

# CHAPTER 3 :

## Information representation



### **3.1. Introduction**

Generally, all data in a computer is represented within computer components in binary form. It is a representation with two states symbolized by 1 (current) and 0 (no current). Therefore, coding is necessary to convert symbolic data from their external (usual) form to the binary form (0 and 1) usable by the computer.

### **3.2. Binary coding**

#### **a. Pure binary coding**

It is the binary code without coding, in other words which results directly from the general principle of numbering. It is the natural code used in digital systems (computer, etc.). The following table (Table 3.1) shows the pure binary code for an example of a 4-bit word ( $A_3 A_2 A_1 A_0$ ):

<b>Valeur décimal</b>	<b>A<sub>3</sub></b>	<b>A<sub>2</sub></b>	<b>A<sub>1</sub></b>	<b>A<sub>0</sub></b>
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>2</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>3</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>
<b>4</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>
<b>5</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>6</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>7</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>8</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>9</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>10</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>11</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>
<b>12</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>
<b>13</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>14</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>15</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>

**Table 3.1:** Pure binary coding.

#### **b. Reflected binary code (or GRAY code)**

Gray's code was established to avoid transition problems when changing from one word to the next word. In the sequential order of the code, there is only one bit that changes between two consecutive words which are, therefore, adjacent. Additionally, this

code is cyclical. The following table (Table 3.2) shows the 4-bit Gray code.

Decimal value	Gray code	Decimal value	Gray code
0	0000	8	1100
1	0001	9	1101
2	0011	10	1111
3	0010	11	1110
4	0110	12	1010
5	0111	13	1011
6	0101	14	1001
7	0100	15	1000

**Table 3.2:** GRAY code example.

**Example :** on 4 bits



To obtain the Gray code from the binary code we do the following operations:

- The first bit remains as it is.
- To obtain the other bits, for each bit we add the previous bit.
- If the result of the sum is '10', we keep the '0' and reject the '1'.

**c. The BCD code (Binary Coded Decimal)**

The BCD code was mainly used in the early days of calculating machines. It is sometimes still used in systems that display digital information to the user or for very simple systems that handle money. The BCD code encodes the number to be represented in a very direct way. Each digit of the number is encoded on 4 bits. Binary possibilities 10 to 15 are not used. The following table (Table 3.3) shows the DCB code.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

**Table 3.3 :** BCD code.

**Example :**

Represent  $(863)_{10}$  using a BCD code:

8	6	3
1000	0110	0011

$$(863)_{10} = (1000\ 0110\ 0011)_{\text{BCD}}$$

**d. Code exceeds by three**

- ❖ Each decimal digit is separately encoded into its binary equivalent plus three (3).

The following table (Table 3.4) shows the **Excess-3** code.

Decimal	BCD	Excess -3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Table 3.4 : Excess -3 code

**Exemple :**

$$\begin{aligned}
 (129)_{10} &= (0001\ 0010\ 1001)_{\text{BCD}} \\
 &\quad \downarrow \quad \downarrow \quad \downarrow \\
 &\quad (+11) \quad (+11) \quad (+11) \\
 &\quad \downarrow \quad \downarrow \quad \downarrow \\
 &= (0100\ 0101\ 1100)_{\text{Excess-3}}
 \end{aligned}$$

### 3.3. Character representation

For better exploitation by computers, characters must also be represented like numbers. Generally, only bits are used for this representation in the memory boxes. So, a numerical code is associated with each character. In the following, the most used character coding systems are shown.

#### a. EBCDIC code (Extended Binary Coded Decimal Internal Code)

EBCDIC stands for Extended Binary Coding Decimal Interchange Code. IBM invented this code to extend the binary coded decimal that existed at that time. All IBM computers and devices use this code. It is an 8-bit code and therefore can accommodate 256 characters. Below, some characters coded with the EBCDIC Code.

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX		PT			GE				FF	CR		
1	DLE	SBA	EUA	1C		NL				EM			DUP	SF	FM	ITB
2							ETB	ESC						ENQ		
3			SYN					EOT					RA	NAK		
4	SP										€	.	<	(	+	
5	&										!	\$	*	)	:	~
6	-	/										.	%	_	>	?
7											:	#	@	'	=	*
8		a	b	c	d	e	f	g	h	i						
9		j	k	l	m	n	o	p	q	r						
A		-	s	t	u	v	w	x	y	z						
B																
C	{	A	B	C	D	E	F	G	H	I						
D	}	J	K	L	M	N	O	P	Q	R						
E	\		S	T	U	V	W	X	Y	Z						
F	0	1	2	3	4	5	6	7	8	9						
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

**Table 3.5 : EBCDIC character table**

**b. ASCII code (American Standard Code for Information Interchange)**

The following table (Table 3.6) of the ASCII code shows the numerical equivalents of upper and lower case characters of the Latin alphabet, numbers and some punctuation marks. Used at the time (1960s) in the exchange of data between terminals and computers. A 7-bit code is required to represent one hundred and twenty-eight different combinations. Each character is encoded on 1 byte by two hexadecimal symbols. The column number gives the most significant hexadecimal symbol and the row number the least significant symbol.

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

**Table 3.6:** ASCII character table

### c. UTF : variable length coding

In UTF, each character is encoded on one, two or three bytes. All characters in the ASCII code are encoded in single-byte UTF with the same value as in ASCII. In contrast, a byte that contains a value greater than 127 announces one character over two or three bytes and UTF is constructed in such a way that all of these bytes contain a code greater than 127. This mechanism ensures backward and forward compatibility with the ASCII.

## 3.4. Representation of numbers

### 3.4.1. Integer numbers

There are two types of integers:

- ❖ Unsigned integers (positive) ;
- ❖ Signed integers (positive or negative).

Generally, there are three methods for representing negative numbers:

- Sign/Absolute value;
- 1's complement (restricted complement);

- 2's complement (true complement).

### a. Unsigned representation

A natural number is a positive integer or zero. The choice to make (i.e. the number of bits to use) depends on the range of numbers you wish to use. To encode natural integers between 0 and 255, we will only need 8 bits (one byte) because  $2^8=256$ . Generally speaking, n-bit coding can be used to represent natural integers between 0 and  $(2^n - 1)$ .

To represent a natural integer after having defined the number of bits on which it is coded, simply place each bit in the binary cell corresponding to its binary weight from right to left, then “fill” the unused bits by zeros.

### b. Representation with sign and absolute value

Dedicate one bit, the most significant, to represent the sign. The range of possible values to represent, for an n-bit word, goes from  $-(2^{n-1}-1)$  to  $+(2^{n-1}-1)$  and there are two possible representations for the value 0.


#### Example :

+26 =	<b>0</b>	0011010
	Sign bit (+)	Representation of 26
-26 =	<b>1</b>	0011010
	Sign bit (-)	Representation of 26

### c. 1's complement (or restricted complement)

In a 1's complement code, we obtain the negative code of a number by complementing the code word of this number, that is to say by inverting all the bits. With an n-bit word, it is possible to represent the values  $-(2^{n-1}-1)$  to  $+(2^{n-1}-1)$  and there are two possible representations for the value 0.

**Example:**  $(-26)_{10} = (?)_{1's\text{complement}}$  sur 8 bits.

<b>Positive number :</b>	$(+26)_{10} =$	$(00011010)_{1's\text{ complement}}$ 
<b>1's complement :</b>	$(-26)_{10} =$	$(11100101)_{1's\text{ complement}}$



**d. 2's Complement (or True Complement)**

In a 2's complement code, we obtain the negative code of a number by completing the complement of the code word of this number and adding 1 to it without retaining. With an n-bit word, it is possible to represent the values  $-(2^{n-1})$  to  $+(2^{n-1}-1)$  and there is only one possible representation for the value 0.

**Example :**

Positive number:	+26 =	00011010 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
Inversion		11100101
Adding 1:		+        1
Negative number :	-26 =	(11100110) <sub>2's complement</sub>

The 2's complement code is very important since it allows you to perform arithmetic operations with signs without having to worry about it.

**Example :**

$$7-4 = 7 + TC(4) = 0111 + TC(0100) = 0111 + (1011+1) = \mathbf{1}0011 = 0011 = (+3)_{10}$$

↙  
The withholding must be rejected

Decimal	Binary	2's complement
127	01111111	01111111
- 26	- 00011010	+(-00011010) ↓ ↑
= 101	= 01100101	01111111 + 11100101
		= $\mathbf{1}01100101_{c\grave{a}-2} = (101)_{10}$  The hold is rejected

Decimal	Binary	2's complément
26	00011010	00011010
-127	-01111111	+(-01111111) ↕
= -101	We can't.	00011010 + 10000001
		= 10011011 <sub>1'sC</sub> The result is negative since the most significant bit is "1". We must find the positive value.
Subtract the value 1		10011011 -        1
We do the reverse.		= 10011010 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
We convert to decimal		01100101 = (+101) <sub>10</sub>
Then the decimal result is		(10011011) <sub>2'sC</sub> = (-101) <sub>10</sub>

2's complement coding allows the representation of negative numbers used in calculators. The sign is the most significant bit: **0** for the + sign (compatible with unsigned coding) and **1** for the negative sign (-).

Decimal	2's Complement	Decimal	2's Complement
+7	0111	-7	1001
+6	0110	-6	1010
+5	0101	-5	1011
+4	0100	-4	1100

+3	0011	-3	1101
+2	0010	-2	1110
+1	0001	-1	1111
+0	0000	-0	0000

#### 4.1. **Fractional numbers:**

There are two representations used to represent fractional numbers in calculators:

- ✓ fixed point coding;
- ✓ floating point coding.

Generally, a fractional number is a number that has two parts::

- An integer value;
- A fractional value.

The two parts are separated by a comma which is placed to the right of the least significant digit of the integer part (unit weight digit).

##### **a. Fixed point**

At the machine level, computers do not have commas. The numbers are considered as integers; it is left to the programmer to move the decimal point (virtual comma). Depending on the number of bits, we cannot represent the largest ones. The Binary number system is a weighted system; each bit of a binary number has a weight depending on its rank (Numbering and representation of numbers).

##### **Example :**

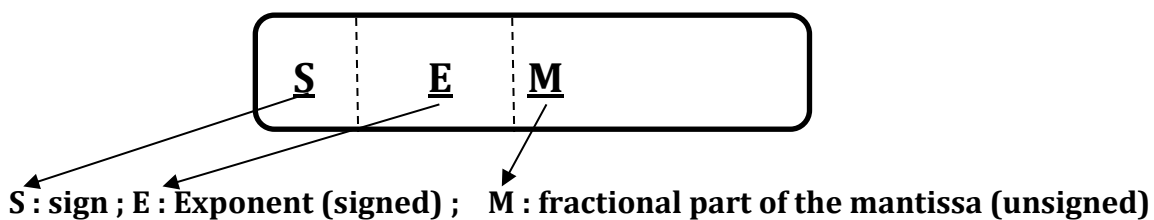
Let the binary number  $N = (1\ 0\ 0\ 1.0\ 1\ 0\ 0\ 1)_2$

Fixed-point coding	
Integer part:	Fractional part:
$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$	$0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}$
8+1	0,25+0,125
9	0,375
$(1001.01001)_2 = (9,375)_{10}$	

### b. Floating point (IEEE 754 standard)

Floating numbers make it possible to represent, in an approximate manner, a part of the real numbers. The value of a real cannot be too large or too precise. It depends on the number of bits used (usually 32 or 64 bits).

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). The binary encoding is of the following form:



☞ We call scientific notation each number written in the form  $1, M \cdot 2^E$ , where:

- M is called the mantissa of the number ;
- E is the exponent.

☞ As the mantissa always begins with an integer part equal to 1, we do not write it and we only express the fractional part, M, called “pseudo-mantissa”.

The IEEE 754 standard representation defines two main types of floating point numbers, 32-bit single precision and 64-bit double precision.

- ❖ Single precision numbers have a 23-bit pseudo-mantissa (corresponding to negative powers of 2), an 8-bit exponent and a sign bit (Table.3.7) :

<b>1 bit</b>	<b>8 bits</b>	<b>23 bits</b>	<b>-126</b>	<b>+127</b>
<b>Sign</b>	<b>Exponent</b>	<b>Pseudo Machine</b>	<b>E min</b>	<b>E max</b>

**Table 3.7:** The simple precision

**Example 1:**

We want to represent 278 in simple precision IEEE 754 format.

We start by writing 278 in base 2, and then we put it in scientific form.

$$(278)_{10} = (100010110)_2 = 1,0001011 2^8.$$

So we have:

☞ **S** = 0 because 278 is positive.

☞ **E<sub>b</sub>** = E + 127 = 8 + 127 = (135)<sub>10</sub> = (10000111)<sub>2</sub> in base 2 on 8 bits.

☞ **M** = 00010110000000000000000.

We obtain the following representation:

<b>1 bit</b>	<b>8 bits</b>	<b>23 bits</b>
<b>Sign</b>	<b>Exponent</b>	<b>Pseudo Machine</b>
<b>0</b>	<b>1000111</b>	<b>00010110000000000000000</b>

**Example 2:**

We want to represent -6.53125 in simple-precision IEEE 754 format.

We start by writing 6.53125 in base 2, and then we put it in scientific form.

$$(6,53125)_{10} = (110,10001)_2 = 1,1010001 2^2.$$

So we have:

☞ **S** = 1 because -6,53125 is negative.

☞ **E<sub>b</sub>** = E + 127 = 2 + 127 = (129)<sub>10</sub> = (10000001)<sub>2</sub> in base 2 on 8 bits.

☞ **M** = 10100010000000000000000.

We obtain the following representation:

1 bit	8 bits	23 bits
<b>Sign</b>	<b>Exponent</b>	<b>Pseudo Machine</b>
1	10000001	10100010000000000000000

- ❖ Double-precision numbers have a 52-bit pseudo-mantissa, an 11-bit exponent, and a sign bit (Table 3.8).

1 bit	11 bits	52 bits	-1022	+1023
<b>Sign</b>	<b>Exponent</b>	<b>Pseudo Machine</b>	$E_{\min}$	$E_{\max}$

**Table 3.8:** The double precision