Séquence 4 - Mémoires - Compléments Hors-Programme

I. Mémoires physiques

I. 1. Rappels sur les différents types de mémoires

Comme nous l'avons déjà vu, il existe deux types de mémoires :

les mémoires volatiles (mémoires vives) : ce sont des mémoires qui s'effacent lorsque les circuits électroniques de cette mémoire ne sont plus alimentés (souvent des mémoires à support électronique)

les mémoires non volatiles (mémoires mortes) : ce sont des mémoires pérennes, qui permettent de conserver les données enregistrées même en l'absence de courant électrique (supports papiers, magnétiques, optiques, EEPROM, flash).

Une autre distinction concerne les modalités d'accès à une donnée stockée dans la mémoire. L'accès peut être :

séquentiel : pour lire ou modifier une donnée cible, il faut parcourir toutes les données stockées avant sur le support physique. C'est le cas de tous les supports en bandes magnétiques par exemple (pensez aux K-7 et aux VHS). Dans ce cas, le temps d'accès à une donnée dépend linéairement de sa localisation sur le support de stockage ;

direct : dans ce cas, le support physique permet d'accéder directement à n'importe quelle donnée sans avoir à parcourir une partie de la mémoire. Les mémoires RAM (*Random Access Memory*), qui sont des mémoires électroniques agencées en grille, sont de ce type. Le temps d'accès à une donnée est donc le même quelque soit la donnée ciblée.

On distingue aussi :

les mémoires de masse : mémoires non volatiles destinées à stocker de grandes quantités de données de façon pérenne ; par exemple le disque dur HDD (magnétique) ou SSD (mémoires non-volatiles électroniques de type Flash ou EEPROM) de votre ordinateur ;

les mémoires de travail : mémoires volatiles électroniques utilisées par l'ordinateur pour exécuter des instructions (SRAM, DRAM, SDRAM...etc)

Enfin, certaines mémoires ne sont accessibles qu'en lecture, on les appelle des **ROM** *Read-Only Memory*. Ces mémoires étaient par exemple utilisées pour stocker le programme d'amorçage de votre ordinateur, le tout premier programme lancé au démarrage de la machine.

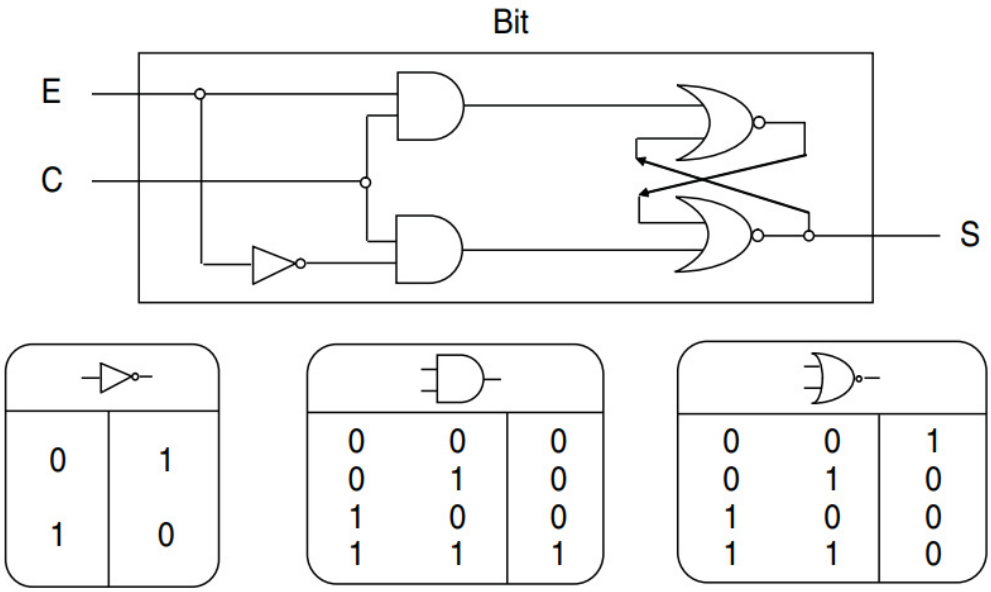
I. 2. Focus sur les différentes mémoires de travail

Les mémoires de travail sont les **mémoires volatiles électroniques utilisées par l'ordinateur pour mener à bien l'exécution** des différentes instructions qui lui sont soumises.

I. 2. a. Implémentation physique des mémoires de travail

Ces mémoires électroniques volatiles de travail peuvent être divisées en deux catégories :

statiques (S) : ces mémoires dites statiques sont physiquement réalisées à l'aide d'un enchaînement de circuits logiques séquentiels à bascule comme celui de la figure ci-dessous, qui permet de stocker, de lire et de mettre à jour 1 bit :



0	1
1	0

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

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

Ces mémoires statiques sont peu énergivores (et chauffent donc moins), très rapides mais coûtent cher !

dynamiques (D) : ces mémoires dites dynamiques sont réalisées à l'aide condensateurs permettant de stocker (ou non) une charge et donc une information. Ils sont plus énergivores car des courants de fuite au niveau du condensateur font perdre la charge au cours du temps, qui doit être réalimentée régulièrement. Ils ont aussi un temps d'accès un peu plus important. Mais leur coût est bien moindre.

Le tableau ci-dessous donne un résumé des points évoqués à ce stade :

Noms	Volatile?	Dynamique?	ROM?	Coût
SRAM	oui	non	non	cher
DRAM	oui	oui	non	moyen
ROM	non	non	oui	faible
UVPROM	non	non	oui, mais peut être reprogrammée par UV	moyen
EEPROM	non	non	oui, mais peut être reprogrammée électroniquement	moyen
Flash	non	non	oui, mais peut être reprogrammée électroniquement	cher
Magnétique	non	non	non	faible

I. 2. b. Différentes mémoires de travail

Les différentes mémoires de travail sont organisées selon une **hiérarchie** qui permet d'optimiser la rapidité des communications et des accès à la mémoire.

Registres : les registres sont de petites unités de mémoire de type statique situées au cœur du microprocesseur, à proximité immédiate de l'Unité Arithmétique et Logique et de l'Unité de Contrôle. Ce sont des petits mémoires électroniques, de l'ordre de quelques dizaines de Ko, coûteuses à fabriquer, mais très rapides (6 à 25 ns de temps d'accès) et très peu énergivores donc créant un minimum de chaleur. Il existe :

des registres internes au processeur, qui ne peuvent être pilotés par un programmeur même à bas niveau ; en d'autres termes, ils ne sont pas exposés par le jeu d'instruction

des registres externes dont l'usage est prévu dans le jeu d'instruction associé au microprocesseur. Parmi ces registres, on distingue des registres d'adresses, des registres de données pour les entiers, pour les flottants, des registres de statut (PSW par exemple)...etc

Caches : ce sont des mémoires intermédiaires de type statique, situées entre la mémoire vive et les registres. Elles servent en quelque sorte d'entrepôts intermédiaires. Elles sont organisées hiérarchiquement sur 3 ou 4 niveaux, le cache de plus petite capacité (L1 [1]) se trouvant à proximité des registres du micro-processeur, le cache de plus grande capacité (L3 ou L4 selon les architectures) se trouvant à proximité de la mémoire vive. Ces mémoires permettent d'éviter des aller-retours systématiques et coûteux entre les registres et la mémoire vive (RAM) en conservant des données récemment demandées par le processeur, ainsi que les données physiquement proches en mémoire des données récemment demandées.

Mémoire principale, aussi improprement appelée RAM : il s'agit d'une mémoire plus grande (aujourd'hui plusieurs Go) de type dynamique qui stocke l'ensemble des instructions et des données de tous les processus en cours d'exécution sur la machine.

Le tableau ci-dessous donne quelques ordres de grandeurs des temps d'accès et des capacité mémoire des différents niveaux de cache sur les ordinateurs personnels actuels.

1. L pour *level*, niveau en anglais

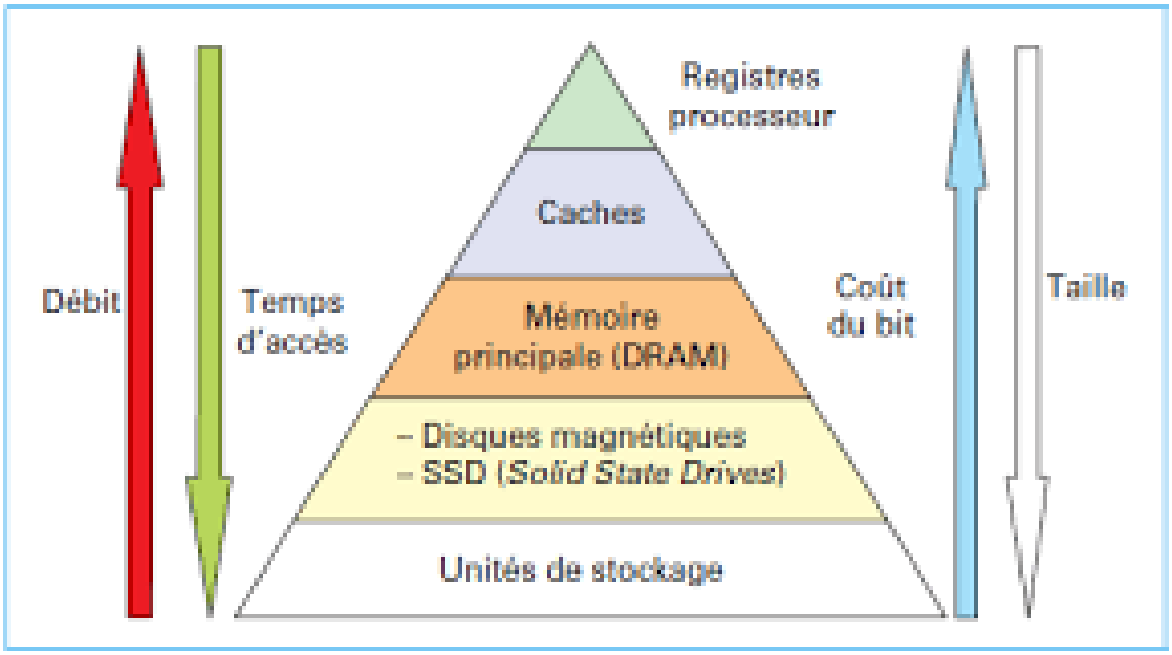
	Technologie	Temps accès mémoire (latence)	Vitesse de transfert (débit)	Capacité
Registres	SRAM (Static RAM) avec des bascules	0.25 ns (proc à 4GHz)	ultra rapide	32 ou 64 bits (taille du bus), certains registres internes ont 128 bits
Cache L1	SRAM (Static RAM) avec des bascules	1-10 ns	$\gg$ 1Go/s	256 Ko
Cache L2	SRAM (Static RAM) avec des bascules	1-10 ns	$\gg$ 1Go/s	1 Mo
Cache L3	SRAM (Static RAM) avec des bascules	1-10 ns	$\gg$1Go/s	8 Mo
Mémoire principale	DDR4 SDRAM (Double Data Rate 4 Synchronous Dynamic RAM) avec des condensateurs	10-20 ns	200 Mo/s à 5 Go/s	Standard 8 ou 16 Go. Jusqu'à 4Go (32 bits). Jusqu'à 128 Go (en 64 bits).
Mémoire de masse	magnétique HDD (Hard Disk Drive)	3-15 ms	100 Mo/s – 1 Go/s	Jusqu'à plusieurs To
Mémoire de masse	électronique EEPROM ou Flash SSD (Solid-State Drive)	25-100 μs en lecture 250 μs en écriture	100 Mo/s – 5 Go/s	Jusqu'à plusieurs To

I. 3. Les mémoires caches

I. 3. a. Organisation hiérarchique des mémoires de travail

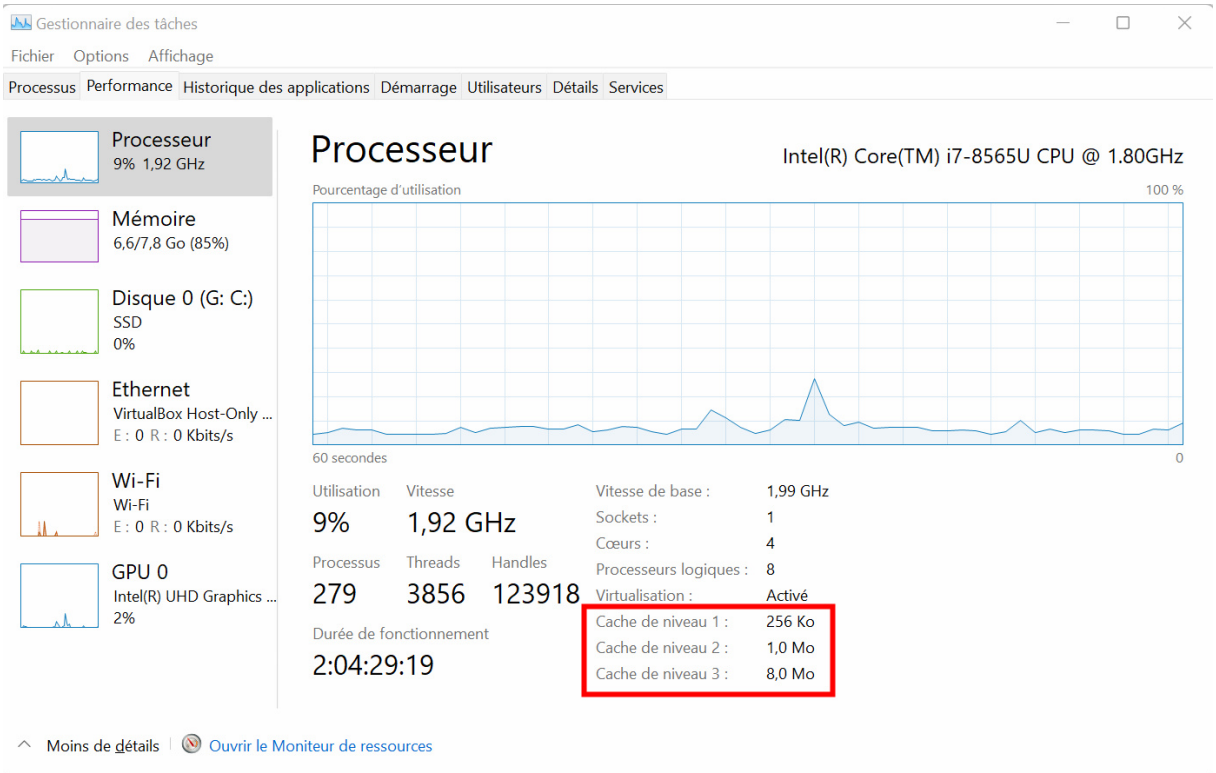
Les mémoires caches sont agencées hiérarchiquement du cache L1, le plus rapide en temps d'accès et le plus petit en capacité de stockage, jusqu'au cache L3, moins rapide mais de plus grande capacité. Cette architecture est conçue pour accélérer le va-et-vient des informations entre la mémoire principale et le processeur en proposant des

stockages intermédiaires permettant d'éviter l'accès systématique à la mémoire principal, plus coûteux.



I. 3. b. Connaître son architecture mémoire

Si vous voulez connaître votre configuration matérielle et vos tailles de cache, la figure ci-dessus montre comment obtenir la taille des différentes mémoires caches d'une machine fonctionnant sous le système d'exploitation Windows (clic-droit sur le symbole Windows de la barre des tâches puis sélectionner Gestionnaire des tâches)



Sous Linux, la commande `lscpu` permet d'obtenir toutes les informations sur les tailles de cache :

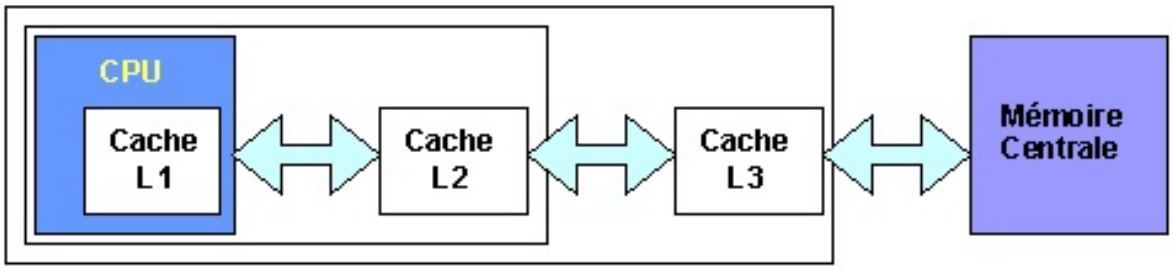
```
golivier@ordiprof:~$ lscpu
Architecture :                          x86_64
Mode(s) opératoire(s) des processeurs : 32-bit, 64-bit
Boutisme :                              Little Endian
Address sizes:                          39 bits physical, 48 bits virtual
Processeur(s) :                         1
Liste de processeur(s) en ligne :       0
Thread(s) par cœur :                    1
Cœur(s) par socket :                    1
Socket(s) :                             1
Nœud(s) NUMA :                          1
Identifiant constructeur :              GenuineIntel
Famille de processeur :                 6
Modèle :                                142
Nom de modèle :                         Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
Révision :                              11
Vitesse du processeur en MHz :          1991.999
BogoMIPS :                              3983.99
Constructeur d'hyperviseur :            KVM
Type de virtualisation :                complet
Cache L1d :                             32 KiB
Cache L1i :                             32 KiB
Cache L2 :                              256 KiB
Cache L3 :                              8 MiB
```

I. 3. c. Circulation et stockage intermédiaire des données

L'algorithme de gestion du cache consiste à y charger toute zone de mémoire demandée par le processeur à la place de la zone la moins récemment utilisée de celles qui étaient déjà là, en spéculant sur le fait que si cette zone est demandée maintenant elle va l'être souvent (des milliers de fois) dans les instants qui suivent (quelques milli-secondes). Cela fonctionne ainsi :

- Le microprocesseur demande une donnée localisée à une certaine adresse mémoire ;
- Le cache L1 vérifie s'il possède cette information :
  - S'il la possède, il la retransmet au microprocesseur et un accès à la mémoire principale a été évité : on parle alors de succès de cache (*cache hit* en anglais).
  - S'il ne la possède pas, il la demande au cache de niveau L2 : on parle alors de **défaut de cache** (*cache miss* en anglais).
    Le cache L2 vérifie s'il possède cette information.
    - S'il possède l'information parmi ces données, il retransmet au cache L1 la donnée et tout le bloc mémoire autour de cette donnée pouvant être hébergé dans le cache L1, plus petit. Le cache L1 stocke ce bloc de données pour utilisation ultérieure au besoin et transmet la donnée souhaitée au microprocesseur
    - S'il ne la possède pas, il la demande au cache L3.
      Le cache L3 vérifie s'il possède cette information.
      - S'il possède l'information parmi ces données, il retransmet au cache L2 la donnée et tout le bloc mémoire autour de cette donnée pouvant être hébergé dans le cache L2, plus petit. Le cache L2 stocke ce bloc de données pour utilisation ultérieure au besoin et transmet un sous-bloc contenant la donnée souhaitée au cache L1 qui lui-même retransmettra la donnée ciblée au microprocesseur
      - S'il ne la possède pas, il la demande directement à la mémoire principale

Les données circulent ainsi de la mémoire principale vers le cache L3, puis le L2, et enfin L1.



Ces algorithmes doivent être implémentés en hardware pour les mémoires caches de bas niveau afin d'être les plus rapides possible et de ne pas ralentir le processeur. Cependant, ils peuvent être implémentés en software pour des caches de niveau supérieur.
La partie plus compliquée de l'algorithme consiste à maintenir la cohérence entre les caches et la mémoire principale par la réécriture des zones modifiées dans le cache. Le lecteur trouvera dans le livre de Hennessy et Patterson [2] toutes informations souhaitables sur ce mécanisme de hiérarchisation de la mémoire.

I. 3. d. Principes de localité

Si les mémoires cache permettent d'accroître les performances, c'est grâce au principe de localité qui a été mis en évidence dans différentes études sur le comportement des programmes informatiques. On a en effet observé qu'à une échelle de temps petite pour l'observateur mais grande par rapport à la vitesse du processeur, disons pendant l'exécution d'un sous-programme moyen, le processeur accède toujours à peu près aux mêmes zones de la mémoire. Donc si on charge ces zones dans le cache on va accélérer les traitements. En fait, deux principes de localité ont pu être mis en évidence :

le principe de localité spatiale qui indique que l'accès à une donnée située à une adresse X va probablement être suivi d'un accès à une zone très proche de X. C'est évidemment vrai dans le cas d'instructions exécutées en séquence, et plus vrai encore pour les boucles courtes.

le principe de localité temporelle qui indique que l'accès à une zone mémoire à un instant donné a de fortes chances de se reproduire dans la suite immédiate du programme. C'est évidemment vrai dans le cas des boucles de quelques instructions seulement.

Ainsi, si le principe de localité spatiale et temporelle est vérifié par le programme exécuté, la plupart des échanges de données se feront entre les premiers niveaux de cache et le processeur. Plus le niveau de cache est élevé, moins les mises à jour de données seront fréquentes, ce qui est une bonne chose car les temps d'accès augmentent avec le niveau des caches. Grâce aux principes de localité, la hiérarchie des cache procure des augmentations de performances très spectaculaires

II. Mémoire physique et mémoire virtuelle

2. John L. Hennessy David A. Patterson. Computer architecture : a quantitative approach. Morgan Kaufman Publishers, San Mateo, Calif., USA, 1996.

II. 1. Adresses physiques

Une mémoire électronique peut être représentée par une grille de bits, chaque bits correspondant physiquement à un circuit logique réalisant la fonction de mémorisation.

Définition 1 (Mot mémoire)

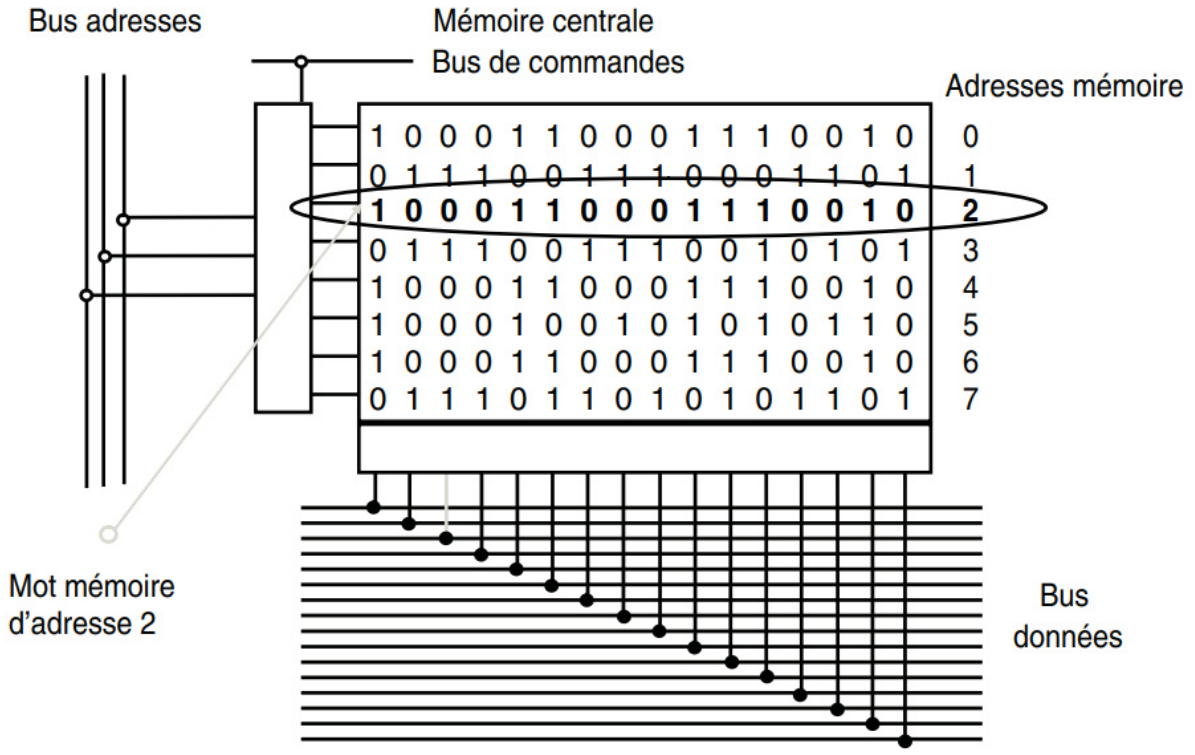
> Les bits sont regroupés en zones de stockage mémoire élémentaires appelés mots : un mot correspond ainsi à la plus petite zone de stockage mémoire pouvant être adressée. En général, la taille d'un mot mémoire est fixée à 8 bits (1 octet).

Pour pouvoir adresser chaque mot, chaque ligne de la grille de bits correspond à un nombre de mots :

— 8 mots de $n = 8bits$ pour les architecture 64 bits (lignes de 64 bits)

— 4 mots de $n = 8bits$ pour les architectures 32 bits (lignes de 32 bits)

Le bus d'adresse (entrée) étant constitué de 64 fils, 2^{64} mots sont adressables en théorie. La façon de définir l'adresse réelle, physique, dépend de l'architecture du microprocesseur : un mot peut par exemple être adressé en donnant son numéro de ligne et son numéro de colonne (de 0 à 7 en 64 bits) (adressage matriciel)
La figure ci-dessous [3] montre une représentation de la mémoire principale en grille, on se limitant à un système d'adressage à 3 bits et donc à un nombre de mots adressables égal à $2^3 = 8$.



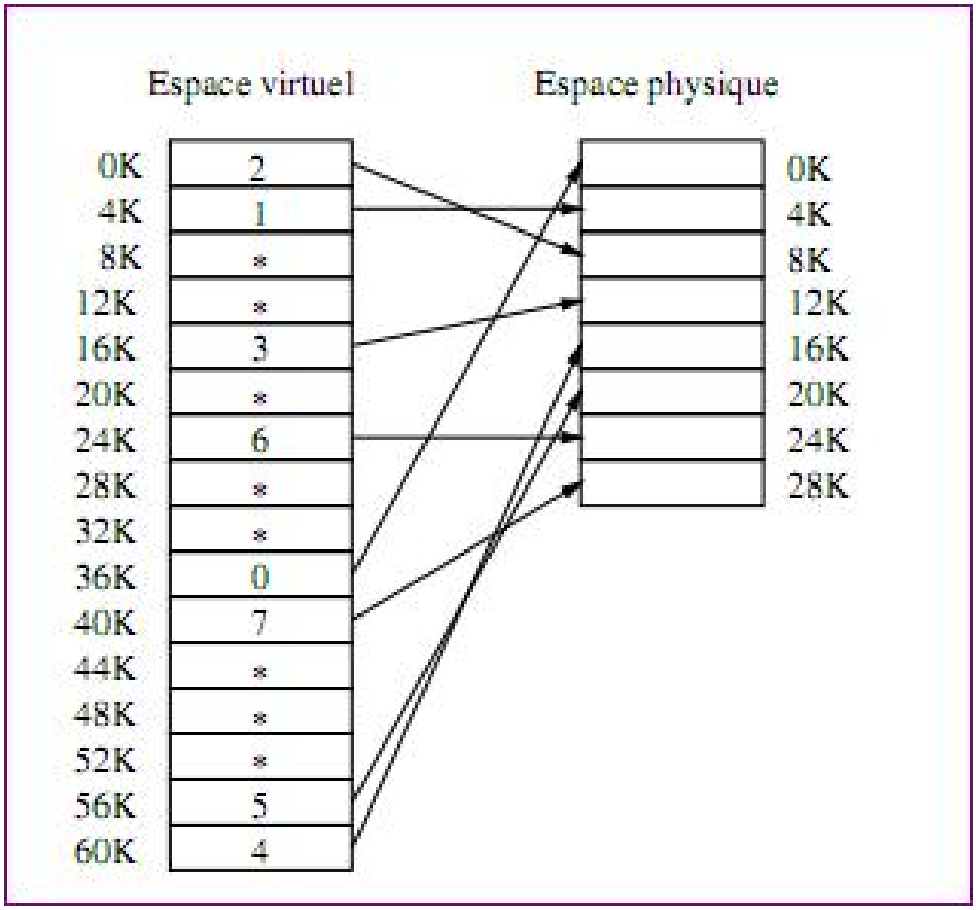
Le sélecteur (carré blanc) est un composant électronique (circuit combinatoire) qui va permettre de sélectionner le mot ciblé par une adresse et de l'envoyer sur un bus de sortie.

II. 2. Adresses virtuelles

En réalité, le microprocesseur ne travaille pas avec des adresse physiques adressant directement la mémoire principale. Il fonctionne avec des adresses virtuelles, qui seront traduites en adresses physiques par l'Unité de Gestion Mémoire (MMU *Memory Management Unit*).

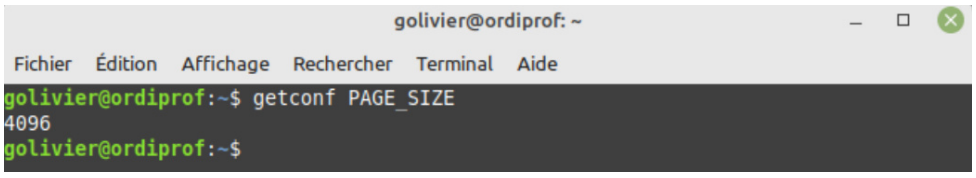
3. extraite de l'excellent livre d'Alain Cazes *Architecture des ordinateurs*

La MMU est l'un des composants du microprocesseur. Elle est elle aussi constituée de transistors et de portes logiques, dont la logique d'agencement permet traduire des adresses virtuelles en adresses physiques.

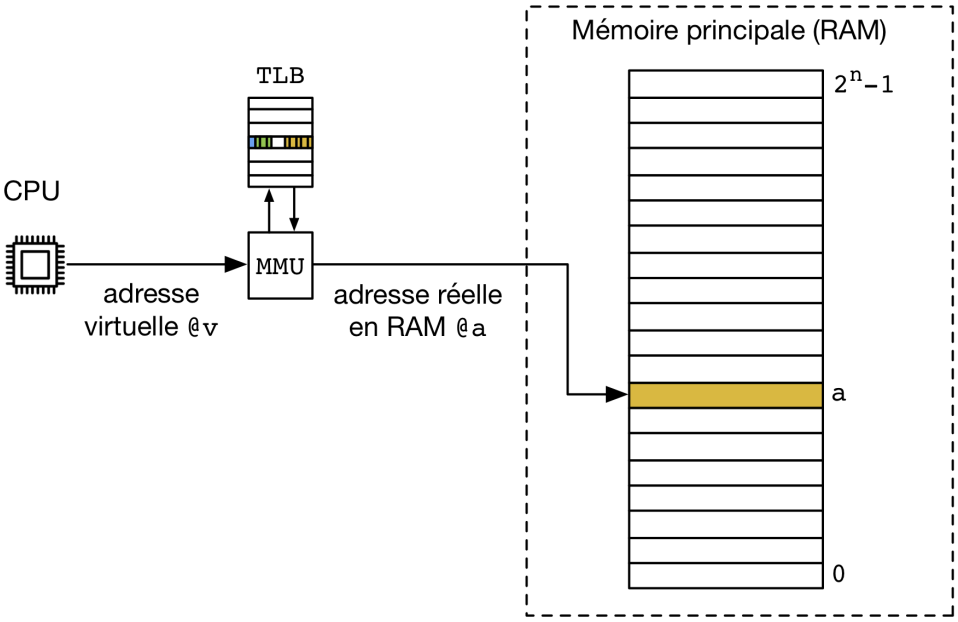


II. 3. Des pages de mémoire virtuelle aux adresses physiques

La mémoire virtuelle est découpée en pages, généralement des pages de 4096 octets (4Ko). On peut obtenir la taille des pages de mémoire virtuelle sous Linux de la manière suivante :



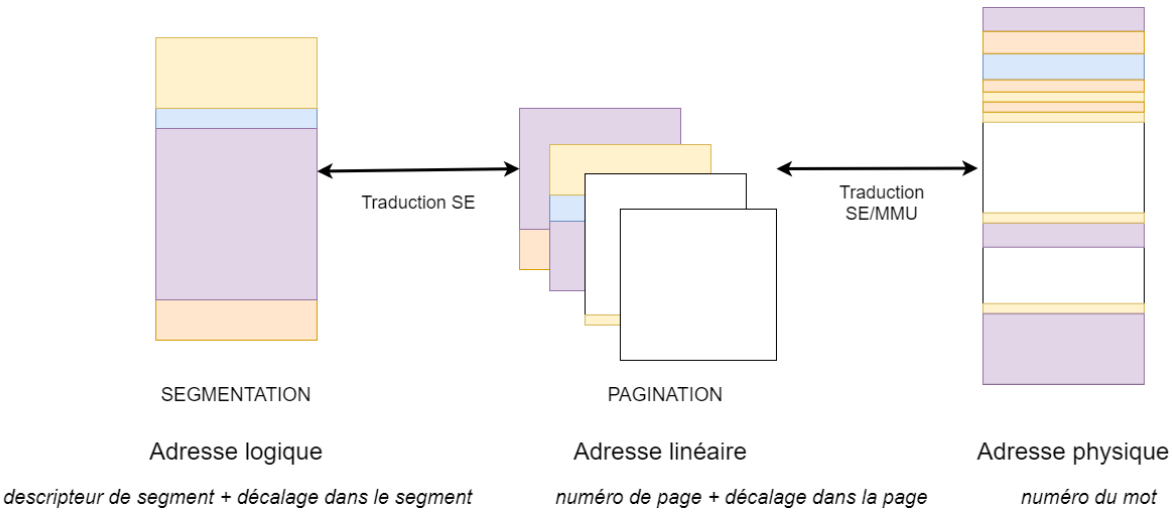
Une page n'est pas forcément associée à un espace mémoire physique existant. Une adresse virtuelle (adresse linéaire) est constituée d'un numéro de page et d'un décalage (*offset*) indiquant la localisation de la donnée relative au début de cette page. La traduction d'une adresse virtuelle en une adresse physique réelle est gérée par le MMU.
La MMU utilise la **table des pages** (PT *Page Table*), stockée en mémoire principale et qui permet de garder en mémoire la correspondance entre les pages virtuelle et leur emplacement physique en mémoire principale, si elle a été définie.



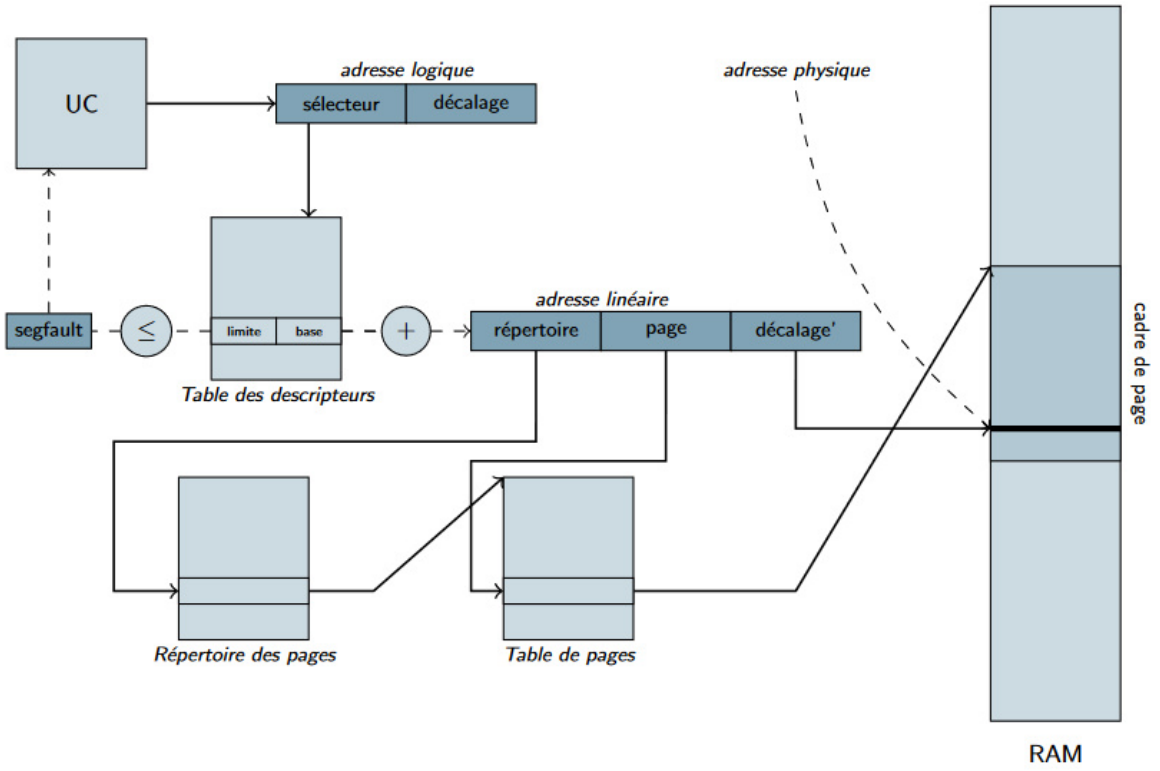
En fait, la MMU comporte travaille également avec une mémoire cache permettant de stocker une table appelée TLB *Translation Lookafter Buffer* contenant les traduction adresses virtuelles vers adresse physique récemment effectuées. Cela permet d'éviter des accès incessants et coûteux à la **table des pages** située en mémoire principale.

II. 4. Des segments aux pages

La mémoire virtuelle allouée à un processus réalise une abstraction supplémentaire en la divisant en segments qui permettent de rendre compte des différents types de zones mémoire nécessaires à l'exécution du programme (code, données, pile, tas...etc). Chaque segment du processus est associé à plusieurs pages de mémoire virtuelle.



Pour résumer, voici comment se déroule la chaîne de traduction d'adresses, de l'adresse logique donnée en terme de segment jusqu'à la traduction en une adresse physique présente en mémoire principale :



II. 5. Intérêt de la mémoire virtuelle

Les avantages de cette **virtualisation** de la mémoire sont multiples :

Augmenter artificiellement les capacités de stockage de la mémoire principale : une page de mémoire virtuelle peut n'avoir aucune correspondance dans la mémoire physique. Ce n'est absolument pas un problème : tant qu'aucune opération de lecture ou d'écriture en mémoire principale n'est réellement exigée, le processus peut continuer à manipuler des adresses virtuelles. Cela permet au processus de continuer à travailler sans avoir à allouer réellement d'espace en mémoire principale. On augmente ainsi artificiellement la mémoire principale disponible. Bien entendu, si tous les processus se mettent en même temps à lire ou écrire de nombreuses données en mémoire principale, des défauts de pages pourront avoir lieu et le système sera ralenti. Mais même dans ce cas, la technique du *swapping* permet d'empiéter sur les disques secondaires et d'augmenter l'espace de la mémoire de travail sans perturber le fonctionnement du microprocesseur, pour qui ces ajustements et optimisations sont transparents.

Cacher la fragmentation de la mémoire au microprocesseur : pour le processus qui manipule des adresses virtuelles, les adresses manipulées correspondent à des zones de mémoire virtuelles contiguës dans une page, alors qu'en réalité, la mémoire physique correspondante peut être fractionnée en plusieurs morceaux dans la mémoire principale. Le fait que le processeur calcule sur des adresses virtuelles contiguës facilite énormément la compilation, l'édition de lien et l'exécution, et permet de nombreuses optimisations au niveau des micro-instructions.

Instaurer des mécanismes de protection : un programme donné ne doit pas pouvoir accéder (en lecture ou écriture) à la mémoire utilisée par un autre programme, voire par le système d'exploitation lui-même. D'une manière simple, chaque programme exécuté par le système d'exploitation se voit attribuer une zone mémoire virtuelle, dans laquelle aucun autre programme ne peut écrire. Le système d'exploitation programme le MMU en déclarant une zone mémoire précise comme appartenant à un programme précis (une zone exécutable de la mémoire). Si une tentative d'accès

à de la mémoire hors plage est détectée, une interruption est levée par le MMU. Celle-ci est interceptée par le processeur et cela a généralement pour effet de stopper le programme, qui reçoit par exemple : un signal `SIGSEGV` (signal de violation de segmentation) sous Unix. Nous verrons d'où vient ce mot de **segment**.

II. 6. Pilotage de la mémoire virtuelle par le système d'exploitation

II. 6. a. Mise à jour de la table des pages de la MMU

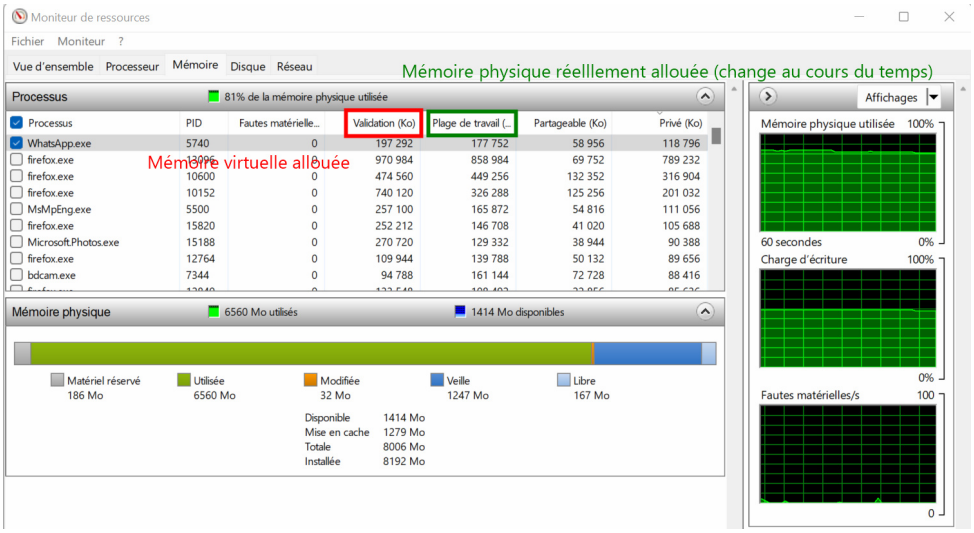
C'est le **système d'exploitation qui pilote la mise à jour de la table des pages** de façon à adapter en permanence la traduction pour optimiser les ressources mémoire de la machine et réguler leur usage. En fait, la correspondance entre adresses virtuelles et physiques n'est pas du tout bijective : certaines pages n'ont tout simplement pas de correspondance dans la mémoire physique, d'autres sont en fait associées à de nombreux fragments dispersés dans la mémoire... Le système d'exploitation se charge à tout moment de gérer et d'optimiser la correspondance mémoire virtuelle/mémoire physique
Sous Linux, la commande `vmstat` (pour *Virtual Memory Statistics*) permet de connaître à un instant donné l'usage de la mémoire virtuelle du système

```
golivier@ordiprof:~$ vmstat
procs ----------mémoire---------- -échange- -----io---- -système- ------cpu-----
 r  b   swpd  libre tampon  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 318948 138024 953424    0    0   407   161  132  454  4  1 95  0  0
golivier@ordiprof:~$
```

La consultation du manuel de cette commande permet de comprendre le sens des différents champs :

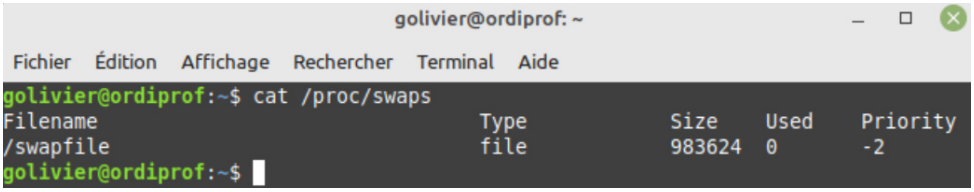
```
FIELD DESCRIPTION FOR VM MODE
   Procs
       r: The number of runnable processes (running or waiting for run time).
       b: The number of processes in uninterruptible sleep.

   Memory
       These are affected by the --unit option.
       swpd: the amount of virtual memory used.
       free: the amount of idle memory.
       buff: the amount of memory used as buffers.
       cache: the amount of memory used as cache.
       inact: the amount of inactive memory.  (-a option)
       active: the amount of active memory.  (-a option)

   Swap
       These are affected by the --unit option.
       si: Amount of memory swapped in from disk (/s).
       so: Amount of memory swapped to disk (/s).

   IO
       bi: Blocks received from a block device (blocks/s).
       bo: Blocks sent to a block device (blocks/s).

   System
       in: The number of interrupts per second, including the clock.
       cs: The number of context switches per second.
Manual page vmstat(8) line 72 (press h for help or q to quit)
```

Sous Windows, l'application `Moniteur de ressources` permet d'obtenir des informations en temps réel sur l'utilisation de la mémoire virtuelle et de la mémoire physique :



II. 7. Défaut de page et phénomène de swapping

Dans le cas où la MMU demande la conversion d'un page virtuelle qui n'a pas de réalité physique dans la mémoire principale, un **défaut de page** (*page default*) est relevé et c'est le système d'exploitation qui prend la main pour trouver de l'espace en mémoire principale.
Si nécessaire, il peut décider, en dernier recours, de sauvegarder certaines données stockée en mémoire principale un disque dur (mémoire secondaire, moins rapide). On parle alors de **swapping**. Ce phénomène n'est pas souhaitable car l'accès au disque dur ralentit considérablement l'exécution des processus. Outre les partitions de swap, il est également possible de stocker des pages de la mémoire virtuelle dans des fichiers. Dans ce cas, la localisation d'une page comprend le dispositif, l'identifiant du fichier et l'offset à partir duquel la page est accessible dans ce fichier. Par exemple, sous Windows, les données de la mémoire principale sont recopiées dans un **fichier d'échange** stocké sur le disque dur et appelé **pagefile.sys**. Sous Linux, il faut afficher le contenu du fichier `/proc/swaps` pour savoir s'il s'agit de partitions ou de fichier de swaps. Dans l'exemple ci-dessous sur notre machine virtuelle, c'est un fichier :



En pratique, les partitions de swap sont un peu plus rapides que les fichiers de swap car les secteurs qui composent une telle partition sont contigus, ce qui n'est pas toujours le cas avec un fichier de swap. D'un autre côté, il est plus facile d'ajouter ou de retirer des fichiers de swap que des partitions de swap sur un dispositif de stockage. En pratique, les deux techniques peuvent être utilisées.