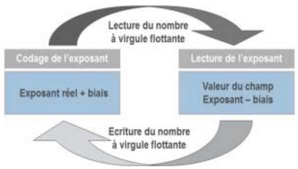
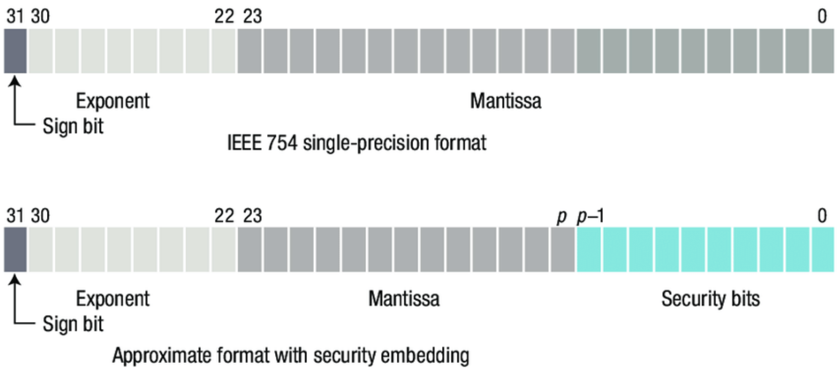
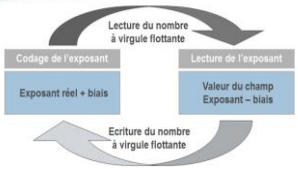


Codage des nombres réels IEEE 754



Introduction:



En informatique, le codage des nombres réels repose sur le principe de la notation scientifique (notation exponentielle) des nombres et le stockage en mémoire de la position de la virgule en plus des chiffres de la mantisse et de l'exposant (format IEEE 754).

<u>Format de normalisation :</u>

$$(1110{,}101101)_2 = (1{,}110101101)_2 \times 2^3$$

Format normalisée : 1, mantisse × 2^{exp}

<u>Exemples :</u>

- $(2019)_{10}=(111\ 1110\ 0011)_2=(1{,}1111100011)_2.2^{10}$
- $(-0{,}6875)_{10}=(-0{,}1011)_2=(1{,}011)_2.2^{-1}$

0,6875 ×2=**1**,375

0,375 ×2=**0**,75

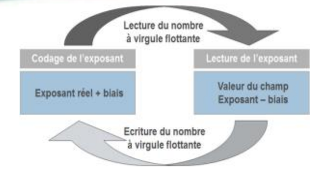
0,75 ×2=**1**,5

0,5 ×2=**1**

valeur	puissances de 2
1	2^0
2	2^1
4	2^2
8	2^3
16	2^4
32	2^5
64	2^6
128	2^7
256	2^8
512	2^9

valeur	puissances de 2
0.5	2^{-1}
0.25	2^{-2}
0.125	2^{-3}
0.0625	2^{-4}
0.03125	2^{-5}
0.015625	2^{-6}
0.0078125	2^{-7}
0.00390625	2^{-8}
0.001953125	2^{-9}

La norme IEEE 754:



Format simple précision : 32 bits

- Bit du signe (1 bit)
- Exposant (8 bits)
- Mantisse (23 bits)

signe	exposant sur 8 bits	mantisse sur 23 bits

- taille totale : 1 + 8 + 23 = **32 bits**
- exposant sur 8 bits => décalage = $2^{8-1} - 1$ = **127**
- Dans de nombreux langages de programmation (C, C++, Java...) le type de donnée associé est nommé **float**

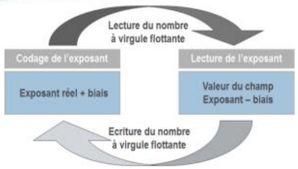
Format double précision : 64 bits

- Bit du signe (1 bit)
- Exposant (11 bits)
- Mantisse (52 bits)

signe	exposant sur 11 bits	mantisse sur 52 bits

- taille totale : 1 + 11 + 52 = **64 bits**
- exposant sur 11 bits => décalage = $2^{11-1} - 1$ = **1023**
- Dans de nombreux langages de programmation (C, C++, Java...) le type de donnée associé est nommé **double**

La norme IEEE 754:



Stockage de l'exposant (position de la virgule)

Excentrement de l'exposant :

Pour pouvoir représenter des exposants **positifs** ou **négatifs**

$$décalage = 2^{n-1} - 1$$

n : nb de bits pour le stockage de l'exposant

Valeur stockée de l'exposant :

Si l'exposant de la représentation normalisée vaut **exp**, la valeur stockée sera :

$$valeur\ stockée\ de\ l'exposant = exp + décalage$$

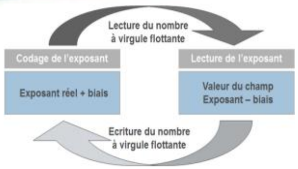
Stockage de la mantisse :

A une exception près, tous les nombres ont une représentation normalisée sous la forme :

Format normalisée : 1, mantisse × 2^{exp}

Par conséquent, il n'est pas nécessaire de stocker le 1 situé à gauche de la virgule.

La norme IEEE 754:

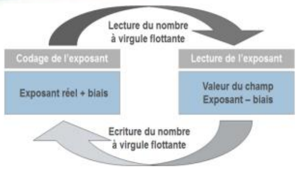


Valeurs particulières :

La norme IEEE 754 réserve les ***exposants 000...000 (uniquement des 0) et 111...111 (uniquement des 1) pour coder des valeurs particulières***

exposant	mantisse	valeur représentée
000...000	000...000	**0** (zéro)
000...000	000...001 à 111...111	nombre **dénormalisé** valeur = $\pm$ **0**,mantisse * $2^{\text{-126 ou -1022}}$
111...111	000...000	± infini
111...111	000...001 à 111...111	**NaN** (Not a Number - *pas un nombre*) exemple : 0 / 0

Java:

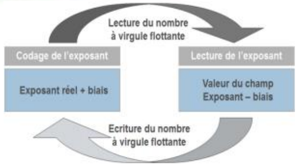



```
System.out.println(1.3-1.2);

                                        ->  0.10000000000000009
```

En fait, il est impossible de représenter exactement 0.1 ou n'importe quelle puissance négative de 10 au moyen d'un **float** ou d'un **double**. La meilleure solution pour résoudre ce problème est d'utiliser la classe **java.math.BigDecimal**.

```
import java.math.BigDecimal;
…
BigDecimal bd1 = new BigDecimal("1.3");
BigDecimal bd2 = new BigDecimal("1.2");
System.out.println(bd1.subtract(bd2));

                                        ->  0.1
```

EXERCICES



- Donnez la représentation flottante des nombres ci-dessous (ici pas de simple ou double précision le format et fixé) :
  - $(6,875)_{10} \rightarrow$ 1 bit de signe, 4 bits d'exposants,6 bits de mantisse
  - $(-24)_{10} \rightarrow$ 1 bit de signe, 5 bits d'exposants,5 bits de mantisse
  - $(0,240234375)_{10} \rightarrow$ 1 bit de signe, 4 bits d'exposants,6 bits de mantisse
  - $(31)_{10} \rightarrow$ 1 bit de signe, 5 bits d'exposants,5 bits de mantisse

- Donnez la représentation décimale des nombres ci-dessous (ici pas de simple ou double précision le format est fixé) :
  - 0 011 1000
  - 1 0110 1100
  - 0 11001 1111010111
  - 1 01001 10110

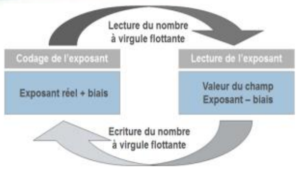
- Donnez la représentation flottante, en ***simple précision***, des nombres suivants :
  - -32,75
  - 0,0625

- Donnez la représentation flottante, en ***double précision***, des nombres suivants :
  - 12,06640625
  - 0,2734375

- Donnez la représentation décimale des nombres codés en ***simple précision*** :
  - $(1011\ 1101\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000)_2$
  - $(0000\ 0000\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000)_2$
  -

- Donnez la représentation décimale des nombres codés en ***double précision*** :
  - $(40\ 3D\ 48\ 00\ 00\ 00\ 00\ 00)_{16}$
  - $(BF\ C0\ 00\ 00\ 00\ 00\ 00\ 00)_2$

PROBLEME (qui rend  JAVA)



I. Quel est le plus petit nombre strictement positif qui, ajouté à 1, donne un résultat différent de 1 ?

- Simple précision
- Double précision

II. Soit le programme suivant en Java, en vous aidant des résultats de la question I :

```java
package com.company;

public class Main {

    public static void main(String[] args) {
        float f1,f2,f3,r;
        f1= 1e25f;    //1e25=2e83
        f2=16f;
        f3=f1+f2;
        r=f3-f1;
        System.out.printf("r=%f",r);
    }
}
```

Type	Taille (en octets)	Valeur minimale	Valeur maximale
float	4	-1.40239846E-45	3.40282347E38
double	8	4.9406564584124654E-324	1.797693134862316E308

- Quelle est la valeur de r qui est affichée à la fin de l'exécution de la méthode main(…) ? Expliquez votre raisonnement.

- Dans le programme, on a $f1=10^{25}$. Supposons maintenant que $f1=10^{n}$ avec n entier positif. Jusqu'à quelle valeur de n un résultat correct apparaîtra-t-il sur r ?