Cours

qkzk

Le complément à deux : comment représenter les entiers relatifs ?

Les entiers relatifs

- **Entiers naturels :** entiers positifs ou nuls $\{0, 1, 2...\}$
- **Entiers relatifs :** entiers de n'importe quel signe $\{..., -2, -1, 0, 1, ...\}$

Le problème du signe :

Un signe n'est pas un nombre... On ne peut pas l'encoder directement en binaire.

Le principe est d'attribuer au *bit de poids fort* (premier bit) le signe du nombre.

- Si le *bit de poids* fort est **0**, le nombre est **positif**,
- Si le *bit de poids* fort est **1**, le nombre est **négatif**.

Nombres encodés sur un octet

Contrainte immédiate :

Il faut que la machine sache quelle est la taille du nombre !

Sinon :

- Comment déterminer "le bit de poids fort" ?
- Comment savoir où commence le nombre ?

Dans tout ce document, on encodera **nos nombres entiers sur 8 bits.**

Considérons l'entier $42 = 101010_2$

Encodé **sur 6 bits, son bit de poids fort est 1**.

```
position :  123456
bits     :  101010
bit fort :  ^
```

Mais, le même entier, **encodé sur 8 bits a un bit de poids fort valant 0** :

```
position :  12345678
bits     :  00101010
bit fort :  ^
```

Il est donc nécessaire de **connaître la taille de l'encodage pour déterminer le bit de poids fort**.

Approche naïve : binaire signé

Essayons avec cette simple règle :

Pour encoder un entier sur 8 bits,

- On détermine la représentation binaire de sa valeur absolue
- Ensuite on remplit de `0` à gauche.

Signe

- Si le nombre est positif, on garde le bit de poids fort à 0,

- **Sinon, on met le bit de poids fort à 1.**

Ce qui fait donc :

Binaire signé sur un octet * 1 bit de signe (0: positif, 1, négatif), * 7 bits pour la valeur absolue.

On peut donc représenter de 1111 1111 à 0111 1111 soit de -127 à 127.

Exercice 1

En binaire signé sur 8 bits, quelles sont les valeurs de 1000 0000 et 0000 0000 ?

Ils valent tous deux 0... on les note -0 et +0.

Premier exemple : 27

```
27 = 0b11011

On complète sur 8 bits :
27 = 0001 1011

27 > 0 on garde le premier bit à 0
```

La représentation en binaire signé sur un octet de 27 est 0001 1011

Autre exemple : -9

```
La valeur absolue de -9 est 9.

9 = 0b1001

On complète sur 8 bits :
9 = 0000 1001

-9 < 0 on remplace le premier bit par -1 :

-9 = 1 000 1001
```

La représentation en binaire signé sur un octet de -9 est 1000 1001

Jusqu'ici tout va bien...

Et soudain, c'est le drame...

Essayons d'ajouter ces exemples :

Vérifions que $27 + (-9) = 18$

```
   0001 1011
+  1000 1001
------------
=  1010 0100
```

... mais 1010 0100 = −36 en binaire signé sur un octet...

Le binaire signé ne permet pas de réaliser les additions habituelles

---

Exercice 2

On suppose toujours nos entiers encodés en binaire signé sur un octet.

1. Donner la représentation binaire signé de 12, de -100 et de -88.
2. Réaliser l'addition binaire bit à bit 12 + (-100).
3. Comparer avec le résultat obtenu.

Avec le binaire signé, **on ne peut plus réaliser d'opération naturelle sur les entiers**. On a maintenant deux objectifs :

1. Représenter les entiers relatifs,
2. Conserver le même algorithme pour l'addition

Le complément à deux

On doit encore travailler avec **une taille fixe**. On choisit à nouveau 8 bits par simplicité.

Comment obtenir la représentation, en complément à 2 sur 8 bits, d'un entier ?

Entiers positifs

1. Coder l'entier en binaire comme d'habitude,
2. Compléter l'octet avec des 0 devant.

Entiers négatifs

1. Coder la valeur absolue du nombre en binaire,
2. Compléter l'octet avec des 0 devant,
3. Échanger tous les bits ($1 \leftrightarrow 0$),
4. Ajouter 1.

Signe du complément à deux

- Si le bit de poids fort est 0, le nombre est positif
- Si le bit de poids fort est 1, le nombre est négatif

Exemple 1 : 27

```
1. coder l'entier en binaire comme d'habitude,
    27 = 0b11011
2. compléter l'octet avec des 0 devant.
    27 = 0b 0001 1011
```

Le complément à 2 sur un octet de 27 est 0001 1011

Exemple 2 : -9

```
1. coder la valeur absolue du nombre :
    9 = 0b1001
2. compléter l'octet :
    0000 1001
3. échanger tous les bits :
    1111 0110
4. ajouter 1 :
    1111 0111
```

Le complément à 2 sur un octet de -9 est 1111 0111

Complément à 2, méthode rapide

La méthode précédente se code facilement, elle est plus pénible à la main.

La méthode rapide : complément à 2 sur un octet de n :

Si l'entier est négatif :

1. donner la représentation binaire de la valeur absolue
2. Compléter à gauche jusqu'à avoir la taille voulue
3. de droite à gauche,**conserver tous les bits jusqu'au premier 1 inclus** 4.**changer tous les bits à gauche**

Exemple $n = -108$

1. Valeur absolue :

   $|n| = 108 = 64 + 32 + 8 + 4$

   Représentation binaire :

   $|n| = 108 = 110\ 1100$

2. Compléter :

   $|n| = 108 = 0110\ 1100$

3. Conserver les 0 à droite jusqu'au premier 1 inclus, changer tous les bits à gauche.

   $n = -108 = 10010\ \underline{100}$

La représentation en complément à 2 sur un octet de -108 est 1001 0100.

Vérifions : $27 + (-9)$

```
  0001 1011
+ 1111 0111
-----------
= 0001 0010
```

Remarque : la dernière retenue (tout à gauche) disparait.

On a bien $0b\ 0001\ 0010 = 18$

Exercice 3

1. Donner les compléments de à 2 sur un octet de 12, -100 et de -88.
2. Vérifier l'addition binaire $12 + (-100) = -88$

Soustraction

On dispose d'une représentation des entiers *négatifs* qui permette de conserver les circuits électroniques de l'addition.

Pour la soustraction $12 - 100$ on transforme ça en addition : $12 + (-100) = -88$

$a - b = a + (-b)$

Et voilà, pas besoin d'inventer une circuiterie pour la soustraction.

Exercice 4

1. Déterminer la représentation en complément à 2 sur un octet de $30, -50$ et -20.
2. En déduire le résultat en complément à 2 sur un octet de $30 - 50$.

Du complément à deux au décimal

Si l'entier est positif (son premier bit est 0)

On fait comme d'habitude.

Exemple : $0b\ 0001\ 1011$

$1 \times 1 + 1 \times 2 + 1 \times 8 + 1 \times 16 = 27$

Si l'entier est négatif (son premier bit est 1)

1. Échanger tous les bits $0 \leftrightarrow 1$,
2. Ajouter 1,
3. Convertir en décimal normalement,
4. Changer le signe.

Exemple : $0b$ 1111 0111**, en complément à 2 sur 8 bits,**

```
1. On échange tous les bits,
    0b 0000 1000
2. On ajoute 1,
    0b 0000 1001
3. On converti en binaire comme d'habitude,
    0b 0000 1001 = 1 * 1 + 1 * 8 = 9
4. On change le signe.
    0b 1111 0111 = -9
```

Du complément à 2 au le décimal, méthode rapide.

Si le nombre est positif, comme d'habitude.

Si le nombre est négatif :

1. De droite à gauche, **conserver tous jusqu'au premier 1 inclus,**
2. Échanger tous les bits **à gauche** de ce 1,
3. Convertir en décimal normalement,
4. Changer le signe,

Exemple : $0b$ 1010 1000**, en complément à 2 sur 8 bits**

1. De droite à gauche, conserver tout jusqu'au premier 1 inclus,
   1010 et 1000 qui sera conservé
2. Échanger tous les bits à gauche de ce 1,
   0101 et 1000
3. Convertir en décimal normalement,
   $0b$ 0101 1000 $= 8 + 16 + 64 = 88$
4. Changer le signe :
   en complément à 2 sur un octet $0b$ 1010 1000 $= -88$

Exercice 5

Donner les notations décimales des compléments à deux sur un octet suivants :

1. $a = 0b$ 1111 1111
2. $b = 0b$ 1000 0000
3. $c = 0b$ 0111 1111
4. $d = 0b$ 1010 0011
5. Calculer $a + b$ et $c + d$ en binaire et vérifier le résultat.

Propriétés

Table de valeurs

La table de valeur est remarquable, en particulier les "petits" nombres négatifs sont remplis de 1.

```
 bit
 de
signe
    0 1 1 1 1 1 1 1 =  127
    0       ...       =  ...
```

```
0 0 0 0 0 0 1 0 =    2
0 0 0 0 0 0 0 1 =    1
0 0 0 0 0 0 0 0 =    0
1 1 1 1 1 1 1 1 =   -1
1 1 1 1 1 1 1 0 =   -2
1       ...       =  ...
1 0 0 0 0 0 0 1 = -127
1 0 0 0 0 0 0 0 = -128
```

Combien d'entiers relatifs sur un octet ?

Sur 8 bits en complément à deux, on peut encoder de -128 à 127, soit 256 valeurs

Combien d'entiers relatifs sur avec `n` bits ?

Sur n bits on peut encoder de -2^{n-1} à $2^{n-1}-1$ soit 2^n valeurs

Exercice 6

1. Combien d'entiers positifs peut-on encoder en binaire sur 32 bits ?
2. Déterminer les valeurs minimale et maximale qu'on peut encoder en complément à 2 sur 32 bits.

Complément à 2 : résumé

- Le complément à 2 sur n bits permet de représenter des entiers positifs et négatifs,
- Le signe du nombre est donné par le premier bit,
- L'addition usuelle fonctionne aussi avec les entiers *négatifs*.
- Cette méthode permet d'encoder sur n bits les entiers de -2^{n-1} à $2^{n-1}-1$
- La méthode rapide pour encoder un entier en complément à 2 sur un octet :
  - s'il est positif, on l'écrit en binaire et on complète l'octet avec des 0 à gauche,
  - s'il est négatif,
    - on écrit sa valeur absolue en binaire qu'on complète à gauche,
    - on conserve tous les 0 à droite jusqu'au premier 1 inclus,
    - on inverse tous les bits à gauche de ce premier 1.
- Pour décoder on fait le contraire

Python et le complément à 2.

Les opérations précédentes imposent de choisir une **taille maximale pour les entiers**

Dans Python les entiers ont une **taille arbitraire**, il ne peut afficher nativement le complément à deux.

```
>>> bin(12)
'0b1100'
>>> bin(-12)
'-0b1100'
```

Pour obtenir le complément à 2, il faut le programmer.

Dans de nombreux langages, on distingue les entiers positifs des négatifs et différentes tailles sont proposées.

Compléments hors programme

Python et le complément à 2, si vous insistez

Voici une approche pour obtenir la représentation en complément à 2 :

Méthode initiale

def flip_bits_and_add_one(a: int, size: int) -> int: “““ Démarche du complément à 2 sur size bits :

```
1. Echanger les bits d'un entier
2. Ajouter 1
3. Enlever les bits trop à gauche

"""
# on crée un masque rempli de 1 : "11111111...1111"
mask = (1 << size) - 1
# 1. échanger tous les bits
a ^= mask
# 2. ajouter 1
a += 1
# 3. retirer les bits trop à gauche
a &= mask

return a
```

def comp2(n: int, size: int) -> str: “““Renvoie le complément à 2 sur `size` bits de `n`””” if n >= 0: return bin(n) a = flip_bits_and_add_one(-n, size) return bin(a)

def read_comp2(bits: str, size: int) -> int: “““Evalue la valeur d’un entier en complément à 2 sur size bits””” # retirer “0b” au début bits = bits.replace(“0b”, ““)

```
# s'il est positif, rien à faire
if len(bits) < size or bits[0] == 0:
    return int(bits, 2)

# convertir en entier, comme s'il était positif
a = int(bits, 2)
a = flip_bits_and_add_one(a, size)
return -a
```

def table(): ““” Dessine une table de valeurs du complément à 2 sur 8 bits. Profite de l’occasion pour tester toutes les valeurs. ““” for n in range(127, -129, -1): assert n == read_comp2(comp2(n, 8), 8) print(f”{n}ĉomp2(n, 8)“)

table()

Code

Types entiers dans d’autres langages

Python n’aide pas à comprendre l’intérêt du complément à deux : les entiers sont d’un seul type, indépendemment de leur signe ou de leur taille.

Lorsqu’on a besoin de vitesse, il est important de ne pas gaspiller la mémoire. Ainsi, dans de nombreux langages il existe plusieurs types pour les entiers.

Sans entrer dans les détails, voici quelques types de base du langage C :

Type	Description	Min	Max	Taille
`char`	un octet	0	255	8 bits
`signed char`	un octet en complément à 2	-127	128	8 bits
`short`	entier en complément à 2 sur 2 octets	-32768	32767	16 bits
`unsigned short`	entier naturel sur 2 octets	0	65535	16 bits
`int`	entier en complément à 2 sur 4 octets	-2147483648	2147483647	32 bits
`unsigned int`	entier naturel sur 4 octets	0	4294967295	32 bits

Et il y en a bien d'autres... L'idée générale est la souplesse, on peut employer exactement le type d'entier dont on a besoin tout en conservant la vitesse maximale.

Au passage, 32 bits dans une adresse IPv4, 32 bits dans un `unsigned int`... Une adresse IPv4 peut être représentée par un seul `unsigned int` en C.