1. Introduction

Les données manipulées dans la vie courante sont des entiers décimaux. L'électronique numérique, support matériel de l'informatique, est basée sur une représentation binaire (base 2 : 0 et 1). Le but de ce chapitre est de présenter les différents codages qui existent pour passer des nombres entiers, décimaux, et aussi des caractères, à une représentation sous forme *binaire* (ou *hexadécimale*).

2. Différents codages existant

2.1. Codage des entiers naturels

2.1.1. Définition

La base de travail de « tous les jours » est la base 10 (10 chiffres de 0 à 9) 456, 9658 etc.

Pour passer d'un entier naturel de base 10 en base 2 (binaire) il suffit de décomposer ce nombre en fonction des puissances de 2 :1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ….

Exemple :

227 = 128 + 64 + 32 + 2 + 1 soit 1 1 1 0 0 0 1 1 en binaire naturel

MSB Most Significant Bit (Bit de poids

1*128 + **1***64 + **1***32 + **0***16 + **0***8 + **0***4 + **1***2 + **1***1

LSB : Least Significant Bit (Bit de poids faible)

Soit 0xE3 ou $E3_{16}$ en hexadécimal

L'avantage de l'hexadécimal est de pouvoir « compacter » les nombres qui deviendraient vite énormes. Par exemple écrire en binaire le nombre hexadécimal suivant : EA56B06

Remarques :

1. on écrit souvent les nombres binaires par groupe de 4 (quartet) pour pouvoir plus facilement les transcrire en hexadécimal.
2. 0xE3 : « 0x »signifie que le chiffre qui suit est en hexadécimal (convention de programmation en langage C)

Voir le tableau de conversion décimal-binaire-hexadécimal en annexe 1.

2.1.2. Opérations en binaire :

0 + 0 = *0* 0 + 1 = *1*

1 + 0 = *1* 1 + 1 = *10*

	0	1	1	1
+	0	1	1	1
	1	*1*	*1*	*0*

	1	1	1	1
+	0	1	1	1
	1	*0*	*1*	*1*	*0*

2.1.3. Opération en hexadécimal

```
    1  5          1  B  A
+   2  B      +   E  4  8
-------------    ----------
    4  0
```

2.1.4. Exercices

1- Convertir en binaire : 33, 459, 256, 2596
2- Convertir en hexadécimal : 35, 256, 4096
3- Convertir en décimal : 100111_2 , 111_2 , 111_{16}, $ABCD1_{16}$
4- Effectuer 110011 + 1001 ; ABC + 258D

2.2. Codage des entiers relatifs

Pour coder un nombre relatif, il faut avoir une information de signe. Le bit de signe est choisi comme étant le MSB et est égal à 1 pour préciser que le nombre est négatif. Il existe plusieurs types de représentation des entiers relatifs.

2.2.1. Binaire signé

Le bit de poids fort représente le signe (0 pour + ; 1 pour -) :

- +7 s'écrit **0**111 +67 s'écrit **0**100 0011
- -1 s'écrit **1**111 -61 s'écrit **1**100 0011

Sur 4 bits on peut compter de -7 à +7 ; sur 8 bits de -127 à +127

Inconvénients :

- il existe 2 représentation de 0 : 0000 ou 1000,
- L'opération de soustraction de 2 nombres en binaire signé est délicate

2.2.2. Complément à 2

Un nombre complément à 2 s'écrit sur un, deux, ou plusieurs octets (et non sur 3, 5 ou 13 bits). Le nombre positif s'écrit comme en binaire. Le nombre complément à 2 d'un nombre négatif s'obtient en ajoutant 1 au complément de ce nombre.

Par exemple :

- -4 devient
$\overline{0100}+1$ soit $1011+1=1100$
- -33 devient

0010 0001 + 1 soit 1101 1110 + 1 = 1101 1111

En langage C, la représentation d'un type `int` est sur 2 octets en complément à 2. Un type `unsigned short` est sur 1 octet (pas de signe).

2.2.3. Code BCD (Binary Coded Decimal)

Chaque chiffre du nombre est codé en son équivalent binaire.

Par exemple : 492 devient 0100 1001 0010_{BCD}

- 12 devient 0001 0010_{BCD}

Le BCD signé est codé :

- Signe + : 1011 en poids fort
- Signe - : 1101 en poids fort
- -492 se code 1101 0100 1001 $0010_{BCD\ signé}$

2.2.4. Exercices

1- Ecrire en binaire signé : -24, 15, 0, -11

2- Donner la valeur décimale des nombres suivant écrits en code complément à 2 :0111 0011, 1110 0001, 1010.

2.3. Codage des nombres réels

Pour le codage des nombres réels il existe plusieurs normes de représentation. Le format IEEE 754 décrit ci-dessous (pour des nombres en simple précision) est le format standard:

Soit 1 bit de signe (b31), 8 bits exposants (b30..b23), 23 bits mantisse (b22..b0)

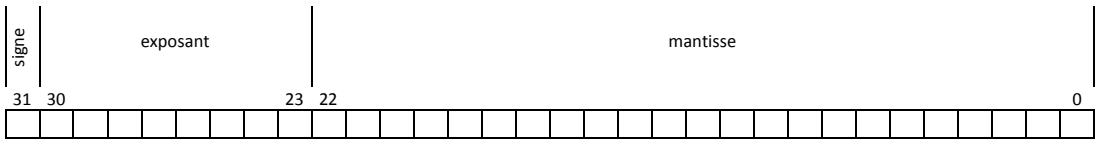
Pour des nombres en double précision on aura 1 bit de signe (b63), 11 bits exposants (b62..b52), 52 bits mantisse (b51..b0).

Progression des puissances négatives de 2 : 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125, 0.00390625, … 0.00000012 ce dernier chiffre étant la valeur de 2^{-23}

Format IEEE 754 : Réels normalisés

Le format IEEE 754 permet de coder sur 32 bits des nombres réels avec 6 chiffres significatifs associé à un facteur allant de 10^{-38} à 10^{+37}.

Les 32 bits sont organisés de la manière suivante :



La traduction décimale N du nombre ainsi codé est :

Exposant — Mantisse

$$N = (-1)^{b_{31}} \times 2^{\left(\left(\sum_{i=0}^{7} b_{(i+23)} \times 2^i\right) - (2^7 - 1)\right)} \times \left(1 + \frac{\sum_{i=0}^{22} b_i \times 2^i}{2^{23}}\right)$$

Où b_i représente le chiffre binaire situé à l'emplacement i.

La traduction décimale m de la mantisse est comprise entre 1 et 2.

$$1 \leq m < 2$$

Un nombre simple précision est codé sur 32 bits, un double précision sur 64 bits.

Exemple :

Le nombre binaire 1011 0100 1111 0000 0000 0000 1000 0010 (simple précision) représente le réel :

$$N = (-1)^1 \times 2^{(105-(2^7-1))} \times \left(1+\frac{7340162}{2^{23}}\right) = -4.47038510^{-7}$$

2.3.1. Exercices

1. Donner la valeur minimale et maximale de la mantisse
2. Donner la valeur décimale du nombre format IEEE754 : C1220000
3. Coder en format IEEE 754 sur 32 bits les nombres suivants 12.125, -2.5625, 3.352
4. Calculs du nombre de chiffres significatifs (codage de 0.4999999)

Solution 12.125 :

12.125 ->1100,001 en binaire ($2^3 + 2^2 + 2^{-3}$)

-> 1,100001 x 2^3 (décalage de la virgule de 3 rangs en binaire)

- Signe : b31 = 0 (positif)
- Exposant : 3 = E - (2^7 - 1) soit E = 1000 0010
- Mantisse : $\left(1+\frac{M}{2^{23}}\right) = 2^0 + 2^{-1} + 2^{-6}$ soit M = $2^{22} + 2^{17}$

12.125 => 0100 0001 0100 0010 0000 0000 0000 0000

Remarque :

1. 0 est codé :
   - Signe x => (existence de zéro « positif » et zéro « négatif)
   - Mantisse 0…0
   - Exposant 0…0
2. +/- ∞ est codé
   - Signe 0 ou 1 (pour + ou -)
   - Mantisse 0…0
   - Exposant 1…1

Il est inutile de connaître par cœur le format IEEE754 mais connaître son existence est indispensable. Il faut par contre savoir qu'il code un nombre flottant **sur 32bits en simple précision avec 6 chiffres significatifs**.

2.4. Codage des caractères

Pour le codage des caractères, le code le plus utilisé est le code ASCII –American Standard Code for Information Interchange- (voir annexe 2). Ce code associe un caractère et un nombre hexadécimal sur 7 bits. Ce codage est surtout employé pour l'envoi de caractère via une liaison série (voir le cours sur les liaisons séries).

Le code ASCII est sur 7 bits, il peut donc coder 128 caractères « éditables » (0,1,…, a, b,…, A, B,…, &, #, $ etc..) et « non éditables » (retour chariot, saut de ligne, etc.…).

Il existe d'autres codes de représentation des caractères :

- EBCDIC Extented Binary Coded Decimal Interchange Code sur 8 bits pouvant donc coder 256 caractères.
- UNICODE (UNIversal CODE) code sur 16 bits soit 65536 caractères utilisé sur processeur Pentium

3. Résumé

Il faut retenir qu'il existe différents types de codage dans les machines. L'utilisateur n'a pas le choix du codage et il peut rester transparent au programmateur. Il est toutefois intéressant de connaître la place mémoire utilisée lors de la déclaration d'une variable. En effet déclarer en C une variable `int`

VAR prendra 2 octets, alors que `unsigned short` VAR n'en prendra qu'un. De la même façon un calcul en virgule flottante (nombre décimal) peut être très gourmand en place mémoire (64 bits soit 8 octets pour `double`).

Déclarations et taille des variables en C++ :

`char :`	-128	à	127	(1 octet)
`unsigned char`	0	à	255	(1 octet)
`short int :`	-32768	à	32767	(2 octets)
`unsigned short int :`	0	a	65535	(2 octets)
`int :`	-2 147 483 648	à	2 147 483 647	(4 octets)
`unsigned int :`	0	à	4 294 967 295	(4 octets)
`float :`	$1{,}17549.10^{-38}$	à	$3{,}40282.10^{+38}$	(4 octets)
`double :`	$2{,}22507.\ 10^{-308}$	à	$1{,}79769.10^{+308}$	(8 octets)
`long double :`	$1.1.\ 10^{-4932}$	à	$1{,}1.10^{+4932}$	(10 octets)

La déclaration d'une variable est donc à adapter aux besoins pour éviter un gaspillage de place mémoire.

ANNEXE 1

Décimal	Binaire naturel	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

Décimal	Binaire signé (complément à 2)
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

ANNEXE 2

CODE ASCII

ASCII	Hex	Symbol
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI

ASCII	Hex	Symbol
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US

ASCII	Hex	Symbol
32	20	(space)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

ASCII	Hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

ASCII	Hex	Symbol
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII	Hex	Symbol
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII	Hex	Symbol
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

ASCII	Hex	Symbol
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	□