Université 8 mai 45 de Guelma
Faculté des Mathématiques, D'Informatique
et des Sciences de la Matière
Département : MI

Niveau : 1ère Mathématique-Informatique
Matière : Structure Machine 1

Chapitre 3
La Représentation de l'Information

I. Introduction

Les informations établies avec un bon nombre de terminaux sont basées sur des échanges de caractères alphanumériques codés dans un code déterminé.
Le **morse** a été le premier codage à permettre une communication longue distance. C'est ***Samuel F.B.Morse*** qui l'a mis au point en 1844. Ce code est composé de points et de tirets. L'interpréteur était l'homme à l'époque, il fallait donc une bonne connaissance du code.
De nombreux codes furent par la suite inventés à savoir le code d'Émile Baudot (portant d'ailleurs le nom de *code Baudot*, les anglais l'appelaient en revanche *Murray Code*). Les caractères étaient codés sur 5 bits, il y avait donc 32 caractères uniquement.
Dans les années 60, le code **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) est adopté comme standard. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles.

II. Définition

Codé une information (donnée ou caractère alphanumérique) revient à donner une nouvelle représentation (nombre) à l'information (caractère) c.à.d affecté à chaque information un mot du code. On rencontre habituellement des codages sur 1, 2 ou 4 octets, rarement sur 64 bits (8 octets par exemple sur le processeur DEC Alpha). Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et $2^n - 1$.

Exemple
Sur 1 octet on pourra coder des nombres de 0 à 255 = $2^8 - 1$.

Les systèmes numériques sont souvent appelés à traiter de l'information textuelle, c'est-à-dire qui est composée de caractères. Il y a quatre classes principales de caractères à représenter :

a. Les Lettres : q, w, E, r, T, t, à, É, ü, ì, Ç, µ, θ, λ, Œ, Ю, Я, ى, ﭖ, ﭦ
b. Les chiffres : 0, 1, 3, 4, 5, 6, 7, 8, 9.
c. Les caractères Typographique : !, @, /, $, %, ?, ,, (, →, ◆, ∈
d. Les Caractères de contrôle non imprimables : « Début-ligne », « Entrée», « Saut-Ligne», « Espace » « Saut-page »…etc.

III. Codage binaire

Parmi les codes binaires les plus répondus on distingue :

- Le code binaire pur,
- Le code BCD,
- Le code binaire réfléchi (code de Gray),
- Le code excède de trois.

III.1 Code binaire pur (PURE BINARY)

C'est la représentation de la valeur d'un nombre dans le système de numération binaire.

Exemple

Avec 4 bits on peut représenter tous les nombres entiers allons de 0 (0000) à 15 (1111)

	Binaire Pur			
Poids / Décimal	**2^3**	**2^2**	**2^1**	**2^0**
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

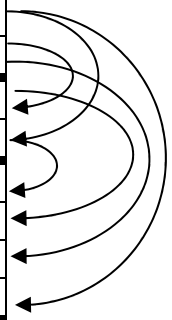
Tab 3.1 : Table binaire pur sur 4 bits

III.2 Code Gray (binaire réfléchi)

C'est un code descendant du code binaire pur. Appeler réfléchi car n-1 de ces bits peuvent être générer par réflexion tel illustrer par le tableau ci-dessous.
C'est un code très utilisé en transmission grâce à la disposition de ses combinaisons car chaque deux combinaison successives ne change que d'un seul bit ce qui facilitera la détection des erreurs.

Nous supposons que les nombres à coder sont sur 4 bits :

Décimal	Gray			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0



Tab 3.3: Table du code GRAY sur 4 bits

III.3 Code BCD (Binary Coded Decimal)

Le code **BCD** ou **DCB** (Décimal Codé Binaire) est l'un des codes les plus répondus, il a permis de rapprocher l'homme de la machine. Il a pour but de coder chaque chiffre d'un nombre décimal sur 4 bits.

On a besoin de 4 bits pour coder les dix chiffres décimaux mais les valeurs représentables sur 4 bits sont au nombre de $2^4 = 16$ donc il y a 6 configurations non utilisables dont il faut en tenir compte pour les opérations arithmétiques en BCD.

	BCD				
Décimal	8	4	2	1	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	**1**	**0**	**1**	**0**	**6 Combinaisons interdites en BCD**
11	**1**	**0**	**1**	**1**	
12	**1**	**1**	**0**	**0**	
13	**1**	**1**	**0**	**1**	
14	**1**	**1**	**1**	**0**	
15	**1**	**1**	**1**	**1**	

Tab 3.2: Table du code BCD

Exemple

$(501)_{10} = (0101\ 0000\ 0001)_{BCD}$

Remarque

- Le code BCD est appelé aussi le code 8421.
- Les notations 1010, 1011, 1100, 1101, 1110, 1111 sont interdites en BCD.
- Il existe d'autres codes binaire ayant des pondérations différentes du BCD tel que : le code Aiken (2421) et le code 7421..etc.

III.4 Code Excède de trois

Appeler aussi code « plus trois » parce qu'il attribut à chacun des nombres décimaux de 0 à 9 le code BCD plus trois ce qui donne le tableau ci-dessous :

Décimal	**Excède de trois = BCD + 3**			
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Tab 3.4: Table du code Excède de trois

Exemple

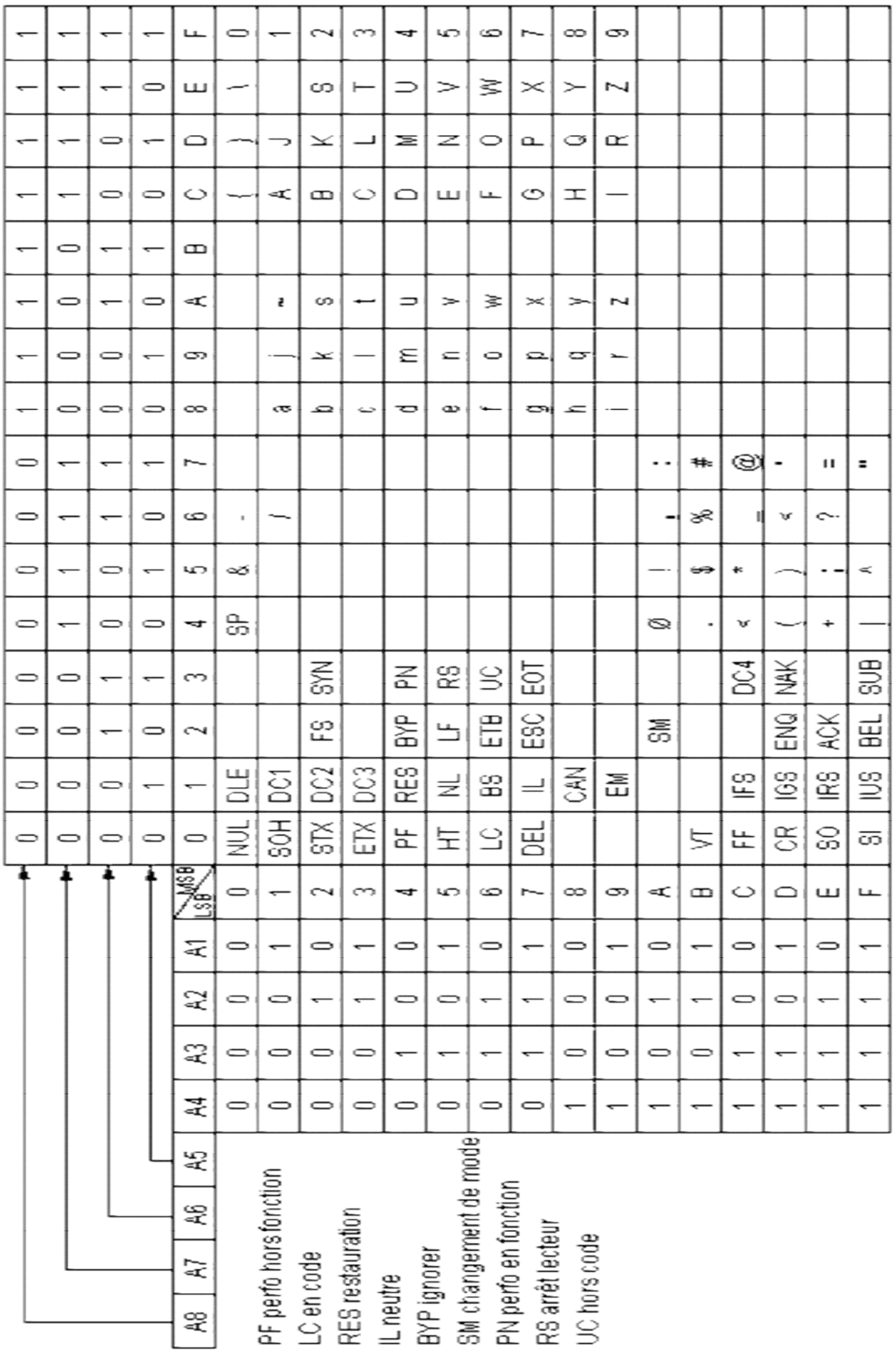
$(501)_{10} = (1000\ 0011\ 0100)_{Ex\text{-}3}$

IV. Représentation des caractères

La communication et l'échange d'information entre terminaux sont basés sur la transmission de caractères alphanumériques codés dans un code donné appelé code alphanumérique. Les codes les plus fréquents sont constitués de mot de 6, 7, 8, ou 9 bits.
Parmi les codes les plus utilisés on trouve le code EBCDIC, le code ASCII et le code UNICODE…etc.

IV.1 Code EBCDIC (Extended Binary-Coded Decimal Interchange Code)

Le code *EBCDIC* développé par IBM, permet de coder des caractères sur 8 bits. Bien que largement répandu sur les machines IBM. Il existe au moins 6 versions incompatibles entre elles. Ce mode de codage a été critiqué pour cette raison, mais aussi parce que certains caractères de ponctuation ne sont pas disponibles dans certaines versions. Il a été par la suite remplacé par le code ASCII.



A8	A7	A6	A5	A4	A3	A2	A1	MSB / LSB	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	NUL	DLE			SP	&	-						{	}	\	0
0	0	0	0	0	0	0	1	1	SOH	DC1					/		a	j	~		A	J		1
0	0	0	0	0	0	1	0	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0	0	0	0	0	0	1	1	3	ETX	DC3							c	l	t		C	L	T	3
0	0	0	0	0	1	0	0	4	PF	RES	BYP	PN					d	m	u		D	M	U	4
0	0	0	0	0	1	0	1	5	HT	NL	LF	RS					e	n	v		E	N	V	5
0	0	0	0	0	1	1	0	6	LC	BS	ETB	UC					f	o	w		F	O	W	6
0	0	0	0	0	1	1	1	7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
0	0	0	0	1	0	0	0	8		CAN							h	q	y		H	Q	Y	8
0	0	0	0	1	0	0	1	9		EM							i	r	z		I	R	Z	9
0	0	0	0	1	0	1	0	A			SM		ø	!	¦	:								
0	0	0	0	1	0	1	1	B	VT				.	$	%	#								
0	0	0	0	1	1	0	0	C	FF	IFS		DC4	<	*	=	@								
0	0	0	0	1	1	0	1	D	CR	IGS	ENQ	NAK	(	)	<	-								
0	0	0	0	1	1	1	0	E	SO	IRS	ACK		+	;	?	=								
0	0	0	0	1	1	1	1	F	SI	IUS	BEL	SUB	|	^		"								

Tab 3.5: Table du code EBCDIC

IV.2 Code ASCII (**A**merican **S**tandard **C**oded **I**nformation for **I**nterchange**)**

La mémoire de l'ordinateur conserve toutes les données sous forme numérique. Il n'existe pas de méthode pour stocker directement les caractères. Chaque caractère possède donc son équivalent en code numérique: c'est le **code ASCII** (*American Standard Code for Information Interchange* - traduisez " Code Americain Standard pour l'Echange d'Informations") fut créé pour la transmission de textes vers des terminaux ou des imprimantes

Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127) (voir Tab. 3.6)

- Les codes 0 à 31 ne sont pas des caractères. On les appelle *caractères de contrôle* (voir Tab 3.7) car ils permettent de faire des actions telles que: retour à la ligne (CR), Bip sonore (BEL)
- Les codes 48 à 57 représentent les chiffres dans l'ordre (0,1,…,9)
- Les codes 65 à 90 représentent les majuscules (A,…,Z)
- Les codes 97 à 122 représentent les minuscules (a,….,z)

Remarque

Il suffit de modifier le 6ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale.

La table 3.7 illustre les désignations des caractères de contrôle de la table ASCII de base

			6	0	0	0	0	1	1	1	1
Bits			5	0	0	1	1	0	0	1	1
3	2	1	4 / 0	0	1	0	1	0	1	0	1
0	0	0	0	NUL	DLE	SPACE	0	@	P	`	p
0	0	0	1	SOH	DC1	§	1	A	Q	a	q
0	0	1	0	STX	DC2	"	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	‘	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	del

Tab 3.6: Table du code ASCII de Base

Exemple
A = $(100\ 1000)_2 = (41)_H$
B = $(100\ 0010)_2 = (42)_H$
a = $(110\ 0001)_2 = (61)_H$

Caractères de contrôle		
Car.	**Sigle**	**Désignation**
NUL	Null	Nul
SOH	Start Of Heading	Début d'en-tête
STX	Start Of Text	Début de texte
ETX	End Of Text	Fin de texte
EOT	End Of Transmission	Fin de transmission
ENQ	Enquiry	Interrogation
ACK	Acknowledge	Accusé de (bonne) réception
BEL	Bell	Sonnerie
BS	Back Space	Espace arrière
HT	Horizontal Tabulation	Tabulation horizontale
LF	Line Feed	Interligne (saut de ligne)
VT	Vertical Tabulation	Tabulation verticale
FF	Form Feed	Présentation de feuille (saut page)
CR	Carriage Return	Retour chariot
SO	Shift Out	Code spécial
SI	Shift In	Code normal
DLE	Data Link Escape	Échappement de la liaison
DC1	Device Control 1	Commande dispositif 1 (XON)
DC2	Device Control 2	Commande dispositif 2
DC3	Device Control 3	Commande dispositif 3
DC4	Device Control 4	Commande dispositif 4 (XOFF)
NAK	Negative Acknowledge	Accusé de non réception
SYN	Synchronous idle	Synchronisation
ETB	End of Transmission Block	Fin de bloc de transmission
CAN	Cancel	Annulation
EM	End of Medium	Fin du support
SUB	Substitute	Substitution
ESC	Escape	Échappement
FS	File Separator	Séparateur de fichiers
GS	Group Separator	Séparateur de groupe
RS	Record Separator	Séparateur d'enregistrement
US	Unit Separator	Séparateur d'unité
SP	Space	Espace
DEL	Delete	Effacement (d'un caractère)

Tab 3.7 Caractères de contrôle ASCII

VI.3 Code ASCII Etendu

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Pour coder ce type de caractère il faut recourir à un autre code. Le code ASCII a donc été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu). Ce code attribue les valeurs 0 à 255 aux lettres majuscules et minuscules, aux chiffres, aux marques de ponctuation et aux autres symboles (caractères accentués).

Les caractères supplémentaires sont essentiellement :

- Les caractères accentués utilisés dans la langue française.
- Des jeux de caractères utilisés dans d'autres langues.
- Quelques symboles mathématiques.
- Des caractères semi-graphiques qui permettent de réaliser des petits dessins géométriques (cadres, soulignés, etc,…)

Les deux jeux de caractères ASCII étendus les plus couramment utilisés sont :

- Le code ASCII étendu OEM (c'est le jeu de caractère standard du Dos), c'est-à-dire celui qui équipait les premières machines de type **IBM PC**
- Le code ASCII étendu **ISO-ANSI** (jeu de caractères international utilisé dans Windows), utilisé par les systèmes d'exploitation récents.

Ces deux jeux différents notamment au niveau des caractères nationaux accentués et des caractères semi-graphiques.
Le code ASCII étendu qui équipait les premières machines de type IBM PC est donné par le tableau Tab 3.8.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	ä	Å
9	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ç	£	¥	₧	ƒ
A	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B	▒	▓	▓	|	┤	╡	╢	╖	╕	╣	╗	╝	╜	╛	┐	
C	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	=	╬	╧
D	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	▉	▄	▌	▐	▀
E	α	ß	Γ	π	Σ	σ	µ	τ	Φ	Θ	Ω	δ	∞	ø	∈	∩
F	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	ⁿ	²	▮	

Tab 3.8 : Code ASCII étendu d'IBM PC

Le code ASCII étendue ANSI utilisé par les systèmes d'exploitation récents PC est donné par la table 3.9.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	□	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	□	□
9	□	‘	’	“	”	•	–	—	˜	™	š	›	œ	□	□	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Tab 3.9 : Code ASCII étendu ANSI

IV.4 Code Unicode

Le développement mondiale de l'informatique et la diversité de plus en plus importante des caractères a stocké a pousser les organismes de normalisation ISO de travailler depuis 1988 pour mettre en œuvre un nouveau code universel d'où la naissance du système de codage des caractères UNICODE (**UNI**versal **CODE**) en 1991

Le système Unicode permet de représenter n'importe quel caractère par un code indépendamment de tout système d'exploitation ou langage de programmation. Il regroupe ainsi la quasi-totalité des alphabets existants (arabe, arménien, cyrillique, grec, hébreu, latin, chinois ...) et est compatible avec le code ASCII.

IL se présente sous trois formes :

1. **UTF8** (Unicode transformation Format) très utilisé sur internet et présente un codage de taille variable où tous les caractères de base ASCII 7 bits sont codés sur 1 octet. Au-delà, les caractères peuvent prendre deux à 4 octets.
2. **UTF16** codage de taille fixe sur 2 octets (16 bits) soit une représentation de 65 535 caractères différents. Par exemple, les codes Latin-1 sont codés de 0000 à 00FF, le grec de 0370 à 03FF, l'arabe de 0600 à 06FF…etc.
3. **UTF32** codage sur 4 octets codage d'environ quatre milliards de caractères.

NB : vous pouvez voir l'ensemble des codes **Unicode** sur le site : **www.unicode.org**

Unicode Text	
A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
س	0000 0110 0011 0011
ل	0000 0110 0100 0100
ط	0000 0110 0011 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
≤	0010 0010 0111 0000
γ	0000 0011 1011 0011

Tab 3.10 : Exemple de caractères codés en UNICODE

V. Représentation des nombres

V.1 Représentation des nombres entiers signés

Les nombres binaires peuvent être soit positifs ou négatifs. Pour différencier ces deux types de données on utilise un bit qu'on appel «bit de signe » lorsqu'il est à zéro il désigne un nombre positif et lorsqu'il est à un il désigne un nombre négatif.
Ce bit est ajouté au code binaire du nombre, généralement il est mis à gauche (poids fort) mais il n'entre pas dans l'évaluation du nombre.

Remarque

La limitation des bits de représentation des nombres dans l'ordinateur nous oblige à faire un format normalisé avec un nombre de bits fixe pour les nombres signés.

Exemple

Si nous avons des mots mémoire de 8 bits le bit le plus à gauche représentera le signe tandis que les 7 bits qui restent représenteront la valeur du nombre.

Les nombres signés peuvent être représenté par trois méthodes différentes :

- La représentation en valeur absolue plus le signe (SVA).
- La représentation en complément à 1 (Cà1)
- La représentation en complément à 2 (Cà2)

V.1.1 Représentation en valeurs absolues plus le signe :

Dans cette représentation le nombre est donné par sa valeur absolue plus le bit de signe.

Exemple

Représenter sur 8 bits le nombre (-24) par la méthode de valeur absolue plus le signe.

Solution

(-24) Valeur absolue (+ 24) = (**0** 0011000)$_2$ donc $(-24)_{10} = (10011000)_2$

Sur 8 bits

Bit de signe **Valeur absolue**

V.1.2. Représentation en complément à 1 (complément restreint)

Définition 1

Le complément à un d'un nombre binaire s'obtient en inversant les « 1 » en « 0 » et les « 0 » en « 1 ».

Remarque

On obtient aussi le complément à 1 (on note C à 1) en soustrayant de 1 chaque chiffre de ce nombre.

Exemple

Donner le complément à 1 du nombre suivant : 11101010

11101010 **C à 1** 00010101

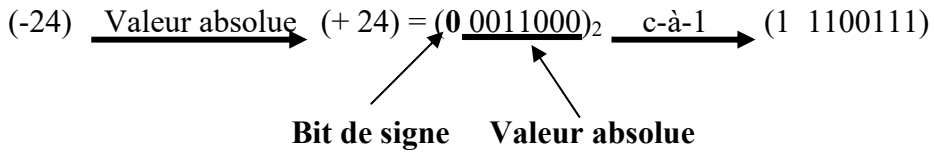
Définition 2

Les nombres signés positif en représentation c-à-1 sont obtenus comme en valeur absolue plus le signe.

Les nombres négatifs sont obtenus en faisant le c-à-1 de leurs équivalents positifs.

Exemple

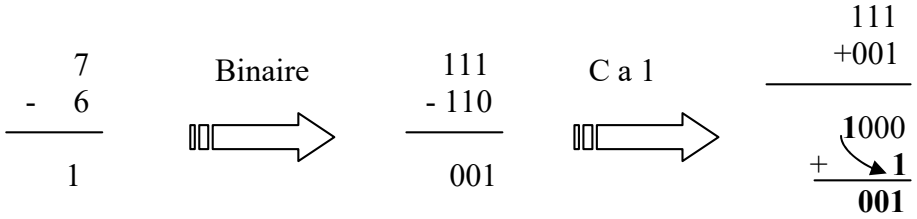
Ecrire sur 8 bits le nombre (-24) par c-à-1 :



Soustraction binaire par complément à 1

La soustraction devient une addition du complément a un avec report de la retenue si elle existe.

Exemple



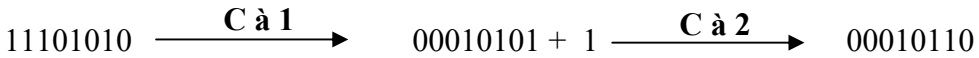
V.1.3 Représentation en complément à 2 (complément vrai)

Définition 1

Le complément à 2 d'un nombre représenté en binaire est obtenu en ajoutant « 1 » à son complément à 1.

Exemple

Donner le complément à 2 du nombre suivant : 11101010



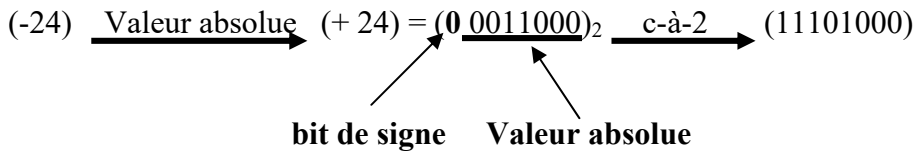
Définition 2

Les nombres signés positifs en représentation complément à deux sont représentés comme en valeur absolue plus le signe.

Les nombres négatifs sont obtenus en faisant le c-à-2 de leurs équivalents positifs.

Exemple

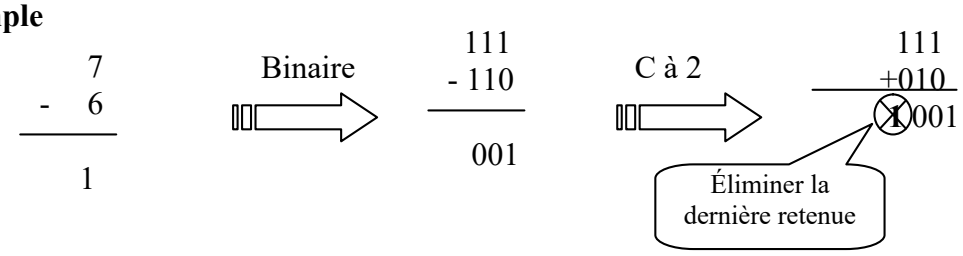
Ecrire sur 8 bits le nombre (-24) par la méthode du complément à 2



Soustraction binaire par complément à 2

En complément on effectue une addition mais la dernière retenue est négliger la retenue si elle existe.

Exemple



V.2 Représentation des nombres fractionnaires signés

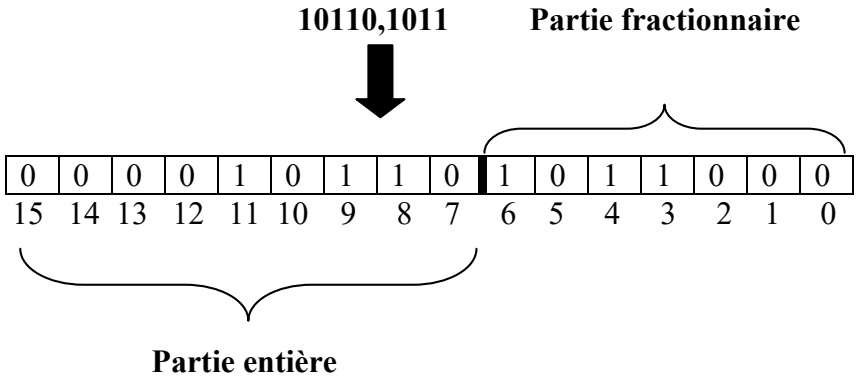
Rappelons que les nombres fractionnaires sont les nombres qui comportent une partie inférieure à 1, dans ces nombres on distingue la représentation en virgule fixe et celle en virgule flottante.

V.2.1 Virgule fixe

C'est un mode de représentation des nombres fractionnaires utilisant un nombre déterminé de positions tel que la virgule qui sépare la partie entière de la partie fractionnaire occupe une position fixe.

Exemple

Dans un mot de 16 bits, les 7 bits de droite représentent la partie fractionnaire et le reste la partie entière.



Remarque

L'inconvénient de cette méthode est qu'elle limite les valeurs à représenter.

V.2.2 Virgule flottante

C'est un mode de représentation des nombres fractionnaires dans lequel la virgule qui sépare la partie entière de la partie fractionnaire occupe une position quelconque gérée automatiquement.

Cette méthode consiste à représenter un nombre réel sous la forme :

$$N = S.M.B^E$$

Avec :

S : Signe du nombre.

M : La mantisse qui est un nombre purement fractionnaire (pas de chiffre significative avant la virgule)
B : La base du système de numération (2, 8, 10, 16,…etc.)
E : L'exposant

Exemple

$125\ 000 = +\ 125\ .\ 10^3 = +\ 0{,}125\ .\ 10^6$ avec : $S = +,\ M = 0{,}125,\ B = 10,\ E = 6$

Remarque

L'avantage de la représentation en virgule flottante par rapport à la virgule fixe est que l'intervalle des valeurs possibles est plus étendu.

V.2.3 Standard IEEE-754

Chaque ordinateur avait sa propre représentation en virgule flottante, jusqu'à ce que la norme IEEE-754 soit établie par le comité IEEE [**I**nstitut of **E**lectrical and **E**lectronics **E**ngineers]

Le standard IEEE définit trois formats de représentation de nombres en virgule flottante :

1. La simple précision : sur 32 bits (1 bit de signe de la mantisse + 8 bits de l'exposant + 23 bits pour la mantisse).
2. La double précision : sur 64 bits (1 bit de signe de la mantisse + 11 bits de l'exposant + 52 bits pour la mantisse).
3. La précision étendue : sur 80 bits.

Dans ce type de représentation la mantisse est normalisée en mettant le bit le plus à gauche avant la virgule, ce qui ne sera pas représenté, et ne représente que la partie fractionnaire comme mantisse.

L'exposant est représenté toujours par la caractéristique (*C*)

$$C = R + 2^{n-1} - 1$$

R : l'exposant après normalisation.
n : le nombre de bits de la caractéristique (sur lequel nous allons représenter l'exposant).

Exemple

Représenter le nombre N selon le standard IEEE-754 : $N = 87{,}125$

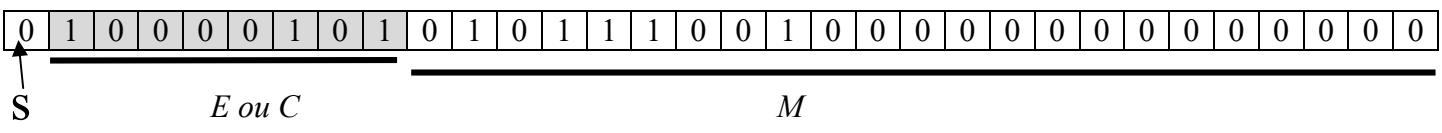
Solution

1. le nombre $(87{,}125)_{10} = (1010111{,}001)_2$
2. Normalisation : $1010111{,}001 = 1{,}010111001 \times 2^6$
3. La caractéristique : $C = 6 + 2^{(8-1)} - 1 = 133 = (10000101)_2$
4. Le nombre est positif alors S = 0.
5. La représentation sur 32 bits :

M sur 23 bits

M = 01011100100000000000000 (14 bits de zéros ajouté pour compléter la représentation).

6. Le nombre N en virgule flottante est :



7. Pour faciliter la représentation on la convertit en hexadécimal en regroupant 4 à 4 soit : *N* = 42AE4000.