Devoir maison N°3

Le fichier **arithmetique.py** se trouve sur Moodle dans les fichiers associés à ce devoir maison. Il contient le squelette des différentes fonctions de cette feuille. Vous pouvez le l'utiliser pour compléter les fonctions à trous ou alors, vous pouvez créer une nouvelle feuille et essayer d'écrire intégralement les fonctions, en allant voir éventuellement le fichier si vous êtes bloqué.

L'objectif est de programmer la multiplication binaire sur les nombres binaires positifs.

Pour cela, vous devrez respecter les contraintes suivantes :

- ➔ Les nombres binaires sont représentés par des chaînes de caractères, comme "0111".
- ➔ Vous ne pouvez utiliser le symbole + uniquement pour concaténer des chaînes de caractères.
- ➔ Même si pour les tests, vous pouvez vous restreindre aux nombres représentés sur 4 bits, vos fonctions doivent marcher pour un nombre arbitraire de bits.
- ➔ On notera généralement `nb_bin` les nombres binaires.

Pour plusieurs fonctions, vous aurez besoin de parcourir et de construire les chaînes de droite à gauche. Vous pouvez vous inspirer des deux fonctions ci-dessous, où `nb_bin` est un nombre en binaire :

```
def copie_inverse1(nb_bin):
    ''' Retourne la chaine de caractere nb_bin renversée parcourue de droite à
    gauche
    nb_bin : chaine de caractères
    → Test :
    >>> copie_inverse1('10010')
    '01001'
    '''
    res = ""
    i = len(nb_bin)
    while i > 0:
          i = i - 1 # laisser au début de la boucle
          res = nb_bin[i] + res
    return res

def copie_inverse2(nb_bin):
    ''' Retourne la chaine de caractere nb_bin renversée parcourue de droite à
    gauche
    nb_bin : chaine de caractères
    → Test :
    >>> copie_inverse2('10010')
    '01001'
    '''
    res = ""
    for el in nb_bin :
          res = el + res
    return res
```

Lorsqu'il faut convertir un chiffre en un texte, ou un texte en chiffre, vous pouvez utiliser les fonctions suivantes :

```
>>>str(0)     # chiffre -> texte
'0'
>>>str(1)
'1'
>>> int("1")   # texte -> chiffre
1
>>> int("0")
0
```

Q1. Écrire une fonction **`conversion(nb, lg)`** qui prend deux entiers nb et lg et qui renvoie l'écriture binaire de nb sur lg bits. Si lg est inférieur à la longueur minimale nécessaire à l'expression binaire de nb, la fonction renvoie alors le nombre binaire complet sans troncature.

```
>>> conversion(13, 8)
'00001101'
>>> conversion(6, 4)
'0110'
>>> conversion(6, 2)
'110'
```

Q2. Ecrire une fonction **`booleen(bit)`** qui prend en argument un caractère bit et renvoie la valeur booléenne TRUE si bit vaut `'1'` et renvoie FALSE dans le cas contraire.

```
>>> booleen('1')
True
>>> booleen('0')
False
>>> booleen('n')
False
>>> booleen(1)
False
```

Q3. Ecrire une fonction **`caractere(bit_bool)`** qui prend en argument un booléen bit_bool et renvoie un caractère `'1'` ou `'0'` suivant l'état de bit_bool. Si bit_bool vaut `True` la fonction renvoie `'1'`, elle renvoie `'0'` dans le cas contraire.

L'addition s'il vous plaît

Pour faire l'addition en binaire, il faut parcourir les deux nombres de droite à gauche en faisant l'addition bit à bit, en tenant compte d'une éventuelle retenue. La table de vérité ci-contre donne l'état binaire du résultat b3 et de la retenue r2 en fonction des états des deux bits b1 et b2 additionnés à la retenue initiale r1.

On rappelle que les équations logiques du résultat b3 et de la retenue b2 sont :

b3 = b1 XOR b2 XOR r1
r2 = (b1 AND b2) OR (b1 AND r1) OR (b2 AND r1)

b1	b2	r1	b3	r2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1: Table de vérité de l'additionneur bit à bit

Sous python l'écriture des opérateurs logiques sont :

Opération	Ecriture Python	Exemple
OU	|	>>> TRUE | FALSE TRUE
ET	&	>>> TRUE & FALSE FALSE
XOR	^	>>> TRUE ^ FALSE TRUE

Q4. Écrire une fonction **`addition(nb1, nb2)`** qui renvoie la somme des nombres binaires nb1 et nb2. S'il reste une retenue à la fin, elle n'est pas ajoutée à gauche. Le résultat a le même nombre de bits que nb1 et nb2.

```
>>> addition("0101", "0011")
'1000'
>>> addition("1101", "0010")
'1111'
>>> addition("1001", "1000")
'0001'
```

Il était une fois. . .

Il existe plusieurs façons de faire les multiplications en binaire. On peut simplement "dépiler" un des deux facteurs :

$$A \times B = \underbrace{A + A + A + \cdots + A}_{\text{B fois}}$$

Nous allons plutôt poser les multiplications de nombres positifs, comme "à l'école". Prenons pour exemple les nombres positifs, dans l'exemple ci-contre.
Le résultat obtenu a 8 bits car lors d'une multiplication de deux nombres de k bits, on peut obtenir un résultat à 2k bits. On peut noter que la multiplication est relativement simple à poser puisqu'à chaque bit du nombre du bas, s'il est égal à 1, on additionne le nombre du haut décalé à gauche, sinon on additionne 0. Il n'y a donc aucune multiplication à faire, juste des additions.

```
   1 1 0 1
 × 1 0 1 1
 ---------
   1 1 0 1
 1 1 0 1
 0 0 0 0
 1 1 0 1
 ---------
 1 0 0 0 1 1 1 1
```

Figure 2: Exemple de multiplication binaire

Q5. Vérifier le résultat de la multiplication précédente en convertissant les nombres en base 10.

Q6. Poser en binaire les multiplications suivantes :

- 0011 × 0011
- 1011 × 1001
- 1111 × 1111

Q7. Écrire une fonction **`multiplication(nb1, nb2)`** qui renvoie le produit des nombres positifs nb1 par nb2.

```
>>> multiplication("1101", "1011")
'10001111'
>>> multiplication("0011", "0011")
'00001001'
>>> multiplication("1111", "1111")
'11100001'
```