

Représentation et codage

Introduction

Quelle que soit la nature de l'information traitée par un ordinateur (image, son, texte, video), elle l'est toujours sous la forme d'un ensemble de nombres écrits en base 2, par exemple 1001011. Le terme bit (b minuscule dans les notations) signifie « binary digit », c'est-à-dire 0 ou 1 en numérotation binaire. Il s'agit de la plus petite unité d'information manipulable par une machine numérique. Il est possible de représenter physiquement cette information binaire par un signal électrique ou magnétique, qui, au-delà d'un certain seuil, correspond à la valeur 1.

Aujourd'hui nos ordinateurs, téléphones et autres appareils savent manipuler aussi bien des nombres et du texte que des images, de la vidéo ou de la musique... Mais comment représenter, au sein d'un système numérique, cette diversité des objets du monde réel ou virtuel ?

Quelles sont les techniques utilisées pour représenter numériquement les grandeurs qui nous entourent ?

A. Définitions

UNITÉ DE CODAGE

Les composants constituant un système informatique réagissent, de manière interne, à des signaux « **Tout Ou Rien** ». On représente les deux états stables ainsi définis par les symboles « **0** » et « **1** » ou encore par « **L** » (Low) et « **H** » (High).

Le système de numération adaptée à la représentation de tels signaux est **la base 2**, on parle alors de **codage binaire**.

L'unité de codage de l'information est un élément ne pouvant prendre que les valeurs 0 ou 1. C'est le **bit** (contraction de **binary digit**).

UNITÉ DE TRANSFERT

Pour les échanges de données, les informations élémentaires (bits) sont manipulées par groupes qui forment ainsi des mots binaires. La taille de ces mots est le plus souvent un multiple de 8.

L'unité de transfert utilisée pour les échanges de données est le mot de 8 bits appelé **octet**.

Exemples : (2 octets)

1111 0011
1010 1111

Remarques :

Un octet est **un byte** particulier contenant 8 bits.

Pour faciliter les manipulations, un octet peut être divisé en deux mots de 4 bits que l'on appelle des **quartets** : celui situé à gauche est le quartet de poids fort, **MSQ (Most Significant Quartet)**, et celui situé à droite, le quartet de poids faible, **LSQ (Less Significant Quartet)**.

Exemples :

MSQ				LSQ			
1	0	0	1	1	1	1	0
quartet de poids fort				quartet de poids faible			
octet							

Remarque :

Un quartet est un byte particulier contenant 4 bits.

MOTS BINAIRES

Dans un mot binaire, le bit situé le plus à gauche est le bit de poids fort, **MSB (Most Significant Bit)**, celui situé le plus à droite est le bit de poids faible, **LSB (Less Significant Bit)**.

Exemple :

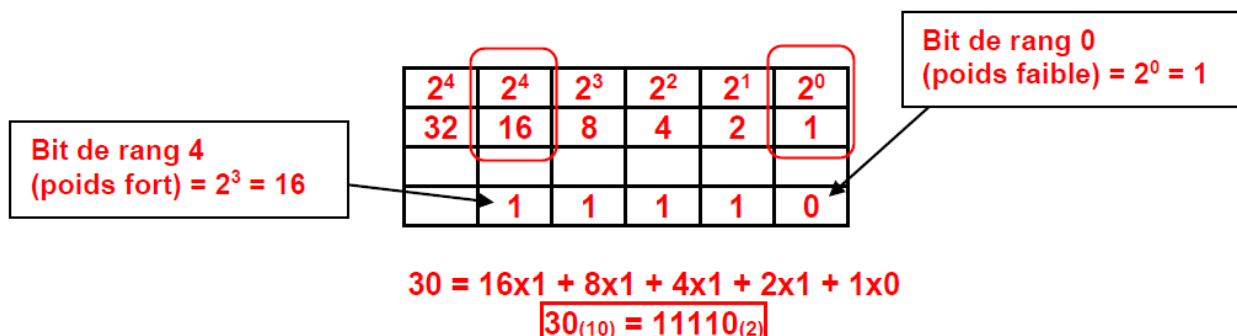
MSB																LSB	
1	0	0	0	1	1	1	0	1	0	0	0	0	0	1	1		
octet de poids fort								octet de poids faible									
mot (16 bits)																	

Beaucoup d'informaticiens ont appris que 1 kilooctet = 1024 octets. Or, depuis décembre 1998, l'organisme international IEC a statué.

Préfixes décimaux (SI)	ko (kB) =	kilo-octet (kiloByte)	=	10^3 octets	=	1000 octets
	Mo (MB) =	Méga-octet (MegaByte)	=	10^6 octets	=	1000 ko
	Go (GB) =	Giga-octet (GigaByte)	=	10^9 octets	=	1000 Mo
	To (TB) =	Téra-octet (TeraByte)	=	10^{12} octets	=	1000 Go
Préfixes binaires (IEC)	kio (kiB) =	kibi-octet (kibiByte)	=	2^{10} octets	=	1024 octets
	Mio (MiB) =	Mébi-octet (MebiByte)	=	2^{20} octets	=	1024 kio
	Gio (GiB) =	Gibi-octet (GibiByte)	=	2^{30} octets	=	1024 Mio
	Tio (TiB) =	Tébi-octet (TebiByte)	=	2^{40} octets	=	1024 Gio

IEC : International Electrotechnical Commission

La capacité en octets des différents constituants tels que circuits mémoires, disques durs, ... est souvent importante : il devient indispensable d'utiliser **des unités multiples de l'octet**. En dehors de l'unité de transfert (octet), des regroupements plus importants sont couramment utilisés : **le mot de 16 bits** = 2 octets (word), **le mot de 32 bits** = 4 octets (double word), et **le mot de 64 bits** = 8 octets (quad word)...



DÉCIMALE → HEXADÉCIMALE

Le binaire, s'il est très représentatif du codage interne des machines, reste très délicat et fastidieux à manipuler. Les programmeurs ont très vite ressenti la nécessité d'utiliser une représentation plus rapide des nombres binaires.

Imaginons de représenter chaque quartet binaire par un unique symbole. Un quartet permettant de coder 24 valeurs (soit de 0 à 15). Il faut donc trouver une base de représentation disposant de 16 symboles : il s'agit de **la base 16** (appelée base hexadécimale).

Notation :

Des indices ou des préfixes peuvent être utilisés pour les nombres hexadécimaux : 15(16), 23(HEX), 0x55F, \$AF4, &h38, #44B.

Remarques :

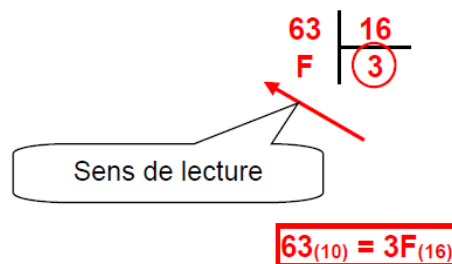
Le préfixe 0x est utilisé dans le langage C, C++ et JAVA ; le \$ est utilisé dans le langage Pascal ; le &h dans le langage Basic et le # dans le HTML.

Une autre écriture courante est l'ajout du suffixe « h » à la fin du nombre (F15Ah par exemple). Dans le système hexadécimale les dix premiers symboles correspondent à ceux utilisés dans le système décimal : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9, et les six derniers correspondent aux premières lettres de l'alphabet latin : A, B, C, D, E et F, lesquelles valent respectivement 10, 11, 12, 13, 14 et 15 en base 10.

Pour coder un nombre décimal en hexadécimale, on peut utiliser plusieurs méthodes.

Exemple : codage des nombres 63₍₁₀₎ et 80₍₁₀₎ en hexadécimale.

1^{ère} méthode : la division successive par 16



2^{ème} méthode : le tableau

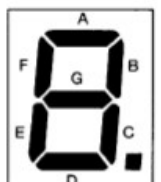
16 ²	16 ¹	16 ⁰
256	16	1
	5	0

$$80 = 5 \times 16 + 0 \times 1$$

80₍₁₀₎ = 50₍₁₆₎

DÉCIMALE * BCD (OU DCB)

Sens de lecture DCB signifie Décimal Codé Binaire. Ce code est utilisé principalement pour les afficheurs 7 segments. On le retrouve également pour l'adressage et les données des capteurs selon les constructeurs sur les bus série (I2C, SPI, ...).

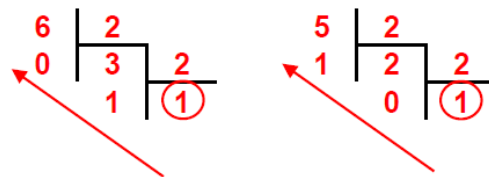


Il faut ici coder les chiffres décimaux individuellement afin d'obtenir pour chaque chiffre décimal son équivalent codé en binaire sur 4 bits (quartet).

Pour coder un nombre décimal en BCD, on peut utiliser plusieurs méthodes.

Exemple : codage des nombres $65_{(10)}$ et $78_{(10)}$ en BCD.

1^{ère} méthode : la division successive par 2



$$65_{(10)} = 01100101_{(BCD)}$$

2^{ème} méthode : le tableau

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
8	4	2	1	8	4	2	1
0	1	1	1	1	0	0	0

$$7 = 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$$

$$8 = 1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1$$

$$78_{(10)} = 01111000_{(BCD)}$$

DES CARACTÈRES (ASCII)

Une grosse part des informations manipulées par les systèmes numériques concerne le langage parlé ou écrit matérialisé sous formes de textes, eux-mêmes constitués de caractères typographiques. Comment coder universellement ces caractères et permettre ainsi l'échange d'informations entre machines et/ou utilisateurs, quelle que soit la langue utilisée ?

Remarque :

Le morse inventé en 1844 est le premier codage à permettre une communication orientée caractère à longue distance. Ce code est composé de points et de tirets (une sorte de codage binaire).

SOS : . . . - - - . . .

Le jeu de caractères codés ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) ou code américain normalisé pour l'échange d'informations, inventé par Bob BERNER en 1961, est la norme de codage de caractères en informatique la plus connue, la plus ancienne et la plus largement compatible.

Le code ASCII est un code sur 8 bits (valeurs 0 à 255), il permet de définir :

- des caractères imprimables universels : lettres minuscules et majuscules, chiffres, symboles,...
- des codes de contrôle non imprimables : indicateur de saut de ligne, de fin de texte, codes de contrôle de périphériques, ...

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Complément : <http://www.table-ascii.com/>

D. Exercices d'applications

EXERCICE N°1

À l'aide de la table ASCII ci-dessus, « décrypter » la chaîne ASCII ci-dessous représentée sous la forme d'une suite d'octets.

01010011 01001110 00101101 01001001 01010010

.....

EXERCICE N°2

Retrouver la chaîne ASCII sous la forme décimale du texte ci-dessous.

Texte à coder en ASCII : **J'aime l'informatique !**

.....

UNICODE

Pour coder de manière universelle l'ensemble des symboles utilisés quelque soit la langue (anglais, français, grec, chinois,...) il faut attribuer à tout caractère ou symbole de n'importe quel système d'écriture de langue un nom et un identifiant numérique, et ce de manière unifiée, quelle que soit la plate-forme informatique ou le logiciel.

C'est ce que propose la norme Unicode (www.unicode.org).

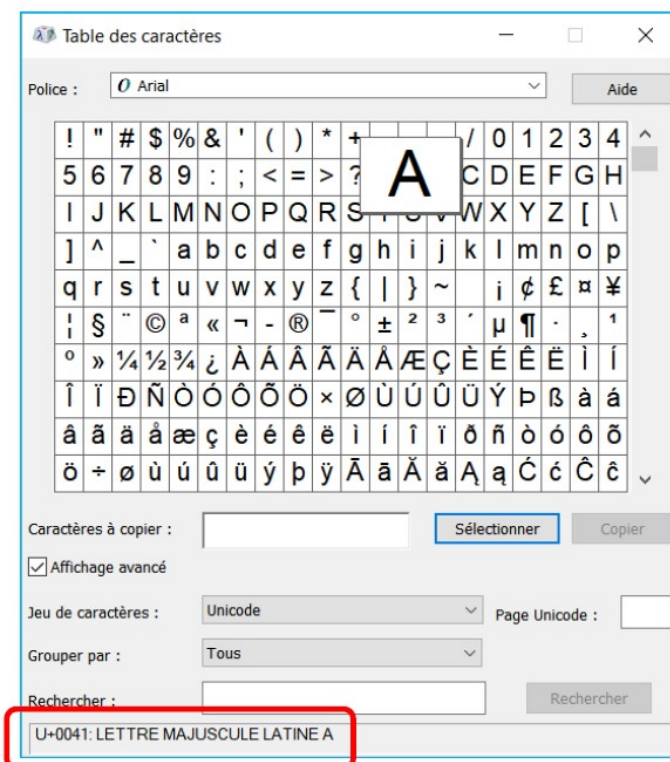
Exemples :

Chaque symbole d'écriture est représenté par un nom et une valeur hexadécimale préfixée par « U+ ».

A « lettre majuscule latine A » **U+0041**

é « lettre minuscule latine e accent aigü » **U+00E9**

€ « symbole euro » **U+20AC**



Pour stocker sur un support informatique un texte constitué de caractères Unicode, il faut encore choisir un procédé transformant chaque définition Unicode en une suite d'octets et réciproquement... C'est le **processus d'encodage**.

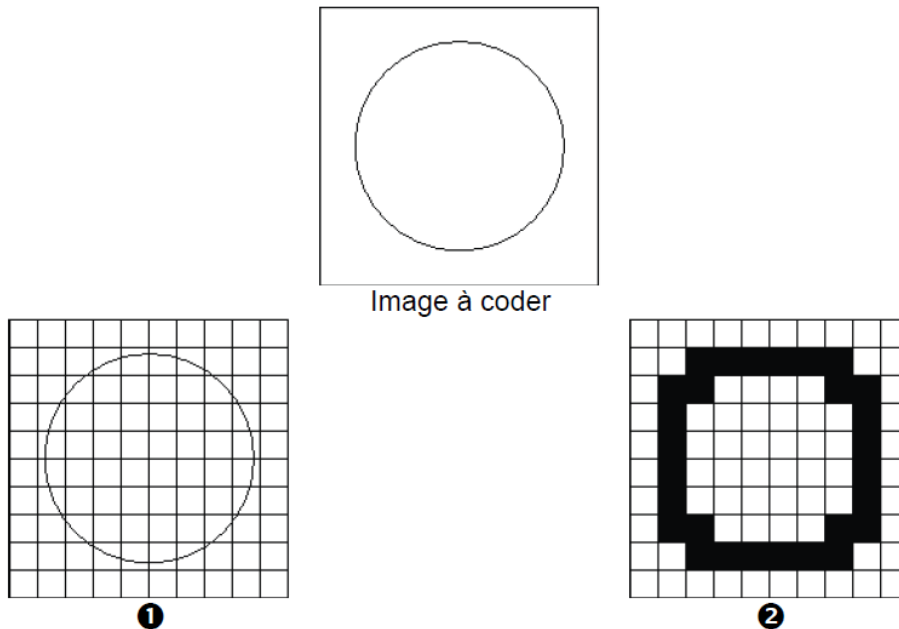
Actuellement, un des systèmes d'encodage couramment utilisés est **UTF-8** (Unicode Transformation Format).

DES IMAGES

Après le texte, l'image est un support très utilisé pour communiquer.

Exemple :

Pour coder l'image ci-dessous, il suffit de superposer un quadrillage à l'image (❶). Chacune des cases de ce quadrillage s'appelle un **pixel** (picture element). On noircit ensuite les pixels qui contiennent une portion de trait (❷).



Puis, il suffit d'indiquer la couleur de chacun des pixels, en les lisant de gauche à droite et de haut en bas, comme un texte. Ce dessin se décrit donc par une suite de mots « blanc » ou « noir ». Comme seuls les mots « noir » ou « blanc » sont utilisés, on peut être plus économe et remplacer chacun de ces mots par un bit, par exemple 1 pour « noir » et 0 pour « blanc ».

L'image (❷), avec une grille de 10×10 , se décrit alors par la suite de 100 bits suivante :

```
0000000000001111110001100001100100000010010000001001000000100100000010
011000011000111111000000000000
```

Cette description est assez approximative, mais on peut la rendre plus précise en utilisant un quadrillage, non plus de 10×10 pixels, mais de 100×100 pixels. À partir de quelques millions de pixels, notre œil n'est plus capable de faire la différence entre les deux images.

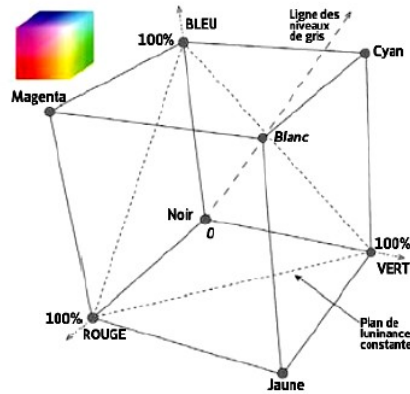
Cette manière de représenter une image sous la forme d'une suite de pixels, chacun exprimés sur un bit, s'appelle une **bitmap**. C'est une méthode approximative, mais universelle: n'importe quelle image en noir et blanc peut se décrire ainsi.

DE LA COULEUR

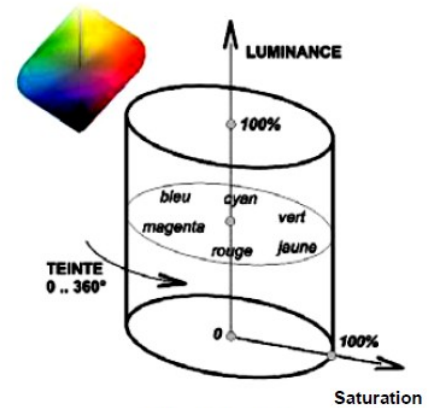
Le mode de représentation **RVB** (Rouge, Vert et Bleu, ou en anglais RGB) correspond à celui fourni par la plupart des caméras couleur; il est naturellement utilisé pour la reproduction de couleurs sur écran (base noire). C'est le mode de composition des couleurs basé sur le principe des couleurs **additives** : le rouge, le vert et le bleu sont les trois primaires utilisés dans la constitution de couleurs à partir de sources lumineuses.



Synthèse additive



Modèle RVB



Modèle TSL

Saturation

Le modèle **TSL** (Teinte-Saturation-Luminance) est un autre modèle plus proche de la perception humaine des couleurs. Ses coordonnées se calculent à partir des proportions RVB. La composition de la teinte et de la saturation est appelée **chrominance**. Pour représenter les couleurs, les écrans utilisent le codage RVB (Rouge, Vert, Bleu). Ce système, basé sur la synthèse additive des couleurs, représente chacune d'entre elles par ses niveaux de rouge, vert et bleu. Chacun de ces niveaux est codé par un nombre allant de 0 à 255. On retrouve donc $256 \times 256 \times 256 = 16\,777\,216$ couleurs.

Exemple : (codage RVB)

Pixel blanc * codage RVB = #FFFFFF

Le système **CMJN** (Cyan-Magenta-Jaune-Noir) est réservé aux périphériques d'impression, qui travaillent en **synthèse soustractive**. La quatrième couleur (Noir) sert essentiellement à imprimer la gamme des gris. La reproduction du noir par mélange des trois primaires n'est jamais parfaite. De plus, cela permet d'économiser les encres de couleur lors de l'impression de documents en niveaux de gris.

DE LA PROFONDEUR D'IMAGE

Les images codées sur 24 bits sont dites en vraies couleurs (true color) ; une composante Alpha permettant d'inclure une information de transparence peut être ajoutée à ce type de codage, chaque pixel est alors codé sur 32 bits.

Le terme de profondeur d'image est utilisé pour spécifier le nombre de bits alloué au codage de chaque pixel ; les valeurs courantes de profondeur sont 1 (image binaire), 8 (256 couleurs ou niveaux de gris) et 32 (vraies couleurs avec canal alpha).

E. Exercices d'applications

Quelle est la taille (en octets) d'une image non compressée, de définition 640 x 480 et de profondeur 24 bits (RVB) ?

.....

L'image est incorporée à un document destiné à être distribué sous forme de photocopies noir et blanc. Quelle économie de taille réalisez-vous en convertissant l'image en 256 niveaux de gris ?

.....

ENTIER RELATIF

Jusqu'à présent nous avons codé des entiers naturels qui sont des nombres entiers positif ou nul. Pour coder des nombres entiers naturels compris entre 0 et 255, il nous suffira de 8 bits (un octet) car $2^8 = 256$. D'une manière générale un codage sur n bits pourra permettre de représenter des nombres entiers naturels compris entre 0 et $2^n - 1$.

Un entier relatif est un entier pouvant être négatif. Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif, et il faut de plus que les règles d'addition soient conservées.

Le bit le plus significatif est utilisé pour représenter le signe du nombre :

- si MSB = 1 alors nombre négatif
- si MSB = 0 alors nombre positif

Les autres bits codent la valeur absolue du nombre

Exemple :

Sur 8 bits, codage des nombres -24₍₁₀₎, 24₍₁₀₎ et -128₍₁₀₎ en binaire signé :

-24 est codé en binaire signé par : 10011000(bs)

24 est codé en binaire signé par : 00011000(bs)

-128 hors limite nécessite 9 bits au minimum

Étendu de codage :

Avec n bits, on code tous les nombres entre $-(2^{n-1}-1)$ et $(2^{n-1}-1)$

Avec 4 bits : -7 à +7

F. Exercices d'applications

Question 1

Codez 100₍₁₀₎ et -100₍₁₀₎ en binaire signé sur 8 bits.

.....

.....

Question 2

Décodez en décimal 11000111_(bs sur 8 bits) et 1101_(bs sur 4 bits)

.....

.....

COMPLÉMENT À 1

Aussi appelé Complément Logique (CL) ou Complément Restreint (CR) :

- les nombres positifs sont codés de la même façon qu'en binaire pure.
- un nombre négatif est codé en inversant chaque bit de la représentation de sa valeur absolue
- le bit le plus significatif est utilisé pour représenter le signe du nombre :

- si MSB = 1 alors le nombre est négatif
- si MSB = 0 alors le nombre est positif

Exemple :

-24 en complément à 1 sur 8 bits :

$|-24|$ en binaire pur 00011000(2)

Puis on inverse (on complémente) les bits = 11100111(cà1)

G. Exercices d'applications

Codez $100_{(10)}$ et $-100_{(10)}$ par complément à 1 sur 8 bits.

.....

Décodez en décimal $11000111_{(cà1 \text{ sur } 8 \text{ bits})}$ et $00001111_{(cà1 \text{ sur } 8 \text{ bits})}$.

.....

Remarque :

Le 4 juin 1996, une fusée Ariane 5 a explosé 40 secondes après l'allumage. La fusée et son chargement avaient coûté 500 millions de dollars. La commission d'enquête a rendu son rapport au bout de deux semaines. Il s'agissait d'une erreur de programmation dans le système inertiel de référence. À un moment donné, un nombre codé en virgule flottante sur 64 bits (qui représentait la vitesse horizontale de la fusée par rapport à la plate-forme de tir) était converti en un entier sur 16 bits. Malheureusement, le nombre en question était plus grand que 32767 (le plus grand entier relatif que l'on puisse coder sur 16 bits) et la conversion a été incorrecte.

COMPLÉMENT À 2

Aussi appelé Complément Vrai (CV) :

- les nombres positifs sont codés de la même manière qu'en binaire pure
- un nombre négatif est codé en ajoutant la valeur 1 à son complément à 1
- le bit le plus significatif est utilisé pour représenter le signe du nombre

Exemple :

-24 en complément à 2 sur 8 bits :

24 est codé par $00011000_{(2)}$

-24 donne $11100111_{(cà1)}$

donc -24 est codé par $11101000_{(cà2)}$

Remarque :

Pour transformer de tête un nombre binaire en son complément à deux, on parcourt le nombre de droite à gauche en laissant inchangés les bits jusqu'au premier 1 (compris), puis on inverse tous les bits suivants. Prenons comme exemple le nombre $20_{(10)}$: $00010100_{(2)}$.

On garde la partie à droite telle quelle : **00010100**

On inverse la partie de gauche après le premier un : **11101100**

Donc $-20_{(10)}$ donne $11101100_{(cà2)}$

H. Exercices d'applications

Codez $100_{(10)}$ et $-100_{(10)}$ par complément à 2 sur 8 bits.

.....

Décodez en décimal $11001001_{(cà2 \text{ sur } 8 \text{ bits})}$ et $01101101_{(cà2 \text{ sur } 8 \text{ bits})}$

.....

NOMBRE RÉEL

Les formats de représentations des nombres réels sont :

✓ Format virgule fixe :

utilisé par les premières machines possède une partie 'entière' et une partie 'décimale' séparées par une virgule. La position de la virgule est fixée d'où le nom.

Exemples : 54,25(10) ; 10,001(2)

✓ Format virgule flottante (utilisé actuellement sur machine) :

défini par : $\pm m . b^e$

un signe + ou -

une mantisse m (en virgule fixe)

une base b (2,8,10,16,...)

un exposant e (un entier relatif)

Exemples : $0,5425 \cdot 10^2(10)$; $10,1 \cdot 2^{-1}(2)$

VIRGULE FIXE

Exemple : base 2 vers base 10

$101,01(2) = ?(10)$

$101,01(2) = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 5,25(10)$

Exemple : base 10 vers base 2

$4,625(10) = ?(2)$

$4(10) = 100(2)$

$0,625 \times 2 = 1,250$ ce qui donne **1**

$0,250 \times 2 = 0,500$ ce qui donne **0**

$0,500 \times 2 = 1,000$ ce qui donne **1**

donc $4,25(10) = 100,101(2)$

Attention ! Un nombre à développement décimal fini en base 10 ne l'est pas forcément en base 2.

Cela peut engendrer de mauvaises surprises.

Remarque :

Le 25 février 1991, pendant la Guerre du Golfe, une batterie américaine de missiles Patriot, à Dhahran (Arabie Saoudite), a échoué dans l'interception d'un missile Scud irakien. Le Scud a frappé un baraquement de l'armée américaine et a tué 28 soldats. La commission d'enquête a conclu à un calcul incorrect du temps de parcours, dû à un problème d'arrondi. Les nombres étaient représentés en virgule fixe sur 24 bits. Le temps était compté par l'horloge interne du système en dixième de seconde. $1/10$ n'a pas d'écriture finie dans le système binaire : $1/10 = 0,1$ (dans le système décimal) $= 0,0001100110011001100110...$ (dans le système binaire). L'ordinateur de bord arrondissait $1/10$ à 24 chiffres, d'où une petite erreur dans le décompte du temps pour chaque $1/10$ de seconde. Au moment de l'attaque, la batterie de missile Patriot était allumée depuis environ 100 heures, ce qui a entraîné une accumulation des erreurs d'arrondi de 0,34 s. Pendant ce temps, un missile Scud parcourt environ 500 m, ce qui explique que le Patriot soit passé à côté de sa cible.

VIRGULE FLOTTANTE (NORME IEEE 754)

La norme IEEE 754 définit la façon de coder un nombre réel. Cette norme se propose de coder le nombre sur 32 bits et définit trois composantes :

- le signe est représenté par un seul bit, le bit de poids fort
- l'exposant est codé sur les 8 bits consécutifs au signe
- la mantisse (les bits situés après la virgule) sur les 23 bits restants

Certaines conditions sont toutefois à respecter pour les exposants :

- l'exposant 00000000 est interdit.
- l'exposant 11111111 est interdit. On s'en sert toutefois pour signaler des erreurs, on appelle alors cette configuration du nombre NaN, ce qui signifie « Not a number ».
- il faut rajouter 127 (01111111) à l'exposant pour une conversion de décimal vers un nombre réel binaire. Les exposants peuvent ainsi aller de -254 à 255.

La formule d'expression des nombres réels est ainsi la suivante :

$$(-1)^S * 2^{(E - 127)} * (1 + F)$$

S est le bit de signe et l'on comprend alors pourquoi 0 est positif ($-1^0 = 1$), E est l'exposant auquel on doit bien ajouter 127 pour obtenir son équivalent code, F est la partie fractionnaire.

Exemple :

Traduisons en binaire, en utilisant la norme IEEE 754, le nombre -6,625(10).

Codons d'abord la valeur absolue en binaire : 6,625(10) = 110,1010(2)

Nous mettons ce nombre sous la forme : 1, **partie fractionnaire**

110,1010 = 1, **101010** · 2² (2² décale la virgule de 2 chiffres vers la droite)

La partie fractionnaire étendue sur 23 bits est donc **101 0100 0000 0000 0000 0000**.

Exposant = 127 + 2 = 129(10) = 1000 0001(2)



Signe	Exposant	Mantisse
1	1 0 0 0 0 0 0 1	1 0 1 0 1 0

I. Exercices de synthèse

Question 1
Complétez le tableau page suivante.

Base 2	Base 10	Base 16	BCD	ASCII (char)	Unicode
11100001					
	123				
		7E			
			101000100		
				&	
					U+0023

Question 2
Réalisez une bitmap 20 x 20 pixels de l'image ci-dessous et donnez la suite de bit correspondante.

	
Image en noir et blanc de dimensions : 5 cm x 5 cm	Image à « Bitmaper »

.....

.....

.....

Question 3

Complétez le tableau ci-dessous.

Base 10	bs	cà1	cà2
	sur 8 bits		
123			
-96			

Question 4

Complétez le tableau ci-dessous.

	IEEE 754	Base 10	Base 2
12,855 ₍₁₀₎			
%110,11			

Question 5a

La notice technique d'un ordinateur portable donne les indications suivantes :
mémoire vive 2048 Mio, écran SXGA (Super eXtended Graphics Array, 1 280 x 1 024 pixels),
disque dur 250 Go.

Calculez, en octets, la capacité de la mémoire vive.

.....

Question 5b

Calculez la taille en kio de la plus grande image RGBA que cette machine puisse afficher.

.....

Question 5c

Calculez, en Mio, la taille d'une partition égale à 40% de la capacité totale du disque dur.

.....

.....

Question 6

Un système de localisation est basé sur un récepteur GPS. Les trames envoyées par le récepteur sont constituées de caractères ASCII imprimables. Une latitude de 21° 14' 21" Sud est par exemple envoyée sous la forme de la suite de caractères 2114.35,S qui correspondent à 21 degrés et 14,35 minutes (0,35 x 60 = 21").

La chaîne de caractères (en hexadécimale) ci-dessous indique une longitude :

35 35 34 30 2E 33 30 35 2C 45

Décodez cette coordonnée GPS (en degrés, minutes, secondes).

.....

.....