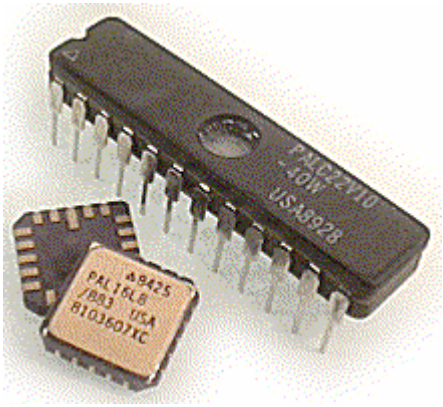
ELECTRONIQUE NUMERIQUE
Logique combinatoire et multiplexage

EPMI Cergy
1AING



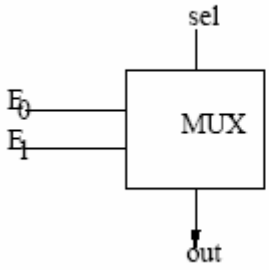
I- MULTIPLEXAGE:

I-1-DEFINITION :

Circuit à 2^n entrées d'informations, n entrées de sélection, et une sortie. Il permet l'aiguillage de l'une de ces entrées vers la sortie.

I-2-EXEMPLE :

- Multiplexeur à deux entrées



E_1	E_0	sel	out
X	X	0	E_0
X	X	1	E_1

$\rightarrow S = \overline{sel}.E_0 + sel.E_1$

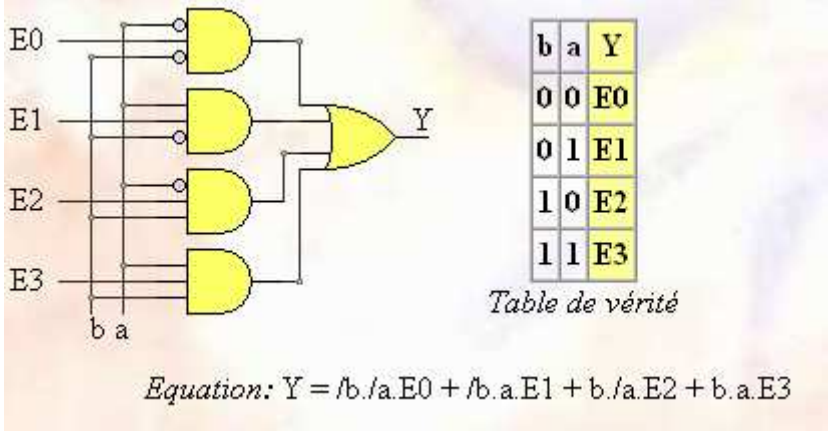
Remarque :

La table de vérité devient rapidement très importante (à partir de 4 entrées : 16 lignes). On exprime alors directement la fonction de sortie.

- Multiplexeur à deux fils d'adresse :

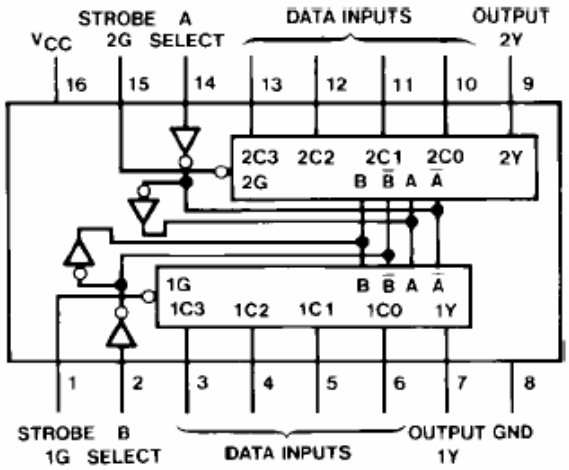
Deux bits d'adresse nous permettent d'aiguiller $2^2 = 4$ données.

E0	E1	E2	E3	A1	A0	S
X	X	X	X	0	0	E0
X	X	X	X	0	1	E1
X	X	X	X	0	0	E2
X	X	X	X	0	1	E3
X	X	X	X	1	0	E4
X	X	X	X	1	1	E5
X	X	X	X	1	0	E6
X	X	X	X	1	1	E7



- Multiplexeur Réel : 74LS153

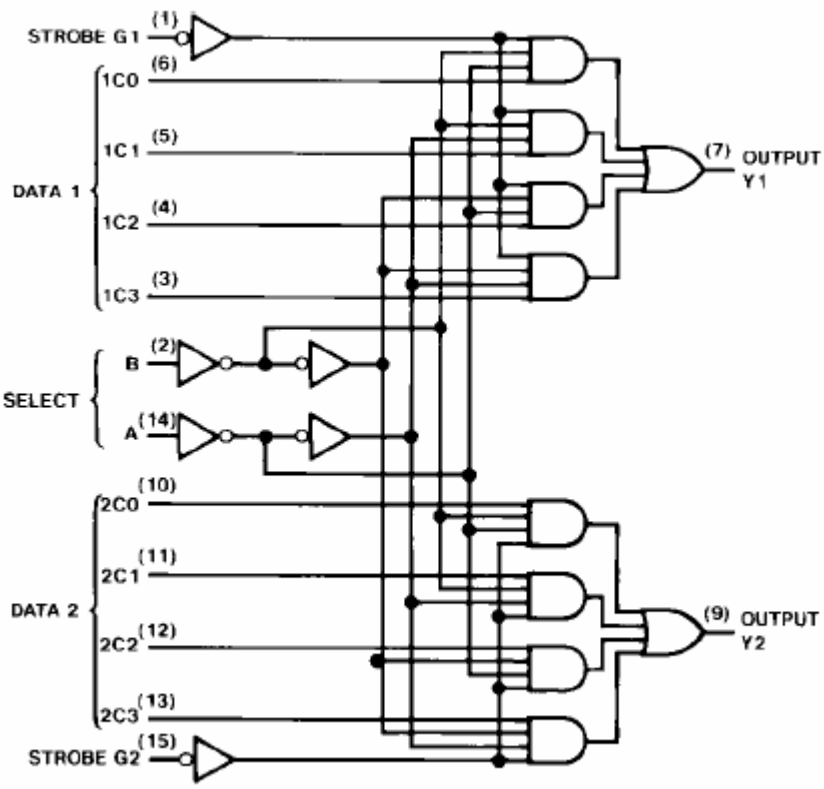
Connection Diagram



Function Table

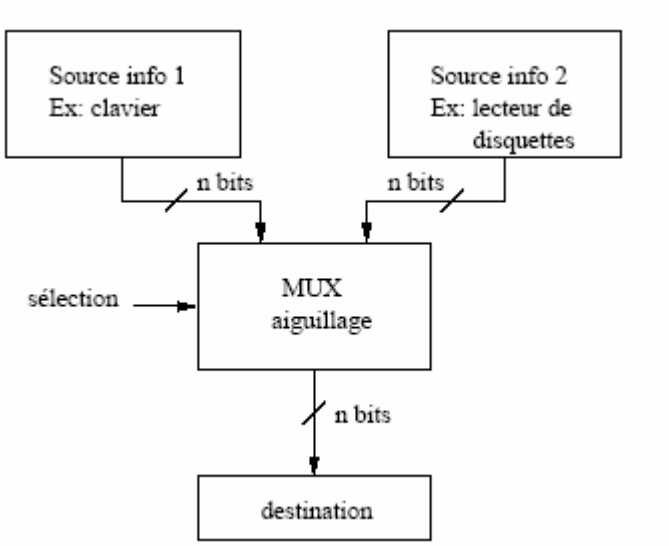
Select Inputs		Data Inputs				Strobe	Output
B	A	C0	C1	C2	C3	G	Y
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

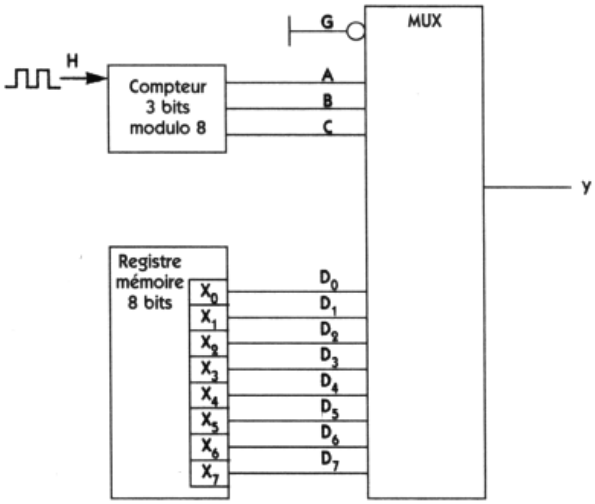
Select inputs A and B are common to both sections.
H = HIGH Level
L = LOW Level
X = Don't Care



I-3-APPLICATONS :

Une application concrète des multiplexeurs est la transformation de données parallèles en données séries. La transmission doit être régulière et contrôlée, il faut donc une validation (strobe, enable) et une horloge.





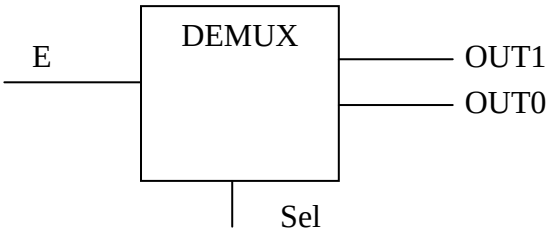
II- DEMULTIPLEXAGE :

I-1-DEFINITION :

C'est l'opération inverse on doit aiguiller une information type série sur 2^n sorties n étant le nombre de fils d'adresse.

II-2-EXEMPLE :

- Démultiplexeur à une entrée et deux sorties



E	Sel	OUT1	OUT0
X	0	0	E
X	1	E	0

$OUT0 = E.\overline{Sel}$

$OUT1 = E.Sel$

Remarque :

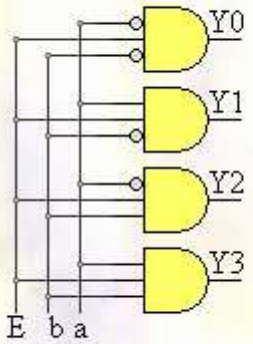
La table de vérité devient rapidement très importante (à partir de 4 fils de sélection : 16 lignes). On exprime alors directement la fonction de sortie.

- Demultiplexeur à trois fils d'adresse :

Les fabricants de composants valident la solution active à l'état bas

E	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
X	0	0	0	1	1	1	1	1	1	1	0
X	0	0	1	1	1	1	1	1	1	0	1
X	0	1	0	1	1	1	1	1	0	1	1
X	0	1	1	1	1	1	1	0	1	1	1
X	1	0	0	1	1	1	0	1	1	1	1
X	1	0	1	1	1	0	1	1	1	1	1
X	1	1	0	1	0	1	1	1	1	1	1
X	1	1	1	0	1	1	1	1	1	1	1

Démultiplexeur 2 vers 4

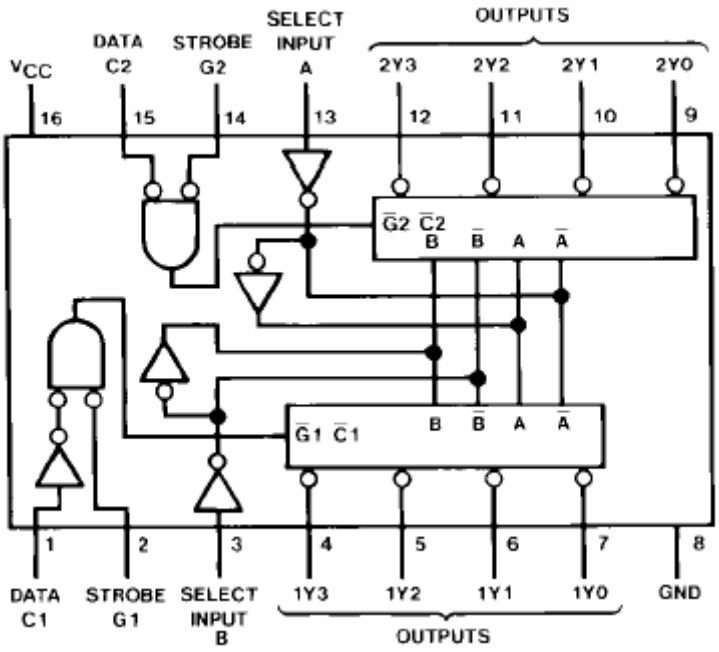


b	a	Y0	Y1	Y2	Y3
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E

Y0 = /b./a.E
Y1 = /b.a.E
Y2 = b./a.E
Y3 = b.a.E

Table de vérité *Equations*

Démultiplexeur Réel 4514



Function Tables

3-Line-to-8-Line Decoder or 1-Line-to-8-Line Demultiplexer

Inputs				Outputs							
Select			Strobe Or Data	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
C (Note 1)	B	A	G (Note 2)	2Y0	2Y1	2Y2	2Y3	1Y0	1Y1	1Y2	1Y3
X	X	X	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H
L	H	L	L	H	H	L	H	H	H	H	H
L	H	H	L	H	H	H	L	H	H	H	H
H	L	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	L	H	H
H	H	L	L	H	H	H	H	H	H	L	H
H	H	H	L	H	H	H	H	H	H	H	L

2-Line-to-4-Line Decoder or 1-Line-to-4-Line Demultiplexer

Inputs				Outputs			
Select		Strobe	Data	1Y0	1Y1	1Y2	1Y3
B	A	G1	C1				
X	X	H	X	H	H	H	H
L	L	L	H	L	H	H	H
L	H	L	H	H	L	H	H
H	L	L	H	H	H	L	H
H	H	L	H	H	H	H	L
X	X	X	L	H	H	H	H

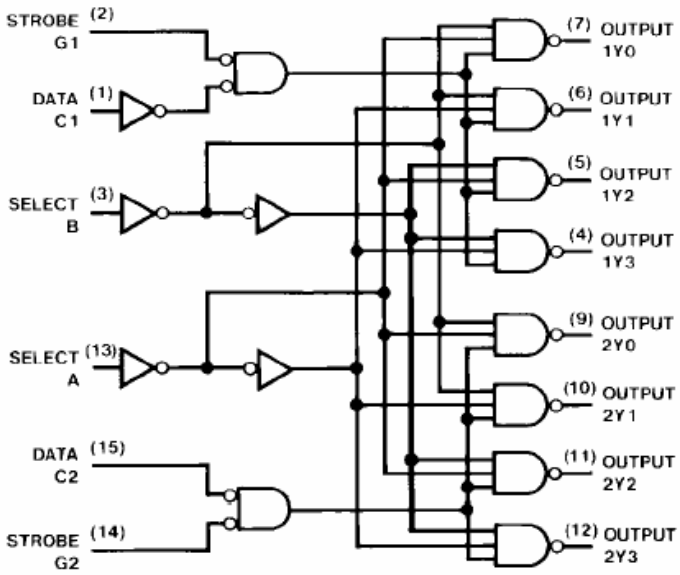
Inputs				Outputs			
Select		Strobe	Data	2Y0	2Y1	2Y2	2Y3
B	A	G2	C2				
X	X	H	X	H	H	H	H
L	L	L	L	L	H	H	H
L	H	L	L	H	L	H	H
H	L	L	L	H	H	L	H
H	H	L	L	H	H	H	L
X	X	X	H	H	H	H	H

H = HIGH level
L = LOW level
X = don't care

Note 1: C = inputs C1 and C2 connected together

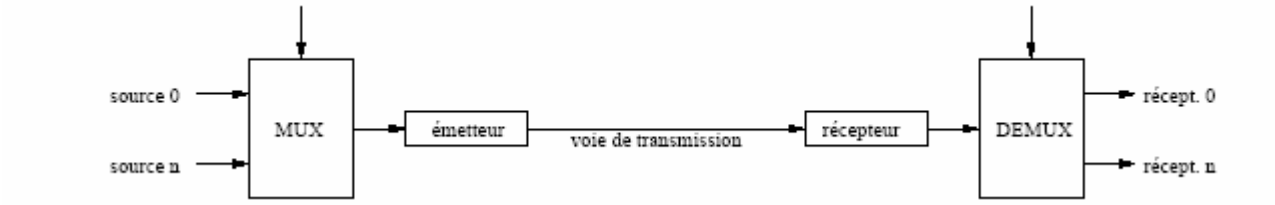
Note 2: G = inputs G1 and G2 connected together

Logic Diagram



III- APPLICATION :

L'application la plus connue est le modem MODulateur (multiplexeur) DEModulateur (démultiplexeur)



IV- COMPARTEUR :

IV-1- DEFINITION :

Il détecte l'égalité entre deux nombres A et B. Certains circuits permettent également de détecter si A est supérieur ou bien inférieur à B.

a_i	b_i	E_i	S_i	I_i
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

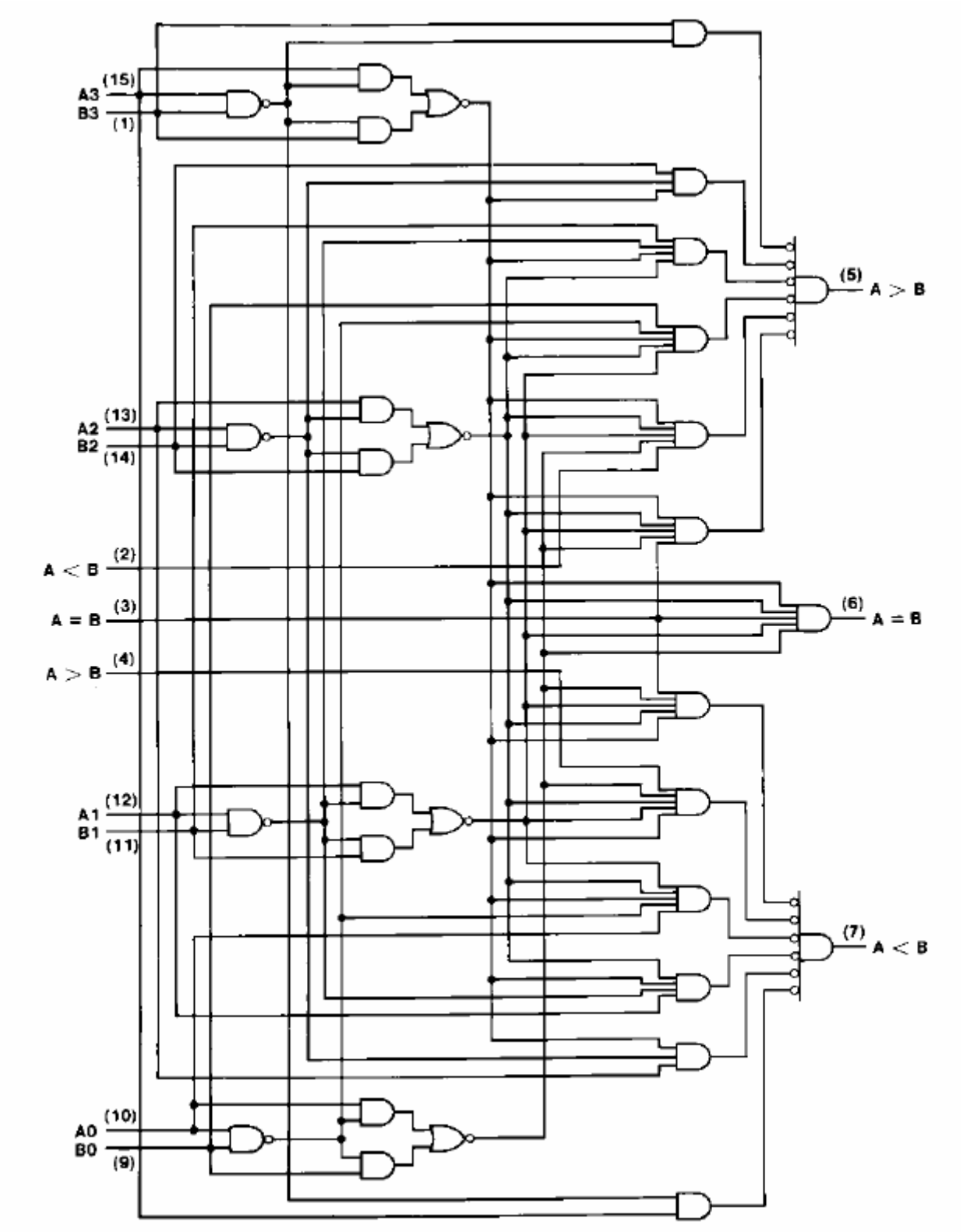
$$E_i = a_i = b_i = \overline{a \oplus b}$$
$$S_i = a_i > b_i = a.\overline{b}$$
$$I_i = a_i < b_i = \overline{a}.b$$
$$D_i = a_i \neq b_i = a \oplus b$$

IV-2-EXEMPLE :7485

Function Table

Comparing Inputs				Cascading Inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	A > B	A < B	A = B	A > B	A < B	A = B
A3 > B3	X	X	X	X	X	X	H	L	L
A3 < B3	X	X	X	X	X	X	L	H	L
A3 = B3	A2 > B2	X	X	X	X	X	H	L	L
A3 = B3	A2 < B2	X	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 > B1	X	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 < B1	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	X	X	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	H	L	L	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	L	H	H	L

H = HIGH Level, L = LOW Level, X = Don't Care

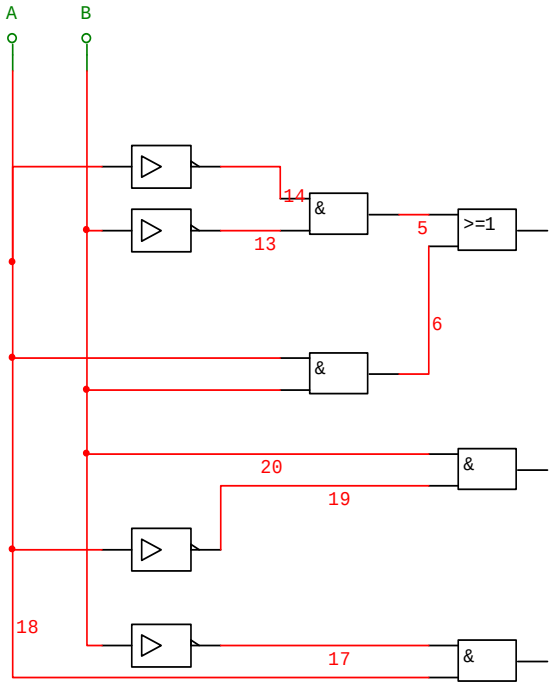

A3
B3
(15)
(1)
A2
B2
(13)
(14)
A1
B1
(12)
(11)
A0
B0
(10)
(9)
A < B
(2)
A = B
(3)
A > B
(4)
(5)
A > B
(6)
A = B
(7)
A < B

IV-3- EXERCICE :

Construisons un comparateur 1 bit.
Il y a 3 cas de figure

B	A	A=B	A>B	A<B
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

Equation A=B donne A'B'+AB
Equation A<B donne A'.B
Equation A>B donne A.B'



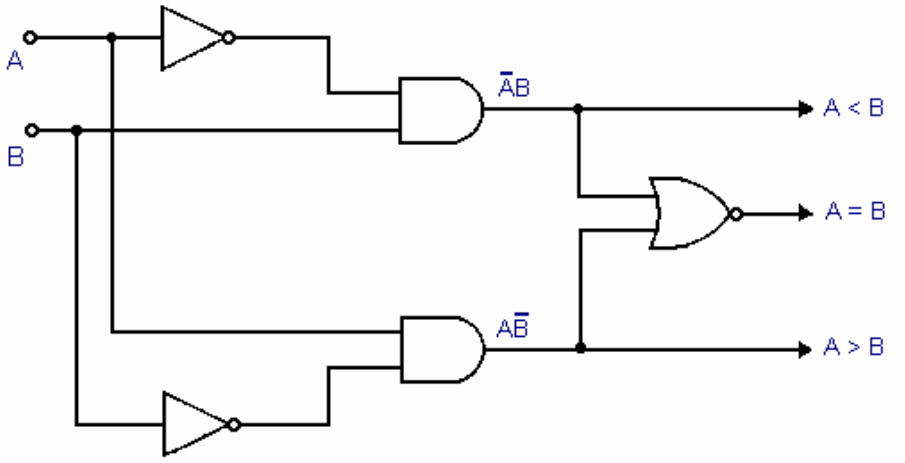


Table de vérité d'un comparateur 2 bits

A_1, B_1	A_0, B_0	A=B	A>B	A<B
$A_1 > B_1$	X	0	1	0
$A_1 < B_1$	X	0	0	1
$A_1 = B_1$	$A_0 > B_0$	0	1	0
$A_1 = B_1$	$A_0 < B_0$	0	0	1
$A_1 = B_1$	$A_0 = B_0$	1	0	0

Equations: $(A=B)=A_1B_1A_0B_0 + A_1\,B_1\,A_0'\,B_0'$

$A<B = A_1'\,B_1 + A_1\,B_1\,A_0'\,B_0$

$A>B = A_1\,B_1' + A_1\,B_1\,A_0\,B_0'$

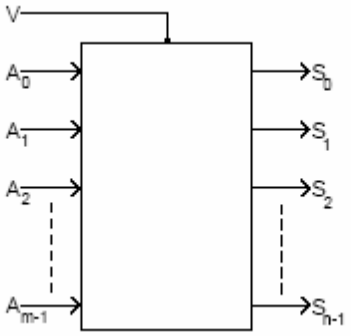
Construire le logigramme

V- CODEUR (encodeur) / DECODEUR:

V-1- DEFINITION :

Circuit à $M=2^N$ entrées et N sorties qui code en binaire le rang de la seule entrée active. (Etat bas ou haut, il faut le définir)

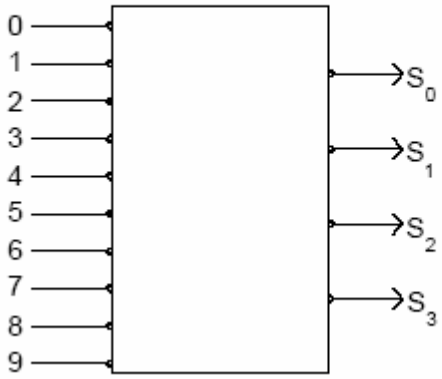
une seule entrée active à la fois



représentation en sortie sur N bits

V-2- EXEMPLE:

Codeur décimal-DCB : 10 entrées, 4 sorties



$\overline{A_9}$	$\overline{A_8}$	$\overline{A_7}$	$\overline{A_6}$	$\overline{A_5}$	$\overline{A_4}$	$\overline{A_3}$	$\overline{A_2}$	$\overline{A_1}$	$\overline{A_0}$	$\overline{S_3}$	$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$
1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	0	1	0	0	0	1
1	1	1	1	1	1	1	0	1	1	0	0	1	0
1	1	1	1	1	1	0	1	1	1	0	0	1	1
1	1	1	1	1	0	1	1	1	1	0	1	0	0
1	1	1	1	0	1	1	1	1	1	0	1	0	1
1	1	1	0	1	1	1	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	1	0	0	1

Construisons un décodeur 4 bits

A_3	A_2	A_1	A_0	S_1	S_0
1	1	1	0	0	0
1	1	0	1	0	1
1	0	1	1	1	0
0	1	1	1	1	1

Equations de $S_0 = A_0\, A'_1\, A_2\, A_3 + A_0\, A_1\, A_2\, A'_3$ $S_1 = A_0\, A_1\, A'_2\, A_3 + A_0\, A_1\, A_2\, A'_3$

V-3- ENCODEUR DE PRIORITE:

Le fonctionnement est le même que le codeur normal à l'exception près que si un bit « n » est actif tous les bits « n-1 » ne sont pas pris en comptent. On choisit donc le bit de poids le plus élevé actif.

Codeur de priorité 8 vers 3

0	1	2	3	4	5	6	7	A_2	A_1	A_0
X	X	X	X	X	X	X	0	0	0	0
X	X	X	X	X	X	0	1	0	0	1
X	X	X	X	X	0	1	1	0	1	0
X	X	X	X	0	1	1	1	0	1	1
X	X	X	0	1	1	1	1	1	0	0
X	X	0	1	1	1	1	1	1	0	1
X	0	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1

Les équations s'en trouvent simplifiées :

A_0=7'+5'67+3 '4567+1'234567 A_1=7'+6'7+3'4567+2'34567 A_2=7'6'7+5'67+4'567

V-4- EXEMPLE:

Encodeur prioritaire 4 vers 2

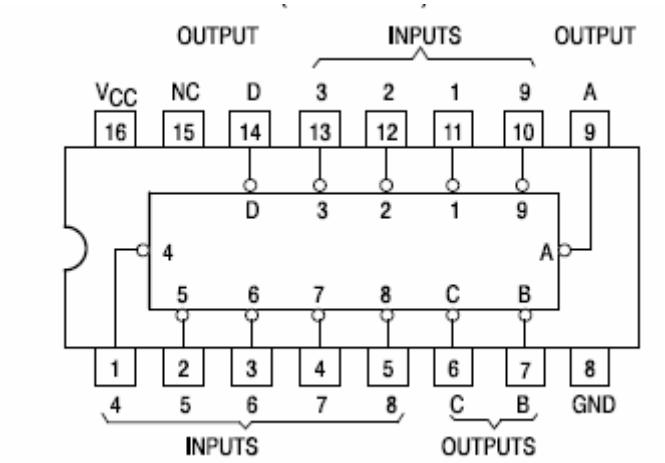
E0	E1	E2	E3	x	b	a
1	-	-	-	1	**0**	**0**
0	**1**	-	-	1	**0**	**1**
0	0	**1**	-	1	**1**	**0**
0	0	0	**1**	1	**1**	**1**
0	0	0	0	0	0	0

Entrée "basse" E0 prioritaire

E0	E1	E2	E3	x	b	a
1	0	0	0	1	**0**	**0**
-	**1**	0	0	1	**0**	**1**
-	-	**1**	0	1	**1**	**0**
-	-	-	**1**	1	**1**	**1**
0	0	0	0	0	0	0

Entrée "haute" E3 prioritaire

74148



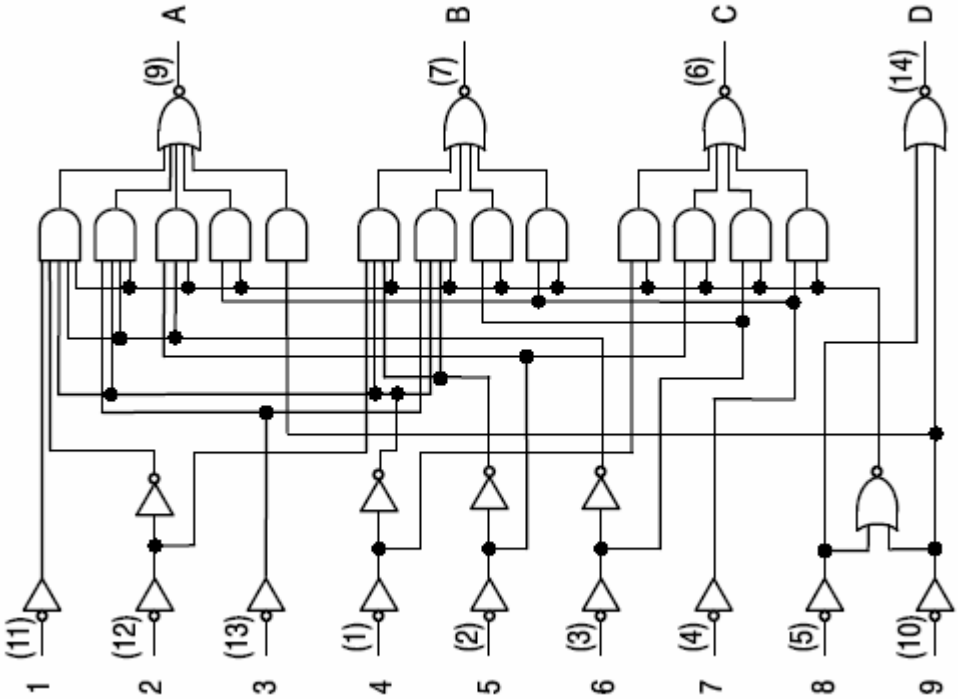
SN54/74LS147
FUNCTION TABLE

INPUTS									OUTPUTS			
1	2	3	4	5	6	7	8	9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Logic Level, L = LOW Logic Level, X = Irrelevant

SN54/74LS748
FUNCTION TABLE

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H



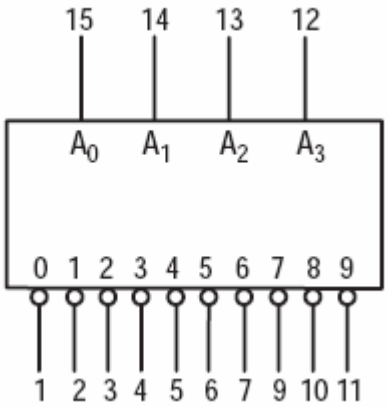
Rechercher l'équation de A et faire le logigramme

V-4- DECODEUR:

Le décodeur est un circuit qui établit la correspondance entre un code d'entrée sur N bits et M lignes de sortie ($M \leq 2^N$).
Pour chacune des combinaisons d'entrée, une seule ligne de sortie est validée.

La plupart des décodeurs sont dotés d'une ou plusieurs entrées de validation qui commandent son fonctionnement.

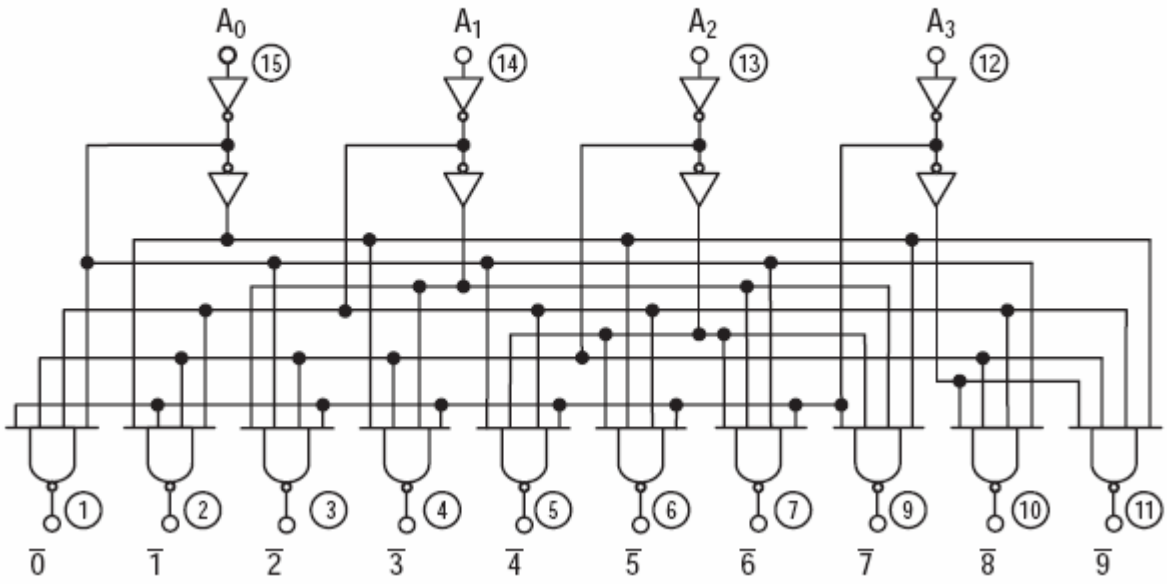
Exemple le 7442



TRUTH TABLE

A_0	A_1	A_2	A_3	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{8}$	$\overline{9}$
L	L	L	L	L	H	H	H	H	H	H	H	H	H
H	L	L	L	H	L	H	H	H	H	H	H	H	H
L	H	L	L	H	H	L	H	H	H	H	H	H	H
H	H	L	L	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	H	H	H	L	H	H	H	H	H
H	L	H	L	H	H	H	H	H	L	H	H	H	H
L	H	H	L	H	H	H	H	H	H	L	H	H	H
H	H	H	L	H	H	H	H	H	H	H	L	H	H
L	L	L	H	H	H	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L
L	H	L	H	H	H	H	H	H	H	H	H	H	H
H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H	H	H	H	H	H

LOGIC DIAGRAM



VI- TRANSCODEUR :

Les **transcodeurs** permettent le passage d'un code binaire de n bits à un autre code binaire de p bits, n pouvant être égal à p.

En voici quelques exemples:
Passage d'un code binaire vers un autre code binaire (parmi tous les codes possibles et imaginables); parmi les plus connus:

- **codes pondérés** :
  - binaire naturel (pondération 8, 4, 2, 1)
  - DCB ="Décimal Codé Binaire" ; en anglais: BCD ="Binary Coded Decimal"
  - Aiken (pondération 2, 4, 2, 1)
- **codes non pondérés** :
  - binaire réfléchi ou code Gray
  - binaire excédent 3 ou Stibiz
  - 2 parmi 5
  - "Hexadécimal" ou "Décimal Codé Binaire" vers afficheur 7 segments appelé aussi "décodeur" au lieu de transcodeur
    - hexadécimal-7segments: MC14495, TIL311
    - DCB-7segments: 7446, 7447, 7448, 4511, 4543

Truth Table

Decimal Or Function	Inputs						Outputs							
	$\overline{LT}$	$\overline{RBI}$	A_3	A_2	A_1	A_0	$\overline{BI/RBO}$	a	b	c	d	e	f	g
0 (Note 1)	H	H	L	L	L	L	H	H	H	H	H	H	H	L
1 (Note 1)	H	X	L	L	L	H	H	L	H	H	L	L	L	L
2	H	X	L	L	H	L	H	H	H	L	H	H	L	H
3	H	X	L	L	H	H	H	H	H	H	H	L	L	H
4	H	X	L	H	L	L	H	L	H	H	L	L	H	H
5	H	X	L	H	L	H	H	H	L	H	H	L	H	H
6	H	X	L	H	H	L	H	L	L	H	H	H	H	H
7	H	X	L	H	H	H	H	H	H	H	L	L	L	L
8	H	X	H	L	L	L	H	H	H	H	H	H	H	H
9	H	X	H	L	L	H	H	H	H	H	L	L	H	H
10	H	X	H	L	H	L	H	L	L	L	H	H	L	H
11	H	X	H	L	H	H	H	L	L	H	H	L	L	H
12	H	X	H	H	L	L	H	L	H	L	L	L	H	H
13	H	X	H	H	L	H	H	H	L	L	H	L	H	H
14	H	X	H	H	H	L	H	L	L	L	H	H	H	H
15	H	X	H	H	H	H	H	L	L	L	L	L	L	L
$\overline{BI}$ (Note 2)	X	X	X	X	X	X	L	L	L	L	L	L	L	L
$\overline{RBI}$ (Note 3)	H	L	L	L	L	L	L	L	L	L	L	L	L	L
$\overline{LT}$ (Note 4)	L	X	X	X	X	X	H	H	H	H	H	H	H	H

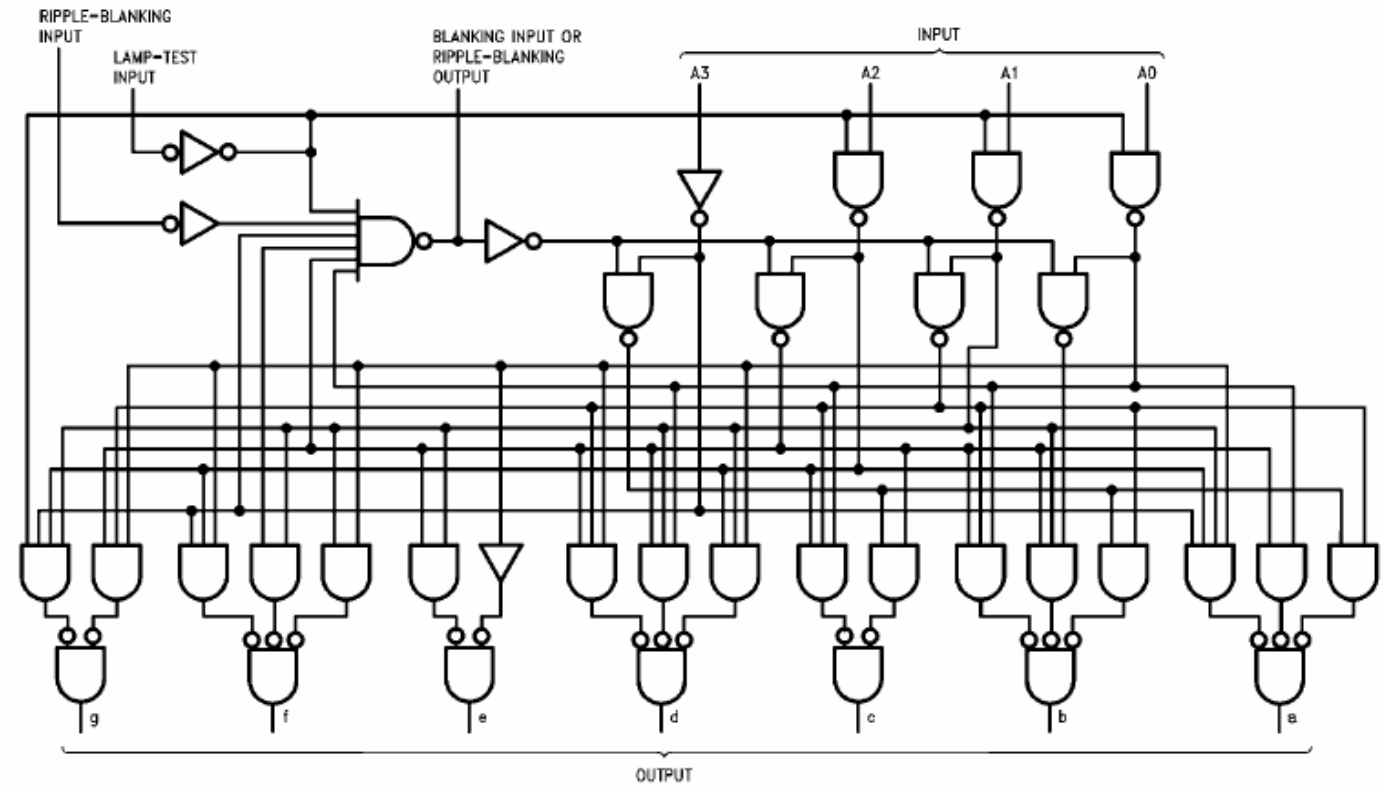
Note 1: $\overline{BI/RBO}$ is wired-AND logic serving as blanking input ($\overline{BI}$) and/or ripple-blanking output ($\overline{RBO}$). The blanking out ($\overline{BI}$) must be open or held at a HIGH level when output functions 0 through 15 are desired, and ripple-blanking input ($\overline{RBI}$) must be open or at a HIGH level if blanking of a decimal 0 is not desired. X = input may be HIGH or LOW.

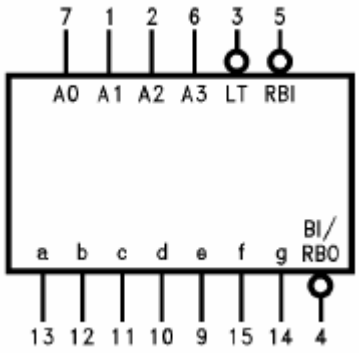
Note 2: When a LOW level is applied to the blanking input (forced condition) all segment outputs go to a LOW level, regardless of the state of any other input condition.

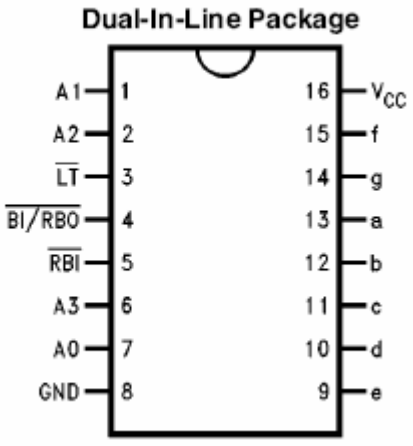
Note 3: When ripple-blanking input ($\overline{RBI}$) and inputs A_0, A_1, A_2, and A_3 are at LOW level, with the lamp test input at HIGH level, all segment outputs go to a LOW level and the ripple-blanking output ($\overline{RBO}$) goes to a LOW level (response condition).

Note 4: When the blanking input/ripple-blanking output ($\overline{BI/RBO}$) is open or held at a HIGH level, and a LOW level is applied to lamp test input, all segment outputs go to a HIGH level.

Logic Diagram







Etablir les équations de a, b, c, …F par karnaugh faire les logigrammes

VII- ADDITIONNEUR : pris sur internet Daniel Robert

VII-1- ADDITION DE DEUX CHIFFRES BINAIRES

Dans le système binaire, on peut représenter n'importe quel nombre comme dans le système décimal et l'on peut effectuer les quatre opérations arithmétiques élémentaires **:** addition, soustraction, multiplication et division.

Comme nous le savons, les trois dernières opérations peuvent être toutes ramenées à l'addition qui est donc la plus importante.

Nous allons tout d'abord rappeler l'addition de deux nombres binaires de **1 bit**, nous obtenons les 4 sommes suivantes **:**

- **0 + 0 = 0**
- **0 + 1 = 1**
- **1 + 0 = 1**
- **1 + 1 = 10**

1er cas **:** les deux chiffres sont **0** et la somme est **0**.

```
  0  <-- 1er terme
+ 0  <-- 2ème terme
  0  <-- Résultat ou somme
```

2ème et 3ème cas **:** un chiffre vaut **0**, l'autre vaut **1 :** la somme vaut **1**.

```
  0                    1  <-- 1er terme
+ 1        ou        + 0  <-- 2ème terme
  1                    1  <-- Résultat ou somme
```

4ème cas **:** Les deux chiffres valent **1** et la somme vaut **10** (**= 210**).

```
          1  <-- 1er terme
        + 1  <-- 2ème terme
Retenue --> 10 <-- Résultat ou somme
```

On remarque que dans les trois premiers cas, il suffit d'un seul chiffre binaire (ou bit) pour indiquer le résultat. Dans le quatrième cas, il faut deux chiffres **:** celui situé le plus à droite est le résultat (ici **0**) et l'autre est la retenue (ici **1**).

La procédure décrite est la même que pour l'addition dans le système décimal. Elle diffère seulement par la quantité de chiffres mise en jeu **:** les deux chiffres binaires contre les dix décimaux.

La figure 1 montre les additions des chiffres **0** et **1** relatives aux deux systèmes.

a)

Termes		Somme	
A	B	C*	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

b)

Termes		Somme
A	B	S
0	0	0
0	1	1
1	0	1
1	1	2

C* est l'initiale de Carry (retenue en anglais)

Fig. 1. - Addition des chiffres 0 et 1 dans les deux systèmes binaire a) et décimal b).

Nous remarquons que les résultats sont les mêmes, bien que dans le système binaire il faille tenir compte de la retenue pour exprimer le résultat deux.

1. 2. - CIRCUIT ADDITIONNEUR

Puisque nous connaissons les règles de l'addition binaire, nous allons voir à présent comment cette opération peut être réalisée par des circuits logiques.

Il faut réaliser un circuit combinatoire (figure 2) dont les deux entrées **A** et **B** et les sorties **S** et **C** répondent à la table de vérité de la figure 1.

On remarque que **S** est à l'état **1** si une seule des entrées est à l'état **1**.

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

Fig. 2. - Dans un additionneur à 2 bits, la somme du bit A et du bit B donne le résultat S et la retenue C.

Nous avons donc affaire à la fonction logique **OU Exclusif**, soit **:**

$$S = A \oplus B$$

D'autre part, on remarque que **C** est à l'état **1** uniquement dans le cas où **A** et **B** sont à l'état **1**.

On en déduit donc que **:**

$$C = A \cdot B$$

Le circuit qui effectue la somme de deux bits peut être obtenue en associant une porte **OU Exclusif** et une porte **ET** comme le montre la figure 3.

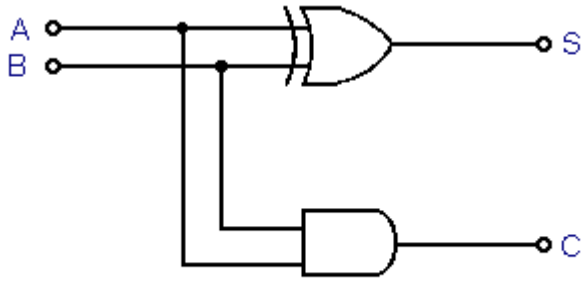


Fig. 3. - Circuit effectuant la somme de 2 bits A et B.

1. 3. - ADDITION DE NOMBRES BINAIRES DE PLUSIEURS CHIFFRES

Le circuit additionneur examiné précédemment est en mesure d'additionner entre eux deux nombres binaires d'un seul chiffre. Pour cette raison, il est appelé demi-additionneur.

En effet, lorsque l'on doit additionner des nombres de plus d'un chiffre, il devient nécessaire de disposer de circuits qui tiennent compte de la retenue de la somme effectuée sur les chiffres de rang immédiatement inférieur.

Pour comprendre cela, nous allons examiner comment on effectue l'addition de deux nombres décimaux, par exemple **:**

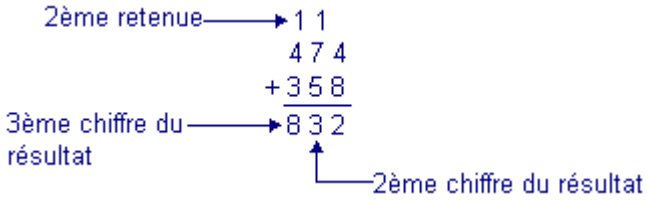
$$\begin{array}{r} 474 \\ +\ 358 \\ = \end{array}$$

Cette opération s'effectue par étapes successives **:** on additionne d'abord les chiffres de droite, puis les suivants en ajoutant l'éventuelle retenue.

Dans un premier temps, on fait l'addition de **4** et **8** dont la somme est **12 ;** on écrit le résultat **2** et on retient **1**.

$$\begin{array}{rl} 1 & \longleftarrow \text{1ère retenue} \\ 474 & \\ +\ 358 & \\ \hline 2 & \longleftarrow \text{1er chiffre du résultat} \end{array}$$

Dans l'étape suivante, on doit faire une addition de **3 chiffres** parce qu'on doit tenir compte de la retenue (ici **1**). La somme de **7** et **5** plus la retenue **1**, donne **13 ;** on écrit donc **3** et on retient **1**.



La dernière étape est semblable à la précédente **:** la somme **8** est cependant sans retenue.

La même procédure s'applique aussi aux nombres binaires.

Il faut donc réaliser un circuit qui puisse additionner les deux chiffres de même rang d'un nombre binaire avec la retenue de l'étage précédent, soit trois chiffres binaires.

L'additionneur complet dispose donc de trois entrées, deux pour les termes et une pour la retenue.

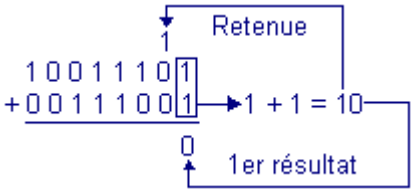
Soit à effectuer la somme des deux nombres binaires de 8 bits suivants **:**

```
  10011101
+ 00111001
=
```

Ce qui donne en code décimal **:**

```
  157
+  57
= 214
```

On part de la dernière position à droite, où se trouvent deux **1**. On effectue la somme de ces deux chiffres selon la table de la figure 1, ce qui donne comme résultat **0** et comme retenue **1**.



A l'étape suivante, on doit additionner **3 chiffres** alors que la table de la figure 1 se limite à la somme de 2 chiffres.

Nous allons donc construire une table indiquant la somme de 3 chiffres.

Avec **3 chiffres**, il y a **8 possibilités** qui vont de **0 + 0 + 0** à **1 + 1 + 1.**

Pour chacune de ces possibilités, il est facile de relever la somme.

Par exemple **:**

- **0 + 0 + 0 = 0** (résultat **0**, retenue **0**)
- **0 + 1 + 1 = 210 = 10** (résultat **0**, retenue **1**)
- **1 + 1 + 1 = 310 = 11** (résultat **1**, retenue **1**).

La table de la figure 4 résume toutes les combinaisons possibles.

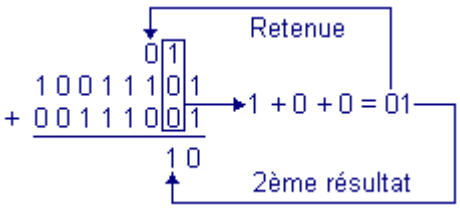
Ai	Bi	Ci	Ci + 1	Si
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig. 4. - Table de la somme de 3 bits.

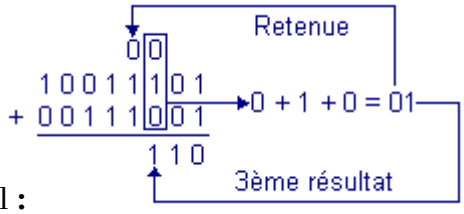
Dans cette table, **Ai** et **Bi** sont les termes de rang **: Ci** est la retenue relative à la somme de **Ai** et **Bi ; Ci + 1** est la retenue relative à la somme de **Ai, Bi** et **Ci**. **Si** est le résultat de la somme **Ai, Bi** et **Ci.**

Revenons maintenant à la somme prise en exemple **;** en utilisant la table de la figure 4, on obtient pour les termes de rang 2 **:**

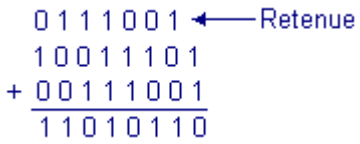
1 + 0 + 0 = 1 avec une retenue égale à **0.**



Si l'on additionne les chiffres suivants, on a **:**



Et ainsi de suite jusqu'au résultat final **:**



Vérifions Le résultat **:**

11010110 = (1 x 128) + (1 x 64) + (0 x 32) + (1 x 16) + (0 x 8) + (1 x 4) + (1 x 2) + (0 x 1) = 128 + 64 + 16 + 4 + 2 = 214.

VII-2- CIRCUIT ADDITIONNEUR COMPLET

Il faut donc réaliser un circuit qui corresponde à la table de vérité de la figure 4, on obtient le schéma de la figure 5 qui représente un additionneur complet.

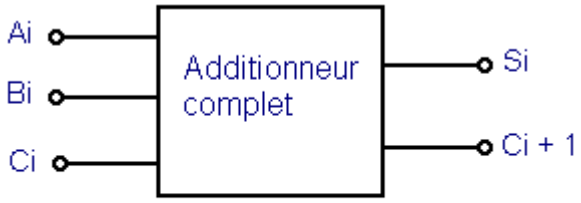


Fig. 5. - Représentation schématique d'un additionneur complet.

Cherchons à présent l'équation de **Ci + 1** et **Si** en utilisant la table de la figure 4.

Pour cela, dressons les tableaux de Karnaugh correspondants reportés à la figure 6.

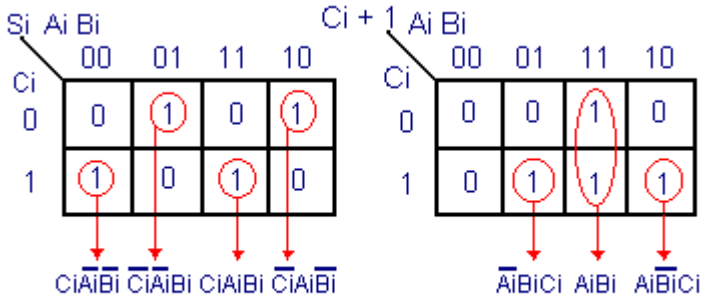


Fig. 6. - Tableau de Karnaugh relatifs à l'additionneur complet.

Du premier tableau de Karnaugh, on tire l'équation de **Si** suivante **:**

- **Si = Ci . $\overline{A}$ i . $\overline{B}$ i + $\overline{C}$ i . $\overline{A}$ i . Bi + Ci . Ai . Bi + $\overline{C}$ i . Ai . $\overline{B}$ i**
- **= Ci . ($\overline{A}$ i . $\overline{B}$ i + Ai . Bi) + $\overline{C}$ i . ($\overline{A}$ i . Bi + Ai . $\overline{B}$ i)**
- **= Ci $\overline{(Ai \oplus Bi)}$ + $\overline{C}$ i . (Ai $\oplus$ Bi)**
- **= Ci $\oplus$ (Ai $\oplus$ Bi)**

Dans le deuxième tableau de Karnaugh, Nous n'avons pas recherché les groupements optimaux et ce, pour pouvoir mettre en évidence la fonction **Ai $\oplus$ Bi** déjà réalisé avec la somme **Si**.

En effet, les 3 groupements indiqués nous donnent l'équation de **Ci + 1** suivante **:**

- **Ci + 1 = AiBi + $\overline{A}$ iBiCi + Ai $\overline{B}$ iCi**
- **= AiBi + Ci ($\overline{A}$ iBi + Ai $\overline{B}$)**
- **= AiBi + Ci (Ai $\oplus$ Bi)**

Les deux expressions **Si** et **Ci + 1** qui viennent d'être calculées, nous déduisons le schéma logique d'un additionneur complet représenté à la figure 7.

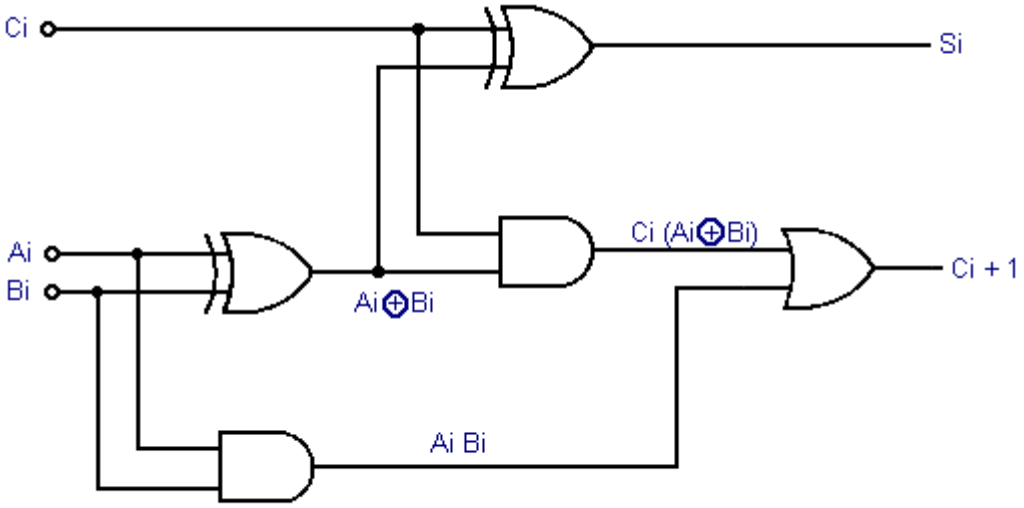


Fig. 7. - Exemple de schéma logique d'un addtionneur complet.

L'additionneur complet est le circuit de base pour effectuer la somme de nombres de plusieurs bits.

Il existe deux méthodes d'addition des nombres binaires.

La première utilise un seul additionneur complet auquel on présente les chiffres de même rang des nombres à additionner. Il s'agit de la **somme en série.**

La deuxième fait appel à autant d'additionneurs complets qu'il y a de chiffres dans les nombres à ajouter. Il s'agit de la **somme en parallèle.**

VII-3- SOMME EN SÉRIE

Les deux nombres (ici de **8 bits**) à additionner sont chargés dans deux registres **A** et **B** comme on le voit à la figure 8. Le résultat de la somme est stocké dans un troisième registre **S**. Il faut aussi disposer d'une bascule synchrone de type **D** qui sert à mémoriser la retenue de la somme partielle précédente.

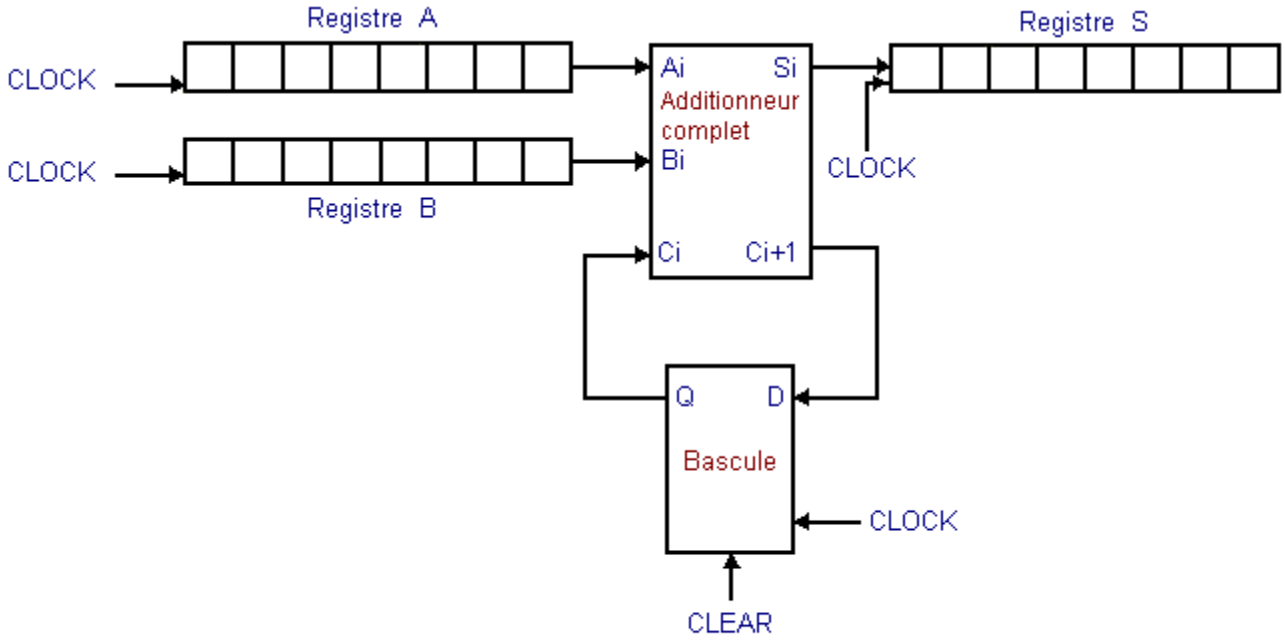


Fig. 8. - La somme en série nécessite un seul additionneur complet mais il faut ajouter 3 registres et une bascule D.

Les 3 registres et la bascule sont commandés par le même signal d'horloge qui synchronise toute l'opération.

Le fonctionnement du circuit est le suivant. Au début, la bascule doit être mise à **0** en activant l'entrée **CLEAR**. Par contre, les 3 registres n'ont pas besoin d'être remis à **0**.

Les deux termes de la somme sont chargés dans les registres **A** et **B** avec une première impulsion d'horloge. Les deux premiers chiffres de chaque terme (**L.S.B.**) sont alors présents à la sortie des registres et donc aux entrées de l'additionneur.

Ainsi, on trouve à la sortie **Si** le premier résultat partiel et à la sortie **Ci + 1** la première retenue.

La situation est celle qui apparaît à la figure 9 ou l'on additionne les deux nombres de l'exemple précédent.

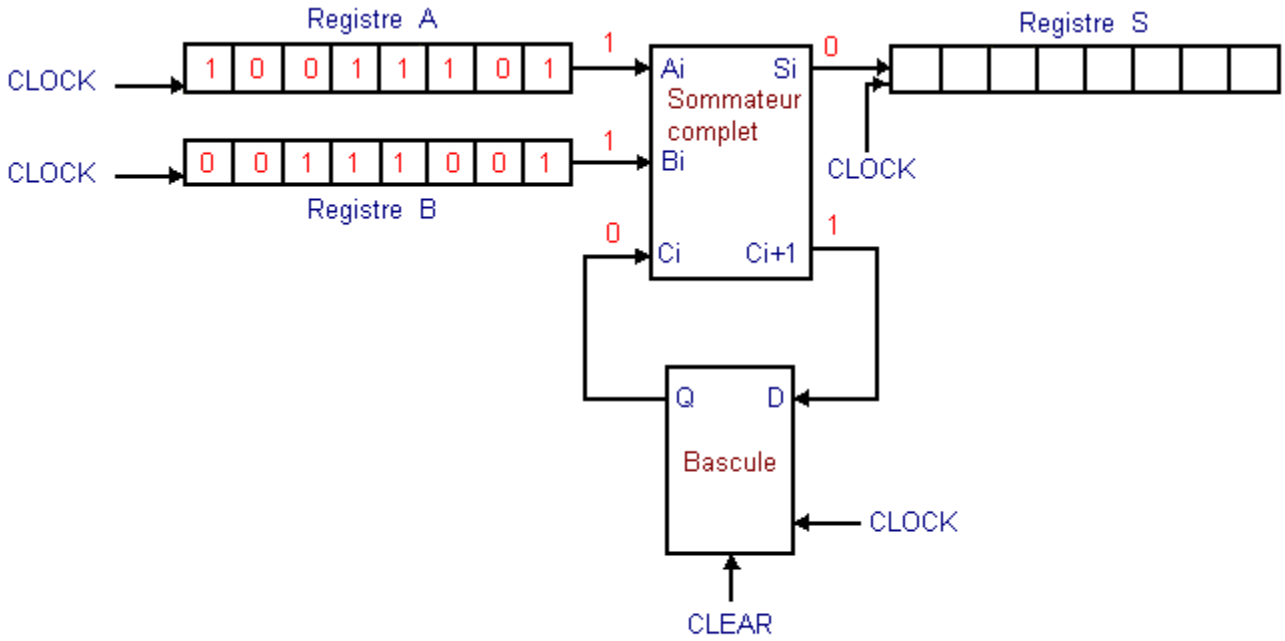


Fig. 9. - Au début de l'addition des 2 nombres, la bascule est mise à 0 et les termes sont chargés dans les registres A et B.

La deuxième impulsion d'horloge produit les faits suivants **:**

- Le premier résultat partiel est stocké dans le premier étage du registre **S**.
- La première retenue est mémorisée par la bascule.
- Les contenus des registres **A** et **B** se décalent d'un étage vers la droite **;** ainsi les chiffres de poids immédiatement supérieur se présentent aux entrées du sommateur. Nous nous trouvons alors dans la situation de la figure 10. La bascule est désormais symbolisée par un carré à l'intérieur duquel est inscrit son état.

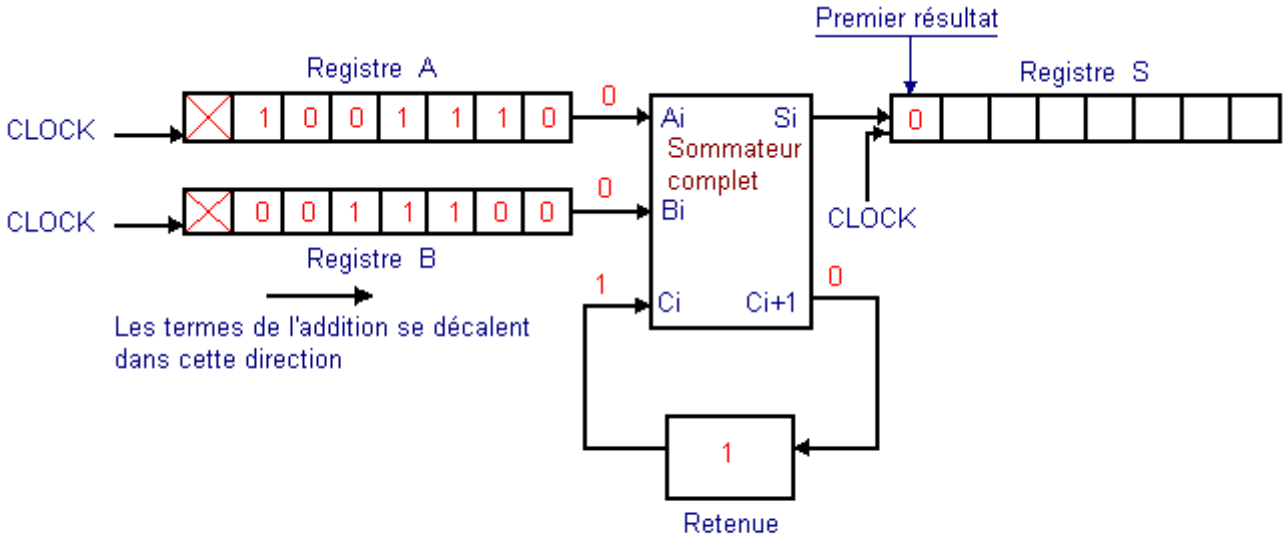


Fig. 10. - Le premier résultat partiel est stocké dans le premier étage du registre S et les termes sont décalés d'un cran vers la droite dans les registres A et B.

La donnée présente sur l'entrée série des registres est sans importance. L'addition se déroule de façon identique pour les chiffres suivants.

Après 9 impulsions d'horloge (une pour le chargement de **A** et **B** et 8 pour effectuer la somme), le résultat de la somme se trouve stocké dans le registre **S** tandis que la sortie de la bascule indique la retenue. Nous nous trouvons dans la situation de la figure 11.

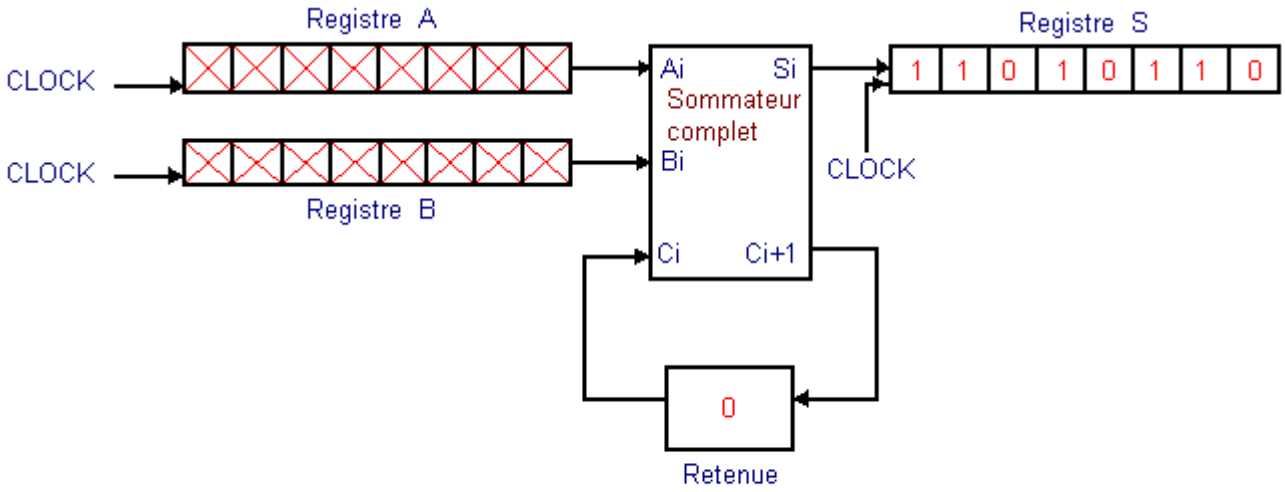


Fig. 11. - A la fin de l'addition, le contenu de S indique le résultat de la somme et l'état de la bascule indique la retenue.

L'addition prise en exemple a pour retenue finale **0**, ce qui signifie que le résultat **1101 0110** est juste.

Si la bascule est à l'état **1**, cela signifie que la dernière somme a donné lieu à une retenue de **1**. On dépasse ainsi la capacité du circuit, cela est désigné par le terme anglais **overflow** qui signifie déborder.

Il y a overflow lorsque le nombre qui est le résultat de la somme a plus de bits que ceux qui peuvent être contenus dans le registre (dans notre cas 8).

Avec **8 bits**, le nombre le plus élevé que l'on peut représenter est **: 1111 1111$_2$ = 255$_{10}$**.

Avec des registres à **8 bits**, on peut donc additionner les nombres compris entre **0** et **255** (exprimés en code décimal), mais le résultat de leur somme ne doit pas lui-même dépasser **255**.

Dans le cas contraire, on obtiendrait un résultat qui, pour être stocké, nécessiterait un registre de 9 bits.

En utilisant un ordinateur ou un circuit sommateur, il est toujours nécessaire de faire attention à ne jamais en dépasser la capacité. L'overflow donne des résultats erronés. Supposons que l'on effectue la somme suivante **:**

```
  1 1 1 1  0 0 0 0          C'est-à-dire      240
+ 0 1 0 1  0 0 0 0                          +  80
---------------------                       -----
1 0 1 0 0  0 0 0 0                          320
```

Le dernier chiffre à gauche du résultat ne trouvant pas de place dans le registre **S** est perdu. Le résultat qui est indiqué par le contenu de **S** est **0100 0000**, ce qui équivaut à **64** en code décimal et non **320** qui est le vrai résultat.

Pour savoir s'il y a dépassement, il suffit d'examiner l'état de la bascule à la fin de l'addition **:** si elle est à l'état **0**, le résultat est juste **;** par contre, si elle est à l'état **1**, cela indique qu'il y a eu une retenue de **1** lors de la dernière addition et que l'on a dépassé la capacité du circuit.

On peut faire l'économie du registre **S** en rebouclant la sortie **S** de l'additionneur sur l'entrée série du registre **A** ou **B**.

Si l'on relie par exemple la sortie **S** à l'entrée série du registre **A**, comme illustré à la figure 12, le résultat de l'addition apparaîtra dans le registre **A**.

En effet, à chaque impulsion d'horloge, le résultat partiel de chaque somme se trouve décalé dans le registre **A**.

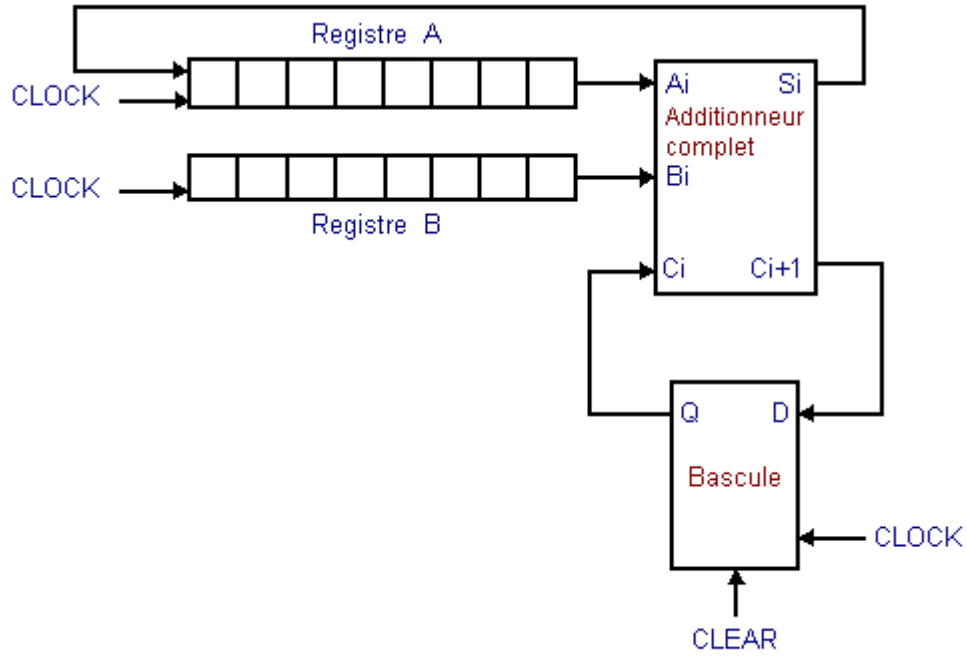


Fig. 12. - Le registre A peut aussi servir pour stocker le résultat de la somme.

La méthode de la somme en série est la plus proche de notre façon usuelle d'effectuer des additions **:** on additionne un chiffre à la fois en partant de celui de plus faible poids.

Toutefois, elle est plutôt lente parce qu'elle requiert autant d'impulsions d'horloge qu'il y a de chiffres à additionner.

Pour plus de rapidité, on fait appel à la méthode de la somme en parallèle où tous les chiffres sont additionnés simultanément.

Selon le mode de calcul de la retenue, on distingue la somme en parallèle avec retenue série et la somme en parallèle avec retenue anticipée.

1. 6. - SOMME EN PARALLÈLE AVEC RETENUE SÉRIE

La figure 13 représente un circuit de somme en parallèle de 8 bits avec retenue série.

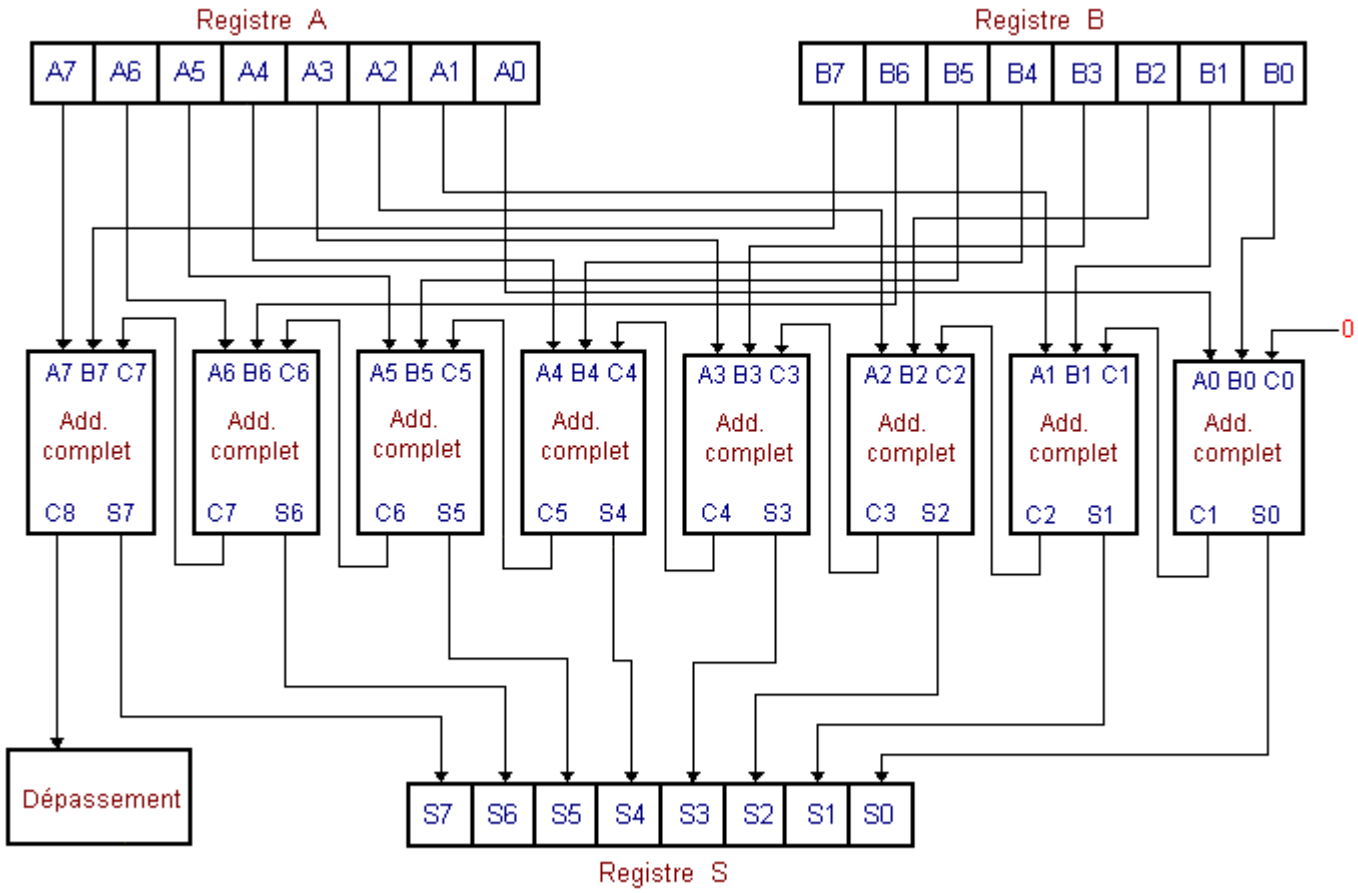


Fig. 13. - Circuit de somme en parallèle avec retenue série.

Nous constatons qu'un circuit de somme en parallèle nécessite autant d'additionneurs complets qu'il y a de chiffres à additionner.

D'autre part, puisque la sortie retenue d'un additionneur est reliée à l'entrée retenue du suivant, le circuit sommateur de la figure 13 est dit à **retenue série**. Il est à noter que l'entrée retenue **C0** du premier additionneur doit être portée à l'état **0**.

La méthode de la somme en parallèle est beaucoup plus rapide que celle de la somme en série et le temps total pour effectuer l'opération dépend essentiellement du temps requis pour la propagation de la retenue.

En effet, même si tous les chiffres sont additionnés simultanément, la retenue doit se propager du premier au dernier additionneur.

Ainsi, le résultat présenté sur les **8 sorties** et sur la retenue **C8** ne sera exact que lorsque cette propagation se sera effectuée.

Le mécanisme de l'addition est le suivant.

Le premier sommateur additionne les deux chiffres **A0** et **B0** et génère la somme **S0** et la retenue **C1**.

Le deuxième sommateur additionne les chiffres **A1** et **B1** avec la retenue **C1** produite par le premier sommateur. Il ne pourra additionner **A1, B1** et **C1** seulement lorsque la retenue **C1** de la première somme aura été calculée par le premier sommateur.

Il faut donc attendre un certain temps que la retenue se soit propagée d'étage en étage pour que la somme **S7** et la retenue **C8** soient établis (les sommes **S0** à **S6** seront déjà établies). Avant ce temps, le résultat contenu dans **S** n'est pas forcément correct.

Ce mécanisme, semblable à celui rencontré dans les compteurs asynchrones, présente le même avantage (simplicité du circuit) et le même inconvénient (lenteur).

La méthode de somme en parallèle avec propagation de la retenue est cependant plus rapide que celle de la somme en série.

Le temps nécessaire pour qu'un additionneur complet calcule la retenue est très court, dans le cas des circuits **C-MOS** quelques dizaines de nanosecondes.

Toutefois, le temps total de l'addition est le produit de ce temps par le nombre de chiffres à additionner.

Il ne peut plus alors être négligé surtout dans les ordinateurs qui doivent pouvoir effectuer des millions d'addition par seconde. On a recours à la méthode de somme en parallèle à retenue anticipée.

1. 7. - SOMME EN PARALLÈLE À RETENUE ANTICIPÉE

Pour effectuer la somme plus rapidement, il faut compliquer le circuit précédent.

On se base sur le fait que les termes de la somme sont connus et disponibles avant même que commence l'opération d'addition. On peut alors calculer, **en anticipant,** la retenue pour chaque étage indépendamment des étages précédents. Il s'agit de pouvoir disposer de toutes les retenues simultanément et dans un temps le plus court possible.

Autrement dit, il faut calculer la retenue **C1** à partir des bits **A0, B0** et **C0**, la retenue **C2** à partir des bits **A0, B0, C0, A1** et **B1** et ainsi de suite.

La figure 14 montre le synoptique d'un additionneur 4 bits à retenue anticipée.

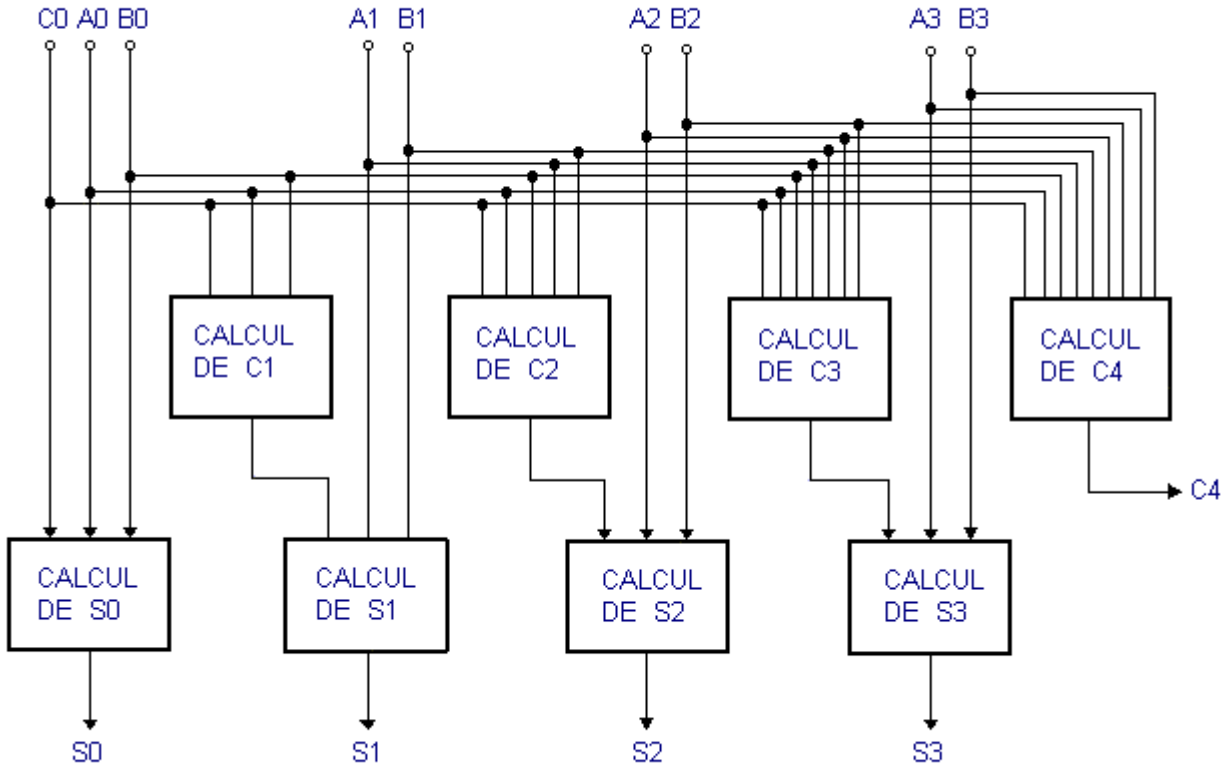


Fig. 14. - Synoptique d'un additionneur 4 bits à retenue anticipée.

Pour effectuer le calcul des retenues de façon anticipée, il faut transformer l'équation de la retenue **Ci + 1** vu précédemment.

$$Ci + 1 = Ai \overline{B}\, iCi + AiBi + \overline{A}\, iBiCi$$

Puisque **Ci + 1** vaut **1** lorsque **Ai = Bi = Ci = 1**, on peut ajouter les termes **AiBiCi** à l'expression de **Ci + 1** autant de fois que l'on veut (ici 2 fois).

- D'où $Ci + 1 = Ai \overline{B}\, iCi + AiBiCi + AiBi + \overline{A}\, iBiCi + AiBiCi$
- $= AiCi (\overline{B}\, i + Bi) + AiBi + BiCi (\overline{A}\, i + Ai)$
- Soit $Ci + 1 = AiCi + AiBi + BiCi$
- $= AiBi + Ci (Ai + Bi)$

Posons **:** produit **AiBi = pi** et somme **Ai + Bi = Si**

D'où **Ci + 1 = pi + CiSi**

L'expression de la retenue du premier étage devient **:**

$$C1 = p0 + C0S0 \quad (1)$$

et celle du deuxième étage **:**

$$C2 = p1 + C1S1$$

Remplaçons **C1** par sa valeur calculée en ① dans cette expression de **C2 :**

- **C2 = p1 + (po + C0S0) S1**
- **C2 = p1 + poS1 + C0S0S1** ②

De même **:**

- **C3 = p2 + C2S2**
- **= p2 + (p1 + p0S1 + C0S0S1) S2**
- **C3 = p2 + p1S2 + p0S1S2 + C0S0S1S2** ③
- **C4 = p3 + C3S3**
- **= p3 + (p2 + p1S2 + p0S1S2 + C0S0S1S2) S3**
- **C4 = p3 + p2S3 + p1S2S3 + p0S1S2S3 + C0S0S1S2S3** ④

Les expressions ①, ②, ③, et ④ des retenues **C1, C2, C3** et **C4** sont remarquables par le fait qu'elles réclament le même temps de calcul et qu'elles ne tiennent pas compte de la retenue de l'étage précédent (donc pas de retard dû à la propagation de la retenue.

Pour expliquer cela, nous allons parler de **«couche logique».**

Une couche logique correspond au temps de propagation d'une porte élémentaire type **ET** ou **OU**.

Par exemple, le calcul de **C1 = p0 + C0S0** nécessite **3 couches logiques** comme le montre la figure 15.

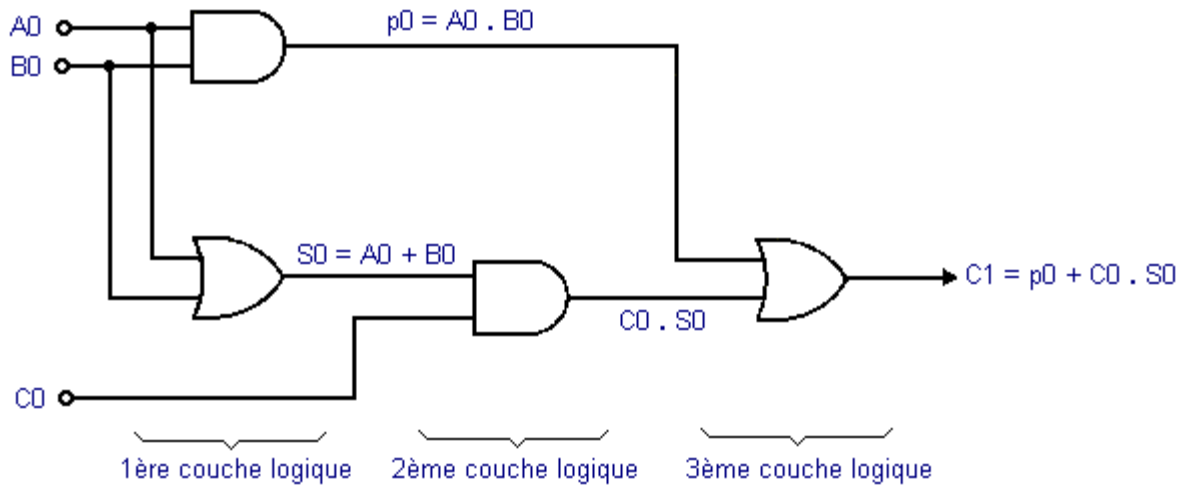


Fig. 15. - Schéma logique montrant le calcul de C1.

Bien que les expressions ②, ③ et ④ des retenues **C2, C3** et **C4** soient plus complexes, celles-ci ne nécessitent pour leur calcul que **3 couches logiques** comme **C1**.

Nous allons voir maintenant un exemple d'additionneur intégré 4 bits à retenue anticipée **:** le **7483**.

La figure 16 présente le brochage et le schéma logique du circuit intégré **7483**.

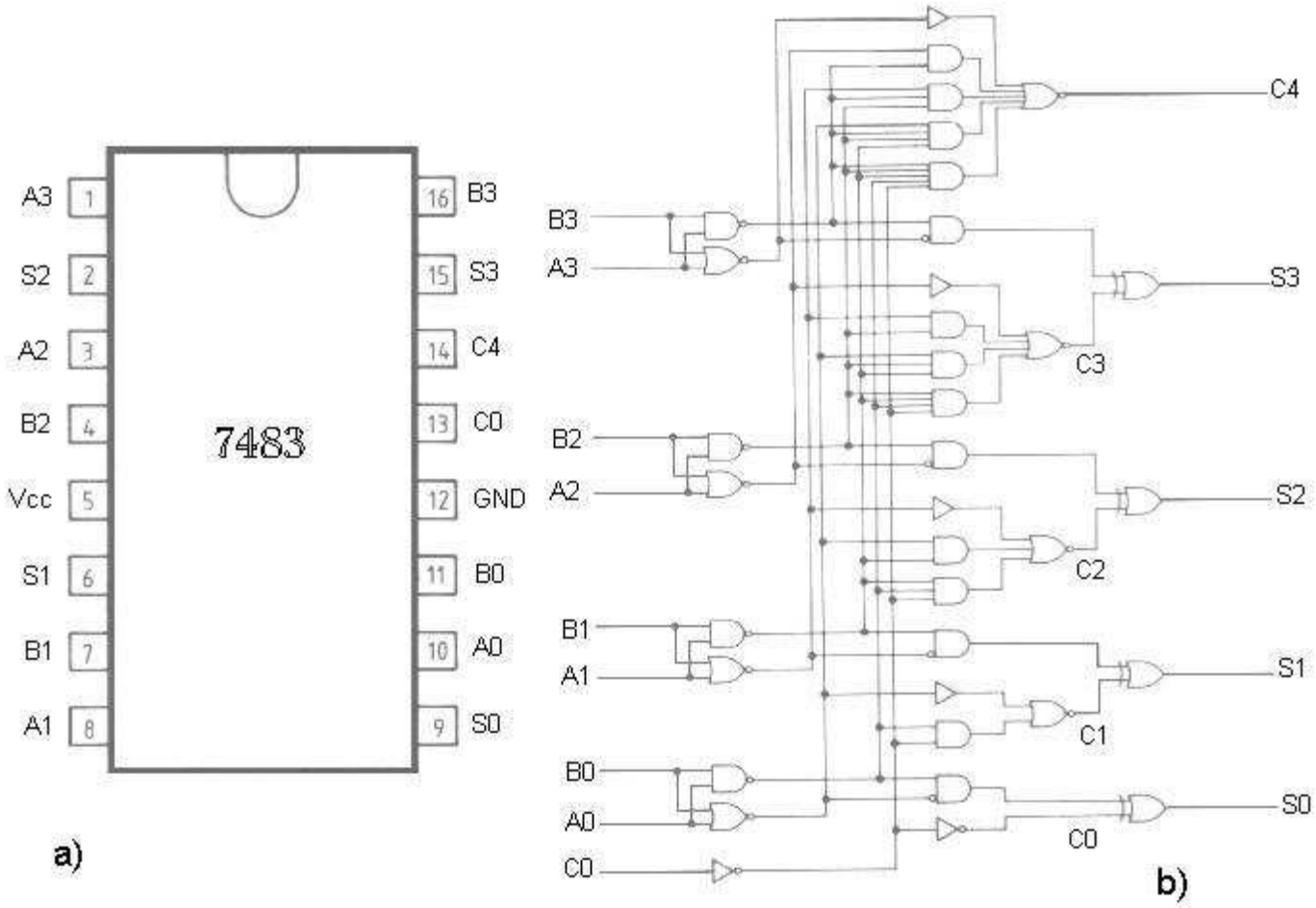


Fig. 16. - Brochage a) et schéma logique b) du circuit intégré 7483.

Les temps de propagation des différentes entrées vers les différentes sorties du circuit sont rassemblés dans le tableau de la figure 17.

Fig. 17. - Temps maximaux de propagation du circuit intégré 7483.

Entrées	Sorties	Temps maximal de propagation (en ns)
C0	Si	21
Ai ou Bi	Si	24
C0	C4	16
Ai ou Bi	C4	16

Avec ce circuit intégré, on additionne **2 nombres de 4 bits en 24 ns maximum**.

Il est à noter que le circuit intégré **74LS83** qui est un additionneur de 4 bits à retenue série effectue la même opération en **72 ns maximum, soit 3 fois plus**.

Si l'on veut additionner 2 nombres de plus de 4 bits, il faut utiliser plusieurs additionneurs intégrés et les relier en cascade.

Pour exemple, la figure 18 montre la mise en cascade de **2 additionneurs 4 bits type 7483** pour obtenir un **additionneur 8 bits**. Il suffit de relier la sortie **C4** du premier additionneur à l'entrée **C0** du second.

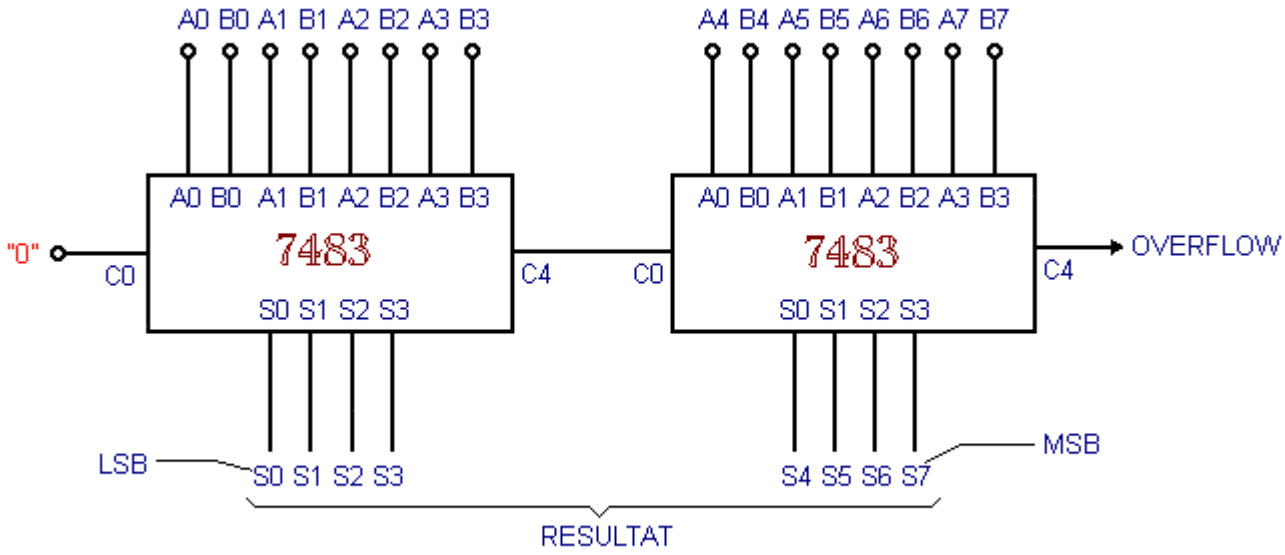


Fig. 18. - Mise en cascade de 2 additionneurs de 4 bits.

L'additionneur obtenu n'est que partiellement à retenue anticipée.

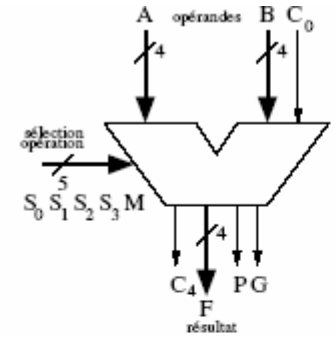
En effet, on retrouve le mécanisme de la retenue à propagation série dû à la **sortie C4** reliée à **l'entrée C0**.

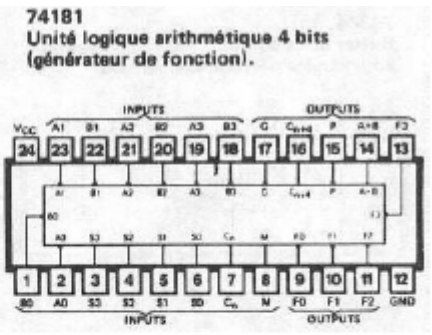
D'après le tableau de la figure 17, la sortie **C4** du premier **7483** est disponible au bout de **16 ns**. D'autre part, comme les sorties **S4** à **S7** sont disponibles **21 ns** après l'apparition de la retenue en **C0** du deuxième **7483**, nous en déduisons que le résultat de la somme des **2 nombres de 8 bits** est disponible après **16 + 21 = 37 ns** maximum.

Chaque nouvel additionneur **7483** mis en cascade apporte un retard supplémentaire de **21 ns**. Ainsi avec **3 circuits 7483**, l'addition de **2 nombres de 12 bits** nécessitera **37 + 21 = 58 ns** maximum.

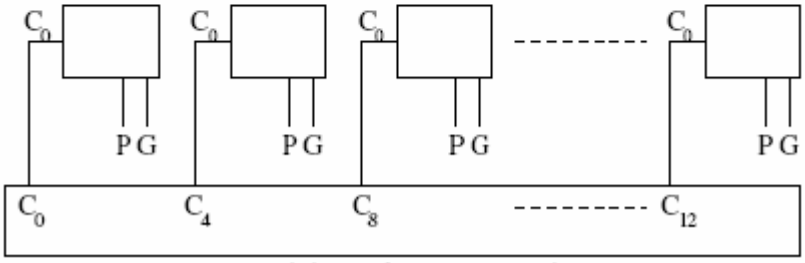
VIII- ALU

Utilisée dans pratiquement tous les systèmes informatiques, elle réalise des opérations arithmétiques (addition, soustraction, etc.) et logiques (ET, OU, etc.). C'est un circuit programmable : les relations entre les données en sortie et les données en entrée sont modifiables.





Les sorties P et G servent à la mise en cascade des ALUs, et donc au calcul de retenue anticipée.



Générateur de retenue anticipée

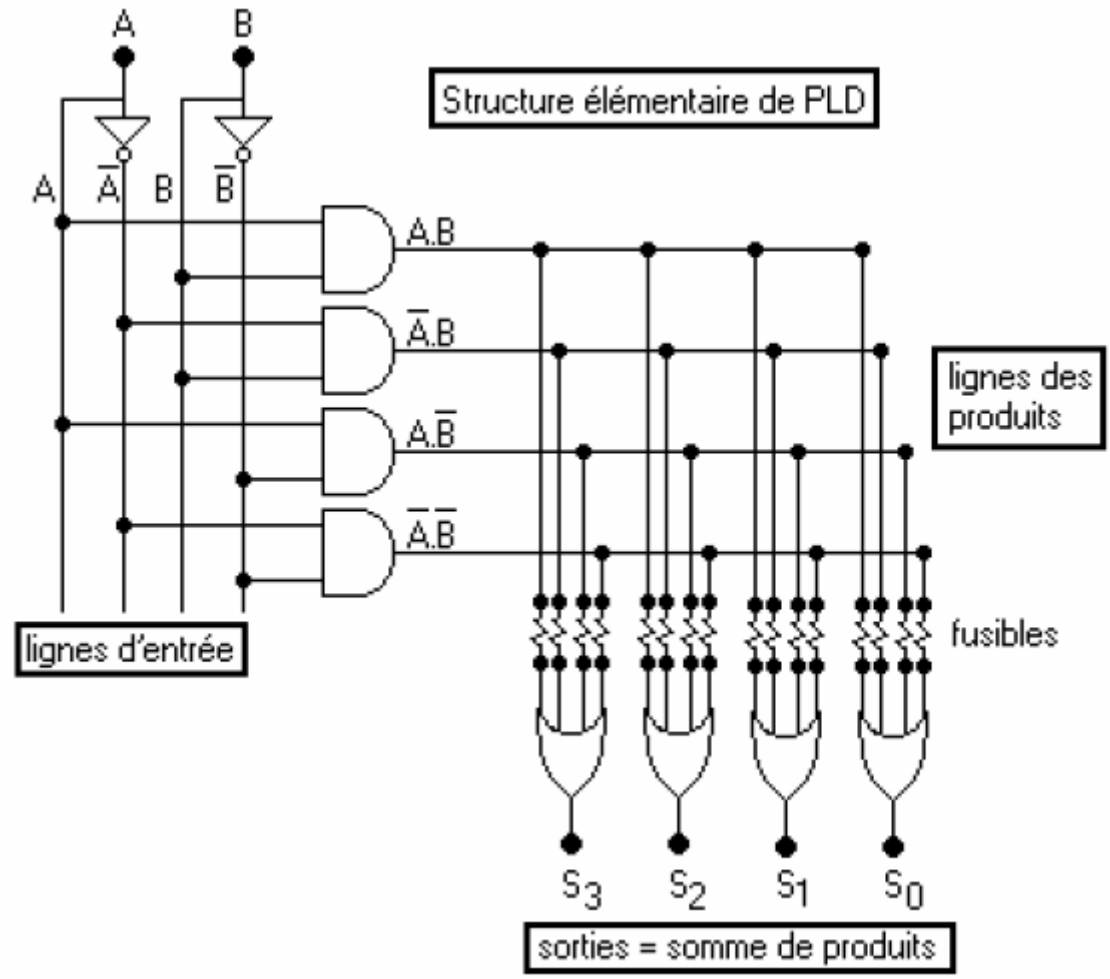
IX- COMPOSANTS LOGIQUES PROGRAMMABLES:

Nous avons vu qu'il existait des circuits combinatoires permettant de réaliser des fonctions complexes données (multiplexage, addition, etc…). On peut ainsi réaliser, au moyen d'un seul composant, ce qui demanderait d'utiliser un grand nombre de portes élémentaires.

Les composants logiques programmables (PLD), permettent de réaliser des applications complexes personnalisées (autres celles que nous venons d'énoncer), au moyen d'un seul composant. Ce dernier renferme un très grand nombre de transistors et de portes raccordés entre eux. Certaines liaisons sont des fusibles qui peuvent être fondus par l'intermédiaire d'un dispositif extérieur. On a donc bien un composant programmable.

De par leur structure standardisée (quel que soit la fonction à réaliser, on part du même élément), ces composants sont peu coûteux et offrent une grande souplesse dans la réalisation de circuits combinatoires (on réalise directement ce dont on a besoin). On les emploiera donc de préférence à des associations de portes discrètes qui sont techniquement fastidieuses à mettre ne oeuvre.

• Structure générale: La structure de base de tout circuit programmable se présente sous laforme suivante



On observe un réseau de portes AND, un réseau de portes OR capable de fournir n'importe quelle combinaison des deux variables A et B ainsi que de leurs compléments.

On remarque les fusibles en entrée des portes OR. Suivant les fusibles que l'on choisit de griller, on obtiendra différentes les valeurs choisies en sortie.

Plus le nombre d'entrées sera grand, plus le circuit mettra en jeux un grand nombre de portes intégrées.

Par la suite, on utilisera le formalisme suivant, afin de simplifier les schémas de principe et de les rendre plus lisibles. On ne fait apparaître qu'une seule ligne d'entrée pour les portes sur laquelle on trouve toutes les données et on représente différemment les fusibles et les liaisons permanentes.