Semestre 2 / Cours 6 - Représentation des nombres

ITC MPSI & PCSI – Année 2024-2025

LYCÉE MONTAIGNE

Sommaire

1. Représentation des entiers positifs

2. Représentation binaire des entiers signés

3. Représentation des réels

Objectifs

- Appréhender les limitations intrinsèques à la représentation et la manipulation informatique des nombres.

Objectifs

- Appréhender les limitations intrinsèques à la représentation et la manipulation informatique des nombres.
- Initier un sens critique au sujet de la qualité et de la précision des résultats de calculs numériques sur ordinateur.

Objectifs

- Appréhender les limitations intrinsèques à la représentation et la manipulation informatique des nombres.
- Initier un sens critique au sujet de la qualité et de la précision des résultats de calculs numériques sur ordinateur.

```
>>> 10**16+1-10**16
1
>>> 10**16+1.0-10**16
0.0
>>> 0.1+0.2 == 0.3
False
```

Problématique

- Taille finie de la mémoire.

Problématique

- Taille finie de la mémoire.
- Contraintes sur la représentation des nombres.

Problématique

- Taille finie de la mémoire.
- Contraintes sur la représentation des nombres.
- Conséquences sur la manipulation des nombres en machine.

Représentation des entiers positifs

De la base 10 à la base 2

En base 10
12345 désigne l'entier naturel n

$$n = 1 \times \underbrace{10000}_{10^4} + 2 \times \underbrace{1000}_{10^3} + 3 \times \underbrace{100}_{10^2} + 4 \times \underbrace{10}_{10^1} + 5 \times \underbrace{1}_{10^0}.$$

De la base 10 à la base 2

En base 10
12345 désigne l'entier naturel n

$$n = 1 \times \underbrace{10000}_{10^4} + 2 \times \underbrace{1000}_{10^3} + 3 \times \underbrace{100}_{10^2} + 4 \times \underbrace{10}_{10^1} + 5 \times \underbrace{1}_{10^0}.$$

En base 2
101001 désigne l'entier naturel n

$$n = 1 \times \underbrace{32}_{2^5} + 0 \times \underbrace{16}_{2^4} + 1 \times \underbrace{8}_{2^3} + 0 \times \underbrace{4}_{2^2} + 0 \times \underbrace{2}_{2^1} + 1 \times \underbrace{1}_{2^0}.$$

qui vaut 41 en base 10.

Plus formellement

En base 10
Les chiffres c_k sont dans $\{0, 1, \ldots, 9\}$ et l'écriture $c_{N-1} \ldots c_1 c_0$ désigne l'entier naturel n

$$n = \sum_{k=0}^{N-1} c_k \times 10^k.$$

Plus formellement

En base 10
Les chiffres c_k sont dans $\{0, 1, \ldots, 9\}$ et l'écriture $c_{N-1} \ldots c_1 c_0$ désigne l'entier naturel n

$$n = \sum_{k=0}^{N-1} c_k \times 10^k.$$

En base 2
Les chiffres c_k sont dans $\{0, 1\}$ et l'écriture $c_{N-1} \ldots c_1 c_0$ désigne l'entier naturel n

$$n = \sum_{k=0}^{N-1} c_k \times 2^k.$$

Plus formellement

En base 10
Les chiffres c_k sont dans $\{0, 1, \ldots, 9\}$ et l'écriture $c_{N-1} \ldots c_1 c_0$ désigne l'entier naturel n

$$n = \sum_{k=0}^{N-1} c_k \times 10^k.$$

En base 2
Les chiffres c_k sont dans $\{0, 1\}$ et l'écriture $c_{N-1} \ldots c_1 c_0$ désigne l'entier naturel n

$$n = \sum_{k=0}^{N-1} c_k \times 2^k.$$

En base b
Soient $b \geq 2$ et $n \geqslant 1$ deux entiers. Alors :

$$\exists\,!\, N \geqslant 1,\ \ \exists\,!\,(c_0, \ldots, c_{N-1}) \in [\![0\ ,\ b-1]\!]^N, \quad n = \sum_{k=0}^{N-1} c_k b^k \quad \underline{\textbf{et}} \quad c_{N-1} \neq 0.$$

Plus formellement

En cas d'ambiguïté, il faut préciser la base :

- $(101001)_{10}$ vaut cent-un-mille-un.
- $(101001)_2$ vaut quarante-et-un.

Conversion en base 2

Les chiffres binaires s'obtiennent en calculant les **restes** successifs de la **division euclidienne** de n par 2. En effet :

- $n = c_0 + 2\sum_{k=1}^{N-1} c_k 2^{k-1}$, donc : $c_0 = n\%2$.

Conversion en base 2

Les chiffres binaires s'obtiennent en calculant les **restes** successifs de la **division euclidienne** de n par 2. En effet :

- $n = c_0 + 2\sum_{k=1}^{N-1} c_k 2^{k-1}$, donc : $c_0 = n\%2$.
- Ensuite, comme $n//2 = \sum_{k=1}^{N-1} c_k 2^{k-1}$ on obtient comme précédemment en sortant le 1er terme : $c_1 = (n//2)\%2$.

Conversion en base 2

Les chiffres binaires s'obtiennent en calculant les **restes** successifs de la **division euclidienne** de n par 2. En effet :

- $n = c_0 + 2\sum_{k=1}^{N-1} c_k 2^{k-1}$, donc : $c_0 = n\%2$.
- Ensuite, comme $n//2 = \sum_{k=1}^{N-1} c_k 2^{k-1}$ on obtient comme précédemment en sortant le 1er terme : $c_1 = (n//2)\%2$.
- Et ainsi de suite....

Exemple
Écriture binaire de 25 :

25	2				
1	12	2			
	0	6	2		
		0	3	2	
			1	1	2
				1	0

donc

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

(Attention à l'ordre !).

Autre présentation (les chiffres sont dans le bon ordre)

					25

				12	25
					1

			6	12	25
				0	1

		3	6	12	25
			0	0	1

	1	3	6	12	25
		1	0	0	1

0	1	3	6	12	25
	1	1	0	0	1

Une addition en binaire

Calcul de $22 + 15$
22 s'écrit 10110 en binaire

15 s'écrit 1111 en binaire

Une addition en binaire

Calcul de $22 + 15$
22 s'écrit 10110 en binaire

15 s'écrit 1111 en binaire

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

100101 est bien l'écriture binaire de 37.

Une addition en binaire

Calcul de $22 + 15$
22 s'écrit 10110 en binaire

15 s'écrit 1111 en binaire

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

100101 est bien l'écriture binaire de 37.

Notez l'utilisation des **retenues**.

Exercice

Calculer la somme $13 + 11$ et le produit 13×11 en utilisant les représentations binaires de ces nombres.

Solution

Représentation binaire
On a $(13)_{10} = (1101)_2$ et $(11)_{10} = (1011)_2$

Solution

Représentation binaire
On a $(13)_{10} = (1101)_2$ et $(11)_{10} = (1011)_2$

Somme

$$
\begin{array}{cccccc}
 & {\scriptstyle (1)} & {\scriptstyle (1)} & {\scriptstyle (1)} & {\scriptstyle (1)} & \\
 & & 1 & 1 & 0 & 1 \\
+ & & 1 & 0 & 1 & 1 \\
\hline
= & 1 & 1 & 0 & 0 & 0
\end{array}
$$

Produit

$$
\begin{array}{cccccccccc}
 & & & & & & 1 & 1 & 0 & 1 \\
\times & & & & & & 1 & 0 & 1 & 1 \\
\hline
 & & (1) & (1) & (1) & & & & & \\
 & & & & & & 1 & 1 & 0 & 1 \\
+ & & & & & 1 & 1 & 0 & 1 & 0 \\
+ & & & & 0 & 0 & 0 & 0 & 0 & 0 \\
+ & & & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
\hline
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{array}
$$

donne $13 \times 11 = 143$.

Stockage sur N bits

En utilisant N bits, on peut stocker **uniquement** les entiers positifs de l'intervalle

$$[\![0; 2^N - 1]\!]$$

0 est représenté par $(00\ldots0)_2$
$2^N - 1$ est représenté par $(11\ldots1)_2$

Stockage sur N bits

En utilisant N bits, on peut stocker **uniquement** les entiers positifs de l'intervalle

$$[\![0; 2^N - 1]\!]$$

0 est représenté par $(00\ldots0)_2$
$2^N - 1$ est représenté par $(11\ldots1)_2$
On ajoute éventuellement des 0 à gauche de l'écriture.
Exemple : 25 est stocké sur 8 bits sous la forme $(00011001)_2$.

Stockage sur N bits

En utilisant N bits, on peut stocker **uniquement** les entiers positifs de l'intervalle

$$[\![0; 2^N - 1]\!]$$

0 est représenté par $(00\ldots0)_2$
$2^N - 1$ est représenté par $(11\ldots1)_2$
On ajoute éventuellement des 0 à gauche de l'écriture.
Exemple : 25 est stocké sur 8 bits sous la forme $(00011001)_2$.

Les autres entiers ne sont pas représentables.
Exemple : sur 4 bits on représente les entiers de $[\![0; 15]\!]$, donc 25 n'est pas représentable sur 4 bits (il faut au moins 5 bits).

Stockage sur N bits

En utilisant N bits, on peut stocker **uniquement** les entiers positifs de l'intervalle

$$[\![0; 2^N - 1]\!]$$

0 est représenté par $(00\ldots0)_2$
$2^N - 1$ est représenté par $(11\ldots1)_2$
On ajoute éventuellement des 0 à gauche de l'écriture.
Exemple : 25 est stocké sur 8 bits sous la forme $(00011001)_2$.

Les autres entiers ne sont pas représentables.
Exemple : sur 4 bits on représente les entiers de $[\![0; 15]\!]$, donc 25 n'est pas représentable sur 4 bits (il faut au moins 5 bits).
Sur $N = 32$ bits (4 octets) on représente $[\![0; 4294967295]\!]$.
Sur $N = 64$ bits (8 octets) on représente $[\![0; 18446744073709551615]\!]$.

Conséquence

En limitant le nombre de bits stockés, les calculs ne sont plus nécessairement **exacts**.

Conséquence

En limitant le nombre de bits stockés, les calculs ne sont plus nécessairement **exacts**. Exemple : $22 + 15$ sur 5 bits :

$$\begin{array}{cccccc} & {\scriptstyle (1)} & {\scriptstyle (1)} & {\scriptstyle (1)} & & \\ & 1 & 0 & 1 & 1 & 0 \\ + & 0 & 1 & 1 & 1 & 1 \\ \hline & 0 & 0 & 1 & 0 & 1 \end{array}$$

donc :

$$22 + 15 = 5.$$

Conséquence

En limitant le nombre de bits stockés, les calculs ne sont plus nécessairement **exacts**. Exemple : $22 + 15$ sur 5 bits :

$$\begin{array}{cccccc} & {\scriptstyle (1)} & {\scriptstyle (1)} & {\scriptstyle (1)} & & \\ & 1 & 0 & 1 & 1 & 0 \\ + & 0 & 1 & 1 & 1 & 1 \\ \hline & 0 & 0 & 1 & 0 & 1 \end{array}$$

donc :

$$22 + 15 = 5.$$

Explication : la perte de la dernière retenue provoque un écart de $2^5 = 32$. Il faut en fait lire :

$$22 + 15 \equiv 5 \qquad [32]\,.$$

Conversion décimal vers binaire

Écrire une fonction `dec2bin(n : int, N : int) -> list` renvoyant la liste formant l'écriture binaire sur N bits de l'entier positif d'écriture décimale n.

Par exemple, `dec2bin(25,6)` renverra la liste `[0, 1, 1, 0, 0, 1]`.

L'entier positif n est supposé être représentable sur N bits, ce dont la fonction s'assurera en utilisant une instruction **`assert`**.

Solution

Conversion décimal vers binaire

```
def dec2bin(n : int, N : int) -> list :
    assert 0 <= n < 2**N
    L = [0]*N
    i = N-1
    while n! = 0 :
        L[i] = n%2
        n = n//2
        i -= 1
    return L
```

Conversion binaire vers décimal

On souhaite écrire une fonction `bin2dec(L : list) -> int` renvoyant l'écriture décimale de l'entier positif dont la liste L contient l'écriture binaire.

Par exemple, `bin2dec([0, 1, 1, 0, 0, 1])` renverra l'entier `25`.

1. Proposer une première version de cette fonction qui, pour $L = [c_{N-1}c_{N-2}\dots c_1c_0]$, calcule la somme $\sum_{k=0}^{N-1} c_k \times 2^k$ sans chercher à optimiser l'évaluation des puissances de 2.

Conversion binaire vers décimal

On souhaite écrire une fonction `bin2dec(L : list) -> int` renvoyant l'écriture décimale de l'entier positif dont la liste L contient l'écriture binaire.

Par exemple, `bin2dec([0, 1, 1, 0, 0, 1])` renverra l'entier `25`.

1. Proposer une première version de cette fonction qui, pour $L = [c_{N-1}c_{N-2}\dots c_1c_0]$, calcule la somme $\sum_{k=0}^{N-1} c_k \times 2^k$ sans chercher à optimiser l'évaluation des puissances de 2.Exprimer, en fonction de $N = \text{len}(L)$, la complexité de cette fonction (le calcul de 2^k comptera pour k opérations).

Conversion binaire vers décimal

On souhaite écrire une fonction `bin2dec(L : list) -> int` renvoyant l'écriture décimale de l'entier positif dont la liste L contient l'écriture binaire.

Par exemple, `bin2dec([0, 1, 1, 0, 0, 1])` renverra l'entier `25`.

1. Proposer une première version de cette fonction qui, pour $L = [c_{N-1}c_{N-2}\dots c_1c_0]$, calcule la somme $\sum_{k=0}^{N-1} c_k \times 2^k$ sans chercher à optimiser l'évaluation des puissances de 2.Exprimer, en fonction de $N = \text{len}(L)$, la complexité de cette fonction (le calcul de 2^k comptera pour k opérations).
2. On souhaite maintenant réduire cette complexité en optimisant le calcul des puissances.

Conversion binaire vers décimal

On souhaite écrire une fonction `bin2dec(L : list) -> int` renvoyant l'écriture décimale de l'entier positif dont la liste L contient l'écriture binaire.
Par exemple, `bin2dec([0, 1, 1, 0, 0, 1])` renverra l'entier `25`.

1. Proposer une première version de cette fonction qui, pour $L = [c_{N-1}c_{N-2}\dots c_1c_0]$, calcule la somme $\sum_{k=0}^{N-1} c_k \times 2^k$ sans chercher à optimiser l'évaluation des puissances de 2.Exprimer, en fonction de $N = \text{len}(L)$, la complexité de cette fonction (le calcul de 2^k comptera pour k opérations).
2. On souhaite maintenant réduire cette complexité en optimisant le calcul des puissances.
   - **2.1)** On note $L' = [c_{N-1}c_{N-2}\dots c_1]$ la sous liste de L amputée de son dernier élément. Déterminer une relation liant l'entier n représenté par L et l'entier n' représenté par L'.

Conversion binaire vers décimal

On souhaite écrire une fonction `bin2dec(L : list) -> int` renvoyant l'écriture décimale de l'entier positif dont la liste L contient l'écriture binaire.
Par exemple, `bin2dec([0, 1, 1, 0, 0, 1])` renverra l'entier `25`.

1. Proposer une première version de cette fonction qui, pour $L = [c_{N-1}c_{N-2}\dots c_1c_0]$, calcule la somme $\sum_{k=0}^{N-1} c_k \times 2^k$ sans chercher à optimiser l'évaluation des puissances de 2.Exprimer, en fonction de $N = \text{len}(L)$, la complexité de cette fonction (le calcul de 2^k comptera pour k opérations).
2. On souhaite maintenant réduire cette complexité en optimisant le calcul des puissances.
   - **2.1)** On note $L' = [c_{N-1}c_{N-2}\dots c_1]$ la sous liste de L amputée de son dernier élément. Déterminer une relation liant l'entier n représenté par L et l'entier n' représenté par L'.
   - **2.2)** En déduire une version **récursive** de la fonction `bin2dec`.

Conversion binaire vers décimal

On souhaite écrire une fonction `bin2dec(L : list) -> int` renvoyant l'écriture décimale de l'entier positif dont la liste L contient l'écriture binaire.
Par exemple, `bin2dec([0, 1, 1, 0, 0, 1])` renverra l'entier `25`.

1. Proposer une première version de cette fonction qui, pour $L = [c_{N-1}c_{N-2}\dots c_1c_0]$, calcule la somme $\sum_{k=0}^{N-1} c_k \times 2^k$ sans chercher à optimiser l'évaluation des puissances de 2.Exprimer, en fonction de $N = \text{len}(L)$, la complexité de cette fonction (le calcul de 2^k comptera pour k opérations).
2. On souhaite maintenant réduire cette complexité en optimisant le calcul des puissances.
   - **2.1)** On note $L' = [c_{N-1}c_{N-2}\dots c_1]$ la sous liste de L amputée de son dernier élément. Déterminer une relation liant l'entier n représenté par L et l'entier n' représenté par L'.
   - **2.2)** En déduire une version **récursive** de la fonction `bin2dec`.
   - **2.3)** Préciser la complexité de cette version.

Conversion binaire vers décimal

On souhaite écrire une fonction `bin2dec(L : list) -> int` renvoyant l'écriture décimale de l'entier positif dont la liste L contient l'écriture binaire.
Par exemple, `bin2dec([0, 1, 1, 0, 0, 1])` renverra l'entier `25`.

1. Proposer une première version de cette fonction qui, pour $L = [c_{N-1}c_{N-2}\dots c_1c_0]$, calcule la somme $\sum_{k=0}^{N-1} c_k \times 2^k$ sans chercher à optimiser l'évaluation des puissances de 2.Exprimer, en fonction de $N = \text{len}(L)$, la complexité de cette fonction (le calcul de 2^k comptera pour k opérations).
2. On souhaite maintenant réduire cette complexité en optimisant le calcul des puissances.
   - **2.1)** On note $L' = [c_{N-1}c_{N-2}\dots c_1]$ la sous liste de L amputée de son dernier élément. Déterminer une relation liant l'entier n représenté par L et l'entier n' représenté par L'.
   - **2.2)** En déduire une version **récursive** de la fonction `bin2dec`.
   - **2.3)** Préciser la complexité de cette version.
   - **2.4)** Écrire une version impérative de la fonction `bin2dec` fondée sur la même idée.

Solution

Question 1

```
def bin2dec(L : list) -> int :
    n = 0
    N = len(L)
    for k in range(N) :
        n += L[N-k-1]*2**k
    return n
```

Solution

Question 1

```
def bin2dec(L : list) -> int :
    n = 0
    N = len(L)
    for k in range(N) :
        n += L[N-k-1]*2**k
    return n
```

On compte 3 opérations en lignes #2 et #3, et pour $i \in [\![1; N]\!]$ le i^e passage dans la boucle for (correspondant à $k_i = i - 1$) donne $4 + i$ opérations en ligne #5, donc :

$$C_n = 3 + \sum_{i=1}^{N} (4 + i) = 3 + 4N + \frac{N(N+1)}{2} = \mathrm{O}\left(N^2\right)$$

Solution

Question 2

a. On a $n' = \sum_{k=0}^{N-2} c_{k+1} 2^k$ donc $n = 2n' + c_0$.

Solution

Question 2

a. On a $n' = \sum_{k=0}^{N-2} c_{k+1} 2^k$ donc $n = 2n' + c_0$.

b.

```
def bin2dec(L : list) -> int :
    if len(L) == 0:
        return 0
    else :
        return 2*bin2dec(L[:-1]) + L[-1]
```

Solution

Question 2

a. On a $n' = \sum_{k=0}^{N-2} c_{k+1} 2^k$ donc $n = 2n' + c_0$.

b.

```
def bin2dec(L : list) -> int :
    if len(L) == 0:
        return 0
    else :
        return 2*bin2dec(L[:-1]) + L[-1]
```

c. On a $C_0 = 1$ (pour le test) et si $N \geqslant 1$ on compte $C_N = 3 + C_{N-1}$ donc

$$C_N = 1 + 3N = \mathrm{O}(N).$$

Solution

Question 2

a. On a $n' = \sum_{k=0}^{N-2} c_{k+1} 2^k$ donc $n = 2n' + c_0$.

b.

```
def bin2dec(L : list) -> int :
    if len(L) == 0:
        return 0
    else :
        return 2*bin2dec(L[:-1]) + L[-1]
```

c. On a $C_0 = 1$ (pour le test) et si $N \geqslant 1$ on compte $C_N = 3 + C_{N-1}$ donc

$$C_N = 1 + 3N = \mathrm{O}(N).$$

d.

```
def bin2dec(L : list) -> int :
    n = 0
    for c in L :
        n = 2*n+c
    return n
```

Représentation binaire des entiers signés

Des entiers signés

On souhaite dans cette partie coder en binaire, toujours sur un nombre N de bits fixé, des entiers **relatifs**.

Une mauvaise solution

Un bit de signe
Le premier bit stocke le signe de l'entier (par exemple 0 si positif et 1 si négatif) et les $N - 1$ bits suivants sa valeur absolue.

Une mauvaise solution

Un bit de signe
Le premier bit stocke le signe de l'entier (par exemple 0 si positif et 1 si négatif) et les $N-1$ bits suivants sa valeur absolue.

Par exemple, sur $N = 8$ bits :

- 3 serait représenté par $(00000011)_2$;
- -4 serait représenté par $(10000100)_2$.

Une mauvaise solution

Problème
Les algorithmes usuels pour les opérations ne fonctionnent plus.

Une mauvaise solution

Problème
Les algorithmes usuels pour les opérations ne fonctionnent plus.

Exemple, pour $3 + (-4)$ sur 8 bits, la méthode d'addition donne :

$$\begin{array}{cccccccccc} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ + & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Donc on obtient dans ce cas

$$3 + (-4) = -7.$$

(Faux même modulo $2^8 = 256$.)

Une meilleure idée

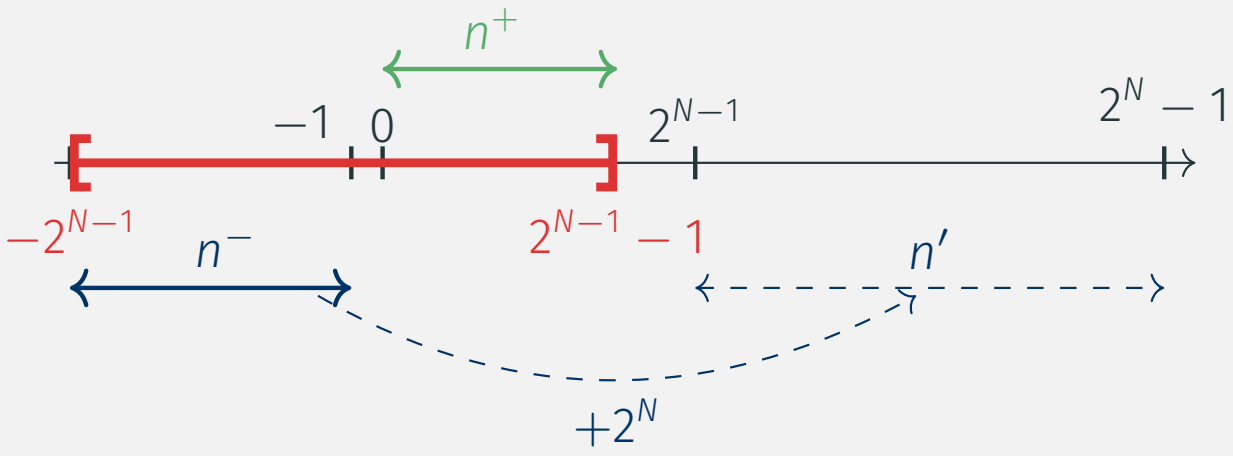
Principe
Représenter sur N bits les entiers n de l'intervalle

$$[\![-2^{N-1}; 2^{N-1} - 1]\!]$$

de la manière suivante :

- si $n \geqslant 0$ (donc $n \in [\![0; 2^{N-1} - 1]\!]$) alors n est codé sur N bits (c.f. partie précédente);
- si $n < 0$ (donc $n \in [\![-2^{N-1}; -1]\!]$) alors on calcule l'entier naturel $n' = n + 2^N$ (donc $n' \in [\![2^{N-1}; 2^N - 1]\!]$) dont le codage en tant qu'entier naturel sous N bits sera la représentation de n.

Visualisation linéaire des correspondances



(n^+ codé de manière habituelle, n^- après décalage)

Figure 1 – Codage des entiers signés

Tableau des correspondances

Nombre	Représentation
0	0000 0000
1	0000 0001
. . .	. . . . .
$2^{N-1} - 1$	0111 1111
-2^{N-1}	1000 0000
. . .	. . . . .
-1	1111 1111

Figure 2 – Les entiers relatifs codés sur N bits, classés par représentation croissante.

Une meilleure idée

L'idée, finalement ...
L'entier négatif à coder est remplacé par son représentant positif le plus proche modulo 2^N.

(Idée assez naturelle, puisque les calculs eux-mêmes sont faits modulo 2^N)

Exemples

Sur $N = 8$ bits
On code l'intervalle

$$[\![-2^7; 2^7 - 1]\!] = [\![-128; 127]\!].$$

- $n = 45$ se code 00101101 (puisque $n \geqslant 0$ donc se code directement).
- $n = -45$ se code 11010011 (puisque $n < 0$, on code $n' = n + 2^8 = -45 + 256 = 211$).

Un bit de signe

Avec ce codage, le bit de poids fort (celui de gauche) est un bit de signe.

Un bit de signe

Avec ce codage, le bit de poids fort (celui de gauche) est un bit de signe.

En effet sur N bits le premier bit est un paquet de 2^{N-1} et

- si $n \geqslant 0$ alors $n \in [\![0; 2^{N-1} - 1]\!]$ donc le premier bit vaut 0 ;
- si $n < 0$ alors $n \in [\![-2^{N-1}; 0]\!]$ et on code $n' = 2^N + n \in [\![2^{N-1}; 2^N - 1]\!]$ donc le premier bit vaut 1.

Un bit de signe

Avec ce codage, le bit de poids fort (celui de gauche) est un bit de signe.

En effet sur N bits le premier bit est un paquet de 2^{N-1} et

- si $n \geqslant 0$ alors $n \in [\![0; 2^{N-1} - 1]\!]$ donc le premier bit vaut 0 ;
- si $n < 0$ alors $n \in [\![-2^{N-1}; 0]\!]$ et on code $n' = 2^N + n \in [\![2^{N-1}; 2^N - 1]\!]$ donc le premier bit vaut 1.

(Mais les bits suivants ne codent pas la valeur absolue !)

Décodage

- si le codage commence par 0 : on décode l'entier naturel qui est n ;
- si le codage commence par 1 : on décode l'entier naturel qui est n', et alors $n = n' - 2^N$.

Décodage

- si le codage commence par 0 : on décode l'entier naturel qui est n ;
- si le codage commence par 1 : on décode l'entier naturel qui est n', et alors $n = n' - 2^N$.

Exemples

- $(00010110)_2$ code $n = 22$.
- $(10010110)_2$ code $n' = 150$, donc $n = n' - 2^8 = 150 - 256 = -106$.

Des opérations sur les entiers relatifs

L'entier représenté est congru à l'entier codé, et l'addition et la multiplication sont compatibles avec les congruences : on peut utiliser les mêmes algorithmes que pour les entiers naturels !

Exercice

Calculer $3 + (-4)$ et $3 \times (-4)$ à partir de leurs représentations binaires sur 8 bits.

Solution

Représentation binaire
Sur 8 bits :

- 3 se code $(00000011)_2$;
- -4 se code $(11111100)_2$ (on code $-4 + 256 = 252$).

Solution

Représentation binaire
Sur 8 bits :

- 3 se code $(00000011)_2$;
- -4 se code $(11111100)_2$ (on code $-4 + 256 = 252$).

Somme

$$\begin{array}{cccccccccc} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ + & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ \hline & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

qui représente bien -1 (on décode $n' = 255$ donc $n = n' - 256 = -1$).

Produit

$$
\begin{array}{rcccccccc}
 & & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\times & & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
 & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & & & \\
 & & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
+ & & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\hline
 & & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}
$$

qui représente bien -12 (on décode $n' = 244$ et $n = n' - 256 = -12$).

Problème pratique
Pour coder $n < 0$, il faut représenter $n' = n + 2^N$, mais l'ordinateur ne sait pas faire cette addition.

Codage par complément à deux

Problème pratique
Pour coder $n < 0$, il faut représenter $n' = n + 2^N$, mais l'ordinateur ne sait pas faire cette addition.

Méthode du complément à deux
Pour coder $n < 0$:

- On code l'entier naturel $-n$ sur N bits ;
- puis on inverse tous les bits de cette écriture (les 0 en 1, et inversement) ;
- on ajoute 1 au résultat (addition binaire sur N bits).

Codage par complément à deux

Problème pratique
Pour coder $n < 0$, il faut représenter $n' = n + 2^N$, mais l'ordinateur ne sait pas faire cette addition.

Méthode du complément à deux
Pour coder $n < 0$:

- On code l'entier naturel $-n$ sur N bits ;
- puis on inverse tous les bits de cette écriture (les 0 en 1, et inversement) ;
- on ajoute 1 au résultat (addition binaire sur N bits).

Exemple
Exemple : codage de $n = -45$ sur 8 bits :

Codage par complément à deux

Problème pratique
Pour coder $n < 0$, il faut représenter $n' = n + 2^N$, mais l'ordinateur ne sait pas faire cette addition.

Méthode du complément à deux
Pour coder $n < 0$:

- On code l'entier naturel $-n$ sur N bits ;
- puis on inverse tous les bits de cette écriture (les 0 en 1, et inversement) ;
- on ajoute 1 au résultat (addition binaire sur N bits).

Exemple
Exemple : codage de $n = -45$ sur 8 bits :

- on code $-n = 45$, ce qui donne 00101101 ;

Codage par complément à deux

Problème pratique
Pour coder $n < 0$, il faut représenter $n' = n + 2^N$, mais l'ordinateur ne sait pas faire cette addition.

Méthode du complément à deux
Pour coder $n < 0$:

- On code l'entier naturel $-n$ sur N bits ;
- puis on inverse tous les bits de cette écriture (les 0 en 1, et inversement) ;
- on ajoute 1 au résultat (addition binaire sur N bits).

Exemple
Exemple : codage de $n = -45$ sur 8 bits :

- on code $-n = 45$, ce qui donne 00101101 ;
- on inverse tous les bits, ce qui donne 11010010 ;

Codage par complément à deux

Problème pratique
Pour coder $n < 0$, il faut représenter $n' = n + 2^N$, mais l'ordinateur ne sait pas faire cette addition.

Méthode du complément à deux
Pour coder $n < 0$:

- On code l'entier naturel $-n$ sur N bits ;
- puis on inverse tous les bits de cette écriture (les 0 en 1, et inversement) ;
- on ajoute 1 au résultat (addition binaire sur N bits).

Exemple
Exemple : codage de $n = -45$ sur 8 bits :

- on code $-n = 45$, ce qui donne 00101101 ;
- on inverse tous les bits, ce qui donne 11010010 ;
- on ajoute 1 au résultat, ce qui donne 11010011.

Preuve

- On note $c_{N-1}c_{N-2}\dots c_1c_0$ le codage de $-n$ (donc $-n = \sum_{k=0}^{N-1} c_k 2^k$);

Preuve

- On note $c_{N-1}c_{N-2}\dots c_1c_0$ le codage de $-n$ (donc $-n = \sum_{k=0}^{N-1} c_k 2^k$);
- en inversant tous les bits on écrit $d_{N-1}d_{N-2}\dots d_1d_0$ avec $d_k = 1 - c_k$;

Preuve

- On note $c_{N-1}c_{N-2}\dots c_1c_0$ le codage de $-n$ (donc $-n = \sum_{k=0}^{N-1} c_k 2^k$);
- en inversant tous les bits on écrit $d_{N-1}d_{N-2}\dots d_1d_0$ avec $d_k = 1 - c_k$;
- en ajoutant 1 on obtient le codage de

$$1 + \sum_{k=0}^{N-1} d_k 2^k = 1 + \sum_{k=0}^{N-1} (1 - c_k) 2^k = 1 + \underbrace{\sum_{k=0}^{N-1} 2^k}_{\frac{1-2^N}{1-2} = 2^N - 1} - \underbrace{\sum_{k=0}^{N-1} c_k 2^k}_{-n} = n + 2^N = n'.$$

Décodage par complément à deux

Problème pratique
Si le bit de poids fort est 1 on décode n' puis $n = n' - 2^N$, mais l'ordinateur ne sait pas faire cette soustraction.

Décodage par complément à deux

Problème pratique
Si le bit de poids fort est 1 on décode n' puis $n = n' - 2^N$, mais l'ordinateur ne sait pas faire cette soustraction.

Décodage par complément à deux

- on inverse tous les bits de cette écriture ;
- puis on ajoute 1 au résultat (addition binaire sur N bits) ;
- on décode l'entier naturel obtenu : l'opposé de ce nombre est le résultat.

Décodage par complément à deux

Problème pratique
Si le bit de poids fort est 1 on décode n' puis $n = n' - 2^N$, mais l'ordinateur ne sait pas faire cette soustraction.

Décodage par complément à deux

- on inverse tous les bits de cette écriture ;
- puis on ajoute 1 au résultat (addition binaire sur N bits) ;
- on décode l'entier naturel obtenu : l'opposé de ce nombre est le résultat.

Exemple : décodage de 10010110

- on inverse tous les bits, ce qui donne 01101001 ;
- on ajoute 1, ce qui donne 01101010 ;
- on décode l'entier naturel 106, donc le résultat est -106.

Preuve

- On note n l'entier relatif de codage $c_{N-1}c_{N-2}\dots c_1c_0$ avec $c_{N-1}=1$ (donc $2^N+n=n'=\sum_{k=0}^{N-1}c_k2^k$);
- en inversant tous les bits on écrit $d_{N-1}d_{N-2}\dots d_1d_0$ avec $d_k=1-c_k$;
- en ajoutant 1 on obtient le codage de l'entier

$$1+\sum_{k=0}^{N-1}d_k2^k=1+\sum_{k=0}^{N-1}(1-c_k)2^k=1+\underbrace{\sum_{k=0}^{N-1}2^k}_{\frac{1-2^N}{1-2}=2^N-1}-\underbrace{\sum_{k=0}^{N-1}c_k2^k}_{2^N+n}=-n.$$

Exercice

Cet exercice utilise les fonctions `dec2bin` et `bin2dec` écrites dans des exercices précédents.

1. Écrire une fonction `dec2binSigne(n : int, N : int) -> list` renvoyant la liste formant l'écriture binaire sur N bits de l'entier signé d'écriture décimale n.
   Par exemple, `dec2bin(-45,8)` renverra la liste `[1, 1, 0, 1, 0, 0, 1, 1]`.
   L'entier signé n est supposé être représentable sur N bits, ce dont la fonction s'assurera en utilisant une instruction **`assert`**.
   La méthode utilisée sera celle utilisant une addition pour calculer l'entier n' si nécessaire (la méthode utilisant le complément à deux sera programmée en TP).

Exercice

Cet exercice utilise les fonctions `dec2bin` et `bin2dec` écrites dans des exercices précédents.

1. Écrire une fonction `dec2binSigne(n : int, N : int) -> list` renvoyant la liste formant l'écriture binaire sur N bits de l'entier signé d'écriture décimale n.
   Par exemple, `dec2bin(-45,8)` renverra la liste `[1, 1, 0, 1, 0, 0, 1, 1]`.
   L'entier signé n est supposé être représentable sur N bits, ce dont la fonction s'assurera en utilisant une instruction **`assert`**.
   La méthode utilisée sera celle utilisant une addition pour calculer l'entier n' si nécessaire (la méthode utilisant le complément à deux sera programmée en TP).
2. Écrire une fonction `bin2decSigne(L : list) -> int` renvoyant l'écriture décimale de l'entier signé dont la liste L (non vide) contient l'écriture binaire.
   Par exemple, `bin2decSigne([1, 0, 0, 1, 0, 1, 1, 0])` renverra l'entier `-106`.
   Dans cette question également, la méthode utilisée ne sera pas celle du complément à deux.

Solution

Q1

```
def dec2binSigne(n : int, N : int) -> list :
    assert -2**(N-1) <= n < 2**(N-1)
    if n >= 0:
        return dec2bin(n, N)
    else :
        return dec2bin(n+2**N, N)
```

Solution

Q2

```
def bin2decSigne(L : list) -> int :
    if L[0] == 0:
        return bin2dec(L)
    else :
        return bin2dec(L)-2**len(L)
```

Et en langage Python ?

Dans de nombreux langages, les entiers signés sont codés, comme nous l'avons vu, avec un nombre de bits N fixé une fois pour toutes (généralement 32 ou 64 bits), et c'était encore le cas pour le langage Python jusqu'aux versions 2.xx.

Et en langage Python ?

Dans de nombreux langages, les entiers signés sont codés, comme nous l'avons vu, avec un nombre de bits N fixé une fois pour toutes (généralement 32 ou 64 bits), et c'était encore le cas pour le langage Python jusqu'aux versions 2.xx.

À partir des versions suivantes, pour éviter les problèmes de dépassement (opérations vraies uniquement modulo 2^N), le langage Python modifie **dynamiquement** la taille mémoire allouée au stockage des entiers manipulés pour permettre une représentation et des calculs **exacts**.

Et en langage Python ?

Dans de nombreux langages, les entiers signés sont codés, comme nous l'avons vu, avec un nombre de bits N fixé une fois pour toutes (généralement 32 ou 64 bits), et c'était encore le cas pour le langage Python jusqu'aux versions 2.xx.

À partir des versions suivantes, pour éviter les problèmes de dépassement (opérations vraies uniquement modulo 2^N), le langage Python modifie **dynamiquement** la taille mémoire allouée au stockage des entiers manipulés pour permettre une représentation et des calculs **exacts**.

Le type `int` dans le langage Python n'est donc pas de taille prédéfinie, mais permet de représenter des entiers arbitrairement longs (leurs tailles n'est en pratique limitée que par la quantité de mémoire disponible sur la machine).

Et en langage Python ?

Dans de nombreux langages, les entiers signés sont codés, comme nous l'avons vu, avec un nombre de bits N fixé une fois pour toutes (généralement 32 ou 64 bits), et c'était encore le cas pour le langage Python jusqu'aux versions 2.xx.

À partir des versions suivantes, pour éviter les problèmes de dépassement (opérations vraies uniquement modulo 2^N), le langage Python modifie **dynamiquement** la taille mémoire allouée au stockage des entiers manipulés pour permettre une représentation et des calculs **exacts**.

Le type `int` dans le langage Python n'est donc pas de taille prédéfinie, mais permet de représenter des entiers arbitrairement longs (leurs tailles n'est en pratique limitée que par la quantité de mémoire disponible sur la machine).

Une difficulté pour les calculs de complexité : le temps de calcul correspondant à un même type d'opération sur des entiers peut fortement dépendre de la taille effective de leurs représentations.

Représentation des réels

Notation scientifique décimale

Rappels
Pour les nombres réels, il est fréquent d'utiliser la notation scientifique.

- $\pi = 3,14159265$
- $c = 2,99792458 \times 10^8$
- $N_A = 6,022142 \times 10^{23}$
- $h = 6,62606957 \times 10^{-34}$

Notation scientifique décimale

Forme générale

On peut écrire tout réel x <u>**non nul**</u> sous la forme suivante :

$$x = (-1)^S \times 10^E \times (d_0 + d_1 \times 10^{-1} + d_2 \times 10^{-2} + \cdots + d_i \times 10^{-i} + \cdots)$$

Notation scientifique décimale

Forme générale
On peut écrire tout réel x **non nul** sous la forme suivante :

$$x = (-1)^S \times 10^E \times (d_0 + d_1 \times 10^{-1} + d_2 \times 10^{-2} + \cdots + d_i \times 10^{-i} + \cdots)$$

- $S = 0$ ou $S = 1$ (fixe le signe de x);

Notation scientifique décimale

Forme générale
On peut écrire tout réel x <u>**non nul**</u> sous la forme suivante :

$$x = (-1)^S \times 10^E \times (d_0 + d_1 \times 10^{-1} + d_2 \times 10^{-2} + \cdots + d_i \times 10^{-i} + \cdots)$$

- $S = 0$ ou $S = 1$ (fixe le signe de x);
- E est un entier relatif;

Notation scientifique décimale

Forme générale
On peut écrire tout réel x <u>**non nul**</u> sous la forme suivante :

$$x = (-1)^S \times 10^E \times (d_0 + d_1 \times 10^{-1} + d_2 \times 10^{-2} + \cdots + d_i \times 10^{-i} + \cdots)$$

- $S = 0$ ou $S = 1$ (fixe le signe de x);
- E est un entier relatif;
- $d_i \in \{0, 1, 2, \cdots, 9\}$ avec le cas particulier $\boxed{d_0 \neq 0}$.

le nombre de décimales peut être infini :

$$\frac{4}{3} = 1,33333\cdots$$

Notation scientifique décimale

Forme générale

On peut écrire tout réel x **non nul** sous la forme suivante :

$$x = (-1)^S \times 10^E \times (d_0 + d_1 \times 10^{-1} + d_2 \times 10^{-2} + \cdots + d_i \times 10^{-i} + \cdots)$$

- $S = 0$ ou $S = 1$ (fixe le signe de x);
- E est un entier relatif;
- $d_i \in \{0, 1, 2, \cdots, 9\}$ avec le cas particulier $\boxed{d_0 \neq 0}$.

le nombre de décimales peut être infini :

$$\frac{4}{3} = 1,33333\cdots$$

ou pas :

$$\frac{5}{4} = 1,25.$$

Notation scientifique binaire

Exemple

On peut étendre en binaire l'écriture de nombres à virgules, ainsi que la notation scientifique. Ainsi :

$$\begin{aligned}(110,01)_2 &= 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}\\ &= 2^2\left(1 + 1\times 2^{-1} + 0\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}\right).\end{aligned}$$

Notation scientifique binaire

Exemple

On peut étendre en binaire l'écriture de nombres à virgules, ainsi que la notation scientifique. Ainsi :

$$\begin{aligned}(110,01)_2 &= 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}\\ &= 2^2\left(1 + 1\times 2^{-1} + 0\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}\right).\end{aligned}$$

Généralisation

On peut écrire tout réel x <u>non nul</u> sous la forme suivante :

Notation scientifique binaire

Exemple

On peut étendre en binaire l'écriture de nombres à virgules, ainsi que la notation scientifique. Ainsi :

$$\begin{aligned}(110,01)_2 &= 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}\\ &= 2^2\left(1 + 1\times 2^{-1} + 0\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}\right).\end{aligned}$$

Généralisation

On peut écrire tout réel x <u>non nul</u> sous la forme suivante :

$$x = (-1)^s \times 2^E \times (b_0 + b_1\times 2^{-1} + b_2\times 2^{-2} + \cdots + b_i\times 2^{-i} + \cdots)$$

Notation scientifique binaire

Exemple

On peut étendre en binaire l'écriture de nombres à virgules, ainsi que la notation scientifique. Ainsi :

$$\begin{aligned}(110,01)_2 &= 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}\\ &= 2^2\left(1 + 1\times 2^{-1} + 0\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}\right).\end{aligned}$$

Généralisation

On peut écrire tout réel x <u>non nul</u> sous la forme suivante :

$$x = (-1)^s \times 2^E \times (b_0 + b_1\times 2^{-1} + b_2\times 2^{-2} + \cdots + b_i\times 2^{-i} + \cdots)$$

- $s = 0$ ou $s = 1$ (fixe le signe de x);

Notation scientifique binaire

Exemple

On peut étendre en binaire l'écriture de nombres à virgules, ainsi que la notation scientifique. Ainsi :

$$\begin{aligned}(110,01)_2 &= 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}\\ &= 2^2\left(1 + 1\times 2^{-1} + 0\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}\right).\end{aligned}$$

Généralisation

On peut écrire tout réel x <u>non nul</u> sous la forme suivante :

$$x = (-1)^s \times 2^E \times (b_0 + b_1\times 2^{-1} + b_2\times 2^{-2} + \cdots + b_i\times 2^{-i} + \cdots)$$

- $s = 0$ ou $s = 1$ (fixe le signe de x);
- E est un entier relatif (choisi de sorte que $1 \leqslant \frac{|x|}{2^E} < 2$);

Notation scientifique binaire

Exemple

On peut étendre en binaire l'écriture de nombres à virgules, ainsi que la notation scientifique. Ainsi :

$$\begin{aligned}(110,01)_2 &= 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}\\ &= 2^2\left(1 + 1\times 2^{-1} + 0\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}\right).\end{aligned}$$

Généralisation

On peut écrire tout réel x <u>non nul</u> sous la forme suivante :

$$x = (-1)^s \times 2^E \times (b_0 + b_1\times 2^{-1} + b_2\times 2^{-2} + \cdots + b_i\times 2^{-i} + \cdots)$$

- $s = 0$ ou $s = 1$ (fixe le signe de x);
- E est un entier relatif (choisi de sorte que $1 \leqslant \frac{|x|}{2^E} < 2$);
- $b_i \in \{0,1\}$ avec le cas particulier $b_0 \neq 0$ (et donc ici $\boxed{b_0 = 1}$).

Notation scientifique binaire

Exemple

On peut étendre en binaire l'écriture de nombres à virgules, ainsi que la notation scientifique. Ainsi :

$$\begin{aligned}(110,01)_2 &= 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}\\ &= 2^2\left(1 + 1\times 2^{-1} + 0\times 2^{-2} + 0\times 2^{-3} + 1\times 2^{-4}\right).\end{aligned}$$

Généralisation

On peut écrire tout réel x <u>non nul</u> sous la forme suivante :

$$x = (-1)^s \times 2^E \times (b_0 + b_1\times 2^{-1} + b_2\times 2^{-2} + \cdots + b_i\times 2^{-i} + \cdots)$$

- $s = 0$ ou $s = 1$ (fixe le signe de x);
- E est un entier relatif (choisi de sorte que $1 \leqslant \frac{|x|}{2^E} < 2$);
- $b_i \in \{0,1\}$ avec le cas particulier $b_0 \neq 0$ (et donc ici $\boxed{b_0 = 1}$).

Pour déterminer x, il faut connaître s, E et l'ensemble des b_i.

Contraintes machine

La taille de la mémoire machine est limitée, on a donc des contraintes :

Contraintes machine

La taille de la mémoire machine est limitée, on a donc des contraintes :

- $\{b_i\}$ en nombre fini ;

Contraintes machine

La taille de la mémoire machine est limitée, on a donc des contraintes :

- $\{b_i\}$ en nombre fini ;
- valeurs de E comprises dans un intervalle fini.

Mantisse

- Définition : Soit
$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \in \mathbb{R}$$
On appelle *mantisse de x sur m bits* ou simplement *mantisse de x* la quantité :
$$M = 1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}.$$

Mantisse

- Définition : Soit
$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \in \mathbb{R}$$
On appelle *mantisse de x sur m bits* ou simplement *mantisse de x* la quantité :
$$M = 1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}.$$

- Par construction $1 \leqslant M < 2$;

Mantisse

- Définition : Soit
$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \in \mathbb{R}$$
On appelle *mantisse de x sur m bits* ou simplement *mantisse de x* la quantité :
$$M = 1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}.$$

- Par construction $1 \leqslant M < 2$;
- $b_0 = 1$ fixé $\Rightarrow$ on mémorise les $\{b_i\}$ uniquement pour $1 \leqslant i \leqslant m$.

Mantisse

- Définition : Soit
$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \in \mathbb{R}$$
On appelle *mantisse de x sur m bits* ou simplement *mantisse de x* la quantité :
$$M = 1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}.$$

- Par construction $1 \leqslant M < 2$;
- $b_0 = 1$ fixé $\Rightarrow$ on mémorise les $\{b_i\}$ uniquement pour $1 \leqslant i \leqslant m$.
- La mantisse est codée sur m bits, et se note en binaire :
$$1, b_1 b_2 \ldots b_m.$$

Mantisse

- Définition : Soit
$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \in \mathbb{R}$$
On appelle *mantisse de x sur m bits* ou simplement *mantisse de x* la quantité :
$$M = 1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}.$$

- Par construction $1 \leqslant M < 2$;
- $b_0 = 1$ fixé $\Rightarrow$ on mémorise les $\{b_i\}$ uniquement pour $1 \leqslant i \leqslant m$.
- La mantisse est codée sur m bits, et se note en binaire :
$$1, b_1 b_2 \ldots b_m.$$

- Qualitativement, plus m est grand, plus le réel x sera représenté précisément en machine.

Exposant

- Nombre de bits utilisés pour coder l'exposant noté e.

Exposant

- Nombre de bits utilisés pour coder l'exposant noté e.
- Intervalle des valeurs de E :

$$E \in [\![-2^{e-1} + 1; 2^{e-1}]\!]$$

Exposant

- Nombre de bits utilisés pour coder l'exposant noté e.
- Intervalle des valeurs de E :

$$E \in [\![-2^{e-1}+1; 2^{e-1}]\!]$$

  Pour information : cet intervalle est différent de celui correspondant au codage d'un entier relatif sur e bits.
- Exposant décalé E' défini par $E' = E + 2^{e-1} - 1$ tel que $E' \in [\![0; 2^e - 1]\!]$, avec $E' = \sum_{i=0}^{e-1} c_i 2^i$, que l'on notera :

$$c_{e-1} \dots c_1 c_0, \quad \text{avec} : c_i \in \{0, 1\}.$$

Bilan : représentation d'un réel

- coder x nécessite $m + e + 1$ bits,

Bilan : représentation d'un réel

- coder x nécessite $m + e + 1$ bits,
- notation générale :

$$\underbrace{s}_{\text{signe}}\ \underbrace{c_{e-1} \cdots c_1 c_0}_{\text{exposant } E'}\ \underbrace{b_1 b_2 \cdots b_m}_{\text{mantisse}}$$

avec

Bilan : représentation d'un réel

- coder x nécessite $m + e + 1$ bits,
- notation générale :

$$\underbrace{s}_{\text{signe}}\ \underbrace{c_{e-1}\cdots c_1 c_0}_{\text{exposant } E'}\ \underbrace{b_1 b_2 \cdots b_m}_{\text{mantisse}}$$

  avec
  - ◇ s : bit de signe,
  - ◇ (c_j) : représentation binaire de $E' = E + 2^{e-1} - 1$,
  - ◇ (b_i) : représentation binaire de la mantisse.
- Valeur de x : $\quad x = (-1)^s \times 2^E \times M$

Bilan : représentation d'un réel

- coder x nécessite $m + e + 1$ bits,
- notation générale :

$$\underbrace{s}_{\text{signe}}\ \underbrace{c_{e-1}\cdots c_1 c_0}_{\text{exposant } E'}\ \underbrace{b_1 b_2 \cdots b_m}_{\text{mantisse}}$$

  avec
  - s : bit de signe,
  - (c_j) : représentation binaire de $E' = E + 2^{e-1} - 1$,
  - (b_i) : représentation binaire de la mantisse.
- Valeur de x : $\quad x = (-1)^s \times 2^E \times M$ une fois les valeurs de s, E, M récupérées (*via* des fonctions de décodage).

Cas particulier : Zéro réel

Le Zéro réel est un cas particulier car il n'est pas représentable sans règle supplémentaire.

Cas particulier : Zéro réel

Le Zéro réel est un cas particulier car il n'est pas représentable sans règle supplémentaire. En effet :

$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \neq 0$$

Cas particulier : Zéro réel

Le Zéro réel est un cas particulier car il n'est pas représentable sans règle supplémentaire. En effet :

$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \neq 0$$

Convention pour le codage du zéro :

- $E' = 0$ (ou $E = -2^{e-1} + 1$),
- pour tout i, $b_i = 0$.

Le bit de signe n'est pas fixé.

Cas particulier : Zéro réel

Le Zéro réel est un cas particulier car il n'est pas représentable sans règle supplémentaire. En effet :

$$x = (-1)^s \times 2^E \times \left(1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_i \times 2^{-i} + \cdots\right) \neq 0$$

Convention pour le codage du zéro :

- $E' = 0$ (ou $E = -2^{e-1} + 1$),
- pour tout i, $b_i = 0$.

Le bit de signe n'est pas fixé.

On a donc deux codages possibles pour le zéro réel :

signe	exposant E'	mantisse
0	$0 \cdots 0$	$0 \cdots 0$
1	$0 \cdots 0$	$0 \cdots 0$

Valeurs particulières de l'exposant

Pour information
Certaines valeurs de l'exposant E' sont réservées à des codages particuliers.

Valeurs particulières de l'exposant

Pour information
Certaines valeurs de l'exposant E' sont réservées à des codages particuliers.

- Le cas $E' = 2^e - 1$ (ou $E = 2^{e-1}$), correspond à des résultats non numériques, comme l'infini (∞), ou comme *NaN* (Not a Number)(comme $\frac{0}{0}$ par exemple).

Valeurs particulières de l'exposant

Pour information
Certaines valeurs de l'exposant E' sont réservées à des codages particuliers.

- Le cas $E' = 2^e - 1$ (ou $E = 2^{e-1}$), correspond à des résultats non numériques, comme l'infini (∞), ou comme *NaN* (Not a Number)(comme $\frac{0}{0}$ par exemple).
- Le cas $E' = 0$ (ou $E = -2^{e-1} + 1$), correspond au cas des nombres dits dénormalisés. Pour ces nombres, on a une nouvelle définition de la mantisse : $M = b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}$ et donc $M \leqslant 1$.

Pour information
Certaines valeurs de l'exposant E' sont réservées à des codages particuliers.

- Le cas $E' = 2^e - 1$ (ou $E = 2^{e-1}$), correspond à des résultats non numériques, comme l'infini (∞), ou comme *NaN* (Not a Number)(comme $\frac{0}{0}$ par exemple).
- Le cas $E' = 0$ (ou $E = -2^{e-1} + 1$), correspond au cas des nombres dits dénormalisés. Pour ces nombres, on a une nouvelle définition de la mantisse : $M = b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}$ et donc $M \leqslant 1$. Le cas de $x = 0$ appartient à cette catégorie de nombre.

Intervalle des exposants correspondants aux nombres normalisés :

Valeurs particulières de l'exposant

Pour information
Certaines valeurs de l'exposant E' sont réservées à des codages particuliers.

- Le cas $E' = 2^e - 1$ (ou $E = 2^{e-1}$), correspond à des résultats non numériques, comme l'infini (∞), ou comme *NaN* (Not a Number)(comme $\frac{0}{0}$ par exemple).
- Le cas $E' = 0$ (ou $E = -2^{e-1} + 1$), correspond au cas des nombres dits dénormalisés. Pour ces nombres, on a une nouvelle définition de la mantisse : $M = b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}$ et donc $M \leqslant 1$. Le cas de $x = 0$ appartient à cette catégorie de nombre.

Intervalle des exposants correspondants aux nombres normalisés :

$$E_{\text{restreint}} \in [\![-2^{e-1} + 2; 2^{e-1} - 1]\!]$$

Résumé

Type d'objets	Nom de bits	Entiers codés	Principe du codage
Entiers naturels n ($\star$)	N	$[\![0\,,\,2^N-1]\!]$	$(c_{N-1}\dots c_0)$ avec $n=\sum_{k=0}^{N-1} c_k 2^k$
Entiers relatifs n	N	$[\![-2^{N-1}\,,\,2^{N-1}-1]\!]$	$n \geqslant 0$: codage ($\star$) de n, $n<0$: codage ($\star$) de $n'=n+2^N-1$
Nombres réels x	$x=(-1)^s\times 2^E\times M,\quad M\in[1,2[$		
	1	s	$+$ codé avec 0, $-$ codé avec 1
	e	$E\in[\![-2^{e-1}+1;2^{e-1}]\!]$	$E'=E+2^{e-1}-1$ codage ($\star$)
	m	M	$1+b_1\times 2^{-1}+b_2\times 2^{-2}+\cdots+b_i\times 2^{-i}+\cdots$, série tronquée sur m bits : $1+b_1\times 2^{-1}+b_2\times 2^{-2}+\cdots+b_m\times 2^{-m}$

Détermination pratique de E et M

Pour x donné, on cherche les valeurs de E et M (valeurs décimales pour l'instant).

Détermination pratique de E et M

Pour x donné, on cherche les valeurs de E et M (valeurs décimales pour l'instant).

Calcul de E et M

Données : Un réel x

Résultat : Exposant E et mantisse M de x

$M \leftarrow |x|$, $E \leftarrow 0$.

- Tant que $M < 1$ faire : $M \leftarrow 2M$ et $E \leftarrow E - 1$.
- Tant que $M \geqslant 2$ faire : $M \leftarrow M/2$ et $E \leftarrow E + 1$.

renvoyer E, M.

Détermination pratique de E et M

Pour x donné, on cherche les valeurs de E et M (valeurs décimales pour l'instant).

Calcul de E et M

Données : Un réel x

Résultat : Exposant E et mantisse M de x

$M \leftarrow |x|$, $E \leftarrow 0$.

- Tant que $M < 1$ faire : $M \leftarrow 2M$ et $E \leftarrow E - 1$.
- Tant que $M \geqslant 2$ faire : $M \leftarrow M/2$ et $E \leftarrow E + 1$.

renvoyer E, M.

On a pour invariant

$$|x| = 2^E M$$

et à la fin de l'algorithme $1 \leq M < 2$.

Exemple de détermination pratique de E et M

Pour $x = 6.28$

M	E
6.28	0

Exemple de détermination pratique de E et M

Pour $x = 6.28$

M	E
6.28	0
3.14	1
1.57	2

Exemple de détermination pratique de E et M

Pour $x = 6.28$

M	E
6.28	0
3.14	1
1.57	2

On a donc $x = 6.28 = 2^2 \times 1.57$

Exemple de détermination pratique de E et M

Pour $x = 0.1$

M	E
0.1	0

Exemple de détermination pratique de E et M

Pour $x = 0.1$

M	E
0.1	0
0.2	-1
0.4	-2
0.8	-3
1.6	-4

Exemple de détermination pratique de E et M

Pour $x = 0.1$

M	E
0.1	0
0.2	-1
0.4	-2
0.8	-3
1.6	-4

On a donc $x = 0.1 = 2^{-4} \times 1.6$

Représentation binaire M

Pour $M \in [1, 2[$, on cherche à déterminer $(b_i)_{1 \leqslant i \leqslant m}$ tel que $M = (1, b_1 b_2 \ldots b_m)_2$.

Représentation binaire M

Pour $M \in [1, 2[$, on cherche à déterminer $(b_i)_{1 \leqslant i \leqslant m}$ tel que $M = (1, b_1 b_2 \ldots b_m)_2$.

- Construction de la suite y_i de premier terme $y_1 = M - 1 = (0, b_1 b_2 \cdots b_m)_2$

Représentation binaire M

Pour $M \in [1, 2[$, on cherche à déterminer $(b_i)_{1 \leqslant i \leqslant m}$ tel que $M = (1, b_1 b_2 \ldots b_m)_2$.

- Construction de la suite y_i de premier terme $y_1 = M - 1 = (0, b_1 b_2 \cdots b_m)_2$
- $y_1 \geqslant 0.5 \Leftrightarrow b_1 = 1$, on pose alors $y_2 = 2y_1 - 1 = (0, b_2 \cdots b_m)_2$

Représentation binaire M

Pour $M \in [1, 2[$, on cherche à déterminer $(b_i)_{1 \leqslant i \leqslant m}$ tel que $M = (1, b_1 b_2 \ldots b_m)_2$.

- Construction de la suite y_i de premier terme $y_1 = M - 1 = (0, b_1 b_2 \cdots b_m)_2$
- $y_1 \geqslant 0.5 \Leftrightarrow b_1 = 1$, on pose alors $y_2 = 2y_1 - 1 = (0, b_2 \cdots b_m)_2$
- $y1 < 0.5 \Leftrightarrow b_1 = 0$, et donc $y_1 = (0, 0b_2 \cdots b_m)_2$. On pose alors $y_2 = 2y_1 = (0, b_2 \cdots b_m)_2$.

Représentation binaire M

Pour $M \in [1, 2[$, on cherche à déterminer $(b_i)_{1 \leqslant i \leqslant m}$ tel que $M = (1, b_1 b_2 \ldots b_m)_2$.

- Construction de la suite y_i de premier terme $y_1 = M - 1 = (0, b_1 b_2 \cdots b_m)_2$
- $y_1 \geqslant 0.5 \Leftrightarrow b_1 = 1$, on pose alors $y_2 = 2y_1 - 1 = (0, b_2 \cdots b_m)_2$
- $y1 < 0.5 \Leftrightarrow b_1 = 0$, et donc $y_1 = (0, 0b_2 \cdots b_m)_2$. On pose alors $y_2 = 2y_1 = (0, b_2 \cdots b_m)_2$.

On peut alors itérer le processus, d'où l'algorithme suivant :

Représentation binaire M

Pour $M \in [1, 2[$, on cherche à déterminer $(b_i)_{1 \leqslant i \leqslant m}$ tel que $M = (1, b_1 b_2 \ldots b_m)_2$.

- Construction de la suite y_i de premier terme $y_1 = M - 1 = (0, b_1 b_2 \cdots b_m)_2$
- $y_1 \geqslant 0.5 \Leftrightarrow b_1 = 1$, on pose alors $y_2 = 2y_1 - 1 = (0, b_2 \cdots b_m)_2$
- $y1 < 0.5 \Leftrightarrow b_1 = 0$, et donc $y_1 = (0, 0b_2 \cdots b_m)_2$. On pose alors $y_2 = 2y_1 = (0, b_2 \cdots b_m)_2$.

On peut alors itérer le processus, d'où l'algorithme suivant :

Calcul de la représentation binaire de la mantisse

Données : Un réel $M \in [1, 2[$

Résultat : La liste b des $(b_i)_{1 \leqslant i \leqslant m}$

$b = []$ et $y = M - 1$

pour i allant de 1 à m faire :

- Si $y \geqslant 0.5$, $b \leftarrow b + [1]$, $y \leftarrow 2y - 1$.
- Sinon, $b \leftarrow b + [0]$, $y \leftarrow 2y$.

renvoyer b.

Exemple de représentation pour $m = 4$

Pour $x = 1.0625$, on a $s = 0$, $E = 0$ et $M = (1.0625)_{10}$. On cherche les (b_i).

Exemple de représentation pour $m = 4$

Pour $x = 1.0625$, on a $s = 0$, $E = 0$ et $M = (1.0625)_{10}$. On cherche les (b_i).

i	y_i	b_i
1	0.0625	

Exemple de représentation pour $m = 4$

Pour $x = 1.0625$, on a $s = 0$, $E = 0$ et $M = (1.0625)_{10}$. On cherche les (b_i).

i	y_i	b_i
1	0.0625	0
2	0.125	0
3	0.25	0
4	0.5	1

Exemple de représentation pour $m = 4$

Pour $x = 1.0625$, on a $s = 0$, $E = 0$ et $M = (1.0625)_{10}$. On cherche les (b_i).

i	y_i	b_i
1	0.0625	0
2	0.125	0
3	0.25	0
4	0.5	1

On a donc $M = (1.0001)_2$.

Exemple de représentation pour $m = 4$

Pour $x = 0.1$, on a $s = 0$, $E = -4$ et $M = (1.6)_{10}$. On cherche les (b_i).

Exemple de représentation pour $m = 4$

Pour $x = 0.1$, on a $s = 0$, $E = -4$ et $M = (1.6)_{10}$. On cherche les (b_i).

i	y_i	b_i
1	0.6	

Exemple de représentation pour $m = 4$

Pour $x = 0.1$, on a $s = 0$, $E = -4$ et $M = (1.6)_{10}$. On cherche les (b_i).

i	y_i	b_i
1	0.6	1
2	0.2	0
3	0.4	0
4	0.8	1
5	0.6	1
	$\vdots$	

Exemple de représentation pour $m = 4$

Pour $x = 0.1$, on a $s = 0$, $E = -4$ et $M = (1.6)_{10}$. On cherche les (b_i).

i	y_i	b_i
1	0.6	1
2	0.2	0
3	0.4	0
4	0.8	1
5	0.6	1
	$\vdots$	

On a donc $M = (1.1001)_2$ avec $m = 4$ bits.

Exemple de représentation pour $m = 4$

Pour $x = 0.1$, on a $s = 0$, $E = -4$ et $M = (1.6)_{10}$. On cherche les (b_i).

i	y_i	b_i
1	0.6	1
2	0.2	0
3	0.4	0
4	0.8	1
5	0.6	1
	$\vdots$	

On a donc $M = (1.1001)_2$ avec $m = 4$ bits.

Rem : la représentation de $(1.6)_{10}$, finie en base 10, est infinie en base 2.

La norme IEEE754

Nécessité d'imposer une norme commune pour le partage des informations entre différentes machine ou programmes.

La norme IEEE754

Nécessité d'imposer une norme commune pour le partage des informations entre différentes machine ou programmes.

Principe

- Codage des réels sur 64 bits. $e + m + 1 = 64$

La norme IEEE754

Nécessité d'imposer une norme commune pour le partage des informations entre différentes machine ou programmes.

Principe

- Codage des réels sur 64 bits. $e + m + 1 = 64$
- 11 bits pour l'exposant. On a donc $e = 11$

La norme IEEE754

Nécessité d'imposer une norme commune pour le partage des informations entre différentes machine ou programmes.

Principe

- Codage des réels sur 64 bits. $e + m + 1 = 64$
- 11 bits pour l'exposant. On a donc $e = 11$
- 52 bits pour la mantisse. On a donc $m = 52$

La norme IEEE754

Nécessité d'imposer une norme commune pour le partage des informations entre différentes machine ou programmes.

Principe

- Codage des réels sur 64 bits. $e + m + 1 = 64$
- 11 bits pour l'exposant. On a donc $e = 11$
- 52 bits pour la mantisse. On a donc $m = 52$

On se limite dans cette partie à la description des nombres **normalisés**.

Intervalle représenté

- Comme pour les entiers, l'intervalle des réels représentés est forcément limité.

Intervalle représenté

- Comme pour les entiers, l'intervalle des réels représentés est forcément limité.
- C'est l'intervalle des valeurs de E qui limite principalement cet intervalle.

Intervalle représenté

- Comme pour les entiers, l'intervalle des réels représentés est forcément limité.
- C'est l'intervalle des valeurs de E qui limite principalement cet intervalle.

Précision

- Tout intervalle contient une infinité de réels.

Limitations

Intervalle représenté

- Comme pour les entiers, l'intervalle des réels représentés est forcément limité.
- C'est l'intervalle des valeurs de E qui limite principalement cet intervalle.

Précision

- Tout intervalle contient une infinité de réels.
- Représentation approchée des réels.

Limitations

Intervalle représenté

- Comme pour les entiers, l'intervalle des réels représentés est forcément limité.
- C'est l'intervalle des valeurs de E qui limite principalement cet intervalle.

Précision

- Tout intervalle contient une infinité de réels.
- Représentation approchée des réels.
  La taille finie de la mantisse impose un intervalle minimal entre deux réels représentés successifs.

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

$\rightarrow$ On a $E_{\text{normalisé}} \in [\![-2^{10}+2; 2^{10}-1]\!] = [\![-1022; 1023]\!]$.

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

$\rightarrow$ On a $E_{\text{normalisé}} \in [\![-2^{10}+2; 2^{10}-1]\!] = [\![-1022; 1023]\!]$.

- Déterminer une estimation du plus grand réel représentable, et donner sa valeur numérique sous la forme $r \times 10^n$ avec n entier et $|r| < 10$.

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

$\rightarrow$ On a $E_{\text{normalisé}} \in [\![-2^{10}+2; 2^{10}-1]\!] = [\![-1022; 1023]\!]$.

- Déterminer une estimation du plus grand réel représentable, et donner sa valeur numérique sous la forme $r \times 10^n$ avec n entier et $|r| < 10$.

$\rightarrow$ On a $x_{max} = 2^{E_{\max}} \times M_{max}$ avec $M_{max} \simeq 2$.

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

$\rightarrow$ On a $E_{\text{normalisé}} \in [\![-2^{10}+2; 2^{10}-1]\!] = [\![-1022; 1023]\!]$.

- Déterminer une estimation du plus grand réel représentable, et donner sa valeur numérique sous la forme $r \times 10^n$ avec n entier et $|r| < 10$.

$\rightarrow$ On a $x_{max} = 2^{E_{\max}} \times M_{max}$ avec $M_{max} \simeq 2$.
On a donc $x_{max} = 2^{p_{max}}$ avec $p_{max} \simeq 1024$.

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

$\rightarrow$ On a $E_{\text{normalisé}} \in [\![-2^{10}+2; 2^{10}-1]\!] = [\![-1022; 1023]\!]$.

- Déterminer une estimation du plus grand réel représentable, et donner sa valeur numérique sous la forme $r \times 10^n$ avec n entier et $|r| < 10$.

$\rightarrow$ On a $x_{max} = 2^{E_{\max}} \times M_{max}$ avec $M_{max} \simeq 2$.
On a donc $x_{max} = 2^{p_{max}}$ avec $p_{max} \simeq 1024$.
On cherche $2^{p_{max}} = 10^{n+\epsilon}$

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

$\rightarrow$ On a $E_{\text{normalisé}} \in [\![-2^{10}+2; 2^{10}-1]\!] = [\![-1022; 1023]\!]$.

- Déterminer une estimation du plus grand réel représentable, et donner sa valeur numérique sous la forme $r \times 10^n$ avec n entier et $|r| < 10$.

$\rightarrow$ On a $x_{max} = 2^{E_{\max}} \times M_{max}$ avec $M_{max} \simeq 2$.
On a donc $x_{max} = 2^{p_{max}}$ avec $p_{max} \simeq 1024$.
On cherche $2^{p_{max}} = 10^{n+\epsilon}$
d'où $n + \epsilon = p_{max} \frac{ln(2)}{ln(10)}$.

Intervalle Représenté

On se limite ici aux réels positifs.

- Donner numériquement l'intervalle des exposants correspondant aux nombres normalisés.

$\rightarrow$ On a $E_{\text{normalisé}} \in [\![-2^{10}+2; 2^{10}-1]\!] = [\![-1022; 1023]\!]$.

- Déterminer une estimation du plus grand réel représentable, et donner sa valeur numérique sous la forme $r \times 10^n$ avec n entier et $|r| < 10$.

$\rightarrow$ On a $x_{max} = 2^{E_{\max}} \times M_{max}$ avec $M_{max} \simeq 2$.

On a donc $x_{max} = 2^{p_{max}}$ avec $p_{max} \simeq 1024$.

On cherche $2^{p_{max}} = 10^{n+\epsilon}$

d'où $n + \epsilon = p_{max} \frac{ln(2)}{ln(10)}$.

On trouve à la calculatrice, $n = 308, \epsilon = 0,25$, d'où : $\boxed{x_{max} = 1,8 \times 10^{308}}$

Précision absolue
On s'intéresse à la différence entre deux nombres réels successifs x et x' représentés.

Précision absolue
On s'intéresse à la différence entre deux nombres réels successifs x et x' représentés.

- Déterminer la plus petite variation de mantisse δM possible.

Précision absolue
On s'intéresse à la différence entre deux nombres réels successifs x et x' représentés.

- Déterminer la plus petite variation de mantisse δM possible.

$\rightarrow$ On a $\delta M = 2^{-m} = 2^{-52}$.

Précision absolue
On s'intéresse à la différence entre deux nombres réels successifs x et x' représentés.

- Déterminer la plus petite variation de mantisse δM possible.

$\rightarrow$ On a $\delta M = 2^{-m} = 2^{-52}$.

- En déduire $\delta x = |x' - x|$ pour les plus petits nombres représentés, et pour les plus grands.

Précision absolue
On s'intéresse à la différence entre deux nombres réels successifs x et x' représentés.

- Déterminer la plus petite variation de mantisse δM possible.

$\rightarrow$ On a $\delta M = 2^{-m} = 2^{-52}$.

- En déduire $\delta x = |x' - x|$ pour les plus petits nombres représentés, et pour les plus grands.

$\rightarrow$ Avec $x = (-1)^s \times 2^E \times M$, on a $\delta x = 2^{E-m}$.

Précision absolue
On s'intéresse à la différence entre deux nombres réels successifs x et x' représentés.

- Déterminer la plus petite variation de mantisse δM possible.

$\rightarrow$ On a $\delta M = 2^{-m} = 2^{-52}$.

- En déduire $\delta x = |x' - x|$ pour les plus petits nombres représentés, et pour les plus grands.

$\rightarrow$ Avec $x = (-1)^s \times 2^E \times M$, on a $\delta x = 2^{E-m}$.
Pour les plus petits nombres, $E = -1022$, d'où $\delta x = 2^{-1074} \simeq 10^{-323} << 1$.

Précision absolue
On s'intéresse à la différence entre deux nombres réels successifs x et x' représentés.

- Déterminer la plus petite variation de mantisse δM possible.

$\rightarrow$ On a $\delta M = 2^{-m} = 2^{-52}$.

- En déduire $\delta x = |x' - x|$ pour les plus petits nombres représentés, et pour les plus grands.

$\rightarrow$ Avec $x = (-1)^s \times 2^E \times M$, on a $\delta x = 2^{E-m}$.
Pour les plus petits nombres, $E = -1022$, d'où $\delta x = 2^{-1074} \simeq 10^{-323} << 1$.
Pour les plus grands nombres, $E = 1023$, d'où $\delta x = 2^{971} \simeq 10^{292} >> 1$.

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

$$\epsilon = \frac{|x' - x|}{|x|}$$

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

$$\epsilon = \frac{|x' - x|}{|x|}$$

- Déterminer les valeurs possibles de ϵ pour les nombres normalisés.

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

$$\epsilon = \frac{|x' - x|}{|x|}$$

- Déterminer les valeurs possibles de ϵ pour les nombres normalisés.

$\rightarrow$ On a directement $\epsilon = \frac{2^{E-m}}{2^E M} = \frac{2^{-m}}{M}$.

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

$$\epsilon = \frac{|x' - x|}{|x|}$$

- Déterminer les valeurs possibles de ϵ pour les nombres normalisés.

$\rightarrow$ On a directement $\epsilon = \frac{2^{E-m}}{2^E M} = \frac{2^{-m}}{M}$.
Avec $1 \leqslant M < 2$, on a donc $2^{-m-1} \leqslant \epsilon < 2^{-m}$.

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

$$\epsilon = \frac{|x' - x|}{|x|}$$

- Déterminer les valeurs possibles de ϵ pour les nombres normalisés.

$\rightarrow$ On a directement $\epsilon = \frac{2^{E-m}}{2^E M} = \frac{2^{-m}}{M}$.
Avec $1 \leqslant M < 2$, on a donc $2^{-m-1} \leqslant \epsilon < 2^{-m}$.

- En déduire le nombre de chiffres significatifs correctement représentés pour les réels normalisés lorsqu'ils sont écrits sous la forme $r \times 10^n$.

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

$$\epsilon = \frac{|x' - x|}{|x|}$$

- Déterminer les valeurs possibles de ϵ pour les nombres normalisés.

$\rightarrow$ On a directement $\epsilon = \frac{2^{E-m}}{2^E M} = \frac{2^{-m}}{M}$.
Avec $1 \leqslant M < 2$, on a donc $2^{-m-1} \leqslant \epsilon < 2^{-m}$.

- En déduire le nombre de chiffres significatifs correctement représentés pour les réels normalisés lorsqu'ils sont écrits sous la forme $r \times 10^n$.

$\rightarrow$ On a $2^{-52} \simeq 10^{-16}$.

Précision relative
On définit la précision entre ces deux mêmes nombres normalisés successifs relative par :

$$\epsilon = \frac{|x' - x|}{|x|}$$

- Déterminer les valeurs possibles de ϵ pour les nombres normalisés.

$\rightarrow$ On a directement $\epsilon = \frac{2^{E-m}}{2^E M} = \frac{2^{-m}}{M}$.
Avec $1 \leqslant M < 2$, on a donc $2^{-m-1} \leqslant \epsilon < 2^{-m}$.

- En déduire le nombre de chiffres significatifs correctement représentés pour les réels normalisés lorsqu'ils sont écrits sous la forme $r \times 10^n$.

$\rightarrow$ On a $2^{-52} \simeq 10^{-16}$.

On a donc 16 chiffres significatifs correctement représentés, et ceci dans tout l'intervalle des réels représentés.

Précision de la représentation

Répartition des nombres représentés sur la droite réelle

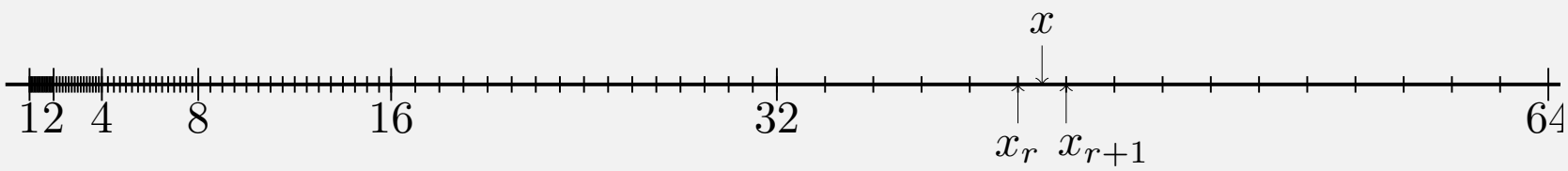
- Dans $[2^p, 2^{p+1}[$, on a des réels représentés distants de 2^{p-m}.

Répartition des nombres représentés sur la droite réelle

- Dans $[2^p, 2^{p+1}[$, on a des réels représentés distants de 2^{p-m}.
- Pour p petit, deux nombres successifs sont plus proches que pour p grand.

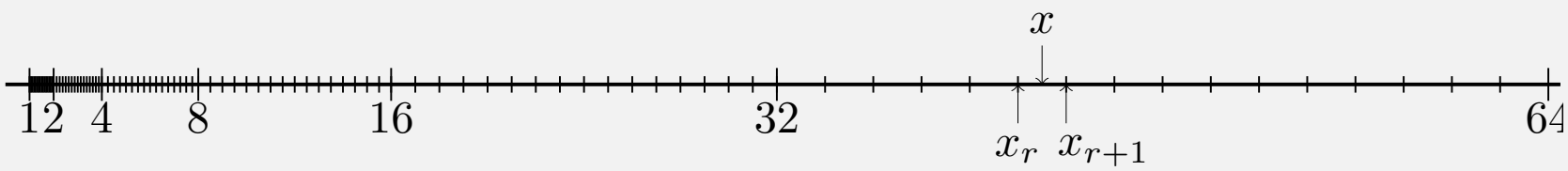
Répartition des nombres représentés sur la droite réelle

- Dans $[2^p, 2^{p+1}[$, on a des réels représentés distants de 2^{p-m}.
- Pour p petit, deux nombres successifs sont plus proches que pour p grand.
- Ci-dessous, on a pris $m = 4$, ce qui fait $2^4 = 16$ réels représentés dans chaque intervalle $[2^p; 2^{p+1}[$ (soit une densité de $\frac{16}{2^p}$).



Répartition des nombres représentés sur la droite réelle

- Dans $[2^p, 2^{p+1}[$, on a des réels représentés distants de 2^{p-m}.
- Pour p petit, deux nombres successifs sont plus proches que pour p grand.
- Ci-dessous, on a pris $m = 4$, ce qui fait $2^4 = 16$ réels représentés dans chaque intervalle $[2^p; 2^{p+1}[$ (soit une densité de $\frac{16}{2^p}$).



- Lorsque x est entre deux réels représentés consécutifs : $x \in [x_r, x_{r+1}[$, alors x est représenté par x_r ou x_{r+1} suivant la convention sur l'arrondi.

Somme de deux nombres très différents : phénomène d'absorption

- soit $x = 1$ et $y = 2^{-54}$. Calcul de $x + y$ en machine ?

Somme de deux nombres très différents : phénomène d'absorption

- soit $x = 1$ et $y = 2^{-54}$. Calcul de $x + y$ en machine ?

$\rightarrow$ On a $x + y = 1 + \frac{1}{2^{-54}}$, mais $m = 52$, $x + y$ est représenté en machine par 1. Donc pour la machine $x + y = x$!

Somme de deux nombres très différents : phénomène d'absorption

- soit $x = 1$ et $y = 2^{-54}$. Calcul de $x + y$ en machine ?

$\rightarrow$ On a $x + y = 1 + \frac{1}{2^{-54}}$, mais $m = 52$, $x + y$ est représenté en machine par 1. Donc pour la machine $x + y = x$!

- On a perdu l'information sur le plus petit nombre.

Différence de deux nombres proches : phénomène de cancellation

- Soient $x = 1$ et $y = 1 + \frac{1}{2^4} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^8}$,

Différence de deux nombres proches : phénomène de cancellation

- Soient $x = 1$ et $y = 1 + \frac{1}{2^4} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^8}$,
- représentations binaires exactes de x, y et $y - x$:

	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
$y =$	1,	0	0	0	1	0	1	1	1
$x =$	1,	0	0	0	0	0	0	0	0
$y - x =$	0,	0	0	0	1	0	1	1	1

Différence de deux nombres proches : phénomène de cancellation

- Soient $x = 1$ et $y = 1 + \frac{1}{2^4} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^8}$,
- représentations binaires exactes de x, y et $y - x$:

	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
$y =$	1,	0	0	0	1	0	1	1	1
$x =$	1,	0	0	0	0	0	0	0	0
$y - x =$	0,	0	0	0	1	0	1	1	1

- Avec une mantisse de 4 bits, on a $y_r = 2^0 \times 1,\underline{0001}$ et $x_r = 2^0 \times 1,\underline{0000}$ et donc $y_r - x_r = 2^{-4} \times 1,\underline{0000}$, alors que $y - x$ admet une représentation exacte $y - x = 2^{-4} \times 1,\underline{0111}$, on voit ainsi qu'il y a eu une **perte de chiffres significatifs**.

Test d'égalité
Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Test d'égalité
Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Test d'égalité

Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Raisonnement en se limitant à une mantisse de $m = 4$ bits :

Test d'égalité

Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Raisonnement en se limitant à une mantisse de $m = 4$ bits :

- pour x_r, $E = -4$, $M = 1.6$, d'où $x_r = 2^{-4} \times 1,1001 = 2^{-4}(1 + 2^{-1} + 2^{-4})$;

Test d'égalité

Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Raisonnement en se limitant à une mantisse de $m = 4$ bits :

- pour x_r, $E = -4$, $M = 1.6$, d'où $x_r = 2^{-4} \times 1,1001 = 2^{-4}(1 + 2^{-1} + 2^{-4})$;
- pour y_r, $E = -3$, $M = 1.6$, d'où $y_r = 2^{-3} \times 1,1001 = 2^{-3}(1 + 2^{-1} + 2^{-4})$;

Test d'égalité

Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Raisonnement en se limitant à une mantisse de $m = 4$ bits :

- pour x_r, $E = -4$, $M = 1.6$, d'où $x_r = 2^{-4} \times 1,1001 = 2^{-4}(1 + 2^{-1} + 2^{-4})$;
- pour y_r, $E = -3$, $M = 1.6$, d'où $y_r = 2^{-3} \times 1,1001 = 2^{-3}(1 + 2^{-1} + 2^{-4})$;
- pour z_r, $E = -2$, $M = 1.2$, d'où $z_r = 2^{-2} \times 1,0011 = 2^{-2}(1 + 2^{-3} + 2^{-4})$;

Test d'égalité

Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Raisonnement en se limitant à une mantisse de $m = 4$ bits :

- pour x_r, $E = -4$, $M = 1.6$, d'où $x_r = 2^{-4} \times 1,1001 = 2^{-4}(1 + 2^{-1} + 2^{-4})$;
- pour y_r, $E = -3$, $M = 1.6$, d'où $y_r = 2^{-3} \times 1,1001 = 2^{-3}(1 + 2^{-1} + 2^{-4})$;
- pour z_r, $E = -2$, $M = 1.2$, d'où $z_r = 2^{-2} \times 1,0011 = 2^{-2}(1 + 2^{-3} + 2^{-4})$;
- On écrit x_r et y_r avec le même exposant :
  - $x_r = 2^{-4}(1 + 2^{-1} + 2^{-4}) = 2^{-3}(2^{-1} + 2^{-2} + 2^{-5})$ (décalage à droite des bits de la mantisse de x_r);

Test d'égalité

Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Raisonnement en se limitant à une mantisse de $m = 4$ bits :

- pour x_r, $E = -4$, $M = 1.6$, d'où $x_r = 2^{-4} \times 1,1001 = 2^{-4}(1 + 2^{-1} + 2^{-4})$;
- pour y_r, $E = -3$, $M = 1.6$, d'où $y_r = 2^{-3} \times 1,1001 = 2^{-3}(1 + 2^{-1} + 2^{-4})$;
- pour z_r, $E = -2$, $M = 1.2$, d'où $z_r = 2^{-2} \times 1,0011 = 2^{-2}(1 + 2^{-3} + 2^{-4})$;
- On écrit x_r et y_r avec le même exposant :
  - $x_r = 2^{-4}(1 + 2^{-1} + 2^{-4}) = 2^{-3}(2^{-1} + 2^{-2} + 2^{-5})$ (décalage à droite des bits de la mantisse de x_r);
  - $x_r + y_r = 2^{-3}(2 + 2^{-2} + 2^{-4}) = 2^{-2}(1 + 2^{-3} + 2^{-5})$;

Test d'égalité

Soient $x = 0.1$; $y = 0.2$ et $z = 0.3$.

Résultat du test `x+y == z` ?

Raisonnement en se limitant à une mantisse de $m = 4$ bits :

- pour x_r, $E = -4$, $M = 1.6$, d'où $x_r = 2^{-4} \times 1,1001 = 2^{-4}(1 + 2^{-1} + 2^{-4})$;
- pour y_r, $E = -3$, $M = 1.6$, d'où $y_r = 2^{-3} \times 1,1001 = 2^{-3}(1 + 2^{-1} + 2^{-4})$;
- pour z_r, $E = -2$, $M = 1.2$, d'où $z_r = 2^{-2} \times 1,0011 = 2^{-2}(1 + 2^{-3} + 2^{-4})$;
- On écrit x_r et y_r avec le même exposant :
  - $x_r = 2^{-4}(1 + 2^{-1} + 2^{-4}) = 2^{-3}(2^{-1} + 2^{-2} + 2^{-5})$ (décalage à droite des bits de la mantisse de x_r);
  - $x_r + y_r = 2^{-3}(2 + 2^{-2} + 2^{-4}) = 2^{-2}(1 + 2^{-3} + 2^{-5})$;
- Soit avec $m = 4$: $x_r + y_r = 2^{-2} \times 1,0010 \neq z_r = 2^{-2} \times 1,0011$.

Recommandations

- Ne pas additionner des nombres de valeur absolue trop différentes.

Recommandations

- Ne pas additionner des nombres de valeur absolue trop différentes.
- Ne pas soustraire des nombres trop proches.

Recommandations

- Ne pas additionner des nombres de valeur absolue trop différentes.
- Ne pas soustraire des nombres trop proches.
- Ne pas utiliser le test d'égalité.

Précision de la représentation

Recommandations

- Ne pas additionner des nombres de valeur absolue trop différentes.
- Ne pas soustraire des nombres trop proches.
- Ne pas utiliser le test d'égalité.

Compromis Précision – Intervalle représenté
On travaille à $e + m$ constant.

Précision de la représentation

Recommandations

- Ne pas additionner des nombres de valeur absolue trop différentes.
- Ne pas soustraire des nombres trop proches.
- Ne pas utiliser le test d'égalité.

Compromis Précision – Intervalle représenté
On travaille à $e + m$ constant.

- Pour augmenter la précision, il faut augmenter m, mais alors e diminue et on peut représenter moins d'entiers.

Recommandations

- Ne pas additionner des nombres de valeur absolue trop différentes.
- Ne pas soustraire des nombres trop proches.
- Ne pas utiliser le test d'égalité.

Compromis Précision – Intervalle représenté
On travaille à $e + m$ constant.

- Pour augmenter la précision, il faut augmenter m, mais alors e diminue et on peut représenter moins d'entiers.
- Pour augmenter la portée (i.e. la taille de l'intervalle représenté), il faut augmenter e, mais alors m diminue et la précision également.

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).
  - ◇ vitesse horizontale v stockée dans un entier signé sur $N = 16$ bits,

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).
  - vitesse horizontale v stockée dans un entier signé sur $N = 16$ bits,
  - tous les tests des programmes réussis (mais avec les données d'Ariane 4),

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).
  - vitesse horizontale v stockée dans un entier signé sur $N = 16$ bits,
  - tous les tests des programmes réussis (mais avec les données d'Ariane 4),
  - pour Ariane 5, $v > 32767 = 2^{15} - 1$, et problèmes de dépassement de capacité, entrainant le crash.

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).
  - vitesse horizontale v stockée dans un entier signé sur $N = 16$ bits,
  - tous les tests des programmes réussis (mais avec les données d'Ariane 4),
  - pour Ariane 5, $v > 32767 = 2^{15} - 1$, et problèmes de dépassement de capacité, entrainant le crash.
- Missile Patriot : en 1991, pendant la Guerre du Golfe, échec d'interception d'un missile Scud irakien par un missiles Patriot.

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).
  - vitesse horizontale v stockée dans un entier signé sur $N = 16$ bits,
  - tous les tests des programmes réussis (mais avec les données d'Ariane 4),
  - pour Ariane 5, $v > 32767 = 2^{15} - 1$, et problèmes de dépassement de capacité, entrainant le crash.
- Missile Patriot : en 1991, pendant la Guerre du Golfe, échec d'interception d'un missile Scud irakien par un missiles Patriot.
  - Temps t mesuré en en 1/10 de seconde,

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).
  - vitesse horizontale v stockée dans un entier signé sur $N = 16$ bits,
  - tous les tests des programmes réussis (mais avec les données d'Ariane 4),
  - pour Ariane 5, $v > 32767 = 2^{15} - 1$, et problèmes de dépassement de capacité, entrainant le crash.
- Missile Patriot : en 1991, pendant la Guerre du Golfe, échec d'interception d'un missile Scud irakien par un missiles Patriot.
  - Temps t mesuré en en 1/10 de seconde,
  - 1/10 n'a pas d'écriture finie en binaire (0,000110011...),

Exemples célèbres d'erreurs

- Crash d'Ariane 5 (1996).
  - vitesse horizontale v stockée dans un entier signé sur $N = 16$ bits,
  - tous les tests des programmes réussis (mais avec les données d'Ariane 4),
  - pour Ariane 5, $v > 32767 = 2^{15} - 1$, et problèmes de dépassement de capacité, entrainant le crash.
- Missile Patriot : en 1991, pendant la Guerre du Golfe, échec d'interception d'un missile Scud irakien par un missiles Patriot.
  - Temps t mesuré en en 1/10 de seconde,
  - 1/10 n'a pas d'écriture finie en binaire (0,000110011...),
  - Erreurs de calculs dus à la limitation de la taille de la mantisse.