TD11: Numérisation des nombres entiers

Exercice 1 : Vrai-Faux ¶

1) Si l'écriture binaire d'un entier naturel se termine par n zéros alors cet entier est divisible par 2^n.

2) Pour écrire un nombre en base 8, on utilise des chiffres 1 à 8.

3) Avec n bits, les entiers représentables à l'aide de leur écriture binaire sont inférieurs à 2^n.

4) Soit m le mot binaire de n bits associés à un entier naturel A en base 10. Pour convertir A/2 il suffit de supprimer un bit à la fin de m.

5) L'entier 170 s'écrit AA en hexadécimal.

6) Si les entiers non signés sont codés sur des mots de 4 bits alors 1011+0101=0000

7) Avec des mots de 4 bits, on peut représenter tous les entiers relatifs de -8 à +8 compris

8) On code les entiers relatifs en complément A2 sur 5 bits. L'entier 10 est codé par 01010 et l'entier -10 par 11010.

9) Si des entiers non signés sont codés sur 8 bits alors il y a un nécessairement dépassement de capacité si on calcule la somme de 2 nombres dont le MSB est 1.

10) Si des entiers signés sont codés en complément A2 alors si la somme de deux nombres commençant par zéro est un nombre commençant par 1, alors il y a une erreur.

1) Vrai Si on écrit : $k = b_{n+1}2^{n+1} + b_n2^n + 0 * 2^{n-1} + \ldots + 0 * 2^0$

Alors $\frac{k}{2^n} = \frac{2^n*(b_{n+1}+b_n)}{2^n} = (b_{n+1} + b_n)$

2)Faux, Pour écrire un nombre en base b, on utilise les chiffre des 0 jusqu'à b-1 Donc en en octale, de 0 jusqu'à 7

3) Vrai strcitement inférieur car avec n bits, le plus grand des entiers vaut :
$2^0 + 2^1 + \ldots + 2^{n-1} = \frac{1-2^n}{1-2} = 2^n - 1$

4) Vrai, le principe de la convesrion binaire consiste à

- diviser successivement par 2
- le reste (0 ou 1) est le coefficient c_i tel que $c_i 2^i$
- on continue tant que le quotient est non nul

Donc passer de A à A/2 c'est supprimer une étape (la 1e) : on supprime un bit du mot binaire

5) Vrai Pour répondre à cette question, on peut remarquer que :
170=160+10=10*16^1+10*16^0 donc $(AA)_16$

il peut aussi faire la conversion en binaire puis regrouper les bits par 4 :

170/2 =85 +0

85/2 =42 +1

42/2 = 21 +0

21/2 =10 +1

10/2 =5 +0

5/2 =2 +1

2/2 =1 +0

1/2= 0+ 1

donc$(170)_2 = (10101010)_2 = (AA)_{16}$

Entrée [5]:
```
# test
print(hex(0b10101010))
print(hex(170))
```

```
0xaa
0xaa
```

6) si entier non signé alors sur 4 bits : 2^4 valeurs et 2^4-1 est la valeur max et un dépassement de capacité est possible après 1111 : ce qui est le cas ici

7) Faux : Avec des entiers non signés sur 4 bits on a:

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

avec des valeurs négatives (entiers signés), en complément A2 alors :

1000 -8

1001 -7

1010 -6

1011 -5

1100 -4

1101 -3

1110 -2

1111 -1

0000 0

0001 1

0010 2

0011 3

0100 4

0101 5

0110 6

0111 7

Donc entre -8 et 7 inclus

8)Faux, il ne suffit pas de changer le 1e digit !

Sur 5 bits,$(10)_{10}$ est codé sous la forme :01010

Donc en complément A2 : d'abord 10101 puis 10110

9)Vrai

1.... +

1....

Aboutit forcément à un dépassement de capacité

10) Vrai car la somme de deux nombres positifs (MSB 0) ne peut aboutir à un nombre négatif dont le MSB est 1

Exercice 2 : somme en binaire

Soient des nombres écrits en base binaire :

1) Calculer la somme de 100110 + 001101 en posant l'addition

2) Traduire ensuite le calcul en décimal

100110 => 2+4+32 =38

001101 => 1+4+8 =13

110011 => 1+2+16+32 =51

On peut aussi se lancer dans une soustraction

100110 => 2+4+32 =38

001101 => 1+4+8 =13

011001 => 1+8+16 =25

Exercice 3 :

Quelle affirmation est exacte :

1) Un nombre occupe 8 fois moins de place en mémoire s'il est représenté en octet plutôt que par des bits

2) Un nombre écrit en hexadécimal comporte 8 fois moins de caractères que s'il est écrit en binaire

3) Un nombre pair a une écriture binaire qui se termine par un 0

1) un octet reste 8 bits qui ne prennnent pas plus ou pas moins de place

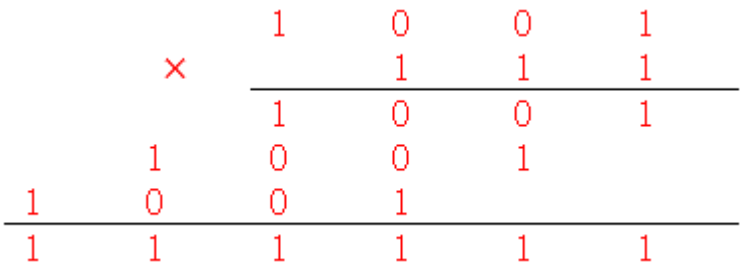
2) l'écriture hexa compacte les bits par 4 : donc de 8 caractères on passe à 2 soit 4 fois moins

3) Si paire alors la 1e divison par 2, qui fixe le LSB, est sans reste : Vrai

Exercice 4 : multiplication en binaire

Quelle la valeur en binaire de 1001×111 ?

Les différents étages aboutissent à trois additions:



On a 1001=9 et 111=7 donc 9*7=63 ce est bien 111111 On peut aussi faire la division : on obtient un quotient de 1 et un reste de 010

Soit 9=1*7+2

Exercice 5 : Complément A2

On utilise des mots de 5 bits pour coder des entiers relatifs en complément A2. Comment est codé -2 ?

Sur 5 bits : $(2)_{10} = (00010)_2$ Donc (11110) au final

Exercice 6 : Conversion

1) Ecrire une fonction qui prend en paramètre un entier naturel exprimé en base 10 et renvoie l'écriture en base deux de ce nombre. Le paramètre en entrée est de type int et la valeur en sortie est de type str.

2) Ecrire une fonction qui prend en paramètre un entier naturel exprimé en base deux et renvoie l'écriture en base 10 de ce nombre. Le paramètre en entrée est de type str (du MSB au LSB) et le paramètre en sortie de type int

3) Reprendre le programme précédent en remarquant qu'il est possible d'écrire :

$$(nbre)_{10} = b_0 + 2(b_1 + 2(b_2 + \cdots 2(b_{n-1} + 2b_n)..)$$

Entrée [3]:
```
def dec_2_bin(x):
    chaine=""
    while x!=0:
        x,r=x//2,x%2
        chaine = str(r)+chaine
    return chaine
print(dec_2_bin(10))
```

```
1010
```

Entrée [2]:
```
print(bin(10))
```

```
0b1010
```

Entrée [4]:
```
def bin_2_dec(chaine):
    x=0
    n=len(chaine)
    for i in range(n):
        x=x+int(chaine[i])*2**(n-1-i)
    return x
```

```
10
```

Entrée [10]:
```
def bin_2_dec2(chaine):
    x=int(chaine[0])
    for i in chaine[1:]:
        x=x*2+int(i)
    return x
print(bin_2_dec2("1010"))
```

```
10
```

Entrée [12]:
```
def bin_2_dec3(chaine):
    if len(chaine)==0:
        return 0
    else :
        x=int(chaine[-1])
        return x+2*bin_2_dec3(chaine[:-1])
print(bin_2_dec3("1010"))
```

```
10
```

Exercice 7 : Conversion sur 5 bits

Pour coder des entiers relatifs on utilise ici 5 bits.

1) Combien de nombres peut-on coder et lesquels ?

2) Comment est codé le nombre 6 ?

3) Comment est codé le nombre -7 ?

1) On peut donc coder 32 nombres entiers compris entre $-2^4 = -16$ à $2^4 - 1 = 15$

2)$(6)_{10} = (00110)_2$

3)$(7)_{10} = (00111)_2$ donc $(-7)_{10} = (11001)_2$

Exercice 8 : Conversion sur 32 bits

1) Combien de secondes positives peut-on écrire sur un mot de 32 bits signé ?

2) Est-il possible de coder le nombre de secondes qui se sont écoulées depuis le 1e janvier 1970 à 0h ?

1) Avec 32 bits signé, on peut atteindre 2^31-1

2) N≈52*365*24*3600<10^31-1 donc pas de dépassement. Il faut attendre typiquement 68 ans pour avoir un problème de codage

Entrée [12]:
```
print((2**31-1)/(365*(24+30)*3600))
```

```
30.26500432662495
```

Exercice 9 : Capacité numérique

1) Soit un nombre entier naturel x nécessitant n bits pour être décrit en binaire. Estimer, en fonction de n, le nombre de chiffres m nécessaire pour exprimer $(x)_{10}$.

2) Commenter le résultat précédent pour n=10.

3) Donner un ordre de grandeur, évaluée en Go, de la place occupée par 2^30 blocs de quatre octets chacun gravé sur un DVD.

4) La valeur de la capacité de mémoire vive d'un ordinateur est de 16Go, estimer le plus grand entier numérisable (ce nombre sera exprimé en base 10) ?

1) On a typiquement $x \approx 10^m \approx 2^n$ donc $m = nlog_{10}(2) \approx 0,3n$

2) m≈3, on retrouve que 10^3≈2^10 ce qui explique la confusion acceptée entre 1ko=1000o et 2^10 o=1024o=1kio≈1ko

3) le nombre de Go est : $\frac{2^{30}*4}{10^9} \approx \frac{(2^{10})^3*4}{10^9} \approx \frac{10^9*4}{10^9} \approx 4Go$ (Capacité de stockage typique d'un DVD)

4) $n = 16 * 10^9 * 8$ représente le nombre de bits utilisables soit n=38531839444≈10^11 et donc un nombre pouvant dépasser 2^(10^11)=10^m et avec m≈10^10. On pourra remarque qu'à partir de x=10^(10^6) l'ordinateur commence à mettre beaucoup de temps à écrire ce nombre.

Entrée [13]:
```
import numpy as np
print(np.log10(2))
```

```
0.3010299956639812
```

Entrée [ ]:
```

```