Données de base 1
Représentation des nombres entiers, booléens

« Il existe 10 sortes de personnes, ceux qui connaissent le binaire et ceux qui ne le connaissent pas. »
Auteur inconnu.

Introduction

On sait bien qu'un ordinateur ne manipule que des 0 et des 1 (des « *bits* ») mais on ne sait pas très bien comment, ni pourquoi il le fait.

Dans ce chapitre, nous allons commencer à comprendre pourquoi toute information peut-être représentée par une suite de 0 et de 1 (on parle de nombre binaire), en étudiant comment les nombres entiers sont représentés dans un ordinateur. Nous comprendrons ainsi comment compte un ordinateur.

Ensuite, nous nous pencherons sur les booléens, ces variables à deux états (0 et 1, faux et vrai) avec lesquelles les ordinateurs font de la logique !

Rappels

Un entier naturel est un nombre entier positif ou nul. Ainsi, 345, 2, et 0 sont des exemples d'entiers naturels, tandis que -12 n'en est pas un, pas plus que 13,5.

I Écriture d'un entier naturel dans différentes bases

I.1 Qu'est-ce-qu'une base de numération et comment en changer ?

I.1.1 Comment fonctionne notre système de numération ?

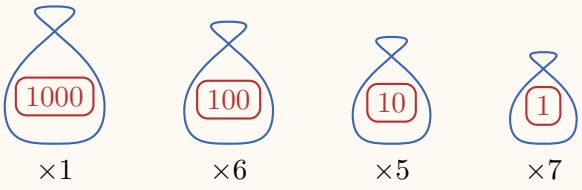
Pour écrire un entier naturel, on imagine que l'on remplit *complètement* des « sacs » dont les capacités sont des puissances de 10 : 1, 10, 100, 1000, etc. Pour chaque capacité, on note le nombre de sacs remplis. L'écriture du nombre est alors la suite des chiffres qui indiquent, pour chaque capacité, le nombre de sacs remplis.

C'est ce que l'on appelle la notation *décimale à position*. Décimale : les capacités des sacs sont des puissances de 10 ; position : c'est l'ordre des chiffres qui permet d'identifier, pour chaque capacité, le nombre de sacs.

Exemple

Par exemple, pour écrire le nombre 1657 en base 10,

- on remplit 1 sac de capacité 1000, il reste donc 657 unités ;
- puis 6 sacs de capacité 100, il reste donc 57 unités ;
- puis 5 sacs de capacité 10, il reste donc 7 unités ;
- et enfin, 7 sacs de une unité.



I.1.2 Changement de base

Dans la méthode que nous avons décrite, les capacités des sacs sont des puissances de 10 : $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$, etc. On dit que l'on écrit les nombres *en base 10*.

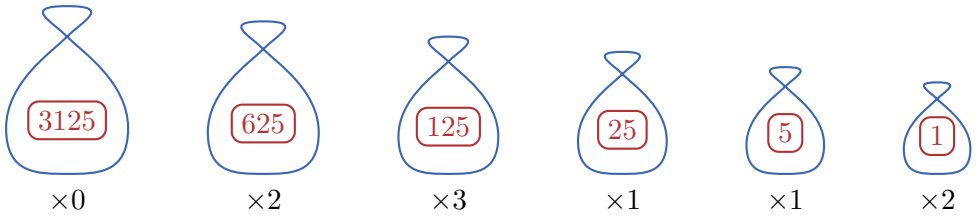
Que se passe-t-il si les capacités des sacs sont des puissances de 5 ? On commence par calculer les puissances successives de 5 : $5^0 = 1$, $5^1 = 5$, $5^2 = 25$, $5^3 = 125$, $5^4 = 625$, $5^5 = 3125$, etc. Puis on remplit les sacs !

Savoir-faire

Passer de la base 10 à une autre base (première méthode)

Par exemple, pour écrire le nombre 1657 en base 5,

- on remplit 0 sac de capacité 3125 car on se souvient que les sacs doivent être pleins ;
- on effectue la division euclidienne de 1657 par 625, on trouve 2, reste 407 ; on remplit donc 2 sacs de capacité 625 et il reste 407 unités ;
- on effectue la division euclidienne de 407 par 125, on trouve 3, reste 32 ; on remplit donc 3 sacs de capacité 125 et il reste 32 unités ;
- on effectue la division euclidienne de 32 par 25, on trouve 1, reste 7 ; on remplit donc 1 sac de capacité 25 et il reste 7 unités ;
- on effectue la division euclidienne de 7 par 5, on trouve 1, reste 2 ; on remplit donc 1 sac de capacité 5 et il reste 2 unités ;
- et enfin, on remplit 2 sacs de une unité.



L'écriture en base 5 de 1657 est donc « 23112 ».

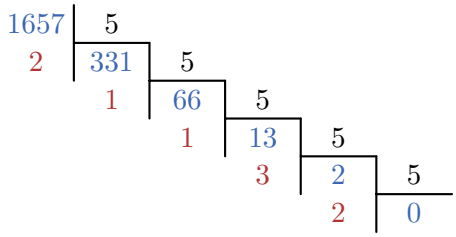
Savoir-faire

Passer de la base 10 à une autre base (à utiliser en pratique)

L'inconvénient de la première méthode est qu'il faut d'abord calculer les capacités des sacs (les puissances de 5). On peut se passer de cette étape en faisant successivement des paquets de 5 :

- $1657 = 331 \times 5 + 2$, donc on sait que le chiffre des unités et un 2 et qu'il y a 331 paquets de 5 (qui iront dans des sacs de capacité 5 ou plus) ;
- $331 = 66 \times 5 + 1$, le chiffre suivant est donc un 1 et il y a 66 paquets de 25 (qui iront dans des sacs de capacité 25 ou plus) ;
- $66 = 13 \times 5 + 1$, le chiffre suivant est donc un 1 et il y a 13 paquets de 125 (qui iront dans des sacs de capacité 125 ou plus) ;
- $13 = 2 \times 5 + 3$, le chiffre suivant est donc un 3 et il y a 2 paquets de 625 (qui iront dans des sacs de capacité 625 ou plus) ;
- $2 = 0 \times 5 + 2$, le chiffre suivant est donc un 2 et il y a 0 paquets de 3125.

Dans les deux cas, on a donc démontré que 1657 (en base 10) s'écrit en base 5 : « 23112 ».
En pratique, on présente les calculs par des divisions successives, comme ceci :



Ensuite, l'écriture du nombre dans la base cherchée se trouve en lisant les restes « en remontant ».

Astuce

Cette méthode est celle que l'on utilise habituellement quand on veut convertir des secondes en `heures:minutes:secondes`. On fait des divisions euclidiennes par 60 successives.

Un nombre a donc différentes écritures suivant la base que l'on choisit. Pour ne pas les confondre, on

indique la base en indice, comme ceci 23112_5. On peut donc écrire l'égalité suivante

$$1657_{10} = 23112_5.$$

Lorsqu'un nombre est écrit en base 10, on peut omettre de l'indiquer puisque c'est la base de notre système de numération. On écrira donc plutôt,

$$1657 = 23112_5$$

qui signifie : « 1657 s'écrit en base 5 : "23112" ».

Attention

Si l'on écrit un nombre dans une base autre que la base 10, il ne faut pas oublier de l'indiquer en indice, sinon, impossible de savoir de quelle base il s'agit !

Exercice

Exercice 1

Écrire 58 en base 5.

Exercice

Exercice 2

Écrire 872 en base 5.

Exercice

Exercice 3

Écrire 1207 en base 5.

Algorithme

Algorithme du changement de base. Le passage de la base 10 à une autre base est pour le moins répétitif. L'algorithme suivant, traduit en Python, décrit la méthode du paragraphe précédent.

Pseudo langage

Données : x, *base*
Initialiser L à une liste vide
tant que $x \neq 0$ **faire**
  Ajouter, au début de L, le reste de la division euclidienne de x par *base*
  $x \leftarrow$ quotient de la division euclidienne de x par *base*
Résultat : L

Python

```
# Écrit x dans une autre base.
# Conditions : x et base sont des int
#              x > 0 et base > 1.
def changer_base(x, base):
    res = ''
    while x != 0:
        x, r = divmod(x, base)
        res = str(r) + res
    return res
```

Teste de la fonction Python :

```
changer_base(1657, base=5)
```

```
'23112'
```

I.2 La base 2

Pourquoi les ordinateurs comptent en base 2 ? La mémoire des ordinateurs est constituée d'une multitude de petits circuits électroniques qui, chacun, ne peuvent être que dans deux états : basse tension ou haute tension [1]. On note ces états 0 et 1, (mais on aurait pu tout aussi bien les appeler A et B, faux et vrai, etc).

Le fonctionnement de la mémoire est alors tout à fait adapté pour représenter des nombres en base 2 car les chiffres qui composent un nombre en base 2 ne peuvent être que 0 ou 1. C'est pour cela que les ordinateurs comptent « naturellement » en base 2.

1. Pour plus de détails, consulter `https://en.wikipedia.org/wiki/Memory_cell_(computing)#DRAM_memory_cell`.

Définition

La base 2 porte aussi le nom de base *binaire*. Chaque chiffre d'un nombre écrit en binaire est appelé un *bit* (abréviation de *binary digit*).

Info

Une quantité qui ne peut prendre que deux valeurs (0 ou 1) est dite booléenne. Elle est représentée par un bit. Nous étudierons ceci dans la section IV.

Exemple

Écrivons 45 en base 2 :

45	2
1	22
	2
0	11
	2
1	5
	2
1	2
	2
0	1
	2
1	0

Ainsi, nous avons montré que $45 = 101101_2$.

python

Pour convertir un nombre en binaire, on peut utiliser la fonction `format` de Python[2], comme ceci :

```
format(45, 'b')  # 'b' comme binaire
```

```
'101101'
```

Info

Dans la mémoire d'un ordinateur, on regroupe souvent les bits par paquets de 8, que l'on appelle un *octet*[3].

Exercice

Exercice 4

Écrire 58 en base 2.

Exercice

Exercice 5

Écrire 872 en base 2.

Exercice

Exercice 6

Quel est le plus grand entier naturel que l'on peut représenter avec un octet ?

Exercice

Exercice 7

Notre méthode pour compter avec nos doigts n'est vraiment pas la meilleure : elle ne permet que d'aller de 0 à 10. Il existe une méthode qui permet de compter de 0 à 1023, pouvez-vous deviner laquelle ?

Exercice

Exercice 8

Il y a quelques années, l'architecture standard des ordinateurs utilisait des registres de taille 32 bits (on parle d'architecture 32 bits).

1) Quel est l'entier naturel le plus grand que l'on peut représenter en utilisant un seul registre d'une architecture 32 bits ?

2. On peut aussi utiliser la fonction `bin`.
3. Ou *byte* en anglais, à ne pas confondre avec bit.

2) De nos jours, l'architecture standard utilise 64 bits. Répondre à la question précédente pour une taille de 64 bits.

Info

Lorsqu'un entier est représenté en binaire, son bit des unité (le plus à droite) est appelé le *bit de poids faible*, tandis que le bit le plus à gauche est appelé le *bit de poids fort*.

I.3 Problème des bases supérieures à 10, exemple de la base 16

Pour écrire en base 10, nous avons besoins de 10 chiffres : de 0 à 9 ; pour la base 5, 5 chiffres : de 0 à 4 ; pour la base 2, seulement deux chiffres : 0 et 1. Pour les bases supérieures à 10, on complète les chiffres de 0 à 9 par autant de lettres que nécessaire, en commençant dans l'ordre alphabétique.

Par exemple, en base 16, les nombres s'écrivent avec les chiffres de 0 à 9 et les lettres de a à f.

La base hexadécimale. La base 16, appelée base *hexadécimale*, joue un rôle particulier en informatique. L'écriture d'un nombre en base 2 est en moyenne plus de 3 fois plus longue que l'écriture d'un nombre en base 10 ce qui la rend difficile à mémoriser pour un humain. Pourtant, en informatique, on a souvent besoin de saisir, lire ou mémoriser des suites de bits, comme par exemple une clé de connexion à un réseau WiFi ou une adresse dans la mémoire. Il devient vite peu commode de le faire en base deux et on multiplie les risques d'erreurs alors que ce sont justement des données sur lesquelles une erreur est fatale.

On préfère donc regrouper les bits par paquets de 4 ce qui donne des valeurs en base $2^4 = 16$. L'écriture d'une suite de bits est alors 4 fois plus compacte et reste manipulable par un humain.

Info

Un octet est donc représenté par deux chiffres hexadécimaux. Par exemple,

$$1011\ 0011_2 = b3_{16}.$$

Exemple

Écrivons 2687 en base 16 :

2687	16		
15	167	16	
	7	10	16
		10	0

On peut donc écrire $2687 = a7f_{16}$.

python

Pour convertir un nombre en base hexadécimale, comme en binaire, on peut utiliser la fonction `format` de Python [4], comme ceci :

```
format(2687, 'x')  # 'x' comme dans hexadécimal
```

```
'a7f'
```

Exercice

Exercice 9

Écrire 91 en base hexadécimale.

Exercice

Exercice 10

Écrire 1657 en base hexadécimale.

4. On peut aussi utiliser la fonction `hex`.

I.4 Revenir en base 10

Savoir-faire

Passer d'une base à la base 10

La méthode pour passer d'une base quelconque à la base 10 est plus simple. Il suffit en effet de multiplier le nombre de sacs par leur capacité et d'ajouter les nombres obtenus.
Reprenons l'exemple de 23112_5. On peut écrire :

$$\begin{aligned} 23112_5 &= 2\times 5^4 + 3\times 5^3 + 1\times 5^2 + 1\times 5^1 + 2\times 5^0 \\ &= 2\times 625 + 3\times 125 + 1\times 25 + 1\times 5 + 2\times 1 \\ &= 1657. \end{aligned}$$

Pour simplifier la réalisation de ce calcul, on peut commencer par construire le tableau suivant :

Rang n	4	3	2	1	0
5^n	625	125	25	5	1
Nombre	2	3	1	1	2

Exercice

Exercice 11

Calculer la représentation en base 10 de 202413_5.

Exercice

Exercice 12

Calculer la représentation en base 10 de 101011_2.

Exercice

Exercice 13

Calculer la représentation en base 10 de $5ad9_{16}$.

Astuce

Plutôt que de commencer par calculer les puissances de la base de départ (les capacités des sacs) on peut les calculer en même temps que l'on calcule le résultat, comme dans la fonction Python ci-dessous.

```
# Écrit x en base 10.
# Conditions : x est de type string et base > 1.
def base_10(x, base):
    res = 0
    sac = 1
    for a in reversed(x):
        res = res + int(a) * sac
        sac = sac * base
    return res
```

```
base_10('23112', base=5)
```

```
1657
```

python

En fait, Python sait déjà faire ceci. En effet, la commande `int` de Python permet de passer d'une base à la base 10 :

```
int('1110', base=2)
```

```
14
```

Exercice

Exercice 14 (difficile)

Pourquoi la fonction Python suivante permet-elle bien de convertir un nombre en base 10 ?

```
# Écrit x en base 10.
# Conditions : x est de type string et base > 1.
def base_10(x, base):
    res = 0
    for a in x:
        res = res * base + int(a)
    return res
```

Exercice

Exercice 15 Télépathie

Un tour de magie de « télépathie » consiste à demander à un spectateur de penser à un nombre entier entre 1 et 60 inclus puis d'indiquer au magicien parmi les cartes ci-dessous, celles qui comportent le nombre choisi. Le magicien fait alors, dans sa tête, la somme des nombres en haut à gauche de chaque carte contenant la valeur, le total obtenu est le nombre choisi par le spectateur.

1	3	5	7	9	11	13	15
17	19	21	23	25	27	29	31
33	35	37	39	41	43	45	47
49	51	53	55	57	59	61	63

4	5	6	7	12	13	14	15
20	21	22	23	28	29	30	31
36	37	38	39	44	45	46	47
52	53	54	55	60	61	62	63

16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

2	3	6	7	10	11	14	15
18	19	22	23	26	27	30	31
34	35	38	39	42	43	46	47
50	51	54	55	58	59	62	63

8	9	10	11	12	13	14	15
24	25	26	27	28	29	30	31
40	41	42	43	44	45	46	47
56	57	58	59	60	61	62	63

32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

1) Vérifier que le tour fonctionne avec les nombres 10 puis 57.
2) En remarquant que les premiers nombres de chaque carte sont des puissances de 2, expliquer comment fonctionne ce tour.
3) En déduire une méthode (ou mieux, un programme Python) pour construire les cartes.

II Addition de deux nombres binaires

On s'intéresse dans cette partie à l'addition de deux entiers naturels écrits en base 2.

Savoir-faire

Addition de deux nombres binaires

Propriété (admise)**.** Les règles pour poser une addition pour la base 10 fonctionnent aussi pour ajouter deux nombres en base 2.

Essayons avec 11101_2 et 1010_2 :

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ \ \ } \\ 1\,1\,1\,0\,1_2 \\ +\,0\,1\,0\,1\,0_2 \\ \hline 1\,0\,0\,1\,1\,1_2 \end{array}$$

Vérifions : $11101_2 = 29$, $1010_2 = 10$, $29 + 10 = 39 = 100111_2$.

Exercice

Exercice 16

Ajouter 1010_2 et 111_2 puis vérifier le résultat en base 10.

Exercice

Exercice 17

Ajouter 1010001_2 et 10011_2 puis vérifier le résultat en base 10.

III Représentation binaire d'un entier relatif

Dans cette section, on s'intéresse à la représentation binaire d'entiers qui peuvent être positifs ou négatifs.

Rappels

Un entier relatif est un nombre entier qui peut être positif, négatif ou nul. Ainsi, -128, 0 et 7 sont des entiers relatifs tandis que 0,5 ou π n'en sont pas.

III.1 Une approche naïve

Après tout, un entier relatif est juste un entier naturel auquel on donne un signe. On pourrait alors représenter un entier relatif comme un entier naturel puis utiliser un bit supplémentaire pour coder le signe (0 pour « − » et 1 pour « + », par exemple).

Ainsi, sur 4 bits, 7 s'écrit « 1111 » et -7 s'écrit « 0111 ».

Problèmes. Cette méthode comporte au moins deux inconvénients importants :

- il y a deux manières d'écrire 0 : $+0$, qui s'écrit « 1000 » et -0 qui s'écrit « 0000 » ;
- les règles de l'addition de la section précédente ne fonctionnent plus... Si l'on essaye, par exemple, d'ajouter 7 avec -7, on ne trouve pas 0...

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ } \\ 1\,1\,1\,1_2 \\ +\,0\,1\,1\,1_2 \\ \hline 1\,0\,1\,1\,0_2 \end{array}$$

III.2 Le complément à 2

Dans cette section, on étudie comment les entiers relatifs sont en réalité représentés dans un ordinateur. Cette méthode ne possède pas les inconvénients de la méthode précédente, en particulier, elle est compatible avec l'addition. Elle porte le nom de *complément à 2*.

Dans la méthode du complément à 2, on commence *toujours* par fixer le nombre de bits avec lequel on travaille. Disons 4 bits pour fixer les idées. Ensuite, on décide de coder chaque entier relatif comme un entier naturel : les nombres de 0 à 7 sont codés de la même manière alors que les nombres de -8 à -1 sont codés comme les entiers naturels de 8 à 15, comme sur la figure ci-dessous.

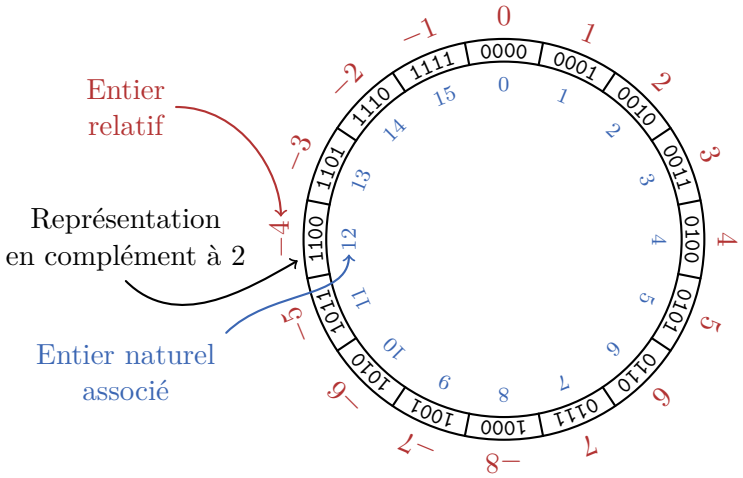


FIGURE 1 – Représentation des entiers relatifs en complément à 2, sur 4 bits.

Nous avons vu que sur n bits, on peut représenter 2^n valeurs, soit les nombres de 0 à $2^n - 1$. Le plus grand entier relatif que l'on peut représenter avec la méthode du complément à 2 est donc $2^{n-1} - 1$, le plus petit est -2^{n-1}.

Sur 4 bits, on représentes donc les entiers de -8 à 7 alors que sur 8 bits, on peut représenter les entiers de -128 à 127 comme le montre la figure ci-dessous.

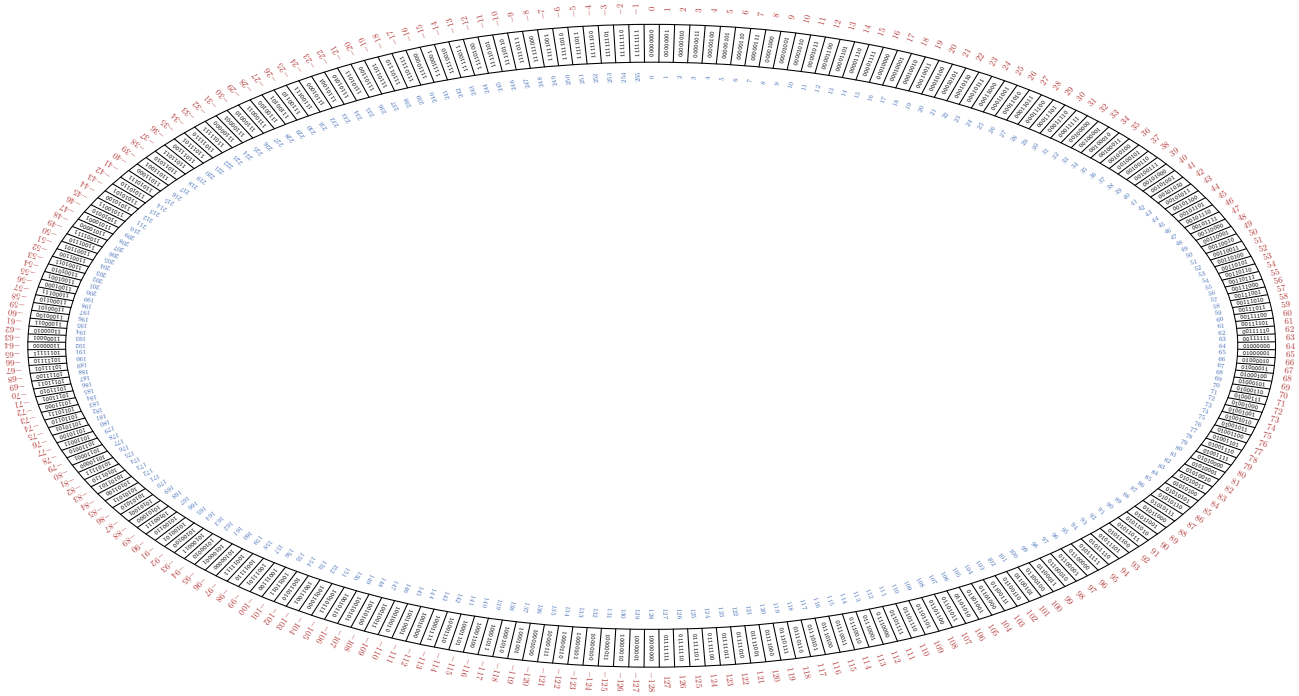


FIGURE 2 – Représentation des entiers relatifs en complément à 2, sur 8 bits.

Exercice

Exercice 18

Indiquer dans le tableau ci-dessous les plus grands et plus petits entiers relatifs représentables avec la méthode du complément à 2.

Nombre de bits	Plus petit entier relatif	Plus grand entier relatif
4	-8	7
8	-128	127
16		
32		
64		

Attention

Si l'on oublie de fixer clairement au départ le nombre de bits avec lequel on travaille, on risque de se tromper. En effet, par exemple, si l'on travaille sur 4 bits, -2 est codé comme 1110_2 alors que sur 8 bits, il est codé comme $1111\ 1110_2$.

À ce stade, on peut se demander pourquoi la méthode du complément à 2 se nomme ainsi. Si l'on travaille sur n bits, on peut remarquer que la somme (en binaire) d'un entier et de son opposé donne toujours 2^n !

Exemple

Sur 4 bits par exemple, si l'on ajoute 6 et -6, comme -6 est codé comme 10 (voir la figure 1) on a bien $6 + 10 = 16 = 2^4$.

Autrement dit, l'opposé d'un nombre est représenté en binaire comme le nombre qu'il faut lui ajouter pour trouver 2^n. Cette méthode devrait plus précisément s'appeler la « méthode du complément à 2^n » mais on la nomme toujours la *méthode du complément à 2*.

Astuce

Si p et p' sont deux nombres qui en binaire sur n bits représentent des nombres opposés, alors $p + p' = 2^n$, c'est donc que $p' = 2^n - p$.

Exercice

Exercice 19

Trouver la représentation en binaire sur 8 bits de -100.

Exercice

Exercice 20

Trouver la représentation en binaire sur 16 bits de -859.

III.3 Addition binaire de deux entiers relatifs

Savoir-faire

Addition binaire de deux entiers relatifs

Les entiers relatifs s'ajoutent comme les entiers naturels, en posant l'addition. Si on travaille sur 4 bits et que l'on cherche par exemple à ajouter 5 et -7 qui sont respectivement représentés par 0101_2 et 1001_2 (voir la figure 1), on trouve

$$\begin{array}{r} {\scriptstyle 1} \\ 0\ 1\ 0\ 1_2 \\ +\ 1\ 0\ 0\ 1_2 \\ \hline 1\ 1\ 1\ 0_2. \end{array}$$

Or, on sait que 1110_2 est la représentation de -2 (voir la figure 1). On retrouve bien évidement que $5 + (-7) = -2$.

Exercice

Exercice 21

Sur 4 bits, réaliser, en binaire, l'opération $-5 + 6$.

Exercice

Exercice 22

Toujours sur 4 bits, réaliser, en binaire, l'opération $-3+3$. Pourquoi ne trouve-t-on pas 0 ?

Info

Lorsqu'il réalise une addition en binaire, un ordinateur n'a pas besoin de savoir s'il est en train d'ajouter deux entiers naturels ou deux entiers relatifs car la méthode est la même !

III.4 Manipulation des entiers relatifs

Savoir-faire

Trouver l'écriture binaire d'un entier relatif donné en décimal

Si p est un entier relatif et que l'on cherche sa représentation en binaire sur n bits, il y a deux cas :

- soit p est positif ou nul et on le représente comme un entier naturel (voir la section I.2) ;
- soit p est strictement négatif et on le représente comme l'entier naturel $2^n + p$.

Exercice

Exercice 23

Quelle est l'écriture binaire de -51 sur 8 bits ?

Exercice

Exercice 24

Quelle est l'écriture binaire de -1000 sur 16 bits ?

Astuce

Si un entier relatif est représenté en binaire, on peut connaître son signe (positif ou négatif) sans calculer sa représentation décimale. En effet, sur n bits, les entiers de 0 à $2^{n-1}-1$ ont leur bit de poids fort qui vaut 0. Tandis que les entiers de -2^{n-1} à -1 s'écrivent comme les entiers de 2^{n-1} à 2^n-1 qui ont leur bit de poids fort qui vaut 1. On peut facilement vérifier ceci sur la figure 1.

Savoir-faire

Trouver l'écriture décimale d'un entier relatif donné en binaire

Soit m la représentation en binaire d'un entier relatif p sur n bits. On commence par calculer p, l'entier naturel représenté par m, comme à la section I.2. Ensuite, il y a deux cas :

- soit le bit de poids fort de m est un 0, l'entier cherché est donc positif ou nul, c'est p lui-même ;
- soit le bit de poids fort de m est un 1, l'entier cherché est donc strictement négatif, c'est $p-2^n$.

Exercice

Exercice 25

Quel est l'entier relatif dont l'écriture binaire sur 8 bits est $0000\ 1111_2$?

Exercice

Exercice 26

Quel est l'entier relatif dont l'écriture binaire sur 16 bits est $1000\ 1000\ 1000\ 1000_2$?

Savoir-faire

Trouver la représentation en binaire de l'opposé d'un entier naturel donné en binaire

Soit p un entier relatif dont on connais la représentation en binaire sur n bits. On cherche la représentation en binaire de $-p$, l'opposé de p. Deux méthodes sont possibles :

1) On sait que la somme en binaire de p et de $-p$ doit donner 2^n (c'est le complément à 2). On pose donc l'addition à l'envers : en connaissant le résultat (un 1 suivit de n zéros).
2) On inverse tous les bits de la représentation donnée puis on lui ajoute 1 (en binaire).

Exemple

Supposons que l'on travaille sur 8 bits et que l'on cherche l'écriture binaire de l'opposé du nombre relatif représenté par $1101\ 1001_2$.

1) Si l'on utilise la première méthode, on pose (à l'envers) :

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ \ 1\ 1\ 1} \\ 1\,1\,0\,1\ \ 1\,0\,0\,1_2 \\ +\,0\,0\,1\,0\ \ 0\,1\,1\,1_2 \\ \hline 1\,0\,0\,0\,0\ \ 0\,0\,0\,0_2. \end{array}$$

L'écriture cherchée est donc $0010\ 0111_2$.

2) Avec la deuxième méthode, on inverse les bits et on ajoute 1 :

$$\begin{array}{r} 1\,1\,0\,1\ \ 1\,0\,0\,1_2 \\ 0\,0\,1\,0\ \ 0\,1\,1\,0_2 \\ +\,0\,0\,0\,0\ \ 0\,0\,0\,1_2 \\ \hline 0\,0\,1\,0\ \ 0\,1\,1\,1_2. \end{array}$$

On peut vérifier que $1101\ 1001_2$ est l'écriture de -39 et que $0010\ 0111_2$ est bien celle de 39.

Exercice

Exercice 27

Quelle est l'écriture de l'opposé de l'entier relatif représenté par $0101\ 0101_2$ sur 8 bits ? Vérifier le résultat avec les valeurs décimales.

Exercice

Exercice 28

Quelle est l'écriture de l'opposé de l'entier relatif représenté par 1000_2 sur 4 bits ? Où est l'erreur ?

Exercice

Exercice 29 (difficile)

Démontrer que la méthode qui consiste à inverser chaque bit et à ajouter 1 permet bien de calculer l'opposé d'un entier relatif.

python

Dans la plupart des langages de programmation, l'écriture des entiers est limitée à un nombre de bits fixé à l'avance. En Python, pas besoin de se poser de question car on peut représenter des entiers de taille arbitrairement grande. On est seulement limité par la mémoire de l'ordinateur et plus par la taille des registres.. Par exemple, le nombre d'atomes dans l'univers est estimé à 10^{80},

```
a = 10 ** 80
print("10^80 =", a)
print("En base 2 :", format(a, 'b'))
```

```
10^80 = 100000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000
En base 2 : 1101011111100111011110101000111110000111110110101111011111111011110111000
0110011011101000101111011001001011110111111010010000011000110010100110001110100010001
0011101100100110011101100000100000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
```

Mais si l'on désire jouer avec des entiers sur un nombre fixé de bits, on peut utiliser la bibliothèque `numpy`[5], comme ceci :

```python
import numpy as np

x = np.uint8(2 ** 8 - 1)  # un entier naturel sur 8 bits
y = np.int8(2 ** 7 - 1)   # un entier relatif sur 8 bits
print("x =", x, "y =", y)
```

```
x = 255 y = 127
```

```python
x + np.uint8(1)  # on teste que 256 n'est pas représentable sur 8 bits
```

```
0
```

```python
y + np.int8(1)  # devrait donner -128 d'après le complément à 2
```

```
-128
```

Exercice

Exercice 30

On considère les entiers relatifs $x = 31$ et $y = -7$.

1) Quel est le plus petit nombre de bits nécessaire à l'écriture de l'entier relatif x en base 2 ?
2) Quel est le plus petit nombre de bits nécessaire à l'écriture de l'entier relatif y en base 2 ?
3) Quel est le plus petit nombre de bits nécessaire à l'écriture de l'entier relatif $x + y$ en base 2 ?
4) Quel est le plus petit nombre de bits nécessaire à l'écriture de l'entier relatif $x \times y$ en base 2 ?

IV Les booléens

Dans la section précédente, nous avons vu combien la représentation des entiers [6] par un ordinateur se résume à la manipulation de quantités binaires, 0 et 1. Ces quantités sont appelées des booléens et nous les étudions en détail ici.

IV.1 Valeurs booléennes

Définition

Un booléen est une quantité qui ne peut prendre que deux valeurs, comme un bit. Nous noterons donc ces valeurs 0 et 1.

Un booléen peut donc représenter l'état d'un bit dans la mémoire d'un ordinateur ou encore une proposition logique qui serait soit vraie, soit fausse, ou même l'état d'un interrupteur dans un circuit électrique (ouvert ou fermé)...

python

Le type `bool` de Python sert à manipuler des expressions logiques, les valeurs utilisés ne sont pas 0 et 1 mais `False` et `True`.

```python
type(True)
```

```
bool
```

5. Numpy (comme sa sœur Scipy) est une bibliothèque de calcul très puissante utilisée par les scientifiques.
6. En fait, c'est toutes les données numérique qui est représentée ainsi.

```
type(False)
```

```
bool
```

```
type(1 == 2)
```

```
bool
```

Si l'on veut vraiment travailler avec les valeurs 0 et 1, il suffit de demander à Python de convertir le booléen en un entier.

```
int(True)
```

```
1
```

```
int(False)
```

```
0
```

On remarque que, par convention, Python associe l'entier 1 à la valeur logique « vrai » et 0 à « faux ». C'est une convention standard en informatique.

Dans la pratique, nous utiliserons essentiellement les booléens pour étudier des expressions logiques. Cependant, nous verrons dans la section suivante que ces expressions permettent de faire bien plus que seulement de la logique !

IV.2 Opérateurs booléens

Définition

Un opérateur booléen est une fonction qui prend une ou plusieurs variables booléennes et dont le résultat est un booléen. On décrit un opérateur booléen à l'aide d'une *table logique*.

IV.2.1 L'opérateur `not`

C'est l'opérateur de la négation [7] (non) : vous dite « vrai », il répond « faux » et inversement. C'est un des rares opérateurs qui ne prend qu'un seul argument.

x	`not`(x)
0	1
1	0

IV.2.2 L'opérateur `and`

L'opérateur `and` est l'opérateur logique « et » [7] : pour qu'il réponde vrai, il faut que l'un *et* l'autre des deux arguments qui lui sont passés soient vrais.

x	y	x `and` y
0	0	0
0	1	0
1	0	0
1	1	1

7. Avec la convention que 0 représente « faux » et 1 représente « vrai ».

IV.2.3 L'opérateur `or`

L'opérateur `or` est l'opérateur logique « ou » [7] : pour qu'il réponde vrai, il faut que l'un *ou* l'autre des deux arguments qui lui sont passés soient vrais. Autrement dit, au moins un de ses argument doit être vrai pour qu'il réponde vrai.

x	y	x `or` y
0	0	0
0	1	1
1	0	1
1	1	1


Info

On dit que l'opérateur `or` est « inclusif » : il répond 1 dans le cas où les deux variables sont à 1. La version « exclusive », qui retourne 0 lorsque les deux variables sont à 1 est appelée « `xor` » (pour *eXclusive or* en anglais), elle fait l'objet de l'exercice 34.
Dans la phrase « Je viendrais s'il y a un bus ou un train. », le « ou » est de type inclusif (comme `or`) alors que dans la phrase « Tu dois choisir : aller à la mer ou aller à la montagne. » le ou est de type exclusif (comme `xor`).

IV.3 Expressions booléennes

Définition

Une expression booléenne est une formule faisant intervenir des variables booléennes et des opérateurs booléens.


Exemple

Par exemple, « `not`(x `and` y) » est une expression booléenne. Si $x = 0$ et $y = 1$, x `and` y est à 0 et `not`(0) = 1, donc l'expression retourne 1.


Exercice

Exercice 31

Évaluer l'expression booléenne « `not`(x `or` y) » lorsque $x = 0$ et $y = 0$.


Attention

Comme la multiplication et l'addition dans une expression mathématique, les opérateurs `not`, `and` et `or` n'ont pas la même priorité.
Par exemple, dans l'expression mathématique « $x \times y + z$ », on effectue d'abord la multiplication et ensuite l'addition (on dit que la multiplication est prioritaire sur l'addition).
De la même manière, dans l'expression « x `and` y `or` z », le `and` est prioritaire sur le `or`. D'autre part, le `not` est prioritaire sur le `and`. Voici donc la liste des trois opérateurs du plus prioritaire eu moins prioritaire : `not`, `and` et `or`.
D'une manière générale, lorsqu'on utilise des expressions booléennes, on rend plus claire pour le lecteur la priorité des opérateurs à l'aide de parenthèses. Par exemple, plutôt que d'écrire « x `and` y `or` z », on écrira plutôt « (x `and` y) `or` z ».


Exercice

Exercice 32

Évaluer l'expression booléenne « `not` x `or` y `and` z » lorsque $x = 1$, $y = 1$ et $z = 0$. Clarifier cette expression à l'aide de parenthèses.

Python utilise les mêmes règles de priorité que celles que nous venons de décrire. Ainsi,

```
True or False and False
```

```
True
```

```
(True or False) and False
```

```
False
```

```
not False and False
```

```
False
```

```
not (False and False)
```

```
True
```

Savoir-faire

Dresser la table logique d'une expression booléenne

Pour dresser la table logique d'une expression booléenne,

- on commence par compter le nombre de variables, notons le n ;
- on trace un tableau à $n+1$ colonnes (une de plus pour l'expression) et 2^n+1 lignes (une de plus pour le nom des colonnes) ;
- on rempli les valeurs des variables pour que toutes les combinaisons soient présentes ;
- on complète la colonne qui donne le résultat de l'expression pour chaque valeurs des variables.

Exemple

Dressons la table logique de l'expression « x `or` `not`(y) ». Il y a deux variables : x et y, il y aura donc 3 colonne et 5 lignes. Lorsque $x=0$, l'expression se résume à `not`(y). Lorsque $x=1$, l'expression est évaluée à 1.

x	y	x `or` `not`(y)
0	0	1
0	1	0
1	0	1
1	1	1

Exercice

Exercice 33

Donner la table logique de l'expression « `not`$(x$ `and` `not`$(y))$ ».

Exercice

Exercice 34 L'opérateur « ou exclusif », `xor`

L'opérateur « ou exclusif », noté `xor` retourne 1 si une et une seule des deux variables est à 1.

1) Donner la table logique de l'expression « x `xor` y ».
2) Est-il possible d'écrire l'opérateur `xor` à partir des trois opérateurs de base ?

Exercice

Exercice 35 `and`-free

Donner la table logique de l'expression « `not`(`not`(x) `or` `not`$(y))$ ». Que remarque-t-on ?

Exercice

Exercice 36 Sans `or`

Donner la table logique de l'expression « `not`(`not`(x) `and` `not`$(y))$ ». Que remarque-t-on ?

Exercice

Exercice 37 Le multiplexeur binaire

Le multiplexeur est un opérateur qui prend trois variables, x, y et z, on le note `mux`(x,y,z). Il permet de choisir entre les variables y et z en fonction de l'état de la variable x : si x est à 1, il retourne la valeur de y, sinon, il retourne celle de z.

1) Donner la table logique de l'expression « `mux`(x, y, z) ».
2) Donner la table de l'expression « (`not`(x) `and` y) `or` (x `and` z) ».
3) En déduire une écriture de l'opérateur `mux` à l'aide des trois opérateurs de base `not`, `and` et `or`.

Exercice

Exercice 38 Électricité et booléens

Quand deux interrupteurs sont en parallèle, la lumière s'allume quand l'un d'eux est fermé. Quand ils sont en série, la lumière s'allume quand les deux sont fermés. Quand ils sont en va-et-vient la lumière s'allume quand les deux sont fermés ou les deux sont ouverts. Donner la table logique de l'opérateur booléen dans chacun des trois cas puis exprimer ces trois opérateurs avec `not` et `or`.

Exercice

Exercice 39 Addition binaire

On s'intéresse à l'addition de deux entiers naturels, x et y, codés sur un seul bit. Le résultat obtenu est noté s, il est lui aussi codé sur un seul bit. On note r le résultat de la retenue, encore codée sur un bit. Compléter la table logique de l'addition binaire ci-dessous.

x	y	somme s	retenue r
0	0		
0	1		
1	0		
1	1		

Écrire s et r à l'aide des opérateurs de base.

Exercice

Exercice 40

Olivier est content si Xavier et Yann sont tous les deux là, mais sans Zinédine, ou si Zinédine est là soit avec Xavier, soit avec Yann.

1) Construire une table logique avec en entrée la présence de Xavier, Yann et Zinédine, et en sortie le booléen qui vaut 1 si Olivier est content.
2) Exprimer le choix de Olivier par une phrase plus simple.

python

En Python, on dit que l'évaluation des opérateurs booléen est paresseuse. Ceci signifie que Python n'ira pas continuer l'évaluation d'un `and` si le premier argument est à 0 (le résultat sera 0) ou d'un `or` si le premier argument est à 1 (le résultat sera 1).
Pour s'en convaincre, on peut utiliser le code suivant [8] :

```
def vrai():
    print("Je suis une fonction qui retourne toujours True")
    return True

def faux():
    print("Je suis une fonction qui retourne toujours False")
    return False
```

```
faux() and vrai()
```

```
Je suis une fonction qui retourne toujours False
False
```

```
vrai() or faux()
```

```
Je suis une fonction qui retourne toujours True
True
```

On peut aussi vérifier que parfois, Python a besoin de tester tous les arguments :

```
vrai() and faux()
```

```
Je suis une fonction qui retourne toujours True
Je suis une fonction qui retourne toujours False
False
```

Exercice

Exercice 41

Quel sera l'affichage si l'on demande à l'interpréteur d'évaluer l'expression Python « `(vrai() or faux()) or vrai()` » ?

python

En Python, il est possible de manipuler des booléens par paquets. Pour cela, on représente le paquet de booléens par un paquet de bits, c'est-à-dire un entier ! On dit alors que l'on applique des opérateurs booléens en bit-à-bit. Le `not` bit-à-bit se note `~` (tilde), tandis que le `and` et le `or` se notent respectivement `&` (esperluette) et `|` (barre verticale ou *pipe*).

```
9 & 10
```

```
8
```

En effet, puisque $9 = 1001_2$ et $10 = 1010_2$, un `and` bit-à-bit donne la valeur $1000_2 = 8$.

```
~5
```

```
-6
```

Sur 4 bits par exemple, $5 = 0101_2$, la négation bit-à-bit donne 1010_2 qui n'est autre que l'écriture en complément à 2 de -6 (voir la figure 1).

Exercice

Exercice 42

Comment Python évalue la commande « `9 | 10` » ?

Exercice

Exercice 43 (difficile)

Expliquer pourquoi l'expression Python « `x + (~x + 1)` » retourne toujours 0 quelque-soit la valeur de l'entier `x`.

Info

En pratique, il est très rare d'avoir à utiliser les opérateurs bit-à-bit. C'est probablement la dernière fois que nous en parlerons.

8. Les fonctions créées ici ne font pas que renvoyer un résultat, elles modifient l'état d'un élément externe (ici l'affichage). On dit qu'elles ont un effet de bord.