Numérique et science informatique

Lycée Hoche

année scolaire 2024-2025

Contents

1 Introduction

Vus de l'extérieur, les ordinateurs et les programmes que l'on utilise tous les jours permettent de mémoriser, de transmettre et de transformer des nombres, des textes, des images, des sons, etc.

Pourtant, quand on les observe à une plus petite échelle, ces ordinateurs ne manipulent que des objets beaucoup plus simples : des 0 et des 1. Mémoriser, transmettre et transformer des nombres, des textes, des images ou des sons demande donc d'abord de les représenter comme des suites de 0 et de 1.

La mémoire des ordinateurs est constituée d'une multitude de petits circuits électroniques qui ne peuvent être, chacun, que dans deux états . Comme il fallait donner un nom à ces états, on a décidé de les appeler 0 et 1, mais on aurait pu tout aussi bien les appeler A et B, froid et chaud ou faux et vrai.

Un tel circuit à deux états s'appelle un circuit mémoire un bit et son état se décrit donc par le symbole 0 ou par le symbole 1. L'état d'un circuit, composé de plusieurs de ces circuits mémoire un bit, se décrit par une suite finie de 0 et de 1, que l'on appelle un mot. Par exemple, le mot 100 décrit l'état d'un circuit composé de trois circuits mémoire un bit, respectivement dans l'état 1, 0 et 0.

Dans un circuit, on dispose de deux niveaux de voltage (parfois notés V SS et V DD) pour représenter toute information, qu'elle soit de nature logique ou numérique. On représente donc les nombres en base 2 en associant par exemple la valeur binaire 0 au voltage V SS , et la valeur binaire 1 au voltage V DD .

Les circuits arithmétiques opèrent sur des nombres binaires, et la taille des circuits est généralement corrélée à la taille des nombres. La taille des circuits étant bornée par des contraintes matérielles, il est également nécessaire de restreindre la taille des nombres qui peuvent être traités par un opérateur arithmétique.

Définitions

1. Une telle valeur, 0 ou 1, s'appelle un **booléen**, un chiffre binaire ou encore un **bit** (binary digit).
2. Le bit le plus à gauche d'un nombre binaire est appelé **bit de poids fort**, et le bit le plus à droite est appelé **bit de poids faible**.
3. Lorsqu'un processeur comporte des opérateurs qui ne peuvent effectuer des calculs que sur des nombres d'au plus N bits, on dit que le processeur a un mot de taille N ou qu'il s'agit d'un processeur N-bits .

Le premier microprocesseur (Intel 4004 [1]) était un processeur 4-bits, le Pentium 4 [2] est un processeur 32-bits, et l'Itanium est un processeur 64-bits.

Exercice

1. On imagine un ordinateur dont la mémoire est constituée de quatre circuits mémoire un bit. Quel est le nombre d'états possibles de la mémoire de cet ordinateur ?
2. Même question pour un ordinateur dont la mémoire est constituée de dix circuits mémoire un bit.
3. Même question pour un ordinateur dont la mémoire est constituée de 34 milliards de tels circuits.
4. Trouvez trois informations de la vie courante qui peuvent être exprimées par un mot d'un bit.

processeur n-bits	nombre d'états possibles	nombres entiers codés
2 bits	4	de 0 à 3
3 bits	8	de 0 à 7
8 bits	256	de 0 à 255
16 bits	65536	de 0 à 65535

2 Les représentations binaires et hexadécimales

Définition

L'action qui consiste à représenter un objet en un mot binaire s'appelle un codage. Un codage peut être standardisé internationalement ou être localisé.

2.1 Un peu d'histoire

Depuis le Moyen Âge, on écrit les nombres entiers naturels en notation décimale à position. Cela signifie que, pour écrire le nombre entier naturel n, on commence par imaginer n objets, que l'on groupe par paquets de dix, puis on groupe ces paquets de dix objets en paquets de dix paquets, etc. À la fin, il reste entre zéro et neuf objets isolés, entre zéro et neuf paquets isolés de dix objets, entre zéro et neuf paquets isolés de cent, etc. Et on écrit cet entier naturel en écrivant de droite à gauche, le nombre d'objets isolés, le nombre de paquets de dix, le nombre de paquets de cent, le nombre de paquets de mille, etc.
Chacun de ces nombres étant compris entre zéro et neuf, seuls dix chiffres sont nécessaires : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. Par exemple, l'écriture 2359 exprime un entier naturel formé de 9 unités, 5 dizaines, 3 centaines et 2 milliers.
Le choix de faire des paquets de dix est arbitraire : on aurait pu tout aussi bien décider de faire des paquets de deux, de cinq, de douze, de vingt, de soixante, etc. On écrirait alors les nombres entiers naturels en notation à position en base deux, cinq, douze, vingt ou soixante. La notation décimale à position s'appelle donc aussi la notation à position en base dix.

2.2 Le système décimal

Tout nombre entier naturel peut être écrit comme une combinaison linéaire de puissances de dix:

$$164 = 1 \times 10^2 + 6 \times 10^1 + 4 \times 10^0$$

Plus généralement tout entier a s'écrit sous la forme:

$$a = d_n \times 10^n + d_{n-1} \times 10^{n-1} + ...d_1 \times 10^1 + d_0 \times 10^0$$

où chaque d_i est un des chiffres $0, 1, ..., 9$ avec $d_n \neq 0$.

Cette écriture est la *décomposition décimale* de l'entier a et le $n-uplet$ $(d_n, d_{n-1}, ..., d_0)$ est la *représentation décimale* de a notée plus simplement $d_n d_{n-1} ... d_0$.

Exercice

Ecrire la représentation décimale des nombres entiers 2020 et de 1239.

Pour obtenir les chiffres de la représentation décimale d'un entier , deux opérateurs sont utiles:

- La division entière, //en Python , qui à deux entiers a et b associe leur quotient q
- Le reste de la division entière %, qui à deux entiers a et b associe le reste r de la division euclidienne :$r = a - b \times q$.

On peut alors utiliser l'algorithme suivant pour obtenir l'écriture en binaire d'un entier n:

```
Choisir un entier naturel n.
Initialiser r à n.
Tant que r n'est pas nul, répéter les instructions suivantes.
    Calculer r % 10 et stocker le résultat.
    Remplacer r par r // 10
Renvoyer les chiffres stockés
```

2.3 La représentation en base deux

Nous pouvons représenter des nombres entiers positifs par la suite de bits de leur représentation binaire.Ce codage est important car il permet de faire directement des opérations sur ces nombres.Ainsi pour l'addition de deux nombres.Plus précisément, toutes les valeurs numériques sont non seulement stockées en binaire, mais aussi manipulées en binaire dans les ordinateurs.A un entier donné, on associe un ensemble de **bits** stockés dans la mémoire qui correspond à la présence ou à l'absence d'une tension.

Définition

Tout entier a possède une *représentation binaire* à l'aide des signes 0 et 1 :
le $n-uplet$ $(b_n, b_{n-1}, ..., b_0)$ où les b_i sont 0 ou 1 sauf b_n qui est obligatoirement le signe 1:
La *représentation binaire* de a notée plus simplement $(b_n b_{n-1} ... b_0)_2$.

Exemples : $(10)_2$ et $(11)_2$ sont les représentations binaires des entiers 2 et 3 respectivement.

$2 = 1 \times 2^1 + 0 \times 2^0$ et $3 = 1 \times 2^1 + 1 \times 2^0$

Algorithme de conversion

Soit a un entier écrit dans le système décimal:

```
1.Si le reste de la division de a par 2 est égal à zéro
écrire 0 , sinon écrire 1 à gauche de ce qui a déjà été écrit.

2. Remplacer a par la partie entière de a/2 .

3.Si a=0 c'est terminé, sinon on retourne à l'étape 1.
```

Exécutons l'algorithme avec $a = 13$

a	13	6	3	1	0
$a\%2$	1	1	0	1	
codage	1	11	011	$\boxed{1011}$	

2.4 Le codage binaire en machine sur N bits

Si N est un entier naturel non nul, 2^N entiers peuvent être codés en binaire sur N bits : $0, 1, 2, ..., 2^N - 1$
La fonction suivante renvoie, sous la forme d'une chaîne de caractères, le codage binaire d'un entier naturel sur N bits.

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.

Ce résultat prend toute son importance dès qu'on veut réaliser des opérations arithmétiques élémentaires comme l'addition ou la multiplication. La soustraction sera abordée plus loin. La division entière

n'est pas abordée.

Illustrons notre propos avec N = 4 bits et deux entiers $n1 = 3$ et $n2 = 4$. L'addition de ces entiers est l'entier $n3 = 7$.
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 $n1$ et $n2$ 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 $n1 = 11$ et $n2 = 13$ 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. L'addition des codages réalise la somme modulo 2 puissance le nombre de bits de codage. Ceci est à rapprocher de la phrase de l'introduction selon laquelle 255 + 1 renvoie 0 plutôt que 256 sur certaines machines, en l'occurence les machines codant les entiers naturels sur 8 bits ou moins !

Pour conclure cet exposé, un code réalisant l'addition bit à bit est proposé. Des variations et des prolongements sur ce sujet sont possibles : multiplication binaire, notions d'additionneurs pouvant être abordée à travers la partie consacrée à l'architecture des machines, notamment lors de l'introduction aux circuits logiques.

2.5 La représentation hexadécimale d'un entier (base 16)

De même, tout entier a possède une *représentation hexadécimale*: $(h_n h_{n-1} ... h_0)$ où les h_i sont l'un des symboles 0 ,1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,A ,B ,C ,D ,E ,F avec $h_n \neq 0$.

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

Décimal	binaire	hexadécimal	Décimal	binaire	hexadécimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Le passage de binaire à hexadécimale peut se faire de la manière suivante:

- Soit le nombre $4E$ en hexadécimal .
  $4 = (0100)_2$ et $E = (1110)_2$ d'où $4E = (01001110)_2$.
- Soit 1101110 un nombre écrit en base 2:
  Le bloc de 4 chiffres (à partir de la droite) 1110 vaut E en base 16 et le bloc 0110 vaut 6 .
  D'où 1101110 a pour représentation $6E$ en base 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.

3 Conversions en Python

En Python, quelques fonctions permettent le passage d'une représentation à une autre. Tout d'abord, pour passer une représentation décimale à une représentation binaire ou hexadécimale, Python dispose des deux fonctions :

▷ **bin** qui reçoit un entier naturel et renvoie une chaîne de caractères associée à la représentation binaire de cet entier ; cette chaîne débute systématiquement par le préfixe 0b

```
>>> bin(123)
'0b1111011'
```

▷ **hex** qui procède de la même façon et renvoie une chaîne de caractères associée à la représentaton hexadécimale de l'entier, avec le préfixe 0x

```
>>> hex(123)
0x7b
```

Les opérations inverses peuvent se faire à l'aide de la fonction int qui reçoit deux arguments : une chaîne de caractères représentant un entier dans une base donnée et un entier représentant cette base

```
>>> int('0b1111011', 2)
123
```

```
>>> int('0x7b', 16)
123
```

4 Représentation binaire des entiers relatifs

4.1 Première approche

Une première idée intuitive consiste à utiliser la représentation binaire d'un entier naturel en lui ajoutant à gauche *un bit de signe* ,bit de poids le plus fort.

Par exemple si l'entier est positif ou nul , le bit de signe vaut 0 et si l'entier est négatif , le bit de signe vaut 1.

Exemples: Avec -3 : Comme $3 = (11)_2$, on prend 111 comme représentation binaire de -3

Un premier inconvénient se présente : le nombre 0 aurait deux représentations binaires :

00...0 et 10...0

Un autre inconvénient apparaît lorsqu'on effectue une addition:

Soit à additionner 0101 et 1011 représentations binaires de 5 et -3 suivant le procédé décrit ci-dessus. La somme donne 0000 ce qui est incohérent.

4.2 Addition modulaire

Considérons une machine N bits.

Les entiers naturels qu'il est possible de coder sont les entiers compris entre 0 et $2^N - 1$ inclus.

Considérons deux entiers naturels x et y compris entre 0 et $2^N - 1$:

- Si $x + y < 2^N$, le code binaire renvoyé est celui de $x + y$.
- Si $x + y \geqslant 2^N$, le code renvoyé est celui de $x + y$ modulo 2^N , c'est-à-dire le reste de la division par 2^N.

Considérons un entier x compris entre 1 et $2^N - 1$ et choisissons l'entier $y = 2^N - x$, entier compris entre 1 et $2^N - 1$.

La somme $x + y$ est égal à 2^N.
L'addition des codes binaires renvoie donc 0 (reste de la division par 2^N).

Autrement dit y correspond à l'opposé de x dans cette opération modulo 2^N.

Ce qui permet de construire des nombres négatifs pour lesquels l'addition des codes binaires renvoie un résultat conforme à l'arithmétique sur les décimaux.

D'où la définition:

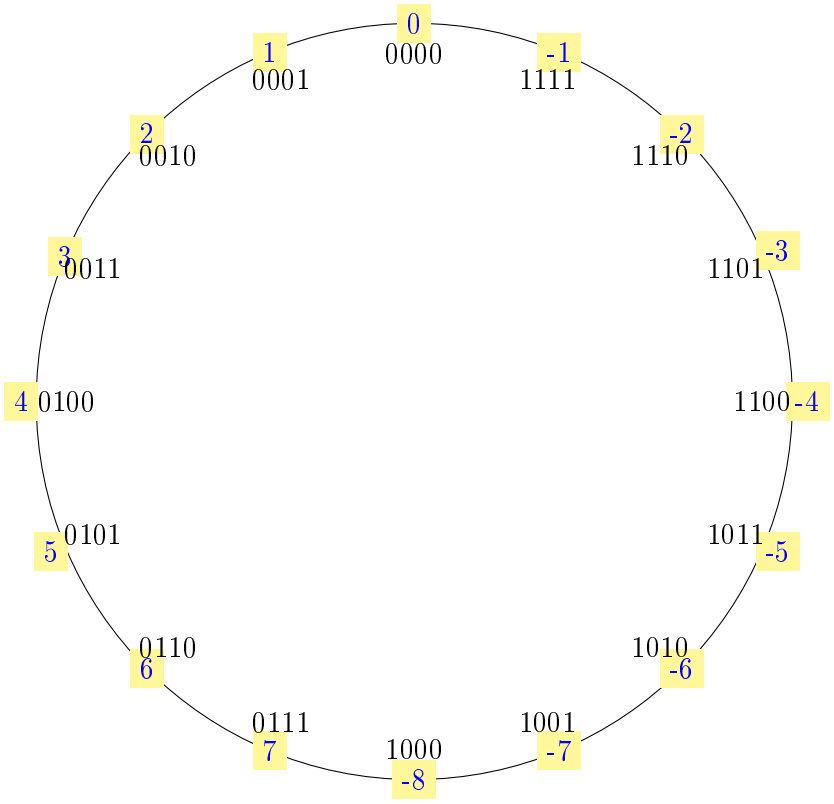
Définition

- Si x est un entier compris entre 0 et $2^{N-1} - 1$, le codage binaire est celui de l'entier naturel x sur N bits.
- Si x est un entier compris entre -2^{N-1} et -1 , le codage binaire est celui de l'entier naturel $2^N - |x|$ sur N bits.

Ce qui donne le tableau de correspondances :

-2^{N-1}	10...00
$-2^{N-1}+1$	10...01
...	...
-1	11...11
0	00...00
1	00...01
...	...
$2^{N-1}-1$	01...11

Dans cette représentation le zéro admet un seul codage.Par ailleurs, le bit **de poids fort** est 0 pour les entiers positifs et 1 pour les entiers négatifs.On retrouve l'idée du *bit de signe* vu précédemment.
On peut présenter ce qui précède visuellement à l'aide d'un cercle dans le cas d'une machine 4 bits.



Exemples:

- L'entier codé $(0110)_2$ a son bit de poids fort égal à 0:
  c'est l' entier positif égal à $1.2^2 + 1.2^1 + 0.2^0 = 6$
- l'entier codé $(1101)_2$ a son bit de poids fort égal à 1.
  Ce codage est celui de l'entier naturel $2^4 - |x|$ sur 4 bits .
  L'entier naturel ayant pour code binaire $(1101)_2$ est égal à 13.
  D'où: $|x| = 16 - 13 = 3$.
  C'est donc l'entier relatif -3.

4.3 Complément à deux

L'algorithme précédent permet effectivement de coder sur un nombre de bits fixé des entiers naturels mais il n'est pas commode à utiliser. On lui préfère la technique du complément à deux (CP2). Avant de la détailler, observons les codages suivants sur N = 4 bits et, plus particulièrement, le passage du codage d'un entier positif à celui d'un entier négatif.

1	0001	$\longrightarrow$	$\longleftarrow$	1111	-1
2	0010	$\longrightarrow$	$\longleftarrow$	1110	-2
3	0011	$\longrightarrow$	$\longleftarrow$	1101	-3
4	0100	$\longrightarrow$	$\longleftarrow$	1100	-4
5	0101	$\longrightarrow$	$\longleftarrow$	1011	-5
6	0110	$\longrightarrow$	$\longleftarrow$	1010	-6
7	0111	$\longrightarrow$	$\longleftarrow$	1001	-7
	inversion	$\longrightarrow$		ajouter 1	

La technique du complément à deux s'applique en deux temps :

1. Inversion des bits 0 et 1 de l'entier positif (c'est la phase dite du complément à 1)
2. Addition de 1 au code obtenu.

Appliquons cette technique pour coder l'entier -5 vu dans le paragraphe précédent, dont le codage binaire sur 4 bits est 1011.

- On code d'abord l'entier positif 5 sur 4 bits : 0101.
- On procède ensuite aux inversions (complément à un) : 1010.
- On ajoute 1 : 1011. Appliquons cette technique au code obtenu 1011.
- Inversions (complément à un) : 0100.
- Ajout de 1 : 0101.

On retrouve le code de 5. On dit que l'opération de complément à deux est involutive : pour tout entier n :

$$n \xrightarrow{CP2} (-n) \xrightarrow{CP2} n$$