Bits à bits - cours

qkzk

On peut appliquer les opérateurs booléens sur les différents bits d'un nombre ou de deux nombres.

Ce sont des "*opérations bit à bit*".

Opérateurs en taille quelconque

On applique, bit par bit nos opérateurs usuels :

NOT bit à bit

Chaque bit est inversé.

Sur 4 bits, NOT 7 = 8

```
NOT 0111
  = 1000
```

Pour cet opérateur, il est nécessaire de connaître la *capacité* employée, ici 4 bits.

ET bit à bit

On fait un `and` sur chaque bit des nombres en respectant les positions.

Par exemple : 5 AND 3 = 1.

$5 = 0b101$, $3 = 0b11$ donc :

```
   101
AND 011
  = 001
```

OU bit à bit

5 OR 3 = 7 :

```
   101
OR 011
 = 111
```

XOR bit à bit

5 XOR 3 = 6 :

```
   101
XOR 011
  = 110
```

Python

Opérateurs bits à bits en Python

```
~x          # NON bit à bit (tilde)
x & y       # ET  bit à bit (ampersand)
x | y       # OU  bit à bit (tuyau)
x ^ y       # XOR bit à bit (accent circonflexe)
```

Par exemple, pour le XOR bit à bit

```
0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0

60       = 0b111100
30       = 0b011110
60 ^ 30  = 0b100010 # XOR bit à bit en colonne
0b100010 = 34
```

```
>>> 60 ^ 30
34
>>> bin(60 ^ 30)
'0b100010'
```

Décalages à gauche et à droite

Il existe aussi deux opérations courantes lorsqu'on manipule des bits :

```
x << y      # x décalé de y bits à gauche
x >> y      # x décalé de y bits à droite
```

Par exemple

- `111 << 2 = 11100` : décalage de 2 bits vers la gauche.
- `101110 >> 3 = 101` : décalage de 3 bits vers la droite (les bits trop à droite sont supprimés).

Ces opérations correspondent à des produits ou des divisions par 2 :

- décaler d'un bit vers la gauche c'est multiplier par 2,
- décaler d'un bit vers la droite c'est diviser (entièrement) par 2.

```
>>> (x << y) == x * 2 ** y
True
>>> (x >> y) == x // (2 ** y)
True
```

Un schéma electronique représentant un décalage à gauche

Exercices

Exercice 1

1. Calculez la représentation binaire de 29.
2. Calculez la représentation binaire de 15.
3. Démontrer que le ET bit à bit entre 29 et 15 vaut 13.

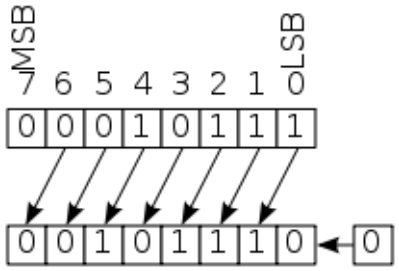

Figure 1: Décalage à gauche

Les masques de sous-réseau

Très largement inspiré de cet article de Wikipedia.

Les adresses IP de version 4, IPv4, sont codés sur **32 bits**.

En notation décimale : 4 nombres compris entre 0 et 255, séparés par des points.

En fait, ce sont 4 *octets* généralement notés en décimal.

Par exemple : 192.168.100.2.

Elles sont ont composées de deux parties : le *sous-réseau* et l'*hôte*. Ils utilisent la même représentation.

On utilise des masques constitués (sous leur forme binaire) d'une suite de 1 suivis d'une suite de 0, il y a donc 32 masques réseau possibles.

Exemple de masque

Un exemple possible est le masque `255.255.255.0`.

Pour obtenir l'adresse du sous-réseau on applique l'opérateur ET entre les notations binaires de l'adresse IP et du masque de sous-réseau.

L'adresse de l'hôte à l'intérieur du sous-réseau est quant à elle obtenue en appliquant l'opérateur ET entre l'adresse IPv4 et la négation (NON) du masque.

Exercice 2 - Masques de sous-réseau

1. Calculez le code binaire correspondant à l'adresse `192.168.100.2`
2. Calculez le code binaire correspondant au masque `255.255.255.0`.
3. Vérifier que l'adresse binaire du sous-réseau est `192.168.100.0`
4. Vérifier que l'adresse de l'hôte est `0.0.0.2`

Conclusion : si le masque n'est constitué que de 255 et de 0 c'est facile.
