Architecture des ordinateurs

Représentation des nombres non entiers

Nombres fractionnels (1/2)

- Comment représenter un nombre comme $(0.3)_{10}$ en binaire ?
- Convertir 3 en binaire puis le placer après la virgule : **ne fonctionne pas** !
- $0.1_{10} = 1/10$
  $0.01_{10} = 1/100 = 1/10^2$
  $0.001_{10} = 1/1000 = 1/10^3, \ldots$

$$(0.a_{-1}a_{-2}\cdots a_{-m})_{10} = \sum_{-m \leq i < 0} a_i 10^i$$

Ceci est vrai dans une base b quelconque. On aura donc

$$(0.a_{-1}a_{-2}\cdots a_{-m})_b = \sum_{-m \leq i < 0} a_i b^i$$

Nombres fractionnels (2/2)

- Pour n'importe quel nombre avec une partie entière (de $n-1$ à 0) et une partie fractionnelle (de -1 à $-m$) :

$$a_{n-1} \cdots a_0 \,.\, a_{-1} \cdots a_{-m} = \sum_{-m \leq i < n} a_i b^i$$

- $(0.1)_2 = (1/2)_{10} = (0.5)_{10}$
  $(0.01)_2 = (1/4)_{10} = (0.25)_{10}$
  $(0.001)_2 = (1/8)_{10} = (0.125)_{10}$, ...

- On peut utiliser l'idée inverse pour faire une conversion...

Conversion de la partie fractionnelle (1/4)

- Du décimal vers le binaire

travail	représentation partielle	reste à faire
0.3	$0.\ldots$	0.3
$2 \times 0.3 = 0.6$	$0.0\ldots$	
$2 \times 0.6 = 1.2$	$0.01\ldots$	$0.3 - 1/4$
$2 \times 0.2 = 0.4$	$0.010\ldots$	
$2 \times 0.4 = 0.8$	$0.0100\ldots$	
$2 \times 0.8 = 1.6$	$0.01001\ldots$	$0.3 - 1/4 - 1/16$

$$(0.3)_{10} = (0.0\,1001\,1001\,1001\ldots)_2$$

Conversion de la partie fractionnelle (2/4)

- Du décimal vers le binaire

travail	représentation partielle	reste à faire
0.7	$0.\ldots$	0.7
$2 \times 0.7 = 1.4$	$0.1\ldots$	$0.7 - 1/2$
$2 \times 0.4 = 0.8$	$0.10\ldots$	
$2 \times 0.8 = 1.6$	$0.101\ldots$	$0.7 - 1/2 - 1/8$
$2 \times 0.6 = 1.2$	$0.1011\ldots$	$0.7 - 1/2 - 1/8 - 1/16$
$2 \times 0.2 = 0.4$	$0.10110\ldots$	

$(0.7)_{10} = (0.1\,0110\,0110\ldots)_2$

Conversion de la partie fractionnelle (3/4)

- Du binaire vers le décimal

$$
\begin{aligned}
n &= (0.101001)_2 \\
&= (0.1)_2 + (0.001)_2 + (0.000001)_2 \\
&= 2^{-1} + 2^{-3} + 2^{-6} \\
&= 1/2 + 1/8 + 1/64 \\
&= (0.5)_{10} + (0.125)_{10} + (0.015625)_{10} \\
&= (0.640625)_{10}
\end{aligned}
$$

- Inconvénient : autant de divisions que de bits...

Conversion de la partie fractionnelle (4/4)

- Du binaire vers le décimal

$$
\begin{aligned}
n &= (0.101001)_2 \\
&= (101001)_2/64_{10} \\
&= (51)_8/64_{10} \\
&= 41_{10}/64_{10} \\
&= (0.640625)_{10}
\end{aligned}
\qquad
\begin{aligned}
n &= (0.1010011)_2 \\
&= (101001100)_2/512_{10} \\
&= (514)_8/512_{10} \\
&= 332_{10}/512_{10} \\
&= (0.6484375)_{10}
\end{aligned}
$$

- Conversion de la base 2 vers la base 8 puis 10, puis une seule division.

Exercice

- Convertir $(0.9)_{10}$ en binaire
- Convertir $(1000100101.10011010010)_2$ en décimal

Représentation des nombres en virgule fixe

- N bits avant la virgule, M après
- Addition et multiplication simple
- On ne peut pas représenter à la fois de très petits nombres et de très grand nombres

=> solution : la virgule flottante

Représentation des nombres en virgule flottante

- En base 10 : notation scientifique

1.2345×10^0	$1.2345E0$	1.2345
1.2345×10^1	$1.2345E1$	12.345
1.2345×10^2	$1.2345E2$	123.45
1.2345×10^{10}	$1.2345E10$	12345000000.
1.2345×10^{-1}	$1.2345E{-1}$	0.12345
1.2345×10^{-2}	$1.2345E{-2}$	0.012345
1.2345×10^{-10}	$1.2345E{-10}$	0.00000000012345

- signe s, mantisse m et exposant e en base b
  Valeur : $(-1)^s \times m \times b^e$.
- multiplication : multiplier les mantisses et additionner les exposants
- addition : ramener au même exposant puis additionner

Virgule flottante en binaire

- signe s, mantisse m et exposant e
  Valeur : $(-1)^s \times m \times 2^e$.

flottant binaire	exposant décimal	valeur binaire	valeur décimale
$1.101_2 \times 2^0$	0	1.101_2	1.625_{10}
$1.101_2 \times 2^1$	1	11.01_2	3.25_{10}
$1.101_2 \times 2^{10_2}$	2	110.1_2	6.5_{10}
$1.101_2 \times 2^{-1}$	-1	0.1101_2	0.8125_{10}
$1.101_2 \times 2^{-10_2}$	-2	0.01101_2	0.40625_{10}
$1.101_2 \times 2^{1000_2}$	8	110100000_2	416_{10}
$1.101_2 \times 2^{-1000_2}$	-8	0.00000001101_2	0.00634765625_{10}

Virgule flottante dans la mémoire

- Plusieurs mots mémoire pour représenter un flottant
- Certains bits représentent la mantisse, d'autres l'exposant
- Rôle des bits en général fixé d'une manière spécifique au processeur
- norme IEEE 754 : La plus utilisée pour représenter les nombres en virgule flottante dans les ordinateurs.
  (Institute of Electrical and Electronics Engineers)

- flottant $\neq$ réel : la représentation avec une mantisse à m chiffres : $1.a_{-1} \cdots a_{-m}$ implique une approximation

La norme IEEE 754 (1/4)

- Fixe la représentation des nombres flottants sur 32 (simple précision) et 64 bits (double précision)
- Spécifie la manière d'arrondir le résultat des opérations non représentable de manière exacte
  et traite de quelques cas particuliers.
- Signe : 1 bit
- Exposant : 8 bits en excédent 127 (simple précision) ou 11 bits en excédent 1023 (double précision)
- Mantisse : 23 bits (simple précision)
  ou 52 (double précision)
- La mantisse appartient à l'intervalle $[1,0; 10,0[$ (en binaire)
- Le 1 à gauche n'est pas représenté
  (0 est donc un cas particulier)

La norme IEEE 754 (2/4)

Exemple : (1 10000000 01001000000000000000000)

Signe $(1)_2$
$\rightarrow$ nombre négatif

Exposant $(10000000)_2$
$\rightarrow 128_{10}$ en excédent 127
$\rightarrow 128 - 127 = 1.$

Mantisse $(01001000000000000000000)_2$
$\rightarrow (1.01001)_2 = 1 + 1/4 + 1/32 = 1 + 9/32 = 1.28125$

La valeur du nombre est donc

$$-1.28125 \times 2^1 = -2.5625$$

La norme IEEE 754 (2/4)

Exemple : (1 10000000 01001000000000000000000)

Signe $(1)_2$
$\rightarrow$ nombre négatif

Exposant $(10000000)_2$
$\rightarrow 128_{10}$ en excédent 127
$\rightarrow 128 - 127 = 1.$

Mantisse $(01001000000000000000000)_2$
$\rightarrow (1.01001)_2 = 1 + 1/4 + 1/32 = 1 + 9/32 = 1.28125$

$$\begin{aligned} \textit{valeur} &= -1 \times (1.01001)_2 \times 2^{(10000000)_2-(127)_{10}} \\ &= -(1 + 1/2^2 + 1/2^5) \times 2^{128-127} \\ &= -(2 + 1/2 + 1/2^4) \\ &= -2.5625 \end{aligned}$$

La norme IEEE 754 (3/4)

- Représentation de 21.78125 en simple précision
- On commence par convertir le nombre en binaire

$$21_{10} = 25_8 = 10101_2$$

$$\begin{array}{r|l}
0.78125 & 0.\ldots \\
2 \times 0.78125 = 1.5625 & 0.1\ldots \\
2 \times 0.56250 = 1.125 & 0.11\ldots \\
2 \times 0.125 = 0.25 & 0.110\ldots \\
2 \times 0.25 = 0.5 & 0.1100\ldots \\
2 \times 0.5 = 1.0 & 0.11001
\end{array}$$

$$\rightarrow (21.78125)_{10} = (10101.11001)_2$$

La norme IEEE 754 (3/4)

- Représentation de 21.78125 en simple précision
- On commence par convertir le nombre en binaire
- $(21.78125)_{10} = (10101.11001)_2$

La norme IEEE 754 (3/4)

- Représentation de 21.78125 en simple précision
- On commence par convertir le nombre en binaire
- $(21.78125)_{10} = (10101.11001)_2$
- Le nombre est positif, bit de signe $(0)_2$

La norme IEEE 754 (3/4)

- Représentation de 21.78125 en simple précision
- On commence par convertir le nombre en binaire
- $(21.78125)_{10} = (10101.11001)_2$
- Le nombre est positif, bit de signe $(0)_2$
- On calcule la valeur de l'exposant

$$\begin{aligned}(10101.11001)_2 &= (10101.11001)_2 \times (2^0)_{10} \\ &= (1.010111001)_2 \times (2^4)_{10}\end{aligned}$$

8 bits en excédent 127 :

$$127 + 4_{10} = 131_{10} = (10000011)_2$$

La norme IEEE 754 (3/4)

- Représentation de 21.78125 en simple précision
- On commence par convertir le nombre en binaire
- $(21.78125)_{10} = (10101.11001)_2$
- Le nombre est positif, bit de signe $(0)_2$
- On calcule la valeur de l'exposant

$$\begin{aligned}(10101.11001)_2 &= (10101.11001)_2 \times (2^0)_{10} \\ &= (1.010111001)_2 \times (2^4)_{10}\end{aligned}$$

$$\text{8 bits en excédent 127 :}$$

$$127 + 4_{10} = 131_{10} = (10000011)_2$$

- La mantisse est donc $(01011100100000000000000)_2$

La norme IEEE 754 (3/4)

- Représentation de 21.78125 en simple précision
- On commence par convertir le nombre en binaire
- $(21.78125)_{10} = (10101.11001)_2$
- Le nombre est positif, bit de signe $(0)_2$
- On calcule la valeur de l'exposant

$$\begin{aligned}(10101.11001)_2 &= (10101.11001)_2 \times (2^0)_{10} \\ &= (1.010111001)_2 \times (2^4)_{10}\end{aligned}$$

8 bits en excédent 127 :

$$127 + 4_{10} = 131_{10} = (10000011)_2$$

- La mantisse est donc $(01011100100000000000000)_2$
- Le nombre est représenté par :
  0 10000011 01011100100000000000000

La norme IEEE 754 (4/4)

- Cas particuliers : exposant avec tous les bits à 1 ou à 0
- Bits de l'exposant à 0 et mantisse à 0 : nombre 0
  (deux représentations en fonction du bit de signe)
- Bits de l'exposant à 0 et mantisse $\neq 0$:
  représentation *dénormalisée* d'un nombre (très petit)
  exposant : -126 (-1022 en double precision)
  nombre à gauche de la virgule : 0
- Bits de l'exposant à 1 et mantisse à 0 : $+\infty$ ou $-\infty$
  débordement de capacité (`Inf`)
- Bits de l'exposant à 1 et mantisse $\neq 0$:
  ce n'est pas un nombre, exemple : $\sqrt{-2}$ (`NaN`)

Exercice

- Donnez la représentation de -6.843750_{10} avec la norme IEEE754 en simple précision
- Donnez la valeur décimale du flottant en simple précision 01010000100110111000000000000000 représenté avec la norme IEEE754

Supplément

- Commande `bc`
- taille des nombres limitée que par celle de la mémoire disponible
- scale : nombre de chiffres après la virgule
- obase : base de sortie (ex : obase=2)
- ibase : base d'entrée
- Ctrl+D : pour sortir