TD 2 & 3 : représentation des nombres

I. Les nombres entiers

On distingue la représentation des nombres entiers naturels de celle des entiers relatifs. On parle alors de représentation non signée (entiers naturels) ou signée (entiers relatifs).

Représentation non signée

C'est la représentation classique en binaire naturel. Avec n bits, on représente alors les entiers de l'intervalle $[0, 2^n-1]$. Ainsi sur 8 bits, on peut coder des nombres entre 0 et 255.

Question 1

Avec 8 bits, on a vu que l'on peut représenter des nombres entre 0 et 255. Donnez l'intervalle pour 16 bits et 32 bits.

Question 2

Convertissez de décimal en binaire les nombres donnés, et effectuez les additions sur 8 bits :

141 + 43 87 + 18 193 + 101

Quel lien faites-vous avec le registre d'état d'un processeur ?

Représentation signée

Il existe différentes représentations possibles, mais la plus utilisée est le « complément à deux ». Dans ce cas, le premier bit indique le signe (0 si le nombre est positif – ou nul, 1 s'il est négatif). Ainsi, sur n bits, on représente les nombres de l'intervalle $[-2^{n-1}, 2^{n-1}-1]$. Après le bit de signe, les nombres positifs sont représentés par leur valeur non signée, tandis que les nombres négatifs, cette valeur non signée est complémentée : l'avantage majeur étant que l'addition est aisée.

Question 3

Cherchez sur Wikipédia « complément à deux » et lisez l'explication. Expliquez alors le *pourquoi* et le *comment*.

Question 4

Donnez l'intervalle des nombres signés pour 16 bits et 32 bits

Question 5

Donnez la représentation, sur 8 bits, des nombres décimaux suivants : 112, -112, 250, -250

Donnez la valeur décimale des nombres suivants : 01101100, 11100010, 10000011, 00001101

Question 6

Convertissez de décimal en binaire les nombres donnés, et effectuez les additions sur 8 bits :

145 + (-15) (-32) + (-64) (-201) + (-66)

Question 7

Voici sur 3 bits les représentations non signées et signées en complément à 2 :

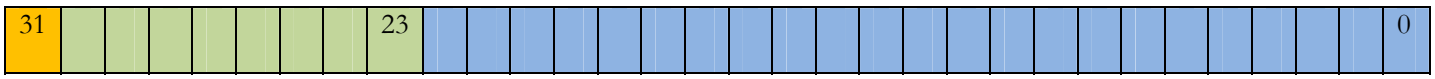
Décimal Non signée	Binaire naturel	Décimal signée	Binaire Comp. 2
0	000	-4	100
1	001	-3	101
2	010	-2	110
3	011	-1	111
4	100	0	000
5	101	1	001
6	110	2	010
7	111	3	011

Réorganisez le tableau en faisant correspondre les zéros. Que remarquez-vous ?

II. Les flottants et la norme IEEE-754

La version 1985 de cette norme définit deux formats principaux pour représenter les nombres à virgule flottante : simple précision (32 bits) ou double précision (64 bits). Nous allons étudier le format simple précision.

Le nombre est codé sur 32 bits avec le bit de poids fort pour le signe (0 positif, 1 négatif), 8 bits pour l'exposant, et 23 bits pour la mantisse :



Exposant et décalage

L'exposant peut être positif ou négatif, cependant la représentation habituelle en complément à 2 n'est pas utilisée, on utilise la technique du décalage afin de le stocker sous forme d'un nombre non signé. Si on utilise e bits pour l'exposant, le décalage est de $2^{e-1} - 1$, dans notre cas e=8 donc le décalage est de 127.

Selon la valeur E de l'exposant et P de la mantisse, on distingue plusieurs cas :

Exposant	Mantisse	Valeur
0000 0000	Tous les bits à 0	Valeur zéro
0000 0000	Au moins un bit non nul	Valeur dénormalisée : $(-1)^S * 2^{1-(2e-1-1)} * (P/2^p)$
1111 1111	Tous les bits à 0	Valeur infinie
1111 1111	Au moins un bit non nul	Pas un nombre (NaN)
Autre	Quelconque	$(-1)^S * 2^{E-(2e-1-1)} * (1+P/2^p)$

Prenons l'exemple du nombre binaire 0 01111100 01000000000000000000000. Le bit de signe est nul donc le nombre est positif. L'exposant vaut 01111100 soit 124 en décimal ce qui donne avec le décalage 124 - 127 = -3. Enfin la mantisse donne 1,01 en binaire soit 1,25 en décimal ($1\times2^0 + 0\times2^{-1} + 1\times2^{-2}$). Le nombre est donc $+1.25*2^{-3} = +0.15625$

Question 1

Donnez la valeur décimale du nombre 11000000010010010000111001010110

Question 2

Combien faut-il de puissances négatives de 2 pour représenter la valeur décimale 0,1 ?Qu'en déduisez-vous, par exemple pour une boucle du type `for (i=-5 ; i<= 5 ; i+= 0.1)` ?

Codage d'un flottant

Voyons comment représenter la valeur 114,3125 : tout d'abord c'est un nombre positif, donc le bit de signe vaut zéro. Ensuite nous écrivons le nombre (sans le signe) en binaire : 114 décimal vaut 1110010. Pour la partie décimale, on va soustraire successivement les puissances négatives de 2 (0.5, 0.25, 0.125, 0.0625, 0.0312, …). On obtient 0,3125 codé en binaire 0101, soit le nombre complet 1110010,0101.

Ensuite, on décale la virgule vers la gauche de façon à ne laisser qu'un 1 sur la gauche, c'est la normalisation : on obtient alors $1,1100100101 \times 2^6$. La mantisse est alors à droite de la virgule, complétée à droite avec des zéros pour obtenir 23 bits. On obtient donc : 1100100101 0000000000000.

Enfin pour l'exposant, qui vaut 6, on le convertit en binaire puis on le décale : 6+127 = 133, soit 1000 0101 en binaire.

On obtient donc au final 114.3125 codé 01000010111001001010000000000000

Arrondi

Si un nombre ne peut pas être représenté exactement, on utilise une valeur arrondie. La norme 1985 définit quatre modes possibles :

- Le mode par défaut est l'arrondi au plus proche, en cas d'égalité on choisit la valeur paire (bit à 0).
- L'arrondi vers 0 arrondit à la valeur la plus proche de 0, c'est une troncature
- L'arrondi vers +∞ arrondit à la plus proche valeur la plus grande
- L'arrondi vers -∞ arrondit à la plus proche valeur la plus petite

Question 2

Donnez le codage de la valeur décimale -1250,33

Addition sur les flottants

Considérons les valeurs décimales 1,5 et 0,5 et leur codage :

- $1,5 = 1.1 \times 2^0 = 00111111110000000000000000000000$
- $0,5 = 1.0 \times 2^{-1} = 00111111000000000000000000000000$.

On ramène les deux nombres au même exposant, le plus grand des deux. Ainsi 1.0×2^{-1} devient 0.1×2^0. Le zéro le plus à droite est supprimé. On ajoute ensuite les deux mantisses en tenant compte du signe : 1.1 + 1.0 = 10.0

On renormalise ensuite le nombre obtenu, on obtient donc 0 10000000 00000000000000000000000 soit 2.

Question 3

Effectuer l'addition 50,34 + 100,01

Précision et erreurs d'arrondi

La suite de J.M Muller est définie de la façon suivante :

- $u_0 = 2$
- $u_1 = -4$
- $u_{n+1} = 111 - 1130 / u_n + 3000 / u_n.u_{n-1}$

Nous allons utiliser MuPad Pro, un logiciel de calcul formel, pour évaluer cette suite, notamment sa limite quand n tend vers l'infini. Cette partie du TD a été réalisée avec J.C. Jolly, et certains aspects seront abordés ultérieurement en mathématiques : nous n'expliciterons pas ici les concepts mathématiques sous-jacents.

Dans un premier temps, nous allons considérer la fonction associée et voir sa représentation. Pour cela, saisissez dans MuPad :

```
eq := (111-1130/x+3000/x^2-x) * x^2

plotfunc2d (eq, x=0..110)

plotfunc2d (eq, x=4..7)
```

Question 4

Quelles sont les racines de cette fonction ?

La présence de ces racines signifie, pour la suite u_n, qu'elle est susceptible de converger vers chacune de ces valeurs : cela dépend des valeurs initiales u_0 et u_1.

Nous allons maintenant saisir cette suite afin d'en calculer les termes successifs. Pour cela nous allons légèrement transformer l'écriture :

```
h := (u,v) -> (v,111-1130/v+3000/(u*v))
```

Ensuite nous donnons les valeurs initiales :

```
u0 := 2; v0 := -4;
```

Nous pouvons alors calculer la valeur suivante de la suite (en formel) :

```
h(u0,v0)
```

A n'importe quel moment, le résultat précédent d'une opération (noté %) peut être évalué :

```
map(%,float)
```

On peut également calculer plusieurs valeurs en combinant l'appel de la suite :

```
(h@@10)(u0,v0)
```

Question 5

Vers quelle valeur cette suite converge-t-elle ?

Question 6

Remplacez 3000 par 2999 et 3001. La suite converge-t-elle toujours vers la même valeur ? Reprenez la valeur 3000 et changez u_0. Quelle est la convergence ? Qu'en déduisez-vous sur la stabilité de la convergence de suite ?

Question 7

Programmez cette suite dans le langage de votre choix et vérifiez le résultat obtenu précédemment (moins de 20 itérations sont en général suffisantes).

Question 8

Comment expliquez-vous ce résultat ?