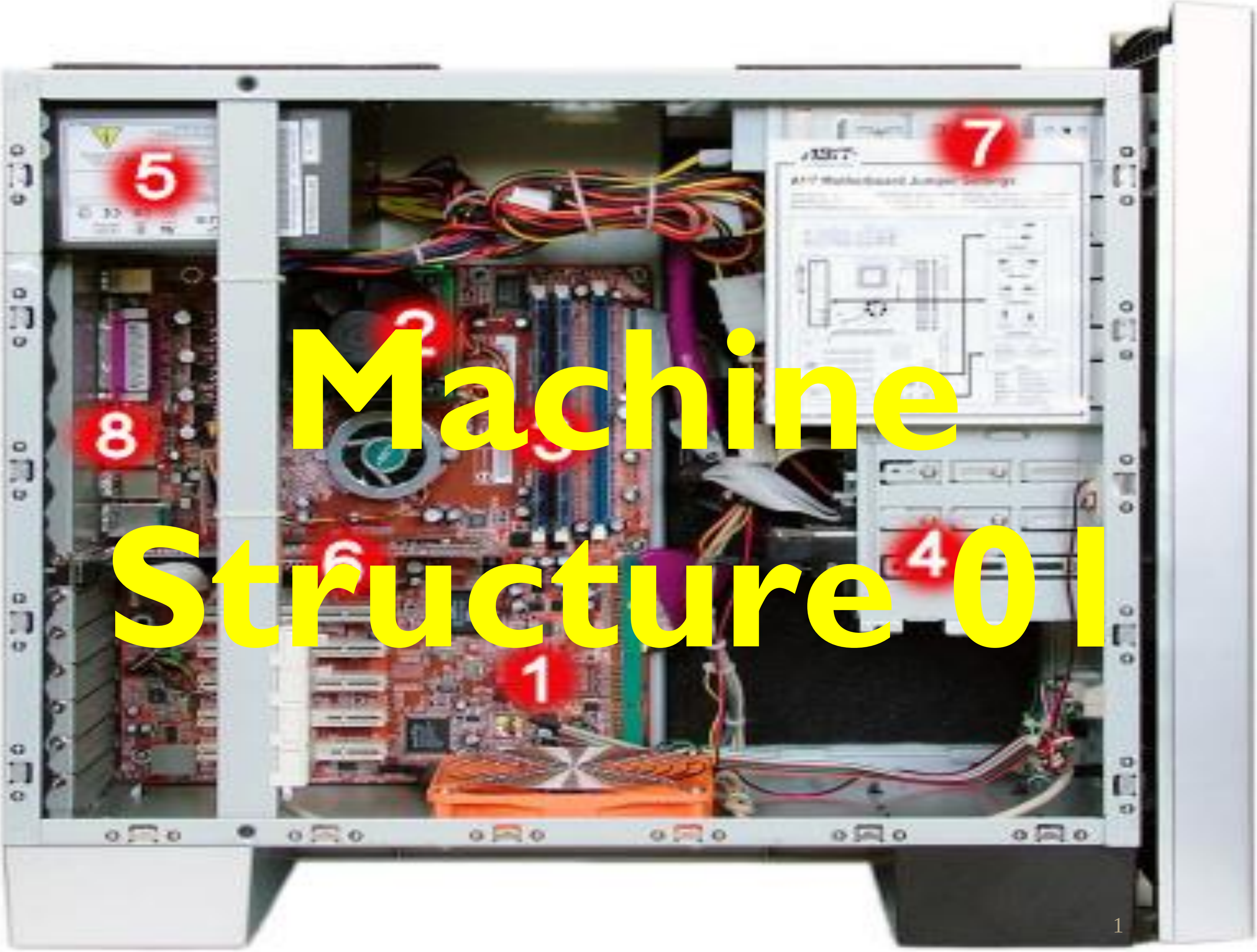
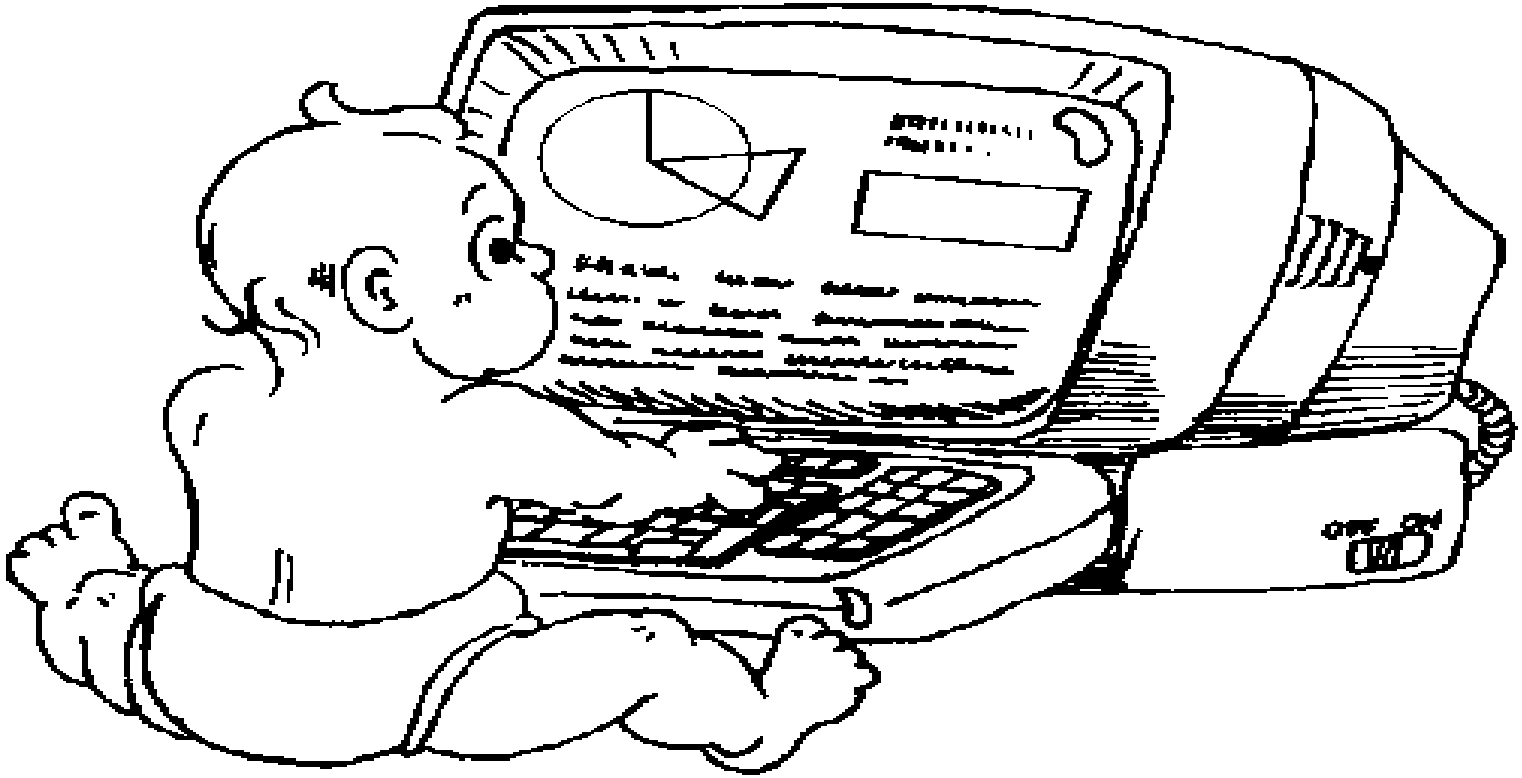
Machine Structure 01



Aim of this course

- **Not to turn you into machine structure experts**

- **Rather, to give you a general knowledge and tools for your future profession!**



Chapter 01

Number systems and information representation

Number systems and information representation

- Introduction
- Decimal system
- Binary, octal and hexadecimal systems
- Converting from one numbering system to another
- Arithmetic operations in binary, octal and hexadecimal
- Floating-point numbers
- Character coding

- **If you don't understand something…**

Don't hesitate to speak up!

Objectives of the chaptre

- **Understanding what a numbering system is.**
- **Learn how to convert from one system to another.**
- **A Learn to perform arithmetic operations in binary.**

Introduction

- We have become accustomed to representing numbers using ten different symbols :   **0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9**
- This system is called the decimal system (deci means ten).
- There are, however, other forms of numeration for machine that operate using a number of distinct symbols.

  Example :

  - binary system (bi : two),
  - octal system (oct : eight),
  - hexadecimal system (hexa : sixteen).
- In fact, any number of different symbols can be used (not necessarily numbers; letters can be used).
- In a numeration system: the number of distinct symbols is called the base of the numeration system..

Decimal system

- Ten different symbols (numbers) are used : **0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9**
- Any combination of the symbols **0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9** gives us a number.
- **Example :**

$$\boxed{\mathbf{2}}\ \mathbf{3\ 3\ 4\ 5\ 6}\ \boxed{\mathbf{7}}$$

High weight **Low weight**

$$\boxed{\mathbf{345}}\,\mathbf{,}\,\boxed{\mathbf{567}}$$

Whole part **Fractional part**

Polynomial expansion of a number in the decimal system

- Given the number 1978, this number can be written in the following form:

$$1978 = 1000 + 900 + 70 + 8$$

$$1978 = 1*1000 + 9*100 + 7*10 + 8*1$$

$$1978 = 1*10^3 + 9*10^2 + 7*10^1 + 8*10^0$$

This forma is called **the polynomial form**.

A **real number** can also be written in **polynomial form**

$$1978{,}265 = 1*10^3 + 9*10^2 + 7*10^1 + 8*10^0 + 2*10^{-1} + 6*10^{-2} + 5*10^{-3}$$

Decimal counting

- On a single position : 0 ,1,2,3,4,5,….9 = 10^1-1
- On two positions : 00 , 01,02, …..,99 = 10^2-1
- On three positions : 000,001,……,999 = 10^3-1

- On n positions : minimum 0

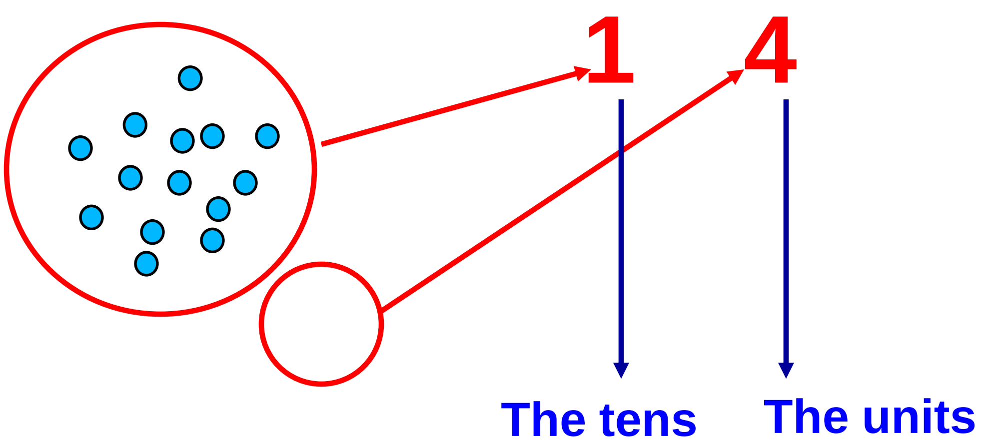
  maximum 10^n-1

  number of combinations 10^n

Binary system
(base-2 system)

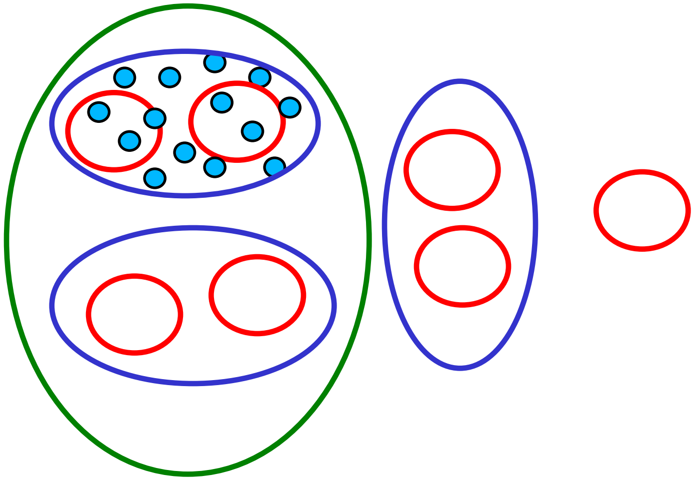
An illustrative example

- Let's suppose we have 14 tokens.
- If we form groups of 10 tokens.
- We'll get only 1 group of 10 and have 4 tokens left..



We still have 14 tokens

- **Now we're going to form groups of 2 tokens: we get 7 groups**
- **The 7 groups are then grouped 2 by 2 to form 3 groups.**
- **These are also grouped 2 by 2 : we obtain only 1 group**



Number of tokens remaining outside the groups : **0**
Number of groups containing **2** tokens : **1**
Number of groups containing **2** groups of **2** tokens : **1**
Number of groups containing groups of **2** groups of **4** tokens : **1**

If we combine the various figures, we obtain : 1110
1110 is the representation of 14 in base 2

Binary system

- In the binary system, **only 2 symbols** are used to express any value: **0** and **1**.

The bit $\boxed{1}101_2$ The base

$\boxed{1}\ 1\ 0\ \boxed{1}$

The most significant bits

The least significant bit

A number in **base 2** can also be written in **polynomial form** :

$$1110_2 = 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0$$

$$1110{,}101_2 = 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$$

Binary counting

- **On a single bit : 0 , 1**
- **On 3 Bits : 8 combinations =** 2^3

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

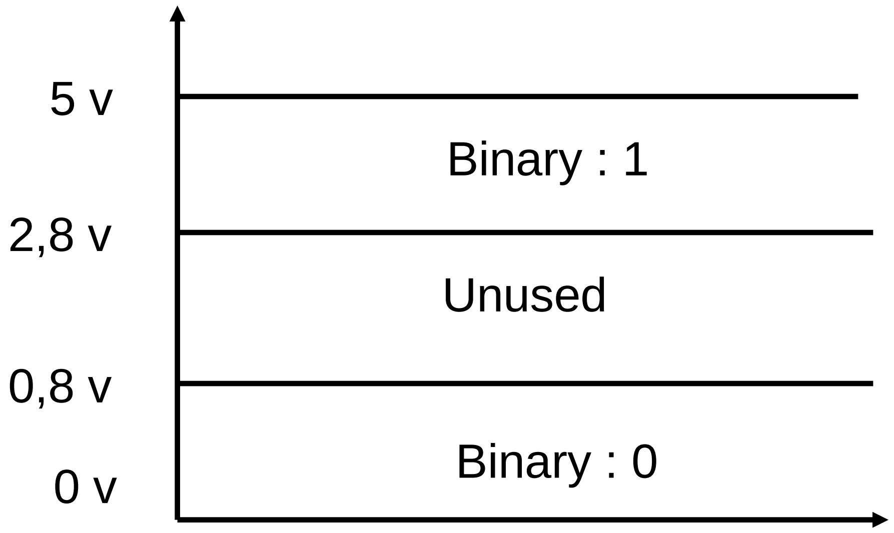
- **On 2 bits : 4 combinations =** 2^2

Binary	Decimal
00	0
01	1
10	2
11	3

What is the system used in digital devices ?

. Digital machines use the binary system.
. In the binary system: only 2 symbols are used: 0 and 1.
. It's easy to represent these two symbols in digital machines.
. The 0 and 1 are represented by two voltages.

Binary	Voltage
0	0 V
1	5 V



The octal system (base 8)

- **Eight (08) symbols are used in this system :**

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

Example 1 :

$$127_8 = 1*8^2 + 2*8^1 + 7*8^0$$

$$127{,}65_8 = 1*8^2 + 2*8^1 + 7*8^0 + 6*8^{-1} + 5*8^{-2}$$

Example 2 :

The number (**1289**) does not exist in base 8, since symbols **8** and **9** **do not belong to base 8**.

The hexadecimal system (base 16)

- **Sixteen (16) different symbols are used in this system :**

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Example :

$\mathbf{BAC}_{16} = B*16^2 + A*16^1 + C*16^0$

$\mathbf{FAC}_{16} = F*16^2 + A*16^1 + C*16^0$

Summary

- In a **base X** : **X distinct symbols** are used to represent numbers..
- The value of each symbol must be strictly less than the **base X**.
- Each number in a **base X** can be written in its polynomial form.

Converting base X to base 10

- This conversation is quite simple since you just have to do the **polynomial expansion** of this number in the **base X**, and do **the sum** afterwards..

Example :

$$A3C_{16} = 10*16^2 + 3*16^1 + 12*16^0 = 2620_{10}$$

$$101_2 = 1*2^2 + 0*2^1 + 1*2^0 = 5_{10}$$

$$43_5 = 4*5^1 + 3*5^0 = 23_{10}$$

Exercise

- **Perform the following transformations to base 10 ?**

$(123)_6 = (?)_{10}$

$(45,76)_8 = (?)_{10}$

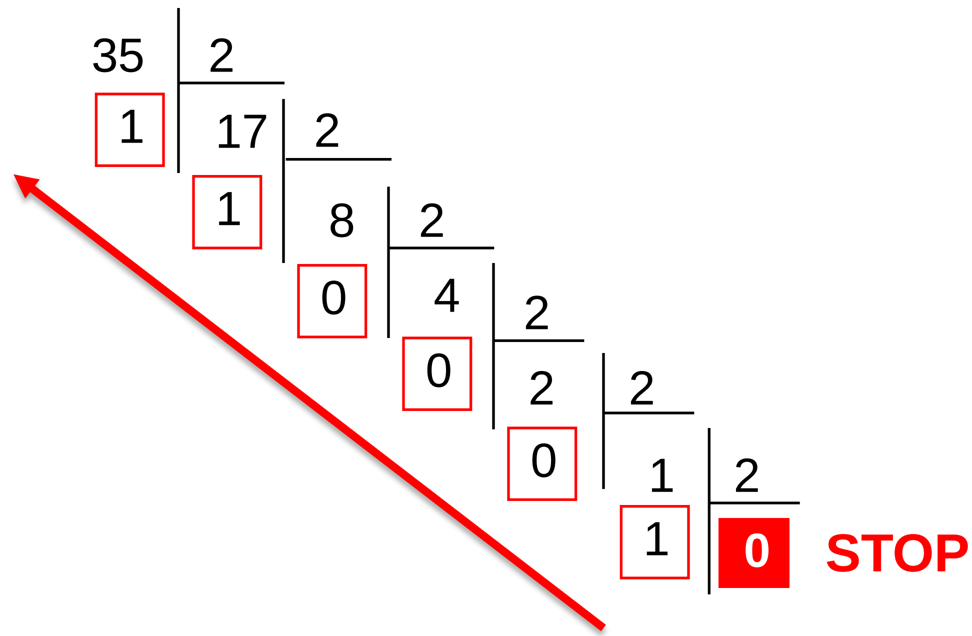
$(1100,11)_2 = (?)_{10}$

$(1ABC)_{16} = (?)_{10}$

Converting from base 10 to base 2

The principle is to make **successive divisions** of the number by 2 until a **quotient of zero** is obtained, and take the remainders of the divisions in **reverse order**.

Example 1 : $(35)_{10}=(\mathbf{?})_2$



A After division, we obtain :

$\mathbf{(35)_{10} = (100011)_2}$

Converting from base 10 to base 2 : case of a real number

- A real number is made up of two parts: the integer part and the fractional part. We treat each part separately.

- **The integer part is transformed by successive divisions by 2.**

- **The fractional part is transformed by successive multiplication by 2.**

Converting from base 10 to base 2 : case of a real number

Example : 35,625 = $(?)_2$

I.P = 35 = $(100011)_2$

F.P = 0,625 = $(?)_2$

$$0{,}625 \times 2 = \boxed{1},\underbrace{250}$$

$$0{,}250 \times 2 = \boxed{0},\underbrace{500}$$

$$0{,}500 \times 2 = \boxed{1},\underbrace{000}$$

STOP

(0,625) = **$(0{,}101)_2$**

Result : 35,625 = $(100011{,}101)_2$

Converting from base 10 to base 2 : case of a real number

$(0{,}6)_{10} = (?)_2$

0,6 * 2 = 1,2

0,2 * 2 = 0,4

0,4 * 2 = 0,8

0,8 * 2 = 1,6

$(0{,}6) = (0{,}1001)_2$

Note :

The number of bits after the decimal point will determine precision.

Exercise :

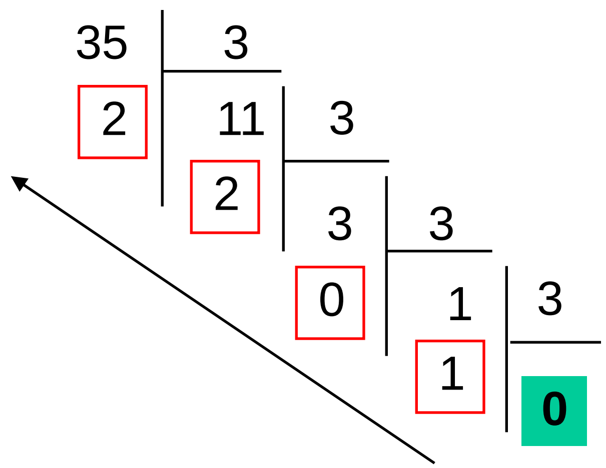
Perform the following transformations : $(23{,}65) = (?)_2$

$(18{,}190) = (?)_2$

Converting from decimal to base X

- The conversion is done by taking the remainders of **successive divisions** on the **base X** in the opposite direction

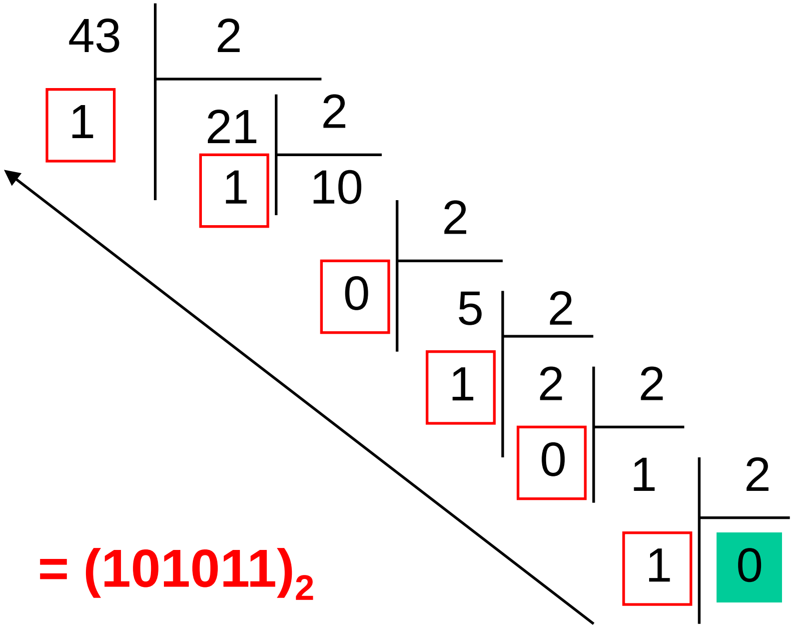
Example : $35 = (?)_3$



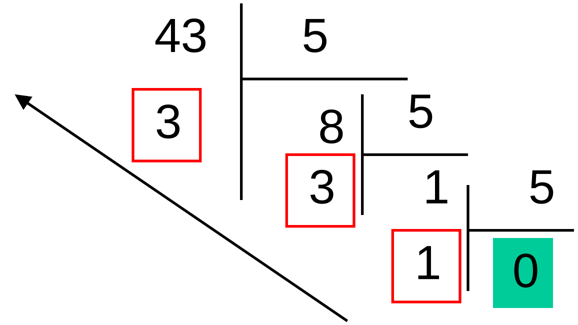
So : $35 = (1022)_3$

- **Question :** Perform the following transformations :

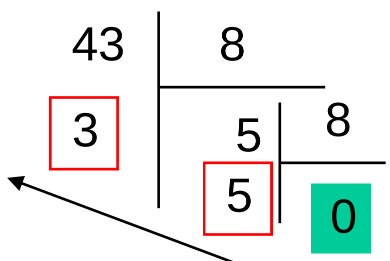
$$(43)_{10} = (?)_2 = (?)_5 = (?)_8 = (?)_{16}$$



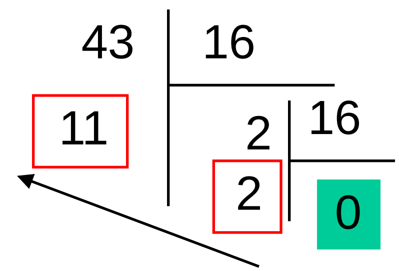
= $(101011)_2$



= $(133)_5$



= $(53)_8$



= $(2B)_{16}$

Conversion : hexadcimal → binary

- **Each symbol of the Hexadecimal base is written on 4 bits in binary**
- **The basic idea is to replace each Hexadecimal symbol by its 4-bit binary value ($16 = 2^4$)**

$0_{16} \rightarrow 0000_2$	$8_{16} \rightarrow 1000_2$
$1_{16} \rightarrow 0001_2$	$9_{16} \rightarrow 1001_2$
$2_{16} \rightarrow 0010_2$	$A_{16} \rightarrow 1010_2$
$3_{16} \rightarrow 0011_2$	$B_{16} \rightarrow 1011_2$
$4_{16} \rightarrow 0100_2$	$C_{16} \rightarrow 1100_2$
$5_{16} \rightarrow 0101_2$	$D_{16} \rightarrow 1101_2$
$6_{16} \rightarrow 0110_2$	$E_{16} \rightarrow 1110_2$
$7_{16} \rightarrow 0111_2$	$F_{16} \rightarrow 1111_2$

Conversion : hexadecimal → binary

Replace each hexadecimal digit with its binary equivalent (in 4 digits) :

$$A\ 4\ 8\ C_{16}$$

$$1010\ 0100\ 1000\ 1100_{2}$$

Note : we can remove the 0 at the beginning of the number

Example : $39D_{16} = 11\ 1001\ 1101_{2}$

$(345B)_{16}=(0011\ \ 0100\ \ 0101\ \ 1011)_{2}$

$(AB3,4F6)_{16} = (\ 1010\ \ 1011\ \ 0011\ ,\ 0100\ \ 1111\ \ 0110\ )_{2}$

Conversion : binary → hexadecimal

- The basic idea is to make groupings of 4 bits starting from the least significant (starting from the end of the number).
- Then replace each grouping with the corresponding Hexadecimal value.

$$\underbrace{1010}_{A}\ \underbrace{0100}_{4}\ \underbrace{1000}_{8}\ \underbrace{1100}_{C}{}_2$$

$$1010010010001100_2 = A48C_{16}$$

Note :

Grouping is from right to left for the integer part and from left to right for the fractional part

Conversion : binary → hexadecimal

Example :

$$\underbrace{0011}_{3}\ \underbrace{1001}_{9}\ \underbrace{1011}_{B}\ {}_2$$

$1110011011_2 = 39B_{16}$

Example :

$(11001010100110)_2 = (\underline{0011}\ \ \underline{0010}\ \ \underline{1010}\ \ \underline{0110})_2 = (32A6)_{16}$

$(110010100{,}10101)_2 = (\underline{0001}\ \ \underline{1001}\ \ \underline{0100}\ ,\ \underline{1010}\ \ \underline{1000})_2 = (194{,}A8)_{16}$

Conversion : octal → binary

- Each **symbol in the octal base** is written on **3 bits in binary**.
- The basic idea is to replace each **symbol in the octal base** with its value **in 3-bit binary** **($8 = 2^3$).**

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

Examples :

$(345)_8=(\underline{011}\ \ \underline{100}\ \ \underline{101})_2$

$(65,76)_8=(\underline{110}\ \ \underline{101}\ ,\ \underline{111}\ \ \underline{110})_2$

$(35,34)_8=(\underline{011}\ \underline{101}\ ,\ \underline{011}\ \ \underline{100})_2$

Conversion : binary → Octal

- The basic idea is **to group 3 bits** together starting from **the least significant bit**.
- Then **replace each grouping** with the corresponding **octal value**.

Example :

$(11001010010110)_2 = (\underline{011}\ \underline{001}\ \underline{010}\ \underline{010}\ \underline{110})_2 = (31226)_8$

$(110010100{,}10101)_2 = (\underline{110}\ \underline{010}\ \underline{100}\ ,\ \underline{101}\ \underline{010})_2 = (624{,}52)_8$

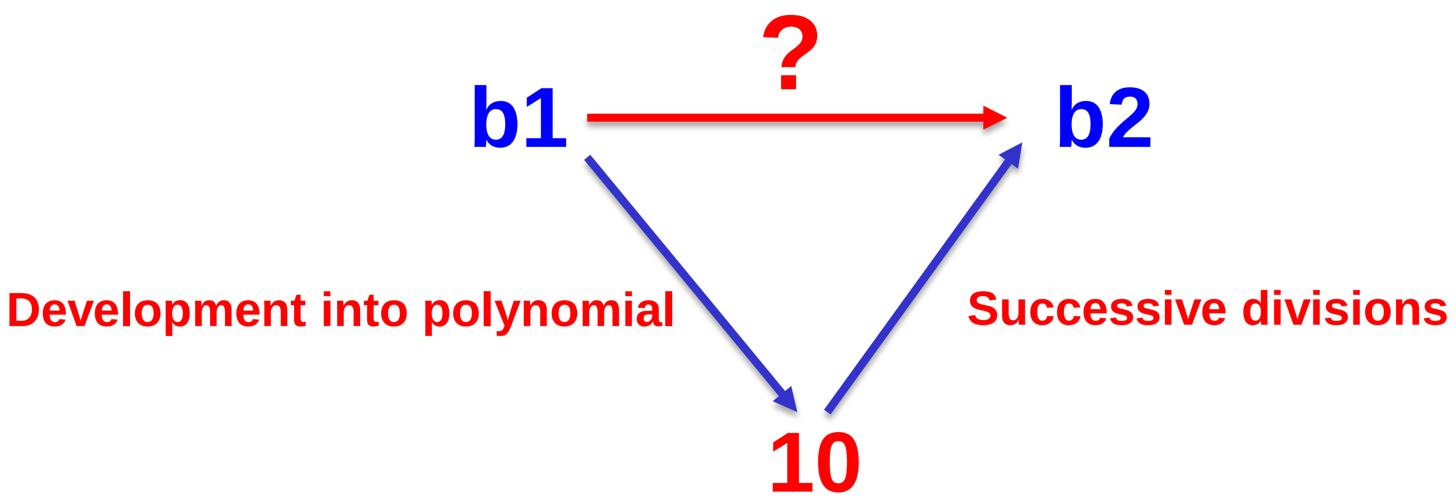
Note :

Grouping is from right to left for the integer part and from left to right for the fractional part.

Converting from base b1 to base b2

- There is no method for switching from one **base b1** to another **base b2** directly.
- The idea is to convert the number from **base b1** to **base 10** Next…

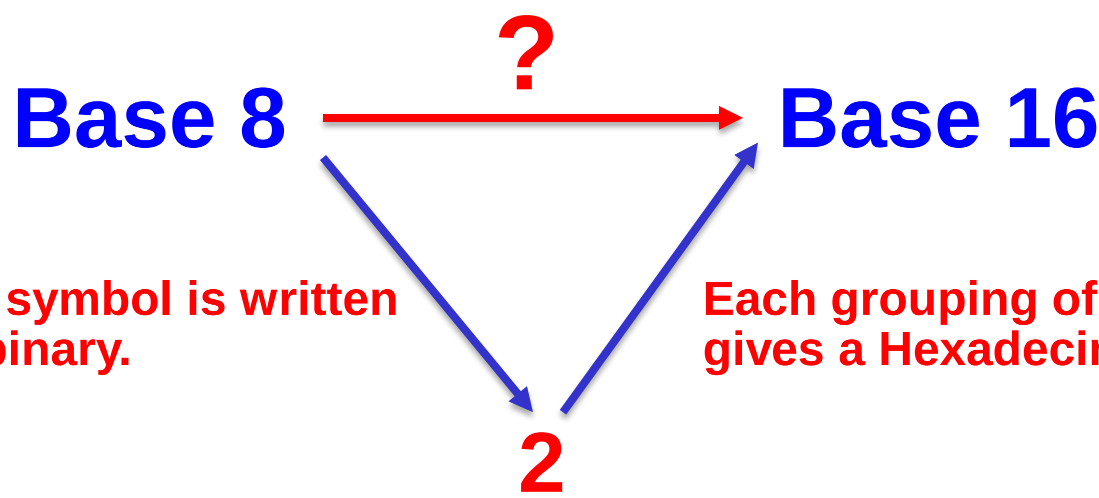
convert the result from **base 10** to **base b2**



Converting from base b1 to base b2

- If **b1** and **b2** are both **powers of 2**
- In this case, the number must be **converted** from **base b1** to **base 2**, Next…

convert the result from **base 2** to **base b2**



Each base-8 symbol is written on 3 bits in binary.

Each grouping of 4 binary bits gives a Hexadecimal symbol

Example :

$$(34)_5 = (?)_7$$

$\mathbf{34_5} = 3*5^1 + 4*5^0 = 15 + 4 = \mathbf{19_{10}}$

$\mathbf{19_{10}} = (\mathbf{?})_{\mathbf{7}}$

$$\begin{array}{r|ll} 19 & 7 & \\ \hline \boxed{5} & 2 & \multicolumn{1}{|l}{7} \\ \cline{3-3} & \boxed{2} & \multicolumn{1}{|l}{0} \end{array}$$

$\mathbf{19_{10} = 25_7}$ **So** $\mathbf{34_5 = 25_7}$

Exercise : perform the following transformations

$(43)_6 = (?)_5 = (?)_8$

$(2A)_{16} = (?)_9$

Binary coded decimal numbers (BCD)

- Conversion to natural binary is a time-consuming and tedious operation. To overcome this drawback, we use **the BCD code**, which has the advantages of both decimal and binary..
- In **the BCD code**, we **associate a four-bit** code word with each **symbol of the decimal** system.

Example : le nombre décimal **317,25** s'écrira en **BCD (8421)**

3	1	7,	2	5
↓	↓	↓	↓	↓
0011	0001	0111,	0010	0101

Binary coded decimal numbers (BCD)

- There are several **BCD codes** : the **8421**, the **6311**, **excess-3**, **2 out-of 5** and **code gray** (reflected binary).

Decimal	***BCD8421***	***BCD6311***	***Excess-3***	***2 out-of 5***	***Code Gray***
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	0111
6	0110	1000	1001	10001	0101
7	0111	1001	1010	10010	0100
8	1000	1011	1011	10100	1100
9	1001	1100	1100	11000	1101

Binary arithmetic

The same calculation rules apply to all numbering systems; binary arithmetic is similar to decimal arithmetic.

Base 2 addition :

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline \mathbf{0} \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline \mathbf{1} \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline \mathbf{1} \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline \mathbf{1\ 0} \end{array}$$

I set 0 and I retain 1

Example :

$$\begin{array}{rcccccccc} & & ^{} & & & & 1 & 1 & \\ & & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ + & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline = & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{array}$$

Binary arithmetic

Subtraction in base 2 :

0 - 0 = 0

1 - 0 = 1

1 - 1 = 0

0 - 1 => I lend a **1 to 0** which becomes **10**, without forgetting to subtract the **1** from the number on the left..

Example :

$$\begin{array}{rccc} & 1 & {}^{1}1 & {}^{1}0 \\ - & {}_{1}0 & {}_{1}1 & 1 \\ \hline & 0 & 1 & 1 \end{array} \qquad \begin{array}{rcccc} & 1 & {}^{1}0 & 1 & 0 \\ - & {}_{1}0 & 1 & 0 & 0 \\ \hline & 0 & 1 & 1 & 0 \end{array}$$

Binary arithmetic

Multiplication in base 2 :

Example :

```
      1 0 1 1
   x    1 0 1
  ------------
      1 0 1 1
  1 0 1 1 . .
 -------------
  1 1 0 1 1 1
```

Binary arithmetic

Division in base 2 :

Example :

110111	101
101	1011
....	
00111	
....	
0101	
000	

Or :

21 - 7 = 14

14 - 7 = 7

7 - 7 = 0

three operations the rest **"0"** so the result is : **3**.

Exercise

- Perform the following operations and convert the result to a decimal each time :

  - ✓ $(1101,111)_2+(11,1)_2=(?)_2$
  - ✓ $(43)_8+(34)_8=(?)_8$
  - ✓ $(4B)_{16}+(F4)_{16}=(?)_{16}$
  - ✓ $(AB1)_{16}-(2E7)_8=(?)_{16}$

Representation of negative integers

The signed integers

- **To use an adder as a subtractor, you need a suitable representation of negative numbers.**
- **The simplest way to represent the sign is to reserve one binary digit for it, with the other bits representing the absolute value of the number.**
- **La The convention used is to set the first bit (the strongest bit) to :**

  - **0 : to find a positive number**
  - **1 : for a negative number**

- **Example :**
  - $(0\ 0\ 0\ 1\ 1\ 0\ 1\ 1)_2 \rightarrow (27)_{10}$
  - $(1\ 0\ 0\ 1\ 1\ 0\ 1\ 1)_2 \rightarrow (-27)_{10}$

- **With this method, we can represent with 8 bits the interval of the following integers: $[-(2^7 - 1), (2^7 - 1)]$ either $[-127, +127]$**

- **More generally, on n bits, we have the interval : $[-(2^{n-1} - 1), (2^{n-1} - 1)]$**

Representation of negative integers

Les entiers signés

Inconvenience : such a representation would require special sign processing, and different electronic circuits depending on whether you wanted to add or subtract.

$$
\begin{array}{rcr}
7 & & \mathbf{0}\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \\
+\ (-5) & & +\ \mathbf{1}\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\
\text{---------} & & \text{--------------------} \\
+\ \ 2 & \neq & \left.1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\right\}\ (-12)_{10}
\end{array}
$$

This result is false !

Solution : Use of another form of negative number representation, known as complement representation.

Representation of negative integers

Representation by complement

So there are 2 ways of representing negative numbers:

- **Representation by ones' complement**

$$\overline{N} = \left(2^n - 1\right) - N$$

- **Representation by two's complement**

$$N^* = 2^n - N$$

Representation of negative integers

Representation by complement

1 – Representation by ones' complement

- The 1's complement of a binary number is obtained by simply inverting the values of the bits making up the number : bits at 1 change to 0 and bits at 0 change to 1.

- **Example :** Or the number coded on one byte (n = 8)

0 0 1 1 0 0 1 0

C1 1 1 0 0 1 1 0 1

Representation of negative integers

Representation by complement

2 – Representation by two's complement

- The 2's complement of a binary number is obtained by **adding 1** to the **1's complement**, obtained using the method previously explained.

Example : the number $(-74)_{10}$ coded on one byte (n = 8)

(+74) in binary : 0 1 0 0 1 0 1 0

(c à1)

One's complément : 1 0 1 1 0 1 0 1

\+ 1

Two's complément : 1 0 1 1 0 1 1 0

1	0	1	1	0	1	1	0
Sign	Number representation						

$(-74)_{10}$ will coded **$(1\ 0\ 1\ 1\ 0\ 1\ 1\ 0)_2$**

But $(74)_{10}$ will always be coded **$(0\ 1\ 0\ 0\ 1\ 0\ 1\ 0)_2$**

Representation of negative integers

Representation in complement

2 – Representation by two's complement

Codage in two’s complement

To illustrate the principle of **2's complement coding**, useful for distinguishing between positive and negative numbers, let's take the following absurd example. Assume you're driving a car with a speedometer reading **00000 Km**.

- If you set off in forward gear, after 1km the counter will read **00001Km** then **00002Km**, etc. ...

Representation of negative integers

Representation in complement

2 – Representation by two's complement

Codage in two's complement

To illustrate the principle of **2's complement coding**, useful for distinguishing between positive and negative numbers, let's take the following absurd example. Assume you're driving a car with a speedometer reading **00000 Km**.

- If you set off in forward gear, after 1km the counter will read **00001Km** then **00002Km**, etc. ...

Representation of negative integers

Representation in complement

2 – Representation by two's complement

Codage in two's complement

Afin To illustrate the principle of **2's complement coding**, useful for distinguishing between positive and negative numbers, let's take the following absurd example. Assume you're driving a car with a speedometer reading **00000 Km**..

- If you set off in forward gear, after **1km** the counter will read **00001Km** then **00002Km**, etc. ...
- Suppose that from **00000**, we go in reverse and the counter counts down: it will show the largest possible number, i.e. **99999Km** after **1Km**, then **99998Km** after **2Km** of walking, etc. .

Representation of negative integers

Representation in complement

2 – Representation by two's complement

Codage in two’s complement

Afin To illustrate the principle of **2's complement coding**, useful for distinguishing between positive and negative numbers, let's take the following absurd example. Assume you're driving a car with a speedometer reading **00000 Km**

- If you set off in forward gear, after **1km** the counter will read **00001Km** then **00002Km**, etc. ...
- Suppose that from **00000**, we go in reverse and the counter counts down: it will show the largest possible number, i.e. **99999Km** after **1Km**, then **99998Km** after **2Km** of walking, etc. .
- We can then consider that **99999Km** means **(- 1)** and that **99998Km** means **(- 2)**, etc. . .
- But it's going to be simpler once it's transposed into binary: this time, our odometer is graduated in binary and starts from **00000000** :
- If you're driving forwards, it will indicate **00000001** after **1Km,** then **00000010** after **2Km**, etc. ... .
- When reversing, it marks **11111111** after **1Km**, then **11111110** after **2Km**, etc. . . .

Representation of negative integers

2 – Two's complément : Problems related to the length of numbers

- **1st case :** No overflow, Negative result

Example : (-64) + (16) = (- 48)

```
  (- 64)          1 1 0 0 0 0 0 0
+   (16)        + 0 0 0 1 0 0 0 0
-----------     -----------------
  (- 48)        = 1 1 0 1 0 0 0 0
```

```
  0 1 0 0 0 0 0 0   (+64)
  -----------------------
  1 0 1 1 1 1 1 1   (c1)
+               1
  ---------------
  1 1 0 0 0 0 0 0   (c2)
```

Negative result
the 1st bit is at 1

```
  1 1 0 1 0 0 0 0   (c2)
-               1
  ---------------
  1 1 0 0 1 1 1 1   (c1)
  0 0 1 1 0 0 0 0
```

|(- 48)| : 0 0 1 1 0 0 0 0

Result is : (- 48)

Representation of negative integers

2 – two's complement : **Problems related to the length of numbers**

- **2nd case :** Overflow, Positive result. In case of overflow, we ignore it.

Example : 64 + (-16) = (+48)

(64)	**0 1 0 0 0 0 0 0**
+ (- 16)	+ **1 1 1 1 0 0 0 0**

(+ 48)	**0 0 0 1 1 0 0 0 0**

0 0 0 1 0 0 0 0	(+16)
1 1 1 0 1 1 1 1	(c1)
+	**1**
-----------------------	
1 1 1 1 0 0 0 0	**(c2)**

Positive result
the 1^{st} bit is at 0

Résult is : (+ 48)

The holdback
Ignore the holdback : outside 8 bits

Representation of negative integers

2 - two's complement : **Problems related to the length of numbers**

3rd case : Overflow, Negative result

Example : (- 64) + (-16) = (- 80)

```
   ( - 64)        1 1 0 0 0 0 0 0
+  ( - 16)     +  1 1 1 1 0 0 0 0
-----------     ------------------
   ( - 80)        1 0 1 1 0 0 0 0
```

	0 1 0 0 0 0 0 0	(+64)
	1 0 1 1 1 1 1 1	(c1)
+	1	
	----------------------	
	1 1 0 0 0 0 0 0	(c2)

	0 0 0 1 0 0 0 0	(+16)
	1 1 1 0 1 1 1 1	(c1)
+	1	
	----------------------	
	1 1 1 1 0 0 0 0	(c2)

	1 0 1 1 0 0 0 0	(c2)
-	1	
	----------------------	
	1 0 1 0 1 1 1 1	(c1)
	0 1 0 1 0 0 0 0	

Negative result
the 1st bit is at 1

Ignore the holdback

|(-80)| : 0 1 0 1 0 0 0 0

Result is : (- 80)

Floating point numbers

Coding real numbers

The formats for representing real numbers are :

1. Fixed point format :

$54{,}25_{(10)}$; $10{,}001_{(2)}$; $A1{,}F0B_{(16)}$

2. Floating point format :

$0{,}5425 \;.\; 10^{2}{}_{(10)}$; $10{,}1 \;.\; 2^{-1}{}_{(2)}$; $A0{,}B4.16^{-2}{}_{(16)}$

Floating point numbers

Floating-point numbers are generally **non-integer numbers** whose digits are **written only after the decimal point, and to which an exponent is added to specify how many positions the decimal point must be moved**. **For example**, **in decimal notation**, we write the number **31.41592** as **0.3141592×10^2** and the number **- 0.01732** as **-0.1732×10^{-1}**.

In both cases, **the mantissa** is the number following the decimal point (**3141592** in the first case, and **1732** in the second).

Floating point numbers

There are therefore several ways of representing the exponent and mantissa

Example: base-2 coding, floating-point format of (3,25)

$$
\begin{aligned}
3{,}25_{(10)} &= 11{,}01_{(2)} \qquad (\text{ Fixed point })\\
&= 1{,}101 \,.\, 2^{1}{}_{(2)}\\
&= 110{,}1 \,.\, 2^{-1}{}_{(2)}
\end{aligned}
$$

In computing, a standard has been established for the representation of floating-point numbers. This is the IEEE 754 standard.

The IEEE 754 standard

Dans In **the IEEE 754 standard**, a **floating-point number** is always represented by a triplet: (**s, e, m**). This triplet is assembled in the order : **sign**, **exponent**, **mantissa** to form a number.

$$N = s \,.\, m \,.\, B^{e}$$

Or

$$N = (-1)^{s} \,.\, m \,.\, B^{e}$$

Thus, for base **B = 10**, the two preceding numbers are represented by the triplets (**0, 2, 3141592**) and (**1, - 1, 1732**):

(**0, 2, 3141592**) = **0,3141592 x** $\mathbf{10^{2}}$

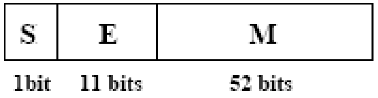
(**1, - 1, 1732**) = **- 0,1732 x** $\mathbf{10^{-1}}$

The IEEE 754 standard

The IEEE 754 standard then defines the coding, with base **B = 2**, of a number **in single-precision on 32 bits** and **in double-precision on 64 bits**. A standardized format:

1. **Single-precision format: 32 bits**
    - **Sign bit (1 bit)**
    - **Exponent (8 bits)**
    - **Mantissa (23 bits)**

2. **Double precision format: 64 bits**
    - **Sign bit (1 bit)**
    - **Exponent (11 bits)**
    - **Mantissa (52 bits)**



The IEEE 754 standard

s : is the **sign** coded on 1 bit (MSB is the most significant bit): **0** for a **positive number**, **1** for a **negative number**;

- ***e***: designates the **exponent**, which is an integer coded on $\mathbf{n_e = 8}$ bits for **single precision** and on $\mathbf{n_e = 11}$ bits for **double precision**. The exponent can be **positive** or **negative**. n_e represents the number of bits of the exponent:

$$e_b = e + 2^{(ne-1)} - 1$$

$e_b = e + 127$: for single precision

$e_b = e + 1023$: for double precision

- ***m***: designates the **mantissa**, which is a decimal point number coded on **23** bits in **single precision**, **52** in **double precision**. In the binary representation of m, the bit to the left of the decimal point is necessarily a 1 (m = 1.....). **It is therefore unnecessary to represent it**. Only the digits after the decimal point in the mantissa are coded in binary.

The IEEE 754 standard

Example:

- Represent 525,5 in IEEE 754 single precision format.

Write 525.5 in binary: $(525,5)_{10} = (1000001101,1)_2$
Find the exponent: $(525,5)_{10} = (1,0000011011)2 \,.\, 2^9$
Identify the triplet (s, e, m):

- $s = 0$
- $e_b = e + \underbrace{2^{n_e - 1} - 1}_{biais} = 9 + 2^{8-1} - 1 = (136)_{(10)} = (10001000)_{(2)}$
- $m = (00000110110000000000000)_{(2)}$

$$(525,5)_{(10)} = (0\ 10001000\ 00000110110000000000000)_{(2)}$$

The IEEE 754 standard

Example : Decimal conversion - IEEE 754 single precision :

$$35{,}5_{(10)} = ?_{(IEEE\ 754\ SP)}$$

$$S = 0$$

$$35{,}5_{(10)} = 100011{,}1_{(2)}$$

$$= 1{,}000111 \cdot 2^5{}_{(2)}$$

$$E = E_b - 127 = 5$$

$$E_b = 132$$

$$1{,}M = 1{,}000111 \qquad M = 00011100\ldots$$

$$\overbrace{0}^{S}\underbrace{10000100}_{E_b}\overbrace{00011100000000000000000}^{M}{}_{(IEEE\ 754\ SP)}$$

The IEEE 754 standard

Example : IEEE 754 single precision - Decimal conversion :

$$\overbrace{0}^{S}\underbrace{10000001}_{E_b}\overbrace{11100000000000000000000}^{M}{}_{(\text{IEEE 754 SP})}$$

$S = 0$, so positive number

$E_b = 129$, so exponent $= E_b - 127 = 2$

$1{,}M = 1{,}111$

$$+\ 1{,}111\ .\ 2^2{}_{(2)} = 111{,}1_{(2)} = 7{,}5_{(10)}$$

CHARACTER ENCODING

CHARACTER ENCODING

ASCII encoding

The computer's memory stores all data in **digital form**. There is no method of storing characters directly. Each character therefore has its equivalent **in binary code**: this is **the ASCII code** (American Standard Code for Information Interchange). (standardized in 1963)

This coding consists of **associating a binary numerical value** (which can be interpreted as hexadecimal, decimal, etc.) **with each character used in computer data exchange**: **alphabetic** and **numeric** (**alphanumeric**) characters, punctuation, various control codes.

CHARACTER ENCODING

ASCII encoding

basic **ASCII code** uses **one byte** to **encode 128 characters**, representing **the characters on 7 bits** (128 possible characters) and **leaving the 8th bit at 0** :

- Codes 0 to 31 are not characters. They are called control characters because they allow actions such as :
    - line feed (CR)
    - Beep (BEL)
- Codes 65 to 90 represent uppercase letters
- Codes 97 to 122 represent lower-case letters

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

CHARACTER ENCODING

Extended ASCII encoding

The ASCII code was developed for the English language, so it does not contain accented or language-specific characters. To encode such characters, another code must be used. **The ASCII code has therefore been extended** to a single byte, using the 8th bit to define characters numbered from 128 to 255 (this is known as **the extended ASCII code**).

The ISO/IEC 8859 standard provides extensions for various languages. For example, ISO 8859-1, also known as Latin-1, extends ASCII with accented characters useful for Western European languages such as French and German.

Table ASCII étendue (128 - 255)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

CHARACTER ENCODING

UNICODE

Unicode encodes characters as **sequences of 1 to 4 code** elements of one byte each. Among other things, the Unicode standard defines a character set. **Each character is identified within this set by an integer index**, also known as a code point. For example, the character **€ (euro)** is the **8365th** character in the Unicode repertoire, so its index, or code point, **is 8364**.

The Unicode directory can contain over a **million characters**, which is far too many to be encoded by a single byte. The Unicode standard therefore defines standardized methods for encoding and storing this index as a sequence of bytes: **UTF-8** is one of these, along with **UTF-16**, **UTF-32** and their various variants.

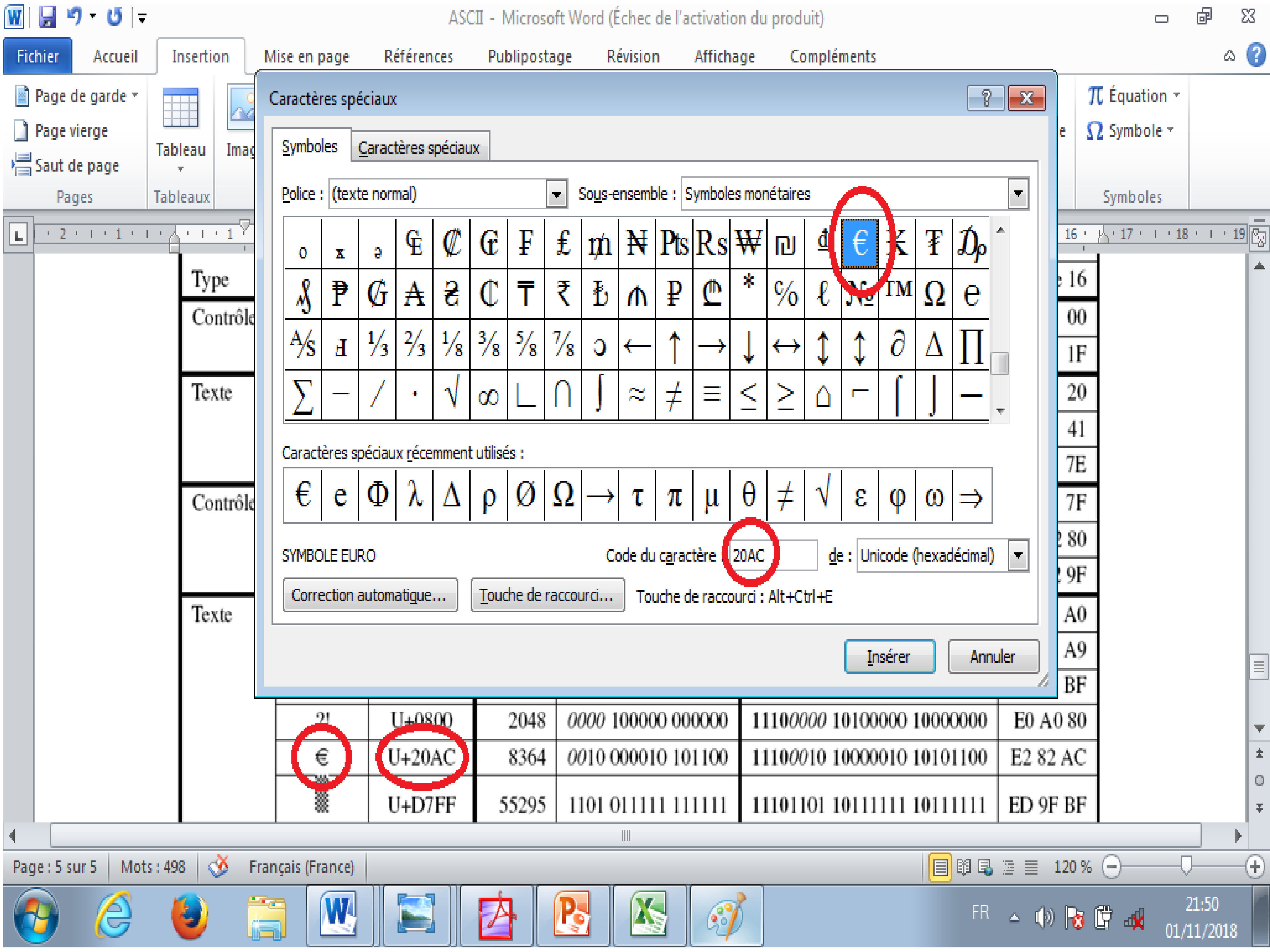
CHARACTER ENCODING

UNICODE

The main feature of **UTF-8 is that it is backward-compatible with the ASCII standard**, i.e. every ASCII character is encoded in UTF-8 as a single byte, **identical to the ASCII code**. For example, A (capital A) has ASCII code 65 and is encoded in UTF-8 by byte 65. Each character with a code point greater than 127 (non-ASCII character) is coded on 2 to 4 bytes. The **€ (euro)** character, for example, is encoded **on 3 bytes: 226, 130 and 172** (**111000101000001010101100**).

CHARACTER ENCODING

Type	Caractère	Point de code (Hex)	Valeur scalaire		Codage UTF-8	
			Base 10	binaire	binaire	Base 16
Contrôles	[NUL]	U+0000	0	0000000	00000000	00
	[US]	U+001F	0	0011111	00011111	1F
Texte	[SP]	U+0020	32	0100000	00100000	20
	A	U+0041	65	1000001	01000001	41
	~	U+007E	126	1111110	01111110	7E
Contrôles	[DEL]	U+007F	127	1111111	01111111	7F
	[PAD]	U+0080	128	00010 000000	11000010 10000000	C2 80
	[APC]	U+009F	159	00010 011111	11000010 10011111	C2 9F
Texte	[NBSP]	U+00A0	160	00010 100000	11000010 10100000	C2 A0
	é	U+00E9	233	00010 101001	11000010 10101001	C3 A9
	??	U+07FF	2047	11111 111111	11011111 10111111	DF BF
	?!	U+0800	2048	0000 100000 000000	11100000 10100000 10000000	E0 A0 80
	€	U+20AC	8364	0010 000010 101100	11100010 10000010 10101100	E2 82 AC
	▒	U+D7FF	55295	1101 011111 111111	11101101 10111111 10111111	ED 9F BF


ASCII - Microsoft Word (Échec de l'activation du produit)
Fichier
Accueil
Insertion
Mise en page
Références
Publipostage
Révision
Affichage
Compléments
Page de garde
Page vierge
Saut de page
Pages
Tableau
Tableaux
Équation
Symbole
Symboles
Caractères spéciaux
Symboles
Caractères spéciaux
Police : (texte normal)
Sous-ensemble : Symboles monétaires
Caractères spéciaux récemment utilisés :
SYMBOLE EURO
Code du caractère : 20AC
de : Unicode (hexadécimal)
Correction automatique...
Touche de raccourci...
Touche de raccourci : Alt+Ctrl+E
Insérer
Annuler
Type
Contrôle
Texte
Contrôle
Texte
U+0800
2048
0000 100000 000000
11100000 10100000 10000000
E0 A0 80
€
U+20AC
8364
0010 000010 101100
11100010 10000010 10101100
E2 82 AC
U+D7FF
55295
1101 011111 111111
11101101 10111111 10111111
ED 9F BF
Page : 5 sur 5
Mots : 498
Français (France)
120 %
FR
21:50
01/11/2018

In the end, it's not as complicated as all that... you just have to... !!! ???

End of chapter
Do you have any questions?