Codage numérique des nombres relatifs et décimaux

JP Vallon

11 janvier 2013

Pourquoi la base deux ?

La réponse à cette question est à la fois :

- théorique : arithmétique et algèbre de Boole
- technologique :relais électriques , transistors.

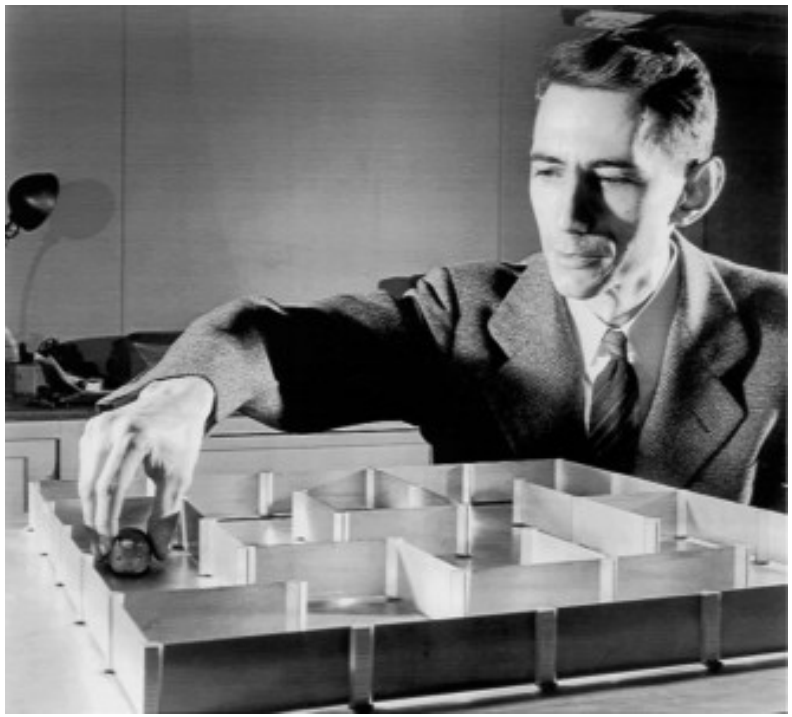


Figure: Claude Shannon(1916-2001)

Claude Shannon apporta des contributions importantes au sujet de l' information numérique.

Analogue Between the Calculus of Propositions

and the Symbolic Relay Analysis

Symbol	Interpretation in relay circuits	Interpretation in the Calculus of Propositions
X	The circuit X.	The proposition X.
0	The circuit is closed.	The proposition is false.
1	The circuit is open.	The proposition is true.
X + Y	The series connection of circuits X and Y	The proposition which is true if either X or Y is true.
XY	The parallel connection of circuits X and Y	The proposition which is true if both X and Y are true.
X'	The circuit which is open when X is closed, and closed when X is open.	The contradictory of proposition X.
=	The circuits open and close simultaneously.	Each proposition implies the other.

Extrait de la thèse de Claude Shannon : A symbolic analysis of relay and switching circuits (1938).

- Tout nombre entier naturel est codé en un entier en base deux.
- La représentation en machine d'un nombre est de taille limité sur 32 bits ou 64 bits.
- Par exemple 4 est représenté sur 32 bits par :
  00000000000000000000000000000100
- Pour obtenir cela on divise 4 par 2 et on obtient : $4 = 2 \times 2 + 0$
  On divise 2 par 2 et on obtient : $2 = 2 \times 1 + 0$
  On divise 1 par 2 et on obtient : $1 = 2 \times 0 + 1$
- La suite des restes dans l'ordre inverse des divisions donne le représentant de 4 :
  $4 = (100)_2$

Il s'agit de représenter aussi les entiers négatifs dans un espace limité.
Imaginons que nous ne pouvons travailler que sur 3 bits.
Si nous utilisons ces 3 bits que pour coder des entiers positifs nous ne pouvons représenter en base deux tous les entiers de 000 jusqu'à 111, en décimal de 0 jusqu'à 7.

Nous avons 8 représentants disponibles de 000 jusqu'à 111.
Nous pouvons coder 0,1,2,3 et -3,-2,-1. Il reste une place 4 ou -4 ?
et comment faire le codage ou la traduction ?

Puisque 4 s'écrit en binaire 100, nous ne le prendrons pas car il va se différencier des autres entiers positifs par son bit de poids fort 1.
Donc nous prenons -4. Il faut maintenant associer 100 puis 101 puis 110 et enfin 111 à -1, -2, -3 et -4.
Comment faire ?

Ce qui va nous aider dans le choix est la volonté de conserver l'algorithme d'addition de deux entiers.

Par exemple :

001 +

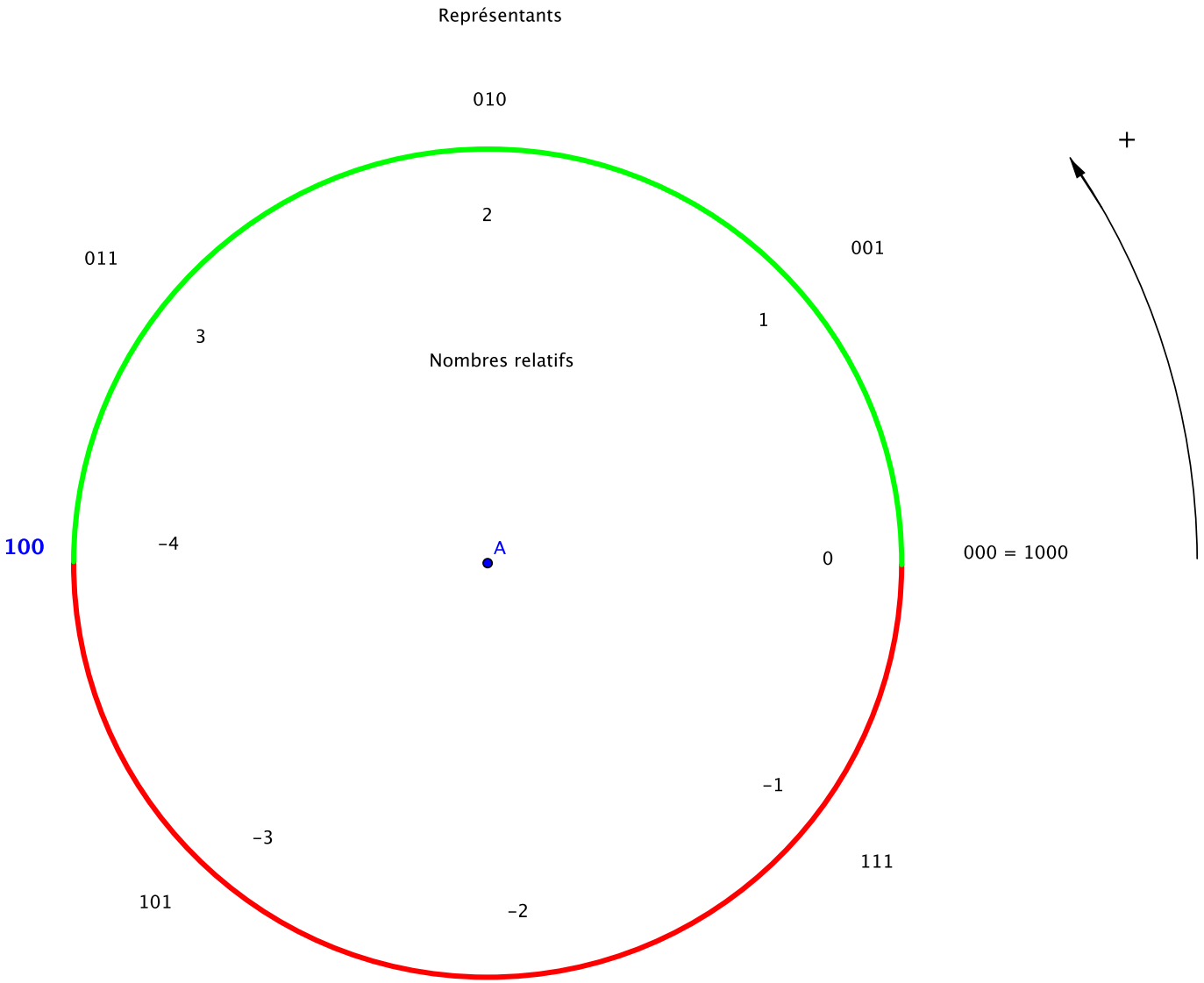
010 =

011

Donc par quoi remplacer x, y et z ?

$$001 + xyz = 000$$

z ne peut être égal qu'à 1.
Avec la retenue y aussi et x aussi.
Par contre on a $001 + 111 = 1\ 000 = 000$
Mais avec une retenue de 1 !
Autrement dit $1000 = 000$ comme $24h = 12h$ ou encore $2\pi = 0$


Représentants
010
2
011
001
3
1
Nombres relatifs
100
-4
A
0
000 = 1000
+
-1
-3
111
101
-2

Généralisons avec n bits par exemple $n = 32$ ou $n = 64$

- Il y a donc 2^n représentants et nombres relatifs
- Il y a $2^{n-1} - 1$ nombres positifs $\sum_{k=0}^{n-1} a_k 2^k$ de 1 jusqu'à $2^{n-1} - 1$ représentés par les mots $0a_{n-1}a_{n-2}...a_0$ où les a_i sont égaux à 0 ou 1.
- Il y a 2^{n-1} nombres négatifs x de -1 à -2^{n-1} représentés par les mots $1a_{n-1}a_{n-2}...a_0$ écriture binaire du nombre décimal $x + 2^n$
- On parle de complémentation à 2^n ou plus simplement à 2

D'où un algorithme pour transformer un entier relatif en un mot de n bits :
(on suppose que l'on a une fonction decToBin qui transforme un entier en base 10 en un entier en base 2)

Entrée : x un entier relatif.

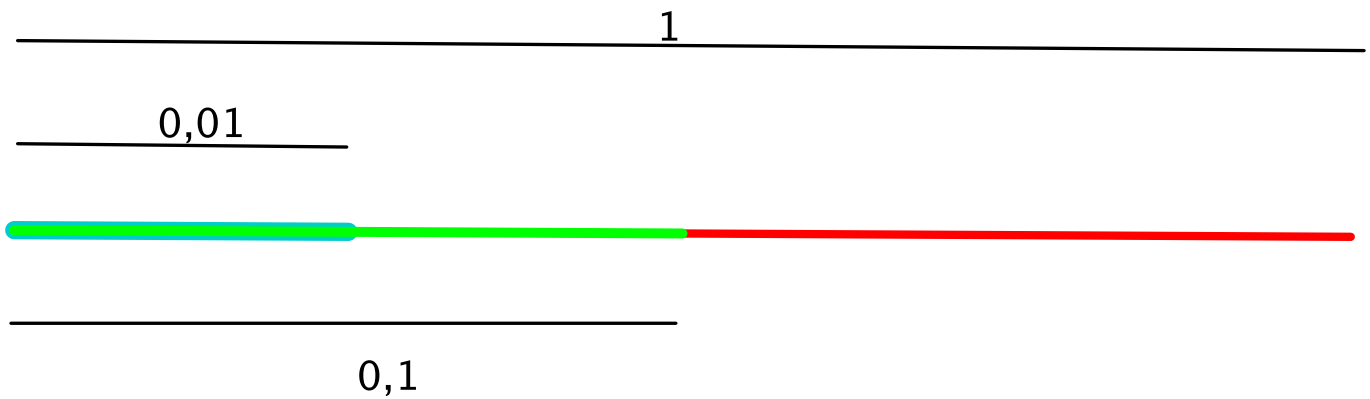
Si $x \geq 0$ alors retourner decToBin(x)
Sinon retourner decToBin($x + 2^n$)

$m = a_7 a_6 ... a_0$ un octet avec $a_i \in \{0, 1\}$
L'entier décimal correspondant est $n = a_7 2^7 + a_6 2^6 + ... + a_0$
Soit m' l'octet obtenu à partir de m en changeant chaque bit en son contraire, c'est à dire si un bit est à 0 on met 1 et vice-versa.
On note $\overline{a_i}$ le contraire du bit a_i
Donc $m' = \overline{a_7}\,\overline{a_6} ... \overline{a_0}$ et l'entier correspondant est
$n = \overline{a_7} 2^7 + \overline{a_6} 2^6 + ... + \overline{a_0}$
Or $n + n' = (a_7 + \overline{a_7}) 2^7 + (a_6 + \overline{a_6}) 2^6 + ... + a_0 + \overline{a_0} =$
$1 \times 2^7 + 1 \times 2^6 + ... + 1 = 255$

Que l'on peut écrire $n' = 255 - n$

Si l'entier relatif x est compris entre 0 et 127 alors il est représenté par l'entier naturel $p = x$ sur 8 bits, et son opposé $-x$ est représenté par l'entier naturel $p' = -x + 2^8 = 256 - p$
Si l'entier relatif x est compris entre -127 et -1 alors il est représenté par l'entier naturel $p = x + 2^8$ sur 8 bits, et son opposé $-x$ qui est positif, est représenté par l'entier naturel
$p' = -x = 2^8 - p$ encore.
Or $256 - p = (255 - p) + 1 = (11111111 - p) + 1$.
Voir page précédente $11111111 - p$ revient à changer tous les bits de p en leur contraire.

Que signifie 0,1 ou 0,01 en binaire ?



0,1 est la moitié de l'unité 1 et 0,01 la moitié de 0,1 ...

Donc 0,1 correspond à 2^{-1} en décimal.

Donc 0,01101 correspond à
$2^{-2} + 2^{-3} + 2^{-5} = 0,25 + 0,125 + 0,03125 = 0,40625$ en décimal.

Algorithme :

Entrée : m la mantisse en base dix, n le nombre de bits pour représenter m en binaire

Sortie : la mantisse en binaire sur n bits

---

indice = 0

Tant que $m! = 1$ indice <= n faire

Début

Tant que m < 1 faire

Début

$m = m * 2$

indice = indice + 1

Fin

Bin[indice] = 1

m= frac(m)

Fin

Exemples :

- $m = 0,1875$
  $0,1875 \times 2 = 0,375$
  $0,375 \times 2 = 0,75$
  $0,75 \times 2 = 1,5$
  on a donc multiplié 3 fois par 2 donc pour l'instant 001
  $0,5 \times 2 = 1$ Fin
  Donc m est représentée par 00110000 sur un octet
- $m = 0,6$ on obtient m représentée par 01000100

Une norme IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) fixe depuis 1985 la représentation des nombres décimaux.
Un réel $R = (-1)^s \times (1 + m) \times 2^e$ avec $0 \leq m < 1$ est réprésenté de la manière suivante :

- sur 32 bits (type float) simple précision : 1 bit pour le signe s, 8 bits pour l'exposant e et 23 bits pour la mantisse m
- sur 64 bits (type double) double précision : 1 bit pour le signe, 11 bits pour l'exposant et 52 bits pour la mantisse m

Codage de 4,75 :

1. 4,75 est positif donc $s = 0$
2. Détermination de l'exposant e et la mantisse m. Il s'agit de diviser ou multiplier ce nombre autant de fois que nécessaire jusqu'à ce que le résultat soit dans l'intervalle [1 ;2[
   $4,75/2 = 2,375$ puis $2,375/2 = 1,1875$ donc :
   - l'exposant est $e = 2$ (le nombre de divisions par 2)
   - la mantisse est $1,1875 - 1 = 0,1875$
3. pour coder m il faut décomposer m en puissances négatives de 2
   $m = 0,1875 = 0,125 + 0,0625 = \frac{1}{8} + \frac{1}{16}$ donc en binaire sur 23 bits 00110000 0000 0000 0000 000

Bien que l'exposant soit un entier relatif on n'utilise pas le complément à 2 pour représenter e mais on s'arrange pour que le représentant de 0 soit au milieu de l'intervalle des représentants.
Ainsi en simple précision il y a 256 représentants possibles pour l'exposant (8 bits) de 0 à 255. On a décidé que 127 représente 0.
Donc 2 est représenté par $129 = 10000001$
Plus généralement l'exposant e est codé par $E = 127 + e$ (8 bits) ou $E = 1023 + e$ (11bits)

$$4,75 = \underbrace{0}_{s}\ \underbrace{10000001}_{e}\ \underbrace{0011\ 0}_{m}\ \underbrace{}_{19}$$ en simple précision sur 32 bits

$$4,75 = \underbrace{0}_{s}\ \underbrace{10000000001}_{e}\ \underbrace{0011\ 0}_{m}\ \underbrace{}_{48}$$ en double précision sur 64 bits

Cas particuliers de la norme IEEE 754 :

- Il y a deux zéros +0 et -0
- Nombre le plus petit : $e_{min} = -126$ donc $E = 00000001$ et $R = 2^{-126}$
- Nombre le plus grand : $e_{max} = 127$ donc $E = 11111110$ et $R = 2^{127}$
- $+\infty$ est représenté par $s = 0$ et $E = 11111111$ et $m = 0$
- $-\infty$ est représenté par $s = 1$ et $E = 11111111$ et $m = 0$
- NaN (Not a Number) Tous les bits sont à 1