Représentation Binaire

David Algis

27 mars 2025

1 Introduction

Les nombres binaires constituent la base de la représentation de l'information dans les systèmes numériques. En informatique, le système binaire est privilégié en raison de sa simplicité : il ne comporte que deux chiffres, `0` et `1`. Cette dualité permet une implémentation aisée sur les circuits électroniques, qui ne peuvent distinguer que deux états (par exemple, *on* et *off*).

2 Représentation Binaire des Entiers Positifs

Un nombre entier positif peut être écrit en base 2 comme la somme de puissances de 2. Plus précisément, un entier N s'exprime comme :

$$N = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0,$$

où chaque coefficient a_i appartient à l'ensemble $\{0, 1\}$.

2.1 Exemple de Conversion

Prenons l'exemple du nombre 10 en base décimale. Pour convertir 10 en binaire, on décompose :

$$10 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0,$$

ce qui donne la représentation binaire `1010`.

Exemple : Convertissons 13 en binaire.

— Divisons 13 par 2 : $13 \div 2 = 6$ avec un reste de 1.
— Divisons 6 par 2 : $6 \div 2 = 3$ avec un reste de 0.
— Divisons 3 par 2 : $3 \div 2 = 1$ avec un reste de 1.
— Divisons 1 par 2 : $1 \div 2 = 0$ avec un reste de 1.

En lisant les restes de bas en haut, on obtient la représentation binaire :

$$13_{\text{décimal}} = 1101_{\text{binaire}}.$$

Méthode de Conversion :

Algorithm 1 Conversion d'un entier décimal en représentation binaire par divisions successives

```
procedure DecimalEnBinaire(n)                    // Convertir n en représentation binaire
    if n = 0 then
        return "0"                                // Si n est nul, retourner "0"
    end if
    bits ← ""                                     // Initialiser la chaîne de bits
    while n > 0 do
        reste ← n mod 2                           // Calculer le reste de la division de n par 2
        bits ← str(reste) + bits                  // Ajouter le reste au début de la chaîne
        n ← n ÷ 2                                 // Diviser n par 2 et mettre à jour n
    end while
    return bits                                   // Retourner la représentation binaire obtenue
end procedure
```

3 Types et stockage

Les types de données en informatique sont définis pour garantir un stockage efficace et sûr de l'information dans une mémoire finie. Chaque type a une taille fixe en mémoire, ce qui influe sur la plage de valeurs qu'il peut représenter.

3.1 Entier signé et entier non signé

Les entiers peuvent être stockés en deux formats principaux : signé et non signé.

— **Entier non signé (unsigned)** : Dans ce format, tous les bits sont utilisés pour représenter la valeur numérique. Par exemple, un entier non signé sur 32 bits (`uint32`) peut représenter des valeurs de 0 à $2^{32} - 1$ (soit de 0 à 4 294 967 295).

— **Entier signé** : Ici, un bit (le plus significatif) est dédié au signe. La représentation la plus courante est le complément à deux. Un entier signé sur 32 bits (`int32`) représente des valeurs comprises entre -2^{31} et $2^{31} - 1$ (soit de -2 147 483 648 à 2 147 483 647). Le choix du format signé permet d'effectuer des opérations arithmétiques avec des nombres négatifs, au prix d'une réduction de la plage des valeurs positives.

3.2 Écriture binaire des nombres décimaux inférieur à un

Représentation binaire :

$$x = a_n \cdot 2^{-n} + a_{n-1} \cdot 2^{-n+1} + \cdots + a_1 \cdot 2^{-1},$$

où chaque coefficient a_i appartient à l'ensemble $\{0, 1\}$.

Pour représenter un nombre décimal x tel que $0 \leq x < 1$ en binaire, on utilise la méthode dite de la "multiplication par 2". L'idée est de multiplier x par 2 et d'extraire successivement la partie entière du résultat (qui sera 0 ou 1) afin de construire la partie fractionnaire en binaire. L'algorithme s'arrête soit lorsqu'une précision souhaitée est atteinte, soit lorsque x devient exactement 0.

L'algorithme peut être exprimé en pseudo-code de la manière suivante :

Algorithm 2 Conversion d'un nombre décimal x (avec $0 \leq x < 1$) en représentation binaire

```
1:  procedure FractionEnBinaire(x, precision)      // Convertir x en binaire avec la précision spécifiée
2:      resultat ← ""                              // Initialiser la chaîne de résultat
3:      for i ← 1 à precision do
4:          x ← x × 2
5:          bit ← partie entière de x              // Extraire le bit (0 ou 1)
6:          resultat ← resultat ‖ str(bit)         // Ajouter le bit à la fin de la chaîne
7:          x ← x − bit                            // Conserver uniquement la partie fractionnaire
8:          if x = 0 then
9:              arrêter                            // Sortir de la boucle si la fraction devient nulle
10:         end if
11:     end for
12:     return resultat
13: end procedure
```

Exemple : Convertissons 0.625 en binaire.

— $0.625 \times 2 = 1.25 \rightarrow$ le premier chiffre vaut `1` et la nouvelle fraction est 0.25.

— $0.25 \times 2 = 0.5 \rightarrow$ le deuxième chiffre vaut `0` et la nouvelle fraction est 0.5.

— $0.5 \times 2 = 1.0 \rightarrow$ le troisième chiffre vaut `1` et la fraction devient 0.

Ainsi, 0.625 se convertit en binaire comme 0.101.

3.3 Flottant simple et double précision

Les nombres à virgule flottante permettent de représenter des nombres réels en stockant à la fois la mantisse et l'exposant, ce qui permet de couvrir une large plage de valeurs. La norme **IEEE 754** définit les formats les plus courants comme les **Flottant simple précision (32 bits)** : Ce format se compose de :

— 1 bit pour le signe,

— 8 bits pour l'exposant,

— 23 bits pour la mantisse (fraction).

La valeur d'un flottant simple précision est calculée selon la formule :

$$(-1)^{\text{signe}} \times 1.\text{mantisse} \times 2^{\text{exposant}-127}$$

où 127 est le **biais** de l'exposant.

Exemple : Considérons le nombre 6.375. Nous allons détailler sa conversion en représentation binaire normalisée pour un flottant en simple précision.

1. **Conversion en binaire** :
   — La partie entière de 6 est convertie en binaire :

   $$6 = 4 + 2 = 2^2 + 2^1 \quad \Rightarrow \quad 6_{\text{décimal}} = \texttt{110}_{\text{binaire}}.$$

   — La partie fractionnaire $0.375_{\text{décimal}} = 0.011_{\text{binaire}}$.
   La représentation binaire complète de 6.375 est donc :

   $$6.375_{\text{décimal}} = 110.011_{\text{binaire}}.$$

2. **Mise en forme normalisée** : Pour obtenir la notation normalisée, on déplace la virgule binaire de manière à avoir un seul chiffre non nul avant la virgule :

   $$110.011 = 1.10011 \times 2^2.$$

   Le déplacement de la virgule de deux positions à gauche indique que l'exposant réel est 2.

3. **Détail du codage en simple précision (32 bits)** :
   — **Bit de signe :** Le nombre 6.375 est positif, donc le bit de signe est `0`.
   — **Exposant :** En simple précision, l'exposant est stocké avec un biais de 127. Ici, l'exposant réel est 2, donc l'exposant stocké sera :

   $$2 + 127 = 129.$$

   La valeur 129 s'écrit en binaire sur 8 bits comme :

   $$129 = \texttt{10000001}.$$

   — **Mantisse :** La mantisse correspond aux chiffres après la virgule dans la notation normalisée, ici `10011`. Comme le format simple précision exige 23 bits pour la mantisse, on complète avec des zéros à droite :

   $$\texttt{10011} \rightarrow \texttt{10011000000000000000000}.$$

Ainsi, la représentation complète de 6.375 en flottant simple précision est :

$$\underbrace{0}_{\text{Signe}} \quad \underbrace{10000001}_{\text{Exposant}} \quad \underbrace{10011000000000000000000}_{\text{Mantisse}}.$$