

# Architecture des ordinateurs

Licence Informatique - Université de Provence

**Peter Niebert**

[peter.niebert@lif.univ-mrs.fr](mailto:peter.niebert@lif.univ-mrs.fr)



# A propos du cours

- 10 heures de Cours, 10 heures de TD, 10 heures de TP
- TD et TP commencent la semaine prochaine (8 octobre)
- Site du Cours

<http://www.cmi.univ-mrs.fr/~niebert/archi09>

En TP :

- TkGate, un logiciel de conception de circuit
- Assembleur MIPS (SPIM, ...)

# Evaluation

- 2 interrogations écrites en Cours (environ une heure) = CC
- 1 projet en TP = TP
- 1 examen final en janvier = E

**Note UE (première session) :** 
$$\frac{3 * \max(E, (2E + CC)/3) + TP}{4}$$

## **Seconde session :**

On remplace  $E$  par  $E'$ , la note d'examen de seconde session

# Evaluation : Projet

- Le projet se réalise au plus en binome
- Il faut rendre un rapport en plus des sources
- La présence à la soutenance est indispensable
- Les 2 dernières séances de TP sont consacrées au projet

Les projets ne sont pas optionnels  
Absence à la soutenance = DEF à l'UE = pas de L3

# Plan du cours (peut changer en cours de route)

- 1 Codage de l'information
- 2 Algèbre de Boole - Fonctions booléennes - Circuits combinatoires
- 3 Circuits séquentiels
- 4 Mémoires
- 5 Machines de Moore - Machines de Mealy - Machines synchrones - Microprogrammation
- 6 Programmation d'un processeur - Assembleur
- 7 Fonctionnement d'un processeur MIPS
- 8 Exemples d'autres processeurs - Bus
- 9 Optimisation : Pipeline - Mémoire cache

# Bibliographie

- A. Tanenbaum. “*Architecture de l’ordinateur*” (4ième édition) InterEdition.
- J.-J. Schwarz “*Architecture des ordinateurs*” Eyrolles
- W. Stallings “*Organisation et architecture de l’ordinateur*”, Pearson Education
- D. Patterson J. Hennessy “*Organisation et conception des ordinateurs*” Dunod.
- ..

# Objectifs du cours (I)

## Fonctionnement d'un ordinateur

Applications
Systemes d'exploitation
<b>Matériel</b>

gcc, emacs, ...

Linux/Unix, Windows, ...

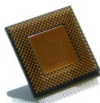
PC, Mac, ...

# Objectifs du cours (II)



On va s'intéresser à ce qu'on trouve sur la carte mère  
et principalement

- au processeur



- à la mémoire



# Systemes de numération

# La base 10 (I)

10 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

$$145 = 1 * 10^2 + 4 * 10^1 + 5 * 10^0$$

$$1 = 1$$

$$14 = 1 * 10 + 4$$

$$145 = (1 * 10 + 4) * 10 + 5$$

Pour bien préciser qu'il s'agit d'un nombre en base 10

$$(145)_{10}$$

# La base 10 (II)

## Partie "fractionnaire"

$$0,329 = 3 * 10^{-1} + 2 * 10^{-2} + 9 * 10^{-3}$$

$$0,9 = 9 * 10^{-1}$$

$$0,29 = ((9 * 10^{-1}) + 2) * 10^{-1}$$

$$0,329 = (((9 * 10^{-1}) + 2) * 10^{-1} + 9) * 10^{-1}$$

# Autres bases

Quelques bases  $B$  utiles :

- $B = 2$  binaire
  - ▶ chiffres : 0,1
- $B = 8$  octal
  - ▶ chiffres : 0,1,2,3,4,5,6,7
- $B = 16$  hexadécimal
  - ▶ chiffres : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

# Nombres représentés

Nombre représenté en base  $B$  :

$$\underbrace{d_n d_{n-1} \dots d_2 d_1 d_0}_{\text{partie entière}}, \underbrace{d_{-1} d_{-2} \dots d_{-m}}_{\text{partie fractionnaire}}$$

- par position :

$$\sum_{i=-m}^n d_i * B^i$$

- par multiplications successives :

$$\begin{aligned} &((( (d_n * B + d_{n-1}) * B + \dots + d_2) * B + d_1) * B + d_0) \\ &+ \\ &((( (d_{-m} * \frac{1}{B} + d_{-m+1}) * \frac{1}{B} + \dots + d_{-2}) * \frac{1}{B} + d_{-1}) * \frac{1}{B} \end{aligned}$$

## Exemple

Nombre représenté en base 4 :

$$(301,23)_4$$

- par position :

$$3 * 4^2 + 0 * 4^1 + 1 * 4^0 + 2 * 4^{-1} + 3 * 4^{-2} = (49,6875)_{10}$$

- par multiplications/divisions successives :

$$\underbrace{((3 * 4 + 0) * 4 + 1)}_{49} + \underbrace{\underbrace{((3 * 1/4) + 2) * 1/4}_{0.75}}_{0,6875}$$

## D'une base à une autre

- de la base  $B$  vers la base 10 : le calcul du nombre représenté donne un algorithme
- de la base 10 vers la base  $B$  : divisions et multiplications successives

$$(35, 25)_{10} \rightarrow (100011, 01)_2$$

Partie entière

Partie fractionnaire

$$35 / 2 = 17 \text{ reste } 1$$

$$17 / 2 = 8 \text{ reste } 1$$

$$8 / 2 = 4 \text{ reste } 0$$

$$4 / 2 = 2 \text{ reste } 0$$

$$2 / 2 = 1 \text{ reste } 0$$

$$1 / 2 = 0 \text{ reste } 1$$

$$0,25 * 2 = 0,5$$

$$0,5 * 2 = 1,0$$

## D'une base à une autre : pourquoi ça marche ?

$(d_n d_{n-1} \dots d_2 d_1 d_0)_B$  vaut

$$((( (d_n * B + d_{n-1}) * B + \dots + d_2) * B + d_1) * B + d_0)$$

On a

$$\frac{((( (d_n * B + d_{n-1}) * B + \dots + d_2) * B + d_1) * B + d_0)}{B}$$

égal à

$$((( (d_n * B + d_{n-1}) * B + \dots + d_2) * B + d_1) \text{ reste } d_0)$$

car  $0 \leq d_0 < B$ .      On continue récursivement ....

de même pour la partie fractionnaire ...

## Base 2/ base 16

- De la base 2 vers la base 16 :
  - ▶ on regroupe les chiffres par 4 en partant de la virgule
  - ▶ chaque groupe de 4 bits représente un chiffre hexadécimal

101110000011

*B*      8      3

- De la base 16 vers la base 2 : on convertit chaque chiffre hexadécimal en un nombre binaire de 4 chiffres équivalents.

# Codage de l'information

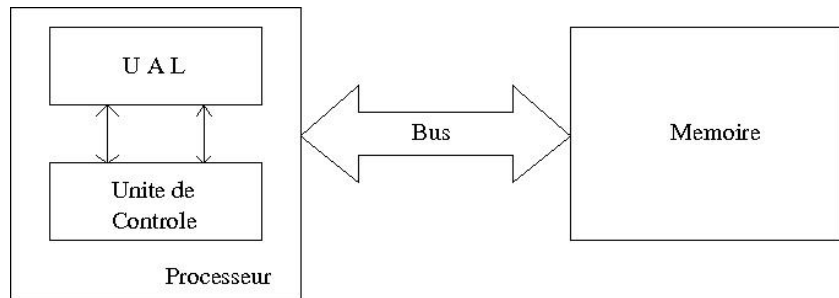
# Qu'est ce qu'un ordinateur ?

## Modèle de von Neumann :

modèle proposé par John von Neumann (1903-1957) en 1944 qui définit un ordinateur en 3 parties :

- **le processeur** composé
  - ▶ d'une **ALU**, unité arithmétique et logique (opérations sur les données)
  - ▶ d'une **unité de contrôle** (traitement des instructions à exécuter)
- **la mémoire** (stockant à la fois les données et les instruction à exécuter)
- **le bus** assurant la liaison entre la mémoire et le processeur

# Modèle de von Neumann



# Instruction/données

- Différence entre instruction et données en mémoire ?

Il n'y en a pas ; c'est juste de l'**information**.

Le **bit** (**b**inary **d**igit) est la plus petite quantité d'information ; il prend 2 valeurs, 0 ou 1.

- Dans un ordinateur, tout n'est que 0 et 1
  - ▶ 1 (du courant)
  - ▶ 0 (pas de courant)
- Souvent l'information est stockée sur des (suites de) mots binaires d'une taille fixée (par le processeur, la mémoire, ...) :  
ex. 8 bits, 16, bits, ...

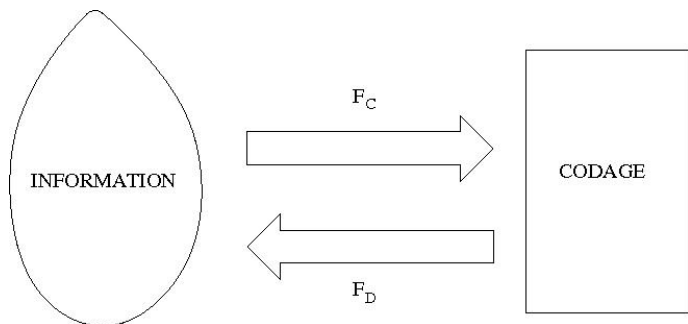
# Codage de l'information

Le **codage de l'information** est la représentation d'informations diverses (nombres entiers, nombres flottants, caractères, chaînes de caractères, images, ....) sous forme binaire, c'est-à-dire sous la forme d'une suites de bits.

Les points importants sont :

- Le passage de l'information à son codage
- Le passage du codage à l'information

# Codage de l'information



En général,  $F_D = F_C^{-1}$

Mais pas toujours ....

# Caractères ASCII

L'**ASCII** (American Standard Code for Information Interchange), créé en 1961, associe un nombre à chaque caractère. Les caractères sont codés sur 7 bits. Il y a donc 128 caractères différents (de 00 à 7F en hexadécimal).

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

# Caractères ASCII étendu

Le code ASCII est anglophone : beaucoup de caractères présents dans les alphabets français, nordique, .... en sont absents. Il existe des extensions (sur 8 bits, de 80 à FF). Cependant, ces extensions ne sont pas standardisées.

128	Ç	144	É	160	á	176	☐	193	⊥	209	⌈	225	β	241	±
129	ù	145	æ	161	í	177	☐	194	⌊	210	⌋	226	Γ	242	≥
130	é	146	Æ	162	ó	178	☐	195	⌈	211	⌋	227	π	243	≤
131	â	147	ô	163	ú	179		196	—	212	⌋	228	Σ	244	∫
132	ä	148	ö	164	ñ	180	†	197	‡	213	⌈	229	σ	245	∫
133	à	149	ò	165	Ñ	181	‡	198	‡	214	⌈	230	μ	246	+
134	â	150	û	166	ª	182	‡	199	‡	215	‡	231	τ	247	±
135	ç	151	ù	167	º	183	⌈	200	⌋	216	‡	232	Φ	248	°
136	ê	152	—	168	¿	184	⌈	201	⌈	217	∫	233	⊙	249	·
137	ë	153	Ö	169	—	185	‡	202	⌋	218	⌈	234	Ω	250	·
138	è	154	Û	170	¬	186	‡	203	⌈	219	■	235	δ	251	√
139	ï	156	£	171	½	187	⌈	204	‡	220	■	236	∞	252	—
140	î	157	₣	172	¾	188	‡	205	=	221	■	237	φ	253	²
141	í	158	—	173	¡	189	‡	206	‡	222	■	238	ε	254	■
142	Ä	159	ƒ	174	«	190	∫	207	⊥	223	■	239	∩	255	
143	Å	192	ℒ	175	»	191	⌈	208	⌋	224	α	240	≡		

# Caractères ISO-8859-1

iso-8859-1										
+	0	1	2	3	4	5	6	7	8	9
160		í	í	£	¤	¥	ì	é	¨	ø
170	¸	«	¬	-	¸	-	°	±	²	³
180	ˆ	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

## Caractères sur 16 bits : Unicode

Unicode propose de coder les caractères sur 16 bits (65536 valeurs possibles -  $(FFFF)_{16}$ ).

Il permet de coder notamment les alphabets latins, cyrilliques, les caractères chinois, indiens, arabes.

Unicode code point	character	UTF-8 (hex.)	Unicode code point	character	UTF-8 (hex.)
U+0400	È	d0 80	U+3130	ᄀ	e3 84 b0
U+0401	É	d0 81	U+3131	ᄁ	e3 84 b1
U+0402	Ђ	d0 82	U+3132	ᄂ	e3 84 b2
U+0403	Ѓ	d0 83	U+3133	ᄃ	e3 84 b3
U+0404	Є	d0 84	U+3134	ᄄ	e3 84 b4
U+0405	Ѕ	d0 85	U+3135	ᄅ	e3 84 b5
U+0406	І	d0 86	U+3136	ᄆ	e3 84 b6
U+0407	Ї	d0 87	U+3137	ᄇ	e3 84 b7
U+0408	Ј	d0 88	U+3138	ᄈ	e3 84 b8
U+0409	Љ	d0 89	U+3139	ᄉ	e3 84 b9
U+040A	Њ	d0 8a	U+313A	ᄊ	e3 84 ba
U+040B	Ћ	d0 8b	U+313B	ᄋ	e3 84 bb
U+040C	Ќ	d0 8c	U+313C	ᄌ	e3 84 bc

## Entiers binaires positifs (non signés)

On utilise simplement la représentation du nombre en base 2

Sur  $n$  bits, on code les naturels (entiers non signés) de 0 à  $2^n - 1$ .

### Addition

$$\begin{array}{r} 1 \\ 0110 \\ + 0101 \\ \hline = 1011 \end{array}$$

Attention au débordement (sur un nombre fixé de bits) :

$$0110 + 1011 = 10001$$

# Entiers binaires positifs et négatifs (signés)

## Entiers signés

- représentation valeur absolue et signe
- représentation par excès
- représentation à complément 1
- représentation à complément 2

On s'intéresse aux nombres codés sur  $n$  bits ( $n$  fixé)

Rappel : avec  $n$  bits, on code  $2^n$  valeurs différentes

## Valeur absolue et signe

- on utilise le bit le plus à gauche pour représenter le signe :
- $0 = "+"$        $1 = "-"$
- sur 4 bits :

$$0100 \Leftrightarrow 4 \quad 1100 \Leftrightarrow -4$$

### Inconvénients :

- 2 représentations pour 0 : (sur 4 bits) 0000 et 1000
- la somme (binaire) est incorrect

$$\begin{array}{rcccc} & 0 & 1 & 0 & 0 & & 4 \\ + & 1 & 0 & 1 & 1 & + & -3 \\ \hline = & 1 & 1 & 1 & 1 & = & -7 \end{array}$$

⇒ l'addition est un opération plus complexe

# Représentation par excès

On code le nombre en décalant les valeurs d'un biais *biais* (les valeurs seront donc en excès)

La suite  $b_{n-1} \dots b_1 b_0$  représente le nombre  $(b_{n-1} \dots b_1 b_0)_2 - \textit{biais}$ .

Exemple : sur 4 bits avec  $\textit{biais} = 7$

$$1011 \Leftrightarrow 4 \quad 0011 \Leftrightarrow -4 \quad 0111 \Leftrightarrow 0$$

## Inconvénients :

- addition binaire ne fonctionne pas
- selon le biais, l'inversion peut être difficile

## Complément à 1 (I)

- Si  $i$  un nombre positif binaire, alors son bit le plus à gauche vaut 0. Pour calculer  $-i$ , on inverse tous les bits de  $i$  (le bit le plus à gauche étant alors 1)

- sur 4 bits :

$$0011 \Leftrightarrow 3 \quad 1100 \Leftrightarrow -3$$

- sur 8 bits :

$$00000011 \Leftrightarrow 3 \quad 11111100 \Leftrightarrow -3$$

# Complément à 1 (II)

## Addition

- On ajoute les deux nombres binaires et on ajoute la retenue éventuelle
- 

$$\begin{array}{r} \phantom{+} 0\ 0\ 0\ 1 \\ + 1\ 1\ 1\ 0 \\ \hline 0\ 1\ 1\ 1\ 1 \\ = 1\ 1\ 1\ 1 \end{array} \quad \begin{array}{r} \phantom{+} 1 \\ + -1 \\ \hline = -0 \end{array} \quad \begin{array}{r} \phantom{+} \phantom{0}\ 1\ 0\ 0 \\ + 1\ 1\ 0\ 0 \\ \hline 1\ 0\ 0\ 0\ 0 \\ \phantom{1}\phantom{0}\phantom{0}\phantom{0}\ 1 \\ = 0\ 0\ 0\ 1 \end{array} \quad \begin{array}{r} \phantom{+} 4 \\ + -3 \\ \hline = 1 \end{array}$$

## Inconvénients

- 2 représentations pour 0 : (sur 4 bits) 0000 et 1111

## Complément à 2 (I)

- Si  $i$  un nombre positif binaire, alors son bit le plus à gauche vaut 0. Pour calculer  $-i$ , on inverse tous les bits de  $i$  et on ajoute 1 (le bit le plus à gauche étant alors 1).
- sur 4 bits

$$0011 \Leftrightarrow 3 \quad 1101 \Leftrightarrow -3$$

- - ▶ permet de représenter sur  $n$  bits les entiers compris entre  $-2^{n-1}$  et  $2^{n-1} - 1$
  - ▶ la valeur en entier codé en complément à 2 de la suite de bits  $b_m b_{m-1} \dots b_1 b_0$  est

$$\left( \sum_{i=0}^{m-1} b_i 2^i \right) - b_m 2^m$$



## Dépassement de capacité

La somme de 2 nombres (de  $n$  bits) ne peut être codé sur  $n$  bits (la somme est trop grande ou trop petite)

- en complément à 2

$$\begin{array}{rcccc} & 0 & 1 & 1 & 1 & & 7 \\ + & 0 & 0 & 1 & 1 & + & 3 \\ \hline & 1 & 0 & 1 & 0 & = & -6 \end{array}$$

Le résultat est bien-sûr incorrect.

En complément à 2,

Un **dépassement de capacité** se produit lorsque les deux opérandes ont le même signe et le résultat a un signe différent des opérandes.

# Nombres à virgules

Attention ceci n'a rien à voir avec des réels !

- Nombres à virgule fixe
- Nombres à virgule flottante

# Nombres à virgule fixe

On utilise les nombres fractionnaires comme vus précédemment.

La position de la virgule est déterminée de manière fixe dans la représentation du nombre.

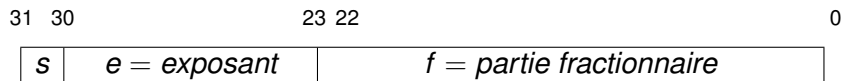
$$b_{n-1} \dots b_2 , b_1 b_0$$

Ici, 2 chiffres après la virgule.

## Inconvénients

- impossible de représenter à la fois des nombres très petits et des très grands.
- Imprécision dans les calculs pouvant être importante.

# Flottants IEEE 754



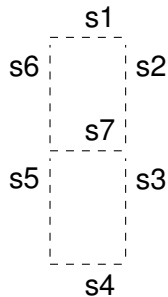
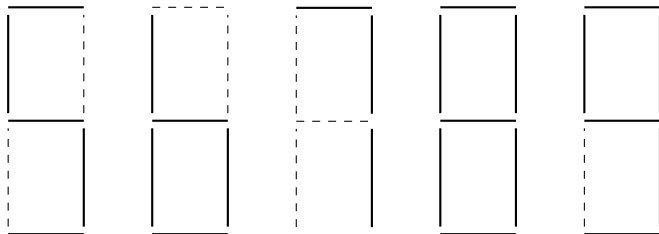
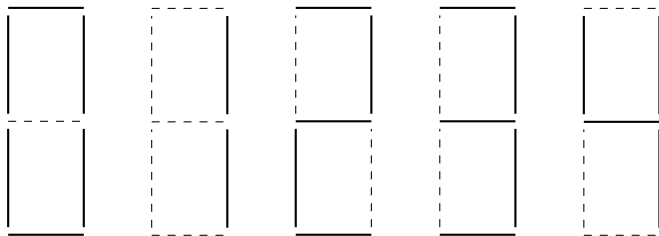
$$r = (-1)^s \cdot \left(1 + \sum_{i=1}^{23} f_i \cdot 2^{-i}\right) \cdot 2^{E-127}$$

$$\text{où } E = (e_7 \dots e_0)_2$$

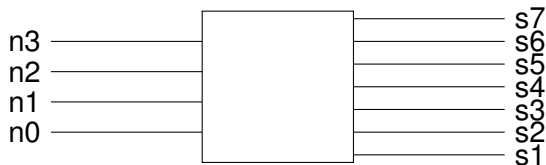
.... la suite en TD

# Algèbre de Boole - Fonctions Booléennes

# Contrôleur d'affichage 7 segments



## Contrôleur d'affichage 7 segments (II)



$$\mathcal{F} : \{0, 1\}^4 \rightarrow \{0, 1\}^7$$

## Contrôleur d'affichage 7 segments (III)

$\mathcal{F}$	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$	$s_6$	$s_7$
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	0	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1
1010	0	0	0	0	0	0	0
1011	0	0	0	0	0	0	0
1100	0	0	0	0	0	0	0
1101	0	0	0	0	0	0	0
1110	0	0	0	0	0	0	0
1111	0	0	0	0	0	0	0

# Algèbre de Boole

C'est une structure algébrique

- donnée par un ensemble à deux valeurs  $\{0, 1\}$  (ou  $\{\perp, \top\}$ ,  $\{\text{vrai}, \text{faux}\}$ ) et les trois opérations suivantes :
  - ▶ la conjonction (/2) :  $\cdot$  (“et” - “and”)
  - ▶ la disjonction (/2) :  $+$  (“ou” - “or”)
  - ▶ le complément (/1) :  $^-$  (“non” - “not”)
- satisfaisant les axiomes suivants
  - ▶ Commutativité

$$a.b = b.a \quad a + b = b + a$$

- ▶ Associativité

$$a + (b + c) = (a + b) + c \quad a.(b.c) = (a.b).c$$

# Algèbre de Boole : axiomes (suite)

- ▶ Distributivité

$$a.(b + c) = a.b + a.c \quad a + (b.c) = (a + b).(a + c)$$

- ▶ Éléments neutres :

$$1.a = a.1 = a \quad 0 + a = a + 0 = a$$

- ▶ Complément

$$a.\bar{a} = 0 \quad a + \bar{a} = 1$$

**Convention** : la conjonction est plus prioritaire que le disjonction.

# Algèbre de Boole : opérateurs

Le définition suivante des opérateurs satisfait l'ensemble des axiomes

- le complément

$a$	$\bar{a}$
0	1
1	0

- la conjonction

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

- la disjonction

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

# Algèbre de Boole : propriétés

Les propriétés suivantes peuvent être déduites des axiomes

- Élément absorbant :

$$a.0 = 0.a = 0 \quad a + 1 = 1 + a = 1$$

- Absorption

$$a.(a + b) = a \quad a + (a.b) = a$$

- Idempotence

$$a.a = a \quad a + a = a$$

- Involution

$$\overline{\overline{a}} = a$$

- Lois de De Morgan

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

## Fonctions booléennes

Une fonction booléenne (d'arité  $n$ )  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  peut être donnée

- de manière extensionnelle par sa **table de vérité**

$x$	$y$	$z$	$m = f(x, y, z)$
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

- par **une expression booléenne**, qui est une expression définie avec les constantes et les opérateurs de l'algèbre de Boole et un certain nombre de variables  $x_1, \dots, x_n$

$$\bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$$

# Fonctions booléennes : FND (I)

## Formes normales disjonctives

Une expression booléenne est en forme normale disjonctive si elle est écrit comme

- une disjonction de monômes
- chaque monôme étant une conjonction de littéraux
- un littéral étant soit les constantes 0, 1, soit une variable  $x$ , soit le complément de  $x$ ,  $\bar{x}$ .

Exemple :

$$(x.y.\bar{z}) + (\bar{x}.z) + y$$

## Fonctions booléennes : FND (II)

### théorème

*Toute expression booléenne est équivalente à une expression en forme normale disjonctive.*

### preuve

*on suppose les variables comme étant  $x_1, \dots, x_n$ ,*

- 1 On considère la table de vérité associé à l'expression et appelons  $f$  la fonction booléenne ainsi représentée*
- 2 supposons que  $f$  vaille 1 pour  $k$  entrées dans cette table (et donc 0 pour les  $2^n - k$  autres entrées). Comme  $0 + 1 = 1$  on peut écrire  $f$  comme  $f_1 + \dots + f_k$  où  $f_i : \{0, 1\}^n \rightarrow \{0, 1\}$  n'est à 1 que pour une seule entrée.*
- 3 Maintenant pour chaque  $f_i$ , il est simple de voir que le monôme  $y_1 \dots y_n$  défini par  $y_j = x_j$  si la  $i$ ème valeur de l'entrée est 1 et  $y_j = \bar{x}_j$  représente précisément la fonction  $f_i$ .*

## Fonctions booléennes : FND (III)

Une fonction booléenne  $f$  à trois paramètres  $x, y, z$  :

$x$	$y$	$z$	$f$	=	$f_1$	+	$f_2$	+	$f_3$	+	$f_4$
0	0	0	0		0		0		0		0
0	0	1	1		1		0		0		0
0	1	0	0		0		0		0		0
0	1	1	1		0		1		0		0
1	0	0	0		0		0		0		0
1	0	1	1		0		0		1		0
1	1	0	1		0		0		0		1
1	1	1	0		0		0		0		0

avec  $f_1 = \bar{x}.\bar{y}.z$      $f_2 = \bar{x}.y.z$      $f_3 = x.\bar{y}.z$      $f_4 = x.y.\bar{z}$

donc  $f(x, y, z) = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.z + x.y.\bar{z}$

# Fonctions booléennes

## corolaire

*Toute fonction booléenne peut être représentée comme une expression booléenne.*

## preuve

*Il suffit d'extraire l'expression en forme normale disjonctive de la table de vérité de la fonction booléenne.*

# Fonctions booléennes : FNC (I)

## Formes normales conjonctives (FNC)

Une expression booléenne est en forme normale conjonctive si elle écrit comme

- une conjonction de sommes
- chaque somme étant une disjonction de littéraux
- un littéral étant soit les constantes 0, 1, soit une variable  $x$ , soit le complément de  $x$ ,  $\bar{x}$ .

Exemple :

$$(x + y + \bar{z}).(\bar{x} + z).1$$

## Fonctions booléennes : FNC (II)

### théorème

*Toute expression booléenne est équivalente à une expression en forme normale conjonctive.*

On considère le produit d'une somme  $y_1 + \dots + y_n$  introduite pour chaque ligne de la table de vérité valant 0 avec  $y_i = x_j$  si la  $i$ ème valeur de l'entrée vaut 0 et  $y_j = \bar{x}_j$  sinon.

# Fonctions booléennes : simplifications

## Simplifications de fonctions booléennes

- utilisation des axiomes et des propriétés de l'algèbre de Boole.
- la méthode des tableaux de Karnaugh

## Tableaux de Karnaugh

Méthode de simplification basée sur la propriété

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

## Tableaux de Karnaugh (I)

- La méthode par les tableaux de Karnaugh est une méthode visuelle pour la simplification de formules logiques.
- Table de vérité  $\Rightarrow$  tableau bi-dimensionnel (séparation en deux l'ensemble des variables (une pour les lignes, un pour les colonnes))
- Lignes et colonnes sont indexées par toutes les valuations des variables correspondantes **tel que** entre deux lignes (resp. colonnes) **une seule valeur booléenne change**.

pour les variables  $x, y, z$

$xy$ $z$	00	01	11	10
0				
1				

## Tableaux de Karnaugh (II)

- On remplit ce tableau avec les valeurs de la fonction booléenne.

$\begin{matrix} xy \\ z \end{matrix}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Cette fonction booléenne a pour forme normale disjonctive

$$x.y.z + \bar{x}.y.z + x.\bar{y}.z + x.y.\bar{z}$$

## Tableaux de Karnaugh (III)

- Si deux cases adjacentes contiennent tous les deux un 1 alors les deux monômes correspondants dans la forme normale disjonctives ne diffèrent que sur une variable, alors une simplification est possible.

$\begin{matrix} xy \\ z \end{matrix}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$\begin{matrix} xy \\ z \end{matrix}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$\begin{matrix} xy \\ z \end{matrix}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

## Tableaux de Karnaugh (IV)

$\begin{matrix} xy \\ z \end{matrix}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$x.y.z + x.y.\bar{z} = x.y.(z + \bar{z}) = x.y$$

- On ne garde dans le monôme d'un regroupement que les variables dont la valeur ne change pas.

$\begin{matrix} xy \\ z \end{matrix}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$x.y + x.z$$

$\begin{matrix} xy \\ z \end{matrix}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$x.y + x.z + y.z$$

## Tableaux de Karnaugh (III)

- On ne garde dans le monôme d'un regroupement que les variables dont la valeur ne change pas.
- Tous les 1 de la table de vérité doivent être considérés : un 1 peut être utilisé dans plusieurs regroupement ; au pire un 1 isolé est un regroupement de taille un.
- Le tableau doit être considéré de façon circulaire (on peut replier le tableau comme une sphère).
- Les regroupements peuvent également être de taille 4,8, ... (toutes puissances de 2)

$\begin{matrix} xy \\ zt \end{matrix}$	00	01	11	10
00	0	1	1	0
10	1	0	0	0
11	0	0	0	0
01	0	1	1	0

$$y.\bar{z} + \bar{x}.\bar{y}.z.\bar{t}$$

# Circuits combinatoires

# Portes Logiques (I)

Porte ET



0	0	0
0	1	0
1	0	0
1	1	1

Porte OU



0	0	0
0	1	1
1	0	1
1	1	1

Porte NON



0	1
1	0

# Portes Logiques (II)

Porte NON-ET



0	0	1
0	1	1
1	0	1
1	1	0

Porte NON-OU



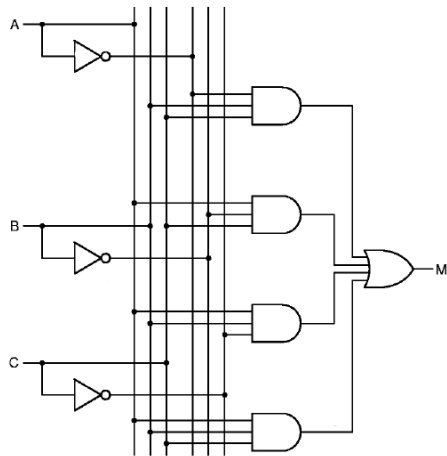
0	0	1
0	1	0
1	0	0
1	1	0

Porte OU-X



0	0	0
0	1	1
1	0	1
1	1	0

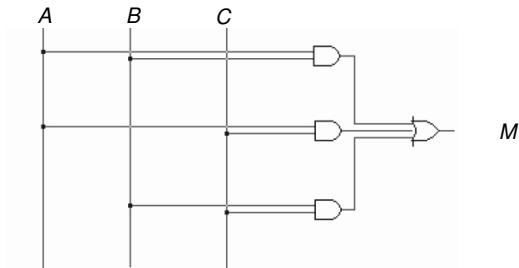
## Le circuit "Majorité"



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

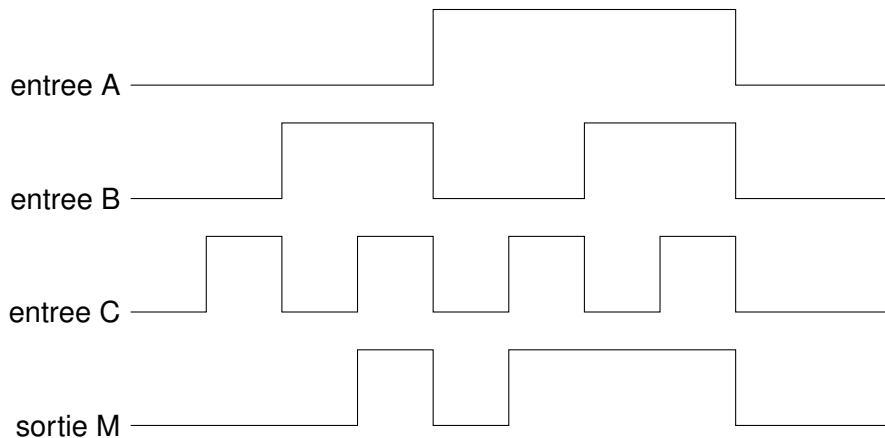
$$\bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + A.B.C$$

## Le circuit "Majorité" (II)



$$M = A.B + A.C + B.C$$

## Chronogramme du circuit "Majorité"



La sortie *M* ne dépend que des entrées *A*, *B* et *C*.

On parle alors de **circuit combinatoire**

# Circuits combinatoires

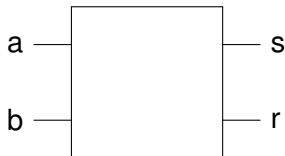
## un circuit combinatoire

- défini par un ensemble de portes reliées les unes aux autres.
- les sorties des portes sont reliés aux entrées d'autres portes (définissant une orientation des connexions)
- en suivant l'orientation des connections, il est impossible que partant de la sortie d'une porte, on revient à l'une des ses entrées (**graphe acyclique**)

Un circuit combinatoire peut être vu comme une porte logique (à plusieurs sorties).

# Demi-additionneur (I)

- entrées :  $a$  et  $b$
- sorties :  $s$  la somme et  $r$  la retenue



$a$	$b$	$s$	$r$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On parle de **demi-additionneur** (additionneur 1 bit) : ne permet pas d'être étendu en un additionneur  $n$  bits.

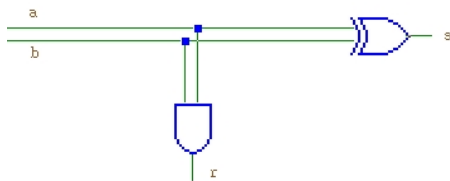
## Demi-additionneur (II)

<i>a</i>	<i>b</i>	<i>s</i>	<i>r</i>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s = a \oplus b$$

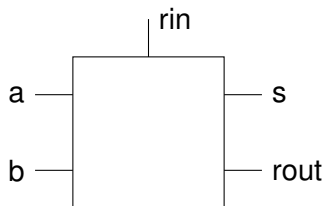
$$r = a.b$$

$\oplus$  est l'opérateur de ou-exclusif (XOR)



# Additionneur (I)

- entrées :  $a$ ,  $b$  et  $rin$  la retenue d'entrée
- sorties :  $s$  la somme et  $rou$  la retenue de sortie

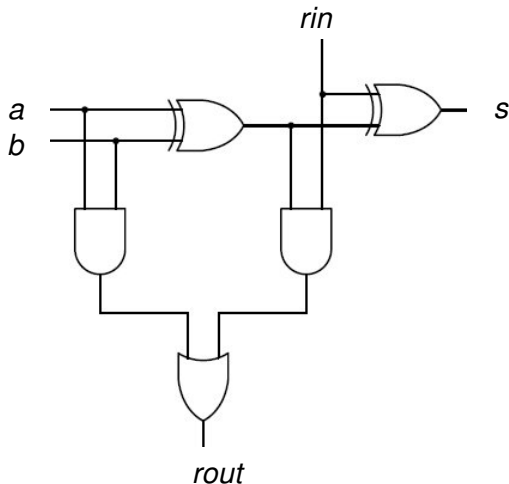


$a$	$b$	$rin$	$s$	$rou$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

## Additionneur (II)

$$s = a \oplus b \oplus \text{rin}$$

$$\text{rout} = \text{majorite}(a, b, \text{rin})$$

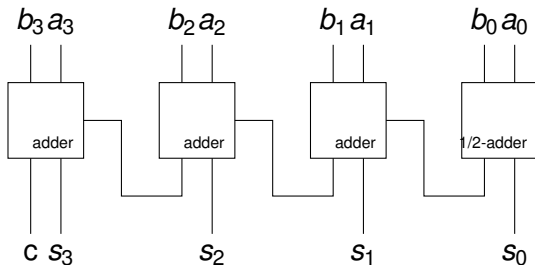


## Additionneur 4 bits (I)

On ajoute les nombres  $(b_3b_2b_1b_0)_2$  et  $(a_3a_2a_1a_0)_2$ , on obtient un résultat  $(s_3s_2s_1s_0)_2$  et un bit de débordement  $c$  (correspondant à une retenue éventuelle).

- on utilise un demi-additionneur et 3 additionneurs.
- la retenue est propagée d'un additionneur à l'autre.

## Additionneur 4 bits (II)



- **Inconvénient** : la propagation des retenues nécessite du temps qui ralentit l'opération.

## De l'idéal à la réalité

Les circuits combinatoires sont **une idéalisation** dans lesquels

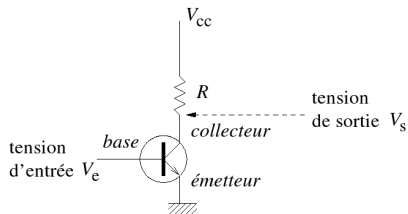
- le temps de propagation n'est pas pris en compte
- la sortie est disponible dès que les entrées sont présentes

**En réalité** le temps de passage de 0 à 1 **n'est pas**

- immédiat (temps de parcours du courant électrique)
- instantané (temps de réponse d'une porte)

# De l'idéal à la réalité (II)

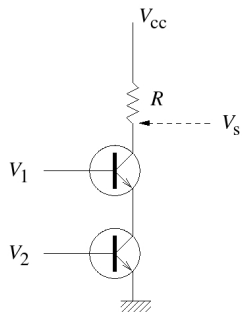
## Porte NOT



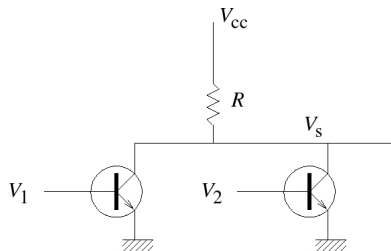
- lorsque  $V_e < crit$  alors  $V_s \approx V_{cc}$
- lorsque  $crit < V_e$  alors  $V_s \approx 0$

# De l'idéal à la réalité (II)

## Porte NAND

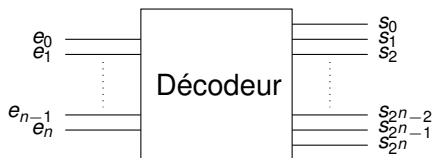


## Porte NOR



# Décodeurs (I)

- Un **décodeur** permet de **décoder un mot binaire** : il comprend  $n$  entrées et  $2^n$  sorties.
- la  $i$ ème sortie de décodeur vaut 1 si les  $n$  entrées forment l'entier  $i$ , ie  $(e_n e_{n-1} \dots e_1 e_0)_2 = (i)_{10}$ .



## Décodeurs (II)

un décodeur 2 vers 4 :

$e_0$	$e_1$	$s_0$	$s_1$	$s_2$	$s_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$$s_0 = \overline{e_0} \cdot \overline{e_1}$$

$$s_1 = \overline{e_0} \cdot e_1$$

$$s_2 = e_0 \cdot \overline{e_1}$$

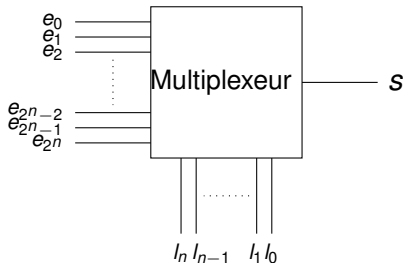
$$s_3 = e_0 \cdot e_1$$

Utilisation :

- Décodage d'une adresse : (adresse  $\rightarrow$  cellule mémoire)
- Décodage d'une instruction : (code opérande  $\rightarrow$  commande d'un circuit)

# Multiplexeur

- Un **multiplexeur** comporte  $2^n$  entrées, 1 sortie et  $n$  lignes de sélection (entrées).
- la sortie du multiplexeur vaut la valeur de la  $i$ ème entrée si l'entier  $i$  est codé sur les lignes de sélection,  $ie (l_n l_{n-1} \dots l_1 l_0)_2 = (i)_{10}$ .



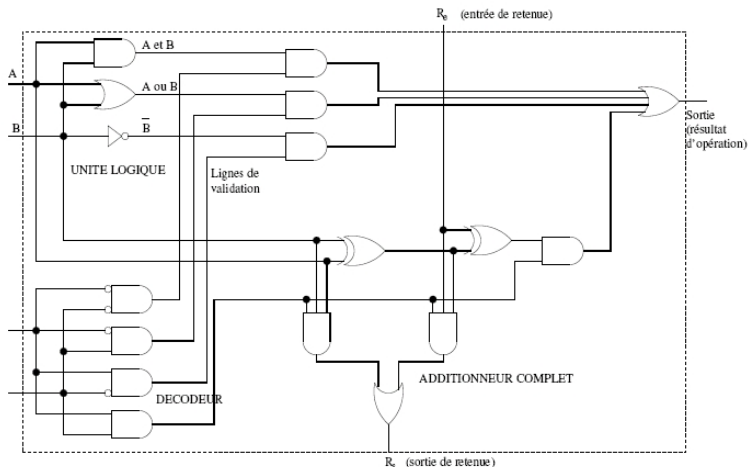
# Autres circuits combinatoires

- Comparateurs
- Sélecteurs
- Codeurs
- Démultiplexeurs
- ...

# Unité arithmétique et logique (I)

L'unité arithmétique et logique (ALU) regroupe au sein d'un même circuit différentes opérations arithmétique et logique.

## ALU 1 bit (et,ou,non,addition)



# Unité arithmétique et logique (II)

## ALU 4 bits à base d'ALU 1 bits

