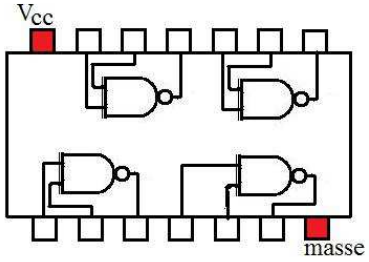
VII- Circuits combinatoires élémentaires

Par circuit combinatoire, on entend que ses sorties sont des fonctions de ses entrées. Cela par opposition aux circuits séquentiels, que nous verrons plus loin, où les sorties dépendent aussi d'évènements passés. Pour un circuit combinatoire, chaque évènement dans les entrées – un certaine combinaison de 0 et de 1 – donne un résultat en sortie. C'est ce que montre la table de vérité du circuit, qui prend tous les cas possibles en entrée comme nous l'avons vu au chapitre précédent.

Un circuit intégré est en général formé d'une plaquette de silicium intégrant les portes du circuit, elle-même encapsulée dans un boîtier, d'où sortent des broches, ou des pattes assurant les connexions électriques d'entrée, de sortie, et d'alimentation. A cause de son aspect, on l'appelle une puce. Selon la densité du nombre de transistors et de portes qu'il possède, on distingue plusieurs catégories de produits :

* SSI (*small scale integration*) avec moins de 10 portes par circuit.
* MSI (*medium scale integration*) avec quelques dizaines des portes
* LSI (*large scale integration*) avec quelques centaines ou milliers de portes, et jusqu'à 100 000 portes.
* VLSI (*very large scale integration*) avec plus de 100 000 portes par circuit, et souvent des millions.



Un cas classique de circuit SSI consiste en quatre portes NON ET indépendantes, avec pour chacune deux entrées et une sortie. Avec l'alimentation électrique qui demande une broche mise à la masse (0 volt) et une broche avec V_{cc} =5 volts, cela fait un total de 14 broches pour seulement 4 portes logiques.

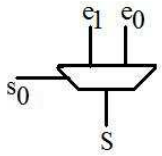
Si l'on faisait de même avec un circuit VLSI de 5 millions de portes, il faudrait 15 millions de broches, et comme le précise A. Tanenbaum [1] : « Avec un espacement standard de 2,3 mm entre deux broches, il mesurerait 18 km de long » ! Aussi doit-on faire en sorte de maximiser le nombre de portes par rapport au nombre de broches. C'est ce que nous allons voir dans des cas simples de circuits MSI que nous allons maintenant étudier, à savoir les multiplexeurs, décodeurs, comparateurs, additionneurs …

1) Le multiplexeur

On a 2^n entrées, *n* fils de sélection qui sont aussi des entrées, et une seule sortie. Quel est le rôle du multiplexeur ? Il sélectionne l'une des 2^n entrées pour l'envoyer en sortie. En effet chacune des 2^n entrées est numérotée par un nombre en binaire de longueur *n*. Le choix se fait par le mot de longueur *n* des fils de sélection, et c'est l'entrée correspondant à ce mot qui passe en sortie. Cela s'appelle plus précisément un multiplexeur un parmi 2^n.

[1] A. Tanenbaum, Architecture de l'ordinateur, éditions Dunod, 2001.

1) Le multiplexeur 1 parmi 2 : il a deux entrées e_0 e_1 et un fil de sélection s_0 pour choisir l'une des deux entrées et la mettre en sortie S. Il est ainsi représenté :



On a la table de vérité :

s_0	e_0	e_1	S
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

On en déduit l'équation :

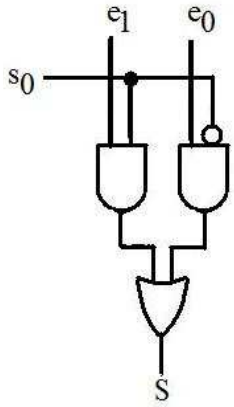
$$S = \overline{s_0} e_0 \overline{e_1} + \overline{s_0} e_0 e_1 + s_0 \overline{e_0} e_1 + s_0 e_0 e_1$$
$$= \overline{s_0} e_0 (\overline{e_1} + e_1) + s_0 e_1 (\overline{e_0} + e_0)$$
$$= \overline{s_0} e_0 + s_0 e_1$$

On peut retrouver cette simplification grâce au tableau de Karnaugh :

s_0 \ $e_0 e_1$	00	01	11	10
0	0	0	1	1
1	0	1	1	0

d'où $S = \overline{s_0} e_0 + s_0 e_1$

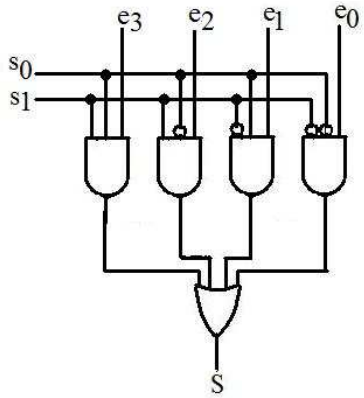
On en déduit le schéma électronique :



On a donc besoin de trois portes.

2) Le multiplexeur un parmi 4

Pour les mêmes raisons que précédemment, on aboutit au schéma suivant :



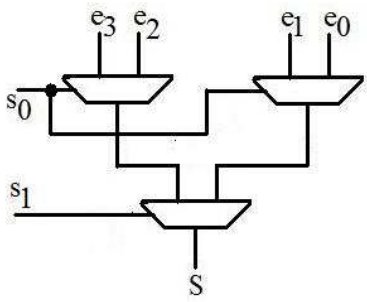
Par exemple ce qui passe dans le fil e_1, correspondant au numéro 01, et qui est envoyé en sortie, est obtenu avec s_0=1 et s_1=0, étant entendu que la sélection s'écrit s_1 s_0 dans cet ordre.

Le multiplexeur 1 parmi 4 demande 9 portes

Exercice 1:

1) En s'inspirant de ce qui précède, combien de portes NON comporte un multiplexeur 1 parmi 2^n ? Et combien de portes au total ?

2) Vérifier que le multiplexeur 1 parmi 4 peut aussi être obtenu avec trois multiplexeurs 1 parmi 2 de la façon suivante :



On en déduit un procédé récursif pour construire un multiplexeur 1 parmi 2^n.

Ainsi pour avoir un multiplexeur 1 parmi 8, il faudrait deux multiplexeurs 1 parmi 4 et un multiplexeur 1 parmi 2. Dans ce cas, combien faudrait-il de multiplexeurs 1 parmi 2 ? Et combien de portes ?

En déduire le nombre de portes qu'il faudrait pour un multiplexeur 1 parmi 2^n construit selon ce procédé. Comparer avec le nombre de portes qu'il faudrait si on le construisait par la méthode du 1).

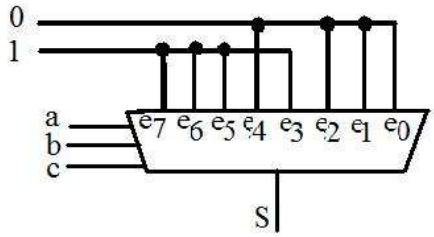
Exercice 2 : faire le schéma d'un démultiplexeur 2 bits : il a une entrée, deux fils de sélection, et 4 sorties. C'est l'inverse du multiplexeur 1 parmi 4. Suivant la sélection correspondant à un certain numéro (de longueur 2), il envoie ce qui est dans l'entrée sur la sortie qui porte ce numéro.

Intérêt du multiplexeur

* La conversion parallèle-série. Imaginons une instruction codée sur huit bits qui arrive en parallèle sur les huit lignes d'un multiplexeur. Si on lâche séquentiellement les nombres de 000 à 111 sur les trois lignes de sélection, les huit bits passent dans la ligne de sortie *S* l'un après l'autre. C'est notamment intéressant lorsqu'il s'agit de passer des informations de l'ordinateur vers une ligne téléphonique.

* Une application du multiplexeur parmi d'autres est la fabrication de la fonction majoritaire, vue précédemment. Reprenons cette fonction lorsqu'elle a trois entrées *a*, *b*, *c*, et qui sort 1 si deux au moins des trois entrées sont à 1, et 0 sinon. Le 1 sort lorsque les nombres *abc* sont en binaire 011, 101, 110 ou 111, soit 3, 5, 6 ou 7. Le 0 sort lorsque les nombres sont 0, 1, 2, 4. Il suffit alors

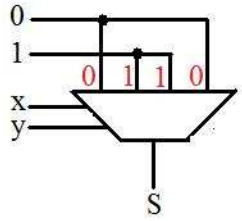
d'utiliser un multiplexeur 1 parmi 8, avec *a*, *b* et *c* comme lignes de sélection, et de mettre les entrées 0, 1, 2, 4 à 0, et les autres entrées à 1.



* L'exemple précédent se généralise à toute fonction logique dont on connaît la table de vérité. Il suffit d'envoyer en entrées les 0 et les 1 obtenus dans la colonne des résultats dans l'ordre où ils apparaissent, et d'utiliser comme sélections les entrées de la table. Par exemple, avec cette table de vérité associée à l'équation $S = x\overline{y} + \overline{x}y$:

x	*y*	*S*
0	0	0
0	1	1
1	0	1
1	1	0

on peut construire ce circuit avec un multiplexeur 4 bits :



Exercice 3: Circuit testant la parité du nombre de 1 dans un mot

A l'entrée du circuit se trouvent *n* entrées, ce qui correspond à un mot de *n* bits. On veut qu'à la sortie on ait 1 si le nombre de bits à 1 dans le mot d'entrée est impair, et 0 sinon.

1) On commence par *n* = 2. Faire la table de vérité du circuit, avec *a* et *b* en entrées et *S* en sortie. Puis donner l'équation associée. Quelle porte classique trouve-t-on ?

2) On passe à *n* = 3. Faire la table de vérité, donner l'équation de ce circuit, et dessiner le schéma correspondant en utilisant deux portes XOR. On admettra que le XOR est associatif et l'on rappelle que le XOR est la porte de la discorde.

3) On fait *n* = 4. Donner la table de vérité, en déduire l'équation du circuit. Puis en s'aidant des questions précédentes, et en utilisant $a \oplus b$ et $c \oplus d$, montrer que le circuit peut être constitué avec trois portes XOR, et dessiner le schéma correspondant.

4) En généralisant, combien de portes XOR faut-il pour un circuit ayant un mot de *n* bits en entrée ?

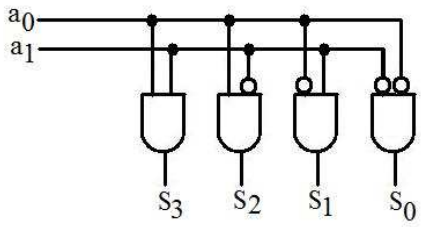
2) Le décodeur

Un décodeur *k* bits est un circuit à *k* entrées et 2^k sorties. Les *k* entrées forment un nombre en binaire de longueur *k*. Le décodeur fait en sorte que la sortie qui porte le numéro correspondant au nombre en entrée passe à 1 (elle est activée) tandis que toutes les autres sont à 0 (inactives). Traitons le cas du décodeur deux bits, *k* = 2. Sa table de vérité est la suivante :

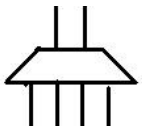
*a*0	*a*1	*S*0	*S*1	*S*2	*S*3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Les équations du circuit s'en déduisent : $S_0 = \overline{a_0}\,\overline{a_1}$, $S_1 = \overline{a_0}\,a_1$, $S_2 = a_0\,\overline{a_1}$, $S_3 = a_0\,a_1$

On en déduit son schéma électronique :



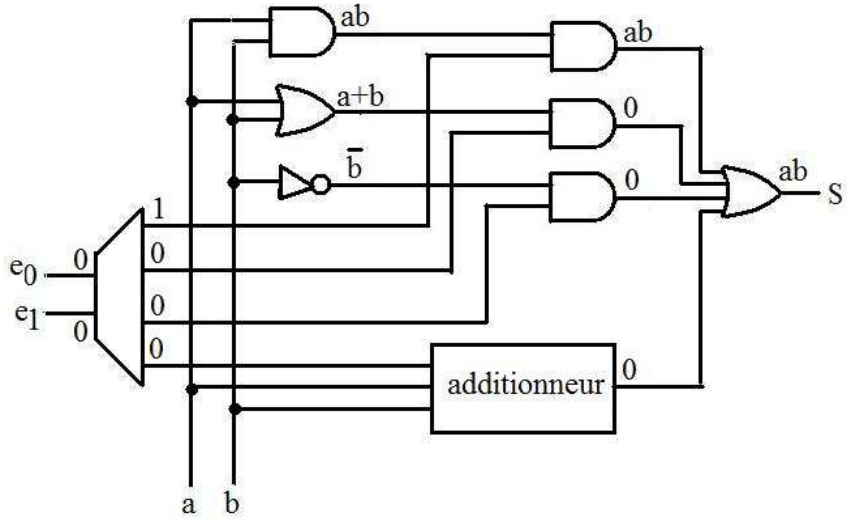
ainsi dessiné symboliquement



Intérêt des décodeurs

* Un décodeur est un dispositif essentiel à l'entrée de l'unité logique et arithmétique (UAL) de l'unité centrale de l'ordinateur. Nous allons donner une version très simplifiée de cette unité logique et arithmétique en supposant qu'elle est capable d'exécuter quatre opérations : le ET, le OU, le NON et l'addition, à partir de deux nombres binaires *a* et *b* de un bit chacun. A la sortie on aura le résultat de l'opération concernée sur un bit. Il s'agit d'une UAL de 1 bit.

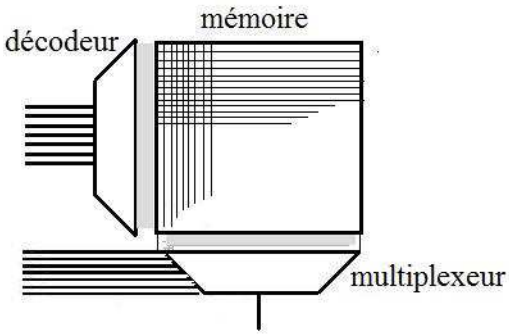
Rappelons-nous : l'unité de commande de l'unité centrale recueille dans un registre le nombre codé associé à une opération à réaliser, et portant sur les deux opérandes *a* et *b*. Elle se charge de décoder cette opération pour qu'elle soit dirigée vers l'opération concernée dans l'UAL. C'est là qu'intervient le décodeur. Le code à deux chiffres de l'instruction entre dans le décodeur par les fils e_0 et e_1. Le décodeur a quatre fils en sortie, et selon l'opération à réaliser un de ces fils est activé (mis à 1) tandis que les autres restent à 0. On en déduit ce schéma de l'UAL :



Nous avons pris l'exemple où l'instruction 00 placée dans e_0 e_1 est le code du ET, c'est le câblage qui le veut. Sur les quatre lignes d'activation à la sortie du décodeur, la ligne 0 est activée, elle est à 1, tandis que les autres restent à 0. Seul le résultat *ab* traverse les portes jusqu'à la sortie. Pour l'additionneur, nous laissons son schéma en attente, nous le verrons plus tard.

* Une autre application des décodeurs se situe à l'entrée de la mémoire principale. Imaginons une mémoire d'un Mo, c'est-à-dire $10^6 = (10^3)^2 = (2^{10})^2 = 2^{20}$ octets. En supposant qu'un octet occupe une cellule mémoire, il faudrait utiliser 2^{20} fils d'adresses pour pouvoir accéder à n'importe quelle cellule mémoire. Cela est impensable.

Aussi met-on en place une autre organisation, avec un décodeur et un multiplexeur qui vont permettre de réduire le nombre de fils de transfert d'adresses à l'entrée de la mémoire. D'abord la mémoire est rendue rectangulaire, avec des lignes et des colonnes, par exemple avec 2^{10} colonnes et autant de lignes. Chaque cellule a son adresse définie par la ligne et la colonne où elle se trouve. D'autre part on utilise 20 fils où passent les adresses, une adresse ayant pour longueur 20, soit $a_{19}\ a_{18}\ \ldots\ a_1\ a_0$. Ces adresses sont ensuite coupées en deux : $a_{19}\ a_{18}\ \ldots\ a_{10}$, et $a_9\ a_8\ \ldots\ a_1\ a_0$. La demi-adresse $a_9\ a_8\ \ldots\ a_1\ a_0$ est envoyée sur les 10 lignes d'entrée d'un décodeur qui les dispatche en sortie sur les 2^{10} lignes de la mémoire rectangulaire (ici carrée). On accède ainsi à la ligne où se trouve la cellule concernée. Reste à trouver sa colonne, ce qui donnera la position exacte de la cellule dont on pourra envoyer le contenu en sortie. Pour cela on utilise un multiplexeur avec 10 fils de sélection contenant l'autre moitié de l'adresse, soit $a_{19}\ a_{18}\ \ldots\ a_{10}$ correspondant à l'adresse de la colonne concernée. Ce multiplexeur a aussi 2^{10} lignes d'entrée provenant des 2^{10} colonnes de la mémoire, permettant grâce aux fils de sélection d'atteindre la colonne où se trouve la cellule concernée, et d'envoyer dans le fil de sortie le contenu de la cellule.

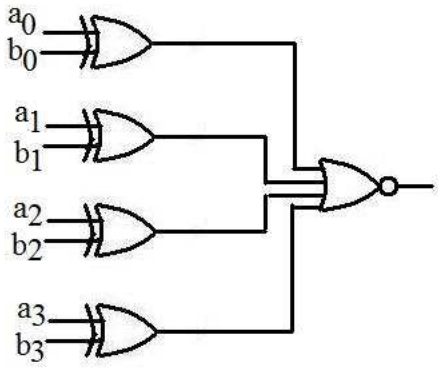


3) Le comparateur

Ce circuit logique prend deux nombres *A* et *B* de *n* bits chacun en entrée, soit 2*n* entrées. La sortie est à 1 si *A* est égal à *B*, et 0 sinon.

Si les deux nombres ont un bit chacun, il suffit de faire jouer la porte de la discorde. On sait que $a \oplus b$ donne 0 si *a* et *b* sont différents et 1 s'ils sont égaux. La négation de ce résultat est ce qui est attendu en sortie.

Cela se généralise à deux nombres ayant un nombre quelconque de bits. Par exemple pour $n = 4$, les deux nombres A et B s'écrivent $a_0\ a_1\ a_2\ a_3$ et $b_0\ b_1\ b_2\ b_3$. On envoie les 4 chiffres respectifs de chacun des deux nombres sur quatre portes XOR. Il y aura égalité de *A* et *B* si et seulement si les quatre portes sortent un 0. On envoie alors ces résultats dans une porte NON OU, qui sort 1 en cas d'égalité de *A* et *B*, et 0 sinon.



Exercice 4 : Conception d'un circuit logique

Ce circuit comporte *k* entrées notées a_1, a_2, …, a_k, et une sortie *S*. Appelons s le nombre d'entrées égales à 1. Le nombre *s* est compris entre 0 et *k*. On veut que la sortie *S* ramène 0 si $s = 0$, et sinon qu'elle ramène a_s , c'est-à-dire la valeur de l'entrée numéro *s* lorsque *s* est compris entre 1 et *k*.

Pour traiter ce problème, on commence par les cas les plus simples.

1) $k = 1$. On a comme table de vérité

*a*1	*s*	*S*
0	0	0
1	1	1

d'où l'équation $S = a_1$.

2) $k = 2$. La table de vérité est encore très simple :

*a*1	*a*2	*s*	*S*
0	0	0	0
0	1	1	0
1	0	1	1
1	1	2	1

d'où $S = a_1$.

3) $k = 3$.

$a1$	$a2$	$a3$	s	S
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	2	1
1	0	0	1	1
1	0	1	2	0
1	1	0	2	1
1	1	1	3	1

On en déduit l'équation : $S = \overline{a_1}\, a_2\, a_3 + a_1\, \overline{a_2}\, \overline{a_3} + a_1\, a_2\, \overline{a_3} + a_1\, a_2\, a_3$

Simplification :

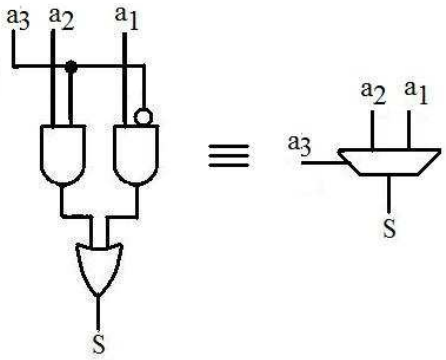
$$S = \overline{a_1}\, a_2\, a_3 + a_1\, a_2\, a_3 + a_1 \overline{a_3}\, \overline{a_2} + a_1\, \overline{a_3}\, a_2$$
$$= (\overline{a_1} + a_1)\, a_2\, a_3 + a_1 \overline{a_3} (\overline{a_2} + a_2)$$
$$= a_2\, a_3 + a_1 \overline{a_3}$$

Cela se retrouve aussi par le tableau de Karnaugh

a3 \ a1 a2	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$S = a_2\, a_3 + a_1 \overline{a_3}$

Le dessin du circuit en découle



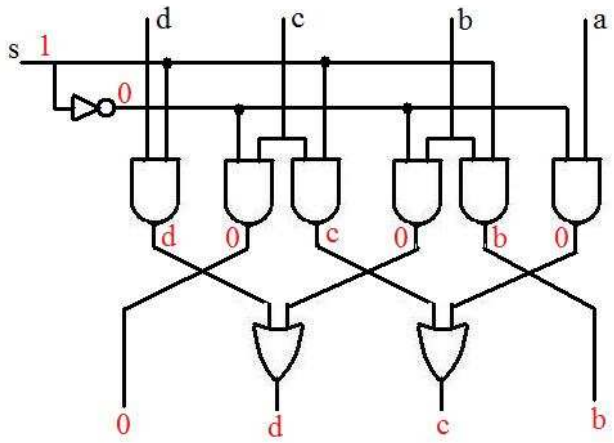
On reconnaît le schéma d'un multiplexeur deux bits, avec a_1 et a_2 en entrée, et a_3 qui joue le rôle du sélectionneur.

4) $k = 4$. A faire.

Nous allons maintenant passer aux circuits proprement arithmétiques. Comme nous l'avons vu, toutes les opérations réalisées en binaire, qu'il s'agisse d'additions, de soustractions ou de multiplications, se font avec des additions, avec en plus des décalages quand on pratique une multiplication ou une division. C'est notamment le cas pour les multiplications ou les divisions par 2 qui sont des décalages. Des additions particulières, et très courantes, consistent à ajouter ou retrancher 1, elles sont appelées incrémentation ou décrémentatation, l'incrémentation jouant notamment dans le compteur de programme (PC ou compteur ordinal). Deux circuits arithmétiques se dégagent de cela : le décaleur et l'additionneur.

4) Le décaleur

On a un mot avec *n* bits en entrées, ainsi qu'une entrée *sens* notée *s*, et *n* sorties. Si *s* = 1, il se produit un décalage du mot d'un cran vers la droite. Par exemple pour *n* = 4, le mot *d c b a* en entrée devient *0 d c b*. Si *s* = 0, le décalage se fait vers la gauche, le mot *d c b a* devient *c b a 0* en sortie. Sauf aux deux extrémités du mot où l'on n'utilise qu'une porte ET, les bits intermédiaires ont chacun deux portes ET, l'une étant à 0 à sa sortie et l'autre laissant passer le bit, comme indiqué sur le dessin dans le cas où *s* = 1 :



5) L'additionneur

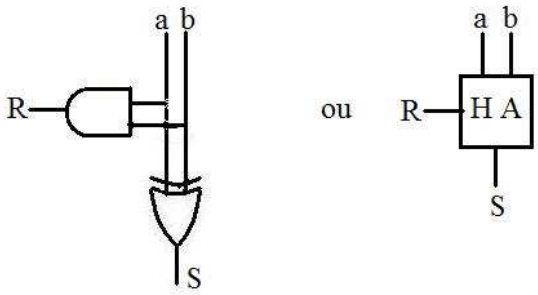
Commençons par l'addition de deux bits *a* et *b* en entrée, avec en sortie la somme *S* et la retenue *R*. Cela s'appelle le demi-additionneur, parce qu'il ne tient pas compte de la retenue qui peut aussi arriver en entrée, provenant de calculs précédents.

a) Le demi-additionneur

Sa table de vérité s'écrit :

a	b	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On en déduit les équations $S = a \oplus b$ et $R = ab$, d'où le schéma du circuit, avec à sa droite son dessin simplifié, HA signifiant *half adder* (demi-additionneur en anglais) :



b) L'additionneur un bit complet

On tient maintenant compte de la retenue *r* provenant éventuellement d'un calcul en amont. L'additionneur complet a trois entrées *a*, *b* et *r*, ainsi que deux sorties *S* et *R*. Avec comme table de vérité :

a	b	r	S	R
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

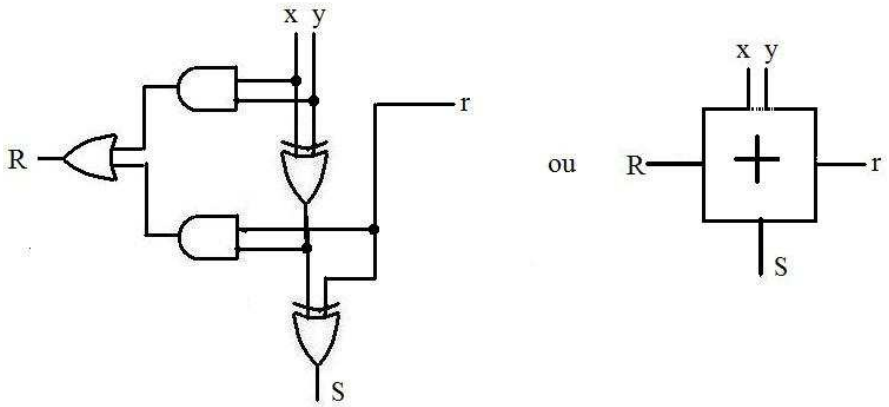
Cela donne comme équations :

$$S = \bar{a}\bar{b}r + \bar{a}b\bar{r} + a\bar{b}\bar{r} + abr$$
$$= \bar{r}(\bar{a}b + a\bar{b}) + r(ab + \bar{a}\bar{b})$$
$$= \bar{r}(a \oplus b) + r(\overline{a \oplus b})$$
$$= r \oplus (a \oplus b) = a \oplus b \oplus r$$
$$R = \bar{a}br + a\bar{b}r + ab\bar{r} + abr = ab + ar + br$$

Pour les simplifications, nous avons utilisé des formules liées au XOR dans *S*, et nous avons reconnu dans *R* la fonction majorité (*cf. chapitre 5*). Cela permet de construire le circuit additionneur, mais on peut encore le simplifier en utilisant la formule

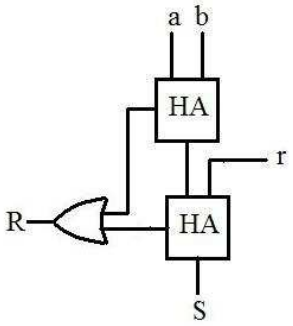
$$R = ab + r(a \oplus b)\,.$$

On pourra vérifier la validité de cette formule en prenant sa table de vérité et en la comparant avec celle de $R = ab + ar + br$.



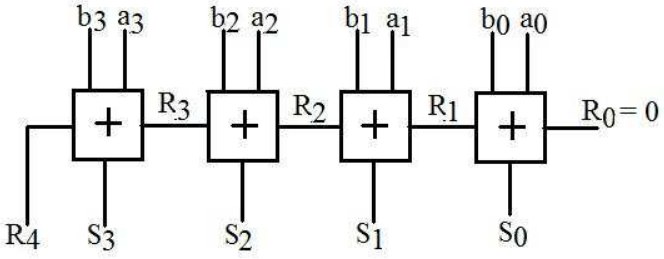
Exercice 5 : Faire le circuit de l'additionneur complet en utilisant deux demi-additionneurs

La somme de *a* et *b* obtenue par le premier semi-additionneur est envoyée dans le deuxième avec en même temps la retenue d'entrée *r*. Il en sort la somme finale, et les deux retenues sorties des deux demi-additionneurs passent par une porte OU pour donner finalement la retenue finale *R*.



c) L'additionneur *n* bits par propagation de la retenue

La méthode manuelle pour additionner deux nombres se traduit par un circuit où l'on met des additionneurs 1 bit en série, la retenue de sortie de l'un devenant la retenue d'entrée du suivant, ce qui correspond à la propagation de la retenue de droite à gauche. Par exemple un additionneur 4 bits, pour les nombres $A = a_3\, a_2\, a_1\, a_0$ et $B = b_3\, b_2\, b_1\, b_0$ se construit de la façon suivante, avec une propagation de droite à gauche :

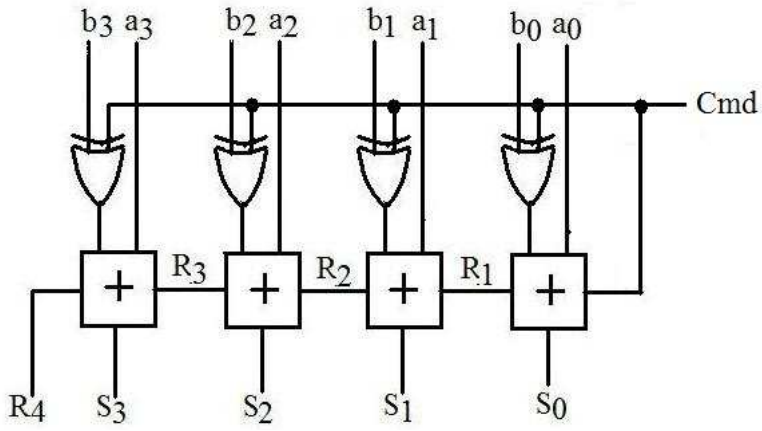


Un tel circuit, où les calculs se font l'un après l'autre, a le défaut de demander un temps assez long. Avec des nombres de 16 bits, et a fortiori de 32 ou 64 bits, on utilise d'autres méthodes, comme celle par anticipation de la retenue.

Exercice 6 : Additionneur soustracteur

Construire un additionneur-soustracteur en utilisant un signal de sélection *Cmd* qui, lorsqu'il vaut 0, provoque l'addition $A + B$, et lorsqu'il vaut 1 donne $A - B$. Rappelons que soustraire revient à ajouter le complémentaire de B et à ajouter 1.

Il suffit de placer le signal *Cmd* en R_0, la retenue d'origine. On met ce signal à 0 pour l'addition, et à 1 pour la soustraction, ce qui revient à ajouter 1. Pour avoir le complémentaire de B, il suffit de placer des portes XOR ayant deux entrées : b_i et $Cmd = 1$. Si b_i vaut 0, la porte XOR délivre 1, et si b_i vaut 1, XOR ramène 0, justement ce que l'on veut.



Exercice 7 : Rajout d'indicateurs à l'additionneur-soustracteur 4 bits

Ces indicateurs sont :

* N pour négatif, ramenant 1 si le résultat de l'addition $A + B$ est négatif, et 0 sinon.
* Z pour zéro, ramenant 1 si le $A + B$ est nul.
* C pour retenue, indiquant, dans le cas d'entiers sans signe, si le résultat provoque une retenue finale, provoquant un débordement avec un bit supplémentaire. En tenant compte de cette retenue, le résultat est juste.
* O pour *overflow* (débordement) dans le cas d'entiers signés, auquel cas l'opération devient fausse.

Pour l'indicateur N, il suffit d'utiliser la dernière sortie S_3. Si elle ramène 1, le nombre est bien négatif, et si c'est 0 il est positif ou nul.

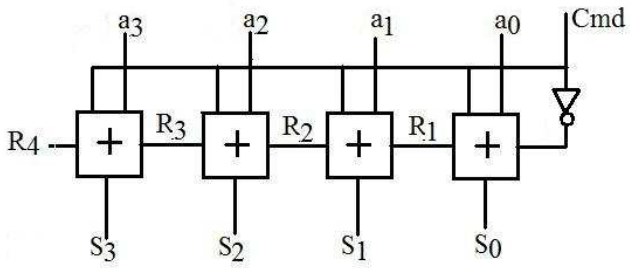
Pour avoir l'indicateur Z, il suffit de prendre les quatre sorties S_3 S_2 S_1 S_0, et de les faire passer par une porte NON OU. La sortie donne Z.

Pour l'indicateur C, on prend la dernière retenue R_4.

Pour l'indicateur O, on a vu dans le chapitre 4 que le dépassement de capacité se produisait lorsque les bits en position n et n – 1 (pour des nombres de longueur n) valaient 0 1 ou 1 0. Pour savoir s'il y a *overflow*, il suffit de faire $R_n \oplus R_{n-1}$.

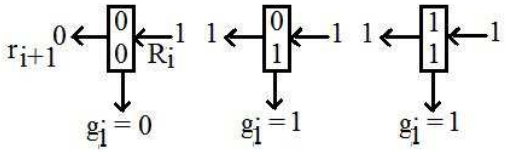
Exercice 8 : Incrémenteur-décrémenteur 4 bits

On utilise une commande de sélection *Cmd* qui, lorsqu'elle vaut 0, fait passer du nombre A au nombre $A + 1$, et lorsqu'elle vaut 1, fait passer de A à $A - 1$. Pour obtenir cela, il suffit d'aménager l'additionneur- soustracteur 4 bits. Pour $Cmd = 0$, on additionne le nombre A avec le nombre 0, mais en prenant 1 comme retenue initiale. Pour $Cmd = 1$, on ajoute à A le nombre $B = 1111$, avec comme retenue initiale 0.



d) L'additionneur avec anticipation de la retenue

En fait il n'y a pas vraiment anticipation. Les retenues sont calculées simultanément dans un autre circuit. Plaçons-nous au niveau du bit numéro i (i de 0 à $n-1$ pour un nombre $a_{n-1\ \ldots}\, a_3\, a_2\, a_1\, a_0$ de n bits). On distingue deux types de retenues. D'abord celle qui est provoquée par les deux bits a_i et b_i des deux nombres. On sait qu'il s'agit de $g_i = a_i\, b_i$ (g pour générateur de retenue). Mais il y a aussi celle, R_i, qui arrive de l'addition précédente, et dont il s'agit de savoir si elle se propage à la sortie ou non, dans ce que nous appellerons r_{i+1}, indépendamment de l'autre retenue g_i. D'abord si la retenue qui arrive est 0, elle ne peut que rester 0 en sortie. Par contre, si elle vaut 1 en arrivant, elle peut soit devenir 0 en sortie, soit rester 1. Voici les cas qui se présentent :



On en déduit que $r_{i+1} = R_i\,(a_i + b_i) = p_i\, R_i$ en posant $p_i = a_i + b_i$ (p signifiant propagation).[2]
Finalement la retenue sortante est la somme de la retenue g_i et de r_{i+1}, soit :

$R_{i+1} = g_i + p_i\, R_i$.

Pour un additionneur 4 bits, cela donne :

$R_1 = g_0 + p_0 R_0$

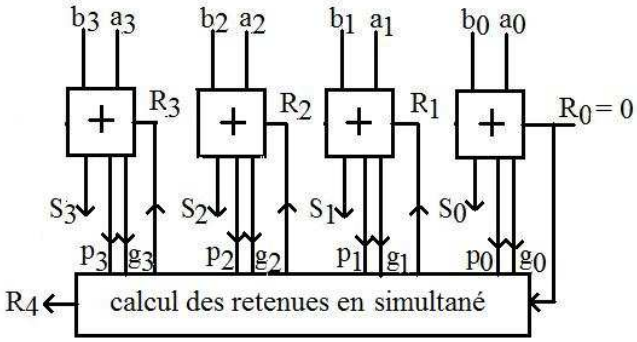
$R_2 = g_1 + p_1 R_1 = g_1 + p_1 g_0 + p_1 p_0 R_0$

$R_3 = g_2 + p_2 R_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 R_0$

$R_4 = g_3 + p_3 R_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 R_0$

Nous avons gardé R_0 mais si l'on se contente de faire une addition (et pas une soustraction), on peut prendre $R_0 = 0$. Toujours est-il que les quatre retenues sont calculées uniquement à partir des g_i et des p_i. Or ces derniers peuvent être obtenus simultanément à partir des quatre additionneurs 1 bit. D'où le schéma :

[2] Certains auteurs prennent $p_i = a_i \oplus b_i$ au lieu de $p_i = a_i + b_i$. C'est faux. Mais si l'on prend la formule finale $R_{i+1} = g_i + p_i\, R_i$, elle donne le même résultat que l'on prenne l'un ou l'autre des p_i. Le fait d'utiliser $a_i \oplus b_i$ a alors l'avantage d'être une opération déjà traitée dans l'additionneur.



Il a suffi d'ajouter des sorties supplémentaires p_i et g_i à chaque additionneur. Cela dit, si ce circuit est performant en termes de vitesse, il nécessite de nombreuses portes pour faire le calcul des retenues, comme la complexité des formules précédentes l'indique déjà pour un nombre de quatre bits. Pour des nombres de 16 bits, on est amené à utiliser quatre blocs de 4 bits mis en cascade, avec quelques aménagements supplémentaires.