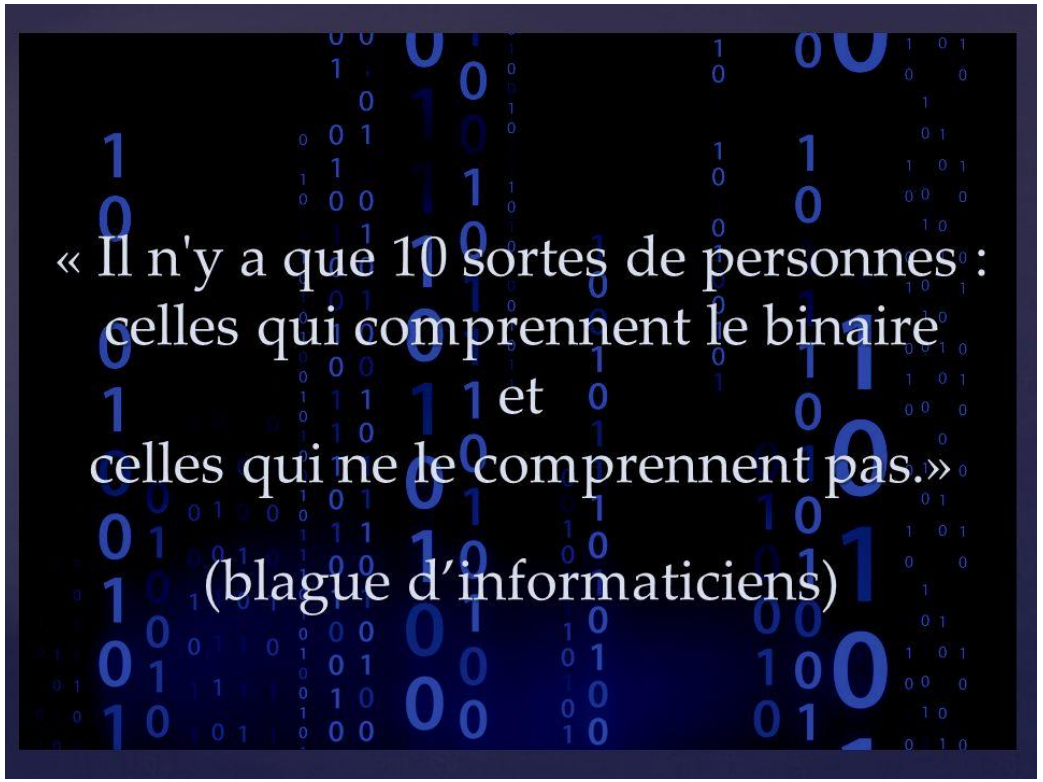
INFORMATIQUE 1

II. REPRESENTATION DES NOMBRES (PARTIE 2)



Eva Ansermin & Romuald Grignon

CY TECH

Rappel

- Tableau des puissances de 2

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	512	256	128	64	32	16	8	4	2	1

- Exemple :

$$1000\ 1010_2 \rightarrow 2 + 8 + 128 = 138$$

Eva ANSERMIN & Romuald GRIGNON

L'hexadécimal

- En **base hexadécimale** (ou base 16):
  - Les chiffres sont : {0,1,2,3,4,5,6,7,8,9,**A,B,C,D,E,F**}
  - On multiplie les chiffres par des puissances de 16 :

Soit N un entier dont les chiffres sont $\{p_n, p_{n-1}, p_{n-2}, \dots, p_0\}$ en base décimale :

$$N = \sum_{i=0}^{n} p_i * 16^i$$

Exemple : $(2C)_{16} \rightarrow 2 * 16^1 + \mathrm{C} * 16^0 = 32 + 12 = 44$

L’hexadécimal

- Connaître les puissances de 16 n’est pas simple ! Mais on peut remarquer que :

$$2^4 = 16 \quad et \quad (1111) = 15$$

->**Un chiffre en hexadécimal correspond donc à 4 bits en binaire.**

Il est souvent plus aisé de passer par le binaire quand on souhaite faire une conversion décimal/hexadécimal.

Exemple: $173 \rightarrow 1010\ 1101 \rightarrow A\ D$

Exemple: $240 \rightarrow 1111\ 0000 \rightarrow F\ 0$

Eva ANSERMIN & Romuald GRIGNON

Valeurs représentables :

- Soit N, un nombre entier naturel codé sur n bits
  - La plus petite valeur possible est 0
  - La plus grande valeur possible est 1111…11

Valeurs représentables :

- Soit N, un nombre entier naturel codé sur n bits
  - La plus petite valeur possible est 0
  - La plus grande valeur possible est 1111…11

On remarque que :

$$\underbrace{11111\ldots111}_{n} + 1 = 1\,\underbrace{0000\ldots00}_{n}$$

$$= 2^n$$

Donc $\mathbf{1111\ldots111 = 2^n - 1}$

Eva ANSERMIN & Romuald GRIGNON

Valeurs représentables :

- Soit N, un nombre entier naturel codé sur n bits, la gamme de valeurs représentables en base 2 est : $[\mathbf{0 : 2^n - 1}]$

- Soit sur :
  - 8 bits : $[0 : 255]$
  - 16 bits : $[0 : 65535]$
  - 32 bits : $[0 : 4\,294\,967\,295]$

Eva ANSERMIN & Romuald GRIGNON

Valeurs représentables : overflow

- Soit un ordinateur dont les nombres sont encodés sur un octet.
- Que se passe-t-il lorsqu'on effectue l'opération suivante ?

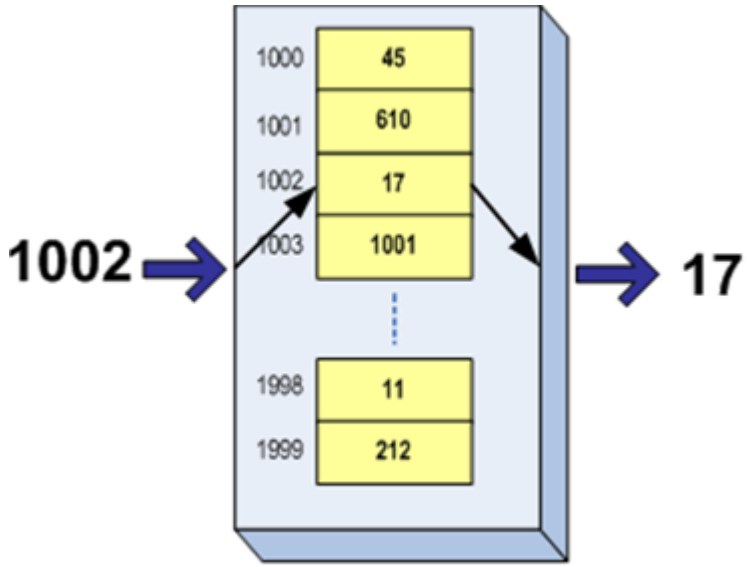
```
   1110  0011
+
   0110  0110
-------------
 1 0100  1001
```

Le résultat est sur 9 bits : le premier chiffre ne peut pas être stocké !

- Lorsque le résultat d'une opération ne peut pas être représenté avec le nombre de chiffres disponible on parle d'**overflow** (dépassement en français).

Conséquence sur la gestion de mémoire

- Une mémoire est divisée en différentes « cases » de même capacité (~ octet).



- **Il faut prendre en compte la taille de nos données pour les stocker dans la mémoire!**

Autre types de données

- La gestion des textes (données alphanumériques) : Des standards (code ASCII, unicode…) permettent d'associer des caractères à des nombres :

Decimal	Char	Decimal	Char	Decimal	Char
32	[SPACE]	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	[DEL]

Extrait de la table ASCII

Représentation du signe (1/2)

- **I. Signe grandeur:** Le bit de signe est le bit de plus fort poids (celui situé le plus à gauche du nombre binaire)
  -> **Il vaut 0 lorsque le nombre est positif et 1 lorsqu'il est négatif.**

- Exemple :

  01001001 représente + 73_{10}
  11001001 représente - 73_{10}

- **Inconvénients** :
  - la soustraction ne marche pas avec cette méthode (A - A ≠ 0)
  - Il y a deux manières de représenter le zéro

Eva ANSERMIN & Romuald GRIGNON

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - On appelle **complément à 1** d'un nombre binaire $\overline{N}$ le nombre N dont les 1 et les 0 sont inversés.
  - On représente –N de la manière suivante en complément à 2 :

$$-N = \overline{N} + 1$$

  - Le bit de poids fort indique le signe :
    - si 0, nombre positif
    - si 1 nombre négatif

Eva ANSERMIN & Romuald GRIGNON

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - On appelle **complément à 1** d'un nombre binaire $\overline{N}$ le nombre N dont les 1 et les 0 sont inversés.

  - On représente –N de la manière suivante en complément à 2 :

$$-N = \overline{N} + 1$$

  - Méthode décimal -> binaire :

1. On transforme la valeur absolue du nombre

$$73_{10} = 0100\ 1001$$

2. On passe au complément :

$$\overline{73} = \overline{0100\ 1001} = 1011\ 0110$$

3. On ajoute +1 :

$$-73 = 1011\ 0110 + 1 = 1011\ 0111$$

Eva ANSERMIN & Romuald GRIGNON

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - On appelle **complément à 1** d'un nombre binaire $\overline{N}$ le nombre N dont les 1 et les 0 sont inversés.

  - On représente –N de la manière suivante en complément à 2 :

$$-N = \overline{N} + 1$$

  - Méthode binaire -> décimal :

1 . On regarde le bit de poids fort : s'il vaut 0 alors le nombre est positif, traduire directement.

2. Si le bit de poids fort vaut 1, le nombre est négatif, il faut faire l'opération inverse

a. Retirer 1

$$1011\ 0111 - 1 = 1011\ 0110$$

b. Faire le complément

$$\overline{1011\ 0110} = 0100\ 1001$$

c. Traduire le nombre obtenu et le mettre en négatif

$$0100\ 1001 \Rightarrow 73_{10} \Leftrightarrow 1011\ 0111 = -73_{10}$$

Eva ANSERMIN & Romuald GRIGNON

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - On appelle **complément à 1** d'un nombre binaire $\overline{N}$ le nombre N dont les 1 et les 0 sont inversés.
  - On représente –N de la manière suivante en complément à 2 :

$$-N = \overline{N} + 1$$

  - Valeurs représentables sur n bits:
    - Le plus grand nombre :

Eva ANSERMIN & Romuald GRIGNON

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - On appelle **complément à 1** d'un nombre binaire $\overline{N}$ le nombre N dont les 1 et les 0 sont inversés.
  - On représente –N de la manière suivante en complément à 2 :

$$-N = \overline{N} + 1$$

  - Valeurs représentables sur n bits:
    - Le plus grand nombre :

$$\underbrace{0111\ldots111}_{\text{n-1 bits}} = 2^{n-1} - 1$$

Eva ANSERMIN & Romuald GRIGNON

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - On appelle **complément à 1** d'un nombre binaire $\overline{N}$ le nombre N dont les 1 et les 0 sont inversés.

  - On représente –N de la manière suivante en complément à 2 :

$$-N = \overline{N} + 1$$

  - Valeurs représentables sur n bits:
    - Le plus grand nombre :

$$\underbrace{0111\ldots111}_{\text{n-1 bits}} = 2^{n-1} - 1$$

    - Le plus petit nombre :

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - On appelle **complément à 1** d'un nombre binaire $\overline{N}$ le nombre N dont les 1 et les 0 sont inversés.

  - On représente –N de la manière suivante en complément à 2 :

$$-N = \overline{N} + 1$$

  - Valeurs représentables sur n bits:
    - Le plus grand nombre :

$$\underbrace{0111\ldots111}_{\text{n-1 bits}} = 2^{n-1} - 1$$

    - Le plus petit nombre :

$$1000\ldots000 \Leftrightarrow -(0111\ldots111 + 1) = -2^{n-1}$$

Eva ANSERMIN & Romuald GRIGNON

Représentation du signe (2/2)

- **II. Le complément à 2:**
  - Intérêts :
    - Une seule manière de représenter le 0 : on couvre un valeur de plus sur le même nombre de bits: **sur n bits ϵ** $\left[-2^{n-1}; 2^{n-1}-1\right]$
    - **Les soustractions / additions fonctionnent !**
  - Soit sur
    - 8 bits : [-128 : 127]
    - 16 bits : [-32768 : 32767]
    - 32 bits : [- 2 147 483 648 : 2 147 483 647]
  - C'est cette **convention** qui est utilisée pour représenter les **entiers négatifs** dans les ordinateurs.

Eva ANSERMIN & Romuald GRIGNON

Représentation nombres réels (1/3)

- Principe
  - Comme pour le symbole “-” il n’y a pas de moyen de représenter explicitement la virgule en binaire.
  - En decimal, les chiffres derrière la virgule sont multipliés par des puissance de 10 negatives :

$$\text{Ex : } 26{,}75 = 2 * 10 + 6 * 10^{0} + 7 * 10^{-1} + 5 * 10^{-2}$$

  - Plus généralement :

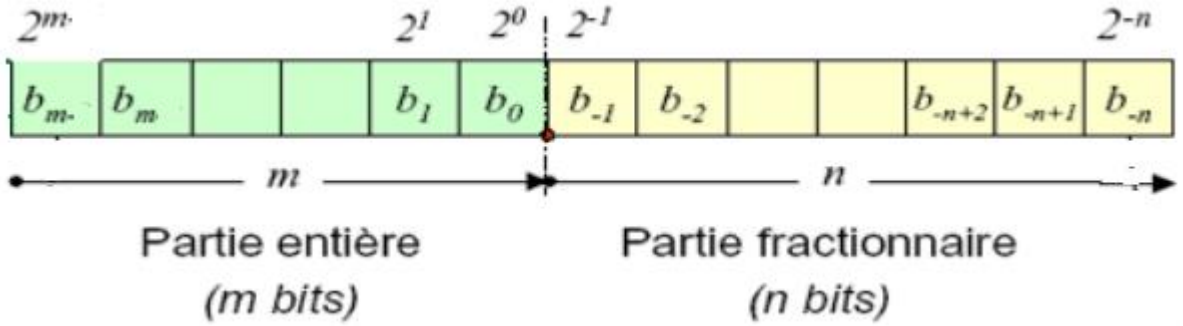
$$N = a_n a_{n\_1} \dots a_0 \, . \, a_{-1} a_{-2} \dots a_{-p} = \sum_{i=-p}^{n} a_i * 10^i$$

  - En binaire, on utilise **des puissances de 2 négatives**.

Représentation nombres réels (2/3)

- **I. La virgule fixe**

  - On défini explicitement le nombre de bits avant (puissance de 2 positive) et après (puissance de 2 negatives) la virgule.



$$N = b_m b_{m-1} \dots b_0 \, b_{-1} b_{-2} \dots a_{-n} = \sum_{i=-n}^{m} a_i \ * 2^i$$

Représentation nombres réels (2/3)

I. La virgule fixe

- Exemple : Sur 8 bits, on définit 5 bits avant et 3 bits après la virgule.

$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 0{,}5$	$2^{-2} = 0{,}25$	$2^{-3} = 0{,}125$

- $11000{,}110_2 \rightarrow 2^4 + 2^3 + 2^{-1} + 2^{-2} = 16 + 8 + 0{,}5 + 0{,}25 = 24{,}75$

- $10{,}625_{10} \rightarrow 8 + 2 + 0{,}5 + 0{,}125 = 2^3 + 2^1 + 2^{-1} + 2^{-3} = 01010101_2$

Eva ANSERMIN & Romuald GRIGNON

Représentation nombres réels (2/3)

I. La virgule fixe :

- Exemple : Sur 8 bits, on définit 5 bits avant et 3 bits après la virgule.

$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 0{,}5$	$2^{-2} = 0{,}25$	$2^{-3} = 0{,}125$

- 32,5 ? Il faudrait plus de bits avant la virgule et moins après.

$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 0{,}5$	$2^{-2} = 0{,}25$

$$32{,}5 \rightarrow 100000{,}10$$

Mais ici on ne peut plus representer 10,625!

Eva ANSERMIN & Romuald GRIGNON

Représentation nombres réels (2/3)

I. La virgule fixe :

- Exemple : Sur 8 bits, on définit 5 bits avant et 3 bits après la virgule.

$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 0{,}5$	$2^{-2} = 0{,}25$	$2^{-3} = 0{,}125$

- 3,5625 ? Il faudrait plus de bits après la virgule et moins avant.

$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 0{,}5$	$2^{-2} = 0{,}25$	$2^{-3} = 0{,}125$	$2^{-4} = 0{,}0625$

$$3{,}5625 \rightarrow 0011{,}1001$$

Mais dans ce cas 16 n'est plus représentable !

Eva ANSERMIN & Romuald GRIGNON

Représentation nombres réels (2/3)

- **I. La virgule fixe :**
  - Avantages:
    - facile à calculer
    - Opération faciles à effectuer
  - Inconvénients : **nécessite de trouver un compromis entre précision et grand nombre !**
  - Ce n'est pas ce qui est utilisé dans les ordinateurs du fait du manque de souplesse de cette représentation.

Représentation nombres réels (3/3)

- **II. La virgule flottante :**
  - **La virgule flottante est la convention utilisée dans les ordinateurs.**
  - La norme d'utilisation des virgules flottantes est la norme IEEE754 avec :
    - Simple precision pour les nombres sur 32 bits
    - Double précision pour les nombres sur 64 bits
  - Plus complexe que la virgule fixe, elle permet de représenter des nombres à la fois précis et grands sans changer de règle de représentation.

Représentation nombres réels (3/3)

➢ **II. La virgule flottante : principe**

En decimal, on peut mettre les nombres sous forme “scientifique”:

$$N = \pm a_n, a_{n-1}a_{n-2} \dots a_0 * 10^p$$

Un seul chiffre avant la virgule

Une puissance >0 ou <0

Exemple : $548{,}96 = 5{,}4896 * 10^2$

Représentation nombres réels (3/3)

➢ **II. La virgule flottante : principe**

En decimal, on peut mettre les nombres sous forme “scientifique”:

$$N = \pm a_n, a_{n-1}a_{n-2} \dots a_0 * 10^p$$

Un seul chiffre avant la virgule

Une puissance >0 ou <0

De la même façon en binaire :

$$N = \pm b_n, b_{n-1}b_{n-2} \dots b_0 * 2^p$$

Exemple : $7{,}125 \rightarrow 111{,}001 = 1{,}11001 * 2^2$

Représentation nombres réels (3/3)

➢ II. La virgule flottante : principe

L'écriture en virgule flottante consiste à traduire séparément les différentes parties d'un nombre binaire écrit sous cette forme :

$$N = \pm b_n, b_{n-1} b_{n-2} \dots b_0 * 2^p$$

$$= (-1)^s * 1, m \ * 2^e$$

Avec

- s : le signe (qui vaut 0 ou 1)
- m : la mantisse
- e : l'exposant

Représentation nombres réels (3/3)

➢ II. La virgule flottante : principe

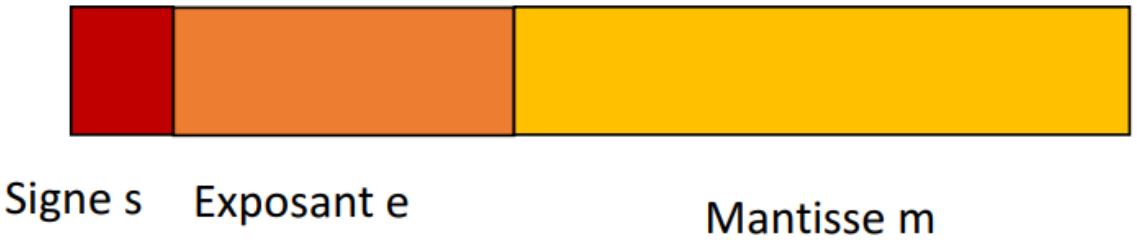
L'écriture en virgule flottante consiste à traduire séparément les différentes parties d'un nombre binaire écrit sous cette forme :

$$N = \pm b_n, b_{n-1} b_{n-2} \dots b_0 * 2^p$$

$$= (-1)^s * 1, m \ * 2^e$$

Avec

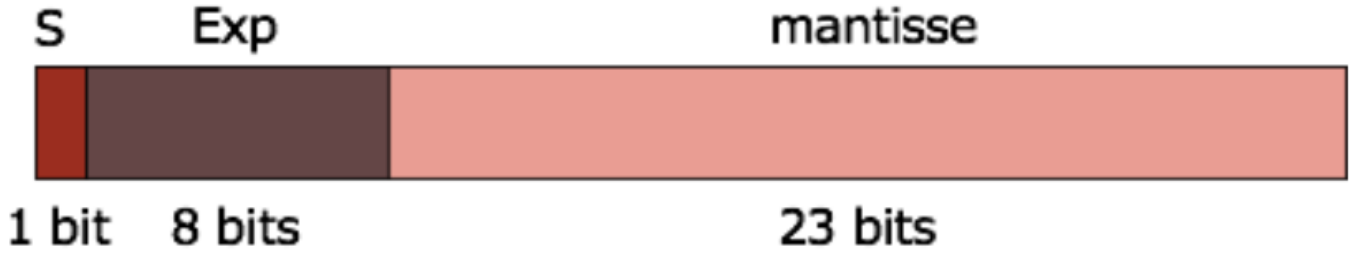
- s : le signe (qui vaut 0 ou 1)
- m : la mantisse
- e : l'exposant



Représentation nombres réels (3/3)

➢ **II. La virgule flottante : simple precision**

En norme simple precision (32 bits) un nombre binaire sera représenté:

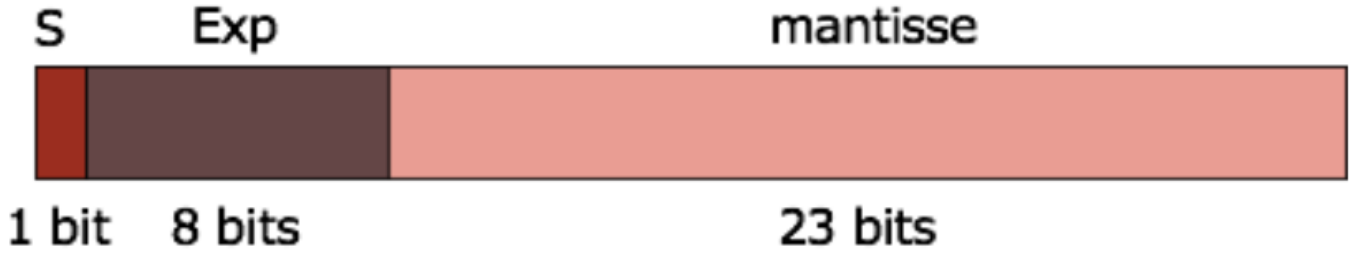


$$(-1)^s * 1,m * 2^e$$

Représentation nombres réels (3/3)

➢ **II. La virgule flottante : simple precision**

En norme simple precision (32 bits) un nombre binaire sera représenté:



$$(-1)^s * 1,m\ * 2^e$$

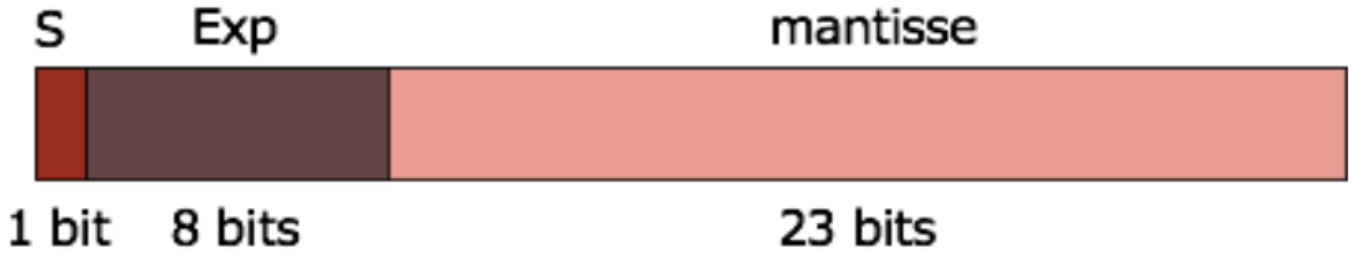
Méthode de calcul de decimal à simple precision :

1. Traduire de decimal en binaire et mettre sous la forme
2. En déduire les différentes parties
    a. le signe ‘s’ sera à 0 si le nombre est positif, à 1 sinon

Représentation nombres réels (3/3)

➢ **II. La virgule flottante : simple precision**

En norme simple precision (32 bits) un nombre binaire sera représenté:



$$(-1)^s * 1,m\ * 2^e$$

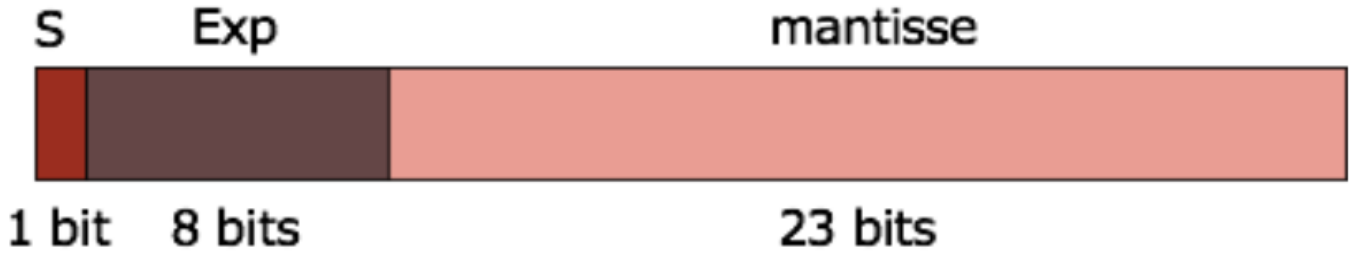
Méthode de calcul de decimal à simple precision :

1. Traduire de decimal en binaire et mettre sous la forme
2. En déduire les différentes parties
    a. le signe ‘s’ sera à 0 si le nombre est positif, à 1 sinon
    b. la mantisse ‘m’ sur 23 bits

Représentation nombres réels (3/3)

➢ **II. La virgule flottante : simple precision**

En norme simple precision (32 bits) un nombre binaire sera représenté:



$$(-1)^s * 1,m\ * 2^e$$

Méthode de calcul de decimal à simple precision :

1. Traduire de decimal en binaire et mettre sous la forme
2. En déduire les différentes parties
    a. le signe ‘s’ sera à 0 si le nombre est positif, à 1 sinon
    b. la mantisse ‘m’ sur 23 bits
    c. $Exp = e + biais = e + 127$. Le biais sert à s’assurer que l’exposant est positif

Représentation nombres réels (3/3)

➢ **II. La virgule flottante : simple precision**

Exemple : $45{,}25$

1. Traduire de décimal en binaire

$$\text{-}45{,}25 = -(32 + 8 + 4 + 1 + 0{,}25) = -(101101.01)$$
$$= -\,1.0110101 * 2^5$$

2. En déduire les différentes parties :
   a. le signe : **1** car nombre négatif
   b. la mantisse = $\mathbf{0110101}$
   c. $Exp = \ 5 + 127 = 132$
   $= 128 + 4 \ = \mathbf{1000\ 0100}$

Donc on a $-45{,}25 \ \rightarrow 1 \ \ \underbrace{1000\ 0100}_{\text{Exposant (8 bits)}} \ \ \underbrace{01101010000000000000000}_{\text{Mantisse (23 bits)}}$

signe

Représentation nombres réels (3/3)

➢ II. **La virgule flottante : remarques**

➢ Valeurs particulières

•Zéro	+/-	0	0
•+/- infini	+/-	111…1111	0
•NaN : not a number	+/-	111…1111	Configuration quelconque de bits

➢ Il est impossible d'obtenir un nombre à la fois très grand et très précis (comme en décimal sur un nombre de chiffres fixe!).

➢ Certains nombres comme les irrationnels ne sont pas représentables : on utilise des arrondis.

Représentation des nombres

- Lorsque l'on va stocker/utiliser un nombre binaire on doit faire plusieurs choix :

1. Quelle **convention de representation**?

$\mathbb{N}$ $(entier\ naturel)$	$\mathbb{Q}$ $(entier\ relatif)$	$\mathbb{Q}$ $(nombre\ rationel)$
Binaire classique	Complément à 2	Virgule flottante

Pour savoir comment interpréter un nombre binaire il faut connaître la convention avec laquelle il est encodé !

2. Quelle **taille** (nombre de bits)?
   - Pas assez de chiffres : nombre non représentable
   - Trop : mémoire gâchée

- Lorsque l'on programme, on indique à l'ordinateur ces différentes propriétés. C'est **le type** .

Conclusion

- L'ordinateur ne pouvant stocker que des 1 et des 0, il est impossible d'utiliser les symboles habituels mathématiques pour représenter des nombres.
- On utilise diverses conventions (virgule flottante, complement à 2) pour gérer la représentation des nombres.
- Il est important de savoir la taille d'une donnée pour gérer son stockage ainsi que sa convention de représentation pour que l'ordinateur sache l'interpreter.
- Nous connaissons la forme de stockage des données d'un ordinateur. Comment se déroule des opérations sur ces données?