Spécialité NSI

LYCÉE de l'ÉLORN
le choix de votre réussite

Fiche de cours
Opérations sur les nombres binaires
Complément à 2

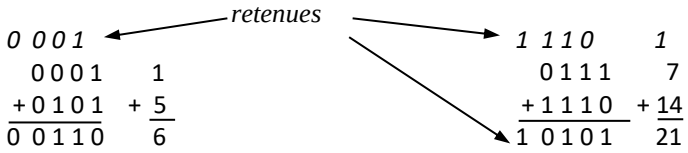
I Opérations arithmétiques

1.1 Instructions d'additions

Les opérations arithmétiques se font de la même façon en binaire qu'en base décimale. Lorsque l'on dépasse la valeur maximale au niveau du rang *n* (résultat > 1) on génère une retenue au bit de rang *n+1*. De fait, une addition de 2 mots de n bits donne un résultat sur n+1 bits.

Ainsi : 0+0=00 ; 0+1 = 01 ; 1+0=01 ; 1+1=10

Voyons cela sur une addition de deux mots de 4 bits :



1.2 Instructions de soustractions

Même principe que précédemment en admettant les bases suivantes : 0–0=00 ; 1-0=01 ; 1-1=00 ; 0-1=11 ; 0-1-1=10

Ce qui donne sur 4 bits :

☞ *Faites l'opération 20-14 en binaire.*

0 1 1 1 *1*
1 1 1 0 14
- 0 1 1 1 - 7
0 0 1 1 1 07

2 Complément à 1 et à 2

2 .1 Le complément à 1 (ou complément restreint)

Le complément à un d'un nombre binaire se forme en soustrayant de 1 chaque bit de ce nombre. Cela revient à inverser chacun des bits. Exemple : Si A = 00010101 alors $\overline{A}^1$ = 11101010

2.2 Le complément à 2 (ou complément vrai)

Soit A un nombre binaire, le complément à deux de A est $\overline{A}^2 = \overline{A}^1 + 1$

Exemple : Si A = 00001001 → $\overline{A}^1$ = 11110110 → $\overline{A}^2$ = 11110111

On présente ainsi une notion de **nombre négatif** en binaire puisque quel que soit A écrit sur n bits :

$A + \overline{A}^2 = 0$ d'où $\overline{\mathbf{A}}^{\mathbf{2}} = \mathbf{-A}$

2.3 Les nombres négatifs en base 2

Par convention, le bit de poids fort (MSB) d'un nombre défini son signe (+ ou -). Un nombre négatif s'exprime en complément à 2 en ajoutant un bit de poids fort pour intégrer le signe.

Soit un nombre de 8 bits : **$A = a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$**

Bit de signe
A > 0 $a_7 = 0$
A < 0 $a_7 = 1$

Signification
A > 0 : binaire pur
A < 0 : complément à 2

Exemple A = - 3 → |A| = 3 ≡ **0**0000011 d'où $\overline{A^1}$ = 11111100 puis $\overline{A^2}$ = **1**1111101

0 : Signe **+**

Valeur absolue

1 : Signe -

Complément à 2 de +3 ≡ -3

Tableau des nombres positifs et négatifs sur un format de 4 bits :

0	0 000	*-1*	1 111
1	0 001	*-2*	1 110
2	0 010	*-3*	1 101
3	0 011	*-4*	1 100
4	0 100	*-5*	1 011
5	0 101	*-6*	1 010
6	0 110	*-7*	1 001
7	0 111		

A consulter : Connaissez-vous le Bug de l'an 2038 ? → https://fr.wikipedia.org/wiki/Bug_de_l'an_2038

Application :

Donner sur 8 bits la réprésentation binaire des nombres suivants : 1, -1 , +125, -125, +110, -110

……
……

Soit A=125 et B=110. Réaliser en binaire sur 8 bits signés les opérations A+B; A-B; B-A; -A-B puis valider les résultats trouvés :

……
……
……
……
……
……
……
……

3 Instructions logiques bit à bit

Les instructions logiques permettent de faire des opérations bit-à-bit sur des nombres binaires. C'est-à-dire en considérant chacun des bits de même poids indépendamment des autres, sans se soucier de la retenue.

- L'instruction **AND (ET)** (& en python) : Cette instruction met le bit du résultat à 1 si les deux bits de même poids des opérandes sont à 1, sinon il le met à zéro.
- L'instruction **OR (OU)** (| en python) : Met le bit du résultat à 0 si les deux bits de même poids des opérandes sont à 0, sinon il le met à un.
- La fonction **XOR (OU exclusif)**, (^ en python) : Met le bit du résultat à 1 si les deux bits de même poids sont différents.
- La fonction **NOT (NON)** (~ en python) inverse chacun des bits d'un nombre binaire.

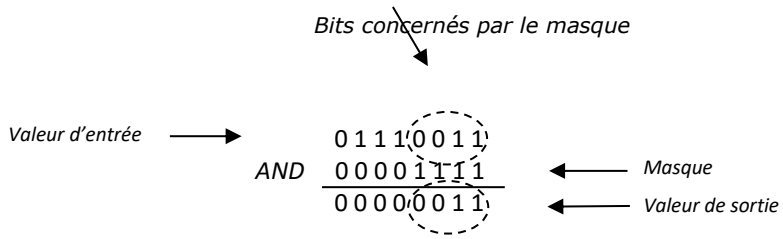
Exemples :

```
    0 1 1 1                0 1 1 1
AND 1 1 1 0            OR  0 1 1 0
    -------                -------
    0 1 1 0                0 1 1 1
```

NOT (1 1 0 0) = 0 0 1 1

☞Justifier les opérations suivantes **:** 12 | 3 = 15 12 & 8 = 8 12^7 = 11 ~2 = -3 ~12 = -13

Application du Masquage :
Il est possible de masquer, c'est-à-dire mettre à zéro tous les bits qui ne nous intéressent pas dans un nombre. Dans l'exemple ci-dessous, seuls les 4 derniers bits de l'octet 01110011 seront récupérés dans le résultat de l'opération 01110011 AND 00001111

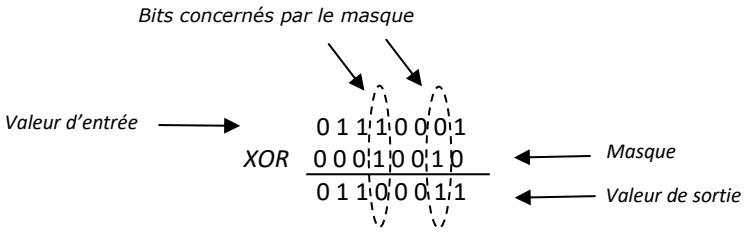


En python :

```
115&15
3

bin (0b01110011 ^ 0b00001111)
'0b11'
```

Ici l'on procède à l'inversion des bits 2 et 5 de l'octet 01110011 avec l'opération 01110011 XOR 00010010



En python :

```
113^22
103

bin (0b01110001 ^ 0b00010010)
'0b1100011'
```

Exemple d'application sur l'adressage IP :
On utilise le principe du masquage avec l'adressage IP. A partir d'une adresse IP et d'un masque de sous-réseau on peut générer l'adresse commune dite « adresse réseau » des systèmes intégrés à un même réseau.

☞ Compléter ce tableau :

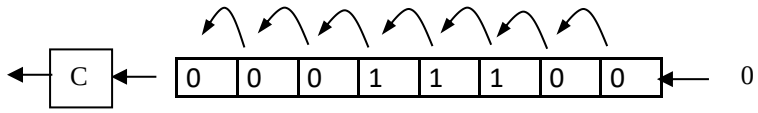
IP décimale	192	168	155	24
IP binaire				
Masque décimal	255	255	240	0
Masque binaire				
Adresse réseau (netid) **= IP and Masque**				

4 Décalage et rotation

Ces instructions permettent de décaler d'un côté ou de l'autre les bits contenus dans une cellule mémoire (plus précisément un des registres d'un microprocesseur). Cette opération a plusieurs applications très intéressantes :

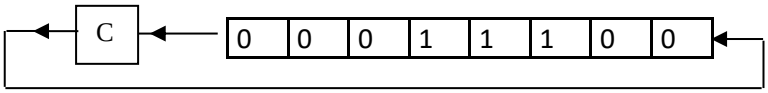
- permettre de lire un à un les bits d'un registre (cellule mémoire)
- permettre une multiplication par 2^n (en effet le fait de décaler un nombre binaire d'un chiffre à gauche le multiplie par 2, ainsi en effectuant cette opération n fois on obtient une multiplication par 2^n)
  exemple:
  00010 (2 en décimale)
  00100 (on décale à gauche on obtient 4)
  01000 (on décale à gauche on obtient 8)
- permettre une division par 2^n (comme précédemment mais en effectuant une rotation sur la droite)

Une opération de décalage déplace chacun des bits d'un nombre binaire sur la gauche (ou la droite), mais ceux-ci sortent vers le bit de retenue C lorsqu'ils arrivent au bit de poids fort (ou de poids faible) puis sont définitivement perdus.
exemple:



00011100
00111000 (on décale d'un bit à gauche)
01110000 (on décale d'un bit à gauche)
11100000 (on décale d'un bit à gauche)
11000000 (on décale d'un bit à gauche)

Une opération de rotation agit comme une opération de décalage à la différence près que les bits qui sortent d'un côté rentrent de l'autre.
exemple:



00011100
00111000 (on effectue une rotation d'un bit à gauche, C est initialement à 0)
01110000 (on effectue une rotation d'un bit à gauche, C est à 0)
11100000 (on effectue une rotation d'un bit à gauche, C est à 0)
11000000 (on effectue une rotation d'un bit à gauche, C est à 1)
10000001 (on effectue une rotation d'un bit à gauche, C est à 1)

En python :

```
bin(0b0011 << 2)
'0b1100'

3 << 2
12

8 >> 2
2
```