Chapter 02: Data Representation

1. Introduction

1.1. Representation of Data Processed by Computers

The information processed by a computer can be in different formats (text, numbers, images, sound, videos, etc.), but it is always represented and manipulated by the computer in digital form. In fact, all information are processed as a sequence of 0_s and 1_s. The unit of information is the binary digits (0 and 1) known as bits (short for binary digit).

Binary representation is used because it is simple and technically easy to implement using bistables (systems with two stable states realized with transistors).

The data encoding refers to the process of converting information from one format (text, image, etc.) into another (internal machine representation, which is always a sequence of bits), typically in a more suitable representation for storage, transmission, or processing. This transformation ensures that data can be efficiently and accurately handled by computer systems, communication networks, or other devices.

1.2. Quantity of processed Data

The basic unit of measure for the quantity of data in computer science is the **bit**, where a bit can take the value 0 or 1.

Question: How many states can be represented with 3 bits? with 4 bits? and with n bits in general?

- With 3 bits, we can represent $2^3 = 8$ different states.
- With 4 bits, we can represent $2^4 = 16$ different states.
- And with n bits in general, we can represent 2^n different states.

Each group of 8 bits constitutes 1 **Byte**, symbolized by *B*.

Also:

❖ 2^{10} bits = 1024 bits = 1 **Kbits**   / 2^{10} B = 1024 B = 1 **KB** (1 Kilo BYTE)

❖ 2^{10} Kbits = 1024 Kbits = 1 **Mbits** / 2^{10} KB = 1024 KB = 1 **MB** (1 Méga BYTE)

❖ 2^{10} Mbits = 1024 Mbits = 1 **Gbits** / 2^{10} MB = 1024 MB = 1 **GB** (1 Giga BYTE)

❖ 2^{10} Gbits = 1024 Gbits = 1 **Tbits** / 2^{10} GB = 1024 GB = 1 **TB** (1 Téra BYTE)

Question: Convert 2 GB to bits and then to Kbits?

It is clear that data is processed in binary form in a computer, whether it is text (associated with standardized codes for each character), images (associated with codes for each pixel's color in the image), sound (associated with codes for each sound frequency), …etc. Therefore, it is essential to take a closer look at the manipulation of binary data and its relationship with other Numeral systems.

2. Numeral systems

Over time, several Numeral systems (also known as numbering systems) have emerged. From the Mesopotamian positional system (where the position of the digit indicates its place value, similar to the Arabic numeration we use today) to the Egyptian and Roman additive systems (where the represented number is the sum of the symbols), to the Chinese system, which excelled in calculations (with the invention of the abacus) and is also positional, …etc.

Example: Egyptian numeration

 = **345**

2.1. Representation

A number: $(XXX)_b$ indicates the representation of a number XXX in base b.

The common base we know and use every day is base 10 (decimal system) for representing various quantities, digits, and numbers (currency, phone numbers, sizes, dates, etc.), and base 60 (sexagesimal system) for representing time.

How is a number represented in a base b?

1. If **b ≤ 10**, we simply use the digits from 0 to b-1.

Example: Base 8 (octal system): Any number will be a combination of digits belonging to the set {0, ..., 7}.

2. If **b > 10**, we simply use the digits from 0 to 9, and then use letters in alphabetical order.

Example: Base 16 (hexadecimal system): Any number will be a combination of symbols belonging to {0, ..., 9, A, B, C, D, E, F}, where (A=10, ..., F=15).

So, each numeral system uses a set of symbols (digits) to represent different numbers. The number of these digits is always equal to the base of the numeral system itself. In other words, the base of the numeral system is equal to the cardinality of the set of symbols used in that base.

Example:

- In binary, base of the binary system = 2; set of symbols used: A = {0, 1}, Card (A) = 2 = base of the binary system.
- In octal, base of the octal system = 8; set of symbols used: A = {0, 1, 2, 3, 4, 5, 6, 7}, Card (A) = 8 = base of the octal system.

So:

- A number with n digits (symbols) is represented as a sequence (a_i), where $0 \leq i \leq n-1$: a_{n-1} ... a_1 a_0

where a_0 is the least significant digit and a_{n-1} is the most significant digit.

Example: Let's consider: 10011101.

1	0	0	1	1	1	0	1
7	6	5	4	3	2	1	0

Most significant bit   **Least significant bit**
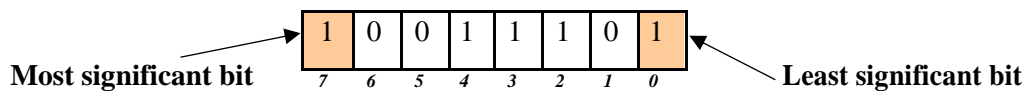
The numeral systems that interest us in the field of computer science are: decimal, binary, octal, and hexadecimal.

2.2. The decimal system

The decimal system, also known as the base-10 system, is the most commonly used numbering system in everyday life. It uses ten different digits to represent numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Each digit's value is determined by its position within the number, and the system is positional.

In the decimal system, the rightmost digit represents the units place, the next digit to the left represents the tens place, then the hundreds, and so on, with each position representing increasing powers of 10.

For instance, the number 9745 in the decimal system can be interpreted as:

- $9745 = 9{\times}1000 + 7{\times}100 + 4{\times}10 + 5{\times}1$

- $9745 = 9{\times}10^3 \times 7{\times}10^2 + 4{\times}10^1 + 5{\times}10^0$

We notice that each digit of the number is multiplied by a power of 10. This power represents the weight of the digit.

9	7	4	5
3	2	1	0

heavy weight   Low weight
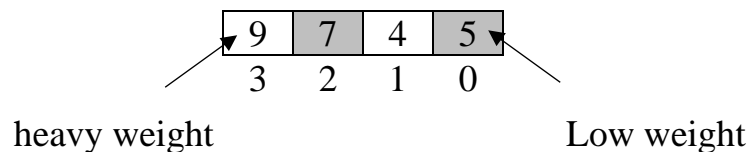
The exponent of this power is zero for the rightmost digit and increases by one for each digit to the left.

Note: This way of writing numbers is called a positional numeral system. It is applicable to all the numeral systems we will see in this course (decimal, binary, octal, and hexadecimal).

2.3. The octal system

Following what we mentioned in *section 2.1*, the octal system uses a numbering system with a base of 8 (octal: Latin octo=eight) and, therefore uses 8 symbols: from 0 to 7. Thus, a number expressed in base 8 will be represented in the following way, for example: $(745)_8$.

Reminder: When writing a number, it is essential to specify the base in which it is expressed to avoid any ambiguity (for example, 745 also exists in base 10). Thus, the number will be enclosed in parentheses (745 in our example) and indexed with a number representing its base (8 is placed as an

index). By convention, when the base is not explicitly stated, it is assumed to be 10 by default.

2.4. The binary system

As we saw earlier, in the binary system, each digit can only have one of two values: 0 or 1. Therefore, the system has a base of 2.

Example: Representation of numbers from 0 to 16 in decimal and their equivalents in binary and octal.

(Decimal system)$_{10}$	(Octal system)$_8$	(Binary system)$_2$
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111
16	20	10000

2.5. The hexadecimal system

The hexadecimal system uses the following 16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Therefore, the system has a base of 16.

Example: If we revisit the previous table but with decimal values and their equivalents in binary and hexadecimal, we will have:

(Decimal system)$_{10}$	(Hexadecimal system)$_{16}$	(Binary system)$_2$
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	10000

3. Conversion between different bases

3.1.   Convert from any base to decimal

To convert a number from any base to decimal, you can use the positional value method. Here's a step-by-step guide:

1. Write down the number in its original base representation (e.g., binary, octal, hexadecimal, etc.).
2. Identify the base of the number you are converting. For example, binary is base 2, octal is base 8, and hexadecimal is base 16.
3. Assign each digit a positional value based on its position in the number, starting from the rightmost digit. The rightmost digit is the least significant digit.
   For example, in base 2 (binary):
   - The rightmost digit has a positional value of $2^0 = 1$ (1's place).
   - The next digit to the left has a positional value of $2^1 = 2$ (2's place).
   - The next digit to the left has a positional value of $2^2 = 4$ (4's place).

And so on.

For base 8 (octal), each digit's positional value is a power of 8, and for base 16 (hexadecimal), it is a power of 16.

4. Multiply each digit by its positional value.
5. Sum up the results from step 4 to obtain the decimal representation of the number.

Let $(a_n\, a_{n-1}\, ... \, a_2\, a_1\, a_0)_b$ be a number expressed in base b. The value of this number in decimal is equal to:

$$(a_n \times b^n) + (a_{n-1} \times b^{(n-1)}) + ... + (a_2 \times b^2) + (a_1 \times b^1) + (a_0 \times b^0)$$

For example, let's convert to decimal the binary numbers: $(1011)_2$, $(16257)_8$ et $(F53)_{16}$

1. Binary number: 1011
2. Base: 2 (binary)
3. Positional values: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$
4. Multiply each digit by its positional value: $1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 8 + 0 + 2 + 0 = 11$
5. Decimal representation: 11

So, the binary number 1011 is equal to the decimal number 11.

- $(1011)_2 = (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10} = (1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1)_{10} = (11)_{10}$

- $(16257)_8 = 1 \times 8^4 + 6 \times 8^3 + 2 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 = 1 \times 4096 + 6 \times 512 + 2 \times 64 + 5 \times 8 + 7$
  $= 4096 + 3072 + 128 + 40 + 7 \quad = \quad (7343)_{10}$

- $(F53)_{16} = 15 \times 16^2 + 5 \times 16^1 + 3 \times 16^0 = 15 \times 256 + 5 \times 16 + 3 = 3840 + 80 + 3 = (3923)_{10}$

Note: In the case where there is a fractional part $a_1 a_2 \ldots\ldots a_n$ (fractional numbers are those that have digits after the decimal point), its value in decimal will be equal to the following sum:

$$a_1 \times b^{-1} + a \times b^{-2} + \ldots\ldots + a_n \times b^{-n}$$

Example: Let's convert the following fractional numbers to decimal: $(1110{,}101)_2$, $(642{,}21)_8$, and $(A3F{,}C)_{16}$

- $(1110{,}101)_2 = (1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3})_{10}$
  $= (8 + 4 + 2 + 0 + 1/2 + 1/4 + 1/8)_{10} = (14{,}625)_{10}$

- $(642{,}21)_8 = (6 \times 8^2 + 4 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} + 1 \times 8^{-2})_{10}$
  $= (6 \times 64 + 4 \times 8 + 2 \times 1 + 2 \times 0{,}125 + 1 \times 0{,}015625)_{10}$
  $= (384 + 32 + 2 + 0.25 + 0.015625)_{10} = (418.265\ 625)_{10}$

- $(A3F{,}C)_{16} = (10 \times 16^2 + 3 \times 16^1 + 15 \times 16^0 + 12 \times 16^{-1})_{10}$
  $= (10 \times 256 + 3 \times 16 + 15 \times 1 + 12 \times 0{,}0625)_{10}$
  $= (2623.75)_{10}$

3.2.   Conversion of a decimal number to binary

To convert a decimal number to a binary number, you can use the process of repeated division by 2. Here's a step-by-step guide:

1. Start with the decimal number you want to convert to binary.
2. Divide the number by 2 and note the quotient and the remainder. The remainder will be the least significant bit (LSB) of the binary representation.

3. Continue dividing the quotient from step 2 by 2 and note the remainders each time. Write down the remainders in reverse order, as they will form the binary representation from right to left.
4. Repeat step 3 until the quotient becomes 0.
5. The binary representation is the series of remainders obtained in step 3, read from right to left.

> **For example,** let's convert the decimal number 44 to binary:

- Step 1: Start with 44.
- Step 2: $44 \div 2 = 22$ with a remainder of 0 (LSB).
- Step 3: $22 \div 2 = 11$ with a remainder of 0.
- Step 4: $11 \div 2 = 5$ with a remainder of 1.
- Step 5: $5 \div 2 = 2$ with a remainder of 1.
- Step 6: $2 \div 2 = 1$ with a remainder of 0.
- Step 6: $1 \div 2 = 0$ with a remainder of 1 (MSB).
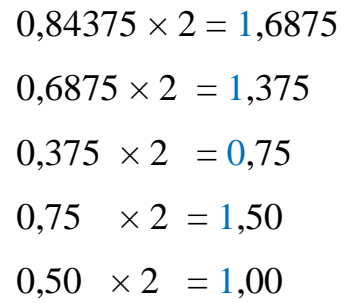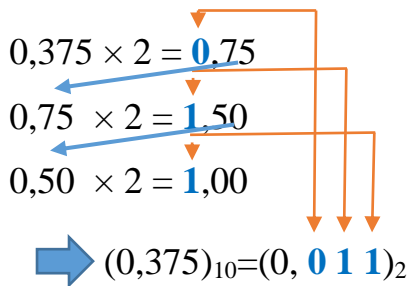
The binary representation of 44 is:

$(44)_{10} = (101100)_2$
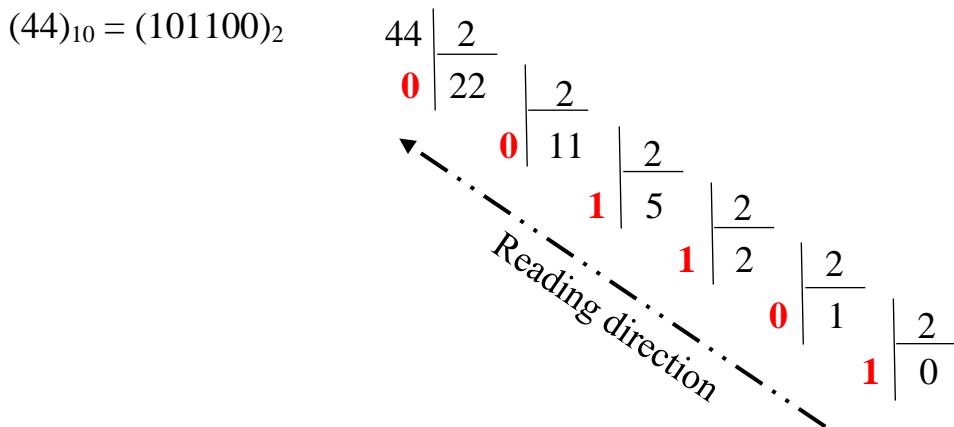
3.2.1. Conversion of the Fractional Part "Decimal Part" to Binary

To convert the fractional part of a decimal number to binary, we have to follow these steps:

1. Multiply the fractional part by 2,
2. Take the whole number part of the result as the first binary digit after the decimal point,
3. Keep the fractional part of the result,
4. Repeat steps 1 to 3 with the fractional part obtained in step 3 until the fractional part becomes zero or until you reach the desired number of binary digits after the decimal point.

Let's illustrate this with examples:

Convert the decimal fractional numbers **0.375** and **0.84375** to binary

$0,375 \times 2 = \mathbf{0},75$

$0,75 \ \times 2 = \mathbf{1},50$

$0,50 \ \times 2 = \mathbf{1},00$

➡ $(0,375)_{10} = (0, \mathbf{0\ 1\ 1})_2$

$0,84375 \times 2 = 1,6875$

$0,6875 \times 2 \ = 1,375$

$0,375 \ \times 2 \ = 0,75$

$0,75 \ \ \times 2 \ = 1,50$

$0,50 \ \ \times 2 \ = 1,00$

➡ $(0,84375)_{10} = (0,\mathbf{11011})_2$

3.3. The Relationship Between Binary Numbers and Octal Numbers

First, if we want to obtain the octal expression of a number expressed in decimal, we simply need to follow the method of successive division by 8 (just as we did to convert to base 2) until the obtained quotient is equal to 0. The remainders of these divisions, read from bottom to top, represent the octal number.

Example: Let's convert $(47)_{10}$ to the octal system and the binary system. We wil get :
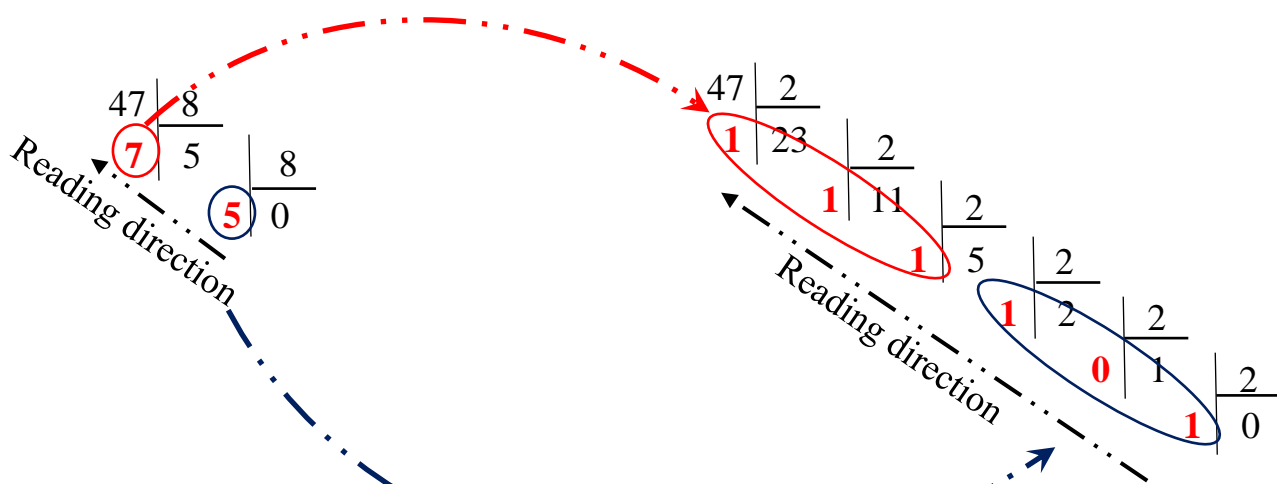
- $(47)_{10} = (57)_8$

- $(47)_{10} = (101111)_2$

- So $(57)_8 = (\ 101\,111\ )_2$

  5   7

We can observe that after 3 divisions in binary, we obtain the same quotient as after a single division in octal. Furthermore, the first remainder obtained in octal can be directly related to the first three remainders in binary.

- $(111)_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 1 \times 4 + 1 \times 2 + 1 \times 1 = (7)_8$

And the same goes for the following octal character:

- $(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1 \times 4 + 0 \times 2 + 1 \times 1 = (5)_8$

This property of equivalence between each octal digit and each group of 3 binary digits comes from the fact that 8 is a power of 2: $8 = 2^3$. It allows us to easily convert between an octal base system and a binary base system, and vice versa.

Example of binary to octal and octal to binary conversion:

Binary ($\underline{101}$ $\underline{111}$ $\underline{100}$ $\underline{001}$)$_2$

Octal (5   7   4   1)$_8$

Octal   ($\underline{3}$ $\underline{6}$ $\underline{2}$)$_8$

Binary (011 101 111)$_2$

3.4. The relationship between Binary and Hexadecimal Numbers

If we need to obtain the hexadecimal expression of a number expressed in decimal, we must always follow the method of successive division, this time by 16 (as we did for converting to bases 2 and 8) until the quotient obtained is equal to 0. The remainders of these divisions, read from bottom to top, represent the hexadecimal number (taking into account that remainders from 10 to 15 are coded as A to F).

The equivalence property we saw earlier between binary and octal in *section 3.3* also exists between hexadecimal and binary, as 16 is also a power of 2: $16 = 2^4$. Therefore, the rule is the same, but we will work in groups of 4 binary digits now instead of 3.

Binaire   (1101 0000 1100)$_2$

Hexadécimal   (**D**   **0**   **C**   )$_{16}$

Hexadécimal (1   A   F   3   )$_{16}$

Binaire   (0001 1010 1111 0011)$_2$

Remarks:

1: For the conversion of any whole number from base 10 to any other base, we always proceed with successive divisions. We divide the number to be converted by the base we want to convert it into, then divide the quotient obtained by the base again, and so on until we get a quotient of zero. The sequence of remainders obtained corresponds to the digits in the target base.

*2: For the conversion of the decimal fractional part to its equivalent octal (or hexadecimal), we always proceed with successive multiplications, just as we did for the decimal-fractional to binary conversion in **section 3.2.1**. We multiply the fractional part by 8 (or 16), the whole number part of the result becomes part of the octal (or hexadecimal) fractional part. Then, the fractional part of the result from the multiplication is multiplied again by 8 (or 16), and so on until we obtain a result equal to 0.00.*

Example : Let's convert the decimal number 418,265625 to octal:

- $418 \div 8 = 52$ <u>Remainder</u> 2 $0,265\,625 \times 8 = \mathbf{2}.125$
- $52 \div 8 = 6$ <u>Remainder</u> 4 $0,125 \times 8 = \mathbf{1}.00$
- $6 \div 8 = 0$ <u>Remainder</u> 6 $0,00 \times 8 = \mathbf{0}.00$

So :

$(418,265\,625)_{10} = (642,21)_8$

(In the same way, we proceed for the conversion of the decimal fractional part to its hexadecimal equivalent).

4. The basic operations in binary

4.1. Binary addition and multiplication

The principle of numerical calculation is the same in positional numeral systems. Therefore, we reason in the same way as we do to perform operations in the decimal system, where we are accustomed to carrying out our daily arithmetic operations.

Binary Addition

Let's first review the familiar decimal addition. Addition of 2 decimal numbers is carried out using a 3-step algorithm:

Step 1 : Add the rightmost digits (first column),

Step 2 : Note the unit digit of this sum in the same column as before, and if this sum exceeds 9, carry over the tens digit to the next column,

Step 3 : If there are other columns, repeat the previous 2 steps, making sure to add the carry-over until there are no more columns,

In binary, the algorithm is the same, except that the carry-over will occur when the sum exceeds 1 (instead of 9), as follows:

0 + 0 = 0   0 + 1 = 1   1 + 0 = 1   1 + 1 = 0 with a carry-over of

1 + 1+ 1 = 1 with a carry-over of 1

Example : let's evaluate the binary sum:111+101

```
      1 1
    1 1 1
  + 1 0 1
  ─────────
  1 1 0 0
```

a. Binary Multiplication

In binary, multiplication can be summarized as multiplying numbers by digits followed by shifted additions, just like in decimal. In fact, in binary, it's even simpler since multiplication by 0 or 1 results in 0 or the number itself (no multiplication tables to memorize as in decimal!).

Let's illustrate with an example: evaluate the binary product: 1101011×10110.

It comes down to doing this if we proceed with adding the terms one by one, without forgetting to take the carry (shift) into consideration:

So : $1101011 \times 10110 = 100100110010$

```
            1 1 0 1 0 1 1
        X       1 0 1 1 0
      _____
            0 0 0 0 0 0 0
    +     1 1 0 1 0 1 1 .
    +   1 1 0 1 0 1 1 . .
    + 0 0 0 0 0 0 0 . . .
    + 1 1 0 1 0 1 1 . . . .
      _____
      1 0 0 1 0 0 1 1 0 0 1 0
```

Note: For the multiplication of fractional numbers, the rule is the same as in decimal.

Example: Let's evaluate the binary product: $11{,}01 \times 101{,}1$

```
            1 1 , 0 1
        X   1 0 1 , 1
      _____
              1 1 0 1
    +       1 1 0 1 .
    +     0 0 0 0 . .
    +   1 1 0 1 . . .
      _____
      1 0 0 0 1 1 , 1 1 1
```

b. Binary Subtraction

The subtraction is performed following the principle of borrowing, just like in decimal.

0 - 0 = 0   1 - 1 = 0   1 - 0 = 1   0 - 1 = 1 with a borrow of 1 in the next column.

Example: Let's evaluate the following subtractions:

```
      0                      0 1 1                  0 0   0 1   0 0
   1 1 1 0 1                1 1 0 0 0              1 1 0 1 , 0 0 1 1 0
 -   1 0 1 1              - 1 0 0 1 1            -   1 1 0 , 1 1 0 1 1
 _____              _____            _____
   1 0 0 1 0                  1 0 1                1 1 0 , 0 1 0 1 1
```

Remark:

1: In the case of fractions, it is necessary to vertically align the decimal points before starting the subtraction operation.

2: When a difference of 0 - 1 appears in a column, we borrow from the first nonzero column to the left, and all the 0's just before it become 1's.

c. Binary Division

It involves successive multiplications and subtractions, just like in decimal. In the case of fractional numbers, we first move the decimal point, and then perform the division operation.

Example: let's make the divisions:

$1010001 \div 11$ and $111{,}00001 \div 1{,}01$,   we have:

```
1 0 1 0 0 1 | 1 1
  1 0 0     |
    1 0     | 1 1 0 1 1
    1 0 0   |
      1 1   |
        0   |
```

```
1 1 1 0 0,0 0 1 | 1 0 1
    1 0 0       |
    1 0 0 0     | 1 0 1,1 0 1
      1 1 0     |
        1 0     |
        1 0 1   |
```

*Note: We have studied these operations from a purely arithmetic perspective, but from a **'machine structure'** point of view, there can be some issues that may lead to incorrect results. For example, if we are working with 6 bits, the following addition: 111001 + 010010 provides a result of 7 bits: 1001011, causing the leftmost 1 to be lost! This is referred to as "overflow." Therefore, there should be an overflow indicator, and the error must be signaled.*