Chapitre :
Représentation numérique

I. Le courant passe... ou pas

Une machine électrique fonctionne si elle a du courant, ne fonctionne pas si elle n'en a pas. Ainsi fait l'ampoule : elle s'allume avec du courant et reste éteinte sans (suffisamment de) courant. Ces deux états « courant » et « pas de courant » peuvent être notés respectivement 1 et 0. On obtient ce que l'on appelle un codage binaire. C'est sur ce codage que fonctionnent les ordinateurs, qui sont capables de manipuler des informations sous cette forme. Manipuler signifie entre autres recevoir les informations et en envoyer.
L'alphabet de l'ordinateur n'est donc composé que de deux lettres : 0 et 1.
Une telle lettre est appelée **bit**, mot qui est la contraction de **bi**nary digi**t** (chiffre binaire).
La question que l'on peut se poser est : comment la machine manipule-t-elle ce langage ?
La réponse repose sur un composant électronique très particulier : le transistor.
Un transistor, qui possède une entrée et une sortie, permet de créer en sortie le signal contraire d'un signal donné en entrée. Autrement dit il permet de transformer un 1 en 0 et un 0 en 1.

La machine ne traite généralement pas qu'un bit à la fois, mais des paquets de bits.
Un paquet de huit bits est appelé un **octet**.

Exemple Voici donc un octet (dans le langage binaire) : 10011101.

N'a-t-on pas l'impression de voir un nombre ? Avec ce principe on peut effectivement coder les entiers. Mais attention ! L'écriture est trompeuse. Ce sont des nombres écrits en binaire, en mathématiques on dit en base 2. Pour éviter l'ambiguïté on écrit $(10011101)_2$.

II. Codage binaire des nombres

Notre notation habituelle des nombres est un codage en base 10 : nous utilisons dix symboles de chiffres pour écrire les nombres : $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.
Pour coder en base 2 nous n'allons utiliser que deux symboles : 0 et 1. Pour compter, le principe est le même qu'en base 10, sauf qu'il ne faut pas oublier que l'on ne dépasse pas le chiffre 1.
Comptons en binaire : 0, 1, 10, 11, 100, 101,...

Rappelons que tout nombre (en base 10) peut être décomposé à l'aide de puissances de 10 :

Exemple $(562)_{10} = 5 \times 100 + 6 \times 10 + 2 \times 1 = 5 \times 10^2 + 6 \times 10^1 + 2 \times 10^0$.

Remarque On peut observer que :

- la puissance la plus petite (qui se trouve pour le chiffre le plus à droite) est 0 ;
- pour un nombre de 3 (resp. n) chiffres, la puissance (de dix) la plus grande est 2 (resp. $n-1$).

Il y a donc une sorte de décalage auquel il faut faire attention.

Un nombre en base 2 est lui décomposé (en base 10 !) à l'aide de puissances de 2 sur le même principe :

Exemple $(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$.
En calculant le nombre de droite (écrit en base 10 rappelons-le), on obtient : $4 + 1 = 5$.
On a donc bien : $(101)_2 = (5)_{10}$.

On vient de voir dans cet exemple comment traduire (ou coder) un nombre écrit en base 2 en nombre en base dix.

Exercice 1 Donner les valeurs des nombres suivants en base 10 :

$$(1000)_2 \quad ; \quad (11011)_2 \quad ; \quad (10011101)_2$$

Exercice 2 Si l'on ne dépasse pas 8 bits (donc un octet), combien peut-on écrire de nombres en binaire ? Quelle est la valeur en base 10 du plus grand de ces nombres ?

Réciproquement, comment coder un nombre écrit en base 10 en un nombre écrit en base 2 ?
On peut imaginer au moins deux méthodes, nous en proposons une seule ici.

Soit a un nombre entier écrit en base 10.
Soit b son code binaire, que nous supposerons composé de n bits :
$b = (b_{n-1} \dots b_2 b_1 b_0)_2$, avec les b_i valant 0 ou 1.
On appelle b_0 le **bit de poids faible** et b_{n-1} le **bit de poids fort**.
Alors a se décompose sous la forme suivante en base 10 :

$$a = b_{n-1}2^{n-1} + \cdots + b_2 2^2 + b_1 2 + b_0 \qquad (E)$$

On remarque que l'on peut factoriser une partie de l'écriture dans (E) par 2 :

$$a = 2\left(b_{n-1}2^{n-2} + \cdots + b_2 2 + b_1\right) + b_0$$

Par conséquent, le bit de poids faible b_0, qui est strictement inférieur à 2, est le reste de la division entière de a par 2. Le nombre $b_{n-1}2^{n-2} + \cdots + b_2 2 + b_1$, quotient de cette division, est un nombre dont le code binaire est composé de $n-1$ bits et dont le bit de poids faible est b_1. On lui applique la même opération que précédemment, et on continue ainsi jusqu'à obtenir le bit de poids fort de b (obtenu au premier quotient valant 0).

Exemple Soit $a = 241$.

- $a = 2 \times 120 + 1$ donc $b_0 = 1$
- $120 = 2 \times 60 + 0$ donc $b_1 = 0$
- $60 = 2 \times 30 + 0$ donc $b_2 = 0$
- $30 = 2 \times 15 + 0$ donc $b_3 = 0$
- $15 = 2 \times 7 + 1$ donc $b_4 = 1$
- $7 = 2 \times 3 + 1$ donc $b_5 = 1$
- $3 = 2 \times 1 + 1$ donc $b_6 = 1$
- $1 = 2 \times 0 + 1$ donc $b_7 = 1$

Le dernier quotient étant nul, on en déduit que $(241)_{10} = (11110001)_2$

Exercice 3 Coder les nombres suivants en binaire :

$$(17)_{10} \quad ; \quad (64)_{10} \quad ; \quad (341)_{10}$$

Finalement, on peut s'intéresser aux opérations sur les nombres binaires. Les méthodes sont les mêmes qu'en base 10 lorsque l'on pose les opérations, mais il faut se rappeler que l'on ne dépasse pas le chiffre 1.

Exercice 4 Calculer $(1000)_2 + (11011)_2$ et $(101)_2 \times (11)_2$.

Remarque Nous n'avons vu que les nombres **entiers naturels**. Il est possible de coder les nombres entiers **relatifs**, voire même des nombres réels, mais cela est plus compliqué.
En particulier le calcul avec les nombres réels pose des difficultés (lesquelles ?).

III. Le binaire partout !

Le binaire ne permet pas de coder uniquement des nombres, mais peut servir à coder beaucoup de types d'information ou formats de données.

Exemples

- le codage ASCII pour les textes ;
- le codage UTF-8 pour les textes (plus complet que l'ASCII) ;
- le codage BMP pour les images (matricielles) ;

Ces codages sont relativement simples par rapport à d'autres plus évolués mais même les codages les plus complexes développés ont leur traduction en codage binaire, puisqu'il s'agit du seul codage manipulable par l'ordinateur.

Remarque Parfois, pour « simplifier » la lecture humaine, on utilise un codage différent, le système **hexadécimal**, c'est à dire avec 16 digits (ou « chiffres ») : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
Ce codage a un avantage, en relation avec le binaire, sur le codage décimal. En effet, avec 4 bits on code exactement $2^4 = 16$ nombres, donc l'ensemble des digits de la base 16. Ainsi, tout octet est un code pour deux digits en base 16.
Cependant, si en base 10 il y a également besoin de 4 bits, tous les nombres binaires sur 4 bits ne codent pas un digit en base 10 (en effet, $2^3 < 10 < 2^4$).
On donne alors parfois les codages non pas en binaire, mais en hexadécimal, où la notation est donc plus courte (deux digits remplacent huit digits).

On rencontre le binaire également sur tous les supports physiques de données numériques :

- Disques durs ;
- CD, DVD, etc. ;
- mémoire flash (clés USB, cartes mémoires).

La mesure de la taille des supports est donnée le plus souvent en multiples d'octets.

Exercice 5 Combien d'octets fait un mégaoctet ? un mébioctet ? Et un mégabit (débits de connexion donnés en Mbit/s) ?

Exercice 6 Par groupe, faire des recherches sur l'utilisation du langage binaire.
Les thèmes suivants sont proposés :

- Divers codages des nombres relatifs en binaire.
- Codage du texte (ASCII, UTF-8,...)
- Codage des images matricielles sans compression (BMP)

- Supports physiques de données numériques (disques durs et disques optiques)

Préparer une présentation pour la classe mettant en évidence, pour les formats de données et les supports, leur aspect binaire.
La présentation sera faite lors de séances ultérieures de cours (une présentation par séance).