

# T.D. 1 – Corrigé

## Systèmes de numération entière

### Exercice 1

Représentez les nombres  $28_{10}$ ,  $129_{10}$ ,  $147_{10}$ ,  $255_{10}$  sous leur forme binaire par une autre méthode que les divisions successives. À partir de cette représentation binaire, vous en déduirez leur représentation hexadécimale.

À partir de la valeur des différents poids binaires, et en commençant par le poids le plus fort, on positionne les bits à 0 ou à 1 en fonction de la somme de leur poids.

|            |   | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|------------|---|-----|----|----|----|---|---|---|---|
| $28_{10}$  | → | 0   | 0  | 0  | 1  | 1 | 1 | 0 | 0 |
| $129_{10}$ | → | 1   | 0  | 0  | 0  | 0 | 0 | 0 | 1 |
| $147_{10}$ | → | 1   | 0  | 0  | 1  | 0 | 0 | 1 | 1 |
| $255_{10}$ | → | 1   | 1  | 1  | 1  | 1 | 1 | 1 | 1 |

Le passage d'une représentation binaire (base 2) vers une représentation hexadécimale (base 16) s'obtient assez facilement en regroupant les bits par paquets de quatre ( $2^4 = 16$ ) ; chaque paquet de quatre bits correspond à un chiffre hexadécimal.

$$28_{10} = 0001\ 1100_2 = \mathbf{1C}_{16}$$

$$129_{10} = 1000\ 0001_2 = \mathbf{81}_{16}$$

$$147_{10} = 1001\ 0011_2 = \mathbf{93}_{16}$$

$$255_{10} = 1111\ 1111_2 = \mathbf{FF}_{16}$$

### Exercice 2

1. Les nombres  $11000010_2$ ,  $10010100_2$ ,  $11101111_2$ ,  $10000011_2$ ,  $10101000_2$  sont-ils pairs ou impairs ?

Les nombres pairs se terminent par au moins un zéro :  **$11000010_2$ ,  $10010100_2$ ,  $10101000_2$**

2. Lesquels sont divisibles par 4, 8 ou 16 ?

- Les nombres divisibles par 4 se terminent par au moins deux zéros :  **$10010100_2$ ,  $10101000_2$**
- Les nombres divisibles par 8 se terminent par au moins trois zéros :  **$10101000_2$**
- Les nombres divisibles par 16 se terminent par au moins quatre zéros : **Aucun nombre.**

3. Donnez le quotient et le reste d'une division entière par 2, 4 et 8 de ces nombres.

|    | 11000010 |       | 10010100 |       | 11101111 |       | 10000011 |       | 10101000 |       |
|----|----------|-------|----------|-------|----------|-------|----------|-------|----------|-------|
|    | quotient | reste | quotient | reste | quotient | reste | quotient | reste | quotient | reste |
| /2 | 1100001  | 0     | 1001010  | 0     | 1110111  | 1     | 1000001  | 1     | 1010100  | 0     |
| /4 | 110000   | 10    | 100101   | 00    | 111011   | 11    | 100000   | 11    | 101010   | 00    |
| /8 | 11000    | 010   | 10010    | 100   | 11101    | 111   | 10000    | 011   | 10101    | 000   |

4. En généralisant, que suffit-il de faire pour obtenir le quotient et le reste d'une division entière d'un nombre binaire par  $2^n$  ?

- Pour le quotient : il faut réaliser un **décalage de  $n$  bits vers la droite** du nombre.
- Pour le reste : il faut réaliser un **ET logique de  $2^n - 1$**  avec le nombre.

Les décalages et les opérations logiques sont nettement plus rapides à réaliser pour un microprocesseur que l'opération de division.

5. Si l'on souhaite multiplier un nombre binaire quelconque par une puissance de 2, quelle méthode peut-on utiliser afin d'éviter la multiplication ?

Un décalage logique d'un seul bit vers la gauche est équivalent à une multiplication par 2. Ainsi, un décalage logique de  $n$  bits vers la gauche est équivalent à une multiplication par  $2^n$ .

6. Si l'on souhaite multiplier un nombre binaire quelconque par 3 ou par 10, quelle méthode peut-on utiliser pour éviter la multiplication ?

- $3n = 2n + n$

Sous cette forme, il apparaît une multiplication par 2 (équivalente à un décalage d'un bit vers la gauche) et une addition.

- $10n = 8n + 2n$

Sous cette forme, il apparaît une multiplication par 8 (équivalente à un décalage de 3 bits vers la gauche), une multiplication par 2 (équivalente à un décalage d'un bit vers la gauche), et une addition.

Si le multiplicateur est connu, on peut le décomposer de sorte à n'avoir comme opérations que des décalages et des additions. Ces dernières sont beaucoup plus rapides à réaliser pour un microprocesseur que la multiplication.

**Exercice 3**

Donnez les valeurs décimales, minimales et maximales, que peuvent prendre des nombres signés et non signés codés sur 4, 8, 16, 32 et  $n$  bits.

| Bits | Non Signés                   | Signés                             |
|------|------------------------------|------------------------------------|
| 4    | 0 $\rightarrow$ 15           | -8 $\rightarrow$ 7                 |
| 8    | 0 $\rightarrow$ 255          | -128 $\rightarrow$ 127             |
| 16   | 0 $\rightarrow$ 65535        | -32768 $\rightarrow$ 32767         |
| 32   | 0 $\rightarrow$ $2^{32} - 1$ | $-2^{31} \rightarrow 2^{31} - 1$   |
| n    | 0 $\rightarrow$ $2^n - 1$    | $-2^{n-1} \rightarrow 2^{n-1} - 1$ |

**Exercice 4**

Soit les deux nombres binaires suivants :  $11111111_2$  et  $10110110_2$ .

1. Donnez leur représentation décimale s'ils sont codés sur 8 bits signés.

- **$11111111_2$**

Sur 8 bits signés, le bit de poids fort vaut 1 : le nombre est négatif.

On effectue son complément à 2 puis on convertit le résultat en décimal :

$$(11111111_2)_{C2} = 00000000_2 + 1_2 = 1_2 = 1$$

**La représentation décimale est donc de -1.**

- **$10110110_2$**

Sur 8 bits signés, le bit de poids fort vaut 1 : le nombre est négatif.

On effectue son complément à 2 puis on convertit le résultat en décimal :

$$(10110110_2)_{C2} = 01001001_2 + 1_2 = 01001010_2 = 64 + 8 + 2 = 74$$

**La représentation décimale est donc de -74.**

2. Donnez leur représentation décimale s'ils sont codés sur 16 bits signés.

- **$11111111_2$**

Sur 16 bits signés, le bit de poids fort vaut 0 ( $0000000011111111_2$ ) : le nombre est positif.

On effectue une simple conversion binaire-décimal :

$$11111111_2 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

**La représentation décimale est donc de +255.**

- **$10110110_2$**

Sur 16 bits signés, le bit de poids fort vaut 0 ( $0000000010110110_2$ ) : le nombre est positif.

On effectue une simple conversion binaire-décimal :

$$10110110_2 = 128 + 32 + 16 + 4 + 2 = 182$$

**La représentation décimale est donc de +182.**

Soit le nombre entier négatif suivant :  $-80_{10}$ .

3. On souhaite le coder sur 8 bits signés. Donnez sa représentation binaire et sa représentation hexadécimale.

On convertit sa valeur absolue en binaire :  $80_{10} = 01010000_2$

On effectue son complément à 2 :  $(01010000_2)_{C2} = 10101111_2 + 1_2 = 10110000_2$

Ce qui donne : **10110000<sub>2</sub> en binaire.**

**B0<sub>16</sub> en hexadécimale.**

4. On souhaite le coder sur 16 bits signés. Donnez sa représentation binaire et sa représentation hexadécimale.

Une simple extension de signe suffit pour passer de 8 bits à 16 bits signés.

Ce qui donne : **111111110110000<sub>2</sub> en binaire.**

**FFB0<sub>16</sub> en hexadécimale.**

## **Exercice 5**

1. Donnez, en puissance de deux, le nombre de bits que contiennent les grandeurs suivantes : 128 Kib, 16 Mib, 2 Kio, 512 Gio.

On sait que :

- $1 \text{ Ki} = 2^{10}$  ;  $1 \text{ Mi} = 2^{20}$  ;  $1 \text{ Gi} = 2^{30}$ .
- $1 \text{ octet} = 8 \text{ bits} = 2^3 \text{ bits}$ .

On a donc :

- **128 Kib** =  $2^7 \times 2^{10} \text{ bits} = 2^{17} \text{ bits}$ .
- **16 Mib** =  $2^4 \times 2^{20} \text{ bits} = 2^{24} \text{ bits}$ .
- **2 Kio** =  $2^1 \times 2^{10} \text{ octets} = 2^1 \times 2^{10} \times 2^3 \text{ bits} = 2^{14} \text{ bits}$ .
- **512 Gio** =  $2^9 \times 2^{30} \text{ octets} = 2^9 \times 2^{30} \times 2^3 \text{ bits} = 2^{42} \text{ bits}$ .

2. Donnez, à l'aide des préfixes binaires (Ki, Mi ou Gi), le nombre d'octets que contiennent les grandeurs suivantes : 2 Mib,  $2^{14} \text{ bits}$ ,  $2^{26} \text{ octets}$ ,  $2^{32} \text{ octets}$ . Vous choisirez un préfixe qui permet d'obtenir la plus petite valeur numérique entière.

- **2 Mib** =  $2^1 \times 2^{20} \text{ bits} = 2^1 \times 2^{20} / 2^3 \text{ octets} = 2^{18} \text{ octets} = 2^8 \times 2^{10} \text{ octets} = 256 \text{ Kio}$ .
- **$2^{14} \text{ bits}$**  =  $2^{14} / 2^3 \text{ octets} = 2^{11} \text{ octets} = 2^1 \times 2^{10} \text{ octets} = 2 \text{ Kio}$ .
- **$2^{26} \text{ octets}$**  =  $2^6 \times 2^{20} \text{ octets} = 64 \text{ Mio}$ .
- **$2^{32} \text{ octets}$**  =  $2^2 \times 2^{30} \text{ octets} = 4 \text{ Gio}$ .