Codage des nombres

- Coder une information ------→ établir une correspondance entre sa représentation externe et sa représentation interne dans la machine, qui est une suite de bits.
- La représentation (codification) des nombres est nécessaire afin de les stocker et de les manipuler par un ordinateur.
- Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans la machine doit s'effectuer sur un nombre fixe de bits.

Codage des entiers naturels

- Les entiers naturels (positifs ou nuls) sont codés sur un nombre fixe d'octets (1 octet = 8 bits).
- On rencontre habituellement des codages sur 1 , 2, 4 octets, et plus rarement sur 8 octets (64 bits)
- Un codage sur ***n bits permet de représenter tous les entiers naturels compris entre:***

$$0 \text{ et } 2^n-1$$

- Exemple : avec un octet (8 bits), on peut représenter (coder) les nombres appartenant à l'intervalle :

$$[0, 2^8-1] = [0,255]$$

Codage des entiers négatifs

- Les signes **+ et –** ne sont pas reconnus par un ordinateur lequel ne connait que deux états : 0 et 1.

  On les représente donc par un bit qui occupera la case de gauche du nombre considéré.

  Ce bit s'appelle le bit de signe. Donc, **par convention**, on représente le signe « **+** » par **0** et signe « **–** » par **1**.

  Les nombre négatifs sont représentés en machine par une des trois méthodes :

  Signe et Valeur absolue, en complément à 1 ou en complément à 2.

Codage des entiers relatifs (nombres signés)

- La représentation des entiers signés pose des problèmes surtout au niveau de la représentation du signe. Il existe plusieurs manières pour coder les nombres signés :
  - ❑ Codage en signe + valeur absolue
  - ❑ Codage en complément restreint (C à 1)
  - ❑ Complément vrai (C à 2)

Convention : Quelques soit le codage utilisé, le bit de poids fort est réservé pour la représentation du signe : Un nombre négatif a un bit de signe à 1 et un nombre positif a un bit de signe à 0.

Codage en signe + valeur absolue

- Avec n bits, le **n**ième bit est réservé pour le signe et les **n-1** bits restants sont utilisés pour la représentation de la valeur absolue du nombre à coder. Un codage sur n bits permet de coder tous les entiers appartenant à l'intervalle :

$$[-(2^{n-1}-1), +(2^{n-1}-1)]$$

1111	1110	1101	1100	1011	1010	1001	1000
- 7	- 6	- 5	- 4	- 3	- 2	- 1	- 0

0000	0001	0010	0011	0100	0101	0110	0111
+ 0	+ 1	+ 2	+ 3	+ 4	+ 5	+ 6	+ 7

Codage en signe + valeur absolue

	Sur n = 4 bits	Sur n = 8 bits
Intervalle des valeurs couvertes	[- 7, + 7]	[- 127 , + 127]
Statut de la valeur zéro	La valeur zéro est 0000 = 1000	La valeur zéro est 00000000 = 1000000
Exemples	$+(3)_{10} = (\mathbf{0}\ 011)_{SVA}$	$+(3)_{10} = (\mathbf{0}\ 0000011)_{SVA}$
	$-(3)_{10} = (\mathbf{1}\ 011)_{SVA}$	$-(3)_{10} = (\mathbf{1}\ 0000011)_{SVA}$
	$+(15)_{10}$ et $-(15)_{10}$ impossible à représenter	$+(15)_{10} = (\mathbf{0}\ 0001111)_{SVA}$
		$-(15)_{10} = (\mathbf{1}\ \underbrace{0001111}\ )_{SVA}$

Codage en signe + valeur absolue

- Avantages :

Facile à interpréter

- Inconvénients:

2 représentations pour le 0 (+0 et -0) et Problème d'addition de deux nombres de signes opposés

Sur n = 16 bits
[- 32767 , + 32767]
La valeur zéro est = 0000 0000 0000 0000 = 1000 0000 0000 0000
$+(3)_{10}$ = (**0** 000 0000 0000 0011)$_{SVA}$
$-(3)_{10}$ = (**1** 000 0000 0000 0011)$_{SVA}$
$+(15)_{10}$ = (**0** 000 0000 0000 1111)$_{SVA}$
$-(15)_{10}$ = (**1** 000 0000 0000 1111)$_{SVA}$

Codage en signe + valeur absolue

- **Question :**
- Peut-on représenter le nombre -8 sur 04 bits?

- **Réponse :**
- Il est impossible de représenter le chiffre -8 sur 4 bits car sa valeur absolue $|-8_{(10)}|$ qui est égale à $1000_{(2)}$ prends déjà 04 bits et donc on aura besoin au minimum de 5 bits pour pouvoir représenter son bit de signe.

Codage en complément restreint (CR) ou complément à 1 (C à 1)

On obtient le complément à 1 d'un nombre binaire en inversant (le **1 devient 0**, et le **0 devient 1**) chacun de ses bits.

- Les nombres positifs sont codés comme dans le codage en signe plus valeurs absolue (SVA).
- Les nombres négatifs sont déduits des nombres positifs par complémentation bit à bit, c'est-à-dire :
- ***(-N) = CR(N) (en supposant que N est un nombre positif).***
- Un codage sur ***n bits permet de coder tout entier relatif appartenant à l'intervalle*** $[-(2^{n-1}-1), +(2^{n-1}-1)]$

Codage en complément restreint (CR) ou complément à 1 (C à 1)

- **Exemple :**

coder +15 et -15 sur 8 bits en utilisant le codage en CR :

$$(+15)_{10} = (00001111)_2$$

$$(-15)_{10} = CR(00001111) = (11110000)_{CR}$$

Inconvénient :

Deux représentations pour le zéro

Codage en complément vrai (CV) ou complément à 2 (C à 2)

Pour obtenir le complément vrai d'un nombre entier, il suffit d'ajouter un 1 à son complément restreint ():

- ***CV(N) = CR(N)+1 où N est un entier quelconque.***

En codage en complément à 2 :

- Un nombre positif est représenté de la même manière qu'en codage en signe plus valeurs absolue.
- Un nombre négatif est représenté par le complément vrai de son opposé (qui est bien évidemment positif)
- Un codage sur *n bits permet de coder tout entier relatif appartenant à l'intervalle* $[-2^{n-1}, +(2^{n-1}-1)]$

Codage en complément vrai (CV) ou complément à 2 (C à 2)

- **Exemple :**

  Coder +15 et -15 sur 8 bits en utilisant le codage en CV

$$(+15)_{10} = (00001111)$$

$$(-15)_{10} = CR(00001111) + 1 = (11110001)_{CV}$$

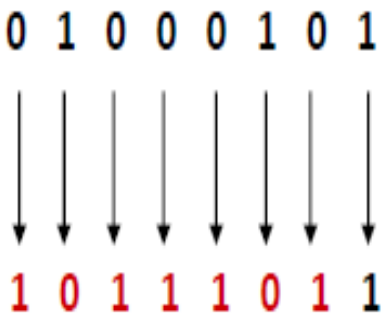
- **Avantage :** Un seul codage pour 0 et pas de problème pour effectuer l'opération d'addition
- **Inconvénient :** Difficile à interpréter.

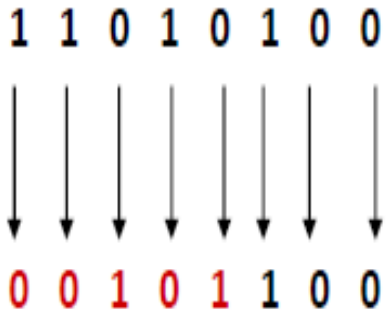
Codage en complément vrai (CV) ou complément à 2 (C à 2)

- **Remarque:**

Autrement pour trouver le complément à 2 d'un nombre : il faut parcourir les bits de ce nombre à partir du poids faible et garder tous les bits avant le **premier 1** et inverser les autres bits qui viennent après.

Exemple:





Soustraction par la méthode du complément

A/ Complément restreint

- Pour une machine travaillant en complément restreint, la soustraction sera obtenue par **l'addition** du complément restreint du nombre à soustraire avec le nombre dont il doit être soustrait et report de la retenue c.-à-d. (ajout de la retenue).

  S'il n'ya pas de retenue, cela signifie que le nombre est négatif. Il se présente sous la forme complémentée (complément restreint). Il suffira de retrouver le CR de ce résultat pour obtenir la valeur recherchée.

Soustraction par la méthode du complément

- **Exemple 1** :

Effectuer l'opération suivante en utilisant la technique CR sur 8 bits : $(63)_{10}$- $(28)_{10}$.

- $(63)_{10} = (00111111)_2$
- $(28)_{10} = (00011100)_2$.
- $CR(28) = CR(00011100) = (11100011)_{CR}$.

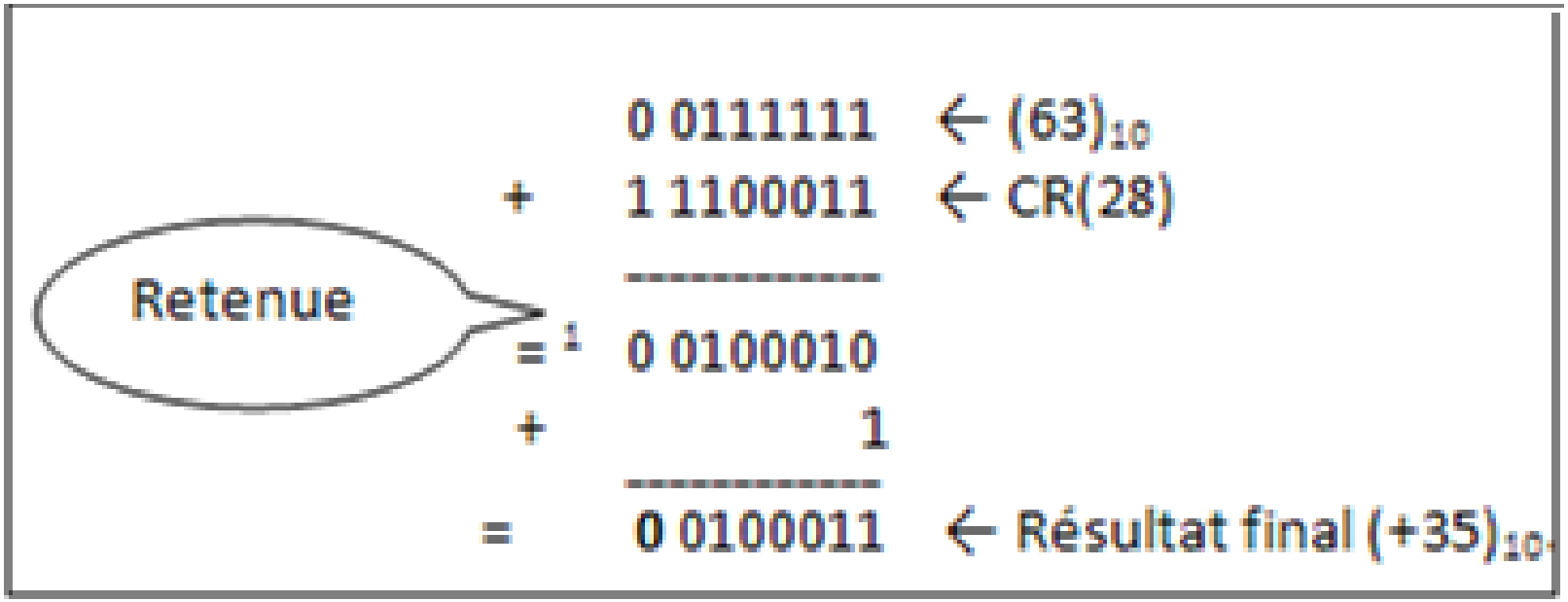
Ensuite faire l'addition de: $(00111111)_2 + (11100011)_{CR}$

Dans cet exemple *est*-ce *qu'il y a* une retenue,

Alors:

Soustraction par la méthode du complément

- **Exemple 1** :



Soustraction par la méthode du complément

- **Exemple 2** :

Effectuer l'opération suivante en utilisant la technique CR sur 8 bits :

- $(28)_{10} - (63)_{10}$
- $(63)_{10} = (00111111)_2$ et $(28)_{10} = (00011100)_2$.
- $CR(63) = CR(00111111) = (11000000)_{CR}$.

Soustraction par la méthode du complément

- **Exemple 2**

$$
\begin{array}{lll}
 & 0\ 0011100 & \leftarrow (28)_{10} \\
 & 1\ 1000000 & \leftarrow \text{CR}(63) \\
 & \text{---------} & \\
= & 1\ 1011100 & \leftarrow \text{Résultat final } (-35)_{10}.
\end{array}
$$

il n'ya pas de retenue

Soustraction par la méthode du complément

- **Exemple 2**

```
  0 0011100        ←(28)10
  1 1000000        ←CR(63)
  ---------
= 1 1011100        ←Résultat final (-35)10.
```

- Il n'ya pas de retenue, le résultat est négatif, alors on calcul son CR :

  CR(11011100) = $(00100011)_2$ = $(35)_{10}$
  confirmant l'égalité : (28)10 - (63)10=(-35)10.

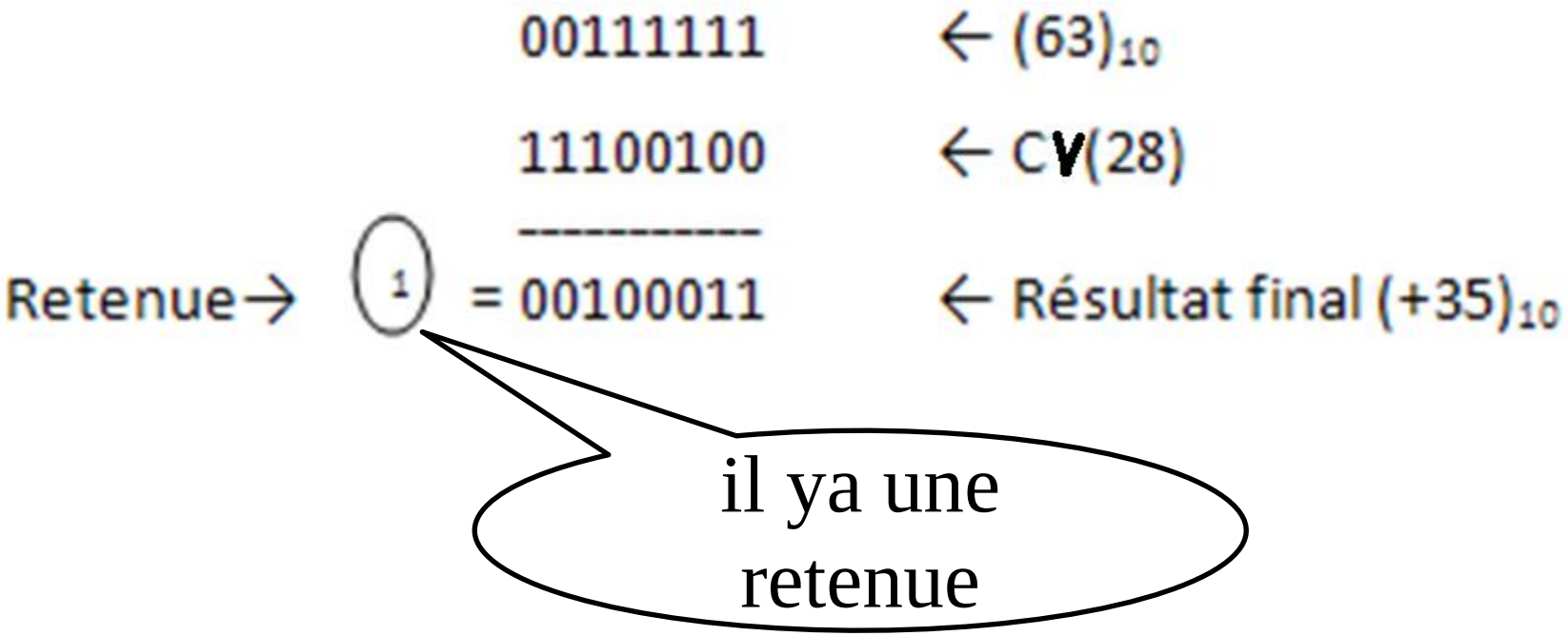
Soustraction par la méthode du complément

B/ Complément vrai:

- Le principe est le même que pour le CR sauf que cette fois-ci on ignore la retenue. Au lieu de travailler avec des CR, on détermine des compléments Vrais.
- **Exemple 1** : Effectuer l'opération suivante en utilisant la technique CV sur 8 bits :
- $(63)_{10}-(28)_{10} = (63)_{10} + CV(28)$
- $(63)_{10} = (00111111)_2$
- $(28)_{10} = (00011100)_2$.
- $CV(28) = CR(00011100) + 1 = (11100100)_{cv}$.

Soustraction par la méthode du complément

- Dans cet exemple, il y a une retenue, alors il faut l'ignorer. On obtient un résultat positif :
- (63)10-(28)10 = (+35)10.



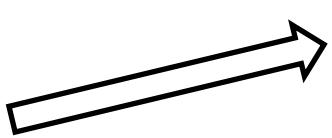
Soustraction par la méthode du complément

- **Exemple 2** :

Effectuer l'opération suivante en utilisant la technique CV sur 8 bits : $(28)_{10}$- $(63)_{10}$

- $(28)_{10}$- $(63)_{10}$ = $(28)_{10}$ + CV(63)
- $(63)_{10} = (00111111)_2$
- $(28)_{10} = (00011100)_2$
- CV(63) = CR(0 0111111) + 1

  = $(1\ 1000001)_{CV}$.



0 0011100
+ 1 1000001

Soustraction par la méthode du complément

```
  0 0011100      ← (28)10
+ 1 1000001      ← CV(63)
  ---------
= 1 1011101      ← Résultat final (-35)10.
```

- Dans cet exemple, il n'y a pas de retenue, le résultat est négatif alors on calcul son CV :
- CV(11011101) = CR(11011101) +1 = 00100010 + 1 = 00100011. On obtient au final : $(28)_{10}$-$(63)_{10}$ = $(-35)_{10}$.

Problèmes liés à la longueur des nombres

- **Rappel** : En complément vrai (ou à 2) sur *n bits*, *les nombres sont compris entre* -2^{n-1} et $+(2^{n-1}-1)$

- **Addition de deux nombres positifs :** En additionnant deux nombres positifs, on peut obtenir un résultat négatif (le bit de signe du résultat à 1). Ceci est du au fait que le résultat n'appartient pas à l'intervalle autorisé avec le nombre de bits prévu.

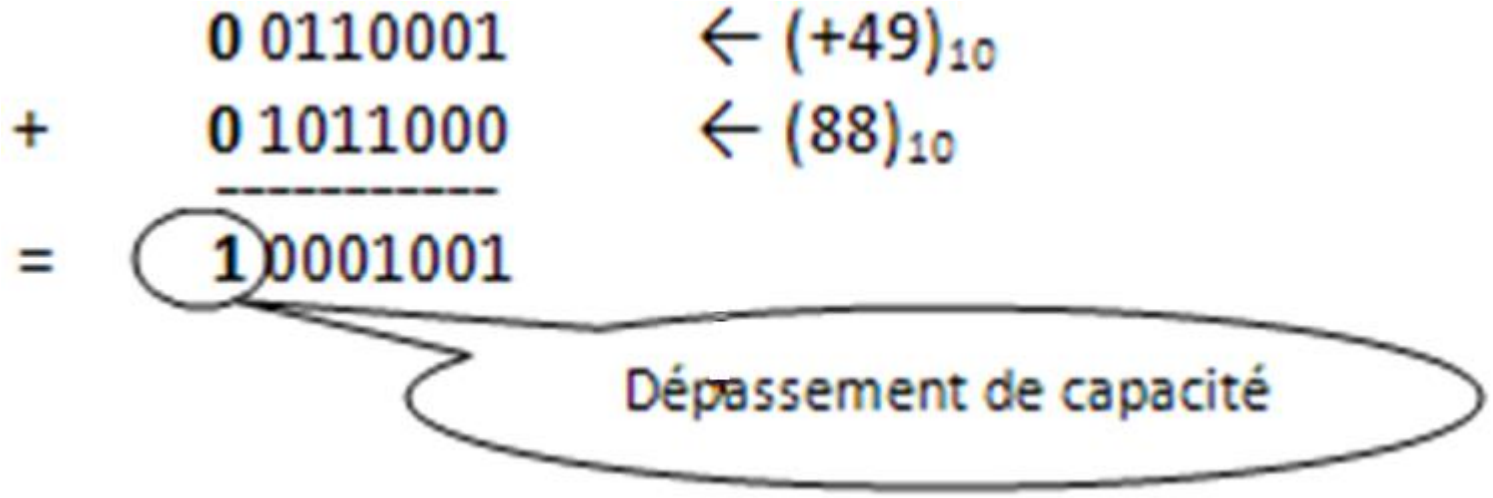
Problèmes liés à la longueur des nombres

- **Exemple** :

Effectuer l'opération suivante en utilisant la technique CV sur 8 bits : $(+49)_{10} + (88)_{10}$

- Dans cet exemple nous avons additionné deux nombres positifs, tous les deux tenant sur 8 bits. Malheureusement, nous avons obtenu un résultat qui est situé en dehors de l'intervalle des valeurs autorisées pour le codage sur 8 bits. $[- 2^{n-1} , + (2^{n-1} -1)] = [- 128 , + 127]$
- avec n = 8. En effet, le résultat de 137 (49+88=137) est en dehors de cet intervalle.

Problèmes liés à la longueur des nombres



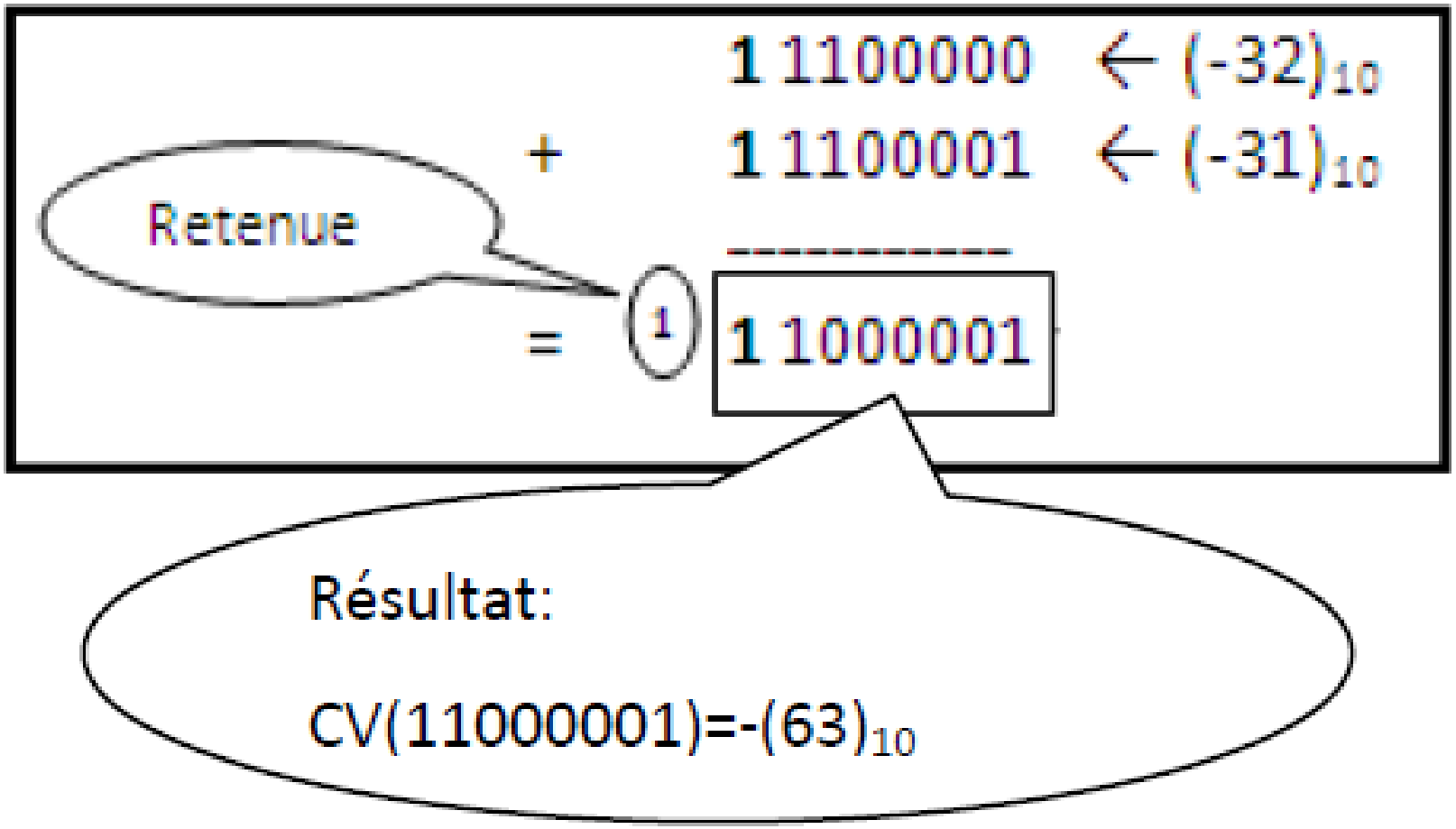
Problèmes liés à la longueur des nombres

- **Addition de deux nombres négatifs :**
- En additionnant deux nombres négatifs représentés par leurs CV (bit de signe à 1), on peut obtenir un résultat positif (le bit de signe du résultat à 0).

  En effet, il y a toujours une retenue car les bits de poids fort des nombres à additionner sont à 1.
- Exemple 1 : Effectuer l'opération suivante en utilisant la technique CV sur 8 bits : $(-32)_{10} + (-31)_{10}$

Problèmes liés à la longueur des nombres

- $(-32)_{10} + (-31)_{10}$



Problèmes liés à la longueur des nombres

- Dans cet exemple, nous avons additionné deux nombres négatifs (voir le bit de signe qui est à 1) et nous avons obtenu un nombre négatif (voir le bit de signe à 1).

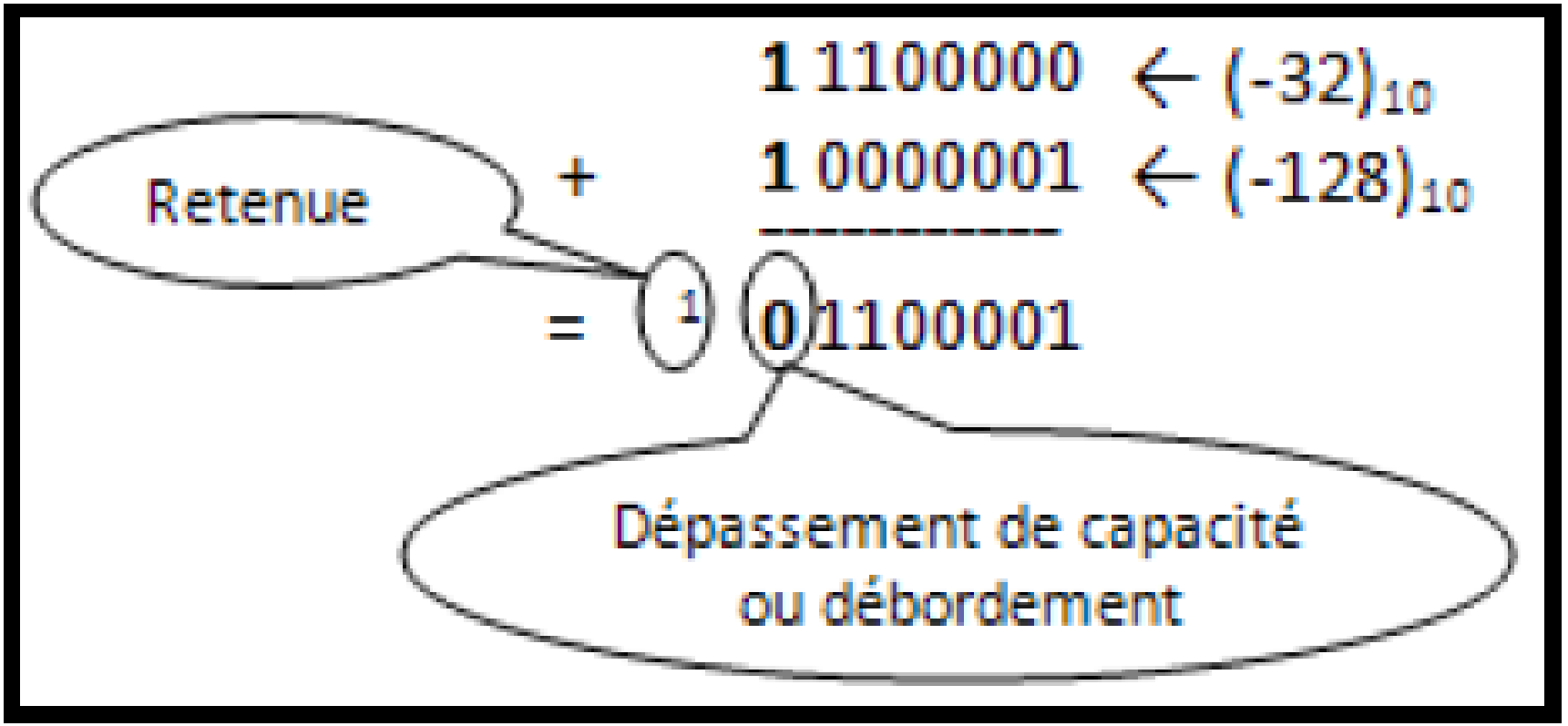
- Malgré que nous avons obtenue une retenue, notre résultat est correcte, il faut juste ignorer cette retenue (car nous utilisons ici un codage en complément à 2 ou complément vrai).

Problèmes liés à la longueur des nombres

- **Exemple 2** :

  Effectuer l'opération suivante en utilisant la technique CV sur 8 bits : $(- 32)_{10} + (- 128)_{10}$
- En ignorant la retenue, on obtient un résultat positif (bit de signe à 0) donc, on déduit qu'il y a débordement ou dépassement de capacité.
- En décimal : (-32)10 + (-127)10 = (-159)10 -159 n'est pas situé dans l'intervalle [-128 et + 127].

Problèmes liés à la longueur des nombres



Problèmes liés à la longueur des nombres

- **Indicateur de dépassement de capacité :**

- Les calculateurs utilisent un indicateur de dépassement de capacité (*Overflow) qui est mis à 1 si le bit de signe du résultat est 0 alors que les deux nombres à additionner sont* négatifs ou lorsque le bit de signe du résultat est à 1 alors que les deux nombres à additionner sont positifs.

Codage des réels

- Beaucoup d'applications ont besoin de nombres qui ne sont pas des entiers. Il y a plusieurs manières de représenter ces nombres.

  L'une d'elle est d'utiliser la virgule fixe, c'est-à-dire d'utiliser l'arithmétique entière et d'imaginer simplement la virgule binaire quelque part ailleurs qu'à la droite du chiffre le moins significatif. L'addition de deux nombres sous cette forme peut être faite avec un additionneur entier, alors que la multiplication nécessite quelques décalages supplémentaires.

Codage des réels

- Par ailleurs, il y a une seule représentation non entière qui est largement répondue : c'est la représentation en **virgule flottante**.
- Dans ce système, un mot machine est divisé en deux parties : un exposant et une mantisse.
- Afin de pouvoir effectuer correctement les opérations arithmétiques sur les nombres à représentation par la virgule flottante, il est nécessaire de procéder à leur normalisation.

Codage des réels

- ***Virgule fixe***
- La représentation de la virgule qui sépare la partie entière de la partie décimale, dans un nombre fractionnaire (réel) pose un problème au niveau de la machine.
  La première solution adoptée était de ne pas représenter matériellement la virgule, et de traiter le nombre comme s'il était entier. On dira que la virgule est fictive (ou virtuelle), elle est gérée par le programmeur, c'est lui qui définit sa position au fur et à mesure des calculs, chose qui n'est pas évidente, d'où son inconvénient.

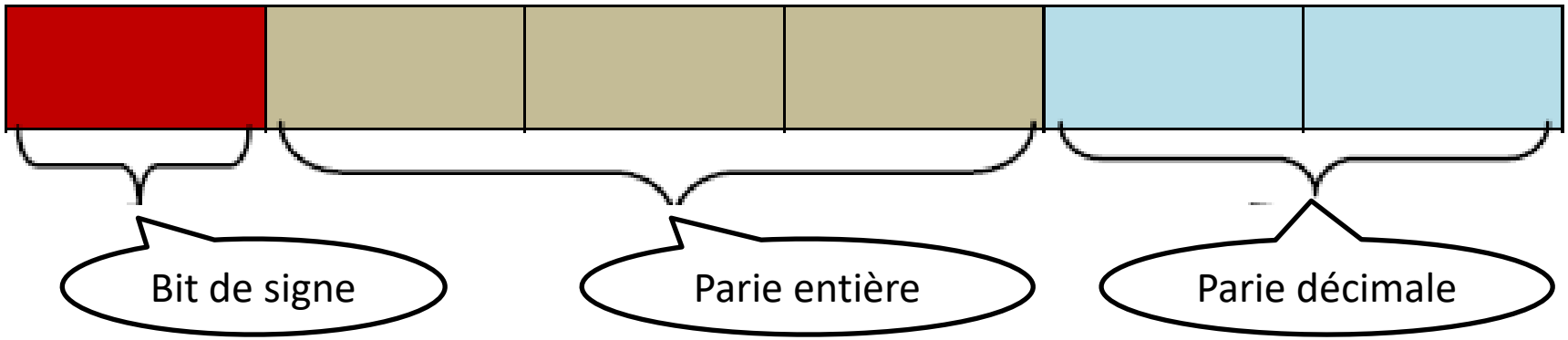
Codage des réels

- **Exemple** : On considère un nombre réel représenté sur 6 bits (en représentation binaire signée : signe + valeur absolue), tel que :

✓ 1 bit pour le signe (0 pour positif, 1 pour négatif)

✓ 3 bits pour la partie entière

✓ 2 bits pour la partie décimale



Codage des réels

- Le plus grand nombre fractionnaire représentable sur ces 6 bits est :

- La plus grande valeur absolue de la partie entière à représenter est égale à $(111)_2 = 2^3-1 = 7$.
- La plus grande valeur absolue de la partie décimale à représenter est égale à $(0,11)_2 = 0,75$.
- Ainsi, le plus grand nombre représentable est égal à +7,75.

Codage des réels

- Le plus petit nombre fractionnaire représentable sur ces 6 bits est :

- La plus grande valeur absolue de la partie entière à représenter est égale à $(111)_2 = 2^3-1 = 7$.
- La plus grande valeur absolue de la partie décimale à représenter est égale à $(0,11)_2 = 0,75$.
- Ainsi, le plus petit nombre représentable est égal à -7,75.

Codage des réels

- Le tableau ci-après donne la représentation de quelques nombres fractionnaires en représentation signe + valeur absolue sur 6 bits :

Nombre fractionnaire en représentation signe+valeur absolue			L'équivalent du nombre en décimal
0	111	11	+7,75
0	111	10	+7,50
0	000	10	+0,50
0	000	01	+0,25
0	000	00	+0,00
1	000	00	-0,00
1	000	01	-0,25
1	000	10	-0,50
1	111	10	-7,50
1	111	11	-7,75

Codage des réels

- **virgule flottante**
- Dans le monde réel, il nous arrive de manipuler des nombres appartenant à un intervalle très grand.
- Dans les nombres que nous utilisant habituellement sont en notation exponentielle :
- Par exemple pour représenter le nombre 127845000000, on peut utiliser l'une des représentations suivantes :

- $12,7\,845 \cdot 10^{7}$.
- $127,845 \cdot 10^{6}$.
- $0,127845 \cdot 10^{9}$.
- Etc.

Codage des réels

- Nous voyons bien que les représentations exponentielles évite de traîner beaucoup de chiffres souvent non significatifs. Cette nouvelle représentation est basée sur la précision d'une mantisse et d'un exposant :
- $X = \pm m.10^{e}$ où « m » est la mantisse et « e » est l'exposant
- Dans le système binaire, la notion **exponentielle** est appelée notation en **virgule flottante**. Elle est représentée de cette façon :
- $X = \pm m.2^{e}$ où « m » est la mantisse et « e » est l'exposant

Codage des réels

- **1- Décalage :**

  Pour éviter d'avoir des exposants négatifs, on utilise un exposant dit **décalé** au lieu de l'exposant simple.

  La valeur de cet **E**xposant **D**écalé (**ED**) est égale à:

  la valeur de l'**E**xposant **R**éel (**ER**), additionnée de la valeur de décalage.

  La valeur du décalage doit être suffisamment grande pour pouvoir décaler tous les exposants de valeur négative.

Codage des réels

- **Exemple :**

  On suppose un décalage de 16, de cette manière la valeur 16 est additionnée à la valeur réelle de l'exposant :

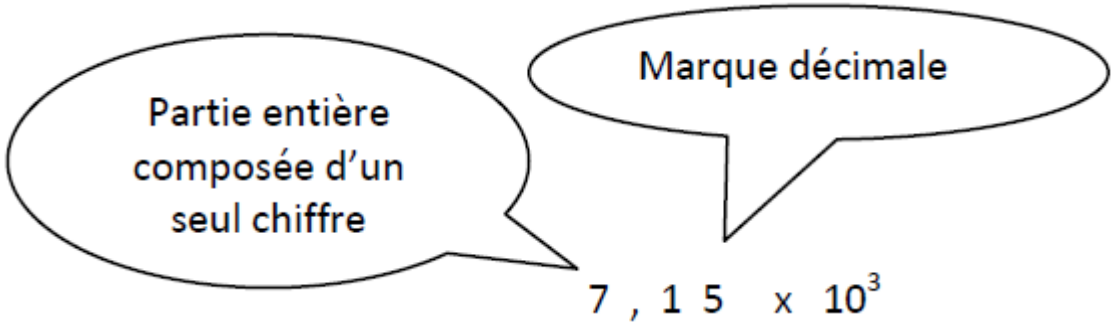
- Soit le nombre $X = 23 \times 10^{-7}$
- En appliquant le décalage de l'exposant, on obtient $16 + (-7) = 9$
- La nouvelle représentation du nombre X devient 23×10^{9}.

Codage des réels

- **2- Normalisation**

  On dit qu'un nombre notation scientifique est **normalisé** si sa partie entière n'est composée que d'un seul chiffre.

- **Exemple**
  - Le nombre $(0{,}715 \times 10^3)$ n'est pas normalisé
  - Le nombre $(7{,}15 \times 10^2)$ est normalisé
  - Le nombre $(71{,}5 \times 10^1)$ n'est pas normalisé
  - Le nombre $(715{,}0 \times 10^0)$ n'est pas normalisé



Codage des réels

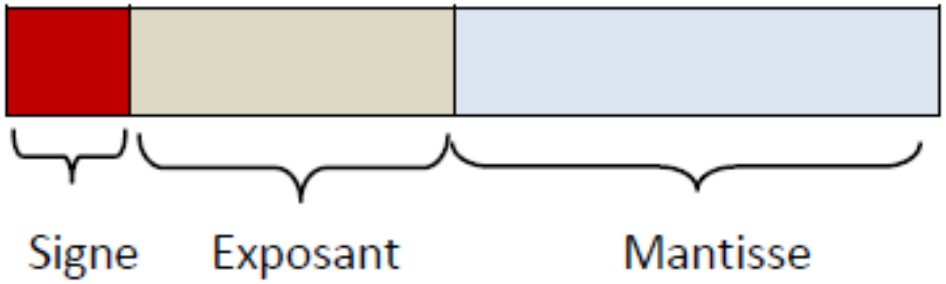
- **Représentation d'un nombre flottant avec une exponentiation en base 2**
- Avant les années 80 diverses représentations des nombres réels à virgule flottante étaient utilisées. Après 1985, une norme a été adoptée par la majorité des constructeurs d'ordinateurs c'est la **norme IEEE 754**. C'est pour cette raison que l'on ne va présenter que cette représentation.
- D'une façon générale, un nombre à virgule flottante est représenté dans les ordinateurs comme une succession de bits décomposée en trois zones :

Codage des réels

D'une façon générale, un nombre à virgule flottante est représenté dans les ordinateurs comme une succession de bits décomposée en trois zones :

- Bit de signe
- Exposant
- Mantisse

Dans les exemples ci-après, on va adopter le format suivant pour la représentation des nombres flottants :



Codage des réels

- **La représentation IEEE 754**
- Cette représentation des nombres réels est en fait une norme internationale qui est largement reconnue et utilisée par la majorité des constructeurs d'ordinateurs actuellement. Cette norme définie trois formats pour les nombres flottants :
  - Simple précision sur 32 bits
  - Double précision sur 64 bits
  - Et précision étendue sur 80 bits

Codage des réels

- **La représentation IEEE 754**
  - Simple précision sur 32 bits

Signe mantisse	Exposant décalé	Mantisse normalisée
1 bits	8 bits	23 bits

  - Double précision sur 64 bits

Signe mantisse	Exposant décalé	Mantisse normalisée
1 bit	11 bits	52 bits

  - Et précision étendue sur 80 bits

Signe mantisse	Exposant décalé	Mantisse normalisée
1 bit	15 bits	64 bits

Codage des réels

Que l'on soit dans l'une ou l'autre représentation, un certain nombre de **conventions** ont été prises :

1. L'ordre de représentation est fait comme suit : d'abord le signe, puis l'exposant ensuite la mantisse.
2. Les exposants sont décalés d'une valeur de décalage de sorte à éviter d'avoir recours à la représentation en complément à 2 des exposants négatifs.
3. La normalisation des nombres vers IEEE 754 consiste à s'assurer que la mantisse commence par:
   un seul chiffre à 1 avant la virgule. Ce chiffre est implicite (il **n'apparaît pas** dans la représentation).

Codage des réels

- La norme IEEE 754 prévoit aussi des cas d'exception issus de calculs. En effet, quant on fait des calculs, on peut être amené à faire des divisions par zéro, à trouver des nombres approchant $+\infty$ ou $-\infty$ etc.
  Par exemple dans la norme IEEE 754 simple précision, la valeur (mantisse=0 et exposant=255) indique que le nombre est infini. Par ailleurs la valeur 0 est représentée par une mantisse à 0 et un exposant à 0. Et enfin, quant l'exposant est nul alors que la mantisse est différente de zéro, cela indique que la représentation indiquée n'est pas un nombre.

Codage des réels

Nombre en simple précision : Le format est comme suit :

- ✓ 1 bit pour le signe
- ✓ 8 bits pour l'exposant (la plus grande valeur est 127, la plus petite est -126, le décalage est 127)
- ✓ 23 bits pour la mantisse

Nombre en double précision : Le format est comme suit :

- ✓ 1 bit pour le signe
- ✓ 11 bits pour l'exposant (la plus grande valeur est 1023, la plus petite est -1022, le décalage est 1023)
- ✓ 52 bits pour la mantisse

Représentation en IEEE 754 simple précision

1. Convertir le nombre en binaire
2. Ecrire le nombre sous **$N= (+/-)(1,m)_2*2^{ER}$**
3. Calculer l'exposant décalé ED = ER + 127, L'exposant est décalé de $2^{8-1}-1=127$; (ER: Exposant Réel)

Exemple : représenter en IEEE 754 simple précision le nombre $N= (-3.625)_{10}$

Remarque:

- Le **1** précédant la virgule n'est pas codé en machine (appelé bit cache)

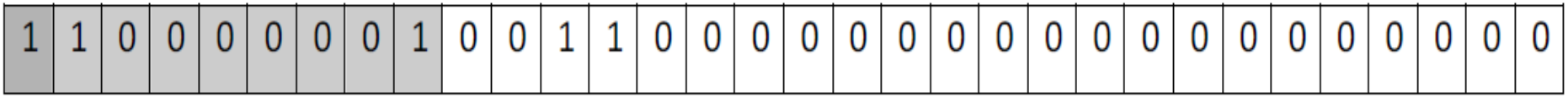
Représentation en IEEE 754 simple précision

Pour convertir un nombre écrit sous la forme IEEE754 :

1. Calculer l'Exposant Réel ER=ED-127
2. Calculer la valeur = signe (1, mantisse $)_2$ * 2^{ER}
   avec signe= ±1

- **Exemple :**
- Quelle est la valeur en décimal du nombre suivant

représenté en IEEE 754



Représentation en IEEE 754 simple précision

- On peut trouver la formule d'expression des nombres réels suivante (simple précision) :

$$(-1)^{s} \cdot 2^{(E-127)} \cdot 1,M$$

- **Remarque :**

Le 127 du (E-127) vient de $2^{\text{nbre bit de l'exposant } -1} - 1$

- Si le nombre est positif alors: ➔ S=0
- Si le nombre est positif alors: ➔ S=1

Codage des réels

- **Exemple 1**:

  La représentation du nombre réel $(-42{,}375)_{10}$ selon la norme IEEE754 en simple précision sera donc calculée comme suit : $X = \pm m.2^{e}$

Signe mantisse	Exposant décalé	Mantisse normalisée
1 bits	8 bits	23 bits

$$(-42,375)_{10} = (-101010,011)_2.$$

$$(-101010,011)_2 = (-1)^1 \times 1,01010011 \times 2^5.$$

$$ED = E + D = 5 + 127 = (132)_{10} = (10000100)_2$$

1	10000100	01010011000000000000000

Codage des réels

Exemple 2: La valeur décimale représentée en virgule flottante par le code:

$(C26D0000)_{IEEE754}$ sera donc calculée comme suit :

1	10000100	11011010000000000000000

$$ED=(10000100)_2=(132)_{10} \Rightarrow E=ED-D=132-127=(5)_{10}.$$

$$N=(-1)^S\times 1,M\times 2^E=(-1)^1\times 1,1101101\times 2^5=(-111011,01)_2$$

$$(-111011,01)_2=(-59,25)_{10}$$

- Si le nombre est négatif alors: S=1
- Si le nombre est positif alors: S=0

Codage des réels

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 les erreurs, on appelle alors cette configuration NaN (Not a Number)

- Il faut ajouter 127 à l'exposant (cas de la simple précision) pour une conversion de décimal vers un nombre réel binaire. Les exposants peuvent ainsi aller de: -256 à 255.

Codage de l'information

- **Notion de codage de l'information**

- l'information est codée en binaire et uniquement en binaire. C'est la raison pour laquelle nous avions traité le système de numération binaire au premier chapitre.
- Mais cela, n'est pas suffisant: Il faut pouvoir coder des nombres, du texte, des images, du son, de la vidéos et divers autres objets informatique

Codage de l'information

- **1 – Codages binaires**
- Définition: un code est un ensemble de symboles (alphabet d'une langue par exemple) représentant des informations utiles. En informatique, ces symboles se résument aux deux objets que sont le « 0 » et le « 1 ». Donc, dans ce domaine, toutes les informations sont représentées sous la forme de configurations binaires. Que ce soit du texte, des images, du son, de la vidéo ou simplement des nombres, c'est le codage binaire que l'on utilise.

Le binaire pur

- Le binaire pur est aussi qualifié de binaire naturel. Ce codage a déjà été abordé dans le cadre du chapitre 1 traitant des systèmes de numération.

  En effet, dans ce codage on associe à chaque entier positif la valeur qui lui correspond selon le système de numération binaire. Ainsi, en ayant ***n bits on pourra coder les valeurs comprises*** entre [0 et 2^n].
- Exemple :

  Sur 6 bits : $(35)_{10} = (100011)_2$

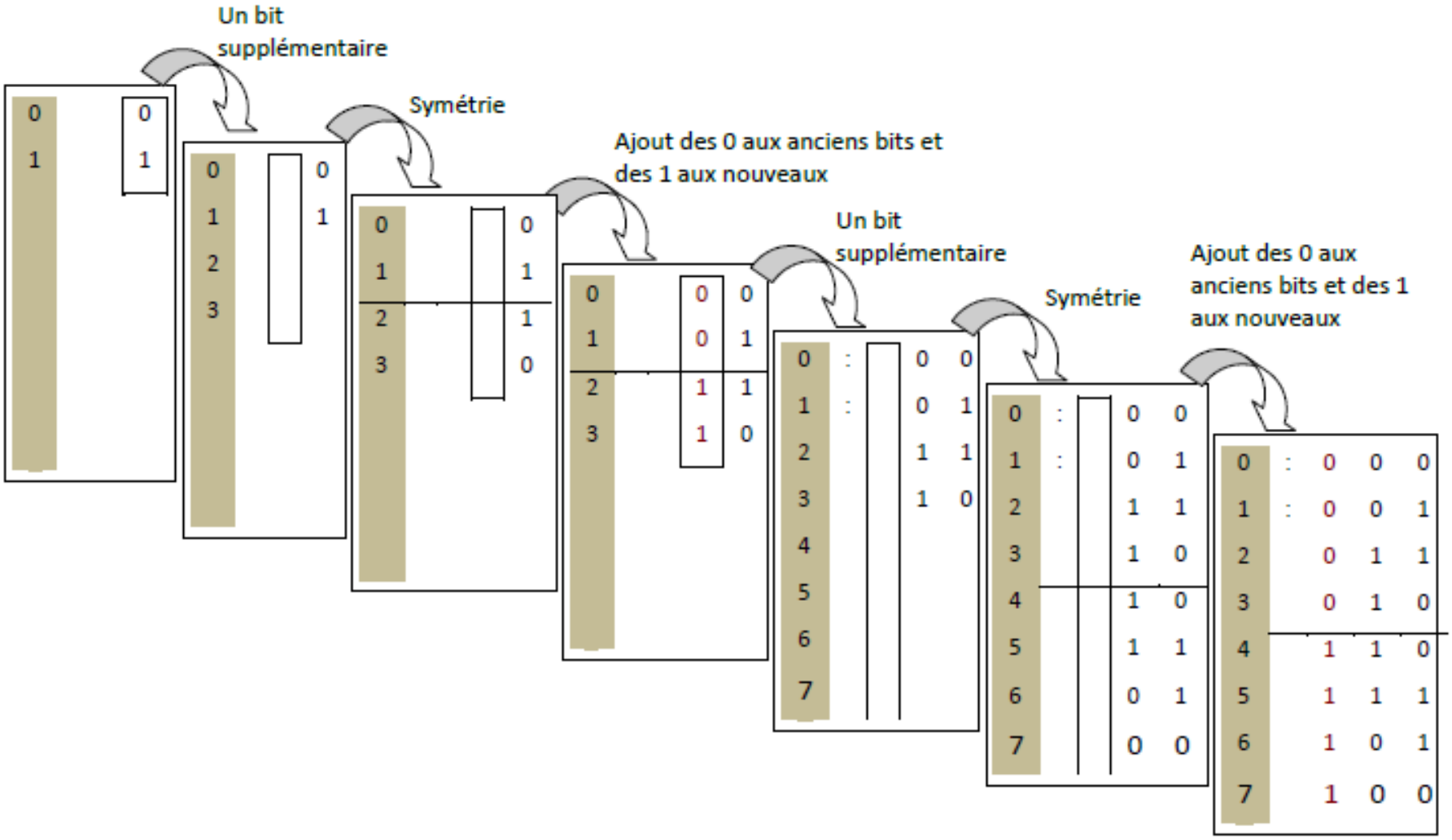
Code Gray

- **2 – Code Gray**
- Le code Gray est utilisé lorsque l'on désire une progression numérique binaire sans parasite transitoire.
- Il sert également dans les tableaux de Karnaugh utilisés lors de la conception de circuits logiques.
- Il est construit de telle façon qu'à partir du chiffre 0 chaque nombre consécutif diffère du précédent immédiat d'un seul digit (bit).

Code Gray

- Une des méthodes de construction du code Gray s'appelle la méthode du code binaire réfléchi ou code REFLEX. Voici son principe:
- 1. On établi un code de départ : zéro est codé 0 et un est codé 1.
- 2. Puis, à chaque fois qu'on a besoin d'un bit supplémentaire,
- 3. on symétrise les nombres déjà obtenus (comme une réflexion dans un miroir)
- 4. et on rajoute un 1 au début des nouveaux nombres et un zéro sur les anciens.

Code Gray


Un bit supplémentaire
Symétrie
Ajout des 0 aux anciens bits et des 1 aux nouveaux
Un bit supplémentaire
Symétrie
Ajout des 0 aux anciens bits et des 1 aux nouveaux

Code Gray

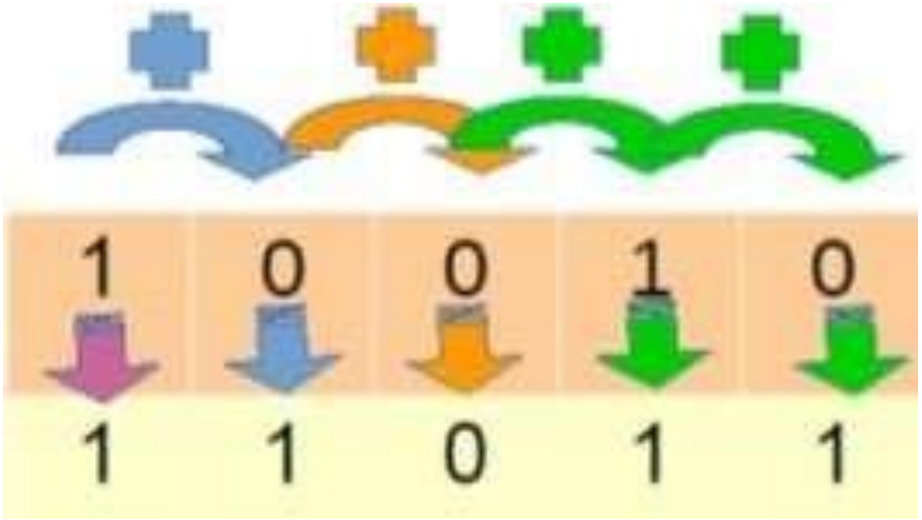
- **Il existe d'autres méthodes pour calculer code de gray**

Le premier Bit à gauche reste le même

Puis, De gauche à droite faire la somme des bits adjacents sans retenue

Exemple:

$(10010)_2=(11011)_{Gray}$



Les décimaux codés binaires

- Les informations traitées par l'ordinateur ne sont pas toujours converties selon le système de numération binaire. En effet, ces informations peuvent être manipulées sous forme décimale codée binaire.
  Un certain nombre de codes sont définis à cet effet :

- Code BCD (*binary coded decimal) ou DCB (décimal codé binaire) ou encore code 8421*
- Code excédent 3
- Code 2 parmi 5 (ou code de parité

1 – code BCD

Le BCD (**Binary Coded Decimal**), ou Décimal Codé en Binaire en français code décimal le plus utilisé en électronique.

Il contient des mots-code qui sont la traduction en binaire naturel (sur 4 bits) de chacun des dix chiffres du système décimal.

Principe :

Associer un code binaire (04 bits) à chaque chiffre décimal.

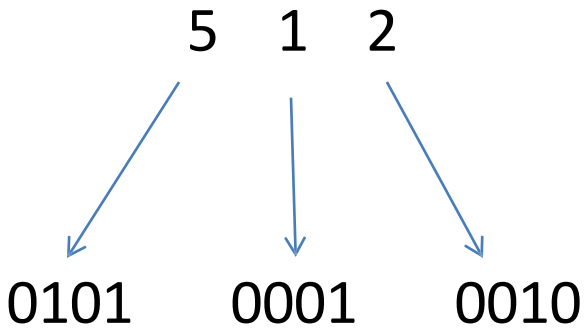
1 – code BCD

- Exemple :

Le nombre 512 en décimal équivaut à $(1000000000)_2$.

En code BCD ce nombre sera codé comme suit:
$(0101\ 0001\ 0010)_{BCD}$

La conversion de code est établie comme suit :



1 – code BCD

On peut le constaté, qu'il y a des configurations binaires non exploitées.
Vu que le nombre de configurations que l'on peut représenter avec 4 bits est 16, on déduit qu'il y 6 configurations non utilisées (entre 10 et 16).

Chiffre décimal	Code BCD	
0	0000	
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
/	1010	Non utilisées
/	1011	
/	1100	
/	1101	
/	1110	
/	1111	

1 – code BCD

- **Remarque**

Pour l'addition dans le code BCD, s'il y a une retenue où le résultat obtenu n'appartenant pas au BCD (le résultat est > 9) on doit y ajouter $6=(0110)_2$

- **Si** le résultat ne comporte pas de configurations non autorisées (entre 10 et 16), alors il est correct

  Sinon il faut rajouter la valeur 6 (0110) au résultat et cela pour chaque configuration non autorisée.

1 – code BCD

- **Exemple:** effectuer l'addition suivante: 16+25

	16	0001 0110	
+	25	0010 0101	
=		0011 1011	> 9
	+	0110	+6
		0100 0001	

1 – code BCD

- **Exemple:** effectuer l'addition suivante: 137+99

137=	0001	0011	0111
+ 99 +	0000	1001	1001
	0001	1101	0000
+	0110	0110	
	0010	0011	0110
=	2	3	6

2 – Le code excédent 3

Ce code ressemble beaucoup au code BCD.
Principe est basé sur le fait:
d'associer à chaque chiffre décimal
son équivalent binaire additionné de 3.

- **Exemple :**

512 en décimal vaut

(0101 + 0011) (0001 + 0011) (0010 + 0011)

ce qui donne (1000 0100 0101)Excédent 3

- Le tableau suivant donne l'équivalent

Excédent 3 des chiffres décimaux

Chiffre décimal	Code BCD	Code excédent 3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

2 – Le code excédent 3

- **Remarque**
- Dans le code Excédent 3: Pour l'addition,
- s'il y a une retenue on doit y ajouter **3 $(0011)_2$**
- **sinon on soustrait 3 $(-0011)_2$**

45	=	0011 | 0111 | 1000	
+90	+	0011 | 1100 | 0011	
		0111 | 0011 | 1011	
		-0011 | +0011 | -0011	
	=	0100 | 0110 | 1000	
	=	1 3 5	

137	=	0100 | 0110 | 1010	
+ 90	+	0011 | 1100 | 1100	
		1000 | 0011 | 0110	
		-0011 | +0011 | +0011	
	=	0101 | 0110 | 1001	
	=	2 3 6	

3- Code 2 parmi 5

Ce code est en fait un code correcteur.
On s'arrange à coder les chiffres sur 5 bits, mais en s'assurant que chaque configuration binaire ne comporte que deux bits à 1. Le tableau suivant donne l'équivalent des chiffres décimaux dans le code 2 parmi 5:

Chiffre décimal	Code BCD	Code excédent 3	Code 2 parmi 5
0	0000	0011	00011
1	0001	0100	00101
2	0010	0101	00110
3	0011	0110	01001
4	0100	0111	01010
5	0101	1000	01100
6	0110	1001	10001
7	0111	1010	10010
8	1000	1011	10100
9	1001	1100	11000

Représentation des caractères

- En fait, il s'agit de coder les lettres de l'alphabet y compris les chiffres et les caractères de contrôle du clavier. Le choix d'un code dépend du pays et de la langue.
- Différents codes sont utilisés:
- ASCII (7 bits)
- ASCII étendue (8 bits)
- Unicode (16 bits)
- Etc.

Représentation des caractères

- **Le code ASCII** : est une norme de codage de caractères
- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. C'est codage sur 7 bits ce qui donne 128 codes différents :
- 0 à 31 : caractères de contrôle (retour à la ligne, etc.…)
- 65 à 90 : majuscules
- 97 à 122 : minuscules
- Dans ce code, les caractères accentués (é, à,..) ne sont pas représentées et c'est la raison pour laquelle il est limité uniquement à quelques langues (anglais notamment).

Représentation des caractères

bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Les caractères repérés par un fond bleu ne peuvent pas être affichés, ce sont des caractères de contrôle.
- Pour lire le tableau, on associe au caractère le code binaire suivant: n°de ligne_n° de colonne ex: le code a est 01100001 =(97)10

Représentation des caractères

La nécessité de représenter des textes comportant des caractères non présents dans la table ASCII tels ceux de l'alphabet latin utilisés en français comme le 'à', le 'é' ou le 'ç' impose l'utilisation d'un autre codage que l'ASCII.

ASCII étendue: Afin de représenter plus de langues et notamment des langues occidentales comme la France, on a étendu le code de 7 à 8 bits.

Codage Unicode: C’est un codage sur 16 bits, Il permet de représenter tous les caractères spécifiques aux différentes langues. Il se développe de plus en plus.

Représentation des caractères

- **Codage UTF8**
- dans la pratique, pour chaque lettre il occupe 2 octets (16 bits). C’est du gaspillage car plusieurs langues partagent les mêmes lettres (français, anglais, …) notamment les lettres de l’alphabet français non accentués. Pour optimiser cela, on utilise UTF-8:

Un texte en UTF-8 est simple: il est partout en ASCII, et dès qu'on a besoin d'un caractère appartenant à l'Unicode, on utilise un caractère spécial signalant "attention, le caractère suivant est en Unicode".

Codage du son

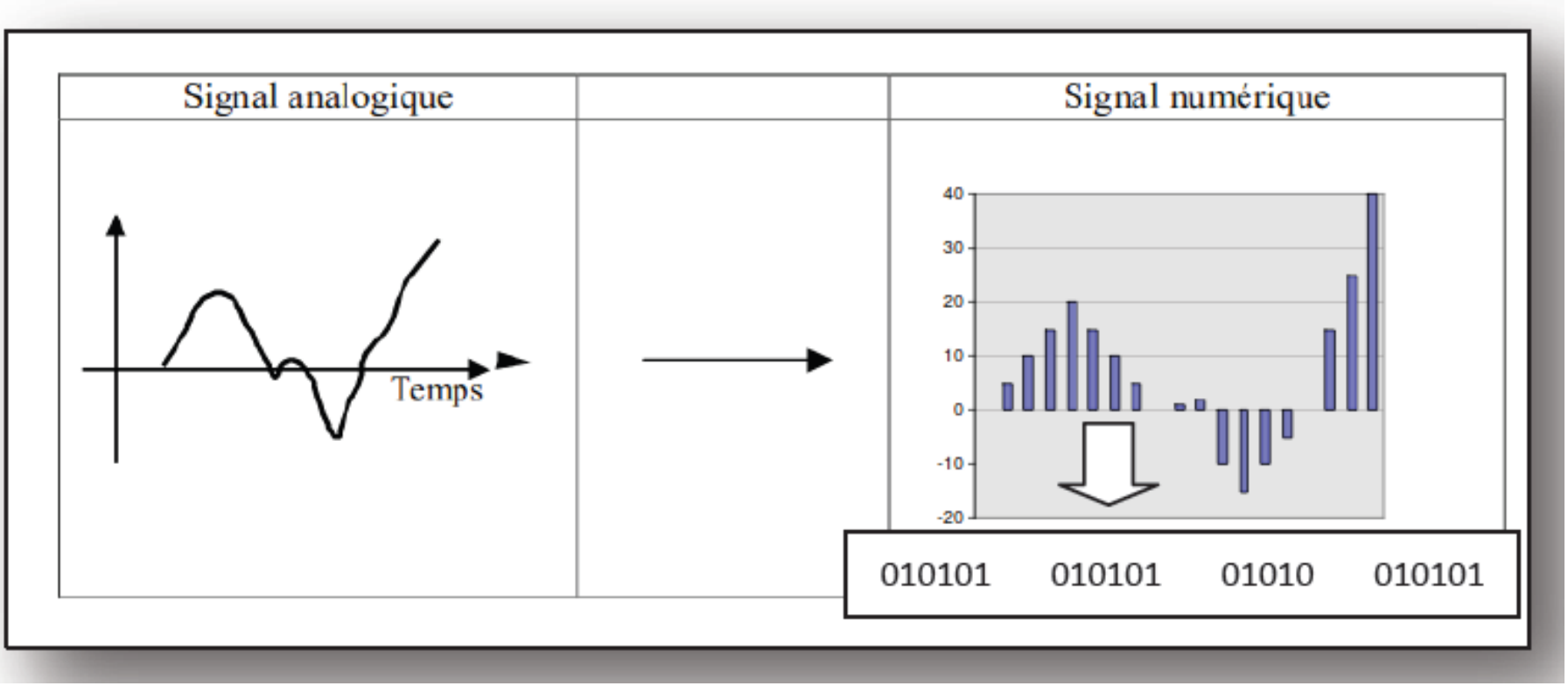
- ✓ Le son se représente naturellement sous la forme d'un signal analogique.
- ✓ Le codage d'un tel signal consiste à le représenter en binaire afin de pouvoir le transmettre à l'ordinateur qui ne peut stocker et traiter les informations qu'en binaire.
- ✓ Le signal analogique du son peut être déterminé par une fonction continue représenté par une fréquence et une amplitude.
- ✓ Pour passer d'un signal analogique vers une représentation binaire. C'est ce qu'on appelle par numérisation du son.

Codage du son

- L'échantillonnage consiste à capter la valeur de l'amplitude du signale analogique à des laps de temps répétitifs constant.
- La valeur de l'amplitude, est codée sur ***n bits à chaque période*** d'échantillonnage.
- La période d'échantillonnage est très importante car elle a un impact par rapport à la fidélité du son numérisé. Elle doit être fixer de sorte à ce que la perception humaine ne détecte pas de différences entre le signal d'origine et celui numérisé

Codage du son

- **L'échantillonnage**



Codage des images

- On distingue 2 catégories de codage des images :

Les images vectorielles :
L'image est codée par un ensemble de formules mathématique.

Les images bitmap (carte de bits) ou matricielles :
L'image est codée comme un tableau de points. C'est ce codage qui permet de représenter numériquement les photos.

Codage des images

- l’image est décrite point par point.
- Les points d’une image sont appelés des pixels.
- Chaque pixel est décrit par un nombre indiquant sa couleur.
- L’image est donc représentée par une série de nombres.

Le codage de l’image se fait en écrivant successivement les bits correspondant à chaque pixel, ligne par ligne, en commençant par le pixel en bas à gauche. Le codage est simple mais l’image bitmap occupe beaucoup de mémoire : plus les pixels sont petits, plus nombreux ils sont! Ce qui explique la nécessité de compression.

Codage des images

- Trois paramètres définissent une image bitmap :

❑ Le nombre de colonnes
❑ Le nombre de lignes
❑ Le nombre de couleur par pixel

- Les 2 premiers paramètres déterminent ce qu'on appel par définition de l'image.
- Par exemple 800x600 pixels.
- Le dernier paramètre détermine ce qu'on appelle par profondeur de l'image. C'est lui qui définie les couleurs de l'image pixel par pixel.

Codage des images

- Comment coder la couleur ?
  En se basant sur 3 couleurs primaires que sont :
- le rouge (R), le vert (V) et le bleu (B), abrégé en **RVB**.

- A partir de ces trois couleurs primaires, on peut générer toutes les autres en allant du noir jusqu'au blanc en passant par les autres.
- Du point de vue du stockage et du traitement des couleurs des pixels, on associe un nombre de bits fixe pour chacun des couleurs rouge, vert et bleu.

Codage de la vidéo

- Une vidéo est une succession d'images effectuée à un certain rythme.
- Ces images peuvent être de natures et de formats différents (dessins, graphiques, photos).
- Les animations peuvent avoir des caractéristiques différentes : Suivant qu'elles proviennent d'une série de photos ou de dessins, accompagnée d'une séquence audio ou non. Les vidéos sont composées de 2 parties synchronisées: une partie sonore et une partie vidéo (images).

Codage de la vidéo

Le nombre d'image à défiler par seconde.

- Lorsqu'il est >10, on voit un effet d'animation. En dessous de 10 images par seconde, l'animation apparaitra saccadée.
- A 25 images par seconde, l'animation se rapprochera de la réalité et l'œil humain ne sera pas capable de voir des imperfections dans l'animation.
- Au cinéma, le nombre d'images par seconde est normalisé à 24. À la télévision, le système européen PAL (ou SÉCAM en France) est de 25 images par seconde. Aux États-Unis et au Japon, la norme NTSC est de 30 images par seconde.

Fin du chapitre