REPRÉSENTATION DES DONNEES : TYPES ET VALEURS DE BASE

I. Introduction

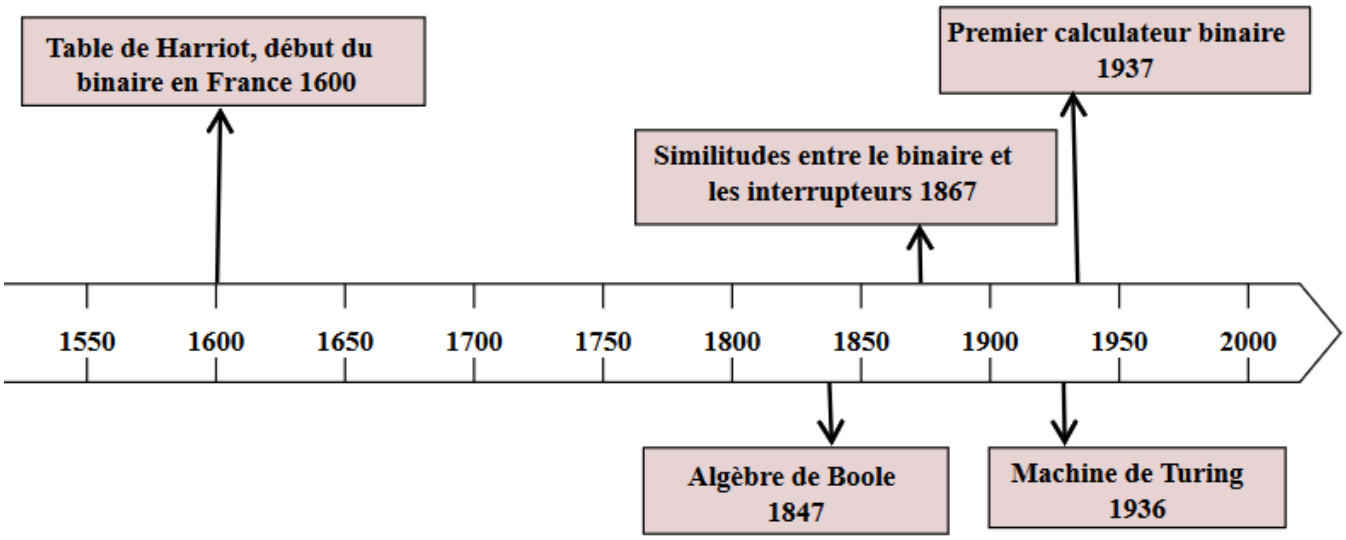
Vu de l'utilisateur, un ordinateur mémorise, transmet, transforme toute sorte d'informations (nombres, textes, images, sons, ...) mais en réalité, il ne manipule que des objets beaucoup plus simples (des 0 et des 1).

En effet, la mémoire des ordinateurs est constituée d'une multitude de petits circuits électroniques qui ne peuvent être chacun que dans deux états qui ont été appelés arbitrairement 0 et 1.

Ainsi toute information (nombre réel, entier, caractère, texte, image, …) est représentée par une suite de 0 et de 1, autrement dit, sous une forme binaire par un ensemble de nombres écrits en base 2 selon un format de stockage prédéfini.

Nous verrons que cette contrainte de format de stockage (ou « de représentation ») induit nécessairement une numérisation de l'information et des limites liées au compromis entre qualité de l'information et espace de stockage nécessaire.

Par contre, le codage binaire permet de ramener tout problème à de simples opérations logiques sur des 0 et des 1, et donc de s'appuyer sur l'énorme capacité de traitement électronique.



II. Bases décimale, binaire et hexadécimale

1. Système décimal

Nous utilisons le système décimal (base 10) dans nos activités quotidiennes. Ce système est basé sur dix chiffres, de 0 à 9. Pour compter, nous regroupons les objets par paquets de 10 appelés dizaines. Les dizaines sont ensuite regroupés par paquet de 10 appelés centaines et ainsi de suite...

Exemple : le nombre 156 de ce système s'écrit $N = 156_{10}$.

L'indice 10 indique la base dans laquelle le nombre est écrit. Le chiffre 6 représente le nombre d'unités, le chiffre 5 le nombre de dizaine et le chiffre 1 le nombre de centaines. Ce nombre N peut ainsi être écrit sous la forme d'un polynôme :

$N = 156_{10} = 100 + 50 + 6 = 1 \times 10^2 + \ldots . \times 10^{\cdots} + \ldots\ldots\ldots\ldots$

2. Système binaire (base 2)

En informatique, on utilise très fréquemment le système binaire (base 2) qui utilise seulement deux chiffres : 0 et 1. Ainsi les objets à compter sont regroupés par paquet de 2.

Exemple : $N= 1010_2$. Ce nombre N peut être écrit sous la forme d'un polynôme :

$N = 1010_2 =$ .....

Le bit le plus à droite est appelé bit de poids faible et celui le plus à gauche est appelé bit de poids fort.

Avec un système à 3 bits, on peut former $2^3 = 8$ nombres différents allant de 0 à $2^3 - 1 = 7$.
Avec un système à n bits, on peut former …… nombres différents allant de 0 à .....

Les ordinateurs utilisent 32 ou 64 bits pour coder les nombres entiers.

entiers	0 à 1	0 à 3	0 à 7	0 à 15	0 à 255	0 à 1023	0 à 4.10^9	0 à 2.10^{20}
codés sur	1 bit	2 bits	3 bits	4 bits	8 bits	10 bits	32 bits	64 bits

3. Système hexadécimal (base 16)

On utilise aussi très souvent le système hexadécimal (base 16) du fait de sa simplicité d'utilisation et de représentation pour les mots machines.
En effet les nombres écrit en base 16 sont plus compacts que les nombres écrits en base 2. De plus, on passe très facilement de l'écriture binaire à l'écriture hexadécimale.
Ce système dit de base 16 utilise ..... chiffres :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (10), B (11), C (12), D (13), E (14) et F(15).

Exemple : $N = BD6_{16}$. Ce nombre N peut être sous la forme d'un polynôme :

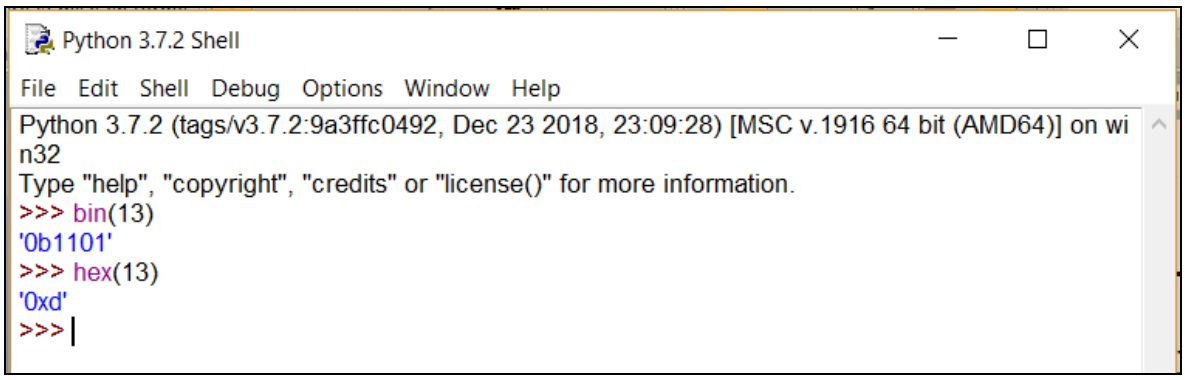
$N= BD6_{16} =$.....

4. Correspondance entre nombres de différentes bases

Compléter le tableau suivant :

Décimal	Binaire	hexadécimal		Décimal	Binaire	hexadécimal
0				8		
1				9		
2				10		
3				11		
4				12		
5				13		
6				14		
7				15		

Vérifier les résultats en utilisant la console « python » :



III. Quelques définitions et conventions : bit, octet, mot

- **Bit :** Le bit qui signifie « binary digit », est une unité élémentaire d'information ne pouvant prendre que deux valeurs distinctes : *0 ou 1.* Il s'agit de la plus petite unité d'information manipulable par une machine numérique.
- **Octet :** Les bits sont regroupés par paquets adjacents pour représenter de l'information.
  L'octet (en anglais byte est une unité d'information composée de 8 bits. Il permet par exemple de stocker un caractère comme une lettre ou un chiffre.
- **Mot binaire :** En informatique, l'unité de traitement de l'information est le mot binaire.
  Un mot (machine) est la quantité de bits standards manipulés par un microprocesseur (CPU). La taille du mot peut s'exprimer en bits ou en octets.
  Une unité d'information composée de 16 bits est généralement appelée mot (en anglais word).

IV. Passage de la représentation d'une base à une autre

1. Conversion d'un entier en base b

Méthode par division (euclidienne) : On réalise une suite de division entière par b jusqu'à obtenir un quotient égal à 0.

- **Conversion décimal - binaire**

Exemple : convertir 13 en base 2

1ère méthode

2ème méthode

Quotient	Reste
13	1
6	0
3	1
1	1

Réponse : $13_{10} = 1101_2$

Exercice 1 : Convertir N = 86_{10} en base 2

.....

.....

- **Conversion décimal - hexadécimal**

12 = C_{16}
10 = A_{16}

Réponse : $172_{10} = AC_{16}$

Exercice 2 : Convertir N = 89_{10} en base 16

.....

.....

- **Programmation en python** : proposer un pseudo-code puis le programmer
  Exercice 3 : Ecrire une fonction qui convertit un nombre entier en base 2, elle prend en entrée un nombre entier et renvoie un nombre binaire sous forme d'une chaine de caractères.
  Exercice 4 : Ecrire une fonction qui convertit un nombre entier en base 16, elle prend en entrée un nombre entier et renvoie un nombre hexadécimal

2. Conversion d'un nombre en base b en base 10

Pour passer d'un nombre en base b à un nombre en base 10, on utilise l'écriture polynomiale :

$$N = a_n b^n + a_{n-1} b^{n-1} + \ldots + a_1 b^1 + a_0 b^0$$

- **Conversion binaire - décimal**
  Il suffit d'écrire le nombre sous la forme d'une somme de puissance de 16.
  Exemple : converti le nombre $2B_{16}$ en décimal
  $10101_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 21$

  Exercice 5 : Convertir N = 1011010_2 en base 10
  .....
  .....
- **Conversion hexadécimal - décimal**
  Il suffit d'écrire le nombre sous la forme d'une somme de puissance de 16.
  Exemple : converti le nombre $2B_{16}$ en décimal
  $2B_{16} = 2 \times 16^1 + 11 \times 16^0 = 32$

  Exercice 6 : Convertir N = $B4_{16}$ en base 10
  .....
  .....
- **Programmation en python** : proposer un pseudo-code puis le programmer
- **Exercice 7 :** Ecrire une fonction qui convertit un nombre binaire en base 10, elle prend en entrée un nombre binaire sous forme de chaîne de caractères et renvoie un nombre entier.
- **Exercice 8 :** Ecrire une fonction qui convertit un nombre hexadécimal en base 10, elle prend en entrée un nombre en base 16 sous forme de chaine de caractères et renvoie un nombre entier.

3. Conversions hexadécimal/binaire ou binaire/hexadédimal

- **Conversion hexadécimal – binaire**
  Chaque symbole du nombre écrit dans le système hexadécimal est remplacé par son équivalent écrit dans le système binaire.
  Exemple : converti le nombre $2B_{16}$ en binaire
  2 B_{16} = 0010 1011
  Exercice 9 : Convertir N = CB_{16} en base 2
  .....
  .....
- **Conversion binaire – hexadécimal**
  C'est l'inverse de la précédente. Il faut donc regrouper les 1 et les 0 du nombre par 4 en commençant par la droite, puis chaque groupe est remplacé par le symbole hexadécimal correspondant.
  Exercice 10 : Convertir N = $1\ 1010\ 0010\ 1011_2$
  .....
  .....

V. Opérations binaires

Les principes de l'arithmétique binaire sont exploités par les microprocesseurs pour réaliser les opérations d'addition, soustraction, multiplication, division, comparaison.

1. Addition binaire

L'addition de deux nombres de 1 bit nécessite déjà un circuit à 3 entrées et 2 sorties pour gérer correctement la propagation de la retenue.

Règles : 0 + 0 = 0 1 + 0 = 1 0 +1 = 1 1+ 1 = 10 (0 et on retient 1 à gauche)

Exercice 11 : Effectuer ces additions en binaire, puis vérifier en décimal.

		1	**0**	**1**	**1**	**1**	**0**	**0**										**1**	**0**	**1**	**1**	**1**	**0**	**1**	0			
	+			**1**	**0**	**1**	**1**	**0**									**+**			**1**	**1**	**0**	**1**	**1**	**0**				

2. Soustraction binaire

Règles : 0 – 0 = 0 1 – 1 = 0 1 – 0 = 1 (1)0 – 1 = 1 (1 et on retient 1 à gauche)

Exercice 12 : Effectuer ces soustractions en binaire, puis vérifier en décimal.

					1	**0**	**1**	**1**	**0**	**0**										**1**	**0**	**1**	**0**	**0**	**1**	**1**			
				-		**1**	**1**	**0**	**0**	**1**									**-**		**1**	**1**	**0**	**0**	**1**	**1**			

3. Multiplication binaire

Exercice 13 : Effectuer ces multiplications en binaire, puis vérifier en décimal.

					1	**1**	**0**	**0**	**1**	**1**	**0**	**1**										**1**	**1**	**1**	**0**	**1**					
				*****						**1**	**1**	**1**									*****		**1**	**1**	**0**	**1**					

- **Programmation en python** :

  Exercice 14 : Addition – binaire

  Ecrire une fonction qui additionne deux nombre binaires, la fonction prend en paramètre deux nombre binaire sous forme de chaîne de caractères et retourne le résultat sous forme de chaîne de caractères.

VI. Codage des entiers

1. Codage des entiers positifs

Un entier naturel est un nombre entier positif ou nul. Le nombre de bits nécessaire pour coder ce nombre dépend du nombre de chiffres qu'il contient. Ainsi, pour coder des nombres entiers naturels compris entre 0 et 255, il nous suffira de 8 bits (un octet) car 2^8 =256. D'une manière générale un codage sur n bits pourra permettre de représenter des nombres entiers naturels compris entre 0 et 2^n –1.

Exemple : $5 = 00000101_2$; $23 = 00010111_2$

2. Nombres relatifs signés : entiers positifs ou négatifs

Un entier relatif est un entier pouvant être négatif, positif ou nul. Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif, et il faut de plus que les règles d'addition soient conservées.

- Méthode 1 : Complément à 1

Un entier relatif positif ou nul sera représenté en binaire (base 2) comme un entier naturel, à la seule différence que le bit de poids fort (le bit situé à l'extrême gauche) représente le signe. Il faut donc s'assurer pour un entier positif ou nul qu'il est à zéro (0 correspond à un signe positif, 1 à un signe négatif). Ainsi, si on code un entier relatif sur 4 bits, le nombre le plus grand sera 0111 (c'est-à-dire 7 en base décimale).

Exemple :

– 7 = 1111 et + 7 = 0111.

Inconvénient : Les règles d'addition ne sont pas conservées car 0111 + 1111 ≠ 0 en base 2**.** Il existe deux zéros, l'un positif et l'autre négatif (0000 et 1111).

- Méthode 2 : le complément à 2 (celle qui est utilisée) :

Pour éviter ces inconvénients, un entier relatif négatif sera représenté grâce au codage en complément à deux.

Principe du complément à 2

- Écrire la valeur absolue du nombre en base 2. Le bit de poids fort doit être égal à 0.
- Inverser les bits : les 0 deviennent des 1 et vice versa (Complément à 1).
- On ajoute 1 au résultat en ignorant les dépassements au niveau du bit le plus à gauche (on incrémente le complément à 1).

Exemple : Coder le nombre - 7 en utilisant le complément à deux

+ 7	=	0111	Vérification 7 - 7 = 0	Si une retenue est
Inversion des bits		1000	0111	engendrée au niveau du bit
Ajout de 1		0001	1001	le plus à gauche, elle est ignorée.
- 7	=	1001	0000	

On peut vérifier quelques avantages de cette méthode :

- le nombre 0 est codé d'une seule manière ;
- si on additionne +7 et -7, on obtient bien 0 (car on garde toujours le même nombre de bits pour coder le résultat, en ignorant les dépassements au niveau du bit le plus à gauche).

Coder le nombre - 11 en utilisant le complément à deux

+ 11	=	**01011**	Vérification	
Inversion des bits				
Ajout de 1				
- 11	=			