# Corrigé du contrôle d'informatique

## Partie I. Représentation binaire des nombres entiers

**Question 1.** 13 est représenté par la suite de bits `01101` et 25 par `11001` ; on en déduit que :

`13 & 25` est représenté par `01001`, soit `13 & 25 = 9` ;

`13 | 25` est représenté par `11101`, soit `13 | 25 = 29` ;

`13 ^ 25` est représenté par `10100`, soit `13 ^ 25 = 20`.

15 est représenté par la suite de bits `00...01111` et −7 par `11...11001` on en déduit que :

`15 & −7` est représenté par `00...01001`, soit `15 & −7 = 9` ;

`15 | −7` est représenté par `11...11111`, soit `15 | −7 = −1` ;

`15 ^ −7` est représenté par `11...10110`, soit `15 ^ −7 = −10`.

**Question 2.** D'après le cours, la représentation binaire de $-x$ est obtenue en appliquant l'opérateur ~ à la représentation binaire de $x$ puis en lui rajoutant 1. Autrement dit, `~x + 1 = −x`. Ainsi, `x & −x` est équivalent à `x & (~x + 1)`.

Notons $k$ le rang du bit égal à 1 le plus à droite : $x$ est représenté par $x_{n-1}\cdots x_{k+1}10\cdots 0$, $\sim x$ par $\overline{x_{n-1}}\cdots\overline{x_{k+1}}01\cdots 1$ et $-x$ par $\overline{x_{n-1}}\cdots\overline{x_{k+1}}10\cdots 0$. On en déduit que `x & −x` calcule l'entier représenté par $0\cdots 010\cdots 0$ à savoir $2^k$, la plus grande puissance de 2 qui divise $x$.

**Question 3.** Notons $x$ et $y$ les entiers contenus initialement dans les variables `a` et `b`.
À la fin de la première instruction, la variable `a` contient l'entier $u$ défini par $u_i = x_i \wedge y_i$ pour $0 \leqslant i \leqslant n-1$.
À le fin de la deuxième instruction, `b` contient l'entier $v$ défini par $v_i = u_i \wedge y_i = (x_i \wedge y_i) \wedge y_i = x_i \wedge (y_i \wedge y_i) = x_i \wedge 0 = x_i$, autrement dit l'entier $x$.
À la fin de la troisième instruction, `a` contient l'entier $w$ défini par $w_i = u_i \wedge v_i = (x_i \wedge y_i) \wedge x_i = (x_i \wedge x_i) \wedge y_i = 0 \wedge y_i = y_i$, autrement dit l'entier $y$.
Le script réalise donc la permutation du contenu des variables `a` et `b`.

**Question 4.**

*a)* Notons $x_{n-1}\cdots x_1x_0$ la représentation binaire de $x$ ; alors `x << 1` calcule le nombre $y$ représenté par $x_{n-2}\cdots x_1x_00$.

Lorsque $0 \leqslant x < 2^{n-2}$ on a $x_{n-1} = x_{n-2} = 0$ donc $y$ est un nombre positif et $y = 2x$.

Lorsque $-2^{n-2} \leqslant x < 0$ on a $2^{n-1} + 2^{n-2} \leqslant 2^n + x < 2^n$ donc $x_{n-1} = x_{n-2} = 1$. $y$ est donc un nombre négatif et $2^n + y = 2(2^n + x - 2^{n-1})$ donc $y = 2x$.

*b)* Notons $x_{n-1}\cdots x_1x_0$ la représentation binaire de $x$ ; alors `x >> 1` calcule le nombre $y$ représenté par $x_{n-1}x_{n-1}\cdots x_1$.

Lorsque $0 \leqslant x < 2^{n-1}$, $x_{n-1} = 0$ donc $y$ est positif et $y = \lfloor x/2 \rfloor$.

Lorsque $-2^{n-1} \leqslant x < 0$, $x_{n-1} = 1$ donc $y$ est un nombre négatif, et $2^n + y = \lfloor (2^n + x)/2 \rfloor + 2^{n-1} = 2^n + \lfloor x/2 \rfloor$ donc $y = \lfloor x/2 \rfloor$.

**Question 5.**

*a)* Si $x_{n-1}\cdots x_1x_0$ est la représentation binaire de $x$, alors `x & 1` calcule $x_0$ et `x >> 1` l'entier représenté par $0x_{n-1}\cdots x_2x_1$.
Le premier script calcule donc la somme $x_0 + x_1 + \cdots + x_{n-1}$, soit le nombre de bits égaux à 1 dans la décomposition de $x$.

*b)* Notons maintenant $k$ le plus petit des entiers pour lequel $x_k = 1$. Ainsi, $x$ est représenté par $x_{n-1}\cdots x_{k+1}10\cdots 0$.

$x-1$ est représenté par $x_{n-1}\cdots x_{k+1}01\cdots 1$ donc `x & (x−1)` calcule l'entier représenté par $x_{n-1}\cdots x_{k+1}00\cdots 0$. Cette opération supprime donc le bit égal à 1 le plus à droite dans la représentation binaire de $x$. Il en résulte que le second script calcule le nombre de bits égaux à 1 dans la décomposition de $x$.

*c)* Les deux scripts retournent le même résultat, mais le premier effectue un nombre d'opérations arithmétiques toujours supérieur ou égal au second. En effet, notons $p$ le plus grand des indices $x_i$ égaux à 1 et $q$ le nombre de bits égaux à 1. On a toujours $q \leqslant p$.

Dans le corps de la boucle conditionnelle, chacun des deux scripts réalise 3 opérations arithmétiques, mais le premier script exécute $p$ fois la boucle, tandis que le second n'exécute sa boucle que $q$ fois. Le second script est donc plus efficace.

*d)* Lorsque $x < 0$ on a $x_{n-1} = 1$ donc au bout de $n-1$ itérations de l'instruction `x = x >> 1` la variable `x` contiendra le nombre représenté par `11...1`, à savoir $-1$, et cette valeur ne sera plus jamais modifiée. Le script 1 ne se termine donc jamais.

En revanche, lorsque $x < 0$ l'instruction `x = x & (x-1)` continue de supprimer le bit égal à 1 le plus à droite (le raisonnement précédent tient toujours) donc le script 2 reste valable pour un entier négatif.

**Question 6.** Notons que `1 << k` calcule l'entier représenté par `00...0100...0` (avec $k$ zéros à droite et $n-1-k$ à gauche), valeur qui va nous servir de masque pour modifier le $k$-ième bit de $x$.
Chacune des fonctions demandées peut être définie en une ligne :

```python
def toggleBit(x, k):
    return x ^ (1 << k)
```

```python
def clearBit(x, k):
    return x & ~(1 << k)
```

```python
def setBit(x, k):
    return x | (1 << k)
```

```python
def getBit(x, k):
    return 1 & (x >> k)
```

# Partie II. Représentation machine des nombres flottants

**Question 7.** $2{,}5 = +1{,}25 \times 2^1$ donc $m = 1{,}25$ et $e = 1$. l'exposant $e$ est représenté par $1 + 2^3 - 1 = 8$ soit `1000`. La mantisse $m$ est représentée par `0100000` donc 2,5 est représenté par $\boxed{\texttt{0|1000|0100000}}$

$-42 = -(32+8+2) = -(2^5+2^3+2^1) = -(1+2^{-2}+2^{-4}) \times 2^5$ donc $m = 1+2^{-2}+2^{-4}$ et $e = 5$. L'exposant $e$ est donc représenté par $5 + 2^3 - 1 = 12$ soit `1100` et la mantisse $m$ par `0101000`. Ainsi, $-42$ est représenté par $\boxed{\texttt{1|1100|0101000}}$

$12{,}34 = 8 + 4 + 0{,}34 = (1 + 2^{-1} + 0{,}34/8) \times 2^3$. On a donc $e = 3$, représenté par $3 + 2^3 - 1 = 10$ soit `1010`.
Par ailleurs, $\dfrac{0{,}34}{8} = \dfrac{0{,}34}{2^3} = \dfrac{0{,}68}{2^4} = \dfrac{1{,}36}{2^5} = \dfrac{1}{2^5} + \dfrac{0{,}36}{2^5} = \dfrac{1}{2^5} + \dfrac{0{,}72}{2^6}$.
On a $\dfrac{1}{2} < 0{,}72 < 1$ donc $\dfrac{1}{2^7} < \dfrac{0{,}72}{2^6} < \dfrac{1}{2^6}$ et $m$ est compris entre $1+2^{-1}+2^{-5}+2^{-7}$ et $1+2^{-1}+2^{-5}+2^{-6}$. La plus proche de ces deux approximations est la première donc $m$ est représenté par `1000101` et 12,34 par $\boxed{\texttt{0|1010|1000101}}$

**Question 8.** Dans le format $(4,7)$, l'exposant $e$ vérifie $-7 \leqslant e \leqslant 8$ donc tout entier codé sur 8 bits (c'est-à-dire inférieur ou égal à 255) peut être représenté : si $i \leqslant 7$ l'entier $x = (x_i \cdots x_1 x_0)_2$ (avec $x_i = 1$) est représenté par la mantisse $m = (1, x_{i-1} \cdots x_1 x_0)_2$ et l'exposant $e = 2^i$.
L'entier suivant, $256 = 2^8$ reste représentable, avec $m = 1$ et $e = 2^8$. En revanche, $257 = (100000001)_2$ n'est pas représentable (il faudrait une mantisse à 8 bits).

Dans le format $(3,7)$, l'exposant $e$ vérifie $-3 \leqslant e \leqslant 4$ donc pour représenter un entier seuls les 4 premiers bits de la mantisse peuvent être non nuls. Le plus grand entier représentable est donc ici $(1,1111)_2 \times 2^4 = (11111)_2 = 31$, et $32 = 2^5$ est le premier entier non représentable (il faudrait un exposant à 5 bits).

**Question 9.**

```python
def lire_sequence(s):
    x = 0.
    for c in s:
        if c == 'O':
            x = x / 2
        elif c == 'I':
            x = (x + 1) / 2
    return x
```

**Question 10.**

*a)* En base 2 la division par 2 revient à un décalage des bits vers la droite, donc $p/2 = (0{,}0p_1p_2 \cdots p_n)_2$.
Par ailleurs, $p + 1 = (1{,}p_1p_2 \cdots p_n)_2$ donc $(p+1)/2 = (0{,}1p_1p_2 \cdots p_n)_2$.

*b)* La question précédente montre que la décomposition en base 2 permet de calculer aisément la succession d'opérations décrite par la chaîne de caractères : un `'O'` insère un 0 juste après la virgule, un `'I'` insère un 1 juste après la virgule. Par exemple,

$$(0)_2 \xrightarrow{\texttt{'I'}} (0{,}1)_2 \xrightarrow{\texttt{'O'}} (0{,}01)_2 \xrightarrow{\texttt{'O'}} (0{,}001)_2 \xrightarrow{\texttt{'I'}} (0{,}1001)_2 \xrightarrow{\texttt{'I'}} (0{,}11001)_2 \xrightarrow{\texttt{'O'}} (0{,}011\,001)_2 \xrightarrow{\texttt{'I'}} (0{,}101\,1001)_2$$

Le nombre décrit par la séquence est donc $(0{,}101\,1001)_2 = \dfrac{1}{2} + \dfrac{1}{8} + \dfrac{1}{16} + \dfrac{1}{128} = 0{,}695\,312\,5$.

**Question 11.** Réciproquement, étant donné un nombre dyadique $x = (0, x_1 x_2 \cdots x_n)_2$ on a $2x = (x_1, x_2 \cdots x_n)_2$. Ainsi,

$$x_1 = \begin{cases} 1 & \text{si } 2x \geqslant 1 \\ 0 & \text{sinon} \end{cases} \qquad \text{et} \qquad (0, x_2, \ldots, x_n)_2 = \begin{cases} 2x - 1 & \text{si } 2x \geqslant 1 \\ 2x & \text{sinon} \end{cases}$$

```python
def calcule_sequence(x):
    s = ''
    while x > 0:
        x *= 2
        if x >= 1:
            s = 'I' + s
            x -= 1
        else:
            s = 'O' + s
    return s
```