Chapitre ITC4

Représentation des nombres dans un ordinateur

Section 1

Écriture en base n

Principes généraux

Codage

- On doit disposer de n symboles
- $$\begin{aligned}\overline{a_k a_{k-1} \dots a_2 a_1 a_0}^n &= a_k.n^k + \dots + a_2.n^2 + a_1.n^1 + a_0 \\ &= \sum_{i=0}^{k} a_i.n^i\end{aligned}$$

Exemple en base 10

$7231 = 7 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 1 \times 10^0$

Notation binaire

Caractéristiques

- Deux symboles : 0 et 1, correspondant à deux états possible d'une mémoire d'ordinateur
- Tables d'addition et de multiplication très simples
- Noté précédé d'un `b` en Python

Exemple

$b100101 = 2^5 + 2^2 + 2^0 = 37$

Valeurs importantes

$2^8 = 256$	$2^{10} = 1024 \approx 10^3$
$2^{16} = 65536$	$2^{20} = 1048576 \approx 10^6$

Notation hexadécimale (pas à connaître)

Caractéristiques

- 16 symboles : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Correspondance simple avec 4 chiffres binaires

x0=b0000	x1=b0001	x2=b0010	x3=b0011
x4=b0100	x5=b0101	x6=b0110	x7=b0111
x8=b1000	x9=b1001	xA=b1010	xB=b1011
xC=b1100	xD=b1101	xE=b1110	xF=b1111

- Noté précédé d'un `x` en Python

Exemple

$xD4 = 13 * 16 + 4 = 212 (= b11010100)$

Section 2

Codage des nombres

Regroupement par octets

Définition d'un octet

Une mémoire contient des éléments appelés *bits* pouvant prendre deux valeurs notées 0 et 1.
Les bits sont regroupés par 8 pour former un *octet* (en anglais : *byte*).

Attention ! bit ou octet ?

- Internet par ADSL : jusqu'à 20 Mbits/s...= 2,5 Mo/s
- USB-3 : jusqu'à 4,8 Gbits/s...= 600 Mo/s

Codage des entier positifs

Les types d'entiers positifs

Nombre d'octets	Nombre de bits	Valeur maximale
1	8	$2^8 = 256$
2	16	$2^{16} = 65536$
4	32	$2^{32} = 4.294.967.296$
8	64	$2^{64} \approx 1,8.10^{19}$

Dépassement de capacité

- En typage statique, on risque un dépassement de capacité :

  Exemple en C ;

  ```c
  unsigned char a;
  a=120;
  a+=300;
  ```

  à la fin,

  a=(120+300)%256=164

- En typage dynamique, par exemple en Python, on ne risque à peu près rien.

Codage des entier positifs

Méthode de calcul du codage d'un entier

- on effectue une division entière du nombre par 2
- on effectue la division entière du quotient par 2, de façon récurrente, jusqu'à ce que le quotient obtenu vaille 0
- le nombre en binaire est obtenu en écrivant de gauche à droite les restes obtenus de bas en haut

Codage des entier positifs

Codage de 78

- $78 = 2 * 39 + 0$
- $39 = 2 * 19 + 1$
- $19 = 2 * 9 + 1$
- $9 = 2 * 4 + 1$
- $4 = 2 * 2 + 0$
- $2 = 2 * 1 + 0$
- $1 = 2 * 0 + 1$

On écrit les restes de bas en haut : $78 = b1001110$.
Si on code sur un octet, on ajoute des zéros devant : $b01001110$.

Codage des entiers relatifs

Principe du codage

- Le bit le plus fort indique le signe : $0 \Leftrightarrow +$ et $1 \Leftrightarrow -$
- Pour les nombres positifs, on code normalement
- Pour les nombres négatifs, on code la valeur absolue, on applique une opération NOT à tous les bits, et on ajoute 1

Quelques types d'entiers négatifs

Octets	Bits utiles	Plage de valeurs
1	7	[-128..127]
2	15	[-32768..32767]
4	31	[-2147483648..2147483647]

Exemple de Codage/décodage d'un entier négatif

Codage de −78 sur 8 bits :

- on montre d'abord que 78=b01001110
- on code -78 comme `NOT(b01001110)+1=b10110001+1=b10110010`

Décodage de $b10011010$

C'est un entier négatif. Pour trouver sa valeur absolue, on refait la même opération : NOT puis $+1$

- opérateur NOT : $b01100101$
- ajout de 1 : $b01100110$
- calcul de $2^6 + 2^5 + 2^2 + 2^1 = 102$
- le nombre codé est donc 102

Notation scientifique en base 10

- On écrit un réel sous la forme $4,213.10^8$
- Si on fixe le nombre maximal de chiffres significatifs, tout réel peut s'écrire comme un **entier** relatif multiplié par une puissance **entière** de 10 ; par exemple $4,213.10^8 = 4213.10^5$

Notation scientifique en base 10

- On écrit un réel sous la forme $4,213.10^8$
- Si on fixe le nombre maximal de chiffres significatifs, tout réel peut s'écrire comme un **entier** relatif multiplié par une puissance **entière** de 10 ; par exemple $4,213.10^8 = 4213.10^5$

Notation «scientifique» en base 2

- Si on fixe le nombre maximal de chiffres significatifs, tout réel peut s'écrire comme un **entier** relatif multiplié par une puissance **entière** de 2 : $x = \text{mantisse} \times 2^{exposant}$
- La taille de la **mantisse** m (-1bit de signe) indique le nombre de chiffres significatifs selon la règle approximative

  10 bits $\Leftrightarrow$ 10 chiffres en base 2 $\Leftrightarrow$ 3 chiffres en base 10
- La taille de l'**exposant** e indique les puissances maximales et minimales représentables.

Codage des réels selon la norme IEEE 754

La norme IEEE 754 utilise une technique un peu différente

Chaque réel est codé avec :

- un bit de signe s
- un exposant non signé e
- une mantisse non signée m

Sa valeur vaut alors $(-1)^s \times (1 + \frac{m}{2^n}) \times 2^{(}e - d)$ où d est un décalage donné par la norme et n le nombre de bits de m.

La norme IEEE 754 définit des réels particuliers :

- 0
- $+\infty$ (en Python, `float("inf")`)
- $-\infty$ (en Python, `float("-inf")`)
- Not a Number (en Python, `float("nan")`) pour les résultats indéfinis comme $0/0$, $\sqrt{-2}$ ou encore $0 \times \infty$

Codage des réels selon la norme IEEE 754

Python utilise par exemple des réels codés sur 64 bits

- 1 bit de signe
- 11 bits d'exposant
- 52 bits de mantisse
- un décalage de 1023

Ainsi le nombre (s=1,m=2746133873243244,e=1036) représente $(-1)^1 \times \left(1 + \frac{2746133873243244}{2^5 2}\right) \times 2^{1036-1023} = -13187,188416148$

Danger avec les fractions de grands/petits nombres

Notebook

Dans le calcul d'une fraction $\frac{a}{b}$, il est possible que le résultat soit dans les bornes autorisées, mais que le numérateur ou le dénominateur soit trop petits.
Solution : si on travaille sur des grandeurs énormes ou toutes petites, on prend des unités plus adaptées :

- pour la physique atomique : le nanomètre, l'unité de masse atomique, l'électron-Volt,...
- pour l'astronomie : l'unité astronomique ou l'année lumière, l'année, la masse solaire,...

1. Écriture en base n

2. Codage des nombres
   - Entiers
   - Réels
   - Limites de l'exposant
   - Limites de précision

Danger des erreurs à croissance exponentielle

Erreur à croissance exponentielle Notebook

Si à chaque étape d'un calcul, l'erreur est multipliée par $\alpha > 1$, alors la suite des erreurs est une suite géométrique explosive.

Qui est en cause ?

- Parfois c'est l'algorithme qui est mal conçu
- Parfois c'est une propriété fondamentale du système (système chaotique)

Danger des nombres très proches

Limite de précision

Le plus petit nombre en Python tel que $1 + \varepsilon \neq 1$ vaut $\varepsilon = 2,2.10^{-16}$

Notebook

Le calcul de la différence de deux nombres très proches est peu précis : $1 + 10^{-18} - 1 = 0$ pour Python !.

Problème de comparaison à zéro

Notebook

Une erreur d'arrondi peut transformer un 0 en une valeur faible mais non nulle, entrainant :

- des boucles conditionnelles qui ne se terminent jamais
- des valeurs complexes dans un problème à valeurs réelles

À connaître

À connaître

- Le codage/décodage des entiers positifs
- Le codage/décodage des entiers relatifs
- Le principe général de stockage des entiers et des réels (mantisse + exposant)
- Le décodage d'un réel
- Le problème de la comparaison d'un réel à 0