PARTIE 2 : représentation des entiers relatifs

1. Première approche intuitive

Une première idée serait de coder le signe + par un 0 et le signe - par un 1.

Ainsi $6_{10} = 110_2$ se coderait alors en 0110_2. Mais -6_{10} se coderait alors en 1110_2.

Exercice 1 : Représenter 14_{10} en binaire. Que constatez-vous ?

Par conséquent, nous allons devoir nous fixer un nombre de bits n comme espace mémoire. On conserve alors le premier pour le signe et les n −1 autres pour le codage de la valeur absolue du nombre.

Ainsi que 8 bits : -6_{10} se code 10000110_2.

Exercice 2 : Déterminer le codage en binaire sur 8 bits puis sur 9 bits, des nombre suivants :

1. 11_{10} =
2. -35_{10} =
3. 26_{10} =
4. -42_{10} =

Exercice 3 : Sur 8 bits, comment codera-t-on 0 ?

Un premier problème se pose avec cette méthode : le chiffre 0_{10} a deux représentations !

Exercice 4 :

1. Coder avec cette méthode les nombres 17_{10} et -17_{10} sur 8 bits

2. Effectuer l'addition binaire de ces deux nombres

L'addition binaire ne donne pas des résultats corrects !
Il faut donc oublier cette méthode !!!

2. Le complément à deux

2.1. Cas des entiers naturels

PROPRIÉTÉ 1 : La représentation d'un nombre entier naturel en binaire se fait de la même manière que dans le cours précédent.

Si on dispose de n bits, le premier sera 0 pour indiquer que l'entier est positif. Et les $n-1$ autres seront le codage en binaire de l'entier.

Cela signifie aussi que si l'on dispose de n bits, nous ne pourrons pas dépasser un entier plus grand que $\mathbf{2^{n-1}-1}$ puisque nous ne disposons que de $\boldsymbol{n-1}$ **bits**.

EXEMPLE 1 : Sur 5 bits, on a $12_{10} = 01100_2$. Si on en dispose que de 4 bits, on ne peut pas coder 12 en binaire si nous considérons les entiers relatifs. En effet, $12 > 2^{4-1} - 1$

2.2. Cas d'entiers négatifs

PREMIÈRE MÉTHODE

1. On code la valeur absolue du nombre en binaire
2. On remplace tous les 0 par des 1 et tous les 1 par des 0
3. On ajoute 1

EXEMPLE 2 : Codage de −12 sur 8 bits en complément à 2.

1. Codage de la valeur absolue : $12_{10} = 0000\ 1100_2$ sur 8 bits
2. Changement des bits : 0000 11002 devient $1111\ 0011_2$
3. On ajoute 1 :

$$\begin{array}{ccccccccc}
 & & & & & & 1 & 1 & \\
 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}$$

Il est possible de vérifier le résultat en additionnant −12 et 12 en binaire :

$$\begin{array}{ccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & & & \\
 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
+ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Nous sommes **sur 8 bits**, le 1 en première position doit être enlevé. **Le résultat est bien 0.**

DEUXIÈME MÉTHODE

Une méthode beaucoup plus rapide permet de faire cette opération :

1. On code la valeur absolue du nombre en binaire
2. On recopie les valeurs des bits de droite à gauche jusqu'au premier 1 inclus.
3. Ensuite on inverse les bits jusqu'au dernier (celui de gauche)

EXEMPLE 3 : Codage de −12 sur 8 bits en complément à 2.

1. Codage de la valeur absolue : $12_{10} = 0000\ 1100_2$ sur 8 bits
2. On recopie les bits de droite à gauche jusqu'au premier 1 inclus : $0000\ 1100_2$ devient $0000\ 1100_2$
3. On inverse les bits restants à gauche : $0000\ 1100_2$ devient $1111\ 0100_2$

Exercice 5 : Déterminer la représentation des valeurs suivantes et faites la vérification avec l'addition binaire :

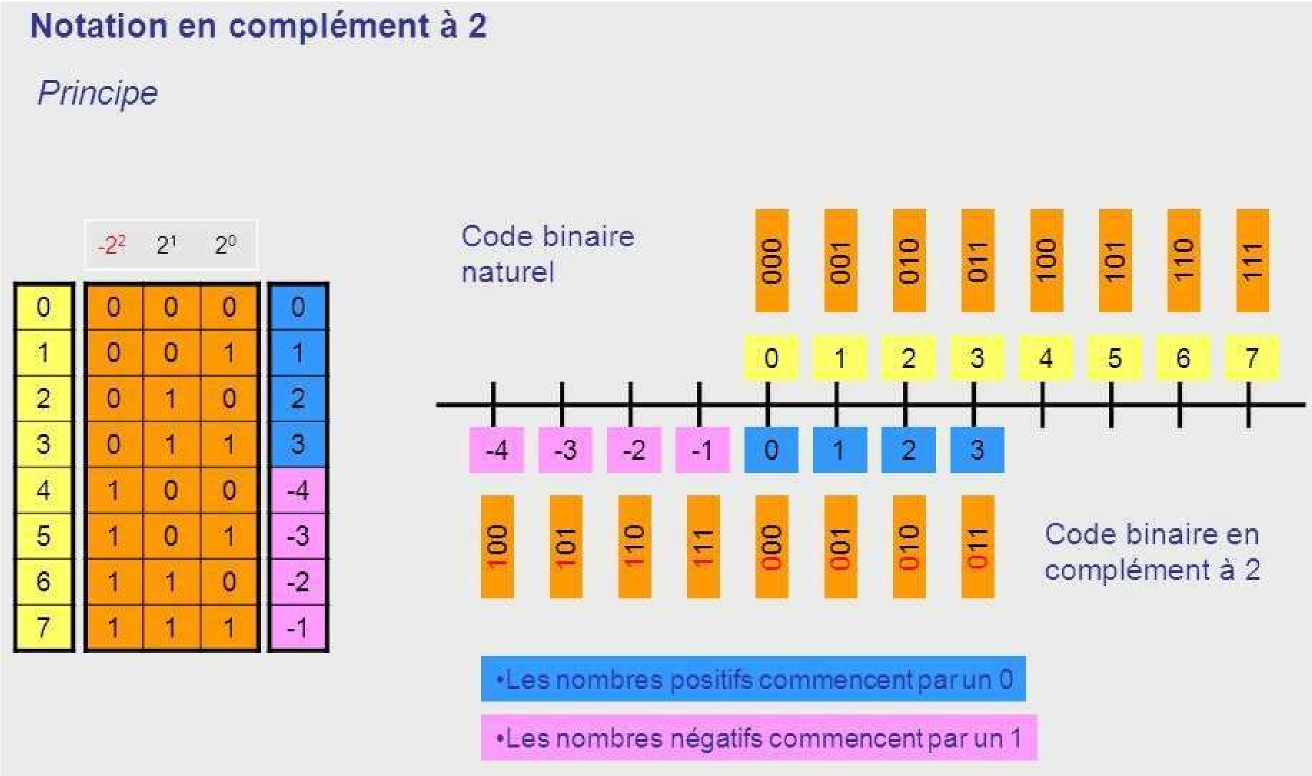
1. −19 codé sur 8 bits en complément à 2

2. −72 codé sur 16 bits en complément à 2

3. −124 codé sur 8 bits en complément à 2

PROPRIÉTÉ 2 : Avec **n** bits, nous pouvons coder :

2^n valeurs comprises entre **-2^{n-1} et $2^{n-1}-1$**



Exercice 6 :

1. Sur 8 bits, donner un encadrement des entiers relatifs codables

2. Même question sur 9 bits ?

PARTIE 3 : Représentation des flottants

1. Les nombres dyadiques

DÉFINITION 1 : Nombres décimaux

Un nombre décimal est un nombre s'écrivant sous la forme $\frac{x}{10^n}$ où x est un entier relatif.

Visuellement un nombre décimal est un nombre acceptant un nombre fini de chiffres après la virgule

EXEMPLE 1 :

Nombre	4	-3	0.25	$\frac{1}{3}$	$\frac{1}{4}$	π	$\frac{10}{3}$
Décimal ?	OUI	OUI	OUI	NON	OUI	NON	NON

DÉFINITION 2 : Nombre dyadiques

Par analogie avec les nombres décimaux, on appelle nombres dyadiques les nombres s'écrivant sous la forme $\frac{x}{2^n}$ avec x entier relatif et n entier naturel.

Exercice 1 : Dire si les nombres suivants sont dyadiques

Nombre	$\frac{13}{4}$	$\frac{1}{5}$	2.5	25	$\frac{10}{3}$	π	3.125
Dyadique ?							

DÉFINITION 3 : Développement dyadique d'un nombre

On appelle développement dyadique d'un nombre l'écriture binaire de ce nombre.

Ce développement dyadique est l'écriture en base 2 d'un nombre.

PROPRIÉTÉ 1 : Écriture binaire d'un nombre dyadique

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

✔ EXEMPLE 2 : Écrire le développement dyadique de 2.5

• 2.5 est un nombre dyadique car $2.5 = \frac{5}{2} = \frac{5}{2^1}$ donc ici x = 5 et n = 1

• On détermine l'écriture binaire de 5 : $5_{10} = 101_2$

• On insère une virgule avant le *n* ième bit en partant de la fin : $2.5_{10} = 10.1_2$

Exercice 2 : Écrire le développement dyadique de 31.25_{10} :

2. De l'écriture décimale à la notation binaire

La méthode précédente est un peu fastidieuse. De plus les nombres non dyadiques ne peuvent pas être codés. Nous allons donc utiliser une nouvelle méthode.

✔ EXEMPLE 3 : Essayons de convertir un nombre : 5.1875 en binaire.

1. Conversion de la partie entière : $5_{10} = 101_2$

2. Multiplications successives par 2 de la partie décimale

$$0.1875\times2 = 0.375 = \underline{0} + 0.375$$

$$0.375\times2 = 0.75 = \underline{0} + 0.75$$

$$0.75\times2 = 1.5 = \underline{1} + 0.5$$

$$0.5\times2 = 1 = \underline{1} + 0.0$$

Dès que la multiplication par 2 donne une partie décimale nulle on s'arrête.

On a donc récupéré un certain nombre de parties entières que l'on va assembler : 0011

Il suffit ensuite de regrouper la partie entière et la partie décimale. Ainsi :

$$5.1875 = 101{,}0011_2$$

Exercice 3 : Conversion de 12.6875 en binaire

1. Conversion de la partie entière : 12_{10} =
2. Multiplications successives par 2 de la partie décimale :

Exercice 4 : Convertir en binaire les nombres suivants :

7.09375

13.325

Que remarquez-vous ?

Exercice 5 : Convertir en binaire le nombre suivant :

4,125

3. De l'écriture binaire à la notation décimale

Il est possible de retrouver une représentation décimale à partir d'une représentation en binaire.

EXEMPLE 4 : Convertir $100{,}0101_2$

1. Convertir la partie entière : $100_2 = 4_{10}$

2. Convertir la partie décimale. On utilise un tableau des puissances de 2 négatives.

Rang du bit	-1	-2	-3	-4	-5	-6
Puissancede2	$2^{-1} = 0.5$	$2^{-2} = 0.25$	$2^{-3} = 0.125$	$2^{-4} = 0.0625$	$2^{-5} = 0.03125$	$2^{-6} = 0.015625$
Bit	0	1	0	1	0	0
Valeur	0	0.25	0	0.0625	0	0

Il suffit ensuite d'additionner toutes les valeurs calculées :

$100{,}0101_2 = 4+0.25+0.0625 = 4.3125_{10}$

Exercice 6 : Trouver la représentation décimale de 101,001

1. Convertir la partie entière :

2. Convertir la partie décimale. On utilise un tableau des puissances de 2 négatives.

Rang du bit	-1	-2	-3	-4	-5	-6
Puissancede2	$2^{-1} = 0.5$	$2^{-2} = 0.25$	$2^{-3} = 0.125$	$2^{-4} = 0.0625$	$2^{-5} = 0.03125$	$2^{-6} = 0.015625$
Bit						
Valeur						

Exercice 7 : Convertir en décimal les nombres suivants :

1101,100101 1000,11101

4. Notation scientifique

En base dix, il est possible d'écrire les très grands nombres et les très petits nombres grâce aux "puissances de dix" (exemples $6{,}02.10^{23}$ ou $6{,}67.10^{-11}$).
Il est possible de faire exactement la même chose avec une représentation binaire, puisque nous sommes en base 2, nous utiliserons des "puissances de deux" à la place des "puissances dix" (exemple $101{,}1101.2^{10}$).

✔ EXEMPLE 5 : Pour passer d'une écriture sans "puissance de deux" à une écriture avec "puissance de deux", il suffit décaler la virgule : $1101{,}1001 = 1{,}1011001.2^{11}$.

Pour passer de 1101,1001 à 1,1011001 nous avons décalé la virgule de 3 rangs vers la gauche d'où le 2^{11}.
(Attention de ne pas oublier que nous travaillons en base 2 le « 11 » correspond bien à un décalage de 3 rangs de la virgule, car $11_2 = 3_{10}$.

✔ EXEMPLE 6 : Si l'on désire décaler la virgule vers la droite, il va être nécessaire d'utiliser des "puissances de deux négatives". $0{,}0110_2 = 1{,}10.2^{-10}$. Nous décalons la virgule de 2 rangs vers la droite, d'où le « -10 ».

✍ Exercice 8 : Trouver l'écriture binaire scientifique de :

1. $1011{,}0111101_2$

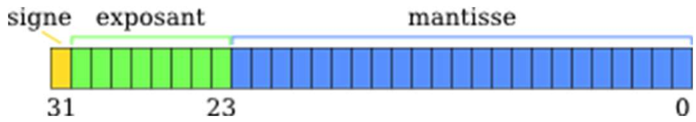
2. 0.0745_{10}

Compléments sur la norme IEEE-754 (hors programme)

L'IEEE 754 est une norme pour la représentation des nombres à virgule flottante en binaire. Elle est la norme la plus employée actuellement pour le calcul des nombres à virgule flottante dans le domaine informatique.

Cette norme définit notamment 2 formats pour représenter des nombres à virgule flottante :

- *Simple précision* (32 bits : 1 bit de signe, 8 bits d'exposant (-126 à 127), 23 bits de mantisse),



- *Double précision* (64 bits : 1 bit de signe, 11 bits d'exposant (-1022 à 1023), 52 bits de mantisse).

✔ EXEMPLE 6 bis : nombre à virgule flottante simple précision : codage du nombre décimal $10{,}8125_{10}$
Pour le bit de signe : le signe - est codé par un 1 et le signe + par un 0.
Dans l'exemple, le bit de signe sera donc 0.
Ensuite, pour utiliser la norme on doit écrire le nombre à représenter sous la forme binaire $1{,}XXXX \times 2^{exp}$, comme nous avons pu le voir précédemment.
$10{,}8125_{10}$ => $1010{,}1101 = 1{,}0101101 \times 2^{11}$
La mantisse est la partie désignée par XXXX. Si elle ne fait pas 23 bits, on la complète par des 0 à droite.
Dans notre exemple, la mantisse est égale à 0101101. On notera donc **0101101 00000000 00000000** sur 23 bits.
L'exposant est codé sur 8 bits. Il peut être positif ou négatif. La norme impose de procéder à un décalage de 127 (en simple précision) de sorte à pouvoir coder sur ces 8 bits tous les exposants entiers de -127 à +128 (soit 256 valeurs différentes). On ajoute donc systématiquement 127 à la valeur de l'exposant avant de la coder en binaire. L'exposant 0 sera donc représenté par le nombre 127 en binaire, l'exposant 128 par 255, l'exposant – 127 sera lui codé comme l'entier 0.
Dans notre exemple, l'exposant est égal à 3, il sera donc représenté par le nombre 130 en binaire, soit **1000 0010**.
Remarque :
Il peut parfois être nécessaire de rajouter des zéros pour avoir les 8 bits exigés. On les ajoutera à gauche cette fois-ci.
En appliquant la norme IEEE 754, le nombre 10,8125 sera donc représenté par les 32 bits suivants :

0100 0001 0010 1101 0000 0000 0000 0000

✍ Exercice 8 bis : Convertir les nombres réels suivants en binaire simple précision en suivant la norme IEEE 754 :

$G = -8{,}125_{10}$ $H = 0{,}375_{10}$ $I = 0{,}1_{10}$

✔ EXEMPLE 6 ter :

On peut également faire le travail inverse. Pour interpréter un nombre codé en virgule flottante, on utilisera la « formule suivante » : $valeur = signe \times mantisse \times 2^{exposant - décalage}$

Par exemple, on peut convertir en décimal le nombre 0011 1110 1000 0000 0000 0000 0000 0000.

Le bit de signe est égal à 0 : signe +

La mantisse est égale à 000 0000 0000 0000 0000 0000, soit 0 si on élimine les zéros « inutiles » à droite.

L'exposant décalé vaut 0111 1101 soit 125. Il faut alors retirer 127 l'exposant est donc égal à - 2.

En base 10, le nombre cherché est donc égal à $1{,}0 \times 2^{-2} = 1 \times 0{,}25 = 0{,}25$

Exercice 8 ter : Convertir le nombre flottant simple précision suivant :

$J = 1\ 1000000110101000000000000000000_2$

$K = 00111101110011001100110011001100_2$

5. Nombres non dyadiques

✔ EXEMPLE 7 : Prenons le nombre 3.3 et appliquons-lui la méthode pour le transformer en binaire.

1. Conversion de la partie entière : 310 = 112
2. Multiplications successives par 2 de la partie décimale

$0.3 \times 2 = 0.6 = 0+0.6$
$0.6 \times 2 = 1.2 = 1+0.2$
$0.2 \times 2 = 0.4 = 0+0.4$
$0.4 \times 2 = 0.8 = 0+0.8$
$0.8 \times 2 = 1.6 = 1+0.6$
$0.6 \times 2 = 1.2 = 1+0.2$
$0.2 \times 2 = 0.4 = 0+0.4$
$\ldots = \ldots = \ldots$

Ce phénomène cyclique nous empêche de tomber sur une multiplication par deux, égale à 0. Le nombre n'est donc pas codable correctement en binaire.

Exercice 9 : Tester sur la console python et expliquer :

```
>>> 3.3 == 3.1 + 0.2
```

Exercice 9 bis : Pour trouver la représentation binaire d'un nombre voulu inférieur à 1 mais positif, vous pouvez appliquer la méthode suivante :

- Bits ← « 0, » // Premier bit avant la virgule
- **nombre** ← **nombre voulu**
- TANT QUE **nombre** > 0
  - **nombre** ← **nombre** * 2
  - Bits ← Bits+partie entière de **nombre** (0 ou 1)
  - SI **nombre** >= 1
    - **nombre** ← **nombre** – 1

1- Donnez le script Python de la fonction decomposition() qui prend pour paramètre un nombre positif inférieur à 1 et qui renvoi la représentation binaire de ce nombre sous la forme d'une chaine de caractère

```
>>> decomposition(0.25)
'0,01'
```

2- Testez votre script pour decomposition(0.1)

La séquence 1100 doit revenir à l'infini.
Pourtant, si vous utilisez le programme fourni, vous pourrez constater qu'il s'arrête !
Pourquoi ? Simplement car l'ordinateur travaille avec des flottants et ne fait donc pas de calculs justes. Et les petites erreurs s'accumulent et finissent par donner un résultat erroné.
Constatez-le en rajoutant un print sur la variable nombre dans la boucle while.

```
>>> decomposition(0.1)
0.2
0.4
0.8
1.6
1.2000000000000002
0.4000000000000036
0.800000000000007
1.600000000000014
1.200000000000028
0.400000000000057
0.800000000000114
1.600000000000227
1.200000000000455
0.4000000000009095
0.800000000001819
1.600000000003638
1.200000000007276
0.400000000014552
0.800000000029104
1.600000000058208
1.200000000116415
0.4000000002328306
0.800000000465661
1.600000000931323
1.200000001862645
0.4000000037252903
0.800000007450581
1.600000014901161
1.200000029802322
0.400000059604645
0.80000011920929
1.60000023841858
1.200000476837158
0.4000009536743164
0.800001907348633
1.600003814697266
1.200007629394531
0.4000152587890625
0.800030517578125
1.60006103515625
1.20001220703125
0.4000244140625
0.800048828125
1.60009765625
1.2001953125
0.400390625
0.80078125
1.6015625
1.203125
0.40625
0.8125
1.625
1.25
0.5
1.0
'0,0001100110011001100110011001100110011001100110011001101'
```