Chapitre 2 : Représentation des données

1 représentation des nombres

1.1 Un peu d'histoire

Histoire

Que ce soient des images, des vidéo, des photos, des fichiers de traitement de texte..., toute donnée stockées sur un ordinateur ou transmise vers un ordinateur, ne l'est que par l'intermédiaire de circuits électroniques : les transistors.
Les transistors ne peuvent se trouver que dans deux états : sous tension (on note cela l'état 1) et hors tension (on note cela l'état 0).
En 1970, le premier microprocesseur,Intel 4004, comptait environ 2000 transistors. Aujourd'hui, le microprocesseur Intel Core i7 en compte 2.3 milliard !
Cette unité de stockage est appelée bit, qui peut prendre les valeurs conventionnelles 0 et 1.
Ainsi, toute action informatique est une suite d'opérations sur des paquets de 0 et de 1, regroupés par huit : les octets.
Les processeurs récents sont des processeurs 32 bits ou 64 bits ; ce qui signifie qu'ils disposent de 32 bits ou 64 bits pour stocker un nombre.

1.2 Représentation d'un entier positif en base 2

1.2.1 rappel sur la base 10

Exemple

$135_{10} =$

1.2.2 La base 2

Exemple

$135_{10} =$

Définition 2.1

Soit n un entier naturel.
Il existe un entier p et $a_1, a_2, a_3 ... a_p$ des nombres entiers égaux à 0 ou 1 tels que :
$n = (a_0 a_1 a_2 ... a_p)_2 = a_0 \times 2^p + a_1 \times 2^{p-1} + ... + a_p \times 2^0$.
$(a_0 a_1 a_2 ... a_p)_2$ est la représentation de n en base 2.

Savoir-Faire 2.1
Savoir passer d'un nombre représenté en base 2 à sa valeur en base 10

...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...								

Savoir-Faire 2.2
Savoir passer d'un nombre représenté en base 10 à sa représentation en base 2

- On divise par 2 le nombre donné. On note le quotient et le reste qui est 0 ou 1.
- Puis on divise par 2 le quotient, on note le reste.
- On recommence l'étape 2 jusqu'au moment où le quotient est égal à 0.
- On note ensuite les restes obtenus, en commençant par le dernier et en remontant jusqu'au premier.

Combien d'entier peut-on représenter avec un codage en binaire de n chiffre(s) ?

1.2.3 Représentation d'un entier positif en base 16

Écrire en binaire est fastidieux et source d'erreur quand il y a de grandes séries de bits.
On utilise alors le système hexadécimale (base 16). Les nombres sont écrits à l'aide de 16 symboles :0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F avec (A, B, C, D, E, F) valant respectivement en décimal (10,11, 12, 13, 14, 15)
On a donc le tableau suivant :

Base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base 16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Pour indiquer qu'un nombre est en base 16 on peut utiliser

- l'indice 16 à la fin du nombre : $(AA)_{16}$
- On place \$ devant la nombre : \$AA
- On place les symboles $0x$ devant le nombre : $0xAA$

Exemple

$219_{10} = 11011011_2 = DB_{16} = \$DB = 0xDB$ .....
.....
.....
.....
.....
.....
.....
.....

Exercice 2.1
Combien d'entiers peut-on représenter avec un hexadécimal de 4 chiffres ?

Savoir-Faire 2.3
Savoir passer d'un nombre hexadécimal (représenté en base 16) à sa représentation décimale (en base 10)

...	16^4	16^3	16^2	16^1	16^0
...					

1. $5D_{16} =$
2. $F3C_{16}$

Savoir-Faire 2.4
Savoir passer d'un nombre représenté en base 10 à sa représentation en base 16

1. 978_{10}
2. 184_{10}
3. 252_{10}

☛ Méthode :

- On divise par 16 le nombre donné. On note le quotient et le reste qui est entre 0 et 15.
- Puis on divise par 16 le quotient, on note le reste.
- On recommence l'étape 2 jusqu'au moment où le quotient est égal à 0.
- On note ensuite les restes obtenus (en remplaçant bien sûr 10 par A, 11 par B...et 15 par E), en commençant par le dernier et en remontant jusqu'au premier.

Savoir-Faire 2.5
Savoir passer d'un nombre hexadécimal (représenté en base 16) au nombre binaire (représentation en base 2

1. $F4_{16}$
2. $45E_{16}$
3. $E8E_{16}$

Méthode :

- On convertit en base 2 et sur 4 bits chaque chiffre donné en hexadécimal.
- On réunit ensuite les nombres obtenus pour obtenir la conversion en base 2.

Savoir-Faire 2.6
Savoir passer d'un nombre représenté en base 2 à sa représentation en base 16

1. 01101111_2
2. 10101111110_2
3. 1101011001_2

Méthode :

- On regroupe les bits par 4 (en ajoutant éventuellment des zéros à gauche)
- On convertit en base 16 chaque groupe de 4 bits en binaire

Exercice 2.2
L'adresse MAC $ac :87 :a3 :a8 :c0 :f2 est codée sur combien de bits ?
Convertissez l'adresse MAC en binaire.

1.3 Représentation des entiers relatifs

1.3.1 Introduction

Pour traiter les nombres négatifs, l'ordinateur ne dispose pas du signe moins.
Il faut donc mettre en place une convention pour représenter en binaire des nombres entiers négatifs.
Il existe plusieurs méthodes, et notamment :

- Représentation du bit signé
- le complément à 2
- Représentation en excédent

1.3.2 La représentation des entiers à l'aide du bit signé

Pour coder un entier relatif, une méthode simple consiste à utiliser le bit de poids fort pour indiquer le signe :

- 0 pour un nombre positif
- 1 pour un nombre négatif

Exemples

12_{10} = ...

-12_{10} = ...

Remarque

Il se pose deux problèmes :

- Il y a deux zéros : Sur 4 bits, 0000 et 1000
- Cette représentation induit de la complexité pour des opérations simples (addition de deux nombres de signes différents par exemple)
- Essayez de faire $2 + (-2)$ pour en avoir le coeur net :-)

1.3.3 Le complément à deux

Comme précédemment, on convient que :

- Bit de signe à 0 pour un entier positif
- Bit de signe à 1 pour un entier négatif

Exemple sur 8 bits :

s								valeur décimale	compl à 2^7
1	1	1	1	1	1	1	1	-1	128-127
1	1	1	1	1	1	1	0	-2	128 -126
1	1	1	1	1	1	0	1	-3	128 -125
1				...					
1	1	1	1	0	1	0	0	-12	128-116
1				...					
1	0	0	0	0	0	0	1	-127	128-1
1	0	0	0	0	0	0	0	-128	128-0
0	1	1	1	1	1	1	1	127	
0	1	1	1	1	1	1	0	126	
0				...					
0	0	0	0	0	0	1	0	2	
0	0	0	0	0	0	0	1	1	
0	0	0	0	0	0	0	0	0	

Le complément à 2 d'un nombre binaire s'obtient de la façon suivante :
Voici un exemple pour -12 :

Poids en binaire	128	64	32	16	8	4	2	1
Valeur absolue en binaire								
Inversion des bits								
On ajoute 1								

Donc -12 est représenté par 11110100 sur 8 bits, en complément à 2.

Exercice 2.3
Trouver le complément à 2 de -59 avec une mémoire de 8 bits.
Faire l'addition binaire $59 + (-59)$

Savoir-Faire 2.7

Savoir passer d'un entier relatif à sa représentation en binaire en complément à 2.

- Représenter -8_{10} sur 8 bits, en complément à 2.
- Représenter -121_{10} sur 8 bits, en complément à 2.
- Représenter -100_{10} sur 8 bits, en complément à 2.
- Représenter -4_{10} sur 3 bits, en complément à 2.

Exercice 2.4
Trouver le complément à 2 de -59 avec une mémoire de 8 bits.
Faire l'addition binaire $59 + (-59)$

Savoir-Faire 2.8

Savoir passer d'un entier binaire en complément à 2 à sa représentation en base 10.
Trouvez la représentation décimale des entiers relatifs dont la représentation binaire sur 8 bits est 0000 0000, 1000 0000, 0111 1111 et 1001 1001.

Savoir-Faire 2.9

Savoir effectuer des opérations avec les entiers relatifs

On considère les opérations suivantes :

- $49 + 25$
- $35 + 65$
- $45 - 12$
- $67 - 13$
- $29 - 84$

Pour chaque calcul :

1. Effectuer le calcul directement avec les représentations décimales (trop facile :-))
2. Représenter chaque opérande en binaire sur 8 bits, en complément à deux.
3. Effectuer l'opération binaire
4. Représenter le résultat de l'opération en base 10 et comparer avec le résultat attendu.

1.4 Représentation d'un réel en base 2

1.4.1 Virgule fixe

Le codage des nombres à virgules fixe nous permettra ensuite d'étudier le codage avec virgule flottante.

Dans le système décimal, écrire 56.375 signifie :
$56.375 = 5 \times 10^1 + 6 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$
La nouveauté est ici la présence des puissances de 10 négatives pour les chiffres après la virgule.
Il en est de même en binaire, avec des puissances de 2 :

Exemple
On désire coder en virgule fixe le nombre réel 56.375.

1. Commencer par coder la partie entière : 56

2^6	2^5	2^4	2^3	2^2	2^1	2^0
$= 64$	$= 32$	$= 16$	$= 8$	$= 4$	$= 2$	$= 1$
0	1	1	1	0	0	0

   56 se décompose de façon unique $56_{10} = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
   Donc $56_{10} = 111000_2$

2. En ce qui concerne la partie fractionnaire : 0.375
   Le fonctionnement est identique, mais avec des exposants de 2 négatifs :

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
$= 0.5$	$= 0.25$	$= 0.125$	$= 0.0625$	$= 0.03125$	$= 0.015625$
0	1	1			

   0.375 se décompose de façon unique $0.375 = 0 \times 0.5 + 1 \times 0.25 + 1 \times 0.125$
   Donc $0.375 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$
   Donc $0.375_{10} = 0.011_2$

☞ **Attention :** Seconde méthode pour la partie fractionnaire :

- $0.375 \times 2 = 0.75 = 0(partie\ entière) + 0.75(partie\ fractionnaire)$
- $0.75 \times 2 = 1.5 = \quad 1(partie\ entière) + 0.5$
- $0.5 \times 2 = 1 = \quad 1(partie\ entière) + 0(partie\ fractionnaire)$

Finalement, on retrouve bien $0.375_{10} = 0.011_2$

3. On peut ensuite conclure :
$56.375_{10} = 111000.011_2$

Savoir-Faire 2.10

coder en virgule fixe les nombres suivants :

- 123.6875_{10}
- 14.5_{10}
- 435_{10}
- 171.78515625_{10}

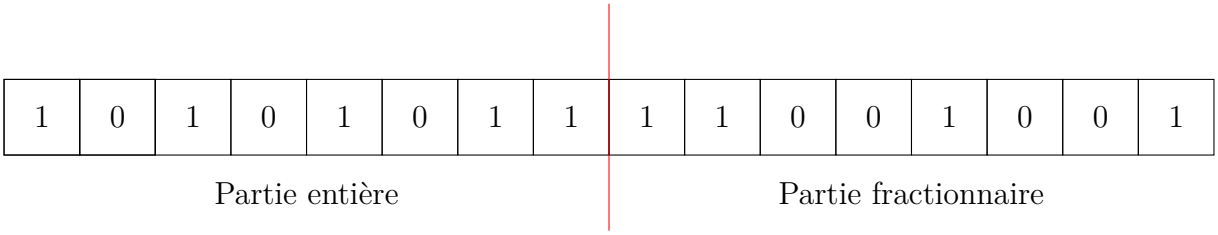
Savoir-Faire 2.11

Décoder maintenant les nombres suivants (virgule fixe)en nombres décimaux :

- 11.101_2
- 100111.101_2
- 1001.111_2

1.4.2 Virgule flottante

Si on reprend le dernier exemple $171,78515625_{10}$ est codé en 10101011.11001001_2, c'est à dire en la succession des bits suivants :



Le trait rouge représente la virgule. La position de cette virgule est à entrer dans le préambule. Ici , on a un codage en virgule fixe "8 × 8"

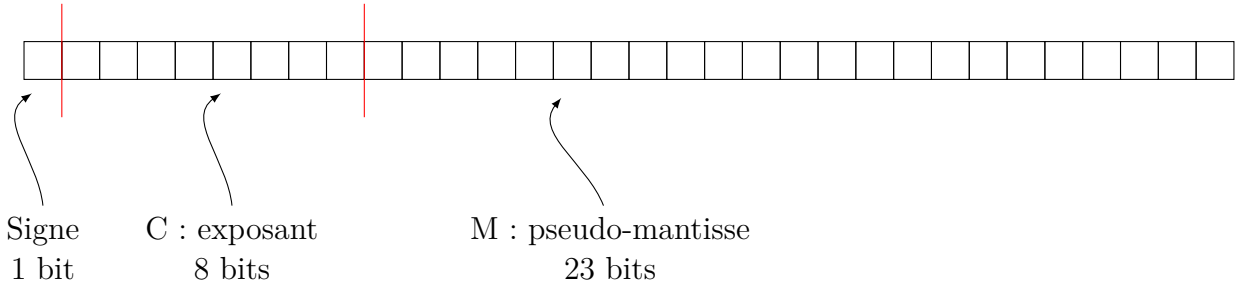
- **Avantages :** L'addition de deux nombres en virgules flottante est très facile et rapide en nombre de calculs : cela se passe comme pour l'addition de deux entiers binaires, en ajoutant juste au "bon endroit" la virgule.
- **Inconvénients :**
  - Le nombre de bits nécessaires pour coder le nombre n'est pas fixe, et dépend du nombre à coder.

- Il faut connaître à l'avance (lors de l'écriture du programme) l'ordre de grandeur des nombres à manipuler afin de positionner au mieux la virgule ; une fois la position de la virgule choisi, on ne peut plus changer. Ceci est un inconvénient majeur : ce type de connaissance a priori existe pour certaines applications, mais dans le cas général ce n'est pas le cas.

☞ Pour pallier à ce manque de flexibilité, le concept de virgule flottante est nécessaire !

Il existe différente façon de coder un nombre en virgule flottante.

L'une d'elle propose ce format (format IEEE754 simple précision) :



Principe :

- Le signe est codé 1 lorsque le nombre est négatif, 0 sinon.

- Il faut ensuite mettre le nombre est sous la forme $\pm 1.M \times 2^C$:
  - On commence par coder le nombre souhaité en utilisant la méthode de la virgule fixe
  - On représente ensuite ce nombre de la forme $1.M \times 2^C$
    - C correspond à l'exposant codé en excédent à 127 (sur 8 bits).
    - M correspond à la pseudo-mantisse (pseudo, car le '1.' n'est pas codé, c'est toujours 1 et on fait donc l'économie d'un bit)(sur 23 bits)

Exemple

On veut ici coder en virgule flottante simple précision le nombre $171,78515625_{10}$.

- Le premier bit est 0 (signe positif)
- On a vu que $171,78515625_{10} = 10101011.11001001_2$.
  Exprimons ce nombre sous la forme $1.M \times 2^C$:
  $10101011.11001001 = 1.1110110100110011 \times 2^7$ (décalage de7 chiffres vers la gauche)
- On a donc $C = 7 + 127$(excédent)$= 134$ donc $C = 10000110_2$.
- On trouve ensuite M, en enlevant $'1.'$ à 1.1110110100110011 : $M = 1110110100110011$.
  On complète ensuite avec des zéros pour arriver à 23 bits : $M = 11101101001100110000000$

- On a donc : $171,78515625_{10} = 01000011001010111100100100000000_2$

Savoir-Faire 2.12

Savoir coder un nombre décimal en utilisant la virgule flottante simple précision.

Coder en virgule flottante simple précision les nombres suivants :

- 123.6875_{10}
- 14.5_{10}
- 435_{10}
- $171,78515625_{10}$
- 0.25
- 0.1 (plus difficile)
- $\frac{1}{3}$ (plus difficile)

Savoir-Faire 2.13

Savoir décoder en nombre décimal un nombre écrit en virgule flottante simple précision.

Décoder en virgule flottante simple précision les nombres suivants : Décode en nombre décimal les nombres binaires suivants, exprimés avec une virgule flottante simple précision :

- $01000000001011100001010001111010_2$
- $01000011000100100001000000000000_2$
- $11000101101100011011001100000000_2$
- $00111000111100001011110010111110_2$