HAUTE ECOLE DE BRUXELLES
Catégories Economique et Technique
Rue Royale 67 - 1000 Bruxelles
☎: 02/219.15.46 - 🗐: 02/219.48.47
🖃: esi@heb.be

haute école de bruxelles

Structure des ordinateurs : Résumé
1ère année

Jeyminee

2009 - 2010

Chapitre 1 : Traitement de l'information

Introduction

BIT **B**inary **D**igi**T** (unité élémentaire de l'information)

Bit : c'est plus petite unité que peut traiter le Pc (0 ou 1 ; vrai ou faux ; le courant passe ou pas)

Combien de chiffre puis-je coder avec 3bits et lesquels ?

2n = N N = le nombre de chiffre n = le nombre de bits

000 = 0
001 = 1
010 = 2
011 = 3
100 = 4 ➔ 23 = 8
101 = 5
110 = 6
111 = 7

Formule Maximum M= 2n-1

Opérations

Addition

Table de vérité	**Ex :**
0 x 0 = 0 0 x 1 = 1 1 x 1 = 0 je retiens 1	111 11 10110111 + 1011110 _____ 100010101

Multiplication

Table de vérité	**Ex :**
0 x 0 = 0 0 x 1 = 0 1 x 1 = 1	101 x 11 _____ 101 + 101x _____ 1111

Soustraction

Table de vérité	**Ex :**
0 - 0 = 0 0 - 1 = 1 1 - 1 = 1 je retire quand possible	(2) 101101 - 101011 1 _____ 000010

Représentation des nombres négatifs (bit de signe)

0111 = 7
0110 = 6
0101 = 5
0100 = 4
0011 = 3
0010 = 2
0001 = 1
0000 = 0
1000 = -0
1001 = -1
1010 = -2
1011 = -3
1100 = -4
1101 = -5
1110 = -6
1111 = -7

Le complément à 2

Le codage est identique que le binaire pour les nombres positifs mais change pour les négatifs.

Ex : $(-20)_{10}$

0 0010100 ➔ +20 avec un premier bit de signe positif
1101011 ➔ complément à 1 (on inverse)
1101100 ➔ complément à 2 (+1)
1 1101100 ➔ -20 en complément à 2 avec un premier bit de signe qui est négatif

Le 0 représente un nombre pair et le 1 un nombre impair.

EX : 10011101_2 en représentation complément à 2 vaut en fait :

10011101_2 est un nombre impair, il faut donc passer par les compléments :

(1)0011101
(1)1100010 →cpl à 1
(1)1100011 →cpl à 2
= -99_{10}

Division

La division binaire s'effectue à l'aide de soustractions et de décalages, comme la division décimale, sauf que les digits du quotient ne peuvent être que 1 ou 0. Le bit du quotient est 1 si on peut soustraire le diviseur, sinon il est 0.

Conversion en binaire

Ex :

```
45853 / 2 = 22926 reste 1
22926 / 2 = 11463 reste 0
11463 / 2 =  5731 reste 1
 5731 / 2 =  2865 reste 1
 2865 / 2 =  1432 reste 1
 1432 / 2 =   716 reste 0
  716 / 2 =   358 reste 0
  358 / 2 =   179 reste 0
  179 / 2 =    89 reste 1
   89 / 2 =    44 reste 1
   44 / 2 =    22 reste 0
   22 / 2 =    11 reste 0
   11 / 2 =     5 reste 1
    5 / 2 =     2 reste 1
    2 / 2 =     1 reste 0
    1 / 2 =     0 reste 1
```

Lire de bas en haut

IEEE 754

L'IEEE 754 est un standard pour la représentation des nombres à virgule flottante en binaire. Il est le plus employé actuellement pour le calcul des nombres à virgule flottante dans le domaine informatique, avec les CPU et les FPU. Le standard définit les formats de représentation des nombres à virgule flottante (signe, mantisse, exposant, nombres dénormalisés) et valeurs spéciales (infinis et NaN), en même temps qu'un ensemble d'opérations sur les nombres flottants. Il décrit aussi quatre modes d'arrondi et cinq exceptions (comprenant les conditions dans lesquelles une exception se produit, et ce qu'il se passe dans ce cas).

1 bit	8 bits	23 bits
Représente Le bit de signe	Représente l'exposant	Représente la mantisse

L'exposant est codé de manière biaisée sur 127 ex : 0 est codé sur 127

Nb= (signe mantisse) + mantisse . 2^{exp}

Ex : $10,4_{10}$ = ? sur 32 bits

Calcul de l'exposant

On divise 10,4 = + 5,4 . 2 = + 2,6 . 2^2 = + 1,3 . $2^{3 \rightarrow \text{l'exposant}}$
Le « + » représente le bit de signe et le 1,3 la mantisse

Signe = 0
Exposant = 3 (valeur réelle) → 3 + 127(le biais)
→ $130_{10} = 10000010_2$

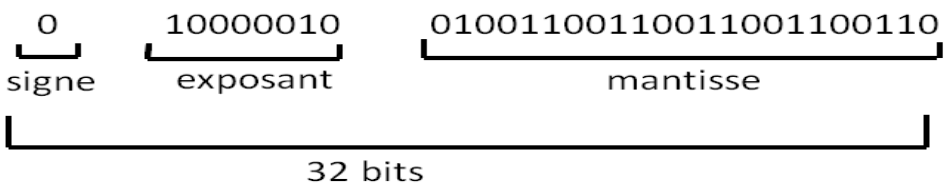
Calcul de la mantisse

On code 0,3 (car le 1 n'est pas codé)

0,	3
0,	6
1,	2
0,	4
0,	8
1,	6

$= 0,0\overline{1001}_2$

1- Mise en forme



Exemple 2 :

L'exposant

0,	1
0,	2
0,	4
0,	8
1,	6
1,	2
0,	4
0,	8
1,	6

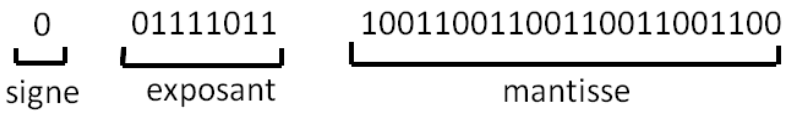
$= 0,0001\overline{1001100}_2$
mais on veut un "1" en début de mantisse donc
--> $= 1,\overline{10011001100} \cdot 2^{-4}$

exposant biaisé sur 127 donc
--> $127 - 4 = 123_{10} = 01111011_2$

La mantisse

La mantisse = 0,00011001100 mais changé (voir 1)
→1,100110011001100….

La mise en forme



Dernier exemple : $-16{,}2_{10}$ = ?

Le nombre est négatif donc le bit de signe sera « 1 ».

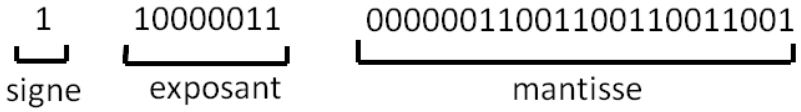
L'exposant

$-\ 16{,}2 = -8{,}1 \cdot 2 = -4{,}05 \cdot 2^2 = -2{,}025 \cdot 2^3 = -1{,}0125 \cdot 2^4$
$127 + 4 = 131_{10} \rightarrow 131_{10} = 10000011_2$

Mantisse

0,	0125 = 0,00000011001100...
0,	0250
0,	05
0,	1
0,	2
0,	4
0,	8
1,	6
1,	2
0,	4

Mise en forme

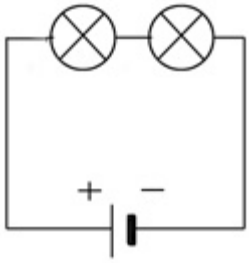


Ou alors :

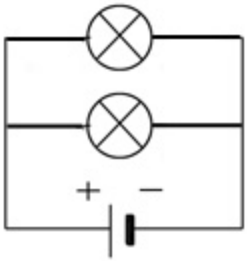
+ 16 = 10000
0,2 = 0,0011001100110011…
16,2 = 10000,001100110011…
On veut que ce soit le 1 souligné qui soit juste avant la virgule.
1,00000011001100110 0… . 2^4
Le biais est donc de $127 + 4 = 131_{10} \rightarrow 10000011_2$
Et la mise en forme est bien sûr identique.

Chapitre 2 : Portes Logiques

Circuit en série



Circuit en parallèle



Ensemble booléen

{0,1}

Variables

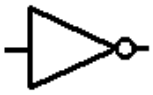
a, b, c

Opérations Logiques

- NON (négation) : a, b, c
- ET « x » « fois » : a.b
- OU « + » « plus » : a+b
- OU exclusif : a ‡(ou-exclu) b

- ! « Plus » logique différent du « plus » arithmétique.
- Les valeurs de vérité de la fonction sont définies en fonction des valeurs de vérité des variables.

Porte NON



La porte *non* a une entrée a et une sortie s, utilisée pour changer d'état.
L'état de s est opposé à celui de a.

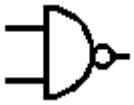
a	sortie
1	0
0	1

Porte ET (AND)

La porte *et* a deux entrées a et b et une sortie s.
Le courant ne passe que si a et b sont tous deux positifs.

a	b	sortie
0	0	0
1	0	0
0	1	0
1	1	1

Porte NON-ET (NAND ; NOT AND)



La porte *non et* est l'opposé de la porte *et*.

a	b	sortie
0	0	1
1	0	1
0	1	1
1	1	0

Porte OU (OR)

La porte *ou* a deux entrées a et b.
Le courant ne passe que si a ou b sont passants, ou a et b sont passants

a	b	sortie
0	0	0
1	0	1
0	1	1
1	1	1

Porte NON-OU (NOR ; NOT OR)

La porte *non ou* est l'opposé de la porte *ou*.

a	b	Sortie
0	0	1
1	0	0
0	1	0
1	1	0

Porte OU exclusif (XOR)

La porte *ou exclusif* a deux entrées a et b et une sortie s. Le courant ne passe que si a ou b est passant mais pas les deux.

a	b	sortie
0	0	0
1	0	1
0	1	1
1	1	0

Chapitre 3 : Lois et règles de l'Algèbre de Boole

Lois de Boole

Lois de commutativité

a + b = b + a // a . b = b . a

a	b	ab	ba	a+b	b+a
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	1	1	1

Lois d'associativité

a . (b . c) = (a . b) . c = (a . c) . b => Multiplication à 3 variables (valable pour l'addition aussi)

a	b	c	(a . b)	c		a	(b . c)	
0	0	0	0	0	= 0	0	0	= 0
0	0	1	0	1	= 0	0	0	= 0
0	1	0	0	0	= 0	0	0	= 0
0	1	1	0	1	= 0	0	1	= 0
1	0	0	0	0	= 0	1	0	= 0
1	0	1	0	1	= 0	1	0	= 0
1	1	0	0	0	= 0	1	0	= 0
1	1	1	1	1	= 1	1	1	= 1

Lois de distributivité

a (b + c) = a.b + a.c

a	b	c	a	(b + c)		a . b	a . c	
0	0	0	0	0	= 0	0	0	= 0
0	0	1	0	1	= 0	0	0	= 0
0	1	0	0	1	= 0	0	0	= 0
0	1	1	0	1	= 0	0	0	= 0
1	0	0	1	0	= 0	0	0	= 0
1	0	1	1	1	= 1	0	1	= 1
1	1	0	1	1	= 1	1	0	= 1
1	1	1	1	1	= 1	1	1	= 1

Règles de Boole

1. a + 0 = a	7. a . a = a
2. a + 1 = 1	8. a . ā = 0
3. a . 0 = 0	9. $\overline{\overline{a}} = a$
4. a . 1 = a	10. a + ab = a
5. a + a = a	11. a + āb = a + b
6. a + ā = 1	12. (a + b) (a + c) = a + bc

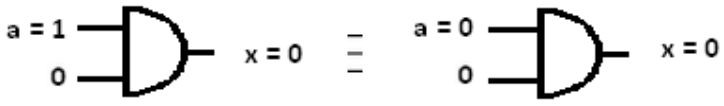
6 et 8 → dualité
7 → idempotence
2 → absorption

Démonstration

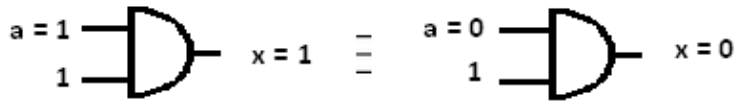
1. a + 0 = a

2. a + 1 = 1

3. a . 0 = 0



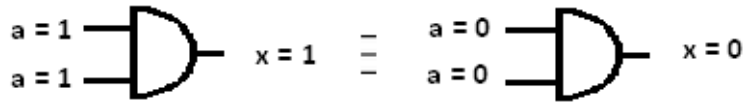
4. a . 1 = a



5. a + a = a

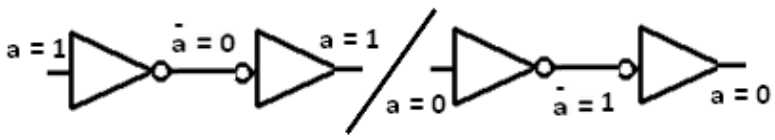
6. a + ā = 1

7. a . a = a



8. a . ā = 0

9. $\overline{\overline{a}} = a$



10. a + ab = a

= a + ab = a (1 + b) → mise en facteur

= a . 1 → règle 2 : (1 + b) = 1

= a → règle 4 : (a . 1) = a

11. a + āb = a + b

= (a + ab) + āb → règle 10 : a = a + ab

= (aa + ab) + āb → règle 7 : a = aa

= aa + ab + aā + āb →règle 8 : aā = 0

= (a + ā) (a + b) → mise en facteur

= 1 (a + b) → règle 6 : a + ā = 1

= a + b → règle 4 : a . 1 = a

12. (a + b) (a + c) = a + bc

= aa + ac + ab + bc → distributivité

= a + ac + ab + bc → règle 7 : a = aa

= a (1 + c) + ab + bc → mise en évidence de a

= a . 1 + ab + bc → règle 2 : 1 + c = 1

= a + ab + bc → règle 4 : 1 . a = a

= a (1 + b) + bc → mise en évidence de a et règle 2 : 1 + b = 1

= a . 1 + bc → règle 4 : a . 1 = a

= a +bc

Chapitre 4 : Théorème de De Morgan

Fonction **Table de vérité**

$\overline{a+b} = \overline{a}.\overline{b}$

a	b	a+b	$\overline{a+b}$	$\overline{a}$	$\overline{b}$	$\overline{a}.\overline{b}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Dans les deux cas, l'expression ne sera VRAIE que si a et b sont fausses.

Fonction **Table de vérité**

$\overline{a.b} = \overline{a} + \overline{b}$

a	b	*a.b*	$\overline{a.b}$	$\overline{a}$	$\overline{b}$	$\overline{a} + \overline{b}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

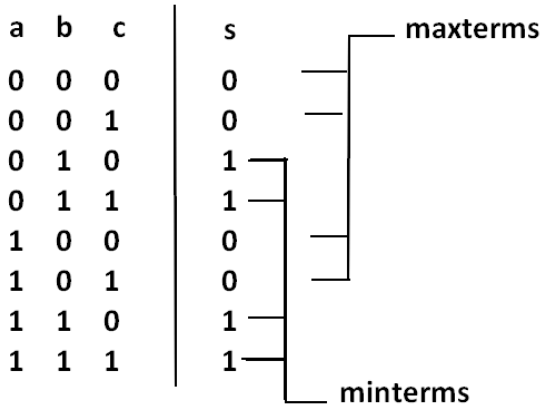
Dans les deux cas, l'expression ne sera FAUSSE que si a et b sont vraies.

Enoncé des théorèmes de De Morgan

- ➔ Le complément d'un produit de variables est égal à la somme des compléments des variables.
- ➔ Le complément d'une somme de variables est égal au produit des compléments des variables.

Chapitre 5 : Circuits logiques

Minterms : Sommes de multiplications (le plus utilisé)
Maxterms : Multiplications de sommes



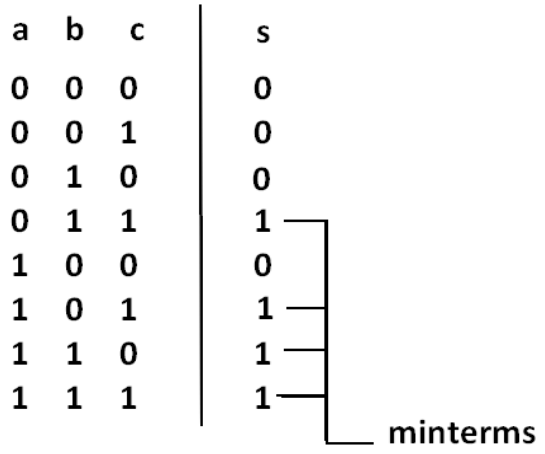
En Minterms : On complémente les « 0 »

$S1 = (\bar{a} \cdot b \cdot \bar{c}) + (\bar{a} \cdot b \cdot c) + (a \cdot b \cdot \bar{c}) + (a \cdot b \cdot c)$

En Maxterms : On complémente les « 1 »

$S2 = (\bar{a} + \bar{b} + \bar{c}) \cdot (a + b + c) \cdot (\bar{a} + b + c) \cdot (\bar{a} + b + \bar{c})$

S1 et S2 sont des expressions algébriques et elles sont égales.
On peut représenter ces expressions sous forme de tables de Karnaugh.



C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

En Minterms : on regroupe les « 1 » mais on ne peut les regrouper par puissances de 2 (2, 4, 8,…).
Ici on fait donc 3 regroupements ; pour écrire ces groupements sous forme algébrique, on regarde les termes qui ne changent pas d'état et on complémente les « 0 ».

$S = (a \cdot b) + (b \cdot c) + (a \cdot c)$

Tables de Karnaugh

Elles permettent de simplifier des fonctions logiques.

Avec 2 variables :

A \ B	0	1
0	00	01
1	10	11

Avec 3 variables :

A \ BC	00	01	11	10
0	000	001	011	010
1	100	101	111	110

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

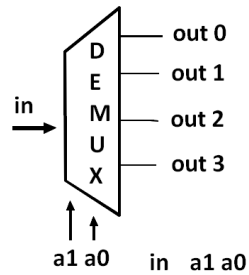
Avec 4 variables :

AB \ CD	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

AB \ CD	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	1	1
10	0	0	1	1

Circuits combinatoires standard

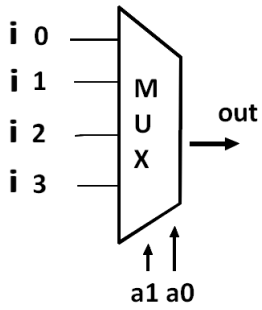
Démutiplexeurs



in	a1 a0	out0	out1	out2	out3
C	(0 0)	C	0	0	0
C	(0 1)	0	C	0	0
C	(1 0)	0	0	C	0
C	(1 1)	0	0	0	C

a1 et a0 est le codage qui permet de savoir sur quel fil va sortir l'entrée « in ».

Multiplexeur

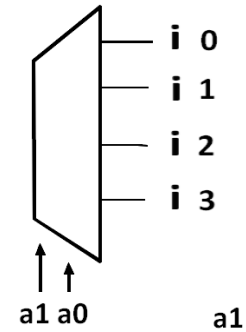


C = 0 ou 1

i 0	i 1	i 2	i 3	a1 a0	ou
C	0	0	0	(0 0)	C (i
0	C	0	0	(0 1)	C (i
0	0	C	0	(1 0)	C (i
0	0	0	C	(1 1)	C (i

a1 et a0 codent (en binaire) le numéro du fil qu'on regarde pour le mettre en sortie (out).

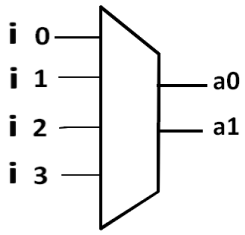
Décodeur (DEMUX sans entrée de sortie)



a1	a0	i 0	i 1	i 2	i
0	0	i	0	0	0
0	1	0	i	0	0
1	0	0	0	i	0
1	1	0	0	0	i

Envoie un « 1 » au port donné par les 2 entrées d'adresse

Codeur



Le codeur fournit en sortie l'adresse du fil d'entrée à 1 mais les autres entrées doivent être à 0.

i0	i1	i2	i3	a1	a0
i	0	0	0	0	0
0	i	0	0	0	1
0	0	i	0	1	0
0	0	0	i	1	1

a 1

i2 i3\ i0 i1	0 0	0 1	1 1	1 0
0 0	0	X	O	X
0 1	X	1	X	X
1 1	X	X	X	X
1 0	X	1	X	X

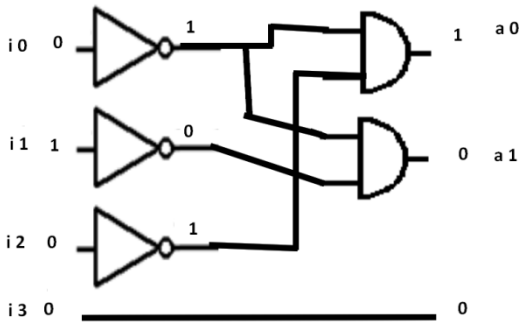
Ici, on peut changer les « x » pour la valeur qu'on souhaite : on change ceux surlignés en 1 le reste en « 0 ».

S 1 = ī0 . ī1

a 0

i2 i3\ i0 i1	0 0	0 1	1 1	1 0
0 0	0	X	1	X
0 1	X	1	X	X
1 1	X	X	X	X
1 0	X	0	X	X

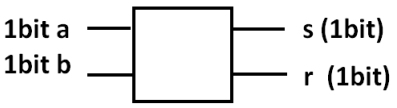
S 2 = ī0 . ī2



Additionneurs

Le demi-additionneur

On additionne deux entrées qui tiennent sur 1bit et ont sort l somme algebrique sur 1bit.



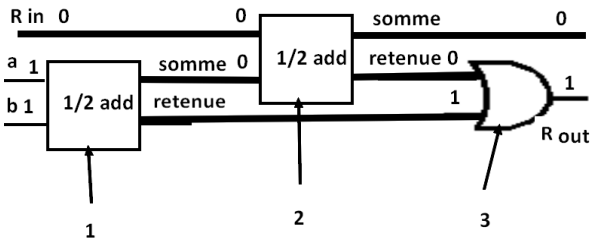
somme = ou exclusif
retenue = ET

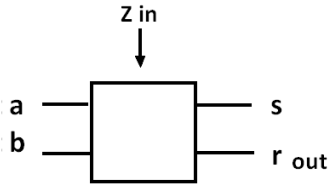
a b	somme	retenue
0+0	0	0
0+1	1	0
1+0	1	0
1+1	0	1

1+1=2 mais réponse sur 1bit donc on peut pas le coder: S = 0 mais la retenue = 1

L'additionneur 1bit

Idem que le semi-add. : mais on a une retenue en entrée.



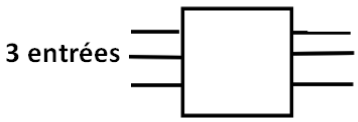


a	b	R in	s	R out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3 Entrées :

1→ d'abord on additionne les entrées a et b 2sorties
2→ ensuite on additionne ce résultat à l'éventuel R in
3→ puis on compare les retenues

Incrémenteur Octal



a2	a1	a0	S2	S1	S0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Tables :

S2

a0\ a2 a1	00	01	11	10
0	0	0	1	1
1	0	1	0	1

$S2 = (a2 . \bar{a}0) + (a2 . \bar{a}1) + (\bar{a}2 . a1 . a0)$

S1

a2 a1 \ a0	0	1
00	0	1
01	1	0
11	1	0
10	0	1

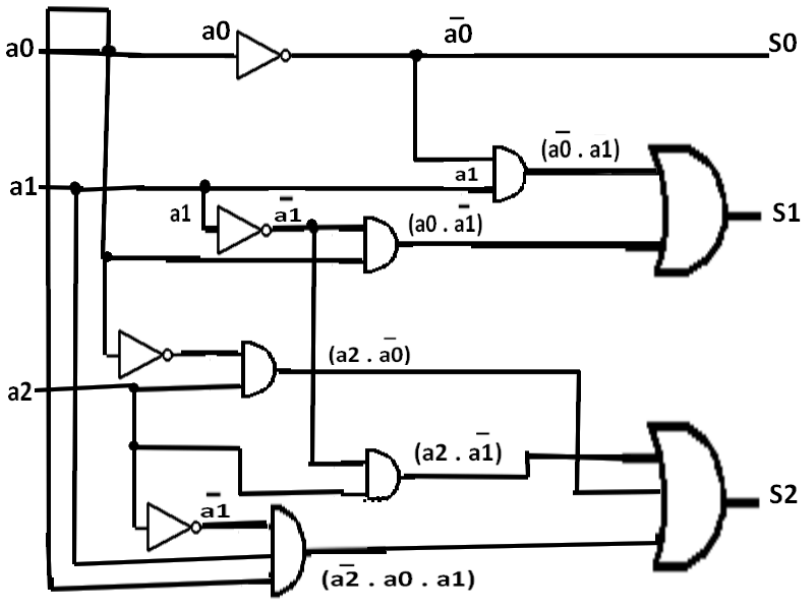
$S1 = (\bar{a}0 . a1) + (a0 . \bar{a}1)$

S0

a0\ a2 a1	00	01	11	10
0	1	1	1	1
1	0	0	0	0

$S0 = \bar{a}0$

Logigramme :



Circuits logique séquentiels

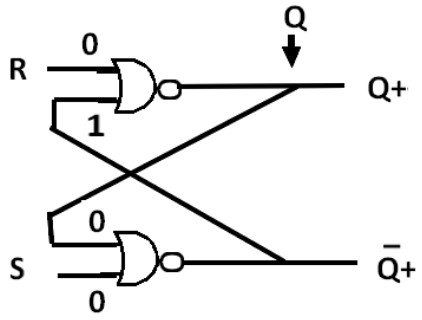
Une bascule ou un verrou est un circuit logique doté d'une ou deux sorties et d'une ou plusieurs entrées. La sortie peut être au niveau logique 0 ou 1. Les changements d'état de la sortie sont déterminés par les signaux appliqués aux entrées et le type d'opérateur.

Ce qui différencie les bascules des circuits logiques combinatoires (portes ET, OU, OU Exclusif, etc.), c'est que la sortie maintient son état même après disparition du signal de commande. Comme l'état précédent et la mémorisation interviennent, on parle de logique séquentielle.

La bascule est l'élément de base de la logique séquentielle. En effet, en assemblant des bascules, on peut réaliser des compteurs, des registres, des registres à décalage, des mémoires.

Certaines bascules, appelées à fonctionner dans des systèmes synchrones, possèdent une entrée d'horloge de synchronisation. Il existe donc des bascules asynchrones et des bascules synchrones.

Bascule RS



Q = état initial du système
Q+ = état du système à l'équilibre suivant

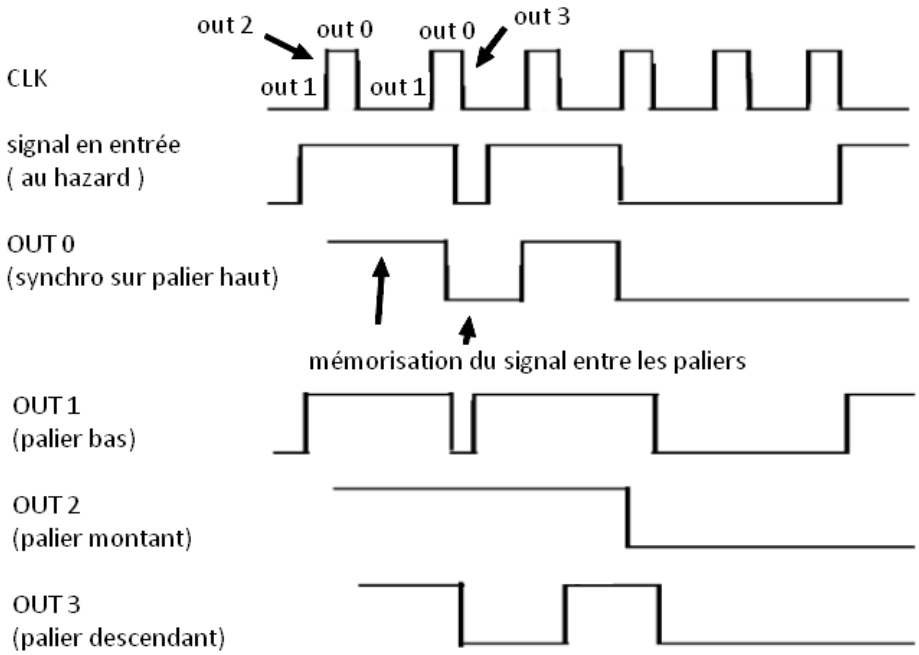
a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

R	S	Q	Q+	$\bar{Q}$+	
0	0	0	0	1	a
0	0	1	1	0	
0	1	0	1	0	b
0	1	1	1	0	
1	0	0	0	1	c
1	0	1	0	1	
1	1	0	X	X	d
1	1	1	X	X	

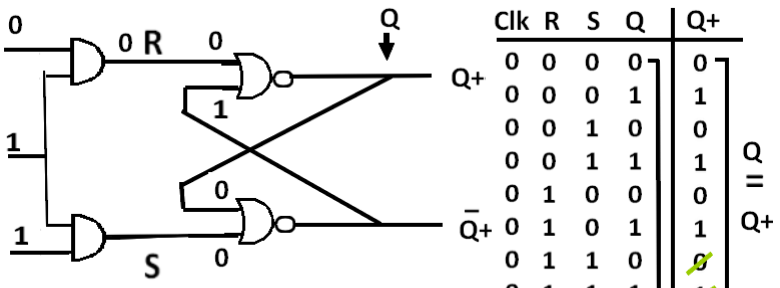
a → Q = Q+ : Il y a mémorisation du système si on ne touche plus aux entrées.
b → SET : La sortie passe à « 1 » d'office quel qu'était l'ancienne valeur du système (Q).
c → RESET : La sortie passe à « 0 » quelque soit la valeur de Q.
d → Interdit : Quand R et S sont à 1 ensemble, c'est le bordel, ça oscille.

Les chronographes

On a une clock qui indique quand on « prend en compte » le signal entrant.



Bistable R-S (= bascule R-S avec Clk)



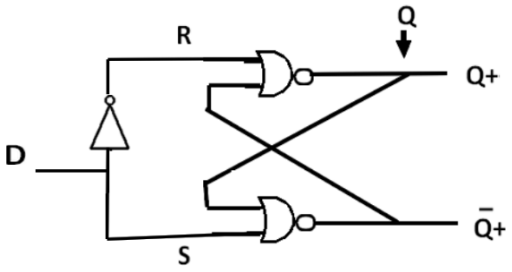
Clk	R	S	Q	Q+	
0	0	0	0	0	Q = Q+
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	1	1	
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	BASCULE R-S
1	0	0	1	1	mémorisation
1	0	1	0	1	SET
1	0	1	1	1	
1	1	0	0	0	RESET
1	1	0	1	0	
1	1	1	0	X	
1	1	1	1	X	

On prend la syncronisation de cette bistable sur niveau 1 (c-à-d sur palier bas) de la clock.

Quand la clock sera à 0, on a 0 après les portes ET et donc l'état de Q+ est conservé (comme le point « a » de la bascule R-S.

Bascule D asynchrone (sans clock)

Cette bascule ne mémorise RIEN, R est toujours égal à S. Cette bascule est un RETARDATEUR : Il faut l'équilibre s'installe et que les infos passent les portes logiques. Le paramètre D est toujours renvoyé mais avec un délai.



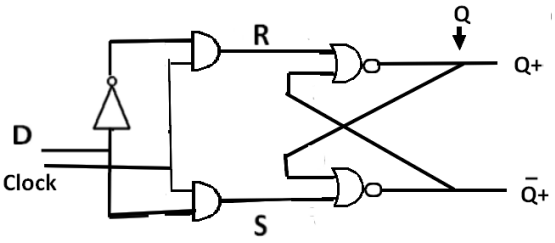
D = paramètre
Q = état initial
Q+ = sortie

D = Q+

D	Q	Q+
0	0	1
0	1	0
1	0	0
1	1	0

Bascule D asynchrone (avec clock)

Syncro sur niveau 1 d'horloge (palier bas). Quand la clock est à 0, on retrouve la mémorisation de l'ensemble du bistable.



C	D	Q	Q+	
0	0	Q	Q	mémorisation de Q
0	1	Q	Q	
1	0	Q	0	bascule D asynchrone avec délai
1	1	Q	1	

Q est soit à 1
soit à 0

Bistable D – Flip/Flop

Syncro sur front d'horloge montant. Ce circuit offre une mémorisation sur une période d'horloge.

C	D	Q+	
0	D	Q	
front montant	D	D	(car front montant)
1	D	Q	
	D	Q	mémorisation