Série 6 : Solutions

1 Opérations binaires

(a) Additions (on effectue les retenues de droite à gauche) :

1. $110 + 101 = 1011$ $\quad(6 + 5 = 11\ \checkmark)$
2. $1001 + 0111 = 10000$ $\quad(9 + 7 = 16\ \checkmark)$
3. $10110 + 01101 = 100011$ $\quad(22 + 13 = 35\ \checkmark)$
4. $11011 + 10101 = 110000$ $\quad(27 + 21 = 48\ \checkmark)$
5. $111111 + 000001 = 1000000$ $\quad(63 + 1 = 64\ \checkmark)$

(b) Soustractions (on effectue les emprunts de droite à gauche) :

1. $1111 - 0010 = 1101$ $\quad(15 - 2 = 13\ \checkmark)$
2. $1010 - 0011 = 0111$ $\quad(10 - 3 = 7\ \checkmark)$
3. $10000 - 00001 = 01111$ $\quad(16 - 1 = 15\ \checkmark)$
4. $11101 - 10110 = 00111$ $\quad(29 - 22 = 7\ \checkmark)$
5. $11111 - 01010 = 10101$ $\quad(31 - 10 = 21\ \checkmark)$

2 Conversion : Nombres naturels – Entiers non signés

Binaire → décimal (on somme les puissances de 2 correspondant aux bits à 1) :

1. $(11010110)_2 = 128 + 64 + 16 + 4 + 2 = 214$
2. $(10001001)_2 = 128 + 8 + 1 = 137$
3. $(101010101)_2 = 256 + 64 + 16 + 4 + 1 = 341$
4. $(01000001)_2 = 64 + 1 = 65$
5. $(10111011)_2 = 128 + 32 + 16 + 8 + 2 + 1 = 187$

Décimal → binaire (on divise successivement par 2 et on lit les restes de bas en haut) :

1. $(77)_{10} = (01001101)_2$ $\quad$car $64 + 8 + 4 + 1 = 77$
2. $(200)_{10} = (11001000)_2$ $\quad$car $128 + 64 + 8 = 200$
3. $(333)_{10} = (101001101)_2$ $\quad$car $256 + 64 + 8 + 4 + 1 = 333$
4. $(1000)_{10} = (1111101000)_2$ $\quad$car $512 + 256 + 128 + 64 + 32 + 8 = 1000$
5. $(2026)_{10} = (11111101010)_2$ $\quad$car $1024 + 512 + 256 + 128 + 64 + 32 + 8 + 2 = 2026$

$(001011)_2 = 11_{10}$. $(101100)_2 = 44_{10} = 4 \times 11$. On a simplement décalé le motif binaire de deux positions vers la gauche, ce qui correspond à une **multiplication par** $2^2 = 4$. De manière générale, un décalage de k bits vers la gauche multiplie la valeur par 2^k, et vers la droite la divise par 2^k (division entière).

3 Dépassement et capacité

(a) Interprétation en complément à deux sur 8 bits (le bit de poids fort vaut $-2^7 = -128$) :

- 11001010 : MSB= 1, donc négatif. Complément à deux : $\overline{11001010} + 1 = 00110101 + 1 = 00110110 = 54$. Valeur : $\mathbf{-54}$.
- 01111110 : MSB= 0, donc positif. Valeur : $64 + 32 + 16 + 8 + 4 + 2 = \mathbf{126}$.
- 10000001 : MSB= 1, négatif. $\overline{10000001} + 1 = 01111110 + 1 = 01111111 = 127$. Valeur : $\mathbf{-127}$.

(b) Encodage en complément à deux sur 8 bits (on part du positif, on inverse les bits, on ajoute 1) :

- $-3 : 3 = 00000011 \xrightarrow{\text{inv.}} 11111100 \xrightarrow{+1} \mathbf{11111101}$
- $-64 : 64 = 01000000 \xrightarrow{\text{inv.}} 10111111 \xrightarrow{+1} \mathbf{11000000}$
- -128 : cas particulier : **10000000** (la valeur +128 dépasse la capacité des 8 bits signés ; la procédure donne 01111111 + 1 = 10000000, ce qui est bien le résultat attendu).

(c) Sur 8 bits non signés : 0 à $255 = 2^8 - 1$. Sur 8 bits signés (complément à deux) : -128 à $+127$, soit 2^8 valeurs mais avec une **asymétrie** : il existe une valeur négative de plus (-128) que de valeurs positives ($+127$). Ceci parce que 0 occupe une des 2^8 combinaisons.

(d) $80_{10} = 01010000_2$.

$$01010000 + 01010000 = 10100000$$

Interprété en complément à deux : $10100000_2 = -128 + 32 = -96_{10}$.

Le résultat est négatif alors que les deux opérandes sont positifs : c'est un **dépassement de capacité** (*overflow*). Il se produit ici parce que $160 > 127$, la valeur maximale représentable sur 8 bits signés. On peut détecter l'overflow lorsque deux opérandes de même signe donnent un résultat de signe opposé.

(e) $x = 90_{10} = 01011010_2$, $\quad y = 45_{10} = 00101101_2$.

$$\begin{array}{rr} & 01011010 \\ \oplus & 00101101 \\ \hline & 01110111 \end{array}$$

$(01110111)_2 = 64 + 32 + 16 + 4 + 2 + 1 = \mathbf{119}_{10}$.

On remarque que $x \oplus y = 01110111$ contient exactement les bits où x et y diffèrent. Ici x et y ont des bits parfaitement complémentaires (chaque bit de x est l'opposé du bit correspondant de y), donc le XOR donne tous les bits à 1, sauf le bit de poids fort : résultat $01111111 - 00001000 = 01110111 = 119$. (Note : on peut vérifier : $90 + 45 = 135 = 10000111_2$; le XOR est différent de l'addition car il ignore les retenues.)

4 Représentation des fractions en binaire

(a)

$$\begin{aligned} \tfrac{1}{2} &= 2^{-1} && \Rightarrow \quad 0.1_2 \\ \tfrac{3}{4} &= 2^{-1} + 2^{-2} && \Rightarrow \quad 0.11_2 \\ \tfrac{5}{8} &= 2^{-1} + 2^{-3} && \Rightarrow \quad 0.101_2 \\ \tfrac{7}{16} &= 2^{-2} + 2^{-3} + 2^{-4} && \Rightarrow \quad 0.0111_2 \end{aligned}$$

Seules les fractions dont le dénominateur est une puissance de 2 admettent une représentation binaire finie.

(b) Méthode des multiplications successives par 2 (on note la partie entière à chaque étape) :

$0.1 \times 2 = 0.2$	bit :	0	
$0.2 \times 2 = 0.4$	bit :	0	
$0.4 \times 2 = 0.8$	bit :	0	
$0.8 \times 2 = 1.6$	bit :	1	(reste 0.6)
$0.6 \times 2 = 1.2$	bit :	1	(reste 0.2)
$0.2 \times 2 = 0.4$	bit : 0	← même reste qu'à l'étape 2 : la séquence se répète !	

On obtient $0.1 = 0.0\overline{0011}_2$ (la séquence 0011 se répète indéfiniment). Il est donc **impossible de représenter** 0.1 **exactement** avec un nombre fini de bits : on ne peut stocker qu'une approximation en mémoire.

(c) En Python :

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 == 0.3
False
```

Ni 0.1 ni 0.2 ne sont représentés exactement : leurs approximations binaires cumulent de petites erreurs d'arrondi lors de l'addition. Le résultat diffère légèrement de la valeur exacte 0.3 (elle-même approchée). Il ne faut donc **jamais tester l'égalité exacte** entre deux flottants ; on utilisera plutôt `abs(a - b) < epsilon` pour un seuil ε adapté.

5 Représentation des nombres à virgule flottante

Rappel du modèle : 6 bits, **2 bits exposant** (valeurs 0–3), **4 bits mantisse** (valeurs 0–15). Valeur = mantisse $\times\, 2^{\text{exposant}}$.

(a)

— **Maximum** : mantisse = 15, exposant = 3 $\Rightarrow$ $15 \times 2^3 = 15 \times 8 = \mathbf{120}$.
— **Minimum non nul** : mantisse = 1, exposant = 0 $\Rightarrow$ $1 \times 2^0 = \mathbf{1}$.

(b) Pour chaque exposant, la mantisse varie de 0 à 15, donc les valeurs sont espacées de 2^{exposant}. On s'intéresse aux *nouvelles* valeurs introduites au-delà de la plage couverte par l'exposant précédent :

Exposant	Valeurs représentables	Plage *nouvelle*	Écart (erreur abs. max.)
0	$0, 1, 2, \ldots, 15$	$[0, 15]$	≤ 0 (entiers exacts)
1	$0, 2, 4, \ldots, 30$	$(15, 30]$	≤ 1
2	$0, 4, 8, \ldots, 60$	$(30, 60]$	≤ 2
3	$0, 8, 16, \ldots, 120$	$(60, 120]$	≤ 4

(c) L'erreur absolue **n'est pas constante** : elle double à chaque fois que l'exposant augmente d'une unité (de 0 à 1, puis à 2, puis à 4).

En revanche, l'**erreur relative** est approximativement constante. Dans chaque plage nouvelle, la valeur minimale est 2^e (mantisse = 1, exposant = e) et l'erreur absolue maximale est 2^{e-1} (la moitié de l'espacement 2^e). L'erreur relative maximale vaut donc :

$$\frac{2^{e-1}}{2^e} = \frac{1}{2} \quad \text{... mais la mantisse apporte un facteur supplémentaire.}$$

Plus précisément, dans la plage $(15, 30]$: erreur ≤ 1, valeur minimale = 16, erreur relative $\leq \frac{1}{16} \approx 6.25\%$. Dans $(30, 60]$: erreur ≤ 2, min = 32, erreur relative $\leq \frac{2}{32} = 6.25\%$. Dans $(60, 120]$: erreur ≤ 4, min = 64, erreur relative $\leq \frac{4}{64} = 6.25\%$.

La propriété fondamentale de la virgule flottante est que **l'erreur relative est approximativement constante**, quel que soit l'ordre de grandeur du nombre représenté : ici $\sim 1/16$, soit une mantisse de 4 bits.

(d*) Dans notre modèle simple, un nombre comme 8 peut être représenté de plusieurs façons (8×2^0, 4×2^1, 2×2^2, 1×2^3), ce qui est redondant. La normalisation IEEE 754 impose que la mantisse vérifie $1 \leq \text{mantisse} < 2$ (écrit $1.f$ avec f les bits fractionnaires). Le "1" avant la virgule n'a pas besoin d'être stocké explicitement (*bit caché*) : on gagne ainsi 1 bit de précision gratuit. Par exemple, avec 4 bits de mantisse normalisée (en plus du bit caché), on obtient en réalité 5 bits de précision, soit une erreur relative $\leq 1/32 \approx 3.1\%$ au lieu de 6.25%.

6 Questions d'examens passés – Solutions

Question 6. Réponses correctes : 40 (en décimal) et 101000 (motif binaire).

$001010_2 = 8 + 2 = 10_{10}$. Multiplier par 4 revient à décaler le motif de **deux positions vers la gauche** : $001010 \rightarrow 101000_2 = 40_{10}$. Les deux réponses sont équivalentes et correctes. 000101 correspond à un décalage vers la *droite* ($\div 4$), et 50 est incorrect.

Question 7.

Les affirmations correctes sont les affirmations 1 et 2.

Format : $(-1)^s \times 2^e \times \left(1 + \frac{m}{4}\right)$, avec $s \in \{0, 1\}$, $e \in \{0, \ldots, 15\}$ (exposant non signé sur 4 bits), $m \in \{0, 1, 2, 3\}$ (mantisse sur 2 bits). Le facteur $\left(1 + \frac{m}{4}\right)$ prend les valeurs 1,00, 1,25, 1,50, 1,75. Dans chaque binade $[2^e, 2^{e+1})$, les quatre valeurs représentables sont espacées de $\frac{2^e}{4} = 2^{e-2}$.

— **Aff. 1 – VRAI.** Dans $[64, 128)$, on a $2^e = 64$, soit $e = 6$. Les quatre valeurs représentables sont :

$$64 \times \{1;\ 1{,}25;\ 1{,}50;\ 1{,}75\} = \{64,\ 80,\ 96,\ 112\}.$$

L'espacement entre valeurs consécutives est $\frac{64}{4} = 16$. Pour tout $x \in [64, 128)$, l'erreur absolue (arrondi au plus proche) est au plus la moitié de cet espacement :

$$|\Delta x| \leq \frac{16}{2} = 8 < 16.$$

L'affirmation est **vraie**.

— **Aff. 2 – VRAI.** Dans la binade $[2^e, 2^{e+1})$, l'espacement entre valeurs consécutives est 2^{e-2}. L'erreur absolue maximale (arrondi au plus proche) est $\frac{2^{e-2}}{2} = 2^{e-3}$. La valeur minimale dans cette binade est 2^e, donc :

$$\text{erreur relative} \leq \frac{2^{e-3}}{2^e} = 2^{-3} = \frac{1}{8} \leq \frac{1}{4} = 2^{-2}.$$

(La borne 2^{-2} est en fait non atteinte ; la borne serrée est 2^{-3}.) L'affirmation est **vraie**.

— **Aff. 3 – FAUX.** Les valeurs représentables dans $[32, 64)$ (binade $e = 5$) sont :

$$32 \times \{1;\ 1{,}25;\ 1{,}50;\ 1{,}75\} \ = \ \{32,\ 40,\ 48,\ 56\}.$$

Remarquons d'abord que 60 n'est pas représentable. Le nombre 60,5 est compris entre 56 (à distance 4,5) et 64 (à distance 3,5) ; le plus proche est **64**. L'affirmation est **fausse**.

— **Aff. 4 – FAUX.** L'exposant est non signé ($e \geq 0$), donc la valeur absolue minimale représentable est $2^0 \times 1 = 1$. Or $0{,}75 < 1$: ce nombre n'est pas représentable dans ce format. (Contrairement à un format avec exposant signé, on ne peut pas ici encoder d'exposant négatif.) L'affirmation est **fausse**.

Question 8. Réponse correcte : -86.

$01011010_2 = 90$, $01010000_2 = 80$.

$$\begin{array}{r} 01011010 \\ +\ 01010000 \\ \hline 10101010 \end{array}$$

Le résultat est 10101010_2. Interprété en complément à deux : MSB = 1, donc négatif. On inverse et ajoute 1 : $\overline{10101010}+1 = 01010101 + 1 = 01010110_2 = 86$. Valeur = $\mathbf{-86}$.

Il y a bien un **dépassement de capacité** : $90 + 80 = 170 > 127$, limite des 8 bits signés. Les deux opérandes sont positifs (MSB = 0) mais le résultat est négatif (MSB = 1) : signe de l'overflow. La valeur 170 ne peut pas être représentée en complément à deux sur 8 bits.

Question 9. Réponses correctes : 01001011 et 10110101.

$75_{10} = 64 + 8 + 2 + 1 = 01001011_2 \ \Rightarrow |x| = 75$ si $x = +75$: motif **01001011**.

-75 en complément à deux : $75 = 01001011 \xrightarrow{\text{inv.}} 10110100 \xrightarrow{+1} \mathbf{10110101} \ \Rightarrow |x| = 75$ si $x = -75$: motif **10110101**.

$00101101_2 = 32 + 8 + 4 + 1 = 45 \neq 75$. 10110100_2 : inverse+1 $= 01001100 = 76 \neq 75$.

Question 10. FAUX.

Avec 8 bits en complément à deux, on peut représenter les entiers de $\mathbf{-128}$ à $\mathbf{+127}$ (soit $2^8 = 256$ valeurs distinctes). L'énoncé a inversé les extrémités : -127 à $+128$ est incorrect. L'asymétrie vient du fait que 0 occupe une combinaison ; il existe donc un négatif de plus (-128) que de positifs ($+127$).