

# ARCHITECTURE DES ORDINATEURS

# CHAPITRE 1

## INTRODUCTION



# Objectifs du cours

---

---

Comprendre l'organisation interne d'un ordinateur, les principes de son fonctionnement et les problèmes principaux à résoudre.

## Pourquoi ?

- Nécessaire à la bonne compréhension d'autres cours
  - de Systèmes d'exploitation à Algorithmique
- Étude de problèmes et solutions généraux en informatique
  - de la modularité aux questions de performance

# Quelques cours et questions connexes

---

---

- **Systèmes d'exploitation**

- Comment un système d'exploitation fait-il pour exécuter plusieurs programmes "en même temps" ?

- **Programmation concurrente**

- Mes deux threads incrémentent chacun 1 000 fois la variable grandTotal, mais j'obtiens 1 534 au final. Pourquoi ?

- **Programmation parallèle**

- Je suis passé d'une machine mono- à quadri-processeurs, et mon calcul de nombres premiers ne va pas plus vite !?

# Quelques cours et questions connexes

---

---

- **Programmation réseau**

- Je recopie mon tableau de mesures de la machine Jupiter à la machine Venus et les valeurs ne sont plus les mêmes !?

- **Compilation**

- Quelle séquence d'instructions machine générer pour la recopie d'une chaîne de caractère ?

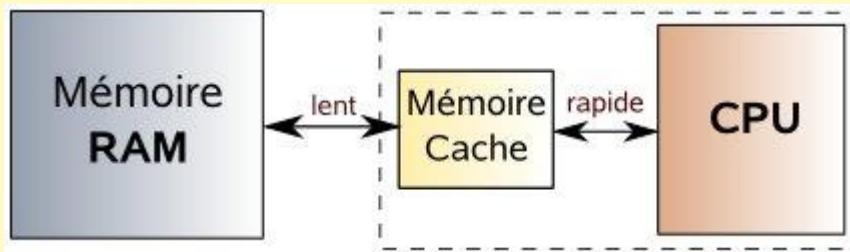
- **Algorithmique**

- Pourquoi le *quick sort* est généralement meilleur que le *heap sort* alors qu'ils ont la même complexité algorithmique moyenne  $O(n \log_2 n)$  ?

# Quelques problèmes et solutions généraux

---

- Diminuer la latence d'accès :
  - à la mémoire, au disque dur, à un serveur, etc.
  - solution : utiliser un cache, i.e. une mémoire locale à accès rapide



- Augmenter la vitesse de traitement :
  - des instructions, des calculs d'un programme, etc.
  - solution : paralléliser, ou pipeliner les traitements
- Etc ...

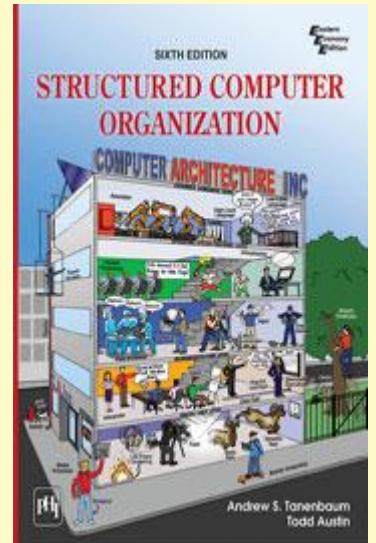
# Références

---

---

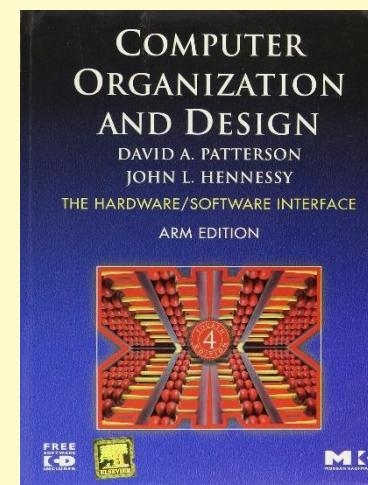
- Livres :

- Structured Computer Organization. *Andrew S. Tanenbaum*
- Computer Organization and Design. *John Hennessy, David Patterson*



- Documentation constructeur :

- Architectures processeurs Intel ® 64 et IA-32
- Manuel du développeur Motorola 68000



# AGENDA DU CHAPITRE

---

---

1  
Histoire de  
l'informatique

3  
La machine de  
von Neumann

4  
codage de  
l'information

2  
Structure d'un  
ordinateur

5  
Le langage des  
ordinateurs

# AGENDA DU CHAPITRE

---

---

1

Histoire de  
l'informatique

3

La machine de  
von Neumann

4

codage de  
l'information

2

Structure d'un  
ordinateur

5

Le langage des  
ordinateurs

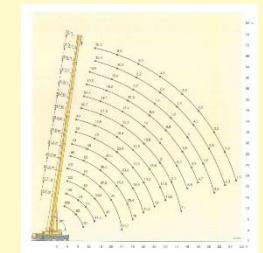
# Avant notre ère

---

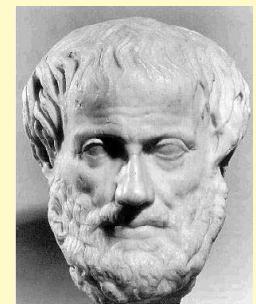
- 3000 : période de l'empereur Chinois **Fou-Hi** dont le symbole magique, **l'octogone à trigramme** contient les 8 premiers nombres représentés sous forme binaire par des traits interrompus ou non : 000 001 010 011 etc ...



- 2000 : apparition au Moyen Orient du premier « outil » de calcul : **l'abaque**



- 1000 : invention du **boulier** en Chine

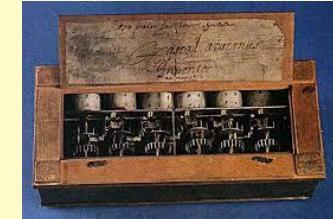


- 300 : le philosophe Grec **Aristote** définit dans son oeuvre ce qu'est la **logique** (ou Organon)

# Les premières calculatrices mécaniques

---

- 1623 : Wilhelm Schickard invente une **horloge calculante**
- 1632 : L'Anglais Oughtred invente **la règle à calcul**
- 1642 : Pascal met au point, pour aider son père collecteur des impôts à Rouen, **la Pascaline**
- 1679 : Gottfried Wilhelm von Leibniz découvre et met au point une arithmétique binaire
- 1694 : Leibniz invente une machine à calculer dérivée de la Pascaline mais capable de traiter les multiplications et divisions
- 1820 : Charles-Xavier Thomas de Colmar invente **l'arithmomètre** sur la base de la machine de Leibniz

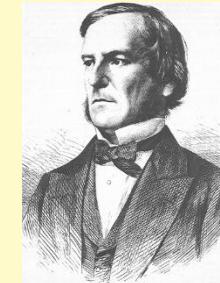


# Les grands noms

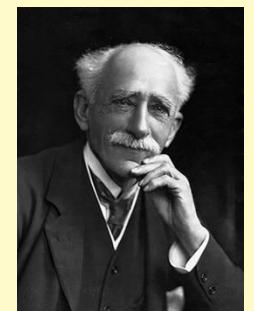
---

---

- 1854 : **George Boole** publie un ouvrage dans lequel il démontre que tout processus logique peut être décomposé en une suite d'opérations logiques appliquées sur deux états



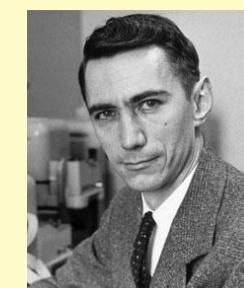
- 1904 : Invention du premier tube à vide, la diode par **John Ambrose Fleming**



- 1937 : **Alan M. Turing** publie un document sur les nombres calculables et invente la Machine de Turing (puis le test de Turing en 1950)



- 1938 : Thèse de **Claude E. Shannon** qui le premier fait le parallèle entre les circuits électriques et l'algèbre Booléenne. Il définit le chiffre binaire : bit (**BInary digiT**)

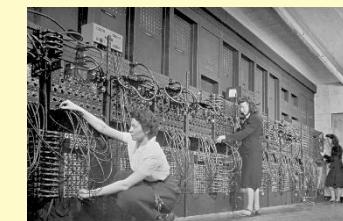
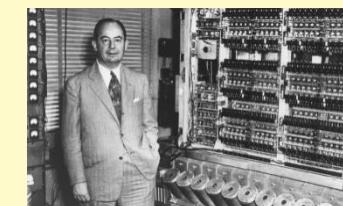
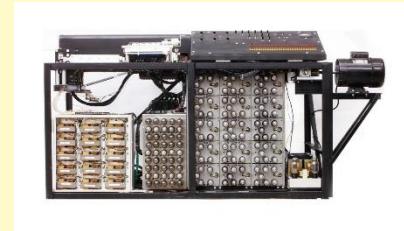


# Les premiers ordinateurs

---

---

- **1941**: création du calculateur binaire **ABC** par John Atanasoff et Clifford Berry - premier calculateur à utiliser l'algèbre de Boole
- **1941** : Konrad Zuse met au point le **Z3**, le premier calculateur avec programme enregistré (premier véritable ordinateur)
- **1945** : John Von Neumann décrit l'**EDVAC** (Electronic Discrete Variable Automatic Computer) ⇒ architecture Von Neumann
- **1946**: Création de l'**ENIAC** (Electronic Numerical Integrator and Computer) par P. Eckert et J. Mauchly



# Les premiers langages de programmation

---

---

- **1950** : invention de **l'assembleur** par Maurice V. Wilkes de l'université de Cambridge. Avant, la programmation s'effectuait directement en binaire.
- **1955** : IBM lance l'IBM 704 développé par Gene Amdahl, machine sur laquelle sera développé le langage **FORTRAN**.
- **1957** : Création du premier langage de programmation universel, le **FORTRAN** (FORmula TRANslator) par John Backus d'IBM.
- **1964** : Thomas Kurtz et John Kemeny créent le langage **BASIC** (Beginner's All-purpose Symbolic Instruction Code) au Dartmouth College pour leurs étudiants.
- **1968** : Création du langage **PASCAL** par Niklaus Wirth.

# D'autres dates et personnages importants

- **1964** : Création du code **ASCII** (American Standard Code for Information Interchange), normalisé en 1966 par l'ISO

ASCII Code Chart																
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	!	"	#	\$	%	&	*	( )	*	+/-	,	-	.	/		
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	> ?	
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\]	^	_	
6	-	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	}	~	DEL	

- **1965** : Gordon Moore écrit la première « **loi de Moore** » disant que la complexité des circuits intégrés doublera tous les ans



- **1969** : Ken Thompson et Dennis Ritchie mettent au point **UNIX** sur un Dec PDP 7
- ...



Plus d'informations sur <http://histoire.info.online.fr>

# AGENDA DU CHAPITRE

---

---

1  
Histoire de  
l'informatique

3  
La machine de  
von Neumann

4  
codage de  
l'information

2  
Structure d'un  
ordinateur

5  
Le langage des  
ordinateurs

# Types d'ordinateurs

---

---

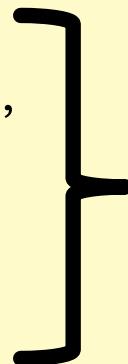
- Ordinateurs visibles :

- Ordinateurs personnels et stations de travail (portables et fixes),
- Ordinateurs « serveurs »,
- Ordinateurs de contrôle de processus,
- Super calculateurs.

- Ordinateurs invisibles :

- Véhicules (automobiles, trains, bateaux, avions, fusées, ...),
- Communications,
- Électro-ménager,
- Horlogerie,
- ...

De loin les plus nombreux !



# La machine perçue par l'utilisateur

---

Interactions avec la machine au travers des périphériques :

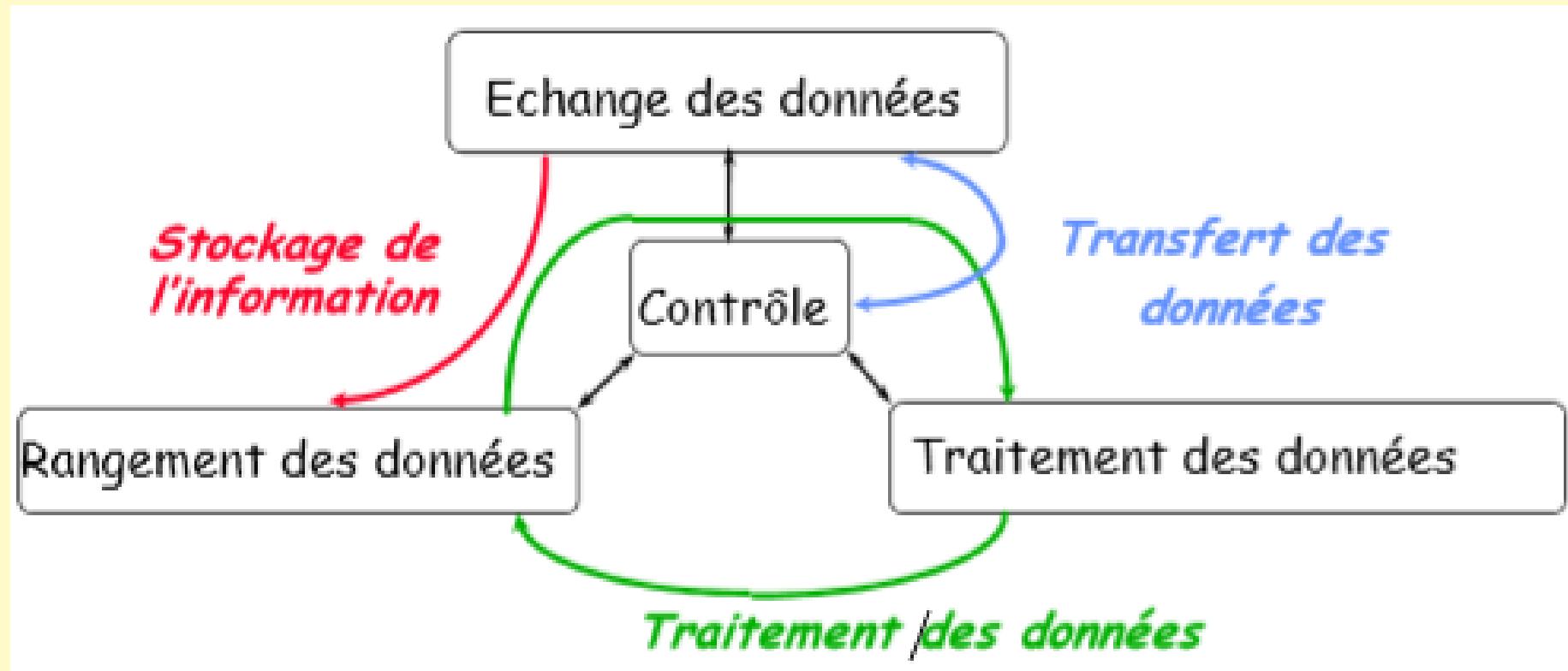
- clavier,
- souris,
- écran,
- imprimante,
- lecteur DVD,
- etc...



# Services rendus par un ordinateur

---

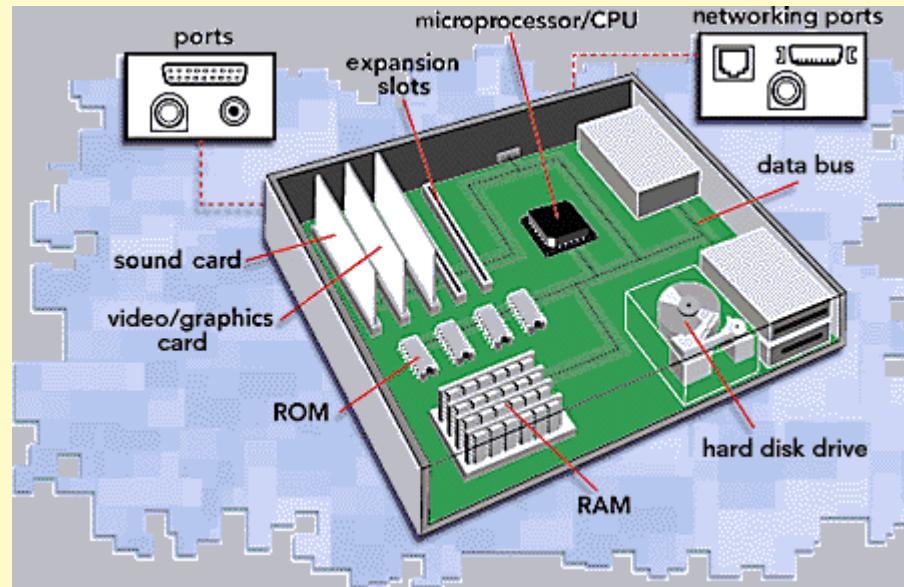
---



# Décomposition matérielle d'un ordinateur

---

- Un ordinateur est constitué de plusieurs parties :
  - souris,
  - écran,
  - clavier,
  - unité centrale,
  - lecteur de disquettes ...
- A l'intérieur de l'unité centrale :
  - une carte mère,
  - une carte vidéo,
  - des disques ....
- Sur la carte mère :
  - un microprocesseur,
  - de la mémoire (ROM, RAM) ...
- Dans le microprocesseur : la puce.



# Architecture et organisation

---

---

- Toute la famille x86 d'Intel partage la même architecture de base.
- Toute la famille System/370 d'IBM partage la même architecture de base.
- Mais l'organisation est différente d'une version sur l'autre.



# Les cinq générations d'ordinateurs

---

---

Génération	Période	Technologie	Vitesse (nombre opérations/s)
1	1946 - 1957	Tube à vide	40 000
2	1958 - 1964	Transistor	200 000
3	1965 - 1971	SSI/MSI	1 000 000
4	1972 - 1977	LSI	10 000 000
5	Post 1978	VLSI	100 000 000

# Période 1945-1958

---

---

- Ordinateurs spécialisés, exemplaires uniques.
  - Machines volumineuses et peu fiables.
  - Technologie à lampes, relais, résistances.
  - $10^4$  éléments logiques – programmation par cartes perforées.
- 
- Représentant : ENIAC (1946) pour l'étude de faisabilité de la bombe H.



# Période 1958-1964

---

---

- Usage général, machines fiables.
- Technologie à transistors.
- $10^5$  éléments logiques.
- Premiers langages de programmation évolués (COBOL, FORTRAN, LISP).
- Représentant : mini ordinateur DEC PDP-1 (1961).

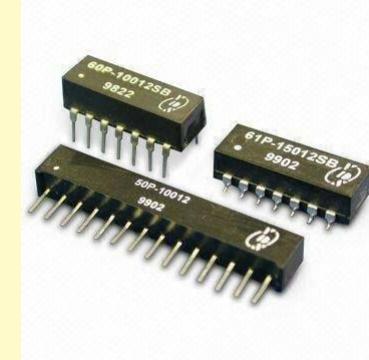


# Période 1965-1971

---

---

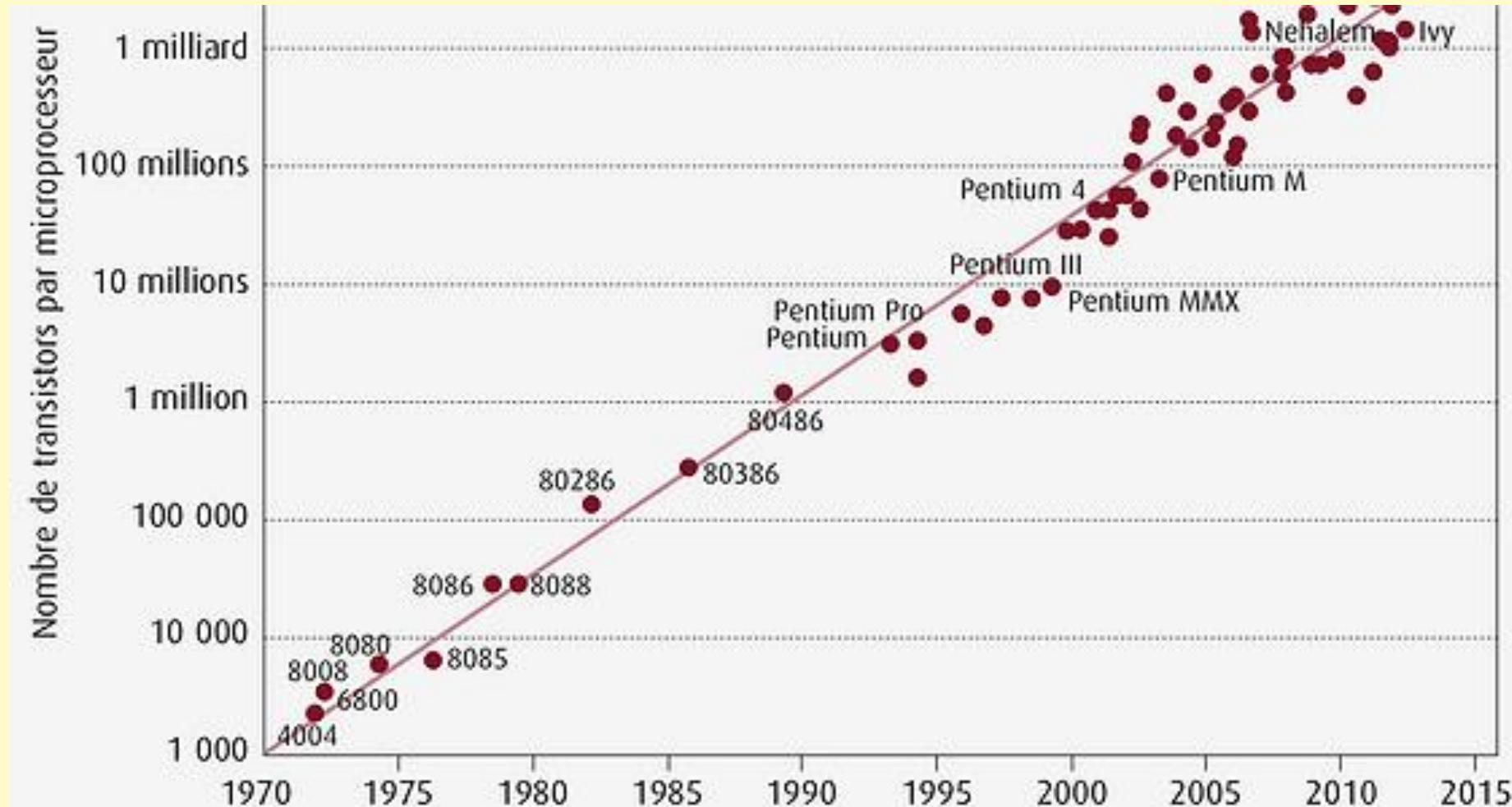
- Technologie des circuits intégrés (S/MSI Small/medium scale integration).
  - $10^6$  éléments logiques.
  - Avènement du système d'exploitation complexe.
  - UNIX, Pascal, Basic, ...
  - CISC (*Complex Instruction Set Computer*).
- 
- **Loi de Moore** : le nombre de transistors intégrables sur une seule puce double chaque année.



# Loi de Moore

---

---



# Période 1972-1977

---

---

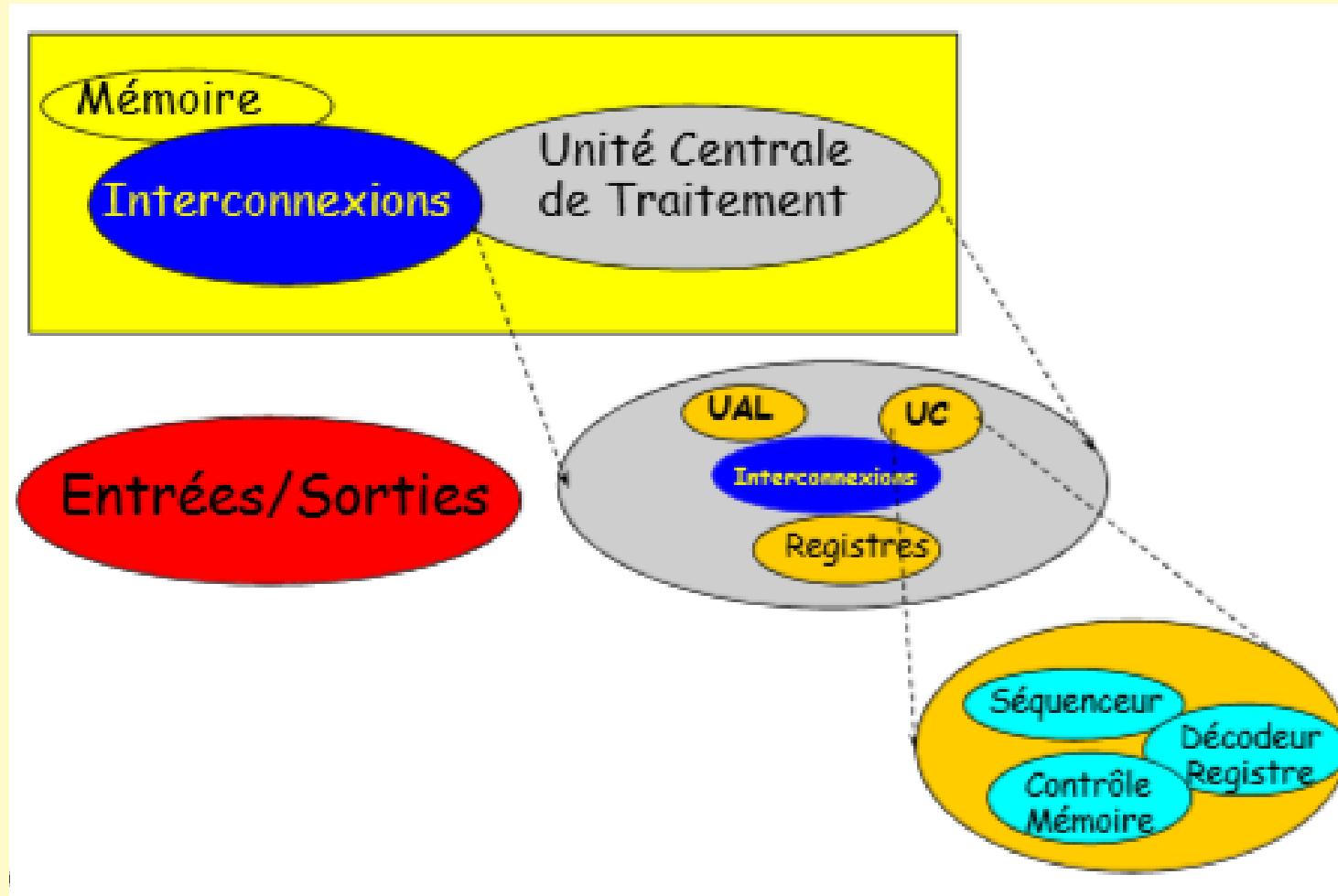
- Technologie LSI (large SI).
  - $10^7$  éléments logiques.
  - Avènement de réseaux de machines.
  - Traitement distribué/réparti.
- 
- 1971 : premier microprocesseur 4004 de INTEL, toutes les composantes de la CPU sont réunies sur une même puce.



# Décomposition fonctionnelle d'un ordinateur

---

---



# AGENDA DU CHAPITRE

---

---

1  
Histoire de  
l'informatique

3  
La machine de  
von Neumann

4  
codage de  
l'information

2  
Structure d'un  
ordinateur

5  
Le langage des  
ordinateurs

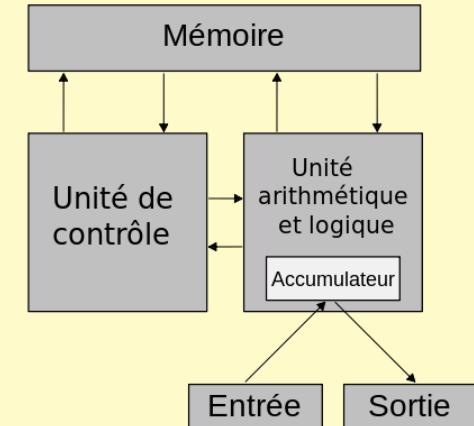
# La machine de Von Neumann

---

---

Modèle de machine universelle possédant :

- une **mémoire** : contient instructions et données,
- une **unité arithmétique et logique (ALU)** : effectue les calculs,
- une **unité d'entrées/sorties** : échange d'informations (I/O) avec les périphériques,
- une **unité de commande** : contrôle (UC).



# Fonctionnement schématique

---

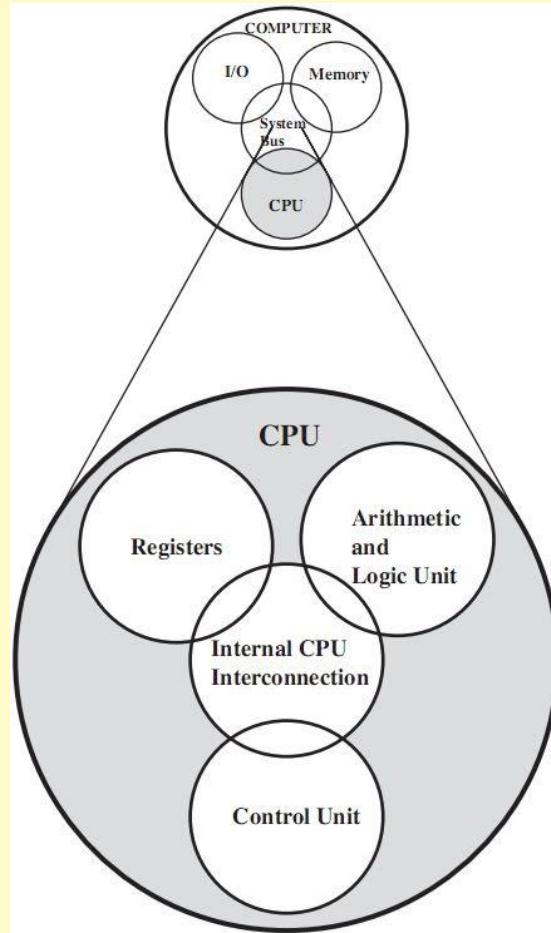
---

- L'Unité de Commande :
  1. extrait une instruction de la mémoire,
  2. analyse l'instruction,
  3. recherche dans la mémoire les données concernées,
  4. déclenche l'opération adéquate sur l'ALU ou l'E/S,
  5. range le résultat dans la mémoire.

# La machine de Von Neumann

---

---



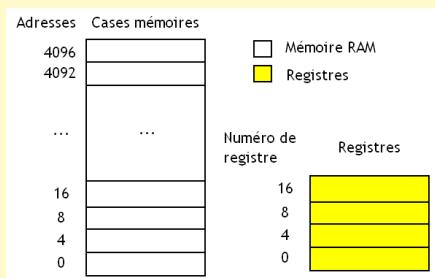
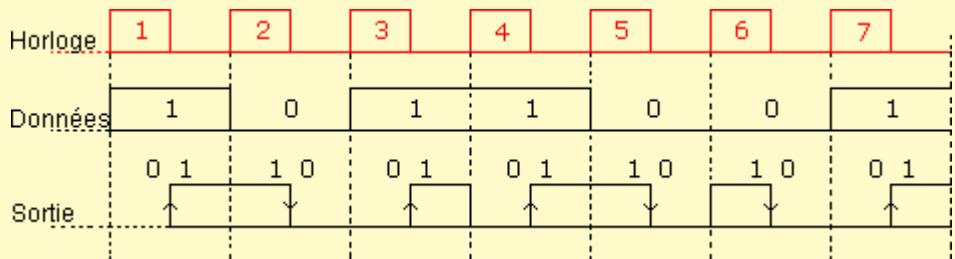
# Aperçu

---

---

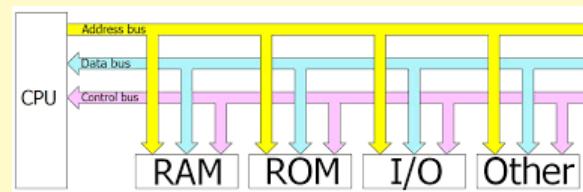
- Dispositifs de base :

- signal et chronogramme,
- horloge,
- registres,
- bus.



- Unités fonctionnelles :

- mémoire,
- ALU,
- E/S,
- UC.

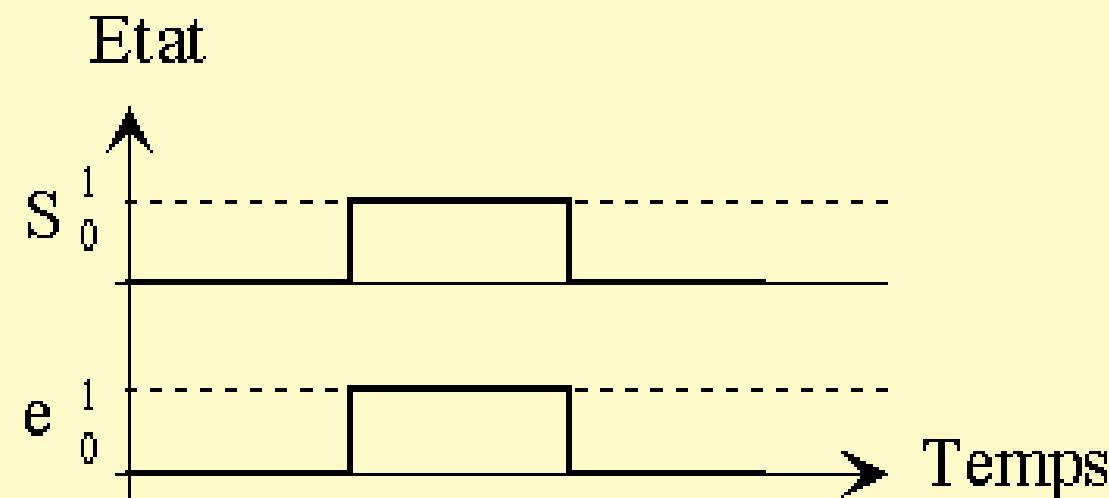


# Signal et chronogramme

---

---

- Un **signal** est une grandeur discrète appartenant à [0,1].
- Un **chronogramme** est la représentation graphique d'un signal évoluant dans le temps.

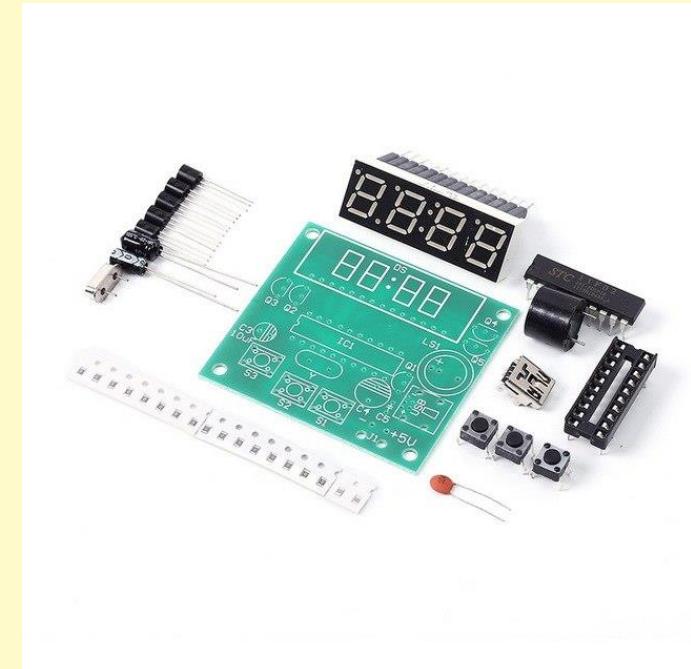


# Horloge

---

---

- Signal de séquences régulières de 0 et de 1.
- Une séquence est appelée **cycle**.
- Exemple: une fréquence d'horloge de 500MHz donne des cycles de 2 nanosecondes.
- Synchronise l'ensemble des dispositifs.

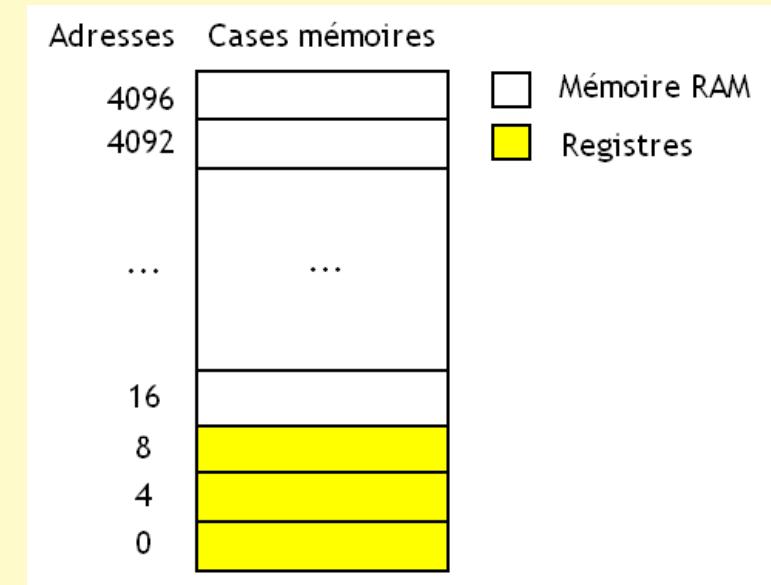


# Registres

---

---

- Eléments de mémoire rapide internes à la CPU.
- Un signal commande la mémorisation :
  - chargement sur niveau,
  - chargement sur front.

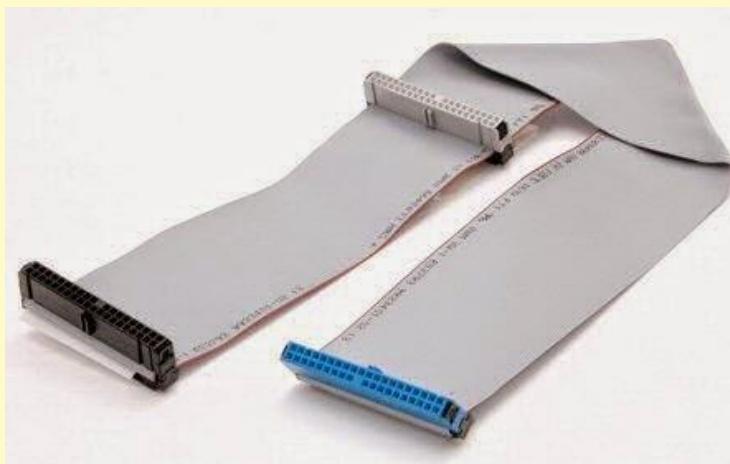


# Bus

---

---

- Ensemble de fils électriques sur lesquels transitent les signaux.
- Relie les unités entre elles.
- **Largeur du bus :**
  - nombre de fils constituant le chemin,
  - nombre de signaux pouvant être envoyés en même temps.



# Mémoire

---

---

- Vecteur dont chaque composante est accessible par une **adresse**.
- Opérations permises :
  - Lecture,
  - Écriture.
- **Mot** = unité d'information accessible en une seule opération de lecture.

Adresse	Valeur
0	145
1	3.8028322
2	0.827551
3	3901930
...	...
3 448 765 900 126 (et des poussières)	940.5118

# Fonctionnement

---

---

1. L'UC inscrit l'adresse d'une cellule dans un registre d'adresse (RA).
2. L'UC demande une opération.
3. Les échanges se font via un registre de mot (RM).

## Lecture

$RA \leftarrow \text{adresse}$   
 $RM \leftarrow \text{mémoire}[RA]$

## Ecriture

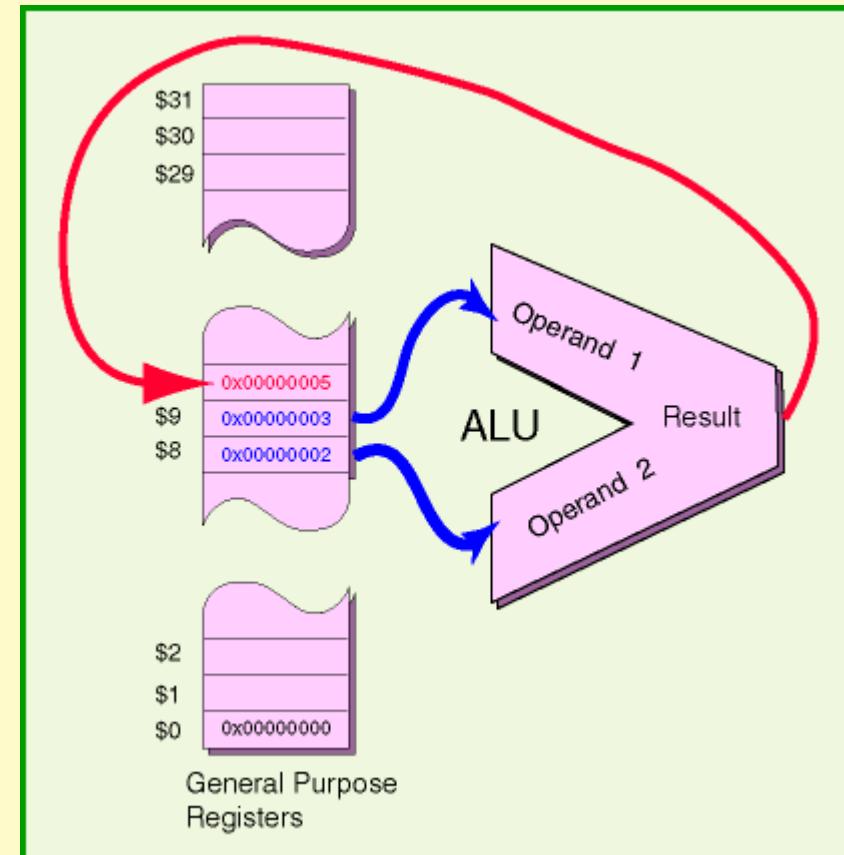
$RM \leftarrow \text{valeur}$   
 $RA \leftarrow \text{adresse}$   
 $\text{mémoire}[RA] \leftarrow RM$

# ALU

---

---

- Fonction à 3 paramètres :
  - 1 opération
  - 2 arguments
  - 1 résultat
- Nécessite un/des registre(s) de mémorisation :
  - entrées, sortie
  - résultats intermédiaires



# Unité Entrées / Sortie

---

---

- Sert d'interface avec les périphériques.
- Les opérations associées sont fonction du périphérique.
- Fonctionnement similaire à la mémoire :
  - un registre mémorisant l'adresse du périphérique (Registre de Sélection du Périphérique),
  - un Registre d'Echange de données.

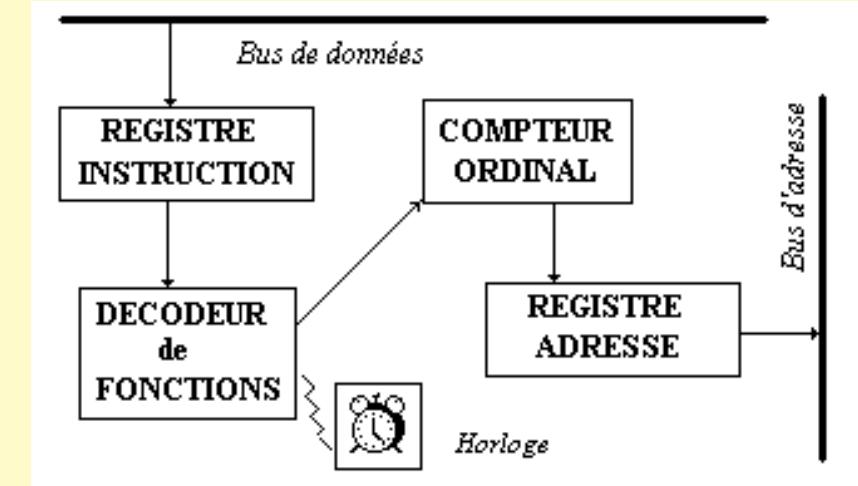
# Unité de commandes

---

---

Son **fonctionnement** est celui du modèle Von Neumann associée aux registres suivants:

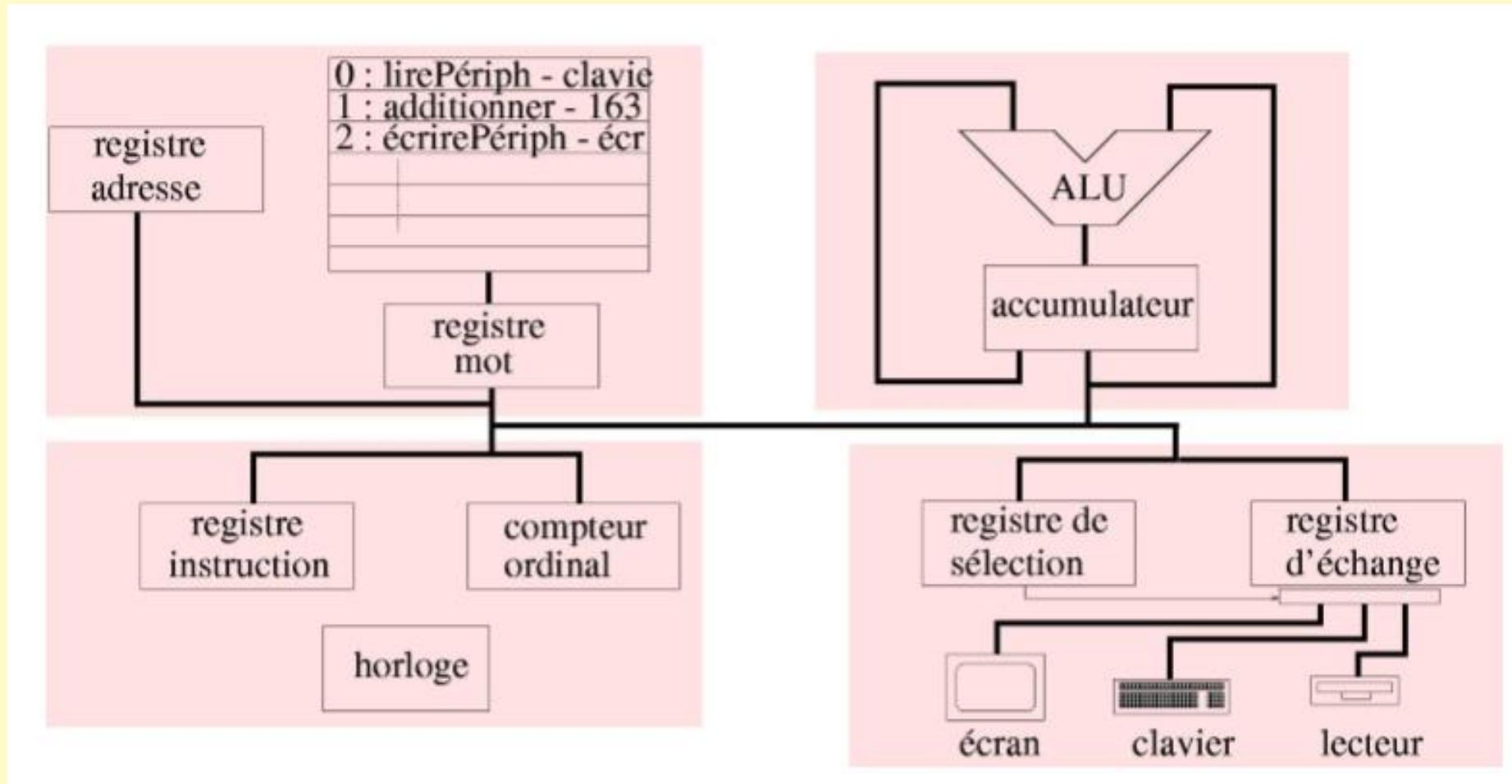
- Compteur ordinal (PC) : adresse mémoire de l'instruction à exécuter,
- Registre d'instruction (RI) : instruction découpée en différentes parties.



# La machine complète

---

---



# AGENDA DU CHAPITRE

---

---

1  
Histoire de  
l'informatique

3  
La machine de  
von Neumann

4  
codage de  
l'information

2  
Structure d'un  
ordinateur

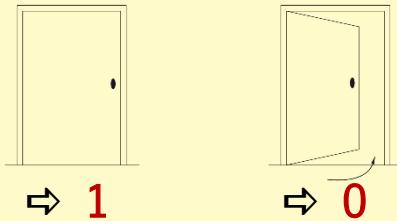
5  
Le langage des  
ordinateurs

# Information binaire

---

---

- L'information élémentaire est codée en binaire.



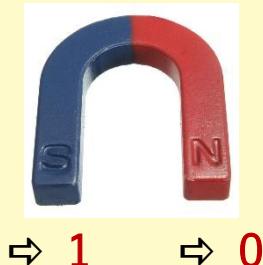
Porte ouverte ou  
fermée

5V  $\Rightarrow$  1  
0V  $\Rightarrow$  0

Tension électrique

Vrai  $\Rightarrow$  1  
Faux  $\Rightarrow$  0

Proposition



Aimantation

- Il s'agit d'un codage purement conventionnel (il peut être inversé !).

# Pourquoi du binaire ?

---

---

- Parce que c'est plus simple à réaliser !
- Parce que cela suffit !
- Machines électroniques :
  - Codage de tensions électriques :
    - $5V = 1$
    - $0V = 0$
- Note : il a existé des machines ternaires et quaternaires.

# Unités de mesure des tailles

---

---

- L'information est codée en binaire :
  - Un bit peut prendre 2 valeurs : 0 ou 1
  - Un kilobit (noté 1 Kb) est égal à  $2^{10}$  bits
  - Combien bits exactement vaut 1 Kb ?
- Un mégabit (noté 1 Mb) vaut  $2^{10}$  kilobits
  - Combien de bits exactement vaut 1 Mb ?
- Un gigabit (noté 1 Gb) vaut  $2^{10}$  mégabits
  - Combien de bits exactement vaut 1 Gb ?

# Unités de mesure des tailles

---

---

- Dans la plupart des ordinateurs, chaque caractère est codé sur 8 bits.
  - Combien de caractères différents est-il possible de coder avec 8 bits ?
- **Un octet = 8 bits**
- **Attention** : la traduction américaine d'un octet est « **byte** »
  - 1 KB = 1 kilobyte = 1 kilooctet  $\neq$  1 Kb = 1 kilobit
  - 1 MB = 1 mégabyte = 1 mégaoctet  $\neq$  1 Mb = 1 mégabit
  - 1 GB = 1 gigabyte = 1 gigaoctet  $\neq$  1 Gb = 1 gigabit

# Unités de mesure des tailles

---

---

- $1 \text{ Ko} = 2^{10}$  octets
- $1 \text{ Mo} = ?? \text{ Ko} = ?? \text{ Octets}$
- $1 \text{ Go} = ?? \text{ Mo} = ?? \text{ Ko} = ?? \text{ octets}$
  
- Un kilooctet (noté 1 Ko) est égal à ??? kilobits, soient ??? bits
- Un mégaoctet (noté 1 Mo) est égal à ??? mégabits, soient ??? bits
- Un gigaoctet (noté 1 Go) est égal à ??? gigabits, soient ??? bits

# AGENDA DU CHAPITRE

---

---

1  
Histoire de  
l'informatique

3  
La machine de  
von Neumann

4  
codage de  
l'information

2  
Structure d'un  
ordinateur

5  
Le Langage des  
ordinateurs

# Jeu d'instructions

---

---

- Ensemble des instructions exécutables sur une machine.
- Différents **formats d'instruction** suivant le nombre de parties réservées aux opérandes (ou adresses) :

| opération | opérande | (format 1 adresse)

| opération | opérande 1 | opérande 2 | (format 2 adresses)

...

# Exemple d'instructions au format « 1 adresse »

---

---

lire Périm nomPérim  
écrirePérim nomPérim  
chargerAcc adMémoire  
mémoAcc adMémoire  
additionner adMémoire  
soustraire adMémoire  
multiplier adMémoire  
diviser adMémoire  
testZéro adMémoire  
stop

# Autre exemple

---

---

Horloge à 4 phases (1 par cycle de l'UC) :

- acquisition au clavier
- addition de la valeur lue avec une donnée en mémoire
- affichage du résultat à l'écran

lirePéroph clavier

additionner 163

écrirePéroph écran

H0 : RA  $\leftarrow$  PC

H1 : RI  $\leftarrow$  RM, PC  $\leftarrow$  PC +1

H2 : RSP  $\leftarrow$  RI<sub>p</sub>

H3 : ACC  $\leftarrow$  RE

H0 : RA  $\leftarrow$  PC

H1 : RI  $\leftarrow$  RM, PC  $\leftarrow$  PC +1

H2 : RA  $\leftarrow$  RI<sub>p</sub>

H3 : ACC  $\leftarrow$  ACC + RM

H0 : RA  $\leftarrow$  PC

H1 : RI  $\leftarrow$  RM, PC  $\leftarrow$  PC +1

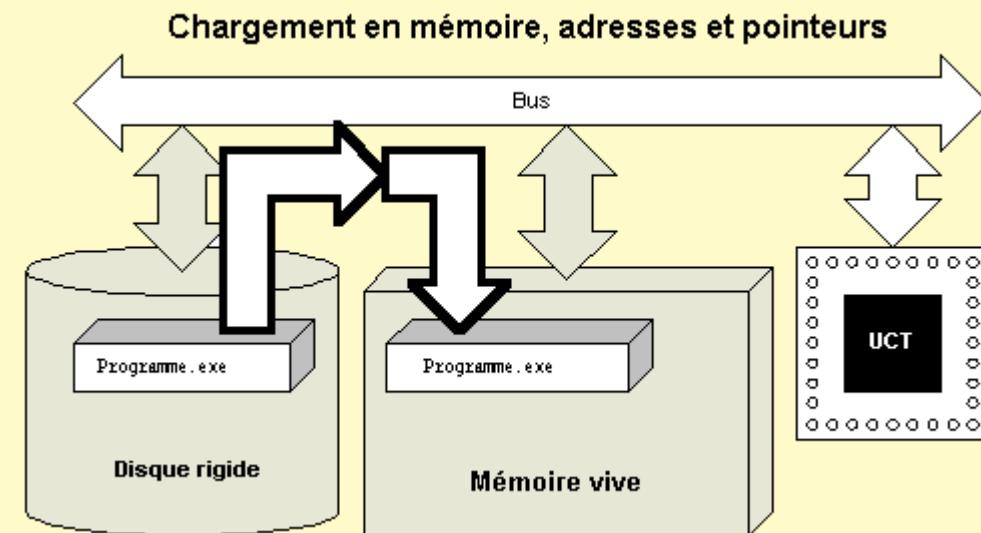
H2 : RSP  $\leftarrow$  RI<sub>p</sub>

H3 : RE  $\leftarrow$  ACC

# Chargement

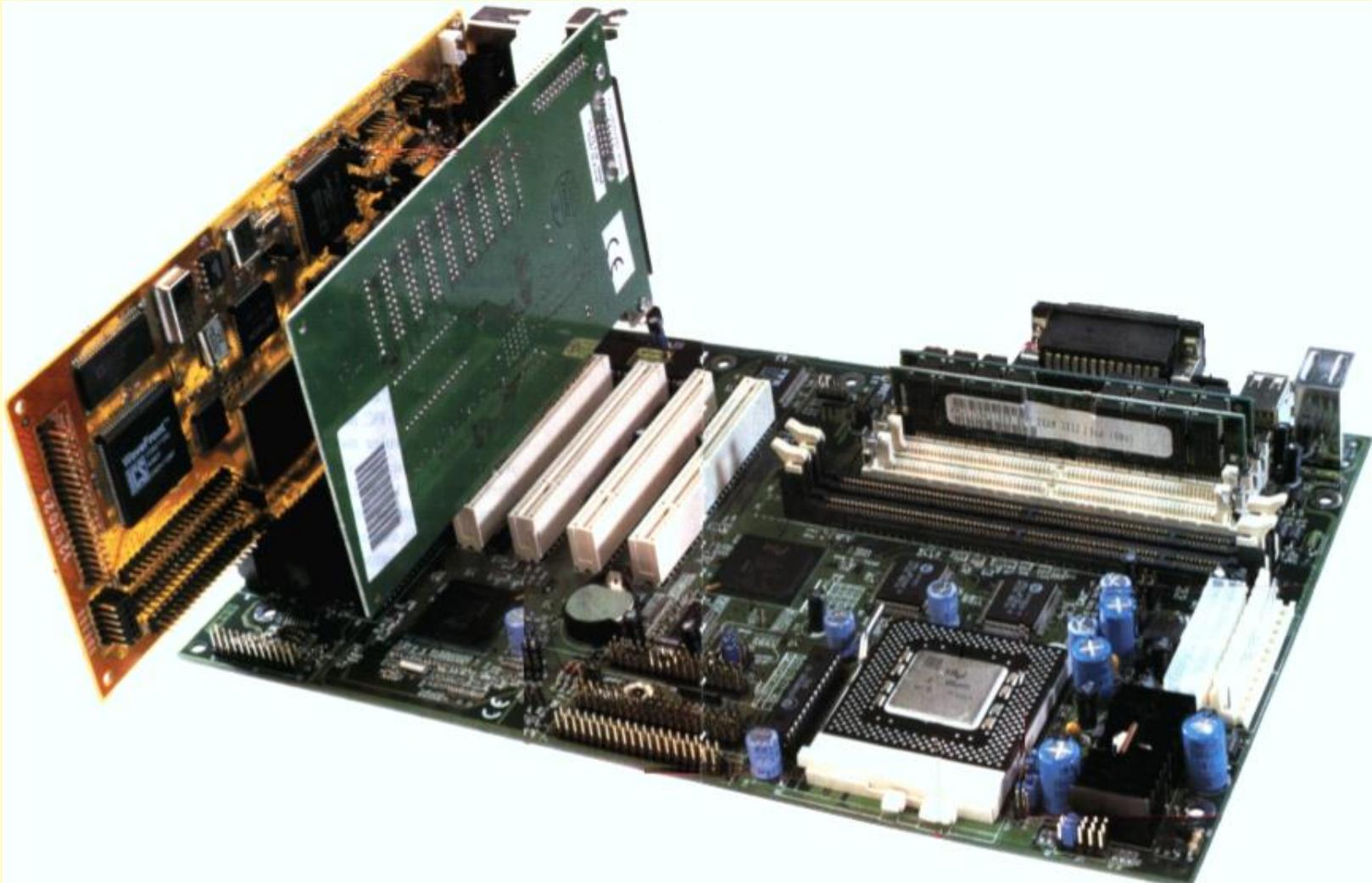
---

- Chargeur = programme qui lit un programme et l'écrit en mémoire.
- Le chargeur est chargé par un dispositif matériel appelé micro chargeur.



# Physiquement ...

---





# Synthèse

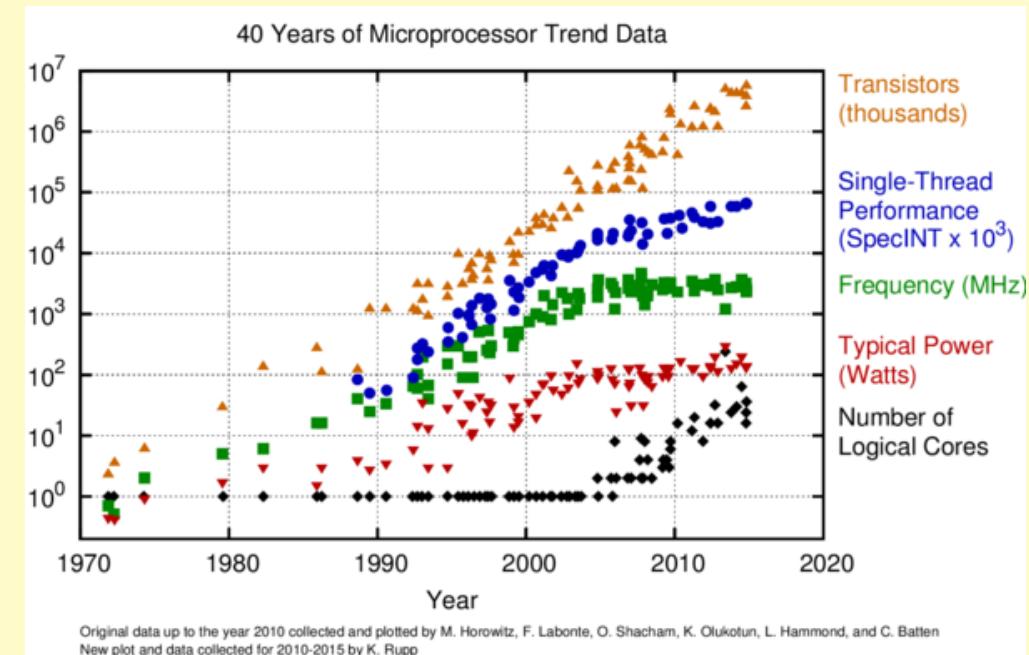
copyrights 365jourspourapprendre.f

# Une course perpétuelle à la performance

---

---

- L'évolution des ordinateurs est liée à :
  - l'évolution des besoins (i.e., des applications),
  - l'évolution des performances de chaque composant.
- Le principe de conception d'une machine est lié à la performance.



# Les solutions

---

---



- Augmenter le nombre de bits manipulés simultanément.
- Changer la structure de la mémoire.
- Réduire la fréquence des accès mémoire.
- Augmenter la bande passante.

# Mesures de performance

---

---

- La performance dépend de l'architecture :

Mesure	Pertinence
La fréquence d'horloge	Objectif pour une même architecture
Nombre d'instruction /s (MIPS, MFLOPS)	Objectif pour un même jeu d'instructions
Temps d'exécution des programmes (benchmarks)	Le plus objectif



# ARCHITECTURE DES ORDINATEURS

## CHAPITRE 2 REPRESENTATION DE L'INFORMATION



# AGENDA DU CHAPITRE

---

---

1  
codage de  
l'information

3  
Implantation  
de  
l'information

2  
Arithmétique  
des  
ordinateurs

# AGENDA DU CHAPITRE

---

---

1  
codage de  
l'information

3  
Implantation  
de  
l'information

2  
Arithmétique  
des  
ordinateurs

# Principe du codage

---

---

 $I = \{i_1, \dots, i_m\}$ 

ensemble d'informations

 $A = \{a_1, \dots, a_n\}$ 

alphabet

 $a_i$ 

caractères de  $A$

 $a_1 a_3 a_4 a_8$ 

mot de  $A$

 $|A|$ 

base du codage

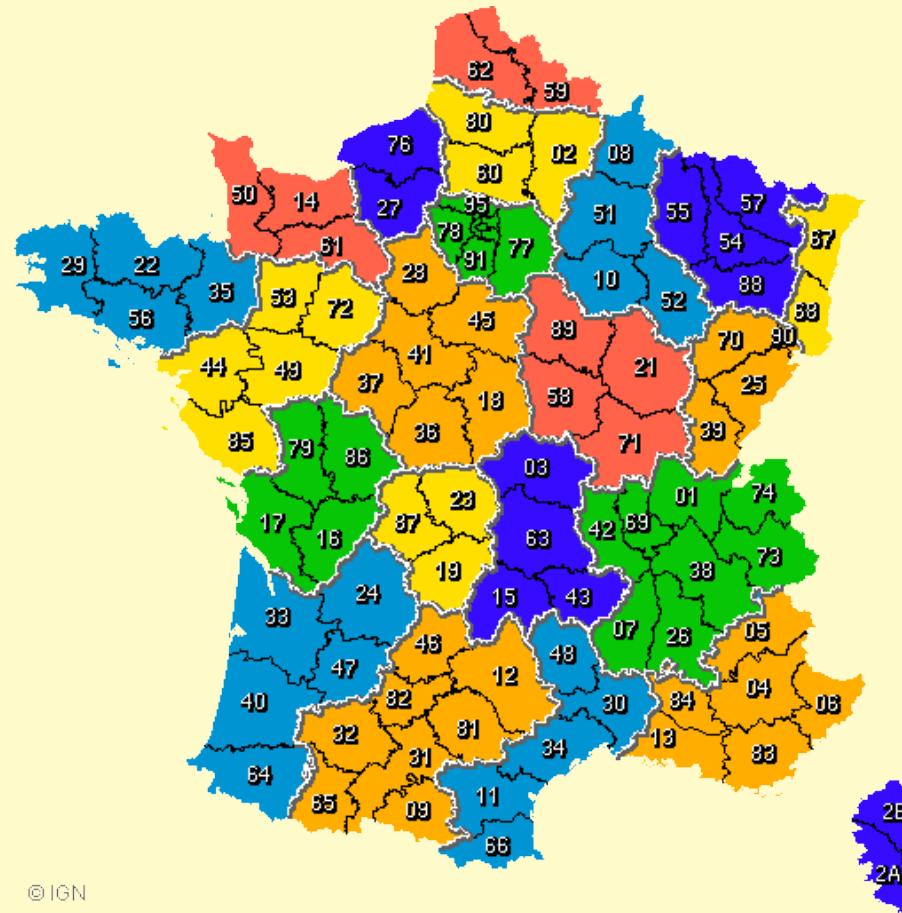
$\Rightarrow$  coder  $I$ : associer à chaque  $i \in I$  un mot de  $A$

# Exemple

---

---

- $/=\{\text{Ain, Aisne, Allier, ...}\}$
- $A =\{0,\dots,9\}$
- $|A|=10$
- Ain est associé à 1
- Loir-et-Cher est associé à 41
- ...



# Quelques propriétés

---

---

- Codage **redondant** : numéro de téléphone
- Codage **non redondant** : numéro de sécurité sociale
- Codage à **longueur fixe** : code postal
- Codage à **longueur variable** : alphabet morse

# Quelques codes

---

---

- Codage des nombres
- Codage des données alphanumériques
- Codage détecteur d'erreur (contrôle de parité, code de Hamming, ...)



# Notation positionnelle

---

---

- La représentation d'un nombre est un codage.

*Ne pas confondre un nombre avec sa représentation !*

- La quantité **dix** peut être représentée par :
  - dix,
  - 10,
  - \*\*\*\*\*,
  - 1010,
  - A,
  - X,
  - etc...

# Représentation de l'information

---

---

- Information externe :
  - formats multiples et variés
  - textes, images, sons ...
  - systèmes d'acquisition des données (micros, capteurs, cartes d'acquisition, scanners )
  
- Information interne :
  - binaire 0101111...
  - nécessité d'avoir des unités d'échanges : transformation de l'information en binaire

# Représentation de l'information

---

---

- Un système informatique :
  - ensemble de composants = quelques centaines de milliers de transistors /composants
  - qui fonctionnent selon deux états logiques notés 0 et 1 (logique binaire)
  - Qui correspond à deux niveaux électriques 0 et + 5 /3,3 volts
- Information logique (0,1) représente :
  - des chiffres, nombres (entiers, réels)
  - des caractères
  - des chaînes de caractères
- Autres possibilités de codage (extension du binaire)
  - hexadécimal, Binary Coded Decimal , ASCII ...

# Les principales bases utilisées en informatique

---

- Habitude de travailler en base 10 (système décimal)
- 10 symboles distincts : les chiffres !
- En base  $b$ , on utilise  $b$  chiffres

Nom de la base	$b$	Chiffres
Binaire	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Décimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadécimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

# Représentation d'un nombre dans une base **b**

---

---

$$a_n a_{n-1} \dots a_1 a_0 \color{red}{b} = \sum_{i=0}^n a_i \times b^i$$

- $a_n, a_{n-1}, \dots$  : **poids forts**
- ...,  $a_1, a_0$  : **poids faibles**

# Représentation d'un nombre en base 2

---

---

- Alphabet de 2 caractères binaires (appelés bit) : {0,1}
- $01101_2 = 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$   
=  $8_{10} + 4_{10} + 1_{10}$   
=  $13_{10}$

# Représentation d'un nombre en base 16

---

---

- Alphabet de 16 caractères hexadécimaux : {0,...,9,A,B,C,D,E,F}
- $01B_{16} = 0 \times 16^2 + 1 \times 16^1 + B \times 16^0$   
=  $16_{10} + 11_{10}$   
=  $27_{10}$

# Exemple détaillé avec cette année

---

---

- Binaire

$$\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\ 1024 + 512 + 256 + 128 + 64 + 32 + 0 + 0 + 4 + 0 + 0 \end{array}$$

- Octal

$$\begin{array}{cccc} 3 & 7 & 4 & 4 \\ 3 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 4 \times 8^0 \\ 1536 + 448 + 32 + 4 \end{array}$$

- Décimal

$$\begin{array}{cccc} 2 & 0 & 2 & 0 \\ 2 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 0 \times 10^0 \\ 2000 + 0 + 20 + 0 \end{array}$$

- Hexadécimal

$$\begin{array}{ccc} 7 & E & 4 \\ 7 \times 16^2 + 14 \times 16_1 + 4 \times 16^0 \\ 1792 + 224 + 4 \end{array}$$

# Notations

---

---

- Notation d'indice :

- par défaut, un nombre est écrit en décimal (base 10)
- un nombre  $\alpha$  en base  $b$  se note  $\alpha_b$  ou  $(\alpha)_b$

- Exemple :

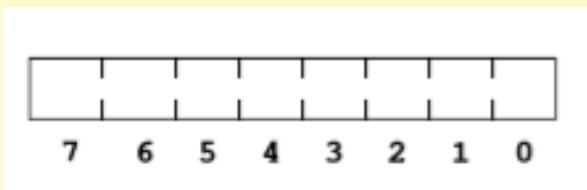
$$\begin{aligned} 2020 &= 11111100100_2 \\ &= 3744_8 \\ &= (3744)_8 \\ &= 2020_{10} \\ &= 7E4_{16} \\ &= (7E4)_{16} \end{aligned}$$

# Représentation des nombres

---

---

- Ordinateurs travaillent sur un nombre fixe de bits.
- La notion d'octet :
  - un octet = 8 bits



- La notion de mot :
  - un mot = 8, 16, 32 64 bits
  - Intel 80386 et 80486 -> un mot = 32 bits
  - Intel Pentium 4 -> un mot = 128 bits

# Exemple en langage C

---

---

TYPE	DESCRIPTION	TAILLE
<b>int</b>	entier standard signé	4 octets: - $2^{31}$ à $2^{31}-1$
<b>unsigned int</b>	entier positif	4 octets: 0 à $2^{32}$
<b>short</b>	entier court signé	2 octets: - $2^{15}$ à $2^{15}-1$
<b>unsigned short</b>	entier court non signé	2 octets: 0 à $2^{16}$
<b>char</b>	caractère signé	1 octet : - $2^7$ à $2^7-1$
<b>unsigned char</b>	caractère non signé	1 octet : 0 à $2^8$

# Représentation des données

---

---

- Caractères à coder, alphabet + , ! \* " %
- Code **ASCII** (7 bits + 1 bit de parité)
  - American Standard Code for Information Interchange
  - permet 128 combinaisons différentes ( $2^7$ )
  - utilisation de cette table ...
- Code **EBCDIC** (8 bits, 256 caractères)
  - Extended Binary Coded Decimal Interchange Code
  - utilisé principalement par IBM
- Code **ANSI**
  - American National Standard Institute
  - utilisés par certains logiciels (Windows)
  - code ASCII + extensions multilingue alphabets occidentaux

# Le code ASCII

---

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0 000	MUL	(null)	32	20 040	4#32;	Space		64	40 100	4#64;	Ø	96	60 140	4#96;	~		
1	1 001	SOH	(start of heading)	33	21 041	4#33;	:	:	65	41 101	4#65;	A	97	61 141	4#97;	a		
2	2 002	STX	(start of text)	34	22 042	4#34;	"	"	66	42 102	4#66;	B	98	62 142	4#98;	b		
3	3 003	ETX	(end of text)	35	23 043	4#35;	#	#	67	43 103	4#67;	C	99	63 143	4#99;	c		
4	4 004	EOT	(end of transmission)	36	24 044	4#36;	\$	\$	68	44 104	4#68;	D	100	64 144	4#100;	d		
5	5 005	ENQ	(enquiry)	37	25 045	4#37;	%	%	69	45 105	4#69;	E	101	65 145	4#101;	e		
6	6 006	ACK	(acknowledge)	38	26 046	4#38;	&	&	70	46 106	4#70;	F	102	66 146	4#102;	f		
7	7 007	BEL	(bell)	39	27 047	4#39;	'	'	71	47 107	4#71;	G	103	67 147	4#103;	g		
8	8 010	BS	(backspace)	40	28 050	4#40;	(	(	72	48 110	4#72;	H	104	68 150	4#104;	h		
9	9 011	TAB	(horizontal tab)	41	29 051	4#41;	)	)	73	49 111	4#73;	I	105	69 151	4#105;	i		
10	A 012	LF	(NL line feed, new line)	42	2A 052	4#42;	*	*	74	4A 112	4#74;	J	106	6A 152	4#106;	j		
11	B 013	VT	(vertical tab)	43	2B 053	4#43;	+	+	75	4B 113	4#75;	K	107	6B 153	4#107;	k		
12	C 014	FF	(NP form feed, new page)	44	2C 054	4#44;	,	,	76	4C 114	4#76;	L	108	6C 154	4#108;	l		
13	D 015	CR	(carriage return)	45	2D 055	4#45;	-	-	77	4D 115	4#77;	M	109	6D 155	4#109;	m		
14	E 016	SO	(shift out)	46	2E 056	4#46;	.	.	78	4E 116	4#78;	N	110	6E 156	4#110;	n		
15	F 017	SI	(shift in)	47	2F 057	4#47;	/	/	79	4F 117	4#79;	O	111	6F 157	4#111;	o		
16	10 020	DLE	(data link escape)	48	30 060	4#48;	0	0	80	50 120	4#80;	P	112	70 160	4#112;	p		
17	11 021	DC1	(device control 1)	49	31 061	4#49;	1	1	81	51 121	4#81;	Q	113	71 161	4#113;	q		
18	12 022	DC2	(device control 2)	50	32 062	4#50;	2	2	82	52 122	4#82;	R	114	72 162	4#114;	r		
19	13 023	DC3	(device control 3)	51	33 063	4#51;	3	3	83	53 123	4#83;	S	115	73 163	4#115;	s		
20	14 024	DC4	(device control 4)	52	34 064	4#52;	4	4	84	54 124	4#84;	T	116	74 164	4#116;	t		
21	15 025	NAK	(negative acknowledge)	53	35 065	4#53;	5	5	85	55 125	4#85;	U	117	75 165	4#117;	u		
22	16 026	SYN	(synchronous idle)	54	36 066	4#54;	6	6	86	56 126	4#86;	V	118	76 166	4#118;	v		
23	17 027	ETB	(end of trans. block)	55	37 067	4#55;	7	7	87	57 127	4#87;	W	119	77 167	4#119;	w		
24	18 030	CAN	(cancel)	56	38 070	4#56;	8	8	88	58 130	4#88;	X	120	78 170	4#120;	x		
25	19 031	EM	(end of medium)	57	39 071	4#57;	9	9	89	59 131	4#89;	Y	121	79 171	4#121;	y		
26	1A 032	SUB	(substitute)	58	3A 072	4#58;	:	:	90	5A 132	4#90;	Z	122	7A 172	4#122;	z		
27	1B 033	ESC	(escape)	59	3B 073	4#59;	:	:	91	5B 133	4#91;	[	123	7B 173	4#123;	{		
28	1C 034	FS	(file separator)	60	3C 074	4#60;	<	<	92	5C 134	4#92;	\	124	7C 174	4#124;			
29	1D 035	GS	(group separator)	61	3D 075	4#61;	=	=	93	5D 135	4#93;	]	125	7D 175	4#125;	}		
30	1E 036	RS	(record separator)	62	3E 076	4#62;	>	>	94	5E 136	4#94;	^	126	7E 176	4#126;	~		
31	1F 037	US	(unit separator)	63	3F 077	4#63;	?	?	95	5F 137	4#95;	_	127	7F 177	4#127;	DEL		

Source: [www.LookupTables.com](http://www.LookupTables.com)

# Evolutions du code ASCII

---

---

- "iso-latin-1", également connu sous le nom de iso-8859-1
  - le huitième bit qui servait pour le contrôle de parité, va être utilisé pour coder plus de caractères.
  - les codes ASCII de 0 à 7F (127 en décimal) demeurent inchangés,
  - les codes supérieurs (ceux qui ont le bit 7 à 1) représentent quelques symboles supplémentaires, ainsi que les lettres accentuées qui satisfont aux exigences des langues de l'Europe de l'Ouest.
- Unicode
  - plus de déclinaison comme dans les codes ISO,
  - codage sur 16 bits de l'ensemble des signes,
  - codage contient des informations telles que le sens d'écriture.

# AGENDA DU CHAPITRE

---

---

1  
codage de  
l'information

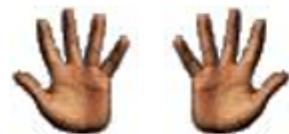
3  
Implantation  
de  
l'information

2  
Arithmétique  
des  
ordinateurs

# Les principaux systèmes de numération

---

---



système de numération décimal



système de numération binaire

# Transformation base 10 vers base b

---

---

- Partie entière :
  - divisions entières successives par la base
  - lecture du reste
  
- Partie fractionnaire :
  - multiplications successives par la base
  - lecture de la partie entière

$$\begin{array}{r} 5 \Big| 53 \text{ Remainder} \\ 5 \Big| 10 \dots\dots 3 \\ 5 \Big| 2 \dots\dots 0 \\ \hline 0 \dots\dots 2 \end{array}$$

To convert from  
base 10 to base 5

$\therefore 53_{10} = 203_5$

## Exemples avec $b = 2$

---

---

- $25_{10} / 2 = 12_{10}$  reste 1      ■  $0,78125_{10} \times 2 = 1,5625_{10}$  partie entière 1
  - $12_{10} / 2 = 6_{10}$  reste 0      ■  $0,5625_{10} \times 2 = 1,125_{10}$  partie entière 1
  - $6_{10} / 2 = 3_{10}$  reste 0      ■  $0,125_{10} \times 2 = 0,25_{10}$  partie entière 0
  - $3_{10} / 2 = 1_{10}$  reste 1      ■  $0,25_{10} \times 2 = 0,5_{10}$  partie entière 0
  - $1_{10} / 2 = 0_{10}$  reste 1      ■  $0,5_{10} \times 2 = 1_{10}$  partie entière 1
- donc  $25_{10} = 11001_2$       donc  $0,78125_{10} = 0,11001_2$

# Transformation base **b** vers base 10

---

---

- $a_n a_{n-1} \dots a_1 a_0$  exprimé en base **b** (noté  $a_n a_{n-1} \dots a_1 a_0$ ) vers une représentation en base 10 :

$$a_n \times b^n + a_{n-1} \times b^{n-1} + \dots + a_1 \times b + a_0$$

- La partie fractionnaire (si nombre réel) :

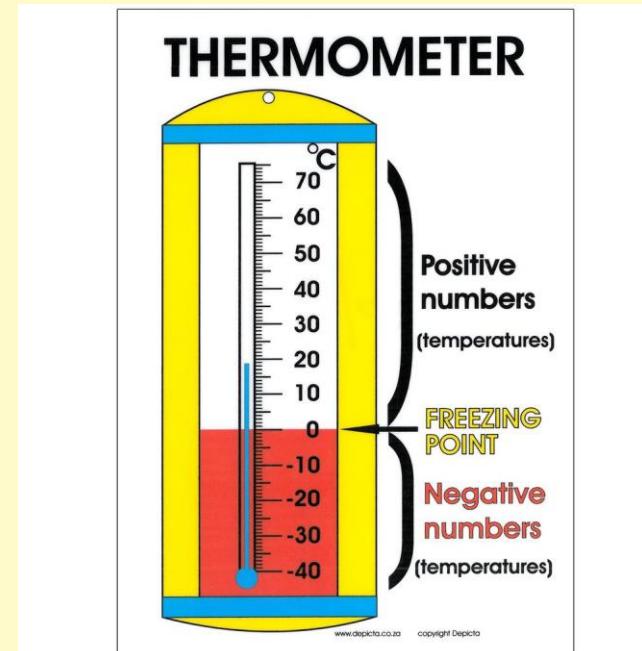
$$a_1 \times b^{-1} + a_2 \times b^{-2} + \dots + a_n \times b^{-n}$$

# Représentations des nombres négatifs

---

---

- signe et valeur absolue
- notations complémentées (à 1 ou 2)
- notations excédentaires



# Signe et valeur absolue

---

---

Sur  $n$  bits :

- signe : bit de poids fort (0: positif, 1: négatif)
- valeur absolue :  $n-1$  bits
- intervalle de valeurs représentées:  $[-2^{n-1} + 1, 2^{n-1} - 1]$

Exemple sur 3 bits :  $[-3, +3]$

000	0
001	1
010	2
011	3
100	-0
101	-1
110	-2
111	-3

# Complément à 2

---

---

Sur  $n$  bits :

- Complément à 1 + 1
- $|x| - |x| = 2^n$
- intervalle de valeurs représentées:  $[-2^{n-1}, 2^{n-1} - 1]$

Exemple sur 3 bits :  $[-3, +3]$

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

# Illustration

---

---

- Conversion entre différentes longueurs de bits :
  - Signe valeur absolue :
    - + 18 sur 8 bits = ?
    - + 18 sur 16 bits = ?
    - - 18 sur 8 bits = ?
    - - 18 sur 16 bits = ?
  - Complément à deux :
    - + 18 sur 8 bits = ?
    - + 18 sur 16 bits = ?
    - - 18 sur 8 bits = ?
    - - 18 sur 16 bits = ?
- Que remarque-t-on à chaque conversion ?

# Additionner des nombres binaires

---

---

- Addition avec des nombres binaires **en complément à 2**.
- Faire les additions suivantes (sur 4 bits) :
  - $(-7) + 5$
  - $3 + 4$
  - $(-4) + 4$
  - $(-4) + (-1)$
  - $5 + 4$
  - $(-7) + (-6)$
- Que remarquez-vous ?

# Soustraire des nombres binaires

---

---

- Pour la soustraction, on prend simplement le complément à 2 du deuxième membre.
- Faire les soustractions suivantes (sur 4 bits) :
  - $2 - 7$
  - $5 - 2$
  - $(-5) - 2$
- Que remarquez-vous ?

# Multiplier des nombres binaires

---

---

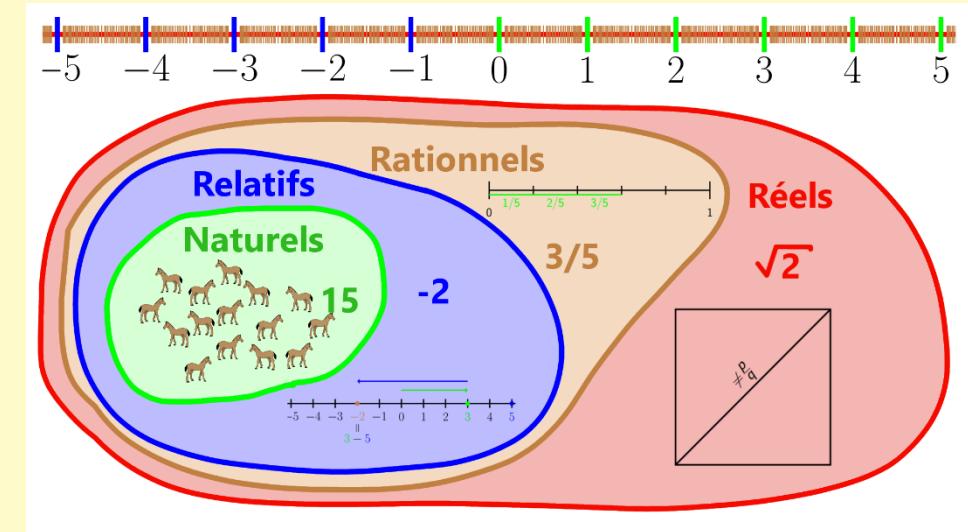
- La multiplication binaire non signée se fait exactement comme en base décimale.
- Faire les multiplications suivantes :
  - $11 \times 13$
  - $15 \times 9$
  - $6 \times 5$
  - $11 \times 11$
- Convertissez les résultats en base décimale.
  
- En fait la multiplication en complément à deux faite par un ordinateur est beaucoup plus complexe.
  - Elle utilise [l'algorithme de Booth](#).

# Représentations des réels

---

---

- Virgule fixe
- Couple (numérateur, dénominateur)
- Virgule flottante



# Virgule flottante

---

---

- Un réel  $\pm m \times b^e$  est représenté par :

- un signe  $\pm$
- une mantisse  $m$
- un exposant  $e$
- une base  $b$

$$1.3254 = \underbrace{13254}_{\text{mantisse}} \times 10^{-4} \quad \begin{array}{c} \text{exposant} \\ \overbrace{\phantom{13254}}^{\text{exposant}} \\ \times \\ \overbrace{\phantom{10^{-4}}}_{\text{mantisse}}^{\text{mantisse}} \end{array}$$

- C'est la représentation la plus utilisée.
- Infinité de représentations possibles d'un même nombre.

# Représentation normalisée

---

---

- Pour une base  $b$  la mantisse  $m \in [1,b[$ .
- Représentation particulière pour zéro (mantisse et exposant = 0).
- La précision et l'intervalle de valeurs représentées dépendent :
  - du nombre de bits de la mantisse,
  - du nombre de bits de l'exposant.

# Norme IEEE 754

---

---

- La mantisse  $m \in [1.02,10.02[$ .
- Le 1 à gauche de la virgule n'est pas représenté.
- Nombres codés sur :
  - 32 bits (simple précision) avec un exposant codé avec un excès de 127,
  - 64 bits (double précision) avec un exposant codé avec un excès de 1023.
- C'est la norme la plus utilisée pour représenter les réels



# Simple précision

---

---

$$(-1)^S \times 1.M \times 2^{E-127}$$

S 1 bits	exposant E 8 bits	mantisso M 23 bits
-------------	----------------------	-----------------------

## Exemple

---

---

$-5_{10}$  est codé par :

1 100 0000 1 010 0000 0000 0000 0000 0000

c'est à dire  $-1 \times 1,25 \times 2^{129 - 127}$

# Précision

---

---

- La représentation des réels sur une machine se base sur un nombre fini de valeurs.
  - C'est donc une représentation approchée !
- 
- Précision de la représentation = différence entre les mantisses de deux nombres réels consécutifs.
  - IEEE 754, simple précision: la précision est de  $2^{-23}$ .

# AGENDA DU CHAPITRE

---

---

1  
codage de  
l'information

3  
Implantation  
de  
l'information

2  
Arithmétique  
des  
ordinateurs

# Circuits combinatoires et séquentiels

---

---

- Eléments de l'algèbre de Boole => circuits combinatoires
  - circuits de base des ordinateurs
- Théorie des automates => circuits séquentiels
  - modèle de base pour le fonctionnement des circuits
- Signaux logiques et analogiques :
  - traitement de l'information
    - » traiter
    - » mémoriser des signaux électriques
    - » transférer

# Différents types de variables

---

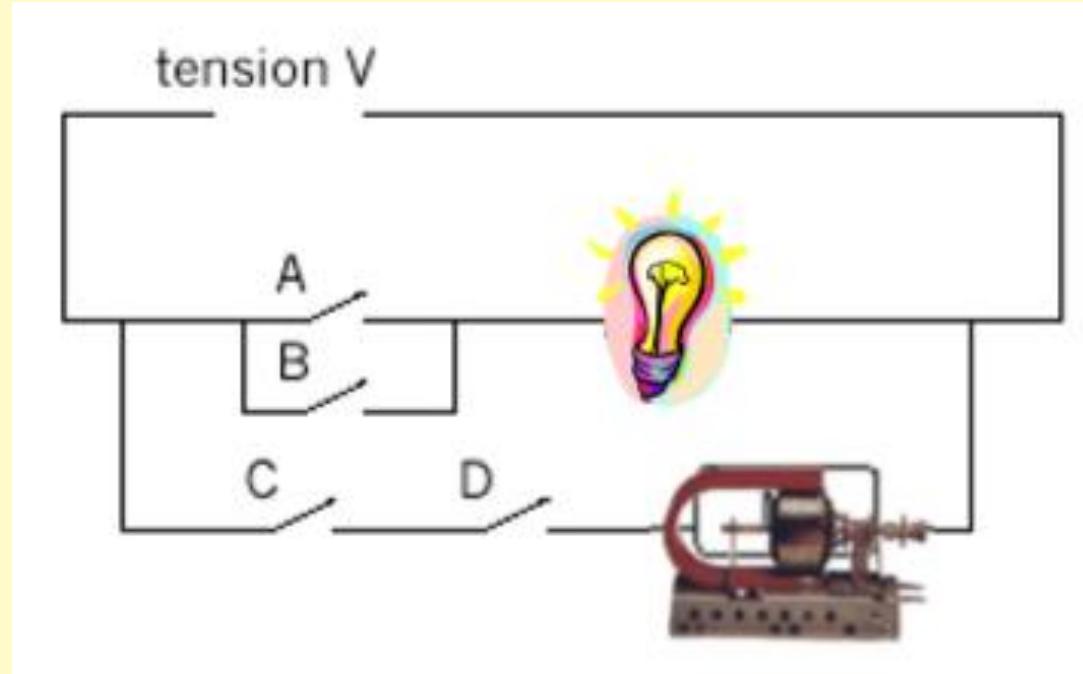
---

- De nombreuses grandeurs peuvent prendre une infinité de valeurs => vitesse d'une voiture, température ... **Valeurs continues**
  - Ces valeurs continues dans les systèmes analogiques peuvent être représentées par un nombre fini de **Valeurs discrètes** dans les systèmes numériques.
  - D'autres systèmes travaillent avec des valeurs ne prenant que 2 états : **vrai** ou **faux**.
    - interrupteur ouvert ou fermé
    - lampe allumée ou pas
    - affirmation vraie ou fausse
    - grandeurs  $>$  ou  $<$  à un seuil
- 
- Valeurs logiques**

# Variables logiques : opérateurs de base ET et OU

---

---



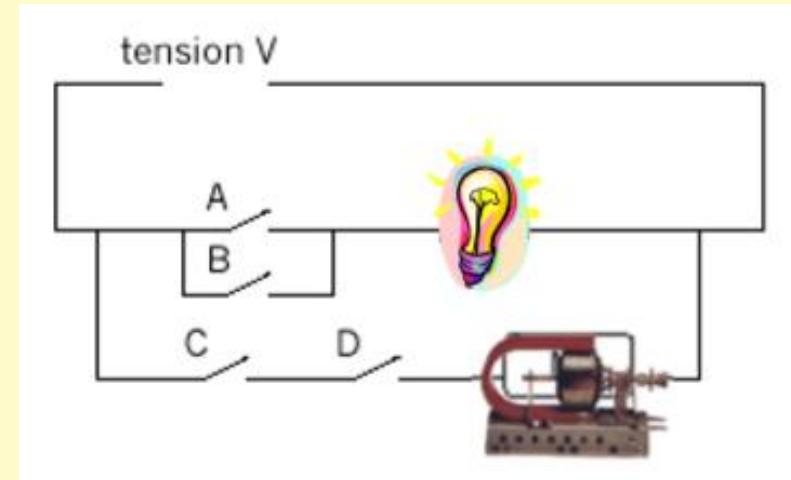
Comment exprimer ce problème par des variables logiques ?

# Variables logiques : opérateurs de base ET et OU

---

---

- Interrupteurs : Variables A, B, C, D
  - Lampe : Variable L
  - Moteur : Variable M
- 
- Logique Positive : 1=Vrai=Actif 0=Faux=Inactif
    - OU est noté +
    - ET est noté x
  - Equations logiques :
    - $L = A + B$
    - $M = C \times D = C . D$

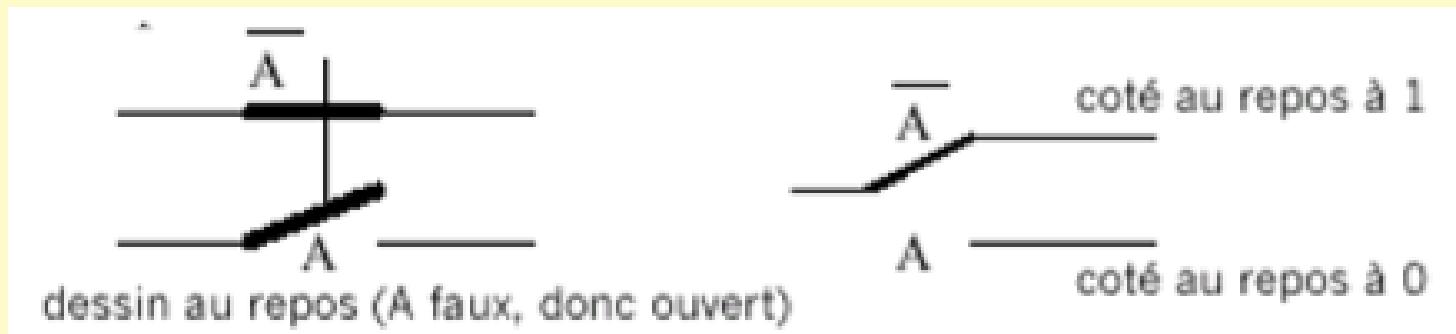


# Variables logiques : opérateurs de base NON

---

---

- Lampe A allumée :  $A = 1$
- Lampe A éteinte :  $A = 0$  ou  $A = 1$  (Complément de  $A$ .)
- Un interrupteur peut être caractérisé par deux variables  $A$  et  $\bar{A}$  soit 2 interrupteurs différents pouvant être actionnés en même temps.



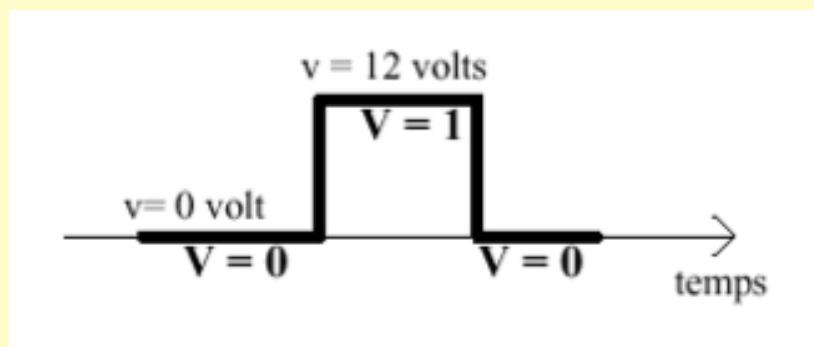
# Variables logiques et signal électrique

---

---

Un signal électrique peut prendre deux valeurs de tension :

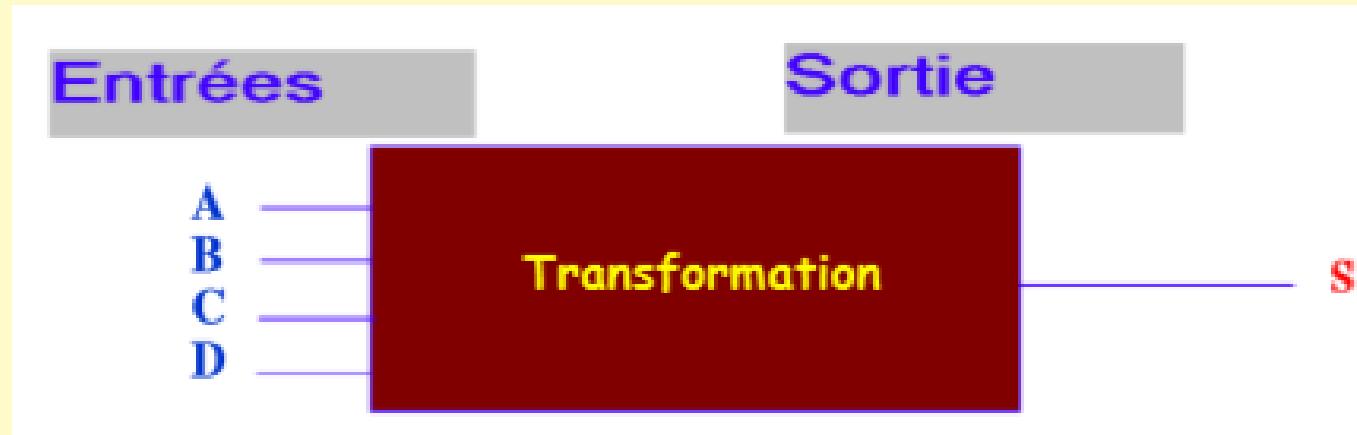
- $v$  variant de 0 à 12 Volts
- $V$  variable logique est associée à  $v$  , telle que
  - »  $V = 0$  quand la tension est nulle
  - »  $V = 1$  quand la tension =12 Volts



# Fonction logique

---

---



- Une **fonction logique** exprime une transformation des entrées d'un circuit pour donner une ou plusieurs sorties.
- Table de vérité pour la conception de fonction logique complexe.

# Tables de vérité

---

---

Description d'une opération logique :

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

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

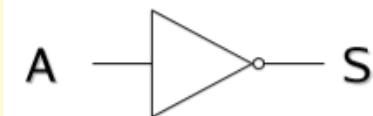
$a$	$b$	$ab$
0	0	0
0	1	0
1	0	0
1	1	1

# Exemples de fonctions logiques

---

---

En identifiant l'état haut à la valeur 1 et l'état bas à la valeur 0, on peut exprimer la valeur de sortie de ces trois circuits à partir des valeurs de leurs entrées :



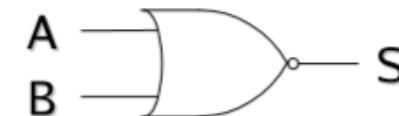
A	S
0	1
1	0

NON



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

NAND



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

NOR

# Portes logiques

---

---

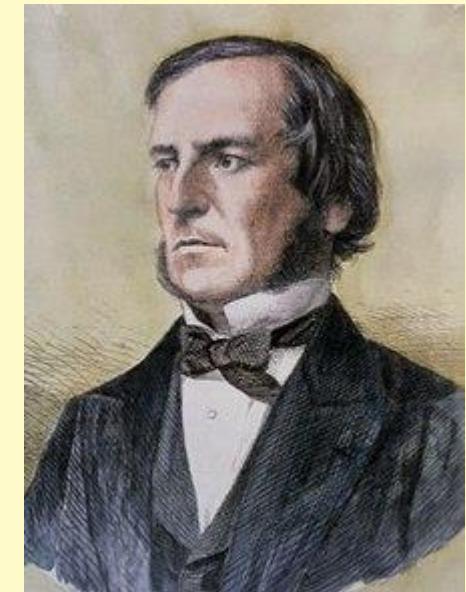
- Mises en œuvre des opérations logiques de base.
- Les portes NAND et NOR sont :
  - complètes : on peut réaliser n'importe quelle fonction booléenne avec l'une ou l'autre,
  - faciles à fabriquer,
  - à la base de la plupart des circuits intégrés des ordinateurs actuels.

# Comment réaliser des circuits combinatoires ?

---

---

- Pour décrire les circuits réalisables en combinant des portes logiques, on a besoin d'une algèbre opérant sur les variables 0 et 1.
- C'est l'algèbre booléenne définie par G. Boole (1815-1864 ).
- A noter que l'algèbre binaire étudiée par Leibniz dès 1703.



# Fonctions booléennes

---

---

- Une fonction booléenne à une ou plusieurs variables est une fonction qui renvoie une valeur ne dépendant que de ces variables.
- La fonction NON est ainsi définie comme :
  - $f(A) = 1$  si  $A = 0$
  - $f(A) = 0$  si  $A = 1$
- Toute fonction peut être décrite en spécifiant lesquelles des combinaisons d'entrée donnent 1.
- On peut donc représenter une fonction logique comme le « ou » logique (OR) d'un ensemble de conditions « et » (AND) sur les combinaisons d'entrée.

# Fonctions booléennes

---

---

Terme construit à partir de :

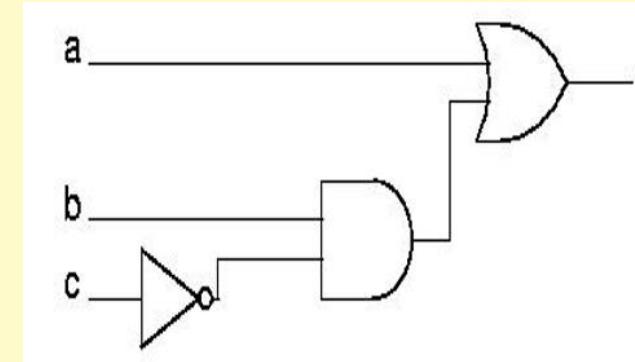
- variables booléennes,
- d'opérateur booléens.

$$a$$

$$\bar{c}$$

$$a + b\bar{c}$$

- Peut être exprimée (réalisée) par des portes logiques.



# Fonctions booléennes

---

---

- En notant :

- $\bar{A}$  le NOT de A
- $A + B$  le OR de A et B
- $A \cdot B$  ou  $AB$  le AND de A et B

on peut représenter une fonction comme somme logique de produits logiques .

- Par exemple :

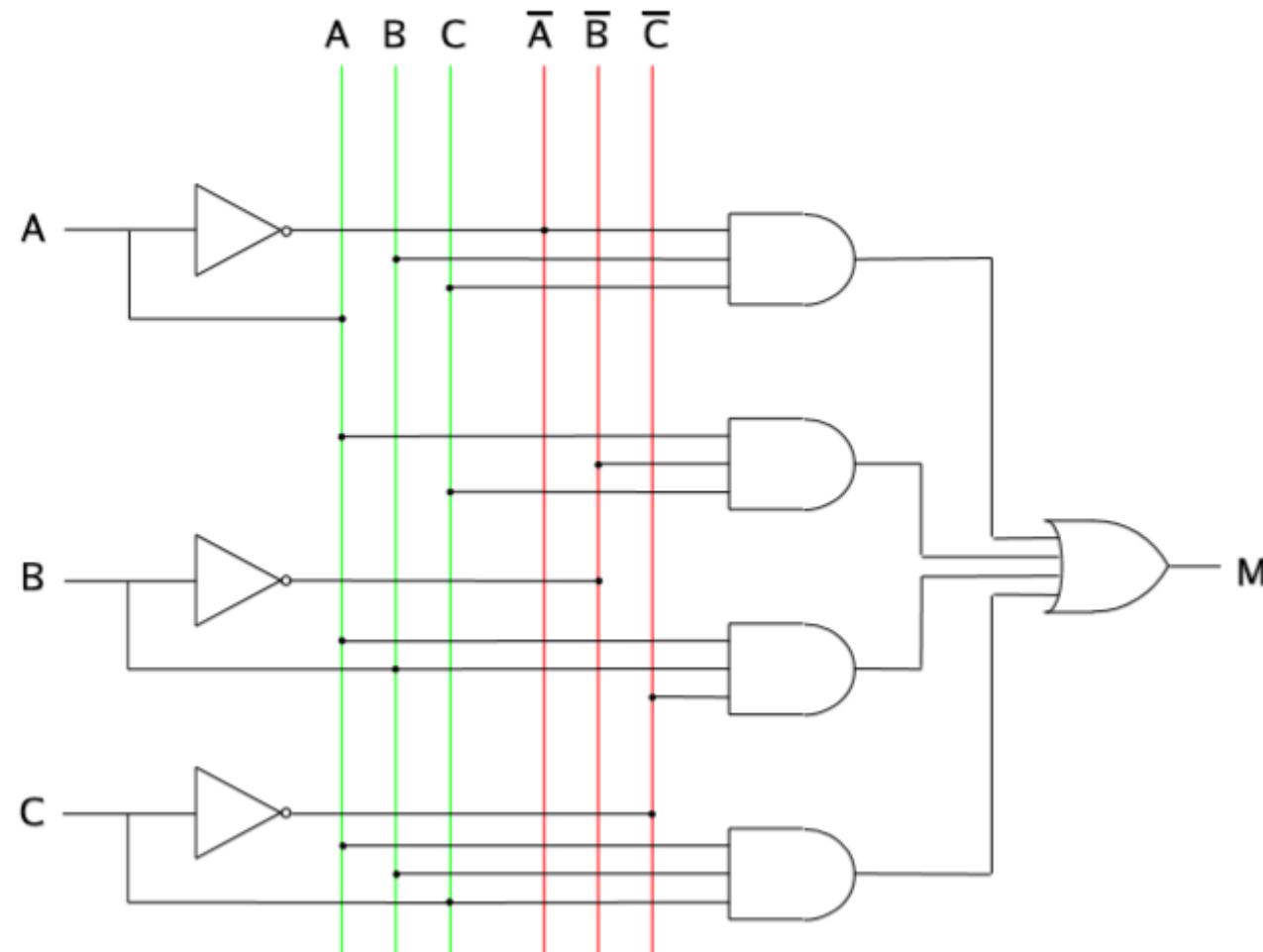
- $\bar{A}\bar{B}C$  vaut 1 seulement si  $A = 1$  et  $B = 0$  et  $C = 1$
- $\bar{A}\bar{B} + \bar{B}\bar{C}$  vaut 1 si et seulement si  $(A = 1 \text{ et } B = 0)$  ou bien  $(B = 1 \text{ et } C = 0)$

# Exemple : la fonction majorité M

---

---

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



# Implantation des fonctions booléennes

---

---

- Toute fonction logique de  $n$  variables peut donc être décrite sous la forme d'une somme logique d'au plus  $2^n$  produits de termes.
  - Par exemple :  $M = \bar{A}BC + A\bar{B}C + ABC + A\bar{B}\bar{C}$
- Cette formulation fournit une méthode directe pour implanter n'importe quelle fonction booléenne.

# Simplification d'expressions booléennes

---

---

- Deux fonctions sont équivalentes si et seulement si leurs tables de vérité sont identiques.
- Il est intéressant d'implanter une fonction avec le moins de portes possible :
  - économie de place sur le processeur,
  - réduction de la consommation électrique,
  - réduction du temps de parcours du signal.

# Simplification d'expressions booléennes

---

---

- L'algèbre booléenne peut être un outil efficace pour simplifier les fonctions.
- La plupart des règles de l'algèbre ordinaire restent valides pour l'algèbre booléenne.
- Exemple :  $AB + AC = A(B + C)$ 
  - on passe de trois portes à deux,
  - le nombre de niveaux de portes reste le même.

# Simplification d'expressions booléennes

---

- Pour réduire la complexité des fonctions booléennes, on essaye d'appliquer des identités simplificatrices à la fonction initiale.
- Besoin d'identités remarquables pour l'algèbre booléenne :

Nom	Forme AND	Forme OR
Identité	$1A = A$	$0 + A = A$
Nul	$0A = 0$	$1 + A = 1$
Idempotence	$AA = A$	$A + A = A$
Inverse	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutativité	$AB = BA$	$A + B = B + A$
Associativité	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributivité	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorbtion	$A(A + B) = A$	$A + AB = A$
De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\overline{B}$



# ARCHITECTURE DES ORDINATEURS

## CHAPITRE 3 LA STRATEGIE MEMOIRE

A ce jour, il n'existe pas de technologie optimale pour satisfaire le besoin en mémoire d'un ordinateur.



# AGENDA DU CHAPITRE

---

---

1  
Introduction

3  
Mémoire  
principale

4  
Mémoire cache

2  
Modes d'accès

5  
Mémoire  
externe

# AGENDA DU CHAPITRE

---

---

1  
Introduction

3  
Mémoire  
principale

4  
Mémoire cache

2  
Modes d'accès

5  
Mémoire  
externe

# Caractéristiques

---

- types physiques de mémoires
- durée de mémorisation
- emplacement
- capacité
- performance
- hiérarchie mémoire



# Définitions

---

---

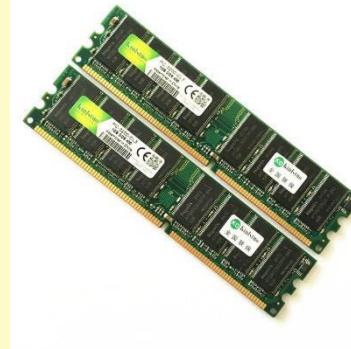
- **Mémoire** : dispositif capable de conserver et restituer une information
- **Mot mémoire** : ensemble de bits pouvant être lus ou écrits simultanément

# Les différents types physiques de mémoires

---

---

- Semi-conducteur (registre, mémoire principale, ...)



- Magnétique (disque dur, clé USB, ...)



- Optique (cd-rom, dvd-rom, ...)



# Durée de mémorisation / Emplacement

---

---

- Fonction du temps :
  - quasi-permanente : disque, ROM
  - temporaire : RAM
- Fonction de la présence d'alimentation électrique :
  - sensible : RAM
  - insensible : disque
- Interne au processeur :
  - registre
- Interne à la carte mère:
  - mémoire principale
- Externe à la carte mère :
  - mémoire secondaire : disque
- Externe à l'unité centrale:
  - mémoire tertiaire : bande magnétique

# Capacité

---

---

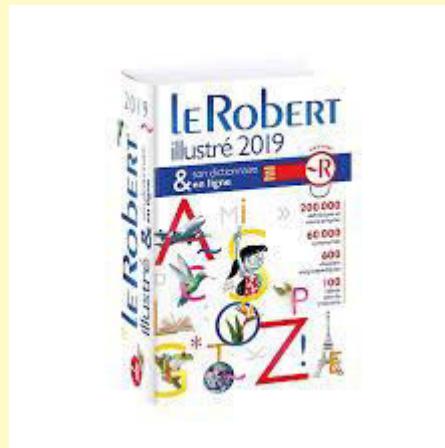
- Nombre d'informations stockables
- Exprimée en octet (byte) ou en multiple d'octet :
  - kilo  $1K = 2^{10} = 1024$
  - mega  $1M = 2^{20} = 1\ 048\ 576$
  - giga  $1G = 2^{30} = 1\ 073\ 741\ 824$
  - tera  $1T = 2^{40} = 1\ 099\ 511\ 627\ 776$
  - péta  $1P = 2^{50} = 1\ 125\ 899\ 906\ 842\ 620$

# Capacité

---

---

- 1 caractère ('a', '?', '2', ...) = 1 octet
- Le petit Robert (2600 pages) = environ 180 Mbits



# Performance

---

---

- **Temps d'accès** : temps nécessaire à une opération de lecture/écriture.
- **Débit** : quantité d'informations lues/écrites par unités de temps (exemple: Mo/s).

Type	Register	Cache	Main memory	Disk
Capacity	1 KB	0,1 – 1 MB	0,1 – 100 GB	Several TB
Access time		1 ns	1 – 10 ns	1 – 10 ms
Flow rate	100 GB/s	10 GB/s	1-10 GB/s	100 MB/s

4 Go/s: 32 bits toutes les nanosecondes !

# Hiérarchie

---

---

- L'idéal serait de posséder une mémoire illimitée et très rapide !
- Mais :
  1. on ne sait pas fabriquer une mémoire illimitée,
  2. le temps d'accès augmente avec la capacité.
- L'idée : seules les données les plus utilisées nécessitent un temps d'accès très petit.

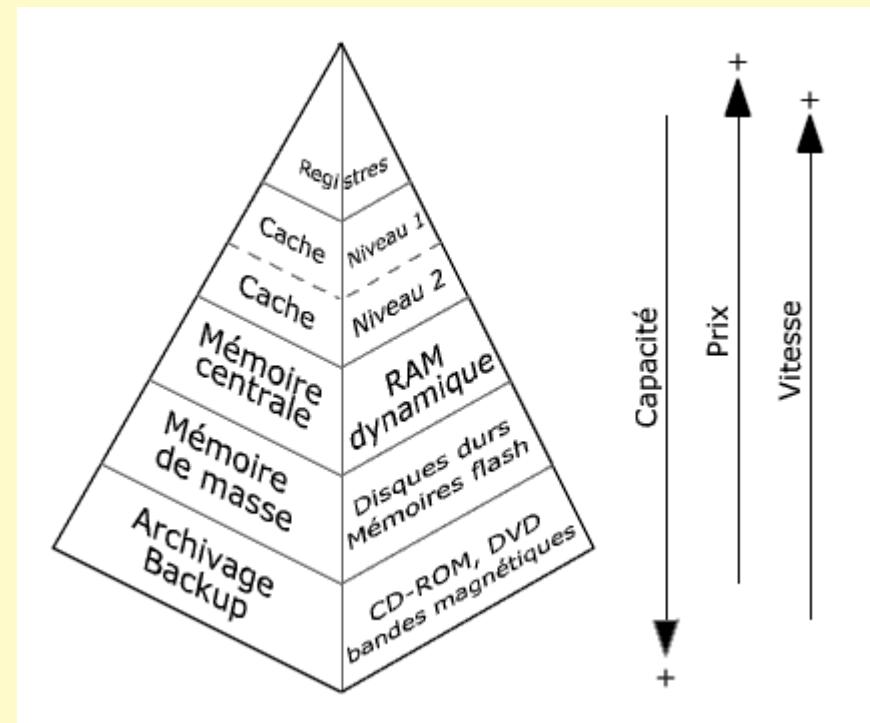
# Hiérarchie

---

---

La mémoire est organisée en une hiérarchie :

- de la mémoire la plus rapide à la moins rapide,
- de la capacité la plus faible à la capacité la plus grande,
- du composant le plus coûteux au composant le moins coûteux.



# AGENDA DU CHAPITRE

---

---

1  
Introduction

3  
Mémoire  
principale

4  
Mémoire cache

2  
Modes d'accès

5  
Mémoire  
externe

# Modes d'accès

---

---

- Accès aléatoire
- Accès par le contenu
- Accès séquentiel
- Accès direct

# Accès aléatoire

---

---

- Mode d'accès le plus employé :
  - mémoire principale
  - mémoires caches
- Chaque mot mémoire est associé à une adresse unique.
- La taille d'une adresse dépend de la capacité :
  - capacité : 4 Go (i.e.,  $4 \times 2^{30} \times 8$  bits)
  - taille du mot adressable : 1 octet
  - taille de l'adresse: 32 bits ( $\log_2(4 \times 2^{30})$ )
- Opérations associées à ce mode d'accès :
  - lecture (adr)
  - écriture (adr, donnée)
- Temps d'accès indépendant des accès précédents.

# Accès LIFO

---

---

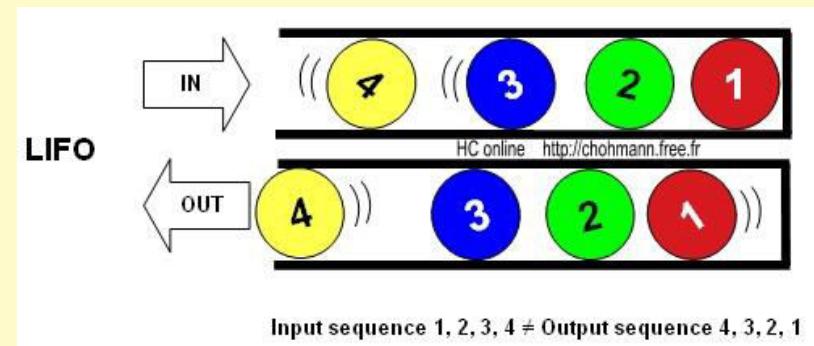
- Accès à des données résidant dans une pile.

- Utilisation :

- appel et de retour de sous programme
- sauvegarde de contexte
- suspension de programme
- interruption

- Opérations associées à ce mode d'accès :

- écriture(donnée)
- lecture
- sommet
- vide



# Exemple

---

---

opération	donnée lue
écriture(abc)	
écriture(def)	
écriture(ghi)	
lecture()	ghi
écriture(jkl)	
lecture()	jkl
lecture()	def
lecture()	abc

# Accès par le contenu

---

- Employé principalement par les mémoires caches.
- Pas de notion d'adresse !
- Un mot est retrouvé par une partie de son contenu.
- Divisée en 2 parties :
  - une partie contenant un descripteur unique (clé),
  - une partie contenant le mot associé à la clé.
- Lors d'une opération (lecture/écriture) une clé peut être comparée en parallèle avec toutes les clés stockées.

# Accès par le contenu

---

---

- Opérations associées à ce mode d'accès :
  - écriture(clé, donnée)
  - lecture(clé)
  - existe(clé)
  - retirer(clé)
- Temps d'accès constant.

# Exemple

---

---

clé	donnée
$c_3$	données <sub>3</sub>
$c_2$	données <sub>2</sub>
$c_1$	données <sub>1</sub>
$c_0$	données <sub>0</sub>

lecture( $c_x$ )

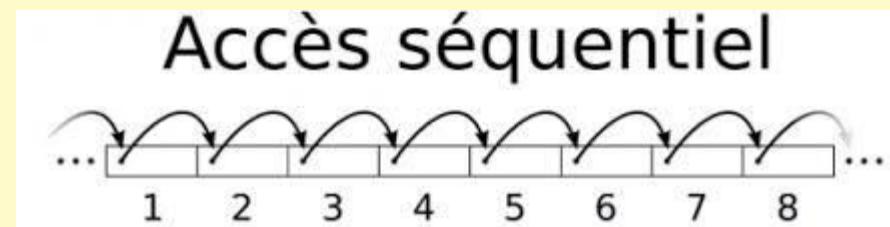
- $c_x$  est comparé simultanément avec tous les  $c_i$
- si  $c_i = c_x$  alors renvoyer données  $x$

# Accès séquentiel

---

---

- Archivage d'importants volumes de données.
- Employé par les bandes magnétiques.
- Informations écrites les une derrière les autres.
- Pour accéder à une donnée, il faut avoir lu les précédentes.



# Accès séquentiel

---

---

- Opérations associées à ce mode d'accès :
  - début : se positionner sur la première donnée
  - lecture : lire une donnée
  - écriture(données) : écrire donnée
  - fin : se positionner après la dernière donnée
- Temps d'accès variable.

# Accès direct

---

---

- Employé par les disques, les CD, ...
- Données regroupées en blocs.
- Chaque bloc est associé à une adresse unique.
- Accéder à une donnée :
  - accéder au bloc qui la contient,
  - se déplacer séquentiellement jusqu'à sa position.

# Accès direct

---

---

- Opérations associées à ce mode d'accès :
  - lecture (bloc, déplacement)
  - écriture (bloc, déplacement, donnée)
- Temps d'accès variable.

# AGENDA DU CHAPITRE

---

---

1  
Introduction

3  
Mémoire  
principale

4  
Mémoire cache

2  
Modes d'accès

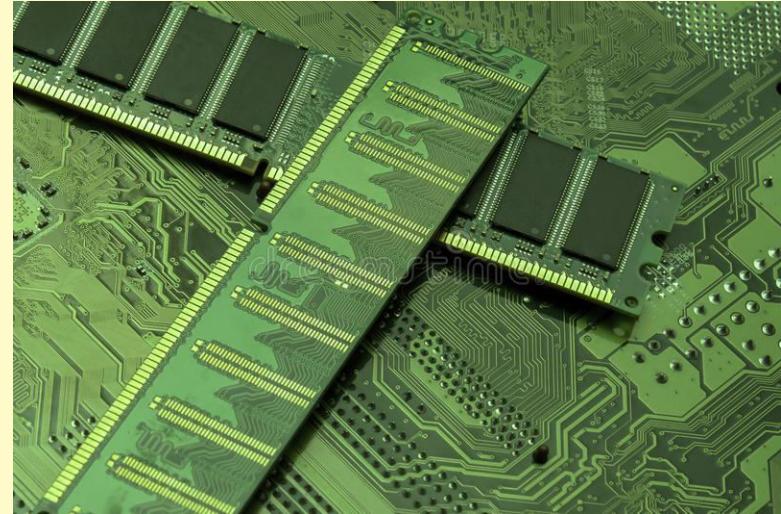
5  
Mémoire  
externe

# Définition

---

---

- Mémoire :
  - interne,
  - à base de semi-conducteurs,
  - mode d'accès aléatoire.
  
- Types :
  - mémoire volatile : RAM (*Random Access Memory*)
  - mémoire non volatile : ROM (*Read Only Memory*)



# La mémoire RAM

---

---

- Stocke des données temporaires.
- Principalement 2 types :
  - RAM dynamique (DRAM) : utilisée pour la mémoire principale,
  - RAM statique (SRAM) : principalement utilisée pour les caches.

# La mémoire RAM

---

---

- DRAM :

- condensateurs comme unités de mémorisation,
- nécessitent un rafraîchissement périodique
- simples, denses, peu couteuses

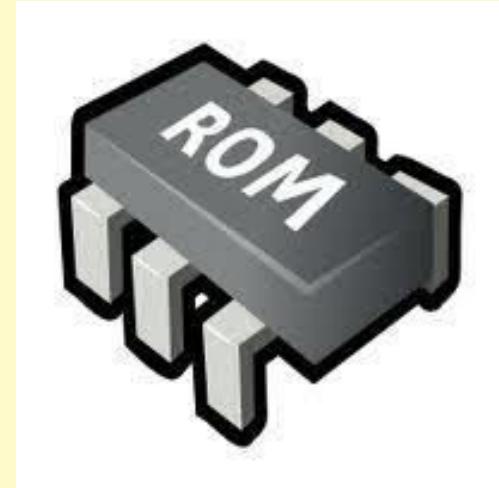
- SRAM :

- bascules comme unités de mémorisation
- rafraîchissements inutiles
- rapide, coûteuse

# La mémoire ROM

---

- *Read-Only, Read-Mostly Memory*
- Stocke des données permanentes :
  - programmes systèmes,
  - microprogrammes,
  - . . .



# La mémoire ROM

---

---

Plusieurs types :

- ROM : écriture unique lors de la fabrication
- PROM : écriture unique après fabrication
- EPROM :
  - admet un nombre d'écritures limité,
  - effaçable par ultra-violet.
- Mémoire flash : effaçable électriquement

# Synthèse

---

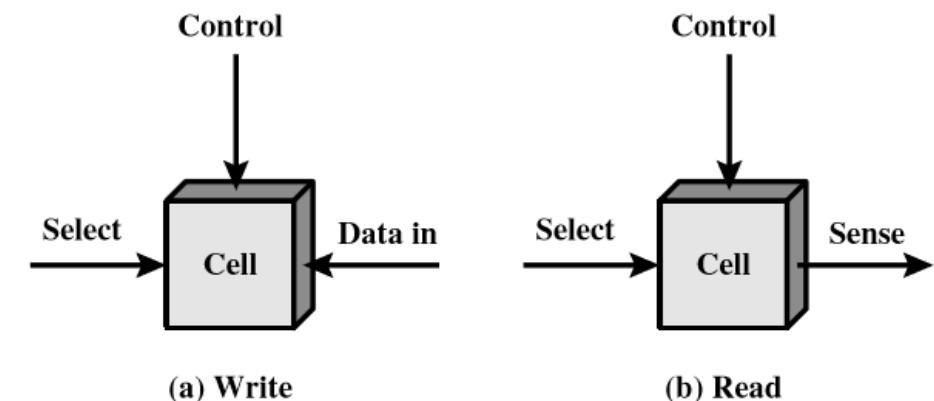
Memory Type	Category	Erasure	Write Mechanism	Volatility	
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile	
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile	
Programmable ROM (PROM)			Electrically		
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level			
Electrically Erasable PROM (EEPROM)		Electrically, byte-level			
Flash memory		Electrically, block-level			

# Organisation

---

---

- Elément de base : la cellule mémoire.
- 3 connexions :
  - une entrée de sélection,
  - une entrée de contrôle (*Output Enable* ou *Write Enable*),
  - une ligne bidirectionnelle de données.



# Organisation

---

---

- Circuit mémoire RAM de  $M$  mots de  $B$  bits
- Organisée comme une matrice de  $M$  lignes et  $B$  colonnes
- $\log^2(M)$  lignes d'adresses
- $B$  lignes de données

# Exemple

---

---

- Un espace adressable de 216 mots de 32 bits.
- 4096 mots mémoires en RAM
- 4096 mots mémoire en ROM
- Utilisation de :
  - circuits RAM de  $1024 \times 8$  bits
  - circuits ROM de  $4096 \times 8$  bits

# Exemple

---

---

- Un mot = 4 circuits de 8 bits en parallèle.
- Carte d'adresse mémoire :
  - 4096 mots de mémoires RAM aux adresses les plus basses (de 0 à 4096),
  - 4096 mots de mémoires ROM aux adresses les plus élevées (de 61440 à 65535),
  - autres adresses inoccupées.

# Exemple

---

---

- 4 étages de RAM (adresses de 0 à 4096) :
  - adresses de 0 à 1023
  - adresses de 1024 à 2047
  - adresses de 2048 à 3071
  - adresses de 3072 à 4095
- 1 étage de ROM (adresses de 61440 à 65535)

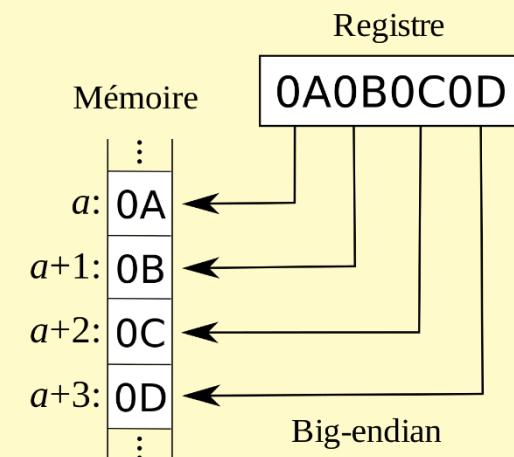
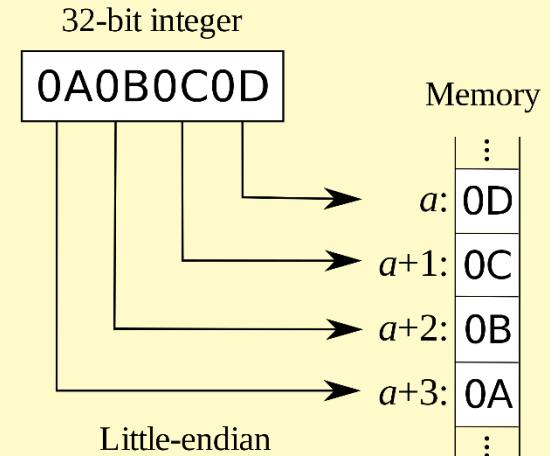
étage	bits d'adresse
RAM 0	0000 00xx xxxx xxxx
RAM 1	0000 01xx xxxx xxxx
RAM 2	0000 10xx xxxx xxxx
RAM 3	0000 11xx xxxx xxxx
inoccupé	
ROM 4	1111 xxxx xxxx xxxx

# Convention

---

---

- Adresse de l'octet de poids faible dans le mot.
- Convention petit bout (*little endian*) :
  - adresse la plus faible du mot,
  - employée par la famille i386.
- Convention gros bout (*big endian*) :
  - adresse la plus élevée du mot,
  - employée par la famille 68000.



# Convention

---

---

petit bout	gros bout
03 02 01 00	00 01 02 03
07 06 05 04	04 05 06 07
11 10 09 08	08 09 10 11
15 14 13 12	12 13 14 15
...	...

# Mémoire dynamique DRAM

---

---

- Nécessite un rafraîchissement.
  - Grande capacité.
  - Organisée en matrice.
  - Multiplexage du bus adresse.
- 
- Base de la mémoire principale depuis plus de 20 ans.

# SDRAM

---

---

- Evolution synchrone de la DRAM
- La DRAM standard est asynchrone :
  - demande de lecture/écriture,
  - attente temps d'accès du circuit DRAM,
  - confirmation du succès de l'opération.
- Echange des données avec le processeur en se basant sur un signal d'horloge externe :
  - informations à traiter stockées dans des registres internes
  - répond après un nombre de cycles d'horloge fixé.

# SDRAM

---

---

Mode d'accès “en rafale” (*burst mode*)

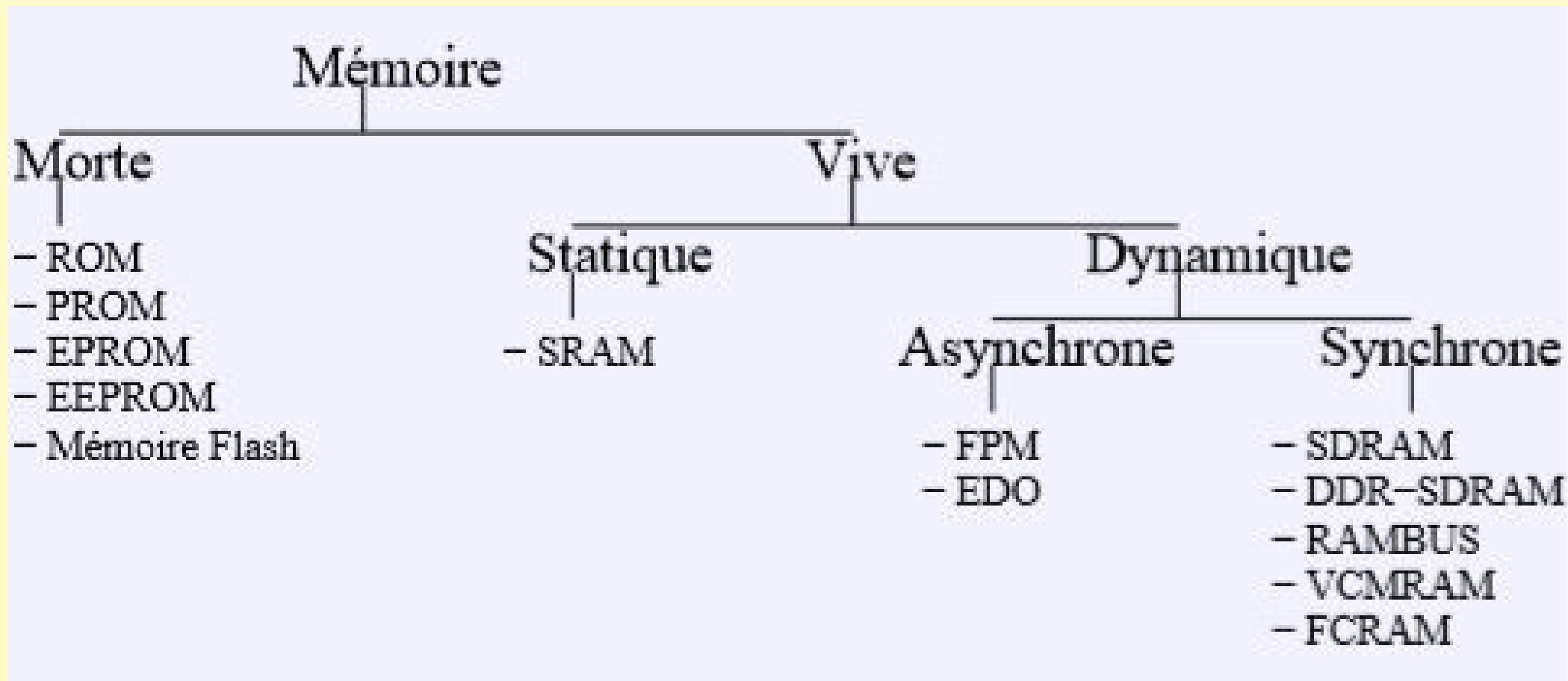
:

- lecture/écriture de données d'adresses contiguës
- seule l'adresse de début d'une séquence de mots est donnée
- pas de décodage des adresses suivantes



# Synthèse

---



# AGENDA DU CHAPITRE

---

---

1  
Introduction

3  
Mémoire  
principale

4  
Mémoire cache

2  
Modes d'accès

5  
Mémoire  
externe

# Mémoire cache

---

---

- Augmentation des performances :
  - microprocesseurs : environ 55% par an
  - mémoire : environ 7% par an
- Comment compenser cette différence ?



# Définition

---

---

Niveau de mémorisation :

- intermédiaire,
- rapide,
- de petite capacité,
- stockant les données les plus récemment accédées.

Accès moins couteux que l'accès à la mémoire principale.

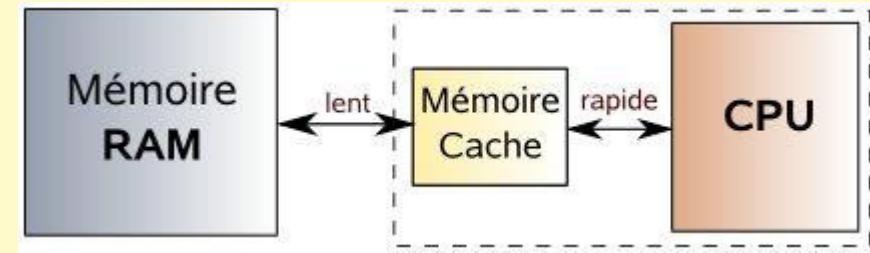
# Définition

---

---

Typiquement située entre :

- le processeur et la mémoire principale
- le processeur et un autre cache
- le processeur et un disque
- etc...



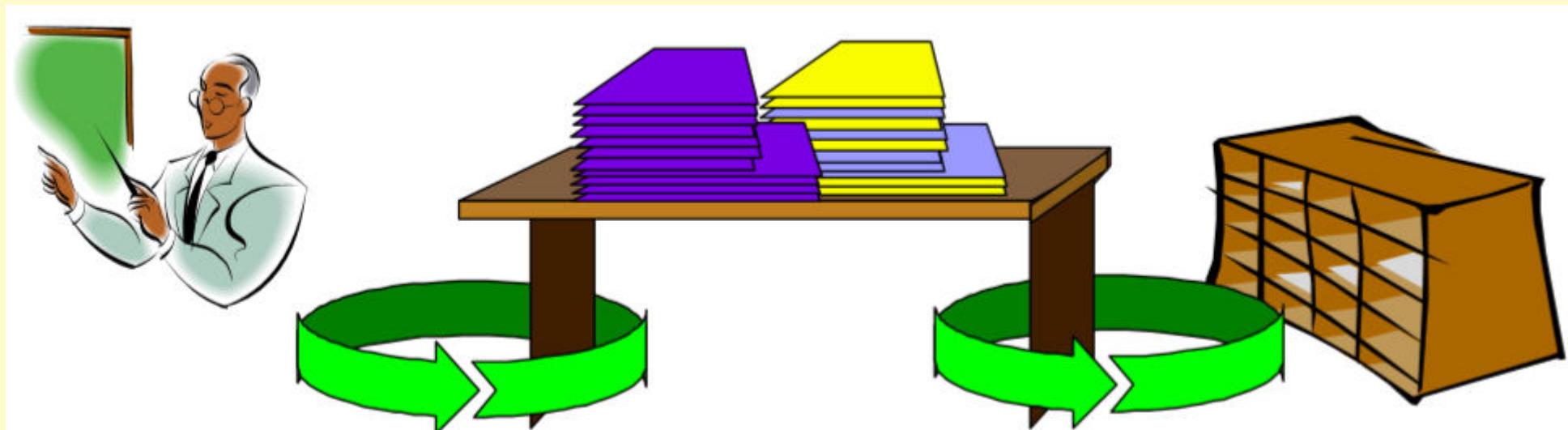
Découpée en **blocs** : ensemble de mots d'adresses contiguës.

# Analogie

---

---

- L'homme = unité de calcul
- Le bureau = mémoire cache
- La bibliothèque = mémoire centrale



# Principe

---

---

- Rechercher dans le cache avant de rechercher dans la mémoire principale.
- **Succès cache** : la donnée est présente dans le cache et il n'y a pas d'accès à la mémoire principale.
- **Défaut de cache** : la donnée est absente du cache et il y a accès à la mémoire principale.

# Principe

---

---

Mémoire principale découpée en blocs de même taille.

Accès à une adresse mémoire :

- si le bloc contenant l'adresse est dans le cache
  - la donnée à l'adresse est lue
- sinon
  - le bloc contenant l'adresse est copié de la mémoire principale dans le cache
  - la donnée à l'adresse est lue

# Caractéristiques d'une mémoire cache

---

---

- Nombre de caches
- Localisation
- Contenu
- Taille
- Correspondance
- Accès
- Remplacement
- Politique d'écriture

# Nombre de caches et localisation

---

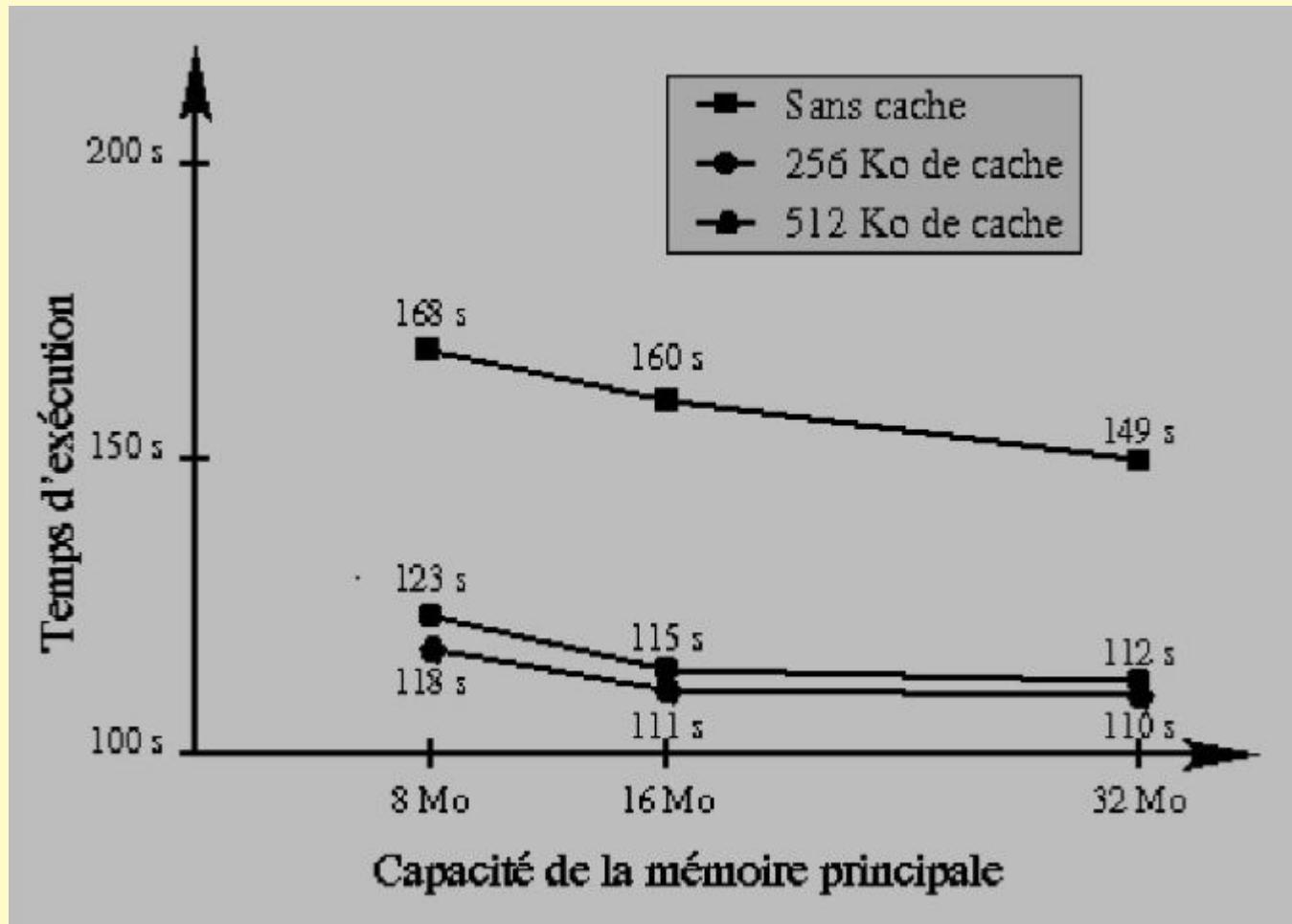
---

- Utilisation de plusieurs caches :
  - situés dans le processeur (**on-chip / on-die / internal**) : l'utilisation d'un cache interne permet d'augmenter les performances et de laisser le bus externe disponible,
  - accessible via un bus externe au processeur (**external**).
- Organisés en niveaux (L1, L2, . . .) :
  - un ou plusieurs cache de niveau 1 (interne),
  - un cache de niveau 2 (interne ou externe),
  - parfois un cache de niveau 3 (externe),
  - chaque niveau fonctionnant à une vitesse différente : on admet généralement que  $|L2| \geq 4 \times |L1|$ .

# Nombre de caches et localisation

---

---



# Contenu du cache

---

---

- Cache interne :
  - un cache dédié au stockage des instructions,
  - un cache dédié au stockage des données.
- Le processeur pourra exécuter en parallèle :
  - la recherche des instructions,
  - l'exécution des instructions.

# Taille du cache

---

---

- Suffisamment petit :
  - coût peu élevé,
  - temps d'accès le plus intéressant possible.
- Suffisamment grand :
  - éviter les défauts de cache.
- Les performances dépendent beaucoup de la nature des applications.

# Correspondance

---

---

- Taille du cache << taille de la mémoire.
- 3 stratégies de copie des blocs mémoire dans le cache :
  - **correspondance directe** : 1 emplacement unique dans le cache pour un bloc mémoire
  - **correspondance totalement associative** : 1 bloc mémoire peut être placé n'importe où dans le cache
  - **correspondance associative par ensemble** : 1 bloc mémoire peut être placé dans n'importe quel bloc du cache parmi un ensemble de  $n$  blocs
- La majorité des caches est soit par correspondance directe, soit par correspondance associative par ensemble de 2 ou 4 blocs.

# Accès à un bloc du cache

---

- Construction des adresses mémoires en fonction de la correspondance.
- L'adresse mémoire d'un mot permet de trouver :
  - le bloc auquel elle appartient,
  - sa place dans le bloc,
  - la place de ce bloc dans le cache.

# Adresse mémoire d'un mot

---

---

Décomposée en deux parties :

- un numéro de bloc :
  - un **index** : l'emplacement du bloc dans le cache
  - une **étiquette** : identifie le bloc parmi les blocs destinés au même emplacement
- un **déplacement**: l'adresse du mot dans le bloc.

# Adresse mémoire d'un mot

---

---

Le cache maintient une table d'étiquettes :

emplacement de bloc 1 étiquette de bloc mémoire 1

emplacement de bloc 2 étiquette de bloc mémoire 2

.....

emplacement de bloc n étiquette de bloc mémoire  $n$

$n$  = taille du cache / taille de bloc

# Algorithme de remplacement

---

---

Problème lors :

- d'un défaut de cache
- d'une correspondance associative

Quel bloc du cache va recevoir le nouveau bloc mémoire ?

# Algorithme de remplacement

---

Diverses stratégies sont employées :

- choisir un bloc candidat de manière aléatoire
- choisir le plus ancien bloc du cache (FIFO, *First In First Out*)
- choisir le bloc le moins récemment utilisé (LRU, *Least Recently Used*)
- choisir le bloc le moins fréquemment utilisé (LFU, *Least Frequently Used*)

# Algorithme de remplacement

---

---

Les plus efficaces :

- LFU
- LRU
- aléatoire

Les plus faciles à implanter :

- aléatoire
- FIFO

# Politique d'écriture

---

Problème lors :

- d'une opération d'écriture en mémoire
- de la présence dans le cache du mot à écrire

Où écrire ?

# Politique d'écriture

---

Plusieurs méthodes :

- écriture simultanée (*write through*) :
  - écrire dans le bloc du cache
  - réécrire dans le bloc de la mémoire
- réécriture (*write back*) :
  - écrire uniquement dans le bloc du cache
  - attendre que l'emplacement soit réquisitionné
  - écrire ce bloc en mémoire

# Politique d'écriture

---

Et si le bloc n'est pas dans le cache ?

- écriture allouée :
  - charger le bloc de la mémoire dans le cache
  - effectuer l'opération d'écriture
- écriture non allouée :
  - effectuer l'écriture directement dans la mémoire

# Performance

---

---

- **Temps d'accès mémoire moyen**  
= temps d'accès succès + taux d'échec × pénalité d'échec
- **Temps d'accès succès** = temps d'accès à une donnée du cache
- **Taux d'échec** = nombre de défauts de cache / nombre d'accès cache

# Exemple

---

---

Exécution d'une instruction :

- décodage de l'instruction
- recherche des données en mémoire
- déclenchement des opérations sur ces données

Durée d'un cycle horloge :  $\tau$

Pénalité d'échec : 10 cycles

Durée d'une instruction : 2 cycles  
(sans référence mémoire)

Nombre de références mémoire : 1,33 par instruction

Taux d'échec : 2%

Temps d'accès succès : négligeable

Temps d'exécution moyen d'une instruction =  $(2 + 1,33 \times 2\% \times 10) \times \tau = 2,27\tau$

Et s'il n'y a pas de cache : temps d'exécution moyen d'une instruction =  $(2 + 1,33 \times 10) = 15,3\tau$

# AGENDA DU CHAPITRE

---

---

1  
Introduction

3  
Mémoire  
principale

4  
Mémoire cache

2  
Modes d'accès

5  
Mémoire  
externe

# Différents types de mémoire externe

---

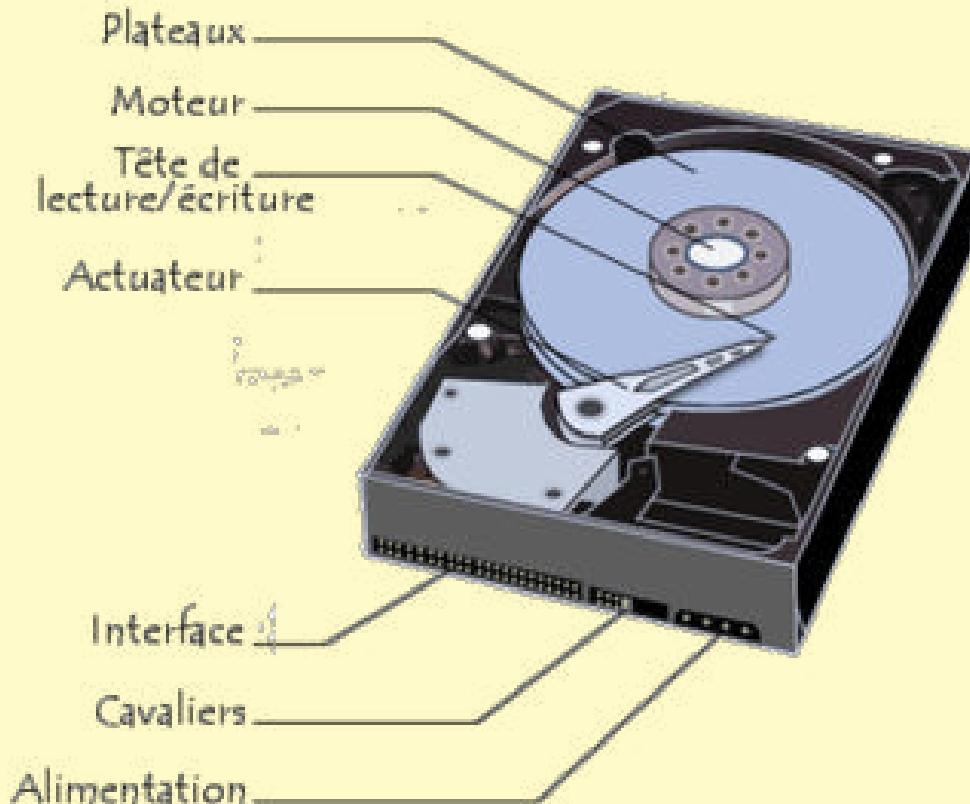
- Disque magnétique
- Mémoire optique
- Bande magnétique
- Mémoire flash externe



# Le disque magnétique

---

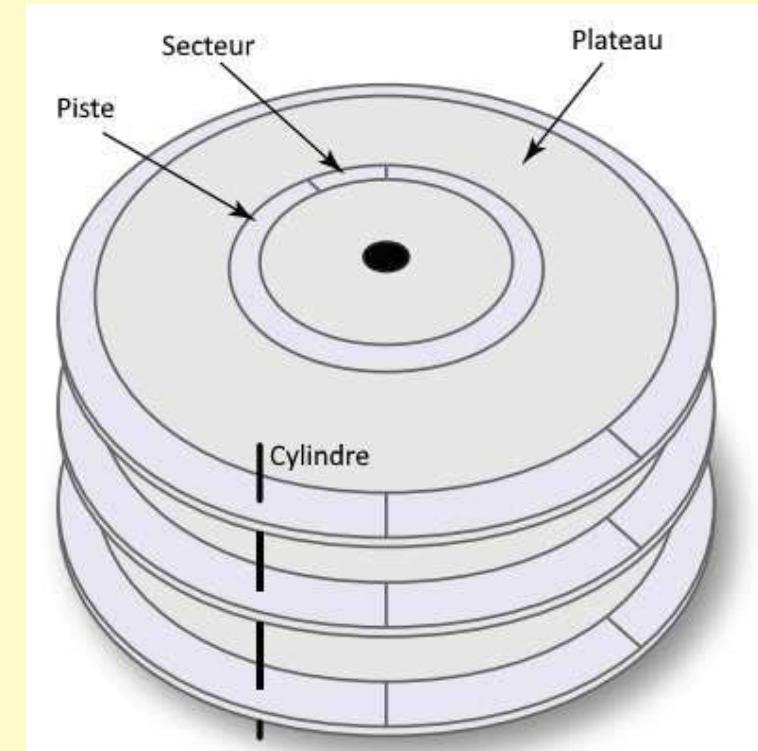
- Le disque dur est un organe de stockage de grande capacité.
- Il peut contenir plusieurs centaines de milliards d'octets.
- Les informations sont stockées de manière magnétique sur des disques en rotation.



# Le disque magnétique

---

- Un disque est un plateau rond composé d'un matériau non magnétique (substrat) recouvert d'un matériau magnétique.
- Sur le plateau les données sont organisées en anneaux concentriques : **pistes**.
- Une surface contient des milliers de pistes.
- Les pistes sont divisées en **secteurs**.
- Il existe quelques centaines de secteurs par piste.
- La longueur d'un secteur est fixe : 512 octets.
- Les pistes d'une surface de plateau constituent un **cyclindre**.



# Anatomie d'un disque dur

---

---

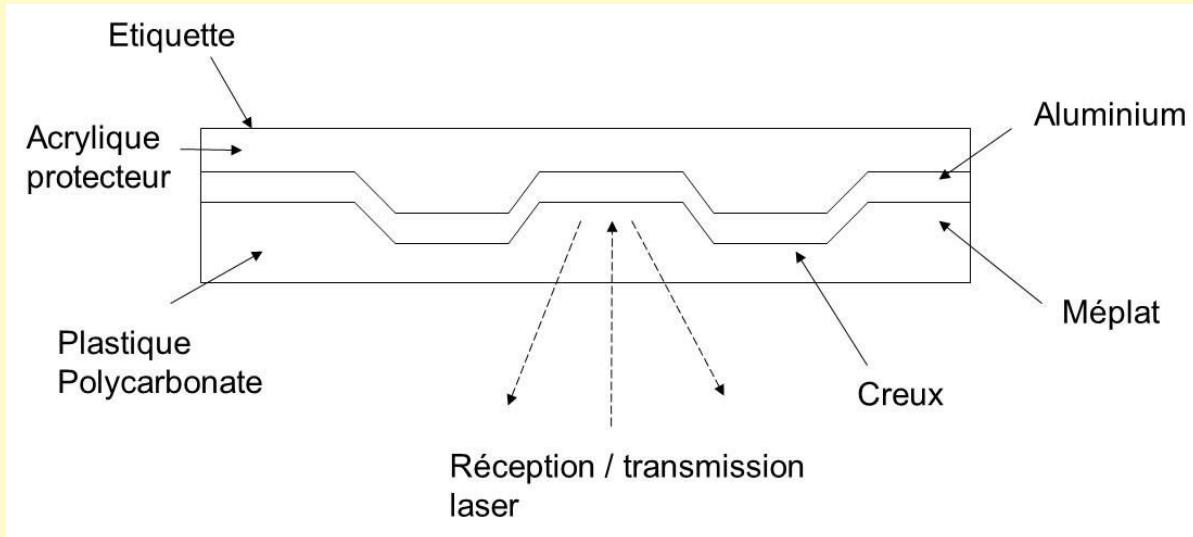
- Des bras déplacent des têtes d'enregistrement-lecture au dessus des disques magnétiques.
- Celles-ci volent sur l'air entraîné par les disques en rotation.



# Les mémoires optiques

---

---



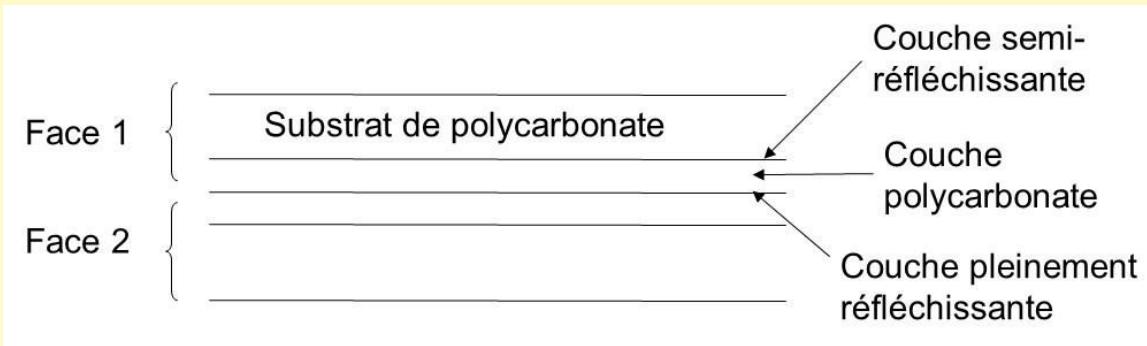
Le CD

- Un moteur fait tourner le disque sous un laser faible puissance :
  - Creux → lumière dispersée : faible intensité en retour.
  - Méplats → lumière d'intensité plus grande en retour.
- Un capteur détecte les changements entre creux et méplats.
- Les données sont organisées en spirale et non en disques : on balaie selon une vitesse **linéaire** constante et non plus angulaire.

# Les mémoires optiques

---

---



Le  
DVD

- Les bits sont disposés plus étroitement que sur un CD.
- Seconde couche de creux et de méplats sur la première couche :
  - en ajustant la focale des lecteurs,
  - lecture couches semi-réfléchissante et réfléchissante individuellement.
- Un DVD peut posséder 2 faces.

# Les bandes magnétiques



- Même principe que les disques magnétiques.
  - Bande polyester souple recouverte d'une pellicule de matériau magnétique.
  - Données organisées en pistes parallèles (18 à 36 pistes parallèles).
  - Enregistrement en serpentin :
    - une piste en aller,
    - l'autre piste en retour dans l'autre sens et ainsi de suite.
  - La tête peut lire et écrire sur plusieurs pistes en même temps.
  - L'accès aux données sur bande est séquentiel.
  - C'est le stockage le moins couteux encore aujourd'hui.

# Autres mémoires externes

---

---

- Clé USB :

- mémoire à semi-conducteurs (puce),
- EEPROM
- effaçable octet par octet.



- Mémoire flash externe :

- mémoire flash,
- effaçable bloc par bloc.





# ARCHITECTURE DES ORDINATEURS

## CHAPITRE 4 LE JEU D'INSTRUCTIONS



# AGENDA DU CHAPITRE

---

---

1

Le langage  
machine

3

Jeu  
d'instructions

4

TraITEMENT des  
instructions

2

Eléments  
d'architecture

5

Un petit  
benchmark

# AGENDA DU CHAPITRE

---

---

1  
Le langage  
machine

4  
Traitement des  
instructions

3  
Jeu  
d'instructions

5  
Un petit  
benchmark

2  
Eléments  
d'architecture

# Communiquer avec l'ordinateur

---

---

- Un langage de programmation **bas niveau** est un langage qui dépend fortement de la structure matérielle de la machine.
- Cette caractéristique offre des avantages et des inconvénients divers.
- Par exemple, elle permet :
  - d'avoir un bon contrôle des ressources matérielles,
  - d'avoir un contrôle très fin sur la mémoire,
  - souvent, de produire du code dont l'exécution est très rapide.
- En revanche, elle ne permet pas :
  - d'utiliser des techniques de programmation abstraites;;
  - de programmer rapidement et facilement.

# Le langage machine

---

- Le **langage machine** est un langage compris directement par un processeur donné en vue d'une exécution.
- C'est un langage binaire : ses seules lettres sont les bits 0 et 1.
- Chaque modèle de processeur possède son propre langage machine.
- Étant donnés deux modèles de processeurs  $x_1$  et  $x_2$ , on dit que  $x_1$  est compatible avec  $x_2$  si toute instruction formulée pour  $x_2$  peut être comprise et exécutée par  $x_1$ .



# Le langage machine

---

---

- Dans la plupart des langages machine, une instruction commence par un opcode, une suite de bits qui porte la nature de l'instruction.
  - Celui-ci est suivi des suites de bits codant les opérandes de l'instruction.
- 
- Par exemple, la suite **01101010 00010101** est une instruction dont le opcode est **01101010** et l'opérande est **00010101**.
    - Elle ordonne de placer la valeur  $(21)_{10}$  en tête de la pile.

# Langages d'assemblage

---

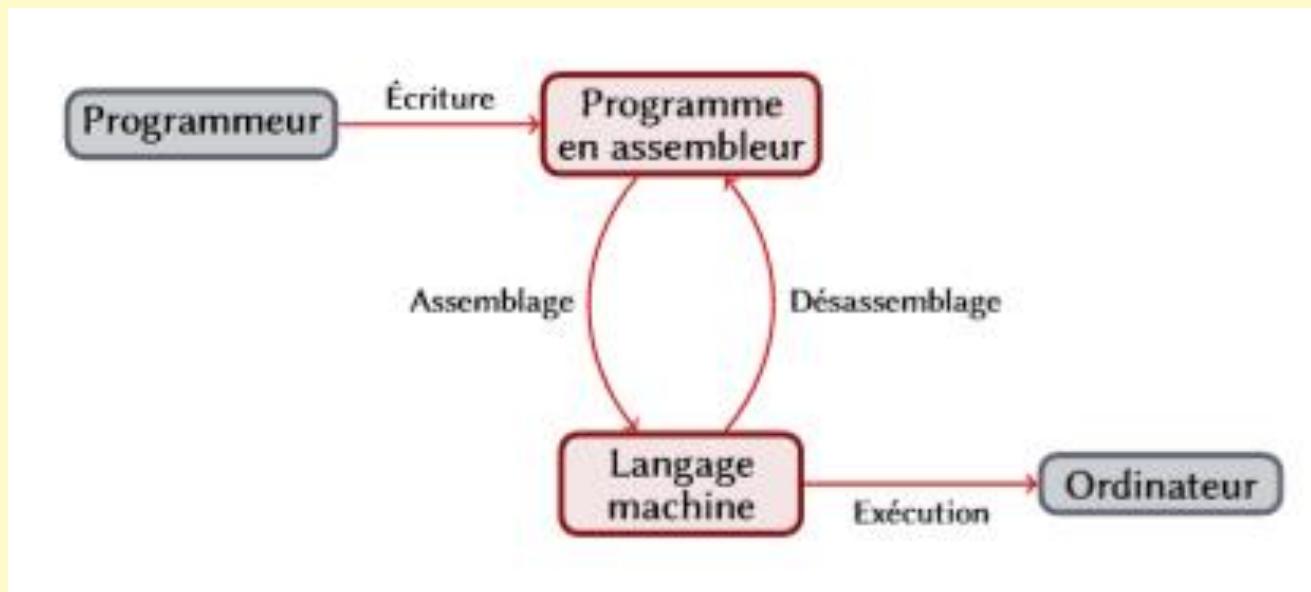
---

- Un **langage d'assemblage** (ou **assembleur**) est un langage qui se trouve à mi-distance entre le programmeur et la machine en terme de facilité d'accès.
- En effet, d'un côté, la machine peut convertir presque immédiatement un programme en assembleur vers du langage machine.
- De l'autre, l'assembleur est un langage assez facile à manipuler pour un humain.
- Les opcodes sont codés via des **mnémoniques**, mots-clés bien plus manipulables pour le programmeur que les suites binaires associées.
- Du fait qu'un langage d'assemblage est spécifiquement dédié à un processeur donné, il existe presque autant de langages d'assemblage qu'il y a de modèles de processeurs.

# Langages d'assemblage et assembleurs

---

- L'**assemblage** est l'action d'un programme nommé **assembleur** qui consiste à traduire un programme en assembleur vers du langage machine.
- Le **désassemblage**, réalisé par un **désassembleur**, est l'opération inverse : elle permet de retrouver, à partir d'un programme en langage machine, un programme en assembleur qui lui correspond.



# Langages de haut et bas niveaux

---

---

- Compilation d'un langage haut niveau (**HLL**) vers un langage de bas niveau (**LLL**).
- Une instruction **HLL** correspond à plusieurs instructions **LLL**.
- Le jeu d'instruction doit être suffisamment expressif pour coder toute instruction d'un langage de haut niveau.

# Jeu d'instructions

---

---

- Ensemble des instructions exécutables par une CPU.
- Une instruction doit contenir :
  - le code de l'opération à exécuter (*opcode*)
  - la référence aux opérandes sources
  - la référence à l'opérande résultat
  - la référence à la prochaine instruction
- Certaines de ces informations peuvent être implicites.

# Représentation de l'instruction

---

---

- Séquence de bits découpée en champs.
- Représentation symbolique :
  - l'*opcode* est représenté par une abréviation (*mnémonique*),
  - les registres sont représentés par leur nom,
  - les adresses mémoires par leur valeur ou un nom de variable
  - des symboles particuliers pour les contenus, les modes d'adressage, ...

# Exemple

---

---

adresse mémoire	contenu			
0100	0010	0010	0000	1100
0101	0001	0010	0000	1101
0110	0001	0010	0000	1110
0111	0011	0010	0000	1111
1100	0000	0000	0000	0010
1101	0000	0000	0000	0011
1110	0000	0000	0000	0100
1111	0000	0000	0000	0000

# Exemple

---

---

adresse	contenu
0100	LOAD (1100)
0101	ADD (1101)
0110	ADD (1110)
0111	STORE (1111)
1100	0002
1101	0003
1110	0004
1111	0000

## Exemple

---

---

$X = X + Y$  se traduit en

```
LOAD  X, R1
ADD   R1, Y
STORE R1, X
```

# AGENDA DU CHAPITRE

---

---

1  
Le langage  
machine

4  
Traitement des  
instructions

3  
Jeu  
d'instructions

2  
Eléments  
d'architecture

5  
Un petit  
benchmark

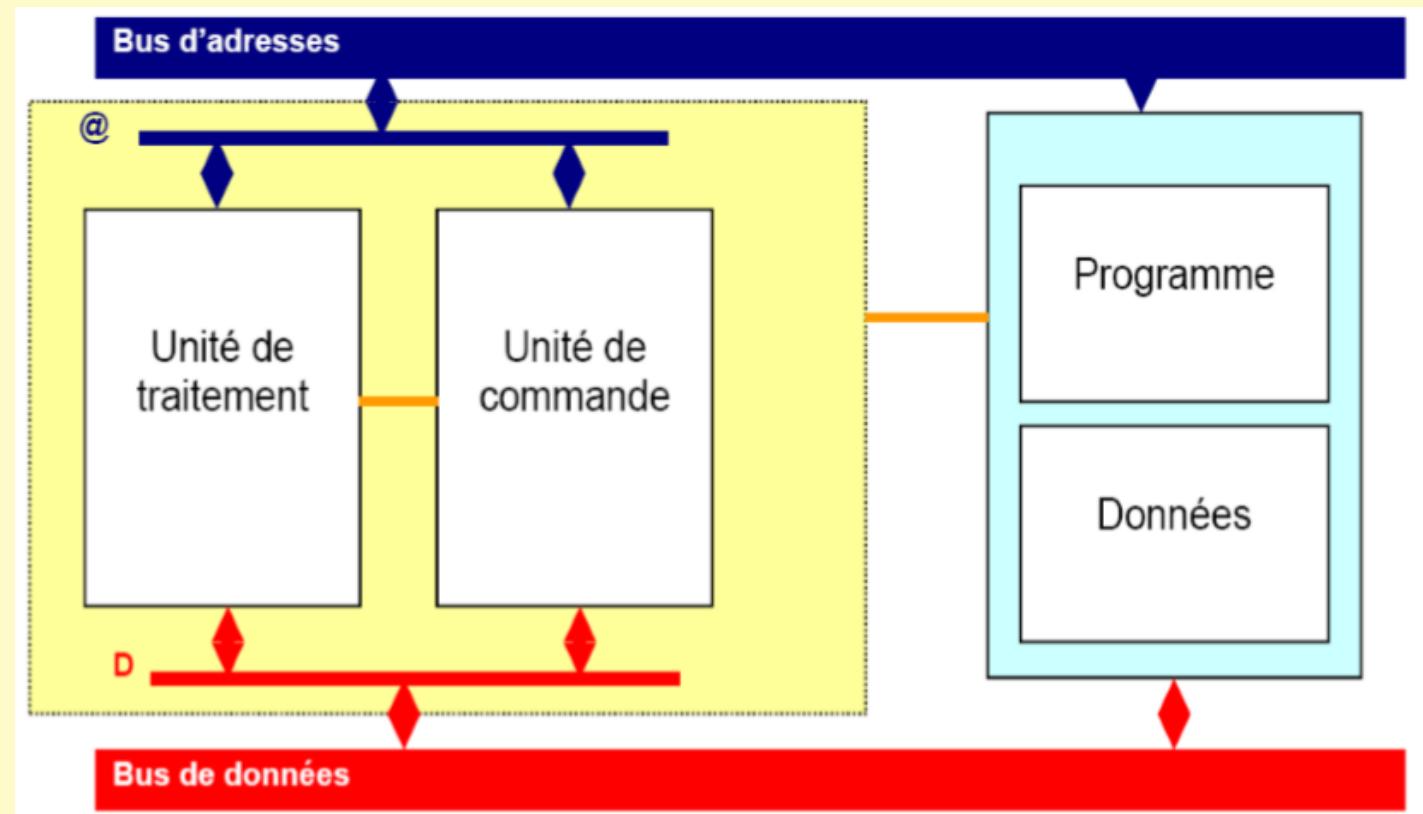
# Anatomie d'un processeur

---

---

Un microprocesseur est construit autour de deux éléments principaux :

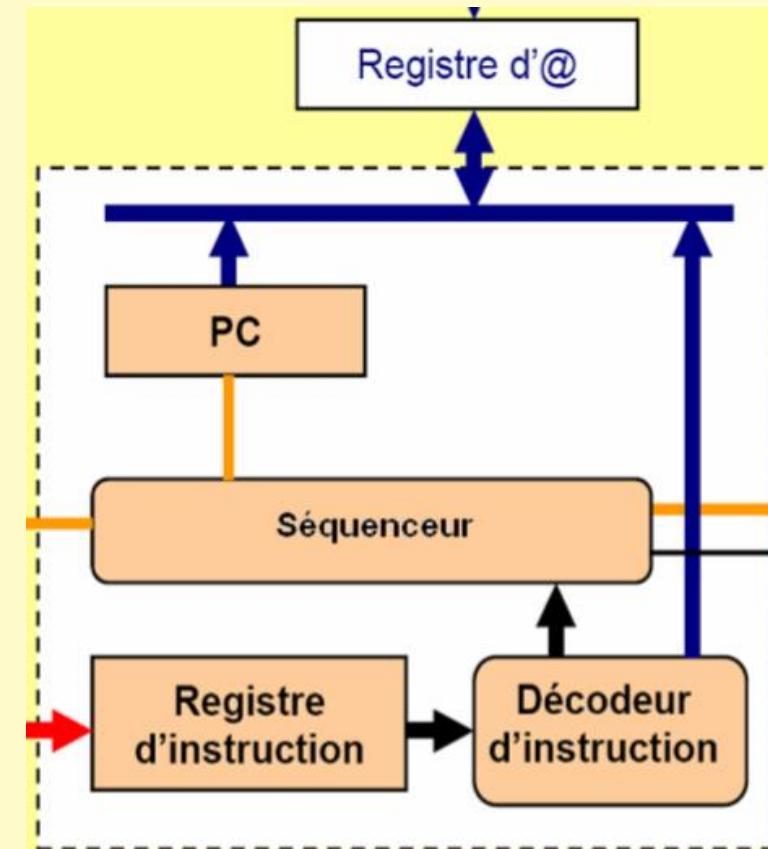
- une unité de commande,
- une unité de traitement.



# L'unité de commande

---

- Elle permet de séquencer le déroulement des instructions.
- Elle effectue la recherche en mémoire de l'instruction, le décodage de l'instruction codée sous forme binaire.
- Enfin elle pilote l'exécution de l'instruction.

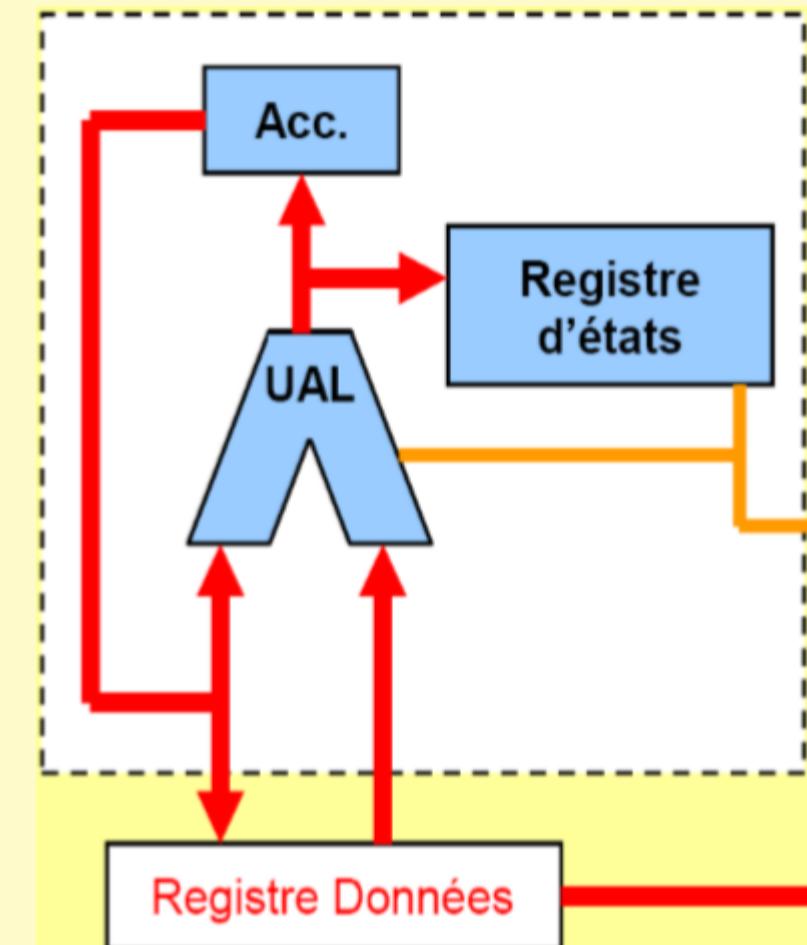


# L'unité de traitement

---

Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions :

- les **accumulateurs** sont des registres de travail qui servent à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.
- l'**Unité Arithmétique et Logique (UAL)** est un circuit complexe qui assure les fonctions logiques (ET, OU, comparaison, décalage, etc...) ou arithmétiques (addition, soustraction...).
- le **registre d'état** est généralement composé de 8 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle **indicateur d'état** ou **flag** ou **drapeaux**.



# Un registre plus précisément ...

---

- Registre = mots mémoire internes au processeur
- Les registres de fonctionnement :
  - Compteur Ordinal (CO),
  - Registre Instruction (RI), ...
  - Accumulateur
    - registres internes à l'ALU
    - stockent les opérandes et le résultat d'un calcul
- Les registres généraux :
  - registres accessibles par le programme,
  - servent à stocker des valeurs souvent utilisées et des résultats intermédiaires sans avoir besoin de repasser par la mémoire,
  - gain de performance et de temps.

# AGENDA DU CHAPITRE

---

---

1

Le langage  
machine

3

Jeu  
d'instructions

4

TraITEMENT des  
instructions

2

Eléments  
d'architecture

5

Un petit  
benchmark

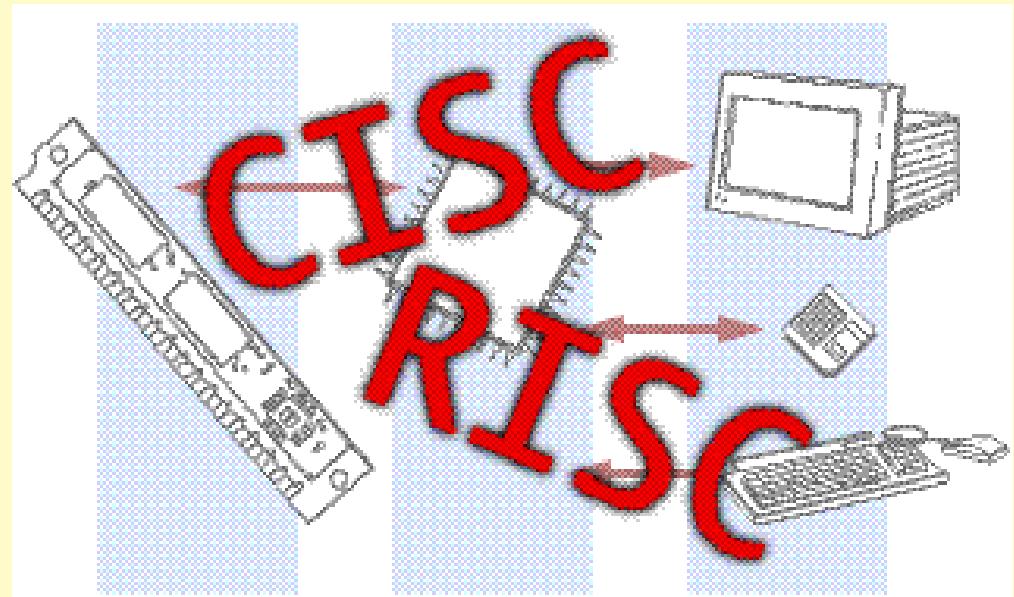
# Conception d'un jeu d'instruction

---

---

Compromis entre :

- nombre d'opérations
- complexité des opérations
- nombre de types de données
- nombre de registres
- utilisation des registres
- nombre de modes d'adressage
- nombre de champs
- taille des champs



# Classification et format

---

---

## Classification

- Selon le format d'instruction.
- Selon la relation entre mémoire et registres.

## Format

- Nombre de champs réservés aux adresses.
- En théorie :
  - premier opérande source,
  - deuxième opérande source,
  - opérande résultat,
  - prochaine instruction.

## En pratique

---

---

- Format 3 adresses : peu courant (instructions longues).
- Format 2 adresses : une adresse fait office de source et de destination.
- Format 1 adresse : l'accumulateur garde une opérande et le résultat.
- Format 0 : une pile stocke opérandes et résultats.

# Nombre d'adresses

---

---

nombre d'adresses	représentation	interprétation
3	OP A, B, C	$A \leftarrow B \text{ OP } C$
2	OP A, B	$A \leftarrow A \text{ OP } B$
1	OP A	$\text{ACC} \leftarrow \text{ACC OP } A$
0	OP	$T \leftarrow T \text{ OP } T - 1$

ACC : l'accumulateur

T : sommet de pile

# Choix du nombre d'adresses

---

---

Moins il y a d'adresses :

- plus les instructions sont courtes,
- moins la CPU est complexe.

Plus les instructions sont nombreuses :

- plus les programmes sont lents à s'exécuter.

Actuellement : mélange des formats 2 adresses et 3 adresses.

# Relation mémoire & registres

---

---

- Disposer de registres permet de minimiser les accès à la mémoire et d'accélérer l'exécution.
- Registre-registre (chargement-rangement) :
  - opérations LOAD et STORE
  - instructions simples
  - exécution rapide (nombre de cycles fixe)
  - nombre d'instructions générées important
  - très utilisé

# Catégories d'instructions

---

---

- Transfert de données
- Opérations arithmétiques :
  - entiers signés
  - flottants
- Logique
- Conversion
- Entrées / Sorties
- Contrôle système
- Transfert de contrôle (test et branchement)
  - test du registre condition

# Exemple

---

---

```
i = 0;  
while(i < 10) i ++;  
...  
STORE i, 0  
LOAD R1, 10  
LOAD R2, i  
N1 ADD R2, R2, 1  
SUB R3, R1, R2  
JNZ N1  
...
```

# Catégories d'opérandes

---

---

- Adresses
- Nombres :
  - entiers (*fixed-point*)
  - flottants (*floating point*)
  - DCB
- Caractères (ASCII)
- Données logiques
- Voire des types de données plus évoluées (liste, chaîne de caractères)

# Modèles d'accès mémoire

---

---

- Le processeur exécute des opérations avec des opérandes comme paramètres.
- Plusieurs combinaisons possibles.
- Exemple sur une opération d'addition :  $A = B + C$ 
  - A, B et C étant des valeurs se trouvant en mémoire centrale
  - peut-on y accéder directement ?
  - doit-on les placer avant dans des registres généraux ?
  - doit-on les placer dans l'accumulateur de l'UAL ?

# Quelques modes d'adressage classique

---

---

registre	ADD R4, R3
immédiat	ADD R4, 3
direct	ADD R4, (0011)
indirect par registre	ADD R4, (R3)
indirect par mémoire	ADD R4, @(0011)
avec déplacement	ADD R4, (R3)100

# Problème lié à l'adressage

---

---

- Comment adresser un espace mémoire important en conservant une taille d'instruction raisonnable ?
- Adressage indirect par mémoire :
  - plusieurs références à la mémoire
- Adressage indirect par registre
- Adressage avec déplacement :
  - flexible
  - complexe

# Taille d'une instruction

---

---

- Plus le jeu d'instructions est complexe
  - Plus la quantité d'instructions dans un programme est faible
  - Plus la longueur d'instruction est importante
  - Plus la consommation en espace est importante
- 
- La longueur d'une instruction doit prendre en considération :
    - la taille et l'organisation de la mémoire,
    - la structure d'interconnexion,
    - la complexité et la vitesse de la CPU.

# Taille d'une instruction

---

---

La taille de l'instruction peut être :

- fixe
- variable
  - autorise une large gamme d'*opcodes* (de tailles différentes)
  - une plus grande flexibilité d'adressage
  - accroît la complexité de la CPU

# Allocation des bits

---

---

Le découpage en champs dépend :

- du nombre d'opérandes
- du nombre d'opérations
- du nombre de modes d'adressage
- de l'utilisation des registres
- de l'espace adressable
- de la façon d'adresser la mémoire

# Taille de l'*opcode*

---

---

Si on souhaite à la fois :

- une taille d'instruction raisonnable,
  - une capacité d'adressage raisonnable,
  - un nombre d'*opcodes* important,
- on peut utiliser une taille d'*opcode* variable.

# AGENDA DU CHAPITRE

---

---

1

Le langage  
machine

3

Jeu  
d'instructions

4

Traitemen<sup>t</sup> des  
instructions

2

Éléments  
d'architecture

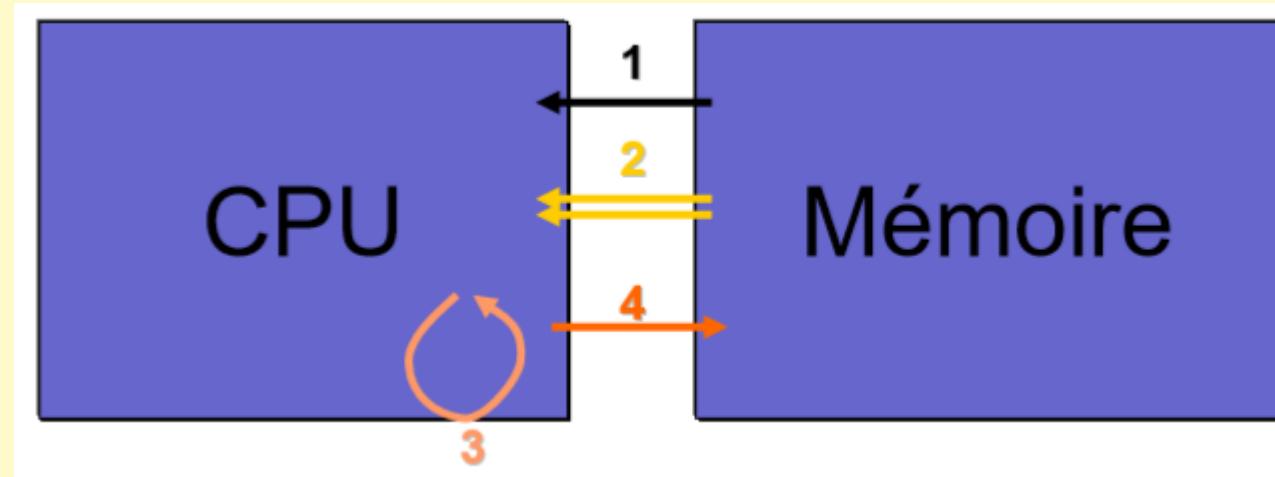
5

Un petit  
benchmark

# Fonctionnement basique d'une opération de calcul

---

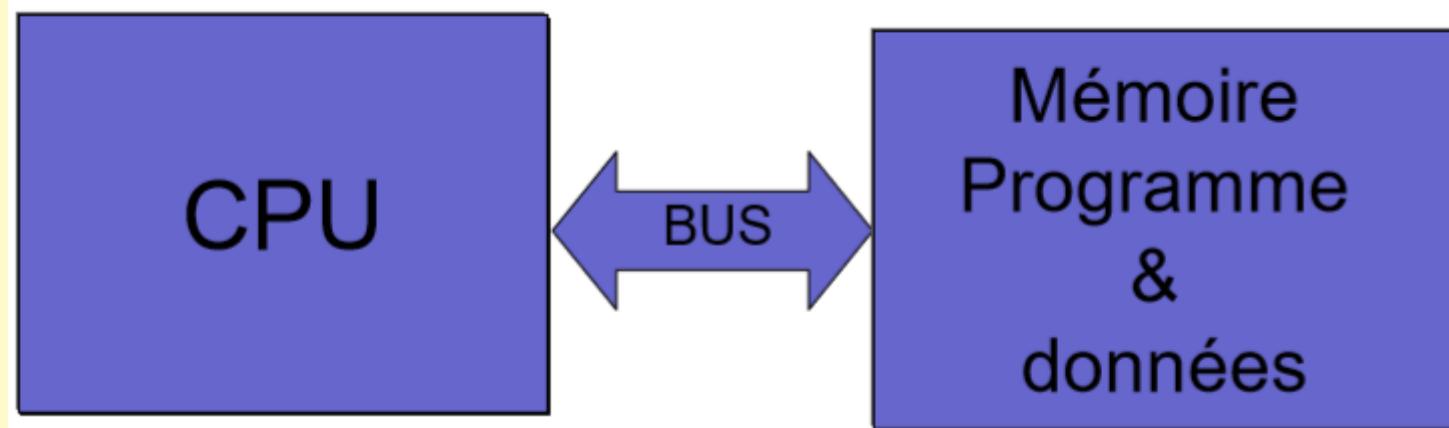
---



- (1) Charger une instruction depuis la mémoire
- (2) Charger les opérandes depuis la mémoire
- (3) Effectuer les calculs
- (4) Stocker le résultat en mémoire

# L'architecture Von Neumann

---

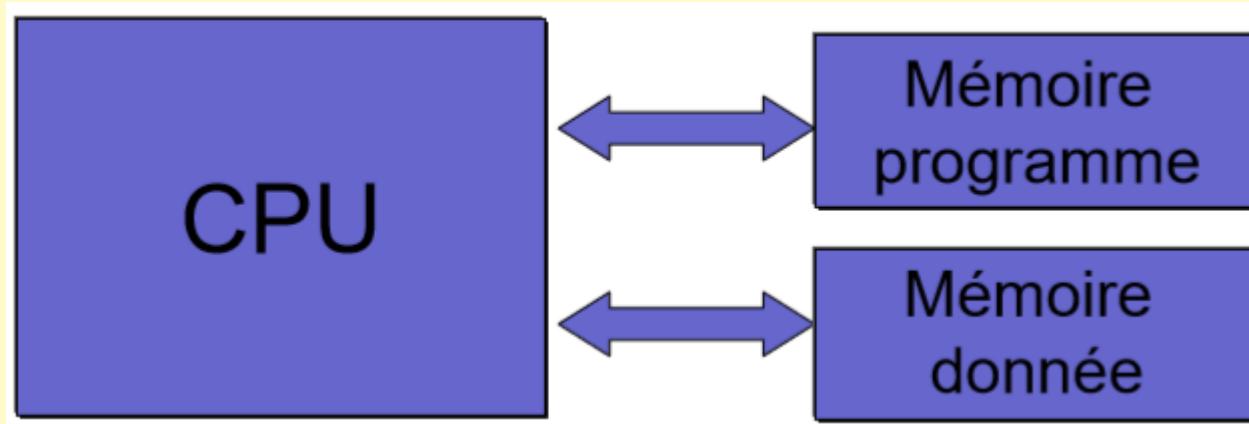


- Un seul chemin d'accès à la mémoire :
  - un bus de données (programme et données),
  - un bus d'adresse (programme et données).
- Architecture des processeurs d'usage général
- Goulot d'étranglement pour l'accès à la mémoire

# L'architecture Harvard

---

---



- Séparation des mémoires programme et données :
  - un bus de données programme,
  - un bus de données pour les données,
  - un bus d'adresse programme,
  - un bus d'adresse pour les données.
- Meilleure utilisation du CPU :
  - chargement du programme et des données en parallèle.

# Organisation d'une instruction

---

---

- Le processeur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire.
- Une instruction est composée de deux éléments :
  - le **code opération** : c'est un code binaire qui correspond à l'action à effectuer par le processeur,
  - le **champ opérande** : donnée ou bien adresse de la donnée.
- La taille d'une instruction peut varier, elle est généralement de quelques octets (1 à 8), elle dépend également de l'architecture du processeur.

# Exemple d'instruction

---

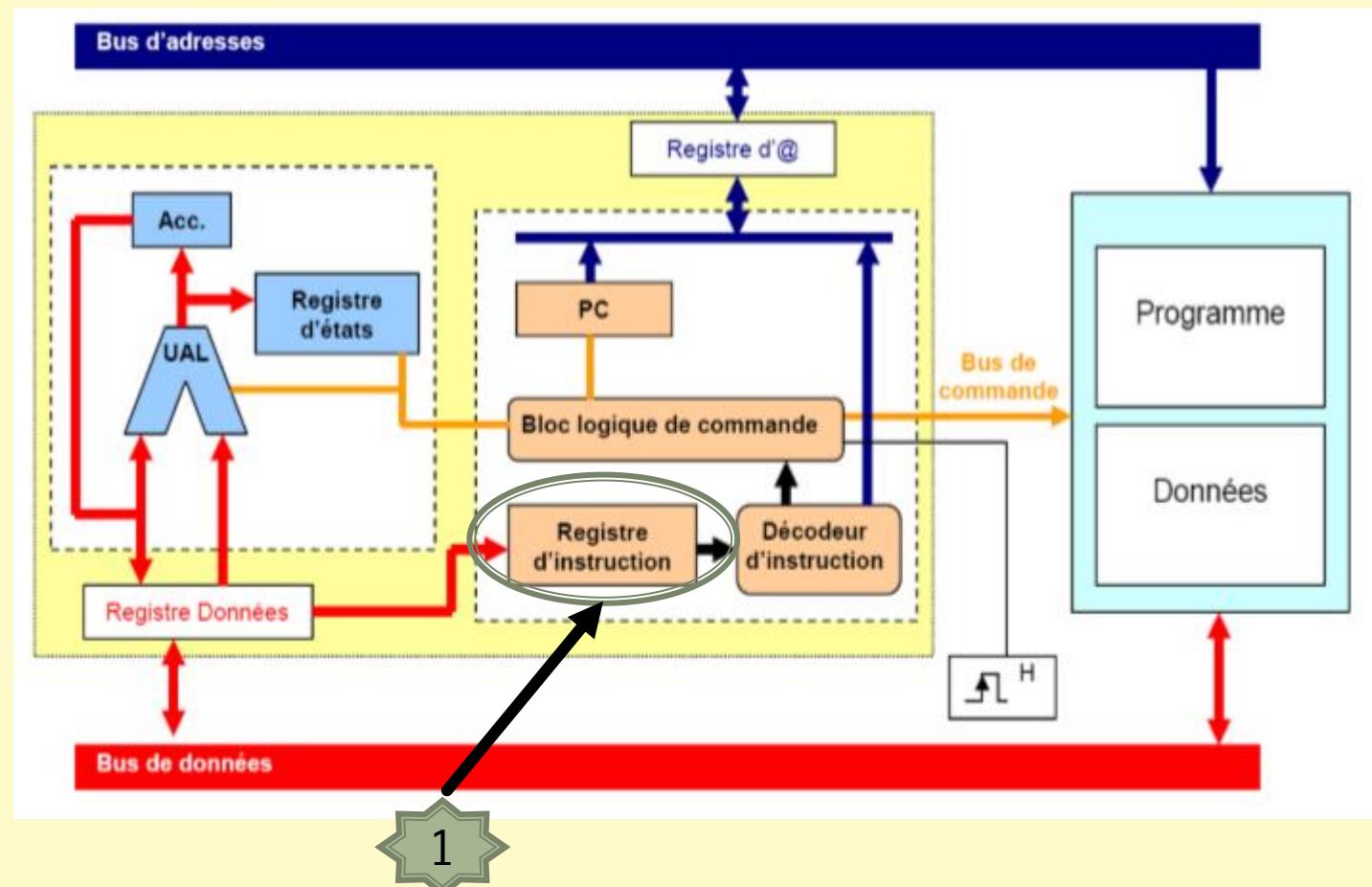
- Instruction Addition : Accumulateur = Accumulateur + Opérande
- Correspond à l'instruction **ADD A,#2**

Instruction (16 bits)	
Code opératoire (5 bits)	Champ opérande (11 bits)
ADD A	#2
11001	000 0000 0010

- Cette instruction est comprise par le processeur par le mot binaire :  
**11001000 0000 0010** = code machine

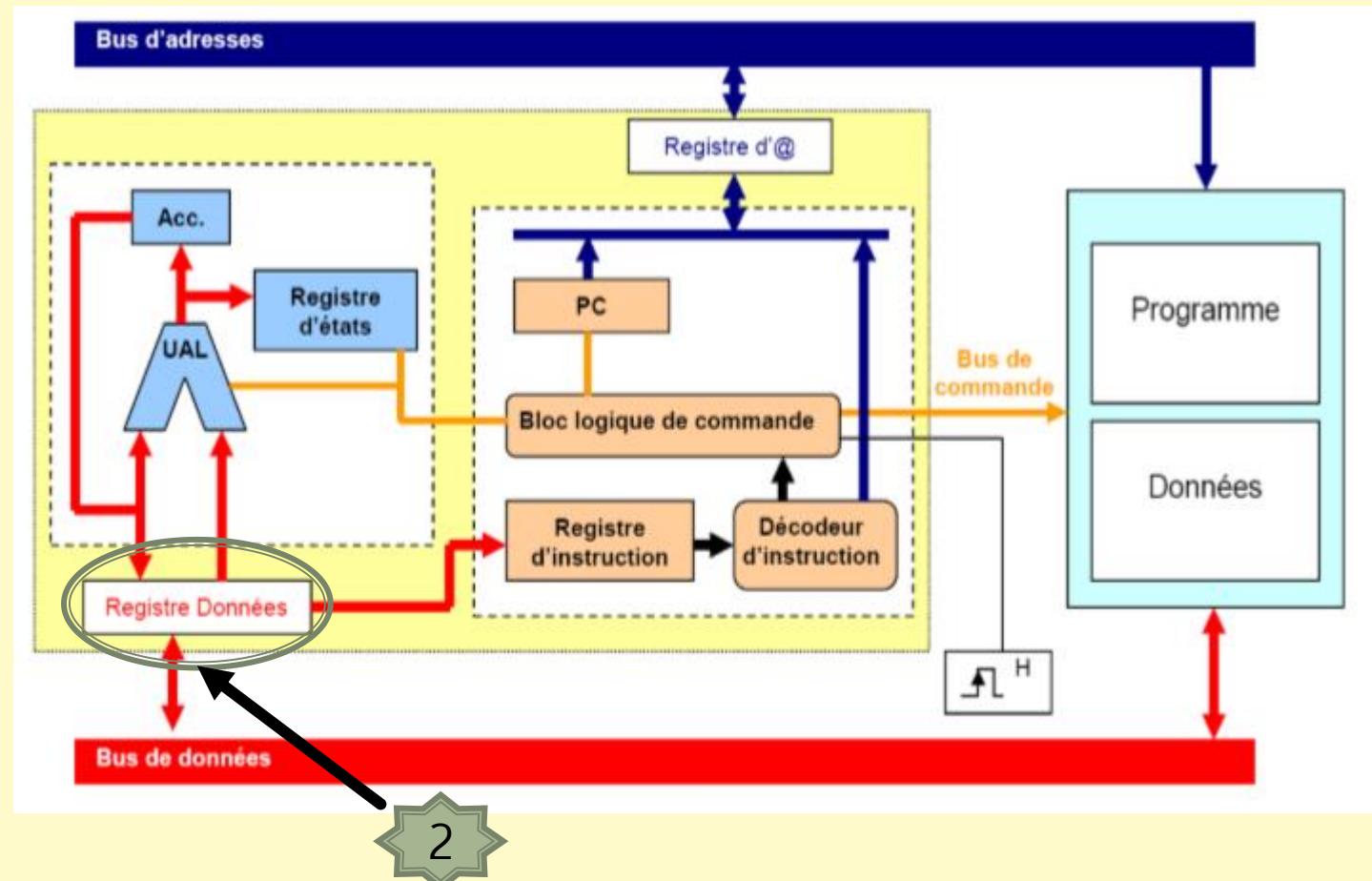
# Phase 1 : Recherche de l'instruction en mémoire

- La valeur du PC est placée sur le bus d'adresse par l'unité de commande qui émet un ordre de lecture.
- Après le temps d'accès à la mémoire, le contenu de la case mémoire sélectionnée est disponible sur le bus des données.
- L'instruction est stockée dans le registre d'instruction du processeur.



# Phase 2 : Décodage et recherche de l'opérande

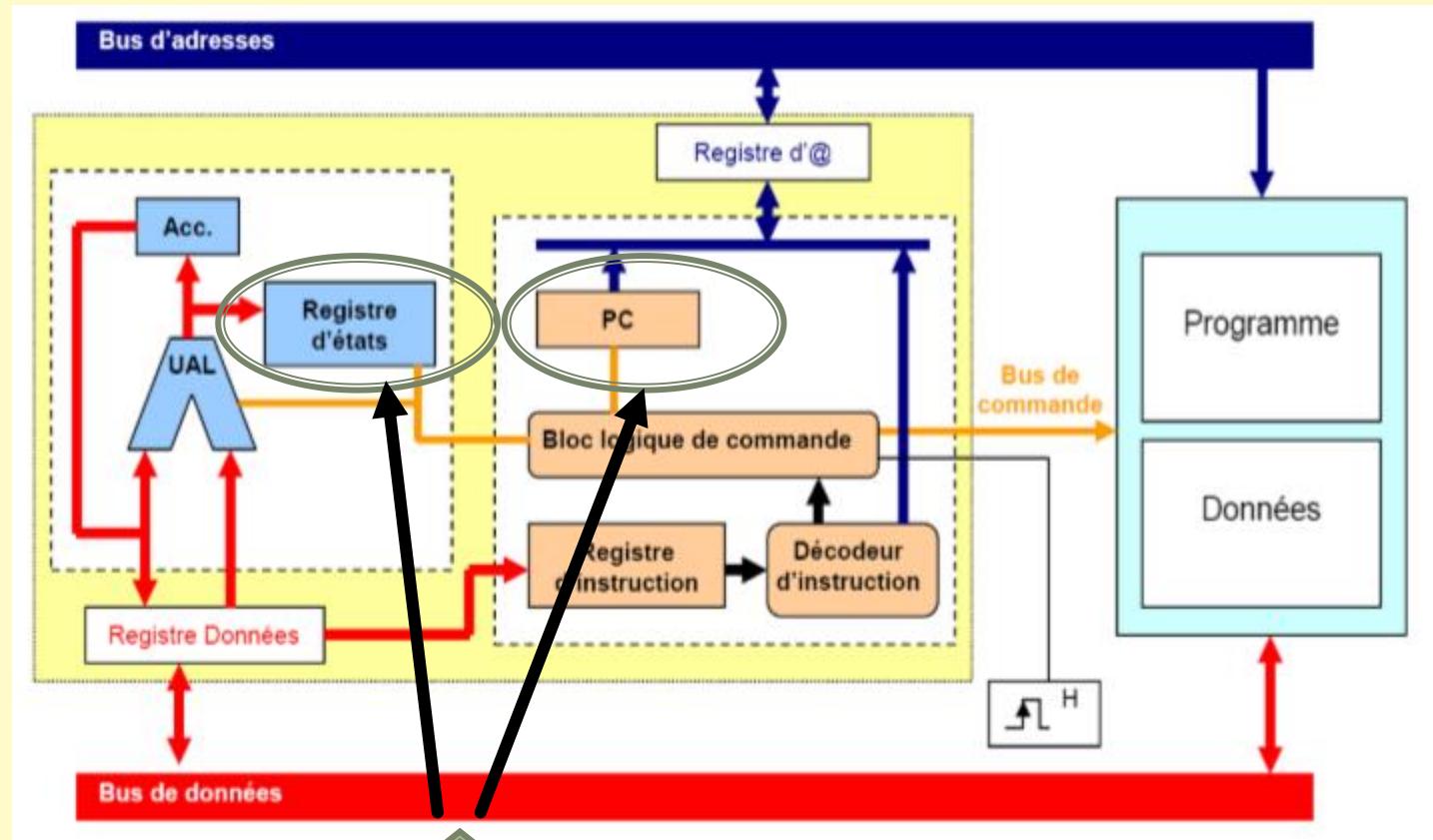
- L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.
- Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.
- L'opérande est stocké dans le registre de données.



# Phase 3 : Exécution de l'instruction

---

- Le séquenceur réalise l'instruction.
- Les drapeaux sont positionnés (registre d'état).
- L'unité de commande positionne le PC pour l'instruction suivante.



3

# Les architectures RISC et CISC

---

---

Actuellement l'architecture des microprocesseurs se composent de deux grandes familles :

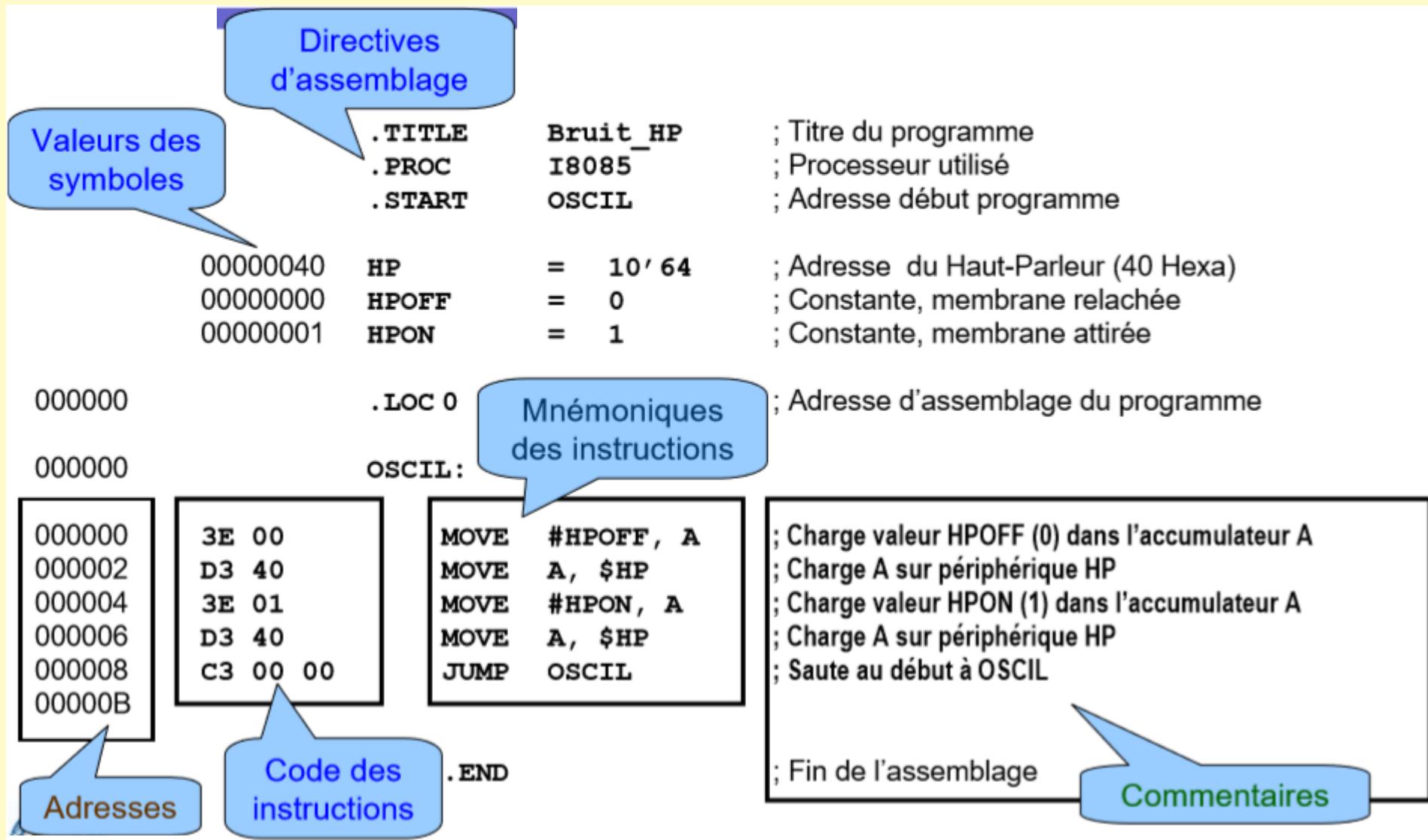
Architecture RISC	Architecture CISC
<ul style="list-style-type: none"><li>+ instructions simples ne prenant qu'un seul cycle</li><li>+ instructions au format fixe</li><li>+ décodeur simple (câblé)</li><li>+ beaucoup de registres</li><li>+ peu de modes d'adressage</li><li>+ compilateur complexe</li></ul>	<ul style="list-style-type: none"><li>+ instructions complexes prenant plusieurs cycles</li><li>+ instructions au format variable</li><li>+ décodeur complexe (microcode)</li><li>+ peu de registres</li><li>+ beaucoup de modes d'adressage</li><li>+ compilateur simple</li></ul>

Reduce  
Instruction  
Set  
Computer

Complex  
Instruction  
Set  
Computer

qui sont en train de converger ...

# Exemple d'exécution d'un programme



# PLAN DU CHAPITRE

---

---

1  
Le langage  
machine

3  
Jeu  
d'instructions

4  
Traitement des  
instructions

2  
Eléments  
d'architecture

5  
Un petit  
benchmark

# But de l'exercice

---

---

Comparer 4 architectures :

- accumulateur
- mémoire-mémoire
- pile
- chargement-rangement

sur l'instruction  $D=A+B+C$

Hypothèses :

- *opcode* sur 1 octet
- adresse mémoire sur 2 octets
- opérande sur 4 octets
- taille d'instruction multiple d'1 octet

# Accumulateur

---

---

opcode 8 bits	adresse 16 bits
------------------	--------------------

LOAD	B
ADD	C
STORE	A
ADD	C
STORE	B
SUBINV	A
STORE	D

$7 \times 24 \text{ bits} = 168 \text{ bits}$

# Mémoire-mémoire

---

---

opcode 8 bits	source 1 16 bits	source 2 16 bits	destination 16 bits
------------------	---------------------	---------------------	------------------------

ADD B, C, A

ADD A, C, B

SUB A, B, D

$3 \times 56 \text{ bits} = 168 \text{ bits}$

# Pile

---

---

opcode  
8 bits

opcode  
8 bits

adresse  
16 bits

PUSH	B	ADD
PUSH	C	POP B
ADD		PUSH A
POP	A	PUSH B
PUSH	A	SUB
PUSH	C	POP D

$$9 \times 24 \text{ bits} + 3 \times 8 \text{ bits} = 240 \text{ bits}$$

# Chargement-rangement

---

opcode 8 bits	Ri 4 bits	xx 4 bits	adresse 16 bits
------------------	--------------	--------------	--------------------

opcode 8 bits	R1 4 bits	R2 4 bits	Rd 4 bits	xx 4 bits
------------------	--------------	--------------	--------------	--------------

LOAD	Rb, B	ADD	Ra, Rc, Rb
LOAD	Rc, C	STORE	B, Rb
ADD	Rb, Rc, Ra	SUB	Ra, Rb, Rd
STORE	A, Ra	STORE	D, Rd

$$5 \times 32 \text{ bits} + 3 \times 24 \text{ bits} = 232 \text{ bits}$$

# Accès mémoire

---

---

- Accumulateur :

- 7 transferts de données ( $7 \times 32 = 224$ )

- + taille du code (168)

- = **392 bits sur 7 instructions** (56 bits/instruction)

- Mémoire-mémoire :

- 9 transferts de données ( $9 \times 32 = 288$ )

- + taille du code (168)

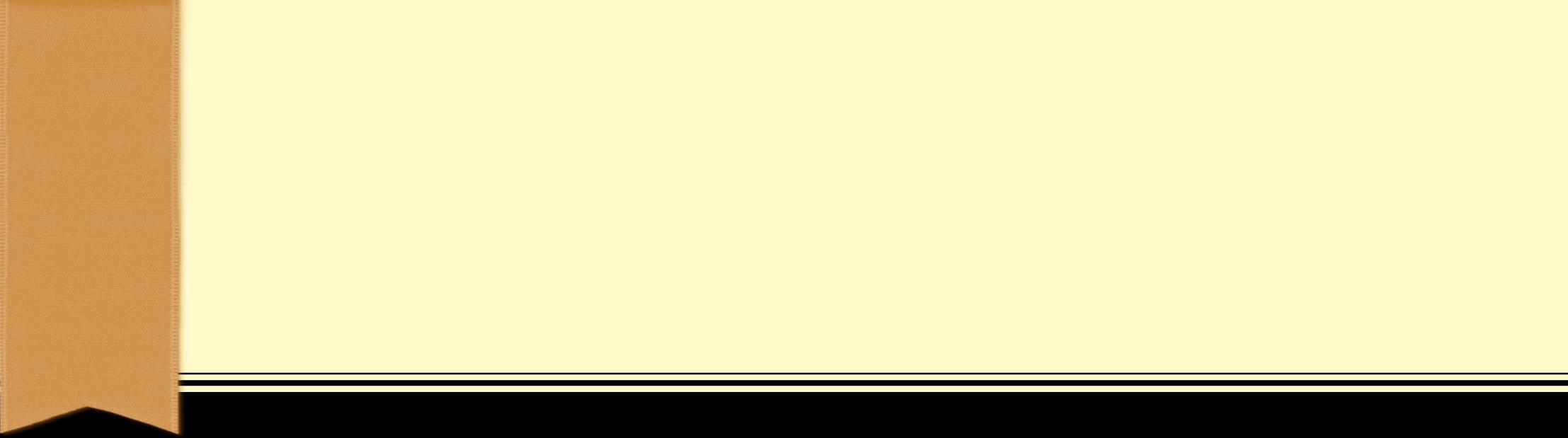
- = **456 bits sur 3 instructions** (152 bits/instructions)

# Accès mémoire

---

---

- Pile :
  - 9 transferts de données ( $9 \times 32 = 288$ )
  - + taille du code (240)
  - = **528 bits sur 12 instructions** (44 bits/instructions)
- Chargement-rangement :
  - 5 transferts de données ( $5 \times 32 = 160$ )
  - + taille du code (232)
  - = **392 bits sur 8 instructions** (49 bits/instructions)



# TEST DE CONNAISSANCE

# Petit exercice en assembleur

---

---

Considérons la suite d'instructions suivantes :

```
ADD Im R1 -10
ADD Im R1 100
JMPZ ET1
JMPN ET2
PUSH Rg1 R1
STOP
ET1: STORE D R1 100
STOP
ET2 : STORE D R1 50
STOP
```

Les nombres signés sont codés sur 8 bits en complément à 2.

1. R1 contient **20**. Où écrivez-vous R1 ? Pourquoi ?
2. R1 contient **-90**. Où écrivez-vous R1 ? Pourquoi ?
3. R1 contient **-120**. Où écrivez-vous R1 ? Pourquoi ?



# ARCHITECTURE DES ORDINATEURS

CHAPITRE 5  
LA CPU (CENTRAL PROCESS UNIT)



# PLAN DU CHAPITRE

---

---

1  
Organisation  
de la CPU

2  
Cycle de  
l'instruction

4  
Pipelines

3  
Les  
interruptions

5  
Quelques mots  
sur les accès  
DMA

# PLAN DU CHAPITRE

---

---

1

Organisation  
de la CPU

2

Cycle de  
l'instruction

4

Pipelines

3

Les  
interruptions

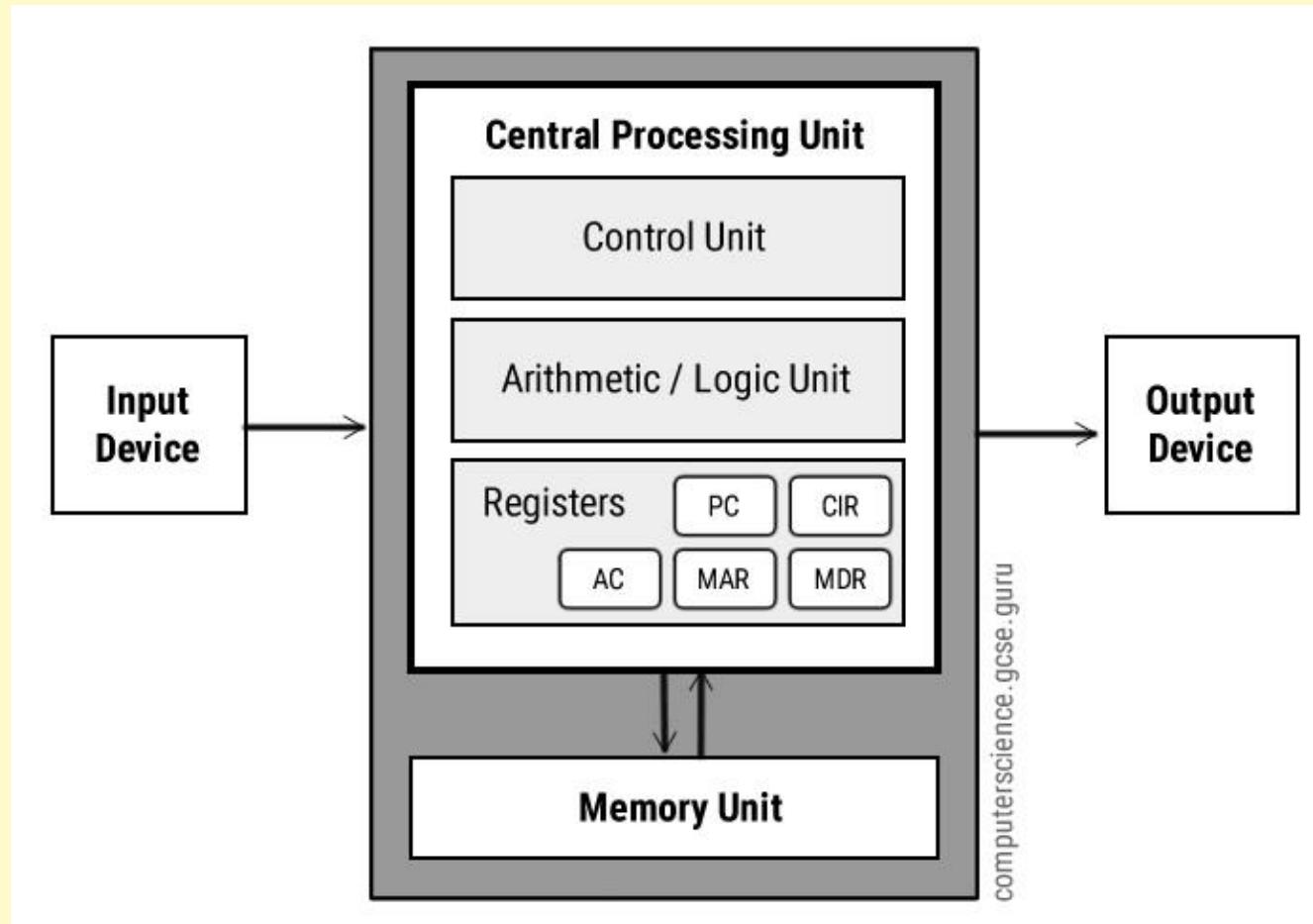
5

Quelques mots  
sur les accès  
DMA

# Organisation

---

---



# Unités de calcul

---

---

Unités réalisant des calculs : 3 types

- **Calculs logiques et arithmétiques sur les entiers : ALU (Arithmetic and Logic Unit)**
  - la plus importante, utilisée par tous les programmes,
  - calculs simples sur les entiers,
  - calculs logiques (comparaison, OR, NOT, ...) .
- **Calculs sur les flottants : FPU (Floating Point Unit)**
  - calculs sur des flottants,
  - fonctions mathématiques avancées : sqrt, sin ...
- **Calculs multimédia ou vectoriels**
  - diffère selon le type et la marque : Intel MMX et SSE, AMD 3D now ! ...
  - fait principalement des calculs vectoriels (exécution en parallèle d'une même instruction sur plusieurs données).

Un processeur peut intégrer plus d'une unité de chaque type.

# Unité de commande

---

---

- Unité qui coordonne le fonctionnement des autres éléments dans le but d'exécuter une séquence d'instructions (le programme).
- Pour exécuter une instruction, 2 cycles se succèdent :
  - cycle de recherche de l'instruction : lecture en mémoire de l'instruction à exécuter puis décodage de son contenu,
  - cycle d'exécution de l'instruction :
    1. lecture en mémoire ou des registres pour envoyer les opérandes à l'unité de calcul ou d'accès en mémoire,
    2. exécution du calcul ou de l'accès mémoire et enregistrement du potentiel résultat en mémoire ou dans un registre.

# Unité de commande

---

---

- Constituée de plusieurs éléments
  - le **Compteur Ordinal** (PC) : registre contenant l'adresse du mot mémoire stockant le code de la prochaine instruction,
  - le **Registre d'Instruction** (IR) : reçoit le code de la prochaine instruction à exécuter,
  - le **décodeur** : à partir du code de l'instruction, détermine l'opération à exécuter,
  - l'**horloge** : pour synchroniser les éléments,
  - le **séquenceur** : coordonne le tout,
- Utilise également pour l'accès en mémoire :
  - le **Registre d'Adresse** (RA) : registre contenant l'adresse du mot à accéder en mémoire,
  - le **Registre Mémoire** (RM) : registre contenant le mot lu ou à écrire en mémoire.

# L'horloge

---

---

- Horloge :
  - définit le cycle de base : cycle machine (*clock cycle*),
  - utilisée pour synchroniser chaque étape des cycles de recherche et d'exécution.
- Temps exécution d'un cycle de recherche ou d'exécution : prend un certain nombre de cycles de base.
- Cycle CPU = temps d'exécution minimal d'une instruction (recherche + exécution).

# Le séquenceur

---

---

- Automate générant les signaux de commande contrôlant les différentes unités.
- 2 façons de réaliser cet automate :
  - séquenceur câblé,
  - séquenceur micro-programmé (*firmware*).
- Avantages :
  - câblé : un peu plus rapide,
  - micro-programmé : plus souple et plus simple à réaliser.

# Les registres

---

---

- Mémoire interne à la CPU.
- Registres visibles par le programmeur :
  - génériques : pas de restriction de contenu,
  - adresses : souvent dévolus à un mode d'adressage : adresse de base de segment (e.g., pentium), index, pointeur de pile en mémoire,
  - conditions (flags) : suite de bits indépendants, positionnés en fonction du résultat d'une opération, en lecture seule.
- Registres de contrôle et de statuts :
  - utilisés par la CPU,
  - utilisés par le système.

# Choix de conception

---

---

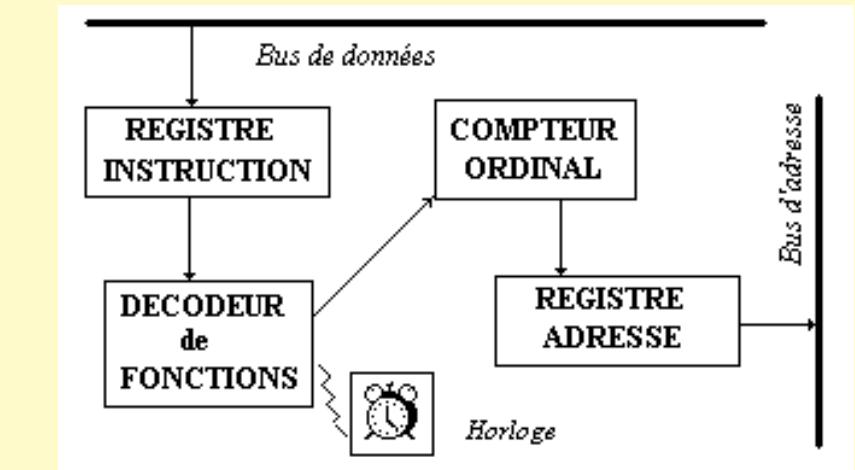
- Quelle proportion de registres spécialisés adopter ?
- Quel nombre de registres adopter ?
- Quelle taille de registres adopter ?



# Registre de contrôle et statuts

d

- Echange avec la mémoire principale.
- Compteur Ordinal (PC) :
  - adresse de la prochaine instruction à exécuter.
- Registre d'Instruction (IR) :
  - instruction en cours d'exécution.
- Registre d'adresse mémoire (MAR) :
  - contient une adresse mémoire
  - directement connecté au bus d'adresse
- Registre tampon mémoire (MBR) :
  - contient un mot de données
  - directement connecté au bus de données



Registres intermédiaires

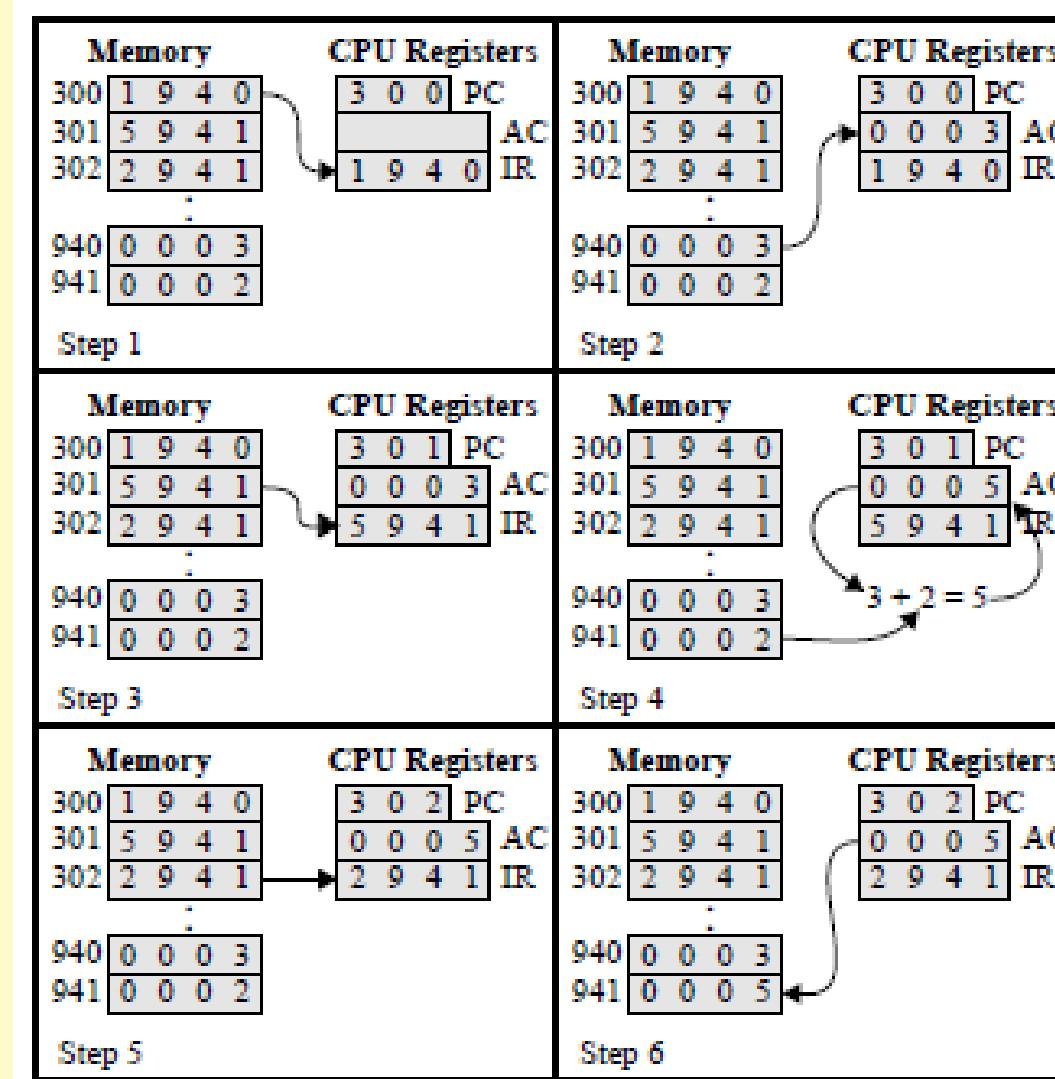
# Registre PSW (Program Status Word)

Inclut les booléens suivants :

- The diagram shows the PSW (Processor Status Word) register. It consists of three adjacent boxes. The first box contains the letter 'CY'. The second box is yellow and contains the letters 'MSB'. The third box contains the letter 'A'.
- bit de signe du résultat
  - le résultat est 0
  - l'opération a généré une retenue
  - le résultat d'une comparaison est une égalité
  - une opération a provoqué un débordement
  - le fonctionnement normal peut être interrompu
  - mode privilégié
  - ...

	0	0	0	0	0	0	0	(Value on RESET)
	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
PSW	CY	AC	F0	RS1	RS0	OV	--	P
MSB				LSB				

# Exemple d'exécution d'un programme



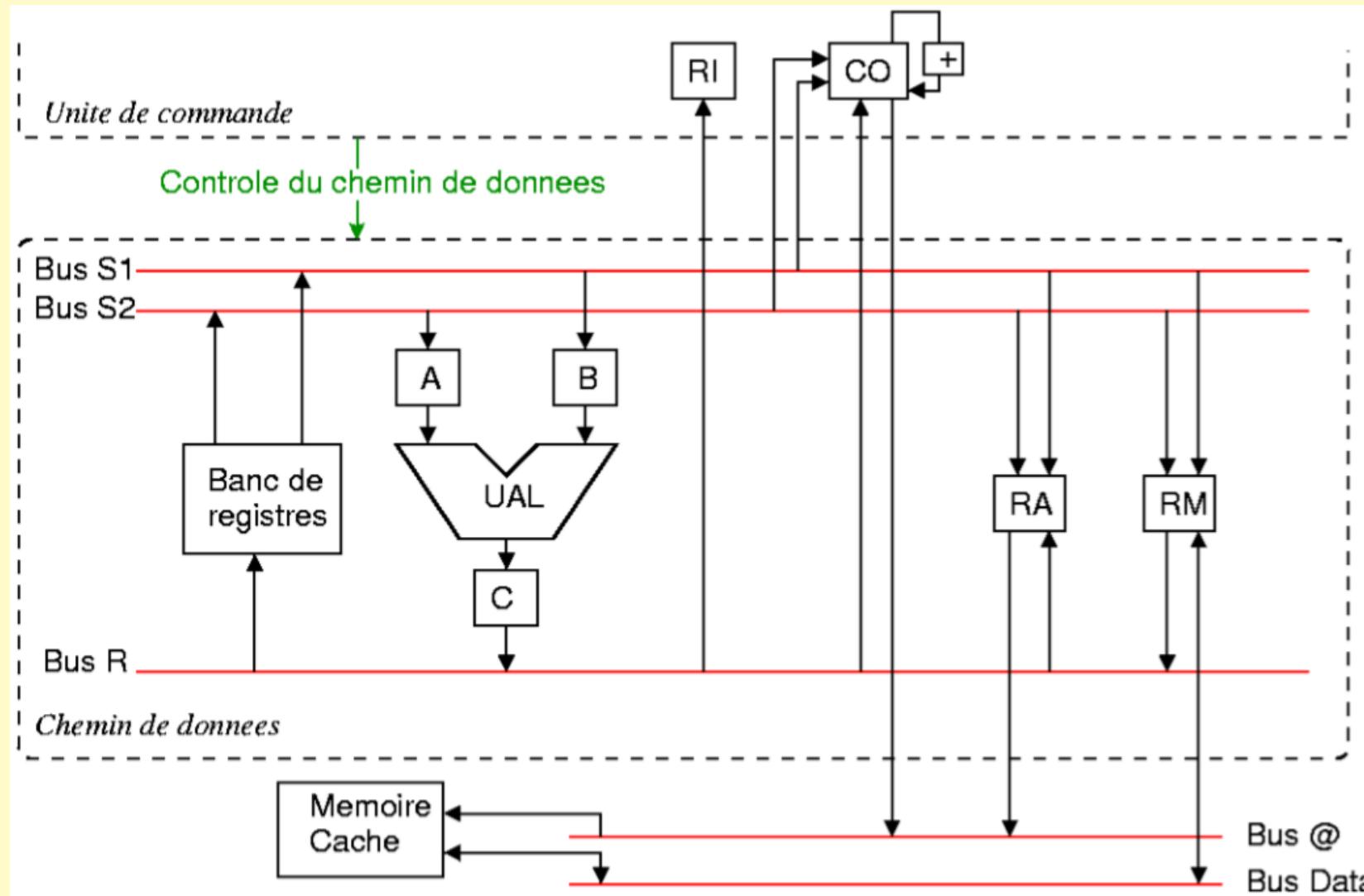
# Des bus internes à la CPU

---

---

- Utilisation de bus au cœur de la CPU :
  - pour envoi de données entre les éléments : contrôleur mémoire, unités de calculs, registres ...
  - pour envoi d'adresses : lecture/écriture en mémoire ou dans un registre ...
  - pour commander, coordonner les éléments.
- Chemin de données :
  - unités gérant le traitement des données :
    - unités de calculs,
    - registres de tout type.

# Des bus internes à la CPU



- **S1** et **S2** : entrées de l'UAL
- **R** : résultat, sortie de l'UAL
- **@** : bus d'adresse pour accès en mémoire centrale (via le cache)
- **Data** : bus de données à écrire ou lues dans mémoire centrale
- Manque le bus de commande entre tous les éléments

# Architecture X bits

---

---

- Les processeurs sont souvent différenciés selon leur architecture 16, 32 ou 64 bits.
- Historiquement : taille des registres (8, 16 bits...).
- Mais dans processeurs récents : registres de toute taille (16, 32, 64, 80 ou 128 bits) selon que l'on manipule des entiers, des adresses, des flottants, des vecteurs ...
- Norme de fait de nos jours : taille des registres généraux.
  - Un processeur 64 bits a des registres généraux de 64 bits.
- Conséquences :
  - les unités de calcul entier doivent gérer des nombres de même taille que les registres généraux,
  - les bus internes doivent avoir aussi cette même taille.

# PLAN DU CHAPITRE

---

---

1  
organisation  
de la CPU

2  
Cycle de  
l'instruction

4  
Pipelines

3  
Les  
interruptions

5  
Quelques mots  
sur les accès  
DMA

# Cycle de l'instruction

---

---

- Recherche de l'instruction (*fetch*) :
  - lecture de l'instruction depuis la mémoire
- Interprétation de l'instruction (*decode*) :
  - détermination de l'opération correspondant à l'instruction
  - recherche des données (*fetch data*) : lecture de données dans la mémoire ou depuis un module d'E/S
- Exécution (*execute*) :
  - traitement des données (*process data*) : opérations arithmétiques ou logiques
  - écriture des données (*write data*) : écriture du résultat dans la mémoire ou vers un module d'E/S
- Interruption (*interrupt*) : vérifie si une interruption est apparue

# Flot de données

---

---

## Fetch :

- $\text{MAR} \leftarrow \text{PC}$
- L'unité de contrôle demande une lecture de la mémoire principale.
- Le résultat de la lecture est placé dans MBR.
- $\text{IR} \leftarrow \text{MBR}$
- $\text{PC} \leftarrow \text{PC} + 1$

# Flot de données

---

---

Exemple de cycle indirect pour **decode** :

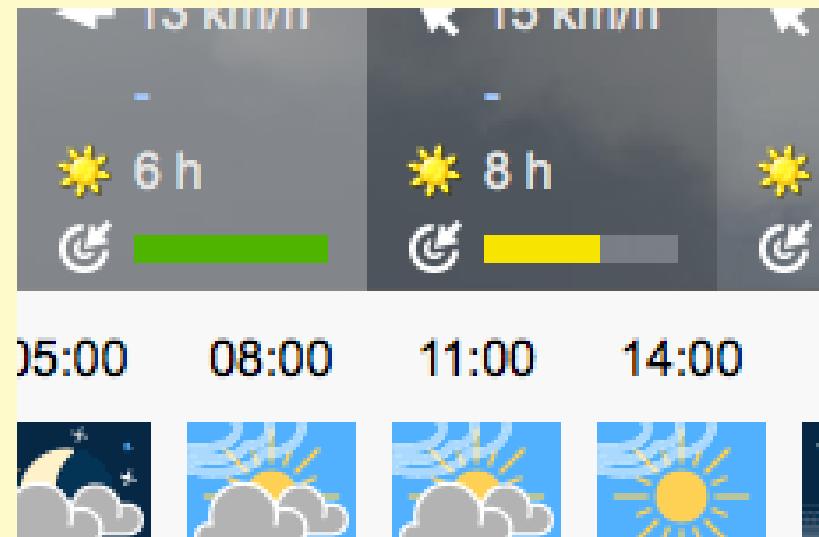
- $\text{MAR} \leftarrow$  les N bits de poids faibles de MBR
- L'unité de contrôle demande une lecture de la mémoire principale.
- Le résultat de la lecture est placé dans MBR.
- $\text{MAR} \leftarrow$  MBR
- L'unité de contrôle demande une lecture de la mémoire principale.
- Le résultat de la lecture est placé dans MBR.

# Prévisibilité

---

---

- Les cycles **fetch** et **decode** sont simples et prévisibles.
- Le cycle **execute** est imprévisible.



# Modes d'adressage

---

---

- Permettent la manipulation des différents structures de données.
- Permettent l'accès des données en mémoire :
  - des variables par leur valeur,
  - accès aux données via des pointeurs,
  - des éléments de tableaux,
  - accès aux éléments de structures.
- Ces données sont les opérandes pour des instructions :
  - arithmétiques ...
  - déplacement ...

# Modes d'adressage

---

---

- Les opérandes peuvent être dans :
  - des registres,
  - une zone mémoire: on utilise alors l'adresse effective qui peut être une combinaison de 3 éléments :
    - la base ,
    - l'index,
    - le déplacement .
- Cela donne des adressages :
  - **immédiat** : le champs d'adresse contient la valeur de l'opérande ou son adresse physique,
  - **direct** : l'adresse est l'adresse effective ou l'offset de la variable par rapport au segment de base.,
  - **indirect** (par registre) : l'offset de la variable est contenu dans un registre de base ou d'index,
  - **base** : similaire à l'adressage indirect par registre sauf qu'un déplacement est ajouté à la base,
  - **indexé** : on utilise un registre d'index plus un déplacement (l'adressage est noté par des crochets ).

# PLAN DU CHAPITRE

---

---

1  
organisation  
de la CPU

2  
cycle de  
l'instruction

4  
Pipelines

3  
Les  
interruptions

5  
Quelques mots  
sur les accès  
DMA

# Problématique & définition

---

- Un système informatique n'est utile que s'il communique avec l'extérieur :
  - l'objectif est de pouvoir prendre connaissance que le périphérique sollicite le processeur,
  - cette sollicitation arrive de façon totalement asynchrone.
- Deux modes sont possibles:
  - une méthode par **scrutation** (*polling*) permet d'interroger régulièrement les périphériques afin de savoir si une nouvelle donnée est présente,
  - une méthode par **interruption** permet au périphérique lui-même de faire signe au processeur de sa présence.

# Scrutation *vs* interruption

---

---

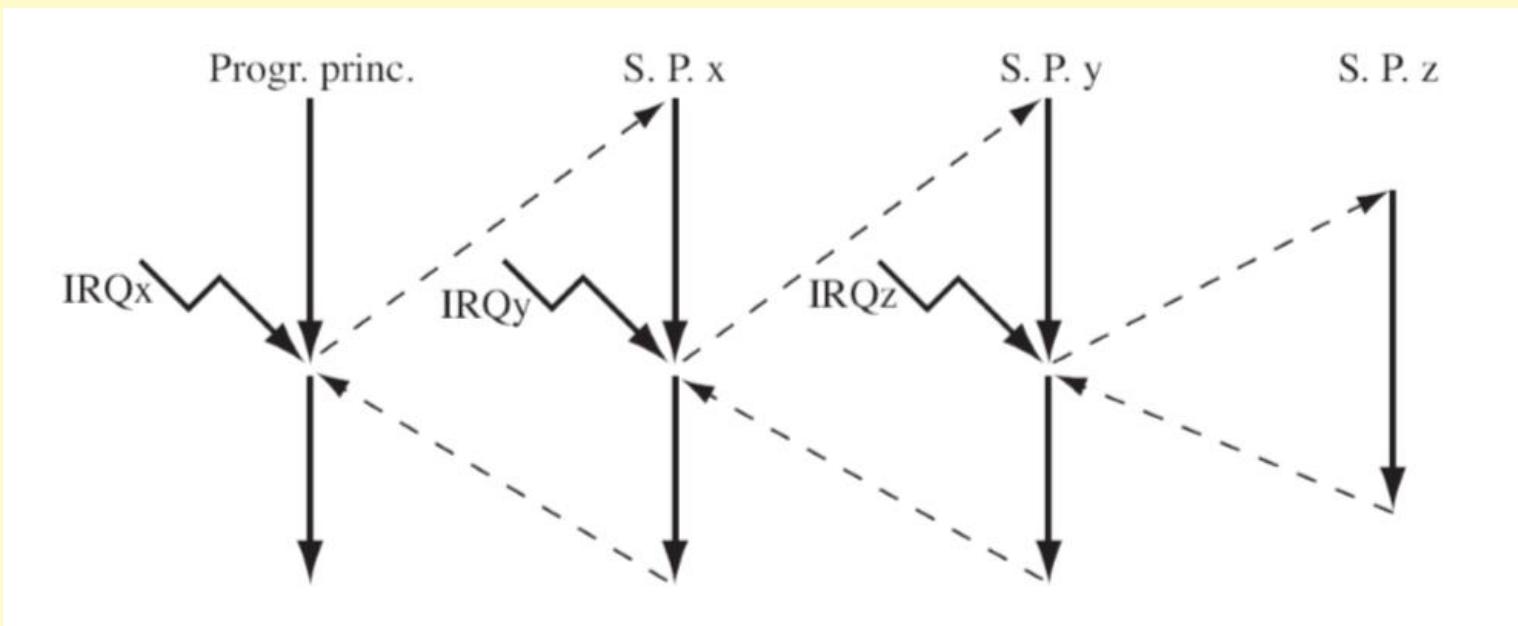
- Scrutation (polling) :
  - coûteux en temps (multiplier par le nombre de périphérique à interroger),
  - implémentation : appel classique à une fonction dans le programme.
- Interruption :
  - demande à l'initiative du périphérique,
  - prise en compte rapide de l'évènement,
  - implémentation : interruption asynchrone d'un programme puis retour au même endroit à la fin du traitement.

# Schéma d'une interruption

---

---

Une interruption est un arrêt temporaire de l'exécution normale d'un programme informatique par le microprocesseur afin d'exécuter un autre programme (appelé routine d'interruption).



# Types d'interruption

---

---

- Interruption **masquable** :

- un masque d'interruption est un mot binaire de configuration du microprocesseur qui permet de choisir (démasquer)quels modules pourront interrompre le processeur parmi les interruptions disponibles.

- Interruption **non masquable** :

- elles s'exécutent quoi qu'il arrive, souvent avec une priorité élevé (exemple : Reset).

# Fonctionnement d'une interruption

---

---

En cas d'interruption :

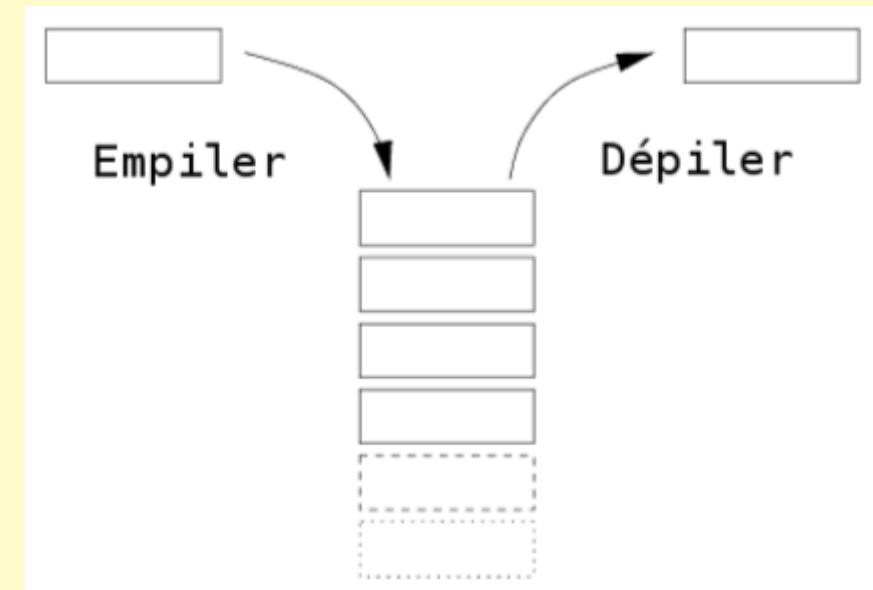
- sauvegarde de l'état courant :
  - PC
  - PSW
  - - ...
- exécution d'une suite d'instructions traitant l'interruption (routine d'interruption),
- retour à l'état sauvégardé.

# Rôle de la pile dans une interruption

---

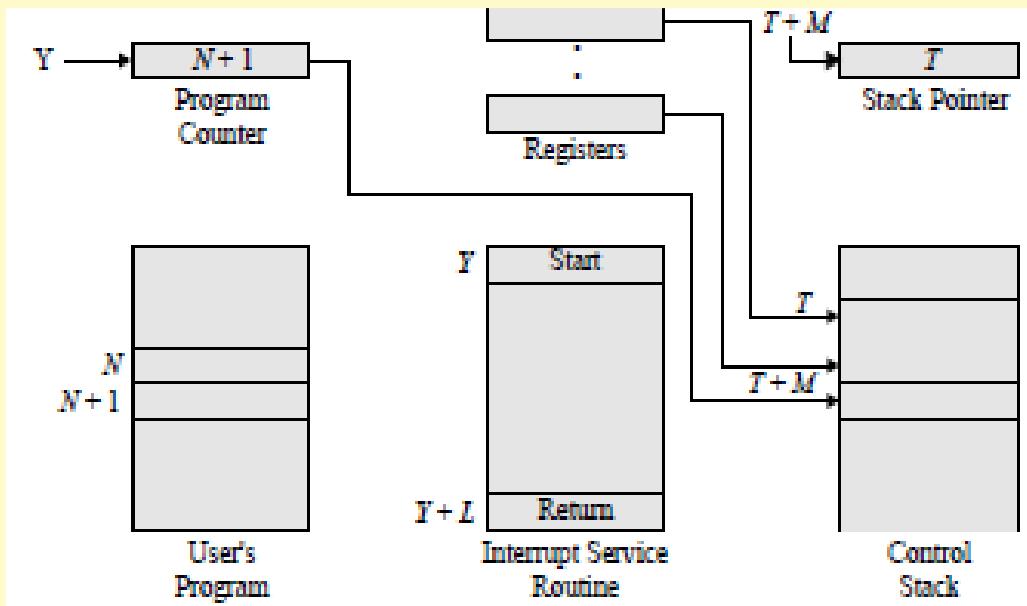
---

- La pile est une mémoire LIFO (*Last In First Out*) dans laquelle on stocke des variable temporaire (donnée ou adresse).
- Le haut de la pile est pointé par le registre SP (*Stack Pointer*).
- Elle va servir à :
  - sauvegarder le contexte d'environnement (adresse du programme et valeur des registres au moment de l'interruption),
  - restituer le contexte à la fin de l'interruption.

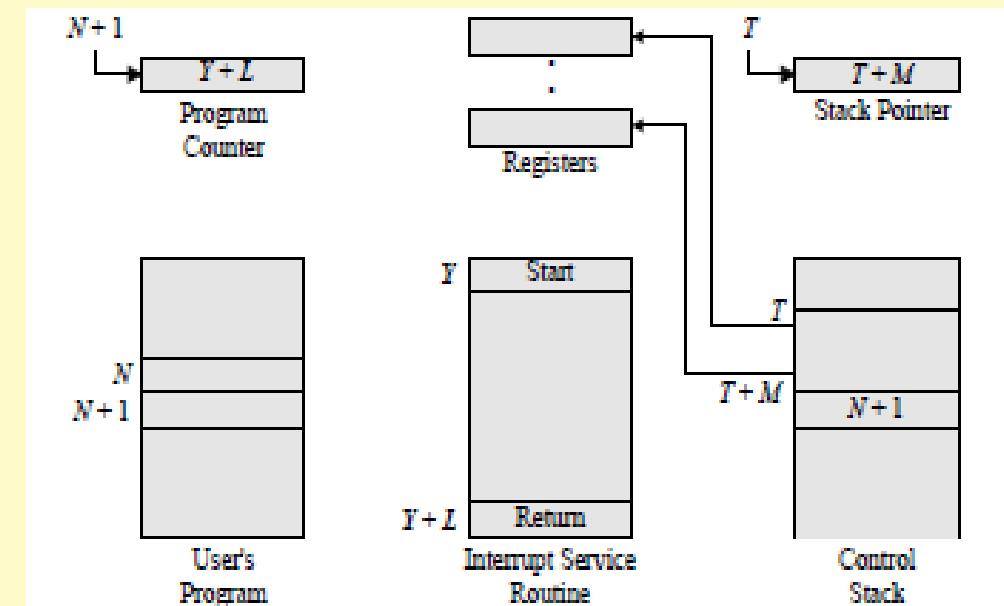


# Changements dans la mémoire et les registres lors d'une interruption

---



L'interruption se produit après l'instruction  
à l'adresse  $N$



Retour de l'interruption

# Cycle d'interruption

---

---

Simple et prévisible :

- MBR  $\leftarrow$  PC
- MAR  $\leftarrow$  une adresse mémoire où sauvegarder PC
- L'unité de contrôle demande une écriture dans la mémoire principale.
- PC  $\leftarrow$  adresse de la routine d'interruption

Le nouveau cycle commence par le **fetch** de la première instruction de la routine d'interruption.

# PLAN DU CHAPITRE

---

---

1  
organisation  
de la CPU

2  
cycle de  
l'instruction

4  
Pipelines

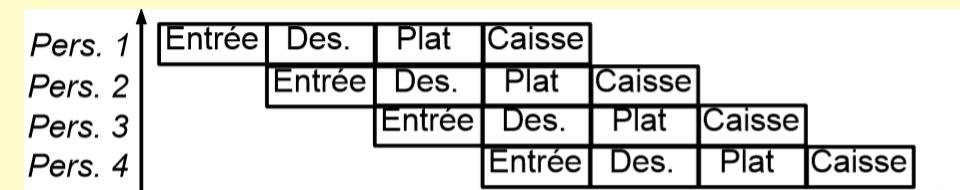
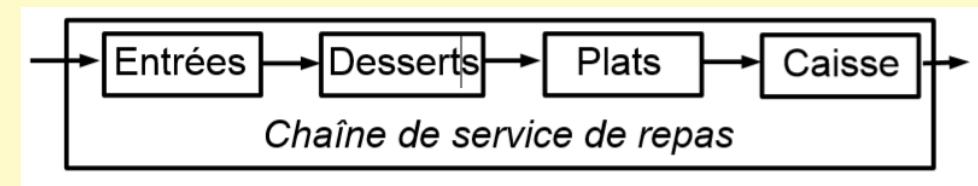
3  
Les  
interruptions

5  
Quelques mots  
sur les accès  
DMA

# Le principe

---

- Restaurant Universitaire := on passe, dans l'ordre devant 4 éléments :
  - un présentoir pour les entrées,
  - un présentoir pour les desserts,
  - un présentoir pour les plats de résistance,
  - une caisse.
- 2 modes d'utilisation pour se servir un repas :
  - une seule personne à la fois dans toute la chaîne de service : quand elle a passé toute la chaîne et est sortie, une autre personne entre se servir,
  - plusieurs personnes à la fois, en décalé :
    - une personne à chaque présentoir/élément,
    - une personne passe à l'élément suivant quand il est libre et qu'elle en a fini avec son élément courant.



# Le principe

---

---

- Intérêts du deuxième mode :

- plusieurs personnes se servent en même temps,
- gain de temps : plus de personnes passent pendant une même durée,
- meilleure gestion des éléments : toujours utilisés.

- Inconvénients du deuxième mode :

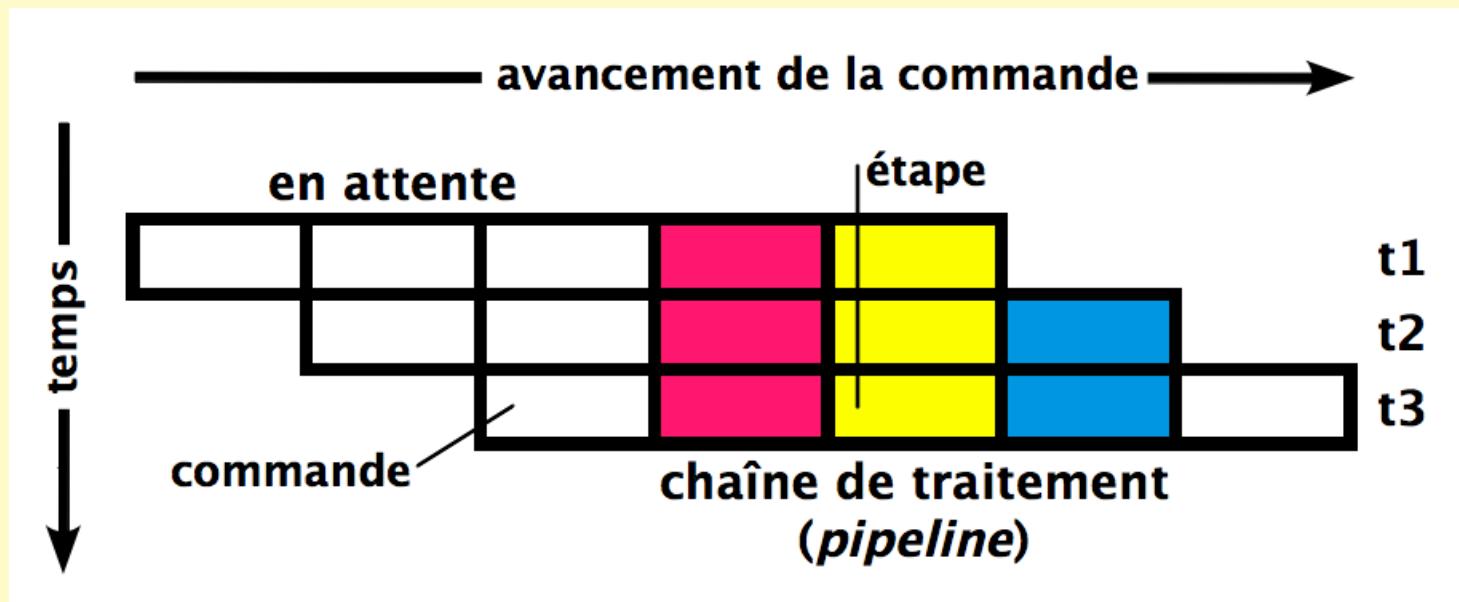
- plus difficile de faire « demi-tour » dans la chaîne d'éléments,
- nécessite des synchronisations supplémentaires et des éléments dont le parcours prend un temps proche pour une bonne optimisation.

# Dans un processeur ...

---

---

- Objectif : diminuer le temps d'exécution d'une instruction en faisant travailler à la chaîne les différentes unités fonctionnelles.



# Pipeline à trois étages

---

---

- 3 phases indépendantes (*fetch*, *decode*, *execute*) réalisées chacune en 1 cycle d'horloge.
- Travail en parallèle des 3 unités responsables des 3 phases au cycle  $i$ :
  - on recherche l'instruction  $i$ ,
  - on décode l'instruction  $i-1$ ,
  - on exécute l'instruction  $i-2$ .

# Pipeline à trois étages

---

---

cycle	1	2	3	4	5	6
fetch	inst 1	inst 2	inst 3	inst 4	inst 5	inst 6
decode		inst 1	inst 2	inst 3	inst 4	inst 5
execute			inst 1	inst 2	inst 3	inst 4

- Temps moyen d'exécution d'une instruction divisé par 3
- Temps d'exécution de n instructions est de  $2 + n$  cycles d'horloge.

# Pipeline à cinq étages

---

---

cycle	1	2	3	4	5	6
LI	inst 1	inst 2	inst 3	inst 4	inst 5	inst 6
DI		inst 1	inst 2	inst 3	inst 4	inst 5
EX			inst 1	inst 2	inst 3	inst 4
MEM				inst 1	inst 2	inst 3
ER					inst 1	inst 2

- Nécessité de :
  - rendre les 5 phases indépendantes,
  - utiliser des registres servant d'interface entre les étages.

# Performance

---

---

Exécuter  $n$  instruction dans un pipeline à  $k$  étages :

- nombre de cycle :

$$T_k = k + (n - 1)$$

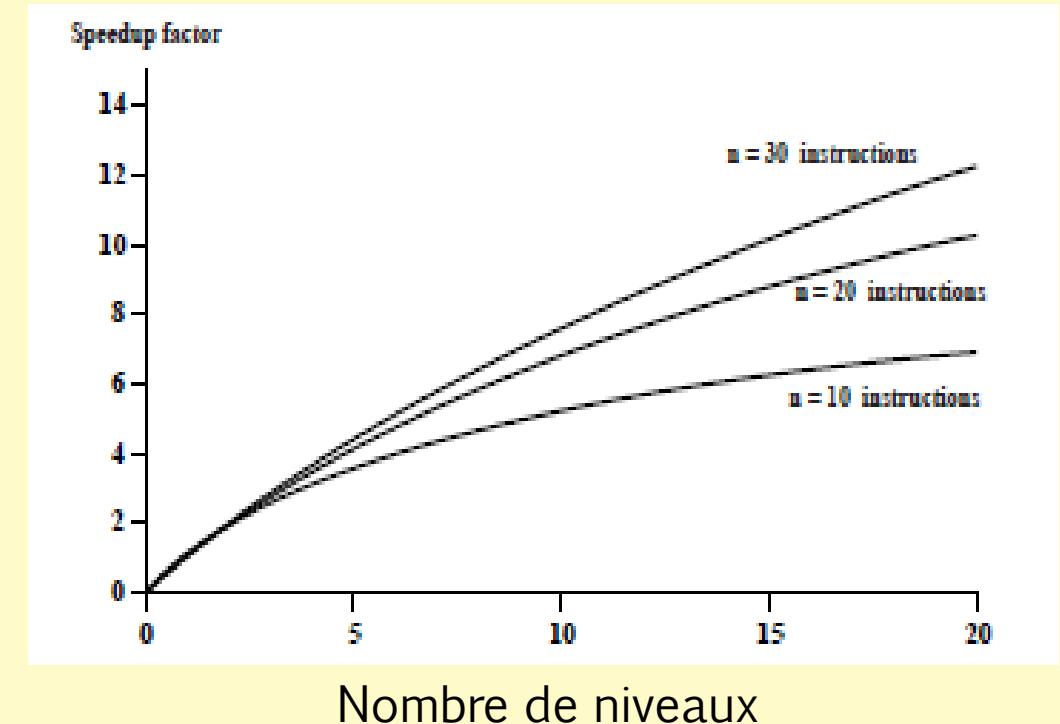
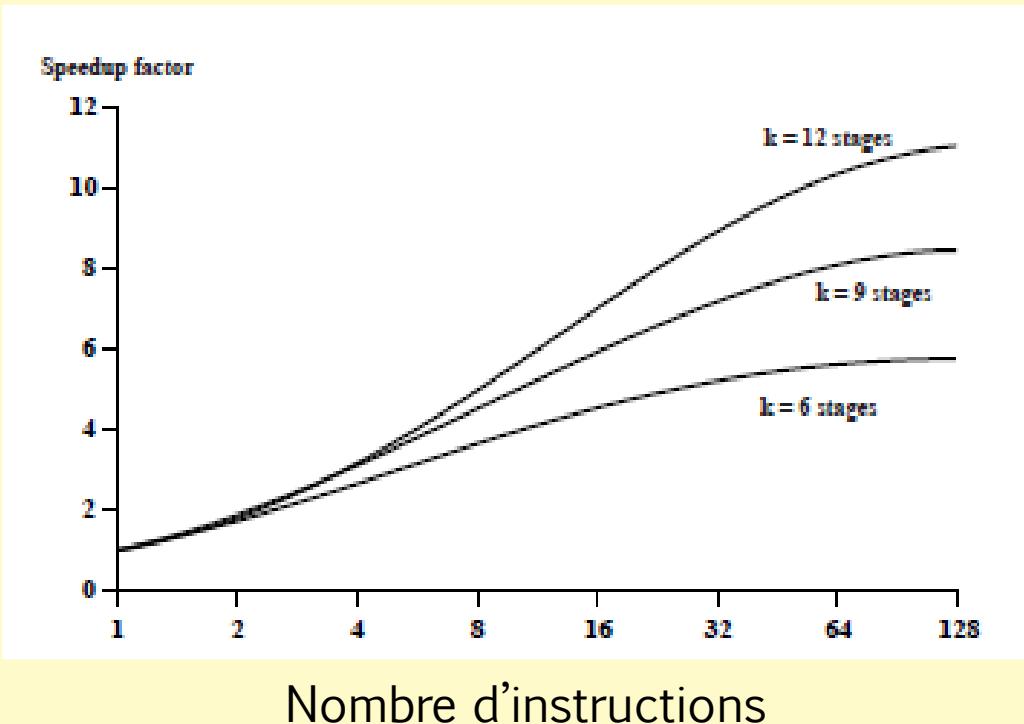
- accélération :

$$\frac{T_1}{T_k} = \frac{nk}{k+(n-1)}$$

# Facteurs d'accélération avec l'usage du pipeline

---

---



# Aléas

---

---

Situation qui empêche l'instruction suivante de s'exécuter au cycle d'horloge prévu :

- aléa structurel : conflit de ressources ne pouvant pas être géré par le matériel
  - exemple : mémoire unique pour instructions et données
  - solution : suspendre le pipeline durant un cycle
- aléa de données : dépendance de données entre les instructions,
- aléa de contrôle : modification du Compteur Ordinal
  - exemple : une instruction de branchement conditionnel,
  - jusqu'à l'exécution d'une telle instruction, il est impossible de savoir si un branchement est effectué ou non,
  - principal facteur de dégradation de performance dans une architecture pipeline.

# PLAN DU CHAPITRE

---

---

1  
organisation  
de la CPU

2  
cycle de  
l'instruction

4  
Pipelines

3  
Les  
interruptions

5  
Quelques mots  
sur les accès  
DMA

# Pourquoi les accès DMA ?

---

---

- Le système doit récupérer des données en provenance de ces périphériques externes.
- Plusieurs méthodes sont possibles :
  - une méthode par scrutation (polling) permet d'interroger régulièrement les périphériques afin de savoir si une nouvelle donnée est présente.
  - une méthode par interruption permet au périphérique lui-même de faire signe au processeur de sa présence.
  - une méthode par Accès Direct à la Mémoire (DMA) permet de gérer le transfert de façon autonome.

# Définition

---

---

- L'accès direct à la mémoire ou DMA est un procédé où des données circulant de ou vers un périphérique (port de communication, disque dur) sont transférées directement par un contrôleur adapté vers la mémoire centrale de la machine, sans intervention du processeur.
- Le processeur interviendra seulement pour initier et conclure le transfert.
- La conclusion du transfert ou la disponibilité du périphérique peuvent être signalés par interruption.

(Source : Wikipédia)

# Les interfaces des disques

---

---

- Le standard **ATA** (*Advanced Technology Attachment*) est une interface permettant la connexion de périphérique de stockage sur les ordinateurs de type PC.
- Ce standard apparu en 1994 tend à disparaître au profit du **SATA**. Il est aussi connu sous le nom **IDE** (*Integrated Drive Electronics*) ou **E-IDE** (*Enhanced IDE*).
- Initialement pour connecter les disques dur, il a été étendu pour pouvoir interfacer d'autre périphériques de stockage (Interface **ATAPI** = *ATA-Packet Interface*).



# Les interfaces SATA

---

---

Les interfaces SATA (Serial ATA), permettent de transférer les données en série :

- gain de place,
- branchement à chaud,
- résolution de problème de CEM (compatibilité électromagnétique).



# Les modes de transfert

---

---

- Mode PIO: Programme d'Input/Output:
  - permet d'échanger des données avec la mémoire vive,
  - ces transferts sont gérés entièrement par le processeur.
- Mode DMA:
  - la technique du DMA (Direct Memory Access) permet de désengorger le processeur en permettant à chacun des périphériques d'accéder directement à la mémoire.

# Mode PIO

---

---

- Des commandes gérées directement par le processeur permettent la gestion du transfert.
- Toutefois, de gros transferts de données peuvent rapidement imposer une grosse charge de travail au processeur et ralentir l'ensemble du système.
- Il existe 5 modes PIO définissant le taux de transfert maximal.

Mode PIO	Débit (Mo/s)
Mode 0	3.3
Mode 1	5.2
Mode 2	8.3
Mode 3	11.1
Mode 4	16.7

# Mode DMA

---

---

- La technique du DMA (Direct Memory Access) permet de désengorger le processeur en permettant à chacun des périphériques d'accéder directement à la mémoire.
- Deux types de DMA existent:
  - le DMA dit "single word" permet de transmettre un mot simple à chaque session de transfert,
  - le DMA dit "multi-word" permet de transmettre successivement plusieurs mots à chaque session de transfert.
- Le tableau ci-contre liste les différents modes DMA et les taux de transfert associés:

Mode DMA	Débit (Mo/s)
0 (Single word)	2.1
1 (Single word)	4.2
2 (Single word)	8.3
0 (Multiword)	4.2
1 (Multiword)	13.3
2 (Multiword)	16.7

# Mode Ultra DMA

---

---

- L'idée est d'augmenter la fréquence du signal d'horloge pour augmenter la rapidité.
- Toutefois sur une interface où les données sont envoyées en parallèle, l'augmentation de la fréquence pose des problèmes d'interférence électromagnétiques.
- Deux solutions ont été apportées qui vont être en étroite relation :
  - augmentation de la fréquence : utilisation des front montants et descendant,
  - amélioration du connecteur ATA ( à partir de l'Ultra DMA mode 4 un nouveau type de nappe a été introduit afin de limiter les interférences ; il s'agit d'une nappe ajoutant 40 fils de masse entrelacés avec les fils de données,
  - apparition du CRC.

# Mode Ultra DMA

---

---

## ■ Fonctionnement :

- la fréquence de transfert augmente tant que les données transmises se font sans erreur,
- lorsque qu'une erreur est rencontrée, le transfert passe dans un mode Ultra DMA inférieur (voire sans Ultra DMA).

Mode Ultra DMA	Débit (Mo/s)
UDMA 0	16.7
UDMA 1	25.0
UDMA 2 (Ultra-ATA/33)	33.3
UDMA 3	44.4
UDMA 4 (Ultra-ATA/66)	66.7
UDMA 5 (Ultra-ATA/100)	100
UDMA 6 (Ultra-ATA/133)	133

# Les vitesses de transfert

---

---

Architecture	Name	Time	Transfer Speed	Note
Serial	Serial ATA <sup>2</sup>	Mid-2007	600 MB/S (generation 3)	CRC / Package transfer (bits of data together)
		Mid-2004	300 MB/S (generation 2)	
		Fall-2002	150 MB/S (generation 1)	
Parallel	Ultra DMA	Current mainstream	16.7/25.0/33.3/44.4/66.7	CRC / Multi-word
			100.0/133.0 MB/S	
	DMA	1990	4.2/13.3/16.7MB/S	Multi-word
		1980	2.1/4.2/8.3 MB/S	Single word

Les vitesses de transfert (mode DMA ou Ultra DMA restent donc toujours en étroite relation avec l'architecture utilisée (ATA, Serial ATA, ...).



# ARCHITECTURE DES ORDINATEURS

## CHAPITRE 6 LES INTERCONNEXIONS



# PLAN DU CHAPITRE

---

---

1  
bus : définition,  
structure et  
caractéristiques

3  
Techniques  
d'arbitrage

4  
Exemple :  
le bus PCI

2  
Synchronisation  
des échanges

5  
Exercice

# PLAN DU CHAPITRE

---

---

1

bus : définition,  
structure et  
caractéristiques

3

Techniques  
d'arbitrage

4

Exemple :  
le bus PCI

2

Synchronisation  
des échanges

5

Exercice

# Rappels

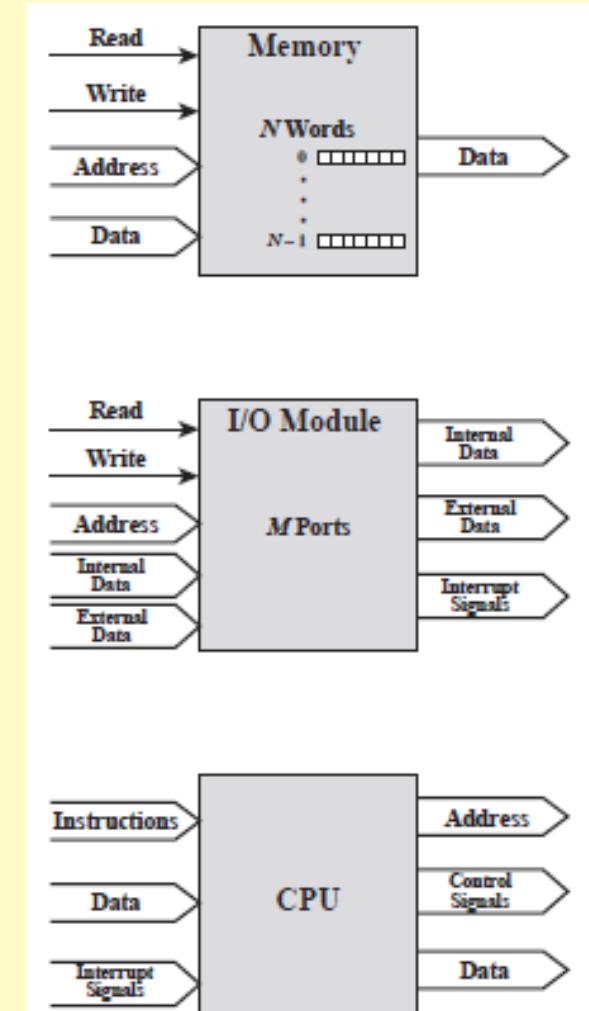
---

3 unités fonctionnelles :

- la mémoire : lecture, écriture d'un mot, à une adresse,
- l'unité d'entrées-sorties,
- le processeur : contrôle le système.

Elles communiquent les unes avec les autres.

*Structure d'interconnexion : l'ensemble des chemins les connectant.*



# Type de transfert

---

mémoire vers processeur	lecture d'instructions
processeur vers mémoire	écriture de données
E/S vers processeur	lecture de données transmises par un périphérique
processeur vers E/S	transfert de données vers un périphérique
E/S vers mémoire	accès direct à la mémoire (DMA)
mémoire vers E/S	communication sans passer par le processeur
E/S vers E/S	ex. acquisition

# Bus : définition et structure

---

---

- **Bus** : chemin partagé entre plusieurs unités.
- Un seul équipement transmet à un instant donné.
- Structure : 50 à 100 lignes transmettant des signaux.
- Chaque ligne possède une fonction propre :
  - les lignes de données (bus de données),
  - les lignes d'adresses (bus d'adresse),
  - les lignes de contrôle (bus de contrôle).

# Groupe de fonctions

---

- Bus de données pour les mots de données
- Bus d'adresse pour les adresses
- Bus de contrôle pour l'accès et l'utilisation :
  - des signaux de timing : validité des informations d'adresse ou de données,
  - des signaux de commandes : type d'opérations à effectuer.

# Signaux de contrôle

---

Les plus courants :

memory write écriture de la donnée sur le bus de données  
à l'adresse mémoire du bus d'adresse

memory read lecture à l'adresse mémoire  
indiquée sur le bus d'adresse

I/O write écriture de la donnée sur le bus de données  
sur le port dont l'adresse est indiquée  
sur le bus d'adresse

I/O read lecture sur le port dont l'adresse  
est indiquée sur le bus d'adresse

# Signaux de contrôle

---

Les plus courants :

data ready	les données sont prêtes
data acknowledge	les données ont été reçues
bus busy	le bus est occupé
bus request	l'accès au bus est demandé
bus grant	l'accès au bus a été obtenu
clock	synchronise les transferts
reset	réinitialisation de tous les modules
...	des signaux gérant les interruptions

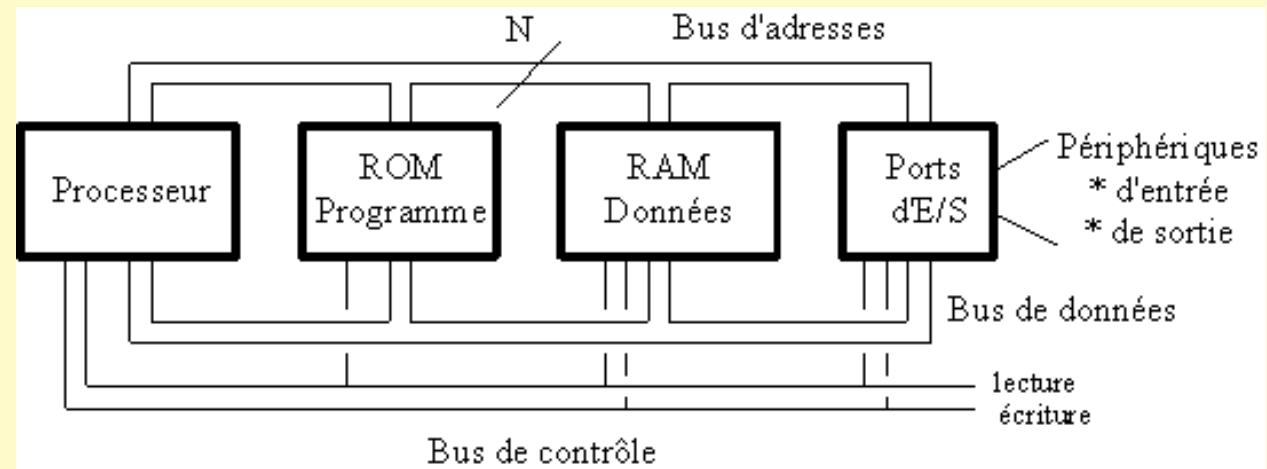
# Fonctionnement schématique

---

---

Une transaction typique se compose de 3 parties

1. l'obtention du bus,
2. l'envoi d'une adresse,
3. l'envoi des données.



# Type de transaction

---

---

Transaction de type écriture :

- obtention du bus
- transmission de données

Transaction de type lecture :

- obtention du bus
- requête au module destination
- attente des données



# Bus = goulot d'étranglement

---

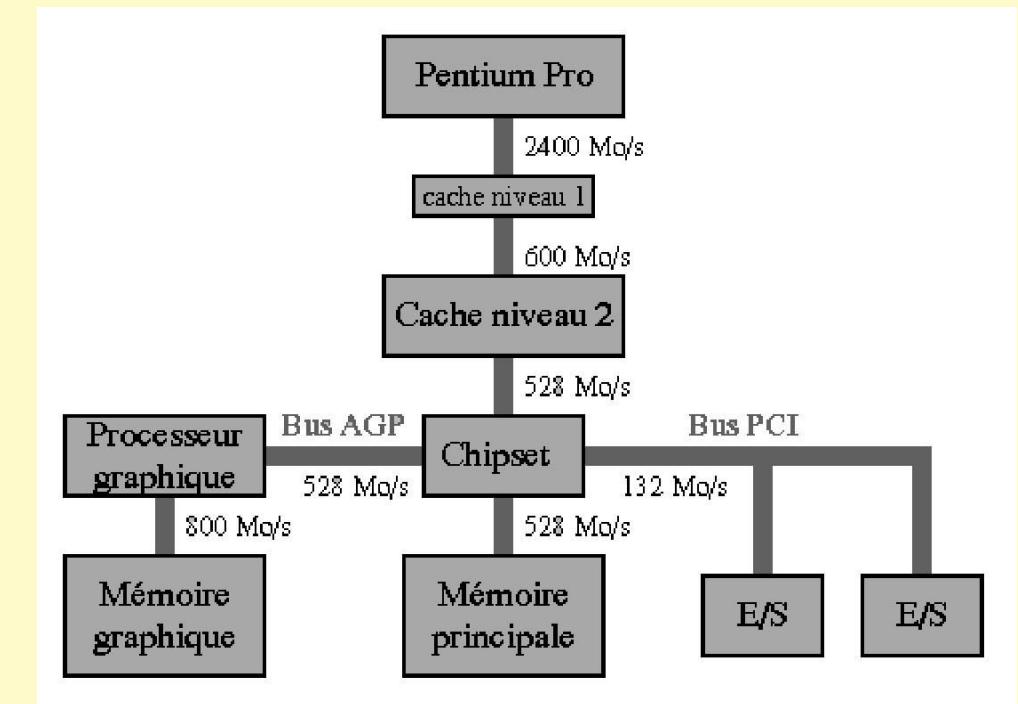
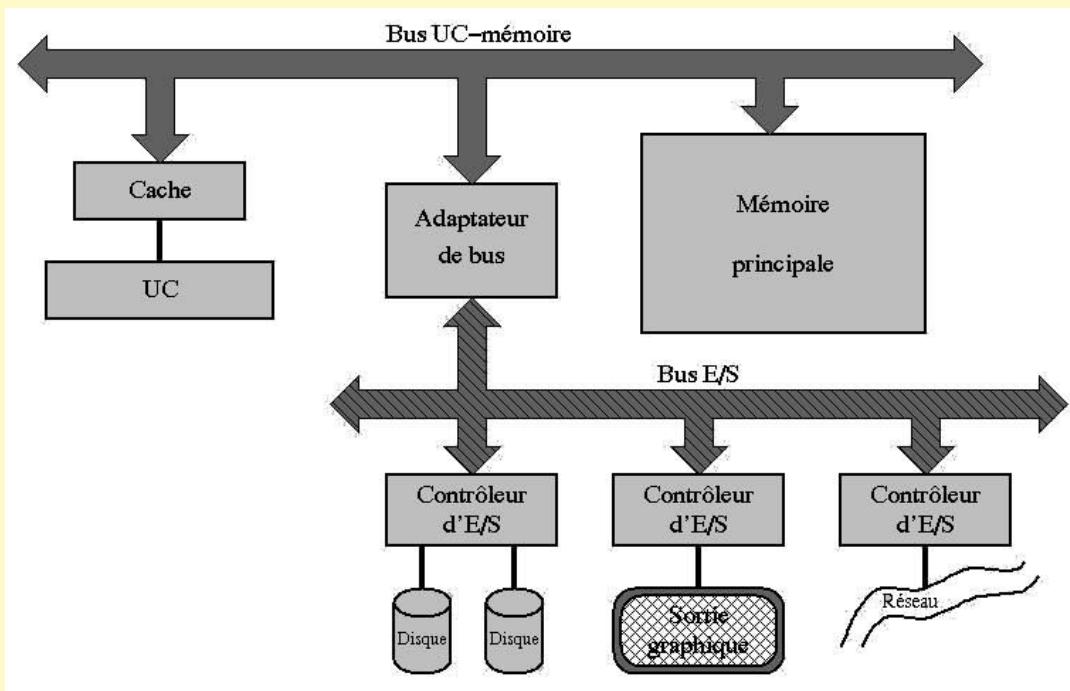
---

- Plus il y a d'équipement connectés `a un bus
- Plus le bus est physiquement long
- Plus les performances décroissent



# Hiérarchie de bus

---



# Hiérarchie de bus

---



**Les règles :**

- 1. séparer communication processeur/mémoire de communication E/S**
- 2. étager en fonction de la performance des unités**

# Deux familles de bus

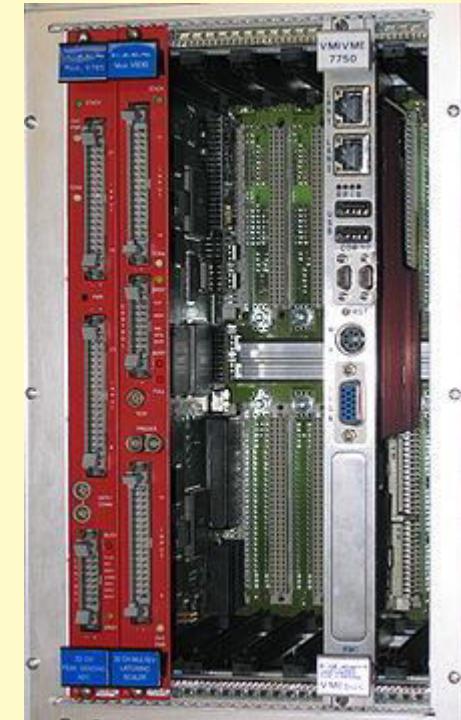
---

- Les bus UC-mémoire (bus système) :

- courts,
- rapides,
- les unités connectées sont connues dès la conception.

- Les bus E/S (bus d'extension) :

- plus long,
- moins rapides,
- offrant une gamme étendue de débits,
- souvent l'objet de normalisation.



# Deux familles de bus

---

- Les bus UC-mémoire (bus système) :
  - les composants conduisent la conception du bus.
  
- Les bus E/S (bus d'extension) :
  - le bus dicte la conception des composants.



# Caractéristiques

---

---

- largeur de bus
- type de transfert
- synchronisation
- arbitrage
- performance



# Largeur du bus

---

---

- Nombre d'information pouvant être envoyées en parallèle.
- Plus la largeur est élevée, plus le débit est grand.
- Elle influe sur :
  - le multiplexage des lignes données/adresse,
  - le nombre d'accès mémoire,
  - la taille de la capacité mémorielle.
- Déterminante pour les performances du système.

# Largeur et bus de données

---

- Bus peu large = multiplexage des données et adresses.
- Bus de données de 8 bits et instructions sur 16 bits.
- Deux accès mémoire pour lire une instruction.
- Typiquement :
  - poids fort : sélection mémoire ou E/S
  - poids faible : sélection port ou emplacement mémoire

# Type de transfert de données

---

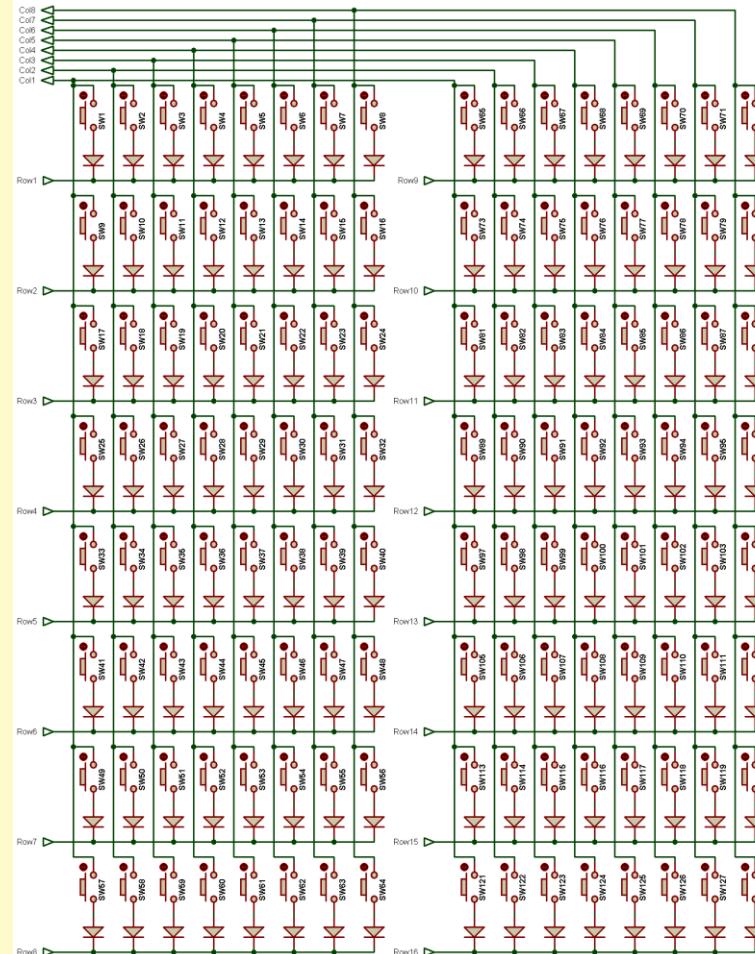
---

- Ecriture multiplexée :

1. adresse placée sur le bus
2. données placées sur le bus

- Lecture multiplexée :

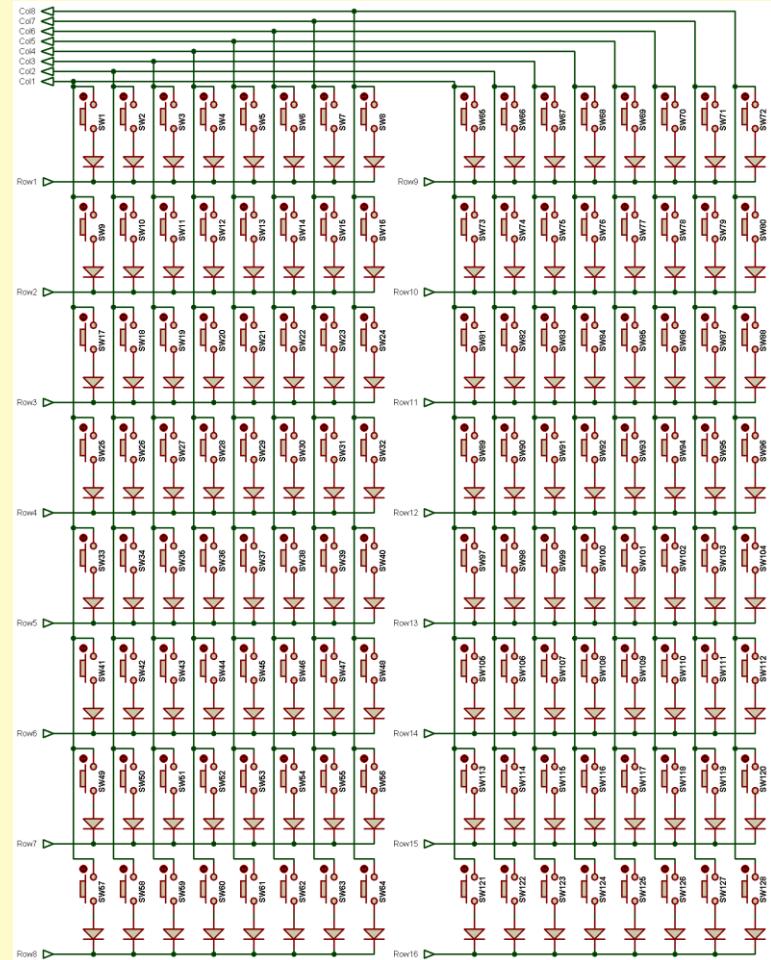
1. l'adresse placée sur le bus
2. temps d'accès aux données
3. données placées sur le bus



# Type de transfert de données

---

- Lecture / écriture non multiplexée :
  - adresse et données placées simultanément sur le bus
- Lecture-modification-écriture :
  1. lecture à une adresse
  2. écriture à cette adresse
- Transfert de bloc de données :
  1. adresse placée sur le bus
  2. données d'adresses consécutives placées sur le bus



# Synchronisation

---

La transmission peut être :

- synchrone
- asynchrone



En général :

- le bus système fonctionne de manière synchrone,
- un bus d'E/S fonctionne de manière asynchrone.

# Technique d'arbitrage

---

- **Maître du bus** : un composant pouvant démarrer une transaction.
- **Esclave** : un composant non maître.
- **Il ne peut y avoir qu'un seul maître à la fois !**
- En cas de plusieurs maîtres de bus potentiels, un mécanisme d'arbitrage est nécessaire.
- Une transaction faisant intervenir plusieurs esclaves est appelée diffusion (broadcast).

# Performances

---

---

Définie par les critères suivants :

- la **bande passante** : quantité d'informations échangées par unité de temps,
- la **latence** : temps de réponse du bus à une requête de transfert,
- la **charge** : nombre maximum d'unités pouvant être connectées,
- la **longueur physique du bus**.

# PLAN DU CHAPITRE

---

---

1  
bus : définition,  
structure et  
caractéristiques

3  
Techniques  
d'arbitrage

4  
Exemple :  
le bus PCI

2  
Synchronisation  
des échanges

5  
Exercice

# Synchronisation des échanges

---

3 manières différentes d'implanter le timing :

- **synchrone** : les évènements ont lieu à des moments précis dans le temps,
- **asynchrone** : les évènements peuvent avoir lieu à des moments arbitraires,
- **semi-asynchrone** : les évènements peuvent avoir lieu de manière asynchrone lors des différentes phases d'une horloge.

# Communication synchrone

---

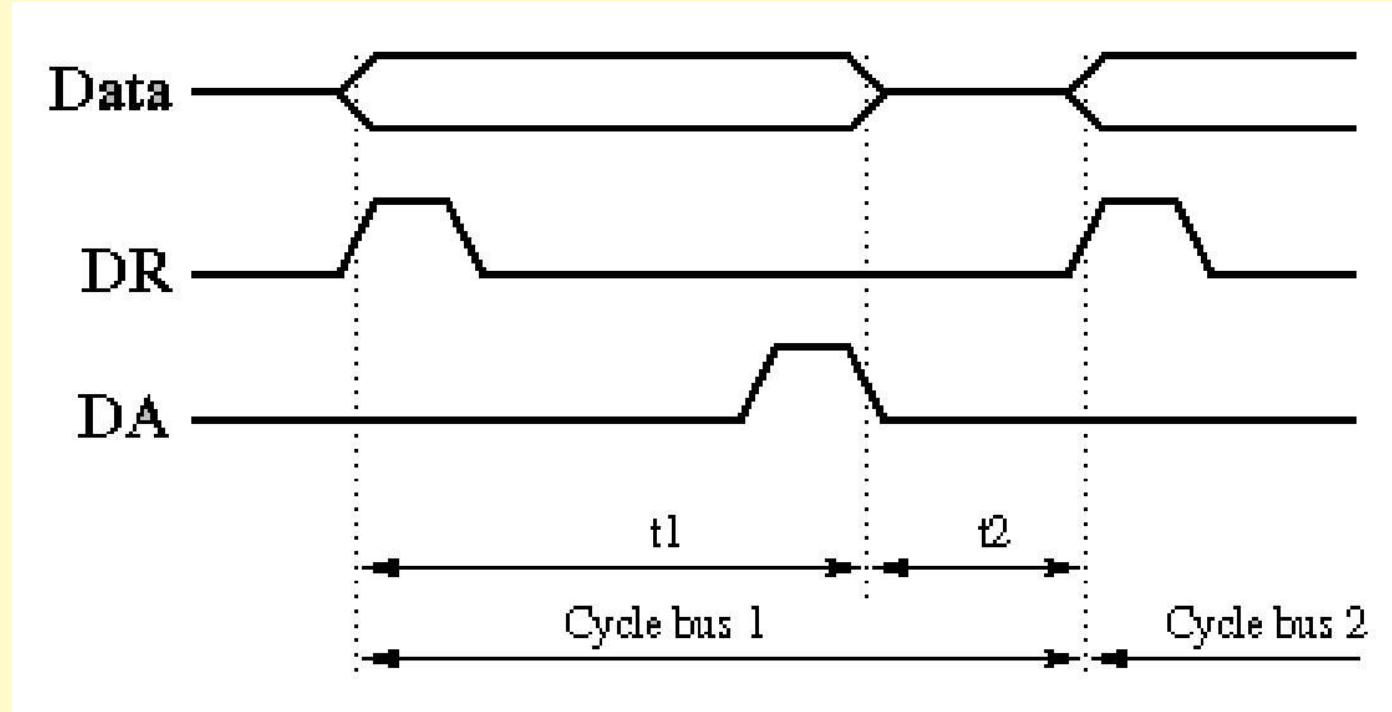
---

- Timing des opérations contrôlé par une horloge.
- Pas de dialogue entre émetteur et récepteur pour le contrôle de l'échange.
- Utilisation de 2 signaux générés par l'horloge :

nom	utilisé par	les données
DR (Data Ready)	l'émetteur	sont placées sur le bus
DA (Data Acknowledge)	le récepteur	ont été reçues

# Chronogramme

---



**Doit être adapté au récepteur le plus lent !**

# Communication asynchrone

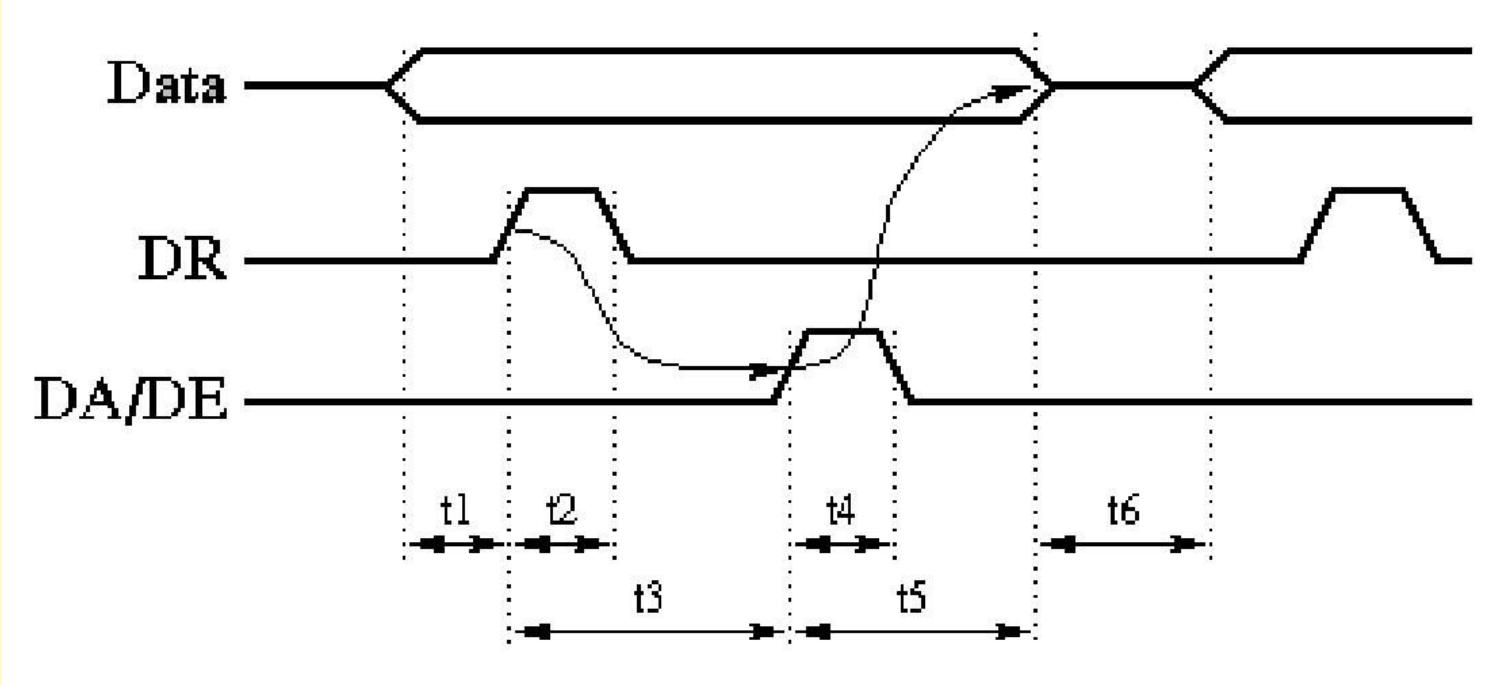
---

---

- Pas d'horloge fixe.
- Signaux générés par les composants.
- Trois type de protocoles asynchrone :
  - non-entrelacé
  - semi-entrelacé
  - complètement entrelacé

# Transaction asynchrone non-entrelacée

---



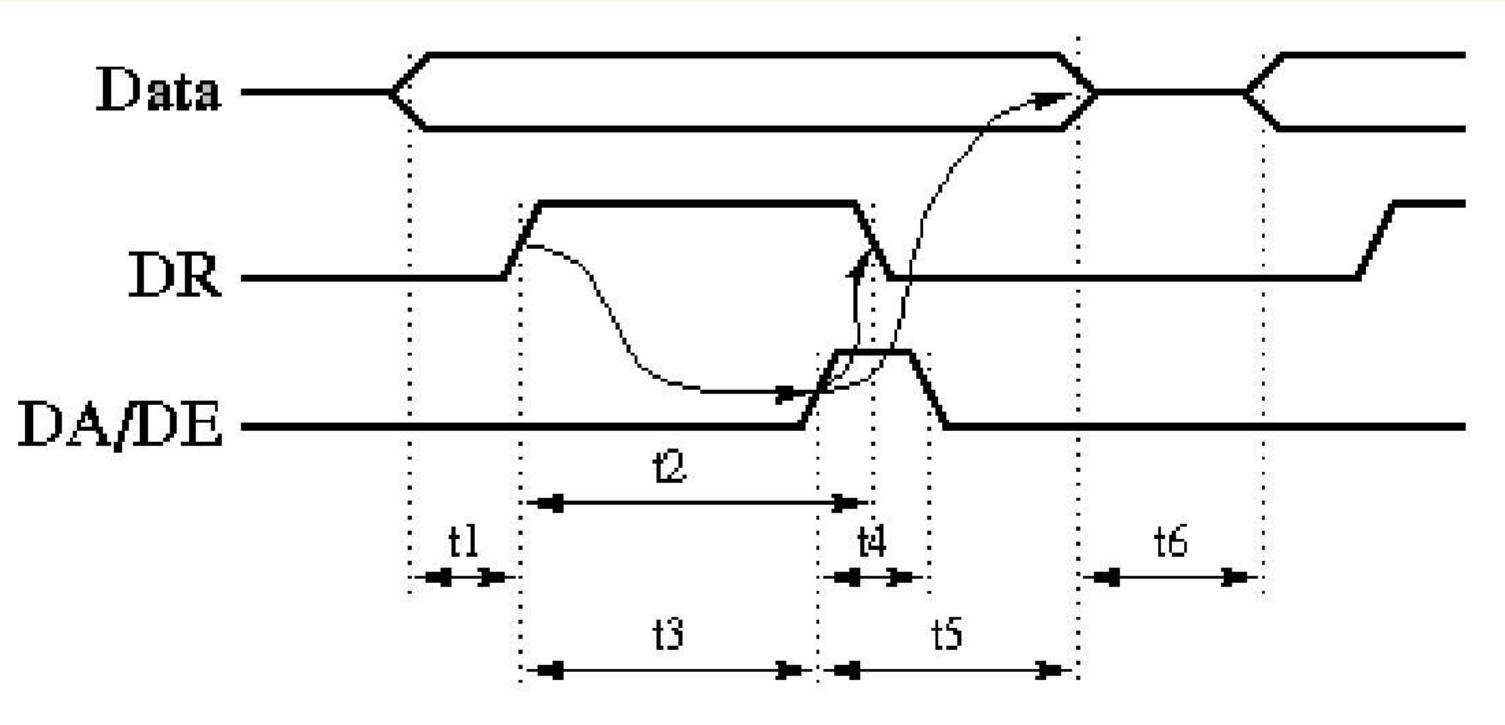
Supposons que l'émetteur soit très rapide.

t5 est inférieur à t4.

Le prochain cycle de bus pourrait commencer avant la retombée du signal DA.

# Transaction asynchrone semi-entrelacée

---



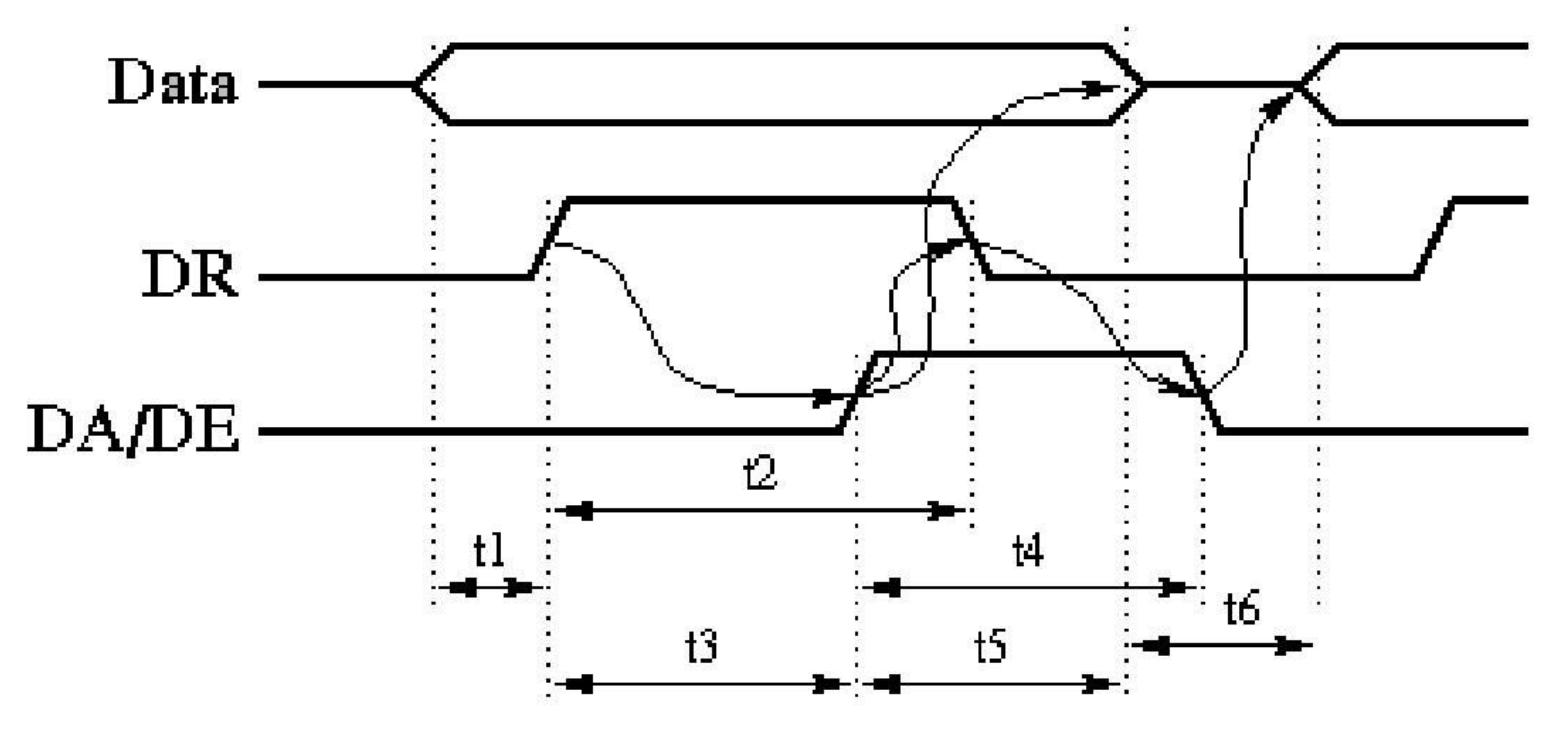
L'émetteur fait retomber le DR en réponse à DA.

Problème partiellement résolu.

L'émetteur peut commencer une nouvelle transaction trop tôt.

# Transaction asynchrone complètement entrelacée

---



L'émetteur est forcé de débuter une nouvelle transaction lorsque DA est retombé.

# Handshake

---

---

Protocole asynchrone complètement entrelacée :

1. l'émetteur place les données sur le bus,
2. l'émetteur lève le signal DR,
3. le récepteur lit la donnée,
4. le récepteur lève le signal DA,
5. l'émetteur remet DR à zéro,
6. l'émetteur retire sa donnée du bus,
7. le récepteur remet DA à zéro.

# Handshake

---

---

- Protocole très souvent utilisé.
- Synchronise des composants de vitesses très différentes.
- *Time-out* pour prendre en compte les pannes de composants.
- Inconvénients :
  - communication ralentie par la transmission des signaux de contrôle,
  - sensible au bruit.

# Communication semi-asynchrone

---

- Les transitions des signaux de contrôle ne peuvent avoir lieu qu'à des instants déterminés par une horloge.
- Le temps entre deux transitions successives peut être variable.
- Modes non-entrelacé, semi-entrelacé, et entrelacé.
- Protocole moins sensible au bruit.

# PLAN DU CHAPITRE

---

---

1  
bus : définition,  
structure et  
caractéristiques

3  
Techniques  
d'arbitrage

4  
Exemple :  
le bus PCI

2  
Synchronisation  
des échanges

5  
Exercice

# Techniques d'arbitrage

---

Garantir un seul maître de bus.

2 types d'arbitrage :

- Statique :

- les candidats deviennent maître à tour de rôle dans un ordre fixé.
- inconvénient : le bus peut être inutilisé (non-opération).
- avantage : simple à mettre en œuvre.
- utilisé avec un protocole synchrone lorsqu'il y a peu de maîtres potentiels.

- Dynamique :

- allocation du bus sur demande, lorsqu'il est libre, à un composant qui le demande.
- utilisation du signal BR (*Bus Request*).

# Arbitrage dynamique

---

Comment départager plusieurs demandeurs ?

- suivant une priorité affectée de manière unique à chaque maître potentiel (bus d'E/S),
- de manière équitable : éviter qu'un demandeur de petite priorité soit constamment rejeté,
- en combinant les deux premières politiques : un choix équitable départage deux demandes de même priorité.

# Arbitrage dynamique

---

La libération du bus peut avoir lieu :

- en fin de transaction,
- sur demande : le maître conserve le bus jusqu'à une nouvelle demande (exemple : le processeur qui demande le plus souvent)
- par préemption : un module prioritaire peut devenir maître avant la fin d'une transaction.

# Mécanismes matériels d'arbitres

---

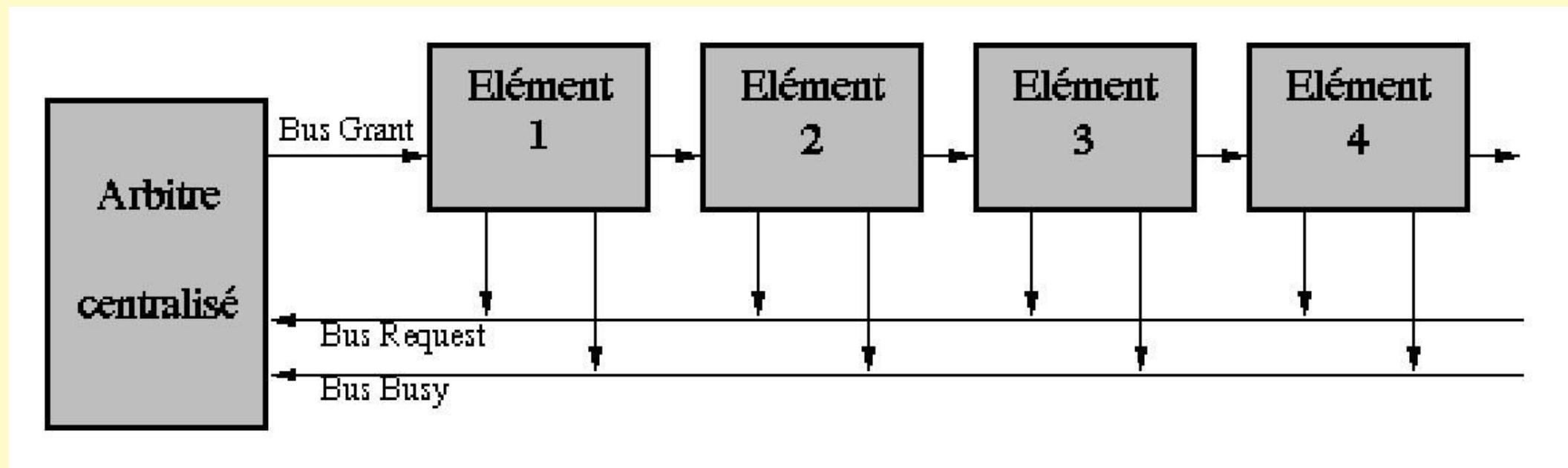
---

Un mécanisme d'arbitrage peut être :

- distribué : réparti sur l'ensemble des modules connectés au bus,
- centralisé :
  - sur un seul des modules connectés au bus,
  - sur un module dédié appelé bus arbitre ou contrôleur de bus.

# Exemple : *daisy chain* (structure en guirlande)

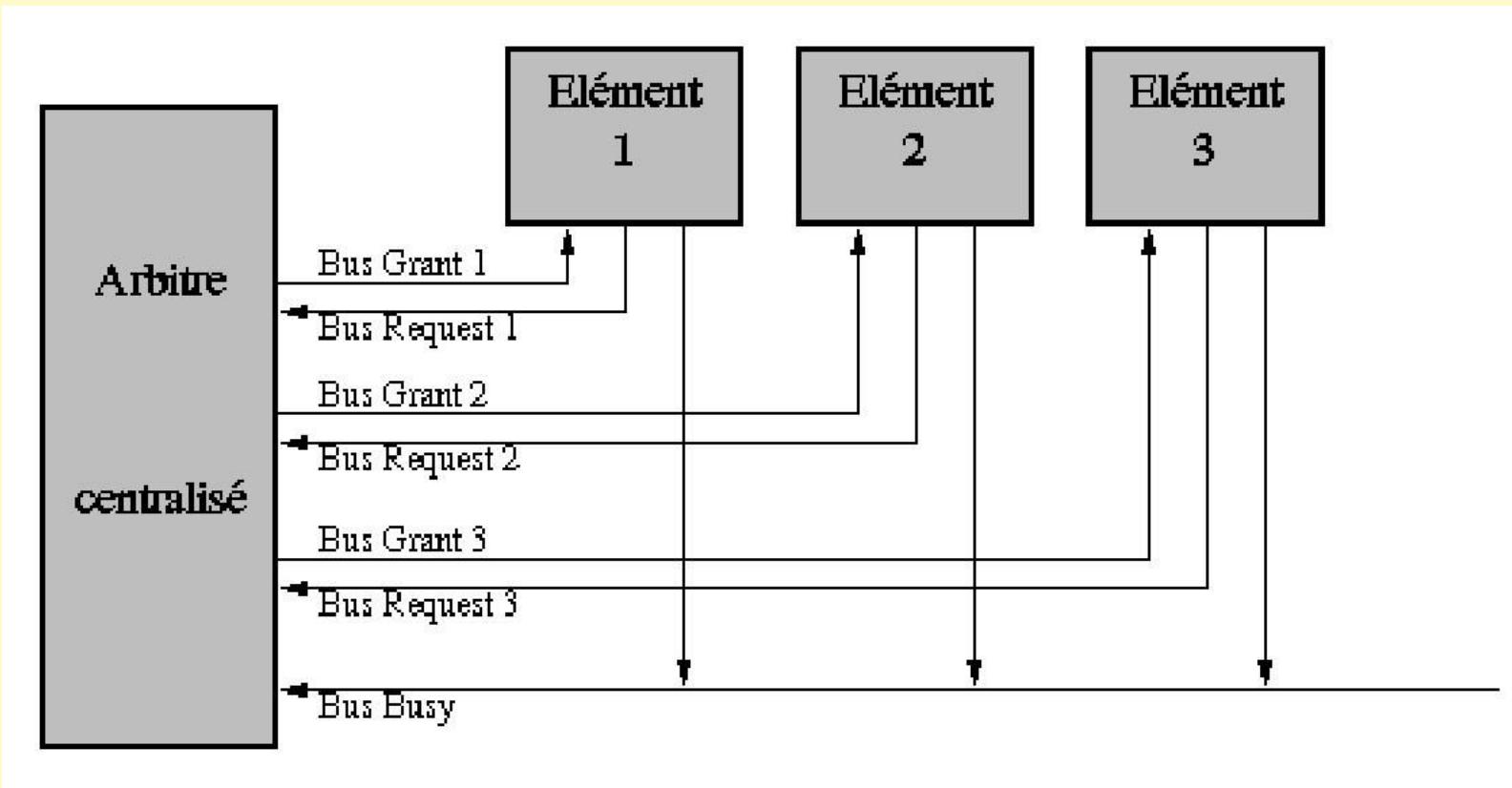
---



# Requête-autorisation

---

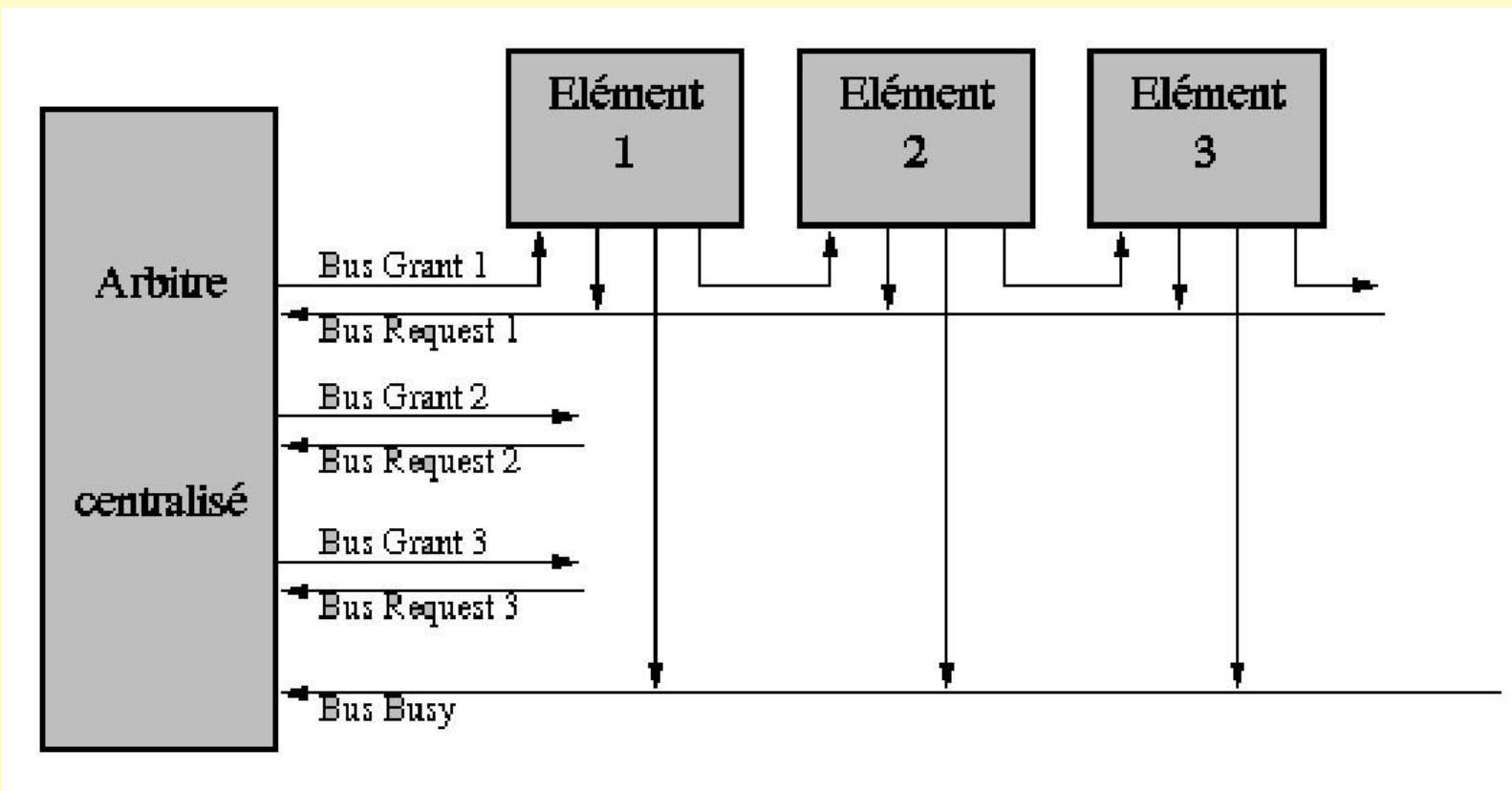
Chaque demandeur dispose de lignes BG et BR propre :



# Arbitrage mixte

---

Combinaison des deux techniques précédentes :



# Arbitrage décentralisé

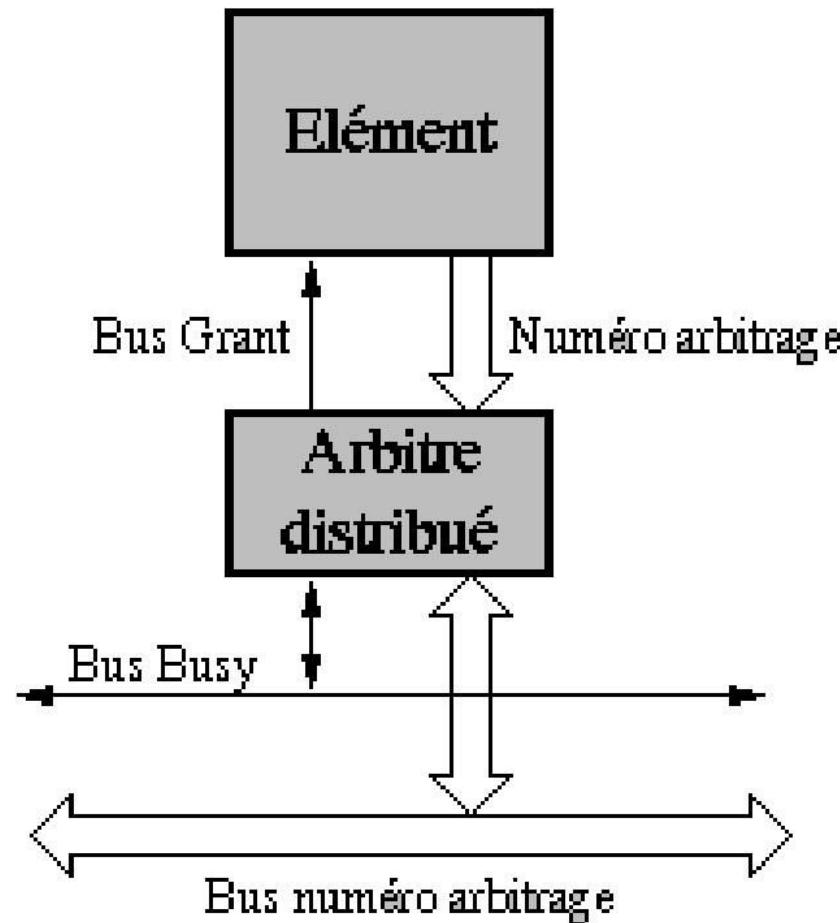
---

---

- L'arbitrage centralisé est très sensible aux pannes.
- Un arbitre décentralisé diminue cette sensibilité.
- Chaque module :
  - possède un circuit d'arbitrage,
  - possède un numéro de priorité unique,
  - dialogue avec son propre arbitre.
- Utilisé dans les configurations multiprocesseur.

# Arbitrage décentralisé

---



- Lorsqu'un composant souhaite avoir le bus :
1. il envoie son numéro de priorité à son arbitre,
  2. l'arbitre place ce numéro sur le bus numéro arbitrage,
  3. calcul du OU logique de tous les numéros de priorité des demandeurs
  4. chaque arbitre compare ce numéro calculé au numéro du composant,
  5. si les deux sont différents, l'arbitre retire son numéro, sinon il le maintient
  6. il reste sur les lignes le numéro du composant pouvant être maître
  7. l'arbitre signale à son composant qu'il est le nouveau maître (BG) et lève BB

# Stratégies d'arbitrage

---

- Départager les demandeurs si la priorité n'est pas fixée statiquement.
  
- Deux notions sont examinées :
  - le niveau de priorité,
  - l'ancienneté de la demande.



# Stratégie linéaire (L)

---

- Un numéro fixe est attribué à chaque demandeur.
- Les numéros sont rangés par ordre de priorité décroissante.
- Exemple : 4 L 3 L 1 L 2
  - le module le plus prioritaire est 4,
  - le module le moins prioritaire est 2,
  - simple à réaliser, mais risque de famine.

# Stratégie circulaire (R)

---

- Les numéros sont placés dans une liste circulaire.
- Le demandeur dont le numéro est placé à droite du numéro du dernier maître devient maître à son tour.
- Exemple : 4 R 3 R 1 R 2
  - si 2 est maître du bus, le module le plus prioritaire est le module 4,
  - plus difficile à réaliser mais évite la situation de famine.

# Stratégie cyclique (C)

---

- Pour chaque demandeur, on garde l'antériorité des demandes précédentes.
- L'arbitre applique une stratégie linéaire sur les modules classés par ordre d'antériorité décroissante.
- Exemple : 1 C 2 C 3 C 4
  - si le plus anciennement servi est :
    - l'élément numéro 4
    - puis l'élément numéro 1
    - puis l'élément numéro 3
    - et enfin l'élément numéro 2
  - alors l'élément le plus prioritaire sera le 4

# Stratégie multiple

---

---

- Combinaison de plusieurs stratégies.
- Exemple : 1 R (3 C 4) R (5 L 6)
  - 5 sera le plus prioritaire si 3 ou 4 ont été servis en dernier.

# PLAN DU CHAPITRE

---

---

1  
bus : définition,  
structure et  
caractéristiques

3  
Techniques  
d'arbitrage

4  
Exemple:  
le bus PCI

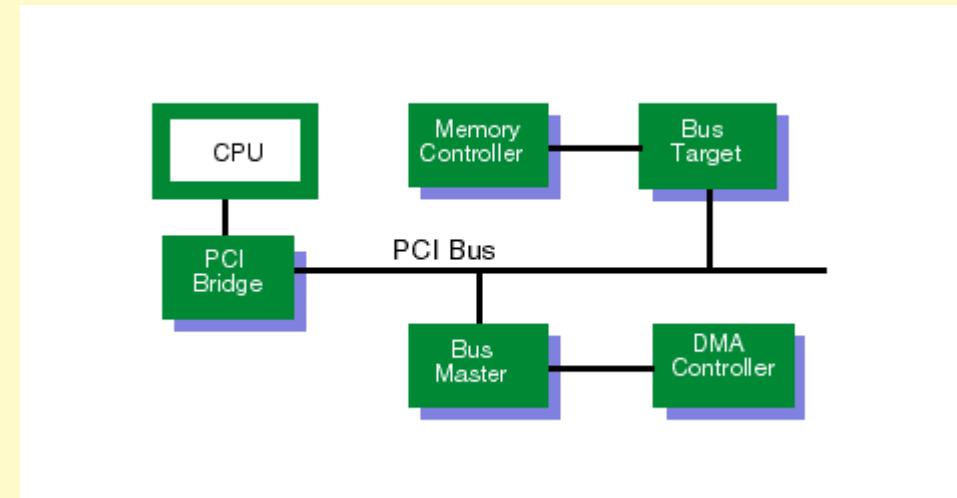
2  
Synchronisation  
des échanges

5  
Exercice

# Le bus PCI (*Peripheral Component Interconnect*)

---

- Standard développé par Intel.
- Utilisé pour les communications avec les E/S.
- Processeur-indépendant, configurable, haut débit.
- Très populaire et très utilisé.



# Caractéristiques (version 2.1)

---

---

largeur	32 ou 64 bits
cadencé à	66 Mhz
débit max	528 Mo/s
transfert	multiplexage adresses et données en rafale
timing	semi-synchrone
arbitrage	centralisé synchrone requête-autorisation caché

# Transactions

---

---

- Des lignes de contrôle pour détailler :
  - type d'opération (lecture/écriture)
  - destinataire (mémoire, E/S)
- Une transaction consiste en :
  - une phase d'émission d'une adresse,
  - une ou plusieurs phase d'émission de données
- Synchronisation des évènements sur fronts descendants.

# Arbitrage

---

---

- Centralisé et reposant sur deux signaux BG et BR pour chaque module.
- La spécification ne fixe pas de stratégie.
- L'arbitre comprend plusieurs types de stratégies (FIFO, circulaire, fixation de priorité).
- L'arbitrage est effectué pendant une transaction.

# PLAN DU CHAPITRE

---

---

1  
bus : définition,  
structure et  
caractéristiques

3  
Techniques  
d'arbitrage

4  
Exemple :  
le bus PCI

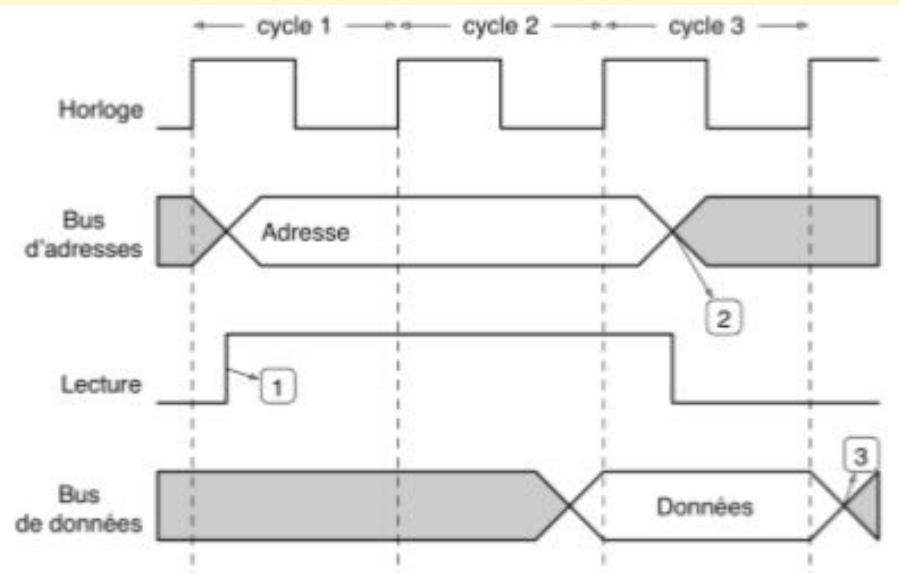
2  
Synchronisation  
des échanges

5  
Exercice

# Le problème : calcul des débits d'un bus

---

1. A l'aide de la figure suivante (bus synchrone), calculer le débit du bus pendant une opération de lecture sachant que sa fréquence est de 100 MHz et que le bus transmet 16 bits à la fois.



2. Un bus a une fréquence de 500 MHz et la mémoire peut envoyer 64 bits sur chaque front d'horloge (haut et bas). Calculer le débit maximum, en ignorant les temps d'accès à la mémoire (en supposant que tous les cycles sont occupés par des données).

