Sujet 2

Victoria Vila

Système binaire (hex), opérations logiques (and, or, xor, not), négation , incrémenter, décrémenter, modulo

1.Système binaire (hex)

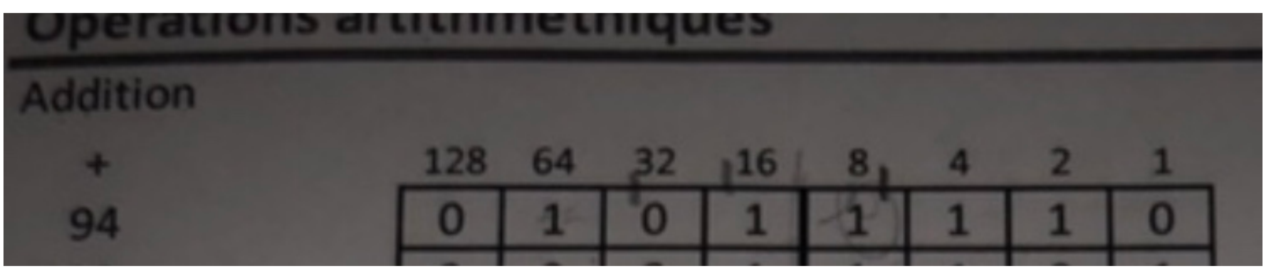
Le système binaire est le système de numération utilisant la base 2, que l'on nomme bit. Chaque symbole peut avoir une valeure de 0 ou 1 (soit 2^n combinaisons) . Le système binaire est utile pour représenter le fonctionnement de l'électronique numérique utilisée dans les ordinateurs. Il est utilisé pour les langages de programmations. Cette base binaire est justement pas compliquée à representer pour l'esprit humain.

Les deux chiffres 0 et 1 se traduisent par la tension du courant (soit 0 = tension nul). Il peuvent aussi représenter les valeures True ou False [wikipédia (https://fr.wikipedia.org/wiki/Syst%C3%A8me_binaire)](https://fr.wikipedia.org/wiki/Syst%C3%A8me_binaire)

1.1 Calcul binaire sur 8 bits (octets)

Le bit (01) est l'unité de mesure de l'informatique. Ici nous sommes sur 8bits soit un octet permettant de représenter 2^8 nombres (256 valeurs différentes).

Pour représenter 94 par exemple. Il faut tout d'abord poser la valeure 1 le plus proche de 94 mais inférieur (ici 64). Ensuite on additionne avec les chiffres inférieurs pour arriver à 94 (ici on pose la valeure 1 sous le bits correspondant à 16, 8, 4, et 2). On pose la valeure 0 dans les autres bits.


Operations arithmetiques

Addition

+	128	64	32	16	8	4	2	1
94	0	1	0	1	1	1	1	0

Pour représenter les résultat d'une division en système binaire, seulement la partie entière est décrite. Le modulo (`%`) représente le reste de la divion en système binaire. Donc en système binaire seul les nombres entier sont représenté .

1.1.2 opération binaire et logique (exemple fait sur 4 bits)

- right shift : 14>>1 : Retourne 14 avec les bits décalés à droite de 1 place. Exemple : 1110>>0001 = 0111
- left shift : 3<<2 : Retourne 3 avec les bits décalés à gauche de 1 place. Exemple : 0011<<0010 = 1100
- INV : Il suffit de mettre l'inverse de la valeure du bit. Exemple :2 : 0010, inverse de 2 : 1101
- AND : Le réslutat est toujour 0 sauf lorsque dans les deux bits la valeure est 1.Exemple : 1101 & 0110 = 0100
- Or : Dès qu'il y a la valeure 1, le résultat est 1. Exemple : 1110 OR 0011 = 1111
- Xor : Dès qu'il y a une valeure double à la suite, le résultat est 0. Exemple : 1110 XOR 0011 = 1101

1.2 Nombres négatifs :

Il n'est pas évident de les représenter vu que nous avons que la valeurs 1 et 0. Ce sont des valeures qui ne sont pas signées.

Le système binaire décrit des nombres dont leur longueur n'est pas modifiable, ils ont ce qu'on appelle des dimensions fixes. C'est comme les compteurs kilométriques de voiture. Si la voiture roule 1 km en marche arrière, le compteur soustrait 1 km et affiche 999.999 km. Ce code est la valeure –1 car on obtient 0 si on lui ajoute 1. Le principe est le même pour exprimer les chiffre binaires négatifs.

La solution est donc de remplacer tous les 0 par des 1 et tous les 1 par des 0. Ensuite il faut ajouter un bit devant notre chiffre en base binaire qui indique le signe. 0 pour dire que c'est postif et 1 pour négatifs.

Exemple : 1 = 0000 0001 ; -1 = 1111 1111 ; 2 = 0000 0010 ; -2 = 1111 1110

Les premiers chiffres ici indiquent le signe.

1.2 Hexadécimal

Par contre la notation binaire peut avoir trop de chiffre et devient difficile à lire et peut mener à des erreures de retranscription. On utilise donc la forme hexadécimal pour se représenter la valeur de notre chiffre à base binaire. Le système hexadécimal est un système de numération positionnel en base 16. Il utilise ainsi 16 symboles, en général les chiffres arabes pour les dix premiers chiffres et les lettres A à F pour les six suivants.[wikipédia (https://fr.wikipedia.org/wiki/Syst%C3%A8me_hexad%C3%A9cimal)](https://fr.wikipedia.org/wiki/Syst%C3%A8me_hexad%C3%A9cimal).

Ce tableau est un convertisseure binaire.

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

- Exemple pour déccrire le chiffre 94 en hex : 0101 1110.

Sur les 4 premiers bit on peut lire la valeure 5 donc en hex 5. Sur les 4 dernieres bits on lit la valeure 14 donc E en hex. 94 = 5E

2. Opérations logiques

Un opérateur est un signe qui effectue une opération sur une opérande. Une opérande peut être une variable, un chiffre ou une expression. Une expression est une suite d'opérandes et d'opérateur.

- Il existe les opréateurs avec lesquels nous sommes très familié tel que =, +, - ect...
- Il existe aussi des opérateurs de comparaison >, < ... Ils sont souvent utiles pour poser des conditions.

In [21]:
```
c = (1,2,4,5)
for i in c :
    if i < 3:
        print( str(i) + ' is smaller than 3')

    else :
        print (str(i) + ' is taller than 3')
```

```
1 is smaller than 3
2 is smaller than 3
4 is taller than 3
5 is taller than 3
```

Voci une liste de tout les opérateurs de comparaison :

"<" strictement inférieur, ">" strictement supérieur, "<=" inférieur ou égal ">=" supérieur ou égal, "==" égal, "!" = différent

- Il existent quatres opérateurs logiques: `and` , `or` , `^` (xor) et `not` . Leur valeure de retour est True ou False

2.1 AND

Il est utile pour vérifier si deux propositions sont justes l'opérateur `and` est utilisé. Il renvoie True lorsque c'est le cas.

En logique : La **conjonction** $P \land Q$ de deux propositions P et Q est la proposition qui est vraie uniqement quand les deux propositions P ou Q sont vraies simultanément. On dit que c'est la proposition P **et** Q.

In [89]:
```
(a, b) = (4, 9)
```

In [90]:
```
y = 6
(a<=y) and (y<=b)
```

Out[90]: `True`

2.2 OR

Il est utile pour vérifier si au moins une des deux propostions est justes. Il renvoie True si c'est le cas.

En logique : La **disjonction** $P \lor Q$ de deux propositions P et Q est la proposition qui est vraie dès qu'au moins une des deux propositions P ou Q est vraie. On dit que c'est la proposition P **ou** Q.

In [91]:
```
y = 6
(a>=y) or (y<=b)
```

Out[91]: `True`

2.3 XOR

C'est un OR exclusif, il doit y avoir seulement une proposition juste. Il renvoie True si c'est le cas.

In [92]:
```
y = 6
(a<=y) ^ (y<=b)
```

Out[92]: `False`

In [93]:
```
y = 6
(a>=y) ^ (y<=b)
```

Out[93]: True

2.4 NOT

La propostion `not` change la valeure de vérité de la propostion.

In [94]:
```
P = True
for P in (False, True):
    print(P, not P)
```

```
False True
True False
```

In [95]:
```
inverseParNot = True
print("La valeure d'origine:")
print(inverseParNot)
print("La valeure inversée:")
print(not inverseParNot)
```

```
La valeure d'origine:
True
La valeure inversée:
False
```

3. Modulo, incrémenter décrémenter

3.1 Modulo

Le modulo (`%`) représente le reste de la divions euclidienne. Il est souvent utile pour des cycles par exemples ou pour définir si un nombre est paire ou pas, ou encore il peut aider à créer pleins d'autres astuces.

Tester si un nombre est pair ou pas:

In [3]:
```
x = 45
if x%2 == 0:
    print('x est paire')
else:
    print('x est impaire')
```

```
x est impaire
```

Tester si un nombre est divisible par 5

In [97]:
```
x = 15

if x%5==0:
    print(x/5)
else :
    print('pas divisible par 5')
```

```
3.0
```

3.2 Incrémenter et décrémenter

Incrémentation est le fait d'ajouter 1 à un compteur, à une variable. Décrémenter est le fait de retirer 1 à un compteur. Ces opérations sont souvent utilisées dans la boucle while

L'instruction `i ++` (`i=i+1`) incrémente. L'instruction `i --` (`i=i-1`) décrémente.

In [98]:
```
c = 1
while c <= 5:
    print (c, "x", 10, "=", c*5)
    c = c + 1
```

```
1 x 10 = 5
2 x 10 = 10
3 x 10 = 15
4 x 10 = 20
5 x 10 = 25
```

L'opérateur modulo permet une incrémentation ou décrémentation cyclique.

In [99]:
```
x = 1
for i in range(4):
    x = (x+1)%4
    print(x, end=', ')
```

```
2, 3, 0, 1,
```

In [100]:
```
x = 3
for i in range(10):
    x = (x-1)%4
    print(x, end=', ')
```

```
2, 1, 0, 3, 2, 1, 0, 3, 2, 1,
```

4. Négation

```
Si une proposition est vraie, sa négation est fausse. Si une proposition est fausse, sa négation est vraie.

La **négation** d'une proposition $P$, notée $\neg P$ est la proposition obtenu en affirmant son contraire. On utilise donc la proposition `not`.
```

Exemple : La négation de "15 est divisible par x", est "15 n'est pas divisible par x".

In [101]:
```
a>b
```
Out[101]: False

In [124]:
```
not a>b
```
Out[124]: True

In [125]:
```
test = 0

for i in range(10):
    test += test
if not test > 10000000:
    print ("La condition est bien fausse, et donc grace au not elle es
else:
    print("la condition est vrai car le test est bien plus grand que 1

```
```
La condition est bien fausse, et donc grace au not elle est considéré com
me true, c'est pourquoi je suis imprimé et pas la phrase en dessous
```

La loi de Morgan

- La négation(`not`) d'une conjonction (`and`) est la disjonction(`or`) mais avec la négations des propositions A et B

In [126]:
```
a = 4
b = 9
y = 6

not((a<=y) and (y<=b))
```
Out[126]: False

In [127]:
```
not(a<=y) or not(y<=b)
```
Out[127]: False

Nous constatons bien que cela renvoie la même valeure.

- La négation(`not`) d'une disjonction (`or`) est la conjonction (`and`) mais avec la négations des propositions A et B

In [105]:
```
not ((a>=y) or (y<=b))
```
Out[105]: False

In [106]:
```
not (a<=y) and not(y<=b)
```

Out[106]: False

Nous constatons bien que cela renvoie la même valeure.