TD 1 – Binary coding

Exercise 1: Conversion from base 10 to bases 2 and 16

A number in base 10 can be written, for example, as 345.112. We distinguish in base 10 the integer part, (which corresponds to powers greater than zero of 10) and the fractional part (which corresponds to negative powers of 10).
Thus, 345,112 = $3 * 10^2 + 4 * 10^1 + 5 * 10^0 + 1 * 10^{-1} + 1 * 10^{-2} + 2 * 10^{-3}$.
Note that the "digits" correspond to the weights of each power of 10.
To transform a number from base 10 to base 2, we start by coding the integer part. To do this, we use the principle of integer division. Thus, if A and B are two integers, then A/B gives an integer Q with an integer remainder R. We successively divide the number by 2 until there is only 0 in the quotient Q. The remainders are the required bits. The first remainder is the least significant bit (the most right) and the last remainder is the most significant bit (the most left). To encode the fractional part in base 2, we multiply by 2. The number (0 or 1) which passes in integer part is the most significant bit. We start again with what remains.
For example, code 0.625 in base 2:
0.625 x 2 = 1.250 1 is the highest weight (leftmost)
0.250 x 2 = 0.500 0 is the next weight
0.500 x 2 = 1.000 1 is the weakest weight (the most right)
So 0.625 is coded 0.101 in base 2

a) Code 327 and 15.1 in base 2.
b) Code 45 from base 10 to base 4

Two bases play, in addition to binary, important roles in computer science: base 8 (octal) where the possible digits are 0, 1, 2, 3, 4, 5, 6, 7 and the hexadecimal (base 16) where the symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
It's important to note that all integer values are expressed in computer memory by setting the values of binary digits. However, long binary digit sequences are difficult for us to deal with, that's why we use octal and hexadecimal.

c) What does the number 3A,7B represent in base 10?

We can see that $2^3 = 8$ and $2^4=16$. To quickly convert a binary to base 8, we make packets of 3 starting from the left of the comma for the integer part and to the right of the comma for the fractional part. We proceed in the same way for a base 16 with packets of 4.
To finish a packet, we complete with 0's. A packet of three bits corresponds to a number between 0 and 7 and a packet of 4 bits corresponds to a number between 0 and 15

d) Transcribe 1001011,1101 and 111001011,10011 in base 8 then in base 16

Exercise 2: Addition and multiplication in binary

The mechanism of addition in base 2 is absolutely identical to addition in base 10. Each identical power of 2 is added in what is called a stage. It is possible that there is a carry (which comes from the previous stage, i.e. the stage on the right). For a stage, the total varies between 00 (zero) and 11 (three). The least significant bit is set and the most significant bit is retained for the next stage

a) Add 1001101 with 11000100
b) Add in one operation 11111 + 110011 + 100011 then 11111 + 100 + 1111

The multiplication in binary is very simple because the multiplication table is very short! The mechanism is the same as in decimal. A multiplication is a sequence of additions.

c) Write the multiplication table
d) Multiply 1100 by 101

Exercise 3: Integer representation

In all computers, all programming languages, integer data are coded in the same way. For example, with 8 bits, we can represent 2_ different values, or 256 integers.

a) If we code an integer on 2 bytes (16 bits), what is the number of possibilities?

Classical coding is done on 2, 4, 8 bytes. Contrary to a program like Mathematica or Maple, there is no dynamic adaptation of the size to the value in the classical computer languages (C, Visual Basic, java...).
The sign (negative or positive) is carried by the most significant bit (**sign bit**). A positive number has a sign bit of 0 and a negative number has a sign bit of 1. The value 0 is coded with one as sign bit.
To encode a positive number, we make a conversion from base 10 to base 2 and we recopy simply on 8, or 16, or 32 bits by completing by zeros the strong weights not used.

b) Encode 312 on 16 bits

To code a negative number, we start by coding its absolute value (positive!) on N bits.
Then, we determine its **complement to 1** by replacing the 1's by 0's and 0's by 1's. Finally, we add arithmetically to the result the value 1: it is the **complement to 2**. We obtain the coding of the negative number.

Example: to code -28 on 8 bits:
+28= 00011100
Complement to 1: 11100011
Addition: 11100011+00000001 = 11100100 it is the complement to 2 and the final result.

c) Encode -43 on 16 bits

Exercise 4: Representation of the floats (IEEE754.2 standard)

Real numbers (real, float) have had in the history of computers several different representations, even from one brand to another (IBM, BULL,...). At the beginning of the 80's, a standard was fixed.

A real number is, in base 10, defined by three elements: the sign, the mantissa and the exponent. In our usual use, we can write for example 245.5 or $2.455*10^{2}$ or even $2455*10^{-1}$.

There are therefore **several possible external representations. The internal representation is unique.**

We take as an example the simplest representation on 32 bits.
Let's encode the decimal number -118.625 using the IEEE 754 mechanism.

1. First, we need the **sign**, the **exponent,** and the **fractional part**.
   This is a negative number, so the sign is "1".

2. Then we write the number (without the sign) in binary. We get 1110110,101.

3. Then we shift the decimal point to the left, so that only a 1 is left:
   1110110,101 = 1,110110101 × 2^{6}. It is a normalized floating number: the **mantissa** is the part on the right of the comma, filled with 0 towards the right to obtain 23 bits. This gives:
   1101101 0100 0000 0000 0000 (we omit the 1 before the decimal point, which is implicit).

4. The exponent is equal to 6, and we need to convert it to binary and shift it. For the 32-bit IEEE 754 format, the shift is 127. So, 6 + 127 = 133 (decimal) = 1000 0101 (binary).
   The shift is always 127 on 32 bits.

We have therefore -118,625 (decimal) = 1100 0010 1110 1101 0100 0000 0000 0000 (binary).

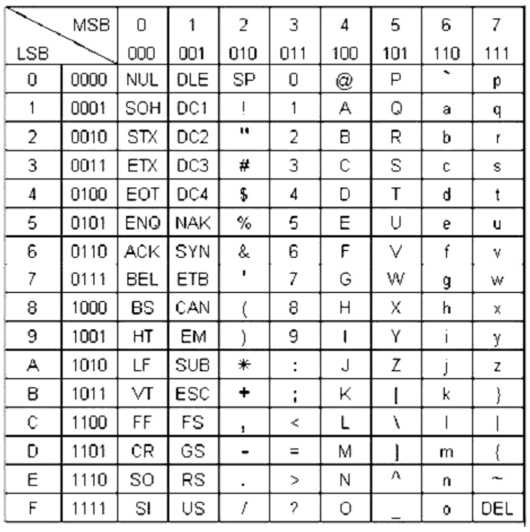
a) Represent the float number 32,1 in IEEE754.2 format
b) How to represent the float number 0?

Exercise 5: Other data representations

The characters were represented in a very simple way by using the ASCII code on 7 then on 8 bits, which made it possible to have 128 different characters then 255 by adding the 8th bit.

The following table represents the table of codes on 7 bits. This table presents the LSB (Least significant bit) and MSB (Most significant bit).

TD 1 – Binary coding



LSB \ MSB		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	}
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	{
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

a) Encode "TrUc" in 7 bit ASCII code

There is now the UNICODE which allows to code any writing system (Chinese, Arabic, Greek, Indian,...) and any symbols (mathematics, music...). This allows you to send in Chinese and receive mail in Chinese. This was not the case in the 90s!

b) If you have the courage, take a look at Wikipedia: it's not easy at all!

Images are arrays of colored dots called "Pixels". Very classically, a pixel is represented on 24 bits: 8 bits for red, 8 bits for green and 8 bits for blue.

c) Give the number of bytes of an image of 1024 x1024 pixels
d) Transform this value in KO (1024 bytes), in MO (1024 KO)
e) How to code the white and the black?