Représentation des données dans un ordinateur :

✓Les ordinateurs traitent des signaux qui sont **binaires** , c'est-à-dire des signaux qui ont deux valeurs possibles **(bit).**

✓Ils ne manipulent et ne mémorisent donc que des nombres binaires : 0 ou 1.

✓Pour traiter des informations, il faut établir une correspondance entre la représentation habituelle de l'information (caractère, nombre, image , etc.) et sa représentation dans la machine.

Il faut donc effectuer un codage.

Quantité d'information a coder:

Dépend du nombre de bits:

✓Avec 1 bit : $2^1 = 2$ états différents: 0 1

✓Avec 2 bits : $2^2 = 4$ états différents : 00 01 10 11

✓Avec 3 bits : $2^3 = 8$ états différents : 000 001 010 011 100 101 110 111

✓Avec 8 bits (1 octet) on peut représenter $2^8 = 256$ états différents

✓Avec **n bits** on peut représenter: $\mathbf{2^n}$ valeurs possibles (**0 à $\mathbf{2^n - 1}$**)

Les multiples

1 kilo-octet (Ko) vaut 2^{10} octets = 1024 octets

1 méga-octet (Mo) vaut 2^{10} Ko = 1024 Ko= 2^{20} octets =1048576 octets

Systèmes de numération:

Il existe plusieurs systèmes de numération possibles :

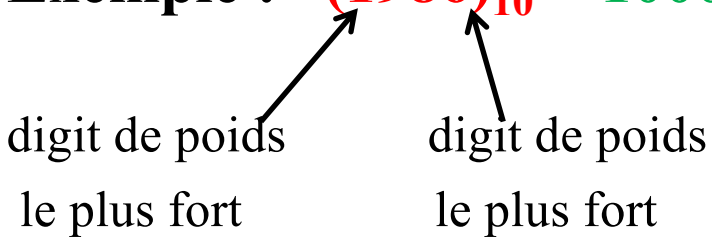
1. **Système décimal: système de calcul usuel**

Le système de numération à base 10 est un moyen de représenter les nombres avec 10 symboles :

0 1 2 3 4 5 6 7 8 9

Selon sa position, le symbole indique une valeur particulière. Chaque position successive vers la gauche indique une valeur dix fois plus importante que celle juste à droite:

Exemple : $(1986)_{10} = 1000 + 900 + 80 + 6 = 1.10^3 + 9.10^2 + 8.10^1 + 6.10^0$



Systèmes de numération:

2. Système binaire (2)

- ✓ Le système de base 2 permet de représenter les nombres avec deux symboles 0 ou 1.
- ✓ Origine de l'utilisation du binaire : absence ou présence de courant électrique .

Selon sa position, le symbole (bit) indique une valeur particulière.

Exemple : $(1010)_2 = 1.2^3 + 0.2^2 + 1.2^1 + 0.2^0$

MSB : bit de poids le plus fort

LSB

Systèmes de numération:

3. Système hexadécimal :

Le système de base 16 permet de représenter les nombres avec 16 symboles :

0 1 2 3 4 5 6 7 8 9 A B C D E F

Exemple : \$1AE = $1.16^2 + 10.16^1 + 14.16^0$

La numération hexadécimale permet de représenter d'une manière plus compacte le code binaire :

Exemple : \$FF= 1111 1111

Conversion du système décimal au système binaire :

Exemples : $25_{10} = N_2$

1. Divisions successives :

25 /2 = 12 reste 1 LSB
12 /2 = 6 reste 0
6 / 2 = 3 reste 0
3 / 2 = 1 reste 1
1 / 2 = 0 reste 1 MSB

$N_2 = 11001$

2. Soustractions successives :

$25 - 2^4 = 9$
$9 - 2^3 = 1$
$1 - 2^0 = 0$

$N_2 = 1.2^4+1.2^3+0.2^2+0.2^1+1.2^0$

$N_2 = 11001$

Conversion du système binaire au système décimal :

Exemples : $11001_2 = N_{10}$

$N_{10} = 1.2^4+1.2^3+0.2^2+0.2^1+1.2^0 = 16 + 8 +0 + 0 + 1 = 25$

Conversion du système décimal au système hexadécimal :

Exemples : **$4113_{10} = N_{16}$**

Divisions successives :

4113 /16 = 257 reste 1 LSB

257 /16 = 16 reste 1

16 / 16 = 1 reste 0

1 / 16 = 0 reste 1

$N_{16} = 1011_{(16)}$

Conversion du système hexadécimal au système décimal :

Exemples : $1011_{16} = N_{10}$

$\$1011 = 1.16^3 + 0.16^2 + 1.16^1 + 1.16^0$

$= 4096 + 0 + 0 + 16 + 1$

$= 4113_{10}$

Conversion du système binaire au système hexadécimal

✓ **1ère méthode :**

On procède en 2 étapes :

1. passer du binaire au décimal
2. passer du décimal à l'hexadécimal

Exemples : $10110111101_2 = N_{16}$

$10110111101_{(2)} = 1469_{(10)} = 5BD_{(16)}$.

✓ **2ème méthode**

On découpe le nombre binaire en **quartets,** à partir de la droite puis on remplace chaque quartet par le symbole hexadécimal correspondant.

Exemples : $10110111101_2 = N_{16}$

$10110111101_{(2)} = 0101\ 1011\ 1101 = 5BD_{(16)}$

Codage des nombres entiers relatifs:

Codage des nombres entiers relatifs:

Bit de signe: c'est le bit de poids fort

MSB = 0 → nombre positif

MSB = 1 → nombre négatif

Problème 1 : Deux codes différents pour le Zéro!: Ambigüité

Problème 2 : Règles de l'addition non respectée.

Exemple : 5 – 3 = 5 + (-3)

5→ 0101

-3 → + 1011

1 0000 → erreur!

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

Codage des nombres entiers relatifs:

Pour éliminer les problèmes cités précédemment, on utilise un codage en **complément à 2**

Principe du complément à deux :

Soit à représenter un nombre négatif. **-5** sur **4 bits**

Prenons son opposé (son équivalent en positif) : **5**

On le représente en base 2 sur 3 bits : **101**

On complémente chaque bit (remplacer les zéros par des 1 et vice-versa) **: 010**

On ajoute 1 : **011**

Et on complète avec le bit de signe

Résultat: -5 → 1011

Remarque 1:
la somme d'un nombre et de son complément à deux est nulle.

Remarque 2:
Si on travaille sur n bits, on peut représenter des nombres :

$0 \rightarrow 2^{(n-1)}-1$ $\quad$ $-1 \rightarrow -2^{(n-1)}$

$-2^{(n-1)} < N < 2^{(n-1)}-1$

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

Conversion des nombres réels en binaire:

Exemple: conversion de **12,6875** en binaire :

$\mathbf{(12)_{10} = (1010)_2}$ obtenu par division successive par 2 par exemple

La conversion de **0,6875** se fait par multiplication successive par 2.

0,6875 x 2 = 1,375

0,375 x 2 = 0,750

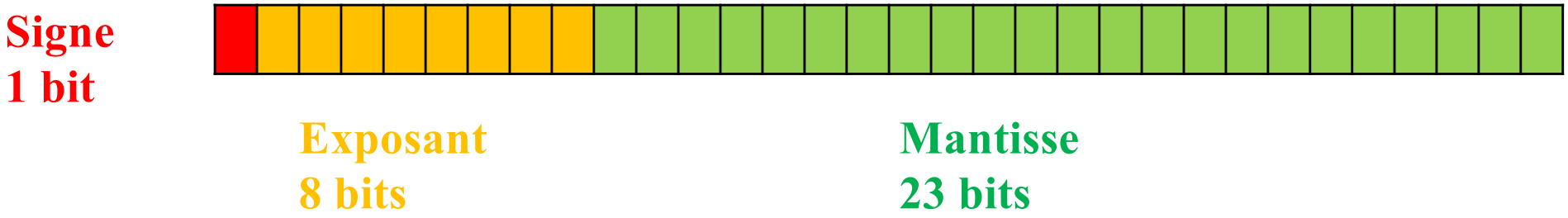
0,750 x 2 = 1,5 ⟶ $\mathbf{(0,6875)_{10} = (0,1011)_2}$

0,5 x 2 = 1,0

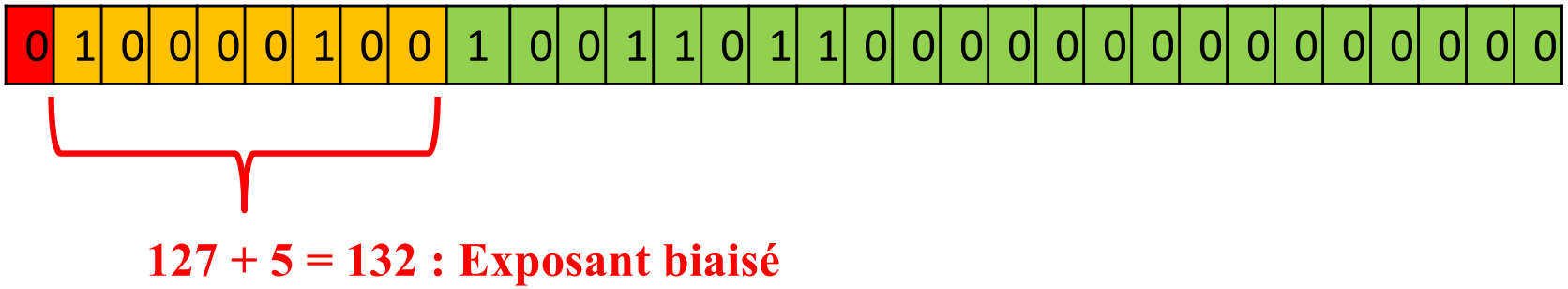
$(12,6875)_{10}=(1010,1011)_2 = 1.2^3 + 0.2^2 +1.2^1 + 0.2^0 + 1.2^{-1} + 0.2^{-2} +1.2^{-3} + 1.2^{-4}$

Codage des nombres réels: norme IEEE 754

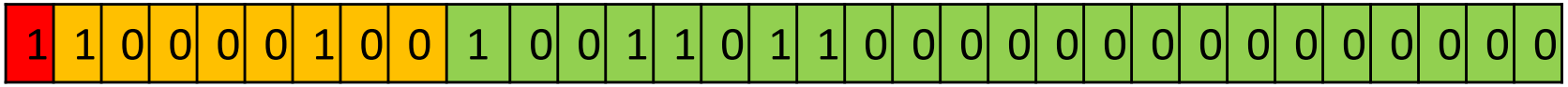
Codage en virgule flottante: 32 bits répartis comme indiqué ci-dessous



Exemple : **51,37510** = $(110011,0110)_2 = (1,100110110 \cdot 2^5)_2$



-51,37510



Codage des caractères :

Un ordinateur ne traite que des nombres. Pour lui permettre de traiter des caractères, il faut affecter à chacun des caractère un nombre qui le représente.

Le code ASCII utilise 7 bits pour encoder les caractères. Cela permet d'encoder 128 signes différents. les 32 premières valeurs sont des caractères de contrôles (tabulation, fin de ligne...). Dans les 96 valeurs restantes on trouvera 26 lettres minuscules, 26 lettres majuscules, 10 chiffres et les signes de ponctuation (,;:.!?=+...).

Standard ISO 8859-1 (Latin 1) : évolution du code ASCII, huit bits pour représenter entre autres les caractères accentués.(256 caractères).

Unicode : deux ou trois octets pour coder des jeux de caractères non latins.

L'UTF-8 (Unicode Transformation Format), est l'encodage le plus répandu. Il contient la plupart des caractères utilisés par les langues modernes les plus courantes dans le monde.(120000 caractères)

MSB

LSB

		000	001	010	011	100	101	110	111
		0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	10	LF	SUB	*	:	J	Z	j	z
1011	11	VT	ESC	+	;	K	[	k	{
1100	12	FF	FS	,	<	L	\	l	!
1101	13	CR	GS	-	=	M	]	m	}
1110	14	SO	RS	.	>	N	'	n	~
1111	15	SI	US	/	?	O	-	o	DEL

NUL	Absence de caractère,blanc,espace
SOH	Start of Heading : début en-tête
STX	Start of Text
ETX	End of Text
EOT	End of Transmission
ENQ	Enquiry Demande
ACK	Acknowledge, accusé réception
BEL	Bell, sonnette
BS	Backspace marche arrière 1 caractère
HT	Horizontale Tabulation
LF	Line Fed retour à la nvelle ligne
VT	Vertical Tabulation
FF	Form Fed, passage page suivante
CR	Carriage Return, retour chariot
SO	Shift Out caractère suivant non std
SI	Shift In retour au caractères std
DLE	DataLink Escape chgmt de signific.
NAK	Negative Acknoledgment
SYN	Synchronous, caractère de synchro.
ETB	End Of Transmission Block
CAN	Cancel annulation de la donnée précédente
SUB	Substitute remplacement
ESC	Escape caractère de ctrl d'extension
FS	File Separator
GS	Groupe Separator
RS	Record Separator

TD1 : codage des entiers naturels :

1. Ecrire $35_{(10)}$ et $70_{(10)}$ en binaire.
2. Ecrire un algorithme qui convertit un nombre entier positif saisi auprès de l'utilisateur en sa représentation binaire associée. Implémenter en Python.
3. Ecrire les nombres $6887_{(10)}$ et $1123_{(10)}$ en code hexadécimal.
4. Ecrire un algorithme qui convertit un nombre entier positif saisi auprès de l'utilisateur en sa représentation hexadécimale associée. Implémenter en Python.
5. Ecrire $2A3_{(16)}$ et $1AD7_{(16)}$ en code décimal.
6. Ecrire le nombre $70E_{(16)}$ en code binaire.
7. Implémenter un algorithme qui convertit une représentation binaire saisie auprès de l'utilisateur en son nombre décimal positif associé.
8. Elaborer un programme qui permet de convertir un nombre entier positif N exprimé en base 10 en un nombre exprimé en base B quelconque.

TD1: Codage des caractères :

1. Dans le code ASCII, la lettre (« o ») s'écrit $111_{(10)}$. Quel est son code binaire ?
2. Dans ce même code, la lettre (n) s'écrit $01101110_{(2)}$. Quel est son code décimal ?
3. Codez en ASCII la phrase suivante : **nous sommes en terminale.**
4. Décoder la séquence suivante :

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	4F	4E	20	53	27	45	43	4C	41	54	45	20	45	4E	20	49
00000010	53	4E														

TD1: Codage des caractères :

Editeur hexadécimal :

1. Ouvrir l'accessoire « Bloc-notes » présent dans les accessoires de Windows. (On peut aussi utiliser un autre éditeur de texte (notepad++))
2. Ecrire en majuscule votre PRENOM, un espace puis votre NOM puis aller à la ligne.
3. Ecrire **TERMINALE S** puis aller à la ligne.
4. Ecrire le nom de la ville toujours en majuscule.
5. Enregistrer ce texte en tant que « texte brut » (extension .txt) en précisant ANSI dans la rubrique codage.
6. Ouvrir ce fichier dans l'éditeur hexadécimal « EditHexa » ou un autre éditeur hexadécimal.
7. Relever le codage hexadécimal des données saisies et vérifier la correspondance avec l'ASCII.