Chapitre 12:
Petites et grandes mémoires

12.1 - Introduction

1

La mémorisation est une fonction essentielle!

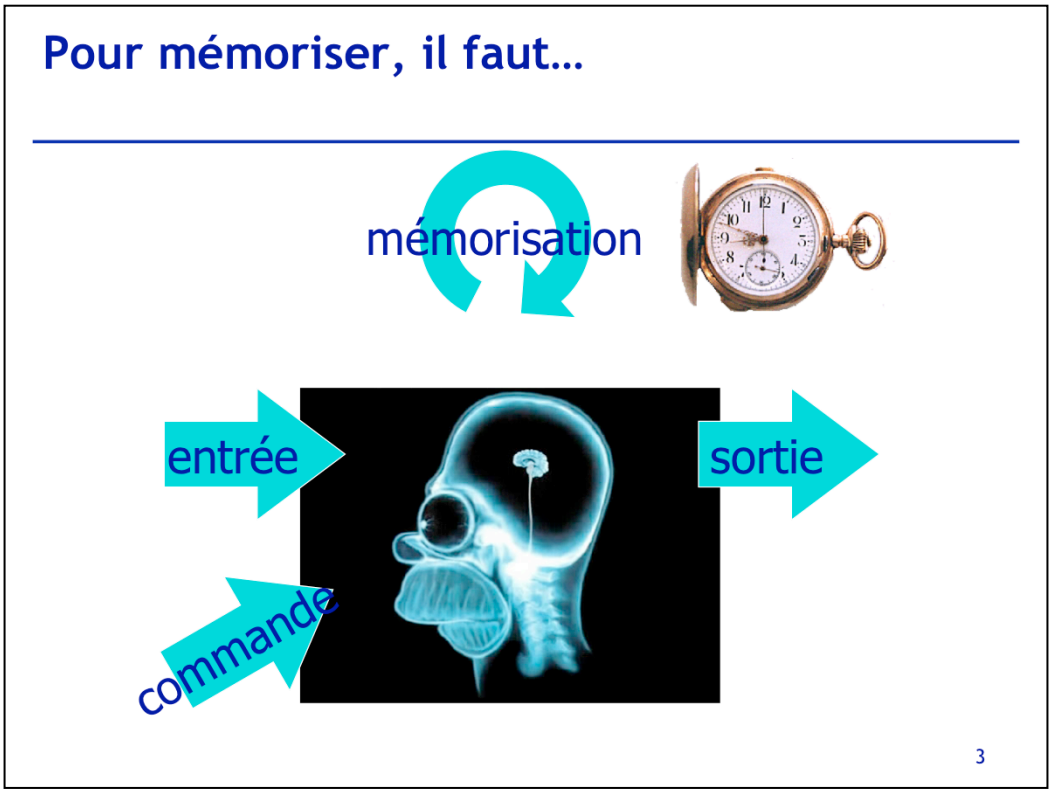
2

Nous avons vu au chapitre précédent (logique combinatoire) qu'une information disponible sous forme logique ou numérique peut être *traitée* (c'est-à-dire transformée) au moyen de portes logiques.

A côté de cette fonction de *traitement* de l'information qui est le but même de tout système électronique existe une seconde fonction toute aussi importante: la fonction de *mémorisation*. En pratique, tout montage d'électronique numérique non trivial comporte ces deux composantes: traitement et mémorisation. (C'est notamment le cas des systèmes informatiques: voir publicité ci-dessus.)

Les composants qui permettent la mémorisation –les mémoires– existent sous de nombreuses formes. Ce chapitre présente les principales formes de mémoire rencontrées en électronique.

En ajoutant aux fonctions purement combinatoires et asynchrones (chapitre 11) la fonction de mémoire, on ouvre de nouvelles et vastes possibilités (logique synchrone et logique séquentielle) qui seront traitées en détail dans le chapitre suivant.



Mémoriser semble évident. Cette fonction n'est cependant pas triviale…
Que faut-il pour faire une mémoire?

1) tout dispositif destiné à mémoriser doit… posséder des entrées et des sorties pour les *données*
En effet, pour être utile toute mémoire doit (au moins à un moment donné) *recevoir* les données à mémoriser (et donc doit posséder une *entrée*) et doit (au moins à un moment donné) être capable de *restituer* les données mémorisées (et donc doit posséder une *sortie*).

2) tout dispositif destiné à mémoriser doit… être capable de mémoriser
Qu'est-ce que mémoriser? Nous proposons la définition suivante: un dispositif est une mémoire s'il est capable de produire une information invariable dans le temps alors même que cette information a disparu de ses propres entrées.
On notera que la notion de mémoire fait inévitablement intervenir la notion de temps, ce que nous détaillerons par la suite dans le cas de l'électronique…

3) tout dispositif destiné à mémoriser doit… être commandé
Compte tenu des deux remarques précédentes, on peut considérer qu'une mémoire doit assurer *trois* opérations distinctes (et non une seule):
-l'écriture (c'est-à-dire l'enregistrement en mémoire d'une information)
-la lecture (c'est-à-dire la restitution de l'information mémorisée)
-la mémorisation proprement dite
En conséquence, il faut pouvoir indiquer à la mémoire quelle opération elle doit effectuer à un instant donné. Une mémoire requiert donc, en plus de l'entrée destinée à l'information, une entrée pour la *commande*.

4) Enfin on se souviendra que les circuits électroniques doivent, de manière générale, être alimentés en énergie pour fonctionner. Ce sera également le cas des mémoires électroniques (exception faite des mémoires dites "non volatiles").

Il existe de très nombreux types de mémoires suivant…

- …la quantité d'information à stocker
- …la volatilité de la mémoire
- …la manière (ordre) d'accéder à l'information
- …le temps d'accès à l'information

4

Nous avons donné la définition générale d'une mémoire à la page précédente. Il y a de nombreuses manières de répondre à cette définition générale: il existe donc de très nombreux types de mémoires.

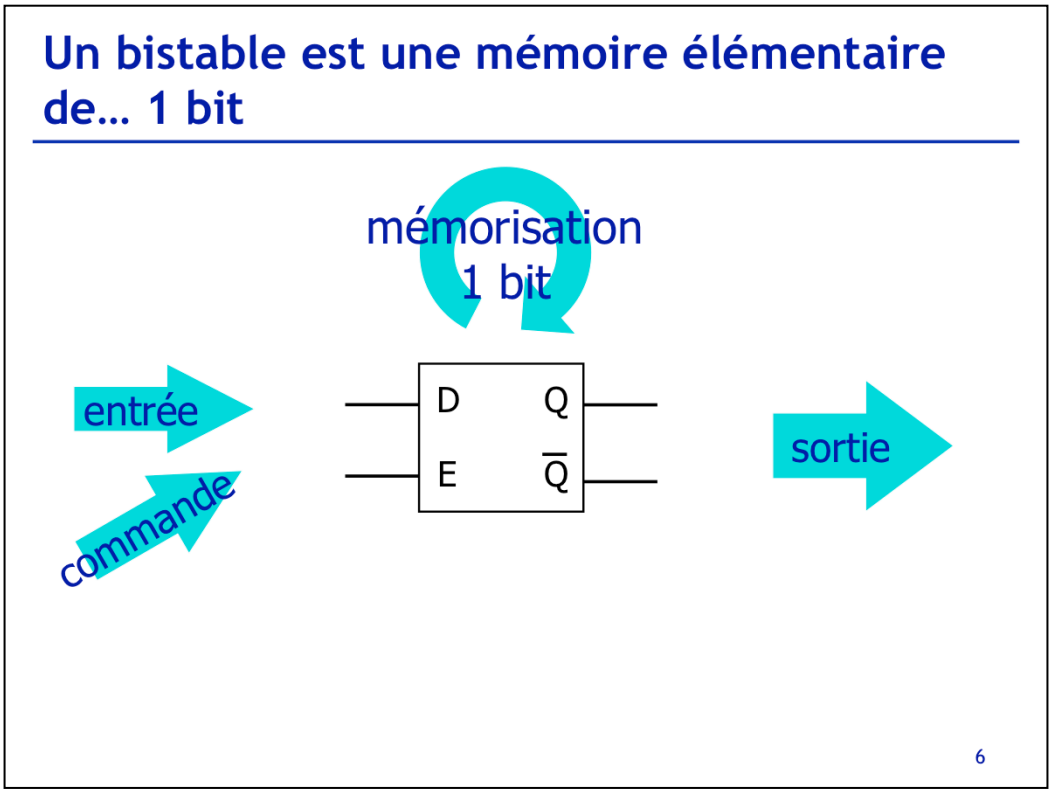
On peut notamment classer les mémoires suivant:
-leur capacité (la quantité d'information qu'elles peuvent stocker, qui peut varier de 1 bit à qques téraoctets dans le domaine informatique courant)
-la "volatilité" de la mémoire (l'information est-elle perdue lorqu'on cesse d'alimenter la mémoire?)
-la manière d'accéder à l'information (dans les mémoires "séquentielles", on ne peut accéder à une information qu'en ayant lu les informations précédentes; dans les mémoires "à accès aléatoire" comme les mémoires de PC on peut accéder directement à n'importe quelle information)
-le coût
-etc

Ce chapitre est organisé sur base d'un classement des mémoires en fonction de leur capacité. On examinera successivement:
-les bistables (1 bit)
-les registres (quelques bits)
-les mémoires à semi-conducteurs (quelques Ko à quelques Go)
-les mémoires de masse (quelques dizaines de Go et davantage)

Chapitre 12:
Petites et grandes mémoires

12.2 – Bistables
(1 bit)

5



Le bit peut être vu comme la quantité d'information minimale en électronique numérique.

Un bistable est une mémoire élémentaire capable de mémoriser 1 bit.

Le terme bistable vient du fait qu'un tel circuit possède deux états "stables" (on appelle état stable une situation électrique dans laquelle le circuit –pour autant qu'il soit alimenté- est capable de se maintenir indéfiniment): l'un correspond à la mémorisation de la valeur logique "0" et l'autre à la mémorisation de la valeur logique "1".

Comme expliqué en introduction, un bistable possède un certain nombre d'entrées et de sorties (alimentations non comprises) à propos desquelles on peut faire les remarques suivantes:
-ces entrées et sorties sont elles-mêmes des signaux logiques, c'est-à-dire des bits
-pour assurer toutes les opérations requises, un bistable doit posséder au moins deux entrées binaires (information et commande compris)
-pour des raisons pratiques, la plupart des bistables possèdent deux sorties binaires qui sont l'opposé l'une de l'autre: la présence de deux sorties opposées augmente les possibilités de connexion avec d'autres circuits et permet de choisir un signal actif à l'état haut ou à l'état bas pour commander les circuits placés en aval.

A titre d'exemple, on a représenté ci-dessus un bistable de type "D" (voir plus loin) qui possède:
-une entrée "D" (pour DATA) et une entrée "E" (pour ENABLE)
-deux sortie Q et ~Q
Un tel bistable possède en plus des alimentations, non représentées ci-dessus.

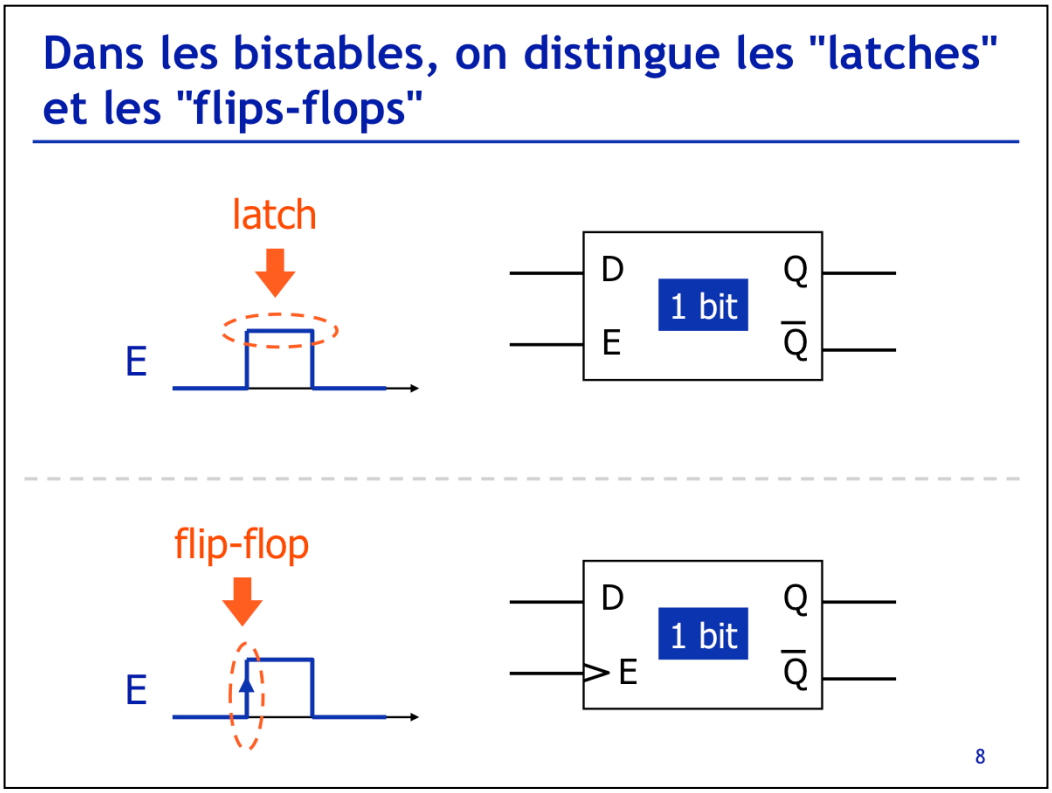
Chapitre 12:
Petites et grandes mémoires

12.2.1 – Les types de bistables (composant idéal)

7

Mémoriser un bit peut être fait de diverses façons. Les slides qui suivent présentent différents types de bistables qui se distinguent essentiellement sur base de leurs signaux d'entrée (ainsi que sur base de leur structure interne).

Les applications de ces différents bistables seront présentées dans le chapitre suivant.

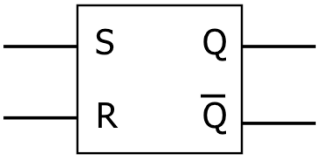


On fait classiquement une première distinction entre:
-les "latches" (singulier: "latch")
-et les "flips-flops" (singulier: "flip-flop")

Cette distinction se réfère à l'entrée de *commande* du bistable (pour le bistable D représenté ci-dessus, c'est l'entrée "E", voir plus loin): les latches sont activés par un *niveau* (haut ou bas suivant le type de latch) tandis que les flips-flops sont activés par un *flanc* (montant ou descendant suivant le type de flip-flop).

Sur base des autres signaux d'entrée, on définit quatre types de bistables dont le comportement est légèrement différent:
-le bistable RS
-le bistable D
-le bistable T
-le bistable JK
En théorie, chacun de ces bistables peut exister en version "latch" ou "flip-flop". En pratique, nous retiendrons cinq types:
-le RS latch
-le D latch
-le D flip-flop
-le T flip-flop
-le JK flip-flop

Le bistable RS (SR-latch)



R	S	Q_{n+1}	
0	0	Q_n	mémorisation
0	1	1	"set"
1	0	0	"reset"
1	1	✗	interdit!

9

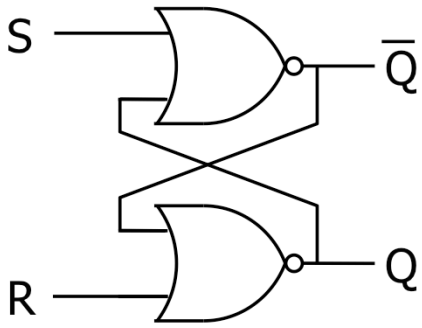
Voyons un premier type de bistable: le bistable RS déclenché par niveau ou "SR-latch".

Un bistable RS possède deux entrées binaires notées R et S (d'où le nom du bistable). Ces entrées ont les fonctions suivantes:
-"S" signifie "SET", c'est-à-dire la mise à l'état logique "1" du bistable
-"R" signifie "RESET", c'est-à-dire la mise à l'état logique "0" du bistable (d'où le terme "reset" pour désigner la remise à zéro d'un circuit)

En faisant l'hypothèse que ces entrées sont actives à l'état haut, on peut dresser la table de vérité d'un tel bistable:
- R=0 et S=0 (les deux entrées sont inactives): cette situation correspond à la mémorisation, c'est-à-dire que la sortie du bistable reste constante dans le temps (ce qui se note $Q_{n+1}=Q_n$ dans la table de vérité, l'indice se référant à des instants successifs).
-R=0 et S=1 (entrée SET active): cette situation correspond à l'écriture de la valeur logique "1" dans le bistable
-R=1 et S=0 (entrée RESET active): cette situation correspond à l'écriture de la valeur logique "0" dans le bistable
-R=1 et S=1 (deux entrées actives): cette situation est interdite puisqu'elle correspondrait à demander l'écriture simultanée d'un 0 et d'un 1 dans le bistable. Néanmoins rien n'empêche d'envoyer de tels signaux au bistable!

On notera que, en conséquence, les signaux de commande du bistable sont la plupart du temps *inactifs*, ce qui correspond à l'état de la mémorisation. Lors d'une écriture (d'un 0 ou d'un 1), un des deux signaux passe *temporairement* à l'état actif puis revient à l'état inactif. Les signaux R et S sont donc majoritairement à l'état inactif et comportent de temps en temps une *impulsion* (carrée) qui correspond à la mémorisation d'un "1" (impulsion sur S) ou d'un "0" (impulsion sur R).

Le bistable RS: schéma interne (NOR)



R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	X

10

Pour construire un bistable RS, il suffit de 2 portes logiques comme montré dans le schéma ci-dessus. On remarquera que:
1) la structure du montage est symétrique
2) chacune des portes reçoit un des deux signaux de commande (R ou S) par une de ses entrées
3) l'autre entrée d'une porte est connectée à la sortie de l'autre porte: on retrouve ici le concept de rétroaction déjà rencontré en électronique analogique (ampli-op). Celle-ci est une rétroaction positive (sans démonstration). On se rappellera qu'une rétroaction positive favorise l'instabilité du montage. C'est bien le cas ici car ce montage, comme on va le voir, peut se bloquer dans deux états extrêmes et opposés qui correspondent respectivement à un 1 et un 0 logiques. On remarquera dans ce cadre que les circuits *combinatoires* (voir chapitre précédent), eux, ne comportent aucune rétroaction. C'est donc la rétroaction qui offre au montage sa possibilité de mémoriser une information.
4) le bistable comporte deux sorties: Q et ~Q, néanmoins ~Q n'est pas obtenue en inversant (via une porte NOT) la valeur Q.

Analysons en détail le fonctionnement de ce bistable (4 états de la table de vérité):

1) R=S=0: c'est l'état de repos (aucune entrée n'est active). On sait que la porte supérieure a son entrée S à 0, mais cette information n'est pas suffisante pour trouver l'état de sortie de cette porte. Il faut donc faire une hypothèse sur Q [*], qui est la seconde entrée de la porte (par la rétroaction): supposons que Q vaut 0. Dans ce cas, la sortie de la porte, ~Q ,vaut 1. En conséquence, la porte inférieure voit à ses entrées les signaux R=0 et ~Q=1 et délivre donc un signal Q=0, ce qui correspond bien à l'hypothèse que nous venons de faire. Cet état est donc possible. Mieux: si R et S restent à 0, cet état n'a pas de raison d'évoluer: c'est un état stable.
Nous aurions également pu faire l'hypothèse Q=1. On peut vérifier qu'elle mène à l'état stable symétrique: Q=1 et ~Q=0.
Au repos, le bistable réalise donc bien la fonction de mémorisation puisqu'il est capable de "garder" un signal Q qui vaut soit 1 soit 0 (ainsi que la valeur ~Q opposée qui y correspond). Pour réaliser une mémoire, il faut néanmoins encore trouver le moyen de *choisir* l'état stocké dans le bistable.

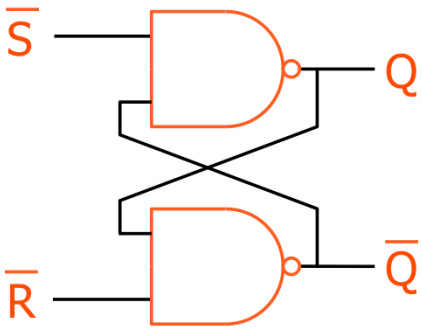
2) S=1 (actif), R=0: c'est l'opération "SET". Comme dans l'état précédent, R=0 ne fixe pas en soi l'état de la sortie. Par contre S=1 implique que quelle que soit la valeur de Q, la sortie de la porte supérieure ~Q vaut 0. En conséquence, la sortie de la porte inférieure, Q, vaut 1. En mettant S à 1, on force donc le signal Q à valoir 1.

3) S=0, R=1 (actif): c'est l'opération "RESET". S=0 ne fixe pas l'état de la sortie. Par contre R=1 implique que quelle que soit la valeur de ~Q, la sortie de la porte inférieure Q vaut 0. En conséquence, la sortie de la porte inférieure, ~Q, vaut 1. En mettant R à 1, on force donc le signal Q à valoir 0.

4) Enfin mettre S et R simultanément à 1 n'a pas beaucoup de sens puisque cela revient à vouloir mettre simultanément la mémoire dans l'état 1 et dans l'état 0. Cet état, s'il est indésirable, est néanmoins possible. On vérifiera que dans ce cas les signaux Q et ~Q valent tous les deux 0, ce qui est évidemment incohérent. Cet état est donc logiquement interdit!

[*] Tout comme pour les diodes, il faut ici faire un raisonnement par hypothèse.

Le bistable RS: schéma interne (NAND)



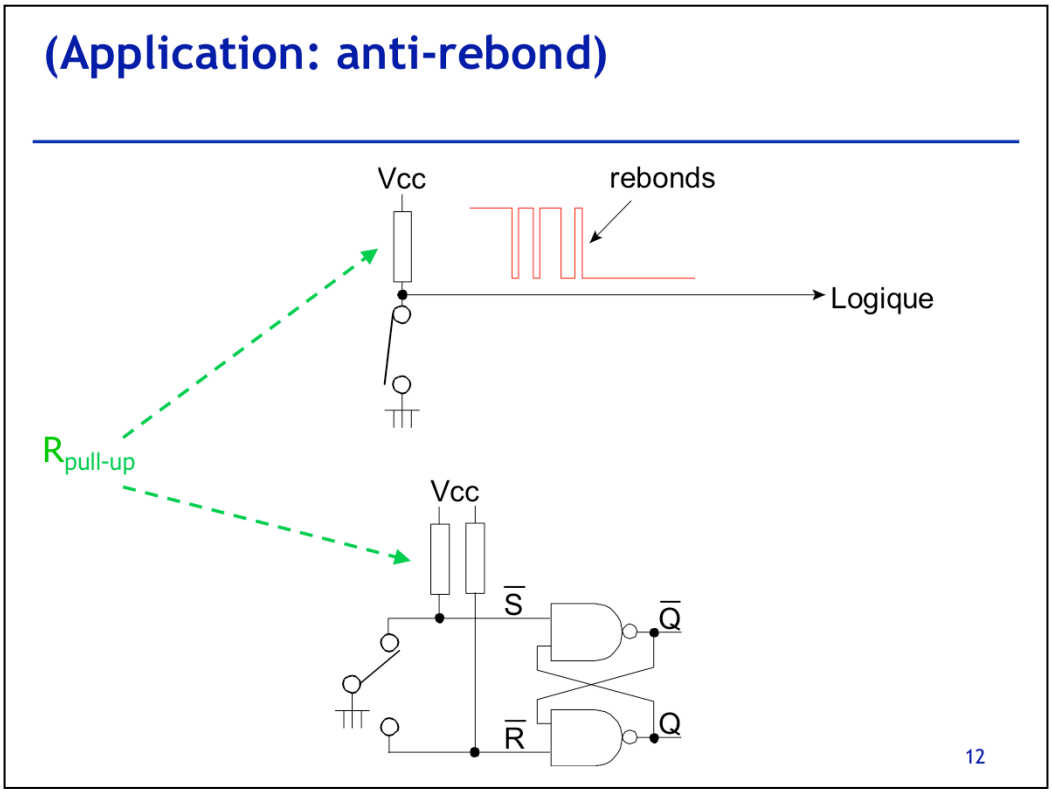
$\overline{R}$	$\overline{S}$	Q_{n+1}
1	1	Q_n
1	0	1
0	1	0
0	0	X

11

La figure ci-dessus présente une autre manière de faire un bistable RS, cette fois sur base de portes NAND.

La structure du montage est la même (rétroaction positive). On remarque que pour ce bistable, les entrées sont actives à l'état bas (voir table de vérité qu'on pourra vérifier à titre d'exercice). Il s'agit donc d'un bistable "~R/~S" (NOT_R / NOT_S)

question subsidiaire: que valent les sorties dans l'état interdit (R=S=0)?



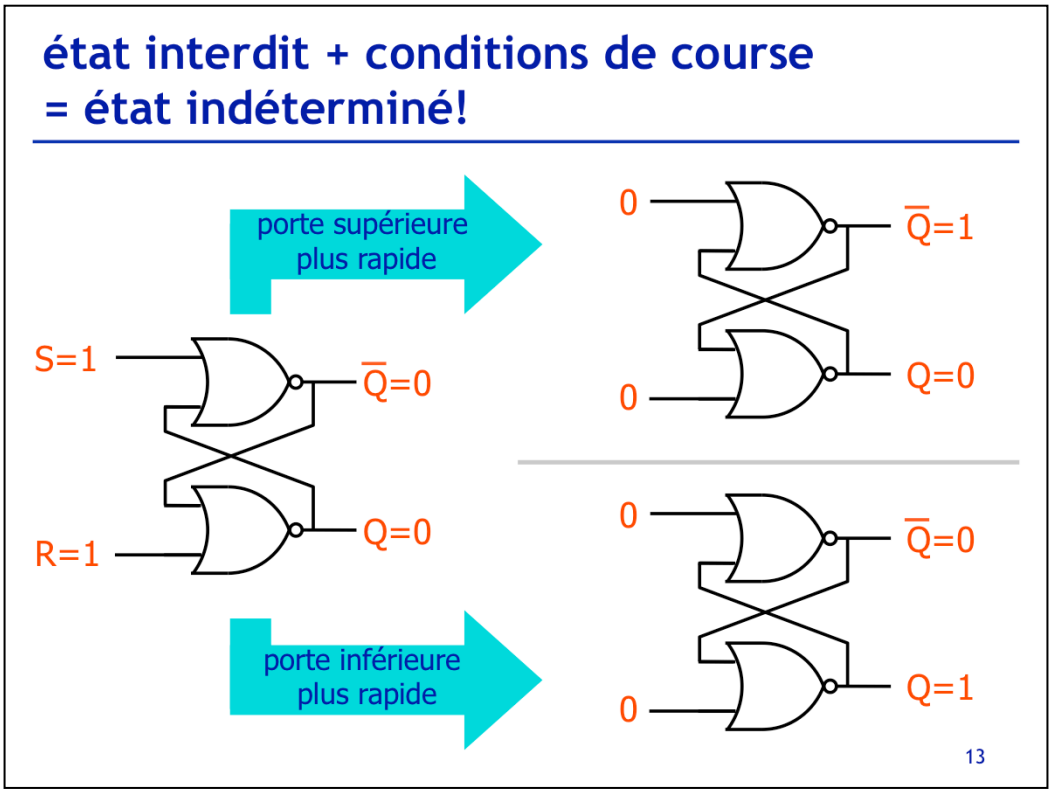
Voici une application très simple mais très courante d'un bistable RS : l'anti-rebond.

L'interrupteur, ou le bouton poussoir, reste l'une des interfaces homme/machine les plus simples, les plus utilisées et les moins chères. C'est également un des principaux moyen d'indiquer une "fin de course" dans un déplacement mécanique (porte motorisée, chariot de machine-outil, niveau d'une cuve, ...).

Le schéma supérieur montre comment transformer l'état d'un interrupteur en signal électrique, ce qui est nécessaire lorsqu'on désire commander un équipement électrique ou électronique (c'est fondamentalement le principe du transistor en commutation, sauf que cette fois-ci "l'interrupteur" n'est pas commandé électroniquement mais mécaniquement):
- lorsque l'interrupteur est fermé, il établit un court-circuit entre la sortie et la masse. Le potentiel de la sortie vaut donc 0V, soit l'état logique 0;
- lorsqu'il est ouvert, il interrompt tout courant dans la résistance. Il n'y a donc pas de chute de potentiel sur celle-ci et la sortie est portée au potentiel Vcc, soit l'état logique 1.
N.B.: La résistance qui se trouve dans ce schéma est appelée "**résistance de pull-up**" (littéralement: pull-up = tirer vers le haut). Ce nom vient du fait qu'une telle résistance a pour rôle de fixer (à la valeur VCC) le potentiel de la sortie lorsque l'interrupteur est ouvert. Néanmoins lorsque l'interrupteur est fermé, cette résistance n'empêche pas le potentiel de la sortie d'être celui de la masse. En d'autres termes, une telle résistance fixe une valeur "par défaut" du potentiel de sortie. Similairement, on parlera de résistance de pull-down lorsqu'une telle résistance connecte par défaut un noeud à la masse ou à un potentiel négatif.

Le défaut le plus gênant d'un tel interrupteur est qu'il a tendance à "rebondir" lorsqu'on l'actionne, ce qui génère des transitions parasites dans l'état logique de sortie. Ces oscillations logiques sont parfois intolérables; citons par exemple:
- les circuits logiques séquentiels dans lesquels deux états 0 successifs ont des effets différents sur le système (ex: poussoir d'ouverture-fermeture de porte motorisée)
- plus généralement, les circuits où le nombre de manoeuvres de l'interrupteur fait l'objet d'un comptage

Dans ce cas, le bistable RS constitue un excellent "anti-rebond". L'interrupteur simple à 2 pôles doit être remplacé par un inverseur (schéma du bas, à 3 bornes) dont la borne commune est à la masse et les deux sorties sont reliées aux entrées ~R et ~S placées à leur état inactif par des "pull-up".
L'action anti-rebond résulte alors de deux propriétés:
- les entrées du bistable ne sont actives qu'à tour de rôle; le premier passage à 0 de l'entrée ~S fait basculer Q à 1;
- les rebonds suivants sont alors ignorés pour autant que l'amplitude mécanique du rebond soit inférieure à la distance qui sépare les deux contacts. Dans cette hypothèse en effet, le signal ~S oscille bien entre 1 et 0 lors des rebonds, mais ceci n'implique pas de changement d'état du bistable puisque sa commande oscille entre "mémorisation" et "set".
Il en va de même lors du basculement de l'interrupteur dans l'autre sens.



Comme on l'a dit, l'utilisation correcte d'un bistable RS consiste à le commander par impulsions: à partir de l'état de repos (R=S=0) qui doit être son état par défaut, on fait passer R *ou* S temporairement à 1 (pour mémoriser respectivement un "0" ou un "1"). De cette manière, on évite que R et S soient simultanément à 1, ce qui provoquerait une incohérence dans les sorties.

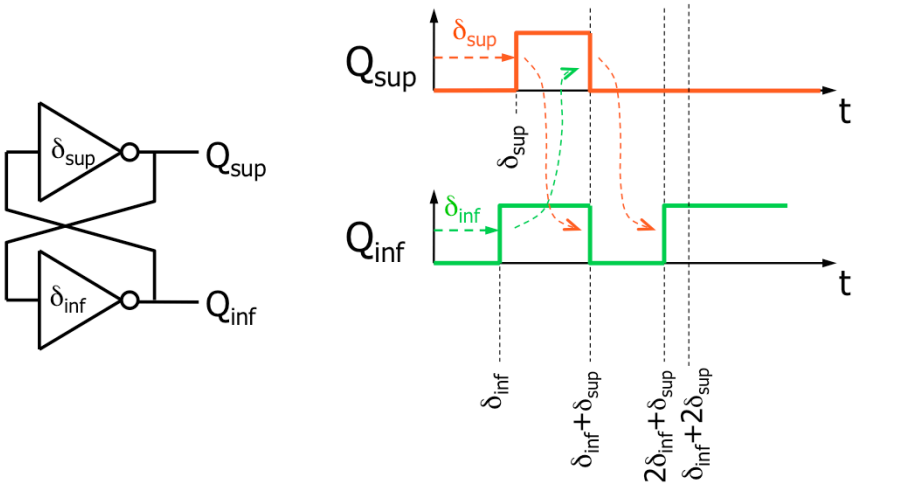
Supposons néanmoins (*) qu'une "fausse manoeuvre" mette R et S simultanément à 1. Une telle fausse manoeuvre pourrait par exemple résulter des conditions de course dans les circuits situés en amont du bistable: elle est donc tout-à-fait possible dans un système réel.

Supposons en plus qu'on quitte l'état interdit (R=S=1) en revenant directement à l'état de mémorisation (R=S=0) (**). Il se produit alors un phénomène particulier: le bistable mémorise un état, mais celui-ci est *indéterminé*, c'est-à-dire qu'on ne contrôle pas quel est l'état logique ("1" ou "0") mémorisé!
On peut montrer en effet (voir slide suivant) que:
-si la porte supérieure est plus rapide que la porte inférieure, le bistable mémorise un état logique "0"
-si la porte inférieure est plus rapide que la porte supérieure, le bistable mémorise un état logique "1"
C'est gênant pour une mémoire…

En résumé, on peut faire tomber le bistable RS dans un état indéterminé (ce qui fait perdre toute fiabilité au système) si l'on passe directement de l'état interdit à l'état de repos. Or, compte tenu des conditions de course, on ne peut pas garantir qu'on pourra éviter une telle transition dans les circuits réels si on a placé en amont du bistable des circuits combinatoires asynchrones.

(*) dans un bistable NOR pour fixer les idées.
(**) en réalité, une transition rigoureusement simultanée des deux signaux est impossible. Néanmoins la conclusion est la même: suivant le signal le plus rapide (R ou S), le bistable tombe dans l'un ou l'autre état sans qu'on puisse prévoir lequel.

état interdit + conditions de course = état indéterminé!



14

Voici ce qui se passe, par exemple, lorsque la porte inférieure est la plus rapide.

N.B.: nous utilisons deux propriétés:
1) en supposant que les signaux R et S étaient à 1 et passent simultanément à 0 à l'instant t=0 (et restent ensuite dans cet état), on peut vérifier que les portes NOR se comportent pour la suite comme des portes NOT inversant chacune la sortie de l'autre porte.
2) pour que la sortie d'une porte bascule, il faut que son (ou ses) entrée(s) reste(nt) constante(s) pendant une durée au moins égale au délai de cette porte (ceci vient du fonctionnement analogique des transistors internes à la porte).

On peut tenir le raisonnement suivant:
A l'instant 0, R et S (non représentés ci-dessus) passent tous les deux de 1 à 0. Ces transitions provoquent le basculement (de 0 à 1) des deux portes, chacune après leur propre délai, de sorte que:

- après δ_{inf}, Q_{inf} passe de 0 à 1
- après δ_{sup}, Q_{sup} passe de 0 à 1

Mais…

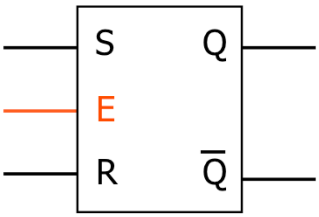
- Q_{sup} est aussi l'entrée de la porte inférieure, de sorte que le basculement de 0 à 1 de Q_{sup} en δ_{sup} provoque le basculement de Q_{inf} (de 1 à 0) après un nouveau délai δ_{inf}, c'est-à-dire en $\delta_{inf}+\delta_{sup}$
- de la même manière, Q_{inf} est aussi l'entrée de la porte supérieure, de sorte que le basculement de 0 à 1 de Q_{inf} en δ_{inf} provoque le basculement de Q_{sup} (de 1 à 0) après un nouveau délai δ_{sup}, c'est-à-dire en $\delta_{inf}+\delta_{sup}$

Ensuite…

- le basculement de 1 à 0 de Q_{sup} en $\delta_{sup}+\delta_{inf}$ provoque à nouveau un basculement (de 0 à 1) de Q_{inf} après un nouveau délai δ_{inf}, c'est-à-dire en $2\delta_{inf}+\delta_{sup}$
- on pourrait croire que de la même manière le basculement (de 1 à 0) de Q_{inf} en $\delta_{inf}+\delta_{sup}$ provoquera un nouveau un basculement (de 0 à 1) de Q_{sup} après un nouveau délai δ_{sup}, c'est-à-dire en $\delta_{inf}+2\delta_{sup}$. Il n'en est cependant rien car, la porte inférieure étant plus rapide, Q_{inf} aura déjà rebasculé en $2\delta_{inf}+\delta_{sup}$ (voir ci-dessus), c'est-à-dire *avant* $\delta_{inf}+2\delta_{sup}$, de sorte que la porte supérieure ne verra pas l'état logique "0" de son entrée (Q_{inf}) pendant suffisamment longtemps pour changer d'état.

En conlusion, le circuit ci-dessus se bloque finalement dans l'état Q_{sup}=0 et Q_{inf}=1 lorsque c'est la porte inférieure qui est la plus rapide. On serait arrivé à la conclusion inverse en supposant que c'est la porte supérieure qui est la plus rapide. Si l'on revient au bistable RS du slide précédent, on vérifie bien que c'est la porte la plus rapide qui délivre en finale l'état logique "1".

Remède: le bistable RS avec entrée d'activation ("gated SR-latch")...



E	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	1	0	1
1	0	1	0
1	1	1	! ! !

15

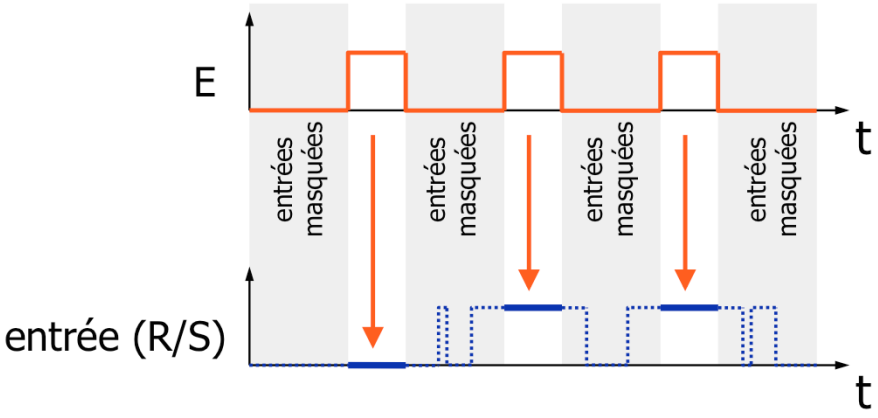
Pour résoudre le problème précédent (un bistable RS simple peut tomber dans un état indéterminé), qui n'est en fait rien d'autre qu'un problème de condition de course des portes logiques, on introduit le concept **d'entrée d'activation** (ENABLE en anglais). [*]

Le bistable RS avec entrée d'activation ("gated SR-latch" en anglais) comporte une entrée supplémentaire. Cette entrée permet de masquer (=inhiber) les entrées S et R. En d'autres termes (voir table de vérité):
-lorsque E=1 (entrées non masquées), le bistable fonctionne comme précédemment en fonction des valeurs présentes sur R et S
-lorsque E=0 (entrées masquées), les entrées R et S sont sans effet: le bistable est en mémorisation quelles que soient les valeurs de R et S

…voir suite sur slide suivant

[*] Nous avons déjà abordé ce concept lorsque nous avons introduit dans le chapitre 11 la possibilité d'inhiber (ou de masquer) un signal à l'aide d'une porte AND ou NAND.

…résoud le problème *si* les impulsions parasites ont lieu quand les entrées sont masquées



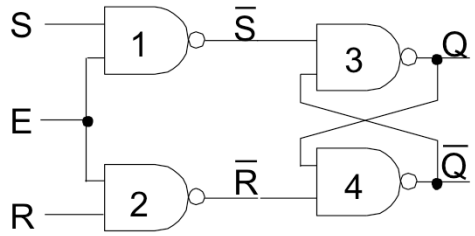
16

…L'idée consiste alors à ne faire passer E à 1 que lorsqu'on est sûr que les valeurs des signaux S et R sont stabilisées.
En d'autres termes, on s'arrange pour que les entrées S et R soient masquées au moment où ces entrées subissent les éventuelles impulsions parasites typiques des conditions de course (voir figure ci-dessus).

Le signal E est donc typiquement un signal périodique dont chaque période comprend:
-une durée pendant laquelle le signal est inactif (E=0 => les entrées sont masquées) et pendant laquelle peuvent se produire des impulsions parasites sans avoir d'effet néfaste sur le système
-une durée pendant laquelle le signal est actif (E=1 => les entrées sont valides)

Remarques:
1) Cette technique (entrée d'activation périodiquement active puis inactive) est en fait l'idée fondatrice de la logique *synchrone* qui sera détaillée dans le chapitre suivant. (Au sens de l'informatique ou de l'électronique numérique, le signal E est en fait un signal d'*horloge*.)
2) L'entrée d'activation utilisée comme ci-dessus garantit que les *impulsions parasites* dues aux conditions de course ne seront pas "vues" par le bistable. Il faut encore s'assurer qu'on n'enverra pas par erreur la commande R=S=1 au bistable.

Bistable RS avec entrée d'activation: schéma interne



E	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	1	0	1
1	0	1	0
1	1	1	! ! !

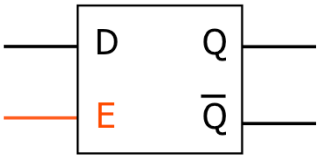
17

Pour réaliser en pratique l'entrée d'activation, on procède par exemple de la manière suivante:

En partant d'un bistable ~R/~S réalisé à l'aide de deux portes NAND (portes 3 et 4), il suffit d'ajouter en amont deux portes NAND supplémentaires (portes 1 et 2) pour valider/inhiber les entrées par un signal appelé E (pour "Enable"=autoriser). Comme il s'agit de portes NAND (réalisant une négation), les signaux seront en plus inversés. Les entrées R et S des portes 1 et 2 seront donc bien actives à l'état haut.

Dans cette configuration en effet, la mise à 0 de E force les signaux ~S et ~R à 1, ce qui correspond à l'état de repos (mémorisation) du bistable. Toute variation des signaux S et R n'est pas vue par les portes 3 et 4. Au contraire, lorsque E est activé (E=1), on peut vérifier que ~S vaut bien le complément de S et ~R vaut le complément de R. Les entrées R et S sont alors effectives (=non masquées). Un tel bistable reste dans la catégorie des "latches", c'est-à-dire des bistables déclenchés par un niveau puisque le bistable "voit" ses entrées R et S lorsque le signal E vaut 1.

Le bistable D déclenché sur niveau (D-latch)



E	D	Q_{n+1}
0	0	Q_n
0	1	Q_n
1	0	0
1	1	1

E	D	Q_{n+1}
0	X	Q_n
1	X	D

18

Passons maintenant au second type de bistable: le bistable D.

Tout comme le bistable précédent, le bistable D possède une entrée d'activation E qui permet de s'affranchir des conditions de course.

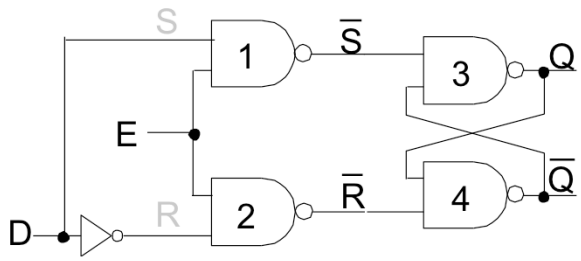
L'autre entrée du bistable est appelée D pour "DATA": elle reçoit tout simplement la donnée à mémoriser (0 ou 1).

Bien qu'il ne possède que deux entrées, ce bistable offre les mêmes possibilités que le bistable RS avec entrée d'activation (qui possède, lui, trois entrées). Il offre même un avantage supplémentaire: il n'y a pas d'état interdit (=> impossible d'obtenir des sorties incohérentes en pilotant mal les entrées).

La commande de ce bistable est en fait plus simple que celle du RS:
-lorsque E=0, le bistable est en mémorisation (il ignore l'entrée D)
-lorsque E=1, le bistable est "transparent": il recopie à tout instant son entrée D sur sa sortie
Pour écrire un "1" ou "0" dans le bistable, il suffit de présenter la donnée voulue à l'entrée D et, cette entrée D restant constante, de faire passer E de 1 à 0 (puis de maintenir E=0 tant qu'on veut la mémorisation).

Ce bistable est en fait très souvent rencontré en pratique. On en verra des applications dans le chapitre suivant.

D-latch: schéma interne

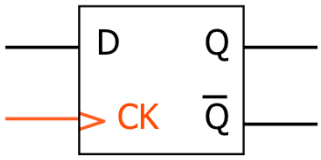


E	D	Q_{n+1}
0	X	Q_n
1	X	D

19

A titre d'information, voici une manière de réaliser un bistable D-latch: il s'agit en fait d'un bistable RS avec entrée d'activation dont les deux entrées R et S sont pilotées par un même signal D (S est égal à D tandis que R est égal à ~D, de sorte qu'il est impossible d'obtenir R=S=1)

Le bistable D déclenché sur flanc (D-flip-flop)



CK	D	Q_{n+1}
X	X	Q_n
↑	X	D

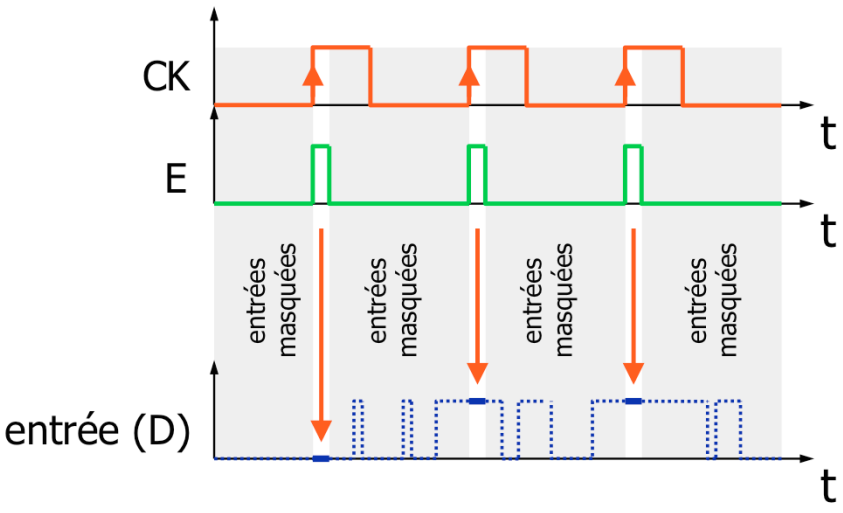
21

Le bistable D existe aussi en version "flip-flop". La différence est que l'entrée d'activation (appelée maintenant "CK" pour CLOCK = horloge) est active sur flanc (ici sur flanc montant) plutôt que sur niveau.

L'avantage est le suivant: en utilisant correctement l'entrée d'activation d'un latch, on élimine le problème des conditions de course. Mais ceci implique qu'il faut être sûr qu'aucune impulsion parasite n'intervient pendant que l'entrée d'activation est active, ce qui représente 50% du temps (c'est-à-dire beaucoup!) si cette entrée d'activation reçoit une onde carrée.

On peut faire mieux en construisant un bistable tel que les signaux d'entrée (ici D) ne sont valides que pendant la *transition* de l'entrée d'activation: on obtient alors un bistable déclenché par flanc, c'est-à-dire un flip-flop. Le bistable "voit" alors l'entrée D pendant une fraction du temps beaucoup plus faible, pendant laquelle il est beaucoup plus facile de s'assurer qu'il n'y a pas d'impulsions parasites.

Il est plus facile d'éviter les impulsions parasites lorsque l'entrée d'activation est activée sur flanc

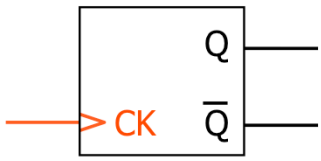


Ce schéma est à comparer au schéma similaire présenté pour l'entrée d'activation déclenchée par flanc.

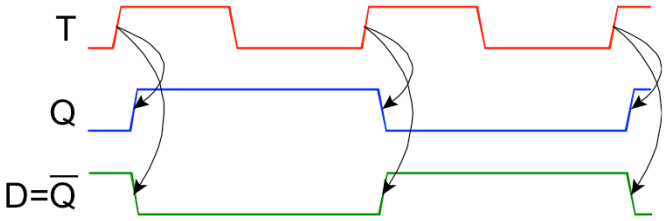
Pour le schéma ci-dessus, les entrées sont masquées tout le temps sauf pendant une brève période déclenchée par le flanc montant du signal CK.

En pratique, il suffit pour obtenir un tel fonctionnement de l'entrée d'activation d'intercaler un petit circuit combinatoire ("détecteur de flanc" que nous ne verrons pas ici) qui transforme le flanc montant du signal CK en une courte impulsion, qui va alimenter l'entrée ENABLE d'un bistable D déclenché par niveau.

Le bistable T (flip-flop) "commute" et divise la fréquence par 2



T	Q_{n+1}
X	Q_n
↑	$\overline{Q_n}$



23

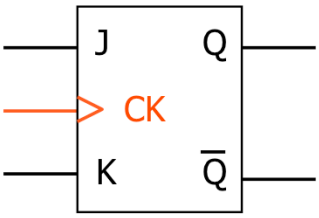
Troisième type de bistable: le bistable T est un bistable qui ne possède pas d'entrée autre (*) que celle prévue pour l'entrée d'activation (notée E ou CK).
La fonction de ce bistable est simple: à chaque flanc montant du signal CK, il change d'état (d'où le nom du bistable: "to Toggle" = basculer):
-s'il délivrait précédemment un "1", il délivre un "0"
-s'il délivrait précédemment un "0", il délivre un "1".

Un tel bistable a la propriété que le signal de sortie varie à la moitié de la fréquence du signal d'entrée CK (voir chronogramme ci-dessus): c'est donc aussi un "diviseur de fréquence".
Nous en verrons des applications dans le chapitre suivant.

(*) Si c'était strictement vrai, on ne pourrait pas fixer l'état de départ du bistable. En réalité, il existe une ou plusieurs entrées supplémentaires qui permettent de fixer l'état initial d'un tel bistable. Néanmoins un bistable T ne possède pas d'entrée de donnée au sens des bistables précédents (voir aussi plus loin).

(N.B.: Un bistable T peut notamment être réalisé par une rétroaction sur un bistable D. On pourra vérifier à titre d'exercice qu'une rétroaction de la sortie ~Q sur l'entrée D donne bien un comportement de bistable T.)

Le bistable JK (flip-flop)… résume tous les autres



J	K	CLK	Q_{n+1}	
LO	LO	X	Q_n	mem
HI	LO	↑	HI	set
LO	HI	↑	LO	reset
HI	HI	↑	$\overline{Q_n}$	toggle

24

Le dernier type de bistable est appelé JK.

La table de vérité du JK est identique à celle du RS pour les 3 combinaisons licites de ce dernier. Par contre, l'activation simultanée de J et K est autorisée, alors qu'elle est interdite dans le RS. Dans ce dernier cas, le JK change d'état à chaque flanc d'horloge, comme un bistable T.
Le JK offre donc simultanément les possibilités d'un bistable RS et d'un bistable T.

Types de bistables: que retenir?

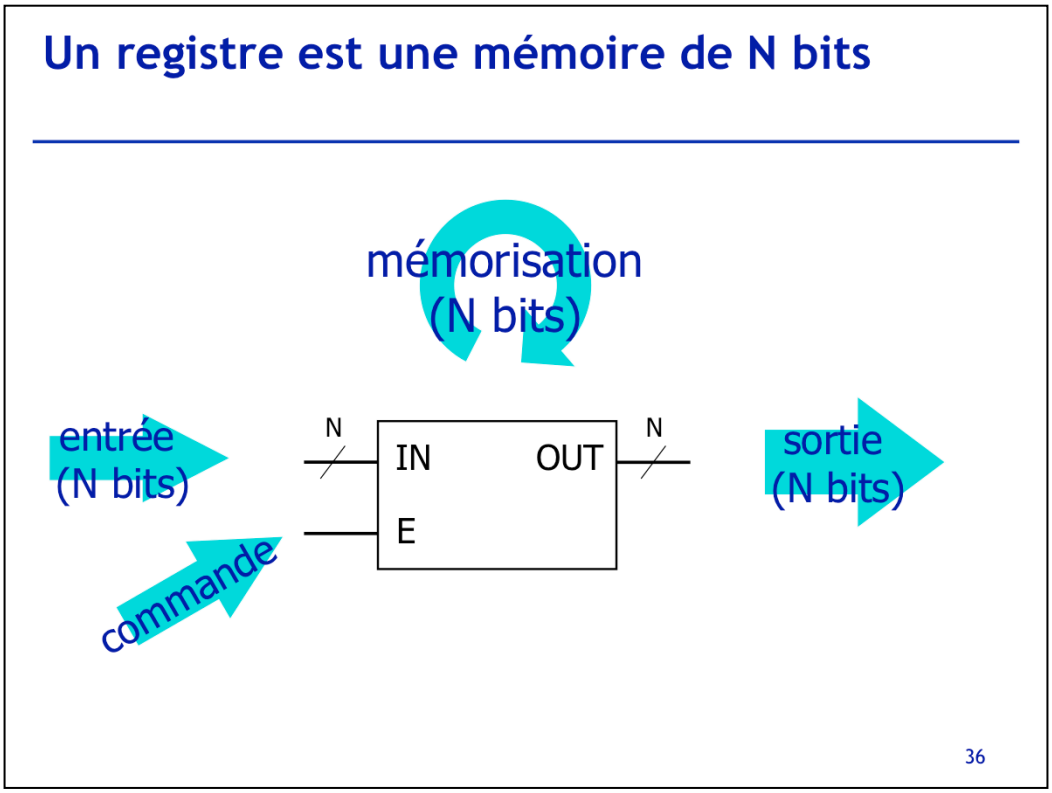
- 1) latch >< flip-flop
- 2) 4 types
  - RS (latch)
  - D (latch / flip-flop)
  - T (flip-flop)
  - JK (flip-flop)
- 3) principe de l'entrée d'activation
  - latch >< flip-flop

25

Chapitre 12:
Petites et grandes mémoires

12.3 – Les registres

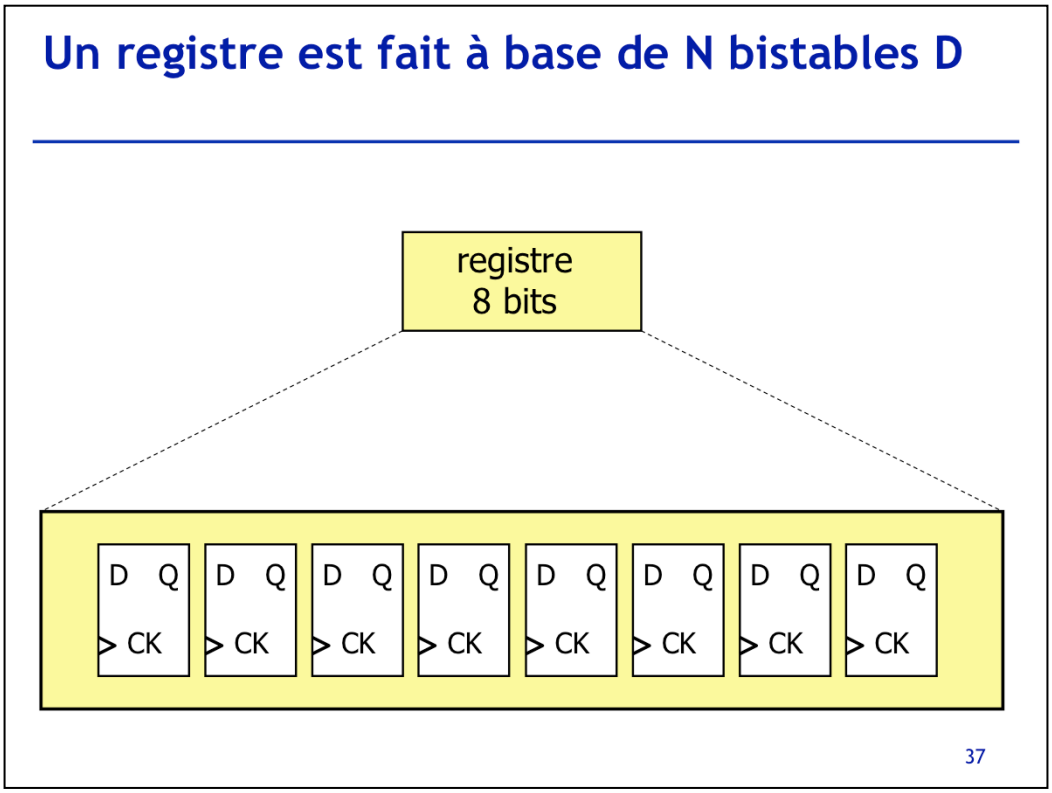
35



Si un bistable est une mémoire de 1 bit, un **registre** est une mémoire capable de stocker un *mot de N bits* (N étant relativement faible: au maximum quelques dizaines de bits).

Les registres contiennent le plus couramment 8 bits (un octet) ou un multiple de 8 bits (quelques octets).

Comme pour les autres mémoires, un registre a besoin d'entrées et de sorties pour les données, ainsi que d'une ou plusieurs entrées de commande, ce que nous allons détailler dans les slides suivants.

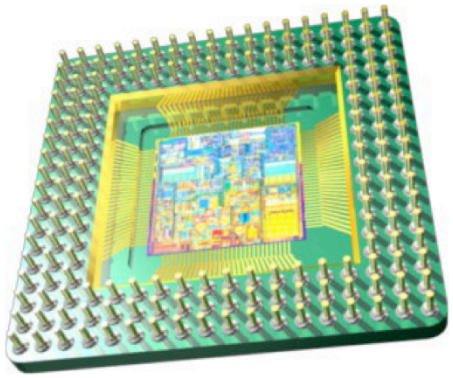


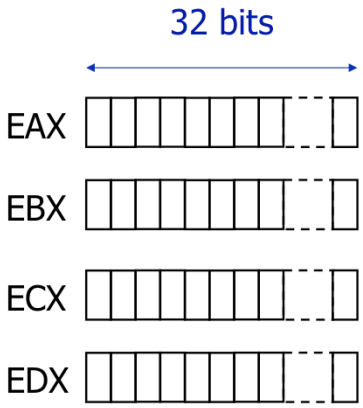
Un registre capable de stocker N bits est tout simplement fait à base de N bistables, en particulier des bistables D-latch ou D-flip-flop qui se prêtent particulièrement bien à cette utilisation.

La manière de connecter entre eux les différents bistables d'un registre dépend du type d'entrée/sortie du registre (série ou parallèle: voir slides suivants).

N.B.: On notera que puisqu'un registre contient N bistables, qu'un bistable contient typiquement 4 portes logiques et qu'une porte logique contient au moins 4 transistors, un registre de 8 bits contient au moins… 8*4*4 = 64 transistors!

Les microprocesseurs contiennent des registres



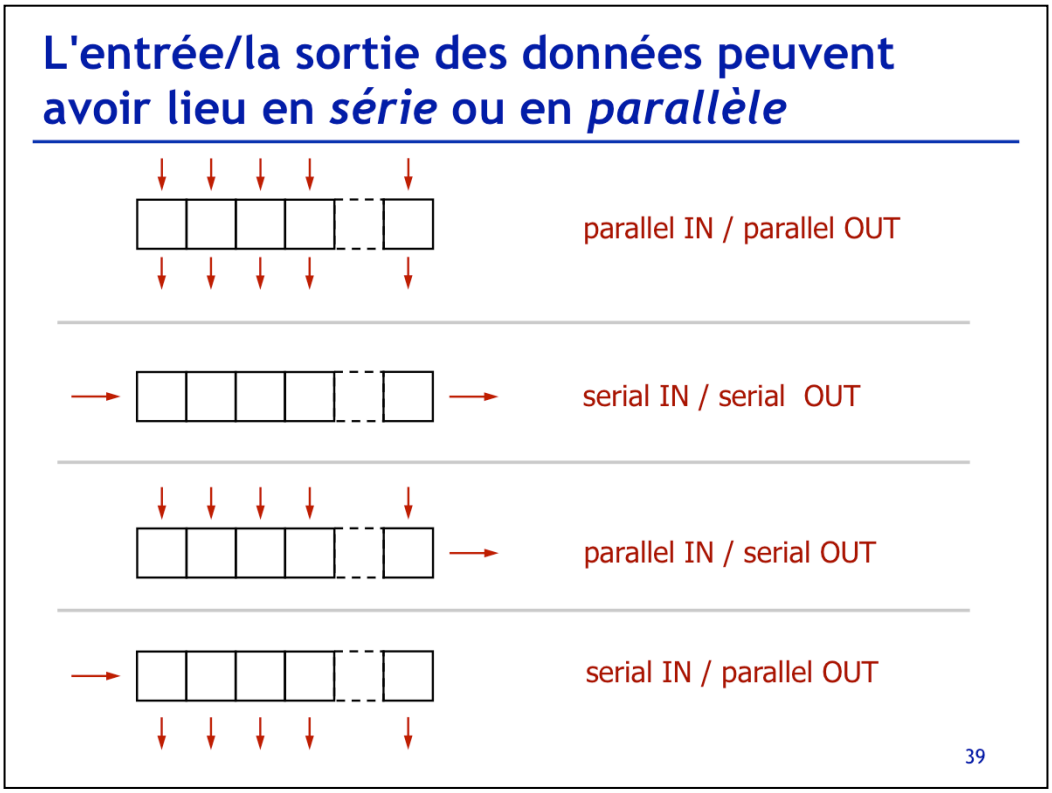


38

Les registres sont très couramment utilisés en électronique numérique (puisque les bits sont rarement manipulés un par un, mais plus souvent par mots).

En particulier, tout microprocesseur doit être capable de stocker de manière interne (pour les traiter ou pour son propre fonctionnement) un certain nombre de mots (de 8, 16, 32 ou 64 bits suivant le type de processeur) et contient à cette fin un certain nombre de registres. Dans les microprocesseurs, certains registres sont dits "à usage général" (general purpose registers) tandis que d'autres sont dédiés à des fonctions particulières.

Dans la série des Pentium, on trouve par exemple dans les registres "généraux" quatre registres de 32 bits chacun:
-EAX "accumulator": stockage temporaire de données et d'opérandes
-EBX "base register": stockage temporaire d'une valeur
-ECX "count register": nombre de répétition (boucles, décalages, etc)
-EDX "data register": stockage temporaire d'une donnée

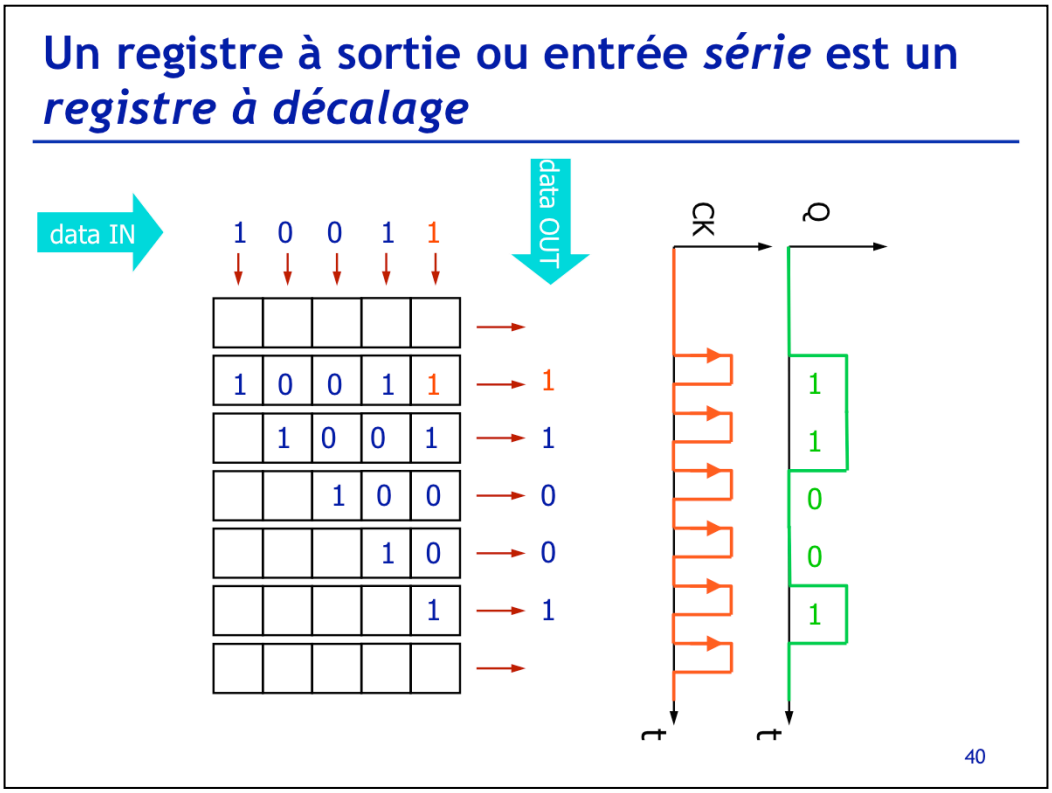


On a vu au chapitre 10 qu'un mot de N bits peut être transmis soit en série (N bits successifs sur une ligne) soit en parallèle (N bits simultanés sur N lignes).

Ces mêmes possibilités existent à l'entrée et à la sortie d'un registre. On peut donc imaginer différents types de registres:
-entrée parallèle / sortie parallèle
-entrée série / sortie série
-entrée parallèle / sortie série
-entrée série / sortie parallèle
(Il existe encore en fait davantage de nuances car lorsqu'on transmet des bits en série, on peut commencer par le bit de poids faible ou par le bit de poids fort.)

Les deux derniers types de registres sont en particulier utilisés pour changer le mode de transmission des données (série vers parallèle ou inversément).

Indépendemment, un registre comprendra au moins une entrée de commande qui s'utilisera comme l'entrée d'activation (sur niveau ou sur flanc) d'un bistable D.



Dans les registres dont au moins un des accès de donnée (entrée ou sortie) est au format série, l'écriture ou la lecture (suivant le cas) se fait en plusieurs coups d'horloge pour cet accès: à chaque coup d'horloge, tous les bits sont décalés d'un bistable (voir dessin ci-dessus) au sein du registre.

De tels registres portent le nom de registres à décalage. Il en existe de nombreux types qui ne seront pas détaillés ici.

Chapitre 12:
Petites et grandes mémoires

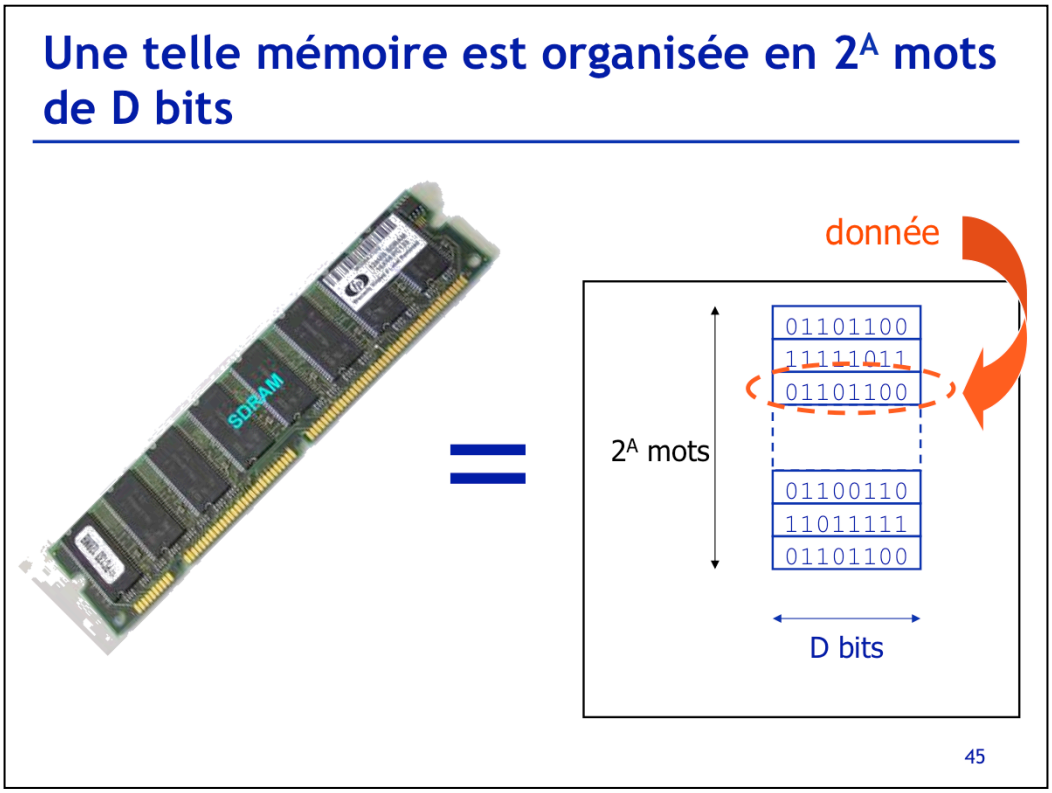
12.4 – Mémoires à semi-conducteurs

42

Mémoires à semi-conducteurs: de quelques Ko à quelques Go

44

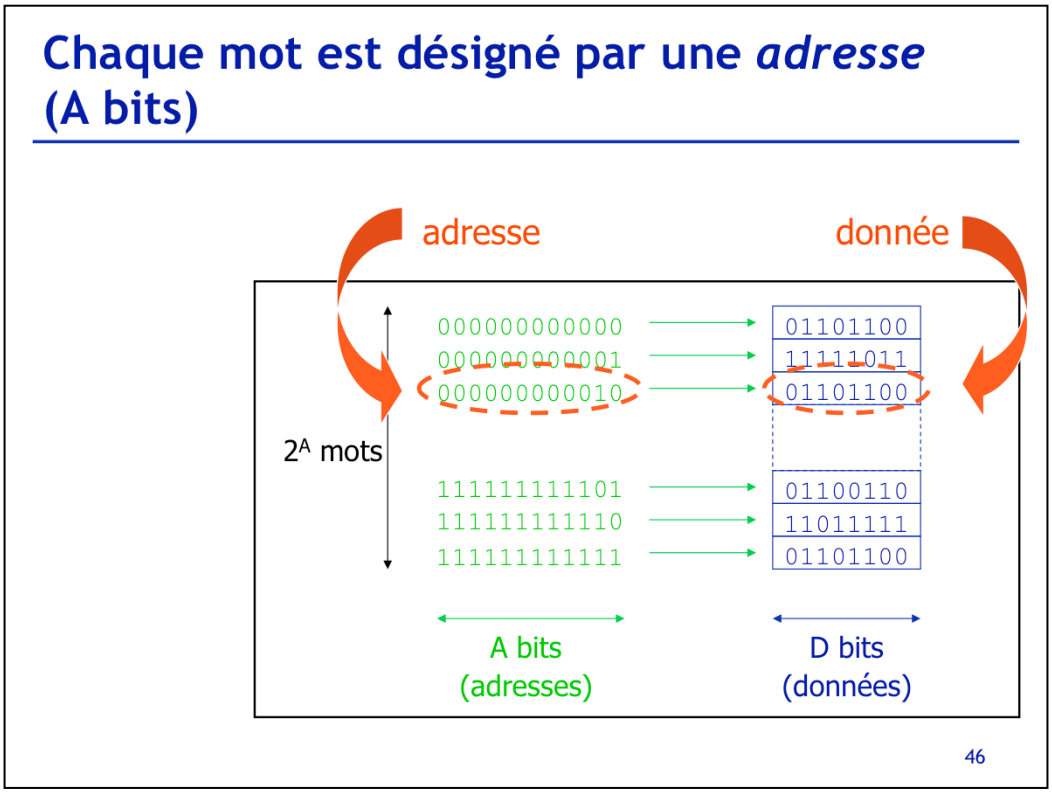
Par mémoire à semi-conducteurs, nous entendons (dans ce cours) une mémoire:
-réalisée en semi-conducteurs (=silicium)
-dont la capacité est entre quelques kilo-octets et quelques gigaoctets



Dans les systèmes numériques, l'information est souvent organisée en mots (de 8, 16, 32 ou 64 bits typiquement).

Les mémoires à semi-conducteurs sont donc également organisées en mots, c'est-à-dire qu'on y lit ou qu'on y écrit des mots entiers (et non des bits individuels).

Une telle mémoire peut être vue comme un ensemble de "cases", chaque case étant capable de stocker un mot de D bits (D=8 pour un octet).



- Un *bistable* peut stocker un bit.
- Un *registre* peut stocker un mot.
- Une *mémoire à semi-conducteur* peut stocker *beaucoup* de mots.

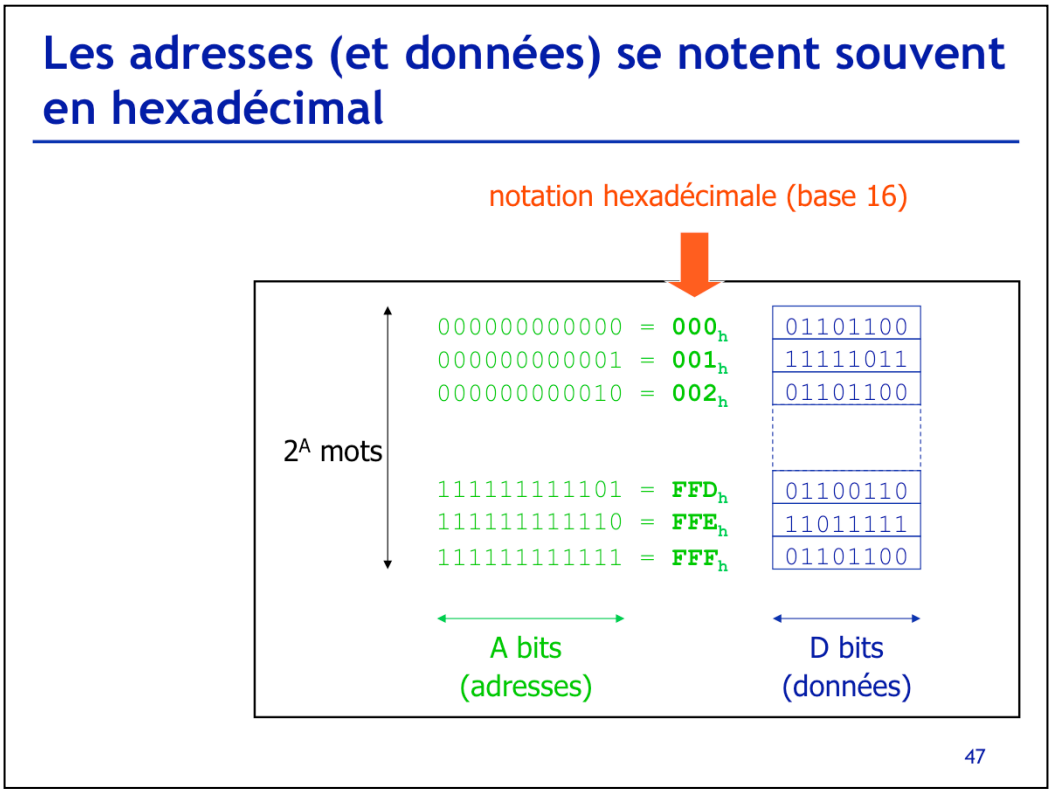
Le fait de stocker plusieurs mots différents implique un problème nouveau: il faut pouvoir choisir la "case" (appelée plutôt "emplacement mémoire") dans laquelle on veut lire ou écrire un mot binaire. Pour résoudre ce problème, on attribue simplement à chaque "case" une **adresse**, c'est-à-dire un *nombre binaire* qui désigne de manière univoque cette case. Comme N bits peuvent représenter 2^N valeurs différentes, une mémoire dont les adresses comportent A bits contient 2^A emplacements mémoires (cases) de D bits chacun.

Il est important de ne pas confondre le nombre de bits d'adresses (A) avec le nombre de bits de données (D). Ces deux valeurs sont tout-à-fait indépendantes.

La taille totale de la mémoire, en bits, dépend de ces deux nombres:
-une mémoire dont les adresses comportent A bits a une capacité de 2^A mots
-chacun de ces mots comporte D bits
-donc la capacité totale de la mémoire vaut 2^A.D bits

Exemple:
- une mémoire dont l'adresse est en 12 bits (A=12) contient 2^{12}=4096 mots, soit 4Kmots.
- si chacun des emplacements mémoire peut stocker un octet (D=8), la mémoire a une capacité de 4096 octets ou 4Ko
-si chacun des emplacements mémoire peut stocker deux octets (D=16), la mémoire a une capacité de 8192 octets ou 8Ko
-etc

Petit exercice: combien de bits d'adresse possède une mémoire de 128Mo (en supposant que chaque emplacement mémoire stocke un octet)?

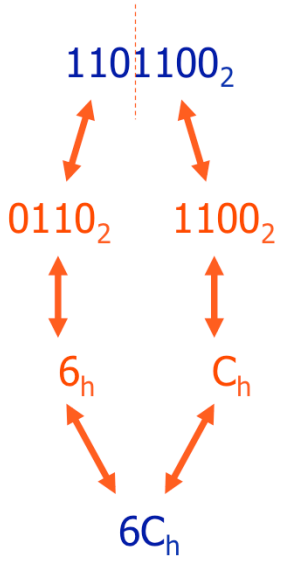


Les longues suites de 0 et de 1 étant difficiles à manipuler (pour un humain), on a l'habitude d'exprimer les longs nombres binaires, et donc notamment les adresses et les données, sous forme hexadécimale, c'est-à-dire en base 16.

La base 16 a l'avantage d'être beaucoup plus concise que la base 2, tout en permettant (au contraire de la base 10) une conversion très facile vers cette base 2, la base "naturelle" du numérique (voir slide suivant).

Conversion binaire <> hexadécimal

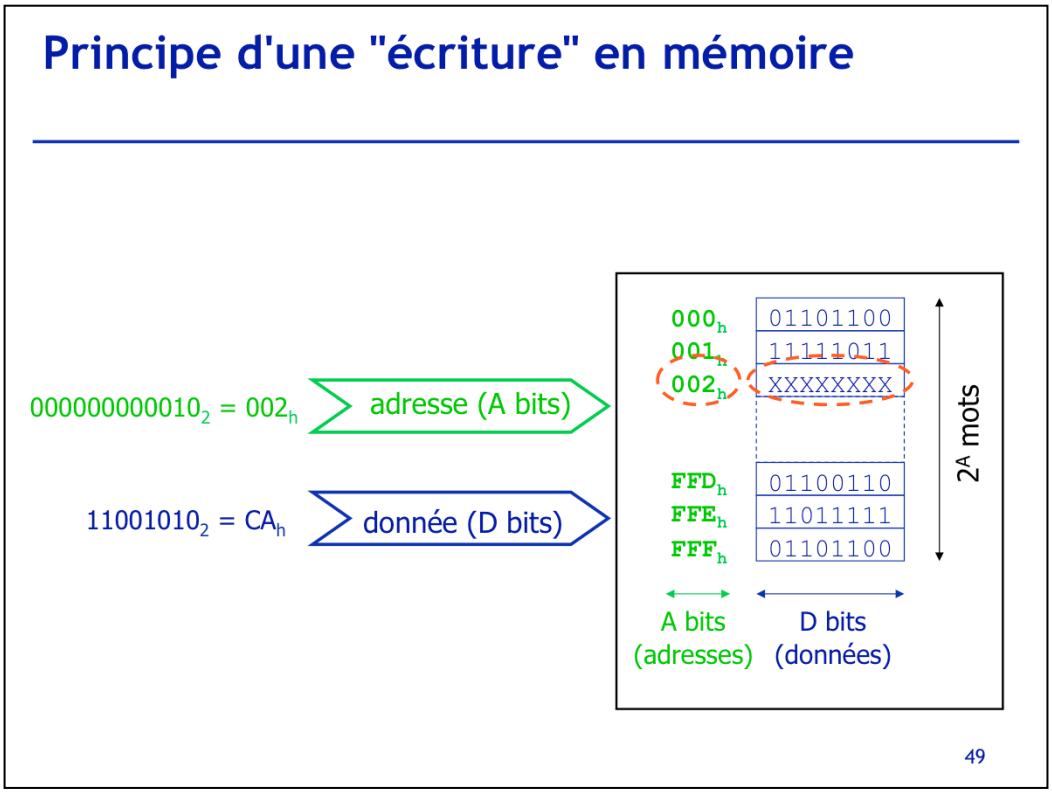
base 10	base 2	base 16
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



48

On a représenté ci-dessus les valeurs de 0 à 15 dans les trois bases (décimal, binaire, hexadécimal).
Attention: en hexadécimal, les valeurs A à F représentent bien chacune un *chiffre*!

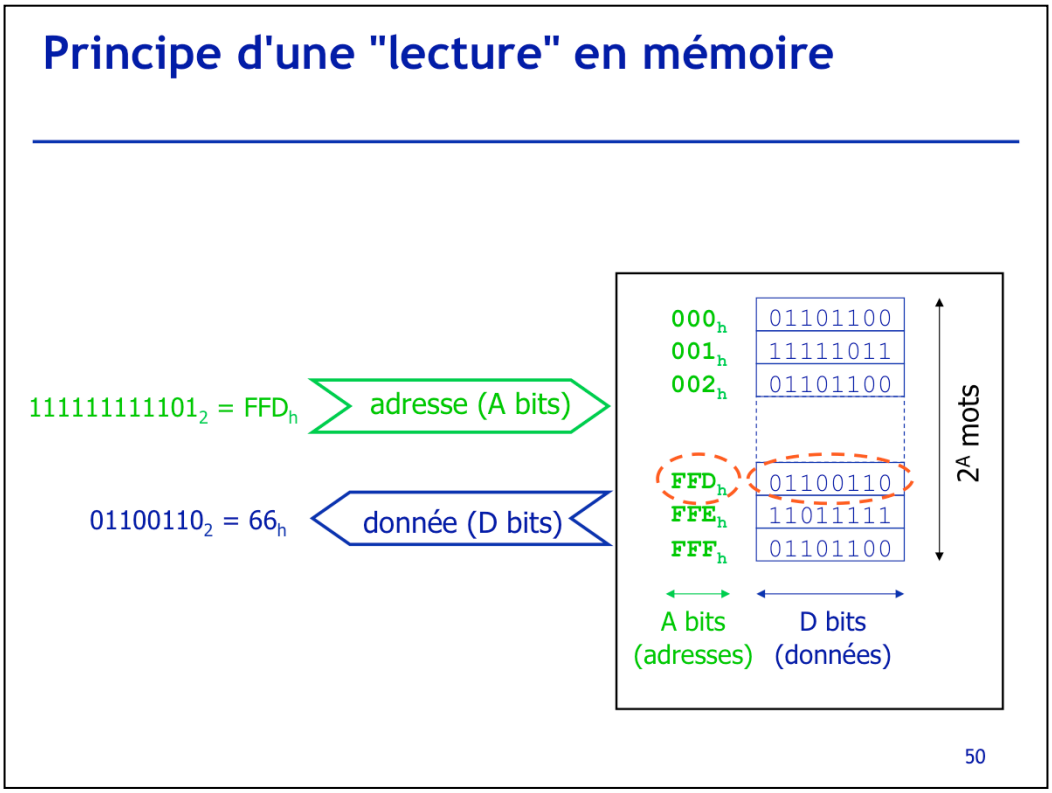
A droite:
- pour passer du binaire à l'hexadécimal, il suffit de grouper les bits par 4 (en commençant par la droite!) et de remplacer chaque groupe de 4 bits par le chiffre hexadécimal qui lui correspond dans le tableau de gauche.
- pour passer de l'hexadécimal au binaire, c'est l'inverse: chaque chiffre hexadécimal (0 à F) est remplacé par 4 bits.



Ayant défini ce qu'est une adresse, intéressons-nous maintenant aux signaux d'entrée/ sortie de la mémoire (*).

On peut facilement conclure que pour faire une *écriture* dans une mémoire à semi-conducteurs, il faut:
-envoyer une adresse (A bits)
-envoyer une donnée (D bits)

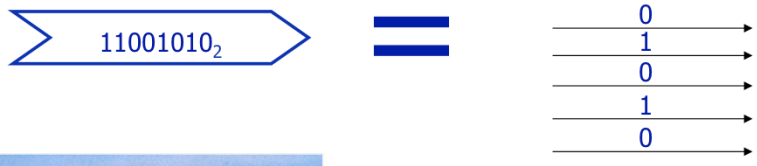
(*) on suppose ici qu'un autre circuit numérique (non représenté ci-dessus) communique avec la mémoire.



Inversément pour faire une *lecture* il faut simplement envoyer une adresse (A bits).

Après un certain délai, appelé "temps d'accès", la mémoire renvoie la donnée (D bits) correspondante.

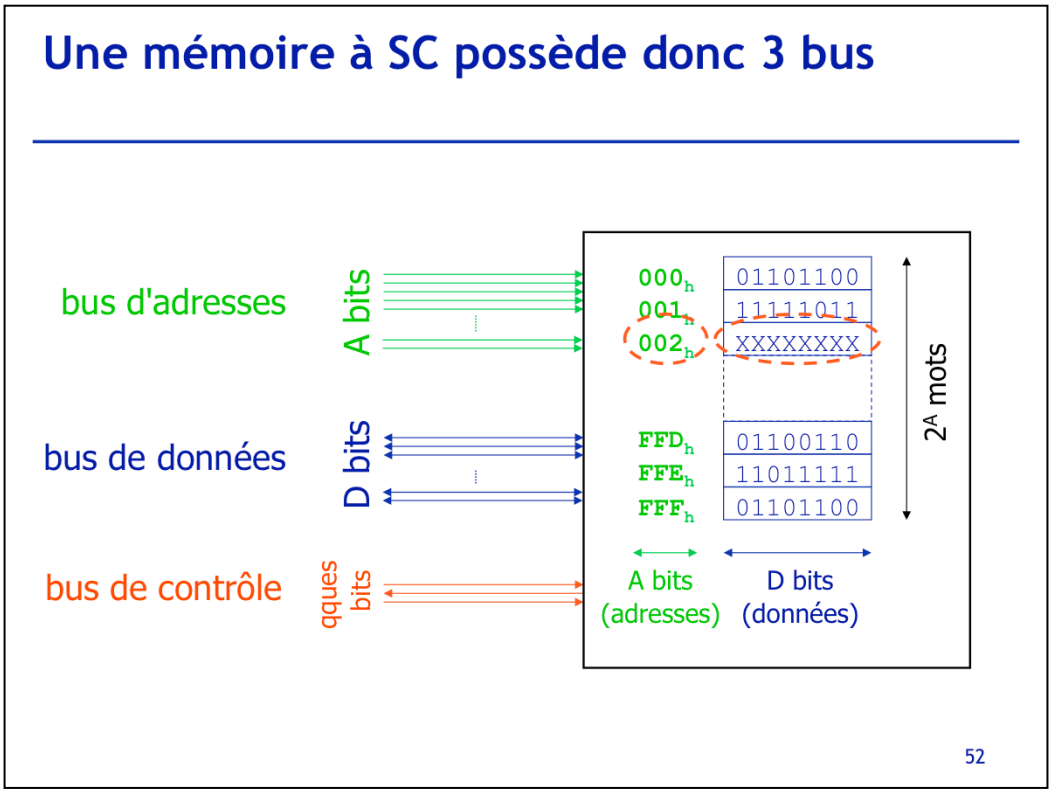
Un *bus* est un ensemble de conducteurs destinés à véhiculer un mot de N bits



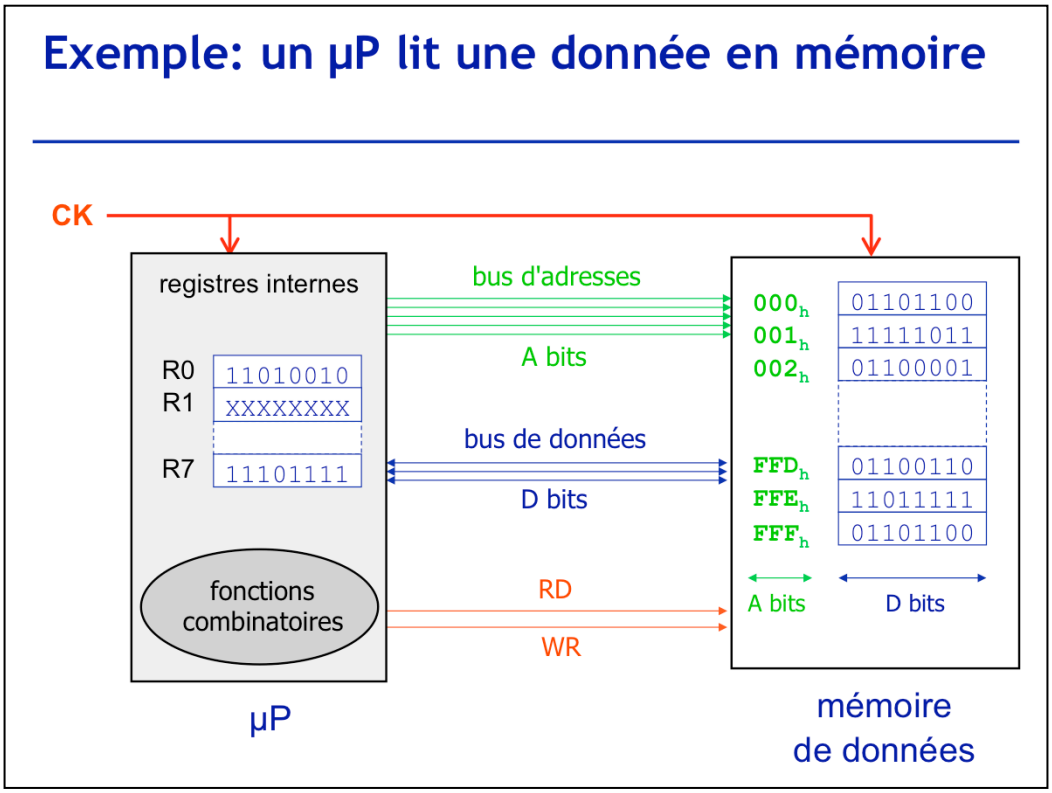
51

En pratique, les adresses et les données sont transmises entre la mémoire et les autres circuits via des "bus". Un **bus** est un ensemble de fils qui véhiculent ensemble un mot binaire.

N.B.: Dans le principe, il faut N ddps (donc 2N fils) pour transmettre N bits. En pratique, on peut parfois se contenter de N+1 fils (N signaux + 1 masse commune à tous les signaux).



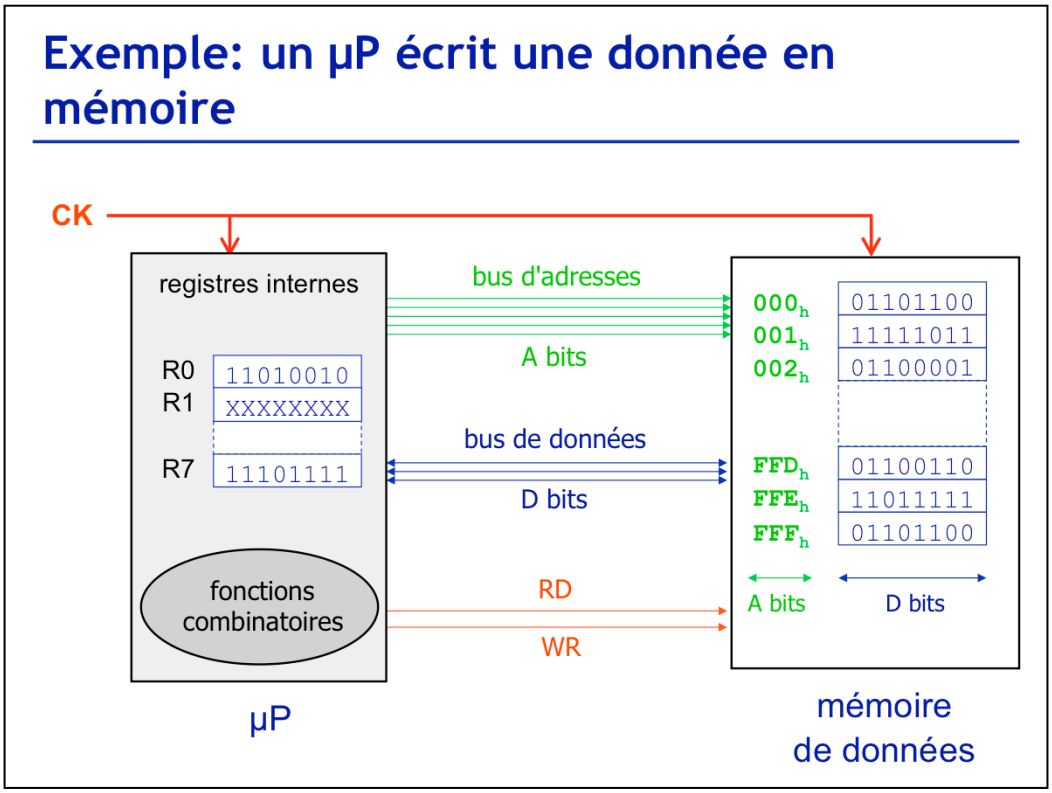
En pratique, une mémoire à semi-conducteurs échange donc de nombreux signaux binaires avec les autres circuits. Ces signaux sont groupés en trois catégories:
• le **bus d'adresse (A bits)**, qui est unidirectionnel (lors d'une lecture et lors d'une écriture, la mémoire *reçoit* toujours l'adresse)
• le **bus de données (D bits)**, qui est bidirectionnel (lors d'une écriture la mémoire *reçoit* la donnée, lors d'une lecture elle l'envoie)
• le **bus de contrôle**: un ensemble de quelques signaux binaires qui permettent à un circuit extérieur de commander la mémoire, et notamment d'indiquer un ordre d'écriture ou de lecture. Chacun de ces signaux est unidirectionnel, mais tous les signaux de commande ne vont pas forcément dans le même sens.



Explicitons le mécanisme d'une lecture en mémoire entre un circuit (ici un microprocesseur ou "µP") et une mémoire de données.

Remarques préliminaires:
-le circuit (ici le µP) qui lit ou écrit dans une mémoire à semi-conducteur doit lui-même posséder un endroit pour stocker la donnée lue ou écrite. Un tel circuit possède donc typiquement des **registres internes** (appelés ici R0 à R7) dans ce but. C'est seulement une fois que la donnée est copiée dans un registre interne que le circuit peut la manipuler (il ne peut pas la traiter directement lorsqu'elle est dans la mémoire).
-on suppose ici que le bus de commande est simplement constitué de deux signaux: un signal WR (=WRITE) commandant l'écriture lorsqu'il est actif et un signal RD (=READ) commandant la lecture lorsqu'il est actif.
-en principe le circuit et la mémoire reçoivent un même signal d'horloge (voir chapitre suivant: logique synchrone).

Lecture en mémoire
Imaginons par exemple que le µP veut copier la donnée située en mémoire à l'adresse 002_h dans son registre interne R1. Voici la succession d'étapes que cela nécessite:
• le µP place la valeur 002_h sur le bus d'adresses
• le µP active le signal RD du bus de commande
• le µP attend ensuite un certain délai, légèrement plus long que le temps d'accès de la mémoire
• lorsqu'elle reçoit le signal RD du µP, la mémoire place sur le bus de données la donnée qui est contenue à l'adresse 002_h (adresse qu'elle lit sur le bus d'adresses), ce qui prend un certain délai (temps d'accès)
• à la fin du délai d'attente, le µP écrit la valeur présente sur le bus de données dans le registre R1
• le µP désactive le signal RD du bus de commande et arrête d'imposer la valeur du bus d'adresses
• de la même manière, la mémoire arrête d'imposer la valeur du bus de données



Explicitons maintenant le mécanisme d'une écriture en mémoire d'un circuit (ici un microprocesseur ou "µP") vers une mémoire de données.

Ecriture en mémoire

Imaginons par exemple que le µP veut écrire la donnée située dans son registre R0 vers l'adresse FFD_h de la mémoire de données. Voici la succession d'étapes que cela nécessite:

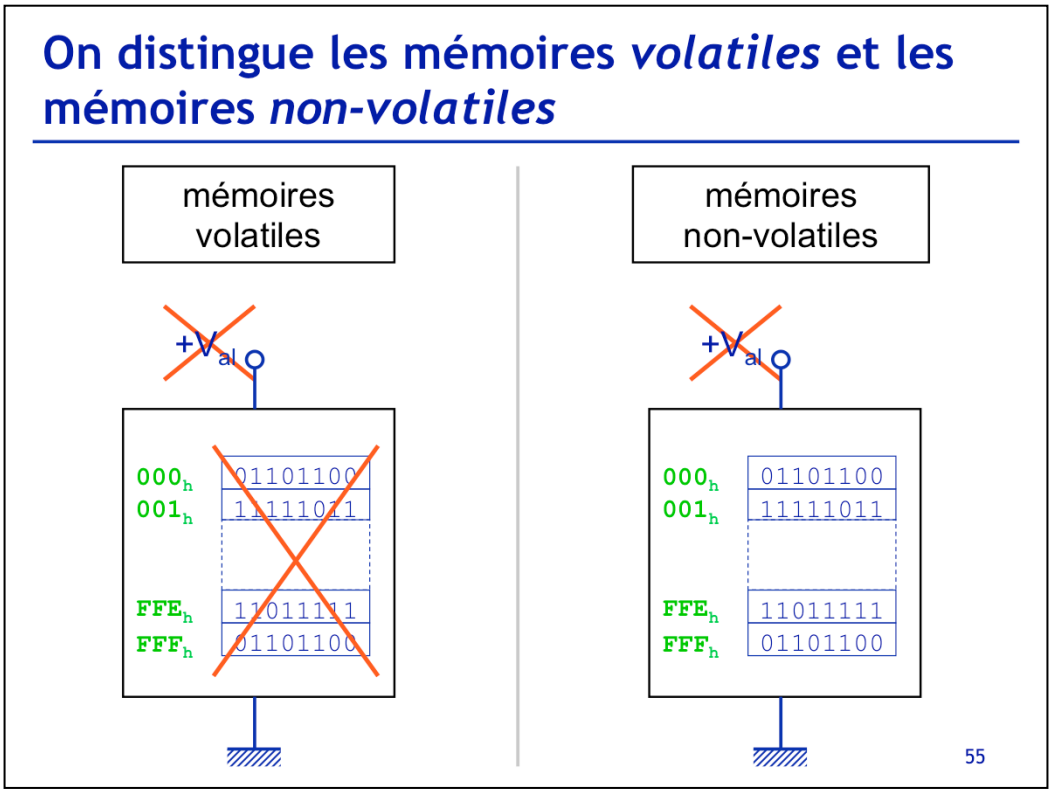
- le µP place la valeur FFD_h sur le bus d'adresses
- le µP place la valeur contenue dans le registre R0 sur le bus de données
- le µP active le signal WR du bus de commande
- le µP attend ensuite un certain délai
- lorsqu'elle reçoit le signal WR du µP, la mémoire copie la donnée présente sur le bus de données vers l'emplacement mémoire dont l'adresse est FFD_h (adresse qu'elle lit sur le bus d'adresses), ce qui prend un certain délai. La donnée présente précédemment dans FFD_h est écrasée.
- à la fin du délai d'attente, le µP désactive le signal WR du bus de commande et arrête d'imposer la valeur du bus d'adresses et du bus de données

Remarques:

-les opérations d'écriture et de lecture reposent sur le fait que le µP "connaît" les délais nécessaires à la mémoire pour effectuer ces opérations

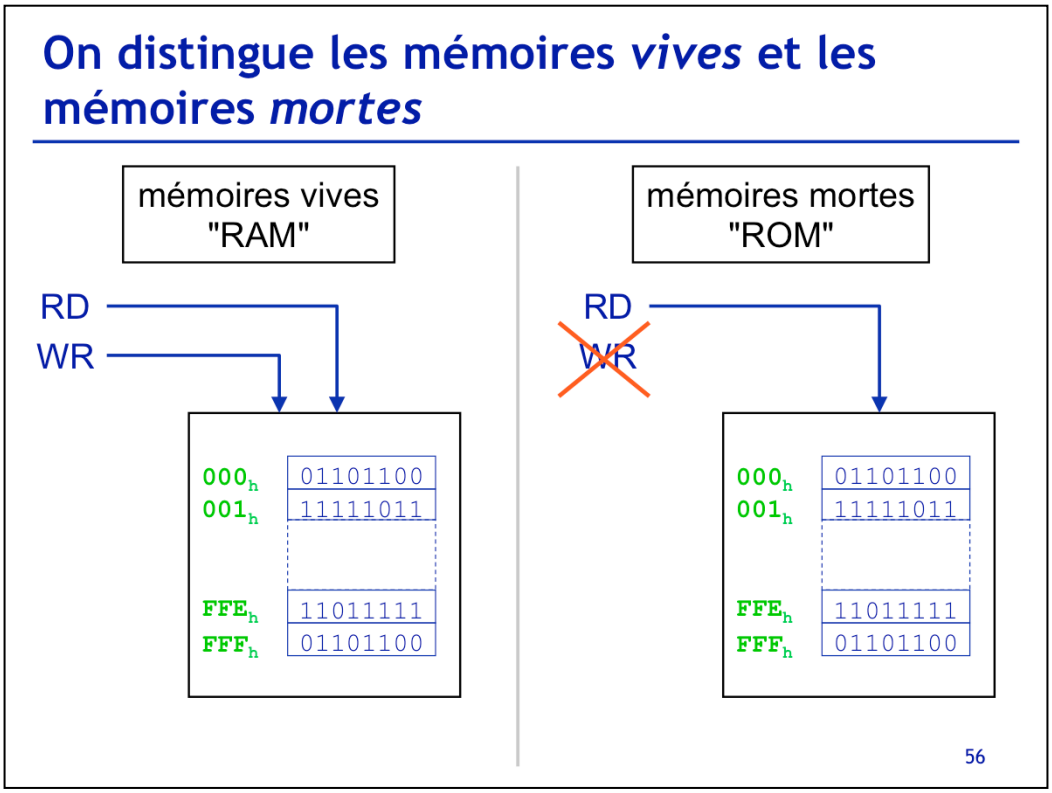
-les opérations d'écriture et de lecture étant une suite d'actions, le circuit (ici le µP) qui effectue ses opérations doit forcément être un circuit séquentiel (voir chapitre suivant)

-électriquement parlant, les valeurs "0" ou "1" transitant sur les bus correspondent à des tensions (ex: 0V ou 5V) imposées soit par le µP soit par la mémoire



Indépendamment de ce qui a déjà été dit, on distingue :

- Les mémoires "**volatiles**": les données stockées dans la mémoire sont perdues lorsqu'on coupe l'alimentation de cette mémoire
- Les mémoires "**non-volatiles**": les données stockées restent mémorisées lorsqu'on coupe l'alimentation



Et on distingue encore:

-Les mémoires "**mortes**" ou "read only" ou "**ROM**": les mémoires dans lesquelles on peut uniquement lire des informations (pas en écrire, sauf peut-être la première fois)
-Les mémoires "**vives**" ou "read/write" : les mémoires dans lesquelles on peut aussi bien écrire que lire des informations

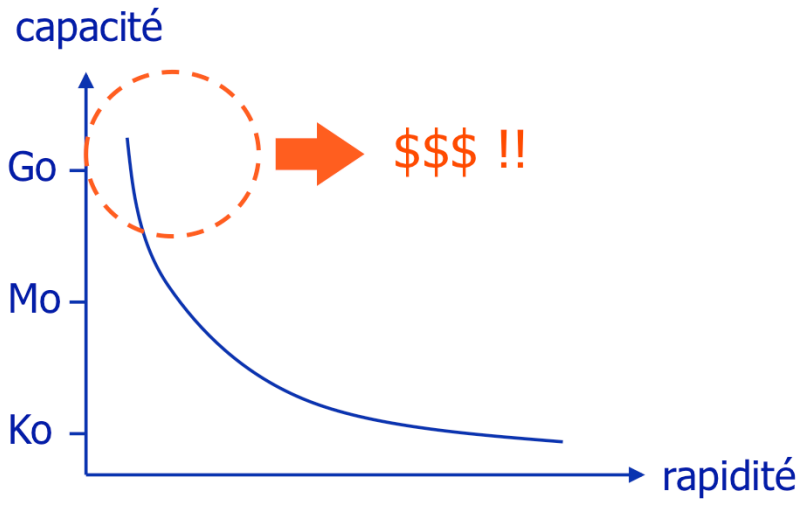
Remarque:
- Les mémoire vives sont souvent appelées "mémoires RAM". Ceci est un abus de langage car RAM signifie "Random Access Memory", c'est-à-dire "mémoire à accès aléatoire", en d'autres termes une mémoire dans laquelle on peut accéder directement à n'importe quelle donnée (par opposition aux mémoires à accès séquentiel).
-Les mémoires mortes sont souvent appelées "mémoires ROM". Ceci est plus logique puisque ROM signifie bien "read only memory".

Chapitre 12:
Petites et grandes mémoires

12.5.2 – Mémoires de masse

74

Pour les très grandes capacités (>>Go), les mémoires à SC sont trop chères…



75

...il faut donc utiliser d'autres technologies: mémoires de masse

HD
>1To

CD
650Mo

DVD
4,7 à 17Go

76

Les mémoires de masse utilisent d'autres technologies que l'électronique (semi-conducteurs) pour stocker l'information

Un cas particulier: la mémoire FLASH

77

On a vu apparaître ces derniers années un type particulier de mémoire qui, bien qu'étant réalisée à base de semi-conducteurs, possède de nombreux avantages typiques des mémoires de masse: les mémoires FLASH.

Les mémoires FLASH possèdent de nombreux avantages:
-une capacité qui dépasse aujourd'hui très largement le Go
-une grande robustesse (aucune pièce mobile)
-un encombrement très réduit