Ch. 3
Nombres Flottants

B. Quoitin
(bruno.quoitin@umons.ac.be)

Objectifs

Au delà de la représentation des entiers ?

- Jusqu'à présent, nous nous sommes focalisés sur la représentation de nombres entiers. Comment étendre cette représentation aux nombres réels ?

Méthodologie

- Nous allons commencer par aborder la **représentation en virgule fixe**, une extension simple de la représentation entière que nous avons vue jusqu'ici.
- Ensuite, nous passerons à la **représentation en virgule flottante**, représentation supportée par la plupart des processeurs modernes. Nous nous intéresserons en particulier au standard **IEEE 754**.

Limites

- Finalement, nous identifierons un certain nombre de limites de la représentation en virgule flottante qu'un informaticien / mathématicien doit **absolument** comprendre.

Table des matières

Représentation en virgule fixe

- Limites de représentation
- Approximation et erreurs

Représentation en virgule flottante

- Représentation normalisée et bit caché

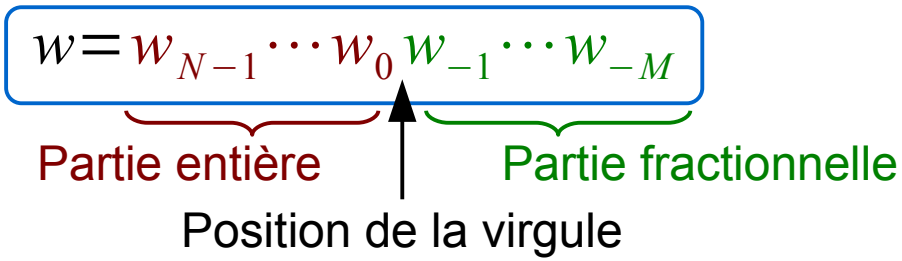
Standard IEEE 754

- Borne sur l'erreur relative
- Arrondis
- Addition
- *Swamping* et *cancellation*

Représentation en virgule fixe

Représentation en virgule fixe *(fixed point)*

- Dans la représentation en virgule fixe, une séquence w de $N+M$ bits est utilisée.
  - N bits de poids fort représentent la partie entière
  - M bits de poids faible représentent la partie fractionnelle



- Le nombre x représenté est obtenu par

$$x = \sum_{i=-M}^{N-1} w_i . 2^i$$

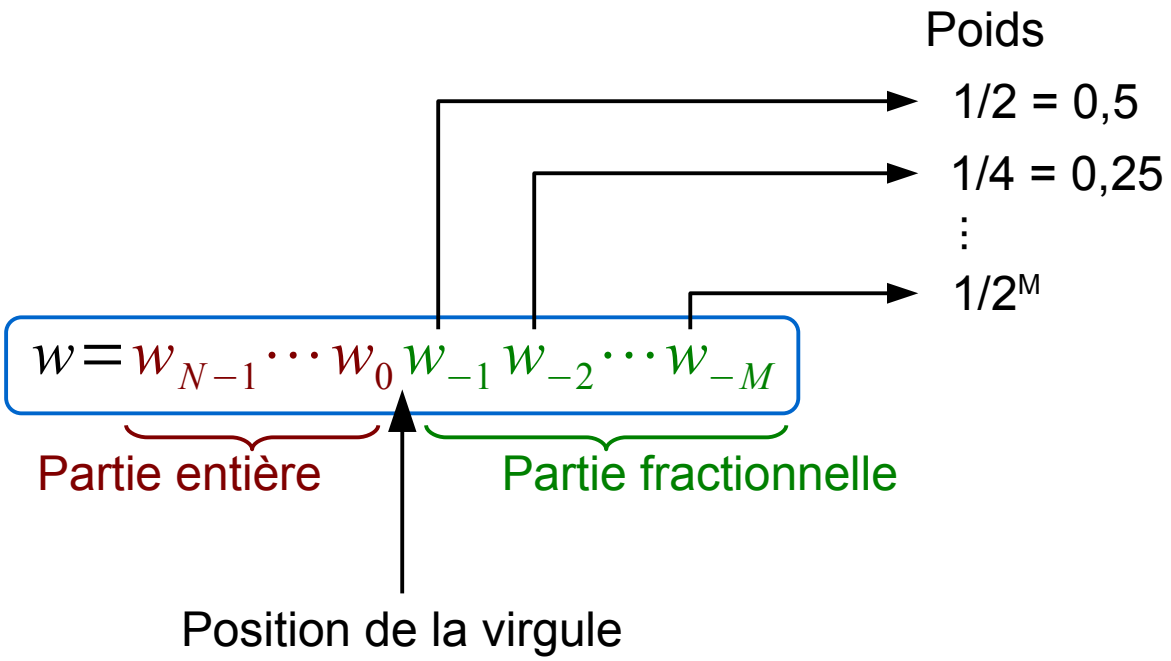
Indices (et exposants) négatifs.

- Nous ne représentons que des nombres positifs, mais il est aisé de modifier la représentation pour supporter les nombres négatifs.

Représentation en virgule fixe

Représentation en virgule fixe *(fixed point)*

- Chaque bit de la partie fractionnelle a un poids égal à une fraction fixe dont le dénominateur est un exposant de 2.



Représentation en virgule fixe

Algorithme - Conversion partie fractionnelle en binaire

- hypothèse : $0 \leq x < 1$
- multiplications successives par 2 + test de la partie entière

```
w = ""
bits = 0
while x > 0 and bits < M:
    x = x * 2
    bits += 1
    if int(x) > 0:
        x = x - 1
        w = w + "1"
    else:
        w = w + "0"
```

retourne la partie entière

limite le nombre de bits de la représentation

0.6875
1.375
0.75
1.5
1.0

⬇

0.1011

0.815
1.63
1.26
0.52
1.04
0.08
0.16
0.32
0.64
1.28
...

⬇

0.110100001.....

Représentation en virgule fixe

Exemple

- Avec $N=8$ pour la partie entière et $M=4$ pour la partie fractionnelle, que représente le mot w = `010011000011` ?

$$
\begin{aligned}
x &= 2^6 + 2^3 + 2^2 + 2^{-3} + 2^{-4} \\
&= 64 + 8 + 4 + \frac{1}{8} + \frac{1}{16} \\
&= 76{,}1875
\end{aligned}
$$

Représentation en virgule fixe

Propriété

- Il est possible de ré-organiser sensiblement l'équation précédente afin d'exprimer le nombre représenté sous forme d'une fraction dont le dénominateur est fixé.

$$x = \sum_{i=-M}^{N-1} w_i . 2^i$$

$$= \sum_{i=0}^{N+M-1} w_{i-M} . 2^{i-M}$$

$$= \frac{1}{2^M} \cdot \left(\sum_{i=0}^{N+M-1} w_{i-M} . 2^i \right)$$

Dénominateur fixe dépendant uniquement de M

Numérateur variable appelé la *"mantisse"* (m) représenté sur $N+M$ bits

$$= m \cdot 2^{-M}$$

- Note : la valeur de M (fixe) dicte quelle est la "position de la virgule".

Représentation en virgule fixe

Exemple

- Avec N=8 pour la partie entière et M=4 pour la partie fractionnelle, que représente le mot w = `010011000011` ?

$$x = \frac{1}{2^M} \cdot \left(\sum_{i=0}^{N+M-1} w_{i-M} \cdot 2^i \right)$$

$$= \frac{1}{2^4} \cdot \left(2^{10} + 2^7 + 2^6 + 2^1 + 2^0 \right)$$

$$= \frac{1024 + 128 + 64 + 2 + 1}{16}$$

$$= \frac{1219}{16}$$

$$= 76{,}1875$$

Représentation en virgule fixe

Propriété - Limites de représentabilité

- Etant donnés N et M
- Plus grand nombre représentable $\frac{2^{N+M}-1}{2^{M}}$ (tous les bits valent 1)
- Plus petit nombre non-nul représentable $\frac{1}{2^{M}}$ (seul le bit de poids le plus faible vaut 1)
  - C'est aussi le plus petit écart entre deux nombres représentables exactement.

- Exemple: avec N=8 et M=4
  - plus grand = 255,9375 `111111111111`
  - plus petit = 0,0625 `000000000001`

Représentation en virgule fixe

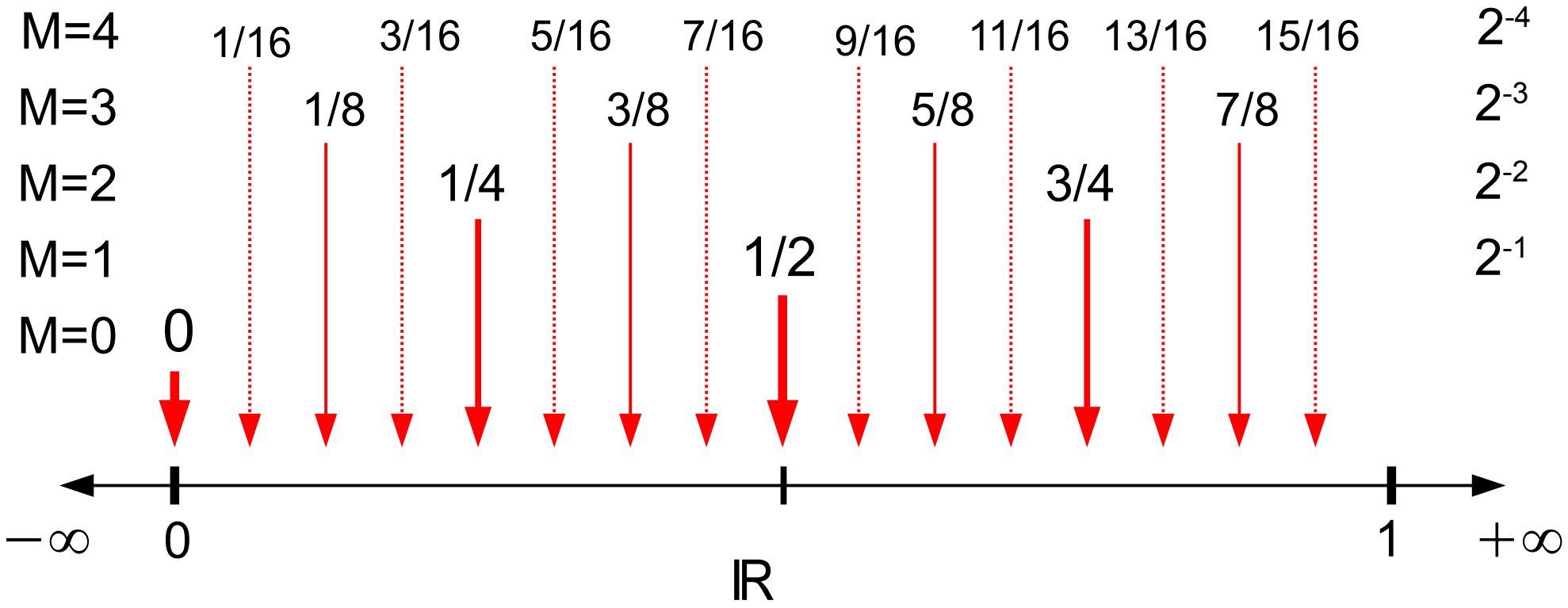
Observation - Représentation d'un sous-ensemble fini de réels

- La représentation en virgule fixe ne permet pas de représenter tous les nombres réels dans un intervalle, comme cela était possible pour les entiers.
- Il existe une infinité de réels entre deux entiers alors que pour une taille de mots $N+M$, il est possible de représenter seulement 2^{N+M} nombres différents !

Représentation en virgule fixe

Observation - Représentation d'un sous-ensemble fini de réels (suite)

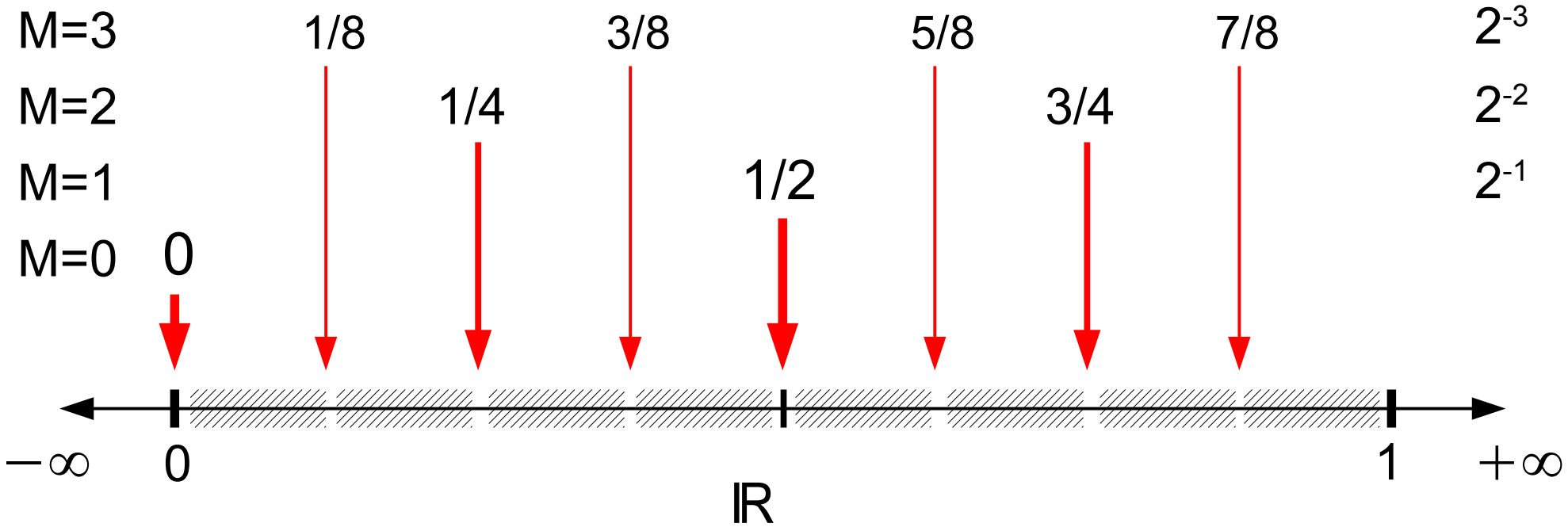
- La figure ci-dessous illustre les nombres représentables en virgule fixe dans l'intervalle $[0, 1[$ avec différentes tailles M pour la partie fractionnelle. Seulement 2^M nombres sont représentables dans cet intervalle !



Représentation en virgule fixe

Observation - Représentation d'un sous-ensemble fini de réels (suite)

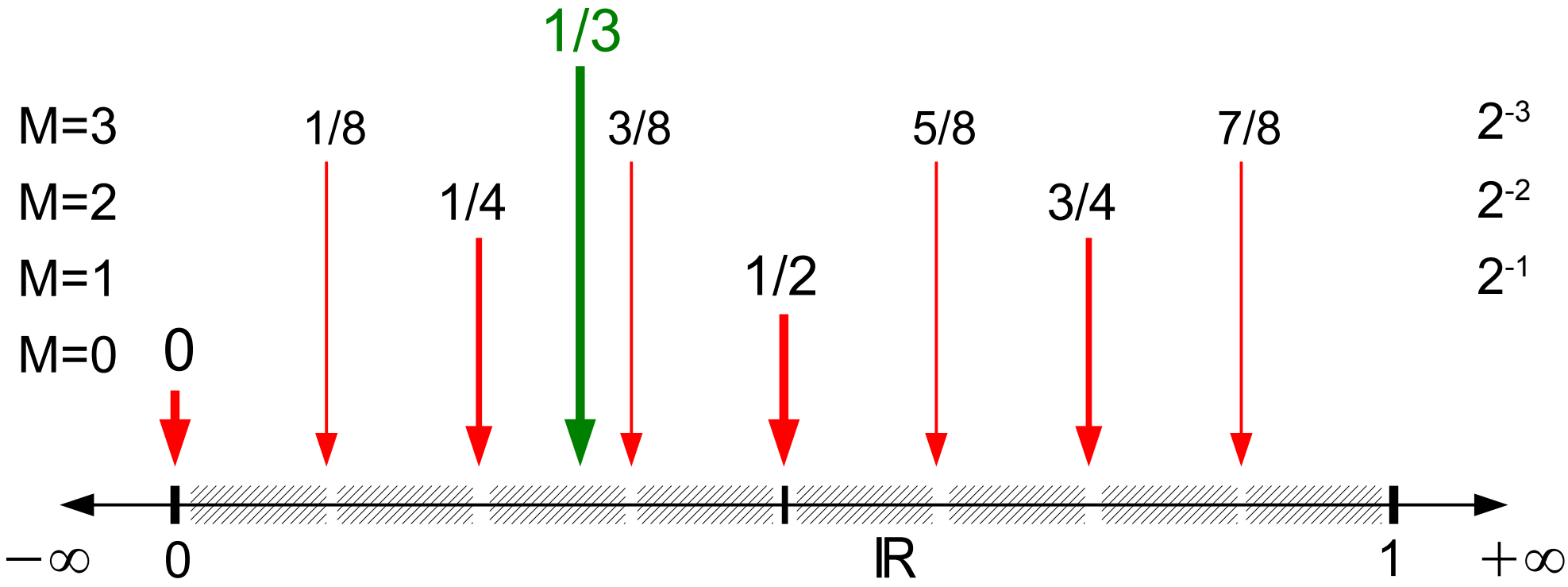
- Avec une taille de partie fractionnelle M donnée, on ne peut représenter les nombres dans les intervalles] $k.2^{-M}$, $(k+1).2^{-M}$ [pour k entier tel que $0 \leq k < 2^{M}$.
- Ces intervalles de nombres non représentables sont représentés par la partie hachurée dans la figure ci-dessous.



Représentation en virgule fixe

Représentation approximative

- Quelque soit la taille de la partie fractionnelle M, il est impossible de représenter certains nombres. On ne peut en représenter qu'une **approximation**.
- C'est le cas de la fraction **1/3** illustrée ci-dessous !



- Quelle est la plus proche approximation de 1/3 pour M fini donné ?

Représentation en virgule fixe

Représentation approximative

- Soit x un nombre à représenter.
- Son approximation $\hat{x}$ dans la représentation choisie sera le nombre le plus proche de x représentable exactement.

- Utiliser une approximation lors de la représentation d'un nombre engendre une erreur de représentation. On exprime souvent cette erreur avec les mesures suivantes

- **Erreur vraie** $\Delta_x = x - \hat{x}$
- **Erreur absolue** $|\Delta_x| = |x - \hat{x}|$
- **Erreur relative** $\epsilon_x = \frac{|x - \hat{x}|}{|x|}$

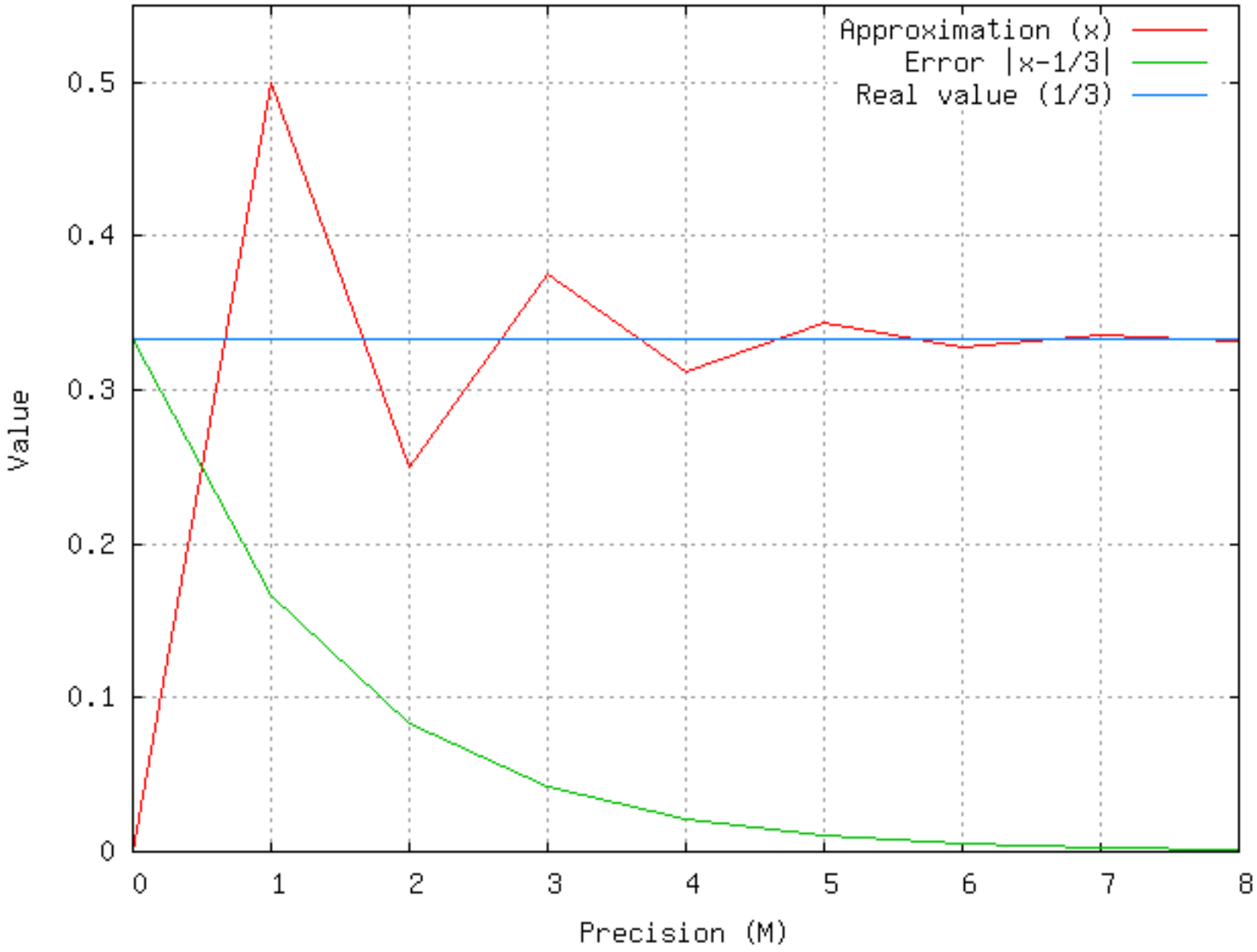
Représentation en virgule fixe

Approximation et erreur de représentation de 1/3

- Plus la taille de la partie fractionnelle M augmente, plus l'erreur d'approximation Δ_x diminue.

M	Plus petit intervalle représentable où se situe x = 1/3	Approximation $\hat{x}$	Erreur absolue Δ_x	Représentation w
0	[**0**, 1]	0	1/3	`0`
1	[0, **1/2**]	0,5	1/6	`0.1`
2	[**1/4**, 1/2]	0,25	1/12	`0.01`
3	[1/4, **3/8**]	0,375	1/24	`0.011`
4	[**5/16**, 3/8]	0,3125	1/48	`0.0101`
5	[5/16, **11/32**]	0,34375	1/96	`0.01011`
6	[**21/64**, 11/32]	0,328125	1/192	`0.010101`
7	[21/64, **43/128**]	0,3359375	1/384	`0.0101011`
8	[**85/256**, 43/128]	0,33203125	1/768	`0.01010101`

Représentation en virgule fixe



Représentation en virgule fixe

Bits significatifs

- Définition : Les bits significatifs sont tous les bits qui suivent le premier bit non-nul, ce dernier y compris.

- Exemple : Considérons deux nombres représentés en virgule fixe avec $N=8$ et $M=4$

$0\underbrace{10000100010}$ (66,625)

11 bits significatifs

$000000\underbrace{100111}$ (2,4375)

6 bits significatifs

- Observation : en représentation en virgule fixe, les grands nombres ont plus de bits significatifs que les petits nombres. Pour les petits nombres, les bits de poids fort de la représentation ne sont pas utilisés efficacement. **Comment améliorer cela ?**

Table des matières

Représentation en virgule fixe

- Limite de représentation
- Approximation et erreurs

Représentation en virgule flottante

- Représentation normalisée et bit caché

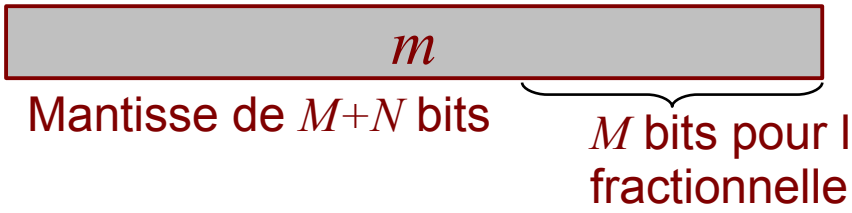
Standard IEEE 754

- Borne sur l'erreur relative
- Arrondis
- Addition
- *Swamping* et *cancellation*

Représentation en virgule flottante

- La représentation en virgule flottante est similaire à la représentation en virgule fixe à l'exception de la position de la virgule qui est variable (flottante).

Virgule fixe

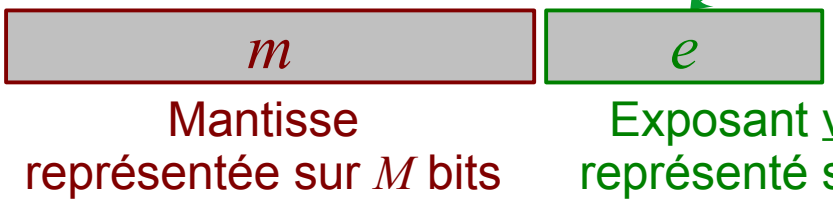


Exposant fixe (-M)

$$x = m.2^{-M} = \frac{m}{2^M}$$

- La position de la virgule est encodée dans la représentation, sous forme d'une valeur d'exposant e pour le dénominateur, encodée sur E bits.

Virgule flottante

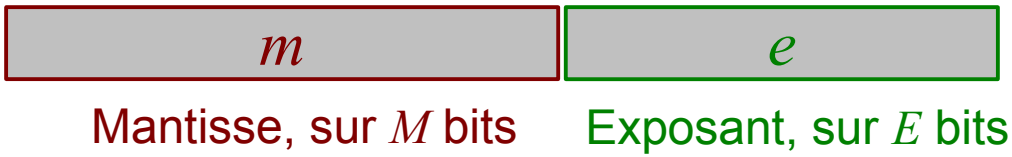


$$x = m.2^{e} = \frac{m}{2^{-e}}$$

Représentation en virgule flottante

Principe

- La valeur x représentée est obtenue par l'équation suivante.



$$x = m.2^{e}$$
$$= \left(\sum_{i=0}^{M-1} m_i.2^{i} \right).2^{\left(\sum_{j=0}^{E-1} e_j.2^{j} \right)}$$

  - La valeur de m est obtenue en interprétant les M bits correspondants comme un naturel. M contrôle le nombre de bits significatifs supportés par la représentation.
  - De façon similaire, la valeur de e est obtenue en interprétant les E bits correspondants comme un naturel. E contrôle la position de la virgule et donc la possibilité de représenter des nombres plus ou moins grands/petits.

Biais de l'exposant

- Afin de permettre des exposants positifs et négatifs, on convient d'un **biais** B fixe qui est soustrait de la valeur encodée de l'exposant e.
- La valeur de B est choisie de façon à permettre l'encodage d'autant d'exposants positifs que d'exposants négatifs. Elle est généralement dérivée du nombre de bits E utilisés pour encoder l'exposant e, comme

$$B = 2^{E-1} - 1 \quad \text{ou} \quad B = 2^{E-1}$$

  - Exemple: Supposons E=8, B vaut alors 127, ce qui permet d'exprimer des exposants (e - B) allant de -127 à 128.

- La valeur x représentée est alors obtenue par l'équation suivante. La valeur du biais B, fixée par convention, n'est pas encodée dans la représentation.

$$x = m.2^{(e-B)}$$

Note: comme il y a 2^E naturels représentables avec E bits, même en prenant B comme point milieu, il est impossible d'avoir autant de valeurs positives que de valeurs négatives.

Représentation en virgule flottante

Exemple

- Supposons taille mantisse $M=12$; taille exposant $E=4$; biais $B=8$.
- Que valent les nombres représentés par les mots suivants ?

$w = \underbrace{0100110000110100}$

11 chiffres significatifs

$$\begin{aligned} x &= (2^{10}+2^{7}+2^{6}+2^{1}+2^{0}).2^{(2^{2})-8} \\ &= (1024+128+64+2+1).2^{4-8} \\ &= (1024+128+64+2+1).2^{-4} \\ &= 76{,}1875 \end{aligned}$$

$w = \underbrace{1001100001100011}$

12 chiffres significatifs

$$\begin{aligned} x &= (2^{11}+2^{8}+2^{7}+2^{2}+2^{1}).2^{(2^{1}+2^{0})-8} \\ &= (2048+256+128+4+2).2^{3-8} \\ &= (2048+256+128+4+2).2^{-5} \\ &= 76{,}1875 \end{aligned}$$

- Observation : **il peut exister plusieurs représentations d'un même nombre, avec un nombre de chiffres significatifs différents !**

Représentation en virgule flottante

Représentation normalisée

- La représentation en virgule flottante permet de maximiser le nombre de bits significatifs. A cet effet, on privilégie l'usage d'une mantisse qui est normalisée.

- Une mantisse m est **normalisée** si et seulement si m est de la forme

$$m = 1\, m_{M-2} \cdots m_0$$

M bits significatifs (maximum)

- La représentation en virgule flottante $w = (m, e)$ d'un nombre est **normalisée** si sa mantisse m est normalisée.

- Note: la représentation normalisée ne permet pas de représenter le nombre 0.

Représentation en virgule flottante

Algorithme - Normalisation

- Soit un nombre $x \neq 0$
- On souhaite trouver (m, e) tel que m est normalisé (i.e. de la forme $1,\ldots$) et $x = m.2^e$

```
m = x
e = 0
while m < 1:
    m = m * 2
    e = e - 1
while m >= 2:
    m = m / 2
    e = e + 1
```

m	e
17,33	0
8,665	1
4,3325	2
2,16625	3
1,083125	4

$17,33 = 1,08\ldots \times 2^4$

m	e
0,19	0
0,38	-1
0,76	-2
1,52	-3

$0,19 = 1,52\ldots \times 2^{-3}$

Représentation en virgule flottante

Bit caché

- En représentation normalisée, le bit le plus significatif est toujours égal à 1. Il est donc inutile de le représenter.

$$m = 1\, m_{M-2} \cdots m_0$$

M bits significatifs (maximum)

- On peut donc utiliser un **bit caché** *(hidden digit)*, c-à-d. pris en compte mais non représenté.

$$m = (1)\, m_{M-1} \cdots m_0$$

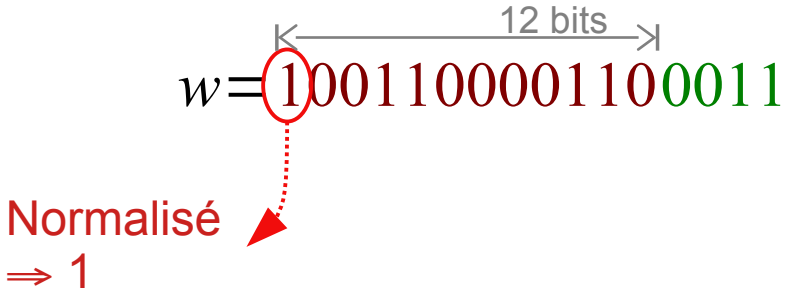
$M+1$ bits significatifs

  - Cela permet de récupérer 1 bit supplémentaire pour la mantisse, ce qui permet d'augmenter la précision.

Représentation en virgule flottante

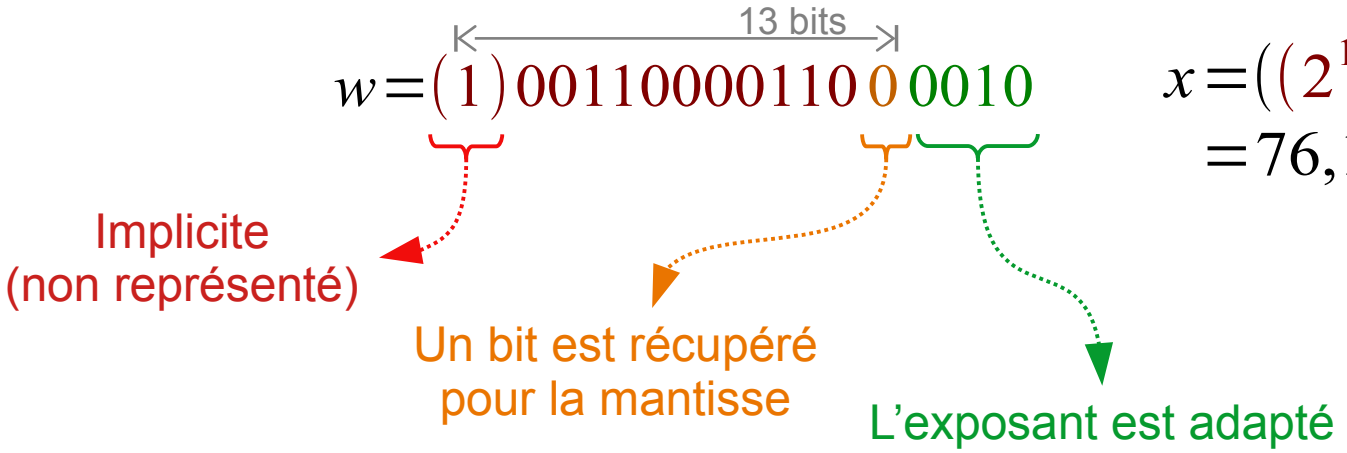
Exemple

- Supposons: taille mantisse $M=12$; taille exposant $E=4$; biais $B=8$
- En <u>représentation normalisée</u>, le nombre $76{,}1875$ est représenté comme



$$x = (2^{11} + 2^8 + 2^7 + 2^2 + 2^1).2^{(2^1+2^0)-8} = 76{,}1875$$

- En <u>représentation normalisée avec bit caché</u>, le même nombre est représenté comme



$$x = ((2^{12}) + 2^9 + 2^8 + 2^3 + 2^2).2^{(2^1)-8} = 76{,}1875$$

Table des matières

Représentation en virgule fixe

- Limite de représentation
- Approximation et erreurs

Représentation en virgule flottante

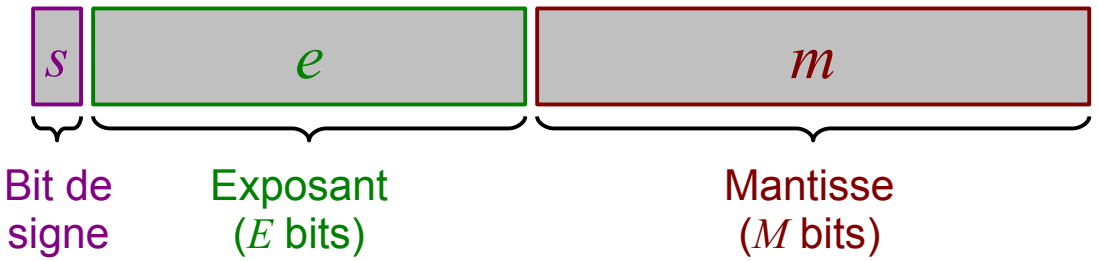
- Représentation normalisée et bit caché

Standard IEEE 754

- Borne sur l'erreur relative
- Arrondis
- Addition
- *Swamping* et *cancellation*

Principes

- Le standard IEEE 754 décrit une représentation de nombres en virgule flottante très largement employée sur les ordinateurs et processeurs aujourd'hui.
- Les caractéristiques importantes de la représentation IEEE 754 sont
  - convention sur le format de représentation (tailles des mantisse, exposant et valeur du biais).
  - bit de signe
  - représentation normalisée avec bit caché
  - représentation dénormalisée pour 0 et des nombres proches de 0

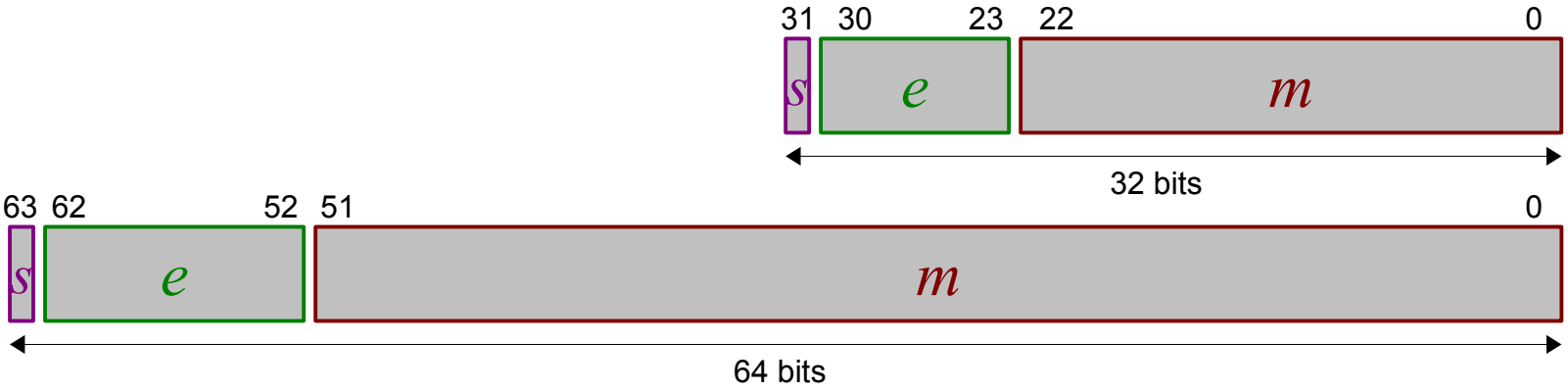


Paramètres

- Deux tailles de mots sont répandues : **simple précision** (4 octets) et **double précision** (8 octets).

Type	Taille de Mantisse (M)	Taille d'Exposant (E)	Biais (B)	Domaine représentable (approximatif)
Simple précision (32 bits) [type `float`]	23	8	127	[-3.4x10^{38}, 3.4x10^{38}]
Double précision (64 bits) [type `double`]	52	11	1023	[-1.8x10^{308}, 1.8x10^{308}]

- Le biais B est calculé comme $B = 2^{E-1}-1$



Représentation normalisée

- La valeur de l'exposant e indique aussi comment interpréter le nombre
  - lorsque $0 < e < 2^E\text{-}1$, il s'agit d'une représentation **normalisée**.
  - lorsque $e = 0$, représentation **dénormalisée**
  - lorsque $e = 2^E\text{-}1$, valeurs **spéciales** (infini, NaN)

- Dans le cas **normalisé**, la valeur du nombre x représenté est obtenue avec l'équation suivante:

$$x = (-1)^s . \left(1 + \frac{m}{2^M}\right) . 2^{e-B}$$

$$où \quad \begin{cases} 0 \leq m \leq 2^M - 1 \\ 0 < e < 2^E - 1 \end{cases}$$

Note: les valeurs e=0 et e=2^E-1 sont réservées
(cf. représentation dé-normalisée et valeurs spéciales)

Exemple - simple précision, normalisé

- Quel nombre est représenté en IEEE 754 simple précision par le mot suivant ?

$$w = 0 \quad 10000101 \quad 00110000110000000000000$$

- Le premier bit représente le signe (s), les 8 bits suivants représentent l'exposant (e) et les 23 derniers bits représentent la mantisse (m).

$$m = 2^{20} + 2^{19} + 2^{14} + 2^{13} = 1597440$$

$$e = 2^7 + 2^2 + 2^0 = 133$$

(0 < 133 < 255 → normalisé)

$$x = (-1)^s . \left(1 + \frac{m}{2^M}\right) . 2^{e-127}$$

$$= \left(1 + \frac{1597440}{2^{23}}\right) . 2^6$$

$$= 76.1875$$

Représentation dé-normalisée

- La représentation **dé-normalisée** permet de représenter la valeur 0 ainsi que des valeurs proches de 0.
- Dans ce cas, l'équation suivante est utilisée pour déterminer la valeur du nombre x représenté. Il n'y a plus de bit caché et la valeur effective de l'exposant est égale à 1-B.

$$x = (-1)^s . \left(0 + \frac{m}{2^M}\right) . 2^{1-B}$$

$$où \quad 0 \leq m \leq 2^M - 1$$

Exemple - simple précision, dé-normalisé

- Quel nombre est représenté en IEEE 754 simple précision (32 bits) par le mot suivant ?

$$w = 0 \quad 00000000 \quad 00110000110000000000000$$

$$m = 2^{20} + 2^{19} + 2^{14} + 2^{13} = 1597440$$

$$e = 0$$ (e = 0 → dé-normalisé)

$$x = (-1)^s . \left(0 + \frac{m}{2^M}\right) . 2^{1-127}$$

$$= \left(\frac{1597440}{2^{23}}\right) . 2^{-126}$$

$$\approx 2.23849 \times 10^{-39}$$

Valeurs spéciales

- Le standard IEEE 754 réserve certaines valeurs de mot à la représentation de valeurs spéciales. La valeur d'exposant $e = 2^E\text{-}1$ (tous les bits à 1) est réservée pour les valeurs spéciales.
  - +/-∞ : valeur plus grande/petite que ce qui est représentable. Par exemple, résultat d'un dépassement de capacité ou d'une division par 0.
  - *NaN* (*not a number*) : valeur non réelle ou indéfinie. Par exemple racine de -1 ou $0.\infty$
- Exemple : valeurs spéciales de la représentation en simple précision

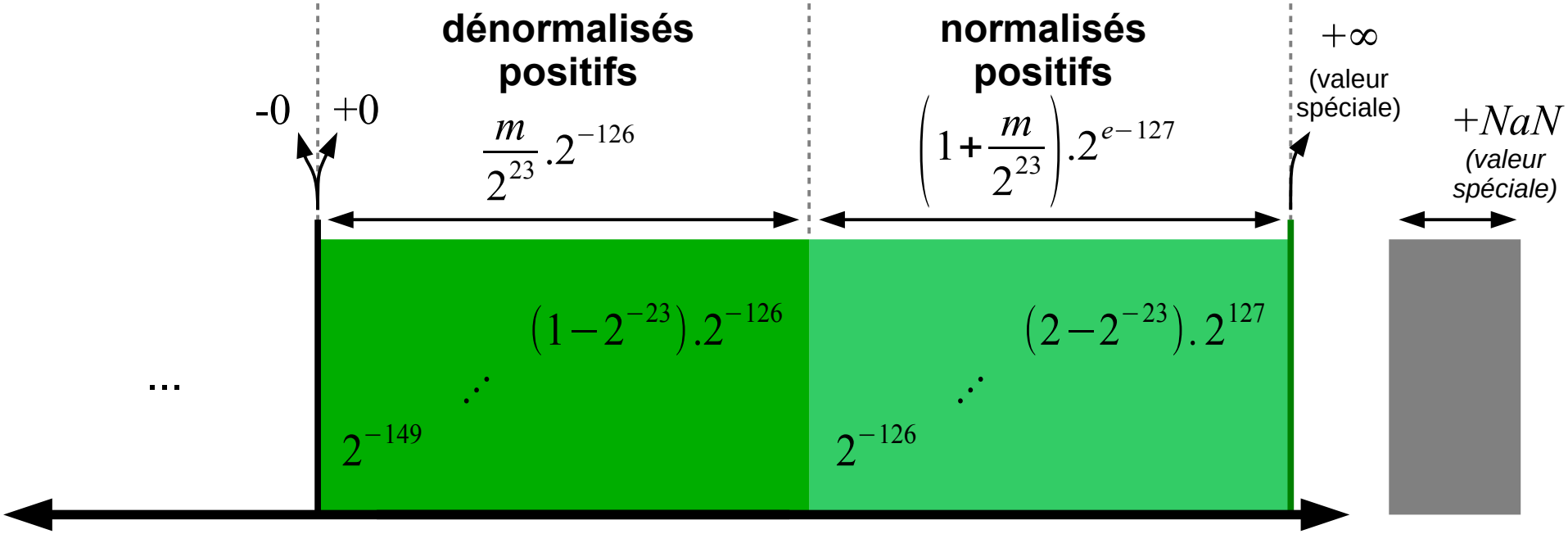
s	Exposant (e)	Mantisse (m)	Valeur
0	11111111	00000000000000000000000	+∞
1	11111111	00000000000000000000000	-∞
0 / 1	11111111	différent de 0	*NaN*

Valeurs spéciales

- **Exemples de conditions pour obtenir +/-∞**
  - $1/0 \rightarrow \infty$
  - $-1/0 \rightarrow -\infty$
- **Exemples de conditions pour obtenir *NaN***
  - $0/0$; $0 \times \infty$; ∞/∞ ; $\infty+(-\infty)$
  - sqrt(-1) ; log(-10)
- **Ordre**
  - $-\infty <$ (toute value autre que +/-∞ et *NaN*) $< +\infty$
  - *NaN* est non-ordonné, i.e. il n'est comparable (égal, plus petit ou plus grand) à aucun nombre (y compris *NaN*)

Résumé

- **Ordre des représentations** (*simple précision*)



La partie à gauche de 0 est symétrique (nombres négatifs).

$e = 0$
$0 < m \leq 2^{23}-1$

$0 < e < 255$
$0 \leq m \leq 2^{23}-1$

$e = 255$
$m > 0$

Table des matières

Représentation en virgule fixe

- Limite de représentation
- Approximation et erreurs

Représentation en virgule flottante

- Représentation normalisée et bit caché

Standard IEEE 754

- Borne sur l'erreur relative
- Arrondis
- Addition
- *Swamping* et *cancellation*

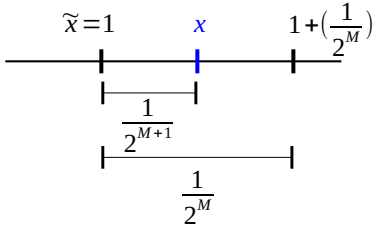
Précision machine: borne sur l'erreur relative

- La **précision machine** ou "*epsilon machine*" est une borne sur l'erreur relative qui dépend du format de représentation, en particulier de la taille de la mantisse M.

$$\frac{|x-\hat{x}|}{|x|} \leq \epsilon_M$$

- Pour IEEE 754, on peut montrer que l'epsilon machine est donné par

$$\epsilon_M = 2^{-(M+1)}$$



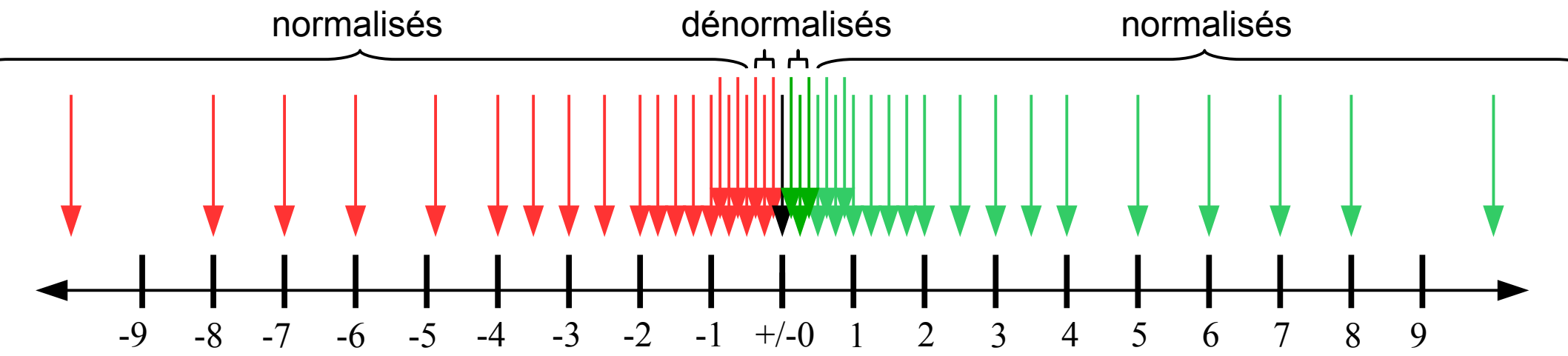
Cas avec pire erreur

$\epsilon_x = \frac{|x-\tilde{x}|}{|x|} \approx \epsilon_M$

- en double précision : $M = 52 \Rightarrow \epsilon_M = 2^{-53} \approx 1{,}1102 \times 10^{-16}$
- en simple précision : $M = 23 \Rightarrow \epsilon_M = 2^{-24} \approx 5{,}9605 \times 10^{-8}$

Précision près et loin de zéro

- Supposons un format de représentation flottante similaire à IEEE 754 (bit de signe, normalisé/dé-normalisé) mais avec des tailles d'exposant et de mantisse plus petites.
- Exemple : $M=2$, $E=3$ et $B=2$.



- On peut observer que les nombres représentables sont de moins en moins “serrés” lorsque l'on s'éloigne de zéro.

L'**erreur d'approximation** n'est donc pas la même pour un nombre proche ou éloigné de 0 !

IEEE 754

$M=2$
$E=3$
$B=2$

$\epsilon_M = 2^{-3} = \frac{1}{8}$

m	e	x
00	000	0
01	000	0.125
10	000	0.25
11	000	0.375
00	001	0.5
01	001	0.625
10	001	0.75
11	001	0.875
00	010	1
01	010	1.25
10	010	1.5
11	010	1.75
00	011	2
01	011	2,5
10	011	3
11	011	3,5
00	100	4
01	100	5
10	100	6
11	100	7
00	101	8
01	101	10
10	101	12
11	101	14
...	...	...

→ $x = 0{,}5626 \Rightarrow \hat{x} = 0{,}625$

$$\frac{|0{,}5626 - 0{,}625|}{0{,}5626} \approx 0{,}11 \leq \epsilon_M$$

→ $x = 1{,}8 \Rightarrow \hat{x} = 1{,}75$

$$\frac{|1{,}75 - 1{,}8|}{1{,}8} \approx 0{,}028 \leq \epsilon_M$$

→ $x = 8{,}99 \Rightarrow \hat{x} = 8$

$$\frac{|8{,}99 - 8|}{8{,}99} \approx 0{,}11 \leq \epsilon_M$$

Exercice

- Nous utilisons une représentation similaire à IEEE 754, avec des tailles réduites de mantisse et d'exposant
- Soit M=3, E=2 et un biais B=4. Il n'y a pas de valeurs spéciales (NaN, +/-∞) et seule la représentation normalisée est supportée.
- Comment représenter les nombres suivants
  - 1,5 ?
  - 0,625 ?
  - 0,815 ?

Solutions

- Représentation de 1,5
  - Conversion en binaire `1.1` (déjà normalisé)
  - L'exposant e-B doit valoir 0, or B=4 et $0 \leq e \leq 3$, ce qui est impossible.
  - Le nombre ne peut donc pas être représenté[(1)].
- Représentation de 0,625
  - Conversion en binaire `0.101`
  - Normalisation `1.01`
  - L'exposant e-B doit valoir -1, par conséquent e = -1+4 = 3
  - Le nombre est représenté avec m=`010` et e=`11`
- Représentation de 0,815
  - Conversion en binaire `0.11010000101000111101011`10…
  - Impossible à représenter exactement sur M=3 bits

(1) On peut d'ailleurs vérifier que le plus grand nombre représentable avec M=3, E=2 et B=4 est $(1+7/8)\cdot 2^{3-4} = 0{,}9375$

Table des matières

Représentation en virgule fixe

- Limite de représentation
- Approximation et erreurs

Représentation en virgule flottante

- Représentation normalisée et bit caché

Standard IEEE 754

- Borne sur l'erreur relative
- Arrondis
- Addition
- *Swamping* et *cancellation*

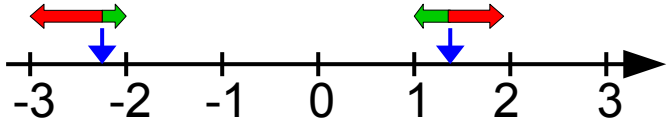
Arrondi (*rounding*)

- La représentation en virgule flottante permet de représenter un ensemble fini de nombres et ce avec une précision limitée. Dans la plupart des cas, seule une *approximation* peut être représentée.
- Soit un nombre x non représentable exactement. L'opération d'**arrondi** $round(x)$ permet d'obtenir un nombre représentable « proche » de x. Typiquement, $x^- \leq x \leq x^+$ où x^- et x^+ sont les deux nombres représentables les plus proches de x et $round(x) \in \{ x^-, x^+ \}$.
- Il existe plusieurs stratégies pour arrondir un nombre. Le standard IEEE 754 supporte 4 stratégies différentes[1].
  - *Round-to-nearest-even*
  - *Round-toward-zero*
  - *Round-down*
  - *Round-up*

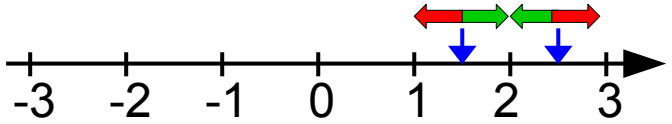
[1]Note : la révision du standard IEEE 754 en 2008 introduit un 5ème mode.

Arrondi au plus proche pair (*round-to-nearest-even*)

- Consiste à arrondir vers le **plus proche** (*nearest*) nombre représentable. Il s'agit de la méthode d'arrondi utilisée par défaut.



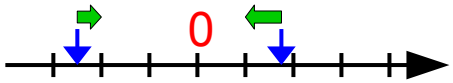
- **Egale distance aux 2 plus proches.** Si le nombre à arrondir se situe exactement entre les deux nombres représentables les plus proches, alors le plus proche **pair** (*even*) est choisi.



- Exemples : (basés sur une représentation décimale)
  - 1,4 arrondi entier à 1 (plus proche)
  - 1,5 arrondi entier à 2 (égale distance → plus proche pair)
  - 2,5 arrondi entier à 2 (égale distance → plus proche pair)
  - 2,54 arrondi avec un chiffre après la virgule à 2,5
  - 2,55 arrondi avec un chiffre après la virgule à 2,6
  - 2,45 arrondi avec un chiffre après la virgule à 2,4

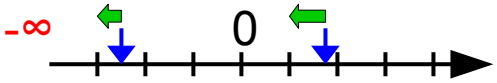
Autres arrondis

- **vers zéro *(round-toward-zero)***



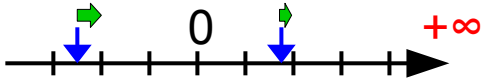
  - Consiste à **tronquer** le nombre de la partie non-représentable.
  - Exemples : 1,4 arrondi à 1 ; 1,5 arrondi à 1 ; -2,5 arrondi à -2

- **vers -∞ *(round-down)***



  - Consiste à prendre le nombre inférieur le plus proche
  - Exemples : 1,4 arrondi à 1 ; 1,5 arrondi à 1 ; -2,5 arrondi à -3

- **vers +∞ *(round-up)***



  - Consiste à prendre le nombre supérieur le plus proche.
  - Exemples : 1,4 arrondi à 2 ; 1,5 arrondi à 2 ; -2,5 arrondi à -2

Exemples en binaire

- `100.001` (4,125) à représenter sur 4 bits
  - arrondir vers le bas → `100.0` (4)
- `100.100` (4,5) à représenter sur 4 bits
  - arrondir vers le bas → `100.1` (4,5)
- `100.100` (4,5) à représenter sur 3 bits
  - égale distance de `100` (4) et `101` (5)
  - prendre le plus proche pair → `100` (4)
- `100.101` (4,625) à représenter sur 4 bits → `100.1` (4,5)
- `100.101` (4,625) à représenter sur 3 bits → `101` (5)
- `100.110` (4,75) à représenter sur 4 bits
  - égale distance de `100.1` (4,5) et `101.0` (5)
  - prendre le plus proche pair → `101.0` (5)
- `100.110` (4,75) à représenter sur 3 bits → `101` (5)

Exercice

- Soit M=3, E=2 et un biais B=4. Il n'y a pas de valeurs spéciales (NaN, +/-∞) et seule la représentation normalisée est supportée.
- Comment représenter le nombre $x = 0{,}815$?

- Solution
  - Conversion en binaire `0.1101000010100011110101110`…
  - Pas représentable exactement sur M=3 bits
  - Normalisation `1.1010000101000111101011`10…
  - Arrondi `1.101`
  - L'exposant e-B doit valoir -1, par conséquent $e = -1+4 = 3$
  - Le nombre arrondi est représenté avec m=`101` et e=`11`
    ce qui correspond à $x' = (1+5/8)\cdot 2^{3-4} = 0{,}8125$
  - L'erreur absolue est égale à $\Delta_x = |x - x'| = 0{,}025$ et l'erreur relative est égale à $\varepsilon_x = \Delta_x / |x| \approx 0{,}03064 < \varepsilon_M = 2^{-(M+1)} = 0{,}0625$

Table des matières

Représentation en virgule fixe

- Limite de représentation
- Approximation et erreurs

Représentation en virgule flottante

- Représentation normalisée et bit caché

Standard IEEE 754

- Borne sur l'erreur relative
- Arrondis
- Addition
- *Swamping* et *cancellation*

Virgule flottante: addition

Addition de nombres en virgule flottante

- Le principe est le suivant

1. Avant d'effectuer une addition, les deux nombres sont alignés sur leur virgule, i.e. ils sont alignés sur le nombre dont l'exposant est le plus élevé.
2. Les deux nombres alignés sont additionnés.
3. Si nécessaire, le résultat est normalisé afin de maximiser le nombre de bits significatifs.
   - durant la normalisation, il est possible que l'exposant devienne trop grand ou trop petit → erreur
4. Afin de tenir dans la taille limitée de la mantisse, le résultat est éventuellement arrondi.
   - il est possible qu'après l'arrondi, il faille retourner à l'étape 3. *(pourquoi ?)*

Virgule flottante: addition

Exemple en décimal

- Représentation décimale, mantisse composée de M=4 chiffres, addition de $x = 1{,}610\text{E-}1$ et $y = 9{,}999\text{E}1$

1) Alignement

$$0.016\text{E}1$$
$$9.999\text{E}1$$

l'exposant de x est adapté lors de l'alignement !

2) Addition

$$\begin{array}{rr} & 0.016\text{E}1 \\ + & 9.999\text{E}1 \\ \hline & 10.015\text{E}1 \end{array}$$

3) Normalisation

$$10.015\text{E}1 \rightarrow 1.0015\text{E}2$$

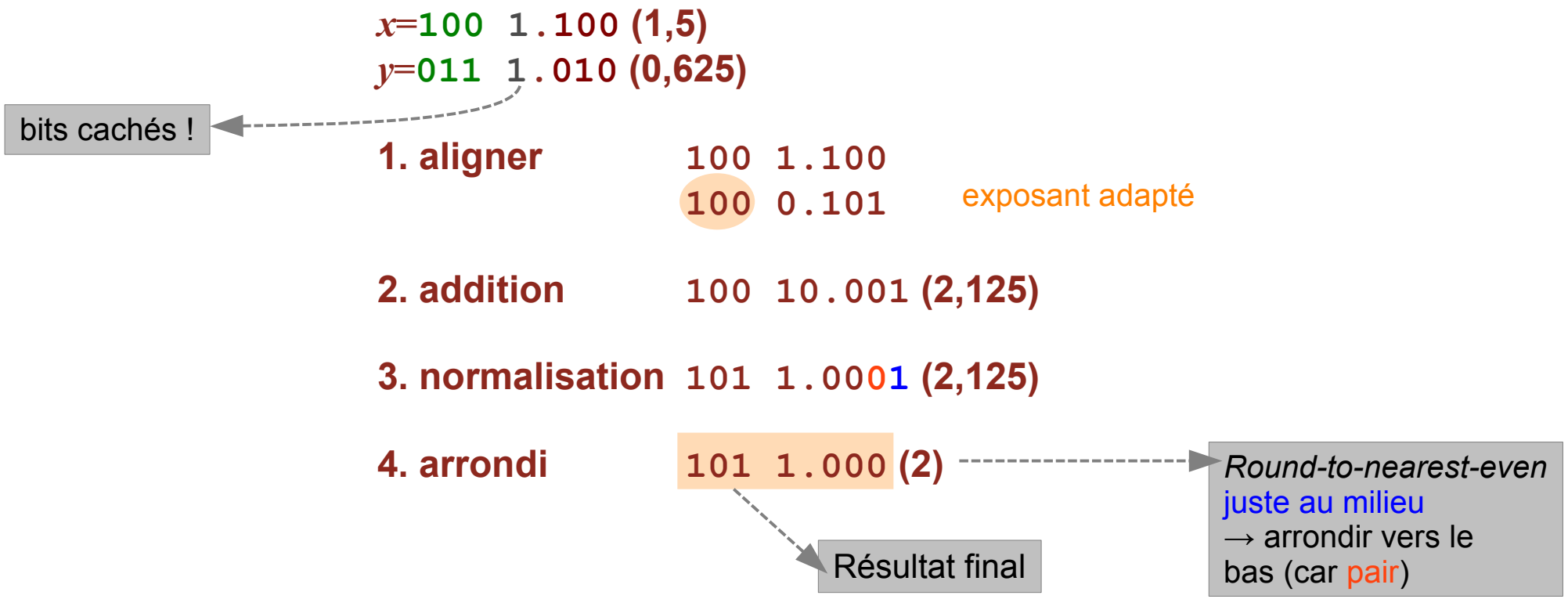
4) Arrondi

$$1.0015\text{E}2 \rightarrow 1.002\text{E}2$$

Virgule flottante: addition

Exemple en virgule flottante

- Représentation binaire, mantisse de taille $M=3$, exposant de taille $E=3$, biais $B=4$.



Virgule flottante: addition

Exemple en virgule flottante

- Représentation binaire, mantisse de taille $M=3$, exposant de taille $E=3$, biais $B=4$.

x=100 1.010 (1,25)
y=011 1.011 (0,6875)

1. aligner 100 1.0100
   100 0.1011 exposant adapté
2. addition 100 1.1111 (1,9375)
3. normalisation 100 1.1111 (1,9375)
4. arrondi 100 10.000 (2)
3. normalisation 101 1.0000 (2)
4. arrondi 101 1.000 (2)

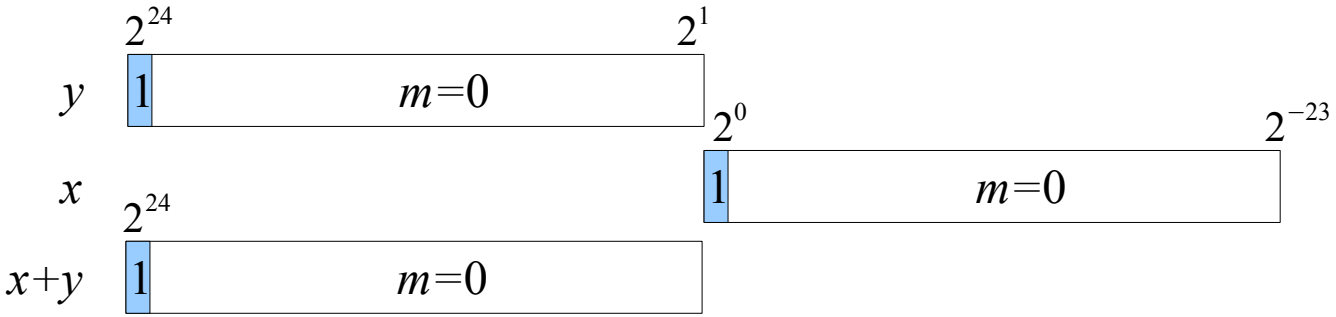
Round-to-nearest-even
juste au milieu
→ arrondir vers le haut (car impair)

Résultat final

Virgule flottante: limitations

Swamping : addition de grand et petit nombres

- Soient les nombres x=1 (2^0) et y=16777216 (2^{24})
- Leur représentation normalisée en IEEE 754 simple précision est composée comme suit. Il s'agit d'une <u>représentation exacte</u>.
  - x : (s=0, m=0, e=127)
  - y : (s=0, m=0, e=151)
- Pour effectuer l'addition de x et y, les mantisses des deux nombres dont d'abord alignés sur base de leur virgule (i.e. leur exposant).



Virgule flottante: limitations

Swamping : addition de grand et petit nombres

- Le résultat de l'addition est tel que

$$x \hat{+} y = y$$

- Quelle est l'erreur obtenue ?
  - erreur absolue

$$\Delta_{x+y} = |(x+y) - (x \hat{+} y)| = |x+y-y| = |x| = 1$$

  - erreur relative

$$\epsilon_{x+y} = \frac{\Delta_{x+y}}{|x+y|} = \frac{|x|}{|x+y|} = \frac{1}{(1+16777216)} \approx 5{,}96.10^{-8}$$

- Conclusion : une simple addition en virgule flottante peut introduire une erreur.

Virgule flottante: limitations

Addition non associative !

- L'addition de nombres réels est associative, i.e.

$$\forall x, y, z \in \mathbb{R}, x+(y+z)=(x+y)+z$$

- En revanche, l'addition de nombres représentés en virgule flottante n'est **pas associative**. La raison de l'absence de cette propriété provient de l'arrondi éventuel qui est effectué en fin d'opération.
- Exemple :
  - $1 + (2^{24} - 2^{24}) = 1$
  - $(1 + 2^{24}) - 2^{24} = 0 \rightarrow$ problème de *swamping*

- L'absence de l'associativité pour l'addition implique que les compilateurs ne peuvent ré-ordonner des instructions qui impliquent des nombres flottants, sous peine de changer les résultats !
- La parallélisation de telles opérations peut également poser problème.

Virgule flottante: limitations

Cancellation : soustraction de nombres proches

- Soit

$$\left.\begin{aligned}\hat{a} &= a\,.\,(1+\Delta_a)\\ \hat{b} &= b\,.\,(1+\Delta_b)\end{aligned}\right\} \quad \hat{x} = \hat{a} - \hat{b}$$

erreur relative petite

typiquement, erreur relative inférieure epsilon machine

$$\frac{|a\,\Delta_a|}{|a|} \leq \epsilon_M \qquad \frac{|b\,\Delta_b|}{|b|} \leq \epsilon_M$$

- Question : que vaut l'erreur relative de la somme ?

$$\begin{aligned}\frac{|x-\hat{x}|}{|x|} &= \frac{|(a-b)-(\hat{a}-\hat{b})|}{|a-b|}\\ &= \frac{|(a-b)-(a+a\,.\,\Delta_a-b-b\,.\,\Delta_b)|}{|a-b|}\\ &= \frac{|a\,.\,\Delta_a-b\,.\,\Delta_b|}{|a-b|}\end{aligned}$$

L'erreur relative peut être amplifiée (et exploser) si a et b sont très proches !!! (dénominateur très petit)

Virgule flottante: limitations

Cancellation : soustraction de nombres proches

- Exemple numérique : soustraction de deux valeurs a et b telles que leur approximation cause une erreur relative inférieure à un *epsilon machine*.

$$a = 1 + \frac{3}{4} . \epsilon_M \quad , \quad \epsilon_a \leq \epsilon_M$$

$$b = 1 - \frac{3}{4} . \epsilon_M \quad , \quad \epsilon_b \leq \epsilon_M$$

$$a - b = \frac{3}{2} . \epsilon_M$$

- Le résultat de la soustraction de a et b cause une erreur relative supérieure à un epsilon machine !

$$a \hat{-} b = \hat{a} - \hat{b} = 1 - 1 = 0 \quad \Rightarrow \quad \Delta_{a-b} = \frac{3}{2} . \epsilon_M$$

$$\epsilon_{a-b} = \frac{|\Delta_{a-b}|}{|a-b|} = 1 > \epsilon_M$$

Références

- **Computer Organization and Design: The Hardware/Software Interface, 4th Edition, D. Patterson and J. Hennessy, Morgan-Kaufmann, 2009**
- **Digital Design: Principles and Practices (3rd Edition), J. Wakerly, Prentice Hall, 2001**
- **Computer Systems: A Programmer's Perspective (2nd edition), R. E. Bryant and D. R. O'Hallaron, Addison-Wesley, 2010**
- **What Every Computer Scientist Should Know About Floating-Point Arithmetic, D. Goldberg, ACM Computing Surveys, Vol 23, No 1, March 1991**
- **A Practical Introduction to Computer Architecture, D. Page, Springer, 2009**

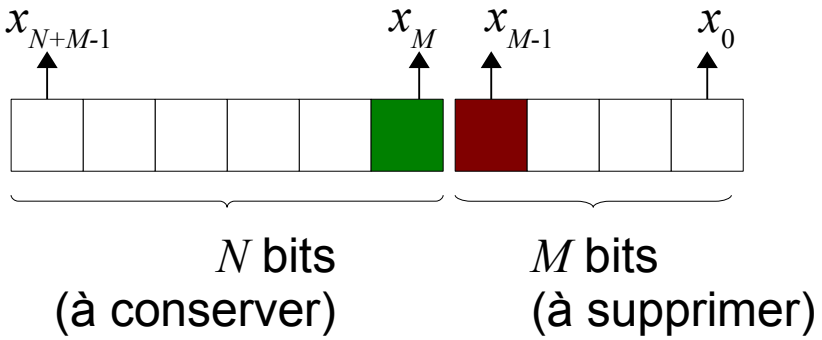
Remerciements

Merci à toutes les personnes qui ont permis par leurs remarques de corriger et d'améliorer ces notes de cours.

Annexes

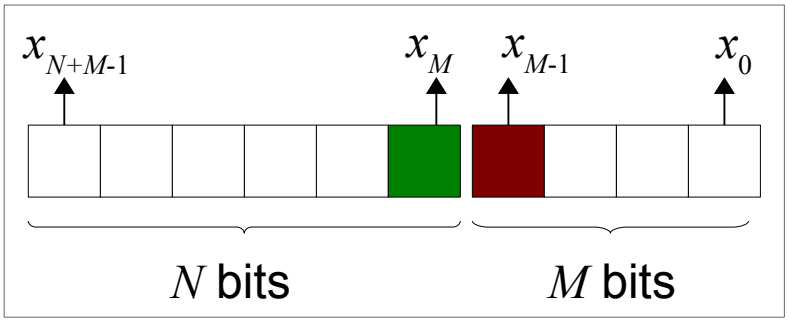
Arrondir en hardware

- Objectif : on veut arrondir à N bits un nombre représenté sur $N+M$ bits. Comment réaliser cet arrondi en hardware ?



- notion de nombre binaire pair
  - nombre dont le bit de poids faible vaut 0
  - exemples
    - `0011.10` est pair
    - `10010.1` est impair

Arrondir en hardware



- vers zéro
  - Oublier les M bits de poids faible ($x_{M-1} \ldots x_0$)
  - Exemples (à arrondir à 4 bits) :

    100**00**1 → 100**0**

    100**11**1 → 100**1**

- vers +∞ (vers -∞ similaire)
  - Oublier les M bits de poids faible.
  - Pour un nombre positif, si au moins un des M bits de poids faible vaut 1, alors ajouter 1 aux N bits de poids fort (et propager le report éventuel)
  - Exemples (à arrondir à 4 bits) :

    100**00**1 → 100**1**

    100**10**0 → 100**1**

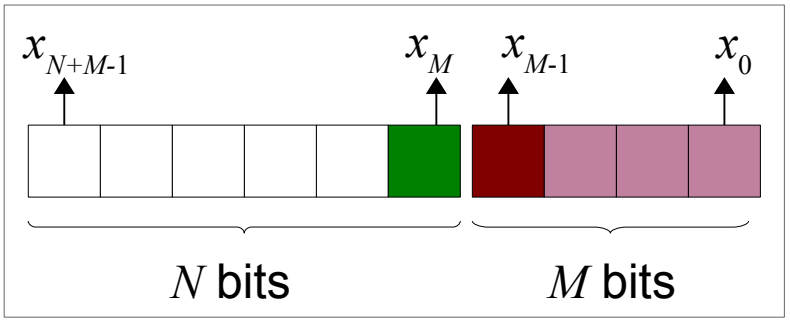
    100**10**1 → 101**0**

    011**11**1 → 100**0**

    -100**00**1 → -100**0**

condition détectable avec une porte OU à M entrées.

Arrondir en hardware



- au plus proche pair
  - Si $x_{M-1} = 0$ → arrondir vers zéro
    - en effet, $x_{M-1} = 0$ signifie plus proche du nombre représentable inférieur en valeur absolue
  - Si $x_{M-1} = 1$ et il existe au moins un $x_i = 1$ $(i < M\text{-}1)$ → arrondir vers ∞
    - en effet, plus proche du nombre supérieur en valeur absolue
  - Sinon, le nombre à arrondir est *exactement situé entre 2 nombres représentables*, i.e. $x_{M-1} = 1$ et $x_i = 0$ $\forall i < M\text{-}1$ (les M bits de poids faible ont la forme `100`…)
    → arrondir vers le plus proche pair.
    - Si x_M = 0 → arrondir vers 0
    - Sinon → arrondir vers ∞