Numération

Compter et calculer :

- *dans l'antiquité*
  - Les assyriens
  - Les romains
- *de nos jours*
  - Le système décimal
- *avec une machine*
  - Le binaire
  - L'octal
  - L'hexadécimal

1500 avant JC : la numération décimale assyrienne

83(dec) =

37(dec) =

exemples:

1:

60:

3600:

73:

4325:

1*3600+12*60+5*1

Numération Romaine

650 Avant JC : la numération décimale étrusque

- I un, V cinq
- X dix, L cinquante
- C cent, D cinq cent
- M mille

Soit les nombres

- XVI = 1 + 5 + 10 = 16
- XIV = 5 - 1 + 10 = 14, *car I est inférieur à V*
- DIX = 10 - 1 + 500 = 509, *car I est inférieur à X*
- MMMMCMXCIX = 10 - 1 + 100 - 10 + 1 000 - 100 + 1 000*4 = 4 999
- 1 975 = MCMLXXV = 1 000 + (1 000 - 100) + 50 + 10×2 + 5

On ne peut pas faire d'opération avec les chiffres romains.

La numération décimale « moderne »

Pour écrire un nombre en base 10 il faut :

- 10 chiffres (ou graphismes) : 0 1 2 3 4 5 6 7 8 9, c'est une convention. On aurait pu choisir n'importe quelle représentation
- Définir la position des chiffres. Chaque position, en partant de la droite vers la gauche définie une puissance de 10 :

Exemple : 32 768 = 3x10 000 + 2x1 000 + 7x100 + 6x10 + 8x1

Rang 4	Rang 3	Rang 2	Rang 1	Rang 0
10^4	10^3	10^2	10^1	10^0
10 000	1 000	100	10	1
3	2	7	6	8

La numération binaire

Pour écrire un nombre en base 2 il faut :

- 2 chiffres (ou graphismes) : 0 1, c'est là aussi une convention. On aurait pu choisir n'importe quelle représentation (des carrés et des ronds …)
- Définir la position des chiffres. Chaque position, en partant de la droite vers la gauche définie une puissance de 2 :

Exemple : 1100 0110 = 1x128 + 1x64 + 0x32 + 0x16 + 0x8 +1x4 +1x2 + 0x1 = 198

on notera : $1100\ 0110_2 = 198_{10}$

Rang 7	Rang 6	Rang 5	Rang 4	Rang 3	Rang 2	Rang 1	Rang 0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	0	0	0	1	1	0

La numération binaire : vocabulaire

- un chiffre binaire est appelé un bit (binary digit)
- Celui le plus à droite est dit de poids faible ou le moins signifiant

  Less Significant Bit = LSB
- Celui le plus à gauche est dit de poids fort ou le plus signifiant

  More Significant Bit = MSB
- Un groupe de 8 bits est un octet ou byte

La numération binaire : conversion D>B

- Convertir 357_{10} en binaire :
  - 2^8 est immédiatement inférieur à 357 ($2^8=256<357<2^9=512$)
  - 357-256=101 2^6 est immédiatement inférieur à 101 ($2^6=64<101<2^7=128$)
  - 101-64=37 2^5 est immédiatement inférieur à 37 ($2^5=32<37<2^6=64$)
  - 37-32=5 2^2 est immédiatement inférieur à 5 ($2^2=4<5<2^3=8$)
  - 5-4=1 $2^0=1$
  - On a donc $357_{10} = 2^8 + 2^6 + 2^5 + 2^2 + 2^0 = 1\ 0110\ 0101_2$

La numération octale

La base est 8, on a donc 8 chiffres de 0 à 7 pour représenter les nombres. Chaque position sera une puissance de 8.

- Soit à convertir 2385_{10} en octal : $4x512 + 5x64 + 2x8 + 1x1 = 4521_8$
  - 512 < 2385 < 4096 soit 4 x 512 = 2048 < 2385. Il reste 2385 – 2048 = 337
  - 64 < 337 < 512 soit 5 x 64 = 320 < 337. Il reste 337 – 320 = 17
  - 8 < 17 < 64 soit 2 x 8 = 16 < 17. Il reste 17 – 16 = 1
  - $1 = 8^0$

On a donc

Rang 6	Rang 5	Rang 4	Rang 3	Rang 2	Rang 1	Rang 0
8^6	8^5	8^4	8^3	8^2	8^1	8^0
262 144	32 768	4096	512	64	8	1
0	0	0	4	5	2	1

La numération hexadécimale

La base est 16 on a donc 16 graphismes : les 10 chiffres plus 6 lettres de A à F. Chaque position sera une puissance de 16.

Soit à convertir 5432_{10} en hexadécimal : 1x4096 + 5x256 + 3x16 + 8x1 = 1538_{16}

- 4096 < 5432 < 65 536 soit 1 x 4096 Il reste 5432 – 4096 = 1336
- 256 < 1336 < 4096 soit 5 x 256 = 1280 Il reste 1336 – 1280 = 56
- 16 < 56 < 256 soit 3 x 16 = 48 Il reste 56 – 48 = 8
- 8 = 8 x 1

On a donc :

Rang 5	Rang 4	Rang 3	Rang 2	Rang 1	Rang 0
16^5	16^4	16^3	16^2	16^1	16^0
1 048576	65536	4096	256	16	1
0	0	1	5	3	8

En résumé

- On reconnait, par convention, les chiffres dans les différentes bases en les regroupant en :
  - 4 rangs pour les binaires
  - 3 rangs pour les octaux
  - 2 rangs pour les hexadécimaux
- Exemple 689_{10}
  - $2^9 + 2^7 + 2^5 + 2^4 + 2^0$ = 0010 1011 0001
  - $1x8^3 + 2x8^2 + 6x8^1 + 1x8^0$ = 001 261
  - $2x16^2$ + B(11)$x16^1 + 1x16^0$ = 02 B1
- Pour passer directement du binaire à l'octal on regroupe le binaire par 3 chiffres, les nombres formés sont les valeurs octales

  001 010 110 001 = 0 1 2 6 1
- Pour passer directement du binaire à l'hexadécimal on regroupe le binaire par 4 chiffres, les nombres formés sont les valeurs hexadécimales

  0000 0010 1011 0001 = 0 2 B(11) 1

Addition binaire simple

Les nombres sont représentés sur 8 bits

- Soit à calculer : $132_{10} + 120_{10} = 252_{10}$

```
      1000 0100
  +   0111 1000
  =   1111 1100
```

- Soit à calculer : $70_{10} + 122_{10} = 192_{10}$

```
Retenues   1111 11
           0100 0110
       +   0111 1010
       =   1100 0000
```

Addition binaire simple : suite

Soit à calculer : $179_{10} + 133_{10} = 312_{10}$

```
Retenues  1      111
                1011 0011
          +     1000 0101
          =   1 0011 1000
```

On remarque que la dernière additions de deux nombres de 8 bits donne un nombre sur 9 bits. Il y a un débordement de capacité qu'il faudra gérer. Si le résultat ne peut être évalué que sur 8 bits il sera **faux**.

Addition octale

- Soit à calculer : $14_8 + 23_8 = 37_8$ ($12_{10} + 19_{10} = 31_{10}$)

```
        14
+       23
=       37
```

- Soit à calculer : $14_8 + 27_8 = 43_8$ ($12_{10} + 23_{10} = 35_{10}$)

```
Retenues  1
          14
+         27
=         43
```

- Soit à calculer : $25_8 + 67_8 = 43_8$ ($21_{10} + 55_{10} = 76_{10}$)

```
Retenues 11
          25
+         67
=        114
```

Addition hexadécimale

- Soit à calculer : $1A_{16} + C3_{16} = DD_{16}$ ($26_{10} + 195_{10} = 221_{10}$)

```
      1A
+     C3
=     DD
```

- Soit à calculer : $1A_{16} + B7_{16} = D1_{16}$ ($26_{10} + 183_{10} = 209_{10}$)

```
Retenues   1
           1A
+          B7
=          D1
```

- Soit à calculer : $5F_{16} + BD_{16} = 11C_{16}$ ($95_{10} + 189_{10} = 284_{10}$)

```
Retenues  11
           5F
+          BD
=         11C
```

Compléments logique / vrai

- Le complément à un d'un nombre n de rang r est la valeur qu'il faut ajouter à ce nombre pour obtenir la valeur maximale dans son rang. Ce complément est aussi nommé « complément logique » noté $C_1(n)$
  - Exemple : quel est le complément à un de 1010
    - Son rang est 3, la valeur maximale dans le rang est 1111
    - 1111 – 1010 = 0101
    - **0101 est le complément à un de 1010 ou $C_1(1010)=0101$**

  On remarque que pour obtenir le complément à un il suffit de changer les valeurs 0 en 1 et les valeurs 1 en 0
- Le complément à deux est le complément à un + 1. Ce complément est aussi nommé « complément vrai » noté $C_2(n)$
  - Exemple : quel est le complément à deux de 1010
    - Le complément à un est 0101
    - 0101 + 1 = 0110
    - **0110 est le complément à deux de 1010 ou $C_2(1010)=0110$**
- **Complément à deux du complément à deux.**
  - Exemple : le complément à deux de 0110 est 1001+1=1010 soit $C_2(C_2(n))=n$

Signe des nombres binaires

- Par convention les nombres positifs on le MSB positionné à 0. Un nombre de 4 bits sera codé sur 5 bits avec le MSB à 0

  Exemple : on note $10_{10} = 1010_2$ en valeur absolue. On écrira $+10_{10} =$ **0**1010_2

  Le plus souvent on représente les nombres par groupes de huit bits (octet ou byte). . On écrira alors $+10_{10} =$ **0**$000\ 1010_2$

- Les nombres négatifs sont représentés par le complément à 2 de leur valeur signée.

  Exemple : on représente -10_{10} par son complément à deux soit

  complément à un 10101 ou en 8 bits : 1111 0101

  complément à deux 10101 + 1 = 10110 1111 0110

- Vérification : $10_{10} - 10_{10} = 0\ 1010 + 1\ 0110 = 10\ 0000$

  L'addition de ces deux nombres de 5 bits donne un résultat sur 6 bits. A condition d'ignorer le MSB (ce qui est une convention) on obtient bien le résultat attendu. On notera cette opération $n + C_2(n) = 0$

- En 8 bits on a : 0000 1010 + 1111 0110 = 1 0000 0000. Même remarque que ci-dessus pour le MSB est repoussé en neuvième position.

Addition binaire de nombres signés

- Exemple 1
  - **0**011 0100 + **0**100 1001 = **0**111 1101 (52_{10} + 73_{10} = 125_{10})
- Exemple 2
  - **0**111 0011 + **0**100 0101 = **1**011 1000 (115_{10} + 69_{10} # **-** 72_{10})

L'addition de ces deux nombres signés (positivement) représentés chacun par un octet est fausse car sur 8 bits elle donne un nombre négatif (MSB=1). Il y a là aussi débordement de capacité car la retenue a été placée sur le huitième bit qui est le bit de signe. Le résultat correct est codé sur 9 bits soit : **0** 1011 1000 = 184_{10}

Là aussi, si le résultat ne peut être évalué que sur 8 bits il sera **faux**.

Soustraction binaire

La soustraction A-B est ramené à l'addition de A+C_2(B)

Exemple $55_{10} - 58_{10}$

$+55_{10}$ = **0**011 0111 nombre signé représenté sur 8 bits

-58_{10} $+58_{10}$ = **0**011 1010

complément à 1 : 1100 0101

complément à 2 : 1100 0110

-58_{10} = **1**100 0110 nombre représenté par un octet

```
          11   (retenues)
     0011 0111
+    1100 0110
=    1111 1101
```

On constate qu'il n'y a pas de débordement (aucune retenue à gauche). Le résultat est un nombre négatif puisque son MSB=1. On va calculer son C_2 pour avoir sa valeur : C_2(1111 1101) = 0000 0010 + 1 = 0000 0011 = 3_{10}

Le résultat est bien **-** 3_{10}

Soustraction binaire : suite

- Exemple : $55_{10} - 42_{10}$

  $+55_{10}$ = **0**011 0111

  $- 42_{10}$ $+42_{10}$ = **0**010 1010

  complément à 1 : 1101 0101

  complément à 2 : 1101 0101 + 1 = 1101 0110

  -42_{10} = 1101 0110

```
  1  111  11        (retenues)
     0011 0111
+    1101 0110
= 1  0000 1101
```

  Il y a débordement de capacité mais le résultat est correct si on néglige le neuvième bit. En effet le huitième bit est 0 donc le résultat est positif : 0000 1101 = 13_{10}

  Ceci se produit lorsque les deux dernières retenues sont à 1

- Exemples : $121_{10} - 116_{10}$ puis $119_{10} - 64_{10}$

Soustraction binaire : suite

- Exemple : - 121_{10} - 42_{10}

 $-121_{10} = C_2($ **0**111 1001) = 1000 0111

 $- \ 42_{10} = C_2($ **0**010 1010) = 1101 0110

```
  1      11      (retenues)
     1000 0111
+    1101 0110
= 1  0101 1101
```

 Là aussi il y a débordement de capacité. Le résultat est faux si on ne peut l'évaluer que sur 8 bits. Il serait positif et vaudrait 0101 1101 = 93_{10}

 En revanche si on considère le résultat sur 9 bits. Le nombre est bien négatif (MSB=1) et vaut -163 (C_2(1 0101 1101) = 163)

 Ceci se produit lorsque les deux dernières retenues sont `10`.

- Exemple : - 125_{10} - 64_{10}

Soustraction binaire : fin

- Exemple : 38_{10} - 90_{10}

38_{10} = **0**010 0110

\- 90_{10} = C_2(**0**101 1010) = 1010 0110

```
      1     11      (retenues)
     0010 0110
+    1010 0110
=    1100 1100
```

C_2(1100 1100) = 0011 0100 = 52_{10}. Le résultat est bien -52

- Exemple : 55_{10} - 96_{10}

Nombres binaires décimaux

- Les nombres après la virgule sont des puissances négatives de 2. A droite de la virgule on a :

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
½	¼	1/8	1/16	1/32	1/64
0.5	0.25	0.125	0.0625	0.03125	0.015625

- Exemples :
  - $2.625 = 10.101$ soit $1x2^{1} + 0x2^{0}$. $1x2^{-1} + 0x2^{-2} +1x2^{-3} = 2 + 0$. $0.5 + 0 + 0.125$
  - Convertir 0.34375 en binaire
    - 0.34375x2=0.6875 < 1 on pose 0
    - 0.6875x2=1.375 > 1 on pose 1 et on retient 1 du nombre suivant
    - 0.375x2=0.75 < 1 on pose 0
    - 0.75x2=1.5 > 1 on pose 1 et on retient 1 du nombre suivant
    - 0.5x2 = 1 on pose 1 et on a atteint une valeur exacte

    Soit $0.34375 = 0.01011 = 0$. $0x2^{-1} + 1x2^{-2} + 0x2^{-3} + 1x2^{-4} + 1x2^{-5}$

Nombres binaires décimaux : suite

- Montrer que en binaire avec 10 bits pour la partie décimale :

$$\pi \sim 11.0010010001$$

- Quelle est la précision obtenue ?
- On donne : $\pi \sim 11.0010\ 0100\ 0011\ 1111$

  Quelle est l'amélioration de la précision ?

Nombres binaires en virgule flottante

- Représentation scientifique décimale : $0.34375 = 3.4375 \times 10^{-1}$. On notera :
  - Mantisse : le nombre à gauche de la multiplication soit 3.4375
  - Exposant : la puissance de 10 soit -1
- Exemple : 4.5_{10}

$4_{10}=100_2$

$0.5_{10}=0.1_2$

$4.5_{10}=100.1_2=1.001x2^2$

- Soit :

$1.001=2^0+2^{-3}=1.125$

$1.125x2^2=4.5$

Nombres binaires en virgule flottante

- En binaire on écrira les nombres selon la **norme IEEE 754** sous la forme

$$(-1)^{signe} \times 1.M \times 2^{E}$$

  - M = mantisse, n'a pas le sens commun de la notation scientifique. C'est en réalité la partie fractionnaire de la mantisse.
  - E = exposant, précise la valeur et le sens du décalage de la virgule.
- Le nombre de bits alloués à la mantisse et à l'exposant est variable suivant la longueur de représentation :
  - 32 bits (4 octets) MSB=signe, 8 bits d'exposant et 23 bits de mantisse
  - 64 bits (8 octets) MSB=signe, 11 bits d'exposant et 52 bits de mantisse
  - Au-delà chaque constructeur (Cray, Fujitsu …) adopte ses propres conventions
- Remarque : Dans la norme le 1 de la mantisse est implicite, il n'est donc jamais affecté, ce qui fait gagner un bit. En 32 bits la mantisse sera donc de 24 bits.

Nombres binaires en virgule flottante : suite

- Représentation biaisée ou par excès

  Dans la représentation binaire déjà vue précédemment les nombres ne peuvent pas être ordonnés car les négatifs représentés par leur complément à deux sont en ordre inverse et supérieurs aux positifs. Il existe une méthode de représentation qui consiste à décaler tous les nombres d'une valeur donnée qui est appelée le biais ou l'excès.

- Exemple sur 8 bits signés

  Le plus grand nombre positif est 127 = 0111 1111

  Le plus petit nombre négatif est -127 = 1000 0001 = C2(0111 1111)

  On a donc l'ordre suivant :

```
 127 = 0111 1111
 126 = 0111 1110
   1 = 0000 0001
   0 = 0000 0000
  -1 = 1111 1111
-126 = 1000 0010
-127 = 1000 0001
```

Nombres binaires en virgule flottante : suite

- Exemple sur 8 bits en représentation biaisée à 127

  Les nombres vont varier de 0 (-127 + 127) à 255 (128 + 127) sans se soucier du bit de signe puisque ils sont tous positifs.

  On a donc l'ordre suivant :

```
 128 + 127 = 255 = 1111 1111
 127 + 127 = 254 = 1111 1110
 126 + 127 = 253 = 1111 1101
   2 + 127 = 129 = 1000 0001
   1 + 127 = 128 = 1000 0000
   0 + 127 = 127 = 0111 1111
  -1 + 127 = 126 = 0111 1110
  -2 + 127 = 125 = 0111 1101
-126 + 127 = 1   = 0000 0001
-127 + 127 = 0   = 0000 0000
```

  On observe qu'en représentation biaisée à 127 les nombres de -127 à 128 sont ordonnés de `0000 0000` à `1111 1111`.

Nombres binaires en virgule flottante : suite

- Exemple de représentation normalisée de l'exposant en 32 bits
  - La norme prévoit un biais de 127 à ajouter à la valeur du décalage de la virgule
  - L'exposant pour l'infiniment grand ou « not a number » est 1000 0000 = 128_{10}
    Sa valeur biaisée sera 1111 1111 = 255_{10} = 128 + 127
    L 'exposant maxi sera donc 0111 1111 = 127_{10} valeur biaisée 1111 1110 = 254_{10}
  - L'exposant pour l'infiniment petit ou « denormalized » est 1000 0001 = -127_{10}
    Sa valeur biaisée sera 0000 0000 = 0
    L'exposant mini sera donc 1000 0010 = -126_{10} valeur biaisée 0000 0001 = 1
  - L'exposant nul 0000 0000 a pour valeur biaisée 0111 1111 = 127_{10}
  - Le zéro ne pouvant être représenté car 1 est implicite dans la mantisse on convient que lorsque E=M=0 la valeur représentée est 0

Les exposants biaisés varient donc de 1 à 254 avec 0 et 255 valeurs d'erreurs

Nombres binaires en virgule flottante : suite

- Exemples
  - 525.5_{10} = 10 0000 1101.1 = 1. 0000011011 x 2^9
    Attention : 2^9 est une représentation, ce n'est pas une valeur numérique. L'exposant 9 est un décalage positif de 9 positions

  En représentation biaisée l'exposant E = 9 + 127 = 136 = 1000 1000

  La mantisse (relative) est 0000011011

  Le bit de signe vaut 0 (positif)

  Représentation IEEE 754 `0 10001000 0000011 01100000 00000000`

  `S EEEEEEEE M------- 23 bits -------M`

  - -6.625 Le signe étant donné par le MSB on ne s'intéresse qu'à la valeur positive soit 6.625 = 110.1010 = 1.101010 x 2^2

  En représentation biaisée l'exposant E = 2 + 127 = 129 = 1000 0001

  La mantisse (relative) est 101010

  Le bit de signe vaut 1 (négatif)

  Représentation IEEE 754 `1 10000001 1010100 00000000 00000000`

  `S EEEEEEEE M------- 23 bits -------M`

Nombres binaires en virgule flottante : suite

- Quel est le valeur décimale de la représentation IEEE 754 suivante :

```
1 01111101 0110000 00000000 00000000
S EEEEEEEE M------- 23 bits -------M
```

  - Le signe est négatif puisque le MSB=1
  - L'exposant E = x + 127 = 125 soit x = -2
  - La mantisse relative est 011 la mantisse réelle est 1.011
  - Le décalage négatif de deux positions donne enfin 0.01011 = 0.34375_{10}
  - Le résultat est **- 0.34375**

Nombres binaires en virgule flottante : fin

- Montrer que la valeur décimale de la représentation IEEE 754 suivante :

```
 0 1000 0110 100 1011 1001 0101
 S EEEE EEEE M-------- 23 bits -------M
```

- est : 203.5820 3125