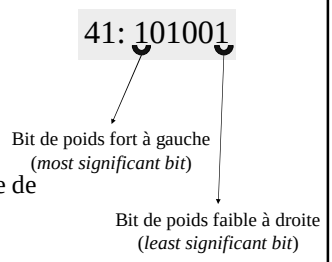
Plan

- **Circuits logiques**
- **Représentation des nombres**
- **Unité Arithmétique et Logique**
- **Notions de temps et de mémorisation**
- **Contrôle et jonction des composants**
- **Evolution des ordinateurs – Historique**
- **Un microprocesseur simple**
- **Programmation d'un microprocesseur**
- **Système complet**
- **Les microprocesseurs actuels**
- **Exploitation de la performance des microprocesseurs**

Représentation des Nombres

- Deux niveaux de valeur: V_{DD} et V_{SS} → 1/0.
- Représentation des nombres en base 2.
- Chiffre binaire = *bit* (***bi****nary digi****t***).
- **Mot n-bits** : nombre de bits utilisés dans les opérateurs d'un processeur.



Nombres Signés

- Trouver une représentation qui affecte peu le coût des circuits de calcul (taille et temps d'exécution) → représentation **transparente** pour principaux opérateurs (addition).
- 1ère solution:
  - Signe = bit de poids fort.
  - **0**: >0, **1**: <0.
- Mot = 4 bits.
- Deux représentations de 0:
  - 0000
  - 1000
- Représentation non transparente.

(-1)		1	0	0	1
(1)	+	0	0	0	1
(-2)		**1**	**0**	**1**	**0**

Nombres Signés

- 2ème solution:
  - Opposé = complément
    -A = A'.
  - A + (-A) = 0.
  - «Complément à 1».

(-1)		1	1	1	0
(1)	+	0	0	0	1
(-0)		**1**	**1**	**1**	**1**

(-2)		1	1	0	1
(1)	+	0	0	0	1
(-1)		**1**	**1**	**1**	**0**

- Mot = 4 bits.
- Deux représentations de 0:
  - 0000 (0)
  - 1111 (-0)
- Toujours signe = bit de poids fort.
- Représentation non transparente.

5		^{1} 0	1	0	1
(-2)	+	1	1	0	1
(??)		**1**	**0**	**0**	**1**	**0**

Nombres signés

0 1 …… 7 -7 -6 …… -1 0 1 …… 7 -7 …… -1 0

0 1 …… 7 8 9 …… 14 15 16 ……22 23 …… 29 30

- Complément à 1 = convention:
  - $0 \le x \le 7$: $X_{CPLT} = x$
  - $-7 \le x \le 0$: $X_{CPLT} = x + 15$ {$\Leftrightarrow X_{CPLT}=15-|x|=1111-x_3x_2x_1x_0=(1-x_3)\,(1-x_2)\,(1-x_1)\,(1-x_0)=|x|'$}
- $x \ge 0$, $y \ge 0$: si pas de dépassement, $x+y \le 7$ et $X_{CPLT}+Y_{CPLT} = x+y$.
- $x \ge 0$, $y \le 0$: $X_{CPLT}+Y_{CPLT} = x + y + 15$
  - $x+y \le 0$: résultat dans [0;15]
  - $x+y > 0$: résultat valide mais dans [16;22] (sur 5 bits)
- $x \le 0$, $y \le 0$: si pas de dépassement, $-7 \le x+y$ et $X_{CPLT}+Y_{CPLT} = 30 + x + y \Rightarrow$ résultat valide mais dans [23;30] (5 bits)

$\Rightarrow$ Résultat addition en complément à 1 obtenu par $Z = (X_{CPLT}+Y_{CPLT})$ *modulo 15*.

Nombres Signés

- Représentation non transparente. Impact sur le circuit ?
- Revenir sur 4 bits (mot du processeur)
  - Résultat $30 \ge Z > 15 \rightarrow$ sur 5 bits: $1\,z_3\,z_2\,z_1\,z_0$ et $z_3\,z_2\,z_1\,z_0 \neq 1111$.
  - $Z \bmod 15 = (10000 + z_3\,z_2\,z_1\,z_0) \bmod 15 = (15 + 1 + z_3\,z_2\,z_1\,z_0) \bmod 15 = (1 + z_3\,z_2\,z_1\,z_0) \underline{\bmod 15}$

  $\Leftrightarrow \underline{1 + z_3\,z_2\,z_1\,z_0}$

$\Rightarrow$ Addition de la retenue, soit **deux étapes ou deux circuits d'addition**, pour une seule opération d'addition.

Nombres Signés

0 3 …… 7 **-8** -7 …… -2 -1 0 3 ……

0 3 …… 7 8 9 …… 14 15 16 19 ……

- 3ème solution:
  - Convention modifiée: $X_{CPLT} = x + \mathbf{16}$.
  - Raisonnement similaire et $Z=(X_{CPLT}+Y_{CPLT})$ *modulo* ***16***.
  - Modulo 16 ⇔ conserver les 4 bits de poids faible
    → **Pas de surcoût pour le circuit**.
  - 15ème valeur (8↔1000):
    - soit +8, soit -8:
    - fixée à **-8** car bit poids fort =1 ⇒ négatif.
  - Opposé: y=-x, $0 \le x \le 7$: $Y_{CPLT} = -x + 16$
    $= 1 + 15 - x = \mathbf{1 + x'}$
    -0 = 0 mais -(-8) = -8

5		1	0	1	0	1
(-2)	+	1	1	1	0
(19)	**1**	**0**	**0**	**1**	**1**

3 ← Modulo 16

  - **Complément à 2** (n bits):
    - $-2^{n-1} \le X \le 2^{n-1}-1$
    - Virgule fixe sur n bits:
      $x=x_{n-1},x_{n-2}x_{n-3}\ldots x_0$
      Opposé: -x + 16
      $= x_{n-1},x_{n-2}x_{n-3}\ldots x_0 + 10,00\ldots0$
      $= x_{n-1},x_{n-2}x_{n-3}\ldots x_0 + 2,00\ldots0$

Codage BCD

- Principales utilisations des calculateurs: calcul scientifique et gestion → calculs en base 10.
- Codage BCD (*Binary Coded Decimal*).
- Codage sur 4 bits des chiffres 0 à 9 → stockage des nombres décimaux dans des registres 32-bits généraux.
- Opérateurs dédiés.
- Facilite les entrées/sorties (périphériques utilisant un chiffre/octet; 4 premiers bits: 0000 pour +, 1111 pour -).

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

Codage des Caractères

- Représentation (et format d'échange) des caractères.
- Standardisation en 1968; IBM garde un format propriétaire (EBCDIC: *Extended Binary Coded Decimal Interchange Code* qui est une extension de BCD).
- ASCII (*American Standard Code for Information Interchange*)
  - codage sur 7 bits,
  - ASCII convient pour Anglais, Latin et quelques autres langues sans caractère spécial; ensuite 8ème bit utilisé pour extensions internationales mais pas de standard,
  - ISO-646-*: internationalisation de ASCII (Américain: ISO-646-US ou *US-ASCII*) sur 8 bits, puis ISO-8859-1 sur 8 bits également.
- Aujourd'hui: Unicode (*Universal Character Set*; UCS ou ISO-10646):
  - proposé pour l'internationalisation du HTML et de tous les protocoles Internet.
  - début Unicode=US-ASCII puis caractères internationaux, sur 16 bits; extension possible.
- *http://www.iso.ch/*

Hexadécimal

- Plus une *représentation* qu'un *codage*.
- Permet d'exprimer une suite binaire de façon compacte.
- Base 16.
- Un chiffre binaire = 4 bits.

Exemple:
$210_{10\text{ ou }d}$
$11010010_{2\text{ ou }b}$
$D2_{16\text{ ou }x}$

Décimal	Binaire	**Hexadécimal**
⋮	⋮	⋮
9	1001	**9**
10	1010	**A**
11	1011	**B**
12	1100	**C**
13	1101	**D**
14	1110	**E**
15	1111	**F**

Représentation des Nombres Réels

- Codage sur n bits:
  - En entier: de 0 à 2^n-1.
  - Représentation décimale à **virgule fixe** (position fixe de la virgule).
- Représentation à **virgule flottante** (*floating-point)*:
  - $Mx2^E$; *M*=Mantisse, *E*=Exposant.
  - Intervalle nettement supérieur à virgule fixe.
  - Représentation normalisée (IEEE 754).
- 32 bits:
  - Virgule fixe:
    - Intervalle de valeurs: au plus 0 à 2^{32}-1.
  - Virgule flottante:
    - 1 bit de signe, 8 bits pour *E*, 23 bits pour *M*.
    - Intervalle des valeurs: environ de 2^{-126} à 2^{+127}.

Problèmes de la Représentation des Nombres Réels

- Nombre fini de valeurs

  $\Rightarrow$ les calculs sont nécessairement *arrondis*

  $\Rightarrow$ erreurs d'arrondis; erreurs de précision.

$\sum_{i=1}^{+\infty} \frac{1}{i}$ converge

```
main()
{
  float i=0.0;
  float somme;

  do {
    i = i + 1.0;
    somme = somme + 1/i;
  } while (1);
}
```

```
i=100000:  i=100000:          12 090860366821289010
i=200000:  i=200000:          12 782761573791504 10001
i=300000:  i=300000:          13 195327758789063 10000
i=400000:  i=400000:          13 491430053710938 10000
i=500000:  i=500000:          13 690693855285645 10101
i=600000:  i=600000:          13 891428718566895 10101
i=700000:  i=700000:          14 071356637573242 11110
i=800000:  i=800000:          14 166624069213867 11110
i=900000:  i=900000:          14 261991500854492 11110
i=1000000: i=1000000:         14 357358932495117 11110
i=1100000: i=1100000:         14 452726364135742 11110
i=1200000: i=1200000:         14 548093795776367 11110
i=1300000: i=1300000:         14 643461227416992 11110
i=1400000: i=1400000:         14 738828659057617 11110
i=1500000: i=1500000:         14 834196090698242 11110
i=1600000: i=1600000:         14 929563522338867 11110
i=1700000: i=1700000:         15 024930953979492 11110
i=1800000: i=1800000:         15 120298385620117 11110
i=1900000: i=1900000:         15 215665817260742 11110
i=2000000: i=2000000:         15 311033248901367 11110
i=2100000: i=2100000:         15 406827087402344 11100
i=2200000: i=2200000:         15 406827087402344 11100
i=2300000: i=2300000:         15 406827087402344 11100
i=2400000: i=2400000:         15 406827087402344 11100
i=2500000: i=2500000:         15 406827087402344 11100
```

Exécution sur Compaq Alpha 21264

Normalisation de la Représentation des Nombres Réels

- Normalisation (IEEE 754)
  → Améliorer la précision/stabilité des résultats et la portabilité des programmes.
  – Représentation à la fois des très grands nombres et des très petits nombres.

Nombre de bits	Taille de *f*	Taille de *e*	E_{min}	E_{max}
32 Simple précision	23	8	-126	+127
64 Double précision	52	11	-1022	+1023

Norme IEEE 754

- Exposant:
  – **Biaisé**: *e*
  – **Non biaisé**: e-E_{max}
  – Intérêt: ordre lexicographique ⇒ comparaison facilitée. Sur 4 bits:
    - biais = 7; -1→0110, 1→1000
    - complément à 2: -1→1111, 1→0001
- Ordre signe-mantisse-exposant:
  – Ordre lexicographique pour nombres positifs ⇒ comparaison facilitée.
  – Exemple (*f* sur 4 bits, *e* sur 4 bits):
    - $1{,}10x2^2$ → 0 1001 1000
    - $1{,}01x2^4$ → 0 1011 0100
- $Emin \leq e\text{-}Emax \leq Emax$:
  – 1 «implicite» pour que mantisse ≥ 1,
  – nombres **normalisés**.

e-E_{max}	*f*	**Nombre représenté**
E_{min}-1	*0*	**±0**
E_{min}-1	*≠0*	$(-1)^s x 0.f \times 2^{Emin}$
$E_{min} \leq \leq E_{max}$	-	$(-1)^s x 1.f \times 2^{e\text{-}Emax}$
E_{max}+1	*0*	±∞
E_{max}+1	*≠0*	**NaN**

- 2 représentations de 0:
  – 0^+ (*s*=0) et 0^- (*s*=1).
  – 0^+: tous les bits à 0
  – La norme impose que le test $0^+=0^-$ soit vrai.
- Infini:
  $+\infty$ (*s*=0)
  $-\infty$ (*s*=1)

Norme IEEE 754

- $e=E_{min}$-1, $f \neq 0$:
  – Nombres **dénormalisés**.
  – Nombres inférieurs à 1.0×2^{Emin} peuvent être représentés
  → *Underflow* graduel: limiter les erreurs d'arrondi avec les petits nombres.
  – Exemple (Mantisse = 4 bits):
    - $A=1{,}0010 \times 2^{Emin}$, $B=1{,}0001 \times 2^{Emin}$
    - Normalement *A-B* résultat non représenté, donc 0, or $A \neq B$.
    - Avec nombres dénormalisés: $A\text{-}B=0{,}0001 \times 2^{Emin}$
  – Exemple:
    - Explorer l'espace des valeurs d'une fonction *f* dont les caractéristiques sont inconnues.

e-E_{max}	*f*	**Nombre représenté**
E_{min}-1	*0*	**±0**
E_{min}-1	*≠0*	$(-1)^s x 0.f \times 2^{Emin}$
$E_{min} \leq \leq E_{max}$	-	$(-1)^s x 1.f \times 2^{e\text{-}Emax}$
E_{max}+1	*0*	±∞
E_{max}+1	*≠0*	**NaN**

- *NaN* (*Not a Number*):
  – $\sqrt{-1}, \frac{0}{0}, \infty - \infty$
  – Evite une interruption du programme (doit être gérée et dépend de l'environnement).
  – Si une des variables est *NaN*, la sortie est toujours *NaN*.

Exceptions

- Par défaut, continuer dans arithmétique étendue:

$$\frac{1}{0^+} = +\infty$$

$$\frac{1}{-\infty} = 0^-$$

$$(+\infty) + (+\infty) = +\infty$$

$$+\infty \times (-\infty) = -\infty$$

$$+\infty - \infty = NaN$$

$$\frac{0}{0} = NaN$$

$$0 \times \infty = NaN$$

$$F(..., NaN, ...) = NaN$$

Dépassement de capacité $\rightarrow NaN$

- En cas d'exception, choix possible entre:
  - Interrompre le programme (gestion exception par l'environnement logiciel).
  - L'utilisateur gère l'erreur dans le programme (des drapeaux signalent les exceptions).
  - Continuer le calcul.
- Exceptions:

Drapeau	Résultat
Underflow	**0, $\pm x_{min}$, nombre dénormalisé**
Overflow	**$\pm\infty$, x_{max}**
Division par 0	**$\pm\infty$**
Résultat *NaN* et opérandes $\neq NaN$	***NaN***
Inexact	**Résultat arrondi**

Quelques Exemples

Normalisé:

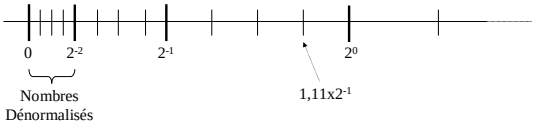
$$s, f, e \rightarrow (-1)^s \times \left(1 + \sum_{i=1}^{23} f_i 2^{-i}\right) \times 2^{e - E_{max}}$$

e-E_{max}	f	Nombre représenté
E_{min}-1	0	**±0**
E_{min}-1	≠0	**$(-1)^s$x0.f x 2^{Emin}**
$E_{min} \leq \quad \leq E_{max}$	-	**$(-1)^s$x1.f x 2^{e-Emax}**
E_{max}+1	0	**±∞**
E_{max}+1	≠0	**NaN**

- 32 bits.
- A= 1 10000010 00110000000000000000000
  - s= 1 (-) , e=130-127=3, f=2^{-3}+2^{-4}=0,125+0,0625
  - A= -1,1875 x 2^3 = -9,5
- A= 1 11111111 00110000000000000000000
  - e=255 → e-E_{max}=E_{max}+1, f≠0
  - A= NaN
- A= 1 00000000 00110000000000000000000
  - s= 1 (-), e=0 → e-E_{max}=E_{min}-1, f=0,1875
  - A= -0,1875 x 2^{-126} (dénormalisé).

Arrondis

- Pour chaque valeur d'exposant, distance double entre valeurs de mantisse.
- Mais, erreur relative constante.
- Exemple: f=2 bits, e=3 bits.



Arrondis

- 4 types d'arrondi fournis par la norme:
  - Vers *0*
  - Vers $+\infty$
  - Vers $-\infty$
  - Au plus près
- Théoriquement, on peut sélectionner le mode d'arrondi, mais choix souvent imposé dans les langages.
- Arrondi **exact** sur les opérations élémentaires ($+,-,\times,/,\sqrt{}$):
  - Résultat = arrondi (calcul effectué avec une précision infinie).
  - Pour obtenir ce résultat, modes étendus non accessibles à l'utilisateur:
    - 32 bits, simple précision: $e \geq 11$, $f \geq 32$, ≥ 6 octets.
    - 64 bits, double précision: $e \geq 64$, $f \geq 15$, ≥ 10 octets.
- L'arrondi exact n'est pas encore garanti pour les fonctions élémentaires (*sin*, *cos*,…)

Multiplication Flottante

- f=3 bits, e=4 bits.
- -7 x 15:
  $x_1 = -7 \rightarrow -1{,}110 \times 2^2 \rightarrow 1\ 1001\ 110$
  $x_2 = 15 \rightarrow 1{,}111 \times 2^3 \rightarrow 0\ 1010\ 111$
- $s = s_1 \oplus s_2$.
- Addition des exposants:
  - $e_1 + e_2 - E_{max}$: chaque exposant contient un biais; après addition, il faut donc en soustraire un biais ($-E_{max}$).

  $\rightarrow e$=1100 soit 5 (biais=7 sur 4 bits; addition sur 5 bits).
- Multiplication entière des mantisses (en parallèle):
  - Décompacter f_1 et f_2 (bit 1 implicite):
    - $f^{\#}_1$=1,110, $f^{\#}_2$=1,111
    - $f^{\#}$=11,010010 sur 8 bits.
  - Normaliser:
    - Revenir à 1,…..
    - Incrémenter exposant de 1.
    - $f^{\#}$=1,1010010
  - Arrondir (au plus près):
    - Résultat: $1,b_1b_2b_3$
    - 11010010 → bits d'arrondi
    - f=101
- Résultat:
  - 1 1101 101
  - Soit -104 (perte de précision)

Précision des Calculs

- La norme n'apporte pas de garantie sur la précision de séquences de calcul.

$$F = \frac{x^2}{\sqrt{1+x^3}}$$

- $x \rightarrow +\infty$, $F \rightarrow +\infty$.
- Pour x grand, x^3 devient $+\infty$, donc F devient 0.
- Lorsque x^2 devient $+\infty$, F devient *NaN*.
- Pas de distinction entre ∞ et dépassement de capacité.

- Un test dont une opérande est *NaN* est faux sauf pour l'égalité.
  $\Rightarrow a \geq b$ et $a < b$ ne sont pas exactement contraires.

$$(-1{,}0111 \times 2^4 + 1{,}0111 \times 2^4) + 1{,}1000 \times 2^{-3}$$
$$= 1{,}1000 \times 2^{-3}$$

- L'addition n'est pas associative.

$$-1{,}0111 \times 2^4 + (1{,}0111 \times 2^4 + 1{,}1000 \times 2^{-3})$$
$$= -1{,}0111 \times 2^4 + 1{,}0111 \times 2^4 = 0$$

- Absorption après décalage.

Interprétation des Erreurs d'Arrondi

- i=100000
  - S_{i-1}= 0 10000010 10000101110100000010110
  - 1/i= 0 01101110 01001111100010110101100
- Aligner les exposants:
  - 10000010-01101110 = 20; décalage de 20 bits à droite de la mantisse
  - Addition + arrondi

1,10000101110100000010110
+0,00000000000000000001010
1,10000101110100000100000

- i=2100000
  - S_{i-1}= 0 10000010 11101100111010101111100 (15,403683)
  - 1/i= 0 01101001 11111111010011100011111
- Aligner les exposants:
  - 10000010-01101001 = 25; décalage de 25 bits à droite de la mantisse
  - Addition + arrondi

1,11011001110101011111100
+0,00000000000000000000000
1,11011001110101011111100