ARCHITECTURE DES ORDINATEURS (ADO)

COURS 1
REPRÉSENTATION DES DONNÉES CODAGE

MAROUA MASMOUDI KOTTI

Avec les contributions de : TAISA GUIDINI GONCALVES

Ordinateur

Pour qu'un ordinateur fonctionne correctement il a besoin de :

HARDWARE **Matériel**	**SOFTWARE** **Logiciel**	**USER** **Utilisateur**
Ensemble de composantes matérielles constituant l'ordinateur : entrées, sorties, processeur et mémoires.	Ensemble de programmes permettant de combler en terme de traitement un besoin spécifique. Les instructions qui indiquent au «*Hardware*» ce qu'il doit faire.	Personne qui se sert de l'ordinateur pour faire son travail ou pour s'amuser.

Types d'information

- L'ordinateur manipule deux types d'informations :
  - les **instructions** : écrites en langage machine, elles représentent les opérations que l'ordinateur est capable d'effectuer.
  - les **données** : les opérandes sur lesquels portent les instructions ou produites par celles-ci.

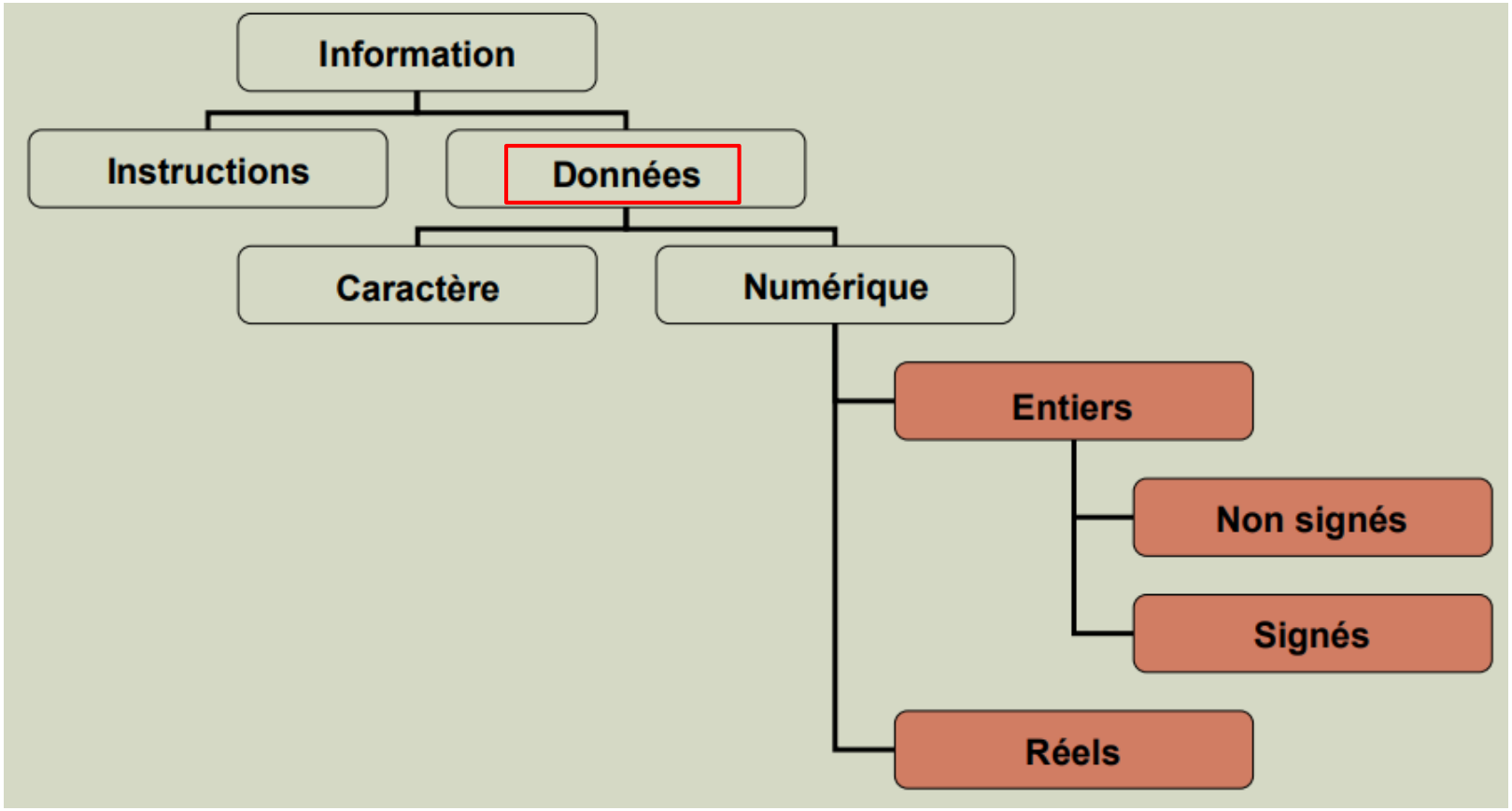
→Exemple : Une addition s'applique à deux opérandes donnant un résultat qui est leur somme.

Instruction et Programme

- Une **instruction** est une opération de base qu'un ordinateur est capable d'exécuter.
  - Exemple : l'addition de deux nombres
- N'importe quel traitement revient à exécuter une **séquence d'instructions** dans un ordre précis.
  - Exemple : le calcul de la moyenne de deux nombres
- Un **programme** est constitué de deux parties :
  - Une partie contenant les données.
  - Une partie code qui représente la séquence des instructions à exécuter.

CODAGE : Représentation des données

- Type d'information

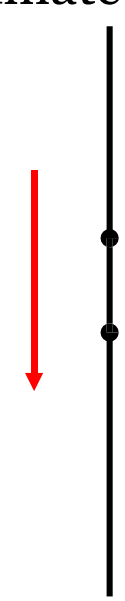


7

-7

CODAGE

- Le codage de l'information concerne les moyens de formaliser l'information afin de pouvoir la manipuler, la stocker ou la transmettre.
- Les ordinateurs ne savent traiter que des 0 et des 1. On parle de **BIT** (binary digit) : la plus petite unité de mesure en informatique.
- L'ordinateur n'est qu'une multitude de circuits électriques.



Courant passe =1

Courant ne passe pas =0

CODAGE

- Les ordinateurs ne savent traiter que des **0** et des **1**.
- Si on souhaite mémoriser un nombre tel que 2, 3 ou plus, il faut trouver un moyen pour le représenter uniquement avec des **0** et des **1**.
- Le **codage** est donc le moyen qui permet de mémoriser dans l'ordinateur toute sorte de nombre, et plus généralement toute sorte d'information.
- Pour plus de facilité, il existe d'autres unités de mesure telle que Octet (Byte), composé de 8 bits (**Exemple :** 0 1 1 0 0 0 1 0), KiloOctet (KiloByte), MegaOctet (MegaByte), GigaOctet (GigaByte), etc.

CODAGE

- Le codage d'une information revient à établir une **correspondance** entre :

la **représentation externe**
de l'information
Exemple : lettre A

et sa **représentation interne**
sous forme de suite de bits
00001010

Codage des données numériques

1. **Entiers positifs ou nuls**
2. **Entiers négatifs**
3. **Nombres fractionnaires/ réels**

Bases de numération

- Les systèmes de numération les plus utilisés sont :
  - **Base 10 ou décimale** avec les chiffres {0,1,2,3,4,5,6,7,8,9}.
  - → Nous savons tous compter en base 10 depuis la maternelle.
  - **Base 2 ou binaire** avec les chiffres {0, 1}.
  - → C'est avec ce système que fonctionnent les ordinateurs.
  - **Base 8 ou octale** avec les chiffres {0,1,2,3,4,5,6,7}.
  - → Utilisé il y a un certain temps en Informatique. Elle permet de coder 3 bits par un seul symbole.
  - **Base 16 ou hexadécimale** avec les chiffres : {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}.
  - → Cette base est très utilisée dans le monde de la micro informatique. Elle permet de coder 4 bits par un seul symbole.

Bases de numération

Décimal	Binaire	Octal	Hexadécimal
0	0000	0	0
1	**0001**	**1**	**1**
2	0010	2	2
3	**0011**	**3**	**3**
4	0100	4	4
5	**0101**	**5**	**5**
6	0110	6	6
7	**0111**	**7**	**7**
8	1000	10	8
9	**1001**	**11**	**9**
10	1010	12	A
11	**1011**	**13**	**B**
12	1100	14	C
13	**1101**	**15**	**D**
14	1110	16	E
15	**1111**	**17**	**F**

Transcodage

- Le transcodage (ou conversion de base) permet de passer de la représentation d'un nombre exprimé dans une base, à la représentation du même nombre exprimé dans une autre base.

- Les conversions possibles :
  - Binaire, Octale ou Hexadécimale **vers** Décimale
  - Décimale **vers** Binaire, Octale et Hexadécimale
  - Binaire **vers** Octale ou Hexadécimale

Transcodage

- ***On a vu que :***
  - $11_{10} = 1011_2 = 13_8 = B_{16}$

→ ***Mais comment fait-on ces différents changements de bases ?***

Conversion de base b vers b_{10}

- Pour passer d'un nombre en base b à un nombre en base 10, on utilise l'écriture polynomiale.

$$(s_3\ s_2\ s_1\ s_0)_b = (s_3 * b^3 + s_2 * b^2 + s_1 * b^1 + s_0 * b^0)_{10}$$

  - les s_i sont les symboles entre 0 et b-1
  - Les b^i sont les poids des symboles s_i.

Exemple:

- De la base binaire vers base 10:

$$(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}$$

- De la base octale vers base 10 :

$$(65)_8 = 6 \times 8^1 + 5 \times 8^0 = (53)_{10}$$

- De la base hexadécimale vers base 10:

$$(B5)_{16} = 11 \times 16^1 + 5 \times 16^0 = 176 + 5 = (181)_{10}$$

Conversion de b_{10} vers une base b

- Pour passer d'un nombre en base 10 à un nombre en base b, on peut utiliser deux méthodes :

1. Méthode par multiplication (Division successive)
2. Méthode par soustraction

Conversion de b_{10} vers une base b

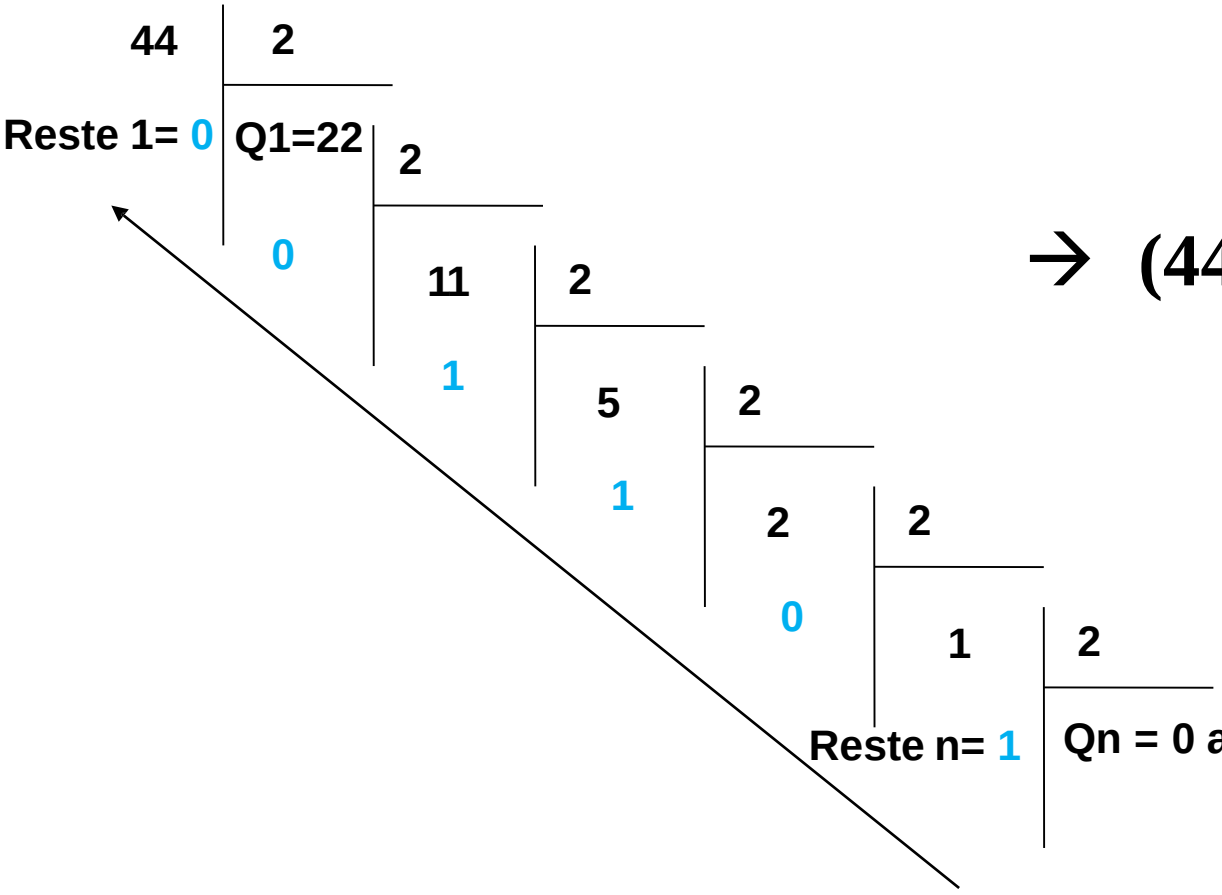
Méthode par multiplication (Divisions successives)

- Soit un nombre entier N représenté dans la base 10.
- On effectue des opérations de **division** euclidienne successive de ce nombre par la base **b** jusqu'à l'obtention d'un **quotient nul**.

  - ➔ Le nombre est obtenu en base **b** en lisant les restes du dernier vers le premier.
  - ➔ Et en écrivant de gauche à droite.

Conversion de b_{10} vers une base b

- Convertir 44 de la base 10 vers la base 2



→ $(44)_{10} = (101100)_2$

Conversion de b_{10} vers une base b

Méthode par soustraction

- Soit n=173 à convertir en base 2.
- Comme $2^7 \leq 173 \leq 2^8$, on a besoin de 8 bits.

Dans N_1 =173, y a t-il 2^7 = 128 ?	**Oui**	**1**
N_2=173-128 = 45, dans 45, y a-t-il 2^6=64 ?	Non	0
Il reste danc N_3=45, dans 45 y a-t-il 32 ?	Oui	1
N_4= 45-32=13, dans 13 y a-t-il 16 ?	Non	0
N_5=13, dans 13 y a-t-il 8 ?	Oui	1
N6=13-8=5, dans 5 y a-t-il 4 ?	Oui	1
N_7=5-4=1, dans 1, y a-t-il 2 ?	Non	0
Il reste N_8=1, la conversion est finie.		1

- Le résultat est donc $(173)_{10} = (10101101)_2$

Conversion de b_2 vers b_8 ou b_{16}

- La conversion d'un nombre binaire dans le système octal ou hexadécimal est très facile et ne demande pas de gros calculs.

b_2 vers b_8

- **En partant de la droite,** on divise le nombre binaire par groupes de **3 bits (triplets),** en ajoutant éventuellement des zéros à gauche pour compléter le **triplet** de gauche.
- On écrit, pour chaque triplet, la valeur en chiffres octaux.
- En mettant ces chiffres l'un à côté de l'autre, dans le même ordre, on obtient le nombre octal recherché.

Exemple :

010 101 001 110 100

2 5 1 6 4

$(101010011110100)_2 = (25164)_8$

b_2 vers b_{16}

- **En partant de la droite,** on divise le nombre binaire par groupes de **4 bits (quadruplets),** en ajoutant éventuellement des zéros à gauche pour compléter le **quadruplet** de gauche.
- On écrit, pour chaque quadruplet, la valeur en chiffres hexadécimaux.
- En mettant ces chiffres l'un à côté de l'autre, dans le même ordre, on obtient le nombre hexadécimal recherché.

Exemple :

0010 1010 0111 0100

2 A 7 4

$(101010011110100)_2 = (2A74)_{16}$

Conversion de b_8 ou b_{16} vers b_2

- La conversion correspond à l'éclatement de chaque chiffre octal (respectivement hexadécimal) en son équivalant binaire sur 3 (respectivement 4) bits.

- Exemple :
  - $(6F5)_{16} = (0110\ 1111\ 0101)_2$
  - $(135)_8 = (001\ 011\ 101)_2$

Passage de la base 8 à la base 16 et inversement

- Pour passer de la base octale à la base hexadécimale ou inversement, il suffit d'utiliser la base décimale ou binaire comme base intermédiaire.

- Exemple :
  - $(135)_8 = (001\ 011\ 101)_2$
  - $(001\ 011\ 101)_2 = (93)_{10}$
  - $(93)_{10} = (5D)_{16}$

Codage des entiers négatifs

- Un **entier** est représenté en mémoire en écrivant ses **bits de poids faible** de la droite vers la gauche, dans des cases de la mémoire en complétant à gauche par des zéros s'il le faut.
- Les **entiers négatifs** peuvent être codés avec l'une des trois méthodes :
  - Signe et valeur absolue
  - Complément à 1
  - Complément à 2

Signe et valeur absolue

- Les nombres sont codés de la façon suivante :
  - Le bit le plus significatif est utilisé pour représenter **le signe** du nombre :
    - si le bit le plus fort = 1, nombre négatif
    - si le bit le plus fort = 0, nombre positif
  - Les autres bits codent la valeur absolue du nombre.

- Exemple :

Sur 8 bits, codage du nombre -24 en valeur absolue (bs).

-24 est codé en binaire signé par : $1\ 0\ 0\ 1\ 1\ 0\ 0\ 0_{(bs)}$

Signe et valeur absolue

- Cette méthode a les inconvénients suivants :
  - "0" a deux représentations 0000000 et 1000000 (7 bits).
  - La somme (binaire) est incorrecte.

→Pour cela on préfère la complémentation.

	4		0 1 0 0
+		+	
	- 3		1 0 1 1
	-----		--------
	- 7		1 1 1 1

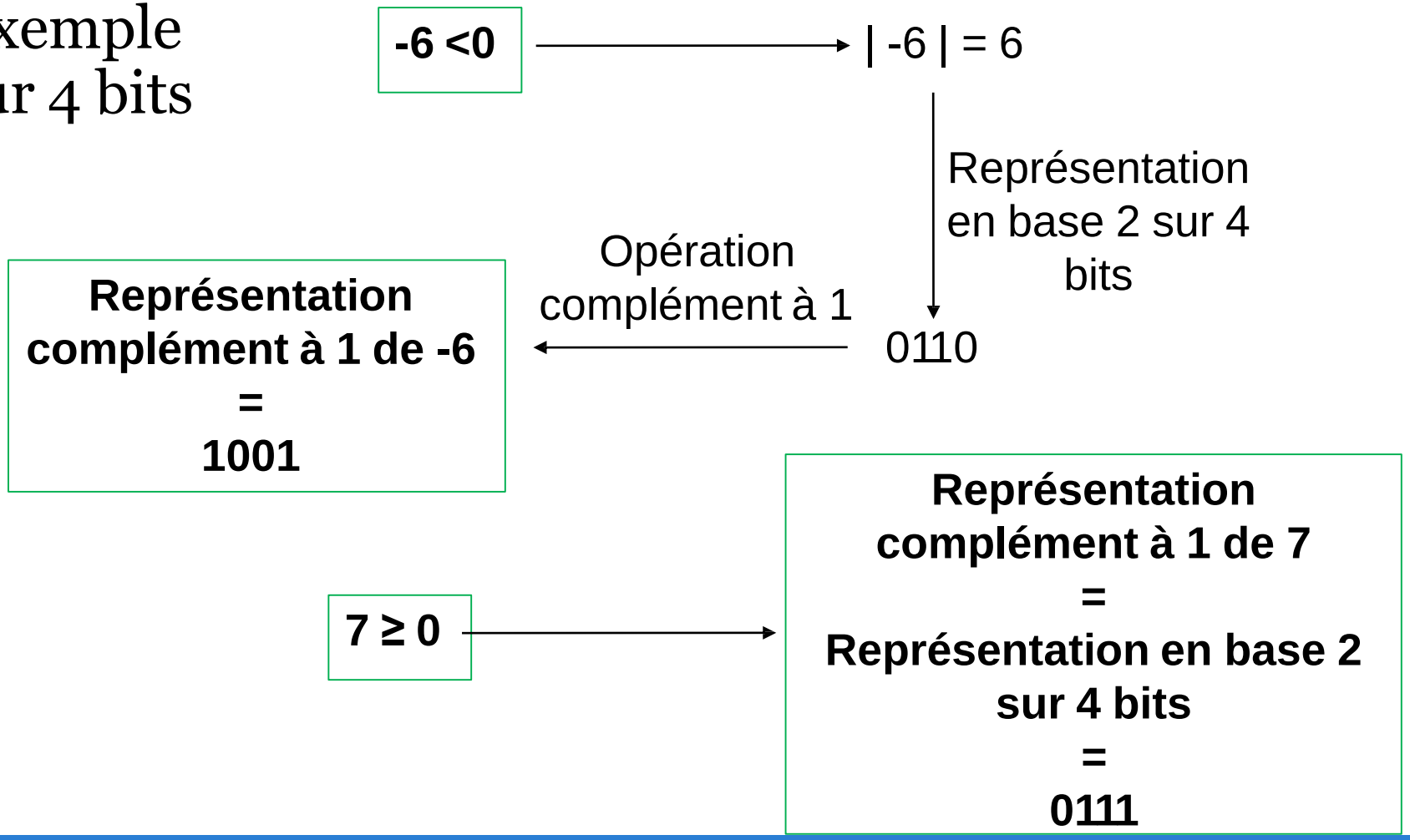
Complément à 1

- Les **nombres positifs** sont codés de la même façon qu'en **binaire signé**.

- Un **nombre négatif** est codé en inversant chaque bit de la représentation de sa valeur absolue en binaire signé.

Complément à 1

- Exemple sur 4 bits



Complément à 1

- ☹ Cette méthode a l'inconvénient suivant :
  - ☹ 0 a deux représentations :
    - ☹ 0000000 et 1111111 (7 bits)

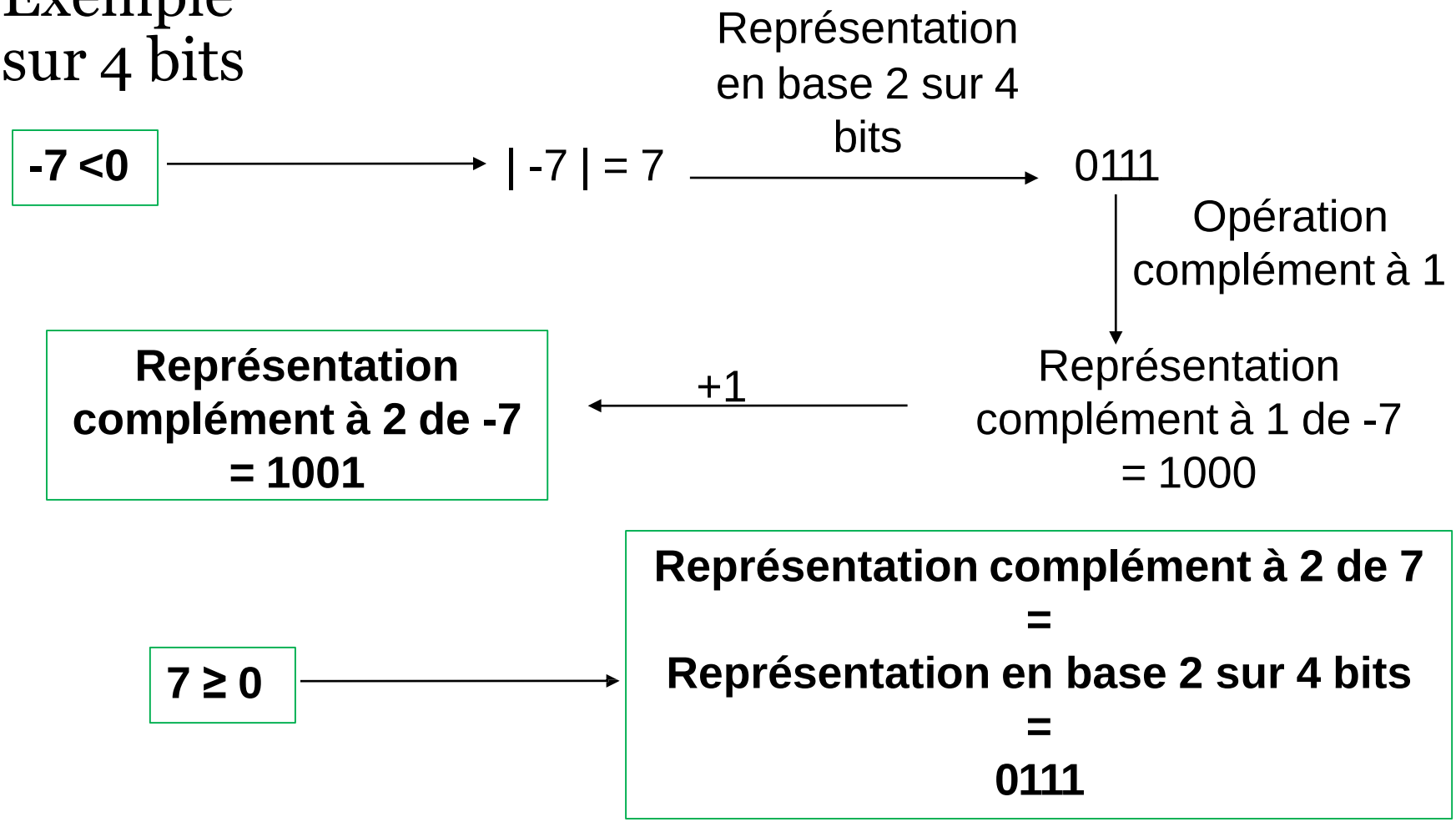
- ☺ Mais elle a l'avantage suivant :
  - ☺ L'addition devient simple.

Complément à 2

- Les **nombres positifs** sont codés de la même manière qu'en **binaire pure**.
- Un **nombre négatif** est codé en ajoutant la valeur 1 à son complément à 1.
- Le bit le plus significatif est utilisé pour représenter le signe du nombre :
  - si le bit le plus fort = 1, nombre négatif
  - si le bit le plus fort = 0, nombre positif

Complément à 2

- Exemple sur 4 bits



Complément à 2

- ☺Cette méthode a les avantages suivants :
  - ☺Une seule représentation pour 0.
  - ☺Les additions deviennent de plus en plus simples : si une retenue est engendrée au niveau du bit le plus à gauche, elle est ignorée.
  - ☺Le bit de poids fort d'un **nombre négatif** est **toujours 1** même en complément à 2.

Codage des nombres réels

- Les nombres réels, dit aussi fractionnaires, sont les nombres qui comportent une partie décimale (après la virgule) non nulle.
- Dans la machine, tout nombre est codé avec un nombre fini de chiffres.
- Il n'est pas possible de représenter tous les rationnels, et à fortiori tous les réels.

Codage des nombres réels
Représentation en Virgule fixe

- Utilisée par les premières machines, chaque nombre est séparé en deux parties :
  - les chiffres avant
  - et les chiffres après la virgule

- Les ordinateurs n'ont pas de virgule.

  →Virgule virtuelle gérée par le programmeur

- Ce dernier doit donc connaître et faire évoluer, au cours des opérations la place de la virgule.

  →De façon à conserver le maximum de chiffres significatifs.

Codage des nombres réels
Représentation en Virgule fixe

- Exemple : de la base décimale vers la base binaire

Représenter 125,375 en virgule fixe

1. Partie entière: Convertir 125

125	:	2	=	62	reste	1
62	:	2	=	31	reste	0
31	:	2	=	15	reste	1
15	:	2	=	7	reste	1
7	:	2	=	3	reste	1
3	:	2	=	1	reste	1
1	:	2	=	0	reste	1

$125_{(10)} = 01\ 11\ 1101_{(2)}$ sur 8 bits

Codage des nombres réels
Représentation en Virgule fixe

2. Partie fractionnaire: 0.375

On multiple la partie fractionnaire par 2. La partie entière obtenue est le poids binaire. La nouvelle partie fractionnaire est de nouveau multipliée par 2, etc.

0,375	X	2	=	0,75
0,75	X	2	=	1,50
0,50	X	2	=	1,00

$0{,}375_{(10)} = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 0{,}011_{(2)}$

Et comme $125_{(10)} = 0111\ 1101_{(2)}$ on a que :

$125{,}375_{(10)} = 0111\ 1101{,}0110\ 0000_{(2)}$

Codage des nombres réels
Représentation en Virgule fixe

- Exemple : de la base binaire vers la base décimale

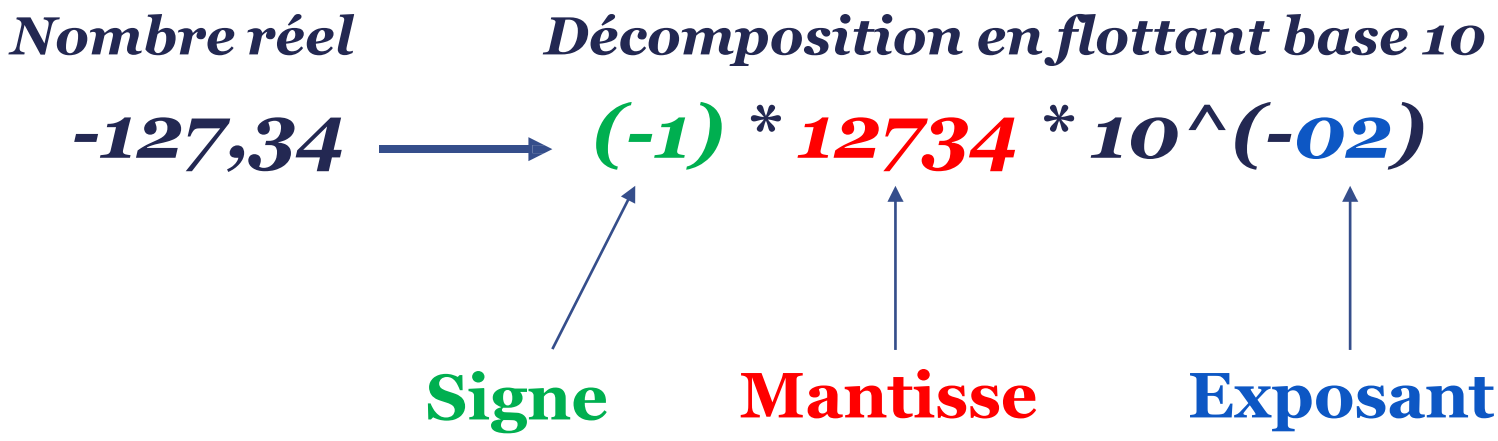
Convertir $0{,}011_{(2)}$ en virgule fixe

1. La partie fractionnaire représente les coefficients des puissances négatives de 2 :

$$0{,}011_{(2)} = (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) = 0{,}25 + 0{,}125$$
$$= 0{,}375_{(10)}$$

Codage des nombres réels
Virgule flottante

La virgule flottante est une méthode d'écriture de nombres fréquemment utilisée dans les ordinateurs. Elle consiste à représenter un nombre réel par :



Codage des nombres réels
Virgule flottante

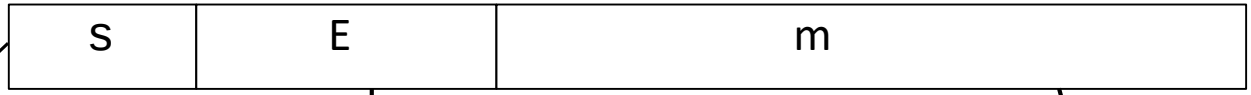
- **Représentation virgule flottante : Norme IEEE 754**
  - Chaque ordinateur avait sa propre représentation des nombres en virgule flottante
    - → besoin d'un standard
    - → la ***Norme IEEE 754*** *(Institute of Electrical and Electronics Engineers)* offre deux representations :
      - short real : sur 32 bits (simple précision)
      - long real : sur 64 bits (double précision)
      - il y a aussi la précision étendue sur 80 bits (pour réduire les erreurs d'arrondis)

Codage des nombres réels
Virgule flottante

- La ***Norme IEEE 754*** consiste à représenter les nombres sous la forme :

$$N = (-1)^s * 1,m * b^e$$

b : base (2, 8, 10, 16) e : exposant = E - biais



signe (1 bit) qui veut 1 ou 0

Exposant biaisé = e + biais
8 bits : simple precision
Biais = 127
11 bits : double precision
Biais = 1023

Mantisse
23 bits : simple précision
52 bits : double précision

Codage des nombres réels
Virgule flottante

- **En simple précision (7 décimales) :**
  - La mantisse est normalisée.
  - L'exposant est un entier sans signe mais biaisé de 127 (e = E – 127), -126 < e < 127.
  - La valeur du nombre codé se lit $\mathbf{(-1)^{signe} * b^{(E-127)} * 1,m}$
  - L'exposant 00000000 signifie que le nombre est dénormalisé.
  - L'exposant maximum est 127 et pas 128 qui est réservé pour des configurations spéciales
    - 0 1111 1111 000…000 = + ∞
    - 1 1111 1111 000…000 = - ∞
    - On note cette configuration NaN (Not a Number) et on l'utilise pour signaler des erreurs (exemple : division par 0)

Codage des nombres réels
Virgule flottante

- **En double précision (16 décimales) :**
  - La mantisse est normalisée.
  - L'exposant est un entier sans signe mais biaisé de 1023 (e = E – 1023).
  - La valeur du nombre codé se lit $\mathbf{(-1)^{signe} * b^{(E-1023)} * 1,m}$

Codage des nombres réels
Virgule flottante

- **Exemple :**
  - Coder le nombre décimal 2.5 en flottant de type short ensuite en flottant de type long selon la norme IEEE 754
  - $(2.5)_{10} = (10.1)_2 = 1.01 * 2^1$

- En simple précision
  - E = 1 + 127 = 128
  - signe 0 ; E = 10000000 et m = 01(000…000)

23-2=21 zéros

- En double précision
  - E = 1 + 1023 = 1024
  - signe 0 ; E = 10000000000 et m= 01(000…000)

52-2=50 zéros

Fonctions arithmétiques

1. **Addition**
2. **Soustraction**
3. **Multiplication**

Addition dans une base b

- **Addition binaire :**

Exemple :

9	1 (retenue au-dessus du 2e 0)
	1 0 0 1
+	+
5	0 1 0 1
-------	------------
14	1 1 1 0

Table d'addition élémentaire

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0
  - avec une retenu de 1

Addition dans une base b

- **Addition octale ou hexadécimale :**

Les opérations d'addition s'effectuent en base quelconque b avec les mêmes méthodes qu'en base 10. Une retenue lorsque l'on atteint ou dépasse la valeur b de la base.

Exemple b=8

	1 1 1	
$(12)_8$	1 0 1 0	
+	+	
$(7)_8$	1 1 1	
-----	--------	
$(21)_8$	1 0 0 0 1	

Exemple b=16

	1 1 1	
$(A)_{16}$	1 0 1 0	
+	+	
$(7)_{16}$	1 1 1	
-----	--------	
$(11)_{16}$	1 0 0 0 1	

Débordement

Exemple : addition binaire sur 4 bits

```
  1111 1
+
  0 1 0 1
- - - - - -
1 0 1 0 0
```

- La valeur obtenue ne peut pas être représentée sur 4 bits, on parle de **débordement** ou de dépassement de capacité (***Overflow***).
- Quand un débordement se produit, un indicateur est mis à 1.
- Dans certains ordinateurs, les calculs continuent, dans d'autres, le traitement s'arrête.
- Dans les deux cas, le problème est signalé.

Retenue et débordement

- On dit qu'il y a une **retenue** si une opération arithmétique génère un report (il suffit alors d'ignorer ce bit de report).
- On dit qu'il y a un **débordement** (*overflow*) ou dépassement de capacité, si le résultat de l'opération sur n bits est faux.
  - Le nombre de bits utilisés est insuffisant pour contenir le résultat.
  - Autrement dit le résultat dépasse l'intervalle des valeurs sur les n bits utilisés.

Soustraction binaire

- **Soustraction binaire :**

Peut se faire comme en decimal.

- Exemple : 1101 – 1011

```
    1101 = 13
-   1011 = 11
    -----
    0010 = 2
```

Table de soustraction élémentaire

- 1 - 0 = 1
- 1 - 1 = 0
- 0 - 0 = 0
- 0 - 1 = 1
  - avec une retenu de 1

- Autre technique : Utiliser les compléments à 2 et ne faire que des additions.

Addition / Soustraction binaire en signé

- Codage en complément à 2.
- Simplifie les additions et soustractions.
- On peut additionner directement des nombres, quels que soient leurs signes, le résultat sera directement correct (**si pas de débordement**) et « bien codé ».

- Soustraction d'un nombre => addition de son complément à 2
  - A – B = A + C2(B)
  - Valable dans tous les cas, quels que soient les signes de A et B
  - Là aussi le résultat est directement valide, si pas de débordement.

Addition/ Soustraction binaire en signé

- Exemple

Addition de deux nombres binaires signés : 21 et -21

```
  1 1 1 1 1
  0 0 1 0 1 0 1        21
+                    +
  1 1 0 1 0 1 1       -21
 ---------------     -----
1 0 0 0 0 0 0 0        0
```

Multiplication binaire

- Comme en decimal.
- N'utilise que du décalage de bits et des additions.

- Exemple : 101 x 110

```
    1 0 1          5
  x 1 1 0        x 6
-------------    -----
    0 0 0         30
+ 1 0 1
  1 0 1
-------------
1 1 1 1 0
```

Observation

Décalage d'un bit vers la gauche, multiplication par 2.

Décalage d'un bit vers la droite, division entière par 2.

Addition en virgule flottante

- Dénormaliser les deux nombres afin d'avoir le même exposant (le plus elevé)
- Additionner les mantisses
- Normaliser le résultat

Nombres Décimaux	Binaire en virgule flottante	Opérandes Alignés	Résultat normalisé
1 + 0,75	$1,0 * 2^{0}$ + $1,1 * 2^{-1}$	$1,0 * 2^{0}$ + $0,11 * 2^{0}$ ——— $1,11 * 2^{0}$	$1,11 * 2^{0}$
1 + 1,5	$1,0 * 2^{0}$ + $1,1 * 2^{0}$	$1,0 * 2^{0}$ + $1,1 * 2^{0}$ ——— $10,1 * 2^{0}$	$1,01 * 2^{1}$

Multiplication en virgule flottante

- Dénormaliser les deux nombres (exposants naturels)
- Additionner les exposants naturels
- Multiplier les mantisses
- Normaliser le résultat

Multiplication en virgule flottante

- Exemple:

$(0\,10000001\,11000000000000000000000)_{IEEE754}$

✖

$(0\,10000010\,00110000000000000000)_{IEEE754}$

- Dénormalisation

10000001 E = 129 e = E - 127 = 2

10000010 E = 130 e = E - 127 = 3

$(0\,10000001\,11000000000000000000000)_{IEEE754} = 1{,}11 * 2^2$

$(0\,10000010\,00110000000000000000)_{IEEE754} = 1{,}0011 * 2^3$

Multiplication en virgule flottante

- Addition des exposants : e = 2 + 3 = 5

- Multiplication des mantisses :

1,11 x 1,0011 = 10,000101 → Résultat = $10{,}000101 * 2^5$

- Normalisation :

$10{,}000101 * 2^5 = 1{,}0000101 * 2^6$

s = 0 ; e = 6 → E = 6 + 127 = 133 = $(10000101)_2$

m = 0000101

Donc le résultat est :

$(0\ 10000101\ 00001010000000000000000)_{IEEE754}$