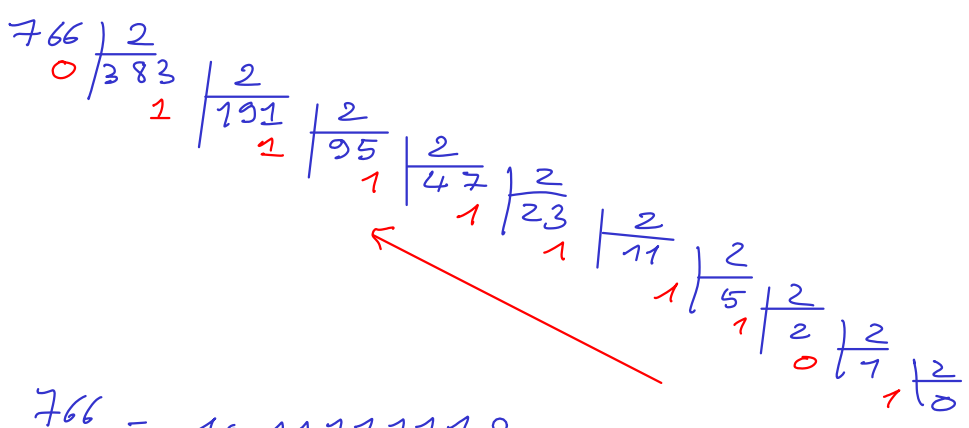


I1. Base deux

1. $\underline{11010001} = 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 1 \times 2^7$
 $= 1 + 16 + 64 + 128$
 $= 209$

2. 766 en base 2 ?

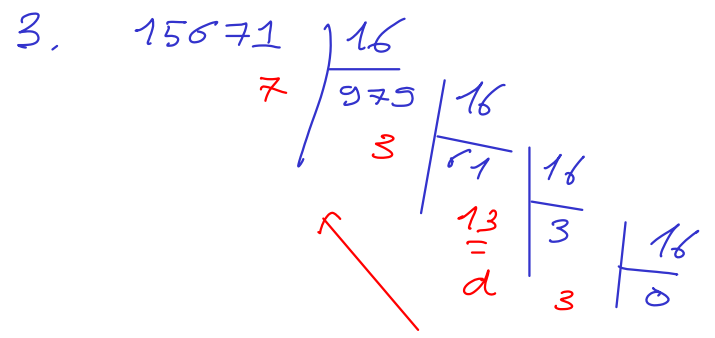


$766 = \underline{101111110}$

I2. Base hexadécimale

1. $a = 10, b = 11, c = 12, d = 13, e = 14, f = 15$

2. $(ef54a)_{16} = 10 \times 16^0 + 4 \times 16 + 5 \times 16^2 + 15 \times 16^3 + 14 \times 16^4$
 $= 980298$



$15671 = (3d37)_{16}$

I3. Complément à deux

1. $n = 8$ bits ;
- $2^{n-1} \leq N \leq 2^n - 1$
 - $2^7 \leq N \leq 2^8 - 1$
 - $128 \leq N \leq 255$
- ici

$$\begin{array}{r}
 2. \quad 101 \quad | \quad 2 \\
 \quad \quad 1 \quad | \quad 50 \\
 \quad \quad \quad 0 \quad | \quad 25 \\
 \quad \quad \quad \quad 1 \quad | \quad 12 \\
 \quad \quad \quad \quad \quad 0 \quad | \quad 6 \\
 \quad \quad \quad \quad \quad \quad 0 \quad | \quad 3 \\
 \quad \quad \quad \quad \quad \quad \quad 1 \quad | \quad 1 \\
 \quad \quad \quad \quad \quad \quad \quad \quad 1 \quad | \quad 0
 \end{array}$$

101 = 01100101 ⚠ 8 bits!

- 101?

$$\begin{array}{r}
 01100101 \\
 \quad \quad \downarrow \\
 \quad \quad 10011010 \\
 \text{puis} \quad + \quad \quad \quad \quad 1 \\
 \hline
 10011011 = -101
 \end{array}$$

Somme

$$\begin{array}{r}
 101 \\
 -101 \\
 \hline
 \end{array}
 \quad \leftarrow \quad
 \begin{array}{r}
 1 \quad 11 \quad 11 \quad 11 \\
 01100101 \\
 + 10011011 \\
 \hline
 \cancel{1}00000000
 \end{array}$$

bit excédentaire.

99 = 01100011
 57 = 00111001

$$\begin{array}{r}
 99 \\
 + 57 \\
 \hline
 156
 \end{array}
 \quad \text{valeur attendue.}$$

$$\begin{array}{r}
 1 \quad 1 \quad 11 \quad 11 \\
 01100011 \\
 + 00111001 \\
 \hline
 10011100
 \end{array}$$

↑ négatif : absurde!

Explication : 156 n'est pas représentable en complément à 2 sur 8 bits!
 Dépassement de capacité.

I4 - Conversion binaire → décimale.

```
def bin_to_dec(a):
```

```
    """
```

```
    N : list
```

```
    Bits de poids fort à gauche
```

```
    """
```

```
    x=0
```

```
    for i in range(len(a)):
```

```
        x=x+a[i]*2**(len(a)-i-1)
```

```
    return x
```

$$a = a_n a_{n-1} \dots a_1 a_0$$

$$n = \sum_{i=0}^n a_i \cdot 2^i$$

Il faut

I5 - Conversion décimale → binaire

```
def dec_to_bin(n):
```

```
    b=[]
```

```
    while n>0:
```

```
        b.append(n%2)
```

```
        n=n//2
```

```
    b.reverse()
```

```
    return b
```

```
print(dec_to_bin(10))
```

$$\begin{array}{r} 15 \ 2 \\ 1 \ 7 \\ 1 \ 3 \\ 1 \ 1 \\ 1 \ 0 \end{array}$$

↑ codage et arrêt.
le quotient vaut 0

← renvoie [1,0,1,0]

I6. Addition binaire

```
def add_bin(a,b,N):
```

```
    """
```

```
    Calcule c=a+b en representation binaire  
    a,b,c : list, liste de taille N. Representation binaire  
    en complement a deux. Bits de poids fort a l'indice 0.  
    N : taille de l'entier en bits.
```

```
    """
```

```
    c=init_list(N)#liste nulle de longueur N
```

```
    i=N-1#initialisation au bit de poids faible
```

```
    r=0#retenue
```

```
    while i>=0:
```

```
        s_bit=a[i]+b[i]+r
```

```
        if s_bit==0:
```

```
            c[i]=0
```

```
            r=0
```

```
        elif s_bit==1:
```

```
            c[i]=1
```

```
            r=0
```

```
        elif s_bit==2:
```

```
            c[i]=0
```

```
            r=1
```

```
        else:#a[i]+b[i]+r==2
```

```
            c[i]=1
```

```
            r=1
```

```
        i=i-1
```

```
    return c
```

```
print(add_bin([0,1,1,1],[0,0,0,1],4))#-1+1 en complement a deux
```

Renvoie [0,0,0,0] ; fonctionne !

```
def init_list(N):
```

```
    l=[]
```

```
    for i in range(N):
```

```
        l.append(0)
```

```
    return l
```