Binary number

From Wikipedia, the free encyclopedia

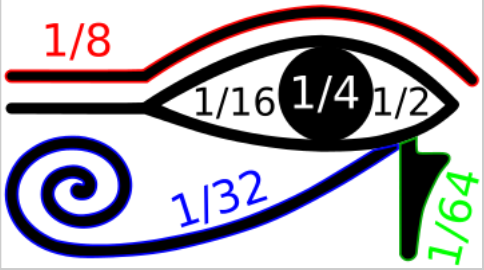
In mathematics and digital electronics, a **binary number** is a number expressed in the **binary numeral system** or **base-2 numeral system** which represents numeric values using two different symbols: typically 0 (zero) and 1 (one). The base-2 system is a positional notation with a radix of 2. Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used internally by almost all modern computers and computer-based devices. Each digit is referred to as a bit.

Contents

History

The modern binary number system was devised by Gottfried Leibniz in 1679 and appears in his article *Explication de l'Arithmétique Binaire* (published in 1703). Systems related to binary numbers have appeared earlier in multiple cultures including ancient Egypt, China, and India. Leibniz was specifically inspired by the Chinese I Ching.

Egypt



Arithmetic values represented by parts of the Eye of Horus

The scribes of ancient Egypt used two different systems for their fractions, Egyptian fractions (not related to the binary number system) and Horus-Eye fractions (so called because many historians of mathematics believe that the symbols used for this system could be arranged to form the eye of Horus, although this has been disputed). Horus-Eye fractions are a binary numbering system for fractional quantities of grain, liquids, or other measures, in which a fraction of a hekat is expressed as a sum of the binary fractions 1/2, 1/4, 1/8, 1/16, 1/32, and 1/64. Early forms of this system can be found in documents from the Fifth Dynasty of Egypt, approximately 2400 BC, and its fully developed hieroglyphic form dates to the Nineteenth Dynasty of Egypt, approximately 1200 BC.[1]

The method used for ancient Egyptian multiplication is also closely related to binary numbers. In this method, multiplying one number by a second is performed by a sequence of steps in which a value (initially the first of the two numbers) is either doubled or has the first number added back into it; the order in which these steps are to be performed is given by the binary representation of the second number. This method can be seen in use, for instance, in the Rhind Mathematical Papyrus, which dates to around 1650 BC.[2]

China

The I Ching dates from the 9th century BC in China.[3] The binary notation in the *I Ching* is used to interpret its quaternary divination technique.[4]

Daoist Bagua

It is based on taoistic duality of yin and yang.[5] eight trigrams (Bagua) and a set of 64 hexagrams ("sixty-four" gua), analogous to the three-bit and six-bit binary numerals, were in use at least as early as the Zhou Dynasty of ancient China.[3]

The Song Dynasty scholar Shao Yong (1011–1077) rearranged the hexagrams in a format that resembles modern binary numbers, although he did not intend his arrangement to be used mathematically.[4] Viewing the least significant bit on top of single hexagrams in Shao Yong's square (http://www.biroco.com/yijing/sequence.htm) and reading along rows either from bottom right to top left with solid lines as 0 and broken lines as 1 or from top left to bottom right with solid lines as 1 and broken lines as 0 hexagrams can be interpreted as sequence from 0 to 63. [6]

India

The Indian scholar Pingala (c. 2nd century BC) developed a binary system for describing prosody.[7][8] He used binary numbers in the form of short and long syllables (the latter equal in length to two short syllables), making it similar to Morse code.[9][10] Pingala's Hindu classic titled Chandaḥśāstra (8.23) describes the formation of a matrix in order to give a unique value to each meter. The binary representations in Pingala's system increases towards the right, and not to the left like in the binary numbers of the modern, Western positional notation.[11][12]

Other cultures

The residents of the island of Mangareva in French Polynesia were using a hybrid binary-decimal system before 1450.[13] Slit drums with binary tones are used to encode messages across Africa and Asia.[5] Sets of binary combinations similar to the I Ching have also been used in traditional African divination systems such as Ifá as well as in medieval Western geomancy. The base-2 system utilized in geomancy had long been widely applied in sub-Saharan Africa.

Western predecessors to Leibniz

In 1605 Francis Bacon discussed a system whereby letters of the alphabet could be reduced to sequences of binary digits, which could then be encoded as scarcely visible variations in the font in any random text.[14] Importantly for the general theory of binary encoding, he added that this method could be used with any objects at all: "provided those objects be capable of a twofold difference only; as by Bells, by Trumpets, by Lights and Torches, by the report of Muskets, and any instruments of like nature".[14] (See Bacon's cipher.)

Leibniz and the I Ching

The full title of Leibniz's article is translated into English as the *"Explanation of Binary Arithmetic, which uses only the characters 1 and 0, with some remarks on its usefulness, and on the light it throws on the ancient Chinese figures of Fu Xi"*.[15] (1703). Leibniz's system uses 0 and 1, like the modern binary numeral system. An example of Leibniz's binary numeral system is as follows:[15]

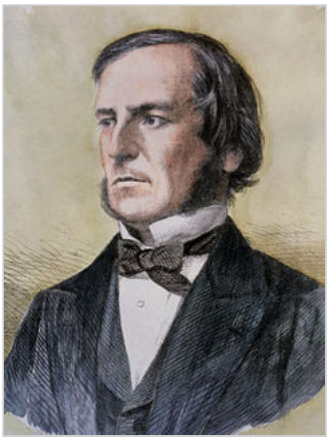
0 0 0 1	numerical value 2^0
0 0 1 0	numerical value 2^1
0 1 0 0	numerical value 2^2
1 0 0 0	numerical value 2^3


Gottfried Leibniz

Leibniz interpreted the hexagrams of the I Ching as evidence of binary calculus.[16] As a Sinophile, Leibniz was aware of the I Ching, noted with fascination how its hexagrams correspond to the binary numbers from 0 to 111111, and concluded that this mapping was evidence of major Chinese accomplishments in the sort of philosophical mathematics he admired.[17] Leibniz was first introduced to the *I Ching* through his contact with the French Jesuit Joachim Bouvet, who visited China in 1685 as a missionary. Leibniz saw the *I Ching* hexagrams as an affirmation of the universality of his own religious beliefs as a Christian.[16] Binary numerals were central to Leibniz's theology. He believed that binary numbers were symbolic of the Christian idea of *creatio ex nihilo* or creation out of nothing.[18]

> [A concept that] is not easy to impart to the pagans, is the creation *ex nihilo* through God's almighty power. Now one can say that nothing in the world can better present and demonstrate this power than the origin of numbers, as it is presented here through the simple and unadorned presentation of One and Zero or Nothing.
>
> — Leibniz's letter to the Duke of Brunswick attached with the *I Ching* hexagrams[16]

Later developments


George Boole

In 1854, British mathematician George Boole published a landmark paper detailing an algebraic system of logic that would become known as Boolean algebra. His logical calculus was to become instrumental in the design of digital electronic circuitry.[19]

In 1937, Claude Shannon produced his master's thesis at MIT that implemented Boolean algebra and binary arithmetic using electronic relays and switches for the first time in history. Entitled *A Symbolic Analysis of Relay and Switching Circuits*, Shannon's thesis essentially founded practical digital circuit design.[20]

In November 1937, George Stibitz, then working at Bell Labs, completed a relay-based computer he dubbed the "Model K" (for "**K**itchen", where he had assembled it), which calculated using binary addition.[21] Bell Labs authorized a full research program in late 1938 with Stibitz at the helm. Their Complex Number Computer, completed 8 January 1940, was able to calculate complex numbers. In a demonstration to the American Mathematical Society conference at Dartmouth College on 11 September 1940, Stibitz was able to send the Complex Number Calculator remote commands over telephone lines by a teletype. It was the first computing machine ever used remotely over a phone line. Some participants of the conference who witnessed the demonstration were John von Neumann, John Mauchly and Norbert Wiener, who wrote about it in his memoirs.[22][23][24]

The Z1 computer, which was designed and built by Konrad Zuse between 1935 and 1938, used Boolean logic and binary floating point numbers.[25]

Representation

Any number can be represented by any sequence of bits (binary digits), which in turn may be represented by any mechanism capable of being in two mutually exclusive states. Any of the following rows of symbols can be interpreted as the binary numeric value of 667:

```
1 0 1 0 0 1 1 0 1 1
| — | — — | | — | |
☒ ☐ ☒ ☐ ☐ ☒ ☒ ☐ ☒ ☒
y n y n n y y n y y
```

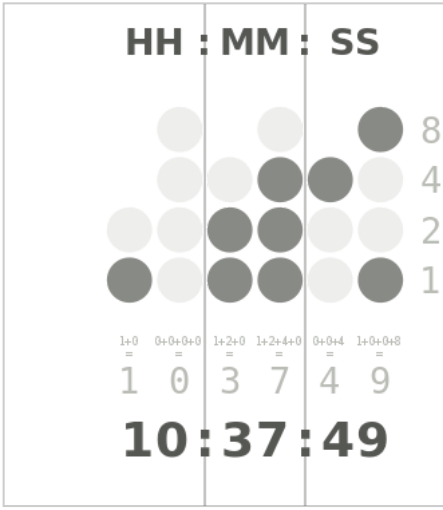
The numeric value represented in each case is dependent upon the value assigned to each symbol. In a computer, the numeric values may be represented by two different voltages; on a magnetic disk, magnetic polarities may be used. A "positive", "yes", or "on" state is not necessarily equivalent to the numerical value of one; it depends on the architecture in use.

In keeping with customary representation of numerals using Arabic numerals, binary numbers are commonly written using the symbols **0** and **1**. When written, binary numerals are often subscripted, prefixed or suffixed in order to indicate their base, or radix. The following notations are equivalent:

- 100101 binary (explicit statement of format)
- 100101b (a suffix indicating binary format; also known as Intel convention[26][27])
- 100101B (a suffix indicating binary format)
- bin 100101 (a prefix indicating binary format)
- 100101_2 (a subscript indicating base-2 (binary) notation)

- %100101 (a prefix indicating binary format; also known as Motorola convention[26][27])
- 0b100101 (a prefix indicating binary format, common in programming languages)
- 6b100101 (a prefix indicating number of bits in binary format, common in programming languages)

When spoken, binary numerals are usually read digit-by-digit, in order to distinguish them from decimal numerals. For example, the binary numeral 100 is pronounced *one zero zero*, rather than *one hundred*, to make its binary nature explicit, and for purposes of correctness. Since the binary numeral 100 represents the value four, it would be confusing to refer to the numeral as *one hundred* (a word that represents a completely different value, or amount). Alternatively, the binary numeral 100 can be read out as "four" (the correct *value*), but this does not make its binary nature explicit.



A binary clock might use LEDs to express binary values. In this clock, each column of LEDs shows a binary-coded decimal numeral of the traditional sexagesimal time.

Counting in binary

Counting in binary is similar to counting in any other number system. Beginning with a single digit, counting proceeds through each symbol, in increasing order. Before examining binary counting, it is useful to briefly discuss the more familiar decimal counting system as a frame of reference.

Decimal pattern	Binary number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Decimal counting

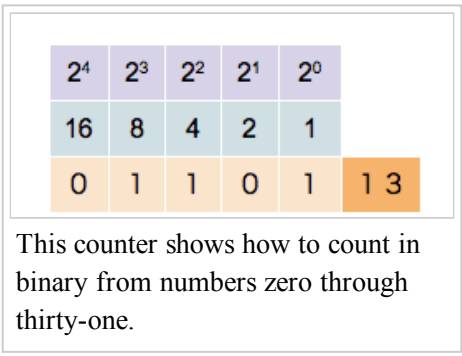
Decimal counting uses the ten symbols *0* through *9*. Counting begins with the incremental substitution of the least significant digit (rightmost digit) which is often called the *first digit*. When the available symbols for this position are exhausted, the least significant digit is reset to *0*, and the next digit of higher significance (one position to the left) is incremented (*overflow*), and incremental substitution of the low-order digit resumes. This method of reset and overflow is repeated for each digit of significance. Counting progresses as follows:

000, 001, 002, ... 007, 008, 009, (rightmost digit is reset to zero, and the digit to its left is incremented)
0**1**0, 011, 012, ...
...
090, 091, 092, ... 097, 098, 099, (rightmost two digits are reset to zeroes, and next digit is incremented)
100, 101, 102, ...

Binary counting

Binary counting follows the same procedure, except that only the two symbols *0* and *1* are available. Thus, after a digit reaches 1 in binary, an increment resets it to 0 but also causes an increment of the next digit to the left:

0000,
000**1**, (rightmost digit starts over, and next digit is incremented)
00**1**0, 0011, (rightmost two digits start over, and next digit is incremented)
0**1**00, 0101, 0110, 0111, (rightmost three digits start over, and the next digit is incremented)
1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 ...



This counter shows how to count in binary from numbers zero through thirty-one.

In the binary system, each digit represents an increasing power of 2, with the rightmost digit representing 2^0, the next representing 2^1, then 2^2, and so on. The equivalent decimal representation of a binary number is sum of the powers of 2 which each digit represents. For example, the binary number 100101 is converted to decimal form as follows:

$$100101_2 = [\,(\mathbf{1}) \times 2^5\,] + [\,(\mathbf{0}) \times 2^4\,] + [\,(\mathbf{0}) \times 2^3\,] + [\,(\mathbf{1}) \times 2^2\,] + [\,(\mathbf{0}) \times 2^1\,] + [\,(\mathbf{1}) \times 2^0\,]$$

$$100101_2 = [\,\mathbf{1} \times 32\,] + [\,\mathbf{0} \times 16\,] + [\,\mathbf{0} \times 8\,] + [\,\mathbf{1} \times 4\,] + [\,\mathbf{0} \times 2\,] + [\,\mathbf{1} \times 1\,]$$

$$\mathbf{100101_2 = 37_{10}}$$

Fractions

Fractions in binary only terminate if the denominator has 2 as the only prime factor. As a result, 1/10 does not have a finite binary representation, and this causes 10 × 0.1 not to be precisely equal to 1 in floating point arithmetic. As an example, to interpret the binary expression for 1/3 = .010101..., this means: $1/3 = 0 \times \mathbf{2^{-1}} + 1 \times \mathbf{2^{-2}} + 0 \times \mathbf{2^{-3}} + 1 \times \mathbf{2^{-4}} + \ldots = 0.3125 + \ldots$ An exact value cannot be found with a sum of a finite number of inverse powers of two, the zeros and ones in the binary representation of 1/3 alternate forever.

Fraction	Decimal	Binary	Fractional approximation
1/1	1 or 0.999...	1 or 0.111...	1/2 + 1/4 + 1/8...
1/2	0.5 or 0.4999...	0.1 or 0.0111...	1/4 + 1/8 + 1/16 . . .
1/3	0.333...	0.010101...	1/4 + 1/16 + 1/64 . . .
1/4	0.25 or 0.24999...	0.01 or 0.00111...	1/8 + 1/16 + 1/32 . . .
1/5	0.2 or 0.1999...	0.00110011...	1/8 + 1/16 + 1/128 . . .
1/6	0.1666...	0.0010101...	1/8 + 1/32 + 1/128 . . .
1/7	0.142857142857...	0.001001...	1/8 + 1/64 + 1/512 . . .
1/8	0.125 or 0.124999...	0.001 or 0.000111...	1/16 + 1/32 + 1/64 . . .
1/9	0.111...	0.000111000111...	1/16 + 1/32 + 1/64 . . .
1/10	0.1 or 0.0999...	0.000110011...	1/16 + 1/32 + 1/256 . . .
1/11	0.090909...	0.0001011101000101110 1...	1/16 + 1/64 + 1/128 . . .
1/12	0.08333...	0.00010101...	1/16 + 1/64 + 1/256 . . .
1/13	0.076923076923...	0.000100111011000100111011...	1/16 + 1/128 + 1/256 . . .
1/14	0.0714285714285...	0.0001001001...	1/16 + 1/128 + 1/1024 . . .
1/15	0.0666...	0.00010001...	1/16 + 1/256 . . .
1/16	0.0625 or 0.0624999...	0.0001 or 0.0000111...	1/32 + 1/64 + 1/128 . . .

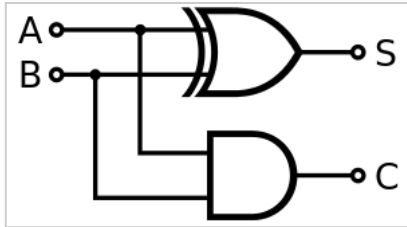
Binary arithmetic

Arithmetic in binary is much like arithmetic in other numeral systems. Addition, subtraction, multiplication, and division can be performed on binary numerals.

Addition

The simplest arithmetic operation in binary is addition. Adding two single-digit binary numbers is relatively simple, using a form of carrying:

$0 + 0 \rightarrow 0$
$0 + 1 \rightarrow 1$
$1 + 0 \rightarrow 1$
$1 + 1 \rightarrow 0$, carry 1 (since $1 + 1 = 2 = 0 + (1 \times 2^1)$)



The circuit diagram for a binary half adder, which adds two bits together, producing sum and carry bits.

Adding two "1" digits produces a digit "0", while 1 will have to be added to the next column. This is similar to what happens in decimal when certain single-digit numbers are added together; if the result equals or exceeds the value of the radix (10), the digit to the left is incremented:

$5 + 5 \rightarrow 0$, carry 1 (since $5 + 5 = 10 = 0 + (1 \times 10^1)$)
$7 + 9 \rightarrow 6$, carry 1 (since $7 + 9 = 16 = 6 + (1 \times 10^1)$)

This is known as *carrying*. When the result of an addition exceeds the value of a digit, the procedure is to "carry" the excess amount divided by the radix (that is, 10/10) to the left, adding it to the next positional value. This is correct since the next position has a weight that is higher by a factor equal to the radix. Carrying works the same way in binary:

```
 1 1 1 1 1    (carried digits)
   0 1 1 0 1
+  1 0 1 1 1
-------------
= 1 0 0 1 0 0 = 36
```

In this example, two numerals are being added together: 01101_2 (13_{10}) and 10111_2 (23_{10}). The top row shows the carry bits used. Starting in the rightmost column, $1 + 1 = 10_2$. The 1 is carried to the left, and the 0 is written at the bottom of the rightmost column. The second column from the right is added: $1 + 0 + 1 = 10_2$ again; the 1 is carried, and 0 is written at the bottom. The third column: $1 + 1 + 1 = 11_2$. This time, a 1 is carried, and a 1 is written in the bottom row. Proceeding like this gives the final answer 100100_2 (36 decimal).

When computers must add two numbers, the rule that: x xor y = (x + y) mod 2 for any two bits x and y allows for very fast calculation, as well.

Long carry method

A simplification for many binary addition problems is the Long Carry Method or Brookhouse Method of Binary Addition. This method is generally useful in any binary addition where one of the numbers contains a long "string" of ones. It is based on the simple premise that under the binary system, when given a "string" of digits composed entirely of *n* ones (*where: n* is any integer length), adding 1 will result in the number 1 followed by a string of *n* zeros. That concept follows, logically, just as in the decimal system, where adding 1 to a string of *n* 9s will result in the number 1 followed by a string of *n* 0s:

```
    Binary                      Decimal
   1 1 1 1 1    likewise       9 9 9 9 9
+          1                +          1
 ------------                ------------
 1 0 0 0 0 0                 1 0 0 0 0 0
```

Such long strings are quite common in the binary system. From that one finds that large binary numbers can be added using two simple steps, without excessive carry operations. In the following example, two numerals are being added together: $1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0_2$ (958_{10}) and $1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1_2$ (691_{10}), using the traditional carry method on the left, and the long carry method on the right:

```
Traditional Carry Method                    Long Carry Method
                          vs.
  1 1 1   1 1 1 1 1     (carried digits)   1 ←     1 ←           carry the 1 until it is one digit past the "string" below
    1 1 1 0 1 1 1 1 1 0                      ~~1 1 1~~ 0 ~~1 1 1 1 1~~ 0  cross out the "string",
+   1 0 1 0 1 1 0 0 1 1                  +   1 0 ~~1~~ 0 1 1 0 0 ~~1~~ 1  and cross out the digit that was added to it
-----------------------                  ---------------------
= 1 1 0 0 1 1 1 0 0 0 1                    1 1 0 0 1 1 1 0 0 0 1
```

The top row shows the carry bits used. Instead of the standard carry from one column to the next, the lowest-ordered "1" with a "1" in the corresponding place value beneath it may be added and a "1" may be carried to one digit past the end of the series. The "used" numbers must be crossed off, since they are already added. Other long strings may likewise be cancelled using the same technique. Then, simply add together any remaining digits normally. Proceeding in this manner gives the final answer of $1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1_2$ (1649_{10}). In our simple example using small numbers, the traditional carry method required eight carry operations, yet the long carry method required only two, representing a substantial reduction of effort.

Addition table

	0	**1**
0	0	1
1	1	10

The binary addition table is similar, but not the same, as the truth table of the logical disjunction operation $\vee$. The difference is that $\mathbf{1 \vee 1 = 1}$, while $\mathbf{1 + 1 = 10}$.

Subtraction

Subtraction works in much the same way:

$0 - 0 \rightarrow 0$
$0 - 1 \rightarrow 1$, borrow 1
$1 - 0 \rightarrow 1$
$1 - 1 \rightarrow 0$

Subtracting a "1" digit from a "0" digit produces the digit "1", while 1 will have to be subtracted from the next column. This is known as *borrowing*. The principle is the same as for carrying. When the result of a subtraction is less than 0, the least possible value of a digit, the procedure is to "borrow" the deficit divided by the radix (that is, 10/10) from the left, subtracting it from the next positional value.

```
   *   * * *   (starred columns are borrowed from)
 1 1 0 1 1 1 0
-    1 0 1 1 1
---------------
= 1 0 1 0 1 1 1
```

```
 *            (starred columns are borrowed from)
 1 0 1 1 1 1 1
-  1 0 1 0 1 1
---------------
= 0 1 1 0 1 0 0
```

Subtracting a positive number is equivalent to *adding* a negative number of equal absolute value. Computers use signed number representations to handle negative numbers—most commonly the two's complement notation. Such representations eliminate the need for a separate "subtract" operation. Using two's complement notation subtraction can be summarized by the following formula:

A − B = A + not B + 1

Multiplication

Multiplication in binary is similar to its decimal counterpart. Two numbers A and B can be multiplied by partial products: for each digit in B, the product of that digit in A is calculated and written on a new line, shifted leftward so that its rightmost digit lines up with the digit in B that was used. The sum of all these partial products gives the final result.

Since there are only two digits in binary, there are only two possible outcomes of each partial multiplication:

- If the digit in B is 0, the partial product is also 0

- If the digit in B is 1, the partial product is equal to A

For example, the binary numbers 1011 and 1010 are multiplied as follows:

```
        1 0 1 1   (A)
      × 1 0 1 0   (B)
      ---------
        0 0 0 0   ← Corresponds to the rightmost 'zero' in B
  +     1 0 1 1     ← Corresponds to the next 'one' in B
  +   0 0 0 0
  + 1 0 1 1
  ---------------
  = 1 1 0 1 1 1 0
```

Binary numbers can also be multiplied with bits after a binary point:

```
            1 0 1 . 1 0 1     A (5.625 in decimal)
          × 1 1 0 . 0 1       B (6.25  in decimal)
          ------------------
                1 . 0 1 1 0 1   ← Corresponds to a 'one' in B
    +         0 0 . 0 0 0 0     ← Corresponds to a 'zero' in B
    +       0 0 0 . 0 0 0
    +     1 0 1 1 . 0 1
    +   1 0 1 1 0 . 1
    -------------------------
    =   1 0 0 0 1 1 . 0 0 1 0 1  (35.15625 in decimal)
```

See also Booth's multiplication algorithm.

Multiplication table

	0	**1**
0	0	0
1	0	1

The binary multiplication table is the same as the truth table of the logical conjunction operation ∧.

Division

Long division in binary is again similar to its decimal counterpart.

In the example below, the divisor is 101_2, or 5 decimal, while the dividend is 11011_2, or 27 decimal. The procedure is the same as that of decimal long division; here, the divisor 101_2 goes into the first three digits 110_2 of the dividend one time, so a "1" is written on the top line. This result is multiplied by the divisor, and subtracted from the first three digits of the dividend; the next digit (a "1") is included to obtain a new three-digit sequence:

```
            1
        ___________
1 0 1   ) 1 1 0 1 1
        - 1 0 1
          -----
          0 0 1
```

The procedure is then repeated with the new sequence, continuing until the digits in the dividend have been exhausted:

```
          1 0 1
        _________
1 0 1  ) 1 1 0 1 1
       - 1 0 1
         -----
             1 1 1
         -   1 0 1
             -----
               1 0
```

Thus, the quotient of 11011_2 divided by 101_2 is 101_2, as shown on the top line, while the remainder, shown on the bottom line, is 10_2. In decimal, 27 divided by 5 is 5, with a remainder of 2.

Square root

The process of taking a binary square root digit by digit is the same as for a decimal square root, and is explained here. An example is:

```
           1 0 0 1
          ---------
         √ 1010001
           1
          ---------
     101    01
            0
          --------
     1001    100
               0
          --------
     10001   10001
             10001
            -------
                 0
```

Bitwise operations

Though not directly related to the numerical interpretation of binary symbols, sequences of bits may be manipulated using Boolean logical operators. When a string of binary symbols is manipulated in this way, it is called a bitwise operation; the logical operators AND, OR, and XOR may be performed on corresponding bits in two binary numerals provided as input. The logical NOT operation may be performed on individual bits in a single binary numeral provided as input. Sometimes, such operations may be used as arithmetic short-cuts, and may have other computational benefits as well. For example, an arithmetic shift left of a binary number is the equivalent of multiplication by a (positive, integral) power of 2.

Conversion to and from other numeral systems

Decimal

To convert from a base-10 integer to its base-2 (binary) equivalent, the number is divided by two. The remainder is the least-significant bit. The quotient is again divided by two; its remainder becomes the next least significant bit. This process repeats until a quotient of one is reached. The sequence of remainders (including the final quotient of one) forms the binary value, as each remainder must be either zero or one when dividing by two. For example, $(357)_{10}$ is expressed as $(101100101)_2$.[28]

Conversion from base-2 to base-10 simply inverts the preceding algorithm. The bits of the binary number are used one by one, starting with the most significant (leftmost) bit. Beginning with the value 0, the prior value is doubled, and the next bit is then added to produce the next value. This can be organized in a multi-column table. For example, to convert 10010101101_2 to decimal:

Prior value	× 2 +	Next bit	Next value
0	× 2 +	**1**	= 1
1	× 2 +	**0**	= 2
2	× 2 +	**0**	= 4
4	× 2 +	**1**	= 9
9	× 2 +	**0**	= 18
18	× 2 +	**1**	= 37
37	× 2 +	**0**	= 74
74	× 2 +	**1**	= 149
149	× 2 +	**1**	= 299
299	× 2 +	**0**	= 598
598	× 2 +	**1**	= **1197**

Conversion of $(357)_{10}$ to binary notation results in (101100101)

The result is 1197_{10}. Note that the first Prior Value of 0 is simply an initial decimal value. This method is an application of the Horner scheme.

Binary	1	0	0	1	0	1	0	1	1	0	1
Decimal	$1\times2^{10} + 0\times2^{9} + 0\times2^{8} + 1\times2^{7} + 0\times2^{6} + 1\times2^{5} + 0\times2^{4} + 1\times2^{3} + 1\times2^{2} + 0\times2^{1} + 1\times2^{0} = 1197$										

The fractional parts of a number are converted with similar methods. They are again based on the equivalence of shifting with doubling or halving.

In a fractional binary number such as 0.11010110101_2, the first digit is $\frac{1}{2}$, the second $(\frac{1}{2})^2 = \frac{1}{4}$, etc. So if there is a 1 in the first place after the decimal, then the number is at least $\frac{1}{2}$, and vice versa. Double that number is at least 1. This suggests the algorithm: Repeatedly double the number to be converted, record if the result is at least 1, and then throw away the integer part.

For example, $(\frac{1}{3})_{10}$, in binary, is:

Converting	Result
$\frac{1}{3}$	0.
$\frac{1}{3} \times 2 = \frac{2}{3} < 1$	0.0
$\frac{2}{3} \times 2 = 1\frac{1}{3} \geq 1$	0.01
$\frac{1}{3} \times 2 = \frac{2}{3} < 1$	0.010
$\frac{2}{3} \times 2 = 1\frac{1}{3} \geq 1$	0.0101

Thus the repeating decimal fraction $0.\overline{3}...$ is equivalent to the repeating binary fraction $0.\overline{01}...$.

Or for example, 0.1_{10}, in binary, is:

Converting	Result
0.1	0.
$0.1 \times 2 = \mathbf{0.2} < 1$	0.0
$0.2 \times 2 = \mathbf{0.4} < 1$	0.00
$0.4 \times 2 = \mathbf{0.8} < 1$	0.000
$0.8 \times 2 = \mathbf{1.6} \geq 1$	0.0001
$0.6 \times 2 = \mathbf{1.2} \geq 1$	0.00011
$0.2 \times 2 = \mathbf{0.4} < 1$	0.000110
$0.4 \times 2 = \mathbf{0.8} < 1$	0.0001100
$0.8 \times 2 = \mathbf{1.6} \geq 1$	0.00011001
$0.6 \times 2 = \mathbf{1.2} \geq 1$	0.000110011
$0.2 \times 2 = \mathbf{0.4} < 1$	0.0001100110

This is also a repeating binary fraction $0.0\overline{0011}...$. It may come as a surprise that terminating decimal fractions can have repeating expansions in binary. It is for this reason that many are surprised to discover that 0.1 + ... + 0.1, (10 additions) differs from 1 in floating point arithmetic. In fact, the only binary fractions with terminating expansions are of the form of an integer divided by a power of 2, which 1/10 is not.

The final conversion is from binary to decimal fractions. The only difficulty arises with repeating fractions, but otherwise the method is to shift the fraction to an integer, convert it as above, and then divide by the appropriate power of two in the decimal base. For example:

$$\begin{aligned} x &= 1100.1\overline{01110}\ldots \\ x \times 2^6 &= 1100101110.\overline{01110}\ldots \\ x \times 2 &= 11001.\overline{01110}\ldots \\ x \times (2^6 - 2) &= 1100010101 \\ x &= 1100010101/111110 \\ x &= (789/62)_{10} \end{aligned}$$

Another way of converting from binary to decimal, often quicker for a person familiar with hexadecimal, is to do so indirectly—first converting (x in binary) into (x in hexadecimal) and then converting (x in hexadecimal) into (x in decimal).

For very large numbers, these simple methods are inefficient because they perform a large number of multiplications or divisions where one operand is very large. A simple divide-and-conquer algorithm is more effective asymptotically: given a binary number, it is divided by 10^k, where k is chosen so that the quotient roughly equals the remainder; then each of these pieces is converted to decimal and the two are concatenated. Given a decimal number, it can be split into two pieces of about the same size, each of which is converted to binary, whereupon the first converted piece is multiplied by 10^k and added to the second converted piece, where k is the number of decimal digits in the second, least-significant piece before conversion.

Hexadecimal

Binary may be converted to and from hexadecimal somewhat more easily. This is because the radix of the hexadecimal system (16) is a power of the radix of the binary system (2). More specifically, $16 = 2^4$, so it takes four digits of binary to represent one digit of hexadecimal, as shown in the adjacent table.

$\mathbf{0}_{hex} = 0_{dec} = 0_{oct}$	0	0	0	0
$\mathbf{1}_{hex} = 1_{dec} = 1_{oct}$	0	0	0	1
$\mathbf{2}_{hex} = 2_{dec} = 2_{oct}$	0	0	1	0
$\mathbf{3}_{hex} = 3_{dec} = 3_{oct}$	0	0	1	1
$\mathbf{4}_{hex} = 4_{dec} = 4_{oct}$	0	1	0	0
$\mathbf{5}_{hex} = 5_{dec} = 5_{oct}$	0	1	0	1
$\mathbf{6}_{hex} = 6_{dec} = 6_{oct}$	0	1	1	0
$\mathbf{7}_{hex} = 7_{dec} = 7_{oct}$	0	1	1	1
$\mathbf{8}_{hex} = 8_{dec} = 10_{oct}$	1	0	0	0
$\mathbf{9}_{hex} = 9_{dec} = 11_{oct}$	1	0	0	1
$\mathbf{A}_{hex} = 10_{dec} = 12_{oct}$	1	0	1	0
$\mathbf{B}_{hex} = 11_{dec} = 13_{oct}$	1	0	1	1
$\mathbf{C}_{hex} = 12_{dec} = 14_{oct}$	1	1	0	0
$\mathbf{D}_{hex} = 13_{dec} = 15_{oct}$	1	1	0	1
$\mathbf{E}_{hex} = 14_{dec} = 16_{oct}$	1	1	1	0
$\mathbf{F}_{hex} = 15_{dec} = 17_{oct}$	1	1	1	1

To convert a hexadecimal number into its binary equivalent, simply substitute the corresponding binary digits:

$3A_{16} = 0011\ 1010_2$
$E7_{16} = 1110\ 0111_2$

To convert a binary number into its hexadecimal equivalent, divide it into groups of four bits. If the number of bits isn't a multiple of four, simply insert extra **0** bits at the left (called padding). For example:

1010010_2 = 0101 0010 grouped with padding = 52_{16}
11011101_2 = 1101 1101 grouped = DD_{16}

To convert a hexadecimal number into its decimal equivalent, multiply the decimal equivalent of each hexadecimal digit by the corresponding power of 16 and add the resulting values:

$C0E7_{16} = (12 \times 16^3) + (0 \times 16^2) + (14 \times 16^1) + (7 \times 16^0) = (12 \times 4096) + (0 \times 256) + (14 \times 16) + (7 \times 1) = 49{,}383_{10}$

Octal

Binary is also easily converted to the octal numeral system, since octal uses a radix of 8, which is a power of two (namely, 2^3, so it takes exactly three binary digits to represent an octal digit). The correspondence between octal and binary numerals is the same as for the first eight digits of hexadecimal in the table above. Binary 000 is equivalent to the octal digit 0, binary 111 is equivalent to octal 7, and so forth.

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Converting from octal to binary proceeds in the same fashion as it does for hexadecimal:

$65_8 = 110\ 101_2$
$17_8 = 001\ 111_2$

And from binary to octal:

$101100_2 = 101\ 100_2$ grouped $= 54_8$
$10011_2 = 010\ 011_2$ grouped with padding $= 23_8$

And from octal to decimal:

$65_8 = (6 \times 8^1) + (5 \times 8^0) = (6 \times 8) + (5 \times 1) = 53_{10}$
$127_8 = (1 \times 8^2) + (2 \times 8^1) + (7 \times 8^0) = (1 \times 64) + (2 \times 8) + (7 \times 1) = 87_{10}$

Representing real numbers

Non-integers can be represented by using negative powers, which are set off from the other digits by means of a radix point (called a decimal point in the decimal system). For example, the binary number 11.01_2 thus means:

$\mathbf{1} \times 2^1$ $(1 \times 2 = \mathbf{2})$ plus
$\mathbf{1} \times 2^0$ $(1 \times 1 = \mathbf{1})$ plus
$\mathbf{0} \times 2^{-1}$ $(0 \times \frac{1}{2} = \mathbf{0})$ plus
$\mathbf{1} \times 2^{-2}$ $(1 \times \frac{1}{4} = \mathbf{0.25})$

For a total of 3.25 decimal.

All dyadic rational numbers $\frac{p}{2^a}$ have a *terminating* binary numeral—the binary representation has a finite number of terms after the radix point. Other rational numbers have binary representation, but instead of terminating, they *recur*, with a finite sequence of digits repeating indefinitely. For instance

$$\frac{\mathbf{1}_{10}}{\mathbf{3}_{10}} = \frac{\mathbf{1}_2}{\mathbf{11}_2} = 0.0101010\overline{101}\ldots_2$$

$$\frac{\mathbf{12}_{10}}{\mathbf{17}_{10}} = \frac{\mathbf{1100}_2}{\mathbf{10001}_2} = 0.10110100\ 10110100\ \overline{10110100}\ldots_2$$

The phenomenon that the binary representation of any rational is either terminating or recurring also occurs in other radix-based numeral systems. See, for instance, the explanation in decimal. Another similarity is the existence of alternative representations for any terminating representation, relying on the fact that 0.111111… is the sum of the geometric series $2^{-1} + 2^{-2} + 2^{-3} + \ldots$ which is 1.

Binary numerals which neither terminate nor recur represent irrational numbers. For instance,

- 0.1010010001000010000010 0… does have a pattern, but it is not a fixed-length recurring pattern, so the number is irrational
- 1.0110101000001001111001100110011111110… is the binary representation of $\sqrt{\mathbf{2}}$, the square root of 2, another irrational. It has no discernible pattern. See irrational number.

See also

- Binary code

- Binary-coded decimal
- Finger binary
- Gray code
- Linear feedback shift register
- Offset binary
- Quibinary
- Reduction of summands
- Redundant binary representation
- Repeating decimal
- SZTAKI Desktop Grid searches for generalized binary number systems up to dimension 11.
- Two's complement

Notes

1. Chrisomalis, Stephen (2010), *Numerical Notation: A Comparative History*, Cambridge University Press, pp. 42–43, ISBN 9780521878180.
2. Rudman, Peter Strom (2007), *How Mathematics Happened: The First 50,000 Years*, Prometheus Books, pp. 135–136, ISBN 9781615921768.
3. Edward Hacker; Steve Moore; Lorraine Patsco (2002). *I Ching: An Annotated Bibliography*. Routledge. p. 13. ISBN 978-0-415-93969-0.
4. Redmond & Hon (2014), p. 227.
5. Jonathan Shectman (2003). *Groundbreaking Scientific Experiments, Inventions, and Discoveries of the 18th Century*. Greenwood Publishing. p. 29. ISBN 978-0-313-32015-6.
6. Zhonglian, Shi; Wenzhao, Li; Poser, Hans (2000). *Leibniz' Binary System and Shao Yong's "Xiantian Tu" in :Das Neueste über China: G.W. Leibnizens Novissima Sinica von 1697 : Internationales Symposium, Berlin 4. bis 7. Oktober 1997*. Stuttgart: Franz Steiner Verlag. pp. 165–170. ISBN 3515074481.
7. Sanchez, Julio; Canton, Maria P. (2007). *Microcontroller programming: the microchip PIC*. Boca Raton, Florida: CRC Press. p. 37. ISBN 0-8493-7189-9.
8. W. S. Anglin and J. Lambek, *The Heritage of Thales*, Springer, 1995, ISBN 0-387-94544-X
9. Binary Numbers in Ancient India (http://home.ica.net/~roymanju/Binary.htm)
10. Math for Poets and Drummers (http://www.sju.edu/~rhall/Rhythms/Poets/arcadia.pdf) (pdf, 145KB)
11. "Binary Numbers in Ancient India".
12. Stakhov, Alexey; Olsen, Scott Anthony (2009). *The mathematics of harmony: from Euclid to contemporary mathematics and computer science*. ISBN 978-981-277-582-5.
13. Bender, Andrea; Beller, Sieghard (16 December 2013). "Mangarevan invention of binary steps for easier calculation". *Proceedings of the National Academy of Sciences*. **111**: 1322–1327. doi:10.1073/pnas.1309160110.
14. Bacon, Francis (1605). "The Advancement of Learning". London. pp. Chapter 1.
15. Leibniz G., Explication de l'Arithmétique Binaire, Die Mathematische Schriften, ed. C. Gerhardt, Berlin 1879, vol.7, p.223; Engl. transl.[1] (http://www.leibniz-translations.com/binary.htm)
16. J.E.H. Smith (2008). *Leibniz: What Kind of Rationalist?: What Kind of Rationalist?*. Springer. p. 415. ISBN 978-1-4020-8668-7.
17. Aiton, Eric J. (1985). *Leibniz: A Biography*. Taylor & Francis. pp. 245–8. ISBN 0-85274-470-6.
18. Yuen-Ting Lai (1998). *Leibniz, Mysticism and Religion*. Springer. pp. 149–150. ISBN 978-0-7923-5223-5.
19. Boole, George (2009) [1854]. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities* (Macmillan, Dover Publications, reprinted with corrections [1958] ed.). New York: Cambridge University Press. ISBN 978-1-108-00153-3.
20. Shannon, Claude Elwood (1940). *A symbolic analysis of relay and switching circuits*. Cambridge: Massachusetts Institute of Technology.
21. "National Inventors Hall of Fame – George R. Stibitz". 20 August 2008. Retrieved 5 July 2010.
22. "George Stibitz : Bio". Math & Computer Science Department, Denison University. 30 April 2004. Retrieved 5 July 2010.
23. "Pioneers – The people and ideas that made a difference – George Stibitz (1904–1995)". Kerry Redshaw. 20 February 2006. Retrieved 5 July 2010.
24. "George Robert Stibitz – Obituary". Computer History Association of California. 6 February 1995. Retrieved 5 July 2010.
25. "Konrad Zuse's Legacy: The Architecture of the Z1 and Z3" (PDF). *IEEE Annals of the History of Computing*. **19** (2): 5–15. 1997. doi:10.1109/85.586067.

26. Küveler, Gerd; Schwoch, Dietrich (2013) [1996]. *Arbeitsbuch Informatik - eine praxisorientierte Einführung in die Datenverarbeitung mit Projektaufgabe* (in German). Vieweg-Verlag, reprint: Springer-Verlag. doi:10.1007/978-3-322-92907-5. ISBN 978-3-528-04952-2. 9783322929075. Retrieved 2015-08-05.
27. Küveler, Gerd; Schwoch, Dietrich (2007-10-04). *Informatik für Ingenieure und Naturwissenschaftler: PC- und Mikrocomputertechnik, Rechnernetze* (in German). **2** (5 ed.). Vieweg, reprint: Springer-Verlag. ISBN 3834891916. 9783834891914. Retrieved 2015-08-05.
28. "Base System". Retrieved 31 August 2016.

References

- Sanchez, Julio; Canton, Maria P. (2007). *Microcontroller programming: the microchip PIC*. Boca Raton, FL: CRC Press. p. 37. ISBN 0-8493-7189-9.
- Redmond, Geoffrey; Hon, Tze-Ki (2014). *Teaching the I Ching*. Oxford University Press. ISBN 0-19-976681-9.

External links

- Binary System (http://www.cut-the-knot.org/do_you_know/BinaryHistory.shtml) at cut-the-knot
- Conversion of Fractions (http://www.cut-the-knot.org/blue/frac_conv.shtml) at cut-the-knot
- Binary Digits (http://www.mathsisfun.com/binary-digits.html) at Math Is Fun (http://www.mathsisfun.com/)
- How to Convert from Decimal to Binary (http://www.wikihow.com/Convert-from-Decimal-to-Binary) at wikiHow
- Learning exercise for children at CircuitDesign.info (http://www.circuitdesign.info/blog/2008/06/the-binary-number-system-part-2-binary-weighting/)
- Binary Counter with Kids (http://gwydir.demon.co.uk/jo/numbers/binary/kids.htm)
- "Magic" Card Trick (http://gwydir.demon.co.uk/jo/numbers/binary/cards.htm)
- Quick reference on Howto read binary (http://www.mycomputeraid.com/networking-support/general-networking-support/howto-read-binary-basics/)
- Binary converter to HEX/DEC/OCT with direct access to bits (http://calc.50x.eu/)
- Sir Francis Bacon's BiLiteral Cypher system (http://www.baconlinks.com/docs/BILITERAL.doc), predates binary number system.
- Leibniz' binary numeral system, 'De progressione dyadica', 1679, online and analyzed on *BibNum (https://www.bibnum.education.fr/calculinformatique/calcul/de-la-numeration-binaire)* [click 'à télécharger' for English analysis]