Révisions 3: Représentation des nombres en machine

(Révisions MPSI)

Lycée Montaigne

Septembre 2019

Plan

1. Représentation des entiers naturels
   - Pour l'homme...
     - La base 10
     - Les autres bases
   - ...et pour la machine : la base 2 obligatoire ! ! !
     - Principe
     - Conversion
2. Représentation des entiers relatifs
   - Notation en complément à deux
   - Le problème du dépassement de capacité - cas particulier de Python
3. Représentation des réels
   - La notation scientifique décimale
     - Nécessité des "flottants" en machine
     - Décomposition en notation scientifique décimale
   - Décomposition en base binaire
     - Principe
     - Ecriture normalisée
   - Norme IEEE754 - limitation des nombres en virgule flottante
   - Exemples de représentation en mémoire et conversions (partie optionnelle !)

Pour l'homme...

La base 10

Le principe de décomposition d'un entier en base 10 est d'écrire le nombre en une somme d'autant de puissances de 10 que nécessaire. Par exemple :

Pour l'homme...

La base 10

Le principe de décomposition d'un entier en base 10 est d'écrire le nombre en une somme d'autant de puissances de 10 que nécessaire. Par exemple :

$$6753 = 3.10^{\mathbf{0}} + 5.10^{\mathbf{1}} + 7.10^{\mathbf{2}} + 6.10^{\mathbf{3}}$$

Pour l'homme...

La base 10

Le principe de décomposition d'un entier en base 10 est d'écrire le nombre en une somme d'autant de puissances de 10 que nécessaire. Par exemple :

$$6753 = 3.10^{\mathbf{0}} + 5.10^{\mathbf{1}} + 7.10^{\mathbf{2}} + 6.10^{\mathbf{3}}$$

Généralisant à tout nombre entier naturel n :

Pour l'homme...

La base 10

Le principe de décomposition d'un entier en base 10 est d'écrire le nombre en une somme d'autant de puissances de 10 que nécessaire. Par exemple :

$$6753 = 3.10^{\mathbf{0}} + 5.10^{\mathbf{1}} + 7.10^{\mathbf{2}} + 6.10^{\mathbf{3}}$$

Généralisant à tout nombre entier naturel n :

$$\boxed{n = \left(\underset{\substack{\uparrow \\ \text{poids le plus fort}}}{d_{n_d}} \quad \ldots d_1 \quad \underset{\substack{\uparrow \\ \text{poids le plus faible}}}{d_0} \right)_{10} = \sum_{k=0}^{k=n_d} d_k \cdot 10^k}$$

avec $d_k \in \{0, 1, 2, \ldots, 9\}$

Pour l'homme

La base 10

Exercice n°1:

Interpréter le fonctionnement du script python suivant :

```python
def Chiffre (n) :
    """n est un entier naturel donnÃ© par son ecriture decimale"""
    c=[]
    while n!=0 :
        c.append(n%10)
        n//=10
    c.reverse() # inverse l'ordre des elements d'une liste
    return c
print Chiffre (4587)
```

Pour l'homme

Les autres bases

GÉNÉRALISATION : un nombre en base k peut s'écrire :

$$(d_{n_k} \ldots, d_1, d_0)_k = d_0 \cdot k^0 + d_1 \cdot k^1 + \ldots + d_{n_k} \cdot k^{n_k}$$

REMARQUE : la conversion en base décimale est immédiate en réalisant à la main la somme décomposée précédente. Voyons cela sur l'exemple du nombre suivant exprimé en base 5 :

$$(204003)_5 = 3 \times 5^0 + 0 \times 5^1 + 0 \times 5^2 + 4 \times 5^3 + 0 \times 5^4 + 2 \times 5^5 = 6753$$

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Les puces mémoires/processeurs sont constituées de pattes, chacune pouvant être "au chiffre" **0** ou **1**. On appelle cela un **bit** pour l'anglais **BI**nary digi**T**.

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Les puces mémoires/processeurs sont constituées de pattes, chacune pouvant être "au chiffre" **0** ou **1**. On appelle cela un **bit** pour l'anglais **BI**nary digi**T**.
$\Longrightarrow$ **les machines travaillent en base 2 uniquement**

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Les puces mémoires/processeurs sont constituées de pattes, chacune pouvant être "au chiffre" **0** ou **1**. On appelle cela un **bit** pour l'anglais **BI**nary digi**T**.
$\Longrightarrow$ **les machines travaillent en base 2 uniquement**

<u>QUESTION :</u> Combien d'états et donc de valeurs peut recevoir une puce constituée de n pattes ?

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Les puces mémoires/processeurs sont constituées de pattes, chacune pouvant être "au chiffre" **0** ou **1**. On appelle cela un **bit** pour l'anglais **BI**nary digi**T**.
$\Longrightarrow$ **les machines travaillent en base 2 uniquement**

QUESTION : Combien d'états et donc de valeurs peut recevoir une puce constituée de n pattes ?

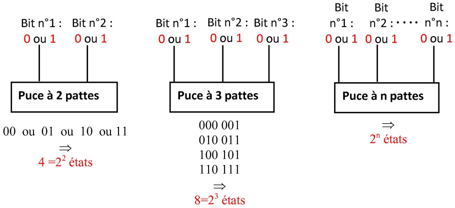

FIGURE: Etats possibles d'une puce à n pattes

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Selon le même principe qu'en base 10, un entier naturel n s'écrit en base 2 :

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Selon le même principe qu'en base 10, un entier naturel n s'écrit en base 2 :

$$\boxed{n = (d_{n_b} \dots d_1 d_0)_2 = \sum_{k=0}^{k=n_b} d_k \cdot 2^k} \qquad \text{avec } d_k \in \{0, 1\}$$

IMPORTANT :

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Selon le même principe qu'en base 10, un entier naturel n s'écrit en base 2 :

$$\boxed{n = (d_{n_b} \dots d_1 d_0)_2 = \sum_{k=0}^{k=n_b} d_k \cdot 2^k} \qquad \text{avec } d_k \in \{0, 1\}$$

<u>IMPORTANT :</u>

- Souvent regroupement par lots de 8 bits ≡ **octet**.

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Selon le même principe qu'en base 10, un entier naturel n s'écrit en base 2 :

$$\boxed{n = (d_{n_b} \dots d_1 d_0)_2 = \sum_{k=0}^{k=n_b} d_k \cdot 2^k} \qquad \text{avec } d_k \in \{0,1\}$$

IMPORTANT :

- Souvent regroupement par lots de 8 bits ≡ **octet**.
- Les nombres informatiques sont ainsi exprimés sur 1, 2, 4, ou encore 8 octets, soit respectivement 8 bits, 16 bits, 32 bits ou 64 bits.

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Selon le même principe qu'en base 10, un entier naturel n s'écrit en base 2 :

$$\boxed{n = (d_{n_b} \dots d_1 d_0)_2 = \sum_{k=0}^{k=n_b} d_k \cdot 2^k} \qquad \text{avec } d_k \in \{0,1\}$$

IMPORTANT :

- Souvent regroupement par lots de 8 bits ≡ **octet**.
- Les nombres informatiques sont ainsi exprimés sur 1, 2, 4, ou encore 8 octets, soit respectivement 8 bits, 16 bits, 32 bits ou 64 bits.
- Une profondeur d'écriture binaire de N *bits* permet de représenter les nombres entiers en base 10 de l'intervalle $\left[0..2^N - 1\right]$

...et pour la machine : la base 2 obligatoire ! ! !

Principe

Intervalles de codages des entiers naturels en fonction du nombre d'octets réservés

Profondeur en octets	intervalle de codage entiers base 2			intervalle de codage entiers base 10		
1 (8 *bits*)	00000000	$\longrightarrow$	11111111	0	$\longrightarrow$	255
2 (16 *bits*)	00000000 00000000	$\longrightarrow$	11111111 11111111	0	$\longrightarrow$	65535
4 (32 *bits*)	00000000 00000000 00000000 00000000	$\longrightarrow$	11111111 11111111 11111111 11111111	0	$\longrightarrow$	4294967295

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

<u>QUESTION</u> : comment représenter en base 2 format 16 bits un nombre entier naturel donné en base 10 ?

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

<u>QUESTION :</u> comment représenter en base 2 format 16 bits un nombre entier naturel donné en base 10 ?

<u>RÉPONSE :</u> 2 méthodes : la méthode dite «par soustraction» et la méthode dite «par division» (entière). On donne ici le principe de la seconde :

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

QUESTION : comment représenter en base 2 format 16 bits un nombre entier naturel donné en base 10 ?

RÉPONSE : 2 méthodes : la méthode dite «par soustraction» et la méthode dite «par division» (entière). On donne ici le principe de la seconde :

On souhaite décomposer le nombre n_{10} en base 2. Pour cela :

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

<u>QUESTION</u> : comment représenter en base 2 format 16 bits un nombre entier naturel donné en base 10 ?

<u>RÉPONSE</u> : 2 méthodes : la méthode dite «par soustraction» et la méthode dite «par division» (entière). On donne ici le principe de la seconde :

On souhaite décomposer le nombre n_{10} en base 2. Pour cela :

- Division entière de n_{10} par 2. Si le quotient n'est pas nul, la valeur du reste (0 ou 1) correspond au 1er bit : 0 ou 1.

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

QUESTION : comment représenter en base 2 format 16 bits un nombre entier naturel donné en base 10 ?

RÉPONSE : 2 méthodes : la méthode dite «par soustraction» et la méthode dite «par division» (entière). On donne ici le principe de la seconde :

On souhaite décomposer le nombre n_{10} en base 2. Pour cela :

- Division entière de n_{10} par 2. Si le quotient n'est pas nul, la valeur du reste (0 ou 1) correspond au 1er bit : 0 ou 1.
- Division entière du précédent quotient par 2. De même si le quotient n'est pas nul, la valeur du reste (0 ou 1) correspond au 2ème bit.

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

QUESTION : comment représenter en base 2 format 16 bits un nombre entier naturel donné en base 10 ?

RÉPONSE : 2 méthodes : la méthode dite «par soustraction» et la méthode dite «par division» (entière). On donne ici le principe de la seconde :

On souhaite décomposer le nombre n_{10} en base 2. Pour cela :

- Division entière de n_{10} par 2. Si le quotient n'est pas nul, la valeur du reste (0 ou 1) correspond au 1^er^ bit : 0 ou 1.
- Division entière du précédent quotient par 2. De même si le quotient n'est pas nul, la valeur du reste (0 ou 1) correspond au 2^ème^ bit.
- Ainsi de suite jusqu'au bit de poids le plus fort.

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

PREMIER EXEMPLE SIMPLE : cherchons à décomposer 24_{10} en base 2 par division entières successives :

:

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

PREMIER EXEMPLE SIMPLE : cherchons à décomposer 24_{10} en base 2 par division entières successives :

$$24 = (12 \times 2) + 0$$

:

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

Premier exemple simple : cherchons à décomposer 24_{10} en base 2 par division entières successives :

$$24 = (12 \times 2) + 0$$

$$= ((6 \times 2 + 0) \times 2) + 0$$

$$\vdots$$

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

PREMIER EXEMPLE SIMPLE : cherchons à décomposer 24_{10} en base 2 par division entières successives :

$$24 = (12 \times 2) + 0$$

$$= ((6 \times 2 + 0) \times 2) + 0$$

$$= (((3 \times 2 + 0) \times 2 + 0) \times 2) + 0$$

$$\vdots$$

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

Premier exemple simple : cherchons à décomposer 24_{10} en base 2 par division entières successives :

$$24 = (12 \times 2) + 0$$

$$= ((6 \times 2 + 0) \times 2) + 0$$

$$= (((3 \times 2 + 0) \times 2 + 0) \times 2) + 0$$

$$= ((((1 \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2) + 0$$

:

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

PREMIER EXEMPLE SIMPLE : cherchons à décomposer 24_{10} en base 2 par division entières successives :

$$24 = (12 \times 2) + 0$$

$$= ((6 \times 2 + 0) \times 2) + 0$$

$$= (((3 \times 2 + 0) \times 2 + 0) \times 2) + 0$$

$$= ((((1 \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2) + 0$$

et finalement ... : $= (((((0 \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2) + 0$

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

PREMIER EXEMPLE SIMPLE : cherchons à décomposer 24_{10} en base 2 par division entières successives :

$$24 = (12 \times 2) + 0$$

$$= ((6 \times 2 + 0) \times 2) + 0$$

$$= (((3 \times 2 + 0) \times 2 + 0) \times 2) + 0$$

$$= ((((1 \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2) + 0$$

$$= (((((0 \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2) + 0$$

et finalement ... :

$$= \mathbf{0} \times 2^5 + \mathbf{1} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{0} \times 2^0$$

Ainsi la représentation binaire de $(24)_{10}$ est donc : $(11000)_2$

...et pour la machine : la base 2 obligatoire ! ! !

Conversion

<u>SECOND EXEMPLE :</u> écriture en base 2 du nombre entier 1515 - présentation commode

1515	2										
1	757	2									
	1	378	2								
		0	189	2							
			1	94	2						
				0	47	2					
					1	23	2				
						1	11	2			
							1	5	2		
								1	2	2	
									0	1	2
										1	0

La représentation binaire de $(1515)_{10}$ est donc : $\boxed{(10111101011)_2}$

Notation en complément à deux

Première idée d'écriture des nombres entiers relatifs :

Notation en complément à deux

Première idée d'écriture des nombres entiers relatifs :

- Ecriture de la valeur absolue soit un entier naturel sur une certaine profondeur,*par exemple 7 bits ou 15 bits*

Notation en complément à deux

Première idée d'écriture des nombres entiers relatifs :

- Ecriture de la valeur absolue soit un entier naturel sur une certaine profondeur,*par exemple 7 bits ou 15 bits*
- bit réservé pour le signe, pour obtenir dans notre exemple *8 bits ou 16 bits*
  Pour un codage sur N bits, le bit réservé pour le signe S est d_{N-1} avec :

$$S = (-1)^{d_{N-1}} \Longrightarrow \begin{cases} S = +1 \ (n \text{ positif}) \text{ si } d_{N-1} = 0 \\ S = -1 \ (n \text{ négatif}) \text{ si } d_{N-1} = 1 \end{cases}$$

Notation en complément à deux

Première idée d'écriture des nombres entiers relatifs :

- Ecriture de la valeur absolue soit un entier naturel sur une certaine profondeur,*par exemple 7 bits ou 15 bits*
- bit réservé pour le signe, pour obtenir dans notre exemple *8 bits ou 16 bits* Pour un codage sur N bits, le bit réservé pour le signe S est d_{N-1} avec :

$$S = (-1)^{d_{N-1}} \Longrightarrow \begin{cases} S = +1 \ (n \text{ positif}) \text{ si } d_{N-1} = 0 \\ S = -1 \ (n \text{ négatif}) \text{ si } d_{N-1} = 1 \end{cases}$$

- Donc représentation binaire de n :

$$\boxed{n = (-1)^{d_{N-1}} \sum_{k=0}^{N-2} d_{n_b} 2^k}$$

Notation en complément à deux

Première idée d'écriture des nombres entiers relatifs :

- Ecriture de la valeur absolue soit un entier naturel sur une certaine profondeur,*par exemple 7 bits ou 15 bits*
- bit réservé pour le signe, pour obtenir dans notre exemple *8 bits ou 16 bits*
  Pour un codage sur N bits, le bit réservé pour le signe S est d_{N-1} avec :

$$S = (-1)^{d_{N-1}} \Longrightarrow \begin{cases} S = +1 \ (n \text{ positif}) \text{ si } d_{N-1} = 0 \\ S = -1 \ (n \text{ négatif}) \text{ si } d_{N-1} = 1 \end{cases}$$

- Donc représentation binaire de n :

$$\boxed{n = (-1)^{d_{N-1}} \sum_{k=0}^{N-2} d_{n_b} 2^k}$$

- Afin d'être distingué d'un entier naturel, un entier relatif est parfois représenté avec le bit de signe **souligné** :

$$(\underline{1}0010011)_2 = (-19)_{10}$$

Notation en complément à deux

Cette convention de représentation présente cependant deux inconvénients majeurs (par exemple en 8 bits) :

Notation en complément à deux

Cette convention de représentation présente cependant deux inconvénients majeurs (par exemple en 8 bits) :

- présence de deux zéros : $(\underline{0}0000000)_2 = (0)_{10}$ et $(\underline{1}0000000)_2 = (-0)_{10}$

Notation en complément à deux

Cette convention de représentation présente cependant deux inconvénients majeurs (par exemple en 8 bits) :

- présence de deux zéros : $(\underline{0}0000000)_2 = (0)_{10}$ et $(\underline{1}0000000)_2 = (-0)_{10}$
- la somme en binaire ne fonctionne plus, avec par exemple :

$$\underbrace{(\underline{0}0000011)_2}_{=(+3)_{10}} + \underbrace{(\underline{1}0000010)_2}_{=(-2)_{10}} = \underbrace{(\underline{1}0000101)_2}_{=(-5)_{10} \neq 1}$$

Notation en complément à deux

Cette convention de représentation présente cependant deux inconvénients majeurs (par exemple en 8 bits) :

- présence de deux zéros : $(\underline{0}0000000)_2 = (0)_{10}$ et $(\underline{1}0000000)_2 = (-0)_{10}$
- la somme en binaire ne fonctionne plus, avec par exemple :

$$\underbrace{(\underline{0}0000011)_2}_{=(+3)_{10}} + \underbrace{(\underline{1}0000010)_2}_{=(-2)_{10}} = \underbrace{(\underline{1}0000101)_2}_{=(-5)_{10} \neq 1}$$

<u>CONCLUSION :</u>

Notation en complément à deux

Cette convention de représentation présente cependant deux inconvénients majeurs (par exemple en 8 bits) :

- présence de deux zéros : $(\underline{0}0000000)_2 = (0)_{10}$ et $(\underline{1}0000000)_2 = (-0)_{10}$
- la somme en binaire ne fonctionne plus, avec par exemple :

$$\underbrace{(\underline{0}0000011)_2}_{=(+3)_{10}} + \underbrace{(\underline{1}0000010)_2}_{=(-2)_{10}} = \underbrace{(\underline{1}0000101)_2}_{=(-5)_{10} \neq 1}$$

<u>CONCLUSION :</u> **Convention de représentation inadaptée ! ! !**

Notation en complément à deux

IDÉE : On garde la même notation pour les entiers positifs, et on adopte la notation dite en **complément à deux** pour les entiers négatifs. Le principe est le suivant, on veut coder un entier relatif négatif n :

Notation en complément à deux

IDÉE : On garde la même notation pour les entiers positifs, et on adopte la notation dite en **complément à deux** pour les entiers négatifs. Le principe est le suivant, on veut coder un entier relatif négatif n :

- On garde le N^ième bit d_{N-1} pour le signe et on prend les $N-1$ bits restants qui représentent la valeur absolue du nombre

Notation en complément à deux

IDÉE : On garde la même notation pour les entiers positifs, et on adopte la notation dite en **complément à deux** pour les entiers négatifs. Le principe est le suivant, on veut coder un entier relatif négatif n :

- On garde le N$^{\text{ième}}$ bit d_{N-1} pour le signe et on prend les $N-1$ bits restants qui représentent la valeur absolue du nombre
- On inverse les $N-1$ bits de la valeur absolue du nombre (opération booléenne NON). Cette opération s'appelle le complément à 1.

Notation en complément à deux

IDÉE : On garde la même notation pour les entiers positifs, et on adopte la notation dite en **complément à deux** pour les entiers négatifs. Le principe est le suivant, on veut coder un entier relatif négatif n :

- On garde le N^ième bit d_{N-1} pour le signe et on prend les $N-1$ bits restants qui représentent la valeur absolue du nombre
- On inverse les $N-1$ bits de la valeur absolue du nombre (opération booléenne NON). Cette opération s'appelle le complément à 1.
- On ajoute le $N^{\text{ième}}$ bit de signe.

Notation en complément à deux

IDÉE : On garde la même notation pour les entiers positifs, et on adopte la notation dite en **complément à deux** pour les entiers négatifs. Le principe est le suivant, on veut coder un entier relatif négatif n :

- On garde le N$^{\text{ième}}$ bit d_{N-1} pour le signe et on prend les $N-1$ bits restants qui représentent la valeur absolue du nombre
- On inverse les $N-1$ bits de la valeur absolue du nombre (opération booléenne NON). Cette opération s'appelle le complément à 1.
- On ajoute le $N^{\text{ième}}$ bit de signe.
- On ajoute 1 à ce résultat. c'est le complément à 2.

Notation en complément à deux

<u>EXEMPLE :</u>
Codons par exemple $n = (-7)_{10}$ avec cette méthode sur 8 bits :

Notation en complément à deux

EXEMPLE :

Codons par exemple $n = (-7)_{10}$ avec cette méthode sur 8 bits :

- Valeur absolue $\longrightarrow |n|_{10} = 7_{10} = (0000111)_2$

Notation en complément à deux

EXEMPLE :

Codons par exemple $n = (-7)_{10}$ avec cette méthode sur 8 bits :

- Valeur absolue $\longrightarrow |n|_{10} = 7_{10} = (0000111)_2$
- Complément à 1 $\longrightarrow \overline{|n|} = (1111000)_2$

Notation en complément à deux

EXEMPLE :

Codons par exemple $n = (-7)_{10}$ avec cette méthode sur 8 bits :

- Valeur absolue $\longrightarrow |n|_{10} = 7_{10} = (0000111)_2$
- Complément à 1 $\longrightarrow \overline{|n|} = (1111000)_2$
- Bit de signe $\longrightarrow (11111000)_2$

Notation en complément à deux

EXEMPLE :

Codons par exemple $n = (-7)_{10}$ avec cette méthode sur 8 bits :

- Valeur absolue $\longrightarrow |n|_{10} = 7_{10} = (0000111)_2$
- Complément à 1 $\longrightarrow \overline{|n|} = (1111000)_2$
- Bit de signe $\longrightarrow (11111000)_2$
- Complément à 2 $\longrightarrow (-7)_2 = (11111001)_2$

Notation en complément à deux

REMARQUE : les défauts de la précédente méthode de codage sont éliminés :

Notation en complément à deux

<u>REMARQUE :</u> les défauts de la précédente méthode de codage sont éliminés :

- codage de zéro (toujours sur 8 bits)

$$\begin{cases}(+0)_{10} = (00000000)_2 \xrightarrow{\text{inv 7 bits}} (01111111)_2 \xrightarrow{+1} (10000000)_2 = (-0)_{10} \\ (-0)_{10} = (10000000)_2 \xrightarrow{\text{inv 7 bits}} (11111111)_2 \xrightarrow{+1} (00000000)_2 = (+0)_{10}\end{cases}$$

Notation en complément à deux

REMARQUE : les défauts de la précédente méthode de codage sont éliminés :

- codage de zéro (toujours sur 8 bits)

$$\begin{cases} (+0)_{10} = (00000000)_2 \xrightarrow{\text{inv 7 bits}} (01111111)_2 \xrightarrow{+1} (10000000)_2 = (-0)_{10} \\ (-0)_{10} = (10000000)_2 \xrightarrow{\text{inv 7 bits}} (11111111)_2 \xrightarrow{+1} (00000000)_2 = (+0)_{10} \end{cases}$$

- $(+3)_{10} + (-2)_{10} = (1)_{10}$ soit en binaire avec $(+3)_{10} = (00000011)_2$ et $(-2)_{10} = (11111110)_2$

$$\begin{array}{cc} & (00000011)_2 \\ + & (11111110)_2 \\ = & \underbrace{(00000001)_2}_{=(+1)_{10}} \end{array}$$

Notation en complément à deux

CONCLUSION : ainsi donc pour un nombre binaire codés sur N bits, on peut coder :

$$\begin{array}{lccc} 2^{N-1} & \text{nombres positifs} & \longrightarrow & n \in [0, 2^{N-1} - 1] \\ 2^{N-1} & \text{nombres négatifs} & \longrightarrow & n \in [-2^{N-1}, -1] \end{array}$$

Intervalle totale de représentation : $\boxed{n \in [-2^{N-1}; 2^{N-1} - 1]}$

Total de $2^{N-1} + \underbrace{1}_{\text{pour } 0} + 2^{N-1} - 1 = \boxed{2^N \text{ valeurs}}$

Donc par exemple pour un codage sur 8 bits, on peut représenter l'intervalle d'entiers relatifs : $[-128; +127]$ soit 256 valeurs comme attendu.

Notation en complément à deux

Intervalles de codages des entiers relatifs en fonction du nombre d'octets réservés

Profondeur (octets	intervalle de codage entiers base 2			intervalle de codage entiers base 10		
1 (8 *bits*)	10000000	$\longrightarrow$	01111111	-128	$\longrightarrow$	127
2 (16 *bits*)	10000000 00000000	$\longrightarrow$	01111111 11111111	-32768	$\longrightarrow$	32767
4 (32 *bits*)	10000000 00000000 00000000 00000000	$\longrightarrow$	01111111 11111111 11111111 11111111	-2147483648	$\longrightarrow$	2147483647

Le problème du dépassement de capacité - cas particulier de Python

<u>QUESTION (LÉGITIME !!!) :</u> que se passe-t-il lorsque l'on demande à une machine (via un langage) de traiter des nombres plus grand que le dimensionnement prévu pour les variables ?

<u>RÉPONSE :</u>

Le problème du dépassement de capacité - cas particulier de Python

QUESTION (LÉGITIME !!!) : que se passe-t-il lorsque l'on demande à une machine (via un langage) de traiter des nombres plus grand que le dimensionnement prévu pour les variables ?
RÉPONSE :

- Pour certains langage : déclaration préalable du **type** de chacune des variables employées (et de leur "longueur") en préambule de tout programme. Ex : Pascal ou du VisualBasic, Fortran.

Le problème du dépassement de capacité - cas particulier de Python

QUESTION (LÉGITIME ! ! !) : que se passe-t-il lorsque l'on demande à une machine (via un langage) de traiter des nombres plus grand que le dimensionnement prévu pour les variables ?
RÉPONSE :

- Pour certains langage : déclaration préalable du **type** de chacune des variables employées (et de leur "longueur") en préambule de tout programme. Ex : Pascal ou du VisualBasic, Fortran.
- Si dépassement par rapport à la déclaration : erreur de capacité de type *overflow* (Pascal), ou plus grave : erreur de calcul non signalée par "écrasement" mémoire sans précaution ! ! !) (Fortran 77)

Exercice n°2:

Tester la "profondeur" mémoire de votre calculatrice en insérant au clavier le plus grand nombre binaire entier positif[a]

---

a. sur les modèles TI nspire CX/CX CAS, il suffit par exemple d'inscrire "0b" comme préfixe à toute écriture de nombre binaire. C'est le cas sur bon nombre de calculatrices.

Le problème du dépassement de capacité - cas particulier de Python

Avec Python :

Le problème du dépassement de capacité - cas particulier de Python

Avec Python : aucune limite de profondeur fixée ! ! !

Le problème du dépassement de capacité - cas particulier de Python

Avec Python : aucune limite de profondeur fixée ! ! !

PRINCIPE :

Lorsque l'on tente de manipuler un nombre plus grand que la profondeur maximale prévu (32 ou 64 bits suivant les machines et les systèmes d'exploitation), Python découpe l'entier en lots de 15 bits et place les fragments dans un tableau d'entier chacun de largeur 16 bits. Ce tableau contient donc le codage du nombre en base 2^{15} ; on ajoute également dans ce tableau un entier codé sur 16 bits indiquant le nombre de fragments et dont le signe est celui du nombre codé.

Le problème du dépassement de capacité - cas particulier de Python

Lorsque Python passe dans ce type "propriétaire" de représentation des nombres, il le signale par l'écriture d'un suffixe "*L*" pour *Large Integer* à la suite du nombre. Par exemple en se plaçant à la limite de codage des entiers relatifs en 32 bits, soit :

$$(01111111111111111111111111111111)_2 = (2147483647)_{10}$$

et en ajoutant 1 :

Le problème du dépassement de capacité - cas particulier de Python

Lorsque Python passe dans ce type "propriétaire" de représentation des nombres, il le signale par l'écriture d'un suffixe "*L*" pour *Large Integer* à la suite du nombre. Par exemple en se plaçant à la limite de codage des entiers relatifs en 32 bits, soit :

$$(01111111111111111111111111111111)_2 = (2147483647)_{10}$$

et en ajoutant 1 :

```
>>> 2147483647+1
2147483648L
>>>
```

Le problème du dépassement de capacité - cas particulier de Python

REMARQUE :
La notation des entiers négatifs en complément à 2 **nécessite de connaître la profondeur en bits disponible sur la machine**, puisque l'on devra inscrire obligatoirement 1 sur le bit de signe qui se trouve **complètement à gauche**.
Pour éviter ce souci, les calculatrices permettent de placer un signe $-$ devant la valeur absolue d'une entier relatif binaire pour coder le nombre entier négatif correspondant.

La notation scientifique décimale

Nécessité des "flottants" en machine

Nombres à virgule très fréquents en sciences :

Ex : mesure d'une épaisseur de lame de verre : $e = 1,12\ mm$.

La notation scientifique décimale

Nécessité des "flottants" en machine

Nombres à virgule très fréquents en sciences :

Ex : mesure d'une épaisseur de lame de verre : $e = 1,12\ mm$.

Question : **Si nous souhaitons exploiter cette mesure avec un ordinateur, doit-il absolument comprendre ce qu'est un nombre à virgule ?**

La notation scientifique décimale

Nécessité des "flottants" en machine

RÉPONSE : Non ! ! ! En choisissant comme unité d'écriture le μm, le résultat devient $h = 1120\ \mu m$, c'est à dire un entier que l'ordinateur comprend ! ! !

La notation scientifique décimale

Nécessité des "flottants" en machine

RÉPONSE : Non ! ! ! En choisissant comme unité d'écriture le μm, le résultat devient $h = 1120\ \mu m$, c'est à dire un entier que l'ordinateur comprend ! ! !

PROBLÈME :
intervalle des nombres entiers représentables en machine limité par la profondeur de bits choisie.

La notation scientifique décimale

Nécessité des "flottants" en machine

Réponse : Non ! ! ! En choisissant comme unité d'écriture le μm, le résultat devient $h = 1120\ \mu m$, c'est à dire un entier que l'ordinateur comprend ! ! !

Problème :
intervalle des nombres entiers représentables en machine limité par la profondeur de bits choisie.

Donc si "translation" de la virgule complètement à droite pour **tous les opérandes d'un calcul** $\Longrightarrow$ risque d'obtenir des nombres plus grands que la limite représentable en mémoire $\Longrightarrow$ risque d'"*overflow*".

La notation scientifique décimale

Nécessité des "flottants" en machine

Imaginons par exemple que nous codions les entiers sur 32 bits dans une machine et qu'à l'aide de celle-ci nous souhaitions calculer le volume de la lame de verre ; un objectif plutôt raisonnable pour les ordinateurs modernes et pourtant ! ! ! Cela donne :

$$\begin{bmatrix} \text{epaisseur} & e = 1120\ \mu m \\ \text{largeur} & l = 2,5\ cm = 25000\mu m \\ \text{longueur} & L = 6\ cm = 60000\mu m \end{bmatrix}$$

$$\Longrightarrow V(\mu m^3) = 1120 * 25000 * 60000 = 1680000000000\mu m^3 > \underbrace{2147483647}_{\text{limite 32 bits}}$$

Ainsi, ce simple calcul provoque un dépassement ! ! !

La notation scientifique décimale

Nécessité des "flottants" en machine

SOLUTION :

Exemples de représentation en mémoire et conversions (partie optionnelle !)

La notation scientifique décimale

Nécessité des "flottants" en machine

<u>SOLUTION :</u> étendre la fourchette des nombres représentables avec la notation scientifique décimale :

La notation scientifique décimale

Nécessité des "flottants" en machine

<u>SOLUTION :</u> étendre la fourchette des nombres représentables avec la notation scientifique décimale :

$$x = \text{chiffres significatifs} \cdot \underbrace{10^n}_{\text{ordre de grandeur}}$$

Par exemple : $c \simeq 3{,}0.10^8 \ m.s^{-1}$ et $\mu_0 = 4\pi.10^{-7} \ H.m^{-1}$

La notation scientifique décimale

Nécessité des "flottants" en machine

<u>Solution :</u> étendre la fourchette des nombres représentables avec la notation scientifique décimale :

$$x = \text{chiffres significatifs} \cdot \underbrace{10^n}_{\text{ordre de grandeur}}$$

Par exemple : $c \simeq 3{,}0.10^8\ m.s^{-1}$ et $\mu_0 = 4\pi.10^{-7}\ H.m^{-1}$

Volume de la lame de verre en μm^3 :

$$\begin{bmatrix} \text{epaisseur} & e = 1120\ \mu m = 112.10^1 \mu m \\ \text{largeur} & l = 2{,}5\ cm = 25000 \mu m = 25.10^3 \mu m \\ \text{longueur} & L = 6\ cm = 60000 \mu m = 6.10^4 \mu m \end{bmatrix} \Longrightarrow V(\mu m^3) = 112*25*6 = \underbrace{16800}_{<2147483647} .10^8 \mu m^3$$

La notation scientifique décimale

Nécessité des "flottants" en machine

SOLUTION : étendre la fourchette des nombres représentables avec la notation scientifique décimale :

$$x = \text{chiffres significatifs} \cdot \underbrace{10^n}_{\text{ordre de grandeur}}$$

Par exemple : $c \simeq 3{,}0.10^8 \ m.s^{-1}$ et $\mu_0 = 4\pi.10^{-7} \ H.m^{-1}$

Volume de la lame de verre en μm^3 :

$$\left[\begin{array}{cc} \text{epaisseur} & e = 1120 \ \mu m = 112.10^1 \mu m \\ \text{largeur} & l = 2{,}5 \ cm = 25000 \mu m = 25.10^3 \mu m \\ \text{longueur} & L = 6 \ cm = 60000 \mu m = 6.10^4 \mu m \end{array}\right] \Longrightarrow V(\mu m^3) = 112 * 25 * 6 = \underbrace{16800}_{<2147483647} .10^8 \mu m^3$$

On évite ainsi les dépassements.

Décomposition en notation scientifique décimale

Notation scientifique décimale très utile notamment en physique :

Décomposition en notation scientifique décimale

Notation scientifique décimale très utile notamment en physique :

$$\begin{cases} c = 2{,}99792458.10^{8} \;\; m.s^{-1} \\ k = 1{,}3806488.10^{-23} \;\; J.K^{-1} \\ \mathcal{N}_a = 6{,}022142.10^{23} \, mol^{-1} \end{cases}$$

Décomposition en notation scientifique décimale

Notation scientifique décimale très utile notamment en physique :

$$\begin{cases} c = 2,99792458.10^{8} \ \ m.s^{-1} \\ k = 1,3806488.10^{-23} \ \ J.K^{-1} \\ \mathcal{N}_a = 6,022142.10^{23}\, mol^{-1} \end{cases}$$

$\forall x \in \mathbb{R}$:

$$\boxed{x = (-1)^{S} \times 10^{E} \times \left(d_0 + d_1.10^{-1} + d_2.10^{-2} + \ldots + d_i.10^{-i} + \ldots\right)}$$

$$\text{avec} \begin{cases} S = 0,1 \text{ qui fixe le signe de x} \\ E \text{ entier relatif (exposant)} \\ d_i \in \{0,1,2,\ldots,9\} \ \textbf{et on fixe } d_0 \neq 0 \end{cases}$$

Décomposition en notation scientifique décimale

Exemple : constante de Boltzmann k :

$$k = (-1)^0 \times 10^{-23} \left(1 + 3.10^{-1} + 8.10^{-2} + 0.10^{-3} + 6.10^{-4} + 4.10^{-5} + 8.10^{-6} + 8.10^{-7}\right) \; J.K^{-1}$$

Décomposition en base binaire

Principe

Même idée qu'en base 10 :

Décomposition en base binaire

Principe

Même idée qu'en base 10 :
$\forall x \in \mathbb{R}$:

$$\boxed{x = (-1)^S \times 2^E \times \left(b_0 + b_1.2^{-1} + b_2.2^{-2} + \ldots + b_i.2^{-i} + \ldots\right)} \qquad (1)$$

$$\text{avec} \begin{cases} S = 0,1 \text{ qui fixe le signe de x} \\ E \text{ entier relatif (exposant)} \\ b_i \in \{0,1\} \textbf{ et on fixe } b_0 = 1 \end{cases}$$

Décomposition en base binaire

Principe

Même idée qu'en base 10 :
$\forall x \in \mathbb{R}$:

$$\boxed{x = (-1)^S \times 2^E \times \left(b_0 + b_1.2^{-1} + b_2.2^{-2} + \ldots + b_i.2^{-i} + \ldots\right)} \qquad (1)$$

$$\text{avec } \begin{cases} S = 0,1 \text{ qui fixe le signe de x} \\ E \text{ entier relatif (exposant)} \\ b_i \in \{0,1\} \textbf{ et on fixe } b_0 = 1 \end{cases}$$

<u>CONCLUSION :</u>

Décomposition en base binaire

Principe

Même idée qu'en base 10 :
$\forall x \in \mathbb{R}$:

$$\boxed{x = (-1)^S \times 2^E \times \left(b_0 + b_1.2^{-1} + b_2.2^{-2} + \ldots + b_i.2^{-i} + \ldots\right)} \quad (1)$$

$$\text{avec } \begin{cases} S = 0,1 \text{ qui fixe le signe de x} \\ E \text{ entier relatif (exposant)} \\ b_i \in \{0,1\} \textbf{ et on fixe } b_0 = 1 \end{cases}$$

<u>Conclusion :</u>
Pour connaître le réel x il faut connaître S, E et les b_i.

Décomposition en base binaire

Ecriture normalisée

Profondeur de codage des nombres en machine limitée (en python, c'est la taille mémoire qui est le facteur limitant)
$\Longrightarrow$ on doit fatalement limiter E et la donnée des b_i.

Décomposition en base binaire

Ecriture normalisée

Profondeur de codage des nombres en machine limitée (en python, c'est la taille mémoire qui est le facteur limitant)
$\Longrightarrow$ on doit fatalement limiter E et la donnée des b_i.

Idée : **On normalise l'écriture des réels flottants**

Décomposition en base binaire

Ecriture normalisée

Définition de la **mantisse** M

Décomposition en base binaire

Ecriture normalisée

DÉFINITION DE LA **mantisse** M

On garde $m+1$ premiers termes b_i ce qui permet de définir **la mantisse** avec $b_0 = 1$:

$$\boxed{M = 1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + b_3 \times 2^{-3} + \ldots + b_m \times 2^{-m}}$$

avec $M \overset{b_0=1}{=} \sum_{k=0}^{m} \frac{b_k}{2^k} < \sum_{k=0}^{m} 2^{-k} < \sum_{k=0}^{\infty} 2^{-k} = \frac{1}{1-2^{-1}} = 2$ ainsi $\boxed{1 \leq M < 2}$

Décomposition en base binaire

Ecriture normalisée

Définition de la **mantisse** M

On garde $m+1$ premiers termes b_i ce qui permet de définir **la mantisse** avec $b_0 = 1$:

$$\boxed{M = 1 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + b_3 \times 2^{-3} + \ldots + b_m \times 2^{-m}}$$

avec $M \overset{b_0=1}{=} \sum_{k=0}^{m} \frac{b_k}{2^k} < \sum_{k=0}^{m} 2^{-k} < \sum_{k=0}^{\infty} 2^{-k} = \frac{1}{1-2^{-1}} = 2$ ainsi $\boxed{1 \leq M < 2}$

$\Longrightarrow$ **la mantisse est connue lorsque l'on connait les** m **premiers** b_i :

le codage de M occupe m bits dans la mémoire

Décomposition en base binaire

Ecriture normalisée

Définition de l'exposant E

Décomposition en base binaire

Ecriture normalisée

Définition de l'exposant E

On réserve e bits pour coder l'exposant.

Décomposition en base binaire

Ecriture normalisée

Définition de l'exposant E

On réserve e bits pour coder l'exposant.

Plutôt en mémoire : exposant décalé E' tel que :

$$E' = E + 2^{e-1} - 1 \qquad \text{en imposant } E' \in [0, 2^e - 1] \text{ donc } E \in \mathbb{N}$$

Décomposition en base binaire

Ecriture normalisée

Définition de l'exposant E

On réserve e bits pour coder l'exposant.

Plutôt en mémoire : exposant décalé E' tel que :

$$E' = E + 2^{e-1} - 1 \qquad \text{en imposant } E' \in [0, 2^e - 1] \text{ donc } E \in \mathbb{N}$$

Intervalle des valeurs possibles de l'exposant E :

$$0 \leq E' \leq 2^e - 1$$
$$\Leftrightarrow 0 + 1 - 2^{e-1} \leq E' + 1 - 2^{e-1} \leq 2^e - 1 + 1 - 2^{e-1}$$
$$\Leftrightarrow 1 - 2^{e-1} \leq E \leq 2^{e-1}$$

Décomposition en base binaire

Ecriture normalisée

soit :

$\longrightarrow \quad E_{norm} \in [1-2^{e-1}, 2^{e-1}]$

Décomposition en base binaire

Ecriture normalisée

soit :

$\longrightarrow \quad E_{norm} \in [1-2^{e-1}, 2^{e-1}]$

<u>ATTENTION :</u> cet intervalle est différent de celui employé pour coder un entier relatif sur e bits. La convention adoptée ici ne vaut que pour l'exposant.

Décomposition en base binaire

Ecriture normalisée

soit :

$\longrightarrow \quad E_{norm} \in [1-2^{e-1}, 2^{e-1}]$

ATTENTION : cet intervalle est différent de celui employé pour coder un entier relatif sur e bits. La convention adoptée ici ne vaut que pour l'exposant.

IMPORTANT : certaines valeurs réservées :

Décomposition en base binaire

Ecriture normalisée

soit :

$\longrightarrow \quad E_{norm} \in [1-2^{e-1}, 2^{e-1}]$

ATTENTION : cet intervalle est différent de celui employé pour coder un entier relatif sur e bits. La convention adoptée ici ne vaut que pour l'exposant.

IMPORTANT : certaines valeurs réservées :

- $E' = 2^e - 1$ soit $E = 2^{e-1}$ est employé pour coder l'infini ou un calcul conduisant à coder un "NaN" i.e. **N**ot **A** **N**umber, comme par exemple la division par 0.

Décomposition en base binaire

Ecriture normalisée

soit :

$\longrightarrow \quad E_{norm} \in [1-2^{e-1}, 2^{e-1}]$

ATTENTION : cet intervalle est différent de celui employé pour coder un entier relatif sur e bits. La convention adoptée ici ne vaut que pour l'exposant.

IMPORTANT : certaines valeurs réservées :

- $E' = 2^e - 1$ soit $E = 2^{e-1}$ est employé pour coder l'infini ou un calcul conduisant à coder un "NaN" i.e. **N**ot **A** **N**umber, comme par exemple la division par 0.
- $E' = 0$ soit $E = 1 - 2^{e-1}$ est employé pour coder les nombres dénormalisés pour lesquels $M < 1$.

Décomposition en base binaire

Ecriture normalisée

soit :

$\longrightarrow \quad E_{norm} \in [1 - 2^{e-1}, 2^{e-1}]$

ATTENTION : cet intervalle est différent de celui employé pour coder un entier relatif sur e bits. La convention adoptée ici ne vaut que pour l'exposant.

IMPORTANT : certaines valeurs réservées :

- $E' = 2^e - 1$ soit $E = 2^{e-1}$ est employé pour coder l'infini ou un calcul conduisant à coder un "NaN" i.e. **N**ot **A** **N**umber, comme par exemple la division par 0.
- $E' = 0$ soit $E = 1 - 2^{e-1}$ est employé pour coder les nombres dénormalisés pour lesquels $M < 1$.

Finalement :

$$\boxed{E_{norm.poss} \in [2 - 2^{e-1}, 2^{e-1} - 1]}$$

Décomposition en base binaire

Ecriture normalisée

A RETENIR :

CONCLUSION : un nombre binaire codé sur $m+e+1$ *bits* s'écrit :

$$\underbrace{S}_{\text{signe}}\ \underbrace{c_{e-1}\ldots c_1\,c_0}_{\text{exposant E'}}\ \underbrace{b_1\,b_2\ldots b_m}_{\text{mantisse}}$$

et sa valeur décimale est donnée par la relation 1 une fois normalisée, soit :

$$\boxed{x=(-1)^s\times 2^E\times M}$$

IMPORTANT :

Décomposition en base binaire

Ecriture normalisée

A RETENIR :

CONCLUSION : un nombre binaire codé sur $m+e+1$ *bits* s'écrit :

$$\underbrace{S}_{\text{signe}}\ \underbrace{c_{e-1}\ldots c_1 c_0}_{\text{exposant E'}}\ \underbrace{b_1 b_2 \ldots b_m}_{\text{mantisse}}$$

et sa valeur décimale est donnée par la relation 1 une fois normalisée, soit :

$$\boxed{x = (-1)^s \times 2^E \times M}$$

IMPORTANT :

- Intervalle des réels représentables en mémoire limité.

Décomposition en base binaire

Ecriture normalisée

A RETENIR :

CONCLUSION : un nombre binaire codé sur $m+e+1$ *bits* s'écrit :

$$\underbrace{S}_{\text{signe}}\underbrace{c_{e-1}\dots c_1 c_0}_{\text{exposant E'}}\underbrace{b_1 b_2 \dots b_m}_{\text{mantisse}}$$

et sa valeur décimale est donnée par la relation 1 une fois normalisée, soit :

$$\boxed{x = (-1)^s \times 2^E \times M}$$

IMPORTANT :

- Intervalle des réels représentables en mémoire limité.
- Impossible de tous les représenter dans un intervalle car infinité ! ! !

Décomposition en base binaire

Ecriture normalisée

A RETENIR :

CONCLUSION : un nombre binaire codé sur $m+e+1$ *bits* s'écrit :

$$\underbrace{S}_{\text{signe}}\ \underbrace{c_{e-1}\dots c_1 c_0}_{\text{exposant E'}}\ \underbrace{b_1 b_2 \dots b_m}_{\text{mantisse}}$$

et sa valeur décimale est donnée par la relation 1 une fois normalisée, soit :

$$\boxed{x = (-1)^s \times 2^E \times M}$$

IMPORTANT :

- Intervalle des réels représentables en mémoire limité.
- Impossible de tous les représenter dans un intervalle car infinité ! ! !
- Nombre de bits consacrés à la mantisse $b_1 \dots b_m$ limité à $m \Longrightarrow \exists$ intervalle minimal "incompressible" entre deux réels successifs représentables.

Décomposition en base binaire

Ecriture normalisée

A RETENIR :

CONCLUSION : un nombre binaire codé sur $m+e+1$ *bits* s'écrit :

$$\underbrace{S}_{\text{signe}}\ \underbrace{c_{e-1}\dots c_1 c_0}_{\text{exposant E'}}\ \underbrace{b_1 b_2 \dots b_m}_{\text{mantisse}}$$

et sa valeur décimale est donnée par la relation 1 une fois normalisée, soit :

$$\boxed{x = (-1)^s \times 2^E \times M}$$

IMPORTANT :

- Intervalle des réels représentables en mémoire limité.
- Impossible de tous les représenter dans un intervalle car infinité ! ! !
- Nombre de bits consacrés à la mantisse $b_1 \dots b_m$ limité à $m \Longrightarrow \exists$ intervalle minimal "incompressible" entre deux réels successifs représentables.

NB : la plupart des réels décimaux ne sont pas représentables exactement en mémoire avec ce format $\longrightarrow$ écriture approchée et précision dépendant de la profondeur de bits.

Norme IEEE754 - limitation des nombres en virgule flottante

Depuis 1985 : **norme IEEE754**

Norme IEEE754 - limitation des nombres en virgule flottante

Depuis 1985 : **norme IEEE754**
$\longrightarrow$ fixe le nombre de bits consacrés à M,E et s en fonction de la profondeur de représentation choisie :

Norme IEEE754 - limitation des nombres en virgule flottante

Depuis 1985 : **norme IEEE754**
$\longrightarrow$ fixe le nombre de bits consacrés à M,E et s en fonction de la profondeur de représentation choisie :

Profondeur	Signe S	Exposant décalé E'	Mantisse M
32 *bits*	1 *bits*	8 *bits*	23 *bits*
64 *bits*	1 *bits*	11 *bits*	52 *bits*

Norme IEEE754 - limitation des nombres en virgule flottante

Exercice n°3:

Quelques précisions sur cette norme

1. *Quel est l'intervalle des exposants représentables en norme IEEE754 pour une profondeur de 64 bits ?*
2. *Déterminer l'expression du plus grand réel positif représentable x_M, et donner sa valeur numérique sous la forme $r \times 10^n$ avec n entier et $|r| < 10$, toujours en 64 bits.*
3. *Déterminer l'expression du plus petit réel positif normalisé représentable x_m , et donner sa valeur numérique sous la forme $x \times 10^n$ encore une fois en 64 bits.*

Norme IEEE754 - limitation des nombres en virgule flottante

RÉPONSE :

1. $E \in [2 - 2^{10} + 1, 2^{10} - 1] = [-1022, 1023]$
2. $x_M = (-1)^0 \times 2^{1023} \times \left[1 + \sum_{i=1}^{52} \frac{1}{2^i}\right] = 1,79769.10^{308}$
3. $x_m = (-1)^0 \times 2^{-1022} \times \underbrace{\left[1 + \sum_{i=1}^{52} \frac{0}{2^i}\right]}_{=1} = 2,22507.10^{-308}$

Norme IEEE754 - limitation des nombres en virgule flottante

Dans la mesure où le nombre de bits consacré à M est limité, il existe une précision absolue de représentation des nombres réels en virgule flottante. Cette précision absolue est définie par la différence entre deux réels flottants consécutifs représentables :

$$\delta x = |x' - x|$$

Norme IEEE754 - limitation des nombres en virgule flottante

EXERCICE N°4:

Précision de représentation (à faire)

1. *Déterminer la plus petite variation de mantisse possible.*
2. *En déduire la precision absolue δx pour les plus petits nombres représentables en 64 bits et pour les plus grands.*

Norme IEEE754 - limitation des nombres en virgule flottante

Remarque :

Extension possible de l'intervalle des flottants représentables / augmentation précision sur un flottant :

Norme IEEE754 - limitation des nombres en virgule flottante

REMARQUE :
Extension possible de l'intervalle des flottants représentables / augmentation précision sur un flottant :
$\longrightarrow$ il faut sortir de IEEE754

Norme IEEE754 - limitation des nombres en virgule flottante

Remarque :

Extension possible de l'intervalle des flottants représentables / augmentation précision sur un flottant :

$\longrightarrow$ il faut sortir de IEEE754

Principe :

Norme IEEE754 - limitation des nombres en virgule flottante

REMARQUE :
Extension possible de l'intervalle des flottants représentables / augmentation précision sur un flottant :
$\longrightarrow$ il faut sortir de IEEE754

PRINCIPE :

- Modification nombre de bits consacrés à e et m avec contrainte $e + m = cste = 7,\ 15,\ 31,$ *ou* 63 *bits* (le bit manquant étant celui réservé au signe du flottant).

Norme IEEE754 - limitation des nombres en virgule flottante

REMARQUE :
Extension possible de l'intervalle des flottants représentables / augmentation précision sur un flottant :
$\longrightarrow$ il faut sortir de IEEE754

PRINCIPE :

- Modification nombre de bits consacrés à e et m avec contrainte $e + m = cste = 7,\ 15,\ 31,\ ou\ 63\ bits$ (le bit manquant étant celui réservé au signe du flottant).
- Compromis entre portée (intervalle des réels représentés) et précision :

Norme IEEE754 - limitation des nombres en virgule flottante

REMARQUE :
Extension possible de l'intervalle des flottants représentables / augmentation précision sur un flottant :
$\longrightarrow$ il faut sortir de IEEE754

PRINCIPE :

- Modification nombre de bits consacrés à e et m avec contrainte $e + m = cste = 7,\ 15,\ 31,\ ou\ 63\ bits$ (le bit manquant étant celui réservé au signe du flottant).
- Compromis entre portée (intervalle des réels représentés) et précision :
  - Augmentation de précision : augmentation m, diminution e (on peut représenter moins de flottants)

Norme IEEE754 - limitation des nombres en virgule flottante

REMARQUE :
Extension possible de l'intervalle des flottants représentables / augmentation précision sur un flottant :
$\longrightarrow$ il faut sortir de IEEE754

PRINCIPE :

- Modification nombre de bits consacrés à e et m avec contrainte $e + m = cste = 7,\ 15,\ 31,\ ou\ 63\ bits$ (le bit manquant étant celui réservé au signe du flottant).
- Compromis entre portée (intervalle des réels représentés) et précision :
  - Augmentation de précision : augmentation m, diminution e (on peut représenter moins de flottants)
  - Augmentation portée : augmentation e, diminution m (la précision chute)

Exemples de représentation en mémoire et conversions (partie optionnelle !)

<u>QUESTIONS :</u> comment représenter en base 10 un nombre donné en binaire format 64 bits IEEE754 ?

Exemples de représentation en mémoire et conversions (partie optionnelle !)

QUESTIONS : comment représenter en base 10 un nombre donné en binaire format 64 bits IEEE754 ?

RÉPONSE : par simple "lecture" ! ! ! !

Exemples de représentation en mémoire et conversions (partie optionnelle !)

QUESTIONS : comment représenter en base 10 un nombre donné en binaire format 64 bits IEEE754 ?

RÉPONSE : par simple "lecture" ! ! ! !
Sur un exemple : $(x)_2 =$
0000010001101001001111000011100000000000000000000000000000000000

Exemples de représentation en mémoire et conversions (partie optionnelle !)

<u>QUESTIONS :</u> comment représenter en base 10 un nombre donné en binaire format 64 bits IEEE754 ?

<u>RÉPONSE :</u> par simple "lecture" ! ! ! !
Sur un exemple : $(x)_2 =$
0000010001101001001111000011100000000000000000000000000000000000

- bit de signe $= 0 \Longrightarrow$ nombre positif

Exemples de représentation en mémoire et conversions (partie optionnelle !)

QUESTIONS : comment représenter en base 10 un nombre donné en binaire format 64 bits IEEE754 ?

RÉPONSE : par simple "lecture" ! ! ! !
Sur un exemple : $(x)_2 =$
0000010001101001001111000011100000000000000000000000000000000000

- bit de signe $= 0 \Longrightarrow$ nombre positif
- bits d'exposant décalé $E' = 00001000110 = 70$ soit $E = E' - 2^{e-1} + 1 = 70 - 2^{10} + 1 = 70 - 1024 + 1 = -953$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

QUESTIONS : comment représenter en base 10 un nombre donné en binaire format 64 bits IEEE754 ?

RÉPONSE : par simple "lecture" ! ! ! !

Sur un exemple : $(x)_2 =$
0000010001101001001111000011100000000000000000000000000000000000

- bit de signe $= 0 \Longrightarrow$ nombre positif
- bits d'exposant décalé $E' = 00001000110 = 70$ soit $E = E' - 2^{e-1} + 1 = 70 - 2^{10} + 1 = 70 - 1024 + 1 = -953$
- bits de mantisse : $m = 1 + 2^{-1} + 2^{-4} + 2^{-7} + 2^{-8} + 2^{-9} + 2^{-10} + 2^{-15} + 2^{-16} + 2^{-17} = \dfrac{(2^{17} + 2^{16} + 2^{13} + 2^{10} + 2^{9} + 2^{8} + 2^{7} + 2^{2} + 2 + 1)}{2^{17}}$

  soit $m = \dfrac{206727}{131072}$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

Finalement :

$$\Longrightarrow \quad \boxed{x_{10} = +2^{-953} \times \frac{206727}{131072}}$$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

QUESTIONS : comment représenter en base 2 format 64 bits IEEE754 un nombre donné en décimal ?

Exemples de représentation en mémoire et conversions (partie optionnelle !)

QUESTIONS : comment représenter en base 2 format 64 bits IEEE754 un nombre donné en décimal ?
RÉPONSE : les choses se compliquent un peu ici ! ! ! Il existe plusieurs méthodes ; nous proposons ici la méthode de décomposition par division déjà exploitée pour la conversion des entiers.

Exemples de représentation en mémoire et conversions (partie optionnelle !)

On rappelle qu'un réel flottant normalisé s'écrit de la manière suivante :

$$x = (-1)^S \times 2^E \times \underbrace{\left[1 + \sum_{i=0}^{52} b_i \cdot 2^{-i}\right]}_{1 \leq M < 2}$$

Exemple : on veut coder le décimal $x = 0,4$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

BIT DE SIGNE :

$$x > 0 \quad \Longrightarrow \quad \text{le bit de signe est } 0$$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

Bit de signe :

$$x > 0 \quad \Longrightarrow \quad \text{le bit de signe est } 0$$

Bits d'exposant :

Exemples de représentation en mémoire et conversions (partie optionnelle !)

BIT DE SIGNE :

$$x > 0 \quad \Longrightarrow \quad \text{le bit de signe est } 0$$

BITS D'EXPOSANT :

- $x < 1 \quad \Longrightarrow \quad E < 0$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

Bit de signe :

$$x > 0 \quad \Longrightarrow \quad \text{le bit de signe est } 0$$

Bits d'exposant :

- $x < 1 \quad \Longrightarrow \quad E < 0$
- $x < 0.5 = 2^{-1} \quad \Longrightarrow \quad E \neq -1$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

BIT DE SIGNE :

$$x > 0 \quad \Longrightarrow \quad \text{le bit de signe est } 0$$

BITS D'EXPOSANT :

- $x < 1 \quad \Longrightarrow \quad E < 0$
- $x < 0.5 = 2^{-1} \quad \Longrightarrow \quad E \neq -1$
- $x > \dfrac{1}{2^2} = 0,25 \quad \Longrightarrow \quad E = -2 \quad \Longrightarrow \quad E' = E + 1023 = 1021$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

BIT DE SIGNE :

$$x > 0 \quad \Longrightarrow \quad \text{le bit de signe est } 0$$

BITS D'EXPOSANT :

- $x < 1 \quad \Longrightarrow \quad E < 0$
- $x < 0.5 = 2^{-1} \quad \Longrightarrow \quad E \neq -1$
- $x > \dfrac{1}{2^2} = 0,25 \quad \Longrightarrow \quad E = -2 \quad \Longrightarrow \quad E' = E + 1023 = 1021$
- Ainsi : $E' = (01111111101)_2$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

<u>BITS DE MANTISSE :</u>
Reste à coder la somme apparaissant dans la mantisse :

$$\sum_{i=0}^{52} b_i \cdot 2^{-i} = x \times 2^{-E} - 1 = 0.4 \times 4 - 1 = 0,6$$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$$0,6 \times 2 = 1,2 \qquad \text{le coefficient } b_1 \text{ de } 2^{-1} \text{ est} \quad 1$$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$$\begin{array}{lll} 0,6 \times 2 = 1,2 & \text{le coefficient } b_1 \text{ de } 2^{-1} \text{ est} & 1 \\ 0,2 \times 2 = 0,4 & \text{le coefficient } b_2 \text{ de } 2^{-2} \text{ est} & 0 \end{array}$$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de $2^{-i} \equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$$\begin{array}{lllll} 0,6 \times 2 = 1,2 & \text{le coefficient } b_1 \text{ de } 2^{-1} \text{ est} & 1 \\ 0,2 \times 2 = 0,4 & \text{le coefficient } b_2 \text{ de } 2^{-2} \text{ est} & 0 \\ 0,4 \times 2 = 0,8 & \text{le coefficient } b_3 \text{ de } 2^{-3} \text{ est} & 0 \end{array}$$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$$
\begin{array}{lll}
0,6 \times 2 = 1,2 & \text{le coefficient } b_1 \text{ de } 2^{-1} \text{ est} & 1 \\
0,2 \times 2 = 0,4 & \text{le coefficient } b_2 \text{ de } 2^{-2} \text{ est} & 0 \\
0,4 \times 2 = 0,8 & \text{le coefficient } b_3 \text{ de } 2^{-3} \text{ est} & 0 \\
0,8 \times 2 = 1,6 & \text{le coefficient } b_4 \text{ de } 2^{-4} \text{ est} & 1
\end{array}
$$

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$0,6 \times 2 = 1,2$	le coefficient b_1 de 2^{-1} est	1
$0,2 \times 2 = 0,4$	le coefficient b_2 de 2^{-2} est	0
$0,4 \times 2 = 0,8$	le coefficient b_3 de 2^{-3} est	0
$0,8 \times 2 = 1,6$	le coefficient b_4 de 2^{-4} est	1
$0,6 \times 2 = 1,2$	le coefficient b_5 de 2^{-5} est	1

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$$
\begin{array}{lll}
0,6 \times 2 = 1,2 & \text{le coefficient } b_1 \text{ de } 2^{-1} \text{ est} & 1 \\
0,2 \times 2 = 0,4 & \text{le coefficient } b_2 \text{ de } 2^{-2} \text{ est} & 0 \\
0,4 \times 2 = 0,8 & \text{le coefficient } b_3 \text{ de } 2^{-3} \text{ est} & 0 \\
0,8 \times 2 = 1,6 & \text{le coefficient } b_4 \text{ de } 2^{-4} \text{ est} & 1 \\
0,6 \times 2 = 1,2 & \text{le coefficient } b_5 \text{ de } 2^{-5} \text{ est} & 1
\end{array}
$$

…périodique ! ! !

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$0,6 \times 2 = 1,2$	le coefficient b_1 de 2^{-1} est	1
$0,2 \times 2 = 0,4$	le coefficient b_2 de 2^{-2} est	0
$0,4 \times 2 = 0,8$	le coefficient b_3 de 2^{-3} est	0
$0,8 \times 2 = 1,6$	le coefficient b_4 de 2^{-4} est	1
$0,6 \times 2 = 1,2$	le coefficient b_5 de 2^{-5} est	1

...périodique ! ! !

$\Longrightarrow$ **codage périodique**, $\Longrightarrow$ mantisse limitée par la profondeur de représentation choisie, soit 52 bits ici.

CONCLUSION :

Exemples de représentation en mémoire et conversions (partie optionnelle !)

$\Longrightarrow$ méthode par division afin de décomposer 0,6 en puissance de 2^{-i} $\equiv$ méthode par multiplication car diviser par 2^{-i} revient à multiplier par 2^{i} :

$$
\begin{array}{lll}
0,6 \times 2 = 1,2 & \text{le coefficient } b_1 \text{ de } 2^{-1} \text{ est} & 1 \\
0,2 \times 2 = 0,4 & \text{le coefficient } b_2 \text{ de } 2^{-2} \text{ est} & 0 \\
0,4 \times 2 = 0,8 & \text{le coefficient } b_3 \text{ de } 2^{-3} \text{ est} & 0 \\
0,8 \times 2 = 1,6 & \text{le coefficient } b_4 \text{ de } 2^{-4} \text{ est} & 1 \\
0,6 \times 2 = 1,2 & \text{le coefficient } b_5 \text{ de } 2^{-5} \text{ est} & 1
\end{array}
$$

...périodique ! ! !

$\Longrightarrow$ **codage périodique**, $\Longrightarrow$ mantisse limitée par la profondeur de représentation choisie, soit 52 bits ici.

<u>CONCLUSION :</u> 0,4 va s'écrire en norme IEEE754 :

$$\underbrace{0}_{=S}\ \underbrace{01111111101}_{=E'}\ \underbrace{1001100110011001100110011001100110011001100110011001}_{\text{coeff. } b_1, b_2, \ldots, b_{52}}$$