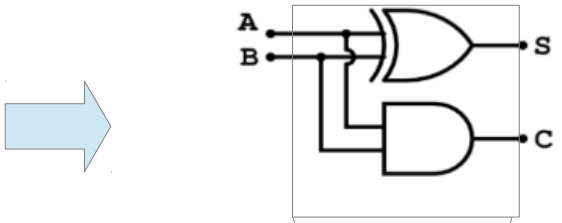
Quelques exemples de circuits combinatoires usuels

Additionneur

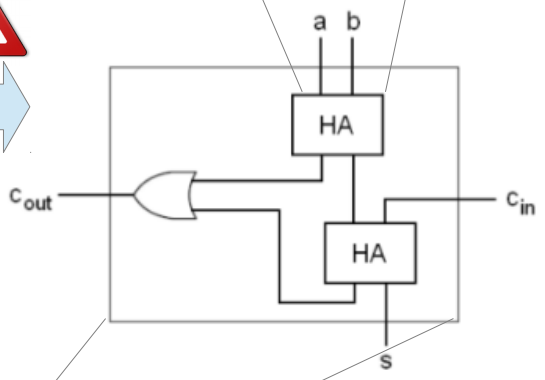
On établit la table de vérité.

a	b	f	cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

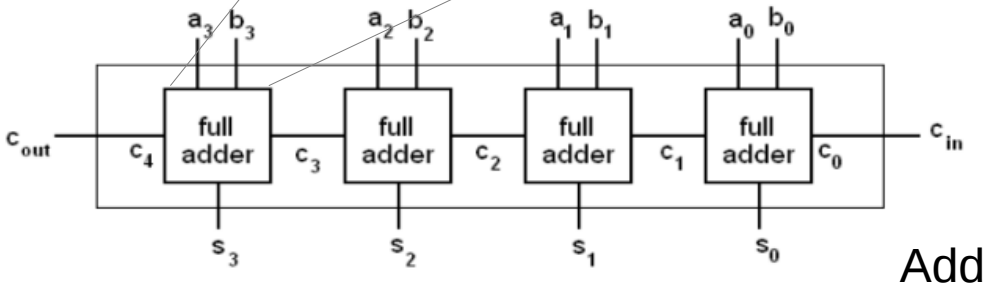


Demi-additionneur

a	b	cin	S	cout
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



Additionneur 1 bit



Additionneur 4 bit
(ripple carry)

Additionneur

a	b	cin	S	cout
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

Rappel. On a :

$$\overline{a \otimes b} = \overline{a.\bar{b} + \bar{a}.b} = (\bar{a} + b).(a + \bar{b}) = \bar{a}.\bar{b} + a.b$$

On a :

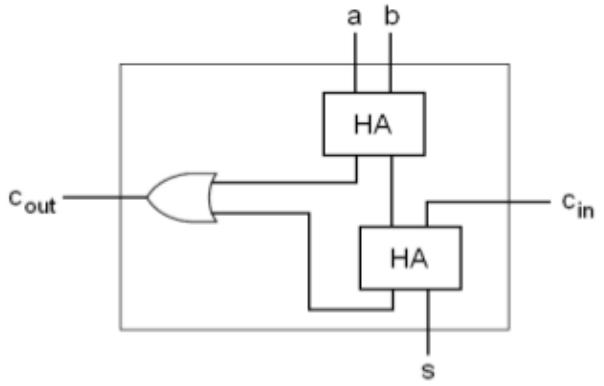
$$S = \bar{a}.\bar{b}.c_i + \bar{a}.b.\bar{c_i} + a.\bar{b}.\bar{c_i} + a.b.c = \bar{a}.(a \otimes c_i) + a.(b \otimes \bar{c_i})$$

$$S = a \otimes b \otimes c_i$$

De même, on trouve :

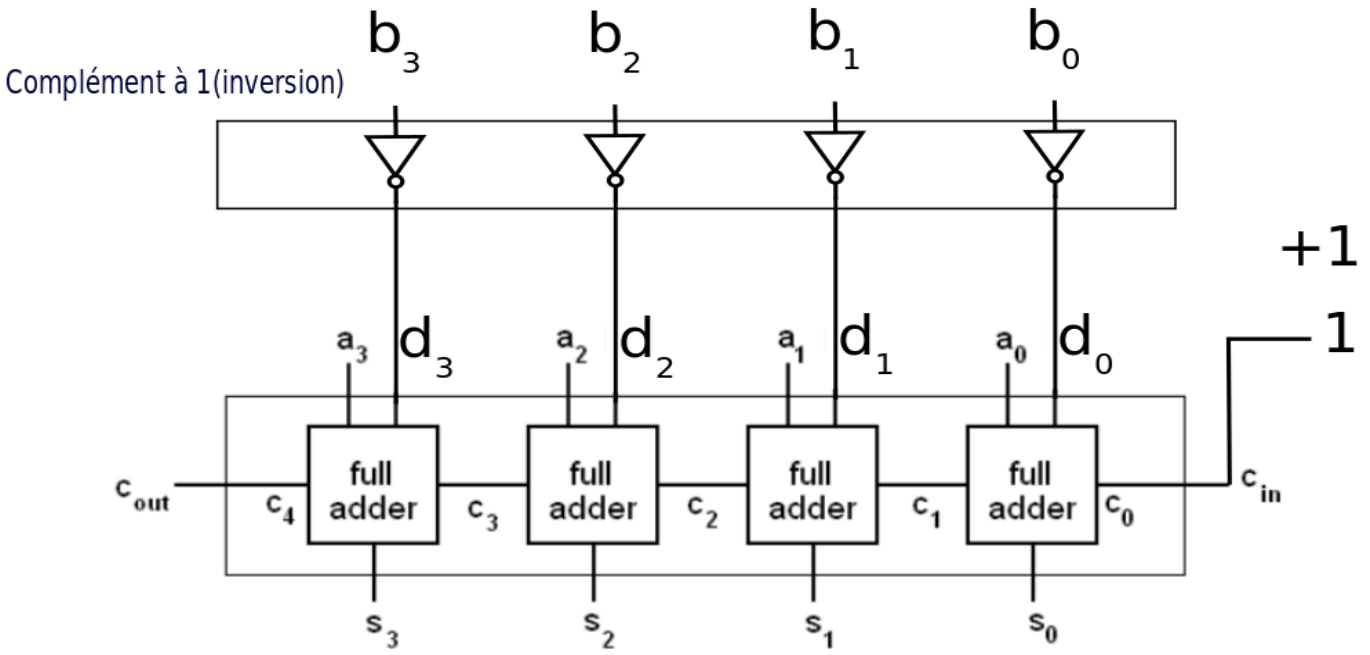
$$c_o = a.b + (a \otimes b).c_i$$

On peut donc utiliser judicieusement le demi additionneur précédent de la manière suivante.



Soustracteur

$$
\begin{array}{rl}
 & A_3\,A_2\,A_1\,A_0 \\
+ & D_3\,D_2\,D_1\,D_0 \quad \leftarrow \text{2's complement of } B_3\,B_2\,B_1\,B_0 \\
\hline
 & C_4\,S_3\,S_2\,S_1\,S_0 \\
 & X \leftarrow \text{carry is ignored}
\end{array}
$$

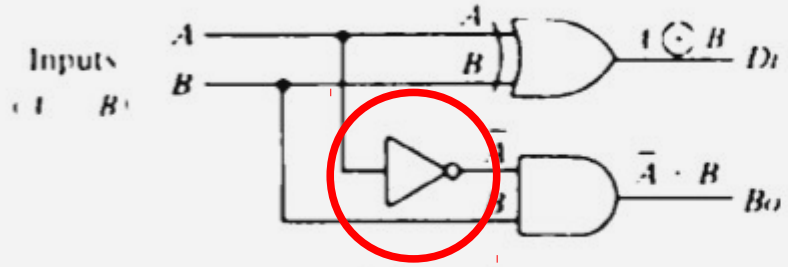


Soustracteur

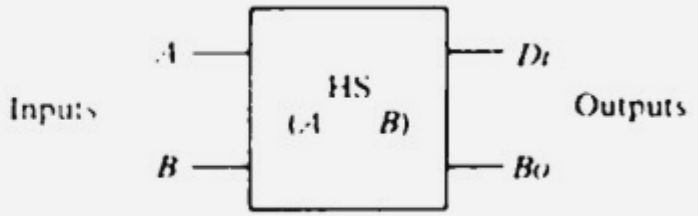
par la table de vérité

0	–	0	=	0	
0	–	1	=	1	and borrow 1
1	–	0	=	1	
1	–	1	=	0	

A	*B*	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
A – *B*		*Di*	*Bo*



(a) Logic diagram



(b) Block symbol

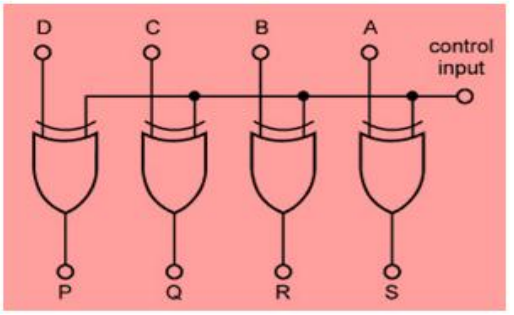
A comparer au HA

Additionneur-soustracteur

When DCBA = 1011 and C.I. = 0, then, PQRS = 1011

When DCBA = 1011 and C.I. = 1, then, PQRS = 0100, which is the 1's complement of 1011.

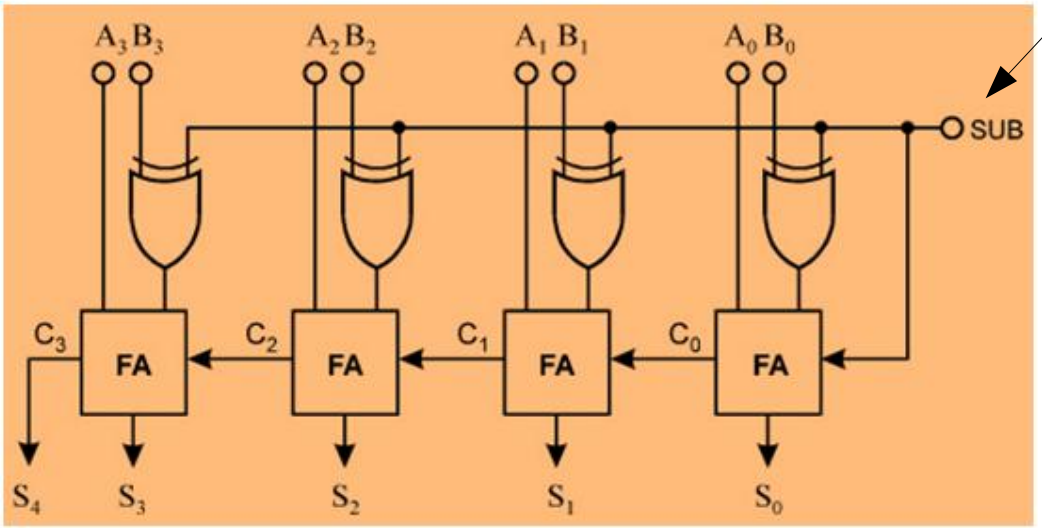
1



Signal de contrôle permettant la configuration du composant en additionneur Ou en additionneur

application

2

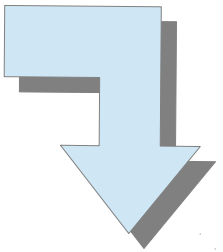


Multiplieur

multiplicand 1101 (13)
multiplier * 1011 (11)

1101
1101
0000
1101
10001111 (143)

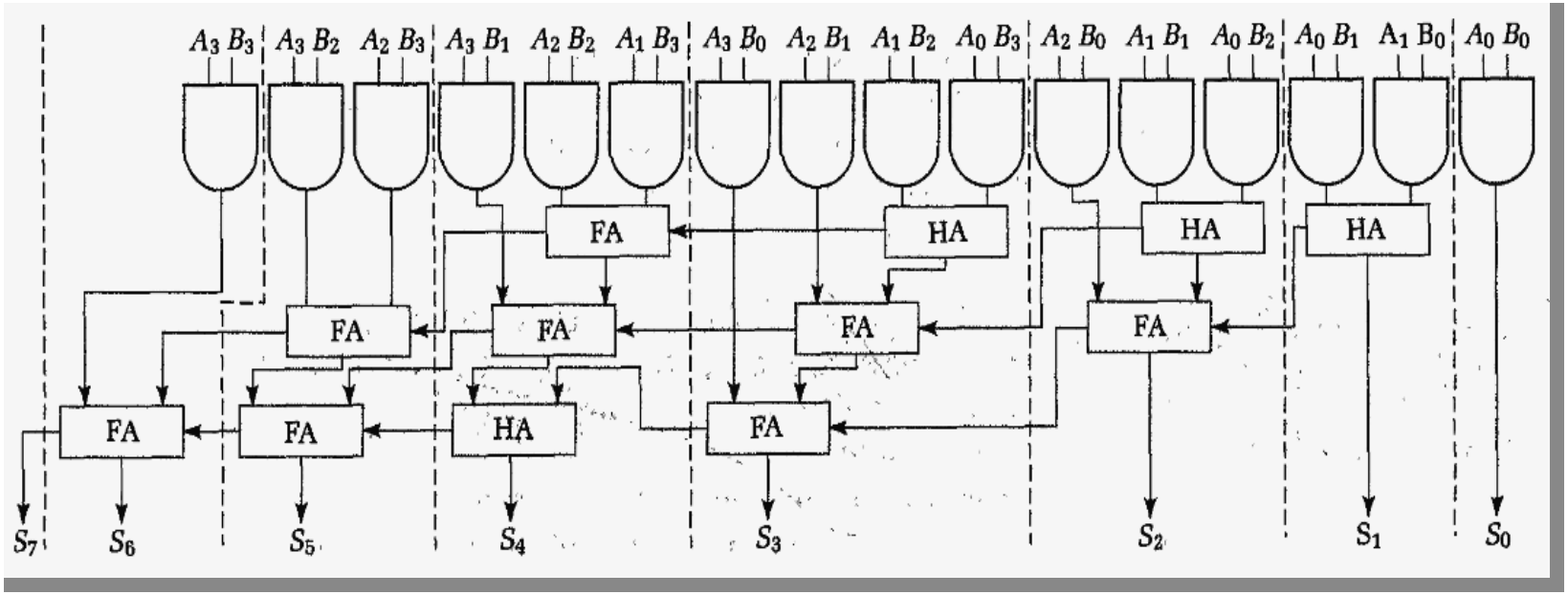
128 + 8 + 4 + 2 + 1 = 143



			A_3	A_2	A_1	A_0
			B_3	B_2	B_1	B_0
			$A_3 \bullet B_0$	$A_2 \bullet B_0$	$A_1 \bullet B_0$	$A_0 \bullet B_0$
		$A_3 \bullet B_1$	$A_2 \bullet B_1$	$A_1 \bullet B_1$	$A_0 \bullet B_1$	
	$A_3 \bullet B_2$	$A_2 \bullet B_2$	$A_1 \bullet B_2$	$A_0 \bullet B_2$		
$A_3 \bullet B_3$	$A_2 \bullet B_3$	$A_1 \bullet B_3$	$A_0 \bullet B_3$			
S_6	S_5	S_4	S_3	S_2	S_1	S_0

Multiplieur

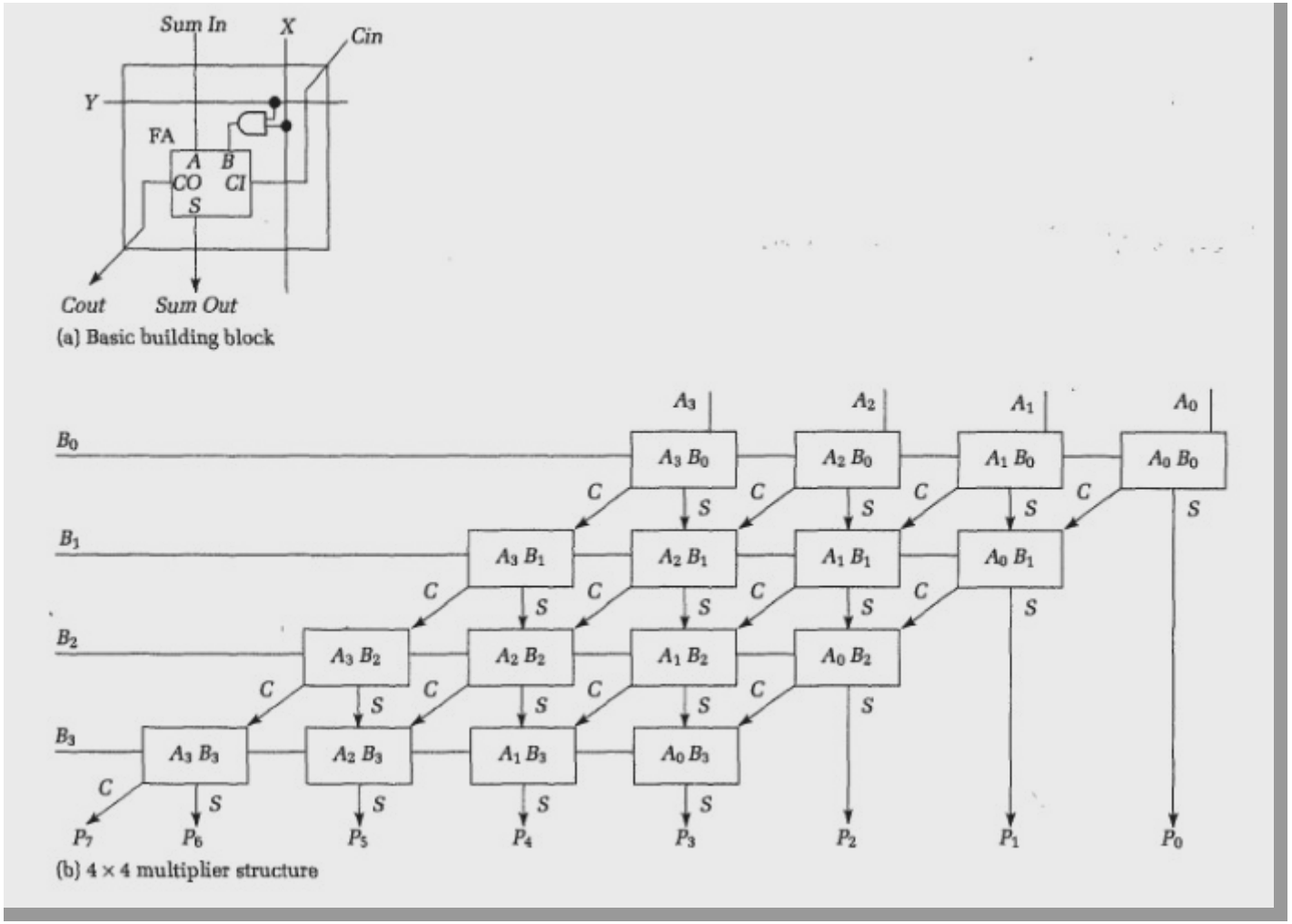
			A_3	A_2	A_1	A_0
			B_3	B_2	B_1	B_0
			$A_3 \bullet B_0$	$A_2 \bullet B_0$	$A_1 \bullet B_0$	$A_0 \bullet B_0$
		$A_3 \bullet B_1$	$A_2 \bullet B_1$	$A_1 \bullet B_1$	$A_0 \bullet B_1$	
	$A_3 \bullet B_2$	$A_2 \bullet B_2$	$A_1 \bullet B_2$	$A_0 \bullet B_2$		
$A_3 \bullet B_3$	$A_2 \bullet B_3$	$A_1 \bullet B_3$	$A_0 \bullet B_3$			
S_6	S_5	S_4	S_3	S_2	S_1	S_0



(Randy Katz)

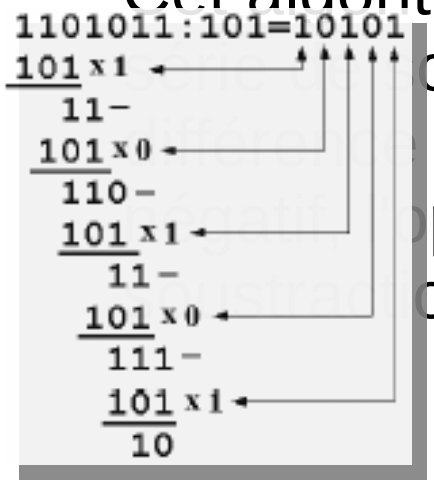
Multiplieur

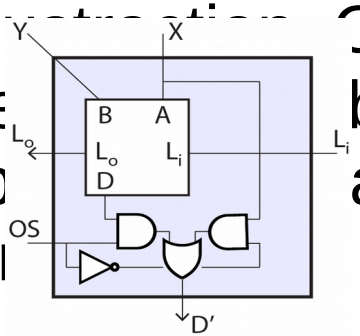
version plus évoluée : topologie régulière

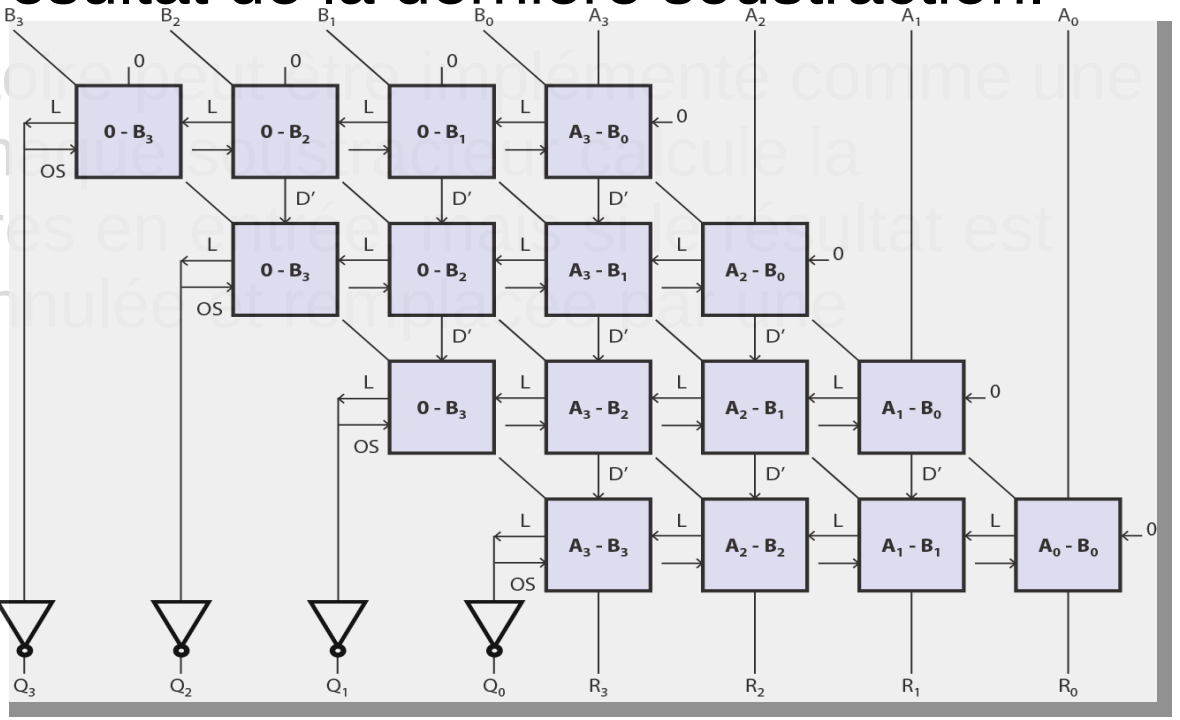


Diviseur

- Un peu plus complexe, mais on repart de la division classique
- Lors du calcul de A / B, on soustrait le divieur B de manière répétitive aux bits du dividende A, après l'avoir multiplié par ’1′ ou ’0′. Ce bit de multiplication (’1′ or ’0′) est sélectionnée pour chaque étape de soustraction de telle manière que le résultat de soustraction n'est jamais négatif.
- Le résultat est composé des bits de multiplication successifs, alors que le reste est le résultat de la dernière soustraction.
- Cet algorithme combinat

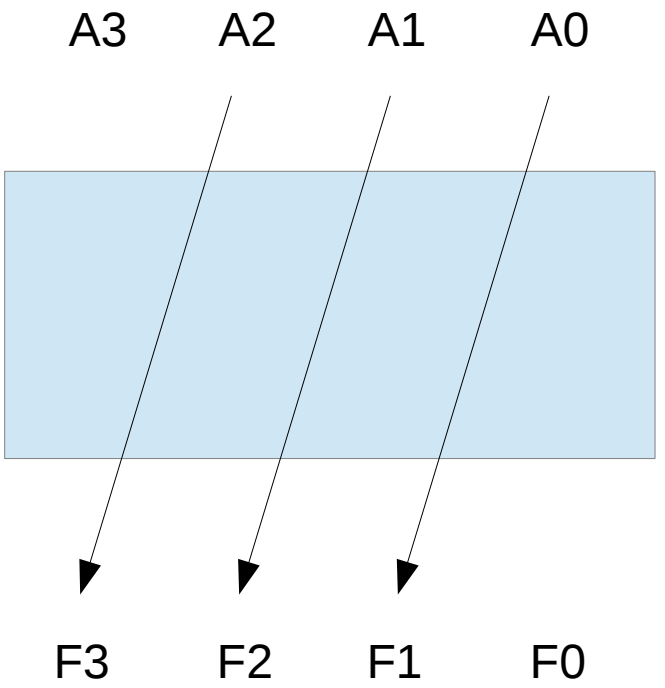






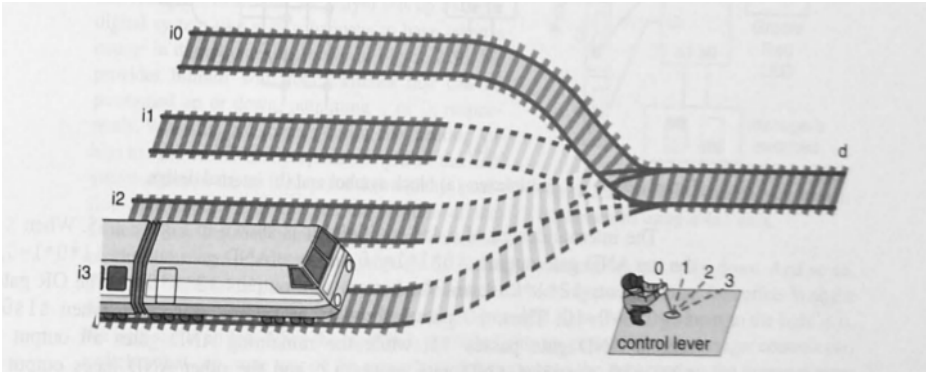
Shifter

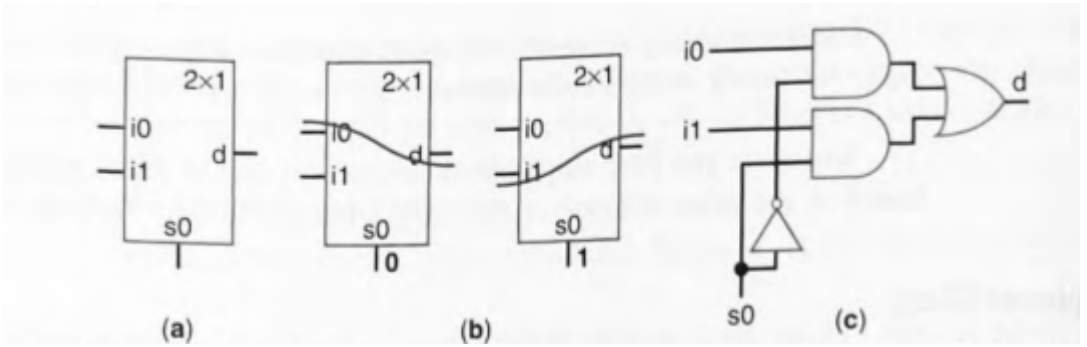
Décaler d'un rang **fixe** ne nécessite aucune porte logique !!!!

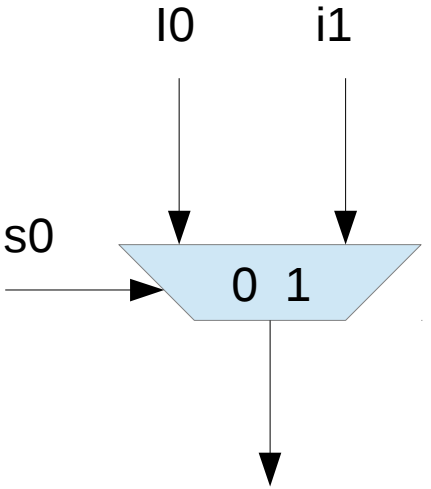


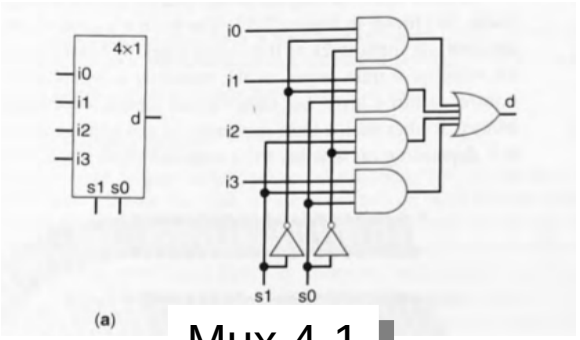
Multiplexeur

un routeur miniature





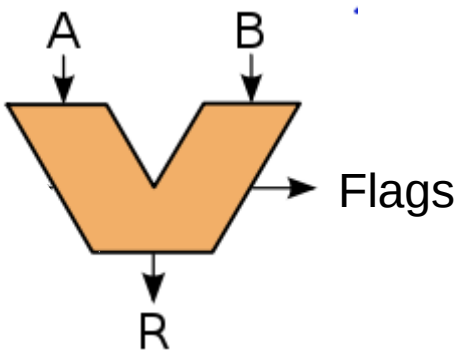




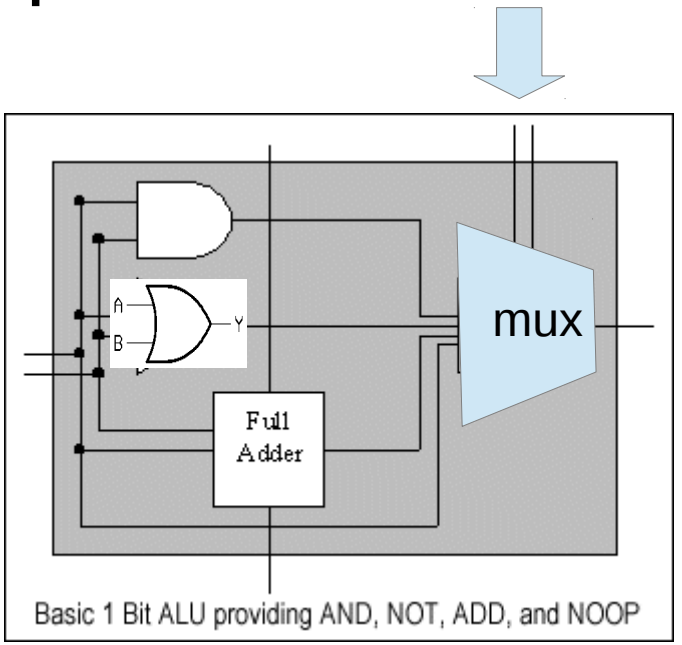
Mux 4-1

ALU

centre du processeur



- Opérateur arithmétique et logique



Basic 1 Bit ALU providing AND, NOT, ADD, and NOOP

Version simple, 1 bit

Opérations possibles :
AND,OR, ADD, bypass A

Instructions => contrôle

Données
entrées

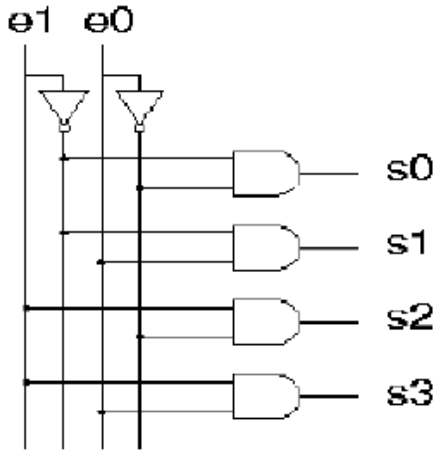
Données
sortie

Décodeur

Circuit permettant d'envoyer un signal à une sortie choisie. Il dispose de n lignes d'entrées et 2^n lignes de sortie. La table de vérité d'un décodeur "2 vers 4" (n = 2) est la suivante :

e1	e0	s0	s1	s2	s3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

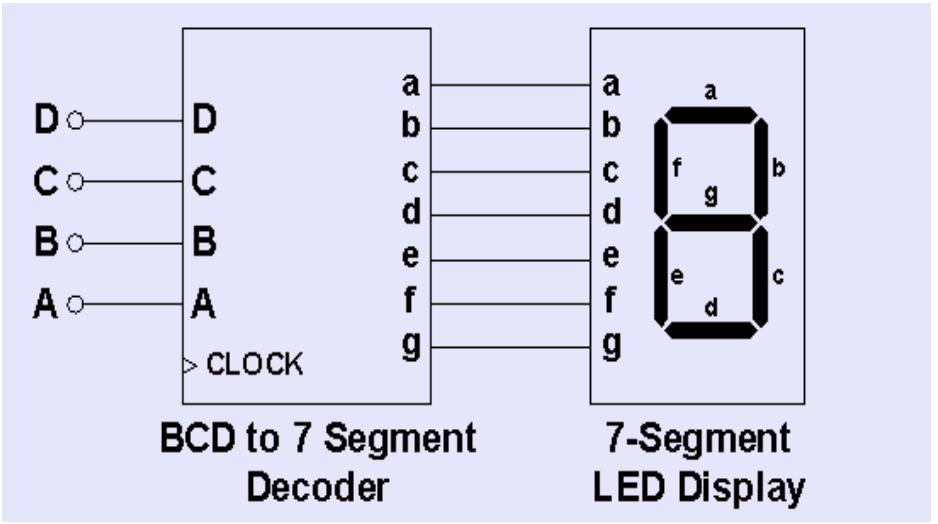
Et voici la réalisation d'un tel décodeur :



Décodeur

BCD vers 7-segments

Entrée BCD
(chiffre sur 4 bits)



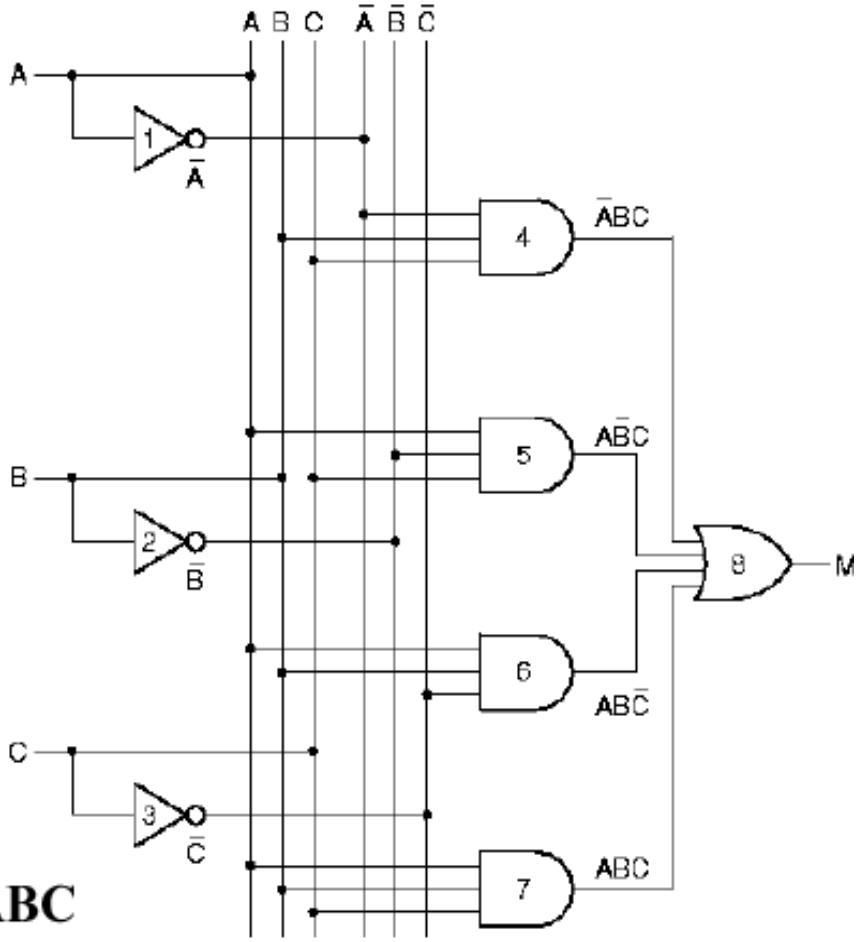
Binary Inputs				Decoder Outputs							7-Segment Display Outputs
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

7 fonctions binaires (a...g) des entrées (A,B,C,D)

Vote majoritaire

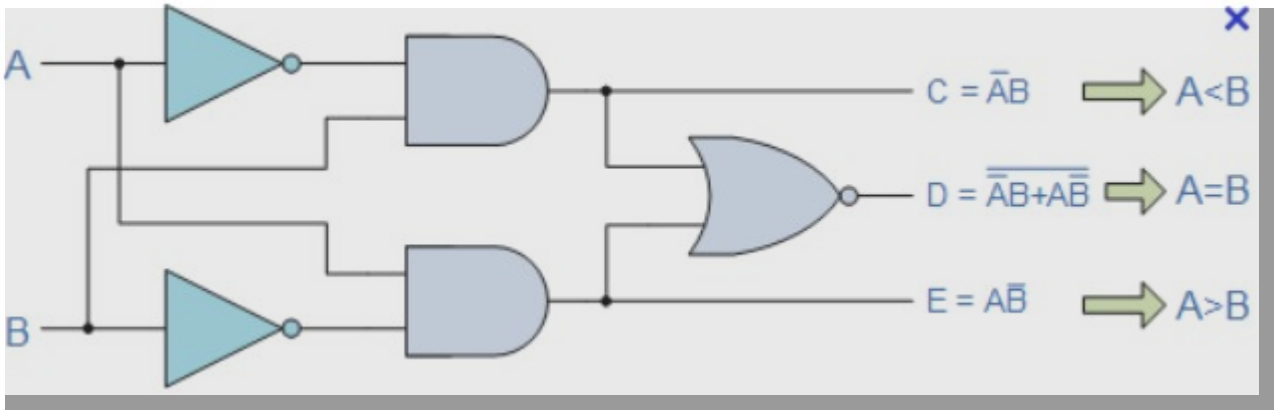
$M = f(A, B, C)$

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



$$M = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

Comparateurs



1 bit