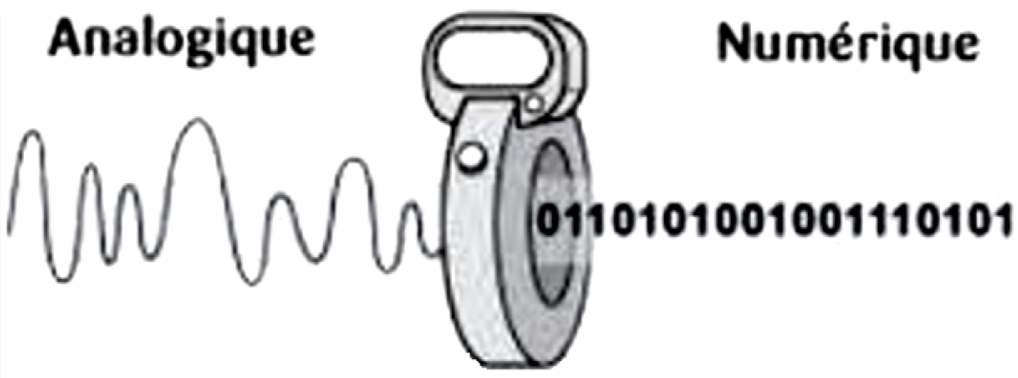
|  |  |  |
| --- | --- | --- |
| Numérique et Sciences Informatiques | | |
| 5h | **Numérique V2** |  |
| **Objectif** : coder des informations sous forme numérique. | | |
| **Matériel**: | | |

**De l'analogique au numérique**



Lorsque les informations sont **analogiques** il faut les mettre sous forme **numérique.**

|  |  |  |
| --- | --- | --- |
| **Information :** | **Exemple :** | |
| Analogique | Température | Une multitude d'état : -10,7°C, 12,5°C, 25°C ... |
| Numérique | Température | 25°C = 0001 1001 |

L'invention du transistor (composant ayant deux états : courant bloqué ou courant passant) au milieu de XXème siècle a entrainé le développement de l'électronique **numérique**.

Les informations **numériques** sont constituées de plusieurs **bits** sous forme de code.

Un **bit** (contraction de **b**inary dig**it**) est une information ayant 2 états : 0 (courant bloqué) ou 1 (courant passant).

**Codage Décimal (base 10) et Binaire (base 2)**

Notre système de comptage est décimal car l'homme à commencé à compter sur ses 10 doigts de 1 à 10 (ou de 0 à 9).

Les nombres s’écrivent en dénombrant les unités, dizaines, centaines…, c’est à dire les puissances successives de 10.

Exemple : le nombre 5216 représente 5 milliers : 5 x 1000 = 5 x 10 3

+ 2 centaines : 2 x 100 = 2 x 10 2

+ 1 dizaine : 1 x 10 = 1 x 10 1

+ 6 unités : 6 x 1 = 6 x 10 0

*poids*

*base*

Pour un nombre décimal à n chiffres : "cx" compris entre 0 et 9 →

(N)10 = (cn-1 … c2 c1 c0)10 = cn-1x10n-1 + … + c2x102 + c1x101 + c0x100

Pour un nombre **binaire** à n chiffres : "bx" compris entre 0 et 1 →

**(N)10 = (bn-1 … b2 b1 b0)2 = bn-1x2n-1 + … + b2x22 + b1x21 + b0x20**

Poids des bits d'un octet : pour coder de 0 à 255 (255 = 28- 1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| 27= 128 | 26= 64 | 25= 32 | 24= 16 | 23= 8 | 22= 4 | 21= 2 | 20 = 1 |

Poids des bits si un octet ne suffit pas : pour coder de 0 à 65535 (65535 = 216- 1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| b15 | b14 | b13 | b12 | b11 | b10 | b9 | b8 |
| 215= 32768 | 214= 16384 | 213= 8192 | 212= 4096 | 211= 2048 | 210= 1024 | 29= 512 | 28 = 256 |

On constate que le bit de droit est le bit qui à le **poids le plus faible** : **LSB** (Least Significant Bit).

On constate que le bit de gauche est le bit qui à le **poids le plus fort** : **MSB** (Most Significant Bit).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| (1101 1001)2 → (N)10  b7.27 + b6.26 +b5.25 + b4.24 + b3.23+ b2.22 + b1.21 + b0.20  = 1.27 + 1.26 + 1.24 + 1.23+ 1.20 = 128 + 64 + 0 + 16 + 8 + 0 + 0 + 1 = (217)10 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| (157)10 → (N)2  Méthode par décomposition :  (157)10 = 128 + 16 + 8 + 4 + 1 = 1.b7 + 0.b6 + 0.b5 + 1.b4 + 1.b3 + 1.b2 + 0.b1 + 1.b0 → (1001 1101)2 | | | | | | | |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| (157)10 → (N)2  Méthode par complémentation :  157⊥2  1 78⊥2  0 39⊥2  1 19⊥2  1 9⊥2  1 4⊥2  0 2⊥2  0 1  (157)10 = (1001 1101)2  157=(((((((0x2+**1**)x2+**0**)x2+**0**)x2+**1**)x2+**1**)x2+**1**)x2+**0**)x2+**1** | | | | | | | |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

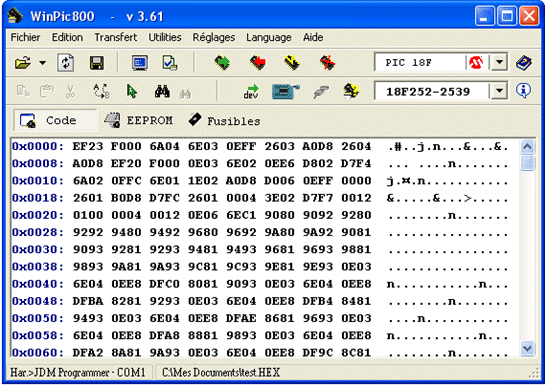
* Combien de valeurs peut-on coder avec 3 bits ? 23 = 8
* Combien de valeurs peut-on coder avec 16 bits ? 216 = 65536
* (0100 1101)2 → (N)10 64 + 8 + 4 + 1 = 77
* (0001 0100 0001)2 → (N)10 256 + 64 + 1 = 321
* (67)10 → (N)2 64 + 2 + 1 -> 0100 0011
* (201)10 → (N)2 128 + 64 + 8 + 1 -> 1100 1001

**A faire à la maison**

* (0010 0110)2 → (N)10 32 + 4 + 2 = 38
* (1111 1111)2 → (N)10 28 = 255
* (148)10 → (N)2 128 + 16 + 4 -> 1001 0100
* (1023)10 → (N)2 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 -> 0011 1111 1111

(Autre méthode : 1023 = 1024 – 1 -> 10000000000-1)

**Codage Hexadécimal (base 16)**

Code machine d'un programme informatique : 

Le codage hexadécimal permet de condenser l’écriture des informations codées en binaire.

16 = 24 → chaque chiffre hexadécimal remplace un groupe de 4 chiffres binaires.

En hexadécimal pour les nombres de 10 à 15, on utilise les lettres suivantes :

**A**(=10), **B**(=11), **C**(=12), **D**(=13), **E**(=14) et **F**(=15).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Décimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Binaire | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| **Hexa** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Binaire : | 1001 | 0101 | 1010 | 1111 |
| Décimal : | 8+1=9 | 4+1=5 | 8+2=10 | 8+4+2+1=15 |
| Hexadécimal : | 9 | 5 | A | F |
| (1001 0101 1010 1111)2 → (N)16 = (95AF)16 | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hexadécimal : | 0 | 1 | C | 9 |
| Décimal : | 0 | 1 | 12 | 9 |
| Binaire : | 0000 | 0001 | 1100 | 1001 |
| Poids : | 163= 4096 | 162= 256 | 161= 16 | 160 = 1 |
| (01C9)16 → (N)10 1x256 + 12x16 + 9x1 = (457)10  Ou  (01C9)16 → (N)2 → (N)10  0000 0001 1100 1001 → 256 + 128 + 64 + 8 + 1 = (457)10 | | | | |

* (1000 1010)2 → (N)16 8A
* (1D8)16 → (N)2 0001 1101 1000
* (67)10 → (N)16 64 + 2 + 1 → (0100 0011)2 → (43)16
* (DE15)16 → (N)10 13 x 4096 + 14 x 256 + 1 x 16 + 5 x 1 = (56853)10

**A faire à la maison**

* (0100 0111)2 → (N)16 47
* (2016)16 → (N)2 0010 0000 0001 0110
* (138)10 → (N)16 128 + 8 + 2 → (1000 1010)2 → (8A)16
* (5F4)16 → (N)10 5 x 256 + 15 x 16 + 4 x 1 = (1524)10

**Tableau des unités**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Préfixes en base 10** | | | **Préfixes en base 2** | | |
| 1 kilo €uro (k) | 103 € | 1000 € | 1 kilo octet (Ko) | 210 octets | 1024 o |
| 1 méga €uro (M) | 106 € | 1000 k€ | 1 méga octet (Mo) | 220 octets | 1024 Ko = 1024x1024o = 1 048 576 o |
| 1 giga €uro (G) | 109 € | 1000 M€ | 1 giga octet (Go) | 230 octets | 1024 Mo = 1024x1024x1024 o = 1 073741824 o |

La norme est de dire **kibi** octet et non **kilo** octet (1 mébi octet, 1 gibi octet ...) mais cette norme n'est pas respectée, on ne va donc pas en tenir compte !

**Conversions entre bases en python**

* Testez la calculatrice Windows en mode numérique.
* Instructions à tester à la console (attention : "42" est différent de 42) :  
  >>>bin(42)  
  >>>hex(42)  
  >>>int(0x2A)  
  >>>hex(0b101010)  
  >>>int("101010",2)  
  >>>int("42",10)  
  >>>int("2A",16)
* Réalisez un programme :

Qui demande à l'utilisateur de choisir la base d'écriture du nombre à convertir : 2, 10 ou 16.  
Qui demandera le nombre à convertir dans la base choisie.  
Qui donnera la conversion du nombre à convertir en base 2, 10 et 16.  
Exemple : on choisit la base 2, le nombre à convertir est 1010, le résultat est :

Conversion en binaire    : 0b101010  
 Conversion en décimal    : 42  
 Conversion en hexadécimal: 0x2a

base=int(input("Choisir la base d'écriture de votre nombre : 2, 10 ou 16 : "))  
nombre=input("Nombre à convertir : ")

...

**Addition des nombres binaires**

Vous savez faire une addition en base 10 : (6)10 + (3)10 = (9)10

L'ordinateur fera l'addition en base 2 car l'unité de calcul (ALU) des microprocesseurs doit réaliser des opérations sur des nombres binaires : (0110)10 + (0011)10 = (1001)10

Exemples d'addition binaire réalisée par l'unité de calcul (ALU) des microprocesseurs :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Retenues |  |  |  |  |  |  |  |
| Opérandes |  | 6 |  | 0 | 1 | 1 | 0 |
| + | 3 | + | 0 | 0 | 1 | 1 |
| Résultat |  | 9 |  | 1 | 0 | 0 | 1 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Retenues |  |  | 1 |  |  |  |  |  | 1 | 1 | 1 | 1 |  |
| Opérandes |  | 1 | 0 | 3 |  | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| + |  |  | 9 | + | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| Résultat |  | 1 | 1 | 2 |  | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

* (0010)2 + (0110)2 = 1000
* (0110)2 + (0110)2 = 1100
* (1011)2 + (0110)2 = 0001 0001
* (1111)2 + (0001)2 = 0001 0000
* (A7)16 + (02)16 = A9
* (A7)16 + (05)16 = AC
* (A7)16 + (26)16 = CD

**Jusque-là nous avons vu comment coder en binaire (binaire naturel) un entier positif.**

**Codage (complément à 2) d'un nombre entier relatif (entier signé)**

Pour coder un nombre négatif en **binaire signé** sur un octet, on affecte le signe "-" au MSB (bit de poids fort) :

* Nombres positifs : MSB = 0 (sur 8 bits : 0 à +127)
* Nombres négatifs : MSB = 1 (sur 8 bits : -1 à -128)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Codage binaire signé :** | | | | | | | | **Décimal :** | |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0+64+32+16+8+4+2+1 | **= 127** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0+0+0+0+0+0+2+0 | **= 2** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0+0+0+0+0+0+0+1 | **= 1** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0+0+0+0+0+0+0+0 | **= 0** |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | -128+64+32+16+8+4+2+1 | **= -1** |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | -128+64+32+16+8+4+2+0 | **= -2** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -128+0+0+0+0+0+0+1 | **= -127** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -128+0+0+0+0+0+0+0+0 | **= -128** |

Cette méthode appelée complément à deux permet d’éviter d’utiliser deux représentations différentes pour le nombre 0 (On évite d’avoir un +0 et un -0).

Les entiers (int) prennent 4 ou 8 octets (32 ou 64 bits), selon votre build Python

En 32 bits on peut coder les entiers de -2147483648 à +2147483647.

* (-118)10 → (N)2signé = 1000 1001
* (-37)10 → (N)2signé = 1101 1011
* Testez avec python : 0.1 + 0.2 == 0.3
* Testez avec python : 0.1 + 0.2
* Testez avec python : 1/3
* Testez avec python : 0.1 + 2.75 == 3.85
* Testez avec python : 2.75 + 5.1875 == 7.9375

**Tout cela s'explique !**

**Codage d'un nombres réel (nombre à virgule signé)**

**Représentation des nombres réels à virgule fixe**

En base 10, un nombre à virgule peut être décomposé à partir de puissance de 10 :

(435,375)10 = 4 x 102 + 3 x 101 + 5 x 100 + 3 x 10-1 + 7 x 10-2 + 5 x 10-3

Poids des bits :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2-1** = 1/2  = 0,5 | **2-2** = 1/4  = 0,25 | **2-3** = 1/8  = 0,125 | **2-4** = 1/16  = 0,0625 | **2-5** = 1/32  = 0,03125 |
| **2-6** = 1/64  = 0,015625 | **2-7** = 1/128  = 0,0078125 | **2-8** = 1/256  = 0,00390625 | **2-9** = 1/512  = 0,001953125 | **2-10** = 1/1024  = 0,0009765625 |

De même un nombre à virgule écrit en base 2 peut se décomposer à partir de puissance de 2 :

(101,011)2 = 1 x 22 + 0 x 21 + 1 x 20 + 0 x 2-1 + 1 x 2-2 + 1 x 2-3 = 5 + 0,25 + 0,125 = (5,375)10

Une machine n’a pas une mémoire infinie et ne peut donc pas gérer les nombres réels avec une partie décimale (partie après la virgule) infinie comme par exemple , 2, 1/3, cos(1)...

La machine peut seulement faire une approximation de la partie décimale dépendant du nombre de bits utilisés pour la représentation du nombre.

Codage choisi pour les calculs ci-dessous : 4 bits pour la partie entière + 4 bits pour la partie décimale.

* (0100,0010)2  (N)10
* (01100,1010)2  (N)10
* (2,75)10  (N)2
* (5,1875)10  (N)2
* (0,1)10  (N)2

**On aura le même problème avec 0,2 avec 0,25, avec plein de nombres, il faudra donc faire attention lors de la manipulation des nombres à virgule en programmation.**

**Représentation des nombres à virgule flottante**

Pour augmenter la précision du codage des nombres à virgule, la machine travaille avec ce qu’on appelle des nombres à virgule flottante (= les flottants).

Pour savoir où se trouve la virgule pour un **nombre à virgule fixe** on doit préciser le nombre de bits utilisés pour la représentation et placer autant de bits avant et après la virgule.

Les nombres décimaux sont représentés en machine sur 32 ou 64 bits.

Avec 32 bits : (5,375)10 = (0000000000000101,0110000000000000)2

* Pour les nombres inférieurs à 1 que constate-t-on sur la partie à gauche de la virgule ?

Rappel sur la notation scientifique :

* (23542,67)10 = 2,354267 x 104
* (0,000145879)10 = 1,458 79 x 10-4
* (-398879,62)10 = -3,9887962 x 105

**Représentation des flottants dans un ordinateur**

Norme IEEE-754 en 32 bits (il existe également une représentation en double précision sur 64 bits) :

|  |  |  |
| --- | --- | --- |
| 1 bit de signe | 8 bits pour l'exposant | 23 bits pour la mantisse |
| 0 | **1000 0001** | 000000000000000000 **01011** |

5,375  (101,011)2 1,**01011** x 2**10000001**

Détails : 1,**01011** x 2(2)  1,01011 x 2(2+127) 1,01011 x 2(129) 1,01011 x 2(10000001)2

Il faut rajouter 127 à l'exposant pour une conversion de décimal vers un nombre réel binaire.

**Conclusion**

Il existe plusieurs codages binaires pour représenter différentes données :

* Binaire naturel (nombre entier)
* Binaire signé (nombre entier avec un signe)
* Binaire flottant (nombre réel)
* Binaire ascii (caractères du clavier)
* ...