Jean-Jacques Schwarz

Architecture des ordinateurs

ISBN : 2-212-11635-7
EYROLLES

1

Information et codages

Des chiffres et des lettres.

Ou comment matérialiser l'information ?

Vers une première approche de l'information

La matière première sur laquelle travaille un ordinateur est l'information et il est donc nécessaire de donner à celle-ci une représentation adaptée.

Le fonctionnement des ordinateurs actuels reposant entièrement sur le système binaire, il est alors naturel que la représentation de l'information se fasse également sous la forme de données binaires.

Les informations traitées dans ce chapitre sont dites *informations simples* : il s'agit essentiellement de nombres et de mots. Après avoir introduit le bit comme unité de mesure de l'information, la première partie est consacrée aux codages des nombres (entiers et réels) et des caractères (lettres d'un alphabet).

Les dispositifs de stockage (mémoires) et de transport de l'information (réseaux) sont dépendants de l'environnement physique dans lequel ils opèrent. À ce titre, ils peuvent introduire des altérations et ainsi provoquer la restitution de données erronées. La deuxième partie du chapitre traite des techniques de détection et de correction de ces erreurs.

Sur un autre plan, le stockage et la transmission de données génèrent des coûts dépendant de la taille de ces données. En se servant de la redondance inhérente à toute information réelle, la dernière partie du chapitre donne un aperçu sur les techniques de compactage des données.

Information et quantité d'information

L'information est un élément de connaissance, donc une notion abstraite, concernant un élément du monde réel (un objet, un événement réel, mais aussi une idée) et qui, pris dans un contexte déterminé, a une signification et une pertinence humaine. Cet élément de connaissance est susceptible d'être codé pour être conservé, traité ou communiqué.

Une information est fondamentalement un élément de savoir, c'est-à-dire une idée dont la résidence ou source première est notre cerveau. Tant que nous n'avons qu'à réfléchir au sujet de cette idée, il suffit de laisser faire nos neurones… Le souci de la représentation de l'information apparaît lorsque survient le besoin de procéder à une mémorisation externe, même temporaire :

- soit parce que le traitement est trop fastidieux ou complexe pour être effectué en une seule fois (nécessité de noter des résultats intermédiaires) ;
- soit pour mémoriser cet élément de savoir pour le retrouver ultérieurement (numéro de téléphone) ;
- soit encore parce que l'on souhaite communiquer cette information à un tiers (transmission d'un savoir).

Il faudra alors bien donner à cette information une consistance physique. De l'idée en tête à la transmission, il faut être capable de donner à l'information une matérialisation permettant de la transporter. La problématique est la même pour le stockage. L'écriture et la parole sont des exemples de représentation d'une information, statique dans le premier cas (permet du stockage), dynamique dans le second (permet le transport).

Donnée

La donnée est une représentation conventionnelle d'une information sous une forme adaptée à son traitement par ordinateur.

L'opération de transformation d'une information lui donnant une réalité physique est appelée *codage*. Le codage de l'information produit une donnée dont on attend qu'elle puisse restituer aussi fidèlement que possible l'information d'origine (interprétation de la donnée). La donnée est prise ici dans le contexte de son traitement par un ordinateur.

Message

Dans le cas d'une communication, la donnée doit être déplacée (voire recopiée) d'une source vers un destinataire en utilisant un certain support de transmission, support papier (carte, ruban), magnétique (disquette), optique (CD-Rom), onde électromagnétique, acoustique…

Un ensemble de données en cours de transport constitue un message.

Il est tout naturel qu'une même information puisse être codée (matérialisée) de différentes manières : par exemple, la notion de date peut être exprimée à l'aide d'un texte (suite de caractères, « cinq mai mille neuf cents ») ou d'un mot numérique, c'est-à-dire une suite de chiffres comme « 05051900 ». Pour être transmis, le texte résultant peut être

supporté par des signaux sonores (tam-tam) ou lumineux (volutes de fumée) pour devenir le message.

Il n'y a ainsi pas de manière unique d'opérer une représentation de l'information : on utilise le mode de représentation le plus approprié au contexte ou au traitement à faire. Une donnée est un modèle de représentation d'une information comme le portrait est un modèle de représentation d'un personnage : il y a autant de portraits possibles que de manières de voir le personnage, et ces portraits sont tous plus ou moins fidèles…

Unité de mesure de l'information : le bit

Pourquoi mesurer l'information et surtout quoi mesurer ? Tant que notre élément de savoir occupe simplement nos neurones, il n'est pas vraiment utile d'avoir une mesure de l'information. La mesure devient utile dès l'instant où elle doit être représentée concrètement. Elle occupe alors un espace et du temps sur un support physique. Combien de place faut-il pour stocker cette information ? Combien de temps faut-il pour la transmettre ?

Plutôt que de vouloir évaluer le sens ou la pertinence de cette information (sémantique), il est avant tout utile de l'évaluer sous la forme d'un volume ou d'une quantité. Ainsi, mesurerons-nous l'information pour estimer son coût de stockage ou son coût de transport, au même titre que le nombre de pages d'un livre et son poids permettent d'estimer la possibilité de le ranger sur une étagère ou d'évaluer les éventuels frais d'expédition.

Comment mesurer la quantité d'une entité, a priori, purement abstraite ? Certaines professions utilisent une mesure de la quantité d'information. Le journaliste, le romancier peuvent être rémunérés en fonction d'une quantité d'information mesurable en nombre des lignes ou mots (c'est-à-dire en fonction de la place prise par la représentation de l'information sur le support), ou alors, et cela risque d'être plus rémunérateur, en fonction de l'originalité, de l'inattendu de l'information délivrée (le scoop).

Cette notion d'inattendu est le point de départ de la construction de l'unité de mesure faite par Shannon : l'inattendu apporte une grande quantité d'information, alors que le connu n'en apporte aucune. Dire que le journal télévisé aura lieu ce soir à 20h00 n'apporte aucune information, alors qu'un flash d'information en pleine journée sera supposé être associé à une grande quantité d'information.

L'information étant par définition abstraite, la mesure de l'information ne peut s'appuyer que sur un outillage destiné à la manipulation des abstractions. Dans le domaine concerné, celui de la quantification de l'inattendu, ce sont les probabilités qui constituent le formalisme mathématique le plus approprié.

Pour définir la quantité d'information Q d'une information, une variable aléatoire X lui est associée. Le domaine de définition de cette variable est l'ensemble des « valeurs » x pouvant être prises par cette information. Ces valeurs sont supposées être en nombre fini (la variable est dite *discrète*) et suivre une loi de probabilité connue. La détermination de Q est faite sur la base suivante : sa valeur doit être d'autant plus grande que l'information est inattendue et être nulle dans le cas d'une information connue a priori. Elle est

positive ou nulle : il n'y a pas d'anti-information… D'un point de vue mathématique, Q est d'autant plus grande que la probabilité d'une valeur x est faible et est nulle si la probabilité est égale à 1 (ce qui est le cas d'une constante). Q est construite comme une fonction, restant encore à définir, inverse de la probabilité :

$Q = f(1/p(x))$ avec $p(x) = 1 \Rightarrow Q = 0$

La loi de probabilité doit être connue. En réalité, la définition utilisée ne s'applique qu'aux cas où toutes les valeurs sont équiprobables. Le calcul de la quantité d'information est limité au cas d'une loi de probabilité uniforme discrète. Ainsi, le calcul de la quantité d'information de la variable associée au résultat d'un lancer de dé n'a de sens que si celui-ci n'est pas pipé… (ce qui se traduit par l'hypothèse de l'équiprobabilité). Si n est le nombre de valeurs possibles alors p(x) est égale à 1/n, soit p = 1/6 pour le dé.

Comment choisir la fonction f ? La définition de la quantité d'information est conçue pour intégrer le fait qu'une variation de probabilité de 0,1 n'a pas le même effet, du point de vue de l'inattendu, si l'on passe d'une probabilité de 0,8 à 0,9 ou d'une probabilité de 0,1 à 0,2. Dans le premier cas, la probabilité ne change en relatif que de 10 %, alors que dans le second cas elle double la valeur. La fonction à définir n'est donc pas linéaire : elle doit être plus « sensible » pour les valeurs faibles que pour les valeurs fortes. Ce type d'évolution est habituellement bien pris en compte par la fonction logarithme. Dans ces conditions Q s'exprime sous la forme :

$Q = \log_b (1/p(x))$ où b est la base du logarithme.

Cette fonction étant notre « appareil » de mesure de la quantité d'information, il est maintenant nécessaire de l'étalonner, c'est-à-dire de définir son unité de mesure. Avec la technologie actuelle de l'informatique, l'information est représentée à l'aide de variables binaires (0 ou 1) qu'il est facile de caler sur des phénomènes physiques (un courant qui passe ou ne passe pas, une orientation nord-sud d'un aimant…). L'étalonnage est fait par rapport à ce système binaire.

Par définition, le bit (binary digit) est l'unité d'information apportée par une variable binaire dont les deux valeurs sont équiprobables.

D'après cette définition, on a :

$Q = \log_b (1/p(x)) = 1$ bit et comme $p(x) = 1/2$, il s'en suit que $b = 2$.

La formulation finale de la quantité d'information donnée par Claude Shannon est ainsi :

$Q = \log_2 (1/p(x))$ bits

Le bit est l'unité de mesure de la quantité d'information dans un système binaire.

Il faut faire la distinction entre le *contenant* et le *contenu*. Le contenant est l'élément de mémoire capable de stocker de l'information. La technologie actuelle de mémorisation, aussi bien pour la mémoire centrale à semi-conducteurs, que pour les unités de disques magnétiques ou encore les CD-Rom optiques, est entièrement basée sur le système binaire : le plus petit élément mémorisable correspond à un 0 ou à un 1, c'est-à-dire à une mémoire à deux états. Le plus petit contenant d'information vaut 1 bit ($\log_2 (1/0.5)$).

Une mémoire de un bit a deux états possibles, une mémoire de deux bits a quatre états et ainsi de suite (2^n états pour n bits). Il est à noter, qu'alors que la quantité d'information d'une variable (le contenu) peut prendre n'importe quelle valeur (un nombre réel comme dans le cas du dé), la mémoire (le contenant) est forcément exprimée en nombre entier de bits.

Pour la mémorisation d'une information, il faut donc disposer d'une mémoire avec assez de bits pour que le nombre de ses états soit supérieur au nombre de valeurs possibles de la variable aléatoire associée à cette information. Cette règle est illustrée avec l'exemple du dé (figure 1-1).

Figure 1-1
Dé

Dé	Mémoires		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Les valeurs possibles étant au nombre de 6, la quantité d'information Q de la variable associée au résultat d'un jet de dé est égale à $\log_2(6)$, soit Q = 2,58 bits. Il faut une mémoire de 3 bits – donc à 8 états – pour stocker le résultat d'un jet. Huit configurations sont donc à disposition pour mémoriser les 6 valeurs. La règle de codage utilisée pour la mémorisation correspond ici à l'utilisation des 6 premiers nombres entiers pour représenter les 6 symboles chiffres inscrits sur le dé. Cela revient à laisser inutilisées les deux configurations extrêmes : le contenant vaut 3 bits et le contenu vaut 2,58 bits.

À noter

Si le dé est jeté 10 fois, on utilisera, pour des questions de commodité, une mémoire de 30 bits, alors qu'une mémoire de 26 bits serait suffisante (10 jets indépendants valent $10 \times 2{,}58$ soit 25,8 bits). Dans ce cas, il y a une perte de place, mais au profit de la commodité d'emploi.

Le même raisonnement s'applique à la quantité d'information d'un chiffre décimal. Quelle est la quantité d'information d'un chiffre ? Il y a 10 chiffres que l'on suppose équiprobables. Le calcul donne $Q = \log_2(10) \approx 3{,}32$ bits : il faut évidemment 4 bits mémoire pour stocker un chiffre décimal (il y a 16 configurations possibles mais 6 d'entres elles ne sont pas utilisées).

Un bit est rarement stocké tout seul : généralement la mémoire est structurée sur la base d'un groupe ordonné de 8 bits appelé « octet » (*byte* en anglais) Un octet est une sorte de panier dans lequel chacun des 8 bits est défini par sa position ou son rang. Un octet permet de mémoriser une information à 2^8 (256) valeurs possibles. Il sert couramment à stocker les caractères d'un texte (un caractère par octet) ou les nuances de gris d'une image (une couleur par octet), voire même des nombres si la précision n'est pas un critère retenu.

La notion de quantité d'information étant précisée, voyons maintenant les informations élémentaires qui feront l'objet d'un codage : les nombres et les caractères. Commençons par les nombres, d'abord les plus « simples », les entiers, puis les réels.

Les nombres et leur codage

Le nombre est un concept que nos ancêtres ont créé pour répondre à deux besoins : celui de pouvoir compter les éléments d'un ensemble (nombres cardinaux) – par exemple le comptage des animaux d'un troupeau – et celui de pouvoir ordonner (nombres ordinaux) des objets suivant un critère déterminé (classement d'objets par ordre de valeurs).

Un peu d'histoire

La première unité de troc admise dans la Grèce pré-hellénistique fut le bœuf. Au VIII^e^ siècle av. J.-C., dans l'*Iliade* d'Homère (XXIII, 705, 749-751 et au VI, 236), *une femme habile à mille travaux* est ainsi évaluée à 4 bœufs, l'armure en bronze de Glaucos à 9 bœufs et celle de Diomède (en or) à 100 bœufs ; de plus, dans une liste de récompenses, on voit se succéder, dans l'ordre des valeurs décroissantes, une coupe d'argent ciselé, un bœuf et un demi-talent d'or.

Et ce n'est pas le hasard si le mot latin *pecunia* dont dérivent nos termes *pécule* et *pécuniaire* veut dire « fortune, monnaie, argent » : il provient en effet de *pecus*, qui signifie « bétail, troupeau ». En outre, le sens propre du mot *pecunia* correspond à l'« avoir en bœufs ». Le mot sanscrit *rupa* (d'où vient la roupie), comme les termes germains *feo* et *vieh* (auxquels est apparenté le mot anglais *fee*, « salaire »), constituent de même un souvenir du temps où les propriétés, les honoraires, les offrandes, voire les sacrifices rituels étaient évalués en têtes de bétail. C'est d'ailleurs encore en bœufs que se fait l'évaluation de la dot des jeunes filles dans certaines régions d'Afrique orientale. Et l'on comprend que le latin *capita*, « têtes » (d'où *décapiter*, « couper la tête »), ait donné le mot français capital. … *Histoire Universelle des Chiffres*, p 180, *G. Ifrah*

Les nombres cardinaux ont des modes de représentation qui ont beaucoup varié au cours du temps en partant de simples encoches sur des bâtons, des empreintes sur des tablettes, des nœuds dans des cordelettes pour arriver aux symboles stylisés que constituent nos chiffres dits « arabes ».

Un nombre est une entité abstraite, c'est-à-dire une forme d'information qui, d'une manière ou d'une autre, se doit d'être « concrétisée » pour être mémorisée, traitée ou communiquée. Le même concept prend ainsi, en fonction du contexte d'utilisation, des représentations différentes mais supposées équivalentes. Ainsi, 156, CLVI, « cent cinquante-six » et

   (hiéroglyphe) sont toutes des représentations du même nombre. Ce nombre cardinal est appelé *nombre entier naturel*.

Les nombres ordinaux sont, de leur côté, généralement construits à partir des nombres cardinaux en les suffixant de *ier* ou *ième*. Mais à l'usage, la différence entre le cardinal et l'ordinal n'est pas toujours bien faite dans toutes les langues. Ainsi, nous dirons le « premier juin », mais le lendemain sera couramment désigné par le « 2 juin ». En informatique, le nombre ordinal est représenté comme un nombre cardinal.

Représentation des entiers naturels, les bases

Notre système de représentation décimal des nombres est un système dit positionnel (NPZ, Numération de position munie d'un zéro).

Positionnel et additionnel

Le système est dit *positionnel* par opposition à *additionnel* où chaque chiffre apporte sa contribution dans l'addition qui définit le nombre. Les chiffres égyptiens sont additionnels, les chiffres romains le sont partiellement. Dans un système additionnel, il n'y a pas besoin de zéro.

La représentation de tout nombre s'appuie sur un alphabet réduit de symboles, les chiffres de 0 à 9, et la position d'un chiffre dans cette représentation indique son poids en puissance de 10 (unités, dizaines, centaines, etc.). Le cardinal de l'alphabet, soit 10, est appelé « base » de la représentation.

Le nombre *quatre mille trois cent vingt sept* devrait de fait s'écrire sous la forme $4 \times 1\,000 + 3 \times 100 + 2 \times 10 + 7$, c'est-à-dire : $4 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$, mais se réduit à 4 327 en NPZ. Par contre, toute position doit comporter un chiffre : si un poids, par exemple la centaine, n'est pas présent dans le nombre, il faut faire apparaître la nullité de ce poids en mettant le chiffre 0 au rang des centaines.

Étymologie du mot chiffre

Le mot français *chiffre* vient de l'arabe *sifr* qui exprime l'idée de vide pour nommer le zéro. Quant au mot *zéro*, il vient de *zero* en italien qui serait une contraction de *zefiro*, avec là encore la même racine arabe.

La représentation décimale d'un nombre N est la suite ordonnée de chiffres

$\{a_p a_{p-1} \ldots a_1 a_0\} \quad 0 \leq a_i < 10$

qui est une représentation implicite d'un développement du type :

$N = a_p 10^p + a_{p-1} 10^{p-1} + \ldots + a_0 10^0$

Par exemple, pour le nombre 147 on obtient : $a_0 = 7$, $a_1 = 4$, $a_2 = 1$.

On notera que dans cet exemple, on part déjà du résultat connu puisque l'on écrit 147…

La représentation a également la caractéristique d'être à longueur variable, c'est-à-dire que nous utilisons autant de chiffres que nécessaire : peu de chiffres pour un petit nombre, davantage pour un grand nombre, mais toujours en nombre suffisant. En d'autres termes, il nous est possible d'avoir une précision quasi infinie.

Les bases

En arithmétique, tout nombre peut être représenté dans n'importe quelle base sous forme de somme des puissances de la base retenue.

Ainsi pour une base S donnée, on a :

$$N = a_p S^p + a_{p-1} S^{p-1} + \ldots + a_0 S^0 \quad 0 \leq a_i < S$$

$$N = \sum_{i=0}^{p} a_i S^i$$

Les bases usuelles sont : 2, 8, 10 et la base 16. À chaque base correspond un alphabet avec autant de symboles.

2	0 1	en décimal	$9 + 1 = 10_{10}$
8	0 1 2 3 4 5 6 7	en hexadécimal	$F + 1 = 10_{16}$
10	0 1 2 3 4 5 6 7 8 9	en octal	$7 + 1 = 10_{8}$
16	0 1 2 3 4 5 6 7 8 9 A B C D E F	en binaire	$1 + 1 = 10_{2}$

Dans le cas d'une base de valeur inférieure à 10, les chiffres de la base 10 sont repris dans l'ordre et en quantité suffisante. Par contre, pour la base 16, les 10 chiffres sont complétés avec les six premières lettres de notre alphabet, de A à F, en majuscules ou minuscules (figure 1-2).

Figure 1-2
Entiers en décimal, binaire, octal et hexadécimal

0	0	0	0		**16**	10000	20	10
1	1	1	1		**17**	10001	21	11
2	10	2	2		**18**	10010	22	12
3	11	3	3		**19**	10011	23	13
4	100	4	4		**20**	10100	24	14
5	101	5	5		**21**	10101	25	15
6	110	6	6		**22**	10110	26	16
7	111	7	7		**23**	10111	27	17
8	1000	10	8		**24**	11000	30	18
9	1001	11	9		**25**	11001	31	19
10	1010	12	A		**26**	11010	32	1A
11	1011	13	B		**27**	11011	33	1B
12	1100	14	C		**28**	11100	34	1C
13	1101	15	D		**29**	11101	35	1D
14	1110	16	E		**30**	11110	36	1E
15	1111	17	F		**31**	11111	37	1F

Le changement de base

Mathématiquement toutes les bases sont équivalentes et les changements de base sont parfaitement valides. Le changement de base le plus fréquent, en informatique, est le passage de la base 10 vers la base 2 (et réciproquement).

L'algorithme de changement de base, pour une base S cible donnée, consiste à extraire itérativement les coefficients b_i de la nouvelle base S en prenant le reste de la division du nombre N par S. Le processus est réitéré sur le quotient résultant jusqu'à l'obtention d'un quotient nul. N est exprimé dans sa base d'origine, la division est faite dans la base d'origine et le reste est représenté dans la base de destination.

Ainsi, le reste de la division de N par S donne l'unité b_0, le reste de la division du quotient obtenu par S donne l'élément de rang 1 b_1 et ainsi de suite… L'algorithme est valable pour tout changement de base : il suffit de faire les divisions dans la base de départ (il faudra cependant réapprendre à faire les divisions dans cette base…).

Dans le cas d'une conversion de base 10 vers une base 2, nous obtenons par exemple :

$26_{10} = ?_2$	$26_{10}/2$	$= 13_{10}$	reste = $\mathbf{0_2}$
	$13_{10}/2$	$= 6_{10}$	reste = $\mathbf{1_2}$
	$6_{10}/2$	$= 3_{10}$	reste = $\mathbf{0_2}$
	$3_{10}/2$	$= 1_{10}$	reste = $\mathbf{1_2}$
d'où : $26_{10} = 11010_2$	$1_{10}/2$	$= 0_{10}$	reste = $\mathbf{1_2}$

Utilisation de la base 16 (hexadécimal)

L'ordinateur travaille en base 2 et l'homme en base 10. Lorsqu'il s'agit de visualiser une longue suite binaire, le passage à la base 10 n'est pas forcément le plus utile pour l'homme. Par contre, la représentation en base 16 peut, elle, se révéler efficace. En effet le passage de la base 2 à la base 16 (et réciproquement) est particulièrement aisé avec un codage par bloc : tout quartet (suite de quatre bits) peut être interprété comme un nombre entre 0 et 15, soit entre 0 et F en base 16. La lecture résultante est plus « aisée » pour nous.

1010 0111_2 devient $\mathbf{A}7_h$ et $FA59_h$ devient 1111 1010 0101 1001 en binaire.

L'hexadécimal est ainsi fréquemment utilisé pour visualiser des contenus binaires.

Représentation machine

En machine, sauf cas particuliers, la représentation des nombres est faite avec un nombre fixe de symboles binaires (donc de bits). L'ensemble des *symboles* utilisés dans un codage constitue un *alphabet* (0 et 1 dans le cas binaire). Un assemblage de symboles constitue un *mot du code* ou *codet*. La liste des mots de code autorisés constitue le *code*.

Dans nos langues naturelles, les mots ont une longueur variable. L'évolution normale d'une langue vivante a tendance à faire bouger cette longueur. Lorsqu'un mot apparaît pour la première fois, il est souvent long car construit comme une périphrase.

Loi de Zipf

Cette constatation est à mettre en parallèle avec la loi de Zipf. Dans les années 1930, Georges K. Zipf observe qu'en classant les mots d'un texte, quelle que soit sa langue, par fréquence d'utilisation décroissante, cette fréquence est inversement proportionnelle à son rang. Par exemple si le mot le plus fréquent apparaît 100 fois, de deuxième mot le plus fréquent apparaît 50 fois, etc. Prenons l'exemple du mot automobile. Au fur et à mesure du plus grand usage du mot, l'efficacité tend à le raccourcir, par exemple en *auto*. Les mots les plus fréquents vont donc naturellement raccourcir et les mots utilisés plus rarement auront tendance à rester long. Il en est de même pour *chemin de fer métropolitain* devenu *métro, machine à laver le linge* devenu *lave-linge*. Les sigles (SNCF) rentrent également dans ce cadre.

Pour la machine à l'intelligence très rudimentaire qu'est l'ordinateur, l'efficacité s'obtient par la standardisation et la régularité dans les traitements. Un ordinateur travaille mieux sur des mots de longueur (taille) fixe. Cette taille a eu du mal à se figer, au cours du temps, sur une valeur bien déterminée, mais actuellement elle est calée sur des multiples de l'octet : les mots sont de 8, 16, ou 32 bits et maintenant 64 bits. On parle alors de processeur 8 bits, 32 bits ou 64 bits. Il existe aussi des processeurs simples à 4 bits. Le mot *bit* a ici bien sûr le sens du contenant.

La conséquence d'une *taille fixe* en nombre de bits du mot est qu'un mot de ce code ne peut représenter qu'un nombre *fini* d'entiers positifs. Cette limitation a un certain nombre de conséquences, non seulement sur l'étendue des entiers représentables, mais aussi sur les calculs intermédiaires. Pour illustrer le propos, prenons une calculette décimale à affichage de 3 chiffres. Les nombres représentables sont donc les entiers positifs N avec $0 \leq N \leq 999$. La limitation impose aussi que le résultat final d'une opération, ou un calcul intermédiaire, ne sorte pas de cette limite. S'il est possible de calculer 105 – 15, le résultat de 15 – 105 sera faux car il n'y a pas de place pour mettre le signe. De plus, les axiomes d'associativité et de commutativité de l'addition ne sont plus forcément vérifiés.

Par exemple : $700 + (400 - 300) \neq (700 + 400) - 300$. Dans le premier cas, le résultat est juste, dans le second il est faux car il donne lieu à un débordement (700 + 400) au niveau du calcul intermédiaire.

Représentation des entiers naturels N

Les entiers positifs sont codés en binaire avec leur représentation positionnelle en base 2. Tout nombre entier positif est ainsi défini comme une somme de puissances de 2. Le codage est fait en opérant la décomposition en puissance de 2 à l'aide de l'algorithme des divisions successives par 2 vu précédemment comme dans l'exemple du changement de base 10 vers la base 2.

Si la représentation s'appuie sur un mot de 8 bits, alors ce sont les nombres de 0 à 255 qui sont représentables. Aucun nombre en dehors de cette plage de valeur ne pourra être représenté sur un octet. Plus généralement, un mot de n bits permet de coder les nombres entiers naturels de 0 à 2^{n-1}.

Un processeur travaille généralement avec une longueur de mot fixée, mais en fonction de l'étendue des entiers naturels sur lesquels on veut travailler, il existe des artifices pour travailler avec des mots plus longs. Les langages de programmation de haut niveau (C, Java, Fortran…) permettent cette adaptation, indépendamment du type de processeur, en introduisant les types de variables avec différentes longueurs (entier court, long…).

Représentation des entiers relatifs

Les entiers relatifs, ou entiers signés, sont les entiers positifs et négatifs. Si la représentation des entiers naturels est unique de par l'utilisation de la base 2 (unicité de la représentation positionnelle), il y a par contre plusieurs manières de coder les entiers signés.

Les codages diffèrent par la manière de « partager » les possibilités de toutes les configurations des bits du mot entre les nombres positifs et négatifs et la manière de gérer le signe. Par exemple, un des choix consiste à représenter autant de nombres positifs que de négatifs dans l'étendue des valeurs possibles.

Le codage avec bit de signe (signe et valeur absolue)

Ce codage est le plus proche de notre système décimal. Le nombre est décrit par deux entités ou *champs* différentes : le signe et le nombre positif qui correspond à la valeur absolue de l'entier signé. Il y a donc deux éléments à coder : le signe d'une part – un bit est suffisant –, la valeur absolue d'autre part. En reprenant un codage sur 8 bits, il faut donc réserver un bit pour le signe, et il n'en reste plus que 7 de disponibles pour coder l'entier naturel de la valeur absolue.

On voit apparaître la notion de *champ* dans la représentation : le mot est structuré en champs à l'intérieur duquel chaque suite de bits code un élément différent.

La manière la plus naturelle de coder le signe est de coder le signe + par un 0 : il n'y a ainsi pas de différence, à l'étendue des valeurs près, entre le codage d'un entier naturel et un entier relatif positif. Cette représentation est dite avec *bit de signe* : le signe et la valeur absolue sont codés séparément. On peut remarquer que ce codage fait apparaître deux représentations du zéro. Il y a par conséquent autant de nombres positifs que de négatifs.

Exemple : codage de 26 et – 26 sur 8 bits :

26 est codé par **0**001 1010 et

– 26 est codé par **1**001 1010.

Si l'on fait l'addition binaire de 26 avec – 26 en binaire avec retenue on obtient :

```
     1 1   1 (retenues)
   0001 1010
+  1001 1010
-------------
=  1011 0000   ce qui vaut – 52 dans ce codage… et non 0 !
```

On ne peut donc pas appliquer les règles de l'addition binaire à ce type de représentation. Le codage avec bit de signe est simple, mais ce n'est pas celui qui rend le plus efficace le calcul en machine.

Le codage en complément à 2

Le principe du codage dit *en complément à 2* consiste à ramener le cas des nombres négatifs à un nombre positif et rendre applicable directement le codage, maintenant connu, des entiers naturels.

Pour un code sur n bits, la règle de codage est :

- Si le nombre q à coder est positif, on prend son code en base 2 classique.
- Si le nombre q est négatif, on prend le code en base 2 classique de : $q + 2^n$.

Les nombres représentables sont ceux compris entre -2^{n-1} et $2^{n-1} - 1$.

L'appellation *complément à 2* est abusive : de fait, il faudrait dire « complément à 2 puissance n ». Par exemple, si la représentation est faite sur 8 bits, alors il s'agit du complément à 256.

Ainsi le codage de – 26 donne, en complément à 2, la valeur – 26 + 256, c'est-à-dire 230. L'entier négatif – 26 est codé dans cette technique comme s'il s'agissait de l'entier positif 230. Le plus grand entier positif représentable est $2^{8-1} - 1$, soit 127. *Tous les codes à valeur entre 128 et 255 représentent donc des entiers négatifs.*

La décomposition en puissance de 2 de 230 donne 1110 0110.

D'où -26_{10} = 1110 0110 en complément à deux.

Notons que, cette fois-ci, si l'on ajoute – 26 à 26 le résultat obtenu est bien celui attendu avec les règles de l'addition binaire.

```
  1111  11 (retenues)
  0001 1010
+ 1110 0110
= 0000 0000   ce qui vaut 0 dans ce code.
```

Il faut aussi remarquer que la retenue finale (à gauche) est perdue : il n'y a pas de bit pour la représenter.

Représentations particulières et valeurs limites

Il résulte de cette méthode de codage que, pour les nombres négatifs, le bit de poids fort est toujours à 1 (mais à ne pas confondre avec un bit de signe positionné indépendamment du codage de la valeur absolue).

Le zéro n'a qu'une seule représentation : 0000 0000 et – 1 est codé par 1111 1111. L'intervalle de représentation n'est pas symétrique : sur 8 bits, il va de – 128 (10000000) à + 127 (01111111).

Il faut bien faire attention aux « effets de bords » :

maximum + 1 = minimum !!! (127 + 1 = – 128)

Technique de codage des nombres en complément à 2

Nous venons de décrire le principe du codage, voyons maintenant la technique utilisée pour le réaliser. En fait, plutôt que de procéder par ajout de la puissance de 2, il existe un moyen très simple de calculer le complément à 2 d'un nombre. Il s'agit du passage par le complément à 1 auquel on ajoute ensuite la valeur 1.

La règle est la suivante :

- Si le nombre est positif et inférieur à $2^{n-1} - 1$, il est codé comme un entier naturel (décomposition en puissance de 2).
- Si le nombre est négatif et supérieur à -2^{n-1}, alors :
  - On code la valeur absolue du nombre comme un entier naturel (décomposition en puissance de 2).
  - On prend le complément à 1 de ce codage : cela revient à « inverser » tous les bits de la représentation (le complément à 1 de 1 est 0, le complément à 1 de 0 est 1).
  - On ajoute 1 au résultat obtenu (addition binaire).

Exemple : codage de – 26 :

26 =	0001 1010	est le codage de la valeur absolue de – 26.
	1110 0101	est le complément à 1 (inversion de chacun des bits).
+	1	est l'ajout de 1 au résultat obtenu (addition binaire).
soit	1110 0110	ce qui est bien la valeur codée de – 26 trouvée précédemment.

La transposition au système binaire est immédiate : le complément à 10 devient le complément à 2 et le complément à 9 devient celui à 1. La particularité réside dans le fait que ce complément à 1 est trivial à obtenir, il suffit d'inverser le bit.

À titre d'exemple, l'opération précédente peut être revue en binaire avec l'utilisation du complément à 2 sur 10 bits.

$A = 713_{10} = 10\ 1100\ 1001_2$

$B = 418_{10} = 01\ 1010\ 0010_2$

$B_1^* = 10\ 0101\ 1101$, est le complément à 1

$B_2^* = 10\ 0101\ 1110$, est le complément à 2 ($= B_1^* + 1$)

A – B : $10\ 1100\ 1001_2$

$+ \ 10\ 0101\ 1110_2$

$= 01\ 0010\ 0111_2$: le bit de retenue à gauche est perdu.

Le résultat correspond bien à 295 en décimal.

Un petit retour sur la soustraction décimale

Pour mieux comprendre les fondements de cet algorithme, il est bon de revenir à notre système décimal et de revisiter quelque peu la soustraction.

Dans le système décimal, les entiers naturels sont codés par les poids des puissances de 10 qui interviennent dans le nombre. Avant d'en venir à la soustraction et les nombres négatifs, revenons rapidement sur l'addition. Celle-ci consiste à sommer chacun des poids de même rang des 2 nombres en partant de la droite (les unités) et en effectuant des reports sur le poids suivant (supérieur) si nécessaire. Que faut-il connaître pour faire une addition élémentaire de 2 entiers naturels à 1 chiffre ? Tout simplement la table d'addition des nombres à 1 chiffre.

Pour introduire les nombres négatifs, nous allons procéder en deux temps en utilisant ce qui vient d'être appris pour l'addition de deux entiers positifs. Pour ôter une valeur à un entier, on va soustraire un entier positif à ce nombre. Comme on ne sait pas faire la soustraction, on va se ramener au cas connu de l'addition. Soustraire un nombre positif revient à additionner un nombre négatif. Donc, si on sait faire la soustraction par l'intermédiaire d'une addition, on aura aussi résolu la question de la représentation de l'entier négatif.

Ainsi, pour retrancher B à A il suffit de lui ajouter un nombre suffisamment bien choisi. La technique utilisée est celle du complément à 10 (avec le même abus de langage que pour le complément à 2 : il faudrait dire « complément à la puissance de 10 »). Illustrons le propos avec le cas de la calculette à 3 digits (chiffres décimaux). Le complément à 10 est alors le complément à 10^3 c'est-à-dire le complément à 1000.

Soit à faire le calcul de C = A – B avec A = 713 et B = 418.

Pour retrancher B à A on lui ajoute B10*, son complément à 10.

B10* = 10^3 – B c'est-à-dire (1000 – 418 = 582) d'où :

C = A – B = A + (10^3 – B) – 10^3

= A + B10* – 1000

(= 713 + 582 – 1000 = 1295 – 1000 = 295)

Quel est l'intérêt de cette opération ? On vient de remplacer une soustraction de deux entiers quelconques par une addition et une soustraction particulière avec 1000 (le complément à 10). C'est toujours une soustraction, mais elle est plus simple à réaliser car elle se fait par rapport à une valeur fixe. Le nouveau problème est : que faut-il ajouter à 418 pour obtenir 1000, au lieu de la question : que faut-il ajouter à 418 pour que le résultat fasse 713 ? La connaissance de la table des additions à un chiffre suffit pour calculer le complément à 10. Nous avons transformé la soustraction en addition, mais il reste la quantité 10^3 (1000) qu'il faut retrancher : comme on ne dispose que de 3 chiffres sur notre calculette, il suffit donc de l'ignorer.

Plutôt que de calculer directement le complément à 10 (B10*), il est plus facile de le définir par rapport au complément à 9, B9* (soit 999), par B10* = B9* + 1. Or le complément à 9 se calcule directement à partir des tables apprises pour l'addition : il suffit pour un nombre à 1 chiffre donné de connaître le nombre qui doit lui être ajouté pour que le résultat soit égal à 9. Cette opération est répétée pour tous les chiffres du nombre.

Le complément à 9 de 418 est déterminé par : 8 + 1 = 9 ; 1 + 8 = 9 ; 4 + 5 = 9.

Ce qui donne B9* = 581, et finalement B10* = B9* + 1 = 581 + 1 = 582 et, pour l'addition finale, nous obtenons : 713 + 582 = 1295.

Pour retrancher 1000, il suffit d'ignorer le quatrième chiffre.

La soustraction devient l'addition de l'opposé codé en complément à 10. On peut donc partir sur cette base pour la représentation d'un nombre entier négatif : il suffit de le coder sous la forme de complément à 10.