|  |
| --- |
|  |
| Ryan’s Encyclopedia of Computer Science |
|  |

|  |
| --- |
| Ryan Krage  1-1-2017 |

Contents

[Binary 2](#_Toc481938412)

[Binary/denary conversion 2](#_Toc481938413)

[Binary arithmetic 3](#_Toc481938414)

[Addition 3](#_Toc481938415)

[Multiplication 3](#_Toc481938416)

[Two’s Compliment 4](#_Toc481938417)

[Subtraction 4](#_Toc481938418)

[Binary Fractions 5](#_Toc481938419)

[Fixed point 5](#_Toc481938420)

[Floating point 5](#_Toc481938421)

[Error 5](#_Toc481938422)

# Binary

Binary is the way that all computers store and manipulate information. It is a number system, like denary1, but instead of using tens, binary only has two numbers – 0 or 1. Technically, two of anything will work – on/off, true/false, red/purple – so long as there are two states. Computers use binary because they use electricity -and electricity can be on/off, making it ideal for binary. While using different voltages to have a more complex number system is possible, it tends to be impractical.  
Binary works just like denary, but instead of increasing in orders of ten, binary increases in orders of two.  
For example, with denary there are units (1-10), 10’s (10-100), hundreds (100-1000), thousands (1000-10000), tens of thousands (10000-100000) and so on. With binary, the ‘units’ are 0-1, and then because it uses a system of two, the next order of magnitude doubles and becomes 1-2, then 2-4, 4-16, 16-32, and so on.  
When written in binary, these would be 0-1, 1-01, 01-001, 001-0001, 0001-00001 and so on.  
In computer science, each 0 or 1 is called a ‘bit’. Eight bits make a byte, and 1000 bytes make a kilobyte, 1000 kilobytes make a megabyte, 1000 megabytes make a gigabyte, and so on. Technically, it’s 1024 of the previous unit to make the next one, but it is usually rounded to 1000 for simplicity. This is also the standard when measuring the storage capacity of a device.

## Binary/denary conversion

Converting between binary and denary is very easy – it just uses addition. Say you needed to convert the byte 00101101 into denary.  
First, label each column with its order of magnitude, like so;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 256 | 128 | 64 | 32 | 16 | 4 | 2 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

As you can see, the order of magnitude doubles each time. This makes it easy to convert longer binary numbers – just keep doubling as you move further left.  
Next, add up the columns that have a 1 in them – in this case 1 + 4 + 16 + 64 = 85. Ignore the columns with a 0.  
85 is the answer. 00101101 = 85.

This works for any number system – increase the order of magnitude of each column according to the name of the number system (binary is two, trinary is 3, denary is 10, hexadecimal is 16, etc) and then multiply the order of magnitude in each column by the number in that column, then add all the results.  
With binary, you can skip the multiplication step because the number is always 1, and 1 \* =

Here’s an example with denary.

|  |  |  |  |
| --- | --- | --- | --- |
| 1000 | 100 | 10 | 1 |
| 3 | 0 | 5 | 7 |

1 \* 7 = 7  
10 \* 5 = 50  
100 \* 0 = 0  
1000 \* 3 = 3000  
3000 + 0 + 50 + 7 = 3057

1 Often mistakenly called decimal. Decimal numbers have a decimal point.

To convert the other way, take any denary number, and then split into units of 1,2,4,8,16,32, etc.  
For smaller numbers, you can do this in your head. For example, working out that 12 is equal to 8 + 4 is trivial.  
For larger numbers, find the closest binary order of magnitude that is *smaller* than your number. For example, to convert 1345, start with 1024. Take that away from your target number (1345 – 1024) and then you are left with a much smaller number – 321. The closest to 321 is 256, so you know you need 1024 and 256 in your number. So far the binary number is 10100000000. Next is to convert 65 (321 – 256). The closest to 65 is 64, leaving us with 1 – which is just 1. That gives us 1024 + 256 + 64 + 1, or 101001000001.

## Binary arithmetic

There are two methods for multiplying, adding and subtracting binary numbers.  
The ‘cheat’ method is to convert the binary into denary, perform the calculation, and then convert the result into binary. For very small numbers, this may be quicker.  
For larger numbers, the conversion process takes longer than just doing the maths in binary.

### Addition

When adding bits together, there are four rules;  
0 + 0 = 0  
1 + 0 = 1 *or* 0 + 1 = 1  
1 + 1 = 0 carry 1  
1 + 1 + 1 = 1 carry 1

carry 1 means ‘write in this column and add 1 to the next column’. Binary addition is done right-to-left, starting with the smallest bit.  
0110 + 0101. Going column by column, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0 carry 1, 0 + 1 = 1, giving a final answer of 1101.  
When writing out the calculation, use the ‘column method’, which is writing each number below the previous one, like so;  
 0110  
+ 0101  
+ 1  
----------  
 1011

### Multiplication

There is simple step-by-step method for binary multiplication.

0110 \* 0101

1. Write the smaller number (0101) vertically, like so;

0  
1

0

1

1. Multiply each number by the larger number, and then by an order of ten descending from the top.  
   0 \* 0110 \* 1000

1 \* 0110 \* 100  
0 \* 0110 \* 10  
1 \* 0110 \* 0

Multiplication by 10 works the same way as denary – add the corresponding number of zero’s to the end.  
Ignore any row starting with 0 and the last row – they will just be 0.

1. Add the results. The columns may not line up in the working below, due to the use of a non-monospaced font. The row for carrying is not shown.  
    0110000

+ 01100

= 111100

## Two’s Compliment

Two’s Compliment is a way of storing negative binary numbers.  
The easiest way to understand two’s Compliment is to think of it like a car milometer. Once it reaches the maximum number, it rolls over and starts again at 0. Two’s Compliment works in a similar way. However, in order to make it clear whether the number is positive or negative, there are a few differences.  
The largest 4-bit binary number is 1111. In Two’s Compliment, the largest positive binary number is 0111, and 1111 would be the smallest negative number. This is because of the way that negative numbers are converted in order to be read.  
If a number starts with 1, then it is negative. If it starts with 0, it is positive, and no conversion is needed.  
To convert a negative number, the rule is ‘flip the bits and add 1’. This means swap every 1 for a 0, and vice-versa, and then add 1 to the result (by addition, not appending 1). The result is the actual negative number.  
As an example, 1010 becomes 0101, then 0110 when 1 is added. 0110 is 6, so 1010 is -6.  
The opposite can be done to work out how to write negative binary number. In order to write -5 in binary, first convert +5 to binary – 0101. Then, take away one and flip the bits, to make 1011, 1011 is -5 in Two’s Compliment binary.  
Not all binary uses Two’s Compliment – it is only if specified.

### Subtraction

Subtraction uses the same rules as addition, but there is an extra step. The number that is being taken away should be made into a Two’s Compliment negative number, and then added to the other number.  
10-4 would be 1010 – 0100. However, 0100 needs to be converted, so it becomes 1100. The 10 also needs to be converted to a positive Two’s Compliment number by appending a 0 onto the left. The last carry is ignored

01010  
+ 1100  
----------  
 0110 = 6

## Binary Fractions

While in denary each number after the decimal place is a tenth of the number before it, in binary each number is half of the one before it.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | . | 1 | 0 | 1 | 1 |
| 8 | 4 | 2 | 1 | . | 1/2 | 1/4 | 1/8 | 1/16 |

Conversion works the same way – just add up every number with a ‘1’ above it. In this case, the number is 11.9375, or 11 and 11/16.  
A shortcut to adding up the fractions is to start with smallest number (in the example above, 1/16). Then, each number closer to the decimal point will be double that fraction – so 1/8is 2/16, and 1/4 is 4/16, and so on.  
The position of the point is specified by the type of binary fraction – fixed or floating.

### Fixed point

In fixed point, the position of the point is the same in every number (it is ‘fixed’ in place). Having more digits after the point allows for greater precision, while having more digits before the decimal point allows for a greater range of numbers.

### Floating point

In floating point, the position of the decimal point can be different in every number.  
To specify the position of the point, the number is split into two parts – mantissa and exponent. The mantissa is the actual number, the exponent specifies the position of the decimal point within the mantissa. The decimal point always starts after the first digit. The length of each part will be specified, and may be denoted with a gap or space in the binary number.  
The mantissa and exponent have separate Two’s Compliment, unless otherwise specified.  
To convert a floating point number into decimal, first convert the exponent using Two’s Compliment. If the exponent is positive, move the decimal point the specified number to the right. If it is negative, move it left. This generally means that negative exponents are for small but precise numbers, and positive exponents result in larger but less precise numbers.

### Error

While binary fractions allow for greater precision, they are not always precise enough to represent a number completely accurately. This results in two types of error.

#### Absolute error

This is the difference between the real number and the inaccurate binary representation.  
Real number: 0.0078025  
Desired number: 0.0078100  
absolute error: 0.0000125

#### Relative error

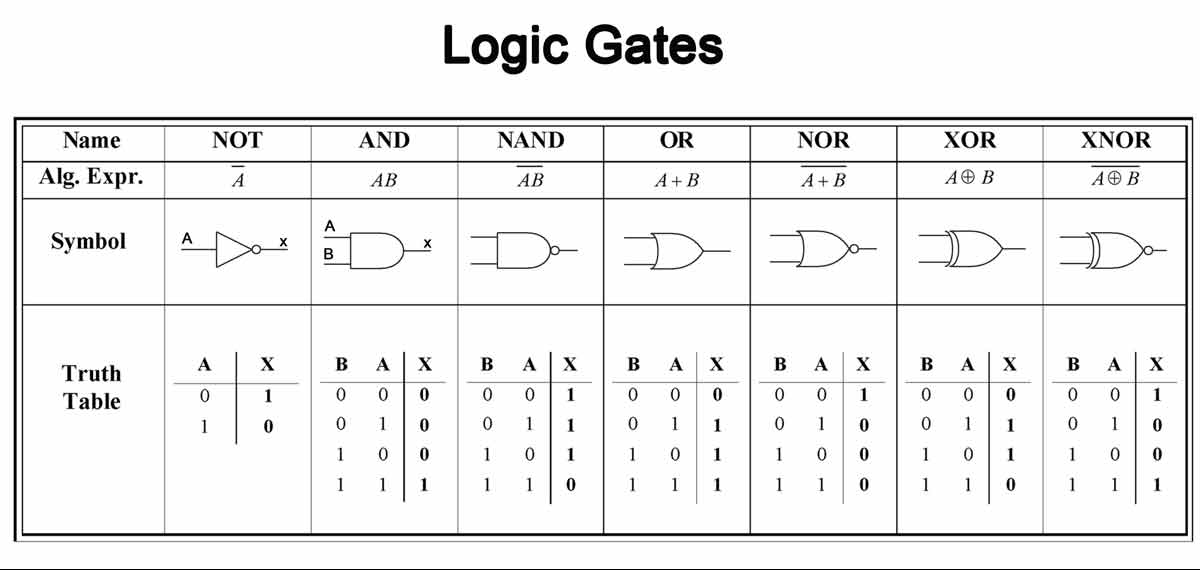
Relative error is the absolute error divided by the desired number. This shows how large the margin of error is.  
0.0000125/0.0078025 = 0.0016020506247997

# Logic

Logic is often used casually to mean ‘straightforward’ or ‘common sense’, but in computer science it refers to the method of manipulating binary data. Logic that deals with other number systems (i.e, more than two states) is called ‘fuzzy logic’ and is used in A.I and machine learning research.

## Logic gates

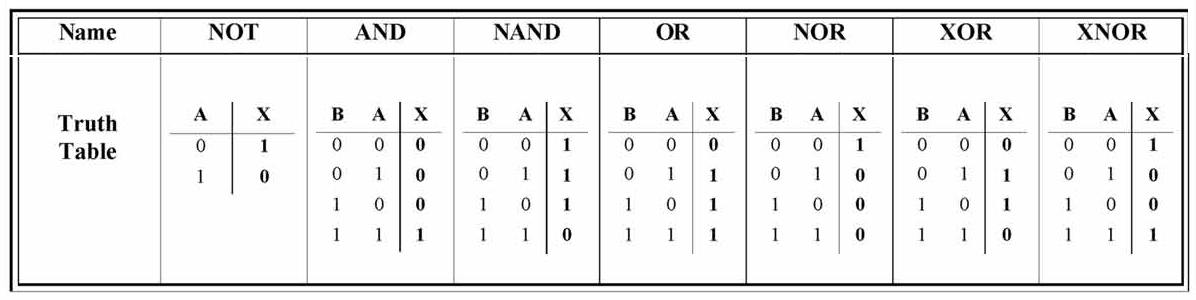
The most fundamental aspect of logic is logical operations. These are represented by logic gates. In the real world, they carry out a logical operation on a signal, and are also useful for representing sets of logical operations. When discussing gates, the terms ‘True’ and ‘False’ are used to mean 1 and 0, to distinguish it from binary.  
There are 7 different logic gates. Each has their own symbol, used to represent the gates in circuit diagrams.



AND, NOT and OR are the simplest and most common. In circuits, NAND is used exclusively as every other type of logic gate can be made using circuits of NAND gates.  
NOT takes a single input and reverses it – if True goes in, 0 comes and vice-versa.  
AND will only output True if it receives two True inputs. Otherwise, it will output False.  
OR will output True if *either* of the two inputs are True. It will also accept two True inputs.  
NAND (meaning NOT AND) will only output True if both inputs are NOT True.  
NOR (NOT OR) will output True if *neither* input is True.  
XOR (eXclusive OR) works just like OR, but does not output True if *both* inputs are True at the same time.  
XNOR (eXclusive NOT OR) will output False if one input is True, but will output True if both or neither inputs are True.

### Truth Tables

Truth tables show every combination of inputs and their corresponding output for a logic gate.



They can do the same for combinations of logic gates too. Each input gets a corresponding letter. Inputs that are the outputs of previous gates get their own column. X or Q are usually used to denote the final output.