Représentation des nombres

Algorithmique et programmation niveau 2

Frédéric Goualard

2021-01-28, v. 0.1

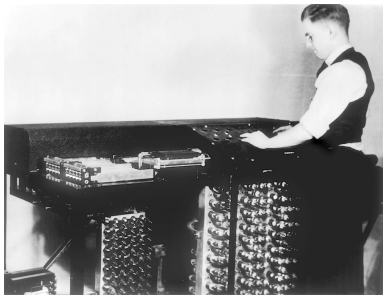


Figure 1 : Clifford Berry et l'*Atanasoff-Berry-Computer*, le premier ordinateur électronique utilisant la représentation binaire, c. 1942. Crédit photo : Encyclopedia Britannica.

In spite of the long-standing tradition of building digital machines in the decimal system, we feel strongly in favor of the binary system for our device. Our fundamental unit of memory is naturally adapted to the binary system since we do not attempt to measure gradations of charge at a particular point in the selectron but are content to distinguish two states. — Arthur W. Burks, Herman H. Goldstine et John *von Neumann*, In [1], 1946

D'un trait supplémentaire de son calame sur la tablette d'argile, un berger sumérien de la troisième dynastie d'Ur (22e–21e s. AEC) pouvait facilement ajouter une nouvelle brebis à son maigre troupeau : un trait pour chaque brebis, cela ne faisait guère que quelques dizaines de traits pour les bergers les plus riches. Avec l'avènement de cités-États de plus en plus grandes et riches, cette manière de comptabiliser les ressources atteint vite ses limites.

Vers 2000 AEC, les babyloniens[2] mirent au point un moyen astucieux d'écrire des grands nombres sur leurs tablettes sans avoir besoin d'un nombre important de symboles. Ils utilisaient le symbole « 𒁹 » pour la valeur « 1 » et le symbole « 𒌋 » pour la valeur « 10 ». En combinant ces deux symboles, ils représentaient tous les entiers entre 1 et 59 (Table 1).

Table 1 : Représentation babylonienne des entiers de 1 à 59.

𒁹	1	𒌋𒁹	11	𒌋𒌋𒁹	21	𒌋𒌋𒌋𒁹	31	𒌋𒌋𒌋𒌋𒁹	41	𒌋𒌋𒌋𒌋𒌋𒁹	51
𒁺	2	𒌋𒁺	12	𒌋𒌋𒁺	22	𒌋𒌋𒌋𒁺	32	𒌋𒌋𒌋𒌋𒁺	42	𒌋𒌋𒌋𒌋𒌋𒁺	52
𒁻	3	𒌋𒁻	13	𒌋𒌋𒁻	23	𒌋𒌋𒌋𒁻	33	𒌋𒌋𒌋𒌋𒁻	43	𒌋𒌋𒌋𒌋𒌋𒁻	53
𒁼	4	𒌋𒁼	14	𒌋𒌋𒁼	24	𒌋𒌋𒌋𒁼	34	𒌋𒌋𒌋𒌋𒁼	44	𒌋𒌋𒌋𒌋𒌋𒁼	54
𒁽	5	𒌋𒁽	15	𒌋𒌋𒁽	25	𒌋𒌋𒌋𒁽	35	𒌋𒌋𒌋𒌋𒁽	45	𒌋𒌋𒌋𒌋𒌋𒁽	55
𒁾	6	𒌋𒁾	16	𒌋𒌋𒁾	26	𒌋𒌋𒌋𒁾	36	𒌋𒌋𒌋𒌋𒁾	46	𒌋𒌋𒌋𒌋𒌋𒁾	56
𒁿	7	𒌋𒁿	17	𒌋𒌋𒁿	27	𒌋𒌋𒌋𒁿	37	𒌋𒌋𒌋𒌋𒁿	47	𒌋𒌋𒌋𒌋𒌋𒁿	57
𒂀	8	𒌋𒂀	18	𒌋𒌋𒂀	28	𒌋𒌋𒌋𒂀	38	𒌋𒌋𒌋𒌋𒂀	48	𒌋𒌋𒌋𒌋𒌋𒂀	58
𒂁	9	𒌋𒂁	19	𒌋𒌋𒂁	29	𒌋𒌋𒌋𒂁	39	𒌋𒌋𒌋𒌋𒂁	49	𒌋𒌋𒌋𒌋𒌋𒂁	59
𒌋	10	𒌋𒌋	20	𒌋𒌋𒌋	30	𒌋𒌋𒌋𒌋	40	𒌋𒌋𒌋𒌋𒌋	50		

Pour représenter un nombre supérieur ou égal à 60, ils considéraient le nombre de paquets de 60 qu'il contenait. Par exemple, le nombre 63 contient 1 paquet de 60 et 3 unités. Sa représentation était donc 𒁹 𒐗 : le nombre d'unités se trouve à droite et le nombre de paquets de 60 à gauche. En considérant chacun des cinquante-neuf graphismes de la table 1 comme un symbole unique, on peut ainsi représenter avec deux symboles tous les entiers entre 60 et $59 \times 60 + 59 = 3599$. La représentation de 3599

Figure 2 : Écriture cunéiforme sur une tablette d'argile à l'aide d'un calame.
Crédit photo : Shevlin Sebastian.

[1] Arthur W. Burks, Herman H. Goldstine et John von Neumann. *Preliminary Discussion of the logical design of an electronic computing instrument*. Technical Report. The Institute for Advanced Study, juin 1946

[2] Stephen Chrisomalis. *Numerical notation : a comparative history*. Cambridge, New York : Cambridge University Press, 2010

est simple : 𒌋𒐗 𒌋𒐗. On constate qu'il y a cependant un problème pour représenter 60 car il se décompose en $1 \times 60 + 0$, or les babyloniens n'avaient pas de représentation du zéro. Ils le remplaçaient par un espace dans la notation ; 60 s'écrit donc 𒐗 , soit exactement comme 1, et c'est le seul contexte qui permet de différencier les deux.

Pour représenter les nombres au-delà de 3599, il suffit de faire des paquets de $60 \times 60 = 3600$, puis des paquets de 60^3, 60^4, ... Le nombre $2\,808\,307 = 13 \times 60^3 + 0 \times 60^2 + 5 \times 60^1 + 7 \times 60^0$ s'écrit, par exemple, 𒐝𒐐 𒐖 𒐗. Notez l'espace entre la représentation du 13 (𒐝𒐐) et celle du 5 (𒐖) pour représenter l'absence de paquet de 60^2.

L'absence de symbole spécifique pour représenter le zéro devait singulièrement compliquer la tâche des scribes chargés de relire les valeurs inscrites sur les tablettes : un espace entre les symboles représente t-il une absence de puissance ou est-ce seulement un décalage malheureux lors de l'impression de la tablette ? Malgré cela, le zéro semble être apparu très tardivement car les preuves matérielles les plus anciennes de son existence datent des alentours de 875 EC en Inde.

Si l'on fait abstraction de l'absence de zéro dans le système babylonien, ce système de représentation des nombres correspond à ce que l'on appelle aujourd'hui une *représentation positionnelle en base 60* : chaque nombre s'exprime sous la forme d'une chaîne de symboles, le symbole le plus à droite représentant le nombre d'unités (ou, le nombre de fois où 60^0 apparait dans le nombre), le symbole à sa gauche représente le nombre de fois où 60^1 apparait, puis le nombre de fois où 60^2 apparait, ... La valeur d'un même symbole dépend donc de sa position dans la chaîne.

Si l'on remplace la base 60 par la base 10 et que l'on introduit un symbole pour le zéro, de façon à ne plus devoir introduire des espaces dans les représentations des nombres, on n'a plus besoin que de dix symboles pour représenter n'importe quel nombre. Dans la base 10, le symbole le plus à droite représente le nombre d'unités (le nombre de fois où 10^0 apparait), le symbole à sa gauche les *dizaines* (le nombre de fois où 10^1 apparait), etc.

En utilisant le symbole moderne « 0 » pour le zéro et les symboles 𒐗, 𒐐, 𒐝, 𒐖, 𒐘, 𒐙, 𒐚, 𒐛, on pourrait représenter le nombre $307 = 3 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$ par :

𒐝	0	𒐘
10^2	10^1	10^0

En remplaçant les symboles babyloniens par les chiffres arabes 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9, on retrouve la notation des nombres dont nous avons l'habitude.

Pour récapituler, on peut choisir de représenter les nombres entiers dans n'importe quelle base. Pour une base b donnée, on doit posséder b symboles différents et la représentation d'un nombre dans la base se fait en déterminant le nombre de paquets de chaque puissance de la base.

Exemple — Représentation en base 7

Déterminons la représentation de 357 en base 7. Pour cela, on va utiliser les sept symboles 0, 1, 2, 3, 4, 5 et 6. On a :

$$357 = \mathbf{1} \times 7^3 + \mathbf{0} \times 7^2 + \mathbf{2} \times 7^1 + \mathbf{0} \times 7^0$$

La représentation en base 7 de 357 est donc 1020.

Dans la suite, on sera appelé à représenter des nombres dans différentes bases. Afin d'éviter toute ambiguïté (voir figure 3), on écrira désormais la base b d'un nombre a de la façon suivante : a_b. On aura donc, par exemple :

$$357_{10} = 1020_7$$

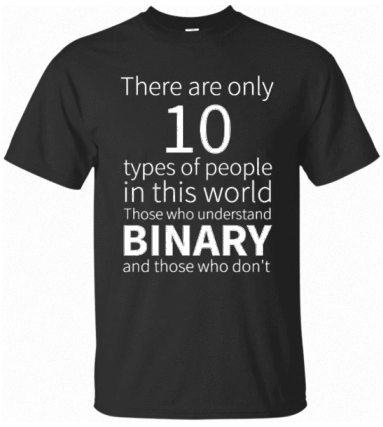


FIGURE 3 : Un tee-shirt jouant sur les ambiguïtés de la représentation positionnelle si l'on ne connait pas la base utilisée.

La base 10 est la base communément utilisée aujourd'hui, même si la base 60 est encore rencontrée dans certains domaines ; c'est par exemple le cas pour exprimer une durée en heures, minutes et secondes : lorsque l'on écrit un temps sous la forme :

$$04 : 13 : 07$$

c'est à dire quatre heures, treize minutes et sept secondes, cela correspond à :

$$4 \times 60^2 + 13 \times 60^1 + 7 \times 60^0$$

secondes.

Comme rappelé dans l'épigraphe de ce fascicule et évoqué par Buchholz [3], les mécanismes internes des ordinateurs justifient l'emploi de la base 2 plutôt que de la base 10 en leur sein. Une autre base communément utilisée en informatique est la base 16, car 16 étant une puissance de 2, il est très facile de passer d'une base à l'autre et la représentation en base 16 d'un nombre requiert quatre fois moins de symboles qu'en base 2.

[3] W. Buchholz. "Fingers or fists? (the choice of decimal or binary representation)". In : *Communications of the ACM* 2.12 (déc. 1959), p. 3-11

Table 2 : Correspondance symboles hexadécimaux / chaînes de quatre bits.

Binaire	Hexadécimal	Décimal	Binaire	Hexadécimal	Décimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

Le passage de la base 2 à la base 16 se fait en considérant les chiffres binaires (ou « *bits* », pour *Binary digits*) par paquets de quatre de la droite vers la gauche du nombre à recoder et en remplaçant chaque groupe par un des seize symboles hexadécimaux conformément à la table 2. Notez que, la base 16 nécessitant seize symboles différents, on est obligé de trouver six symboles à rajouter aux dix chiffres arabes. Par convention, on utilise les lettres de « A » à « F ».

Exemple — Passage de la base 2 à la base 16

Le nombre binaire 1011010_2 se recode en base 16 en regroupant les bits par paquets de quatre *en partant de la droite* et en ajoutant des zéros à gauche si besoin, puis en appariant chaque groupe avec sa représentation hexadécimale :

$$\underbrace{0101}_{5}\ \underbrace{1010}_{\mathrm{A}}$$

La représentation hexadécimale de 1011010_2 est donc $5A_{16}$.

Le passage de la base 16 à la base 2 se fait de la même manière : chaque chiffre hexadécimal est remplacé par une chaîne de quatre bits conformément à la table 2.

Exemple — Passage de la base 16 à la base 2

Le nombre hexadécimal $3A7_{16}$ est représenté en binaire en concaténant les chaînes

de quatre bits pour chaque symbole hexadécimal :

$$\overbrace{0011}^{3}\ \overbrace{1010}^{A}\ \overbrace{0111}^{7}$$

La représentation binaire de $3A7_{16}$ est donc 001110100111_2.

Le passage d'une base b_1 à une base b_2 lorsqu'il n'existe pas de relation spéciale entre b_1 et b_2 demande un peu plus de calculs. En pratique, on s'intéressera seulement au passage de la base 10 à une base b ou d'une base b à la base 10[4].

[4] La maîtrise de ces deux techniques de recodage permet évidemment de passer indirectement d'une base b_1 à une base b_2 en transitant par la base 10.

Passage d'une base b à la base 10

Considérons un nombre quelconque x exprimé dans la base b. Soit :

$$d_k d_{k-1} \cdots d_1 d_{0_b}$$

la représentation de x en base b. Comme on l'a vu précédemment, cela veut dire que l'on a :

$$x = d_k \times b^k + d_{k-1} \times b^{k-1} + \cdots + d_1 \times b^1 + d_0 \times b^0 \tag{1}$$

En remplaçant chaque symbole d_i par sa valeur, il est possible d'effectuer le calcul de l'équation (1) en base 10 pour obtenir la valeur de x en base 10.

Exemple — Passage de la base 5 à la base 10

Soit le nombre x exprimé en base 5 par la chaîne 321014_5. On a :

$$\begin{aligned} x &= 3 \times 5^5 + 2 \times 5^4 + 1 \times 5^3 + 0 \times 5^2 + 1 \times 5^2 + 4 \times 5^0 \\ &= 10779_{10} \end{aligned}$$

Exemple — Passage de la base 31 à la base 10

Comme pour la base 16, on utilise les vingt-et-unes lettres de A à U en plus des chiffres de 0 à 9 pour avoir trente-et-un symboles. La lettre A vaut 10, la lettre B vaut 11, ...jusqu'à la lettre U qui vaut 30.

Le nombre x exprimé en base 31 par la chaîne HELLO_{31} s'exprime en base 10 par :

$$\begin{aligned} x &= \mathsf{H} \times 31^4 + \mathsf{E} \times 31^3 + \mathsf{L} \times 31^2 + \mathsf{L} \times 31^1 + \mathsf{O} \times 31^0 \\ &= 17 \times 31^4 + 14 \times 31^3 + 21 \times 31^2 + 21 \times 31^1 + 24 \times 31^0 \\ &= 16137787_{10} \end{aligned}$$

Passage de la base 10 à la base b

On connait désormais un nombre x en base 10 et on souhaite l'exprimer dans une base b. On sait que x s'écrit en base b avec k symboles :

$$x = d_k d_{k-1} \cdots d_1 d_{0_b}$$

On cherche à trouver les symboles $d_0, \cdots, d_k$.

Par définition, on a :

$$\begin{aligned} x &= d_k d_{k-1} \cdots d_1 d_{0_b} \\ &= d_k \times b^k + d_{k-1} \times b^{k-1} + \cdots + d_1 \times b^1 + d_0 \times b^0 \end{aligned}$$

En considérant la partie droite de la deuxième équation comme un polynôme en la variable b, on peut réécrire l'équation avec une forme de Horner :

$$\begin{aligned} x &= d_k \times b^k + d_{k-1} \times b^{k-1} + \cdots + d_1 \times b^1 + d_0 \times b^0 \\ &= ((\cdots(((d_k \times b + d_{k-1}) \times b + d_{k-2}) \times b + \cdots) \times b + d_1) \times b + d_0 \end{aligned}$$

Exemple — Représentation sous forme de Horner

Le nombre 72431_{10} vaut :

$$\begin{aligned} 72431_{10} &= 7 \times 10^4 + 2 \times 10^3 + 4 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 \\ &= (((7 \times 10 + 2) \times 10 + 4) \times 10 + 3) \times 10 + 1 \end{aligned}$$

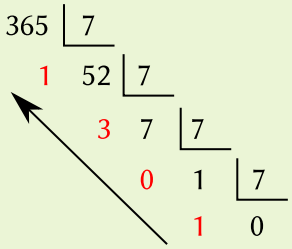
Considérons la forme factorisée :

$$x = (\underline{(\cdots(((d_k \times b + d_{k-1}) \times b + d_{k-2}) \times b + \cdots)} \times b + d_1) \times b + d_0$$

On voit que si l'on divise x par b en suivant les règles de la base 10, on obtient un quotient correspondant à l'expressions soulignée en bleu et un reste égal à d_0. De la même façon, si l'on divise ce quotient par b de nouveau, on obtient un quotient correspondant à l'expression soulignée en rouge et un reste égal à d_1. En continuant à diviser chaque quotient successif par b jusqu'à obtenir un quotient nul, on peut ainsi obtenir en reste chacun des symboles servant à représenter x en base b. Notez cependant que l'on obtient les « chiffres » de x à l'envers : le premier « chiffre » trouvé est celui le plus à droite (les unités).

Exemple — Conversion de base 10 en base 7

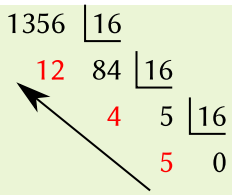
On cherche la représentation en base 7 de 365_{10}. On effectue des divisions successives par 7 :



On en déduit 365_{10}=1031_7.

Exemple — Conversion de base 10 en base 16

On veut convertir le nombre 1356_{10} en hexadécimal. On effectue les divisions successives par 16 :



On doit ensuite convertir tous les restes suivant la table 2 pour que chaque reste corresponde à un seul symbole. Il vient : 1356_{10}=$54C_{16}$.

Représentation des bases en C++

Le langage C++ offre la possibilité de représenter directement les nombres dans les bases les plus utilisées en informatique : les bases 2, 8, 10 et 16. Le choix de la base utilisée est indiqué par un préfixe :

- 0b ou 0B : base 2 (exemple : 0b1001 pour 9_{10}) ;
- 0 : base 8 (exemple : 056 pour 46_{10}) ;
- 0x ou 0X : base 16 (exemple : 0xa2 pour 162_{10}). Les symboles hexadécimaux peuvent être écrits indifféremment en minuscules ou majuscules.

Les nombres exprimés en base 10 n'ont pas de préfixe. C'est la base par défaut.

Exemple — Représentation des nombres en C++

Soit le programme :

```
#include <iostream>

using namespace std;

int main(void)
{
   int a = 0x13;
   cout << a << endl;
   if (a == 0b10011 && 034 == 28) {
      cout << "Vrai" << endl;
   } else {
      cout << "Faux" << endl;
   }
}
```

Il affiche :

```
19
Vrai
```

Il est possible d'utiliser certains des préfixes ci-dessus pour entrer un nombre au clavier. Il faut cependant insérer au préalable dans le flux d'entrée un *manipulateur* indiquant la base souhaitée avant de lire la variable elle-même[5]. Les manipulateurs disponibles sont :

dec. Lecture en décimal (par défaut) ;

hex. Lecture en hexadécimal ;

oct. Lecture en octal.

Le nombre sera lu dans la base correcte (et stocké en machine sous forme binaire, évidemment). L'utilisation des manipulateurs `hex` et `oct` dispense de l'emploi des préfixes correspondants pour la saisie.

[5] Les manipulateurs sont accessibles après inclusion du fichier « iomanip ».

À l'inverse, l'affichage dans une base autre que la base 10 est aussi possible. Plus précisément, il est aisé de préciser un affichage dans les bases 8, 10 et 16 en utilisant la fonction `setbase()` comme montré dans l'exemple ci-dessous.

Exemple — Entrées/sorties avec différentes bases en C++

Soit le programme :

```
#include <iostream>
#include <iomanip>

using namespace std;

int main(void)
{
   int a;
   cout << "Valeur décimale ? ";
   cin >> a;
   cout << "Valeur lue en octal: " << setbase(8) << a << endl;
   cout << "Valeur lue en hexadécimal: " << setbase(16) << a << endl;
   cout << "Valeur octale ? ";
   cin >> oct >> a;
   cout << "Valeur lue en décimal: " << setbase(10) << a << endl;
   cout << "Valeur lue en hexadécimal: " << setbase(16) << a << endl;
   cout << "Valeur hexadécimale ? ";
   cin >> hex >> a;
   cout << "Valeur lue en octal: " << setbase(8) << a << endl;
   cout << "Valeur lue en décimal: " << setbase(10) << a << endl;
}
```

Une exécution possible serait :

```
Valeur décimale ? 45
Valeur lue en octal: 55
Valeur lue en hexadécimal: 2d
Valeur octale ? 056
Valeur lue en décimal: 46
Valeur lue en hexadécimal: 2e
Valeur hexadécimale ? 0xaae
Valeur lue en octal: 5256
Valeur lue en décimal: 2734
```

Vous aurez noté que, s'il est possible de spécifier des constantes immédiates en binaire dans un programme C++, rien n'est prévu pour lire ou écrire dans cette base.

Pour lire et écrire en binaire, il va falloir ruser en utilisant le type « `bitset` » correspondant à un tableau de bits.

Exemple — Entrées/sorties en binaire avec C++

Soit le programme :

```
#include <iostream>
#include <string>
#include <bitset>

using namespace std;
```

```cpp
int main(void)
{

   string s;
   cout << "Valeur en binaire (ex.: 1010) ? ";
   cin >> s;
   auto v = bitset<64>(s).to_ulong();
   cout << v << endl;

   cout << "Valeur en décimal ? ";
   cin >> v;
   cout << bitset<64>(v) << endl;
}
```

Une exécution possible est :

```
Valeur en binaire (ex.: 1010) ? 11101
29
Valeur en décimal ? 34
00000000000000000000000000000000000000000000000000000000000100010
```

Pour la lecture en binaire, on lit une chaîne de caractères composée uniquement de zéros et de uns, on crée un `bitset` à partir de cette chaîne, puis on utilise la méthode `to_ulong()` de `bitset` pour transformer le tableau de bits en un entier.

L'affichage en binaire se fait de manière analogue : on crée un `bitset` à partir d'une variable entière puis on demande d'afficher le tableau de bits correspondant sous forme de chaîne de caractères. Un affichage en binaire sans tous les zéros à gauche demande un peu plus de travail sur la chaîne de caractères retournée par `bitset<64>(v)`, mais reste tout à fait faisable.

1 Licence de ce document