Math93.com

Cours - NSI
Représentation des données
Types et valeurs de base

1 Un peu d'histoire

Voir le site : https ://www.math93.com/histoire-des-maths/histoire-des-nombres/154-histoire-des-nombres.html

1.1 Règles de construction des numérotations

Les règles de **construction des numérotation** écrite sont simples :

1. il faut permettre une **lecture sans ambiguïté**, une même écriture ne devant pas représenter deux nombres différents.
2. Il faut représenter un **maximum de nombres** avec un minimum de symboles.

1.2 Les bases

C'est l'usage d'une base qui permettra de répondre au mieux aux contraintes posées.
Au lieu de compter uniquement par unités, on compte "par paquets". La plus fréquente est la base décimale (10), mais on trouve également :

- la base sexagésimale (60), utilisée par les Sumériens et parfois au moyen âge en France,
- la base vicésimale (20), utilisée par les Mayas ou duodécimale (12),
- la base 16 (système hexadécimal), en informatique, facilitant les conversions en base 2 en regroupant des chiffres binaires, 16 étant une puissance de 2 ;
- la quinaire (5), utilisée aussi par les Mayas
- et la binaire (2) utilisé en électronique numérique et informatique .

Chez les Mayas, le moyen le plus simple pour représenter les nombres était un système utilisant, le point valait 1, la barre 5 et un zéro. On les trouve sur le codex de Dresde.

1.3 La numération indienne de position

Remarque historique

L'invention de cette numération dans l'Inde au 5e siècle.
Les chiffres de "un" à "neuf" ont été inventés en Inde avant notre ère. Ils apparaissent dans des inscriptions de Nana Ghât au 3e siècle av.J.-C. , mais le principe de position n'y est pas appliqué.

La numération de position avec un zéro (qui était un point à l'origine), a été inventé au cours du 5e siècle. Dans un traité de cosmologie écrit en sanscrit en 458, le "LOKAVIBHAGA", "les parties de l'univers", on voit apparaître le nombre 14 236 713 écrit en toute lettres ("un" "quatre" ...). dans ce texte, on trouve aussi le mot "sunya", "le vide", qui représente le zéro. C'est à ce jour le document le plus ancien faisant référence de cette numération.

Propagation de cette numération.
En 773, arriva à Bagdad une ambassade indienne avec un présent pour le calife MANSOUR et les savants arabes qui l'entouraient : le calcul et les chiffres.
Muhammad ibn Musa al-Khuwârizmi a écrit le premier ouvrage en langue arabe présentant la numération indienne de position au 9e siècle, "*livre de l'addition et de la soustraction d'après le calcul des Indiens*".
C'est par cet ouvrage que le calcul indien pénétra dans l'Occident chrétien.

Maintes fois traduit en latin à partir du 12e siècle, sa célébrité fût telle que ce calcul fut nommé algorisme, d'Algorismus latinisation d'al-Khuwârizmi.
Au Xe siècle, le moine français Gerbert d'AURILLAC apprit la nouvelle numération chez les Maures d'Espagne et, grâce aux chaires qu'il occupa dans les établissement religieux d'Europe, il put introduire le nouveau système en occident.

2 Représentation des entiers naturels

2.1 La notion de base

— **Une base (de numération)** désigne la valeur dont les puissances successives interviennent dans l'écriture des nombres dans la numération positionnelle.

— En base N, on a donc besoin de N chiffres (digits), de 0 à $N-1$.
Par exemple, en base 10, on a besoin de 10 chiffres, de 0 à 9, en base 2, on a besoin de 2 chiffres de 0 et 1, etc.

— Pour les bases supérieures à 10, il faut d'autres chiffres. On aurait pu inventer des symboles nouveaux. On convient plutôt d'utiliser les premières lettres de l'alphabet.
Pour la base 16, on utilise les chiffres de 0 à 9 puis A (pour 10), B (pour 11), C (pour 12) ... , F (pour 15)

Exemple

— BASE 2 :

$$1011_2 = 1 \times \boxed{2^3} + 0 \times \boxed{2^2} + 1 \times \boxed{2^1} + 1 \times \boxed{2^0}$$

— BASE 10

$$2048_{10} = 2 \times \boxed{10^3} + 0 \times \boxed{10^2} + 4 \times \boxed{10^1} + 8 \times \boxed{10^0}$$

— BASE 16

$$204C_{16} = 2 \times \boxed{16^3} + 0 \times \boxed{16^2} + 4 \times \boxed{16^1} + 12 \times \boxed{16^0}$$

2.2 Bases et notations

— **En mathématiques : base en index**
Pour n'importe quelle base, on a l'habitude de l'indiquer en indice du nombre.
Par exemple $(100111)_2$ pour le nombre dont le développement en base 2 est 100111, ou encore $(172)_8$ pour le nombre dont le développement en base 8 est 172.

— **En informatique : base en préfixe**
En plus de cette notation, il en existe d'autres, notamment employées en informatique.
Par exemple sous Python (ou en C, C++, Java) pour indiquer qu'on est en **base 2 on ajoute le préfixe 0b** et le **préfixe 0x pour la base 16** :

```
# CODE PYTHON
>>> bin(45) # la fonction bin permet de convertir un décimal en base 2
'0b101101'
# Remarquez l'utilisation des guillemets pour le résultat,
# il est de type chaîne de caractères
>>> hex(45) # la fonction hex permet de convertir un décimal en base 16
'0x2d'
```

2.3 Le binaire

2.3.1 Vocabulaire

Tout information manipulée par un ordinateur est représentée par des mots composés uniquement de 0 et 1 (les bits). Un octet (byte) est un mot binaire de 8 bits et un quartet = un mot binaire de 4 octets ou 32 bits.

2.3.2 Méthode de conversion : de base 10 en base 2

En binaire chaque rang ne peut prendre que deux valeurs (il pouvait en prendre dix en décimal). Donc, dès que le rang atteint sa deuxième - la plus haute - valeur on change de rang. En binaire, un rang commence à 0 et se termine à 1.

base 10	0	1	2	3	4	5	6	7	8	9	10	11	12
base 2	0_2	1_2	10_2	11_2	100_2	101_2	110_2	111_2	1000_2	1001_2	1010_2	1011_2	1100_2

Exercice 2.1 : Compléter le tableau

base 10	13	14	15	16	17	18	19	20	21
base 2									

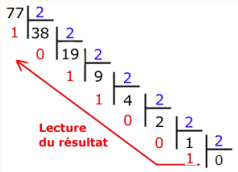
Méthode : De la base 10 à la base 2

On procède par **divisions successives par 2 des quotients**, et on **récupère les restes** :

- On considère un entier en base 10.
- On le divise par 2 et on note le reste de la division (c'est soit un 1 soit un 0).
- On refait la même chose avec le quotient précédent, et on met de nouveau le reste de coté.
- On re-itère la division, et ce jusqu'à ce que le quotient est 0.
- Le nombre en binaire apparaît : le premier à placer est le dernier reste non nul. Ensuite, on remonte en plaçant les restes que l'on avait. On les place à droite du premier 1.

Exemple

Écrivons 77 en base 2. Il s'agit de faire une suite de divisions euclidiennes par 2. Le résultat sera la juxtaposition des restes. Le schéma ci-dessous explique la méthode :



$77_{10} = 1001101_2$

```
# On peut vérifier avec la fonction bin de Python :
>>> bin(77)
'0b1001101'
# Ou pour passer de la base 2 à la base 10
>>> int('1001101',2)
77
```

2.3.3 Méthode de conversion : de base 2 en base 10

Méthode : De la base 2 à la base 10

On utilise le fait qu'en base 2, le chiffre le plus à droite représente les unités, celui d'avant les paquets de 2, puis les paquets de 2^2 ...

$$1011_2 = 1 \times \boxed{2^3} + 0 \times \boxed{2^2} + 1 \times \boxed{2^1} + 1 \times \boxed{2^0} = 11_{10}$$

```
# Sur Python, il suffit d'écrire le préfixe 0b devant le nombre :
>>> 0b1011
11
```

2.3.4 Exercices

Exercice 2.2

Trouver la représentation en base 10 du nombre $1111\ 1111_2$.

Corrigé

— Méthode 1.

$$1111\ 1111_2 = 1 \times \boxed{2^7} + 1 \times \boxed{2^6} + \cdots + 1 \times \boxed{2^1} + 1 \times \boxed{2^0} = 255_{10}$$

— Méthode 2 : On peut aussi plus simplement considérer que c'est l'entier qui précède

$$10000\ 0000_2 = 2^8_{10} = 256_{10}$$

on retrouve bien 255

Exercice 2.3

En base 10, une série de 2 caractères peuvent représenter 100 nombres entiers différents (de 0 à 99). Sur le même principe combien de nombres entiers naturels peut on représenter avec un nombre n de caractères en binaire.

Corrigé

Avec un nombre n de caractères en binaire ou un mot de n bits, on peut représenter les nombres entiers de 0 à $2^n - 1$ soit 2^n entiers.

2.4 Représentation hexadécimale

Comme les représentations décimale et binaire, la représentation hexadécimale est un système de numération positionnel en base 16. Tout entier naturel non nul n dans ce système peut s'écrire sous la forme :

$$n = (h_p ... h_0)_{16}$$

avec,

$$h_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$$

Le tableau suivant présente les conversions des premiers entiers naturels entre les trois représentations précédentes.

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Décimal	Binaire	Hexadécimal
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

2.4.1 Conversion décimal/Hexadécimal

Là encore, convertir un entier du système hexadécimal au système décimal et réciproquement peut aisément se faire.

$$(B7)_{16} \leftrightarrow 11 \times 16^1 + 7 \times 16^0 = 183$$

On procède par divisions successives par 16 pour convertir en entier base 10 en base 16.
Par exemple ici :

		Reste base 10		Base 16
$183 = 16 \times 11 + 7$	$\Longrightarrow$	reste 7	$\longleftarrow$	7
$11 = 16 \times 0 + 11$	$\Longrightarrow$	reste 11	$\longleftarrow$	B

Puis on lit les restes de bas en haut :

$$(B7)_{16} \leftrightarrow (183)_{10}$$

2.4.2 Conversion Binaire/Hexadécimal

La conversion hexadécimal $\leftrightarrow$ binaire peut également se faire rapidement : chaque chiffre hexadécimal équivaut à exactement quatre chiffres binaires.

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Décimal	Binaire	Hexadécimal
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

— Reprenant l'exemple du nombre B7, le tableau précédent indique que 7 peut être converti en 0111 et B en 1011. Ce qui donne la représentation binaire 01111011 ou 1111011 si on omet le premier zéro.

$$\begin{cases}(7)_{16} = 7_{10} = (0111)_2 \\ (B)_{16} = 11_{10} = (1011)_2\end{cases} \Longrightarrow (B7)_{16} = (1011\ 0111)_2$$

— À la représentation binaire 110 1110 peut être associée deux blocs de quatre chiffres : 0110 (on ajoute 1 zéro) et 1110. Le premier bloc se convertit en 6 et le second en E de sorte que la représentation hexadécimale de l'entier est 6E.

$$\begin{cases}(0110)_2 = (6)_{16} \\ (1110)_2 = (E)_{16}\end{cases} \Longrightarrow (0110\ 1110)_2 = (6E)_{16}$$

Particulièrement pratique pour représenter de grands entiers sous forme compacte, la représentation hexadécimale est fréquemment utilisée en électronique numérique et dans le monde des réseaux pour décrire des adresses.

Exercice 2.4

Donner la représentation binaire de $(ABC1)_{16}$ et la représentation en hexadécimale de $(1011\ 0110\ 1101)_2$

Corrigé

— $(ABC1)_{16} = (1010\ 1010\ 1100\ 0001)_2$

— $(1011\ 0110\ 1101)_2 = (B6D)_{16}$

2.5 Codage binaire

2.5.1 Des bits

D'un point de vue matériel, un ordinateur est un ensemble de composants électroniques parcourus par des courants électriques. Par convention, le passage d'un courant dans un composant est codé par le chiffre 1, l'absence de courant étant codé par un 0. Ainsi, toute information stockée dans un ordinateur peut être codée par une suite finie de 0 et de 1 appelée suite de bits (binary digits). Une convention fixe la taille de ces suites finies de bits.

Par exemple, un codage sur 4 bits signifie qu'une information est représentée en machine à l'aide de quatre 0 ou 1. L'ordre de ces bits importe de sorte que 0111 ne code pas la même information que 1110. En outre, les zéros présents en début de codage sont indispensables. Ainsi, sur 4 bits, $2^4 = 16$ entiers naturels peuvent être codés ; par exemple, les entiers de 0 à 15.

$0 \leftrightarrow 0000$; $1 \leftrightarrow 0001$; $2 \leftrightarrow 0010$; $3 \leftrightarrow 0011$; $4 \leftrightarrow 0100$; $5 \leftrightarrow 0101$; $6 \leftrightarrow 0110$; $7 \leftrightarrow 0111$; $8 \leftrightarrow 1000$; $9 \leftrightarrow 1001$

$10 \leftrightarrow 1010$; $11 \leftrightarrow 1011$; $12 \leftrightarrow 1100$; $13 \leftrightarrow 1101$; $14 \leftrightarrow 1110$; $15 \leftrightarrow 1111$

Nombre d'entiers codés sur n bits

Si n est un entier naturel non nul, **2^n entiers** peuvent être codés en binaire sur **n bits** :

$$0 \longrightarrow \cdots \longrightarrow \left(2^n - 1\right)$$

- Pour $n = 4$ bits : on a $2^4 = 16$ entiers possibles de 0 à $2^4 - 1 = 15$
- Pour $n = 8$ bits : on a $2^8 = 256$ entiers possibles de 0 à $2^8 - 1 = 255$
- Pour $n = 16$ bits : on a $2^{16} = 65\,536$ entiers possibles de 0 à $2^{16} - 1 = 65\,535$

2.5.2 Arithmétique élémentaire

Cela signifie que les entiers plus grands que 2^n ne peuvent pas être codés sur n bits. La capacité de codage sur n bits est dépassée. Soit on ajoute des bits pour coder de plus grands entiers, soit on se contente de coder les entiers sur n bits. Alors attention aux additions ...

Exemple

Illustrons notre propos avec $n = 4$ bits et deux entiers $n_1 = 3$ et $n_2 = 4$. L'addition de ces entiers est l'entier $n_3 = 7$. En binaire, les additions sont effectuées suivant les règles suivantes.

- 0 + 0 donne 0 avec une retenue égale à 0 ;
- 0 + 1 donne 1 avec une retenue égale à 0 ;
- 1 + 0 donne 1 avec une retenue égale à 0 ;
- 1 + 1 donne 0 avec une retenue égale à 1 ;

Le codage de n_1 et n_2 sur 4 bits donne respectivement 0011 et 0100. L'addition de ces codages peut être posée et s'écrit :

Retenue	0	0	0	0	
		0	0	1	1
+		0	1	0	0
	0	0	1	1	1

Le résultat obtenu, à savoir 0111 sur 4 bits, correspond au codage binaire de 7.

Choisissons à présent $n_1 = 11$ et $n_2 = 13$ dont les codages binaires sur 4 bits sont respectivement 1011 et 1101. L'addition de ces codages donne alors :

Retenue	1	1	1	1	
		1	0	1	1
+		1	1	0	1
	1	1	0	0	0

Le résultat obtenu est 11000. En codant sur 4 bits, seuls les 4 bits de plus faibles poids sont conservés, à savoir 1000. Ce codage est celui de l'entier 8 et non celui de la somme $11 + 23 = 24$. On peut remarquer $8 = 24 - 16$; le résultat est la somme modulo 16 (c'est à dire à une nombre de fois 16 près).

L'addition des codages réalise la somme modulo 2^n.
Par exemple 255 + 1 renvoie 0 plutôt que 256 sur certaines machines, en l'occurrence les machines codant les entiers naturels sur 8 bits ou moins!

2.5.3 Overflow

Remarque historique

Le vol inaugural 501 du lanceur européen Ariane 5 du 4 juin 1996 s'est soldé par un échec, causé par un dysfonctionnement informatique, qui vit la fusée se briser et exploser en vol seulement 36,7 secondes après le décollage.
Tout tenait à une seule petite variable : celle allouée à l'accélération horizontale. En effet, l'accélération horizontale maximum produite par Ariane 4 donnait une valeur décimale d'environ 64. La valeur d'accélération horizontale de la fusée étant traitée dans un registre mémoire à 8 bits, cela donne en base binaire $2^8 = 256$ valeurs disponibles, un nombre suffisant pour coder la valeur 64, qui donne en binaire $(100\,0000)_2$ et ne nécessite que 7 bits.
Mais Ariane 5 était bien plus puissante et brutale : son accélération pouvait atteindre la valeur 300, qui donne $(100101100)_2$ en binaire et nécessite un registre à 9 bits.
Ainsi, la variable codée sur 8 bits a connu un dépassement de capacité. Il aurait fallu la coder sur un bit de plus, donc 9 bits, ce qui aurait permis de stocker une valeur limite de $2^9 - 1 = 511$, alors suffisante pour coder la valeur 300. De ce dépassement de capacité a résulté une valeur absurde dans la variable, ne correspondant pas à la réalité. Par effet domino, le logiciel décida de l'autodestruction de la fusée à partir de cette donnée erronée.

3 Représentation des entiers relatifs

3.1 Faire le TD d'introduction

▷ **TD NSI** : représentation binaire d'un entier relatif
Disponible sur votre page : https ://www.math93.com/lycee/nsi-1ere/nsi-1ere.html

3.2 Écriture d'un entier relatif en binaire sur N bits

Méthode : complément à 2

On considère un nombre relatif n en base 10.

1. Si $n \geq 0$, on utilise la méthode classique de conversion en base 2 (cf. 2.3.2).
2. Si n est négatif, on code $|n|$ en binaire puis :
   - Étape 1 : On prend le complément à 1 (NON logique) de l'écriture binaire, c'est à dire qu'on remplace les 1 par des 0 et les 0 par des 1.
   - Étape 2 : On ajoute 1 au résultat, plus exactement $(0000\ 0001)_2$ si on est sur 8 bits.
   - On remarque que l'écriture binaire de l'entier négatif n correspond à celle $n + 2^N$.

Exemple

Représentation binaire de $(-8)_{10}$ sur 8 bits.

- Étape 1 : On prend le non logique ou complément à 1.
  Par exemple (sur 8 bits) :

$$(8)_{10} = (0000\ 1000)_2 \underset{\text{Par complément à 1}}{\Longrightarrow} (1111\ 0111)_2$$

- Étape 2 : On ajoute 1 (d'où le nom, complément à 2)

$$(1111\ 0111)_2 + (0000\ 0001)_2 = (1111\ 1000)_2$$

- Donc

$$(8)_{10} = (0000\ 1000)_2 \underset{\text{Par complément à 2}}{\Longrightarrow} (-8)_{10} = (1111\ 1000)_2$$

- On remarque que l'écriture binaire de l'entier négatif n correspond à celle $n + 2^8$ car en binaire

$$(1111\ 1000)_2 = 248_{10} \Longrightarrow 248 - 2^8 = -8$$

3.3 Astuce

Astuce

Il suffit en fait en lisant le nombre binaire de droite à gauche :

- Étape 1 : de conserver tous les 0 et le premier 1 ;
- Étape 2 : de changer tous les autres digits à gauche du premier 1.

Une petite vidéo : https ://www.youtube.com/watch ?v=bNTyHfTnqEU
Juste une petite confusion entre droite et gauche mais cela ne gène pas la compréhension.

3.4 Définition et exemple

Définition 1

Soit la représentation binaire d'un entier naturel sur n bits :

$$\left(\boxed{x_n}\, x_{n-1}\cdots x_1\right)_2$$

— x_0 est le bit de poids faible ;

— $\boxed{x_n}$ est le **bit de poids fort** (et indique **le signe**).
S'il vaut 1, le nombre en base 10 est négatif, s'il vaut 0, le nombre en base 10 est positif.

On a vu dans le TD d'approche que sur 4 bits on a les résultats suivants. Notez que le bit de poids fort donne bien le signe :

−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7
1 000	1 001	1 010	1 011	1 100	1 101	1 110	1 111	0000	0001	0010	0011	0100	0101	0110	0111

3.5 Plus grand et plus petit relatifs sur n bits

Soit une machine programmée sur n bits. Il y a donc 2^n nombres relatifs possibles.
On garde le $(000\cdots00)_2$ pour le 0. On va donc en prendre la moitié soit $\dfrac{2^n}{2} = 2^{n-1}$ pour les négatifs et l'autre moitié pour les positifs ou nul.

-2^{n-1}	...	-1	0	1	...	$2^{n-1}-1$
$100\cdots00$	...	$111\cdots11$	$00\cdots00$	$00\cdots001$	...	$011\cdots11$

Plus grand et plus petit relatifs sur n bits ($n > 0$)

1. Sur n bits on peut représenter 2^n entiers relatifs.
2. La moitié soit 2^{n-1} sont des entiers positifs ou nuls : de 0 à $\boxed{2^{n-1}-1}$.
3. La moitié soit 2^{n-1} sont des entiers strictement négatifs : de $\boxed{-2^{n-1}}$ à (-1).
4. Les 2^n entiers relatifs sont donc compris entre $\boxed{-2^{n-1}}$ et $\boxed{2^{n-1}-1}$:

$$\boxed{-2^{n-1}}\ldots \longrightarrow -1 \longrightarrow 0\ldots \longrightarrow \boxed{2^{n-1}-1}$$

Plus grand et plus petit relatifs sur n bits

— Sur 4 bits : on a $2^4 = 16$ relatifs possibles de -8 à 7 :

$$-2^3 = -8 \longrightarrow \cdots \longrightarrow \cdots \longrightarrow 2^3 - 1 = 7$$

— Sur 8 bits : on a $2^8 = 256$ relatifs possibles de -128 à 127 :

$$-2^7 = -128 \longrightarrow \cdots \longrightarrow \cdots \longrightarrow 2^7 - 1 = 127$$

— Sur 16 bits : on a $2^{16} = 65\,536$ relatifs possibles de $-32\,768$ à $32\,767$:

$$-2^{15} = -32\,768 \longrightarrow \cdots \longrightarrow \cdots \longrightarrow 2^{15} - 1 = 32\,767$$

— Sur 32 bits : on a $2^{32} = 4\,294\,967\,296$ relatifs possibles de $-2\,147\,483\,648$ à $2\,147\,483\,647$:

$$-2^{31} = -2\,147\,483\,648 \longrightarrow \cdots \longrightarrow \cdots \longrightarrow 2^{31} - 1 = 2\,147\,483\,647$$

4 Représentation approximative des réels (flottants)

4.1 Nombres décimaux : rappels

Définition 2

Un **nombre décimal** est un nombre qui peut s'écrire sous la forme $\frac{a}{10^n}$, où a est un entier relatifs et n un entier.

Par exemple :

$$\frac{1235}{10^2} = 12,35 = 1\times 10^1 + 2\times 10^0 + \underbrace{3\times 10^{-1}}_{\text{3 dixièmes}} + \underbrace{5\times 10^{-2}}_{\text{5 centièmes}}$$

4.2 Nombres dyadiques (par analogie)

Définition 3

Un **nombre dyadique** est un nombre qui peut s'écrire sous la forme $\frac{a}{2^n}$, où a est un entier relatifs et n un entier.

Par exemple :

$$\frac{13}{4} = 3,25 = \frac{13}{2^2} = 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}$$

Méthode

Pour obtenir le développement dyadique d'un nombre dyadique $\frac{a}{2^n}$ on prend le binaire correspondant à a et on insère une virgule avant le n-ième bit en partant de la fin.

Par exemple : pour $\frac{13}{2^2}$

- Étape 1 : $a = 13 = (1101)_2$;
- Étape 2 : Le développement dyadique de $\frac{13}{2^2}$ est donc $(11,01)_2$

$$\boxed{\frac{13}{2^2} = 3,25 = (11,01)_2} \iff 3,25 = \frac{13}{2^2} = \boxed{1}\times 2^1 + \boxed{1}\times 2^0 + \boxed{0}\times 2^{-1} + \boxed{1}\times 2^{-2}$$

On peut faire une analogie avec les décimaux, par exemple :

13	$13 = (1101)_2$
$\frac{13}{10} = 1,3$	$\frac{13}{2} = (110,1)_2$
$\frac{13}{10^2} = 0,13$	$\frac{13}{2^2} = (11,01)_2$
$\frac{13}{10^3} = 0,013$	$\frac{13}{2^3} = (1,101)_2$

4.3 Conversion d'un nombre décimal en base 2

Comment passer d'un décimal à son écriture en binaire :

Méthode

Pour obtenir le développement dyadique d'un nombre décimal quelconque .

- Étape 1 : on convertit sa partie entière en base 2 ;
- Étape 2 : on multiplie la partie fractionnaire par 2 puis on note sa partie entière .
- Étape 3 : on recommence cette opération avec la partie fractionnaire du résultat et ainsi de suite
- Test d'arrêt : On s'arrête quand la partie fractionnaire est nulle ou quand la précision souhaitée est atteinte.
- Conclusion : la partie fractionnaire dans la base 2 est alors la concaténation des parties entières obtenues dans l'ordre de leur calcul.

Par exemple : pour $3,25$

- $3 = \boxed{(11)_2}$
- $0,25 \times 2 = 0,5 = \boxed{0} + 0,5$
- $0,5 \times 2 = 1 = \boxed{1} + 0$

On retrouve bien :

$$\boxed{3,25 = (11,01)_2}$$

Exercice 4.5

Conversion de 12,6875 en binaire :

Corrigé

La partie entière : $12 = (1100)_2$

$$\begin{array}{llllll} 0,6875 & \times 2 & = & 1,375 & = & \boxed{1} + 0,375 \\ 0,375 & \times 2 & = & 0,75 & = & \boxed{0} + 0,75 \\ 0,75 & \times 2 & = & 1,5 & = & \boxed{1} + 0,5 \\ 0,5 & \times 2 & = & 1 & = & \boxed{1} + 0 \end{array}$$

$$\boxed{12,6875 = (1100,1011)_2}$$

4.4 Nombre décimaux non dyadiques

Il existe des décimaux qui ne sont pas dyadiques, qui n'admettent pas de développement dyadique fini.
Par exemple on obtient :

$$0,1 = (0,000110011001100110011001100110011\cdots)_2$$

Retrouvez ce résultat :

Exercice 4.6

Conversion de 0,1 en binaire :

Corrigé

La partie entière : $0 = (0)_2$

$$\begin{array}{llllll}
0,1 & \times 2 & = & 0,2 & = & \boxed{0} + 0,2 \\
0,2 & \times 2 & = & 0,4 & = & \boxed{0} + 0,4 \\
0,4 & \times 2 & = & 0,8 & = & \boxed{0} + 0,8 \\
0,8 & \times 2 & = & 1,6 & = & \boxed{1} + 0,6 \\
0,6 & \times 2 & = & 1,2 & = & \boxed{1} + 0,2 \\
0,2 & \times 2 & = & 0,4 & = & \boxed{0} + 0,4 \\
0,4 & \times 2 & = & 0,8 & = & \boxed{0} + 0,8 \\
0,8 & \times 2 & = & 1,6 & = & \boxed{1} + 0,6 \\
0,6 & \times 2 & = & 1,2 & = & \boxed{1} + 0,2
\end{array}$$

Le processus se répète ... donc :

$$\boxed{0,1 = (0,000110011001100110011001100110011\cdots)_2}$$

ou

$$\boxed{0,1 = (0,0001\underline{10011}\cdots)_2}$$

Par conséquent, leur représentation en machine sera nécessairement tronquée et engendrera des erreurs dans les opérations.

```
# Dans la console PYTHON
>>>0.1+0.1+0.1
0.30000000000000004
```

4.5 Représentation des réels : première approche

De façon analogue aux nombres décimaux, un nombre réel x sera codé en binaire à l'aide de son écriture décimale approchée.

Exercice 4.7

Conversion de $\frac{1}{3}$ en binaire :

Corrigé

On pose $x = \frac{1}{3} \approx 0,33333\cdots$, la partie entière : $0 = (0)_2$ et :

$$\begin{array}{lllll} 0,33333\cdots & \times 2 & = & 0,66666\cdots & = \boxed{0} + 0,66666\cdots \\ 0,66666\cdots & \times 2 & = & 1,333333\cdots & = \boxed{1} + 0,333333\cdots \\ 0,333333\cdots & \times 2 & = & 0,666666\cdots & = \boxed{0} + 0,666666\cdots \end{array}$$

Le processus se répète ... donc :

$$\boxed{\frac{1}{3} = (0,0101010101\cdots\cdots)_2}$$

4.6 Représentation des réels : Norme IEEE-754 (Complément) et nombre flottant

Pour un espace mémoire donné, on souhaite obtenir la représentation qui maximise à la fois l'effectif des nombres codés sur cet espace et la précision.
Représenter des réels en informatique consiste à l'approcher par un nombre décimal (un flottant). Une infinité de réels voisins auront donc la même représentation.

4.6.1 Principe de la méthode de la virgule flottante

L'utilisation de la virgule flottante est la plus répandue. Son principe consiste à séparer les chiffres significatifs et la position de la virgule dans le but d'avoir un maximum de chiffres significatifs.
Ce type de représentation est implémentées directement dans le micro-processeur.
On distingue trois parties :

- **le signe *s*** ,
- **la mantisse *m*** ,
- **l'exposant *e*** ,

Exemple

Exemple en base 10.

$$-1234,56489 = -1,23456789 \times 10^3$$

$$\begin{cases} m = 1,23456789 \\ e = 3 \end{cases}$$

Exemple en base 2.

$$(10001,11001)_2 = (1,000111001)_2 \times 2^4$$

$$\begin{cases} m = 1,000111001 \\ e = 4 \end{cases}$$

En base 2, le premier bit de la mantisse (le bit avant la virgule) sera toujours 1. On ajuste ensuite l'exposant.

4.6.2 Norme IEEE-754

La **norme IEEE-754** utilisée depuis 1995 et révisée en 2008 est la plus utilisée pour représenter les nombres à virgule flottante en base 2 sur les ordinateurs.
Elle propose deux formats (nombres de bits pour représenter les nombres) : simple précision (32 bits) et double précision (64 bits). Le premier bit de la mantisse étant toujours 1, on ne le représente pas (le cas 0 se traite à part). Sans entrer dans les détails on a :

- **Simple précision : 32 bits**
  - **le signe s** : 1 bit de signe (0 pour positif et 1 négatif)
  - **l'exposant e** : 8 bits pour l'exposant (décalé de 127).
  - **la mantisse m** : 23 bits pour la mantisse.
  - Les flottants positifs en simple précision peuvent représenter des nombres décimaux compris (approximativement) **entre 10^{-38} et 10^{38}** c'est à dire entre **entre 2^{0-127} et $2^{255-127}$**.
- **Double précision : 64 bits**
  - **le signe s** : 1 bit de signe (0 pour positif et 1 négatif)
  - **l'exposant e** : 11 bits pour l'exposant (décalé de 1023).
  - **la mantisse m** : 52 bits pour la mantisse.
  - Les flottants positifs en double précision peuvent représenter des nombres décimaux compris (approximativement) **entre 10^{-308} et 10^{308}** c'est à dire entre **entre 2^{0-1023} et $2^{2047-1023}$**.

Exemple

Par exemple le mot de 32 bits suivant :

1	10000110	10101101100000000000000

représente le décimal calculé ainsi :

- **le signe $s = -1$** : car le bit de signe est 1
- **l'exposant $e = 10000110_2$** : 8 bits pour l'exposant (décalé de 127).

$$e_{10} = 2^7 + 2^3 + 2^2 - 127 = 7$$

- **la mantisse $m = 10101101100000000000000_2$** : 23 bits pour la mantisse donc

$$m_{10} = 1 + 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-5} + 1 \times 2^{-6} + 1 \times 2^{-8} + 1 \times 2^{-9} = 1,677734375$$

Remarquer que le premier 1 de la mantisse correspond à 2^{-1}.

— Soit au final le nombre :

$$-1,677734375 \times 2^7 = -214,75$$

Exercice 4.8

A quel nombre décimal correspond le nombre suivant écrit sur 32 bits :

0	01011100	11100000000000000000000

Corrigé

— **le signe $s = +1$** : car le bit de signe est 0

— **l'exposant $e = 01011100_2$** : 8 bits pour l'exposant (décalé de 127).

$$e_{10} = 2^6 + 2^4 + 2^3 + 2^2 - 127 = -35$$

— **la mantisse $m = 11101000000000000000000_2$** : 23 bits pour la mantisse donc

$$m_{10} = 1 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = 0,875$$

— Soit au final le nombre :

$$1,875 \times 2^{-35}$$

Exercice 4.9

Ecrire la représentation sur 32 bits de $-1,25 \times 2^{11}$:

Corrigé

— **le signe $s = -1$** : donc le bit de signe est 1

— **l'exposant $e_{10} = 11_{10}$** : 8 bits pour l'exposant (décalé de 127).

$$e = 11 + 127 = 138_{10} = 128 + 8 + 2 = 10001010_2$$

.

— **la mantisse $m_{10} = 1,25 = 1 + 1 \times 2^{-2}$** : 23 bits pour la mantisse donc

$$m = 01000000000000000000000_2$$

— Soit au final le nombre :

1	10001010	01000000000000000000000

5 Représentation des chaînes de caractères

Les caractères sont des données non numériques, ce sont des symboles alphanumériques :

— Les lettres majuscules et minuscules,

— Les symboles de ponctuation (, ? . ; : !),
— Les caractères dit spéciaux ($,@, %, ...)
— Les chiffres.
— Les contrôles.

Un texte, ou chaîne de caractères, sera représenté comme une suite de caractères.
À chaque caractère correspond un nombre qui sera codé en binaire. On doit pour cela faire un choix arbitraire.

5.1 Le codage ASCCII

Remarque historique

Historiquement l'ASCII (American Standart Code for Information Interchange) a été définie pour écrire des textes en anglais. Il n'y a donc pas de lettre accentuées

C'est le codage de base en informatiques pour les caractères.

La construction de la table ASCII

— Le code ASCII est un code sur 7 bits, il y a donc $2^7 = 128$ codages possibles.
— Il y a 33 caractères de contrôle, retenons surtout l'espace ou fin de ligne.
— Les lettres se suivent dans l'ordre alphabétique.
  * Les majuscules codée de 65 à 90
  * Les minuscules codée de 97 à 122
  * Pour passer de majuscule à minuscule à on ajoute $32 = 2^5$. Cela revient alors à modifier le 6ème bit.
    G codé $71 = 0100\ \ 0111_2$ devient g codé $103 = 0110\ \ 0111_2$
— Les chiffres sont rangés dans l'ordre croissant codé entre 48 et 57. Les quatre premiers bits définissent la valeur en binaire du chiffre.
— Même si 7 bits sont suffisants pour représenter les 128 caractères, en pratique chaque caractère occupe 1 octet (8 bits) en mémoire. Le bit de poids fort est utilisé pour un somme de contrôle afin de vérifier des erreurs de transmission.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

En python, on peut manipuler le code ASCII avec chr et ord :

```
# Dans la console PYTHON
>>>chr(65)
'A'
>>>ord('a')
92
```

Retenez deux caractères de contrôle classiques (le passage à la ligne et la tabulation) :

```
# Dans la console PYTHON
>>>chr(10)
'\n'
>>>chr(9)
'\t'
```

Exercice 5.10

1. A l'aide de la table ASCII, coder la phrase suivante "L'an qui vient!"
2. Retrouver l'expression codée en binaire :
   0100 0010 0111 0010 0110 0001 0111 0110 0110 1111 0010 1100
3. Peut on coder la phrase "Un âne est passé par là?" à l'aide de la table ASCII.

5.2 Norme ISO 8859

Pour pouvoir écrire en français, on a besoin d'accent comme é, è, à, ù etc... et aussi de ç.
Pour cela l'ISO (*Organisation Internationale de Normalisation*) a proposé la **norme ISO 8859**, une extension de l'ASCII mais qui utilise 8 bits au lieu de 7. Au total 256 caractères peuvent alors être encodés.

Malgré un nombre de caractères doublé, cela reste insuffisant pour représenter tous les caractères, on définit alors plusieurs tables de correspondances (on parle aussi de pages) notées $ISO-8859-n$, où n est le numéro de la table. Bien d'indépendantes, ces tables sont conçues pour qu'elles soient compatibles entre elles. Les 128 premiers caractères sont ceux de la norme ASCII et les autres sont spécifiques.

Par exemple le code **ISO 8859-1 (latin-1)** correspond à la zone Europe occidentale

5.3 Codage Unicode : norme ISO-10646

Ce standard est lié à la norme ISO/CEI 10646 qui décrit une table de caractères équivalente. La dernière version, Unicode 12.0, a été publiée en mars 2019 (et couvre 137 929 caractères).

Cette norme associe à chaque caractère (lettre, nombre, idéogramme, etc.) un nom unique (en anglais et en français) ainsi qu'un numéro (entier positif en base 10 appelé **point de code**).

Cette norme Unicode : ISO-10646 a une capacité maximale de **2^{32} caractères soit plus de 4 milliards** (puisque l'on code sur 4 octets, 32 bits au maximum).

On utilise la notation U+xxx pour désigner les points de code du jeu universel de caractères.
La norme unicode définit plusieurs techniques d'encodage pour représenter les points de code. Ces encodages, appelés *formats de transformation universelle* ou ***Universal Transformation Format*** (**UTF**) portent les noms UTF-n, où n désigne le nombre minimal de bits pour représenter un point de code.

5.4 Norme UTF-8

Le format UTF-8 est le format le plus utilisé (*-8 donc 8 bits ou 1 octet au minimum*). Il permet un encodage **de 1 à 4 octets au maximum**.
Comme son nom l'indique, il faut seulement 8 bits pour coder les premiers caractères. L'UTF-8 est compatible avec le standard ASCII, les 127 premiers caractères sont représentés sur 1 octet, comme en ASCII.
Vous n'aurez normalement pas de problème d'accent si vous commencez l'en tête de vos programmes en python avec :

```
# -*- coding: utf-8 -*-
```

Bilan encodage

- **ASCII** : table sur **7 bits** codant 128 caractères.
- **Norme Unicode : ISO-10646** a une capacité maximale de **2^{32} caractères soit plus de 4 milliards**.
- Le **format UTF-8** est le format d'encodage le plus utilisé. Il permet un encodage **sur 1 à 4 octets au maximum**