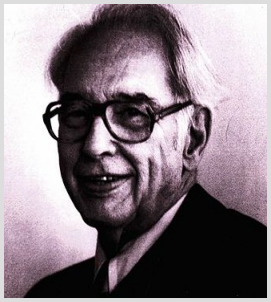
A. Le principe de la numérisation de l'information :

Ce qu'il faut savoir :



John Atanasoff

• A la fin des années 1930, ***John Atanasoff*** (physicien et mathématicien américain d'origine bulgare) a mis au point avec un de ses étudiants, ***Clifford Berry***, l'ABC (Atanasoff Berry Computer), un ordinateur de 300 kg, capable d'effectuer une opération toutes les 15 secondes. L'ABC est considéré comme ***le premier ordinateur***. Pour créer cet appareil, Atanasoff a imaginé un système électronique composé d'interrupteurs ouverts ou fermés qui permettent d'obtenir des 0 et des 1 et de réaliser des opérations avec ces données : c'est ***le système binaire***.

• Pour représenter l'information dans un ordinateur, ***seuls deux symboles sont utiles***. En effet, un ordinateur fonctionne avec des circuits électroniques et ne peut distinguer que deux état :
- 1er état : le circuit électrique est ***ouvert***, ce qui correspond au ***chiffre binaire 0***.
- 2è état : le circuit électrique est ***fermé***, ce qui correspond au ***chiffre binaire 1***.

• En informatique, ***un bit*** (contraction de ***B***inary dig***IT***), est un chiffre binaire (0 ou 1). C'est la base du système binaire. ***Un octet*** est un paquet de 8 bits, par exemple (11010110).

Remarque : Il ne faut pas confondre ***bit*** et ***byte***. En effet, en Anglais byte signifie un octet, soit 8 bits !

Activité 1 : Nous avons vu dans la séquence précédente que nous pouvons représenter tous les nombres entiers naturels en binaire.
Par exemple, avec 1 bit, nous pouvons représenter deux nombres entiers naturels : $(0)_2=0$ et $(1)_2=1$.

1. Écris les nombres entiers naturels que tu peux représenter en binaire avec 2 bits. Combien y-en-a-t-il ?
Avec 2 bits on a $(00)_2$ ***= 0 ,*** $(01)_2$ ***= 1 ,*** $(10)_2$ ***= 2 et*** $(11)_2$ ***= 3. Donc les*** $4=2^2$ ***premiers entiers naturels.***

2. Écris les nombres entiers naturels que tu peux représenter en binaire avec 3 bits. Combien y-en-a-t-il ?
Avec 3 bits on a $(000)_2$ ***= 0 ,*** $(001)_2$ ***= 1 ,*** $(011)_2$ ***= 3 ,*** $(100)_2$ ***= 4 ,*** $(101)_2$ ***= 5 ,*** $(110)_2$ ***= 6 et*** $(111)_2$ ***= 7.***
Donc les $8=2^3$ ***premiers entiers naturels.***

3. Quels nombres entiers naturels peut-on représenter en binaire avec 6 bits ? Et avec 1 octet (8 bits) ?
Avec 6 ou 8 bits on peut coder les $2^6=64$ ***ou*** $2^8=256$ ***premiers entiers naturels.***

Ce qu'il faut savoir :

• La numérisation de toutes les informations (sons, images, textes, nombres), ***repose sur le codage des nombres entiers naturels en binaire***.
Avec n bits, on peut coder 2^n ***nombres entiers naturels : tous les entiers naturels de 0 à*** 2^n-1 .

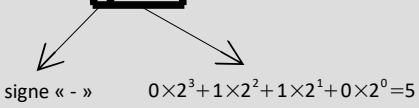
• Pour les nombres entiers et les textes qui sont ***des informations discrètes*** (il y a un nombre fini de valeurs différentes de lettres et un nombre fini d'entiers si on se fixe des bornes), il suffit d'associer chaque information à un nombre entier naturel codé en binaire.

• Pour les nombres réels, les sons et les images qui sont ***des informations continues*** (il y a un nombre infini de valeurs différentes), il faut les ***discrétiser***. C'est à dire que nous allons approximer ce nombre infini d'informations par un nombre fini d'informations que nous allons une fois encore associer à des nombres entiers naturels codés en binaire.

B. Premier codage des entiers relatifs, le binaire signé :

Ce qu'il faut savoir : Dans ***le binaire signé***, ***le bit de poids fort*** (=le premier bit le plus à gauche) est utilisé pour codé le signe (0 pour « + » et 1 pour « - ») et les bits suivants sont utilisés pour coder la valeur absolue du nombre entier relatif.

Exemple : Avec 4 bits, on a $(10110)_2$ code le nombre entier relatif -5.



Activité 2 : Dans cette activité, nous allons travailler sur 6 bits.

1. Quels sont les entiers relatifs codés par les mots binaires ci-dessous ?

a. $(111111)_2$. ***b.*** $(011111)_2$. ***c.*** $(101010)_2$. ***d.*** $(010101)_2$.

-31 ***31*** ***-9*** ***21***

2. a. Quels sont les entiers relatifs codés par $(100000)_2$ et par $(000000)_2$? ***Les deux codes représentent 0.***

b. Que peut-on conclure ? ***C'est problématique.***

3. a. Code, en binaire signé sur 6 bits, les entiers relatifs 11 et -12. $(\mathbf{001011})_2$ ***et*** $(\mathbf{101100})_2$ **.**

b. Pose et effectue la somme (en faisant attention aux retenues) des codes en binaire signé de 11 et -12.

$$\begin{array}{r} 0\,0\,1\,0\,1\,1 \\ +\ 1\,0\,1\,1\,0\,0 \\ \hline 1\,1\,0\,1\,1\,1 \end{array}$$

$(\mathbf{001011})_2$ **+** $(\mathbf{101100})_2$ **=** $(\mathbf{110111})_2$ **.**

c. Quel entier relatif est codé par le résultat du ***b.*** ? Que peux-tu en déduire ?

$(\mathbf{110111})_2$ ***représente le nombre -23. Comme*** $\mathbf{11}+(-\mathbf{12})=-\mathbf{1}$ ***et pas -23, c'est problématique.***

C. Second codage des entiers relatifs, le complément à 2 :

Ce qu'il faut savoir : Le binaire signé comportant des inconvénients mis en évidence dans l'activité 2, on préférera utiliser ***la méthode de codage du complément à 2***.

Avec cette méthode, si on code un entier relatif ***sur n bits***, on pourra coder $\mathbf{2}^n$ ***nombres entiers relatifs différents***, à savoir tous les nombres entiers relatifs ***compris entre*** $-\mathbf{2}^{n-1}$ ***et*** $\mathbf{2}^{n-1}-\mathbf{1}$.

Si x est un entier relatif compris entre -2^{n-1} et $2^{n-1}-1$:

• Si *x est positif* ($0\leqslant x\leqslant 2^{n-1}-1$) : On code x , qui est un entier naturel, classiquement.

• Si *x est négatif* ($-2^{n-1}\leqslant x\leqslant -1$) : On code x par le code de $x+2^n$ qui est lui aussi un entier naturel.

Activité 3 :

1. Quels sont les nombres entiers relatifs que l'on peut coder avec la méthode du complément à 2 sur 4 bits, sur 1 octet, 4 octets ou 8 octets ?

Sur 1 octet = 8 bits, on peut coder $2^8=256$ ***nombres : les nombres de*** $-2^7=-128$ ***à*** $2^7-1=127$ **.**
Sur 4 octets = 32 bits, on peut coder $2^{32}=$ ***4 294 967 296 nombres :***
les nombres de $-2^{31}=$ ***-2 147 483 648 à*** $2^{31}-1=$ ***2 147 483 647.***
Sur 8 octets = 64 bits, on peut coder $2^{64}=$ ***18 446 744 073 709 551 616 nombres :***
les nombres de $-2^{63}=$ ***-9 223 372 036 854 775 808 à*** $2^{31}-1=$ ***9 223 372 036 854 775 807.***

2. a. Détermine le code avec la méthode du complément à 2, sur 1 octet, du nombre 4. $(\mathbf{00000100})_2$

b. Dans le code obtenu, remplace les 0 par des 1 et les 1 par des 0, puis ajoute $(1)_2=(00000001)_2$ au résultat obtenu (en prenant bien garde aux retenues en base 2).
Dans $(\mathbf{00000100})_2$ **,** ***on remplace les 0 par des 1 et les 1 par des 0 :*** $(\mathbf{11111011})_2$ **.**
On ajoute $(\mathbf{00000001})_2$ ***et on obtient*** $(\mathbf{11111100})_2$ **.**

c. Détermine ensuite le code avec la méthode du complément à 2, sur 1 octet, de -4 . Que remarques-tu ?
Avec la méthode du complément à 2, sur 1 octet=8bits, -4 se code par le nombre entier naturel $-4+2^8=252$ **.** ***On a donc*** $(\mathbf{11111100})_0$ **.**
On remarque que le procédé de la question b. permet de retrouver le code de -4, l'opposé de 4.

3. Cette remarque précédente est-elle encore vérifiée pour 127 et -127 ? (réalise les calculs utiles à répondre)
On admettra que cette remarque est toujours vérifiée !
Avec la méthode du complément à 2, sur 1 octet=8bits, 127 se code $(\mathbf{01111111})_2$ **.**
Dans $(\mathbf{01111111})_2$ **,** ***on remplace les 0 par des 1 et les 1 par des 0 :*** $(\mathbf{0101})_2$ **.**
On ajoute $(\mathbf{00000001})_2$ ***et on obtient*** $(\mathbf{10000001})_2$ **.**
Avec la méthode du complément à 2, sur 1 octet=8bits, -127 se code par le nombre entier naturel $-127+2^8=129$ **.** ***On a donc*** $(\mathbf{10000001})_2$ **.**
Une fois encore que le procédé de la question b. permet de retrouver le code de -127, l'opposé de 127.

4. a. Sur 4 bits, quels sont les nombres entiers relatifs qui peuvent être codés avec la méthode du complément à 2 ? Code tous ces nombres entiers relatifs avec la méthode du complément à 2.
Sur 4 bits, on peut coder $2^4=16$ ***nombres : les nombres de*** $-2^3=-8$ ***à*** $2^3-1=7$ **.**
-8 : $(\mathbf{1000})_2$ **,** ***-7 :*** $(\mathbf{1001})_2$ **,** ***-6 :*** $(\mathbf{1010})_2$ **,** ***-5 :*** $(\mathbf{1011})_2$ **,** ***-4 :*** $(\mathbf{1100})_2$ **,** ***-3 :*** $(\mathbf{1101})_2$ **,** ***-2 :*** $(\mathbf{1110})_2$ **,**
-1 : $(\mathbf{1111})_2$ **,** ***0 :*** $(\mathbf{0000})_2$ **,** ***1 :*** $(\mathbf{0001})_2$ **,** ***2 :*** $(\mathbf{0010})_2$ **,** ***3 :*** $(\mathbf{0011})_2$ **,** ***4 :*** $(\mathbf{0100})_2$ **,** ***5 :*** $(\mathbf{0101})_2$ **,**
6 : $(\mathbf{0110})_2$ ***et 7 :*** $(\mathbf{0111})_2$ **.**

b. Que peux-tu remarquer pour le bit de poids fort des nombres négatifs ? Et des nombres positifs ?
On admettra que cette remarque est toujours vérifiée !
Je remarque que le bit de poids fort des nombres négatifs est toujours 1 et que celui des nombres positifs est toujours 0.

5. a. Code le nombre relatif -1 sur 6 bits, puis sur 1 octet.
Sur 6 bits, on code -1 par l'entier naturel $-1+2^6=63$ **,** ***soit*** $(\mathbf{111111})_2$ **.**
Sur 1 octet = 8 bits, on code -1 par l'entier naturel $-1+2^8=255$ **,** ***soit*** $(\mathbf{11111111})_2$ **.**

b. Que remarques-tu ? ***On admettra que cette remarque est toujours vérifiée !***
Je remarque que le code du nombre -1 n'est toujours constitué que de 1 quelque soit le nombre de bits utilisés.

Ce qu'il faut savoir :

• Si on connaît le code avec la méthode du complément à 2 d'un entier relatif, pour obtenir le code de son opposé, on réécrit le code du nombre de départ en remplaçant les 1 par des 0 et les 0 par des 1, puis on ajoute $(1)_2$ au code obtenu(en prenant garde aux retenues en base 2).

Exemple : $(110100)_2$ est le code de -12 sur 6 bits.

Je remplace les 1 par des 0 et les 0 par des 1 : $(001011)_2$.

Puis j'ajoute $(1)_2=(000001)_2$: $(001011)_2+(000001)_2=(001100)_2$.

$(001100)_2$ représente bien l'entier naturel 12 (compris entre -2^5 et 2^5-1).

• Avec la méthode du complément à 2, le code d'un entier relatif positif commence toujours par 0 et celui d'un entier relatif négatif commence toujours par 1.

• Avec la méthode du complément à 2, le code de l'entier -1 n'est constitué que de 1.

Exemple : le code de -1 est $(1111)_2$ sur 4 bits, ou $(111111)_2$ sur 6 bits ou encore $(11111111)_2$ sur un octet.

Activité 4 : Quels nombres entiers relatifs sont représentés, par la méthode du complément à 2 par

1. $(00000000)_2$ ***0***

2. $(10000000)_2$ $x+2^8=2^7$ $x=2^7-2^8=-128$

3. $(01111111)_2$ $2^6+\ldots+2^0$ $=127$

4. $(10000001)_2$. $x+2^8=2^7+2^0$ $x=2^7+2^0-2^8=-127$

aide : Commence par déterminer si cela représente un nombre positif ou négatif.
Puis convertis ces codes binaires en des entiers naturels. Pour finir conclus.

Activité 5 :

1. Considérons le programme ci-dessous sur Python.

```
from time import time
t=time()
for i in range(5000) :
    a=2**i
    print(a)
print(time()-t)
```

a. Teste et explique ce que fait ce programme.

Ce programme calcule et affiche 2^i pour i variant de 0 à 4 999 (5 000 calculs). Puis il calcule et affiche le temps nécessaire à Python pour réaliser ces calculs.

b. Quel est le dernier résultat affiché ?

Le dernier résultat affiché est le temps mis pour que Python réalise les calculs (ici moins de 0,5 s).

2. Considérons le programme ci-dessous sur Python.

```
from time import time
t=time()
for i in range(5000) :
    a=2222**i
    print(a)
print(time()-t)
```

a. Teste (sois patient!) et explique ce que fait ce programme.

Ce programme calcule et affiche 2222^i pour i variant de 0 à 4 999 (5 000 calculs). Puis il calcule et affiche le temps nécessaire à Python pour réaliser ces calculs.

b. Quel est le dernier résultat affiché ?

Le dernier résultat affiché est le temps mis pour que Python réalise les calculs (ici près de 40 s soit 80 fois plus lent).

3. Comment peux-tu expliquer la différence de résultats pour les questions ***c.*** des questions ***1.*** et ***2.*** ?

Python a besoin de beaucoup plus de temps pour calculer avec des nombre entiers très grands.

Ce qu'il faut savoir :

• Avec certains langages, le nombre d'octets avec lequel on travaille pour représenter les entiers relatifs peut être précisé (***1, 2, 4 ou 8 octets***, c'est à dire ***8, 16, 32 ou 64 bits***).

• Avec Python, il est inutile de se soucier de ce nombre, car c'est le logiciel qui décide seul combien d'octets sont nécessaires pour coder les entiers relatifs. Avec Python, le type des nombres entiers se nomme ***int*** (pour integer = entier en Français).

• Cependant, si un entier dépasse une certaine taille, le logiciel Python le découpe en plusieurs parties pour traiter les opérations. Ceci a pour conséquence de ***ralentir de façon importante les calculs*** et de ***nécessiter davantage d'espace mémoire***.

D. Exercices d'application :

Exercice 1 : Donne, avec la méthode du complément à 2, code les entiers ci-dessous sur 1 octet.

a. 100. $(\mathbf{01100100})_2$ ***b.*** 75. $(\mathbf{01001011})_2$ ***c.*** -50. $(\mathbf{11001110})_2$ ***d***. -128. $(\mathbf{10000000})_2$ ***e.*** -1. $(\mathbf{11111111})_2$ ***f.*** 128. ***Impossible*** ***g.*** -89. $(\mathbf{10100111})_2$

Exercice 2 : Dans chaque cas, donne le code des entiers a et b sur 1 octet avec la méthode du complément à 2. Pose et effectue la somme des 2 codes obtenus, puis vérifie que c'est bien le code de l'entier $a+b$.

1. $a=35$ et $b=65$.
$(00100011)_2+(01000001)_2=(01100100)_2$.
Or $(01100100)_2$ *représente* $2^6+2^5+2^2=100$.
Et on a bien $35+65=100$.

2. $a=-12$ et $b=45$.
$(11110100)_2+(00101101)_2=(100100001)_2$.
Mais comme sur 1 octet on ne garde que 8 bits, le bit de poids fort n'est pas gardé et le résultat est $(00100001)_2$ *qui représente* $2^5+2^0=33$.
Et on a bien $-12+45=33$.

3. $a=-84$ et $b=29$.
$(10101100)_2+(00011101)_2=(11001001)_2$
Or $(11001001)_2$ *représente l'entier relatif x, avec* $x+2^8=2^7+2^6+2^3+2^0$, *soit* $x=-55$.
Et on a bien $-84+29=-55$.

Exercice 3 : Détermine les nombres entiers relatifs qui sont représentés, avec la méthode du complément à 2, par les codes suivants. ***a.*** $(11001011)_2$. ***b.*** $(11010100)_2$. ***c.*** $(10000010)_2$. ***d.*** $(10101010)_2$.

a. $x+2^8=2^7+2^6+2^3+2^1+2^0$, *donc x=-53.*

b. $x+2^8=2^7+2^6+2^4+2^2$, *donc x=-44.*

c. $x+2^8=2^7+2^1$, *donc* $x=-126$.

d. $x+2^8=2^7+2^5+2^3+2^1$, *donc* $x=-86$.

Exercice 4 : VRAI ou FAUX ? (Justifie tes réponses)

1. Les premiers ordinateurs sont apparus au début du XIXè siècle.
Le premier ordinateur (le ABC) est apparu à la fin des années 1930 (au début du XXè siècle). Donc c'est FAUX.

2. 1 000 s'écrit aussi $(ABC)_{16}$ en hexadécimal.
$(\mathbf{ABC})_{16}$ ***Représente le nombre*** $10\times16^2+11\times16^1+12\times16^0$ ***= 2 748. Donc c'est FAUX.***

3. $(1001)_2\times(111)_2=(111111)_2$.

```
  1001
X  111
------
  1001
 10010
+100100
------
111111
```

Ainsi (1001)$_2$ X (111)$_2$ = (111111)$_2$. Donc c'est VRAI.

4. L'entier relatif -2 est codé sur 5 bits $(10010)_2$ avec la méthode du complément à 2.
-2 est codé par l'entier naturel $-2+2^5=-2+32=30$ ***, ce qui donne*** $(11110)_2$ ***, donc c'est FAUX.***

5. Tout ce qui est numérisé dans un ordinateur est identique aux informations réelles.
Les informations discrètes (Nombres entiers et les lettres) qui sont numérisées sont identiques aux informations réelles. Cependant, pour les informations continue (sons, images, nombres réels), les infirmations numérisées sont des approximations des informations réelles. Donc c'est FAUX.

Exercice 5 : Écrire un programme sur Python qui demande à l'utilisateur de saisir un entier relatif r codable sur 1 octet avec la méthode du complément à 2, qui détermine le code et le renvoie.
Aide : Revoir l'exercice 6 de la séquence précédente (=Écriture d'un entier naturel en base b⩾2).

```
n=-130
#On donne à n une valeur non comprise entre -128 et 127 pour que la boucle
#while puisse démarrer

while n<-2**7 or n>2**7-1:    #-2**7=-128 et 2**7-1=127
    n = int(input("Entrez un nombre entier compris entre -128 et 127: "))
#Tant que l'entier saisi n'est pas compris entre -128 et 127 on recommence.

b = ""

if n<0:    #Si n est négatif, on va coder l'entier n+2**8.
    n=n+2**8


while n!= 0:    #On reprend le programme de l'exercice 6 de la séquence précédente.
    q = int(n/2)
    b = str(n-q*2)+b
    n = q

while len(b)<8:  #On ajoute des 0 sur les bits de gauche qui ne seraient pas utilisés.
    b="0"+b


print(b)
```