

Activité 1 / Base décimale et base binaire

FreeBSD et Linux sont deux systèmes d'exploitation, au même titre que Windows. Imaginons que leurs mascottes respectives, le démon de FreeBSD, qui compte sur dix doigts, et le manchot de Linux, qui n'a que deux ailes, discutent ensemble. Sur un même smartphone, le démon dit voir « 13 » applications et le manchot « 1101 ».

Pourraient-ils avoir raison tous les deux ?



Naturellement, nous utilisons l'écriture en base 10, appelée **base décimale**. Pratique pour compter sur les 10 doigts de nos mains, la base décimale utilise les caractères {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. À partir de 10 unités, nous utilisons des dizaines, des centaines, puis des milliers... La **position** appelée « rang » de chaque chiffre est essentielle.

Pour décomposer un entier dans la base 10, il faut identifier le rang de chaque chiffre et le multiplier par la puissance de 10 appropriée, puis faire la somme de tous les termes de la décomposition. Par exemple :

$$256_{10} = 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0.$$

Rang	2	1	0
Chiffre	2	5	6

Remarque : Dans l'écriture 256_{10} , l'indice 10 indique que le nombre entier est écrit en base 10.

1. De la même manière, donner la décomposition du nombre entier 13 dans la base 10.

$$13_{10} = 1 \times 10^1 + 3 \times 10^0$$

Pour stocker une information, les ordinateurs utilisent des circuits électroniques à deux états : présence ou absence de courant ou de tension électrique. Par convention, on représente ces deux états par les caractères 1 et 0 qui composent la base 2 ou **base binaire**. La décomposition d'un entier dans la base 2 sera donc effectuée avec des puissances de 2.

2. a. Le manchot de Linux a indiqué qu'il y avait 1101 applications. Donner le rang de chaque chiffre du nombre entier 1101.

Rang	3	2	1	0
Chiffre	1	1	0	1

b. Effectuer la décomposition du nombre 1101 en puissances de 2 afin de vérifier la réponse du manchot en base décimale.

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

3. Que vaut la somme de tous les termes de cette décomposition ? Du démon ou du pingouin, qui a raison ? Conclure.

La somme vaut 13. Le démon et le pingouin ont

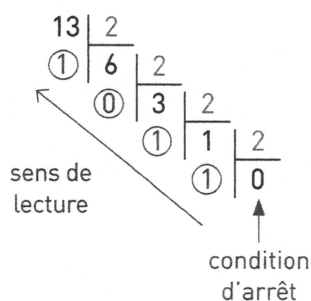
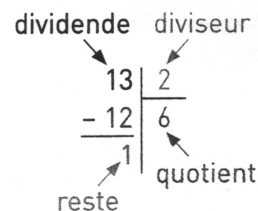
tous les deux raison.

Un même nombre peut donc s'écrire

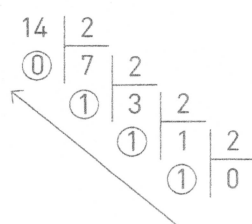
différemment selon la base utilisée.

Pour effectuer la conversion inverse, de la base décimale à la base binaire, une méthode consiste à faire des divisions successives par 2 (puisque l'on veut convertir en base binaire).

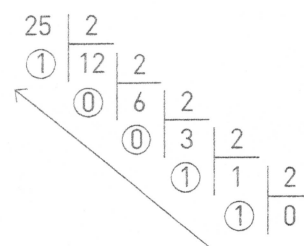
4. À l'aide du rappel sur la division euclidienne ci-contre et de l'exemple ci-dessous, convertir les nombres entiers 14_{10} et 25_{10} de la base décimale vers la base binaire.



$$13_{10} = 1101_2$$



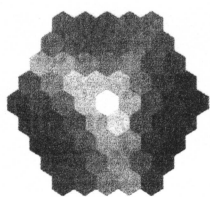
$$14_{10} = 1110_2$$



$$25_{10} = 11001_2$$

Activité 2 / Base hexadécimale

www



Fanny veut modifier la couleur du fond de la page d'accueil de son site web. Elle a l'habitude d'utiliser le format RVB vu en SNT,

où par exemple le bleu est codé `rgb(0, 0, 255)`. Elle a trouvé une couleur qui lui plaît dont la syntaxe est `#F34DC1`. Même si elle peut modifier la couleur avec cette syntaxe, elle se demande quelles sont les valeurs R, V et B correspondantes.

1. Aller sur lycee.editions-bordas.fr/cahier-NSI1re, puis dans « Séquence 1 », sur « Couleurs en HTML ». Cliquer sur des couleurs pour observer les deux syntaxes.

Dans la syntaxe `#F34DC1`, chaque paire de caractères hexadécimaux correspond

F 3 4 D C 1
R V B

à une couleur : rouge, vert et bleu. Pour écrire un nombre entier dans cette base, il faut pouvoir écrire les nombres 10, 11, 12, 13, 14 et 15 avec un seul caractère car les chiffres de 0 à 9 ne suffisent pas ; les six premières lettres de l'alphabet sont donc utilisées ainsi :

Base 10	10	11	12	13	14	15
Base 16	A	B	C	D	E	F

Conversion de la base 16 vers la base 10

Fanny veut calculer les valeurs x , y et z de la syntaxe `rgb(x, y, z)` qui correspond à `#F34DC1`. Pour convertir une valeur de la base hexadécimale à la base décimale, on utilise la **décomposition en base 16** et le tableau de correspondance ci-dessus.

Exemple : $F_{16} = 15 \times 16^1 + 3 \times 16^0 = 243_{10}$.

2. Convertir les deux autres composantes en base décimale, puis compléter la syntaxe `rgb(x, y, z)`.

$$4D_{16} = \dots 4 \dots \times 16^1 + \dots 13 \dots \times 16^0 = \dots 77_{10} \dots$$

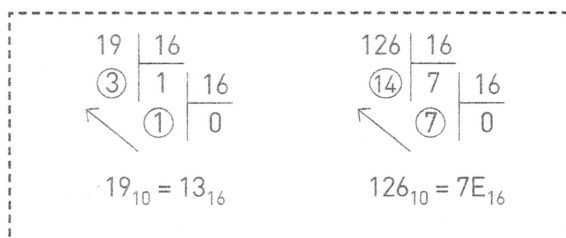
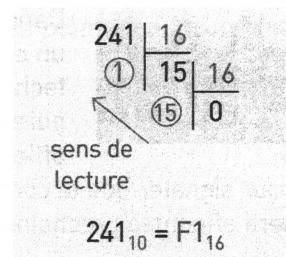
$$C1_{16} = \dots 12 \dots \times 16^1 + \dots 1 \dots \times 16^0 = \dots 193_{10} \dots$$

Ainsi, la syntaxe est `rgb(243, 77, 193)`.

Conversion de la base 10 vers la base 16

Fanny a trouvé une autre couleur codée `rgb(241, 19, 126)` et souhaite l'écrire avec l'autre syntaxe. Pour passer de la base décimale en base hexadécimale, il faut effectuer des **divisions successives** par 16.

3. En observant le changement de base de 241_{10} , convertir les deux autres composantes, puis exprimer la couleur `rgb(241, 19, 126)` avec l'autre syntaxe.

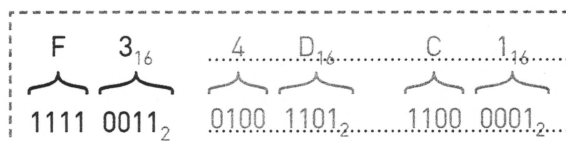


Ainsi, la syntaxe est `#F1137E`.

Conversion de la base 16 vers la base 2

En machine, les données sont stockées en base binaire. Pour convertir une valeur de la base hexadécimale à la base binaire, il suffit de **remplacer chaque caractère** (chiffre ou lettre) par le code binaire associé.

4. En observant l'exemple pour $F3_{16}$, convertir les deux autres composantes de `#F34DC1`.



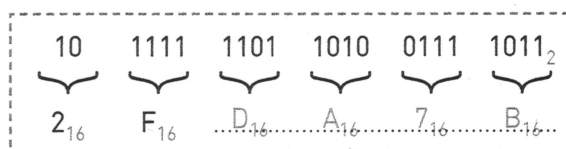
$$F3_{16} = 1111\ 0011_2$$

$$4D_{16} = \dots 0100\ 1101_2 \dots \quad C1_{16} = \dots 1100\ 0001_2 \dots$$

Conversion de la base 2 vers la base 16

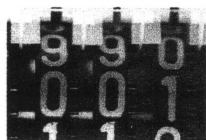
Pour convertir un entier positif de la base binaire à la base hexadécimale, il suffit de parcourir la valeur binaire **de la droite vers la gauche** en regroupant les bits par paquets de 4. Ainsi, chaque paquet de 4 bits est la représentation binaire d'un caractère (chiffre ou lettre) en base 16.

5. En observant l'exemple, convertir $10\ 1111\ 1101\ 1010\ 0111\ 1011_2$ en base 16, puis donner la syntaxe associée.



La syntaxe est donc `#2FDA7B`.

Activité 3 / Integer Overflow



Dans une usine, sur une machine qui fabrique en moyenne 200 pièces par jour, un compteur stocké sur un octet affiche le nombre de pièces produites. Quand un technicien récupère les pièces fabriquées, il vérifie que le nombre est conforme, puis appuie sur un bouton pour remettre le compteur à zéro. Un jour, le compteur affiche 21 alors que 277 pièces ont été produites. Il appelle le numéro d'assistance pour signaler que le compteur ne fonctionne plus. On lui répond que c'est normal et qu'une mise à jour sera effectuée prochainement pour corriger ce défaut. Essayons d'expliquer ce qui est arrivé !

1. Vérifier que l'écriture binaire de 277_{10} est $1\ 0001\ 0101_2$.

$$1\ 0001\ 0101_2 = 1 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 277_{10}$$

2. Sachant qu'un octet est composé de 8 bits, compléter un octet avec le nombre binaire obtenu. Que remarquez-vous ?

1 0 0 0 1 0 1 0 1 Un octet ne suffit pas pour stocker 277_{10} : il faudrait 9 bits.

Un dépassement d'entier (*integer overflow*) se produit lorsqu'une opération mathématique conduit à une valeur numérique supérieure à celle représentable dans l'espace de stockage disponible.

3. Pour stocker $1\ 0001\ 0101_2$ sur un octet, seuls les 8 derniers bits sont conservés. Quel sera alors l'affichage du compteur ? Comment pourrait-on améliorer ce compteur ?

La valeur stockée serait alors $0001\ 0101_2$, qui vaut 21_{10} , au lieu de 277_{10} . La solution serait de coder le nombre de pièces sur 9 bits au lieu de 8. L'overflow a provoqué une erreur d'affichage.

Combien de bits faut-il pour représenter un nombre entier en binaire ?

Pour évaluer le nombre de bits minimum nécessaires à l'écriture en base 2 d'un entier positif, il faut trouver la plus petite puissance de 2 qui soit strictement supérieure à l'entier à écrire.

Exemple : comme $2^8 = 256_{10}$ et $2^9 = 512_{10}$, $2^8 \leq 277_{10} < 2^9$ donc 9 bits sont nécessaires pour écrire 277_{10} .

4. Combien de bits sont nécessaires pour stocker les entiers $1\ 047_{10}$ et $65\ 512_{10}$?

$$2^{10} = 1\ 024 \text{ et } 2^{11} = 2\ 048 \text{ donc } 2^{10} \leq 1\ 047_{10} < 2^{11} \text{ donc 11 bits sont nécessaires pour écrire } 1\ 047_{10}$$

$$2^{15} = 32\ 768 \text{ et } 2^{16} = 65\ 536_{10} \text{ donc } 2^{15} \leq 65\ 512_{10} < 2^{16} \text{ donc il faut 16 bits pour écrire } 65\ 512_{10}$$

5. Remplir le tableau suivant avec les tailles fréquentes des entiers en binaire et la généralisation.

Nombre d'octets utilisés	1	2	4	8	n
Nombre de bits utilisés	8	16	32	64	$8n$
Nombre d'entiers positifs que l'on peut stocker	2^8	2^{16}	2^{32}	2^{64}	2^{8n}
Plus grand entier positif que l'on peut stocker	$2^8 - 1$	$2^{16} - 1$	$2^{32} - 1$	$2^{64} - 1$	$2^{8n} - 1$

6. Quelle est la valeur maximale représentable sur le compteur de la machine ?

La valeur maximale représentable est $1111\ 1111_2$, soit 255_{10} , soit $2^8 - 1$.

Combien de bits faut-il pour effectuer la somme de deux entiers positifs ?

7. En utilisant les deux plus grands nombres entiers représentables sur 8 bits, déterminer le nombre de bits nécessaires pour représenter leur somme.

D'après le tableau ci-dessus, le plus grand entier positif représentable sur 8 bits a pour valeur

$$2^8 - 1 = 255. \text{ La somme } 255 + 255 \text{ vaut } 510, \text{ qui est représentable sur 9 bits. Un seul bit supplémentaire}$$

a été nécessaire pour représenter la somme de deux entiers de 8 bits.

Activité 4 / Découverte du complément à 2 puissance n

Somme de deux entiers positifs

1. En vous aidant de l'exemple d'addition binaire et du tableau, calculer la somme des entiers positifs en base binaire 1001_2 et 0101_2 puis des nombres 0011_2 et 1001_2 .

Somme binaire
$0_2 + 0_2 = 0_2$
$0_2 + 1_2 = 1_2$
$1_2 + 1_2 = 0_2$ on pose 0 et on retient 1

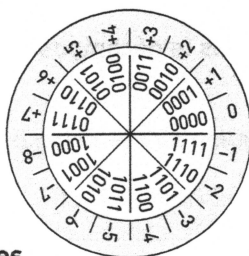
Exemple :

$$\begin{array}{r} \textcircled{1} \\ 0011 \text{ (3}_{10}\text{)} \\ + 0101 \text{ (2}_{10}\text{)} \\ \hline 0101 \text{ (5}_{10}\text{)} \end{array}$$

$\begin{array}{r} \textcircled{1} \\ 1001 \text{ (9}_{10}\text{)} \\ + 0101 \text{ (5}_{10}\text{)} \\ \hline 1110 \text{ (14}_{10}\text{)} \end{array}$	$\begin{array}{r} \textcircled{1}\textcircled{1} \\ 0011 \text{ (3}_{10}\text{)} \\ + 1001 \text{ (9}_{10}\text{)} \\ \hline 1100 \text{ (12}_{10}\text{)} \end{array}$
--	---

Représentation d'un entier relatif, positif ou négatif

Voici ci-contre une proposition de codage des entiers négatifs sur 4 bits.



2. Avec cette méthode de codage, les résultats des deuxième et troisième additions sont-ils toujours corrects ?

$\begin{array}{r} \textcircled{1} \\ 1001 \text{ (-7}_{10}\text{)} \\ + 0101 \text{ (5}_{10}\text{)} \\ \hline 1110 \text{ (-2}_{10}\text{)} \end{array}$	$\begin{array}{r} \textcircled{1}\textcircled{1} \\ 0011 \text{ (3}_{10}\text{)} \\ + 1001 \text{ (-7}_{10}\text{)} \\ \hline 1100 \text{ (-4}_{10}\text{)} \end{array}$
---	--

Avec ce codage, les résultats sont toujours corrects.

Il existe une méthode générale pour coder en binaire un nombre entier relatif :

Méthode de calcul du « complément à 2^n » ou « complément à 2 »

- Pour représenter un nombre positif, on utilise sa représentation en binaire, précédée d'autant de 0 que nécessaire pour avoir n bits au total.
- Pour représenter un nombre négatif $-m$:
 - coder en binaire le nombre m comme précédemment ;
 - inverser tous ses bits (ce qu'on appelle prendre le « complément à 1 ») ;
 - additionner 1 au nombre binaire obtenu, sans oublier les retenues éventuelles.

Exemple de calcul du complément à 2 sur 4 bits de -6_{10} :

Nombre 6_{10} en binaire	0110
Complément à 1	$\textcircled{1}$ 1001
Addition de 1	+ 0001
Complément à 2^4	1010 (-6_{10})

3. Après avoir étudié l'exemple du calcul du complément à 2 sur 4 bits de l'entier -6_{10} , compléter les calculs du complément à 2 sur 8 bits de l'entier -68_{10} .

Nombre 68_{10} en binaire	0100 0100
Complément à 1	$\textcircled{1}\textcircled{1}$ 1011 1011
Addition de 1	+ 0000 0001
Complément à 2^8	1011 1100 (-68_{10})

4. En posant l'addition à la main, effectuer la somme $7 + (-7)$ en binaire.

$\begin{array}{r} \textcircled{1}\textcircled{1}\textcircled{1} \\ 0111 \text{ (7}_{10}\text{)} \\ + 1001 \text{ (-7}_{10}\text{)} \\ \hline 10000 \end{array}$

5. En considérant qu'il y a un *overflow* sur le bit le plus à gauche, au rang 4, donner la valeur obtenue. Ce résultat était-il prévisible ?

La valeur $1\,0000_2$ devient 0000_2 si l'on « oublie » le premier bit, grâce à un *overflow* (dépassement de capacité). On trouve donc bien que la somme $7 - 7$ vaut zéro !

6. À votre avis, quel est le complément à 2 d'un nombre obtenu par complément à 2 ? Montrer que cette propriété est vraie avec les entiers relatifs 5 et (-5) .

Le complément à 2 d'un nombre obtenu par complément à 2 doit certainement être le nombre de départ. 5 est codé par 0101. (-5) est codé par 1011. Le complément à 2 de 1011 est 0101 : on retombe bien sur le nombre 5 du départ.