

Résumé de cours

« **Les fondements de  
l'informatique** »  
INFO-H200

Raphael HANNAERT

2015

# Table des matières

<b>1</b>	<b>Codage et traitement de l'information : le binaire</b>	<b>3</b>
1.1	Fonctionnement global d'un ordinateur . . . . .	3
1.2	Alan Turing et la première machine virtuelle . . . . .	3
1.3	Le binaire : en stockage et en traitement . . . . .	3
1.4	Le Transistor (transfer resistor) . . . . .	4
1.5	Traitement de l'information : le binaire . . . . .	7
1.5.1	Introduction au binaire . . . . .	7
1.5.2	Nombres entiers négatifs . . . . .	8
1.5.3	Nombres à virgule flottante . . . . .	10
1.6	Images et sons . . . . .	10
1.7	Regroupement et compression des données . . . . .	11
1.7.1	Compression de l'information(Zip,Jpeg,MP3,...) . . . . .	12
1.7.2	Compaction de l'information . . . . .	12
1.7.3	Corriger l'information . . . . .	12
1.7.4	Encrypter l'information . . . . .	12
<b>2</b>	<b>Fonctionnement intime du processeur</b>	<b>13</b>
2.1	Von Neumann et les éléments du processeur . . . . .	13
2.2	PHO . . . . .	13
2.3	Éléments fondamentaux de l'architecture . . . . .	17
2.4	CISC vs RISC . . . . .	17
2.5	L'adressage . . . . .	18
2.5.1	Adresse absolue . . . . .	19
2.5.2	Adresse registre . . . . .	19
2.5.3	Adresse relative . . . . .	19
2.5.4	Adresse indirecte . . . . .	19
2.5.5	Valeur immédiate . . . . .	19
2.6	Fonctionnement du processeur . . . . .	20
2.6.1	Compteur d'instructions CI . . . . .	21
2.6.2	Registre d'instructions RI . . . . .	21
2.6.3	Accumulateur ACC . . . . .	21
2.6.4	Registre d'état STAT . . . . .	21
2.6.5	Registres de travail . . . . .	21
2.6.6	Registre d'adresse et registre de données . . . . .	21
2.6.7	Important : Schéma de synthèse bloc du fonctionnement du processeur . . . . .	22
2.7	Étapes primitives . . . . .	23
2.8	Séquenceur . . . . .	23
2.9	Pipeline et parallélisme . . . . .	24

<b>3</b>	<b>Les mémoires</b>	<b>25</b>
3.1	Définitions et rappels . . . . .	25
3.1.1	Classifications des mémoires . . . . .	25
3.1.2	Mémoires vives, mémoires de masse et mémoire morte . . . . .	25
3.1.3	Rappel de deux registres importants . . . . .	25
3.2	Hiérarchie des mémoires . . . . .	26
3.3	Première description de la mémoire centrale . . . . .	27
3.4	Mémoires spécialisées, mémoires secondaires et mémoires d'archivage . . . . .	27
3.4.1	Mémoires spécialisées . . . . .	27
3.4.2	Mémoires secondaires ou de masse . . . . .	27
3.4.3	Mémoires d'archivage . . . . .	28
3.5	Les transferts de niveau . . . . .	28
3.6	Fonctionnement de la mémoire centrale . . . . .	29
3.7	Mémorisation : DRAM ou SRAM . . . . .	30
3.8	Le décodeur d'adresses . . . . .	31
3.9	La mémoire virtuelle . . . . .	32
3.9.1	Description . . . . .	32
3.9.2	La pagination . . . . .	33
3.9.3	Gestion des pages : l'unité de gestion de mémoire et TLB : mémoire associative . . . . .	33
3.10	La mémoire cache . . . . .	34
3.11	Scénario compet de l'accès à la mémoire . . . . .	35
3.12	Interconnexions dans l'unité centrale . . . . .	35
3.13	Jeu de composants . . . . .	37
<b>4</b>	<b>Entrées/Sorties et Périphériques</b>	<b>38</b>
4.1	disque dur magnétique . . . . .	39
4.2	disques électroniques (SSD) . . . . .	39
4.3	disques optiques . . . . .	40
4.4	Mémoires électroniques non volatiles . . . . .	40
4.5	Le clavier à touche . . . . .	40
4.6	L'écran . . . . .	40
4.7	Raccordement des périphériques . . . . .	41
4.7.1	Port, Bus et Contrôleur . . . . .	41
4.8	Les périphériques en action . . . . .	42
4.9	Le mécanisme des interruptions . . . . .	42
4.9.1	4 Catégories d'interruptions . . . . .	44
4.10	Transfert de données et DMA (direct access memory) . . . . .	44
<b>5</b>	<b>Les logiciels</b>	<b>47</b>
5.1	Le système d'exploitation (OS) - généralités . . . . .	47
5.1.1	Description générale . . . . .	47
5.2	Multi-tâches et multi-utilisateurs : Organisation en processus . . . . .	49
5.3	gestion des fichiers . . . . .	51
5.4	stockages physique des fichiers . . . . .	52
5.4.1	Partitions et unité d'allocation . . . . .	52
5.4.2	Stockage contigu . . . . .	52
5.4.3	Stockage en liste liée . . . . .	53
5.4.4	Stockage en table indexée . . . . .	53

# 1 Codage et traitement de l'information : le binaire

## 1.1 Fonctionnement global d'un ordinateur

Le fonctionnement de l'ordinateur peut globalement être séparé en trois parties. Les entrées, ou informations, sont d'abord **stockées** puis **traitées** et enfin **transmises** (-> sorties). La partie de traitement est celle la plus intéressante car la plus cognitive. En effet c'est celle qui nécessite des **programmes**. Le traitement qui consiste en calculs, comparaisons, recombinaisons des informations est le fait du processeur et de ses deux unités : l'unité de commande (CU, control unit) qui séquence et choisit les opérations à effectuer et l'unité arithmétique et logique (ALU, arithmetic and logical unit) qui les effectue.

## 1.2 Alan Turing et la première machine virtuelle

C'est à Alan Turing qu'on doit la première machine universelle, spectaculaire dans sa conception, qui aurait permis de déjouer des navires Allemands lors de la guerre 40-45 grâce aux facultés de décryptage que celle-ci présentait. Elle pouvait exécuter n'importe quel programme en prenant une succession d'**états**. La machine peut lire ou écrire sur un très long ruban composé d'une suite de cellules qui fait office d'organes entrées-sorties mais aussi de mémoire puisqu'il peut mémoriser les écritures produites lors d'itérations précédentes. Chaque fois qu'il en reçoit l'ordre, le ruban est déplacé vers la gauche ou vers la droite par pas de longueur fixe, alternant simplement la cellule présentée au dispositif de lecture-écriture. En fait la machine peut prendre différents états internes si bien qu'elle peut présenter différents comportements pour de mêmes entrées en fonction de l'état dans lequel elle se trouve. Ainsi elle peut fonctionner par exécution d'une suite d'instructions(programme) sous la forme "Si la machine est dans l'état m et que la cellule lue contient le symbole + ou rien, alors la machine remplace ou non ce symbole, déplace le ruban vers la gauche ou vers la droite, et adopte le nouvel état n". C'est pour cela qu'on dit que la machine est **universelle**, car elle peut effectuer toutes les tâches voulues en fonction du programme rentré. Schéma et exemple concret dans le livre pages 14-15.

## 1.3 Le binaire : en stockage et en traitement

Encore présent dans nos machines actuelles, le **binaire** présente de nombreux avantages :

- La perception binaire est **facile** à concrétiser et est plus économique, seulement deux états étant possibles : soit le courant passe, soit il ne passe pas.
- C'est plus **robuste** et résiste bien mieux aux perturbations. Une fonction ne serait pas gênée par du bruit vu que uniquement les deux états sont possibles et pas ceux intermédiaires (CF schéma prise de notes).
- Le circuit électronique binaire peut-être étroitement relié à la logique Booléenne, en effet l'équivalent électronique de AND et OR serait qu'un certain type d'assemblage entre circuits doit permettre, dans

un premier cas, que deux circuits faisant circuler un courant en entrée le fassent circuler en sortie et, dans le deuxième cas, qu'il suffit que l'un des deux soit alimenté pour que le courant apparaisse en sortie.

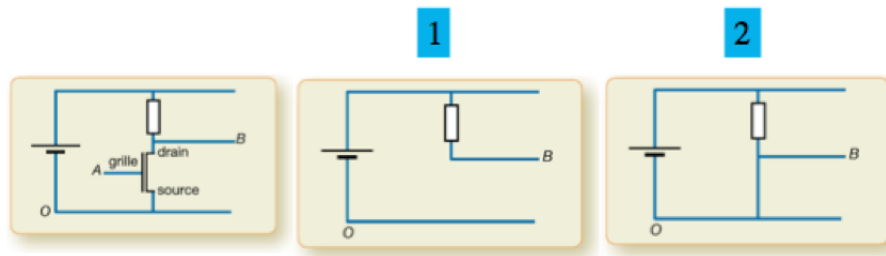
-Le binaire permet de stocker des données de manière permanente (magnétisme, optique, électronique,...).

En conclusion toute information est stockée et traitée en binaire.

## 1.4 Le Transistor (transfer resistor)

La première vague des ordinateurs a été lancée grâce aux développements de constituants importants tels que les bandes magnétiques, le disque magnétique(bouleversement dans le stockage et le traitement de l'information) et surtout le transistor qui feront leur apparition dans les ordinateurs dans le milieu des années soixante et dont le fonctionnement repose sur des phénomènes électroniques complexes.

Il s'agit d'un composant informatique composé notamment de silicium (semi-conducteur) à doper ou non avec du phosphore ou du bore et selon la combinaison, qui amplifie ,bloque ou inverse le courant suivant les conditions auxquelles le transistor est soumis, servant ainsi d'interrupteur qui réalise ces opérations à une vitesse extrêmement rapide. Il a remplacé l'ancienne technologie en place à savoir **les tubes électronique**.



Typiquement un processeur moderne contient des centaines de millions de transistors qui sont gravés ensemble pour ne pas devoir les assembler comme dans le temps,requérant une extrême précision.

Augmenter considérablement le nombre de transistors permet d'augmenter la capacité de traitement et de réduire le délai de transmission entre deux transistors mais augmente aussi les quantité de chaleur dégagée, la complexité et le coût ainsi que la possibilité d'avoir des effets quantiques tels que l'effet tunnel, provoquant des dysfonctionnements.

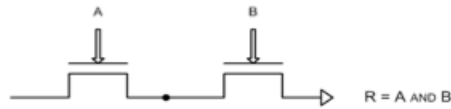
Globalement, comme prévu par la loi de Moore, la densité des transistors sur une puce double tous les 18 mois.

Plusieurs alternatives aux transistors sont en tentative de développement, notamment les nanotubes de carbone qui sont très résistants et capables de différentes propriétés électroniques et qui peuvent améliorer considérablement les performances des processeurs grâce aux diminutions des temps d'accès mémoire. En effet le basculement (passage  $0 \leftrightarrow 1$ ) par l'application d'une force électrostatique peut s'effectuer à des vitesses cent fois plus grandes que les bistables. Cependant les nanotubes se détériorent lorsqu'ils sont mis en contact...

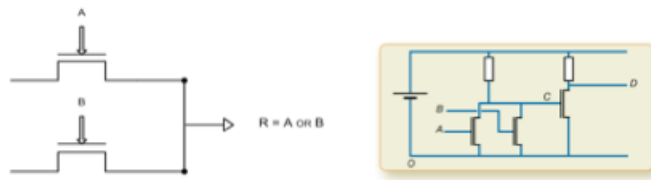
Une autre technologie prometteuse est l'utilisation de la lumière pour les opérations binaires car sa vitesse est imbattable.

Selon la disposition des transistors ( série, parallèle), on peut réaliser les différents fonctions booléennes ( and, or , nor, xor,...), on comprend donc mieux pourquoi nous avons dit plus haut qu'un circuit électronique s'apparentait à un circuit logique.

◆ Mise en série de 2 transistors (fonction logique: AND)










◆ Mise en parallèle de 2 transistors (fonction logique: OR)



NB : Que ce soit pour le AND ou le OR, si  $A = B = 0$  le résultat logique sera 0. Si  $A = B = 1$  le résultat vaudra 1.

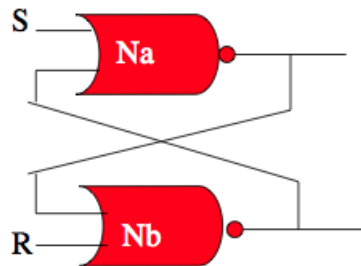
Le schéma suivant illustre la représentation schématique de la logique binaire.

	AND	<table><tr><th>A</th><th>B</th><th>R</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	R	0	0	0	0	1	0	1	0	0	1	1	1	R est VRAI si sont VRAIS A et B
A	B	R																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
	OR	<table><tr><th>A</th><th>B</th><th>R</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	R	0	0	0	0	1	1	1	0	1	1	1	1	A ou B
A	B	R																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
	NAND	<table><tr><th>A</th><th>B</th><th>R</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	R	0	0	1	0	1	1	1	0	1	1	1	0	Pas (A et B)
A	B	R																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
	NOR	<table><tr><th>A</th><th>B</th><th>R</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	R	0	0	1	0	1	0	1	0	0	1	1	0	Pas (A ou B)
A	B	R																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
	XOR	<table><tr><th>A</th><th>B</th><th>R</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	R	0	0	0	0	1	1	1	0	1	1	1	0	(A ou B) et pas (A et B)
A	B	R																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
	XNOR	<table><tr><th>A</th><th>B</th><th>R</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	R	0	0	1	0	1	0	1	0	0	1	1	1	(Ni A ni B) ou (A et B)
A	B	R																
0	0	1																
0	1	0																
1	0	0																
1	1	1																
	NOT	<table><tr><th>A</th><th>R</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	R	0	1	1	0	R est l'inverse de A									
A	R																	
0	1																	
1	0																	

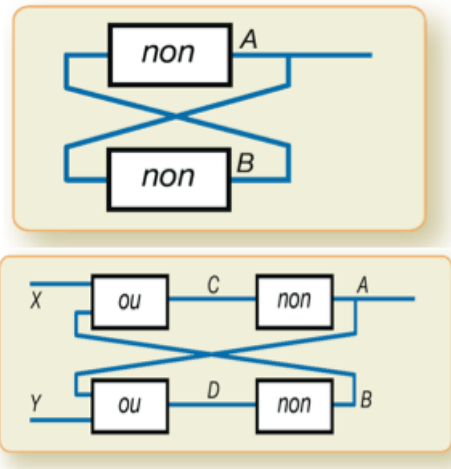
Au niveau de la mémoire on a des éléments à deux états stables, d'où leurs noms : les bistables (flip-flop). Grâce à un circuit bistable on peut mémoriser un bit.

Le schéma ci-dessous montre un bistable formé de 2 NOR associés en boucle. En effet la sortie du premier est l'entrée de l'autre et inversement. La valeur peut donc être piégée à un moment donné et

restera stable tant que rien ne se produit en S et R. Le bit sera donc stocker sous forme de 0 ou 1 c'est à dire que sa valeur sera mémorisée, c'est la base de la mémoire électronique (RAM) sur laquelle nous reviendrons plus loin. Nommons la sortie de Na P et celle de Nb Q et supposons qu'on veuille mémorisée la valeur 1 grâce à Q. Pour cela on rentre en S la valeur 1 et pour toute valeur actuelle de Q P vaudra 0 ( car NOR) et comme R est à 0 ( sinon pas de sens car revient à vouloir écrire 0 et 1 au même endroit) et donc Q passe à 1 et est mémorisé tant qu'on ne modifie pas les valeurs en R et S car le circuit forme une boucle infinie. Ainsi en combinant des bistables on peut mémoriser plusieurs bits, et en faire des registres qui permettent leur lecture. Chaque niveau de complexité est accompagné d'un nouveau niveau d'abstraction. Cette démarche vers l'abstraction est une des caractéristiques de l'informatique, sa nature "fractale". Tout ordinateur peut être construit par la réplication d'un seul type de porte électronique, en de multiples exemplaires connectés d'une manière extrêmement élaborée.



Sur le 1 er schéma ci-dessous on observe les deux états stables de A et B qui satisfont au circuit. En effet les "non" sont les équivalents des NOT du tableau vu précédemment qui correspondent à "A est l'inverse de B" et présentant donc la stabilité si ceux-ci sont effectivement différents ( 0 et 1 ou 1 et 0 respectivement).

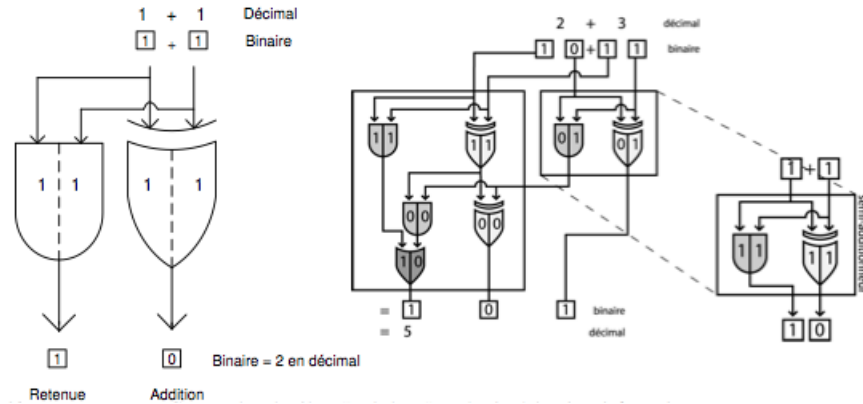


## DEUXIEME SCHEMA A VERIFIER

Le 2ème schéma montre comment on peut mémoriser une valeur entre 0 ou 1 et, en activant une entrée (X ou Y ici), comment change la valeur mémorisée. En effet, si  $A = 1$  les deux portes "ou" sont neutres. Mais si l'entrée  $X = 1$ , comme le "OU" ne permet pas d'avoir simultanément A et X qui valent 1, forcément A prend la valeur 0. A nouveau, si on fait passer le Y à 1 par le meme raisonnement A prend...

Ainsi on peut développer des mémoires plus importantes en combinant des bistables pour former des registres de bistables. Par exemple un registre à 4 bistables constituera une mémoire de 4 bits.

Le transistor peut également effectuer des opérations arithmétiques, en combinant un XOR qui additionne les bits (  $0 + 1$  donne 1,  $0 + 0$  donne 0 et  $1 + 1$  donne 0 ) avec un AND qui retient ( c'est à dire vaudra 0 si 0 et 0 ou 0 et 1 et vaudra 1 si 1 et 1, ce qui suit bien le principe du AND, revoir la liste des expressions en logique binaire plus haut si un doute subsiste).



NB : On notera sur le schéma de droite les différentes combinaisons de retenues et additions pour sommer  $2 + 3$ . L'additionneur est construit avec autant d'étages qu'il y a de rangs de bits à additionner. Chaque étage est constitué par un demi-additionneur réalisée avec une porte XOR pour réaliser l'addition et d'une porte AND pour s'occuper de la retenue. La retenue du rang  $n$  est combinée avec l'addition du rang  $n+1$  pour produire le résultat de l'addition du rang  $n+1$ . Le report du dernier étage est pour sa part ignoré.

Finalement, on retiendra que les transistors en série réamplifient le signal qu'il reçoivent, que la modification du potentiel d'entrée change le potentiel de sortie avec un temps de changement de l'ordre de la nanoseconde mais justifiant donc le délai de réponse, et que les transistors permettent d'effectuer des calculs en plus de raisonner et mémoriser.

## 1.5 Traitement de l'information : le binaire

### 1.5.1 Introduction au binaire

Le sens(valeur) d'un **bit** dépend de son contexte d'utilisation, il est donc nécessaire d'établir des standards, par exemple pour le codage des caractères informatiques. Initialement il y avait **ASCII**, mais ne présentait que 7 ou 8 bits, soit 128 ou 256 espaces de stockage d'un caractère (posant un problème pour des langages tels que le chinois...) donc 256 possibilités (  $\rightarrow$  pour les nombres on peut stocker de 0 à 255).

Ce qui n'était pas suffisant donc **UNICODE** a été développé, constitué de 16 ou 32 bits ( $2^{16}$  ou  $2^{32}$  caractères différents possibles à stocker). Ce qui prend pas mal de place.

Finalement **UTF-8** , une déclinaison de UNICODE qui est plus ingénieuse. Elle permet de coder les caractères les plus fréquents sur le - de bits  $\rightarrow$  Codage du caractère en fonction de sa fréquence d'utilisation. Cette réduction du nombre de bits en fonction de la fréquence des caractères usités est propre au mécanisme de compression d'information que nous allons découvrir par la suite.

Soit des chiffres entiers non signés codés sur 8 bits ( = 1 octet  $\simeq$  1 byte)

NB : J'ai mis le signe  $\simeq$  car en anglais comme en français, si l'on veut explicitement désigner une quantité de huit bits, on utilise le mot octet ; tandis que si l'on veut exprimer l'unité d'adressage indépendamment du nombre de bits, on utilise le mot byte, qui maintenant vaut 8 bits dans notre domaine d'étude mais dans d'autres domaines ou précédemment il a pris d'autres valeurs (5,6,7,9).



Position	8	7	6	5	4	3	2	1
Signific.	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Valeur	128	64	32	16	8	4	2	1

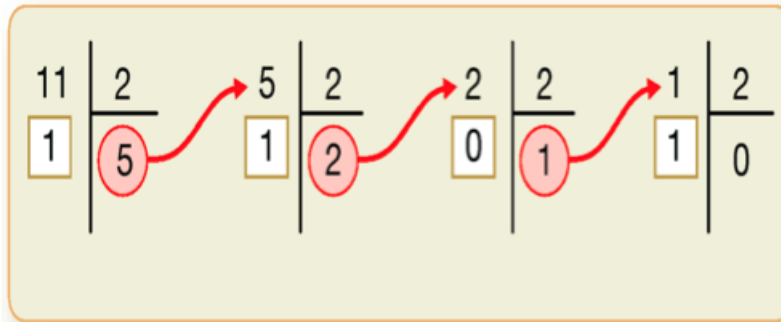
La clé permet d'interpréter les bits.

Par exemple 01010101 donnera :

$$0 \times 128 + 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 85$$

Et de la même façon  $11111111 = 255$ , puisque  $00000000 = 0$ .

Le schéma ci-dessous permet de trouver la représentation binaire de tout nombre. Il est nécessaire d'être familier avec cette conversion ! L'exemple convertit 11 en 1011.



On conclura que les entiers non-signés codés sur  $x$  bits permettent de stocker les valeurs allant de 0 à  $2^x - 1$

$$\begin{array}{r}
 8 = 00001000 \\
 + 9 = 00001001 \\
 \hline
 = 17 = 00010001
 \end{array}$$

### 1.5.2 Nombres entiers négatifs

Maintenant si on désire représenter un signe négatif, il est donc nécessaire de dédier 1 bit au signe. La notation dite en **2 complément** permet de coder aisément les nombres binaires en prenant la notation binaire de la valeur absolue du nombre, en y inversant chaque bit et en ajoutant 1 ( en binaire ) au résultat. Ce qui donne pour coder 3 sur 4 bits :  $0011 \rightarrow -3 = 1100 + 0001 = 1101$

NB : on remarquera que le 1er bit, "le bit le plus significatif", nous informe sur le signe. Ainsi on peut stocker sur  $x$  bits les valeurs de  $-2^{x-1}$  à  $2^{x-1} - 1$ . Cette notation permet de n'avoir qu'un seul zéro et de pouvoir traiter l'addition de nombres négatifs et positifs de la même manière. Le bit le plus significatif ou MSB(most significant bit) sera à 0 pour les nombres positifs et à 1 pour les négatifs.

Nombre	Binaire
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
+1	0001
+2	0010
+3	0011
+4	0100
+5	0101
+6	0110
+7	0111

Un peu d'arithmétique élémentaire en 2's complément :

$$\begin{array}{r} \text{EX1} \\ 0011 \text{ (3)} \\ + 0010 \text{ (2)} \\ \hline = 0101 \end{array}$$

$$\begin{array}{r} \text{EX2} \\ 0011 \text{ (3)} \\ + 1110 \text{ (-2)} \\ \hline = 0001 \text{ (1)} \end{array}$$

(on s'est débarrassé du bit en trop)

$$\begin{array}{r} \text{EX3} \\ 1010 \text{ (-6)} \\ + 1100 \text{ (-4)} \\ \hline = 0110 \rightarrow \text{OVERFLOW} \end{array}$$

(détectée par le fait que le bit significatif de la somme (0) est  $\neq$  de ceux des deux nombres (1),  
l'addition est donc infaisable d'où le Overflow.

En quelque sorte la notion de différence disparaît, on somme juste des termes signés. ( 4 + (-3)). Il y a Overflow quand le nombre ne tient pas sur le nombre maximum de bits dont on dispose.

Par le procédé ADD and SHIFT on ramène une multiplication à une addition.

$$\begin{array}{r} \phantom{*} 00011 \text{ (3)} \\ * \phantom{000} \underline{00011 \text{ (3)}} \\ \phantom{*} 00011 \\ \phantom{*} 00011 \\ \phantom{*} 00000 \\ \phantom{*} 00000 \\ \phantom{*} \underline{00000} \\ \phantom{*} 000001001 \end{array}$$

### 1.5.3 Nombres à virgule flottante

Un nombre réel est constitué de 3 composantes à savoir son **signe**, son **exposant** et sa **mantisse** (partie après la virgule). 64 sont répartis de la manière suivante : 1 bit pour le signe, 11 pour l'exposant et 52 pour la mantisse. L'exposant est compris entre -1022 et 1023, on le trouve en soustrayant 1023 (CF plus bas).

Trouver le nombre à virgule représenté par le mot  
11000100011010010011110000111000000000000000000000  
0000000000000000

*Le signe est représenté par 1.  
L'exposant est représenté par 10001000110.  
La mantisse est représentée par  
1001001111000011100000000000  
000000000000000000000000.  
Le signe du nombre est donc -.  
Le nombre 100 0100 0110 est égal à 1 094 et  
l'exposant du nombre est  $n = 1094 - 1023 = 71$ .*

$$\begin{aligned} m &= 1.1001\ 0011\ 1100\ 0011\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000 \\ &= 1 + 1/2 + 1/2^4 + 1/2^7 + 1/2^8 + 1/2^9 + 1/2^{10} + 1/2^{15} + 1/2^{16} + \\ &\quad 1/2^{17} \\ &= (2^{17} + 2^{16} + 2^{13} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^2 + 2 + 1) / 2^{17} \\ &= 206727/131072. \end{aligned}$$

*Le nombre représenté est donc  $-206\ 727 / 131\ 072 \times 2^{71}$   
 $= -3.724... \times 10^{21}$ .*

Actuellement il existe des ANSI/IEEE standards pour ces représentations et opérations des virgules flottantes. Tous les processeurs les traitent de la même façon.

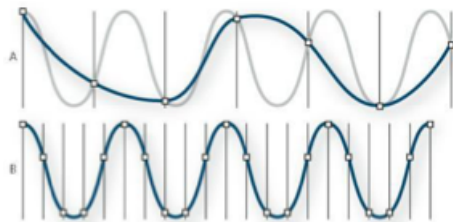
Les additions et soustractions sont plus compliquées que les multiplications car il faut aligner les nombres (SHIFT) puis renormaliser le résultat. Pour la multiplication, il suffit de faire un shift final.

Il y a un grand nombre d'opérations arithmétiques qui portent et sur l'exposant et sur la mantisse.

## 1.6 Images et sons

Si on vise l'économie, ce qui en matière de stockage est assez courant, certaines images peuvent être codées juste à partir des propriétés géométriques des différents éléments qui la composent. Par exemple un cercle dans une image est défini par les coordonnées de son centre au sein de l'image, par son rayon et par sa couleur. En général l'image est perçue comme un ensemble de segments paramétrés positionnés relativement l'un à l'autre. C'est la version **PostScript** : On utilise un codage dit "vectoriel", permettant de limiter l'espace. L'avantage est que chaque élément peut être facilement redimensionné. En revanche il n'est généralement pas possible de recoder sous forme vectorielle une image acquise, par exemple, au moyen d'un scanner ou d'un appareil photo. Dans la version **BitMap**, on code une image à partir de la superposition sur celle-ci d'un quadrillage avec un contenu plus ou moins fin. Encore utilisée actuellement, cette version découpe l'image en Pixels ( Picture Element Image) et stocke octet par octet. Pour stocker un pixel coloré il faut généralement 3 octets ( 3 x 8 bits) pour les 3 couleurs primaires. La fidélité de la représentation est fonction de la densité du quadrillage (on parle de résolution de l'image) et de la richesse des nuances dans la description de chaque cellule.

Le son est un phénomène physique ondulatoire. Cette représentation simple est fidèle à la réalité mais elle implique une succession continue de valeurs, elles-mêmes continues, alors que l'ordinateur n'accepte de stocker et de traiter l'information que si celle-ci lui est présentée sous forme binaire. Il est donc indispensable de convertir les signaux continus en valeurs numériques entières et puis en binaire. On opère donc à une **discrétisation ou digitalisation ou enore numérisation**. L'onde est échantillonnée dans le temps et chaque échantillon est codé sur un certain nombre de bits. C'est le rôle de L'Analog to Digital Converter (ADC). La précision dépend de la fréquence d'échantillonnage et des intervalles de numérisation (qui doivent être respectivement la plus grande et les plus petits possibles) pour se rapprocher du signal d'origine.



Ensuite pour rejouer le son il faut le reconvertir, cette fois-ci l'opération est effectuée par le Digital to Analog Converter (DAC).

Par exemple en téléphonie commutée, la voix est généralement transmise sous forme analogique jusqu'au commutateur où elle est échantillonnée 8000 fois par seconde et codée sur 8 bit. La voix est dès lors transmise avec un débit de 64 000 bits par seconde.

Dans le son comme pour les images, c'est la sensibilité de l'oeil humain qui détermine la richesse de la palette de couleurs perceptibles et donc par déduction le nombre de bits nécessaires pour permettre toutes les nuances possibles.

## 1.7 Regroupement et compression des données

Sans les mécanismes de compression, il faudrait près de 86Go pour une heure de film. On atteint ici des volumes d'information qui dépassent largement la capacité de stockage des supports. Il est donc indispensable de compresser les images et autres au maximum par des technologies suffisamment standardisées pour que les dispositifs logiciels de visualisation s'y retrouvent.

Rappel : Byte  $\neq$  Bit, le byte étant généralement plutôt une mesure de stockage et le bit de vitesse de transmission.

Par ailleurs, pour anticiper sur la suite dont les supports magnétiques et optiques, on peut déjà préciser que les bits y sont groupés en bloc de plus grande taille ( ce qui permet de tout déposer et extraire en un coup) par exemple en "secteurs" ( 52 octets).

La notation hexadécimale est une écriture plus concise que le binaire, c'est une manière simple de représenter un demi-octet à l'aide des chiffres de 0 à 9 et des lettres majuscules de A à F. Regroupement de l'information :

Nombre de bits	Appellation anglaise	Appellation française	Unité	Puissance 10	Valeur	Puissance 2	Valeur
1	Bit	Bit	Kilo	$10^3$	1.000	$2^{10}$	1024
4	Half-byte	Demi-octet	Méga	$10^6$	1.000.000	$2^{20}$	1.048.576
8	Byte	Octet	Giga	$10^9$	1.000.000.000	$2^{30}$	1.073.741.824
16	Word	Mot	Téra	$10^{12}$	1.000.000.000.000	$2^{40}$	1.099.511.627.776
32	Double Word	Mot double					
64	Quad Word	Mot quadruple					

La réduction du poids des données peut se faire suivant deux approches :

### 1.7.1 Compression de l'information(Zip,Jpeg,MP3,...)

Un mécanisme de compression des données est un processus totalement **réversible** et le seul qui soit acceptable pour des informations textuelles ou numériques pour lesquelles aucune perte n'est à l'évidence tolérée. Le fait de coder les caractères les plus fréquents sur moins de bits est un exemple typique de compression. Ce processus exploite les mécanismes de redondances : une lettre ou un mot ou encore un pixels qui revient souvent sera codé sur moins de bits. Si on connaît tables de traduction on peut dézipper et donc lire l'information qui a été codée de manière à prendre - de place, cette table est indispensable !

### 1.7.2 Compaction de l'information

C'est une approche alternative qui concerne davantage les sons et les images. Le compactage consiste à alléger le codage des données tout en consentant une dépréciation de la qualité de l'information, considérée comme acceptable. Ce mécanisme perd de l'information, l'information est donc quelque peu dégradée. (suppression de fréquences inaudibles, diminution de la résolution d'une image,...) C'est un processus **irréversible**, l'information dégradée ne peut être reconstituée dans son état original. Cependant le gain en volume de stockage est souvent aussi considérable qu'irréversible. La compression est symétrique, pas la compaction. La notion de symétrie sera expliquée dans la partie encryptage de l'information.

Il est intéressant de noter que les films sont assez facilement compressés car on ne stocke pas les images mais les **différences** entre les images qui composent le film. ( Car les images sont souvent répétitives à peu de choses près).

### 1.7.3 Corriger l'information

Quel que soit le mode de regroupement choisi, ce dernier a toujours pour objectif de constituer des ensembles de bits afin d'en discrétiser le transfert. Quand on transfère de l'information il est nécessaire de le faire par nombre prédéterminés de bits, de taille fixe. Comment remarquer une erreur de transfert par exemple un bit 0 converti en 1 lors du transfert d'un octet ? Une première solution serait de joindre à l'octet deux copies, puis de regarder dans les suites de triplés là où justement on a pas un triplé, c'est là que le bit a changé de valeur, l'erreur est repérée mais ce fut onéreux en bits. Une autre méthode plus économe est celle du bit dit "**de parité**". Ce bit est mis à 1 si le nombre de 1 dans l'octet est pair et à 0 sinon. Cela permet de repérer une erreur de transfert pour un nombre impair de bits en erreur. Corriger des erreurs requiert l'**insertion de bits supplémentaires** (redondants)

### 1.7.4 Encrypter l'information

L'encryptage des données consiste en la protection de celles-ci. Toute technologie de cryptage se base sur la connaissance d'une clé qui détient toujours le secret de l'écriture et de la lecture et qui est unique si on parle de cryptographie **symétrique**. Cette seule clé doit donc parvenir au destinataire le plus discrètement possible, ce qui est complexe, si bien que dans un processus de communication on préférera recourir à une cryptographie **asymétrique**. Celui-ci fonctionne sur le principe de clé publique - clé privée : le destinataire reçoit la clé publique et encrypte le message que seul vous, avec la clé privée pouvez décrypter. Donc l'algorithme qui permet de transformer l'information est connu de tous mais pas la clé.

## 2 Fonctionnement intime du processeur

### 2.1 Von Neumann et les éléments du processeur

Von Neumann a défini l'architecture d'un ordinateur en décrivant les caractéristiques de ses cinq composants essentiels :

- L'unité de commande** qui contrôle le déroulement successif des instructions du programme.
- L'unité arithmétique et logique** qui effectue les opérations de calcul et de comparaison.
- La mémoire centrale** qui contient les programmes et les données.
- L'unité d'entrée** qui s'occupe de l'acquisition des données.
- L'unité de sortie** qui gère la restitution des résultats.

L'unité de commande(CU,control unit) et l'unité arithmétique(ALU,arithmetic and logical unit) forment deux entités distinctes, la première s'occupant de ce qu'il faut faire et la deuxième le réalisant. Les deux sont généralement regroupées sous l'appellation "**unité centrale de traitement**" (**CPU, central processing unit**) ou "**processeur**"

### 2.2 PHO

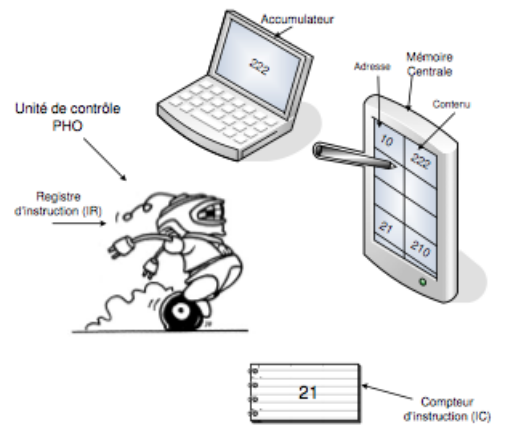
Un ordinateur exécute des programmes, c'est-à-dire une séquence d'instructions. Une fois traduites en instructions élémentaires, le PHO (Petit Homme Ordinateur) prend le relais.

Ce PHO a été conçu par Von Neumann.

Concrètement, le PHO cherche l'adresse de l'instruction, il la lit et la décode avant de l'exécuter et finalement, il incrémente le compteur.

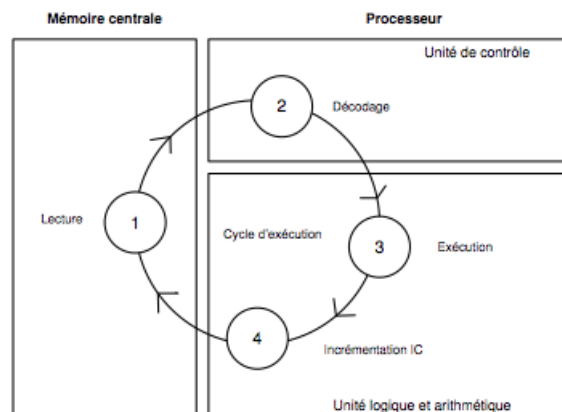
C'est ce qu'on appelle l'Informatique Séquentielle et Programme chargé et stocké en mémoire, impliquant un CPU, ALU des mémoires et registres.

Les **registres** sont des dispositifs auxiliaires associés au processeur et capables, pendant un temps limité, de mémoriser ou de transformer les informations. Les registres se distinguent de la mémoire centrale car ils ont une **capacité de stockage extrêmement réduite**. Ils sont en petit nombre et isolés avec des emplacements et fonctions spécifiques, alors que la mémoire centrale est constituée d'un très grand nombre de cellules adjacentes, différenciées par leur seule adresse.



Nous savons que chaque partie mémorisée est accessible par une adresse, c'est ce qui est représenté sur le schéma pour la **mémoire centrale**, sachant que sur 64 bits on peut stocker  $2^{64}$  infos en mémoire. Le processeur contient trois registres. Le **registre d'instruction (IR)** stocke l'instruction. Pour la faire fonctionner on nécessite l'**accumulateur** et finalement le **registre compteur d'instruction** est incrémenté pour que le programme puisse s'exécuter.

Le PHO va donc chercher l'instruction à l'adresse 21 et y trouve l'instruction 21 et la met dans sa tête (registre d'instruction) Cette étape d'extraction de l'instruction est dénommée l'instruction "fetch". En fait ce n'est pas une extraction à proprement parler, mais une "copie" du contenu de la mémoire dans une autre zone mémoire ou un registre. 210 est séparé en 2 parties à savoir 2 et 10. Le 2 est la partie qui indique de quel type d'instruction il s'agit (addition, transfert vers fichier A,...), c'est le **code de l'instruction**. Par exemple si 2 représentait une addition : 10 serait ce qu'il faut additionner ou plutôt son adresse donc le PHO irait à l'adresse 10, y trouverait 222 comme valeur et l'additionnerait. A quoi ? A ce qu'on avait précédemment c'est à dire qu'il faut bien voir l'addition non comme  $A + B$  avec A et B deux valeurs données mais plutôt comme "je rajoute A" à ce que j'ai déjà dans l'**accumulateur**. Donc la partie exécution se déroule et ensuite le IC est incrémenté de 1 et donc le PHO passe de l'adresse 21 à 22 pour une nouvelle instruction. En pratique 2 n'est pas une addition mais un déplacement de données comme dans 80% des instructions.



La partie décodage sépare 210 en 2 et 10.

C'est suivant ce schéma que TOUS les processeurs fonctionnent ! Cependant, ils diffèrent et sont incompatibles car chacun d'eux possède un ensemble d'instructions ou **jeu d'instructions** qui lui est

propre. Chaque processeur parle un langage particulier construit à partir de ses instructions élémentaires.

Les quatre classes d'instructions élémentaires sont les **déplacement et copie de données, les opérations Arithmétiques et Logiques, les opérations Conditionnelles et de Branchement ou sauts d'instruction (Conditionnels ou Inconditionnels), ainsi que les Opérations d'entrée/sortie avec les périphériques**

Les instructions sont composées de deux parties : **le code instruction et l'adresse de l'opérande**"

Au début il n'y avait que des instructions élémentaires réalisées une par une mais ensuite on a développé des langages de programmation de haut niveau (LPH) dont une instruction réalise plusieurs instructions élémentaires.

Par exemple "c=a+b" est plus simple que la suite d'instructions élémentaires "load a,reg1","load b,reg2",... qui donnent le même résultat.

Les instructions qui suivent seront utilisées plus loin dans ce chapitre. Il est donc intéressant de lire leur signification. De même l'adresse des données et des instructions sera utile pour comprendre le 3ème tableau ( du LPH au binaire) mais n'est pas à retenir.

Codes instruction			Description de l'instruction
Symbole Assembleur	Valeur en binaire	Valeur en décimal	
LOAD	010	2	Copie dans l'accumulateur le contenu de la mémoire dont l'adresse est donnée dans l'opérande
STORE	011	3	Copie le contenu de l'accumulateur dans la mémoire à l'adresse donnée dans l'opérande
ADD	100	4	Ajoute à l'accumulateur le contenu de la mémoire dont l'adresse est donnée dans l'opérande
MPY	101	5	Multiplie l'accumulateur avec le contenu de la mémoire dont l'adresse est donnée dans l'opérande
BR	001	1	Interrompt inconditionnellement le déroulement séquentiel du programme par chargement dans le registre d'instruction de l'adresse donnée dans l'opérande
BRG	000	0	A condition que le contenu de l'accumulateur soit plus grand que zéro, interrompt le déroulement séquentiel du programme par chargement dans le registre d'instruction de l'adresse donnée dans l'opérande; dans le cas contraire, passage à l'instruction suivante.

Adresse		
Symbole	Valeur en décimal	Valeur en binaire 16 8 4 2 1
<b>Données</b>		
A	10	0 1 0 1 0
B	11	0 1 0 1 1
C	12	0 1 1 0 0
D	13	0 1 1 0 1
E	14	0 1 1 1 0
F	15	0 1 1 1 1
Z	20	1 0 1 0 0
<b>Instructions</b>		
LB (LeBoulot)	04	0 0 1 0 0
SEQ2	27	1 1 0 1 1

Pour rappel l'ordinateur ne comprend que le binaire, il est donc nécessaire de traduire les données et les instructions en binaire pour qu'elles soient bien comprises.



Classes d'instructions	Fonctions en langage de haut niveau	Langage assembleur		Langage machine	
		Symboles des Code-instruction	+ op�rande	Binaire Code-instruction	+ op�rande
Copie de donn�es	b = a	LOAD	A	010	01010
		STORE	B	011	01011
Op�rations arithm�tiques	c=d + e * f	LOAD	F	010	01111
		MPY	E	101	01110
		ADD	D	100	01101
		STORE	C	011	01100
Branchements	goto s�quence2	BR	SEQ2	001	11011
	if z > 0 goto leBoulot	LOAD	Z	010	10100
		BRG	LB	000	00100

On constate sur le tableau ci-dessus dans la deuxi me ligne qu'il n'est pas n cessaire de Load E et D car les add et multiply le comprennent d j  et l'appliquent   l'accumulateur il n'est donc pas n cessaire de leur donner deux arguments. Le store est n cessaire pour stocker la nouvelle valeur.

Les 3 tableaux ci-dessus constituent respectivement les actions "Copie de A vers B : b=a", "Op ration arithm tique : c = d + e x f" et "Branchement et boucle : while z > 0 do LeBoulot".

Pour le branchement BRG, la rupture est conditionn e par la valeur de la donn e qui a  t  pr alablement charg e depuis l'adresse 20 dans le registre accumulateur. Si le contenu du registre accumulateur, et donc la donn e Z, est >   0, le compteur d'instruction sera charg  avec l'adresse 5, le d but de la s quence LeBoulot ayant pour symbole LB. Dans le cas contraire l'instruction suivante est ex cut e. Pour le branchement BR qui termine la s quence LeBoulot, la rupture est inconditionnelle et on place dans le registre compteur d'instruction l'adresse 27 correspondant   SEQ2.

Les boucles sont un m canisme essentiel de fonctionnement de l'ordinateur.

Pour r sumer, quelle que soit le processeur, une instruction d but par un code instruction, constitu  d'un certain nombre de bits qui apr s d codage, d clenchera une s rie de sous-op rations. Ce d coupage en instructions  l mentaires est fait par le **s quenceur**.

Adresse de l'instruction	Langage assembleur: Symboles		Langage machine : Binaire	
	Code instruction	Adresse op�rande	Code instruction	Adresse op�rande
21	LOAD	A	010	01010
22	STORE	B	011	01011

Adresse de l'instruction	Langage assembleur: Symboles		Langage machine : Binaire	
	Code instruction	Adresse op�rande	Code instruction	Adresse op�rande
23	LOAD	F	010	01111
24	MPY	E	101	01110
25	ADD	D	100	01101
26	STORE	C	011	01100

Adresse de l'instruction	Langage assembleur : Symboles		Langage machine : Binaire	
	Code instruction	Adresse op�rande	Code instruction	Adresse op�rande
04 LB(LeBoulot)	**	**	**	**
05	**	**	**	**
06	**	**	**	**
07	**	**	**	**
08	BR	SEQ2	001	11011
.....				
27 (SEQ2)	LOAD	Z	010	10100
28	BRG	LB	000	00100
29	**	**	**	**

Toute l'informatique repose sur le **principe de l'abstraction fonctionnelle**, en effet un ordinateur fonctionne   diff rents niveaux d'abstraction. On peut travailler   un niveau sup rieur sans se soucier du niveau inf rieur.

electronique → logique → assembleur → langage de programmation

Ainsi au fil des années on a augmenté dans les niveaux d'abstraction sans plus se préoccuper des niveaux en-dessous, ce qui permet au programmeur de prendre de la distance par rapport au processeur et de pratiquer une programmation plus proche de la manière de poser et de résoudre les problèmes.

## 2.3 Éléments fondamentaux de l'architecture

### Les Registres

Les registres ont déjà été définis plus haut. Ils interviennent dans les instructions pour les données ou les adresses des opérantes et peuvent être chargés, sommés, permutés, translatés. Le transfert entre registres conditionne la vitesse du CPU.

### Les Mémoires

Comme dit précédemment avec un MAR de 32 bits on peut aller jusque  $2^{32}$  (= 4 GigaBytes de mémoires principales) espace mémoire. Actuellement le MAR possède 64 bits.

LE MDR = 1 Byte et donc il faut accéder plusieurs bytes successivement, ou 2 ou 4 bytes.

### Les bus

Ils peuvent être locaux et connecter des registres entre eux. Plus il y en a, plus d'information pourront être transmises simultanément, plus le CPU ira vite.

Ils peuvent également connecter le CPU à la mémoire et le CPU aux périphériques.

En général, les bus sont parallèles au sein du CPU et séries pour connecter des périphériques plus distants.

Un bus contient un ensemble de lignes avec données, adresses ou information de contrôle.

Ils seront approfondis plus tard.

## 2.4 CISC vs RISC

Soit la problématique suivante : on dispose d'un processeur x32 (32 bits). Ces 32 bits sont partagés entre le code et l'opérande et dont leur longueur et la place qu'ils occupent dépendent du nombre d'instructions. Imaginons qu'il puisse exécuter 256 exécutions (8 bits). Et supposons que si besoin de deux opérands et que chacune tient sur 32 bits il faudra donc au total  $8 + 2 \times 32$  bits soit 72 bits or on en dispose que de 32! (car processeur x32). On dit qu'il y a **problème d'adressage**. Cette problématique a donné naissance au débat entre les processeurs **RISC** et **CISC**.

Les instructions d'un processeur **CISC (Complex Instruction Set Computer)** seront de longueur variable, nombreuses et diverses, rendant plus complexes la lecture, le décodage et l'exécution des ces instructions par le processeur.

Dans un processeur **RISC (Reduced Instruction Set Computer)**, les instructions sont de taille fixe et de format semblable (32 ou 64 bits pour la plupart des processeurs actuels) avec utilisation surtout de registres du processeur. Les instructions seront en nombre plus réduit et montreront une structure homogène, autorisant la conception de processeurs plus simples et généralement plus efficaces.

Intel a toujours favorisé le type CISC (pas Motorola qui sont les anciens processeurs des macs, maintenant aussi Intel). Mais il y a actuellement un retour des processeurs RISC dans les tablettes, smartphones, GPS.

### Comparaison entre un processeur Cisc et Risc

Cisc	Risc
Motorola MC68000, Intel 486, Pentium (entre les deux)	PowerPC, Alpha(ex-Digital), Sparc (Sun) Représente la tendance actuelle
Une large variété d'instructions, de tailles d'instruction (jusqu'à 30 octets) et de modes d'adressage	Instructions toutes de même taille (32 ou 64 bits) Adressage simplifié, par l'utilisation massive de registres
Plus cher en mémoire	Moins cher en mémoire
Plus compliqué, plus flexible mais plus lent	Plus simple, plus rapide
Programme plus court contenant moins d'instructions, mais exécution d'instruction plus lente, une instruction pouvant avoir besoin de 20 étapes	Utilisation massive des registres (nécessite beaucoup de registres) et des transferts entre registres. Les programmes s'exécutent plus localement par les registres et la « cache »
Difficile pour le pipeline mais recours intensif à la microprogrammation	Plus adéquat pour le pipeline → une instruction par cycle d'horloge
Beaucoup d'accès mémoire dans la majorité des instructions	Accès mémoires plus rares: uniquement par les instructions « load » et « store »

La longueur de instructions, soit le nombre d'octets qu'elles nécessitent, est une préoccupation majeure lors de la conception d'un processeur. Qu'on soit en RISC où il est obligatoire de faire tenir chaque instructions dans la longueur canonique( 4 ou 8 octets) ou dans la version CISC où la longueur des octets est moins contraignante, il faut raccourcir ces instructions qui prennent de la place en mémoire, ce qui influe notamment sur le temps nécessaire pour alimenter les instructions depuis la mémoire vers le processeur, sur leur temps d'exécution et donc sur la vitesse d'exécution des programmes. La limitation de la longueur des instructions ne peut se faire sur le code d'instruction, c'est pourquoi il faut limiter la taille des champs attribués aux adresses des opérandes surtout dans les processeurs RISC, vu le format unique de l'instruction. C'est pourquoi les processeurs de type RISC font un usage quasi exclusif pour leurs opérandes d'adressage par registre ou relatif (ces adressages sont expliqués dans la section suivante). Finalement, dans un processeur RISC, une addition s'effectuera à partir des données contenues dans les registres et préalablement recopiées de la mémoire centrale. En revanche, dans un processeur CISC, cette même addition pourra s'effectuer directement à partir des données contenues dans la mémoire centrale.

On comprend maintenant pourquoi un programme dans sa version traduite en langage machine ou exécutable, ne peut s'exécuter que sur un type de processeur donné. Les processeurs parlent en effet des langages différents et ne se comprennent pas entre eux, bien qu'ils aient le même mode opératoire.

## 2.5 L'adressage

On a vu qu'il y avait des différences fondamentales entre RISC et CISC, notamment que le RISC doit adresser beaucoup plus à partir des registres pour maintenir des adressages courts et donc des petites instructions.

Pour réduire l'espace pris par l'opérande on réduit la taille de l'adresse. On dit qu'il y a **adressage**. En fait, une partie de l'instruction étant déjà réservée pour le code d'instruction, il faut s'efforcer de réduire au maximum la taille des adresses. L'unique manière de limiter cette taille est de recourir à des modes d'adressages subtils permettant de réduire dans l'instruction le nombre d'adresses possibles et donc finalement le nombre de bits nécessaires à cet adressage. Une autre raison qui conduit à réduire la taille occupée par les adresses est le principe de localité, que nous retrouverons plus tard. En général, un programme, en s'exécutant, n'utilise au fur et à mesure que des instructions et données restant proches dans la mémoire.

Il existe différents types d'adressage, dont certains peuvent se combiner.

### 2.5.1 Adresse absolue

C'est l'adresse mémoire effective. nécessite 32 ou 64 bits, apportant dès lors un effet grossissant sur les instructions, effet croissant encore avec la multiplication des opérandes.

### 2.5.2 Adresse registre

Si on considère que certaines instructions ne porteront que sur des données contenues dans des registres, la taille de l'adresse peut se réduire considérablement au numéro du registre, comme il y a peu de registres. Ce mode d'adressage est très utilisé dans les processeurs RISC pour sa réduction des tailles des instructions.

### 2.5.3 Adresse relative

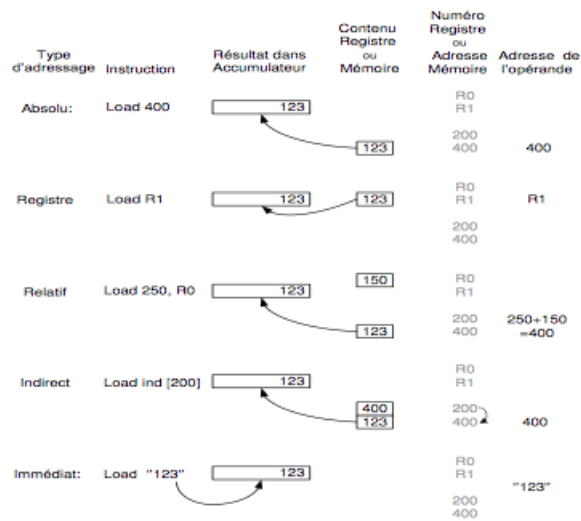
L'adresse indiquée est un "offset" qu'on additionne à une base chargée au préalable dans un registre afin d'obtenir l'adresse effective (adresse absolue). Une adresse relative est automatiquement de taille plus petite qu'une adresse absolue, de plus les données manipulées par un programme sont souvent de type matriciel ou vectoriel, toutes les données se suivant donc, ce qui rend l'usage de l'adressage relatif très utile.

### 2.5.4 Adresse indirecte

L'adresse renvoie soit vers une cellule de la mémoire centrale, soit vers un registre où se trouve l'adresse effective de l'opérande que l'on souhaite traiter. Ce mécanisme peut être répété. Ce principe sert plus à manipuler des adresses qu'à diminuer leur taille. Un exemple d'utilisation intéressante serait pour un programme qui doit effectuer une série d'opérations sur une donnée dont on ignore l'adresse finale ou dont l'adresse peut varier au cours de l'exécution du programme. Peut être combiné avec l'adressage relatif pour faciliter la localisation et le traitement de données.

### 2.5.5 Valeur immédiate

Le champ opérande contient une valeur qui est utilisée immédiatement telle quelle, par exemple une valeur numérique. Exemple : ADD Rn 5 qui se traduit par augmenter de 5. On voit sur le schéma ci-dessous l'illustration des différents types, tous aboutissant au même résultat : la valeur 123 est chargée dans l'accumulateur. Pour réduire l'espace utilisé par l'adressage des opérandes, il est indispensable d'éviter l'adressage absolu. En général il est préférable d'avoir un mécanisme de limitation de l'accès de la mémoire. C'est le principe de la mémoire protégée. Toute zone mémoire occupée par un programme est une propriété privée et inaccessible par les autres programmes, sauf avec une pièce d'identification délivrée par le système d'exploitation. Lorsque plusieurs programmes s'exécutent en même temps (multitâche), chaque programme se voit attribuer sa zone mémoire propre. Finalement, on notera que le code de l'instruction aura également pour fonction d'informer le processeur sur le type d'adressage à utiliser.



## 2.6 Fonctionnement du processeur

Tout le fonctionnement de l'ordinateur se ramène à des **transferts de registres** et à chacun de ces registres correspond une utilisation spécifique. C'est cette vitesse de transfert que l'on caractérise avec les Ghz indiquant la vitesse du processeur. En effet le déroulement d'une instruction élémentaire correspond à plusieurs étapes atomiques. Donc si le processeur est qualifié de 1 GHz il pourra effectuer 1 Milliard de ces étapes atomiques par seconde.

Le **séquenceur** est un peu le comme le poste central de commande qui aiguille les échanges entre registres, à l'image d'un chef d'orchestre. On l'approfondira dans la section suivante.

Le tableau ci-dessous reprend les principaux registres.

Registre	Fonction
IC	Instruction Counter : contient l'adresse de la prochaine instruction à exécuter
IR	Instruction Register : contient l'instruction à exécuter (code-instruction et opérande)
ACC	Accumulator : Registre spécialisé pour le calcul sur des données alimentées depuis les registres ou la mémoire
R <sub>n</sub>	General Purpose Register : 4 registres de travail banalisés de 1 octet, source ou destination de données en mémoire ou dans l'accumulateur
STAT	Status : Reflète (entre autres) l'égalité à zéro du nombre contenu dans ACC, et est examiné par les instructions de branchement conditionnel
MA	Memory Address : reçoit l'adresse de l'emplacement en mémoire où un contenu sera lu ou écrit ; MA est complété par un signal, pour commander la lecture ou l'écriture en mémoire, et par un signal (TERM) indiquant que la mémoire a terminé la lecture ou l'écriture
MD	Memory Data reçoit l'octet lu ou à écrire en mémoire à l'adresse indiquée dans MA.

On a déjà parlé du compteur d'instructions, le registre d'instruction et l'accumulateur mais rappelons brièvement leurs fonctions

### 2.6.1 Compteur d'instructions CI

Il contient l'**adresse** de l'instruction à exécuter, et donc l'adresse mémoire d'où le processeur peut recopier cette instruction. Il est incrémenté pour passer à l'instruction suivante.

### 2.6.2 Registre d'instructions RI

Contient l'**instruction** à exécuter, telle que le processeur l'a lue en mémoire à l'adresse donnée par le compteur d'instruction. Une fois l'instruction stockée dans ce registre, c'est l'emplacement à partir duquel son décodage pourra être entrepris par le séquenceur.

### 2.6.3 Accumulateur ACC

Effectue les opérations arithmétiques et logiques sur les données.

### 2.6.4 Registre d'état STAT

Donne l'état du processeur au terme de l'exécution de l'instruction. Par exemple il peut donner le résultat de l'accumulateur pour une instruction de branchement conditionnel qui teste le résultat d'une instruction précédente afin de décider quoi faire par la suite.

### 2.6.5 Registres de travail

Numérotés  $R_0, \dots, R_3$  sur le schéma, ils sont présents dans tous les processeurs mais en différents nombres, banalisés ou spécialisés. Ce sont eux qui différencient pour une large part les processeurs.

### 2.6.6 Registre d'adresse et registre de données

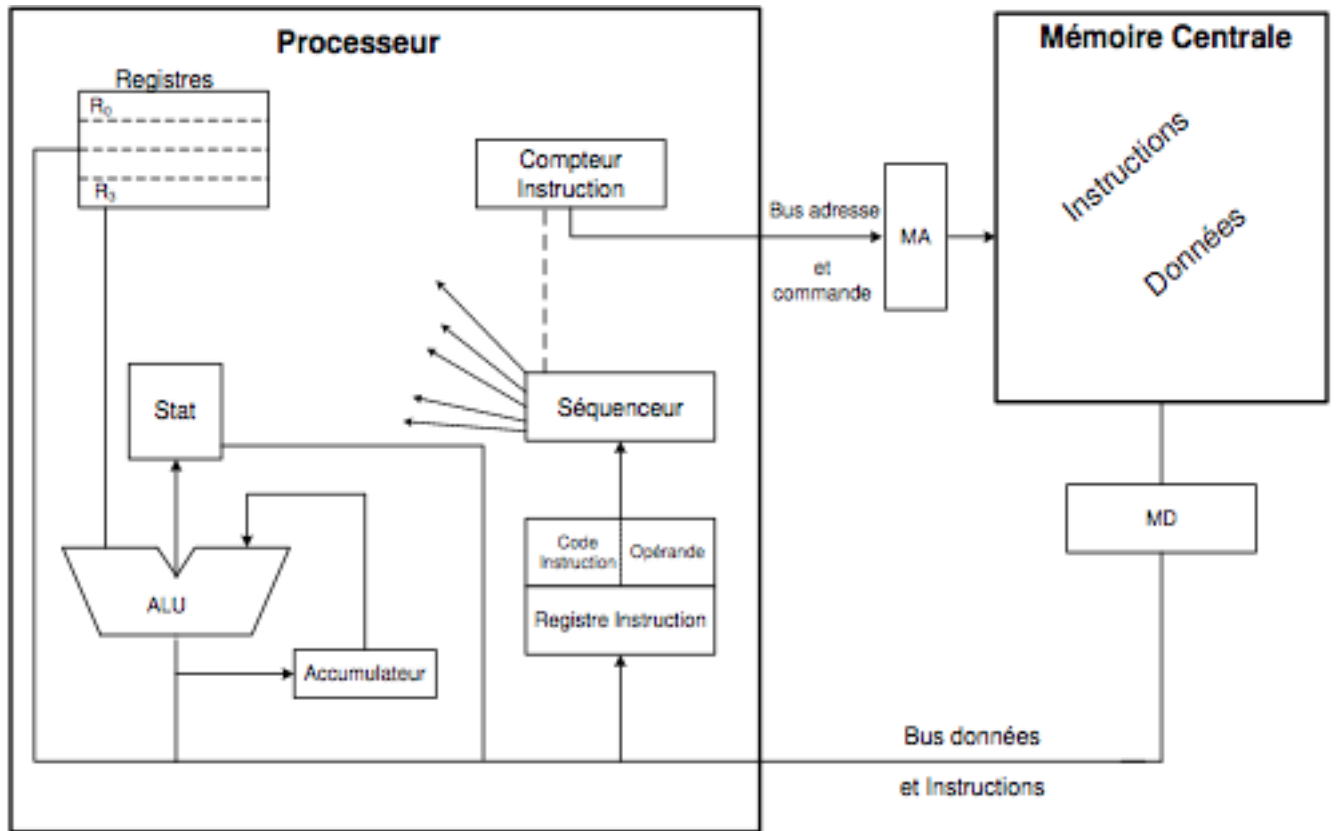
Par opposition avec les registres vus ci-dessus qui sont dits programmables car leur contenu est accessible au programme, le processeur contient d'autres registres intermédiaires qui ne sont pas directement visibles par le programme. Les registres d'adresse et de données gèrent les transferts entre le processeur et la mémoire centrale.

Le **registre d'adresse RA** contient l'adresse du transfert mémoire et des commandes.

NB : Comme on a dit précédemment, pour la mémoire on ne fait pas de distinction entre une instruction et une donnée, elle ne le voit que comme des bits logés dans une adresse, à lire ou à écrire.

Le **registre de données RD** contient soit la donnée à lire depuis la mémoire soit celle à écrire en mémoire.

2.6.7 Important : Schéma de synthèse bloc du fonctionnement du proces-  
seur



## 2.7 Etapes primitives

Le processeur exécute chaque instruction en la découpant en étapes primitives ou étapes atomiques. La mise en oeuvre d'un organe du processeur revient à un transfert de registres, commandé par le séquenceur et consistant à copier dans un registre le contenu d'un autre registre ou à y écrire le résultat de la combinaison de plusieurs registres. Par soucis de stabilité il est important qu'il y ait un délai suffisant entre les différentes étapes, c'est pourquoi chaque étape doit se dérouler dans un temps prévu et défini par une cadence fixée par l'horloge interne du processeur. En fait le temps d'exécution d'une instruction est la somme des temps nécessaires à l'accomplissement de chacune des étapes. On verra dans la prochaine section comment accélérer le fonctionnement du processeur. Pour illustrer la découpe des instructions en étapes atomiques on va prendre l'exemple figurant sur la prochaine figure : la copie ou chargement **LOAD** du contenu d'un registre dans le registre accumulateur, suivie de l'addition **ADD** du contenu d'un autre registre. La première phase de l'instruction est **la lecture** depuis la mémoire vers le processeur. En vue d'obtenir l'instruction à exécuter, son adresse en CI est copiée dans le registre RA, complétée par un ordre de lecture de la mémoire, CI est incrémenté de 1 pour contenir l'adresse de l'instruction suivante. Puis vient le deuxième cycle de base, où l'octet lu en mémoire est copié dans le registre d'instruction RI dès l'envoi par la mémoire du signal **TERM** annonçant son arrivée dans le registre RD. On obtient donc le chronogramme logique de la figure correspondant à la lecture ou au chargement, qui nécessite deux cycles de base. La seconde phase est celle de **l'exécution** qui est composée d'une seule étape, qu'on prenne le cas de **LOAD** ou **ADD**. Le chronogramme complet est donc en 3 étapes.

Soit:

LOAD	R1
ADD	R2

Première phase: le chargement:

Cycle de base	1	2
Etape	MA←IC	IR←MD

Le chronogramme logique complet de l'instruction **LOAD R1** devient donc le suivant:

Seconde phase: l'exécution:

Cycle de base	1	2	3
Phase	Chargement		Exécution
Etape	MA←IC	IR←MD	ACC←R <sub>1</sub>

Celui de l'instruction **ADD R2** est semblable, à la phase d'exécution près :

Cycle de base	1	2	3
Phase	Chargement		Exécution
Etape	MA←IC	IR←MD	ACC + R <sub>2</sub>

## 2.8 Séquenceur

Les transferts entre les registres s'effectuent par des liaisons multiples parsemées de composants électroniques faisant office d'interrupteurs, de sorte que le signal électrique qui représente un bit n'y soit propagé qu'au bon moment, dans la bonne direction et vers le bon destinataire. Pour chaque type d'instructions, le séquenceur dispose d'une liste d'actions à réaliser selon une division spatiotemporelle. Comme dit plus haut le séquenceur est un peu le chef d'orchestre du processeur. Il existe des séquenceurs **cablés** et des séquenceurs **microprogrammés**. Les premiers recourent aux seuls circuits électroniques. Ce câblage devient rapidement complexe avec l'accroissement du nombre d'instructions et entraîne des difficultés pour toute modification du processeur. On peut le résumer à rapidité, complexité et manque d'adaptabilité. Le séquenceur microprogrammé lui est un séquenceur programmable qui, pour chaque instruction à exécuter, va consulter une table avec les différentes étapes.



## 2.9 Pipeline et parallélisme

Le parallélisme est assez facile à comprendre. On peut paralléliser les instructions élémentaires en multipliant le nombre de processeurs dans une même machine ou en améliorant le fonctionnement du processeur en utilisant le pipeline. C'est le principe du temps vaincu par l'espace, où comme avec les autoroutes à plusieurs bandes, quand on vise à diminuer le trafic on étale. En ramenant le séquentiel au **parallélisme** on peut réduire la durée d'exécution, c'est le **pipeline**. Par opposition avec le déroulement des instructions élémentaires une par une, si ce ne sont pas les mêmes parties et transferts du processeurs qui sont concernés donc on peut superposer des instructions.

Il faut bien se rendre compte que ce parallélisme est rendu impossible si les instructions sont dépendantes les unes des autres. Par exemple on ne pourra pas effectuer en même temps les opérations  $c = a * b$  et  $f = c + d$  car la 2ème opération nécessite le résultat de la 1ère à savoir  $c$ . De même une instruction conditionnelle dans une boucle sera conditionnée par la boucle précédente et ne pourra donc pas s'exécuter en même temps. On remarquera quand même que ces problèmes de dépendances entre instructions et d'instruction de branchement peuvent être résolus soit à la compilation, soit en créant des retards, soit en utilisant plusieurs pipelines ou encore en ré-ordonnant les instructions. Une autre approche pour corriger ces problèmes liés aux branchements conditionnels est de dédoubler les unités de traitement du processeur. On traite dans deux branches séparées les deux possibilités de réponse à la condition. Une fois la réponse connue, les résultats de la séquence qui n'apas été choisie sont éliminés. Cela permet d'utiliser au mieux le pipeline. C'est le séquenceur qui dit que 2 instructions sont à exécuter en même temps. Certains processeurs actuels peuvent aller jusqu'à 40 étages de pipeline. Les processeurs RISC ont plus de facilité à recourir au mécanisme du pipeline vu la simplicité et l'uniformité du codage d'instruction. Les processeurs CISC peuvent se rapprocher de ce type de fonctionnement en décomposant leurs instructions en une suite d'instructions adaptées au fonctionnement du pipeline.

Il existe un autre modèle de Pipeline, le **superscalaire**, où les instructions sont séparées en fonction de sur quoi elles portent (Entier, virgule flottante et branchement) et chacun de ces types d'instructions est aiguillé vers un différent CPU. C'est un système multi-processeurs, actuellement Intel met jusque 8 processeurs sur une même puce On notera finalement le cloud computing ou encore le grid computing qui consistent en des partage de tâches entre plusieurs ordinateurs et finalement le recours aux ordinateurs quantiques pour utiliser la superposition d'états en vue d'accélérer encore plus les processeurs mais encore rien de concret actuellement à ce niveau, juste des espérances...

## 3 Les mémoires

### 3.1 Définitions et rappels

#### 3.1.1 Classifications des mémoires

Les mémoires sont les dispositifs qui permettent de stocker des informations, en quantité importante, et sans autre limite théorique que simplement matérielle. Vu leur petite capacité de stockage et leur spécialisation, les différents registres vus ne sont pas vraiment considérés comme des mémoire en ce sens. Les mémoires peuvent être classées en niveau selon leur capacité de stockage, leur niveau de résolution de leur adressage, leur temps d'accès et leur débit, ainsi que selon leur fiabilité, leur volatilité, leur encombrement physique ou leur coût.

#### 3.1.2 Mémoires vives, mémoires de masse et mémoire morte

On peut faire une première distinction suivant la nature technique de la mémoire : Il y a d'une part les mémoires qui se remplissent et s'animent au cours de l'exécution des programmes mais dont le contenu s'efface sitôt que l'ordinateur est mis hors-tension, les mémoires **vives**, et d'autres part les mémoires permanentes dont le contenu est préservé entre deux utilisations des programmes ou de la machine, les mémoires de **masse**. Les mémoires vives s'occupent d'alimenter le processeur en instructions et en données, et les mémoires de masse assurent la permanence des données au-delà de l'exécution ponctuelle d'un programme. Un troisième type de mémoire existe, dont l'usage est plus limité et que l'on qualifie souvent de mémoire **morte** (ROM, Read Only Memory). Les mémoires mortes sont des mémoires destinées à stocker une information qui n'est pas censée être modifiée dans le temps. On veille à protéger son contenu en rendant sa modification impossible. Cette mémoire intervient dans des fonctions vitales telles le démarrage de l'ordinateur. Typiquement elle est utilisée pour contenir des éléments du **BIOS** (basic input/output system), le programme unique qui permet à l'ordinateur de démarrer et d'identifier ses périphériques matériels vitaux. La mémoire contenant le séquenceur du processeur est également du type ROM.

NB : Le terme ROM n'impose pas le type de mémoire qui pourrait très bien être DRAM,SRAM ou encore mémoire flash. Il indique juste que son contenu est protégé.

#### 3.1.3 Rappel de deux registres importants

**MAR ou RA (Registre d'adresses)** Contient l'adresse du mot mémoire. Cette adresse est placée sur le bus d'adresses et devient la valeur d'entrée du circuit de sélection qui va à partir de cette entrée sélectionner le mot correspondant

**MDR ou RD (Registre de données)** Il permet l'échange d'informations (contenu d'un mot mémoire) entre la mémoire centrale et le processeur(registre).

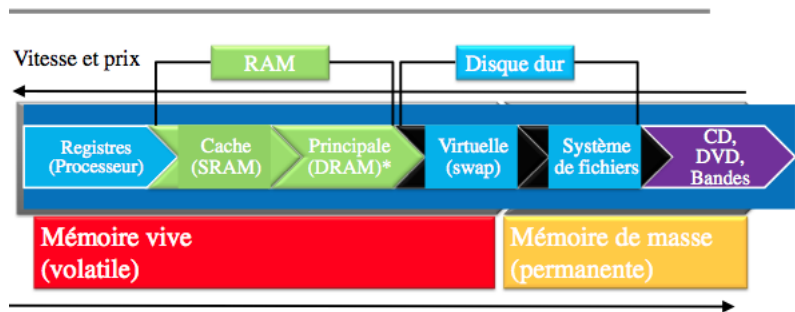
## 3.2 Hiérarchie des mémoires

Il existe plusieurs systèmes de mémoire dans un ordinateur sous différentes formes (magnétique, électronique, RAM, registres,) qui se distinguent par les caractéristiques citées plus haut et notamment par leur vitesse. Par exemple un accès en mémoire magnétique est plus ou moins 1 million de fois plus lent qu'un accès en mémoire RAM. Il existe donc une **hiérarchie** des mémoires.

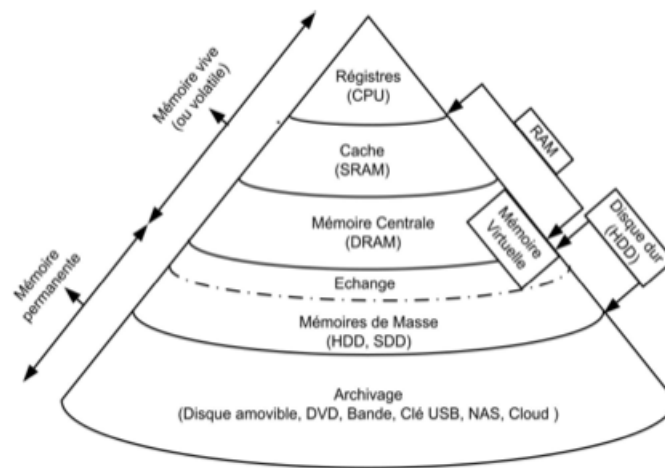
*RAPIDES + registre ← Cache ← Principale ← Disque ← Bande – LENTS*

Les passages d'un niveau à un l'autre sont invisibles à l'utilisateur, mais leurs différences de délai eux se font sentir. Le temps total pour avoir l'information est le temps nécessaire pour avoir l'accès à l'information (temps de latence ou phase de préparation) qui varie considérablement suivant le type de mémoire comme on le verra plus bas, plus le temps dit de "téléchargement" c'est-à-dire la durée de transmission de ces données ou la phase de transfert, fonction du volume de données à transférer et des performances de la mémoire. .

### Vue d'ensemble de la mémoire



\* DRAM et ses dérivées: SDRAM, DDR SDRAM, etc.



On notera sur la pyramide ci-dessus que plus on descend, plus il y a de place disponible dans la mémoire mais moins les accès y sont rapides et donc moins ces mémoires sont coûteuses.

### 3.3 Première description de la mémoire centrale

Les mémoires **RAM**, ou Random Access Memory sont qualifiées de Random par référence à la manière dont on y accède. En effet quelque soit l'emplacement d'un octet (qui peut en effet être aléatoire), son accès se fait de la même manière et prend le même temps. Dans le registre d'instruction mémoire MA on sélectionne une ligne (Cf schéma plus bas) de 8 bits (généralement instructions codées sur 32 ou 64 bits → on prend 4 ou 8 lignes). C'est un mode d'accès fondamentalement différent de l'accès séquentiel. En effet, à part pour la mémoire RAM, les temps d'accès dépendent de la position de l'information..

La mémoire centrale se trouve reliée aux autres éléments de l'ordinateur par des liaisons fonctionnant à des vitesses et des largeurs de bus en croissance incessante, la rapidité de ces liaisons étant un élément influant la performance des systèmes. Mais le débit de la mémoire centrale est loin d'avoir suivi l'évolution des performances des processeurs et il n'est pas en mesure d'alimenter le processeur en données et en instructions à une cadence suffisante pour lui permettre d'atteindre sa pleine performance. C'est pourquoi il y a généralement à proximité du processeur une ou plusieurs mémoires de capacité réduite mais présentant des temps d'accès sérieusement réduits. **Cette réduction est donc obtenue par l'utilisation de mémoire d'un type beaucoup plus rapide et par la proximité de celle-ci ainsi qu'à l'usage exclusif du processeur.** Ce sont les mémoires **caches** introduites dans la partie traitant du processeur, et qui contiennent une copie exacte du contenu d'un très petit nombre de blocs de la mémoire centrale. Ces blocs reprennent les données les plus fréquemment et récemment utilisées en se basant sur le principe de localité mais nous reviendrons plus loin sur le fonctionnement de la mémoire cache. Elles sont toutes deux, ainsi que les registres du processeur, des mémoires **volatiles**. C'est-à-dire qu'elles ne peuvent fonctionner que si elles sont sous alimentation en courant électrique. Une fois le courant coupé, le contenu disparaît. C'est pourquoi l'ordinateur nous invite à sauver notre travail avant de s'éteindre. En pratique cette sauvegarde revient à recopier les données en cours d'édition de la mémoire centrale vers **le support de stockage de masse** (le disque dur généralement).

### 3.4 Mémoires spécialisées, mémoires secondaires et mémoires d'archivage

#### 3.4.1 Mémoires spécialisées

Diverses mémoires spécialisées sont utilisées pour assurer le fonctionnement de certains organes de l'ordinateur. On citera uniquement les tampons ou **buffers** qui sont des mémoires généralement intercalées entre les périphériques et la mémoire centrale et qui permettent d'accélérer notablement certaines opérations d'entrée-sortie en anticipant sur la fin d'une opération d'écriture sur le périphérique ou en mémorisant des données utilisées répétitivement. C'est le même principe d'optimisation que celui utilisé avec la mémoire cache, où on interpose un élément plus rapide agissant de manière transparente, ici avec les périphériques et avec le processeur pour la cache.

#### 3.4.2 Mémoires secondaires ou de masse

Il s'agit là du **deuxième niveau** de mémoire dont le principal représentant est le **disque dur** sur lequel les données sont écrites sous forme de champ magnétique ou d'aimantation, **non volatiles** cette fois. Ces mémoires secondaires sont localisées en dehors de l'unité centrale, et par opposition avec les mémoires centrales, leur contenu est accessible à tout moment dès lors que ces supports sont reliés à l'ordinateur. A nouveau par opposition avec la mémoire centrale, leurs temps d'accès peut dépendre de l'emplacement auquel on souhaite accéder. Plus précisément ce temps d'accès dépendra de la position courante de la tête de lecture et du temps requis pour que celle-ci se rende à l'emplacement désiré (=temps de latence). De plus la taille des cellules transférées (Cf section transfert de niveaux) vers

la mémoire centrale sera beaucoup plus grande, par exemple de 4096 octets! Cette plus grande taille de blocs d'information à transférer s'explique par la lenteur de l'accès à la mémoire secondaire, d'où l'intérêt de limiter les accès et de ramener des gros blocs mais on en discutera plus bas.

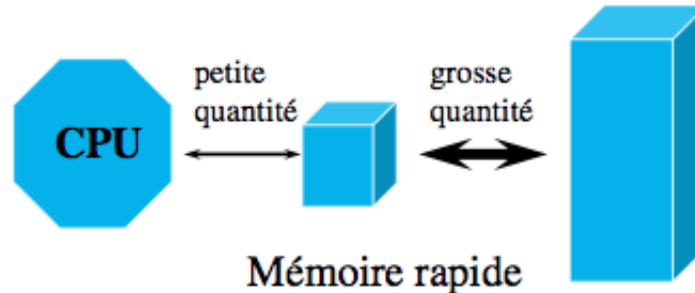
Le contenu de la mémoire centrale étant le seul accessible par le processeur ou par les unités périphériques, toute données indispensables à ces derniers et contenue dans le disque dur se devra de transiter vers la mémoire centrale). Donc les mémoires secondaires sont indispensables au fonctionnement de l'ordinateur car elles ont la responsabilité de stocker tous les programmes susceptibles de s'exécuter à un moment ou à un autre ( y compris les plus vitaux tel le système d'exploitation). Avant de se trouver dans la mémoire centrale et donc en cours d'exécution, ces programmes seront obligatoirement stockés sur le ou les disques durs. C'est ce transfert qui se passe quand on double-clique sur une icône de notre écran.

Le fait que l'accès à ces mémoires secondaires soit **permanent et direct**, au-delà du rôle de **premier hangar de stockage de l'information**, de leur attribuer un second rôle d'**extension de la mémoire centrale**, en repoussant ainsi les contraintes liées à la taille et au prix de celle-ci. C'est le principe de la **mémoire virtuelle** qui sera approfondi plus loin.

### 3.4.3 Mémoires d'archivage

Typiquement les supports dits amovibles tels que les clefs USB ou les disques durs externes, on encore comme dans le passé les bandes magnétiques, les cassettes et disquettes,... Beaucoup d'espace mais temps d'accès très long, l'opposé des registres sur la pyramide de la section 3.2.

## 3.5 Les transferts de niveau



Il existe deux niveaux de mémoire : une rapide transférant à haute fréquence peu de données, l'autre plus lente transférant moins fréquemment beaucoup de données.

Quand PHO va chercher une instruction/opérante, il veut la trouver d'abord dans la mémoire rapide, la chère et petite. S'il ne trouve pas il doit aller chercher dans la grande et plus lente et encore s'il ne trouve pas il va chercher dans le disque dur → De plus en plus lent.

Mais PHO est malin. Il ne va pas "juste" chercher ce qui manque mais aussi un peu tout ce qu'il y a autour. On dit qu'il ramène des **blocs** ou **pages** dans le cas de la mémoire virtuelle. En effet il y a une forte probabilité que la prochaine chose à prendre ne soit pas loin de la première, tout ceci résume le principe de **localité**, qui est à la fois spatial et temporel.

Spatial car on reste dans la même zone mémoire, on ramène des blocs donc il y a une possibilité accrue que les prochaines données dont on aura besoin faisaient parties du bloc et sont donc dans la mémoire rapide. Et temporel car on reste souvent avec les mêmes variables et opérandes dans le temps.

Cela permet donc de ne pas devoir retourner dans la mémoire lente de si tôt.

Comme le modèle New-Yorkais qui attribue à chaque maison un numéro de rue et un numéro dans cette rue, on peut **décomposer l'adresse** en donnant à chaque partie des fonctions différentes, par

exemple 16 bits divisés en 8 bits consacrés à l'adresse des octets et 8 bits pour les blocs. On regroupe donc les octets par bloc et une adresse se compose dès lors du numéro de bloc et de l'adresse de la donnée dans le bloc, ce qui permet de déterminer plus efficacement dans quel bloc une donnée se trouve.

[illegible]

A Partir du numero du bloc on peut dire si l'adresse concerne une information contenue dans le premier niveau de mémoire ou dans le deuxième.

On dit qu'il y a eu un "raté" si on ne trouve pas ce qu'on cherche dans la mémoire rapide. S'entame alors un long voyage pour aller dans la lente. Il faut minimiser ce taux de raté.

Quand PHO transfère les données en les ramenant, si l'espace est inoccupé dans zone rapide on les met et tout est Ok mais souvent l'espace est occupé, la mémoire rapide est dite saturée, et il est donc nécessaire d'écraser des données pour faire de la place pour accueillir le bloc souhaité par un transfert de niveau. Il faut donc effectuer un choix des données à remplacer dans la mémoire rapide. Alors soit il va éliminer ce qui a été modifié le moins fréquemment possible ou il y a le plus longtemps, ou il va sauvegarder ce qui a été modifié et le mettre dans la mémoire lente.

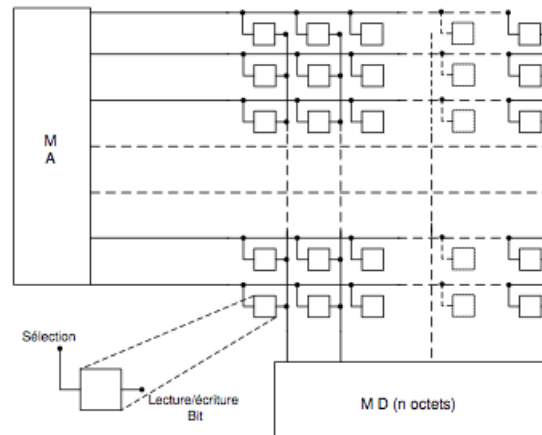
On notera que ce principe de fonctionnement est assez semblable entre la mémoire cache et la RAM et dans le cas de la mémoire virtuelle, entre la mémoire centrale et le disque dur.

### 3.6 Fonctionnement de la mémoire centrale

Nous avons vu plus haut le rôle de la mémoire centrale, qui contient le nécessaire à l'exécution des programmes par le processeur et le nécessaire pour les échanges avec les unités périphériques. La mémoire centrale porte donc bien son nom, elle est le carrefour central de l'ordinateur, un carrefour encombré car y transitent la plupart des échanges.

On voit sur le schéma ci-dessous l'adressage de la mémoire, dont les accès, rappelons-le, constituent un des feins importants de la performance des ordinateurs. Les améliorations portent sur la densité des composants afin d'offrir un volume de mémoire plus important, ainsi que sur les mécanismes d'accès, pour réduire le nombre de cycles nécessaires à une opération de lecture ou d'écriture.

Chaque case du schéma est stockée avec un bistable dans le cas de la mémoire Cache et avec un condensateur sinon. **L'adressage** de la cellule est le rôle du registre d'adresses RA (3.1.3) et les **échanges** celui du registre de données RD. En lisant ce qui a dans la ligne on fait passer l'information au registre MD en "tombant". Toutes les lignes sont donc accessibles à la même vitesse → random car ne dépend pas de la position de l'information, l'accès à une cellule n'est pas conditionné par l'accès précédent, comme expliqué dans la première section sur la mémoire centrale.



### 3.7 Mémorisation : DRAM ou SRAM

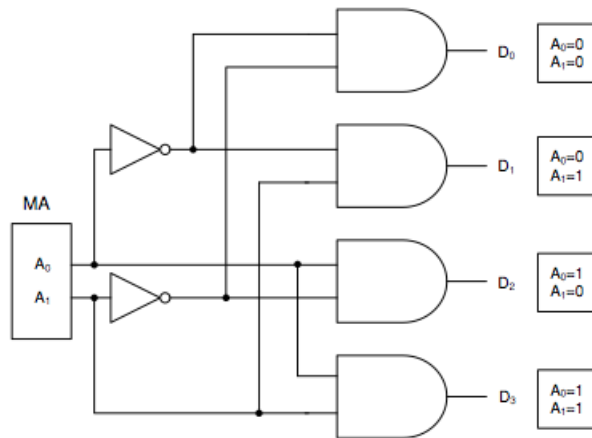
La mémoire RAM est elle-même divisée en deux parties :

- les **DRAM** ou dynamic RAM, sont les plus utilisées et les moins chères car moins élaborées en électronique et car elles permettent une grande capacité. Mais elles sont en revanche plus lentes (temps d'accès plus lents). Chaque bit est matérialisé principalement par un **condensateur** dont l'état de charge traduit sa valeur binaire. Les condensateurs se déchargent progressivement avec le temps et exigent donc des rafraîchissements du contenu des mémoires très fréquents, d'où leur appellation "dynamic" et leur lenteur car cette opération de rafraîchissement doit être exécutée en priorité par rapport aux autres accès et affecte négativement les temps d'accès moyens de la mémoire DRAM.
- les **SRAM** ou static RAM, plus rapides et plus onéreuses, fonctionnent avec bistables qui ne nécessitent pas de rafraîchissement d'où leur appellation static. **Elles sont utilisées pour les mémoires caches.**

Le terme générique DRAM recouvre un ensemble de réalisations plus particulières surtout au niveau de l'optimisation des mécanismes d'accès. Ainsi la **SDRAM** ( synchronous dynamic random access memory) est une mémoire de type DRAM où les accès successifs sont synchronisés avec l'horloge externe du processeur.

Lorsqu'on exécute un programme, une partie est exécutée dans la partie **Cache**, l'autre dans la partie **RAM** (majorité) et le reste dans le **Disque dur** c'est-à-dire en mémoire virtuelle. On fait (quasi) systématiquement un accès disque pour exécuter des programmes.

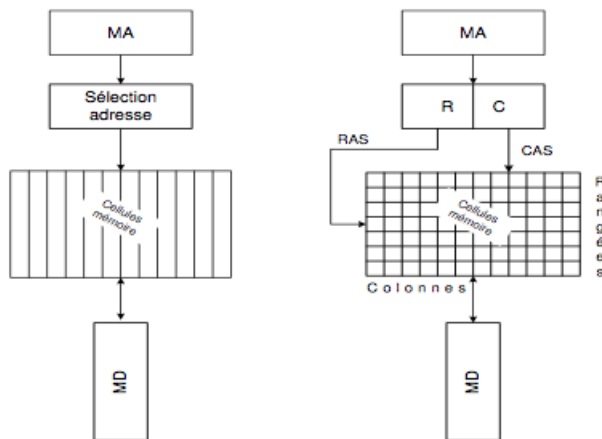
### 3.8 Le décodeur d'adresses



Sur le schéma ci-dessus est décrit le fonctionnement d'un **décodeur d'adresses**. Soit un registre mémoire de 2 bits ( MA a 2 bits). Les triangles sont des inverseurs et les autres des "AND" (cf Chap 1). On a donc que si  $A_0$  et  $A_1$  sont 0, alors  $D_0 = 1$  et  $D_1 = D_2 = D_3 = 0 \rightarrow$  le couple 0;0 donne la ligne 1 et par le même raisonnement le couple 0;1 donne la ligne 2, etc.. Et donc en fonction de  $A_0$  et  $A_1$  donnés on choisit la bonne ligne par ce principe.

NB : Ce décodeur de 2 bits est déjà complexe et la complexité croît exponentiellement avec le nombre de bits !

On peut simplifier ceci par des **mémoires bidimensionnelles en 2 phases**, la première affectée à la sélection de la rangée et la deuxième à celle de la colonne.



Il est important de noter qu'il est plus intéressant de saturer la RAM, c'est-à-dire avoir plus de RAM, qu'acheter un processeur qui va 2x plus vite car une fois qu'il n'y a plus de place en RAM il faut aller chercher ailleurs  $\rightarrow$  beaucoup plus lent.



## 3.9 La mémoire virtuelle

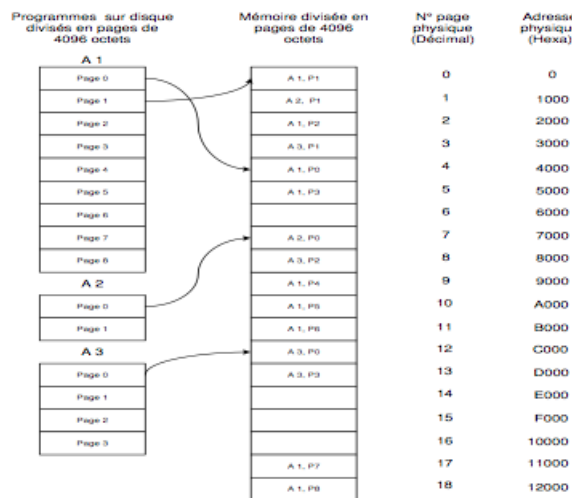
### 3.9.1 Description

Avant de décrire le concept de mémoire virtuelle, citons les raisons qui l'ont initiées, soit les raisons qui font que les programmes ne s'accommodent pas bien d'une mémoire centrale :

- La taille de la mémoire est insuffisante.
- Des parties des programmes sont temporairement superflues.
- Il y a une multiplication des programmes présents simultanément. -Des variations et fractionnement de l'espace mémoire sont disponibles.

Jusqu'à présent on a considéré les octets comme vus dans la mémoire centrale comme une suite linéaire de cellules adjacentes dans un espace à deux dimensions. La réalité est tout autre. Bien sûr, la mémoire centrale ayant les limitations que l'on sait, cet espace idéal ou virtuel ne se traduira pas par un espace contigu de la taille correspondante en mémoire centrale. Un système de gestion de la mémoire, attribuera à chaque programme la partie de mémoire qu'il peut exploiter et s'occupera de la traduction des adresses physiques en adresses réelles( chaque adresse virtuelle est associée à une adresse réelle). Les programmes sont en effet "bluffés", on leur fait croire qu'ils disposent de toute la mémoire souhaitée alors qu'en fait ce qu'ils possèdent est on ne peut plus **virtuel**. On les place en effet au moment de l'exécution à un emplacement quelconque de la mémoire centrale, emplacement choisi et géré par le système d'exploitation. Le système d'adressage interne des programmes n'est en rien modifié, mais il est passé du statut réel à celui de virtuel. Le programme jouera de son espace d'adresses virtuelles comme si celles-ci débutaient à l'adresse zéro de la mémoire centrale et se suivaient en continu. Mais en fait ce programme sera **fragmenté** et dispersé, en partie dans la mémoire centrale et en partie dans la mémoire secondaire. Les fragments s'appellent des "pages".

On voit sur le schéma trois programmes d'application, chacun comprenant un certain nombre de ces pages et se trouvent éclatés dans la mémoire centrale. Cela permet une économie de l'espace avec une plus grande flexibilité lors de l'occupation de celui-ci par des objets de taille très variable.



En fait la gestion de la mémoire centrale est considérablement facilitée en dissociant d'une part l'adressage des cellules tel que le programme le voit( adresses logiques) et d'autre part l'emplacement réel de ces cellules (adresses physiques). La pagination est un des mécanismes permettant de créer cette dissociation.

Le programme sera donc fragmenté en pages de longueur fixe et sera logiquement organisé comme une succession de ces pages. Ce sont ces mêmes pages qui seront physiquement transférées entre la mémoire centrale et un emplacement de la mémoire **secondaire**. Cette mémoire secondaire constituera



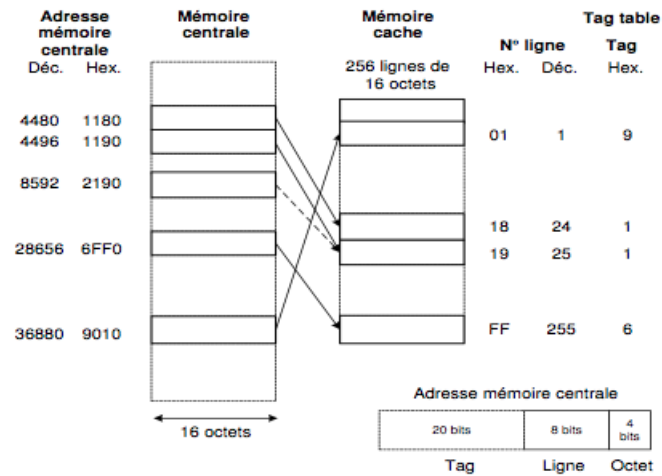
page est réalisée par la MMU ( Memory Management Unit) ou unité de gestion de mémoire, qui opère à partir des informations contenues dans la table des pages. En général il y a un MMU par programme. Si l'adresse logique de la page n'est pas dans la table, c'est un raté. Le CPU doit alors s'interrompre et il faut aller chercher la page sur le disque dur pour la charger dans la mémoire principale. Normalement, la mémoire virtuelle doit être explorée avant la mémoire cache. Cela peut être long et il faut limiter les accès. L'unité de gestion de mémoire se voit donc adjoindre une mémoire de très petite capacité mais ultrarapide et très spécialisée, la TLB (Translation lookaside buffer), une toute petite cache qui contient les pages les plus récentes et les plus utilisées, dans la discontinuité et le désordre. Il s'agit d'une mémoire associative car les pages ne sont plus dans une séquence logique et c'est une mémoire qui doit fonctionner très rapidement. Comme pour la table des pages, chaque entrée du TLB reprend le numéro de page physique en mémoire centrale, mais avec en plus le numéro de page logique. **La grande différence avec la table des pages vue précédemment est qu'ici l'accès ne se fait pas par l'adresse mais par le contenu.** Tous les numéros de page logique sont donc comparés une seule fois avec le numéro recherché. C'est ce qui la rend ultrarapide. dans le cas du TLB, au terme de la recherche par le contenu et exhaustive en un coup, si le numéro de page logique a été trouvé, le numéro de la page physique est généré automatiquement par l'unité de gestion de mémoire. Sinon, on passe à la table des pages en mémoire centrale. Voir la section 3.11.

Table des pages en mémoire		Table des pages en TLB	
Adresse dans la table (n° de page logique)	N° de page physique en Hexadécimal	N° de page logique	N° de page physique en Hexadécimal
0	F	Page 4	A
1	C	Page 0	F
2	4	Page 1	C
3	9	.....	.....
4	A	.....	.....
5	.....	.....	.....
6	.....	.....	.....
7	.....	.....	.....
8	.....	.....	.....

### 3.10 La mémoire cache

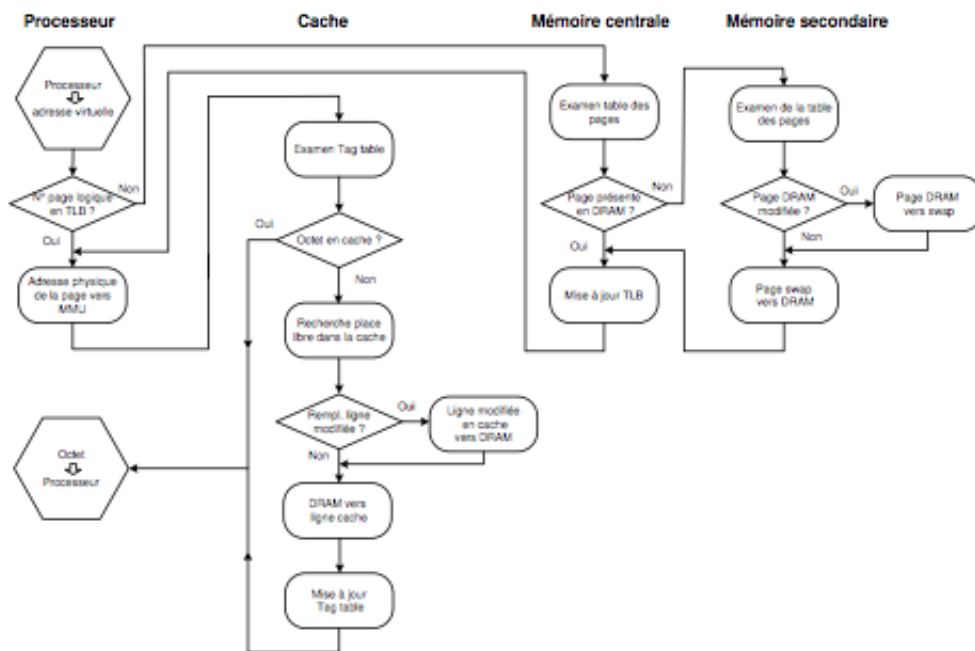
La technologie des mémoires **Cache** fonctionne sur le principe de biportes et est très chère car plus rapide. Il n'est pas non plus nécessaire de les rafraîchir d'où leur prix élevé. Elles sont à portée immédiate du processeur, reprenant une copie des instructions et des données les plus fréquemment ou récemment utilisées.

Sur le schéma ci-dessous on peut observer le fonctionnement de la cache. L'adresse est décomposée en 3 parties : **tag**, **ligne** et **adresse de l'octet dans la ligne**. C'est donc à nouveau un mécanisme de localisation basé sur la scission de l'adresse. Le tag est une notion importante, il donne la position relative de chaque bloc, il permet de retrouver la position d'origine de l'ensemble dont est extrait la ligne dans la mémoire centrale. Il est nécessaire de tenir une correspondance parfaite entre la cache et la RAM, c'est-à-dire savoir pour tout item de la cache d'où il vient dans la RAM. C'est ce qu'effectue le tag. Par exemple pour un tag = 9 correspond l'emplacement 9 dans la mémoire centrale. Il suffit de mémoriser dans la **tag table** pour chaque ligne de cache la valeur du tag afin de déterminer à quelle partie d'origine de la mémoire centrale correspond cette donnée.



### 3.11 Scénario complet de l'accès à la mémoire

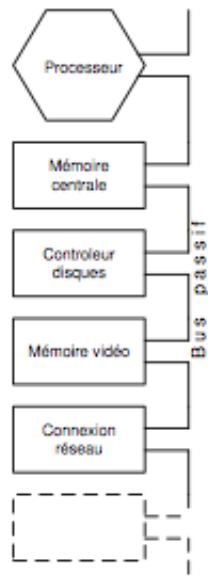
L'existence des différents niveaux de mémoire (cache, secondaire, ...) et des dispositifs qui leur sont associés (tag table, MMU, TLB, ...) rend la gestion de la mémoire très complexe et c'est l'une des tâches les plus exigeantes de l'OS. Le schéma ci-dessous est **important**. Son explication est donnée à la page 128 du livre.



### 3.12 Interconnexions dans l'unité centrale

Au niveau des interconnexions dans l'unité centrale, les éléments sont reliés par des **bus** qui sont des fils électriques. ( Cf schéma)

Il existe différentes technologies, avant on utilisait plutôt un seul bus , dit passif, qui reliait tous les composants, mais comme cela induit des problèmes de vitesse dûs aux composants lents (les autres plus rapides doivent donc s'ajuster aux plus lents...), il y a actuellement aujourd'hui majoritairement des bus multiples, parallèles ou en série. En série c'est typiquement les cables USB, plus facile car pas besoin de synchronisation comme les parallèles mais par contre à vitesse plus lente. Seulement, les nouveaux bus présentent le désavantage de ne plus etre compatible avec certains périphériques dont la technologie n'a pas évoluée.



### 3.13 Jeu de composants

L'alternative à la problématique des bus est celle du **chipset** ou "assemblage de composants". Elle consiste à regrouper les composants de l'ordinateur en des ensembles distincts et spécialisés en fonction de leurs besoins en termes de performance et **d'ensuite** créer des points de jonction. Les composants sont agencés suivant deux types de configuration : rapide dans le NorthBridge et lent dans la partie Sud, qui connecte tous les autres périphériques. La carte mère est le squelette de l'ordinateur, dans laquelle on installe les composants.



## 4 Entrées/Sorties et Périphériques

Sans les périphériques, les effets du CPU ( processeur) sont inaccessibles, on ne sait pas l'utiliser même s'il marche. En effet l'ordinateur agit avec nous via ses périphériques. Il y a des périphériques d'entrée(Input) tels que le clavier, la souris, le micro ou le scanner, ils rentrent de l'information dans l'ordinateur. Et il y a les périphériques de sortie (Output), par exemple l'écran, les haut-parleurs, l'imprimante ou le disque dur. Il faut organiser les interactions du CPU avec ses périphériques. Pour communiquer les périphériques utilisent le **pilote**, c'est le mode de dialogue avec le système d'exploitation. De plus en plus les pilotes sont déjà intégrés dans l'OS. Les interactions CPU/périphériques se caractérisent sur plusieurs aspects :

- les interactions peuvent être sous le contrôle du CPU ou asynchrones( se produisent indépendamment du déroulement normal du CPU, comme les interruptions, lorsque l'opération quitte sans qu'on s'y attende ).
- Il y a plusieurs périphériques avec lequel le CPU interagit et il faut pouvoir les différencier et organiser éventuellement des communications simultanées.
- Les périphériques fonctionnent avec des débits de données et des contraintes internes extrêmement différent ; chaque périphériques s'accompagne donc de son contrôleur pour s'interfacer au CPU.

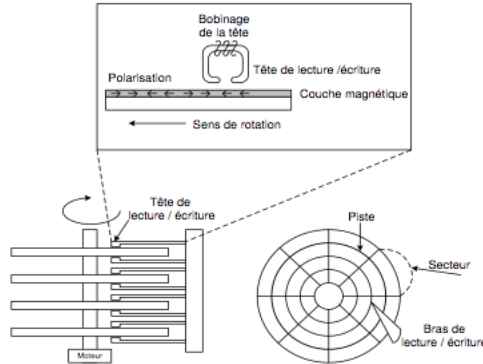
Les périphériques sont de plus en plus intelligents, ils prennent de plus en plus l'initiative dans leur interaction avec le processeur. Il est important que le CPU ne différencie pas les périphériques entre eux, c'est eux-même qui se différencient entre eux par le biais des pilotes et contrôleurs, ce qui simplifie grandement le design du CPU. Concrètement, le pilote et le contrôleur se chargent de gérer les adresses physiques, de synchroniser la communication, de structurer les données, de réaliser les interruptions et de corriger les erreurs de transferts.

C'est les bus qui unifient la connexion aux périphériques, surtout les bus USB. Les périphériques se différencient par leur vitesse, leur capacité,... Il existe des périphériques très rapides par exemple l'écran ou le disque dur et puis des très lents comme la souris ou le clavier.

Périph.	Transfert KB/sec			L E S  P E R I P H E R I Q U E S
		CPU		
Clavier	0.03	I/O registre données	pilote I/O	
Souris	0.02			
Voix	0.02			
Scanner	200			
Imprimante matricielle	0.5			
CD	153	I/O registre adresses	pilote I/O	
DVD	4500			
Disque Dur	150000			

## 4.1 disque dur magnétique

Sauver un fichier se fait en le mettant dans un secteur du disque dur magnétisé. Comme pour les vinyles, il existe un contrôleur qui fait en sorte qu'on puisse lire et écrire dans le disque dur (assemblage de disquettes) avec un bras et une tête ou plutôt une bobine et son noyau. Les faces du plateau sont couvertes d'une mince couche de matériau magnétisable. C'est l'envoi du courant électrique dans la bobine qui se traduit par une magnétisation, fonction du sens du courant dans la bobine et qui correspond à l'écriture d'une valeur binaire 0 ou 1.

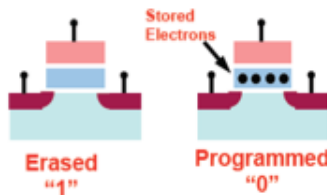


Le compartimentage des surfaces du disque en cylindres, pistes et secteurs est une opération nommée formatage. On écrit sur le disque des informations qui serviront de balises et entre lesquelles les données pourront être écrites et retrouvées.

Les constructeurs de disques durs sont talonnés par les disques électroniques que nous allons voir tout de suite. Les disques durs magnétiques ont une **capacité** qui n'arrête pas d'augmenter, bien plus grandes que les mémoires électroniques. Mais les parties électromécaniques des disques durs les rendent **gourmands en énergie électrique** et de plus ils sont **fragiles**.

## 4.2 disques électroniques (SSD)

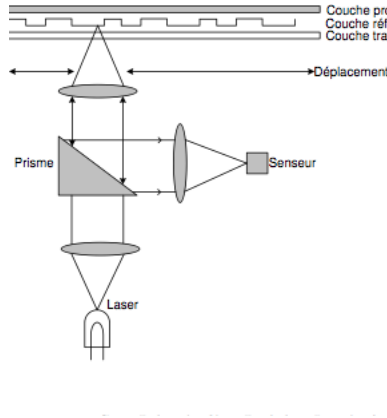
Ils n'ont en fait rien à voir avec un disque quant à leur forme ou leur fonctionnement. Grâce aux transistors, les électrons sont emprisonnés, stockés. c'est ce qui constitue la mémoire électronique. C'est par une tension appliquée sur une grille flottante que les bits se trouvent écrits ou effacés. Mais chaque opération utilise sensiblement la grille, il faut donc bien répartir les opérations sur l'ensemble de la grille, pas comme dans les disques durs magnétiques où on aime stocker l'information dans des endroits concentrés. Elle s'abîme assez rapidement, est plus chère et présente une plus petite capacité de stockage en plus d'être moins robuste que le disque dur magnétique, mais présente une consommation moindre et est plus rapide et plus silencieuse. On recherche donc à faire des solutions hybrides combinant les avantages magnétiques et électroniques. Toutes les deux ne sont pas frappées d'amnésie suite à la coupure de l'alimentation.





### 4.3 disques optiques

De type CD ou DVD, les données sont contenues dans une couche intermédiaire réfléchissante, où elles sont matérialisées par la différence de réflexion du rayon laser entre les surfaces planes et les microcuvettes de la couche du support. Soit la lumière est réfléchi en l'absence de déformations soit elle est plutôt dispersée. Principe simple : 1 quand il y a réflexion et 0 sinon. Il est donc nécessaire de graver les bosses et les réflexions, ce qui rend l'écriture plus difficile.



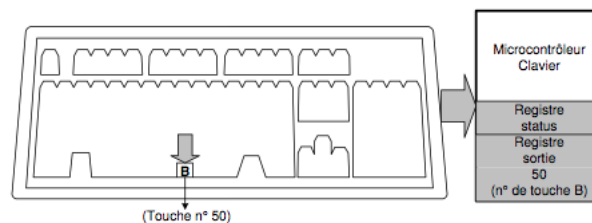
### 4.4 Mémoires électroniques non volatiles

Déjà abordées avec le SSD, ce sont les mémoires de type **flash**, qui sont des mémoires basées sur des circuits électroniques, sans intervention d'éléments mécaniques.

### 4.5 Le clavier à touche

C'est la position de la touche qui importe, pas la lettre au-dessus → on peut aisément transformer un qwerty en azerty.

Une fois la touche enfoncée, le clavier informe par une interruption l'unité centrale de la présence d'un code touche dans son registre de sortie. Un programme dans l'unité centrale prélève le code depuis le registre, puis le transforme en un caractère ou une commande. Enfin, il renvoie un signal en libérant le registre afin qu'un nouveau code touche puisse y être placé.



### 4.6 L'écran

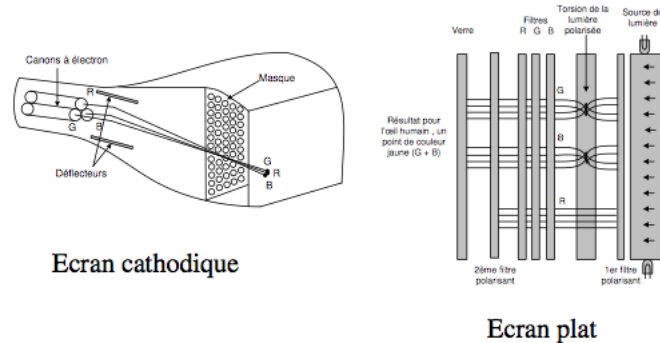
Ecrans cathodiques :

On envoie une matrice avec 3 infos par pixels (R-G-B). Un balayage rapide est constamment effectué

(notre oeil ne peut le voir) avec un canon d'électrons qui acquièrent une énergie cinétique suffisamment important grâce à la profondeur des écrans. S'il y avait une large possibilité de couleur, cela créerait une pression trop forte sur les transferts de données entre le CPU et le moniteur. D'où des choix restreint ( 256 couleurs sur 1 byte). L'image est transférée en binaire, du CPU dans la mémoire interne du moniteur.

Ecrans plats :

Plus complexe, fonctionne grâce à la polarisation de la lumière.



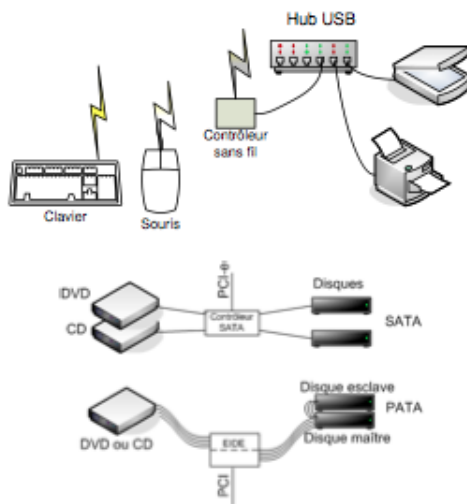
## 4.7 Raccordement des périphériques

Pour réaliser leur fonction d'acquisition ou de restitution des données, les périphériques doivent être raccordés à l'unité centrale de telle sorte qu'ils puissent recevoir des commandes, transférer les données depuis ou vers la mémoire centrale, et annoncer par l'envoi d'une interruption la fin d'une opération ou tout autre changement d'état. Un raccordement = un câblage, une tension électrique, des signaux de commande et d'horloge,... le tout formant l'interface entrées-sorties.

### 4.7.1 Port, Bus et Contrôleur

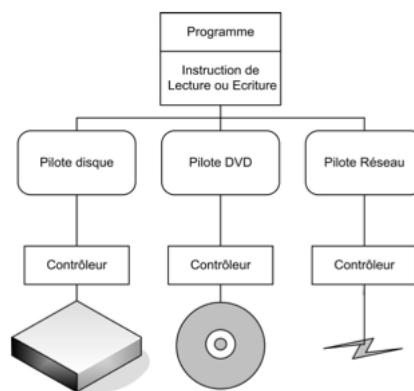
Un port relie un seul périphérique au travers d'une interface parfois spécialisée pour ce type particulier de périphérique. Mais ces ports spécialisés ont tendance à disparaître pour laisser place à des bus USB très populaires.

Le contrôleur assure le dialogue entre son périphérique et son point de raccordement vers l'unité centrale. Il réalise la conversion des commandes en opérations physiques, gère les transferts de données et demande les interruptions de programme. Il est généralement constitué d'un microprocesseur spécialisé.==



## 4.8 Les périphériques en action

Les systèmes informatiques cherchent de plus en plus à réduire l'intervention du processeur lors des échanges de données, c'est-à-dire accroître l'autonomie des périphériques pour que le processeur puissent s'occuper d'autres tâches plus "nobles". Dans cette idée les concepteurs de ces systèmes ont délégué aux périphériques à fort débit la gestion de leurs échanges avec la mémoire centrale, dans laquelle ils peuvent écrire ou lire directement par le biais des DMA (Direct Memory Access). Le programme exprime sa commande en termes généraux, et la transposition en fonction du périphérique est assurée par un logiciel appelé **pilote ou driver** qui lui sera propre à chaque périphérique et permettra de les différencier, tout en maintenant ces différenciations transparentes au programme. Ces pilotes sont indissociables des périphériques !



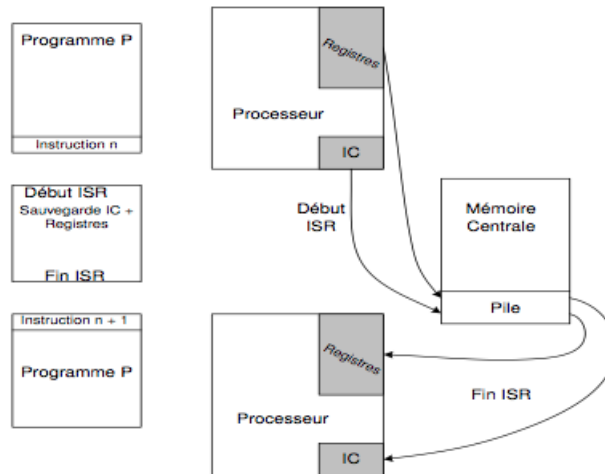
## 4.9 Le mécanisme des interruptions

Les périphériques parlent au CPU grâce aux interruptions. L'OS alloue un canal d'interruption à chaque périphérique, c'est ainsi qu'ils dialoguent.

De manière plus complète :

L'interruption des programmes est la faculté pour le processeur, à l'occasion d'un événement particulier, de quitter le programme en cours d'exécution et de bifurquer, (**sans perdre le contexte du**

**programme interrompu !** ) vers une autre séquence d'instructions. L'interruption est généralement appelée ISR pour interrupt service routine.



Quand on reçoit l'interruption, il faut garantir qu'à la fin de l'interruption on puisse reprendre le programme là où il s'est arrêté. On peut aussi interrompre une interruption. Il faut sauvegarder l'IC et les registres dans la mémoire centrale, dans une mémoire particulière appelée **pile ou stack**. Toute l'information concernant l'état actuel du programme y est sauvee. Ensuite, la première instruction du "service d'interruption" est chargée dans l'IR.

Le périphérique qui demande l'interruption doit s'identifier auprès du CPU, ainsi que la nature du service demandé par le périphérique. Ce service est une routine spécifique qui prendra possession du CPU. Quand la routine est terminée, elle peut soit rendre le contrôle au programme soit modifier complètement le cours des choses (par exemple une interruption d'une imprimante pour dire qu'elle est sans papier). Une interruption d'un événement anormal peut venir de l'extérieur ou être générée par le CPU lui-même. On notera également qu'il peut exister une hiérarchie au sein des interruptions : une plus haute priorité pourra interrompre une plus basse priorité.

Le déroulement du traitement d'une interruption peut se résumer au processus suivant :

- On reçoit un vecteur d'interruption (informant sur sa nature).
- A partir de cette information, on cherche le programme de gestion de cette interruption.
- On **saue** l'état actuel du programme : le registre d'état, le contenu du segment de code et surtout le contenu du PC.
- On charge le PC avec l'adresse du programme de gestion de l'interruption.
- A la suite de ce programme, on restaure tout ce que l'on avait sauve précédemment.

La demande d'interruption d'un périphérique se matérialise par une IRQ (interrupt request). Chaque contrôleur de périphérique reçoit dès sa connexion un numéro d'IRQ. Chaque contrôleur de périphérique se voit donc assigner un canal par lequel il transmet au processeur la demande d'interruption ou IRQ. La demande est envoyée par le contrôleur du périphérique vers un organe annexe du processeur, le contrôleur d'interruptions, qui a une vision globale des demandes d'interruptions, et son rôle est d'arbitrer entre des demandes concurrentes en fonction de leur nature et de leur priorité.

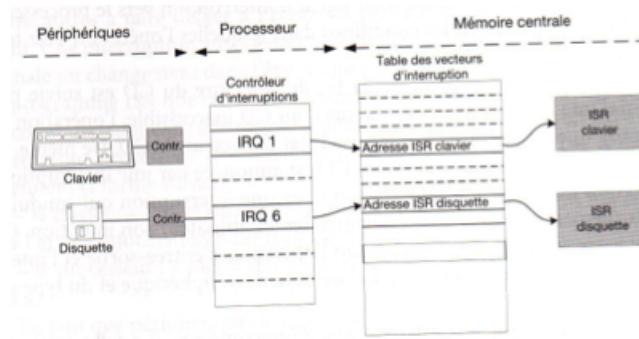


Figure 4.15 Cheminement d'une demande d'interruption

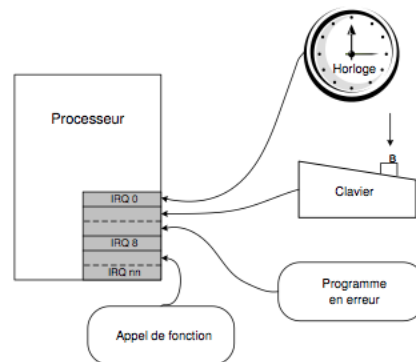
#### 4.9.1 4 Catégories d'interruptions

Un premier type d'interruption est celui déjà vu plus haut où un **périphérique annonce un événement** et déclenche donc un événement.

L'autonomie des périphériques doit être accompagnée d'une surveillance. Si il y a un périphérique défaillant qui oublie de répondre on ne verra jamais la fin de l'opération si bien qu'il faut un **timer**, qui au bout d'un délai fixé provoque une interruption.

Egalement, si le déroulement d'une instruction a causé un **incident dans le processeur**, telle une situation anormale de division par 0, cela pourrait également déclenché une interruption

Finalement, un **programme** peut également emmettre des interruptions, et donc **demande au superviseur l'exécution d'une fonction..**



#### 4.10 Transfert de données et DMA (direct access memory

Afin d'introduire la DMA, analysons ce qui se passe précisément lorsque je clique sur Save dans Word. En cliquant je provoque une interruption Souris, Word reçoit l'info et passe la main à l'OS et lui demande de sauver. L'OS a un pilote qui se connecte avec le disque dur. Un signal est envoyé au disque dur pour lui demander de sauver. **C'est la DMA (Direct Memory Transfer) qui connecte la mémoire au contrôleur de périphériques.** Le dialogue se fait entre eux-deux de manière à ce que le CPU soit court-circuité. Il peut donc s'occuper d'autres process. Donc après avoir inité l'interruption, (car c'est quand même lui qui, à l'aide du pilote du périphérique, constate que l'entrée-sortie peut être réalisée en mode DMA et le communique au contrôleur d'entrées-sorties concerné ) il disparaît

de la circulation et peut faire autre chose (multitasking). Le contrôleur I/O doit connaître l'adresse au niveau du périphérique et de la mémoire, la quantité de données à transférer et s'il s'agit d'une lecture ou écriture. Comme vu au chapitre 1 il est possible de détecter des erreurs de transferts et de les corriger par l'addition de bits de parité.

On notera que quand on installe des périphériques sur notre ordinateur on a besoin de lui donner des **canaux d'interruption et des canaux DMA**

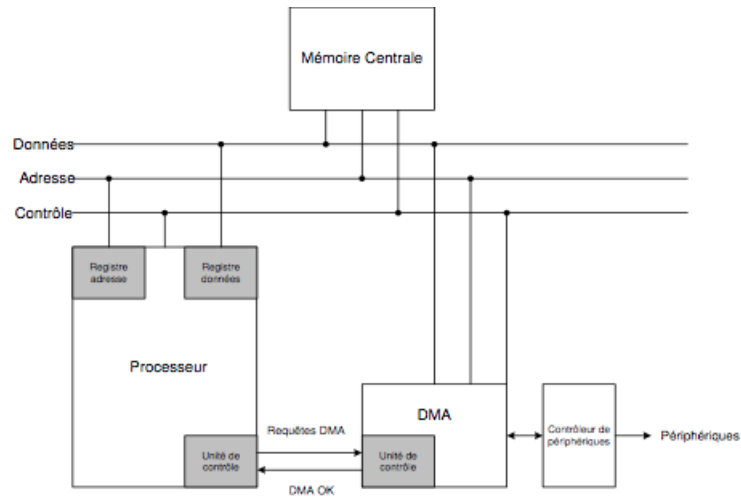
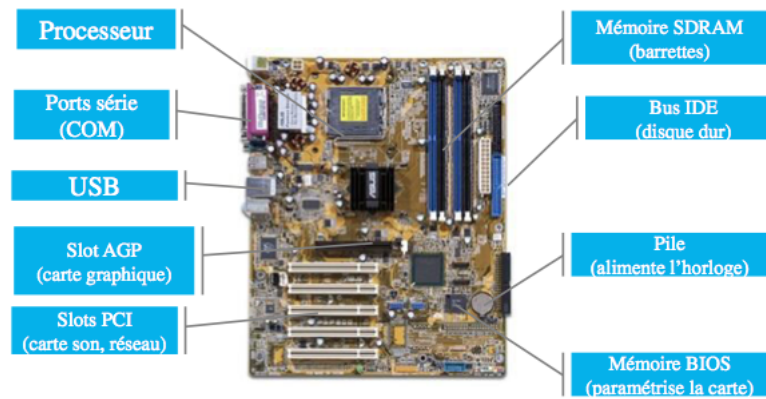
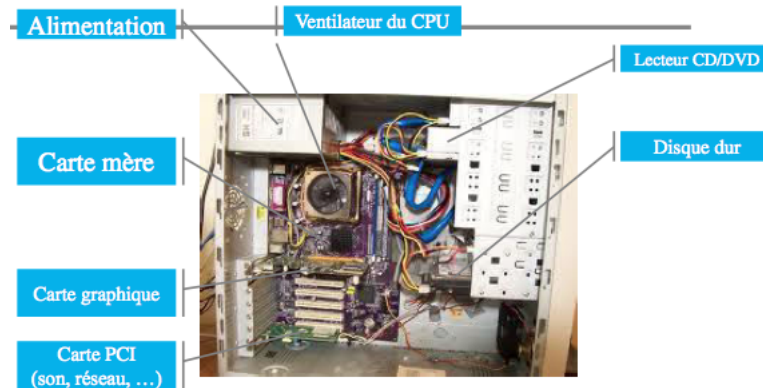


Schéma de la carte-mère :



Les mémoires BIOS sont des mémoires non-effaçables contenant des informations du type quand on allume l'ordi, les choses que doivent se dérouler avant que le processeur ne prenne la main.

### Vue d'ensemble de l'ordinateur



## 5 Les logiciels

### 5.1 Le système d'exploitation (OS) - généralités

#### 5.1.1 Description générale

Un système d'exploitation est un immense logiciel clé, constitué de l'ensemble des services indispensables à la gestion de l'ordinateur et ses périphériques et à l'exécution des applications. L'OS (Operating System) est utilisé tout le temps, il effectue 3 fonctions essentielles :

- **Présenter une interface unifiée** pour les services les plus utilisés
- **Gérer les ressources de l'ordinateur**(processeur, RAM, périphériques,...), c'est-à-dire assurer le partage entre un ensemble d'utilisateurs et organiser la répartition entre les tâches multiples assignées à l'ordinateur. En effet le processeurs se répartit parfois entre plusieurs programmes, et l'unique mémoire entre les différents besoins.
- Assurer le premier point en présentant aux utilisateurs(humains et programmes) **une interface mieux adaptée à leurs besoins** que celle de la machine physique. C'est une interface de "machine virtuelle" qui fournit un ensemble de fonctions pour la gestion et la communication, ainsi que pour la réalisation de logiciel d'application, notamment avec du GUI pour les humains afin de rendre l'interfaçage beaucoup plus facile à utiliser.

De manière synthétique il est possible de regrouper toutes les prérogatives du système d'exploitation en deux niveaux principaux. Le premier niveau, invisible pour les utilisateurs, correspond à la gestion des ressources de l'ordinateur, essentiellement le processeur, les mémoires centrales et secondaires ainsi que les périphériques, assurant leur partage entre les tâches multiples.

Le second niveau est celui par lequel les utilisateurs accèdent à ces services. Services qui leur sont présentés à l'aide d'une interface plus compréhensible et plus facile à manier que celle issue directement des détails physiques de la machine réalisant le service. **On peut donc le résumer en un langage de communication clair et concis entre l'ordinateur et l'utilisateur, permettant une interface plus conviviale entre l'utilisateur et le hardware, plus claire et plus robuste, en plus de faire une gestion des ressources (soumises à plusieurs demandes) et du hardware de manière à optimiser leur utilisation**

NB : Hardware → Matériel et Software → Logiciels

Il existe beaucoup de systèmes d'exploitation, indispensables pour chaque ordinateur. UNIX est l'ancêtre de tous ces OS. Ils fonctionnent tous comme UNIX, certains sont secrets (MAC OS, Windows) et d'autres ouverts (Linux). Linux est à la base du mouvement de l'OPEN SOURCE, qui se résume à "on ouvre et tout le monde collabore".

NB : Libre n'est pas synonyme de gratuit. Des sociétés comme IBM s'enrichissent à partir de logiciels libres. DOS (D pour disc) est un OS que Bill Gates a proposé, faisant essentiellement de la gestion de fichiers.

L'OS est le premier programme qui se charge dans la RAM, on le voit bien sur les Windows avec les écritures précédant les interfaces graphiques. C'est le **Bootstrapping** de l'ordinateur. En français



on utilisera le mot "amorçage" pour désigner la succession d'opérations préliminaires qui aboutissent à la mise en place du système d'exploitation. C'est par cette opération que le processeur vérifie le bon fonctionnement de tous les périphériques, elle s'exécute à partir du BIOS (basic input-output system) qui est donc le premier système d'exploitation à entrer en action lors du démarrage d'un PC. Ainsi, l'OS est responsable de son propre démarrage. Si tout est OK, il va chercher la partie graphique et reste en attente d'événements, dans un état passif dit "event driven".

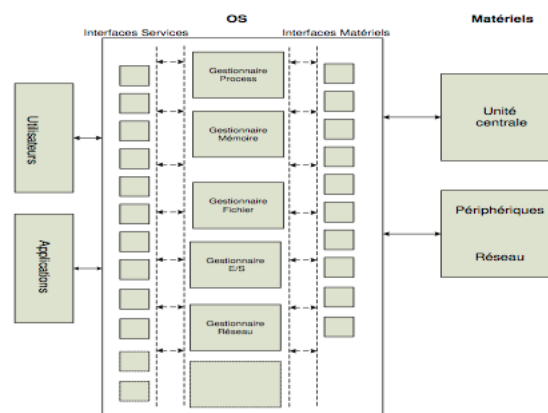
Pour rappel, la mémoire et les périphériques sont de la responsabilité de l'OS, c'est lui qui attribue les canaux d'interruption aux périphériques, c'est lui qui s'occupe des process, de la mémoire, qui gère les I/O, traite les fichiers, ... En pratique, 60-70 % des manipulations de l'OS sont de la gestion de fichiers.

Les OS sont codés dans des langages de programmation à haut niveau (typiquement C/C++), ce qui permet de les compiler et de les exécuter sur de nombreuses plate-formes, les rendant "universels", en plus d'avoir une bonne portabilité des programmes et applications grâce au partage des OS entre plate-formes. D'ailleurs un même ordinateur peut contenir plusieurs plate-formes, on peut en effet avoir Windows et Mac OS sur son ordinateur.

#### En résumé : 10 occupations de l'OS :

- l'interface utilisateur
- la gestion mémoire
- le "dispatching", ou repartition des tâches
- le service fichier
- le service I/O (périphériques)
- la gestion des process et des threads
- la gestion des mémoires disques et autres supports secondaires
- la sécurité et la protection
- réseaux et communications
- support au system manager

La figure ci-dessous montre que l'OS est souvent décomposable en une partie sur la droite, totalement dédiée au matériel et inaccessible tant par les applications que les utilisateurs, et une partie sur la gauche, plutôt tournée vers les utilisateurs et les applications, présentant à ceux-ci une interface plus conviviale et une version plus compréhensible de ce que fait la première. Il s'occupe de faire le pont entre les deux parties.



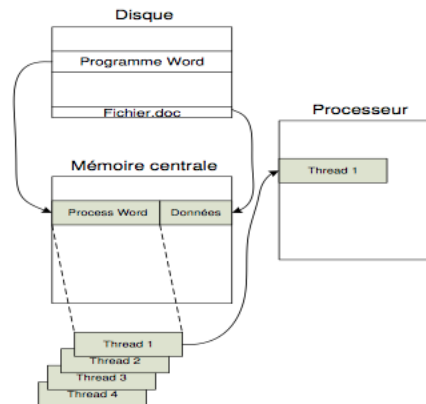
Une séparation importante persiste entre ce qui est appelé le **kernel**, c'est à dire le coeur de l'OS, qui en reprend toutes les fonctionnalités y compris toutes celles directement tournées vers le matériel, et commun à toutes les distributions de Linux, et le **shell**, généralement de type graphique, et qui pourra présenter entre ces distributions commerciales de nombreuses différences.

Dans les entreprises, il existe un administrateur system ou system manager ( Chef de la partie informatique de l'entreprise) → L'OS a "pensé qu'il y aurait une persone privilégiée qui aurait des accès que les autres n'ont pas. En plus de fournir ces informations, l'OS permet des protections et des communications entre utilisateurs et programmes, si bien qu'il permet **l'exécution de plusieurs applications ou programmes avec une impression de simultanéité**. Ce qui nous mène à approfondir le multi-tâches.

## 5.2 Multi-tâches et multi-utilisateurs : Organisation en processus

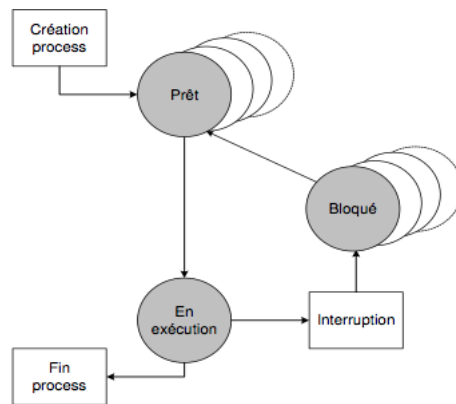
Comme cité précédemment, le dispatching est la répartition equitable du temps CPU entre les programmes.

Un **processus** est un programme qui s'exécute avec ses ressources I/O, les fichiers ouverts, sa mémoire. A chaque processus est associé un **bloc descripteur** qui contient des informations sur le processus telles qu'un identificateur, l'état courant, un espace pour la sauvegarde, l'adresse de sa table, le niveau de priorité, le propriétaire du processus,... Il est important de noter que chaque process est associé à une zone mémoire, la pile, qui permet d'interrompre le process. Car il est nécessaire de stocker les informations sur l'état actuel du process, de sauvegarder ce qui a déjà été fait afin de pouvoir reprendre d'où on était quand le process reprend.



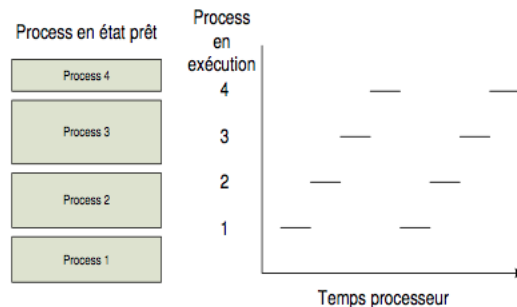
Avant les algorithmes des dispatching fonctionnaient selon le schéma du First-in-First-out, c'est-à-dire un à la fois. Une fois le programme selectionné il s'exécute jusqu'au bout, celui-ci pouvant être choisi sur base de sa taille, sa priorité,... Uniquement Linux procédait au **roundrobin**, où plusieurs processus s'exécutent en "parallèle". Plus précisément il alloue à chaque process un certain "quantum" de temps et tous les processus tournent.

NB : il existe toujours une notion de propriété car on peut décider d'allouer plus de temps CPU à certain qu'à d'autres.

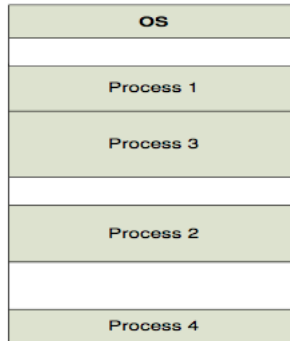


PHO ne peut exécuter qu'un seul process à la fois. Si je déclenche 30 process, ils se mettent dans un état "prêt" et puis PHO en sélectionne 1 et l'exécute pendant x nanosecondes puis un autre, tellement rapide donc qu'on ne voit pas le séquençage d'où l'impression de simultanéité. Si par exemple il est nécessaire d'aller chercher quelque chose dans le disque dur (très lent), au lieu d'attendre qu'il ait été chercher pour l'exécuter, PHO va déjà exécuter d'autres process en attendant, on a donc mis ce premier process en état "bloqué". C'est le dispatcher qui répartit les process en les mettant dans la pile suite à leur déblocage, c'est le multiprocess.

NB : un process est un processus en état prêt.



La gestion mémoire consiste à allouer à chaque programme à exécuter un espace mémoire. La mémoire est partitionnée en plusieurs programmes, de manière équitable ou en prenant en considération la taille du programme. Avec un emplacement mémoire non continu comme on le verra plus bas, avec l'adressage relatif. On veille à protéger l'espace mémoire de l'OS, OS qui gère également la mémoire virtuelle qui, rappelons-le, permet d'accroître considérablement l'espace mémoire en utilisant des adresses logiques en correspondances avec des adresses physiques. On a déjà vu le fonctionnement de la mémoire virtuelle et de la gestion des I/O donc on n'insistera pas plus dessus, si ce n'est qu'on insistera bien sur le fait qu'ils sont la responsabilité de l'OS.



La figure ci dessus montre ce que serait le contenu dans la mémoire dans une version de celle-ci où les processus occuperaient des espaces mémoires continus et de tailles différentes. On y voit l'OS en première place et puis quatre autres processus. On y voit également des espaces mémoires inoccupés, résultant de la distribution au fur et à mesure des espaces pour des processus pouvant être de tailles très différentes. Le recours à la segmentation de la mémoire en pages ( cf mémoire virtuelle) contribue à éviter ce type de gaspillage.

Si un processeur décrète son espace mémoire comme complètement inviolable, un autre processus se plantera s'il essaie d'adresser une information présente dans cet espace. Ainsi, on comprend en quoi l'espace mémoire occupé par le système d'exploitation doit bénéficier d'une protection particulièrement efficace contre les accès intempestifs par d'autres programmes.

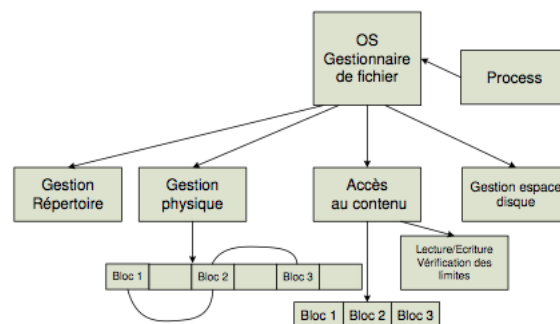
NB ; La gestion de la mémoire virtuelle est également sous la responsabilité de l'OS.

### 5.3 gestion des fichiers

La gestion des fichiers est une partie importante de la gestion de l'OS. En effet tout sur le disque dur est sous forme de fichiers et l'essentiel de nos manipulations et celles de l'OS portent sur les fichiers. Chaque OS a un système de gestion des fichiers qui lui est propre et souvent différente de celle d'un autre. FALT ( ) et NT (New Technology) sont deux exemples manières différentes de les gérer.

un système de dénomination en répertoires est utilisée, avec des répertoires imbriqués dans d'autres répertoires.

NB : Pour Unix et Windows, la gestion du contenu dépend des applications et non pas de l'OS qui voit tout fichier comme une séquence non structurée de bytes.



Le répertoire est essentiellement un mécanisme de dénomination de fichier et forme une structure où chaque nom de fichier est repris, accompagné des informations nécessaires pour le localiser au sein de la mémoire secondaire.

## 5.4 stockages physique des fichiers

### 5.4.1 Partitions et unité d'allocation

Un disque peut être exploité comme un tout ou être divisé en partitions. Chaque partition est considérée par l'OS comme un disque de taille réduite. Les raisons de partitionner un disque sont liées à la contrainte de limite de taille des tables gérant l'utilisation du disque, ou relèvent de la sécurité et de la facilité d'exploitation

Intéressons nous maintenant à la manière d'allouer l'espace sur disque et d'y stocker les fichiers. En général, l'espace sur le disque est divisé logiquement en unités d'allocation, reprises dans la table d'allocation qui décrit leur état en terme d'occupation. La solution la plus élémentaire serait de faire de cette unité la plus petite division physique de l'espace disque, c-à-d le **secteur** et ses 512 ou 4096 octets. Ainsi l'espace alloué à un fichier correspondrait exactement à son besoin en nombre de secteurs. cependant une telle approche nécessiterait la création et la gestion de tables extrêmement volumineuses. Or il est souhaitable de grossir l'unité d'allocation (pour épargner l'occupation et le traitement de ces tables) et de la porter à plusieurs secteurs, formant ainsi des agrégats de secteurs appeles **clusters**. La taille de ces agrégats est déterminée lors de l'opération de formatage qui précède la première utilisation d'un disque.

Il existe plusieurs méthodes de stockages avec leurs inconvénients et avantages propres.

### 5.4.2 Stockage contigu

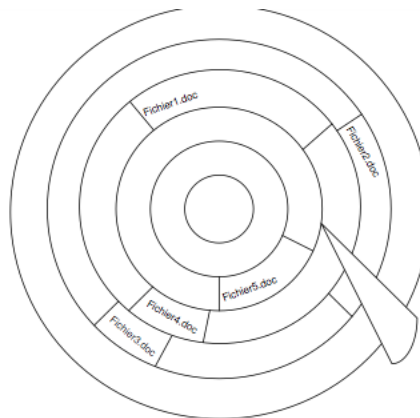
Ce mode de stockage consiste à attribuer aux fichiers un espace contigu égal à leur taille.

-Bonnes performances en matière de temps d'accès, les déplacements du bras restant limités pour un fichier d'un seul tenant.

-Simplicité : la conversion d'une adresse logique en adresse physique se fait par un point de référence.

-chaque suppression laisse un espace qui ne pourra utilement être comblé que par l'allocation à un nouveau fichier dont la taille est la même, le disque se fragmente donc au fur et à mesure que des fichiers se créent sont supprimés ou allongés, et l'OS consacre un temps excessif à réordonner les fichiers.

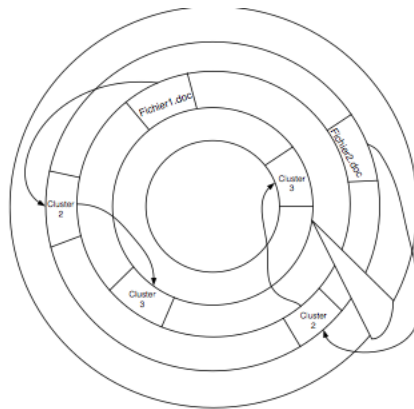
Après de multiples créations/effaçages, il faut des fichiers de mêmes tailles pour récupérer la zone, si plus grand ne rentre pas et si plus petit les zones restantes restent inutilisées et inutilisables et comme ces petites zones s'accumulent on perd au final une relativement grande partie de mémoire. Le **stockage contigu** est donc un gaspillage. Son schéma est représenté ci-dessous.



### 5.4.3 Stockage en liste liée

On préférera alors les fichiers en **liste liée**, où pour récupérer dans l'entièreté le fichier, on récupère un 1er bloc, celui-ci nous renseigne sur la position du 2eme bloc, et ainsi de suite jusqu'à la fin. En termes plus techniques on dit que la liste liée permet, ensuivant les agrégats et les pointeurs qu'ils contiennent, de reconstituer le fichier dans son intégralité. Chaque fichier se trouve donc éclaté sur le disque dur, éclatement qui permet d'éviter tous les problèmes de perte et récupération d'espace disque évoqués juste avant. La défragmentation du disque n'est plus une préoccupation essentielle. Cette méthode était très populaire mais l'est moins maintenant car elle est risquée : en effet si un bloc défaille la chaîne est coupée et on ne sait rien faire du fichier, elle présente donc une certaine fragilité.

De plus une sollicitation mécanique plus importante pour le disque est nécessaire car les distances à parcourir sont plus grandes lors de la lecture du fichier. On notera finalement que cette méthode convient tout aussi bien pour les fichiers séquentiels.



### 5.4.4 Stockage en table indexée

Finalement, la solution actuellement préférée est celle de la **table indexée**, qui présente une table avec les adresses de tous les blocs. C'est donc clairement la meilleure solution pour des accès logiques de type direct ou aléatoire car il suffit de localiser l'index dans la table pour récupérer l'adresse physique. Elle présente néanmoins le défaut de nécessiter un espace mémoire pour stocker la table. La lecture séquentielle se fera également plus lentement car exigeant le parcours de la table entrée par entrée. C'est l'équivalent au niveau du stockage physique de l'accès séquentiel indexé.

