Architecture des ordinateurs 1

Jean-Luc Scharbarg - ENSEEIHT - Dpt. SN

Octobre 2020

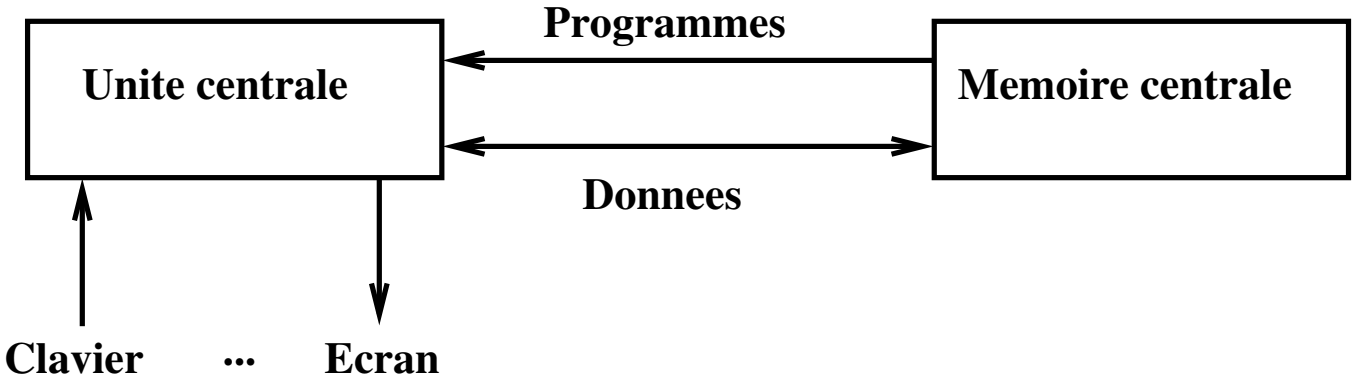
Organisation de l'enseignement

- Semestre 1 (de l'algorithme au circuit configurable)
  - 4 Cours, 3 TD, 4 TP, 1 examen écrit + 1 note de TP (présence + travail réalisé)
  - Introduction à la représentation des données
  - Circuits logiques
    - ★ Logique combinatoire
    - ★ Logique séquentielle
    - ★ Circuits séquentiels remarquables
    - ★ Construction de circuits séquentiels complexes
- Semestre 2 (de l'algorithme au processeur)
  - 3 Cours, 3 TD, 6 TP, 1 examen écrit + 1 note de TP (présence + travail réalisé)
  - Introduction à l'exécution d'un programme écrit dans un langage de haut niveau sur une architecture matérielle
  - Introduction à l'échange d'informations entre un processeur et son environnement
  - Introduction aux caractéristiques des processeurs actuels
  - Mise en œuvre d'un processeur simple

Un ordinateur, c'est quoi ?

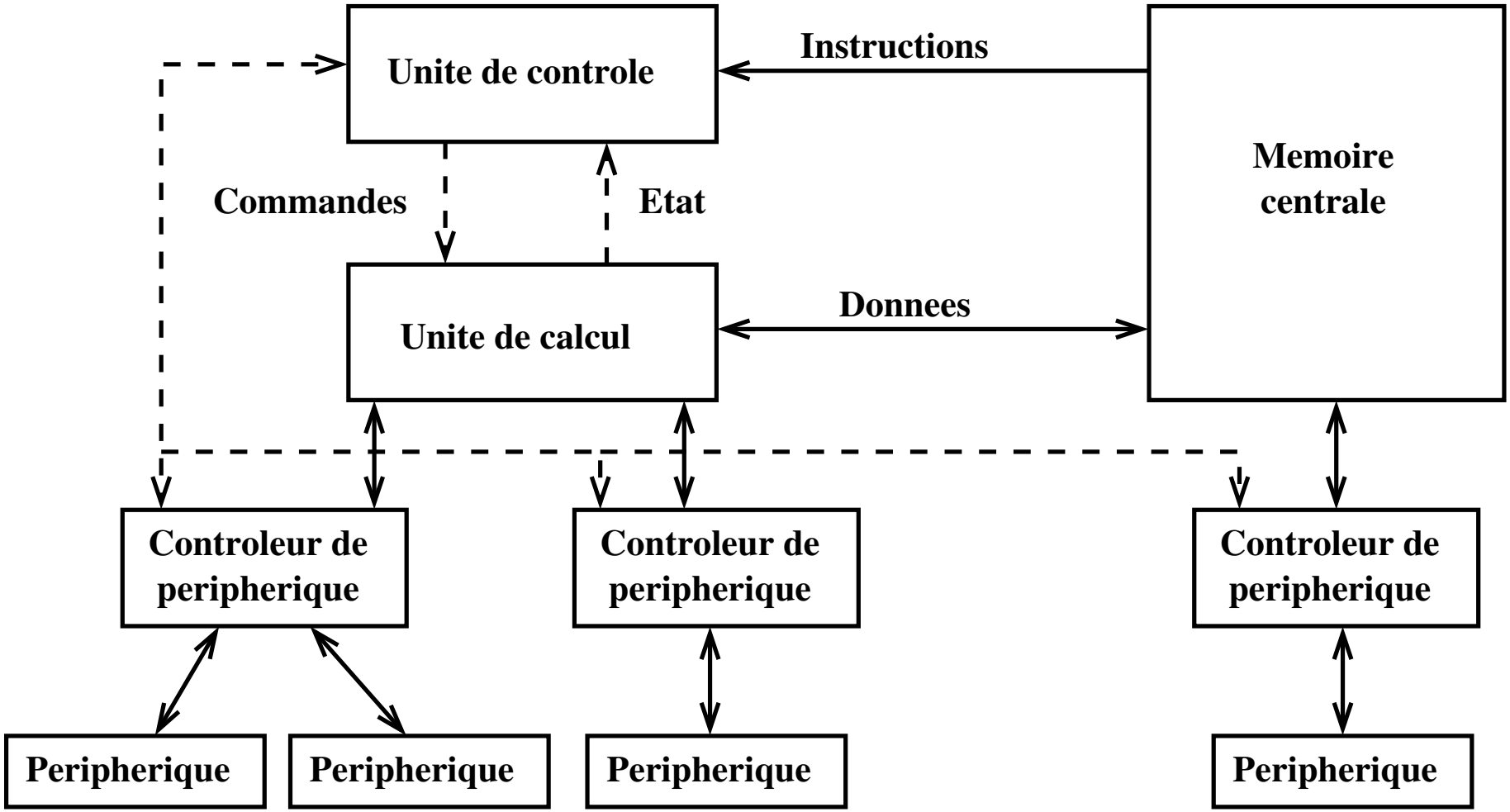
- Un ou plusieurs processeurs qui exécutent des programmes,
- Des moyens pour envoyer des ordres (clavier, souris, ...),
- Des moyens pour récupérer des résultats (écran, ...),
- Des moyens pour stocker de l'information (mémoire, disques, ...)
- Des moyens pour dialoguer avec d'autres dispositifs (interface réseau, ports, ...)

Organisation générale d'un ordinateur



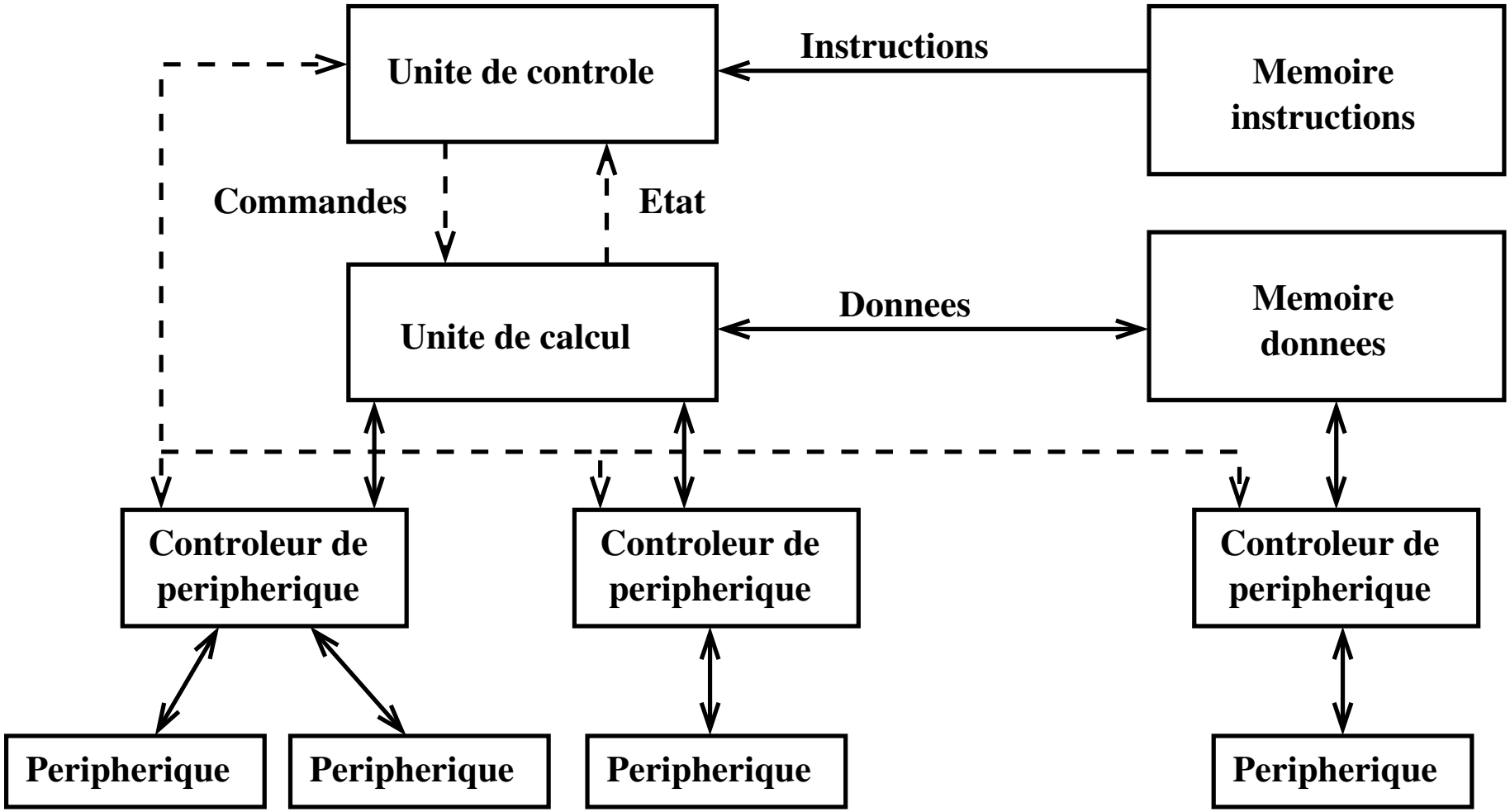
- Exécution de programmes par l'unité centrale
  - Lecture et écriture de données en mémoire centrale
  - Interactions avec l'extérieur
- Programmes et données stockées et véhiculées logiquement sous la forme de chiffres binaires ou bits (binary digits), physiquement sous la forme de signaux électroniques
- Développement d'un programme
  - Ecriture du programme dans un langage de "haut niveau" (e.g. Pascal)
  - Traduction du programme en langage machine (instructions plus rudimentaires)
  - Exécution du programme en langage machine

Modèle de Von Neuman



- Une mémoire commune aux données et aux programmes

Modèle de Harvard



- Deux mémoires séparées pour les données et les programmes

Représentation de l'information

- Calculateur : traitement automatisé des données $\Rightarrow$ représentation de ces données en machine :
  - Nombres entiers signés ou non (toutes les variables entiŕes)
  - Nombres réels (toutes les variables réelles)
  - Caractères (les variables caractères ou chaînes de caractères, le code source d'un programme, *ldots*)
  - Structures de données (assemblage des autres types de données)
  - Instructions (le code exécutable du programme)
- Représentation numérique en binaire (suites de 0 et de 1)
- Un chiffre binaire : un binary digit (bit)
- Dans cette partie du cours :
  - Représentation des entiers non signés en binaire pur
  - Représentation des entiers signés en complément à deux
  - Représentation des réels en virgule fixe
  - Représentation des réels en virgule flottante
  - Représentation des caractères

Entiers non signés en binaire pur

- Vaieur de l'entier naturel en base deux sur n bits
  $\Rightarrow$ représentation des entiers naturels α tels que $0 \leq \alpha \leq 2^n - 1$
- Le nombre de valeurs représentables est fini
- $A^n = (a^n_{n-1}, \ldots, a^n_0)$: représentation de α : $\alpha = \sum_{i=0}^{n-1}(a^n_i \times 2^i)$
- Exemple :

$$\alpha_2 = 10110011$$

$$\alpha_{10} = 1\times 2^7 + 0\times 2^6 + 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0$$

$$\alpha_{10} = 128 + 32 + 16 + 2 + 1 = 179$$

Entiers non signés en binaire pur

- Passage décimal $\rightarrow$ binaire pur :
  - Divisions successives par 2

	Quotient	Reste
$37/2$	18	1
$18/2$	9	0
$9/2$	4	1
$4/2$	2	0
$2/2$	1	0
$1/2$	0	1

$\Rightarrow 37_{10} = 100101_2$

  - Justification

$$\alpha = a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \ldots + a_1 \times 2^1 + a_0 \times 2^0$$

$$\alpha = 2 \times (a_{n-1} \times 2^{n-2} + a_{n-2} \times 2^{n-3} + \ldots + a_1 \times 2^0) + a_0$$

$$\alpha = 2 \times \mathit{Quotient} + \mathit{Reste}$$

Entiers non signés en binaire pur

- Passage binaire pur $\Rightarrow$ décimal :
  - schéma de Horner : $\alpha = \sum_{i=0}^{n-1}(a_i^n \times 2^i)$
    - $S_0 = 0$
    - $S_i = 2 \times S_{i-1} + a_{n-1}^n$ pour $1 \leq i \leq n$
    - S_n : valeur de α en décimal
  - Exemple : $\alpha = 111001_2$
    - $S_0 = 0$
    - $S_1 = 2 \times S_0 + a_5 = 2 \times 0 + 1 = 1$
    - $S_2 = 2 \times S_1 + a_4 = 2 \times 1 + 1 = 3$
    - $S_3 = 2 \times S_2 + a_3 = 2 \times 3 + 1 = 7$
    - $S_4 = 2 \times S_3 + a_2 = 2 \times 7 + 0 = 14$
    - $S_5 = 2 \times S_4 + a_1 = 2 \times 14 + 0 = 28$
    - $S_6 = 2 \times S_5 + a_0 = 2 \times 28 + 1 = 57$

Entiers non signés en binaire pur

- Opérations arithmétiques similaires aux opérations arithmétiques en décimal

$$\begin{array}{rcccccccc} & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ + & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array} \qquad \begin{array}{rcccccccc} & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ - & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}$$

- Ecriture plus compacte :
  - Représentation en octal (base huit $\Rightarrow$ chiffres de 0 à 7)

Base 10 :	8607				
Base 2 :	10	000	110	011	111
Base 8 :	2	0	6	3	7

  - Représentation en hexadécimal (base 16 $\Rightarrow$ chiffres $0, \ldots, 9, A, \ldots, F$)

Base 10 :	8607			
Base 2 :	10	0001	1001	1111
Base 16 :	2	1	9	F

Entiers signés

- Représentation d'un entier (positif, nul ou négatif) α par un nombre binaire sur n bits $A^n = (a^n_{n-1}, \ldots, a^n_0) \in [0, 2^n - 1]$
- Solution la plus immédiate : un bit pour le signe et $n-1$ bits pour la valeur absolue
  - a^n_{n-1} : bit de signe : 0 si $\alpha \geq 0$, 1 sinon
  - $(a^n_{n-2}, \ldots, a^n_0)$: représentation de $|\alpha|$ en binaire pur
  - exemples sur 8 bits :

$$\begin{array}{lcl} \alpha = 11 & \Rightarrow & A^n = 00001011 \\ \alpha = -13 & \Rightarrow & A^n = 10001101 \end{array}$$

  - Nombres représentables sur n bits : $-(2^{n-1} - 1) \leq \alpha \leq 2^{n-1} - 1$
  - Deux représentations pour $\alpha = 0$: $00\ldots0$ et $10\ldots0$
  - Intervalle des nombres représentables symétrique
  - Passage d'une représentation sur n bits à une représentation sur r bits avec $r > n$: $A^n = (a^n_{n-1}, \ldots, a^n_0) \Rightarrow A^r = (a^n_{n-1}, 0, \ldots, 0, a^n_{n-2}, \ldots, a^n_0)$
  - Pas de bonnes propriétés arithmétiques $\Rightarrow$ opérations de base complexes
- Représentation avec de bonnes propriétés arithmétiques : le complément à deux

Entiers signés en complément à deux

- Objectif : rendre le signe des opérandes transparents pour les opérations arithmétiques (addition, soustraction, ...)
- $A^n = (a^n_{n-1}, \ldots, a^n_0)$: représentation en complément à deux sur n bits de l'entier signé α

  $\alpha \geq 0 \quad \Rightarrow \quad A^n$: représentation de α en binaire pur

  $\alpha < 0 \quad \Rightarrow \quad A^n$: représentation de $\alpha + 2^n$ en binaire pur
- Nombres signés représentables sur n bits : $-(2^{n-1}) \leq \alpha \leq 2^{n-1} - 1$
- Exemple pour $n = 3$: $-4 \leq \alpha \leq 3$

$\alpha = -4$	$A^n = 100$
$\alpha = -3$	$A^n = 101$
$\alpha = -2$	$A^n = 110$
$\alpha = -1$	$A^n = 111$

$\alpha = 0$	$A^n = 000$
$\alpha = 1$	$A^n = 001$
$\alpha = 2$	$A^n = 010$
$\alpha = 3$	$A^n = 011$

- Passage d'une représentation sur n bits à une représentation sur r bits avec $r > n$:

  $A^n = (a^n_{n-1}, \ldots, a^n_0) \Rightarrow A^r = (a^n_{n-1}, \ldots, a^n_{n-1}, a^n_{n-2}, \ldots, a^n_0)$

Entiers signés en complément à deux

- Passage de la représentation de α à la représentation de $-\alpha$: Compémentation bit à bit et ajout de 1 *modulo* 2^n

$$\begin{array}{lllll} \alpha & = & 6 & \Rightarrow & A^8 = 00000110 \\ & & & & 11111001 \\ & & & & + \quad 1 \\ -\alpha & = & -6 & \Rightarrow & M\_A^8 = 11111010 \end{array}$$

- Opérations arithmétiques : A^n : représentation de α, B^n : représentation de β
  - S^n : représentation de $\alpha + \beta$
    si pas de débordement ($\alpha + \beta$ représentatble en complément à deux sur n bits) alors
    $$S^n = (A^n + B^n) \bmod 2^n$$
  - D^n : représentation de $\alpha - \beta = \alpha + (-\beta) \Rightarrow$ on se ramène à une addition

Représentation des nombres réels

- Représentation par une séquence de longueur finie de bits $\Rightarrow$ nombre fini de valeurs représentables $\Rightarrow$ pas de représentation des nombres réels au sens mathématique du terme
- Critères d'évaluation d'une représentation des nombres réels
  - Intervalle des nombres représentables
  - Précision (pourcentage d'erreur)
  - Complexité de mise en œuvre
- Deux grandes classes de représentations
  - Représentations en virgule fixe
    - ★ Les plus simples
    - ★ Pas très bonnes en terme de précision et d'intervalles de nombre représentables
  - Représentations en virgule flottante
    - ★ Plus complexes à mettre en œuvre
    - ★ bonnes en terme de précision et d'intervalle de nombres représentables
    - ★ Norme : format IEEE 754

Nombres réels en virgule fixe

- Utilisation de la représentation des nombres entiers, avec une virgule implicite toujours au même endroit dans la représentation du nombre
- Nombre réel α non signé : représentation $A^n = (a^n_{n-1}, \ldots, a^n_0)$ sur n bits, dont d après la virgule

$$\alpha = \sum_{i=0}^{n-1} (a^n_i \times 2^{i-d})$$

- Exemple avec $n = 16$ et $d = 8$

$$A^n = 00100001\ 10010000 \Rightarrow \begin{aligned} \alpha &= 2^5 + 2^0 + 2^{-1} + 2^{-4} \\ \alpha &= 32 + 1 + \tfrac{1}{2} + \tfrac{1}{16} \\ \alpha &= 33.5625 \end{aligned}$$

- Calcule de la représentation en virgule fixe

	Quotient	Reste
6/2	3	0
3/2	1	1
1/2	0	1

	Produit	Partie entière
0.375×2	0.75	0
0.75×2	1.5	1
0.5×2	1.0	1

$\Rightarrow 6.375_{10} = 110.011_2$

Nombres réels en virgule fixe

- Représentation non exacte de la plupart des nombres réels
  Exemple avec $n = 16$ et $d = 8$: $11.8_{10} = 00001011.11001100_2$
- Opérations arithmétiques en virgule fixe similaires aux opérations arithmétiques sur les entiers non signés

$$\begin{array}{rcr} \alpha & = & 2.75 \\ \beta & = & 3.625 \\ \hline & & 6.375 \end{array} \quad \left| \quad \begin{array}{rcccccccc} & 0 & 0 & 1 & 0 & . & 1 & 1 & 0 \\ + & 0 & 0 & 1 & 1 & . & 1 & 0 & 1 \\ \hline & 0 & 1 & 1 & 0 & . & 0 & 1 & 1 \end{array} \right.$$

- Nombres réels signés en virgule fixe : utilisation du complément à deux en faisant abstraction de la virgule
  Exemple pour $n = 7$ et $d = 3$

$$\begin{array}{rrcrcl} & \alpha & = & 2.75 & \Rightarrow & \text{Représentation : } 0010.110 \\ - & \alpha & = & -\ 2.75 & \Rightarrow & \text{Représentation : } 1101.010 \end{array}$$

- Opérations arithmétiques comme sur les entiers signés

$$\begin{array}{rcr} \alpha & = & 2.75 \\ \beta & = & -\ 3.625 \\ \hline & & -\ 0.875 \end{array} \quad \left| \quad \begin{array}{rcccccccc} & 0 & 0 & 1 & 0 & . & 1 & 1 & 0 \\ + & 1 & 1 & 0 & 0 & . & 0 & 1 & 1 \\ \hline & 1 & 1 & 1 & 1 & . & 0 & 0 & 1 \end{array} \right.$$

Nombres réels en virgule fixe

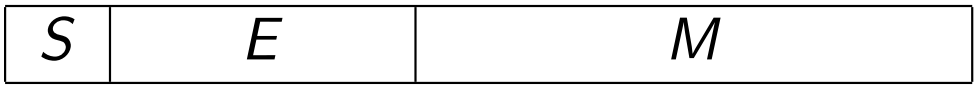
- Différence entre deux valeurs consécutives représentables exactement : 2^{-d}, constante sur tout l'intervalle de représentation
  Exemple pour $n = 7$ et $d = 3$
  $$\begin{array}{llll} 0000.000 & \Rightarrow & \alpha & = & 0 \\ 0000.001 & \Rightarrow & \alpha & = & 2^{-3} \\ 0000.010 & \Rightarrow & \alpha & = & 2\times 2^{-3} \\ \ldots & & & & \end{array}$$
- Pourcentage d'erreur potentielle d'autant plus fort que la valeur absolue du nombre réel représenté est petite
- Intervalle des valeurs représentable relativement réduit : $[-2^{n-d-1}, 2^{n-d-1} - 2^{-d}]$
  Pour $n = 32$ et $d = 16$: $[-2^{15}, 2^{15} - 2^{-16}]$
- Représentation en virgule fixe de moins en moins utilisée de nos jours

Nombres réels en virgule flottante : principe

- S'inspire de la notation scientifique des nombres réels :

$$\alpha = f \times 10^e \ (\textit{en base } 10)$$

- f : mantisse, e : exposant
- Notation non unique : $3.14 \times 10^0 = 0.314 \times 10^1 = 314 \times 10^{-2}$
- Virgule flottante : notation scientifique en base 2
  α représenté par $\mu_2 \times 2^{\epsilon_2}$
- Deux possibilités pour la normalisation de l'écriture :
  $\mu \in [0.5, 1[\quad \Rightarrow \quad \mu = 0.1\ldots$
  $\mu \in [1, 2[\quad \Rightarrow \quad \mu = 1.\ldots$
- Représentation en machine avec trois champs



S : signe du nombre sur 1 bit
E : exposant en excédent sur e bits
M : mantisse sur m bits

Nombres réels en virgule flottante : IEEE 754

- Standard défini en 1985 par l'IEEE (Institute of Electrical and Electronics Engineers)
- Utilisé par la plupart des processeurs actuels
- Deux formats principaux définis :
  - simple précision sur 32 bits : $e = 8$, $m = 23$
  - double précision sur 64 bits : $e = 11$, $m = 52$
- Nombres réels normalisés : $\alpha = 1.\mu' \times 2^{\epsilon}$

$$M = \mu' \text{ et } E = \epsilon + 2^{e-1} - 1$$

Exemple : $\alpha = -9.5 \Rightarrow \alpha = -1.0011_2 \times 2^{11_2}$

Simple précision : $S = 1$

$E = 3 + 127 = 130_{10} = 10000010_2$

$M = 00110\ldots0$

Double précision : $S = 1$

$E = 3 + 1023 = 1026_{10} = 10\ldots010_2$

$M = 00110\ldots0$

Nombres réels en virgule flottante : IEEE 754

- Nombres réels dénormalisés : représentation des nombres réels à très petite valeur absolue
  - tous les bits du champ E à 0 $\Rightarrow \epsilon = -127$ (simple précision) ou $\epsilon = -1023$ (double précision)
  - la mantisse vaut $0.M$
  - Exemple en simple précision :
    $S = 0,\ E = 0\ldots0\ M = 010\ldots0 \Rightarrow \alpha_{10} = 0.25 \times 2^{-127}$
- Représentation de 0 : $E = 0\ldots0,\ M = 0\ldots0,\ S$ *indifférent*
- Représentation de l'infini : $E = 1\ldots1,\ M \neq 0\ldots0$
- Représentation de NaN (Not a Number : indéfini) :
  $E = 1\ldots1,\ M = 0\ldots0$
- Intervalle des nombres représentables
  - Simple précision : de -10^{38} à 10^{38}
  - Double précision : de -10^{308} à 10^{308}
- Pourcentage d'erreur potentielle constant

Représentation des caractères

- Codage numérique d'un répertoire de caractères
- La longueur du code définit le nombre maximum de caractères du répertoire
- Le code ASCII (American Standard Code for Information Interchange)
  - Le standard sur les machines actuelles
  - Codage sur 7 bits $\Rightarrow$ 128 caractères possibles
  - Code construit pour simplifier la manipulation des caractères
- Le code ISO-8859-1 (ASCII étendu)
  - Codage sur 8 bits $\Rightarrow$ 256 caractères possibles
  - Langues latines (accents, ...)
- Les codes universels sur 2 ou 4 octets

Table ASCII (1)

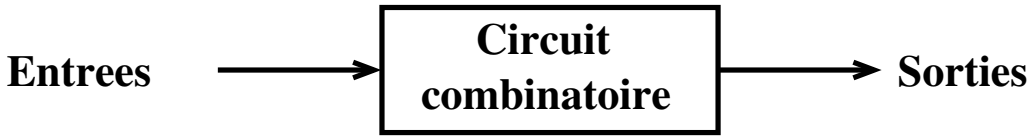
Hexa	Carac	Hexa	Carac	Hexa	Carac	Hexa	Carac
0	NUL	10	DLE	20		30	0
1	SOH	11	DC1	21	!	31	1
2	STX	12	DC2	22	"	32	2
3	ETX	13	DC3	23	#	33	3
4	EOT	14	DC4	24	$	34	4
5	ENQ	15	NAK	25	%	35	5
6	ACK	16	SYN	26	&	36	6
7	BEL	17	ETB	27	'	37	7
8	BS	18	CAN	28	(	38	8
9	TAB	19	EM	29	)	39	9
A	LF	1A	SUB	2A	*	3A	:
B	VT	1B	ESC	2B	+	3B	;
C	FF	1C	FS	2C	,	3C	<
D	CR	1D	GS	2D	-	3D	=
E	SO	1E	RS	2E	.	3E	>
F	SI	1F	US	2F	/	3F	?

Table ASCII (2)

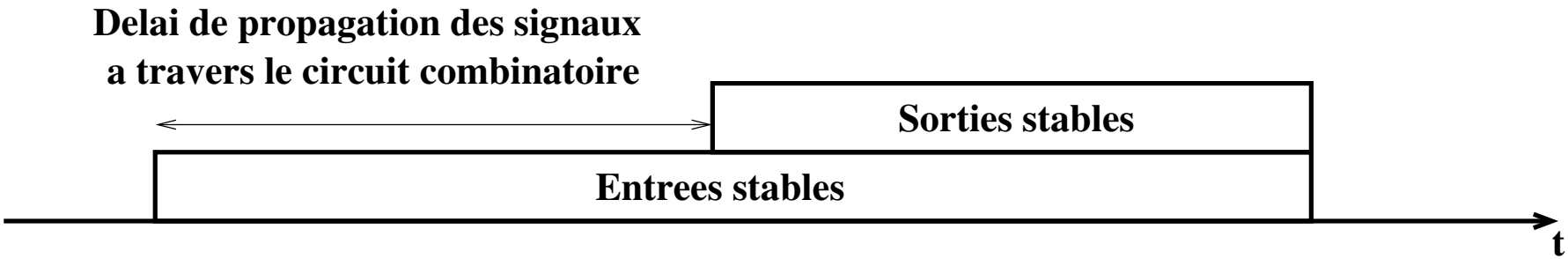
Hexa	Carac	Hexa	Carac	Hexa	Carac	Hexa	Carac
40	@	50	P	60	‘	70	p
41	A	51	Q	61	a	71	q
42	B	52	R	62	b	72	r
43	C	53	S	63	c	73	s
44	D	54	T	64	d	74	t
45	E	55	U	65	e	75	u
46	F	56	V	66	f	76	v
47	G	57	W	67	g	77	w
48	H	58	X	68	h	78	x
49	I	59	Y	69	i	79	y
4A	J	5A	Z	6A	j	7A	z
4B	K	5B	[	6B	k	7B	{
4C	L	5C	\	6C	l	7C	|
4D	M	5D	]	6D	m	7D	}
4E	N	5E	^	6E	n	7E	~
4F	O	5F	_	6F	o	7F	DEL

Circuit combinatoire

- Circuit tel que l'état des sorties ne dépend que de l'état des entrées (pas d'état interne du circuit)



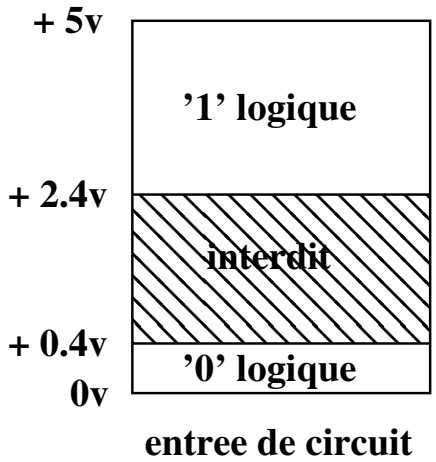
- Pas de rebouclage dans le circuit
- Evaluation de l'état des sorties lorsque les entrées sont stables, en tenant compte du délai de propagation des signaux

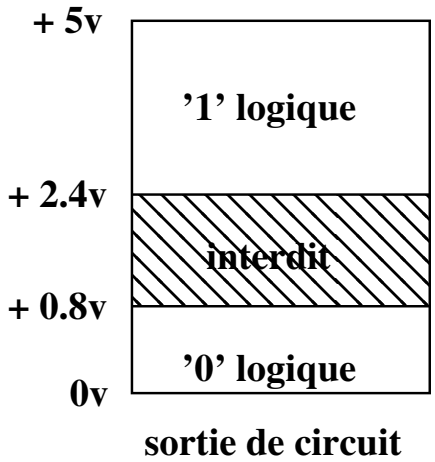


- Exemple de circuit combinatoire : l'additionneur 1 bit

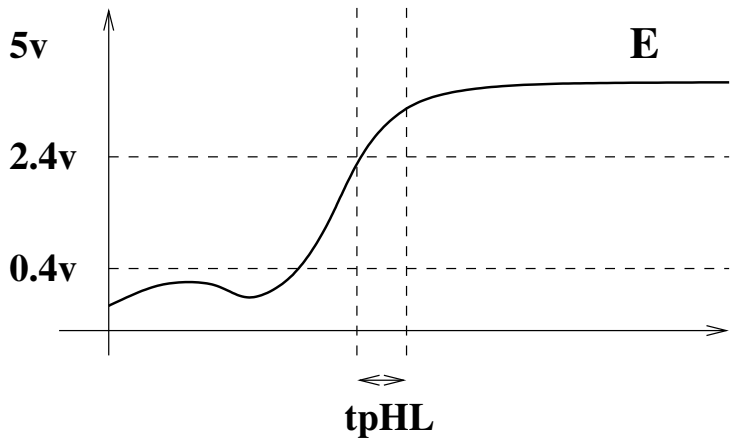
Bits et signaux électriques

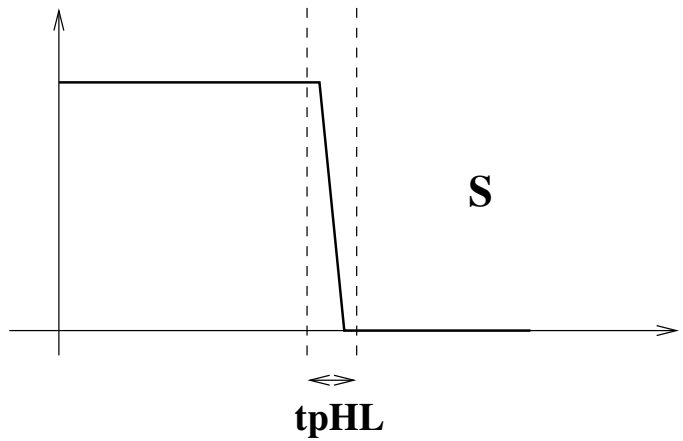
- Assignation des tensions électriques (technologie TTL)





- Remise en forme d'un signal (après traversée d'une porte NON)



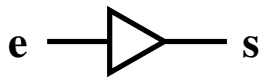


Notion de portes logiques

- Porte logique : brique de base pour la description d'un circuit combinatoire
- Porte logique = circuit logique réalisant une fonction logique élémentaire
- Une porte logique est définie par :
  - La fonction logique mise en œuvre
  - Le schéma électrique
  - Les caractéristiques temporelles et électriques (temps de propagation, temps de transition, ...)
- Une famille de portes logiques est définie par la technologie utilisée
  - TTL (Transistor-transistor Logic)
    - ★ Technologie standard des ordinateurs à partir des années 1960
    - ★ Consommation modérée, mais capacités d'intégration faibles
    - ★ Encore utilisée pour de petites réalisations
  - ECL (Emitter-coupled logic)
    - ★ Grande rapidité, mais consommation élevée
    - ★ Utilisée dans des petites réalisations où la vitesse est critique
  - CMOS (Complementary metal-oxide-semiconductor)
    - ★ La technologie reine
    - ★ Consommation réduite, taux d'intégration très élevé

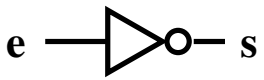
Symboles non normalisés des portes logiques

Amplificateur



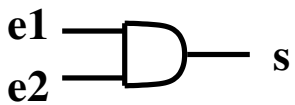
e	s
0	0
1	1

Inverseur (non, not)



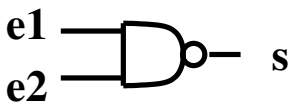
e	s
0	1
1	0

Et (and)



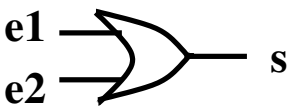
e1	e2	s
0	0	0
0	1	0
1	0	0
1	1	1

Non et (nand)



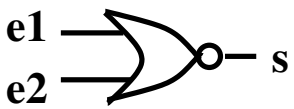
e1	e2	s
0	0	1
0	1	1
1	0	1
1	1	0

Ou (or)



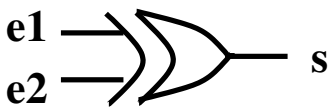
e1	e2	s
0	0	0
0	1	1
1	0	1
1	1	1

Non ou (nor)



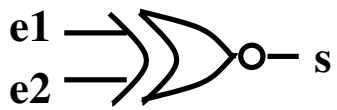
e1	e2	s
0	0	1
0	1	0
1	0	0
1	1	0

Ou exclusif (xor)



e1	e2	s
0	0	0
0	1	1
1	0	1
1	1	0

Non ou exclusif (xnor)



e1	e2	s
0	0	1
0	1	0
1	0	0
1	1	1

Une porte universelle : la porte nand

- a nand b = non (a et b)
- Permet de construire n'importe quelle fonction booléenne

$$
\begin{array}{lcl}
\text{non a} & = & \text{non (a et a)} \\
 & = & \text{a nand a} \\
\text{a et b} & = & \text{non (non (a et b))} \\
 & = & \text{(a nand b) nand (a nand b)} \\
\text{a ou b} & = & \text{non ((non a) et (non b))} \\
 & = & \text{(a nand a) nand (b nand b)}
\end{array}
$$

- Il en est de même pour la porte nor

Conception d'un circuit combinatoire

- Un circuit combinatoire met en œuvre une fonction booléenne
  - application de $\{0,1\}^n$ dans $\{0,1\}^m$, avec n et m, entiers naturels positifs
    exemple : fonction booléenne f de $\{0,1\}^2$ dans $\{0,1\}^2$ telle que $f(x_1, x_2) = (x_1 + x_2, x_1 \cdot x_2)$
  - $m = 1 \Rightarrow$ fonction simple, $m > 1 \Rightarrow$ fonction générale
  - Fonction générale : ensemble de fonctions simples
  - Une fonction booléenne simple peut s'écrire sous la forme d'une somme de monômes
    Exemple: $f(x_1, x_2, x_3) = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2 \cdot x_3$
  - Cette écriture n'est pas unique
    $x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2 \cdot x_3 = x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot x_2 \cdot x_3$
- Un circuit combinatoire est mis en œuvre par un ensemble de portes logiques
- De nombreuses mises en œuvre possibles
- Essayer de minimiser la complexité du circuit

Représentation d'une fonction booléenne :

$f(x_1, x_2, x_3) = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2 \cdot x_3$

Table de vérité

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Tableau de Karnaugh

$x_1x_2 \backslash x_3$	0	1
00		
01		1
11		
10	1	1

Fonctions booléennes simples

- Fonctions d'une variable simple

	$x=1$	$x=0$	Expression	Remarque
$f(x)$	0	0	$f_0 = 0$	Nulle
	0	1	$f_1 = \overline{x}$	Complément
	1	0	$f_2 = x$	Identité
	1	1	$f_3 = 1$	Tautologie

- Fonctions de deux variables simples

	$xy =$				Expression	Remarque
	11	10	01	00		
$f(x,y)$	0	0	0	0	$f_0 = 0$	Nulle
	0	0	0	1	$f_1 = \overline{(x+y)}$	Nor
	0	0	1	0	$f_2 = \overline{x} \cdot y$	
	0	0	1	1	$f_3 = \overline{x}$	Complément de x
	0	1	0	0	$f_4 = x \cdot \overline{y}$	

Fonctions booléennes simples

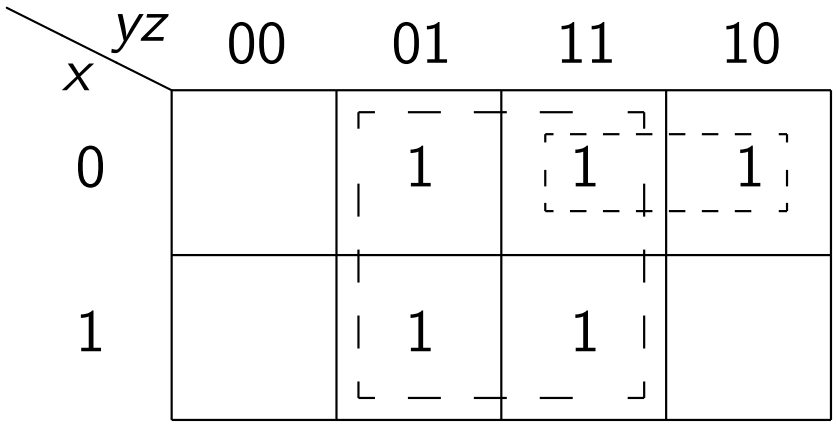
- Fonctions de deux variables simples

	$xy =$				Expression	Remarque
	11	10	01	00		
$f(x,y)$	0	1	0	1	$f_5 = \overline{y}$	Complément de y
	0	1	1	0	$f_6 = x \oplus y$	Ou exclusif
	0	1	1	1	$f_7 = \overline{(x \cdot y)}$	Nand
	1	0	0	0	$f_8 = x \cdot y$	Et
	1	0	0	1	$f_9 = x \Longleftrightarrow y$	Equivalence
	1	0	1	0	$f_{10} = y$	Identité de y
	1	0	1	1	$f_{11} = x \Rightarrow y$	Implication
	1	1	0	0	$f_{12} = x$	Identité de x
	1	1	0	1	$f_{13} = y \Rightarrow x$	Implication
	1	1	1	0	$f_{14} = x + y$	Ou
	1	1	1	1	$f_{15} = 1$	Tautologie

Minimisation des fonctions booléennes : définitions

- Monôme m premier dans une fonction booléenne f :
  - $m \leq f$
  - il n'existe pas de monôme $m' \neq m$ tel que $m' \leq f$ et $m \leq m'$

  Sur un tableau de Karnaugh, plus grand regroupement de 2^p cases adjacentes où la fonction vaut 1



Monômes premiers :

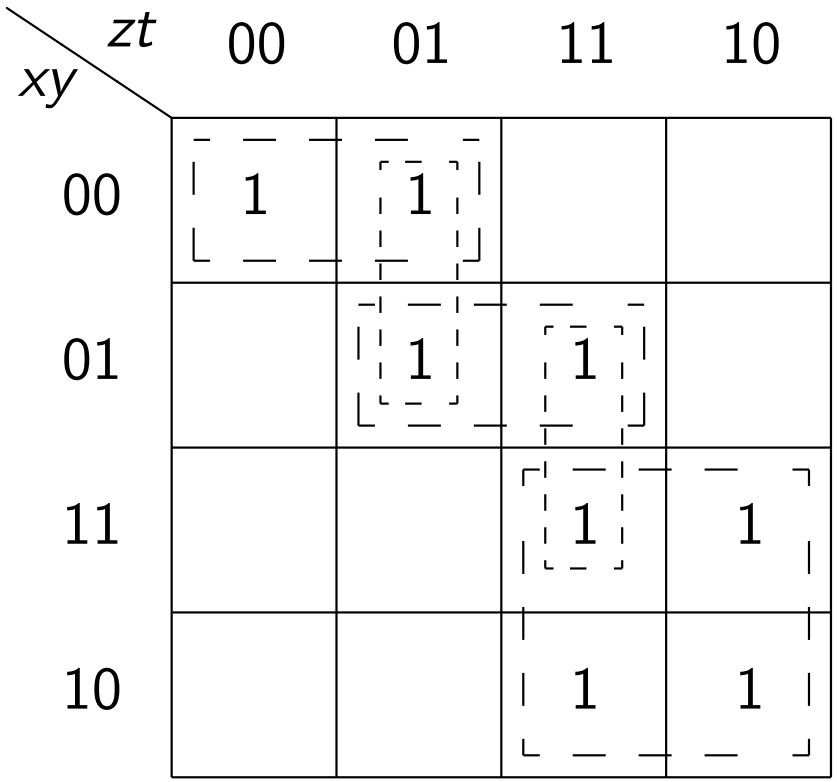
z

$\overline{x} \cdot y$

- Base d'une fonction booléenne : somme de monômes premiers égale à la fonction
- Base complète : somme de tous les monômes premiers de la fonction
- Base irredondante : cesse d'être une base quand on lui enlève un de ses monômes

Minimisation des fonctions booléennes : définitions

- Monôme premier obligatoire dans une base : le seul parmi tous les monômes de la base à couvrir un ou plusieurs points $\Rightarrow$ ne peut pas être supprimé de la base
- Base irredondante : tous ses monômes sont obligatoires
- Monôme premier essentiel : obligatoire dans toutes les bases de la fonction



Monômes premiers :
$x \cdot z$, $y \cdot z \cdot t$, $\overline{x} \cdot y \cdot t$, $\overline{x} \cdot \overline{z} \cdot t$, $\overline{x} \cdot \overline{y} \cdot \overline{z}$

Monômes essentiels :
$x \cdot z$, $\overline{x} \cdot \overline{y} \cdot \overline{z}$

Base irredondante :
$x \cdot z$, $\overline{x} \cdot \overline{y} \cdot \overline{z}$, $\overline{x} \cdot y \cdot t$

Base irredondante :
$x \cdot z$, $\overline{x} \cdot \overline{y} \cdot \overline{z}$, $y \cdot z \cdot t$, $\overline{x} \cdot \overline{z} \cdot t$

Minimisation des fonctions booléennes : principe

- Recherche d'une écriture minimale de la fonction sous la forme d'une somme de monômes
- Principe général :
  - Etablissement de la liste des monômes premiers
  - Choix d'une base irredondante
- Utilisation d'un tableau de Karnaugh
- D'autres types de méthodes, plus algébriques :
  - Méthode de Quine / Mac Cluskey
  - Méthode de Tison (consensus)

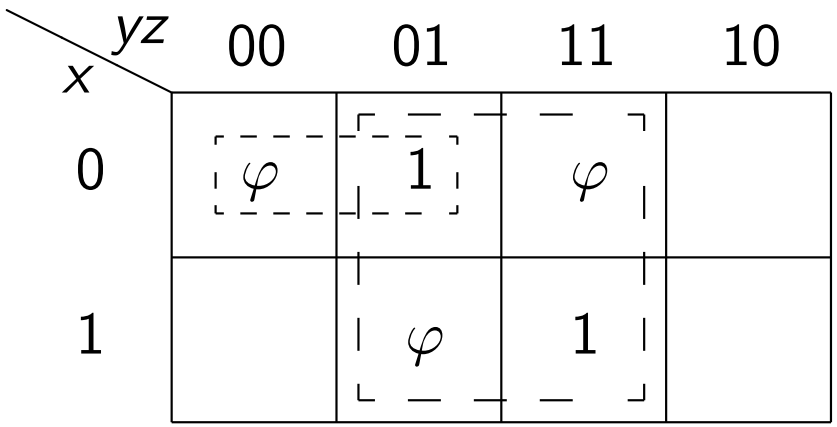
Minimisation des fonctions booléennes : exercice

- Minimiser la fonction booléenne décrite par la table de vérité suivante

a	b	c	d	f(a,b,c,d)
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Notion de fonction φ-booléenne

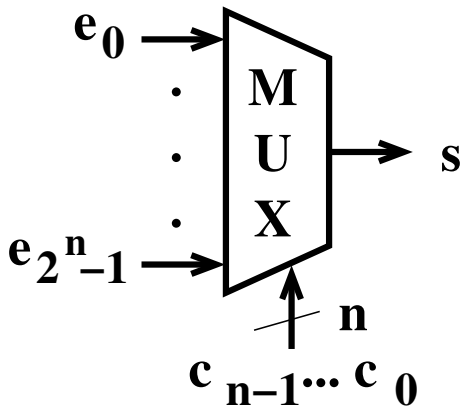
- Prise en compte de la notion d'indifférence du résultat : par exemple valeur d'une fonction correspondant à une configuration impossible des entrées
- La fonction peut prendre les valeurs 0, 1 ou φ
- Principe de la minimisation d'une fonction φ-booléenne sur un tableau de Karnaugh
  - Monômes premiers en considérant les φ à 1
  - Base irredondante en considérant les φ à 0



Monômes premiers :
z, $\overline{x} \cdot \overline{y}$
Base irredondante :
z

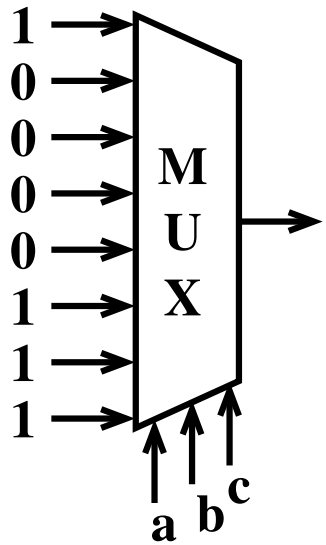
Le multiplexeur

- Macro-fonction combinatoire : sélection d'une entrée parmi 2^n, en fonction de n commandes



$$s = e_i \ avec \ i = \sum_{k=0}^{n} c_k 2^k$$

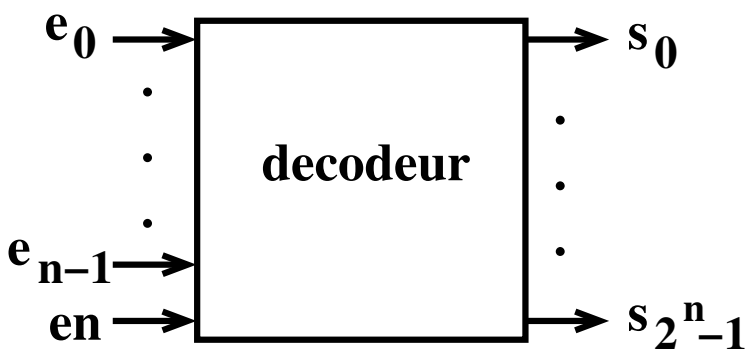
- Mise en œuvre d'une fonction booléenne



$$f(a,b,c) = \bar{a}\,\bar{b}\,\bar{c} + a\,\bar{b}\,c + a\,b\,c + a\,b\,\bar{c}$$

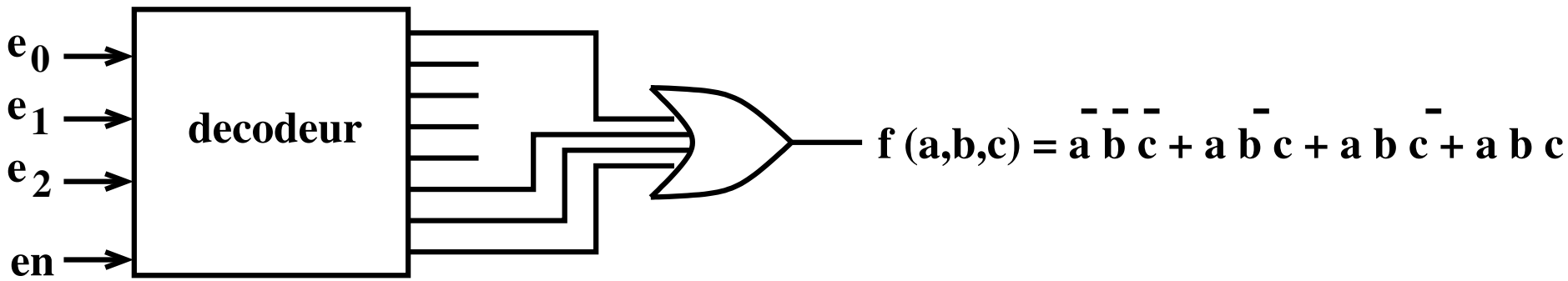
Le décodeur

- Macro-fonction combinatoire : activation de la sortie correspondant à la valeur binaire présente sur les entrées, lorsque le composant est validé



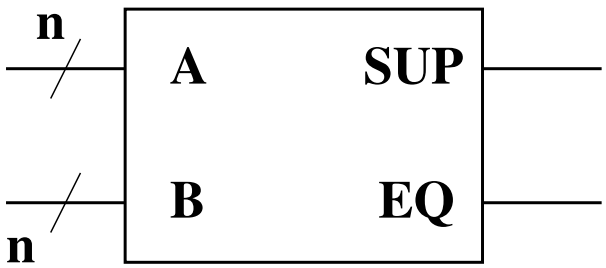
$$
\begin{array}{lll}
en = 1 & : & s_i = 1 \ pour \ i = \sum_{k=0}^{n} e_k 2^k \\
 & & s_i = 0 \ pour \ i \neq \sum_{k=0}^{n} e_k 2^k \\
en = 0 & : & s_i = 0 \ pour \ tout \ i
\end{array}
$$

- Mise en œuvre d'une fonction booléenne



Comparateurs

- Vue générale du circuit : entiers sur n bits



A > B : SUP = 1
A >= B : SUP = 1 ou EQ = 1
A = B : EQ = 1
A < B : SUP = 0 et EQ = 0
A<= B : SUP = 0

- Comparateur non signé avec $n = 2$

$$\begin{aligned} EQ &= (a_1 = b_1) \cdot (a_0 = b_0) \\ SUP &= a_1 \cdot \overline{b_1} + ((a_1 = b_1) \cdot a_0 \cdot \overline{b_0}) \end{aligned}$$

- Comparateur non signé avec n quelconque

$$\begin{aligned} EQ &= (a_{n-1} = b_{n-1}) \cdot (a_{n-2} = b_{n-2}) \cdot \ldots \cdot (a_0 = b_0) \\ SUP &= a_{n-1} \cdot \overline{b_{n-1}} + ((a_{n-1} = b_{n-1}) \cdot a_{n-2} \cdot \overline{b_{n-2}} + \\ &\quad ((a_{n-2} = b_{n-2}) \cdot a_{n-3} \cdot \overline{b_{n-3}} + \cdots \\ &\quad ((a_1 = b_1) \cdot a_0 \cdot \overline{b_0}) \ldots)) \end{aligned}$$

Comparateurs

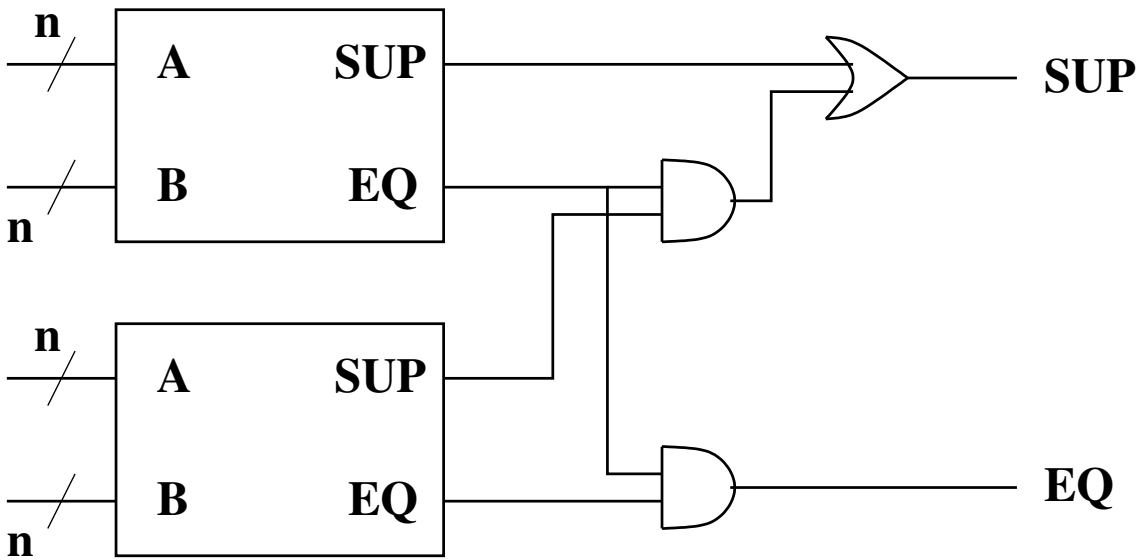
- Comparateur signé avec $n = 2$

$$\begin{aligned} EQ &= (a_1 = b_1) \cdot (a_0 = b_0) \\ SUP &= \overline{a_1} \cdot b_1 + ((a_1 = b_1) \cdot a_0 \cdot \overline{b_0}) \end{aligned}$$

- Comparateur signé avec n quelconque

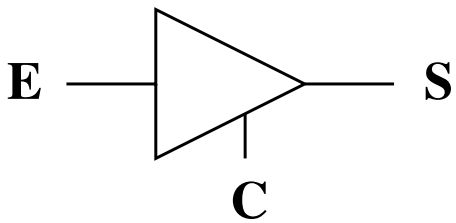
$$\begin{aligned} EQ &= (a_{n-1} = b_{n-1}) \cdot (a_{n-2} = b_{n-2}) \cdot \ldots \cdot (a_0 = b_0) \\ SUP &= \overline{a_{n-1}} \cdot b_{n-1} + ((a_{n-1} = b_{n-1}) \cdot a_{n-2} \cdot \overline{b_{n-2}} + \\ &\quad ((a_{n-2} = b_{n-2}) \cdot a_{n-3} \cdot \overline{b_{n-3}} + \cdots \\ &\quad ((a_1 = b_1) \cdot a_0 \cdot \overline{b_0}) \ldots)) \end{aligned}$$

- Assemblage de comparateurs : entiers sur $2n$ bits



Sortie haute impédance et buffers 3-états

- Etat “haute impédance” (High-Z) en plus des états '0' et '1'
- Sortie en haute impédance $\Rightarrow$ comme si elle n'était plus connectée.
- Permet de relier ensemble plusieurs sorties de ce type, sous réserve de garantir qu'à tout instant, une seule des sorties n'est pas en haute impédance



E	C	S
0	0	High–Z
0	1	0
1	0	High–Z
1	1	1

- Une solution pour mettre en œuvre des multiplexeurs

Exercice

- On souhaite mettre en œuvre la fonction *Majorité* de 5 variables $x_0, \ldots, x_4$, qui vaut 1 si une majorité des variables d'entrée (au moins 3) vaut 1 et 0 sinon
- Proposer une mise en œuvre en utilisant des multiplexeurs à trois commandes et un multiplexeur à deux commandes