Représentation des données	Chap 03	**Représentation des nombres**

3.1 Représenter des nombres : cela dépend de la base

Idée centrale : la valeur dépend des poids des chiffres

Dans la vie courante, on utilise la **base 10** : les chiffres vont de 0 à 9, nous avons dix doigts après tout.
Mais un ordinateur représente les données avec des **bits** (0 ou 1), le plus pratique quand on manipule du courant, donc naturellement en **base 2**. Nous avons déjà vu ce principe pour l'algèbre booléenne.
Dans cette section nous cherchons à exprimer un nombre dans plusieurs bases pour le traduire d'humain à ordinateur notamment et comment faire des opérations dessus.

3.1.1 Écriture en base 10 (décimale)

Définition 1 — Écriture décimale

En **base 10**, les chiffres autorisés sont :

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9.$$

Les poids des positions sont des puissances de 10 :

$$1,\ 10,\ 10^2 = 100,\ 10^3 = 1000,\ \ldots$$

Exemple 1 — Décomposer un nombre en base 10

$$3472_{10} = 3 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0.$$

Autrement dit :

$$3472_{10} = 3000 + 400 + 70 + 2.$$

Notation

On peut écrire un nombre sans préciser la base (c'est implicite en base 10). Pour éviter toute ambiguïté, on note parfois :

$$(3472)_{10}.$$

Exercice 1 — Décomposition en base 10

1. Ecrire la décomposition et la valeur de $(12756)_{10}$

---

2. Ecrire la décomposition et la valeur de $(608)_{10}$

3.1.2 Écriture en base 2 (binaire)

Définition 2 — Binaire

En **base 2**, on n'a que deux chiffres :

$$0 \quad \text{et} \quad 1.$$

Les poids des positions sont des puissances de 2 :

$$1,\ 2,\ 4,\ 8,\ 16,\ 32,\ \ldots$$

Mais le principe reste absolument le même !

Exemple 2 — Décomposer un binaire

$$(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0.$$

Donc :

$$(101101)_2 = 32 + 0 + 8 + 4 + 0 + 1 = 45_{10}.$$

Lecture rapide

Dans un nombre binaire, un chiffre **1** signifie que le poids correspondant **est présent**, un chiffre **0** signifie qu'il **n'est pas présent**.

Exercice 2 — Décomposition en base 2

1. Ecrire la décomposition et la valeur de $(11111111)_2$

---

2. Ecrire la décomposition et la valeur de $(101010)_2$

3.1.3 Écriture en base 16 (hexadécimale)

Définition 3 — Hexadécimal

En **base 16**, on a 16 symboles. Comme on ne peut pas écrire un chiffre " 10 ", " 11 ", etc., on utilise des lettres pour compléter.
Les chiffres sont donc $1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$.
Les poids des positions sont des puissances de 16 :

$$1,\ 16,\ 16^2 = 256,\ 16^3 = 4096,\ \ldots$$

Exemple 3 — Décomposer un hexadécimal

$$(2A3)_{16} = 2 \times 16^2 + A \times 16^1 + 3 \times 16^0.$$

Or $A = 10$, donc :

$$(2A3)_{16} = 2 \times 256 + 10 \times 16 + 3 = 512 + 160 + 3 = 675_{10}.$$

Pourquoi l'hexadécimal ?

L'hexadécimal est très utilisé en informatique car il permet d'écrire des suites binaires **beaucoup plus courtes** : un chiffre hexadécimal représente exactement **4 bits**.

Exercice 3 — Décomposition en base 16

1) Décomposer le nombre héxadécimal $(B2E)_{16}$ et donner sa valeur.

---

2) Décomposer le nombre héxadécimal $(1297)_{16}$ et donner sa valeur.

Autres bases

De manière générale, on peut écrire un nombre dans n'importe quelle base b, dans ce cas :

- on dispose de b **symboles** : $0, 1, \ldots, b-1$;
- chaque position a un **poids** : $1, b, b^2, b^3, \ldots$;
- la valeur du nombre est une **somme pondérée** des chiffres.

$$(a_k a_{k-1} \ldots a_1 a_0)_b = \sum_{i=0}^{k} a_i \times b^i \qquad \text{avec} \qquad 0 \leq a_i < b.$$

Exercice 4 — Tableau de conversion des bases

Compléter le tableau suivant en écrivant la représentation de chaque nombre en **base 10**, en **base 2** (binaire) et en **base 16** (hexadécimal).

Base 10	Base 2	Base 16
0		
1		
2		
3		

3.1.4 Convertir d'une base à une autre

Convertir, c'est conserver la valeur

Un même nombre peut s'écrire de plusieurs façons selon la base, mais sa **valeur** ne change pas.
Exemple :

$$13_{10} = 1101_2 = D_{16}.$$

Base 2 ou 16 vers base 10 : méthode par décomposition

Méthode

Pour passer de la base 2 (ou 16) à la base 10, on l'a déjà fait précédemment :

— on multiplie chaque chiffre par son poids (2^i ou 16^i),

— puis on additionne.

Exemple 4 — Binaire → décimal

$$(110010)_2 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 32 + 16 + 2 = 50_{10}.$$

$$(AE1)_16 = 10 \times 16^2 + 14 \times 16^1 + 1 \times 16^0 = 2560 + 224 + 1 = 2785_{10}.$$

Exercice 5 — On se refait un peu de décomposition

Convertir en base 10 :

1) $(011011)_2$

---

2) $(EDF3)_{16}$

---

3) $(11001010)_2$

---

4) $(111)_{16}$

Base 10 vers elle-même

Définition 4 — Méthode des divisions successives en base 10

Cela ne parait peut-être pas très intéressant mais essayons de traduire un nombre N écrit en base 10 dans la base 10 !
Pour écrire un nombre N_{10} en base 10 :

- on divise N par 10,
- on note le reste (entre 0 et 9),
- on recommence avec le quotient,
- on lit les restes **de bas en haut**.

Exemple 5 — Décimal $\rightarrow$ décimal

Convertissons $12\,348_{10}$ en base 10 à l'aide de la méthode des divisions successives.

Nombre	Division par 10	Reste
12 348	$12\,348 = 10 \times 1\,234 + 8$	8
1 234	$1\,234 = 10 \times 123 + 4$	4
123	$123 = 10 \times 12 + 3$	3
12	$12 = 10 \times 1 + 2$	2
1	$1 = 10 \times 0 + 1$	1

On lit les restes du bas vers le haut :

$$12\,348_{10} = (12348)_{10}.$$

Interprétation

Le reste correspond au **chiffre des unités**, le quotient correspond à ce qu'il reste à écrire.

Autrement dit :

$$12\,348 = 1\,234 \times 10 + 8.$$

En répétant le raisonnement sur le quotient, on retrouve l'écriture complète du nombre en base 10.

Pourquoi on a fait ça

Ce qui est génial, c'est que cette méthode fonctionne pour **toutes les bases** !

Base 10 vers base 2 : divisions successives par 2

Définition 5 — Méthode des divisions successives

Pour convertir un nombre décimal en un nombre binaire, on applique la méthode vue précédemment.

Pour convertir N_{10} en binaire :

- on divise N par 2,
- on note le reste (0 ou 1),
- on recommence avec le quotient,
- on lit les restes **de bas en haut**.

Exemple 6 — Décimal $\rightarrow$ binaire

Convertissons 45_{10} en binaire.

Nombre	Division par 2	Reste
45	$45 = 2 \times 22 + 1$	1
22	$22 = 2 \times 11 + 0$	0
11	$11 = 2 \times 5 + 1$	1
5	$5 = 2 \times 2 + 1$	1
2	$2 = 2 \times 1 + 0$	0
1	$1 = 2 \times 0 + 1$	1

On lit les restes du bas vers le haut :

$$45_{10} = (101101)_2.$$

On peut vérifier en décomposant :

$$(101101)_2 = 32 + 8 + 4 + 1 = 45_{10}.$$

Exercice 6 — Conversion base 10 / base 2

Convertir en base 2 les nombres suivants :

1. 615_{10}

2. 2048_{10}

Base 10 vers base 16 : divisions successives par 16

Définition 6 — Méthode des divisions successives par 16

Pour passer de la base décimal à la base héxadécimal, on fait exactement la même chose mais en divisant par 16.
Pour convertir N_{10} en hexadécimal :

- on divise N par 16,
- on note le reste (entre 0 et 15),
- on remplace $10, 11, 12, 13, 14, 15$ par A, B, C, D, E, F,
- on lit les restes **de bas en haut**.

Exemple 7 — Décimal $\rightarrow$ hexadécimal

Convertissons 675_{10} en base 16.

Nombre	Division par 16	Reste
675	$675 = 16 \times 42 + 3$	3
42	$42 = 16 \times 2 + 10$	$10 = A$
2	$2 = 16 \times 0 + 2$	2

Donc :

$$675_{10} = (2A3)_{16}.$$

Exercice 7 — Conversion base 10 / base 16

Convertir en base 16 les nombres suivants :

1. 3741_{10}

2. 2048_{10}

Base 2 vers base 16 (et inversement) : paquets de 4 bits

Correspondance 1 chiffre hexa = 4 bits

Comme $16 = 2^4$, un chiffre hexadécimal correspond exactement à **4 bits**, il s'agit là de tout l'intérêt de la base 16 pour les informaticien.nes.

Exemples :

$$A_{16} = 10_{10} = 1010_2 \qquad F_{16} = 15_{10} = 1111_2 \qquad 2_{16} = 0010_2$$

Exemple 8 — Binaire → hexadécimal

Convertissons :

$$(101101011100)_2$$

On regroupe par 4 bits à partir de la droite :

$$1011 \,|\, 0101 \,|\, 1100$$

Puis on convertit chaque paquet :

$$1011_2 = B_{16}, \quad 0101_2 = 5_{16}, \quad 1100_2 = C_{16}.$$

Donc :

$$(101101011100)_2 = (B5C)_{16}.$$

Exemple 9 — Hexadécimal → binaire

$$(3F2)_{16} = 3\ F\ 2$$

Donc :

$$3_{16} = 0011_2, \quad F_{16} = 1111_2, \quad 2_{16} = 0010_2$$

Ainsi :

$$(3F2)_{16} = (0011\ 1111\ 0010)_2.$$

Exercice 8 — Conversions base 2 / base 16

Effectuer les conversions suivantes :

1) Convertir de la base 2 vers la base 16 :

a) $\left(1011\,0100\,1110\right)_2$

b) $\left(1101\,0011\,1001\right)_2$

2) Convertir de la base 16 vers la base 2 :

a) $(2F9)_{16}$

b) $(A7C)_{16}$

3.1.5 Additions en base 10, base 2 et base 16

Rappel : addition en base 10

Retenue en base 10

En base 10, dès que la somme dépasse 9, on écrit le chiffre des unités et on **retient** 1 dizaine.

Exemple 10 — Addition en base 10

$$\begin{array}{cccc} & 2 & 7 & 8 \\ + & 6 & 4 & 9 \\ \hline & 9 & 2 & 7 \end{array}$$

Explication rapide :

- $8 + 9 = 17$: j'écris 7 et je retiens 1 ;
- $7 + 4 + 1 = 12$: j'écris 2 et je retiens 1 ;
- $2 + 6 + 1 = 9$: j'écris 9.

Addition en base 2

Table d'addition binaire

En base 2, il n'y a que 4 cas possibles :

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	10 (donc somme 0 et retenue 1)

Exemple 11 — Addition en base 2

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

On peut lire cela comme :

$$101101_2 + 011011_2 = 1001000_2.$$

Exercice 9 — Additions

Effectuer l'addition en base 2 :

$$(110101)_2 + (10111)_2$$

Effectuer l'addition en base 2 :

$$(1110111)_2 + (100101)_2$$

Vérifier vos résultats en repassant en base 10.

Addition en base 16

Même principe, mais avec 16 symboles

En base 16, on additionne colonne par colonne. Dès que la somme atteint 16 (ou plus), on écrit le **reste de la division par 16** et on retient 1.

Exemple 12 — Addition en base 16

Calculons :

$$(7A9)_{16} + (3F5)_{16}.$$

$$\begin{array}{cccc} & 7 & A & 9 \\ + & 3 & F & 5 \\ \hline & B & 9 & E \end{array}$$

Détails :

- $9 + 5 = 14$ donc E ;
- $A + F = 10 + 15 = 25 = 16 + 9$ donc j'écris 9 et je retiens 1 ;
- $7 + 3 + 1 = 11$ donc B.

Ainsi :

$$(7A9)_{16} + (3F5)_{16} = (B9E)_{16}.$$

Exercice 10 — Additions

Effectuer l'addition en base 16 :

$$(124)_{16} + (89AE)_{16}$$

Effectuer l'addition en base 16 :

$$(9F)_{16} + (2B7)_{16}$$

Vérifier vos résultats en repassant en base 10.

3.2 Représentation des entiers relatifs

Une contrainte matérielle

Nous nous sommes jusque-là intéressé à l'aspect **mathématiques** de la représentation des nombres, mais notre but est **d'implémenter ces nombres en machine !**
Un ordinateur ne stocke pas des nombres " abstraits " : il les stocke dans des **zones mémoire de taille finie**.

Ces zones sont composées d'un nombre fixé de **bits** :

$$n = 8,\ 16,\ 32,\ 64,\ \ldots$$

Chaque entier est donc représenté par une **suite de n bits**.

Définition 7 — Mot binaire

Une suite de n bits stockée en mémoire s'appelle un **mot binaire**.

$$b_{n-1}\, b_{n-2}\, \ldots\, b_1\, b_0$$

Chaque bit ne peut valoir que 0 ou 1.
Avec n bits, on peut former 2^n mots binaires différents.
Dans un mot binaire :

- le **bit de poids faible** (LSB — **Least Significant Bit**) est le bit le plus à droite ;
- le **bit de poids fort** (MSB — **Most Significant Bit**) est le bit le plus à gauche.

Exemple 13 — Exemple sur 8 bits

$$\underbrace{0}_{\text{bit de poids fort}}\ 010110\ \underbrace{1}_{\text{bit de poids faible}}$$

Le bit de poids faible correspond au poids $2^0 = 1$, le bit suivant au poids $2^1 = 2$, etc. Notons qu'on garde le bit à gauche même s'il vaut 0 puisque dans l'exemple, on encode tout nos nombres sur 8 bits.

3.2.1 Entiers naturels codés sur n bits

Représenter un entier naturel

Pour représenter un **entier naturel**, la machine ne fait rien de particulier : elle écrit simplement le nombre en **binaire**, en plaçant les bits **dans l'ordre, du bit de poids fort au bit de poids faible**.

Chaque bit correspond à une puissance de 2, et la valeur du nombre est obtenue en additionnant les poids associés aux bits égaux à 1.

Bit	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeur	128	64	32	16	8	4	2	1

Par exemple, le nombre :

$$13_{10} = 8 + 4 + 1$$

est représenté par les bits :

$$00001101_2$$

car seuls les poids 8, 4 et 1 sont présents.

Il n'y a ici **aucune ambiguïté** : le nombre est positif, et sa valeur est entièrement déterminée par la position des bits à 1.

Plage de valeurs

Si nous souhaitons coder uniquement des **entiers naturels**, les valeurs possibles m sont :

$$0 \leq m \leq 2^n - 1.$$

En effet, le plus petit nombre est $(00\ldots0)_2 = 0_{10}$ et le plus grand nombre est

$$(111\ldots11)_2 = \sum_{0}^{n-1} 2^k = 2^n - 1$$

Exemple 14 — Cas de 8 bits

Sur 8 bits, on peut prendre valeurs m tel que :

$$0 \leq m \leq 255.$$

En effet,

$$00000000_2 = 0_{10}.$$

et

$$11111111_2 = 255_{10}.$$

3.2.2 Le cas des entiers négatifs

Un problème à résoudre

Les entiers naturels suffisent rarement, des nombres comme $-1,\ -2,\ -10,\ \ldots$ sont indispensables pour représenter des températures, des coordonnées, des calculs intermédiaires, etc.

Or un mot binaire ne **dit pas** s'il est positif ou négatif.

Première idée : le bit de signe

Définition 8 — Codage avec bit de signe

La première idée est de consacrer un bit, par exemple **le bit de poids fort** pour coder le **signe**.
Ainsi si le bit de poids fort vaut 0, le nombre est positif. Et s'il vaut 1, le nombre est négatif, donc comment faire ?
Les $n-1$ autres bits codent la valeur absolue.
Certes, on "perd" de l'information car ce bit ne sert plus à encoder une valeur mais on "retrouve" cette information en autorisant les nombres négatif.

Exemple 15 — Exemples sur 8 bits avec bit de signe

On code les entiers sur **8 bits** :

- le **bit de poids fort** représente le **signe** ;
- les **7 bits suivants** représentent la valeur absolue.

Exemple 1 : représentation de $+5$

S	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	1	0	1

Ici, le bit de signe vaut 0 : le nombre est positif.

Exemple 2 : représentation de -5

S	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	0	1	0	1

Ici, le bit de signe vaut 1 : le nombre est négatif.

Dans les deux cas, la **valeur absolue** est identique (5) : seul le bit de signe change.

Plage de valeur

Supposons que l'on code sur 8 bits, il existe donc $2^7 = 128$ valeurs possibles pour la valeur absolue (car 7 bits dispo) et un signe qui change avec le bit de signe.

On peut ainsi représenter les entiers :

$$-127 \leq m \leq +127.$$

Les valeurs $+128$ et -128 ne peuvent pas être représentées, car elles nécessiteraient une valeur absolue codée sur plus de 7 bits.

Exercice 11

On suppose qu'on code sur 8 bits. Transformer les entiers relatifs suivants en écriture binaire (en utilisant le **bit de signe**) :

1. $+18_{10}$

2. -18_{10}

3. $+42_{10}$

4. -7_{10}

Deux zéros

Avec cette représentation :

$$00000000 = +0 \qquad 10000000 = -0$$

Deux écritures pour un seul nombre : cela complique les calculs.

Problème pour l'addition

Considérons l'addition :

$$(+5) + (-3)$$

La représentation avec bit de signe de chacun des deux nombres est :

$$+5 = 00000101 \qquad -3 = 10000011$$

Effectuons l'addition binaire :

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

Résultat obtenu :

$$10001000$$

Interprétation avec bit de signe :

$$10001000 \rightarrow -8$$

Or :

$$(+5) + (-3) = +2$$

Le résultat est donc **faux**.

La solution utilisée en pratique : le complément à 2

Idée clé

La représentation en **complément à 2** permet de contrer tous les soucis de la représentation avec bit de signe :

- un seul zéro ;
- une addition unique pour tous les entiers ;

C'est le standard universel des processeurs modernes.

Définition 9 — Complément à 2 sur n bits

Sur une machine, un entier est stocké dans un mot de n **bits**. Il existe donc exactement :

$$2^n$$

mots binaires possibles.

La représentation en **complément à 2** repose sur l'idée suivante :

- les entiers positifs (ou nuls) sont représentés comme en binaire classique ;
- un entier négatif $-m$ est représenté par le nombre :

$$2^n - m.$$

Autrement dit, un nombre négatif est codé comme ce qu'il faut ajouter à m pour " revenir à zéro " après avoir parcouru tout l'espace des 2^n valeurs possibles.

Technique pour utiliser le complément à 2

On souhaite représenter $-m$ en machine, pour cela on respecte la méthode suivante

1. écrire $|m|$ en binaire sur n bits ;
2. inverser tous les bits (complément à 1) ;
3. ajouter 1.

Ces trois étapes permettent de calculer automatiquement $2^n - m$ et donc de représenter $-m$ dans l'ordinateur en complément à 2.

Exemple 16 — Représenter -12 sur 8 bits (complément à 2)

Sur 8 bits, il existe $2^8 = 256$ mots binaires possibles : de 0 à 255. En complément à 2, le nombre -12 est représenté par :

$$2^8 - 12 = 256 - 12 = 244.$$

On va retrouver ce résultat avec la méthode vue précédemment.

Étape 1 : représenter $|12|$ sur 8 bits

$$12_{10} = 00001100_2$$

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	1	1	0	0

Ici, on représente simplement la valeur **positive** 12.

Étape 2 : inversion des bits (complément à 1)
On inverse chaque bit : $0 \leftrightarrow 1$.

$$00001100_2 \longrightarrow 11110011_2$$

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	1	1	1	0	0	1	1

Ce mot binaire représente alors :

$$2^8 - 1 - 12 = 255 - 12 = 243.$$

C'est pour cela qu'on doit encore **ajouter 1** pour obtenir $2^8 - 12$.

Étape 3 : ajout de 1 (on obtient le complément à 2)

$$11110011_2 + 1 = 11110100_2$$

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	1	1	1	0	1	0	0

Ce mot binaire représente :

$$2^8 - 12 = 244.$$

Conclusion :

$$-12_{10} = 11110100_2 \quad \text{(sur 8 bits, en complément à 2).}$$

Exercice 12

Représenter sur 8 bits :

a) -1

b) -42

c) $+64$

Reconnaître un nombre négatif

Lorsque le bit de poids fort vaut 1 : le nombre est négatif, attention à ne pas confondre avec la représentation avec bit de signe.

Interpréter un nombre négatif en complément à 2

Pour trouver la valeur décimale d'un nombre négatif représenté en complément à 2 sur n bits :

1. on inverse tous les bits ;
2. on ajoute 1 ;
3. on interprète le résultat comme un entier positif ;
4. on prend l'opposé du nombre obtenu.

Cette méthode permet de retrouver la valeur absolue du nombre, puis son signe négatif.

Exercice 13

Interpréter :

11101001_2 10000000_2 01010100_2

Définition 10 — Plage de valeurs sur 8 bits

Sur 8 bits, il existe :

$$2^8 = 256$$

mots binaires possibles.

En complément à 2 :

- le mot 00000000_2 représente 0 ;
- le mot 01111111_2 représente $+127$;
- le mot 10000000_2 représente -128 ;
- le mot 11111111_2 représente -1.

On obtient donc la plage de valeurs :

$$-128 \leq m \leq 127.$$

Addition en complément à 2

L'un des grands avantages du **complément à 2** est qu'il permet d'utiliser **un seul algorithme d'addition** pour tous les entiers, qu'ils soient positifs ou négatifs.

En complément à 2 :

- on effectue une **addition binaire classique** bit à bit ;
- la **retenue finale éventuelle** est ignorée ;
- le résultat est correct **tant que la valeur obtenue reste dans l'intervalle représentable**.

En effet, sur n bits, les calculs sont réalisés sur un ensemble fini de 2^n valeurs. L'addition fonctionne donc comme une addition **modulo** 2^n, ce qui garantit un résultat cohérent tant qu'il n'y a pas de dépassement de capacité.
Par exemple, sur 8 bits, $-m$ est représenté par $2^8 - m$. Ajouter m donne 2^8, qui correspond à 0 sur 8 bits. C'est pour cela que l'addition fonctionne en complément à 2.

Exemple 17 — Addition $+7 + (-7)$ sur 8 bits

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

On ignore la retenue finale :

$$00000000_2 = 0.$$

Exercice 14

Effectuer l'addition sur 8 bits (et vérifier le résultat)

$$(-15) + 20$$

Effectuer l'addition sur 8 bits (et vérifier le résultat)

$$(-15) + (-33)$$

Pouvons-nous faire (-110) + (-105) ? Pourquoi ?