Chapitre 2 - La représentation des données

Cours de Bernard BoigelotUniversité de Liège

2025

Table des matières

1 Introduction

Ce chapitre s'intéresse à la manière dont l'information est codée, manipulée et représentée dans un ordinateur. Il couvre les représentations binaires, les bases numériques, les entiers (signés et non signés), les nombres réels (virgule fixe et flottante), ainsi que le codage des caractères.

Objectifs

- Comprendre les systèmes de numération : base 2, 10, 16.
- Maîtriser les conversions entre les représentations.
- Représenter et manipuler des entiers non signés et signés.
- Représenter des réels en virgule fixe et flottante (norme IEEE 754).
- Comprendre le codage des caractères (ASCII, ISO, Unicode).

2 Représentation des entiers non signés

2.1 Définition et principe

Un entier non signé est un nombre entier **positif ou nul**. Il est représenté en binaire à l'aide d'un certain nombre de bits. Chaque bit a un poids correspondant à une puissance de 2, en fonction de sa position (bit de poids faible à droite, bit de poids fort à gauche).

Formule générale :

$$V = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

Où b_i représente le bit à la position i, et n est le nombre total de bits.

2.2 Exemple 1 :

Représentation de 13 sur 4 bits :

$$13 = 8 + 4 + 0 + 1 = 2^3 + 2^2 + 2^0 \Rightarrow (1101)_2$$

2.3 Exemple 2 :

Convertir 255 en binaire sur 8 bits.

$$255 = 2^7 + 2^6 + ... + 2^0 \Rightarrow (11111111)_2$$

2.4 Exemple 3 : tableau binaire / décimal / hexadécimal

Décimal	Binaire	Hexadécimal
5	00000101	05
15	00001111	0F
63	00111111	3F
127	01111111	7F

2.5 Limites selon le nombre de bits

— Sur 4 bits : $[0, 15]$

— Sur 8 bits : $[0, 255]$

— Sur 16 bits : $[0, 65535]$

— Sur 32 bits : $[0, 4294967295]$

2.6 Exercices types examens

— **Q1 :** Convertir 42 en binaire et en hexadécimal.

— **Q2 :** Donner la valeur décimale de $(10101101)_2$.

— **Q3 :** Quel est l'entier maximum représentable sur 12 bits ?

2.7 Astuce : Conversion rapide binaire → hexadécimal

Découper la suite binaire en groupes de 4 bits depuis la droite, et convertir chaque groupe avec la table :

Bin	Hex	Bin	Hex	Bin	Hex
0000	0	0100	4	1000	8
0001	1	0101	5	1001	9
0010	2	0110	6	1010	A
0011	3	0111	7	1011	B
				1100	C
				1101	D
				1110	E
				1111	F

3 Représentation des entiers signés

3.1 Pourquoi une représentation spécifique ?

Les entiers signés permettent de représenter des valeurs positives et négatives. Cela nécessite une convention pour différencier un nombre comme $+5$ de -5.

3.2 Méthodes de codage

1. **Bit de signe (sign-magnitude)** : 1 bit pour le signe, les autres pour la valeur absolue.
2. **Complément à un** : inverser les bits du positif.
3. **Complément à deux (C2)** : méthode standard.

3.3 Complément à deux

Pour obtenir le complément à deux d'un nombre :

— Prendre sa valeur absolue en binaire

— Inverser tous les bits

— Ajouter 1

Exemple :

$$+5 = 00000101 \Rightarrow -5 : 11111010 + 1 = 11111011$$

3.4 Intervalles représentables

Bits	Non signé	Signé (C2)
4	0 à 15	-8 à +7
8	0 à 255	-128 à +127
16	0 à 65535	-32768 à +32767

3.5 Exemple d'interprétation

Interpréter 11111100 sur 8 bits en signé (C2) :

— Valeur décimale : $2^8 - 4 = 252$ (en non signé)

— En signé : $256 - 252 = 4 \Rightarrow -4$

3.6 Exercices type examen

— **Q1 :** Donner le C2 de -14 sur 8 bits.

— **Q2 :** Quelle est la représentation de -1 sur 8 bits ? (Réponse : 11111111)

— **Q3 :** Quelle valeur signe-t-on si on lit 10000000 ? (Réponse : -128 sur 8 bits)

3.7 Astuces / erreurs fréquentes

— -1 = tous les bits à 1 !

— $+0 = -0$ en C2 (unique représentation du zéro)

— Valeur min en C2 = pas de positif opposé exact

4 Représentation des nombres réels

4.1 Pourquoi une représentation spéciale ?

Contrairement aux entiers, les nombres réels ne peuvent pas être représentés de manière exacte dans tous les cas en binaire (ex : 0.1 est infini en binaire). Deux approches principales existent : la **virgule fixe** et la **virgule flottante**.

4.2 Virgule fixe

— Les bits sont divisés entre partie entière et partie fractionnaire (ex : 4 bits + 4 bits).
— Simple à implémenter, utilisé dans les microcontrôleurs.
— Exemple : 00011010 = 1.625 si 4.4 bits avec poids 2^3 à 2^{-4}.

4.3 Virgule flottante (IEEE 754)

Représentation standardisée pour les calculs scientifiques. Un nombre est représenté par :

$$x = (-1)^s \cdot 1.m \cdot 2^{e-\text{biais}}$$

4.4 Simple précision (32 bits)

— 1 bit de signe
— 8 bits pour l'exposant (biais = 127)
— 23 bits pour la mantisse (fractionnaire)

4.5 Exemple : représenter -6.25

— Binaire : $-110.01 = -1.1001 \cdot 2^2$
— Signe = 1
— Exposant = $2 + 127 = 129 \Rightarrow 10000001$
— Mantisse : 10010000000000000000000

IEEE final : `1 10000001 10010000000000000000000`

4.6 Catégories spéciales (IEEE 754)

— Zéro (exposant et mantisse = 0)
— Infini (+, -)
— NaN (Not a Number)
— Dénormalisés (exposant nul, mantisse 0)

4.7 Erreurs d'arrondi fréquentes

— 0.1 0001100110011... (infini binaire)
— Des tests comme `if(x == 0.1)` peuvent échouer
— Utiliser un epsilon de tolérance dans les comparaisons

4.8 Exercices type examen

— Q1 : Écrire 5.75 en IEEE simple précision.
— Q2 : Pourquoi 0.2 ne peut pas être codé exactement ?
— Q3 : Que signifie un exposant de 255 en IEEE ? (Rép : Infini ou NaN)

5 Codage des textes (ASCII, ISO, Unicode)

5.1 ASCII (American Standard Code for Information Interchange)

— Codage sur 7 bits (128 caractères)

— Inclut : lettres, chiffres, symboles, codes de contrôle (`\n`, `\r`)

— Exemple : A = 65, B = 66, a = 97

5.2 ISO 8859-1 (Latin-1)

— Extension ASCII en 8 bits (256 caractères)

— Ajoute caractères accentués : é, è, à, ç, etc.

— Très utilisé en Europe occidentale

5.3 Unicode

— Codage universel pour toutes les langues

— UTF-8 : taille variable (1 à 4 octets), compatible ASCII

— UTF-16 : 2 ou 4 octets

— Exemple : `é` = C3 A9 (UTF-8), `€` = E2 82 AC

5.4 Exemples de codage

Caractère	ASCII (déc)	UTF-8 (hex)	Description
A	65	41	Lettre majuscule A
`é`	-	C3 A9	Accentué latin
`€`	-	E2 82 AC	Euro (symbole monétaire)
	-	E4 B8 AD	Caractère chinois

5.5 Compatibilité entre formats

— UTF-8 conserve la compatibilité avec ASCII pour les 128 premiers codes

— Attention aux erreurs d'encodage : ouvrir un fichier UTF-8 avec ISO cause des caractères illisibles

5.6 Exercices types examens

— Q1 : Donner la valeur ASCII de la lettre Z.

— Q2 : Combien d'octets sont utilisés pour le caractère `€` en UTF-8 ?

— Q3 : Quelle différence entre ASCII et UTF-8 ?

5.7 Astuce

ASCII UTF-8 Tout fichier ASCII est valide en UTF-8 !

6 Exemples types examens, flashcards et astuces

6.1 Exemples type examen corrigés

Exemple 1 — Conversion base : Convertir 2023 en binaire et hexadécimal.

$$2023 = 11111100111_2 = 7E7_{16}$$

Exemple 2 — Complément à deux : Donner C2 de -45 sur 8 bits.

$$45 = 00101101 \Rightarrow \text{inverse} : 11010010 + 1 = 11010011$$

Exemple 3 — IEEE 754 : Codage de $+3.5$ en simple précision.

- $3.5 = 1.11 \cdot 2^1$
- Exposant : $127 + 1 = 128 = 10000000$
- Mantisse : 11000000000000000000000

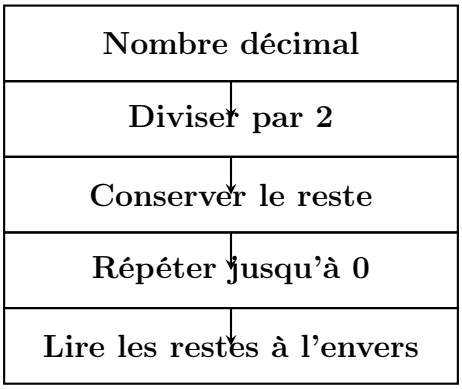
```
0 10000000 11000000000000000000000
```

6.2 Schéma : format IEEE 754 (32 bits)

6.3 Flashcards à mémoriser

- C2 de -1 sur 8 bits : `11111111`
- $[0, 2^n - 1]$: plage non signée
- ASCII A : 65
- IEEE mantisse : sans le 1 implicite
- UTF-8 : variable, compatible ASCII
- $I = \log_2(1/p)$: information
- 4 bits : max 15
- 8 bits C2 : [-128 ; 127]

6.4 Schéma : conversion décimal → binaire



6.5 Comment représenter un entier signé étape par étape (ex : -18)

Pour représenter correctement un entier négatif comme -18 sur 8 bits, on passe par plusieurs étapes logiques. Voici le processus détaillé :

1. **Conversion en binaire de la valeur absolue**
   - On commence par convertir 18 en binaire. Cela donne :

$$18 = 16 + 2 = 2^4 + 2^1 \Rightarrow 00010010 \text{ (sur 8 bits)}$$

2. **Représentation avec bit de signe (valeur signée)**
   - On conserve les 7 bits de la valeur absolue, et on ajoute un **bit de signe** en tête : 0 pour positif, 1 pour négatif.
   - Donc ici : -18 devient `10010010` (1 pour le signe, 0010010 pour 18).
   - **Problème :** cette représentation est ambigüe pour les opérations binaires.
3. **Complément à un (C1)**
   - On prend la représentation positive (00010010) et on **inverse chaque bit** :

$$00010010 \Rightarrow 11101101$$

   - Cette opération retourne la version C1 de -18.
4. **Complément à deux (C2)**
   - On ajoute 1 au résultat précédent (C1) :

$$11101101 + 1 = 11101110$$

   - Ce résultat est la **représentation finale en complément à deux** de -18.
   - Elle est utilisée par les ordinateurs pour les entiers signés car elle permet d'effectuer les additions et soustractions comme pour les non signés.

Résumé :

- Décimal : -18
- Binaire absolu (8 bits) : 00010010
- Signe-magnitude : 10010010
- Complément à un : 11101101
- Complément à deux : 11101110 **(utilisé en pratique)**

6.6 Astuces et erreurs fréquentes

- Toujours tester le nombre de bits nécessaires avant de stocker un entier
- Le zéro n'a qu'une seule représentation en C2
- IEEE 754 n'est pas exact : arrondis fréquents
- Ne jamais comparer des réels directement (préférer une tolérance ϵ)
- En UTF-8, `é` n'est pas en un seul octet

7 Conclusion

Ce chapitre vous a permis de poser des bases solides sur la manière dont les ordinateurs codent l'information. En comprenant comment fonctionnent les entiers, les flottants et les caractères, vous êtes mieux préparé pour les couches logicielles et matérielles à venir. Maîtriser ces codages est essentiel pour tout développement efficace en bas niveau ou dans les systèmes embarqués.