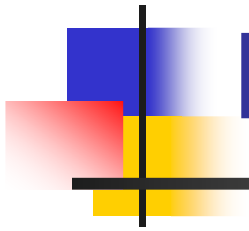
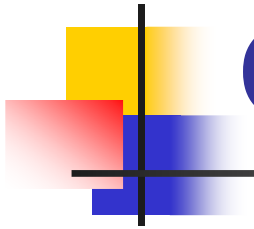


Représentation de données



Codage binaire

- Ordinateur
  - Systèmes à deux états : transistors
- Informations de plusieurs types
  - texte, nombre, image, etc..
  - traitée comme suite de 0 et de 1
  - unité d'information est le **bit** (binary digit)
- Codage de l'information
  - D'une représentation de données vers une autre
  - Représentation externe et interne

Calcul de bases

- La base habituelle est la base 10
- En base b, on utilise b chiffres

$X = a_n a_{n-1} \ldots a_1 a_0$

$b = 10 : a_i \in \{0,1,2,3,4,5,6,7,8,9\}$

$b = 2 \quad : a_i \in \{0,1\}$

$b = 16 : a_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

Les nombres entiers

- En base 10:

  $2007 = 2*10^3 + 0*10^2 + 0*10^1 + 7*10^0$

- En base b, soit la suite : $a_n a_{n-1} \ldots a_1 a_0$

$$a_n a_{n-1} \ldots a_1 a_0 = \sum_{i=0}^{n} a_i b^i$$

Exemple : $(101)_2 = 1*2^2 + 0*2^1 + 1*2^0 = 5$

Les nombres réels

- $12,34 = 1*10^{1} + 2*10^{0} + 3*10^{-1} + 4*10^{-2}$

- $a_n a_{n-1} \ldots a_1 a_0, a_{-1} a_{-2} \ldots a_{-p} =$
  $a_n * b^n + a_{n-1} * b^{n-1} + \ldots + a_0 * b^0 + a_{-1} * b^{-1} + \ldots + a_{-p} * b^{-p}$

- Exemple : $(11,01)_2 =$
  $1*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2} = 3,25$

De la base 10 vers une base b? Nombre entier

- Méthode
  - On **divise** le nombre par la base b, puis le quotient obtenu de nouveau par la base et ainsi de suite jusqu'à l'obtention d'un quotient nul.
  - La *suite des restes* $a_n a_{n-1} \ldots a_1 a_0$ correspond au nombre représenté en base b.

Exemple : $44_{10} = (?)_2$

- De la base 10 à la base 2
  - 44 = 22*2 + 0
    22 = 11*2 + 0
    11 = 5*2 + 1
    5 = 2*2 + 1
    2 = 1*2 + 0
    1 = 0*2 + 1
  - $(44)_{10}=(101100)_2$

De la base 10 vers une base b? Nombre réel

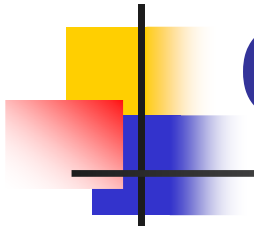
- Méthode
  - On **multiplie** la partie fractionnaire par la base en répétant l'opération sur la partie fractionnaire du produit jusqu'à ce qu'elle soit nulle.
  - La suite des parties entières $a_{-1}a_{-2}$ ...a_{-p} correspond aux chiffres dans la base b.

Exemple : $(54,125)_{10} = (?)_2$

- Partie entière
  - $(54)_{10} = (110110)_2$
- Partie fractionnaire
  - 0,125 * 2 = 0,25
  - 0,25 * 2 = 0,50
  - 0,50 * 2 = 1,00
- $(54,125)_{10} = (110110,001)_2$

Les bases 2, 8 et 16

- Base 2: notation binaire.
- Base 8: notation octale.
- Base 16: notation hexadécimale.
- Un chiffre octal représente 3 bits.
- Un chiffre hexadécimal représente 4 bits.
- Un octet = 8 bits
  - $(10011011)_2 = (010\ 011\ 011)_2 = (233)_8$
  - $(10011011)_2 = (1001\ 1011)_2 = (9B)_{16}$

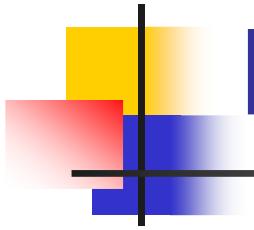


Opérations arithmétiques

- Les opérations s'effectuent dans une base quelconque b en utilisant les mêmes méthodes qu'en base 10.
- Une retenue se produit quand on atteint ou dépasse b.
- Exemples

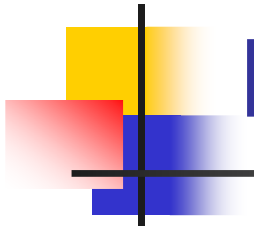
$(110)_2 + (11)_2 = (1001)_2$

$(110)_2 * (11)_2 = (10010)_2$



Représentation des

- Caractères
- Nombres entiers
- Nombres réels



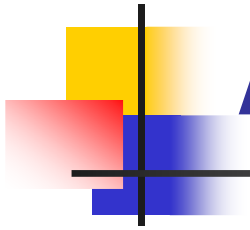
Représentation des caractères

- Symboles
  - Lettres majuscules, minuscules
  - Ponctuations (& . ~ , ; # ...)
  - ☺ ☯ ☠ ☎
- Code ASCII (American Standard Code for Information Interchange - 1961)
  - Un symbole codé sur 7 bits (0…127)

Code ASCII

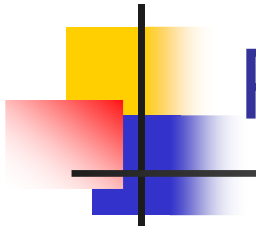
- 0 - 31: caractères de contrôle (retour à la ligne, tabulation, bip sonore, …)
- 32 - 127: caractères de ponctuation, chiffres, caractères alphabétiques,…
  - 65 - 90 : les lettres majuscules
  - 97 - 122 : les lettres minuscules
  - Passage de majuscules en minuscules : ajouter 32 au code (forcer le 6ième bit à 1)

0 NUL	32 espace	64 @	96 `
1 SOH	33 !	65 A	97 a
2 STX	34 "	66 B	98 b
3 ETX	35 #	67 C	99 c
4 EOT	36 $	68 D	100 d
5 ENQ	37 %	69 E	101 e
6 ACK	38 &	70 F	102 f
7 BEL	39 '	71 G	103 g
8 BS	40 (	72 H	104 h
9 HT	41)	73 I	105 i
10 LF	42 *	74 J	106 j
11 VT	43 +	75 K	107 k
12 FF	44 ,	76 L	108 l
13 CR	45 -	77 M	109 m
14 SO	46 .	78 N	110 n
15 SI	47 /	79 O	111 o
16 SLE	48 0	80 P	112 p
17 CS1	49 1	81 Q	113 q
18 DC2	50 2	82 R	114 r
19 DC3	51 3	83 S	115 s
20 DC4	52 4	84 T	116 t
21 NAK	53 5	85 U	117 u
22 SYN	54 6	86 V	118 v
23 ETB	55 7	87 W	119 w
24 CAN	56 8	88 X	120 x
25 EM	57 9	89 Y	121 y
26 SIB	58 :	90 Z	122 z
27 ESC	59 ;	91 [	123 {
28 FS	60 <	92 \	124 |
29 GS	61 =	93]	125 }
30 RS	62 >	94 ^	126 ~
31 US	63 ?	95	127 ■



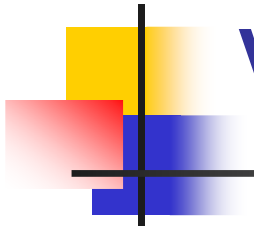
Autres codages

- ASCII étendu
  - 8 bits (0…255): OEM, ANSI, ISO
  - N’est pas unique et dépend de la plateforme
- UNICODE (1991)
  - 16 bits (0…65535)
  - 49194 caractères, 25 alphabets



Représentation des nombres entiers

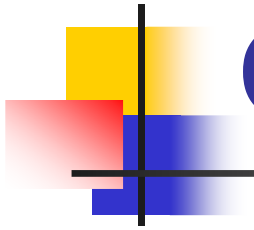
- Problème 1 : Limitation de la taille de codage
  - un nombre fixe d'octets (1, 2, 4)
  - n bits : [0 .. 2^n-1]
  - 1 octet = 8 bits : [0 .. 2^8-1]
- Problème 2: Entiers négatifs
  - Valeur signée
  - Complément à 1
  - Complément à 2



Valeur signée

- Le bit le plus à gauche pour représenter le signe
  - Codage de 6 sur 4 bits : $(6)_{10} = (0110)_2$
  - Codage de -6 sur 4 bits : $(-6)_{10} = (1110)_2$

- Problèmes :
  - Il y a 2 zéros : 00..0 et 10..0
  - Incompatible avec l'addition usuelle



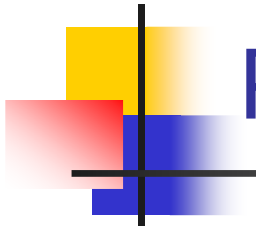
Complément à 1

- Pour représenter un nombre négatif, on inverse tous les bits : 1 devient 0 et 0 devient 1
  - Codage de 6 sur 4 bits : $(6)_{10} = (0110)_2$
  - Codage de -6 sur 4 bits : $(-6)_{10} = (1001)_2$
- Avantages
  - Compatible avec l'addition usuelle: n + (-n) = 1111
- Problème
  - Il y a toujours 2 zéros : 00..0 et 11..1

Complément à 2

- Pour représenter un nombre négatif, inverser tous les bits puis ajouter 1 au résultat obtenu
  - Codage de 6 sur 4 bits : $(6)_{10} = (0110)_2$
  - Codage de -6 sur 4 bits :

    $(-6)_{10} = (1001)_2 + 1 = (1010)_2$
- Avantages
  - Compatible avec l'addition usuelle
  - Il y a un seul zéro : 00..0
- n bits codent l'intervalle $[-2^{n-1}, 2^{n-1}-1]$



Représentation des nombres réels

- Représentation en virgule flottante : $m * b^e$

  m : mantisse, b : base, e : exposant

- Exemples
  - $6,15 = 61,5*10^{-1} = 0,615*10^{1} = \ldots$ (base 10)
  - $-0,002 = -0,02*10^{-1} = \ldots$ (base 10)
  - $1011 = 1,011*10^{11} = \ldots$ (base 2)

Norme IEEE 754 (1985)

	Nombre de bits	Signe	Exposant	Mantisse
Simple précision	32 bits	1 bit	8 bits	23 bits
Double précision	64 bits	1 bit	11 bits	52 bits

Norme IEEE 754 (suite)

- Signe (S) :
  - 0 : nombre positif ou zéro
  - 1 : nombre négatif
- Exposant (E) est codé avec un excès de
  - 127 $(01111111)_2$ en simple précision
  - 1023 $(01111111111)_2$ en double précision
- Mantisse (M) normalisée :
  - $0 \leq M < 1$
  - unité 1 n'est pas codée explicitement

$$\mathbf{(-1)^S * (1+M) * 2^{E\text{-excès}}}$$

Exemple 1 : format IEEE => nombre décimal

1 10000000 01001000000000000000000

S E (8 bits) M (23 bits)

$= -\mathbf{1},01001 * 10^{10000000-01111111}$

$= -(1 + 2^{-2} + 2^{-5}) * 2^{128-127}$

$= -(2 + 1/2 + 1/16)$

$= -2,562510$

Exercice:

0 10000010 11110000000000000000000 = ?

Exemple 1 : format IEEE => nombre décimal

0 10000010 11110000000000000000000

S E (8 bits) M (23 bits)

$= \mathbf{1},1111 * 10^{10000010-01111111}$

$= (1 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}) * 2^{3}$

$= (1 + 1/2 + 1/4 + 1/8 + 1/16) * 8$

$= 15,5$

Exemple 2 : nombre décimal => format IEEE

$11{,}625_{10} = 1011{,}101_2$

$= 1{,}011101 * 10^{11}$

(*normalisation en binaire*)

S = 0

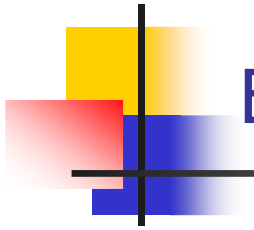
E = $11_2 + 01111111_2 = 10000010_2$

$(= 3_{10} + 127_{10} = 130_{10})$

M = $01110100000000000000000_2$

0 10000010 01110100000000000000000

Exercice: $9{,}75_{10}$ = ?



Exemple 3 : nombre décimal => format IEEE

$9{,}75_{10} = 1001{,}11_2$

$= 1{,}00111 * 10^{11}$

(*normalisation en binaire*)

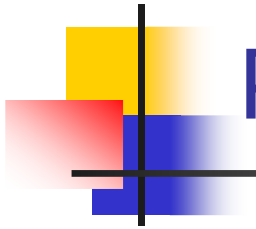
S = 0

$E = 11_2 + 01111111_2 = 10000010_2$

$(= 3_{10} + 127_{10} = 130_{10})$

$M = 00111000000000000000000_2$

0 10000010 00111000000000000000000



Perte de précision possible

$(1,6)_{10}$ n'est pas précisément représentable en virgule flottante.

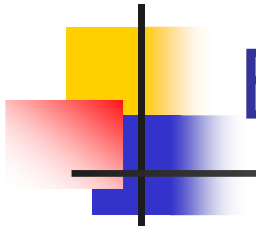
Exercice: $1,6_{10}$ = ?

Exceptions (en simple précision)

- Exposant 00000000_2 est réservé:
  - Flottants plus petits que 2^{-127} (dénormalisé)
    - 0 00000000 (mantisse ≠ 0)
    - 1 00000000 (mantisse ≠ 0)
  - Exemple

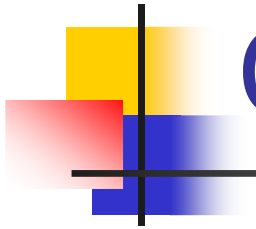
    0 00000000 1010..0 =

    0,1010..0 * 2^{0-127} = (2^{-1} + 2^{-3}) * 2^{-127}
  - En particulier: 0 00000000 0000..0 = 0.



Exceptions (en simple précision)

- Exposant 11111111_2 (255_{10}) est réservé:
  - Nombre excessif ou infinité (overflow)
    - 0 11111111 00000…000 = +INF
    - 1 11111111 00000…000 = -INF
    - Par exemple: x/0 = INF, INF+x = INF, …
  - Arithmétique invalide (not a number)
    - 0 11111111 (mantisse ≠ 0) = NaN
    - 1 11111111 (mantisse ≠ 0) = NaN
    - Par exemple: 0/0, INF/INF, racine d'un négatif, …



Opérations avec les flottants

Addition/Soustraction $1{,}01*10^{11} + 1{,}1*10^{100}$

Opérations avec les flottants

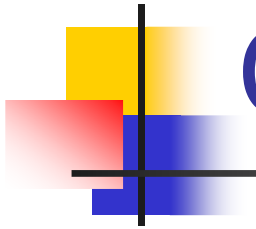
Addition/Soustraction $1{,}01*10^{11} + 1{,}1*10^{100}$

1. Dénormaliser (chercher le même exposant)

   $1{,}1*10^{100} = 11*10^{11}$
2. Additionner/soustraire les mantisses

   $1{,}01 + 11 = 100{,}01$
3. Normaliser le résultat

   $100{,}01*10^{11} = 1{,}0001*10^{101}$



Opérations avec les flottants

Multiplication/Division $1{,}11*10^{-10} * 1{,}01*10^{111}$

Opérations avec les flottants

Multiplication/Division $1{,}11*10^{-10} * 1{,}01*10^{111}$

1. Additionner/soustraire les exposants

   $-10 + 111 = 101$

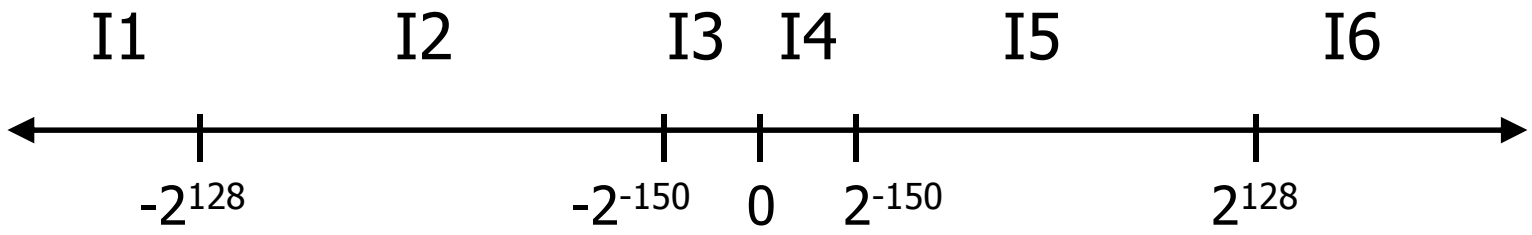
2. Multiplier/diviser les mantisses

   $1{,}11 * 1{,}01 = 10{,}0011$

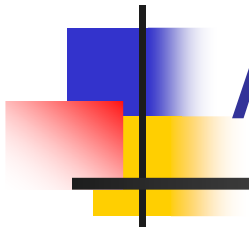
3. Normaliser le résultat

   $10{,}0011*10^{101} = 1{,}00011*10^{110}$

Les intervalles représentables



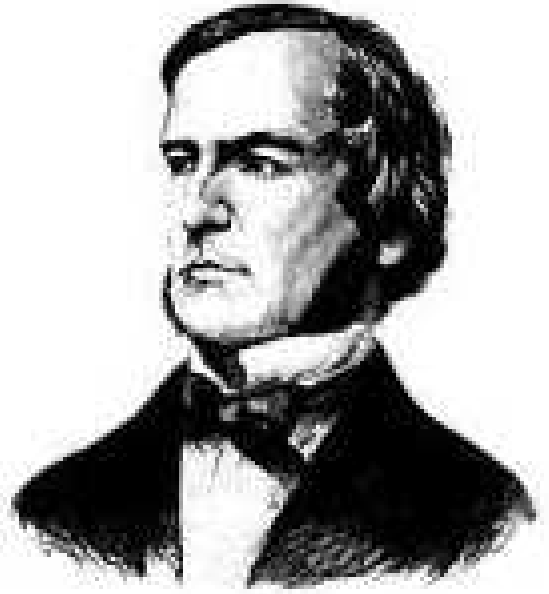
- I1 et I6 : dépassement de capacité (*overflow*)
- I2 et I5 : représentable mais avec éventuellement une perte de précision
- I3 et I4 : ne pourra être représenté et sera approximé par zéro (*underflow*)



Algèbre de Boole

Définitions

- Algèbre binaire
- Variables booléennes : ne prennent que deux valeurs **VRAI** ou **FAUX**.
- Opérateurs décrits par une **table de vérité** ou une **porte logique**



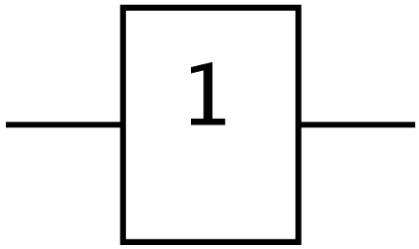
George BOOLE
(1815-1864)

Opérateur : IDENTITY

Table de vérité

X	S
0	0
1	1

Symbole



Équation

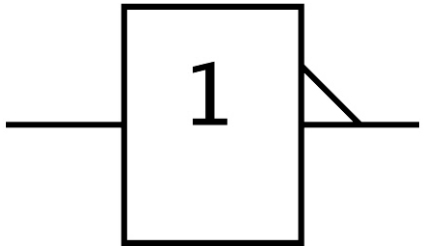
$S = X$

Opérateur : NOT

Table de vérité

X	S
0	1
1	0

Symbole



Équation

$S = \neg X = \overline{X}$

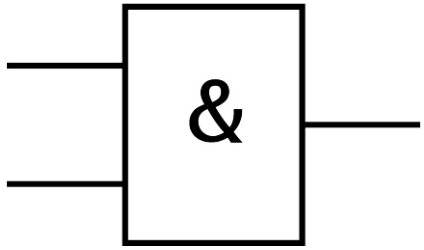
Remarque : La barre oblique est utilisée dans tous les symboles pour représenter la fonction de négation

Opérateur : AND

Table de vérité

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

Symbole



Équation

$S = AB = A.B = A*B$

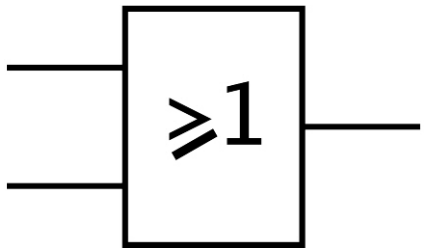
$= A \cap B = A \wedge B$

Opérateur : OR

Table de vérité

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

Symbole



Équation

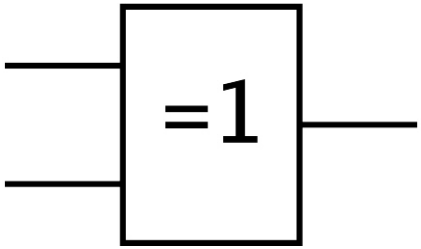
$S = A+B = A \cup B = A \vee B$

Opérateur : XOR (OU exclusif)

Table de vérité

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

Symbole



Équation

$S = A \oplus B$

Propriétés

Lois	ET	OU	Lois	ET	OU
Identité	1.A = A	0+A = A	**Nullité**	0.A = 0	1+A = 1
Associativité	(A.B).C = A.(B.C)	(A+B)+C = A+(B+C)	**Commutativité**	A.B = B.A	A+B = B+A
Distributivité	A.(B+C) = A.B + A.C		**Idempotence**	A.A = A	A+A = A
Inversion	$A.\overline{A}=0$	$A+\overline{A}=1$	**Absorption (1)**	A.(A+B) = A	A+A.B = A
Absorption (2)	$A+\overline{A}B=A+B$		**Loi de De Morgan**	$\overline{A.B}=\overline{A}+\overline{B}$	$\overline{A+B}=\overline{A}.\overline{B}$

Problème logique

- Plusieurs variables
- Expressions possibles
  - Français
  - Table de vérité
  - Équation
  - Circuit logique
- Exemple

  Fonction « Majorité »:

  F(A,B,C) = 1 ⇔ La majorité des variables = 1

Table de vérité

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table de vérité => équation?

- Forme disjonctive

- F =

  (¬A.B.C) + (A.¬B.C)

  + (A.B.¬C) + (A.B.C)

Table de vérité

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table de vérité => équation?

- Forme conjonctive

- F =

  (A+B+C).(A+B+¬C).
  (A+¬B+C).(¬A+B+C)

Table de vérité

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table de vérité => équation?

- Simplifier (par ex. la forme disjonctive)

F = ¬A.B.C + A .¬B.C+ A.B.¬C + A.B.C

= ¬A.B.C + A .¬B.C+ A.B.¬C + A.B.C + A.B.C + A.B.C

= ¬A.B.C + A.B.C + A .¬B.C + A.B.C + A.B.¬C + A.B.C

= (¬A+A).B.C + (¬B+B).A.C +(¬C+C).A.B

= A.B + A.C + B.C