Mathématiques en technologies de l'information 1

Plan du cours

- Ch. 1 Nombres et représentation machine
- Ch. 2 Géométrie vectorielle
- Ch. 3 Introduction à la cryptographie et théorie des codes

Evaluation*: Moyenne arithmétique sur 2 épreuves et 1 rapport de TP

*peut varier selon avancement du cours

Nombres et types de nombres

Il existe divers ensembles de nombres :

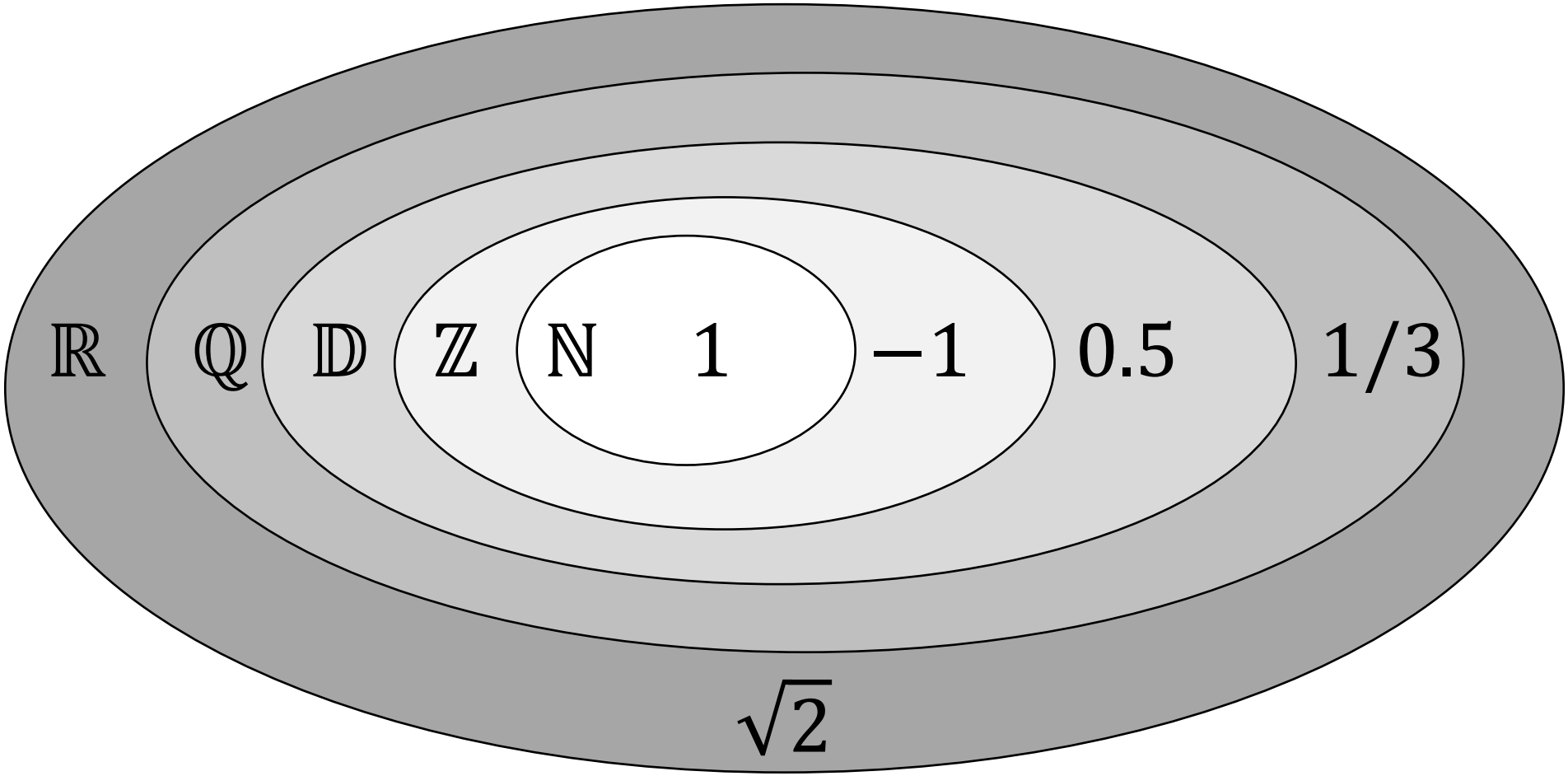
- $\mathbb{N} = \{0,1,2, \dots\}$ les entiers naturels ;
- $\mathbb{N}^* = \{1,2, \dots\}$ les entiers naturels (sans le 0);
- $\mathbb{Z} = \{\dots, -2, -1,0,1,2, \dots\}$ les entiers relatifs ;
- $\mathbb{Z}^* = \mathbb{Z} \setminus \{0\}$;
- $\mathbb{D} = \left\{ \frac{a}{10^n} \middle| \, a \in \mathbb{Z}, n \in \mathbb{N} \right\}$ les nombres décimaux ;
- $\mathbb{Q} = \left\{ \frac{a}{b} \middle| \, a \in \mathbb{Z}, b \in \mathbb{N}^* \right\}$ les nombres rationnels ;
- $\mathbb{R}$, l'ensemble des nombres réels.

Propriétés

Diverses propriétés des nombres

- $\mathbb{D}$: tous les nombres avec un nombre fini de décimales ;
- $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{D} \subset \mathbb{Q} \subset \mathbb{R}$;
- Les inclusions sont strictes (pas égales);

Propriétés des nombres



Proposition: $\sqrt{2} \notin \mathbb{Q}$

Preuve par l'absurde

- Supposons que $\sqrt{2} \in \mathbb{Q}$ - nous allons montrer que cela mène à une contradiction !
- Par hypothèse, il existe une unique fraction **<u>irréductible</u>** $\frac{a}{b} = \sqrt{2}$ avec $a \in \mathbb{N},\ b \in \mathbb{N}^*$ $(PGCD(a,b) = 1)$,
- Mettons l'équation au carré :

$\frac{a^2}{b^2} = 2$, donc $a^2 = 2b^2$

- a^2 est donc PAIR, ce qui implique que a est pair aussi.
- Donc on peut écrire $a = 2k$ avec $k \in \mathbb{N}$,
- En remplaçant ci-dessus, on obtient que $4k^2 = 2b^2$ ou, autrement dit, $b^2 = 2k^2$;
- Par le même raisonnement, b est donc PAIR – la fraction $\frac{a}{b}$ n'est donc pas irréductible car a et b sont divisibles par 2…

Question:

Existe-t-il des nombres en dehors de $\mathbb{R}$?

Réponse : OUI !

Exemple : $\sqrt{2}$ n'est pas défini dans $\mathbb{R}$, alors qu'est-ce ?

L'ensemble des nombres contenant les racines de nombres négatives est l'ensemble des nombres complexes $\mathbb{C}$ (hors du cadre de ce cours) !

Opérateurs

+	Addition
-	Soustraction (addition de l'opposé)
x ou ·	Multiplication
/ ou ÷	Division (multiplication par l'inverse)
% ou mod	Modulo

Rappel

107	10
-100	10
7	

Dividende	***Diviseur***
...	***Quotient***
Reste	

$$107\%10 = \mathbf{7}$$

Propriétés de l'addition

- admet un ***élément neutre*** (0):

$$x + 0 = x, \forall x$$

- est ***commutatif***

$$x + y = y + x, \forall x, y$$

- est ***associatif***

$$(x + y) + z = x + (y + z), \forall x, y, z$$

Propriétés de la multiplication

- admet un ***élément absorbant*** (0) $0 \cdot x = 0, \forall x$
- admet un ***élément neutre*** (1) $1 \cdot x = x, \forall x$
- est ***commutatif*** $x \cdot y = y \cdot x, \forall x, y$
- est ***associatif*** $(x \cdot y) \cdot z = x \cdot (\, y \cdot z)$
- est ***distributif par rapport à +***

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

Question d'intégrité

Les opérateurs sont-ils bien définis ?

En d'autres termes, restent-ils dans leur espaces respectif ?

Exemple : l'addition sur les entiers naturels $\mathbb{N}$:

$$+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

Oui : l'addition de deux entiers naturels donne TOUJOURS un entier naturel.

Idem pour la multiplication !

Pour la soustraction et la division, ce n'est pas le cas !

Contre-exemples

$- : \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

En effet : $5 - 6 = -1 \notin \mathbb{N}$!!

$\div : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$

En effet : $\frac{1}{3} = 0.333 \notin \mathbb{N}$!!

Contre-exemples

$- : \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

En effet : $5 - 6 = -1 \notin \mathbb{N}$!!

$\div : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$

En effet : $\frac{1}{3} = 0.333 \notin \mathbb{N}$!!

Quel intérêt ?

En informatique, selon le langage, l'opérateur est déduit contextuellement – exemple de c#

```
int i = 5;
int j = 2;
double res = i / j; // Quel est la valeur de res?
double res1 = 1.0 * i / j; // Quel est la valeur de res1?
double res2 = i / (1.0 * j); // Quel est la valeur de res2?
double res3 = 1.0 * i / (1.0 * j); // Quel est la valeur de res3?
```

Quel intérêt ?

```java
int i = 5;
int j = 2;
double res = i / j; // Quel est la valeur de res?
double res1 = 1.0 * i / j; // Quel est la valeur de res1?
double res2 = i / (1.0 * j); // Quel est la valeur de res2?
double res3 = 1.0 * i / (1.0 * j); // Quel est la valeur de res3?
```

Réponse:

res = 2.0 !!!!

Ce n'est ni l'arrondi ni la division normale, mais la division ENTIERE !!!

En revanche, on a bien

res1 = res2 = res3 = 2.5

Représentation machine

Problématique

- Sur un ordinateur, on ne dispose que d'une précision ***FINIE***, alors comme représenter des nombres comme $\frac{1}{3}$ ou π ?

Les bases – comment écrit-on un nombre ?

Nous considérons comme un acquis que «cent-deux» s'écrit ***102***, qui peut se décomposer comme suit

1000	100	10	1	
0	1	0	2	
0	+100	+0	+2	=102

L’écriture en base

Il s’agit de la base 10 – cette notation est donc la suivante

10^3	10^2	10^1	10^0	
0	1	0	2	
0	+100	+0	+2	=102

La position détermine la valeur !!!

NOTE IMPORTANTE :
Il a fallu «INVENTER» le symbole 0 pour que cette notation fasse sens (il fut inventé par babyloniens en 300 Av. J.-C.)

Et si nous changions la base

Au lieu de la base 10, prenons la base 8…

$8^3 = 512$	$8^2 = 64$	$8^1 = 8$	$8 = 1$	
0	1	0	2	
0	+64	+0	+2	=66

Notation

Nous noterons $(102)_{10}$ ou $(102)_8$ pour spécifier la base dans laquelle le nombre est représenté.

Si rien n'est précisé, nous considérons que nous travaillons en base 10 !

Donc

$$(102)_{10} = 102$$

$$(102)_8 = (66)_{10} = 66$$

Généralisation – décomposition en base *n*

Soit $a \in \mathbb{N}$ et la base $b \in \mathbb{N}^*, b > 1$, alors

$$a = \sum_{i=0}^{p} a_{p-i} b^{p-i}$$
$$= a_p b^p + a_{p-1} b^{p-1} + \cdots + a_2 b^2 + a_1 b^1 + a_0 b^0$$

Avec $a_i \in \mathbb{N}$ et $a_p \in \mathbb{N}^*$, pour un $p \in \mathbb{N}$ donné.

Représentation symbolique

La position détermine la valeur.

Pour une base $b \in \mathbb{N} \setminus \{0, 1\}$, il faut b-1 symboles plus 1 symbole pour décrire une valeur nulle pour écrire un nombre

<u>Exemples:</u>

Base 10: {***0***,1,2,3,4,5,6,7,8,9}

Base 2: {***0***,1}

Quelle base choisir

Le choix d'une base est un compromis entre nombre de symboles à apprendre et nombre de symboles à écrire pour un chiffre:

Exemple :

$$(10001)_{60} = (12960000)_{10}$$

Ou en base 2 (seulement 2 symboles, mais 24 chiffres à écrire!)

$$(110001011100000100000000)_2$$

Le binaire – base 2

$$X = \sum_{i=0}^{p} x_{p-i} 2^{p-i}$$

Avec $x_i \in \{0,1\}$ et $x_p = 1$, pour un $p \in \mathbb{N}$ donné.

On appelle x_i un ***bit***, x_0 est le ***bit de poids faible*** (le plus à droite) et x_p le ***bit de poids fort*** (le plus à gauche)***.***

Conversion – méthode de division

Objectif : convertir $(X)_{10}$ vers $(X)_b$

Initialisation : $p = 0; D = X; q = 1; r = X;$

Tant que $q\ != 0$, répéter

$q = \lfloor D/b \rfloor$

$r = D\ - q \times b$

$X_p = r$

$D = q$

$p = p + 1;$

FIN

Résultat : $(\mathrm{X})_\mathrm{b} = (X_{FIN} X_{FIN-1} \dots X_0)_b$

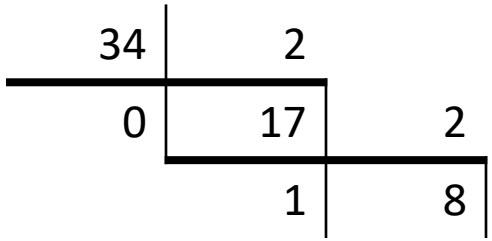
NOTE:

$\lfloor x \rfloor$ est l'opérateur «partie entière» (on ne garde que la partie avant la virgule : $\lfloor 19.72354 \rfloor = 19$)

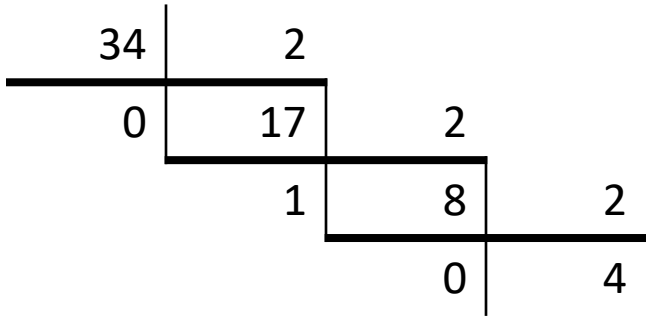
Exemple – Conversion de 34

34	2
0	17

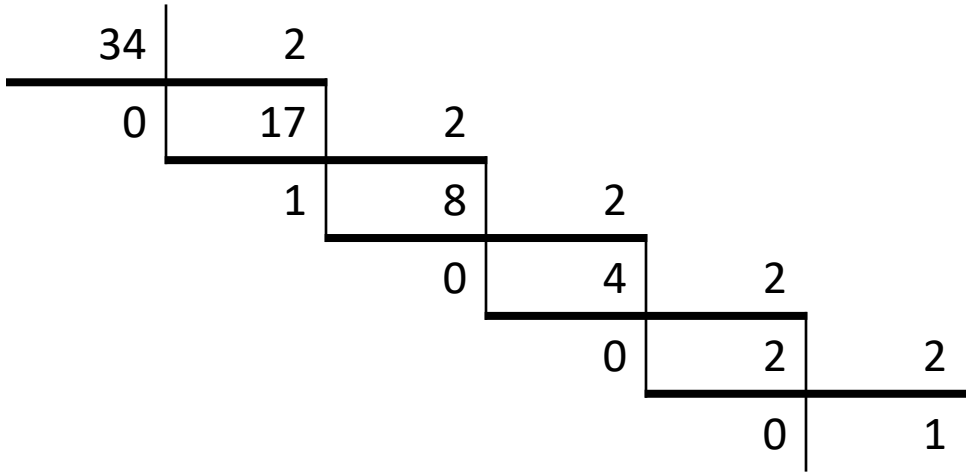
Conversion – méthode de division



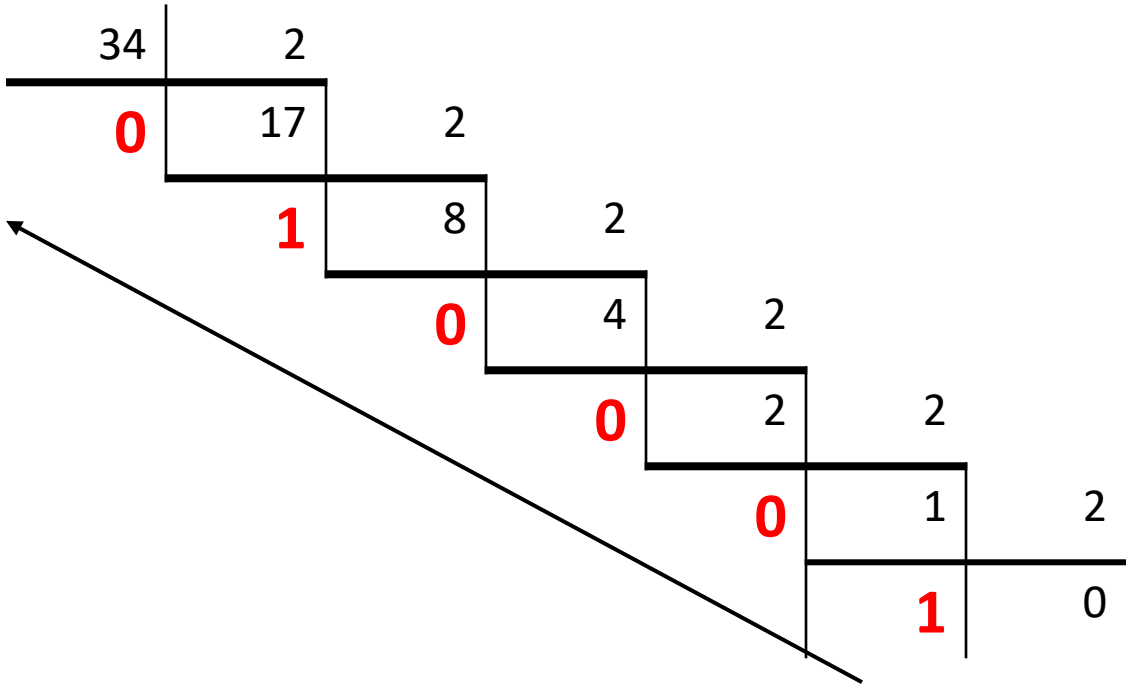
Conversion – méthode de division



Conversion – méthode de division



Conversion – méthode de division



$$(34)_{10} = (100010)_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + \cdots + 1 \cdot 2^1 + 0 \cdot 2^0$$

$$= 32 + 2$$

Conversion – méthode de soustraction

Objectif : convertir $(X)_{10}$ vers $(X)_b$

Initialisation : $R = X$;

Tant que $R\ ! = 0$, répéter

trouver la plus grande puissance p de b telle que $b^p \leq R$:

$p = \max\{i\ |b^i \leq R\}$

$q = \lfloor R/b^p \rfloor$

$X_p = q$ et $X_i = 0, i = 0, \dots, p-1$

$R = R - b^p \times q$;

$p = p + 1$;

FIN

Résultat : $(\mathrm{X})_\mathrm{b} = (X_{FIN} X_{FIN-1} \dots X_0)_b$

Conversion – méthode de soustraction

- Chercher la plus grande puissance de 2 inférieure ou égale au nombre
- Soustraire cette puissance, et recommencer avec le reste, jusqu'à obtenir 0

Méthode par soustraction

Exemple – 34

$(34)_{10}$: la plus grande puissance de 2 précédent est

$$(32)_{10} = 2^5$$

Le reste de la soustraction est

$$(34)_{10} - (32)_2 = 2 = 2^1$$

Le reste est 0 !

Donc

$$\begin{aligned}(32)_{10} &= 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= (100010)_2\end{aligned}$$

Exercices :

Quelle est la représentation binaire de $(173)_{10}$?

Quelle est la plus grande valeur que l'on peut exprimer avec 32 bits (en base 2 donc) ?

Quelle est la valeur maximale pour un nombre en base 2 avec N bits $(11 \ldots .1)_2$?

Exercices :

Quelle est la représentation binaire de $(173)_{10}$?

Réponses :

$$(173)_{10} = (10101101)_2$$

4’294’967’296

$$(11..1)_2 = 2^N - 1$$

Par exemple pour N = 8, la valeur maximale est 255.

Exercices :

Quelle est la représentation binaire de $(17F)_{16}$ en base 60 ?

Indice : passez par la base 10 puis écrivez les symboles de la base 60 avec 2 chiffres, par exemple $(0100)_{60} = 60$!

Exercices :

Quelle est la représentation binaire de $(173)_{10}$?

Réponses :

$$(173)_{10} = (10101101)_2$$

$$(11..1)_2 = 2^N - 1$$

Par exemple pour N = 8, la valeur maximale est 255.

Exercices :

Quelle est la représentation binaire de $(173)_{10}$?

Réponse : $(10101101)_2$

Changement de base

Cas particulier du passage de la base 2 à la base 16 :

1. Rajouter des 0 à gauche jusqu'à ce que le nombre de bits atteigne un multiple de 4

2. Découper les bits par paquets de 4

3. Remplacer chaque paquet de 4 par sa valeur en base 16 (p.ex. $(0000)_2 = (0)_{16}$, $(0111)_2 = 7, (1111)_2 = (A)_{16}$)

Changement de base

Cas particulier du passage de la base 16 à la base 2 :

1. Remplacer chaque élément en base 16 par sa valeur en binaire

2. Accoler les paquets de 4 bits ainsi obtenus (dans le même ordre)

Exemple $(10110)_2 = (?)_{16}$

1. $(10110)_2 = (0001\ 0110)_2$
2. $(0001)_2 = (1)_{16},\ (0110)_2 = (6)_{16}$
3. Le résultat est donc

$$(16)_{16}$$

Binaire	Hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Exemple $(A01)_{16} = (?)_2$

1. $(A)_{16} = (1010)_2$, $(0)_{16} = (0000)_2$, $(1)_{16} = (0001)_2$
2. Le résultat est donc

$$(1010\ 0000\ 0001)_2$$

Représentation des entiers signés Complément à base deux

L'écriture en complément à base deux d'un nombre X sur N bits s'effectue comme suit :

- Si $X \geq 0$, alors le 1er bit (à gauche) est 0, et X s'écrit sur les $N-1$ bits restants
- Si $X < 0$, alors le 1er bit (à gauche) est 1, les $N-1$ bits restants contiennent $2^{N-1} + X$.

Notation: si X est exprimé en complément à deux sur N bits, on notera $(\mathrm{X})_{\overline{2}^N}$.

Exemple avec $N = 4$

- Si $X \geq 0$, alors le 1er bit (à gauche) est 0, et X s'écrit sur les 3 bits restants

Valeur en base 10	Complément à base deux avec 4 bits Noté $(X)_{\overline{2}^4}$
0	$(0 \quad 000)_{\overline{2}^4}$
1	$(0 \quad 001)_{\overline{2}^4}$
2	$(0 \quad 010)_{\overline{2}^4}$
3	$(0 \quad 011)_{\overline{2}^4}$
4	$(0 \quad 100)_{\overline{2}^4}$
5	$(0 \quad 101)_{\overline{2}^4}$
6	$(0 \quad 110)_{\overline{2}^4}$
7	$(0 \quad 111)_{\overline{2}^4}$

Exemple avec $N = 4$

- Si $X < 0$, alors le 1er bit (à gauche) est 1, les 3 bits restants contiennent $8 + X$.

Valeur en base 10	$8 + X$	Complément à base deux avec 4 bits
-8	0	$(1\ \ 000)_{\overline{2}^4}$
-7	1	$(1\ \ 001)_{\overline{2}^4}$
-6	2	$(1\ \ 010)_{\overline{2}^4}$
-5	3	$(1\ \ 011)_{\overline{2}^4}$
-4	4	$(1\ \ 100)_{\overline{2}^4}$
-3	5	$(1\ \ 101)_{\overline{2}^4}$
-2	6	$(1\ \ 110)_{\overline{2}^4}$
-1	7	$(1\ \ 111)_{\overline{2}^4}$

Reconversion en base 10

L'écriture en complément à base deux d'un nombre X sur N bits s'effectue comme suit :

- Si le premier bit est 0, alors $X = (\dots)_2$,
- Si le premier bit est 1, alors $X = -2^{N-1} + (\dots)_2$,

où $(\dots)_2$ sont les $N - 1$ bits restants, après avoir enlevé le bit le plus à gauche (bit de poids fort) en base 2 standard.

Exercices :

Ecrire 36 et -72 en complément à base deux sur 8 bits.

Exercices :

Ecrire 36 et -72 en complément à base deux sur 8 bits.

Réponse :

$$36 = (0\ 0100100)_{\overline{2}^8}$$

$$-72 \Rightarrow (128 - 72) = (56) = (0111000)_2$$

Et donc, le passage en complément à base 2 :

$$-72 = (10111000)_{\overline{2}^8}$$

Exercices :

Que valent $(00101011)_{\overline{2}^8}$ et $(10101011)_{\overline{2}^8}$?

Réponse :

$$(00101011)_{\overline{2}^8} = 43$$

$$(10101011)_{\overline{2}^8} = -(2^7 - 43)$$

Addition en complément à base deux

L'addition de 2 nombres en complément à base deux sur $N = 4$ bits

	$(1)_{10}$	$(0001)_{\overline{2}^8}$
+	$(5)_{10}$	$(0101)_{\overline{2}^8}$
=	$(6)_{10}$	$(0110)_{\overline{2}^8}$

Addition en complément à base deux

L'addition de 2 nombres en complément à base deux sur $N = 4$ bits

	$(3)_{10}$	$(0011)_{\overline{2}^8}$
+	$(-4)_{10}$	$(1100)_{\overline{2}^8}$
=	$(-1)_{10}$	$(1111)_{\overline{2}^8}$

Addition en complément à base deux

L'addition de 2 nombres en complément à base deux sur $N = 4$ bits

	$(-1)_{10}$	$(1111)_{\overline{2}^8}$
+	$(-4)_{10}$	$(1100)_{\overline{2}^8}$
=	$(-5)_{10}$	$(1011)_{\overline{2}^8}$

En effet:

$$(1011)_{2^*} = -2^3 + (011)_2 = -8 + 3 = -5$$

ATTENTION AUX DEPASSEMENTS ! (overflow)

Que se passe-t-il si le résultat est plus grand que 2^{N-1} ?

Or:

	$(5)_{10}$	$(0101)_{\overline{2}^8}$
+	$(5)_{10}$	$(0101)_{\overline{2}^8}$
=	$(10)_{10}$	$(1010)_{\overline{2}^8}$

$$(1010)_{2^*} = -2^3 + (010)_2 = -8 + 2 = -6$$

Pourquoi le complément à base 2

1. Conservation de opérations : «grâce» à l'overflow, l'addition de *int* donne toujours un *int*.

2. Une suite logique «cyclique» (d'où le overflow ou underflow)

-4	-3	-2	-1	0	1	2	3
$(1\ 00)_{\overline{2}^3}$	$(1\ 01)_{\overline{2}^3}$	$(1\ 10)_{\overline{2}^3}$	$(1\ 11)_{\overline{2}^3}$	$(0\ 00)_{\overline{2}^3}$	$(0\ 01)_{\overline{2}^3}$	$(0\ 10)_{\overline{2}^3}$	$(0\ 11)_{\overline{2}^3}$

Note: 3 + 1 revient à écrire (en base 2) $(100)_2$ qui, en complément en base 2, est -4 (on revient au début de l'intervalle).

Int vs uint

Note: les entiers non-signés (*uint*) fonctionnent sur le même principe d'overflow, mais sans les valeurs négatives.

Sur N bits, le «cycle» *int* va de $[-2^{N-1}, 2^{N-1} - 1]$

alors que pour les entiers non-signés *uint* va de $[0, 2^N - 1]$.

Si $N = 8$:
int va de $[-128, 127]$ alors que *uint* va de $[0,255]$!

Exercice

Quel est l'ensembles des nombres que l'on peut écrire en complément à base 2 avec $N = 8$ bits ?

De manière générale, qu'en est-il pour N bits ?

Exercice

Quel est l'ensembles des nombres que l'on peut écrire en complément à base 2 avec $N = 8$ bits ?

De manière générale, qu'en est-il pour N bits ?

Réponse :

$N = 8$:

le plus petit nombre est $(10000000)_{\overline{2}^8} = -2^7 + 0 = -128$

le plus grand nombre est $(01111111)_{\overline{2}^8} = 127$

De manière générale, cette méthode permet d'écrire les nombres dans l'intervalle $[-2^{N-1}, 2^{N-1} - 1]$.

Overflow – un exemple !

4 Juin 1996

- Le vol 501 se termine après 37 secondes par une explosion de la fusée Arianne 5
- L'enquête révèle qu'à l'origine se trouve la conversion d'un double (sur 64bits) vers un entier signé (sur 16 bits),
- Le nombre en question était une mesure de l'accélération horizontale,
- Le code provenait de la fusée Arianne 4 dont l'accélération pouvait être codée sur 16 bit…
- MAIS sur Arianne 5, l'accélération était jusqu'à 5 fois supérieure…
- Verdict : un bug causant une perte de $370Mio…

Et pour les autres ?

Qu'en est-il pour les autres nombres ?

- C'est ok si $x \in [-2^{N-1}, 2^{N-1} - 1]$ avec le complément à 2, mais….
- Si $x \geq 2^{N-1}$, ou
- Si $x < -2^{N-1}$, ou
- Si $x \in \mathbb{Q} \setminus [-2^{N-1}, 2^{N-1} - 1]$,
- Ou pire, $x \in \mathbb{R} \setminus \mathbb{Q}$???

Limitation : précision finie !!!

Quelle que soit l'approche, en précision finie, nous ne pouvons travailler qu'avec des nombres DECIMAUX (nombre de décimales fini) !!

Pour tous les nombres non-décimaux (ou avec trop de décimales), nous travaillerons forcément avec une valeur arrondie !

Principe : écriture scientifique

Pour les nombres très grands (ou très petits), il est trop encombrant d'écrire toutes les décimales.

Exemple : masse de l'électron

$$0.00000000000000000000000000009109\ kg$$

Ce n'est pas très pratique à lire ! Nous utilisons donc l'écriture suivante :

$$9.109\ \times 10^{-31}\ kg = 9.109\ E - 31\ kg.$$

Principe : écriture scientifique

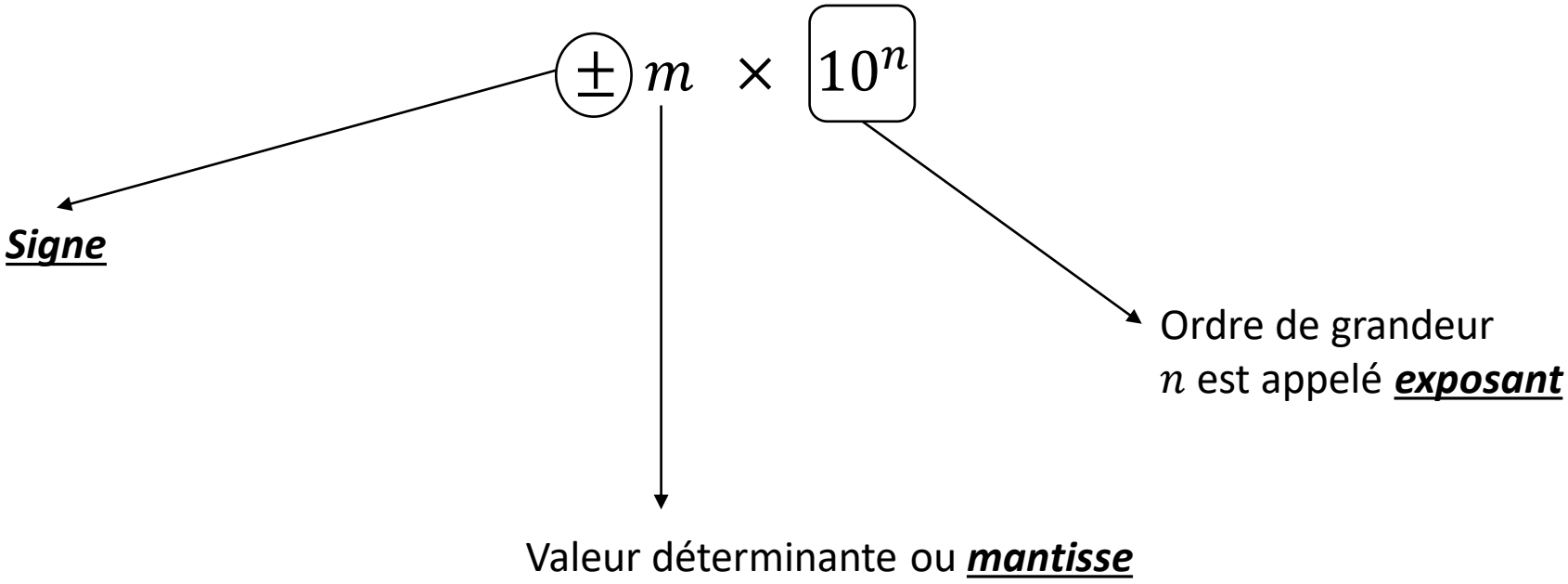
<u>Généralisation :</u>
En écriture scientifique, tout nombre s'écrit sous la forme suivante

$$\pm m \times 10^n$$

où $m \in [1, 10)$ et $n \in \mathbb{Z}$.

Variante:
Il se peut que l'on force $m \in [0.1, 1)$, ce qui donnerait, pour la masse de l'électron, $0.9109\ E-32\ kg$…

Principe : écriture scientifique



Nombres à virgule flottante

En langage machine, on utilise le même principe que l'écriture scientifique :

$$X = (-1)^s \times (1 + m) \times 2^e$$

Standard : Norme IEEE 754
Définit également les modes d'arrondis, calculs, détection des overflow, …

IEE pour ***I****nstitute of* ***E****lectrical and* ***E****lectronics* ***E****ngineers.*

Nombres à virgule flottante en binaire

En 32 bits, le signe est codé sur 1 bit, l'exposant sur 8 bits et la mantisse sur les 23 bits restants

$$X = (-1)^s \times (1 + m) \times 2^e$$

1	2	3	…	8	9	10	11	…	32
$(-1)^s$	e_0	e_1	…	e_6	e_7	m_1	m_2	…	m_{23}

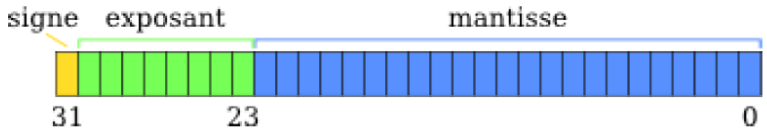
IMPORTANT : En base 2, le bit de poids fort de la mantisse (le plus à gauche) est ***TOUJOURS*** 1 pour un nombre flottant (sauf cas particulier). Il n'est donc pas stocké (cela offre un bit supplémentaire pour la mantisse).

Nombres à virgule flottante

32 bits vs 64 bits

Elément	Bits en 32 bits	Bits en 64 bits
s (signe)	1	1
e (exposant)	8	11
m (mantisse)	23	52

32 bits



64 bits

Encodage des parties (32 bits)

- Signe :

le nombre est positif si le premier bit est 0, négatif s'il est 1.

- Exposant :

Avec 8 bits, on peut encoder des nombres de $[0{,}255 = 2^8 - 1]$.
Afin d'exprimer les exposants négatifs aussi bien que positifs, la norme IEEE-754 définit que

$$e = (e_0 e_1 \dots e_7) - 127$$

- Mantisse :

Le premier bit (caché) est toujours 1 (pour 2^0) et les bits suivants correspondent aux puissances $2^{-1} = \frac{1}{2}, 2^{-2} = \frac{1}{4}, \dots$

Encodage des parties (32 bits)

La formule d'un nombre flottant est donc

$$X = (-1)^s \cdot 2^{(e_0 e_1 \dots e_7)_2 - 127} \cdot (1 + (0.m_1 m_2 \dots m_{23})_2)$$

Où la notation

$$(0.m_1 m_2 \dots m_{23})_2 = 0 \cdot 2^0 + m_1 \cdot 2^{-1} + +\mathrm{m}_2 \cdot 2^{-2} + \cdots + m_{23} \cdot 2^{-23}$$

Cas spéciaux

0 :

- Le nombre encodé uniquement avec des $(0 \dots 0) = +0$ alors que $(10 \dots 0) = -0$ (les cas sont différentiés).

Nombres dénormalisés :

- Si l'exposant est $(e_0 e_1 \dots e_7)_2 = (000000000)_2 = 0$, le premier bit non-stocké de la mantisse est 0, et non 1;
- Le nombre est de la forme $(-1)^s \times (0.m_1 \dots m_{23})_2 \times 2^{-126}$ (nombre dénormalisé).

Infini / NaN (Not a Number):

- Si l'exposant est $(e_0 e_1 \dots e_7)_2 = (11111111)_2 =$255, le nombre est infini si la mantisse est toujours 0, NaN si la mantisse contient des bits non nuls.

Cas spéciaux

Infini :

- Si l'exposant est $(e_0 e_1 \dots e_7)_2 = (11111111)_2$ =255, et la mantisse contient uniquement des 0, on a $\pm\infty$ (selon la valeur du 1er bit)

NaN (Not a Number) :

- Si l'exposant est $(e_0 e_1 \dots e_7)_2 = (11111111)_2$ =255, et la mantisse contient au moins 1 bit de valeur 1, le nombre est «non défini».

32 bits – Valeurs min et max

	Exposant	Mantisse	Valeur base 10 approchée
Min dénormalisé	0000 0000	00…00	1.4×10^{-45}
Max dénormalisé	0000 0000	11…11	$1.17549421 \times 10^{-38}$
Min normalisé	0000 0001	00…00	$1.17549435 \times 10^{-38}$
1	0111 1111	00…00	1.00000000
Successeur de 1	0111 1111	00…01	1.00000012
Max normalisé	1111 1110	11…11	$3.40282346 \times 10^{38}$

Conversion de nombres à virgules

Pour convertir un nombre quelconque $(X, Y)_{10}$ à la norme IEEE 754, il faut

1) Déterminer les signe (détermine le 1er bit),

2) Ecrire les parties entières et décimales sous forme binaire sous la forme $(X.Y)_2$ (arrêter dès que le nombre de bits dépasse 23, soit uniquement pour X s'il s'agit d'un grand nombre, soit $X.Y$. Si le nombre de bits de $X.Y$ < 23, ajouter le nombre de 0 manquants à droite pour obtenir 23 bits !

3) Définir $exp = (127 + \ \#\ de\ bits\ de\ (X)_2\ - 1)$, et convertir $(exp)_2$ pour encoder l'exposant,

Conversion de nombres à virgules

4) Définir $\mathrm{M} = (X.Y)_2$ sur 23 bits, ce qui revient à

- Calculer $(X)_2$ selon l'une des méthodes vues au cours (p.ex. la méthode de la division – éliminez tous les 0 éventuels à GAUCHE)

Si $(X)_2$ a 23 bits ou plus, STOP.

- Calculer $(Y)_2$ avec la méthode de la multiplication (STOP si $(X.Y)_2$ a 23 bits ou si la méthode se termine).

- Garder uniquement les 23 premiers bits OU ajouter 0 à droite jusqu'à avoir 23 bits.

Méthode de multiplication

- Nous avons vu la méthode de division pour décrire tous les bits de la partie entière.

- Pour la partie décimale, il faut utiliser la ***méthode de multiplication*** :

  - Enlever la partie entière
  - Multiplier par 2 et
    - La partie entière correspond au bit à garder
    - Recommencer avec la partie décimale jusqu'à obtenir 0 ou atteindre le nombre de bits souhaités

Exemple: $(0.625)_{10} \Rightarrow (?)_{\frac{1}{2}}$

- Nous noterons $(X)_{1/2}$ pour dénoter l'écriture

$$X = x_0 * 2^{-1} + x_1 * 2^{-2} + \cdots$$

2^{-1}: $0.625 \times 2 = 1.25$
2^{-2}: $0.25 \times 2 = 0.5$
2^{-3}: $0.5 \times 2 = 1.0$
2^{-4}: $0.0 \times 2 = 0.0$
2^{-5}: $0.0 \times 2 = 0.0$
…

Exemple: $(0.625)_{10} \Rightarrow (?)_{\frac{1}{2}}$

- Nous noterons $(X)_{1/2}$ pour dénoter l'écriture

$$X = x_0 * 2^{-1} + x_1 * 2^{-2} + \cdots$$

2^{-1}: $0.625 \times 2 = \mathbf{1}.25$

2^{-2}: $0.25 \times 2 = \mathbf{0}.5$

2^{-3}: $0.5 \times 2 = \mathbf{1}.0$

2^{-4}: $0.0 \times 2 = \mathbf{0}.0$

2^{-5}: $0.0 \times 2 = \mathbf{0}.0$

…

⋮

…

$$(0.625)_{10} = (\mathbf{1}\ \mathbf{0}\ 1\ 0\ 0\ \ldots)_{1/2}$$

ATTENTION

- Pour la méthode de la ***DIVISION***, les bits se lisent dans le sens INVERSE (premier bit = dernier reste !)

- Pour la méthode de la ***MULTIPLICATION***, les bits sont lus dans le sens qu'ils apparaissent (premier bit = première partie entière !)

float IEEE 754 vers $(x)_{10}$

Découper les 32 bits en $(s|e_1 \dots e_8|m_1 \dots m_{23})$

- **Signe :** $(-1)^s$
- **Exposant :** $\mathrm{E} = (e_1 \dots e_8)_2 - 127$
- **Mantisse :**
  - Si $(e_1 \dots e_8) = (00000000)$ alors
    $M = (0.m_1 \dots m_{23})_2,$
  - Sinon
    $M = (1.m_1 \dots m_{23})_2$

Résultat :

$$(x)_{10} = (-1)^s \times M \times 2^E \cong (-1)^s \times M \times 10^{E/3.3}$$

$(x)_{10}$ vers float IEEE 754

Supposons que la mantisse soit dénormalisée, soit

$$(x)_{10} > 1{,}17549421 \times 10^{-38}$$

- **Signe :** $s = 0$ si $(x)_{10} \geq 0$,
  $s = 1$ si $(x)_{10} < 0$
- **Mantisse :**

  Soit $(x)_{10} = (x_0 x_1 \dots x_N . y_1 y_2 \dots)_2$, et $x_0 = 1$ alors
  $M = (x_1 \dots x_N y_1 y_2 \dots)$

(les 23 premiers bits ou, s'il y en a moins, ajouter les 0 à DROITE)

- **Exposant :** $\mathrm{E} = (127 + N - 1)_2$

Résultat :

$$(x)_{10} = [s\ e_1 \dots e_8\ x_1 \dots x_N y_1 \dots]$$

Quelle précision

En 32 bits, la mantisse est codée sur 24 bits (23+1 bit en normalisé).

La valeur maximale de la mantisse est donc

$$(11 \dots 11)_2 = 2^{24} - 1 = 16'777'216 \cong 1.6 \times 10\text{^}7$$

Par conséquent, la précision d'un nombre à virgule flottante (en 32 bits) ne dépasse pas les 7 premiers chiffres!

<u>Conséquence:</u>
En 32 bits, certaines opérations sont «sans intérêt» à cause d'arrondis.

Exemple (c#)

```
float f1 = 1000000.0f;
float f2 = 0.1f;
float res = f1 + f2; // res = 1000000.13
```

```
float f1 = 10000000.0f;
float f2 = 0.1f;
float res = f1 + f2;
bool isEqual = res == 10000000.0f; // true!!!
```

Cas réel – eSwys Comptabilité

- Test avec d'une société fictive ayant un chiffre d'affaires de plusieurs millions (au cumul – aucun montant seul ne dépasse 100'000 Chf)

- Les soldes finaux des comptes (calculés) ne sont pas corrects…

- Car la comptabilité contenant des montants de l'ordre du million ne peut se faire au centime près en 32-bits !!!!

Solution : utiliser des nombres à double précision (sur 64 bits) !!!