Représentation des entiers relatifs.

Nous avons appris à représenter des entiers naturels en représentation binaire ou hexadécimale. Ainsi en utilisant des *mots binaires* de n bits, on peut coder 2^n nombres entiers.

Par exemple sur un octet, soit 8 bits, on peut coder $2^8=256$ valeurs soit dans le cas des entiers naturels des nombres de *0 à 255*.

Cependant dans de nombreux programmes, il est nécessaire d'utiliser d'autres types de nombres comme les entiers relatifs ou les réels.

I- Méthode naïve: utilisation d'un bit de signe

La façon la plus simple de procéder serait de réserver le bit de poids fort pour le signe(0 pour positif et 1 pour négatif), et de garder le reste pour la représentation de la valeur absolue du nombre. Avec un codage utilisant des mots de n bits, on pourrait représenter des nombres entre $-2^{n-1}+1$ et $2^{n-1}-1$.

Par exemple, avec un codage sur 3 bits, des nombres entre $-2^{3-1}+1=-3$ et $2^{3-1}-1=3$:

Représentation binaire	Valeurs décimale
000	+0
001	+1
010	+2
011	+3
100	-0
101	-1
110	-2
111	-3

Malheureusement cette représentation possède deux inconvénients. Le premier (mineur) est que le nombre zéro (0) possède deux représentations. L'autre inconvénient (majeur) est que cette représentation impose de modifier l'algorithme d'addition ; si un des nombres est négatif, l'addition binaire usuelle donne un résultat incorrect.

II- Notation en complément à deux

Cette méthode permet de remédier aux problèmes évoqués ci-dessus.

On utilise toujours un bit de signe tout à gauche:

- les entiers *positifs* sont codés normalement,
- par contre on ajoute 2^n aux entiers *négatifs*.

1) Méthode d'encodage

L'entier négatif x est codé comme s'il s'agissait de l'entier $x+2^n$ ou n est la taille du mot.

Il est possible d'appliquer un algorithme simple pour réaliser cette addition en binaire *(cette méthode sera désignée comme* 2^e *méthode par la suite).*

Représentation des entiers relatifs.

- On inverse les bits de l'écriture binaire de sa valeur absolue.
- On ajoute 1 au résultat (les dépassements sont ignorés).

Avec ce codage utilisant des mots de n bits, on pourrait représenter des nombres entre -2^{n-1} et $2^{n-1}-1$.

Utilisons cet encodage sur 3 bits.

$$-1_{10} = ?_2$$

1ÈRE MÉTHODE

$$-1 => -1 + 2^3 = 7_{10} = 111_2$$

2E MÉTHODE

1. La valeur -1 a pour valeur absolue 1 codé `001` sur 3 bits.
2. On inverse les bits: `110`
3. On ajoute 1: `111`

Les deux méthodes donnent le même résultat:

$$-1_{10} = 111_2$$

Tableau de valeurs

Avec un codage sur **3** bits, on peut coder des nombres entre − $-2^{3-1}=-4$ et $2^{3-1}-1=3$.

Représentation binaire	Valeurs décimale
000	+0
001	+1
010	+2
011	+3
100	-4
101	-3
110	-2
111	-1

On peut alors vérifier avec cette notation que l'algorithme d'addition utilisé pour les entiers naturels donne des résultats corrects avec cette représentation.

2) Méthode de décodage

Pour connaître le nombre que représente un entier négatif, on effectue la démarche inverse:

- On lui retranche 1 puis,
- on inverse tous ces bits,
- On convertit en base 10, et on ajoute le signe -.

Ce qui revient à lui soustraire 2^n .

Toujours en travaillant sur 3 bits:

$$110_2 = ?_{10}$$

1ÈRE MÉTHODE

$$110_2 = 6_{10} => 6 - 2^3 = -2_{10}$$

2E MÉTHODE

1. On retranche 1: `110 - 1 = 101`
2. On inverse les bits: `010`
3. On convertit en base 10: $010_2 = 2_{10}$ donc c'est -2.

Les deux méthodes donnent le même résultat:

$$110_2 = -2_{10}$$