Informatique 1M

A. Ridard

Représentation de l'information

Gymnase Yverdon

canton de vaud

Table des matières

I. Introduction

Dans ce chapitre on s'intéresse à la manière dont un ordinateur représente l'information afin de pouvoir la traiter automatiquement.

Le mot informatique est la concaténation de « information » et « automatique ».

Dans un ordinateur, toutes les informations [1] sont représentées à l'aide de deux chiffres 0 et 1, appelés chiffres binaires (binary digits) ou plus simplement *bits*.

Dans la mémoire d'un ordinateur, ces chiffres binaires sont regroupés en *octets* (bytes) c'est à dire en « paquets » de 8, puis organisés en *mots machine* (words) de 2, 4 ou 8 octets (pour les machines les plus courantes aujourd'hui, dites de 64 bits).

Ce regroupement des bits en octets ou mots machine permet de représenter (et manipuler) d'autres données que des 0 ou des 1, comme par exemple des nombres (entiers et approximation de réels), des caractères, mais aussi des images, des sons et des vidéos!

Pour cela, il est nécessaire de disposer d'encodages pour représenter ces informations.

II. Les entiers

1. Les entiers naturels

L'encodage le plus simple est celui des nombres entiers naturels.
Il consiste à *interpréter* un octet ou un mot machine comme un entier écrit en base 2.

Rappel : écriture en base 10

Un entier naturel en base 10 est une séquence de chiffres entre 0 et 9.
Chaque chiffre est associé à une puissance de 10 selon sa position [2].

séquence	6	1	0	2	7
position	4	3	2	1	0
poids	10^4	10^3	10^2	10^1	10^0

La valeur de la séquence 61027 est l'entier n calculé de la manière suivante :

$$n = 6 \cdot 10^4 + 1 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0$$

Le chiffre le plus à droite est dit de poids faible, et celui le plus à gauche de poids fort.

Écriture en base 2

Un entier naturel en base 2 est une séquence de chiffres binaires (0 ou 1).
Chaque chiffre est associé à une puissance de 2 selon sa position [3].

séquence	0	1	0	0	1	1	0	1
position	7	6	5	4	3	2	1	0
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

La valeur de la séquence 01001101 est l'entier n calculé de la manière suivante :

$$n = 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 77$$

[1]. Données mais aussi programmes
[2]. De la droite vers la gauche, en commençant par la position 0
[3]. De la droite vers la gauche, en commençant par la position 0

Le bit le plus à droite est dit de poids faible, et celui le plus à gauche de poids fort.

Donner la valeur (décimale) des séquences binaires suivantes :

1. 10101101
2. 01110010
3. 1111
4. 11111111

Le passage de la base 10 à la base 2 n'est pas aussi immédiat.

On peut effectuer des divisions euclidiennes successives ou bien utiliser le tableau de conversion suivant et la méthode décrite ci-dessous.

poids	...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
valeur	...	128	64	32	16	8	4	2	1
bit	...								

Méthode pour compléter le tableau de conversion

1. Déterminer le plus grand poids inférieur ou égal à l'entier à convertir
2. Le bit associé à ce poids est 1
3. Soustraire ce poids à l'entier à convertir
4. Recommencer avec ce nouvel entier tant qu'il est différent de 0
5. En complétant les cases vides [a] par des 0, on obtient le résultat

a. A droite du bit de poids fort

Écrire en base 2 les entiers suivants :

1. 14
2. 218
3. 42
4. 57

Écriture en base 16 (hexadécimale)

Un entier naturel en base 16 est une séquence de chiffres hexadécimaux :

$$0, 1, ..., 9, A, B, ..., F$$

La valeur de chaque lettre est donnée par le tableau suivant :

lettre	A	B	C	D	E	F
valeur	10	11	12	13	14	15

Chaque chiffre est associé à une puissance de 16 selon sa position [4].

séquence	2	A	4	D
position	3	2	1	0
poids	16^3	16^2	16^1	16^0

[4]. De la droite vers la gauche, en commençant par la position 0

La valeur de la séquence 2A4D est l'entier n calculé de la manière suivante :

$$n = 2 \cdot 16^3 + 10 \cdot 16^2 + 4 \cdot 16^1 + 13 \cdot 16^0 = 10829$$

La base 16 est souvent utilisée pour simplifier l'écriture en base 2. En effet, on passe facilement de la base 2 à la base 16 en regroupant les chiffres binaires par 4 :

$$\overbrace{1010}^{A}\ \overbrace{0101}^{5}\ \overbrace{1111}^{F}\ \overbrace{0011}^{3}$$

Inversement, pour passer de la base 16 à la base 2, il suffit d'écrire chaque chiffre hexadémical en binaire avec 4 bits.

On pourra utiliser la notation indicielle pour préciser la base utilisée :

- $101_2 = 5$
- $101_{16} = 257$

Donner la valeur (décimale) des séquences hexadécimales suivantes :

1. 31
2. 4D

Transformer en binaire les séquences hexadécimales suivantes :

1. 15
2. 6DF

2. Les entiers relatifs

L'encodage des entiers relatifs est plus délicat.

L'idée principale est d'utiliser le bit de poids fort pour représenter le signe :
0 si l'entier est positif et 1 s'il est négatif.

Pour savoir quel est le bit de poids fort, l'encodage doit précisé la longueur des mots binaires utilisés (4 bits, 8 bits, 16 bits, ...)

Ce simple encodage qui permet de représenter, **sur 4 bits**, les entiers relatifs compris entre -7 et 7 ne se comporte malheureusement pas bien vis à vis de l'addition :

$$\begin{array}{cccccc} & & 0 & 1 & 0 & 1 \\ + & & 1 & 1 & 0 & 1 \\ \hline = & & 0 & 0 & 1 & 0 \end{array}$$

En ignorant la retenue finale, puisque l'encodage est sur 4 bits, on obtient :

$$+5 + (-5) = +2 \quad !!!$$

Complément à 2

Dans cet encodage, le bit de poids fort est toujours utilisé pour représenter le signe, la représentation des entiers positifs est inchangée, mais celle des entiers négatifs utilise un encodage par complément à 2.

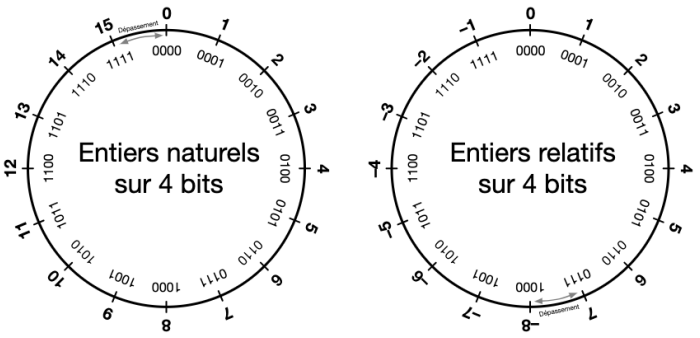
Ce dernier s'obtient, à partir de la valeur absolue du négatif à encoder (le bit de poids fort est donc égal à 0), en inversant la valeur de chaque bit (y compris celui de poids fort) puis en ajoutant 1 (sans tenir compte de la retenue finale).

Ainsi, pour encoder l'entier négatif -5 **sur 4 bits** :

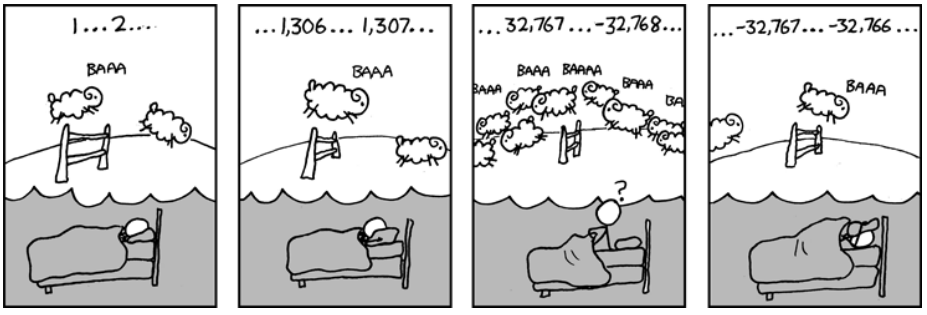
- on part de sa valeur absolue : 0101
- on inverse la valeur de chaque bit : 1010
- on ajoute 1 (sans tenir compte de la retenue finale) : 1011

	1	0	1	0
+				1
=	1	0	1	1

En résumé



Voici un robot comptant des moutons pour s'endormir.
Combien de bits utilise-t-il?



Donner la représentation en complément à 2 sur 8 bits des entiers relatifs :

1. -10
2. -128
3. -42

III. Les caractères

La représentation des caractères dans un ordinateur est l'élément clé pour stocker ou échanger des textes.

En théorie, c'est très simple, il suffit d'associer un numéro unique à chaque caractère, mais en pratique, l'encodage doit être :

- le même pour tous les ordinateurs qui communiquent entre eux
- capable de représenter tous les caractères [5]
- compact pour économiser la mémoire ou le volume des échanges réseaux

1. Codage ASCII

Dans les années 50, les différents encodages de caractères, présents dans les ordinateurs ou les imprimantes, étaient incompatibles entre eux et imposaient l'utilisation de nombreux programmes de conversion.

Au début des années 60, l'ANSI [6] propose une norme de codage des caractères appelée ASCII [7]. Elle définit un jeu de 128 caractères, chacun étant représenté par un octet.

Même si 7 bits sont suffisants pour représenter 128 caractères [a], en pratique chaque caractère occupe 1 octet (8 bits) en mémoire.
En effet, le bit de poids fort est utilisé pour une « somme de contrôle » afin de détecter d'éventuelles erreurs de transmission, mais ce « bit de parité » sera ignoré dans ce cours.

a. De 000 0000 à 111 1111

Cette correspondance est résumée dans la table ASCII :

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

Pour trouver l'octet [8] correspondant à un caractère, il suffit de concaténer le chiffre hexadécimal de sa ligne avec celui de sa colonne.

Par exemple :

- le caractère A correspond au nombre 41_{16} c'est à dire à l'octet 0100 0001
- le caractère + correspond au nombre $2B_{16}$ c'est à dire à l'octet 0010 1011

La table ASCII contient plusieurs catégories de caractères :

- les lettres de l'alphabet latin en majuscule et en minuscule
- les chiffres de 0 à 9
- des signes de ponctuations, des parenthèses ou des crochets
- des opérateurs arithmétiques
- des caractères spéciaux [9] (entre 00 et 20)

Un texte codé en ASCII est alors simplement la suite d'octets correspondant à cette chaînes de caractères.

[5]. Y compris les caractères « non imprimables » qui correspondent à des actions (passer à la ligne, ...), mais aussi à des commandes de protocoles de communication (début de texte, ...)
[6]. American National Standards Institute
[7]. American Standard Code for Information Interchange
[8]. Il est représenté par un nombre hexadécimal à deux chiffres
[9]. Des caractères « blancs » (espace, tabulation, ...), la suppression, l'effacement, ..., mais aussi des caractères « de contrôle » utiles pour les protocoles de communication, le contrôle de périphériques, ...

Donner le codage ASCII, en hexadécimal, de "Ceci est un texte!".

Donner le codage ASCII, en binaire, de "Gymnase".

2. Codage UTF-8

Les caractères de la table ASCII se sont vite avérés insuffisants pour transmettre des textes dans des langues autres que l'anglais. Il manque en effet de nombreux caractères, comme les lettres accentuées, rien qu'en considérant les langues reposant sur un alphabet latin.

l'ISO [10] a alors proposé la norme ISO-8859, une extension de l'ASCII qui utilise les huit bits (et non 7) de chaque octet pour représenter les caractères. Au total, ce sont donc 256 caractères (2 fois plus) qui peuvent être encodés, mais cela reste bien insuffisant.

Pour représenter le plus de caractères possible, la norme ISO-8859 définit 16 tables [11] indépendantes mais compatibles entre elles. Malheureusement, elles ne permettent d'écrire un texte avec des caractères provenant de tables différentes.

L'ISO a défini un jeu universel de caractères (UCS [12]), sous la norme ISO-10646, qui associe à chaque caractère un numéro, appelé point de code et noté U+ xxxx où chaque x désigne un chiffre hexadécimal. Actuellement, cette norme compte plus de 110 000 caractères, mais elle peut en contenir 2^{32}. Par soucis de compatibilité, les 256 premiers points de code sont ceux de la norme ISO-8859-1 (latin-1).

Avec un encodage naïf, cette norme utiliserait quatre octets pour représenter chaque caractère, alors que la grande majorité des échanges repose sur les 256 premiers (un octet) voire les 65536 premiers (deux octets) en élargissant au monde entier.

La norme Unicode définit alors trois techniques d'encodage plus ou moins économiques, appelées UTF [13] et notées UTF-8, UTF-16 et UTF-32 où l'entier désigne le nombre minimal de bits utilisés. Nous nous limiterons ici au format UTF-8.

UTF-8

C'est le format le plus utilisé sous Linux, dans les protocoles réseaux et les sites Web. Il est compatible avec le standard ASCII (les 127 premiers caractères coïncident) et repose sur un principe d'encodage qui est résumé dans le tableau suivant :

Plage	Suite d'octets (en binaire)	bits codant
U+0000 à U+007F	0xxxxxxx	7 bits
U+0080 à U+07FF	110xxxxx 10xxxxxx	11 bits
U+0800 à U+FFFF	1110xxxx 10xxxxxx 10xxxxxx	16 bits
U+10000 à U+10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	21 bits

[10]. Organisation Internationale de Normalisation
[11]. ISO-8859-1, ISO-8859-2, ..., ISO-8859-16
[12]. Universal Character Set
[13]. Universal Transformation Format

Voici trois représentations de points de code selon le format UTF-8 :

Point de code	Point de code (binaire)	UTF-8 (binaire)
U+004B	$\overbrace{0100}^{4}\overbrace{1011}^{B}$	01001011
U+00C5	$\overbrace{1100}^{C}\overbrace{0101}^{5}$	11000011 10000101
U+0A9C	$\overbrace{0000}^{0}\overbrace{1010}^{A}\overbrace{1001}^{9}\overbrace{1100}^{C}$	11100000 10101010 10011100

Donner le codage UTF-8 :

1. du caractère "é" (point de code U+00E9).
2. de la chaîne de caractères "Maturité".

Donner le codage UTF-8 :

1. des caractères :
   - "Ç" (point de code U+00C7)
   - "û" (point de code U+00FB)
   - "€" (point de code U+20AC)
2. de la chaîne de caractères "Ça coûte 10 €!".

IV. Les flottants

L'encodage des nombres flottants est inspiré de l'écriture scientifique :

$$\pm m \cdot 10^n$$

où m est un nombre décimal appartenant à $[1;10[$ appelé *mantisse*
et n un entier relatif appelé *exposant*

Donner l'écriture scientifique des nombres décimaux :

1. 2 156
2. -398 879,62
3. 0,000 142
4. 1,34

Dans la norme internationale IEEE 754, un nombre flottant est de la forme :

$$(-1)^s m \cdot 2^{(e-d)}$$

où $(-1)^s$ est le signe, m la mantisse et e l'exposant décalé de d.

Plus précisément :

- $s = 0$ (respectivement 1) désigne le signe + (respectivement -)
- m appartient à $[1;2[$
- d vaut 127 ou 1023 selon la précision utilisée (cf. diapo suivante)

La mantisse m étant toujours dans $[1;2[$, elle représente un nombre de la forme :

$$1,xx...xx$$

c'est à dire un nombre commençant toujours par le chiffre 1.

Par conséquent, pour économiser un bit de précision, les bits dédiés à la mantisse sont uniquement utilisés pour représenter les chiffres après la virgule que l'on appelle la *fraction* :

$$\text{mantisse} = 1,\text{fraction}$$

Cette norme définit ainsi deux formats selon le nombre de bits utilisés :

1. la *simple précision* sur 32 bits
2. la *double précision* sur 64 bits

Le tableau ci-dessous résume les deux encodages :

	exposant (e)	fraction (f)	valeur
simple précision (32 bits)	8 bits	23 bits	$(-1)^s \cdot 1,f \cdot 2^{(e-127)}$
double précision (64 bits)	11 bits	52 bits	$(-1)^s \cdot 1,f \cdot 2^{(e-1023)}$

Par exemple, le mot de 32 bits suivant :

$$\overbrace{1}^{\text{signe}}\ \overbrace{10000110}^{\text{exposant}}\ \overbrace{10101101100000000000000}^{\text{fraction}}$$

représente le nombre décimal calculé ainsi :

$$\begin{aligned}
\text{signe} &= (-1)^1 \\
&= -1
\end{aligned}$$

$$\begin{aligned}
\text{exposant} &= 2^7 + 2^2 + 2^1 \\
&= 128 + 4 + 2 \\
&= 134
\end{aligned}$$

$$\begin{aligned}
\text{fraction} &= 2^{-1} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-8} + 2^{-9} \\
&= 0,677734375
\end{aligned}$$

Soit, au final, le nombre décimal suivant :

$$-1,677734375 \cdot 2^{(134-127)} = -214,75$$

Donner la valeur décimale des nombres flottants codés en simple précision :

1. 1 01111110 11110000000000000000000
2. 0 10000011 11100000000000000000000

Donner la représentation flottante en simple précision de :

1. 128
2. -32,75

En simple précision, l'exposant e est un entier compris entre 0 et 255 (8 bits). L'exposant décalé $(e-127)$ est alors signé [a] et compris entre -127 et 128.

En réalité, les valeurs extrêmes 0 et 255 de l'exposant e sont réservées pour représenter des valeurs spéciales (cf. diapo suivante), l'exposant décalé $(e-127)$ est donc compris entre -126 et 127.

Ainsi, l'intervalle des nombres décimaux positifs, représentable en simple précision [b], est approximativement $\left[10^{-38};10^{38}\right]$.

a. Les exposants négatifs sont obtenus en retranchant 127 et non par complément à 2
b. Celui en double précision est $\left[10^{-308};10^{308}\right]$

Le tableau ci-dessous résume les encodages des valeurs spéciales :

signe	exposant	fraction	valeur spéciale
0	0	0	$+0$
1	0	0	-0
0	255	0	$+\infty$
1	255	0	$-\infty$
0	0	$\neq 0$	nombre dénormalisé dans $\left[0;2^{-126}\right]$
1	0	$\neq 0$	nombre dénormalisé dans $\left[-2^{-126};0\right]$
0	255	$\neq 0$	NaN

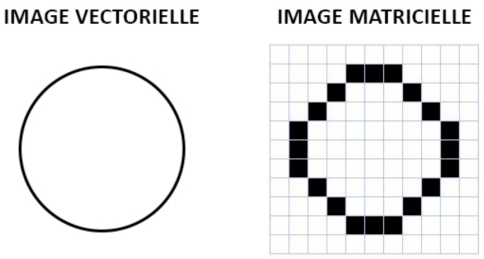
1. Les deux infinis sont utilisés pour indiquer des dépassements de capacité
2. Les nombres dénormalisés qui permettent de rééquilibrer la représentation des nombres autour de 0 sont hors-programme

V. Les images (matricielles)

Les images **vectorielles** sont des images numériques construites à partir de primitives géométriques (segments de droites, arcs de cercles, polygones, ...) auxquelles on peut appliquer différentes transformations (translations, symétries, rotations, ...).

Créées à partir d'équations mathématiques, ces images vectorielles sont stockées dans des fichiers très légers (.ps et .eps) et peuvent être redimensionnées sans perdre en qualité. Malheureusement, elles sont limitées en termes de réalisme et donc inutilisables en photographie par exemple.

A l'inverse, les images **matricielles** sont représentées à l'aide de tableaux [14] de pixels.



Cette section fait l'objet d'un jupyter notebook disponible sur moodle.

Ce cours s'appuie sur :

- le site internet Modulo développé par les concepteurs du programme
- l'ouvrage Numérique et sciences informatiques aux éditions Ellipses

[14]. Ces tableaux à deux dimensions (lignes et colonnes) sont appelés matrices en Mathématiques