Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées

Architecture : Circuits numériques et éléments d'architecture

1ère année

Année scolaire 2014–2015

Consignes

Les exercices de ce recueil sont classés en 4 catégories :
- Les exercices avec la mention "*Préparation*" sont proposés pour vous aider à préparer les séances de TDs ; il est recommandé de résoudre ces exercices avant les séances de TD.
- Les exercices avec la mention "*Pour aller plus loin...*" sont proposés pour vous aider à consolider les notions vues pendant les séances de TD ; il est recommandé de résoudre ces exercices après la séance de TD.
- Les exercices avec la mention "*Méthodologie*" sont étudiés en séance. Leur correction est fournie dans l'enoncé.
- Les exercices sans mention sont ceux qui seront étudiés pendant les séances ; il est recommandé de préparer ces exercices avant la séance de TD.

Toutes les questions traitées en séance disposent d'une correction à la fin du fascicule. L'enseignant vous donnera le numéro correspondant à chaque question.

Grenoble INP ensimag

TD 1
Portes de base et minimisations de fonctions booléennes

Préparation
Ex. 1 : Codage des entiers naturels

Question 1 Remplir un tableau contenant les valeurs en base 2, en base 10 et en base 16 des entiers naturels entre 0 et 15 (inclus).

Question 2 Comment peut-on calculer facilement une valeur en hexadécimal à partir de sa valeur en binaire ? Convertir en hexadécimal la valeur binaire 101101011100_2.

Question 3 Faire les additions suivantes en binaire : $44+21$ et $200+100$ sur 8 bits [1]. Que constatez-vous pour le résultat de la 2e addition ? Comment peut-on détecter ce phénomène ?

Question 4 Quel intervalle d'entiers naturels peut-on coder sur n bits ? Combien de bits faut-il pour coder m valeurs différentes (avec $m \geq 2$) ?

Préparation
Ex. 2 : Circuit comparateur d'entiers

Dans cet exercice, on travaille sur des entiers A et B codés sur 4 bits, ce qu'on notera $A = a_3a_2a_1a_0$ et $B = b_3b_2b_1b_0$. Pour construire les circuits demandés, on suppose qu'on dispose de portes avec au maximum 4 entrées.

Question 1 Quelle porte élémentaire produit 1 en sortie ssi ses deux entrées sont égales ?

Question 2 Construire un circuit prenant en entrée 2 entiers A et B et produisant une sortie eg valant 1 ssi ces 2 entiers sont égaux.

Question 3 Etendre ce circuit pour ajouter une sortie z valant 1 ssi A vaut 0.

Question 4 Ajouter une sortie imp valant 1 ssi le nombre de bits composant B et valant 1 est impair. Par exemple, si $B = 5_{10} = 0101_2$ alors $imp = 0$ et si $B = 14_{10} = 1110_2$ alors $imp = 1$.

Méthodologie
Ex. 3 : Conception et minimisation de circuits combinatoires sur un exemple élémentaire

On travaille sur la fonction majorité de trois variables valant "vrai" ssi la majorité des paramètres de la fonction valent 1.

Table de vérité de la fonction Majorité

1. C'est à dire que les opérandes et le résultat de l'opération sont codés sur 8 bits.

a	b	c	$Maj(a,b,c)$	termes canoniques
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{a}.b.c$
1	0	0	0	
1	0	1	1	$a.\overline{b}.c$
1	1	0	1	$a.b.\overline{c}$
1	1	1	1	a.b.c

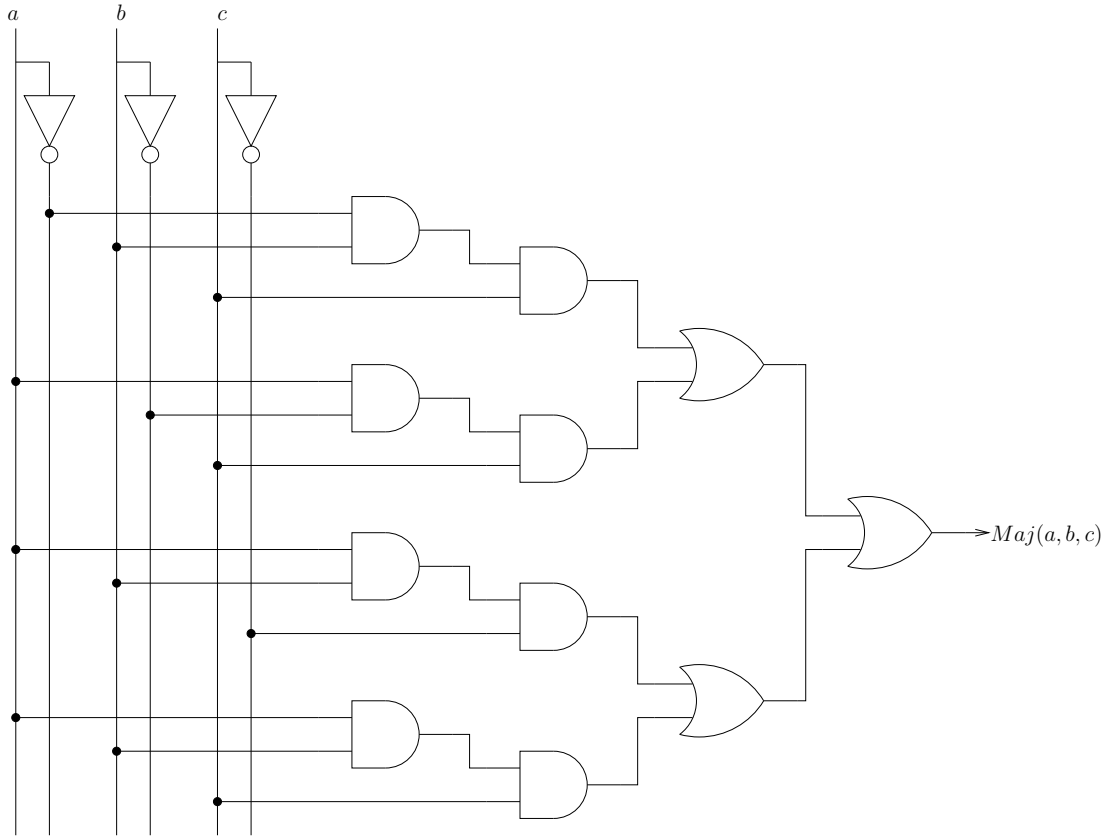
N.B : un "terme canonique" est le produit (AND) de toutes les variables de la fonction sous forme normale (a) ou complémentée ($\overline{a}$). On recherche une expression somme de produits donc on ne s'intéresse qu'aux termes correspondants aux 1 de la fonction.

Expression booléenne canonique de la fonction $Maj(a,b,c)$

C'est l'expression de la fonction $Maj(a,b,c)$ en fonction des paramètres a, b et c, sans chercher à minimiser cette expression. C'est donc la somme des termes canoniques déduits de la table de vérité : $Maj(a,b,c) = \overline{a}.b.c + a.\overline{b}.c + a.b.\overline{c} + a.b.c.$

Circuit implantant cette fonction

Pour dessiner le circuit implantant cette fonction, on n'utilisera ici que des portes de bases AND, OR à 2 entrées ainsi que des portes INV.



Minimisation de l'expression de la fonction

On montre ici comment on peut minimiser l'expression d'une fonction en utilisant le calcul booléen. On part de :

$$Maj(a,b,c) = \overline{a}.b.c + a.\overline{b}.c + a.b.\overline{c} + a.b.c$$

On remarque que les simplifications suivantes sont possibles, grâce à la règle $\overline{x}.y + x.y = y$:

$$\overline{a}.b.c + a.b.c = b.c$$

$$a.\overline{b}.c + a.b.c = a.c$$

$$a.b.\overline{c} + a.b.c = a.b$$

Faut-il choisir une (et une seule) de ces simplifications ? non, car on a aussi la règle $x + x = x$. On peut donc réécrire l'expression de la fonction Majorité comme ci-dessous :

$$\begin{aligned} Maj(a,b,c) &= \overline{a}.b.c + a.\overline{b}.c + a.b.\overline{c} + a.b.c + a.b.c + a.b.c \\ &= \overline{a}.b.c + a.b.c + a.\overline{b}.c + a.b.c + a.b.\overline{c} + a.b.c \\ &= b.c + a.c + a.b \end{aligned}$$

Tableaux de Karnaugh :
Une technique graphique pour la minimisation d'expressions booléennes

Un tableau de Karnaugh est une table de vérité, présentée de façon à ce que les adjacences du type $\overline{x}.y + x.y = y$ soient mises en évidence.

Le tableau de Karnaugh de la fonction $Maj(a,b,c)$ est donné ci-dessous. Remarquez que l'ordre de l'énumération des variables b et c n'est pas quelconque : une seule des variables change de valeur entre deux colonnes (y compris quand on rapproche la dernière colonne de la première : adjacence circulaire).

Remplir un tableau de Karnaugh revient à remplir une table de vérité. Par exemple, la troisième case de la première ligne correspond à $a = 0$ et $b = 1$ et $c = 1$: $Maj(0,1,1) = 1$ donc la case est remplie avec un 1.

a \ bc	**00**	**01**	**11**	**10**
0	0	0	1	0
1	0	1	1	1

Les groupements de points à 1 mettent en évidence les simplifications possibles. Il suffit alors de choisir un nombre minimal de regroupements qui couvrent tous les points à 1 de la fonction.

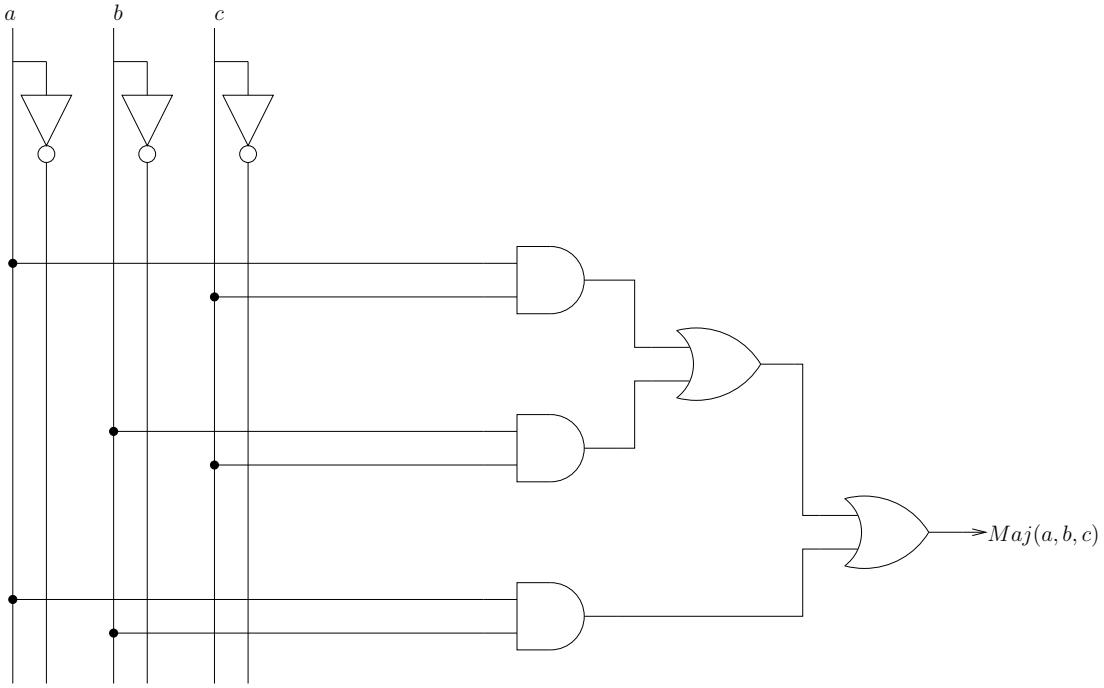
Pour chaque groupe de cases à 1, on extrait un terme construit à partir des variables qui ne changent pas de valeur pour l'ensemble des points regroupés. Par exemple, pour le groupement vertical de la troisième colonne : on remarque que la valeur de a change entre les deux cases regroupées, a n'apparaitra donc pas dans le terme correspondant à ce groupe. Ce qui est commun dans ce groupe ce sont les valeurs de b et c : on extrait donc le terme $b.c$

Sur cet exemple, il y a donc 3 termes qui correspondent aux trois groupements, on retrouve l'expression : $Maj(a,b,c) = b.c + a.c + a.b$.

N.B : Les groupements ne peuvent être que de 2, 4 ou 8 cases (puissances de 2). Par la suite, on utilisera souvent des tableaux avec 4 variables en entrées donc 16 cases. Le nombre de variables dans un terme correspond à la taille du groupement. Par exemple, dans un tableau de 16 cases (4 variables), un groupe de 8 cases correspond à un terme d'une variable, un groupe de 4 à un terme de 2 variables, un groupe de 2 à un terme de 3 variables et si il reste un 1 isolé le terme correspondant utilise les 4 variables.

Circuit minimisé

A partir de l'expression minimisée, on peut dessiner le circuit suivant :



Pourquoi a t on besoin de minimiser ?

On remarque que le circuit obtenu à partir de la forme minimisée a deux avantages : moins de portes au total (5 au lieu de 12) et moins de portes traversées (entre l'entrée et la sortie, au plus 3 portes au lieu de 4). Ce circuit minimisé est donc moins couteux en porte et plus rapide.

N.B : Chaque porte a un temps de propagation (différent suivant la technologie). Le temps de propagation à travers un circuit est déterminé par le nombre maximal de couches de portes à traverser entre une entrée et une sortie.

Remarque : dans la suite du semestre, nous n'utiliserons que les minimisations de fonctions booléennes à base de tableaux de Karnaugh.

Ex. 4 : Reste de la division entière

Soit un entier naturel E dans l'intervalle [0 .. 15], donc codé sur 4 bits e_3, e_2, e_1, e_0 ; on veut réaliser un circuit qui calcule le reste S de la division entière de E par 7.

Question 1 Sur combien de bits doit être codé S ?

Question 2 Représenter les fonctions de sortie du circuit à l'aide de tableaux de Karnaugh.

Question 3 Proposer des expressions minimisées des fonctions de sortie du circuit.

On considère maintenant que l'entier E en entrée est dans l'intervalle [0 .. 9]. Les fonctions en sortie sont donc des fonctions Φ - booléennes.

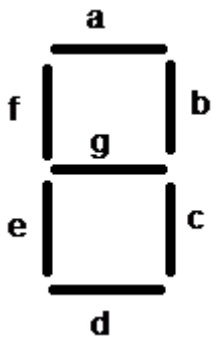
Méthodologie : Pour toutes les cases qui correspondent à une valeur d'entrées non utilisée (ici, les cases correspondant aux valeurs de 10 à 15), la valeur de la sortie est indifférente. On note Φ dans les cases correspondantes : Φ veut dire 0 ou 1, au choix du concepteur. On pourra donc utiliser les Φ pour simplifier davantage l'expression de la fonction (s'ils permettent d'obtenir de plus gros groupements de cases).

Question 4 Modifier les tableaux de Karnaugh obtenus en question 2 et proposer des expressions minimisées des fonctions de sortie du circuit.

Pour aller plus loin...
Question 5 Dessiner le circuit à base de portes AND, OR et INV à partir des équations de la question 3. Peut-on diminuer la complexité de ce circuit en utilisant un même monôme pour plusieurs sorties ?

Ex. 5 : Afficheur 7-segments

Un afficheur 7-segments est un dispositif d'affichage d'un chiffre décimal composé, comme son nom l'indique, de sept segments pouvant être allumés ou éteints indépendamment les uns des autres. Les segments sont disposés comme illustré ci-dessous :



On considère que l'affichage est réalisé comme suit :

Le but de cet exercice est de concevoir un circuit prenant en entrée un chiffre décimal et produisant en sortie les sept sorties booléennes correspondant aux segments de l'afficheur.

Question 1 Sur combien de bits doit-être codée l'entrée ? Quelle politique adopter concernant les valeurs superflues ?

Question 2 En utilisant des tableaux de Karnaugh, donner les expressions minimisées des 7 segments.

Pour aller plus loin...
Question 3 Dessiner le circuit correspondant en utilisant des portes logiques de base.

Pour aller plus loin...
Ex. 6 : Transcodeur en code de Gray

On appelle code de Gray [2] (ou codage binaire réfléchi) un codage binaire ordonné dans lequel le codage ne change que d'un bit entre deux valeurs successives. Le code de Gray sur 2 ou 3 bits est utilisé dans les tableaux de Karnaugh. Soit donc le code de Gray sur 4 bits suivant :

2. Frank Gray, Bell Labs, 1953

Décimal	Gray
0	0000
1	0001
2	0011
3	0010

Décimal	Gray
4	0110
5	0111
6	0101
7	0100

Décimal	Gray
8	1100
9	1101
10	1111
11	1110

Décimal	Gray
12	1010
13	1011
14	1001
15	1000

On veut réaliser un circuit prenant en entrée un entier naturel codé sur 4 bits e_3, e_2, e_1, e_0 et produisant en sortie le code de Gray correspondant s_3, s_2, s_1, s_0.

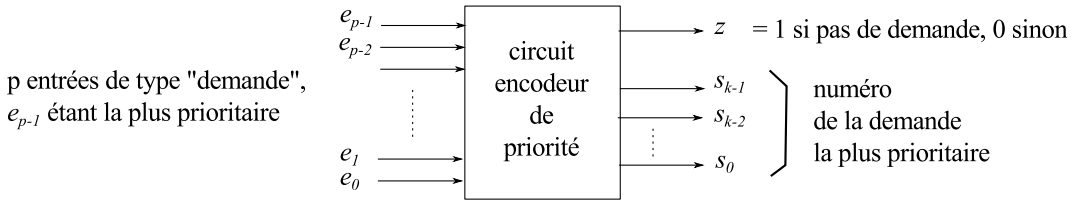
Question 1 Déterminer les équations minimisées des sorties à l'aide de tableaux de Karnaugh.

Question 2 Dessiner le circuit correspondant en utilisant des portes logiques de base.

Question 3 Pour apprendre à repérer les expressions à base de XOR, faire le tableau de Karnaugh de $e_3 \oplus e_2 \oplus e_1 \oplus e_0$; de $e_3.(e_2 \oplus e_1 \oplus e_0)$.

Pour aller plus loin...

Ex. 7 : Encodeur de priorité



Un encodeur de priorité est un circuit qui donne en sortie le numéro (codé en binaire) de l'entrée active la plus prioritaire. Par exemple, si seules les entrées e_5 et e_8 sont à 1, la sortie (bits s_{k-1} à s_0) codera 8 en binaire.

Question 1 Si $p = 2^n$, déterminer le nombre de bits en sortie (la valeur de k). Donner une description formelle du circuit.

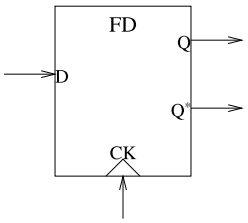
Question 2 Dans le cas $n = 2$, déterminer les équations minimisées de z et $S = s_1 s_0$ en remplissant une table de vérité partielle (*i.e.* réfléchir sur les lignes pouvant être factorisées dans la table).

Question 3 Dessiner le circuit correspondant en utilisant des portes logiques de base.

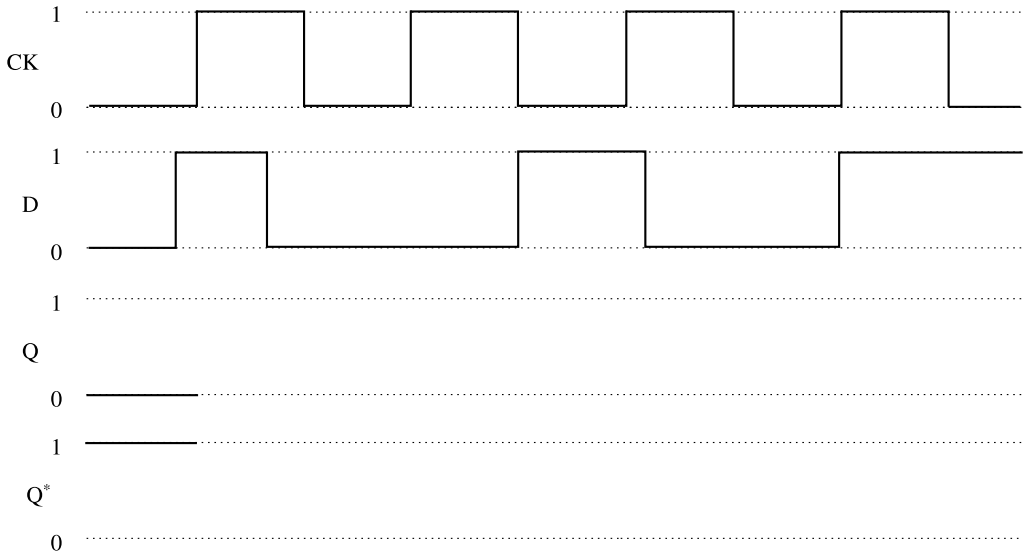
TD 2
Bascules et registres

Ex. 1 : Bascule D

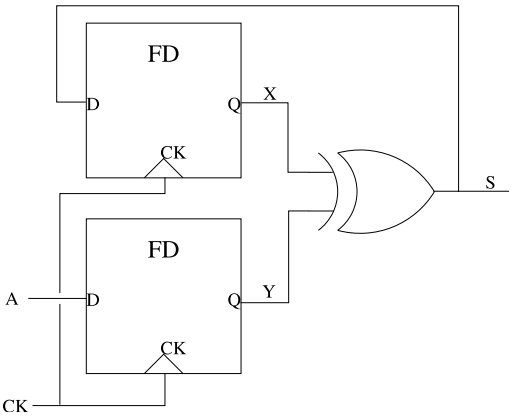
La bascule D (*D flip-flop*) illustrée ci-dessous est un dispositif qui permet d'échantillonner l'entrée D sur un front d'une entrée de commande usuellement notée CK, pour *clock* ou horloge. On suppose dans ce qui suit que l'échantillonnage a lieu sur le front montant (*rising edge*) du signal CK.



Question 1 Complétez le chronogramme ci-dessous en l'appliquant à une bascule D.

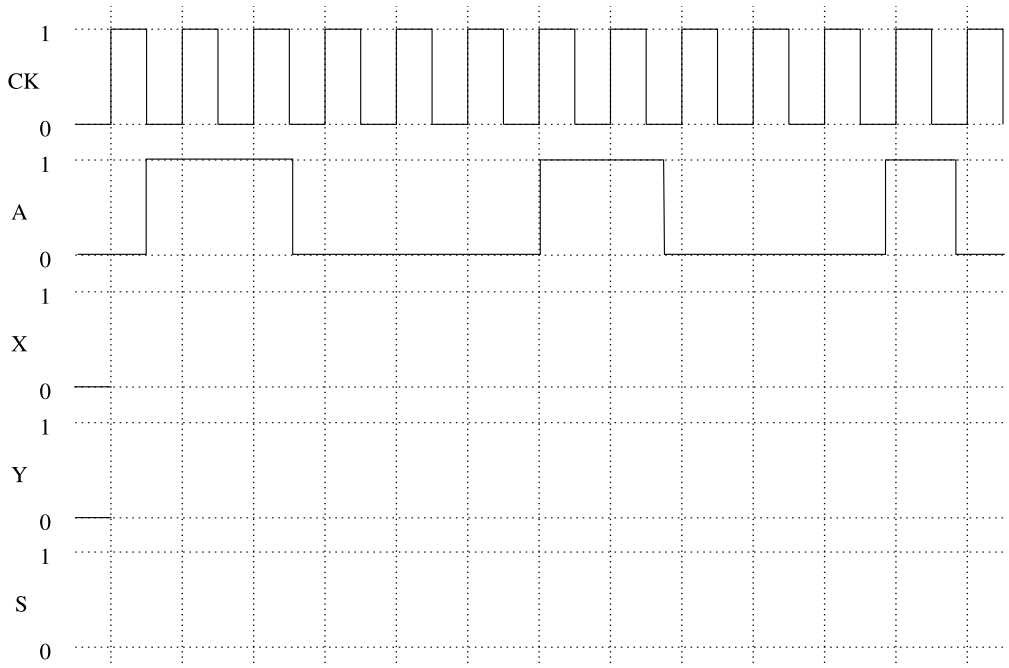


On se propose d'étudier le comportement d'un circuit suivant :



On suppose que l'on force les entrées du circuit comme indiqué sur le chronogramme suivant :

Question 2 Complétez le chronogramme suivant à partir des valeurs fournies sur les entrées du circuit et dans les bascules.



On appelle fréquence maximale de fonctionnement d'un circuit, la fréquence d'horloge au-delà de laquelle il existe un chemin entre deux registres ayant un temps de propagation supérieur à la période de l'horloge. Ce chemin est généralement appelé "chemin critique".

Supposons que :

- le temps de traversée d'une porte XOR2 soit de 0.1ns
- le temps de traversée d'une bascule D soit de 0.1 ns
- le temps de maintien d'une bascule D soit de 0.01 ns
- le temps de prépositionnement d'une bascule D soit de 0.05 ns

Question 3 Donner le temps de traversé du chemin critique du circuit étudié. En déduire la fréquence maximum de fonctionnement.

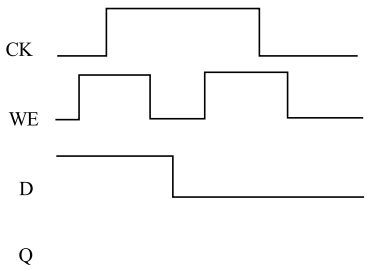
Ex. 2 : Construction d'une bascule à écriture conditionnelle

On cherche à présent à réaliser, à partir d'une bascule D et de portes combinatoires simples, une bascule à écriture conditionnelle : une bascule dont l'écriture n'a pas lieu systématiquement à chaque front d'horloge, mais uniquement lorsqu'un autre signal (usuellement nommé WE pour *write enable*) est à 1 au moment du front d'horloge.

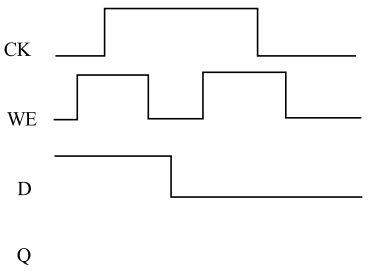
Question 1 Quels sont les signaux sur lesquels on peut agir pour implémenter cette écriture conditionnelle ?

Question 2 Proposez 2 schémas, l'un utilisant une porte AND et l'autre utilisant un multiplexeur, permettant de contrôler le changement de valeur dans la bascule.

Question 3 Compléter les chronogrammes ci-dessous en indiquant l'évolution de la sortie Q pour chacune des solutions . En déduire les avantages et inconvénients de ces 2 approches vis-à-vis des contraintes temporelles sur le signal d'autorisation d'écriture.



Solution avec porte and



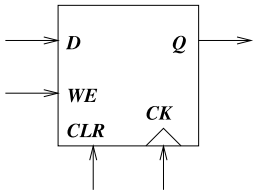
Solution avec multiplexeur

Ex. 3 : Construction de registres

Par définition, un registre est un composant regroupant plusieurs bascules D. Le registre permet donc de mémoriser plusieurs bits dans un même composant.

Pour construire des registres, on va utiliser des bascules D avec écriture conditionnelle dont la table de transition est donnée ci-dessous :

CLR	WE	CK	Q
1	–	–	0
0	0	–	Q_{prec}
0	1	↑	D



L'entrée CLR est une entrée de mise à 0 (*clear*) à effet asynchrone (cette entrée force la mise à 0 indépendamment du front d'horloge). L'entrée WE permet d'ignorer les fronts montants de l'horloge (lorsque $WE = 0$) et donc de mémoriser la même valeur sur plusieurs cycles sans avoir à la recharger.

Question 1 On appelle registre à chargement parallèle un registre pour lequel toutes les bascules sont activées en écriture sur le même front d'horloge. Construire un registre 4 bits à chargement parallèle à partir de ces bascules D, le signal WE autorisant le chargement du registre.

Question 2 Construire un registre 4 bits à chargement parallèle à décalage à gauche à partir de ces bascules D et de multiplexeur 2 vers 1. Le registre doit être conforme à la table de transition suivante :

CLR	WE	SHL	CK	Q
1	–	–	–	0
0	0	–	–	Q_{prec}
0	1	0	↑	D
0	1	1	↑	$Q_{prec} << 1$

Où $A << P$ représente le décalage du nombre A de P bits vers la gauche, avec insertion de 0 à la place des P bits de poids faible. On rappelle que décaler un entier de P bits vers la gauche revient à le multiplier par 2^P.

Question 3 Construire un registre 4 bits à chargement parallèle à décalage à droite à partir de ces bascules D et de $MUX_{1b}(2 \to 1)$. Le registre doit être conforme à la table de vérité suivante :

CLR	WE	SHR	ARI	CK	Q
1	–	–	–	–	0
0	0	–	–	–	Q_{prec}
0	1	0	–	↑	D
0	1	1	0	↑	$Q_{prec} >>> 1$
0	1	1	1	↑	$Q_{prec} >> 1$

Où

- $A >>> P$ représente le décalage du nombre A de P bits vers la droite, avec insertion de 0 à la place des P bits de poids forts : c'est le décalage « logique » à droite.
- $A >> P$ représente le décalage du nombre A de P bits vers la droite, avec insertion du bit de signe à la place des P bits de poids forts : c'est le décalage « arithmétique » à droite.

De façon symétrique au décalage à gauche, décaler un entier de P bits vers la droite revient à le diviser par 2^P : le décalage logique a donc un sens mathématique pour les entiers naturels, et le décalage arithmétique pour les entiers relatifs.

TD 3
Mémoires et macroblocs

Ex. 1 : Etude d'une mémoire vidéo

Soit la mémoire suivante utilisée dans une carte vidéo pour téléphone portable. La définition est de 320×240, 16 bits par pixels.

Il s'agit d'une mémoire asynchrone, on utilisera les caractéristiques temporelles du boîtier mémoire vu en cours :

- *Read Cycle Time* $t_{RC} = 10\ ns$;
- *Output Hold Time* $t_{OHA} = 2\ ns$;
- *Address Access Time* $t_{AA} = 3\ ns$.

Question 1 Combien de mots de 16 bits comporte cette mémoire ? Sur combien de bits doit-on coder les adresses ?

Question 2 Proposer une structure simple de cette mémoire.

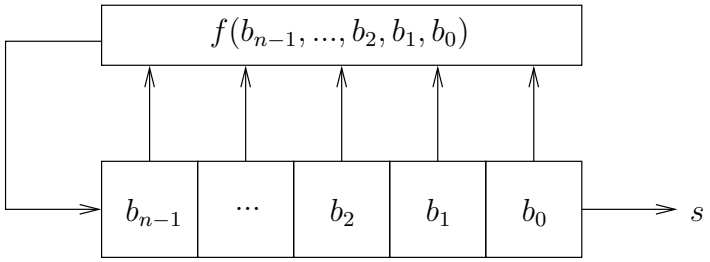
Question 3 Les pixels de coordonnées $(100, 0)$ et $(101, 0)$ affichent du bleu et du violet. On admettra que cela signifie que la mémoire contient les valeurs $0x003F$ et $0xF83F$ aux adresses $0x64$ et $0x65$. Dessiner le chronogramme représentant la lecture des mots mémoires correspondant.

Question 4 Quelle est la fréquence limite de fonctionnement de la carte graphique (*i.e.* le nombre de lectures en mémoire vidéo par seconde) ?

Question 5 Supposons maintenant que la mémoire est synchrone et répond en un cycle. Redessiner le chronogramme de la question 3 en conséquence.

Ex. 2 : Générateur de nombres aléatoires

On travaille dans cet exercice sur un composant appelé registre à décalage à rétroaction linéaire (*Linear Feedback Shift Register*) servant à générer des séquences « pseudo-aléatoires » de bits. Le principe de ce composant est détaillé dans la figure ci-dessous.



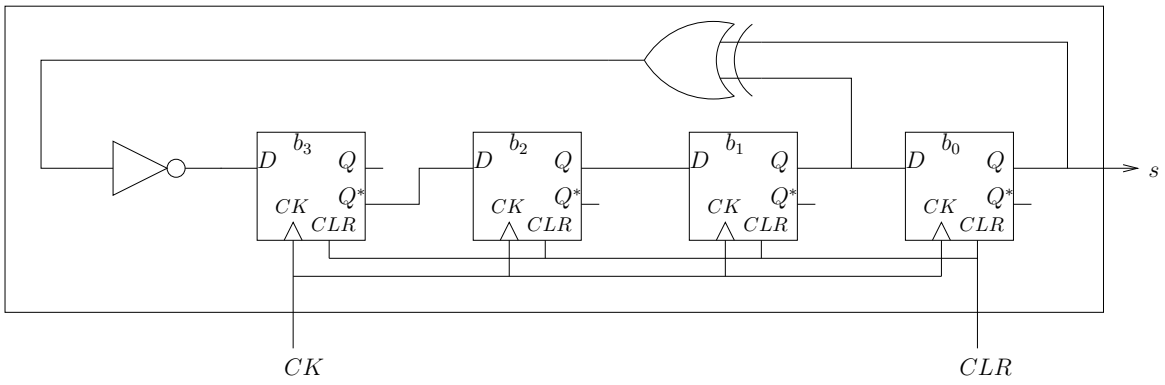
Chaque case correspond à une bascule D : il y a donc n bascules numérotées de 0 à $n-1$, n étant le nombre de bits du LFSR. La sortie s est le bit pseudo-aléatoire généré par le LFSR : elle affiche la valeur de la bascule numéro 0. A chaque cycle, on charge le contenu de la bascule i dans la bascule $i-1$: si on considère la séquence des valeurs contenues dans les bascules comme une valeur $B = b_{n-1}...b_1b_0$, alors B est décalée d'un bit vers la droite à chaque cycle. La valeur insérée dans la bascule $n-1$ est calculée en fonction des valeurs de chaque bascule via une fonction de rétroaction f définie par $f(b_{n-1}, ..., b_2, b_1, b_0) = c_{n-1}.b_{n-1} \oplus ... \oplus c_1.b_1 \oplus c_0.b_0$ où le c_i sont des constantes binaires données.

Dans la suite de l'exercice, on pose $n = 4$ et $f(b_3, b_2, b_1, b_0) = 0.b_3 \oplus 0.b_2 \oplus 1.b_1 \oplus 1.b_0 = b_1 \oplus b_0$.

Question 1 On suppose que les bascules sont initialisées avec les valeurs suivantes : $b_3 = 1$ et $b_2 = b_1 = b_0 = 0$. Remplir un tableau contenant les valeurs des bascules en fonction du temps jusqu'à ce qu'on retombe sur la valeur initiale.

Question 2 Exprimer le nombre de valeurs différentes possible en fonction de n. Que se passe-t'il si on se retrouve avec $b_3 = b_2 = b_1 = b_0 = 0$?

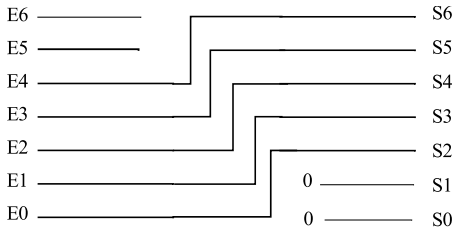
Question 3 Une réalisation d'un LFSR 4 bits (avec la fonction de rétroaction : $f(b_3, b_2, b_1, b_0) = b_1 \oplus b_0$) à l'aide de 4 bascules D est proposée ci-dessous.



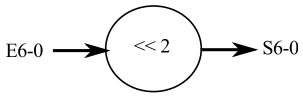
Justifiez l'utilisation de l'inverseur et de la connexion à $\overline{Q}$ sur la bascule $b3$.

Ex. 3 : Décaleur à barillet

Un décalage d'un nombre constant de bits ne nécessite aucune porte logique : il s'agit de connecter les fils de façon adéquate (voir schéma ci-dessous).



Décalage à gauche de 2 positions



Symbole

Un décaleur à barillet (ou *barrel shifter*) est un opérateur combinatoire qui permet de décaler un nombre x, codé sur n bits, d'un certain nombre de bits $p < n$ vers la gauche ou la droite. On note typiquement :

- $x << p$ le décalage de x de p bits vers la gauche en remplissant les cases libérées à droite par des 0 ;
- $x >> p$ le décalage de x de p bits vers la droite en remplissant les cases libérées à gauche par le bit de signe du nombre x initial (on parle de décalage arithmétique) ;
- $x >>> p$ le décalage de x de p bits vers la droite en remplissant les cases libérées à gauche par des 0 (on parle de décalage logique).

Question 1 Sur combien de bits doit être codé le décalage p si on suppose que x est codé sur 32 bits ? Déduisez-en la relation générale entre le nombre de bits de x (n) et de p (m). Mathématiquement

parlant, à quelles opérations élémentaires correspondent les différents décalages ?

Question 2 Proposez l'implantation d'un décaleur à gauche d'une entrée x pour $n = 2$. Quelle porte élémentaire est idéale pour cette implantation ?

Question 3 En vous basant sur le résultat précédent, proposer une implantation pour $n = 4$ et $n = 8$. Evaluer le circuit : nombre de couches logiques à traverser et complexité en surface, l'unité étant le mux 1 bit $2 \rightarrow 1$.

Pour aller plus loin...
Question 4 On veut maintenant gérer le décalage logique vers la droite. On ajoute pour cela une entrée d telle que $d = 1$ si on effectue un décalage à droite. Modifiez le décaleur 4 bits pour gérer le décalage logique à droite en plus du décalage à gauche. Attention, il y a deux solutions :

- la solution triviale est de choisir à chaque étage un bit de l'étage précédent en fonction du sens du décalage à effectuer,
- la solution avec "miroirs" : on utilise le décaleur à gauche vu à la question 3. Si on doit décaler à droite, on permute les bits de l'entrée ("miroir" : on met en entrée du décaleur les bits 0 à 3 au lieu des bits 3 à 0), puis on effectue sur cette nouvelle valeur un décalage à gauche ; les bits du résultat obtenu sont permutés à nouveau en sortie.

Evaluer les deux solutions.

Pour aller plus loin...
Question 5 On veut enfin ajouter la gestion du décalage arithmétique à droite. On dispose d'une nouvelle entrée a valant 1 si on veut effectuer un décalage arithmétique lors d'un décalage à droite. (Notez que le décalage à gauche est toujours arithmétiquement valable, indépendamment de la valeur de a). Modifier le schéma du décaleur gauche/droite 4 bits pour y ajouter la gestion du bit de signe dans les décalages à droite.

TD 4
Opérateurs Arithmétiques et Logiques

Préparation

Ex. 1 : Codage des nombres en complément à deux

Le codage en complément à deux est un codage permettant de représenter des entiers relatifs en binaire. Il existe d'autres codages moins utilisés (*e.g.* signe et valeur absolue, complément à un) que l'on ne détaillera pas ici.

Question 1 Remplir un tableau contenant les valeurs en base 2 et en base 16 des entiers relatifs entre -8 et 7 (inclus), codés en complément à 2^4 . Comment peut-on calculer $-X$ à partir de X en base 2 ?

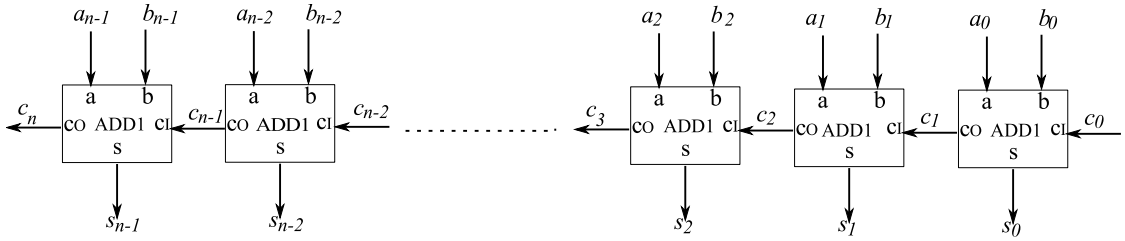
Question 2 Quel intervalle d'entiers relatifs peut-on coder sur n bits ?

Question 3 Effectuer les opérations $2+3$, $-3+3$,$-2+-5$, $7+1$, $-7+-2$ sur 4 bits.

Question 4 Comment peut-on détecter un dépassement de capacité lors d'une opération sur des nombres codés en complément à deux ?

Ex. 2 : Additionneur binaire

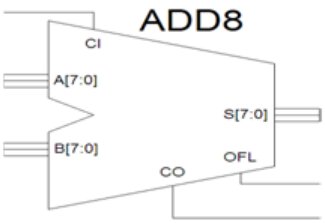
On utilise l'architecture dite de « l'additionneur à propagation de retenue ». Le principe est de construire une cellule élémentaire d'addition 1 bit ADD1 capable d'effectuer une addition entre deux bits a et b, en prenant en compte une retenue entrante cI (carry in), et qui produit en sortie la somme s et une retenue sortante cO (carry out). Une fois cette cellule disponible, il est facile de construire un additionneur n bits en reliant la retenue sortante de la cellule de rang i à la retenue entrante de la cellule de rang $i+1$.



Question 1 Donner les expressions simplifiées de la somme s et de la retenue sortante cO d'une cellule élémentaire ADD1 en fonction des opérandes a et b et de la retenue entrante cI. Dessiner le schéma correspondant. Que doit valoir la retenue entrante c_0 de l'additionneur n bits ? Evaluer le temps de calcul pour une addition n bits.

Question 2 Un additionneur dispose également de deux sorties particulières, sur 1 bit chacune (appelées « indicateurs » (*flags*)) et définies par :

- CO (*Carry Out*) est la retenue sortante c_n générée par l'addition, qui vaut 1 ssi une addition portant sur des entiers naturels a généré un dépassement de capacité ;
- OFL (*Overflow*) vaut 1 ssi une addition portant sur des entiers codés en complément à deux a généré un dépassement de capacité.



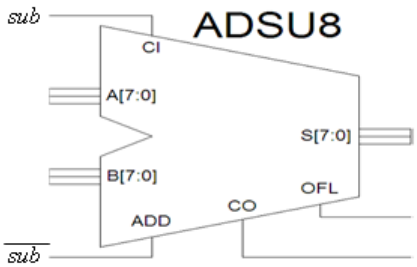
Exprimer le bit OFL en fonction des c_i.

Ex. 3 : Additionneur /soustracteur

On veut maintenant implanter l'opération de soustraction en utilisant l'additionneur de l'exercice 2.

Question 1 En se basant sur la relation $-X = \overline{X}plus1$, expliquer comment on peut simplement modifier le circuit d'une cellule élémentaire d'additionneur pour gérer aussi la soustraction de deux entiers relatifs codés en complément à deux, suivant la valeur d'une entrée *sub* qui vaut 0 pour une addition, 1 pour une soustraction. Dessiner le schéma correspondant à une cellule de rang i. Que faut-il faire pour la cellule de rang 0 ?

Question 2 La sortie CO de l'additionneur/soustracteur est égale à la retenue c_n. Dans le cas d'une soustraction d'entiers naturels, que veut dire la valeur de CO ? Montrez que la variable $C = CO \oplus sub$ vaut 1 ssi une addition ou une soustraction portant sur des entiers naturels génère un résultat qui n'est pas un entier naturel codable sur n bits. On peut aussi dire que CO est actif à 1 pour l'addition, à 0 pour la soustraction des entiers naturels.



Note : l'indicateur OFL est toujours valide pour les entiers codés en complément en 2, que l'opération soit une addition ou une soustraction.

Pour aller plus loin...
Question 3 On cherche à effectuer une opération sur 2n bits par composition d'opérations sur n bits, en utilisant 2 additionneurs/soustracteurs n bits. Comment connecter les deux circuits ?

Pour aller plus loin...
Ex. 4 : Unité Arithmétique et Logique

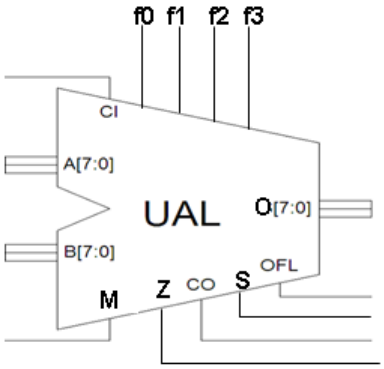
Dans cet exercice, on va étudier l'unité arithmétique et logique, et construire la partie centrale de ce circuit (l'étude complète d'un tel circuit dépasse le cadre du TD).

Une Unité Arithmétique et Logique (UAL ou ALU pour Arithmetic and Logic Unit) est un circuit combinatoire regroupant différentes opérations arithmétiques (addition, soustraction, etc.) et logiques (AND, OR, XOR, etc.) applicables à des valeurs codées sur un nombre n de bits (on parle alors d'« UAL n bits »).

Sur le schéma ci-dessous, le bit M permet de choisir le mode (M=0 : mode logique, M = 1 : mode arithmétique), les bits f_i représentent les entrées de fonction, qui permet de choisir l'opération que doit effectuer l'UAL. Les entrées A et B servent à coder les opérandes et O le résultat de l'opération (sur 8 bits dans l'exemple du schéma).

Outre CO et OFL, l'UAL dispose également de deux indicateurs, définis par :

– Z (*Zero*) vaut 1 ssi le résultat d'une opération est nul ;
– S (*Sign*) vaut 1 ssi le résultat d'une opération, interprété comme un entier relatif codé en complément à deux, est négatif.



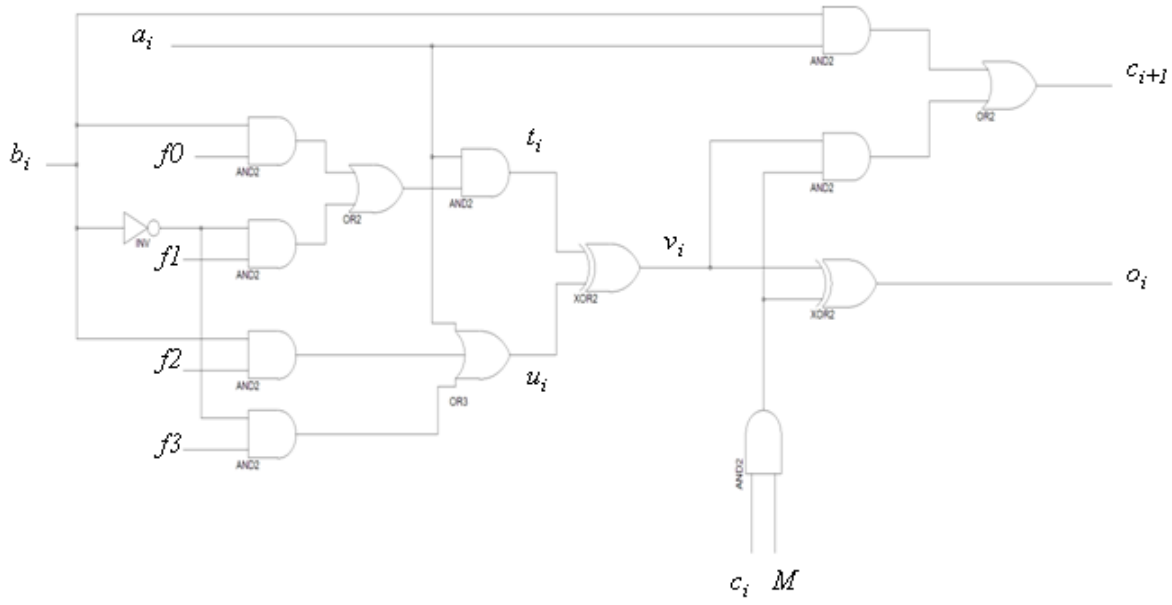
Les opérations que doit réaliser l'UAL que nous concevons sont indiquées dans le tableau ci-dessous.

Opération	Mode M	f0 f1 f2 f3	Résultat
const0	0		0000
const-1	0		1111
add	1		F = A plus B plus CI
sub	1		F = A moins B moins 1 plus CI
notA	0		F = NOT(A)
notB	0		F = NOT(B)
xor	0		F =A XOR B
or	0		F = A OR B
and	0		F = A AND B
nopA	0		F = A
nopB	0		F = B

Question 1 Comment peut-on très simplement créer la génération des indicateurs Z et S par l'UAL ?

Question 2 On essaie de créer l'UAL avec un minimum de modifications de la cellule additionneur/soustracteur déjà conçue. Comment doit agir la commande de mode M sur les retenues c_i ?

On propose la cellule suivante (schéma pour la cellule de rang i) :



La sortie s_i de la cellule d'additionneur/soustracteur est égale à $a_i \oplus (b_i \oplus sub) \oplus c_i$. Si la propagation de la retenue est inhibée, $s_i = a_i \oplus (b_i \oplus sub)$. Pour concevoir l'UAL, on pose $o_i = v_i \oplus (c_i.M)$, avec $v_i = a_i \oplus (b_i \oplus sub)$ pour l'addition/soustraction.

On remarque que : $x \oplus y = x.y \oplus (x + y)$. On a donc :

$v_i = a_i(b_i \oplus sub) \oplus (a_i + b_i \oplus sub) = a_i(b_i.\overline{sub} + \overline{b_i}.sub) \oplus (a_i + b_i.\overline{sub} + \overline{b_i}.sub)$

Pour avoir plus de choix sur les termes pris en compte, on remplace les occurrences de sub et $\overline{sub}$ par $f0, f1, f2, f3$. On obtient donc : $v_i = a_i(b_i.f0 + \overline{b_i}.f1) \oplus (a_i + b_i.f2 + \overline{b_i}.f3)$

Pour simplifier le travail, on notera t_i le premier terme de l'expression de v_i et u_i le deuxième terme. On a donc : $t_i = a_i(b_i.f0 + \overline{b_i}.f1)$ et $u_i = a_i + b_i.f2 + \overline{b_i}.f3$

Question 3 Quelles valeurs doivent prendre M et les variables de sélection f_j pour l'addition ? la soustraction ?

Question 4 Déterminer les valeurs de t_i et u_i suivant les valeurs des f_j, puis ce que vaut v_i. En déduire les valeurs des f_j nécessaires pour les opérations logiques indiquées dans le tableau de fonctionnement de l'UAL.

Question 5 Rechercher l'ensemble des opérations arithmétiques que peut faire cette UAL.

TD 5
Synthèse d'automates

Méthodologie
Ex. 1 : Méthodologie de synthèse d'automates : reconnaisseur de séquences

Soit un circuit prenant en entrée une lettre dans l'ensemble $\{a, b, c\}$ et générant en sortie une lettre dans l'ensemble $\{o, n\}$ qui vaut o ssi on vient de reconnaître la séquence $a + bcc^*$ et n sinon. Noter que la séquence de caractères en entrée est infinie : l'automate ne s'arrête jamais (*i.e.* pas d'état puits).

Graphe d'états de l'automate

On spécifiera en général l'automate comme un automate de Moore (sortie associée aux états).

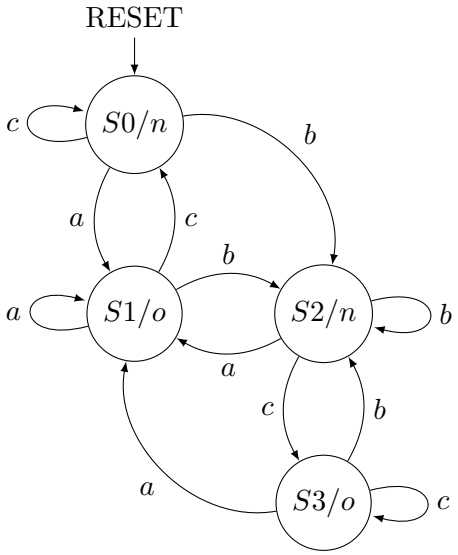


Table de transitions

Cette table est une représentation textuelle du graphe.

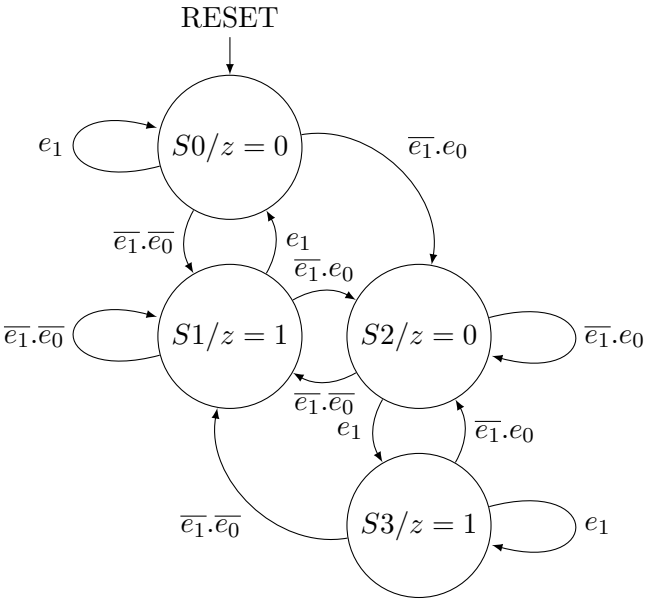
Etat courant	Sortie	Entrée	Etat futur
S0	n	a	S1
		b	S2
		c	S0
S1	o	a	S1
		b	S2
		c	S0
S2	n	a	S1
		b	S2
		c	S3
S3	o	a	S1
		b	S2
		c	S3

Codage des entrées et des sorties

On a besoin de 2 bits e_1 et e_0 pour coder les trois lettres en entrée : $a \equiv 00$, $b \equiv 01$ et $c \equiv 10$. On a besoin de 1 bit z pour coder les deux lettres en sortie : $n \equiv 0$ et $o \equiv 1$.

Remarque : en général, le codage des entrées et des sorties d'un automate sur des variables booléennes est imposé par l'environnement, et cette étape ne sera pas nécessaire.

On peut réécrire le graphe d'états et la table de transitions de l'automate en utilisant ce codage des entrées et de la sortie :



Etat courant	Sortie	Entrée	Etat futur
S0	0	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		e_1	S0
S1	1	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		e_1	S0
S2	0	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		e_1	S3
S3	1	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		e_1	S3

Codage logarithmique des états

On choisit ici un codage logarithmique, c'est-à-dire avec le nombre minimum de variables booléennes (le codage 1 parmi n, qui utilise une variable par état, est présenté à la fin de l'exercice).

En codage logarithmique, pour coder les 4 états possibles, il faut au minimum 2 bits q_1 et q_0. Pour cet exercice, on peut choisir le codage : $S0 \equiv 00$, $S1 \equiv 01$, $S2 \equiv 10$ et $S3 \equiv 11$.

Expressions simplifiées des variables d'état et de la sortie de l'automate

Il s'agit de trouver les expressions des variables q_1 et q_0 à l'instant $t+1$, en fonction de q_1, q_0 et des entrées à l'instant t. Nous utilisons des bascules D : pour obtenir la valeur souhaitée à l'instant $t+1$ en sortie des bascules, il faut mettre à l'instant t cette valeur sur les entrées D des bascules. On cherchera donc les expressons simplifiées de D_1 et de D_0 en fonction de q_1, q_0 et des entrées e_1 et e_0.

q_1q_0 / e_1e_0	**00**	**01**	**11**	**10**
00	0	0	0	0
01	1	1	1	1
11	Φ	Φ	Φ	Φ
10	0	0	1	1

Expression de D_1

q_1q_0 / e_1e_0	**00**	**01**	**11**	**10**
00	1	1	1	1
01	0	0	0	0
11	Φ	Φ	Φ	Φ
10	0	0	1	1

Expression de D_0

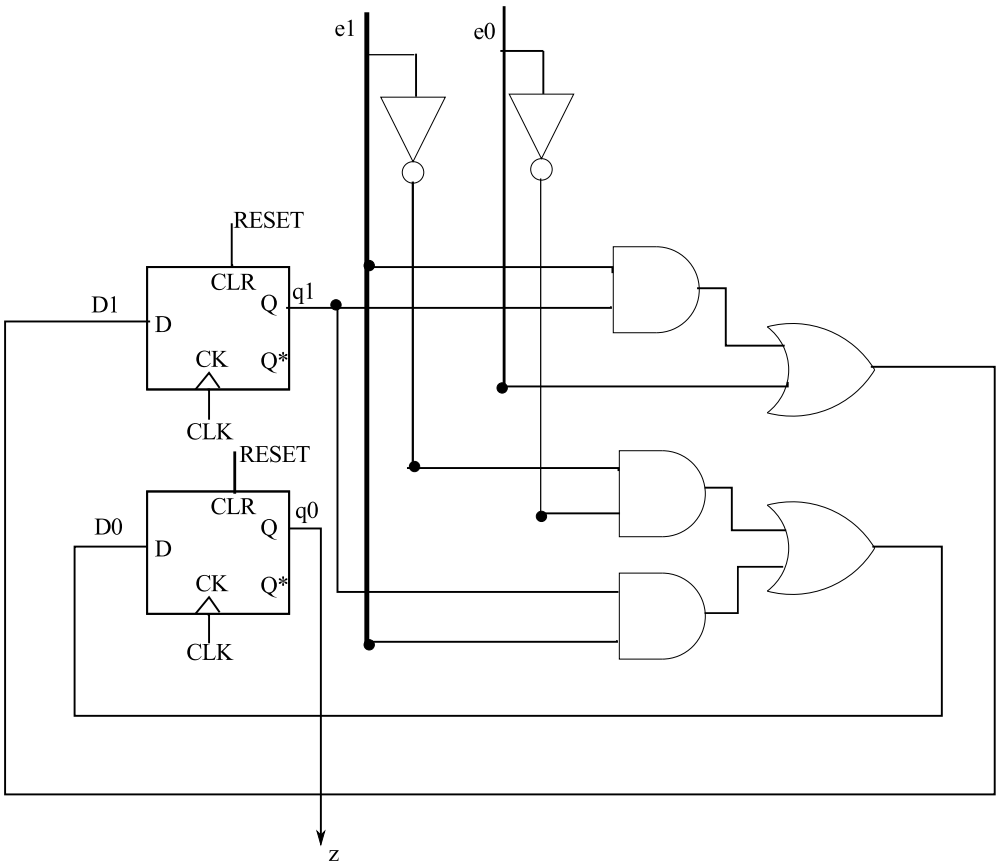
q_1 / q_0	**0**	**1**
0	0	1
1	0	1

Expression de z

On a donc :

$$\begin{aligned} D_1 &= e_0 + e_1.q_1 \\ D_0 &= \overline{e_1}.\overline{e_0} + q_1.e_1 \\ z &= q_0 \end{aligned}$$

Schéma du circuit



Codage 1-parmi-n ou "one-shot"

Le principe de ce codage est d'associer une variable d'état à chaque état (donc une bascule par état). Une variable d'état (ou une bascule) à 1 signifie que l'état correspondant est actif ; quand la variable d'état est à 0, l'état est inactif.

On peut faire la synthèse du circuit à partir du graphe d'états ou de la table de transition.

Pour notre exemple : il y a 4 états, donc 4 bascules (entrées D_0, D_1, D_2 et D_3, sorties Q_0, Q_1, Q_2 et Q_3 avec la bascule i correspondant à l'état S_i). A partir de la table de transition on obtient la table de vérité des entrées de bascules :

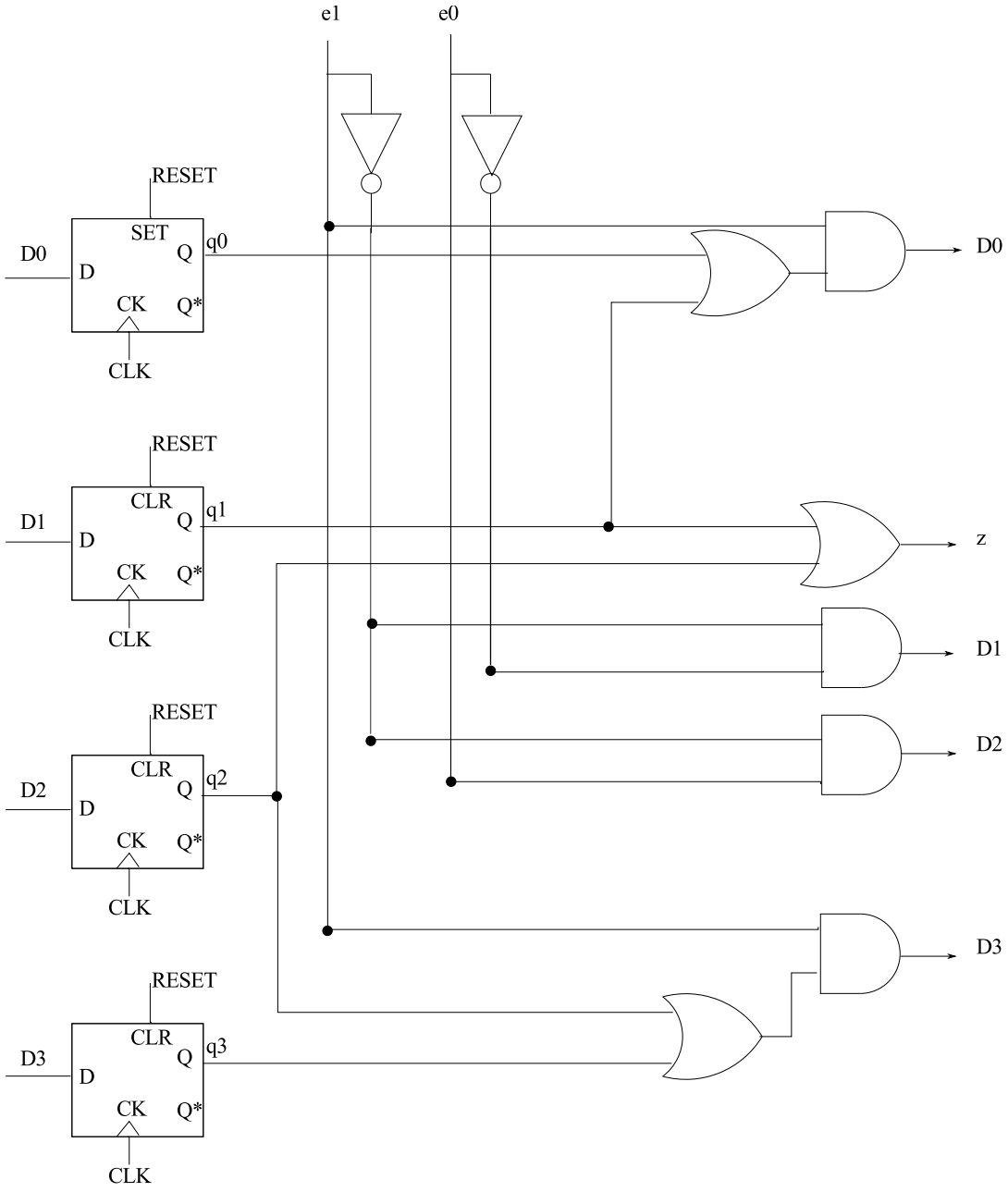
Etat courant	Sortie	Entrée	D0	D1	D2	D3
Q0	0	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		e_1	1	0	0	0
Q1	1	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		e_1	1	0	0	0
Q2	0	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		e_1	0	0	0	1
Q3	1	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		e_1	0	0	0	1

A partir de cette table, on obtient :

$$\begin{aligned} D_0 &= e_1.(Q_0 + Q_1) \\ D_1 &= \overline{e_1}.\overline{e_0}.(Q_0 + Q_1 + Q_2 + Q_3) \quad (1) \\ &= \overline{e_1}.\overline{e_0} \quad (2) \end{aligned}$$

$$
\begin{aligned}
D_2 &= \overline{e_1}.e_0.(Q_0 + Q_1 + Q_2 + Q_3) && (3)\\
&= \overline{e_1}.e_0 && (4)\\
D_3 &= e_1.(Q_2 + Q_3) && (5)\\
z &= Q_1 + Q_3 && (6)
\end{aligned}
$$

Le circuit correspondant est donné ci-dessous. Noter que la bascule associée à l'état initial Q_0 est initialisée à 1 (pour que cet état devienne actif à l'initialisation), toutes les autres sont initialisées à 0.



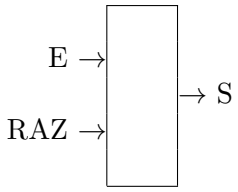
Ex. 2 : Décodeur « Non-Retour à Zéro Inversé »

On désire faire un module matériel permettant de décoder un signal NRZI avec un automate de Moore synchrone possédant 4 états, notés $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ et $\mathcal{D}$. Cet automate possède 2 entrées E et RAZ, et une unique sortie S. Les signaux sont les suivants :

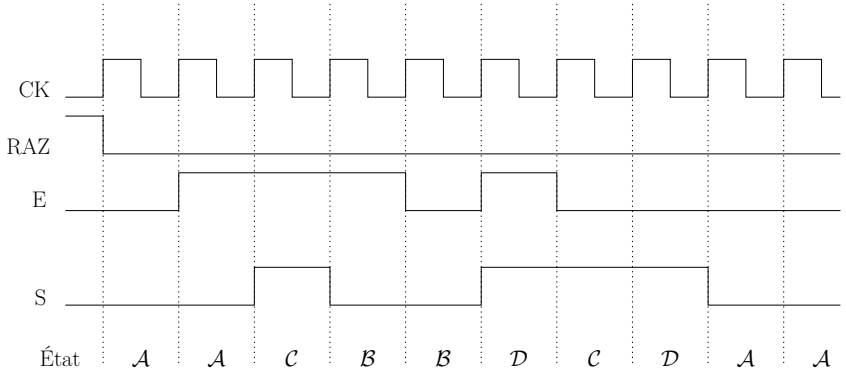
E fournit la donnée série (1 bit), c'est à dire la suite de bits à décoder un par un ;

RAZ force l'automate dans son état initial, l'état $\mathcal{A}$, lorsqu'elle est à '1' ;

S est la sortie série.



Le chronogramme suivant montre un exemple de décodage qui illustre également le fonctionnement de l'automate.



Question 1 En utilisant ce chronogramme, déterminez le graphe d'états, en précisant les conditions de transitions et les valeurs de sortie pour chaque état.

Question 2 Les états de l'automate sont codés ainsi, sur deux bits notés Q1 et Q0.

	Q1	Q0
$\mathcal{A}$	0	0
$\mathcal{B}$	0	1
$\mathcal{C}$	1	0
$\mathcal{D}$	1	1

Construire la table de transition de l'automate, en notant D1 et D0, les valeurs futures à écrire dans les bascules D mémorisant Q1 et Q0.

Question 3 Donnez les expressions booléennes simplifiées de D1, D0 et S.

Pour aller plus loin...
Question 4 Proposez une réalisation de cet automate en utilisant un codage 1 parmi n.

Pour aller plus loin...
Question 5 Dessiner le schéma du circuit permettant de décoder le code NRZI.

Utilisation de cet automate : Le codage NRZI (Non-Retour à Zéro Inversé) consiste à changer le niveau du signal codé chaque fois que l'on code un 1 (si le signal de sortie vaut 0, alors il devra valoir 1 au cycle suivant, et inversement) et à conserver le niveau du signal codé chaque fois que l'on code un 0 (si le signal de sortie vaut 0 au cycle courant, alors il y reste au cycle suivant, *idem* pour 1). Le signal de sortie est initialement à 0. Dans cet exercice, on a proposé un module matériel permettant de décoder un signal NRZI avec un automate de Moore synchrone possédant 4 états.

Ex. 3 : Gestion d'un feu tricolore

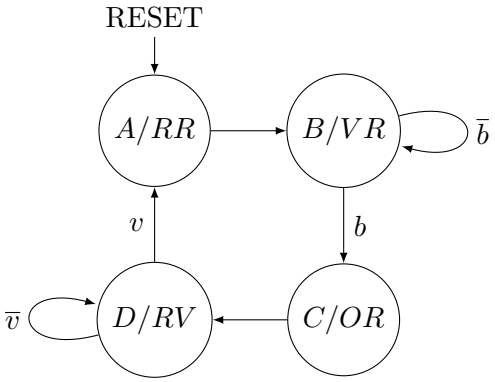
On cherche à modéliser le fonctionnement d'un feu tricolore composé d'un affichage à trois couleurs (Rouge, Orange, Vert) pour les véhicules et d'un affichage bicolore (Rouge, Vert) pour les piétons. Le feu fonctionne selon le principe très simplifié détaillé ci-dessous :

– le feu véhicules reste au vert tant qu'un piéton n'a pas appuyé sur le bouton d'appel ;

– le feu véhicules passe à l'orange dès qu'un piéton appuie sur le bouton d'appel ($b = 1$), puis après un certain temps il passe au rouge et le feu piétons passe au vert simultanément ;
– le feu piétons reste au vert tant qu'aucun véhicule n'est arrêté au feu ;
– lorsqu'un véhicule est détecté en attente devant le feu ($v = 1$), le feu piétons passe au rouge, puis après un certain temps, le feu véhicules passe au vert.

Pour simplifier, on considère que la notion de temps d'attente est manifestée par le passage d'un cycle (*i.e.* on ne prend pas en compte des durées d'attente différentes). Par convention, on décide que l'état initial est celui où les deux feux sont au rouge.

On donne un automate de Moore qui modélise le comportement de ce feu tricolore. Comme un énoncé en langue naturelle est toujours beaucoup plus imprécis qu'un modèle rigoureux, des choix d'interprétation ont été fait lors de la conception de cet automate : on considérera l'automate comme l'énoncé de référence.



Dans chaque état, la sortie est représentée par un couple de couleurs correspondant à l'affichage du feu véhicules suivi de celui du feu piétons.

Question 1 a) Combien y a t-il de combinaisons possibles et de combinaisons réelles pour les valeurs des feux piétons et véhicules ?
b) Peut-on établir un lien avec le nombre d'états de l'automate ?

Question 2 Choisir un codage des états de l'automate et donner les expressions simplifiées des variables d'état de l'automate.

Question 3 Donner les expressions simplifiées des sorties de l'automate.

Pour aller plus loin...
Question 4 Dessiner le schéma du circuit pilotant le feu tricolore.

Pour aller plus loin...
Ex. 4 : Gestion de la température par hysteresis

On cherche à réaliser un système permettant d'assurer le maintien à une température quasi-constante d'un poulailler industriel, afin d'assurer aux volatiles une durée de vie compatible avec les normes Européennes. Ce système repose sur la disponibilité d'un climatiseur pouvant produire du chaud, du froid, ou ne rien faire, en fonction de la température courante de l'entrepôt (qui dépend de la température externe et du degré d'agitation des gallinacées).

Le système est basé sur le principe de l'hysteresis, c'est-à-dire que le comportement lors de l'accroissement de la température est différent du comportement lorsque celle-ci baisse. Trois températures de référence sont nécessaires au fonctionnement du système : $T_{min} < T_{nom} < T_{max}$. La production du chaud ou du froid, indiquée respectivement par un signal C à 1 ou F à 1, se passe comme suit :
– lorsque la température T devient inférieure à T_{min}, il y a production de chaud : $C \leftarrow 1$;
– la production de chaud s'arrête, *i.e.* $C \leftarrow 0$, lorsque la température devient supérieure ou égale à la valeur nominale T_{nom} ;
– lorsque la température devient supérieure ou égale à T_{max}, il y a production de froid $F \leftarrow 1$;
– la production de froid cesse lorsque la température devient inférieure à la valeur nominale T_{nom}.

Question 1 Peut-on représenter les valeurs de C et F en fonction de la température sous la forme de tables de vérité ? Justifiez.

Le système exploite un capteur de température qui fournit une information encodée sur 2 bits comme précisé ici :

b_1	b_0	cas
0	0	$T < T_{min}$
0	1	$T_{min} \leq T < T_{nom}$
1	1	$T_{nom} \leq T < T_{max}$
1	0	$T_{max} \leq T$

On cherche à formaliser le comportement du système comme un automate de Moore, en faisant l'hypothèse que la période de l'horloge de l'automate est très inférieure au temps qu'il faut pour que la température T du poulailler passe de T_{min} à T_{max} ou inversement, à cause de l'inertie thermique.

Question 2 Précisez quelles sont les entrées et les sorties de l'automate.

Question 3 Donnez le nombre d'états de l'automate et représentez le graphe de l'automate en y incluant les conditions de transitions sous forme *d'intervalle de températures* sur les arcs et les valeurs des sorties dans les états. Transformez ensuite les intervalles de températures en conditions booléennes.

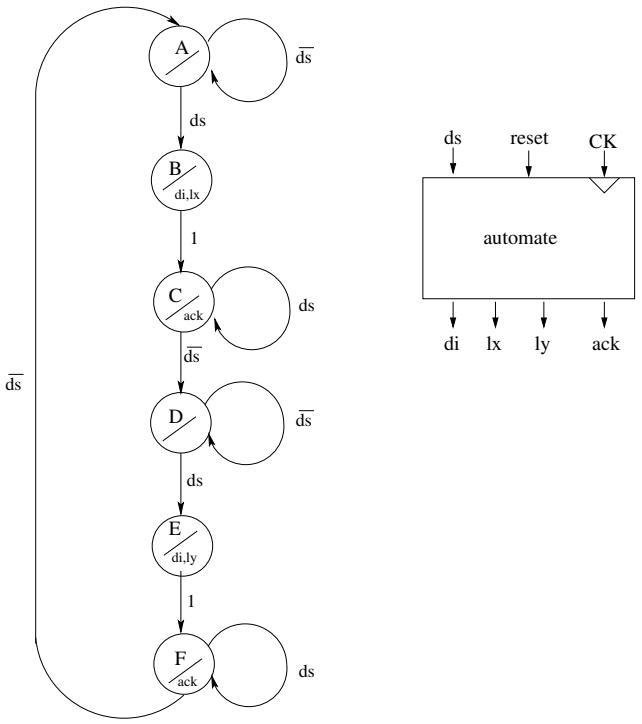
Question 4 Proposez un codage logarithmique des états de façon à ce que les expressions des sorties C et F soient simples et donnez la table de transition qui précise les sorties et l'état futur en fonction des entrées et de l'état courant. On note Q_i les sorties du registre représentant l'état courant et D_i les entrées du registre, représentant l'état futur.

Question 5 Donnez les expressions simplifiées de C, F et des D_i, et faites le schéma en portes correspondant.

Pour aller plus loin...

Ex. 5 : automate de synchronisation

Soit un automate spécifié par le graphe ci-dessous. Outre l'entrée d'initialisation $RESET$, cet automate a une entrée : ds et 4 sorties di, ack, lx, ly. Il évolue au front montant de l'horloge CK.



Question 1 Construire la table de transition de cet automate.

Question 2 Proposez une réalisation de cet automate utilisant un codage 1 parmi n (one shot encoding) à partir de bascules D.

Question 3 On cherche à réaliser cet automate avec un codage logarithmique (avec 3 variables d'état, puisque'il y a 6 états). Rechercher un codage permettant de minimiser le nombre total de monômes dans les expressions de ces variables d'état et des sorties, puis donner les équations.

TD 6
Architecture PC/PO

Ex. 1 : PGCD

On travaille sur un circuit calculant le PGCD de deux entiers naturels strictement positifs, selon l'algorithme ci-dessous :

```
procedure PGCD is
  A, B: Positive; -- on suppose que les entiers sont codés sur 8 bits
begin
  Get(A); -- A :=A0
  Get(B); -- B :=B0
  while A /= B loop
    if A < B then
      B := B - A;
    else
      A := A - B;
    end if;
  end loop;
  Put(B);
end;
```

On décide qu'ici les paramètres ainsi que le résultat du calcul sont codés sur 8 bits.

Question 1 On commence par travailler sur la partie opérative du circuit PGCD. Quels composants de base peuvent être utilisés pour matérialiser les éléments de l'algorithme précédent ?

En se rappelant ce qui a été vu au TD4, à partir d'un soustracteur 8 bits, on peut obtenir le signe du résultat (A-B) à partir de la retenue sortante (son inverse) et l'égalité (A == B) avec un NOR8 sur les bits 0-7. On suppose donc avoir un soustracteur disposant de ces deux sorties.

Question 2 En utilisant ce soustracteur et tous les composants de base nécessaires, construire la partie opérative du circuit PGCD. Identifier les signaux de compte-rendu envoyés à la partie contrôle. Penser à nommer systématiquement tous les signaux de contrôle de la PO.

Question 3 Avec la PO construite, relever les actions pouvant être réalisées en parallèle dans l'algorithme de calcul du PGCD.

Question 4 Dessiner l'automate de contrôle du circuit PGCD en précisant bien tous les signaux de contrôle envoyés à la PO, et en prenant en compte les signaux de compte-rendu envoyés par la PO.

Le circuit PGCD sera en pratique utilisé par un autre circuit qui lui fournira ses opérandes A et B et récupérera le PGCD à la fin du calcul. Pour permettre la synchronisation entre le circuit utilisateur et le circuit PGCD, on introduit deux signaux de commandes :

- *start* est un signal envoyé par le circuit utilisateur au circuit PGCD pour lui demander de démarrer le calcul : il faut donc que les bonnes valeurs de A et B soient stabilisées en entrée du circuit PGCD quand *start* passe à 1 ;
- *done* est un signal envoyé par le circuit PGCD au circuit utilisateur pour lui signifier que le calcul est terminé et que le PGCD de A et B est disponible en sortie.

Question 5 Ajouter la gestion de ces deux signaux de commande. Quelle partie du circuit vous semble la plus adaptée pour gérer les communications avec l'extérieur du circuit PGCD ?

Méthodologie : L'ordre des questions de ce TD représente les différentes étapes permettant de traduire un algorithme en circuit. Ces étapes sont :

1. Identification des variables à mettre en registre
2. Identification des opérations et choix des opérateurs
3. Réalisation de la PO : chaque ligne de l'algorithme définit un chemin de données entre les composants de base retenus précédemment. Lorsque le chemin passe par un composant de base déjà utilisé, un multiplexeur est ajouté sur le chemin de donnée. Le multiplexeur est commandé par un signal de contrôle.
4. Réalisation de la PC sous la forme d'un automate : chaque ligne de l'algorithme (éventuellement réécrit sous une forme optimisée) est affectée à un état. Les états sont reliés entre eux pour exprimer la séquentialité de l'algorithme. Les entrées de contrôle et les signaux de compte-rendus sont utilisés pour exprimer la fonction de transition.

TD 7
Architecture PC/PO

Ex. 1 : Bresenham

Le tracé de lignes, de cercles, d'ellipse, etc, est une opération réalisée très communément sur les cartes graphiques. Des algorithmes spécifiques ont été développés pour faire ces tracés sur des grilles de pixels (par ex. un écran) en utilisant uniquement des nombres entiers et sans recourir à la division (pour la ligne) ou la racine carrée ou les fonctions trigonométriques pour les cercles et ellipses. On se propose d'utiliser un algorithme du à Jack Bresenham [1] qui permet de tracer un quart de cercle pour en faire la conception PC/PO.

L'algorithme est le suivant :

```
   // r est le rayon, disponible au début de l'algorithme
 1 x := 0;
 2 y := r;
 3 m := 5 − 4 × r;
 4 while x ≤ y do
       // out indique que les valeurs (x, y) sont disponible, il n'y a pas
          d'opérateur associé
 5     out(x, y);
 6     if m > 0 then
 7         y := y − 1;
 8         m := m − 8 × y;
 9     end if
10     m := m + 8 × x + 12;
11     x := x + 1;
12 end while
```

On va utiliser la méthode présentée en cours et mise en œuvre en TD pour proposer une implantation de cet algorithme sous la forme PC/PO.

Question 1 Définissez l'ensemble des opérations à réaliser ; déduisez-en le type des unités fonctionnelles (registres ou opérateurs). On cherchera à utiliser les opérateurs *les plus simples* (c'est-à-dire qui implantent une opération arithmétique ou logique et une seule) pour chaque opération. On rappelle que les soustracteurs fournissent (possiblement entre autres) les informations suivantes : z qui vaut 1 si le résultat est nul, 0 sinon et s qui vaut 1 si le résultat est strictement négatif, 0 sinon.

Question 2 Quelles opérations peut-on réaliser en parallèle ?
Réécrivez le programme en utilisant 1) la notion d'affectation concurrente présentée en cours et 2) en remplaçant les opérations par les opérations des opérateurs « simples » déterminés précédemment. Rappel. : $(\alpha, \beta) \leftarrow (3, 1)$ affecte 3 dans α et 1 dans β.

Question 3 Donnez le nombre d'opérateurs de chaque type nécessaire pour exécuter une ligne de l'algorithme en 1 cycle et proposez une partie opérative interconnectant les unités fonctionnelles. Indiquer clairement les signaux de commandes (sélecteurs de multiplexeurs, signaux de chargement de registres, etc) et les comptes-rendus.

Question 4 Proposez une partie contrôle qui pilote cette partie opérative. On fait l'hypothèse que l'entrée r est magiquement disponible lorsqu'on en a besoin, et que *out* est un état dans lequel on

1. *"A linear algorithm for incremental digital display of circular arcs", Jack Bresenham, Communication of the ACM, Feb 1977, vol. 20, n. 2.*

passe un cycle et un seul. Spécifiez les opérations de transfert registre à registre (RTL) dans chaque état. Donnez la fonction de sortie sous la forme d'un tableau.

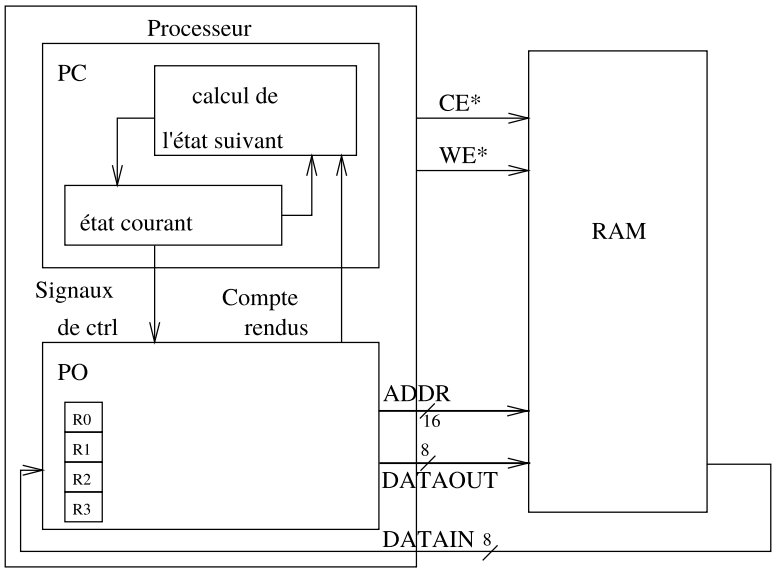
Question 5 Modifiez l'algorithme en utilisant la notion d'affectation conditionnelle pour faire disparaître le **if**. Pour mémoire : $\nu \leftarrow \gamma ? \alpha : \beta$ affecte α dans ν si $\gamma = 1$, β sinon.

Pour aller plus loin...
Question 6 Proposez une PO, puis une PC correspondant à la question précédente.

Pour aller plus loin...
Question 7 En repartant de la forme initiale de l'algorithme, proposez une PO n'utilisant qu'un additioneur/soustracteur. Donnez ensuite la PC correspondante.

TD 8
Conception d'un processeur

L'objectif de ce TD est de construire un processeur capable d'effectuer des instructions très simples. Le processeur construit sera utilisé dans les deux séances de TD suivantes. On suppose que ce dernier est équipé de 4 registres internes de 8 bits chacun (nommés `r0`, `r1`, `r2` et `r3`), d'un bus adresse sur 16 bits et de 2 bus donnée (lecture et écriture) sur 8 bits. On rappelle l'architecture globale du processeur et sa liaison avec la mémoire :



Le jeu d'instructions de ce processeur est le suivant :

- `st ri, @mem` stocke le contenu du registre `ri` dans l'octet situé à l'adresse `@mem` ;
- `ld @mem, ri` stocke le contenu de l'octet situé à l'adresse `@mem` dans le registre `ri` ;
- `op rs, rd` stocke dans le registre `rd` le résultat de l'opération `rd op rs`.

Les opérations à deux opérandes supportées par ce processeur sont l'addition (instruction `add`), la soustraction (instruction `sub`), la conjonction (instruction `and`), la disjonction (instruction `or`) et la disjonction exclusive (instruction `xor`).

Les opérations unaires supportées (`rd := op rd`) sont la négation (instruction `not`), le décalage d'un bit vers la gauche (instruction `shl`) et le décalage arithmétique d'un bit vers la droite (instruction `shr`).

On donne également l'algorithme exécuté par le processeur pour récupérer, décoder et exécuter les instructions d'un programme situé à l'adresse $0x4000$:

```
1  PC ← 0x4000;
2  while True do
3  |  IR ← MEM(PC) ;
4  |  PC ← PC + 1 ;
5  |  switch IR do
6  |  |  case add :  rd ← rd + rs  ;                    // rd, rs ∈ {r0, r1, r2, r3}
7  |  |  case sub :  rd ← rd − rs ;
8  |  |  case and :  rd ← rd AND rs ;
9  |  |  case or :  rd ← rd OR rs ;
10 |  |  case xor :  rd ← rd XOR rs ;
11 |  |  case not :  rd ← NOT rd ;
12 |  |  case shl :  rd ← rd << 1 ;
13 |  |  case shr :  rd ← rd >> 1 ;
14 |  |  case st :
15 |  |  |  AD(15 : 8) ← MEM(PC) ;
16 |  |  |  PC ← PC + 1;
17 |  |  |  AD(7 : 0) ← MEM(PC);
18 |  |  |  PC ← PC + 1;
19 |  |  |  MEM(AD) ← ri  ;                            // ri ∈ {r0, r1, r2, r3}
20 |  |  case ld :
21 |  |  |  AD(15 : 8) ← MEM(PC) ;
22 |  |  |  PC ← PC + 1;
23 |  |  |  AD(7 : 0) ← MEM(PC);
24 |  |  |  PC ← PC + 1;
25 |  |  |  ri ← MEM(AD) ;
26 |  |  endsw
27 |  endsw
28 end while
```

On remarquera que l'algorithme suppose que les instructions d'accès à la mémoire sont sur 3 mots et les autres instructions sur un seul mot.

Ex. 1 : Construction d'un processeur sur le modèle PC-PO

Question 1 Identifier et lister les registres dans cet algorithme. Préciser leur taille, puis expliquer leur rôle et leur fonctionnement.

Question 2 Définir l'ensemble des opérations à réaliser. Proposer des opérateurs pour réaliser ces actions, en cherchant à utiliser le moins d'opérateurs possible.

Question 3 Proposez une partie opérative interconnectant les unités fonctionnelles identifiées (utiliser le composant décrit en annexe). Indiquer clairement les signaux de commandes (sélecteurs de mux, signaux de chargement de registres, etc) et les comptes-rendus.

Question 4 Repérer les actions de l'algorithme pouvant être réalisées en parallèle sur votre PO. Réecrire l'algorithme en utilisant la notion d'écriture concurrente. Pour piloter la PO, proposer une machine d'état correspondant au nouvel algorithme en précisant uniquement les opérations de transfert de registre à registre [1]. Ne préciser ni les signaux de contrôle ni les conditions sur transitions pour l'instant.

Question 5 Simplifier la PC de manière à réduire le nombre d'états successeurs de l'état "decode"

1. C'est à dire les opérations réalisées dans l'algorithme décrivant le processeur.

(correspondant au test du switch).

On va chercher maintenant à écrire les valeurs des conditions sur les transitions de la partie contrôle. Ces conditions de transitions dépendent du codage retenu pour chacune des instructions. Pour cela, on va utiliser des fonctions booléennes (f(IR)) :

- *AccesMem(IR)* : fonction exprimant si le codop de l'instruction stockée dans IR est un accès à la mémoire ;
- *AccesEcr(IR)* : fonction exprimant si le codop de l'instruction stockée dans IR est un accès à la mémoire en écriture.

Question 6

Compléter les conditions manquantes sur les transitions de votre automate.

Question 7

On donne les fonctions booléennes suivantes :

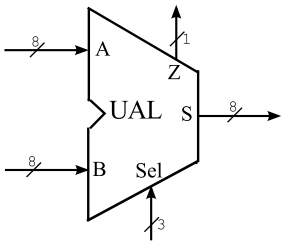
- *selRs(IR)*, *selRd(IR)*, *selRd(IR)* : Fonction prenant les 8 bits de IR en entrée et retournant les 2 bits déterminant le numéro de registre Rs,Rd ou Ri de l'instruction décodée.
- *selUAL(IR)* : Fonction retournant l'opération réalisée par l'instruction en la codant sur 3 bits selon le codage de l'UAL fournit en annexe.

Exprimer, sous forme de tableau, pour chaque état de la PC la valeur des signaux de contrôle. Il s'agit de traduire les opérations registres à registres en signaux de contrôle.

Ex. 2 : Conception du codage du jeu d'instruction

Question 1 Proposer un codage astucieux de chaque instruction en essayant de minimiser les fonctions utilisées précédemment.

Annexe : UAL à utiliser dans ce TD



Sel	S
000	A or B
001	A xor B
010	A and B
011	not A
100	A + B
101	A - B
110	A « 1
111	A » 1

Z=1 ssi S=0

Pour aller plus loin...

Ex. 3 : Réalisation de l'UAL

Construire l'UAL décrite ci-dessus en utilisant l'UAL vu au TD5, dont le fonctionnement est rappelé ci-dessous, et deux multiplexeurs 8 bits 2 vers 1. Donner les fonctions numériques dépendant des bits de **sel** permettant de contrôler l'UAL (via CI, M, f0, f1, f2 et f3) et les multiplexeurs.

Opération	Mode M	f0 f1 f2 f3	Résultat
add	1	1010	F= A + B + CI
sub	1	0101	F= A - B + 1 - CI
notA	0	1111	F=not(A)
xor	0	1010	F=A XOR B
or	0	0010	F = A OR B
and	0	0100	F = A AND B

TD 9
Du programme en mémoire aux actions dans le processeur

L'objectif de ce TD est de détailler le fonctionnement du processeur construit à la séance précédente. Le codage du jeu d'instructions pour cette séance est le suivant :

Instruction	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Opérations de la forme : $rd := rd\ op\ rs$								
`or rs, rd`	0	0	0	0	rs_1	rs_0	rd_1	rd_0
`xor rs, rd`	0	0	0	1	rs_1	rs_0	rd_1	rd_0
`and rs, rd`	0	0	1	0	rs_1	rs_0	rd_1	rd_0
`add rs, rd`	0	1	0	0	rs_1	rs_0	rd_1	rd_0
`sub rs, rd`	0	1	0	1	rs_1	rs_0	rd_1	rd_0
Opérations de la forme : $rd := op\ rd$								
`not rd`	0	0	1	1	0	0	rd_1	rd_0
`shl rd`	0	1	1	0	0	0	rd_1	rd_0
`shr rd`	0	1	1	1	0	0	rd_1	rd_0
Chargement : $rd := MEM(AD)$								
`ld AD, rd`	1	0	0	0	0	0	rd_1	rd_0
	ADH							
	ADL							
Stockage : $MEM(AD) := rs$								
`st rs, AD`	1	1	0	0	0	0	rs_1	rs_0
	ADH							
	ADL							

Question 1 Soit le contenu de la mémoire donné ci-dessous. Ré-écrire ce programme en langage machine. Que fait ce programme ?

Contenu de la mémoire :

Adresse	**Valeur**
$0x1230$	$0x00$
$0x1231$	$0x0A$
$0x1232$	$0x1A$
$0x1233$	$0x12$
...	...
$0x4000$	$0x80$
$0x4001$	$0x12$
$0x4002$	$0x31$
$0x4003$	$0x81$
$0x4004$	$0x12$
$0x4005$	$0x32$
$0x4006$	$0x44$
$0x4007$	$0x60$
$0x4008$	$0x81$
$0x4009$	$0x12$
$0x400A$	$0x33$
$0x400B$	$0x54$
$0x400C$	$0xC0$
$0x400D$	$0x12$
$0x400E$	$0x30$
...	...

Question 2 On s'intéresse à l'instruction présente à l'adresse 0X4006. Quels sont les chemins de données utilisés dans la PO durant l'exécution ? Quels sont les états empruntés par la PC pour traiter cette instruction ? En combien de cycles la traite t-on ? Pour résumer cette analyse, dessiner un chronogramme faisant apparaître l'horloge et la valeur des signaux importants de la PC et de la PO durant le traitement de cette instruction.

Pour aller plus loin...
Question 3 Même exercice avec l'instruction à l'adresse 0x4008

TD 10
Modification d'un processeur

L'objectif de ce TD est de montrer les conséquences de l'ajout d'instructions sur la micro-architecture d'un processeur et sa machine d'état. Le processeur utilisé est le même que dans les séances précédentes.

Méthodologie
Ex. 1 : Chargement immédiat : intérêt et modification sur le processeur

Pour simplifier l'accès aux constantes dans un programme, on propose d'ajouter une instruction qui permet de charger un registre (8 bits) avec une constante (8 bits également). Cette nouvelle instruction sera notée `li IMM, rd`. Généralement nommée *load immediat* en anglais, elle effectue l'affectation suivante : $rd \leftarrow IMM$.

Question 1 Indiquez où doit se trouver la constante à charger dans le registre et proposez un encodage de l'instruction `li` qui s'insère simplement dans l'encodage existant.

La constante doit se trouver en mémoire à l'adresse PC+1. Un encodage possible est le suivant :

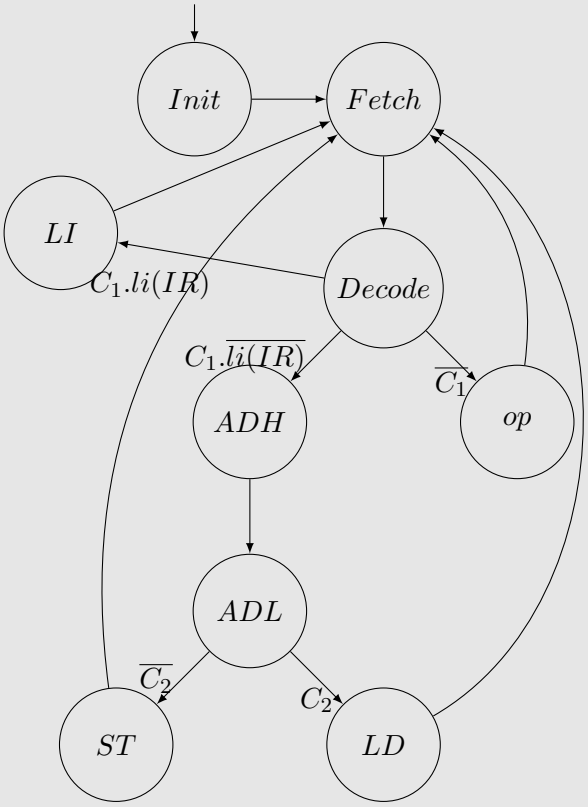
Instruction	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
`li IMM, rd`	1	0	0	0	1	0	rd_1	rd_0

Question 2 Est-il nécessaire de modifier la partie opérative ? Si oui, effectuez les modifications. Si non, justifier votre réponse.

Inutile de modifier la PO car il existe déjà le mux sourceR pour amener dataIN aux registres généraux.

Question 3 Etendez la machine d'état de la PC obtenue au TD9 de manière à supporter cette nouvelle instruction.

li(IR) est la fonction booléenne qui indique si un chargement est immédiat. Dans notre choix de codage, $li((IR) = IR_3$.



où $C_1 = AccesMem(IR)$, $C_2 = AccesEcr(IR)$.

Dans l'état *li*, on doit avoir CD*=0, WE*=1, EPC=1, selAD=1, sourceR=1, ldRx=f(selRd(IR)), tous les autres ld sont à 0, selA,selB,op en ϕ-booléen.

Ex. 2 : Branchement : intérêt et modification sur le processeur

Le programme ci-dessous calcule le produit de la constante `X` (située à l'adresse 0x2000) et la constante `Y` (0x2001) et stocke le résultat à l'adresse 0x1230.

Pour pouvoir exécuter ce programme sur notre processeur, on a besoin de rajouter des instructions de branchement :

- `jmp @mem` fait « sauter » l'exécution à l'adresse `@mem` (*i.e.* `PC := @mem`) ;
- `jz @mem` provoque un branchement à l'adresse `@mem` ssi $Z = 1$ (où Z est l'indicateur valant 1 ssi le résultat de la dernière opération était nul).

```
Cpt := Y;
Res := 0;
while Cpt /= 0 loop
  Res := Res + X;
  Cpt := Cpt - 1;
end loop;
```

```
0x4000: ld 0x2000, r0
0x4003: ld 0x2001, r1
0x4006: li 1, r2
0x4008: xor r3, r3
0x4009: or r1, r1
0x400A: jz 0x____
0x400D: add r0, r3
0x400E: sub r2, r1
0x400F: jmp 0x____
0x4012: st r3, 0x1230
```

Question 1 Associer les lignes de l'algorithme écrit en pseudo-code (ci-dessus à gauche) à celles du programme écrit en langage machine (ci-dessus à droite), justifier les adresses des instructions et

compléter les champs manquants après les instructions de branchement.

Question 2 Que faut-il ajouter à la partie opérative de notre processeur pour supporter ces nouvelles instructions ?

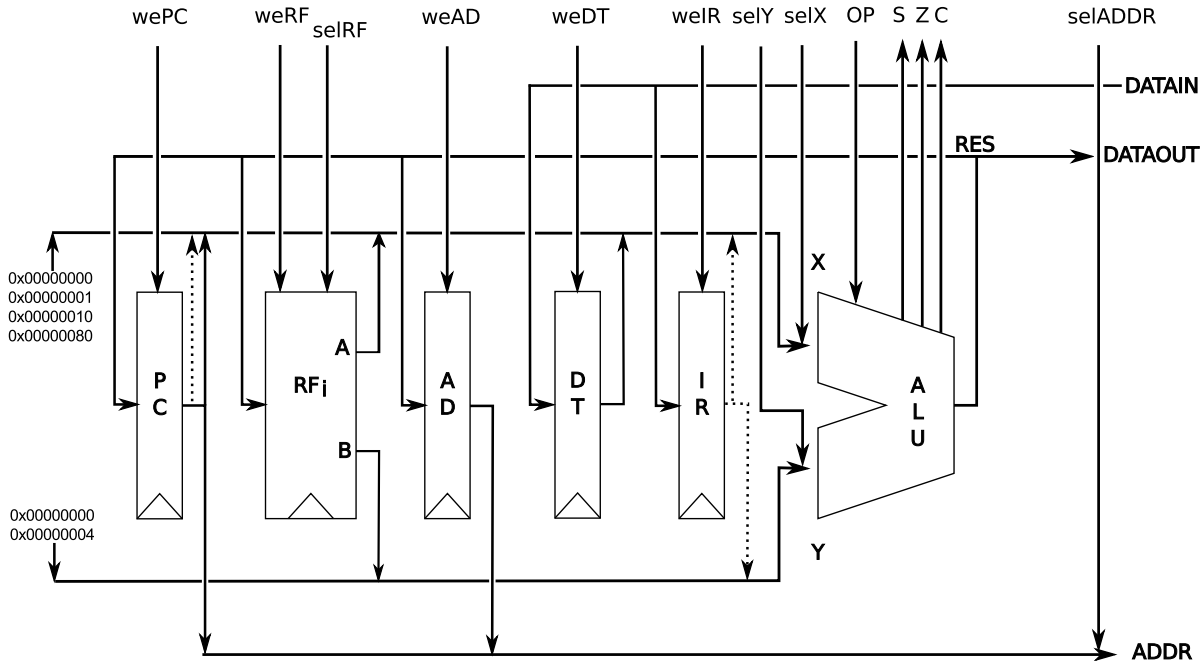
Question 3 Modifier la PC en conséquence et proposer un codage pour ces instructions.

TD 11
Programmation en langage machine MIPS

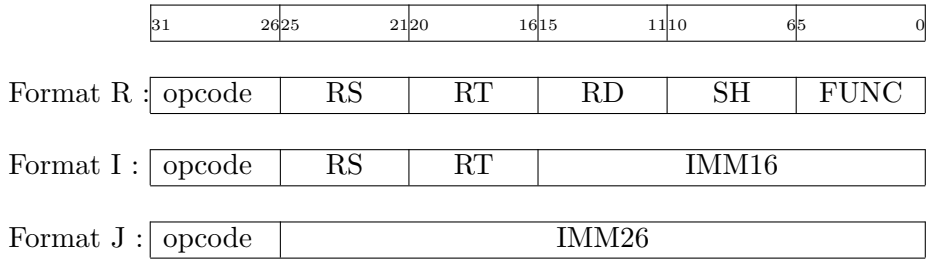
L'objectif de ce TD est de comprendre comment mettre en œuvre un algorithme sur un processeur.

Le processeur utilisé pour ce TD est le processeur MIPS, déjà présenté en cours et que vous utiliserez en TP et concevrez dans le module CEP au second semestre.

Les registres et les bus du processeur MIPS considéré sont tous sur 32 bits et le registre R0 est cablé à zéro. On rappelle l'architecture globale du processeur et sa liaison avec la mémoire :



Le MIPS a un jeu d'instructions à 3 opérandes et ses instructions sont réparties dans 3 formats différents :



Dans ce TD, nous considérerons les instructions suivantes :

Nmémonic/ syntaxe	Format	Opcode/ Func	Opération	Nom Complet
`addiu $rt, $rs, imm`	I	0x9	R[rt]=R[rs]+SignExtImm [1]	addition non signée avec un immédiat
`addu $rd, $rs, $rt`	R	0x0/0x21	R[rd]=R[rs]+R[rt]	addition non signée entre registres
`and $rd, $rs, $rt`	R	0x0/0x24	R[rd]=R[rs] & R[rt]	ET bit à bit entre registres
`andi $rt, $rs, imm`	I	0xC	R[rt]=R[rs] & ZeroExtImm [2]	ET bit à bit avec un immédiat
`beq $rs, $rt, label`	I	0x4	if(R[rs]== R[rt]) PC=PC+4+BranchAddr [3]	Branchement si égalité
`bne $rs, $rt, label`	I	0x5	if(R[rs] != R[rt]) PC=PC+4+BranchAddr [3]	Branchement si inégalité
`j label`	J	0x2	PC = JumpAddr [4]	Saut inconditionnel
`lui $rt, imm`	I	0xF	R[rt] = $IMM16 \| 0^{16}$	Chargement immédiat haut
`lw $rt, imm($rs)`	I	0x23	R[rt] = mem[R[rs]+SignExtImm [1]]	chargement registre
`or $rd, $rs, $rt`	R	0x0/0x25	R[rd] = R[rs] | R[rt]	OU bit à bit registre à registre
`ori $rt, $rs, imm`	I	0xD	R[rt] = R[rs] | ZeroExtImm [2]	OU bit à bit avec immédiat
`slt $rd, $rs, $rt`	R	0x0/0x2A	R[rd] = (R[rs] < R[rt]) ? 1 : 0	Set Less Than registre à registre
`slti $rt, $rs, imm`	I	0xA	R[rt] = (R[rs] < SignExtImm [1]) ? 1 : 0	Set Less Than avec immédiat
`sll $rd, $rt, sh`	R	0x0/0x0	R[rd] = R[rt] « SH	décalage à gauche
`srl $rd, $rt, sh`	R	0x0/0x2	R[rd] = R[rt] » SH	décalage logique à droite
`subu $rd, $rs, $rt`	R	0x0/0x23	R[rd]=R[rs]-R[rt]	soustraction non signée entre registres
`sw $rt, imm($rs)`	I	0x2B	mem[R[rs]+SignExtImm [1]] = R[rt]	stockage registre

1. $SignExtImm = IMM16_{15}^{16}|IMM16$
2. $ZeroExtImm = 0^{16}|IMM16$
3. $BranchAddr = IMM16_{15}^{14}|IMM16|0^2$
4. $JumpAddr = (PC+4)_{31\cdots28}|IMM26|0^2$

Préparation

Ex. 1 : Interprétation d'instructions

Question 1 Traduire en langage machine les instructions MIPS suivantes :
andi $5, $24, 0xFF
subu $16, $8, $4
Interpréter les mots de 32 bits suivant comme des instructions MIPS :
0x8CE3000C
0x000FF9C0

Ex. 2 : Implantations basiques

Question 1 Comment peut on charger un immédiat de 32 bits dans un registre utilisateur ?

Question 2 En supposant que A est dans le registre R1, traduisez la structure conditionnelle suivante :

```
while A /= 0 loop
      A:= A -1;
end loop;
```

Question 3 Donnez les valeurs des champs IMM des instructions de branchement utilisées en supposant que votre programme commence à l'adresse 0x80.

Question 4 En supposant que A et B sont des variables 32 bits non signées situées dans la mémoire aux adresses 0x1000 et 0x1004, traduisez la structure conditionnelle suivante :

```
if A < B then
  A := 1023;
else
  A := 511;
end if;
```

Ex. 3 : Implantation du PGCD en assembleur MIPS

Question 1 En supposant que A0 et B0 sont des valeurs 32 bits non signées qui ont été situées dans la mémoire aux adresses 0x1000 et 0x1004, traduisez l'algorithme PGCD ci-dessous. Le résultat sera mis à l'adresse 0x1008.

```
procedure PGCD is
  A, B: Positive;
begin
  Get(A); -- A :=A0
  Get(B); -- B :=B0
  while A /= B loop
    if A < B then
      B := B - A;
    else
      A := A - B;
    end if;
  end loop;
  Put(B);
end;
```

Pour aller plus loin...
Question 2 Transcrire le programme ci-dessus en langage machine en hexadécimal.

Pour aller plus loin...
Ex. 4 : Comparaison processeur/circuit dédié

Question 1 Comparer le nombre de cycle nécessaire à l'implantation processeur pour exécuter cet algorithme pgcd à celui de l'implantation du TD8 en supposant que le processeur met 3 cycles pour exécuter une instruction.
A votre avis, lequel de ces 2 circuits nécessitent le plus de ressources ?
Quel avantage a le processeur sur le circuit dédié ?