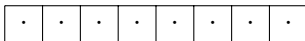


Représentation des nombres

Jean-Pierre Becirspahic
Lycée Louis-Le-Grand

Représentation en mémoire

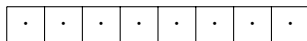
La mémoire des ordinateurs est divisées en blocs de 8 bits (soit un octet) :



Chaque bit peut prendre la valeur 0 ou 1.

Représentation en mémoire

La mémoire des ordinateurs est divisée en blocs de 8 bits (soit un octet) :



Chaque bit peut prendre la valeur 0 ou 1.

Un processeur 64 bits manipule des paquets de 8 octets, soit 64 bits :



On dispose donc de 64 bits pour représenter un nombre, ce qui explique pourquoi nous allons utiliser la décomposition de ceux-ci en base 2.

Représentation dans une base

Pour représenter un nombre n en base 10, on doit utiliser 10 caractères différents pour représenter les 10 premiers entiers : **0 1 2 3 4 5 6 7 8 9**, et décomposer les entiers suivants à l'aide des puissances de 10 successives.

Par exemple, 2985 représente le nombre $2 \times 10^3 + 9 \times 10^2 + 8 \times 10 + 5$.

Représentation dans une base

Pour représenter un nombre n en base 10, on doit utiliser 10 caractères différents pour représenter les 10 premiers entiers : **0 1 2 3 4 5 6 7 8 9**, et décomposer les entiers suivants à l'aide des puissances de 10 successives.

Par exemple, 2985 représente le nombre $2 \times 10^3 + 9 \times 10^2 + 8 \times 10 + 5$.

Pour représenter un nombre n en base b , on doit utiliser b caractères différents pour représenter les b premiers entiers et décomposer les entiers suivants à l'aide des puissances de b successives.

Représentation dans une base

Pour représenter un nombre n en base 10, on doit utiliser 10 caractères différents pour représenter les 10 premiers entiers : **0 1 2 3 4 5 6 7 8 9**, et décomposer les entiers suivants à l'aide des puissances de 10 successives.

Par exemple, 2985 représente le nombre $2 \times 10^3 + 9 \times 10^2 + 8 \times 10 + 5$.

Pour représenter un nombre n en base b , on doit utiliser b caractères différents pour représenter les b premiers entiers et décomposer les entiers suivants à l'aide des puissances de b successives.

Lorsque $b \leq 10$ on utilise les caractères $0 1 2 \dots b - 1$.

Par exemple, en base 3, $(210122)_3$ représente l'entier :

$$2 \times 3^5 + 1 \times 3^4 + 0 \times 3^3 + 1 \times 3^2 + 2 \times 3 + 2 = 584$$

Représentation dans une base

Pour représenter un nombre n en base 10, on doit utiliser 10 caractères différents pour représenter les 10 premiers entiers : **0 1 2 3 4 5 6 7 8 9**, et décomposer les entiers suivants à l'aide des puissances de 10 successives.

Par exemple, 2985 représente le nombre $2 \times 10^3 + 9 \times 10^2 + 8 \times 10 + 5$.

Pour représenter un nombre n en base b , on doit utiliser b caractères différents pour représenter les b premiers entiers et décomposer les entiers suivants à l'aide des puissances de b successives.

Lorsque $b \leq 10$ on utilise les caractères $0 1 2 \dots b - 1$.

Par exemple, en base 3, $(210122)_3$ représente l'entier :

$$2 \times 3^5 + 1 \times 3^4 + 0 \times 3^3 + 1 \times 3^2 + 2 \times 3 + 2 = 584$$

Mais en base 4, $(210122)_4$ représente l'entier :

$$2 \times 4^5 + 1 \times 4^4 + 0 \times 4^3 + 1 \times 4^2 + 2 \times 4 + 2 = 2330$$

Représentation dans une base

Pour représenter un nombre n en base 10, on doit utiliser 10 caractères différents pour représenter les 10 premiers entiers : **0 1 2 3 4 5 6 7 8 9**, et décomposer les entiers suivants à l'aide des puissances de 10 successives.

Par exemple, 2985 représente le nombre $2 \times 10^3 + 9 \times 10^2 + 8 \times 10 + 5$.

Pour représenter un nombre n en base b , on doit utiliser b caractères différents pour représenter les b premiers entiers et décomposer les entiers suivants à l'aide des puissances de b successives.

Lorsque $b > 10$ il faut rajouter de nouveaux caractères. Par exemple, en base 16 les 16 premiers entiers sont notés :

0 1 2 3 4 5 6 7 8 9 a b c d e f

$(a)_{16} = 10$, $(b)_{16} = 11$, $(c)_{16} = 12$, $(d)_{16} = 13$, $(e)_{16} = 14$ et $(f)_{16} = 15$.

Par exemple, $(5ac2)_{16} = 5 \times 16^3 + 10 \times 16^2 + 12 \times 16 + 2 = 23234$.

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\
 \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 \\
 \hline
 1 \\
 \hline
 1
 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 \\
 \hline
 1 \\
 \hline
 1
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r}
 \\
 + \\
 \hline

 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\
 \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 \\
 \hline
 1 \\
 \hline
 1
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r}
 \\
 + \\
 \hline
 1
 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 \\
 \hline
 1 \\
 \hline
 1
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r}
 \\
 - \\
 \hline

 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \\
 \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 1 \\
 \hline
 1
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r}
 \\
 - \\
 \hline

 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 \\
 \hline
 1 \\
 \hline
 1
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r}
 \\
 \times \\
 \hline

 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\
 \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 \\
 \hline
 1 \\
 \hline
 1
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r}
 \\
 \times \\
 \hline
 1
 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \overset{1}{1} \overset{1}{0} \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\
 + \\
 \hline
 1
 \end{array}$$

← retenues

$$\begin{array}{r}
 \\
 \times \\
 \hline
 \\
 \\
 \hline
 1 \\
 \hline
 1
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r}
 1 \\
 \hline
 1
 \end{array}$$

Écriture en base 2

Dans le cas particulier de la base 2, seuls les chiffres 0 et 1 sont utilisés, et les algorithmes de calcul appris à l'école primaire (addition, soustraction, multiplication, division) se généralisent en base 2.

$$\begin{array}{r}
 \color{red}1 \ \color{red}1 \ \color{red}1 \ \color{red}1 \quad \color{red}1 \ \color{red}1 \ \leftarrow \text{retenues} \\
 \\
 + \\
 \hline
 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0
 \end{array}$$

$$\begin{array}{r}
 \ 0 \ 1 \ 0 \\
 \times \ 1 \ 0 \ 1 \\
 \hline
 \ 0 \ 1 \ 0 \\
 \ 0 \ 1 \ 0 \\
 \hline
 1 \ 0 \ 1 \ 0 \cdot \cdot \\
 \hline
 1 \ 1 \ 0 \ 0 \ 1 \ 0
 \end{array}$$

Réaliser en base 2 l'opération suivante :

$$\begin{array}{r|l}
 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 & 1 \ 0 \ 1 \ 1 \\
 \hline
 & \color{red}1 \ 0 \ 0 \ 1 \\
 & \color{red}1 \ 0
 \end{array}$$

Changement de base

Pour convertir le nombre $x = (a_p a_{p-1} \cdots a_1 a_0)_b$ en base 10, il suffit d'ap-

pliquer la formule $x = \sum_{k=0}^p a_k b^k$.

Version naïve (avec $b \leq 10$):

```
def base10(x, b=2):  
    s = 0  
    k = len(x) - 1  
    for a in x:  
        s += int(a) * b ** k  
        k -= 1  
    return s
```

```
In [1]: base10('210122', b=3)  
Out[1]: 584
```

```
In [2]: base10('1011011')  
Out[2]: 91
```

Changement de base

Pour convertir le nombre $x = (a_p a_{p-1} \cdots a_1 a_0)_b$ en base 10, il suffit d'ap-

pliquer la formule $x = \sum_{k=0}^p a_k b^k$.

Méthode de HORNER (avec $b \leq 10$) :

```
def base10(x, b=2):  
    u = 0  
    for a in x:  
        u = b * u + int(a)  
    return
```

Cette méthode utilise l'invariant : «à l'entrée de la k^e boucle la variable u référence $(a_p a_{p-1} \cdots a_{p+2-k})_b$ ».

Changement de base

À l'inverse, pour convertir un entier en base b , il faut observer que si $x = (a_p a_{p-1} \cdots a_1 a_0)_b$ alors le quotient de la division euclidienne de x par b est égal à $(a_p a_{p-1} \cdots a_1)_b$ et le reste à a_0 .

Exemple en base 10 :

$$584 // 10 = 58 \quad 584 \% 10 = 4$$

Changement de base

À l'inverse, pour convertir un entier en base b , il faut observer que si $x = (a_p a_{p-1} \cdots a_1 a_0)_b$ alors le quotient de la division euclidienne de x par b est égal à $(a_p a_{p-1} \cdots a_1)_b$ et le reste à a_0 .

Exemple en base 10 :

$$584 // 10 = 58 \quad 584 \% 10 = 4$$

```
def baseb(x, b):  
    s = ""  
    y = x  
    while y > 0:  
        s = str(y % b) + s  
        y /= b  
    return(s)
```

Changement de base

À l'inverse, pour convertir un entier en base b , il faut observer que si $x = (a_p a_{p-1} \cdots a_1 a_0)_b$ alors le quotient de la division euclidienne de x par b est égal à $(a_p a_{p-1} \cdots a_1)_b$ et le reste à a_0 .

Exemple en base 10 :

$$584 // 10 = 58 \quad 584 \% 10 = 4$$

En base 3 :

$$584 = 3 \times 194 + 2$$

$$194 = 3 \times 64 + 2$$

$$64 = 3 \times 21 + 1$$

$$21 = 3 \times 7 + 0$$

$$7 = 3 \times 2 + 1$$

$$2 = 3 \times 0 + 2$$

En base 2 :

$$91 = 2 \times 45 + 1$$

$$45 = 2 \times 22 + 1$$

$$22 = 2 \times 11 + 0$$

$$11 = 2 \times 5 + 1$$

$$5 = 2 \times 2 + 1$$

$$2 = 2 \times 1 + 0$$

$$1 = 2 \times 0 + 1$$

```
In [3]: baseb(584, b=3)
Out[3]: '210122'
```

```
In [4]: baseb(91)
Out[4]: '1011011'
```

Base 2 et base 16

Il est possible d'introduire directement au clavier un nombre écrit en base 2 ; il suffit de le faire précéder de 0b ou en base 16 (précédé de 0x) :

```
In [5]: 0b1011011
```

```
Out[5]: 91
```

```
In [6]: 0xa27c
```

```
Out[6]: 41596
```

Base 2 et base 16

Il est possible d'introduire directement au clavier un nombre écrit en base 2 ; il suffit de le faire précéder de 0b ou en base 16 (précédé de 0x) :

```
In [5]: 0b1011011
```

```
Out[5]: 91
```

```
In [6]: 0xa27c
```

```
Out[6]: 41596
```

Sachant que $2^4 = 16$, tout nombre écrit en base 2 à l'aide de 4 chiffres s'écrit en base 16 à l'aide d'un seul chiffre :

$(0000)_2 = (0)_{16}$	$(0100)_2 = (4)_{16}$	$(1000)_2 = (8)_{16}$	$(1100)_2 = (c)_{16}$
$(0001)_2 = (1)_{16}$	$(0101)_2 = (5)_{16}$	$(1001)_2 = (9)_{16}$	$(1101)_2 = (d)_{16}$
$(0010)_2 = (2)_{16}$	$(0110)_2 = (6)_{16}$	$(1010)_2 = (a)_{16}$	$(1110)_2 = (e)_{16}$
$(0011)_2 = (3)_{16}$	$(0111)_2 = (7)_{16}$	$(1011)_2 = (b)_{16}$	$(1111)_2 = (f)_{16}$

Base 2 et base 16

Il est possible d'introduire directement au clavier un nombre écrit en base 2 ; il suffit de le faire précéder de 0b ou en base 16 (précédé de 0x) :

```
In [5]: 0b1011011
```

```
Out[5]: 91
```

```
In [6]: 0xa27c
```

```
Out[6]: 41596
```

Sachant que $2^4 = 16$, tout nombre écrit en base 2 à l'aide de 4 chiffres s'écrit en base 16 à l'aide d'un seul chiffre :

$$\begin{array}{llll}
 (0000)_2 = (0)_{16} & (0100)_2 = (4)_{16} & (1000)_2 = (8)_{16} & (1100)_2 = (c)_{16} \\
 (0001)_2 = (1)_{16} & (0101)_2 = (5)_{16} & (1001)_2 = (9)_{16} & (1101)_2 = (d)_{16} \\
 (0010)_2 = (2)_{16} & (0110)_2 = (6)_{16} & (1010)_2 = (a)_{16} & (1110)_2 = (e)_{16} \\
 (0011)_2 = (3)_{16} & (0111)_2 = (7)_{16} & (1011)_2 = (b)_{16} & (1111)_2 = (f)_{16}
 \end{array}$$

Pour convertir un nombre de la base 2 à la base 16, on regroupe les chiffres qui le composent par paquets de 4. Par exemple, $(1011\ 0110\ 1110\ 1001)_2 = (b6e9)_{16}$.

Base 2 et base 16




Il est possible d'introduire directement au clavier un nombre écrit en base 2 ; il suffit de le faire précéder de 0b ou en base 16 (précédé de 0x) :

```
In [5]: 0b1011011
Out[5]: 91
```

```
In [6]: 0xa27c
Out[6]: 41596
```

Intérêt de la base 16 : un octet = 8 bits = deux caractères hexadécimaux.

Par exemple, une couleur web est définie par trois octets représentant ses composantes RVB. Un navigateur interprète le code couleur :

- $(ffa500)_{16}$ comme du orange ($R = 255, V = 165, B = 0$) 
- $(00ff00)_{16}$ comme du vert ($R = 0, V = 255, B = 0$) 
- $(ee82ee)_{16}$ comme du violet ($R = 238, V = 130, B = 238$) 

$256^3 = 16\,777\,216$ couleurs différentes sont donc potentiellement accessibles.

Base 2 et base 16

Il est possible d'introduire directement au clavier un nombre écrit en base 2 ; il suffit de le faire précéder de `0b` ou en base 16 (précédé de `0x`) :

```
In [5]: 0b1011011
Out[5]: 91
```

```
In [6]: 0xa27c
Out[6]: 41596
```

En PYTHON, les fonctions `bin` et `hex` permettent de convertir un entier décimal en un nombre binaire ou hexadécimal :

```
In [7]: bin(41397)
Out[7]: '0b1010000110110101'
```

```
In [8]: hex(41397)
Out[8]: '0xa1b5'
```

Représentation des entiers naturels

Dans un ordinateur, un nombre est représenté par une succession de bits. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fini.

Représentation des entiers naturels

Dans un ordinateur, un nombre est représenté par une succession de bits. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fini.

Les **entiers naturels** sont essentiellement utilisés pour représenter les adresses en mémoire. Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et $2^n - 1$.

Représentation des entiers naturels

Dans un ordinateur, un nombre est représenté par une succession de bits. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fini.

Les **entiers naturels** sont essentiellement utilisés pour représenter les adresses en mémoire. Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et $2^n - 1$.

- un octet code les entiers entre $(00)_{16} = 0$ et $(ff)_{16} = 255$;

Représentation des entiers naturels

Dans un ordinateur, un nombre est représenté par une succession de bits. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fini.

Les **entiers naturels** sont essentiellement utilisés pour représenter les adresses en mémoire. Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et $2^n - 1$.

- un octet code les entiers entre $(00)_{16} = 0$ et $(ff)_{16} = 255$;
- 32 bits (soit 4 octets) les nombres entre $(0000\ 0000)_{16} = 0$ et $(ffff\ ffff)_{16} = 2^{32} - 1 = 4\ 294\ 967\ 295$;

Représentation des entiers naturels

Dans un ordinateur, un nombre est représenté par une succession de bits. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fini.

Les **entiers naturels** sont essentiellement utilisés pour représenter les adresses en mémoire. Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et $2^n - 1$.

- un octet code les entiers entre $(00)_{16} = 0$ et $(ff)_{16} = 255$;
- 32 bits (soit 4 octets) les nombres entre $(0000\ 0000)_{16} = 0$ et $(ffff\ ffff)_{16} = 2^{32} - 1 = 4\ 294\ 967\ 295$;
- 64 bits les nombres entre 0 et $2^{64} - 1 = 18\ 446\ 744\ 073\ 709\ 551\ 615$.

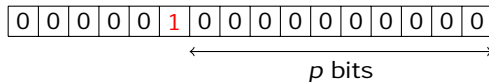
Représentation des entiers naturels

Dans un ordinateur, un nombre est représenté par une succession de bits. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fini.

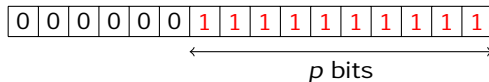
Les **entiers naturels** sont essentiellement utilisés pour représenter les adresses en mémoire. Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et $2^n - 1$.

Deux valeurs à retenir :

- 2^p est représenté par



- $2^p - 1$ est représenté par



Représentation des entiers relatifs

Codage par complément à deux

Pour coder un entier relatif sur n bits, on réserve le premier bit pour coder le signe (0 pour les nombres positifs et 1 pour les nombres négatifs).

Entiers positifs :

0															
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

 représentation binaire de x

Les entiers positifs représentables vérifient $0 \leq x \leq 2^{n-1} - 1$.


Représentation des entiers relatifs

Codage par complément à deux

Pour coder un entier relatif sur n bits, on réserve le premier bit pour coder le signe (0 pour les nombres positifs et 1 pour les nombres négatifs).

Entiers positifs :


0																		
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--


 représentation binaire de x

Les entiers positifs représentables vérifient $0 \leq x \leq 2^{n-1} - 1$.

Entiers négatifs :

1																		
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--


 représentation binaire de $2^n + x$

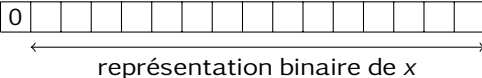
Les entiers négatifs représentables vérifient :

$$2^{n-1} \leq 2^n + x \leq 2^n - 1 \iff -2^{n-1} \leq x \leq -1.$$

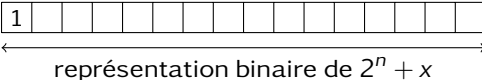
Représentation des entiers relatifs

Codage par complément à deux

Pour coder un entier relatif sur n bits, on réserve le premier bit pour coder le signe (0 pour les nombres positifs et 1 pour les nombres négatifs).

Entiers positifs : 

Les entiers positifs représentables vérifient $0 \leq x \leq 2^{n-1} - 1$.

Entiers négatifs : 

Les entiers négatifs représentables vérifient :

$$2^{n-1} \leq 2^n + x \leq 2^n - 1 \iff -2^{n-1} \leq x \leq -1.$$

Exemple dans le cas d'une configuration sur 8 bits.

105 est représenté par l'octet 01101001 car :

$$105 = 2^6 + 2^5 + 2^3 + 2^0 = (1101001)_2$$

Représentation des entiers relatifs

Codage par complément à deux

Pour coder un entier relatif sur n bits, on réserve le premier bit pour coder le signe (0 pour les nombres positifs et 1 pour les nombres négatifs).

Entiers positifs :

0																
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

 représentation binaire de x

Les entiers positifs représentables vérifient $0 \leq x \leq 2^{n-1} - 1$.

Entiers négatifs :

1																
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

 représentation binaire de $2^n + x$

Les entiers négatifs représentables vérifient :

$$2^{n-1} \leq 2^n + x \leq 2^n - 1 \iff -2^{n-1} \leq x \leq -1.$$

Exemple dans le cas d'une configuration sur 8 bits.

-105 est représenté par l'octet 10010111 car :

$$256 - 105 = 151 = 2^7 + 2^4 + 2^2 + 2^1 + 2^0 = (10010111)_2$$

Représentation des entiers relatifs

Codage par complément à deux

Pour coder un entier relatif sur n bits, on réserve le premier bit pour coder le signe (0 pour les nombres positifs et 1 pour les nombres négatifs).

Entiers positifs :

0														
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--

↔
 représentation binaire de x

Les entiers positifs représentables vérifient $0 \leq x \leq 2^{n-1} - 1$.

Entiers négatifs :

1														
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--

↔
 représentation binaire de $2^n + x$

Les entiers négatifs représentables vérifient :

$$2^{n-1} \leq 2^n + x \leq 2^n - 1 \iff -2^{n-1} \leq x \leq -1.$$

Exercice. Dans une représentation en complément à deux sur 8 bits, quels sont les entiers relatifs représentés par :

- 01101101 ?
- 10010010 ?

Représentation des entiers relatifs

Codage par complément à deux

Pour coder un entier relatif sur n bits, on réserve le premier bit pour coder le signe (0 pour les nombres positifs et 1 pour les nombres négatifs).

Entiers positifs :

0														
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--

 représentation binaire de x

Les entiers positifs représentables vérifient $0 \leq x \leq 2^{n-1} - 1$.

Entiers négatifs :

1														
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--

 représentation binaire de $2^n + x$

Les entiers négatifs représentables vérifient :

$$2^{n-1} \leq 2^n + x \leq 2^n - 1 \iff -2^{n-1} \leq x \leq -1.$$

Exercice. Dans une représentation en complément à deux sur 8 bits, quels sont les entiers relatifs représentés par :

- 01101101? \longrightarrow 109
- 10010010? \longrightarrow -110

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Exemple : calcul de la somme de -91 et de 113 avec un codage sur 8 bits.

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Exemple : calcul de la somme de -91 et de 113 avec un codage sur 8 bits.

$2^8 = 256$ donc -91 est représenté par son complément à deux :

$$256 - 91 = 165 = (10100101)_2$$

113 est positif donc représenté par sa décomposition en base 2 :

$$113 = (01110001)_2$$

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Exemple : calcul de la somme de -91 et de 113 avec un codage sur 8 bits.

$2^8 = 256$ donc -91 est représenté par son complément à deux :

$$256 - 91 = 165 = (10100101)_2$$

113 est positif donc représenté par sa décomposition en base 2 :

$$113 = (01110001)_2$$

On additionne ces deux représentations :

$$\begin{array}{r}
 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1 \\
 +\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1 \\
 \hline
 (1)\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0
 \end{array}$$

On garde les 8 derniers bits : 00010110 ; le premier bit est un 0 donc le résultat est positif ; il représente l'entier $(10110)_2 = 22$.

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Calcul de $a + b$: on suppose que $a + b$ est représentable.

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif);
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Calcul de $a + b$: on suppose que $a + b$ est représentable.

- si $a > 0$ et $b > 0$ on calcule $(a + b)$;

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif);
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Calcul de $a + b$: on suppose que $a + b$ est représentable.

- si $a > 0$ et $b > 0$ on calcule $(a + b)$;
- si $a > 0$ et $b < 0$ on calcule $a + (2^n + b) = (a + b) + 2^n$;

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Calcul de $a + b$: on suppose que $a + b$ est représentable.

- si $a > 0$ et $b > 0$ on calcule $(a + b)$;
- si $a > 0$ et $b < 0$ on calcule $a + (2^n + b) = (a + b) + 2^n$;
 - si $a + b \geq 0$ ce nombre s'écrit sur $n + 1$ bit donc est tronqué : on obtient la représentation de $a + b$;

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Calcul de $a + b$: on suppose que $a + b$ est représentable.

- si $a > 0$ et $b > 0$ on calcule $(a + b)$;
- si $a > 0$ et $b < 0$ on calcule $a + (2^n + b) = (a + b) + 2^n$;
 - si $a + b \geq 0$ ce nombre s'écrit sur $n + 1$ bit donc est tronqué : on obtient la représentation de $a + b$;
 - si $a + b < 0$ ce nombre n'est pas tronqué : on obtient la représentation de $a + b$;

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif) ;
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Calcul de $a + b$: on suppose que $a + b$ est représentable.

- si $a > 0$ et $b > 0$ on calcule $(a + b)$;
- si $a > 0$ et $b < 0$ on calcule $a + (2^n + b) = (a + b) + 2^n$;
 - si $a + b \geq 0$ ce nombre s'écrit sur $n + 1$ bit donc est tronqué : on obtient la représentation de $a + b$;
 - si $a + b < 0$ ce nombre n'est pas tronqué : on obtient la représentation de $a + b$;
- si $a < 0$ et $b > 0$ la situation est identique au cas précédent ;

Représentation des entiers relatifs

Intérêt du complément à deux

- le signe d'un entier se reconnaît par son premier bit (0 pour un nombre positif, 1 pour un nombre négatif);
- unicité de la représentation de nombre compris entre -2^{n-1} et $2^{n-1} - 1$;
- l'addition des entiers relatifs en complément à deux utilise le même algorithme que pour les entiers naturels.

Calcul de $a + b$: on suppose que $a + b$ est représentable.

- si $a > 0$ et $b > 0$ on calcule $(a + b)$;
- si $a > 0$ et $b < 0$ on calcule $a + (2^n + b) = (a + b) + 2^n$;
 - si $a + b \geq 0$ ce nombre s'écrit sur $n + 1$ bit donc est tronqué : on obtient la représentation de $a + b$;
 - si $a + b < 0$ ce nombre n'est pas tronqué : on obtient la représentation de $a + b$;
- si $a < 0$ et $b > 0$ la situation est identique au cas précédent ;
- si $a < 0$ et $b < 0$ on calcule $(2^n + a) + (2^n + b) = 2^n + (2^n + a + b)$; ce nombre s'écrit sur $n + 1$ bits donc est tronqué : il reste $2^n + a + b$ qui est la représentation de $a + b$.

Soustraction de deux entiers relatifs

Calculer $a - b$ c'est additionner a et $-b$; il suffit de savoir calculer l'opposé avec la représentation en complément à deux.

Soustraction de deux entiers relatifs

Calculer $a - b$ c'est additionner a et $-b$; il suffit de savoir calculer l'opposé avec la représentation en complément à deux.

- si $x \geq 0$ il est représenté par l'entier naturel x et son opposé $-x$ par l'entier naturel $2^n - x$.

Notons $x = (0x_{n-2} \cdots x_1 x_0)_2$ et posons $y = (1y_{n-2} \cdots y_1 y_0)_2$ en convenant que $y_i = 1 - x_i$. Alors $x + y = (11 \cdots 11)_2 = 2^n - 1$ donc $2^n - x = y + 1$.

Soustraction de deux entiers relatifs

Calculer $a - b$ c'est additionner a et $-b$; il suffit de savoir calculer l'opposé avec la représentation en complément à deux.

- si $x \geq 0$ il est représenté par l'entier naturel x et son opposé $-x$ par l'entier naturel $2^n - x$.

Notons $x = (0x_{n-2} \cdots x_1 x_0)_2$ et posons $y = (1y_{n-2} \cdots y_1 y_0)_2$ en convenant que $y_i = 1 - x_i$. Alors $x + y = (11 \cdots 11)_2 = 2^n - 1$ donc $2^n - x = y + 1$.

- si $x < 0$ il est représenté par l'entier naturel $2^n + x$ et son opposé $-x$ par l'entier naturel $-x$.

Notons $2^n + x = (1x_{n-2} \cdots x_1 x_0)_2$ et posons $y = (0y_{n-2} \cdots y_1 y_0)_2$ en convenant que $y_i = 1 - x_i$. Alors $2^n + x + y = (11 \cdots 11)_2 = 2^n - 1$ donc $-x = y + 1$.

Soustraction de deux entiers relatifs

Calculer $a - b$ c'est additionner a et $-b$; il suffit de savoir calculer l'opposé avec la représentation en complément à deux.

Bilan : on obtient la représentation de l'opposé de x en :

- 1 en remplaçant tous les bits égaux à 0 par des 1 et réciproquement dans la représentation de x ;
- 2 additionnant 1 au résultat obtenu.

Soustraction de deux entiers relatifs

Calculer $a - b$ c'est additionner a et $-b$; il suffit de savoir calculer l'opposé avec la représentation en complément à deux.

Bilan : on obtient la représentation de l'opposé de x en :

- 1 en remplaçant tous les bits égaux à 0 par des 1 et réciproquement dans la représentation de x ;
- 2 additionnant 1 au résultat obtenu.

Exemple avec un codage sur 8 bits.

- $x = 113$ est représenté par 01110001 ; on a donc $y = 10001110$ et $-x$ est représenté par $y + 1 = 10001111$.
En effet, $256 - 113 = 143 = (10001111)_2$.

Soustraction de deux entiers relatifs

Calculer $a - b$ c'est additionner a et $-b$; il suffit de savoir calculer l'opposé avec la représentation en complément à deux.

Bilan : on obtient la représentation de l'opposé de x en :

- 1 en remplaçant tous les bits égaux à 0 par des 1 et réciproquement dans la représentation de x ;
- 2 additionnant 1 au résultat obtenu.

Exemple avec un codage sur 8 bits.

- $x = 113$ est représenté par 01110001 ; on a donc $y = 10001110$ et $-x$ est représenté par $y + 1 = 10001111$.
En effet, $256 - 113 = 143 = (10001111)_2$.
- $x = -91$ est représenté par 10100101 ; on a donc $y = 01011010$ et $-x$ est représenté par $y + 1 = 01011011$.
En effet, $91 = (1011011)_2$.

Représentation des entiers relatifs

Dépassement de capacité

Si un processeur alloue n bits à la représentation des entiers relatifs, les entiers représentables appartiennent à l'intervalle $\llbracket -2^{n-1}, 2^{n-1} - 1 \rrbracket$. Le résultat d'une opération sur ces entiers peut ne pas appartenir à cet intervalle. On parle alors de **dépassement de capacité**.

Représentation des entiers relatifs

Dépassement de capacité

Si un processeur alloue n bits à la représentation des entiers relatifs, les entiers représentables appartiennent à l'intervalle $\llbracket -2^{n-1}, 2^{n-1} - 1 \rrbracket$.

Le résultat d'une opération sur ces entiers peut ne pas appartenir à cet intervalle. On parle alors de **dépassement de capacité**.

Exemple : addition de 72 et de 55 avec un codage sur 8 bits.

$72 = (1001000)_2$ est représenté par 01001000.

$59 = (111011)_2$ est représenté par 00111011.

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

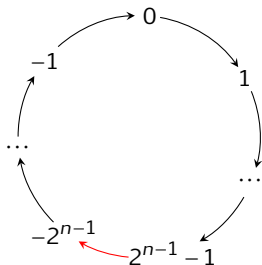
On obtient le nombre négatif représenté par 10000011 c'est à dire -125 , alors que $72 + 59 = 131$. Ce n'est pas anormal puisque sur 8 bits les seuls entiers relatifs représentables sont compris entre -128 et $+127$, ce qui n'est pas le cas de 131.

Représentation des entiers relatifs

Dépassement de capacité

Si un processeur alloue n bits à la représentation des entiers relatifs, les entiers représentables appartiennent à l'intervalle $\llbracket -2^{n-1}, 2^{n-1} - 1 \rrbracket$. Le résultat d'une opération sur ces entiers peut ne pas appartenir à cet intervalle. On parle alors de **dépassement de capacité**.

Dans une représentation en complément à deux sur n bits, le successeur de $2^{n-1} - 1$ est égal à -2^{n-1} .



nombre	représentation
0	0000.....0000
1	0000.....0001
$2^{n-1} - 1$	0111.....1111
-2^{n-1}	1000.....0000
-1	1111.....1111

Représentation des entiers relatifs

Exemples en CAML et PYTHON

En CAML les entiers sont codés en complément à deux sur 63 bits.

$$2^{62} - 1 = 4611686018427387903$$

$$-2^{62} = -4611686018427387904$$

```
# max_int ;;  
- : int = 4611686018427387903  
  
# min_int ;;  
- : int = -4611686018427387904  
  
# max_int + 1 ;;  
- : int = -4611686018427387904
```

Représentation des entiers relatifs

Exemples en CAML et PYTHON

En PYTHON les entiers sont codés sur 64 bits.

$$\begin{aligned}2^{63} - 1 &= 9223372036854775807 \\ &= (7\text{ffffffffffffffff})_{16}\end{aligned}$$

```
In [2]: x = 0x7fffffffffffffffff
```

```
In [3]: x
```

```
Out[3]: 9223372036854775807
```

```
In [4]: x + 1
```

```
Out[4]: 9223372036854775808
```

Lorsque le résultat d'un calcul dépasse le plus grand entier représentable en complément à deux (à savoir $2^{63} - 1$) sa représentation change pour devenir un **entier long**.

Cette nouvelle représentation n'est pas limitée en taille. Cela présente des avantages, mais aussi des inconvénients (que nous ne détaillerons pas).

Nombres décimaux et dyadiques

Un nombre **décimal** s'écrit $\frac{x}{10^n}$ où $x \in \mathbb{Z}$.

Si $x = \pm(a_p a_{p-1} \cdots a_1 a_0)_{10}$, alors $\frac{x}{10^n} = \pm(a_p \cdots a_n , a_{n-1} \cdots a_0)_{10}$.

Nombres décimaux et dyadiques

Un nombre **décimal** s'écrit $\frac{x}{10^n}$ où $x \in \mathbb{Z}$.

Si $x = \pm(a_p a_{p-1} \cdots a_1 a_0)_{10}$, alors $\frac{x}{10^n} = \pm(a_p \cdots a_n , a_{n-1} \cdots a_0)_{10}$.

- $\frac{25}{4}$ est un nombre décimal car $\frac{25}{4} = \frac{625}{100} = 6,25$.

Nombres décimaux et dyadiques

Un nombre **décimal** s'écrit $\frac{x}{10^n}$ où $x \in \mathbb{Z}$.

Si $x = \pm(a_p a_{p-1} \cdots a_1 a_0)_{10}$, alors $\frac{x}{10^n} = \pm(a_p \cdots a_n , a_{n-1} \cdots a_0)_{10}$.

- $\frac{25}{4}$ est un nombre décimal car $\frac{25}{4} = \frac{625}{100} = 6,25$.
- $\frac{1}{3}$ n'est pas un nombre décimal car $\frac{1}{3} = 0,333333333\cdots$

Un nombre décimal possède un développement décimal **fini**.

Nombres décimaux et dyadiques

Un nombre **décimal** s'écrit $\frac{x}{10^n}$ où $x \in \mathbb{Z}$.

Si $x = \pm(a_p a_{p-1} \cdots a_1 a_0)_{10}$, alors $\frac{x}{10^n} = \pm(a_p \cdots a_n, a_{n-1} \cdots a_0)_{10}$.

- $\frac{25}{4}$ est un nombre décimal car $\frac{25}{4} = \frac{625}{100} = 6,25$.
- $\frac{1}{3}$ n'est pas un nombre décimal car $\frac{1}{3} = 0,333333333\cdots$

Un nombre décimal possède un développement décimal **fini**.

Un nombre **dyadique** s'écrit $\frac{x}{2^n}$ où $x \in \mathbb{Z}$.

Si $x = \pm(a_p a_{p-1} \cdots a_1 a_0)_2$, alors $\frac{x}{2^n} = \pm(a_p \cdots a_n, a_{n-1} \cdots a_0)_2$.

Par exemple, $\frac{25}{4}$ est un nombre dyadique et son développement dyadique

est $(110,01)_2$. En effet, $25 = (11001)_2$ donc $\frac{25}{4} = (110,01)_2$.

Nombres décimaux et dyadiques

Problème : un nombre décimal n'est pas forcément dyadique.

Conséquence :

- la conversion décimal (humain) \rightarrow dyadique (machine) peut causer une première approximation ;
- le calcul machine entre nombre dyadiques peut causer une seconde approximation.

```
In [1]: 0.1 + 0.2  
Out[1]: 0.30000000000000004
```

```
In [2]: (0.1 + 0.2) - 0.3  
Out[2]: 5.551115123125783e-17
```

Dans ces deux calculs la conversion décimal \rightarrow dyadique provoque une (faible) erreur ; en revanche la conversion dyadique \rightarrow décimal est exacte.

Nombres décimaux et dyadiques

Problème : un nombre décimal n'est pas forcément dyadique.

Conséquence :

- la conversion décimal (humain) \rightarrow dyadique (machine) peut causer une première approximation ;
- le calcul machine entre nombre dyadiques peut causer une seconde approximation.

```
In [1]: 0.1 + 0.2  
Out[1]: 0.30000000000000004
```

```
In [2]: (0.1 + 0.2) - 0.3  
Out[2]: 5.551115123125783e-17
```

Quand on utilise le type *float*, on utilise jamais l'égalité de valeur `==` ; on fera toujours intervenir une marge d'erreur (absolue ou relative).

```
In [3]: 0.1 + 0.2 == 0.3  
Out[3]: False
```

```
In [4]: abs(0.1 + 0.2 - 0.3) <= 1e-10  
Out[4]: True
```

La norme IEEE-754

Représentation normalisée d'un nombre dyadique x non nul :

$$x = \pm \underbrace{(1, b_1 \cdots b_k)_2}_{\text{mantisse}} \times 2^e \quad \text{avec } e \in \mathbb{Z} \text{ (exposant).}$$

Exemples : $6,25 = (110,01)_2 = (1,1001)_2 \times 2^2$
 $-0,375 = -(0,011)_2 = -(1,1)_2 \times 2^{-2}$

La norme IEEE-754

Représentation normalisée d'un nombre dyadique x non nul :

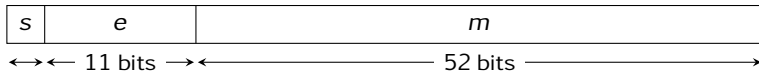
$$x = \pm(1, \underbrace{b_1 \cdots b_k}_m)_2 \times 2^e \quad \text{avec } e \in \mathbb{Z} \text{ (exposant).}$$

Exemples : $6,25 = (110,01)_2 = (1,1001)_2 \times 2^2$

$$-0,375 = -(0,011)_2 = -(1,1)_2 \times 2^{-2}$$

Pour la norme IEEE-754, les nombres dyadiques sont codés sur 64 bits en réservant :

- 1 bit pour le signe ;
- 11 bits pour l'exposant ;
- 52 bits pour la mantisse.



La norme IEEE-754

L'exposant e est codé suivant la technique du décalage : il est représenté en machine par l'entier naturel $e' = e + 2^{10} - 1$.

A priori,

$$0 \leq e' \leq 2^{11} - 1 \iff 1 - 2^{10} \leq e \leq 2^{10} \iff -1023 \leq e \leq 1024$$

mais les valeurs $e' = 0$ et $e' = 2^{11} - 1$ sont réservées à un usage particulier.

La norme IEEE-754

L'exposant e est codé suivant la technique du décalage : il est représenté en machine par l'entier naturel $e' = e + 2^{10} - 1$.

- Si $(00000000001)_2 \leq e' \leq (11111111110)_2$, soit $1 \leq e' \leq 2^{11} - 2$ alors $-1022 \leq e \leq 1023$.

La norme IEEE-754

L'exposant e est codé suivant la technique du décalage : il est représenté en machine par l'entier naturel $e' = e + 2^{10} - 1$.

- Si $(00000000001)_2 \leq e' \leq (11111111110)_2$, soit $1 \leq e' \leq 2^{11} - 2$ alors $-1022 \leq e \leq 1023$.

Si $x > 0$, alors :

$$(1, \underbrace{0000 \dots 0000}_{52 \text{ fois } 0})_2 \times 2^{-1022} \leq x \leq (1, \underbrace{1111 \dots 1111}_{52 \text{ fois } 1})_2 \times 2^{1023}$$

$$\iff 2^{-1022} \leq x \leq 2^{1024} - 2^{971}$$

$$\iff 2,23 \times 10^{-308} \leq x \leq 1,80 \times 10^{308}$$

La norme IEEE-754

L'exposant e est codé suivant la technique du décalage : il est représenté en machine par l'entier naturel $e' = e + 2^{10} - 1$.

- Si $e' = (0000000000)_2$, alors $e = -1023$ et x est représenté sous forme *dénormalisée* :

$$x = \pm(0, b_1 \cdots b_k)_2 \times 2^e$$

(pour manipuler des quantités encore plus petites).

La norme IEEE-754

L'exposant e est codé suivant la technique du décalage : il est représenté en machine par l'entier naturel $e' = e + 2^{10} - 1$.

- Si $e' = (0000000000)_2$, alors $e = -1023$ et x est représenté sous forme *dénormalisée* :

$$x = \pm(0, b_1 \cdots b_k)_2 \times 2^e$$

(pour manipuler des quantités encore plus petites).

Si $x > 0$ alors :

$$(0, \underbrace{0000 \cdots 0000 1}_{51 \text{ fois } 0})_2 \times 2^{-1023} \leq x \leq (0, \underbrace{1111 \cdots 1111}_{52 \text{ fois } 1})_2 \times 2^{-1022}$$

$$\iff 2^{-1074} \leq x \leq 2^{-1022} - 2^{-1074}$$

$$\iff 4,94 \times 10^{-324} \leq x \leq 2,23 \times 10^{-308}$$

La norme IEEE-754

L'exposant e est codé suivant la technique du décalage : il est représenté en machine par l'entier naturel $e' = e + 2^{10} - 1$.

- Si $e' = (0000000000)_2$, alors $e = -1023$ et x est représenté sous forme *dénormalisée* :

$$x = \pm(0, b_1 \cdots b_k)_2 \times 2^e$$

(pour manipuler des quantités encore plus petites).

Si $x > 0$ alors :

$$(0, \underbrace{0000 \cdots 0000}_{{51 \text{ fois } 0}} 1)_2 \times 2^{-1023} \leq x \leq (0, \underbrace{1111 \cdots 1111}_{{52 \text{ fois } 1}})_2 \times 2^{-1022}$$

$$\iff 2^{-1074} \leq x \leq 2^{-1022} - 2^{-1074}$$

$$\iff 4,94 \times 10^{-324} \leq x \leq 2,23 \times 10^{-308}$$

En particulier 0 est représenté sous forme dénormalisée avec une mantisse nulle.

La norme IEEE-754

L'exposant e est codé suivant la technique du décalage : il est représenté en machine par l'entier naturel $e' = e + 2^{10} - 1$.

- Enfin, le cas $e' = (1111111111)_2$ permet de coder des valeurs particulières : l'infini qu'on obtient lorsqu'un calcul dépasse le plus grand nombre représentable, et le NaN ("Not a Number") qu'on obtient comme résultat d'une opération invalide.

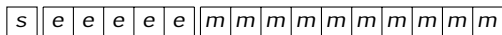
```
In [1]: 2e308          # dépasse le plus grand nombre représentable  
Out[1]: inf
```

```
In [2]: sqrt(-1)     # calcul invalide  
Out[2]: nan
```

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.

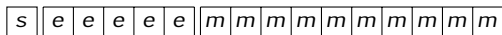


Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

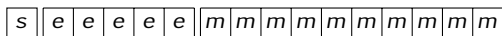
Donner la représentation machine de :

- 1
- -2
- du nombre qui suit 1

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

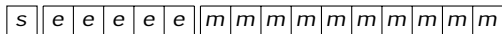
Donner la représentation machine de :

- $1 \rightarrow 0|01111|0000000000$
- -2
- du nombre qui suit 1

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

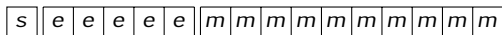
Donner la représentation machine de :

- $1 \rightarrow 0|01111|0000000000$
- $-2 \rightarrow 1|10000|0000000000$
- du nombre qui suit 1

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

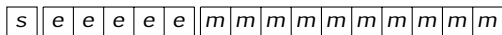
Donner la représentation machine de :

- 1 \rightarrow 0|01111|0000000000
- -2 \rightarrow 1|10000|0000000000
- du nombre qui suit 1 \rightarrow 0|01111|0000000001
 $= 1 + 2^{-10} = 1,0009765625$

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

Donner la représentation machine de :

- 1 \rightarrow 0|01111|0000000000
- -2 \rightarrow 1|10000|0000000000
- du nombre qui suit 1 \rightarrow 0|01111|0000000001

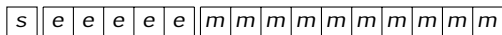
$$= 1 + 2^{-10} = 1,0009765625$$

Donner les représentations machine et la valeur des plus petits et plus grands nombres positifs normalisés.

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

Donner la représentation machine de :

- $1 \rightarrow 0|01111|0000000000$
- $-2 \rightarrow 1|10000|0000000000$
- du nombre qui suit $1 \rightarrow 0|01111|0000000001$

$$= 1 + 2^{-10} = 1,0009765625$$

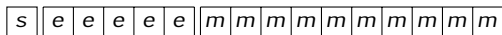
Donner les représentations machine et la valeur des plus petits et plus grands nombres positifs normalisés.

$$\max = 0|11110|1111111111 = 2^{16}579 - 2^5 = 65504$$

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

Donner la représentation machine de :

- 1 \rightarrow 0|01111|0000000000
- -2 \rightarrow 1|10000|0000000000
- du nombre qui suit 1 \rightarrow 0|01111|0000000001

$$= 1 + 2^{-10} = 1,0009765625$$

Donner les représentations machine et la valeur des plus petits et plus grands nombres positifs normalisés.

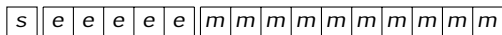
$$\max = 0|11110|1111111111 = 2^{16}579 - 2^5 = 65504$$

$$\min = 0|00001|0000000000 = 2^{-14} \approx 6,1 \times 10^{-5}$$

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

Donner la représentation machine de :

- 1 \rightarrow 0|01111|0000000000
- -2 \rightarrow 1|10000|0000000000
- du nombre qui suit 1 \rightarrow 0|01111|0000000001

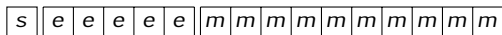
$$= 1 + 2^{-10} = 1,0009765625$$

Déterminer quel nombre est représenté par : 0 | 10000 | 1001001000.

La norme IEEE-754

Exercice

Le type *float16* représente les nombres flottants sur 16 bits : 1 bit pour le signe, 5 bits pour l'exposant, 10 bits pour la mantisse.



Ici le décalage de l'exposant vaut $2^4 - 1 = 15$.

Donner la représentation machine de :

- 1 \rightarrow 0|01111|0000000000
- -2 \rightarrow 1|10000|0000000000
- du nombre qui suit 1 \rightarrow 0|01111|0000000001

$$= 1 + 2^{-10} = 1,0009765625$$

Déterminer quel nombre est représenté par : 0|10000|1001001000.

$$x = (1,1001001)_2 \times 2^1 = (11,001001)_2 = 2 + 1 + \frac{1}{8} + \frac{1}{64} = 3,140625.$$

C'est la meilleure approximation possible de π pour cette représentation.

Calculs sur les flottants

Erreurs d'arrondis dues aux changement de bases

```
In [5]: (0.1 + 0.2) - 0.3
```

```
Out[5]: 5.551115123125783e-17
```

Calculs sur les flottants

Erreurs d'arrondis dues aux changement de bases

```
In [5]: (0.1 + 0.2) - 0.3
Out[5]: 5.551115123125783e-17
```

Règle d'approximation : **round to nearest, ties to even**.

$0,1 = (0,0001\overline{1001} \dots)_2$ possède la représentation machine :

$$1, \underbrace{1001\ 1001 \dots 1001}_{48 \text{ bits}} \mathbf{1010} \times 2^{-4}$$

Calculs sur les flottants

Erreurs d'arrondis dues aux changement de bases

```
In [5]: (0.1 + 0.2) - 0.3
Out[5]: 5.551115123125783e-17
```

Règle d'approximation : **round to nearest, ties to even**.

$0,1 = (0,0001\overline{1001} \dots)_2$ possède la représentation machine :

$$1, \underbrace{1001\ 1001 \dots 1001}_{48 \text{ bits}} \mathbf{1010} \times 2^{-4}$$

$0,2 = (0,001\overline{1001} \dots)_2$ possède la représentation machine :

$$1, \underbrace{1001\ 1001 \dots 1001}_{48 \text{ bits}} \mathbf{1010} \times 2^{-3}$$

Calculs sur les flottants

Erreurs d'arrondis dues aux changement de bases

In [5]: $(0.1 + 0.2) - 0.3$
 Out[5]: 5.551115123125783e-17

Règle d'approximation : **round to nearest, ties to even.**

Addition de 0,1 et de 0,2 :

$$\begin{aligned}
 &0,1100\ 1100\dots 1100\ \mathbf{1101} \times 2^{-3} \\
 &+ 1,1001\ 1001\dots 1001\ \mathbf{1010} \times 2^{-3} \\
 = &10,0110\ 0110\dots 0110\ \mathbf{0111} \times 2^{-3}
 \end{aligned}$$

Calculs sur les flottants

Erreurs d'arrondis dues aux changement de bases

In [5]: $(0.1 + 0.2) - 0.3$
 Out[5]: $5.551115123125783e-17$

Règle d'approximation : **round to nearest, ties to even**.

Addition de 0,1 et de 0,2 :

$$\begin{aligned} & 0,1100\ 1100\dots 1100\ 1101 \times 2^{-3} \\ & + 1,1001\ 1001\dots 1001\ 1010 \times 2^{-3} \\ = & 10,0110\ 0110\dots 0110\ 0111 \times 2^{-3} \end{aligned}$$

et normalisation du résultat (*ties to even*) :

$$= 1,0011\ 0011\dots 0011\ 0100 \times 2^{-2}$$

Calculs sur les flottants

Erreurs d'arrondis dues aux changement de bases

In [5]: $(0.1 + 0.2) - 0.3$
 Out[5]: $5.551115123125783e-17$

Règle d'approximation : **round to nearest, ties to even**.

Addition de 0,1 et de 0,2 :

$$\begin{aligned} & 0,1100\ 1100\dots 1100\ 1101 \times 2^{-3} \\ & + 1,1001\ 1001\dots 1001\ 1010 \times 2^{-3} \\ & = 10,0110\ 0110\dots 0110\ 0111 \times 2^{-3} \end{aligned}$$

et normalisation du résultat (*ties to even*) :

$$= 1,0011\ 0011\dots 0011\ \mathbf{0100} \times 2^{-2}$$

$0,3 = (0,01\overline{0011}\dots)_2$ possède la représentation machine :

$$1,0011\ 0011\dots 0011\ \mathbf{0011} \times 2^{-2}$$

Calculs sur les flottants

Erreurs d'arrondis dues aux changement de bases

```
In [5]: (0.1 + 0.2) - 0.3
Out[5]: 5.551115123125783e-17
```

$(0,1 + 0,2) - 0,3$ est donc représenté en machine par :

$$0,\underbrace{0000\ 0000\ \dots\ 0000}_{48\ \text{bits}}\ 0001 \times 2^{-2}$$

qui est égal à 2^{-54} .

```
In [6]: 2**(-54)
Out[6]: 5.551115123125783e-17
```

Calculs sur les flottants

Absorption et cancellation

Absorption : l'addition entre nombres flottants est impossible lorsque les quantités ont des ordres de grandeurs trop différents.

```
In [7]: (1. + 2.**53) - 2.**53
```

```
Out[7]: 0.0
```


Calculs sur les flottants

Absorption et cancellation

Absorption : l'addition entre nombres flottants est impossible lorsque les quantités ont des ordres de grandeurs trop différents.

```
In [7]: (1. + 2.**53) - 2.**53
Out[7]: 0.0
```

$1 + 2^{53} = (1, \underbrace{0000 \dots 0000}_\text{52 fois 0} 1)_2 \times 2^{53}$ est représenté en machine par :

$$1, \underbrace{0000 \dots 0000}_\text{52 bits} \times 2^{53}.$$

Autrement dit, en arithmétique flottante, $1 + 2^{53} = 2^{53}$.

Calculs sur les flottants

Absorption et cancellation

Absorption : l'addition entre nombres flottants est impossible lorsque les quantités ont des ordres de grandeurs trop différents.

```
In [7]: (1. + 2.**53) - 2.**53
```

```
Out[7]: 0.0
```

$1 + 2^{53} = (1, \underbrace{0000 \dots 0000}_\text{52 fois 0} 1)_2 \times 2^{53}$ est représenté en machine par :

$$\underbrace{1, 0000 \dots 0000}_{\text{52 bits}} \times 2^{53}.$$

Autrement dit, en arithmétique flottante, $1 + 2^{53} = 2^{53}$.

l'addition de deux quantités dont l'écart relatif est très important entraîne l'absorption de la plus petite de ces deux quantités par la plus grande.

```
In [8]: 1. + (2.**53 - 2.**53)
```

```
Out[8]: 1.0
```

Calculs sur les flottants

Absorption et cancellation

Cancellation : la soustraction de deux quantités très proches occasionne une grande perte de précision relative.

Exemple : on suppose connaître deux quantités voisines x et y avec une précision de 20 chiffres significatifs après la virgule.

$$x = 1,10010010000111111011$$

$$y = 1,10010010000111100110$$

Calculs sur les flottants

Absorption et cancellation

Cancellation : la soustraction de deux quantités très proches occasionne une grande perte de précision relative.

Exemple : on suppose connaître deux quantités voisines x et y avec une précision de 20 chiffres significatifs après la virgule.

$$\begin{aligned}x &= 1,10010010000111111011 \\y &= 1,10010010000111100110\end{aligned}$$

On a :

$$x - y = 0,000000000000000010101$$

Le résultat $x - y$ est normalisé pour être représenté en machine par :

$$(1,0101)_2 \times 2^{-16}$$

avec une précision de 4 chiffres après la virgule.

La différence de deux quantités très voisines fait s'évanouir les chiffres significatifs.