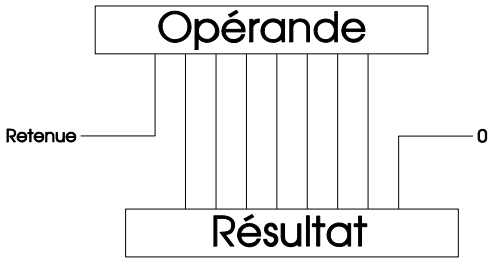
LES CIRCUITS ARITHMETIQUES

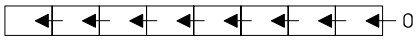
Ce chapitre explique comment les portes sont utilisées pour construire des circuits remplissant des fonctions de calcul arithmétique.

LA MULTIPLICATION PAR 2

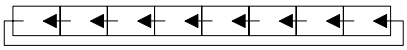
Voyez-vous comment, de manière très simple, multiplier un nombre par deux en binaire ?
Voilà : de la même manière qu'on rajoute un 0 à la droite d'un nombre pour le multiplier par 10 en décimal, rajouter un 0 à la droite d'un nombre en binaire revient à le multiplier par 2 (le poids de chaque bit étant multiplié par 2, l'ensemble est multiplié par 2).
Dans tous les circuits arithmétiques, les nombres sont codés en binaire, un fil correspondant à un bit. Dans le schéma ci-dessous (comme dans les suivants, sauf spécification particulière), l'opérande comme le résultat sont des octets, dont le bit de poids faible est dans le fil de droite, le bit de poids fort dans le fil de gauche. Le calcul se transmet depuis l'opérande vers le résultat.
Le circuit est magnifiquement simple :



(Revers de la médaille : la multiplication par dix est un véritable chemin de croix…)
Cette opération fait partie de la famille des *glissements* : ceux-ci consistent à "faire glisser" dans un sens les bits d'un octet. La multiplication par deux consiste en un glissement à gauche :

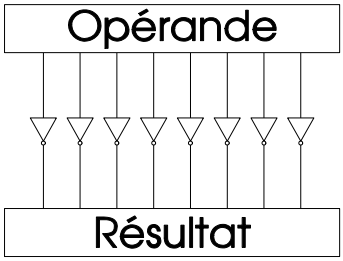


On définit aussi une autre famille de transformations que sont les *rotations*, qui consistent en des glissements en boucle fermée. Par exemple, voici une rotation à gauche (qui ne correspond cependant pas à une opération arithmétique simple) :



LE COMPLEMENTAIRE

Le complémentaire d'un octet est l'octet dont chaque bit résultat est l'inverse de l'opérande. La réalisation est fort simple :



Le complémentaire de l'octet X se note :

$\overline{X}$

L'INCREMENTATION

L'incrémentation d'un nombre consiste à lui ajouter la valeur 1. Il s'agit ici de voir quels composants vont pouvoir effectuer cette opération. Supposons que nous ajoutions 1 à 0000 0110. L'addition se réalise de la même manière que celle que nous avons apprise à l'école primaire : il s'agit d'additionner les nombres colonne par colonne en partant de celle des unités. Mais nous sommes ici en base deux :

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- mais 1 + 1 = 10 !

Sur ces bases, réalisons donc notre incrémentation :

```
  0000 0110
+ 0000 0001
___________
```

On additionne la colonne des unités :

```
  0000 0110
+ 0000 0001
___________
          1
```

Puis de même pour chaque colonne :

```
  0000 0110
+ 0000 0001
___________
  0000 0111
```

On trouve bien 0000 0110 + 1 = 0000 0111.
Supposons maintenant que nous ajoutions 1 à 0001 0111. On additionne la colonne des unités :

```
        1    ligne des retenues
  0001 0111
+ 0000 0001
___________
          0
```

En binaire, 1 + 1 = 10. On obtient donc 0 dans la colonne des unités, et une retenue de 1 dans la colonne des dizaines (terme pratique mais abusif, car nous sommes en base deux : il s'agirait plutôt de la colonne des « deuzaines »). Poursuivons :

```
       11    ligne des retenues
  0001 0111
+ 0000 0001
___________
         00
```

Nous sommes confrontés au même cas. Nous pouvons finir l'opération sans difficulté particulière :

```
      111
 0001 0111
+ 0000 0001
 ___________
 0001 1000
```

On obtient 0001 0111 + 1 = 0001 1000.

Traitement d'une colonne

Hormis pour la colonne des unités, on observe que l'opération se réalise entre les lignes de retenues et d'opérande. Quatre cas sont possibles :

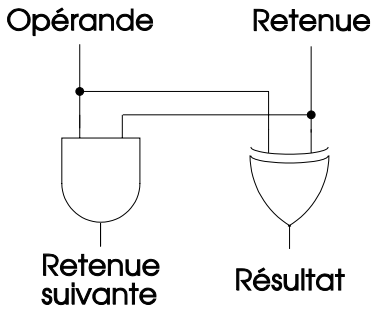
Entrées		Sorties	
Retenue	Opérande	Résultat	Retenue suivante
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Les colonnes "Résultat" et "Retenue suivante" ressemblent furieusement aux résultats des tables de vérité de portes que nous avons déjà vues. Cherchez bien…

Vérifiez vos conclusions avec la solution que voilà :

Résultat = Retenue XOR Opérande
Retenue suivante = Retenue AND Opérande

Voyez-vous alors de quelle manière chaque colonne peut être câblée ? Sinon, voyez ci-dessous :



Traitement de toutes les colonnes

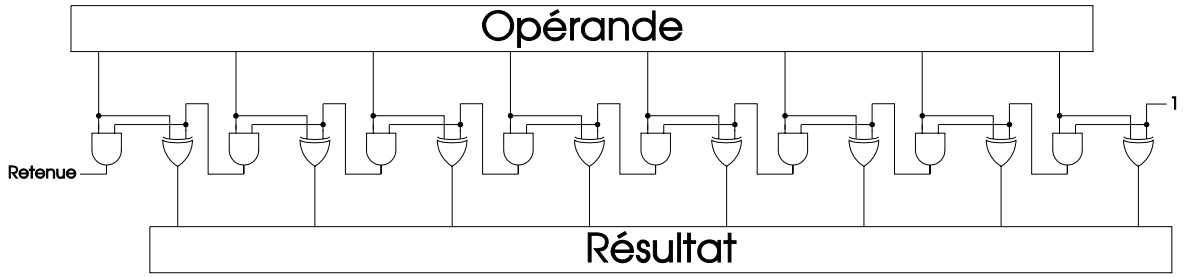
Il suffit alors de relier les colonnes entre elles via les retenues pour obtenir l'octet. On remarquera qu'il n'y a jamais de retenue sur la colonne des unités. Par ailleurs, nous n'avons jusqu'ici pas tenu compte de la deuxième opérande. Cette dernière n'étant utile que sur la colonne des unités, et étant parfaitement connue de par la nature de l'opération à réaliser, il suffit de forcer une retenue à 1 sur la première colonne en guise d'opérande, en considérant la deuxième opérande nulle, et l'incrémentation est réalisée. Ainsi, plutôt que de faire :

```
                    ligne des retenues
 0001 0111
+ 0000 0001
 ___________
```

Nous réalisons l'opération suivante, qui est parfaitement équivalente :

```
          1   ligne des retenues
    0001 0111
+   0000 0000
  ____________
```

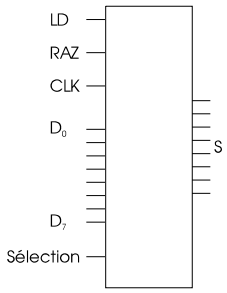
Nous savons, nous humains, que nous injectons une retenue au lieu d'une opérande, mais le transistor, cet âne, n'y voit que du feu : il voit un 1, peu lui importe d'où il vient. Il ne protestera donc pas de notre fourberie. Cela donne le schéma suivant :



La retenue globale sera égale à 0, sauf dans le cas de l'incrémentation de 11111111, dont le résultat sera 00000000, avec une retenue à 1 qui indique le dépassement de la capacité de l'octet, et pourra être éventuellement reprise dans un autre calcul. Pour reprendre l'exemple du compteur kilométrique à 6 chiffres de la voiture qui a tant roulé en si peu de temps, cela correspond au passage de 999 999 à 000 000 km, bien qu'elle ait 1 000 000 km à son actif : il est temps de vendre cette voiture devenue neuve... L'acheteur, souvent plus pertinent que le transistor, réalisera lui-même la retenue sur le prix de vente. Conclusion : mieux vaut vendre le bolide à un transistor...

REALISATION D'UN COMPTEUR

Un compteur est un circuit séquentiel : son état suivant dépend de son état précédent. On lui présente un nombre en entrée (par exemple 43d), qu'il restitue en sortie. Puis, à chaque fois qu'on lui donne un top sur une entrée identifiée, il incrémente la sortie. Le schéma fonctionnel de ce circuit est le suivant (compteur 8 bits) :

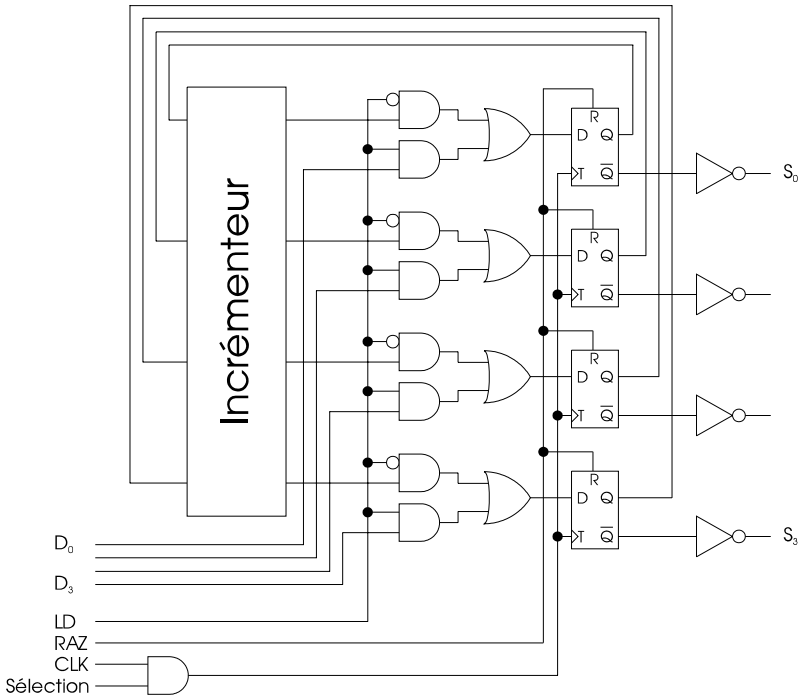


Entrées :

- RAZ (comme Remise A Zéro) : met la sortie S à 00000000b
- LD (comme LoaD = charger) : met la sortie S à la valeur indiquée par D
- D0-D7 (comme Donnée) : nombre D de départ
- CLK (comme CLocK = horloge) : donne les tops de mise à jour du nombre figurant en sortie

Sortie : nombre s'incrémentant à chaque fois que CLK passe de 0 à 1.

Ce circuit comporte bien sûr un incrémenteur, et des bascules, capables de mémoriser S à l'entrée de celui-ci le temps qu'il réalise son calcul et qu'on demande au circuit de mettre à jour S (par le signal CLK). Le signal RAZ (également nommé CLR = CLeaR = effacer) est directement connecté aux entrées R des bascules flip-flop. Le signal LD réalise le choix de l'entrée à retenir pour les bascules : ou bien la sortie de l'incrémenteur, ou bien D. CLK est raccordé aux entrées CLK des bascules. Enfin, le signal *Sélection* inhibe par des portes ET les autres signaux.



Pour des raisons de lisibilité, le schéma se limite à un compteur 4 bits.
La série de 8 portes ET et 4 portes OU réalisent un multiplexage entre la sortie de l'incrémenteur et D0-D3, commandé par LD :

- si LD est à 0, les portes ET à entrée inversée transmettent la sortie de l'incrémenteur alors que les ET à entrée directe transmettent 0.
- si LD est à 1, les portes ET à entrée inversée transmettent 0 alors que les ET à entrée directe transmettent D.
- les OU transmettent le choix sélectionné par la superposition des deux possibilités (car l'une est toujours égal à 0, et X OU 0 = X).

La porte ET traitant *Sélection* et CLK permet de figer l'horloge des bascules à 0 si Sélection est à 0, et donc d'interdire toute évolution des sorties du compteur.
La retenue de l'incrémenteur peut figurer comme sortie du compteur, via bien sûr une bascule de manière à synchroniser son apparition avec S (revenue à 00000000b si la retenue est à 1). Cela permet alors de connecter plusieurs compteurs 4 bits en parallèle de manière à réaliser un compteur 8, 12 ou 16 bits (voire plus).

L'ADDITION

Nous allons voir comment ajouter deux octets. Prenons l'exemple de 10101110 + 11011101. Le calcul en lui-même ne nous oppose plus aucune difficulté :

```
  1 1111 1000   ligne des retenues
    1010 1110
+ 1101 1101
 ____________
 1 1000 1011
```

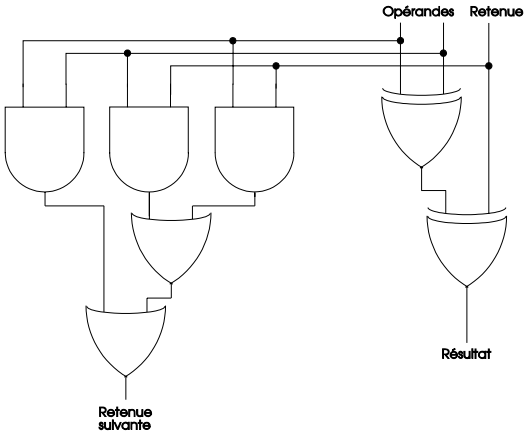
Reste à câbler cela.

Traitement d'une colonne

La table à respecter est la suivante :

Entrées			Sorties	
Retenue	Opérande 1	Opérande 2	Résultat	Retenue suivante
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

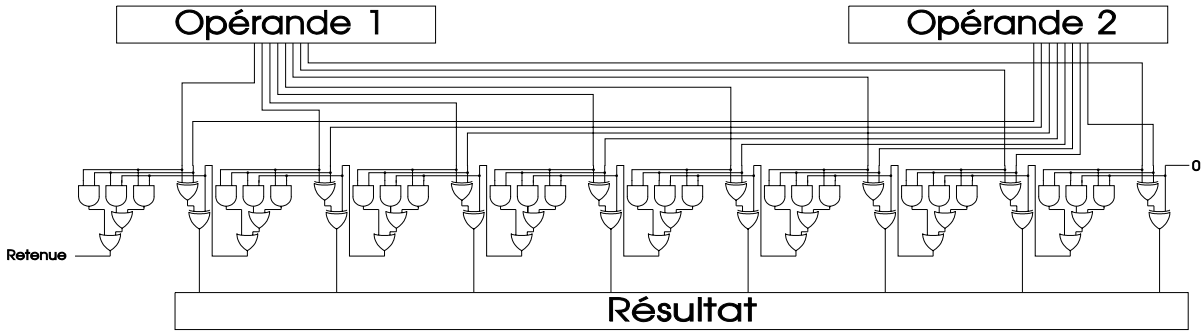
Ca se complique, non ?
Le "Résultat" tient à la parité des entrées, c'est à dire qu'il est à 0 s'il y a un nombre pair de 1 en entrée, à 0 s'il y a un nombre impair de 1 en entrée. Cherchez donc quelle porte réalise un calcul de parité...
La fonction XOU consiste en un test de parité. La fonction de parité étant associative (voir un dictionnaire si besoin), on peut arriver au résultat recherché en chaînant deux XOU.
De même, la "Retenue suivante" passe à 1 s'il y a au moins deux entrées à 1. La fonction ET passant à 1 si ses deux entrée sont à 1, on peut l'utiliser pour obtenir le résultat recherché : une fonction ET entre chacune des trois entrées détecte s'il y a au moins deux entrées à 1, aux sorties desquelles un OU (fonction également associative, pour notre plus grand bonheur) génère l'information "une porte ET au moins a vu deux 1". Cela donne le circuit suivant :



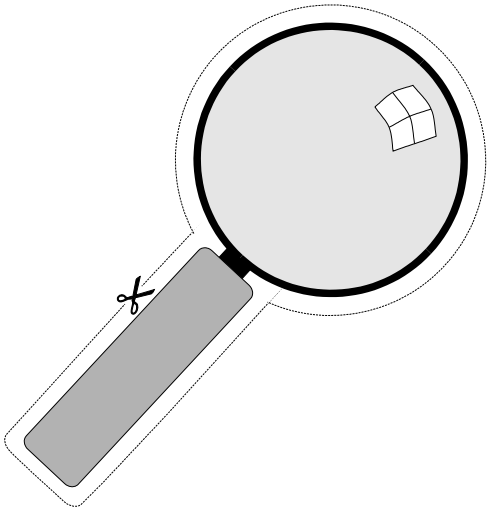
#Nous verrons par la suite une méthode systématique de définition des circuits à partir des tables de vérité, sans avoir à analyser finement le sens profond de ces dernières.

Traitement de toutes les colonnes

Il suffit alors de relier les colonnes entre elles via les retenues pour obtenir l'octet. Dans ce cas précis, on force la retenue de la colonne des unités à 0, car il ne peut pas y avoir là de retenue. Néanmoins, nous utiliserons dans le paragraphe suivant l'*additionneur avec retenue*, dont la retenue de la colonne des unités est une entrée du circuit (cela permet de câbler plusieurs additionneurs en parallèle pour former des additionneurs 16 ou 32 bits, voire plus).
De même que pour l'incrémentation, la dernière retenue indique le dépassement de la capacité de l'octet. Cela donne le schéma suivant :



L'auteur, dans sa grande bonté, vous offre ce superbe outil, à découper suivant les pointillés.



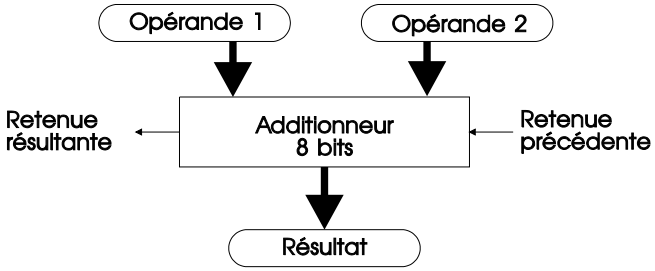
LA MULTIPLICATION

De même que pour l'addition ou la soustraction, le fait de calculer en binaire simplifie beaucoup le calcul des multiplications telles que nous, humains, avons l'habitude de poser. Par exemple :

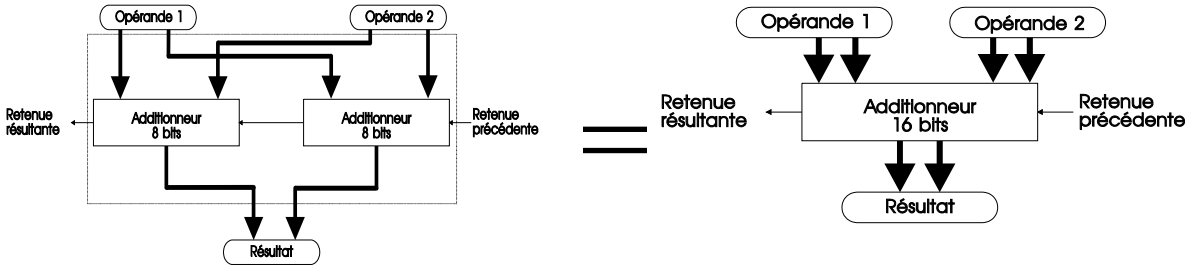
```
    10101010   (multiplicande)
  × 10001011   (multiplicateur)

  ----------
    10101010   (multiplicande)
   10101010    (multiplicande)
  00000000
 10101010      (multiplicande)
00000000
00000000
00000000
10101010      (multiplicande)

----------------
 101110001001110
```

En pratique, on part d'un *résultat intermédiaire* à 0. Puis, en commençant par celui de droite, on scrute chaque bit du multiplicateur : s'il est à 1, on additionne le *résultat intermédiaire* à la multiplicande, décalée au préalable du bon nombre de bits vers la gauche. Sinon, on ne touche pas au *résultat intermédiaire*. Puis on passe au bit suivant du multiplicateur. L'opération nécessite une addition sur 16 bits (*Résultat intermédiaire* + multiplicande décalée de n bits). Or, nous avons déjà réalisé un additionneur 8 bits. Moyennant une légère modification, nous allons en faire un additionneur 16 bits : il suffit de ne pas mettre la retenue de la colonne des unités

systématiquement à 0, mais de définir celle-ci comme une entrée (au même titre que les opérandes). Ainsi, le schéma suivant représente l'*additionneur avec retenue* 8 bits (dorénavant, chaque trait gras représente un ensemble de fils, appelé *bus*, comprenant ici 8 fils pour coder 8 bits) :



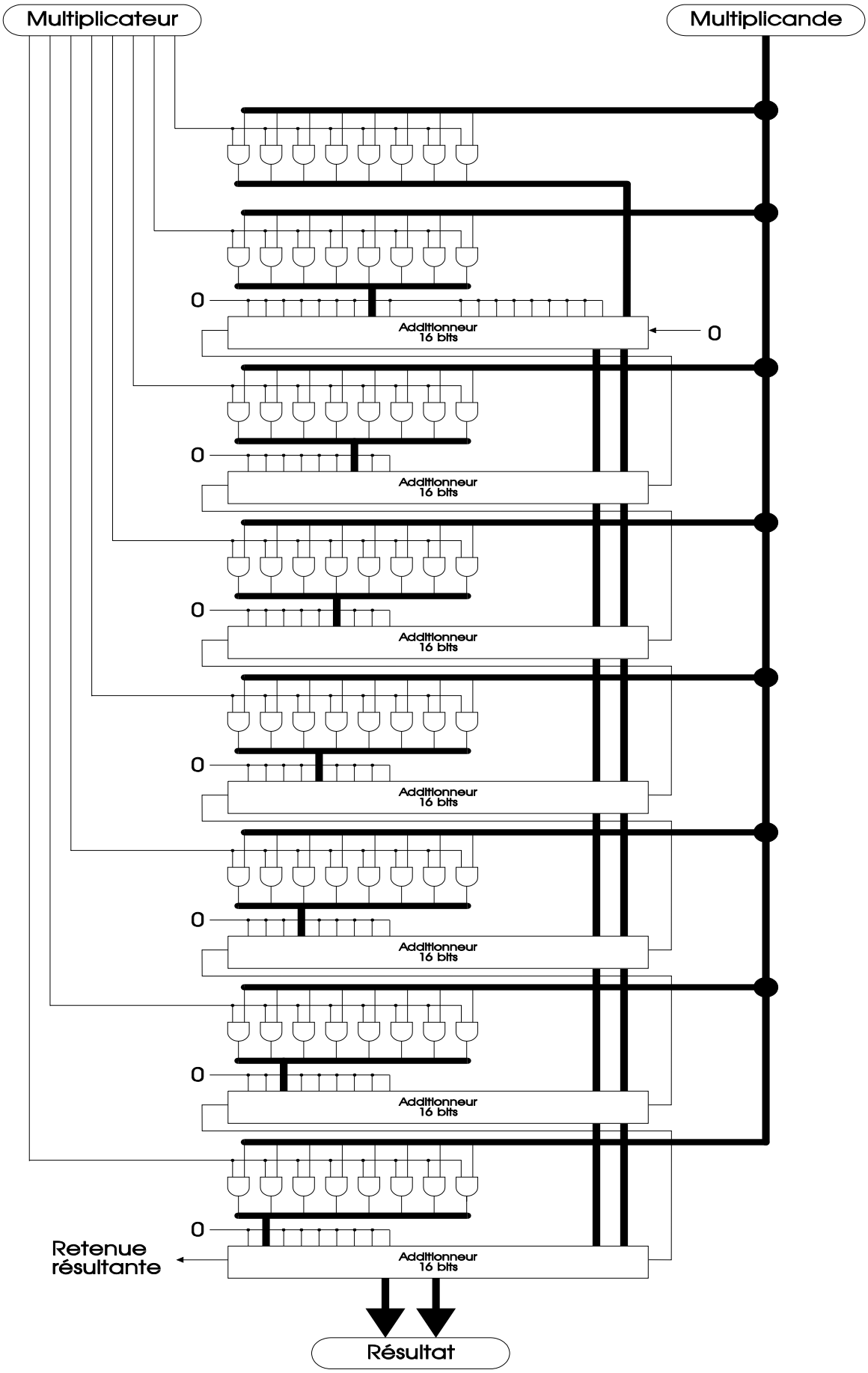
A partir duquel on construit un additionneur 16 bits : l'additionneur de droite effectue l'opération sur les 8 bits de droite des opérandes 16 bits, celui de gauche de leurs 8 bits de gauche. Le Résultat 16 bits est la concaténation des deux Résultats 8 bits.



Exemple pour 1010101010101010 + 0100000111111000 :

Opérandes	1010101010101010		0100000111111000	
	10101010 10101010		01000001 11111000	
Séparation par octets (poids fort et poids faible)	Poids forts		Poids faibles	
	10101010	01000001	10101010	11111000
Calcul des résultats et retenues	Retenue sortie = 0	Résultat = 11101011	Retenue intermédiaire = 1	Résultat = 10100010
	Retenue sortie = 0	Résultat+retenue = 11101100		
Concaténation des résultats	1110110010100010			

Cet additionneur 16 bits nous permet de réaliser notre multiplicateur suivant le principe décrit ci-dessus (respirez un bon coup) :


Multiplicateur
Multiplicande
0
Additionneur
16 bits
0
0
Additionneur
16 bits
0
Additionneur
16 bits
0
Additionneur
16 bits
0
Additionneur
16 bits
0
Additionneur
16 bits
0
Additionneur
16 bits
Retenue
résultante
Résultat

Les étages de AND génèrent 00000000 si le bit testé du multiplicateur est à 0, sinon ils transmettent la multiplicande (bus 8 bits).
Chaque additionneur reçoit en entrée :

- En opérande 1 : la multiplicande décalée du bon nombre de bits vers la gauche (grâce aux fils connectés à 0, et correctement positionnés par rapport au bus 8 bits sortant de l'étage de AND) si le bit correspondant du multiplicateur est à 1, 00000000 00000000b s'il est à 0 (grâce aux étages de AND).
- En opérande 2 : le résultat intermédiaire donné par l'additionneur précédent. Pour le "premier étage", l'additionneur est inutile, puisqu'il s'agit d'additionner le résultat de l'étage AND au résultat intermédiaire qui est 00000000 00000000b. Pour le deuxième étage, l'octet de poids fort de l'opérande 2 est forcément à 0 puisque l'opérande 2 est au plus égale à la multiplicande qui est sur 8 bits : l'octet de poids fort est donc fourni par les 0 servant à la génération de l'opérande 1.
- En retenue : 0 pour le premier additionneur, la retenue du précédent pour les suivants. Notez que les retenues sont parfaitement inutiles arithmétiquement parlant : 11111111b × 11111111b = 11111110 00000001b qui tient sur 16 bits. Les résultats intermédiaires étant forcément inférieurs au résultat final, toutes les retenues seront toujours à 0. Néanmoins, nous utilisons comme circuit l'additionneur 16 bits, pourvu des retenues d'entrée et de sortie (de manière à pouvoir réaliser un additionneur 32 bits si besoin, de la même manière que nous avons réalisé l'additionneur 16 bits avec deux additionneurs 8 bits). En transistor CMOS, il est néfaste de laisser une entrée non connectée. On doit donc connecter les retenues en entrée à une sortie. Vu que les retenues de sortie sont forcément à 0, et que nous avons besoin d'un 0 pour chaque retenue d'entrée, nous les relions entre elles (la retenue de sortie d'un additionneur à la retenue d'entrée de l'additionneur suivant). Relisez doucement et vous comprendrez...

Chaque additionneur génère en sortie le résultat intermédiaire, et une retenue à 0. Le dernier additionneur génère le résultat final, et une retenue (à 0 aussi).

POUR ALLER PLUS LOIN...

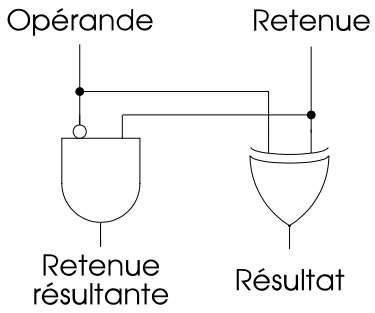
- Comment réaliser un module de décrémentation (soustraction de 1) ?
- Comment réaliser un module de division par 2 ? Ca repose, n'est-ce pas ?
- Comment réaliser un module de soustraction de deux octets ? Essayez déjà de poser la soustraction 10000000 - 00000011. Vous vous rendrez compte que la gestion des retenues est extrêmement délicate, celles-ci pouvant se transmettre de colonne en colonne. Essayez plutôt la méthode suivante (se limitant aux 8 bits de l'octet) :
  - que donne l'addition d'un nombre et de son complémentaire ? Ajoutez 1 au tout, et déduisez-en une manière de calculer l'opposé d'un nombre en binaire sur 1 octet.
  - à partir de ce résultat, réalisez un circuit de soustraction de deux octets en utilisant des circuits que nous avons déjà vus.
- En utilisant le multiplicateur 8bits × 8bits = 16bits, réalisez un multiplicateur 16bits × 16bits = 32bits.
- Question subsidiaire : combien le multiplicateur utilise-t-il de transistors ? Il y a des moyens moins gourmands de réaliser un multiplicateur : nous avons utilisé une méthode pédagogique, car "humaine", pour effectuer notre multiplication. Certaines méthodes typiquement binaires sont bien plus efficaces.

REPONSES

- *Comment réaliser un module de décrémentation (soustraction de 1) ?*
  Table de vérité pour une colonne (dans la soustraction, on génère si besoin une retenue sur la colonne en cours de calcul sur l'opérande 1, en la reportant dans la colonne suivante sur l'opérande 2) :

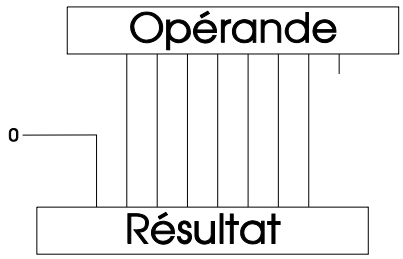
Opérande	Retenue d'entrée	Retenue résultante	Résultat
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

  D'où, pour une colonne :



Puis, il suffit de chaîner ces éléments et de forcer la première retenue à 1, comme pour l'incrémenteur.

- *Comment réaliser un module de division par 2 ?*



- *Comment réaliser un module de soustraction de deux octets ?*
  - *que donne l'addition d'un nombre et de son complémentaire ? Ajoutez 1 au tout, et déduisez-en une manière de calculer l'opposé d'un nombre en binaire.*

    A + ~~A~~ = FFh, donc A + ~~A~~ + 1 = 0 (car nous restons sur 8 bits, et ignorons la retenue)
    D'où : -A = ~~A~~ + 1, et donc B - A = B + (~~A~~ + 1)

    Au passage, on voit comment écrire un nombre négatif : suivant comment on le traite par la suite (convention de traitement), un octet dont le bit7 est à 1 peut être vu comme une valeur positive, comprise entre 128 et 255, mais aussi comme une valeur négative, comprise entre -128 et -1.

  Maintenant, magie...

  Par ailleurs, on peut très bien faire l'exercice en base 10 : de la même manière qu'en base 2 (où A étant un chiffre, on peut définir son complémentaire comme étant le chiffre qui satisfait A + ~~A~~ = 1 qui est le plus grand chiffre de la base), le complémentaire en base 10 d'un chiffre est le chiffre qui, additionné au premier, donne 9 (le plus grand chiffre de la base 10). Ainsi, ~~0~~ = 9, ~~1~~ = 8, ~~2~~ = 7..., ~~9~~ = 0. N étant un nombre, et en raisonnant juste sur le nombre maximum de ses chiffres (par exemple 5 chiffres), on a :
  N + ~~N~~ = 99999, donc N + ~~N~~ + 1 = 0 (car nous restons sur 5 chiffres, et ignorons la retenue)
  D'où : -N = ~~N~~ + 1, et donc M - N = M + (~~N~~ + 1)

  Par exemple, pour calculer 89453 - 12458 = 76995 :

  89453 - 12458
  = 89453 + ~~12458~~ + 1
  = 89453 + 87541 + 1
  = 176995
  = 76995 (on ne retient que les 5 premiers chiffres)

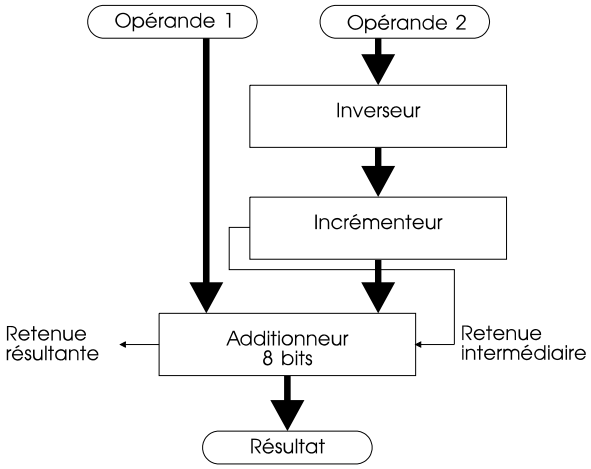
  Inversement, pour calculer 12458 - 89453 = -76995 :

  12458 - 89453
  = 12458 + ~~89453~~ + 1
  = 12458 + 10546 + 1
  = 23005
  = 23004 + 1
  = - ~~23004~~ (car -N = ~~N~~ + 1, donc - ~~N~~ = N + 1)

= - 76995

Attention, il faut que les deux nombres aient autant de chiffres : pour calculer par exemple 18237 - 538, on n'utilise pas ~~538~~ = 461, mais ~~00538~~ = 99461.

– *réalisez un circuit de soustraction de deux octets en utilisant des circuits que nous avons déjà vus.*

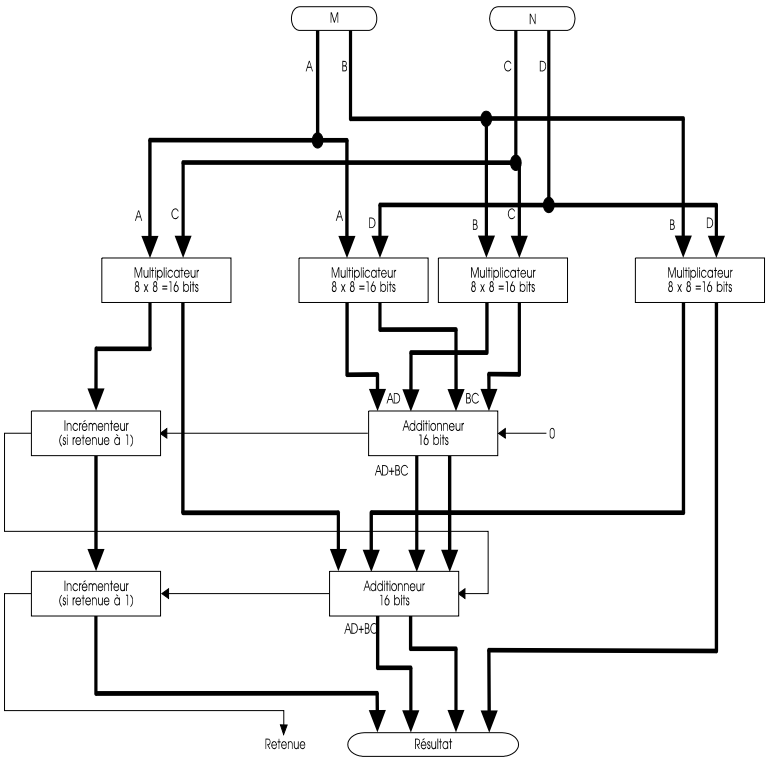


- *En utilisant le multiplicateur 8bits × 8bits = 16bits, réalisez un multiplicateur 16bits × 16bits = 32bits.*
  Multiplions deux nombres de 16 bits chacun : M × N. Appelons A et C les octets de poids fort de M et N, B et D leurs octets de poids faible. On a :

$$M = 256 \times A + B \text{ et } N = 256 \times C + D$$

$$M \times N = (256A + B)\,(256C + D) = 65536AC + 256\,(AD + BC) + BD.$$

  La multiplication par 256 consistant à décaler d'un octet (8 bits) vers la gauche et celle par 65536 de 2, on obtient le câblage suivant :



- Les incrémenteurs sont identiques à celui réalisé dans le présent chapitre, si ce n'est que la retenue d'entrée n'est plus systématiquement forcée à 1, mais est ici une vraie entrée, qui peut prendre la valeur 0 ou 1. Si elle est à 1, l'incrémentation se fait, sinon non.
- D'autre part, la retenue de sortie du premier incrémenteur est toujours 0. En effet, dans le cas le plus défavorable 65535d × 65535d, on a A = 11111110b, dont l'incrémentation ne génère pas de retenue. Comme nous avons besoin d'un 0 en entrée du premier additionneur 16 bits, cette sortie est reliée à cette entrée.
- Le câblage du deuxième additionneur 16 bits s'explique par la formule

  $$M \times N = 65536AC + 256\,(AD + BC) + BD,$$

  multiplier par 256 revenant à décaler de 8 bits, par 16384 de 16 bits. Ces opérations se réalisent donc comme suit :

	4ème octet	3ème octet	2ème octet	1er octet
	A × C		0	
	octet de poids fort	octet de poids faible		
+	0		B × D	
			octet de poids fort	octet de poids faible
+	0	A × D + B × C		0
		octet de poids fort	octet de poids faible	
=	Résultat			

  La réelle addition 16 bits de deux nombres se fait sur les 2ème et 3ème octets. Sur les 1er et 4ème octets, on additionne à 0 : l'opération se réalise donc par une addition 16 bits, grâce à un positionnement adéquat des fils supportant les octets, et une prise en compte des retenues de sortie des additionneurs 16 bits.

- Remarque : la retenue finale est toujours nulle, car 65535d × 65535d = 4294705156d, qui tient sur 32 bits.