UTILISATION DES PORTES LOGIQUES

La dernière fois, nous avons appris comment fonctionnait une porte logique. Une grande question reste cependant en suspens : c'est bien joli tout ça, mais à quoi ça sert ??? Comme nous allons le voir par la suite, à tout plein de choses utiles (si, si, …).

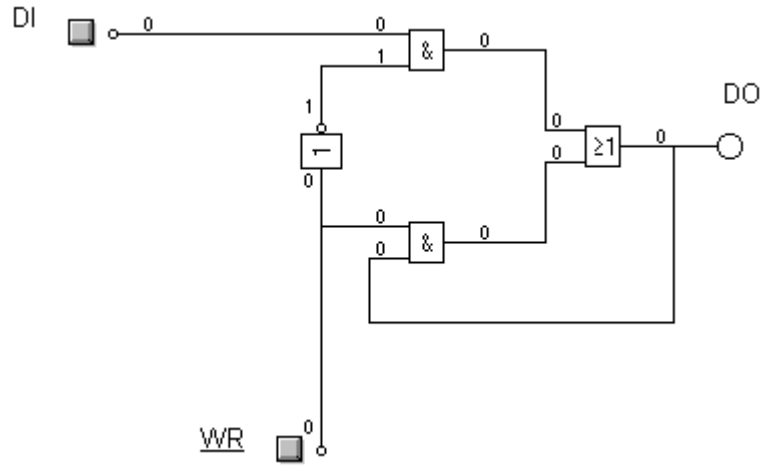
I./ La mémorisation des nombres binaires :

Une des opérations les plus importantes pour un ordinateur est la mise en mémoire des informations. Or nous avons vu que toutes les informations traitables par les moyens informatiques pouvaient être codées par des nombres binaires. Le problème est donc de trouver un moyen de stocker des nombres binaires. Nous allons voir que c'est possible grâce aux portes logiques.

N.B. : dans cette activité, nous allons étudier des circuits pouvant être assez complexes, et comportant pour certains, un grand nombre de portes logiques. Il est évident qu'il ne faut pas retenir par cœur ces circuits, ni leur fonctionnement détaillé, mais seulement leur principe de base.

1./ Circuit mémoire de capacité 1 bit :

Charger dans Crocodile Physics, le fichier « Mémoire RAM 1 bit.cyp ». Ce fichier, ainsi que tous les autres que nous utiliserons dans cette activité se trouvent dans le répertoire « Portes Logiques ». On se retrouve face au circuit suivant :



Les boutons DI et WR permettent d'entrer des données dans le circuit. Ils sont au niveau 0 lorsqu'ils ne sont pas enfoncés, au niveau 1 dans le cas contraire. La lampe témoin DO est au niveau 0 lorsqu'elle est éteinte, au niveau 1 lorsqu'elle est allumée (en rouge).

Etude du fonctionnement du montage :

* Mettre le bouton WR au niveau 0. Actionner le bouton DI et regarder comment est modifié l'état de la sortie DO. Que peut-on dire sur ce montage lorsque WR est au niveau 0 ?

* Alors que la sortie DO est au niveau 0, mettre le bouton WR au niveau 1. Actionner alors le bouton DI. Que remarque-t-on ?

* Faire la même chose lorsque la sortie DO est au niveau 1. Que remarque-t-on ?

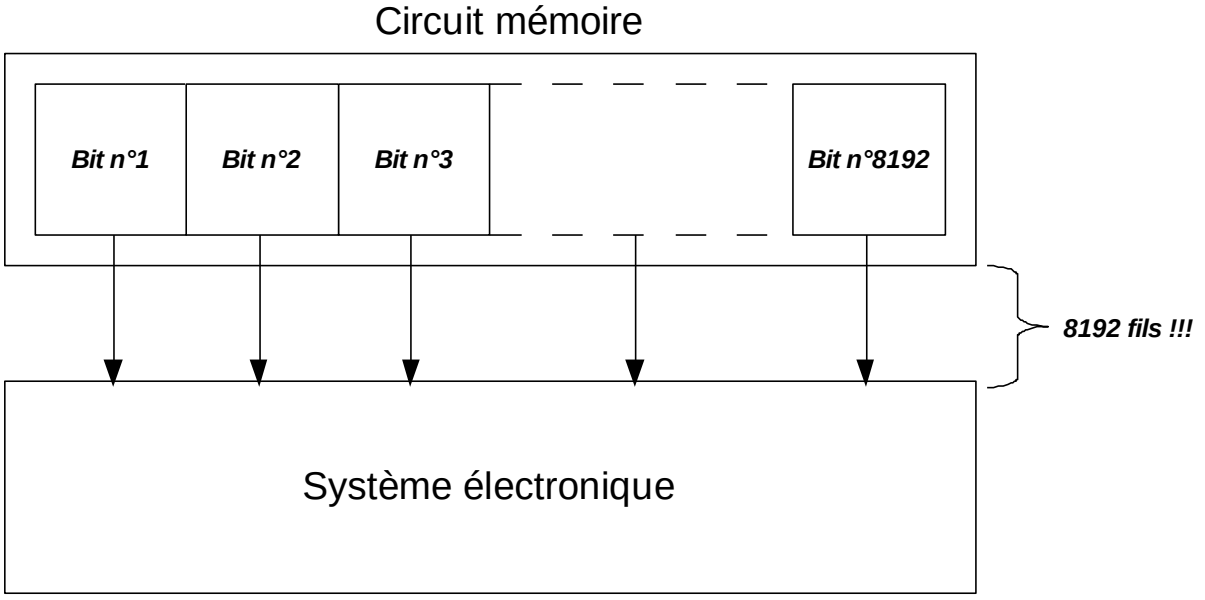
* En quoi ce montage est-il assimilable à une mémoire ?

DI correspond à « Data Input » : Entrée de donnée.
DO correspond à « Data Output » : Sortie de donnée.
WR correspond à « Write » : Ecriture. Le trait de soulignement indique que ce signal de commande est actif à l'état bas (c'est-à-dire au niveau 0) : on peut écrire dans la mémoire lorsque WR = 0.

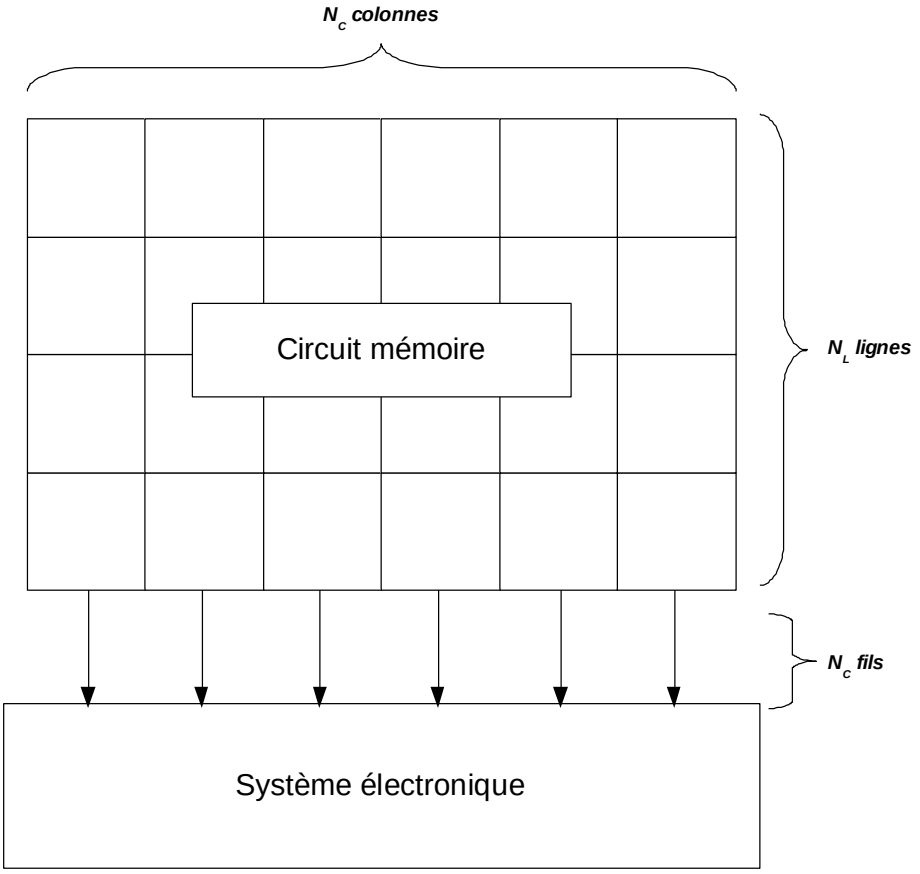
Ok, nous avons vu comment réaliser une mémoire de capacité 1 bit… C'est bien, mais ce n'est vraiment pas beaucoup, quand on pense que la mémoire RAM d'un PC actuel est d'au moins 1 Go (giga-octet). Comment peut-on faire une mémoire d'une telle capacité ? Tout simplement en mettant côte à côte de très nombreuses unités de stockage 1 bit comme celle que nous venons voir (il en faut un peu plus d'un milliard pour 1 Go). C'est pourquoi la capacité des mémoires des ordinateurs est directement reliée au niveau technologique : au début des années 1980, il aurait fallu avec les technologies de l'époque une grosse armoire pour réaliser une mémoire de 1 Go, alors qu'actuellement on trouve des clefs USB de plusieurs Go microscopiques… Ces clefs USB sembleront peut-être ridiculement limitées dans 30 ans… Wait and see…

2./ Un problème embarrassant…

Imaginons une mémoire de capacité très modeste, par exemple 1 ko, soit 1024 octets. Cela représente 8192 bits. Les informations stockées dans cette mémoire sont destinées à commander un système électronique. Par conséquent, il faut que ce système puisse accéder à chacun des bits stockés dans cette mémoire. Il faut donc qu'il y ait une liaison physique (un fil) entre chacune des unités de stockage 1 bit (comme celles que l'on a vu précédemment) et le système électronique à commander :



On s'aperçoit que le nombre de connections entre la mémoire et le circuit électronique est trop élevé pour que la réalisation pratique soit faisable. Et nous sommes parti d'une mémoire de capacité ridiculement faible. Pour une mémoire de 1 Go, il faudrait plus de 8 milliards de fils !!! Si on regarde bien une barrette de RAM de 1 Go, on s'aperçoit que le nombre de connecteurs est bien inférieur à 8 milliards (pour ceux qui ne me croient pas, comptez le nombre de connections quand vous aurez ce type de barrettes entre les mains ;-). Comment résoudre ce problème assez ennuyeux ? Une solution est de ne permettre l'accès à un instant donné qu'à un nombre limité de bits de la mémoire. On décompose la mémoire en N_L lignes de N_C colonnes et on ne permet au système extérieur d'accéder qu'à N_C bits :



Avec ce système, il ne faut plus que N_C fils de connexion entre le circuit mémoire et le monde extérieur. Dans le cas de notre mémoire de 1 ko, on peut choisir, par exemple, $N_C = 8$ et $N_L = 1024$. Il ne faut plus que 8 fils au lieu de 8192.

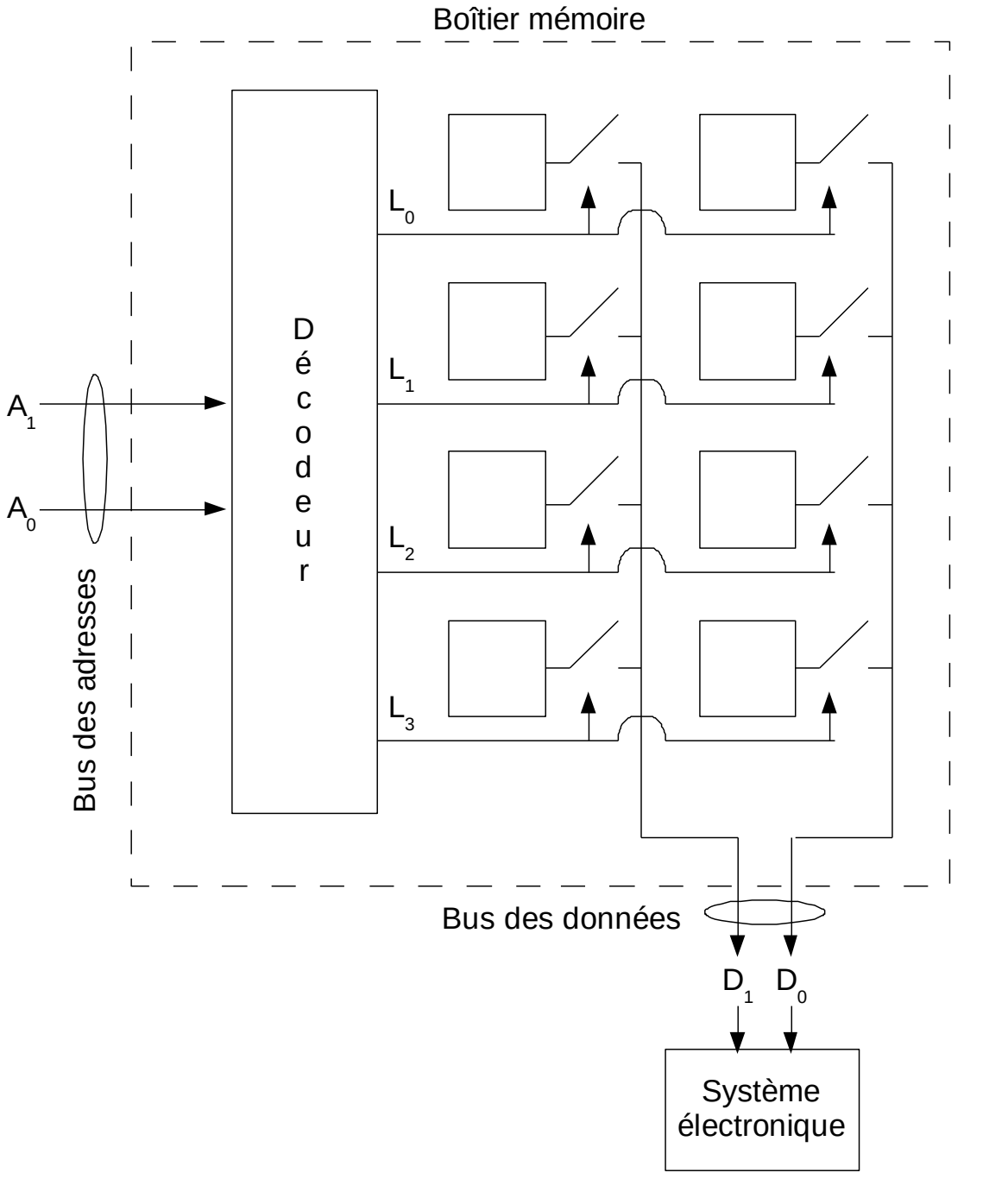
Nous avons résolu un problème, mais nous en avons créé un autre : comment sélectionner une ligne particulière parmi les N_L qui constituent la mémoire ? On pourrait relier chaque ligne à un fil pour alimenter la ligne dont on souhaite se servir : pour cela il nous faut 1024 fils, plus les 8 de départ, ça nous fait 1032 fils. C'est moins que 8192, mais c'est loin d'être satisfaisant. Appelons à notre aide, le système binaire. Combien de bits faut-il pour écrire 1024 valeurs ? Réponse : 10 bits (voir les épisodes précédents). Cela veut donc dire qu'avec 10 fils, on peut transmettre un nombre de 0 à 1023. Un circuit spécialisé, appelé *décodeur*, qui est intégré au circuit mémoire, va décoder ce nombre à 10 bits et relier la ligne correspondante au système électronique. Il ne nous faut plus que 18 fils :

- 10 fils pour sélectionner (on dit aussi *adresser*) une ligne particulière parmi les 1024 lignes constituant la mémoire. Cet ensemble de 10 fils est appelé le *bus des adresses* (ADRESS BUS).
- 8 fils qui relient le monde extérieur à la ligne sélectionnée. Cet ensemble de 8 fils est appelé le *bus des données* (DATA BUS).
- A ces 18 fils, il nous faut ajouter 2 fils pour l'alimentation électrique, plus quelques fils supplémentaires pour divers signaux de commande (écriture, lecture, sélection de boîtier, etc.). Ces fils constituent le *bus de commande* (CONTROL BUS). Nous ne nous occuperons pas de ce bus.

En définitive, le boîtier de mémoire de 1 ko comporte un peu plus d'une vingtaine de fils, ce qui est nettement plus satisfaisant que 8192, ou même 1032. Un peu compliqué ? Certes… C'est pourquoi nous allons raisonner sur un exemple plus simple : une mémoire de 8 bits, dont l'organisation est la suivante : $N_C = 2$ colonnes et $N_L = 4$ lignes. Pour adresser 4 lignes, il nous suffit de deux fils (car un nombre de 2 bits permet de coder 4 nombres différents). Le bus des adresses est donc constitué de deux fils (A_0 et A_1). $N_C = 2$, donc le bus de données comporte lui aussi deux fils (D_0 et D_1). Sur le schéma suivant les interrupteurs munis d'une petite flèche sont des interrupteurs commandés électriquement (des transistors en fait) qui se ferment lorsque la ligne correspondante est au niveau 1. Par exemple, si L_1 est au niveau 1, les deux cases de la deuxième ligne de la mémoire sont reliées au

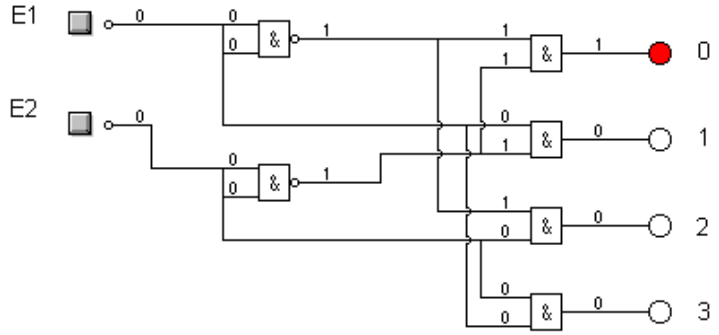
bus de données. Le système extérieur a donc accès aux 2 bits stockés dans la deuxième ligne. Si c'est L_3 qui est au niveau 1, le système extérieur a accès à la quatrième ligne. On sélectionne la ligne L_0, L_1, L_2 ou L_3, en fonction du niveau de A_0 et A_1. Il faut que la table de vérité du circuit décodeur soit la suivante :

A_1	A_0	L_0	L_1	L_2	L_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



3./ Réalisation du circuit décodeur :

Charger dans Crocodile Physics, le fichier « Décodeur 2 – 4.cyp ». On se retrouve face au circuit suivant :



Relever l'état des sorties 0, 1, 2 et 3 en fonction de l'état des entrées E1 et E2, et consigner les résultats dans un tableau :

E2	E1	Sortie 0	Sortie 1	Sortie 2	Sortie 3
0	0				
0	1				
1	0				
1	1				

* Que remarque-t-on ?

* Que peut-on en conclure ?

4./ Synthèse :

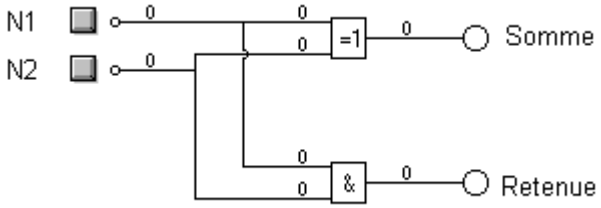
Si on reprend le schéma de la mémoire de 8 bits de la page précédente, on voit que le circuit décodeur peut être réalisé par un ensemble de portes logiques. Chaque case mémoire est en fait une unité de stockage de 1 bit, que nous savons réaliser avec des portes logiques. Les portes logiques permettent donc de réaliser tout circuit mémoire. On a fait le raisonnement sur un circuit mémoire de petite capacité, mais le principe est le même pour une barrette de 1 Go. On peut juste imaginer que le circuit sera très complexe, car comportant plusieurs centaines de millions de portes logiques.

II./ Les opérations de calcul :

C'est bien beau de stocker des nombres binaires en mémoire, mais c'est encore mieux de pouvoir les utiliser. Mais que faire avec des nombres ? Des calculs, pardi ! Là aussi, nous allons voir que les portes logiques vont nous permettre de réaliser des calculs.

1./ Le demi additionneur :

Charger le fichier « Demi-additionneur.cyp » dans Crocodile Physics. On se retrouve avec le circuit suivant :



* Etablir la table d'addition pour le système binaire :

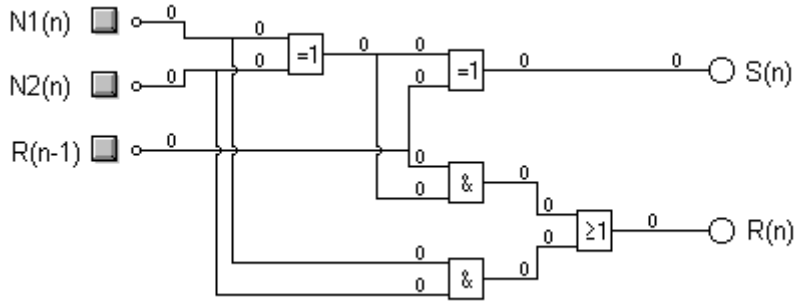
* Vérifier que le montage effectue bien la somme des deux nombres binaires d'un bit N_1 et N_2.

Pour additionner des nombres de plus d'un bit, on peut imaginer qu'il suffit d'utiliser plusieurs montages comme celui-ci combinés entre eux. Le problème est que ce montage ne tient pas compte d'une retenue éventuelle. Il ne fait donc le travail qu'à moitié, d'où son nom de demi additionneur. Voyons maintenant un circuit plus complet.

2./ L'additionneur :

Soient deux nombres binaires N_1 et N_2 à additionner. On note $N_1(n)$ le $n^{ième}$ bit du nombre N_1, et $N_2(n)$ le $n^{ième}$ bit du nombre N_2. S(n) est le $n^{ième}$ bit du nombre représentant la somme de N_1 et N_2. R(n) est la retenue éventuelle e l'addition de $N_1(n)$ et de $N_2(n)$. R(n-1) est la retenue éventuelle de l'addition des bits précédents, $N_1(n-1)$ et $N_2(n-1)$.

Soit le montage suivant :

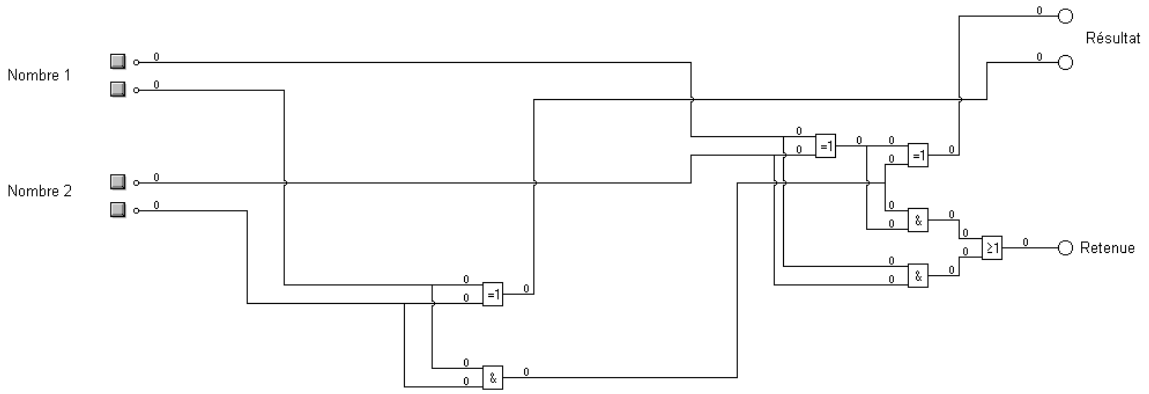


* Etablir une table de vérité de ce montage :

* Vérifier que cet additionneur tient bien compte de la présence d'une retenue éventuelle :

3./ Un exemple concret : un circuit pouvant additionner deux nombres de 2 bits :

Dans Crocodile Physics, charger le fichier « Additionneur 2 bits.cyp ». On se retrouve avec le circuit suivant :



A gauche se trouve quatre boutons qui permettent de rentrer deux nombres binaires comportant chacun 2 bits. Pour chaque nombre, le bouton du haut représente le bit de gauche et le bouton du bas représente le bit de droite.

* Remplir la table d'addition suivante, portant sur des nombres de deux bits :

+	00	01	10	11
00				
01				
10				
11				

* Vérifier que le montage précédent donne bien le résultat correct de l'addition des deux nombres de deux bits.

* Reconnaître dans ce circuit complexe les deux unités plus simples vues précédemment. Les entourer de deux couleurs différentes sur le schéma. Comment se fait la liaison entre ces deux sous unités ?

4./ Les autres opérations :

Faire une *soustraction* revient à additionner l'opposé. Pour la *multiplication*, il suffit de faire une suite d'additions : par exemple, on a : $4 \times 5 = 4 + 4 + 4 + 4 + 4$. La *division* revient à faire une multiplication par l'inverse d'un nombre. Les autres *fonctions mathématiques* (racine carrée, sinus, cosinus, tangente, etc.) peuvent toutes se ramener à une série plus ou moins complexe des quatre opérations de base (ce que l'on appelle des algorithmes). *Conclusion :* en connaissant la méthode pour additionner deux nombres, on peut faire toutes les opérations possibles (mais les circuits peuvent devenir monstrueusement complexes).

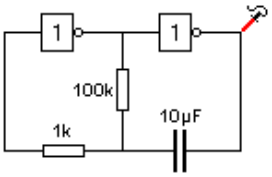
III./ Circuit horloge :

Nous avons vu les deux actions que savait faire un microprocesseur :

- Lire ou écrire des nombres en mémoire.
- Effectuer des calculs sur ces nombres.

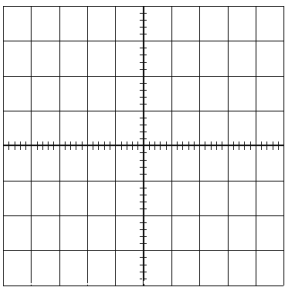
Pour savoir quelles opérations effectuer, et dans quel ordre, le microprocesseur suit un *programme*, constitués *d'instructions*. Comment le microprocesseur sait-il qu'il est temps de passer à l'instruction suivante ? Grâce à son *horloge* : c'est une sorte de métronome qui envoie des impulsions électriques à intervalles réguliers. Chaque fois que le microprocesseur reçoit une impulsion, il passe à l'instruction suivante. Le nombre d'impulsions que l'horloge fournit en une seconde est appelé la fréquence d'horloge (abusivement la fréquence du microprocesseur) est s'exprime en hertz (Hz). Là aussi, les progrès de la technologie ont permis de fabriquer des composants de plus en plus rapides, et on est passé en 20 ans de quelques MHz (10^6 Hz) à quelques GHz (10^9 Hz). Comment réaliser ce type d'horloge ? Grâce à des portes logiques.

* Charger dans Crocodile Physics le fichier « Horloge.cyp ». On se retrouve face au circuit suivant :



Lancer la simulation et regarder l'allure de la tension de sortie, visualisée par la sonde rouge.

* Représenter sur le diagramme suivant l'allure de la courbe représentative de la tension de sortie :



* Quelles sont les caractéristiques de cette tension ?

* Quelle est la valeur de la période de cette tension (exprimée en secondes) ?

* En faisant varier les valeurs de la résistance de 100 kΩ et du condensateur de 10 µF (F = Farad), vérifier la relation suivante : $T \approx 2{,}2 \times R \times C$, où T est la période du signal délivré par l'horloge (en s), R est la valeur de la résistance (en Ω), et C est la valeur du condensateur (en F).

* En se servant de la formule précédente, déterminer la valeur de R et de C pour que T soit égale à 1 seconde. N'y a-t-il qu'une seule solution ? Modifier le circuit en conséquence et vérifier sur le graphique que la valeur de la période est correcte.

R = ***C*** =