



**INF2031L**

**ARCHI : Architecture des ordinateurs**

---

**TD**

**2025 – 2026**

# Table des matières

<b>1</b>	<b>Vue d'ensemble de l'ordinateur</b>	<b>3</b>
1.1	Petits calculs autour d'une carte mère . . . . .	3
1.2	Taille de la mémoire centrale . . . . .	4
1.3	Circuits mémoires . . . . .	4
1.4	Taille du bus, volume de mémoire centrale . . . . .	4
1.5	Langage machine . . . . .	4
<b>2</b>	<b>Logique propositionnelle</b>	<b>6</b>
2.1	Évaluation de formules . . . . .	6
2.2	Validité, satisfiabilité . . . . .	6
2.3	Interprétation de formules . . . . .	6
2.4	Modélisation - Zoé va à Paris . . . . .	6
2.5	Modélisation - Frodon . . . . .	6
<b>3</b>	<b>Codage des nombres</b>	<b>7</b>
3.1	Représentation positionnelle des entiers naturels . . . . .	7
3.2	Représentation en complément à deux sur $p$ bits . . . . .	7
3.3	Codage de nombres en machine . . . . .	8
3.4	Conversion base 10 vers base 2 d'entiers naturels . . . . .	8
3.5	Représentation positionnelle des rationnels . . . . .	8
<b>4</b>	<b>Circuits combinatoires</b>	<b>11</b>
4.1	Comparaison d'entiers naturels . . . . .	11
4.2	Décalage . . . . .	12
4.3	Générateur de parité impaire . . . . .	12
4.4	Des circuits combinatoires . . . . .	12
4.5	Petits circuits combinatoires . . . . .	13
4.6	Encodeur octal . . . . .	13
4.7	Addition en complément à 2 sur $n$ bits . . . . .	14
<b>5</b>	<b>Circuits séquentiels</b>	<b>17</b>
5.1	Génération d'un circuit séquentiel . . . . .	17
5.2	Un générateur de séquence simple . . . . .	17
5.3	Détection de séquences spécifiques . . . . .	18
5.4	Circuit compteur . . . . .	19
5.5	Génération d'un signal périodique . . . . .	20
5.6	Addition séquentielle . . . . .	21
5.7	Analyse d'un circuit séquentiel . . . . .	22
<b>6</b>	<b>Programmation en langage d'assemblage</b>	<b>23</b>
6.1	Autour de la sommation . . . . .	25
6.2	Multiplication par 6 des entiers d'un tableau . . . . .	25
6.3	Programme Mystère LC3 . . . . .	26
6.4	Saisie d'une chaîne de caractères . . . . .	27
6.5	Décompte de bits non-nuls . . . . .	27
6.6	Débogage . . . . .	28
6.7	Nombre d'occurrences . . . . .	28

# TD 1

## Vue d'ensemble de l'ordinateur

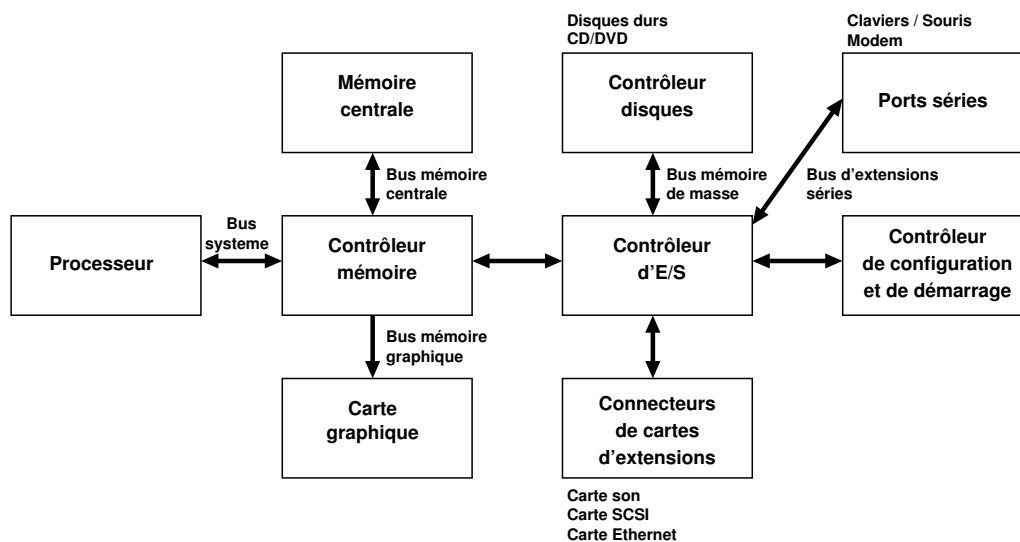
### Exercice 1 : Petits calculs autour d'une carte mère

Les tailles des mémoires sont des puissances de deux ou des multiples de telles puissances. On utilise parfois les préfixes SI en leur associant la puissance de 2 la plus proche : par exemple 1024 octets est souvent abusivement noté 1 ko, bien que 1 ko = 1000 o... La norme CEI-60027-2 définit des *préfixes binaires* pour éviter la confusion :

préfixe standardisé	symbole standardisé	puissance de 2	puissance de 10 la plus proche
tébi	Ti	$2^{40}$	$10^{12} = T$
gibi	Gi	$2^{30}$	$10^9 = G$
mébi	Mi	$2^{20}$	$10^6 = M$
kibi	ki	$2^{10}$	$10^3 = k$

Ainsi, 1024 octets = 1 kibi-octet = 1 kio. On utilisera ces préfixes binaires par la suite.

La *carte mère* d'un PC regroupe des fonctionnalités de base d'un ordinateur : processeur, mémoire centrale et gestion des E/S.



### Partie 1 : Bande passante d'un bus

La bande passante d'un bus, aussi appelée débit crête, est la quantité de données pouvant circuler sur ce bus par unité de temps. Sur la carte mère d'un PC, le bus reliant le processeur au contrôleur mémoire est le bus système, souvent appelé *Front Side Bus* (FSB). Supposons ici que le FSB d'un certain ordinateur est capable d'assurer le transfert de 8 octets à la fréquence de 400 MHz.

Un contrôleur mémoire prend à sa charge les échanges entre le processeur et la mémoire centrale, entre le processeur et le contrôleur d'E/S, et entre le processeur et la mémoire vidéo. Le contrôleur mémoire peut en outre mettre en place un accès direct entre le contrôleur d'E/S et la mémoire centrale ou la mémoire vidéo (accès DMA pour *Direct Memory Access*) : le contrôleur d'E/S pourra par exemple transférer directement des données d'un périphérique à la mémoire vidéo sans qu'elles transitent par le processeur.

- 1) Quelle est la bande passante du bus FSB considéré exprimée en Go/s? Rappelons que  $1 \text{ Go} = 10^9 \text{ o}$ .
- 2) Quelle est la bande passante du bus FSB considérée exprimée en Gio/s? Rappelons que  $1 \text{ Gio} = 2^{30} \text{ o}$ .
- 3) Supposons que le bus mémoire centrale permet le transfert de mots de 32 bits à la fréquence de 266 MHz. Quelle est la bande passante du bus mémoire centrale en Go/s?
- 4) Que penser de la différence de bande passante entre le bus de la mémoire centrale et celle du FSB?

## Partie 2 : Lecture d'un film

Un film est lu à partir d'un disque dur, connecté via le bus IDE au contrôleur de disque. Le film est non-compressé, et constitué d'une succession d'images de  $512 \times 384$  pixels en 256 couleurs. On suppose que le défilement des images se fait en 24 images par seconde.

- 1) Quels sont les bus utilisés pour le transfert ?
- 2) Quel est le débit (en Mo/s) requis pour le transfert du film du disque dur à la mémoire vidéo ?
- 3) Supposons que le bus de la mémoire vidéo a une bande passante identique à celle du bus de la mémoire centrale. Quelle est la part (en pourcentage) de la bande passante du bus de la mémoire vidéo consommée par la lecture du film ?

## Exercice 2 : Taille de la mémoire centrale

Supposons que la mémoire centrale d'un certain ordinateur soit adressée octet par octet.

- 1) Si les adresses sont codées sur 32 bits, quelle est la capacité maximale de la mémoire, exprimée en Mio et en Gio ?

## Exercice 3 : Circuits mémoires

Sur la carte mère d'un certain ordinateur, on trouve :

- un processeur 8 bits, ce qui implique que la mémoire est adressée octet par octet, et que le processeur peut lire et écrire des mots de 8 bits en mémoire centrale ;
- un bus système, dont le bus d'adresses est d'une largeur de 24 bits ;
- un certain nombre de circuits mémoire 8 bits, chacun d'une capacité de 512 Kio.

- 1) Quelle est la largeur, en nombre de bits, du bus de données ?
- 2) En supposant que le bus d'adresses ne sert à adresser que la mémoire centrale de l'ordinateur, combien de circuits mémoire sont présents sur la carte mère ?

## Exercice 4 : Taille du bus, volume de mémoire centrale

Sur un certain ordinateur, on suppose que le bus entre le processeur et la mémoire centrale comporte 32 fils d'adresse.

1. Si à chaque adresse de la mémoire centrale correspond un octet :
  - (a) Quel est le nombre d'octets adressables ?
  - (b) Quelle est la taille maximale de la mémoire ? Exprimez votre réponse en octet puis en Gio.
  - (c) Combien de fils de donnée doit comporter le bus ?
2. Si à chaque adresse correspond un mot de 32 bits :
  - (a) Quel est le nombre de mots adressables ?
  - (b) Quelle est la taille maximale de la mémoire ? Exprimez votre réponse en octet puis en Gio.
  - (c) Combien de fils de donnée doit comporter le bus ?

## Exercice 5 : Langage machine

On se place sur un processeur hypothétique, qui accède à une mémoire centrale dans laquelle la taille d'une case mémoire est de 2 octets. Ce processeur dispose des registres suivants :

- **IR**, le registre d'instruction ;
- **PC**, le compteur de programme ;
- **A** (comme accumulateur), un registre temporaire pour le stockage du résultat des opérations.

Les instructions sont codées comme suit :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				adresse											

On détaille les instructions suivantes :

mnémotique	opcode		opération réalisée
LOAD	$(0001)_2$	$(1)_H$	charger le mot dont l'adresse est donnée dans <b>A</b>
STORE	$(0010)_2$	$(2)_H$	stocker le mot contenu dans <b>A</b> à l'adresse donnée
ADD	$(0101)_2$	$(5)_H$	ajouter l'entier naturel à l'adresse donnée à <b>A</b> .

- 1) Combien d'instructions, suivant le codage indiqué ci-dessus, peut compter le jeu d'instruction?
- 2) Quel est le nombre maximal d'adresses auxquelles une telle instruction peut faire référence?
- 3) On considère le morceau de programme suivant, écrit en « langage d'assemblage » :

```

LOAD (130)H
ADD (131)H
ADD (132)H
STORE (133)H

```

Que fait ce programme?

- 4) Traduisez ce programme en langage machine, et représentez le dans la mémoire centrale en plaçant la première instruction à l'adresse  $(100)_H$ .
- 5) On suppose que le contenu des cases mémoires  $(130)_H$  à  $(133)_H$  est initialement le suivant :

$(130)_H$	$(0002)_H$
$(131)_H$	$(0003)_H$
$(132)_H$	$(0001)_H$
$(133)_H$	$(0022)_H$

Représentez le contenu des cases mémoires  $(130)_H$  à  $(133)_H$ , ainsi que celui des registres **PC**, **IR** et **A** après l'exécution de chacune des instructions considérées.

- 6) Proposez un morceau de programme pour échanger le contenu des cases mémoires  $(130)_H$  et  $(131)_H$ . Écrivez votre proposition en langage d'assemblage.

# TD 2

## Logique propositionnelle

### Exercice 1 : Évaluation de formules

Ecrire les tables de vérité des formules suivantes, préciser si elles sont valides, satisfiables, etc. :

1.  $(p \Rightarrow q) \Rightarrow q \Rightarrow p$
2.  $((p \wedge q) \vee (\neg p \wedge \neg q)) \wedge ((\neg p \wedge q) \vee (p \wedge \neg q))$
3.  $((p \Rightarrow q) \vee s) \wedge ((s \Rightarrow q) \vee \neg p)$
4.  $((p \Rightarrow q) \Rightarrow r) \Rightarrow s$
5.  $(\neg(\neg((p \vee q) \Rightarrow s) \Rightarrow \neg((p \wedge r) \Rightarrow s)))$
6.  $((p \Rightarrow q) \wedge (r \Rightarrow s)) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge s))$
7.  $(\neg p \wedge \neg q) \vee (\neg p \wedge q) \vee (p \wedge q) \vee r$

### Exercice 2 : Validité, satisfiabilité

Montrer qu'une formule  $F$  est valide si et seulement si  $\neg F$  n'est pas satisfiable.

### Exercice 3 : Interprétation de formules

- 1) Soient deux interprétations différentes définies sur le même ensemble de variables.
  - (a) Dire s'il est possible de trouver une formule qui n'est pas évaluée de la même façon par ces interprétations.
  - (b) Dire s'il est possible de trouver une formule évaluée de la même façon par ces interprétations.
- 2) Soit  $F$  une formule propositionnelle n'utilisant que les connecteurs  $\vee$  et  $\wedge$ . Montrer que toute interprétation  $I$  telle que  $I(p) = 1$  si  $p$  apparaît dans  $F$  vérifie  $I \models F$ .

### Exercice 4 : Modélisation - Zoé va à Paris

On considère les propositions suivantes :

- Si Alice et Julie viennent à Paris, Zoé viendra aussi,
  - Si Julie vient à Paris, Alice aussi,
  - Julie ou Zoé, l'une des deux au moins, viendra à Paris.
1. Formaliser ces 3 propositions en logique des propositions.
  2. Démontrer que Zoé viendra à Paris en faisant une démonstration sémantique.

### Exercice 5 : Modélisation - Frodon

On considère les propositions suivantes :

- Si Frodon ne va pas au Mordor, Sauron prend le pouvoir,
  - Si Sauron prend le pouvoir, Frodon est triste,
  - Si Frodon va au Mordor, il ne possède pas l'anneau,
  - Si Frodon ne possède pas l'anneau, il est triste.
1. Formaliser ces propositions en logique propositionnelle.
  2. Montrer que Frodon est triste en utilisant un raisonnement sémantique.

# TD 3

## Codage des nombres

Dans cette partie, le petit encadré « **à savoir !** » est utilisé au début des paragraphes qui peuvent être considérés comme du cours.

### Exercice 1 : Représentation positionnelle des entiers naturels

**à savoir !** Soit  $\beta \in \mathbf{N}$ ,  $\beta > 1$ , une base. Tout  $n \in \mathbf{N}$  peut être représenté de manière unique par sa *représentation positionnelle en base  $\beta$*  :

$$(x_{p-1}x_{p-2}\cdots x_1x_0)_\beta := \sum_{i=0}^{p-1} x_i\beta^i.$$

Les  $x_i \in \{0, 1, \dots, \beta - 1\}$  sont les *chiffres* de l'écriture de  $n$  en base  $\beta$ ,  $p$  est le nombre de chiffres nécessaires pour écrire de l'entier naturel  $n$ . On attribue un symbole à chaque chiffre : chiffres 0 et 1 en binaire, chiffres de 0 à 9 en décimal, chiffres de 0 à F en hexadécimal.

- 1) La définition permet directement d'effectuer des changements de base : si on connaît l'écriture  $(x_{p-1}\dots x_0)_\beta$  de  $n$ , et qu'on veut connaître cette écriture en base  $\gamma$ , il suffit de convertir les chiffres  $x_i$  en base  $\gamma$ , puis de calculer  $\sum_{i=0}^{p-1} x_i\beta^i$  en base  $\gamma$ . En utilisant cette technique, convertissez  $(11011)_2$  puis  $(56)_9$  en décimal.
- 2) Une autre technique de changement de base est celle des divisions euclidiennes successives. Rappelez et justifiez cette méthode. Utilisez là pour convertir  $n = (423)_{10}$  en binaire, puis  $(3452)_{10}$  en base 8.
- 3) Entre les bases 2, 8 et 16, des méthodes plus directes peuvent être utilisées : par exemple, tout chiffre octal est représenté par un entier sur trois bits, et tout entier sur trois bits est représenté par un chiffre octal. Justifiez cette méthode de conversion entre les bases 2 et 8. Convertissez  $(34521)_8$  en base 2 puis 16.

### Exercice 2 : Représentation en complément à deux sur $p$ bits

#### Partie 1 : Définition

**à savoir !** Soit un entier relatif  $n$  à coder en machine sur  $p$  bits. On adopte la définition suivante pour le codage de  $n$  en complément à 2 sur  $p$  bits :

$$(c_{p-1}c_{p-2}\dots c_1c_0)_2 := -c_{p-1}2^{p-1} + \sum_{i=0}^{p-2} c_i2^i.$$

Une manière d'interpréter le codage en complément à 2 est donc de considérer que le bit le plus à gauche a un poids négatif  $(-2^{p-1})$ . On peut montrer que  $n \geq 0$  ssi  $c_{p-1} = 0$ , et  $n < 0$  ssi  $c_{p-1} = 1$ .

- 1) Quelle est la valeur décimale codée par  $(1000011)_2$ ? Par  $(00001010)_2$ ?
- 2) Comment coder  $(-120)_{10}$  en complément à 2 sur 8 bits?
- 3) Quelle est le plus grand entier représentable en complément à 2 sur  $p$  bits? Le plus petit?

#### Partie 2 : Addition et calcul de l'opposé

**à savoir !** Soient  $m = (c_m)_2$  et  $n = (c_n)_2$  en complément à 2 sur  $p$  bits. On dit qu'il y a dépassement de capacité dans une opération en complément à 2 sur  $p$  bits lorsque le résultat de l'opération ne peut pas être représenté sous cette même forme (il est soit trop petit, soit trop grand).

- En l'absence de dépassement de capacité, le codage de  $m + n$  en complément à 2 sur  $p$  bits est le codage de l'entier naturel  $(c_m + c_n) \bmod 2^p$ . Si  $m$  et  $n$  sont de même signe, il y a dépassement ssi le signe du résultat calculé diffère du signe des opérands; s'ils sont de signes opposés, aucun dépassement n'est possible.
- En l'absence de dépassement de capacité, le codage de  $-n$  en complément à 2 sur  $p$  bits est le même que celui de l'entier naturel  $(\bar{c}_{p-1}\bar{c}_{p-2}\dots\bar{c}_1\bar{c}_0)_2 + 1 \bmod 2^p$ . Il y a dépassement ssi le signe du résultat calculé est le même que celui de l'opérande.

- 1) Posez, en complément à deux sur 8 bits, les additions suivantes :  $(10001010)_2 + (00001011)_2$ ,  $(10001010)_2 + (10001011)_2$ ,  $(01001010)_2 + (11001010)_2$ .
- 2) Calculez l'opposé de  $(10001010)_2$  en complément à 2 sur 8 bits, et vérifiez que votre résultat est correct.
- 3) En complément à 2 sur  $p$  bits, quel est le seul cas produisant un dépassement de capacité pour le calcul de l'opposé?

### Partie 3 : Extension de signe en complément à 2

- 1) Comment sont représentés  $(34)_{10}$  et  $(-42)_{10}$  en complément à 2 sur 8 bits (complément à  $2^8$ )?
- 2) Comment sont représentés  $(34)_{10}$  et  $(-42)_{10}$  en complément à 2 sur 12 bits (complément à  $2^{12}$ )?
- 3) Proposez et justifiez une règle permettant de passer de l'écriture d'un entier relatif en complément à 2 sur  $p$  bits à son écriture en complément à 2 sur  $p + k$  bits (en conservant l'égalité des valeurs bien-entendu).

### Exercice 3 : Codage de nombres en machine

- 1) Convertir les nombres suivants en base 10 :  $(1011)_2$ ,  $(10110)_2$ ,  $(101.1)_2$ ,  $(0.1101)_2$ ,  $(110.01)_2$ .
- 2) Convertir les nombres suivants en base 10 :  $(FF)_{16}$ ,  $(1A)_{16}$ ,  $(789)_{16}$ ,  $(0.13)_{16}$ ,  $(ABCD.EF)_{16}$ .
- 3) Convertir les nombres suivants (exprimés en base 10) en base 2 et en base 16 : 12, 24, 192, 2079, 0.25, 0.375, 0.376 et 17.150.

### Exercice 4 : Conversion base 10 vers base 2 d'entiers naturels

Le principe de l'algorithme conversion de Horner de l'entier naturel  $n = (x_3x_2x_1x_0)_\beta$  est le suivant. On a :

$$n = x_3 \cdot \beta^3 + x_2 \cdot \beta^2 + x_1 \cdot \beta^1 + x_0, = (x_3 \cdot \beta^2 + x_2 \cdot \beta^1 + x_1) \cdot \beta + x_0, = ((x_3 \cdot \beta + x_2) \cdot \beta + x_1) \cdot \beta + x_0.$$

Si on exécute l'algorithme suivant,

$$\begin{aligned} r_3 &= x_3 \\ r_2 &= r_3 \times \beta + x_2 \\ r_1 &= r_2 \times \beta + x_1 \\ r_0 &= r_1 \times \beta + x_0 \end{aligned}$$

alors par construction on a l'égalité  $r_0 = x_3 \cdot \beta^3 + x_2 \cdot \beta^2 + x_1 \cdot \beta^1 + x_0$ . Si on effectue tous les calculs dans une base d'arrivée  $\gamma$ , on obtient l'écriture de  $n$  en base  $\gamma$ .

- 1) Testez l'algorithme de Horner pour convertir l'entier  $n = (11010101)_2$  en décimal.
- 2) Plus fastidieux à faire à la main : testez l'algorithme de Horner pour convertir l'entier  $n = (567)_{10}$  en binaire. Combien de multiplications avez vous posées? Combien de multiplications auriez vous posées si vous aviez utilisé la définition de la notation positionnelle pour faire cette conversion?
- 3) Ecrivez maintenant deux fonctions en C pour convertir une chaîne de caractères (tableau de codes ASCII terminé par un 0) contenant l'écriture d'un entier positif en décimal, en un entier non-signé en machine<sup>1</sup> : l'une utilisera naïvement la définition de la notation positionnelle, l'autre l'algorithme de Horner. Dans les deux cas, combien de multiplications sont utilisées?
- 4) Pour poursuivre : dans les deux fonctions de conversion, peut-il se produire un dépassement de capacité? Le cas échéant, que vont retourner ces fonctions? Dans la même veine, mettre au point une fonction pour convertir un entier naturel écrit en hexadécimal vers un entier naturel du type `unsigned int`.

### Exercice 5 : Représentation positionnelle des rationnels

**à savoir !** Les rationnels sont les nombres de la forme  $\frac{p}{q}$ , avec  $p \in \mathbf{Z}$  et  $q \in \mathbf{N} - \{0\}$ . Tout  $x \in \mathbf{Q}$  positif peut être décomposé en une partie entière  $\lfloor x \rfloor \in \mathbf{N}$  ( $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$ ) et une partie fractionnaire  $\{x\} = x - \lfloor x \rfloor$  ( $0 \leq \{x\} < 1$ ). On utilise la notation positionnelle pour l'écriture de  $\{x\}$  : s'il existe  $q \in \mathbf{N}$  t.q.

$$\{x\} = (0, x_{-1} \dots x_{-q})_\beta = \sum_{i=1}^q x_{-i} \beta^{-i},$$

1. Vous avez peut-être déjà utilisé les fonctions `sscanf` ou `atoi` du C99 pour faire cela, mais il s'agit ici de s'en passer.

alors  $x = (x_{p-1}x_{p-2}\dots x_0, x_{-1}x_{-2}\dots x_{-q})_\beta$  est l'écriture  $x$  en base  $\beta$ . Le problème est que l'écriture d'un rationnel en base  $\beta$  n'est pas forcément finie : en machine, il faut souvent se contenter d'une approximation. Par contre, on sait que l'écriture d'un rationnel est nécessairement périodique.

**à savoir !** Soit un rationnel  $0 \leq x < 1$ . On veut déterminer son écriture  $(0, x_{-1}x_{-2}x_{-3}\dots)_2$  en binaire. On a  $2 \times x = (x_{-1}, x_{-2}x_{-3}\dots)_2$ , donc  $x_{-1} = \lfloor 2 \times x \rfloor$ . En procédant par des multiplications successives par 2, on peut ainsi extraire un par un les bits de l'écriture (binaire) de  $x$ .

- 1) Quelle est l'écriture de  $13/7$  en base  $\beta = 10$ ? L'écriture est périodique, on soulignera la période.
- 2) Convertir  $1/10 = (0, 1)_{10}$ , puis  $(5, 3)_{10}$  en écriture binaire.
- 3) Donnez l'écriture décimale de  $(11, 1001)_2$ .
- 4) Donnez une écriture sous la forme d'une fraction de deux nombres décimaux de  $(0, 0101)_2$ .



# TD 4

## Circuits combinatoires

### Exercice 1 : Comparaison d'entiers naturels

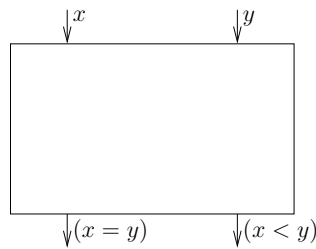
On souhaite mettre au point un circuit permettant de comparer deux entiers naturels codés en binaire,  $a = (a_3 a_2 a_1 a_0)_2$  et  $b = (b_3 b_2 b_1 b_0)_2$ . La sortie du circuit devra prendre pour valeur 1 si  $a \leq b$ , 0 sinon. On va d'abord s'intéresser à la comparaison de deux bits  $x$  et  $y$  avant de passer à celle des entiers naturels : l'expression  $(x = y)$  vaut 1 si  $x$  et  $y$  sont égaux, 0 sinon ; l'expression  $(x < y)$  vaut 1 si  $(x)_2 \leq (y)_2$ , 0 sinon.

- 1) Complétez les tables de vérités suivantes, puis donnez des formules booléennes pour  $(x = y)$  et pour  $(x < y)$ .

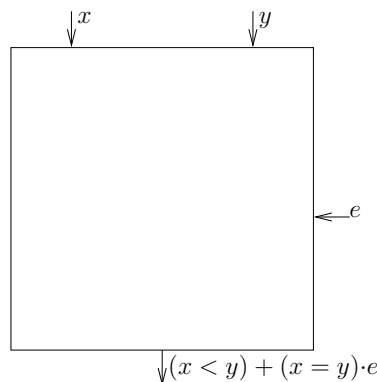
$x$	$y$	$(x = y)$
0	0	
0	1	
1	0	
1	1	

$x$	$y$	$(x < y)$
0	0	
0	1	
1	0	
1	1	

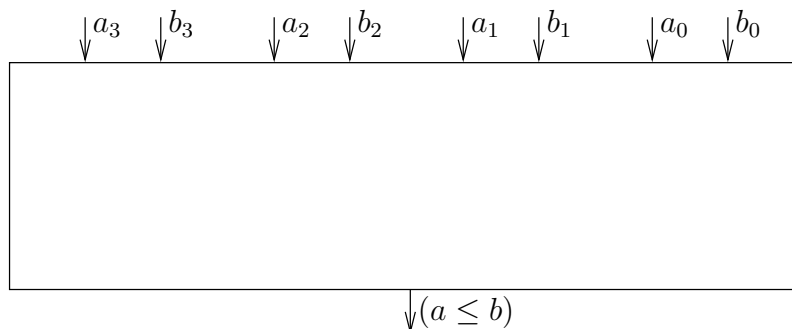
- 2) Complétez le circuit ci-dessous, de manière à ce que ses sorties produisent les valeurs de  $(x = y)$  et  $(x < y)$ . Dans la suite, on appellera HC le composant obtenu.



- 3) Complétez le circuit ci-dessous, qui prend en entrée les deux bits  $x$  et  $y$ , ainsi qu'une entrée supplémentaire  $e$ , et dont la sortie prend la valeur  $s = (x < y) + (x = y) \cdot e$ . Dans la suite, on appellera FC le composant obtenu.



- 4) Complétez le circuit ci-dessous, de manière à réaliser la comparaison entre  $a = (a_3 a_2 a_1 a_0)_2$  et  $b = (b_3 b_2 b_1 b_0)_2$  : la sortie prendra pour valeur 1 si  $a \leq b$ , 0 sinon. Expliquez votre raisonnement.



### Exercice 2 : Décalage

On souhaite mettre au point un circuit permettant le décalage à droite ou à gauche d'un mot de 5 bits. Le circuit prend en entrée un mot  $(e_4 e_3 e_2 e_1 e_0)$  et présente en sortie le résultat du décalage  $(s_4 s_3 s_2 s_1 s_0)$ . Une ligne de contrôle  $c$  permet de spécifier la direction du décalage.

- $c = 1$  correspond à un décalage à droite de 1 bit; un 0 est alors inséré au niveau de  $s_4$ .
  - $c = 0$  correspond à un décalage à gauche de 1 bit; un 0 est alors inséré au niveau de  $s_0$ .
- 1) Pour  $3 \leq i \leq 1$ , proposez un circuit à base de portes NOT, AND et OR permettant de déterminer  $s_i$  en fonction de  $e_{i+1}$ ,  $e_{i-1}$  et  $c$ . On voit que dans les deux cas chaque bit de la sortie  $s$  est calculé à l'aide des deux bits adjacents de l'entrée  $e$ . Par exemple, le bit  $s_2$  vaut soit  $e_1$ , soit  $e_3$  (cela dépend de la valeur de  $c$ ).
  - 2) Proposez un circuit permettant de déterminer  $s_0$  en fonction de  $e_1$ , et  $c$ . Proposez un circuit pour déterminer  $s_4$  en fonction de  $e_3$ , et  $c$ .
  - 3) Représentez le décaleur 5 bits demandé.

Vous pouvez réfléchir à la manière d'associer les décaleurs, de manière par exemple à pouvoir fabriquer un décaleur 4 bits avec deux décaleurs 2 bits.

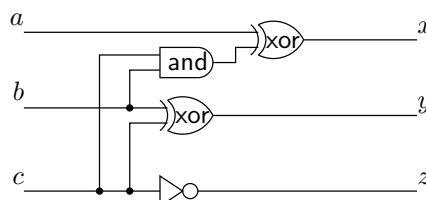
### Exercice 3 : Générateur de parité impaire

Un générateur de parité impaire est une fonction qui retourne 1 si le nombre de bits à 1 parmi ses entrées est impair, 0 sinon. Définissez cette fonction pour un mot de 8 bits, et donnez un circuit logique implantant cette fonction. Vous pouvez commencer par donner une version naïve de votre circuit, mais vous essayerez ensuite de réduire sa latence.

### Exercice 4 : Des circuits combinatoires

#### Partie 1 : Analyse d'un circuit combinatoire

On considère le circuit combinatoire suivant, dont les entrées sont  $a$ ,  $b$  et  $c$ , et les sorties  $x$ ,  $y$  et  $z$  :



- 1) Donnez des expressions booléennes pour  $x$ ,  $y$  et  $z$  en fonction de  $a$ ,  $b$  et  $c$ .
- 2) Complétez la table de vérité suivante :

$a$	$b$	$c$	$x$	$y$	$z$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- 3) Quelle fonction réalise le circuit considéré?

### Partie 2 : Décodeurs

Un décodeur  $k$  bits est un circuits a  $k$  entrées  $e_{k-1}, \dots, e_0$  et  $2^k$  sorties  $s_{2^k-1}, \dots, s_0$  : la sortie  $s_{(e_{k-1}, \dots, e_0)_2}$  dont l'indice est indiqué par les entrées est activée et toutes les autres restent inactives.

1) On représente de la manière indiquée ci-dessous un décodeur 1 vers 2.



Complétez sa table de vérité, et donnez un logigramme pour un décodeur 1 vers 2.

On souhaite fabriquer un décodeur 2 vers 4 à partir de deux décodeurs 1 vers 2. Cela n'est pas facile à réaliser avec des décodeurs classiques, il faut leur ajouter une entrée supplémentaire CS (*Chip Select*). Le rôle de cette entrée est le suivant :

- quand CS=0, les sorties du décodeur restent à 0, quelles que soient les autres entrées;
- quand CS=1, le décodeur se comporte comme un décodeur classique.

1) On représente de la manière indiquée ci-dessous un décodeur 1 vers 2 avec CS.



Complétez sa table de vérité, et donnez un logigramme pour un décodeur 1 vers 2 avec CS.

2) On considère un décodeur 2 vers 4 classique, représenté ci-dessous : dressez sa table de vérité.



3) En utilisant deux décodeurs 1 vers 2 avec CS, ainsi qu'un décodeur 1 vers 2 classique, proposez un circuit réalisant un décodeurs 2 vers 4 classique. Justifiez brièvement votre réponse en vous basant sur la table de vérité de la question précédente.

### Exercice 5 : Petits circuits combinatoires

- 1) Construire un circuit combinatoire à trois entrées  $x_0, x_1, x_2$  capable de détecter si le nombre  $(x_2x_1x_0)_2$  est divisible par 3.
- 2) Le but est de construire un circuit combinatoire à base de portes NOT, AND et OR pour la porte logique XOR à trois entrées  $a, b, c$  : commencez par écrire la table de vérité, puis proposez un circuit en vous basant sur la forme normale disjonctive. La porte XOR à trois entrées réalise la fonction dite d'imparité : pourquoi ce nom?

### Exercice 6 : Encodeur octal

Le but est de concevoir un encodeur octal : il s'agit d'un circuit à 8 entrées  $e_0, \dots, e_7$  et 3 sorties  $s_0, s_1, s_2$ . Si l'entrée  $e_i$  est à 1 et que toutes les autres sont à 0, on veut que les sorties soient telles que  $(s_2s_1s_0)_2 = i$ .

1) Complétez un tableau de la forme suivante, en supposant qu'au plus l'une des entrées  $e_i$  peut être activée à la fois (dans les colonnes  $e_0, \dots, e_7$ , n'indiquez que les 1, pas les 0).

$e_0$	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$	$s_2$	$s_1$	$s_0$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- 2) Exprimez les sorties  $s_2, s_1$  et  $s_0$  chacune sous la forme d'une somme de quatre littéraux.
- 3) Représentez le circuit logique demandé (vous pouvez utiliser des portes OR à 4 entrées).

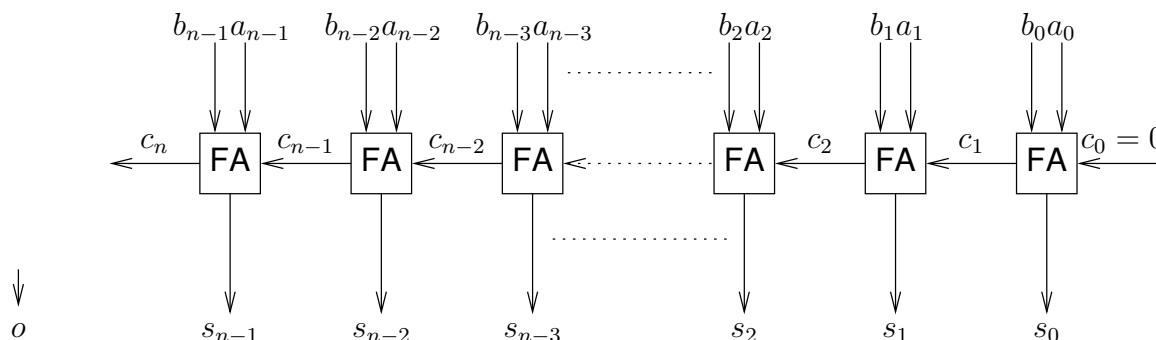
## Exercice 7 : Addition en complément à 2 sur $n$ bits

Soient  $a$  et  $b$  deux entiers relatifs, que l'on connaît par leur représentation en complément à 2 sur  $n$  bits :

$$a = (a_{n-1} \dots a_0)_{\bar{2},n} = -a_{n-1}2^{n-1} + A' \quad \text{avec} \quad A' = (a_{n-2} \dots a_0)_2,$$

$$b = (b_{n-1} \dots b_0)_{\bar{2},n} = -b_{n-1}2^{n-1} + B' \quad \text{avec} \quad B' = (b_{n-2} \dots b_0)_2.$$

On note  $A = (a_{n-1} \dots a_0)_2$  et  $B = (b_{n-1} \dots b_0)_2$ , et  $S = (s_{n-1} \dots s_0)_2 = A + B \pmod{2^n}$ ;  $S$  est donc le résultat de l'addition des représentations de  $a$  et de  $b$  en tant qu'entiers naturels sur  $n$  bits. On étudie l'addition de  $a$  et de  $b$  à l'aide d'un additionneur à propagation de retenue classique, fabriqué à l'aide de *full-adders* (FA) :



### Partie 1 : Dépassement de capacité

On souhaite ajouter à notre additionneur une sortie  $o$  (comme *overflow*) permettant de détecter si on se trouve en présence d'un cas de dépassement de capacité. On dit qu'il y a dépassement de capacité dans une opération en complément à 2 sur  $n$  bits **si le résultat de l'opération ne peut pas être représenté exactement** sur  $n$  bits (il est soit trop petit, soit trop grand).

Rappelons que le plus grand entier représentable en complément à 2 sur  $n$  bits est  $(01 \dots 1)_{\bar{2},n} = 2^{n-1} - 1$ , et que le plus petit est  $(10 \dots 0)_{\bar{2},n} = -2^{n-1}$ .

- 1) Posez, en complément à 2 sur 8 bits (ici,  $n = 8$ ), les additions avec les opérandes  $a$  et  $b$  indiquées ci-dessous; Donnez la valeur des opérandes et celle du résultat calculé (qui peut être différent du résultat exact) en décimal, en précisant bien le signe de ces valeurs. Indiquez clairement les opérations pour lesquelles il y a eu dépassement de capacité.

—  $a = (10001111)_{\bar{2}}, b = (10010000)_{\bar{2}},$

—  $a = (01111111)_{\bar{2}}, b = (10000000)_{\bar{2}},$

—  $a = (01111111)_{\bar{2}}, b = (00000001)_{\bar{2}}.$

- 2) Montrez qu'en l'absence de dépassement de capacité, on a  $a + b = (s_{n-1} \dots s_0)_{\bar{2},n}$ , et qu'il y a dépassement de capacité ssi  $a_{n-1} = b_{n-1} \neq s_{n-1}$ . Pour cela, compléter le tableau suivant. Notez bien que, d'après les notations,  $A' + B' = (\mathbf{c}_{n-1} s_{n-2} \dots s_0)_2$ .

$a, b$	$c_{n-1}$	$S$	$a + b$	$a_{n-1} = b_{n-1} \neq s_{n-1}$
$a \geq 0$ et $b \geq 0$	0	$S = \begin{array}{r} \phantom{0} \\ 0 \ a_{n-2} \ \dots \ a_0 \\ + \ 0 \ b_{n-2} \ \dots \ b_0 \\ \hline ( \ 0 \ s_{n-2} \ \dots \ s_0 \ )_2 \end{array}$	$\begin{aligned} a + b &= A' + B' \\ &= (0s_{n-2} \dots s_0)_2 \\ &= (0s_{n-2} \dots s_0)_{\bar{2},n} \end{aligned}$	faux
	1	$S = \begin{array}{r} \phantom{0} \\ 0 \ a_{n-2} \ \dots \ a_0 \\ + \ 0 \ b_{n-2} \ \dots \ b_0 \\ \hline ( \ 1 \ s_{n-2} \ \dots \ s_0 \ )_2 \end{array}$	$\begin{aligned} a + b &= A' + B' \\ &= (1s_{n-2} \dots s_0)_2 \geq 2^{n-1} \end{aligned}$ <p>Il y a dépassement de capacité.</p>	vrai
$a \geq 0$ et $b < 0$	0	$S = \begin{array}{r} \phantom{0} \\ \phantom{0} \ a_{n-2} \ \dots \ a_0 \\ + \phantom{0} \ b_{n-2} \ \dots \ b_0 \\ \hline ( \phantom{0} \phantom{0} \phantom{0} \ )_2 \end{array}$		
	1	$S = \begin{array}{r} \phantom{0} \\ \phantom{0} \ a_{n-2} \ \dots \ a_0 \\ + \phantom{0} \ b_{n-2} \ \dots \ b_0 \\ \hline ( \phantom{0} \phantom{0} \phantom{0} \ )_2 \end{array}$		
$a < 0$ et $b < 0$	0	$S = \begin{array}{r} \phantom{0} \\ \phantom{0} \ a_{n-2} \ \dots \ a_0 \\ + \phantom{0} \ b_{n-2} \ \dots \ b_0 \\ \hline ( \phantom{0} \phantom{0} \phantom{0} \ )_2 \end{array}$		
	1	$S = \begin{array}{r} \phantom{0} \\ \phantom{0} \ a_{n-2} \ \dots \ a_0 \\ + \phantom{0} \ b_{n-2} \ \dots \ b_0 \\ \hline ( \phantom{0} \phantom{0} \phantom{0} \ )_2 \end{array}$		

3) Proposez une modification du circuit initial permettant de calculer la sortie  $o$  : elle doit prendre la valeur 1 en cas de dépassement de capacité, 0 sinon. Vous exprimerez pour cela  $o$  en fonction de  $c_{n-1}$  et  $c_n$ , en utilisant la question précédente.

### Partie 2 : Additionneur-soustracteur

On souhaite maintenant compléter le circuit de manière à ce qu'il puisse aussi effectuer des soustractions en complément à 2 sur  $n$  bits. Pour cela, on ajoute une entrée  $e$  qui indiquera si le circuit doit calculer l'addition ( $e = 1$ ) ou la soustraction ( $e = 0$ ) des deux entiers placés sur ses entrées.

1) On note  $\bar{B} = (\bar{b}_{n-1} \bar{b}_{n-2} \dots \bar{b}_0)_2$  (complément à 1 de  $B$ ) et  $\bar{b} = (\bar{b}_{n-1} \bar{b}_{n-2} \dots \bar{b}_0)_{\bar{2},n}$ . On admet que, en l'absence de dépassement de capacité, si  $S = (A + \bar{B}) + 1 \pmod{2^n}$ , alors  $(s_{n-1} \dots s_0)_{\bar{2},n} = a - b$ . Testez, en complément à 2 sur 8 bits, la soustraction avec les opérandes  $a$  et  $b$  indiquées ci-dessous. Précisez à chaque fois s'il y a ou non dépassement de capacité.

- $a = (11111111)_{\bar{2}}, b = (10000000)_{\bar{2}}$ ,
- $a = (10000000)_{\bar{2}}, b = (00000001)_{\bar{2}}$ ,
- $a = (00000000)_{\bar{2}}, b = (10000000)_{\bar{2}}$ .

2) Soit  $D = (d_{n-1} d_{n-2} \dots d_0)_2$  l'entier de  $n$  bits tel que

- si  $e = 1$  alors  $d = B$ ,
- si  $e = 0$  alors  $d = (\bar{b}_{n-1} \bar{b}_{n-2} \dots \bar{b}_0)_2$ .

Pour  $0 \leq i < n$ , complétez la table de vérité ci-dessous, et exprimez  $d_i$  par une fonction booléenne.

$b_i$	$e$	$d_i$
0	0	
0	1	
1	0	
1	1	

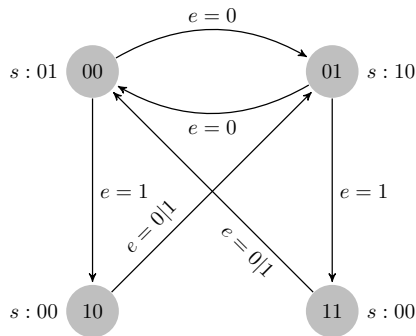
- 3) Donnez le circuit d'un additionneur-soustracteur en complément à 2 sur 8 bits (on admet que les cas de dépassement de capacité peuvent être détectés comme dans la première partie). Veillez à annoter votre schéma (il faut faire apparaître en particulier : les  $a_i$ ,  $b_i$ ,  $c_i$ ,  $d_i$ ,  $s_i$ , ainsi que  $e$  et  $o$ ).

# TD 5

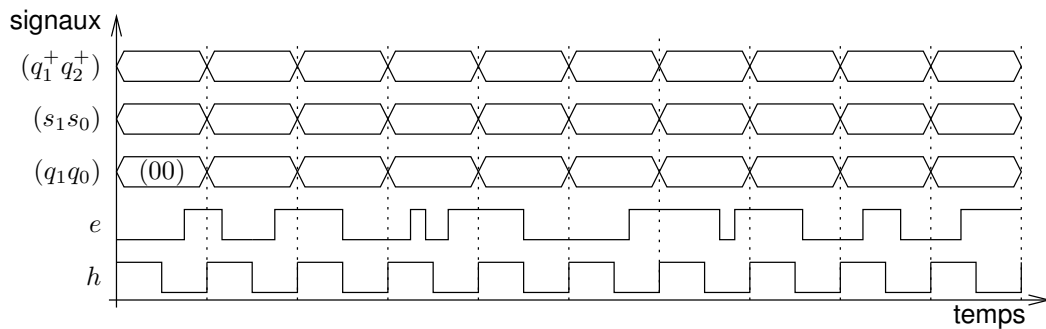
## Circuits séquentiels

### Exercice 1 : Génération d'un circuit séquentiel

On considère l'automate fini séquentiel ci-dessous. Le circuit séquentiel correspondant reçoit un signal d'horloge  $h$ , prend une entrée  $e$ , et produit une sortie  $s = (s_1 s_0)$ . Il comporte 2 bascules flip-flops, régies par le front montant de l'horloge  $h$ , qui stockent l'état courant  $(q_1 q_0)$  du circuit. La fonction de transition du circuit est la fonction  $f$  telle que  $(q_1^+, q_0^+) = f(q_1, q_0, e)$ . La fonction de sortie  $g$  est telle que  $s = g(q_1, q_0)$ .



1) Complétez le chronogramme suivant.



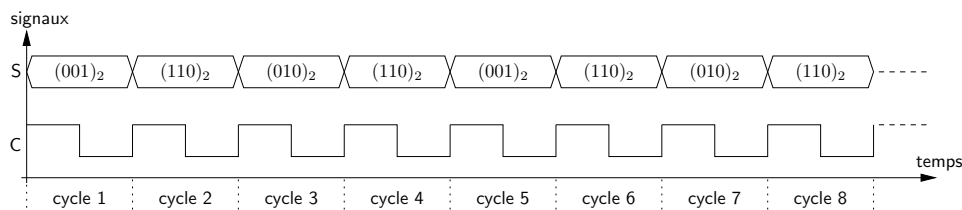
2) Complétez la table de vérité suivante.

$q_1$	$q_0$	$e$	$q_1^+$	$q_0^+$	$s_1$	$s_0$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

- 3) Exprimez  $q_1^+$  et  $q_0^+$  en fonction de  $q_1$ ,  $q_0$  et  $e$  par des expressions booléennes.
- 4) Exprimez  $s_1$  et  $s_2$  en fonction de  $q_1$  et  $q_0$  par des expressions booléennes.
- 5) Donnez un circuit séquentiel implantant l'automate fini étudié.

### Exercice 2 : Un générateur de séquence simple

On souhaite mettre au point un circuit séquentiel permettant de générer la séquence de valeurs  $(001)_2$ ,  $(110)_2$ ,  $(010)_2$ ,  $(110)_2$ , et la répéter de façon périodique. On note  $S = (s_2, s_1, s_0)$  la sortie du circuit. Un chronogramme représentant l'évolution de la valeur de  $S$  au cours du temps sera par exemple le suivant :



Pour concevoir ce circuit séquentiel, on va le modéliser à l'aide d'un automate fini séquentiel.

1) Quatre états sont suffisants pour réaliser l'automate fini demandé : pourquoi?

Comme quatre états sont suffisants, on utilise un registre (à base de bascules flip-flops) à 2 bits pour le stockage de l'état courant  $Q = (q_1, q_0)$  du circuit séquentiel. On choisit la correspondance suivante entre chacun des états de l'automate et sa sortie.

état $Q$	sortie $S$
00	001
01	110
10	010
11	110

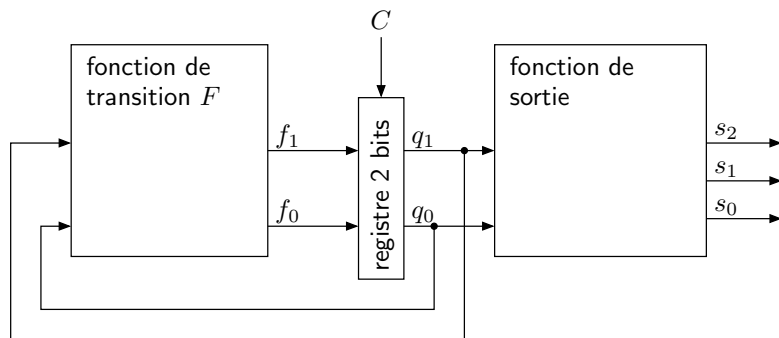
- 1) Donnez une représentation graphique de l'automate fini demandé (graphique comportant les états et les transitions de l'automate).
- 2) Soit  $F$  la fonction de transition de l'automate : donnez la table de vérité de  $F(Q)$  en fonction de  $Q$ . Complétez pour cela le tableau de vérité suivant :

$Q$		$F(Q)$	
$q_1$	$q_0$	$f_1$	$f_0$
⋮	⋮	⋮	⋮

- 3) Exprimez  $f_1$  et  $f_0$  en fonction de  $q_1$  et de  $q_0$  à l'aide de formules booléennes simples.
- 4) Complétez la table de vérité de  $s_2, s_1$  et  $s_0$  en fonction de  $q_1$  et  $q_0$ .

$Q$		$S(Q)$		
$q_1$	$q_0$	$s_2$	$s_1$	$s_0$
⋮	⋮	⋮	⋮	⋮

- 5) Exprimez  $s_2, s_1$  et  $s_0$  en fonction de  $q_1$  et  $q_0$  à l'aide de formules booléennes simples.
- 6) Complétez le circuit ci-dessous, afin d'obtenir le circuit séquentiel demandé.



7) Sur un chronogramme, représenter l'évolution de  $Q, F(Q)$  et  $S(Q)$  sur 8 cycles d'horloge, en supposant que l'état initial est  $Q = (0, 0)$ .

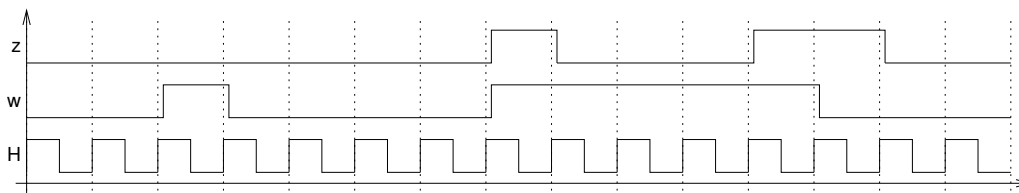
### Exercice 3 : Détection de séquences spécifiques

On souhaite implanter un circuit séquentiel qui reconnaît deux séquences spécifiques quand celles-ci sont reçues à l'entrée  $w$ . Ces séquences consistent en quatre 1 consécutifs ou quatre 0 consécutifs. Le circuit possède une sortie  $z$  sur laquelle il génère le signal suivant :

- $z$  prend la valeur 1 sur un cycle si lors des quatre cycles précédents  $w$  a conservé la valeur 1, ou a conservé la valeur 0;

— sinon,  $z$  prend la valeur 0.

Le chronogramme suivant illustre le fonctionnement du circuit ( $H$  désigne le signal d'horloge) :

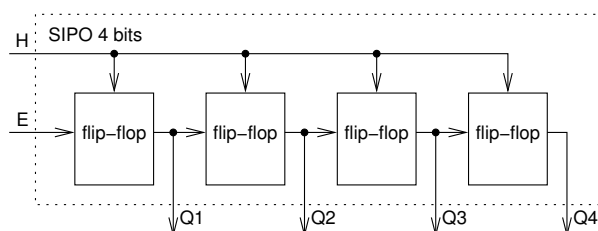


### Partie 1 : Avec un automate séquentiel classique

- 1) Donnez un automate fini séquentiel permettant de modéliser le circuit demandé. Vous choisirez un codage (simple et logique) des états de l'automate, et vous indiquerez la valeur de la sortie  $z$  dans chacun des états de l'automate. Établissez la table de transition de l'automate en encodant les états en binaire, puis la table de vérité pour la sortie  $z$  en fonction de l'état courant.
- 2) Exprimez la fonction de sortie, c'est à dire  $z$  en fonction de l'état courant  $(q_3, q_2, q_1, q_0)$ , par une formule booléenne.
- 3) Combien de termes comportent les formes normales disjonctives pour chaque fonction partielle de la fonction de transition  $F : (q_3, q_2, q_1, q_0) \mapsto (F_3, F_2, F_1, F_0)$ ? Est-ce la bonne façon de concevoir notre circuit?

### Partie 2 : Avec des registres à décalage

Au lieu d'implanter le circuit considéré sous la forme d'un automate, on va essayer d'utiliser un registre à décalage.



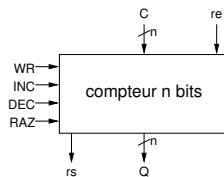
Un registre à décalage est un registre de taille fixe, fabriqué à l'aide de bascules flip-flops, dans lequel les bits sont décalés à fin de cycle de l'horloge  $H$ . On considère un registre de type SIPO (serial in, parallel out) : l'entrée est en série et la sortie est en parallèle.

Prenons l'exemple d'une suite de 4 bits, 1101. Le fait que l'entrée soit en série signifie qu'on utilise un seul fil pour insérer l'entrée dans le registre : on envoie 1, suivi de 0, puis de 1, et encore de 1 sur l'entrée  $E$ . La sortie en parallèle permet de récupérer plusieurs bits en même temps. Donc, pour cet exemple après 4 périodes de l'horloge du circuit on a 4 fils qui renvoient chacun un bit :  $Q_1 = 1, Q_2 = 1, Q_3 = 0, Q_4 = 1$ . Il peut être intéressant de dessiner un chronogramme pour illustrer cela.

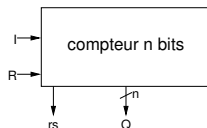
- 1) Proposez un circuit utilisant un registre à décalage pour détecter une suite de quatre 1 ou de quatre 0, comme dans la première partie de l'exercice. Vous vérifierez le bon fonctionnement de votre circuit à l'aide d'un chronogramme, en prenant pour séquence d'entrée : 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 (c'est le signal d'entrée  $w$  du chronogramme de la partie 1).

### Exercice 4 : Circuit compteur

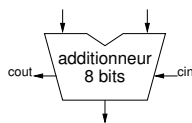
Un compteur se présente, vu de l'extérieur, comme un registre de mémorisation particulier auquel on adjoint, en plus de la fonction d'écriture, les fonctions d'incrément, de décrémentation et de mise à zéro. Ces fonctions sont mises en œuvre par les signaux correspondants : INC, DEC, RAZ, ECR. Lorsqu'aucun de ces signaux n'est activé, le compteur reste à sa valeur courante. Le code mémorisé par un compteur est celui d'un nombre entier naturel, et l'incrément le remplace par le nombre suivant, la décrémentation par le précédent.



Pour simplifier, on n'étudiera que l'incrémentation et la mise à zéro. On appelle Q le compte courant (code sortant) de taille 8 bits (compteur modulo  $2^8$ ), R le signal de mise à 0, et I le signal d'incrément. On suppose que R est prioritaire sur I. Un signal de sortie supplémentaire rs devra indiquer si Q est maximum ( $2^8 - 1$ ), et l'incrément dans ce cas remettra le compte à zéro (compteur modulo  $2^8$ ).



- 1) Interprétez en termes d'actions (incrémenter, mettre à zéro, laisser tel quel) les combinaisons de valeurs des signaux R et I. Caractériser chaque action par une expression logique en R et I, vraie seulement pour cette action. Formalisez chaque action par l'expression algorithmique de l'évolution de Q que cette action doit provoquer.
- 2) Le vecteur de 8 signaux Q code un entier naturel. Sa mémorisation nécessite un registre : sortie Q, entrée D, commande d'écriture W, horloge C. Exprimez W en fonction de R et I.
- 3) Supposons que R ou I est activé. Donnez une expression algorithmique de  $N(D)$  en fonction de Q et de R, comme une fonction de sélection.
- 4) En plus du registre 8 bits déjà décrit, on dispose d'un multiplexeur  $2 \times 8$  vers 8, dont le signal de sélection est s. On dispose également d'un additionneur 8 bits, avec retenue entrant cin (pour *carry in*) et retenue sortante cout (pour *carry out*).

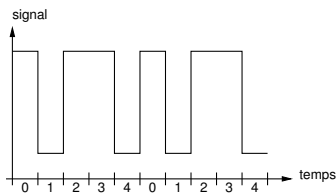


Dans le circuit séquentiel demandé, on suppose que l'additionneur opère de tout façon sur Q à chaque cycle d'horloge. Quelles entrées doivent être données à l'additionneur, et que doit-on faire de sa sortie. Précisez également comment vous comptez utiliser le multiplexeur; donnez le code de sélection que vous choisissez et exprimez s en fonction de R et I.

- 5) Proposez un circuit séquentiel implantant le compteur demandé, en laissant pour l'instant de côté le signal de sortie rs.
- 6) Complétez votre circuit en prenant en compte la sortie rs. Ajouter également une entrée re, telle que le fonctionnement du circuit reste inchangé, si ce n'est que le compteur est maintenant incrémenté uniquement si  $\bar{R} \cdot I \cdot re$ .
- 7) Proposez un chronogramme pour illustrer le fonctionnement du compteur de la question précédente. Vous supposerez que l'entrée re est maintenue à 1, que R est maintenue à 0, et que I est activé à chaque cycle. Au « cycle 0 » du chronogramme, vous supposerez que  $N(Q)=253$ , et vous complétez le chronogramme sur 7 cycles : représentez en particulier les valeurs de  $N(Q)$  et de rs.
- 8) Comment fabriquer un compteur 16 bits à partir de deux compteurs 8 bits? On appelle Q0 les 8 bits de poids faible du compteur, et Q1 les 8 bits de poids forts; on appelle ri la retenue qui se propage d'un compteur 8 bits à l'autre. Illustrer le fonctionnement du compteur 16 bits à l'aide d'un chronogramme et des valeurs initiales des compteurs bien choisies : représentez en particulier les valeurs de  $N(Q0)$ , ri,  $N(Q1)$  et  $N(Q)$ .

## Exercice 5 : Génération d'un signal périodique

Un circuit séquentiel est dépourvu d'entrée, et possède une sortie s sur laquelle il génère le signal périodique suivant : la valeur 1 est émise pendant une période de l'horloge du circuit, puis 0 est produite pendant la période suivante. Ensuite, le signal passe à 1 pendant deux périodes de l'horloge et à nouveau à 0 pendant la période suivante. Enfin, le signal est répété à partir de la première étape pour commencer un nouveau cycle.



- 1) Déterminez le nombre d'états nécessaires pour concevoir un automate réalisant le circuit séquentiel décrit. Donnez une correspondance entre les états de l'automate, et la sortie  $s$  du circuit. Finalement, établir la table de transition de l'automate.
- 2) Exprimez la fonction de transition sous forme de formules booléennes ( $d_2$ ,  $d_2$  et  $d_0$  en fonction de  $q_0$ ,  $q_1$  et  $q_2$ ), ainsi que la fonction de sortie ( $s$  en fonction de  $q_0$ ,  $q_1$  et  $q_2$ )
- 3) Proposez un circuit séquentiel implantant l'automate mis au point.
- 4) La période du signal généré devant être de 50 ns, quelle doit être la fréquence de l'horloge du circuit construit ci-dessus?

### Exercice 6 : Addition séquentielle

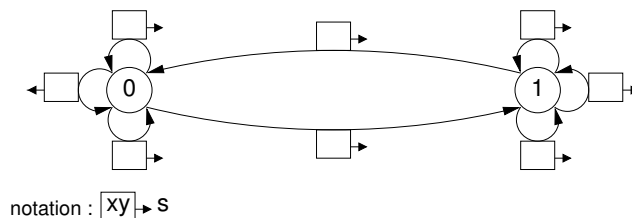
On veut mettre au point un circuit séquentiel pour effectuer une addition binaire en séquentiel; en  $n$  cycles d'horloge l'addition  $(s_{n-1} \dots s_1 s_0)_2 = (x_{n-1} \dots x_1 x_0)_2 + (y_{n-1} \dots y_1 y_0)_2$ . Au cycle  $i$  ( $n - 1 \geq i \geq 0$ ) :

- l'état courant du circuit est donné par la valeur de la retenue  $c_i$  (on suppose  $c_0 = 0$ );
- $s_i = x_i + y_i + c_i \pmod 2$ ;
- la nouvelle valeur de la retenue à propager  $c_{i+1}$  est telle que :  $x_i + y_i + c_i = 2 \times c_{i+1} + s_i$ .

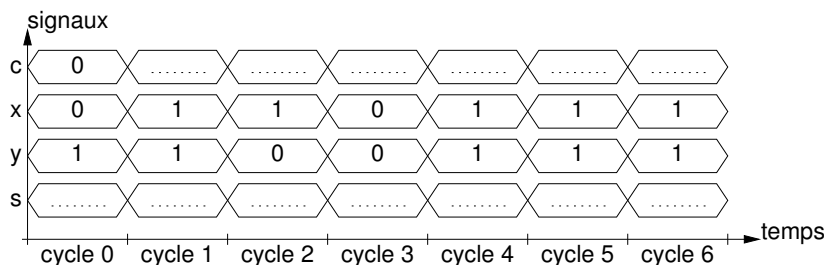
Dans la suite, on s'intéresse essentiellement à ce qui se passe à un cycle  $i$  quelconque, donc on oublie les indices :  $c$  désigne l'état courant du circuit,  $c^+$  l'état suivant,  $x$  et  $y$  sont les deux bits à additionner (entrées),  $s$  est le résultat de l'addition au rang  $i$  (sortie).

La fonction  $(x, y, c) \mapsto c^+$  est la fonction de transition du circuit, et la fonction  $(x, y, c) \mapsto s$  est la fonction de sortie.

- 1) Complétez la représentation graphique de l'automate fini séquentiel pour le circuit séquentiel demandé, en indiquant les valeurs prises par la fonction de sortie.



- 2) Complétez le chronogramme suivant.



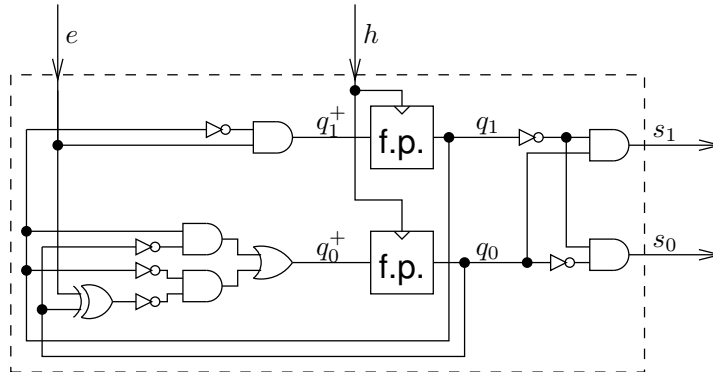
- 3) Complétez le tableau suivant :

$c$	$x$	$y$	$s$	$c^+$
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- 4) Montrez (en justifiant) que  $s = x \oplus y \oplus c$ .
- 5) En partant de la forme norme disjonctive pour  $s$ , montrez (en justifiant) que  $c^+ = (x \oplus y)c + xy$ .
- 6) Déduisez des questions précédentes un circuit implantant un additionneur séquentiel.

## Exercice 7 : Analyse d'un circuit séquentiel

On considère le circuit séquentiel ci-dessous. Ce circuit reçoit un signal d'horloge  $h$ , prend une entrée  $e$ , et produit une sortie  $s = (s_1 s_0)$ . Il comporte 2 bascules flip-flops, régies par le front montant de l'horloge  $h$ , qui stockent l'état courant  $(q_1 q_0)$  du circuit.

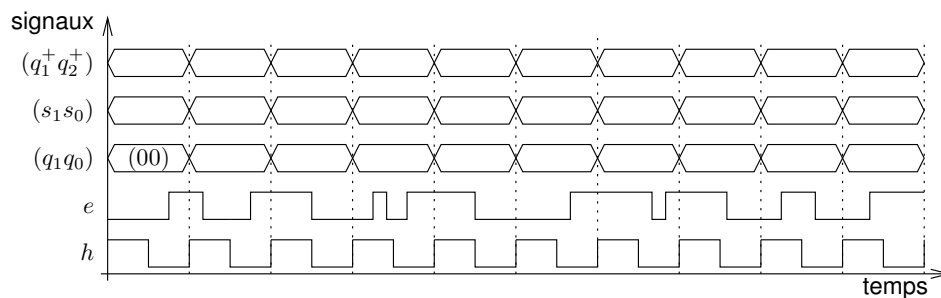


La fonction de transition du circuit est la fonction  $f$  telle que  $(q_1^+, q_0^+) = f(q_1, q_0, e)$ . La fonction de sortie  $g$  est telle que  $s = g(q_1, q_0)$ .

- 1) Exprimez  $q_1^+$  et  $q_0^+$  en fonction de  $q_1$ ,  $q_0$  et  $e$  par des expressions booléennes.
- 2) Exprimez  $s_1$  et  $s_2$  en fonction de  $q_1$  et  $q_0$  par des expressions booléennes.
- 3) Complétez la table de vérité suivante.

$q_1$	$q_0$	$e$	$q_1^+$	$q_0^+$	$s_1$	$s_0$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

- 4) Représentez le comportement du circuit sous la forme d'un automate fini séquentiel. Indiquez bien sur chaque flèche indiquant une transition la valeur de  $e$  provoquant cette transition, ainsi que la valeur de la sortie  $s$  pour chaque état.
- 5) Complétez le chronogramme suivant.



# TD 6

## Programmation en langage d'assemblage

Tous les exercices de ce TD sont à faire en utilisant le langage d'assemblage du LC3.

### Description du LC-3

**La mémoire et les registres :** La mémoire du LC-3 est organisée par mots de 16 bits, avec un adressage également de 16 bits (adresses de  $(0000)_H$  à  $(FFFF)_H$ ).

Le LC-3 comporte 8 registres généraux 16 bits : R0, ..., R7. R6 est réservé pour la gestion de la pile d'exécution, et R7 pour stocker l'adresse de retour des routines. Il comporte aussi des registres spécifiques 16 bits : PC (*Program Counter*), IR (*Instruction Register*), PSR (*Program Status Register*) qui regroupe plusieurs drapeaux.

Le PSR contient trois bits N, Z, P, indiquant si la dernière valeur (regardée comme le code d'un entier naturel en complément à 2 sur 16 bits) placée dans l'un des registres, R0, ..., R7 est négative strictement pour N, nulle pour Z, ou positive strictement pour P.

### Les instructions :

syntaxe	action	NZP
NOT DR,SR	DR <- not SR	*
ADD DR,SR1,SR2	DR <- SR1 + SR2	*
ADD DR,SR1,Imm5	DR <- SR1 + SEXT(Imm5)	*
AND DR,SR1,SR2	DR <- SR1 and SR2	*
AND DR,SR1,Imm5	DR <- SR1 and SEXT(Imm5)	*
LEA DR,label	DR <- PC + SEXT(PCOffset9)	*
LD DR,label	DR <- mem[PC + SEXT(PCOffset9)]	*
ST SR,label	mem[PC + SEXT(PCOffset9)] <- SR	
LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR	
BR[n][z][p] label	Si (cond) PC <- PC + SEXT(PCOffset9)	
NOP	No Operation	
RET	PC <- R7	
JSR label	R7 <- PC; PC <- PC + SEXT(PCOffset11)	

### Directives d'assemblage :

.ORIG adresse	Spécifie l'adresse à laquelle doit commencer le bloc d'instructions qui suit.
.END	Termine un bloc d'instructions.
.FILL valeur	Réserve un mot de 16 bits et le remplit avec la valeur constante donnée en paramètre.
.BLKW nombre	Cette directive réserve le nombre de mots de 16 bits passé en paramètre.
;	Les commentaires commencent par un point-virgule.

**Les interruptions prédéfinies :** TRAP permet de mettre en place des *appels système*, chacun identifié par une constante sur 8 bits, gérés par le système d'exploitation du LC-3. On peut les appeler à l'aide des macros indiquées ci-dessous.

instruction	macro	description
TRAP x00	HALT	termine un programme (rend la main à l'OS)
TRAP x20	GETC	lit au clavier un caractère ASCII et le place dans R0
TRAP x21	OUT	écrit à l'écran le caractère ASCII placé dans R0
TRAP x22	PUTS	écrit à l'écran la chaîne de caractères pointée par R0
TRAP x23	IN	lit au clavier un caractère ASCII, l'écrit à l'écran, et le place dans R0

**Constantes :** Les constantes entières écrites en hexadécimal sont précédées d'un x (en décimal elles peuvent être précédées d'un # optionnel); elles peuvent apparaître comme paramètre : des instructions du LC3 (opérandes immédiats, attention à la taille des paramètres), des directives .ORIG, .FILL et .BLKW.

## Codage des instructions LC3

On donne ici un tableau récapitulatif du codage des instructions LC3.

syntaxe	action	NZP	codage														
			opcode				arguments										
			F	E	D	C	B	A	9	8	7	6	5	4	3	2	1
NOT DR,SR	DR ← not SR	*	1	0	0	1	DR			SR			1 1 1 1 1 1				
ADD DR,SR1,SR2	DR ← SR1 + SR2	*	0	0	0	1	DR			SR1			0	0 0		SR2	
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*	0	0	0	1	DR			SR1			1	Imm5			
AND DR,SR1,SR2	DR ← SR1 and SR2	*	0	1	0	1	DR			SR1			0	0 0		SR2	
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*	0	1	0	1	DR			SR1			1	Imm5			
LEA DR,label	DR ← PC + SEXT(PCOffset9)	*	1	1	1	0	DR			PCOffset9							
LD DR,label	DR ← mem[PC + SEXT(PCOffset9)]	*	0	0	1	0	DR			PCOffset9							
ST SR,label	mem[PC + SEXT(PCOffset9)] ← SR		0	0	1	1	SR			PCOffset9							
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*	0	1	1	0	DR			BaseR			Offset6				
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR		0	1	1	1	SR			BaseR			Offset6				
BR[n z p] label	Si (cond) PC ← PC + SEXT(PCOffset9)		0 0 0 0				n	z	p	PCOffset9							
NOP	No Operation		0 0 0 0				0	0	0	0 0 0 0 0 0 0 0							
RET	PC ← R7		1 1 0 0				0 0 0		1 1 1			0 0 0 0 0 0					
JSR label	R7 ← PC; PC ← PC + SEXT(PCOffset11)		0 1 0 0				1	PCOffset11									

## Traduction de programmes en langage d'assemblage

Il vous est demandé de toujours commencer par écrire un pseudo-code pour le programme ou la routine demandé, en faisant apparaître les registres que vous allez utiliser pour effectuer vos calculs, et en ajoutant tous les commentaires utiles. Vous traduirez ensuite votre pseudo-code vers le langage d'assemblage du LC3 en utilisant les règles de traduction suivantes.

**Traduction d'un « bloc if » :** On suppose que la condition d'entrée dans le bloc consiste simplement en la comparaison du résultat d'une expression arithmétique  $e$  à 0. Dans ce qui suit, `cmp` désigne une relation de comparaison :  $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ,  $\geq$ ,  $>$ . On note `!cmp` la relation contraire de la relation `cmp`, traduite dans la syntaxe des bits `nzp` de l'instruction `BR`. Si par exemple `cmp` est  $<$ , alors `BR!cmp` désigne `BRpz` (pour « positive or zero »).

```

/* En pseudo-code */
if e cmp 0 {
    corps du bloc
}
; En langage d'assemblage du LC3
evaluation de e
BR!cmp endif ; branchement sur la sortie du bloc
corps du bloc
endif:

```

**Traduction d'un bloc « if-else » :**

```

/* En pseudo-code */
if e cmp 0 {
    corps du bloc 1
}
else {
    corps du bloc 2
}
; En langage d'assemblage du LC3
evaluation de e
BR!cmp else ; branchement sur le bloc else
corps du bloc 1
BR endif ; branchement sur la sortie du bloc
else:
    corps du bloc 2
endif:

```

**Traduction d'une « boucle while » :**

```

/* En pseudo-code */
while e cmp 0 {
    corps de boucle
}
; En langage d'assemblage du LC3
loop:
    evaluation de e
    BR!cmp endloop ; branchement sur la sortie de boucle
    corps de boucle
    BR loop ; branchement inconditionnel
endloop:

```

**Quelques « astuces » à connaître :**

- Initialisation d'un registre à 0 : `AND Ri,Ri,#0`
- Initialisation d'un registre à une constante  $n$  (représentable en complément à 2 sur 5 bits) :
 

```
AND Ri,Ri,#0
ADD Ri,Ri,n
```
- Calcul de l'opposé d'un entier (on calcule le complément à 2 de  $R_j$  dans  $R_i$ ) :
 

```
NOT Ri,Rj
ADD Ri,Ri,#1
```
- Multiplication par 2 de  $R_j$ , résultat dans  $R_i$  : `ADD Ri,Rj,Rj`
- Copie du contenu de  $R_j$  dans  $R_i$  : `ADD Ri,Rj,#0`

## Exercice 1 : Autour de la sommation

- 1) On considère le programme incomplet ci-dessous. On souhaite pouvoir ajouter les entiers aux adresses `add0`, `add1`, `add2`, et placer le résultat à l'adresse `resultat`. Donnez un code assembleur pour cela.

```
.ORIG x3000      ; adresse de début de programme
; partie dédiée au code
                ; R1 sera utilisé comme accumulateur
                ### A COMPLETER ###

; partie dédiée au résultat et aux données
resultat:      .BLKW #1      ; espace pour stocker le résultat (résultat attendu ici : 73 soit x49)
add0:          .FILL #12
add1:          .FILL #45
add2:          .FILL #16
.END
```

- 2) Supposons que R0 contient un entier positif  $x$ , et R1 un entier positif  $y$ . On souhaite affecter à R2 l'entier  $x - y$  : proposez un morceau de code pour cela.
- 3) On considère le programme incomplet ci-dessous. On souhaite pouvoir ajouter les entiers compris entre les adresses `debut` et `fin` à l'aide d'une boucle, et placer le résultat de la sommation à l'adresse `resultat`. Il vous est demandé de commencer par donner un pseudo-code pour votre programme, en respectant les directives d'utilisation des registres données ci-dessous, avant de le traduire en langage d'assemblage.

```
.ORIG x3000      ; adresse de début de programme
; partie dédiée au code
; R0 sera utilisé comme un pointeur pour une case du tableau
; R1 contiendra l'opposé de l'adresse fin, pour comparaison avec R0
; R2 servira à l'accumulation des entiers du tableau
; R3 servira de registre temporaire
                ### A COMPLETER ###
; partie dédiée au résultat et aux données
resultat:      .BLKW #1      ; espace pour stocker le résultat (résultat attendu ici : 157 soit x9D)
debut:         .FILL #12
                .FILL #45
                .FILL #16
                .FILL #06
fin:           .FILL #78
.END
```

## Exercice 2 : Multiplication par 6 des entiers d'un tableau

On considère le programme à compléter ci-dessous.

```
.ORIG x3000      ; adresse de début de programme
; partie dédiée au code
LD R6,spinit   ; initialisation du pointeur de pile
LEA R0,debut   ; charge l'adresse de début de tableau
LEA R1,fin     ; charge l'adresse de fin du tableau
JSR mul6tab    ; appel à une routine
HALT           ; termine le programme

; partie dédiée aux données
mask:         .FILL x000F    ; constante x000F
debut:        .FILL 4        ; adresse de début de tableau
                .FILL 5
                .FILL 6
fin:          .FILL 3        ; adresse de fin de tableau

; pile
spinit:       .FILL stackend
                .BLKW #5
stackend:     .BLKW #1      ; adresse du fond de la pile

; Routine mul5tab, pour multiplier les entiers d'un tableau par 6 modulo 16.
; Les multiplications se feront sur place.
; paramètres d'entrée : R0, adresse de début du tableau
;                       R1, adresse de fin du tableau
mul6tab:
: *** A COMPLETER ***
.END
```

Le but est de compléter la routine `mul6tab` pour qu'elle multiplie les entiers d'un tableau par 6 modulo 16 : en entrée `R0` contient l'adresse de la première case du tableau, `R1` l'adresse de la dernière case. Vous traduirez pour cela le pseudo-algorithme suivant :

```

; R2 <- R0
; while( R2 <= R1 ) { // (R2 <= R1) <=> (R2-R1 <= 0)
;   R3 <- mem[R2];
;   R3 <- 2*R3+4*R3; // R3 <- 6*R3
;   R3 <- R3 & 0x000F; // R3 <- R3 modulo 16
;   mem[R2] <- R3;
;   R2++;
; }
}

```

- 1) En utilisant uniquement `R4` comme registre intermédiaire, montrez comment traduire la ligne `R3 <- 2*R3+4*R3`.
- 2) A la ligne `R3 <- R3 & 0x000F`, le `&` désigne le AND bit-à-bit : justifier le fait que l'opération `R3 & 0x000F` calcule bien le reste dans la division euclidienne de `R3` par 16. Comment traduirez-vous cette ligne en langage d'assemblage?
- 3) Traduisez l'algorithme proposé. Vous pouvez utiliser `R4` et/ou `R5` pour les calculs intermédiaires.

### Exercice 3 : Programme Mystère LC3

On (source : <http://castle.eiu.edu/~mathcs/mat3670/index/index.html>, avec l'aimable autorisation des auteurs) fournit un programme LC3 sous forme d'un bout de mémoire dont on donne le contenu en hexadécimal. Il est demandé de décoder ce programme et de dire ce qu'il fait. On fera attention aux sauts de PC (le cas échéant, on rajoutera des *labels* à certaines adresses mémoires).

Adresse	Contenu	Contenu binaire	Détails des instructions	pseudo-code
x3000	x5020	0101 000 000 1 00000	AND (mode cst), DR=SR=R0, Imm5=x00	$R_0 \leftarrow R_0 \& 0 = 0$
x3001	x1221			
x3002	xE404			
x3003	x6681			
x3004	x1262			
x3005	x16FF			
x3006	x03FD			
x3007	xF025		HALT	HALT
x3008	x0006	donnée	-	

## Exercice 4 : Saisie d'une chaîne de caractères

Le système d'exploitation du LC3 fournit une interruption (pour simplifier, disons que c'est une routine) permettant d'afficher une chaîne de caractères (PUTS  $\equiv$  TRAP x22), mais on n'a pas la possibilité de saisir une chaîne de caractères. Le but est ici d'écrire une routine permettant cela. On considère le programme à compléter ci-dessous.

```
.ORIG x3000
; Programme principal
    LEA R6,stackend ; initialisation du pointeur de pile
    ; affiche la chaîne à l'adresse msg1
    LEA R0,msg1
    PUTS
    ; saisie d'une chaîne à l'adresse ch1
    LEA R1,ch1
    JSR saisie
    ; affiche la chaîne à l'adresse msg2
    LEA R0,msg2
    PUTS
    ; affiche la chaîne à l'adresse ch1
    LEA R0,ch1
    PUTS
    HALT

; partie dédiée aux données
msg1: .STRINGZ "Entrez une chaîne : "
msg2: .STRINGZ "Vous avez tapé : "
ch1:  .BLKW #8

; pile
stack: .BLKW #5
stackend: .FILL #0

; Sous-routine pour saisir une chaîne de caractères
; paramètre d'entrée : l'adresse R1 du début de chaîne
saisie:
    ; *** A COMPLETER ***
    .END
```

- 1) Dans une routine, avant d'appeler une autre routine, ou de déclencher une interruption vers le système d'exploitation, il est important de sauvegarder l'adresse de retour contenue dans R7 : pourquoi?
- 2) Complétez la routine `saisie` de manière à ce qu'elle permette de saisir une chaîne de caractères au clavier, en rangeant les caractères lus à partir de l'adresse contenue dans R1. La saisie se termine lorsqu'un retour chariot (code ASCII 13) est rencontré, et la chaîne de caractères doit être terminée par un caractère `'\0'` (de code ASCII 0). Vous utiliserez `GETC`, qui lit un caractère au clavier, et place son code dans R0.

## Exercice 5 : Décompte de bits non-nuls

Écrire une fonction dans l'assembleur du LC3 afin de compter le nombre de bits non-nuls dans un entier. Votre programme principal doit appeler votre fonction afin de compter le nombre de bits non-nuls de l'entier à l'adresse `n`, et placer le résultat à l'adresse `r`. Vous complétez le code ci-dessous, en le commentant.

```
.ORIG x3000
; Programme principal
    ...
; Données
n:      .FILL #78
r:      .BLKW #1

; Sous-routine pour calculer le nombre de bit non-nuls dans R1
; paramètre : R1, inchangé
; retour : R0, le nombre de bits non nuls dans R1
; registres temporaires : R2, R3, R4
cmtbits: ...
.END
```

## Exercice 6 : Débogage

On considère le programme ci-dessous, qui s'assemble sans message d'erreur, est s'exécute sur le LC3; par contre, l'exécution n'aboutit jamais au HALT : le programme boucle indéfiniment...

```
.ORIG x3000
; Programme principal
LD R0, n      ; R0 <- mem[n]
JSR fois20    ; appel à la routine fois20 : R1 <- 20 * R0
ST R1, r      ; mem[r] <- R1
HALT         ; rend la main à l'OS
n:           .FILL 7      ; Donnée, entier 7
r:           .BLKW 1      ; Résultat

; Routine pour effectuer R1 <- 20 * R0
; Attention, R2 est écrasé par l'appel.
fois20: JSR fois10      ; R1 <- 10 * R0
        ADD R1, R1, R1 ; R1 <- 2 * R1 = 20 * R0
        RET           ; retour à l'appelant

; Routine pour effectuer R1 <- 10 * R0
; Attention, R2 est écrasé par l'appel.
fois10: ADD R1, R0, R0 ; .....
        ADD R2, R1, R1 ; .....
        ADD R2, R2, R2 ; .....
        ADD R1, R1, R2 ; .....
        RET           ; retour à l'appelant
.END
```

- 1) Identifiez la raison pour laquelle le programme entre dans une boucle infinie : expliquez clairement les raisons du problème.
- 2) Quelle solution classique permet de remédier à ce problème?
- 3) fois10 effectue  $R1 \leftarrow 10 * R0$  : ajouter des commentaires dans le programme pour justifier cela.

## Exercice 7 : Nombre d'occurrences

Il s'agit de compléter la routine nboccs pour qu'elle compte le nombre d'occurrences d'un caractère dans une chaîne de caractères (par exemple, le nombre d'occurrences de 'o' dans "Toto fait du vélo" est 3). Rappel : le code du caractère de fin de chaîne, noté généralement '\0', est tout simplement l'entier 0. Il vous est demandé de commencer par donner un pseudo-code pour votre routine, en respectant les directives d'utilisation des registres données ci-dessous, avant de le traduire en langage d'assemblage.

```
.ORIG x3000
; Programme principal
LEA R0, ch
LD R1, car
JSR nboccs
HALT

; Une chaîne de caractères
ch: .STRINGZ "Toto fait du vélo"
car: .FILL x006F ; code ASCII du caractère 'o'

; Sous-routine pour compter le nombre d'occurrences
; dans une chaîne de caractères terminée par un '\0'.
; paramètres d'entrée : R0 contient l'adresse du début de la chaîne
; R1 contient le code ASCII d'un caractère
; paramètre de sortie : R2 le nombre d'occurrences du caractère
; registres temporaires : R3 contiendra l'opposé de R1
; R4 servira de pointeur pour parcourir la chaîne
; R5 recevra successivement chacun des caractères de la chaîne
nboccs:
### PARTIE A COMPLETER ###
.END
```