Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

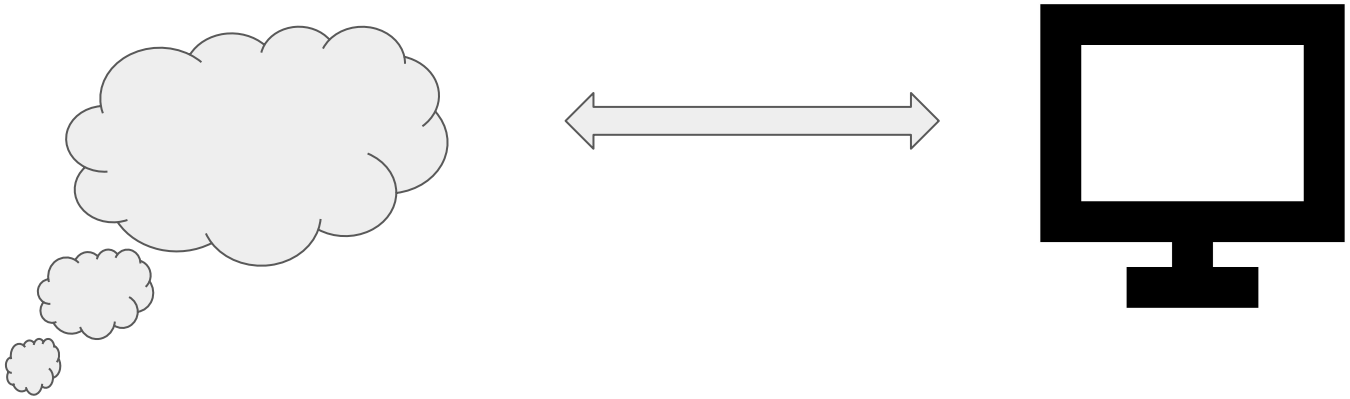
Yann Thorimbert

Chapitre 2 - Partie 1
Codage des entiers

Yann Thorimbert

Codage des nombres

Il faut maintenant passer du monde abstrait des systèmes de numération au monde concret des **états binaires** stockés dans la mémoire physique des machines.



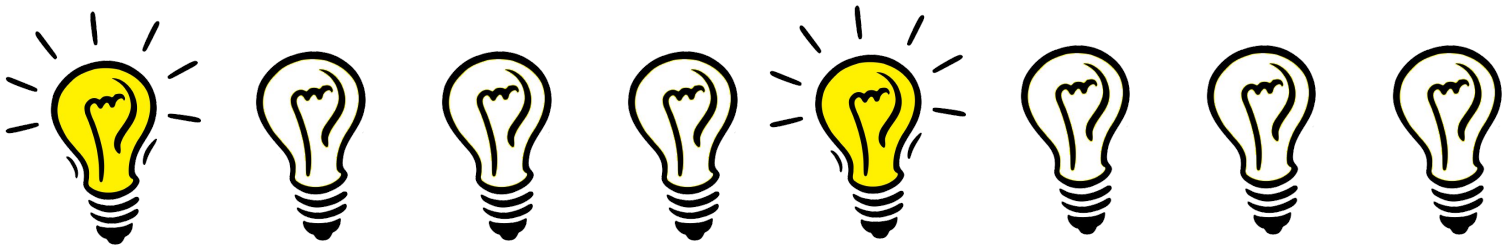
Avertissement

- Concepts fondamentaux, donc à comprendre…
- … mais temps d'apprentissage du chapitre pas proportionnel à son importance réelle

Stockage des états binaires | **Mémoire de la machine**

- Pour le moment, voyons la mémoire comme une séquence d'ampoules soit allumées, soit éteintes.

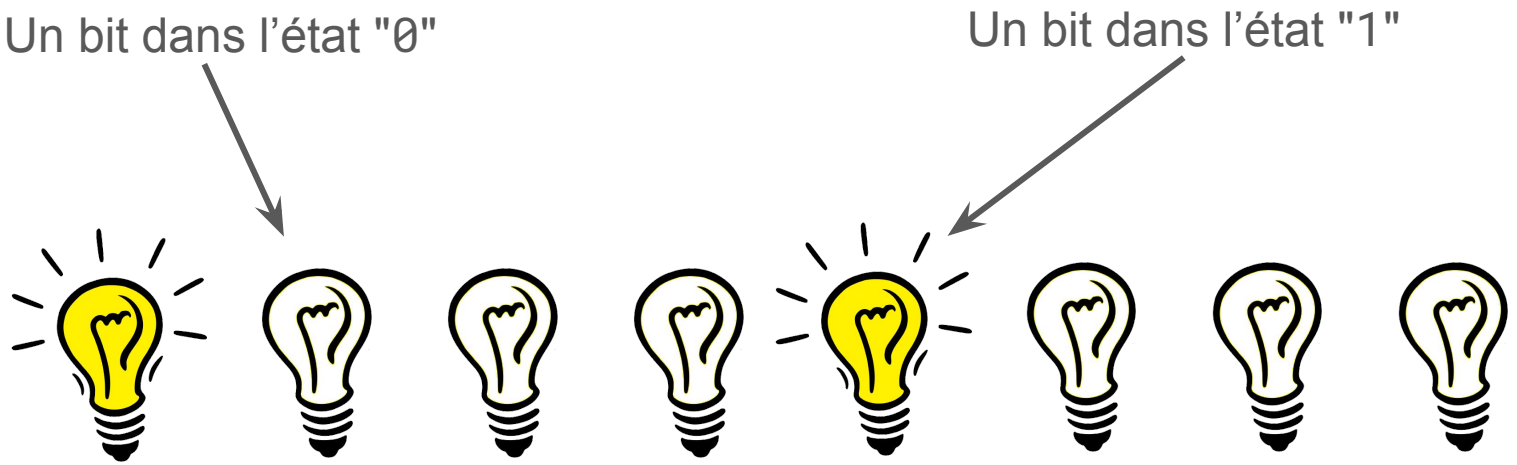
- Exemple pour un octet représentant la séquence de bits `10001000` :



Stockage des états binaires | **Vocabulaire**

Stockage des états binaires | **Vocabulaire**

Voici un octet dans l'**état** s'écrivant `10001000`



Stockage à l'aide de bits | **Nombre d'états possibles**

- Avec *k* bits, on peut fabriquer 2^k configurations différentes.
- À ne pas confondre avec la valeur maximale si on décide d'**interpréter** la configuration comme un entier binaire, qui elle peut aller de 0 à 2^k - 1.
- États possibles avec 3 bits ordonnés :

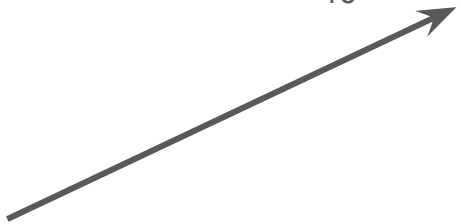
Stockage à l'aide de bits | **Usage de la base 16**

- Nombre de configurations possible d'un octet : $2^8 = 256$.
- Comme $16^2 = 256$, **l'état d'un octet peut être décrit par un nombre à deux chiffres en base 16**.
- Exemple :

correspond à ?

Conversions entre bases qui sont des puissances de 2

- Lorsque deux bases sont une puissance l'une de l'autre : **concaténation** des chiffres.
- Exemple 1 : $A7_{16} = 1010\ 0111_2$
- Exemple 2 : $57_8 = 101\ 111_2$
- Preuve en base 16, pour deux chiffres : $n_{16} = (xy)_{16} = x \cdot 16^1 + y \cdot 16^0$ $= \ldots$



Ceci n'est pas une multiplication entre les nombres x et y.

Stockage à l'aide de bits | **Usage de la base 16**

- Format hexadécimal couramment utilisé en informatique.

- Préfixe "`0x`" venant du langage C.
  Exemple : `0x`B7E4 correspond à la séquence 1011 0111 1110 0100
  (ayant besoin de 2 octets pour être codée)

- Couleurs composée de trois composantes à 256 niveaux.
  Exemple : #87CEEB correspond au triplet RGB (135, 206, 235)
  (ayant besoin de 3 octets pour être codée)

Sondage Votamatic (Notation hexadécimale)

votamatic.unige.ch : PJTF

Exemple de code Python

```
a = 0x1A #entier sous forme hexadécimale
b = 26 #entier sous forme décimale
c = 0b11010 #entier sous forme binaire
if a == c and a == b:
    print("a, b et c sont égaux !")
```

(Diapositive hors champ)

Codage des entiers naturels

Codage des entiers naturels

Notation de codage

- Attention : pour d'autres quantités que les entiers, la séquence de bits du codage ne coïncidera pas toujours avec les chiffres de la quantité à représenter, qui d'ailleurs n'est pas nécessairement un nombre !

- Exemple où l'on nomme "cantons" le codage, sans logique sous-jacente autre que la correspondance arbitraire ci-dessous :

Codage	`000`	`001`	`010`	`011`	`100`	`101`	`110`	`111`
Valeur	Genève	Vaud	Zürich	Jura	Fribourg	Berne	Neuchâtel	Valais

$\text{cod}_{\text{cantons(3)}}(\text{Jura}) = 011$ $\quad$ $\text{dec}_{\text{cantons(3)}}(101) = \text{Berne}$

Codage des entiers naturels

- Un paramètre important : la **taille *k* de la séquence** de bits, définie en pratique par le **type** des variables, lorsqu'on programme.
- Attention : les valeurs valides de *k* dépendent de l'architecture de la machine et sont gérées par le compilateur.
- Alias de types courants en C et leur **taille minimale** standard :
  - char (1 B)
  - short (2 B)
  - int (2 B) mais très souvent 4 B en pratique
  - long (4 B)
  - long long (8 B, architectures avec des mots de 64 bits uniquement)

Exemple de code C pour accéder à la taille d’un type

```
#include <stdio.h>
int main()
{
    printf("Taille d'un int: %zu octets\n", sizeof(int));
    return 0;
}
```

(Diapositive hors champ)

Codage des entiers naturels | **Débordements**

Codage des entiers naturels | **Débordements**

- Le nombre de chiffres du résultat d'une addition peut excéder celui des opérandes.

- Une solution simple, rapide et reproductible ?

- Une solution possible : **ignorer** le problème ("jeter" le bit de retenue).
  A l'avantage d'être simple, **rapide** et **reproductible**.

Codage des entiers naturels | **Débordements**

Dépassement d'entier non prévu en 1996 sur Ariane 5 (coût : + de 370 M$)

Photo: ©ESA

(Diapositive hors champ)

Codage des entiers naturels | **Ignorer le bit de poids fort**

- Exemple sur 3 bits : `111 + 001 = 1000`.
  On ignore le bit $k = 3$, c'est-à-dire la valeur $2^3 = 8$.

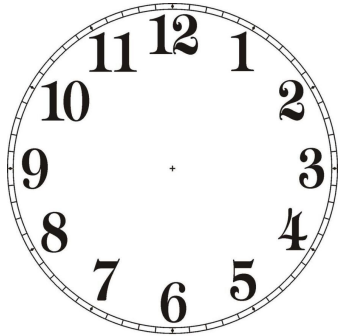
- Dans le cas où on ignore le bit $k + 1$ d'un nombre codé sur k bits, cela signifie que l'on ôte 2^{k+1} au résultat "véritable".

  ⇒ Quand on travaille avec des mots de taille constante, on obtient des débordements **cycliques** (sur les soustractions également).

Codage des entiers naturels | **Débordements cycliques**

- Exemple en base 10 sur un seul chiffre :
  $7 \cdot 5 = \mathbf{3}5$ (vrai résultat)
  $7 \cdot 5 = 5$ (gestion du débordement, on a ôté $3 \cdot 10^1 = 30$ au résultat)

- Exemple en base 12 sur 1 chiffre :
  9h + 8h = **1**7h (vrai résultat)
  9h + 8h = 5h (en ignorant le chiffre débordant)
  (gestion du débordement, on a ôté $1 \cdot 12^1$ au résultat)



Exemple de code C générant un overflow

```c
// attention, comportement indéfini selon le standard
#include <stdio.h>
int main()
{
    unsigned char a = 255; //a est codé sur 8 bits
    printf("Valeur de 'a': %d\n", a);
    a = a + 1;
    printf("Nouvelle valeur de 'a': %d\n", a);
    a = a + 1;
    printf("Nouvelle valeur de 'a': %d\n", a);
    return 0;
}
```

(Diapositive hors champ)

Exemple de code C générant un overflow

```c
// attention, comportement indéfini selon le standard
#include <stdio.h>
#include <math.h>
int main()
{
    unsigned short a = pow(2, sizeof(short)*8) - 1;
    printf("Valeur de 'a': %d\n", a);
    a = a + 1;
    printf("Nouvelle valeur de 'a': %d\n", a);
}
```

(Diapositive hors champ)

Dépassement d'entier en Python

Pourquoi l'adaptation du code précédent ne fonctionne-t-elle pas en Python ?

Codage des entiers naturels | **Gestion débordements**

- De nombreux langages modernes gèrent les débordements *via* une couche **logicielle** (par exemple Python), et permettent d'exprimer des nombres quasi-arbitrairement grands (dans les limites de la mémoire disponible).

- Dans ces cas de programmation "**haut niveau"**, le programmeur peut oublier que les nombres sont codés sur une séquence de bits (de taille variable, dans ce cas), si la performance du code n'est pas critique.

- En programmation dite "**bas niveau"**, on doit se préoccuper des débordements.

Codage des entiers | **Le cas de Python**

- Dans un langage comme le C, le codage des entiers est tel que vu dans ce cours.
- Les entiers en Python ne sont pas directement codés comme cela est vu dans ce cours. Ils suivent une approche souvent nommée "Bignum".
- En Python, la représentation des nombres entiers est automatiquement gérée par l'interpréteur (cf. chapitres suivants) et dépend donc de l'implémentation de ce dernier. Elle dépend également de la version de Python et de la taille des mots de l'architecture.

Bignums | **Principe de base**

- On stocke une séquence de chiffres de longueur "arbitraire". Exemple pour illustrer : `n = [1,0,2,3,4,0,0,7,2]` représente le nombre 102'340'072.
- Dans l'implémentation actuelle (Python 3.14) de CPython, chacun des éléments de la séquence de chiffres est exprimé en base 2^{30} (ou 2^{15} sur les architectures 32 bits), car lui-même constitué de 30 (ou 15) bits. Cf. https://github.com/python/cpython/blob/3.14/Include/cpython/longintrepr.h
- Le nombre est décodé comme $n = a_0 \cdot (2^{30})^0 + a_1 \cdot (2^{30})^1 + a_2 \cdot (2^{30})^2 + \ldots$
- Voir chapitre sur les tableaux pour le stockage de la séquence a_i.

Bignums | **Principe de base**

- Un bignum doit donc également stocker d'autres informations sur lui-même, comme la longueur de la séquence et le signe du nombre (en l'occurrence, le tout est stocké dans la même variable). La longueur maximale de la séquence de chiffres est donc celle d'un entier signé.
- Un entier en CPython sur une architecture 64 bits prends au minimum un espace de 24 B en mémoire
- L'interpréteur Python doit également utiliser une arithmétique adaptée à cette représentation.
- Le programmeur n'a (quasi) pas à se préoccuper de dépassement d'entiers.

Codage des entiers relatifs

$\mathbb{Z}$

Codage des entiers relatifs | **Codage signe-norme**

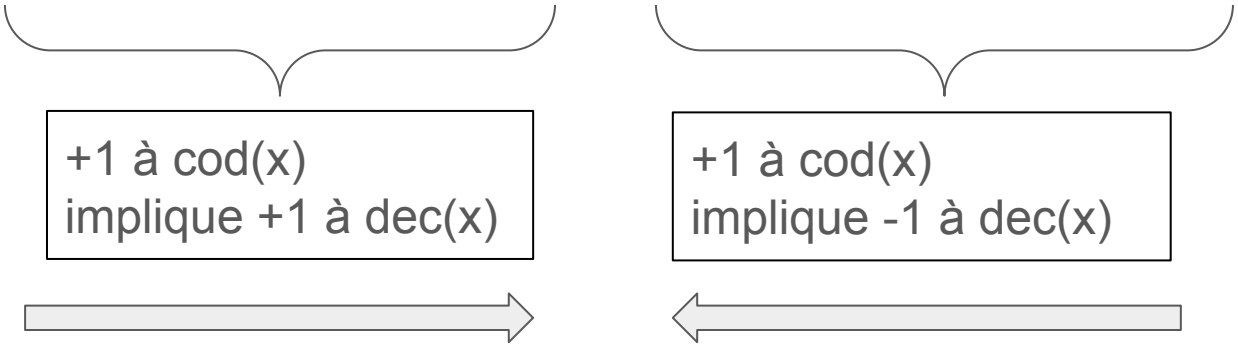
Codage des entiers relatifs | **Plage de valeurs en S-N**

Codage des entiers relatifs | **Inconvénients de S-N**

Codage des entiers relatifs | **Addition en S-N**

Sens de croissance des nombres responsable du problème d'addition

Codage	000	001	010	011	100	101	110	111
Interprétation	0	1	2	3	-0	-1	-2	-3



Codage des entiers relatifs | **Codage avec biais**

Codage des entiers relatifs | **Codage avec biais**

Principe : traiter les nombres relatifs exactement comme les nombres naturels avec un **biais implicite**, sorte de choix de référentiel.

codage	000	001	010	011	100	101	110	111
$dec_{\mathbb{N}(3)}$	0	1	2	3	4	5	6	7
$dec_{\mathbb{Z}B(3)}$								

Codage des entiers relatifs | **Codage avec biais**

Transformation pour obtenir le nombre codé :

Codage des entiers relatifs | **Codage avec biais**

Exemple du codage et décodage de -3 sur 4 bits :

Codage des entiers relatifs | **Codage avec biais**

Toujours un problème avec l'addition. Considérons (1) + (-1) sur 3 bits.

Codage des entiers relatifs | **Codage avec biais**

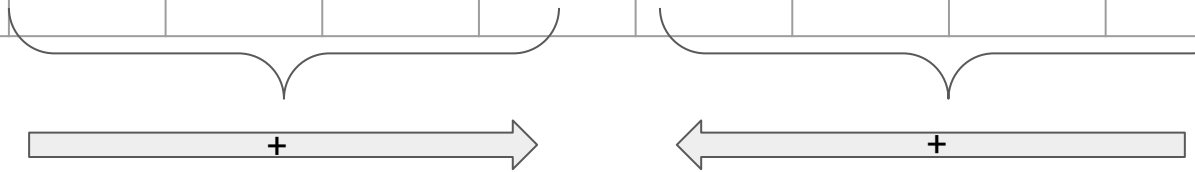
- Toutes les valeurs ont maintenant un **unique** codage. ✓
- Le résultat d'une addition ne peut être décodé directement en appliquant un biais inverse. Ne permet pas un algorithme d’addition unifié quel que soit le signe des opérandes. ❌

Codage des entiers relatifs | **Une nouvelle méthode**

Responsables (entre autres) du problème de l'addition :

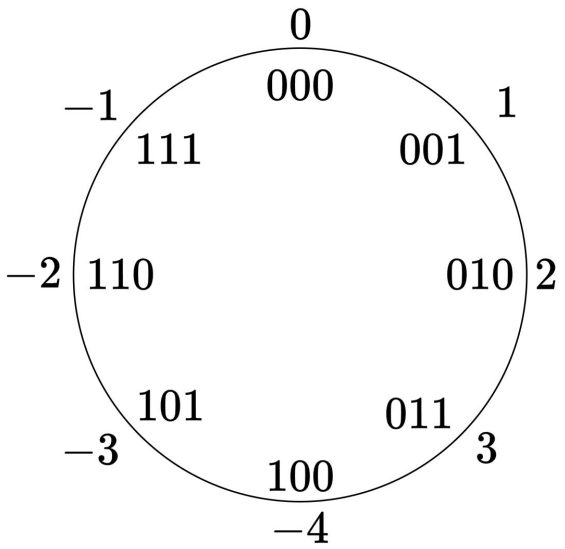
- Sens de croissance des nombres.
- cod(0) ≠ 0.

Codage	000	001	010	011	100	101	110	111
Interprétation en C2	?	?	?	?	?	?	?	?
Interprétation en codage SN	0	1	2	3	-0	-1	-2	-3



Codage des entiers relatifs | **Complément à 2**

- "Bit de signe" naturel.
- cod(0) = `0`
- Même cycle que biais, même signe que signe-norme.
- Norme des nombres négatifs obtenue de façon spéciale.

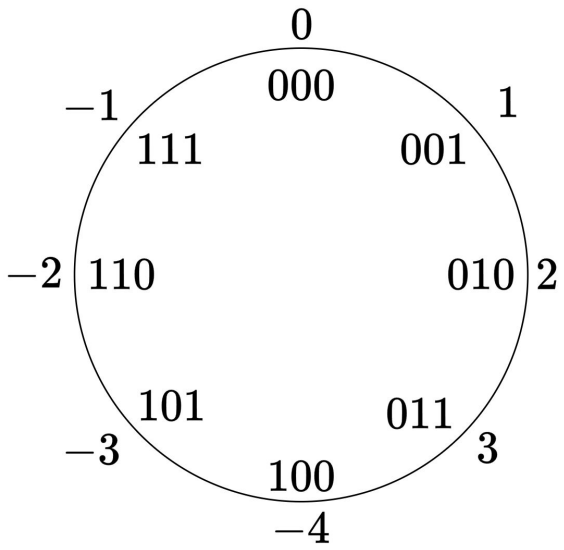


Codage des entiers relatifs | **Complément à 2**

- Addition : sens horaire.
- Soustraction : sens antihoraire.

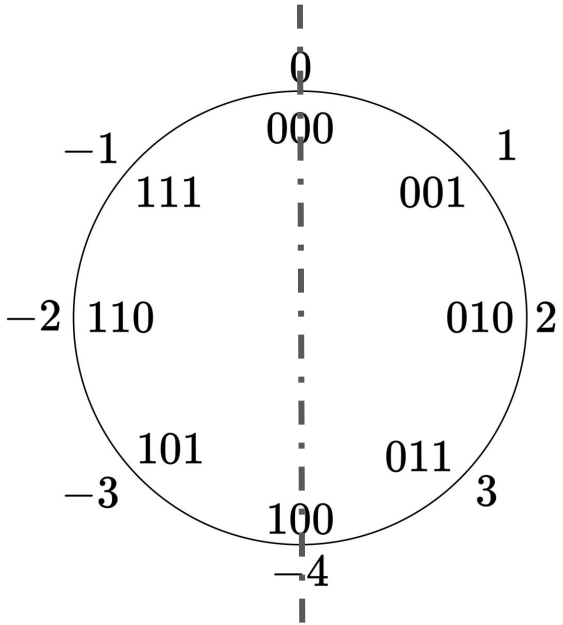
- La machine n'a pas besoin de traiter d'information relative à la nature du nombre codé car elle **effectue l'opération en représentation interne**.

- On peut utiliser le **même circuit** pour additionner des entiers naturels ou relatifs !



Codage des entiers relatifs | **Complément à 2**

- Comment trouver -*n* à partir de *n* ?
- Symétrie par rapport au zéro :
  $\text{cod}_{\mathbb{Z}(k)}(n) = \text{cod}_{\mathbb{N}(k)}(2^k - n)$
- Problème : comment le circuit peut-il calculer $2^k - n$ puisque l'on prévoit d'implémenter a - b comme a + (-b) ?



Codage des entiers relatifs | **Résumé complément à 2**

- Finalement :

  $\text{cod}_{\mathbb{Z}(k)}(n) = \text{cod}_{\mathbb{N}(k)}(n)$ si $n \geq 0$.

  $\text{cod}_{\mathbb{Z}(k)}(n) = \text{cod}_{\mathbb{N}(k)}(2^k - |n|) = \text{flip}(\text{cod}_{\mathbb{N}(k)}(|n|)) + 1$ sinon.

- Ensemble des entiers relatifs codables sur k bits en complément à 2 :
  $\mathbb{Z}_{(k)} = \{ x \in \mathbb{Z} \mid -2^{k-1} \leq x < 2^{k-1} \}$

Codage des entiers relatifs | **Résumé complément à 2**

- Codage :

  $cod_{\mathbb{Z}(k)}(n) = cod_{\mathbb{N}(k)}(n)$ si $n \geq 0$.

  $cod_{\mathbb{Z}(k)}(n) = cod_{\mathbb{N}(k)}(2^k - |n|) = flip(cod_{\mathbb{N}(k)}(|n|)) + 1$ sinon.

- Décodage :

  $dec_{\mathbb{Z}(k)}(b) = dec_{\mathbb{N}(k)}(b)$ si $b_{k-1} = 0$.

  $dec_{\mathbb{Z}(k)}(b) = dec_{\mathbb{N}(k)}(b) - 2^k = -[flip(b) + 1]$ sinon.

Aparté : comment obtenir $dec(b)$ à partir de $cod(n)$

Si n < 0 alors :

Si n > 0 alors :

Sondage Votamatic (Complément à 2)

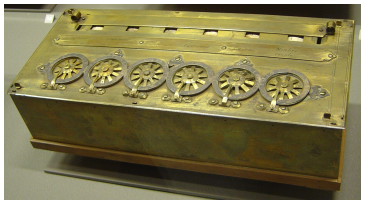
votamatic.unige.ch : PHSS

Exemple de débordement cyclique en complément à 2

La machine "sait-elle" qu'un nombre est relatif ou naturel ?

- Au moment de l'addition, le circuit ne traite aucune information quant au fait que les opérandes soient signés ou pas !

- Analogue à la Pascaline, qui travaillait en complément à 9 pour pouvoir utiliser des rouages à sens unique.



- C'est le programmeur qui choisit comment interpréter (décoder) le résultat.
  Voir exemple diapo suivante.

La machine "sait-elle" qu'un nombre est relatif ou naturel ?

```c
// Comportement indéfini - dépassement d'entier signé
#include <stdio.h>

int main()
{
    char a = 29; // NB: char codé sur 1 octet
    char b = 120;
    char c = a + b; // On crée un dépassement d'entier (149)
    printf("c : %d\n", c); // -107 si on ignore le bit de poids fort
    return 0;
}
```

(Diapositive hors champ)

La machine "sait-elle" qu'un nombre est relatif ou naturel ?

```c
#include <stdio.h>
int main()
{
    char y = -4; // en complément à 2 : état binaire 11111100
    unsigned char z = (unsigned char) y; // toujours 11111100
    printf("y = %d\n", y); // -4
    printf("z = %d\n", z); // 252
    return 0;
}
```

(Diapositive hors champ)

La machine "sait-elle" qu'un nombre est relatif ou naturel ?

```
// NB : 2^16 = 65'536
#include <stdio.h>
int main()
{
    printf("Taille d'un short : %zu octets.\n", sizeof(short));
    printf("Résultat de 2-7 décodé en tant que short signé: %hd\n",
        (short)(2-7));
    printf("Résultat de 2-7 décodé en tant que short non-signé: %hu\n",
        (unsigned short)(2-7));
    return 0;
}
```

(Diapositive hors champ)