Représentation des nombres

Représentation des nombres

Plan de la phase

Introduction
Nombres en virgule fixe
Nombres en virgule flottante
Exercices

Représentation des nombres

Introduction

(note1)

Le choix du **mode de représentation** des nombres en machine est de la **responsabilité** du **programmeur**. En langage C, par exemple il a le choix du format des données (char, int, float...).

Comprendre la représentation des nombres permet de minimiser la place occupée sur les unités de stockage, la place occupée en mémoire et d'accroître la rapidité des traitements sur ces données.

La représentation des nombres se fait selon deux méthodes :

- virgule fixe
- virgule flottante

Représentation des nombres

Nombres en virgule fixe – mot

Les **systèmes** informatiques **manipulent** une longueur fixe de bits, appelé **mot**. Suivant la machine, la taille du mot peut être différente : 32, 64, 128 bits.

- Intel Pentium : mots de 32 bits.
- Athlon 64 : mots de 64 bits.
- Intrel Itanium : mots 128 bits.

- Quarté (nibble) : mot de 4 bits.
- Octet (byte) : mot de 8 bits
- Word : mot de 16 bits
- Double-word : mot de 32 bits

Représentation des nombres

Nombres en virgule fixe

Un nombre en virgule fixe :

- **Valeur** munie d'un signe ou pas.
- Enregistrée sous forme d'un **entier binaire** ou **BCD** (DCB Décimal Codée Binaire).
- La **virgule n'apparaît** pas dans le stockage du nombre, on parle de **virgule virtuelle**.
- C'est le **programmeur** qui **gère** la **virgule** dans le programme qui utilise ce nombre. ***(note 1)***

Représentation des nombres

Nombres en virgule fixe – Nombres entiers binaires

Comment représenter un nombres négatifs et son signe ?

Règles pour la représentation des données signées (*adoptées par les constructeurs*) :

Nombre stocké sur un octet

Signe		***Représentation du nombre***						
7	6	5	4	3	2	1	0	***Poids des bits***

- Si le bit de signe a la valeur **0**, la donnée est **positive**.
- Les valeurs **positives** sont représentées sous leur **forme binaire**.
- La valeur **zéro** est considérée comme une donnée **positive**.
- Si le bit de signe a la valeur **1**, la donnée est **négative**.
- Les valeurs **négatives** sont représentées sous leur forme **complément à deux** ou complément vrai.

Représentation des nombres

Nombres en virgule fixe – Nombres entiers binaires

Il existe deux modes de représentation en complément :***(note 1)***

- **Complément restreint** *(complément à 1)*

  On inverse chaque chiffre du nombre.

  Le nombre ***1 0 0 1 0***
  a pour **complément restreint** ***0 1 1 0 1***

- **Complément vrai** *(complément à 2)*

  On ajoute 1 au complément à 1

7	6	5	4	3	2	1	0
1	***1***	***1***	***1***	***0***	***1***	***1***	***1***
Signe	**Nombre**						

On désire coder la valeur -5 sur 8 bits. Il suffit :
- d'écrire 5 en binaire : ***00000101***
- de complémenter à 1 : ***11111010***
- d'ajouter 1 : ***11111011***
- la représentation binaire de -5 sur 8 bits est ***11111011***

Représentation des nombres

Nombres en virgule fixe – Nombres entiers binaires

Soustraction par la méthode du complément restreint (*Exemple 1)*

Soit à retrancher 28_{10} de 63_{10}

$63_{10} = (00111111)_2$

$28_{10} = (00011100)_2 \rightarrow$ complément restreint : $(11100011)_2$

$$
\begin{array}{lcl}
63_{10} & \rightarrow & (00111111)_2 \\
-\ 28_{10} & \rightarrow & +\ (11100011)_2 \\
\hline
35_{10} & & \\
 & & (00100010)_2 \\
 & & +\quad \mathbf{1} \\
\hline
 & & (00100011)_2
\end{array}
$$

Report (retenue généré **1** prise en compte)

Le résultat

Dans cette exemple, l'addition provoque une **retenue**, le résultat est un **nombre positif**

Représentation des nombres

Nombres en virgule fixe – Nombres entiers binaires

Soustraction par la méthode du complément restreint (*Exemple 2)*

Soit à retrancher 63_{10} de 28_{10}

$28_{10} = (00011100)_2$

$63_{10} = (00111111)_2 \rightarrow$ complément restreint : $(11000000)_2$

$$
\begin{array}{lcr}
28_{10} & \rightarrow & (00011100)_2 \\
-\ 63_{10} & \rightarrow & +\ (11000000)_2 \\
\hline
-\ 35_{10} & & \overline{(11011100)_2} \\
 & & \downarrow \text{complément restreint} \\
 & & (00100011)_2 = 35_{10}
\end{array}
$$

Le résultat, le nombre ***-35*** sous forme complémentée

Dans cette exemple, l'addition ne provoque pas de **retenue**, le résultat se présente comme un **nombre négatif** sous sa forme complémentée. Il suffit de prendre le complément restreint du nombre résultant de l'addition pour en connaître la valeur.

Représentation des nombres

Nombres en virgule fixe – Nombres entiers binaires

Soustraction par la méthode du complément vrai (*Exemple 1)*

Soit à retrancher 28_{10} de 63_{10}

$63_{10} = (00111111)_2$

$28_{10} = (00011100)_2 \rightarrow$ complément vrai : $(11100011)_2 + (1)_2 = (11100100)$

$$
\begin{array}{lcl}
63_{10} & \rightarrow & (00111111)_2 \\
-\ 28_{10} & \rightarrow & +\ (11100100)_2 \\
\hline
35_{10} & & \\
 & & \mathbf{1}\ \boxed{(00100011)_2}
\end{array}
$$

Le résultat

Représentation des nombres

Nombres en virgule fixe – Nombres entiers binaires

Soustraction par la méthode du complément vrai (*Exemple 2)*

Soit à retrancher 63_{10} de 28_{10}

$28_{10} = (00011100)_2$

$63_{10} = (00111111)_2 \rightarrow$ complément vrai: $(11000000)_2 + (1)_2 = (11000001)_2$

$$
\begin{array}{lcr}
28_{10} & \rightarrow & (00011100)_2 \\
-\ 63_{10} & \rightarrow & +\ (11000001)_2 \\
\hline
-\ 35_{10} & & (11011101)_2
\end{array}
$$

Le **MSB** est à **1**, donc c'est bien un nombre **négatif**.
Mais quel nombre décimal représente-t-il ?
Il faut calculer le complément vrai du résultat.

Le résultat

$$
\begin{array}{r}
(11011101)_2 \\
(00100010)_2 \\
+ \qquad 1 \\
\hline
(00100011)_2
\end{array}
\rightarrow 35_{10}
$$

Représentation des nombres

Nombres en virgule fixe – Nombres entiers binaires

Une **représentation signée** nécessite de **fixer** le nombre de bits des nombres à représenter : 8, 16, 32, etc.

Le choix du format (*nombre de bits*) entraîne deux conséquence :

- Le **nombre** de **valeurs** pour un **format** donné est **fixe**.
- Risque d'erreur de **débordement** (overflow)

Valeurs représentables :

- En 8 bits :
  - Nombres positifs : $(000\ 0000)_2$ à $(0111\ 1111)_2$ ou encore de 0_{10} à 127_{10}
  - Nombres négatifs : $(1000\ 0000)_2$ à $(1111\ 1111)_2$ ou encore de -128_{10} à -1_{10}
- En 16 bits :
  - Nombres positifs : 0000 à 7FFF = 0 à 32767
  - Nombres négatifs : 8000 à FFFF = -32768 à -1
- En **n bits** :
  - Nombres positifs : $\boldsymbol{0 \text{ à } 2^{n-1} - 1}$
  - Nombres négatifs : $\boldsymbol{-2^{n-1} \text{ à } -1}$

Faire démo :
- C : format.c
- Java : Depassement.java

Exemple de débordement en 8 bits :

$$\begin{array}{rcl} 127 & \longrightarrow & (01111111)_2 \\ +\ 2 & \longrightarrow & (\underline{00000010})_2 \\ \hline -127 & \longleftarrow & (10000001)_2 \end{array}$$

Représentation des nombres

Nombres en virgule fixe – DCB

DCB = Décimale Codée Binaire

Il existe deux formes de DCB :

- DCB **condensé**
- DCB **étendu**

En BCD chaque **chiffre** du nombre est **codé** sur **4 bits**

Chiffre	Bits	Chiffre	Bits
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

DCB condensé

- Chaque chiffre du nombre à coder occupe un quartet
- Le signe est généralement stocké dans le quartet le moins significatif
- Diverses méthodes de codification du signe :
  - Par exemple : B_{16} (1011_2)pour le signe + et D_{16} (1101_2)pour le signe -

Le nombre **+ *(341)*$_{10}$** en CDB condensé

0011	*0100*	*0001*	*1011*
3	4	1	+

Représentation des nombres

Nombres en virgule fixe – DCB

DCB étendu

- Chaque **chiffre** du nombre à coder **occupe** un **octet**.
- Les 4 bits de droites de l'octet sont suffisant pour coder les chiffres allant de 0 à 9.
- Les 4 bits gauche de l'octet, dits « bits de zone », ne servent à rien.
- C'est la partie zone de l'octet le moins significatif qui joue le rôle de signe.

Le nombre **+ *(341)*$_{10}$** en CDB étendu

0000	***0011***	***0000***	***0100***	***1011***	***0001***
zone	3	zone	4	+	1

Représentation des nombres

Nombres en virgule flottante

La représentation en virgule fixe occupe une place mémoire importante quand on utilise de grand nombres et on lui préfère une autre forme de représentation dite en virgule flottante (float).

La représentation des nombres réels en virgule flottante fait correspondre au nombre trois informations :

- le signe
- l'exposant
- la mantisse

Exemple 1 : 1 200 000 000 = 12 E 8

- mantisse = 12
- exposant = 8

Exemple 2 : -123.45

- on utilise un **exposant décalé**, décalage +5
- signe de l'exposant : + (10^{+3})
- exposant décalé : +8 (5+3)
- mantisse : 0.12345
- signe du nombre : -

Exemple 3 : +0.0000678

- on utilise un **exposant décalé**, décalage +5
- signe de l'exposant : - (10^{-4})
- exposant décalé : +1 (5-4)
- mantisse : 0.67800
- signe du nombre : +

Représentation des nombres

Nombres en virgule flottante

- L'exposant décalé permet d'éviter les exposants négatifs.

- On normalise la forme de la représentation en virgule flottante :
  - le chiffre à gauche de la marque décimale est un zéro (omis lors du stockage du nombre)
  - le chiffre à droite de la marque décimale n'est pas un zéro.

Exemple : 1234
- 0.01234E5 n'est pas normalisé
- 0,1234E4 est normalisé et représenté par, 1234E4

La **norme** la plus utilisée pour le **codage** des **réels** est la norme **IEEE 754 :**
- codage simple précision (*float en C*) sur 32 bits
- codage double précision (*double en C*) sur 64 bits
- codage précision étendue (*long double en C*) sur 80 bits
- l'exposant est codé en décalage :
  - +127 en simple précision
  - +1023 en double précision

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

Représentation des nombres

Nombres en virgule flottante

Exemple 1 : représentation du nombre décimal 6.625 en virgule flottante, format simple précision.

a. On convertit 6.625 en binaire : $(6{,}625)_{10} = (110.1010)_2$

b. On normalise

Dans la norme IEEE 754, en binaire, le décalage de la virgule se fait vers la gauche jusqu'à laisser 1 bit significatif.

$(110.1010)_2$ → $(1.101010)_2$ décalage de 2.

Le 1 significatif n'est pas noté dans la représentation.
La valeur de la **mantisse** est **$(101010)_2$** complété avec des 0 pour les autre bits de poids faibles.

c. L 'exposant

- exposant de référence : +2 (décalage *de la virgule*)
- norme IEEE 754, simple précision → décalage de +127

donc **l'exposant en excès** sera **129** (127 + 2), soit en binaire **$(10000001)_2$**

d. Le nombre est positif, le bit représentatif du signe sera positionné à 0.

$(6.625)_{10}$ =

S	Exposant								Mantisse																							
0	1	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4			0					D				4				0				0				0				0			

Représentation des nombres

Nombres en virgule flottante

Exemple 2 : comment retrouver la valeur décimale du nombre 44F3E000 représenté en virgule flottante, format simple précision.

a. On place les différentes valeurs dans la « structure » du flottant

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	0	0	0
4				4					F				3				E				0				0				0			

Faire démo :
- Java : Imprecision.java

b. Signe : le bit est à 0, c'est un nombre positif

c. Exposant : $(10001001)_2 = (137)_{10}$ donc un décalage de 10 par à 127.

d. Mantisse : .11100111110000000000000 → **1**1110011111.0000000000000

Bit significatif

Virgule repoussée de 10 rang vers la droite.

Soit en décimal : 1951.

Représentation des nombres

Exercices

1. Réaliser l'opération binaire 20_{10}- 30_{10} en utilisant les deux techniques :

a) du complément restreint
b) du complément vrai

2. Coder sous forme d'entier binaire (mot de 16 bits) le nombre : -357_{10}

3. Coder en DCB condensé a) 387_{10} b) 35.47_{10} c) -35.99_{10}

4. Coder en DCB étendu a) 3867_{10} b) 34.675_{10} c) -34.45_{10}

5. En utilisant le format virgule flottante IEEE 754 simple précision, coder le nombre: 27.75_{10}

6. Décoder le nombre IEEE 754 simple précision : 0x44FACC00