

## Chapitre II – Codage de l'information



Ce second chapitre englobe des notions fondamentales que devrait acquérir tout étudiant désirant comprendre la structure des ordinateurs et coder des programmes pour les faire fonctionner. On y traite la notion de codage de l'information. Vous allez voir qu'en informatique, l'information est codée en binaire et uniquement en binaire. C'est la raison pour laquelle nous avons traité le système de numération binaire au premier chapitre. Mais cela, n'est pas suffisant: Il faut pouvoir coder des nombres, du texte, des images, du son, de la vidéos et divers autres objets informatique comme des document pdf, textes avec mise en forme, page web et bien d'autres encore ! C'est ce que nous allons vous faire découvrir dans ce chapitre

Chapitre II – Codage de l'information .....	1
II.1 – Codages binaires .....	2
II.2 – Le binaire pur .....	2
II.2 – Code Gray .....	2
II.3 – Code BCD .....	3
II.2 – Codage des caractères .....	3
II.3 – Codage du son .....	5
II.4 – Codage des images .....	5
II.5 – Codage de la vidéo .....	7
II.6 – Codage des nombres .....	8
II.6.1 – Codage des entiers naturels .....	8
II.6.2 – Codage des entiers relatifs (nombres signés) .....	8
II.6.3 – Codage des réels .....	11

## II.1 – Codages binaires

Le codage est un processus nécessaire à l'être humain pour communiquer. On peut définir un code comme un ensemble de symboles (alphabet d'une langue par exemple) représentant des informations utiles. En informatique, ces symboles se résument aux deux objets que sont le « 0 » et le « 1 ». Donc, dans ce domaine, toutes les informations sont représentées sous la forme de configurations binaires. Que ce soit du texte, des images, du son, de la vidéo ou simplement des nombres, c'est le codage binaire que l'on utilise.

Avant d'aborder les différents codages, on doit connaître deux concepts importants : la quantité d'information qu'on peut coder (représenter) et le nombre de bits que l'on devrait utiliser pour coder

La quantité d'information représentables correspond au nombre de configurations différents que l'on peut avoir dans un codage. Ainsi, en binaire pure (tel qu'on l'a défini au chapitre 1),  **$n$  bits** peuvent coder  **$2^n$  informations différentes**.

Le nombre de bits que l'on utilise pour coder est très important car il limite l'étendue des valeurs (ou informations) que l'on veut coder. Ainsi, dans les ordinateurs, on utilise le plus souvent : 8, 16, 32 ou 64 bits.

Pour mesurer la quantité d'information représentées en binaire, on utilise les bits ou l'octet. L'octet correspondant à 8 bits. A l'image des autres unités de mesures comme le mètre ou le kilogramme, on a défini les mesures suivantes :

Nom	Symbole	Valeur
kilooctet	ko	$10^3$
mégaoctet	Mo	$10^6$
gigaoctet	Go	$10^9$
téraoctet	To	$10^{12}$
pétaoctet	Po	$10^{15}$
exaoctet	Eo	$10^{18}$
zettaoctet	Zo	$10^{21}$
yottaoctet	Yo	$10^{24}$

Nom	Symbole	Valeur
kibioctet	Kio	$2^{10}$
mébioctet	Mio	$2^{20}$
gibioctet	Gio	$2^{30}$
tébioctet	Tio	$2^{40}$
pébioctet	Pio	$2^{50}$
exbioctet	Eio	$2^{60}$
zébioctet	Zio	$2^{70}$
yobioctet	Yio	$2^{80}$

## II.2 – Le binaire pur

Le binaire pur est aussi qualifié de binaire naturel. Ce codage a déjà été abordé dans le cadre du chapitre 1 traitant des systèmes de numération. En effet, dans ce codage on associe à chaque entier positif la valeur qui lui correspond selon le système de numération binaire. Ainsi, en ayant  **$n$  bits** on pourra coder les valeurs comprises entre  $[0 \text{ et } 2^n]$ .

Exemple :

- Sur 6 bits :  $(35)_{10} = (100011)_2$
- Attention la valeur  $(35)_{10}$  n'est pas représentable sur 5 bits. En effet, rappelez-vous que pour coder une valeur sur  **$n$  bits**, elle doit être comprise entre  **$[0 \text{ et } 2^n]$** , dans notre cas entre  $[0 \text{ et } 2^5] = [0 \text{ et } 32]$ .

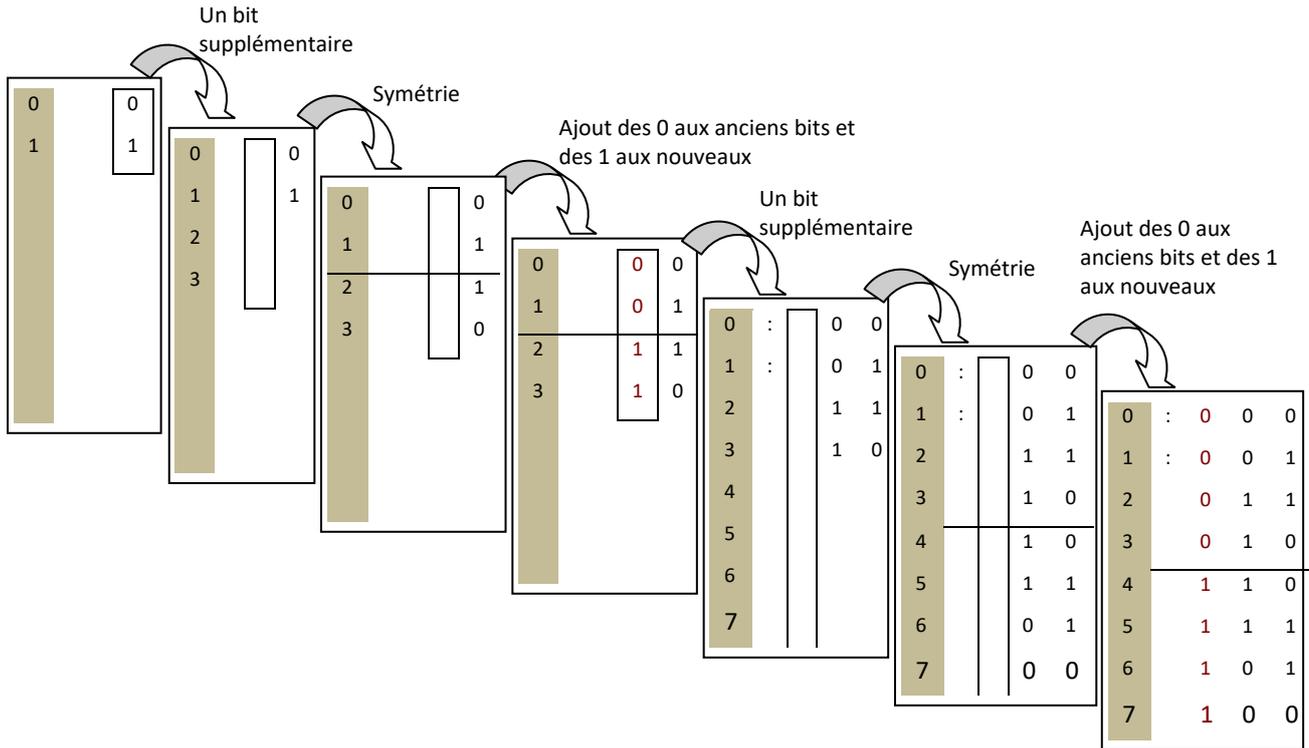
## II.2 – Code Gray

Le code Gray est utilisé lorsque l'on désire une progression numérique binaire sans parasite transitoire. Il sert également dans les tableaux de Karnaugh utilisés lors de la conception de circuits logiques. Il est construit de telle façon qu'à partir du chiffre 0 chaque nombre consécutif diffère du précédent immédiat d'un seul digit (bit).

Une des méthodes de construction du code Gray s'appelle la méthode du code binaire réfléchi ou code REFLEX. Voici son principe:

1. On établit un code de départ : zéro est codé 0 et un est codé 1.
2. Puis, à chaque fois qu'on a besoin d'un bit supplémentaire,
3. on symétrise les nombres déjà obtenus (comme une réflexion dans un miroir)
4. et on rajoute un 1 au début des nouveaux nombres et un zéro sur les anciens.

**Illustration :**



**II.3 – Code BCD**

Le BCD (Binary Coded Decimal), ou Décimal Codé en Binaire en français) est le code décimal le plus utilisé en électronique. Il contient des mots-code qui sont la traduction en binaire naturel (sur 4 bits) de chacun des dix chiffres du système décimal.

**Exemple :**

- $(25)_{10} = (0010\ 0101)_{BCD}$
- $(43)_{10} = (0100\ 0011)_{BCD}$
- $(1111)_2 = (0001\ 0101)_{BCD}$

**II.2 – Codage des caractères**

Il existe plusieurs logiciels traitant du texte de différentes langues. Comment sont codés les caractères pour permettre d'afficher une langue et aussi plusieurs langues? En fait, il s'agit de coder les lettres de l'alphabet y compris les chiffres et les caractères de contrôle du clavier. Le choix d'un code dépend du pays et de la langue.

Différents codes sont utilisés:

- ASCII (7 bits)
- ASCII étendue (8 bits)
- Unicode (16 bits)
- Etc.



## Le code ASCII

Autrement dit : American Standard Code for Information Interchange. C'est codage sur 7 bits ce qui donne 128 codes différents :

- 0 à 31 : caractères de contrôle (retour à la ligne, Bip sonore, etc....)
- 65 à 90 : majuscules
- 97 à 122 : minuscules

Dans ce code, les caractères accentués ne sont pas représentés et c'est la raison pour laquelle il est limité uniquement à quelques langues (anglais notamment).

ASCII Table															
Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

## ASCII étendue

Afin de représenter plus de langues et notamment des langues occidentales comme la France, on a étendu le code de 7 à 8 bits.

## Codage Unicode

Unicode utilise des codes de valeurs bien plus grandes que celle allant de 0 à 127. C'est un codage sur 16 bits! Il permet de représenter tous les caractères spécifiques aux différentes langues. Il se développe de plus en plus.

## Codage UTF8

Unicode, dans la théorie, c'est très bien, mais dans la pratique, pour chaque lettre il occupe 2 octets (16 bits). C'est du gaspillage: Par exemple plusieurs langues partagent les mêmes lettres (français, anglais, ...) notamment les lettres de l'alphabet français non accentués. Pour optimiser cela, on utilise UTF-8:

Un texte en UTF-8 est simple: il est partout en ASCII, et dès qu'on a besoin d'un caractère appartenant à l'Unicode, on utilise un caractère spécial signalant "attention, le caractère suivant est en Unicode".

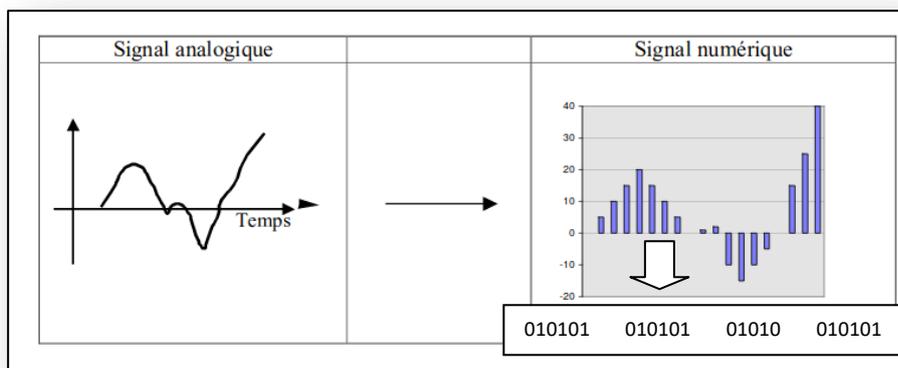
Par exemple, pour le texte "Bienvenue à Bejaia!", seul le « à » ne fait pas partie du code ASCII. Donc seul un caractère sera sur 16 bits, les autres seront représentés sur 8 bits!

## II.3 – Codage du son

Le son se représente naturellement sous la forme d'un signal analogique. Il s'agit d'une onde acoustique issue de la vibration d'une membrane entraînant des mouvements ondulatoires de l'air environnant. Un haut parleur ou les cordes vocales d'un homme peuvent être à l'origine de ces vibrations.

Le codage d'un tel signal consiste à le représenter en binaire afin de pouvoir le transmettre à l'ordinateur qui ne peut stocker et traiter les informations qu'en binaire.

Le signal analogique du son peut être déterminé par une fonction continue représenté par une fréquence et une amplitude. La fréquence définit des sons aigus ou graves alors que l'amplitude définit le volume du son. Pour passer d'un signal analogique vers une représentation binaire. C'est ce qu'on appelle par numérisation du son. Ce processus consiste à capter la valeur de l'amplitude du signal analogique à des laps de temps répétitifs constant qu'on appelle la valeur d'échantillonnage. Cette valeur de l'amplitude, est codée sur  $n$  bits à chaque période d'échantillonnage.



La période d'échantillonnage est très importante car elle a un impact par rapport à la fidélité du son numérisé. En effet, il ne faut pas perdre de vue que le codage binaire résultant ne représente qu'une partie du signal analogique qui est une fonction continue. On doit donc fixer la période d'échantillonnage de sorte à ce que la perception humaine ne détecte pas de différences entre le signal d'origine et celui numérisé.

## II.4 – Codage des images

On distingue 2 catégories de codage des images :

- Les images vectorielles
- Les images bitmap ou matricielles

Dans le codage vectoriel, l'image est codée par un ensemble de formules mathématiques, alors que dans le codage matriciel, l'image est codée comme un tableau de points.

Dans ce qui suit, nous présenterons uniquement le codage matriciel car c'est ce codage qui permet de représenter numériquement les photos.

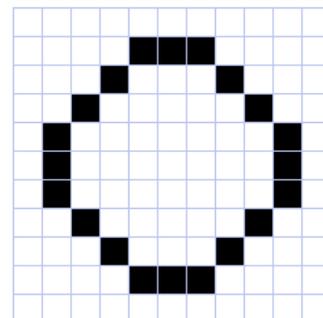
**Bitmap** signifie «carte de bits : l'image est décrite point par point. Les points d'une image sont appelés des pixels («*picture elements*»). Chaque pixel est décrit par un nombre indiquant sa couleur. L'image est donc représentée par une série de nombres. Le codage de l'image se fait en écrivant successivement les bits correspondant à chaque pixel, ligne par ligne, en commençant par le pixel en bas à gauche. Le codage est simple mais l'image bitmap occupe beaucoup de mémoire : plus les pixels sont petits, plus nombreux ils sont! Ce qui explique la nécessité de compression.

Trois paramètres définissent une image bitmap :

- Le nombre de colonnes
- Le nombre de lignes
- Le nombre de couleur par pixel

Les 2 premiers paramètres déterminent ce qu'on appelle par définition de l'image. Par exemple 800x600 pixels. Le dernier paramètre détermine ce qu'on appelle par profondeur de l'image. C'est lui qui définit les couleurs de l'image pixel par pixel.

Voici une illustration d'une image bitmap. Elle a une définition de 11x11 pixels. Elle est en noir et blanc



## Comment coder la couleur ?

On avait dit qu'une image est une matrice de pixels. Chaque pixel est caractérisé par sa position dans l'image (n° de ligne n° de colonne). Mais qu'en est-il de sa couleur. En fait, on code la couleur en se basant sur 3 couleurs primaires que sont le rouge (R), le vert (V) et le bleu (B), abrégé en RVB. Par un principe dit de synthèse additive trichrome, à partir de ces trois couleurs primaires, on peut générer toutes les autres en allant du noir jusqu'au blanc en passant par les autres.

Du point de vue du stockage et du traitement des couleurs des pixels, on associe un nombre de bits fixe pour chacun des couleurs rouge, vert et bleu.

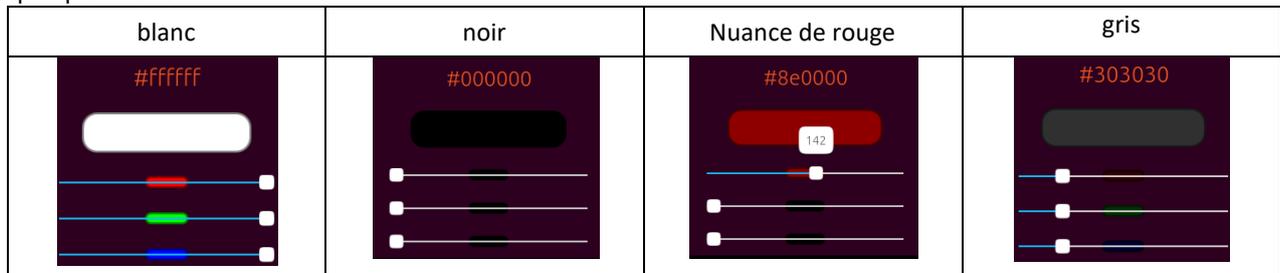
Voici quelques possibilités de codage de la couleur des pixels généralement utilisées:

- Bitmap noir et blanc : on peut coder la couleur sur 1 bit (0 pour le noir, 1 pour le blanc).
- Bitmap 16 couleurs ou 16 niveaux de gris : 4 bits suffisent.
- Bitmap 256 couleurs ou 256 niveaux de gris : Un octet fait l'affaire.
- "Couleurs vraies" (True Color) ou "couleurs réelles" : cette représentation permet de représenter une image en définissant chacune des composantes (RGB, pour rouge, vert et bleu). Chaque pixel est représenté par un entier comportant les trois composantes, chacune codée sur un octet, c'est-à-dire au total 24 bits (16 millions de couleurs). Il est possible d'ajouter une quatrième composante permettant d'ajouter une information de transparence ou de texture, chaque pixel est alors codé sur 32 bits.

Sachant qu'on est en codage 24 bits, voici quelques particularités:

- Pour chaque couleur, les valeurs proches de 0 signifient une faible nuance (intensité) de la couleur alors que les valeurs proches de 255 signifient une forte nuance de couleur. Par exemple le codage : (R, V, B) = (0,0,0) traduit un noir alors que (R, V, B) = (255,255,255) traduit un blanc.
- (R, V, B) = (0,0,255) correspond au bleu
- (R, V, B) = (0,255,0) correspond au vert
- (R, V, B) = (255,0,0) correspond au rouge
- (R, V, B) = (255,255,0) correspond au jaune car c'est un mélange équilibré entre le rouge et le vert
- Lorsque les valeurs des 3 couleurs sont égales, cela se traduit par un gris qui va du noir (R,V,B)=(0,0,0) jusqu'au blanc (R,V,B)=(255,255,255)

Voici quelques illustrations :



## Comment calculer le poids d'une image ?

D'abord on sous-entend par poids la capacité mémoire requise pour stocker l'image sur un support mémoire. On le mesure, généralement, en octet (ou ses différents multiples, kilo-octets, méga-octets).

Pour connaître le poids (en octets) d'une image, il est nécessaire de compter le nombre de pixels que contient l'image, cela revient à calculer le nombre de cases du tableau, soit la hauteur de celui-ci que multiplie sa largeur (nombre de lignes x nombre de colonnes). Le poids de l'image est alors égal à son nombre de pixels que multiplie le poids de chacun de ces éléments.

Voici le calcul pour une image 640x480 en True Color :

- $640 \times 480 = 307\,200$
- $24 \text{ bits} / 8 = 3 \text{ octets}$
- Soit un poids de  $307\,200 \times 3 = 921\,600 \text{ octets}$ .

Voici quelques exemples (en considérant que l'image n'est pas compressée) :

Définition de l'image	Noir et blanc (1 bit)	256 couleurs (8 bits)	65 000 couleurs (16 bits)	True Color (24 bits)
320x200	7,8 ko	62,5 ko	125 ko	187,5 ko
640x480	37,5 ko	300 ko	600 ko	900 ko
800x600	58,6 ko	468,7 ko	937,5 ko	1,4 Mo
1024x768	96 ko	768 ko	1,5 Mo	2,3 Mo

### Qu'en est-il de la compression d'images ?

Les images bitmap stockent tous les pixels d'une image sous forme d'une matrice. Le problème est que ça demande beaucoup d'espace mémoire. Afin de remédier à ce problème divers solutions d'optimisation ou de compression des fichiers d'images sont utilisées. C'est ce qui a donné naissance à plusieurs formats : GIF, JPEG, PNG, etc. Voici un descriptif de quelques formats largement utilisés :

Format	Type	Remarques
BMP : Bitmap (extension : <b>bmp</b> )	Sans compression	Format standard avec Windows Renferme tous types d'images noir et blanc et couleur
PICT (extension : <b>pct</b> )	Sans compression	Format standard avec Machintosh Renferme tous types d'images noir et blanc et couleur
TIFF : Tagged Image Format (extension : <b>tif</b> )	Compression non dégradante possible	Format très classique pour les images
GIF : Graphics Interchange Format (extension : <b>gif</b> )	Forte compression non dégradante	Format d'images compressées destiné surtout pour la diffusion sur les réseaux de télécommunication (256 couleur maximum)
JPEG : Joint Photographic Expert Group (extension : <b>jpg</b> )	Norme internationale de compression dégradante	Renferme tous types d'images noir et blanc et couleur Plusieurs niveaux de qualité selon le taux de compression

## II.5 - Codage de la vidéo

Une animation ou une vidéo peut être définie assez comme une succession d'images effectuée à un certain rythme. Ces images peuvent être de natures et de formats différents (dessins, graphiques, photos). Les animations peuvent avoir des caractéristiques différentes. Suivant qu'elles proviennent d'une série de photos ou de dessins, accompagnée d'une séquence audio ou non, on parlera tantôt de vidéos, tantôt d'images animées.

Les vidéos sont composées de 2 parties : une partie sonore et une partie vidéo (images). Ces 2 parties sont bien évidemment synchronisées.

En plus des caractéristiques du son et des images, dans la vidéo nous avons un paramètre supplémentaire et non des moindres : le nombre d'image à défiler par seconde. Ce paramètre lorsqu'il est supérieur à 10, on voit un effet d'animation. Cela est dû au fait que notre œil n'est pas capable de percevoir une image qui défile à une trop grande vitesse. En dessous de 10 images par seconde, l'animation apparaîtra saccadée. A 25 images par seconde, l'animation (vidéo) se rapprochera de la réalité et l'œil humain ne sera pas capable de voir des imperfections dans l'animation.

## II.6 – Codage des nombres

Coder une information consiste à établir une correspondance entre sa représentation externe (habituelle) et sa représentation interne dans la machine, qui est une suite de bits.

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

### II.6.1 – Codage des entiers naturels

- Les entiers naturels (positifs ou nuls) sont codés sur un nombre fixe d'octets (1 octet = 8 bits). On rencontre habituellement des codages sur 1, 2, 4 octets, et plus rarement sur 8 octets (64 bits)
- Un codage sur  $n$  bits permet de représenter tous les entiers naturels compris entre  $0$  et  $2^n - 1$ .

☞ Exemple : avec un octet (8 bits), on peut représenter (coder) les nombres appartenant à l'intervalle :

$$[0, 2^8 - 1] = [0, 255].$$

### II.6.2 – Codage des entiers relatifs (nombres signés)

La représentation des entiers signés pose des problèmes surtout au niveau de la représentation du signe. Il existe plusieurs manières pour coder les nombres signés :

- Codage en signe + valeur absolue
- Codage en complément restreint (C à 1)
- Complément vrai (C à 2)

Convention : Quelque soit le codage utilisé, le bit de poids fort est réservé pour la représentation du signe : Un nombre négatif a un bit de signe à 1 et un nombre positif a un bit de signe à 0.

#### A - Codage en signe + valeur absolue

Avec  $n$  bits, le  $n^{\text{ième}}$  bit est réservé pour le signe et les  $n-1$  bits restants sont utilisés pour la représentation de la valeur absolue du nombre à coder. Un codage sur  $n$  bits permet de coder tous les entiers appartenant à l'intervalle :  $[-(2^{n-1}-1), + (2^{n-1}-1)]$

☞ Exemple 1 : Avec 4 bits on peut coder les entiers relatifs suivants :

1111	1110	1101	1100	1011	1010	1001	1000	0000	0001	0010	0011	0100	0101	0110	0111
- 7	- 6	- 5	- 4	- 3	- 2	- 1	- 0	+ 0	+ 1	+ 2	+ 3	+ 4	+ 5	+ 6	+ 7

☞ Exemples 2 :

	Sur n = 4 bits	Sur n = 8 bits	Sur n = 16 bits
Intervalle des valeurs couvertes	[- 7, + 7]	[- 127, + 127]	[- 32767, + 32767]
Statut de la valeur zéro	La valeur zéro est 0000 = 1000	La valeur zéro est 00000000 = 10000000	La valeur zéro est = 0000 0000 0000 0000 = 1000 0000 0000 0000
Exemples	$+(3)_{10} = (0\ 011)_{SVA}$	$+(3)_{10} = (0\ 0000011)_{SVA}$	$+(3)_{10} = (0\ 000\ 0000\ 0000\ 0011)_{SVA}$
	$-(3)_{10} = (1\ 011)_{SVA}$	$-(3)_{10} = (1\ 0000011)_{SVA}$	$-(3)_{10} = (0\ 000\ 0000\ 0000\ 0011)_{SVA}$
	$+(15)_{10}$ et $-(15)_{10}$ impossible à représenter	$+(15)_{10} = (0\ 0001111)_{SVA}$	$+(15)_{10} = (0\ 000\ 0000\ 0000\ 1111)_{SVA}$
		$-(15)_{10} = (1\ 0001111)_{SVA}$	$-(15)_{10} = (1\ 000\ 0000\ 0000\ 1111)_{SVA}$

Bit de signe

7 bits pour coder la valeur absolue

Avantages : Facile à interpréter

Inconvénients: 2 représentations pour le 0 (+0 et -0) et Problème d'addition de deux nombres de signes opposés

## B - Codage en complément restreint (CR) ou complément à 1 (C à 1)

- On obtient le complément à 1 d'un nombre binaire en inversant (le 1 devient 0, et le 0 devient 1) chacun de ses bits.
- Les nombres positifs sont codés comme dans le codage en signe plus valeurs absolue (SVA).
- Les nombres négatifs sont déduits des nombres positifs par complémentation bit à bit, c'est-à-dire :  $(-N) = CR(N)$  (en supposant que N est un nombre positif).
- Un codage sur  $n$  bits permet de coder tout entier relatif appartenant à l'intervalle  $[-(2^{n-1}-1), + (2^{n-1}-1)]$

Exemple : coder +15 et -15 sur 8 bits en utilisant le codage en CR :

$$\begin{aligned} (+15)_{10} &= (00001111)_2 \\ (-15)_{10} &= CR(00001111) = (11110000)_{CR} \end{aligned}$$

Inconvénient : Deux représentations pour le zéro

## C - Codage en complément vrai (CV) ou complément à 2 (C à 2)

Pour obtenir le complément vrai d'un nombre entier, il suffit d'ajouter un 1 à son complément restreint :  $CV(N) = CR(N)+1$  où  $N$  est un entier quelconque.

En codage en complément à 2 :

- Un nombre positif est représenté de la même manière qu'en codage en signe plus valeurs absolue.
- Un nombre négatif est représenté par le complément vrai de son opposé (qui est bien évidemment positif)
- Un codage sur  $n$  bits permet de coder tout entier relatif appartenant à l'intervalle  $[-2^{n-1}, + (2^{n-1}-1)]$

☞ Exemple : Coder +15 et -15 sur 8 bits en utilisant le codage en CV

$$\begin{aligned} (+15)_{10} &= (00001111) \\ (-15)_{10} &= CR(00001111) + 1 = (11110001)_{CV} \end{aligned}$$

Avantage : Un seul codage pour 0 et pas de problème pour effectuer l'opération d'addition

Inconvénient : Difficile à interpréter.

## D - Soustraction par la méthode du complément

### Complément restreint

Pour une machine travaillant en complément restreint, la soustraction sera obtenue par l'addition du complément restreint du nombre à soustraire avec le nombre dont il doit être soustrait et report de la retenue (ajout de la retenue). S'il n'y a pas de retenue, cela signifie que le nombre est négatif. Il se présente sous la forme complémentée (complément restreint). Il suffira de retrouver le CR de ce résultat pour obtenir la valeur recherchée.

☞ Exemple 1 : Effectuer l'opération suivante en utilisant la technique CR sur 8 bits :  $(63)_{10} - (28)_{10}$ .

$$\begin{aligned} (63)_{10} &= (00111111)_2 \\ (28)_{10} &= (00011100)_2 \\ CR(28) &= CR(00011100) = (11100011)_{CR} \end{aligned}$$

Dans cet exemple, il y a une retenue, alors on l'a rajouté au résultat ce qui a donné un nombre positif :  $(63)_{10} - (28)_{10} = (+35)_{10}$ .

$$\begin{array}{r} \phantom{0} 00111111 \leftarrow (63)_{10} \\ + \phantom{0} 11100011 \leftarrow CR(28) \\ \hline \text{Retenue} = 1 \phantom{0} 0100010 \\ + \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} 1 \\ \hline = \phantom{0} 0100011 \leftarrow \text{Résultat final } (+35)_{10} \end{array}$$

**Exemple 2 :** Effectuer l'opération suivante en utilisant la technique CR sur 8 bits :

$$(28)_{10} - (63)_{10}$$

$$(63)_{10} = (00111111)_2 \text{ et } (28)_{10} = (00011100)_2.$$

$$CR(63) = CR(00111111) = (11000000)_{CR}.$$

0 0011100	← (28) <sub>10</sub>
1 1000000	← CR(63)
<hr/>	
= 1 1011100	← Résultat final (-35) <sub>10</sub> .

Dans cet exemple, il n'y a pas de retenue, le résultat est négatif, alors on calcule son CR :  $CR(11011100) = (00100011)_2 = (35)_{10}$  confirmant l'égalité :  $(28)_{10} - (63)_{10} = (-35)_{10}$ .

### Complément vrai

Le principe est le même que pour le CR sauf que cette fois-ci on ignore la retenue. Au lieu de travailler avec des CR, on détermine des compléments vrais.

☞ **Exemple 1 :** Effectuer l'opération suivante en utilisant la technique CR sur 8 bits :

$$(63)_{10} - (28)_{10} = (63)_{10} + CV(28)$$

$$(63)_{10} = (00111111)_2$$

$$(28)_{10} = (00011100)_2.$$

$$CV(28) = CR(00011100) + 1 = (11100100)_{CR}.$$

00111111	← (63) <sub>10</sub>
11100100	← CR(28)
<hr/>	
Retenue → 1	= 00100011 ← Résultat final (+35) <sub>10</sub>

Dans cet exemple, il y a une retenue, alors il faut l'ignorer. On obtient un résultat positif :  $(63)_{10} - (28)_{10} = (+35)_{10}$ .

☞ **Exemple 2 :** Effectuer l'opération suivante en utilisant la technique CV sur 8 bits :

$$(28)_{10} - (63)_{10} = (28)_{10} + CV(63)$$

$$(63)_{10} = (00111111)_2$$

$$(28)_{10} = (00011100)_2.$$

$$CV(63) = CR(00111111) + 1 = (11000001)_{CV}.$$

0 0011100	← (28) <sub>10</sub>
+ 1 1000001	← CV(63)
<hr/>	
= 1 1011101	← Résultat final (-35) <sub>10</sub> .

Dans cet exemple, il n'y a pas de retenue, le résultat est négatif alors on calcule son CV :  $CV(11011101) = CR(11011101) + 1 = 00100010 + 1 = 00100011$ . On obtient au final :  $(28)_{10} - (63)_{10} = (-35)_{10}$ .

### E - Problèmes liés à la longueur des nombres

**Rappel :** En complément vrai (ou à 2) sur  $n$  bits, les nombres sont compris entre  $-2^{n-1}$  et  $(2^{n-1} - 1)$ .

**Addition de deux nombres positifs :** En additionnant deux nombres positifs, on peut obtenir un résultat négatif (le bit de signe du résultat à 1). Ceci est dû au fait que le résultat n'appartient pas à l'intervalle autorisé avec le nombre de bits prévu.

☞ **Exemple :** Effectuer l'opération suivante en utilisant la technique CV sur 8 bits :  $(+49)_{10} + (88)_{10}$

Dans cet exemple nous avons additionné deux nombres positifs, tous les deux tenant sur 8 bits. Malheureusement, nous avons obtenu un résultat qui est situé en dehors de l'intervalle des valeurs autorisées pour le codage sur 8 bits.

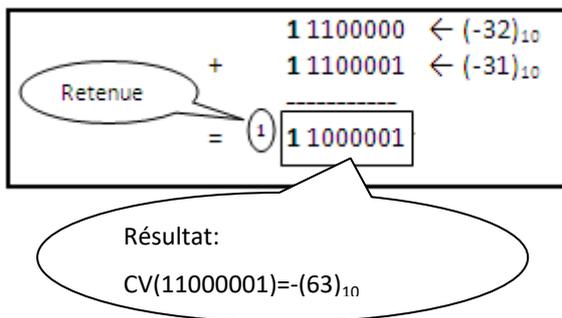
$[-2^{n-1}, + (2^{n-1} - 1)] = [-128, +127]$  avec  $n = 8$ . En effet, le résultat de 137 ( $49+88=137$ ) est en dehors de cet intervalle.

0 0110001	← (+49) <sub>10</sub>
+ 0 1011000	← (88) <sub>10</sub>
<hr/>	
= 1 0001001	

Dépassement de capacité

**Addition de deux nombres négatifs :** En additionnant deux nombres négatifs représentés par leurs CV (bit de signe à 1), on peut obtenir un résultat positif (le bit de signe du résultat à 0). En effet, il y a toujours une retenue car les bits de poids fort des nombres à additionner sont à 1.

☞ Exemple 1 : Effectuer l'opération suivante en utilisant la technique CV sur 8 bits :  $(-32)_{10} + (-31)_{10}$

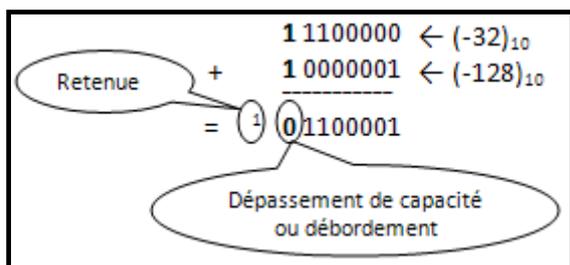


Dans cet exemple, nous avons additionné deux nombres négatifs (voir le bit de signe qui est à 1) et nous avons obtenu un nombre négatif (voir le bit de signe à 1). Malgré que nous avons obtenue une retenue, notre résultat est correcte, il faut juste ignorer cette retenue (car nous utilisons ici un codage en complément à 2 ou complément vrai).

☞ Exemple 2 : Effectuer l'opération suivante en utilisant la technique CV sur 8 bits :  $(-32)_{10} + (-128)_{10}$

En ignorant la retenue, on obtient un résultat positif (bit de signe à 0) donc, on déduit qu'il y a débordement ou dépassement de capacité.

En décimal :  $(-32)_{10} + (-127)_{10} = (-159)_{10}$  -159 n'est pas situé dans l'intervalle  $[-128 \text{ et } +127]$ .



**Indicateur de dépassement de capacité :** Les calculateurs utilisent un indicateur de dépassement de capacité (*Overflow*) qui est mis à 1 si le bit de signe du résultat est 0 alors que les deux nombres à additionner sont négatifs ou lorsque le bit de signe du résultat est à 1 alors que les deux nombres à additionner sont positifs.

## II.6.3 – Codage des réels

### A Virgule fixe

### B virgule flottante

