Codage unaire

Le moyen le plus simple de représenter un entier positif n consiste à le coder par une suite de n bits à 1. Ceci est le codage unaire de n.

– Exemple –

L'entier $(7)_{\text{dix}}$ est codé par la suite $(\texttt{1111111})_{\text{un}}$.

Avec ce codage, les **opérations arithmétiques** sont **très simples** :

- addition : concaténation des codages.

– Exemple –

$(7)_{\text{dix}} + (9)_{\text{dix}} = (\texttt{1111111})_{\text{un}} + (\texttt{111111111})_{\text{un}} = (\texttt{1111111 111111111})_{\text{un}}$

- produit de u et v : substitution de v à chaque 1 de u.

– Exemple –

$(3)_{\text{dix}} \times (5)_{\text{dix}} = (\texttt{111})_{\text{un}} \times (\texttt{11111})_{\text{un}} = (\texttt{11111 11111 11111})_{\text{un}}$

Avantages : opérations arithmétiques faciles sur les entiers positifs.

Inconvénients : mémoire en $\Theta(n)$ pour coder l'entier n ; codage des entiers positifs seulement.

Représentation Binary Coded Decimal (BCD)

Le codage BCD consiste à représenter un entier positif

$$(c_{n-1} \ \dots \ c_1 \ c_0)_{\text{dix}}$$

exprimé en base dix par la suite de $4n$ bits

$$(c'_{n-1} \ \dots \ c'_1 \ c'_0)_{\text{bcd}}$$

où chaque c'_i est le code BCD de c_i.

Celui-ci code chaque chiffre décimal par une suite de quatre bits au moyen de la table

$(0)_{\text{dix}} \to (0000)_{\text{bcd}}$	$(2)_{\text{dix}} \to (0010)_{\text{bcd}}$	$(4)_{\text{dix}} \to (0100)_{\text{bcd}}$	$(6)_{\text{dix}} \to (0110)_{\text{bcd}}$	$(8)_{\text{dix}} \to (1000)_{\text{bcd}}$
$(1)_{\text{dix}} \to (0001)_{\text{bcd}}$	$(3)_{\text{dix}} \to (0011)_{\text{bcd}}$	$(5)_{\text{dix}} \to (0101)_{\text{bcd}}$	$(7)_{\text{dix}} \to (0111)_{\text{bcd}}$	$(9)_{\text{dix}} \to (1001)_{\text{bcd}}$

– Exemple –

$(201336)_{\text{dix}} = (0010\ 0000\ 0001\ 0011\ 0011\ 0110)_{\text{bcd}}$

Représentation Binary Coded Decimal (BCD)

Avantages : représentation très simple des entiers et naturelle car très proche de la base dix.

Inconvénients : gaspillage de place mémoire (six suites de 4 bits ne sont jamais utilisées), les opérations arithmétiques ne sont ni faciles ni efficaces.

Ce codage est très peu utilisé dans les ordinateurs. Il est utilisé dans certains appareils électroniques qui affichent et manipulent des valeurs numériques (calculettes, réveils, *etc.*).

Changement de base — algorithme

Soient $b \geqslant 2$ un entier et x un entier positif. Pour écrire x en base b, on procède comme suit :

1. Si $x = 0$: renvoyer la liste $[0]$
2. Sinon :
   - 2.1 $L \leftarrow [\,]$
   - 2.2 Tant que $x \neq 0$:
     - 2.2.1 $L \leftarrow (x \,\%\, b) \cdot L$
     - 2.2.2 $x \leftarrow x/b$
   - 2.3 Renvoyer L.

Ici, $[\,]$ est la liste vide, $\cdot$ désigne la concaténation des listes, $\%$ est l'opérateur modulo et $/$ est la division entière.

Changement de base

– Exemple –

Avec $x := (294)_{\text{dix}}$ et $b := 3$, on a

x	$x \% 3$	$x/3$	L
$(294)_{\text{dix}}$	$(0)_{\text{dix}}$	$(98)_{\text{dix}}$	$[0]$
$(98)_{\text{dix}}$	$(2)_{\text{dix}}$	$(32)_{\text{dix}}$	$[2, 0]$
$(32)_{\text{dix}}$	$(2)_{\text{dix}}$	$(10)_{\text{dix}}$	$[2, 2, 0]$
$(10)_{\text{dix}}$	$(1)_{\text{dix}}$	$(3)_{\text{dix}}$	$[1, 2, 2, 0]$
$(3)_{\text{dix}}$	$(0)_{\text{dix}}$	$(1)_{\text{dix}}$	$[0, 1, 2, 2, 0]$
$(1)_{\text{dix}}$	$(1)_{\text{dix}}$	$(0)_{\text{dix}}$	$[1, 0, 1, 2, 2, 0]$
$(0)_{\text{dix}}$	–	–	$[1, 0, 1, 2, 2, 0]$

Ainsi, $(294)_{\text{dix}} = (101220)_{\text{trois}}$.

Valeur portée par une suite de chiffres

Soit $b \geqslant 2$ un entier, appelé base.

Soit

$$x := (x_{n-1}x_{n-2}\dots x_1x_0)_b$$

un nombre exprimé en base b. On a ainsi $x_{n-1} \neq 0$ et $0 \leqslant x_i \leqslant b-1$ pour tout $0 \leqslant i \leqslant n-1$.

Les x_i sont les chiffres de x. Il comporte donc n chiffres significatifs.

La valeur dénotée par x est l'entier naturel

$$\sum_{i=0}^{n-1} x_i \times b^i.$$

– Exemple –

La valeur dénotée par $(30042)_{\text{six}}$ est $2 \times 6^0 + 4 \times 6^1 + 3 \times 6^4 = (3914)_{\text{dix}}$.

Note : par convention, dans un calcul, les nombres sont exprimés en base dix.

Représentation binaire des entiers positifs

Le codage binaire consiste à représenter un entier positif en base deux.

En machine, si l'écriture d'un entier est plus longue que huit bits, il est d'usage de le représenter par tranches de huit bits.

Par ailleurs, si son écriture n'est pas de longueur multiple de huit, on complète généralement à gauche par des zéros de sorte qu'elle le soit.

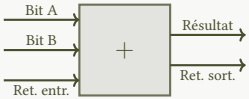
– Exemples –

- $(35)_{\text{dix}} = (100011)_{\text{deux}}$ est représenté par l'octet

- $(21329)_{\text{dix}} = (101001101010001)_{\text{deux}}$ est représenté par la suite de deux octets

L'addition d'entiers — l'additionneur simple

L'additionneur simple est un opérateur qui prend en entrée **deux bits** et une **retenue d'entrée** et qui renvoie deux informations : le **résultat** de l'addition et la **retenue de sortie**.



Il fonctionne selon la table

	Bit A	Bit B	Ret. entr.	Résultat	Ret. sort.
(0)	0	0	0	0	0
(1)	0	0	1	1	0
(1)	0	1	0	1	0
(2)	0	1	1	0	1
(1)	1	0	0	1	0
(2)	1	0	1	0	1
(2)	1	1	0	0	1
(3)	1	1	1	1	1

L'addition d'entiers — l'algorithme usuel

Un additionneur n bits fonctionne de la même manière que l'**algorithme d'addition usuel**.

Il effectue l'addition chiffre par chiffre, en partant de la droite et en reportant les retenues de proche en proche.

– Exemple –

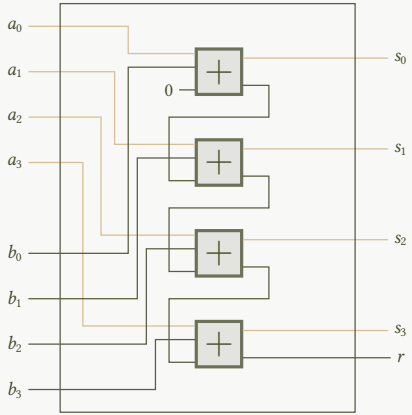
L'addition binaire de $(159)_{\text{dix}}$ et de $(78)_{\text{dix}}$ donne $(237)_{\text{dix}}$:

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

La retenue de sortie est 0.

L'addition d'entiers — l'additionneur 4 bits

On construit un additionneur 4 bits en combinant quatre additionneurs simples :



La multiplication d'entiers — le multiplicateur simple

Le **multiplicateur simple** est un opérateur qui prend **deux bits** en entrée et qui renvoie un **résultat**.

Il fonctionne selon la table

Bit A	Bit B	Résultat
0	0	0
0	1	0
1	0	0
1	1	1

Note : ce multiplicateur simple binaire n'a pas de retenue de sortie, contrairement au multiplicateur simple décimal.

La multiplication d'entiers — l'algorithme usuel

Le multiplicateur simple permet de réaliser la multiplication bit à bit et ainsi, la multiplication de deux entiers codés en binaire, selon l'**algorithme de multiplication usuel**.

– Exemple –

La multiplication binaire de $(19)_{\text{dix}}$ et de $(44)_{\text{dix}}$ donne $(836)_{\text{dix}}$:

						1	0	0	1	1
$\times$					1	0	1	1	0	0
						0	0	0	0	0
					0	0	0	0	0	·
				1	0	0	1	1	·	·
			1	0	0	1	1	·	·	·
		0	0	0	0	0	·	·	·	·
$+$	1	0	0	1	1	·	·	·	·	·
	1	1	0	1	0	0	0	1	0	0

Bit de signe, codage de taille fixée et plage

Pour l'instant, nous avons vu uniquement la représentation d'entiers positifs.

On peut représenter des entiers négatifs et positifs à l'aide d'un bit de signe. C'est le **bit de poids fort** qui renseigne le signe de l'entier

- s'il est égal à 1, l'entier codé est négatif 

;
- s'il est égal à 0, l'entier codé est positif 

.

Dorénavant, on précisera toujours le **nombre n de bits** sur lequel on code les entiers.

Si un entier requiert moins de n bits pour être codé, on **complétera** son codage avec des 0 ou des 1 à gauche en fonction de son signe et de la représentation employée.

À l'inverse, si un entier x demande strictement plus de n bits pour être codé, on dira que x ne peut pas être codé sur n bits.

La plage d'un codage désigne l'ensemble des valeurs qu'il est possible de représenter sur n bits.

Représentation en magnitude signée

Le codage en magnitude signée permet de représenter un entier en le codant en binaire et en lui adjoignant un **bit de signe**.

– Exemples –

En se plaçant sur 8 bits, on a $(45)_{\text{dix}} = (00101101)_{\text{ms}}$ et $(-45)_{\text{dix}} = (10101101)_{\text{ms}}$.

Avantage : calcul facile de l'opposé d'un entier.

Inconvénient : l'addition de deux entiers ne peut pas se faire par l'algorithme classique d'addition.

Il y a de plus deux encodages différents pour zéro (qui est positif et négatif) :

$$(0)_{\text{dix}} = (00\ldots0)_{\text{ms}} = (10\ldots0)_{\text{ms}}.$$

Plage (sur n bits) :

plus petit entier : $(11\ldots1)_{\text{ms}} = -(2^{n-1}-1)$;

plus grand entier : $(01\ldots1)_{\text{ms}} = 2^{n-1}-1$.

Représentation en complément à un

Le complément à un d'une suite de bits

$$u_{n-1} \dots u_1 u_0$$

est la suite

$$\overline{u_{n-1}} \dots \overline{u_1}\ \overline{u_0},$$

où $\overline{0} := 1$ et $\overline{1} := 0$.

Le codage en complément à un consiste à coder un entier négatif par le complément à un de la représentation binaire de sa valeur absolue.

Le codage d'un entier positif est son codage binaire habituel.

Le bit de poids fort doit être un bit de signe : lorsqu'il est égal à 1, l'entier représenté est négatif ; il est positif dans le cas contraire.

Remarque : sur n bits, le complément à un revient à représenter un entier négatif x par la représentation binaire de $(2^n - 1) - |x|$.

Représentation en complément à un

– Exemples –

Sur 8 bits, on a $(98)_{\text{dix}} = (\texttt{01100010})_{\text{c1}}$ et $(-98)_{\text{dix}} = (\texttt{10011101})_{\text{c1}}$.

Avantage : calcul facile de l'opposé d'un entier.

Inconvénient : l'addition de deux entiers ne peut pas se faire par l'algorithme classique d'addition.

Il y a de plus deux encodages différents pour zéro (qui est positif et négatif) :

$$(0)_{\text{dix}} = (\texttt{00}\ldots\texttt{0})_{\text{c1}} = (\texttt{11}\ldots\texttt{1})_{\text{c1}}.$$

Plage (sur n bits) :

plus petit entier : $(\texttt{10}\ldots\texttt{0})_{\text{c1}} = -(2^{n-1}-1)$;

plus grand entier : $(\texttt{01}\ldots\texttt{1})_{\text{c1}} = 2^{n-1}-1$.

Représentation avec biais

On fixe un entier $B \geqslant 0$ appelé biais.

Soit x un entier (positif ou non) que l'on souhaite représenter.

Posons $x' := x + B$.

Si $x' \geqslant 0$, alors le codage de x avec un biais de B est la représentation binaire de x' sur n bits. Sinon, x n'est pas représentable (sur n bits et avec le biais B donné).

– Exemples –

En se plaçant sur 8 bits, avec un biais de $B := 95$, on a

- $(30)_{\text{dix}} = (\texttt{01111101})_{\text{biais=95}}$. En effet, $30 + 95 = 125$ et $(\texttt{01111101})_{\text{deux}} = (125)_{\text{dix}}$.
- $(-30)_{\text{dix}} = (\texttt{01000001})_{\text{biais=95}}$. En effet, $-30 + 95 = 65$ et $(\texttt{01000001})_{\text{deux}} = (65)_{\text{dix}}$.
- l'entier $(-98)_{\text{dix}}$ n'est pas représentable car $-98 + 95 = -3$ est négatif.

Représentation avec biais

Note : ce codage n'est pas compatible avec le bit de signe. En effet, le bit de poids fort d'un entier codé en représentation avec biais ne renseigne pas sur le signe de l'entier.

Avantages : permet de représenter des intervalles quelconques (mais pas trop larges) de $\mathbb{Z}$.

Inconvénient : l'addition de deux entiers ne peut pas se faire par l'algorithme classique d'addition.

Plage (sur n bits) :

plus petit entier : $(0 \dots 0)_{\text{biais=B}} = -B$;

plus grand entier : $(1 \dots 1)_{\text{biais=B}} = (2^n - 1) - B$.

Représentation en complément à deux

On se place sur n bits.

Le complément à deux (puissance n) d'une suite de bits u se calcule en

1. complémentant u à un pour obtenir u' ;
2. incrémentant u' (addition de u' et $0 \ldots 01$) pour obtenir u''.

– Exemple –

Sur 8 bits, avec $u := 01101000$, on obtient

1. $u' = 10010111$;
2. $u'' = 10011000$.

Ainsi, le complément à deux de la suite de bits 01101000 est la suite de bits 10011000.

Remarque : l'opération qui consiste à complémenter à deux une suite de bits est involutive (le complément à deux du complément à deux d'une suite de bits u est u).

Représentation en complément à deux — codage/décodage

Le codage en complément à deux consiste à **coder** un entier x de la manière suivante.

- Si $x \geqslant 0$, le codage de x est simplement son codage binaire habituel.
- Sinon, $x < 0$. Le codage de x est le complément à deux du codage binaire de la valeur absolue de x.

Pour **décoder** une suite de bits u représentant un entier codé en complément à deux, on procède de la manière suivante.

- Si le bit de poids fort de u est `0`, on obtient la valeur codée par u en interprétant directement sa valeur en binaire.
- Sinon, le bit de poids fort de u est `1`. On commence par considérer le complément à deux u'' de u. La valeur codée par u est obtenue en interprétant la valeur de u'' en binaire et en considérant son opposé.

Représentation en complément à deux — remarques

Le codage en complément à deux fait que le **bit de poids fort** est un **bit de signe** : lorsque le bit de poids fort est 1, l'entier représenté est négatif. Il est positif dans le cas contraire.

Un entier x est codable sur n bits en complément à deux si et seulement si la suite de bits obtenue par le codage de x en complément à deux sur n bits possède un **bit de signe cohérent** avec le signe de x.

Remarque : sur n bits, le complément à deux revient à représenter un entier strictement négatif x par $2^n - |x|$.

Représentation en complément à deux

– Exemples –

- **Représentation de $(98)_{\text{dix}}$ sur 8 bits.** On a $(98)_{\text{dix}} = (01100010)_{\text{deux}}$. Cohérence avec le bit de signe. Ainsi, $(98)_{\text{dix}} = (01100010)_{\text{c2}}$.
- **Représentation de $(-98)_{\text{dix}}$ sur 8 bits.** Le complément à deux de 01100010 est 10011110. Cohérence avec le bit de signe. Ainsi, $(-98)_{\text{dix}} = (10011110)_{\text{c2}}$.
- **Représentation de $(130)_{\text{dix}}$ sur 8 bits.** On a $(130)_{\text{dix}} = (10000010)_{\text{deux}}$. Incohérence avec le bit de signe : il est à 1 alors que l'entier est positif. Cet entier n'est pas représentable sur 8 bits.
- **Représentation de $(-150)_{\text{dix}}$ sur 8 bits.** On a $(150)_{\text{dix}} = (10010110)_{\text{deux}}$. Le complément à deux de cette suite est 01101010. Incohérence avec le bit de signe : il est à 0 alors que l'entier est négatif. Cet entier n'est pas représentable sur 8 bits.
- **Représentation de $(-150)_{\text{dix}}$ sur 10 bits.** On a $(150)_{\text{dix}} = (0010010110)_{\text{deux}}$. Le complément à deux de cette suite est 1101101010. Cohérence avec le bit de signe. Ainsi, $(-150)_{\text{dix}} = (1101101010)_{\text{c2}}$.

Représentation en complément à deux

– Exemples –

- **Décodage de $(00110101)_{c2}$ sur 8 bits.** Le bit de poids fort est 0. Ainsi, l'entier codé est positif et $(00110101)_{c2} = (00110101)_{deux}$. On a donc $(00110101)_{c2} = (53)_{dix}$.
- **Décodage de $(10110101)_{c2}$ sur 8 bits.** Le bit de poids fort est 1. Ainsi, l'entier codé est négatif. On doit considérer le complément à deux de la suite 10110101 qui est 01001011. On a maintenant $(01001011)_{deux} = (75)_{dix}$. Ainsi, $(10110101)_{c2} = (-75)_{dix}$.
- **Décodage de $(11111111)_{c2}$ sur 8 bits.** Le bit de poids fort est 1. Ainsi, l'entier codé est négatif. On doit considérer le complément à deux de la suite 11111111 qui est 00000001. On a maintenant $(00000001)_{deux} = (1)_{dix}$. Ainsi, $(11111111)_{c2} = (-1)_{dix}$.

Représentation en complément à deux

Avantage : l'addition de deux entiers se calcule par l'algorithme classique d'addition.

Inconvénient : représentation des entiers négatifs légèrement plus complexe que par les autres codages.

Plage (sur n bits) :

plus petit entier : $(10\ldots0)_{c2} = -2^{n-1}$;

plus grand entier : $(01\ldots1)_{c2} = 2^{n-1} - 1$.

Dans les ordinateurs d'aujourd'hui, les entiers sont habituellement encodés selon cette représentation.

Représentation en complément à deux — calcul rapide

Il existe une **méthode plus rapide** pour calculer le complément à deux d'une suite de bits u que de la complémenter à un et de l'incrémenter.

Elle consiste en les deux étapes suivantes :

1. repérer le bit à 1 de poids le plus faible de u ;
2. complémenter à un les bits de u qui sont de poids strictement plus forts que ce bit.

– Exemple –

L'application de cette méthode sur la suite 01101000 donne :

u	0	1	1	0	1	0	0	0
bit à 1 à droite	0	1	1	0	**1**	0	0	0
cpl. à 1 bits à gauche	1	0	0	1	**1**	0	0	0

La dernière ligne est le complément à deux de u.

Dépassement de capacité

En représentation en complément à deux sur n bits, l'addition de deux entiers x et y peut produire un entier qui sort de la plage représentable. On appelle ceci un dépassement de capacité (*overflow*).

Fait 1. Si x est négatif et y est positif, $x + y$ appartient toujours à la plage représentable.

Fait 2. Il ne peut y avoir dépassement de capacité que si x et y sont de **même signe**.

Règle : il y a dépassement de capacité si et seulement si le **bit de poids fort** de $x + y$ est **différent** du bit de poids fort de x (et donc de y).

– Exemple –

Sur 4 bits, l'addition

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

engendre un dépassement de capacité.

Retenue de sortie

En représentation en complément à deux sur n bits, on dit que l'addition de deux entiers x et y produit une **retenue de sortie** si l'addition des bits de poids forts de x et de y renvoie une retenue de sortie à 1.

– Exemple –

Sur 4 bits, l'addition

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

engendre une retenue de sortie.

Attention : ce n'est pas parce qu'une addition provoque une retenue de sortie qu'il y a dépassement de capacité.

Dépassement de capacité et retenue de sortie

Le dépassement de capacité **(D)** et la retenue de sortie **(R)** sont deux choses disjointes. En effet, il peut exister les quatre cas de figure.

– Exemple –

Sur 4 bits :

- non **(D)** et non **(R)** :

		1	0	0	0
+		0	0	1	0
	(0)	1	0	1	0

- non **(D)** et **(R)** :

		0	1	1	0
+		1	1	0	0
	(1)	0	0	1	0

- **(D)** et non **(R)** :

		0	1	1	0
+		0	1	1	1
	(0)	1	1	0	1

- **(D)** et **(R)** :

		1	0	1	0
+		1	0	0	0
	(1)	0	0	1	0

Résumé des représentations des entiers

- **Représentation en magnitude signée**.
  Plage : de $-2^{n-1}+1$ à $2^{n-1}-1$.
  Avantage : codage simple ; calcul facile de l'opposé.
  Inconvénients : addition non naturelle ; deux codages de zéro.
- **Représentation en complément à un**.
  Plage : de $-2^{n-1}+1$ à $2^{n-1}-1$.
  Avantage : codage simple ; calcul facile de l'opposé.
  Inconvénients : addition non naturelle ; deux codages de zéro.
- **Représentation avec biais** (de $B \geqslant 0$).
  Plage : de $-B$ à $(2^n-1)-B$.
  Avantage : possibilité d'encoder un intervalle arbitraire.
  Inconvénient : addition non naturelle.
- **Représentation en complément à deux**.
  Plage : de -2^{n-1} à $2^{n-1}-1$.
  Avantage : addition naturelle.
  Inconvénient : codage moins intuitif.

L'hexadécimal

Un entier codé en hexadécimal est un entier écrit en base seize.

Ce codage utilise seize symboles : les chiffres `0`, `1`, …, `9` et les lettres `A`, `B`, `C`, `D`, `E`, `F`.

Pour exprimer une **suite de bits** u en hexadécimal, on remplace, en partant de la droite, chaque groupe de quatre bits de u par un chiffre hexadécimal au moyen de la table suivante :

$(\texttt{0000})_{\text{deux}} \to (\texttt{0})_{\text{hex}}$	$(\texttt{0100})_{\text{deux}} \to (\texttt{4})_{\text{hex}}$	$(\texttt{1000})_{\text{deux}} \to (\texttt{8})_{\text{hex}}$	$(\texttt{1100})_{\text{deux}} \to (\texttt{C})_{\text{hex}}$
$(\texttt{0001})_{\text{deux}} \to (\texttt{1})_{\text{hex}}$	$(\texttt{0101})_{\text{deux}} \to (\texttt{5})_{\text{hex}}$	$(\texttt{1001})_{\text{deux}} \to (\texttt{9})_{\text{hex}}$	$(\texttt{1101})_{\text{deux}} \to (\texttt{D})_{\text{hex}}$
$(\texttt{0010})_{\text{deux}} \to (\texttt{2})_{\text{hex}}$	$(\texttt{0110})_{\text{deux}} \to (\texttt{6})_{\text{hex}}$	$(\texttt{1010})_{\text{deux}} \to (\texttt{A})_{\text{hex}}$	$(\texttt{1110})_{\text{deux}} \to (\texttt{E})_{\text{hex}}$
$(\texttt{0011})_{\text{deux}} \to (\texttt{3})_{\text{hex}}$	$(\texttt{0111})_{\text{deux}} \to (\texttt{7})_{\text{hex}}$	$(\texttt{1011})_{\text{deux}} \to (\texttt{B})_{\text{hex}}$	$(\texttt{1111})_{\text{deux}} \to (\texttt{F})_{\text{hex}}$

– Exemple –

$(\texttt{10 1111 0101 1011 0011 1110})_{\text{deux}} = (\texttt{2F5B3E})_{\text{hex}}$.

La conversion dans l'autre sens se réalise en appliquant la même idée.

Ce codage est **concis** : un octet est codé par seulement deux chiffres hexadécimaux.

Les nombres réels

Nombres entiers *vs* nombres réels :

- il y a une infinité de nombre entiers mais il y a un **nombre fini** d'entiers dans tout intervalle $[\![a, b]\!]$ où $a \leqslant b \in \mathbb{Z}$;

- il y a également une infinité de nombre réels mais il y a cette fois un **nombre infini** de réels dans tout intervalle $[\alpha, \beta]$ où $\alpha < \beta \in \mathbb{R}$.

Conséquence : il n'est possible de représenter qu'un sous-ensemble de nombres réels d'un intervalle donné.

Les nombres représentables sont appelés nombre flottants. Ce sont des approximations des nombres réels.

Représentation à virgule fixe

Le codage à virgule fixe consiste à représenter un nombre à virgule en deux parties :

1. sa **troncature**, sur n bits ;
2. sa **partie décimale**, sur m bits ;

où les entiers n et m sont fixés.

La troncature est codée en utilisant la représentation en complément à deux.

Chaque bit de la partie décimale correspond à l'inverse d'une puissance de deux.

– Exemple –

Avec $n = 4$ et $m = 5$,

$$\begin{aligned}(0110.01100)_{\text{vf}} &= 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\ &= (6.375)_{\text{dix}}\end{aligned}$$

Représentation à virgule fixe

Soient $b \geqslant 2$ un entier et $0 < x < 1$ un nombre rationnel. Pour écrire x en **base** b, on procède comme suit :

1. $L \leftarrow [\,]$
2. Tant que $x \neq 0$:
   - 2.1 $x \leftarrow x \times b$
   - 2.2 $L \leftarrow L \cdot \lfloor x \rfloor$
   - 2.3 $x \leftarrow x - \lfloor x \rfloor$
3. Renvoyer L.

Ici, $[\,]$ est la liste vide, $\times$ est la multiplication, $\cdot$ désigne la concaténation des listes et $\lfloor - \rfloor$ est l'opérateur de partie entière inférieure.

Représentation à virgule fixe

– Exemple –

Avec $x := (0.375)_{\text{dix}}$ et $b := 2$, on a

x	$x \times 2$	$x - \lfloor x \rfloor$	L
$(0.375)_{\text{dix}}$	$(\mathbf{0}.75)_{\text{dix}}$	$(0.75)_{\text{dix}}$	$[\mathbf{0}]$
$(0.75)_{\text{dix}}$	$(\mathbf{1}.5)_{\text{dix}}$	$(0.5)_{\text{dix}}$	$[0, \mathbf{1}]$
$(0.5)_{\text{dix}}$	$(\mathbf{1}.0)_{\text{dix}}$	$(0.0)_{\text{dix}}$	$[0, 1, \mathbf{1}]$
$(0)_{\text{dix}}$	–	–	$[0, 1, 1]$

Ainsi, $(0.375)_{\text{dix}} = (0.011)_{\text{vf}}$.

Limites de la représentation à virgule fixe

Appliquons l'« algorithme » précédent sur les entrées $x := (0.1)_{\text{dix}}$ et $b := 2$. On a

x	$x \times 2$	$x - \lfloor x \rfloor$	L
$(0.1)_{\text{dix}}$	$(\mathbf{0}.2)_{\text{dix}}$	$(0.2)_{\text{dix}}$	$[\mathbf{0}]$
$(\mathbf{0}.2)_{\text{dix}}$	$(\mathbf{0}.4)_{\text{dix}}$	$(0.4)_{\text{dix}}$	$[0, \mathbf{0}]$
$(0.4)_{\text{dix}}$	$(\mathbf{0}.8)_{\text{dix}}$	$(0.8)_{\text{dix}}$	$[0, 0, \mathbf{0}]$
$(0.8)_{\text{dix}}$	$(\mathbf{1}.6)_{\text{dix}}$	$(0.6)_{\text{dix}}$	$[0, 0, 0, \mathbf{1}]$
$(0.6)_{\text{dix}}$	$(\mathbf{1}.2)_{\text{dix}}$	$(0.2)_{\text{dix}}$	$[0, 0, 0, 1, \mathbf{1}]$
$(\mathbf{0}.2)_{\text{dix}}$	$(\mathbf{0}.4)_{\text{dix}}$	$(0.4)_{\text{dix}}$	$[0, 0, 0, 1, 1, \mathbf{0}]$
…	…	…	…

Ainsi,

$$(0.1)_{\text{dix}} = (0.000110\ 0110\ 0110 \ldots)_{\text{vf}}.$$

Ce rationnel n'admet pas d'écriture finie en représentation à virgule fixe.

Pour éviter ce comportement, il suffit de borner la taille de la partie décimale calculée (et donc réaliser au plus m tours de boucle).

Représentation IEEE 754

Le codage IEEE 754 consiste à représenter un nombre à virgule x par trois données :

1. un **bit de signe** s ;
2. un **exposant** e ;
3. une **mantisse** m.

Principaux formats (tailles en bits) :

Nom	Taille totale	Bit de signe	Exposant	Mantisse
half	16	1	5	10
single	32	1	8	23
double	64	1	11	52
quad	128	1	15	112

Représentation IEEE 754

Le **bit de signe** s renseigne sur le signe : si $s = 0$, le nombre codé est positif, sinon il est négatif.

L'**exposant** e code un entier en représentation avec biais avec un biais donné par la la table suivante.

Nom	**Biais** B
half	15
single	127
double	1023
quad	16383

On note $\mathrm{vale}(e)$ la valeur ainsi codée.

La **mantisse** m code la partie décimale d'un nombre de la forme $(1.m)_{\mathrm{vf}}$. On note $\mathrm{valm}(m)$ la valeur ainsi codée.

Le nombre à virgule codé par s, e et m est

$$x = (-1)^s \times \mathrm{valm}(m) \times 2^{\mathrm{vale}(e)}.$$

– Exemple –

Codons le nombre $x := (-23.375)_{\text{dix}}$ en single.

1. Comme x est négatif, $s := 1$.
2. On écrit $|x|$ en représentation à virgule fixe. On obtient $|x| = (\texttt{10111.011})_{\text{vf}}$.
   Dans $\mathbf{1}\texttt{0111.011}$, la virgule est placée 4 positions à droite du bit à 1 de poids le plus fort. Ainsi, $|x| = \text{valm}(\texttt{0111011}) \times 2^4$.
3. On en déduit $\text{vale}(e) = 4$ et ainsi, $e = (\texttt{10000011})_{\text{biais}=127}$.
4. On en déduit par ailleurs que $m = \texttt{01110110000000000000000}$.

Finalement, $x = (\texttt{1 10000011 01110110000000000000000})_{\text{IEEE 754 single}}$.

– Exemple –

Codons le nombre $x := (0.15625)_{\text{dix}}$ en single.

1. Comme x est positif, $s := 0$.
2. On écrit $|x|$ en représentation à virgule fixe. On obtient $|x| = (0.00101)_{\text{vf}}$.
   Dans $0.00\mathbf{1}01$, la virgule est placée 3 positions à gauche du bit à 1 de poids le plus fort. Ainsi, $|x| = \text{valm}(01) \times 2^{-3}$.
3. On en déduit $\text{vale}(e) = -3$ et ainsi, $e = (01111100)_{\text{biais}=127}$.
4. On en déduit par ailleurs que $m = 01000000000000000000000$.

Finalement, $x = (0\ 01111100\ 01000000000000000000000)_{\text{IEEE 754 single}}$.

Représentation IEEE 754

Il existe plusieurs **représentations spéciales** pour coder certains éléments.

Valeur	s	e	m
Zéro	0	0…0	000…0
+Infini	0	1…1	000…0
−Infini	1	1…1	000…0
NaN	0	1…1	010…0

« NaN » signifie « Not a Number ». C'est un code qui permet de représenter une valeur mal définie provenant d'une opération qui n'a pas de sens (p.ex., $\frac{0}{0}$, $\infty - \infty$ ou $0 \times \infty$).

Le code ASCII

ASCII est l'acronyme de **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. Ce codage des caractères fut introduit dans les années 1960.

Un caractère occupe **un octet** dont le bit de poids fort vaut 0.

La correspondance octet (en héxa.) / caractère est donnée par la table

	0x	1x	2x	3x	4x	5x	6x	7x
x0	NUL	DLE	esp.	0	@	P	‘	p
x1	SOH	DC1	!	1	A	Q	a	q
x2	STX	DC2	"	2	B	R	b	r
x3	ETX	DC3	#	3	C	S	c	s
x4	EOT	DC4	$	4	D	T	d	t
x5	ENQ	NAK	%	5	E	U	e	u
x6	ACK	SYN	&	6	F	V	f	v
x7	BEL	ETB	’	7	G	W	g	w
x8	BS	CAN	(	8	H	X	h	x
x9	HT	EM	)	9	I	Y	i	y
xA	LF	SUB	*	:	J	Z	j	z
xB	VT	ESC	+	;	K	[	k	{
xC	FF	FS	,	<	L	\	l	|
xD	CR	GS	-	=	M	]	m	}
xE	SO	RS	.	>	N	^	n	~
xF	SI	US	/	?	O	_	o	DEL

Le code ISO 8859-1

Ce codage est également appelé Latin-1 ou Europe occidentale et fut introduit en 1986.

Un caractère occupe **un octet**. La valeur du bit de poids fort n'est plus fixée.

À la différence du code ASCII, ce codage permet en plus de représenter, entre autres, des lettres accentuées.

Ce codage des caractères est aujourd'hui (2021) de moins en moins utilisé.

Le code Unicode

Le codage Unicode est une version encore étendue du code ASCII qui fut introduite en 1991.

Chaque caractère est représenté sur **deux octets**.

Il permet ainsi de représenter une large variété de caractères : caractères latins (accentués ou non), grecs, cyrilliques, *etc.*

Il existe des extensions où plus d'octets encore par caractère sont utilisés.

Problème : un texte encodé en Unicode prend plus de place qu'en ASCII.

Le code UTF-8

Le codage UTF-8 apporte une réponse satisfaisante au problème précédent.

Voici comment un caractère c se représente selon le codage UTF-8 :

- si c est un caractère qui peut être représenté par le codage ASCII, alors c est représenté par son code ASCII ;
- sinon, c peut être représenté par le codage Unicode. Il est codé par **trois octets**

où

est la suite des deux octets du code Unicode de c.

Lorsqu'un texte contient principalement des caractères ASCII, son codage est en général moins coûteux en place que le codage Unicode.

De plus, il est **rétro-compatible** avec le codage ASCII.

Textes

Un texte est une suite de caractères.

Chacun des codages de caractères vus précédemment produit un codage de texte. En effet, un texte est représenté par la suite de bits obtenue en remplaçant chacun de ses caractères par son codage.

Un codage de texte est un code si toute suite de bits se décode en au plus un texte (il n'y a pas d'ambiguïté sur l'interprétation d'une suite de bits).

Exercice : vérifier que les codages ASCII, Latin-1, Unicode et UTF-8 sont bien des codes.