CI 1 : Architecture matérielle et logicielle

Chapitre 4 – Principe de la représentation des nombres réels en mémoire

Savoir

- Capacité Dec - C3 : Initier un sens critique au sujet de la qualité et de la précision des résultats de calculs numériques sur ordinateur
- Principe de la représentation des nombres réels en mémoire

1 Représentation de la partie fractionnaire des réels – $\mathbb{R}$

En notation décimale, les chiffres à gauche de la virgule représentent des entiers, des dizaines, des centaines, etc. et ceux à droite de la virgule, des dizièmes, des centièmes, des millièmes, etc.

Exemple

$$3,3125_{(10)} = 3\cdot 10^0 + 3\cdot 10^{-1} + 1\cdot 10^{-2} + 2\cdot 10^{-3} + 5\cdot 10^{-4}$$

Par analogie, pour écrire un nombre binaire à virgule, on utilise les puissances négatives de 2.

Exemple

$$\begin{aligned} 11,0101_{(2)} &= 1\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4} \\ &= 2+1+0+0,25+0+0,0625 \\ &= 3,3125_{(10)} \end{aligned}$$

Le codage de la partie entière (3 dans l'exemple précédent) ne pose pas de problèmes particuliers. Pour la partie fractionnaire (0,3125), il est nécessaire d'adapter la procédure.

Méthode

Conversion d'une partie fractionnaire en binaire

1. On multiplie la partie fractionnaire par 2.
2. La partie entière obtenue représente le poids binaire (limité aux seules valeurs 0 ou 1).
3. La partie fractionnaire restante est à nouveau multipliée par 2.
4. On procède ainsi de suite jusqu'à ce qu'il n'y ait plus de partie fractionnaire ou que le nombre de bits obtenus correspond à la taille du mot mémoire dans lequel on stocke cette partie.

Exemple

Conversion de la partie fractionnaire 0,3125

$$\begin{array}{lllllllll} 0{,}3125 & \text{x} & 2 & = & 0{,}625 & = & \boxed{0} & + & 0{,}625 \\ 0{,}6250 & \text{x} & 2 & = & 1{,}250 & = & \boxed{1} & + & 0{,}250 \\ 0{,}2500 & \text{x} & 2 & = & 0{,}500 & = & \boxed{0} & + & 0{,}500 \\ 0{,}5000 & \text{x} & 2 & = & 1{,}000 & = & \boxed{1} & + & 0{,}000 \end{array}$$

On considère les parties entières de haut en bas : $0,3125_{(10)} = 0,0101_{(2)}$.

Remarque

Inconvénients

Savoir coder la partie fractionnaire d'un nombre à virgule ne suffit pas pour coder tous les nombres à virgule en binaire. En effet, la gestion d'une virgule virtuelle par programme n'est pas aisée. De plus, cette méthode ne permet pas de représenter des nombres très grands ou très petits comme le nombre d'Avogadro ($6,02214129..\cdot 10^{23}$) ou la constante de Planck ($6,62606957\cdot 10^{-34}$).

Exemple

Convertir la partie fractionnaire 0,1

2 Représentation de la virgule flottante

Pour représenter des réels, nombres pouvant être positifs, nuls, négatifs et non entiers, on utilise la représentation en virgule flottante (*float* en anglais) qui fait correspondre au nombre 3 informations :

$$-243,25_{(10)} = \underbrace{-}_{1}\ \underbrace{0,24325}_{2} \cdot 10^{\overbrace{3}^{3}}$$

On appelle alors :

1. le signe (positif ou négatif) ;
2. la mantisse (nombre de chiffres significatifs) ;
3. l'exposant : puissance à laquelle la base est élevée.

Sous cette forme normalisée, il suffit de mémoriser le signe, l'exposant et la mantisse pour avoir une représentation du nombre en base 10. Il n'est pas utile de mémoriser le 0 avant la virgule puisque tous les nombres vont commencer par 0. En faisant varier l'exposant, on fait « flotter » la virgule décimale.

C'est cette méthode que l'on va adapter pour coder les réels en binaire naturel. Il faut au préalable les écrire sous la forme (norme IEEE 754 – Institute of Electrical and Electronics Engineers) :

$$\text{signe 1, mantisse x } 2^{\text{exposant}}$$

Le mot binaire obtenu sera la juxtaposition de 3 parties :

Le tableau décrit la répartition des bits selon le type de précision : la taille de la mantisse (m bits) donne la précision mais suivant la valeur de l'exposant, la précision sera totalement différente.

	Signe	Exposant	Mantisse
Simple précision – 32 bits	1	8	23
Double précision – 64 bits	1	11	52
Précision étendue – 80 bits	1	15	64

2.1 Procédure de conversion de réel en binaire (hexadécimale)

Méthode

1. Convertir en binaire les partie entière et fractionnaire du nombre sans tenir compte du signe.
2. Décaler la virgule vers la gauche pour le mettre sous la forme normalisée (IEEE 754).
3. Codage du nombre réel avec les conventions suivantes :
   - Signe = 1 : Nombre négatif (Signe = 0 : Nombre positif)
   - Le chiffre 1 avant la virgule étant invariant pour la forme normalisée, il n'est pas codé.
   - On utilise un exposant décalé au lieu de l'exposant simple (complément sur octet). Ainsi, on ajoute à l'exposant simple la valeur 127 en simple précision et 1023 en double précision (c'est à dire $2^{n-1}-1$ où n est le nombre de bits de l'exposant).
   - La mantisse est complétée à droite avec des zéros.

Exemple

On désire représenter le nombre - 243,25 en virgule flottante au format simple précision.

1. $243,25_{(10)} = 11110011,01_{(2)}$
2. $243,25_{(10)} = 1,111001101_{(2)} * 2^7$: décalage de 7 bits vers la gauche
3. Exposant décalé : $7 + 127 = 134_{(10)} = 1000\,0110_{(2)}$ sur $n = 8$ bits
4. 111001101 00000000000000

S	Exposant								Mantisse																			
1	1	0	0	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
C				3				7				3				4				0				0				0	

Exercice

Encodage : Encoder le nombre +16,5 en simple précision.

Corrigé (Vide dans la version élève) :

– Encoder en binaire le nombre à virgule

$$+16,5 = (10000,1)_2$$

– Transformer en notation scientifique (souvenez-vous que décaler la virgule revient à multiplier/diviser par 2)

$$(10000,1)_2 = (1,00001 * 2^4)_2$$

– Identifier les champs
  – signe = 0 (positif)
  – exposant = 4
  – mantisse = 00001 (on ne prend pas le 1,)
– Encoder l'exposant 4 doit être codé sur 8 bits par excès de 127 => on encode $131 = (10000011)_2$
– Résultat final (En contrôle, écrire tous les 0 à la fin pour faire 32 bits)

0 10000011 00001000...

– Regroupement des bits par 4

0100 0001 1000 0100 0000 0000 ...

En Héxa (ne pas oublier les 0 à la fin pour faire 32 bits)

41 84 00 00

2.2 Procédure de conversion binaire (hexadécimale) en réel

On désire retrouver la valeur du nombre 44 *F*3 *E*0 00 représenté en virgule flottante.

On commence par placer les valeurs hexadécimales et leurs équivalents binaires dans la « structure » du flottant simple précision :

S	Exposant								Mantisse																			
0	1	0	0	0	1	0	0	1	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
4				4				F				3				E				0				0				0	

- Signe = 0 : Nombre positif
- Exposant décalé : $1000\ 1001_{(2)} = 137_{(10)}$ donc un exposant simple égal à $137 - 127 = 10_{(10)}$;
- Mantisse : (1,) 11100111110000000000000.

Comme l'exposant simple est égal à 10, on peut « dénormaliser » en décalant la virgule de 10 bits vers la droite puis rajouter le bit 1 invariant non stocké dans la mantisse, ce qui conduit à :

$$\underbrace{11110011111}_{\text{Partie entière 1951}} \quad , \quad \underbrace{0000000000000_{(2)}}_{Partiefractionnaire0} = 1951_{(10)}$$

Savoir

Savoir - Faire : Trouver la représentation en base dix d'un nombre à virgule flottante donné en binaire

On identifie le signe s, la mantisse m et l'exposant n ; on interprète chacun comme un nombre décimal en n'oubliant pas de tenir compte du décalage de 1023 pour l'exposant ; on calcule enfin la quantité $sm2^n$.

Pour un nombre codé en double précision. Si l'on note s le bit de signe, e1 . . . e11 les bits d'exposant et m1 . . .m52 les bits de la mantisse, on peut également donner l'expression directe suivante du nombre représenté :

$$(-1)^s * 2^{e1\ldots e11-1023} * \left(1 + \sum_{i=1}^{52} m_i \frac{1}{2^i}\right)$$

Exercice

Décodage : Décoder le nombre réel $(4024000000000000)_{(16)}$ *qui est en double précision (puisqu'il utilise 64 bits).*

Corrigé (Incomplet dans la version élève) :

- Analyser le nombre en binaire (on ne peut rien extraire de l'hexadécimale)

  0100 0000 0010 0100 0000000

- Identifier les champs
  - signe : 0 (positif)
  - exposant : 100 0000 0010 (11 bits en double précision)
  - mantisse : 010000. . .
- Décoder l'exposant On trouve un exposant affiché de $(10000000010)_2 = (1026)_{10}$ Mais il faut se rappeler que pour coder cet exposant, on lui avait ajouté 1023 (sur 11 bits, codage par excès de 1023). L'exposant réel vaut donc 3.

Exercice

– Écrire le nombre en binaire en notation scientifique

$$(1,01*2^3)_2$$

– En déduire le résultat final :

$$(1,01*2^3)_2 = (1010)_2 = (10)_{10}$$

(multiplier par 23 revient à décaler la virgule de 3 chiffres à droite en binaire).

Exercice

Trouver le nombre à virgule flottante représenté par le mot :

11000100011010010011110000111000000000000000000000000000000000000000.

Corrigé (Vide dans la version élève) :

Le signe est représenté par 1. L'exposant est représenté par 10001000110. La mantisse est représentée par 10010011110000111000000000000000000000000000000000000000. Le signe du nombre est -. Le nombre 10001000110 est égal à 1094 et l'exposant du nombre est donc n = 1094 - 1023 = 71. Sa mantisse est

$$m = 1,10010011110000111000000000000000000000000000000000000000$$

$$m = 1 + 1/2 + 1/2^4 + 1/2^7 + 1/2^8 + 1/2^9 + 1/2^{10} + 1/2^{15} + 1/2^{16} + 1/2^{17}$$

$$m = (2^{17} + 2^{16} + 2^{13} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^2 + 2 + 1)/2^{17}$$

$$m = \frac{206727}{131072}$$

Le nombre représenté est donc $-\frac{206727}{131072} * 2^{71} \approx -3,724...1021$.

Exercice

Compléter le tableau suivant :

Corrigé (Vide dans la version élève) :

Représentation 32 bits	Représent. Hexa	Valeur
0 10000000 10000000000000000000000	40400000	$(1,1 \times 2^{128-127})_{(2)} = (1,1 \times 2^1)_{(2)} = 11_{(2)}$ $3_{(10)}$
0 01111101 01000000000000000000000	3EA00000	$(1,01 \times 2^{125-127})_{(2)} = (1,01 \times 2^{-2})_{(2)}$ $0,0101_{(2)} = 0,3125_{(10)}$
0 10000110 00101111010000000000000	4317A000	$= (1,0010111101 \times 2^{134-127})_{(2)}$ $= (1,0010111101 \times 2^7)_{(2)}$ $= 10010111,101_{(2)}$ $= 151,625_{(10)}$
1 01111110 00000000000000000000000	BF000000	$-(1.0 \times 2^{126-127})_{(2)} = -(1.0 \times 2^{-1})_{(2)}$ $-(0.1)_{(2)} = -0.5_{(10)}$

3 Capacités de la représentation

<table>
<tr><td rowspan="4">Simple précision ou binary32)</td><td>Plus petit nombre positif normalisé</td></tr>
<tr><td>
<table>
<tr><th>S</th><th colspan="8">Exposant</th><th colspan="23">Mantisse</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">0</td><td colspan="2">0</td><td colspan="3">8</td><td colspan="3">0</td><td colspan="3">0</td><td colspan="3">0</td><td colspan="3">0</td><td colspan="3">0</td><td colspan="3">0</td></tr>
</table>
Exposant : 1 – 127 = – 126 ⇒ 1 x 2^{– 126} = 1,175 494 350 82 … x 10^{– 38}
</td></tr>
<tr><td>Plus grand nombre positif normalisé</td></tr>
<tr><td>
<table>
<tr><th>S</th><th colspan="8">Exposant</th><th colspan="23">Mantisse</th></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">7</td><td colspan="2">F</td><td colspan="3">7</td><td colspan="3">F</td><td colspan="3">F</td><td colspan="3">F</td><td colspan="3">F</td><td colspan="3">F</td></tr>
</table>
Exposant : 254 – 127 = 127 ⇒ (1 + 1 x 2^{– 1} + 1 x 2^{– 2} + … + 1 x 2^{– 22} + 1 x 2^{– 23}) x 2¹²⁷
= (2 – 2^{– 23}) x 2¹²⁷ = 3,402 823 466 38 … x 10³⁸
</td></tr>
<tr><td rowspan="4">Double précision ou binary64</td><td>Plus petit nombre positif normalisé</td></tr>
<tr><td>Exposant : 1 – 1023 = – 1022 ⇒ 1 x 2^{– 1022} = 2,225 073 858 51 … x 10^{– 308}</td></tr>
<tr><td>Plus grand nombre positif normalisé</td></tr>
<tr><td>Exposant : 2046 – 1023 = 1023 ⇒ (2 – 2^{– 52}) x 2¹⁰²³ = 1,797 693 134 86 … x 10³⁰⁸</td></tr>
</table>

En simple précision, l'exposant -127, codé 0000 0000, est réservé pour zéro et les nombres « non normalisés ». L'exposant 128 codé 1111 1111 est réservé pour coder $+\infty$ (ou $-\infty$ si signe négatif).

Références

[1] Christophe François, Représentation de l'information, représentation des nombres.

[2] Gaëtan Pruvost http://perso.limsi.fr/pruvost/res/teach-doc/ieee754.pdf.

[3] Manfred GILLI, METHODES NUMERIQUES, Département d'économétrie Université de Genève, 2006.