Représentation des Informations

1. Introduction

Les informations traitées par un ordinateur peuvent être de différents types (texte, nombres, etc.) mais elles sont toujours représentées et manipulées par l'ordinateur sous forme binaire. Toute information est traitée comme une suite de 0 et de 1. L'unité d'information est le chiffre binaire (0 ou 1), que l'on appelle BIT (*BInary digiT*). Le codage d'une information consiste à établir une correspondance entre la représentation externe (habituelle) de l'information et sa représentation interne dans la machine. Le système de numération binaire est le plus répandu sans négliger l'importance des autres systèmes : décimal, octal et hexadécimale. Dans un système numérique, il peut arriver que trois ou quatre de ces systèmes de numération cohabitent, d'où l'importance de pouvoir convertir un système vers un autre.

2. Représentation des nombres entiers

La représentation (ou codification) des nombres est nécessaire afin de les stocker et les manipuler par l'ordinateur. Le problème principal est la limitation de la taille du codage ; un nombre peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fixe.

2.1. Entiers naturels

Les entiers naturels sont codés sur un nombre d'octets fixe (un octet est un groupe de 8 bits). On rencontre habituellement des codages sur 1, 2 ou 4 octets, plus rarement sur 64 bits. Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et 2^n-1. Par exemple sur 1 octet, on peut coder les nombres de 0 à $255 = 2^8-1$.

2.2. Entiers relatifs

La représentation des entiers relatifs pose un problème au niveau de la représentation du signe. On distingue trois méthodes de codage des entiers relatifs, codage en (1) signe et valeur absolue, (2) complément à un et (3) complément à deux.

2.2.1. Codage signe et valeur absolue

On code le signe en utilisant le bit du poids le plus fort et on code la valeur absolue du nombre. Le bit du signe est mis à 0 s'il s'agit d'un entier positif et à 1 s'il s'agit d'un entier négatif. Un codage sur n bits permet de représenter les entiers relatifs compris entre $-(2^{n-1}-1)$ et $+(2^{n-1}-1)$.

Exemple : Codage de –15 sur 8 bits ⇒ –15 = 10001111.

2.2.2. Codage en complément à un

S'il s'agit d'un entier positif, on représente le nombre en binaire comme pour les entiers naturels. Cependant, le bit du poids le plus fort est mis à 0 (alors, on utilise n–1 bits). Le plus grand entier positif représentable sur n bits est donc $+(2^{n-1}-1)$. Dans le cas d'un entier négatif, on représente le nombre en binaire. Ensuite, on remplace les 0 par des 1 et les 1 par des 0. Le plus petit entier négatif représentable sur n bits est donc $-(2^{n-1}-1)$.

Exemple : Codage de –15 sur 8 bits ⇒ –15 = 11110000.

2.2.3. Codage en complément à deux

S'il s'agit d'un entier positif, on représente le nombre en binaire comme pour les entiers naturels. Le bit du poids le plus fort est mis à 0 (on utilise n–1 bits). Dans le cas d'un entier négatif, on représente le nombre en complément à un, ensuite on additionne 1 au résultat. Sur n bits, le plus petit entier négatif représentable est -2^{n-1} et le plus grand entier positif représentable est $+(2^{n-1}-1)$.

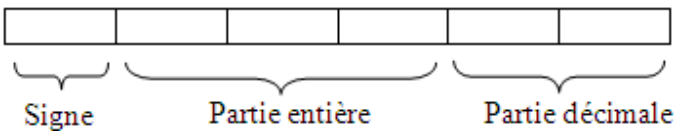
Exemple : Codage de –15 sur 8 bits ⇒ –15 = 11110001.

3. Représentation des nombres réels (fractionnaires)

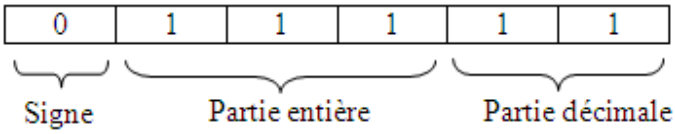
Il y a plusieurs méthodes de représentation des nombres réels. Parmi les plus répandues, on trouve la représentation en virgule fixe et la représentation en virgule flottante.

3.1. Représentation en virgule fixe

On considère un nombre réel représenté sur 6 bits en représentation binaire signée (signe et valeur absolue), tels que : 1 bit pour le signe, 3 bits pour la partie entière et 2 bits pour la partie décimale.

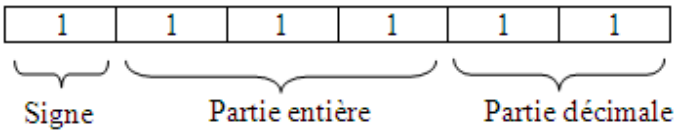


Le plus grand nombre fractionnaire représentable sur 6 bits est :



La plus grande valeur absolue de la partie entière à représenter égale à $(111)_2 = 2^3–1 = 7$. La plus grande valeur absolue de la partie décimale à représenter égale à $(0,11)_2 = 0,75$. Ainsi, le plus grand nombre représentable égale à +7,75.

Le plus petit nombre fractionnaire représentable sur 6 bits est :



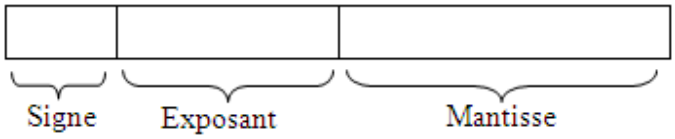
La plus grande valeur absolue de la partie entière à représenter égale à $(111)_2 = 2^3–1 = 7$. La plus grande valeur absolue de la partie décimale à représenter égale à $(0,11)_2 = 0,75$. Ainsi, le plus petit nombre représentable égale à –7,75.

3.2. Représentation en virgule flottante

La virgule flottante est la représentation la plus répandue. Dans ce système, un mot machine est divisé en deux parties : « un exposant » et « une mantisse ». Par exemple, l'exposant de –3 et la mantisse de 1,5 peuvent représenter le nombre $1,5\times10^{-3}$ en décimal. Afin de pouvoir effectuer correctement les opérations arithmétiques sur les nombres à représentation en virgule flottante, il est nécessaire de procéder à leur *normalisation*. On dit qu'un nombre est normalisé si sa partie entière n'est composée que d'un seul chiffre. Par exemple, le nombre $78,5\times10^2$ n'est pas normalisé. Pour le normaliser on doit l'écrire de la forme : $7,85\times10^3$. Un nombre en virgule flottante est représenté dans la machine par une succession de bits décomposée en trois zones : (1) le bit de signe, (2) l'exposant et (3) la mantisse.

Dans les années 80, une norme a été adoptée par la majorité des constructeurs d'ordinateurs, c'est la norme IEEE–754. Cette norme a défini trois formats pour les nombres flottants normalisés : (1) simple précision sur 32 bits, (2) double précision sur 64 bits et (3) précision étendue sur 80 bits. Que l'on soit dans l'une ou l'autre représentation, un certain nombre de conventions ont été prises :

1. L'ordre de la représentation est fait comme suit :



2. La représentation des nombres avec la norme IEEE–754 consiste à s'assurer que la mantisse commence par un seul chiffre à 1 avant la virgule. Ce chiffre est implicite, c'est-à-dire, il n'apparaît pas dans la représentation.
3. Les exposants sont décalés d'une valeur de sorte à éviter d'avoir recours à la représentation en complément à deux des exposants négatifs.

3.2.1. Nombre en simple précision

Le format des nombres en simple précision est comme suit :

- 1 bit pour le signe.
- 8 bits pour l'exposant (la plus grande valeur est 127, la plus petite est –126 et le décalage est de 127).
- 23 bits pour la mantisse.

Exemple : Soit à codifier le nombre +3,25 qui s'écrit en binaire : (11,01). Après la normalisation on obtient : $11,01=1,101\times2^{1}$. On calcule l'exposant décalé qui égale à l'exposant réel plus 127 : 1+127=128=10000000 en binaire sur 8 bits. Selon la norme IEEE 754 simple précision :

- Le signe = 0.
- L'exposant décalé =10000000.
- Mantisse = 10100000000000000000000.

3.2.2. Nombre en double précision

Le format des nombres en double précision est comme suit :

- 1 bit pour le signe.
- 11 bits pour l'exposant (la plus grande valeur est 1023, la plus petite est –1022 et le décalage est de 1023).
- 52 bits pour la mantisse.

3.2.3. Autres formats

En plus des nombres normalisés, trois autres formats de nombres sont définis dans la norme IEEE–754 :

1. Représentation du zéro : deux représentations pour le 0 (+0 et –0).

+ ou –	0	0

2. Représentation de l'infini : $+\infty$ et $-\infty$

+ ou –	11111111	0

3. Représentation d'un nombre indéfini NAN (*Not A Number*)

+ ou –	11111111	toute configuration sauf tous les bits à zéro

4. Les décimaux codés binaire

Les informations traités par ordinateur ne sont pas toujours converties en binaire. En effet, ces informations peuvent être manipulées sous forme décimale codée binaire. Un certain nombre de codes est défini à cet effet.

4.1. Le code BCD

Chaque chiffre d'un nombre est sur codé 4 bits :

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
11	0001 0001
12	0001 0010
13	0001 0011
etc.	

Exemple : $(562)_{10} = (0101\ 0110\ 0010)_{BCD}$

On constate qu'il y a six configurations qui sont non exploitées : 1010, 1011, 1100, 1101, 1110 et 1111. Lors de l'addition de deux nombres écrits suivant le code BCD, si le résultat ne comporte pas de configurations non-autorisées, alors il est correct, sinon il ne l'est pas. Dans ce dernier cas, il suffit de rajouter la valeur 6 (0110) au résultat et cela pour chaque configuration non-autorisée.

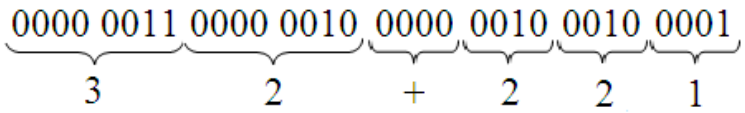
Exemple : en décimal, 15+17=32. En BCD, (0001 0101)+(0001 0111) = (0010 1100). Le résultat comporte une configuration non autorisée (1100). Alors on ajoute au résultat 6 (0110) et on obtient 0011 0010, ce qui donne 32 en décimal.

4.2. Codage DCB des nombres à virgule

Une suite d'octets permet de coder un nombre décimal signé comme suit :

- 1 octet d'entête donne le nombre total *n* de quartets utilisés (ensemble de 4 bits).
- 1 octet spécifie la position de la virgule.
- 1 quartet indique le signe du nombre (0000 : positif, 1111 : négatif).
- *n* quartets représentant les chiffres en DCB.

Exemple : représentation du chiffre 2,21



5. Code excédent 3

Dans ce code, chaque chiffre décimal est codé en son équivalent en binaire additionné de 3. Le tableau suivant donne l'équivalent des chiffres décimaux dans le code excédent 3.

Chiffre décimal	Code excédent 3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Exemple : dans le code excédent 3, le nombre 951 = $(1100\ 1000\ 0100)_{Excédent3}$. 9 = 1001+11 = 1100. 5 = 0101+11 = 1000. 1 = 0001+11 = 0100.

6. Code 2 parmi 5

C'est un code sur 5 bits, contrairement aux codes BCD et excédent 3 qui sont des codes à 4 bits. Un code 2 parmi 5 est constitué de deux bits à 1 et trois bits à 0, c'est-à-dire que deux bits seulement peuvent être à 1. Le tableau suivant donne l'équivalent des chiffres décimaux en code 2 parmi 5.

Chiffre décimal	Code 2 parmi 5
0	00011
1	00101
2	00110
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

7. Représentation des caractères

7.1. Code ASCII

Les caractères (appelés aussi symboles alphanumériques) incluent les lettres majuscules et minuscules, les symboles de ponctuation (& ~ , . ; # " – …) et les chiffres. Un texte, ou une chaîne de caractères est représentée par une suite de caractères. Le codage des caractères se fait par une table de correspondance indiquant la configuration binaire qui représente chaque caractère. Le code le plus répandu est le code ASCII (*American Standard Code for Information Interchange*). Il représente chaque caractère sur 7 bits (on parle parfois de code ASCII étendu ent utilisant 8 bits pour coder des caractères supplémentaires). Le code ASCII original défini pour les besoins de l'informatique en langue anglaise, ne permet pas la représentation des caractères accentués (é, è, ê, û, etc.) et encore moins les caractères chinois ou arabes. Pour ces langues, d'autres codages existent, utilisant 16 bits par caractères. Plusieurs points importants à propos du code ASCII :

- Les codes compris entre 0 et 31 ne représentent pas des caractères, ils ne sont pas affichables. Ces codes, souvent nommés caractères de contrôle sont utilisés pour indiquer des actions comme passer à la ligne, émettre un bip sonore, etc.
    - ➲ NUL (Null) – SOH (début d'entête) – STX (début de texte) – ETX (fin de texte) – EOT (fin de transmission) – ENQ (demande) – ACK (accusé de réception) – BEL (sonnerie) – BS (espace arrière) – TAB (tabulation horizontale) – LF (saut de ligne) – VT (tabulation verticale) – FF (*Form feed*) – CR (retour à la ligne) – SO (hors code) – SI (en code) – DLE (échappement de transmission) – DC1 (marche lecteur) – DC2 (embrayage perforateur) – DC3 (arrêt lecteur) – DC4 (débrayage perforateur) – NAK (accusé de réception négatif) – SYN (synchronisation) – ETB (fin de bloc de transmission) – CAN (annulation) – EM (fin du médium) – SUB (substitut) – ESC (caractère d'échappement) – FS (séparateur de fichier) – GS

(séparateur de groupe) – RS (séparateur d'enregistrement) – US (séparateur d'enregistrement).

– Les lettres se suivent dans l'ordre alphabétique (codes 65 à 90 pour les majuscules et du 97 à 122 pour les minuscules).

– Les chiffres sont rangés dans l'ordre croissant du code 48 au 57.

	6	0	0	0	0	1	1	1	1
	5	0	0	1	1	0	0	1	1
3210	4	0	1	0	1	0	1	0	1
0000		NUL	DLE	ESPACE	0	@	P	`	p
0001		SOH	DC1	!	1	A	Q	a	q
0010		STH	DC2	"	2	B	R	b	r
0011		ETX	DC3	#	3	C	S	c	s
0100		EOT	DC4	S	4	D	T	d	t
0101		ENQ	NAK	%	5	E	U	e	u
0110		ACQ	SYN	&	6	F	V	f	v
0111		BEL	ETB	'	7	G	W	g	w
1000		BS	CAN	(	8	H	X	h	x
1001		HT	EM	)	9	I	Y	i	y
1010		LF	SUB	*	:	J	Z	j	z
1011		VT	ESC	+	;	K	[	k	{
1100		FF	FS	,	<	L	\	l	|
1101		CR	GS	-	=	M	]	m	}
1110		SO	RS	.	>	N	^	n	~
1111		SI	US	/	?	O	_	o	DEL

Figure 1 : Table ASCII

7.2. Les codes détecteurs et correcteurs d'erreurs

Dans la machine, l'information ne cesse de circuler via les bus internes entre les divers composants (unité centrale, mémoire centrale, etc.). De plus, elle est également transmise aux périphériques, voire à d'autres machines distantes à travers des réseaux de communication. Ces communications ont un inconvénient majeur ; des erreurs peuvent survenir pendant le transport de données en raison de parasites ou de pannes. Pour remédier à ce problème, des algorithmes de codage ont été conçus pour permettre la détection, et certains même, la correction des erreurs de transmission.

7.2.1. Contrôle de parité

Cette méthode permet uniquement de détecter une erreur dans un caractère. Elle consiste à rajouter un bit (dit de parité) au *n* bits du caractère à transmettre. En parité paire, le bit rajouté au caractère est calculé de façon à ce que le nombre total de bits à 1 dans le caractère, y compris le

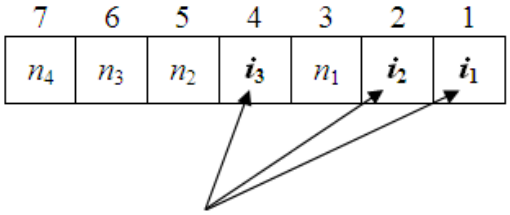
bit de parité, soit pair. En parité impaire, le nombre total de bits à 1, y compris le bit de parité, doit être impair. On rajoute généralement le bit de parité sur le poids le plus fort du caractère (en ASCII sur b7).

Exemple : en ASCII, le caractère "A" se code 100 0001. En parité paire, le code du caractère "A" devient : **0**100 0001. En parité impaire, le code du caractère "A" devient : **1**100 0001.

On remarque que l'altération d'un unique et quelconque bit durant la transmission est facilement détectée à l'arrivée. Cependant, on ne peut pas déterminer sa position.

7.2.2. Code de Hamming

Le code de Hamming est basé sur les tests de parité, et ne permet de corriger qu'une seule erreur. S'il l'on numérote les *m* bits, qui vont être transmis, de droite à gauche à partir de 1, les *i* bits de contrôle, qu'il faut rajouter aux *n* bits nécessaires pour coder l'information sont placés sur les puissances de 2 (bits n° : 1, 2, 4, 8, 16, etc.). Ainsi, chaque bit de contrôle effectue un contrôle de parité sur un certain nombre de données. Par exemple, si *n*=4 bits, on peut construire un code de Hamming sur 7 bits (*m*=7) en rajoutant 3 bits de contrôle (*i*=3).



Les trois bits de contrôle
sont placés sur les puissances de 2

On cherche pour chaque bit, quels sont les bits de contrôle permettant de vérifier sa parité.

Bit du message	**Contrôlé par**
$7 = (111)_2 = 4+2+1$	i_3, i_2, i_1
$6 = (110)_2 = 4+2$	i_3, i_2
$5 = (101)_2 = 4+1$	i_3, i_1
$4 = (100)_2 = 4$	i_3
$3 = (011)_2 = 2+1$	i_2, i_1
$2 = (010)_2 = 2$	i_2
$1 = (001)_2 = 1$	i_1

Pour détecter une éventuelle erreur, on compare la valeur reçue à celle recalculée pour chacun des bits de contrôle. Si elles sont identiques, on assigne la valeur 0 à la variable binaire S_i associée au bit de contrôle i_j, sinon on lui assigne la valeur 1. Par exemple la valeur $S_3S_2S_1$=000 indique une absence d'erreurs ; la valeur $S_3S_2S_1$=010 indique une erreur sur le bit numéro 2.