



Représentation des nombres flottants

Systèmes de numérotation positionnels

- **Nombres à virgule**
- Avant la virgule : des puissances positives de la base
- Après la virgule : des puissances négatives
- **Exemples**
- $23.25_{10} = 2 \times 10^1 + 3 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$
- $1010.01_2 = 2^3 + 2^1 + 2^{-2}$



10^1	10^0		10^{-1}	10^{-2}
2	3	.	2	5

Nombres à virgule, Notation exponentielle

- Représentations équivalentes dans la base 10 du nombre 1.234

$$123.400 \times 10^{-2}$$

$$12.3400 \times 10^{-1}$$

$$1.23400 \times 10^0$$

$$0.12340 \times 10^1$$

$$0.01234 \times 10^2$$

$$0.001234 \times 10^3$$

Le point décimal "flotte"
(ajustement approprié de
l'exposant).

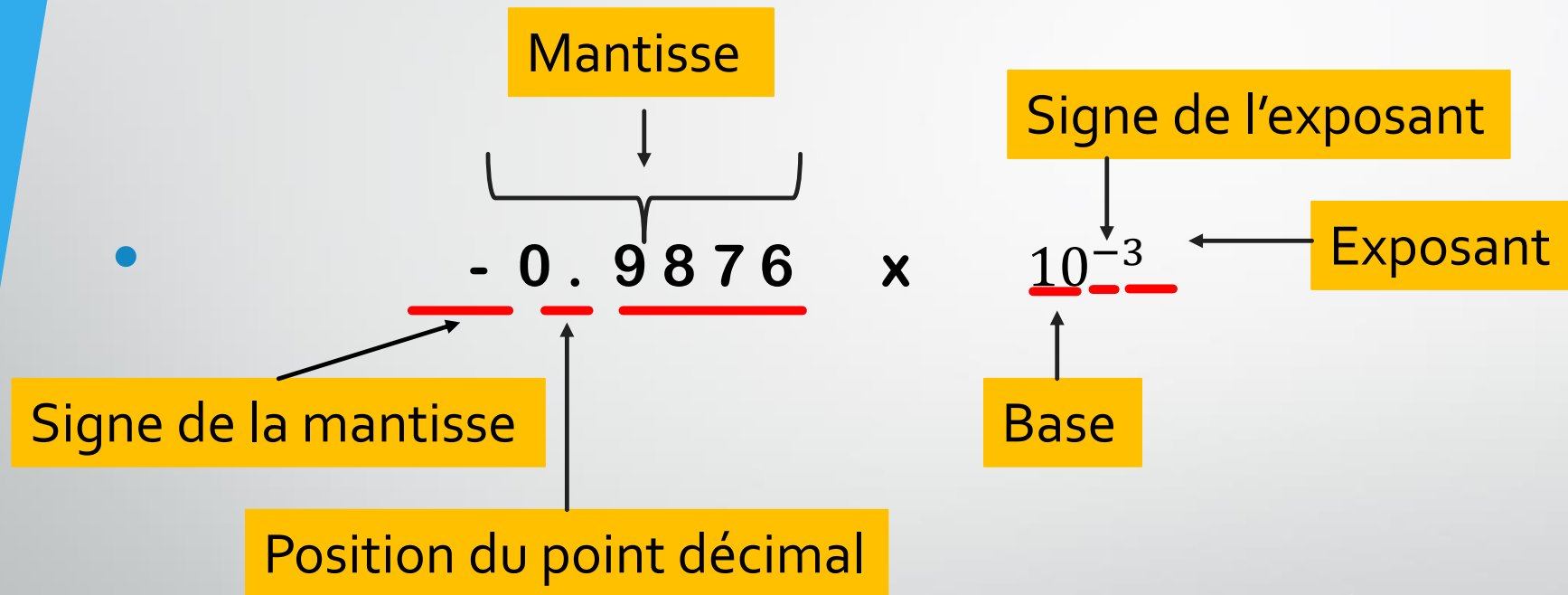
Représentation des nombres



- **Entiers**
 - Entiers en point fixe
 - Signés ou non signés
- **Flottants**
 - La place de la virgule n'est pas fixe
- **Autres**
 - Décimal Codé Binaire(historique)
 - Gros entiers (langages évoluées, logiciels spécialisés)
 - Représentation symbolique (idem)

Éléments de la notation exponentielle

$$\pm M_1.M_2\dots \times X^{\pm c}, X = \textit{base}$$



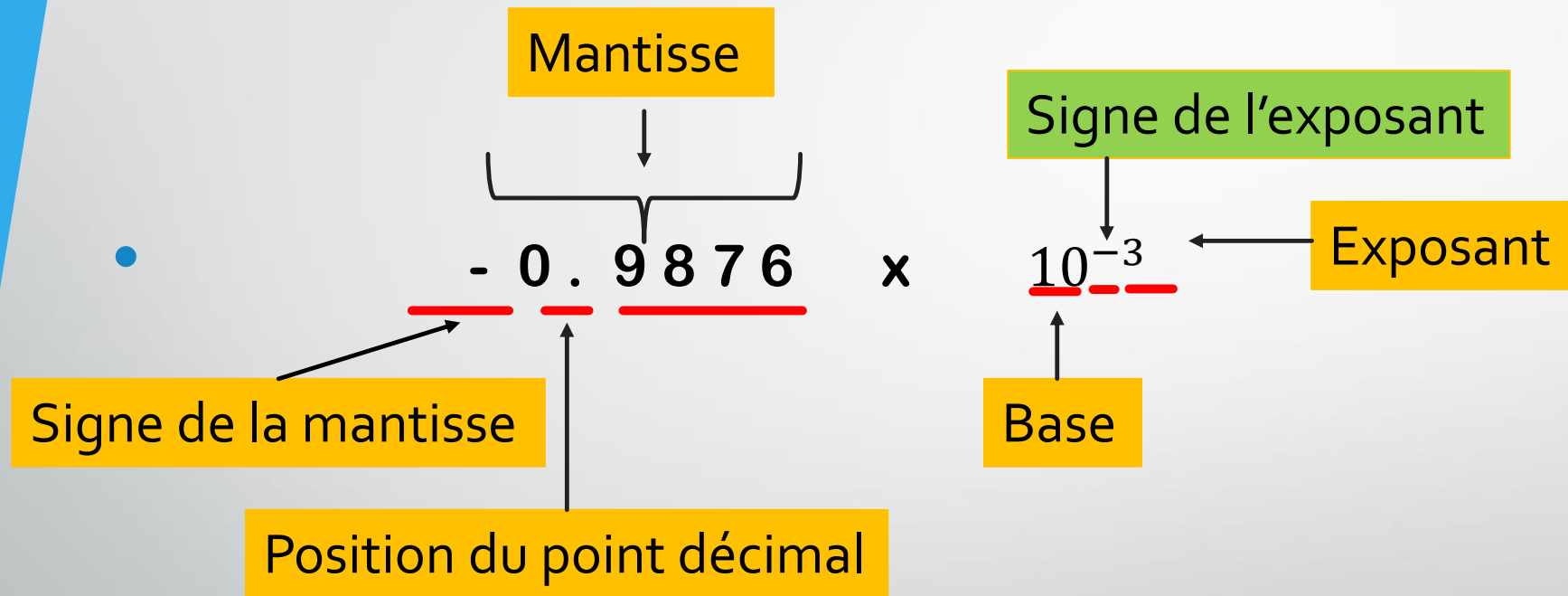
Représentation normalisée



- Un nombre représenté en virgule flottante est normalisé s'il est sous la forme: $\pm 0.M \times X^{\pm C}$
- M – un nombre dont le premier chiffre est non nul, Mantisse
- Exemple:
 - $+ 59,4151 \times 10^{-5} \Rightarrow$ Normalisé: $+0,594151 \times 10^{-3}$
- Il s'agit de représenter la mantisse et son signe, ainsi que l'exposant et son signe

Représentation de l'exposant et de son signe

$$\pm M_1.M_2\dots \times X^{\pm c}, X = \text{base}$$

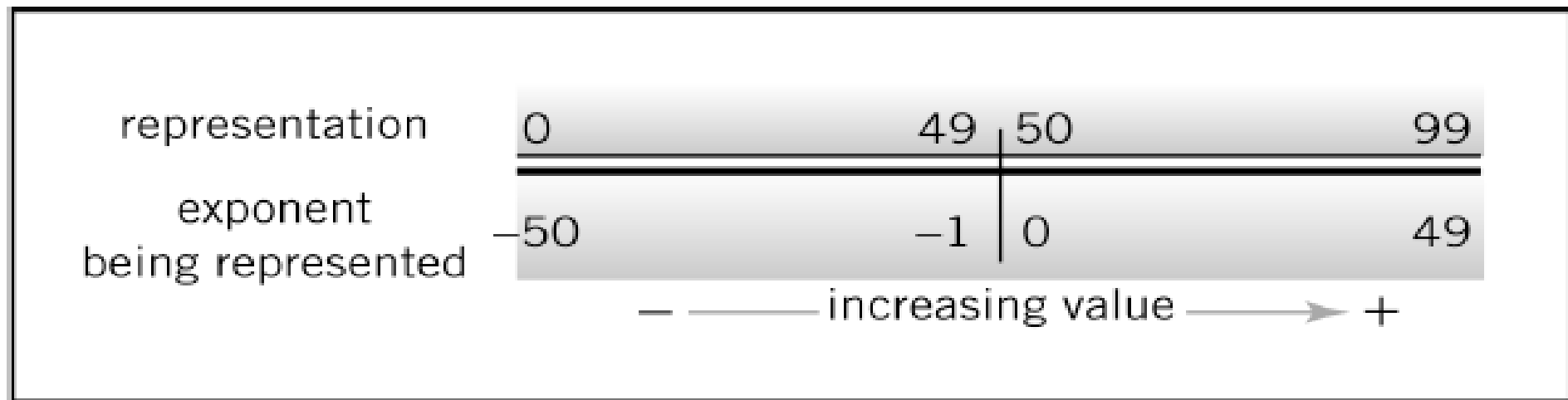


Représentation de l'exposant et de son signe

- Exemple: format en base 10
- -0.9876×10^{-3}
- L'exposant est traduite de manière à toujours coder en interne une valeur positive
- Avec 2 digits réservés au codage de l'exposant: les valeurs positives: [+0, +99]
 - En appliquant une translation $k=50$: les exposants représentables $\Rightarrow [-50, 49]$
 - La constante k est appelée **constante d'excentrement** (**pôle, biais**)

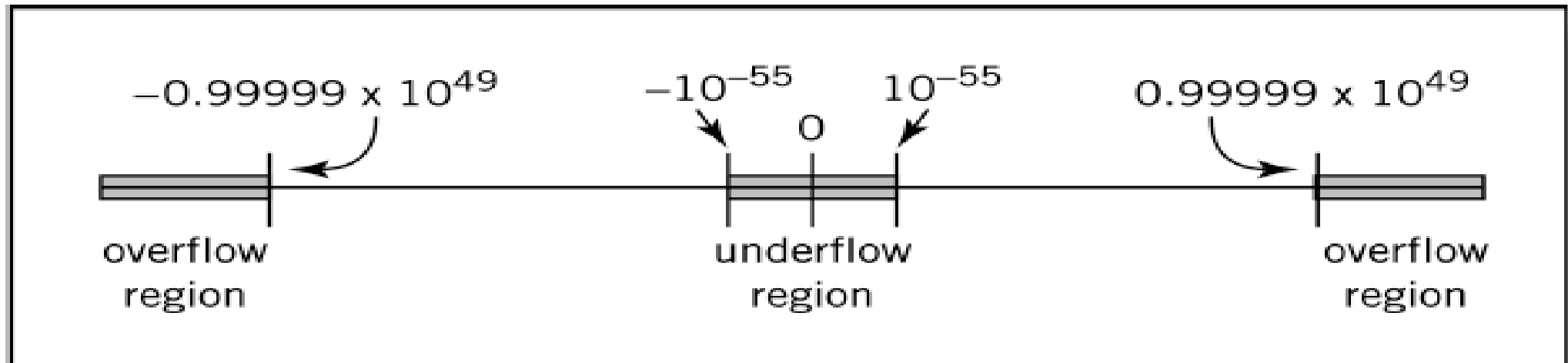
Représentation en virgule flottante

- Avec 2 digits réservés au codage de l'exposant avec un excentrement égal à 50_{10} et 5 digits pour la mantisse on peut représenter de $0.00001 * 10^{-50}$ à $0.99999 * 10^{49}$

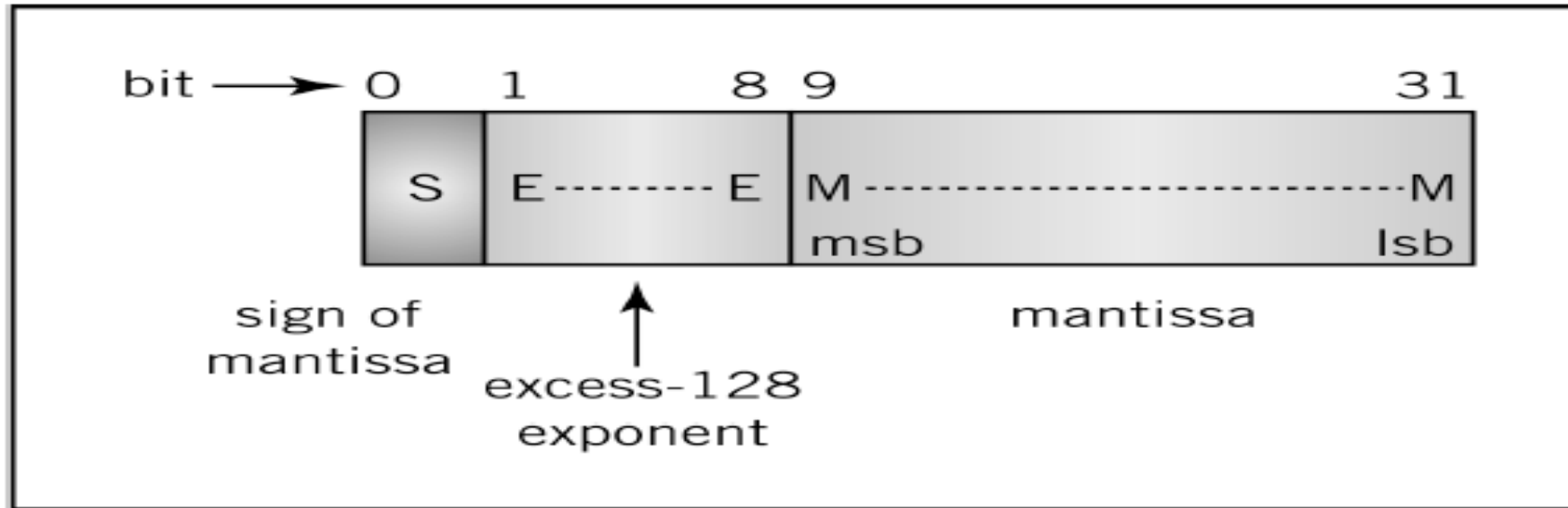


Overflows / Underflows

- De $.000001 \times 10^{-50}$ à $.999999 \times 10^{49}$
 1×10^{-55} à $.999999 \times 10^{49}$




Format typique



Englander: The Architecture of Computer
Hardware and Systems Software, 2nd edition
Chapter 5, Figure 05-04

Conversion d'un nombre fractionnaire

- Conversion de la partie fractionnaire s'effectue en multipliant cette partie fractionnaire par X
- Multiplication est itérée sur la partie fractionnaire du résultat obtenu 
- Conversion de la partie fractionnaire du nombre N est obtenue par la suite des parties entières de chacun des résultats des multiplications effectuées
- Développement s'arrête lorsque la précision voulue est obtenue

Norme IEEE 754

- **Forme normalisée en base 2**
 - Forme normalisé selon $1.f \times 2^e$
 - f est la partie fractionnaire (ou mantisse)
 - e est l'exposant de la puissance de 2
- **Attention**
 - Le premier 1 n'est pas stocké mais est présent
 - Un bit supplémentaire stocke le signe (pas de représentation en complément)
 - Deux représentations : simple précision et double précision



Simple précision (32 bits) IEEE-754

- **Signe (1 bit)**

- 0 = positif
- 1 = négatif

Exposant (8 bits)

00_{16} et FF_{16} : encodage spécial (plus tard)

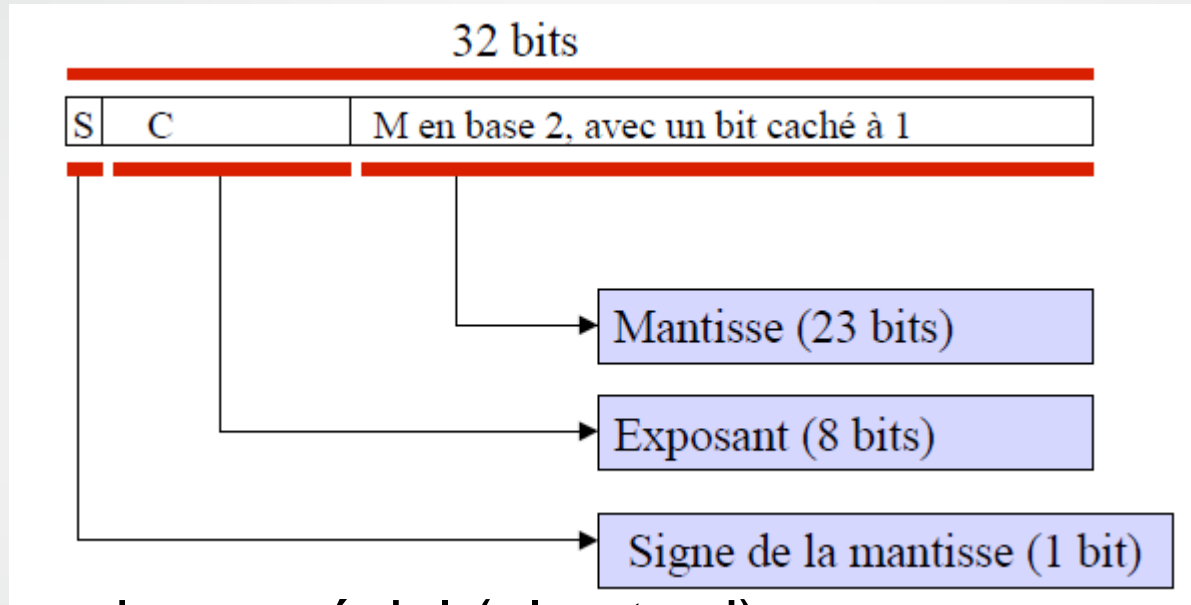
01_{16} à FE_{16} : $2^{-126} - 2^{127}$ - représentation **normalisée**

$7F_{16}$: 127_{10} (on appelle ça le pôle ou la constante d'excentrement)

Mantisse (23 bits)

Représentation traditionnelle

Ne pas oublier le « 1, » implicite devant



Décimal vers simple précision

- Exemple
- $-17.25_{10} = -10001.01_2 = -1.\textcolor{purple}{000101}_2 * 2^4$
- $e = 4 + 127 = 131_{10} = \textcolor{green}{10000011}_2$
- Soit $\textcolor{red}{1} \textcolor{green}{10000011} \textcolor{purple}{000101000000000000000000}$
- Exercices
- Convertir 0.125
- Convertir 0.2



Simple précision vers décimal

- Exemple
- 0 10000010 010010000000000000000000
- signe = 0 → positif
- exposant = 10000010₂ → 130 - 127 = 3
- mantisse = 010010000000000000000000 → 1.01001₂
- résultat = +1.01001₂ × 2³ = 1010.01₂ = 10.25₁₀
- Exercice
- Convertir 01000100100000000000000000000000



Valeurs spéciales IEEE-754

- Cas particuliers



- Zéro
- Valeurs négatives trop petites, $-\infty$
- Valeurs négatives trop proches de zéro
- Valeurs positives trop grandes, $+\infty$
- Valeurs positives trop proches de zéro

Valeur zéro



- **Pas de représentation normalisée**
- On ne peut écrire 0 avec un "1." implicite
- **Représentation dé normalisée**
- Si la mantisse et l'exposant sont tous à zéro, le nombre vaut zéro
- **Deux zéro**
- +0 et -0 ont des représentations différentes

Valeurs spéciales IEEE-754



- **Infini**
- **Utilisé pour les valeurs extrêmes**
 - Mantisse à 0
 - Exposant à FF_{16}
 - Le signe indique $+\infty$ ou $-\infty$

Nombres dénormalisés

- [illegible]



Valeurs spéciales IEEE-754

- Pas des nombres
- NaN (not a number)
- Représente des résultats erronées
- Racine carré d'un négatif
- **Codage**
- Signe quelconque
- Exposant à FF_{16}
- Mantisse $\neq 0$





Table récapitulative

Type	Exposant	Mantisse
Zéros	0	0
Dénormalisés	0	$\neq 0$
Normalisés	01_{16} à FE_{16}	libre
Infinis	FF_{16}	0
NaN	FF_{16}	$\neq 0$

Double précision (64 bits)

- Pareil mais plus grand

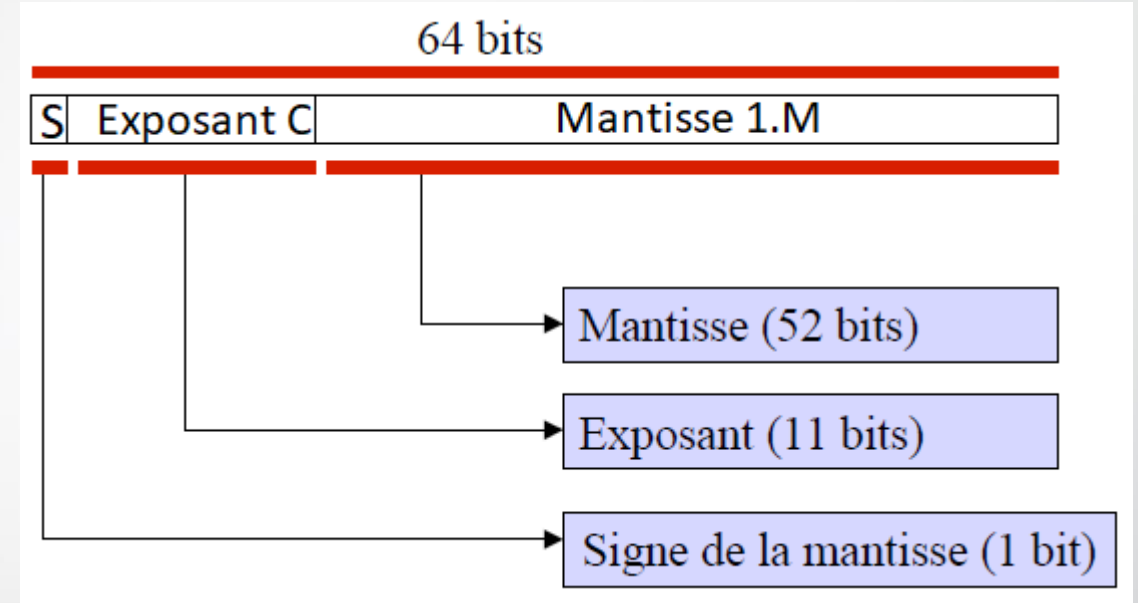
- Signe : 1 bit

- Exposant : 11 bits

- Pôle : 1023_{10} ($3FF_{16}$)

- Mantisse : 52 bits

- Ex: valeurs numériques en JavaScript, `floats` de Python, type `double` Java, C, C++



Perte de précision



- **Problème**
- On ne peut représenter tous les rationnels
- Il y a donc des valeurs non représentables
- Contrairement aux entiers, ces valeurs ne sont pas qu'aux extrémités
- **Travailler avec des flottants**
- Savoir que les flottants ne sont pas précis
- Savoir comment minimiser l'impact de cette imprécision

Représentation infinie

- Rationnels infinis
- Certains rationnels ont une représentation finie en décimal mais infinie en binaire
- Exemple : 0.2_{10}
- Tronquage
- On tronque la mantisse
- Exemple : 0.2 sera représenté par
0.20000000298023223876953125
- **Cette perte d'information se combine**
- $0.37 + 0.2 = ? 0.57$
- JavaScript: $> 0.37 \longrightarrow 0.37 > 0.2 \longrightarrow 0.2$
 $> 0.37 + 0.2 \longrightarrow 0.570000000000000001$




Perte de précision

- **Les gros mangent les petits**
- Additionner (soustraire) deux flottants de magnitudes très différentes n'est pas une bonne idée
- **Exemple**
- $1\text{E}15 + 1\text{E}-15 = ? \quad \longrightarrow \quad 1\text{E}15$
- **Solution**
- Traiter les nombres par groupe de magnitude



Comparaison



- **Comparer c'est se tromper**
- La comparaison par l'égalité de deux flottants n'est pas robuste
- $0.37 + 0.2 - 0.57 == 0$?
- JavaScript: `> 0.37 + 0.2 - 0.57 == 0`  `false`
- **Solution**
- Comparer les distances
- `Math.abs(0.37 + 0.2 - 0.57) < epsilon`
- où epsilon est une constante (petite) définie par la sémantique de programme

Utiliser la bonne représentation

- **Entiers exacts**
 - Utiliser des entiers processeur si les entiers sont petits
 - Utiliser des bibliothèques si les entiers sont grands
- **Décimaux exacts**
 - Utiliser des entiers aussi
 - Exemple : stocker les prix en cents avec un entier
- **Décimaux larges**
 - Utiliser des doubles
 - Prudence vis-à-vis de la perte de précision

