Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

Chapitre 2 - Partie 1
Codage des entiers

Yann Thorimbert

Chapitres du cours

1. Origines des ordinateurs et des réseaux informatiques
2. **Codage des nombres** ←
3. Codage des médias
4. Circuits logiques
5. Architecture des ordinateurs
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Signal digital

- Comme on l'a vu, un signal digital peut représenter un **nombre fini de valeurs** différentes.
- On doit donc se servir de ces différentes valeurs pour exprimer tout ce que nous connaissons :
  - Les nombres
  - Les textes
  - Les sons
  - Les images
  - …

Codage sous forme de nombres

- Idée fondamentale : on peut tout coder sous forme de nombres ou de séquences de nombres.
- Exemple de codage symboles-nombres (**correspondance**) :

Symbole	Codage proposé en base 10
🍕	1
🌹	2
✉	3

- Nous devons donc commencer par être capable d'exprimer n'importe quel nombre à l'aide des différents niveaux que le signal digital peut adopter !

Systèmes de numération

- Problème : comment coder les nombres eux-mêmes ?
- Réponse : en établissant des **règles** et des **symboles** arbitraires.
- Ici, on se situe à un niveau mathématique (oublions la machine pour le moment).

Système de numération unaire

- Utilise un seul symbole.
- Très intuitif car décrit la quantité "telle qu'elle est".
- Fastidieux à utiliser.

Symbole	Signification
🍎	Un
🍎🍎	Deux
🍎🍎🍎🍎🍎🍎🍎	Sept

Systèmes de numération non positionnels

- Dans un système non positionnel, on additionne la valeur des symboles.

Symbole	Signification
🍎	Un
🌻	Trois
⭐	Cinq

- Avec le système ci-dessus, la quantité quatorze s'écrit ⭐⭐🌻🍎(ou toute autre combinaison valable). Elle utilise ici quatre symboles.

Chiffres et nombres

- Les symboles qui servent à écrire les **nombres** s'appellent des **chiffres.**
- Certains nombres sont obtenus à partir d'une **combinaison** de plusieurs chiffres, tandis que d'autres ne sont composés que d'un seul chiffre.
- Différence conceptuelle importante entre chiffre et nombre :
  « combien y a-t-il de chiffres dans ce nombre ? ».
- Notons que dans les systèmes non positionnels, le concept de zéro n'est pas pertinent.

Systèmes de numération non positionnels

- Problèmes des systèmes non positionnels :
  - Représentation non unique des nombres.
  - Fastidieux à généraliser.
  - Méthodes de calcul difficiles à appliquer.

- Solution ?

Systèmes positionnels | **La base 10 (système décimal)**

Centaines	Dizaines	Unités
0	3	8
4	0	2

$$38 =$$

$$402 =$$

Systèmes positionnels | **La base 10 (système décimal)**

Centaines	Dizaines	Unités
0	3	8
4	0	2

$$38 = 3 \cdot 10 + 8 \cdot 1$$

$$402 = 4 \cdot 100 + 0 \cdot 10 + 2 \cdot 1$$

Systèmes positionnels | **La base 10 (système décimal)**

$$n = \sum_{i=0}^{k-1} a_i \cdot 10^i$$

- *n* est la valeur du nombre exprimé par la séquence de chiffres a_i .
- Il y a *k* chiffres composant le nombre, numérotés de 0 à *k* - 1.
- a_i est le chiffre en position *i* depuis la droite, en commençant par $i = 0$.

Systèmes positionnels | **Poids forts et faibles**

- Dans le nombre 218 :
  - 2 est le chiffre de poids le plus fort.
  - 8 est le chiffre de poids le plus faible.

Systèmes positionnels | **La base 2 (système binaire)**

$$n = \sum_{i=0}^{k-1} a_i \cdot 2^i$$

- *n* est la valeur du nombre exprimé par la séquence de chiffres a_i .
- Il y a *k* chiffres composant le nombre, numérotés de *0* à *k* - 1.
- a_i est le chiffre en position *i* depuis la droite, en commençant par i = 0.

Systèmes positionnels | **La base 2 (système binaire)**

$$n = \sum_{i=0}^{k-1} a_i \cdot 2^i$$

- Exemple : $42 = \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{0} \cdot 2^0$
- Pour éviter des confusions, on écrira $42_{10} = 101010_2$
- Dans ce cours, la base 10 est sous-entendue lorsque l'indice est omis.

Systèmes positionnels | **La base 2 (système binaire)**

Convertissons le nombre 58 en binaire.

Systèmes positionnels | **La base 2 (système binaire)**

Convertissons le nombre 58 en binaire.

La puissance de 2 plus petite ou égale à 58 qui en est la plus proche est 32 = $\mathbf{2^5}$.
58 - 32 = 26.

La puissance de 2 plus petite ou égale à 26 qui en est la plus proche est 16 = $\mathbf{2^4}$.
26 - 16 = 10.

La puissance de 2 plus petite ou égale à 10 qui en est la plus proche est 8 = $\mathbf{2^3}$.
10 - 8 = 2.

La puissance de 2 plus petite ou égale à 2 qui en est la plus proche est 2 = $\mathbf{2^1}$.

$$\text{Donc } 58 = 2^5 + 2^4 + 2^3 + 2^1 = 111010_2$$

Systèmes positionnels | **La base 2 (système binaire)**

Convertissons le nombre 58 en binaire (seconde méthode, utile pour base quelconque).

58 // 2 = 29, R = **0**

29 // 2 = 14, R = **1**

14 // 2 = 7, R = **0**

7 // 2 = 3, R = **1**

3 // 2 = 1, R = **1**

1 // 2 = 0, R = **1**

Donc $58 = 2^5 + 2^4 + 2^3 + 2^1 = 111010_2$

Systèmes positionnels | **Nombres à virgule**

- On peut étendre le système positionnel à des valeurs de *i* négatives. L'endroit où *i* devient négatif est délimité par un symbole spécial : la **virgule**.
- Exemple : $42.03 = \mathbf{4} \cdot 10^{1} + \mathbf{2} \cdot 10^{0} + \mathbf{0} \cdot 10^{-1} + \mathbf{3} \cdot 10^{-2}$.
- La conversion devient plus complexe : un nombre peut avoir un **développement fractionnaire** fini dans une base mais infini dans une autre. Exemple : 3 / 10 = 0.3 en base 10, mais en base 2 on ne peut pas l'exprimer avec un nombre fini de chiffres !
- Nous reviendrons sur le sujet quand nous parlerons des **erreurs d'arrondi** du codage des nombres à virgule.

Systèmes positionnels | **Base quelconque**

$$n_b = \sum_{i=-v}^{k-1} a_i \cdot b^i$$

- *b* est la base. Si *b* > *10*, les chiffres après 9 sont souvent notés A, B, C…
- *n* est la valeur du nombre exprimé par la séquence de chiffres a_i .
- a_i est le chiffre en position *i* depuis la droite, en commençant par $i = 0$.
- Il y a *k* chiffres composant la partie entière du nombre (de *0* à *k - 1*).
- Il y a *v* chiffres composant la partie non entière du nombre (de *-v* à *-1*).

Trois questions de conversion

votamatic.unige.ch : PBSJ

Binaire | **Addition de deux nombres**

- On s’inspire de la méthode des colonnes classique
- Effectuons l’addition $11001_2 + 110011_2$

0	0	1	1	0	0	1
0	1	1	0	0	1	1

Binaire | **Addition de deux nombres**

- On s’inspire de la méthode des colonnes classique
- Effectuons l’addition $11001_2 + 110011_2$

0^1	0^1	1	1	0^1	0^1	1
0	1	1	0	0	1	1
1	**0**	**0**	**1**	**1**	**0**	**0**

- Vérification en décimal :

  $11001_2 = 25$ et $110011_2 = 51$

  $25 + 51 = 76 = 1001100_2$

Binaire | **Soustraction de deux nombres**

- On pourrait s'intéresser à la soustraction en colonnes, également inspirée de la méthode décimale…

- … mais il est souhaitable de l'implémenter comme une addition de nombres potentiellement négatifs (**unicité** de la méthode et du circuit électronique correspondant). À rediscuter par la suite.

Binaire | **Multiplication par une puissance de 2**

- Exemple sur 3 chiffres : $(a_2a_1a_0)_2 \cdot 2^k = ?$

Attention, avec cette notation nous parlons de l'écriture du nombre lui-même.

Ce n'est pas une multiplication entre les nombres a_2, a_1 et a_0.

Binaire | **Multiplication par une puissance de 2**

- Exemple sur 3 chiffres : $(a_2 a_1 a_0)_2 \cdot 2^k =$

  $(a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0) \cdot 2^k =$

  $a_2 \cdot 2^{2+k} + a_1 \cdot 2^{1+k} + a_0 \cdot 2^{0+k} + 0 \cdot 2^{k-1} + \ldots + 0 \cdot 2^0$

  ⇒ En base *b*, multiplier un nombre par b^k, c'est ajouter *k* zéros à la fin de la séquence de chiffres qui le compose, *i.e.* décaler la virgule vers la droite.

- On se convaincra aisément que **diviser par b^k** revient à décaler k fois la virgule vers la gauche

Binaire | **Multiplication de deux nombres quelconques**

- On observe que tout nombre binaire à *k* chiffres peut se décomposer comme une addition de *k* nombres binaires n'ayant qu'un seul chiffre non nul.
- Exemple : $1101_2 = 1000_2 + 0100_2 + 1_2$
- Dès lors, une multiplication d'un nombre par 1101_2 revient à additionner chacun des résultats de la multiplication avec 1000_2, 100_2 et 1_2

- $1001_2 \cdot 1101_2 = ?$

Binaire | **Multiplication de deux nombres quelconques**

- On observe que tout nombre binaire à *k* chiffres peut se décomposer comme une addition de *k* nombres binaires n'ayant qu'un seul chiffre non nul.
- Exemple : $1101_2 = 1000_2 + 0100_2 + 1_2$
- Dès lors, une multiplication d'un nombre par 1101_2 revient à additionner chacun des résultats de la multiplication avec 1000_2, 100_2 et 1_2

- $1001_2 \cdot 1101_2 = 1001_2 \cdot (1000_2 + 100_2 + 1_2) =$

  $1001000_2 + 100100_2 + 1001_2 = \ldots$

Binaire | **Plus grand nombre exprimable avec *k* chiffres**

- Si l'on dispose de k chiffres, quelle est le plus grand nombre exprimable ?
- On accepte que la séquence de chiffres $\underbrace{111\ldots1}_{k \text{ chiffres}}$ constitue le plus grand nombre.

Binaire | **Plus grand nombre exprimable avec *k* chiffres**

- Première méthode de calcul : évaluer la série géométrique.

- Seconde méthode de calcul : ?

Binaire | **Plus grand nombre exprimable avec *k* chiffres**

- Première méthode de calcul : évaluer la série géométrique.

$$n = 2^{k-1} + 2^{k-2} + \ldots + 2^0 = \sum_{i=0}^{k-1} 2^i = ?$$

- Seconde méthode de calcul : ?

Binaire | **Plus grand nombre exprimable avec *k* chiffres**

- Première méthode de calcul : évaluer la série géométrique.

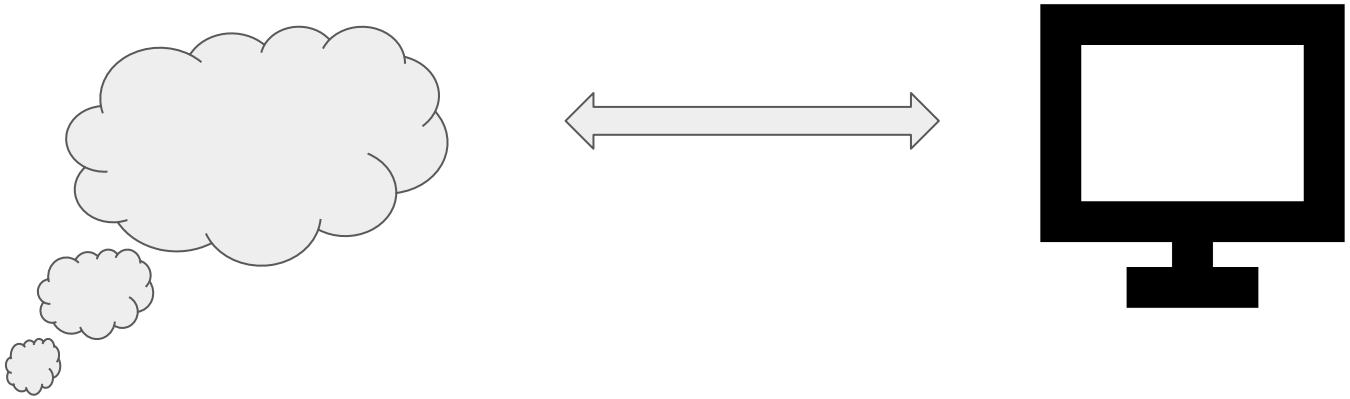
$$n = 2^{k-1} + 2^{k-2} + \ldots + 2^0 = \sum_{i=0}^{k-1} 2^i = ?$$

- Seconde méthode de calcul : n + 1 = $\underbrace{1000\ldots0}_{k+1 \text{ chiffres}}$ = 2^k

⇒ **Réponse :** $n + 1 = 2^k \Leftrightarrow \mathbf{n = 2^k - 1}$

Codage des nombres

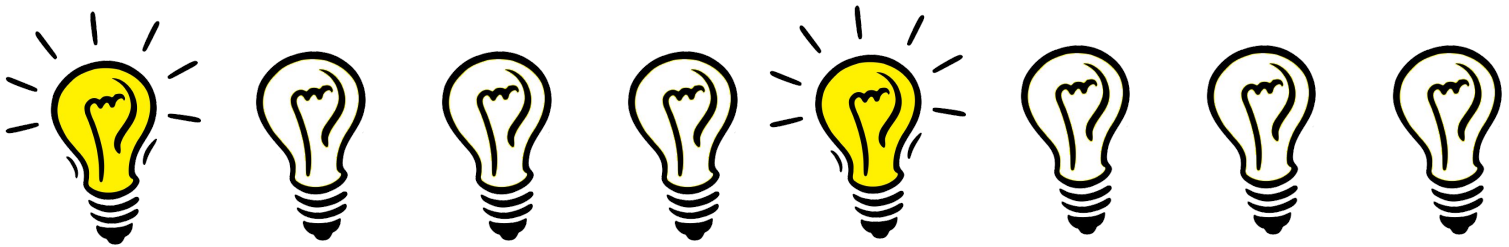
- Il faut maintenant passer du monde abstrait des systèmes de numération au monde concret des **états binaires** stockés dans la mémoire physique des machines.



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` :

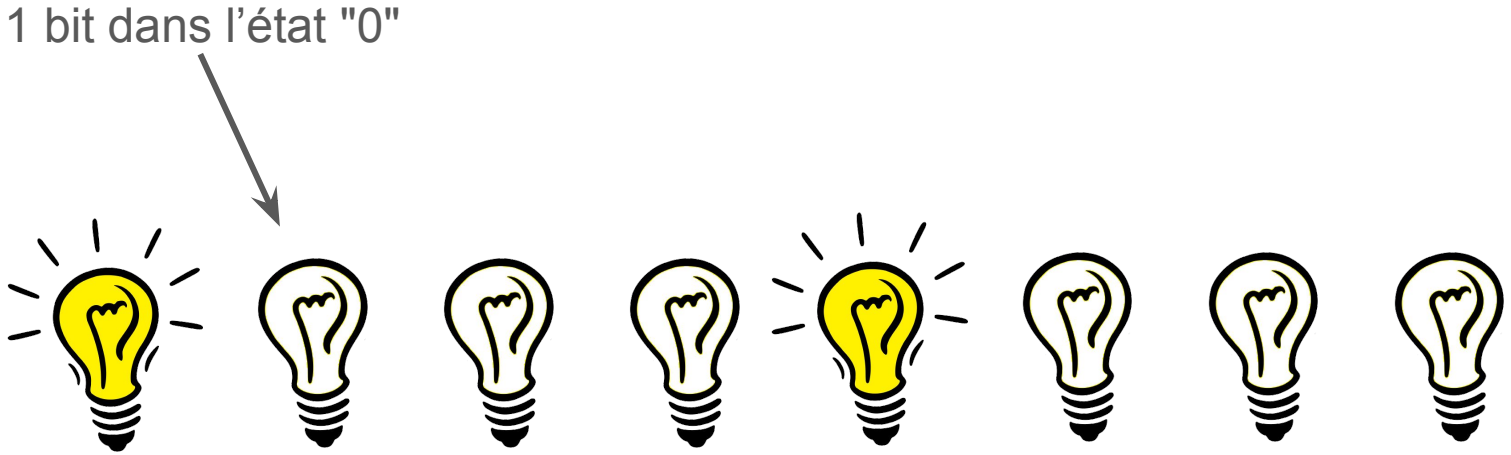


Stockage des états binaires | **Vocabulaire**

- Avant d'aller plus loin dans le codage des nombres au sein de la mémoire de l'ordinateur, il faut **définir** des termes.
- 1 **bit** : chiffre binaire. Abrégé b.
- 1 **octet** : séquence de 8 bits. Abrégé O (à éviter dans ce cours).
- 1 **byte** : 1 octet (anglais). Abrégé B.
- 1 **mot** : séquence de bits d'une taille caractéristique, constante au sein d'un processeur donné. Exemple : "architecture 64 bits".
- Dans une séquence de bits, on parle de bit de **poids** fort (arbitrairement placé tout à gauche lorsqu'on écrit) ou de poids faible (tout à droite).

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 :

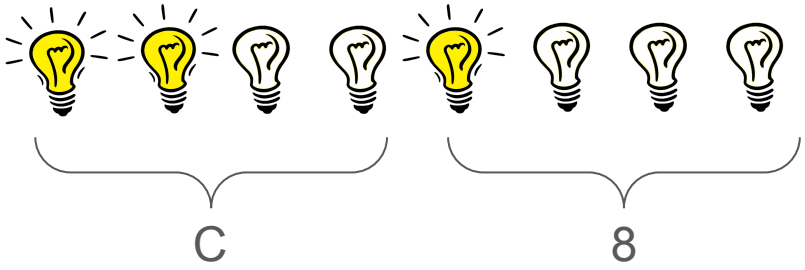
Stockage à l'aide de bits | **Nombre d'états possibles**

- 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 :
  - `000, 001, 010, 011, 100, 101, 110, 111` ⇒ ne s'interprètent pas nécessairement comme les entiers de 0 à 7 !

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

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

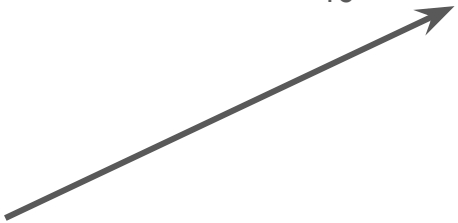
- Nombre de configurations possible d'un octet : $2^8 = 256$.
- Peu commode pour nous de retenir 256 chiffres différents.
- En revanche, comme $16^2 = 256$, **l'état d'un octet peut être décrit par un nombre à deux chiffres en base 16**.
- Exemple :



correspond à C8 en hexadécimal.

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 : $n_{16} = (xy)_{16} = x \cdot 16^1 + y \cdot 16^0$ $= \ldots$



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

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 : $n_{16} = (xy)_{16} = x \cdot 16^1 + y \cdot 16^0$

$$= (x_3 2^3 + \ldots + x_0 2^0) \cdot 16^1 + (y_3 2^3 + \ldots + y_0 2^0) \cdot 16^0$$

$$= (x_3 2^3 + \ldots + x_0 2^0) \cdot 2^4 + (y_3 2^3 + \ldots + y_0 2^0) \cdot 2^0$$

$$= x_3 2^7 + \ldots + x_0 2^4 + y_3 2^3 + \ldots + y_0 2^0 = (x_3 x_2 x_1 x_0 y_3 y_2 y_1 y_0)_2$$

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

votamatic.unige.ch : SJRF

Exemple de code C

```c
#include <stdio.h>
int main()
{
    int a =  0xF081; //entier sous forme hexadécimale
    int b =  61569; //entier sous forme décimale
    int c = 0b1111000010000001; //entier sous forme binaire
    if (a == c && a == b)
        printf("a, b et c sont égaux !\n");
    return 0;
}
```

(Diapositive hors champ)

Codage des entiers naturels

Codage des entiers naturels

- Approche la plus simple : correspondance directe entre les bits en mémoire et les chiffres du nombre représenté.

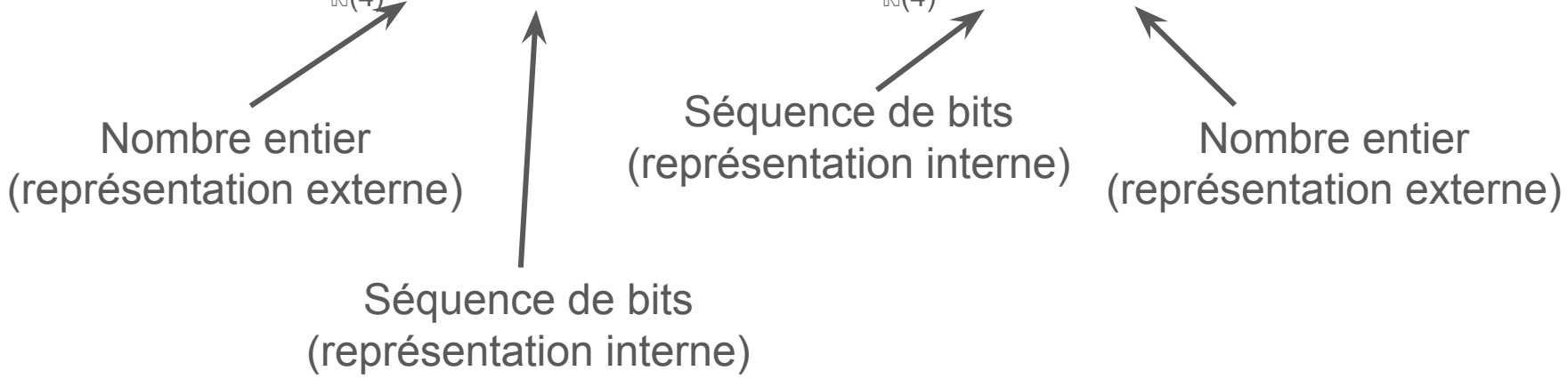
- Par exemple, la séquence de bits `1001` **s'interprète** comme représentant le nombre 9.

- On écrit : $\text{cod}_{\mathbb{N}(4)}(9) = 1001$, ou encore $\text{dec}_{\mathbb{N}(4)}(1001) = 9$.

- Pour *k* bits, on écrira : $\text{cod}_{\mathbb{N}(k)}(n)$ ou encore $\text{dec}_{\mathbb{N}(k)}(b)$

Codage des entiers naturels

- On écrit : $\text{cod}_{\mathbb{N}(4)}(9) = 1001$, ou encore $\text{dec}_{\mathbb{N}(4)}(1001) = 9$.



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

$$cod_{cantons(3)}(Jura) = 011 \qquad dec_{cantons(3)}(101) = Berne$$

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**

- Désigne le fait que le nombre de bits soit trop petit pour coder *n*.

- Ensemble des entiers naturels codables sur *k* bits :
  $\mathbb{N}_{(k)} = \{ x \in \mathbb{N} \mid 0 \leq x < 2^k \}$

- Exemple : le résultat de l'addition de deux nombres pris dans $\mathbb{N}_{(4)}$ peut n'être codable que dans $\mathbb{N}_{(5)}$ ⇒ débordement (**overflow**).

  $111_2 + 001_2 = 1000_2$

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 ?

Codage des entiers naturels | **Débordements**

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

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

- Il faut en tenir compte en programmation.

- Exemple célèbre de dépassement d'entier non prévu ?

Codage des entiers naturels | **Débordements**

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

Photo: ©ESA

Codage des entiers naturels | **Débordements**

- Exemple : `111 + 001 = ~~1~~000`.
  On ignore le bit $k = 3$, c'est-à-dire la valeur $2^3 = 8$.

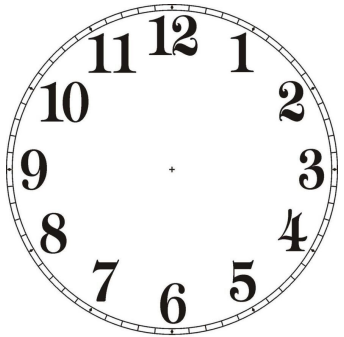
- Dans le cas où on ignore le bit $k + 1$ d'un nombre car il doit être codé sur k bits, cela signifie que l'on ôte 2^{k+1} au résultat "véritable".

  ⇒ 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 :
  $7 \cdot 5 = 35$ (vrai résultat)
  $7 \cdot 5 = 5$ (gestion du débordement, on a ôté 30 au résultat)

- Exemple en base 12 sur 1 chiffre :
  9h + 8h = 17h (vrai résultat)
  9h + 8h = 5h
  (gestion du débordement, on a ôté 12 au résultat)



Exemple de code C générant un overflow

```c
#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

```
#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)

Codage des entiers naturels | **Débordements**

- De nombreux langages modernes gèrent les débordement via une couche **logicielle** (par exemple Python), et permettent d'exprimer des nombres 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).

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

Codage des entiers naturels

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

- Approche directe : réserver un bit pour le signe.

- 1 bit de signe et *k* - 1 bits de norme.

b	a_6	a_5	a_4	a_3	a_2	a_1	a_0

- Signe $s = (-1)^b$, donc b = 0 pour les nombres positifs.
- Exemple pour *k* = 3:

Codage								
Interprétation								

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

- Approche directe : réserver un bit pour le signe.

- 1 bit de signe et *k* - 1 bits de norme.

b	a_6	a_5	a_4	a_3	a_2	a_1	a_0

- Signe $s = (-1)^b$, donc b = 0 pour les nombres positifs.
- Exemple pour $k = 3$:

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

Codage des entiers relatifs | **Représentations**

- Exemple pour $k = 3$:

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

Représentation **interne** à la machine

Représentation **externe** à la machine

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

- Quasiment autant de valeurs représentables que l'entier codé sur k bits.

- **Plage de valeurs** différentes.

  Plus grand nombre possible :

  Plus petit nombre possible :

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

- Quasiment autant de valeurs représentables que l'entier codé sur k bits.

- **Plage de valeurs** différentes.

  Plus grand nombre possible : $\mathrm{dec}_{\mathbb{Z}SN(k)}(0111\ldots1) = 2^{k-1} - 1$

  Plus petit nombre possible : $\mathrm{dec}_{\mathbb{Z}SN(k)}(1111\ldots1) = -2^{k-1} - 1$

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

Deux inconvénients du codage signe-norme :

-
-

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

Deux inconvénients du codage signe-norme :

- Un petit problème : "gaspillage" d'une valeur car le **zéro a deux représentations**. Introduit une source d'erreur possible et complexifie le test de nullité.

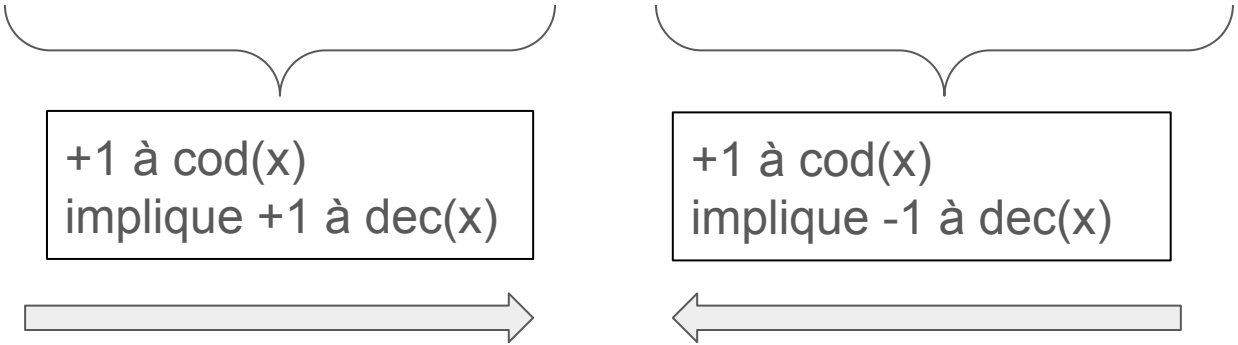
- Un gros problème : casse notre **algorithme d'addition**. Exemple : -1 + 3 ≠ -4. Raison : nombres négatifs ne "grandissent" pas dans le même sens que les positifs.

A	1	0^1	0^1	1
B	0	0	1	1
A + B	**1**	**1**	**0**	**0**

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

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**

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

- On veut "couper en deux" l'intervalle des nombres de 0 à 2^k
  ⇒ On choisit le zéro du référentiel en $2^k / 2$, c'est-à-dire en 2^{k-1} .

- Transformation pour obtenir le nombre codé :

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.

- On veut "couper en deux" l'intervalle des nombres de 0 à 2^k
  ⇒ On choisit le zéro du référentiel en $2^k / 2$, c'est-à-dire 2^{k-1} .

- Transformation pour obtenir le nombre codé : $\text{cod}_{\mathbb{Z}B\text{-}(k)}(n) = \text{cod}_{\mathbb{N}(k)}(2^{k-1} + n)$

- Ou alors : $\text{cod}_{\mathbb{Z}B\text{+}(k)}(n) = \text{cod}_{\mathbb{N}(k)}(2^{k-1} + n - 1)$
  (favorise les quantités positives).

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**

- 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
$dec_{\mathbb{Z}B(3)}$	-4	-3	-2	-1	0	1	2	3

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
$dec_{\mathbb{Z}B\text{-}(3)}$	**-4**	**-3**	**-2**	**-1**	0	1	2	3
$dec_{\mathbb{Z}B+(3)}$	**-3**	**-2**	**-1**	0	1	2	3	4

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

- Transformation pour obtenir le nombre codé :

  $\text{cod}_{\mathbb{Z}B\text{-}(k)}(n) = \text{cod}_{\mathbb{N}(k)}(2^{k-1} + n)$ et $\text{dec}_{\mathbb{Z}B\text{-}(k)}(b) = \text{dec}_{\mathbb{N}(k)}(b) - 2^{k-1}$

- Ou alors :

  $\text{cod}_{\mathbb{Z}B+(k)}(n) = \text{cod}_{\mathbb{N}(k)}(2^{k-1} + n - 1)$ et $\text{dec}_{\mathbb{Z}B+(k)}(b) = \text{dec}_{\mathbb{N}(k)}(b) - 2^{k-1} + 1$

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

- Transformation pour obtenir le nombre codé :

$\text{cod}_{\mathbb{Z}B\text{-}(k)}(n) = \text{cod}_{\mathbb{N}(k)}(2^{k-1} + n)$ $\qquad$ $\text{dec}_{\mathbb{Z}B\text{-}(k)}(b) = \text{dec}_{\mathbb{N}(k)}(b) - 2^{k-1}$

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

$\text{cod}_{\mathbb{Z}B\text{-}(4)}(-3) = \text{cod}_{\mathbb{N}(4)}(2^{4-1} + (-3))$ $\qquad$ $\text{dec}_{\mathbb{Z}B\text{-}(4)}(0101) = \text{dec}_{\mathbb{N}(4)}(0101) - 2^{4-1}$

$= \text{cod}_{\mathbb{N}(3)}(5)$ $\qquad$ $= -3$

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

- Par exemple, pour coder sur 3 bits la valeur $n = 2$ en tant qu'entier relatif avec la méthode du biais, il faut écrire une séquence de bits qui correspond aux chiffres de l'entier naturel donné par $\mathrm{cod}_{\mathbb{N}(3)}(2) = 2^{3-1} + 2 - 1 = 5$.
- Toutes les valeurs ont maintenant un **unique** codage.
- Exemple sur 3 bits :

$\mathbf{cod}_{\mathbb{Z}B+(3)}$	000	001	010	011	100	101	110	111
$\mathbf{dec}_{\mathbb{Z}B+(3)}$	-3	-2	-1	0	1	2	3	4

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**

- Toujours un problème avec l'addition. Considérons (1) + (-1) sur 3 bits.
  - On voudrait que cela donne cod(0) = 011_2
  - cod(1) = 100
  - cod(-1) = 010
  - cod(1) + cod(-1) = 110 ≠ cod(0)

- Ce que l'on voudrait : cod(*n*) + cod(0) = cod(*n*) ⇔ cod(0) = 0, afin de pouvoir décoder le résultat simplement avec le biais inverse.

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

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

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

- Considérons (1) + (-1) sur 4 bits (non traité en présentiel) :
  - Devrait donner cod(0) = $2^{4-1} + 0 - 1 = 7 = 0111_2$
  - cod(1) = $2^{4-1} + 1 - 1 = 8 = 1000_2$
  - cod(-1) = $2^{4-1} + (-1) - 1 = 6 = 0110_2$
  - cod(1) + cod(-1) = $1110_2 \neq$ cod(0)

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

- Un certain agencement de la correspondance entre séquence de bits et entiers relatifs permet de respecter à la fois cod(*n*) + cod(-*n*) = 0 et l'unicité des représentations.

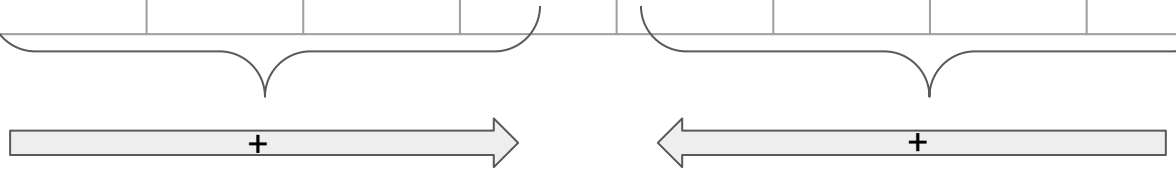
- Critères à respecter :
  - Unicité des représentations.
  - Sens de croissance des nombres.
  - cod(n) + cod(0) = cod(n) ⇔ cod(0) = 0

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
Nouveau codage	?	?	?	?	?	?	?	?
Interprétation en codage SN	0	1	2	3	-0	-1	-2	-3



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

Responsables du problème de l'addition :

- Sens de croissance des nombres.

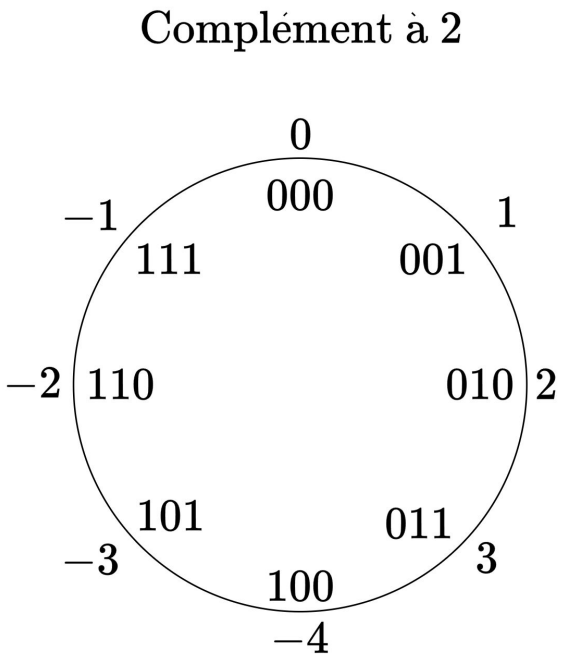
Codage	000	001	010	011	100	101	110	111
Interprétation en codage C2	0	1	2	3	-4	-3	-2	-1
Interprétation en codage SN	0	1	2	3	-0	-1	-2	-3

Débordement externe

Débordement interne

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

- Bit de signe mais sens de croissance régulier.
- cod(0) = 0
- Même cycle que biais, même signe que signe-norme.
- Il semble que la norme des nombres négatifs soit obtenue de façon spéciale.

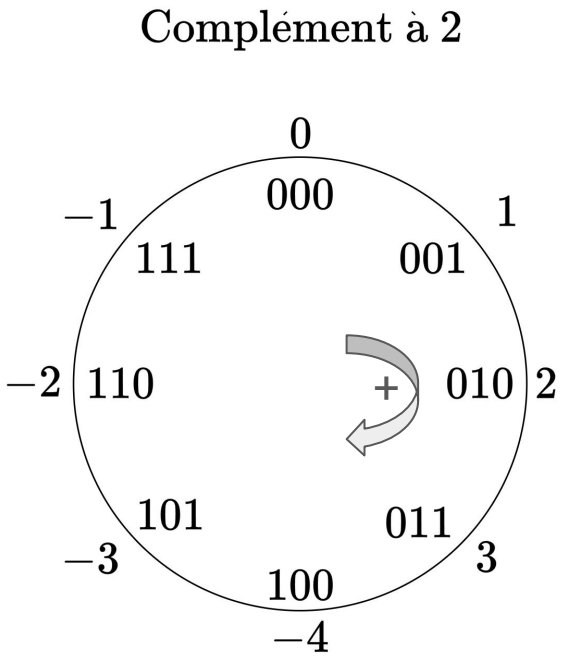


Codage des entiers relatifs | **Complément à 2**

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

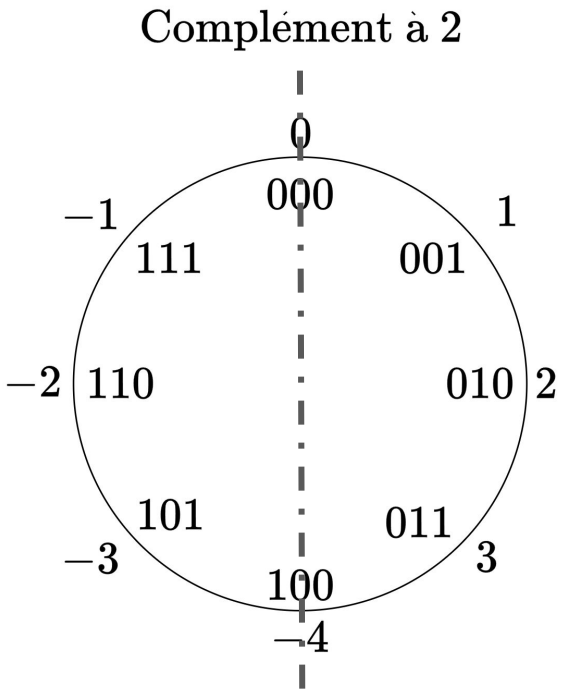
- La machine n'a pas besoin de traiter d'information relative à la nature du nombre codé car **effectue opération en représentation interne**.

- 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 | **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 | **Complément à 2**

- Comment trouver -*n* à partir de *n* ?
- Symétrie par rapport au zéro :
  $\text{cod}_{\mathbb{Z}(k)}(-n) = \text{cod}_{\mathbb{N}(k)}(2^k - n)$

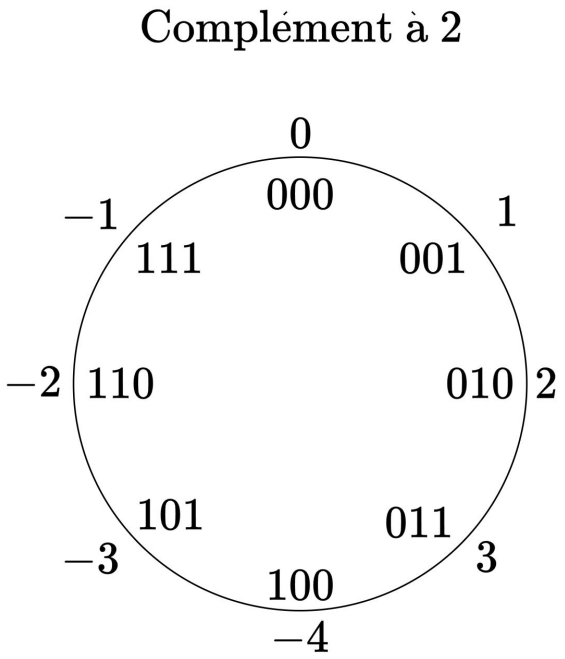
- Problème : comment le circuit peut-il calculer $2^k - n$ puisque l'on prévoit d'implémenter a - b comme a + (-b) ?

- $2^k - n = 2^k - n + 1 - 1$
  $= \underbrace{2^k - 1}_{11\ldots1_2} - n + 1 = \text{flip(n)} + 1$

Codé en complément à 2

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

- Si l'entier *n* est positif : on réutilise $cod_{\mathbb{N}}$.
- Si l'entier *n* est négatif :
  - On inverse tous les bits de $cod_{\mathbb{N}(k)}(|n|)$
  - On ajoute 1
- Exemple pour -2 :
  - $|-n| = 010_2$
  - $flip(|-n|) = 101_2$
  - $flip(|-n|) + 1 = 110_2 \Rightarrow cod_{\mathbb{N}(3)}(-2) = 110$



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

- Finalement :

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

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

- Codage :

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

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

- Décodage :

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

  $dec_{\mathbb{Z}(k)}(b) = dec_{\mathbb{N}(k)}(b) - 2^k$.

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

Si n < 0 alors :

$b := \mathrm{cod}_{\mathbb{Z}(k)}(n) = \mathrm{cod}_{\mathbb{N}(k)}(2^k - |n|) = \mathrm{cod}_{\mathbb{N}(k)}(2^k + n)$

$\Leftrightarrow \mathrm{dec}_{\mathbb{Z}(k)}(b) = n \quad \text{et} \quad \mathrm{dec}_{\mathbb{N}(k)}(b) = 2^k + n$

$\Leftrightarrow \mathrm{dec}_{\mathbb{Z}(k)}(b) = \mathrm{dec}_{\mathbb{N}(k)}(b) - 2^k$

Sinon :

$b := \mathrm{cod}_{\mathbb{Z}(k)}(n) = \mathrm{cod}_{\mathbb{N}(k)}(n) \Leftrightarrow \mathrm{dec}_{\mathbb{Z}(k)}(b) = \mathrm{dec}_{\mathbb{N}(k)}(\mathrm{cod}_{\mathbb{N}(k)}(n)) = n.$

Sondage Votamatic

votamatic.unige.ch : LXZK

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 !

- C’est le programmeur qui choisit comment interpréter (décoder) le résultat.

- Cf. exemple diapo suivante.

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

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