

Fonctionnement des ordinateurs

chapitre IV : fondements (1/3)

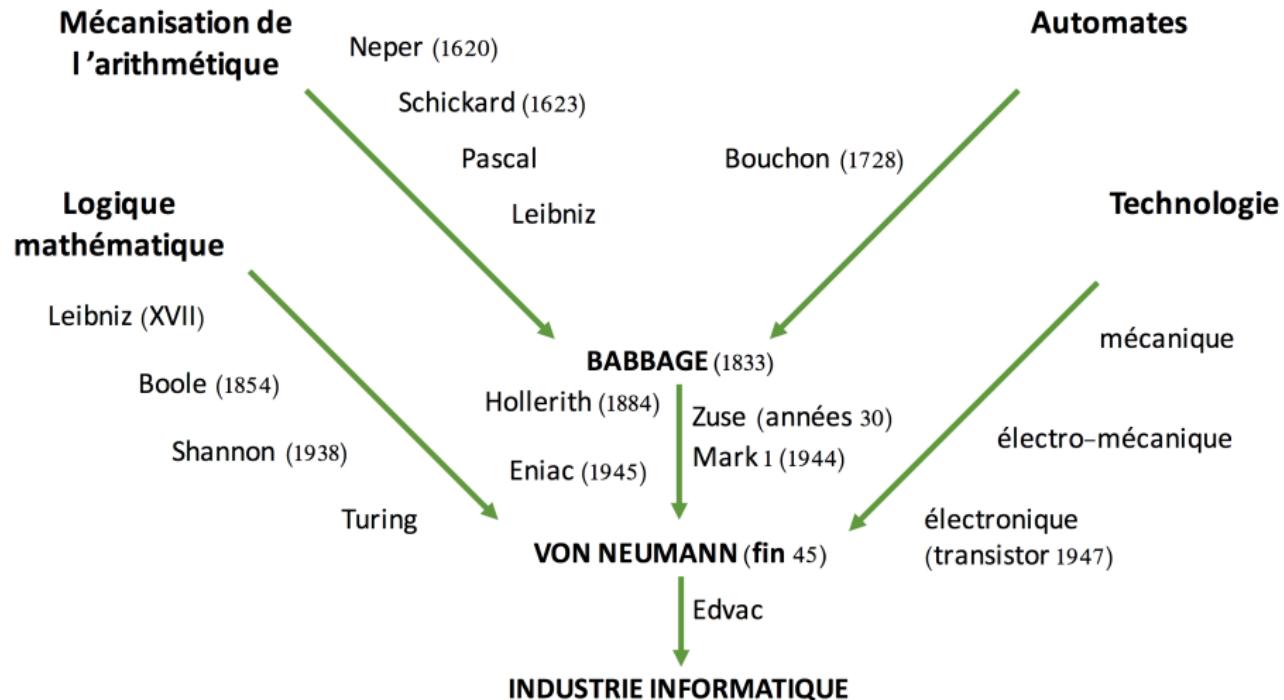
Prof. Xavier Gandibleux

Université de Nantes
Département Informatique – UFR Sciences et Techniques

Année académique 2019-2020

Fondements

Des fondements et des précurseurs



Fondements

Des fondements et des précurseurs

Influence des travaux sur la logique mathématique de

- ▶ Gottfried Wilhelm **Leibniz** (DE, 1646-1716)

En 1666, son ouvrage *De arte combinatoria* décrit le **Calculus Ratiocinator** (concept théorique qui peut être vu comme une méthode, un algorithme, ou une machine, qui permettrait de démêler le vrai du faux dans toute discussion dont les termes seraient exprimés dans la langue philosophique universelle).

- ▶ Georges **Boole** (UK, 1815-1864)

De 1844 à 1854, il crée une algèbre binaire, dite booléenne, n'acceptant que deux valeurs numériques, 0 et 1, appelée **l'algèbre de Boole** (partie des mathématiques qui s'intéresse à une approche algébrique de la logique, vue en termes de variables, d'opérateurs et de fonctions sur les variables logiques.)

NB : Fonctions logiques, table de Karnaugh, algèbre de Boole, circuits logiques sont des notions considérées connues (revoir “Base de logique numérique (X11P020)” si nécessaire).

Des volts à la logique

premier fondamental : le bit
représentation mécanique d'un bit
représentation électrique d'un bit
second fondamental : le transistor

des volts aux booléens avec un transistor bipolaire
des volts aux booléens avec un transistor MOSFET
le transistor pour le fonctionnement de l'ordinateur

Premier fondamental :

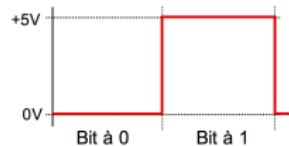
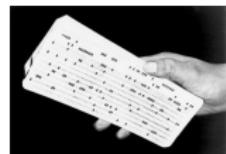
$$\{0, 1\}$$

comme élément atomique de l'ordinateur

Fondements

De la physique à l'ordinateur

- ▶ Bit signifie **binary digit**
 - ▶ vrai ou faux en logique
 - ▶ 0 ou 1 en numérotation binaire
 - ▶ la plus petite unité d'information manipulable par un ordinateur
- ▶ Représentation physique d'un bit :
 - ▶ par des **trous dans une surface** qui peut être réalisé ou pas (exemple : cartes perforées)
 - ▶ par une **tension électrique** qui peut être nulle ou pas (exemple : signal électrique)

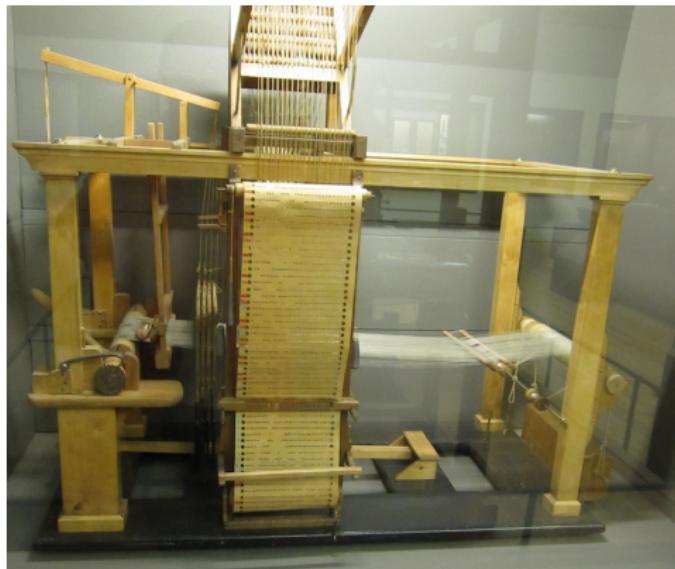


Fondements

De la physique à l'ordinateur : représentation mécanique d'un bit

Fondements

De la physique à l'ordinateur : représentation mécanique d'un bit



1728 : Basile Bouchon (FR)
utilise un ruban perforé pour
programmer un métier à tisser : la
première machine capable d'exécuter
un programme externe.

Fondements

De la physique à l'ordinateur : représentation mécanique d'un bit

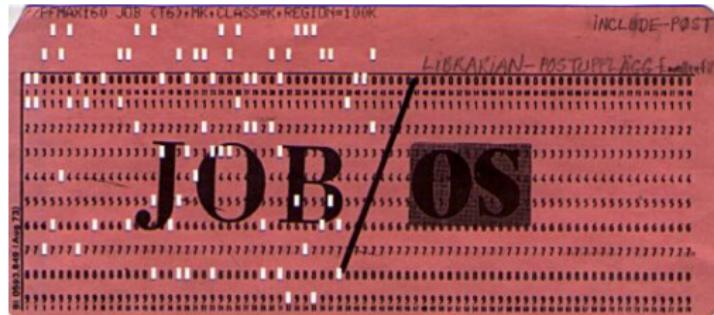


1834 : Charles Babbage (UK)

utilise les cartes pour donner des instructions et des données à sa machine analytique, l'ancêtre des ordinateurs.

Fondements

De la physique à l'ordinateur : représentation mécanique d'un bit



progressivement disparue à partir de 1970,
remplacé par la représentation magnétique

Herman Hollerith (USA)

1884 : il dépose un brevet pour une machine à cartes perforées.

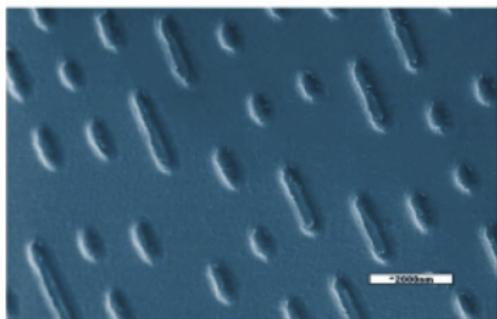
1890 : son invention est utilisée pour le recensement aux États-Unis.

1911 : il fonde la Computing-Tabulating-Recording Corporation, rebaptisée International Business Machines, ou IBM, en 1924

Vidéo de l'INA présentant les cartes perforées en 1960 (<http://www.ina.fr/video/CAF97059686>)

Fondements

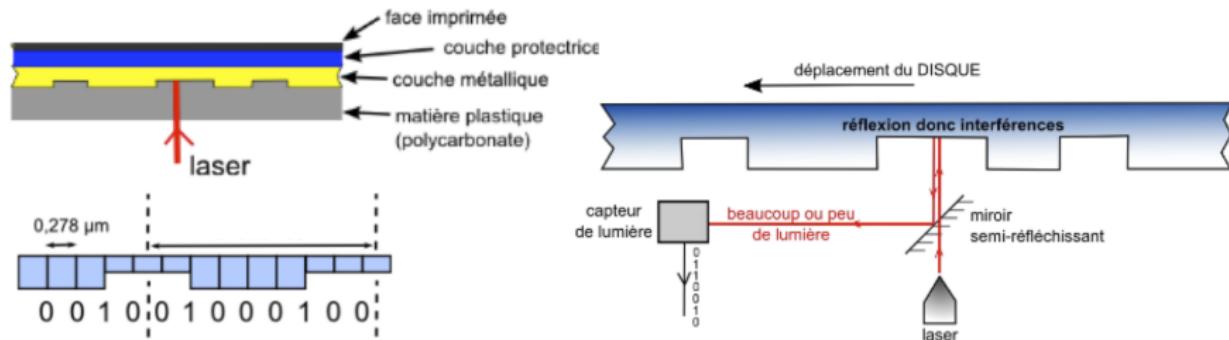
De la physique à l'ordinateur : représentation mécanique d'un bit



en stockage et lecture optique (CD, DVD, BluRay)

Fondements

De la physique à l'ordinateur : représentation mécanique d'un bit



Codage de l'information sur un front (transition)

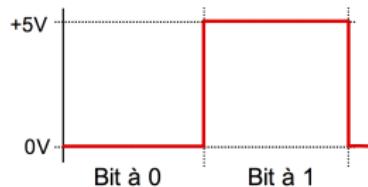
Fondements

De la physique à l'ordinateur : représentation électrique d'un bit

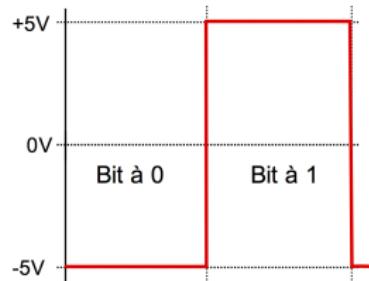
Fondements

De la physique à l'ordinateur : représentation électrique d'un bit

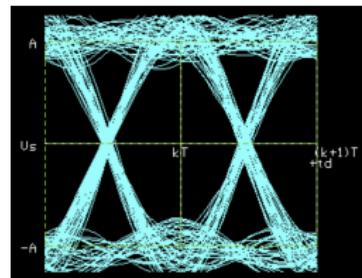
Codage 1 :



Codage 2 :



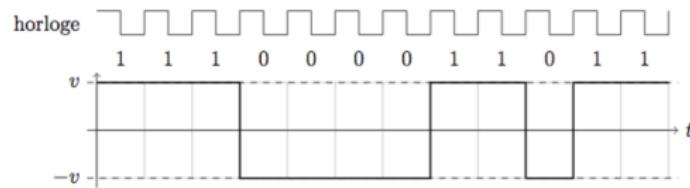
Observation à l'oscilloscope :



Codage NRZ (No Return to Zero)

Diagramme de l'œil

Hypothèse :

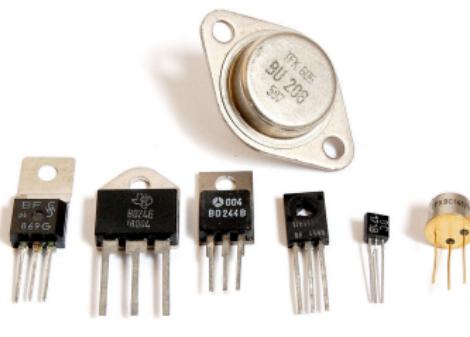


Second fondamental :
le transistor
pour passer de l'électronique à la logique

Fondements

De l'électronique à la logique

- ▶ un **transistor** est un composant électronique semi-conducteur



Différents boîtiers de transistors



Tubes à vide (ancêtre du transistor)

- ▶ inventé le 23 décembre 1947 aux USA par J. Bardeen, W. Shockley et W. Brattain (prix Nobel de physique en 1956)

Fondements

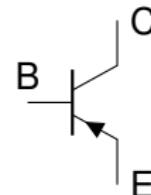
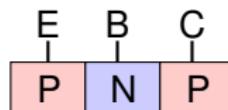
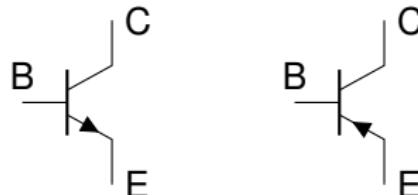
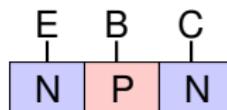
De l'électronique à la logique : transistor

- ▶ matériau :
 - ▶ germanium (à l'origine)
 - ▶ silicium (aujourd'hui)
- ▶ technologie :
 - ▶ TTL (Transistor-Transistor Logic), \geq années 60
 - ▶ MOS (Metal Oxide Semi-conductor), \geq années 80
- ▶ famille :
 - ▶ transistors bipolaires (Bipolar Junction Transistor, BJT)
 - ▶ transistors à effet de champ (Field Effect Transistor, FET)
- ▶ fonctions essentielles :
 - ▶ amplification
 - ▶ commutation

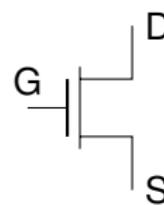
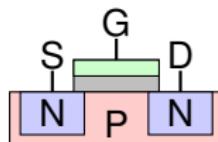
Fondements

De l'électronique à la logique : transistor

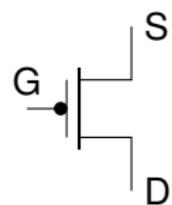
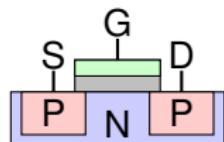
- ▶ transistors bipolaires



- ▶ transistors MOSFET



MOSFET canal N
ou NFET



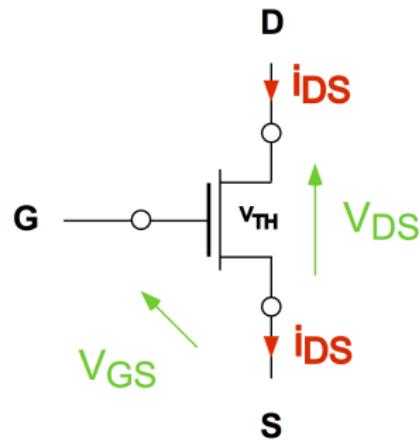
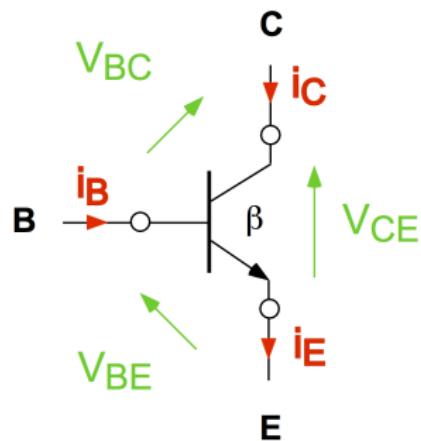
MOSFET canal P
ou PFET

(approfondir : <http://www.electronics-tutorials.ws/category/transistor>)

Fondements

De l'électronique à la logique

Grandeurs caractéristiques :

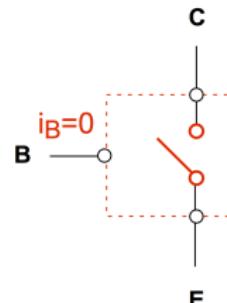


Fondements

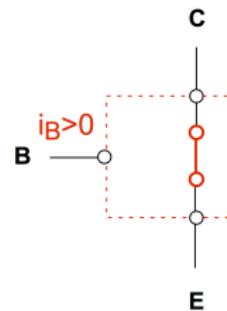
De l'électronique à la logique

Principe de fonctionnement simplifié du transistor(bipolaire NPN) en commutation :

- ▶ Propriété du transistor **bloqué** (état ouvert) :



- ▶ Propriété du transistor **saturé** (état fermé) :

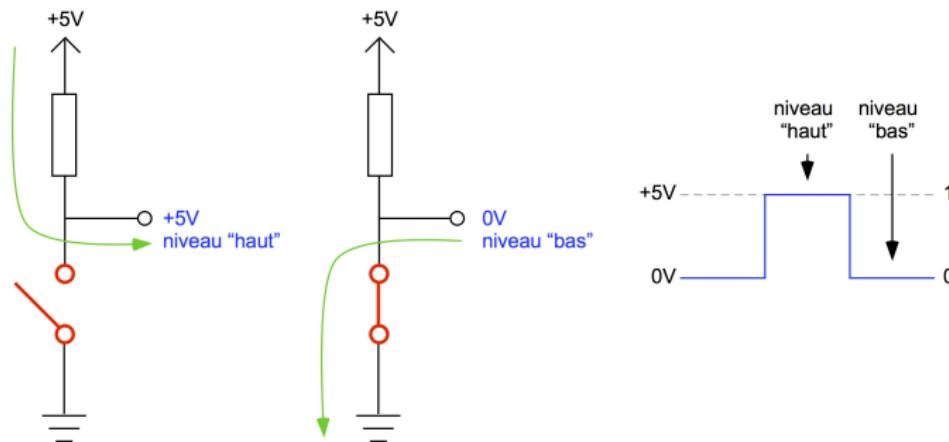


Fondements

De l'électronique à la logique

- ▶ fonction interrupteur

Illustration pour un transistor bipolaire :

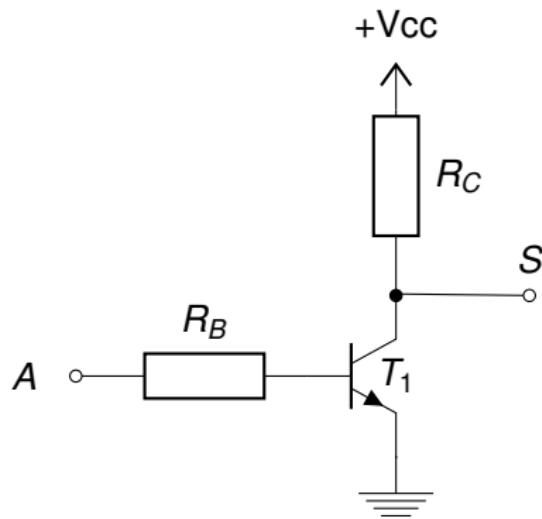


- ▶ réaliser des **fonctions logiques** (NON, ET, OU, etc.)

Des volts aux booléens avec un transistor bipolaire

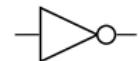
Fondements

Du principe d'un transistor bipolaire aux fonctions logiques



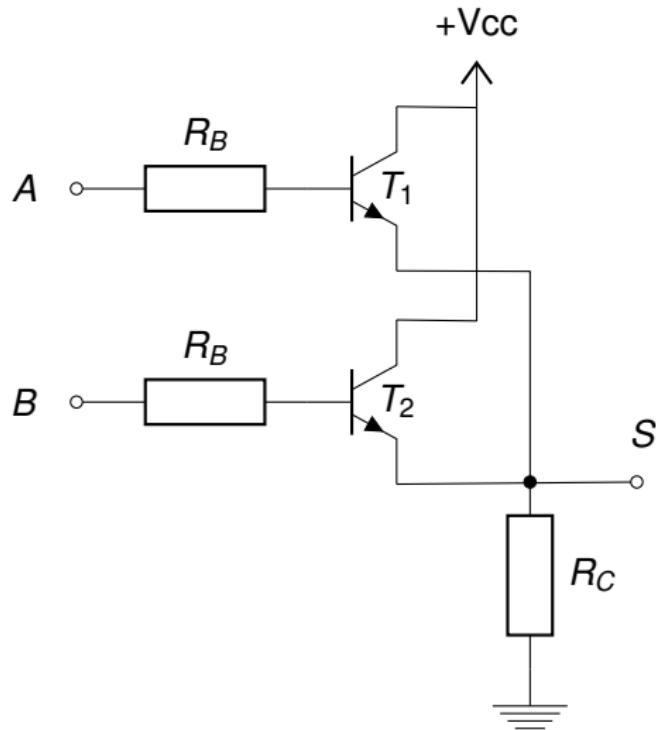
Fonction NOT
($S = \bar{A}$)

A	S
0	1
1	0



Fondements

Du principe d'un transistor bipolaire aux fonctions logiques



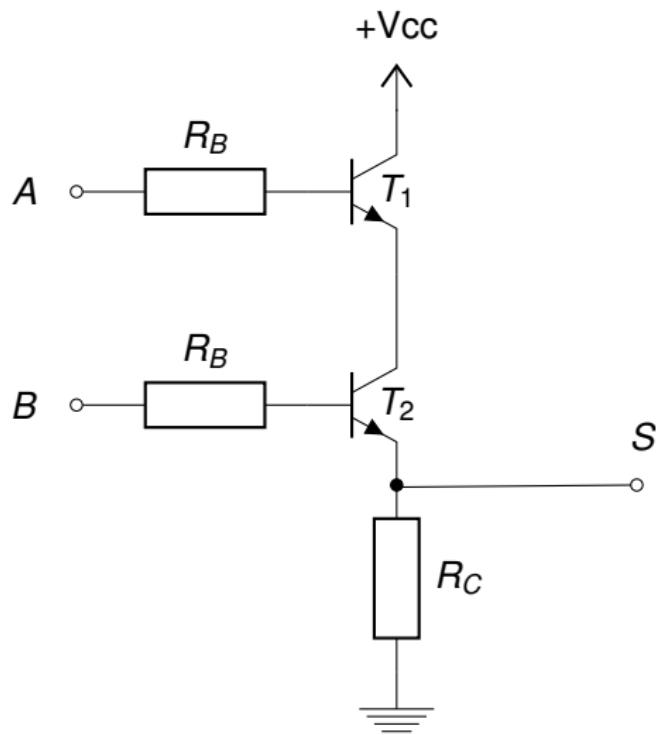
Fonction OR
($S = A + B$)

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



Fondements

Du principe d'un transistor bipolaire aux fonctions logiques



Fonction AND
($S = A \cdot B$)

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



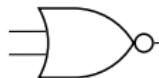
Fondements

Du principe d'un transistor bipolaire aux fonctions logiques

Rappel : les autres fonctions logiques

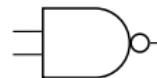
Fonction NOR
 $(S = \overline{A + B})$

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



Fonction NAND
 $(S = \overline{A \cdot B})$

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



Fonction XOR
 $(S = A \oplus B)$

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



Fonction XNOR
 $(S = \overline{A \oplus B})$

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

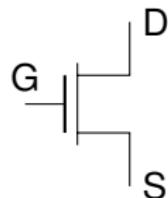


Fondements

Des volts aux booléens avec un transistor MOSFET

Fondements

De l'électronique à la logique : 2 régimes de fct du NFET



Si $V_{GS} \leq V_{TH}$ alors

NFET se comporte comme un interrupteur ouvert
(régime de coupure) et $I_{DS} = 0$.

Si $V_{GS} > V_{TH}$ et $V_{DS} > (V_{GS} - V_{TH})$ alors

NFET se comporte comme un interrupteur fermé
(régime de saturation) et $I_{DS} > 0$.

NB : Le PFET se comporte exactement de manière opposée au NFET.

Supposons :

$$V_D = 5V \quad (V_{dd})$$

$$V_{TH} = 1V$$

$$V_S = 0V \quad (V_{ss})$$

Si $V_{in} = 0V$ (entrée basse) alors pas de connexion

entre V_{out} et V_{ss} , et $V_{out} = V_{dd}$ (sortie haute)

Remarque : $I_{DS} = 0 \rightarrow$ pas de consommation d'énergie

Si $V_{in} = 5V$ (entrée haute) alors relie V_{out} à V_{ss} avec
une faible résistance, et $V_{out} = V_{ss}$ (sortie basse)

Remarque : $I_{DS} > 0 \rightarrow$ consommation d'énergie!

Fondements

De l'électronique à la logique

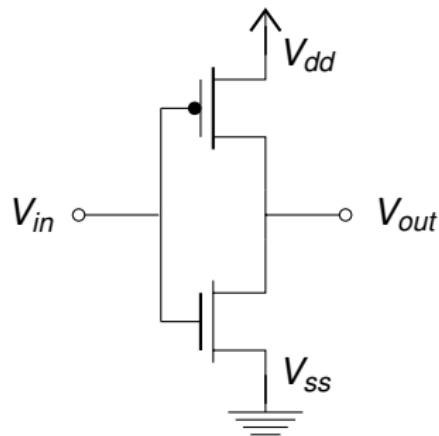
- ▶ Observation sur circuit précédent :
 - ▶ Un NFET est un inverseur logique élémentaire
 - ▶ $V_{in} = 0$, aucun courant ne passe.
 - ▶ $V_{in} = 1$, le NFET s'active et « tire » la sortie vers l'état bas
→ mode « pull-down ».
 - ▶ Symétrie immédiate avec un PFET :
 - ▶ $V_{in} = 0$, le PFET qui s'active « tire » la sortie vers l'état haut
→ mode « pull-up ».
- ▶ Architecture CMOS (Complementary MOS)
 - ▶ pourquoi : éviter de laisser passer un courant
⇒ pas de dissipation d'énergie, pas d'échauffement
 - ▶ but recherché : ne jamais relier alimentation et masse
⇒ pas de circulation de courant
 - ▶ moyen : utiliser simultanément des NFET et PFET de manière complémentaire

De l'électronique à la logique

- ▶ conséquence :
 - ▶ fabriquer un inverseur CMOS avec deux circuits, un pull-up et un pull-down.
- ▶ fonctionnement :
 - ▶ $V_{in} = 0$, le PFET conduit, le NFET bloque
→ sortie directement reliée à l'alimentation \Rightarrow l'état haut.
 - ▶ $V_{in} = 1$, le NFET conduit, le PFET bloque
→ sortie directement reliée à la masse \Rightarrow l'état bas.
- ▶ bilan :
 - ▶ PROS: aucun courant consommé
 - ▶ PROS : aucune dissipation d'énergie
 - ▶ CONS : deux fois plus de transistors utilisés

Fondements

De l'électronique à la logique



Porte NOT

Porte NOT sur logisim

Le transistor pour le fonctionnement de l'ordinateur

Fondements

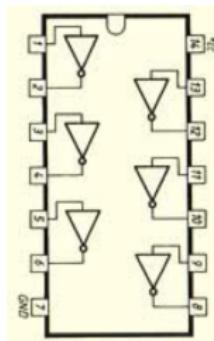
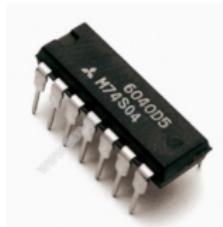
Usage du transistor

► portes logiques

Exemple : le circuit intégré 7404

7404、74H04、74L04、74LS04、74S04、74HC04、

74C04、74F04、74ALS04 六反相器



$$Y = \overline{A}.$$

1A	1	14	VCC
1Y	2	13	6A
2A	3	12	6Y
2Y	4	11	5A
3A	5	10	5Y
3Y	6	9	4A
GND	7	8	4Y



- Comporte 6 portes NON
- Broche 14 : l'alimentation (Vcc)
- Broche 7 : la terre (GND)

Fondements

Usage du transistor

▶ processeurs

intel 4004



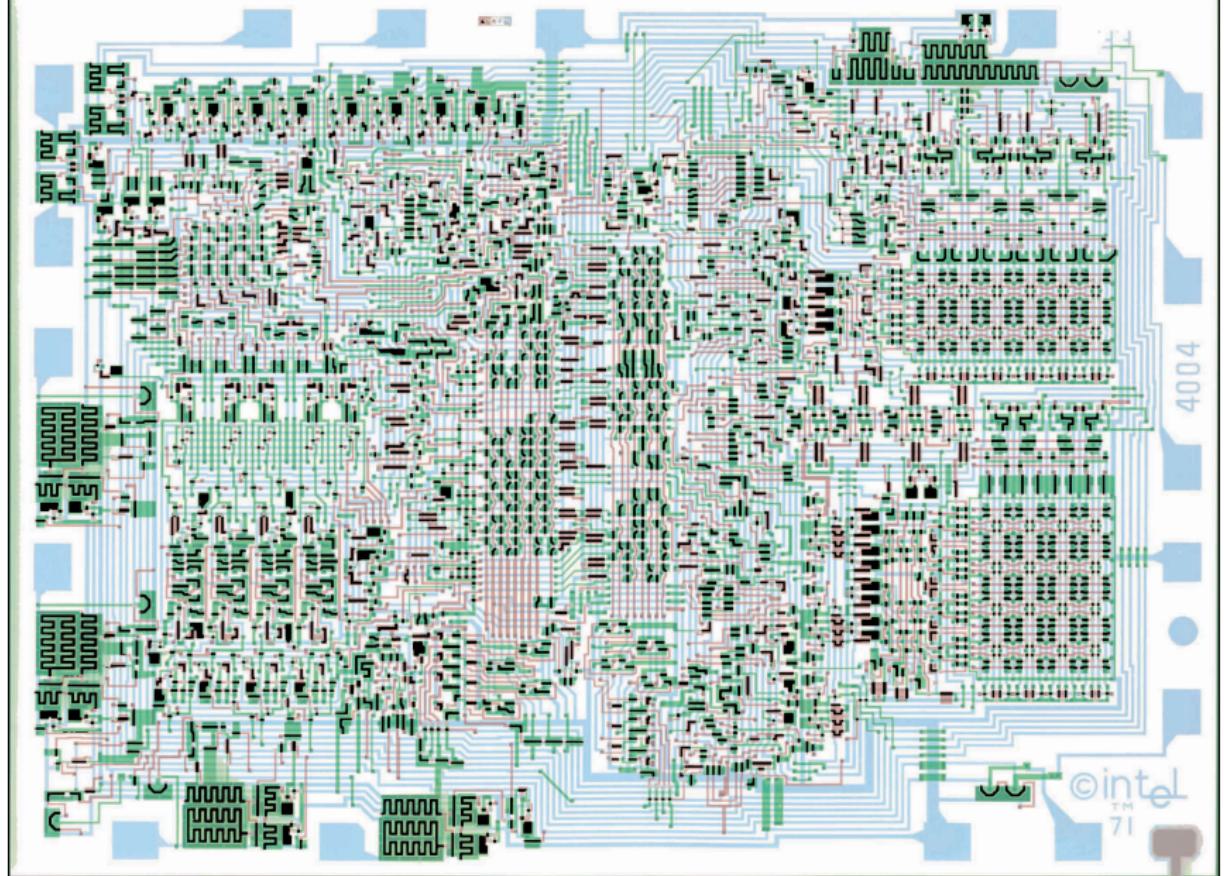
motorola 68000



intel CORE i7

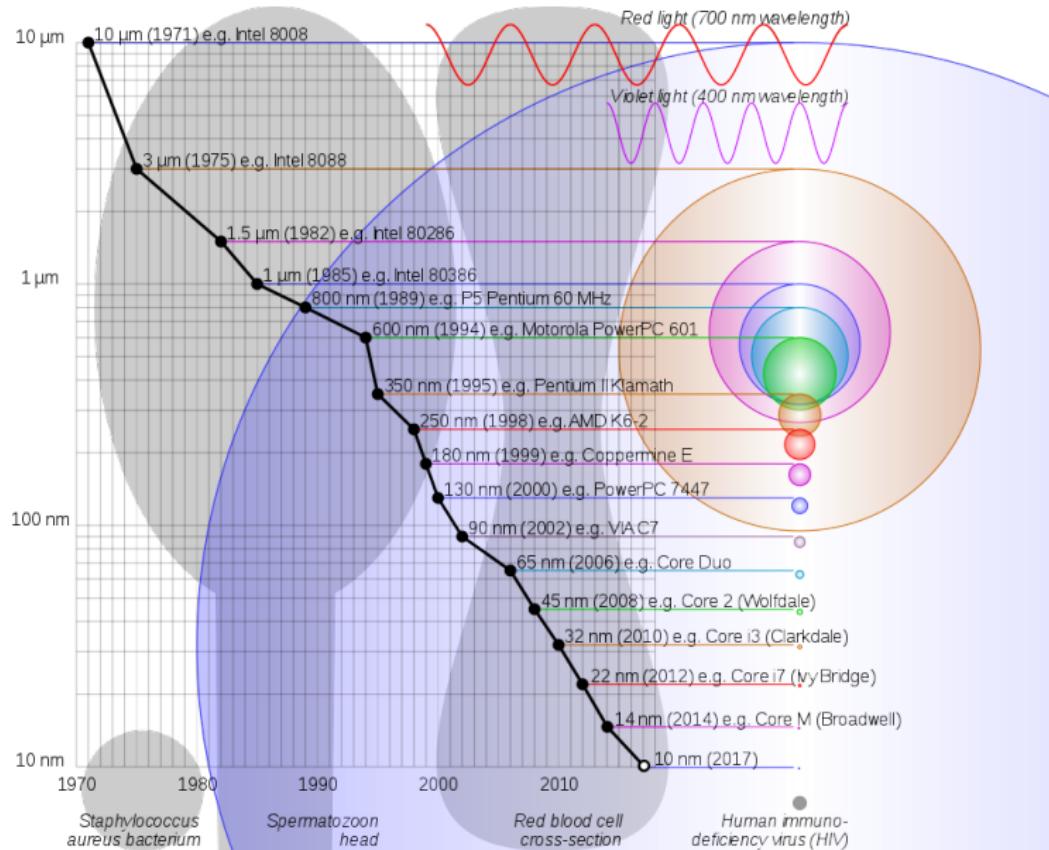


année	modèle	#transistors	fréquence
1971	Intel 4004	2250	108 KHz
1976	Zilog Z80	29000	2.5 MHz
1982	Motorola 68000	68000	16 MHz
1985	Intel 80386	275 000	33 MHz
1996	Intel pentium III	9 500 000	500 MHz
2006	Intel Core Duo T2700	152 millions sur 90mm ²	2.33 GHz
2014	Intel Core i7 Haswell	2,6 milliards	1,60 GHz à 4,0 GHz



intel 4004 (<http://www.4004.com>)

Progrès de la miniaturisation des transistors



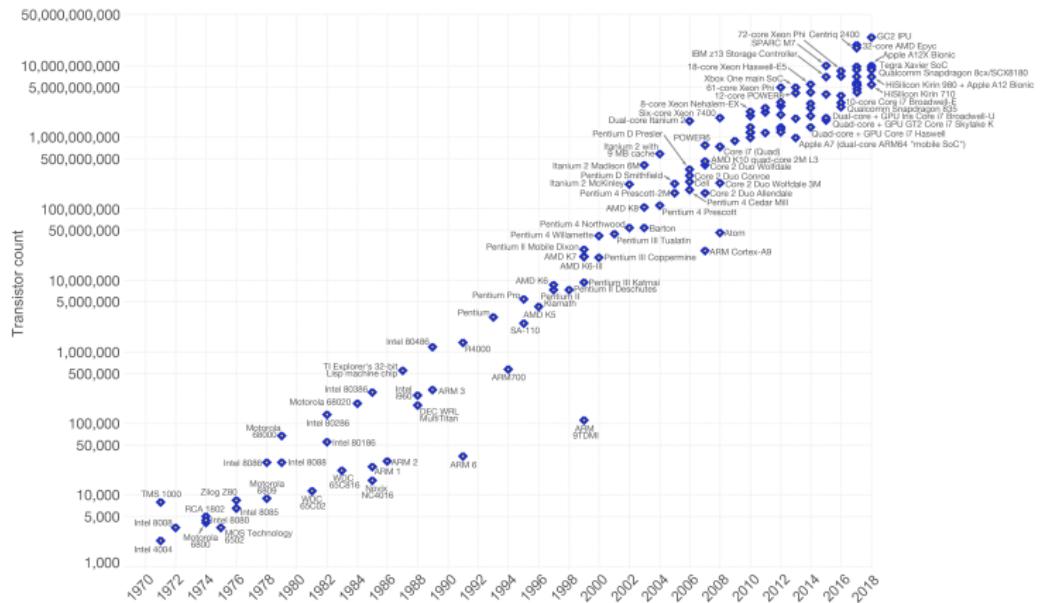
Loi de Moore

Le nombre de transistors des microprocesseurs double tous les 2 ans

Gordon Moore (Fairchild Semiconductor), 1975

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

The data visualization is available at OurWorldInData.org. There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Röser.

Fondements

Usage du transistor

► stockage SSD (solid-state drive)

- ▶ Permet le stockage de données sur de la mémoire flash (mémoire de masse à semi-conducteurs ré-inscriptible).



- ▶ Technologie transistor MOSFET à grille flottante (FG-MOSFET)

Pour approfondir : <https://couleur-science.eu/?d=2015/11/11/17/23/>

08-comment-fonctionne-la-memoire-flash-dun-lecteur-ssd

Fonctionnement des ordinateurs

chapitre IV : fondements (2/3)

Prof. Xavier Gandibleux

Université de Nantes
Département Informatique – UFR Sciences et Techniques

Année académique 2019-2020

De la logique à l'arithmétique

conventions et définitions
nombres et bases
addition sur 1 bit
addition sur k bits
nombres négatifs
additionneur complet

Fondements

De la logique à l'arithmétique : conventions et définitions

Fondements

Représentation :

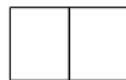
Avec : **1 bit**,



2 informations différentes :

0
1

2 bits,



4 informations différentes :

0	0
0	1
1	0
1	1

3 bits,



8 informations différentes :

k bits $\Rightarrow 2^k$ informations différentes

Fondements

Quelques définitions :

- ▶ bit, byte (octet), word (mot) :

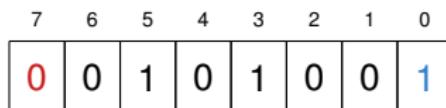
	Définition
$k = 1$	bit
$k = 8$	byte (octet)
k	word (mot)

- ▶ MSB (bpF), LSB (bpf) :

Convention de numérotation des bits (et des bytes) :

- ▶ big-endian (droite-à-gauche)
- ▶ little-endian (gauche-à-droite)

‘



MSB : Most Significant bit
Bit de poids fort (bpF)

LSB : Least Significant bit
Bit de poids faible (bpf)

Fondements

De la logique à l'arithmétique : nombres et bases

Fondements

Bases

L'humain :

- ▶ 10 symboles : 0,1,2,3,4,5,6,7,8,9
 - ↳ **base 10** : décimale

L'ordinateur :

- ▶ 2 symboles : 0,1
 - ↳ **base 2** : binaire
- ▶ 8 symboles : 0, 1, 2, 3, 4, 5, 6, 7
 - ↳ **base 8** : octal
- ▶ 16 symboles : 0, 1, 2, ..., 7, 8, 9, A, B, C, D, E, F
 - ↳ **base 16** : hexadécimal

→ Opérations de changement de base

Fondements

Nombres naturels et bases

Algorithme de conversion d'un nombre en base 2 vers base 10 :

Exemple :

$$\begin{array}{ccccc} 0 & 1 & 1 & 1 & 0 \\ 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 \times 2^4 & 1 \times 2^3 & 1 \times 2^2 & 1 \times 2^1 & 0 \times 2^0 \\ 0 & + & 8 & + & 4 & + & 2 & + & 0 & = & 14 \\ \Rightarrow 01110_2 & = & 14_{10} \end{array}$$

Fondements

Nombres naturels et bases

Généralisation de la conversion d'un nombre n en base b vers base 10

Soit le nombre

$$n_b = d_{k-1} \ d_{k-2} \ \dots \ d_2 \ d_1 \ d_0$$

composé de k chiffres et exprimé dans une base b avec

- ▶ d_{k-1} le chiffre de poids fort et
- ▶ d_0 le chiffre de poids faible

Alors la conversion de n en base $b = \{2, 8, 16\}$ vers la base 10 vaut :

$$n_{10} = \sum_{i=0}^{k-1} (d_i \times b^i)$$

Fondements

Nombres naturels et bases

Exemple :

$$\begin{aligned}10011_2 &= 23_8 \\&= 13_{16} \\&= 34_5 \\&= ?_{10}\end{aligned}$$

Fondements

Nombres naturels et bases

Algorithme de conversion d'un nombre en base 10 vers base 2 :

Exemple :

$$\begin{array}{r} 11 \\ | \quad \quad \quad 2 \\ \hline \end{array}$$

Fondements

Nombres naturels et bases

Généralisation de la conversion d'un nombre n en base 10 vers base b

Algorithme des divisions successives :

n est itérativement divisé par b jusqu'à obtenir un quotient égal à 0.

La conversion du nombre n dans la base b est obtenue en notant les restes de chacune des divisions effectuées depuis la dernière division jusqu'à la première.

Exemples :

$$\begin{array}{r} 131 \longdiv{2} \\ 120 \quad 65 \longdiv{2} \\ \hline 11 \quad 60 \quad 32 \longdiv{2} \\ 10 \quad 5 \quad 20 \quad 16 \longdiv{2} \\ \hline 1 \quad 4 \quad 12 \quad 16 \quad 8 \longdiv{2} \\ 1 \quad 12 \quad 0 \quad 8 \longdiv{2} \\ \hline 0 \quad 0 \quad 4 \quad 2 \longdiv{2} \\ \hline 0 \quad 2 \quad 1 \\ \hline 0 \end{array}$$

$$131 = 10000011_2$$

Fondements

Nombres naturels et bases

Algorithme de conversion entre base 2 et 8 (resp. 16) :

- ▶ Chaque groupe de 3 (resp. 4) bits d'un nombre binaire, en partant du LSB, peut être directement exprimé par un chiffre de 0 à 7 (resp. F).
- ▶ Réciproquement, chaque chiffre de 0 à 7 (resp. F) d'un nombre octal (resp. hexadécimal), en partant du LSB, peut être directement exprimé par sa représentation binaire.

PS: Il peut être nécessaire de compléter le groupe de bits avec des 0 non significatifs pour obtenir le nombre binaire correspondant sur 3 (resp. 4) bits.

Exemple :

Hexadécimal

Binaire 0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1

Octal

Fondements

Nombres naturels et bases

Fonctionnalités disponibles dans certains langages de programmation

Exemples en Julia v1.x (<https://julialang.org>) :

- ▶ manipule des nombres en base 2, 8, 10 et 16.
Exemple pour 13_{10} représenté en base :
 - ▶ $2 \rightarrow 0b1101$
 - ▶ $8 \rightarrow 0o15$
 - ▶ $16 \rightarrow 0xD$
- ▶ présente des fonctions de conversions d'un nombre n donné en base 2, 8, 10, ou 16. Exemples (retourne un string) :
 - ▶ `string(n, base=2)`
 - ▶ `string(n, base=8)`
 - ▶ `string(n, base=16)`
 - ▶ `string(n, base=b) # vers une base b quelconque`
- ▶ Illustration dans le terminal interactif REPL (read-eval-print loop) de Julia

```
Last login: Sun Jan  5 18:51:14 on ttys009
exec '/Applications/Julia-1.2.app/Contents/Resources/julia/bin/julia'
MacBook-Pro-de-xavierg:~ xavierg$ exec '/Applications/Julia-1.2.app/Contents/Resources/julia/bin/julia'

julia> n10=13
[13]

julia> typeof(n10)
[Int64]

julia> string(n10, base=2)
["1101"]

julia> string(n10, base=8)
["15"]

julia> string(n10, base=16)
["d"]

julia> string(n10, base=5)
["23"]

julia> typeof(string(n10, base=5))
[String]

julia> n2 = 0b1101
[0x0d]

julia> n8=0o15
[0x0d]

julia> n16=0xD
[0x0d]

julia> typeof(n16)
[UInt8]

julia> 
```

xavierg — julia — 124x28

```
julia> using Printf
[jl]
julia> for i=0:20
|   @printf("%3d | %5s %3s %3s | %3s \n", i, string(i,base=2), string(i,base=8), string(i,base=16), string(i,base=5))
| end
0 |     0   0   0 |    0
1 |     1   1   1 |    1
2 |     10  2   2 |    2
[ 3 |     11  3   3 |    3
4 |     100 4   4 |    4
5 |     101 5   5 |   10
6 |     110 6   6 |   11
7 |     111 7   7 |   12
8 |    1000 10  8 |   13
9 |    1001 11  9 |   14
10 |   1010 12  a |   20
11 |   1011 13  b |   21
12 |   1100 14  c |   22
13 |   1101 15  d |   23
14 |   1110 16  e |   24
15 |   1111 17  f |   30
16 |  10000 20  10 |   31
17 |  10001 21  11 |   32
18 |  10010 22  12 |   33
19 |  10011 23  13 |   34
20 |  10100 24  14 |   40
julia> 
```

Fondements

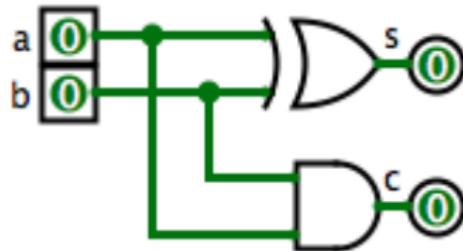
De la logique à l'arithmétique : addition sur 1 bit

Fondements

De la logique à l'arithmétique

- ▶ on compte en binaire : faux = 0, vrai = 1
- ▶ addition :

	0	1
0	00	01
1	01	10



demi additionneur

Fondements

De la logique à l'arithmétique : addition sur k bits (part 1)

Fondements

Arithmétique binaire

- ▶ addition : comme en base 10 (en plus simple...)

$$\begin{array}{r} & 1 & 0 & 1 & 0 & 10_{10} \\ + & 0 & 0 & 1 & 1 & 3_{10} \\ \hline & & 1 & & & \\ = & 1 & 1 & 0 & 1 & 13_{10} \end{array}$$

- ▶ soustraction : addition de l'opposé
 - ↳ distinguer les nombres **signés** et **non signés**
 - ↳ représenter les nombres négatifs

Fondements

De la logique à l'arithmétique : nombres négatifs

Fondements

Représentation des nombres négatifs par bit de signe

Pour un entier de k bits,

- ▶ le MSB code le signe; la convention de codage est :
 - ▶ 0 = positif
 - ▶ 1 = négatif
- ▶ les $k - 1$ bits restants représentent la valeur absolue de l'entier

Exemple (sur $k = 8$ bits) :

- ▶ $+25_{10} = 00011001_2$
- ▶ $-25_{10} = 10011001_2$

Inconvénient(s) ?

Intervalle de représentation ?

Fondements

Représentation des nombres négatifs par complément à 2

Pour un entier de k bits,

- ▶ le MSB code le signe; la convention de codage est :
 - ▶ 0 = positif
 - ▶ 1 = négatif
- ▶ Un nombre négatif s'obtient en ajoutant 1 au complément à 1 de sa valeur absolue (et inversement).

Exemple (sur $k = 8$ bits) :

- ▶ $+25_{10} = 00011001_2$
- ▶ $-25_{10} = \overline{00011001}_2 + 1_2 = 11100110_2 + 1_2 = 11100111_2$

Inconvénient(s) ?

Intervalle de représentation ?

Fondements

Représentation des nombres négatifs par complément à 2

Examen de plus près :

Non signé ($k = 3$)

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

Signé ($k = 3$)

0 000	\leftarrow
1 001	
2 010	
3 011	
100 -4	
101 -3	
110 -2	
111 -1	$+1$

Sur k bits, $-n$ est représenté par $2^k - n$

Exemple : sur 4 bits,

- ▶ -3 est représenté par $16 - 3 = 13$
- ▶ $-3 \rightarrow \overline{0011} + 1 = 1100 + 1 = 1101$

Fondements

De la logique à l'arithmétique : addition sur k bits (part 2)

Fondements

Arithmétique binaire

- addition : addition usuelle

Exemple, $10_{10} + 3_{10}$ et $k = 4$

$$\begin{array}{r} 1 & 0 & 1 & 0 \\ + & 0 & 0 & 1 & 1 \\ \hline = & 1 & 1 & 0 & 1 \end{array} \quad \begin{array}{l} 10_{10} \\ 3_{10} \\ \hline 13_{10} \end{array}$$

- soustraction : addition de l'opposé (en complément à 2)

Exemple, $10_{10} - 3_{10}$ et $k = 4$

$$\begin{array}{r} 1 & 0 & 1 & 0 \\ + & 1 & 1 & 0 & 1 \\ \hline = & 0 & 1 & 1 & 1 \end{array} \quad \begin{array}{l} 10_{10} \\ -3_{10} \\ \hline 7_{10} \end{array}$$

En effet,

$$10 - 3 \text{ devient } 10 + (16 - 3) = 10 + 13 = 23 = \color{red}{16} + \color{green}{7}$$

Fondements

Arithmétique binaire

Débordement lors de l'addition ou la soustraction de 2 nombres

- ▶ opérandes de signes différents → pas débordement possible
- ▶ opérandes de mêmes signes → débordement possible

Exemples :

- ▶ sur 4 bits, le plus grand entier positif est $0111 = 7$
- ▶ sur 4 bits, le plus petit entier négatif est $1000 = -8$
- ▶ débordement :

$$\begin{array}{rcl} \begin{array}{l} \text{▶ } \\ \text{→ } \end{array} & \begin{array}{rcl} 5 + 4 & = & 0101 + 0100 & = & 1001(9) \\ \overline{1001} + 1 & = & 0110 + 1 & = & 0111(-7) \end{array} & \end{array}$$

▶ $-5 + -4 = 1011 + 1100 = 0111(7)$

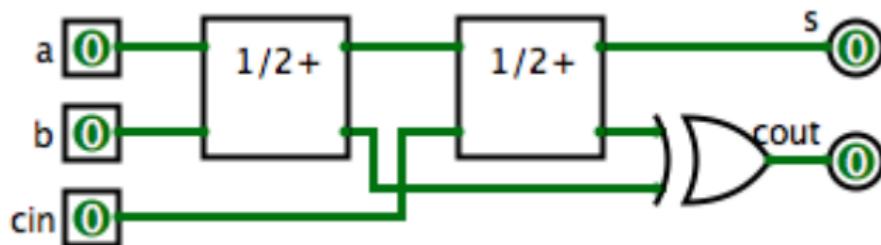
- ▶ Détection **retenue (C : carry)** :
↳ 1 bit de plus en sortie
- ▶ Détection **débordement (V : overflow)** :
↳ opérandes de même signe, résultat de signe différent

Fondements

De la logique à l'arithmétique : additionneur complet

Additionneur complet

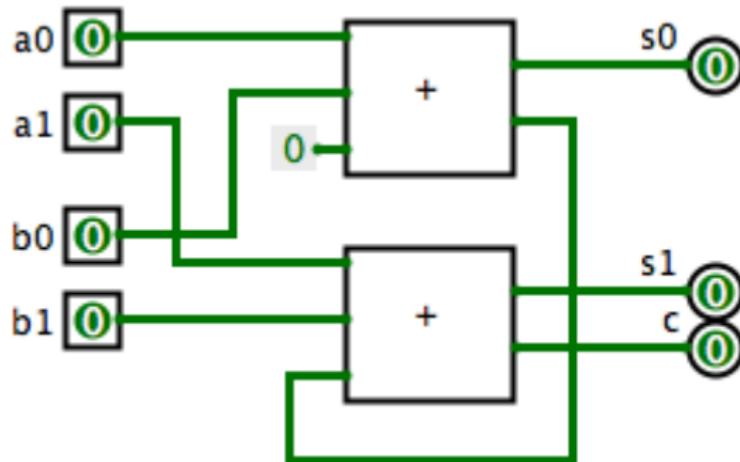
- ▶ a et b : deux bits à additionner
- ▶ c_{in} : retenue venant des additions précédentes



Fondements

Additionneur à 2 bits

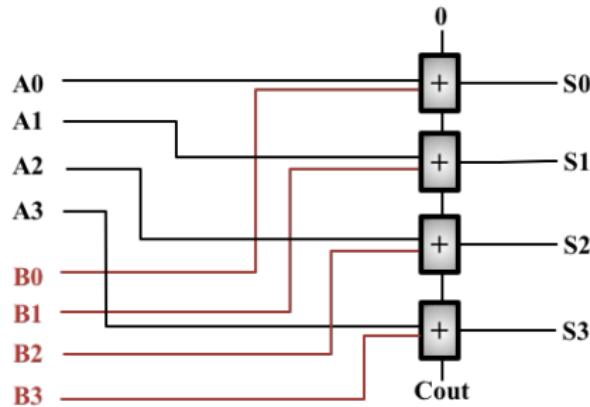
- ▶ $a=a_1a_0$
- ▶ $b=b_1b_0$



Fondements

Généralisation

- ▶ calculer l'addition, la différence de deux nombres binaires de k bits (pour $k = 4$) :

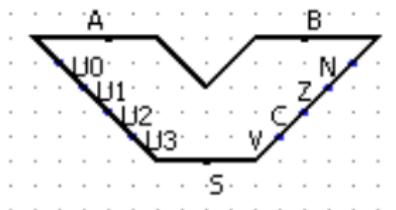


- ▶ Fonctionnalités regroupées dans un composant essentiel d'un processeur : **l'Unité Arithmétique et Logique (ALU)**

Fondements

Généralisation

- ▶ Principe de l'ALU : un composant qui



- ▶ prend en entrée
 - ▶ deux nombres binaires (A, B)
 - ▶ un code (U1 ... U4) indiquant l'opération à faire avec A et B
- ▶ donne en sortie
 - ▶ le résultat de l'opération (S)
 - ▶ des indications sur ce résultat : retenue (C), débordement arithmétique (V), résultat nul (Z) et résultat négatif (N).

Fonctionnement des ordinateurs

chapitre IV : fondements (3/3)

Prof. Xavier Gandibleux

Université de Nantes
Département Informatique – UFR Sciences et Techniques

Année académique 2019-2020

De la logique combinatoire à la logique séquentielle

circuits séquentiels
bascules
registres

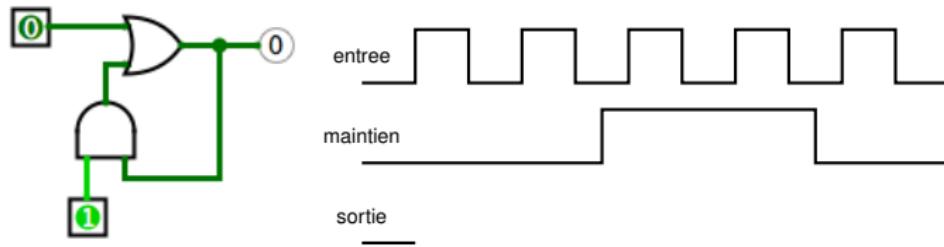
Fondements

De la logique combinatoire à la logique séquentielle :
circuits séquentiels

Logique séquentielle

- ▶ Les circuits vus jusqu'ici calculent des fonctions
 - ↳ pas d'état et ne mémorisent pas de valeurs
- ▶ Les circuits suivant permettent de conserver un état donné
 - ↳ exemple : mémoriser une valeur

Analyse du fonctionnement à l'exécution sous logisim :



Fondements

De la logique combinatoire à la logique séquentielle :
bascules

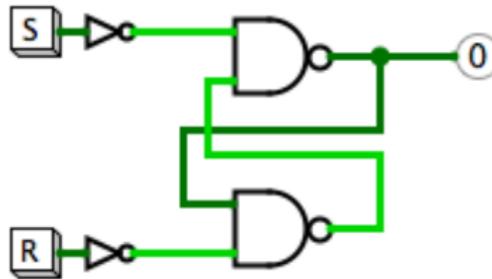
Fondements

Bascule SR

Principe :

- ▶ S (Set) = 1 $\Rightarrow Q$ (sortie) = 1
- ▶ R (Reset) = 1 $\Rightarrow Q$ (sortie) = 0
- ▶ $S=R=0 \Rightarrow$ la bascule est en mode ‘mémoire’
- ▶ $S=R=1$: état interdit

Analyse du fonctionnement à l'exécution :



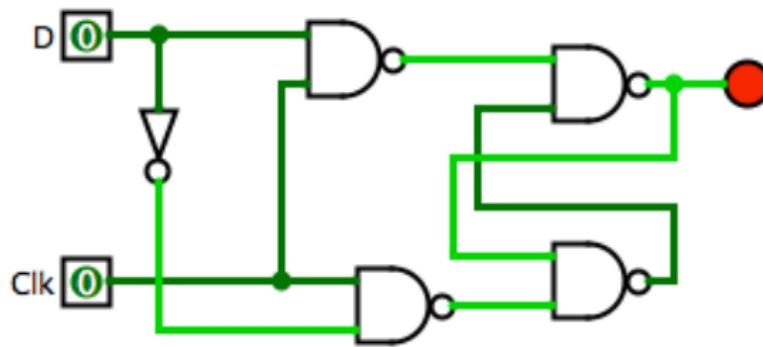
Fondements

Bascule D simple

Principe :

- ▶ Clk (Clock) : horloge périodique
- ▶ D (data) : donnée à mémoriser
- ▶ Q=D (sortie) quand Clk=1
- ▶ bascule en mode 'mémoire' quand Clk=0

Analyse du fonctionnement à l'exécution :



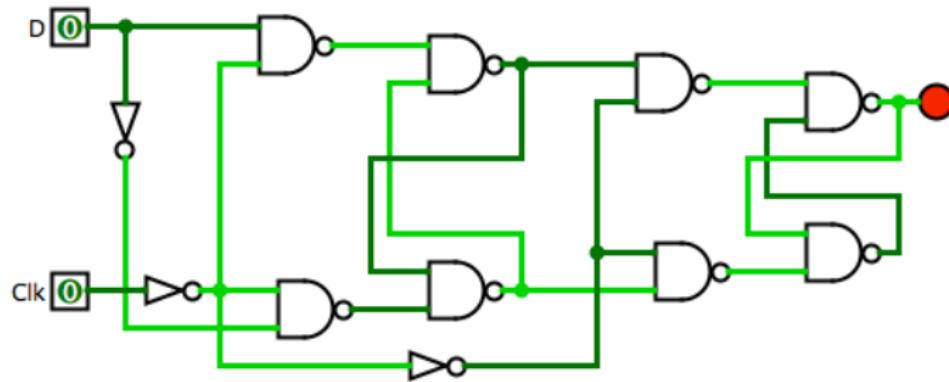
Fondements

Bascule D maître-esclave

Principe :

- ▶ Clk (Clock) : horloge périodique
- ▶ D (data) : donnée à mémoriser
- ▶ Q=D (sortie) quand transition 0 \nearrow 1 de Clk \Rightarrow synchrone
- ▶ bascule en mode 'mémoire' sinon

Analyse du fonctionnement à l'exécution :



Fondements

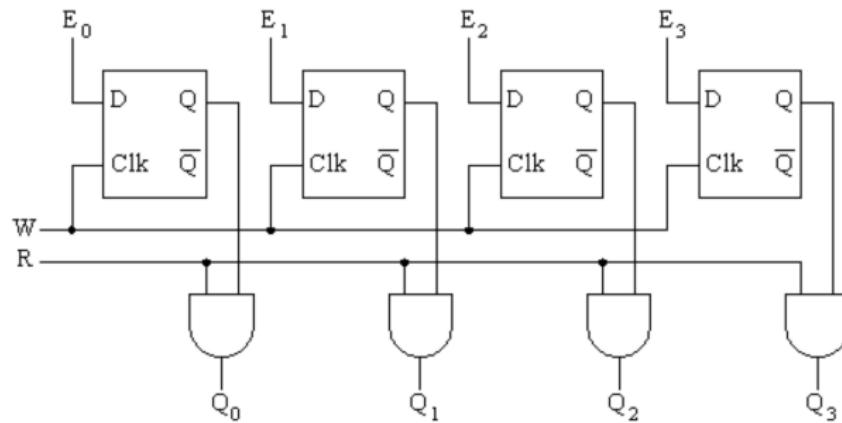
De la logique combinatoire à la logique séquentielle :
Registres

Fondements

Registre

Principe :

- ▶ Exemple : registre de mémorisation de 4 bits :



Analyse du fonctionnement à l'exécution

Suite...

Ordinateurs