info223 : Science informatique
TD 1 : la base 2, représentation des nombres

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : `Pierre.Hyvernat@univ-savoie.fr`
www : `http://www.lama.univ-savoie.fr/~hyvernat/`

Partie 1 : nombres entiers positifs

Exercice 1 : Application du cours

Question 1. Donnez la représentation en base 2 des nombres suivants :

12, 24, 48, 21, 37, 142, 1000.

Question 2. Donnez la valeur (représentation en base 10) des nombres suivants :

$\underline{\texttt{1101010}}_2$, $\underline{\texttt{110101}}_2$, $\underline{\texttt{11010}}_2$, $\underline{\texttt{1010101}}_2$, $\underline{\texttt{1111111111}}_2$, $\underline{\texttt{101110100101}}_2$

Question 3. Pour chacun des nombres de la question précédente, donnez la représentation en hexadécimal (base 16).

Question 4. Quand on veut trouver la représentation en base 2 d'un nombre écrit en base 10, est-il plus simple de commencer par chercher les bits de poids forts, ou ceux de poids faible.

Décrivez un algorithme (méthode formalisée, programmable) pour effectuer cette transformation.

Question 5. Faites les calculs suivants, en détaillant :
- $\underline{\texttt{110101001}}_2 + \underline{\texttt{1111}}_2$,
- $\underline{\texttt{1111111}}_2 + \underline{\texttt{101010}}_2$,
- $\underline{\texttt{110101001}}_2 \times \underline{\texttt{1010}}_2$,
- $\underline{\texttt{1101}}_2 \times \underline{\texttt{100010001}}_2$,
- $\underline{\texttt{11111}}_2 \times \underline{\texttt{11111}}_2$.

Question 6. Comparez les nombres suivants sans l'aide d'un ordinateur :
- 2^{50} et 10^{20},
- 2^{512} et 10^{100}.

Question 7. Convertissez entre les base 16 et 2 :
- $\underline{\texttt{1110001110001110001110001110000}}_2$,
- $\underline{\texttt{111011011110101100010111101011011}}_2$,
- $\underline{\texttt{2468ace}}_{16}$,
- $\underline{\texttt{deadbeef}}_{16}$.

Question 8. Convertissez les nombres de la questions précédentes vers la base 8.

Exercice 2 : un peu plus loin

Question 1. Quelle est la valeur de $\underline{\texttt{10101...101}}_2$?

Question 2. Cherchez une formule pour trouver la valeur de $11\dots11 \times 11\dots11$.

Question 3. Écrivez 42 en base 3.

Question 4. À quoi correspond le nombre $\underline{\texttt{1436}}_7$?

Question 5. Est-il facile de traduire un nombre écrit en base 17 en base 15 ?

Exercice 3 : le code de Gray

Si on énumère tous les entiers écrits en base 2 sur 3 bits dans l'ordre, on obtient :

$$\underline{\texttt{000}}_2,\quad \underline{\texttt{001}}_2,\quad \underline{\texttt{010}}_2,\quad \underline{\texttt{011}}_2,\quad \underline{\texttt{100}}_2,\quad \underline{\texttt{101}}_2,\quad \underline{\texttt{110}}_2,\quad \underline{\texttt{111}}_2.$$

Un *code de Gray sur 3 bits* est une énumération des entiers écrits en base 2 sur 3 bits avec la propriété suivante :

un seul bit est modifié quand on passe d'un entier au suivant.

Question 1. Trouvez un code de Gray sur 3 bits.
La solution est-elle unique ?

Question 2. Trouvez un code de Gray sur 3 bits où la propriété est également vérifiée entre le premier et le dernier entier.

Question 3. On peut générer un code de Gray $\Gamma(n)$ sur n bits de la manière suivante :
- $\Gamma(0) = \_$ (vide),
- $\Gamma(n+1) = \underline{\texttt{0}} \cdot \Gamma(n), \underline{\texttt{1}} \cdot \overline{\Gamma(n)}$.

Ici, le "$\cdot$" signifie qu'on rajoute un bit ($\underline{\texttt{0}}$ ou $\underline{\texttt{1}}$) devant tous les entiers ; et l'opération "$\overline{G}$" signifie qu'on prend les entiers de G dans l'ordre inverse...
Calculez $\Gamma(i)$ pour $i = 1$, 2, 3 et 4.

Question 4. Le code de Gray $\Gamma(n)$ a la propriété suivante : le i-ème entier de $\Gamma(n)$ est égal à $i \oplus i/2$, ou "$\oplus$" est le le XOR bit à bit.
Calculer les entiers 17, 18, 19 et 20 de $\Gamma(6)$, et vérifiez que la propriété des code de Gray est bien vérifiée.

Partie 2 : Nombres entiers négatifs

Question 1. Donnez la représentation en complément à 2 *sur 8 bits* des nombres :
- $a = 42$,
- $b = -70$,
- $c = -88$,
- $d = -34$.

Calculez $e = a + b$ en utilisant exactement l'algorithme d'addition en base 2. Vérifiez que le résultat est correct.
Calculez $f = d + c$ de la même façon et vérifiez que le résultat est correcte.
Calculez maintenant $d + f$. Qu'en pensez-vous ?

Exercice 4 : Pourquoi la représentation en complément à 2

Question 1. En complément à 2 sur k-bit, l'entier "-1" est représenté par

$$\underbrace{\underline{\texttt{11}\ldots\texttt{11}}_2}_{k}$$

Vérifiez qu'en calculant normalement $\underline{\texttt{11}\ldots\texttt{11}}_2 + 1$ sur k bit, on obtient bien $\underline{\texttt{00}\ldots\texttt{00}}_2$.

Question 2. Si on identifie les entiers sur k-bit à l'ensemble des entiers *modulo* 2^k, on a :

$$0 \equiv 2^k \pmod{2^k}$$

càd

$$0 \equiv \underline{\texttt{100}\ldots\texttt{00}}_2 \pmod{2^k}\ .$$

Avec ceci en tête, il est naturel (??) d'avoir

$$-n \equiv 2^k - n \pmod{2^k} \; .$$

Si on note $\overline{n}$ la "négation bit à bit, sur k bit" du nombre n, vérifiez :
- que $n + \overline{n} = \underline{\texttt{11}\ldots\texttt{11}}_2$,
- et donc que $n + \overline{n} + 1 = \underline{\texttt{100}\ldots\texttt{00}}_2 \equiv 0 \pmod{2^k}$.

On a donc bien que la représentation de "$-n$" est celle de $\overline{n} + 1$.

Question 3. Justifiez pourquoi $\overline{n} + 1 = \overline{n - 1}$.

Partie 3 : Les flottants

On rappelle que les flottants IEEE sont représentés avec :
- un signe s (un bit),
- un exposant e en base 2,
- une mantisse de x bits : $m_1, m_2, \ldots, m_k$,

et que la valeur correspondante

$$(-1)^s \times \underline{\texttt{1,}m_1 m_2 \ldots m_k}_2 \times 2^{e-B}$$

Où B est le *biais*.

Question 1. On suppose pour le moment que l'exposant est stocké sur 3 bits et la mantisse sur 4 bits. Le biais B vaut 3.

Quels sont les flottants représentés par :
- <u>`0 110 1011`</u>,
- <u>`1 011 0010`</u>,
- <u>`0 111 1011`</u>.

Question 2. Comment représente-t'on
- -5.5,
- 7.25,
- 3.75.

Question 3. Ceux qui ont un ordinateur sous la main peuvent essayer

```
a = 49.0
b = 1.0/a
c = b*a

if (1.0 == c):
  print("OK")
else:
  print("Oups...")
```

Pour les autres, que pensez-vous qu'il peut se passer ?

Comment corriger le problème ?

Question 4. Que pensez-vous des calculs suivants :
- `49. * (1. / 49.)`,
- `(49. * 1.) / 49.` ?

Question 5. La ligne suivante en Python permet d'afficher la valeur de 1/3 avec 60 chiffres après la virgule :

```
print ("1/3 = %.60f" % (1/3))
```

Voici ce qu'elle affiche sur mon ordinateur :

```
>>> print ("1/3 = %.60f" % (1/3))
1/3 = 0.333333333333333314829616256247390992939472198486328125000000
```

Commentez et expliquez le résultat.

Question 6. Si on affiche 1/4, on obtient le bon résultat :

```
>>> print ("1/4 = %.60f" % (1/4))
1/4 = 0.250000000000000000000000000000000000000000000000000000000000
```

Pourquoi ?

Question 7. Expliquez le résultat de l'expérience suivante :

```
>>> 0.1
0.1
>>> 0.2
0.2
>>> 0.1+0.2
0.30000000000000004
```

Question 8. Pour les flottants IEEE754 (32 bits), les valeurs des paramètres sont :
- 8 bit pour l'exposant,
- 127 pour le biais,
- 23 bit pour la mantisse.

On rappelle que lorsque l'exposant est $\underline{\texttt{00000000}}_2$ ou $\underline{\texttt{11111111}}_2$, on parle de nombre *sous-normal*.

Question 9. Quel est le plus grand nombre réel que l'on peut représenter ?

Question 10. Quel est le plus petit nombre réel strictement positif que l'on peut représenter ? Que se passe-t'il si vous ajouter ce nombre à celui de la question précédente ?