Conversion entre bases

- Pour passer d'un nombre en base b à un nombre en base 10, on utilise l'écriture polynomiale décrite précédemment.
- Pour passer d'un nombre en base 10 à un nombre en base b, on peut utiliser deux méthodes :
  1. Méthode par soustraction ;
  2. Méthode par multiplication.
- Ces méthodes seront présentées grâce à des exemples, d'abord pour des entiers, ensuite pour des rationnels.
- On présentera aussi une méthode simple pour le passage entre les bases binaire, octale et hexadécimale.

Conversion d'un entier. Méthode par soustraction

Description

Soit $(n)_{10} \in \mathbb{N}^*$ à convertir en base b.
On cherche d'abord $k \in \mathbb{N}$ tel que $b^{k-1} \leq n < b^k$
on aura donc besoin de k positions $(s_{k-1} \cdots s_1 s_0)_b$

1. s_{k-1} est le nombre de fois que b^{k-1} est dans $n_1 = n$
2. s_{k-2} est le nombre de fois que b^{k-2} est dans $n_2 = n_1 - s_{k-1} b^{k-1}$
3. s_{k-3} est le nombre de fois que b^{k-3} est dans $n_3 = n_2 - s_{k-2} b^{k-2}$

$\vdots$

k-1. s_1 est le nombre de fois que b^1 est dans $n_{k-1} = n_{k-2} - s_2 b^2$

k. $s_0 = n_k = n_{k-1} - s_1 b^1 \in \{0, 1, \ldots, b-1\}$ est le reste

On détermine d'abord les digits de **plus fort poids** et ensuite les digits de **poids faible**.

Conversion d'un entier

Soit $n = 173$ à convertir en base $b = 8$.

Comme $8^2 \leq 173 < 8^3$, on a besoin de $k = 3$ positions

1. Dans $n_1 = 173$, combien de fois y a-t-il $8^2 = 64$? 2 fois (MSD) 2
   $n_2 = 173 - (2 * 64) = 45$
2. Dans 45, combien de fois y a-t-il 8 ? 5 fois 5
   $n_3 = 45 - 5 * 8 = 5$
3. On s'arrête car $n_3 = s_0 = 5$ est le reste (LSD) 5

Le résultat est donc $(173)_{10} = (255)_8$

Conversion d'un entier

Soit $n = 173$ à convertir en base $b = 2$.

Comme $2^7 \leq 173 < 2^8$, on a besoin de 8 bits

1. Dans $n_1 = 173$, y a-t-il $2^7 = 128$? oui (MSB) 1
2. $n_2 = 173 - 128 = 45$; dans 45, y a-t-il $2^6 = 64$? non 0
3. il reste donc $n_3 = 45$; dans 45, y a-t-il 32 ? oui 1
4. $n_4 = 45 - 32 = 13$; dans 13, y a-t-il 16 ? non 0
5. il reste $n_5 = 13$; dans 13, y a-t-il 8 ? oui 1
6. $n_6 = 13 - 8 = 5$; dans 5, y a-t-il 4 ? oui 1
7. $n_7 = 5 - 4 = 1$; dans 1, y a-t-il 2 ? non 0
8. il reste $n_8 = s_0 = 1$, la conversion est finie (LSB) 1

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

Conversion d'un entier. Méthode par division

Description

Soit $(n)_{10} \in \mathbb{N}^*$ à convertir en base b : $(n)_{10} = (s_{k-1} \dots s_1 s_0)_b$
On utilise la **division euclidienne**, encore appelée **division entière**.

1. on effectue la division entière de n par b :
   $n = d_1 \times b + r_1$, on garde $s_0 = r_1$
2. on effectue la division entière de d_1 par b :
   $d_1 = d_2 \times b + r_2$, on garde $s_1 = r_2$

   $\vdots$

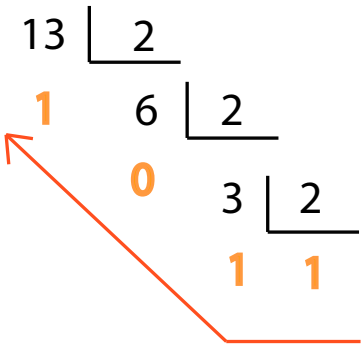
k-1. on effectue la division entière de d_{k-2} par b :
   $d_{k-2} = d_{k-1} \times b + r_{k-1}$, on garde $s_{k-2} = r_{k-1}$

k. quand $d_{k-1} \in \{0, 1, \dots, b-1\}$, $s_{k-1} = d_{k-1}$ est le reste

On détermine d'abord les digits de **faible poids** et ensuite les digits de **poids fort**.

Conversion d'un entier

Soit $n = 13$ à convertir en base $b = 2$

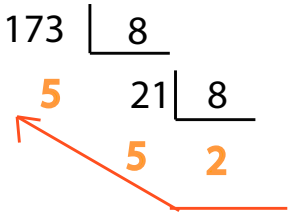


$$13 = 6 \times 2 + 1$$
$$6 = 3 \times 2 + 0$$
$$3 = 1 \times 2 + 1$$

Le résultat est donc $(13)_{10} = (1101)_2$

Conversion d'un entier

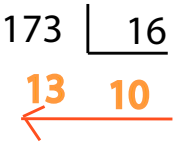
- Soit 173 à convertir en base $b = 8$



$$173 = 21 \times 8 + 5$$
$$21 = 2 \times 8 + 5$$

Le résultat est $(173)_{10} = (255)_8$

- Soit 173 à convertir en base $b = 16$



$173 = 10 \times 16 + 13$ avec $10 = A_{16}$ et $13 = D_{16}$

Le résultat est $(173)_{10} = (AD)_{16}$

Les multiples de la base b

On considère la forme polynomiale d'un entier écrit en base b

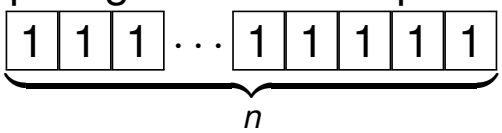
$$n_b = (s_k s_{k-1} \dots s_1 s_0)_b$$
$$= s_k b^k + s_{k-1} b^{k-1} + s_{k-2} b^{k-2} .... + s_1 b^1 + s_0 b^0$$

On constate que

1. un multiple de b se termine par 0, $s_0 = 0$;
   il s'écrit $n = b\,(s_k b^{k-1} + s_{k-1} b^{k-2} + s_{k-2} b^{k-3} + \cdots + s_1)$
2. un multiple de b^2 se termine en 00, $s_0 = s_1 = 0$;
3. un multiple de b^3 se termine en 000 ;
   et ainsi de suite.

Représentation binaire d'entiers naturels

- Avec n bits on peut représenter 2^n valeurs, c'est-à-dire dire tous les entiers de 0 à $2^n - 1$;
- Le plus grand entier représentable sur n bits s'écrit :



et vaut $2^n - 1$;

- L'entier $2^n - 1$ est toujours un nombre impair ;
- Écriture en binaire des nombres 255, 257, 260, 510, 1024, 1019.
  - D'abord chercher les puissances de 2 les plus proches :
    $255 = 2^8 - 1$, $257 = 2^8 + 1$, $260 = 2^8 + 4$,
    $510 = (2^9 - 1) - 1$, $1024 = 2^{10}$ et $1019 = (2^{10} - 1) - 4$;
  - En déduire l'écriture en base 2 :

$$\begin{array}{rclrcl} (255)_{10} &=& (11111111)_2\,, & (257)_{10} &=& (100000001)_2\,, \\ (260)_{10} &=& (100000100)_2\,, & (510)_{10} &=& (111111110)_2\,, \\ (1024)_{10} &=& (10000000000)_2\,, & (1019)_{10} &=& (1111111011)_2\,. \end{array}$$

Conversion facile : Binaire-Octal

- En informatique les bases binaire, octale et hexadécimale sont fréquemment utilisées.
- Toutes ces bases étant des puissances de deux, 2^1, 2^3 et 2^4, il y a des conversions particulièrement simples.
- Pour écrire les 8 symboles de la base octale on a besoin de trois bits

$$(0)_8 = (000)_2,\ (1)_8 = (001)_2, \ldots, (6)_8 = (110)_2\,; (7)_8 = (111)_2\,.$$

1. Pour passer de l'octal en binaire :
   on remplace chaque chiffre octal par les trois bits correspondants.
2. Pour passer du binaire en octal :
   on parcourt le nombre binaire de la *droite vers la gauche* en regroupant les chiffres binaires par paquets de 3 (en complétant éventuellement par des zéros).
   Il suffit ensuite de remplacer chaque paquet de 3 par le chiffre octal.

Conversion facile : Binaire-Hexadécimal

- Pour écrire les 16 symboles de la base hexadécimale on a besoin de quatre bits

$(0)_{16} = (0000)_2$; $(1)_{16} = (0001)_2$; $(2)_{16} = (0010)_2$; $(3)_{16} = (0011)_2$;
$(4)_{16} = (0100)_2$; $(5)_{16} = (0101)_2$; $(6)_{16} = (0110)_2$; $(7)_{16} = (0111)_2$;
$(8)_{16} = (1000)_2$; $(9)_{16} = (1001)_2$; $(A)_{16} = (1010)_2$; $(B)_{16} = (1011)_2$;
$(C)_{16} = (1100)_2$; $(D)_{16} = (1101)_2$; $(E)_{16} = (1110)_2$; $(F)_{16} = (1111)_2$.

1. Pour passer de l'hexadécimal en binaire :
   on remplace chaque chiffre hexadécimal par les quatre bits correspondants.
2. Pour passer du binaire en hexadécimal :
   on parcourt le nombre binaire de la *droite vers la gauche* en regroupant les chiffres binaires par paquets de 4 (en complétant éventuellement par des zéros).
   Il suffit ensuite de remplacer chaque paquet de 4 par le chiffre hexadécimal.

Conversion facile, exemples

1. Convertir $(A01)_{16}$ en binaire :
   - on sait que $A_{16} = (1010)_2$; $0_{16} = (0000)_2$ et $1_{16} = (0001)_2$;
   - donc $(A01)_{16} = (\underbrace{1010}_{A}\,\underbrace{0000}_{0}\,\underbrace{0001}_{1})_2$.

2. Convertir $(10110)_2$ en base 16 :
   - le regroupement par paquets de quatre donne 0001 0110 ;
   - on associe à chaque paquet le chiffre hexadécimal :

$$\underbrace{0001}_{1}\,\underbrace{0110}_{6}$$

   - d'où $(10110)_2 = (16)_{16} = (22)_{10}$

Conversion facile, exemples (2)

Pour transformer des nombres avec une partie fractionnaire on procède de la même façon,
mais les regroupements se font **de part et d'autre** de la virgule !

Exemple : conversion de $(1001101011, 11001)_2$ en base 16

- on regroupe en paquets de 4 bits de part et d'autre de la virgule

$$0010\ 0110\ 1011\ ,\ 1100\ 1000$$

- et on associe les chiffres hexadécimaux

$$\underbrace{0010}_{2}\underbrace{0110}_{6}\underbrace{1011}_{B},\underbrace{1100}_{C}\underbrace{1000}_{8}$$

- d'où

$(1001101011, 11001)_2 = (26B, C8)_{16}$
$= 2 * 16^2 + 6 * 16^1 + 11 * 16^0 + 12 * 16^{-1} + 8 * 16^{-2} = (619, 78125)_{10}$

Conversion facile, exemples (3)

Convertir $(B5, AE)_{16}$ en base 2 :

- on sait que
  $(B)_{16} = (1011)_2$, $(5)_{16} = (0101)_2$; $(A)_{16} = (1010)_2$
  et $(E)_{16} = (1110)_2$
- d'où

$$(\underbrace{B}_{1011}\underbrace{5}_{0101}\ ,\ \underbrace{A}_{1010}\underbrace{E}_{1110})_2$$

- et $(B5, AE)_{16} = (10110101, 10101110)_2$

Conversion facile (4)

Conversion de $(1001101011, 11001)_2$ en octal.

- on regroupe en paquets de 3 bits de part et d'autre de la virgule

$$001\ 001\ 101\ 011\ ,\ 110\ 010$$

- et on associe les chiffres octaux

$$\underbrace{001}_{1}\ \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{011}_{3}\ ,\ \underbrace{110}_{6}\ \underbrace{010}_{2}$$

- d'où

$$(1001101011, 11001)_2 = (1153, 62)_8$$
$$= 1 * 8^3 + 1 * 8^2 + 5 * 8^1 + 3 * 8^0 + 6 * 8^{-1} + 2 * 8^{-2} = (619, 78125)_{10}$$

Conversion de nombres avec partie fractionnaire

- Pour passer d'un nombre en base b, avec partie fractionnaire, à un nombre en base 10,
  on utilise l'écriture polynomiale décrite précédemment.

- Pour passer d'un nombre en base 10, avec partie décimale, à un nombre en base b :

  1. On transforme la partie entière, par la méthode de soustraction ou de division, par rapport à b.
  2. On transforme la partie décimale, par la méthode de soustraction ou de division mais par rapport à b^{-1}
     Note : on verra que cette méthode revient en fait à une **multiplication** par b !

Conversion de la partie fractionnaire (1)

- On s'intéresse à la partie fractionnaire (à droite de la virgule), c'est à dire aux réels dans l'intervalle $(x)_{10} \in]0,1[$ et l'on veut

$$(x)_{10} = (0, s_{-1}s_{-2}\cdots s_{-k}\cdots)_b \quad \text{où } s_{-k} \in \{0,1,\ldots,b-1\}, k \geq 1$$

- Méthode par **soustraction**
  - on détermine d'abord les digits de **plus fort poids** et ensuite les digits de **poids faible**.
  - c'est-à-dire, dans l'ordre, les coefficients de $b^{-1}, b^{-2}, \ldots, b^{-k}, \ldots$
  - pour $k \geq 1$,
    - on détermine combien de fois b^{-k} se trouve dans
      $$x - s_{-1}b^{-1} - \ldots - s_{-(k-1)}b^{-(k-1)}$$
    - on recommence avec $b^{-(k+1)}$ et
      $$x - s_{-1}b^{-1} - \ldots - s_{-(k-1)}b^{-(k-1)} - s_{-(k)}b^{-k}$$
  - ce procédé ne s'arrête pas nécessairement.

Conversion de la partie fractionnaire (2)

- Soit $(x)_{10} \in]0,1[$, on veut

$$(x)_{10} = (0, s_{-1}s_{-2}\cdots s_{-k}\cdots)_b \quad \text{où } s_{-k} \in \{0,1,\ldots,b-1\}, k \geq 1$$

- Méthode par **multiplication**
  - on a $x = d \times b^{-1} + r$ où $d \in \{0,1,\ldots,b-1\}$ est le nombre de fois que b^{-1} est dans x et $r \in [0, x[$ est le reste
  - en pratique, au lieu de diviser par b^{-1}, on **multiplie par** b :
    - on calcule $x \times b = d + b \times r$
    - on garde $d \in \{0,1,\ldots,b-1\}$ qui est à gauche de la virgule, le reste $\tilde{x} = b \times r \in [0,1[$ est à droite de la virgule
    - si $\tilde{x} \neq 0$, on recommence en multipliant $\tilde{x}$ par b
  - ce procédé ne s'arrête pas nécessairement
  - on détermine d'abord les digits de **plus fort poids** et ensuite les digits de **poids faible** !

Conversion de la partie décimale

Convertir $(0,28125)_{10}$ en base 2 par soustraction
On détermine successivement les bits coefficients de $2^{-1}, 2^{-2}, 2^{-3}, \ldots$

	Bit
$2^{-1} = 0,50000$ est 0 fois dans $0,28125$	0
$2^{-2} = 0,25000$ est 1 fois dans $0,28125$	1
et $0,28125 - 0,25000 = 0,03125$	
$2^{-3} = 0,12500$ est 0 fois dans $0,03125$	0
$2^{-4} = 0,06250$ est 0 fois dans $0,03125$	0
$2^{-5} = 0,03125$ est 1 fois dans $0,03125$	1
et $0,03125 - 0,03125 = 0$	

Le reste étant nul, on s'arrête et

$$(0,28125)_{10} = (0,01001)_2$$

Conversion de la partie décimale

Convertir $(0,28125)_{10}$ en base 2 par multiplication

0,28125*2 = 0,5625	le coefficient de 2^{-1} est	0
0,56250*2 = 1,125	le coefficient de 2^{-2} est	1
0,12500*2 = 0,25	le coefficient de 2^{-3} est	0
0,25000*2 = 0,5	le coefficient de 2^{-4} est	0
0,50000*2 = 1,0	le coefficient de 2^{-5} est	1

Le reste étant nul, on s'arrête et

$$(0,28125)_{10} = (0,01001)_2$$

Conversion de la partie décimale

Convertir $(0,408)_{10}$ en base 2 par multiplication

- 0,408 *2 = 0,816 — le coefficient de 2^{-1} est 0
- 0,816 *2 = 1,632 — le coefficient de 2^{-2} est 1
- 0,632 *2 = 1,264 — le coefficient de 2^{-3} est 1
- 0,264 *2 = 0,528 — le coefficient de 2^{-4} est 0
- 0,528 *2 = 1,056 — le coefficient de 2^{-5} est 1
- 0,056 *2 = 0,112 — le coefficient de 2^{-6} est 0
- 0,112 *2 = 0,224 — le coefficient de 2^{-7} est 0
- 0,224 *2 = 0,448 — le coefficient de 2^{-8} est 0
- 0,448 *2 = 0,896 — le coefficient de 2^{-9} est 0
- 0,896 *2 = 1,692 — le coefficient de 2^{-10} est 1

Le processus ne s'arrête pas !
La période de longueur 100 apparaît à partir de s_{-47}

Conversion de la partie décimale en base 8

Convertir $(0,28125)_{10}$ en base 8 par multiplication

- 0,28125*8 = 2,25000 — le coefficient de 8^{-1} est 2
- 0,25000*8 = 2,00000 — le coefficient de 8^{-2} est 2

Le reste étant nul, on s'arrête et $(0,28125)_{10} = (0,22)_8$

On peut vérifier en passant par la base 2 :

On avait trouvé $(0,28125)_{10} = (0,01001)_2$

Il suffit de décomposer par paquets de 3 et écrire les symboles :

$$0\ ,\ \underbrace{010}_{2}\ \underbrace{010}_{2}$$

Codage binaire de l'information

- L'architecture actuelle des ordinateurs nécessite une représentation en binaire de toute information : $\{0, 1\}$, {faux, vrai}, {éteint, allumé}, {noir, blanc},...
- Dans un manuel chinois, le *Yi Jing* (premier millénaire av. JC), on trouve un système binaire lié au {Yin, Yang} ou {actif, passif}
- Leibniz (1646-1716) connaît ces travaux et publie en 1703 un Compte Rendu de l'Académie des Sciences au sujet de la représentation des nombres en binaire.
- Dans le cadre de ses travaux en logique, Boole (1815-1864) crée une algèbre n'acceptant que deux valeurs numériques : 0 et 1.
  C'est la naissance de l'**algèbre de Boole** ou **calcul booléen**.

Comment coder les nombres entiers en machine ?

- Il faut pouvoir représenter les entiers relatifs,
  i.e les entier naturels munis d'un signe.
- Les opérations arithmétiques $+$, $-$, $\times$ et $/$ doivent être faciles à effectuer.
- Quelle que soit l'architecture du matériel,
  la taille des **mots mémoire** est toujours limitée : 16, 32, 64,... bits.

  Il faut

  1. représenter l'information de la façon la plus compacte possible.
  2. avoir des mots mémoire de taille suffisamment grande
     afin de ne pas produire des dépassements de capacité (en anglais *overflow*).

ARIANE 5

Le vol 501 de la fusée Ariane 5 (4 juin 1996) s'est soldé par un échec

37 secondes plus tard. . .

ARIANE 5 (suite)

- La commission d'enquête a notamment relevé qu'une conversion d'un nombre en virgule flottante (sur *64 bits*) vers un entier signé (sur *16 bits*) a causé un dépassement de capacité.
- Entraînant toute une série d'actions, cette erreur a abouti à la destruction de la fusée.
- Le nombre en question provenait des mesures d'accélération horizontale de la fusée Ariane 5.
- Le code en question provenait de la fusée Ariane 4, dont l'accélération maximale pouvait être codée sur 16 bits or les accélérations sont 5 fois plus fortes pour Ariane 5 !
- Avec une perte de matériel d'une valeur totale de 370 M$, c'est l'un des bugs informatiques les plus coûteux de l'histoire.

Voir `http://fr.wikipedia.org/wiki/Vol_501_d'Ariane_5`

Codage des entiers en binaire

On va présenter quatre codages, avec leur avantages et défauts :

1. Codage binaire naturel signé
2. Décimal codé binaire
3. Codage «complément à un»
4. Codage «complément à deux»

Codage binaire naturel signé

- On fixe la longueur des mots
- Le bit de poids fort est réservé au signe

  Exemple : sur 4 bits,
  $+6$ est codé par 0110 et -6 est codé par 1110

- Avantages et inconvénients :
  - (+) Codage/décodage très facile.
  - (+) Représentation des entiers négatifs.
  - (−) Il y a deux représentations de 0.
  - (−) Les opérations arithmétiques ne sont pas faciles.

Décimal codé binaire

Dans ce système, aussi appelé **binary coded decimal** (BCD), chaque chiffre décimal est codé en binaire sur 4 bits :

$$0_{10} = 0000\ ; 1_{10} = 0001\ ; \ldots\ 8_{10} = 1000\ ; 9_{10} = 1001\ .$$

Exemple : L'entier positif 19032 est codé par

0001 1001 0000 0011 0010

Avantages et inconvénients :

- \+ Codage/décodage facile.
- \+ Pas de limitation des grandeurs représentées.
- \- On gâche de l'espace : 6 combinaisons sont non utilisées.
- \- Représentation pas naturelle du signe.
- \- Opérations arithmétiques compliquées.

Remarque : Des représentations de type BCD sont utilisés dans des systèmes de bases de données (SGBD) car elles permettent de stocker des nombres plus grands et de manière plus précise que les représentations standard.

Complément à un

La **taille des mots** est fixée à k et le **bit de signe** est placé en tête.

1. Les entiers positifs $n \in \mathbb{N}$ sont codés en binaire naturel signé ;
2. pour un entier négatif $n \in \mathbb{Z}_-$, on effectue une inversion de chaque bit du code binaire naturel de la valeur absolue $|n|$.

   Exemple : sur $k = 4$ bits, l'entier relatif -6 est codé par 1001

L'opération de **complément à un** correspond à

- l'*opération logique de négation* "$V \leftrightarrow F$", bit par bit ;
- à l'*opération arithmétique* de soustraction : si n est codé sur k bits, son complément à un est $(2^k - 1)_{10} - (n)_{10} = (11\cdots1)_2 - (n)_2$. Pour $k = 4$, $(+6)_{10} = (0110)_2$ et $(1111)_2 - (0110)_2 = (1001)_2$ ce qui est le code de $(-6)_{10}$ et non pas celui de $(9)_{10}$!

Avantages et inconvénients :

- \+ Codage (= décodage) facile.
- \- Deux représentations de zéro.
- \- Opérations arithmétiques pas commodes (*cf.* addition).