binaire, décimal, hexadécimal

Exo 1 : convertir à la main (*i.e.* préciser les étapes) les nombres 24, 162, 43 en base 2 et base 16.

Exo 2 : convertir à la main les nombres $\overline{1010\ 1110}^2$, $\overline{1010\ 1110\ 0011}^2$, $\overline{1111\ 0010\ 1010\ 1110}^2$ en base 10 et base 16.

Exo 3 : convertir à la main les nombres $\overline{14}^{16}$, $\overline{c8}^{16}$, $\overline{3ae}^{16}$ en base 2 et base 10.

Exo 4 : Donner la représentation en complément à 2 sur 8 bits des entiers suivants 46, 103, -12, -78.

Exo 5 : Convertir ces nombres octets signés en nombre décimal $\overline{1110\ 0111}^2$, $\overline{1010\ 1100}^2$, $\overline{0100\ 0110}^2$, $\overline{0101\ 1010}^2$.

Exo 6 : Faire, à la main, la somme et le produit des deux nombres $\overline{1010\ 0110}^2$ et $\overline{0011\ 1011}^2$.

Exo 7 : Compléter ce code qui permet à partir d'un binaire, avoir sa valeur décimale.

Écrire l'organigramme correspondant et proposer un jeu de tests.

```
print("Entrer un nombre ecrit en base 2")
NBinaire = ...
n = ...(NBinaire)
...
for i in ...:
    b = ...(NBinaire[...])
    ValDecimale = ...+...*2**(...)
print(...)
```

Exo 8 : Compléter ce code qui permet à partir d'un nombre décimal, obtenir son écriture binaire.

Écrire l'organigramme correspondant et proposer un jeu de tests.

```
print("Entrer un nombre ecrit en base 10")
ValDec = ...        # ValDec est un entier
NBin = ""           # NBin chaine vide
while (ValDec > ...):
   NBin = str(...)+...
   ValDec = ...
print("Le nombre en base 2 : {NBin}")
```

flottant et erreur

Exo 9 : Quel nombre représente le flottant simple précision 0′1000 1111′1011 0001 0110 0100 1000 000 ? (écrire les étapes de calculs)

Exo 10 : Quel nombre représente le flottant simple précision 1′0111 1100′1110 0000 0000 0000 0000 000 ? (écrire les étapes de calculs)

Exo 11 : Quel nombre représente le flottant simple précision 1′1001 0111′1000 0001 0000 1000 0000 101 ? (écrire les étapes de calculs)

Exo 12 : Quel nombre représente le flottant simple précision 0′0111 1100′1100 1110 0000 0000 0000 000 ? (écrire les étapes de calculs)

Exo 13 : Écrire le nombre 100 comme flottant simple précision (écrire les étapes de calculs).

Exo 14 : Écrire le nombre $-999{,}5$ comme flottant simple précision (écrire les étapes de calculs).

Exo 15 : Écrire le nombre $-7{,}21875$ comme flottant simple précision (écrire les étapes de calculs).

Exo 16 : Écrire le nombre 408,1875 comme flottant simple précision (écrire les étapes de calculs).

Exo 17 : calculer une majoration des erreurs faites lors d'une somme puis lors d'une multiplication pour $a = \sqrt{5} = 2{,}23606\ldots$ et $\tilde{a} = 2{,}2361$, $b = \sqrt{331} = 18{,}19341\ldots$ et $\tilde{b} = 18{,}1934$.

Exo 18 : calculer une majoration des erreurs faites lors d'une somme puis lors d'une multiplication pour $a = \sqrt{10} = 3{,}16228\ldots$ et $\tilde{a} = 3{,}1623$, $b = \sqrt{20} = 4{,}47213\ldots$ et $\tilde{b} = 4{,}4721$.

Exo 19 : calculer une majoration des erreurs faites lors d'une somme puis lors d'une multiplication pour $\tilde{a} = 24{,}04163$ (erreur de 7.10^{-5}) et $\tilde{b} = 10578{,}4563$ (erreur de 1.10^{-7})

Exo 20 : calculer une majoration des erreurs faites lors d'une somme puis lors d'une multiplication pour $\tilde{a} = 13{,}0012$ (erreur de 3.10^{-5}) et $\tilde{b} = 20150{,}0456$ (erreur de 2.10^{-5})

Exo 21 : Trouver le plus petit nombre de sorte qu'en `python x+1==1` soit vraie. On considèrera des flottants simple précision (32b) [⚠ généralement `python` utilise des flottants double précision].

Exo 22 : même exercice avec des flottants double précision.

Exo 23 : même exercice avec `x+100=100` et des flottants simple précision.

Exo 24 : même exercice avec des flottants double précision.

code ascii

Exo 25 : Qu'affiche `print("\x50\x72\x65mi\x65\x72\n\b\x65")` ? Expliquez.

Exo 26 : Qu'affiche `print("\x56\x65r\x73\x61i\x6c\x6c\t\bes")` ? Expliquez.

Exo 27 : écrire le mot "Ordi-01" avec son code Ascii hexadécimal.

Exo 28 : même exercice avec "32 eleves !".

Exo 29 : On considère le code suivant :

```
for e in "maths" :
    print( hex( ord(e) ), end='' )
```

Que renvoie ce code ?
Proposer une procédure `code_ascii(mot)` qui affiche le code Ascii d'une chaîne de caractères `mot` en hexadécimal.

Exo 30 : Proposer une fonction `minus(mot)` qui renvoie une chaîne de caractères correspondante à `mot` tout en minuscules (sans utiliser la fonction `lower()`)

Exo 31 : même excercice avec la fonction `majus(mot)` tout en majuscules (sans `upper()`).

architecture

Par la suite, on considèrera l'architecture vue en cours (i.e. avec un seul registre d'accumulation ACC). On déroulera le code assembleur dans un tableau

instruct°	***acc***	***mem***	***sortie***

Exo 32 : Vrai/Faux (si c'est faux, corriger) :

a) de nos jours, la vitesse d'un ordinateur est plutôt limitée par la cadence du CPU.

b) un bus 64bits permet de faire transiter vers l'ALU un flottant double précision d'un coup.

c) le CPU est composé de la mémoire et de l'unité algorithmique et logique.

d) le langage assembleur est le langage le plus proche du langage machine.

Exo 33 : Dérouler ce code assembleur, que fait-il ? Écrire le code `python` correspondant.

```
10  LDA #5
20  STA 500
30  MUL 500
40  STA 510
50  PRINT
60  END
```

```
10  LDA #3
20  ADD #-1
30  PRINT
40  BRZ 60
50  BRA 20
60  LDA #"BRAVO"
70  PRINT
80  END
```

Exo 34 : même exercice

```
10  LDA #3
20  ADD #-2
30  BRZ 100
40  ADD #-1
50  BRZ 150
60  PRINT
70  END
:
100 LDA #"ABC"
110 BRA 60
:
150 LDA #"XYZ"
160 BRA 60
```

Exo 35 : même exercice (l'entrée vaut 2)

```
10  LDA 500
20  PRINT
30  READ
40  STA 510
50  LDA #5
60  STA 520
70  MUL 510
80  PRINT
90  LDA 510
100 ADD #-1
110 BRZ 130
120 BRA 60
130 END
:
500 "Entier"
510 0
520 0
```

Exo 36 : Dérouler ce code assembleur et écrire le code `python` correspondant.

```
10  LDA #0
20  BRZ 50
30  LDA #"ECHEC"
40  PRINT
50  LDA #"SUCCES"
60  PRINT
70  END
```

Exo 37 : même exercice avec

```
10  LDA #3
20  ADD #-1
30  MUL #4
40  STA 500
50  DECAL #2
60  STA 510
70  END
```

Exo 38 : même exercice avec (déroulé avec les entrées 2, 1, 0)

```
10  LDA #"zero"
20  PRINT
30  READ
40  BRZ 60
50  BRA 30
60  STA 500
70  END
```

Exo 39 : Traduire en code assembleur ce code `python`(on considèrera que la mémoire commence à l'adresse 500) :

```
x = 3
y = 4
z = x + y
```

Exo 40 : même exercice avec

```
n = input("Bonjour")
if n==0 :
   x = 1
else :
   x = n**2
```

logique & circuit

Exo 41 : Dresser la table de vérité de l'expression $(a.b)+c$.

Exo 42 : Dresser la table de vérité de l'expression $(a+b)\oplus c$.

Exo 43 : Dresser la table de vérité de l'expression $(\bar{a}ora).b$ et préciser son circuit logique.

Exo 44 : Dresser la table de vérité de l'expression $(a.b)+\bar{c}$

Exo 45 : Dresser la table de vérité de l'expression $(a.\bar{b}.\bar{c})+(\bar{a}.b.\bar{c})+(\bar{a}.\bar{b}.c)$. Que fait cette expression booléenne ? La simplifier (un XOR sera présent) puis préciser son circuit logique.

Exo 46 : Montrer que $a.(b+\bar{a})=a.b$ et que $(a+\bar{b}).(\bar{a}+\bar{b})=\bar{b}$.

Exo 47 : Montrer que $(a+\bar{b}).(\bar{a}+b)=\bar{a}+a.b$.

Exo 48 : Définir la porte NOT qu'à partir de portes NAND.

Exo 49 : Définir la porte AND qu'à partir de portes NOR.

Exo 50 : Définir la porte OR qu'à partir de portes NAND.

Exo 51 : À l'aide des seuls opérateurs NOT, OR, AND et XOR, définir une expression booléenne contenant les variables a et b qui soit égale à 1 ssi $a=b$. Construire le circuit correspondant.

De même sans la porte XOR.

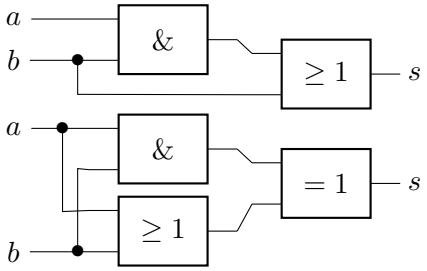
Exo 52 : On considère la table de vérité suivante :
Préciser l'expression logique de s puis le circuit logique correspondant.

a	***b***	***s***
0	0	1
0	1	1
1	0	0
1	1	1

Exo 53 : même exercice avec

a	***b***	***s***
0	0	1
0	1	0
1	0	0
1	1	0

Exo 54 : À partir du circuit logique, dresser la table de vérité et l'expression propositionnelle correspondante.



Exo 55 : Même exercice avec

Exo 56 : À partir de l'additionneur 1bit, proposer un additionneur 2 bits.

Exo 57 : À partir de l'additionneur 2bits, proposer un additionneur 4 bits, puis 8 bits.

Exo 58 : Proposer une fonction `python` `port_and(a,b)` qui code une porte and avec a et b des booléens ; *e.g.* `port_and(True, False)` renverra `False`.

Exo 59 : Proposer une fonction `python` `port_or(a,b)` qui code une porte or avec a et b des booléens.

Exo 60 : Proposer une fonction `python` `port_xor(a,b)` qui code une porte xor avec a et b des booléens.

Exo 61 : Proposer une fonction `python` `f(a,b)` qui renvoie l'expression $\bar{a}.b$ avec a et b des booléens.

Exo 62 : Proposer une fonction `python` `f(a,b)` qui renvoie l'expression $(\bar{a}+\bar{b}).(a+b)$ avec a et b des booléens.

Exo 63 : En `python`, que renvoie `12 & 17` ; puis `12 | 17` ; puis `12` $\wedge$ `17`. Expliquer.

Exo 64 : En `python`, que renvoie `23 & 17` ; puis `23 | 17` ; puis `23` $\wedge$ `17`. Expliquer.

Exo 65 : En `python`, que renvoie `12 & 14` ; puis `12 | 14` ; puis `12` $\wedge$ `14`. Expliquer.

Exo 66 : Proposer une fonction `python` `puissance2_commune(a,b)` qui vérifie si deux entiers a et b ont tous deux une puissance de 2 commune dans leur décomposition. *i.e.* $5=2^2+2^0$ et $9=2^3+2^0$ renverrait vrai.