



Architecture des ordinateurs S2

(Bachelor en cybersécurité)

Les ressources

- ❖ Thierry VAIRA (<http://tvaira.free.fr/>);
- ❖ Luc DE MEY (<https://courstechinfo.be/index.html>),
- ❖ Fabrice SINCERE (<http://fabrice.sincere.free.fr>);
- ❖ David BOUCHET – EPITA (Cours Architecture des ordinateurs);
- ❖ WikiBooks
- ❖ Internet



Préambule

L'informatique existe depuis 1960.

D'après l'Académie Française en 1966, l'informatique est: « *la science du traitement rationnel, par des machines automatiques, de l'information considérée comme le support des connaissances humaines et sociaux* ».

L'ordinateur (terme de 1955) est une machine capable d'exécuter une suite d'instructions (programme) enregistrées dans une mémoire.

Originellement, l'ordinateur fait référence au grand Ordinateur, c'est-à-dire celui qui ordonne, qui donne l'ordre.

Définition Wikipédia : Un ordinateur est une machine électronique qui fonctionne par la lecture séquentielle d'un ensemble d'instructions, organisées en programmes, qui lui font exécuter des opérations logiques et arithmétiques sur des chiffres binaires.

Le mot « ordinateur » fut introduit par le français François Girard chez IBM France en 1955.

Introduction – Les logiques combinatoire et séquentielle

L'automatique est la science de la commande des systèmes. Les outils de traitement numériques, puissants, en constante évolution permettent d'élaborer des systèmes de commande extrêmement sophistiqués.

Les systèmes sont déterminés selon les deux catégories suivantes:

- ❖ les systèmes délivrant et acceptant les signaux dits analogiques tels que les procédés industriels;
- ❖ les systèmes soumis à des signaux naturellement discrétisés comme les commandes numériques de machine-outil.

Une information (généralement électrique) à deux états (tout ou rien) est transportée sur un support de transmission. Cette information pour être décodée doit avoir un très haut niveau de rapport signal/bruit grâce aux calculatrices numériques. Cela n'est pas le cas si l'information est transportée sous sa forme analogique où l'impact des parasites est important.

Dans la logique combinatoire, les sorties des systèmes sont représentées en fonction des entrées. De plus, les fonctions logiques complexes sont réalisées en combinant des portes logiques de base. C'est la logique combinatoire.

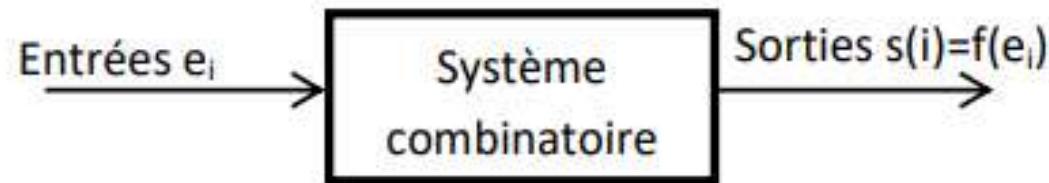
Un système est dit séquentiel quand son comportement est fonction des variables d'entrée mais aussi des états précédents du système (les états du ou des sorties). Le système a une mémoire des états précédents.

Logique combinatoire - Fonctions logiques de base

Définition de la logique combinatoire

Dans un système logique combinatoire (les entrées et les sorties) ne peuvent prendre que 0 ou 1 comme valeur, les sorties ne sont fonctions que des entrées.

L'état de la ou des sortie(s) à un instant donné ne dépend que du circuit et de la valeur des entrées à un instant t.



Exemple & Table de vérité

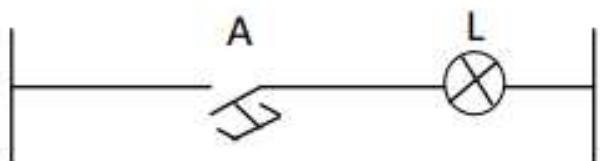


Schéma électrique

Bouton poussoir A	Lampe (L)
0	0
1	1

Logique combinatoire - Fonctions logiques de base

Fonctions logiques Booléennes généralités

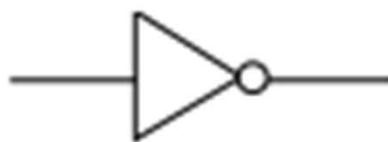
L'outil mathématique permettant de décrire et de simplifier les systèmes combinatoires est l'Algèbre de Boole.

Une fonction booléenne qui prend en arguments n booléens retourne un booléen.

Grâce à la combinaison des trois fonctions logiques de base: NON, OU, ET, nous allons pouvoir décrire chacune des sorties en fonction des entrées.

Rappel des symboles des portes logiques de base :

NON



ET (AND)



OU (OR)



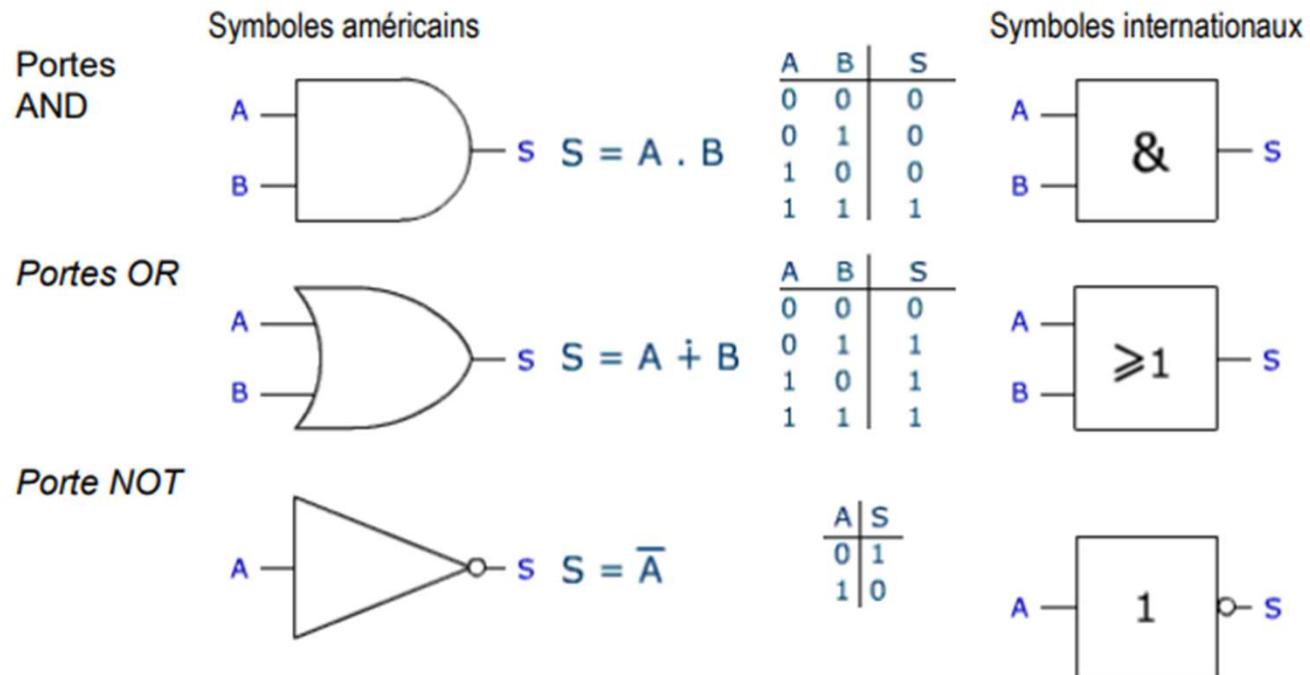
Logique combinatoire - Fonctions logiques de base

Portes logiques de base

Nous avons jusqu'ici utilisé un bouton poussoir et une lampe pour illustrer le fonctionnement des opérateurs logiques.

En électronique digitale, les opérations logiques sont effectuées grâce aux portes logiques. Ce sont des circuits qui combinent les signaux logiques présentés à leurs entrées sous forme de tensions.

Nous aurons par exemple 5 V pour représenter l'état logique 1 et 0 V pour représenter l'état 0.



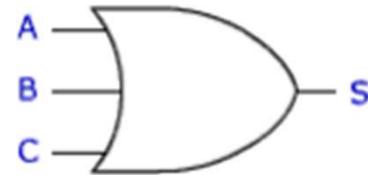
Logique combinatoire - Fonctions logiques de base

Portes logiques de base

Les portes logiques sont des fonctions booléennes élémentaires ; elles disposent d'entrées (à gauche sur les dessins) et d'une sortie (droite). Des signaux arrivent sur les entrées (0 ou 1) et un signal est produit sur la sortie.

Les tables de vérité donnent la valeur de la sortie pour chacune des portes en fonction de la valeur de entrées.

Le nombre d'entrées des fonctions AND et OR n'est pas limité. Voici par exemple une représentation de ces portes avec trois entrées :

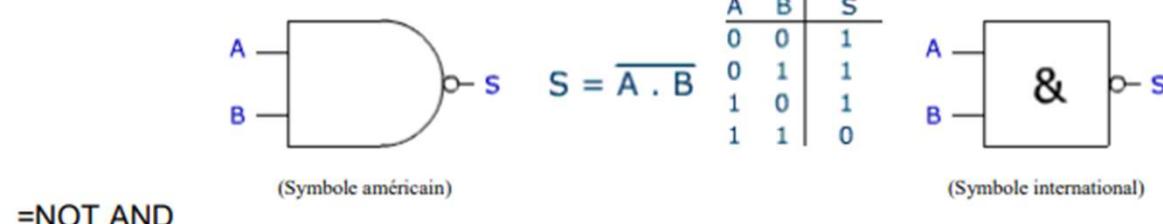


Logique combinatoire - Fonctions logiques de base

Combinaisons de portes logiques de base

Ces trois fonctions logiques de base peuvent être combinées pour réaliser des opérations plus élaborées en interconnectant les entrées et les sorties des portes logiques.

La porte NAND (Non ET)



=NOT AND



Porte NOR (Non OU)



= NOT OR



Logique combinatoire - Fonctions logiques de base

Combinaisons de portes logiques de base

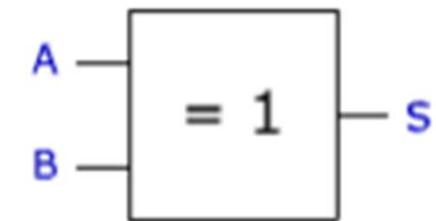
Porte XOR à deux entrées



(Symbole américain)

$$S = A \oplus B$$

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0



(Symbole international)

La fonction "OU Exclusif" est en principe d'une fonction de deux variables :

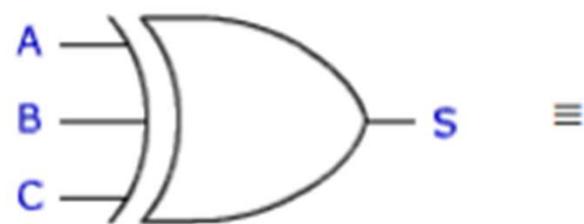
$$S = A \oplus B$$

La sortie est égale 1 si une seule des deux entrées vaut 1, d'où son appellation « Ou exclusif ».

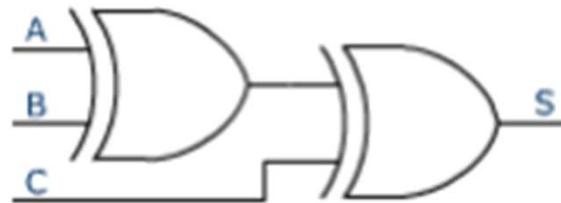
Logique combinatoire - Fonctions logiques de base

Combinaisons de portes logiques de base

Porte XOR à plusieurs entrées



=



A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Pour calculer le résultat de $S = A \oplus B \oplus C$, on doit pouvoir faire d'abord l'opération entre deux termes, puis refaire un ou exclusif entre le résultat obtenu et le troisième terme.

Ce qui se traduit par $S = (A \oplus B) \oplus C$ ou par $S = A \oplus (B \oplus C)$

On constate que l'appellation "Ou exclusif" n'est plus adéquate puisqu'avec trois variables, le résultat vaut 1 si une seule entrée ou toutes les trois valent 1.

Le résultat est en fin de compte un bit de parité. Il vaut 1 si le nombre d'entrées à 1 est impair.

Logique combinatoire - Fonctions logiques de base

Combinaisons de portes logiques de base

L'inverse de la porte XOR à deux entrées



A	B	$S = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Le résultat vaut 1 si les deux entrées sont identiques. Cette porte teste donc l'équivalence des deux entrées. Certains appellent cette fonction logique, "fonction équivalence", d'autres l'appelle "XNOR"

Logique combinatoire - Circuits combinatoires

Définition Circuits combinatoires

Un circuit combinatoire :

- ❖ est défini par un ensemble de portes logiques reliées les unes aux autres;
- ❖ les sorties des portes sont reliées aux entrées d'autres portes logiques (définissant une orientation des connexions);
- ❖ en suivant l'orientation des connections, il est impossible qu'en partant de la sortie d'une porte, nous revenions à l'une des ses entrées (graphe acyclique).

Un circuit combinatoire peut être vu comme une porte logique (à plusieurs sorties).

Pour simplifier l'expression d'une fonction booléenne, nous pouvons tout simplement utiliser les théorèmes et les propriétés de l'algèbre de Boole, afin de passer d'une expression à une autre plus simple.

Une méthode très efficace est l'utilisation des tableaux de Karnaugh, qui permet une simplification visuelle. La méthode de simplification par les tables de KARNAUGH sera présentée dans la suite de ce cours.

Logique combinatoire - Circuits combinatoires

Algèbre de BOOLE - Théorèmes fondamentaux

Opérateur OU	$A + A = A$
	$A + 0 = A$
	$A + 1 = 1$
	$A + \bar{A} = 1$
Opérateur ET	$A \cdot A = A$
	$A \cdot 1 = A$
	$A \cdot 0 = 0$
	$A \cdot \bar{A} = 0$

Logique combinatoire - Circuits combinatoires

Algèbre de BOOLE - Propriétés & Théorème

La commutativité	$A + B = B + A$
	$A \cdot B = B \cdot A$
L'associativité	$A + (B + C) = (A + B) + C = A + B + C$
	$A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C$
La distributivité	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ $A + (B \cdot C) = (A + B) \cdot (A + C)$

Théorème d'absorption	$A + (A \cdot B) = A$
	$A \cdot (A + B) = A$
Théorème d'allongement	$A + (\bar{A} \cdot B) = A + B$
	$A \cdot (\bar{A} + B) = A \cdot B$

Logique combinatoire - Circuits combinatoires

Algèbre de BOOLE - Théorème de De Morgan

C'est une des propriétés les plus importantes des fonctions logiques. Il est défini par les deux relations suivantes :

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

Ainsi le complément d'une fonction logique sera obtenu en remplaçant les variables par leur complément, les signes + par des • et les signes • par des +.

NB : $\bar{\bar{A}} = A$. Les portes NAND et NOR sont des portes universelles, car elles permettent de réaliser toutes les opérations logiques élémentaires.

Logique combinatoire - Circuits combinatoires

Circuits combinatoires – Formes canoniques des fonctions logiques et leurs simplifications

Il existe deux méthodes pour exprimer une fonction logique :

- ❖ soit donner directement son équation logique;
- ❖ soit utiliser une table de vérité.

Pour un nombre fini N de variables d'entrée correspond 2^N combinaisons possibles des variables d'entrée. Par définition la table de vérité d'une fonction c'est un tableau qui représente l'état de sortie en fonction de toutes les combinaisons des variables d'entrée. Voir l'exemple ci-après.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

→ $\bar{A}\bar{B}C$
→ $\bar{A}B\bar{C}$
→ $A\bar{B}\bar{C}$
→ $AB\bar{C}$

Logique combinatoire - Circuits combinatoires

Circuits combinatoires – Formes canoniques des fonctions logiques et leurs simplifications – Maxterme & Minterme

L'expression précédente de F possède la forme d'une somme de produits de variables, appelée somme de produits ou somme de monômes. Elle représente la première forme canonique.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$F = \sum 1, 2, 4, 6 \rightarrow \bar{F} = \sum 0, 3, 5, 7$$

On complémente à $2^{n-1} = 2^{3-1} = 7$ (avec n nombre de variables, dans notre exemple $n = 3$);

$$\text{On obtient : } F = \prod 7, 4, 2, 0$$

$$\text{d'où } F = (A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C})$$

Cette forme est un produit de sommes de variables, appelé produit de somme ou produit de Maxterme, elle représente la deuxième forme canonique $F = \prod \text{Maxterme}$.

$$F = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$$

Logique combinatoire - Circuits combinatoires

Circuits combinatoires – Formes canoniques des fonctions logiques et leurs simplifications – Maxterme & Minterme

Synthèse

Une fonction booléenne peut être exprimée sous quatre formes canoniques :

- ❖ la première forme canonique : sommes de produits : $\Sigma(\Pi)$
- ❖ la deuxième forme canonique : produits de sommes : $\Pi(\Sigma)$
- ❖ la troisième forme canonique : l'écriture avec l'opérateur NAND uniquement
- ❖ la quatrième forme canonique : l'écriture avec l'opérateur NOR uniquement

NB: l'objectif de ces méthodes de simplification vise à réduire le nombre de portes logiques utilisé dans les fonctions logiques.

Par ailleurs, le fait d'écrire les fonctions qu'avec des portes NAND ou NOR permet de trouver sur le marché des circuits intégrés qui ne comportent que des portes NAND ou NOR.

Logique combinatoire - Circuits combinatoires

Exercices d'application à faire en séance ou devoir maison

Démontrez analytiquement par l'algèbre de Boole que :

1. $a + ab = a$
2. $\bar{a}b + a = a + b$
3. $ac + \bar{a}b + bc = ac + \bar{a}b$
4. $\bar{a}bc + ac + a\bar{b}\bar{c} + \bar{a}\bar{b} = c + \bar{b}$
5. $(\bar{a} + b).(a + b + d).\bar{d} = bd$

Logique combinatoire – Circuits combinatoires

Simplification de fonctions par les tables Karnaugh

La méthode est l'une des différentes méthodes permettant d'exprimer une fonction booléenne sous sa forme réduite. La méthode repose sur la recherche des simplifications possibles entre les différents termes d'une expression. Elle est basée sur les termes ou les cases adjacentes.

Équations des fonctions logiques

$$\text{Fonction } 0 = S_0 = a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c = a \cdot \bar{b}$$

$$\text{Fonction } 1 = S_1 = \bar{a} \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + abc = a \cdot b \cdot c + \bar{b} \cdot \bar{c}$$

S_0		bc=00	bc=01	bc=11	bc=10
	a = 0	0	0	0	0
	a = 1	1	1	0	0

S_1		bc=00	bc=01	bc=11	bc=10
	a = 0	1	0	0	0
	a = 1	1	0	1	0

Logique combinatoire – Circuits combinatoires

Simplification par les tables Karnaugh - Méthode

La méthode des tables de Karnaugh est très efficace pour simplifier les fonctions booléennes. **En respectant la méthode suivante:**

- 1) **Regroupement de tous les 1, par groupe de 2, 4, 8, ou 16 les plus gros et les moins nombreux possibles.** Chaque case avec un 1 peut appartenir à plusieurs regroupements;
- 2) **Commencer par les 1 les plus isolés;**
- 3) Parfois, il est plus facile de calculer la fonction \bar{F} en regroupant les 0 puis en revenant à F en complémentant;
- 4) Lorsque la valeur de la fonction logique n'est pas précisée pour certaines combinaisons des variables logiques, on écrit un φ dans la case correspondante de la table de Karnaugh. On choisira pour φ la valeur 0 ou 1 pour simplifier au maximum;
- 5) **Pour la réalisation de la table de Karnaugh, appliquez le binaire réfléchi ou code GRAY pour les combinaisons des variables.**

Logique combinatoire – Circuits combinatoires

Simplification par les tables Karnaugh - Remarques

- ❖ Dans le tableau de Karnaugh, les cases sur le bord supérieur sont adjacentes avec ceux du bord inférieur de même pour le bord gauche avec le bord droit.
- ❖ Une case isolée contenant un 1 peut être commune à plusieurs groupements, càd : on peut utiliser une case plus d'une fois car : $X=X+X+\dots+X$
- ❖ Toutes les cases contenant 1 doivent être entourées.
- ❖ Dans un tableau de Karnaugh à 16 cases ou à 4 variables :
 - ✓ une case isolée correspond à un terme de 4 variables,
 - ✓ une boucle de 2 cases correspond à un terme de 3 variables,
 - ✓ une boucle de 4 cases correspond à un terme de 2 variables,
 - ✓ une boucle de 8 cases correspond à un terme d'une seule variable.

Logique combinatoire – Circuits combinatoires

Simplification par les tables Karnaugh

Exercices d'application à faire en séance ou devoir maison

$$1. F1 = ab\bar{c} + \bar{a}\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c$$

$$2. F2 = abc + \bar{a}bc + \bar{a}\bar{b}c + ab\bar{c}$$

$$3. F3 = \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}d + a\bar{b}c\bar{d} + \bar{a}\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + \bar{a}bcd + a\bar{b}\bar{c}\bar{d}$$

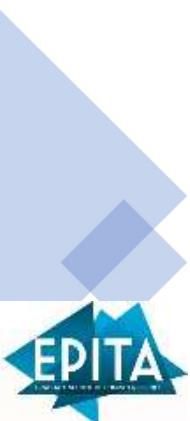
Logique combinatoire – Circuits combinatoires

Additionneurs

Un additionneur calcule la somme de deux quantités A et B codées en binaire.

L'objectif est d'additionner des mots de plusieurs bits. Pour cela, on utilise le même principe que pour l'addition posée en système décimal vue à l'école élémentaire : on calcule la somme des deux chiffres de droite, on pose l'unité du résultat et on note la retenue.

Dans un second temps, on calcule la somme de cette retenue et des deux chiffres des dizaines, on pose l'unité du résultat et on note la retenue. Et ainsi de suite...



Logique combinatoire – Circuits combinatoires

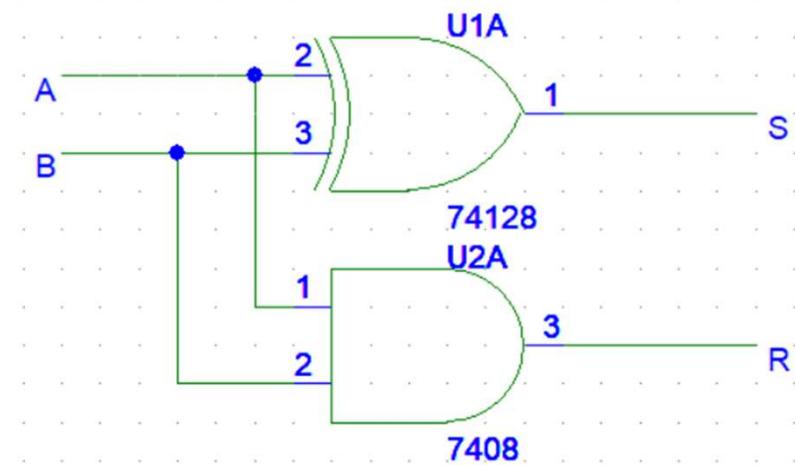
Demi-Additionneur – Définition

Un circuit combinatoire est un circuit logique où chacune des sorties est une fonction logique des entrées.

Table de vérité & Logigramme

S= somme en sortie R= Retenue en sortie

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



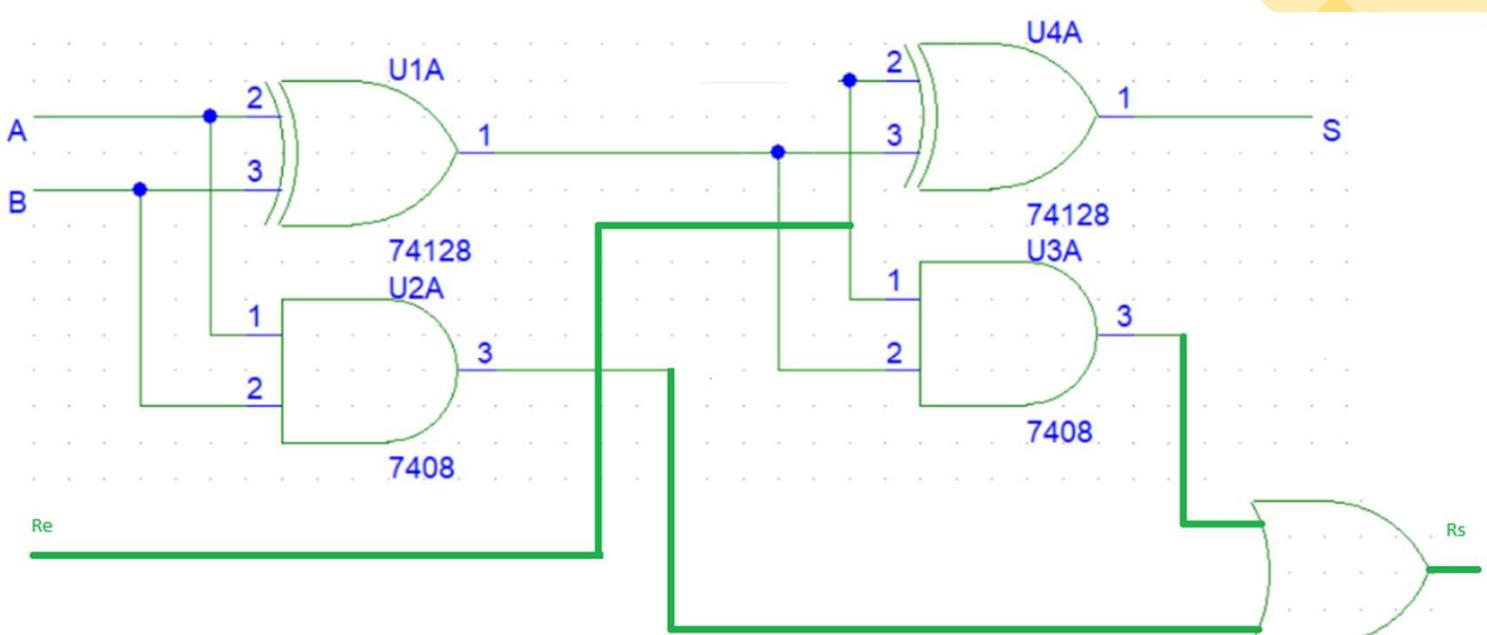
Logique combinatoire – Circuits combinatoires

Additionneur complet

Table de vérité & Logigramme

Rs= Retenue en sortie et Re= Retenue en entrée

A	B	Re	S	Rs
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Logique combinatoire – Circuits combinatoires

Soustracteur

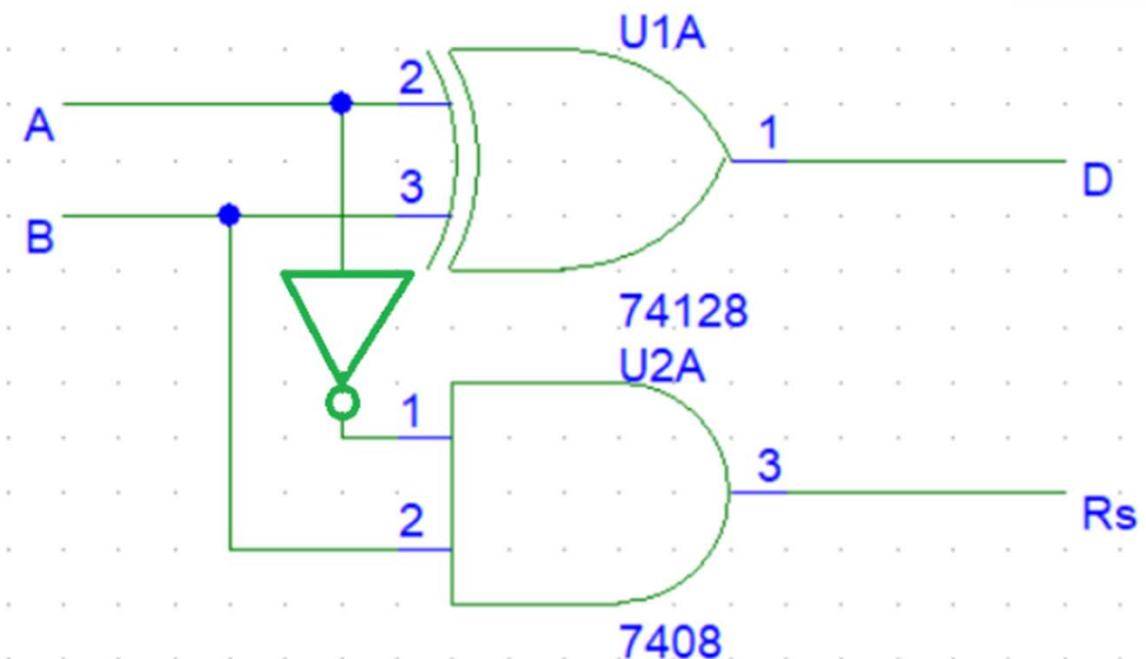
C'est le même principe que l'additionneur, le soustracteur fait la soustraction deux nombres binaires comme le ferait le complémenté vrai (CA2).

Table de vérité & Logigramme

D= différence

R= retenue

A	B	R	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0



Logique combinatoire – Circuits combinatoires

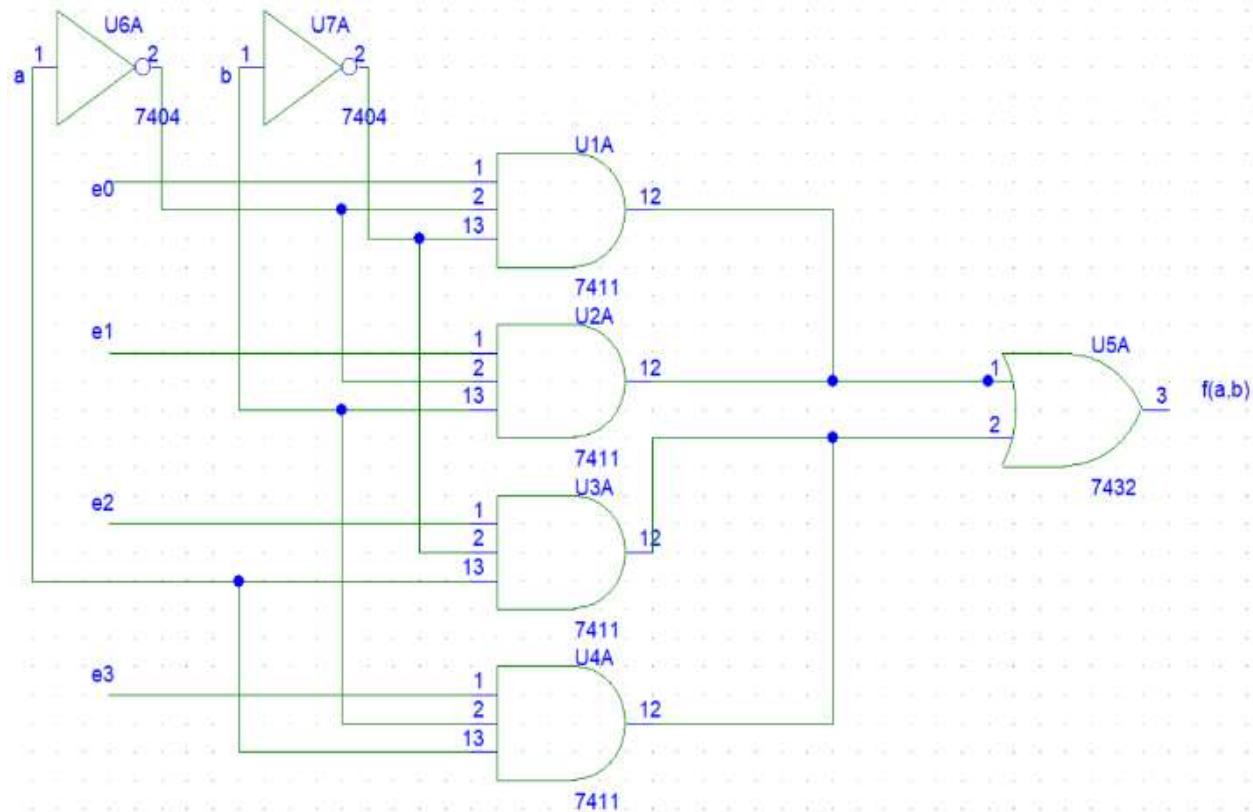
Multiplexeur (Mux) – (1/2)

C'est un système combinatoire qui fait la fonction à **n variables qui correspondent aux n lignes de sélection**, un multiplexeur est composé de :

- ❖ 2^n entrées
- ❖ une seule sortie
- ❖ n lignes de sélection

Logigramme à 2 variables

$$f(a,b) = \bar{a} \cdot \bar{b} \cdot e_0 + \bar{a} \cdot \bar{b} \cdot e_1 + a \cdot \bar{b} \cdot e_2 + a \cdot b \cdot e_3$$



Logique combinatoire – Circuits combinatoires

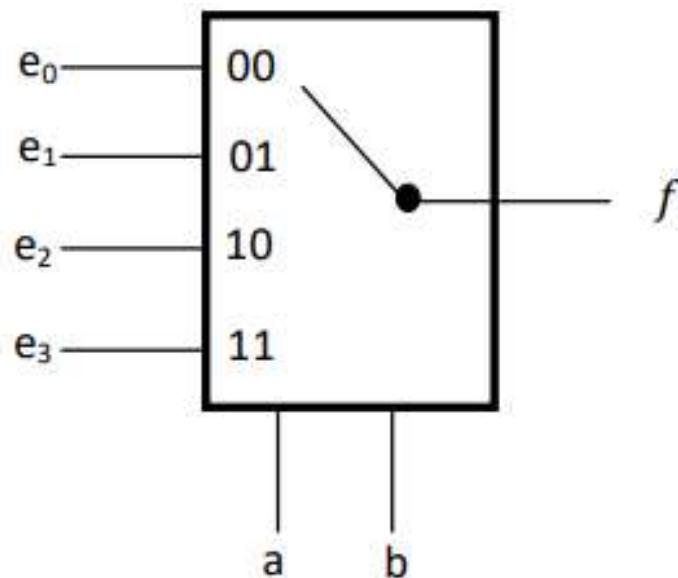
Multiplexeur (Mux) – (2/2)

Logigramme à 2 variables

a et b sont appelées ligne de commande

- ❖ e_0, e_1, e_2, e_3 sont appelées ligne de données;
- ❖ f est le résultat;

Les lignes de commande déterminent quelle entrée se retrouve en sortie. De ce fait, on dira qu'un multiplexeur **est un sélecteur de données**.



Logique combinatoire – Circuits combinatoires

Encodeur prioritaire (1/2)

C'est un circuit à m entrées et n sorties. Les sorties délivrent le code de l'entrée active si il n'y en a qu'une ou de l'entrée prioritaire si il y en a plusieurs.

Si le code est le code binaire standard alors on a $m=2^n$ et l'encodeur est dit binaire. Il existe bien entendu des encodeurs pour les codes BCD, GRAY (Binaire réfléchi), ASCII ...

Dans le cas du code ASCII l'encodeur est utilisé pour coder les touches d'un clavier, c'est à dire pour générer le code ASCII associé au caractère ou à la fonction de la touche enfoncee.

Le fonctionnement d'un encodeur prioritaire à quatre entrées est décrit par la table de vérité suivante.

Les entrées sont actives au niveau haut (1), lorsque plusieurs entrées sont actives, seul le code correspondant à l'entrée d'indice le plus élevé est présent sur les sorties a_1a_0 .

Logique combinatoire – Circuits combinatoires

Encodeur prioritaire (2/2)

Dans cet exemple l'entrée E_3 a la plus forte priorité.

E_0	E_1	E_2	E_3	a_1	a_0
1	0	0	0	0	0
X	1	0	0	0	1
X	1	1	0	1	0
X	X	X	1	1	1

Logique combinatoire – Circuits combinatoires

Le décodeur-démultiplexeur (1/2)

Un démultiplexeur est un aiguilleur à une entrée de donnée, n entrées d'adresse et m sorties. La valeur de l'entrée se retrouve sur la sortie dont le numéro est codé par l'adresse.

Dans cette fonction le circuit joue le rôle inverse du multiplexeur.

Ces circuits sont aussi des décodeurs : si l'entrée est maintenue active, le numéro de la sortie reflète le code de l'adresse. Dans le cas d'un code binaire on a 2^n Mais il existe aussi des décodeurs pour les codes BCD, GRAY , ...

Un démultiplexeur dont l'entrée est maintenue active au niveau 1 peut servir de générateur de fonctions logiques.

En effet, puisque l'on retrouve sur les sorties toutes les combinaisons possibles des entrées d'adresse, une porte OU suffit pour fabriquer une fonction logique sous sa première forme canonique.

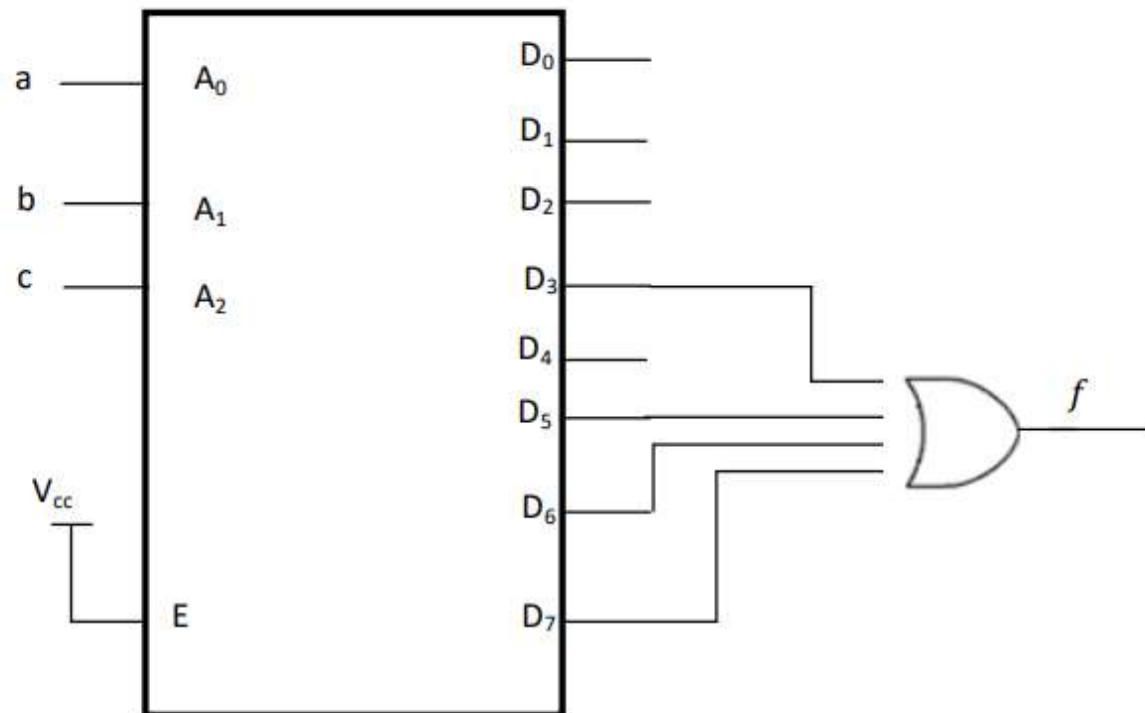
Logique combinatoire – Circuits combinatoires

Le décodeur-démultiplexeur (2/2)

Soit la fonction logique de trois variables.

$$f = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$$

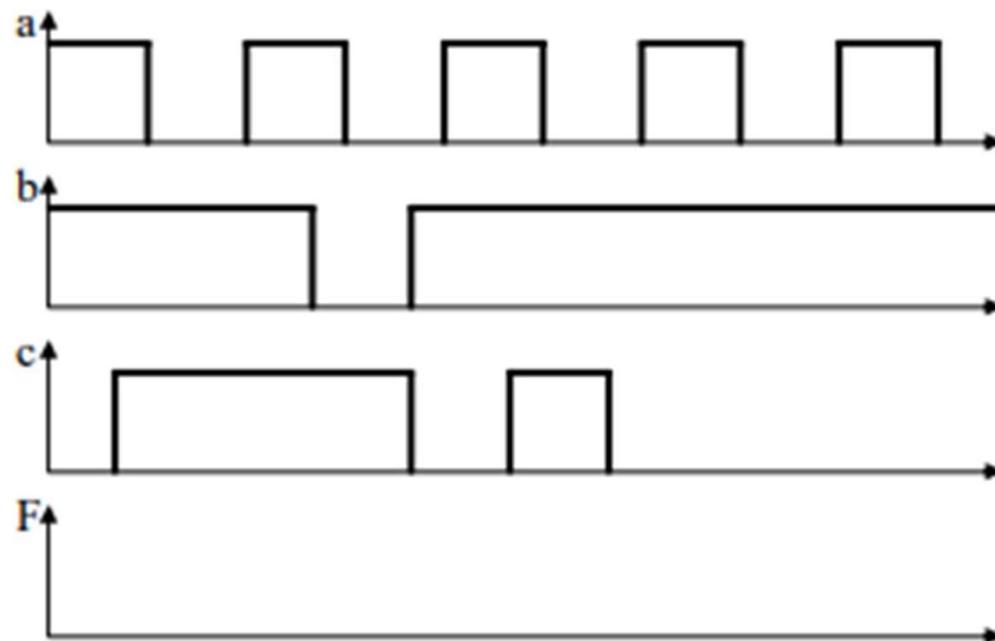
Le codage des sorties avec l'adresse **(c b a)** correspond aux sorties D6, D5 , D3 et D7. D'où le schéma suivant :



Logique combinatoire – Circuits combinatoires

Exercices d'application à faire en séance ou devoir maison

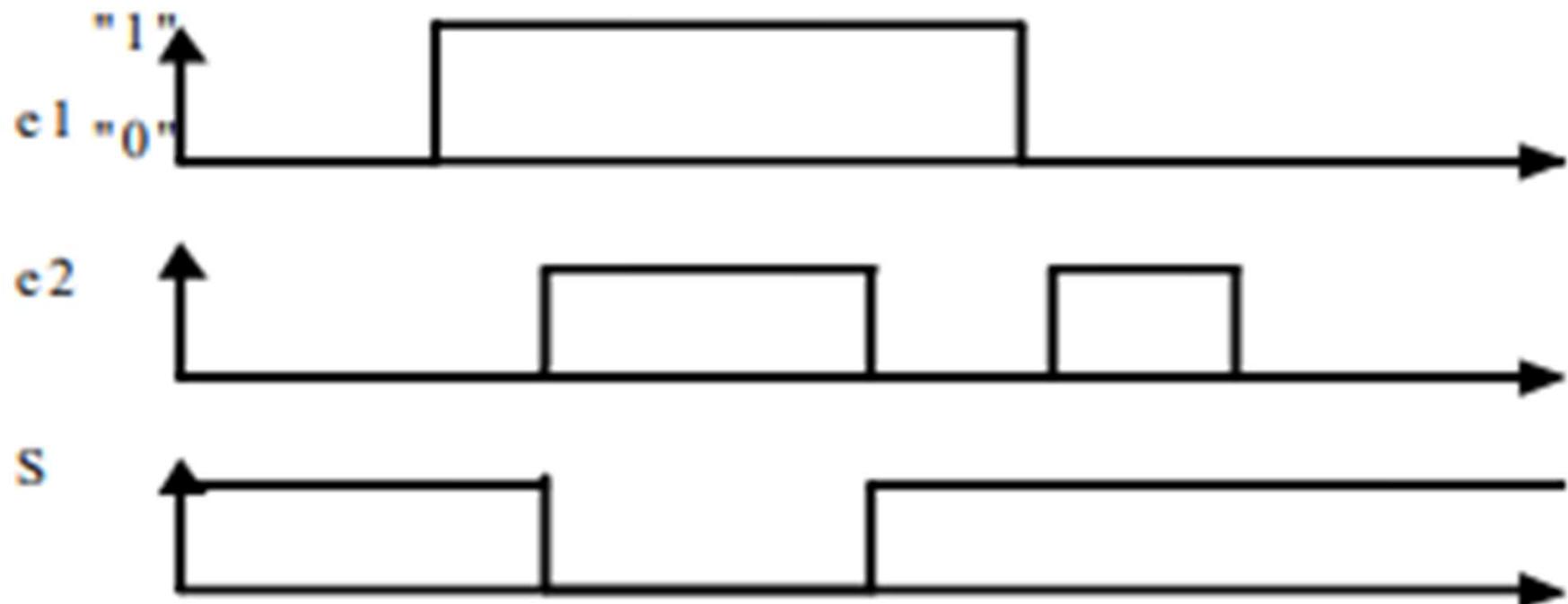
1. Etablir le logigramme de $S = \overline{ab} + \overline{a}bc$ avec des portes à 2 entrées
2. Représenter le logigramme de $S = abc + d$ avec n'importe quelles portes puis avec des portes à 2 entrées
3. Soit la fonction suivante $F = a\overline{b}c + \overline{a}b$
 - a) Etablir la table de vérité
 - b) Compléter le chronogramme



Logique combinatoire – Circuits combinatoires

Exercices d'application à faire en séance ou devoir maison

1. A partir du chronogramme ci-dessous
 - a) Déterminer l'équation de la sortie S par rapport aux entrées e1 et e2
 - b) Identifier l'opérateur logique correspondant
 - c) Réaliser cette fonction logique avec des opérateurs NAND à 2 entrées.



Logique séquentielle - Bascules

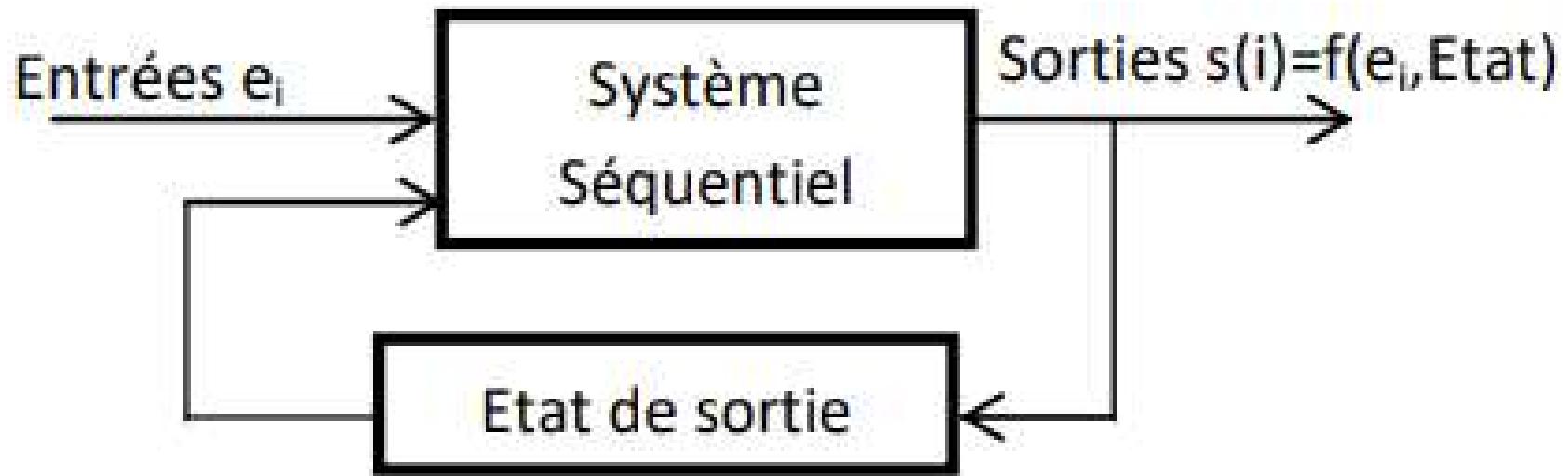
Introduction à la Logique Séquentielle

Dans un circuit séquentiel, les sorties sont des fonctions logiques des entrées et de l'état antérieur des sorties.

Cette logique se distingue de la logique combinatoire par un bouclage de certaines sorties vers les entrées, ce qui génère des éléments de mémorisation.

Les bascules et verrous logiques ont la propriété de mémoriser une information élémentaire (bit).

On peut représenter un système séquentiel par le schéma suivant:



Logique séquentielle - Bascules

Les Bascules - Définition

Les bascules sont des éléments de base de la logique séquentielle. Un peu comme l'étaient les portes logiques en logique combinatoire.

Les bascules permettent la réalisation de nombreux systèmes comme: des compteurs, des registres, des mémoires, etc...d'où leur importance.

De manière générale, une bascule se caractérise par:

- ❖ deux états de sortie stables. Deux états dans lesquels la bascule peut se maintenir indéfiniment sans actions extérieures;
- ❖ des entrées de commande permettant de passer au choix d'un état à un autre.

Logique séquentielle – Bascules

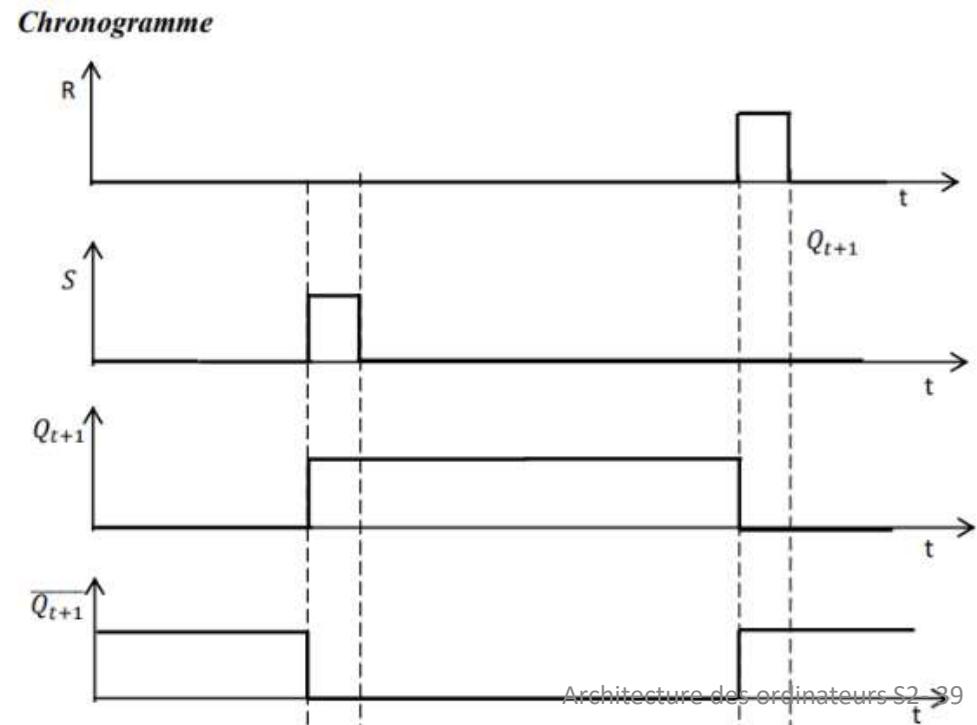
Exemple d'un système électromécanique

Le fonctionnement de ce système est le suivant: l'action sur un Bouton Poussoir A (BpA) provoque le démarrage du moteur et à la libération du bouton, le moteur reste en marche.

Une action sur BpB provoque l'arrêt du moteur et à sa libération, le moteur reste en arrêt.

Pour l'état des entrées Bp A=0 et Bp B =0 la sortie du moteur M prend la valeur 0 et aussi 1. Finalement, on peut dire que le système crée dans un état dans une mémoire qui garde l'état précédent du système.

BpA	BpB	Moteur
0	0	0
1	0	1
0	0	1
0	1	0
0	0	0



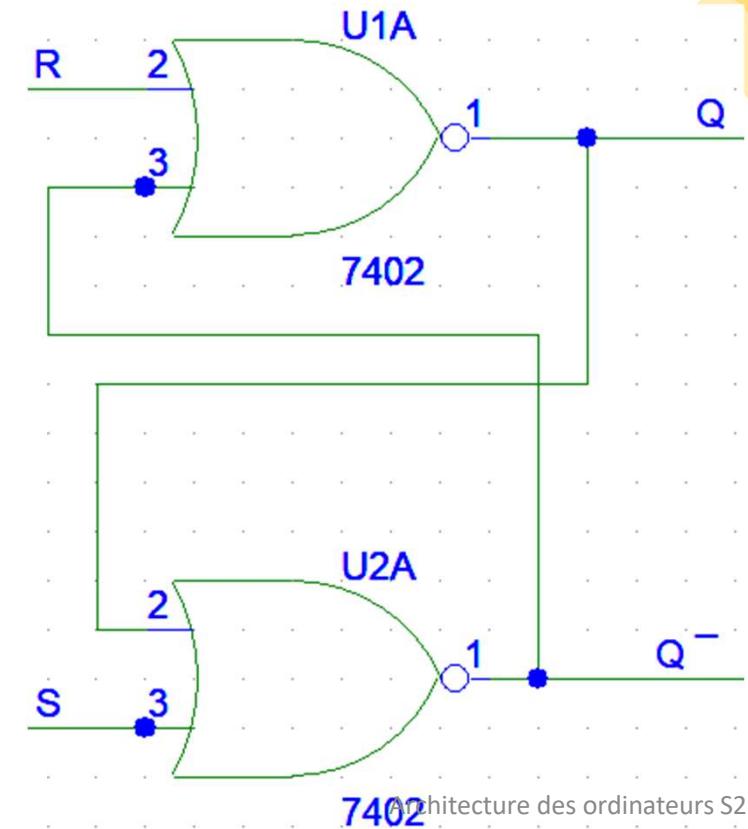
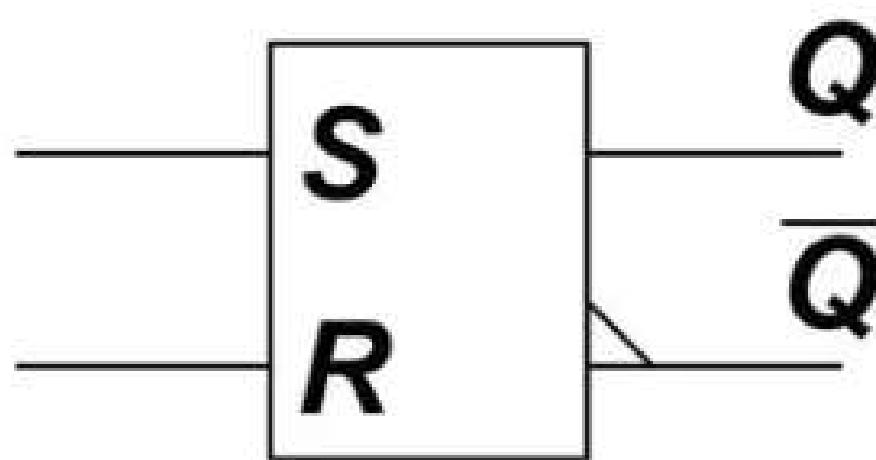
Logique séquentielle – Bascules

Bascule RS (1/8)

C'est la plus simple des bascules. Elle est réalisée grâce à deux portes NOR ou deux portes NAND. R et S sont les entrées de la bascule. Respectivement, \bar{Q} et Q sont les sorties complémentées de la bascule RS.

R = RESET / S= SET

Symbole & Logigramme



Logique séquentielle – Bascules

Bascule RS (2/8)

Analyse de fonctionnement

La bascule constituée sur la base de portes NOR, dès que l'une de ses entrées est au niveau 0, les sorties Q et \bar{Q} complémentaires. **On s'interdit d'avoir en entrée RS = 11.**

En effet, avec la table vérité de la fonction NOR, on remarque que:

- ❖ la combinaison **SR = 10** impose $Q \bar{Q} = 10$, c'est le fonctionnement SET.
- ❖ la combinaison **SR = 01** impose $Q \bar{Q} = 01$, c'est la fonction RESET.

La situation la plus intéressante est **S=R=0**. Cette situation autorise : $Q=1 \& \bar{Q} = 0$ ou $Q=0 \& \bar{Q} = 1$. Ces situations correspondent à deux situations stables de la bascule.

La combinaison présente sur les sorties dépend de la combinaison des entrées qui a précédé la situation **S=R=0**.

SR	$Q\bar{Q}$	Fonction
00	$Q\bar{Q}$	Mémoire
01	01	Reset
10	10	Set
11	00	Interdit

Logique séquentielle – Bascules

Bascule RS (3/8)

	<ul style="list-style-type: none">❖ (RS) \rightarrow (00) la sortie d'une fonction NON OU est à l'état 1 si et seulement si toutes les entrées sont à l'état 0.❖ Les niveaux logiques en sortie sont inchangés : fonction mémoire.
	<ul style="list-style-type: none">❖ (RS) = 01 \Rightarrow Q=1 \rightarrow c'est la fonction SET (mise à 1 de la sortie Q).
	<ul style="list-style-type: none">❖ (RS) = (10) \Rightarrow Q = 0 : fonction RESET (mise à 0 de la sortie Q).
	<ul style="list-style-type: none">❖ Fonction mémoire❖ Les niveaux logiques en sortie sont inchangés : fonction mémoire.

Logique séquentielle – Bascules

Bascule RS (4/8)

Simplification par Karnaugh

R	S	Q _n	Q _{n+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	X
1	1	1	X

Tableau de Karnaugh

		S Q _n			
		00	01	11	10
R	0	0	1	1	1
	1	0	0	X	X

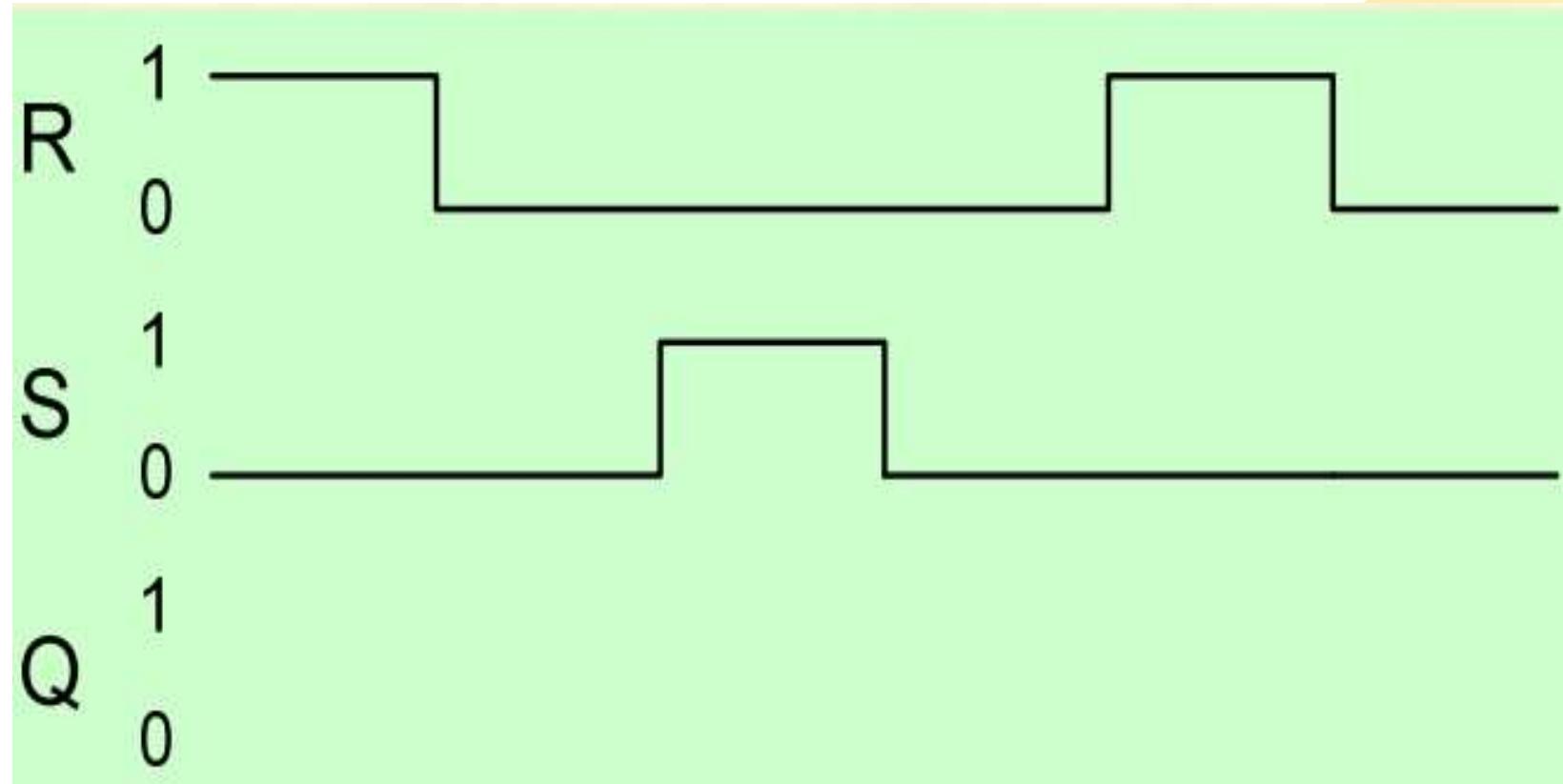
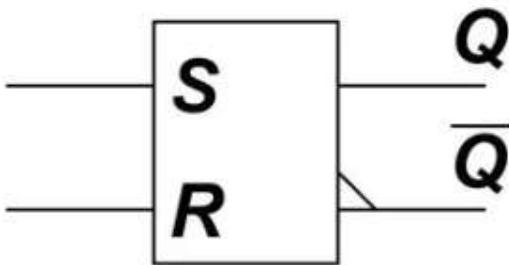
Equation logique :

$$Q_{n+1} = \overline{R} \cdot Q_n + S$$

Logique séquentielle – Bascules

Bascule RS (5/8)

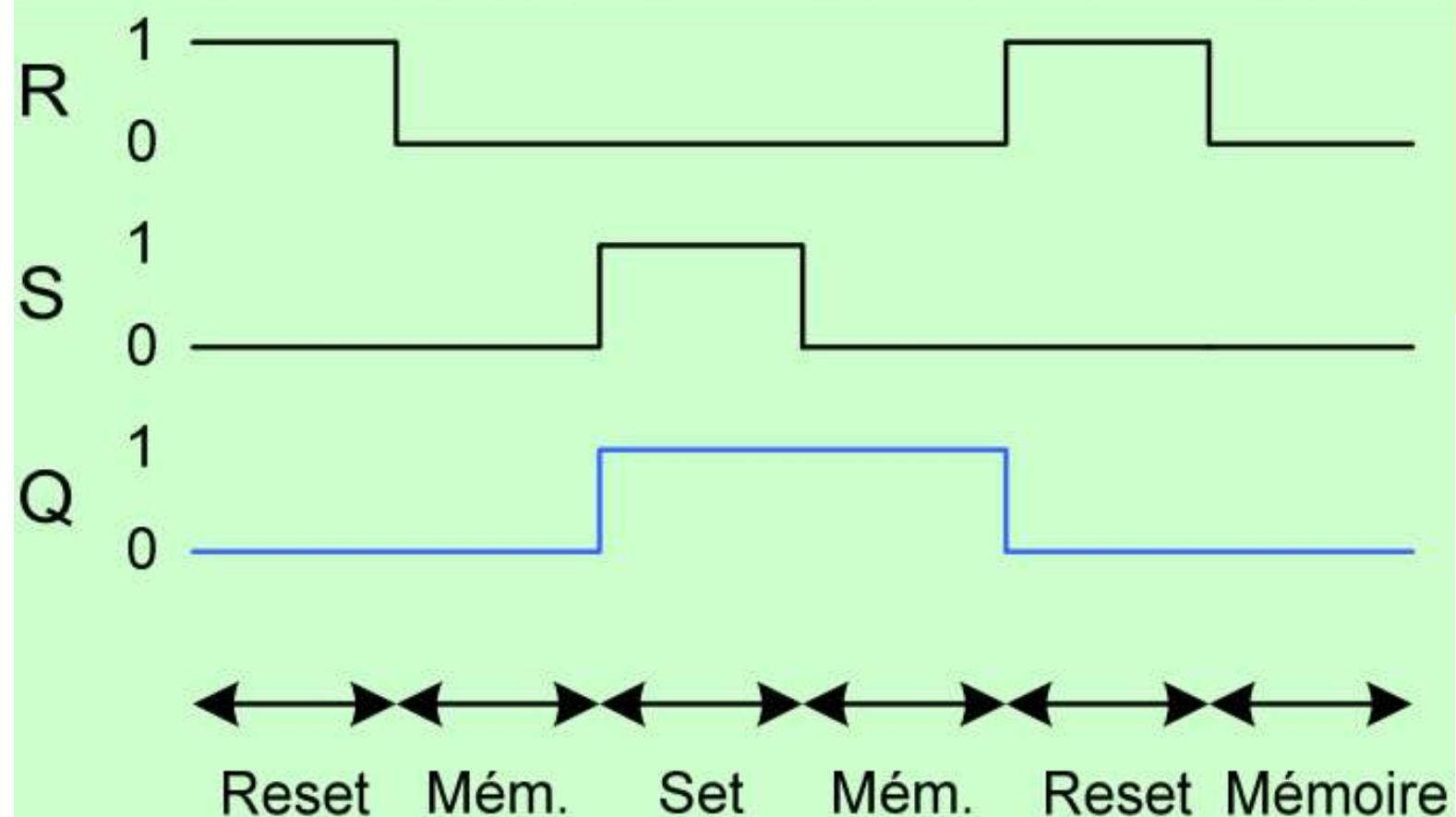
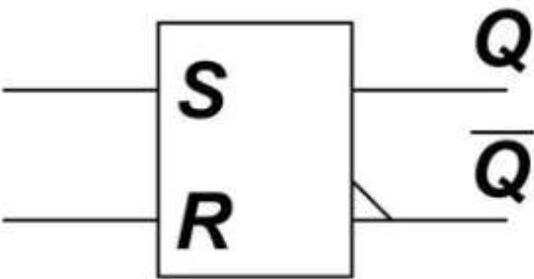
Symbol & Chronogramme (A compléter)



Logique séquentielle – Bascules

Bascule RS (6/8)

Symbol & Chronogramme (Réponse)

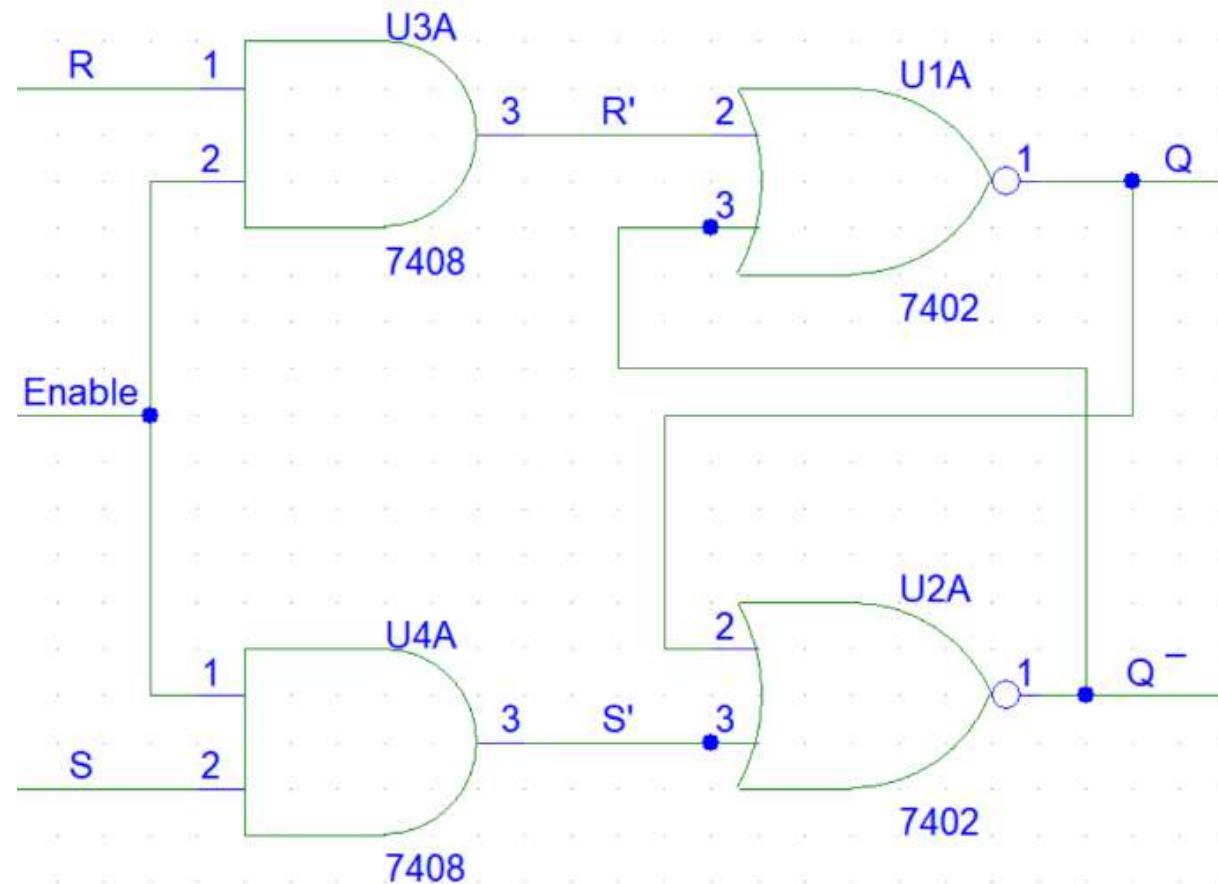


Logique séquentielle – Bascules

Bascule RS avec validation (RS Latch) - (7/8)

C'est le même principe qu'une bascule RS mais avec une validation **Enable (E)**. C'est un genre de verrouillage (Latch en anglais).

Logigramme



Logique séquentielle – Bascules

Bascule RS avec validation (RS Latch) – Fonctionnement (8/8)

Raisonnons en considérant les deux valeurs possibles du signal **Enable de validation**.

E=0 => S'=R'=0

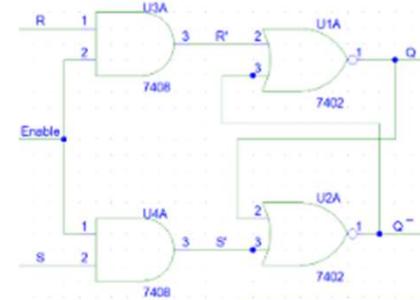
La bascule RS est en mode mémoire. Elle restera dans cet état quelles que soient les valeurs présentes sur les entrées R et S.

E=1 => on distingue 2 cas :

- ❖ si $R=1, S=0$, alors $R'=1$ et $S'=0$ cela porte en sortie $Q = 0$ et $\bar{Q} = 1$. C'est le mode RESET.
- ❖ si $R=0, S=1$, alors $R'=0$ et $S'=1$ cela porte en sortie $Q = 1$ et $\bar{Q} = 0$. C'est le mode SET.

On voit que le fonctionnement de la bascule RS avec la validation est identique à celui de la bascule RS simple lorsque E=1.

Cependant, lorsque E=0, la bascule est bloquée dans l'état imposé par les valeurs de R et S au moment où E passe à 0. Dans cette situation, les variations des entrées R et S n'impactent plus les sorties.



Logique séquentielle – Bascules

Bascule JK

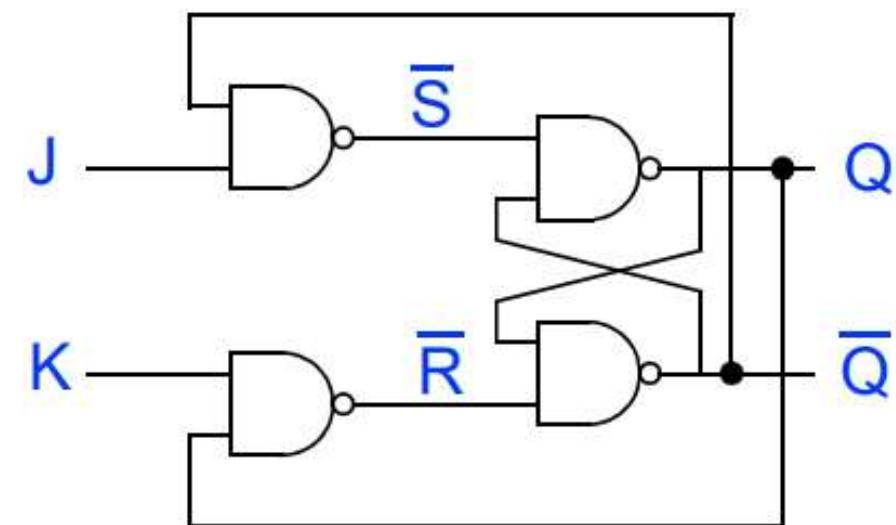
Cette bascule JK est à base d'une bascule RS et autorise RS=11 et aussi telle que S→J et R→K. Limitations des bascules JK :

- ❖ Fonctionnement asynchrone: la sortie réagit "immédiatement" à l'entrée;
- ❖ Les entrées doivent donc rester stables.

Table de vérité & Equation & Logigramme

J	K	Q_n	\bar{Q}_n	Fonction
0	1	0	1	
1	0	1	0	
0	0	Q_{n-1}	\bar{Q}_{n-1}	Etat Mémoire
1	1	\bar{Q}_{n-1}	Q_{n-1}	Etat Mémoire

$$Q_{n+1} = J \bar{Q}_n + \bar{K} Q_n$$



Logique séquentielle – Bascules

Bascule D (1/3)

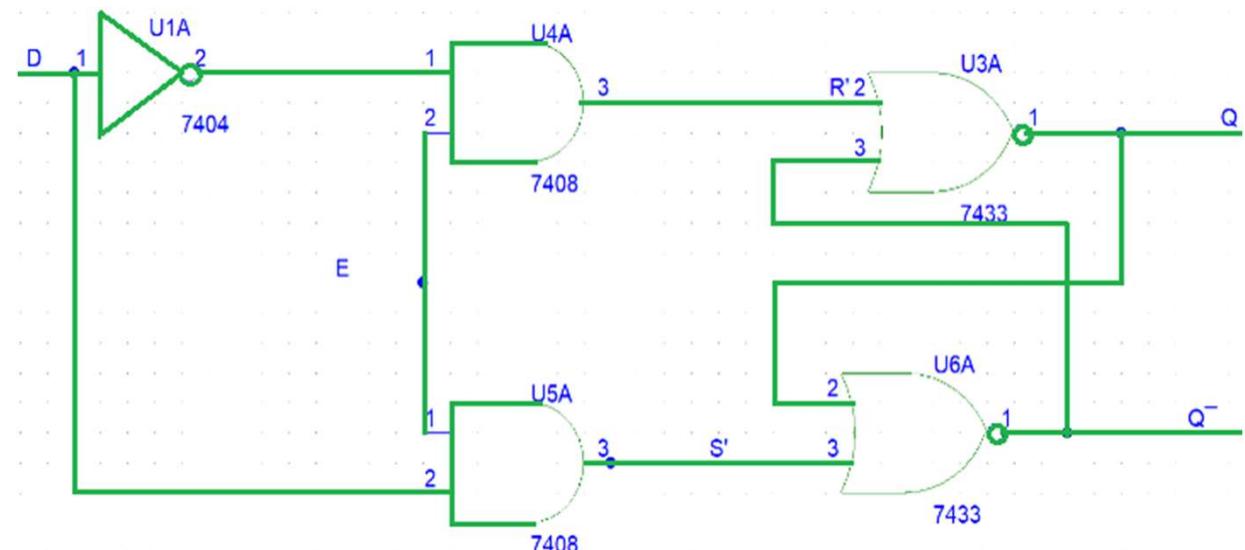
La bascule D (pour Data) est une bascule comportant uniquement une entrée de données : D. La bascule D est en réalité une bascule RS avec une validation sur laquelle on a ajouté un inverseur.

L'inverseur entraîne entraîne $R' = \bar{S}'$ lorsque $E=1$ si bien que dans cette situation, la bascule n'est jamais en mode mémoire.

Table de vérité & Logigramme

L'équation de la bascule D s'obtient à partir de l'équation de la bascule RS en écrivant $R' = \bar{D}$ et $S' = D$

D	Q	\bar{Q}	Fonction
0	0	1	Mise à 0
1	1	0	Mise à 1



Logique séquentielle – Bascules

Bascule D – Fonctionnement & Equation – (2/3)

Comme pour la bascule RS avec validation, nous pouvons raisonner sur les deux valeurs possibles de E.

$$E=0 \Rightarrow S'=R'=0$$

La bascule est en mode mémoire et les variations de D n'affectent pas les sorties.

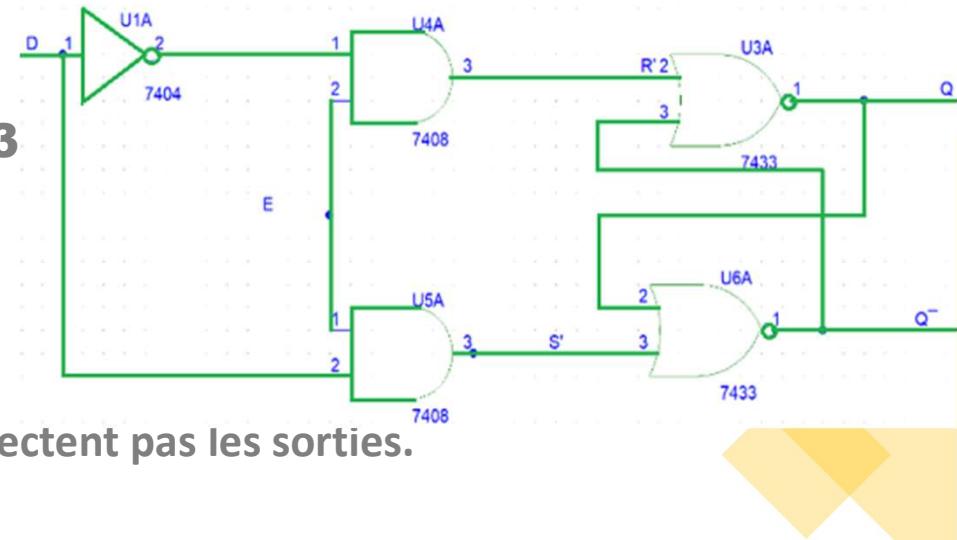
$$E=1 \Rightarrow S'=D \text{ et } R'=\bar{D}$$

Dans ces conditions la sortie Q recopie D.

Equations

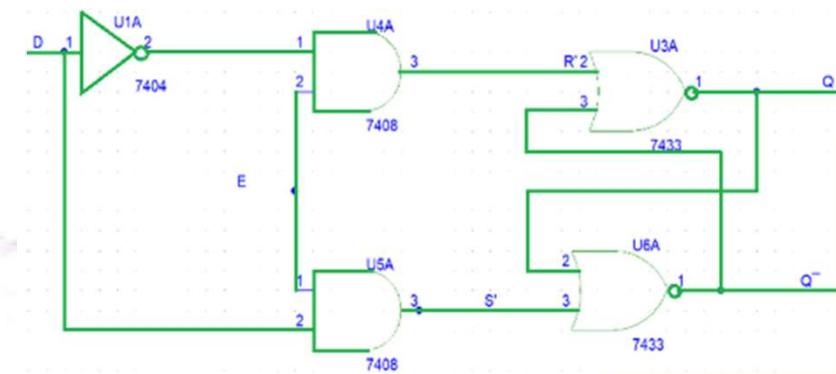
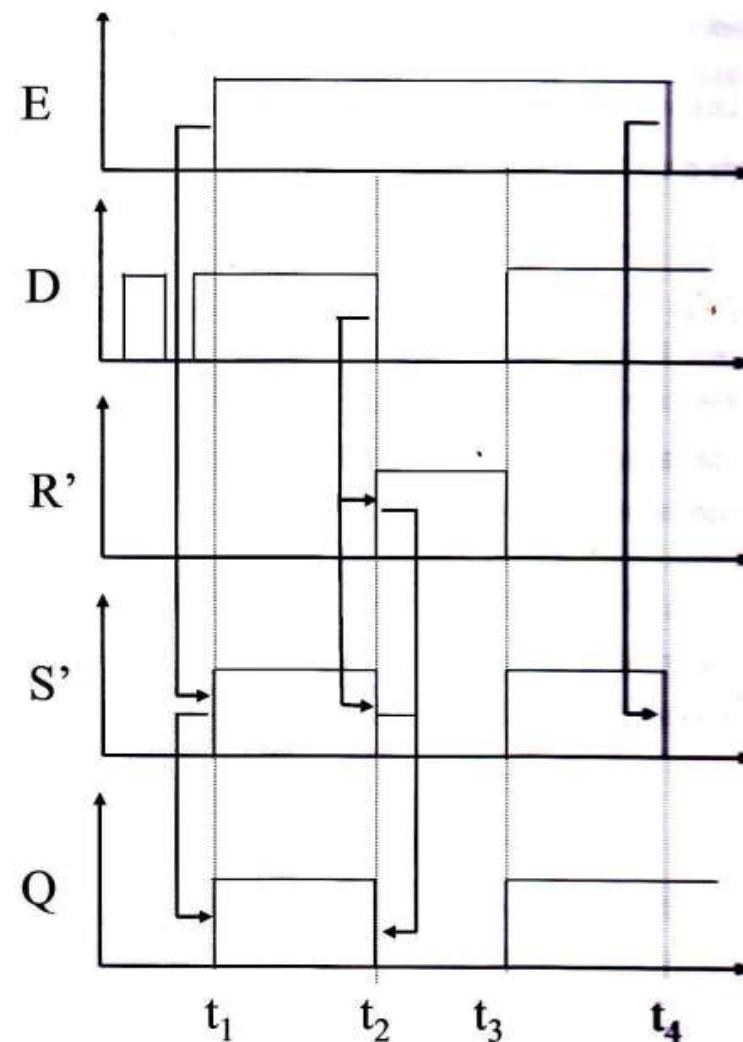
$$Q = \overline{R'}(S' + Q_0) = D(D + Q_0) = D(1+Q_0)$$

$$Q = D$$



Logique séquentielle – Bascules (16/38)

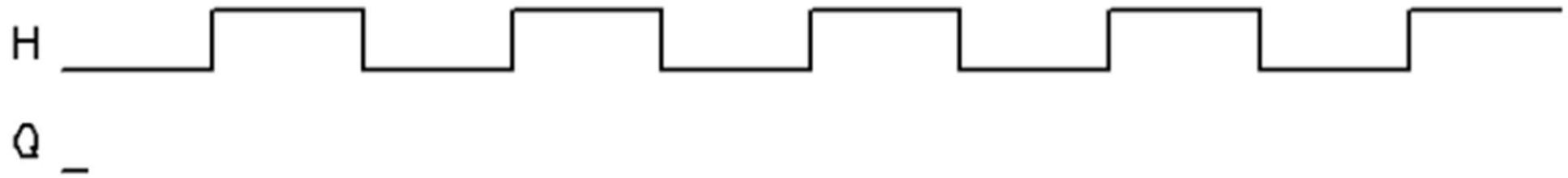
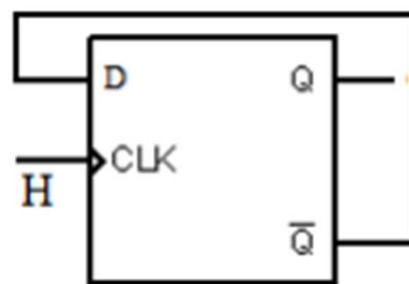
Bascule D – Chronogramme – (3/3)



Logique combinatoire – Circuits combinatoires

Exercices d'application à faire en séance ou devoir maison

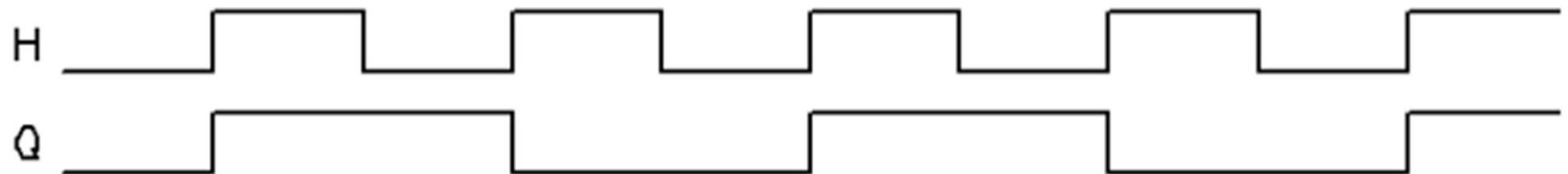
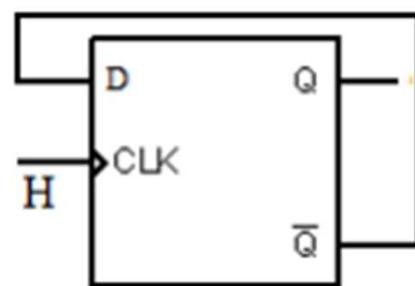
Bascule D

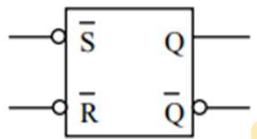


Logique combinatoire – Circuits combinatoires

Exercices d'application à faire en séance ou devoir maison

Bascule D (correction)

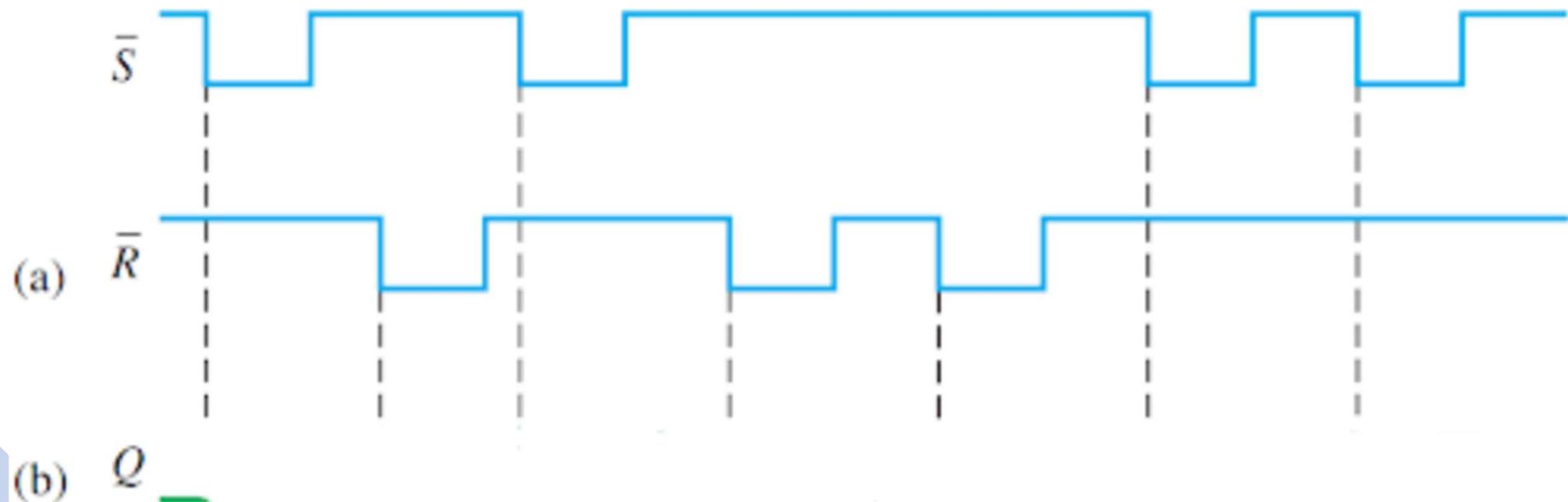


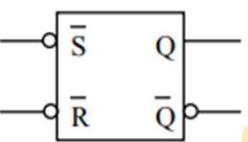


Logique séquentielle - Bascules

Exercices d'application à faire en séance ou devoir maison

Bascule RS chronogramme à compléter. Supposons que Q est initialement 0.

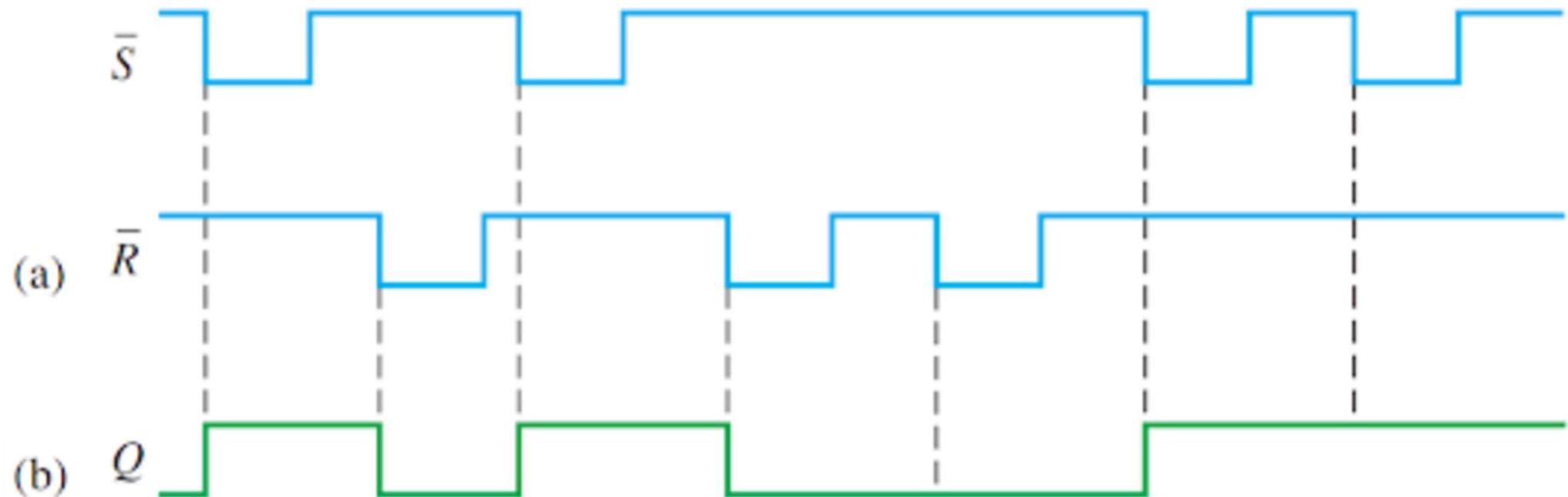




Logique séquentielle - Bascules

Exercices d'application à faire en séance ou devoir maison

Bascule RS chronogramme à compléter. Supposons que Q est initialement 0.



Logique séquentielle – Bascules

Bascules Synchrones et Asynchrone (1/7)

En logique séquentielle, on est amené à définir deux grands types de circuits: les circuits asynchrones et les circuits synchrones.

Bascules Asynchrones

Ce sont des systèmes où les entrées sont sensibles à des niveaux de tension (0 ou 1, bas ou haut...). Après un changement des entrées, le circuit évolue jusqu'à ce qu'il devienne stable. Les transitions d'un état à un autre se produisent à des instants quelconques et non contrôlables.

Bascules Synchrones

Ce sont des systèmes où une des entrées généralement appelée entrée d'horloge (ou CLK pour clock). Les entrées sont sensibles à des impulsions, les autres entrées restantes sont sensibles à des niveaux.

Pour ces systèmes synchrones, quelle que soit la valeur des entrées, le passage d'un état à un autre ne se fait qu'au moment où l'entrée d'horloge reçoit une impulsion.

Les circuits synchrones avec horloge sont très répandus.

Ils sont utilisés dans: les ordinateurs. Où, les opérations doivent être parfaitement cadencées de manière à se produire dans un ordre bien déterminé. La synchronisation par une horloge commune est indispensable.

Logique séquentielle – Bascules

Structure maître-esclave (2/7)

Les bascules synchrones présentent l'avantage d'être peu sensibles aux parasites. En effet, puisque seule compte la valeur des entrées au moment de l'impulsion de l'horloge, toutes les variations imprévues des entrées entre deux impulsions d'horloge sera sans effet sur la bascule.

Ces bascules sont basées sur une structure maître-esclave.

Une structure est dite maître-esclave; la bascule est le maître, la bascule de sortie, qui recopie l'état du maître, est esclave. Cette structure est communément utilisée dans les compteurs.

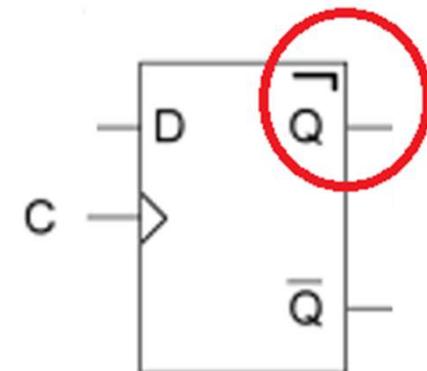
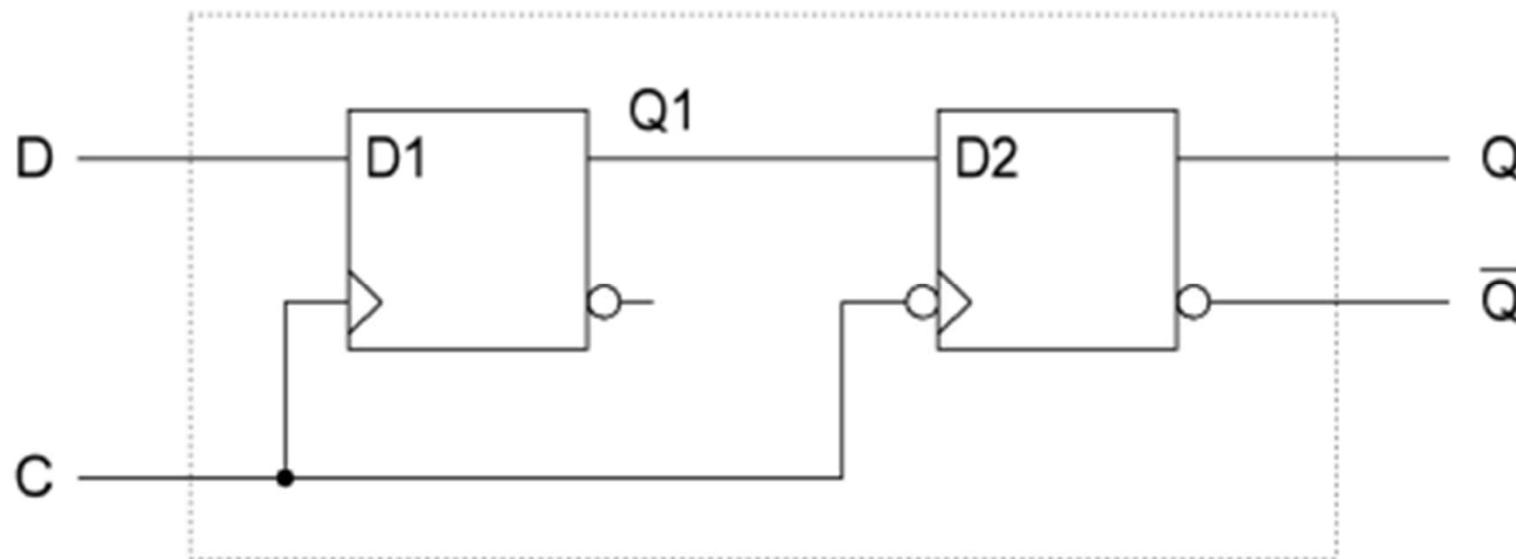
La plupart des bascules synchrones sont basées sur ce type de structure.

Une bascule de type maître-esclave est mise en œuvre en connectant deux bascules, appelées maître et esclave, dont les signaux d'horloge sont complémentaires

Logique séquentielle – Bascules

Bascule D synchronisée structure maître-esclave – Logigramme et symbole (3/7)

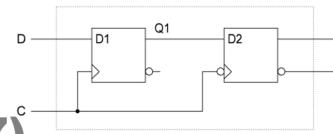
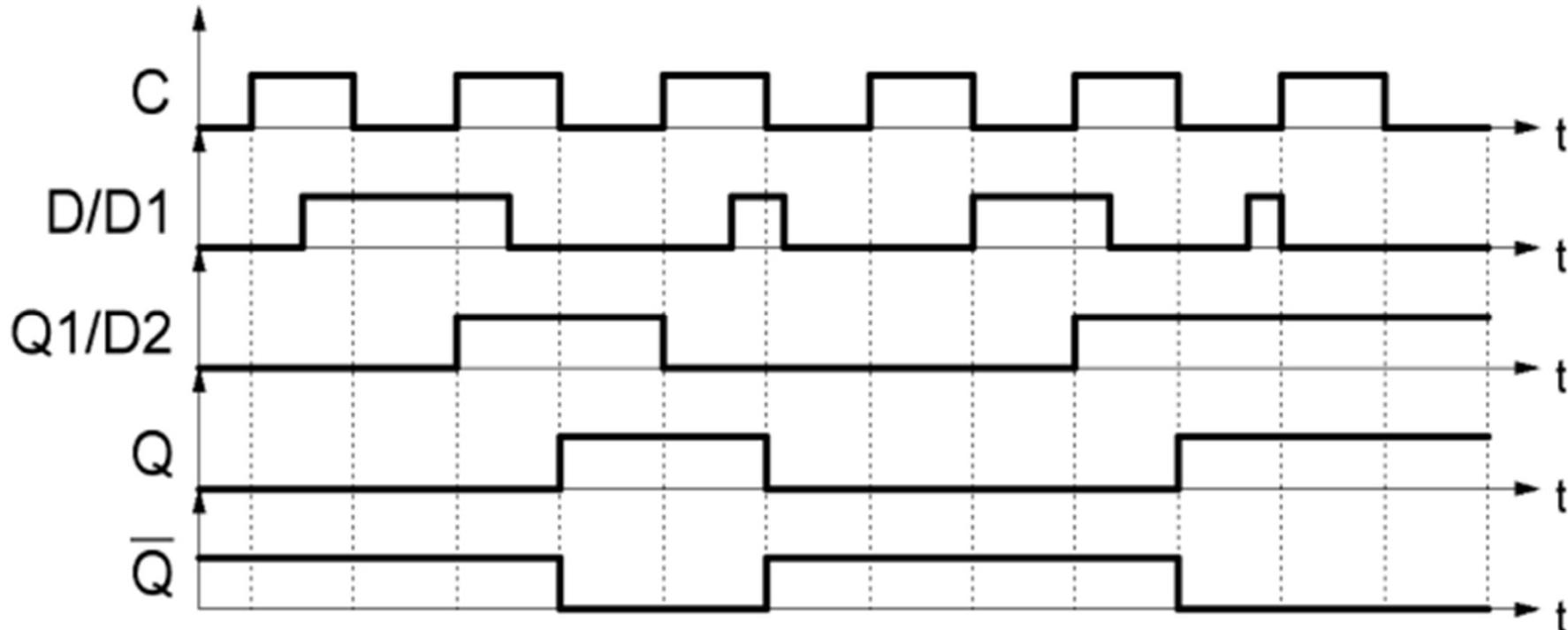
Une bascule D déclenchée par front peut être mise en œuvre en utilisant une structure maître-esclave qui est composée de deux bascules D ;



- ❖ La première bascule joue le rôle de maître. Sa sortie (Q_1/D_2) est modifiée à chaque front montant du signal d'horloge en fonction de l'entrée D. **Mais à ce stade, la sortie Q n'a pas encore été modifiée.**
- ❖ La seconde bascule joue le rôle d'esclave. Ses sorties (Q et \bar{Q}) sont modifiées à chaque front descendant du signal d'horloge en fonction de Q_1/D_2 ; c'est-à-dire en fonction de la valeur de D au moment du précédent front montant.

Logique séquentielle – Bascules

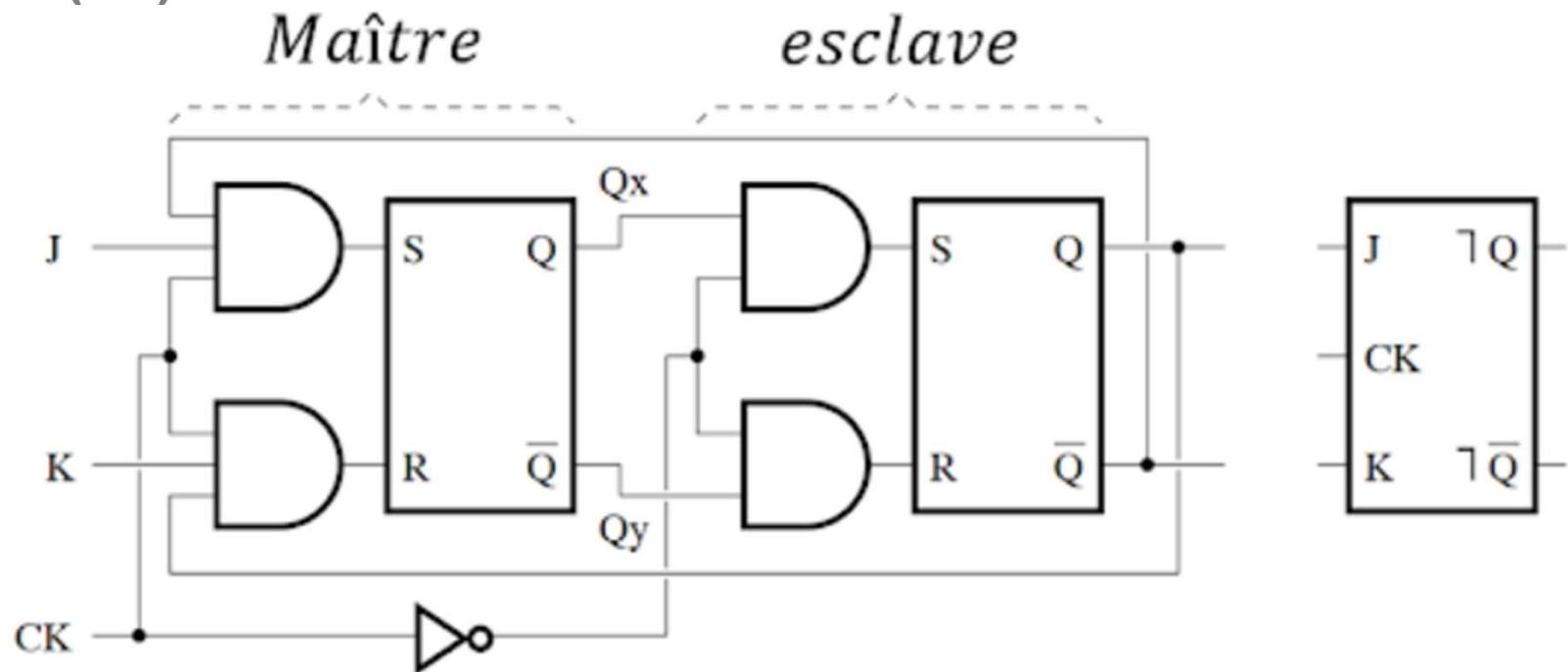
Bascule D synchronisée structure maître-esclave – Chronogramme (4/7)



- ❖ La première bascule joue le rôle de maître. Sa sortie (Q1/D2) est modifiée à chaque front montant du signal d'horloge en fonction de l'entrée D. Mais à ce stade, la sortie Q n'a pas encore été modifiée.
- ❖ La seconde bascule joue le rôle d'esclave. Ses sorties (Q et \bar{Q}) sont modifiées à chaque front descendant du signal d'horloge en fonction de Q1/D2 ; c'est-à-dire en fonction de la valeur de D au moment du précédent front montant.

Logique séquentielle – Bascules

Bascule JK maître-esclave synchronisée structure maître-esclave – Logigramme et symbole (5/7)



Lorsque la bascule maître est activée, son état logique de sortie est déterminé non seulement par les entrées J et K, mais également par les sorties, Q et \bar{Q} , de la bascule esclave. L'état de la bascule maître est ensuite transféré à la bascule esclave uniquement lorsque le signal d'horloge passe de haut en bas (front descendant).

Logique séquentielle – Bascules

Bascule JK maître-esclave synchronisée structure maître-esclave – Table de vérité (6/7)

J	K	CK	Q^+	\bar{Q}^+
x	x	0	Q	\bar{Q}
0	0	⊟	Q	\bar{Q}
0	1	⊟	0	1
1	0	⊟	1	0
1	1	⊟	\bar{Q}	Q

Logique séquentielle – Bascules

Représentation des bascules synchrones - (7/7)

Les principales bascules synchrones disponibles dans le commerce sont :

- ❖ les bascules RS de table de vérité identique à celle de leurs homologues asynchrones à base de NOR;
- ❖ les bascules D;
- ❖ les bascules JK, identiques aux bascules RS pour les entrées autres que $RS = 11$ (l'entrée J correspond à S et l'entrée K à R). Pour les entrées $RS = 11$ ces bascules fonctionnent en mode Toggle à savoir que leur sortie Q change de valeur à chaque impulsion d'horloge. Par rapport aux bascules RS, les bascules JK permettent d'utiliser toutes les combinaisons des entrées.

Logique séquentielle – Bascules

Bascule RSH Synchrone (1/5)

Le verrou RSH est un verrou RS possédant une troisième entrée : H (horloge). Il y a trois modèles de bascule. Ces équipements sont dits **synchrones** car ils dépendent de l'état des entrées de l'horloge.

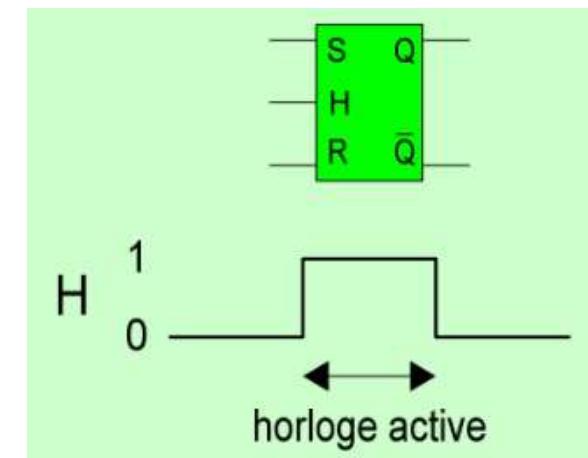
Changements sur un état au niveau haut de l'horloge:

Les états sont :

- ❖ quand H est active (niveau 1) : le verrou RSH se comporte comme un verrou RS.
- ❖ quand H est inactive (niveau 0) : verrouillage (fonction mémoire).

Table de vérité & Symbole

R	S	H	Q_{n+1}	Fonction
X	X	Inactive	Q_n	Mémoire
0	0	Active	Q_n	Mémoire
0	1	Active	1	Mémoire
1	0	Active	0	Set
1	1	Active	X	Reset

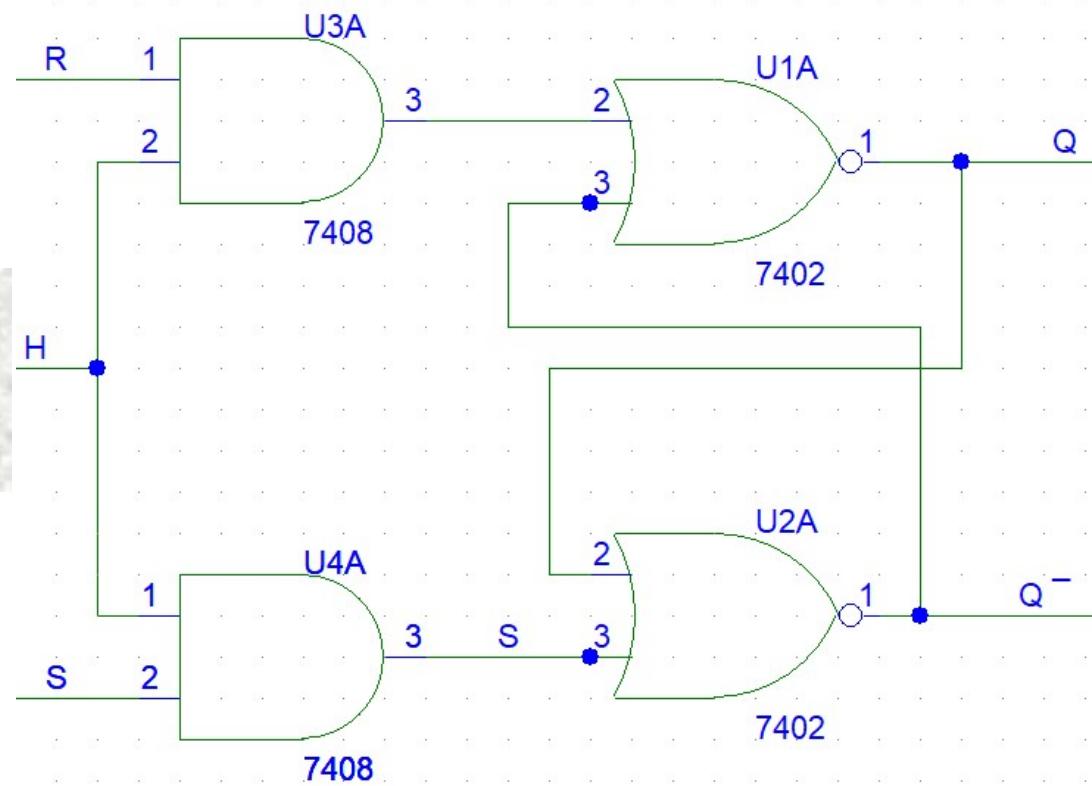


Logique séquentielle – Bascules

Bascule RSH Synchrone (2/5)

Equation logique & Logigramme

$$Q_{n+1} = H \cdot (\overline{R} \cdot Q_n + S) + \overline{H} \cdot Q_n$$



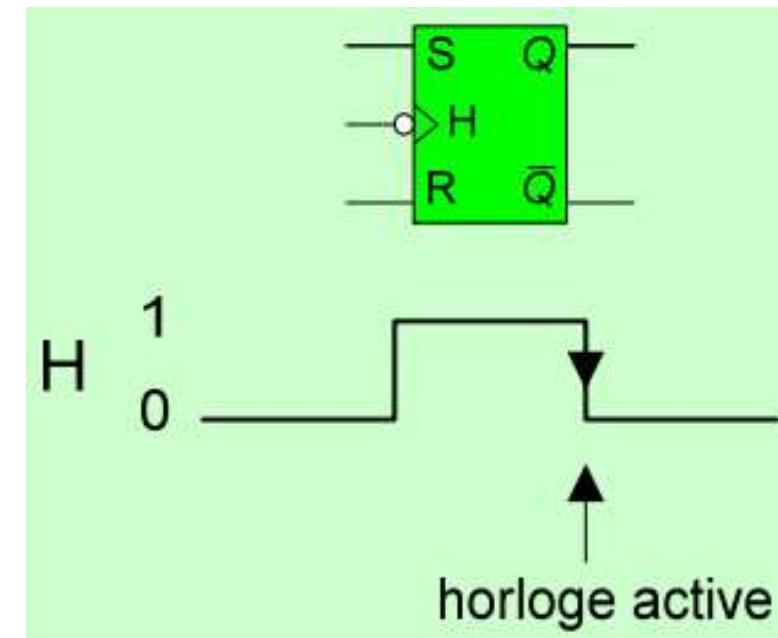
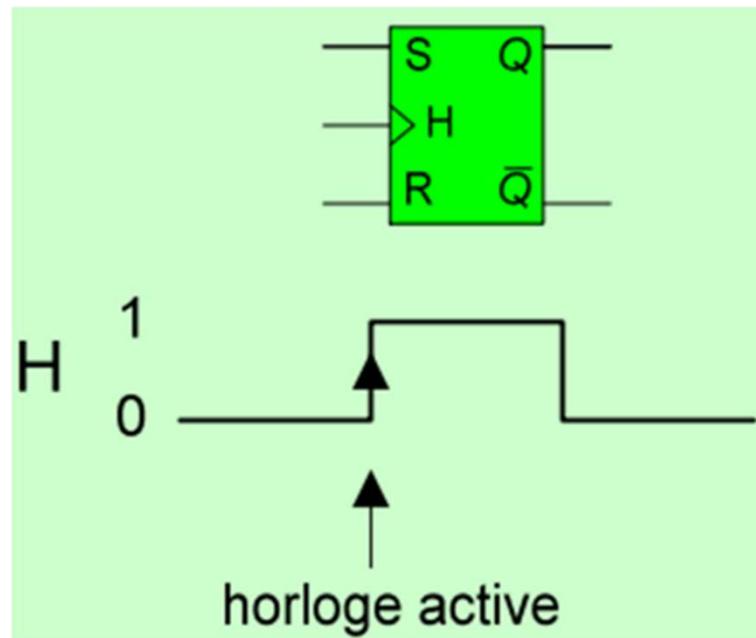
Logique séquentielle – Bascules

Bascule RSH Synchrone (3/5)

Changements sur les fronts de l'horloge & Symboles

Il existe deux types de bascule (flip-flop en anglais) RSH à horloge active :

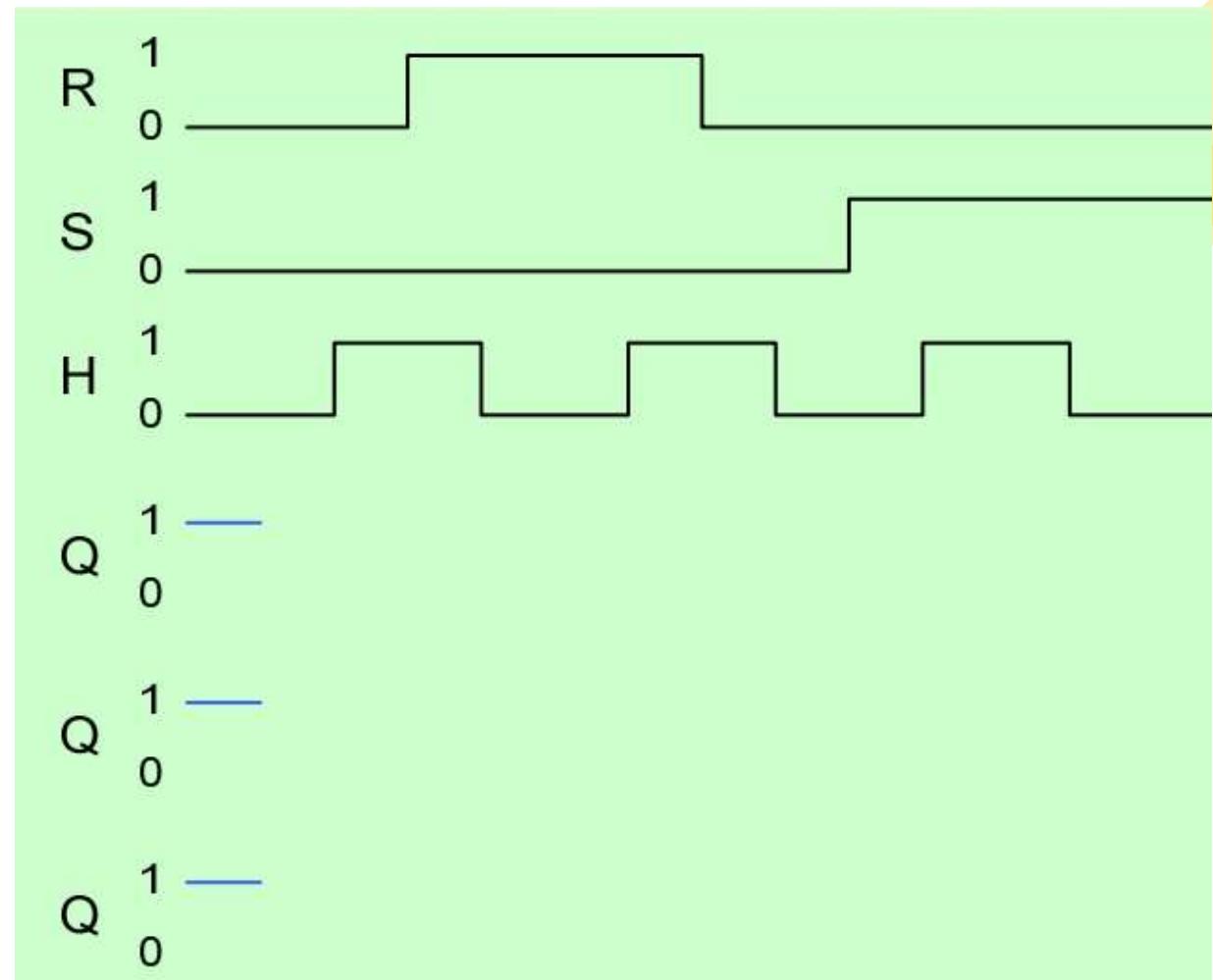
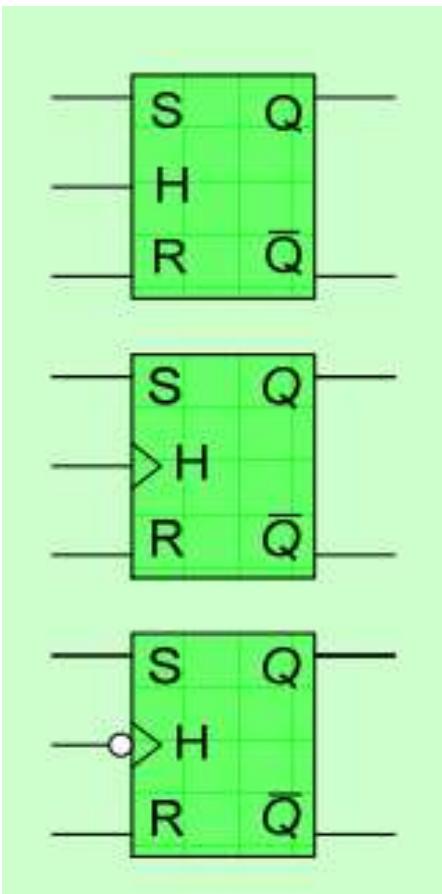
- sur « front montant » (à l'instant où H bascule de 0 à 1);
- sur « front descendant » (à l'instant où H bascule de 1 à 0).



Logique séquentielle – Bascules

Bascule RSH Synchrone (4/5)

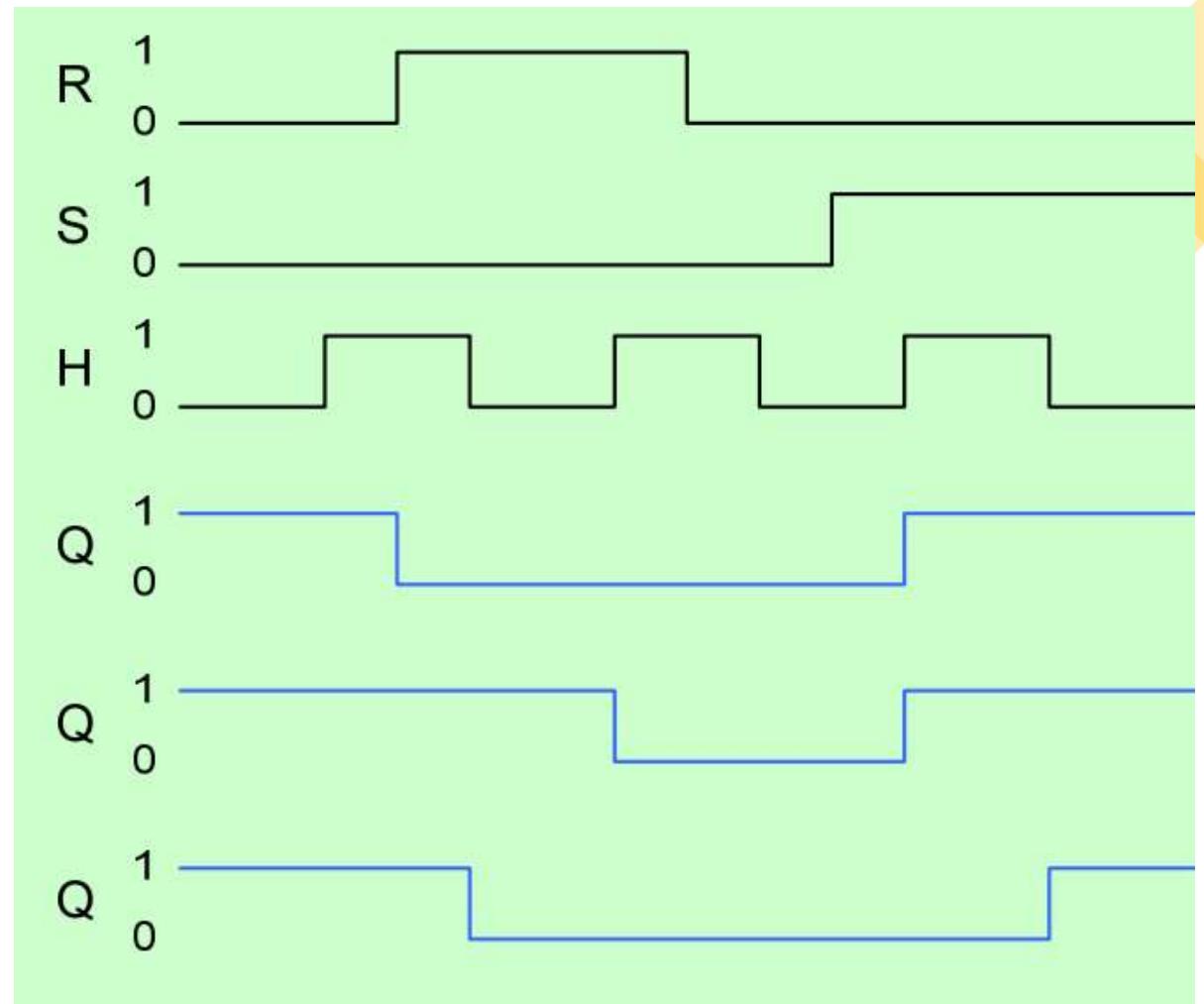
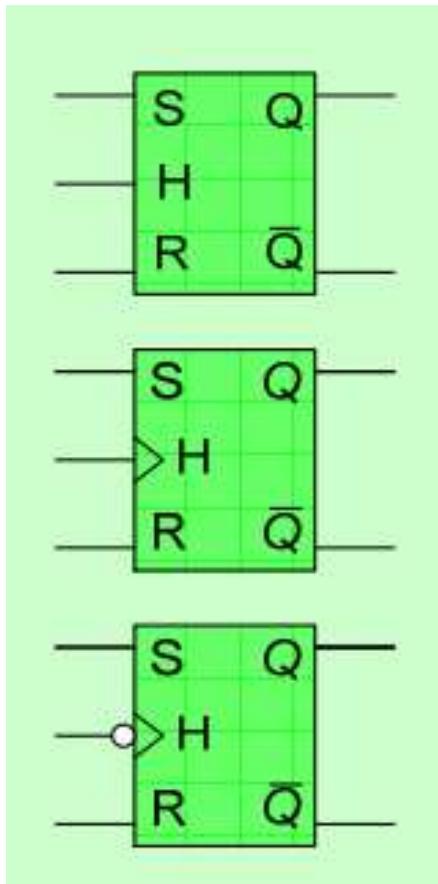
Les chronogrammes (A compléter)



Logique séquentielle – Bascules

Bascule RSH Synchrone (5/5)

Les chronogrammes (Réponses)



Logique séquentielle – Bascules

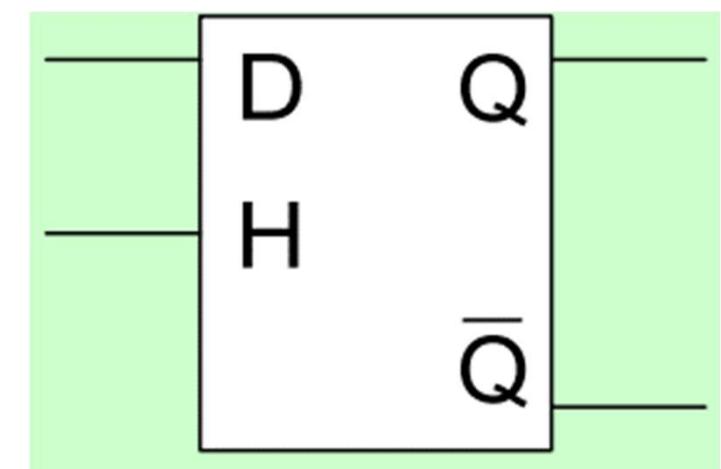
Bascule D synchrone (1/3)

Lorsque l'horloge est active (niveau 1), le niveau présent à l'entrée D est transféré en sortie ($Q_{n+1} = D$).

Lorsque l'horloge est inactive (niveau 0), la sortie est « verrouillée ».

Table de vérité & Symbole

D	H	Q_{n+1}	Fonction
X	Inactive	Q_n	Mémoire
0	Active	0	Reset
1	Active	1	Set

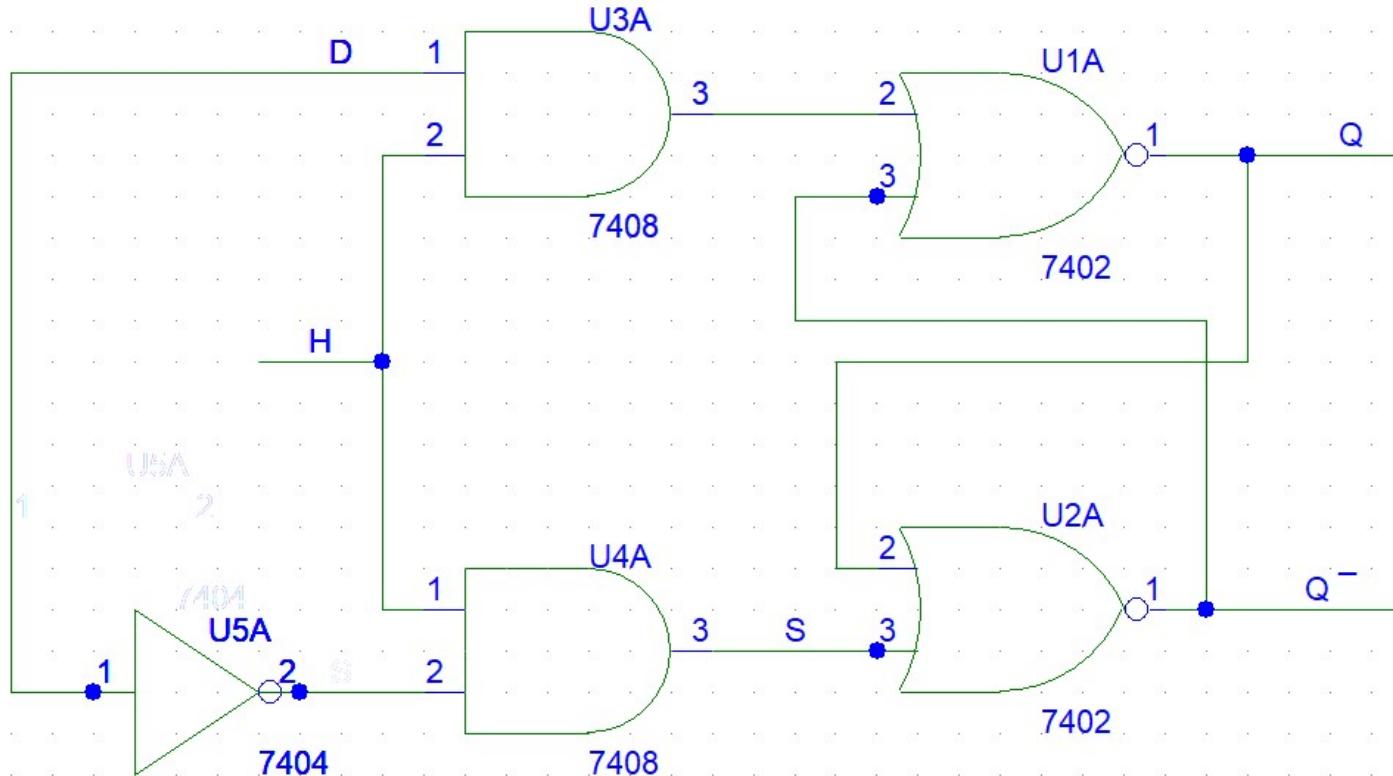


Logique séquentielle – Bascules

Bascule D synchrone (2/3)

Equation & Logigramme

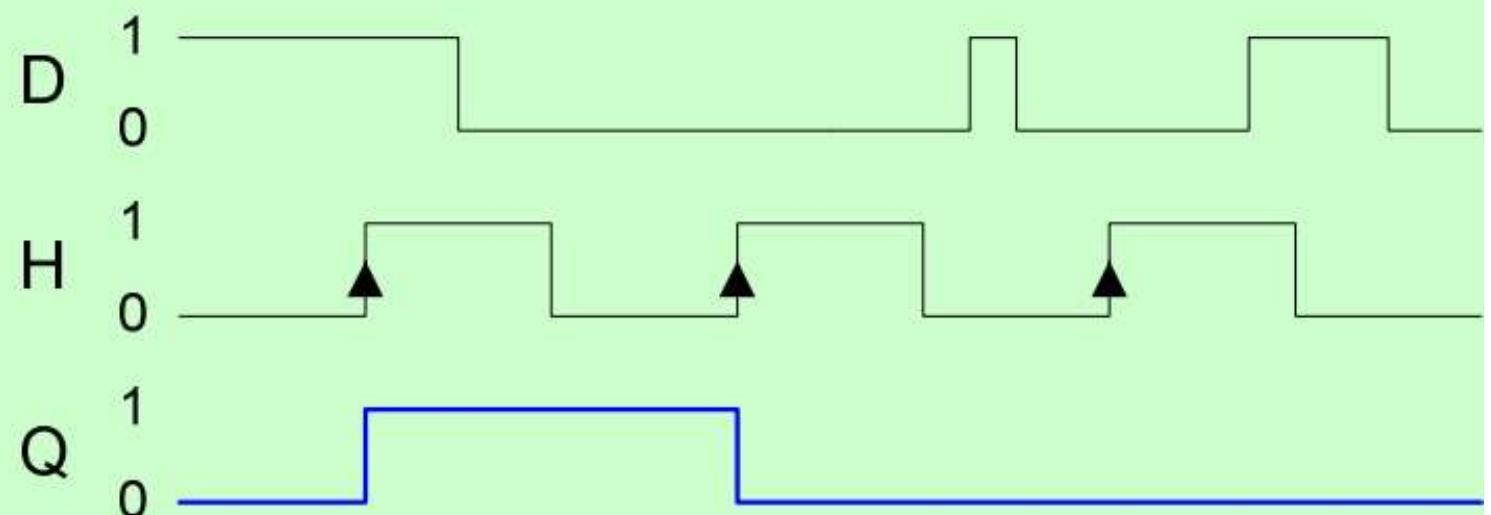
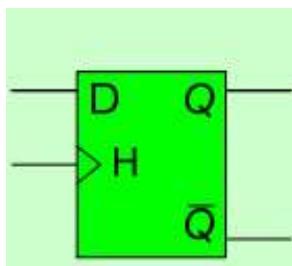
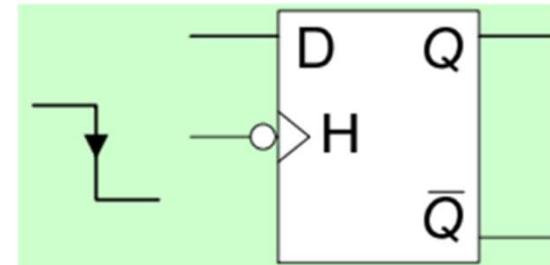
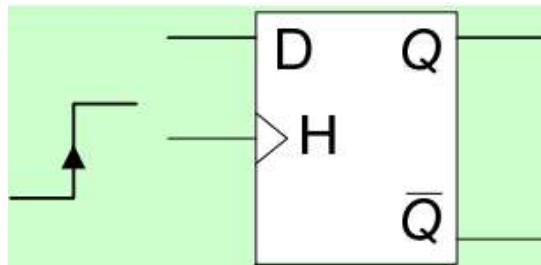
$$Q_{n+1} = H \cdot D + \bar{H} \cdot Q_n$$



Logique séquentielle – Bascules

Bascule D synchrone (3/3)

Symboles & Logigramme

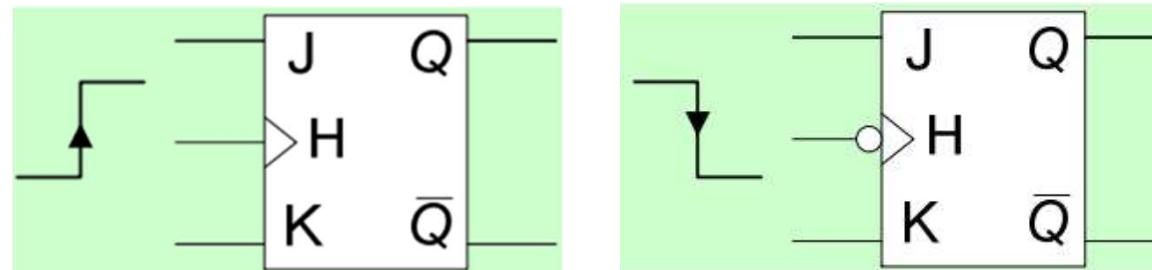


Logique séquentielle – Bascules

Bascule JK Synchrone (1/2)

Bascule JK est une bascule RS mais dans le cas où la valeur de deux entrées est 1 la sortie est porté par la négation de l'état précédente. Telle que $S \rightarrow J$ et $R \rightarrow K$

Table de vérité & Symboles

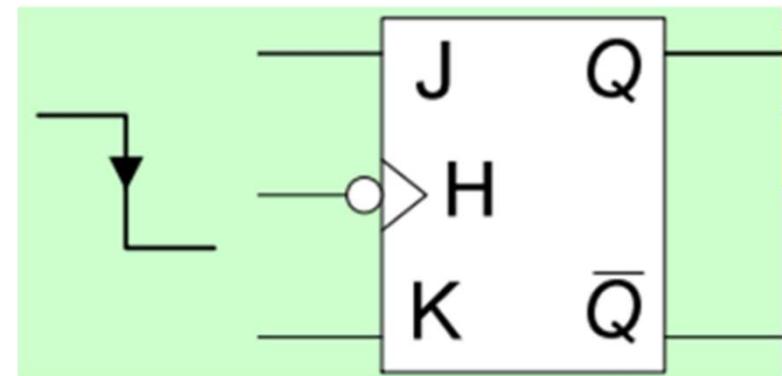
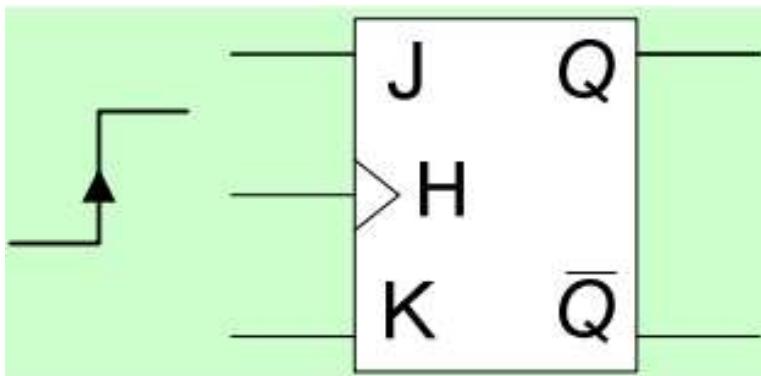


J	K	H	Q_{n+1}	Fonction
X	X	Inactive	Q_n	Mémoire
0	0	Active	Q_n	Mémoire
0	1	Active	0	Reset
1	0	Active	1	Set
1	1	Active	\bar{Q}	<i>Basculement</i>

Logique séquentielle – Bascules

Bascule JK Synchrone (2/2)

Equation & Symboles



$$Q_{n+1} = H \cdot \left(J \cdot \overline{Q_n} + \overline{K} \cdot Q_n \right) + \overline{H} \cdot Q_n$$

Logique séquentielle – Compteurs, registres et mémoires

Compteurs – Généralités

Leur fonction est de coder, dans un système de numérotation voulu, le nombre d'impulsions en un point d'un circuit.

Les compteurs sont des ensembles des bascules montées en série ou en parallèle pour soit incrémenter ou décrémenter, on distingue deux types : les compteurs Asynchrone et Synchrone.

Notions de compteur synchrone et de compteur asynchrone

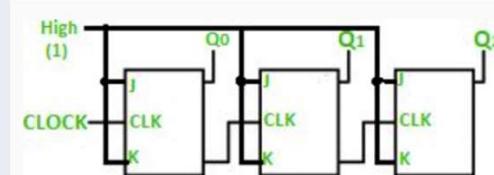
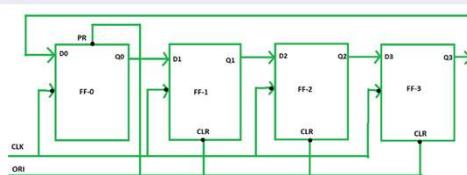
Dans un **compteur Asynchrone**, également connu sous le nom de Ripple Counter , différentes bascules sont déclenchées avec une horloge différente, pas simultanément.

En **mode compteur Synchrone** , toutes les bascules sont déclenchées simultanément avec la même horloge et le compteur synchrone est plus rapide que le compteur asynchrone en fonctionnement.

Logique séquentielle – Compteurs, registres et mémoires

Compteurs – Comparaison Compteur Asynchrone et Compteur Synchrone

Compteur Synchrone	Compteur Asynchrone
Toutes les bascules sont déclenchées simultanément avec la même horloge	Les différentes bascules sont déclenchées avec une horloge différente et pas simultanément
Plus rapide que le compteur asynchrone en fonctionnement	Plus lent que le compteur synchrone en fonctionnement
Aucune erreur de décodage	Peut produire une erreur de décodage
Également appelé compteur parallèle	Appelé compteur série.
La conception et la mise en œuvre du compteur complexes en raison de l'augmentation du nombre d'états.	La conception et la mise en œuvre du compteur sont très simples.
Le compteur fonctionnera dans n'importe quelle séquence de comptage souhaitée.	Le compteur asynchrone fonctionnera uniquement dans une séquence de comptage fixe (UP/DOWN).
Délai de propagation est moindre.	Délai de propagation est élevé.
Exemple: compteur anneau	Exemple: Compteur Ripple



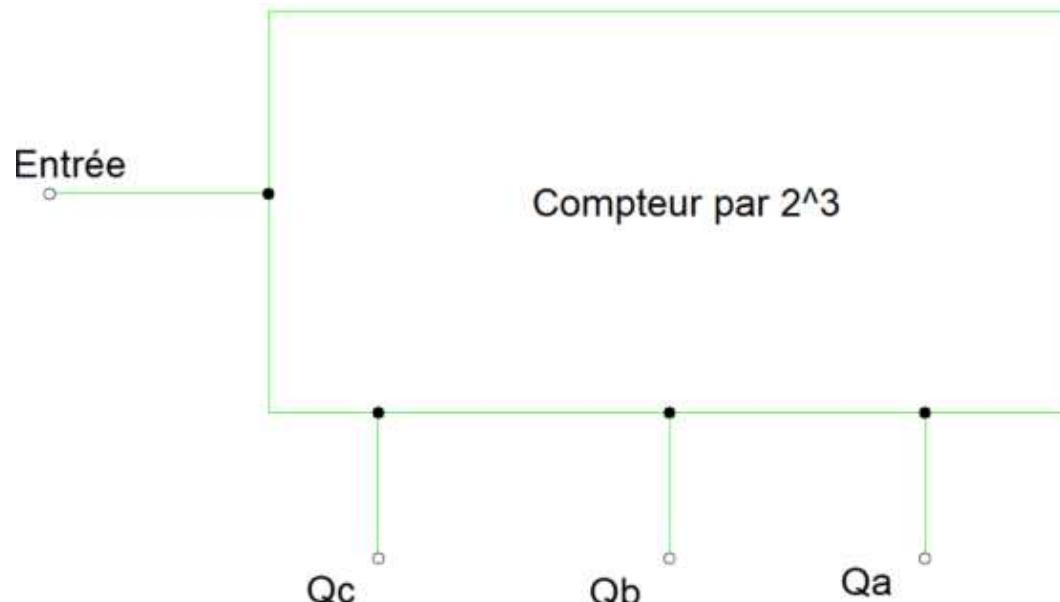
Logique séquentielle – Compteurs, registres et mémoires

Compteurs binaires – Schéma de principe & Table de vérité

Le tableau ci-contre représente un compteur binaire par 8 ($n=3$; $2^3=8$).

Le compteur possède trois bits de sortie notées: Q_C , Q_B , Q_A affectés respectivement des poids 2^2 , 2^1 , 2^0 .

Le bit Q_A est le bit de poids faible (LSB) et Q_C est le bit de poids fort (MSB).



Etat n			Etat n+1		
Q_C	Q_B	Q_A	Q_C	Q_B	Q_A
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Logique séquentielle – Compteurs, registres et mémoires

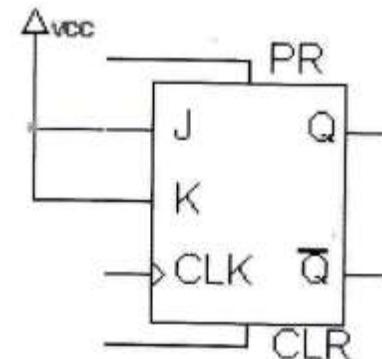
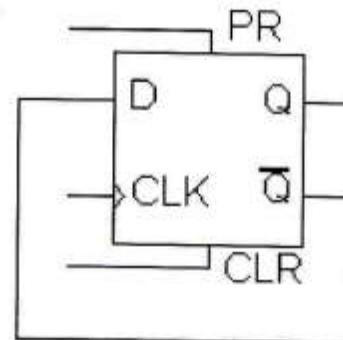
Réalisation d'un compteur binaire

La réalisation d'un compteur binaire repose sur les constats suivants vérifiables dans la table de vérité précédent:

1. **Le bit de poids faible commute à chaque impulsion d'horloge;**
2. Un bit de sortie Q_i commute ($0 \rightarrow 1$ ou $1 \rightarrow 0$) ssi tous les bits de poids plus faible sont au niveau 1;
3. Un bit de sortie Q_i commute ($0 \rightarrow 1$ ou $1 \rightarrow 0$) ssi le bit de poids immédiatement inférieur passe de 1 à 0.

Les points 2 et 3 sont équivalents mais conduisent à la réalisation de deux types de compteurs différents **synchrone et/ou asynchrone ayant leurs avantages et leurs inconvénients.**

Chaque bit de comptage correspond en pratique à une bascule T réalisé soit avec une bascule JK dont $J=K=1$ ou une bascule D synchrone dont la sortie \bar{Q} est rebouclée sur l'entrée.

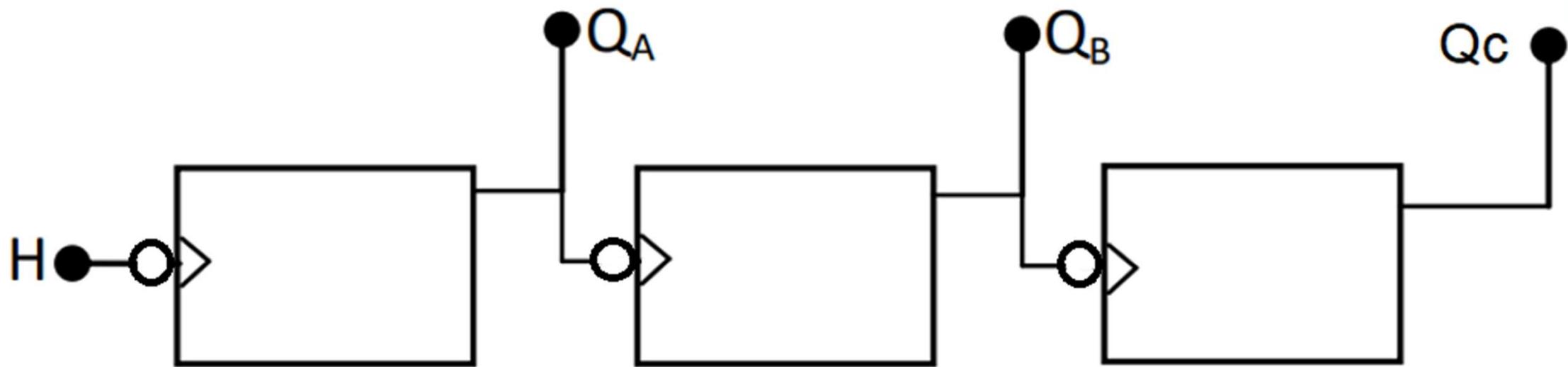


Logique séquentielle – Compteurs, registres et mémoires

Compteur binaire Asynchrone – Schéma de principe (1/4)

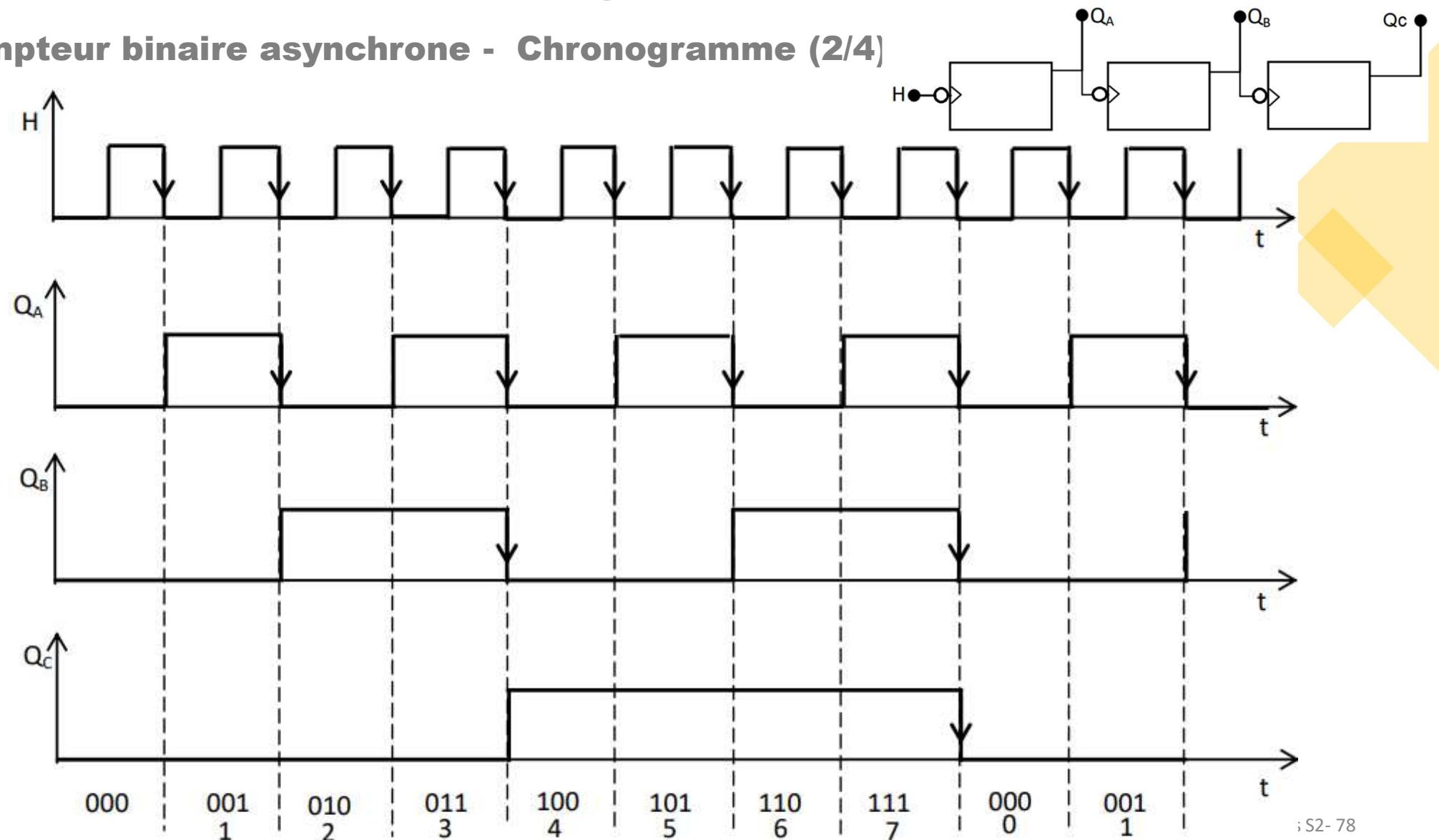
Il est formé par des bascules montées en série telles que chaque bascule donne l'impulsion à la bascule suivante.

Un compteur modulo n peut compter de zéro jusqu'à $(n-1)$. Du coup, si $n=8$, on peut compter de 0 à 7, et si $n=10$ on comptera de 0 à 9.



Logique séquentielle – Compteurs, registres et mémoires

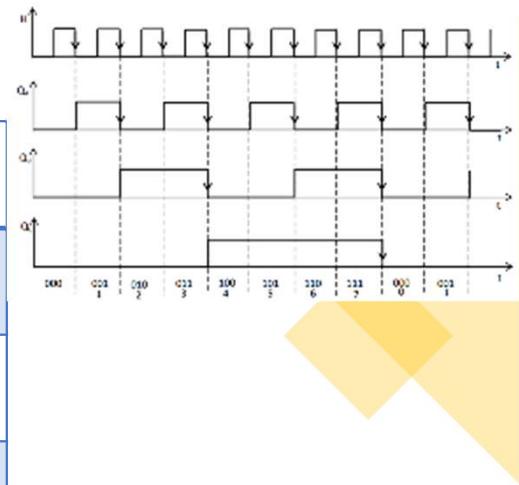
Compteur binaire asynchrone - Chronogramme (2/4)



Logique séquentielle – Compteurs, registres et mémoires

Compteur binaire asynchrone - Table de vérité (3/4)

Q_A	Q_B	Q_C	H
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

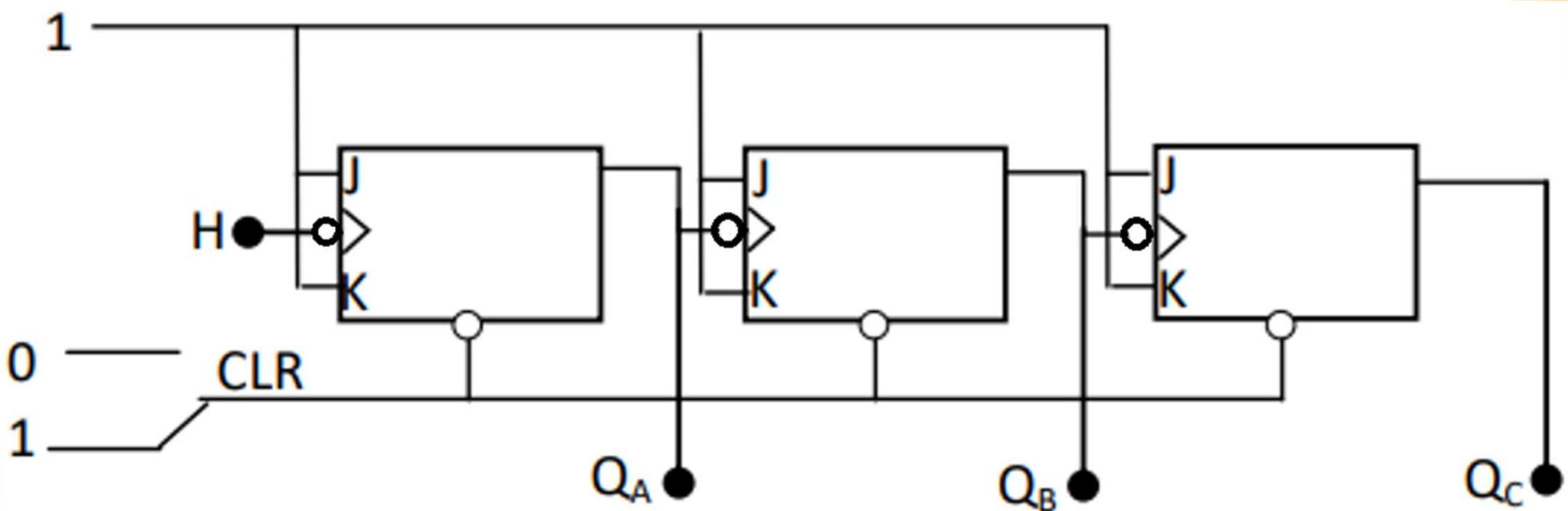
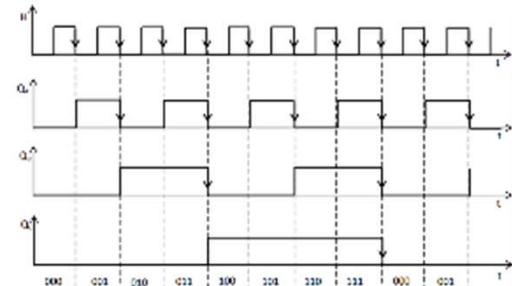


Logique séquentielle – Compteurs, registres et mémoires

Compteur binaire asynchrone - Bascule JK, J=K=1 (4/4)

On peut réaliser le compteur précédent avec des portes JK.

Les entrées CLR forcent le compteur à l'état 0 (mise à 0).



Logique séquentielle – Compteurs, registres et mémoires

Compteur binaire asynchrone - Décimal (modulo 10) – (1/4)

Ce compteur est compté de 0 à 9, le nombre des bascules utilisées pour compter de 0 à 9 est déterminé par l'opération suivante:

- ❖ $2^0 = 1 < 10;$
- ❖ $2^1 = 2 < 10;$
- ❖ $2^2 = 4 < 10;$
- ❖ $2^3 = 8 < 10;$
- ❖ $2^4 = 16 > 10.$

Donc le nombre des bascules utilisées au **compteur décimal (modulo 10)** est 4.

L'opération modulo est une opération binaire qui associe, dans le cas de deux entiers naturels, le reste de la division euclidienne du premier par le deuxième, le reste de la division de a par n ($n \neq 0$) est noté a modulo n (ou $a \% n$ dans certains langages informatiques).

Du coup:

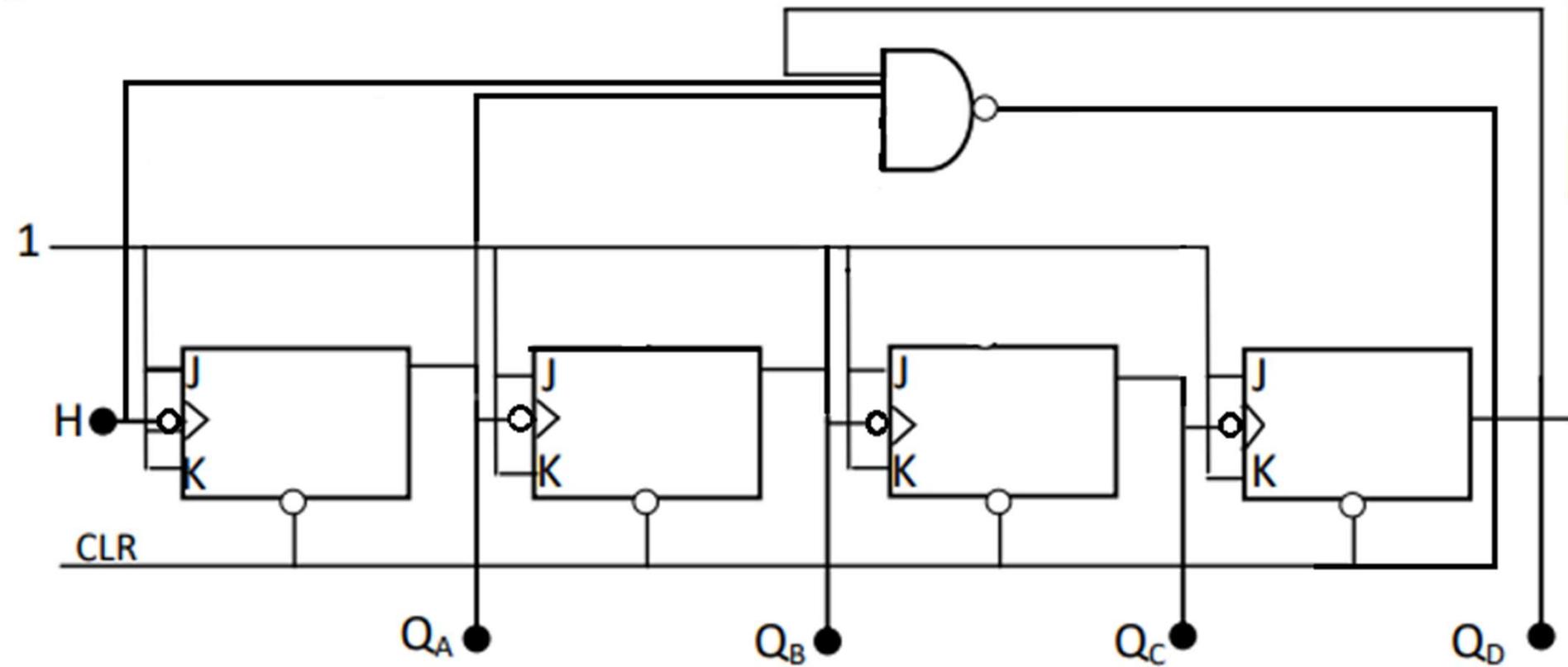
$$9 \text{ mod. } 4 = 1, \text{ car } 9 = 2 \times 4 + 1 \text{ et } 0 \leq 1 < 4.$$

$$\text{De même } 9 \text{ mod. } 3 = 3 \times 3 + 0.$$

Logique séquentielle – Compteurs, registres et mémoires

Compteur Asynchrone - Décimal (modulo 10) – (2/4)

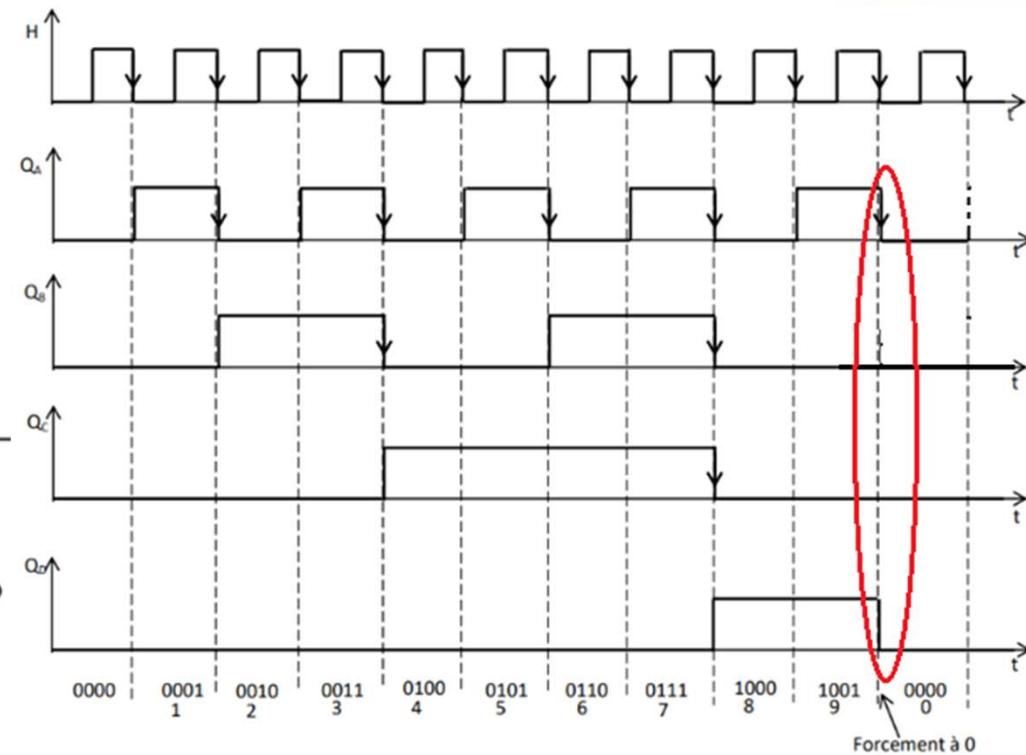
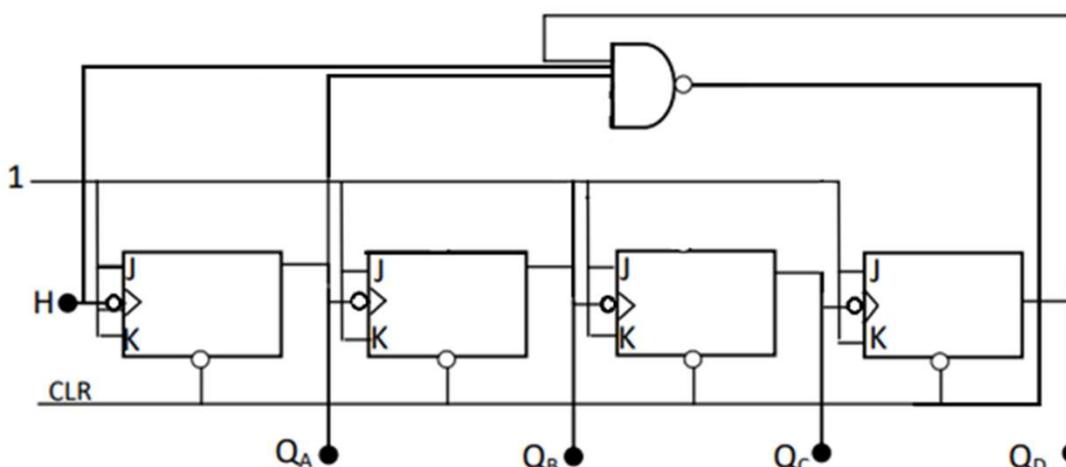
Logigramme



Logique séquentielle – Compteurs, registres et mémoires

Compteur Asynchrone - Décimal (modulo 10) – (3/4)

Ce compteur peut compter de zéro (0000) jusqu'à 9 (1001). Aussi, nous utilisons une porte NAND et on prend les sorties des bascules $Q_A Q_D = 11$ et l'horloge H comme entrées de la porte logique (NAND). Comme l'entrée CLR est complémentée, dès ces valeurs obtenues sur la porte NAND, il y a un forçage à 0 du compteur.



Logique séquentielle – Compteurs, registres et mémoires

Compteur Asynchrone - Décimal (modulo 10) – (4/4)

Table de vérité

On remarque que les sorties des bascules peuvent atteindre jusqu'à 15 (1 1 1 1) mais notre compteur décimal ne peut compter que de (0 à 9).

A quel autre système de codage peut-on associer le compteur Modulo 10 ?

Q_A	Q_B	Q_C	Q_D	H
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Logique séquentielle – Compteurs, registres et mémoires

Compteur Asynchrone - Modulo 5 – (1/2)

Exemple

Réaliser le schéma logique d'un compteur **modulo 5** (càd compté de 0 à 4) en utilisant les bascules JK.

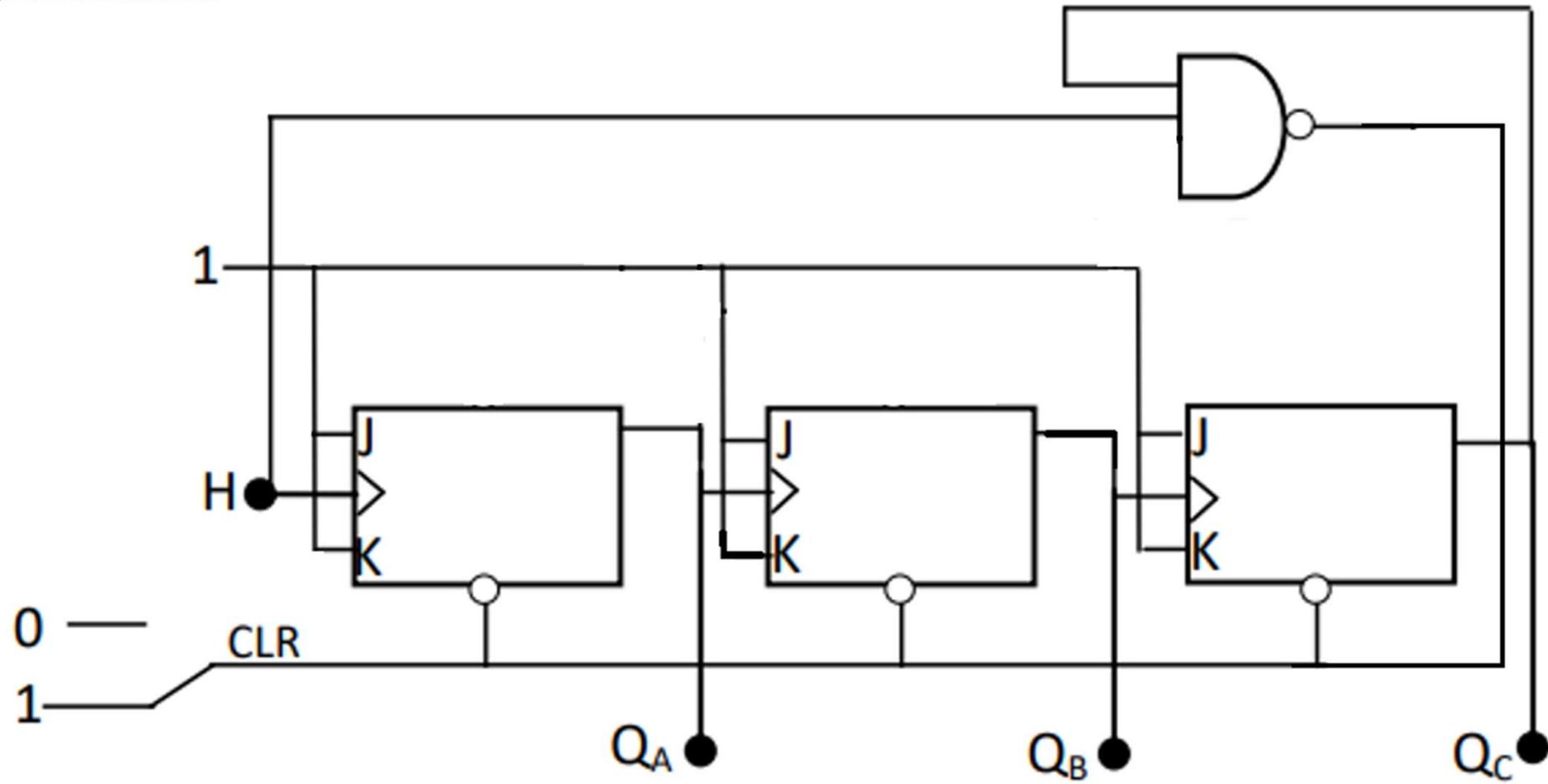
- ❖ $2^0 = 1 < 5;$
- ❖ $2^1 = 2 < 5;$
- ❖ $2^2 = 4 < 5;$
- ❖ $2^3 = 8 > 5;$

Donc le nombre des bascules utilisées pour le compteur modulo 5 est trois (3).

Logique séquentielle – Compteurs, registres et mémoires

Compteur Asynchrone - Modulo 5 – (2/2)

Diagramme



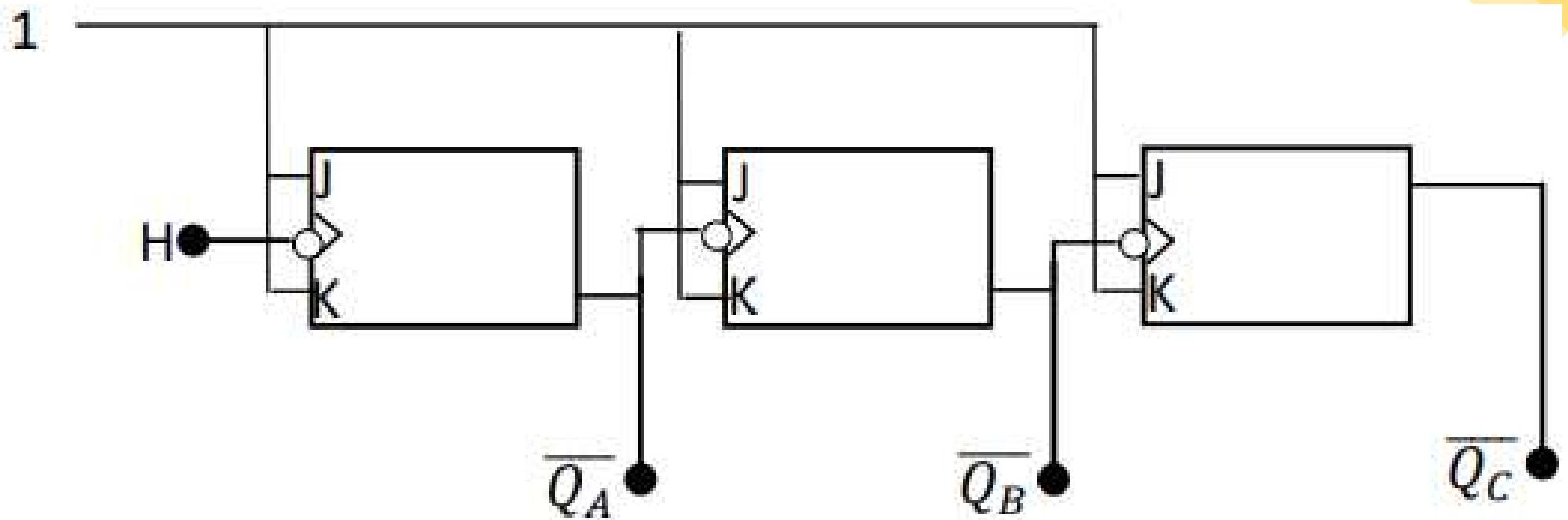
Logique séquentielle – Compteurs, registres et mémoires

Décomptage Asynchrone - (1/3)

Dans ce cas, on utilise les sorties complémentaires comme entrée à la bascule suivante.

Exemple

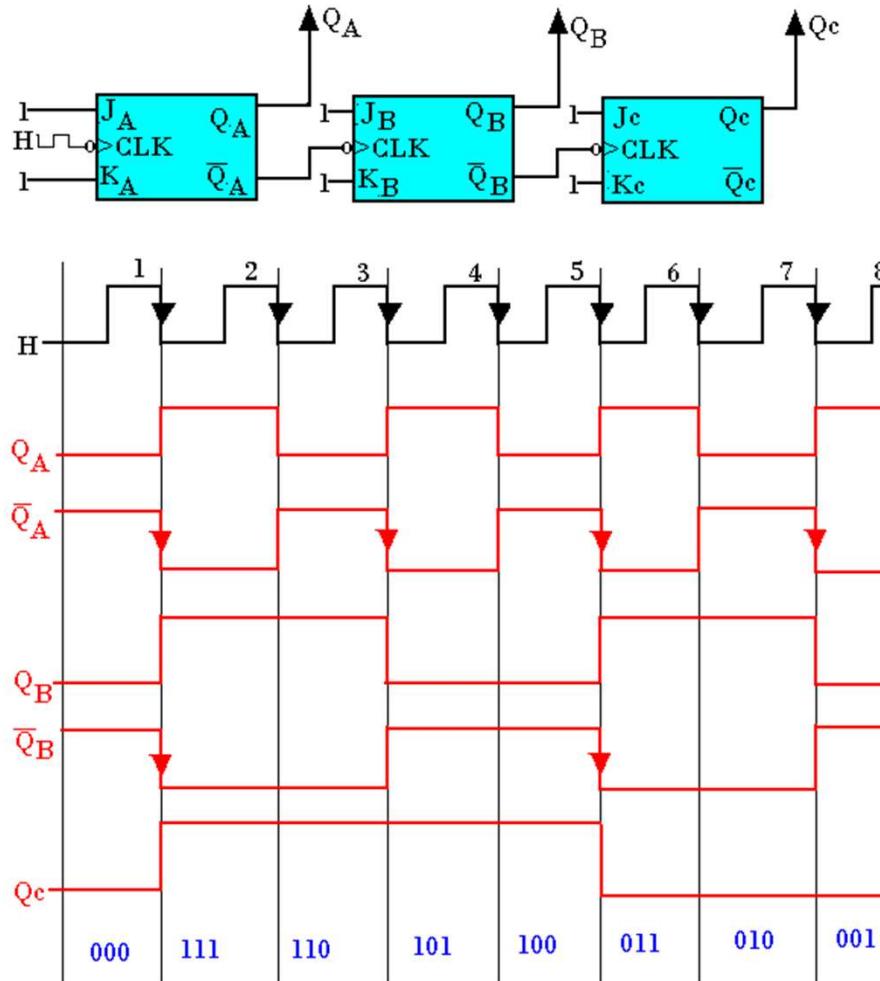
Décompteur Asynchrone modulo 8 : Comme nous avons vu ce compteur nécessite trois bascules



Logique séquentielle – Compteurs, registres et mémoires

Décomptage Asynchrone - (2/3)

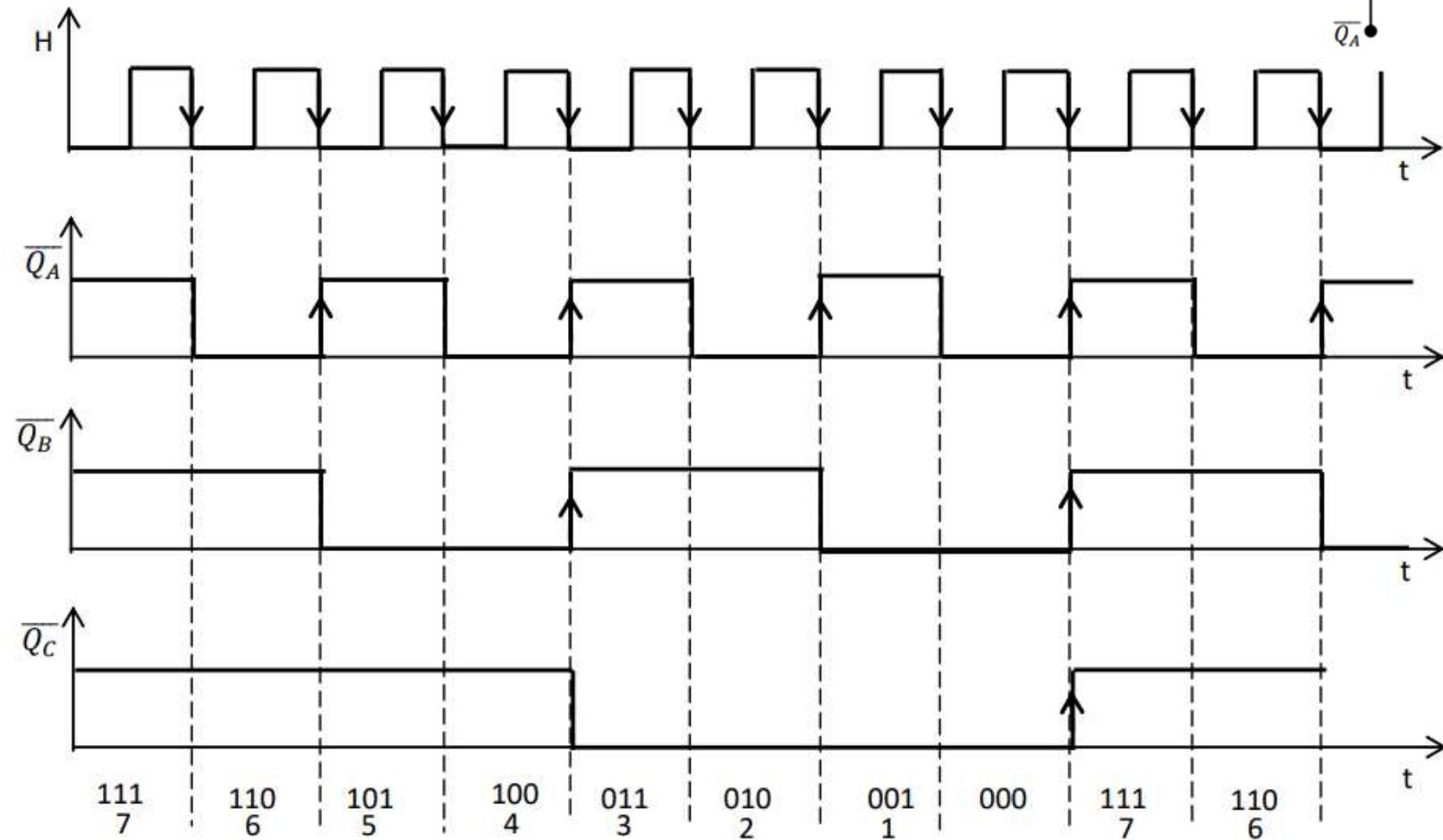
Explication détaillée du décompteur modulo 8



Logique séquentielle – Compteurs, registres et mémoires

Décomptage Asynchrone - (3/3)

Chronogramme



Logique séquentielle – Compteurs, registres et mémoires

Compteurs Synchrones - (1/6)

Compteurs binaires à retenue série

Rappelons que dans un compteur synchrone, chaque bit de sortie Q_i commute ($0 \rightarrow 1$ ou de $1 \rightarrow 0$), ssi tous les bits de poids inférieurs sont au niveau haut.

Par exemple, la bascule représentant le bit Q_D commute si $Q_A.Q_B.Q_C=1$. Cette quantité s'appelle la retenue.

Si la retenue est différente de 1 la bascule doit être en mode mémoire pour ne pas commuter. On réalise généralement ces compteurs en utilisant les bascules JK soit en mode mémoire ($J=K=0$) soit en mode Toggle ($J=K=1$).

Logique séquentielle – Compteurs, registres et mémoires

Compteurs Synchrones - (2/6)

Compteurs binaires à retenue série (suite)

La manière la plus simple de calculer la retenue est d'effectuer le double produit $(Q_A \cdot Q_B) \cdot Q_C$, et ainsi de suite, de proche en proche, en n'utilisant que des portes AND à deux entrées.

On aboutit au schéma de la slide suivante qui est finalement assez peu utilisé en pratique.

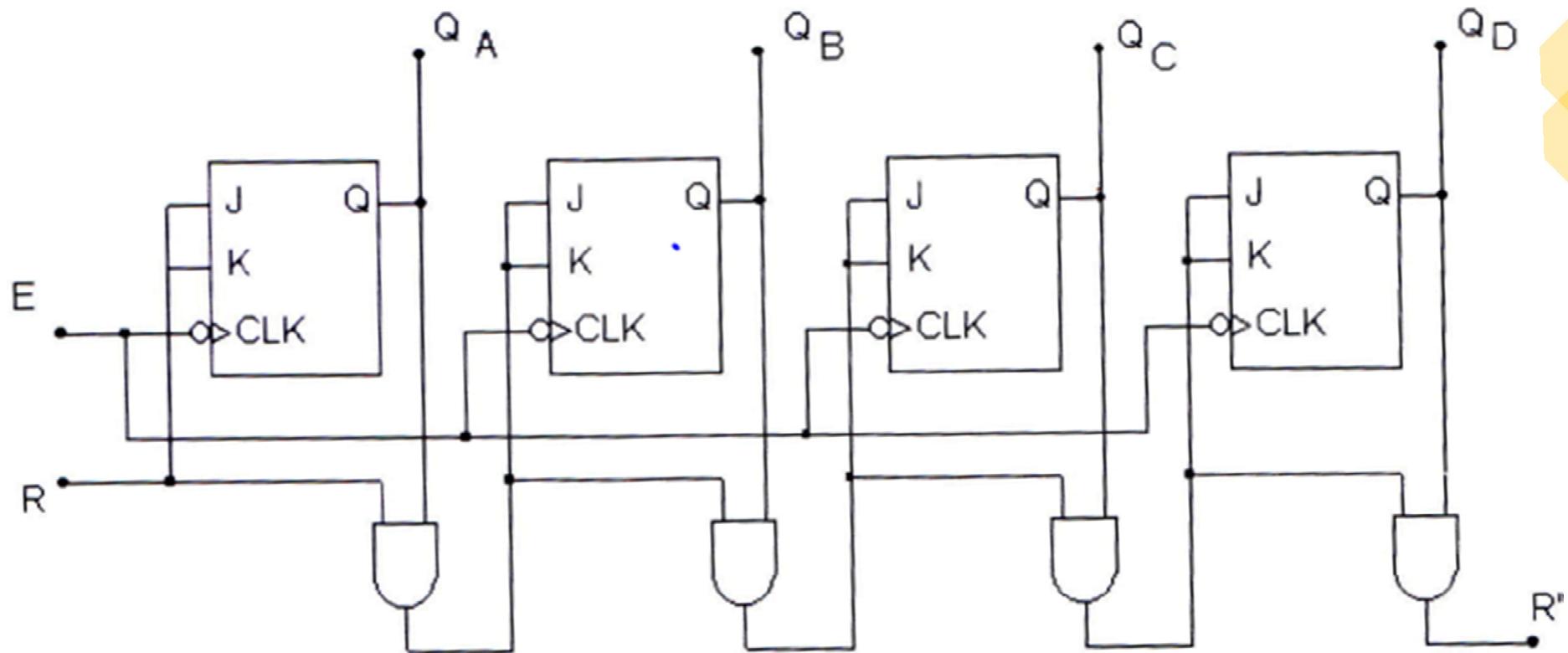
En effet, la retenue se propage d'un bout à l'autre du circuit avec un retard croissant dû à l'accumulation des portes AND.

Comme dans le cas des compteurs asynchrones, ces retards limitent la fréquence maximum de comptage. A l'inverse, il faut signaler que les états transitoires n'existent quasiment plus!

Logique séquentielle – Compteurs, registres et mémoires

Compteurs Synchrones - (3/6)

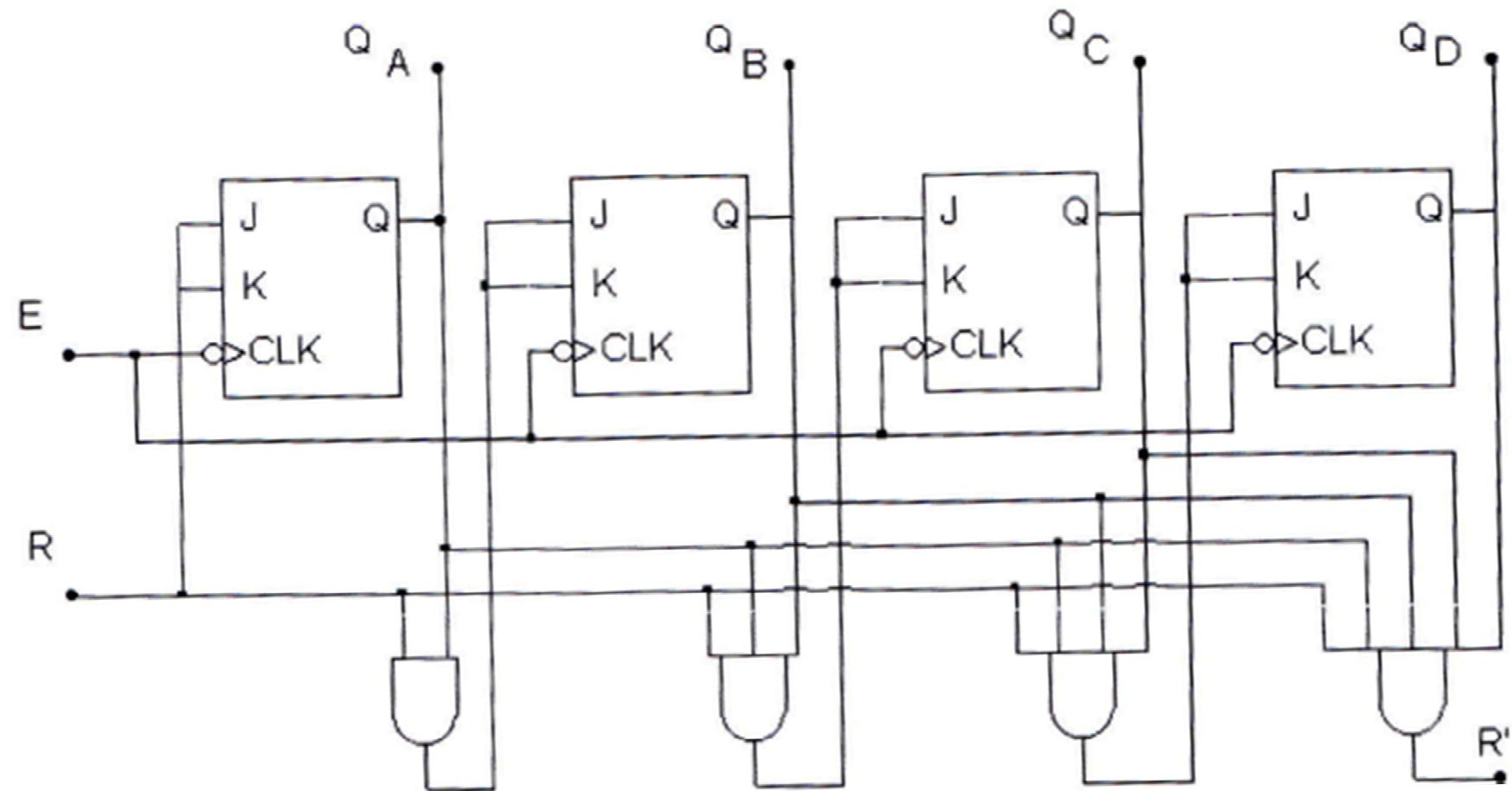
Compteurs binaires à retenue série – Schéma



Logique séquentielle – Compteurs, registres et mémoires

Compteurs Synchrones - (4/6)

Compteurs binaires à retenue parallèle – Schéma



Logique séquentielle – Compteurs, registres et mémoires

Compteurs Synchrones - (5/6)

Compteurs Synchrone par 10 (1/2)

Il est toujours possible d'utiliser l'entrée CLEAR, présente sur chaque compteur, pour remettre à zéro un compteur au cours d'un comptage.

Selon les compteurs, cette remise à zéro peut se faire de deux façons:

- ❖ soit en agissant sur l'entrée CLR de chaque bascule;
- ❖ soit en imposant $J=0$ et $K=1$ sur chacune des bascules.

Dans le premier cas, la remise à zéro est immédiate et indépendante des impulsions arrivant sur le compteur: **on parle alors de remise à zéro asynchrone.**

Dans le second cas la remise à zéro ne sera effective que au moment de la prochaine impulsion arrivant sur le compteur: **la remise à zéro est alors synchrone.**

Logique séquentielle – Compteurs, registres et mémoires

Compteurs Synchrones - (6/6)

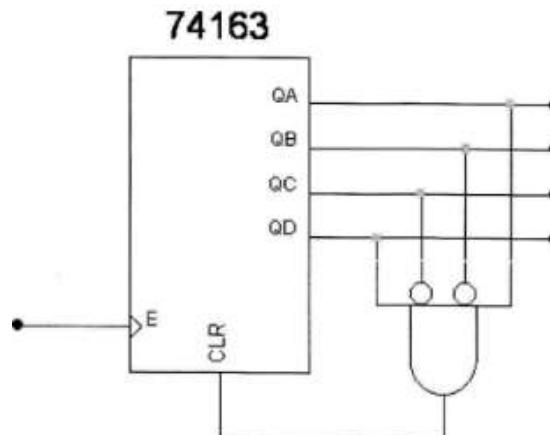
Compteurs Synchrone par 10 (2/2)

Les compteurs disposant d'une remise à zéro synchrone permettent de réaliser facilement des cycles de comptage par $N \neq 2^n$.

Par exemple, pour réaliser un compteur par 10, il suffit de décoder en sortie la valeur 9 ($Q_D Q_C Q_B Q_A = 1001$) et de rétroagir sur l'entrée CLEAR synchrone du compteur.

L'impulsion suivante entraînera alors la remise à zéro du compteur. On réalise de la même façon des compteurs par 7, 5, 11.... La figure ci-dessous propose un exemple de compteur par 10 réalisé à partir d'un compteur synchrone par 16 de référence 74163.

Dans le cas des compteurs asynchrones, la réalisation de cycles de comptage dans des codes autres que le code binaire standard (CBN) nécessite une synthèse complète du compteur.

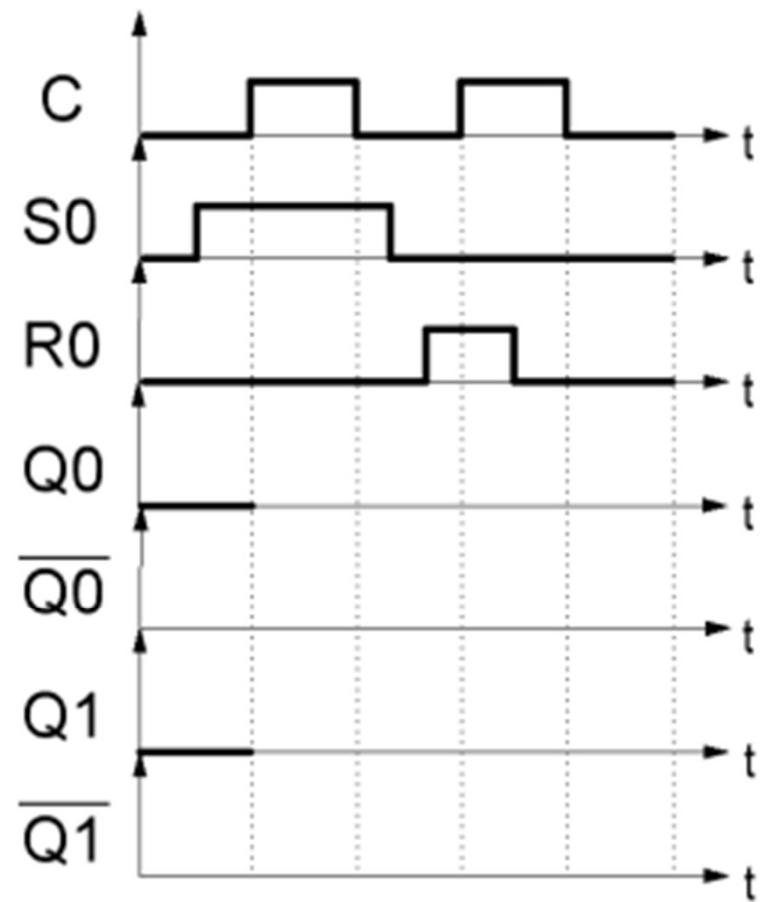
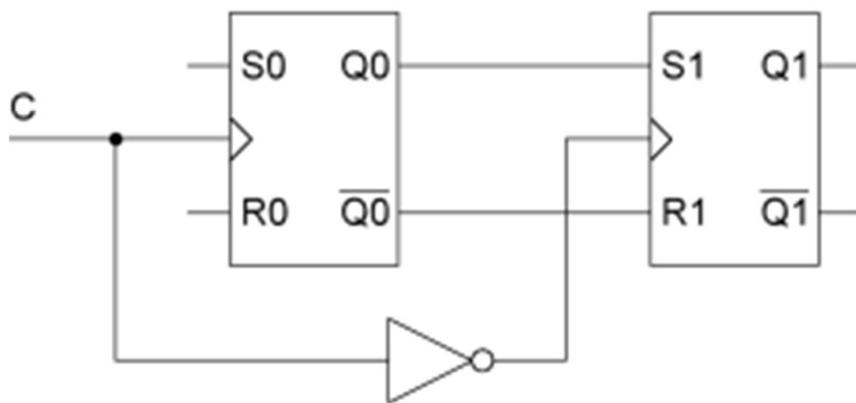


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules RS

1. Complétez le chronogramme du circuit ci-dessous. Si l'on considère la totalité de ce circuit comme une seule bascule RS, quel est son mode de synchronisation ?

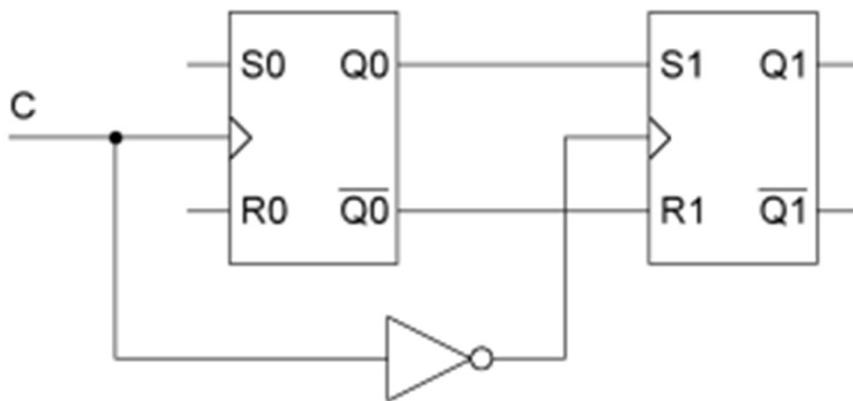


Logique séquentielle – Compteurs, registres et mémoires

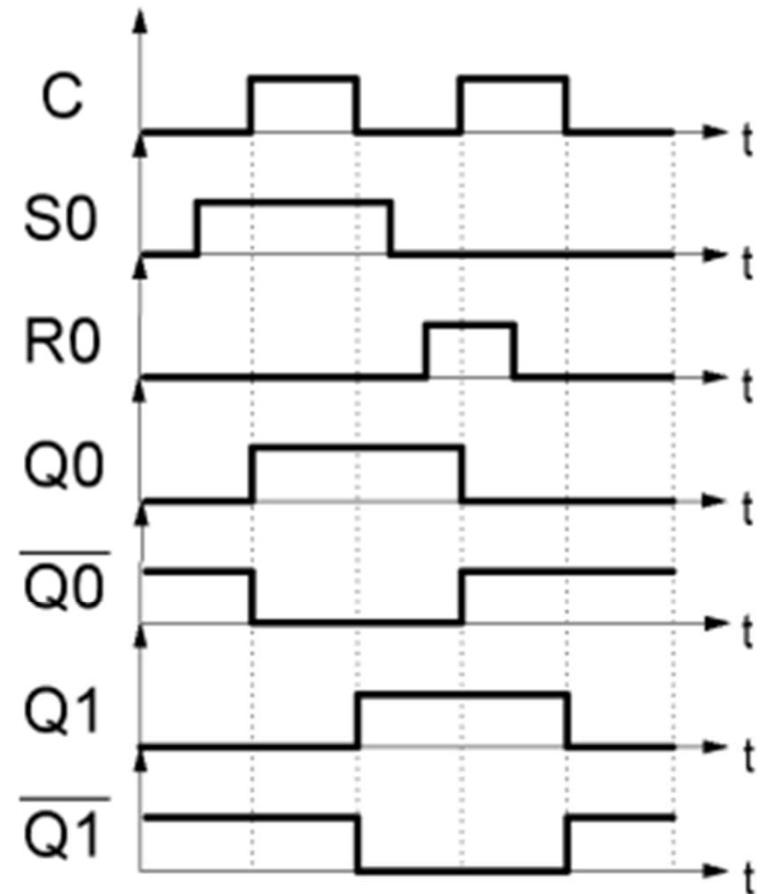
Exercices d'application à faire en séance ou devoir maison

Bascules RS (correction)

1. Complétez le chronogramme du circuit ci-dessous. Si l'on considère la totalité de ce circuit comme une seule bascule RS, quel est son mode de synchronisation ?



Ce circuit est une bascule RS synchronisée sur impulsion (bascule RS maître-esclave).

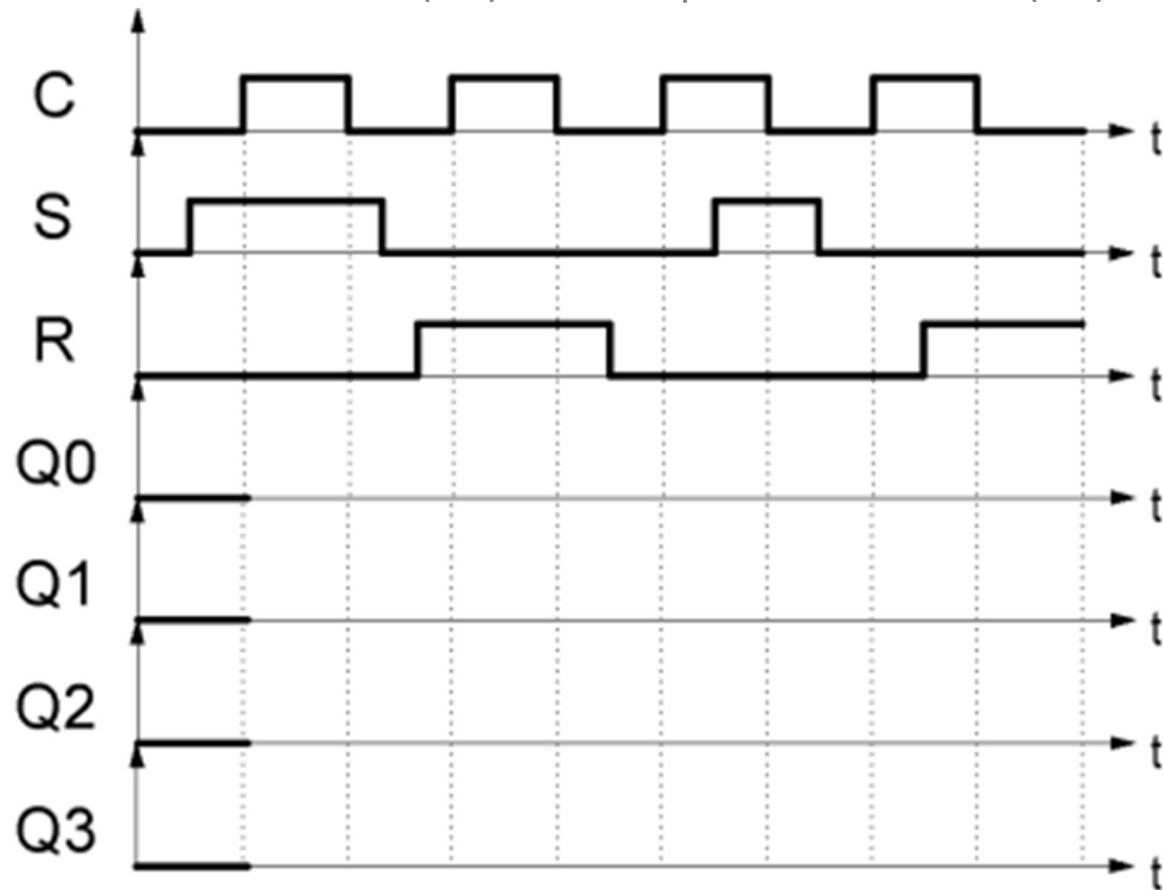


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules RS

2. Complétez les chronogrammes suivants selon que la bascule RS est synchronisée sur état haut (Q0), sur front montant (Q1), sur front descendant (Q2) et sur impulsion ou niveau (Q3).

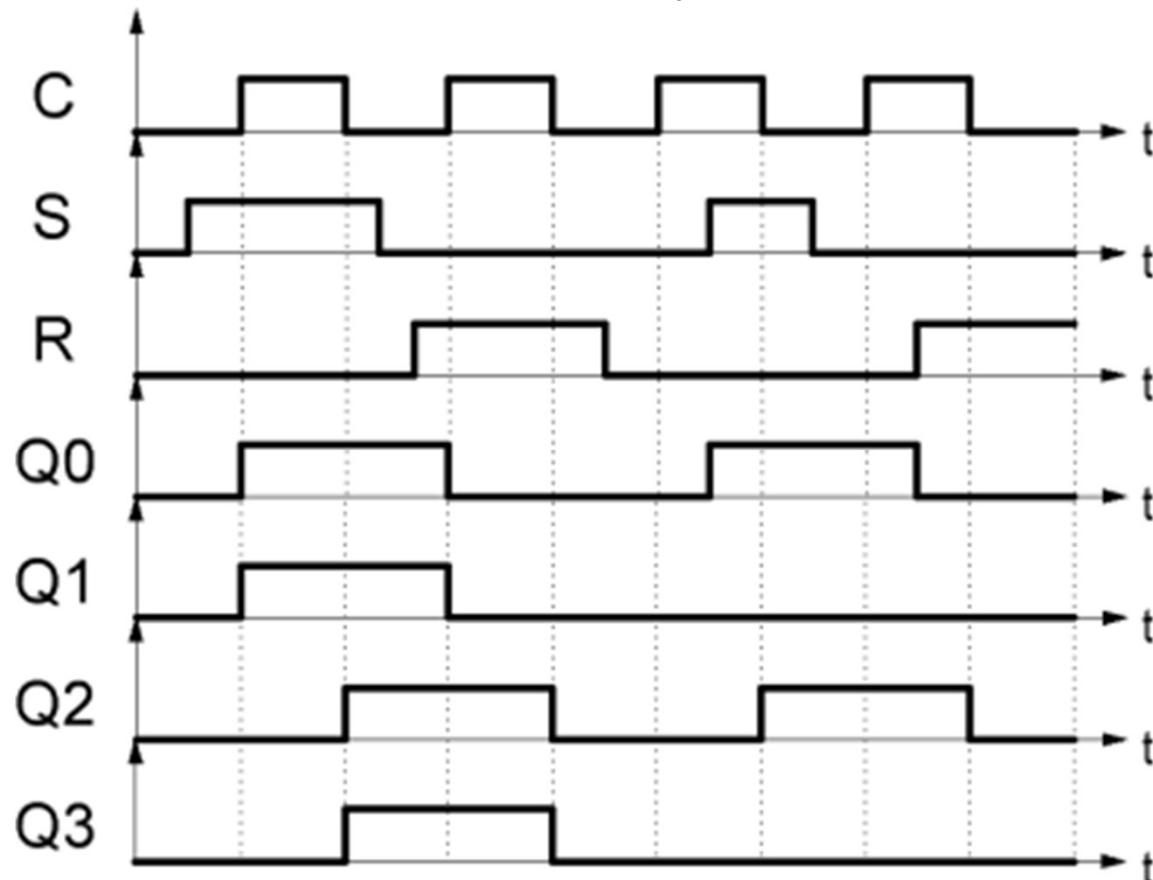


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules RS (correction)

2. Complétez les chronogrammes suivants selon que la bascule RS est synchronisée sur état haut (Q0), sur front montant (Q1), sur front descendant (Q2) et sur impulsion ou niveau (Q3).

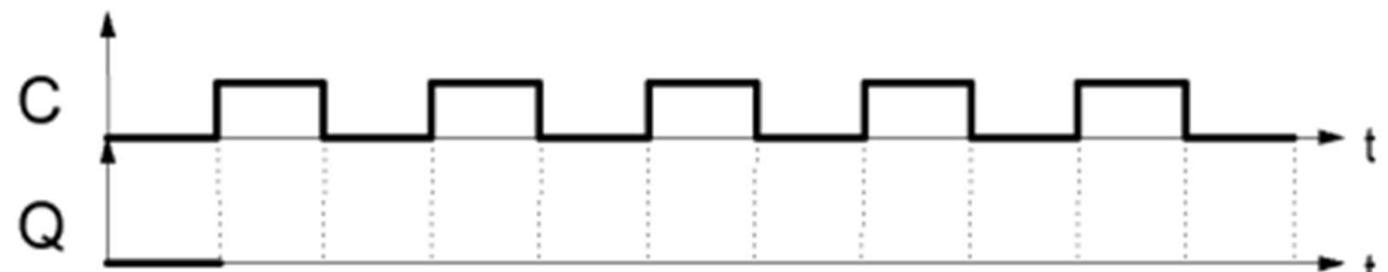
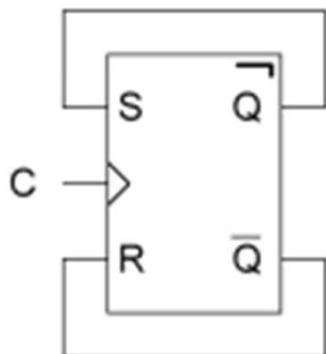
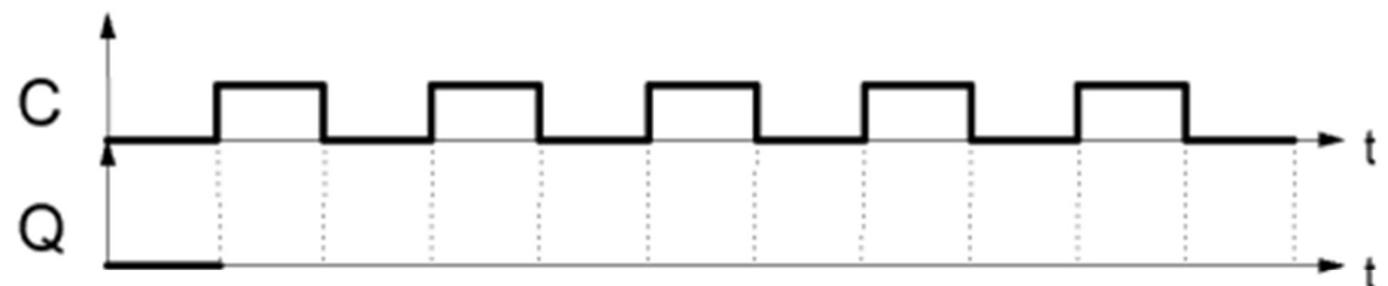
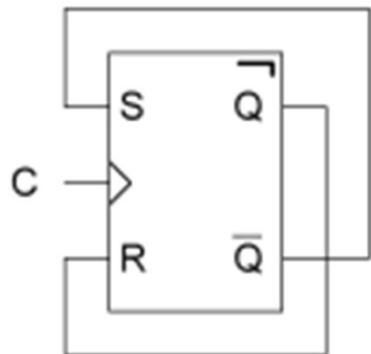


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules RS

3. Tracez le chronogramme de la sortie Q pour chacun des deux circuits ci-dessous. Dans le premier circuit, quel est le rapport entre la fréquence de Q et celle de C ? Comment appelle-t-on ce montage ?

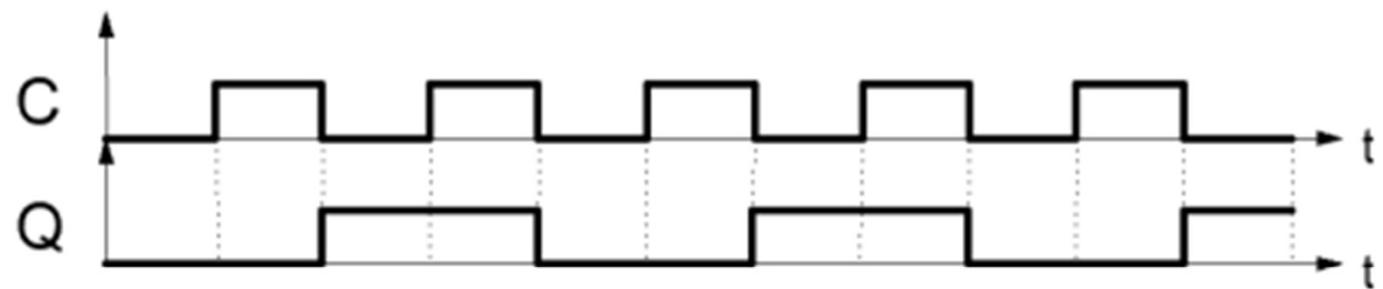
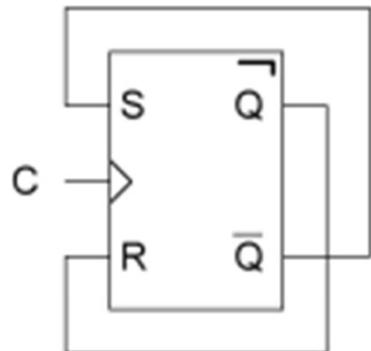


Logique séquentielle – Compteurs, registres et mémoires

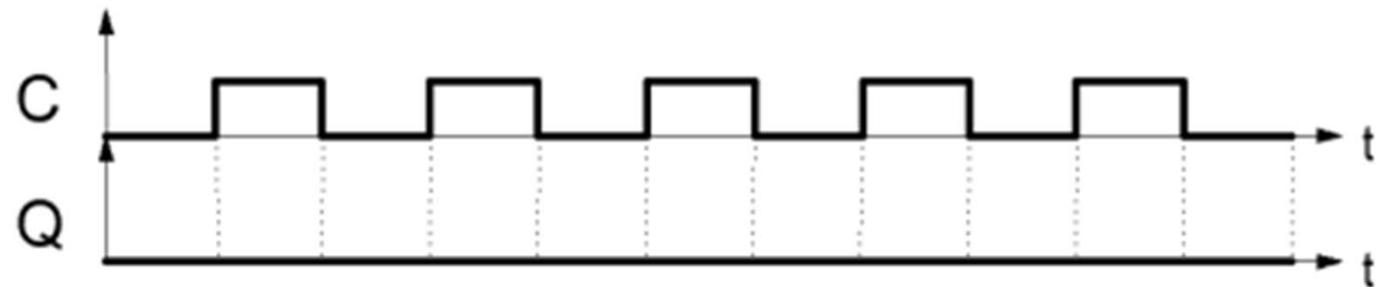
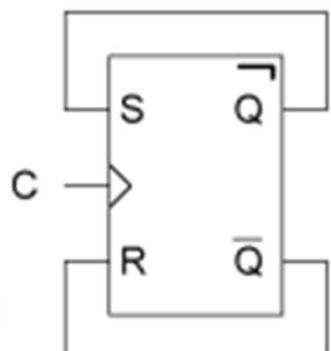
Exercices d'application à faire en séance ou devoir maison

Bascules RS (correction)

3. Tracez le chronogramme de la sortie Q pour chacun des deux circuits ci-dessous. Dans le premier circuit, quel est le rapport entre la fréquence de Q et celle de C ? Comment appelle-t-on ce montage ?



$$f_Q/f_C = \frac{1}{2} \rightarrow \text{Diviseur de fréquence par deux}$$

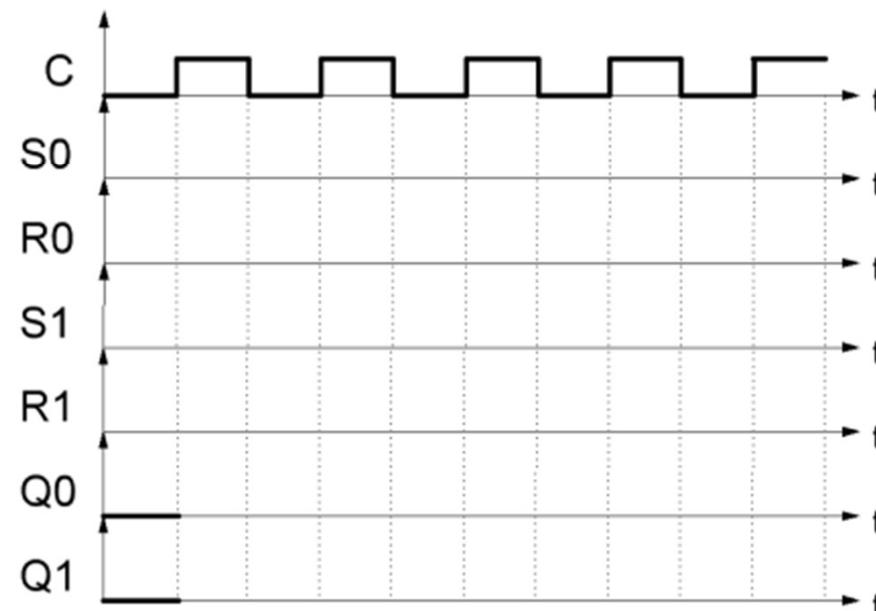
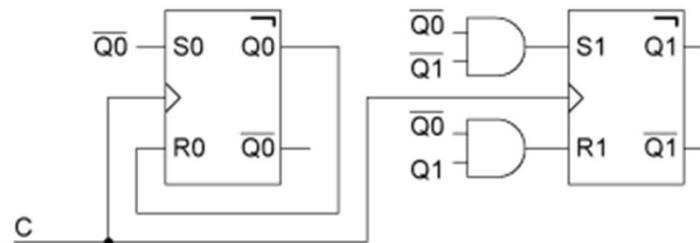


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules RS

4a. Complétez les chronogrammes des circuits ci-dessous.

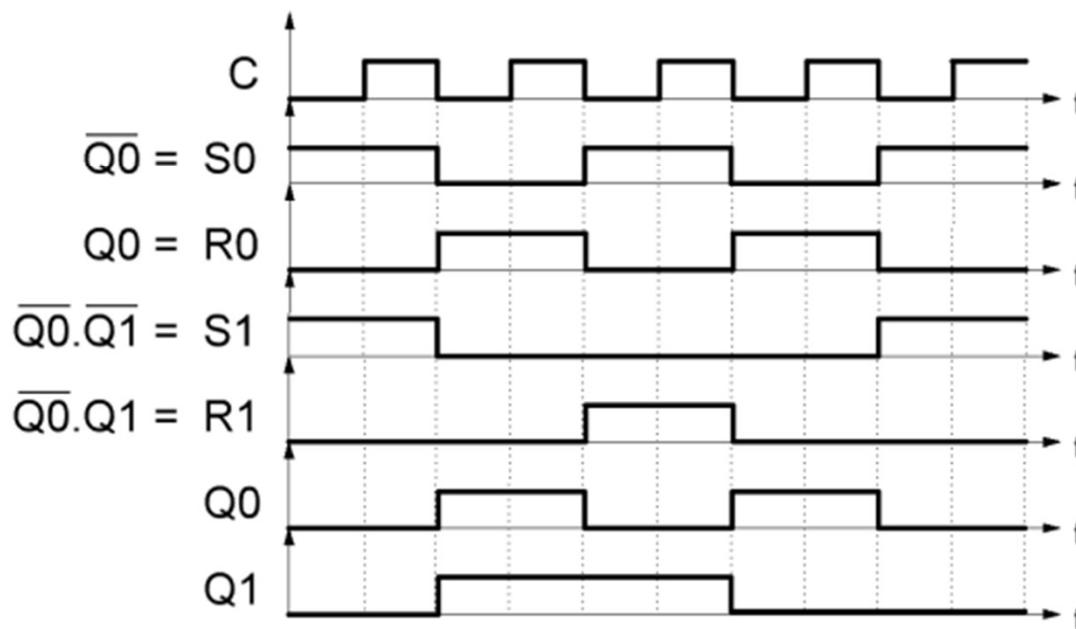
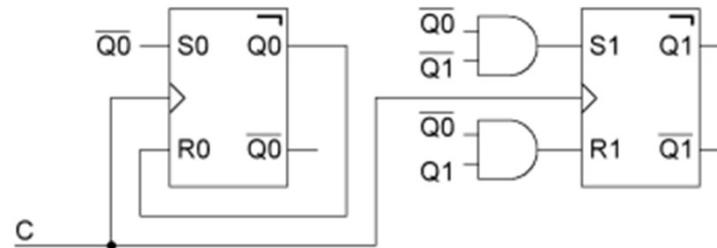


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules RS (correction)

4a. Complétez les chronogrammes des circuits ci-dessous.

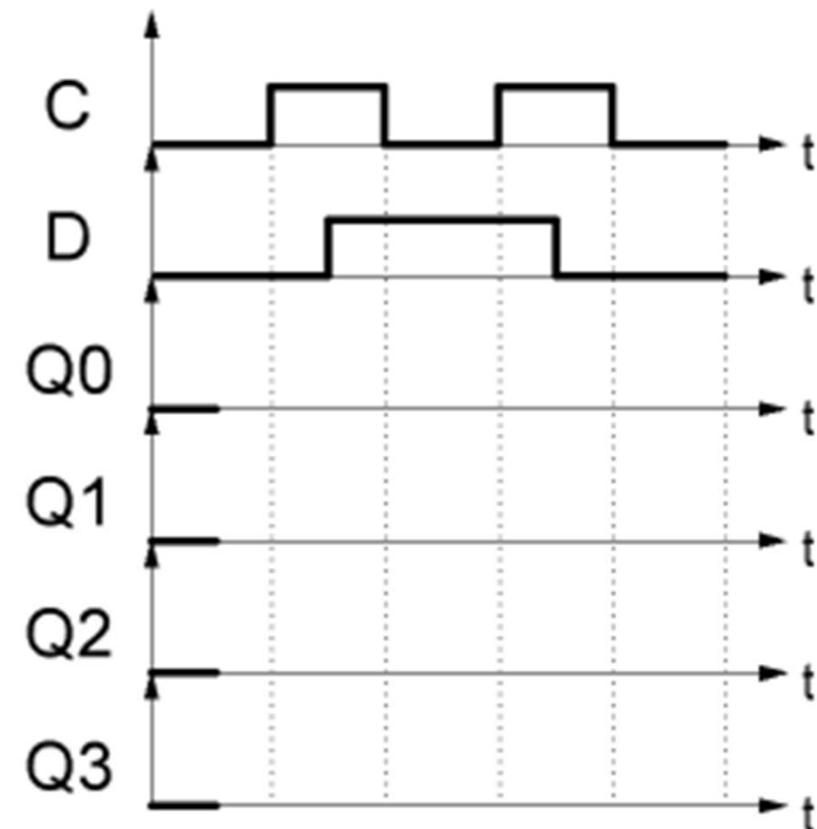
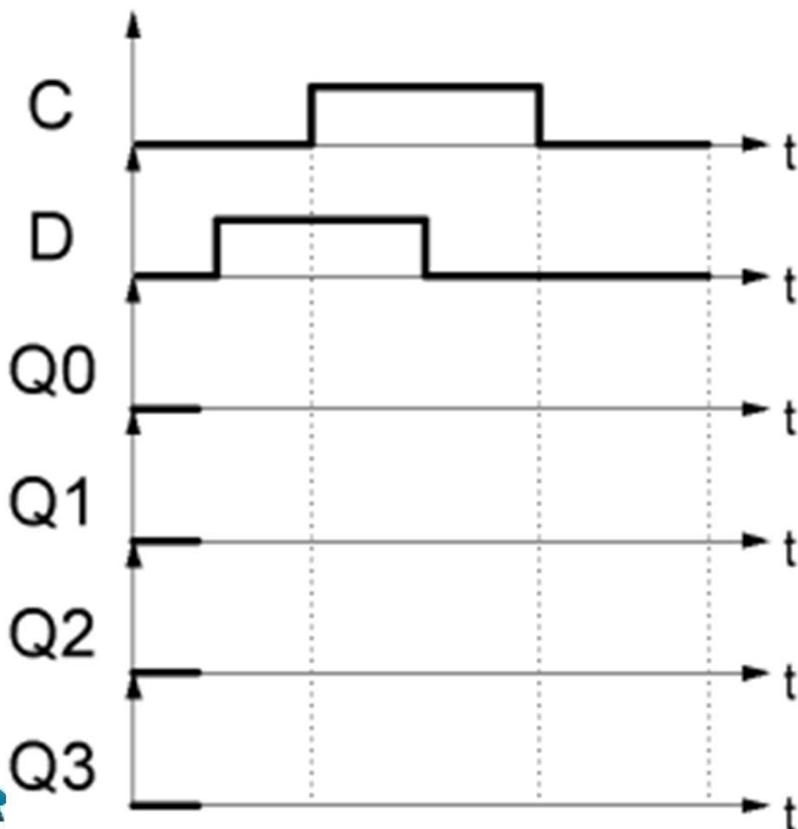


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules D

1. Complétez les chronogrammes suivants selon que la bascule D est synchronisée sur état haut (Q0), sur front montant (Q1), sur front descendant (Q2) et sur impulsion (Q3).

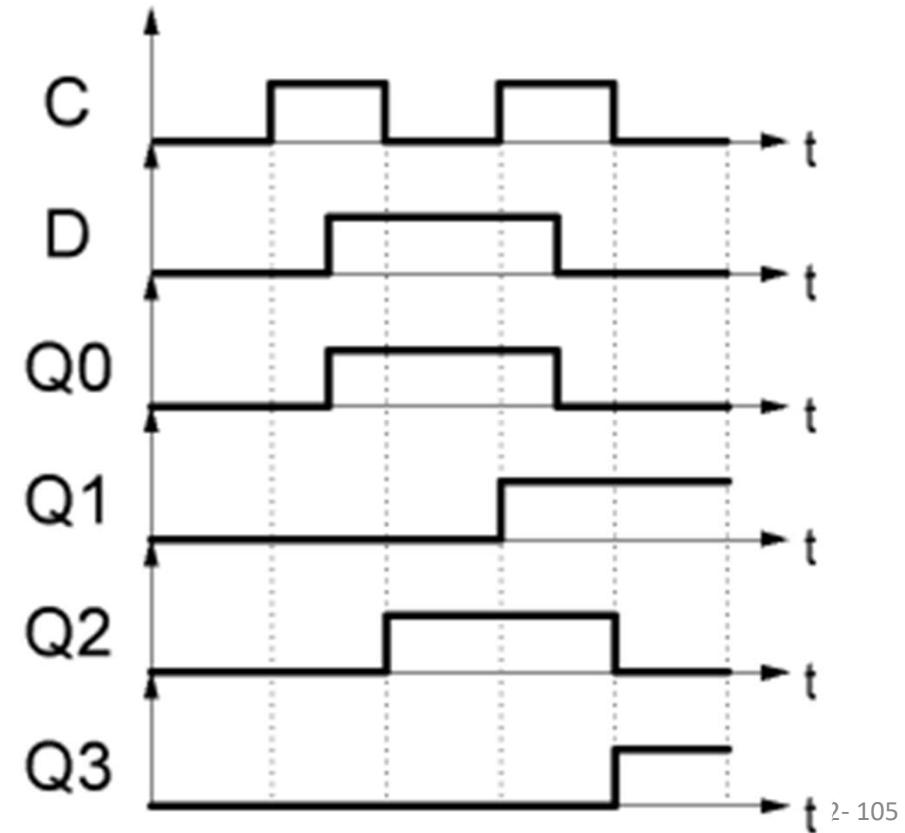
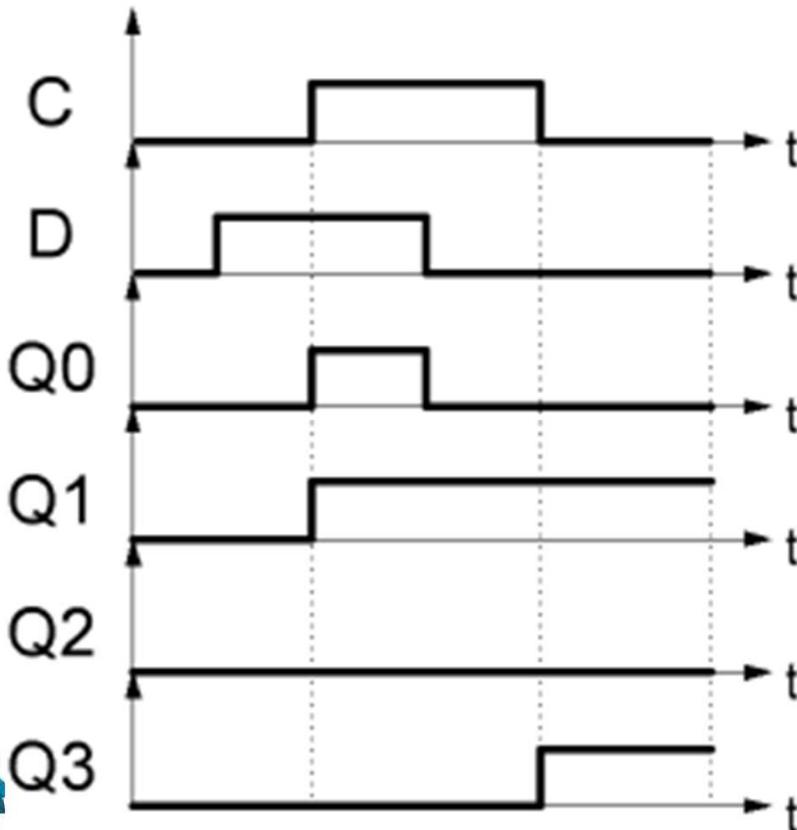


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules D (correction)

1. Complétez les chronogrammes suivants selon que la bascule D est synchronisée sur état haut (Q0), sur front montant (Q1), sur front descendant (Q2) et sur impulsion (Q3).

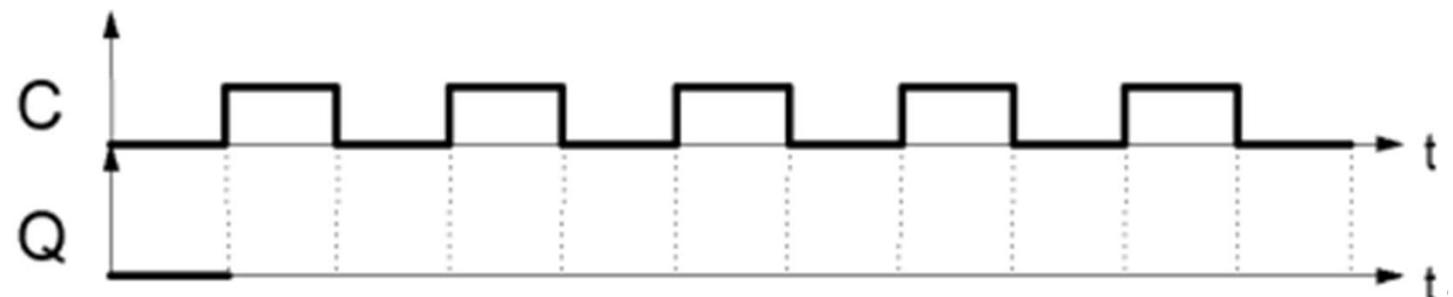
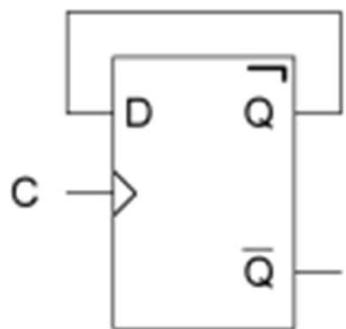
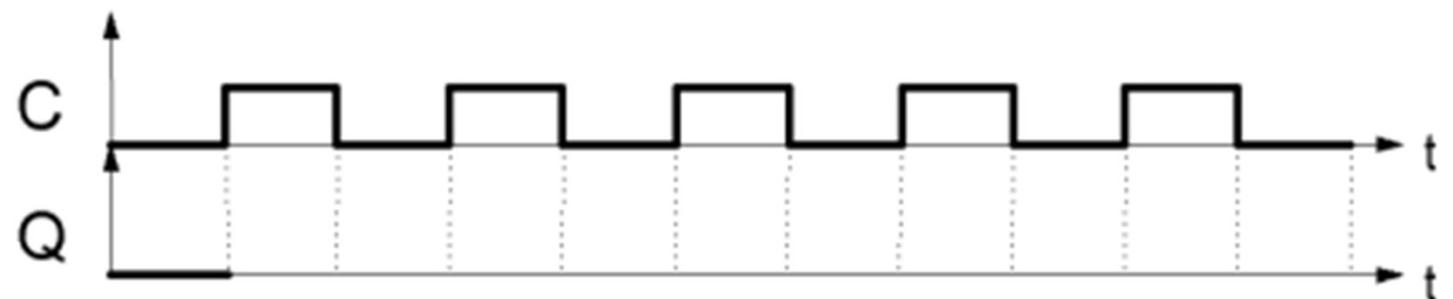
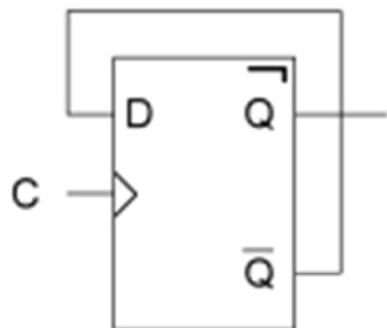


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules D

2. Tracez le chronogramme de la sortie Q pour chacun des deux circuits ci-dessous.

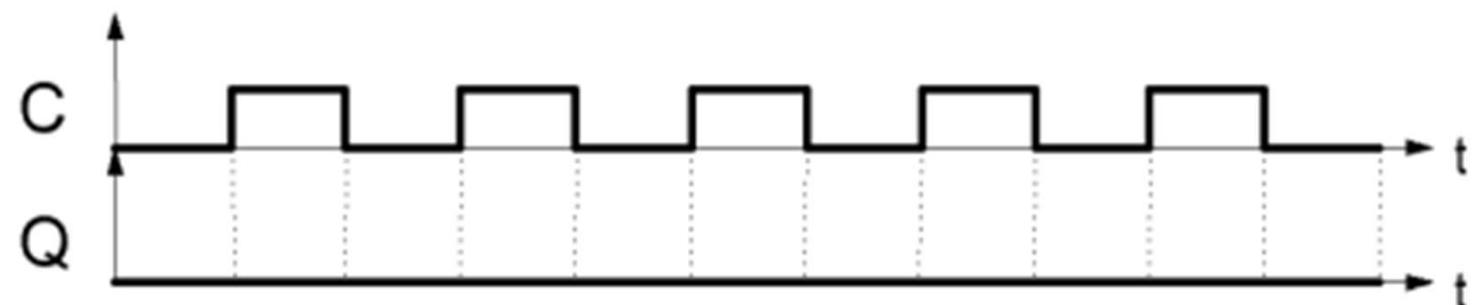
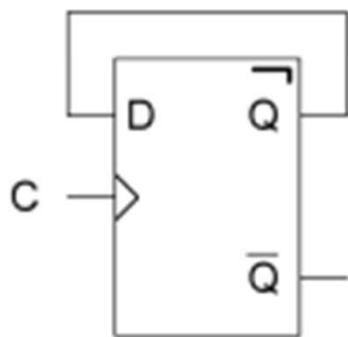
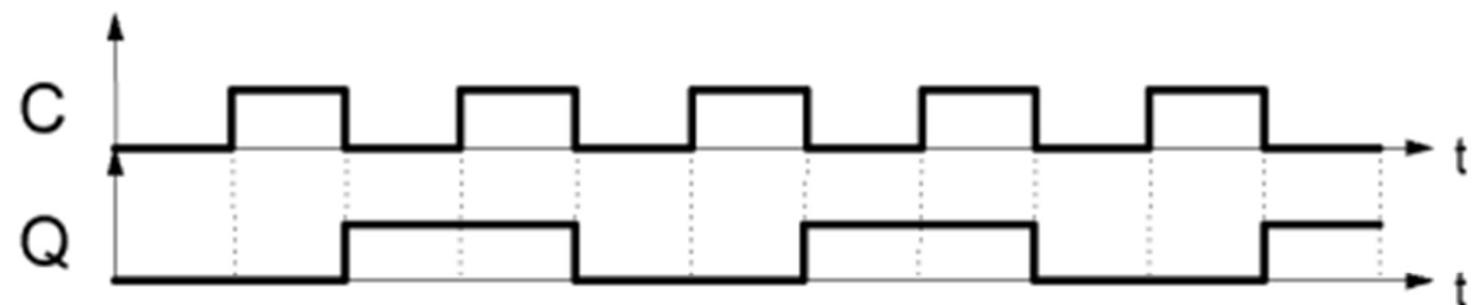
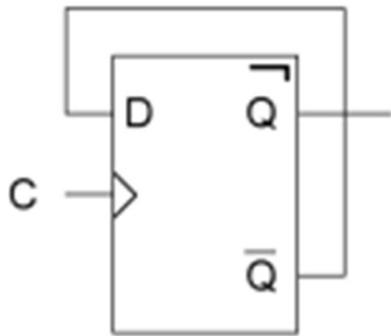


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules D (correction)

2. Tracez le chronogramme de la sortie Q pour chacun des deux circuits ci-dessous.

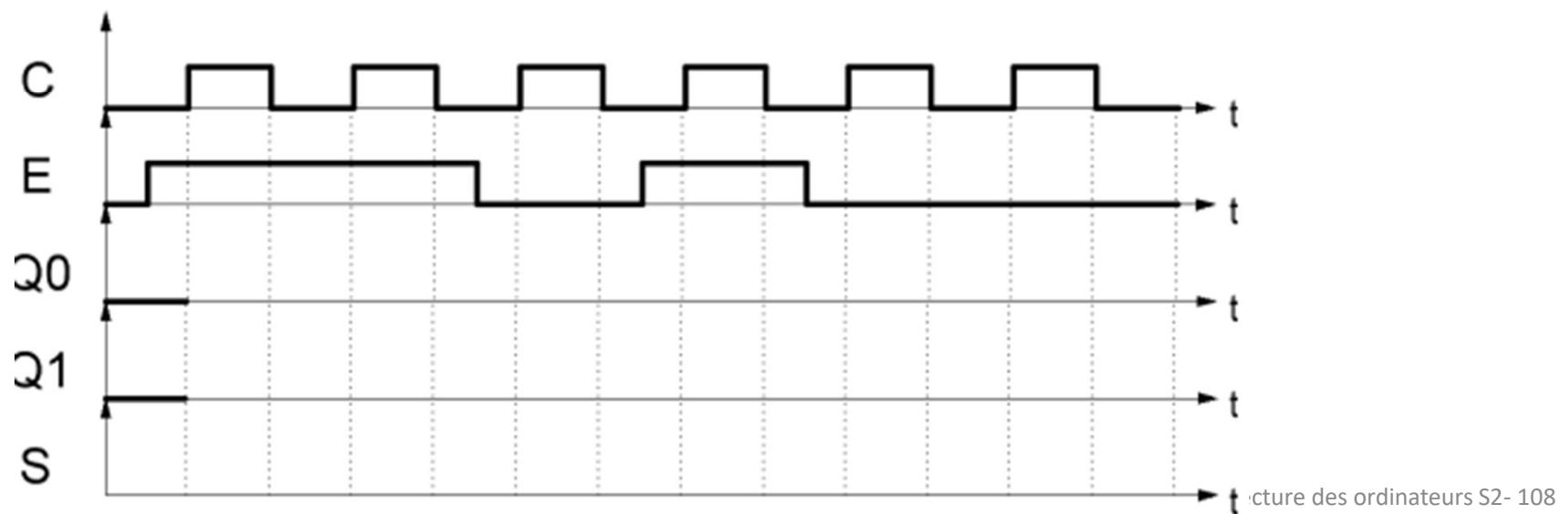
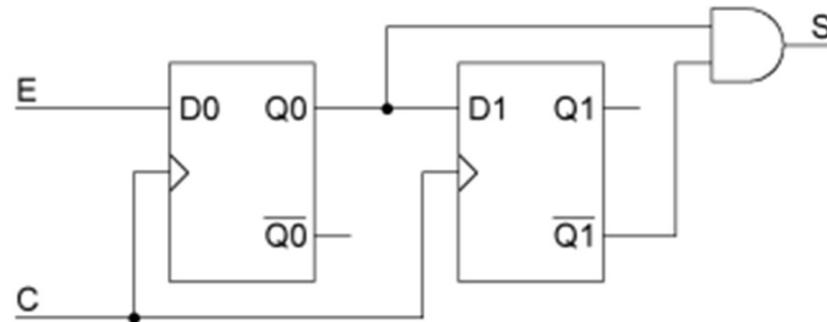


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules D

3. Complétez le chronogramme du circuit suivant.

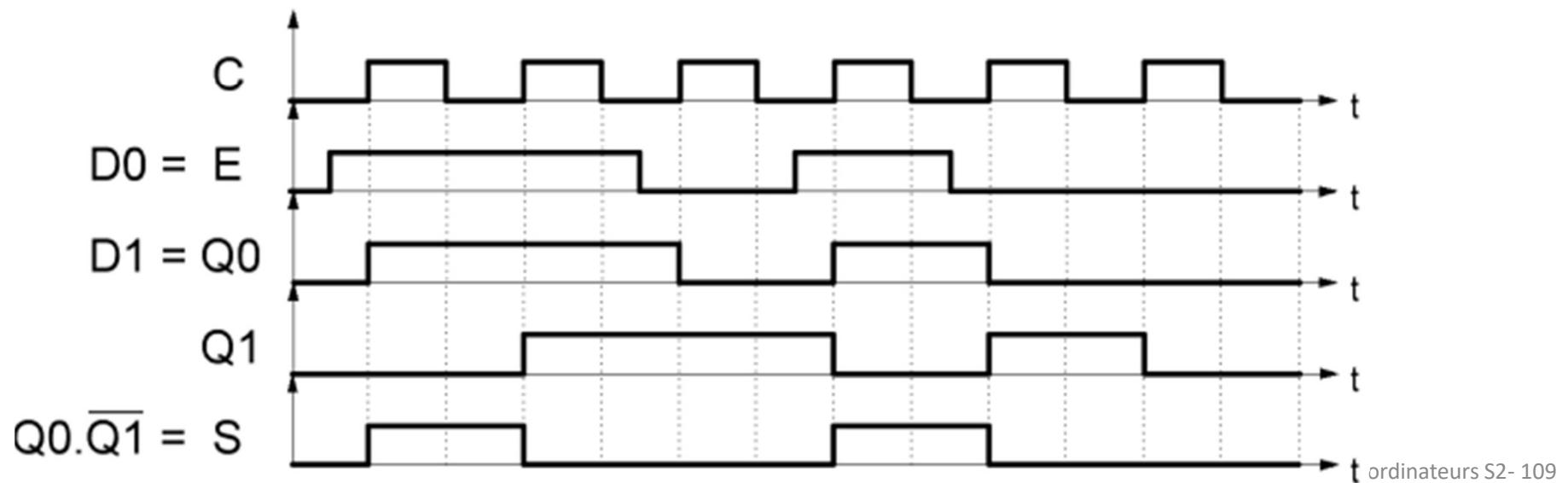
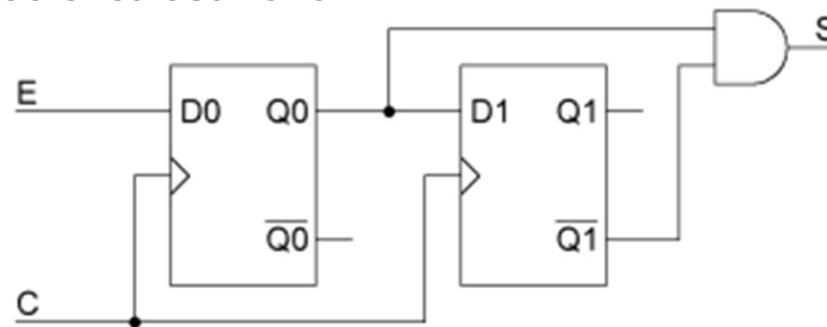


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules D (correction)

3. Complétez le chronogramme du circuit suivant.

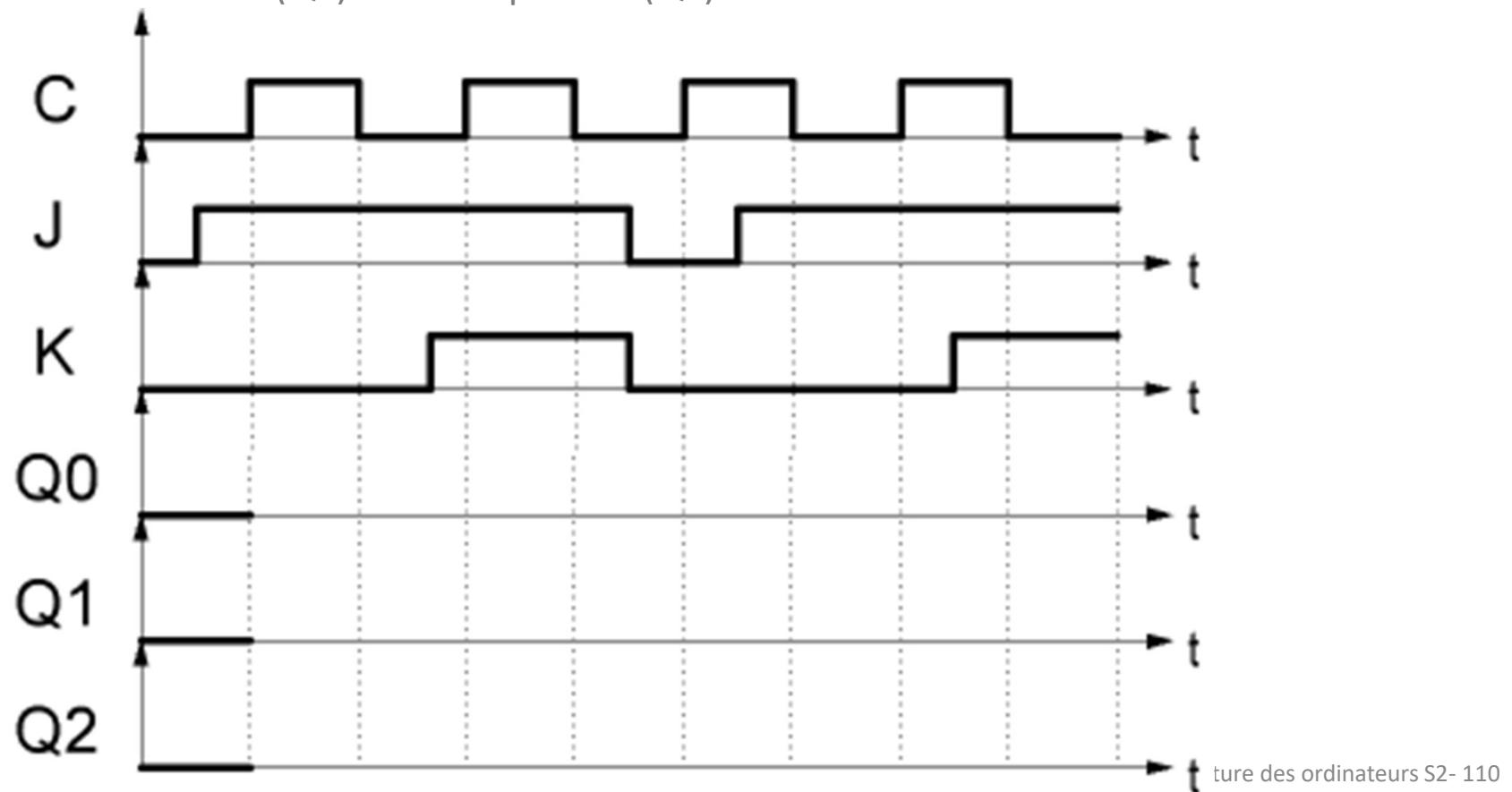


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules JK

1. Complétez les chronogrammes suivants selon que la bascule JK est synchronisée sur front montant (Q0), sur front descendant (Q1) et sur impulsion (Q2).

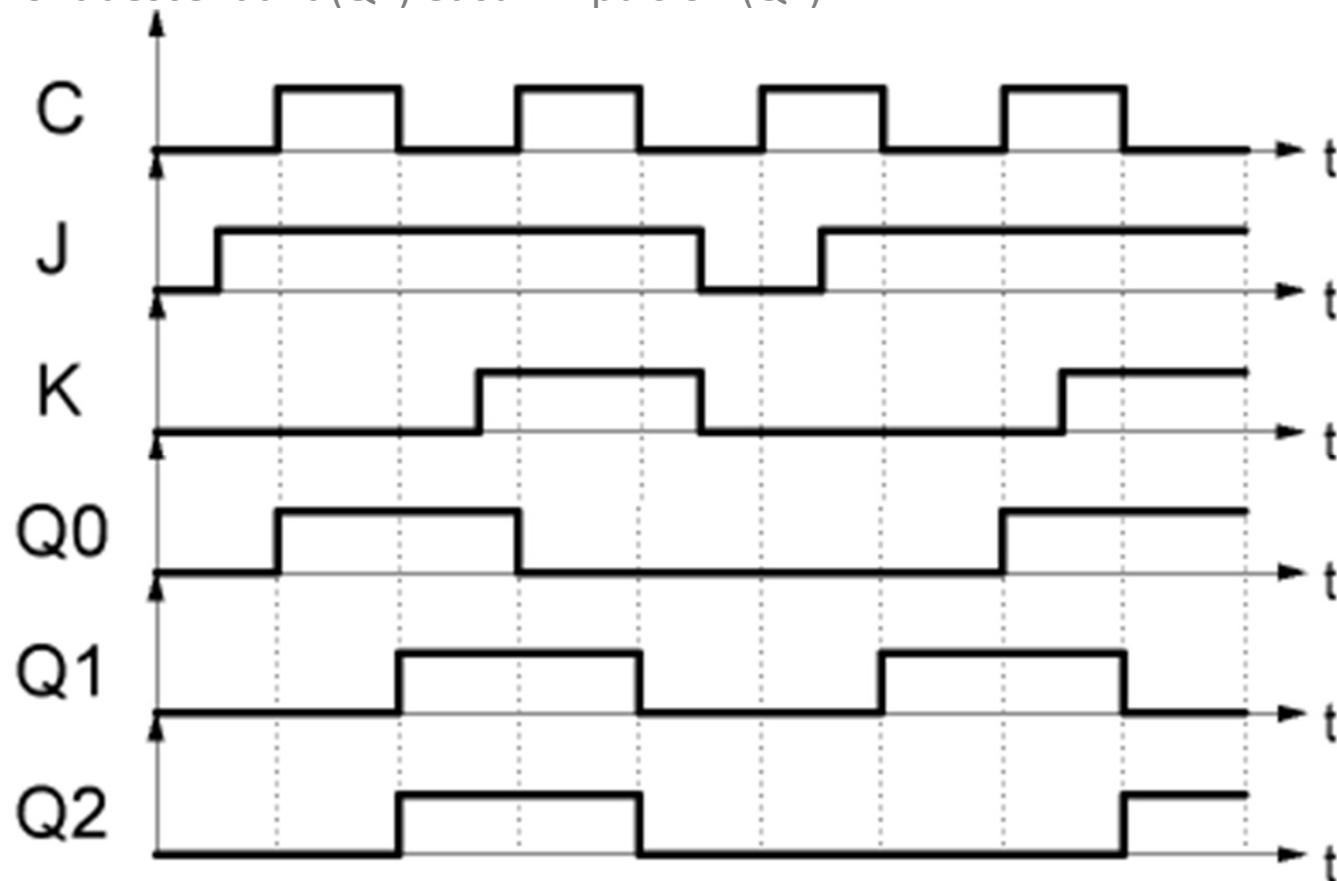


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules JK (correction)

1. Complétez les chronogrammes suivants selon que la bascule JK est synchronisée sur front montant (Q0), sur front descendant (Q1) et sur impulsion (Q2).

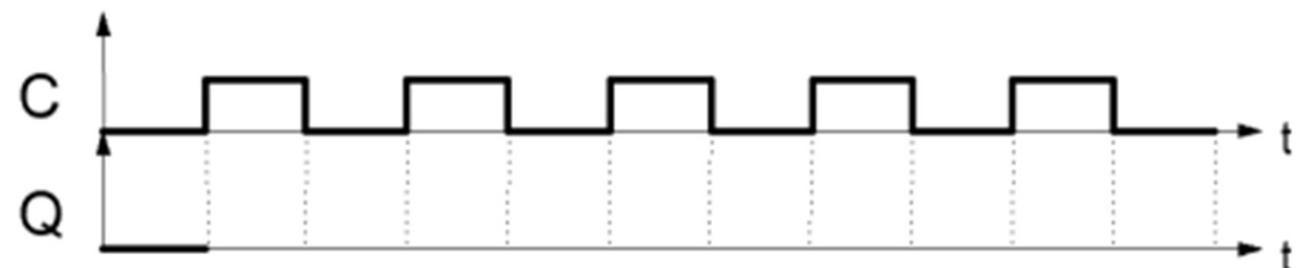
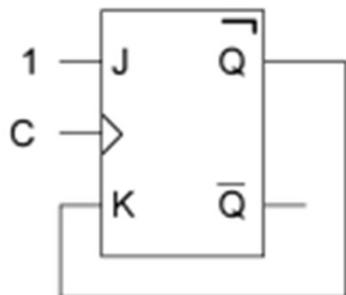
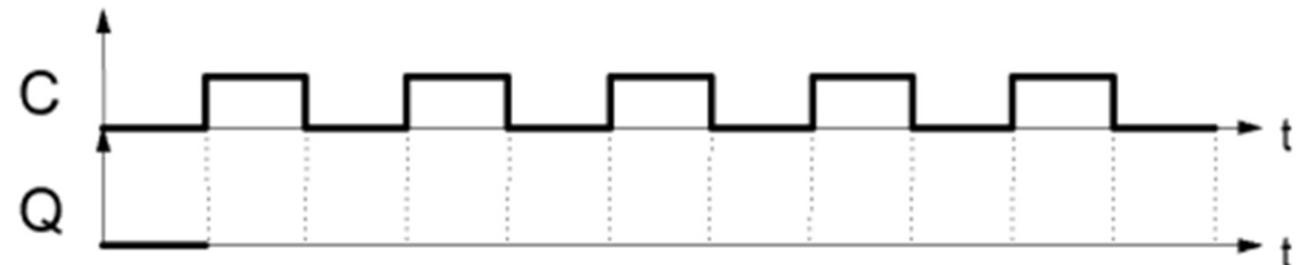
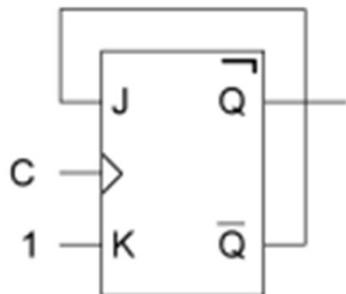


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules JK

2. Tracez le chronogramme de la sortie Q pour chacun des deux circuits ci-dessous. Quel est le rapport entre la fréquence de Q et celle de C ? Comment appelle-t-on ces montages ? Trouvez une autre façon d'obtenir le même rapport entre ces deux fréquences.

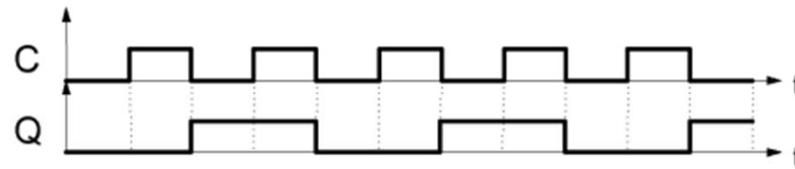
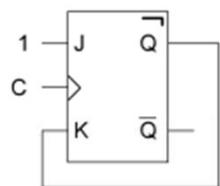
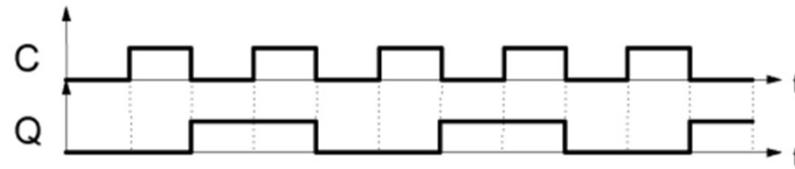
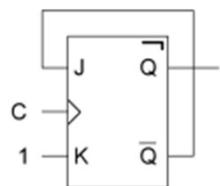


Logique séquentielle – Compteurs, registres et mémoires

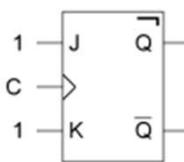
Exercices d'application à faire en séance ou devoir maison

Bascules JK (correction)

2. Tracez le chronogramme de la sortie Q pour chacun des deux circuits ci-dessous. Quel est le rapport entre la fréquence de Q et celle de C ? Comment appelle-t-on ces montages ? Trouvez une autre façon d'obtenir le même rapport entre ces deux fréquences.



$$f_Q/f_C = \frac{1}{2} \rightarrow \text{Diviseur de fréquence par deux}$$

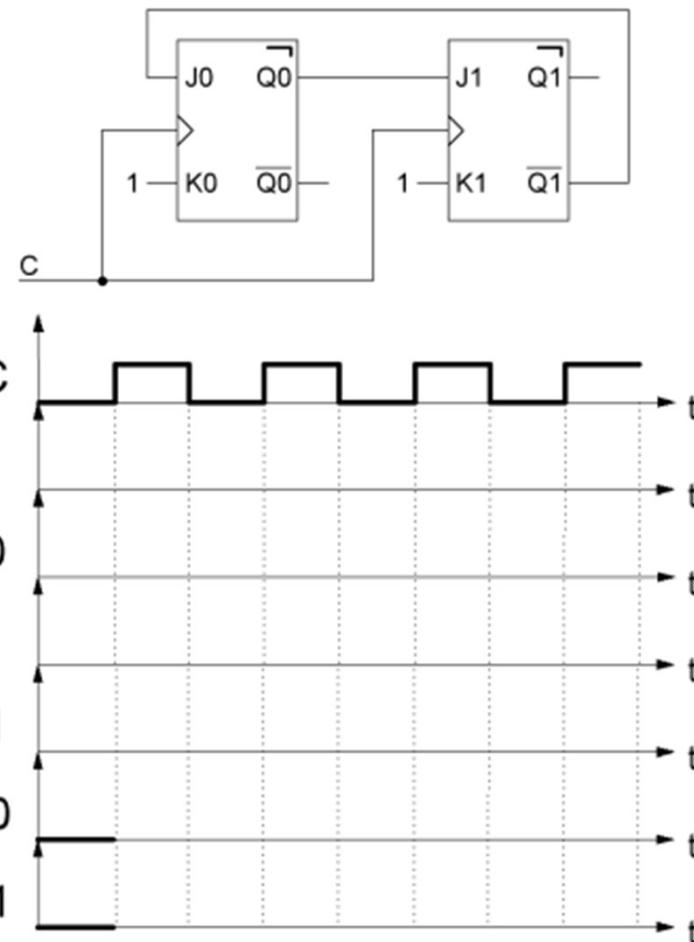


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules JK

3. Complétez les chronogrammes des circuits ci-dessous

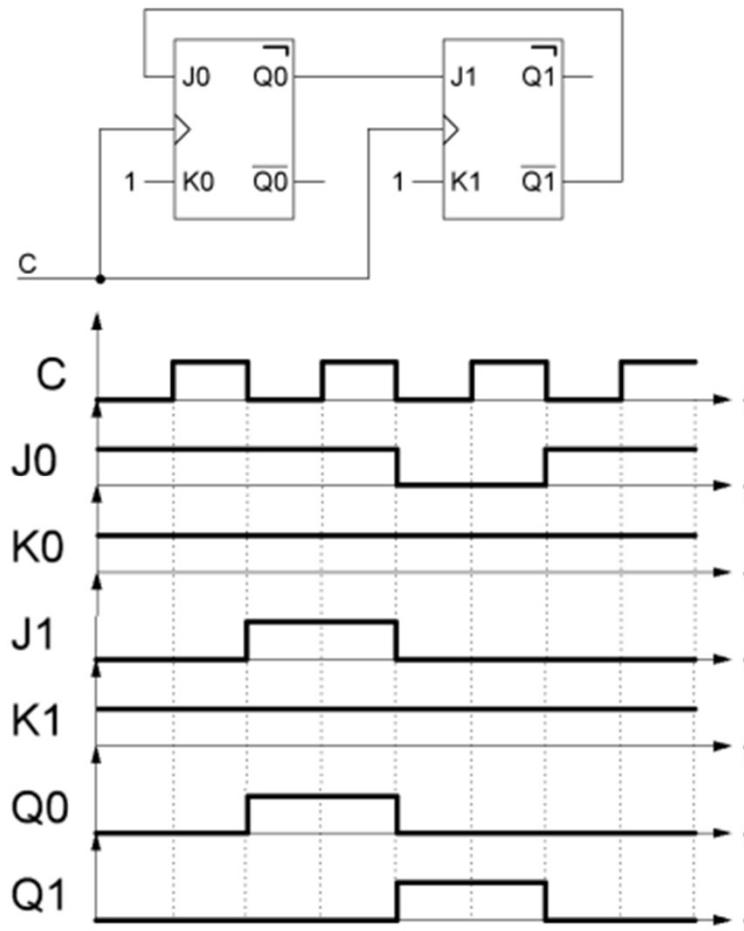


Logique séquentielle – Compteurs, registres et mémoires

Exercices d'application à faire en séance ou devoir maison

Bascules JK (correction)

3. Complétez les chronogrammes des circuits ci-dessous



Logique séquentielle – Compteurs, registres et mémoires

Exercice compteur 1 : À partir du montage de la figure 1, remplissez le chronogramme ci-dessous.
Que réalise le montage de la figure 1 ?

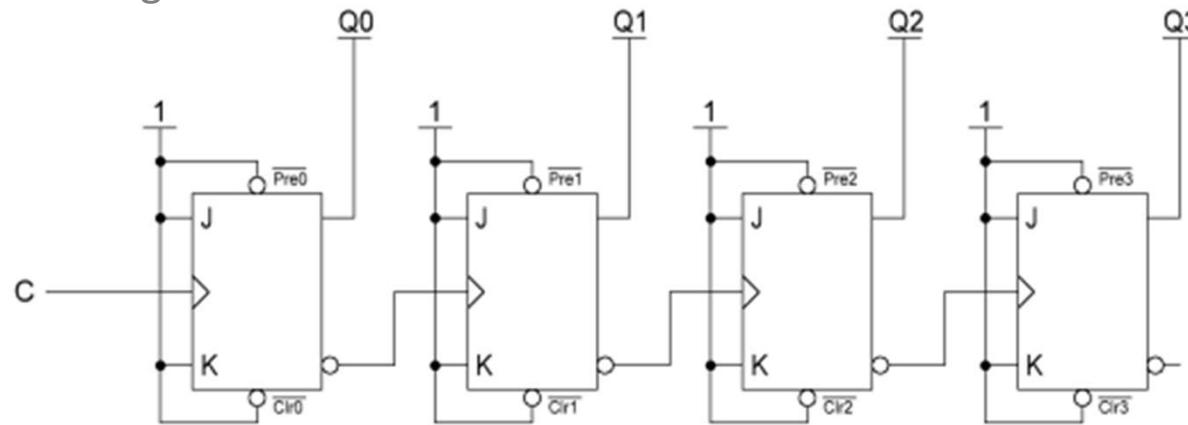
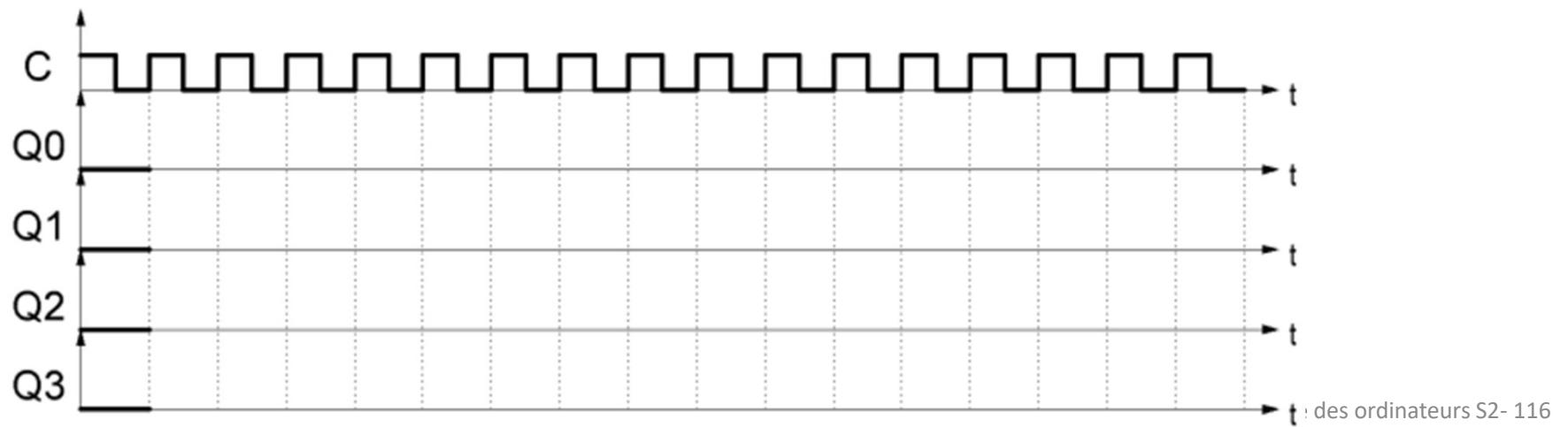


Figure 1



Logique séquentielle – Compteurs, registres et mémoires

Exercices compteur 1 : correction

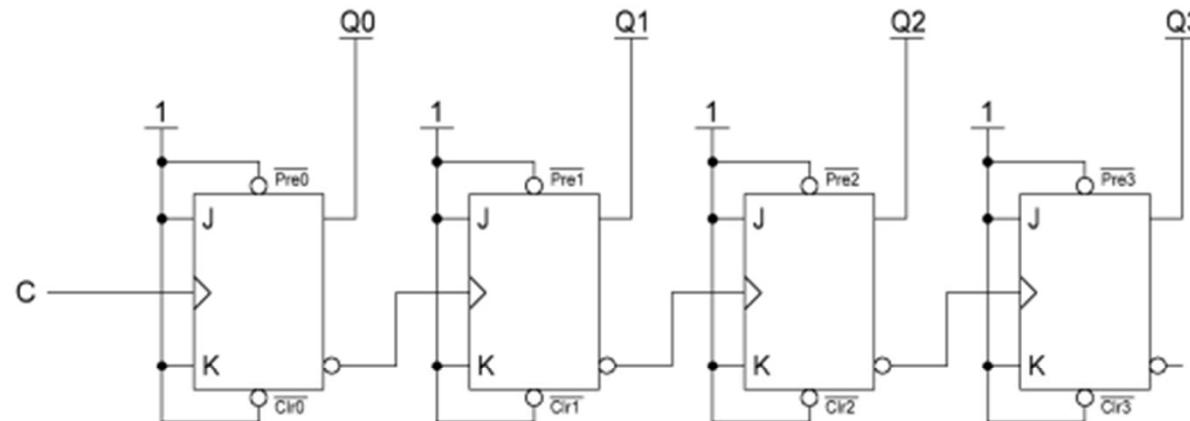
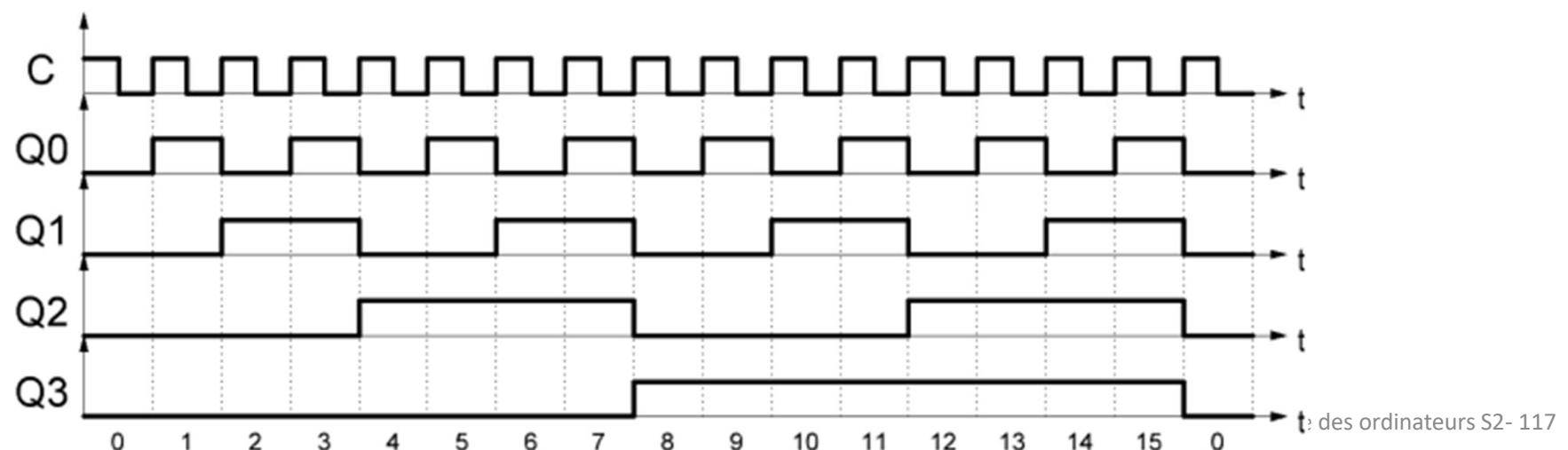


Figure 1



Logique séquentielle – Compteurs, registres et mémoires

Exercices compteur 1 : correction

Les bascules JK sont synchronisées sur front montant et câblées en basculement permanent (J et K sont toujours à 1) :

- ❖ La sortie Q0 bascule sur chaque front montant de C ;
- ❖ La sortie Q1 bascule sur chaque front montant de Q0 (donc chaque front descendant de Q0) ;
- ❖ La sortie Q2 bascule sur chaque front montant de Q1 (donc chaque front descendant de Q1) ;
- ❖ La sortie Q3 bascule sur chaque front montant de Q2 (donc chaque front descendant de Q2).

2. Que réalise le montage de la figure 1 ?

À chaque front d'horloge, la valeur présente sur les sorties est incrémentée de un. Ce montage est un compteur asynchrone modulo 16. Il compte de 0 à 15.

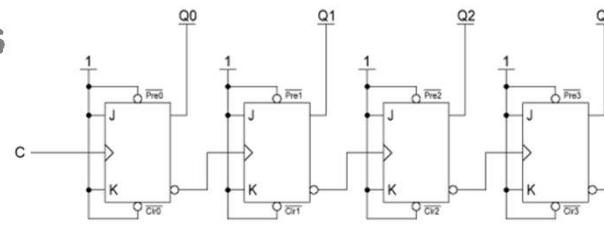


Figure 1

Logique séquentielle – Compteurs, registres et mémoires

Exercice compteur 2: correction on modifie légèrement le montage de la figure 1 afin d'obtenir le montage de la figure 2. En expliquant votre raisonnement, que réalise le montage de la figure 2 ?

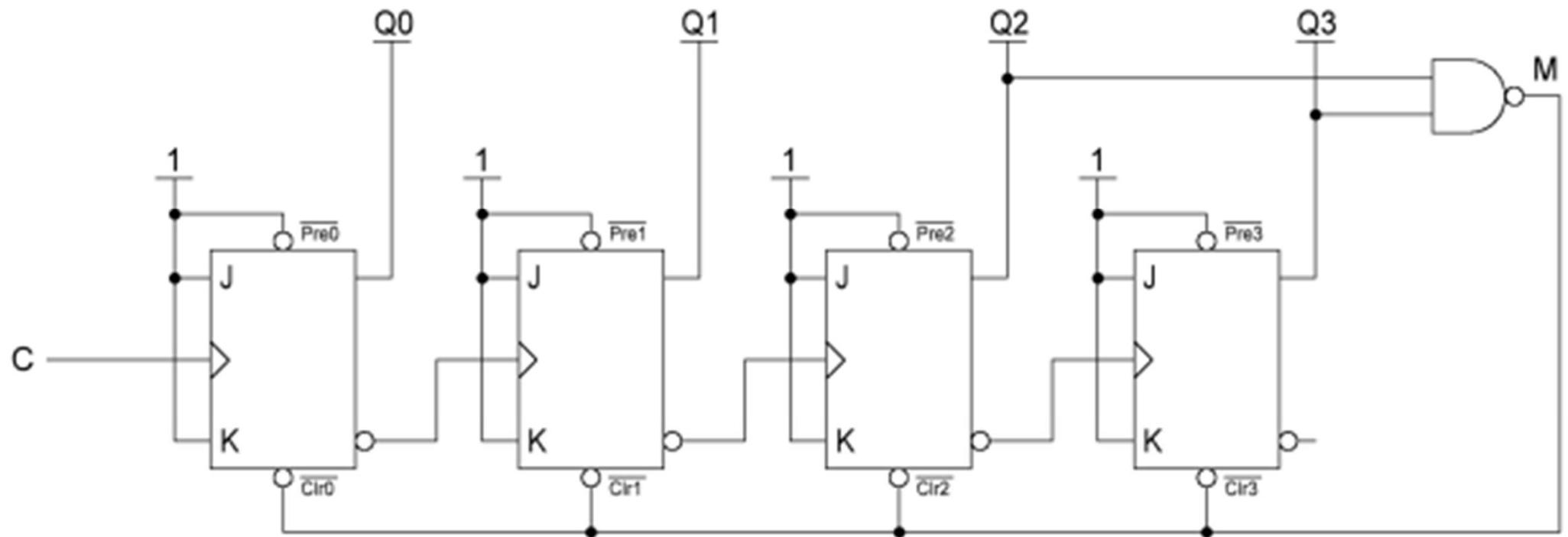


Figure 2

Logique séquentielle – Compteurs, registres et mémoires

Exercice compteur 2: correction

La porte NON-ET sert à détecter la valeur 12 et à la remplacer par la valeur 0.

Soit M, la sortie de la porte NON-ET. Pour rappel, la sortie d'une porte

NON-ET est à 0 uniquement lorsque ses deux entrées sont à 1. M passera donc à 0 lorsque Q2 et Q3 seront à 1 en même temps. Le passage de M à 0 aura pour effet de provoquer un reset sur le compteur et donc de le faire repartir à 0.

Les sorties Q2 et Q3 passent à 1 pour la première fois sur la valeur 12. Le reset s'effectue donc au moment où le compteur atteint la valeur 12. Cette valeur ne reste pas et est immédiatement remplacée par la valeur 0. M repasse alors à 1 et le compteur se remet à compter.

Le temps d'apparition de la valeur 12 se détermine en fonction du temps de réaction de la porte

NON-ET et des bascules JK. En pratique, ce temps es

La valeur 12 est détectée et remplacée par la valeur modulo 12. Il compte de 0 à 11.

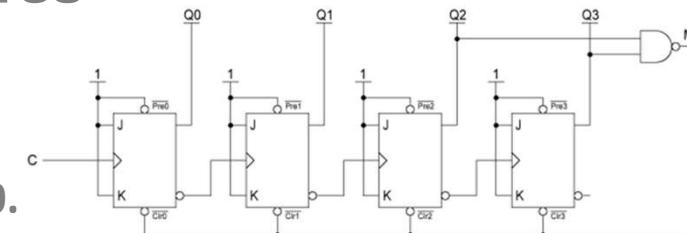


Figure 2

Q	Q3	Q2	Q1	Q0	M
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
⋮	⋮	⋮	⋮	⋮	⋮
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
0	0	0	0	0	1
1	0	0	0	1	1

← Q2 et Q3 sont à 1 : activation du *reset*.

← La valeur 12 est immédiatement remplacée par la valeur 0.

Logique séquentielle – Compteurs, registres et mémoires

Exercice compteur 2: À partir du montage de la figure 3, remplissez le chronogramme ci-dessous.
Que réalise le montage de la figure 3 ?

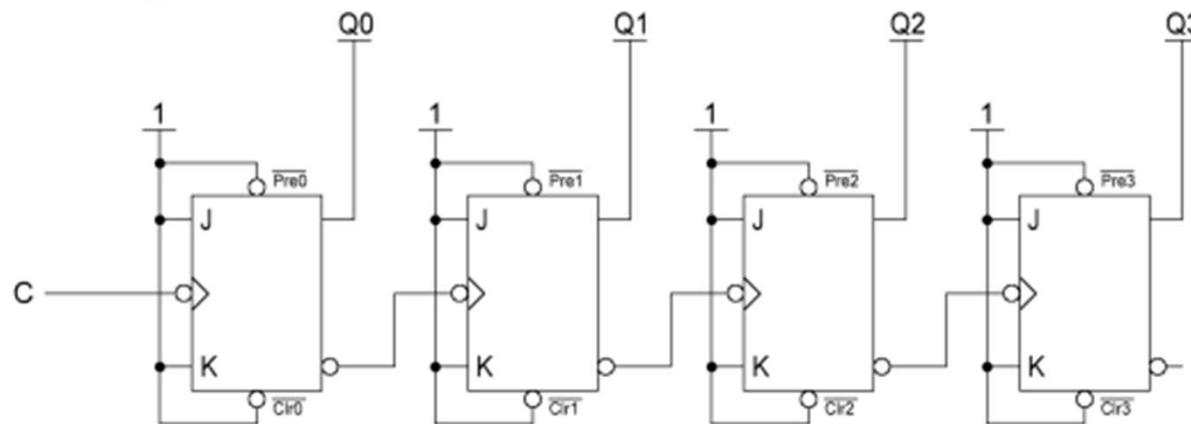
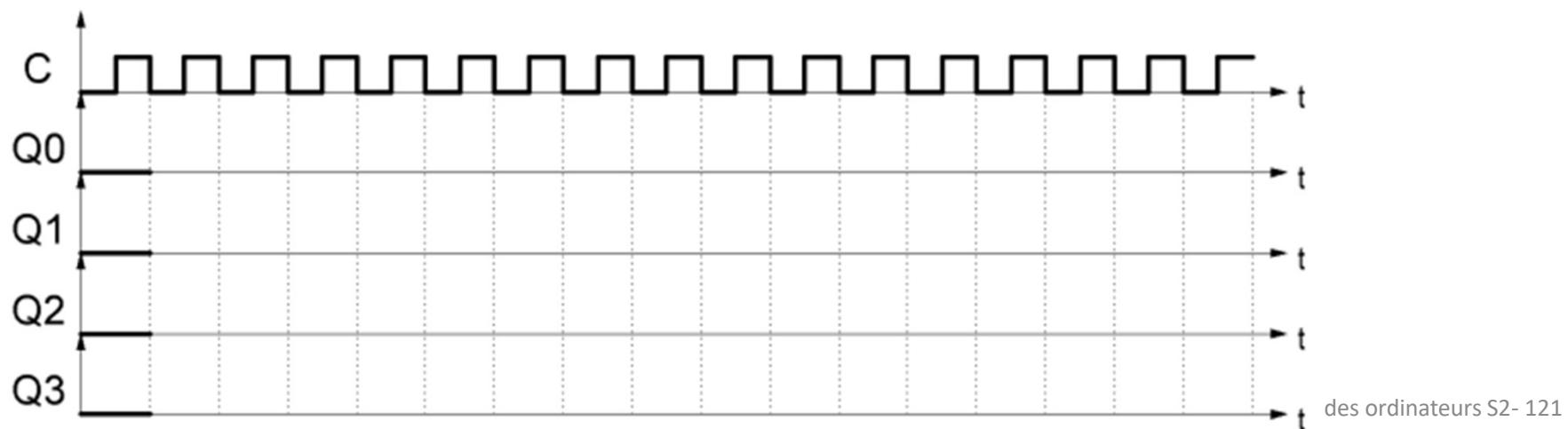


Figure 3



Logique séquentielle – Compteurs, registres et mémoires

Exercice compteur 2: correction À partir du montage de la figure 3, remplissez le chronogramme ci-dessous. Que réalise le montage de la figure 3 ?

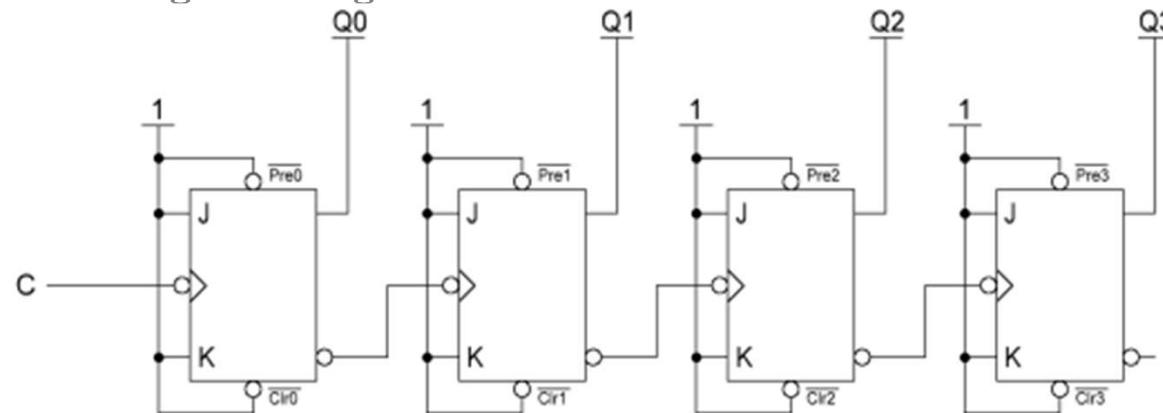
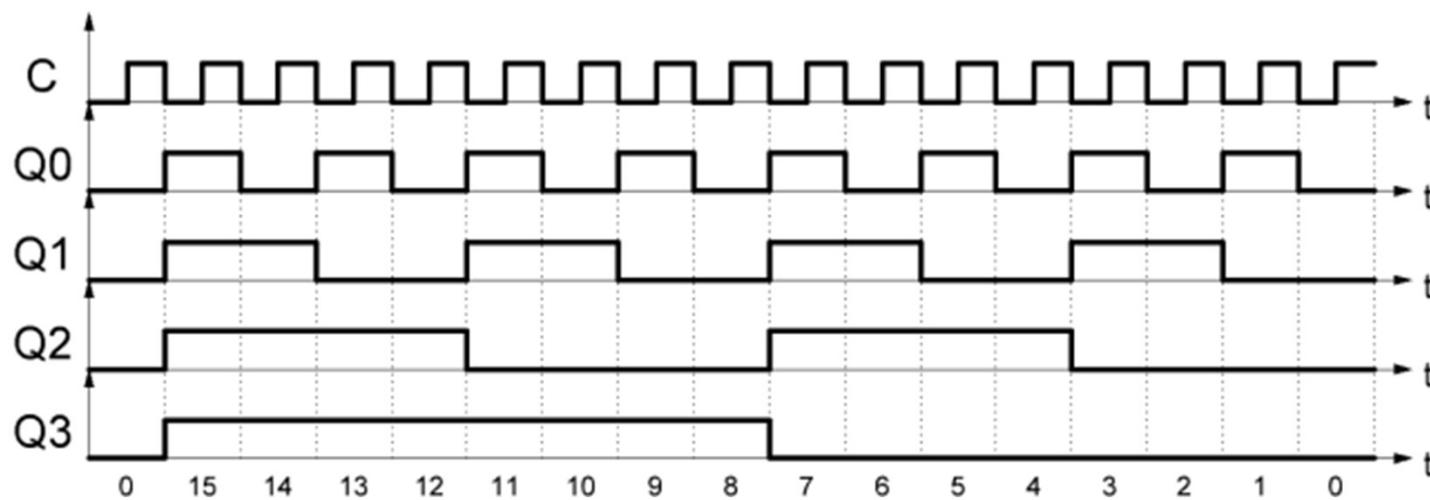


Figure 3



Logique séquentielle – Compteurs, registres et mémoires

Exercice compteur 2: correction À partir du montage de la figure 3, remplissez le chronogramme ci-dessous. Que réalise le montage de la figure 3 ?

Les bascules JK sont synchronisées sur front descendant et câblées en basculement permanent (J et K sont toujours à 1) :

- La sortie Q0 bascule sur chaque front descendant de C ;
- La sortie Q1 bascule sur chaque front descendant de Q0 (donc chaque front montant de Q0) ;
- La sortie Q2 bascule sur chaque front descendant de Q1 (donc chaque front montant de Q1) ;
- La sortie Q3 bascule sur chaque front descendant de Q2 (donc chaque front montant de Q2).

Que réalise le montage de la figure 3 ?

À chaque front d'horloge, la valeur présente sur les sorties est décrémentée de un. Ce montage est un décompteur asynchrone modulo 16. Il décompte de 15 à 0.

Logique séquentielle – Compteurs, registres et mémoires

Les Registres (1/8)

Les registres sont les éléments de base des mémoires réalisées avec des semi-conducteurs. On peut représenter **un registre comme un ensemble de mémoires élémentaires susceptibles de stocker chacune un bit.**

L'entrée des informations dans un registre peut se faire **soit en série (les unes après les autres), soit en parallèle (toutes au même moment).**

De la même façon la présentation des informations sur les sorties peut se faire soit en série soit parallèle.

On aboutit ainsi à **quatre types de fonctionnements différents pour les registres:**

- ❖ parallèle-série;
- ❖ parallèle-parallèle;
- ❖ série-parallèle;
- ❖ série-série.

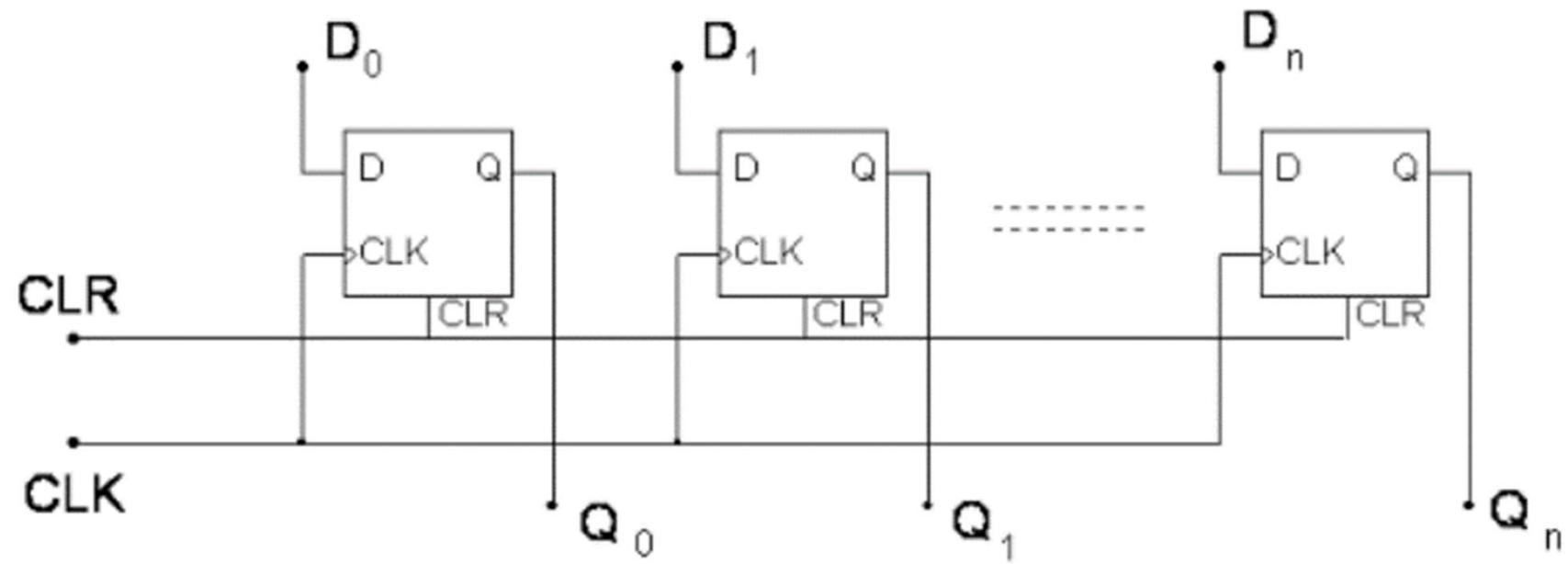
Logique séquentielle – Compteurs, registres et mémoires

Registre à Décalage – Tampon (2/8)

Les registres tampon sont des registres de type parallèle-parallèle constitués de n bascules de type D commandées par une même horloge.

Sur le signal d'horloge (impulsion sur CLK) les entrées D_i sont recopiées sur les sorties Q_i .

Une entrée asynchrone CLR permet, de façon prioritaire, d'effacer le contenu du registre et d'écrire $Q_i = 0$. Entre deux impulsions les sorties sont parfaitement isolées des entrées.



Logique séquentielle – Compteurs, registres et mémoires

Registre à Décalage (3/8)

Les registres à décalage sont des registres de type série-série ou série parallèle, dans lesquels les informations sont décalées d'une bascule vers la suivante au rythme des impulsions d'une horloge.

Ils sont généralement réalisés avec des bascules RS de type maître-esclave.

Le schéma de principe d'un registre à décalage (vers la droite) avec une entrée série est présenté sur la figure à la slide suivante.

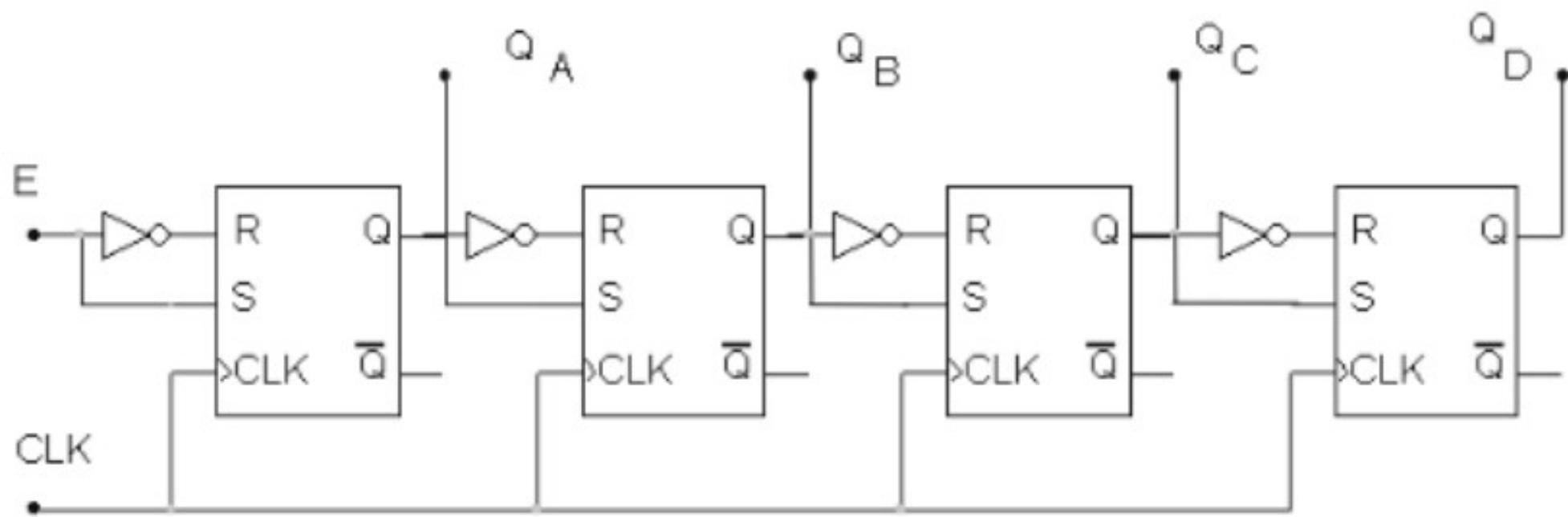
Au fur et à mesure des impulsions d'horloge les données présentes sur l'entrée série E sont transférées sur les différentes bascules.

La présence de l'inverseur entre R et S assure toujours $S = R$. Les bascules ne sont donc jamais en mode mémoire mais toujours en mode SET ou RESET.

Dans ces conditions la sortie recopie, au moment de l'impulsion d'horloge, la valeur présente sur l'entrée

Logique séquentielle – Compteurs, registres et mémoires

Registre à Décalage - Suite (4/8)



Logique séquentielle – Compteurs, registres et mémoires

Registre à Décalage - Suite (5/8)

Schéma de principe d'un registre à décalage avec une entrée série (E). Le décalage se fait de Q_A vers Q_B

En guise d'exemple, on suppose que la situation de départ est la suivante: E = 0 et seule la sortie Q_A de la première bascule est au niveau 1 les autres étant au niveau 0 ($Q_A Q_B Q_C Q_D = 1000$)

- **1ère impulsion** : chaque bascule recopie sur sa sortie Q la valeur présente sur son entrée S. On obtient donc après l'impulsion ($Q_A Q_B Q_C Q_D = 0100$).
- **2ème impulsion** : seule l'entrée S de la troisième bascule est au niveau haut. Après l'impulsion on a donc ($Q_A Q_B Q_C Q_D = 0010$).

On raisonne de la même façon pour la troisième impulsion et on aboutit à $Q_A Q_B Q_C Q_D = 0001$.

Ainsi au fur et à mesure des impulsions sur l'entrée d'horloge (CLK) le 1 présent initialement sur Q_A a été progressivement décalé vers la droite.

Logique séquentielle – Compteurs, registres et mémoires

Registre à Décalage - Universel (6/8)

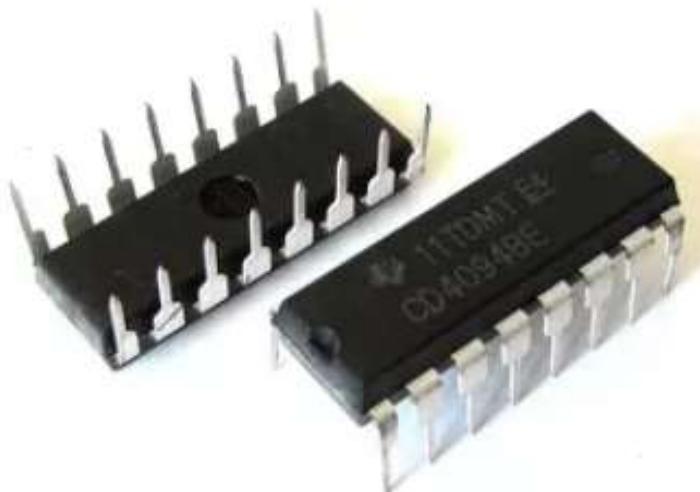
Dans la pratique, il est inutile d'effectuer l'inventaire des registres à décalage : **un choix très vaste est offert par les constructeurs.**

A titre d'exemple citons les registres universels de type 194 dont le schéma est présenté à la slide suivante.

Ce sont des registres à quatre bits à chargement parallèle ou série.

Outre une remise à zéro asynchrone (**CLEAR**), ils offrent la possibilité de déplacer l'information vers la droite (**de Q_A vers Q_D ou vers la gauche ou de Q_D vers Q_A**).

Le mode de fonctionnement est choisi avec les entrées synchrones **S_0 et S_1** .



Logique séquentielle – Compteurs, registres et mémoires

Registre à Décalage - Application des registres à décalage (7/8)

74AC11194
4-BIT BIDIRECTIONAL UNIVERSAL SHIFT REGISTER

SCAS093 – NOVEMBER 1989 – REVISED APRIL 1993

CLEAR	MODE		CLOCK	SERIAL		PARALLEL		Function Table						
	S1	S0			LEFT	RIGHT	A	B	C	D	QA	QB	QC	QD
L	X	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	X	QA0	QB0	QC0	QD0
H	H	H	T	X	X	a	b	c	d	X	a	b	c	d
H	L	H	↑	X	L	H	X	X	X	X	H	QA1	QB1	QC1
H	L	H	↑	X	L	X	X	X	X	X	L	QA1	QB1	QC1
H	H	L	↑	H	X	X	X	X	X	X	QDn	QAn	QBn	QCn
H	H	L	↑	L	X	X	X	X	X	X	QDn	QAn	QBn	QCn
H	L	L	X	X	X	X	X	X	X	X	QAG	QB0	QC0	QD0

H = high level (steady state)

L = low level (steady state)

X = irrelevant (any input, including transitions)

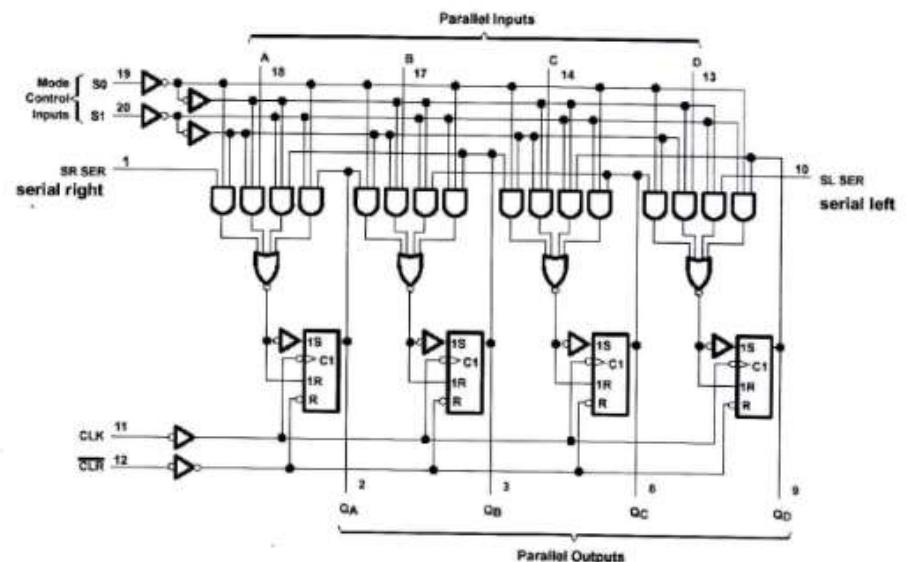
↑ = transition from low to high level

a,b,c,d = the level of steady-state input at inputs A, B, C, or D, respectively.

QA0, QB0, QC0, QD0 = the level of QA, QB, QC, or QD, respectively, before the indicated steady-state input conditions were established.

QA1, QB1, QC1, QDn = the level of QA, QB, QC, or QD respectively, before the most-recent ↑ transition of the clock.

logic diagram (positive logic)



Logique séquentielle – Compteurs, registres et mémoires

Registre à Décalage - Application des registres à décalage (8/8)

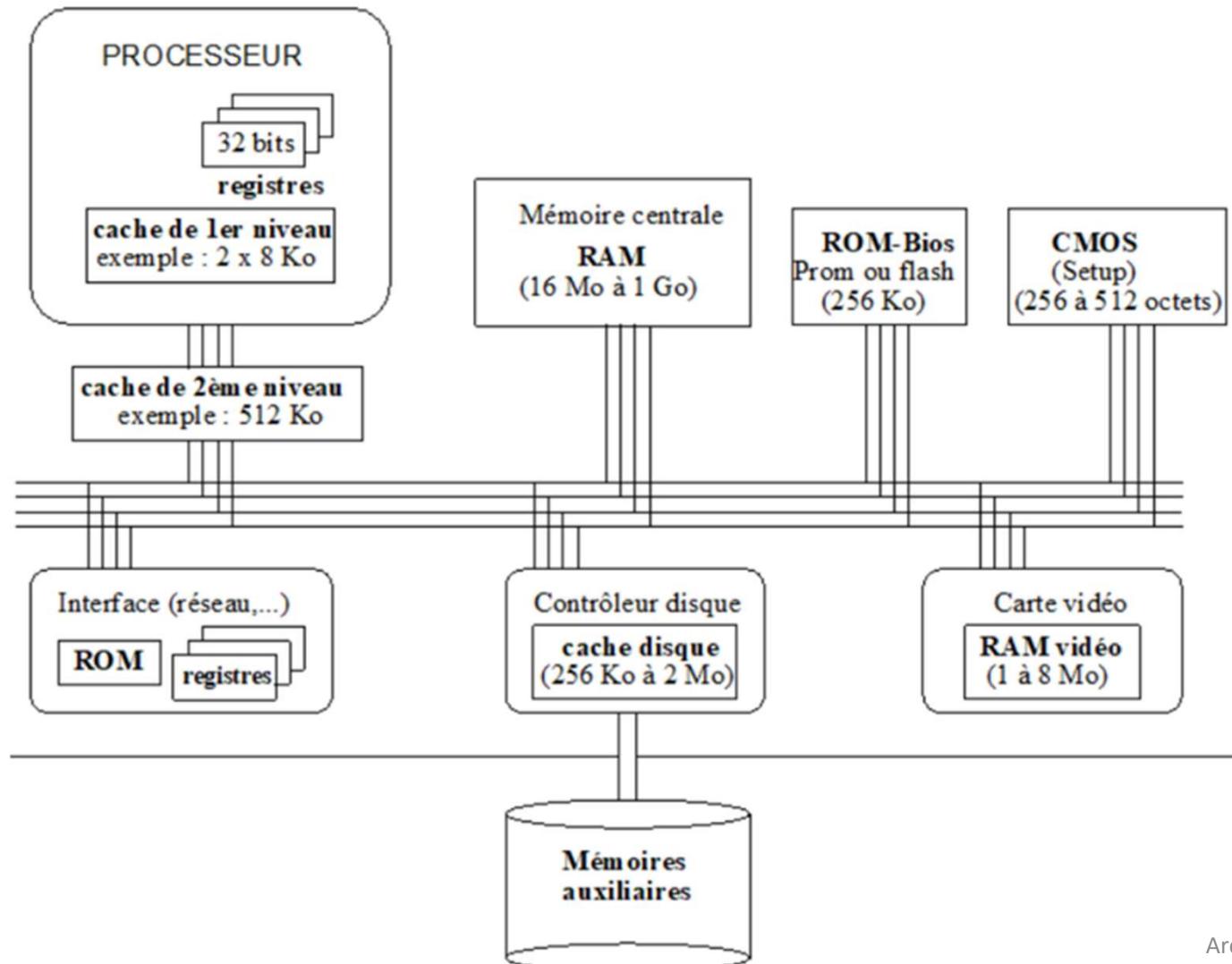
Les registres à décalage sont utilisés dans de nombreuses applications comme par exemple:

- ❖ la mise en mémoire, avant affichage, des sorties d'un compteur (registre à une entrée et une sortie parallèle);
- ❖ la génération de séquences 011001 pseudo aléatoire (registre à une entrée série et une parallèle);
- ❖ la multiplication d'un nombre codé $a_n \dots a_0$ par 2^p se traduit de tous les bits de p cases vers la gauche.



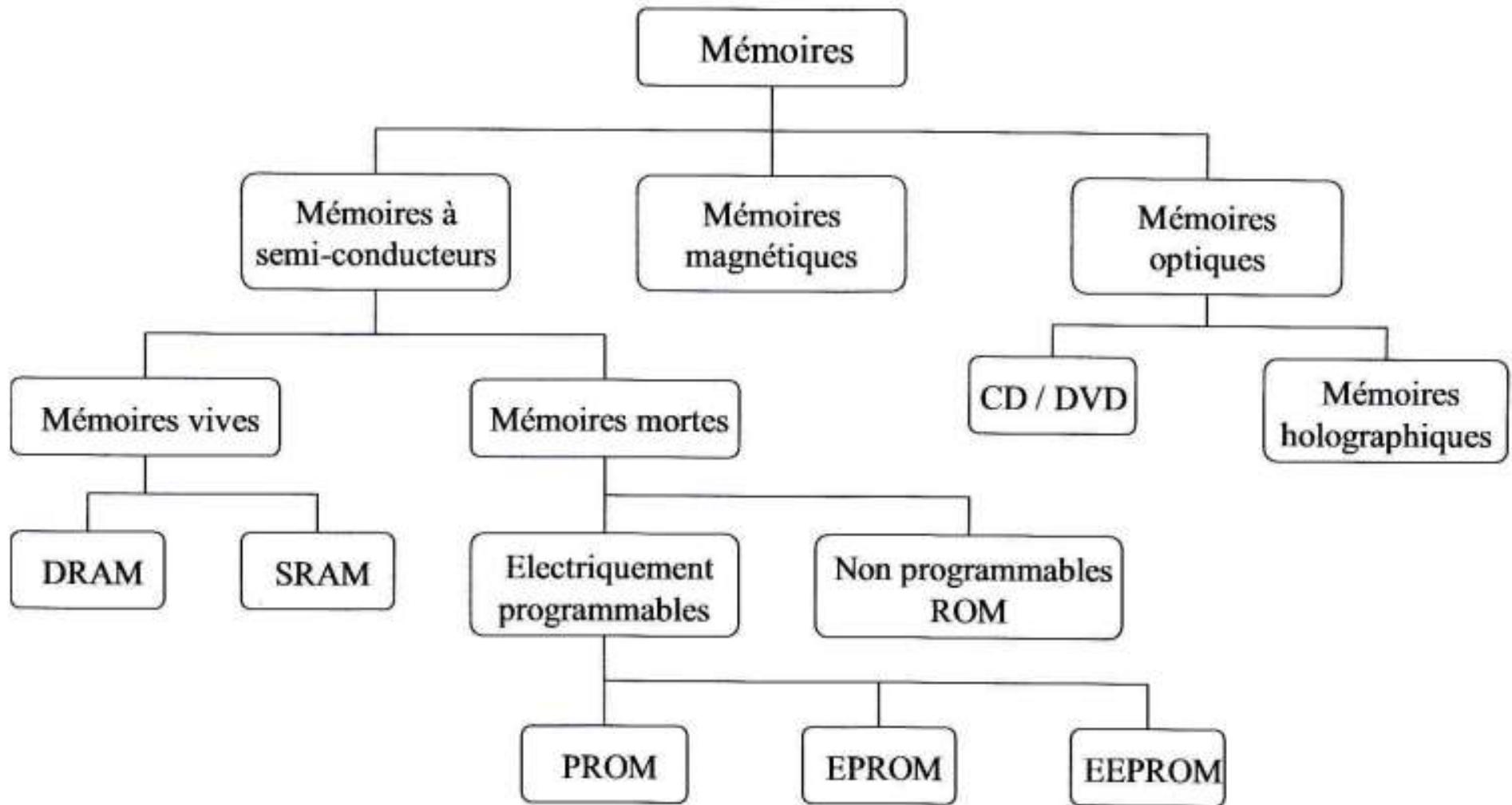
Logique séquentielle – Compteurs, registres et mémoires

Etats des lieux des mémoires d'un ordinateur



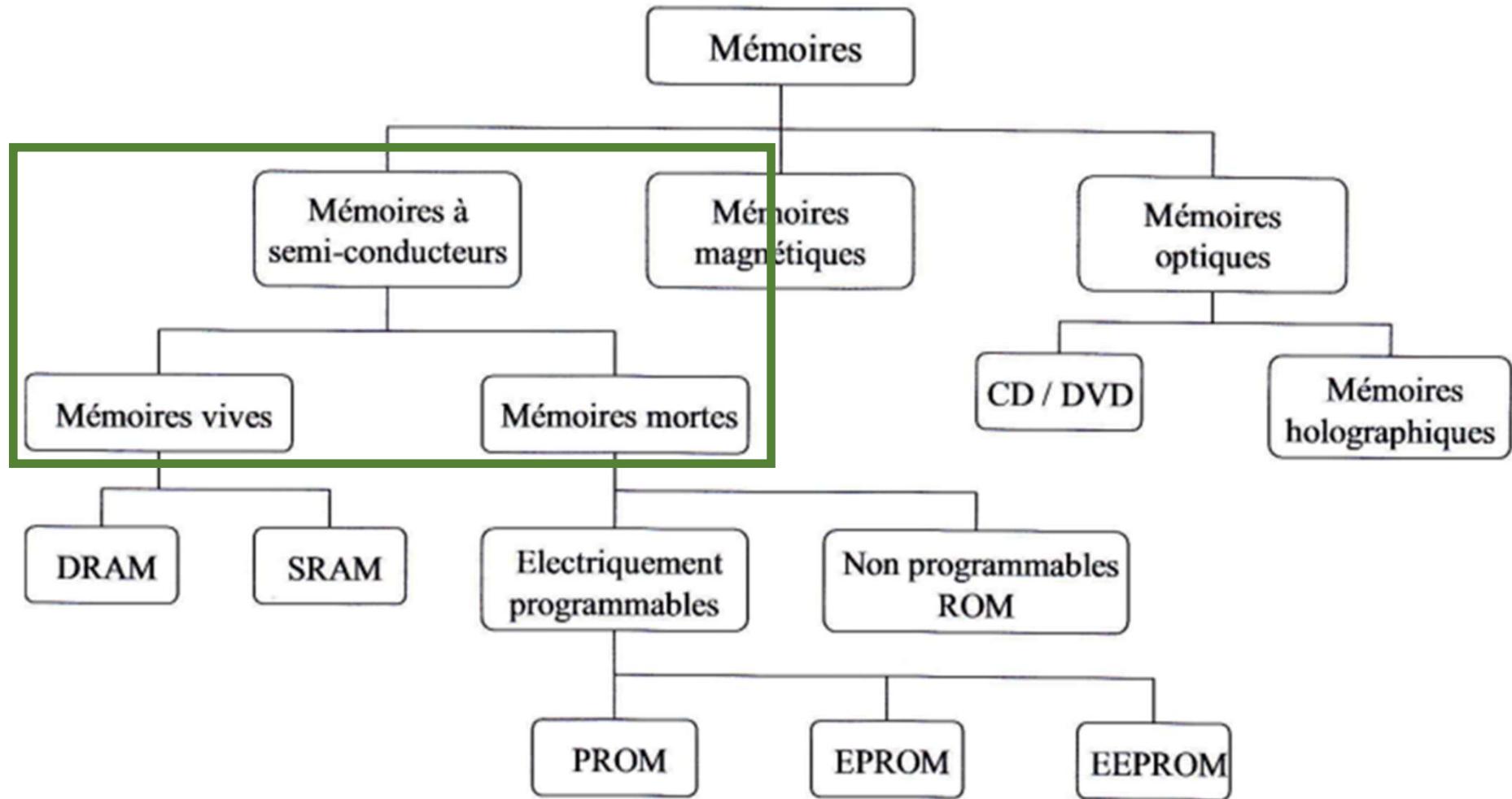
Logique séquentielle – Compteurs, registres et mémoires

Les différentes technologies des mémoires d'un ordinateur



Logique séquentielle – Compteurs, registres et mémoires

La famille de la technologie des semiconducteurs



Logique séquentielle – Compteurs, registres et mémoires

Les mémoires à semi-conducteur - Généralités

Contrairement aux systèmes analogiques, les systèmes numériques permettent de mémoriser des informations avant de les traiter.

On distingue trois grands types de mémoires :

- ❖ les mémoires magnétiques comme les disques durs ou les disquettes ☺ Aie !
- ❖ les mémoires à semi-conducteur telles que les RAM ou les ROM;
- ❖ les mémoires optiques comme les CD et les DVD.

Les mémoires vives (1/3)

Ces mémoires vives ou Random Access Memory (RAM) peuvent être lues ou écrites quasi-instantanément en fonction des besoins de l'utilisateur (temps d'accès de l'ordre de 10 à 50 ns).

La RAM est apparue dans le but d'une évolution et d'une amélioration de la technique des bandes magnétiques. En effet, grâce à la RAM, il n'est plus nécessaire de faire défiler toutes les données de la mémoire avant de trouver l'information cherchée.

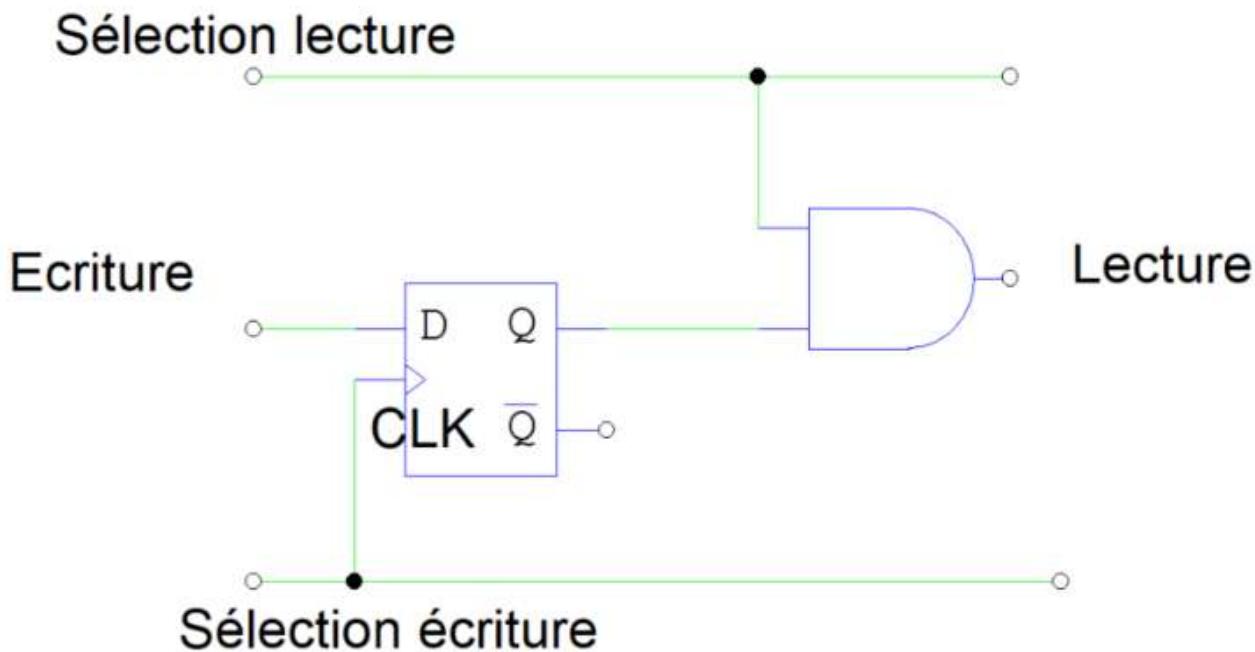
Leur inconvénient est la perte de toute la donnée en cas de coupure d'électricité. En pratique, il existe deux types de mémoires vives SRAM et DRAM présentées par la suite.

Logique séquentielle – Compteurs, registres et mémoires

Les mémoires vives – RAM statiques (SRAM) – (2/3)

L'unité de base d'une SRAM est typiquement une bascule D. En phase d'écriture, la donnée sur D est recopiée sur la sortie Q au moment d'un front montant d'horloge.

La lecture consiste à forcer la ligne de la sélection au niveau haut pour présenter la sortie de la bascule D sur le bus de données.



Logique séquentielle – Compteurs, registres et mémoires

Les mémoires vives - RAM dynamiques (DRAM) – (3/3)

La technologie de la DRAM utilise les phénomènes physiques comme la charge au niveau = 1 et la décharge au niveau = 0 d'un condensateur.

Cependant, le diélectrique (isolant entre les armatures du condensateur) peut vieillir et se dégrader dans le temps.

Ces mémoires doivent périodiquement être rafraîchies (toutes les 20 ms période du réseau EDF).

Malgré cet inconvénient les DRAM sont très fréquemment utilisés car leur simplicité permet de les intégrer en plus grand nombre sur une puce de silicium plus que leurs concurrentes statiques (SRAM).



Logique séquentielle – Compteurs, registres et mémoires

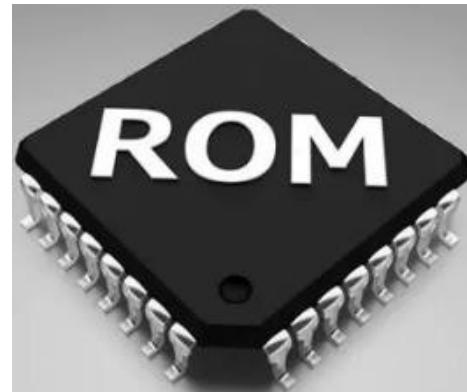
Les mémoires mortes (1/4)

Les mémoires mortes ou Read Only Memory (ROM) ne sont accessibles qu'en lecture. Elles doivent être programmées en dehors du système où elles sont exploitées.

Selon le mode d'écriture et son caractère définitif ou pas, on distingue différents types de mémoires mortes.

Les mémoires mortes – ROM

Les ROM sont écrites une fois pour toute en usine. Cette programmation est faite directement sur le wafer (galette de silicium) à l'aide de masques de programmation. Bien évidemment, la fabrication de ROM ne se conçoit que pour des séries importantes (> 10000 unités).



Logique séquentielle – Compteurs, registres et mémoires

Les mémoires mortes – PROM (2/4)

Certaines applications n'ont pas besoin de conserver des informations de façon permanente même lorsque l'alimentation électrique est interrompue (**exemple: dans les démarages des OS -> BIOS**).

Ces mémoires sont non volatiles. Contrairement aux RAM, elles ne peuvent être que lue.

L'inscription en mémoire des données reste possible mais est appelée programmation. Suivant le type de ROM, la méthode de programmation changera. Il existe donc plusieurs types de ROM :

- ❖ ROM = Programmation en usine;
- ❖ PROM = Programmable ROM;
- ❖ EPROM = Erasable ROM (UV);
- ❖ EEPROM = Electrically Erasable ROM (Électrique);



Logique séquentielle – Compteurs, registres et mémoires

Les mémoires mortes – PROM – Opération de Programmation (3/4)

Après la programmation de la mémoire, c'est-à-dire la destruction de certains fusibles, on peut obtenir le tableau suivant.

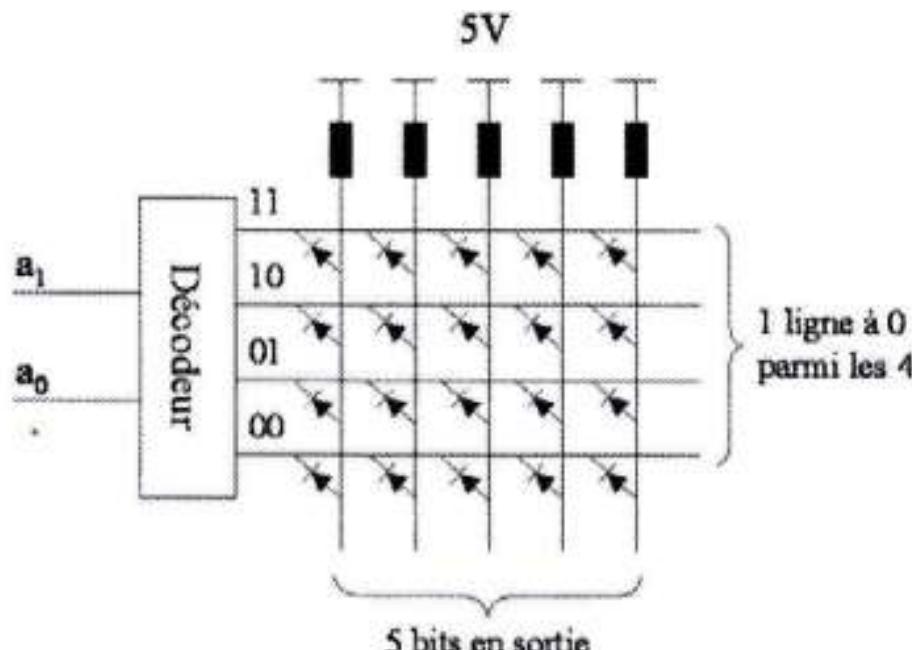
Adresse		Sorties				
a_1	a_0	S_4	S_3	S_2	S_1	S_0
0	0	0	0	1	0	1
0	1	1	0	1	0	1
1	0	0	1	1	0	1
1	1	1	1	1	1	0

Logique séquentielle – Compteurs, registres et mémoires

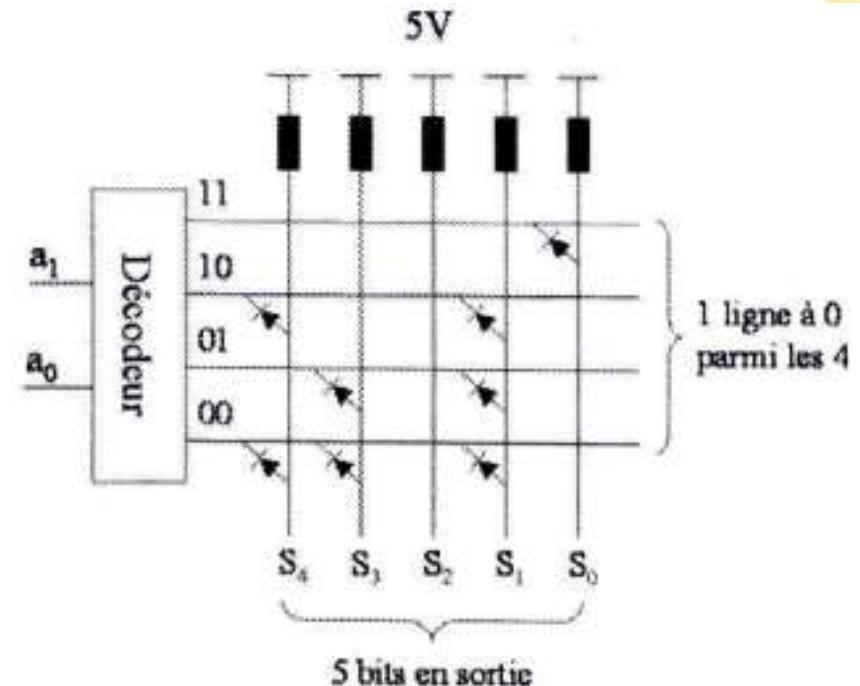
Les mémoires mortes – PROM – Opération de Programmation (4/5)

Les PROM sont plutôt utilisées pour des systèmes fabriqués en séries restreintes ou souvent renouvelées. Le principe de la programmation d'une mémoire PROM est :

- a) représente la situation de la mémoire avant la programmation;
- b) la mémoire après la programmation.



a)



b)

Logique séquentielle – Compteurs, registres et mémoires

Les mémoires mortes – PROM Effaçables (5/5)

Il existe des PROM que l'on peut effacer avec des rayons UV. Ce sont les **EPROM** (Erasable PROM).

On trouve des mémoires mortes effaçables électriquement ce sont les **EEPROM** (Electrical Erasable PROM).

Il est possible de réécrire dans ces mémoires effaçables.

Les **EPROM** et les **EEPROM** sont surtout utilisées dans les phases de développement ou dans les systèmes fabriqués en très petites séries.

Logique séquentielle – Compteurs, registres et mémoires

La mémoire MRAM

La mémoire MRAM (Magnetic Random Access Memory) est une mémoire non volatile de type magnétique.

En développement depuis les années 1990, cette technologie n'a jusqu'à présent jamais été commercialisée à grande échelle à cause de la concurrence des mémoires flash et DRAM.



Logique séquentielle – Compteurs, registres et mémoires

La mémoire FLASH

La mémoire flash est une mémoire à semi-conducteurs, non volatile et réinscriptible.

C'est une mémoire possédant les caractéristiques d'une mémoire vive mais dont les données ne se volatilisent pas lors d'une mise hors tension.

En raison de sa vitesse élevée, de sa durabilité et de sa faible consommation, la mémoire flash est idéale pour de nombreuses applications: les appareils photos numériques, les téléphones cellulaires, les imprimantes, les assistants personnels (PDA), les ordinateurs portables, ou les dispositifs de lecture ou d'enregistrement sonore tels que les baladeurs mp3, etc...

Ce type de mémoire ne possède pas d'éléments mécaniques, ce qui leur confère une grande résistance aux chocs.

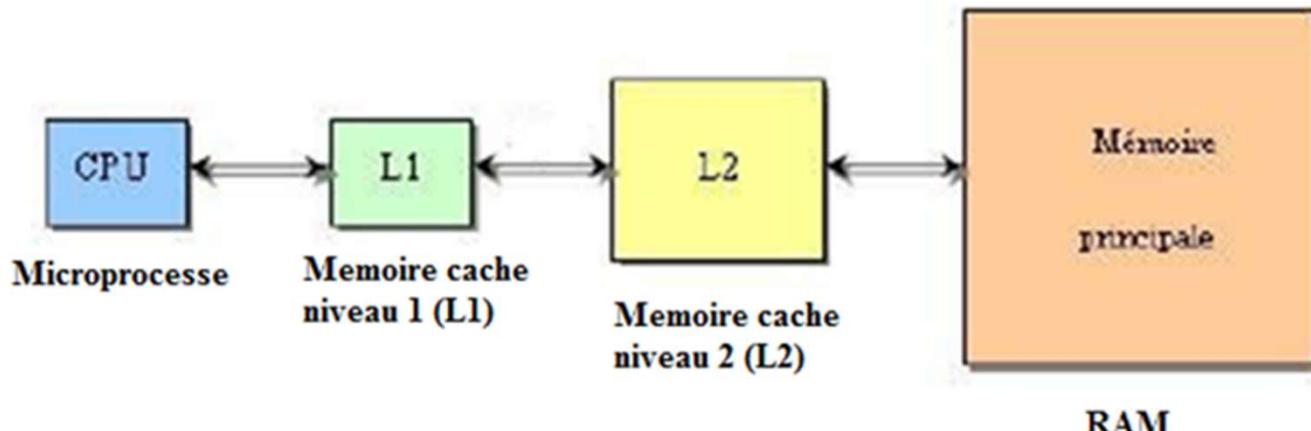


Logique séquentielle – Compteurs, registres et mémoires

La mémoire CACHE

Elle a pour fonction d'accélérer le transfert d'informations entre le processeur et la mémoire de l'ordinateur RAM.

Un processeur peut disposer d'une ou plusieurs niveaux de mémoire cache appelé L1, L2 et L3. Une mémoire cache de taille importante est utile pour les traitements de gros volumes de données.



NB La transmission d'informations entre processeur et mémoire vive est souvent bien plus lente que la vitesse du microprocesseur.

Pour pallier, les processeurs intègrent une petite zone de mémoire ultra-rapide où sont conservées les instructions et données qui reviennent le plus souvent. **Mais cette mémoire interne est de petite taille, quelques dizaines de Ko (voire plus).**

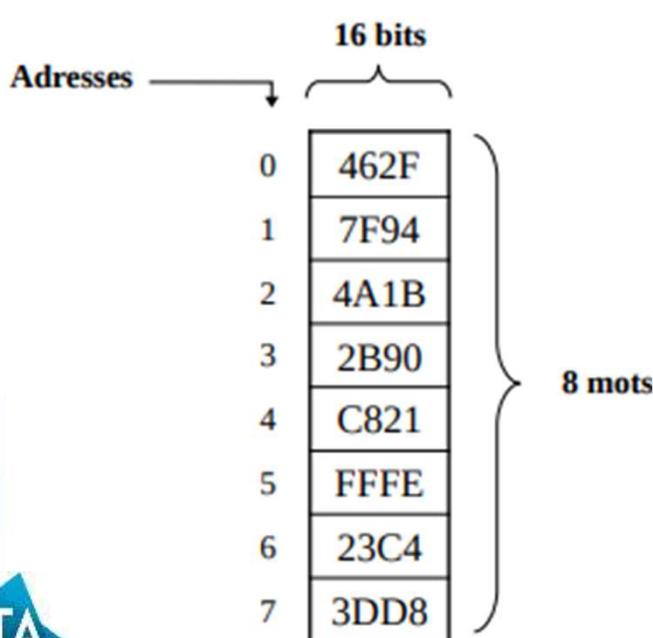
Logique séquentielle – Compteurs, registres et mémoires

Organisation d'une mémoire définition (1/5)

Une mémoire est un composant électronique qui permet de stocker des informations sous la forme de mots binaires. Un mot binaire est une quantité déterminée de bits (par exemple, un octet est un mot de 8 bits).

Nous pouvons représenter une mémoire comme un ensemble de cases contenant chacune un mot binaire. Chaque case est identifiée par un numéro d'adresse unique.

Exemple d'une mémoire contenant 8 mots de 16 bits :



Par exemple :

La donnée contenue dans la case mémoire d'adresse **0** est **462F16**.

La donnée contenue dans la case mémoire d'adresse **4** est **C82116**.

L'organisation interne d'une mémoire se caractérise par les deux grandeurs suivantes :

- le nombre de mots (ou d'adresses) qu'elle contient;
- le nombre de bits par mot.

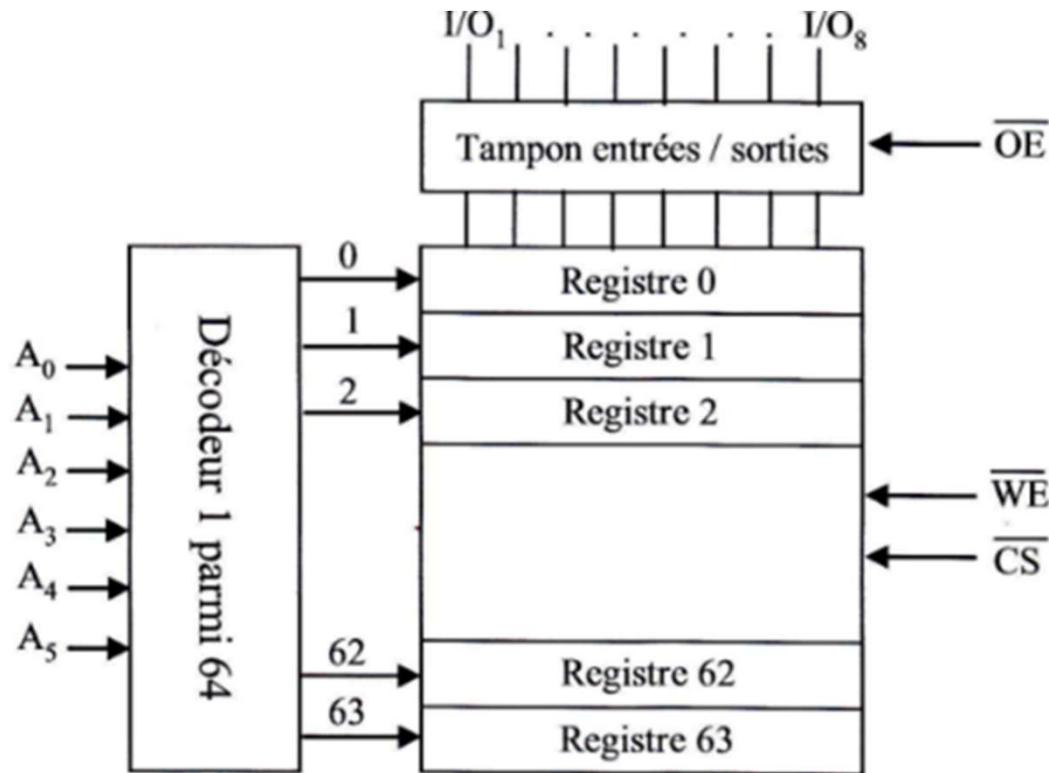
Logique séquentielle – Compteurs, registres et mémoires

Organisation d'une mémoire (2/5)

Une mémoire est constituée d'un ensemble de registres dont l'un est en relation avec l'extérieur soit pour sa lecture (ROM), soit pour sa lecture ou son écriture (RAM).

La figure ci-après représente l'organisation d'une RAM composée de 64 registres de 8 bits.

La taille des registres définit le format de la mémoire et le nombre de registre détermine la capacité de la mémoire.



Logique séquentielle – Compteurs, registres et mémoires

Organisation d'une mémoire (3/5)

Les données

Pour limiter au maximum l'encombrement du composant, on utilise généralement les mêmes pattes pour les entrées (Input) et les sorties (Output) de la RAM.

On gagne ainsi un facteur deux sur le nombre de broches dédiées aux données. Cependant, il faut bien sûr ajouter une entrée de contrôle supplémentaire pour spécifier si la RAM fonctionne en mode lecture ou en mode écriture (\overline{OE} pour Output Enable: $\overline{OE}=0$ la RAM est en mode lecture et $\overline{OE}=1$ la Ram est en mode écriture).

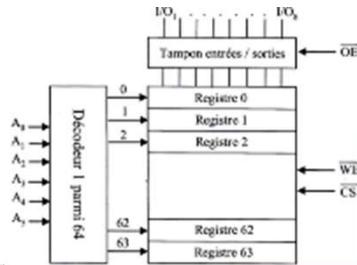
Pour la RAM considérée ici le gain est $16/2 - 1 = 7$ broches.

Les adresses

Le choix du registre en contact avec l'extérieur est réalisé en décodant l'adresse écrite sur les 6 bits:

$A_5A_4A_3A_2A_1A_0$ (A_5 = MSB et A_0 =LSB). L'adresse $A_5A_4A_3A_2A_1A_0 = 001101$ désigne le registre n°13. Des astuces de câblage permettent de réduire le nombre de connexions pour les mémoires de grande capacité.

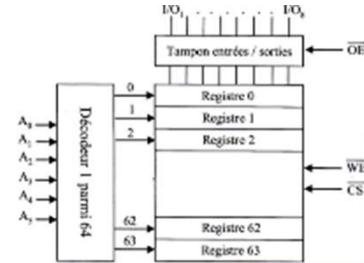
Par exemple, une mémoire de 1 Mo nécessite 20 bits d'adresse donc 20 broches. On peut réduire ce nombre par deux en transmettant l'adresse en deux paquets consécutifs de 10 bits.



Logique séquentielle – Compteurs, registres et mémoires

Organisation d'une mémoire (4/5)

Gestion des cycles d'écriture et de lecture



En plus de l'entrée \overline{OE} décrit précédemment, les RAM possèdent souvent au moins deux autres entrées contrôle.

D'une part l'entrée \overline{CS} (Chip Select) qui permet de mettre la RAM en mode actif ou en mode veille (en mode veille la consommation électrique de la RAM est nettement réduite mais aucune action de lecture ou d'écriture n'est possible).

D'autre part l'entrée \overline{WE} (Write Enable) qui n'autorise l'écriture que si elle est au niveau bas.

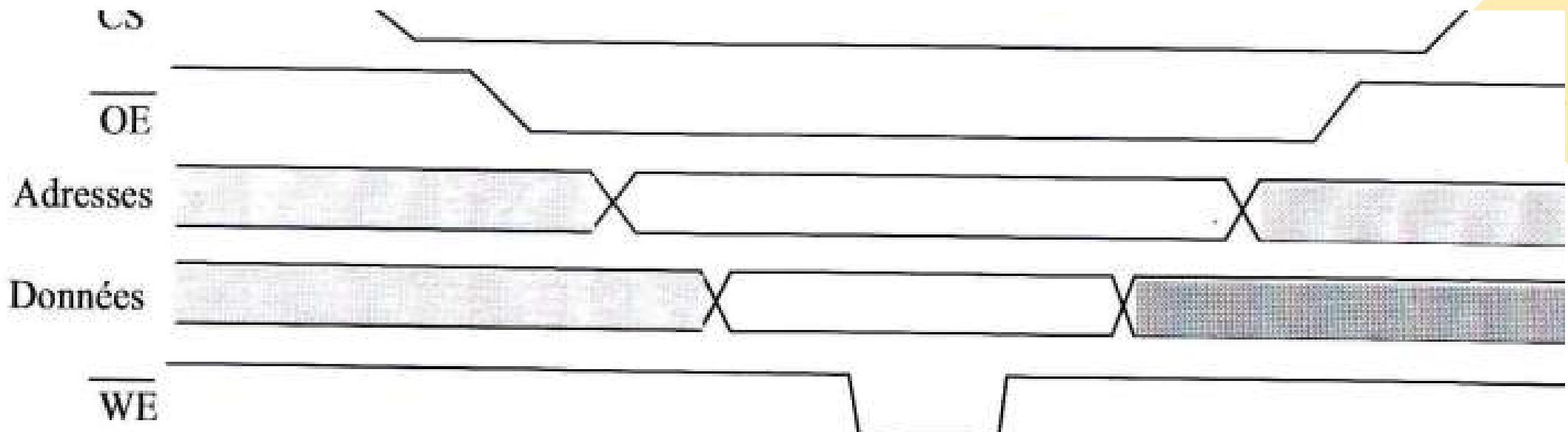
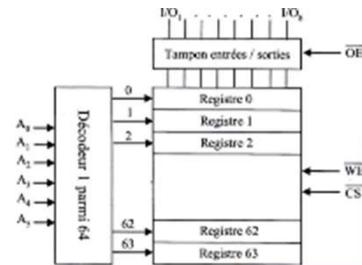
Lors d'un cycle d'écriture, il faut tout d'abord sélectionner le composant $\overline{CS} = 0$, le mettre en mode lecture $\overline{OE} = 0$, puis il faut fixer les adresses et les données.

Ensuite et seulement, il faut donner l'ordre d'écriture $\overline{WE} = 0$. Cette chronologie est décrite à la slide suivante.

Logique séquentielle – Compteurs, registres et mémoires

Organisation d'une mémoire (5/5)

Cycles d'écriture et de lecture



Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (1/11)

En plus des technologies à base de semiconducteurs, il existe d'autres types de mémoires basées sur le stockage optique de l'information.

Ces autres technologies ne seront pas abordées dans le détail dans la suite de ce cours.

Les mémoires optiques CD et DVD

Le Compact Disk (CD) et le Digital Versatil Disc (DVD) sont utilisés pour stocker de la musique, des données informatiques ou de la vidéo. Du fait de leur facilité de fabrication et d'utilisation, ils sont devenus des systèmes standards de diffusion de l'information.

Les principes, techniques, de stockage de l'information sur ces deux types de supports sont semblables.

Les CD – Stockage de l'information

Morceau de polycarbonate (plastique) de 1,2 mm d'épaisseur sur lequel l'information est écrite le long d'une spirale qui se déroule de l'intérieur vers l'extérieur du disque.

A la fabrication, l'information est stockée sous la forme de microcuvettes gravées sur une face du disque. Cette dernière est recouverte d'une couche métallique (aluminium, argent ou or) puis d'un revêtement acrylique pour protéger le dépôt métallique.

Logique séquentielle – Compteurs, registres et mémoires

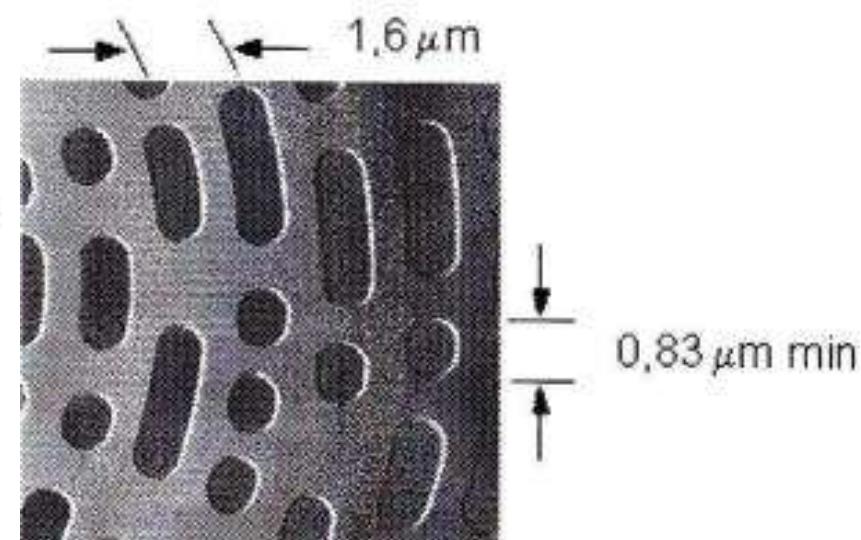
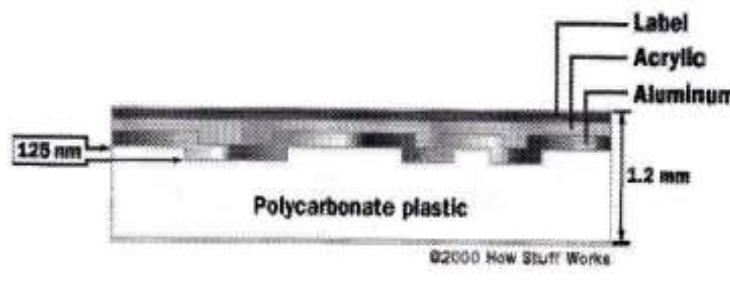
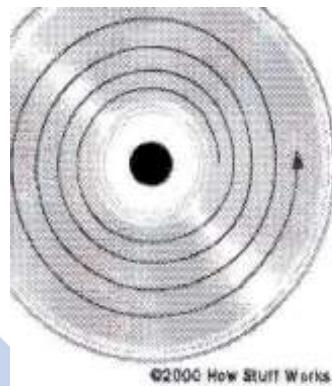
Les autres technologies de mémoire (2/11)

Les CD – Stockage de l'information (suite)

Ce qui est réellement impressionnant sur un CD, ce sont les dimensions de ces microcuvettes. Leur profondeur est de 125 nm, leur largeur de 500 nm et leur longueur de 830 nm au minimum.

$1 \text{ nm} = 10^{-9} \text{ m} = 1 \text{ milliardième de mètre.}$

Les sillons de la spirale étant séparés de 1,6 µm, on peut calculer que l'information est stockée sur 5 Km de long !



Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (3/11)

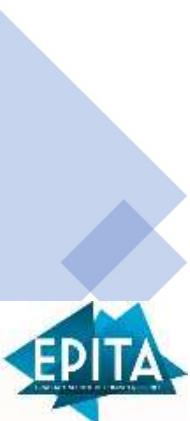
Les CD – Lecture (suite)

Lors de la lecture d'un CD, le faisceau laser passe à travers le polycarbonate et se réfléchit sur la couche métallique.

Selon que le signal laser est réfléchi dans une cuvette ou sur une partie plate, une photodiode détecte, ou pas, le rayonnement.

C'est une modulation de réflexion qui permet de reconstituer le signal inscrit sur le disque. En vérité, c'est le délai entre deux transitions consécutives qui constitue l'information.

Ce type de fonctionnement impose des contraintes très fortes sur la fiabilité et la précision des parties mécaniques d'un lecteur de CD.



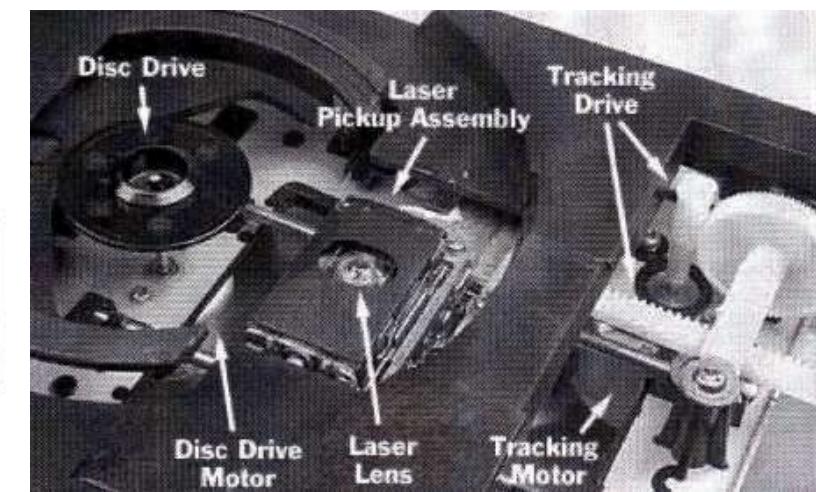
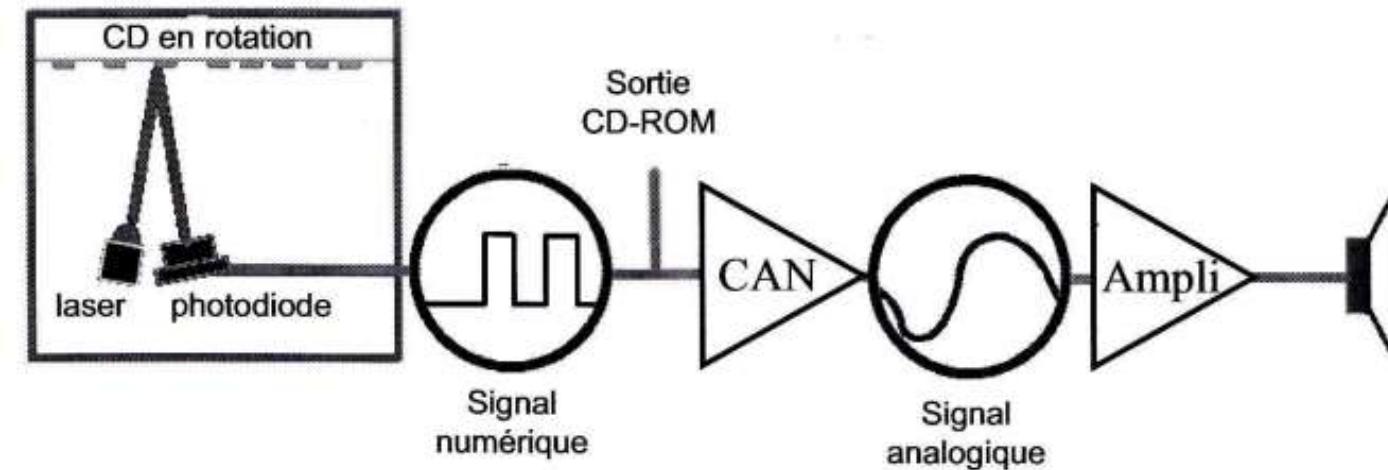
Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (4/11)

Les CD – Lecture (suite)

Un lecteur de CD est composé de 3 parties importantes:

- ❖ un moteur qui fait tourner le disque à une vitesse très précise et contrôler entre 200 et 500 tours/min en fonction de la partie du disque lue;
- ❖ un système mobile comprenant une diode laser, une lentille et une photodiode;
- ❖ un second moteur qui déplace le système laser le long du disque afin de suivre précisément la spirale. Ce système de déplacement doit avoir une précision de l'ordre du micron.



Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (5/11)

Les CD – Enregistrable CD-R

Le disque CD-R est constitué d'un support en plastique revêtu d'une couche de colorant et d'une couche métallique. Il n'y a pas plus de microcuvettes à la surface du disque puisque celui-ci ne contient pas encore de données.

Cependant, la spirale sur laquelle seront enregistrées les informations est matérialisée par un sillon creusé dans la surface du plastique.

L'enregistrement des données se fait grâce à un laser focalisé modulé dans le temps entre deux niveaux de puissance : la puissance de lecture ($> \text{mW}$) et la puissance d'écriture (comprise entre 6 et 12 mW selon les disques).

Le colorant absorbe la lumière et induit un échauffement local proportionnel à la puissance du laser. À la puissance d'écriture, le colorant est dégradé et le substrat en plastique subit une déformation locale. Ces deux effets se traduisent par une baisse de la réflexion.

La modulation de puissance est imposée par le signal que nous voulons écrire.

Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (6/11)

Les CD – Enregistrable CD-RW

Les CD-RW utilisent des matériaux à changement de phase.

Ce sont des matériaux dans lesquels peuvent coexister, deux phases différentes: **une phase cristalline et la phase amorphe.**

Ces deux états ont des priorités optiques très différentes.

Cette différence est mise à profit pour enregistrer l'information sous forme d'une variation locale de la réflexion. L'état initial est l'état cristallin.

Une donnée est inscrite sous la forme d'une marque amorphe ayant une réflexion de basse. Pour effacer les données, il suffit de recristalliser les marques amorphes.

Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (7/11)

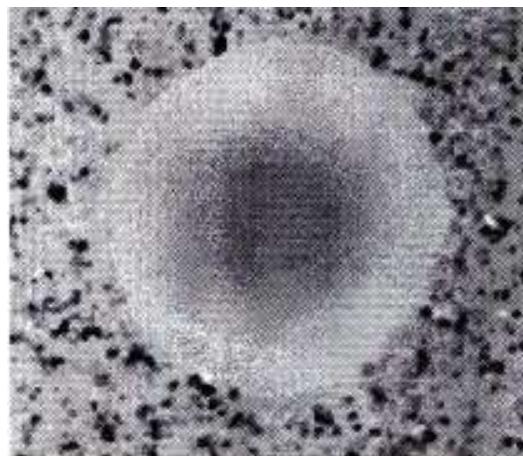
Les CD – Enregistrable CD-RW - Ecriture des données

Pour écrire une marque amorphe, il faut fondre localement le matériau et le refroidir suffisamment vite pour qu'il se solidifie en restant dans un état proche de l'état liquide : l'état amorphe.

Typiquement, la fusion est obtenue autour de 600 °C avec une impulsion laser de 10 mW et d'une durée de 10 ns.

Si le refroidissement n'est pas suffisamment rapide, les atomes peuvent se déplacer et former un édifice cristallin.

Cette recristallisation se traduit par une baisse voir une disparition du contraste. Ci-dessous, une image au microscope électronique d'une marque amorphe.



Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (8/11)

Les CD – Enregistrable CD-RW - Effacement des données

Pour effacer les données on chauffe la marque amorphe à une température inférieure à la température de fusion.

Ceci est toujours obtenu avec le faisceau laser localisé, mais à une puissance intermédiaire entre la puissance d'écriture et la puissance de lecture.

Deux catégories de matériaux à changement de phase sont utilisées:

- ❖ les alliages ternaires de Germanium, d'antimoine et de tellure **GeSbTe**;
- ❖ les alliages d'antimoine et de tellure dopés à l'argent et à l'indium **AgInSbTe**.

Logique séquentielle – Compteurs, registres et mémoires

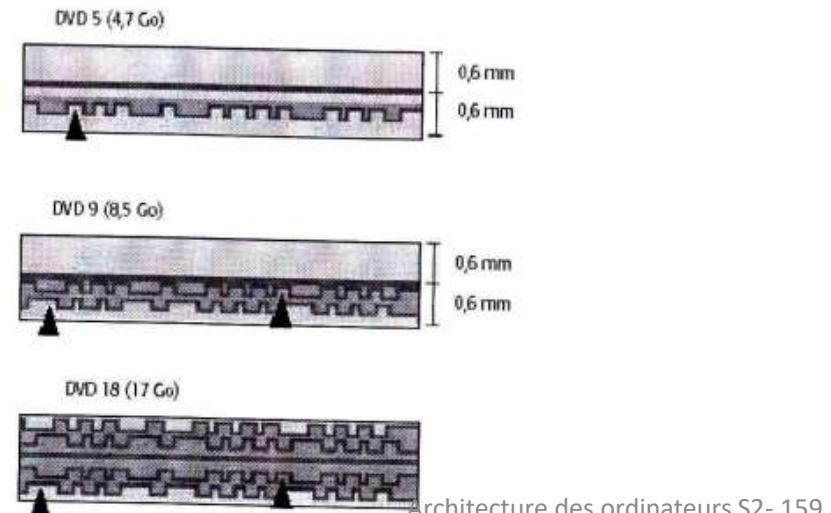
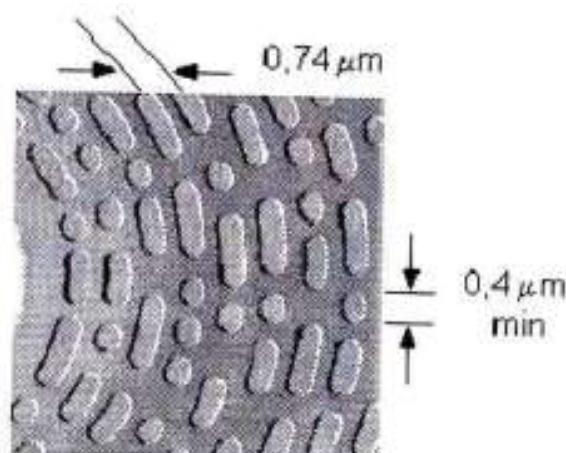
Les autres technologies de mémoire (9/11)

Les DVD

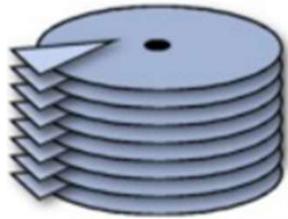
D'aspect équivalent au CD, un DVD standard permet de stocker 7 fois plus d'informations (4,7 Go au lieu de 700 Mo).

Le principe du DVD quel qu'en soit le type (préenregistrer, enregistrable ou réenregistrable) est similaire à celui du CD; la seule différence est la taille des structures inscrites sur le disque.

Par exemple, sur un DVD préenregistré les microcuvettes sont à peu près deux fois plus petites que sur les CD (400 nm au lieu de 830 nm) et la spirale est beaucoup plus serré puisque les sillons ne sont séparés que par 0,74 µm au lieu de 1,6 µm. L'information est ainsi stockée sur une spirale de 11 km de long !



Logique séquentielle – Compteurs, registres et mémoires

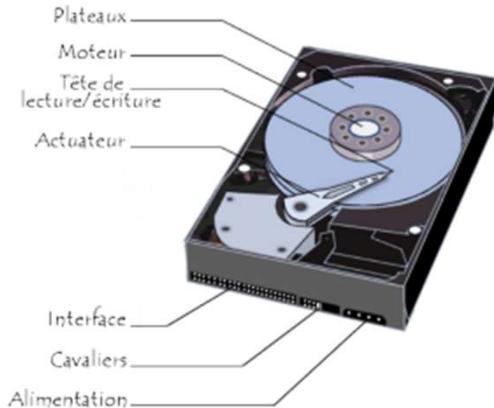


Les autres technologies de mémoire (10/11)

Le disque DUR

Le disque dur est relié à la carte-mère via un contrôleur de disque dur interface entre le processeur et le disque dur. Le contrôleur de disque dur gère les disques reliés, interprète les commandes envoyées par le processeur et les achemine au disque concerné. On distingue généralement les interfaces suivantes :

- ❖ IDE
- ❖ SCSI
- ❖ Serial ATA



La lecture et l'écriture se fait grâce à des têtes de lecture situées de part et d'autre de chacun des plateaux. Ce sont des électro-aimants qui se baissent et se soulèvent pour lire l'information ou l'écrire.

Les têtes sont à quelques microns de la surface, séparées par une couche d'air provoquée par la rotation des disques qui créent un vent d'environ 250km/h !

Ces têtes sont mobiles latéralement afin de pouvoir balayer l'ensemble de la surface du disque.

Logique séquentielle – Compteurs, registres et mémoires

Les autres technologies de mémoire (11/11)

Mémoire SSD

De plus en plus de systèmes de stockage proposent des disques SSD afin d'améliorer les performances. L'utilisation des technologies à base de **mémoire flash** bouleverse le stockage. En effet, dans le cas d'un disque SSD, le temps de lecture et le temps d'écriture peuvent être très différents.

Le débit des SSD est généralement au-dessus de 200 Mo/s, et de l'ordre de 500 Mo/s pour les disques les plus performants, à comparer aux valeurs de l'ordre de 50 à 120 Mo/s atteintes par les disques durs traditionnels.



Logique séquentielle – Compteurs, registres et mémoires

Mémoires - Hiérarchie (1/2)

Cette classification est réalisée grâce à la performance des mémoires. La rapidité d'une mémoire se mesure grâce à deux paramètres :

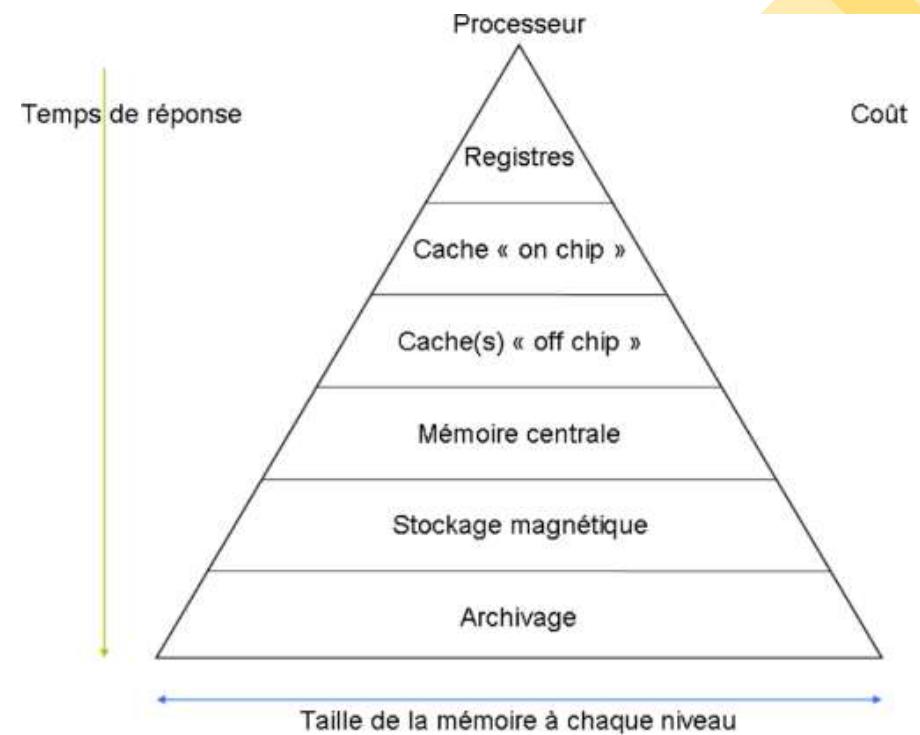
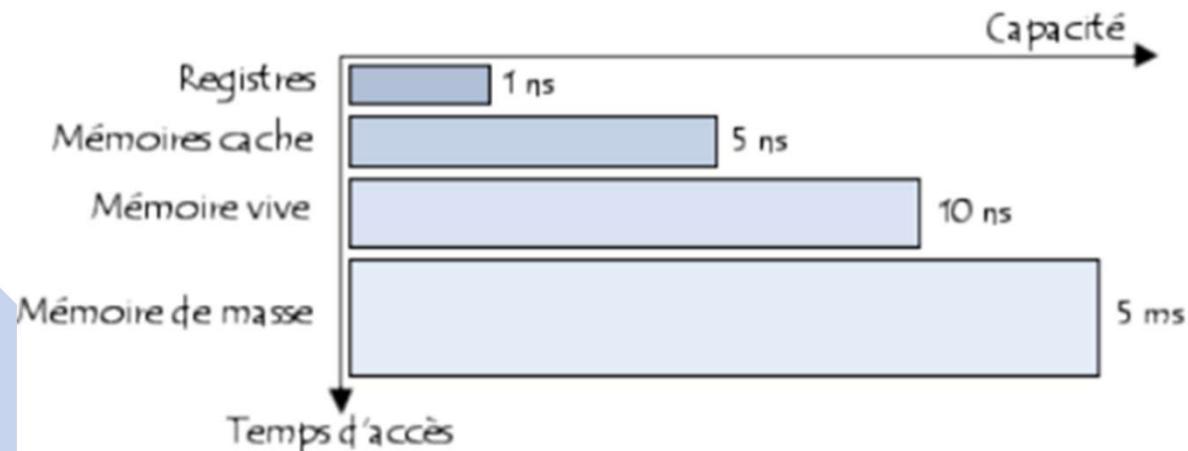
- ❖ **le temps de latence** : c'est le temps pour effectuer une opération de lecture ou une écriture : plus celui-ci est bas, plus la mémoire sera rapide;
- ❖ **le débit mémoire** correspond à la quantité d'informations récupérées ou enregistrées en une seconde dans la mémoire : plus celui-ci est élevé, plus la mémoire sera rapide;
- ❖ **la lenteur d'une mémoire dépend de sa capacité** : plus la capacité est importante, plus la mémoire est lente. Si l'on souhaitait utiliser une seule grosse mémoire dans notre ordinateur, celle-ci serait trop lente et l'ordinateur serait inutilisable.

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Hiérarchie (2/2)

Pour résoudre ce problème, il suffit d'utiliser plusieurs mémoires de taille et de vitesse différentes. Des mémoires très rapides de faible capacité seconderont des mémoires lentes de capacité importante.

On peut regrouper ces mémoires en niveaux : toutes les mémoires appartenant à un même niveau de vitesse. Pour simplifier, il existe quatre grands niveaux de hiérarchie mémoire, indiqués dans les graphiques suivants :



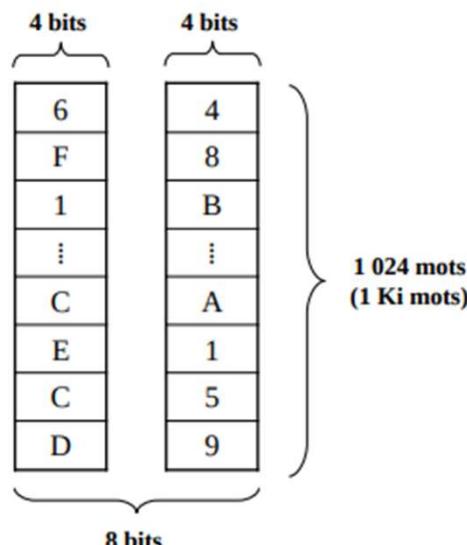
Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

Il est possible d'assembler plusieurs mémoires de même type afin d'accroître la largeur et/ou la profondeur de la mémoire. C'est le type d'assemblage que l'on retrouve sur les modules SIMM et DIMM.

L'assemblage en parallèle

L'assemblage en parallèle permet d'accroître la largeur d'une mémoire. C'est-à-dire la taille des mots qu'elle contient. Cet assemblage augmente donc le nombre de fils du bus de donnée. Prenons l'exemple de deux mémoires RAM possédant chacune un bus d'adresse de 10 bits et un bus de donnée de 4 bits. Ces RAM ont donc une profondeur de 1024 mots (2^{10}) et une largeur de 4 bits par mot.

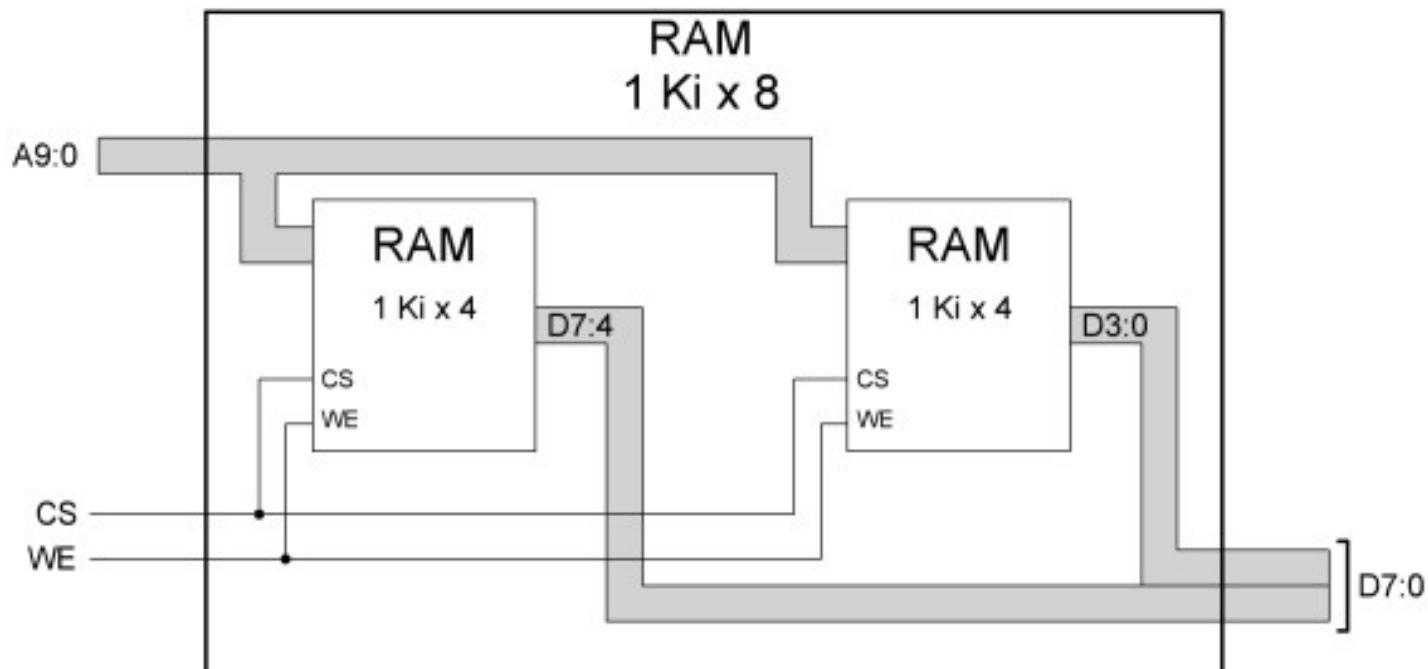


Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

L'assemblage en parallèle (suite)

On peut très bien imaginer que ces deux mémoires d'une largeur de 4 bits par mot ne forment en fait qu'une seule mémoire d'une largeur de 8 bits par mot. Pour arriver à ce résultat, il faut câbler les Mémoires de la façon suivante :



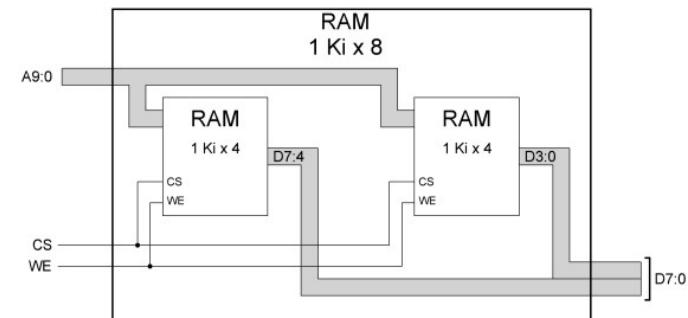
Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

L'assemblage en parallèle (suite)

On remarque que :

- ❖ Les bus d'adresse des mémoires internes sont reliés. Ils forment le bus d'adresse de la mémoire externe ;
- ❖ Les bus de contrôle des mémoires internes sont reliés. Ils forment le bus de contrôle de la mémoire externe ;
- ❖ Les bus de donnée des mémoires internes ne sont pas reliés. Ils sont simplement juxtaposés. Cette juxtaposition forme le bus de donnée de la mémoire externe ;
- ❖ Les deux mémoires doivent être actives en même temps afin d'obtenir une donnée complète sur le bus de donnée de la mémoire externe.



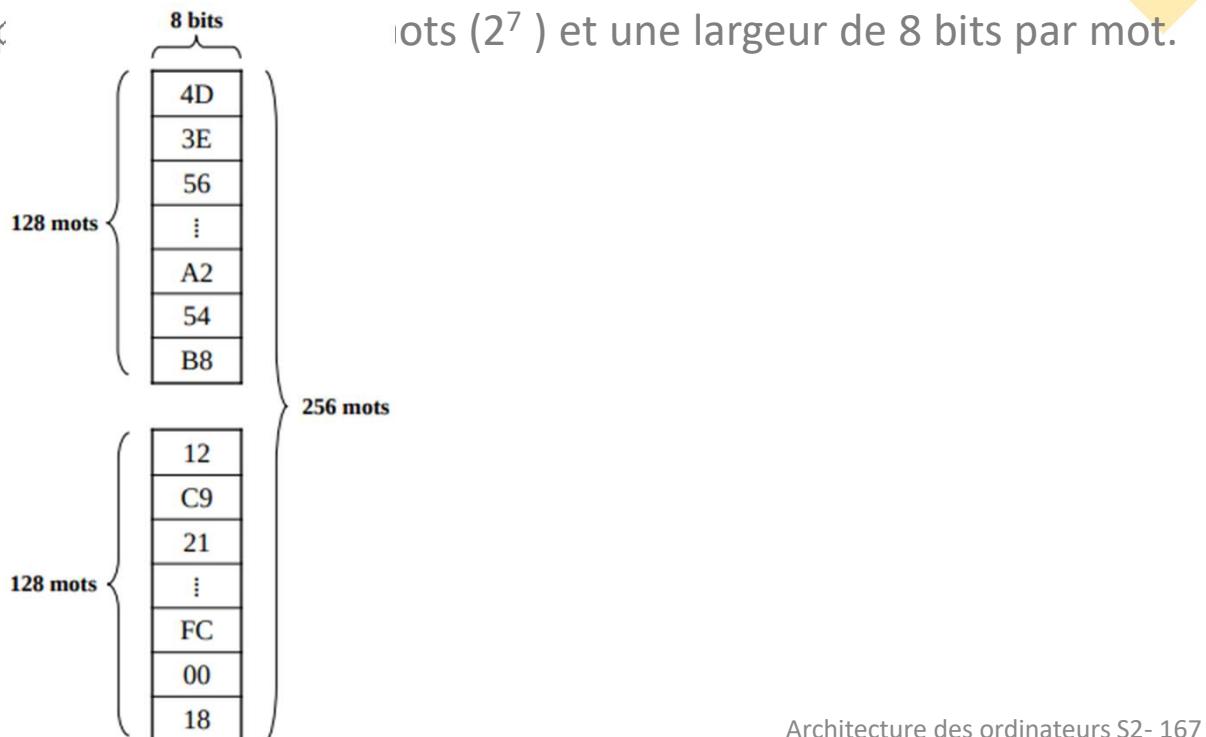
Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

L'assemblage en série

L'assemblage en série permet d'accroître la profondeur d'une mémoire (**nombre de mots**). C'est-à-dire le nombre de mots

qu'elle contient. Cet assemblage augmente donc le nombre de fils du bus d'adresse. Prenons l'exemple de deux mémoires RAM possédant chacune un bus d'adresse de 7 bits et un bus de donnée de 8 bits. Ces RAM ont donc une p



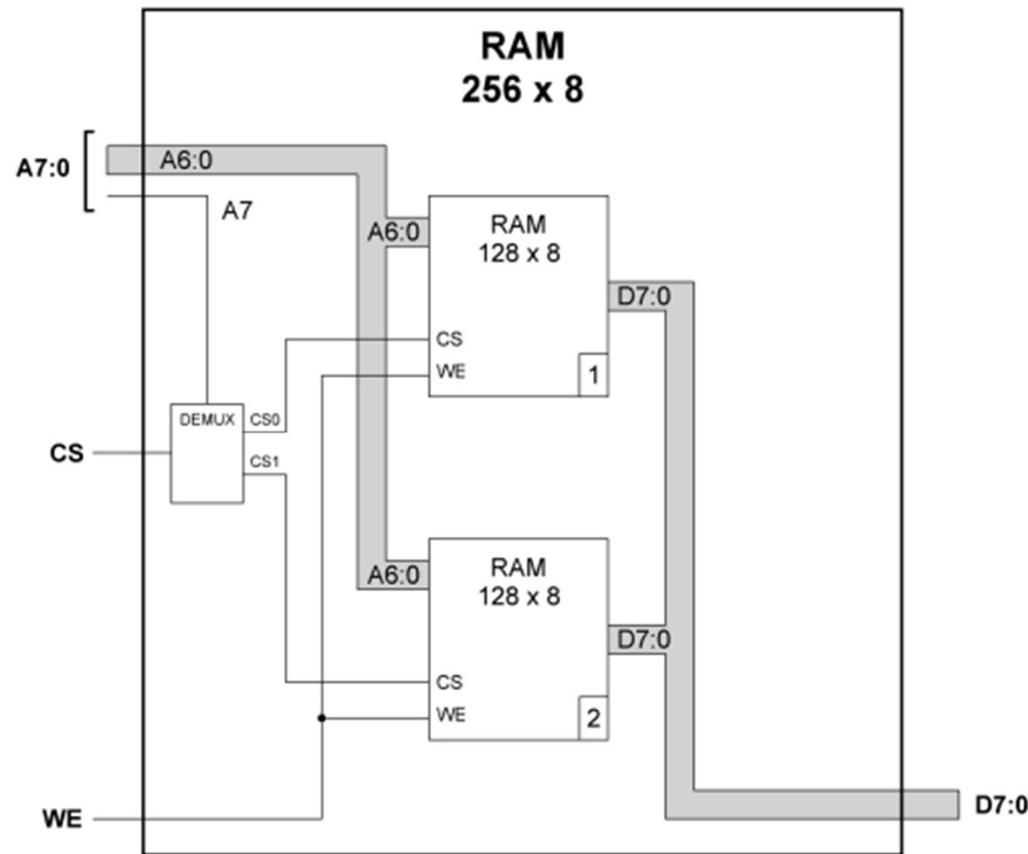
ots (2^7) et une largeur de 8 bits par mot.

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

L'assemblage en série (suite)

On peut très bien imaginer que ces deux mémoires d'une profondeur de 128 mots ne forment en fait qu'une seule mémoire d'une profondeur de 256 mots.



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

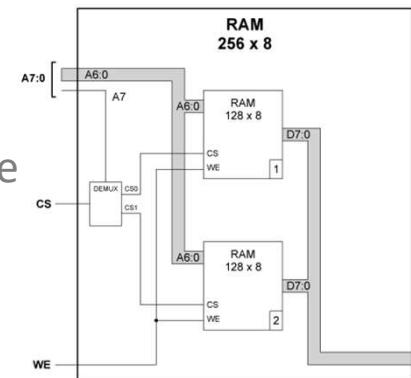
L'assemblage en série (suite)

- ❖ Les bus de donnée des mémoires internes sont reliés et Ils forment le bus de donnée la mémoire externe ;
- ❖ À l'exception des entrées CS, les bus de contrôle des mémoires internes sont reliés. Ils forment le bus de contrôle de la mémoire externe ;
- ❖ Les bus d'adresse des mémoires internes sont reliés. Ils forment les bits de poids faible du bus d'adresse de la mémoire externe ;
- ❖ Il faut ajouter un bit d'adresse à la mémoire externe afin d'obtenir les 256 adresses ($2^8 = 256$). Ce bit supplémentaire va servir à sélectionner l'une des mémoires internes à l'aide d'un démultiplexeur ;
- ❖ Les mémoires internes ne doivent jamais être actives en même temps afin de ne pas créer un conflit au niveau du bus de donnée. Le CS de la mémoire externe est donc propagé vers l'une des mémoires internes pendant que l'autre est désactivée.

Dans le cas où le CS de la mémoire externe est à 1, on a les deux cas suivants :

- Si $A7 = 0$ alors $CS0 = 1$ et $CS1 = 0 \rightarrow$ c'est la mémoire interne numéro 1 qui est active ;
- Si $A7 = 1$ alors $CS0 = 0$ et $CS1 = 1 \rightarrow$ c'est la mémoire interne numéro 2 qui est active.

Dans le cas où le CS de la mémoire externe est à 0, alors toutes les mémoires internes sont désactivées.

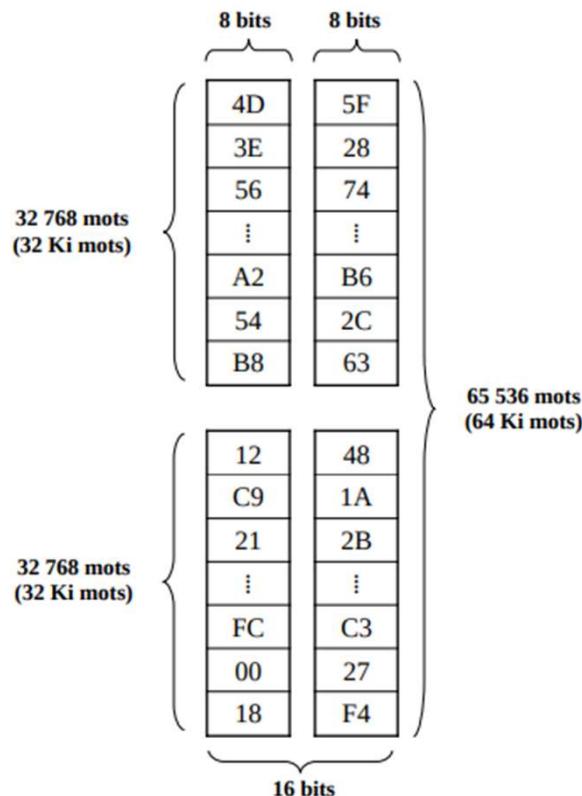


Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

L'assemblage en parallèle et en série

Il est possible de combiner les deux types d'assemblage. Prenons l'exemple de quatre mémoires RAM possédant chacune un bus d'adresse de 15 bits et un bus de donnée de 8 bits. Ces RAM ont donc une profondeur de 32 768 mots (2^{15}) et une largeur de 8 bits par mot.



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

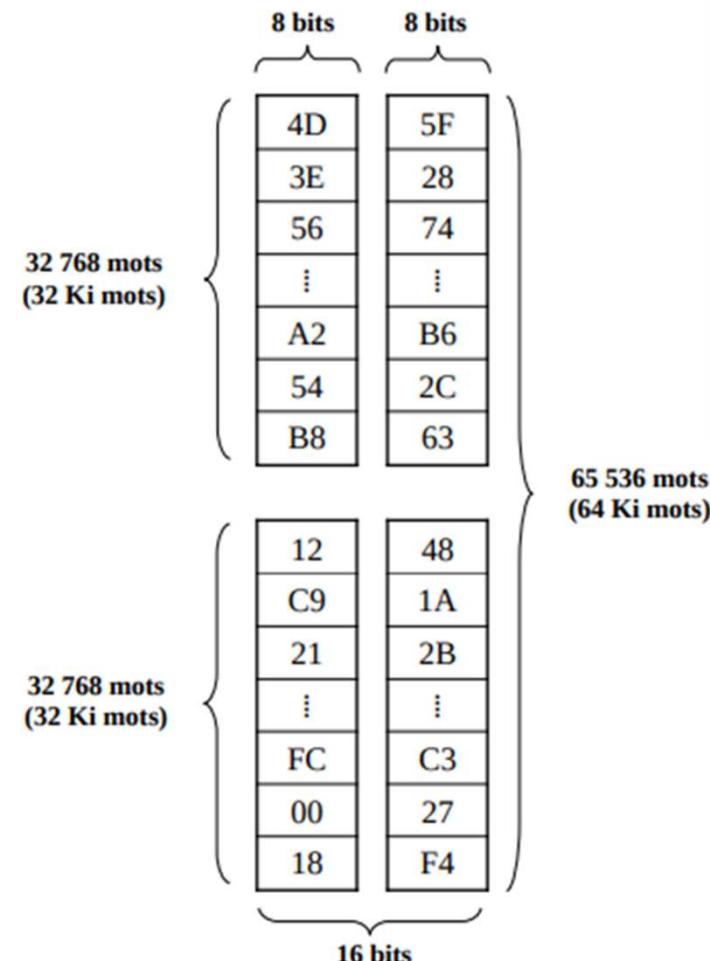
L'assemblage en parallèle et en série (suite)

On peut très bien imaginer que ces quatre mémoires ne forment en fait qu'une seule mémoire d'une profondeur de 65 536 mots et d'une largeur de 16 bits par mot.

Les deux mémoires du haut doivent être actives en même temps afin d'obtenir une donnée complète sur 16 bits (par exemple, 4D5F16 est le contenu de l'adresse 0).

Il en est de même pour les deux mémoires du bas (par exemple, 18F416 est le contenu de l'adresse 65 535).

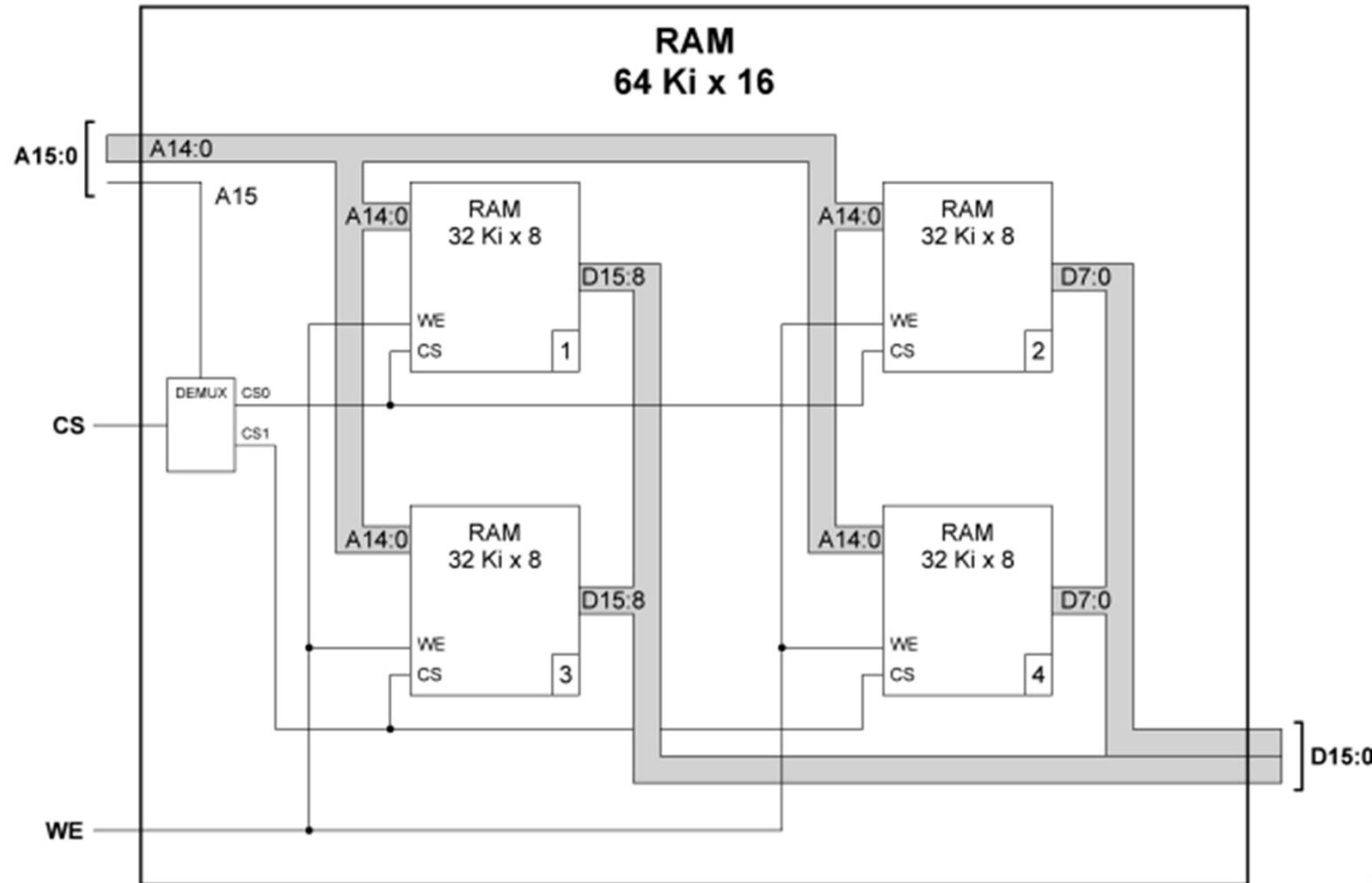
Mais attention, quand les deux mémoires du haut sont actives, alors les deux mémoires du bas doivent être inactives (et inversement), faute de quoi deux données différentes seraient positionnées sur le bus de donnée. Cela provoquerait alors un conflit d'accès.



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

L'assemblage en parallèle et en série (suite)



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Assemblage de la mémoire

L'assemblage en parallèle et en série (suite)

On reconnaît assez facilement les deux types d'assemblage :

- ❖ Les mémoires 1 et 2 sont en parallèle.

Elles doivent être actives en même temps (elles ont le même CS).

- ❖ Les mémoires 3 et 4 sont en parallèle.

Elles doivent être actives en même temps (elles ont le même CS).

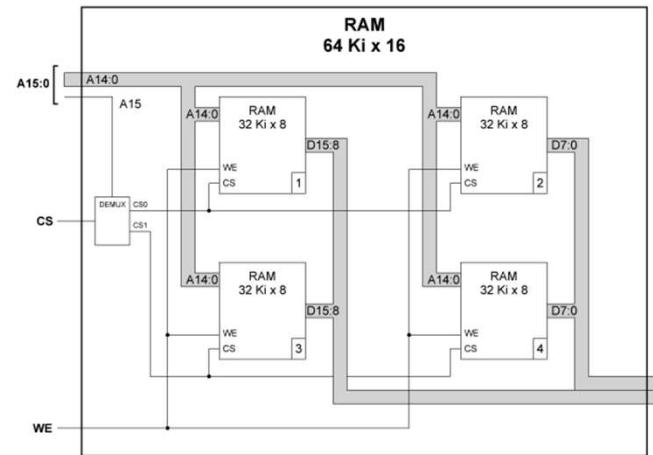
- ❖ Les mémoires 1 et 3 sont en série. Elles ne doivent jamais être actives en même temps.

- ❖ Les mémoires 2 et 4 sont en série. Elles ne doivent jamais être actives en même temps.

Dans le cas où le CS de la mémoire externe est à 1, on a les deux cas suivants :

- Si $A15 = 0$ alors $CS0 = 1$ et $CS1 = 0 \rightarrow$ ce sont les mémoires 1 et 2 qui sont actives ;
- Si $A15 = 1$ alors $CS0 = 0$ et $CS1 = 1 \rightarrow$ ce sont les mémoires 3 et 4 qui sont actives.

Dans le cas où le CS de la mémoire externe est à 0, alors toutes les mémoires internes sont désactivées.



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Décodage d'adresse

L'objectif du décodage d'adresse est de faire communiquer un microprocesseur avec différents composants (mémoires, cartes, périphériques, etc.).

Afin d'illustrer les principales techniques de décodage d'adresse que nous aborderons dans la suite de ce cours, nous allons nous servir d'un exemple concret :

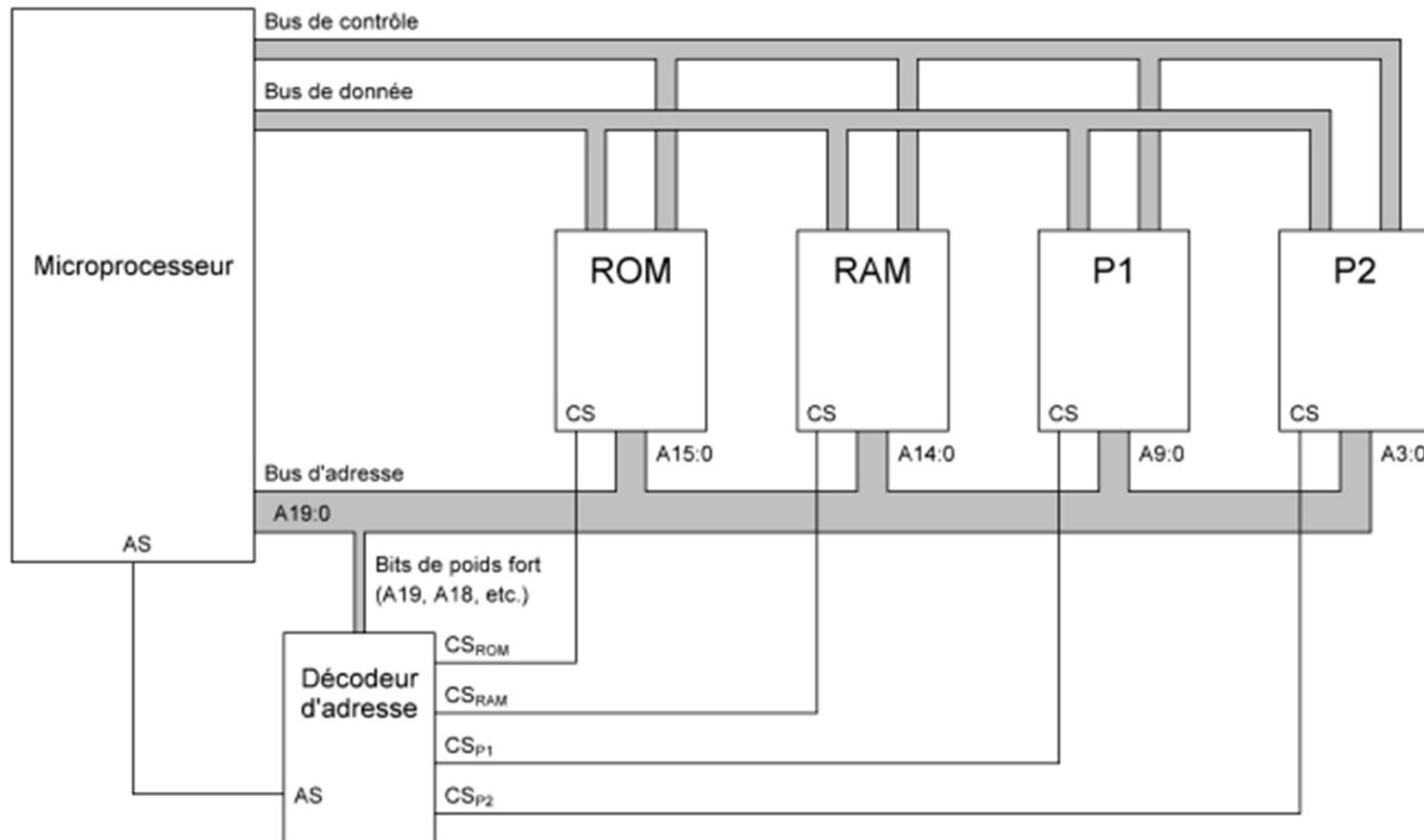
Nous souhaitons faire communiquer un microprocesseur avec quatre composants : une mémoire de type ROM, une mémoire de type RAM et deux périphériques quelconques que l'on notera **P1** et **P2**.

Composant	Bus d'adresse	Bus de données	Capacité d'adressage
Microprocesseur	20 fils	8 fils	1 Mio
ROM	16 fils	8 fils	64 Kio
RAM	15 fils	8 fils	32 Kio
P1	10 fils	8 fils	1 Kio
P2	4 fils	8 fils	16 octets

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Décodage d'adresse (Suite)

La communication entre le microprocesseur et les différents composants se fait à l'aide des bus de donnée, d'adresse et de contrôle. La sélection d'un composant nécessite un décodeur d'adresse.



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Décodage d'adresse (Suite)

Les sorties du décodeur d'adresse sont les entrées CS de tous les composants. Ses entrées sont le bus d'adresse et le signal AS (Address Strobe). Ce dernier est généré par le microprocesseur (bus de contrôle). Il est actif ($AS = 1$) quand l'adresse présente sur le bus d'adresse du microprocesseur est valide.

Par conséquent, quand il est inactif ($AS = 0$) aucun des composants ne doit être activé. C'est pourquoi le décodeur d'adresse doit prendre ce signal en compte.

Tous les composants souhaitant échanger des informations avec le microprocesseur doivent posséder une entrée d'activation CS, car un seul composant à la fois doit accéder aux bus. Si plusieurs composants accèdent aux bus en même temps, cela produira un conflit d'accès.

Le microprocesseur utilise son bus d'adresse pour sélectionner un composant.

Le décodage d'adresse consiste donc à décoder l'adresse présente sur le bus d'adresse du microprocesseur et à activer un seul composant en fonction de la valeur de cette adresse.

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Décodage d'adresse (Suite)

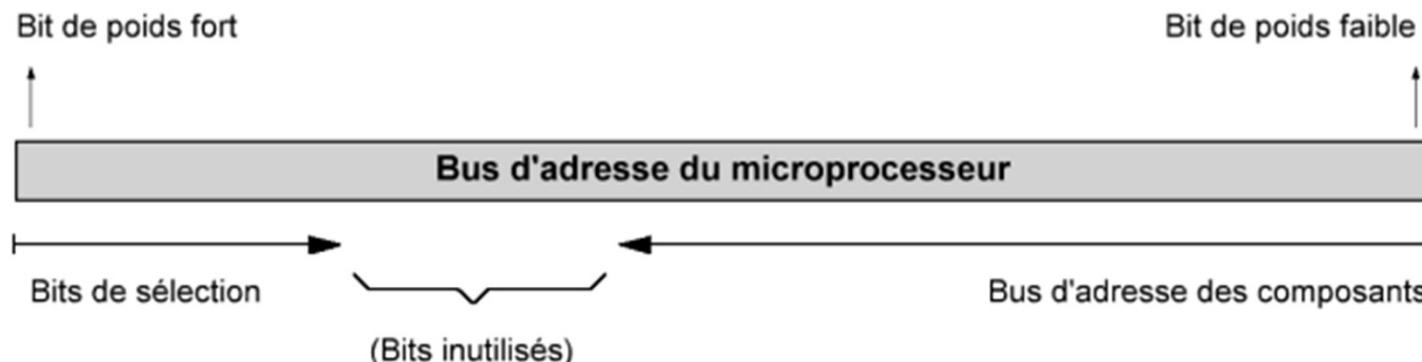
Un décodeur d'adresse doit donc être mis en place. Ce dernier positionnera sur ses sorties la valeur des différents signaux d'activation CS en fonction de l'adresse présente sur le bus d'adresse du microprocesseur.

Il existe de nombreuses techniques de décodage d'adresse. Nous nous limiterons à deux de ces techniques : le décodage linéaire et le décodage par zone. L'apprentissage de ces deux types de décodage suffira à comprendre les principes et les concepts de base du décodage d'adresse.

Nous allons commencer par aborder les points communs à ces deux types de décodage :

Le bus d'adresse du microprocesseur est divisé en deux parties :

- Les bits de poids fort servent à la sélection des composants ;
- Les bits de poids faible servent à contenir les bus d'adresse des composants.



Logique séquentielle – Compteurs, registres et mémoires

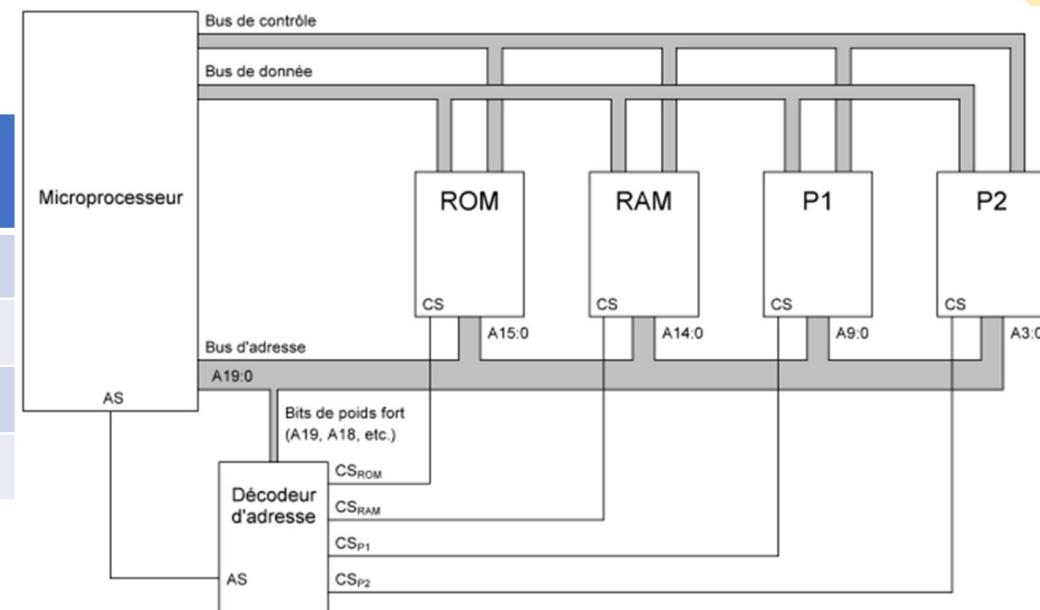
Mémoires – Le décodage linéaire

Le principe du décodage linéaire est d'associer un bit de sélection par composant (en partant du bit de poids fort).

Dans notre exemple, quatre composants nécessitent quatre bits de sélection : **A19, A18, A17 et A16**.

Chacun de ces bits doit être associé à un composant. Dans l'absolu, le choix d'associer tel composant à tel bit de sélection est arbitraire. Le concepteur fera un choix en fonction de ses besoins et de ses contraintes.

Bit de sélection	Composant associé	Composant activé
A16 = 1	ROM	OUI
A17 = 1	RAM	OUI
A18 = 1	P1	OUI
A19 = 1	P2	OUI



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage linéaire - Décodeur d'adresse

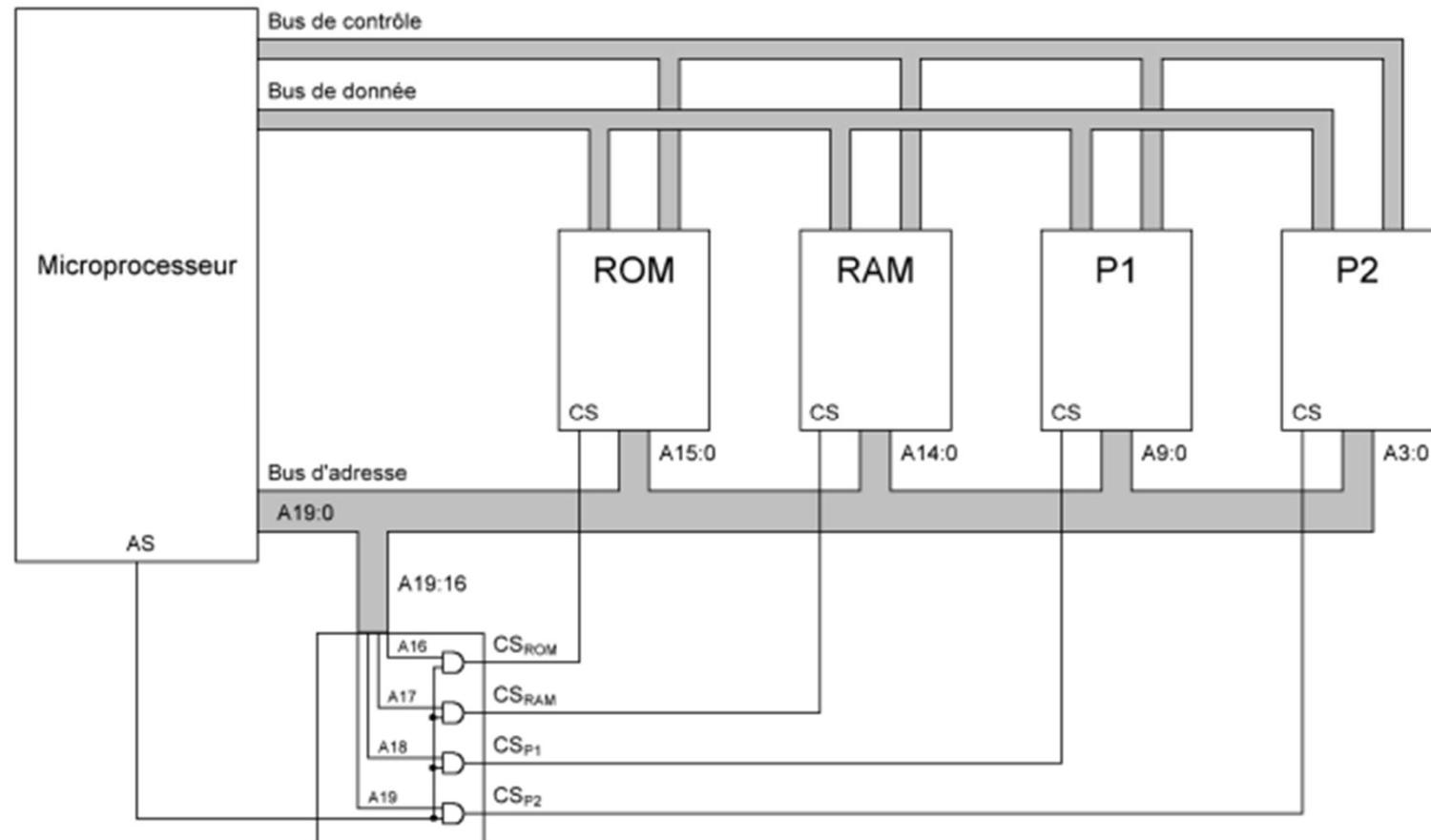
La logique d'un décodage linéaire est très simple. Le CS d'un composant doit être mis à 1 quand son bit de sélection est à 1. Aussi, comme cela a été dit précédemment, quand la valeur du bus d'adresse est invalide (AS = 0), aucun composant ne doit être activé.

$$CS_{ROM} = AS \cdot A16$$

$$CS_{RAM} = AS \cdot A17$$

$$CS_{P1} = AS \cdot A18$$

$$CS_{P2} = AS \cdot A19$$



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage linéaire - Représentation de l'espace mémoire

La représentation de l'espace mémoire d'un microprocesseur indique pour tous les composants, les adresses auxquelles ces derniers sont accessibles par le microprocesseur.

Il faut donc déterminer l'adresse la plus basse et l'adresse la plus haute de chaque composant dans l'espace mémoire.

Il faut pour cela déterminer les bits de poids fort (sélection) et les bits de poids faible (adressage du composant) du bus d'adresse du microprocesseur pour chaque composant. S'il existe des bits inutilisés, qui ne servent ni à l'adressage, ni à la sélection, ils seront positionnés à 0..

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage linéaire - Représentation de l'espace mémoire

Par exemple pour la ROM :

- A16 est à 1 pour sélectionner la ROM ;
- A17, A18 et A19 sont à 0 pour désactiver les autres composants ;
- Pour son adresse basse : on positionne ses 16 bits d'adresse à 0 ;
- Pour son adresse haute : on positionne ses 16 bits d'adresse à 1.

ROM basse : **0001 0000 0000 0000 0000₂** = 10000₁₆

ROM haute : **0001 1111 1111 1111 1111₂** = 1FFFF₁₆

RAM basse : **0010 0000 0000 0000 0000₂** = 20000₁₆

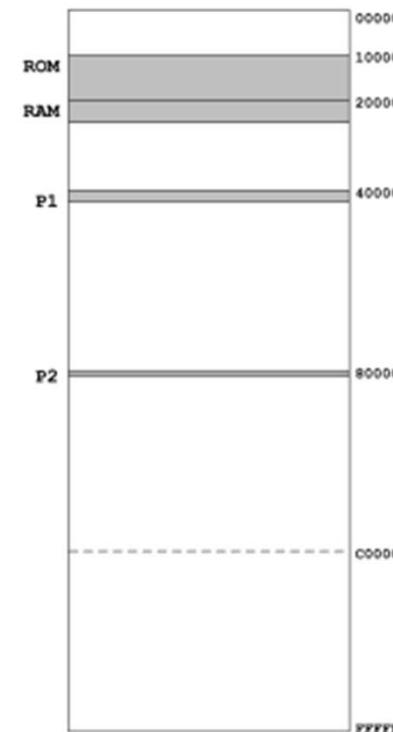
RAM haute : **0010 0111 1111 1111 1111₂** = 27FFF₁₆

P1 basse : **0100 0000 0000 0000 0000₂** = 40000₁₆

P1 haute : **0100 0000 0011 1111 1111₂** = 403FF₁₆

P2 basse : **1000 0000 0000 0000 0000₂** = 80000₁₆

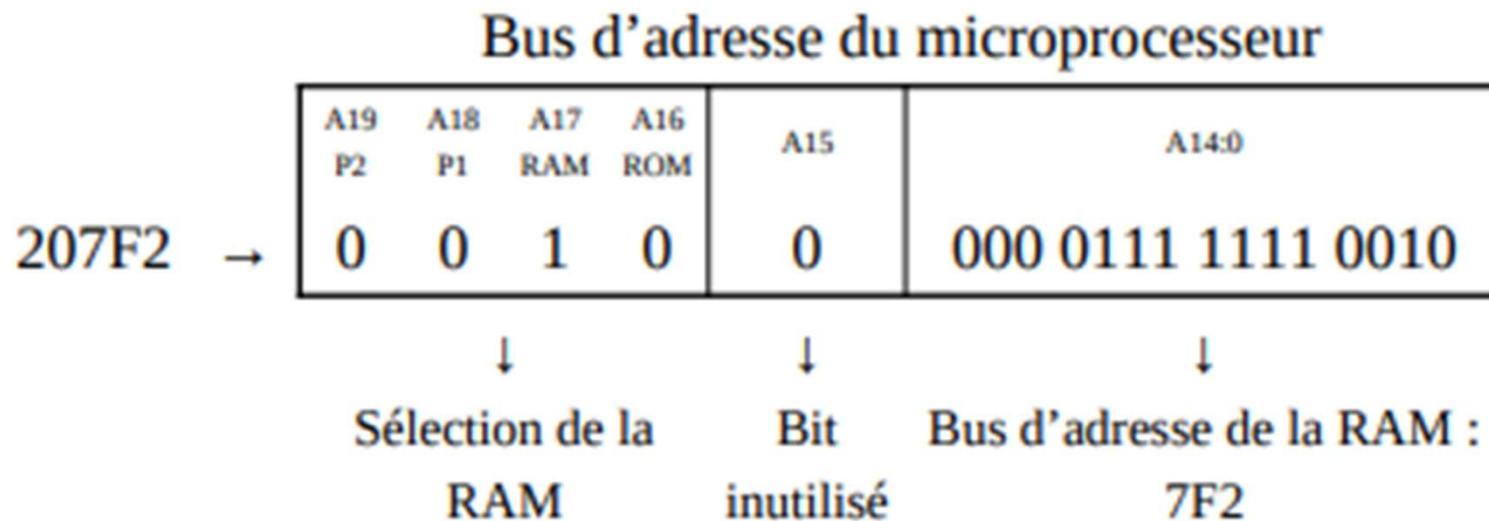
P2 haute : **1000 0000 0000 0000 1111₂** = 8000F₁₆



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage linéaire - Représentation de l'espace mémoire

Par exemple, si le microprocesseur positionne l'adresse 207F216 sur son bus d'adresse, alors c'est l'adresse 7F216 de la RAM qui est sélectionnée :



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage linéaire - Représentation de l'espace mémoire

Finalement, quelques caractéristiques du décodage d'adresse linéaire :

- Il est facile à mettre en œuvre : la logique de décodage ne contient que des fils ;
- Le nombre de composants connectables est vite limité : associer un fil par composant n'est pas très optimisé ;
- Il ne protège pas contre les conflits d'accès.

Remarque :

Le principal défaut de ce décodage est qu'il ne protège pas contre les conflits d'accès. C'est-à-dire que si le microprocesseur positionne sur son bus d'adresse une valeur qui contient plusieurs bits de sélection à 1, alors plusieurs composants seront sélectionnés en même temps.

Il est toutefois possible de modifier facilement la logique de décodage afin de protéger les composants. Il suffit d'activer le CS d'un composant uniquement si les autres sont désactivés. C'est-à-dire si le bit de sélection qui leur est associé est à 0. Ce qui donne :

$$CSROM = AS \cdot \overline{A19} \cdot \overline{A18} \cdot \overline{A17} \cdot A16$$

$$CSRAM = AS \cdot \overline{A19} \cdot \overline{A18} \cdot A17 \cdot \overline{A16}$$

$$CSP1 = AS \cdot \overline{A19} \cdot A18 \cdot \overline{A17} \cdot \overline{A16}$$

$$CSP2 = AS \cdot A19 \cdot \overline{A18} \cdot \overline{A17} \cdot \overline{A16}$$

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone

Le principe du décodage par zone est de diviser l'espace mémoire en un certain nombre de zones de même taille. Une zone correspond à une combinaison des bits de sélection. Les bits de sélection sont toujours les bits de poids fort.

Par exemple :

- 1 bit de sélection donne 2 zones ;
- 2 bits de sélection donnent 4 zones ;
- n bits de sélection donnent 2^n zones.

Dans notre exemple, le plus grand bus d'adresse est celui de la ROM qui possède 16 fils. Il reste donc quatre fils disponibles pour la sélection. Ce qui nous laisse les quatre possibilités suivantes :

- 1 fil de sélection → 2 zones de 512 Kio ;
- 2 fils de sélection → 4 zones de 256 Kio ;
- 3 fils de sélection → 8 zones de 128 Kio ;
- 4 fils de sélection → 16 zones de 64 Kio.

1 bit de sélection A19	
Zone 0	0
Zone 1	1

2 bits de sélection A19:18	
Zone 0	00
Zone 1	01
Zone 2	10
Zone 3	11

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone (suite)

Le décodage par zone permet d'associer un composant par zone. Dans notre exemple nous avons quatre composants, il nous faudra au moins quatre zones ; c'est-à-dire au moins deux fils de sélection. Il est possible d'avoir plus de zones que de composants. Les zones restantes seront inutilisées.

Il faut maintenant associer les composants à leur zone. Dans l'absolu, le choix du nombre de zones et celui d'associer tel composant à telle zone sont arbitraires.

Le concepteur fera un choix en fonction de ses besoins et de ses contraintes. Des zones vides peuvent laisser la possibilité d'ajouter d'autres composants par la suite et donc de faire évoluer plus facilement le système.

Pour la suite de l'exemple, nous choisirons de diviser l'espace mémoire en huit zones et d'associer les composants aux quatre premières zones :

Zone	A19	A18	A17	Composant
0	0	0	0	ROM
1	0	0	1	RAM
2	0	1	0	P1
3	0	1	1	P2
4	1	0	0	zone inutilisée
5	1	0	1	zone inutilisée
6	1	1	0	zone inutilisée
7	1	1	1	zone inutilisée

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone – Décodeur d'adresse

Le CS de chaque composant doit être activé lorsque la combinaison de bits associée au composant est présente sur les bits de sélection. Aussi, comme cela a été dit précédemment, quand la valeur du bus d'adresse est invalide (AS = 0), aucun composant ne doit être activé.

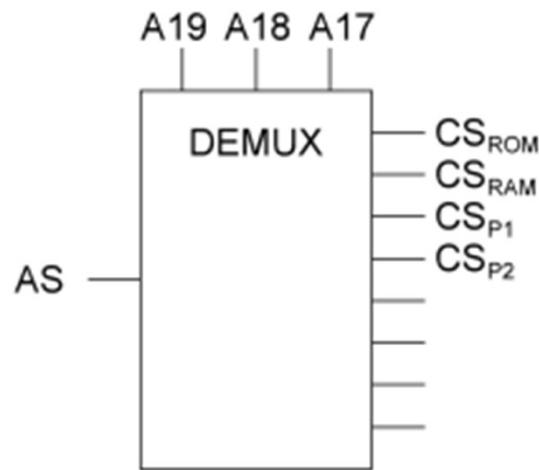
$$CS_{ROM} = AS \cdot \overline{A19} \cdot \overline{A18} \cdot \overline{A17}$$

$$CS_{RAM} = AS \cdot \overline{A19} \cdot \overline{A18} \cdot A17$$

$$CS_{P1} = AS \cdot \overline{A19} \cdot A18 \cdot \overline{A17}$$

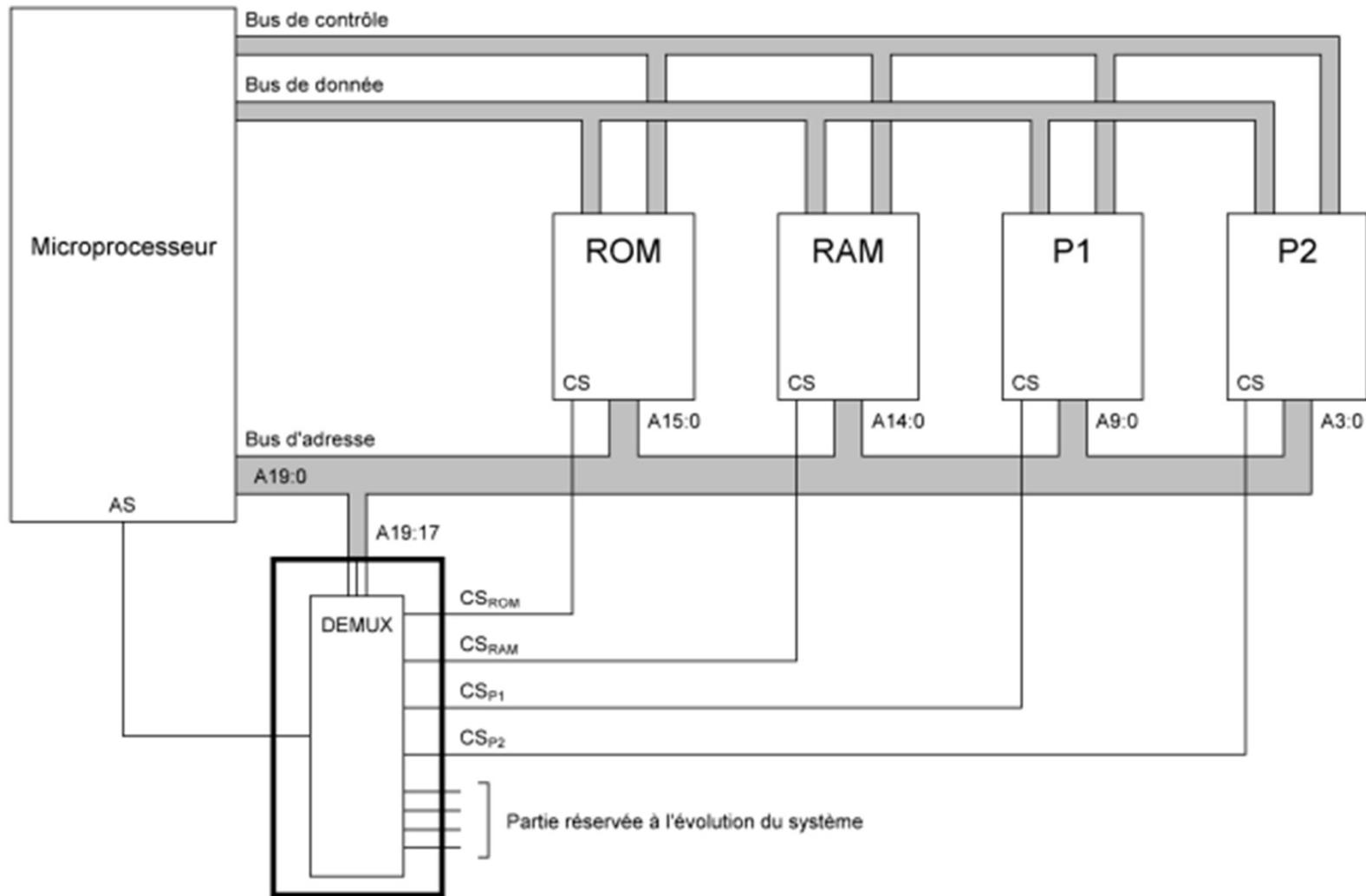
$$CS_{P2} = AS \cdot \overline{A19} \cdot A18 \cdot A17$$

Cette logique de décodage représente celle d'un démultiplexeur (3 lignes de sélection, 8 sorties).



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone – Câblage de la mémoire



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone – Représentation de l'espace mémoire

Le principe est le même que pour le décodage linéaire. Il faut déterminer l'adresse la plus basse et l'adresse la plus haute de chaque composant dans l'espace mémoire.

Il faut pour cela déterminer les bits de poids fort (sélection) et les bits de poids faible (adressage du composant) du bus d'adresse du microprocesseur pour chaque composant.

S'il existe des bits inutilisés, qui ne servent ni à l'adressage ni à la sélection, ils seront positionnés à 0.

Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone – Représentation de l'espace mémoire (suite)

Par exemple pour la ROM :

- A16 = 0, car il est inutilisé ;
- A17, A18 et A19 sont à 0 pour la sélectionner ;
- Pour son adresse basse : on positionne ses 16 bits d'adresse à 0 ;
- Pour son adresse haute : on positionne ses 16 bits d'adresse à 1.

ROM basse : 0000 0000 0000 0000 0000₂ = 00000₁₆

ROM haute : 0000 1111 1111 1111 1111₂ = 0FFFF₁₆

RAM basse : 0010 0000 0000 0000 0000₂ = 20000₁₆

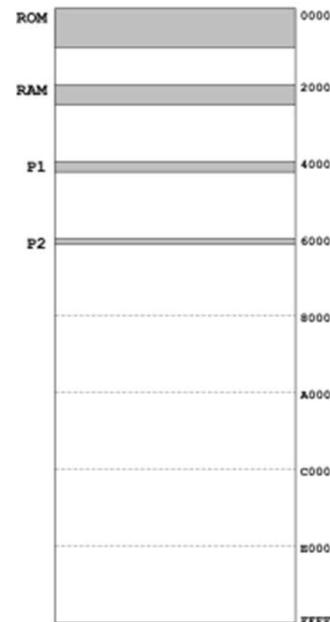
RAM haute : 0010 0111 1111 1111 1111₂ = 27FFF₁₆

P1 basse : 0100 0000 0000 0000 0000₂ = 40000₁₆

P1 haute : 0100 0000 0011 1111 1111₂ = 403FF₁₆

P2 basse : 0110 0000 0000 0000 0000₂ = 60000₁₆

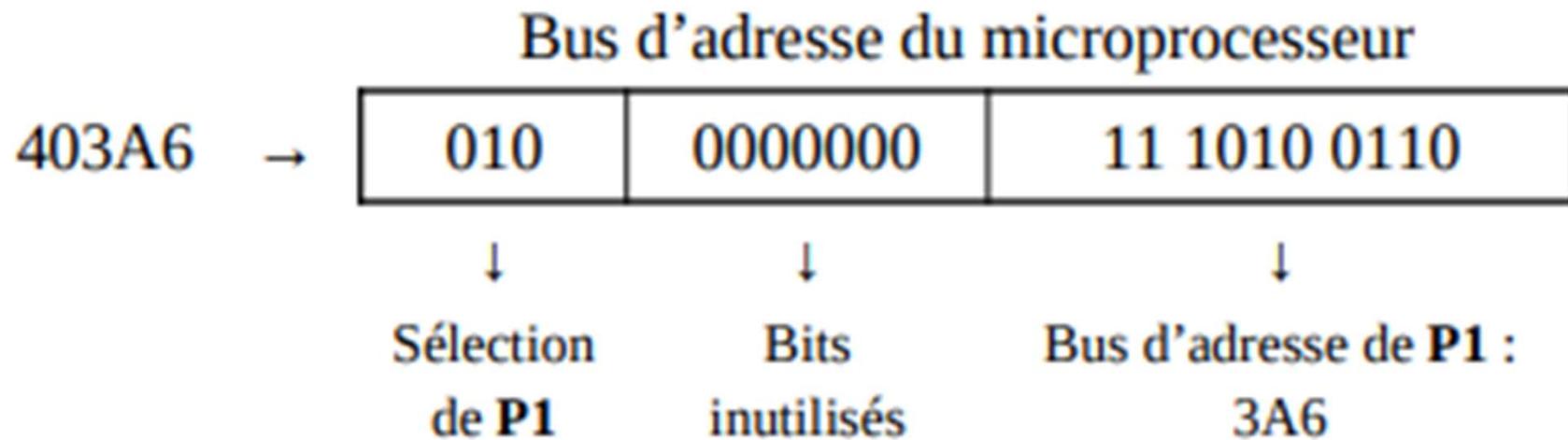
P2 haute : 0110 0000 0000 0000 1111₂ = 6000F₁₆



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone – Représentation de l'espace mémoire (suite)

Par exemple, si le microprocesseur positionne l'adresse 403A616 sur son bus d'adresse, alors c'est l'adresse 3A616 de P1 qui est sélectionnée :



Logique séquentielle – Compteurs, registres et mémoires

Mémoires – Le décodage par zone – Représentation de l'espace mémoire

Conclusion

Quelques caractéristiques du décodage d'adresse par zone :

- Il est facile à mettre en œuvre : la logique de décodage est celle d'un décodeur ;
- Le nombre de composants connectables est plus important que celui d'un décodage linéaire ;
- Plus le nombre de zones est élevé, plus la taille des composants est réduite (et inversement) ;
- Aucun risque de conflit d'accès.

Les BUS (1/6)

Bus et transports en commun

Un bus est une voie de communication, formée d'un ou plusieurs conducteurs, sur laquelle sont disposées plusieurs connexions.

Tout comme les passagers des transports en commun qui vont d'un arrêt à l'autre, les données circulent sur le bus elles aussi d'une connexion à l'autre.

L'analogie s'arrête là car à un moment donné il n'y a jamais qu'un seul équipement qui transmet des données.

Ce concept de "bus" trouve son application aussi bien dans l'unité centrale (bus système, bus d'extension) que pour les connexions vers les périphériques (bus SCSI, USB Universal Serial Bus, Firewire) ou au niveau des topologies de réseaux.

Les BUS (2/6)

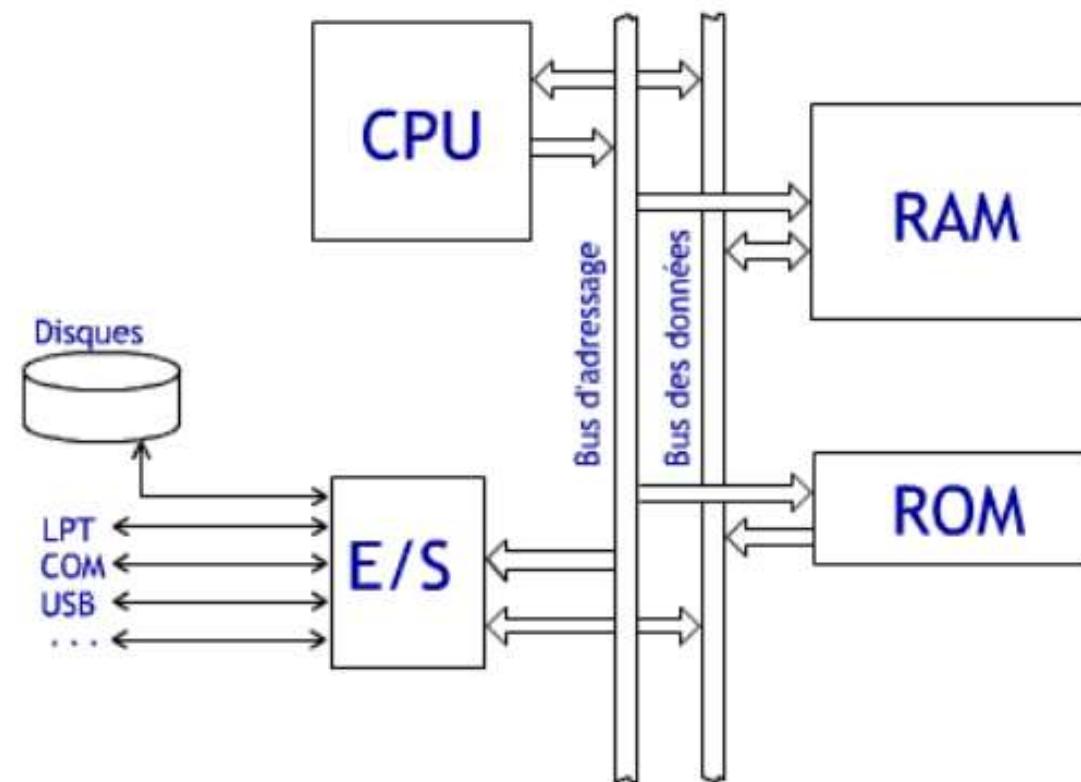
Le rôle du bus de la carte mère

Le processeur est relié aux autres éléments de la carte mère et aux cartes d'extension au moyen de lignes que les informations empruntent pour passer d'un composant à l'autre.

Ces lignes forment ce qu'on appelle le bus

Lors de l'étude du schéma bloc d'un ordinateur, on peut distinguer: le bus de donnée du bus d'adressage et du bus de contrôle.

Cette distinction nous a permis de discerner les catégories d'informations qui circulent entre les différents composants du PC.



Les BUS – Inventaire (3/6)

Les informations échangées entre la mémoire et le processeur circulent sur des bus. Un bus est un ensemble de n fils conducteurs (pistes), utilisés pour transporter n signaux binaires.

Le **bus d'adressage** est un bus unidirectionnel : seul le processeur envoie des adresses. On utilise donc des adresses de n bits.

La mémoire peut posséder au maximum 2^n emplacements (adresses 0 à 2^{n-1}).

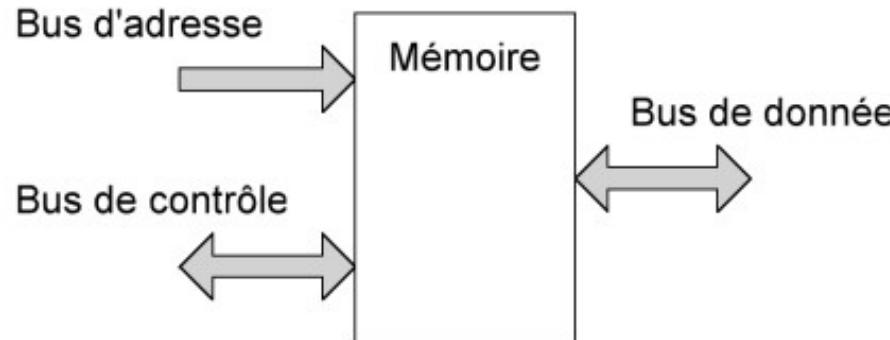
Le **bus de données** est un bus bidirectionnel. Lors d'une lecture, c'est la mémoire qui envoie un mot sur le bus (à l'adresse du contenu de l'emplacement demandé).

Lors d'une écriture, c'est le processeur qui envoie la donnée.

Le **bus de contrôle (ou commande)** est utilisé par le processeur pour communiquer avec les périphériques.

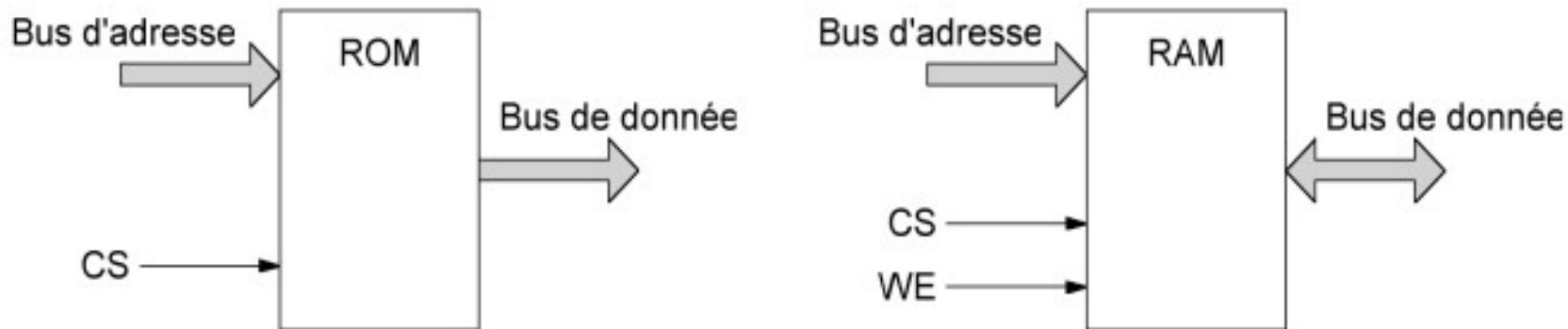
C'est un bus de sortie sur une mémoire de type ROM (la donnée peut uniquement être lue) et un bus bidirectionnel sur une mémoire de type RAM (la donnée peut être lue et écrite).

Si d est le nombre de fils du bus de donnée, alors on a la relation suivante : **largeur = d**.



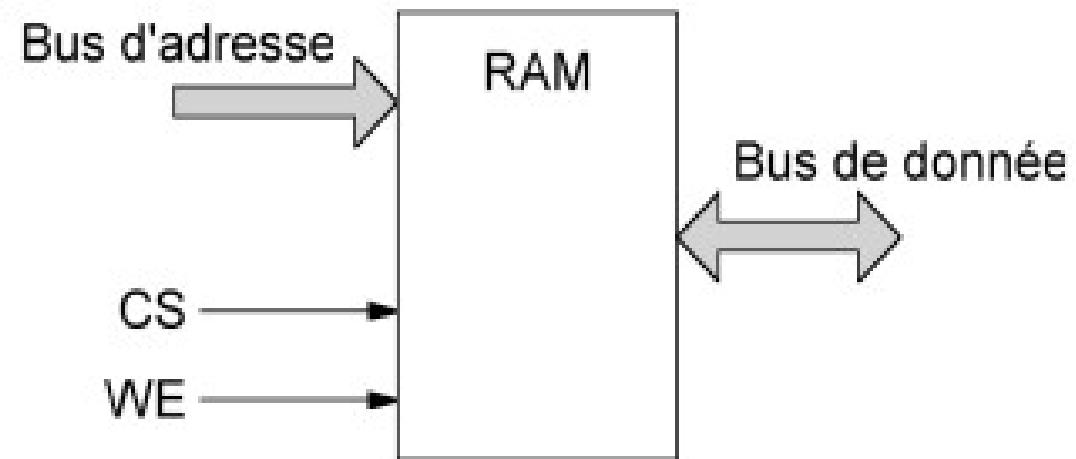
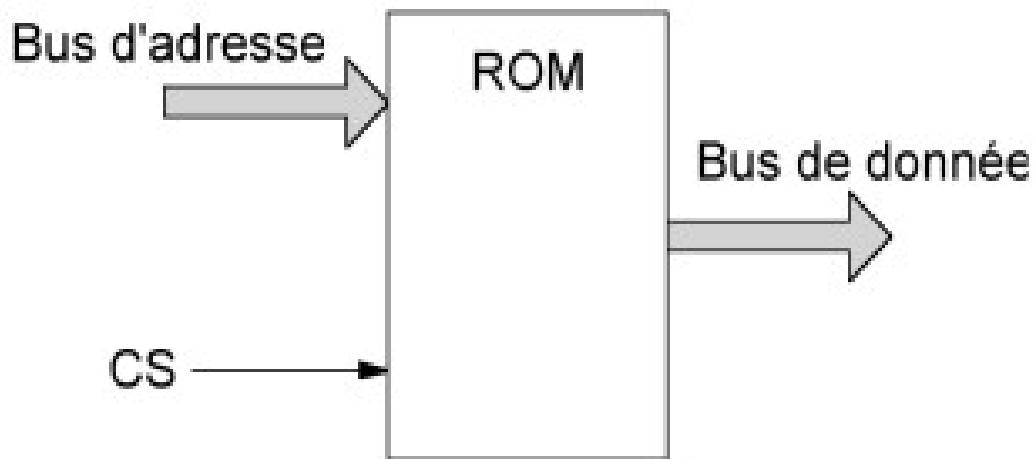
Les BUS – Bus de contrôle (4/6)

- ❖ Le bus de contrôle (ou bus de commande) véhicule les signaux de contrôle et de commande de la mémoire.
- ❖ Signaux coordonnent les échanges d'informations entre la mémoire et d'autres composants.
- ❖ Signaux peuvent être unidirectionnels (en entrées ou en sorties), d'autres peuvent être bidirectionnels. Le nombre de signaux disponibles sur ce bus varie en fonction du type de mémoire et de ses fonctionnalités.
- ❖ Le bus de contrôle réduit à son strict minimum.
- ❖ Cette entrée obligatoire (ROM ou RAM) permet d'activer ou de désactiver une mémoire.
- ❖ Nommée CS (Chip Select). Elle peut porter un nom différent d'une mémoire (exemple, CE pour Chip Enable).
- ❖ Une mémoire est désactivée, elle ignore la tension présente sur ses entrées et n'impose aucune tension sur ses sorties. Ses sorties sont dans un état de haute impédance (un troisième état ni 1, ni 0).



Les BUS – L'entrée de sélection de lecture/écriture (WE) (5/6)

- ❖ Cette entrée permet de choisir si l'accès à une mémoire doit se faire en lecture ou en écriture.
- ❖ Elle est obligatoire sur les mémoires de type RAM et n'existe pas sur les mémoires de type ROM. Nommée WE (Write Enable). Elle peut porter un nom différent d'une mémoire à une autre (par exemple, R/W pour Read / Write).



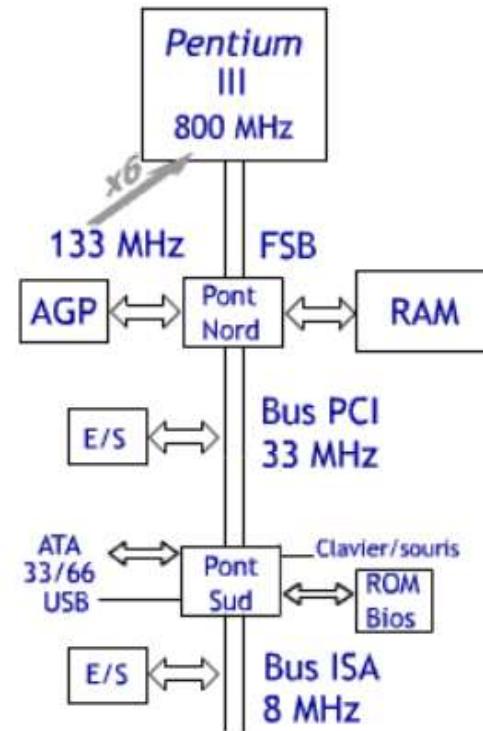
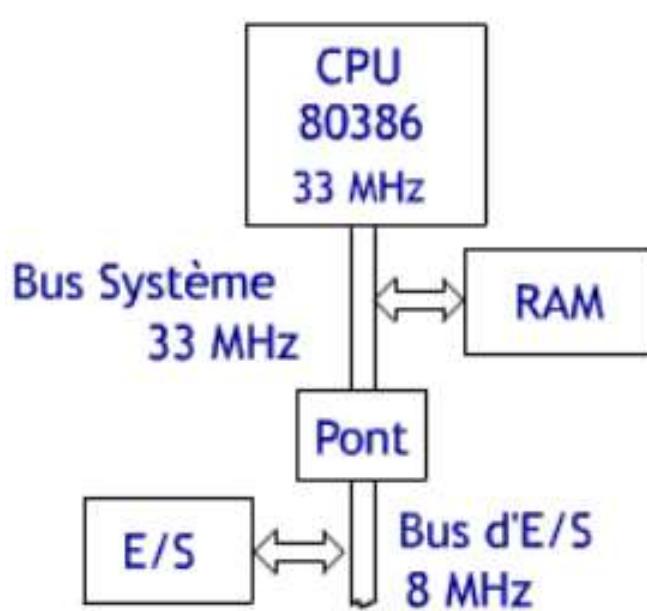
Les BUS – Bande Passante & Architectures (6/6)

La bande passante, aussi appelée taux de transfert, représente le débit maximum auquel peuvent circuler les données. Elle s'exprime en méga octets par seconde (Mo/s).

$$\text{Bande passante (en Mo/s)} = \text{largeur du bus (en octets)} \times \text{fréquence (en Hz)}$$

Il y a deux manières d'accroître ce débit : élargir la voie de circulation pour permettre le déplacement de plus de bits en parallèle ou accélérer la cadence des échanges (la fréquence).

Architecture multi-bus - Architecture Pont Nord / Pont Sud



ASSEMBLEUR – Préambule architecture d'un Système

Déroulement d'une opération de relève d'un compteur EDF :

Supposons que la société EDF vous demande par courrier de réaliser un relevé de compteur à cause d'un manque momentané de personnel. Vous allez réaliser les opérations suivantes:

1. **Allez** relever votre compteur électrique;
2. **Inscrivez** ces chiffres sur le bloc note;
3. **Cherchez** sur votre dernière quittance la valeur du dernier relevé;
4. **Inscrivez** celui-ci sur votre bloc note;
5. **Soustrayez**, à l'aide d'une calculatrice, ces deux nombres;
6. **Inscrivez** le résultat sur votre bloc note;
7. **Envoyez-nous** ce résultat en reportant sur la carte jointe à cet envoi;
8. Terminé. Merci de votre compréhension !

Evidemment, une lettre de ce style serait un peu déplacée car vous n'êtes pas un employé de la société et, en plus, vous êtes facturés pour ce service dans vos quittances EDF !

Mais analysons cette opération d'un point vu d'un système à microprocesseur (un ordinateur par exemple) si vous aviez réalisé toutes ces actions dans l'ordre indiqué.

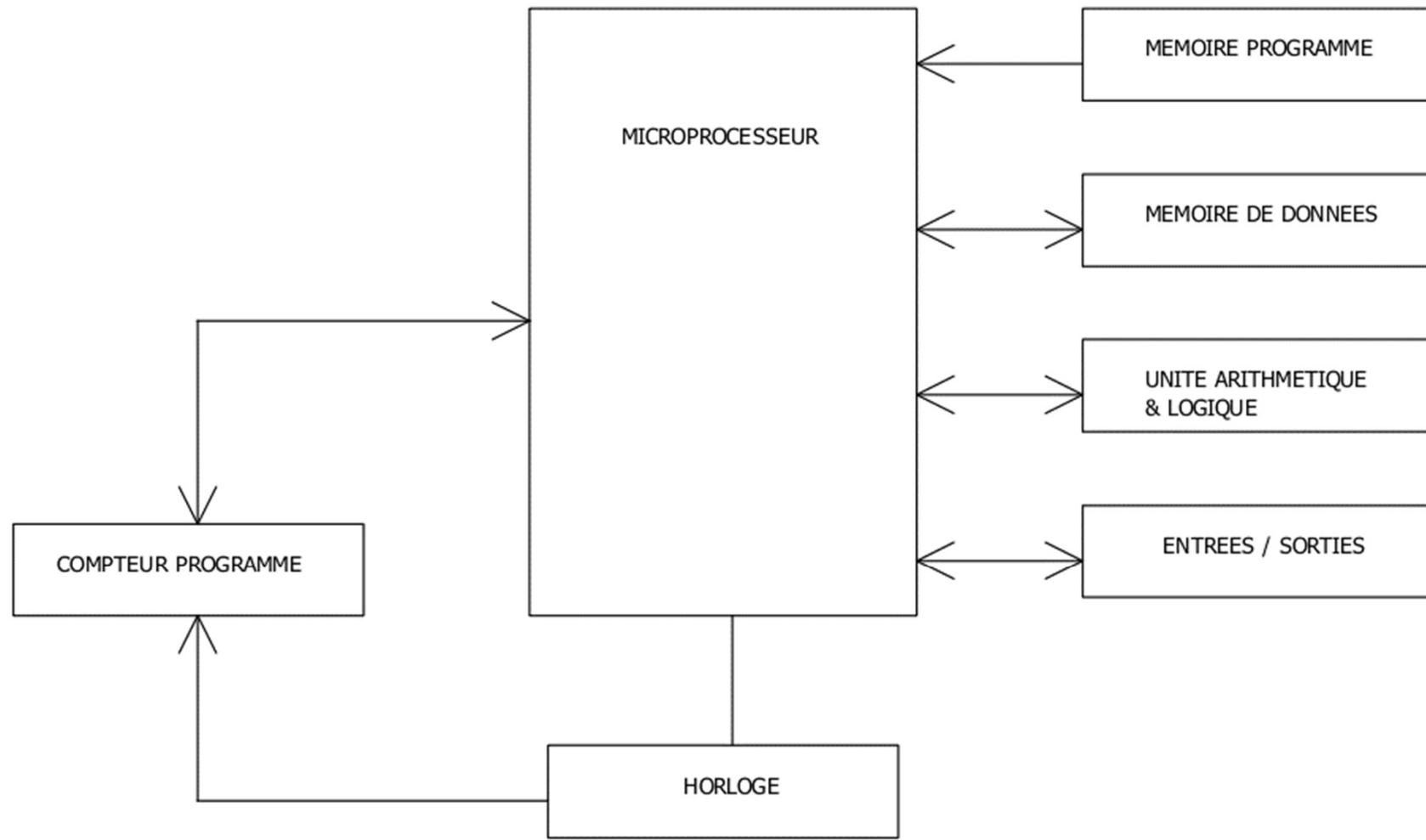
ASSEMBLEUR – Préambule architecture d'un Système

Analyse d'une opération de relève de compteur EDF :

1. Vous venez d'exécuter **un programme** qui se compose **d'une suite d'instructions** impératives. A souhait, vous pourriez réaliser plusieurs fois ce même programme inscrit dans le courrier d'EDF. Ce support représente **une mémoire programme** qui sera **non volatile et difficilement effaçable** (sauf si vous le perdez ou autre...). Maintenant, de quoi avez-vous besoin pour réaliser ce programme ?
2. **D'un bloc note pour ne pas oublier les données** à collecter et les résultats de calculs. S'il est question d'oubli, il s'agit d'une mémoire. Elle sera **nommée mémoire de données**. Elle pourra être volatile c'est-à-dire effaçable facilement une fois que les données ont été utilisées.
3. Ces données, vous les avez collectées sur le compteur, sur la quittance, sur le courrier d'EDF, envoi à l'extérieur, etc...nous nommerons **entrées/sorties** de données.
4. La soustraction est réalisée à l'aide d'une calculatrice. Pour rappel, **c'est l'U.A.L.**
5. Et vous, quel est votre rôle ? Vous êtes considéré comme **le microprocesseur**. Vous avez réalisé les instructions, vous avez calculé, vous avez lu et écrit dans la mémoire, vous avez sauvegardé...
6. Enfin, il reste la notion du temps de réalisation de la procédure de 1 à 8. C'est **le cycle machine**. Pour se souvenir quelle est l'instruction en cours, on pointera (ici de 0 à 8) avec ce qu'on appelle le **compteur de programme**.

ASSEMBLEUR – Préambule architecture d'un Système

Analyse d'une opération de relève de compteur EDF (Suite) :

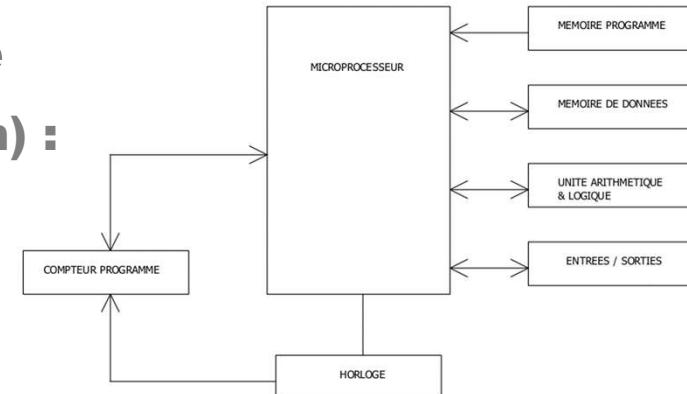


ASSEMBLEUR – Préambule architecture d'un Système

Analyse d'une opération de relève de compteur EDF (Fin) :

Rappel des fonctions de base:

- ❖ **Horloge:** génère les temps de base de gestion du système;
- ❖ **Compteur programme :** pointe l'étape du programme en cours;
- ❖ **Mémoire programme :** contient de façon ordonnée les tâches à exécuter;
- ❖ **Unité Arithmétique et Logique :** effectue les opérations (+,-,/,* , ET, OU, NON, COMP, etc...);
- ❖ **Processeur :** ordonne les transits d'information après lecture et décodage des instructions;
- ❖ **Entrées / Sorties:** permettent le dialogue avec l'extérieur du système;
- ❖ **Mémoire de données:** accumule les données fugitives.
- ❖ **BUS:** moyens importants de transmission des données.



ASSEMBLEUR – Introduction

Rappel

La machine ne connaît que langage binaire. Mais ce langage est long et fastidieux. Du coup, une codification par 4 bits mais représentés par un seul signe (chiffre puis lettre).

Ce système de codage est appelé hexadécimal.

Notions de langage

La mise en place d'un traitement informatique se déroule en **3 phases**:

❖ **Recherche d'un algorithme:** c'est la recherche d'une méthode de traitement. Un algorithme est une description formelle de la méthode à utiliser pour parvenir à résoudre un problème donné.

Exemple d'algorithme, calcul de votre consommation électrique :

Lire le compteur;

Lire le dernier relevé;

Effectuer l'opération : $\text{consommation} = \text{compteur} - \text{dernier relevé}$;

Ranger le résultat dans la mémoire.

ASSEMBLEUR – Introduction

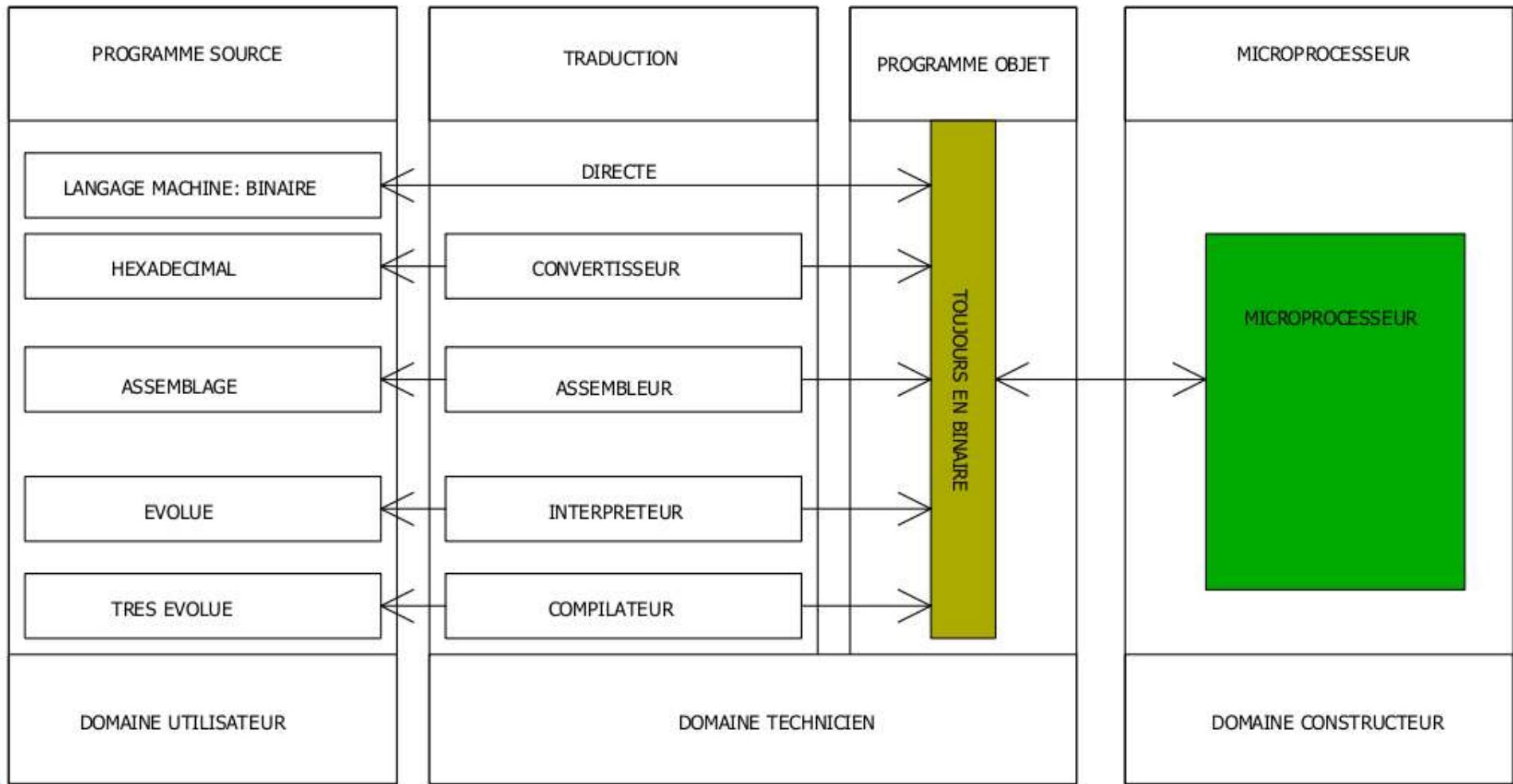
Notions de langage (Suite)

La mise en place d'un traitement informatique se déroule en 3 phases:

- ❖ **Recherche d'un algorithme:** ...
- ❖ **Ecriture d'un programme source:** c'est la traduction de l'algorithme sous forme d'instructions. Le langage utilisé peut être le langage machine, le langage assembleur et le langage évolué ou très évolué. Donc, un programme source est un programme écrit en langage symbolique.
- ❖ **Traduction du programme source en programme objet:** un programme objet est un programme écrit en langage machine. **Le passage de source -> objet se fait suivant les cas par assemblage, interprétation ou compilation.**

ASSEMBLEUR – Introduction

Différents niveaux de langage par domaines



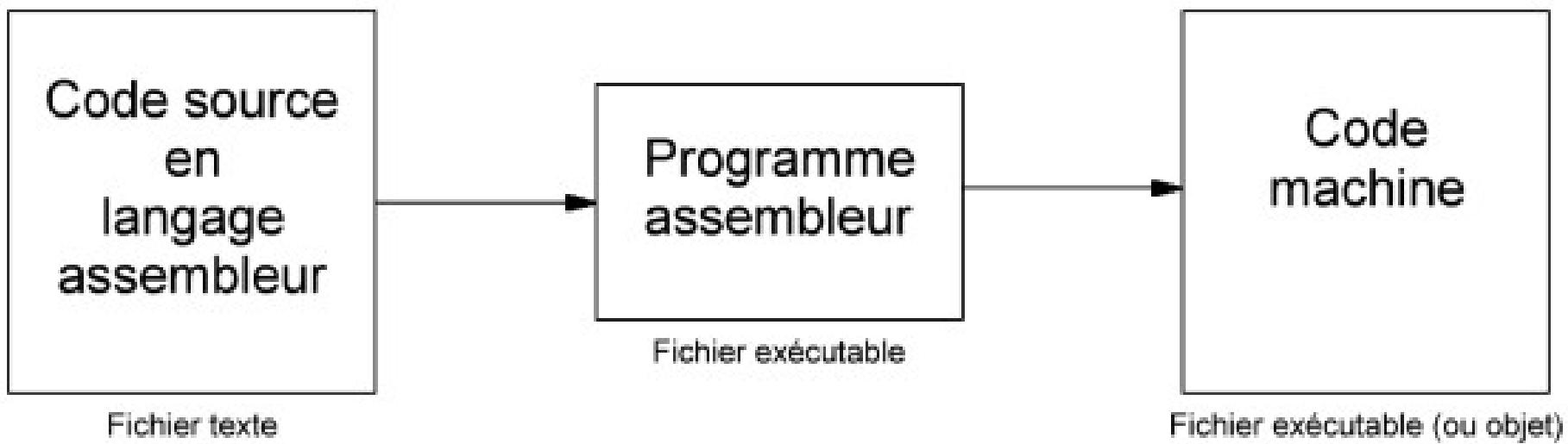
ASSEMBLEUR – Introduction

Différents niveaux de langage (Nota)

- ❖ **Un interpréteur** lira une ligne de programme, il vérifiera si elle est sans erreurs, il la traduira en binaire, et il l'exécutera, il reviendra ensuite lire la ligne suivante.
- ❖ **Un compilateur** vérifiera si tout le programme est sans erreurs, il traduira tout le programme en binaire, il rangera en mémoire où il pourra alors être exécuté en entier et en binaire d'où un gain de temps considérable à l'exécution.

ASSEMBLEUR – Le programme assembleur généralités

- ❖ Programme informatique qui traduit du code assembleur en code machine. Ce processus s'appelle l'assemblage. Il existe plusieurs programmes assembleurs différents pour un même langage assembleur (comme il existe plusieurs compilateurs C différents pour le langage C).
- ❖ Une fois l'assemblage terminé, le code machine généré peut être chargé en mémoire et exécuté par un microprocesseur.
- ❖ Il a été demandé d'étudier les bases de l'assembleur de la famille 68000 du fabricant Motorola. Cet assembleur sera vu en détail en deuxième année de la formation Bachelor.



ASSEMBLEUR – Référence 68000 de Motorola

Structure du code source (1/2)

Il contient un certain nombre d'instructions. Voir l'exemple ci-après qui permet d'identifier les différentes parties d'un code source.

```
ORG    $2000 ; Place le programme à l'adresse $2000.  
START MOVE.B D0,D1 ; D0 → D1.  
        EXT.L  D0 ; Extension de signe.  
LOOP   SUBI.L #1,D0 ; D0 - 1 → D0.  
        BNE    LOOP  ; Saut à LOOP tant que D0 n'est pas nul.  
        RTS   ; Retour de sous-programme.
```

Un code source contient une instruction par ligne. Une instruction est divisée en quatre champs distincts :

Étiquette	Mnémonique	Opérande	Commentaire
START	ORG	\$2000	; Place le programme à l'adresse \$2000.
	MOVE.B	D0,D1	; D0 → D1.
	EXT.L	D0	; Extension de signe.
	SUBI.L	#1,D0	; D0 - 1 → D0.
	BNE	LOOP	; Saut à LOOP tant que D0 n'est pas nul.
	RTS		; Retour de sous-programme.

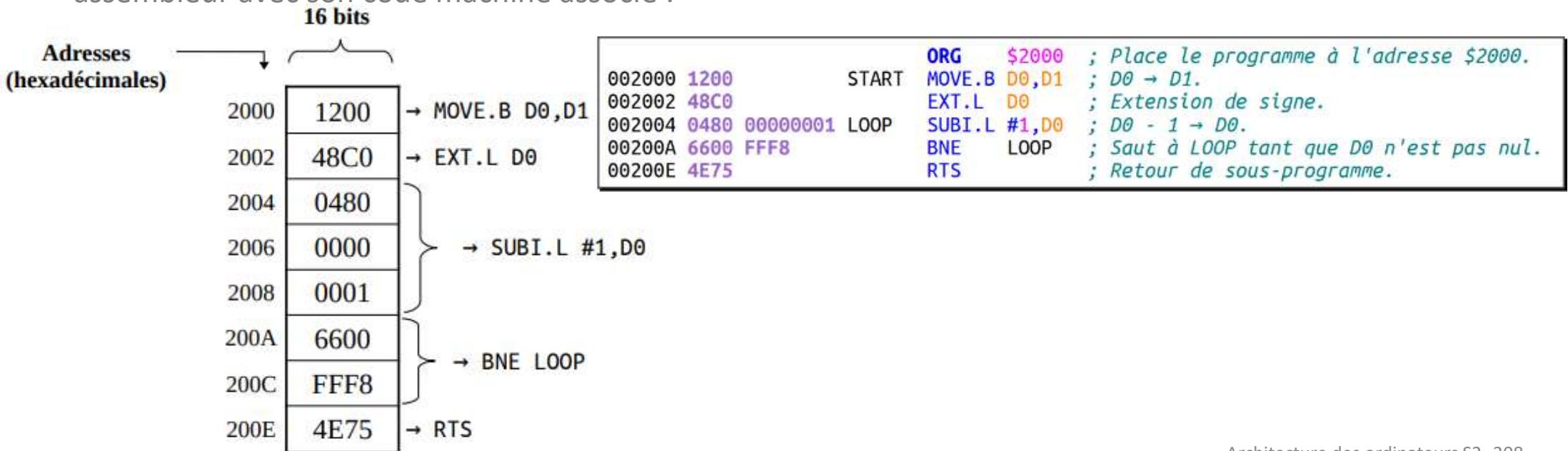
ASSEMBLEUR – Référence 68000 de Motorola

Structure du code source (2/2)

Une fois assemblé et chargé en mémoire, les instructions se retrouvent en mémoire sous la forme de mots de 16 bits (voir le plan de la mémoire ci-dessous). On constate que certaines instructions prennent plus de place en mémoire que d'autres.

Pour le 68000 : **la taille minimale d'une instruction en mémoire est d'un mot de 16 bits (2 octets) ;**
la taille maximale d'une instruction en mémoire est de cinq mots de 16 bits (10 octets).

La première ligne (ORG \$2000) n'a pas été convertie en langage machine. Représentation du code assembleur avec son code machine associé :



ASSEMBLEUR – Référence 68000 de Motorola

Le champ étiquette

C'est le premier champ sur une ligne d'instruction en langage assembleur. Il est facultatif.

Si la ligne contient une étiquette, **elle doit débuter sur le premier caractère de la ligne**. Si la ligne ne contient pas d'étiquette, **le premier caractère de la ligne doit être un espace ou une tabulation**.

L'assembleur attribue à une étiquette la valeur de l'adresse à laquelle sera placée l'instruction dans le code source. Les étiquettes sont utilisées pour les instructions de branchement.

Dans l'exemple ci-après :

- ❖ la valeur de l'étiquette START sera à l'adresse 2000_{16} ;
- ❖ la valeur de l'étiquette LOOP sera à l'adresse 2004_{16} .

002000 1200	START	ORG \$2000 ; Place le programme à l'adresse \$2000.
002002 48C0		MOVE.B D0,D1 ; D0 → D1.
002004 0480 00000001	LOOP	EXT.L D0 ; Extension de signe.
00200A 6600 FFF8		SUBI.L #1,D0 ; D0 - 1 → D0.
00200E 4E75		BNE LOOP ; Saut à LOOP tant que D0 n'est pas nul.
		RTS ; Retour de sous-programme.

Il est possible d'employer ces étiquettes comme opérandes dans le reste du programme. L'assembleur remplacera l'étiquette par la valeur affectée.

Le codeur n'a donc pas à se soucier de la valeur des adresses lors de la phase de développement.

ASSEMBLEUR – Référence 68000 de Motorola

Le champ mnémonique (1/2)

Ce champ contient le nom d'une instruction ou d'une directive d'assemblage.

Une instruction peut être traduite en langage machine. Elle appartient au jeu d'instructions d'un microprocesseur.

Une directive d'assemblage n'est pas une instruction. Elle ne fait pas partie du jeu d'instructions d'un microprocesseur. Elle ne peut pas être traduite en langage machine.

Une directive appartient au programme assembleur et non pas au langage assembleur. Elle permet, comme son nom l'indique, de donner une directive au programme assembleur au moment de l'assemblage.

Voilà pourquoi la première ligne de notre programme n'a pas été traduite en langage machine.

002000	1200	START	ORG	\$2000 ; Place le programme à l'adresse \$2000.
002002	48C0		MOVE.B	D0,D1 ; D0 → D1.
002004	0480	00000001	EXT.L	D0 ; Extension de signe.
00200A	6600	FFF8	SUBI.L	#1,D0 ; D0 - 1 → D0.
00200E	4E75		BNE	LOOP ; Saut à LOOP tant que D0 n'est pas nul.
			RTS	; Retour de sous-programme.

ASSEMBLEUR – Référence 68000 de Motorola

Le champ mnémonique (2/2)

Le ORG n'est pas une instruction, mais une directive d'assemblage. Cette directive indique à l'assembleur l'adresse mémoire à laquelle devront être placées les instructions lors de leur chargement en mémoire (ici l'adresse 2000_{16}).

Certaines mnémoniques du 68000 acceptent une extension. Cette extension sert à préciser la taille de travail d'une instruction ou d'une directive d'assemblage.

Les trois extensions principales sont :

- ❖ l'extension .B (Byte) pour l'octet (8 bits) ;
- ❖ l'extension .W (Word) pour le mot (16 bits) ;
- ❖ l'extension .L (Long) pour le mot long (32 bits).

L'extension .S (Short) peut également être acceptée par certaines instructions de branchement.

Elle est alors équivalente à l'extension .B (Byte).

002000	1200	START	ORG	\$2000 ; Place le programme à l'adresse \$2000.
002002	48C0		MOVE.B	D0,D1 ; D0 → D1.
002004	0480 00000001	LOOP	EXT.L	D0 ; Extension de signe.
00200A	6600 FFF8		SUBI.L	#1,D0 ; D0 - 1 → D0.
00200E	4E75		BNE	LOOP ; Saut à LOOP tant que D0 n'est pas nul.
			RTS	; Retour de sous-programme.

ASSEMBLEUR – Référence 68000 de Motorola

Le champ opérande

Certaines instructions ou directives d'assemblage manipulent des données. Les opérandes indiquent à une instruction où elle peut trouver les données dont elle a besoin.

Une instruction 68000 peut avoir zéro, un ou deux opérandes (jamais plus). Dans le cas de deux opérandes, l'opérande de gauche est l'opérande source, l'opérande de droite est l'opérande destination.

Par exemple, dans l'instruction suivante : **MOVE.B D0,D1**

- ❖ **D0 est l'opérande source (il ne sera pas modifié par l'instruction) ;**
- ❖ **D1 est l'opérande destination (il sera modifié par l'instruction).**

Certains opérandes peuvent être des nombres.

L'assembleur 68000 accepte une représentation des nombres dans les bases 2, 16 et 10.

- ❖ **Un nombre en base 2 se représente à l'aide du préfixe % (par exemple : %10001001) ;**
- ❖ **Un nombre en base 16 se représente à l'aide du préfixe \$ (par exemple : \$2EF8) ;**
- ❖ **Un nombre en base 10 ne prend aucun préfixe.**

ASSEMBLEUR – Référence 68000 de Motorola

Le champ commentaire

Ce champ permet d'introduire des commentaires dans un programme.

Un commentaire débute par un point-virgule. Tout ce qui suit ce point-virgule jusqu'à la fin de la ligne ne sera pas pris en compte par l'assembleur lors de la phase d'assemblage.

Les commentaires n'ont aucun effet sur le code machine généré, mais ils procurent une aide fondamentale à la bonne compréhension d'un code source. De bons commentaires constituent une part essentielle de l'écriture de programmes en langage assembleur.

Remarque :

Un commentaire peut se placer n'importe où sur une ligne, y compris en début de ligne à la place d'une étiquette.

002000 1200	START	ORG \$2000
002002 48C0		MOVE.B D0,D1
002004 0480 00000001	LOOP	EXT.L D0
00200A 6600 FFF8		SUBI.L #1,D0
00200E 4E75		BNE LOOP
		RTS

*; Place le programme à l'adresse \$2000.
; D0 → D1.
; Extension de signe.
; D0 - 1 → D0.
; Saut à LOOP tant que D0 n'est pas nul.
; Retour de sous-programme.*

ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Les bus d'adresse et de données

Le 68000 possède un bus d'adresse de 23 fils et un bus de donnée de 16 fils. Il est donc capable d'adresser 8 Mi mots de 16 bits.

Toutefois, le 68000 possède quelques signaux sur son bus de commande (notamment UDS et LDS). Ceux-ci permettent d'accéder à la mémoire par mot de 8 bits et de se comporter comme si son bus d'adresse était de 24 fils.

Pour la suite du cours, nous considérerons donc que le 68000 possède 24 fils d'adresse et qu'il est capable d'adresser **16 Mi mots de 8 bits, c'est-à-dire 16 Mio.**

Même si le 68000 est capable d'accéder à la mémoire octet par octet, son espace mémoire est physiquement organisé en mots de 16 bits. Cette organisation implique, entre autres, les caractéristiques suivantes :

- ❖ Une instruction est codée au minimum sur un mot de 16 bits ;
- ❖ Une instruction se trouve toujours à une adresse paire ;
- ❖ La lecture ou l'écriture d'un octet est possible à partir d'une adresse paire ou impaire ;
- ❖ La lecture ou l'écriture d'un mot de 16 bits est possible uniquement à partir d'une adresse paire ;
- ❖ La lecture ou l'écriture d'un mot de 32 bits est possible uniquement à partir d'une adresse paire.

ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Modes de fonctionnement

Le 68000 possède deux modes de fonctionnement : le mode superviseur et le mode utilisateur.

Architecture – Le mode superviseur

Dans ce mode, le 68000 ne fait l'objet d'aucune limitation. Il a la possibilité d'exécuter n'importe quelles instructions appartenant à son jeu d'instructions.

Ce mode de fonctionnement est principalement utilisé par les systèmes d'exploitation. Ces derniers nécessitent de posséder un contrôle total sur la machine.

Architecture – Le mode utilisateur

Dans ce mode, le 68000 fait l'objet de quelques limitations. Il ne lui est pas permis d'exécuter un certain nombre d'instructions. Ces instructions sont dites privilégiées et ne peuvent être exécutées qu'en mode superviseur.

Ce mode de fonctionnement est principalement utilisé par les applications s'exécutant sur un système d'exploitation donné.

Ces applications doivent avoir des droits limités afin de réduire les risques de corruption du système lorsqu'une application présente des dysfonctionnements.

ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Les registres (Rappel)

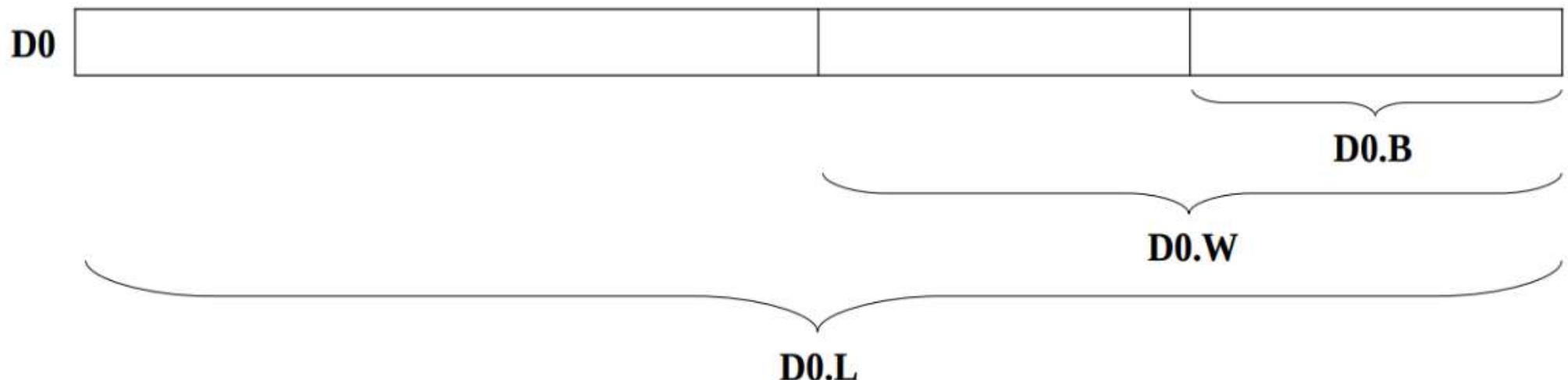
Un registre est un espace de stockage temporaire intégré dans un microprocesseur.

Architecture – Les bus de données

Le 68000 possède huit registres de donnée **d'une taille de 32 bits : D0, D1, D2, D3, D4, D5, D6 et D7.**

Ces registres n'ont aucune fonction prédéfinie et permettent de manipuler tout type de donnée. Ils sont accessibles sur 8, 16 et 32 bits.

Exemple du registre D0 :



ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Les registres d'adresse et les pointeurs de pile (1/2)

Le 68000 possède huit registres d'adresse d'une taille de 32 bits : A0, A1, A2, A3, A4, A5, A6 et A7.

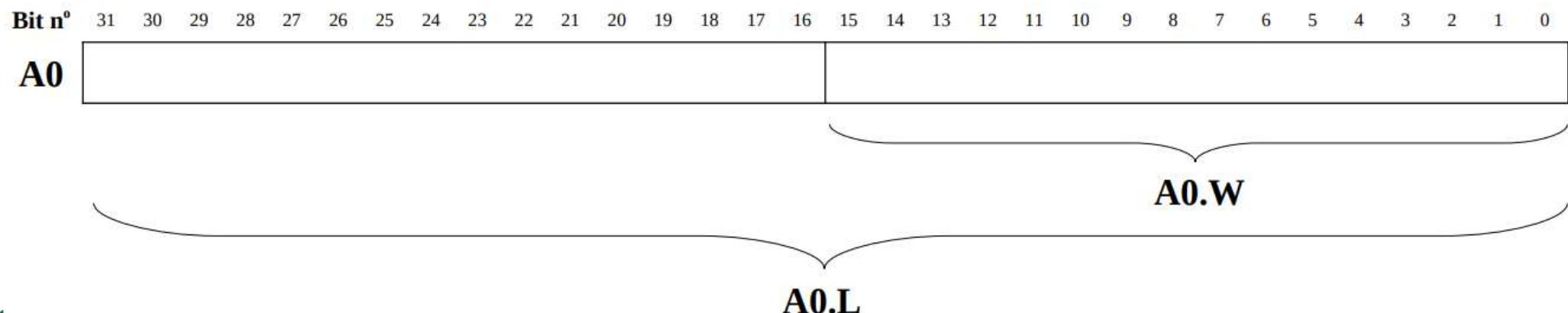
Ces registres servent à contenir des adresses. Ils sont accessibles sur 16 et 32 bits.

Le 68000 possédant 24 lignes d'adresse, les bits 24 à 31 des registres d'adresse sont ignorés. La valeur de ces bits ne sera jamais positionnée sur le bus d'adresse.

Par exemple, que l'on ait $A0 = \$679809AC$, ou bien $A0 = \$F59809AC$, ou encore $A0 = \$009809AC$, l'adresse à laquelle on accédera à partir de ce registre sera l'adresse $\$9809AC$.

Pour la suite, nous laisserons toujours les huit bits de poids fort des registres d'adresse à zéro.

Exemple du registre A0 :



ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Les registres d'adresse et les pointeurs de pile (2/2)

Le 68000 possède deux pointeurs de pile d'une taille de 32 bits :

- ❖ SSP (Supervisor Stack Pointer) qui est le pointeur de pile du mode superviseur ;
- ❖ USP (User Stack Pointer) qui est le pointeur de pile du mode utilisateur.

Le registre d'adresse A7 tient un rôle particulier dans le 68000 puisque c'est à travers lui que se fait l'accès à ces deux pointeurs de pile :

- ❖ Lorsque le 68000 se trouve en mode superviseur, le registre A7 est en fait le registre SSP ;
- ❖ Lorsque le 68000 se trouve en mode utilisateur, le registre A7 est en fait le registre USP.

Ainsi, si l'on écrit en mode superviseur dans A7, et que l'on passe ensuite en mode utilisateur, on lira autre chose dans A7 que ce que l'on vient d'écrire.

Cette duplication de registre entre le mode superviseur et le mode utilisateur permet de gérer des piles distinctes pour chacun de ces modes.

ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Le compteur programme

Le compteur programme PC (Program Counter) est un registre d'une taille de 32 bits qui contient l'adresse de la prochaine instruction à exécuter. Ce registre est modifié automatiquement par le 68000.

Tout comme les registres d'adresse, ses huit bits de poids fort sont ignorés et seuls ses 24 bits de poids faible sont positionnés sur le bus d'adresse.

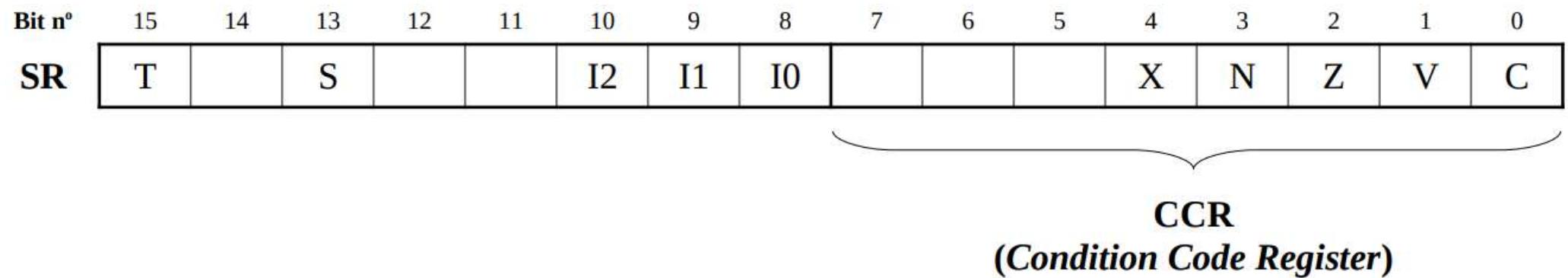
Architecture – Le registre d'état (1/4)

Le registre d'état SR (Status Register) est un registre d'une taille de 16 bits qui contient l'état du microprocesseur.

L'intégralité des 16 bits du registre SR n'est accessible qu'en mode superviseur. Toutefois, le registre CCR (Condition Code Register) qui représente les 8 bits de poids faible du registre SR est, quant à lui, accessible en mode utilisateur. Il est donc possible d'accéder aux 8 bits de poids faible du registre SR en passant par le registre CCR.

ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Le registre d'état (2/4)



Les différents bits constituant le registre d'état sont appelés drapeaux (ou flags en anglais). Ces drapeaux contiennent des informations complémentaires sur le résultat d'une opération ou sur l'état du 68000.

ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Le registre d'état (3/4)

Le comportement général des flags est donné ci-dessous :

- ❖ C (Carry) : Retenue ou dépassement non signé ($C = 1$ si retenue, $C = 0$ si pas de retenue) ;
- ❖ V (Overflow) : Dépassement signé ($V = 1$ si dépassement, $V = 0$ si pas de dépassement) ;
- ❖ Z (Zero) : Zéro ($Z = 1$ si nul, $Z = 0$ si non nul) ;
- ❖ N (Negative) : Négatif ($N = 1$ si négatif, $N = 0$ si positif ou nul) ;
- ❖ X (Extend) : Extension (identique au flag C pour la plupart des instructions) ;
- ❖ I0, I1, I2 (Interrupt Priority Mask) : Masque des priorités d'interruption ;
- ❖ S (Supervisor State) : Indicateur de mode ($S = 0$ si mode utilisateur, $S = 1$ si mode superviseur) ;
- ❖ T (Trace Mode) : Activation du mode trace (nous n'utiliserons pas le mode trace dans ce cours).

Toutefois, la gestion de certains flags peut légèrement varier d'une instruction à l'autre.

Il est donc indispensable de consulter la documentation technique du 68000 afin de connaître précisément comment une instruction modifie les flags.

ASSEMBLEUR – Référence 68000 de Motorola

Architecture – Le registre d'état (4/4)

Voici quelques exemples du positionnement des flags C, V, Z et N pour les instructions d'additions :

- ❖ Addition sur 8 bits (.B) -> \$7A + \$86 = \$100 (le résultat sur 8 bits est \$00)

C = 1, V = 0, Z = 1, N = 0

- ❖ Addition sur 16 bits (.W) -> \$9F00 + \$8E00 = \$12D00 (le résultat sur 16 bits est \$2D00)

C = 1, V = 1, Z = 0, N = 0

- ❖ Addition sur 32 bits (.L) -> \$70000000 + \$10000000 = \$80000000

C = 0, V = 1, Z = 0, N = 1

- ❖ Le flag N est le bit de signe du résultat. Il prend donc la valeur du bit de poids fort du résultat.

- ❖ Le flag Z est à 1 si le résultat est nul.

- ❖ Le flag C est à 1 s'il y a une retenue.

- ❖ Le flag V se détermine en faisant la supposition que les nombres à additionner sont signés. Il est à 1

uniquement si l'une des deux conditions suivantes est vraie :

- ❖ On additionne deux nombres positifs et le résultat est négatif.

- ❖ On additionne deux nombres négatifs et le résultat est positif.

ASSEMBLEUR – Référence 68000 de Motorola

Principales directives d'assemblage – Directive ORG

La directive ORG (Origin) sert à préciser à l'assembleur l'adresse à partir de laquelle seront assemblées les instructions en mémoire.

Un programme peut posséder plusieurs ORG (il faut faire attention au recouvrement).

```
ORG      $1000
; Ces instructions seront assemblées
; à partir de l'adresse $1000.
LEA      $2000,A0
CLR.B    D1
JSR      $5000

ORG      $5000
; Ces instructions seront assemblées
; à partir de l'adresse $5000.
MOVE.B   (A0)+,D0
ADDQ.B   #1,D1
RTS
```

ASSEMBLEUR – Référence 68000 de Motorola

Principales directives d'assemblage – Directive EQU

La directive EQU (Equate) permet d'attribuer explicitement une valeur à une étiquette. L'assembleur remplacera ensuite l'étiquette par la valeur qui lui a été attribuée.

```
START    EQU      $1000
PRINT    EQU      $5000
COUNT1   EQU      200
COUNT2   EQU      850

        ORG      START          ; ORG      $1000
        MOVE.L   #COUNT1,D0     ; MOVE.L   #200,D0
        JSR      PRINT          ; JSR      $5000
        MOVE.L   #COUNT2,D0     ; MOVE.L   #850,D0
        JSR      PRINT          ; JSR      $5000
```

ASSEMBLEUR – Référence 68000 de Motorola

Principales directives d'assemblage – Directive DC

La directive DC (Define Constant) permet de placer des données en mémoire. Ces données peuvent être des nombres en représentation décimale, hexadécimale ou binaire, mais aussi des chaînes de caractères. Dans ce dernier cas, ce sont les codes ASCII des caractères qui sont placés en mémoire.

```
ORG      $1000  
  
DC.B    10,5,7,$7a,255,%11111001  
DC.B    "Hello World",13,10,0  
DC.W    5,6  
DC.L    5,6
```

Le contenu de la mémoire à partir de l'adresse \$1000 sera donc le suivant :

1000 0A 05 07 7A FF F9	DC.B 10,5,7,\$7a,255,%11111001
1006 48 65 6C 6C 6F 20 57 6F 72 6C 64 0D 0A 00	DC.B "Hello World",13,10,0
1014 00 05 00 06	DC.W 5,6
1018 00 00 00 05 00 00 00 06	DC.L 5,6

ASSEMBLEUR – Référence 68000 de Motorola

Principales directives d'assemblage – Directive DS

La directive DS (Define Storage) permet de résERVER un espace de stockage qui pourra être utilisé lors de l'exécution d'un programme.

Cela permet d'éviter au programme d'écrire dans des emplacements qui ne lui seraient pas réservés et d'effacer des données ou du code qui ne doivent pas l'être.

	ORG	\$5000
TAB1	DS.B	4 ; Réserve 4 octets en mémoire (4 x 8 bits) ; TAB1 = \$5000
TAB2	DS.W	3 ; Réserve 3 mots en mémoire (3 x 16 bits) ; TAB2 = \$5004
TAB3	DS.L	1 ; Réserve 1 mot long en mémoire (1 x 32 bits) ; TAB3 = \$500A
NEXT		; NEXT = \$500E

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage

Un mode d'adressage permet de préciser à une instruction où se trouvent les opérandes dont elle a besoin.

À titre d'exemple, nous utiliserons l'instruction MOVE afin d'illustrer certains modes d'adressage.

Cette

instruction permet de copier le contenu d'un opérande source vers un opérande destination.

La taille de l'instruction peut être de 8, 16 ou 32 bits :

```
MOVE.B  source,destination ; source → destination (uniquement sur 8 bits)
MOVE.W  source,destination ; source → destination (uniquement sur 16 bits)
MOVE.L  source,destination ; source → destination (sur 32 bits)
```

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage – L'adressage effective

L'adresse effective (effective address) est l'emplacement d'un opérande. Attention ! une adresse effective n'est pas nécessairement une adresse mémoire ! Un opérande peut se trouver dans un registre ou dans la mémoire. Nous distinguerons deux types de modes d'adressage :

- ❖ Les modes d'adressage qui ne spécifient pas d'emplacement mémoire (l'adresse effective n'est pas une adresse mémoire). Il s'agit des modes d'adressage directs (l'adresse effective est un registre) et du mode d'adressage immédiat (l'adresse effective n'existe pas).
- ❖ Les modes d'adressage qui spécifient un emplacement mémoire (l'adresse effective est une adresse mémoire). Il s'agit des modes d'adressage indirects et des modes d'adressage absolus.

Dans la documentation technique du 68000, l'adresse effective est souvent notée `<ea>`. Nous utiliserons la même notation dans la suite de ce cours.

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage – Pas de spécification d'emplacement en mémoire.

L'opérande spécifié par ces modes d'adressage se trouve soit directement dans un registre, soit immédiatement dans l'instruction. Il n'est pas utile de déterminer une adresse mémoire spécifique.

Direct par registre de donnée: Dn

L'adresse effective est un registre de donnée. L'opérande est lu ou écrit directement dans un registre de donnée. La taille de l'opérande dépend de la taille de l'instruction (8, 16 ou 32 bits).

Valeurs initiales : D0 = \$11223344 et D1 = \$AABBCCDD

```
MOVE.B D0,D1 ; D0.B → D1.B ; D1 = $AABBCC44 (copie sur 8 bits)
MOVE.W D0,D1 ; D0.W → D1.W ; D1 = $AABB3344 (copie sur 16 bits)
MOVE.L D0,D1 ; D0.L → D1.L ; D1 = $11223344 (copie sur 32 bits)
```

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage – Pas de spécification d'emplacement en mémoire.

Direct par registre de donnée: An

L'adresse effective est un registre d'adresse. L'opérande est lu ou écrit directement dans un registre d'adresse. La taille de l'opérande dépend de la taille de l'instruction (16 ou 32 bits).

Valeurs initiales : D0 = \$11223344 et A0 = \$005030B0

```
MOVE.W  A0,D0 ; A0.W → D0.W ; D0 = $112230B0
MOVE.L  A0,D0 ; A0.L → D0.L ; D0 = $005030B0
```

Remarque :

L'instruction MOVE n'accepte pas de registre d'adresse comme opérande destination. Il faut utiliser les instructions MOVEA ou LEA pour assigner une valeur à un registre d'adresse.

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage – Pas de spécification d'emplacement en mémoire.

Immédiat : #<data>

L'opérande est contenu dans le code machine de l'instruction. Il s'agit d'une donnée immédiate qui est précisée dans le programme assembleur. Cette donnée doit être précédée du caractère '#'.

Dans ce mode d'adressage, l'adresse effective n'existe pas, car la donnée est immédiatement traitée par l'instruction.

Valeur initiale : D0 = \$11223344

```
MOVE.B #$FF,D0 ; D0 = $112233FF (La donnée sur 8 bits est copiée dans D0.B)
MOVE.W #$7A8,D0 ; D0 = $112207A8 (La donnée sur 16 bits est copiée dans D0.W)
MOVE.L #$7A8,D0 ; D0 = $000007A8 (La donnée sur 32 bits est copiée dans D0.L)
```

Avec $255_{10} = FF_{16}$ et $1960_{10} = 7A8_{16}$

- ❖ Une donnée immédiate ne peut jamais se retrouver en opérande destination. Il n'est pas possible d'écrire dans une donnée immédiate.

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

L'opérande spécifié par ces modes d'adressage se trouve en mémoire. Il est donc nécessaire de déterminer

une adresse effective qui sera l'adresse mémoire à laquelle se trouve l'opérande.

Nous utiliserons les valeurs initiales ci-dessous pour les modes d'adressage qui nécessitent des exemples.

Nous utiliserons les valeurs initiales ci-dessous pour les modes d'adressage qui nécessitent des exemples.

La mémoire et les registres seront réinitialisés pour chaque mode d'adressage.

Valeurs initiales : D0 = \$11223344 A0 = \$00001000 PC = \$00002000

D1 = \$AABBCCDD A1 = \$00001008

D2 = \$0000FFFF A2 = \$00001010

D3 = \$00000003 A3 = \$00000006

\$001000 21 45 87 AF B5 F3 3C 32

\$001008 AD 45 39 98 9A 9B 9C 9D

\$001010 03 69 01 00 12 0A 0D C9

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect par registre d'adresse : (An)

L'adresse effective est celle du registre d'adresse : <ea> = An

MOVE.B	(A0),D0	; (\$1000) → D0.B ; D0 = \$11223321
MOVE.W	(A0),D0	; (\$1000) → D0.W ; D0 = \$11222145
MOVE.L	(A0),D0	; (\$1000) → D0.L ; D0 = \$214587AF

MOVE.B	D1,(A0)	; D1.B → (\$1000) ; \$1000 DD 45 87 AF B5 F3 3C 32
MOVE.W	D1,(A0)	; D1.W → (\$1000) ; \$1000 CC DD 87 AF B5 F3 3C 32
MOVE.L	D1,(A0)	; D1.L → (\$1000) ; \$1000 AA BB CC DD B5 F3 3C 32

MOVE.B	(A1),(A2)	; (\$1008) → (\$1010) ; \$1010 AD 69 01 00 12 0A 0D C9
MOVE.W	(A1),(A2)	; (\$1008) → (\$1010) ; \$1010 AD 45 01 00 12 0A 0D C9
MOVE.L	(A1),(A2)	; (\$1008) → (\$1010) ; \$1010 AD 45 39 98 12 0A 0D C9

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect par registre d'adresse avec postincrémentation : (An)+

L'adresse effective est celle du registre d'adresse : $\langle ea \rangle = An$

Une fois que le 68000 a accédé à l'opérande, le registre d'adresse est incrémenté. La valeur de l'incrémentation dépend de la taille de l'instruction :

- ❖ $An = An + 1$ si l'extension de l'instruction est .B ;
- ❖ $An = An + 2$ si l'extension de l'instruction est .W ;
- ❖ $An = An + 4$ si l'extension de l'instruction est .L.

```
MOVE.W  (A0)+,D0 ; ($1000) → D0.W ; D0 = $11222145 ; A0 = $1002  
MOVE.L  (A0)+,D0 ; ($1002) → D0.L ; D0 = $87AFB5F3 ; A0 = $1006
```

```
MOVE.B  D1,(A1)+ ; D1.B → ($1008) ; $1008 DD 45 39 98 9A 9B 9C 9D ; A1 = $1009  
MOVE.B  D1,(A1)+ ; D1.B → ($1009) ; $1009 DD DD 39 98 9A 9B 9C 9D ; A1 = $100A  
MOVE.W  D1,(A1)+ ; D1.W → ($100A) ; $1008 DD DD CC DD 9A 9B 9C 9D ; A1 = $100C  
MOVE.L  D1,(A1)+ ; D1.L → ($100C) ; $1008 DD DD CC DD AA BB CC DD ; A1 = $1010
```

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect par registre d'adresse avec pré-décrémentation : -(An)

Le registre d'adresse est décrémenté avant même que le 68000 n'accède à l'opérande. La valeur de la décrémentation dépend de la taille de l'instruction :

- ❖ An = An – 1 si l'extension de l'instruction est .B ;
- ❖ An = An – 2 si l'extension de l'instruction est .W ;
- ❖ An = An – 4 si l'extension de l'instruction est .L.

L'adresse effective est celle du registre d'adresse. **Attention ! il s'agit de la nouvelle valeur du registre qui vient d'être décrémenté : <ea> = An**

MOVE.B	- (A1), D0 ; A1 = \$1007 ; (\$1007) → D0.B ; D0 = \$11223332
MOVE.B	- (A1), D0 ; A1 = \$1006 ; (\$1006) → D0.B ; D0 = \$1122333C
MOVE.W	- (A1), D0 ; A1 = \$1004 ; (\$1004) → D0.W ; D0 = \$1122B5F3
MOVE.L	- (A1), D0 ; A1 = \$1000 ; (\$1000) → D0.L ; D0 = \$214587AF
MOVE.B	D1, - (A2) ; A2 = \$100F ; D1.B → (\$100F) ; \$1008 AD 45 39 98 9A 9B 9C DD
MOVE.B	D1, - (A2) ; A2 = \$100E ; D1.B → (\$100E) ; \$1008 AD 45 39 98 9A 9B DD DD
MOVE.W	D1, - (A2) ; A2 = \$100C ; D1.W → (\$100C) ; \$1008 AD 45 39 98 CC DD DD DD
MOVE.L	D1, - (A2) ; A2 = \$1008 ; D1.L → (\$1008) ; \$1008 AA BB CC DD CC DD DD DD

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect par registre d'adresse avec déplacement : d16(An)

L'adresse effective est la somme du registre d'adresse et d'un déplacement : $\langle ea \rangle = An + d16$. d16 est un déplacement codé sur 16 bits signés : $-32768 \leq d16 \leq +32767$.

Remarque :

Il existe deux syntaxes équivalentes pour ce mode d'adressage : d16(An) et (d16,An).

Nous utiliserons la première : d16(An).

```
MOVE.B -5(A1),D0 ; ($1003) → D0.B ; D0 = $112233AF  
MOVE.W 4(A1),D0 ; ($100C) → D0.W ; D0 = $11229A9B
```

```
MOVE.B D1,-1(A1) ; D1.B → ($1007) ; $1000 21 45 87 AF B5 F3 3C DD  
MOVE.L D1,-4(A1) ; D1.L → ($1004) ; $1000 21 45 87 AF AA BB CC DD
```

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect par registre d'adresse avec déplacement et index : d8(An,Xn)

L'adresse effective est la somme du registre d'adresse, d'un déplacement et d'un index : $\langle ea \rangle = An + d8 + Xn$

- ❖ d8 est un déplacement codé sur 8 bits signés : $-128 \leq d8 \leq +127$;
- ❖ Xn est un index codé sur 16 ou 32 bits signés. Cet index peut être un registre de donnée ou un registre d'adresse : Dn.W, Dn.L, An.W ou An.L.

Remarque :

Il existe deux syntaxes équivalentes pour ce mode d'adressage : d8(An,Xn) et (d8,An,Xn). Nous utiliserons la première : d8(An,Xn).

```
MOVE.B  2(A1,A3.W),D0 ; ($1010) → D0.B ; D0 = $11223303
MOVE.W  1(A1,D3.L),D0 ; ($100C) → D0.W ; D0 = $11229A9B

MOVE.B  D1,0(A1,D2.W) ; D1.B → ($1007) ; $1000 21 45 87 AF B5 F3 3C DD
MOVE.L  D1,-3(A1,D2.W) ; D1.L → ($1004) ; $1000 21 45 87 AF AA BB CC DD
```

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect relatif au compteur programme avec déplacement : d16(PC) – (1/2)

L'adresse effective est la somme du compteur programme et d'un déplacement : $\langle ea \rangle = PC + d16$

d16 est un déplacement codé sur 16 bits signés : $-32768 \leq d16 \leq +32767$.

Remarques :

Il existe deux syntaxes équivalentes pour ce mode d'adressage : d16(PC) et (d16,PC).

Nous utiliserons la première : d16(PC).

Pour le programmeur, la valeur du PC est difficile, voire impossible à connaître lors de la phase de développement. Or, s'il ne connaît pas la valeur du PC, il ne peut pas non plus connaître l'adresse effective.

Ce mode d'adressage serait donc inutilisable si l'on devait utiliser la syntaxe d16(PC) telle quelle. C'est pourquoi, dans un code source, on précise directement l'adresse effective sans se soucier ni de la valeur du PC, ni de la valeur du déplacement.

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect relatif au compteur programme avec déplacement : d16(PC) – (2/2)

En pratique, la syntaxe utilisée est alors la suivante : <ea>(PC)

C'est l'assembleur, au moment de la phase d'assemblage, qui s'occupe de calculer le déplacement en fonction de la valeur du PC.

Ceci simplifie énormément les choses puisque l'adresse effective apparaît directement dans le code source.

Par conséquent, le programmeur n'a pas à réaliser lui-même l'addition (PC + d16).

```
MOVE.B $1010(PC),D0 ; ($1010) → D0.B ; D0 = $11223303  
MOVE.W $100C(PC),D0 ; ($100C) → D0.W ; D0 = $11229A9B  
MOVE.L $1002(PC),D0 ; ($1002) → D0.L ; D0 = $87AFB5F3
```

; L'instruction MOVE n'accepte pas ce mode d'adressage en destination.

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Indirect relatif au compteur programme avec déplacement et index : d8(PC,Xn)

L'adresse effective est la somme du compteur programme, d'un déplacement et d'un index :

$$<ea> = PC + d8 + Xn$$

- ❖ d8 est un déplacement codé sur 8 bits signés : $-128 \leq d8 \leq +127$;
- ❖ Xn est un index codé sur 16 ou 32 bits signés. Cet index peut être un registre de donnée ou un registre d'adresse : Dn.W, Dn.L, An.W ou An.L.

Nous n'utiliserons jamais ce mode d'adressage dans la suite de ce cours.

ASSEMBLEUR – Référence 68000 de Motorola

Modes d'adressage avec spécification d'un emplacement en mémoire.

Absolu long : (xxx).L

L'adresse effective est directement spécifiée dans le code source. L'adresse est codée sur 32 bits.

```
MOVE.B $1010,D0 ; ($1010) → D0.B ; D0 = $11223303
MOVE.W $100C,D0 ; ($100C) → D0.W ; D0 = $11229A9B

MOVE.B D1,$1007 ; D1.B → ($1007) ; $1000 21 45 87 AF B5 F3 3C DD
MOVE.L D1,$1004 ; D1.L → ($1004) ; $1000 21 45 87 AF AA BB CC DD
```

Absolu court : (xxx).W

Ce mode d'adressage est équivalent au mode d'adressage absolu long, mais il ne fonctionne qu'avec des adresses que l'on peut coder sur 16 bits. L'exécution d'une instruction est alors plus rapide.

ASSEMBLEUR – Référence 68000 de Motorola

Exemples

Afin d'illustrer encore plus précisément le fonctionnement des modes d'adressage, vous trouverez ci-dessous une série d'instructions où le calcul des adresses effectives sera détaillé. Les contenus des registres (sauf le PC) et de la mémoire qui viennent d'être modifiés seront également précisés.

Pour chaque instruction, la mémoire et les registres seront réinitialisés aux valeurs ci-dessous :

Valeurs initiales : D0 = \$0000FFFF A0 = \$00001000 PC = \$00002000

D1 = \$00000004 A1 = \$00001008

D2 = \$FFFFF000 A2 = \$00001010

\$001000 21 45 87 AF B5 F3 3C 32

\$001008 AD 45 39 98 9A 9B 9C 9D

\$001010 03 69 01 00 12 0A 0D C9

ASSEMBLEUR – Référence 68000 de Motorola

Exemple - MOVE.W A1,D2

Source	Destination
A1.W	D2.W
#\$1008	

D2 = \$FFFF1008

Exemple - MOVE.W (A1),D2

Source	Destination
(A1)	D2.W
#\$AD45	

D2 = \$FFFFAD45

Exemple - MOVE.L #\$100A,D2

Source	Destination
#\$100A	D2.L
#\$0000100A	

D2 = \$0000100A

Exemple - MOVE.L \$100A,D2

Source	Destination
(\$100A)	D2.L
#\$39989A9B	

D2 = \$39989A9B

ASSEMBLEUR – Référence 68000 de Motorola

Exemple - MOVE.W #36,(A0)

Source	Destination
#36	(A0)
#\$0024	(\$1000)

\$001000 00 24 87 AF B5 F3 3C 32

Exemple - MOVE.B D1,(A1)+

Source	Destination
D1.B	(A1)
#\$04	(\$1008)

\$001008 04 45 39 98 9A 9B 9C 9D

A1 = \$00001009

Exemple - MOVE.L \$1004,-(A2)

Source	Destination
(\$1004)	(A2)
#\$B5F33C32	(\$1010 - 4) (\$100C)

\$001008 AD 45 39 98 B5 F3 3C 32

A2 = \$0000100C

ASSEMBLEUR – Référence 68000 de Motorola

Exemple - MOVE.L -(A2),-(A2)

Source	Destination
(A2)	(A2)
(\$1010 - 4)	(\$100C - 4)
(\$100C)	(\$1008)
#\$9A9B9C9D	

\$001008 9A 9B 9C 9D 9A 9B 9C 9D

A2 = \$00001008

Exemple - MOVE.B 5(A1),-1(A1,D0.W)

Source	Destination
5(A1)	-1(A1,D0.W)
(A1 + 5)	(A1 + D0.W - 1)
(\$1008 + 5)	(\$1008 - 1 - 1)
(\$100D)	(\$1006)
#\$9B	

\$001000 21 45 87 AF B5 F3 9B 32

ASSEMBLEUR – Référence 68000 de Motorola

Exemple - MOVE.L -(A2),-(A2)

Source	Destination
(A2)	(A2)
(\$1010 - 4)	(\$100C - 4)
(\$100C)	(\$1008)
#\$9A9B9C9D	

\$001008 9A 9B 9C 9D 9A 9B 9C 9D

A2 = \$00001008

Exemple - MOVE.B 5(A1),-1(A1,D0.W)

Source	Destination
5(A1)	-1(A1,D0.W)
(A1 + 5)	(A1 + D0.W - 1)
(\$1008 + 5)	(\$1008 - 1 - 1)
(\$100D)	(\$1006)
#\$9B	

\$001000 21 45 87 AF B5 F3 9B 32

ASSEMBLEUR – Référence 68000 de Motorola

Exemple - MOVE.W 2(A1,D1.L),-6(A2)

Source	Destination
2(A1,D1.L)	-6(A2)
(A1 + D1 + 2)	(A2 - 6)
(\$1008 + 4 + 2)	(\$1010 - 6)
(\$100E)	(\$100A)
#\$9C9D	

\$001008 AD 45 9C 9D 9A 9B 9C 9D

Exemple - MOVE.W \$1000(PC),\$100A

Source	Destination
(\$1000)	(\$100A)
#\$2145	

\$001008 AD 45 21 45 9A 9B 9C 9D

ASSEMBLEUR – Référence 68000 de Motorola

Les branchements – Inconditionnels

Le 68000 possède deux instructions de branchement inconditionnel :

BRA	Branchement inconditionnel
JMP	Saut inconditionnel

Ces deux instructions sont similaires. Elles possèdent quelques différences au niveau des modes d'adressage et du codage machine. Nous n'aborderons pas ces différences. Dans le cadre de ce cours d'initiation, nous considérerons que si l'opérande est une adresse (ou une étiquette), alors ces deux instructions sont **équivalentes et interchangeables**.

```
ORG      $1000
BRA      NEXT    ; Branchement inconditionnel à l'étiquette NEXT.
CLR.L    D1      ; Cette instruction ne sera pas exécutée.
NEXT
MOVE.L  #5,D0
RTS
```

On peut remplacer le BRA par un JMP :

```
ORG      $1000
JMP      NEXT    ; Saut inconditionnel à l'étiquette NEXT.
CLR.L    D1      ; Cette instruction ne sera pas exécutée.
NEXT
MOVE.L  #5,D0
RTS
```

ASSEMBLEUR – Référence 68000 de Motorola

Les branchements – Conditionnels (1/6)

Comme son nom l'indique, un branchement conditionnel comporte une condition. Ceci nous amène aux deux cas suivants :

- ❖ Soit la condition est vérifiée, auquel cas le branchement est effectué ;
- ❖ Soit la condition n'est pas vérifiée, auquel cas le branchement n'est pas effectué.

La condition d'un branchement conditionnel se fait sur un flag ou une combinaison de flags.

Les instructions de branchement conditionnel se trouvent dans le manuel à la mnémonique Bcc (Branch condition code).

Les deux c représentent la condition et doivent être remplacés par deux lettres en fonction de la condition souhaitée (par exemple : BNE, BEQ, BGE, etc.).

ASSEMBLEUR – Référence 68000 de Motorola

Les branchements – Conditionnels (2/6)

Les branchements à comparaison sur un flag

Mnémonique	Condition	Branchemen si
BPL	<i>Plus</i>	N = 0
BMI	<i>Minus</i>	N = 1
BNE	<i>Not Equal</i>	Z = 0
BEQ	<i>Equal</i>	Z = 1
BVC	<i>Overflow Clear</i>	V = 0
BVS	<i>Overflow Set</i>	V = 1
BCC	<i>Carry Clear</i>	C = 0
BCS	<i>Carry Set</i>	C = 1

L'instruction TST qui modifie les flags N et Z est très souvent utilisée avec les instructions BPL, BMI, BNE et BEQ.

```
TST.L D1
BEQ NEXT1 ; Saut si D1.L = 0 (si Z = 1)

TST.W D2
BNE NEXT2 ; Saut si D2.W ≠ 0 (si Z = 0)

TST.B D3
BMI NEXT3 ; Saut si D3.B < 0 (si N = 1)

TST.L D4
BPL NEXT4 ; Saut si D4.L ≥ 0 (si N = 0)

TST.B D5
BMI NEXT5 ; Saut si D5.B < 0 (si N = 1)
BEQ NEXT6 ; Saut si D5.B = 0 (si Z = 1)
```

ASSEMBLEUR – Référence 68000 de Motorola

Les branchements – Conditionnels (3/6)

Les branchements à comparaison non signée et à comparaison signée

Comparaison non signée		Comparaison signée		Branchement
Mnémonique	Condition	Mnémonique	Condition	
BHI	<i>Higher</i>	BGT	<i>Greater Than</i>	si supérieur
BHS	<i>Higher or Same</i>	BGE	<i>Greater or Equal</i>	si supérieur ou égal
BLO	<i>Lower</i>	BLT	<i>Less Than</i>	si inférieur
BLS	<i>Lower or Same</i>	BLE	<i>Less or Equal</i>	si inférieur ou égal

Remarques :

- L'instruction BHS est en fait un alias pour l'instruction BCC (ces deux instructions sont identiques).
- L'instruction BLO est en fait un alias pour l'instruction BCS (ces deux instructions sont identiques).

ASSEMBLEUR – Référence 68000 de Motorola

Les branchements – Conditionnels (4/6)

Les branchements à comparaison non signée et à comparaison signée

L'instruction CMP s'utilise principalement avec des branchements conditionnels selon le modèle suivant :

CMP	source,destination	
Bcc	<étiquette>	; Branchement si destination <condition> source

CMP.L	D0,D1	
BHI	NEXT	; Branchement si D1.L > D0.L (comparaison non signée)

CMP.W	D0,D1	
BGT	NEXT	; Branchement si D1.W > D0.W (comparaison signée)

CMP.B	D0,D1	
BLS	NEXT	; Branchement si D1.B ≤ D0.B (comparaison non signée)

CMP.L	D0,D1	
BLE	NEXT	; Branchement si D1.L ≤ D0.L (comparaison signée)

ASSEMBLEUR – Référence 68000 de Motorola

Les branchements – Conditionnels (5/6)

Les branchements à comparaison non signée et à comparaison signée

Exemples de boucle, la structure de boucle suivante est très utilisée en assembleur 68000 :

```
MOVE.W #n,D7      ; n → D7
                  ; (D7.W = compteur de boucle)

LOOP    ; ...
        ; ...
        ; Corps de la boucle
        ; exécuté n fois

        SUBQ.W #1,D7      ; D7.W - 1 → D7.W (si D7.W est nul, Z passe à 1)
        BNE    LOOP          ; Branchement si D7.W ≠ 0 (si Z = 0)
```

Exemples de boucle : code assembleur équivalent à la structure d'une boucle for en langage C :

```
#define MAX 17

int i;

for (i = 0; i < MAX; i++)
{
    /* Corps de la boucle */
}
```

ASSEMBLEUR – Référence 68000 de Motorola

Les branchements – Conditionnels (6/6)

Les branchements à comparaison non signée et à comparaison signée

Exemples de boucle, langage assembleur :

MAX	EQU	17	<i>; Définit la valeur de l'étiquette MAX.</i>
	CLR.L	D0	<i>; D0 → D0</i>
	MOVE.L	#MAX,D7	<i>; MAX → D7</i>
LOOP	CMP.L	D7,D0	
	BGE	DONE	<i>; Branchement à DONE si D0 ≥ D7</i>
	;	...	<i>; Corps de la boucle</i>
	;	...	<i>; (répété tant que D0 < D7)</i>
	ADDQ.L	#1,D0	<i>; D0 + 1 → D0</i>
DONE	BRA	LOOP	<i>; Branchement à LOOP</i>

ASSEMBLEUR – Référence 68000 de Motorola

L'instruction DBRA

L'instruction DBRA est une instruction optimisée pour les boucles. Elle réalise la décrémentation d'un registre sur 16 bits et effectue un branchement tant que la nouvelle valeur du registre est différente de -1.

```
MOVE.W #n,D7      ; n → D7
                  ; (D7.W = compteur de boucle)
LOOP    ; ...
                  ; ...
                  ; Corps de la boucle
                  ; exécuté n + 1 fois
DBRA    D7,LOOP    ; D7.W - 1 → D7.W ; Branchement à LOOP si D7.W ≠ -1
```

Remarque :

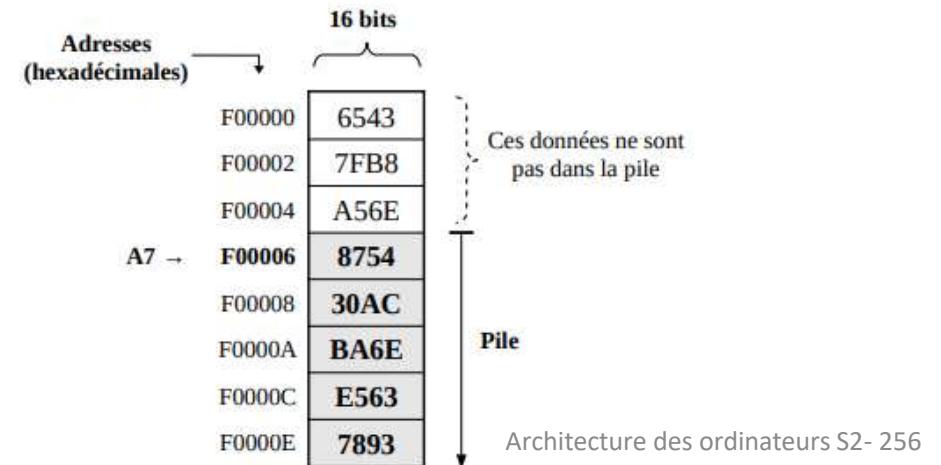
L'instruction DBRA est en fait un alias pour l'instruction DBF qui fait partie de la famille des instructions DBcc.

ASSEMBLEUR – Référence 68000 de Motorola

La pile (1/6)

- ❖ C'est une zone spéciale de la mémoire réservée à un stockage temporaire de données. Elle est de type LIFO (Last In First Out). Le dernier élément empilé sera le premier élément dépiler.
- ❖ Le registre A7 est le pointeur de pile : il pointe le sommet de la pile. Seuls des mots (16 bits) et des mots longs (32 bits) sont empilés ou dépilerés (jamais d'octet).
- ❖ L'espace mémoire réservé à la pile doit être suffisant pour satisfaire les besoins du programme en cours d'exécution. Le programmeur ou le système d'exploitation devra initialiser le pointeur de pile en conséquence.
- ❖ L'adresse en soi n'est pas significative et n'apporte rien à la bonne compréhension du fonctionnement de la pile. Ce qui compte, ce sont les variations du pointeur de pile (incrémentation, décrémentation, etc.). Nous considérerons donc que ce dernier point sur un emplacement mémoire suffisant et seules ses variations seront précisées.

Exemple pour A7 = \$F00006 :



ASSEMBLEUR – Référence 68000 de Motorola

La pile (2/6)

Généralement, deux étapes sont nécessaires pour empiler ou pour dépiler une donnée :

Étape	Empiler une donnée	Dépiler une donnée
1	Décrémenter A7	Lire la mémoire pointée par A7
2	Écrire dans la mémoire pointée par A7	Incrémenter A7

Soit le programme ci-dessous où l'on numérote de un à huit les instructions qui modifient la pile :

```
MOVE.L #$11112222,D0 ; #$11112222 → D0.L
MOVE.L #$AAAAABBBBB,D1 ; #$AAAAABBBBB → D1.L

(1) MOVE.W #$42,-(A7) ; Empile la donnée #$0042
(2) MOVE.L #$53,-(A7) ; Empile la donnée #$00000053
(3) MOVE.W D0,-(A7) ; Empile la donnée #$2222 (contenue dans D0.W)
(4) MOVE.L D1,-(A7) ; Empile la donnée #$AAAAABBBBB (contenue dans D1.L)

CLR.W D0 ; 0 → D0.W ; Le registre D0.W est modifié.
CLR.L D1 ; 0 → D1.L ; Le registre D1.L est modifié.

(5) MOVE.L (A7)+,D1 ; Dépile la donnée #$AAAAABBBBB dans D1.L
(6) MOVE.W (A7)+,D0 ; Dépile la donnée #$2222 dans D0.W
(7) MOVE.L (A7)+,D2 ; Dépile la donnée #$00000053 dans D2.L
(8) MOVE.W (A7)+,D3 ; Dépile la donnée #$0042 dans D3.W
```

ASSEMBLEUR – Référence 68000 de Motorola

La pile (3/6)

Les différents états de la pile seront les suivants :

État de la pile avant l'instruction 1	État de la pile après l'instruction 1	État de la pile après l'instruction 2	État de la pile après l'instruction 3	État de la pile après l'instruction 4
2FA8	2FA8	2FA8	2FA8	2FA8
3642	3642	3642	3642	3642
EB89	EB89	EB89	EB89	AAAA
87AF	87AF	87AF	87AF	BBBB
2118	2118	2118	2222	2222
11BC	11BC	0000	0000	0000
C948	C948	0053	0053	0053
72C8	0042	0042	0042	0042
AF3D	AF3D	AF3D	AF3D	AF3D
CF4C	CF4C	CF4C	CF4C	CF4C
7893	7893	7893	7893	7893

État de la pile après l'instruction 5	État de la pile après l'instruction 6	État de la pile après l'instruction 7	État de la pile après l'instruction 8
2FA8	2FA8	2FA8	2FA8
3642	3642	3642	3642
AAAA	AAAA	AAAA	AAAA
BBBB	BBBB	BBBB	BBBB
2222	2222	2222	2222
0000	0000	0000	0000
0053	0053	0053	0053
0042	0042	0042	0042
AF3D	AF3D	AF3D	AF3D
CF4C	CF4C	CF4C	CF4C
7893	7893	7893	7893

ASSEMBLEUR – Référence 68000 de Motorola

La pile - L'instruction MOVEM (4/6)

L'instruction MOVEM (Move Multiple Registers) est une instruction qui permet de lire ou d'écrire plusieurs registres en mémoire. Elle est très souvent utilisée pour empiler ou dépiler la valeur de plusieurs registres en une seule instruction. Ses syntaxes sont les suivantes :

- ❖ MOVEM <list>,<ea>
- ❖ MOVEM <ea>,<list>

L'opérande <list> représente une liste de registres. Seule l'instruction MOVEM est capable de manipuler une liste de registres. Les adresses effectives <ea>, quant à elles, spécifient nécessairement un emplacement mémoire (mode d'adressage indirect ou absolu).

Une liste de registres peut contenir n'importe quels registres de donnée ou d'adresse. Les registres sont séparés par le caractère « / ». Par exemple : D0/D1/A6/A4/D5.

Quand plusieurs registres de donnée ou d'adresse se suivent, il est possible d'utiliser le caractère « - » afin d'indiquer un groupe de registres. Par exemple, ces deux listes sont équivalentes :

- ❖ D0/D1/D2/D3/A2/A3/A4/A5/A6
- ❖ D0-D3/A2-A6

ASSEMBLEUR – Référence 68000 de Motorola

La pile - L'instruction MOVEM (5/6)

L'ordre dans lequel apparaissent les registres dans la liste n'a aucune importance. Par exemple, ces trois listes sont équivalentes :

- ❖ D6/A4/A2/A5/D1/A6/D2/D3
- ❖ D1/D2/D3/D6/A2/A4/A5/A6
- ❖ D1-D3/D6/A2/A4-A6

L'instruction MOVEM traite les registres dans un ordre prédéfini qui lui est propre. Plus précisément, elle possède deux ordres bien distincts :

- ❖ l'ordre classique : D0, D1, D2, D3, D4, D5, D6, D7, A0, A1, A2, A3, A4, A5, A6, A7
- ❖ l'ordre inversé : A7, A6, A5, A4, A3, A2, A1, A0, D7, D6, D5, D4, D3, D2, D1, D0

Le 68000 utilise l'ordre inversé uniquement lorsque le mode d'adressage de l'adresse effective de destination est un mode d'adressage indirect avec prédécrémation -(An). Dans tous les autres cas, c'est l'ordre classique qui est utilisé.

Par exemple :

- ❖ L'instruction MOVEM.L D0/A4,(A0) copiera d'abord le registre D0 puis le registre A4.
- ❖ L'instruction MOVEM.L D0/A4,-(A0) copiera d'abord le registre A4 puis le registre D0

ASSEMBLEUR – Référence 68000 de Motorola

La pile - L'instruction MOVEM (6/6)

L'instruction MOVEM très souvent utilisée pour empiler ou dépiler la valeur de plusieurs registres en une seule instruction.

Par exemple, l'instruction suivante :

```
MOVEM.L D1-D3/A4/A5,-(A7) ; Empile A5, puis A4, puis D3, puis D2, puis D1.
```

est équivalente aux cinq instructions ci-dessous :

MOVE.L A5,-(A7)	; Empile A5
MOVE.L A4,-(A7)	; Empile A4
MOVE.L D3,-(A7)	; Empile D3
MOVE.L D2,-(A7)	; Empile D2
MOVE.L D1,-(A7)	; Empile D1

Et l'instruction suivante :

```
MOVEM.L (A7)+,D1-D3/A4/A5 ; Dépile D1, puis D2, puis D3, puis A4, puis A5.
```

est équivalente aux cinq instructions ci-dessous :

MOVE.L (A7)+,D1	; Dépile D1
MOVE.L (A7)+,D2	; Dépile D2
MOVE.L (A7)+,D3	; Dépile D3
MOVE.L (A7)+,A4	; Dépile A4
MOVE.L (A7)+,A5	; Dépile A5

ASSEMBLEUR – Référence 68000 de Motorola

Les sous-programmes

```
Main    ...
      ...
      ; Instructions quelconques
      ...
      moveq.l #15,d1 ; 15 → D1
      moveq.l #25,d2 ; 25 → D2
      jsr    GetMin ; Appel à GetMin (15 → D0)

Next1   ...
      ...
      ; Instructions quelconques
      ...
      moveq.l #30,d1 ; 30 → D1
      moveq.l #16,d2 ; 16 → D2
      jsr    GetMin ; Appel à GetMin (16 → D0)

Next2   ...
      ...
      ; Instructions quelconques
      ...
GetMin  cmp.l  d1,d2 ; Compare D1 et D2.
        ble   d2min ; Saut à d2min si D2 ≤ D1 (comparaison signée).

d1min  move.l d1,d0 ; D1 est inférieur à D2, on copie D1 dans D0.
        rts   ; Sortie du sous-programme.

d2min  move.l d2,d0 ; D2 est inférieur ou égal à D1, on copie D2 dans D0.
        rts   ; Sortie du sous-programme.
```

ASSEMBLEUR – Référence 68000 de Motorola

Les principales instructions du 68000 - Les mouvements de donnée

source → destination

MOVE	Transfert de donnée
MOVEA	Transfert d'adresse
MOVEQ	Transfert rapide
MOVEM	Transfert multiple de registres
LEA	Chargement d'une adresse effective

Les principales instructions du 68000 – L'addition

source + destination → destination

ADD	Addition binaire
ADDA	Addition à une adresse
ADDI	Addition immédiate
ADDQ	Addition rapide
ADDX	Addition étendue (source + destination + X → destination)

ASSEMBLEUR – Référence 68000 de Motorola

Les principales instructions du 68000 – La soustraction

destination – source → destination

SUB	Soustraction binaire
SUBA	Soustraction à une adresse
SUBI	Soustraction immédiate
SUBQ	Soustraction rapide
SUBX	Soustraction étendue (destination – source – X → destination)

Les principales instructions du 68000 – La multiplication

source × destination → destination

MULS	Multiplication signée
MULU	Multiplication non signée

Les principales instructions du 68000 – La division

destination ÷ source → destination. Remarque: le quotient est placé dans les 16 bits de poids faible de l'opérande destination. Le reste est placé dans les 16 bits de poids fort de l'opérande destination.

DIVS	Division signée
DIVU	Division non signée

ASSEMBLEUR – Référence 68000 de Motorola

Les principales instructions du 68000 – Autres

CLR	Mise à zéro d'un opérande (0 → destination)
EXT	Extension de signe (destination <i>sign-extended</i> → destination)
NEG	Négation (0 – destination → destination)
NEGX	Négation étendue (0 – destination – X → destination)

Les principales instructions du 68000 – Les opérations logiques

AND	ET
ANDI	ET immédiat
OR	OU
ORI	OU immédiat
EOR	OU EXCLUSIF
EORI	OU EXCLUSIF immédiat
NOT	Complémentation logique

ASSEMBLEUR – Référence 68000 de Motorola

Les principales instructions du 68000 – Les décalages et les rotations

ASL	Décalage arithmétique vers la gauche
ASR	Décalage arithmétique vers la droite
LSL	Décalage logique vers la gauche
LSR	Décalage logique vers la droite
ROL	Rotation vers la gauche
ROR	Rotation vers la droite
ROXL	Rotation étendue vers la gauche
ROXR	Rotation étendue vers la droite
SWAP	Échange les bits de poids faible et de poids fort d'un registre de donnée

Les principales instructions du 68000 – Les manipulations de bit

BTST	Test d'un bit (seul le <i>flag Z</i> est modifié)
BCLR	Test d'un bit et mise à zéro
BSET	Test d'un bit et mise à un
BCHG	Test d'un bit et changement

ASSEMBLEUR – Référence 68000 de Motorola

Les principales instructions du 68000 – Les tests et les comparaisons

TST	Test d'un opérande
CMP	Comparaison
CMPA	Comparaison à une adresse
CMPI	Comparaison immédiate
CMPM	Comparaison mémoire

Les instructions de test et de comparaison n'affectent pas les opérandes. Seuls les flags du registre CCR sont modifiés. L'instruction TST ne fait que tester l'opérande en mettant à jour les flags N et Z. Par exemple, l'instruction TST.L D0 positionnera :

- ❖ Z à 0 si D0 n'est pas nul ;
- ❖ Z à 1 si D0 est nul ;
- ❖ N à 0 si D0 est positif ou nul ;
- ❖ N à 1 si D0 est négatif.

Une instruction de comparaison soustrait un opérande source à un opérande destination (Dst - Src). Les flags N, Z, V et C sont modifiés en fonction du résultat. A la différence d'une instruction de soustraction, le résultat n'est pas stocké dans l'opérande destination (il est perdu). Par exemple, l'instruction CMP.L D0,D1 effectue l'opération D1 – D0 et modifie les flags en fonction du résultat. Le registre D1 reste inchangé.



Merci pour votre attention !