Nous allons voir quelques opérateurs de calcul binaire qui sont utilisés dans les programmes, par exemple pour calculer des conditions.

1. Opérateurs logiques

1.1. Explications

Les microprocesseurs permettent de faire de calculs logiques binaires. Il y a quatre opérateurs à connaître.

- `NON`(N) inverse tous les bits de N. Ainsi, `NON`$(10110)_2 = (01001)_2$
- A `ET` B compare les bits des mêmes rangs dans les deux nombres : a_0 avec b_0, a_1 avec b_1, etc. Si $a_i = b_i = 1$ alors le bit correspondant du résultat est mis à 1, sinon il est mis à 0.
- A `OU` B est similaire. Si $a_i = b_i = 0$, alors le bit correspondant du résultat est mis à 0, sinon il est mis à 1.
- A `OUX` B (*ou exclusif*) est également similaire. Si $a_i \neq b_i$, alors le bit correspondant du résultat est mis à 1, sinon il est mis à 0.

Exemple :

	NON	ET	OU	OUX
A	1011	1010	1010	1010
B		1001	1001	1001
	0100	1000	1011	0011

Leur utilité est un peu moins évidente que les additions et soustractions, mais :

- Les opérateurs logiques permettent par exemple de déterminer si un nombre est négatif : le bit de poids fort est à 1. Le test est facile à faire. Il suffit d'écrire N `ET` $(100...00)_2$. Si le résultat est non-nul, alors N est négatif.
- On peut aussi déterminer si un nombre est pair : le bit de poids faible est à 0. Le test est simple. Si N `ET` $(00...0001)_2$ est nul, alors N est pair.
- L'opérateur CHS s'écrit simplement `NON`$(N) + 1$.
- Plus étonnant, l'addition elle-même s'écrit à l'aide de ces opérateurs logiques. En effet, quand on additionne deux bits a_i et b_i, la somme $a_i + b_i = a_i$ `OUX` b_i et la retenue est a_i `ET` b_i. Dans une unité arithmétique et logique, les additions sont réalisées par des `OUX` et des `ET`.
- L'opérateur A `OUX` B est équivalent à (`NON`(A) `ET` B) `OU` (A `ET` `NON`(B)).
- On peut montrer (algèbre de Boole) qu'il suffit d'un seul opérateur, appelé `NON-ET` qui vaut `NON`(A `ET` B) pour reconstruire tous les autres.

1.2. Exercices

i. Calculer `NON`(0101) `ET` 0110
ii. Calculer 1011 `OUX` (1010 `ET` 0111)
iii. Calculer `NON`(0101) `OU` `NON`(1101)

1.3. Remarques

Ces opérateurs sont définis dans les langages de haut niveau, C, Java, etc. : `NON`(N) s'écrit `~`N, A `ET` B s'écrit A `&` B, A `OU` B s'écrit A `|` B, et A `OUX` B s'écrit A `^` B. Attention à ne pas les confondre avec `!`, `&&` et `||` qui travaillent sur des booléens isolés.

2. Décalages et rotations

2.1. Explications

Voici une autre catégorie d'opérateurs binaires, les décalages et rotations. Il y a trois sortes AS*, LS* et RO*, chacune ayant deux directions, vers la droite (*=R) et vers la gauche (*=L). Voici des schémas.

Figure 1: Rotations

Les rotations servent dans certains algorithmes de cryptage des données (md5, sha-1, sha-256...).

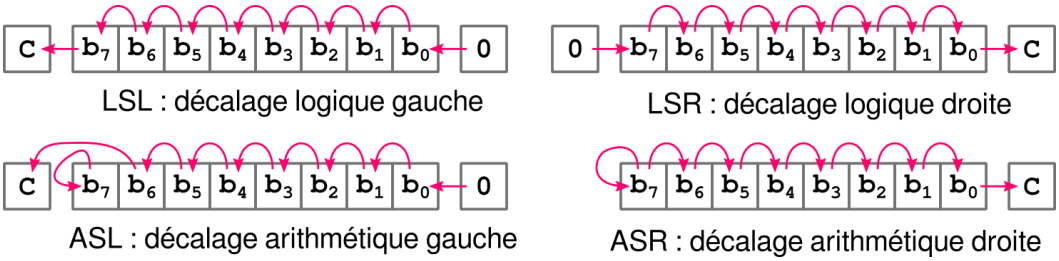

Figure 2: Décalages

Les décalages vers la gauche servent à faire des multiplications et les décalages à droite font des divisions. La différence entre les logiques et les arithmétiques est sur la convention des entiers : naturels pour les décalages logiques et signés en $C2_8$ pour les décalages arithmétiques. Vous remarquez que le bit de poids fort qui représente le signe est conservé dans les décalages arithmétiques. Également, le bit de poids fort ou poids faible est envoyé dans l'indicateur `C` du processeur. Cela permet de chaîner des rotations sur plusieurs octets, ou de détecter des erreurs de calcul.

En général, les opérateurs de décalage sont définis dans les langages de haut niveau. Ils s'écrivent `<<` et `>>`, mais la convention de signe n'est pas normalisée. Il n'y a pas d'opérateur pour les rotations, on doit les calculer avec des `ET`, des `OU` et des décalages.

2.2. Exercices

i. Calculer la rotation à droite de $(10011101)_2$.
ii. Calculer la rotation à gauche de $(10011101)_2$.

iii. Écrire 50 en base 2 sur 1 octet, puis le décaler logiquement à droite, enfin convertir le résultat en base 10.
iv. Écrire 17 en base 2 sur 1 octet, puis le décaler logiquement à gauche, enfin convertir le résultat en base 10.
v. Écrire -24 en $C2^8$, puis le décaler arithmétiquement à droite, enfin convertir le résultat en base 10 signée.
vi. Écrire -13 en $C2^8$, puis le décaler arithmétiquement à gauche, enfin convertir le résultat en base 10 signée.

3. Soustractions

On revient sur la soustraction. Elle est un peu plus compliquée à calculer manuellement que l'addition.

3.1. Soustractions directes

Une soustraction en base 2 est très similaire à celle en base 10. On traite chiffre par chiffre en commençant par les unités. Quand le chiffre du haut est trop petit, on lui ajoute $(10)_2$ qu'on note avec un petit 1 devant le chiffre du haut, et on ajoute +1 au chiffre du bas suivant. Voici un exemple, les explications suivent :

$$\begin{array}{cccccc} & \mathbf{1} & {}_1\mathbf{1} & {}_1\mathbf{0} & \mathbf{1} & \mathbf{1} \\ - & {}_{0+1} & \mathbf{1}_{+1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \hline & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{array}$$

On veut calculer $(11011)_2 - (1101)_2$.

- On commence par les unités $1 - 1 = 0$.
- On passe à la colonne 2 : $1 - 0 = 1$.
- Colonne 3 : le chiffre du haut 0 est plus petit que celui du bas, 1. Donc on ajoute $(10)_2$ au chiffre du haut : $((10)_2 + 0) - 1 = 1$ et il faut incrémenter le prochain chiffre du bas, ce qui se note avec un « +1 » à côté.
- Colonne 4 : on doit calculer $1 - (1 + 1)$. Là aussi, il faut ajouter $(10)_2$ au chiffre du haut : $((10)_2 + 1) - (1 + 1) = 1$ (on peut faire les calculs en base 10, après conversions). Ne pas oublier la retenue sur le chiffre du bas suivant.
- Colonne 5 : en haut, on a 1 et en bas, il y a $0 + 1$. Donc $1 - (0 + 1) = 0$.

3.1.1. Algorithme

On dispose de deux entiers A et B écrits sur n bits :

$$A = a_{n-1}.2^{n-1} + a_{n-2}.2^{n-2} + ... + a_2.2^2 + a_1.2^1 + a_0.2^0$$
$$B = b_{n-1}.2^{n-1} + b_{n-2}.2^{n-2} + ... + b_2.2^2 + b_1.2^1 + b_0.2^0$$

Ces deux nombres sont codés en convention $C2^n$, ou non. Le résultat sera comme le codage.

1. Si $A < B$, alors

- En convention $C2^n$ (entiers relatifs), alors remplacer A par $2^n + A$. Ça revient à placer un 1 en tête de A. Ce 1 ne compte pas dans le résultat final car il est assimilé à zéro et le résultat sera tronqué à n bits.
- Hors convention $C2^n$ (entiers naturels uniquement), la soustraction n'est pas possible.

2. Il faut écrire A et B l'un au dessus de l'autre en alignant les unités à droite. Attention, l'ordre A, B est important car la soustraction n'est pas commutative.
3. On commence par traiter les unités (chiffres de droite) a_0 et b_0, puis on traite les chiffres suivants en allant vers la gauche.
4. Soient a_i et b_i les deux chiffres à traiter. Leur soustraction donne un chiffre appelé c_i.
   - Si $a_i \geq b_i$, alors $c_i = a_i - b_i$. Voici un exemple :

     $$\begin{array}{rccc} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ - & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \hline & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{array}$$

   - Sinon, $c_i = (10)_2 + a_i - b_i$, et remplacer b_{i+1} par $b_{i+1} + 1$. C'est ce qu'on appelle une retenue.
     Exemples :

     $$\begin{array}{rcc} & \mathbf{1} & {}_1\mathbf{0} \\ - & \mathbf{0}_{+1} & \mathbf{1} \\ \hline & \mathbf{0} & \mathbf{1} \end{array} \qquad \begin{array}{rccc} & \mathbf{1} & {}_1\mathbf{1} & {}_1\mathbf{0} \\ - & {}_{0+1} & \mathbf{1}_{+1} & \mathbf{1} \\ \hline & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{array}$$

3.1.2. Exercices

i. Calculer $(101011)_2 - (11101)_2$
ii. Calculer $(100000)_2 - (10101)_2$
iii. Calculer $(101101)_2 - (10110)_2$

3.2. Soustractions en tant qu'additions avec l'opposé

La convention $C2^n$ permet aussi de calculer une soustraction $A - B$ sous la forme $A + CHS_n(B)$.

a. On veut calculer $(101011)_2 - (11101)_2$ sur **6 bits**
   i. Mettre les deux nombres sur 6 bits : $(101011)_2$ et $(011101)_2$
   ii. Calculer $CHS_6((011101)_2)$, c'est l'opposé du second nombre
   iii. Additionner le premier nombre avec l'opposé du second
   iv. Tronquer la somme à 6 bits et comparer avec le résultat obtenu dans la partie précédente.
b. On veut calculer $(100000)_2 - (10101)_2$
   i. Mettre les deux nombres sur 6 bits
   ii. Calculer l'opposé du 2e nombre
   iii. Additionner le premier nombre avec l'opposé du second
   iv. Tronquer la somme à 6 bits et comparer avec le résultat obtenu dans la partie précédente.
c. Calculez $-5 - 2$ sur 4 bits avec cette méthode
   i. Écrire les deux nombres à soustraire sur 4 bits.
   ii. Appliquer CHS_4 sur le second nombre
   iii. Faire l'addition
   iv. Tronquer la somme et vérifier le résultat.