here is an example of multiplication in binary:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | ***B*** | ***A*** |  |
| top row: |  |  |  | 1 | 1 | 1 | 0 | 1 |
| bottom row: |  | \* |  |  |  | 1 | 1 | 0 |
| bit shift A |  |  | 1 | 1 | 1 | 0 | 1 | 0 |
| bit shift B | + | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| carry bits: |  | 1 | 1 |  |  |  |  |  |
| result: | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

**A**​ and ​**B**​ have been used to denote the bits in the bottom row – the number we are multiplying by. A represents the 1​ in the bottom row and B represents the ​ 1​ in the bottom row. The top row is shifted 1​ place to the left – bit shift A (2^1), then shifted 2 places to the left – bit shift B (2^2) Then the bit shift A and bit shift B are added together to give the result of 174 in decimal, as 6 (bottom row) x 29 (top row) =

6 (2^3) x 29 = 174

For performing divisions using numbers with more than 1 bit set to 1 it is necessary to use a form of long division under base 2, which in turn requires the use of subtraction in binary.

Subtraction in binary can be very confusing as it is necessary to convert the value being subtracted to 2’ compliment, which is discussed next

# 2’s Compliment

2’s compliment is a system used to represent negative numbers in binary as an inverted value with an additional sign or carry bit. A sign bit is a bit appended to the left of a binary number that indicates whether the number is positive (0) or negative (1) and in 2’s compliment equates to the carry bit introduced in the second stage of the conversion. The first stage is to convert to 1’s compliment (a simple inversion); in 1’s compliment all the digits are inverted so all 1s become 0s and all 0s become 1s as in the example below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| number | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1’s compliment | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

To convert to 2’s compliment we then add 1 (the carry bit or sign bit) as in the example below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| number | 1 | 0 | 1 | | 1 | 0 | 1 | 0 | 1 |
| 1’s compliment | 0 | 1 | 0 | | 0 | 1 | 0 | 1 | 0 |
| 2’s compliment | 0 | 1 | | 0 | 0 | 1 | 0 | 1 | 1 |

The 1 is actually added and not just appended to the end so here is the final 2’s complimented number (you can see how the bit that is added will be carried through):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| number | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1’s compliment | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 2’s compliment | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

# Binary Subtraction

In order to perform subtraction in binary first append 0s to the left of the number we are subtracting until it has at least the same number of digits as the number we are subtracting from (if it is already ‘longer’ – consists of more bits then we need not do this). We then convert the number we are subtracting to 2’s compliment and add it to the other number. Finally we remove the left most bit (the carry bit). Take a look at the example below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| top row: |  |  |  | 1 | 1 | 1 | 0 | 1 |
| bottom row: |  | - |  | 0 | 0 | 1 | 1 | 0 |
| 1’s compliment: |  |  |  | 1 | 1 | 0 | 0 | 1 |
| 2’s compliment: |  |  |  | 1 | 1 | 0 | 1 | 0 |
| now add: |  |  |  | 1 | 1 | 1 | 0 | 1 |
|  |  | + |  | 1 | 1 | 0 | 1 | 0 |
| carry bits: |  |  |  | 1 |  |  |  |  |
| initial result: |  |  | 1 | 1 | 0 | 1 | 1 | 1 |
| remove the carry bit | | | | | |  |  |  |
| result: |  |  |  | 1 | 0 | 1 | 1 | 1 |

This is fine if we are subtracting a smaller number from a larger number but if we subtract a larger number from a smaller number the result would be negative so we need to add some additional stages. After the initial result, we leave the carry bit in place and we convert the initial result again using 2’s compliment; we then sign the result as negative. You must continue to use the same number of binary digits throughout for the number being subtracted to ensure no bits are lost. Take a look at the example on the next page

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| top row: |  |  |  | 0 | 0 | 1 | 1 | 0 |
| bottom row: |  | - |  | 1 | 1 | 1 | 0 | 1 |
| 1’s compliment: |  |  |  | 0 | 0 | 0 | 1 | 0 |
| 2’s compliment: |  |  |  | 0 | 0 | 0 | 1 | 1 |
| now add: |  |  |  | 0 | 0 | 1 | 1 | 0 |
|  |  | + |  | 0 | 0 | 0 | 1 | 1 |
| carry bits: |  |  |  |  |  | 1 |  |  |
| initial result: |  |  |  | 0 | 1 | 0 | 0 | 1 |
| convert initial result to 2’s compliment | | | | | | | |  |
| 1’s compliment |  |  |  | 1 | 0 | 1 | 1 | 0 |
| 2’s compliment |  |  |  | 1 | 0 | 1 | 1 | 1 |

the final result is currently showing as a positive number so we need to either use a sign bit or to indicate that it is negative in the traditional way, i.e. - 10111

# Binary in Networking

Binary is a crucial number system in networking where although addresses are usually presented as groups (4 groups in IPV4 of 3 decimal digits with a range of 0 to 255) the decimal values when converted to binary often provide information about masking of bits. With that in mind it is useful to practice converting between binary and decimal.

Binary values are also usually used to calculate file sizes, data transmission and storage requirements (as computers internally work entirely in binary) and the effect of this is that the definitions of ‘scales’ of memory or data are based around multiples of 10 bits, which approximates to 1000 in decimal but is in fact 1024. So for example 1KB is 1024 bytes, 1MB is 1024KB and so on. This is important to note as the difference in the number of bytes becomes much greater the larger the quantities of data are if working between a decimal system and a binary system. For consistency unless otherwise stated the convention is to work within the binary system.

# Converting Binary to Hexadecimal

This may seem complex but in practice it is relatively simple thanks to a relationship between the two number systems in that the bases are both powers of 2. Binary is base 2 (2^1) while hexadecimal is base 16 (2^4). We can use that information to simplify the conversion knowing that 1 hexadecimal digit corresponds to 4 binary digits. The hexadecimal decimal system uses 16 digits 0 – 9 and A - F

Look at the table below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *binary* | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |
| *decimal* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| *hexadecimal* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| *binary* | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| *decimal* | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| *hexadecimal* | 8 | 9 | A | B | C | D | E | F |

You can also see here the corresponding decimal values. So knowing this we can then convert each hexadecimal digit into a group of 4 binary digits, which requires remembering only those 16 combinations as follows:

*x before a number signifies it’s a hexadecimal number.*

xAF7C would give us xA= 1010, xF = 1111, x7 = 0111, xC = 1100, therefore xAF7C = 1010111101111100

We should also be able to use this to simplify converting to decimal as follows:

xA = 10, xF = 15, x7 = 7 and xC = 12,

If we number each column that a hexadecimal digit is found in from right to left and starting at 0, we can then multiply the decimal value of each column by 16^ column number,

.˙. xAF7C = 10 \* 16^3 + 15 \* 16^2 + 7 \* 16^1 + 12 \* 16^0

= 10 \* 4096 + 15 \* 256 + 7 \* 16 + 12 \* 1 = 40960 + 3840 + 112 + 12 = 44,924

xAF7C = 1010111101111100 (bin) = 44924 (dec)

Arithmetic in hexadecimal can be difficult to work with so it is often better to convert to either binary or decimal first and then convert back once the answer is found. Of course there are other tools that can be used to help with this and you will find that most calculator applications offer either a scientific calculator that allows for working between number systems or a programmers calculator that will simultaneously work in binary, decimal and hexadecimal.

## Binary, Decimal and Hexadecimal Conversion Practise

Convert between Binary, Decimal and Hexadecimal to complete the table below:

|  |  |  |
| --- | --- | --- |
| ***Binary*** | ***Decimal*** | ***Hexadecimal*** |
|  | 5 |  |
| 111 |  |  |
|  |  | A |
| 1101 |  | D |
| 101010 |  |  |
|  | 27 |  |
|  |  | 1E |
| 100111 | 39 |  |
|  | 178 |  |
|  |  | F7 |
| 10101101 |  |  |
| 11001100 |  | CC |
|  |  | ACE |
| 1101011010 |  |  |
|  | 1271 |  |
| 101010101010 |  |  |
|  |  | 178F |
|  | 3175 |  |
| 101101111010 |  |  |
| 1101001101001111 |  | B34F |

**Reference**

**The small table below can be used for reference when completing the table above:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 | 6 | 7 | 7 |
| 0000 | | 0001 | | 0010 | | 0011 | | 0100 | | 0101 | | 0110 | | 0111 | |
| 8 | 8 | 9 | 9 | 10 | A | 11 | B | 12 | C | 13 | D | 14 | E | 15 | F |
| 1000 | | 1001 | | 1010 | | 1011 | | 1100 | | 1101 | | 1110 | | 1111 | |