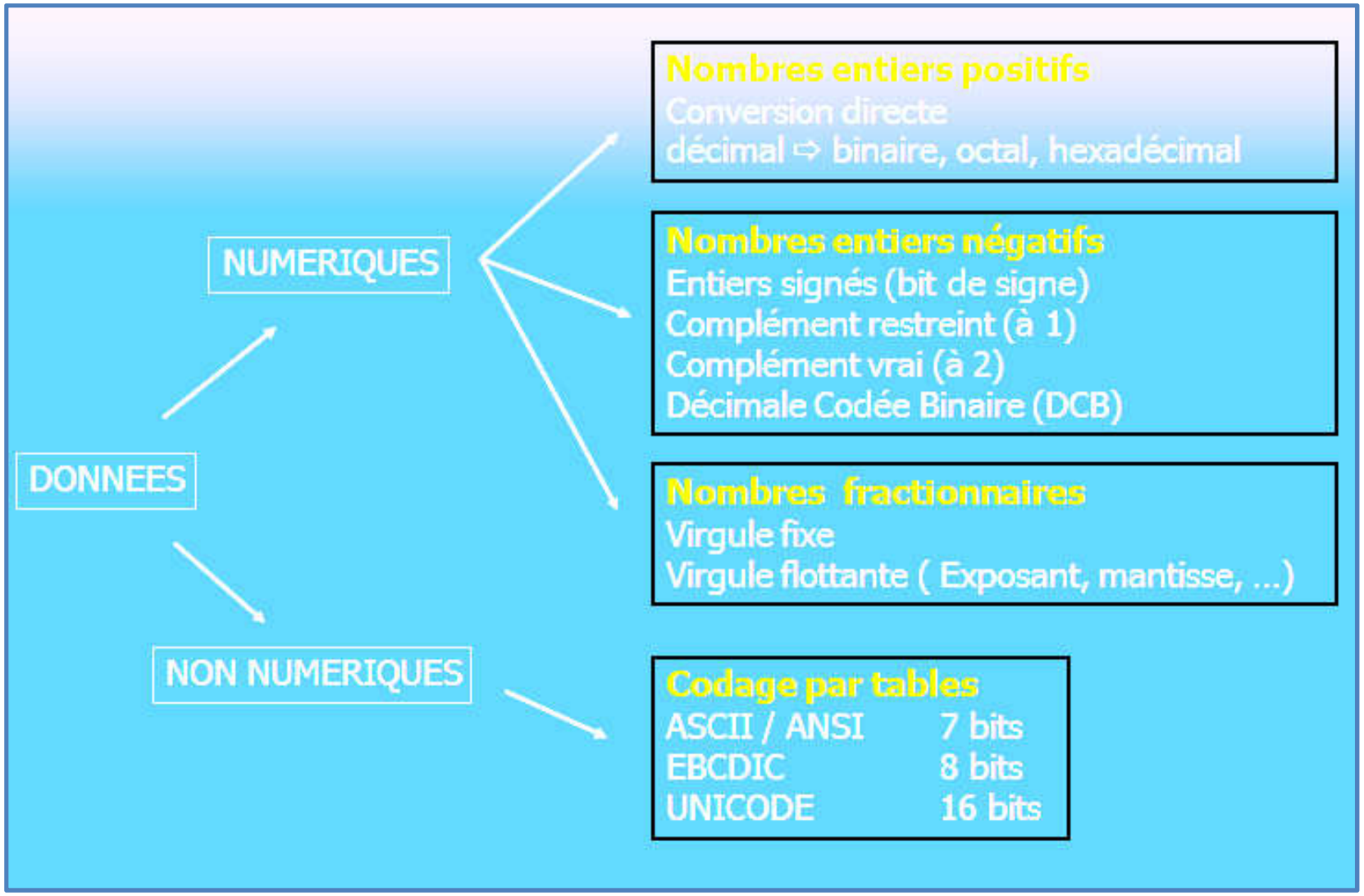
Chapitre 2

Arithmétique à virgule fixe et à virgule flottante

- Partie 1: Format et représentation des nombres
- Partie 2: Virgule fixe et flottante

Format et représentation des nombres



Notions de base: Numération (Binaire, Octale et Hexadécimale)

Système binaire (b=2) utilise deux chiffres : {0, 1}

Système Octale (b=8) utilise huit chiffres :{0, 1, 2, 3, 4, 5, 6, 7}
Il permet de coder 3 bits par un seul symbole.

Système Hexadécimale (b=16) utilise 16 chiffres : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (qui représente le 10), B (11), C (12), D (13), E (14) et F (15)}

NB. Cette base est très utilisée dans le monde de la micro-informatique. Elle permet de coder 4 bits par un seul symbole

Notions de base: Codage des entiers positifs

Codage sur n bits : représentation des nombres de 0 à $2^n - 1$

- sur 1 octet (8 bits) : codage des nombres de 0 à $2^8 - 1 = 255$
- sur 2 octets (16 bits) : codage des nombres de 0 à $2^{16} - 1 = 65535$
- sur 4 octets (32 bits) : codage des nombres de 0 à $2^{32} - 1 = 4\,294\,967\,295$

Notions de base: de la base binaire vers une base b

$35_{10} = (?)_2$

Après division : on obtient : $35_{10} = (100011)_2$

Exemple 2 :

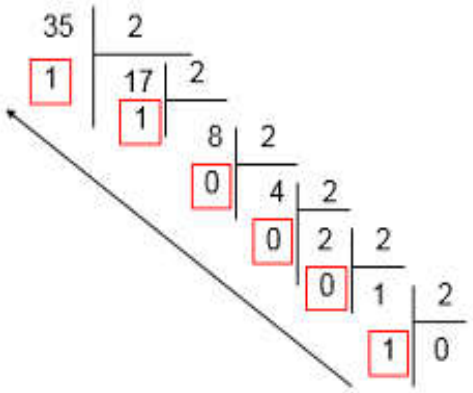
$35_{10} = (?)_8$

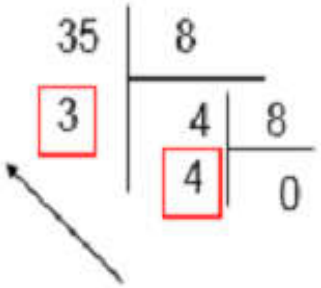
Après division : on obtient : $35_{10} = (43)_8$

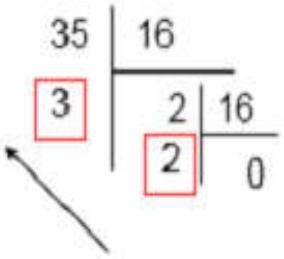
Exemple 3 :

$35_{10} = (?)_{16}$

Après division : on obtient : $35_{10} = (23)_{16}$







Notions de base: de la base binaire vers une base b

$10010_{(2)} = ?_{(8)}$

$10010_{(2)} = 1 \times 2^4 + 0 + 0 + 1 \times 2^1 + 0_{(10)} = 18_{(10)}$

$18_{(10)} = 22_{(8)}$

➔ $10010_{(2)} = 22_{(8)}$

$10010_{(2)} = ?_{(8)}$

$10010_{(2)} = (010)\ (010)_{(2)} = (2)\ (2)_{(8)} = 22_{(8)}$

Symbole Octale	Suite Binaire
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table de correspondance Octale/ Binaire

Notions de base: de la base binaire vers une base b

$10010_{(2)} = ?_{(16)}$

$10010_{(2)} = (0001)\ (0010)_{(2)} = (1)\ (2)_{(16)} = 12_{(16)}$

$1111010_{(2)} = (0111)\ (1010)_{(2)} = 7A_{(16)}$

Symbole Hexadécimale	Suite Binaire	Symbole Hexadécimale	Suite Binaire
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Table de correspondance Hexadécimale / Binaire

Représentation des nombres entiers

Il existe deux types d'entiers :

- les entiers non signés
- et les entiers signés

Problème : Comment indiquer à la machine qu'un nombre est négatif ou positif ?

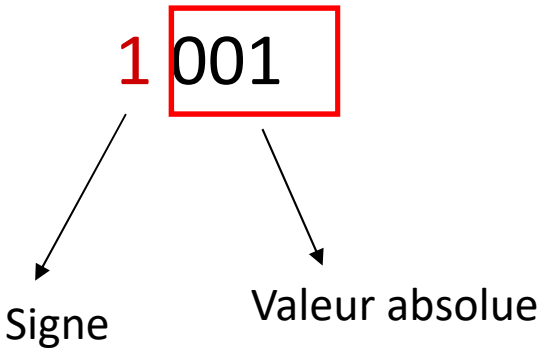
Il existe 3 méthodes pour représenter les nombres négatifs :

- Signe/ valeur absolue
- Complément à 1(complément restreint)
- Complément à 2 (complément à vrai)

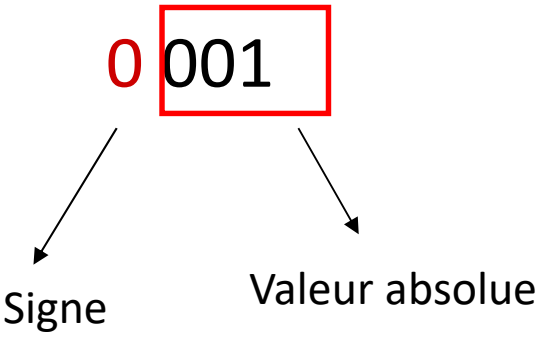
Représentation signe / valeur absolue

- Si on travail sur n bits , alors le bit du poids fort (MSB) est utilisé pour indiquer le signe :
  - 1 : signe négatif
  - 0 : signe positif
- Les autres bits (n -1) désignent la valeur absolue du nombre.

Exemple : Si on travail sur 4 bits.



1001 est la représentation de - 1



0001 est la représentation de + 1

Sur 3 bits on obtient :

signe	VA	valeur
0	00	+ 0
0	01	+ 1
0	10	+ 2
0	11	+ 3
1	00	- 0
1	01	- 1
1	10	- 2
1	11	- 3

• Les valeurs sont comprises entre -3 et +3:

$$-3 \leq N \leq +3$$
$$-(4-1) \leq N \leq +(4-1)$$
$$-(2^2-1) \leq N \leq +(2^2-1)$$
$$-(2^{(3-1)}-1) \leq N \leq +(2^{(3-1)}-1)$$

Si on travail sur **n** bits , l'intervalle des valeurs qu'on peut représenter :

$$-(2^{(n-1)}-1) \leq N \leq +(2^{(n-1)}-1)$$

Avantages et inconvénients de la représentation signe/valeur absolue:

- C'est une représentation assez simple .
- Le zéro possède deux représentations +0 et -0 ce qui conduit à des difficultés au niveau des opérations arithmétiques.
- Pour les opérations arithmétiques il nous faut deux circuits :
  - l'un pour l'addition et le deuxième pour la soustraction .

L'idéal est d'utiliser un seul circuit pour faire les deux opérations, puisque

$$a- b =a + (-b)$$

Complément à 1(complément restreint)

- On appel **complément à un** d'un nombre N un autre nombre N' tel que :

$$N+N'=2^n-1$$

n : est le nombre de bits de la représentation du nombre N .

Exemple :

Soit N=1010 sur 4 bits donc son complément à un de N :

$$N'= (2^4 - 1)-N$$

$N'=(16-1)-(1010)_2= (15) - (1010)_2 = (1111)_2 – (1010)_2 = 0101$

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

Remarque 1 :

- Pour trouver le complément à un d'un nombre, il suffit d'inverser tous les bits de ce nombre

Exemple :

Sur 4 Bits

1 0 1 0

↓ ↓ ↓ ↓

0 1 0 1

Le premier bit est réservé pour le signe et si le nombre est positif alors il garde son format, sinon (il est négatif) alors il est transformé en C1 puis ajouté à 1

L'intervalle des nombres représentables sur n bits en C2 est: $[-(2^{n-1}) , +(2^{n-1}-1)]$

Le nombre de combinaisons possibles sur n bits est: 2^n

Le nombre de valeurs : 2^n

Remarque 2

- le bit du poids fort nous indique le signe (0 : positif , 1 : négatif).
- Le complément à un du complément à un d'un nombre est égale au nombre lui même .

$$CA1(CA1(N))= N$$

Exemple :

Quelle est la valeur décimale représentée par la valeur 101010 en complément à 1 sur 6 bits ?

Le bit poids fort indique qu'il s'agit d'un nombre négatif.

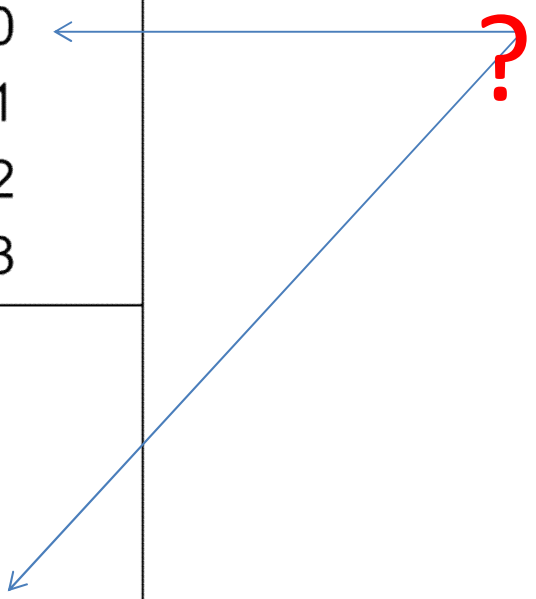
Valeur = - CA1(101010)= - $(010101)_2$= - $(21)_{10}$

•Si on travail sur 3 bits :

•Dans cette représentation , le bit du poids fort nous indique le signe .

•On remarque que dans cette représentation le zéro possède aussi une double représentation (+0 et – 0) .

Valeur en CA1	Valeur en binaire	Valeur décimal
000	000	+ 0
001	001	+ 1
010	010	+ 2
011	011	+ 3
100	- 011	- 3
101	- 010	- 2
110	- 001	- 1
111	- 000	- 0



• Sur 3 bits on remarque que les valeurs sont comprises entre -3 et +3

$$-3 \leq N \leq +3$$
$$-(4-1) \leq N \leq +(4-1)$$
$$-(2^2-1) \leq N \leq +(2^2-1)$$
$$-(2^{(3-1)}-1) \leq N \leq +(2^{(3-1)}-1)$$

Si on travail sur n bits , l'intervalle des valeurs qu'on peut représenter en CA1 :

$$-(2^{(n-1)}-1) \leq N \leq +(2^{(n-1)}-1)$$

Complément à 2 (complément à vrai):

- Si on suppose que **a** est un nombre sur **n** bits alors :

$$a + 2^n = a \text{ modulo } 2^n$$

Exemple : soit a = 1001 sur 4 bits

$2^4 = 10000$

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

Si on prend le résultat sur 4 bits on trouve la même valeur de a = 1001

•Si on prend deux nombres entiers **a** et **b** sur **n** bits , on remarque que la soustraction peut être ramener à une addition : **a – b = a + (-b)**

• Pour cela il suffit de trouver une valeur équivalente à **-b** ?

CA1(b)+1 = CA2(b)

a - b = a+ CA2(b) → transformer la soustraction en une addition .

Exemple

- Trouver le complément à vrai de : 01000101 sur 8 bits ?

CA2(01000101)= CA1(01000101)+ 1

CA1(01000101)= (10111010)

CA2(01000101)=(10111010)+ 1 = (10111011)

Si on travail sur 3 bits :

Valeur en CA2	Valeur en binaire	valeur
000	000	+ 0
001	001	+ 1
010	010	+ 2
011	011	+ 3
100	- 100	- 4
101	- 011	- 3
110	- 010	- 2
111	- 001	- 1

- On remarque que le zéro n'a pas une double représentation.

•Sur 3 bits on remarque que les valeurs sont comprises entre -4 et +3

$$-4 \leq N \leq +3$$
$$-4 \leq N \leq +(4-1)$$
$$-2^2 \leq N \leq +(2^2-1)$$
$$-2^{(3-1)} \leq N \leq (2^{(3-1)}-1)$$

Si on travail sur n bits , l'intervalle des valeurs qu'on peut représenter en CA2 :

$$-(2^{(n-1)}) \leq N \leq +(2^{(n-1)}-1)$$

NB. La représentation en complément à deux (complément à vrai) est la représentation la plus utilisée pour la représentation des nombres négatifs dans la machine.

Opérations arithmétiques en CA2

Effectuer les opérations suivantes sur 5 Bits , en utilisant la représentation en CA2

$$\begin{array}{r} +9 \\ +4 \\ \hline +13 \end{array} \qquad \begin{array}{r} 0\ 1\ 0\ 0\ 1 \\ + \\ 0\ 0\ 1\ 0\ 0 \\ \hline 0\ 1\ 1\ 0\ 1 \end{array}$$

Le résultat est positif

$(01101)_2 = (13)_{10}$

$$\begin{array}{r} +9 \\ -4 \\ \hline +5 \end{array} \qquad \begin{array}{r} 0\ 1\ 0\ 0\ 1 \\ + \\ 1\ 1\ 1\ 0\ 0 \\ \hline 1\ 0\ 0\ 1\ 0\ 1 \end{array}$$

Report

Le résultat est positif

$(00101)_2 = (5)_{10}$

La représentation des nombres réels

Un nombre réel est constitué de deux parties : la partie entière et la partie fractionnelle. 546,0987

Problème : comment indiquer à la machine la position de la virgule ?

Il existe deux méthodes pour représenter les nombre réel :

- ❑ Virgule fixe : la position de la virgule est fixe
- ❑ Virgule flottante : la position de la virgule change (dynamique)

Représentation en virgule fixe

Dans cette représentation la partie entière est représentée sur n bits et la partie fractionnelle sur p bits , en plus un bit est utilisé pour le signe.

Exemple : si n=3 et p=2 on va avoir les valeurs suivantes

Signe	P.Enti	P.Frac	valeur
0	000	00	+ 0,0
0	000	01	+ 0,25
0	000	10	+ 0,5
0	000	11	+ 0,75
0	001	.00	+ 1,0
.	.	.	.
.	.	.	.

NB Dans cette représentation les valeurs sont limitées et nous n'avons pas une grande précision

Utilisée par les premières machines, possède une partie 'entière' et une partie 'décimale' séparées par une virgule. La position de la virgule est fixe d'où le nom.

Exemple:

Dans une représentation sur 8 bits tels que: 1bit représente le signe, 4 bits représentent la partie entière et 3 bits représentent la partie décimale on a:

$(-1011,11)_2 \rightarrow (11011110)_{\text{en machine}}$

Codage en virgule fixe

Ce codage peut s'écrire sous la forme :

[partie entière en binaire , partie décimale en binaire]

- la partie entière d'un nombre se traduisait par des puissances positives de 2.

$$25 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

- La partie décimale va se traduire par des puissances négatives de 2.

$$0.375 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

Exemple : 25,375 $\Rightarrow$ 11001,011

Applications: Décimale -Binaire

(12,6875) =(?)

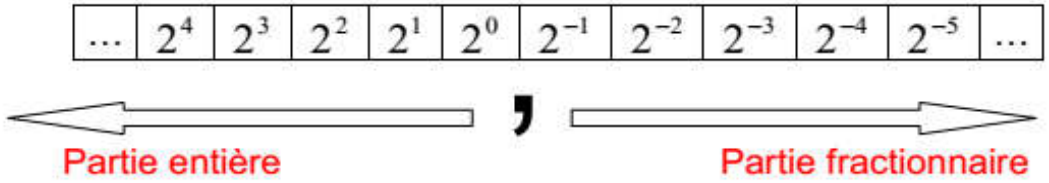
Conversion réel décimal en base B

- Exemple : conversion de 12,6875 en binaire
  - Conversion de 12 : donne $(1100)_2$
  - Conversion de 0,6875
    - 0,6875 x 2 = 1,375 = 1 + 0,375
      0,375 x 2 = 0,75 = 0 + 0,75
      0,75 x 2 = 1,5 = 1 + 0,5
      0,5 x 2 = 1 = 1 + 0

      
    - $(12,6875)_{10} = (1100,1011)_2$

Applications: Binaire- Décimale

La pondération d'un nombre fractionnaire binaire se décompose comme suit :



Exemple : Soit le nombre binaire **1 1 0 1,0 1 1 0 1**

Rang	b_3	b_2	b_1	b_0	b_{-1}	b_{-2}	b_{-3}	b_{-4}	b_{-5}
Poids	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
Nombre	1	1	0	1	0	1	1	0	1

Position de la virgule

Partie entière	Partie fractionnaire
$(1\times2^3)+(1\times2^2)+(1\times2^0)$ $8+4+1$ 13	$(1\times2^{-2})+(1\times2^{-3})+(1\times2^{-5})$ $0.25+0.125+0.03125$ 0.40625
$1101{,}01101_{(2)} = 13{,}40625_{(10)}$	

Applications: Binaire- Décimale

1010,1001=(?) **10,5625**

Représentation en virgule flottante :

Chaque nombre réel peut s'écrire de la façon suivante :

$$N = \pm M * b^{e}$$

M : mantisse ,
b : la base ,
e : l'exposant

Exemple :

$15,6 = 0,156 * 10^{+2}$

$-(110,101)_2 = -(0,110101)_2 * 2^{+3}$

$(0,00101)_2 = (0,101)_2 * 2^{-2}$

Exemple:

$(+3,25)_{10}=(+11,01)_2=[(+0,1101x2^2)=(+1,101x2^1)$
$=(+110,1x2^{-1})=(+1101x2^{-2})]_{\text{en virgule flottante}}$

NB
on dit que la mantisse est normalisée si le premier chiffre après la virgule est différent de 0 et le premier chiffre avant la virgule est égale à 0.

Sous la forme virgule flottante, le nombre décimal positif **1234,56** peut s'écrire :

mantisse → ← exposant

	1234,56	x	10^0
Ou	123,456	x	10^1
Ou	12,3456	x	10^2
...	....	...	....
Ou	0,123456	x	10^4
Ou	123456	x	10^{-2}

De la même façon, le nombre binaire positif 0111 s'écrira :

mantisse → ← exposant

	0111	x	2^0
Ou	011,1	x	2^1
Ou	01,11	x	2^2
Ou	0,111	x	2^3

Signe (+)

Rappel : en représentation signé, le bit le plus à gauche du nombre binaire représente le signe (0) pour le signe + et (1) pour le signe -

Sur n bits :

La mantisse est sous la forme signe/valeur absolue

1 bit pour le signe

et k bits pour la valeur.

L'exposant (positif ou négatif) est représenté sur p bits .

Signe mantisse	Exposant	Mantisse normalisée
1 bit	p bits	k bits

NB. Pour la représentation de l'exposant on utilise :

- Le complément à deux
- Exposant décalé ou biaisé

Représentation de l'exposant en complément à deux

On veut représenter les nombres $(0,015)_8$ et $-(15,01)_8$ en virgule flottante sur une machine ayant le format suivant :

Signe mantisse	Exposant en CA2	Mantisse normalisée
1 bit	4 bits	8 bits

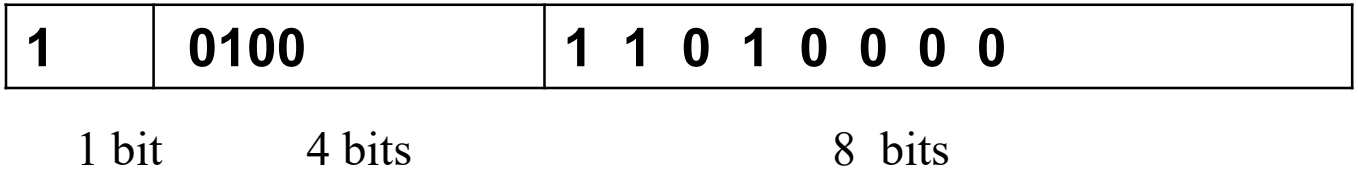
$(0,015)_8=(0,000001101)_2= 0,1101 * 2^{-5}$

Signe mantisse : positif (0)
Mantisse normalisé : 0,1101
Exposant = -5 → utiliser le complément à deux pour représenter le -5
Sur 4 bits CA2(0101)=1011

0	1 0 1 1	1 1 0 1 0 0 0 0
1 bit	4 bits	8 bits

- $(15,01)_8=(001101,000001)_2= 0,1101000001 * 2^4$

Signe mantisse : négatif (1)
Mantisse normalisée : 0,1101000001
Exposant = + 4 =(100)$_2$



L' Exposant décalé (biaisé)

En complément à 2, l'intervalle des valeurs qu'on peut représenter sur p bits :

$$-2^{(p-1)} \leq N \leq 2^{(p-1)} - 1$$

Si on rajoute la valeur $2^{(p-1)}$ à tout les terme de cette inégalité :

$$-2^{(p-1)} + 2^{(p-1)} \leq N + 2^{(p-1)} \leq 2^{(p-1)} - 1 + 2^{(p-1)}$$

$$0 \leq N + 2^{(p-1)} \leq 2^{p} - 1$$

On pose $N' = N + 2^{(p-1)}$ donc : $0 \leq N' \leq 2^{p} - 1$

Dans ce cas on obtient des valeur positives.

La valeur 2^{p-1} s'appelle le biais ou le décalage

Avec l'exposant biaisé on a transformé les exposants négatifs à des exposants positifs en rajoutons à l'exposant la valeur 2^{p-1}.

Exposant Biaisé = Exposant réel + Biais

Exemple

On veut représenter les nombres $(0,015)_8$ et $-(15,01)_8$ en virgule flottante sur une machine ayant le format suivant :

Signe mantisse	Exposant décalé	Mantisse normalisée
1 bit	4 bits	11 bits

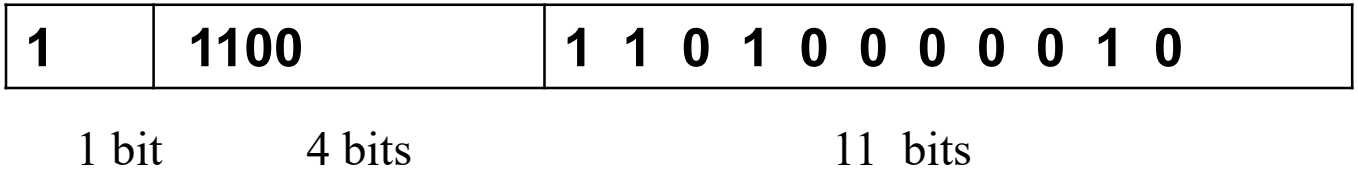
$(0,015)_8=(0,000001101)_2= 0,1101 * 2^{-5}$

Signe mantisse : positif (0)
Mantisse normalisé : 0,1101
Exposant réel = -5
Calculer le biais : $b= 2^{4-1} = 8$
Exposant Biaisé = -5 + 8 = +3 = $(0011)_2$

0	0011	1 1 0 1 0 0 0 0 0 0 0
1 bit	4 bits	11 bits

- $(15,01)_8=(001101,000001)_2= 0,1101000001 * 2^4$

Signe mantisse : négatif (1)
Mantisse normalisée : 0,1101000001
Exposant réel = + 4
Calculer le biais : $b= 2^{4-1} = 8$
Exposant Biaisé = 4 + 8 = +12 = $(1100)_2$



Opérations arithmétiques en virgule flottante

Soit deux nombres réels N1 et N2 tel que

$N1=M1*b^{e1}$ et $N2=M2*b^{e2}$

On veut calculer N1+N2 ?

Deux cas se présentent :

Si e1 = e2 alors $N3= (M1+M2)\ b^{e1}$

Si e1 <> e2 alors élevé au plus grand exposant et faire l'addition des mantisses et par la suite normalisée la mantisse du résultat.

Exemple

Effectuer l'opération suivante : $(0,15)_8 + (1,5)_8 = (?)$:

$(0,15)_8 = (0,001101) = 0,1101 * 2^{-2}$

$(1,5)_8 = (001, 1\ 01) = 0,1101 * 2^{1}$

$(0,15)_8 + (1,5)_8 = 0,1101 * 2^{-2} + 0,1101 * 2^{1}$

$= 0,0001101 * 2^{1} + 0,1101 * 2^{1}$

$= 0, 1110101 * 2^{1}$

0	**0001**	**111010**
1 bit	4 bits	6 bits

Exercice

Donner la représentation des deux nombres **N1= $(-0,014)_8$ et N2=$(0,14)_8$** sur la machine suivante :

Signe mantisse	Exposant biaisé (décalé)	Mantisse normalisée
1 bit	**5 bits**	**10 bits**

- Calculer **N2-N1** ?.

Avant de représenter les deux nombres on doit calculer le biais (décalage)

$B = 2^{5-1} = 2^4 = 16$

$N1 = (-0,014)_8 = (-0,000001100)_2 = (-0,1100)_2 . 2^{-5}$

$ExpB = -5 + 16 = 11 = (01011)_2$

$N2 = (0,14)_8 = (0,001100)_2 = (0,1100)_2 . 2^{-2}$

$ExpB = -2 + 16 = 14 = (01110)_2$

Donc on va avoir la représentation suivante pour N1 et N2:

N1	**1**	**01011**	**1100000000**	**1 010 1111 0000 0000** **$(AF00)_{16}$**
N2	**0**	**01110**	**1100000000**	**0011 10 11 0000 0000** **$(3B00)_{16}$**

$N2 - N1 = 0,14 – (-0,014) = 0,14 + 0,014$

$N2 - N1 = (0,1100)_2 . 2^{-2} + (0,1100)_2 . 2^{-5}$

$= (0,1100)_2 . 2^{-2} + (0,0001100)_2 . 2^{-2}$

$= (0,1101100)_2 . 2^{-2}$

- Si on fait les calculs avec l'exposant biaisé :

$N2 - N1 = (0,1100)_2 . 2^{14} + (0,1100)_2 . 2^{11}$

$= (0,1100)_2 . 2^{14} + (0,0001100)_2 . 2^{14}$

$= (0,1101100)_2 . 2^{14}$

Exposant biaisé = 14

Exposant réel = Exposant biaisé – Biais

Exposant réel = 14 – 16 = -2

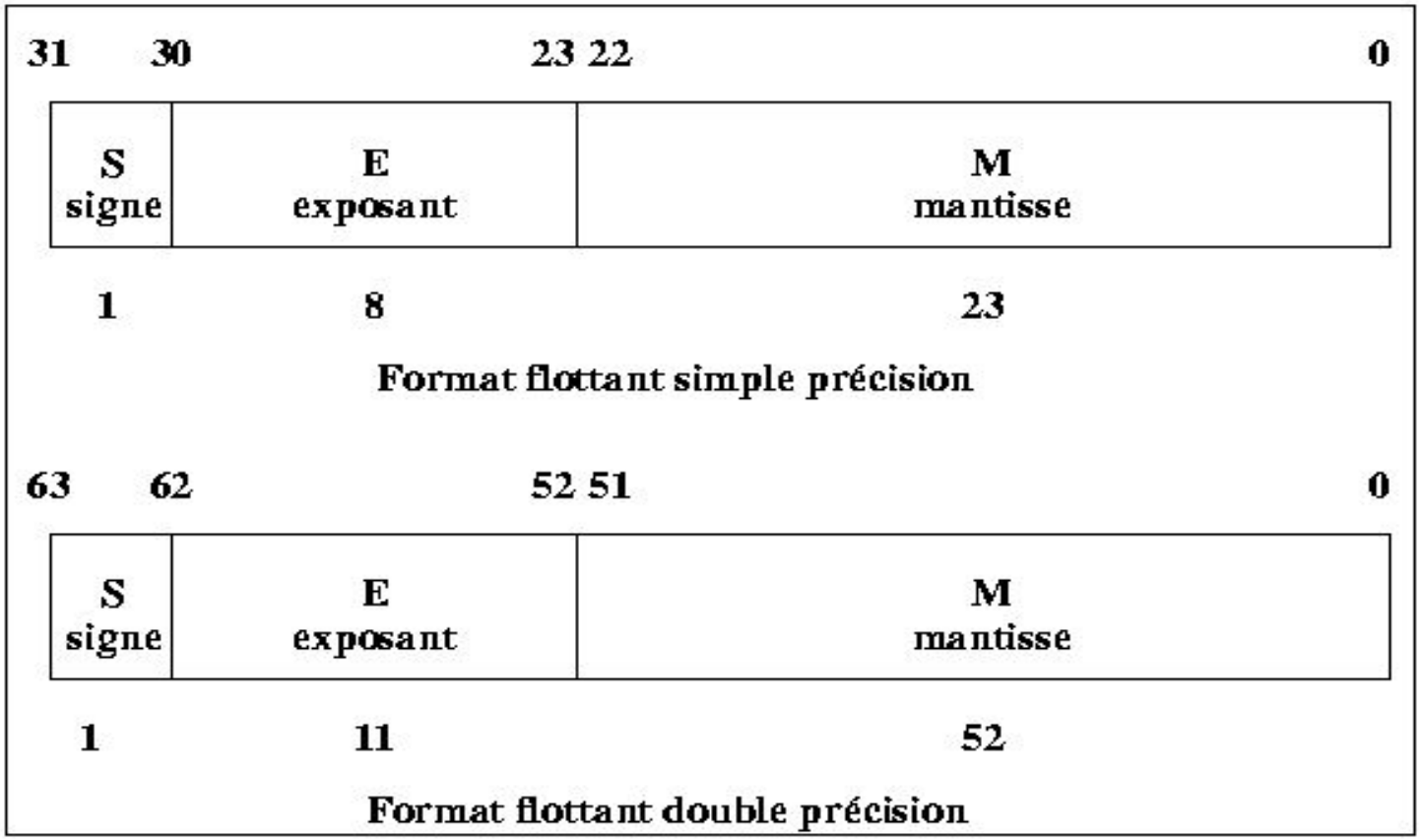
Donc on trouve le même résultat que la première opération.

La norme IEEE 754

- L'IEEE 754 est une norme pour la représentation des nombres à virgule flottante en binaire .
- Définie dans le but d'améliorer la qualité du calcul flottant et la portabilité des applications
- La norme la plus employée actuellement pour le calcul des nombres à virgule flottante dans les CPU .

IEEE Standard for Floating-Point Arithmetic (ANSI/IEEE Std 754-2008)

- ✓ ***Simple précision*** (32 bits : 1 bit de signe, 8 bits d'exposant (−126 à 127), 23 bits de mantisse) ;
- ✓ ***Double précision*** (64 bits : 1 bit de signe, 11 bits d'exposant (−1022 à 1023), 52 bits de mantisse) ;

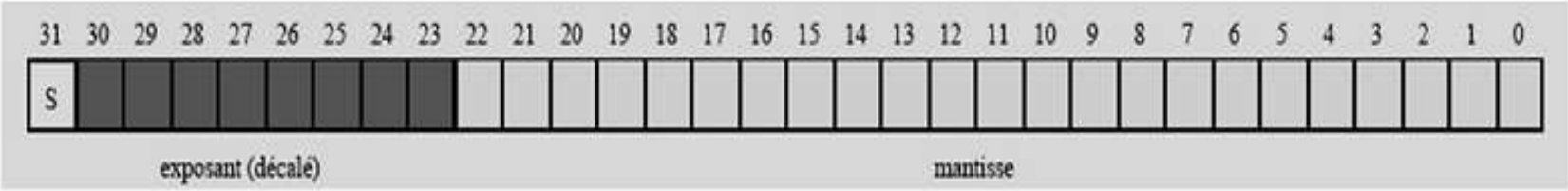


Nombre binaire à virgule flottante

La norme *IEEE* définit la façon de coder un nombre réel:
la forme $1,xxxxx...\times 2^n$ (dans notre exemple $1,0101 \times 2^2$).

Cette norme se propose de coder le nombre sur 32 bits et définit trois composantes :

- le signe est représenté par un seul bit, le bit de poids fort (celui le plus à gauche)
- l'exposant est codé sur les 8 bits consécutifs au signe
- la mantisse (les bits situés après la virgule) sur les 23 bits restants



- Ainsi le codage se fait sous la forme suivante :
  seeeeeeeemmmmmmmmmmmmmmmmmmmmm

Méthode de présentation en virgule flottante IEEE 754:

- Ecrire le nombre sous la forme: $(-1)^S(1+m)\ (2)^e$
  - S=1si N<0 sinon 0
  - *e* et *m* on a trois cas:
    - ✓ abs(N)≥2: on divise par 2 jusqu'a ce que N∈[1 2[, m sera la partie fractionnaire de N et *e*=nbrs de division effectuées
    - ✓ abs(N)<1on multiplier par 2 jusqu'a ce que N∈[1 2[, *m* sera la partie fractionnaire de N et *e*=-nbrs de division effectuées
    - ✓ 1≤abs(N)<2 : *e*=0 et *m* partie fractionnaire de N
  - Présenter *e* et *m* en binaire:

    (*E*=*e*+$(2)^{n-1}-1$, *n*: nombre de bit de l'exposant)

Nombre binaire à virgule flottante

Représentation en simple précision

Exemple 1: Traduire en binaire format flottant simple précision 32 bits le nombre : **- 1039,0**

1) Signe : 1
2) Traduire en binaire $(1039)_{10} = (0000\ 0100\ 0000\ 1111)_2$
3) Constituez la mantisse : **1, mantisse × 2^n**

$$0000\ 0\mathbf{100\ 0000\ 1111} = 1,\mathbf{00\ 0000\ 1111} \times 2^{10}$$

(décalage de dix chiffres vers la droite après la virgule)

Nombre binaire à virgule : Règles

Certaines conditions sont toutefois à respecter pour les exposants :

- l'exposant 00000000 est interdit
- l'exposant 11111111 est interdit. On s'en sert toutefois pour signaler des erreurs, on appelle alors cette configuration du nombre *NaN*, ce qui signifie *Not a Number*
- Il faut rajouter 127 (01111111) à l'exposant pour une conversion de décimal vers un nombre réel binaire. Les exposants peuvent ainsi aller de -254 à 255

Exemple2: $(+7)_{10}$ =(?) IEEE754 simple précision

$(+7)_{10}=(+111)_2=1,11x2^2$

Donc:

SM=0 (signe positif)

M=11000000000000000000000

E=2→Eb=2+127=129=$(10000001)_2$

$(+7)_{10}$=(01000000111000000000000000000000)

Nombre binaire à virgule flottante

Représentation en simple précision

4) Constituez le calage IEEE en simple précision **8 bits** : $2^{8-1} - 1 = \mathbf{127}$

5) Constituez l'exposant : 0000 0**100 0000 1111** = 1,**00 0000 1111** × $\mathbf{2^{10}}$

exposant = 10 + calage = 10 +127= 137

4) Exprimer l'exposant en binaire $(137)_{10} = (1000\ 1001)_2$

5) Etendre la partie fractionnaire à 23 bits

mantisse sur 23 bits = **000 0001 1110 0000 0000 0000**

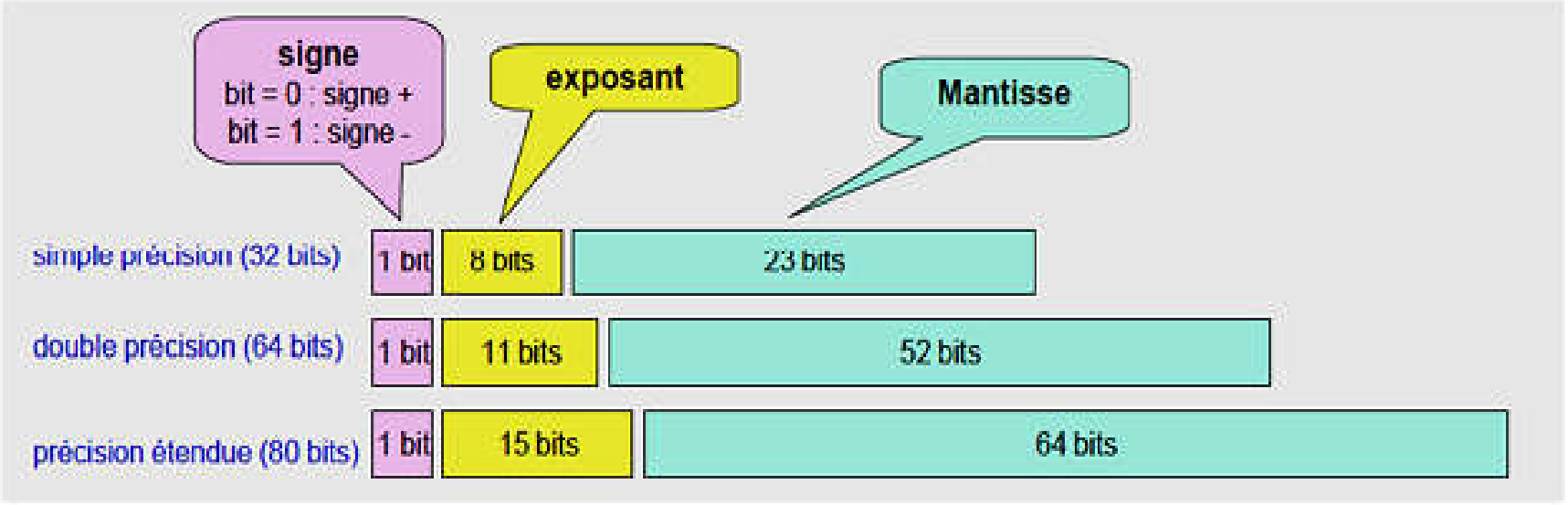
$(-1039,0)_{10} = (1100\ 0100\ 1000\ 0001\ 1110\ 0000\ 0000\ 0000)_2$

En hexadecimal C4 81 E0 00

Nombre binaire à virgule flottante

Représentation en double précision (64 bits)

- On utilise la même méthode que pour le codage en simple précision, sauf que l'exposant est codé sur 11 bits et la mantisse sur 52 bits.



exposant codé = exposant réel + excédent

Nombre binaire à virgule flottante

Limites

Information	Simple précision	Double précision
Bit de signe	1	1
Bit d'exposant	8	11
Bit de mantisse	23	52
Nombre total de bits	32	64
Codage de l'exposant	Excédant 127	Excédant 1023
Variation de l'exposant	-126 à +127	-1022 à +1023
Plus petit nombre normalisé	2^{-126}	2^{-1022}
Plus grand nombre normalisé	$\approx 2^{+128}$	$\approx 2^{+1024}$
Échelle des nombres décimaux	$\approx 10^{-38}$ à 10^{+38}	$\approx 10^{-308}$ à 10^{+308}

Applications

Représenter les nombres suivants en format IEEE 754

- **525.5**
- **-0,625**
- **-6,625**
- **8,625**

Exemples

Nombre binaire à virgule : Exemple

Soit à coder en simple précision la valeur 525,5.

- 525,5 est positif $(-1)^0$ donc le 1er bit sera s = 0.
- Sa représentation en base 2 est la suivante : 1000001101,1
- En normalisant, on trouve : $1,0000011011 \times 2^9$
- On ajoute 127 à l'exposant 9 ce qui donne 136, soit en base 2 : eeeeeeee = 10001000
- La mantisse est composée de la partie décimale de 525,5 en base 2 normalisée, c'est-à-dire 0000011011.
- Comme la mantisse doit occuper 23 bits, il est nécessaire d'ajouter des zéros pour la compléter :

00000110110000000000000

La représentation du nombre 525,5 en binaire avec la norme IEEE est donc :

0 1000 1000 00000110110000000000000

0100 0100 0000 0011 0110 0000 0000 0000

(4403600 en hexadécimal)

Exemples

Exemple 2

Ecrire la valeur -0,625 en simple précision avec le format IEEE754

Le bit s vaut 1 car 0,625 est négatif

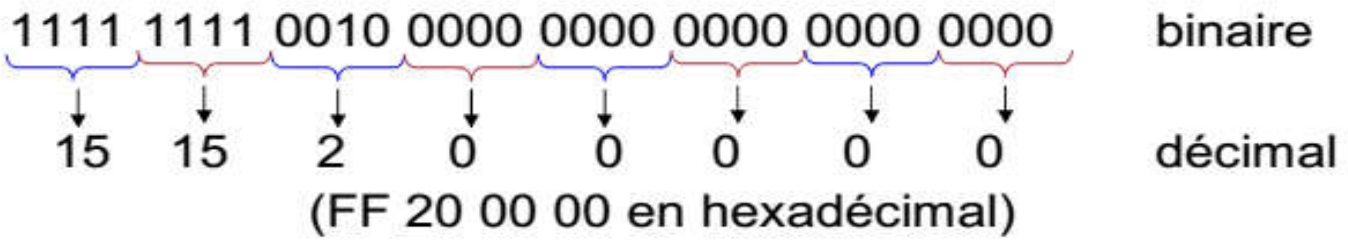
0,625 s'écrit en base 2 de la façon suivante : 0,101

On l'écrit sous la forme normalisée 1.01×2^{-1}

Par conséquent l'exposant E = 127 - 1 ⇨E= 126 (soit 1111110 en binaire)

la mantisse est 01000000000000000000000

La représentation du nombre 0,625 en binaire avec la norme IEEE est : 1 1111 1110 01000000000000000000000



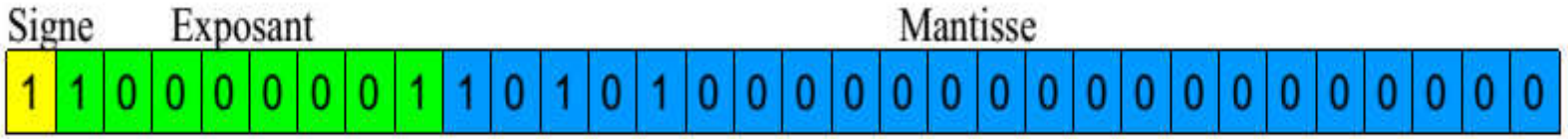
Exemples

Exemple

Traduisons en binaire, en utilisant la norme IEEE 754, le nombre –6,625.

- Codons d'abord la valeur absolue en binaire : $6{,}625_{10} = 110{,}1010_2$
- Nous mettons ce nombre sous la forme : **1,** ***partie fractionnaire***

  $110{,}1010 = 1{,}\mathbf{101010} \cdot 2^2$ (2^2 décale la virgule de 2 chiffres vers la droite)
- La partie fractionnaire étendue sur 23 bits est donc **101 0100 0000 0000 0000 0000.**
- **Exposant** $= 127 + 2 = 129_{10} = 1000\ 0001_2$



En hexadécimal : C0 D4 00 00.

Exemples

Convertir le nombre décimal 8,625 en virgule flottante suivant la norme IEEE 754

- Conversion de 8,625 en binaire : 8,625 => 1000,101 car
  - Partie entière : 8 => 1000
  - Partie décimale : 0,625 => 0,101
- Normalisation : $1000,101 = 1000,101 \times 2^{0} = 0,1000101 \times 2^{4}$

- Normalisation IEEE 754 : $1000,101 = 1,\mathbf{0001010} \times 2^{3}$
  (de la forme 1,xxxx où xxx = pseudo mantisse)
- Décomposition du nombre en ses divers éléments :
  - Bit de signe : **0** (Nombre >0)
  - Exposant sur 8 bits biaisé à 127 => **3** + 127 = **130** => **10000010**
  - Pseudo mantisse sur 23 bits : **000 1010** 00000000 00000000

Signe	Exposant biaisé	Pseudo mantisse
0	**1 0 0 0 0 0 1 0**	**0 0 0 1 0 1 0** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Exemples

Convertir de la virgule flottante IEEE 754 simple précision, vers le décimal:

1 1 01111101 11100000000000000000000

2 1000 1000 1000 1000 1000 0000 0000 0000

3 (3D 2C 00 00)Hex

Solution 1

S=1

E= 01111101= 125, donc e=E-127=125-127=-2

M= 11100000000000000000000=$1x2^{-1} + 1x2^{-2}$+$1x2^{-3}$=0,875

N=$1x(-1)^{S}.(1 + 0{,}875).2^{-2}$=-0,46875

Solution 2

- 0,06640625

Addition en IEEE 754

8

Elle se fait en trois étapes :

1) **Dénormaliser les deux nombres afin d'avoir le même exposant (le plus élevé)**
2) **Additionner les mantisses**
3) **Normaliser le résultat**

Exemple: A=(01000000111000000000000000000000)

+

B=(01000000001000000000000000000000)

1) Dénormalisation

Dénormalisation de A :
SM=0→signe +
Eb=$(10000001)_2$=129
E=Eb-(2^{8-1}-1)=129-127=2
M=11
Donc A=+1,11x2^2

Dénormalisation de B :
SM=0→signe +
Eb=$(10000000)_2$=128
E=Eb-(2^{8-1}-1)=128-127=1
M=001
Donc B=+1,001x2^1=+0,1001x2^2

2) Addition des mantisses

$$\begin{array}{r} 1,11 \\ +\ 0,1001 \\ \hline = 10,0101 \end{array}$$

Donc A+B=10,0101x2^2

3) Normalisation

A+B=10,0101x2^2=1,00101x2^3
E=3→Eb=127+3=130=$(10000010)_2$ et M=00101
Donc: A+B=
$(01000001000101000000000000000000)_{IEEE754}$

Multiplication en IEEE 754

51

Elle se fait en quatre étapes :

1) **Dénormaliser les deux nombres (exposants naturels)**
2) **Additionner les éxposants naturels**
3) **Multiplier les mantisses**
4) **Normaliser le résultat**

52

Exemple : $(01000000111000000000000000000000)_{IEEE754}$

$\times$

$(01000001001100000000000000000000)_{IEEE754}$

Dénormalisation

10000001 → Eb=129 → E=Eb-127=2

10000010 → Eb=130 → E=Eb-127=3

$(01000000111000000000000000000000)_{IEEE754}=1,11*2^{2}$

$(01000001001100000000000000000000)_{IEEE754}=1,0011*2^{3}$

- Addition des exposants : E=2+3=5
- Multiplication des mantisses : 1,11x1,0011=10,000101

Résultat= $10,000101x2^{5}$