Contrôle d'informatique

Durée : 2 heures
Calculatrices interdites

Ce problème est constitué de deux parties totalement indépendantes.

Partie I. Opérateurs logiques sur les nombres entiers

Dans toute cette partie on suppose les entiers Python représentés par complémentation à 2 sur n bits, la valeur de n n'étant pas précisée mais suffisante pour pouvoir représenter tous les entiers qui interviendrons dans cette partie.

On suppose donnés trois opérateurs `&`, `|` et `^` qui calculent respectivement les operations ET, OU et OU-exclusif sur les representations binaires de deux entiers. Plus précisément, si les entiers x et y sont respectivement représentés par les suites de bits $x_{n-1}\cdots x_1x_0$ et $y_{n-1}\cdots y_1y_0$, alors `z = x & y` est l'entier représenté par la suite de bits $z_{n-1}\cdots z_1z_0$ definie par $z_i = \mathrm{ET}(x_i, y_i)$ pour tout i ; de même pour les operations `|` (OU) et `^` (OU-exclusif).
On rappelle que les operations ET, OU et OU-exclusif sont définies par les tables de vérité suivantes :

ET	0	1
0	0	0
1	0	1

OU	0	1
0	0	1
1	1	1

OU-exclusif	0	1
0	0	1
1	1	0

Par exemple, `6 & 12 = 4` car `6` est représenté par `0110` et `12` par `1100` donc `6 & 12` est représenté par `0100`.

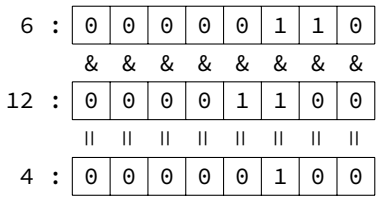

Figure 1 – Calcul sur $n = 8$ bits de `6 & 12`.

De même, on vérifiera que `6 | 12 = 14` et `7 ^ 12 = 10`.

Question 1. Réaliser les opérations suivantes :

```
13 & 25        13 | 25        13 ^ 25
15 & -7        15 | -7        15 ^ -7
```

Python fournit également un opérateur de négation sur les bits d'un entier : si l'entier x est représenté par la suite de bits $x_{n-1}\cdots x_1x_0$ alors `~x` retourne l'entier représenté par $\overline{x_{n-1}}\cdots\overline{x_1}\,\overline{x_0}$ avec $\overline{0} = 1$ et $\overline{1} = 0$.

Question 2. Si `x` est un entier relatif quelconque, que retourne `~x + 1` ? En déduire une caractérisation de l'entier calculé par l'instruction `x & -x` lorsque $x \neq 0$.

Question 3. On suppose que `a` et `b` sont des variables de type *int*. Quel effet le script suivant a-t-il sur le contenu de ces deux variables ? (Justifiez votre réponse.)

```
a = a ^ b
b = a ^ b
a = a ^ b
```

Enfin, Python fournit deux opérations `<<` et `>>` de décalage arithmétique sur les bits. Si x est représenté par la suite de bits $x_{n-1}\cdots x_1x_0$ et si $0 \leqslant k \leqslant n$ alors :

`x << k` est représenté par la suite de bits $x_{n-1-k}\cdots x_1x_0 0\cdots 0$;

`x >> k` est représenté par la suite de bits : $\begin{cases} 0\cdots 0x_{n-1}\cdots x_{k+1}x_k & \text{si } x_{n-1} = 0\,; \\ 1\cdots 1x_{n-1}\cdots x_{k+1}x_k & \text{si } x_{n-1} = 1. \end{cases}$

FIGURE 2 – *Shift left* et *shift right* sur $n = 8$ bits.

Question 4.

a) Montrer que si $-2^{n-2} \leqslant x < 2^{n-2}$ alors `x << 1` est la représentation binaire de $2x$ (distinguer les cas $x \geqslant 0$ et $x < 0$).

b) Montrer que si $-2^{n-1} \leqslant x < 2^{n-1}$ alors `x >> 1` est la représentation binaire de $\lfloor x/2 \rfloor$ (distinguer les cas $x \geqslant 0$ et $x < 0$).

Question 5. On suppose que `x` est une variable de type *int* contenant la représentation binaire d'un entier *positif*, et on considère les deux scripts suivants :

```
# script 1
c = 0
while x != 0:
    c = c + (x & 1)
    x = x >> 1
```

```
# script 2
c = 0
while x != 0:
    x = x & (x - 1)
    c = c + 1
```

a) Exprimer en fonction de la valeur initiale contenue dans la variable `x` la valeur prise par la variable `c` à l'issue du premier script.

b) Exprimer en fonction de la valeur initiale contenue dans la variable `x` la valeur prise par la variable `c` à l'issue du second script.

c) Pour une valeur de x donnée, lequel des deux scripts réalise le moins d'opérations sur les entiers ?

d) Que ce passe-t-il lorsqu'on applique chacun de ces deux scripts à une variable `x` contenant la représentation d'un nombre strictement négatif ?

Question 6. Dans cet exercice, on s'autorise uniquement l'usage des opérateurs `&`, `|`, `^`, `~`, `<<` et `>>`, *à l'exclusion de toute autre opération arithmétique*. Pour un entier x représenté par $x_{n-1} \cdots x_1 x_0$ et $k \in [\![0, n-1]\!]$, rédiger une fonction :

a) `toggleBit(x, k)` qui renvoie l'entier y représenté par $y_{n-1} \cdots y_1 y_0$ où $y_i = \begin{cases} x_i & \text{si } i \neq k \\ 1 - x_k & \text{si } i = k \end{cases}$

b) `setBit(x, k)` qui renvoie l'entier y représenté par $y_{n-1} \cdots y_1 y_0$, où $y_i = \begin{cases} x_i & \text{si } i \neq k \\ 1 & \text{si } i = k \end{cases}$

c) `clearBit(x, k)` qui renvoie l'entier y représenté par $y_{n-1} \cdots y_1 y_0$, où $y_i = \begin{cases} x_i & \text{si } i \neq k \\ 0 & \text{si } i = k \end{cases}$

d) `getBit(x, k)` qui retourne la valeur de x_k.

Partie II. Représentation machine des nombres flottants

Rappelons quelques notions au sujet de la représentation machine des nombres flottants. Ces derniers sont définis par trois composantes : le *signe* s, l'*exposant* e, et la *mantisse* m. Ainsi, le triplet (s, e, m) représente le nombre $\pm m.2^e$.
Le signe est représenté par un simple bit : 0 pour les nombres positifs et 1 pour les nombres négatifs. Nous parlerons de flottants au format (k, ℓ) lorsque k bits seront utilisés pour l'exposant et ℓ bits pour la mantisse. Par exemple, le type *float64* utilisé par PYTHON et décrit en cours correspond au format $(11, 52)$.
Dans le format (k, ℓ), l'exposant e est représenté par l'entier non signé $e + 2^{k-1} - 1$. Enfin, la mantisse, dont la valeur appartient à l'intervalle $[1, 2[$, est représentée par une suite de ℓ bits dont le bit le plus significatif est de poids $1/2$. Par exemple, lorsque $\ell = 3$ la suite de bits `101` représente la mantisse $m = 1 + 1/2 + 1/8 = 1{,}625$.

Remarque. Dans ce problème ne seront pas prises en compte les représentations dénormalisées des nombres ainsi que les représentations de 0, de l'infini et de *not a number*.

Question 7. Dans le format $(4,7)$, donner les 12 bits de la représentation des nombres flottants suivants (sous la forme `s|e|m` où `s`, `e` et `m` désignent les représentations machine de s, e et m) des nombres suivants :

$$2{,}5 \qquad -42 \qquad 12{,}34$$

Dans le cas où l'un de ces nombres ne serait pas exactement représentable dans ce format, il sera représenté par le nombre flottant le plus proche.

Question 8. Dans le format $(4,7)$, quel est le plus petit entier positif non représentable ? Et dans le format $(3,7)$?

Dans la fin de ce problème, on considère une chaîne de caractères constituée uniquement de deux caractères distincts : `'O'` et `'I'`. La lecture de cette chaîne décrit un calcul qui s'effectue de la manière suivante en partant de la valeur `x = 0.` :

- lorsqu'on lit le caractère `'O'` on remplace `x` par `x / 2` ;
- lorsqu'on lit le caractère `'I'` on remplace `x` par `(x + 1) / 2`.

Par exemple, la lecture de la chaîne `'IOII'` conduit aux valeurs successives :

$$\texttt{0.} \longrightarrow \texttt{0.5} \longrightarrow \texttt{0.25} \longrightarrow \texttt{0.625} \longrightarrow \texttt{0.8125}.$$

Question 9. Rédiger en Python une fonction `lire_sequence(s)` qui prend en argument une chaîne de caractères `s` composée exclusivement des caractères `'O'` et `'I'` et qui retourne le résultat du calcul décrit par `s`.
Par exemple, `lire_sequence('IOII')` devra retourner le nombre flottant `0.8125`.

Question 10. On considère un nombre dyadique $p \in [0,1[$ dont la décomposition en base 2 s'écrit : $p = (0, p_1 p_2 \cdots p_n)_2$.

a) Donner la décomposition en base 2 des nombres $\dfrac{p}{2}$ et $\dfrac{p+1}{2}$.

b) En déduire la décomposition binaire du nombre flottant qu'on obtient par le calcul décrit par la chaîne `'IOOIIOI'`.

Question 11. Rédiger en Python une fonction `calcule_sequence(x)` qui prend en argument un nombre flottant `x` de l'intervalle $[0,1[$ et qui retourne la chaîne de caractères s qui décrit le calcul permettant d'aboutir à `x`. Par exemple, `calcule_sequence(0.8125)` devra retourner la chaîne de caractères `'IOII'`.

♦ ♦
♦