I - Information coding

I .1– The Gray Code:

1- Definition of the Gray Code:

Gray code, also known as **Gray code** or **reflected binary code**, is a type of binary coding that allows you to change only one bit at a time when a number is increased by one unit. This property is important for several applications.

The name of the code comes from the American engineer **Frank Gray**who published a patent on this code in 1953, but the code itself is older.

2- Gray Code Principle:

The Gray code is a binary coding, that is to say a function that associates with each number a binary representation. This method is different from natural binary coding. The following table partially shows the 4-bit coding (only the first 8 values are presented, the next eight with the first bit at 1 are not).

Decimal Coding	Natural Binary Coding	Gray Coding or Reflected Binary
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

The main difference between the two is that the Gray coding of two consecutive numbers differs by only one position. For example 5 is coded by 0111, and 6 is coded by 0101: here only the second bit changes.

3- Binary code to Gray code conversion

Let B be a number written in pure natural binary on m bits

$B_{(2)}$= Bm ……$B_4B_3B_2B_1$; Bm is the most significant bit

G is the Gray code equivalent of the number B also written on m bits

$G_{(Gray)}$= Gm ………$G_4G_3G_2G_1$

The transition from pure binary to Gray code is done according to the following two steps:

1- Gm = Bm.
2- Now we compare the pairs:
   a. If $B_n = B_{n-1}$ then $G_{n-1} = 0$:
   b. If $B_n \neq B_{n-1}$ then $G_{n-1} = 1$:

Examples:

Let the binary number = 0101 (5 in decimal, we work or code on 4 bits)

What is its equivalent in Gray code?

B = 0101 = $B_3\ B_2\ B_1\ B_0$

- $\mathbf{G_3 = B_3 = 0}$; (This is the first step).
- The second step is the comparison of the bits of the number B.
  - $B_3 \neq B_2$ then $\mathbf{G_2 = 1}$.
  - $B_2 \neq B_1$ then $\mathbf{G_1 = 1}$.
  - $B_1 \neq B_0$ then $\mathbf{G_0 = 1}$.

We conclude that G = 0111.

Let's take another example:

Let the binary number = 1101101 (we work or code on 7 bits)

What is its equivalent in Gray code?

B = 1101101 = $B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$

- $\mathbf{G_6 = B_6 = 1}$; (This is the first step).
- The second step is the comparison of the bits of the number B.
  - $B_6 = B_5$ then $\mathbf{G_5 = 0}$.
  - $B_5 \neq B_4$ then $\mathbf{G_4 = 1}$.
  - $B_4 \neq B_3$ then $\mathbf{G_3 = 1}$.
  - $B_3 = B_2$ then $\mathbf{G_2 = 0}$.
  - $B_2 \neq B_1$ then $\mathbf{G_1 = 1}$.
  - $B_1 \neq B_0$ then $\mathbf{G_0 = 1}$.

It is concluded that G = 1011011

4- <u>Converting Gary code to pure Binary code:</u>

Let G be a number written in Gray code on m bits

$G_{(Gray)}$= Gm ………$G_4G_3G_2G_1$ Gm is the most significant bit

$B_{(2)}$ is the equivalent in pure or natural binary code of the number G also written on m bits

$B_{(2)}$= Bm ………$B_4B_3B_2B_1$

The transition from the Gray Code to the pure binary code is done according to the following two steps:

3- Bm = Gm.
4- Now we compare the pairs:
   a. If $G_{n-1} = B_n$ then $B_{n-1} = 0$:
   b. If $G_{n-1} \neq B_n$ then $B_{n-1} = 1$:

Examples:

Let the number G written in Gray code = 110010, we want to find its equivalent in pure binary.

- We apply the rule (we work on 6 bits, because G is coded on 6 bits)
    - G = 110010 = $G_5G_4G_3G_2G_1G_0$
    - 1- $B_5 = G_5 = 1$
    - 2-
        - $G_4 = B_5$ then $B_4 = 0$
        - $G_3 = B_4$ then $B_3 = 0$
        - $G_2 = B_3$ then $B_2 = 0$
        - $G_1 \neq B_2$ then $B_1 = 1$.
        - $G_0 \neq B_1$ then $B_0 = 1$.

We can then deduce that B = 100011.

Let's take another example:

Let G be a number encoded in Gray code which is equal to 10111011, find its value in pure binary code.

We apply the rule, but we will schematize it for clarity.

	G_7	G_6	G_5	G_4	G_3	G_2	G_1	G_0
Gray	1	0	1	1	1	0	1	1
	$\downarrow$	$\neq \swarrow$	$= \swarrow$	$\neq \swarrow$	$= \swarrow$	$= \swarrow$	$\neq \swarrow$	$= \swarrow$
Pure binary	1	1	0	1	0	0	1	0
	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

I.2 – The DCB Code: Decimal Binary Coded:

1- DCB Code Definition:

The DCB or BCD code, is the acronym for Binary Coded Decimal in English is a numbering system used in digital electronics and computer science to encode numbers by approximating the usual human representation, in base 10. In this format, numbers are represented by one or more digits between **0 and 9**, and each of these digits is encoded on four bits.

2- DCB Code Principle:

To encode a decimal number in BCD, we will separately encode each digit of the base ten number in Binary according to the table below, remember that we work on **4 bits (4 positions),** so the numbers from 10 to 15 (1010 to 1111) are not supported by 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

Coding of Decimal Digits in DCB.

Example :
We will encode the number $(785)_{10}$ in DCB, for this, each digit of the number will be encoded separately in binary on 4 bits.

Decimal	7	8	5
BCD	0111	1000	0101

So $(785)_{10} = (0111\ \ 1000\ \ 0101)_{DCB}$

3- Addition in DCB code:

As already mentioned, the DCB code allows the coding of only 10 numbers, namely, the digits from 0 to 9 , therefore, the numbers from 10 to 15, are not allowed, that is, the DCB code does not know them.

When we add 2 DCB-coded numbers, we add each 4-bit block of the first number, with its equivalent of the second number.

Now, if the result of one of the blocks exceeds 9, that is to say its value is between 10 and 15 (1010 to 1111), the number 6 (0110) is added to this block.

Examples:

- Either to add $(352)_{10} + (34)_{10}$ in DCB

	Decimal		BCD		
	352		0011	0101	0010
+		+			
	34			0011	0100
=		=			
	386		0011	1000	0110

There, the result is equivalent, and correct.

- Either to add $(153)_{10} + (151)_{10}$ in DCB

	Decimal		BCD		
	153		0001	0101	0011
+		+			
	151		0001	0101	0001
=		=			
	304		0010	1010	0010

There, the result obtained in DCB is not equivalent to that of the Decimal, moreover, we obtained the value 1010 (10 in Decimal which is >9) which is not recognized by the DCB code, there, we must add to this block which exceeds 9, the value 6 (0110).

Let's see, what happens when we add 6 (0110) to this block.

	Decimal		BCD		
	153		0001	0101	0011
+		+			
	151		0001	0101	0001
=		=			
	304		0010	1010	0100
		+	0110		
		=	0011	0000	1000

There, the result is correct!.

One question remains, why add 6 (0110) specifically?.

Well, because as in DCB, we code on 4 bits, so 2^4=16 numbers are possible in binary coderen, but only 10 numbers are allowed (from 0 to 9), the remaining 6 numbers (from 10 to 15) are prohibited, when the sum exceeds 9, we add 6 to make a loop and return to the value 0.

Notes:

- The number encoded in BCD does not correspond to the decimal number converted to natural binary.
- Combinations greater than 9 (from 10 to 15) are prohibited. For example, the combination 1010 does not belong to the BCD code.
- BCD decimal coding is simple, but it is not possible to do mathematical operations directly on it.
- This code is mostly used for the display of decimal data. (In calculators for example)

I.3 – The DCB Code plus three: DCB +3:

1- DCB+3 Code Definition:

The **DCB code plus 3** or **excess 3 code** also called **Stibiz code** of the name of its inventor, is an unweighted code from the **DCB** code to which **3** is systematically added to each digit. This code is often used on arithmetic units that calculate in a decimal number system rather than a binary system.

The code plus 3 makes it possible to perform the arithmetic operations of addition and subtraction with a minimum of logical functions.

2- DCB+3 Code Principle:

To code a decimal number in BCD+3, before coding it in DCB, we add 3 to it, then we code it in DCB, we remember that we work on **4 bits (4 positions),** so we have 16 numbers (from 0 to 15) that we can code in DCB +3, but combinations of 13 to 15 (1101 to 1111) are prohibited, we can just code from 0 to 12.

The table below illustrates the coding of decimal numbers in DCB+3

Decimal	Decimal +3	DCB+3
0	3	0011
1	4	0100
2	5	0101
3	6	0100
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

Coding of Decimal Numbers by DCB+3

Example :

We will encode the number $(785)_{10}$ in DCB+3, for this, we add to each digit 3, then it will be encoded separately in binary on 4 bits, then we

Decimal	7	8	5
Decimal +3	10	11	8
DCB+3	1010	1011	1000

So $(785)_{10}$ = (1010 1011 1000) $_{DCB+3}$.

II- R REPRESENTATION OF INTEGERS

II.1 Representation of a natural number

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

Examples: 9 = $(00000101)_2$, 128 = $(10000000)_2$

II.2 Representation of a relative integer

A relative integer is an integer that can be negative. It is therefore necessary to code the number in such a way that we can know whether it is a positive number or a negative number.

- **Problem:** How to tell the machine that a number is negative or positive????

There are 3 methods to represent negative numbers:

II.2.1 Sign and Absolute Value (S/VA)

If we work on n bits , then the most significant bit is used to indicate the sign (1: negative sign, 0: positive sign) and the other bits (n -1) designate the absolute value of the number.

- **Example**: If we work on 4 bits (-

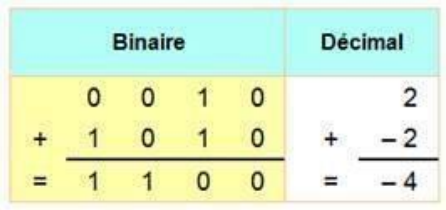
  $5)_{10}$= $(1\ 101)_{2;}$ $(+5)_{10}$ = $(\ 0101)_2$

Sign	VA	Values
0	**.00**	**+ 0**
0	01	1
0	10	2
0	11	3
1	**.00**	**0**
1	01	-1
1	10	-2
1	11	-3

On n bits, the range of values that can be represented in S/VA:

$$-(2^{(n-1)} - 1) \leq N \leq + (2^{(n-1)}-1)$$

Let's try to calculate 2 + (-2) = 0



Binaire					Décimal	
	0	0	1	0		2
+	1	0	1	0	+	− 2
=	1	1	0	0	=	− 4

That's not what we wanted:-4 instead of 0!! **Pros:** It's a fairly simple representation. **Disadvantage:** the zero has two representations +0 and -0

II.2.2 <u>The addition to a</u>

We call complement to one (i.e.1) of a number N another number N 'such that: N+N' $=2^n-1$

n: is the number of bits of the representation of the number N .

Example:

Let N=1010 over 4 bits so its complement to one of N: N'= $(2^4 - 1)$-N

N'=(16-1)-$(1010)_2$= $(15)_{10}$ - $(1010)_2$ = $(1111)_2$ – $(1010)_2$ = 0101

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

To find the one's complement of a number, simply invert all the bits of this number: If the bit is a 0 put in its place a 1 and if it is a 1 put in its place a 0 .

Examples

(10010011) = $(01101100)_{\text{i.e.1}}$

(11001100) = $(00110011)_{\text{i.e.1}}$

In addition to one, the most significant bit indicates the sign (0: positive , 1: negative).

The one-to-one complement of the one-to-one complement of a number is equal to the number itself.

CA1(CA1(N))= N

Example:

What is the decimal value represented by the value 101010 in 1 on 6-bit complement?

- The most significant bit indicates that it is a negative number.
- Value = - CA1(101010)

= - $(010101)_2$= - $(21)_{10}$

If we work on 3 bits

A1:	Binary	Decimal
000	000	**+ 0**
001	001	+ 1
2010	2010	+ 2
011	011	+ 3
100	011-	- 3
101	010	- 2
110	-001	- 1
111	000-	**- 0**

Disadvantage: In this representation the zero has a double representation. If we work on n bits, the range of values that we can represent in CA1:

$$-(2^{(n-1)} -1) \leq N \leq +(2^{(n-1)} -1)$$

II.2.3 <u>The two-way complement</u>

Positive numbers are encoded in the same way as in pure binary whereas a negative number is encoded by adding the value 1 to its complement at 1. The most significant bit is used to represent the sign of the number.

The two's complement representation is the most used representation for the representation of negative numbers in the machine.

The two's complement of the two's complement of a number is equal to the number itself.

CA2(CA2(N))= N

Hippocampus	**binary**	**value**
000	000	**+ 0**
001	001	+ 1
2010	2010	+ 2
011	011	+ 3
100	- 100	- 4
101	011-	- 3
110	010	- 2
111	-001	- 1

Advantage: We note that the zero does not have a double representation

If we work on n bits, the range of values that we can represent in CA2:

$$-(2^{(n-1)}) \leq N \leq +(2^{(n-1)} -1)$$

Example

It is desired to encode the value −19 on 8 bits. Il suffit:

1. to write 19 in binary: 00010011

2. to write its complement at 1 : 11101100

3. and add 1 : 11101101

The binary representation of −19 over 8 bits is therefore 11101101.

- **Two's complement arithmetic operations**

$$\begin{array}{rl} 9 & 01001 \\ \underline{4} & \underline{00100} \\ 13 & -1101 \end{array}$$

(the sign bit = 0 the number is positive)

$(01101)_2 = (13)_{10}$

$$\begin{array}{rl} +9 & 01001 \\ \underline{-} & \underline{4,11100} \\ +5 & 100101 \end{array}$$

We notice that the result is on 6 while we work on 5 bits in this case we ignore the sixth bit (called carry) so the result of the addition will be 00101(Positive result)

$(00101)_2 = (5)_{10}$

$$\begin{array}{rl} 9 & 10111 \\ \underline{4} & \underline{11100} \\ 13 & 110011 \end{array}$$

We ignore the carry forward the result is 110011

(negative number) Result = - CA2 (10011)= -(01101)

-13

$$\begin{array}{rr} 9 & 10111 \\ \underline{9} & \underline{01001} \\ 0 & 100,000 \end{array}$$

The result is positive

$(00000)_2 = (\ 0)_{10}$

➢ **Retention and overflow**

- ✓ It is said that there is a holdback if an arithmetic operation generates a carryover.
- ✓ We say that there is an overflow if the result of the operation on n bits **and** False.
- ✓ The number of bits used is insufficient to contain the result, i.e. the result exceeds the range of values on the n bits used.

➢ **Overflow case**

$$\begin{array}{rr} 9 & 01001 \\ \underline{8} & \underline{01000} \\ 17 & 10001 \end{array}$$

The result is negative whereas it must be positive Overflow!!!

$$\begin{array}{rr} 9 & 10111 \\ \underline{8} & \underline{11000} \\ 17 & 01011 \end{array}$$

The result is positive whereas it must be negative Overflow!!!

We have an overflow if the sum of two positive numbers gives a negative number, or the sum of two negative numbers gives a positive number

There is never an overflow if the two numbers are of different signs.

III. THE REPRESENTATION OF REAL NUMBERS

A real number consists of two parts: the integer part and the fractional part (the two parts are separated by a comma)

Problem: how to tell the machine the position of the comma? There

are two methods for representing real numbers:

III.1 <u>Fixed point numbers</u>

Has an ‘integer‘ part and a ‘decimal‘ part separated by a comma, used by the first machines. The position of the comma is fixed hence the name.

Example: $(11.01)_2$, $(75.23)_8$, $(E7,A4)_{16}$

III.2 Floating point numbers

Floating-point numbers are generally non-integer numbers whose numbers are written only after the decimal point and to which we
adds an exponent to specify how many positions the comma should be moved. For

example, in decimal notation, we will write:

the number 31, 41592 in the form: $0.3141592 * 10^2$

the number -0.01732 in the form: $0.1732 *10^{-1}$

In computer science, a standard has become necessary for the representation of floating points. This is the IEEE 754 standard.

➢ The IEEE 754 standard

IEEE 754 is a standard for the representation of floating-point numbers in binary. It is currently the most used for the calculation of floating-point numbers in the computer field,

In the IEEE 754 standard, a floating point number is always represented by a triplet (S,E,M)

1. The first component S determines the sign of the number represented, this sign being 0 for a positive number, and 1 for a negative number, the sign is represented by a single bit, the most significant bit (the leftmost bit)
2. the second E designates the exponent is coded on 8 bits consecutive to the sign
3. the third M designates the mantissa (the bits located after the comma) on the remaining

23 bits Thus the coding is done in the following form:

S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

IEEE 754 Single Accuracy:

Sign (1 bit)	**Exponent (8 bit)**	**Mantissa (23 bit)**

IEEE 754 Double Precision:

Sign (1 bit)	**Exhibitor (11 bit)**	**Mantissa (52 bit)**

However, some conditions must be met for exhibitors:

- ✓ Exhibitor 00000000 is prohibited
- ✓ Exhibitor 11111111 is prohibited. However, it is used to report errors, this configuration is called NaN (Not a Number)
- ✓ 127 must be added to the exponent (case of simple precision) for a decimal to a binary real number conversion. Exhibitors can thus range from -256 to 255.

The formula for expressing real numbers is as follows:

$$(-1)^{s}.2^{(E-127)}.1,M$$

Note The 127 of the (E-127) comes from 2 $^{\text{nbitsofthe exponent-1}}$ - 1

Example 1: Find the IEEE 754 single-precision representation of the number

$(35.5)_{10}$ The number is positive S=0

$(35.5)_{10}$ = ($100011.1)_{2}$……. Fixed-point

= 1,000111 * 2 ^{5}Floating point (M= 000111)

Exhibitor: E-127 = 5 E= 132 = ($10000100)_{2}$

So

0	10000100	00011100000000000000000
S	E	M

Example 2: Find the single-precision IEEE 754 representation of the number

(- $525.5)_{10}$ The number is negative S=1

$(525.5)_{10}$ = $(1000001101.1)_{2}$.................... Fixed point

= 1.0000011011* 2 ^{9}Floating point (M= 0000011011)

Exhibitor: E-127 = 9 E= 136 = $(10001000)_{2}$

So

1	10001000	00000110110000000000000
S	E	M

Example 3: Find the single-precision IEEE 754 representation of the

number $(-0.625)_{10}$ The number is negative S=1

$(0.625)_{10} = (0.101)_2$……. Fixed-point

$= 1.01 * 2^{-1}$Floating point (M= 01)

Exhibitor: E-127 = -1 E= 126 = $(1111110)_2$

So

1	01111110	01000000000000000000000
S	E	M

Example 4: Find the floating number with the following IEEE754 representation:

0	10000001	11100000000000000000000

S =0 Number is positive

E = $(10000001)_2 = 129$ E-127 = 129 -127 =2

1.M = 1.111

$1.111 * 2^2 = (111.1)_2 = (7.5)_{10}$