I. Les entiers relatifs

Définition : un entier relatif est un entier naturel muni d'un signe. Ainsi, les entiers positifs sont les entiers naturels (0 ; 1; 2 ; 3 ;...), et les entiers négatifs sont leur opposé (0 ; -1 ; -2 ; -3 ;...).

Pour coder ces entiers relatifs, on pourrait choisir d'utiliser un bit pour le signe et les autres bits pour la valeur absolue. Mais cette méthode a des inconvénients.
La méthode privilégiée est le *complément à deux* qui permet de réaliser des opérations arithmétiques sans problème.
Celle-ci consiste à réaliser un *complément à un* de la valeur absolue du nombre, puis d'ajouter 1 au résultat.

Méthode : le complément à *n*
Pour écrire le complément à n d'un nombre, écrit en base $n + 1$, on remplace chaque chiffre x par le chiffre $n – x$.
Attention : ne pas confondre le complément à 2 (en base 3) et le complément à 2 (en base 2 = complément à 1 + 1)

Exercice 1 : écrire le complément à 9 du nombre 49386109_{10}.
Réponse : 50613890.

Exercice 2 : codage du nombre −5 en base deux et ordre des nombres :
1. Écrire le nombre 5 en binaire, sur 6 bits.
2. Écrire son complément à un.
3. Ajouter une unité à la valeur obtenue. C'est la représentation de – 5 en base 2.
4. De la même manière, coder maintenant le nombre −6 en base deux sur 6 bits. L'ordre des nombres est-il respecté ?

Réponse : −5 : 00101 / 11010 / 11011. −6 : 00110 / 11001 / 11010 ; l'ordre est bien respecté (−6 avant −5).

Exercice 3 :
1. Écrire le complément à deux des nombres 0_{10}, -1_{10}, et -10_{10} codés en binaire sur 4 bits et sur 6 bits.
2. Comparer les résultats précédents aux codages des nombres 0_{10}, 6_{10} et 15_{10} en binaire sur 4 bits et sur 6 bits.
3. Qu'en déduisez-vous ?

Réponse :

Nombre	Codage en binaire	Complément à un sur 4 bits	Complément à deux sur 4 bits	Complément à un sur 6 bits	Complément à deux sur 6 bits
0_{10}	0 : 000000	1111	0000	111111	000000
-1_{10}	1 : 000001	1110	1111	111110	111111
-10_{10}	10 : 001010	0101	0110	110101	110110

Nombre	Codage en binaire sur 4 bits	Codage en binaire sur 6 bits
0_{10}	0000	000000
6_{10}	0110	000110
15_{10}	1111	001111

Avec n bits, ce système permet de représenter les nombres entre -2^{n-1} et $2^{n-1} - 1$.
Tous les nombres négatifs ont le bit de poids fort égal à 1, et tous les nombres positifs ont un bit de poids fort à 0.
Les entiers naturels de 0 à $2^{n-1} - 1$ représentent les entiers relatifs positifs ou nul correspondants et les entiers naturels de 2^{n-1} à $2^n - 1$ représentent les entiers relatifs négatifs de -2^{n-1} à −1.
Exemple : entiers naturels ou relatifs codés sur 4 bits :

Entier naturel (binaire)	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Entier naturel (base 10)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Entier relatif (base 10)	0	1	2	3	4	5	6	7	−8	−7	−6	−5	−4	−3	−2	−1

Bit de poids fort = 0 — Bit de poids fort = 1

Exercice 4 :

1. Déterminer l'ensemble des entiers relatifs que l'on peut représenter sur un octet.
2. Écrire les limites en bases 2 et 10 des entiers naturels et des entiers négatifs que l'on peut représenter sur un octet.

Réponse :

1. Avec un octet, soit huit bits, on représente 2^8 nombres différents, soit les entiers naturels n de 0 jusqu'à 255 et donc les entiers relatifs de $-2^7 = -128$ à $2^7 - 1 = 127$.
2. Entiers naturels : de 0 à 127, soit de 0000 0000 à 0111 1111 en base deux.
   Entiers négatifs : de −128 à −1, soit de 1000 0000 à 1111 1111 en base deux, effectivement représentés par les entiers *n* de 128 à 255, avec l'entier $r = n - 256$.

Le nombre $n = 127$ représente l'entier relatif $r = 127$; le nombre $n = 128$ représente l'entier relatif $r = 128 - 256 = -128$; le nombre n = 255 représente l'entier relatif r = 255 − 256 = −1.

Exercice 5 :

1. Écrire la valeur binaire des nombres 9 et 14 sur 5 bits.
2. Effectuer l'addition des nombres 9 et 14 (codés en binaire).
3. Effectuer la soustraction 14 – 9 (codés en binaire).

Réponse :

1. 14 : 01110 ; 9 : 01001
2. 9 + 14 : 10111
3. 14 – 9 : 00101, soit 5

II. Les nombres à virgule

La norme IEEE 754 (Standard for Binary Floating-Point Arithmetic) définit la représentation de nombres réels (appelés flottants).

	Représentation normalisée	Notation scientifique
Notation générale	**s m × 2^n**	**s a × 10^n**
signe	s (+ ou –)	s (+ ou –)
mantisse	$1 \leq m < 2$	$1 \leq a < 10$
exposant	n, entier relatif	n, entier relatif

Dans l'ordinateur, les nombres seront alors codés par : "signe mantisse (tronquée) exposant (biaisé)", chaque partie étant codée en binaire.

Les flottants IEEE disposent de deux niveaux de précision :

Précision	**Espace mémoire utilisé**	**Répartition de l'espace utilisé**		
		signe	exposant	mantisse
simple (*float*)	32 bits	1 bit	8 bits	23 bits
double (*double*)	64 bits	1 bit	11 bits	52 bits

Remarques :

- Dans tous les cas, le signe + est représenté par 0 et le signe − par 1.
- L'exposant n est un entier relatif.
  Sur 11 bits, il peut prendre 2^{11} valeurs différentes, donc il est donc compris entre −1022 et 1023. On le représente alors par l'entier naturel n + 1023 (c'est à dire $n + 2^{11-1} - 1$), qui est compris entre 1 et 2 046 (cette opération *biaise* l'exposant). Les deux entiers naturels 0 et 2 047 sont réservés pour des situations exceptionnelles : +∞, −∞, NaN (not a number), etc.
- La mantisse m est un nombre binaire à virgule compris entre 1 inclus et 2 exclu, comprenant 52 chiffres après la virgule (pour 52 bits). Comme cette mantisse est comprise entre 1 et 2, elle a toujours un seul chiffre avant la virgule et ce chiffre est toujours un 1 ; il est donc inutile de le représenter et tous les bits sont utilisés pour représenter les chiffres après la virgule.
  Ce nombre sans le chiffre avant la virgule est appelé *pseudo-mantisse*.

Les nombres à virgule flottante sont des approximations de nombres réels. En faisant varier l'exposant n, on fait *flotter* la virgule décimale.

Les calculs en virgule flottante sont pratiques, mais attention :

- leur précision est limitée ; cela se traduit par des arrondis qui peuvent s'accumuler de façon gênante. (Pour cette raison, en comptabilité les calculs ne sont pas effectués en virgule flottante).
- il n'y a aucun nombre dans un voisinage suffisamment petit de zéro : une représentation flottante possède une plus petite valeur négative, une plus petite valeur positive, et entre les deux le zéro. mais aucune autre valeur ! Que vaut dans l'ordinateur la plus petite valeur positive divisée par 2 ? Pour comparer deux nombres flottants, on vérifiera donc que la valeur absolue de leur différence est inférieure ou égale à une précision donnée.

Exemple : conversion de 28,8125 en binaire
Conversion de 28 : $(11100)_2$
Conversion de 0,8125 :

$0{,}8125 \times 2 = 1{,}625 = \underline{1} + 0{,}625$
$0{,}625 \times 2 = 1{,}25 = \underline{1} + 0{,}25$
$0{,}25 \times 2 = 0{,}5 = \underline{0} + 0{,}5$
$0{,}5 \times 2 = 1{,}0 = \underline{1} + 0$

28,8125 peut être représenté par $(11100{,}1101)_2$.
Sa représentation normalisée IEEE 754 est alors $1{,}1100\ 1101 \times 2^4$.
On décompose ainsi le nombre de la manière suivante :

- bit de signe : 0 (nombre > 0)
- exposant sur 8 bits, biaisé à $2^{8-1} - 1 = 127 : 4 + 127 = 131$, soit 1000 0011
- pseudo-mantisse sur 23 bits : 110 0110 1000 0000 0000 0000

Signe	Exposant biaisé	Pseudo-mantisse
0	1000 0011	110 0110 1000 0000 0000 0000

Exercice 6 :

1. Convertir $(15{,}624)_{10}$ en binaire (s'arrêter au cinquième chiffre après la virgule).
2. Indiquer sa représentation normalisée.
3. En déduire sa représentation IEEE 754.

Réponse :

1. Conversion de $(15)_{10}$: $(1111)_2$
   Conversion de $(0{,}624)_{10}$:
   $0{,}624 \times 2 = 1{,}248 = \underline{1} + 0{,}248$
   $0{,}248 \times 2 = 0{,}496 = \underline{0} + 0{,}496$
   $0{,}496 \times 2 = 0{,}992 = \underline{0} + 0{,}992$
   $0{,}992 \times 2 = 1{,}984 = \underline{1} + 0{,}984$
   $0{,}984 \times 2 = 1{,}968 = \underline{1} + 0{,}968$

L'écriture de $(15{,}264)_{10}$ en binaire est donc 1111,1001 1
2. Représentation normalisée : $1{,}1111\ 0011 \times 2^3$.
3. Représentation IEEE 754 :
   - bit de signe : 0 (nombre > 0)
   - exposant sur 8 bits, biaisé à $2^{8-1} - 1 = 127 : 3 + 127 = 130$, soit 1000 0010
   - pseudo-mantisse sur 23 bits : 111 1001 1000 0000 0000 0000

Soit : 0 100 0 001 0 111 1001 1000 0000 0000 0000

III. Chaînes de caractères : Code ASCII

Un ordinateur ne manipule que des nombres binaires. Pour écrire des chaînes de caractère, il faut donc une table de correspondance associant un nombre à chaque caractère. C'est le rôle du code ASCII binaire (American Standard Code for Information Interchange, prononcer *aski*) par exemple.

Ce code utilise un octet par caractère, dont le premier bit est toujours 0, et permet donc de représenter $2^7 = 128$ caractères. Ce sont les caractères que l'on trouve sur les touches d'un clavier "qwerty" d'ordinateur : les lettres minuscules et majuscules, les dix chiffres, des symboles, l'espace, le retour à la ligne, etc.
Par exemple, on attribue les nombres de "97" à "122" aux 26 lettres de l'alphabet a, b, c,... , z ; les nombres de "48" à "57" aux neuf chiffres 0, 1, 2,... , 9 ; l'espace est codé par le nombre "32", le point par le nombre "46", etc. (voir une table complète sur le site www.table-ascii.com).

Attention à ne pas confondre un chiffre et le caractère qui le représente. Les caractères 0, 1, 2, 3, 4 correspondent dans le code ASCII aux nombres 48, 49, 50, 51, 52. (30, 31, 32, 33, 34 en hexadécimal).
Ce code a été conçu pour des textes écrits en anglais. Or, en français, nous avons des accents, la cédille, et dans d'autres langues, des caractères complètement différents : caractères chinois, indiens, russes, grecs, arabes, hébreus, éthiopiens,... Il y a donc eu diverses extensions, par exemple le code latin-1, pour le français qui contient 191 caractères. Actuellement, l'Unicode recense près de 110000 caractères et existe en plusieurs versions.
La version UTF-8 (Universal Transformation Format sur 8 bits ou plus), codage de taille variable, a été conçue pour coder l'ensemble des caractères internationaux d'Unicode.
La principale caractéristique d'UTF-8 est qu'elle est compatible avec la norme ASCII, c'est-à-dire que tout caractère ASCII se code en UTF-8 sous forme d'un unique octet, identique au code ASCII. Par exemple "A" (A majuscule) a pour code ASCII 65 et se code en UTF-8 par l'octet 65. Les caractères non ASCII sont codés sur 2 à 4 octets. Le caractère € (euro) se code par exemple sur 3 octets : 226, 130, et 172.
À noter que pour les SMS, messages courts de 160 octets, on a vu la réapparition des techniques de codage de texte sur des mots de sept bits.

Exercice 7 :
On représente un texte en écrivant les caractères les uns après les autres.
Voici l'exemple d'un texte où on a séparé les octets :
en binaire : 00111000 00100000 01100010 01101001 01110100 01110011 00101110
en hexadécimal : 38 20 62 69 74 73 2E en décimal : 56 32 98 105 116 115 46
Traduire ce texte en code ASCII.
Réponse :
D'après le code ASCII, le nombre 56 correspond au caractère "8", le nombre 32 au caractère "espace", les nombres 98, 105, 116, 115 aux caractères "b", "i", "t", "s" et le nombre 46 au "point" ; donc le texte en français est : " 8 bits. "

Exercice 8 : Traduction d'une série de chiffres binaires.
Donnez la traduction à laquelle correspond le mot de 4 octets codé en hexadécimal suivant : 49 53 4E 2B
selon qu'on le lit comme :

- un entier signé
- un entier représenté en complément à 2
- un nombre représenté en virgule flottante simple précision suivant la norme IEEE 754
- une suite de caractères ASCII (représentés chacun sur 8 bits, le bit de plus fort poids étant inutilisé et codé à 0)

On pourra s'aider de la table de correspondance ASCII disponible sur Wikipedia.

Hexadécimal	4	9	5	3	4	E	2	B
Binaire	0 100	1001	0 101	0011	0100	1110	0010	1011
Entier signé	+ 1 230 196 310							
Complément à 2	+ 1 230 196 310							
IEEE 754	0 100	1001	0 101	0011	0100	1110	0010	1011
	+ exp. biaisé : 146 donc exp : 19		mantisse : 1,1010 0110 1001 1100 0101 011					
	+ 1,1010 0110 1001 1100 010,1 011×2^{19} = + 1101 0011 0100 1110 0010,1011×2^0 = 865 702,6875							
ASCII	I		S		N		+	