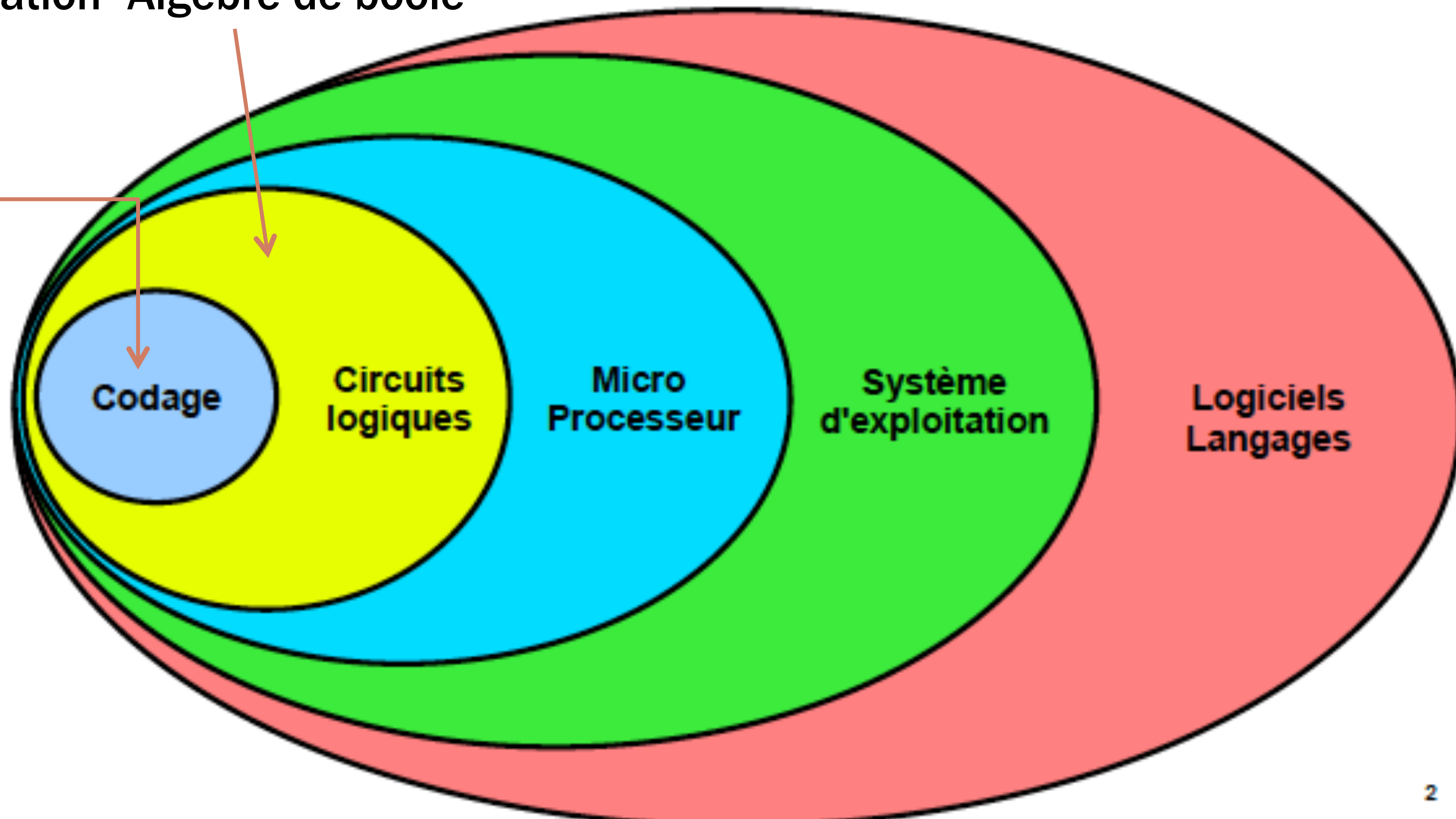
NUMÉRATION, CODAGE ET ALGÈBRE DE BOOLE

Introduction

INTRODUCTION

- But: **Représentation interne des informations**



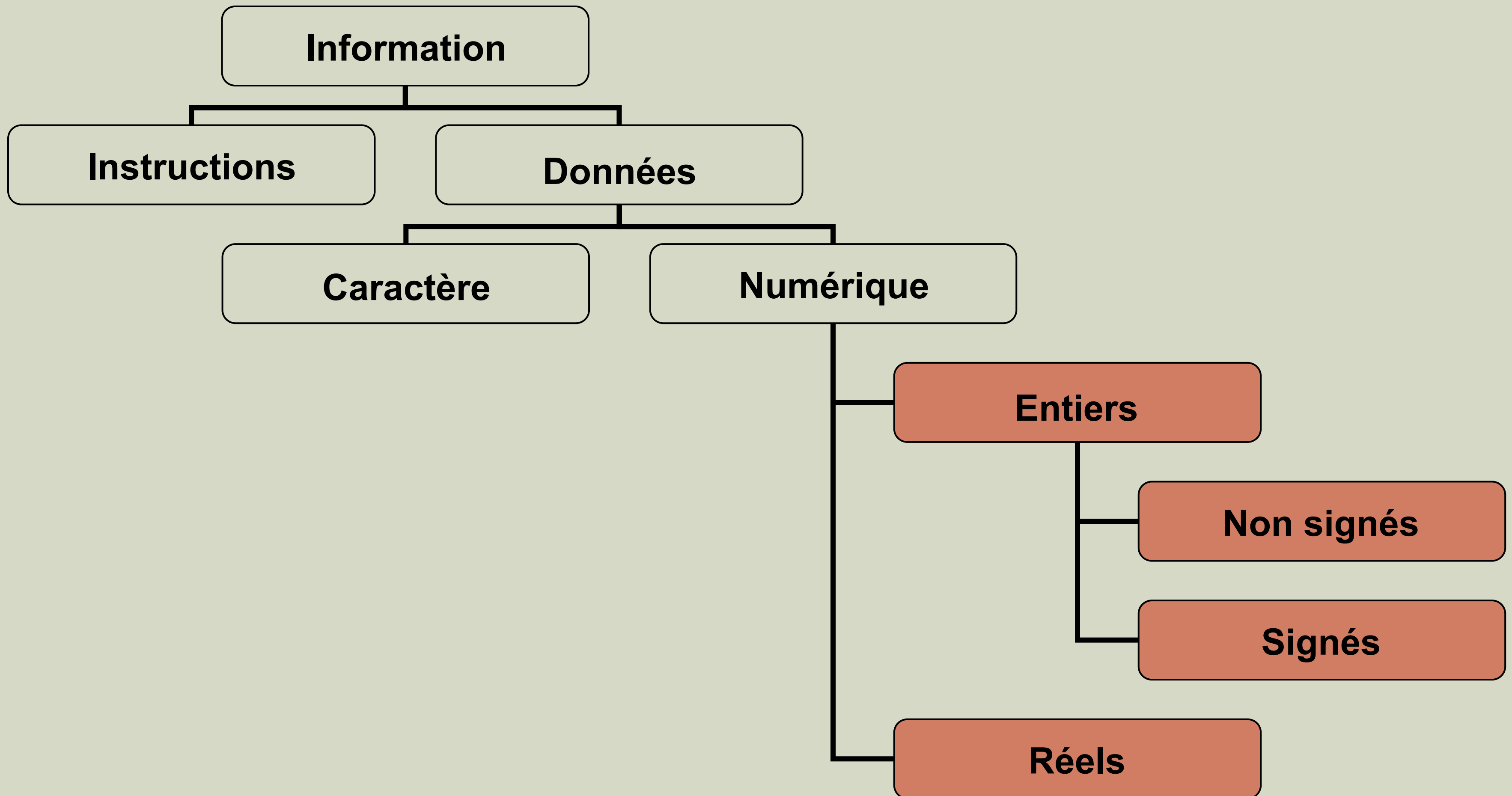
OBJECTIFS

- Les ordinateurs permettent de traiter rapidement de nombreuses données dans de nombreux domaines :
- Calcul scientifique
- Ingénierie
- Bases de données
- Finance
- Etc.

- Quelque soit votre futur métier, vous utiliserez des outils informatiques pour faire des calculs (calculatrice, tableur, etc)

- Vous devez donc **connaitre les limites**, **les biais** et **les erreurs** du calcul par ordinateur.

TYPE D'INFORMATION

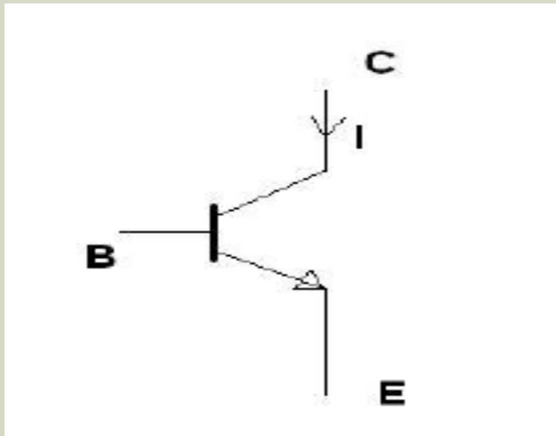


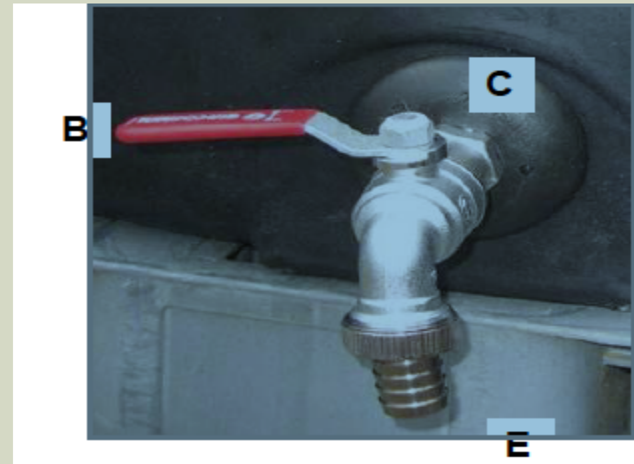
OBJECTIFS

- Créer des programmes (des applications) plus sûrs:
- Ariane 501, le 28/02/1996 a explosée une valeur a dépassée la plage des valeurs autorisées
  - En fait, dans un ordinateur, les capacités de stockage sont limitées
  - chaque nombre ne peut dépasser certaines valeurs (soit trop grandes, soit trop petites)
- Même simple programmeur ou simple utilisateur:
  - Imaginez votre compte en banque passer 99 999€ à 0€ lorsque vous lui ajoutez 1 €.
  - Voir aussi le problème du bug de l'an 2000 (passer de l'an 99 à l'année 00!)
  - Il peut également s'agir d'un simple problème d'arrondi: 0,8 devient 0,799.

QU'EST-CE QUE LE CODAGE ?

- Les ordinateurs ne savent traiter que des 0 et des 1 car ils sont basés sur les transistors :





- Schéma d'un transistor... dont le fonctionnement est similaire à celui d'un robinet ! Le courant I passe (1) ou ne passe pas (0)... L'eau coule ou ne coule pas.
- La tension en B permet de faire passer ou non le courant I... Le levier B permet de faire couler ou non l'eau

QU'EST-CE QUE LE CODAGE ?

- Les transistors sont la base des microprocesseurs
- Si l'on souhaite mémoriser un nombre tel 2 ou 3 ou plus, il faut trouver un moyen pour le représenter uniquement avec des 0 et des 1.
- le codage est donc le moyen qui permet de mémoriser dans l'ordinateur toute sorte de nombre, et plus généralement toute sorte d'information.
- **Avant d'entamer le codage, un chapitre de rappel sur la numération.**

CHAPITRE 1: LA NUMÉRATION
(VOIR SUPPORT DE COURS)

- Système de Numération
  - Définitions
- Système de Numération binaire
  - Conversion Binaire-Décimal
  - Conversion Décimal-Binaire
  - Cas de nombres fractionnaires
  - Précision fractionnaire
- Autres Systèmes de numérations
  - Octale
  - Hexadécimale
  - Transcodage ou changement de base(2,8 ou16)

BASES DE NUMÉRATION (BINAIRE, OCTALE ET HEXADÉCIMALE)

- Système binaire (b=2) utilise deux chiffres : **{0,1}**
  - **C'est avec ce système que fonctionnent les ordinateurs**

- Système Octale (b=8) utilise huit chiffres :**{0,1,2,3,4,5,6,7}**
  - **Utilisé il y a un certain temps en Informatique..**
  - **Elle permet de coder 3 bits par un seul symbole..**

- Système Hexadécimale (b=16) utilise 16 chiffres : **{0,1,2,3,4,5,6,7,8,9,A=$10_{(10)}$,B=$11_{(10)}$,C=$12_{(10)}$,D=$13_{(10)}$,E=$14_{(10)}$,F=$15_{(10)}$}**
  - **Cette base est très utilisée dans le monde de la micro informatique..**
  - **Elle permet de coder 4 bits par un seul symbole..**

TRANSCODAGE (OU CONVERSION DE BASE)

- Le transcodage (ou conversion de base) est l’opération qui permet de passer de la représentation d’un nombre exprimé dans une base à la représentation du même nombre mais exprimé dans une autre base..

- les conversions suivantes:

  - Décimale vers Binaire, Octale et Hexadécimale (vu précédemment)
  - Binaire vers Décimale (vu précédemment)

  - Binaire vers l’octal ou l’hexadécimal
    - Solution 1: passer par le décimale
    - Solution 2: sans passer par le décimale

DE LA BASE BINAIRE VERS UNE BASE B -SOLUTION 2-

- **Binaire vers octale : regroupement des bit en des sous ensemble de trois bits puis remplacé chaque groupe par le symbole correspondant dans la base 8.**
- **Exemple:**

$$\begin{aligned} N &= 001 \quad 010 \quad 011 \quad 101_2 \\ &= 1 \quad 2 \quad 3 \quad 5_8 \end{aligned}$$

$$1010011101_2 = 1235_8$$

- **Binaire vers Hexadécimale: regroupement des bit en des sous ensemble de quatre bits puis remplacé chaque groupe par le symbole correspondant dans la base 16.**

$$\begin{aligned} N &= 0010 \quad 1001 \quad 1101_2 \\ &= 2 \quad 9 \quad D_{16} \end{aligned}$$

$$1010011101_2 = 29D_{16}$$

CHAPITRE 2: CODAGE DE L'INFORMATION

- Codage des nombres
  - Codage des entiers positifs (binaire pur)
  - Codage des entiers relatifs (complément à 2)
  - Codage des nombres réels (virgule flottante)

- Codage des caractères :
  - ASCII et
  - ASCII étendu ,
  - Unicode, …

- Codage du son et des images

DÉFINITION

- Définition :

permet d'établir une correspondance qui permet sans ambiguïté de passer d'une représentation (dite externe) d'une information à une autre représentation (dite interne: sous forme binaire) de la même information, suivant un ensemble de règle précise.

- Le codage en informatique s 'effectue principalement en trois étapes:

1. L'information sera exprimée par une suite de nombres (chapitre 1 Numérisation)
2. Chaque nombre est codé sous forme binaire (suite de 0 et 1)
3. Chaque élément binaire est représenté par un état physique

DÉFINITION
CAPACITÉ ET TAILLE DU CODAGE

Un nombre est représenté en format fixe par l chiffres dans sa base binaire. Il s'écrit donc :

$N_2 = a_n a_{n-1} \ldots\ldots a_1 a_0$ avec l = n+1.

- Le nombre l s'appelle la taille du codage
- La quantité de nombres de l chiffres qu'il est possible de représenter dans ce cas s'appelle : la capacité de représentation
- Elle est définit par C:

$C = (N_{max})_{10}$ + représentation du zéro $= N_{max} + 1 = 2^l$ (dans le cas binaire b=2)

- On a aussi pour n'importe quelle base b

$$l = \log_b C = \frac{Ln(C)}{Ln(b)}$$

CODAGE DES ENTIERS POSITIFS

1. L'entier naturel (positif ou nul) est représenté en base 2,
2. Les bits sont rangés selon leur poids, on complète à gauche par des 0 (bit de signe).

<u>Exemple:</u> le code de $(7)_{10}$ est $(7)_{10} = (0111)_2$

Codage sur n bits: représentation des nombres de 0 à 2^n-1

- Sur 1 octet: 0 à 255
- Sur 2 octets: 0 à 65535
- Sur 4 octets: 0 à 4 294 967 295

ARITHMÉTIQUE EN BASE 2
RAPPEL

- Les opérations sur les entiers s'appuient sur des tables d'addition et de multiplication :

- Addition binaire (8 bits)

```
  10010110
+ 01010101
----------
  11101011
```

- Addition binaire (8 bits) avec (débordement ou overflow) :

```
  10010110
+ 01110101
----------
 100001011
```

overflow

- Multiplication binaire

```
       1 0 1 1   (4 bits)
*      1 0 1 0   (4 bits)
-----------------
       0 0 0 0
     1 0 1 1 .
   0 0 0 0 .
 1 0 1 1 .
-----------------
0 1 1 0 1 1 1 0
```

Sur 4 bits le résultat est faux

Sur 7 bits le résultat est juste

Sur 8 bits on complète à gauche par un 0

CODAGE DES ENTIERS RELATIFS

- Il existe au moins trois façons pour coder :
  - code binaire signé (par signe et valeur absolue)
  - code complément à 1
  - code complément à 2 (Utilisé sur ordinateur)

CODAGE DES NOMBRES RELATIFS -BINAIRE SIGNÉ-

- Le bit le plus significatif est utilisé pour représenter le signe du nombre :
  - si le bit le plus fort = 1 alors nombre négatif
  - si le bit le plus fort = 0 alors nombre positif
- Les autres bits codent la valeur absolue du nombre

- Exemple : Sur 8 bits, codage des nombres -24 et -128 en (bs)

- -24 est codé en binaire signé par : $1\ 0\ 0\ 1\ 1\ 0\ 0\ 0_{(bs)}$
- -128 hors limite, nécessite 9 bits au minimum

CODAGE DES NOMBRES RELATIFS -BINAIRE SIGNÉ- (SUITE)

- Avec n bits, on code tous les nombres entre $\left[-(2^{n-1}-1),(2^{n-1}-1)\right]$
- Avec 4 bits : -7 et +7
- Limitations du binaire signé:
  - Deux représentations du zéro : + 0 et - 0
  - Sur 4 bits : +0 = $0000_{(bs)}$, -0 = $1000_{(bs)}$
  - Multiplication et l'addition sont moins évidentes.

CODAGE DES ENTIERS RELATIFS (CODE COMPLÉMENT À 1)

- Aussi appelé Complément Logique (CL) ou Complément Restreint (CR) :

  1. les nombres positifs sont codés de la même façon qu'en binaire signé,
  2. un nombre négatif est codé en inversant chaque bit de la représentation de sa valeur absolue en binaire signé

- Le bit le plus significatif est utilisé pour représenter le signe du nombre :
  - si le bit le plus fort = 1 alors nombre négatif
  - si le bit le plus fort = 0 alors nombre positif

CODAGE DES ENTIERS RELATIFS -CODE COMPLÉMENT À 1- (SUITE)

- Exemple : -24 en complément à 1 sur 8 bits
  1. |-24|en binaire pur $0\ 0\ 0\ 1\ 1\ 0\ 0\ 0_{(2)}$ puis
  2. on inverse les bits $1\ 1\ 1\ 0\ 0\ 1\ 1\ 1_{(cà1)}$

- Limitation :

- deux codages différents pour 0 (+0 et -0)
- Sur 8 bits : $+0=0\ 0\ 0\ 0\ 0\ 0\ 0\ 0_{(cà1)}$ et $-0=1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_{(cà1)}$
- Multiplication et l'addition sont moins évidentes.

REMARQUES

- **Remarque1:**

Notons CA1 la fonction qui inverse tous les bits un par un d 'un n-uplet.

Alors $CA_1(N)$ est le nombre en base 2 où tous les bits de N ont été inversés un par un.

(Attention c'est différent du codage en complément à 1, quelle est la différence?)

On a : $N+CA_1(N)=2^n-1$

$CA_1(CA_1(N))= N$

Où n : est le nombre de bits de la représentation du nombre N .

Exemple :

Soit N=1010 sur 4 bits donc:

$CA_1(N)= (2^4 - 1)-N$

$CA_1(N)=(16-1)-(1010)_2= (15) - (1010)_2 = (1111)_2 - (1010)_2 = 0101$

- Quel est l 'intervalle qu'on peut coder en complément à 1?

CODAGE DES ENTIERS RELATIFS -CODE COMPLÉMENT À 2- (1)

- Aussi appelé Complément Vrai (CV) :
  - les nombres positifs sont codés de la même manière qu'en binaire pure.
  - un nombre négatif est codé en ajoutant la valeur 1 à son complément à 1

- Le bit le plus significatif est utilisé pour représenter le signe du nombre

- Exemple : -24 en complément à 2 sur 8 bits
  - 24 est codé par $0\ 0\ 0\ 1\ 1\ 0\ 0\ 0_{(2)}$
  - -24 $1\ 1\ 1\ 0\ 0\ 1\ 1\ 1_{(cà1)}$
  - donc -24 est codé par $1\ 1\ 1\ 0\ 1\ 0\ 0\ 0_{(cà2)}$

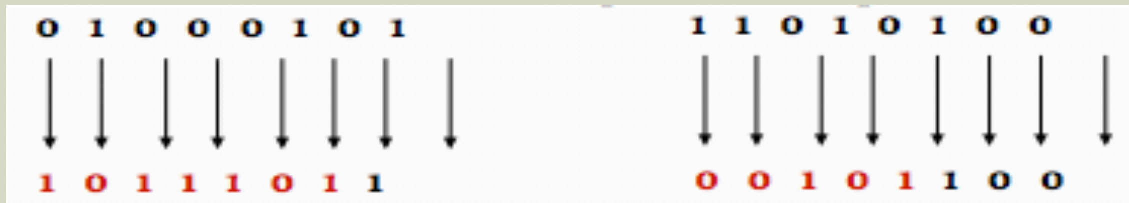
CODAGE DES ENTIERS RELATIFS -CODE COMPLÉMENT À 2- (2)

- Un seul codage pour 0. Par exemple sur 8 bits :
  - +0 est codé par $00000000_{(cà2)}$
  - -0 est codé par $11111111_{(cà1)}$
  - Donc -0 sera représenté par $00000000_{(cà2)}$

- Avec n bits, on peut coder de $-(2^{n-1})$ à $(2^{n-1}-1)$
- Sur 1 octet (8 bits), codage des nombres de -128 à 127
  - +0 = 00000000 -0=00000000
  - +1 = 00000001 -1=111111111
  - … …
  - +127= 01111111 -128=10000000

REMARQUES

1. Pour trouver le compétemment à 2 d'un nombre : il faut parcourir les bits de ce nombre à partir du poids faible et garder tous les bits avant le premier 1 et inverser les autres bits qui viennent après.



2. Notons $CA_2(N)$ le nombre en base 2 où tous les bits de N_2 ont été inversés un par un ensuite on lui ajouté le nombre 1

(Attention c'est différent du codage en complément à 2, quelle est la différence)

On a :

$$CA_2(CA_2(N)) = N$$

et :

$$N+CA2(N) = ?$$

REMARQUES

3. Pour effectuer une soustraction, il suffit de faire une addition avec le complément à deux. Le résultat se lit directement en complément à deux :
   - si le signe est + la lecture est directe,
   - si le signe est - on convertit le résultat en recherchant le complément à deux de celui-ci.

4. Comparer $(+0)_{cà2}$ et $(-0)_{cà2}$

5. Donner l intervalle de codage en complément à2 pour un codage en3 bits.

6. Déduire l'intervalle du codage pour n bits

OPÉRATIONS ARITHMÉTIQUES EN CA_2

Effectuer les opérations suivantes sur 5 Bits , en utilisant la représentation en CA_2

+ 9	0 1 0 0 1
+ 4	+ 0 0 1 0 0
+ 13	0 1 1 0 1

Le résultat est positif

$(01101)_2 = (13)_{10}$

+ 9	0 1 0 0 1
- 4	+ 1 1 1 0 0
+ 5	1 0 0 1 0 1

Report

Il suffit d'ignorer le bit de report à gauche

Le résultat est positif

$(00101)_2 = (5)_{10}$

-9	
-4	
-13	

```
      1 0 1 1 1
+
      1 1 1 0 0
  ___________
  1 1 0 0 1 1
```

Report

Le résultat est négatif :

Résultat = - CA2 (10011)= -(01101)

= - 13

-9	
+9	
+0	

```
      1 0 1 1 1
+
      0 1 0 0 1
  ___________
  1 0 0 0 0 0
```

Report

Le résultat est positif

$(00000)_2 = (0)_{10}$

LA RETENUE ET LE DÉBORDEMENT

- On dit qu'il y a une **retenue** si une opération arithmétique génère un report (il suffit alors d'ignorer ce le bit de report).

- On dit qu'il y a un **débordement (Over Flow) ou dépassement de capacité:** si le résultat de l'opération sur **n** bits et **faux** .

  - Le nombre de bits utilisés est insuffisant pour contenir le résultat
  - Autrement dit le résultat dépasse l'intervalle des valeurs sur les **n** bits utilisés.

CAS DE DÉBORDEMENT

	retenues	0 1
+9		0 1 0 0 1
+8	+	0 1 0 0 0
+17		**1** 0 0 0 1

Négatif

	retenues	1 0
-9		1 0 1 1 1
-8	+	1 1 0 0 0
-17		**0** 1 0 1 1

Positif

Le débordement est détecté si:

- la somme de deux nombres positifs donne un nombre négatif .
- Ou la somme de deux nombres négatifs donne un Nombre positif.
- Identiquement équivalent à la retenue entrante sur le bit de poids fort est différente de la retenue sortante.

EXTENSION DE SIGNE EN ARITHMÉTIQUE SIGNÉE

- Entier signé sur n bits → entier signé sur n + k bits.

Méthode

On rajoute des bits à gauche de la même valeur que le bit de signe.

5_{10} = (sur 4 bits) = (sur 8 bits)

-4 = (sur 4 bits) = (sur 8 bits)

Permet de coder un nombre avec plus de bits

CHAPITRE 2: CODAGE DE L'INFORMATION

- Codage des nombres
  - Codage des entiers positifs (binaire pur)
  - Codage des entiers relatifs (complément à 2)
  - Codage des nombres réels (virgule flottante)
- Arithmétique des codage
- Codage des caractères :
  - ASCII et
  - ASCII étendu ,
  - Unicode, …
- Codage du son et des images

FORMAT EN VIRGULE FLOTTANTE

- C’est une représentation dans la base 10 de **1,234**

123,400	$\times 10^{-2}$
12,3400	$\times 10^{-1}$
1,23400	$\times 10^{0}$
0,123400	$\times 10^{1}$
…	

ELÉMENT DU FORMAT EN VIRGULE FLOTTANTE

$$\overbrace{-}0\underbrace{,}\underbrace{9876}\times\overbrace{10}^{}\underbrace{^{-3}}$$

Signe de la mantisse

Base de système du nombre!

Exposant et signe de l'exposant

Mantisse

Position du point décimal

CODAGE DES NOMBRES RÉELS

Le codage des nombres réels se fait suivant :

- Format virgule flottante (utilisé actuellement sur machine)
  - défini par : $\pm M . b^E$

    un signe + ou –

    une mantisse m (en virgule fixe)

    un exposant e (un entier relative)

    une base b (2,8,10,16,…)

  - Exemple : $0,5425 . 10^2{}_{(10)}$; $10,1 . 2^{-1}{}_{(2)}$; $A0,B4.16^{-2}{}_{(16)}$

REPRÉSENTATION DE L'EXPOSANT ET DE SON SIGNE(1)

- L'exposant est translatée (biaisé) de manière <u>à toujours coder en interne une valeur positive</u>

<u>Exemple</u> :Avec 2 digits réservés au codage de l'exposant

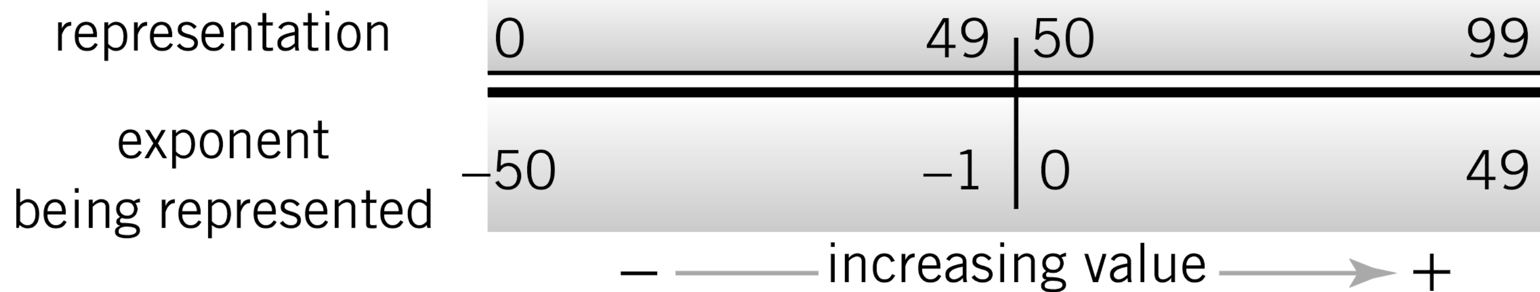
- Les valeurs positives: [+0, +99]

- En appliquant une translation k=50:
  - Les exposants représentables => [-50,49]

- La constante k est appelée le biais

- Avec 2 digits réservés au codage de l'exposant avec un biais égal à 50_{10} et 5 digits pour la mantisse on peut représenter
- de $.00001 \times 10^{-50}$ à $.99999 \times 10^{49}$

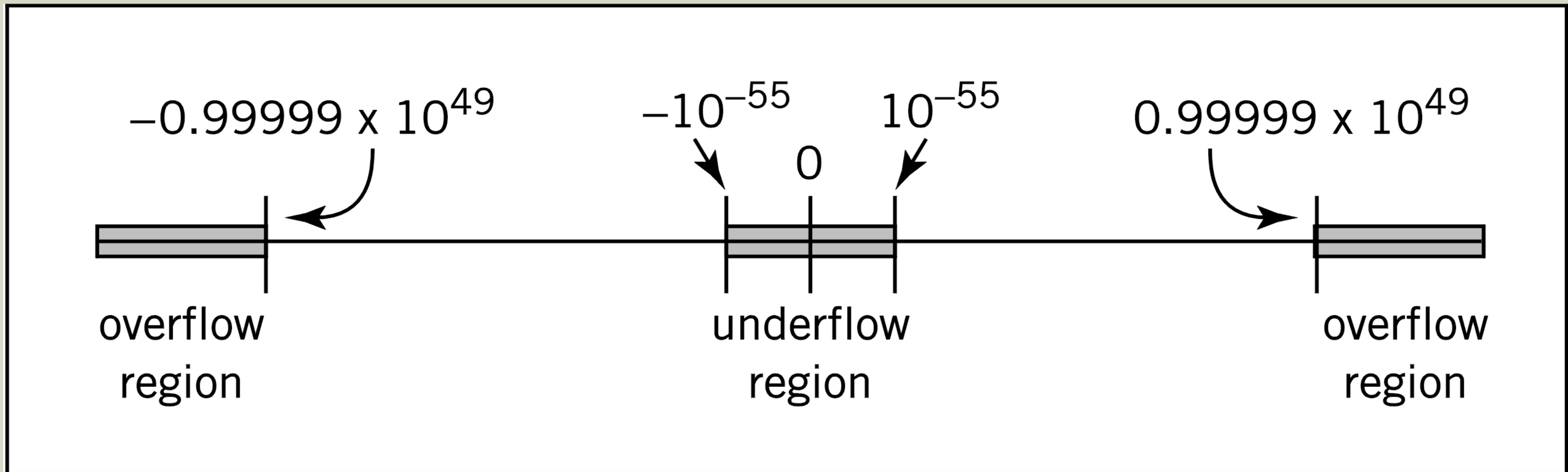
REPRÉSENTATION DE L'EXPOSANT ET DE SON SIGNE (2)



Englander: The Architecture of Computer Hardware and Systems Software, 2nd edition
Chapter 5, Figure 05-01

OVERFLOWS / UNDERFLOWS

- De .00001 x 10^{-50} à .99999 x 10^{49}
- De 1 x 10^{-55} à .99999 x 10^{49}



Englander: The Architecture of Computer Hardware and Systems Software, 2nd edition Chapter 5, Figure 05-02

CODAGE EN VIRGULE FLOTTANTE EN BASE 2

$$x = \pm M \,.\, 2^{E}$$

où M est la mantisse (virgule fixe) et E l'exposant (signé).

- Le codage en base 2, format virgule flottante, revient à coder le signe, la mantisse et l'exposant.

- Exemple : Codage en base 2, format virgule flottante, de:

$$3{,}25_{(10)} = 11{,}01_{(2)} \text{ (en virgule fixe)}$$
$$= 1{,}101 \,.\, 2^{1}_{(2)}$$
$$= 110{,}1 \,.\, 2^{-1}_{(2)}$$

- Pb : différentes manières de représenter E et M

Pour le codage il faut Normaliser

CODAGE EN VIRGULE FLOTTANTE -NORMALISATION

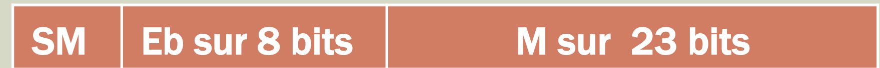
$$x = \pm\, 1{,}M \cdot 2^{Eb}$$

- Le signe est codé sur 1 bit ayant le poids fort :
  - le signe – : bit 1
  - Le signe + : bit 0
- Exposant biaisé (Eb)
  - placé avant la mantisse pour simplifier la comparaison
  - Codé sur p bits et biaisé pour être positif (ajout de $2^{p-1}-1$)
- Mantisse normalisé(M)
  - Normalisé : virgule est placé après le bit à 1 ayant le poids fort (i.e après le premier bit significatif –non nul-)
  - M est codé sur q bits **(avec 1 bit caché)**
- Exemple : N= 11,01, en normalisant N= 1,101 donc M =101

SM	Eb sur p bits	M sur q bits

STANDARD IEEE 754 (1985)

- Simple précision sur 32 bits :
  1 bit de signe de la mantisse
  8 bits pour l’exposant
  23 bits pour la mantisse



- Double précision sur 64 bits :
  1 bit de signe de la mantisse
  11 bits pour l’exposant
  52 bits pour la mantisse

STANDARD IEEE 754 (1985)

		Signe	Exposant	Mantisse	Valeur
Simple précision	32 bits	1 bit	8 bits $1 \leq e \leq 254$	23 bits	$(-1)^s \times 1.m \times 2^{e-127}$
Double précision	64 bits	1 bit	11 bits $1 \leq e \leq 2046$	52 bits	$(-1)^s \times 1.m \times 2^{e-1023}$
Précision étendue	80 bits	1 bit	15 bits $1 \leq e \leq 32766$	64 bits	$(-1)^s \times 1.m \times 2^{e-16383}$

CONVERSION DÉCIMALE – IEEE 754 (CODAGE D'UN RÉEL)

$35{,}5_{(10)} = ?_{(\text{IEEE 754 simple précision})}$

- Nombre positif, donc SM = 0
- $35{,}5_{(10)} = 100011{,}1_{(2)}$ (virgule fixe)
- $35{,}5_{(10)} = 1{,}000111 \; . \; 2^{5}{}_{(2)}$ (virgule flottante)
- Exposant = Eb-127 = 5, donc Eb = 132
- 1,M = 1,000111 donc M = 00011100…

$$\overbrace{0}^{SM}\;\underbrace{10000100}_{Eb}\;\overbrace{00011100000000000000000}^{M}{}_{(IEEE754SP)}$$

CONVERSION IEEE 754 - DÉCIMALE (EVALUATION D'UN RÉEL)

$$\overbrace{0}^{SM}\ \underbrace{10000001}_{Eb}\overbrace{11100000000000000000000}^{M}{}_{(IEEE754SP)}$$

- S = 0, donc nombre positif
- Eb = 129,, donc exposant = Eb-127 = 2
- 1,M = 1,111
- + $1,111 \cdot 2^2{}_{(2)} = 111,1_{(2)} = 7,5_{(10)}$

IEEE 754 : REPRÉSENTATION DU ZÉRO ET CAS PARTICULIER SIMPLE PRÉCISION

Simple précision:

Signe	Exposant	Mantisse
1 bit	8 bits	23 bits

Attention: pour ne pas représenter que des entiers très grands on soustrait le biais = -127

Les valeurs codées sont :

$2^{(exposant-127)}$ ×	1,	mantisse

8 bits pour l'exposant

→ on a 256 valeurs possibles [0,255]

→ Exposant biaisé [-127,128]

→ Interval couvert $[\cong -10^{-38}, \cong 10^{38}]$

Exercice : Examinons sur un l'axe des réels la forme normalisées de la norme

REPRÉSENTATION DU ZÉRO, DES INFINIS, REPRÉSENTATIONS DÉNORMALISÉES (1)

La norme définit des codages spéciaux pour représenter :

les nombres dénormalisés

les valeurs spéciales infinis, 0 et NaN.

elle définit aussi :

les opérations arithmétiques usuelles (+,-,×,/,√)

les arrondis à effectuer pour ces opérations

les gestion des exceptions.

REPRÉSENTATION DU ZÉRO, DES INFINIS, REPRÉSENTATIONS DÉNORMALISÉES (2)

- Si le nombre de bits consacrés à l'exposant est k, la valeur de l'exposant e vérifie $0 < e < 2^k-1$.
- Les valeurs 0 et 2^k-1 sont réservées pour des valeurs spéciales comme les arrondis : vers 0, vers $+\infty$, vers $-\infty$.

Signe	Exposant	Mantisse	Valeur	Commentaire
0	0	0	0	unique représentation de 0
s	0	$m \neq 0$	$(-1)^s \times 0.m \times 2^{-126}$	nombres dénormalisés (trop petit)
0	255	0	$+\infty$	résultat de 1/0
1	255	0	$-\infty$	résultat de -1/0
0	255	$m \neq 0$	NaN	Not a Number : résultat de 0/0 ou $\sqrt{-1}$

CONTINUITÉ DES NOMBRES EN SIMPLE PRÉCISION

Type	Exposant	Mantisse	Valeur approchée	Écart / préc
Zéro	0000 0000	000 0000 0000 0000 0000 0000	0,0	
Plus petit **nombre dénormalisé**	0000 0000	000 0000 0000 0000 0000 0001	$1,4 \times 10^{-45}$	$1,4 \times 10^{-45}$
Nombre dénormalisé suivant	0000 0000	000 0000 0000 0000 0000 0010	$2,8 \times 10^{-45}$	$1,4 \times 10^{-45}$
Nombre dénormalisé suivant	0000 0000	000 0000 0000 0000 0000 0011	$4,2 \times 10^{-45}$	$1,4 \times 10^{-45}$
...	...	...	...	...
Plus grand nombre dénormalisé	0000 0000	111 1111 1111 1111 1111 1111	$1,175\,494\,21 \times 10^{-38}$	$1,4 \times 10^{-45}$
Plus petit **nombre normalisé**	0000 0001	000 0000 0000 0000 0000 0000	$1,175\,494\,35 \times 10^{-38}$	$1,4 \times 10^{-45}$
Nombre normalisé suivant	0000 0001	000 0000 0000 0000 0000 0001	$1,175\,494\,49 \times 10^{-38}$	$1,4 \times 10^{-45}$

CALCUL EN VIRGULE FLOTTANTE: ADDITION

- Pour additionner, deux nombres en virgule flottante:
  1. Nombres doivent être alignés : avoir les mêmes exposants (le plus élevé pour protéger la précision)
  2. Additionner mantisses. Si overflow, ajuster l'exposant

Exemple: Avec 2 digits réservés au codage de l'exposant avec un excentrement égal à 50_{10} et 5 digits pour la mantisse.

0 51 99718 (e = 1) et 0 49 67000 (e = -1)

- Aligner les nombres: 0 51 99718
  0 51 00670

- Additionner: 99718
  + 00670
  1 00388 (Overflow)

- Arrondir le nombre et ajuster l'exposant: 0 52 10039

CALCUL EN VIRGULE FLOTTANTE: MULTIPLICATION

- $(a \times 10^{e})\ (b \times 10^{f}) = a \times b \times 10^{e+f}$
- Règle: multiplier les mantisses; additionner les exposants
- Besoin de soustraire constante du biais n à partir du résultat car:

  $(b + e) + (b + f) = 2 \times b + e + f$

Exemple:

0 51 99718 (e = 1) et 0 49 67000 (e = -1)

- Mantisses: .99718 * .67000 = 0.6681106
- Exposants: 51 + 49 = 100 et 100 – 50 = 50
- Normaliser par rapport à la machine:

  .6681106 devient .66811
- Résultat: $.66811 * 10^{0}$ (50 signifie e = 0)

CODAGE DES CARACTÈRES

- Les caractères englobent : les lettres alphabétiques (A, a, B , B,..) , les chiffres , et les autres symboles (> , ; / : ….) .
- Le codage le plus utilisé est le **ASCII** (American Standard Code for Information Interchange)
- Dans ce codage chaque caractère est représenté sur **8 bits** .
- Avec 8 bits on peut avoir 2^8 = 256 combinaisons
- Chaque combinaison représente un caractère
  - Exemple :
    - Le code 65 $(01000001)_2$correspond au caractère **A**
    - Le code 97 (01100001) correspond au caractère **a**
    - Le code 58 (00111010)correspond au caractère **:**
- Actuellement il existe un autre code sur 16 bits , se code s'appel **UNICODE** .

Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex
[NULL]	32	20	[SPACE]	64	40	@	96	60
[START OF HEADING]	33	21	!	65	41	A	97	61
[START OF TEXT]	34	22	"	66	42	B	98	62
[END OF TEXT]	35	23	#	67	43	C	99	63
[END OF TRANSMISSION]	36	24	$	68	44	D	100	64
[ENQUIRY]	37	25	%	69	45	E	101	65
[ACKNOWLEDGE]	38	26	&	70	46	F	102	66
[BELL]	39	27	'	71	47	G	103	67
[BACKSPACE]	40	28	(	72	48	H	104	68
[HORIZONTAL TAB]	41	29	)	73	49	I	105	69
[LINE FEED]	42	2A	*	74	4A	J	106	6A
[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B
[FORM FEED]	44	2C	,	76	4C	L	108	6C
[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D
[SHIFT OUT]	46	2E	.	78	4E	N	110	6E
[SHIFT IN]	47	2F	/	79	4F	O	111	6F
[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70
[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71
[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72
[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73
[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74
[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75
[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76
[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77
[CANCEL]	56	38	8	88	58	X	120	78
[END OF MEDIUM]	57	39	9	89	59	Y	121	79
[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A
[ESCAPE]	59	3B	;	91	5B	[	123	7B
[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C
[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D
[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E
[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F

CHAPITRE 3: ALGÈBRE DE BOOLE

- Les variables booléennes
- Les opérateurs logiques
- La table de vérité
- Règles de simplification
- Le tableau de Karnaugh
- Spécificités des portes composées

C'EST QUOI?

- Quelque chose que tout programmeur, même débutant, utilise sans le savoir, dans les conditions.
- Très utile en algorithmique, pour la recherche de performances au niveau des conditions,
- En base de donnée, pour les requêtes,
- En électronique, pour limiter le nombre de câblage de portes logiques.
- Utile pour vous, peu importe votre spécialité plus tard !

- **Histoire**

  George Boole a défini vers 1850 une algèbre qui s'applique à des

  fonctions logiques de variables logiques

  - variables booléennes (deux valeurs : 1 (Vrai) et 0 (Faux))

VARIABLES BOOLEENNES (LOGIQUES)

- L'algèbre de Boole est une structure algébrique qui ne contient que deux éléments: des variables booléennes.
- Les variables booléennes ne peuvent avoir que deux états, 1 ou 0 (true ou false dans certains langages de programmation).
- Leur règles de calcul seront détaillées plus loin.
- Exemple:

  Un test exécuté dans une condition est une variable booléenne.
  SI (A=B) ALORS ...

  Le test (A=B) peut avoir deux valeurs : 1 si A est réellement égal à B et 0 sinon.

  On dit que c'est la variable booléenne associée au test A=B.

LES OPÉRATEURS LOGIQUES

- L'algèbre de Boole utilise plusieurs opérateurs nommés: opérateurs booléens, opérateurs logiques, ou encore fonctions logiques ou portes logiques (terme plus propre à l'électronique)

- **L'opérateur OU symbolisé** dans les conditions par un OR ou || En algèbre de Boole, par un +. Les matheux préfèrent le symbole ∨.

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

LES OPÉRATEURS LOGIQUES

- **L'opérateur ET :** symbolisé dans les conditions par un AND ou &&.
  En **algèbre de Boole,** il est symbolisée par un point .
  En maths on écrira plutôt ∧

a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

D’autres opérateurs :

- L'opérateur NON symbolisé par une barre au dessus de(s) variable(s), En maths on utilise le symbole ¬
- **L'opérateur XOR (OU exclusif)** ⊕
- **L'opérateur NXOR** $a \oplus b$
- L'opérateur NAND $\overline{ab}$
- L'opérateur NOR $\overline{a+b}$

THÉORÈMES FONDAMENTAUX DE L'ALGÈBRE DE BOOLE

Attention : dans l'algèbre de Boole 1 + 1 = 1

Fermeture	Si A et B sont des variables booléennes, alors $A+B$, AB sont aussi des variables booléennes
Commutativité	$A+B=B+A$ $A\bullet B=B\bullet A$
Associativité	$A+(B+C)=(A+B)+C$ $A\bullet(B\bullet C)=(A\bullet B)\bullet C$
Distributivité	ET / OU $A(B+C)=AB+AC$ OU / ET $A+(BC)=(A+B)(A+C)$
Idempotence	$A+A=A$ $A\bullet A=A$
Complémentarité	$A+\overline{A}=1$ $A\bullet\overline{A}=0$
Identités remarquables	$1+A=1$ $1\bullet A=A$ $0+A=A$ $0\bullet A=0$
Distributivité interne	$A+(B+C)=(A+B)+(A+C)$ $A\bullet(B\bullet C)=(A\bullet B)\bullet(A\bullet C)$

THÉORÈME DE DE MORGAN

- $\overline{A+B} = \overline{A}.\overline{B}$

Vérification :

A	B	$\overline{A+B}$	$\overline{A}.\overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Equivalent

- $\overline{A.B} = \overline{A}+\overline{B}$

Vérification :

A	B	$\overline{A.B}$	$\overline{A}+\overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Equivalent

FONCTIONS LOGIQUES

- Résultat de la combinaison (logique combinatoire) d'une ou plusieurs variables logiques reliées entre elles par des opérations logique de base .

Proposition (intéressante)
Toute fonction logique peut être réalisée à l'aide d'un petit nombre de fonctions logiques de bases : les opérateurs logiques (ou portes logiques).

FORMES NORMALES

Théorème

Chaque fonction logique a une représentation sous forme normale conjonctive (resp. sous forme normale disjonctive)

FND : grand OU de petits ET = Somme de produits

FNC : grand ET de petits OU = Produit de sommes

Pour mettre une formule sous forme de somme de produits :

1. on fait rentrer les non le plus au cœur de la formule, pour avoir uniquement des négations de variable (lois de Morgan).
2. on utilise la distributivité de + par rapport au produit (respectivement FNC)

Remarque : la formule obtenue à partir d'une table de vérité est une somme de produits (=forme normale disjonctive).

Exemple: Donnez la FNC et la FND de

$$\overline{a+(\bar{b}c)}\times b$$

FONCTIONS BOOLÉENNES (EXEMPLE)

- A toute fonction logique est associée une table de vérité.
  - f_0 :
  - f_1 :
  - f_2 :
  - f_3 :

a	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

- A une variable: 4 fonctions distinctes
  - f_0 :
  - f_1 :
  - f_2 :
  - f_3 :
  - f_4 :

a	b	f_0	f_1	f_2	f_3	f_4
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

- A deux variables: 16 fonctions distinctes (2^{2^n})

COMBIEN DE FONCTIONS LOGIQUES ?

Théorème
Il existe 2^{2^n} fonctions logiques de n variables booléennes

À toute fonction logique est associée une table de vérité.
À toute table de vérité est associée une fonction logique.
une table de vérité à n variables possède 2^n lignes.
Pour chaque ligne la fonction peut prendre 2 valeurs.
Il y a donc 2^{2^n} fonctions

Beaucoup de fonctions logiques ...
... mais on peut tout représenter avec quelques fonctions (c'est ce qu'on va voir)

Remarque : toutes les fonctions logiques (élémentaires) (ce qu'on a appelé opérateurs logiques) décrites plus tôt peuvent s'appliquer à n variables

TOUT FAIRE AVEC OU ET NON

Théorème
Toute fonction logique est représentable à l'aide des fonctions OU et NON.

Preuve : Toute fonction a une table de vérité, et donc :

1. la fonction est vraie si on est sur une des lignes où elle vaut un.
2. si on est sur la première ligne où f vaut 1 OU la deuxième où vaut 1 OU la troisième etc.
3. être sur une ligne c'est dire avoir des valeurs fixes pour les variables : par exemple,

   a vaut 1 ET b vaut 0 ET c vaut 0
4. cela nous donne une formule avec des OU, NON et ET
5. D'après le loi de Morgan, ET exprimable à partir de OU et NON

   Cas particulier : si toutes les lignes sont à 0 alors f = 0 = $\overline{a+\bar{a}}$

   Un exemple :

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

$$f = \overline{a+b} + \overline{a+\bar{b}} + \overline{\bar{a}+\bar{b}}$$

TOUT FAIRE AVEC UNE SEULE PORTE

- On sait tout faire avec OU et NON.
- Il suffit de trouver une porte qui sait représenter les fonctions OU et NON

Théorème

L'opérateurs NAND (resp. NOR) est complet (i.e. il permet de représenter à lui seul tous les opérateurs logiques).

Preuve :

$$NAND(a,a) = \overline{a} = NON(a)$$

D ' après Morgan, $\overline{\overline{a}\,\overline{b}} = a + b$

d'où NAND(NAND(a, a),NAND(b, b)) = a + b

DES PORTES AUX CIRCUITS

SYNTHÈSE DE CIRCUIT

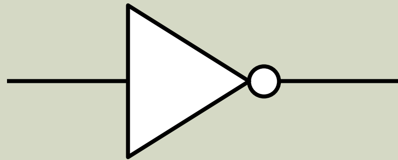
Proposition

Toute fonction logique peut être réalisée à partir de portes logiques, en formant un circuit.

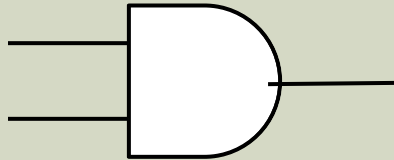
Logique	Circuit
Fonctions logique	Circuits logique
Opérateurs logique	Portes logique
Variables (paramètres)	Fils d'entrée
Résultat (s)	Fil (s) de sortie

PORTES LOGIQUES DE BASE

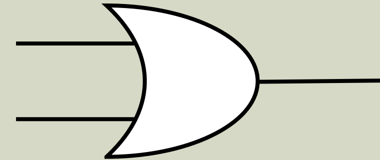
Les portes logiques sont réalisées avec des **transistors** **(exemple slide suivant)**



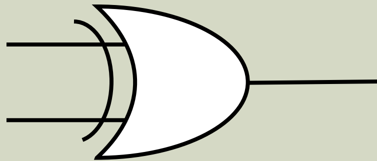
NON



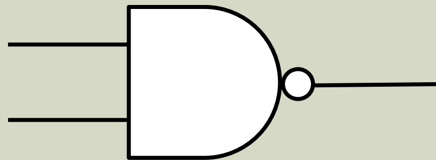
AND



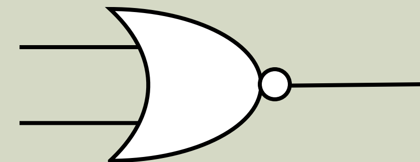
OR



XOR

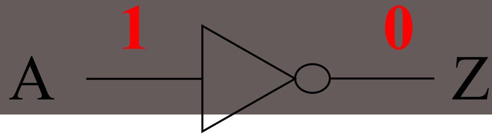


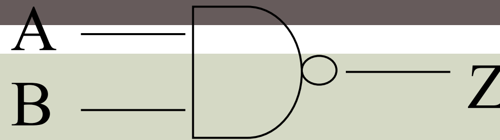
NAND

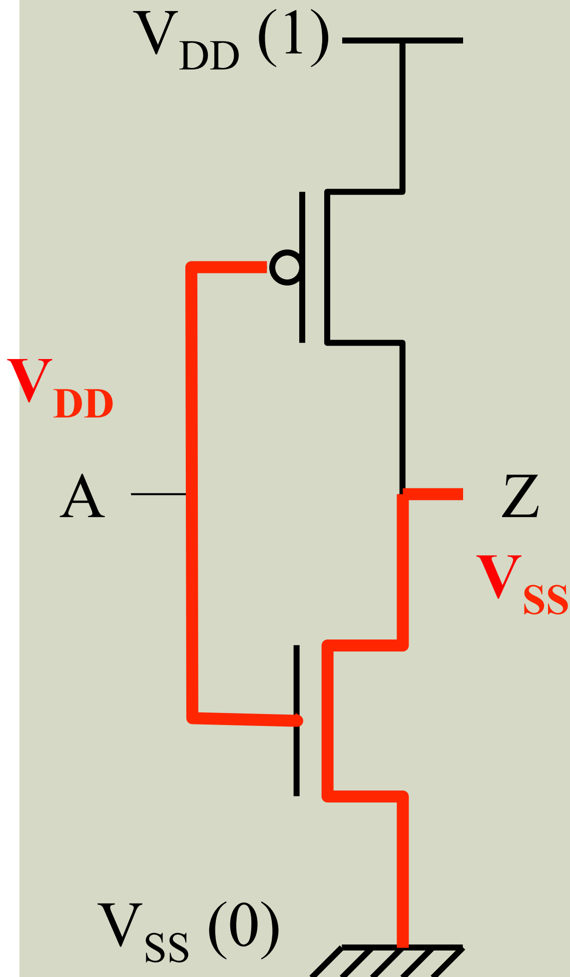


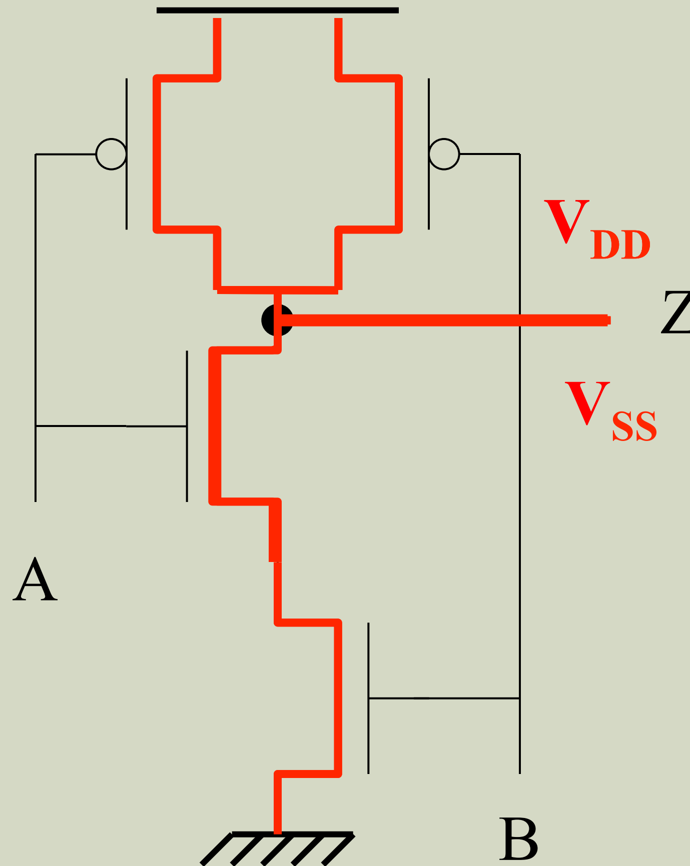
NOR

DES PORTES SIMPLES AVEC DES TRANSISTORS







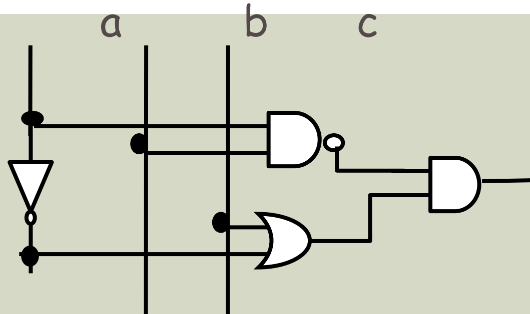


A	B	Z
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

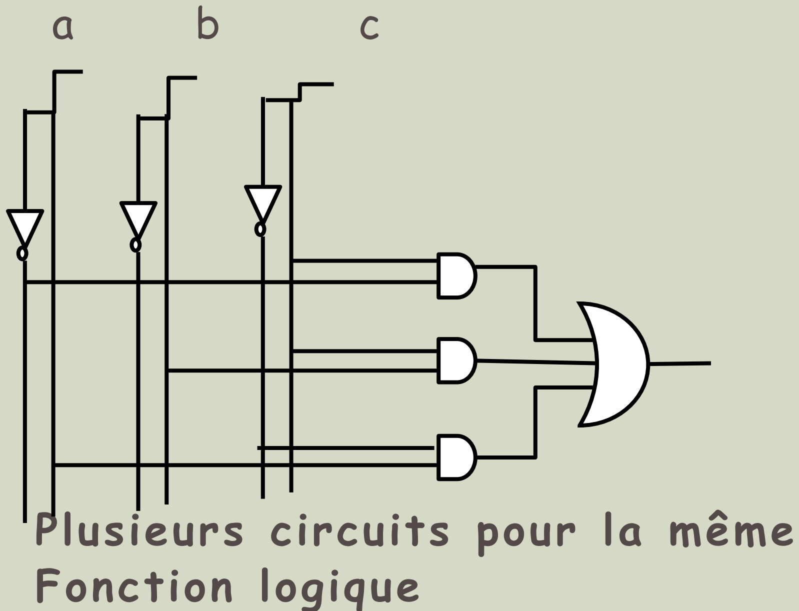
NON-ET

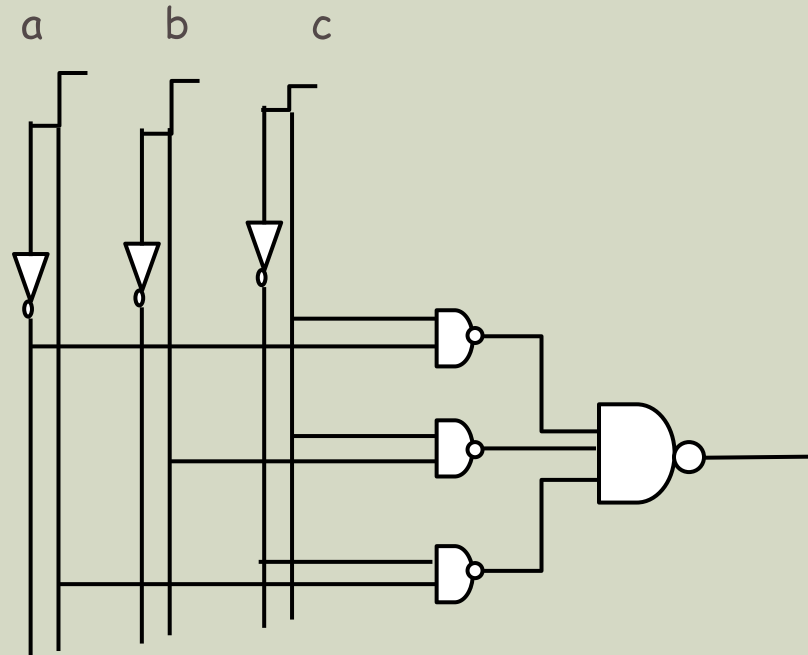
EXEMPLE DE CIRCUIT

- Un fil porte une valeur binaire (0 ou 1)
- gros points = points de contact (soudure)
- Quelle est la fonction correspondante au circuit suivant?



- Analysons ces deux circuits?





Plusieurs circuits pour la même Fonction logique

CRÉER UN CIRCUIT - LA MÉTHODE

Comment créer un circuit pour implanter une fonction :

1. Construire la table de vérité
2. En dériver une somme de produits brute
3. **Simplifier l'expression en une autre, équivalente, mais plus simple**
   (ex. simplification algébrique ou tables de Karnaugh).
4. réaliser la fonction logique à l'aide de portes

Le circuit doit être le moins coûteux possible en terme fils et portes utilisées

SIMPLIFICATION ALGÉBRIQUE

Principe
Appliquer les théorèmes fondamentaux de l'algèbre de Boole afin d'obtenir une expression plus compacte.

(dés)Avantages de la simplification algébrique:

Cette simplification permet de :

- obtenir des formules simples...
- ... si on est malin ...
- ... et on a de la chance ! ! !

mais :

- On peut compliquer en voulant simplifier
- On ne peut plus directement construire le circuit final avec des NAND
- Ce n'est pas automatisable, du moins pas facilement.

EXEMPLE

$$S = A \cdot B \cdot C + A \cdot \overline{B} \cdot \left(\overline{\overline{A} \cdot \overline{C}}\right)$$

- <u>Transformation</u>

$$S = A \cdot B \cdot C + A \cdot \overline{B} \cdot (A + C)$$

$$= A \cdot B \cdot C + A \cdot \overline{B} \cdot A + A \cdot \overline{B} \cdot C$$

$$= A \cdot B \cdot C + A \cdot \overline{B} + A \cdot \overline{B} \cdot C$$

- <u>Variables communes</u>

$$S = A \cdot \overline{B} + A \cdot C \cdot \left(B + \overline{B}\right)$$

$$S = A \cdot \overline{B} + A \cdot C$$

$$\boxed{S = A \cdot \left(\overline{B} + C\right)}$$

TABLES DE KARNAUGH (1)

But : simplifier les fonctions logiques graphiquement

Comment ?

1. on prend une fonction booléenne contenant jusqu'à 4 (à 6) variables
2. on réalise une sorte de table de vérité en deux dimensions, On met des 1 dans les cases quand la fonction est vraie
3. En observant attentivement la table, on regroupe les cases ayant des 1 par "blocs".
4. en utilisant ce regroupement on écrit une fonction plus simple

<u>Code de Gray</u>

1. énumérer toutes les combinaisons à n bits
2. partir de la combinaison avec tous les bits à 0
3. d'un nombre à l'autre un seul bit change
4. à chaque fois changer le bit le plus à droite possible conduisant à une nouveau nombre

Avec un bit   0 1

Avec deux bits   00 ...

Avec trois bits   000 ...

Cela marche avec autant de bits qu'on veut.

TABLEAUX DE KARNAUGH (2)

- Un outil intuitif pour la simplification de fonctions ayant un faible nombre de variables.
- Disposition différente de la table de vérité en suivant le code de Gray.
- Réunir deux «1» adjacents ⇔ éliminer une variable et un terme.
- **Graphiquement: réaliser un recouvrement de tous les 1**
- **Fonction = OU de tous les paquets de «1»**

A	B	Z
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

Table de vérité

B\A	0	1
0	**1**	**1**
1	**1**	**0**

Tableau de Karnaugh

B\A	0	1
0	**1**	**1**
1	**1**	**0**

A’B’+AB’+A’B

=A’B’+AB’

+A’B’+A’B

=B’+A’

CONSTRUCTION DES BLOCS DE 1: LES RÈGLES.

- On construits des blocs dans la table en respectant toutes ces règles :

1. On ne met que des cases avec des 1 dans les blocs.
2. On ne fait que des blocs à 2^n cases.
3. On ne fait que des blocs rectangulaires.
4. On essaie de faire des blocs maximaux.
5. Il faut que chaque case contenant un 1 soit contenue au moins dans un bloc.
6. à la fin on prend le minimum de blocs pour recouvrir tous les 1 de la table.

TABLEAUX DE KARNAUGH À 3 ET 4 VARIABLES

- Deux cases adjacentes ne doivent différer que par une seule variable (code de Gray)
- Heuristique pour le recouvrement.

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

C'

A

B

Z=A+B+C'

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

Z=A'C+B'D'+AB'C'

CAS DES VALEURS NON ASSIGNÉES

- Valeur de sortie non définie.
- Notation = *X* (*don't care*).
- Recouvrement si simplification.

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

Z=A'C+B'D'+**AC'**

GAIN DE LA SIMPLIFICATION

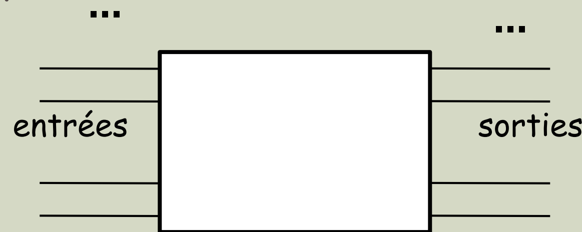
Pour calculer le gain on note :

- le nombre de portes
- leur type
- leur nombre d'entrées

et cela avant simplification et après simplification.

CIRCUITS COMPLEXES

- Pour fabriquer des circuits réalisant des fonctions complexes :
  on ne se contente pas d'assembler les portes logiques séparément,
  mais des circuits intégrés logiques :
    plusieurs portes représentées par une seule boîte (= un composant) muni de broches permettant d'assurer les connections électriques.



- Familles de circuits intégrés
- ... selon la densité d'intégration,
- .. c'est à dire le nombre de portes par circuit :
  - SSI (Small Scale Integration) 1 à 10 portes/circuit,
  - MSI (Medium Scale Integration) 10 à 100 portes/circuit,
  - LSI (Large Scale Integration) 100 à 100000 portes/circuit,
  - VLSI (Very Large Scale Integration) plus de 100000 portes/circuit,