I Informations numériques

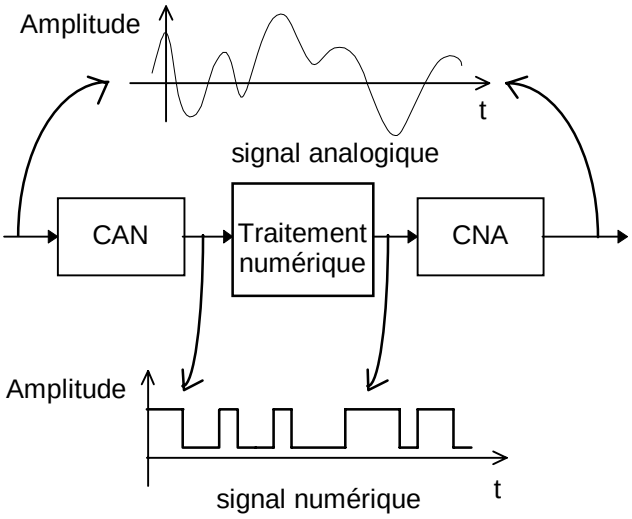
I.1. De l'analogique au numérique

Entre un disque « noir » et un disque « compact » il y a une différence de principe : le premier est *analogique*, le second *numérique*. Que signifient ces termes ?

- Le mot analogique évoque ressemblance, si on regarde avec un microscope une partie de sillon d'un disque noir on verra une sorte de vallée sinueuse dont les flancs reproduisent, à peu près, la forme des signaux électriques transmis aux haut-parleurs. A un son grave correspondent des sinuosités qui ont une période spatiale grande (quelques mm pour 100 Hz), à un son aigu correspondent des sinuosités dont la période spatiale est plus petite (quelques centièmes de mm pour 10 kHz). De même, l'amplitude des sinuosités reproduit, grosso-modo, l'amplitude du son que l'on souhaite reproduire.
- Le mot numérique évoque nombre. Si on regarde au microscope (grossissement supérieur à 100) une plage d'un disque compact on verra une sorte de pointillé de trous ovales, presque identiques, répartis de façon irrégulière sur des pistes quasi-circulaires. Aucun rapport de forme entre le son enregistré et l'allure de la gravure ne peut être observé, présence ou absence de trous constituent les deux valeurs possibles d'un chiffre en base 2. Ces chiffres, regroupés par paquets de 16, constituent des nombres entiers dont la valeur est l'image, via un code, de l'amplitude du signal sonore[1].

[1]En réalité le code est un peu plus compliqué que ne le laisserait supposer cette description, mais c'est une autre histoire.

Copyright (c) 2007, J. Weber et M. Meaudre. Le contenu de ce document peut être redistribué sous les conditions énoncées dans la Licence pour Documents Libres version 1.1 ou ultérieure.



Le passage d'un monde à l'autre se fait par des *convertisseurs analogique-numérique* (CAN) et *numérique-analogique* (CNA), dont nous n'étudierons pas ici le fonctionnement. La différence de principe évoquée plus haut se retrouve évidemment quand on observe le fonctionnement des circuits : un circuit analogique manipule des signaux électriques qui peuvent prendre une infinité de valeurs, qui sont en général des fonctions continues du temps, un circuit numérique manipule des signaux qui ne peuvent prendre qu'un nombre fini (généralement 2) de valeurs conventionnelles, sans rapport avec le contenu de l'information, qui sont des fonctions discontinues du temps.

I.2. Deux niveaux électriques : le bit

Dans toute la suite nous considérerons, ce qui est le cas le plus fréquent, que les signaux numériques représentent des valeurs binaires ; ils ne peuvent prendre que deux valeurs. Une variable binaire porte le nom de *bit*, contraction de binary digit, littéralement chiffre en base 2. Dans un circuit électronique la grandeur physique significative que l'on utilise le plus souvent est la tension (un signal électrique peut très bien être un courant), sauf précision contraire explicite la « valeur » d'un signal électrique binaire se mesurera donc en volts.

Dans un système numérique tous les potentiels sont mesurés par rapport à un potentiel de référence, la masse, qui est une équipotentielle commune à tous les circuits. Cette précision permet de parler du potentiel (ou de la tension) d'un point d'un montage au lieu de spécifier « différence de potentiels entre ... et la masse ». A chaque équipotentielle d'un circuit on peut donc associer un bit qui représente la valeur de la tension de l'équipotentielle considérée.

I.2.1 Conventions logiques

Une entrée ou une sortie d'un circuit numérique ne peut prendre que deux valeurs, notées généralement ***H***, pour High (haut), et ***L***, pour Low (bas) : 3 et 0,2 volts sont des valeurs typiques fréquemment rencontrées. La valeur d'un signal représente en général quelque chose :

- chiffre en base deux, 0 ou 1,
- valeur d'une variable logique, vrai ou faux,
- état d'un opérateur, actif ou inactif,
- état d'une porte, ouverte ou fermée,
- état d'un moteur, arrêté ou en marche,
- etc.

L'association entre la valeur électrique (H ou L) et le sens que l'on donne à cette valeur (0 ou 1, par exemple) constitue ce que l'on appelle une *convention logique*. Il n'y a pas de convention par défaut, cet oubli peut être une source d'erreurs. Dans la famille des microprocesseurs 68xx0 les adresses (des nombres entiers) sont matérialisées par des tensions où ***H*** est associé au 1 binaire et ***L*** au 0 binaire, mais les niveaux d'interruptions (également des nombres entiers) sont matérialisés par des tensions où ***H*** est associé au 0 binaire et ***L*** au 1 binaire. C'est comme çà.

Traditionnellement on qualifie de convention logique *positive* l'association entre ***H*** et 1, ou vrai, ou actif, et de convention logique *négative* l'association entre ***H*** et 0, ou faux, ou inactif. Dans le cas de la porte, qui doit, comme chacun sait, être « ouverte ou fermée », il n'y a pas de tradition. Le circuit 74xx08, par exemple, est connu comme étant une « positive logic and gate », littéralement « porte et en logique positive ». Si l'on change de convention logique le même circuit devient un opérateur Booléen différent (lequel ?).

I.2.2 Immunité au bruit

L'un des intérêts majeurs des signaux numériques est leur grande robustesse vis à vis des perturbations extérieures. L'exemple des enregistrements sonores en est une bonne illustration, le lecteur sceptique n'aura qu'à faire la simple expérience qui consiste à prendre une épingle (fine), à rayer un disque « noir », un disque compact, et à comparer les résultats. Derrière le résultat de cette expérience, quelque peu agressive, se cachent en fait deux mécanismes qui se complètent pour rendre le système numérique plus robuste, une protection au niveau du signal élémentaire, le bit, et une protection au niveau du système par le jeu du codage :

Au niveau du bit

La protection repose sur le fait que l'information n'est pas contenue dans l'amplitude du signal. Un signal analogique direct (on exclut ici les signaux modulés pour lesquels l'analyse devrait être affinée) a une forme qui est l'image de l'information à transmettre. Toute perturbation à cette forme se traduit par une déformation de l'information associée. L'amplitude d'un signal numérique n'a qu'un rapport très lointain avec l'information véhiculée, la seule contrainte est que le système soit encore capable de différencier sans ambiguïté un niveau haut et un

niveau bas. L'écart entre ces deux niveaux étant grand, seule une perturbation de grande amplitude pourra provoquer une erreur de décision[2].

Au niveau du système

Les valeurs élémentaires (bits) sont regroupées en paquets pour former des *mots*, ces mots doivent obéir à certaines règles de construction, des *codes*. Il est parfaitement imaginable, et c'est ce qui est fait dans tous les cas où l'on craint les perturbations, de construire des codes qui permettent de détecter et, dans une certaine mesure, de corriger des erreurs. Un exemple connu de tous, certes assez éloigné de l'électronique numérique, est la langue écrite. Un lecteur qui n'est pas totalement illettré est à même de détecter et de corriger un grand nombre d'erreurs typographiques, même sans faire appel au sens, d'un texte. La raison en est simple : les mots du dictionnaire sont loin de contenir toutes les combinaisons possibles des 26 lettres de l'alphabet, la construction d'une phrase obéit à des règles de grammaire bien connues du lecteur. Ces restrictions (mots du dictionnaire et grammaire) introduisent des redondances qui permettent justement d'assurer la robustesse du texte vis à vis des erreurs typographiques. Les codes détecteurs et/ou correcteurs d'erreurs sont tous fondés sur l'adjonction de redondances à la chaîne de bits transmis, ou inscrits sur un support fragile.[3]

Les choses ont un coût

Sans rentrer ici dans les détails, on peut remarquer que la robustesse des signaux numériques est liée à la très faible quantité d'information véhiculée par chaque signal élémentaire (0 ou 1). Le corrolaire de cette pauvreté du signal élémentaire est que pour traiter une information complexe il faut une quantité énorme de signaux élémentaires, merci M. de La Palice. Cela se traduit par un débit, dans le cas des transmissions, ou par un volume, dans le cas du stockage, très important. A titre d'exercice on calculera le nombre d'*octets* (paquets de 8 bits, *byte* dans le jargon) contenus dans un disque compact qui dure une heure, sachant que le signal est numérisé (CAN) 44 000 fois par seconde, et que chaque point occupe 2*16 bits (sans les redondances du code correcteur !). A titre de comparaison, le micro-ordinateur qui sert à rédiger ce texte dispose d'une mémoire de 8 Méga Octets, et d'un disque de 200 Méga Octets. On répondra ensuite à la question : pourquoi la télévision numérique ne sera-t-elle généralisée qu'au siècle prochain (il est vrai que c'est demain), au prix d'un investissement mathématique considérable pour « comprimer » l'information vidéo.[4]

---

[2]Cette question sera précisée quantitativement lors de la présentation des technologies des circuits (chapitre II).

[3]Un exemple simple de code correcteur sera étudié en exercice, quand les outils d'analyse seront disponibles.

[4]Une image télévision se répète au minimum 25 fois par seconde, contient 600 lignes de 700 points, chaque point est numérisé sur 2 octets.

I.3. Du bit au mot : des codes

Nous avons déjà évoqué que les informations élémentaires, que constituent les bits, sont souvent regroupées dans des paquets plus riches de sens, interprétables dans un code. Le monde du codage est vaste, nous nous contenterons ici de décrire rapidement quelques codes élémentaires, d'usage quotidien, et laisserons à l'initiative du lecteur l'exploration des codes sophistiqués, notamment les codes correcteurs d'erreurs.

I.3.1 Pour les nombres

Deux grandes catégories de nombres existent dans le monde informatico-logique : les entiers et les flottants. Les premiers constituent un sous ensemble *fini* de l'ensemble des entiers cher aux mathématiciens, les seconds tentent d'approcher, par un ensemble fini, les nombres réels. Le distinguo est de taille, les entiers que nous rencontrerons obéissent à une arithmétique euclidienne clairement définie, telle qu'on l'apprend dans les grandes classes de l'école primaire, les seconds obéissent à une arithmétique *approchée*, même si l'approximation est bonne.

Le lecteur averti pourra objecter que l'on rencontre parfois des nombres non entiers, caractérisés par un nombre connu de chiffres après la virgule (un exemple de tels nombres est rencontré sur vos relevés bancaires). Cette catégorie de nombres, connue sous le nom de *virgule fixe*, n'en est pas une : ils obéïssent à l'arithmétique entière (faites vos comptes en centimes), et sont convertis lors des opérations d'entrée-sortie (affichage, impression, saisie clavier).

Entiers naturels

La base 2

Etant donné un mot de n bits (a_{n-1},a_{n-2}, ... a_1,a_0), où les a_i valent 0 ou 1, on peut considérer que ce mot représente la valeur d'un entier A, écrit en base 2 :

$$A = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_1 2 + a_0$$

Les valeurs de A sont limitées par : $0 \leq A \leq 2^n - 1$

Les valeurs couramment rencontrées pour n sont 8 (octet), 16 (entier court) et 32 (entier long). Les bornes supérieurs correspondantes pour la valeur de A sont respectivement de 255, 65 535 et 4 294 967 295.

Quand n est un multiple de 4 (c'est le cas des valeurs évoquées ci-dessus...) il est souvent pratique, car plus compact, d'écrire le nombre en hexadécimal (base 16) :

$$A = (h_{m-1},h_{m-2}, ...,h_0) = h_{m-1}16^{m-1} + h_{m-2}16^{m-2} + ...+ h_0$$

Où m = n/4, et h_i peut prendre l'une des 16 valeurs 0, 1, 2 ....,9, A, B, C, D, F qui sont elle-mêmes représentables en binaire sur 4 bits.

Les opérations arithmétiques classiques sont l'addition, la soustraction, la multiplication et la division. On notera que :

1. Tous les résultats sont obtenus *modulo* 2^n, ce qui confère aux opérations sur l'ensemble des entiers sur n bits un caractère périodique comparable à celui des fonctions trigonométriques.
2. La division est la division entière, dont le résultat est constitué de deux nombres : le quotient et le reste.

La base 10

Les humains actuels ont pris la déplorable habitude de compter en base dix[5] qui n'est pas une puissance de 2 (60 non plus, d'ailleurs). Il n'y a donc pas de correspondance simple entre un nombre écrit en binaire et sa version décimale. Si cette difficulté est la source d'exercices élémentaires de programmation (programmes de changements de bases), elle est parfois gênante en pratique. C'est pour cette raison que l'on rencontre parfois des codes hybrides : le nombre est écrit en *chiffres décimaux*, et chaque chiffre est codé en binaire sur 4 bits. Le code le plus classique, dit *BCD* pour « binary coded decimal » consiste à coder chaque chiffre décimal (0 à 9) en binaire naturel (0000 à 1001). L'arithmétique de ce code n'est pas simple, l'addition de deux chiffres décimaux peut conduire à un résultat hors code (un nombre compris entre 10 et 15), ou faux mais apparemment dans le code (un nombre compris entre 16 et 18) ; après chaque opération élémentaire il faut donc recaler le résultat intermédiaire. Cette opération supplémentaire s'appelle *ajustement décimal*, il faut l'effectuer après tout calcul sur une tranche de 4 bits. Beaucoup de calculettes utilisent un code BCD qui facilite les opérations d'affichage.

D'autres codes décimaux existent, qui facilitent un peu les calculs, mais ils sont d'un usage rarissime actuellement.

Entiers signés

Quand on aborde la question des entiers signés il est essentiel de se souvenir *qu'un mot de n bits ne peut fournir que* 2^n *combinaisons différentes*. Comme on ne peut pas avoir le beurre et l'argent du beurre, il faudra restreindre la plage des valeurs possibles pour la valeur absolue du nombre.

Les êtres humains ont l'habitude de représenter un nombre signé dans un code qui sépare le signe et la valeur absolue du nombre. Le signe étant une grandeur binaire (+ ou -), on peut lui affecter un bit, le *bit de signe*, et garder les n-1 bits restant pour coder la valeur absolue, en binaire naturel, par exemple. Ce type de code, connu sous le nom de « signe-valeur absolue » n'est en fait jamais utilisé pour les nombres entiers (il l'est par contre pour les flottants). La raison en est que l'arithmétique sous-jacente est compliquée ; en effet pour additionner ou soustraire

[5]On notera que vers 3000 avant J.C. les Sumériens avaient fort judicieusement choisi la base 60, multiple de 2, 3, 4, 5, 6, 10, 12. Il est vrai que l'apprentissage des tables de multiplications ne devait pas être à la portée de tous.

deux nombres signés, dans un code signe-valeur absolue, il faut commencer par déterminer quel sera le signe du résultat. Pour ce faire il faut commencer tout calcul par une comparaison qui fait intervenir à la fois les signes et les valeurs absolues des opérandes (remémorez-vous vos débats avec les mathématiques du début du collège).

Les deux codes universellement utilisés pour représenter les entiers relatifs, présentés ci-dessous, évitent cet écueil, additions et soustraction ne sont qu'une même opération, qui ne fait pas intervenir de comparaison, et les reports (ou retenues) éventuels sont simples à traiter.

Dans l'un des codes comme dans l'autre, l'intervalle de définition d'un nombre A, codé sur n bits, est donné par :

$$-2^{n-1} \leq A \leq 2^{n-1} - 1$$

Soit

-128 à 127 pour n = 8,
-32 768 à 32 767 pour n= 16
et -2 147 483 648 à 2 147 483 647 pour n=32.

Attention !

- Le caractère périodique des opérations implique, par exemple, que dans un code signé *sur 8 bits 100 + 100 = -56*, qui est bien égal à 200 modulo 256, il ne s'agit pas là d'une erreur mais d'une conséquence de la restriction à un sous-ensemble fini des opérations sur les entiers.
- Le changement de longueur du code, par exemple le passage de 8 à 16 bits, n'est pas une opération triviale, la combinaison binaire qui représente 200 en binaire naturel, n'a pas forcément la même signification quand on l'interprète dans un code 8 bits ou un code 16 bits.

Le code complément à 2

C'est le code utilisé pour représenter les nombres entiers dans un ordinateur. Il présente l'intérêt majeur de se prêter à une arithmétique simple, mais a pour défaut mineur que la représentation des nombres négatifs n'est pas très parlante pour un être humain normalement constitué. La construction du code complément à deux, *sur n bits*, découle directement de la définition modulo 2^n des nombres. Etant donné un nombre A :

⇒ **Si $A \geq 0$** le code de A est l'écriture en *binaire naturel de A*, éventuellement complété à gauche par des 0.
Exemple : A = 23, codé sur 8 bits s'écrit : 00010111.

⇒ **Si $A < 0$** le code de A est l'écriture en *binaire naturel de* $2^n + A$, ou, ce qui revient au même, de $2^n - |A|$.
Exemple : A = – 23, codé sur 8 bits s'écrit : 11101001, qui est la représentation en binaire naturel de 256 – 23 = 233 (E9 en hexadécimal).

On remarquera que le bit le plus à gauche, le bit de poids fort ou *MSB* (pour *most significant bit*), est le bit de signe, avec la convention logique 1 pour – et 0 pour +.

Le calcul de l'opposé d'un nombre, quel que soit le signe de ce nombre, est une simple conséquence de la définition du code : $-A = 2^n - A$ modulo 2^n.

Par exemple :

$$-(-23) = 256 + 23 \text{ modulo } 256 = 23.$$

Astuce de calcul : Pour obtenir rapidement l'expression binaire de l'opposé d'un nombre dont on connait le code, on peut utiliser l'astuce suivante (que certains, à tort, prennent pour une définition du code) :

$2^n - A = 2^n - 1 - A + 1$ (vérifiez)

$2^n - 1$ est le nombre dont tous les chiffres binaires sont à 1,

$2^n - 1 - A$ est le nombre que l'on obtient en remplaçant dans le code de A les 1 par 0 et réciproquement.

Ce nouveau nombre s'appelle *complément à 1* ou complément *restreint* de A, et se note classiquement $\overline{A}$.

Suivant la tradition on peut alors écrire :

$$-A = \overline{A} + 1.$$

Exemple :

$23 = 00010111$,

$\overline{23} = 11101000$,

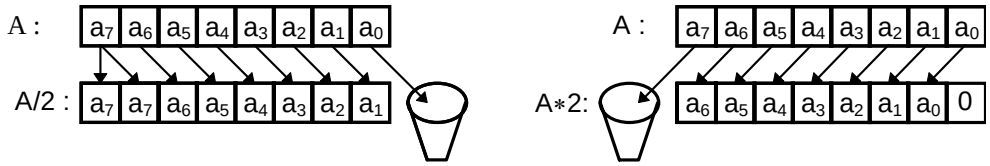
$-23 = \overline{23} + 1 = 11101001$,

qui est le résultat précédent.

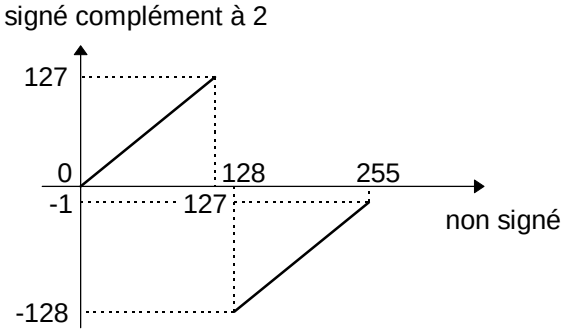
L'augmentation de longueur du code (par exemple le passage de 8 à 16 bits) se fait en complétant à gauche par le bit de signe, 0 pour un nombre positif, 1 pour un nombre négatif. Cette opération porte le nom d'*extension de signe*. Exemple – 23 s'écrit 11101001 sur 8 bits et 1111111111101001 sur 16 bits, ce qui est notablement différent de 0000000011101001 qui est le code de 233, dont l'existence est maintenant légale.

Les additions et soustractions des nombres ne sont qu'une seule et même opération : des additions et des éventuels changements de signes, sans que l'on ait jamais à faire de comparaison, et où les reports (ou retenues) sont générées mécaniquement. La dissymétrie entre addition et soustraction, bien connue des élèves des premières années de collège, a disparu.

Les multiplications et les divisions par des puissances de 2 sont des décalages *arithmétiques* (i.e. avec conservation du signe), par exemple pour des octets :



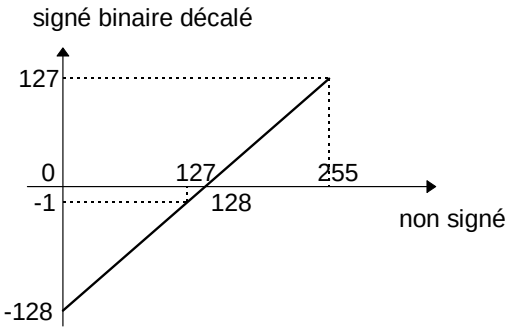
N.B. : Les détails de manipulation des nombres, lors des opérations arithmétiques, sont évidemment transparentes pour le programmeur généraliste, elles sont intéressantes à connaître pour le concepteur d'unités de calcul, et, exceptionnellement, pour le programmeur qui s'occupe des interfaces logiciels avec le matériel (pilotes de périphériques, librairies de bas niveau, etc...).



Relation binaire naturel complément à deux sur 8 bits.

Si le code complément à deux se prête bien aux calculs, il complique les opérations de comparaisons et, plus généralement, les opérations qui font intervenir une relation d'ordre entre les nombres. Précisons cette question de relation d'ordre : le code en complément à 2 de −1, par exemple, correspond à tous les chiffres binaires à 1, qui représente le plus grand nombre possible dans une interprétation non signée. Face à la combinaison binaire correspondant à −1 la réponse à la question « ce nombre est-il supérieur à 10 ? » ne devra pas être traitée de la même façon par un opérateur câblé suivant que le code est signé ou non. La figure ci-dessous tente d'illustrer cette rupture dans la relation d'ordre.

Le code binaire décalé



Relation binaire naturel binaire décalé sur 8 bits.

Le code binaire décalé ne présente pas l'inconvénient évoqué ci-dessus à propos de la relation d'ordre : il possède *la même relation d'ordre que le code binaire naturel* des nombres non signés. On le rencontre dans certains convertisseurs numériques analogiques (ou, plus rarement analogiques numériques) et dans la représentation de l'exposant des nombres flottants. L'application qui fait passer du binaire naturel au

binaire décalé est définie en sorte que le minimum de l'un des codes corresponde au minimum de l'autre, et que le maximum de l'un des codes corresponde au maximum de l'autre :

Un examen rapide de la courbe précédente fournit la formule de génération du code binaire décalé sur n bits : étant donné un nombre entier A tel que $-2^{n-1} \leq A \leq 2^{n-1} - 1$, le code de A est le code binaire naturel de $A + 2^{n-1}$, d'où le nom du code. On peut remarquer que le nombre 2^{n-1} a son MSB égal à 1, tous les autres chiffres étant nuls.

On passe du code binaire décalé au code complément à deux en complémentant le bit de signe.

Flottants

Si A est un nombre flottant il est codé par une expression du type :

$$A = (-1)^s * 2^e * 1,xxxxx...$$

où s est le signe de A, e un nombre *entier signé*, codé en binaire décalé, et xxxxx... la partie fractionnaire de la valeur absolue de la mantisse. A titre d'exemple les flottants double précision (64 bits) suivant la norme ANSI ont les caractéristiques suivantes:

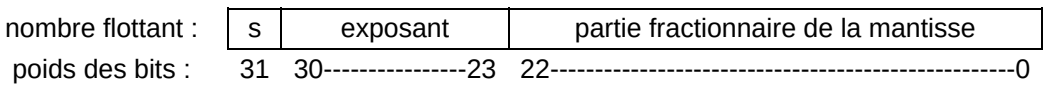
- signe s : 1 élément binaire, 1 pour –, 0 pour +,
- exposant compris entre –1022 et +1023, soit 11 éléments binaires,
- partie fractionnaire de la mantisse sur 52 éléments binaires.
- combinaisons réservées pour le zéro, l'infini (+ et –) et NAN (not a number).

Cela correspond à une dynamique allant de $2,2*10^{-308}$ à $1,8*10^{+308}$, pour une précision de l'ordre de 16 chiffres décimaux.

De ce qui précède il faut tirer deux conclusions importantes :

1. L'arithmétique des nombres flottants et celle des entiers font appel à des algorithmes radicalement différents.
2. Le test d'égalité de deux nombres, qui a un sens clair pour des entiers, fournit un résultat aléatoire dans le cas des flottants. Seule une majoration de leur écart conduit à un résultat prévisible.

Le format binaire des nombres est tel que seule la partie fractionnaire de la mantisse figure, pour des flottants simple précision :



I.3.2 Il n'y a pas que des nombres

Toute catégorie de données un tant soit peu organisée est susceptible de générer un code ; il est évidemment hors de question d'en faire un catalogue un tant soit peu exhaustif. Nous nous contenterons de citer quelques exemples.

Un cas important dans les applications concerne tout ce qui est échange d'informations, par exemple entre deux ordinateurs ou entre un ordinateur et une imprimante, sous forme de texte : cela conduit aux codes alphanumériques. Le code alphanumérique le plus utilisé porte le nom de code ***ASCII*** pour « American standard code for information interchange », décrit ci-dessous.

Le code ASCII

Sept bits pour les caractères anglo-saxons

Pour représenter l'ensemble des lettres de l'alphabet - minuscules et majuscules, sans les accents, les dix chiffres décimaux, les caractères de ponctuation, les parenthèses crochets et autres accolades, les symboles arithmétiques les plus courants et des commandes : 128 combinaisons suffisent. D'où le code ASCII, sur 7 bits ($b_6b_5b_4b_3b_2b_1b_0$), quasi universellement adopté :

				b_6	0	0	0	0	1	1	1	1
				b_5	0	0	1	1	0	0	1	1
				b_4	0	1	0	1	0	1	0	1
b_3	b_2	b_1	b_0	Hex	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	|
1	1	0	1	D	CR	GS	-	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	F	SI	US	/	?	O	_	o	DEL

Les codes de valeurs inférieures à 32 (en décimal) sont des commandes ou des caractères spéciaux utilisés en transmission. Comme commandes on peut citer, à titre d'exemples, CR pour retour chariot, LF pour nouvelle ligne, FF pour nouvelle page, BEL pour « cloche » etc...

Il est clair que le programmeur généraliste qui utilise un langage évolué n'a en fait jamais à connaître les valeurs des codes, ils sont évidemment connus du compilateur, par contre l'auteur d'un pilote d'imprimante peut, lui, avoir à se pencher sur ces choses peu conviviales !

Huit bits pour les accents

Quand on passe aux dialectes régionaux, le français par exemple, un problème se pose : le code ASCII ne connait pas les lettres accentuées. Pour palier ce manque, les auteurs de logiciels et/ou les fabriquants de matériels (imprimantes) ont complété le code, en l'étendant à un octet soit 256 combinaisons différentes. Malheureusement les règles disparaissent quand on passe aux codes caractères sur 8 bits, on peut même trouver des ordinateurs qui, suivant les logiciels, utilisent des combinaisons variables pour représenter les é, è, à, ç, î et autres caractères dialectaux. Nous ne citerons pas de nom.

Un bit de plus pour les erreurs

Quand on craint que des erreurs se produisent au cours d'une transmission (minitel, par exemple), on rajoute parfois un bit supplémentaire (8ème ou 9ème suivant la taille du code initial) on rajoute un chiffre binaire supplémentaire qui est calculé de telle façon que pour chaque caractère transmis :

- Le nombre de bits à un soit impair, on parle alors de parité impaire,

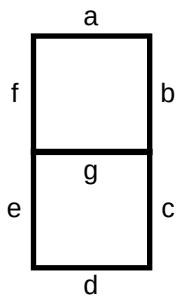
ou :

- le nombre de bits à un transmis soit pair, on parle alors de parité paire.

Si émetteur et récepteur utilisent la même convention de parité, le récepteur est capable de détecter une faute de transmission tant que le nombre de fautes n'excède pas une erreur par caractère. Un procédé aussi rudimentaire ne permet évidemment pas de corriger l'erreur autrement qu'en demandant une retransmission du caractère incriminé.

Autres codes

Concurrent de l'ASCII dans le codage des caractères on peut citer le code EBCDIC (extended binary coded decimal interchange code), encore utilisé dans certains systèmes de gestion.



Pour afficher un chiffre décimal sur un afficheur « 7 segments » il faut associer à chaque chiffre une combinaison qui indique les segments (a, b, c, d, e, f et g) à allumer :

Le passage d'un code à un autre, pour des informations qui ont le même contenu, s'appelle un *transcodage*. A titre d'exemple on peut imaginer comment construire un transcodeur BCD $\rightarrow$ 7 segments.

Exercices

Débordements

A et B sont des nombres entiers, codés en binaire. Pourquoi, même s'il n'y a pas de débordement, l'opération 2*(A/2 + B/2) donne-t-elle parfois un résultat différent de (A + B) ?

Justifiez votre réponse en vous appuyant sur une représentation binaire sur 8 bits. Quelle est la valeur numérique maximum de l'écart entre les deux résultats ?

Nombres entiers signés

Montrer que l'équation

$$x = -x$$

a deux solutions pour des nombres entiers signés, dans le code complément à deux sur n bits.

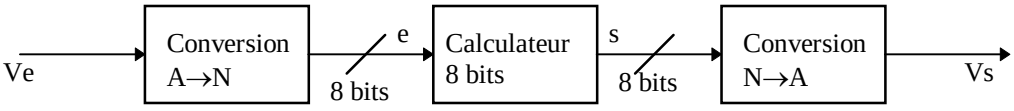
Volumes et débits d'informations

Un enregistrement sur disque compact consiste à prendre une suite d'échantillons (on trace la courbe en pointillé...) à la cadence de 44 kHz. Chaque échantillon est représenté par un nombre de 16 éléments binaires pour chacune des voies droite et gauche (stéréophonie).

- A combien d'octets (paquet de 8 bits) correspond un disque d'une heure de musique enregistrée ?
- A quelle vitesse, en bits par seconde, doit travailler le lecteur pour restituer la musique en temps réel (ce qui est souhaitable) ?
- Sachant que la fréquence maximum des signaux sonores est de 20 kHz, quel est le "coût" du numérique ? Quels sont ses avantages ?
- Reprendre les calculs précédents pour un signal de télévision, de fréquence maximum 5 MHz, codé sur 8 bits (l'oeil est très tolérant) à 2,2 fois la fréquence maximum.
- La télévision numérique peut-elle être une simple transposition des techniques utilisées pour le son ?

Conversions.

Une chaîne de traitement de l'information est constituée comme suit :



Les convertisseurs analogique numérique et numérique analogique ont un pas de 40 mV (écart de tensions analogiques qui correspond à une différence de 1 bit de pods faible). Toute la chaîne est bipolaire : tensions d'entrée et de sortie positive ou négative, nombres entiers signés (sur 8 bits). Quelles sont les tensions maximum et minimum des tensions analogiques d'entrée et de sortie ?

Le calculateur effectue à chaque étape (n) la moyenne des quatre échantillons précédents :

$$s(n) = 1/4*(e(n) + e(n-1) + e(n-2) + e(n-3))$$

1. Un premier programme de calcul est la traduction directe de la formule précédente. En relevant la fonction de transfert Vs = f(Ve), le programmeur s'apperçoit que les résultats en sortie sont parfois étranges. Pourquoi ?
2. Ayant trouvé son erreur, le programmeur utilise cette fois la formule :

$$s(n) = e(n)/4 + e(n-1)/4 + e(n-2)/4 + e(n-3)/4$$

3. Les résultats sont moins absurdes, mais un autre défaut apparaît. Lequel ?
4. Quelle serait la solution du problème ?