L'informatique, comment ça marche ?

Juste un aperçu…

Nos machines informatiques sont des automates électroniques programmés. Elles fonctionnent à l'aide de **commandes** (ou *ordres*) et de **données** écrites en **langage binaire**.
Pourquoi binaire ? Après diverses expériences, on a retenu ce langage parce qu'il traduit bien, avec ses deux seuls symboles, la dualité de nombreux dispositifs électroniques simples *: le courant passe ou ne passe pas, la porte est ouverte ou fermée, la particule magnétique est aimantée nord ou sud ...* C'est parce que on se limite à deux états que ces dispositifs sont extrêmement fiables.

Les symboles du langage binaire sont notés **0** et **1** (on peut aussi les appeler *oui* et *non*, *vrai* ou *faux*...). Avec ce langage simple, on peut exprimer tous les nombres possibles, toutes les lettres connues, toutes les opérations élémentaires et tous les ordres nécessaires.

Numération binaire

Dans la **numération décimale** usuelle, on utilise une notation de position. Par exemple,

l'écriture 1453 signifie qu'on parle du nombre $1\times1000 + 4\times100 + 5\times10 + 3$

Les matheux auront compris qu'on peut aussi l'écrire : $1\times10^3 + 4\times10^2 + 5\times10^1 + 3\times10^0$ (avec les puissances de 10).
La numération binaire procède de la même façon. Exemple

l'écriture 10110101101 ou bien, en décomposant en tranches de 4 chiffres : 101 1010 1101

signifie qu'il s'agit du nombre

$1\times1024 + 0\times512 + 1\times256 + 1\times128 + 0\times64 + 1\times32 + 0\times16 + 1\times8 + 1\times4 + 0\times2 + 1\times1$

ou bien $1\times2^{10} + 0\times2^9 + 1\times2^8 + 1\times2^7 + 0\times2^6 + 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

(en faisant les additions, on voit que ce nombre est égal à 1453_{10} (c-à-d. 1453 exprimé en base 10).

Numération hexadécimale

L'écriture binaire des nombres est sans doute proche des signaux machine, mais elle donne lieu à des écritures longues, difficiles à retenir. On a décidé de grouper par 4 les symboles binaires (les **bits**), de calculer la valeur de chacun de ces groupes. Ces groupes de 4 bits (qu'on peut appeler des *quartets*) ont une valeur au plus égale à 15 : pour les représenter, on utilisera les 16 symboles **0,1,2,3,.. 8, 9, A, B, C, D, E**. Le nombre précédent s'écrit donc alors 5AD. C'est bien plus facile à écrire que du binaire.
Ce nombre est en réalité exprimé en base 16 :

il est égal à $5\times16^2 + A\times16^1 + D\times16^0 = 5\times256 + 10\times16 + 13\times1 = 1453$

On dit que c'est un nombre *hexadécimal*. Il est parfois suivi du symbole h ou H (pour qu'il ne soit pas confondu avec un nombre décimal).

Codage des caractères

C'est le télex qui a le premier obligé à coder les caractères par des **nombres**. Actuellement, l'informatique utilise le **codage ASCII** pour 128 caractères simples : par exemple, les lettres A, B, C... sont codées 65, 66, 67 ..., a, b, c, ... 97, 98, 99, ... Ces codes suffisent pour les textes en anglais. Une *extension* à la table ASCII utilisent les codes 128 à 256 pour représenter des symboles moins courants, comme le ç et les lettres accentuées. Avec cette convention, si la donnée étudiée est censée représenter du texte, un octet (ensemble de 8 bits) représente un caractère. Un octet comprend deux quartets et sa valeur s'exprime donc par deux chiffres hexadécimaux (valeur maximum FF ou 255). On appelle kilo-octet, méga-octet, giga-octet, ... des ensembles de 1024, 1024^2 (≈ 1,048.. million), 1024^3 (≈ 1,07.. milliard) de caractères.

Codage des ordres machine

C'est une table, propre à chaque fabricant de puces électroniques, qui fait correspondre un ordre, une commande à un nombre. La variété d'ordres utilisés n'est pas très élevée : de 50 à 150 (elle occupe un octet).

Ainsi, un octet peut représenter un caractère, un ordre ou un chiffre (partie d'un nombre) : la machine le sait grâce à l'ordre qu'elle est en train d'exécuter.

Opérations logiques

Toute la puissance de nos machines repose sur les opérations logiques, lesquelles sont proches du raisonnement humain dans la discrimination entre le vrai et le faux. Dans ce but, on utilise 3 opérations de base :
- **la réunion** : si *c* = *a* **OU** *b*, *c* est vrai si *a* est vrai OU si *b* est vrai ; sinon *c* est faux ;
- **l'intersection** : si *c* = *a* **ET** *b*, *c* est vrai si *a* est vrai ET si *b* est vrai ; sinon *c* est faux ;
- l'**inversion** (ou *complément*) : on l'écrit *c* = *a* et alors *c* est vrai si *a* est faux et *c* est faux si *a* est vrai.

L'informatique utilise également une opération dont nous n'avons guère l'habitude, le **OU exclusif**, qu'on peut écrire **XOU** ; il ressemble à la réunion, sauf que *c* est faux si *a* et *b* sont tous deux vrais.

On résume ces opérations par des tableaux appelés **tables de vérité**.

OU

a,b	0	1
0	0	1
1	1	1

ET

a,b	0	1
0	0	0
1	0	1

INV

a	0	1
c	1	0

OU EXCLUSIF

a,b	0	1
0	0	1
1	1	0

L'algèbre de Boole développe une théorie de calcul et de raisonnement à partir de ces notions. On a pu déduire des théorèmes importants, comme ceux-ci :

- l'inverse d'une réunion est égal à l'intersection de ses inverses.
- l'inverse d'une intersection est égal à la réunion de ses inverses.
- l'inversion des variables dans le OU exclusif n'en change pas le résultat.

Les 2 premières propositions constituent le célèbre **théorème de De Morgan**. On peut l'illustrer par l'exemple ci-après, citant propositions équivalentes, mais inverses

- demain, je resterai chez moi s'il pleut OU si je suis malade
- demain, je sortirai s'il ne pleut pas ET si je vais bien.

Dans les machines et dans leurs programmes, les informaticiens utilisent constamment ces notions de logique.

Calcul arithmétique.

L'addition de nombres binaires est très simple : ce nombre est fait de chiffres (des bits) qui sont égaux à 0 ou à 1. On aligne les nombres à droite ; dans chaque colonne, le résultat ne peut être que 0 ou 1, avec une retenue nulle ou égale à 1.
On montre facilement que, dans l'addition de chaque colonne, le résultat est égal à l'opération XOU et la retenue à l'opération ET. On procède ainsi de colonne en colonne comme on a appris à le faire à l'école avec des nombres décimaux.

La soustraction se déduit facilement de l'addition si on adopte une convention astucieuse pour l'écriture des nombres négatifs : alors la soustraction se ramène à l'addition d'un négatif.

La multiplication n'est qu'une suite d'additions avec décalage (le décalage est une opération élémentaire importante dans les circuits électroniques) et la division une combinaison de multiplications et de soustractions.

On peut donc conclure que les opérateurs logiques, joints aux décalages, permettent de réaliser toutes les opérations arithmétiques élémentaires sur des nombres binaires.

Un peu d'électronique

Il n'est pas difficile de fabriquer les circuits logiques nécessaires. Il faut rappeler quelques notions d'électricité, la loi d'Ohm, par exemple.

Les circuits sont alimentés en énergie électrique par une alimentation dont la tension est U (5 volts il y a peu, maintenant 3 volts) et dont la borne négative est à la masse.

Un transistor - en informatique binaire - est comparable à un **relais** ; il possède un **circuit d'entrée** et un **circuit de sortie**. Exciter son entrée avec un courant *i* provoque la conduction de son circuit de sortie, c-à-d. le passage d'un courant *I* dans ce circuit. Le rapport *I/i* est appelé *gain* du transistor ; il est de l'ordre de 20 à 100.

La borne de sortie *S* du transistor est reliée à l'alimentation *U* par une résistance *R* (pour ne pas provoquer de court-circuit). La loi d'Ohm nous apprend que, si le transistor ne conduit pas, la tension au point *S* est *U* ; s'il conduit, la tension en *S* est voisine de 0 (parce que le transistor en conduction est quasiment un court-circuit). Si l'on symbolise le bit 1 par la tension *U* et le bit 0 par une tension proche de 0, on voit que le transistor se comporte comme un inverseur, puisque 1 à l'entrée donne 0 en sortie et vice-versa.

Construire un circuit OU n'est guère plus difficile. L'entrée du transistor est toujours protégée par une résistance de grande valeur *r* (pour que le courant d'entrée *i* soit faible).
On peut alimenter l'entrée du transistor par deux circuits (deux résistances *r*) et non plus un seul. Puisque ces résistances r_1 et r_2 sont élevées, les entrées ne réagissent pas l'une sur l'autre (elles restent indépendantes, c'est très important). Mais il suffit qu'une des 2 entrées soit portée à 1 (ou U volts) pour que le transistor conduise : 1 OU 1 donne 0. On a matérialisé un circuit OU inverse ; l'adjonction d'un transistor inverseur le transforme en un circuit OU.

Grâce au théorème de De Morgan, qui jongle avec les inversions pour passer des opérateurs OU aux opérateurs ET, il suffit d'une succession d'inverseurs et de OU inverses pour construire n'importe quel opérateur logique et, par conséquent, n'importe quel opérateur arithmétique.