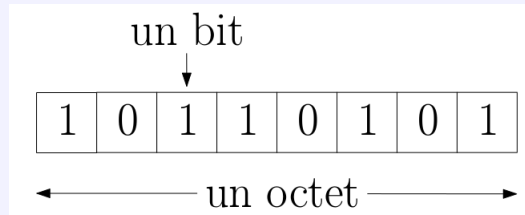


Introduction

La mémoire des ordinateurs est constituée d'une multitude de petits circuits électroniques (des transistors) et chacun ne peut être que dans deux états électriques : notés arbitrairement 0 et 1.

La valeur 0 ou 1 d'un circuit mémoire élémentaire s'appelle un **chiffre binaire**, un **booléen** ou un **bit** (abréviation de **binary digit**).

Depuis les années 1970, l'unité de mesure de l'espace (disque ou mémoire) est l'**octet** ou **byte**. Un octet correspond à 8 bits. Un octet peut donc prendre $2^8 = 256$ valeurs différentes.



En pratique, le processeur d'un ordinateur échange avec la mémoire des informations (données ou adresse mémoire) comportant plusieurs octets de 4 à 8 en général. On parle de **mots** qui caractérisent l'architecture de la machine (32 bits ou 64 bits).

Tout type d'information (nombre, caractère, couleur ...) peut être stocké sous forme de séquence de bits. Une **représentation interne** ou **codage** est nécessaire pour définir une représentation binaire. Dans ce chapitre, nous aborderons les codages simples des nombres entiers positifs et des entiers relatifs.

1 Représentation interne des entiers naturels (ou entiers non signés)

1.1 Représentation en base dix



Théorème-Définition 1

En base dix, les chiffres disponibles sont 0, 1, ..., 9. Le poids d'un chiffre est une puissance de dix qui dépend de sa position : $10^0 = 1$ pour les unités, $10^1 = 10$ pour les dizaines, $10^2 = 100$ pour les centaines, etc. Ainsi, 7307 s'écrit avec **quatre chiffres** :

$$7307 = 7 \times 10^3 + 3 \times 10^2 + 0 \times 10^1 + 7 \times 10^0.$$

Dans toute cette partie, la liste des chiffres est ordonnée **du poids le plus fort au poids le plus faible** : 7307 correspond à [7, 3, 0, 7]. Le chiffre des unités est donc à droite. Les fonctions ci-dessous prennent une liste non vide de chiffres valides ou un entier naturel, selon le sens de conversion.

1.1.1 De la liste des chiffres à l'entier : somme pondérée

💡 Algorithme 1 De la liste des chiffres à l'entier : algorithme de somme pondérée

- ➡ **Étape 1 :** Noter L la longueur de la liste et initialiser l'entier n à zéro.
- ➡ **Étape 2 :** Parcourir la liste de gauche à droite. Pour chaque chiffre d'indice i , calculer son poids 10^{L-1-i} .
- ➡ **Étape 3 :** Multiplier ce chiffre par son poids et ajouter le résultat à n . Répéter les étapes 2 et 3 jusqu'à la fin de la liste.

👉 **Étape 4 :** Renvoyer n , la somme des contributions.

⚠ **Attention :** l'indice dans la liste commence à zéro à gauche, alors que l'exposant commence à zéro à droite.

🔗 Exemple 1

L'entier naturel 7307 s'écrit en base 10 comme une liste de quatre chiffres dont le poids dépend de leur position dans l'écriture.

Indice i	0	1	2	3
Chiffre	7	3	0	7
Exposant $L - 1 - i$	3	2	1	0
Contribution	7000	300	0	7

La décomposition de 7307 en base 10 s'écrit donc : $7307 = 7 \times 10^3 + 3 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$.

🔗 Exercice 1

1. Pour chaque liste, écrire la décomposition avec les puissances de dix, puis calculer la valeur de l'entier.

a. [4, 2, 8].

.....

b. [6, 0, 5, 0].

.....

2. Compléter la fonction Python qui convertit une liste de chiffres en base dix en valeur de l'entier avec l'algorithme de somme pondérée. On donne un jeu de tests unitaires.

```
def chiffres2nombre(t: list[int]) -> int:

    n = 0

    for i in range(len(t)):

        expo = ...

        n = n + ...

    return n
```

```
assert chiffres2nombre([4, 2, 8]) == 428
assert chiffres2nombre([6, 0, 5, 0]) == 6050
assert chiffres2nombre([0]) == 0
```

1.1.2 De la liste des chiffres à l'entier : algorithme d'Horner

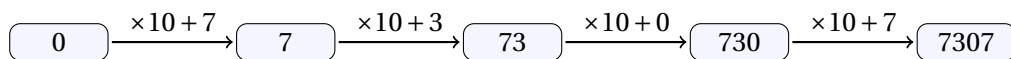
Algorithme 2 Algorithme d'Horner

L'algorithme d'Horner permet de convertir la liste des chiffres en entier. Attention le sens de parcours est fixé : par poids décroissant.

- 👉 **Étape 1 :** Initialiser l'entier n à zéro.
- 👉 **Étape 2 :** Lire les chiffres de la liste par poids décroissant (usuellement de gauche à droite).
- 👉 **Étape 3 :** Pour chaque chiffre lu, multiplier n par 10, puis ajouter ce chiffre : $n \leftarrow 10n + \text{chiffre}$. Répéter les étapes 2 et 3 jusqu'à la fin de la liste.
- 👉 **Étape 4 :** Renvoyer n .

Multiplier par dix décale les chiffres déjà lus d'un rang vers la gauche ; ajouter le chiffre suivant le place au rang des unités.

Exemple 2 La liste [7, 3, 0, 7]



Par exemple, après lecture des deux premiers chiffres, $n = 73$. Après lecture du zéro, $n = 73 \times 10 + 0 = 730$. Au final, $((7 \times 10 + 3) \times 10 + 0) \times 10 + 7 = 7307$.

Exercice 2 Horner en base dix

1. Pour chaque liste, représenter par un schéma comme dans l'exemple, l'application de l'algorithme d'Horner pour construire l'entier correspondant.
 - a. [4, 2, 8].
 - b. [6, 0, 5, 0].
2. Compléter la fonction Python qui convertit une liste de chiffres en base dix en valeur de l'entier avec l'algorithme d'Horner. On donne un jeu de tests unitaires.

```
def horner10(t: list[int]) -> int:
    n = ...
    for chiffre in ...:
        n = ...
    return n
```

```
assert horner10([4, 2, 8]) == 428
assert horner10([6, 0, 5, 0]) == 6050
assert horner10([0]) == 0
```

1.1.3 De l'entier à la liste des chiffres : divisions successives

Algorithme 3 Algorithme des divisions successives

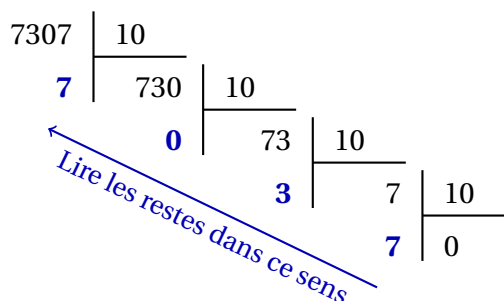
- **Étape 1 :** Si l'entier vaut zéro, renvoyer [0] et arrêter l'algorithme. Sinon, créer une liste vide.
- **Étape 2 :** Effectuer la division euclidienne de l'entier courant par 10 : $n = 10q + r$, avec $0 \leq r < 10$.
- **Étape 3 :** Ajouter le reste r à la fin de la liste : il est le chiffre de poids faible de l'entier courant.
- **Étape 4 :** Remplacer l'entier courant par le quotient q . Si ce quotient n'est pas nul, revenir à l'étape 2.
- **Étape 5 :** Inverser la liste des restes, puis la renvoyer. Les chiffres sont alors ordonnés du poids fort au poids faible.

Exemple 3 Liste des chiffres de 7307 par divisions successives

Division par dix	Quotient suivant	Reste obtenu
$7307 = 10 \times 730 + 7$	730	7
$730 = 10 \times 73 + 0$	73	0
$73 = 10 \times 7 + 3$	7	3
$7 = 10 \times 0 + 7$	0 : arrêt	7

Restes dans l'ordre d'obtention : [7, 0, 3, 7]. Après inversion : [7, 3, 0, 7].

On peut aussi utiliser la présentation en *potence* avec une cascade de divisions euclidiennes qui s'arrête lorsque le quotient est nul. La liste des chiffres est celle des restes lue du bas vers le haut.



$$(7307)_{10} = (7307)_{10}$$

Liste des chiffres : [7, 3, 0, 7]

Remarque 1

Syntaxe Python

- `n // 10` donne le quotient et `n % 10` le reste;
- `chiffres.append(x)` ajoute `x` à la fin de la liste `chiffres`;
- `chiffres.reverse()` inverse la liste sur place.
- Le cas $n = 0$ est traité séparément : sa liste de chiffres est [0].

 Exercice 3 *Algorithme des divisions successives en base dix*

1. Appliquer l'algorithme des divisions successives pour extraire les chiffres en base dix des entiers 442 et 605.

.....

.....

.....

.....

.....

.....

.....

2. Compléter la fonction Python qui renvoie la liste des chiffres d'un entier écrit en base dix avec l'algorithme des divisions successives. On donne un jeu de tests unitaires.

```
def nombre2chiffres_divisions(n: int) -> list[int]:  
    if n == 0:  
        return [0]  
    chiffres = []  
    while n > ... :  
        reste = ...  
        chiffres.append( ... )  
        n = ...  
    chiffres.reverse()  
    return chiffres
```

```
assert nombre2chiffres_divisions(442) == [4, 4, 2]  
assert nombre2chiffres_divisions(6050) == [6, 0, 5, 0]  
assert nombre2chiffres_divisions(0) == [0]
```

1.2 Représentation en base deux



Théorème-Définition 2

La mémoire d'un ordinateur ne pouvant stocker que des séquences de bits à deux états, pour représenter un entier naturel en machine, il faut l'écrire en base deux où les seuls chiffres disponibles sont 0 et 1.

Pour se convaincre que tout entier naturel peut s'écrire en base deux, on pourra visionner la vidéo sur cette page <http://video.math.cnrs.fr/magie-en-base-deux/>.

Tout entier naturel n peut s'écrire de façon unique en base 2 sous la forme $b_k b_{k-1} \dots b_1 b_0$ telle que :

$$n = \sum_{i=0}^k b_i \times 2^i \text{ avec } \forall i \in \llbracket 0; k \rrbracket, b_i \in \llbracket 0; 1 \rrbracket$$

b_k est le bit de poids fort et b_0 est le bit de poids faible. Pour distinguer l'écriture en base 2 de l'écriture en base 10 on pourra écrire $\overline{101}^2$ pour par exemple l'écriture de 5 en base 2.

1.2.1 De la liste des bits à l'entier : somme pondérée

Algorithme 4 De la liste des bits à l'entier : algorithme de somme pondérée

C'est le même algorithme que pour la base dix.

- ✎ **Étape 1 :** Noter L la longueur de la liste de bits et initialiser l'entier n à zéro.
- ✎ **Étape 2 :** Parcourir la liste de gauche à droite. Pour chaque bit d'indice i , calculer son poids 2^{L-1-i} .
- ✎ **Étape 3 :** Multiplier ce bit par son poids et ajouter le résultat à n . Répéter les étapes 2 et 3 jusqu'à la fin de la liste.
- ✎ **Étape 4 :** Renvoyer n , la somme des contributions.



Attention : l'indice dans la liste commence à zéro à gauche, alors que l'exposant commence à zéro à droite.



Exemple 4

Indice	0	1	2	3	4	5
Bit	1	0	0	1	0	1
Poids	32	16	8	4	2	1

La liste $[1, 0, 0, 1, 0, 1]$ est la décomposition en base deux de l'entier :

$$1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 37$$



Exercice 4

- Expliquer la formule suivante : « Il existe 10 catégories de personnes : celles qui comprennent le binaire et les autres. »
.....
- Compter jusqu'à 18 en binaire.
.....
.....
- Quel est le plus grand entier non signé qu'on peut représenter en base 2 sur 3 bits? sur n bits?

-
4. C'est en $\overline{11110010000}_2$ qu'Alan Turing a défini sa machine à calculer universelle. Et en base dix?
-
5. Compléter la fonction `bits2nombre(t: list[int]) -> int` qui renvoie l'entier naturel (en base dix) représenté par la liste de bits `t`. Les bits sont classés par poids décroissant de gauche à droite.

```
def bits2nombre(t: list[int]) -> int:
    n = 0
    for i in range(len(t)):
        expo = ...
        n = n + ...
    return n
```

```
assert bits2nombre([1, 1, 0, 1]) == 13
assert bits2nombre([1, 0, 0, 1, 0]) == 18
assert bits2nombre([0, 0, 1, 1, 0, 1]) == 13
```

1.2.2 De la liste des bits à l'entier : algorithme d'Horner

Algorithme 5 De la liste des bits à l'entier : algorithme d'Horner

- 👉 **Étape 1 :** Initialiser l'entier n à zéro.
- 👉 **Étape 2 :** Lire les bits de la liste de gauche à droite.
- 👉 **Étape 3 :** Pour chaque bit lu, multiplier n par 2, puis ajouter ce bit : $n \leftarrow 2n + \text{bit}$. Répéter les étapes 2 et 3 jusqu'à la fin de la liste.
- 👉 **Étape 4 :** Renvoyer n .

Multiplier par deux décale l'écriture binaire d'un rang vers la gauche.

Exemple 5

Bit lu	1	0	0	1	0	1
Calcul	$0 \times 2 + 1$	$1 \times 2 + 0$	$2 \times 2 + 0$	$4 \times 2 + 1$	$9 \times 2 + 0$	$18 \times 2 + 1$
Valeur de n	1	2	4	9	18	37

Les valeurs calculées dans le tableau sont écrites en base dix. À la fin, on obtient que la liste `[1, 0, 0, 1, 0, 1]` représente l'entier $n = 37$, comme avec la somme pondérée.

Exercice 5 Horner en base deux

Compléter la fonction `horner2(t: list[int]) -> int` qui renvoie l'entier naturel (en base dix) représenté par la liste de bits `t`. Les bits sont classés par poids décroissant de gauche à droite. On donne un jeu de tests unitaires.

```
def horner2(t: list[int]) -> int:
    n = ...
    for bit in ...:
        n = ...
    return n
```

```
assert horner2([1, 1, 0, 1]) == 13
```

```
assert horner2([1, 0, 0, 1, 0]) == 18
assert horner2([0]) == 0
```

1.2.3 De l'entier à la liste des bits : divisions successives

Algorithme 6 *De la liste des bits à l'entier : algorithme des divisions successives*

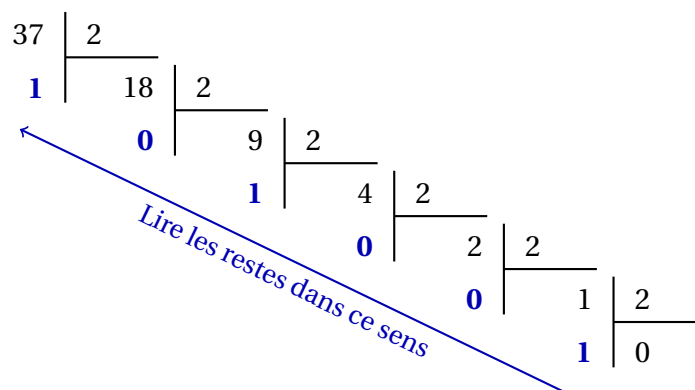
- ➡ **Étape 1 :** Si l'entier vaut zéro, renvoyer $[0]$ et arrêter l'algorithme. Sinon, créer une liste vide.
- ➡ **Étape 2 :** Effectuer la division euclidienne de l'entier courant par 2 : $n = 2q + r$, avec $0 \leq r < 2$.
- ➡ **Étape 3 :** Ajouter le reste r à la fin de la liste : il est le bit de poids faible de l'entier courant.
- ➡ **Étape 4 :** Remplacer l'entier courant par le quotient q . Si ce quotient n'est pas nul, revenir à l'étape 2.
- ➡ **Étape 5 :** Inverser la liste des restes, puis la renvoyer. Les bits sont alors ordonnés du poids fort au poids faible.

Exemple 6 *Convertir 37*

Division par deux	Quotient suivant	Reste obtenu
$37 = 2 \times 18 + 1$	18	1
$18 = 2 \times 9 + 0$	9	0
$9 = 2 \times 4 + 1$	4	1
$4 = 2 \times 2 + 0$	2	0
$2 = 2 \times 1 + 0$	1	0
$1 = 2 \times 0 + 1$	0 : arrêt	1

Restes obtenus : $[1, 0, 1, 0, 0, 1]$. Après inversion : $[1, 0, 0, 1, 0, 1]$, donc $37 = \overline{100101}^{(2)}$.

On peut aussi utiliser la présentation en *potence* avec une cascade de divisions euclidiennes qui s'arrête lorsque le quotient est nul. La liste des bits est celle des restes lue du bas vers le haut.



$$(37)_{10} = (100101)_2$$

Liste des chiffres : $[1, 0, 0, 1, 0, 1]$

Exercice 6 *Divisions successives en base deux*

1. Appliquer l'algorithme des divisions successives pour extraire les chiffres en base deux des entiers écrits 13 et 18 en base dix.

.....

.....

.....

.....

.....

.....

2. Compléter la fonction Python qui renvoie la liste des chiffres en base deux d'un entier écrit en base dix avec l'algorithme des divisions successives. On donne un jeu de tests unitaires.

```
def nombre2bits_divisions(n: int) -> list[int]:
    if n == 0:
        return [0]
    bits = []
    while n > ...:
        reste = ...
        bits.append( ... )
        n = ...
    bits.reverse()
    return bits
```

```
assert nombre2bits_divisions(13) == [1, 1, 0, 1]
assert nombre2bits_divisions(18) == [1, 0, 0, 1, 0]
assert nombre2bits_divisions(0) == [0]
```

1.2.4 De l'entier à la liste des bits : algorithme glouton

Algorithme 7 De la liste des bits à l'entier : algorithme glouton

- 👉 **Étape 1 :** Si l'entier vaut zéro, renvoyer [0] et arrêter l'algorithme. Sinon, créer une liste vide et initialiser le reste à représenter à n .
- 👉 **Étape 2 :** Chercher la plus grande puissance de deux $p \leq n$: c'est le poids du premier bit.
- 👉 **Étape 3 :** Si le reste est au moins égal à p , ajouter 1 à la liste et soustraire p du reste. Sinon, ajouter 0 à la liste et conserver le reste.
- 👉 **Étape 4 :** Diviser le poids p par deux. Tant que $p \geq 1$, revenir à l'étape 3; sinon, renvoyer la liste.

Exemple 7 Convertir 37

On détermine d'abord la plus grande puissance de 2 inférieure ou égale à 37 : c'est 32 car $32 \leq 37 < 64$. On parcourt ensuite toutes les puissances de 2 inférieures ou égales à 32 par poids décroissant.

Poids examiné	Choix et nouveau reste	Bit obtenu
32	$37 \geq 32 : 37 - 32 = 5$	1
16	$5 < 16 : \text{reste } 5$	0
8	$5 < 8 : \text{reste } 5$	0
4	$5 \geq 4 : 5 - 4 = 1$	1
2	$1 < 2 : \text{reste } 1$	0
1	$1 \geq 1 : 1 - 1 = 0$	1



Contrairement à l'algorithme des divisions successives on obtient les chiffres/bits dans l'ordre usuel : par poids décroissant.

Exercice 7 Algorithme glouton en base deux

- Appliquer l'algorithme glouton pour extraire les chiffres en base deux des entiers écrits 13 et 18 en base dix.

.....

.....

.....

.....

.....

- Compléter la fonction Python qui renvoie la liste des chiffres en base deux d'un entier écrit en base dix avec l'algorithme glouton. On donne un jeu de tests unitaires.

```
def nombre2bits_glouton(n: int) -> list[int]:
    if n == 0:
        return [0]
    p = 1
    while p * 2 <= n:
        p = ...
    bits = []
    reste = n
    while p >= 1:
        if reste >= ... :
            bits.append( ... )
            reste = ...
        else:
            bits.append( ... )
        p = ...
    return bits
```

```
assert nombre2bits_glouton(13) == [1, 1, 0, 1]
assert nombre2bits_glouton(18) == [1, 0, 0, 1, 0]
assert nombre2bits_glouton(0) == [0]
```

1.3 Représentation en base 16



Théorème-Définition 3

Plus généralement, on peut représenter un entier naturel dans n'importe quelle base.

Une base très utilisée en informatique est la **base 16** ou **hexadécimale**.

On étend les dix chiffres de la base 10 avec A, B, C, D, E et F pour représenter 10, 11, 12, 13, 14 et 15.

En base 16 chaque octet est représenté par deux chiffres ce qui permet de condenser l'écriture.

Par exemple, chaque carte réseau possède une adresse MAC codée sur 48 bits soit 6 octets et représentée par une séquence de 6 chiffres en base 16 séparés par le caractère : sous la forme : c8:60:00:a4:89:ab

De même, les couleurs en représentation (R,G,B) sont codées sur 3 octets et dans le langage HTML des pages Web on les rencontre souvent notées comme séquence de six chiffres en base 16 préfixés par le caractère # : par exemple #FF0000 va coder un rouge pur.



Exercice 8

1. Convertir en base dix les entiers représentés en base 16 par $\overline{1A}^{16}$, $\overline{AF}^{16}$ et $\overline{FF}^{16}$.

.....

.....

2. La représentation hexadécimale s'obtient facilement à partir de la représentation binaire. Puisque $2^4 = 16$, pour convertir un octet de binaire en hexadécimal, il suffit de regrouper les bits par 4.

Ainsi 154 représenté en binaire sur un octet par $\overbrace{1001}^{9^{16}} \overbrace{1010}^{A^{16}}$ a pour représentation hexadécimale $\overline{9A}^{16}$ car $9 \times 16 + 10 = 154$.

Convertir de même en hexadécimal les octets suivants :

• $\overline{11101010}^2$ | • $\overline{10011011}^2$ | • $\overline{11000000}^2$

.....

.....

Méthode

- En Python, pour convertir l'écriture d'un entier naturel de la base 16 vers la base 10, on préfixe la séquence de bits de 0x :

```
In [5]: 0x9A
Out[5]: 154
```

- Pour convertir l'écriture d'un entier naturel de la base 10 vers la base 16, on utilise la fonction hex :

```
In [6]: hex(154)
Out[6]: '0x9a'
```

2 Représentation interne des entiers relatifs

Contenus inspirés par le cours de mon collègue Pierre Duclosson.

Dans cette partie, on présente deux représentations internes des entiers relatifs en binaire.

2.1 Par signe et valeur absolue

Méthode *Signe et valeur absolue*

Une première idée consiste à réserver le premier bit (celui le plus à gauche, dit "bit de poids fort") au signe de l'entier (par exemple 0 pour positif et 1 pour négatif) et les $n - 1$ autres bits à la valeur absolue. Sur 8 bits, on peut ainsi représenter les entiers entre -127 et 127 :

signe	64	32	16	8	4	2	1	signe	64	32	16	8	4	2	1
0	0	1	1	0	1	1	1	1	0	0	1	0	1	0	1
55								-21							

Exercice 9

1. Le nombre 0 possède-t-il une unique représentation par signe et valeur absolue?

.....

2. Effectuer l'addition binaire ci-dessous en appliquant l'algorithme usuel (bit par bit de la droite vers la gauche avec propagation de retenue).

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} \rightarrow 55 \\
 + \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline \end{array} \rightarrow -21 \\
 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline & & & & & & & \\ \hline \end{array} \rightarrow
 \end{array}$$

À quel problème est-on confronté?



.....

.....

.....

.....

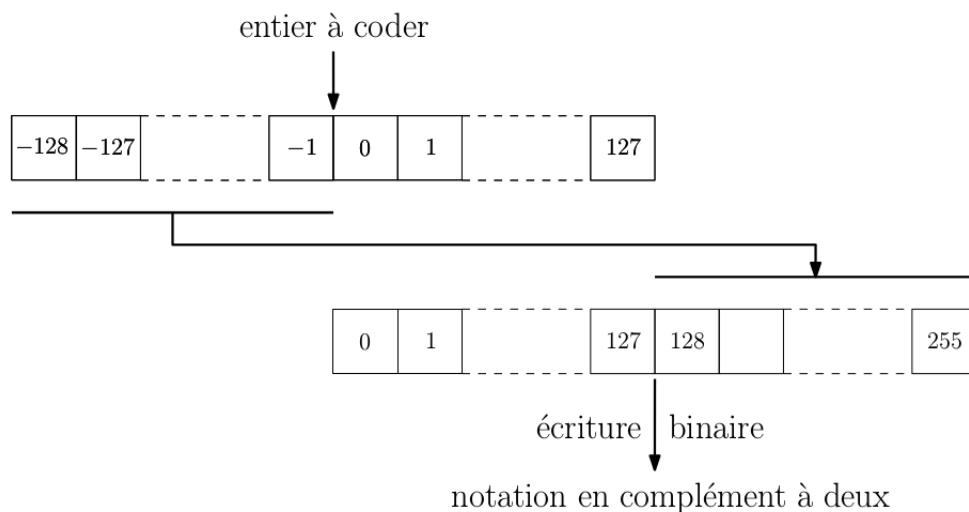
2.2 Par complément à 2

Méthode Complément à 2

Commençons par regarder le cas particulier des entiers codés sur un octet.

La **représentation en complément à 2** consiste à représenter les entiers entre -128 et 127 (au lieu de 0 à 255) de la manière suivante :

- si $x \in \llbracket 0; 127 \rrbracket$ alors x est codé par son écriture binaire sur un octet.
- si $x \in \llbracket -128; -1 \rrbracket$ alors x est codé par l'écriture binaire de $x + 256$ sur un octet.



Ainsi, on fait correspondre à chaque entier de $\llbracket -128; 127 \rrbracket$ un entier de $\llbracket 0; 255 \rrbracket$ qui est ensuite codé sur 8 bits par sa représentation binaire. On remarquera que 0 admet alors une seule écriture : $\overline{00000000}^2$.

Par exemple la représentation en complément à deux sur 8 bits de -43 est $\overline{11010101}^2$ et celle de 117 est $\overline{01110101}^2$.

On remarque que le signe d'un entier est immédiat à déterminer via sa représentation en complément à deux : il est donné par le bit de poids fort. Le nombre est positif ou nul lorsque le bit de poids fort vaut 0 et négatif sinon.

Nous pouvons maintenant généraliser à la représentation en complément à deux sur n bits. On représente les entiers entre -2^{n-1} et $2^{n-1} - 1$ de la manière suivante :

- si $x \in \llbracket 0; 2^{n-1} - 1 \rrbracket$ alors x est codé par son écriture binaire sur n bits.
- si $x \in \llbracket -2^{n-1}; -1 \rrbracket$ alors x est codé par l'écriture binaire sur n bits de $x + 2^n$.

La figure ci-dessous illustre la représentation des entiers relatifs en complément à 2 sur 3 bits

	décimal	
	0	
-1	binaire	1
	000	
	111	001
-2	110	010
	101	011
-3	100	
		3
	-4	



Exercice 10

On souhaite représenter des entiers relatifs sur quatre bits en complément à 2.

- Donner les représentations en complément à 2 sur quatre bits des entiers 1 et -1.

.....

.....

- Vérifier si l'algorithme classique d'addition binaire appliqué à ces représentations donne bien 0 pour l'addition de 1 et de -1 et -2 pour l'addition de -1 et -1 (en excluant la dernière retenue sortante).

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....



Ainsi il n'est pas nécessaire d'avoir deux algorithmes distincts pour l'addition et la soustrac-

tion. Par rapport à la représentation par signe et valeur absolue, cela se traduit dans l'implémentation matérielle de l'addition par une réduction du nombre de portes logiques et donc une économie de transistor et une simplification des circuits.

3. Que donne l'addition des représentations de 7 et 1 sur 4 bits en complément à deux? Commenter

.....

.....

.....

.....

.....

.....

4. Pour obtenir la représentation en complément à 2 sur n bits de l'opposé d'un entier naturel inférieur ou égal à $2^{n-1} - 1$, il suffit de prendre le complément de chaque bit de sa représentation binaire ($0 \rightarrow 1$ et $1 \rightarrow 0$) et d'ajouter 1. Justifier cet algorithme à partir de la définition de la représentation en complément à 2.

.....

.....

.....

.....

.....

.....

3 QCM de synthèse

Exercice 11

Cet exercice est un questionnaire à choix multiples. Chaque question comporte une seule bonne réponse parmi les quatre proposées. Cocher la réponse sélectionnée.

1. On souhaite pouvoir représenter en base deux tous les entiers non signés de 0 à 255 inclus. Quel est le nombre minimal de chiffres binaires nécessaires?
☐ Réponse 1 : 1 ☐ Réponse 2 : 2 ☐ Réponse 3 : 4 ☐ Réponse 4 : 8
2. En complément à 2 sur 4 bits, quelle est la représentation binaire de -6 ?
☐ Réponse 1 : 1110
☐ Réponse 2 : On ne peut pas représenter -6 sur 4 bits en complément à 2
☐ Réponse 3 : 0110
☐ Réponse 4 : 1010
3. On considère un entier codé en complément à 2 sur 4 bits : 1110. Quelle est sa valeur en base dix?
☐ Réponse 1 : -1 ☐ Réponse 2 : 14
☐ Réponse 3 : -2 ☐ Réponse 4 : -14
4. Quelle est la plage des entiers représentables en complément à 2 sur n bits?
☐ Réponse 1 : De -2^{n-1} à 2^{n-1}
☐ Réponse 2 : De $-(2^{n-1} - 1)$ à $2^{n-1} - 1$
☐ Réponse 3 : De -2^n à $2^n - 1$
☐ Réponse 4 : De -2^{n-1} à $2^{n-1} - 1$
5. Quel est l'écriture en base 16 correspondant à l'entier non signé écrit en binaire sur 8 bits : 10110101?
☐ Réponse 1 : AB11 ☐ Réponse 2 : B5
☐ Réponse 3 : C5 ☐ Réponse 4 : D5
6. Sur combien d'octets (1 octet est une séquence de 8 bits), peut-on représenter un nombre de 24 chiffres en base 16?
☐ Réponse 1 : 3 ☐ Réponse 2 : 12
☐ Réponse 3 : 48 ☐ Réponse 4 : 4×24