Représentation des entiers naturels

NSI – Première

Écritures binaire et hexadécimale, changements de base

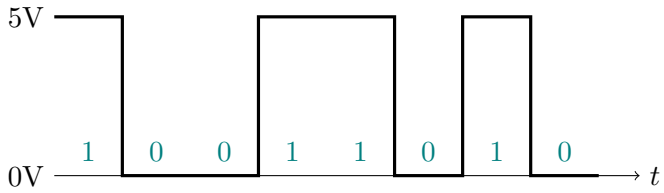
Table des matières

Introduction

Vus de l'extérieur, les ordinateurs que nous utilisons tous les jours permettent de mémoriser, transmettre et transformer des nombres, des textes, des images, des sons, etc. Pour effectuer toutes ces manipulations, ceux-ci utilisent des courants électriques, des aimants, des rayons de lumière, etc., qui peuvent être représentés par deux états possibles :

— tension nulle ou tension non nulle (5V par exemple) pour les courants électriques ;
— aimantation dans un sens ou dans l'autre sens pour les aimants ;
— lumière ou pas de lumière pour les rayons, etc.

D'un point de vue pratique, les informaticiens ont décidé de représenter et de traduire ces deux états à l'aide des deux chiffres 0 et 1. Ainsi, par exemple, le chiffre 1 peut traduire la présence d'un courant sur un fil électrique et le chiffre 0 l'absence de courant. Un signal numérique ressemble alors à l'illustration suivante :



De même, pour traduire la présence ou l'absence de lumière dans un rayon lumineux (par exemple à la sortie d'un laser), on peut convenir que le chiffre 1 équivaut à la lumière et que le chiffre 0 équivaut à l'absence de lumière.

Quelques repères historiques (source : Wikipédia)

— **1650 av. J.-C. :** multiplication égyptienne, décomposition d'un des nombres (généralement le plus petit) en une somme et création d'une table de puissances pour l'autre nombre. Très souvent, cette décomposition s'effectuait suivant les puissances de deux.
— **750 av. J.-C. :** traité du Yi Jing dont les hexagrammes chinois seront plus tard reconnus comme la première expression d'une numération binaire.
— **300 av. J.-C. :** le mathématicien indien Pingala est crédité d'une table représentant 0 à 7 en numération binaire, dans son traité *Chandaḥ-śāstra*.

— **1600 :** table de Thomas Harriot (1560–1621), première expression du binaire connue en France.
— **1617 :** Neper, dans son traité *Rhabdologie*, montre comment effectuer simplement les opérations sur des nombres binaires.
— **1670 :** Juan Caramuel y Lobkowitz fait la première étude raisonnée sur les numérations non décimales.
— **1703 :** Leibniz publie son exposé sur le système binaire devant l'Académie des sciences de Paris dans les *Mémoires*.
— **1847 :** George Boole publie les premiers travaux de son algèbre de Boole.

Dans cette partie, nous allons voir plus précisément comment sont représentés les nombres et les caractères, c'est-à-dire les éléments de base de toutes les données.

1 Représentation binaire des entiers naturels

1.1 Écriture en décimale (base 10)

Nous utilisons tous les jours, dans notre système de calcul quotidien, dix symboles pour noter les quantités. Ces dix symboles, appelés *chiffres*, sont en fait issus d'une écriture arabe, utilisée par le monde entier aujourd'hui. Ils permettent d'écrire de façon **unique** tous les nombres entiers naturels. Ces dix chiffres sont : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

Par exemple, l'écriture du nombre $\mathbf{n = 329}$ se traduit par :

$$n = \mathbf{3} \times 100 + \mathbf{2} \times 10 + \mathbf{9} \times 1 = \mathbf{3} \times 10^2 + \mathbf{2} \times 10^1 + \mathbf{9} \times 10^0$$

On dit alors que 329 est la représentation décimale, ou représentation en **base 10**, du nombre n.

Plus généralement, le nombre $\mathbf{n = a_m \dots a_1 a_0}$ en **base 10** se traduit par l'égalité :

$$n = \mathbf{a_m} \times 10^m + \ldots + \mathbf{a_1} \times 10^1 + \mathbf{a_0} \times 10^0$$

Remarque : Chiffres et puissances décroissantes

Il faut noter que les $\mathbf{a_i} \in [\![\mathbf{0}\,;\,\mathbf{9}]\!]$ et multiplient des puissances décroissantes de **10**.

Exercice 1.1: Écriture en puissances de 10

Écrire la traduction en puissances de **10** (base 10) des nombres **2 134** et **805**.

Correction

$2\,134 = \mathbf{2} \times 10^3 + \mathbf{1} \times 10^2 + \mathbf{3} \times 10^1 + \mathbf{4} \times 10^0$ et $805 = \mathbf{8} \times 10^2 + \mathbf{0} \times 10^1 + \mathbf{5} \times 10^0$

Exercice 1.2: Retour à l'écriture décimale

Écrire en décimal les deux nombres décomposés en base 10 suivants :

$$A = \mathbf{6} \times \mathbf{10^5} + \mathbf{7} \times \mathbf{10^3} + \mathbf{9} \times \mathbf{10^1} \quad \text{et} \quad B = \mathbf{4} \times \mathbf{10^3} + \mathbf{3} \times \mathbf{10^1} + \mathbf{8} \times \mathbf{10^0}$$

Correction

$A = 600\,000 + 7\,000 + 90 = \mathbf{607\,090}$ et $B = 4\,000 + 30 + 8 = \mathbf{4\,038}$

Remarque : D'autres écritures que la décimale

Les écritures décimales des nombres sont les plus utilisées, mais ce ne sont pas les seules, et il en reste quelques-unes encore utilisées aujourd'hui dans la vie courante :
- écriture sexagésimale : minutes, secondes (base 60) ;
- écriture en base 12 : heures ;
- écriture romaine, babylonienne (décimale et/ou sexagésimale mélangée) ;
- écriture en fractions continues ;
- écriture en nombre fractionnaire (décimale et fractionnaire mélangée).

On peut écrire un nombre dans n'importe quelle base $\mathbf{b}$, où b est un entier naturel supérieur à 2. On va s'intéresser à la base $\mathbf{2}$ en particulier.

1.2 Écriture en binaire (base 2)

Dans un ordinateur, la représentation des entiers naturels se fait à l'aide de 0 et de 1 : on parle alors de représentation **binaire** (donc utilisant seulement **deux** symboles pour exprimer tous les nombres). Cette représentation est analogue à la représentation décimale que l'on utilise tous les jours ; elle est juste un peu moins naturelle, car on ne l'utilise pas au quotidien. Mais elle s'apprend comme l'écriture romaine. En décimal, on décompose suivant des puissances de 10 ; en binaire, on décompose en puissances de 2. On verra que dans chaque base, la décomposition est **unique**.

Exemple 1.3: Décomposition de 21 en base 10 et en base 2

En décimal (base 10) : $21 = 20 + 1 = 2 \times \mathbf{10^1} + 1 \times \mathbf{10^0}$
En binaire (base 2) : $21 = 16 + 4 + 1 = 1 \times \mathbf{2^4} + 1 \times \mathbf{2^2} + 1 \times \mathbf{2^0}$

Définition 1.4: Bit et mot binaire

Chacun des chiffres 0 et 1 est appelé chiffre binaire ou **bit** (de **BI**nary digi**T** en anglais). Ainsi, toutes les données présentes dans l'ordinateur vont être représentées par une succession de 0 et de 1, que l'on appelle *mot binaire*.

Remarque : Juxtaposition des chiffres

L'écriture décimale **juxtapose** des chiffres de 0 à 9, chacun associé à des multiples de 10.
L'écriture binaire **juxtapose** des 0 et des 1, chacun associé à des multiples de 2.
En reprenant l'exemple précédent, le mot binaire associé à **21** est $(10101)_2$ car :

$$21 = 1 \times \mathbf{2^4} + 0 \times \mathbf{2^3} + 1 \times \mathbf{2^2} + 0 \times \mathbf{2^1} + 1 \times \mathbf{2^0}$$

Notation 1.5: Écriture d'un nombre dans une base

Pour préciser la base, on écrit le nombre entre parenthèses et la base en indice : $(\dots)_b$.

Concernant la représentation binaire, ou représentation en base 2, on a le résultat suivant :

Théorème 1.6: Représentation binaire

Soit $\mathbf{n}$ un entier naturel. Alors $\mathbf{n = 0}$, ou $\mathbf{n}$ s'écrit de façon unique sous la forme

$$n = 2^m + \mathbf{b_{m-1}} \times 2^{m-1} + \dots + \mathbf{b_1} \times 2^1 + \mathbf{b_0} \times 2^0$$

avec $m \in \mathbb{N}$ et $\mathbf{b_{m-1}}, \dots, \mathbf{b_1}, \mathbf{b_0} \in \{\mathbf{0}\,;\,\mathbf{1}\}$.

On pose alors la définition suivante :

Définition 1.7: Représentation binaire

Soit n un entier naturel.
- Si $\mathbf{n = 0}$, alors la représentation binaire de $\mathbf{n}$ est $(0)_2$.
- Si $\mathbf{n \geq 1}$, alors la représentation binaire de $\mathbf{n}$ est $(\mathbf{1b_{m-1} \dots b_1 b_0})_2$.

Exemple 1.8: Passer d'une écriture à l'autre

- Le nombre $18 = 1 \times \mathbf{2^4} + 0 \times \mathbf{2^3} + 0 \times \mathbf{2^2} + 1 \times \mathbf{2^1} + 0 \times \mathbf{2^0}$ est représenté par le nombre binaire $(1\ 0010)_2$.
- Le nombre binaire $(1101)_2$ représente le nombre entier $1 \times \mathbf{2^3} + 1 \times \mathbf{2^2} + 0 \times \mathbf{2^1} + 1 \times \mathbf{2^0} = \mathbf{13}$.

Remarque : Lecture par groupes de 4 bits

Pour faciliter la lecture des nombres, on groupe par 4 les bits à partir de la fin (en décimal, on groupe par 3).

Exercice 1.9: Du binaire vers le décimal

Traduire en base 10 les nombres binaires $(1010)_2$ et $(1\ 1001)_2$.

Correction

$$(1010)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 0 + 2 + 0 = \mathbf{10}$$

$$(1\ 1001)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 8 + 0 + 0 + 1 = \mathbf{25}$$

Exercice 1.10: Du décimal vers le binaire

Déterminer la représentation binaire des nombres 15 et 158.

Correction

$15 = 8 + 4 + 2 + 1 = 2^3 + 2^2 + 2^1 + 2^0$, donc $15 = (1111)_2$.
$158 = 128 + 16 + 8 + 4 + 2 = 2^7 + 2^4 + 2^3 + 2^2 + 2^1$, donc $158 = (1001\ 1110)_2$.

Remarque : Méthode des puissances de 2

Pour déterminer la représentation binaire d'un nombre entier $\mathbf{n}$, on commence généralement par chercher la plus grande puissance de 2 inférieure à $\mathbf{n}$. On utilise pour cela le tableau suivant :

$\star\ 2^0 = 1$	$\star\ 2^4 = 16$	$\star\ 2^8 = 256$	$\star\ 2^{12} = 4\,096$	$\star\ 2^{16} = 65\,536$
$\star\ 2^1 = 2$	$\star\ 2^5 = 32$	$\star\ 2^9 = 512$	$\star\ 2^{13} = 8\,192$	$\star\ 2^{17} = 131\,072$
$\star\ 2^2 = 4$	$\star\ 2^6 = 64$	$\star\ 2^{10} = 1\,024$	$\star\ 2^{14} = 16\,384$	$\star\ 2^{18} = 262\,144$
$\star\ 2^3 = 8$	$\star\ 2^7 = 128$	$\star\ 2^{11} = 2\,048$	$\star\ 2^{15} = 32\,768$	$\star\ \dots$

Ainsi, pour déterminer par exemple la représentation binaire de 66 601, on procède comme suit :

$$66\,601 = \dots$$

Exercice 1.11: Représentation binaire de grands nombres

Déterminer la représentation binaire de 146 023 et 75 086.

Correction

$146\,023 = 131\,072 + 8\,192 + 4\,096 + 2\,048 + 512 + 64 + 32 + 4 + 2 + 1 = 2^{17} + 2^{13} + 2^{12} + 2^{11} + 2^9 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0$,
donc $146\,023 = (10\ 0011\ 1010\ 0110\ 0111)_2$.
$75\,086 = 65\,536 + 8\,192 + 1\,024 + 256 + 64 + 8 + 4 + 2 = 2^{16} + 2^{13} + 2^{10} + 2^8 + 2^6 + 2^3 + 2^2 + 2^1$,
donc $75\,086 = (1\ 0010\ 0101\ 0100\ 1110)_2$.

Remarque : Mots de taille fixée, dépassement de capacité

Dans la pratique, les longueurs des mots binaires permettant de représenter des données sont fixées à l'avance dans la machine. Il peut donc y avoir des différences entre la représentation binaire du nombre et sa représentation en machine. Par exemple, si on veut représenter 5 avec un mot de 4 bits, on écrira $(0101)_2$ et non $(101)_2$. Ainsi, lorsque le nombre a une représentation binaire de longueur inférieure à celle du mot, il suffit de compléter par des 0 les valeurs à gauche.
En revanche, si le nombre a une représentation binaire de longueur supérieure à celle du mot, il ne peut pas être représenté et on parle alors d'*overflow*, c'est-à-dire de dépassement de capacité, ce qui conduit à une erreur et à un arrêt du programme.
En pratique, Python (et quelques autres langages de programmation) fait en sorte que de tels problèmes n'apparaissent pas pour les entiers naturels, en les écrivant directement dans la mémoire RAM de l'ordinateur. Il n'y a d'*overflow* que si l'on dépasse la capacité de la RAM.

Exercice 1.12: Mots de 8 et 16 bits

1. Peut-on représenter le nombre 250 avec un mot de 8 bits ? Si oui, donner sa représentation.
2. Peut-on représenter le nombre 65 536 avec un mot de 16 bits ? Si oui, donner sa représentation.

Correction

1. Avec 8 bits, on peut représenter tous les entiers de 0 à $2^8 - 1 = 255$. Puisque $250 \leqslant 255$, c'est possible : $250 = 128 + 64 + 32 + 16 + 8 + 2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$, donc $250 = (1111\ 1010)_2$.
2. Avec 16 bits, on peut représenter tous les entiers de 0 à $2^{16} - 1 = 65\,535$. Puisque $65\,536 > 65\,535$, ce n'est **pas** possible : il s'agit d'un dépassement de capacité (il faudrait un 17^{e} bit).

Exercice 1.13: Nombres représentables avec m bits

1. Justifier qu'avec un mot de m bits, on peut représenter tous les nombres entiers de 0 à $\mathbf{2^m - 1}$.
2. En déduire tous les nombres entiers représentables avec un ordinateur possédant une architecture **64 bits**.

Correction

1. Un mot de m bits est une suite de m chiffres, chacun valant 0 ou 1 : il existe donc exactement 2^m mots binaires distincts possibles. Le plus petit $(00\ldots0)$ représente 0, et le plus grand $(11\ldots1)$ représente $2^{m-1} + 2^{m-2} + \ldots + 2^0 = 2^m - 1$. Comme la représentation binaire de chaque entier naturel est **unique** (théorème vu en cours), ces 2^m mots correspondent exactement, un par un, aux 2^m entiers de 0 à $2^m - 1$.
2. Avec $m = 64$, on peut donc représenter tous les entiers de 0 à $2^{64} - 1 = \mathbf{18\,446\,744\,073\,709\,551\,615}$.

Définition 1.14: Poids faible, poids fort

Soit $(\mathbf{a_m} \dots \mathbf{a_1 a_0})_2$ un mot écrit sur $\mathbf{m}$ bits. Alors $\mathbf{a_0}$ est appelé bit de **poids faible** et $\mathbf{a_{m-1}}$ est appelé bit de **poids fort**.

Définition 1.15: Octet

Un mot de huit bits est appelé **octet**.

Exercice 1.16: Un octet

Écrire le nombre 117 sur un octet. Préciser le bit de poids faible et le bit de poids fort.

Correction

$117 = 64 + 32 + 16 + 4 + 1 = 2^6 + 2^5 + 2^4 + 2^2 + 2^0$, donc sur un octet, $117 = (0111\ 0101)_2$. Le bit de poids faible (le plus à droite) vaut **1**, et le bit de poids fort (le plus à gauche, sur les 8 bits) vaut **0**.

Exercice 1.17: Combien d'octets ?

Combien d'octets sont nécessaires pour écrire le nombre 123 456 ?

Correction

On a $2^{16} = 65\,536 \leqslant 123\,456 < 131\,072 = 2^{17}$: il faut donc **17 bits** pour écrire 123 456 en binaire. Comme on ne peut coder que sur un nombre entier d'octets, et que 2 octets (16 bits) ne suffisent pas, il faut **3 octets** (soit 24 bits, dont 7 bits inutilisés).

Le tableau ci-dessous donne quelques multiples et unités employés en informatique :

Préfixes décimaux		
Unités	**Valeurs (SI)**	**Exemples d'utilisation**
1 kilo-octet (Ko)	10^3 octets	Fichier texte
1 méga-octet (Mo)	10^6 octets	Photo numérique
1 giga-octet (Go)	10^9 octets	Mémoire RAM
1 téra-octet (To)	10^{12} octets	Disque dur

Remarque : Le culot de la démesure

Edward Kasner, mathématicien américain, est connu pour avoir popularisé le terme *gogol* (*googol* en anglais), désignant un chiffre 1 suivi de 100 zéros (10^{100}). Ce mot est repris plus tard par les fondateurs de Google pour nommer leur entreprise.
Un disque dur de **3 To** peut être vu comme un ensemble de $3 \times 1\,000\,000\,000\,000 \times 8$ cases mémoire (soit 24×10^{12}), qui sont toutes référencées par une adresse définie comme un *mot binaire*. Une telle adresse est donc compliquée à écrire, car de taille encombrante. Les informaticiens ont donc cherché une écriture plus synthétique et facilement utilisable (notamment au niveau de sa conversion en binaire et vice-versa, car un ordinateur ne comprend de toute façon que le binaire) pour contourner cette difficulté. La numération **hexadécimale** est née.

2 Représentation hexadécimale des entiers naturels

Comme pour la représentation binaire, on a le résultat suivant :

Théorème 2.1: Représentation hexadécimale

Soit $\mathbf{n}$ un entier naturel. Alors $\mathbf{n = 0}$, ou $\mathbf{n}$ s'écrit de façon unique sous la forme

$$n = \mathbf{b_m} \times 16^m + \mathbf{b_{m-1}} \times 16^{m-1} + \ldots + \mathbf{b_1} \times 16^1 + \mathbf{b_0} \times 16^0$$

avec $m \in \mathbb{N}$, $\mathbf{b_m}, \mathbf{b_{m-1}}, \ldots, \mathbf{b_1}, \mathbf{b_0} \in [\![\mathbf{0}\,;\,\mathbf{15}]\!]$ et $\mathbf{b_m \neq 0}$.

On pose alors la définition suivante :

Définition 2.2: Représentation hexadécimale

Soit n un entier naturel.
- Si $\mathbf{n = 0}$, alors la représentation hexadécimale de $\mathbf{n}$ est $(0)_{16}$.
- Si $\mathbf{n \geq 1}$, alors la représentation hexadécimale de $\mathbf{n}$ est $(\mathbf{b_m b_{m-1}} \ldots \mathbf{b_1 b_0})_{16}$.

Cette représentation est également appelée représentation en **base 16**.

Pour pouvoir effectuer la représentation **hexa**décimale d'un nombre entier, il est donc nécessaire d'utiliser **seize** symboles. Ceux-ci sont **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F**, avec les conventions suivantes :

Valeur en hexadécimal	A	B	C	D	E	F
Valeur en décimal	10	11	12	13	14	15

Remarque : Pourquoi des lettres ?

Le choix de lettres pour remplacer les valeurs de 10 à 15 permet d'éviter des confusions. En effet, si on considère le nombre $(10)_{16}$ en utilisant directement les symboles 10, 11, …, celui-ci correspond-il à $10 \times 16^0 = 10$ ou à $1 \times 16^1 + 0 \times 16^0 = 16$?

Exemple 2.3: Lire et écrire en hexadécimal

- Le nombre $\mathbf{10} \times 16^1 + \mathbf{5} \times 16^0$ est représenté par le nombre hexadécimal $(\mathbf{A5})_{16}$.
- Le nombre hexadécimal $(\mathbf{23A})_{16}$ représente le nombre entier $\mathbf{2} \times 16^2 + \mathbf{3} \times 16^1 + \mathbf{10} \times 16^0 = 512 + 48 + 10 = 570$.

Exercice 2.4: De l'hexadécimal au décimal

Traduire en base 10 les nombres hexadécimaux $(\mathbf{B8C})_{16}$ et $(\mathbf{FF})_{16}$.

Correction

$$(B8C)_{16} = 11 \times 16^2 + 8 \times 16^1 + 12 \times 16^0 = 2\,816 + 128 + 12 = \mathbf{2\,956}$$

$$(FF)_{16} = 15 \times 16^1 + 15 \times 16^0 = 240 + 15 = \mathbf{255}$$

Exercice 2.5: Du décimal à l'hexadécimal

Déterminer la représentation hexadécimale des nombres 245 et 1 026.

Correction

$245 = 16 \times 15 + 5$, donc le premier reste est 5 et le quotient 15 = F : $245 = (\text{F5})_{16}$.
$1\,026 = 16 \times 64 + 2$, puis $64 = 16 \times 4 + 0$, puis $4 = 16 \times 0 + 4$. En recopiant les restes du dernier au premier : $1\,026 = (402)_{16}$.

Exercice 2.6: Économie de place

Combien de symboles hexadécimaux sont nécessaires pour écrire un octet ? Justifier la remarque précédente sur le fait que la numération hexadécimale est moins coûteuse en place que la numération binaire.

Correction

Un octet, c'est 8 bits, soit exactement deux tranches de 4 bits : il faut donc **2 symboles hexadécimaux** pour écrire n'importe quel octet (de $(00)_{16}$ à $(FF)_{16}$, soit de 0 à $255 = 2^8 - 1$). En binaire, écrire un octet demande 8 symboles (des 0 et des 1), alors qu'en hexadécimal 2 symboles suffisent : l'écriture hexadécimale est donc **4 fois plus compacte**, puisque chaque symbole hexadécimal condense exactement 4 bits.

Comme nous l'avons dit précédemment, la représentation hexadécimale est beaucoup utilisée pour coder les adresses mémoire dans un ordinateur. Elle est également très utilisée dans le codage des couleurs. Par exemple, le tableau ci-dessous donne les codes hexadécimaux des couleurs primaires et secondaires, ainsi que du noir et du blanc :

Code hex.	000000	FF0000	00FF00	0000FF	FFFF00	FF00FF	00FFFF	FFFFFF
Couleur								
Nom	noir	rouge	vert	bleu	jaune	magenta	cyan	blanc

Nous reviendrons ultérieurement sur le problème du codage des couleurs.

3 Conversion d'un nombre d'une base vers une autre

Comme nous l'avons dit précédemment, un ordinateur ne comprend que le langage binaire, mais il est plus simple pour un informaticien de coder en hexadécimal. Il est donc nécessaire de pouvoir passer facilement d'une numération à une autre. Pour cela, on dispose de tables de correspondance entre les bases, ou de méthodes de calcul, permettant d'effectuer facilement le passage dans un sens ou dans l'autre.

3.1 Conversion entre les numérations binaire et hexadécimale

Pour cela, on dispose d'une table de correspondance entre la base 2 et la base 16, permettant d'effectuer facilement le passage dans un sens ou dans l'autre :

Hex.	**Bin.**	**Hex.**	**Bin.**	**Hex.**	**Bin.**	**Hex.**	**Bin.**
0	0000	4	0100	8	1000	*C*	1100
1	0001	5	0101	9	1001	*D*	1101
2	0010	6	0110	*A*	1010	*E*	1110
3	0011	7	0111	*B*	1011	*F*	1111

Méthode 3.1: Passage de la base 2 à la base 16

- On décompose le nombre écrit en binaire par tranches de quatre bits, à partir du bit de poids faible.
- On complète la dernière tranche à gauche (celle contenant le bit de poids fort) par des 0, s'il y a lieu.
- On convertit chaque tranche en son symbole en hexadécimal.

Exemple 3.2: De la base 2 à la base 16

Convertissons le nombre $(11\ 1101)_2$ en base **16** :

$$\begin{array}{cc} 11 & 1101 \\ \downarrow & \downarrow \\ 0011 & 1101 \\ \downarrow & \downarrow \\ 3 & D \end{array}$$

- On décompose le nombre écrit en binaire par tranches de quatre bits à partir du bit de **poids faible**.
- On complète à gauche (celle du bit de poids fort) par **des 0** s'il y a lieu.
- On convertit chaque tranche en son symbole en hexadécimal.

Ainsi on obtient : $(11\ 1101)_2 = (3D)_{16}$.

Méthode 3.3: Passage de la base 16 à la base 2

- On convertit chaque symbole hexadécimal en son écriture binaire (on obtient alors des tranches de quatre bits).
- On regroupe toutes les tranches de quatre bits, à partir du bit de poids faible, sous forme d'un seul nombre binaire.

Exemple 3.4: De la base 16 à la base 2

Convertissons le nombre $(23D5)_{16}$ en base **2** :

$$\begin{array}{cccc} 2 & 3 & D & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 0010 & 0011 & 1101 & 0101 \end{array}$$

- On convertit chaque symbole hexadécimal en son écriture binaire.
- On regroupe toutes les tranches de quatre bits, à partir du bit de poids faible, sous forme d'un seul nombre binaire.

Ainsi on obtient : $(23D5)_{16} = (10001111010101)_2$.

3.2 Conversion d'une écriture décimale vers la base b

La conversion d'une écriture décimale vers la base b reste la plus délicate. Les algorithmes que nous allons voir utilisent la division euclidienne (ou division entière) de deux entiers, vue en primaire et au collège, ainsi que la partie entière d'un nombre réel x.

Si le nombre x est entier, on peut utiliser un algorithme qui calcule les coefficients a_i suivant les puissances croissantes de b (on sait que l'on part de b^0 !), qui a l'avantage d'être simple et compréhensible.

Méthode 3.5: Passage de la base 10 à la base b

- On calcule les coefficients $\mathbf{a_i}$ en calculant le reste de la division euclidienne de $\mathbf{x}$ par $\mathbf{b}$:

Dividende	Diviseur
Reste	Quotient

- On répète l'opération en remplaçant $\mathbf{x}$ par le **quotient** obtenu.
- On recopie ensuite les restes du dernier au premier, dès que le quotient devient nul.

Exemple 3.6: Convertissons 57 en base 2

On pose successivement les divisions par 2 :

$$57 \left|\begin{array}{l}2\\28\end{array}\right. \rightarrow 28 \left|\begin{array}{l}2\\14\end{array}\right. \rightarrow 14 \left|\begin{array}{l}2\\7\end{array}\right. \rightarrow 7 \left|\begin{array}{l}2\\3\end{array}\right. \rightarrow 3 \left|\begin{array}{l}2\\1\end{array}\right. \rightarrow 1 \left|\begin{array}{l}2\\0\end{array}\right.$$

avec pour restes successifs, dans le même ordre : 1, 0, 0, 1, 1, 1.
Ce qui s'écrit aussi, sous forme d'égalités :

$$\begin{aligned} 57 &= 2 \times 28 + 1 && \text{donc } a_0 = 1 \\ 28 &= 2 \times 14 + 0 && \text{donc } a_1 = 0 \\ 14 &= 2 \times 7 + 0 && \text{donc } a_2 = 0 \\ 7 &= 2 \times 3 + 1 && \text{donc } a_3 = 1 \\ 3 &= 2 \times 1 + 1 && \text{donc } a_4 = 1 \\ 1 &= 2 \times 0 + 1 && \text{donc } a_5 = 1 \end{aligned}$$

En recopiant les restes du dernier au premier $(a_5a_4a_3a_2a_1a_0)$, on obtient $\mathbf{57} = (\mathbf{11\ 1001})_{\mathbf{2}}$.

Exemple 3.7: Convertissons 7654 en base 16

On pose successivement les divisions par 16 :

$$7654 \left|\begin{array}{l}16\\478\end{array}\right. \rightarrow 478 \left|\begin{array}{l}16\\29\end{array}\right. \rightarrow 29 \left|\begin{array}{l}16\\1\end{array}\right. \rightarrow 1 \left|\begin{array}{l}16\\0\end{array}\right.$$

avec pour restes successifs : 6, 14, 13, 1.
Ce qui s'écrit aussi, sous forme d'égalités :

$$\begin{aligned} 7654 &= 16 \times 478 + 6 && \Rightarrow 6 \\ 478 &= 16 \times 29 + 14 && \Rightarrow 14 = \text{E} \\ 29 &= 16 \times 1 + 13 && \Rightarrow 13 = \text{D} \\ 1 &= 16 \times 0 + 1 && \Rightarrow 1 \end{aligned}$$

En recopiant les restes convertis du dernier au premier, on obtient $\mathbf{7654} = (\mathbf{1DE6})_{\mathbf{16}}$.

Remarque : Attention aux restes à deux chiffres

Attention à convertir les restes supérieurs à 9 par les lettres correspondantes lorsque l'on travaille en base 16 (par exemple $14 \rightarrow$ E et $13 \rightarrow$ D dans l'exemple précédent).