Nous nous intéressons à la représentation des données en machine. L'informatique travaille classiquement avec des *bits*, des unités permettant de représenter deux états logiques. D'où leur nom qui provient de la contraction des mots anglais *binary digits*. Les symboles utilisés pour représenter la valeur de bits sont les deux premiers chiffres du système décimal : 0 ou 1.

I_ Représentation des entiers

A - Entiers naturels dans une base quelconque

1. Notation positionnelle

Théorème Soit $\beta \in \mathbf{N}$, $\beta > 1$, une base. Tout entier $n \in \mathbf{N}$ peut être représenté de manière unique par sa **représentation positionnelle en base** β :

$$(a_{p-1}a_{p-2}...a_1a_0)_\beta \quad : \quad n = \sum_{i=0}^{p-1} a_i\beta^i$$

avec $a_i \in 0, ..., \beta - 1$ et $a_{p-1} \neq 0$.
Les notations $(a_{p-1}a_{p-2}...a_1a_0)_\beta$ ou $\overline{a_{p-1}a_{p-2}...a_1a_0}^\beta$ permettent de représenter cette somme.
p est le nombre de chiffres nécessaires pour écrire l'entier n.

Exemple : $(3205)_{10} = 3 \times 10^3 + 2 \times 10^2 + 0 \times 10^1 + 5 \times 10^0 = 3205$ en écriture décimale.

Vocabulaire : On appelle chiffre de *poids faible* le chiffre qui, dans la représentation positionnelle en base β, est associé à la plus petite puissance de β. À l'inverse, le chiffre de *poids fort* est le chiffre non nul associé à la plus grande puissance de β.
En base 2, on parlera donc de *bit de poids faible* et de *bit de poids fort*.
Le *chiffre de poids fort* est par convention représenté à gauche dans la représentation positionnelle usuelle.

2. Divisions euclidiennes successives

Le reste dans la division euclidienne d'un entier naturel n par β donne le chiffre de poids faible dans la représentation de n en base β.

En effet,

$$\begin{aligned} n &= a_{p-1}\beta^{p-1} + a_{p-2}\beta^{p-2} + \cdots + a_2\beta^2 + a_1\beta^1 + a_0, \\ &= \underbrace{(a_{p-1}\beta^{p-2} + a_{p-2}\beta^{p-3} + \cdots + a_2\beta^1 + a_1)}_{\text{quotient}}\beta + \underbrace{a_0}_{\text{reste}} \end{aligned}$$

avec, rappel, $0 \leq a_0 < \beta$. Donc $n \mod \beta - a_0$.

3. Existence de la représentation positionnelle

Démontrons l'existence de la représentation positionnelle dans une base $\beta > 1$ pour tout nombre entier $n \in \mathbf{N}$ par récurrence forte sur n.

Initialisation Pour $n = 1$, $n = 1 \times \beta^0$ puisqu'on choisit $\beta > 1$. Remarque : pour $n = 0$, la représentation est vide par convention.

Hérédité Supposons l'existence de la représentation positionnelle pour tout nombre entier k tel que $0 \leq k < n$ et considérons le nombre n.
Puisque $n > 0$ et $\beta > 1$, il existe des entiers naturels q et r tels que $n = q\beta + r$ avec $0 \leq r < \beta$ et $0 \leq q < \beta$[1]. Si $q = 0$, on a terminé : $n = r\beta^0$.
Sinon, puisque $q < n$, l'hypothèse de récurrence amène $q = \sum_{i=0}^{p-1} a_i\beta^i$, avec $a_{p\ \ 1} \neq 0$.

1. voir cours de maths

On obtient

$$\begin{aligned} n &= \beta \sum_{i=0}^{p-1} a_i \beta^i + r \\ &= \sum_{i=1}^{p} a_{i-1} \beta^i + r \\ &= \sum_{i=0}^{p} \overline{a_i} \beta^i \quad \text{avec} \quad \overline{a_0} = r \quad \text{et} \quad \overline{a_i} = a_{i-1} \quad \text{pour} \quad i \geq 1 \end{aligned}$$

Les $\overline{a_i}$ sont tels que $0 \leq \overline{a_i} < \beta$ et $\overline{a_p} = a_{p-1} \neq 0$. □

4. Nombres de chiffres représentables

En base β, un nombre écrit avec p chiffres pourra représenter β^p valeurs différentes. En choisissant la première valeur à 0, les nombres représentables seront donc compris entre 0 et $\beta^p - 1$.

5. Codage en machine

En machine, le nombre de bits p de la représentation positionnelle est fixé pour les différents *formats de codage* (entiers non signés de taille 8, 16, 32 ou 64 bits).
Lorsqu'on effectue une opération arithmétique entre entiers naturels (types `unsigned` en C) sur p bits, le résultat m est obtenu $\mod 2^p$ ($\mod$ représentant le modulo, c'est-à-dire le reste de la division euclidienne).

B - Entiers relatifs en base 2

Il existe plusieurs manières de représenter un entier relatif.
La représentation choisie est dite la *représentation en complément à 2 sur p bits*.

$$(c_{p-1}c_{p-2}\dots c_1c_0)_{\overline{2}} \quad : \quad -c_{p-1}2^{p-1} + \sum_{i=0}^{p-2} c_i 2^i$$

On peut montrer que :

$$\begin{cases} n \geq 0 & \text{ssi} \quad c_{p-1} = 0, \\ n < 0 & \text{ssi} \quad c_{p-1} = 1. \end{cases}$$

Opposé en complément à 2 sur p bits Le codage de $-n$ en complément à 2 sur p bits est le même que celui de l'entier naturel $(\overline{c}_{p-1}\overline{c}_{p-2}\dots\overline{c}_1\overline{c}_0)_2 + 1 \mod 2^p$, avec $\overline{c}$ valant 1 si $c = 0$ et 0 si $c = 1$, et n représenté par $(c_{p-1}c_{p-2}\dots c_1c_0)_2$.

Addition en complément à 2 sur p bits Soit $m = (c_m)_{\overline{2}}$ et $n = (c_n)_{\overline{2}}$. En l'absence de dépassement de capacité, le codage de $m + n$ en complément à 2 sur p bits est le codage de $(c_m + c_n) \mod 2^p$ en tant qu'entier naturel. Si m et n sont de signes opposés, aucun dépassement n'est possible. Si m et n sont de même signe, il y a dépassement si et seulement si le signe du résultat calculé diffère du signe des opérandes.

II_ Vers les nombres flottants

A - Représentation de rationnels

Les rationnels sont les nombres de la forme $\frac{p}{q}$, avec $p \in \mathbf{N}$ et $q \in \mathbf{N} - \{0\}$.
L'écriture d'un rationnel en base β n'est pas forcément finie, mais si elle n'est pas finie, elle est nécessairement périodique.
Soit un nombre x tel que $0 \leq x < 1$ dont on connaît l'écriture décimale en base 10. On veut déterminer son écriture en binaire de la forme :

$$x = (0, x_1 \dots x_q)_2$$

Puisque $2x = (x_1, x_2 \dots x_q)_2$ on a $x_1 = \lfloor 2x \rfloor$.

Exemple convertir $1/10 = (0,1)_{10}$ en binaire.

II_ Vers les nombres flottants

A - Représentation de rationnels

Rappel 1

Les rationnels sont les nombres de la forme $\frac{p}{q}$ avec $p \in \mathbf{N}$ et $q \in \mathbf{N}^*$.

Certains rationnels ont une écriture finie en base 10. Les autres ont une écriture périodique.
On détermine une *forme normalisée* pour les nombres rationnels (fraction irréductible) de la manière suivante : $\frac{p}{q} = \frac{p/\text{pgcd}(p,q)}{q/\text{pgcd}(p,q)}$.
Cette forme normalisée permet de tester l'égalité de deux rationnels très simplement. Si deux rationnels sont représentés de manière normalisée par $\frac{a}{b}$ et $\frac{c}{d}$, tester l'égalité requiert de comparer les nombres a et c d'une part et les nombres b et d d'autre part.

Rappel 2

L'écriture scientifique d'un nombre réel dans une base $\beta > 1$ est l'écriture sous la forme $m \times \beta^{\mathrm{e}}$ avec $\mathrm{e} \in \mathbf{Z}$ et $1 \leq m < \beta$. Le nombre m est appelé la *mantisse*.

B - Nombres dyadiques

Définition 1

On définit l'ensemble des nombres dyadiques $\mathbf{Y} = \{\frac{a}{2^b} \mid (a,b) \in \mathbf{Z} \times \mathbf{N}\}$.

Propriété 1

On a :
- $\mathbf{N} \subset \mathbf{Y}$ (rappel : $\forall a \in \mathbf{N}, \left(\frac{a}{2^0}\right) \in \mathbf{Y}$)
- $\mathbf{Y} \subset \mathbf{Q}$
- mais $\mathbf{Q} \not\subset \mathbf{Y}$

Les nombres dyadiques sont les réels qui s'écrivent avec un nombre fini de chiffres après la virgule en base 2.

C - Notation scientifique en base 2

Rappel 3

De la même manière, l'écriture scientifique d'un nombre en base 2 aura la forme $m \times 2^{\mathrm{e}}$ avec $\mathrm{e} \in \mathbf{Z}$ et $1 \leq m < 2$.

Puisque dans l'écriture scientifique la position de la virgule (du point !) est connue, ce point n'est pas représenté. Sur 4 bits, la mantisse représentant 1.5 sera codée $(1100)_2$, le bit de poids fort étant écrit à gauche (comme d'habitude).

Exercice 1

Déterminer les réels représentables en écriture scientifique en base 2 si la mantisse est codée sur 3 bits et pour des valeurs d'exposant compris entre -2 et 3.

Les nombres ne sont donc pas répartis uniformément ! ! ! Cela vient du fait que le dernier bit disponible (ici le troisième), n'a pas la même valeur en fonction de l'exposant.

Remarque Si on ajoute un bit de signe, on peut obtenir exactement tous les opposés des nombres positifs.

D - Nombres flottants en machine

Les nombres flottants sont usuellement représentés en machine en suivant la norme IEEE-754, introduite en 1985.
Voici la représentation utilisée avec la manière dont sont utilisés les bits d'un type flottant pour respecter la représentation.

$c =$

s	code de l'exposant : c_e	code de la mantisse : c_m
1 bit	k bits	$p-1$ bits

Dans la représentation ci-dessus, s vaut 1 si le nombre à représenter est négatif, 0 sinon.
Le code c_m de la mantisse utilisé est en fait la représentation de la partie fractionnaire de la mantisse m en binaire, c'est-à-dire sans le chiffre à gauche du point dans l'écriture scientifique en base 2. Le premier 1 est donc codé implicitement : $m = (1, c_m)_2$. Ainsi, si c_m est codé sur $p-1$ bits alors le nombre sera représenté avec p bits de précision.
Le code de l'exposant c_e est une représentation sur k bits de la valeur de l'exposant e avec la signification suivante :

$$\begin{cases} (c_e)_2 = 0 \quad \text{et} \quad (c_m)_2 = 0 & \text{le nombre représenté est zéro} \\ (c_e)_2 = 2^k - 1 \quad \text{et} \quad (c_m)_2 = 0 & \text{le nombre représenté est } \pm\,\text{Inf} \\ (c_e)_2 = 2^k - 1 \quad \text{et} \quad (c_m)_2 \neq 0 & \text{le nombre représenté est NaN} \\ 1 \leq (c_e)_2 \leq 2^k - 2 & \mathrm{e} = (c_e)_2 - (\underbrace{2^{k-1} - 1}_{\text{biais}}) \end{cases}$$

Si le nombre représenté est « normal », il vaut alors $(-1)^s \times m \times 2^{\mathrm{e}}$ avec m et e obtenus comme expliqué ci-dessus.

La norme définit la taille des différentes parties des nombres notamment pour les deux formats ci dessous :

simple precision sur 32 bits, avec 8 bits pour le code de l'exposant, 23 bits pour le code de la mantisse

double precision sur 64 bits, avec 11 bits pour le code de l'exposant, 52 bits pour le code de la mantisse

Les flottants de type `float` en C respectent le format **simple precision**. Les flottants de type `double` en C, `float` en OCaml (et `float` en Python) respectent le format **double precision**.

Exercice 2

Déterminer le plus petit nombre flottant normalisé représentable dans le format simple.

Exercice 3

Déterminer le plus grand nombre flottant normalisé représentable dans le format simple.

Les nombres flottants positifs normalisés représentables le format **simple** sont compris entre envrion 10^{-38} et 10^{38} environ. Les nombres positifs représentables avec le format **double** sont compris entre envrion 10^{-308} et 10^{308}.

Important 1

Les flottants sont une représentation **approchée** des réels.

Les nombres qui ne sont pas des nombres dyadiques n'ont pas de représentation positionnelle finie en base 2 : ils seront approchées. Un nombre qui aurait une représentation finie mais trop longue pour le format utilisée serait également approché.
On parle d'*overflow* quand un nombre est trop grand pour être représenté, il y a alors dépassement et la représentation binaire utilisée correspond à celle de l'infini. On parle d'*underflow* pour les nombres trop petits : dans ce cas ils peuvent être représentés de manière non normalisée.

Définition 2

La **précision machine**, notée ϵ, est définie comme la quantité telle que $x = 1 + \epsilon$ est le plus petit flottant représentable strictement supérieur à 1.

Avec les représentations apportées par la norme IEE-754, le ϵ machine vaut
— $\epsilon_{\text{simple}} = 2^{-23} \approx 1.2 \times 10^{-7}$
— $\epsilon_{\text{double}} = 2^{-52} \approx 2 \times 10^{-16}$
La norme IEEE-754 définit 4 modes d'arrondi : au plus proche, vers 0, vers l'infini positif et vers l'infini négatif.
Lorsqu'on enchaîne les calculs, les erreurs d'arrondis s'ajoutent les unes aux autres. De plus, même une opération s'effectuant sur deux nombres représentables de manière exacte en machine, peut conduire à un résultat faux !

III_ Types et opérations

Rappel 4

En C, il existe des types pour les entiers signés (`int`) et non signés (`unsigned int`). Les variantes permettant de manipuler des entiers non signés dont la taille est fixée sont `uint8_t` , `uint16_t` , `uint32_t` ou `uint64_t` , sur 8, 16, 32 ou 64 bits (`int8_t` , `int16_t` , `int32_t` , `int64_t` pour les signés).

Opérations sur des entiers

Question 1

Quel est le nombre de bits nécessaire pour représenter l'addition de deux nombres entiers non signés représentés sur n et m bits ?

Question 2

Quel est le nombre de bits nécessaire pour représenter la multiplication de deux nombres entiers non signés représentés sur n et m bits ?

Compléments

Nous avons vu qu'en C, il existe des opérateurs binaires utilisables sur des types d'entiers non signés : `|` , `&` , `^` .

L'opérateur `|` effectue un OU bit-à-bit. L'opérateur `&` effectue un ET bit-à-bit. L'opérateur `^` effectue un OU EXCLUSIF bit-à-bit.

Il est également possible d'effectuer des décalages des bits vers la gauche (ou vers la droite). Décaler de p bits vers la gauche revient à ajouter un nombre p de 0 à droite de la représentation binaire du nombre, c'est-à-dire à multiplier le nombre initial par 2^p.

En C, cet opérateur se note `<<` . Par exemple `1 << 2` vaudra $(100)_2$ soit 1×2^2.

Pour un décalage à droite, utiliser `>>` . Par exemple, `111 >> 1` vaudra $(11)_2$ soit $\lfloor 7/2 \rfloor$...

Règle 1

La valeur à droite de l'opérateur `<<` doit être inférieure à la précision du type utilisé !

Noms des opérateurs binaires en OCaml :

- `|` s'écrit `lor`
- `&` s'écrit `land`
- `^` s'écrit `lxor`
- `<<` s'écrit `lsl`
- `>>` s'écrit `lsr`

Opérateur unaire : permet la négation d'un mot.
Cet opérateur s'écrit `~` en C, `lnot` en OCaml.