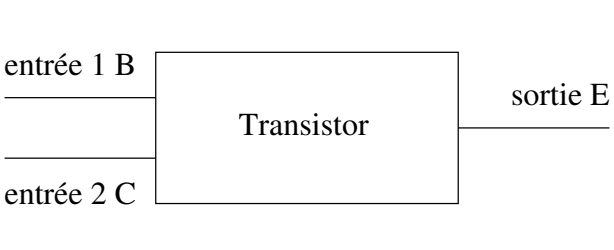
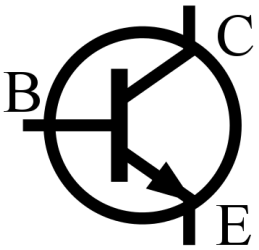
Chapitre :
Représentation numérique

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. Notons ces deux états « courant » et « pas de courant », plus simplement, 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 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 est une « boîte » qui possède deux entrées et une sortie.



Représentation simple



Représentation sur shéma électrique

Représentation physique

Son usage en informatique est le suivant :
Une des deux entrées (la base B) est l'alimentation du transistor (quand le transistor est en fonctionnement, il reçoit nécessairement du courant sur cette entrée). La seconde entrée (le collecteur C) et la sortie (l'émetteur E) sont celles qui permettent le traitement de l'information :
- Si en entrée C il y a du courant, alors il n'y a pas de courant en sortie E.
- Si en entrée C il n'y a pas de courant, alors il y a du courant en sortie E.

Le transistor permet donc de créer en sortie le signal contraire d'un signal donné en entrée.
Mais alors, comment peut-on coder des informations complexes avec seulement 0 et 1 ?
En fait, la machine ne traite pas qu'un bit à la fois, mais un paquet de huit bits. On appelle cela un **octet**.

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

N'a-t-on pas l'impression déjà 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 ?

Et les opérations ? Les méthodes sont les mêmes qu'en base 10, mais encore une fois il faut se rappeler que l'on ne dépasse pas le chiffre 1.

Exercice 3 Calculer $(1000)_2 + (11011)_2$ et $(101)_2 \times (11)_2$.

Comment coder un nombre écrit en base 10 en un nombre écrit en base 2 ?

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 $(241)_2 = (11110001)_2$

Exercice 4 Coder les nombres suivants en binaire :

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

III. Le binaire partout !

1. Codage de toute information numérisée

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

Parfois, pour simplifier la lecture humaine, on utilise un codage intermédiaire 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 l'avantage d'utiliser la totalité des nombres qu'il est possible d'écrire sur 4 bits binaires pour coder les chiffres ($16 = 2^4$), contrairement à ce qui se passe pour la base 10 ($2^3 < 10 < 2^4$).

Exercice 5 Rechercher des informations sur :
– le codage ASCII pour le texte.
– le codage WAV pour le son.
– le codage BMP pour l'image.

Ces codages sont relativement simples, 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.

2. Support numériques

On rencontre le binaire sur tous les supports « numériques » :
– Disques durs ;
– CD, DVD, etc. ;
– mémoire flash (clés USB, cartes mémoires).

Exercice 6 Faire des recherches sur les divers supports numériques et mettre en évidence l'aspect binaire de l'information portée sur ceux-ci.

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

Exemple Qu'est-ce qu'un mégaoctet ? Et un mébioctet ? Et un mégabit (débits de connexion donnés en Mbit/s) ?

IV. Et si 0=Faux et 1=Vrai ? Les booléens

Nous avons vu qu'un transistor, globalement, s'il a en entrée du courant n'en donne pas en sortie et s'il n'a pas de courant en entrée en donne en sortie.
Si, au lieu de considérer que la présence de courant vaut 1 et l'absence de courant vaut 0, on considère que la présence de courant signifie « Vrai » et l'absence de courant signifie « Faux », alors le transistor commence à faire de la logique.
Un transistor fait office de **porte logique non**.
On lui associe la **table de vérité** suivante (à gauche la valeur d'entrée A, à droite la valeur de sortie non A) :

A	**non A**
Vrai	Faux
Faux	Vrai

La logique booléenne est une logique basée sur deux éléments, **Vrai** et **Faux**, la transformation **non** (contraire) que nous venons de voir, et deux lois (ou opérations) logiques, **OU** (disjonction) et **ET** (conjonction).
Voici les tableaux de vérité des deux opérations logiques, qui prennent deux arguments A et B :

$A \setminus B$	Vrai	Faux
Vrai	Vrai	Faux
Faux	Faux	Faux

ET

$A \setminus B$	Vrai	Faux
Vrai	Vrai	Vrai
Faux	Vrai	Faux

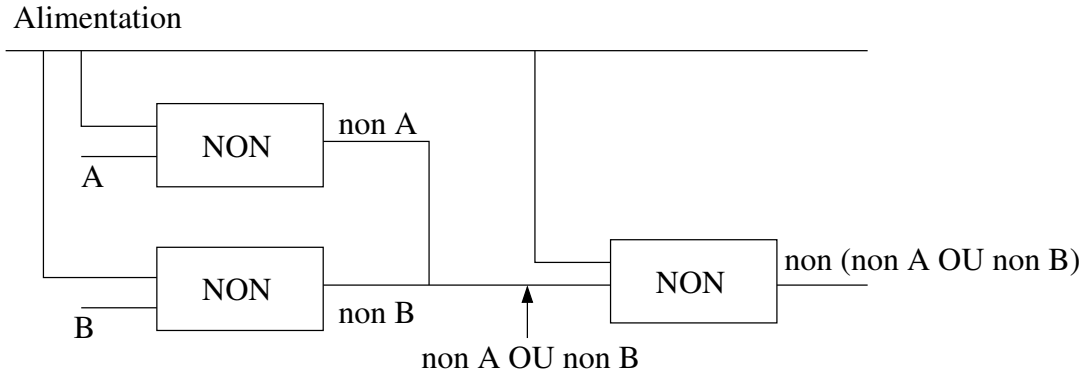
OU

On dispose alors de certaines propriétés (que l'on peut vérifier) comme :
Associativité
A ET (B ET C) = (A ET B) ET C
A OU (B OU C) = (A OU B) OU C
Commutativité
A ET B = B ET A
A OU B = B OU A
Distributivité
A ET (B OU C) = A ET B OU A ET C (noter que le ET est prioritaire sur le OU, comme le $\times$ sur le $+$)
A OU (B ET C) = (A OU B) ET (A OU C)
Éléments neutres
A OU Faux = A
A ET Vrai = A
Absorption
Faux ET A = Faux
Vrai OU A = Vrai
Complémentarité
A = non (non A)
A OU non A = Vrai
A ET non A = Faux

Exercice 7

1. Vérifier que :
   - non (A OU B) = non A ET non B.
   - non (A ET B) = non A OU non B.
2. En déduire que
   - non (non A OU non B) = A ET B.
   - non (non A ET non B) = A OU B.

Cela étant établi, on peut alors brancher plusieurs transistors dans un système pour implanter les deux opérations logiques. D'abord le **ET** :



Ensuite, le **OU** se fait en utilisant le **ET** et le **non** déjà créés, avec l'égalité démontrée en exercice.

Exercice 8 Faire le schéma pour le **OU**.

On définit généralement d'autres opérations logiques pour raccourcir :
- A NOR B = non (A OU B)
- A NAND B = non (A ET B)
- A XOR B = (A ET non B) OU (non A ET B) (c'est le OU eXclusif).

Exercice 9 Faire la table de vérité du OU exclusif.