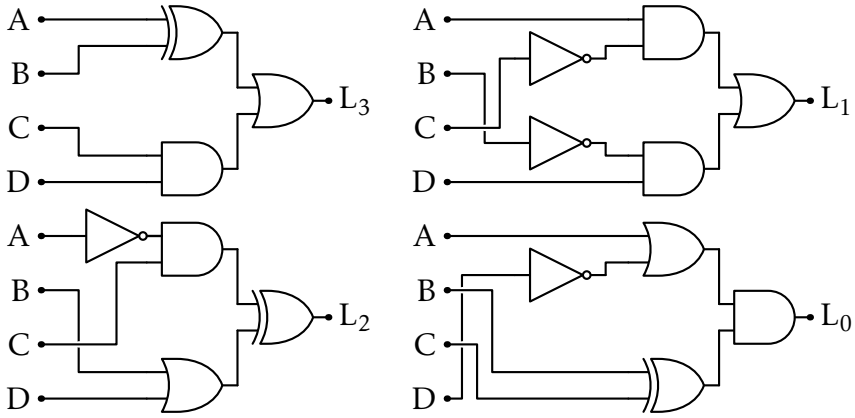
Lycée Les 3 Sources — **Bourg-Lès-Valence**
NSI 1re — Année **2025-26**

Autotest n°2 – correction

Exercice 1 : Lors de la phase finale d'un jeu, les 4 candidats de l'équipe finaliste ont chacun un buzzer. Les 4 buzzers sont reliés à 4 lampes à l'aide de circuits logiques complexes. Appuyer sur un buzzer éteint permet de l'allumer et appuyer une nouvelle fois l'éteint. Les 4 candidats doivent trouver quels buzzers allumer ou pas pour obtenir le motif demandé sur les ampoules. Ils doivent reproduire 4 motifs en 30 s pour gagner.

Partie A : Circuits logiques

Pour simplifier, on donne séparément les circuits de chaque ampoule. Les buzzers sont A, B, C et D, et les lampes sont L_3 à L_0.
On met la valeur 1 à un buzzer ou à une ampoule allumée et 0 s'ils sont éteints.



1) Déterminer les valeurs à donner aux buzzers pour obtenir les motifs demandés sur les lampes.

A	B	C	D	L_3	L_2	L_1	L_0
0	0	0	1	0	1	1	0
1	0	0	1	1	1	1	0
0	0	1	0	0	1	0	1
1	0	1	1	1	1	1	1

Partie B : Nombres binaires

Afin de programmer le processeur contrôlant le dispositif de vérification, qui permet de déterminer si l'équipe gagne ou pas, il faut faire quelques travaux préliminaires sur le binaire. Les résultats attendus sont stockés dans un seul nombre. Par exemple, si on attend, en binaire, 0110_2, 1110_2, 0101_2 puis 1111_2, alors le nombre stocké sera 1111010111100110_2. Pour simplifier l'écriture de ce nombre, on utilisera l'hexadécimal.

2) Expliquer pourquoi un nombre exprimé sur 4 bits en binaire peut être exprimé avec un seul chiffre en hexadécimal. **Solution :** Avec 4 chiffres en binaire, on peut aller de 0, 15, ce qui est exactement les chiffres utilisés en hexadécimal.

3) Donner l'écriture en hexadécimal des 4 nombres représentant les lampes de la partie A. **Solution :** $0110_2 = 6 = 6_{16}$, $1110_2 = 14 = E_{16}$, $0101_2 = 5 = 5_{16}$ et $1111_2 = 15 = F_{16}$.

4) On admet que si on recolle les nombres écrits en hexadécimal ou si on recolle les nombres écrits en binaire sur 4 bits, c'est la même chose.
Donner l'écriture hexadécimale de 1111010111100110_2. **Solution :** $F5E6_{16}$

5) Déterminer les 4 motifs, dans l'ordre dans lequel ils seront demandés, correspondant à $CB27_{16}$. **Solution :** $7_{16} = 0111_2$, $2_{16} = 10_2$, $B_{16} = 11 = 1011_2$ et $C_{16} = 12 = 1100_2$.

Afin d'extraire le premier nombre attendu du résultat global, on utilise le ET bit à bit. Pour cette opération, on applique l'opération booléenne ET entre le dernier bit du premier nombre et celui du deuxième, puis le ET entre l'avant-dernier bit de chacun des deux nombres et ainsi de suite. Il n'y a pas de retenues, comme dans une addition.

```
  0 1 1 0 1
ET 0 1 0 1 1
  ---------
  0 1 0 0 1
```

6) Déterminer le résultat du ET bit à bit entre 110101_2 et 100110_2. **Solution :**

```
  1 1 0 1 0 1
ET 1 0 0 1 1 0
  -----------
  1 0 0 1 0 0
```

7) Afin de ne garder que les 4 derniers bits d'un nombre binaire, avec quel nombre est-ce qu'il faut faire le ET bit à bit? Donner le résultat en binaire et en base 10.
Solution : En faisant le ET avec $1111_2 = 15$, on garde les 4 derniers bits.

Afin de passer au résultat attendu suivant, il va falloir supprimer les 4 derniers chiffres du *grand* nombre. Il y a des instructions spécifiques en assembleur pour faire cela, mais on peut aussi le faire facilement en utilisant la division euclidienne.

8) Par quel nombre, exprimé en base 10, faut-il diviser un nombre pour décaler ses chiffres de 4 rangs vers la droite? Par exemple, pour passer de 11010010_2 à 1101_2.
Solution : À chaque fois qu'on divise par 2, on décale d'un rang vers la droite. Il faut donc diviser 4 fois par 2, ce qui revient à diviser par $2^4 = 16$.

Partie C : Assembleur

Pour tester les résultats donnés par les joueurs, la production du jeu utilise un appareil contenant un processeur 16 bit. Les 4 lampes $L_3L_2L_1L_0$ des joueurs sont directement reliées au registre R10 du processeur. On associe leur état à un nombre binaire de 4 bits, avec la valeur de L_3 correspondant au bit de gauche et celle de L_0 au bit de droite.
Le programme utilisé est le suivant, ainsi que l'explication des instructions :

```
1  init    MOV    R0, #0xCB27
2  debut   MOV    R1, R10
3          AND    R2, R0, #15
4          CMP    R1, R2
5          BNE    perdu
6          LSR    R0, R0, #4
7          CMP    R0, #0
8          BNE    debut
9  gagne   END
10 perdu   END
```

Assembleur	Signification
`MOV dest, op1`	`dest` = `op1`
`AND dest, op1, op2`	Fait le ET bit à bit entre `op1` et `op2`. Place le résultat dans `dest`.
`LSR dest, op1, op2`	Enlève les `op2` bits de droite de `op1`. Place le résultat dans `dest`
`CMP op1, op2` `BNE label`	Aller à l'instruction `label` si `op1` ≠ `op2`

Pour simplifier, on ne tient pas compte des mécanismes permettant de synchroniser la boucle avec les réponses des joueurs. On suppose aussi que lorsqu'on arrive en `gagne`, un bout de code correspondant à la victoire est exécuté. De la même manière, si on arrive en `perdu`, un bout de code correspondant à une défaite est exécuté.

9) Quelles sont les deux conditions qui font sortir de la boucle? **Solution :** On sort si à la ligne 4, R1 ≠ R2 ou si à la ligne 7 R0 ≠ 0. C'est-à-dire si les joueurs rentrent un mauvais code, et dans ce cas ils perdent, ou s'il n'y a plus de code à tester, et là ils gagnent.

10) À quelle opération arithmétique expliquée dans la partie B correspond la ligne 6?
Solution : Cela revient à divisier par 16.

11) À quoi sert le contenu du registre R0? **Solution :** Il contient les codes à tester.

12) Si les joueurs trouvent les bonnes combinaisons à chaque fois, quelles sont les valeurs successives de R0, exprimées en hexadécimal?
Solution : Il vaudra successivement CB27, CB2, CB, C et enfin 0.

Exercice 2 : On considère la fonction suivante, qui prend deux listes de même longueur.

```
def inconnu(liste1, liste2):
    nouvelle_liste = []
    for i in range(len(liste1)):
        if liste2[i] == 1:
            nouvelle_liste.append(liste1[i])
        # compléter une ligne du tableau
    return nouvelle_liste
```

1) Compléter le tableau ci-contre pour l'exécution de `inconnu([3, 1, 9, 0], [1, 1, 0, 1])`.

i	liste1[i]	liste2[i]	nouvelle_liste
			[]
0	3	1	[3]
1	1	1	[3, 1]
2	9	0	[3, 1]
3	0	1	[3, 1, 0]

2) Quelle est la valeur renvoyée pour `inconnu([1, 1, 0, 1, 0], [1, 0, 0, 1, 1])` ?
Solution : On obtient `[1, 1, 0]`.

3) Décrire en français ce que fait cette fonction. Par exemple : "elle fait la somme de tous les éléments de la liste".
Solution : La fonction renvoie une nouvelle liste qui ne contient que les éléments de `liste1` tels que l'élement de `liste2` de même indice vaut 1.