

# Architecture des ordinateurs 1

Jean-Luc Scharbarg - ENSEEIHT - Dpt. SN

Novembre 2024

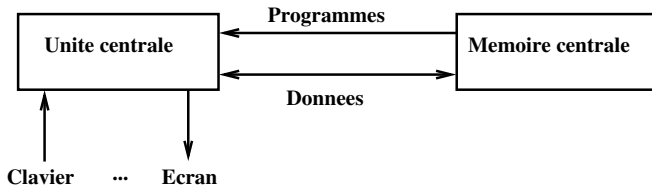
# Organisation de l'enseignement

- 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, 7 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 ?

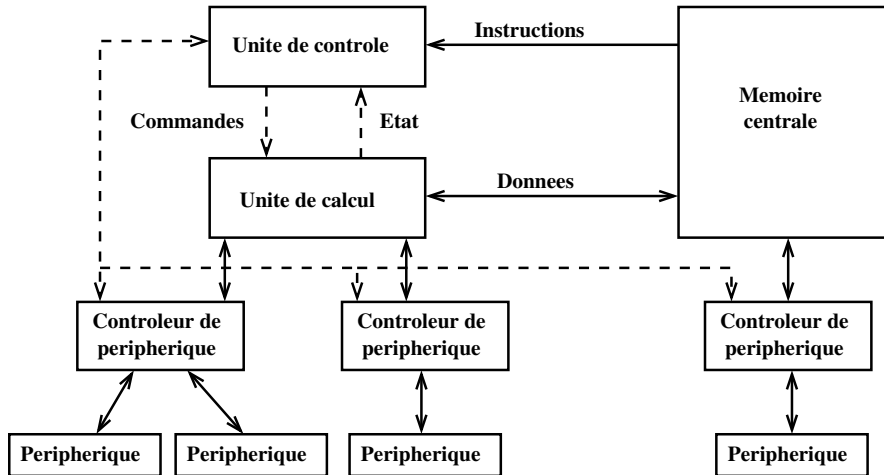
- Un ou plusieurs processeurs qui exécutent des programmes,
- Des moyens pour envoyer des ordres (clavier, souris, ...),
- Des moyens pour récupérer des résultats (écran, ...),
- Des moyens pour stocker de l'information (mémoire, disques, ...)
- Des moyens pour dialoguer avec d'autres dispositifs (interface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

# Organisation générale d'un ordinateur



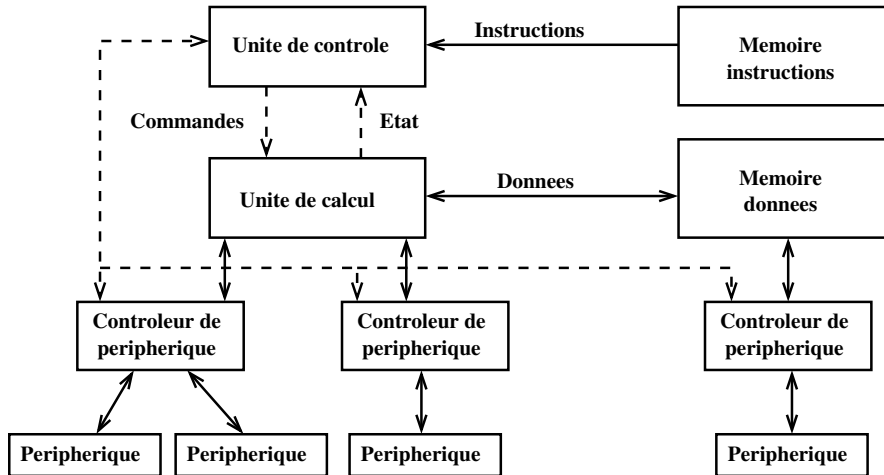
- Exécution de programmes par l'unité centrale
  - ▶ Lecture et écriture de données en mémoire centrale
  - ▶ Interactions avec l'extérieur
- Programmes et données stockés et véhiculés logiquement sous la forme de chiffres binaires ou bits (binary digits), physiquement sous la forme de signaux électroniques
- 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 entières)
  - ▶ Nombres réels (toutes les variables réelles)
  - ▶ Caractères (les variables caractères ou chaînes de caractères, le code source d'un programme, ...)
  - ▶ 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

- Valeur de l'entier naturel en base deux sur  $n$  bits  
 $\Rightarrow$  représentation des entiers naturels  $\alpha$  tels que  $0 \leq \alpha \leq 2^n - 1$
- Le nombre de valeurs représentables est fini
- $A^n = (a_{n-1}^n, \dots, a_0^n)$  : représentation de  $\alpha$  :  $\alpha = \sum_{i=0}^{n-1} (a_i^n \times 2^i)$
- 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

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

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

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

# 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  $\alpha$  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

$$\begin{array}{r} 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ +\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1 \\ \hline 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0 \end{array}$$

$$\begin{array}{r} 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ -\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1 \\ \hline 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0 \end{array}$$

- 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, ..., 9, A, ..., 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)  $\alpha$  par un nombre binaire sur  $n$  bits  $A^n = (a_{n-1}^n, \dots, a_0^n) \in [0, 2^n - 1]$
- Solution la plus immédiate : un bit pour le signe et  $n - 1$  bits pour la valeur absolue
  - ▶  $a_{n-1}^n$  : bit de signe : 0 si  $\alpha \geq 0$ , 1 sinon
  - ▶  $(a_{n-2}^n, \dots, a_0^n)$  : représentation de  $|\alpha|$  en binaire pur
  - ▶ exemples sur 8 bits :

$$\alpha = 11 \quad \Rightarrow \quad A^n = 00001011$$

$$\alpha = -13 \quad \Rightarrow \quad A^n = 10001101$$

- ▶ Nombres représentables sur  $n$  bits :  $-(2^{n-1} - 1) \leq \alpha \leq 2^{n-1} - 1$
  - ▶ Deux représentations pour  $\alpha = 0$  :  $00 \dots 0$  et  $10 \dots 0$
  - ▶ Intervalle des nombres représentables symétrique
  - ▶ Passage d'une représentation sur  $n$  bits à une représentation sur  $r$  bits avec  $r > n$  :  $A^n = (a_{n-1}^n, \dots, a_0^n) \Rightarrow A^r = (a_{n-1}^n, 0, \dots, 0, a_{n-2}^n, \dots, a_0^n)$
  - ▶ 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, ...)
- $A^n = (a_{n-1}^n, \dots, a_0^n)$  : représentation en complément à deux sur  $n$  bits de l'entier signé  $\alpha$ 
  - $\alpha \geq 0 \Rightarrow A^n$  : représentation de  $\alpha$  en binaire pur
  - $\alpha < 0 \Rightarrow A^n$  : 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  $r > n$  :  
 $A^n = (a_{n-1}^n, \dots, a_0^n) \Rightarrow A^r = (a_{n-1}^n, \dots, a_{n-1}^n, a_{n-2}^n, \dots, a_0^n)$

# Entiers signés en complément à deux

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

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

- Opérations arithmétiques :  $A^n$  : représentation de  $\alpha$ ,  $B^n$  : représentation de  $\beta$ 
  - ▶  $S^n$  : représentation de  $\alpha + \beta$   
si pas de débordement ( $\alpha + \beta$  représentable 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  $\alpha$  non signé : représentation  $A^n = (a_{n-1}^n, \dots, a_0^n)$  sur  $n$  bits, dont  $d$  après la virgule

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

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

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

- Calcule de la représentation en virgule fixe

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

	Produit	Partie entière
$0.375 \times 2$	0.75	0
$0.75 \times 2$	1.5	1
$0.5 \times 2$	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

$$\begin{array}{r|l} \alpha = 2.75 & 0 \ 0 \ 1 \ 0 \ . \ 1 \ 1 \ 0 \\ \beta = 3.625 & + \ 0 \ 0 \ 1 \ 1 \ . \ 1 \ 0 \ 1 \\ \hline 6.375 & 0 \ 1 \ 1 \ 0 \ . \ 0 \ 1 \ 1 \end{array}$$

- Nombres réels signés en virgule fixe : utilisation du complément à deux en faisant abstraction de la virgule

Exemple pour  $n = 7$  et  $d = 3$

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

- Opérations arithmétiques comme sur les entiers signés

$$\begin{array}{r|l} \alpha = 2.75 & 0 \ 0 \ 1 \ 0 \ . \ 1 \ 1 \ 0 \\ \beta = -3.625 & + \ 1 \ 1 \ 0 \ 0 \ . \ 0 \ 1 \ 1 \\ \hline -0.875 & 1 \ 1 \ 1 \ 1 \ . \ 0 \ 0 \ 1 \end{array}$$

# 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  $d = 3$

$$0000.000 \Rightarrow \alpha = 0$$

$$0000.001 \Rightarrow \alpha = 2^{-3}$$

$$0000.010 \Rightarrow \alpha = 2 \times 2^{-3}$$

...

- 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 :

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

- $f$  : 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  
 $\alpha$  représenté par  $\mu_2 \times 2^{e_2}$
- Deux possibilités pour la normalisation de l'écriture :  
 $\mu \in [0.5, 1[ \Rightarrow \mu = 0.1 \dots$   
 $\mu \in [1, 2[ \Rightarrow \mu = 1. \dots$
- Représentation en machine avec trois champs

$S$	$E$	$M$
-----	-----	-----

$S$  : 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 :  $S = 1$

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

$$M = 00110 \dots 0$$

Double précision :  $S = 1$

$$E = 3 + 1023 = 1026_{10} = 10 \dots 010_2$$

$$M = 00110 \dots 0$$

# 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 :  
 $S = 0, E = 0 \dots 0, M = 010 \dots 0 \Rightarrow \alpha_{10} = 0.25 \times 2^{-127}$
- Représentation de 0 :  $E = 0 \dots 0, M = 0 \dots 0, S$  indifférent
- Représentation de l'infini :  $E = 1 \dots 1, M \neq 0 \dots 0$
- Représentation de NaN (Not a Number : indéfini) :  
 $E = 1 \dots 1, M = 0 \dots 0$
- 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	?

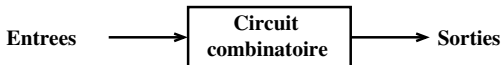
## Table ASCII (2)

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

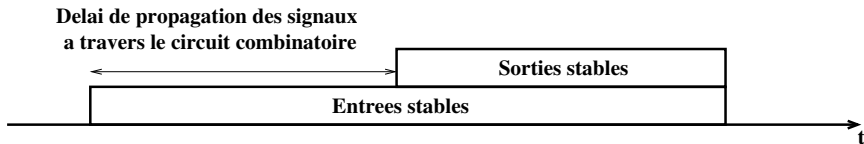


# Circuit combinatoire

- 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

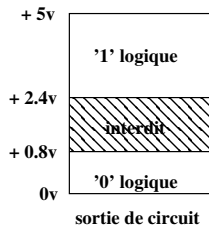
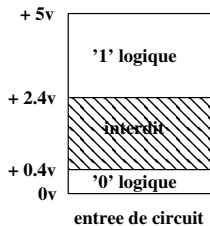


- Exemple de circuit combinatoire : l'additionneur 1 bit

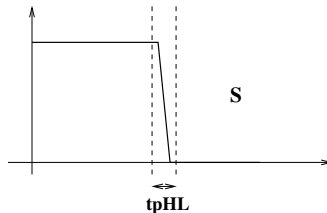
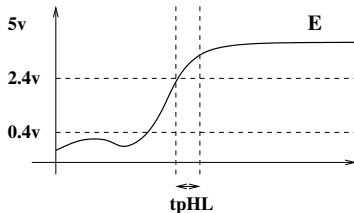


# Bits et signaux électriques

- Assignment 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)**



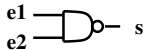
e	s
0	1
1	0

**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

- $a \text{ nand } b = \text{non } (a \text{ et } b)$
- Permet de construire n'importe quelle fonction booléenne

$$\text{non } a = \text{non } (a \text{ et } a)$$

$$= a \text{ nand } a$$

$$a \text{ et } b = \text{non } (\text{non } (a \text{ et } b))$$

$$= (a \text{ nand } b) \text{ nand } (a \text{ nand } b)$$

$$a \text{ ou } b = \text{non } ((\text{non } a) \text{ et } (\text{non } b))$$

$$= (a \text{ nand } a) \text{ nand } (b \text{ nand } b)$$

- 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

$x_1 x_2 \backslash x_3$	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	$f_1 = \bar{x}$	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	$f_2 = \bar{x} \cdot y$	
	0	0	1	1	$f_3 = \bar{x}$	Complément de $x$
	0	1	0	0	$f_4 = x \cdot \bar{y}$	



# Fonctions booléennes simples

- Fonctions de deux variables simples

$xy =$					Expression	Remarque
11	10	01	00			
$f(x, y)$	0	1	0	1	$f_5 = \bar{y}$	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	$f_9 = x \iff y$	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

$x \backslash yz$	00	01	11	10
0		1	1	1
1		1	1	

Monômes premiers :

$z$

$\bar{x} \cdot y$

- 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 irréductible : 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 irrédundante : tous ses monômes sont obligatoires
- Monôme premier essentiel : obligatoire dans toutes les bases de la fonction

xy \ zt	00	01	11	10
00	1	1		
01		1	1	
11			1	1
10			1	1

Monômes premiers :

$x \cdot z, y \cdot z \cdot t, \bar{x} \cdot y \cdot t, \bar{x} \cdot \bar{z} \cdot t, \bar{x} \cdot \bar{y} \cdot \bar{z}$

Monômes essentiels :

$x \cdot z, \bar{x} \cdot \bar{y} \cdot \bar{z}$

Base irrédundante :

$x \cdot z, \bar{x} \cdot \bar{y} \cdot \bar{z}, \bar{x} \cdot y \cdot t$

Base irrédundante :

$x \cdot z, \bar{x} \cdot \bar{y} \cdot \bar{z}, y \cdot z \cdot t, \bar{x} \cdot \bar{z} \cdot t$

# 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 $\varphi$ -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  $\varphi$
- Principe de la minimisation d'une fonction  $\varphi$ -booléenne sur un tableau de Karnaugh
  - ▶ Monômes premiers en considérant les  $\varphi$  à 1
  - ▶ Base irrédundante en considérant les  $\varphi$  à 0

x \ yz	00	01	11	10
	0	1	0	1
0	$\varphi$	1	$\varphi$	
1		$\varphi$	1	

Monômes premiers :

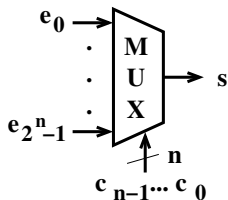
$$z, \bar{x} \cdot \bar{y}$$

Base irrédundante :

$$z$$

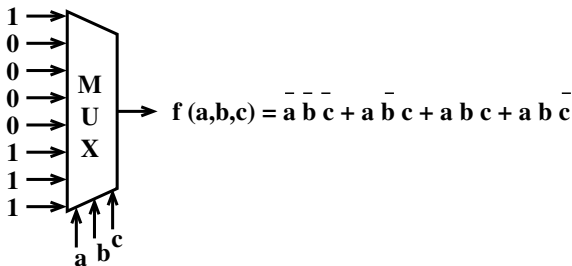
# Le multiplexeur

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



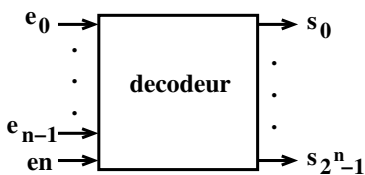
$$s = e_i \text{ avec } i = \sum_{k=0}^n c_k 2^k$$

- Mise en œuvre d'une fonction booléenne



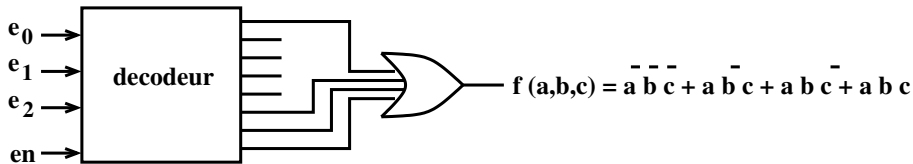
# Le décodeur

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



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

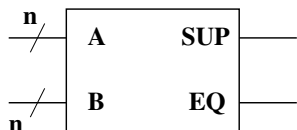
- Mise en œuvre d'une fonction booléenne





# Comparateurs

- Vue générale du circuit : entiers sur  $n$  bits



**$A > B : SUP = 1$**

**$A \geq B : SUP = 1$  ou  $EQ = 1$**

**$A = B : EQ = 1$**

**$A < B : SUP = 0$  et  $EQ = 0$**

**$A \leq B : SUP = 0$**

- Comparateur non signé avec  $n = 2$

$$EQ = (a_1 = b_1) \cdot (a_0 = b_0)$$

$$SUP = a_1 \cdot \overline{b_1} + ((a_1 = b_1) \cdot a_0 \cdot \overline{b_0})$$

- Comparateur non signé avec  $n$  quelconque

$$EQ = (a_{n-1} = b_{n-1}) \cdot (a_{n-2} = b_{n-2}) \cdot \dots \cdot (a_0 = b_0)$$

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

# Comparateurs

- Comparateur signé avec  $n = 2$

$$EQ = (a_1 = b_1) \cdot (a_0 = b_0)$$

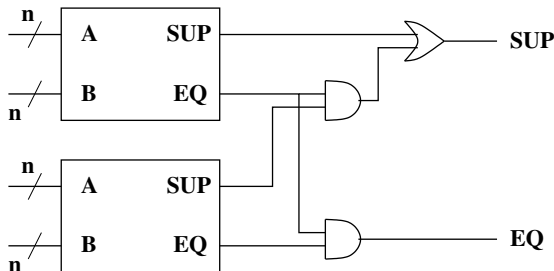
$$SUP = \overline{a_1} \cdot b_1 + ((a_1 = b_1) \cdot a_0 \cdot \overline{b_0})$$

- Comparateur signé avec  $n$  quelconque

$$EQ = (a_{n-1} = b_{n-1}) \cdot (a_{n-2} = b_{n-2}) \cdot \dots \cdot (a_0 = b_0)$$

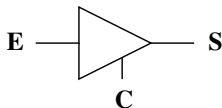
$$SUP = \overline{a_{n-1}} \cdot b_{n-1} + ((a_{n-1} = b_{n-1}) \cdot a_{n-2} \cdot \overline{b_{n-2}} + ((a_{n-2} = b_{n-2}) \cdot a_{n-3} \cdot \overline{b_{n-3}} + \dots ((a_1 = b_1) \cdot a_0 \cdot \overline{b_0}) \dots))$$

- 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

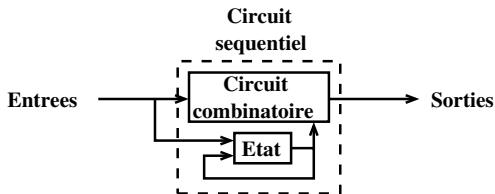
- Une solution pour mettre en œuvre des multiplexeurs

# Exercice

- On souhaite mettre en œuvre la fonction *Majorité* de 5 variables  $x_0, \dots, x_4$ , 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

# Circuits séquentiels

- Circuit tel que l'état des sorties dépend de l'état des entrées et de l'historique du système
- Historique du système : codé par des signaux internes au système
- Ces signaux internes modélisent l'état du système



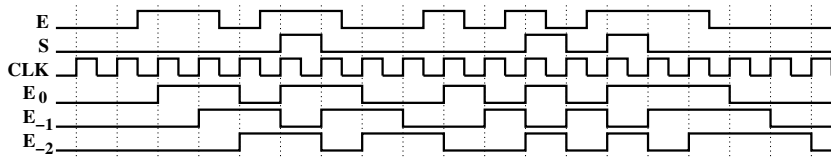
- Exemple : Détection du motif 101 dans une séquence de bits



- Lecture d'un nouveau bit sur E à chaque fois que *CLK* passe de 0 à 1
- $S = 1$  lorsque les trois derniers bits lus sont 101

# Détection du motif 101

- Etat du système : trois derniers bits lus sur  $E$  :  $E_0E_{-1}E_{-2}$
- Exemple d'exécution



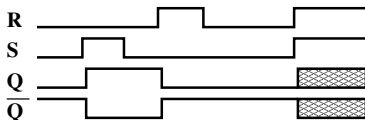
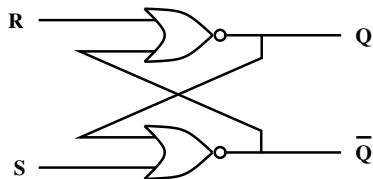
- A chaque fois que  $CLK$  passe de 0 à 1
  - ▶  $E_{-2}$  prend la valeur de  $E_{-1}$
  - ▶  $E_{-1}$  prend la valeur de  $E_0$
  - ▶  $E_0$  prend la valeur de  $E$
- $S$  vaut 1 si  $E_{-2} = 1$  et  $E_{-1} = 0$  et  $E_0 = 1$ ,  $S$  vaut 0 sinon

$$S = E_{-2} \cdot \overline{E_{-1}} \cdot E_0$$

- Mise en œuvre de points de mémorisation pour  $E_0$ ,  $E_{-1}$  et  $E_{-2}$

# Point de Mémorisation élémentaire

- Circuit séquentiel capable de stocker la valeur d'un bit pendant une durée variable
- Le point de mémorisation le plus simple : le bistable RS



Fonction d'évolution

R	S	Effet
0	0	Mémoire
0	1	Mise à 1
1	0	Mise à 0
1	1	Interdit

Commandes à appliquer

$0 \rightarrow 0$      $R = \Phi, S = 0$

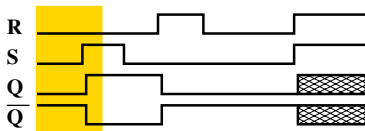
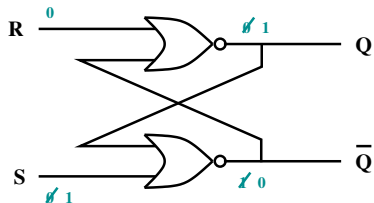
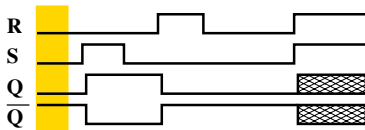
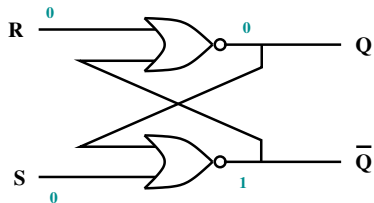
$0 \rightarrow 1$      $R = 0, S = 1$

$1 \rightarrow 0$      $R = 1, S = 0$

$1 \rightarrow 1$      $R = 0, S = \Phi$

# Point de Mémorisation élémentaire

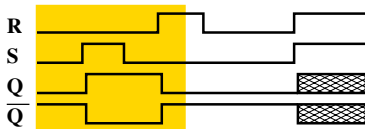
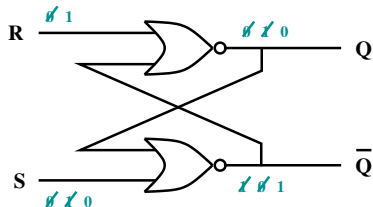
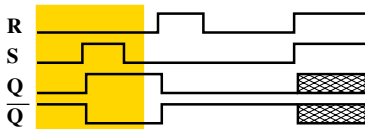
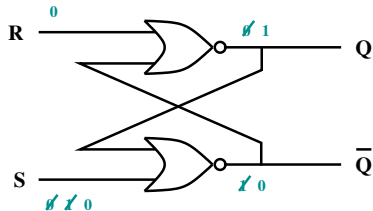
- Rappel :  $x \text{ nor } y = 1$  ssi  $x = 0$  et  $y = 0$





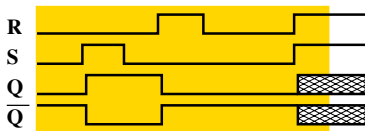
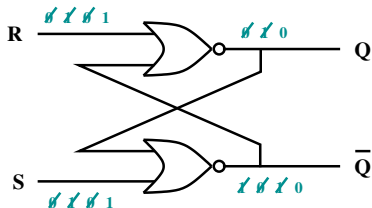
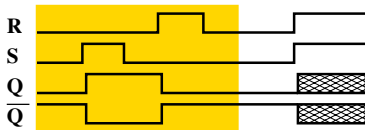
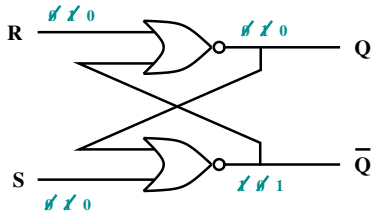
# Point de Mémorisation élémentaire

- Rappel :  $x \text{ nor } y = 1$  ssi  $x = 0$  et  $y = 0$



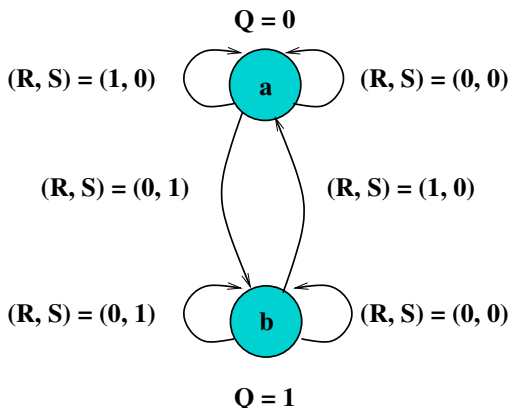
# Point de Mémorisation élémentaire

- Rappel :  $x \text{ nor } y = 1$  ssi  $x = 0$  et  $y = 0$



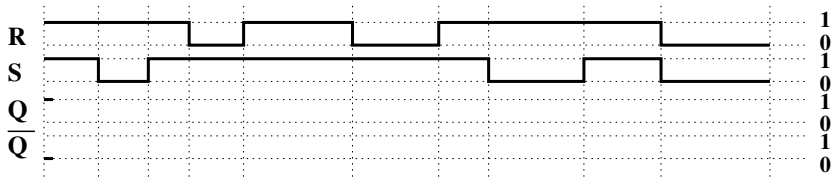
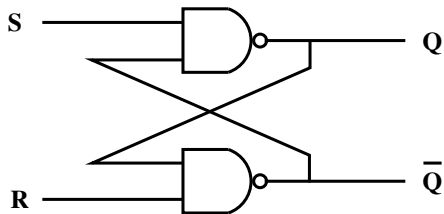
## Graphe d'états du point de mémorisation précédent

- Chaque état ( $a$  ou  $b$ ) correspond à une valeur de la sortie  $Q$
- Chaque flèche est une transition qui est franchie lorsque la condition qui lui est associée est vraie



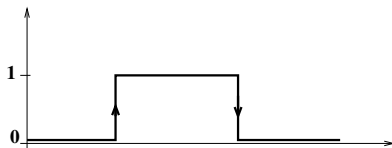
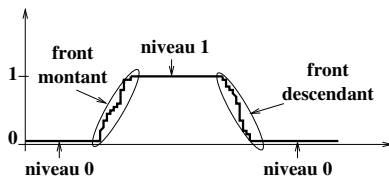
## Exercice

- Donner la fonction d'évolution du bistable RS suivant, construit avec des portes *nand*. On pourra s'aider du chronogramme



# Synchronisation des points de mémorisation

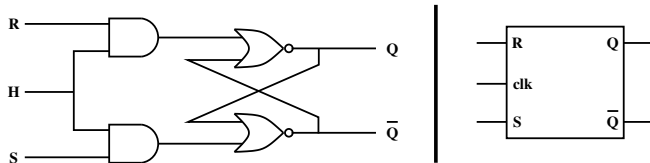
- Latch RS : les variations des sorties ne dépendent que de  $R$  et  $S$  (pas de signal d'horloge)  $\Rightarrow$  circuit séquentiel asynchrone pur
- Circuit synchrone pur : une horloge unique contrôle les instants de prise en compte des entrées
- Signal d'horloge : fronts et niveaux



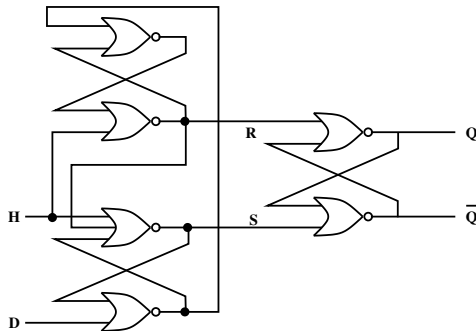
- Synchronisation d'un point de mémorisation
  - ▶ Synchronisation sur un niveau d'horloge : latch actif sur un niveau d'horloge
  - ▶ Synchronisation sur un front d'horloge : bascule active sur un front d'horloge

# Synchronisation d'un point de mémorisation RS

- Latch *RS* actif sur un niveau haut d'horloge

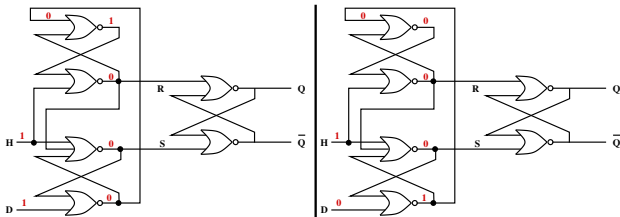


- Bascule active sur un front descendant d'horloge

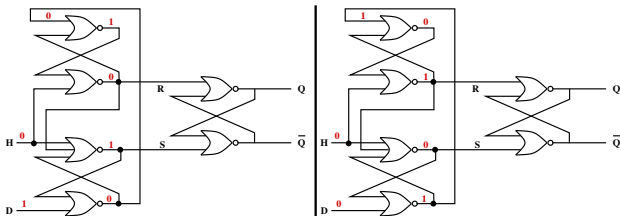


# Illustration du fonctionnement de la bascule

- $H = 1$  : on a nécessairement  $S = 0$  et  $R = 0$



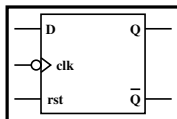
- $H$  passe à 0 :  $S = D$  et  $R = \bar{D}$



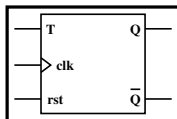
- $H$  reste à 0 :  $D$  sans influence sur  $S$  et  $R$
- $H$  passe à 1 :  $S = 0$  et  $R = 0$

# Les types de bascules

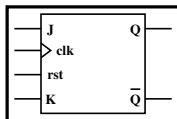
- La bascule  $D$



- La bascule  $T$



- La bascule  $JK$



- ▶  $Q = D$  à chaque front descendant d'horloge

- ▶  $rst$  : remise à 0 asynchrone

- ▶ à chaque front montant d'horloge

- ★ Inversion de  $Q$  si  $T = 1$

- ★ Maintien de  $Q$  si  $T = 0$

- ▶  $rst$  : remise à 0 asynchrone

- ▶ à chaque front montant d'horloge

- ★  $J = 0$  et  $K = 0$  : maintien de  $Q$

- ★  $J = 0$  et  $K = 1$  :  $Q = 0$

- ★  $J = 1$  et  $K = 0$  :  $Q = 1$

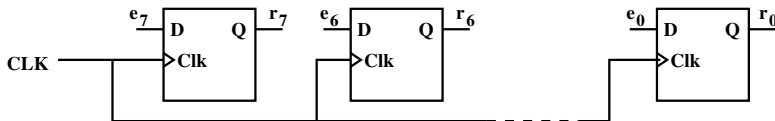
- ★  $J = 1$  et  $K = 1$  : inversion de  $Q$

- ▶  $rst$  : remise à 0 asynchrone

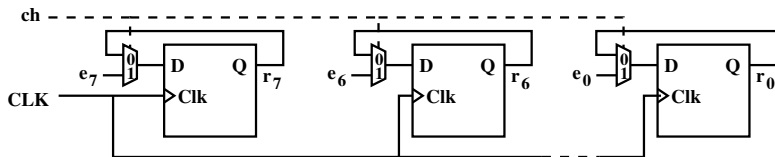


# Le registre : mémorisation d'un ensemble de bits

- Registre de  $n$  bits : ensemble de  $n$  bascules  $D$  synchrones mises en parallèle et partageant la même horloge
- Exemple d'un registre 8 bits : à chaque front montant de l'horloge, les entrées ( $e_7 \dots e_0$ ) sont mémorisées dans le registre ( $r_7 \dots r_0$ )

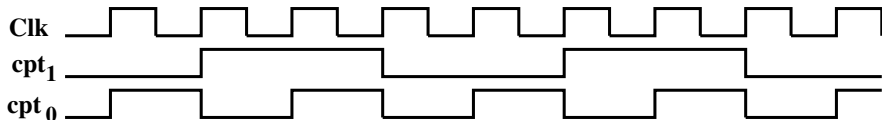
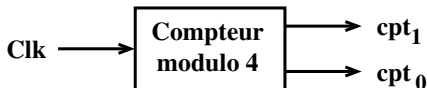


- Contrôle du chargement du registre (pas à chaque front actif d'horloge)  $\Rightarrow$  ajout d'une ligne de sélection



## Mise en œuvre de compteurs

- Compteur modulo  $n$  : incrémentation, à chaque front actif d'une horloge, d'une valeur comprise entre 0 et  $n - 1$  (le successeur de  $n - 1$  est 0)
- Exemple : compteur modulo 4



- Mise en œuvre d'un compteur modulo  $n$  par  $N$  bascules, avec

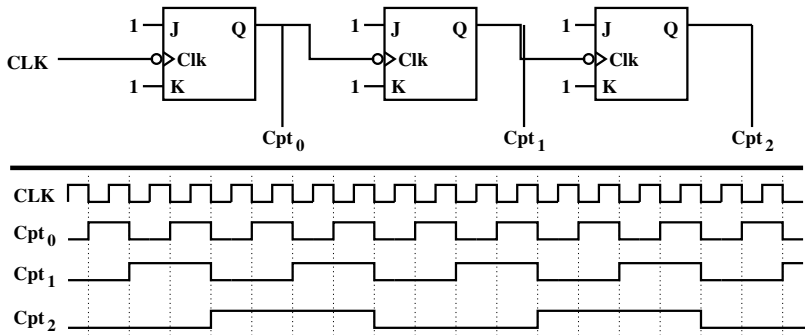
$$2^{N-1} < n \leq 2^N$$

# Mise en œuvre de compteurs

- Deux grandes classes de compteurs
  - ▶ Le compteur asynchrone
    - ★ Les bascules sont commandées par des horloges différentes
  - ▶ Le compteur synchrone
    - ★ Toutes les bascules sont commandées par la même horloge
- Présentation des deux classes de compteurs
- Pour les compteurs synchrones
  - ▶ Cas particulier avec remise à zéro synchrone
  - ▶ Cas général permettant de générer une séquence de nombres dans n'importe quel ordre
- Dans la suite du module, on considèrera des compteurs synchrones

# Mise en œuvre de compteurs asynchrones

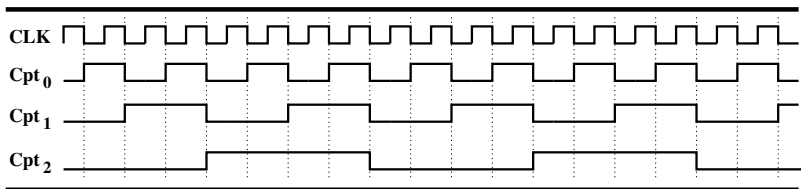
- Pas d'horloge commune entre les bascules
- Exemple : compteur modulo 8 avec des bascules JK actives sur un front descendant d'horloge



- Etats transitoires

# Mise en œuvre de compteurs synchrones

- L'horloge est commune à toutes les bascules
- Exemple : compteur modulo 8 =  $2^3$  avec des bascules D actives sur un front descendant d'horloge



- $Cpt_0$  change de valeur à chaque front descendant d'horloge :  
 $D_0 = \neg Cpt_0$
- $Cpt_1$  change de valeur à chaque fois que  $Cpt_0$  vaut 1 :  
 $D_1 = \neg Cpt_0 * Cpt_1 + Cpt_0 * \neg Cpt_1$
- $Cpt_2$  change de valeur à chaque fois que  $Cpt_0$  et  $Cpt_1$  valent 1 :  
 $D_2 = \neg Cpt_0 * Cpt_2 + \neg Cpt_1 * Cpt_2 + Cpt_0 * Cpt_1 * \neg Cpt_2$



# Mise en œuvre de compteurs synchrones

- Une première solution
  - ▶ Construction d'un signal sclr, qui vaut 0, sauf lorsque le compteur atteint la valeur maximale (par exemple 5 pour un compteur modulo 6)
- Code SHDL d'un compteur modulo 6 avec des bascules D

```
module CompteurModulo6 (rst,clk,en:cpt[2..0])
    sclr = cpt[2]*/cpt[1]*cpt[0]
    cpt[2..0] := d[2..0] on clk, reset when rst, enabled when en
    d[0] = /sclr*/cpt[0]
    d[1] = /sclr*(/cpt[0]*cpt[1]+cpt[0]*/cpt[1])
    d[2] = /sclr*(/cpt[1]*cpt[2]+/cpt[0]*cpt[2]+cpt[1]*cpt[0]*/cpt[2])
end module
```

# Mise en œuvre de compteurs synchrones

- Une deuxième solution
  - ▶ Construction d'une table de vérité avec, pour chaque valeur du compteur, la valeur suivante et les commandes qui en découlent
- Exemple pour un compteur modulo 6 avec des bascules D

$Cpt_2$	$Cpt_1$	$Cpt_0$	$Cpt'_2$	$Cpt'_1$	$Cpt'_0$	$D_2$	$D_1$	$D_0$
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	0	0	0	0	0	0
1	1	0	$\phi$	$\phi$	$\phi$	$\phi$	$\phi$	$\phi$
1	1	1	$\phi$	$\phi$	$\phi$	$\phi$	$\phi$	$\phi$

$$d[2] = cpt[1] * cpt[0] + cpt[2] * /cpt[0]$$

$$d[1] = cpt[1] * /cpt[0] + /cpt[2] * /cpt[1] * cpt[0]$$

$$d[0] = /cpt[0]$$



## Extension aux générateurs de séquences

- Circuit qui génère en boucle une séquence de nombres
- Exemple avec des bascules JK : séquence 3, 0, 2, 1
- Nombres codables sur 2 bits  $\Rightarrow$  2 bascules
- Rappel bascule JK
  - ▶  $0 \rightarrow 0 : J = 0, K = \phi$
  - ▶  $0 \rightarrow 1 : J = 1, K = \phi$
  - ▶  $1 \rightarrow 0 : J = \phi, K = 1$
  - ▶  $1 \rightarrow 1 : J = \phi, K = 0$

$Cpt_1$	$Cpt_0$	$Cpt'_1$	$Cpt'_0$	$J_1$	$K_1$	$J_0$	$K_0$
0	0	1	0	1	$\phi$	0	$\phi$
0	1	1	1	1	$\phi$	$\phi$	0
1	0	0	1	$\phi$	1	1	$\phi$
1	1	0	0	$\phi$	1	$\phi$	1

$$J_1 = K_1 = 1$$

$$J_0 = K_0 = Cpt_1$$

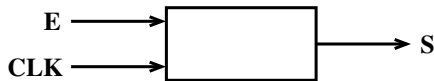
# Exercice

- Construire avec des bascules T un circuit qui génère en boucle la séquence 1, 3, 2, 0

# Synthèse d'un circuit séquentiel synchrone

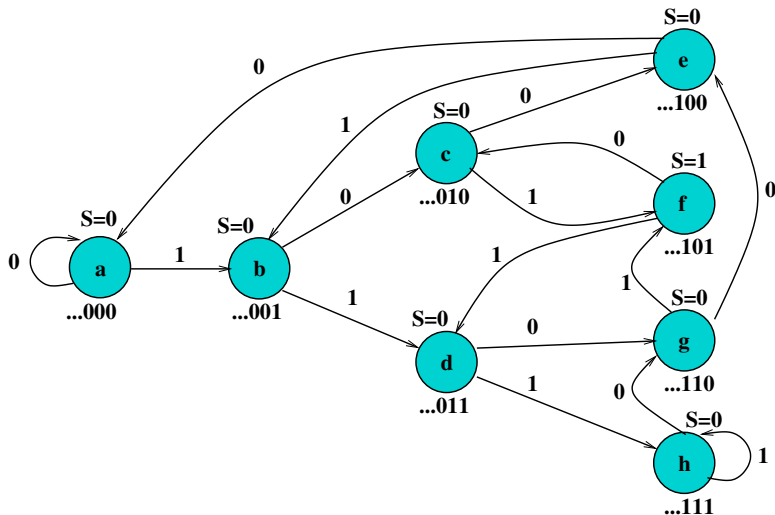
- ➊ On dessine le graphe d'états
- ➋ On en déduit la table des transitions
- ➌ On simplifie la table des transitions
- ➍ On en déduit le nombre de bascules
- ➎ On assigne un code à chaque état
- ➏ On instancie la table des transitions (on remplace les états par leur code)
- ➐ On choisit le type de bascule le plus approprié
- ➑ On calcule les entrées des bascules et les sorties du circuit

# graphe d'états d'un circuit séquentiel synchrone



- A quel moment par rapport à l'horloge la sortie prend-elle sa valeur ?
- Deux possibilités
  - ▶ La sortie ne dépend que de l'état interne du circuit  $\Rightarrow$  graphe de Moore
  - ▶ La sortie dépend de l'état interne du circuit et des entrées  $\Rightarrow$  graphe de Mealy

# Graphe de Moore non simplifié du détecteur de séquences

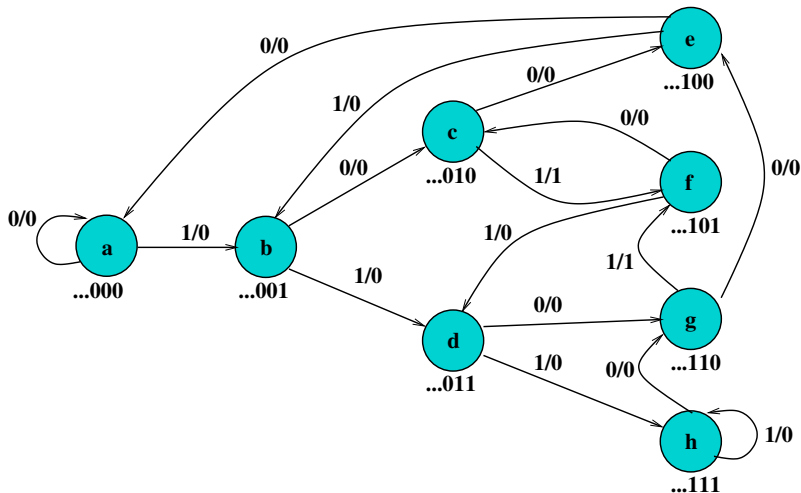


## Table de transition associée au graphe de Moore

état courant	E	état suivant
a	0	a
a	1	b
b	0	c
b	1	d
c	0	e
c	1	f
d	0	g
d	1	h
e	0	a
e	1	b
f	0	c
f	1	d
g	0	e
g	1	f
h	0	g
h	1	h

état	S
a	0
b	0
c	0
d	0
e	0
f	1
g	0
h	0

# Graphe de Mealy non simplifié du détecteur de séquences



## Table de transition associée au graphe de Mealy

état courant	E	S	état suivant
a	0	0	a
a	1	0	b
b	0	0	c
b	1	0	d
c	0	0	e
c	1	1	f
d	0	0	g
d	1	0	h
e	0	0	a
e	1	0	b
f	0	0	c
f	1	0	d
g	0	0	e
g	1	1	f
h	0	0	g
h	1	0	h



# Simplification des tables de transition

- Fusion des états équivalents
- Deux états d'une machine de Moore sont équivalents ssi
  - ▶ Ils sont associés aux mêmes valeurs des sorties
  - ▶ Ils ont les mêmes états suivants pour chaque valeur possible des entrées
- Deux états d'une machine de Mealy sont équivalents ssi
  - ▶ Ils ont les mêmes états suivants et sorties pour chaque valeur possible des entrées

## Exemple du détecteur de séquence(Moore)

- Les états **a** et **e** sont équivalents
  - ▶ La valeur de la sortie *S* y est 0
  - ▶ L'état suivant est **a** lorsque l'entrée *E* vaut 0, **b** lorsqu'elle vaut 1
- Les états **c** et **g** sont équivalents
- Les états **d** et **h** sont équivalents
- On obtient une table de transition plus compacte

état courant	E	état suivant
a	0	a
a	1	b
b	0	c
b	1	d
c	0	a
c	1	f
d	0	c
d	1	d
f	0	c
f	1	d

état	S
a	0
b	0
c	0
d	0
f	1

# Exemple du détecteur de séquence (Moore)

- Les états **a** et **d** sont équivalents

état courant	E	état suivant
a	0	a
a	1	b
b	0	c
b	1	b
c	0	a
c	1	f
f	0	c
f	1	b

état	S
a	0
b	0
c	0
f	1

## Exemple du détecteur de séquence (Mealy)

- Les états **a** et **e** sont équivalents
- Les états **b** et **f** sont équivalents
- Les états **c** et **g** sont équivalents
- Les états **d** et **h** sont équivalents

état courant	E	S	état suivant
a	0	0	a
a	1	0	b
b	0	0	c
b	1	0	d
c	0	0	a
c	1	1	b
d	0	0	c
d	1	0	d

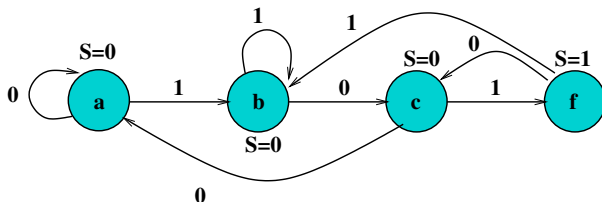
# Exemple du détecteur de séquence (Mealy)

- Les états **b** et **d** sont équivalents

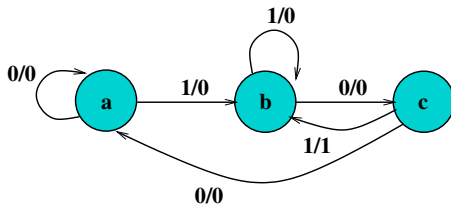
état courant	E	S	état suivant
a	0	0	a
a	1	0	b
b	0	0	c
b	1	0	b
c	0	0	a
c	1	1	b

# Graphes simplifiés du détecteur de séquence

- Graphe de Moore



- Graphe de Mealy



# Synthèse du détecteur de séquence, version Moore

- Nombre de bascules pour mémoriser l'état courant
  - ▶ 4 états  $\Rightarrow$  2 bascules
- Codage des états
  - ▶ essayer de choisir un codage qui simplifie le circuit, en appliquant par exemple les règles heuristiques suivantes
    - ★ Rendre adjacents les états de départ qui ont même état d'arrivée dans le graphe
    - ★ Rendre adjacents les états d'arrivée qui ont même état de départ dans le graphe
    - ★ Rendre adjacents les états qui ont même sorties
  - ▶ Une assignation possible

état	a	b	c	f
XY	00	11	01	10

# Synthèse du détecteur de séquence, version Moore

- Table de transition instanciée

état courant	E	état suivant
00	0	00
00	1	11
11	0	01
11	1	11
01	0	00
01	1	10
10	0	01
10	1	11

état	S
00	0
11	0
01	0
10	1

- Choix des bascules

- ▶ Par exemple une bascule D pour X et une bascule JK pour Y



# Synthèse du détecteur de séquence, version Moore

- Calcul des entrées des bascules et de la sorite du circuit

état courant	E	DX	JY	KY	état suivant
00	0	0	0	$\phi$	00
00	1	1	1	$\phi$	11
11	0	0	$\phi$	0	01
11	1	1	$\phi$	0	11
01	0	0	$\phi$	1	00
01	1	1	$\phi$	1	10
10	0	0	1	$\phi$	01
10	1	1	1	$\phi$	11

état	S
00	0
11	0
01	0
10	1

- On obtient
  - ▶  $DX = E$
  - ▶  $JY = E + X$  et  $KY = /X$
  - ▶  $S = X * /Y$

# Un exemple de circuit séquentiel complexe

- Calcul du PGCD de 2 entiers  $x$  et  $y$
- Construction d'un circuit exécutant l'algorithme suivant

**Tantque**  $x \neq y$  **faire**

**Si**  $x > y$  **alors**

$x \leftarrow x - y;$

**Sinon**

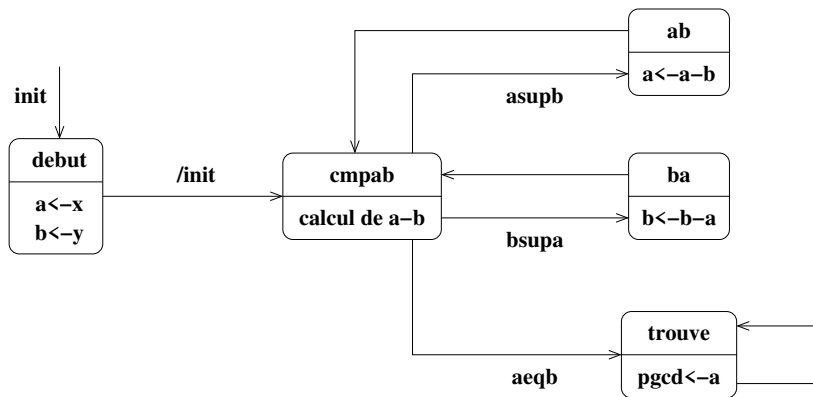
$y \leftarrow y - x;$

**Finsi;**

**Fintantque**

$\text{pgcd} \leftarrow x;$

# Graphe d'état



# Composants du circuit

- Description des états en SHDL

debut := init on clk reset when rst

cmpab := /init\*(debut+ab+ba) on clk reset when rst

ab := cmpab\*asupb on clk reset when rst

ba := cmpab\*bsupa on clk reset when rst

trouve := /init\*(cmpab\*aeqb+trouve) on clk reset when rst

- Un soustracteur utilisé dans 3 états

- ▶ Etat *cmpab* : calcul de  $a - b$  et génération de *asupb*, *bsupa* et *aeqb*
- ▶ Etat *ab* : calcul de  $a - b$ , résultat dans *a*
- ▶ Etat *ba* : calcul de  $b - a$ , résultat dans *b*

- Les valeurs courantes de *a* et de *b* sont mémorisées dans des registres

# Module SHDL

```
module pgcd(clk,rst,init,x[31..0],y[31..0]:fini,pgcd[31..0])
  debut := init on clk reset when rst
  cmpab := /init*(debut+ab+ba) on clk reset when rst
  ab := cmpab*asupb on clk reset when rst
  ba := cmpab*bsupa on clk reset when rst
  trouve := /init*(cmpab*aeqb+trouve) on clk reset when rst
  a[31..0] := init*x[31..0]+/init*(ab*s[31..0]+/ab*a[31..0]) on clk
    reset when rst
  b[31..0] := init*y[31..0]+/init*(ba*s[31..0]+/ba*b[31..0]) on clk
    reset when rst
  e1[31..0] = a[31..0]*(cmpab+ab)+b[31..0]*ba
  e2[31..0] = a[31..0]*ba+b[31..0]*(cmpab+ab)
  AS32(e1[31..0],e2[31..0],1,s[31..0],asupb,bsupa,aeqb)
  fini = trouve
  pgcd[31..0] = trouve*a[31..0]
end module
```