CHAPITRE 3

LA MÉMOIRE PRINCIPALE D'UN ORDINATEUR

1. Introduction

Un ordinateur est composé de plusieurs types de mémoire. À première vue, on peut d'abord distinguer la mémoire principale à l'interne et les mémoires périphériques à l'externe (appelées aussi mémoires auxiliaires ou mémoires de masse).

Les mémoires auxiliaires sont des mémoires de grande capacité qui permettent de stocker les informations pour une plus longue période que ne le fait la mémoire principale de capacité plus limitée. Ce sont par exemple les disques et disquettes, ou les bandes magnétiques. Ces mémoires ont par contre un temps d'accès plus lent que la mémoire principale, mais leur coût de fabrication est bien plus bas.

La mémoire principale, quant à elle, est une mémoire plus rapide, à laquelle l'Unité Centrale (la CPU) est reliée. C'est là que sont emmagasinées les informations et les instructions à exécuter (par exemple les programmes de l'usager) et que transitent les informations permettent à la CPU d'exécuter ces instructions. C'est à la mémoire principale que ce chapitre s'attarde principalement.

Essentiellement, le rôle de la mémoire est celui d'emmagasiner de l'information et de la restituer au besoin. Pour ce faire, l'information contenue dans la mémoire doit être bien identifiée, de façon à pouvoir la retrouver au besoin.

Il y a plusieurs façons d'organiser une mémoire. La taille des unités d'information qu'elle contient peut également varier. Les choix des constructeurs sont en fonction de ce qu'on attend de l'ordinateur à construire. Les caractéristiques des mémoires dépendent, entre autres, des performances désirées, de la capacité souhaitée, et du budget dont on dispose car généralement leur coût est prohibitif, malgré la tendance à la baisse.

Une mémoire se présente comme un ensemble de cellules ou d'unités de rangement identiques destinées à mémoriser de l'information. Chaque unité de rangement est identifiée par un numéro qu'on appelle son adresse. Pour une mémoire comportant N cellules, les adresses sont les entiers compris entre 0 et N-1. Il existe donc un système physique assurant la correspondance entre chaque unité de rangement de la mémoire et son adresse. Le bloc de contrôle de la mémoire (ou contrôleur) effectue ce travail d'aiguillage.

La mémoire doit aussi pouvoir communiquer avec les autres parties de l'ordinateur, l'unité centrale de traitement (CPU) et les unités d'entrées/sorties en particulier. Cette communication est assurée par des organes de liaison formés par les lignes (ou bus), les portes et les registres. Les lignes sont en réalité des "autoroutes" sur lesquels circulent les bits d'information formant les adresses et les données. Les portes servent à contrôler la synchronisation du traitement en laissant passer les impulsions électriques ou non en étant ouvertes ou fermées. Quant aux registres, ils sont en réalité des mémoires très petites et très rapides qui servent à emmagasiner temporairement certaines informations concernant le travail à effectuer.

De façon générale, plus la quantité de mémoire est importante, plus d'applications simultanément peuvent être lancées. D'autre part, plus celle-ci est rapide, plus le système réagit vite. Le but est donc de mieux l'organiser au mieux pour en tirer le maximum de performances.

2. Technologies utilisées

Jusqu'au milieu des années 70, les mémoires principales étaient constituées de tores magnétiques. Depuis, les mémoires à semi-conducteurs se sont imposées et différentes technologies de mémoires à semi-conducteurs ont été mises au point pour construire de mémoires.

2.1 Mémoires à tores magnétiques

Le "tore" magnétique est en fait un petit anneau, fait de ferrite, qui peut prendre deux états: être aimanté dans un sens ou dans l'autre, selon la charge de courant qui lui est appliquée. Les tores, dont le diamètre est d'une fraction de millimètres, sont disposés en rangées et en colonnes pour former un plan où chaque tore représente un bit. Plusieurs plans peuvent être superposés, de façon à ce que les tores dont les coordonnées (rangée, colonne) soient les mêmes sur chaque plan forment des mots.

L'opération de lecture consiste à repérer le tore correspondant à l'adresse recherchée et à utiliser le fil détecteur pour savoir si le bit contient un 0 ou un 1. Comme cette opération se fait en appliquant une charge sur les fils de rangée et de colonne, chaque opération de lecture a pour conséquence de mettre à zéro le contenu de la cellule lue. Il faut donc lire et réécrire le contenu de la mémoire lue à chaque lecture.

Pour effectuer une opération d'écriture, il faut aussi deux étapes: d'abord mettre la cellule à zéro, puis procéder à l'aimantation de la cellule.

Cette nécessité de procéder à la mise à zéro avant d'écrire et à la restauration du contenu après chaque lecture est un des principaux désavantages des mémoires à tores, puisque ces opérations augmentent le temps de lecture et d'écriture ainsi que les risques d'erreur. L'autre inconvénient des mémoires à tores est leur taille qui ne peut concurrencer la miniaturisation possible avec les mémoires à semi-conducteurs.

2.2. Les mémoires à semi-conducteur

Les mémoires à lecture seulement, ou ROM (Read Only Memory) contiennent un contenu qui est enregistré de façon définitive lors de la construction, et par conséquent ce contenu est conservé même en cas de perte de tension électrique. Ce n'est pas le cas des mémoires vives ou RAM (pour Random Acces Memory) qui voient leur contenu s'envoler dès que la tension électrique disparaît, par exemple, dès que l'appareil est éteint. Les mémoires vives peuvent par contre être lues et écrites autant de fois que nécessaire, car leur contenu n'est pas "câblé".
Quoique laisse supposer l'appellation RAM (Random Acces signifie accès aléatoire), les deux types de mémoire ont des accès "aléatoires", en ce sens que le temps nécessaire pour accéder à une information varie "au hasard", par opposition aux mémoires où l'accès se fait de façon séquentielle, comme sur les bandes magnétiques.

2.3. Les mémoires à lecture seule (ROM).

Une mémoire ROM est d'abord un circuit intégré. Cela veut dire qu'il s'agit d'un ensemble de circuits inter-reliés dans le but d'exécuter une fonction. Dans ce cas-ci, la fonction est d'emmagasiner un certain

nombre d'informations et de les restituer au besoin. Le boîtier ROM comporte donc les différentes unités d'information (bits) et les circuits nécessaires pour "livrer" la partie d'information requise lorsqu'on lui en fait la demande et qu'on lui fournit l'adresse de la cellule d'information recherchée.

Les entrées du circuit sont donc les différents bits contenant l'adresse de l'information recherchée, et les sorties sont les différents bits contenant les données. À cela s'ajoute une entrée qui signale l'ordre de lecture.

2.3.1. Les "PROM" ou ROM programmables.

Comme il est coûteux de construire des mémoires ROM pour des applications très spécifiques, il existe des variantes de mémoires ROM qui peuvent être programmées par l'utilisateur. Ce sont les PROM (Programmable ROM), qui sont en fait des ROM "vierges" qui contiennent toutes les connexions possibles et sur lesquelles un appareil spécial, le programmeur de PROM permet de détruire certains fusibles internes. Cela revient à éliminer certaines connections pour ne conserver que celles désirées. Les "EPROM" ou PROM effaçables.

2.3.2. Les "EEPROM" ou PROM effaçables électroniquement.

Cette catégorie de mémoires ROM a l'avantage d'être effaçable électriquement, ce qui est plus simple que d'utiliser le rayonnement ultraviolet comme c'est le cas pour les EPROM.

2.4. Les mémoires `lecture-écriture (RAM).

Les mémoires vives ou RAM sont aussi des circuits intégrés. Comme le contenu de chaque cellule peut être lu ou écrit, il doit pouvoir varier. Contrairement au cas des ROM, la sortie correspondant à une série de bits d'adresse donnée en entrée n'est pas fixée dès la construction, mais elle peut au contraire changer selon le programme utilisé et les données qui l'alimentent.

Les opérations diffèrent selon qu'on procède à une lecture ou à une écriture. Dans le cas d'une lecture, les bits qui constituent l'adresse sont "reçus" par un décodeur d'adresse qui localise la cellule recherchée. Selon que cette cellule contient un 0 ou un 1, la donnée est acheminée en sortie sur la ligne de lecture/écriture d'un 0 ou sur la ligne de lecture/écriture d'un 1.

Pour une opération d'écriture, l'adresse est aussi décodée par le décodeur d'adresse qui localise la cellule recherchée, et selon qu'on veut écrire un 0 ou un 1, la ligne de lecture/écriture d'un 0 ou la ligne de lecture/écriture d'un 1 est utilisée pour acheminer la donnée à la cellule désirée.

2.4.1. Les RAM "statiques" et RAM "dynamiques".

On distingue deux sortes de mémoires vives: les statiques et les dynamiques. Dans une mémoire vive statique, un "bistable" formé de deux transistors est utilisé pour représenter un élément de mémorisation. Sans intervention de l'extérieur, le bistable maintient un état électrique représentant une information binaire. À chacun des états, un des deux transistors est saturé et l'autre bloqué, et seule l'application d'une tension électrique peut faire passer le bistable d'un état à l'autre. C'est le fait qu'un état stable soit maintenu sans intervention extérieure qui fait qu'on l'appelle mémoire statique.

Par contre, les mémoires dites "dynamiques" sont basées sur l'utilisation d'un condensateur qui maintient entre ses électrodes une tension électrique de 5 V ou de 0 V qui équivalent aux états 1 et 0. Cependant,

le condensateur se décharge et il faut donc procéder à un rafraîchissement périodique de la mémoire. Cela signifie qu'on procède à une lecture puis à la réécriture de la mémoire régulièrement. Il faut donc une intervention externe régulière, le rafraîchissement, pour maintenir l'état d'une mémoire dynamique. Malgré cet inconvénient, la simplicité des RAM dynamiques permet de les intégrer en plus grand nombre sur une même puce de silicium que leurs concurrentes statiques. C'est d'ailleurs cette caractéristique qui a contribué à généraliser l'emploi de RAM dynamiques. IBM, notamment, les a introduites sur ses micro-ordinateurs IBM PC

Tableau 1 : Tableau Comparatif DRAM VS SRAM

Caractéristiques	DRAM	SRAM
Données sont emmagasinées dans	Capacitor	Transistor
À besoin d'être rafraîchi	Oui	Non
Lire une données est destructif	Oui	Non
Vitesse	Lente	Rapide
Température	Froide	Chaude
Prix	Bas	Haut

2.4.2. Classification de la mémoire dynamique.

Chaque technologies ont leurs avantages et leurs désavantages, on prend comme règle générale que plus c'est rapide, plus ça coûte cher.

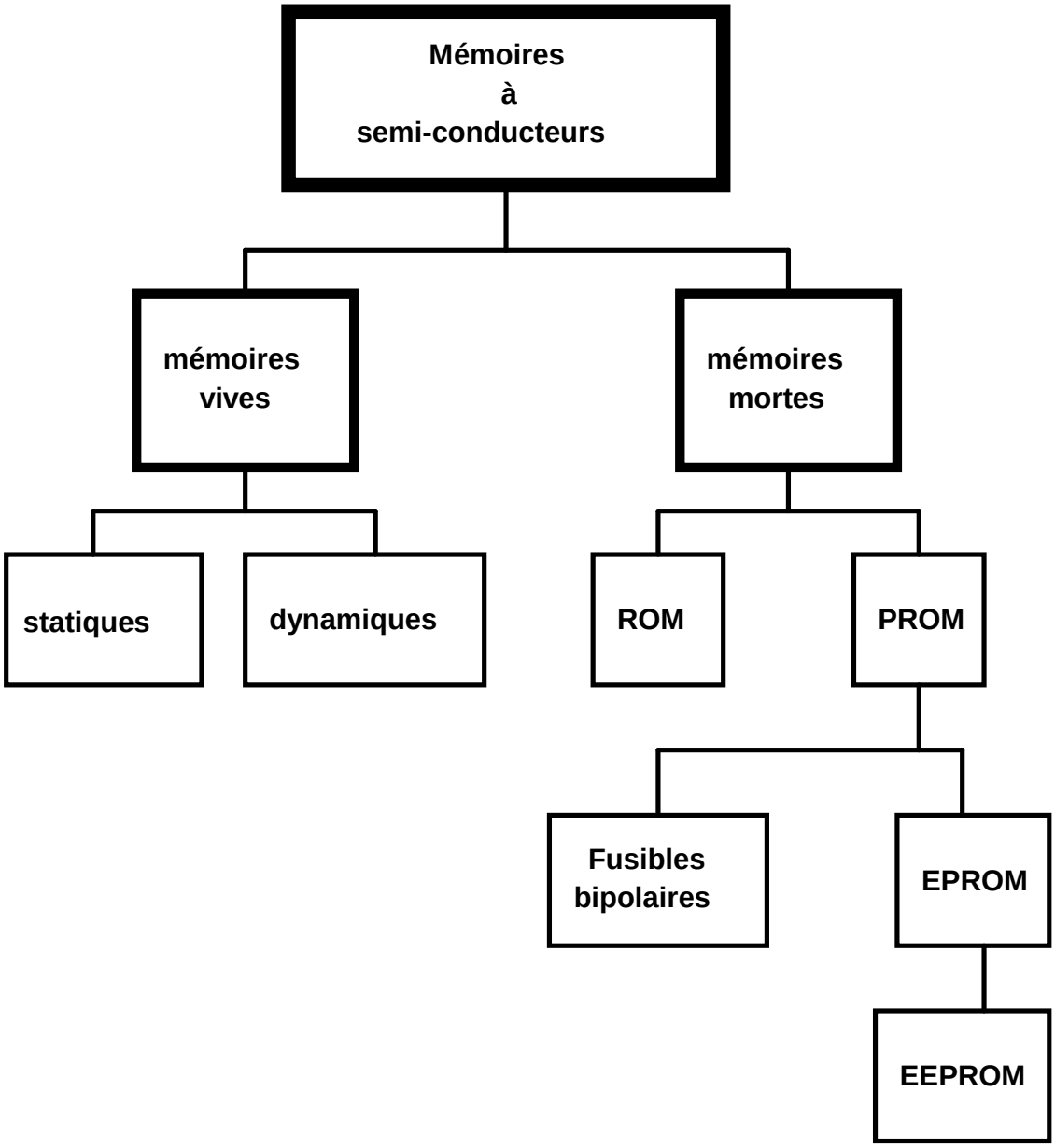
DRAM (Dynamic RAM), est la mémoire la moins chère et la plus répandue des mémoires semi-conducteurs.

FPM (Fast Page Mode), est une mémoire Dram standard qui est disponible avec une vitesse de 60ns.

EDO (Extended Data Output), amélioration du standard FPM.

SDRAM (Synchronous), DDRRAM (Double Data Rate), DRRAM (Direct Rambus), SLRAM (Sync Link), sont tous des type de mémoire qui fonction en synchronisme avec l'horloge du CPU.

Figure 1 : Les mémoires semi-conducteurs.



2.5. Éléments d'information de la mémoire

2.5.1. Le bit

Le bit est la plus petite unité d'information. Il ne peut prendre que deux états différents et par conséquent, il ne peut représenter que deux états différents:

0	ou	1
0 V	ou	5 V
VRAI	ou	FAUX
OUI	ou	NON
PAIR	ou	IMPAIR

2.5.2. Le caractère

Étant donné que le bit est une unité d'information trop petite, on regroupe les bits de façon à créer des unités d'information qui puissent représenter un plus grand nombre d'états différents. Un groupe de 8 bits permet de représenter 28 états différents, donc 256 valeurs au lieu de 2.
Ces groupes de bits sont appelés caractères. Bien qu'on puisse retrouver des caractères de 4 bits, et de 6 bits, les caractères de 8 bits (ou octets) sont les plus répandus. À tel point que les mots octet et caractère sont souvent utilisés comme des synonymes.

L'intérêt de l'octet est facilement compréhensible: d'abord, avec ses 256 possibilités, l'octet permet de représenter les lettres majuscules et minuscules, les chiffres et une série de caractères spéciaux (caractères de contrôle pour l'imprimante, pour le curseur, etc). Ensuite, comme les 10 chiffres du système décimal se codent sur 4 bits (qui offrent somme toute une possibilité de 16 états distincts), l'octet peut aussi être utilisé comme deux caractères de 4 bits chacun, ce qui permet de créer des algorithmes spéciaux pour les calculs décimaux.

2.5.3. Le mot

Bien que le caractère permette de représenter tous les symboles désirés, il ne permet pas de représenter toute l'information nécessaire pour faire un calcul par exemple ou encore représenter une instruction machine. On a donc besoin de chaînes de caractères. Les caractères sont alors regroupés en unités de rangement plus grandes appelées mots.

Grosso modo, le principe du mot est de permettre de contenir une opération, une instruction, c'est-à-dire un opérateur accompagné d'une ou plusieurs opérandes. L'idée est de minimiser le nombre d'accès à la mémoire nécessaires pour exécuter une instruction. Le mot est l'unité de base pour le transfert d'information entre la mémoire et l'unité centrale.

Afin de ne pas gaspiller d'espace mémoire, les mots doivent être assez courts pour être raisonnablement remplis. Ils doivent en même temps être assez longs pour permettre de contenir l'information nécessaire pour exécuter des opérations simples.

Les mots peuvent être de différente longueur selon le type d'ordinateur ou selon le genre d'application. Certaines machines permettent en effet de varier au besoin la longueur des mots. On trouve des mots de 12, 16, 24, 32 et même 60 bits dépendant du type de machine. Sur les micro-ordinateurs, les mots sont de 16 bits, c'est à dire 2 octets. Sur les plus gros ordinateurs, les mots sont de 32 bits et plus. Cependant, les nouveaux microprocesseurs présentent des mots de 32 bits sur les micro-ordinateurs, les rendant beaucoup plus efficaces et plus performants. Certains utilisent le terme mot uniquement pour désigner des unités de 2 octets. Il faut donc se méfier du sens donné à ce terme. Certains parlent aussi de double mots pour désigner des unités de rangement de 4 octets et de quadruple mots pour désigner des unités de 8 octets.

3. Le bloc

Pour faciliter l'adressage et le transfert avec les périphériques, qui sont relativement plus lents à fournir aux demandes dû aux déplacements mécaniques que cela exige, les mots sont groupés en blocs. La taille des blocs varie d'une machine à l'autre. À titre d'exemple, la capacité de votre compte sur le disque relié à l'ordinateur est mesurée en blocs allant de 400 à 1000 blocs. Cette notion sera étudiée plus à fond dans le cadre du cours "structure de fichiers".

4. Les caractéristiques de la mémoire principale

Outre la taille, ou capacité de la mémoire, certaines caractéristiques de la mémoire principale sont des indicateurs importants pour mesurer la performance d'un ordinateur. Les principales sont la taille des mots-mémoire, le temps d'accès et le débit binaire.

4.1. La taille des mots-mémoire

Comme le mot (ou cellule) est l'unité adressable de base, et que la longueur des mots varie selon les machines, la taille des mots mémoire, c'est-à-dire le nombre de bits élémentaires regroupés sous une même adresse et formant le mot, est une caractéristique importante d'un ordinateur. Des unités de rangement plus longues permettront d'emmagasiner des opérations plus complexes ou d'atteindre une plus grande précision de représentation des données numériques.

Ainsi, les ordinateurs spécialisés à une tâche spécifique impliquant des calculs scientifiques ont des unités de mémoire de grande taille afin de mémoriser les nombres avec le plus de chiffres significatifs possibles. Les mini-ordinateurs et les ordinateurs spécialisés dans des tâches spécifiques de contrôle traitent en général des informations plus courtes, aussi réduit-on la taille de leurs unités de rangement afin d'en abaisser le coût. Pour les ordinateurs tout usage, la longueur des unités de base de rangement est comprise entre les valeurs extrêmes.

4.2. Le temps d'accès: le cycle-mémoire

Lorsqu'une unité provoque la consultation à la mémoire, il s'écoule un certain délai entre la demande et l'obtention de la valeur demandée. Ce délai est généralement appelé le temps d'accès à la mémoire. Comme chaque unité de l'ordinateur effectue des lectures et des écritures fréquentes à la mémoire, le temps d'accès à la mémoire est déterminant pour la vitesse de fonctionnement de l'ordinateur également liée directement à la fréquence de son horloge.

Pour les mémoires à semi-conducteurs, le temps de lecture est en général différent du temps d'écriture pour des raisons de nature électroniques. On utilise le plus long des deux pour définir le cycle mémoire, qui est un meilleur indicateur de la performance réelle. Pour les anciennes mémoires à tores, il fallait procéder à une mise à zéro avant d'effectuer une opération d'écriture, et une restauration du contenu était nécessaire après chaque opération de lecture. Le cycle mémoire se définissait alors comme étant le temps total nécessaire avant que la mémoire soit en mesure de répondre à une autre requête.

Plusieurs techniques sont mises au point pour tenter d'augmenter la performance de la mémoire. Citons en particulier les mémoires caches. Il s'agit de mémoires très rapides et de faible capacité servant de tampon entre la mémoire principale proprement dite et l'unité de traitement. Ces mémoires caches de très haute performance électronique contiennent la plupart des informations dont, statistiquement, l'unité centrale peut avoir besoin. Cela diminue le nombre de requêtes à la mémoire principale dont le temps d'accès est plus long.

4.3 Le débit binaire

Le cycle mémoire nous dit combien de temps est nécessaire pour effectuer une requête de transfert à partir de la mémoire ou vers celle-ci. Comme la longueur des mots machines varie d'un appareil à l'autre, la quantité de bits transférée lors d'une requête varie aussi.

Il devient donc difficile de comparer la performance des différentes machines.

Le taux de transfert, ou débit binaire nous permet de remédier à cela. En effet, le débit binaire permet de mesurer le nombre de bits transférés par unité de temps. À titre d'exemple, considérons une mémoire ayant un cycle de 1.2 msec et des mots de 36 bits. Le taux de transfert sera:

$$\frac{36 \text{ bits}}{1.2 \times 10^{-6} \text{ sec}} = 30 \times 10^{6} \text{ bits par secondes}$$

Si les mots étaient d'une longueur de 18 bits pour le même cycle de 1.2 msec, on aurait alors:

$$\frac{18 \text{ bits}}{1.2 \times 10^{-6} \text{ sec}} = 15 \times 10^{6} \text{ bits par secondes}$$

ce qui donne du point de vue débit binaire une performance diminuée de moitié.

Par contre, un cycle-mémoire deux fois plus rapide, soit de 0.6 msec., donnerait avec des mots d'une longueur de 18 bits le même débit binaire.

En effet, on a:

$$\frac{18 \text{ bits}}{0.6 \times 10^{-6} \text{ sec}} = 30 \times 10^{6} \text{ bits par secondes}$$

De plus, il existe plusieurs façons d'augmenter le débit binaire. On peut utiliser une technologie plus poussée ou augmenter la longueur des mots-machine. On peut aussi organiser la mémoire de façon à ce que plus de requêtes puissent être satisfaites.

Calcul de débit binaire

$$\text{Débit binaire} = \frac{\text{Longueur du mot}}{\text{Longueur du cycle}}$$

5. Les organes de liaison (bus, portes et registres)

Afin de communiquer entre eux, les différents blocs de l'ordinateur disposent d'organes de liaison que sont les lignes, qu'on appelle souvent bus, les portes et les registres.

5.1. Les lignes (ou bus)

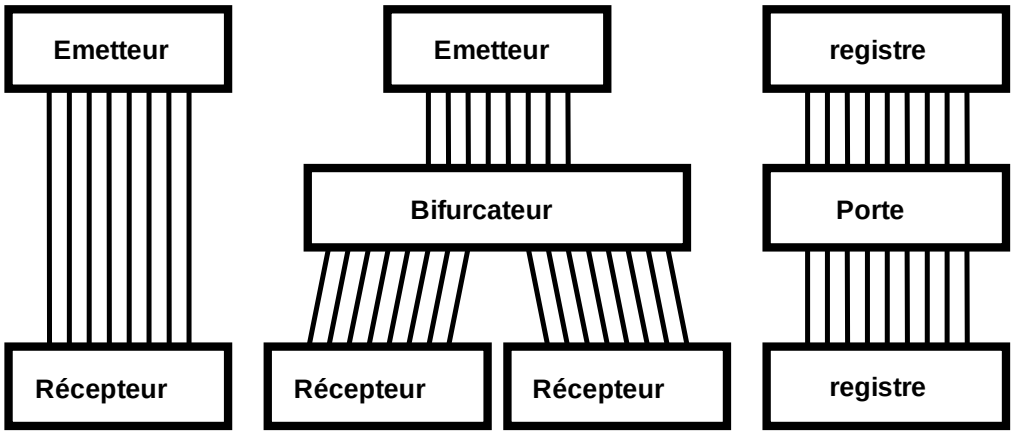
Une ligne est essentiellement un chemin physique entre un émetteur et un récepteur. Une ligne se compose de lignes simples. Il y a autant qu'il y a de bits à transmettre en parallèle. Ce nombre de bits pouvant être transmis parallèlement est appelé largeur de bande de la ligne. C'est-à-dire que des données codées sur 8 bits ne peuvent être transmises que sur une ligne dont la largeur de bande est de 8.

Une ligne est dite orientée si les données ne peuvent y circuler que dans un seul sens. Si l'information peut circuler dans les deux sens, on dit qu'elle est bidirectionnelle.

Sur une ligne, il peut se produire une bifurcation, c'est à dire que la ligne se sépare en deux lignes équivalentes à la première.

Sur une ligne, on peut également installer une ou des portes. Une porte est un dispositif permettant d'autoriser ou d'interdire le passage des informations sur la ligne. l y a toujours une commande spéciale permettant d'ouvrir ou de fermer une porte.

Figure 2 : Bus



5.2. Les registres

Un registre est un dispositif qui permet de mémoriser une information et de la restituer autant de fois que désiré. Un registre est un assemblage de registres élémentaires qui se partageant la même ligne. Comme un registre élémentaire permet de mémoriser un bit, le registre doit avoir la même largeur de bande que la ligne.

Tout registre comporte un mécanisme de remise à zéro (RAZ), qui met tous les registres élémentaires qui le compose à zéro simultanément. L'agencement des matériaux et la technologie utilisés fait en sorte qu'en général, les registres sont **10 fois plus rapides que la mémoire principale**.

Notons qu'on ne retrouve pas seulement des registres dans la mémoire centrale, mais aussi dans les autres composantes de l'ordinateur comme l'unité centrale de traitement (CPU), dans les unités d'entrées/sorties, etc.

Les registres sont utilisés bien différemment que la mémoire principale. Certains registres, les registres à adressage implicite, ne sont même pas accessibles au software, mais sont plutôt nécessaires au fonctionnement propre du processeur central.

D'autres registres, les registres à adressage explicite, servent surtout à mémoriser de façon temporaire des éléments nécessaires pour effectuer une des instructions d'un programme. Il peut s'agir, par exemple, de résultats intermédiaires dans un calcul à plusieurs étapes, de l'adresse d'un opérande, etc. Suivant le type d'information qu'ils sont destinés à retenir, on parlera de registres d'adresses, de registre arithmétique, fixe ou flottant, de registre d'instructions, de registre de base ou de translation, de registre d'index, etc.

On trouve aussi des registres à décalage, qui sont spécialement conçus pour effectuer des décalages sur les chaînes binaires. Le procédé de décalage est particulièrement utile dans les opérations arithmétiques de multiplication et de division binaire que nous aborderons plus loin.

Il y a aussi des registres tampons (buffer) qui permettent de stocker temporairement de l'information entre un dispositif source et un dispositif destinataire non-synchronisés.

5.3. Le bloc de contrôle de la mémoire

Comment la mémoire parvient-elle à retracer une unité d'enregistrement de la mémoire à partir de son adresse? Le décodage de l'adresse est en fait un système d'aiguillage en forme d'arbre binaire. C'est le bloc de contrôle de mémoire qui permet d'obtenir l'information demandée à partir de l'adresse de l'unité d'enregistrement de la mémoire où est logée cette information.

Pour effectuer une requête à la mémoire, au minimum quatre types d'information sont utilisés: si la mémoire est disponible pour accepter une requête; si la requête en est une de lecture ou une d'écriture; connaître l'adresse de l'unité d'enregistrement de la mémoire à laquelle on veut accéder et finalement, il faut la donnée elle-même qui constitue le contenu lu (ou à écrire) dans l'unité d'enregistrement en question. C'est pourquoi le bloc de contrôle de mémoire correspond avec l'extérieur de 4 façons au moins:

- par un registre de données (capacité: 1 mot mémoire)
- par un registre d'adresse (capacité: la largeur d'une adresse)
- par un indicateur libre/occupé
- par un indicateur lecture/écriture

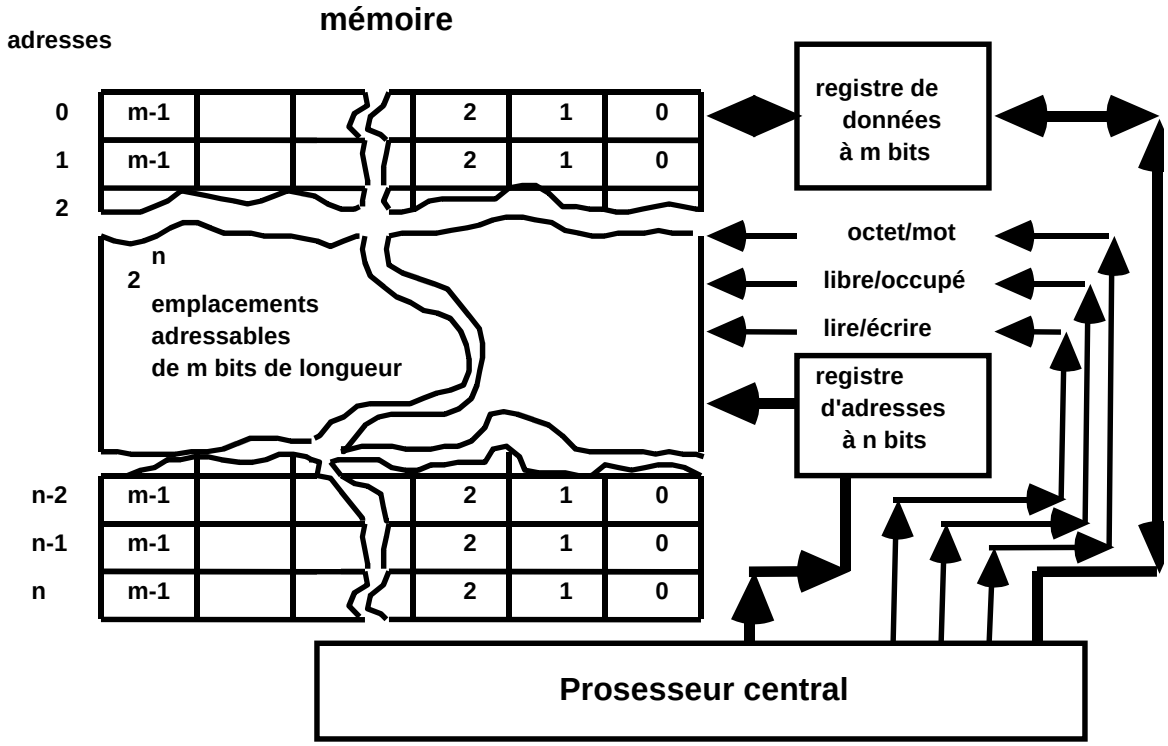
Le registre de données sert à recevoir le mot lu lors d'une lecture ou à garder le mot à écrire avant une écriture. Ce registre doit donc avoir une largeur de bande égale au nombre de bits d'un mot mémoire. S'il s'agit de doubles mots, la largeur doit être de 16 registres élémentaires (soit deux octets).

Le registre d'adresse reçoit l'adresse du mot auquel on veut accéder. Il doit avoir une largeur suffisante. Par exemple, si la mémoire contient 32,768 mots-mémoire, les adresses sont des entiers compris entre 0 et 32,767. Comme il faut 15 bits pour représenter ces entiers en binaire, le registre d'adresse devra avoir une largeur de bande de 15 bits.

5.4. Déroulement d'une requête au bloc de contrôle de mémoire

Le schéma suivant résume les activités du bloc de contrôle de la mémoire.

Figure 3 : Requête à la mémoire



Pour la lecture d'une unité d'enregistrement de la mémoire, la séquence est la suivante:

Si le bloc est occupé, attendre
Placer l'adresse désirée dans le registre d'adresses
Envoyer le signal lire
Attendre que le signal libre soit rétabli. Le registre de données contient alors une copie du mot mémoire désiré.

6. GESTION DE LA MÉMOIRE

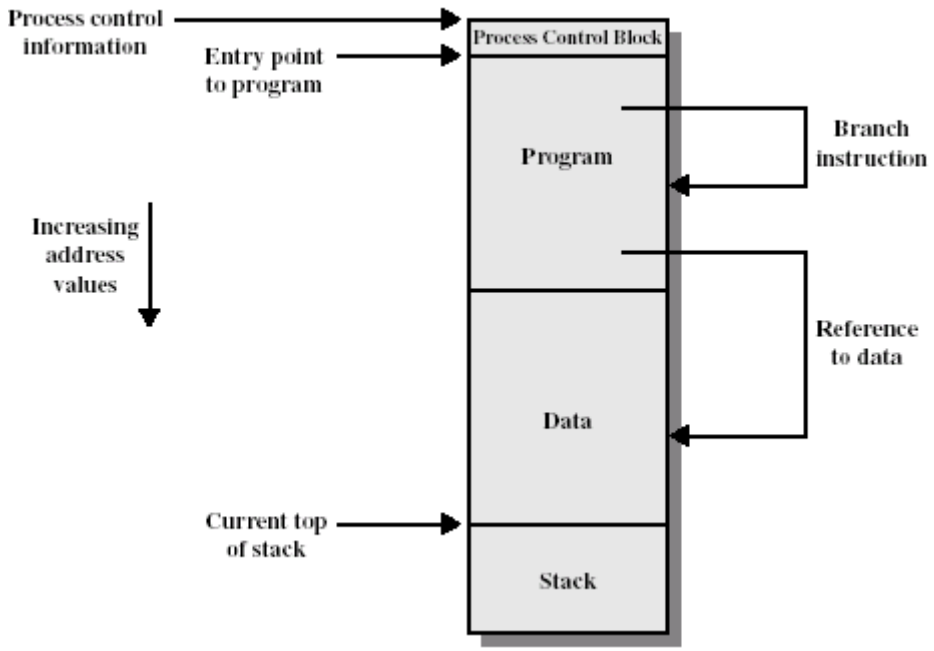
Les mécanisme de gestion de la mémoire doivent être capable de satisfaire à 5 exigences :

- Relocalisation
- Protection
- Partage
- Organisation Logique
- Organisation physique

6.1. Relocalisation

Dans un système supportant plusieurs programmes, l'accessibilité à la mémoire principale est généralement partagée entre plusieurs processus/programmes. Il est presque impossible pour un programmeur de savoir où son programme va résider, ni avec quel programme il devra partager la mémoire. De plus, il est important de pouvoir sortir et entrer (swap) les différents processus pour pouvoir optimiser l'utilisation de la mémoire et du processeur. Lorsqu'un programme transféré sur un disque doit être rechargé en mémoire, nous voulons qu'il soit possible de le **relocaliser** n'importe où en mémoire. Nous serions trop limités si nous avions à le recharger à ça place initiale. Cette possibilité de relocaliser un programme n'importe où en mémoire, nous oblige à gérer l'adressage de notre mémoire

Figure 4 : Adressage pour un programme en mémoire



Le système d'exploitation doit savoir à tout moment l'adresse du programme, de ses données et de sa pile. Il doit être capable de traduire les références à la mémoire dans le code, à l'emplacement physique actuel du programme. Il doit aussi savoir où le début du programme est. Ainsi lorsque nous aurons des références aux données ou encore des instructions de branchement à l'intérieur du programme, le système d'exploration pourra grâce à l'adressage répondre aux demandes.

6.2. Protection

Chaque programme chargé en mémoire doit être avant tout protégé de changements par d'autres programmes. La **relocalisation** des programmes rend la **protection** plus difficile. Étant donné qu'il est impossible de savoir l'emplacement du programme, il est impossible de vérifier l'adresse absolue lors de la compilation du programme et de la protéger. Donc tous les références à des emplacements mémoires sont vérifiés lors de l'exécution, afin de s'assurer que toutes les références ne sont faites qu'à des espaces réservé à se programme Le processeur doit pouvoir arrêter tous programmes qui essaient d'accéder à une espace mémoire qui ne lui appartient pas.

Cette protection doit être faite au niveau du processeur et non du système d'exploitation. Le OS ne peut anticiper tous les tous les références qu'un programme peut faire. Il est donc possible de le faire seulement lors de l'exécution de l'instruction.

6.3. Organisation logique

La mémoire des ordinateurs est généralement organisé de façon linéaire et en une seule dimension. Comme nous l'avons vu, les espaces mémoire consistent en une séquence de bytes ou de mots. Cela ne correspond pas à la méthode utilisée en programmation. La majeure partie des programmes maintenant sont constitué de modules. Les avantages suivants que nous en tirons sont comme suit:

- Modules peuvent être écrit et compilé indépendamment, avec toutes les références entre les modules résolu par le system durant l'exécution.
- Nous pouvons facilement ajouter différents degré de protection (lecture seulement, exécution seulement…).
- Possibilité de partager des modules entre différents programmes.

Nous allons voir un plus loin que l'un des outils utilisé pour satisfaire à ces exigences est la **segmentation.**

6.4. Organisation Physique

La mémoire est organisée en au moins 2 niveaux. La mémoire principale et secondaire. La principale nous donne un accès rapide à un coût plus élevé, elle est volatile, ce qui ne permet pas d'emmagasiner de l'information de façon permanente. La secondaire, plus lente et beaucoup moins dispendieuse, permet d'emmagasiné de large quantité de données de manière permanente.

On utilisera donc la mémoire secondaire pour garder les programmes et les données, tandis que la primaire servira à manipuler les données et les programme présentement en utilisation. Avec cette architecture à deux niveaux, le contrôle de l'échange d'information entre les mémoires primaire et secondaire est primordial. La responsabilité de ce contrôle pourrait être assignée au programmeur, mais ceci est presque impossible pour 2 raisons :

1. La mémoire disponible pour un programme et ses données peut ne pas être suffisante. Le programmeur devra alors utiliser la méthode "overlaying". Cela permet de partager les même espace mémoire pour différents modules du programme et des données. Un programme principal est alors en charge de gérer le chargement/dé-chargement des modules. Cette méthode est une perte de temps pour le programmeur.
2. Dans un environnement multiprogrammé, le programmeur ne sais pas à l'avance, combien d'espace sera disponible et où elle sera localisée.

C'est pour ces raisons que nous laissons le système d'exploitation gérer l'information et échanger entre les deux niveaux de mémoire.

6.5. Partitionnement de la mémoire

La tâche principale de la gestion de la mémoire est de charger des programmes en mémoire pour qu'ils soient exécutés par la CPU. De nos jours, dans tous nos systèmes nous parlons de la **mémoire virtuelle**. Cette méthode est basée sur deux principes de gestions, la **SEGMENTATION** et la **PAGINATION.**

Mais avant de discuter de la mémoire virtuelle et de sa gestion, nous allons parler de techniques plus simples. L'une de ces techniques, le partitionnement, a été utilisée sous plusieurs formes. Les deux autres techniques de bases, pagination simple et segmentation simple ne sont pas utilisées sous cette forme. Cependant, nous nous en servirons à titre d'exemples sans prendre en considération la mémoire virtuelle en tant que telle.

6.5.1. Partitionnement fixe (statique)

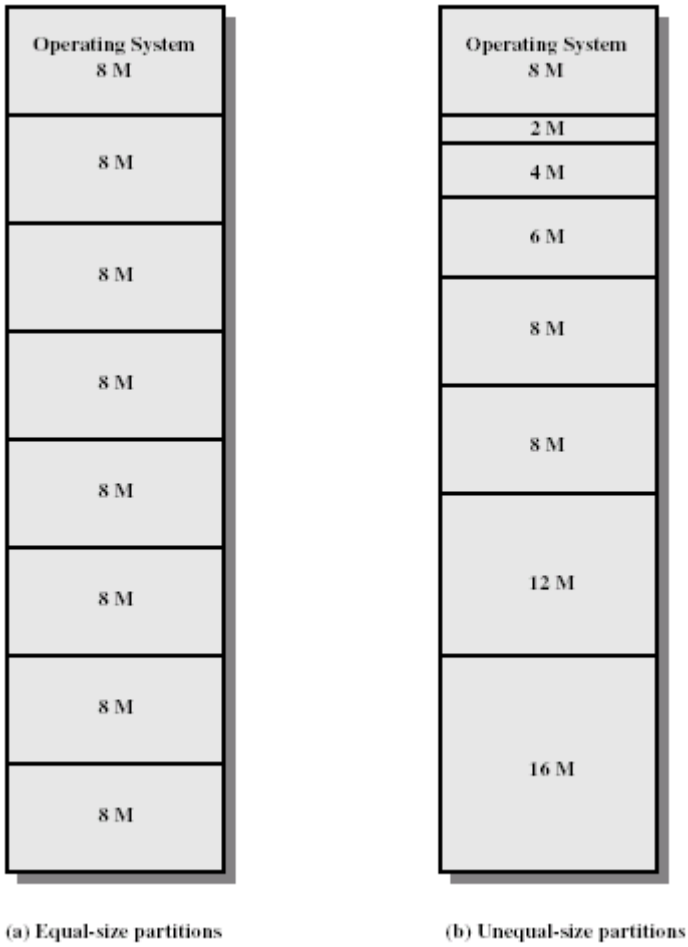
Dans toutes les méthodes de gestion de la mémoire, nous pouvons assumer que les OS occupent une partie fixe de la mémoire principale et que le reste de la mémoire est disponible. Le méthode la plus simpliste est le partitionnement fixe. Il peut exister sous 2 formats, des partitions de même grandeur ou de grandeur différente.

Si un programme est ≤ que la grandeur des partitions il sera chargé dans une des partitions disponibles. Si on veux charger un autre programme en mémoire, mais qu'aucune place n'est disponible, nous devrons alors transférer un programme afin de permettre de charge le nouveau.

De plus, si un programme est plus gros que la grandeur des partitions de la mémoire le programmeur devra utiliser la méthode overlaying. Tous les programmes chargés en mémoire, même ceux qui sont plus petits que la grandeur d'une partition, utilisent l'espace totale de la partition, ce qui cause de la **fragmentation interne**.

Nous pouvons diminuer l'impact de ces 2 problèmes en implantant des partitions de grandeurs différentes. La figure 33b nous montre un partitionnement qui va jusqu'à 16 MB.

Figure 5 : Exemple de partitions fixes



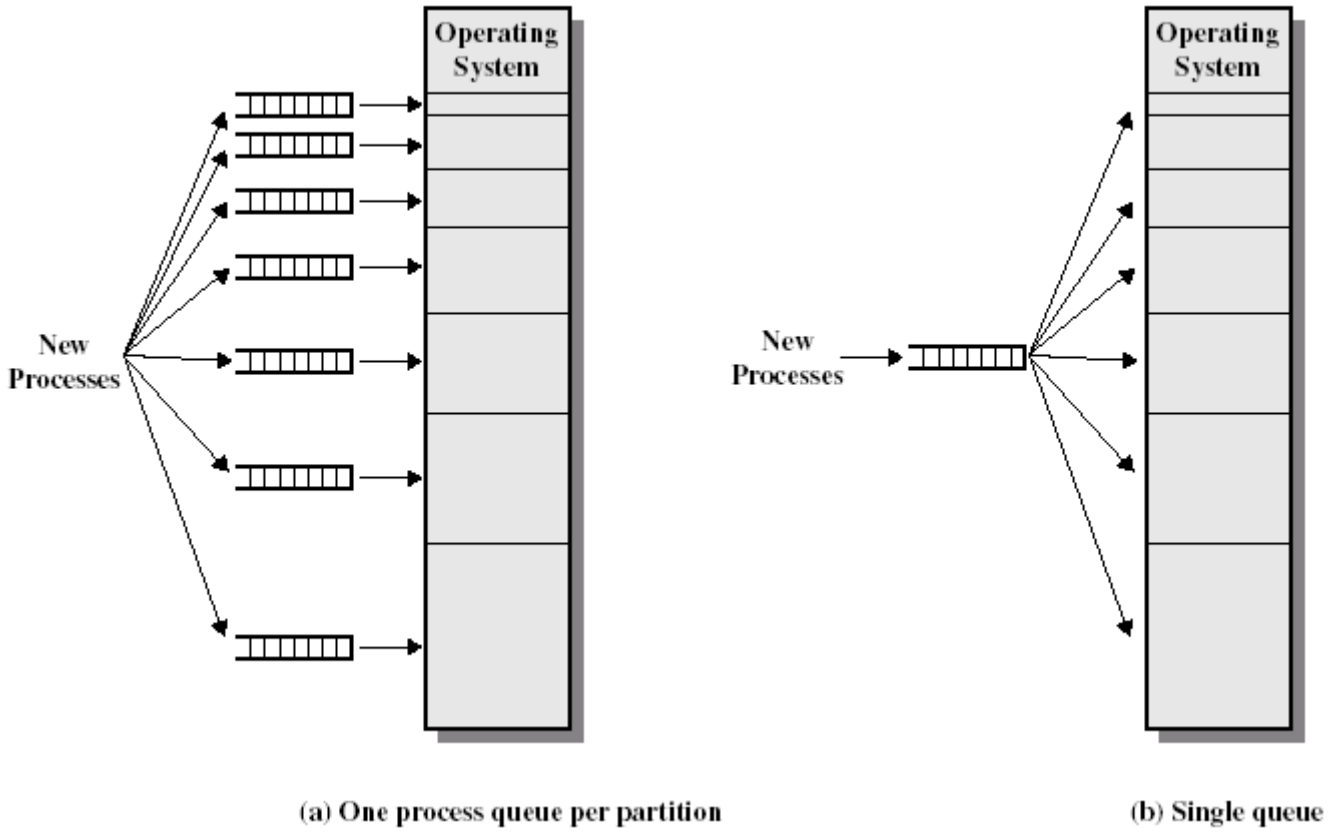
(a) Equal-size partitions (b) Unequal-size partitions

6.5.2. Algorithmes de placement

Lorsqu'un nouveau programme est prêt à être charger en mémoire le système doit alors décider dans quelle partition, il va pouvoir le charger. Dans le cas d'un partionnement de dimension égal fig. 14a, il va choisir un des programmes qui n'est plus en mode actif et le remplacer, étant donné que la dimension du programme ou de la partition n'a pas d'influence sur le choix. Dans le cas de partition de dimension différente fig. 14b, plusieurs possibilités s'offre aux systèmes. Nous allons voir que 2 méthodes existent : une file par partition ou une file pour toutes les partitions.

Si nous avons une queue par partition, le choix de la partition se fera avant même de vérifier la disponibilité. Le système va choisir la partition qui se rapproche le plus du programme et mettre le programme en attente dans la queue de cette partition. Le désavantage est que même s'il reste encore d'autre partition non utilisé, le système ne les utilisera pas. Si au contraire nous avons une queue pour toutes les partitions, le système va essayer de charger le programme dans la partition libre la plus petite. Si tous les partitions sont prises, une décision sera alors prise sur quel programme devra être déchargé.

Figure 6 : Algorithme de placement, partition fixe

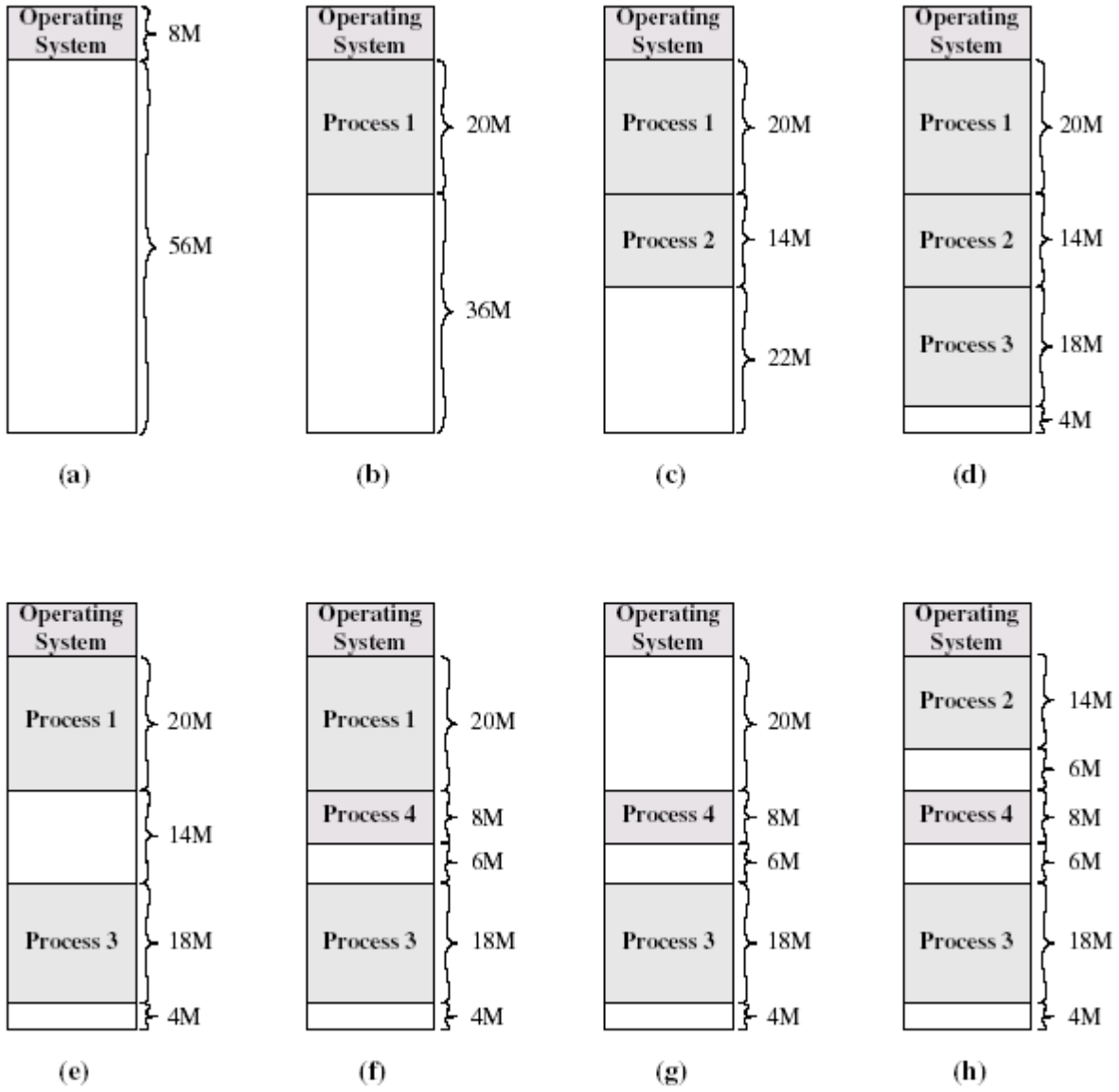


6.5.3. Partitionnement dynamique

Afin de corriger les problèmes reliés au partionnement fixe, une autre méthode a été inventé. Le partionnement dynamique permet d'avoir un nombre différent et de différente dimension pour combler les besoins du système. Quand un programme est chargé en mémoire on lui alloue la quantité exacte de mémoire qu'il a besoin.

Nous pouvons voir à la figure 35, le processus d'allocation dynamique des partitions. Cette technique utilise un nombre variable de partition de dimension variable. Quand un programme est chargé en mémoire, on lui alloue l'espace nécessaire. Cette technique n'est plus utilisé, elle a été remplacé par des techniques plus sophistiqué. L'allocation dynamique cause avec le temps ce que l'on appelle la **fragmentation externe.** On peut voir qu'au fur et à mesure que des programme sont chargés et retirés de la mémoire, il se crée des trous entre les programme. Il a donc été nécessaire de mettre ne place une technique de "**compaction**". LE système va déplacer des programmes en mémoire de façon à combler les trous entre les programmes de manière à offrir de plus grands espaces continues de mémoire. La compaction coûte beaucoup de temps processeur.

Figure 7 : Partionnement dynamique



<

6.5.4. <u>Algorithmes de placement</u>

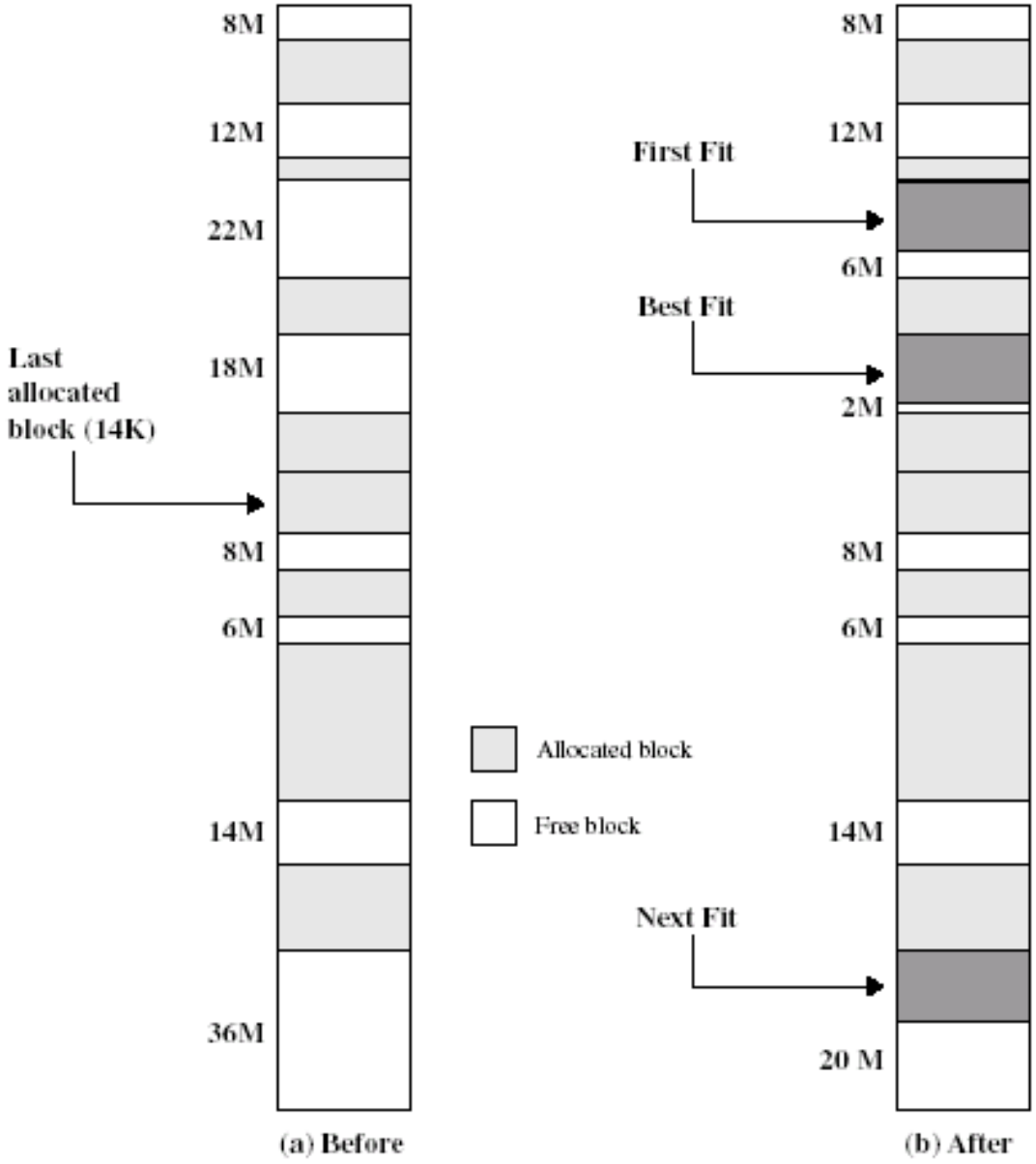
Étant donné que la compaction de la mémoire nous faire perdre beaucoup de temps CPU, il est donc important d'implanter de bon algorithme de placement afin de bien remplir les trous. Nous allons voir 3 méthodes :

1. Best-Fit (meilleur place)
2. First-Fit (première place)
3. Next-Fit (prochaine place).

La meilleure place choisi l'espace disponible qui a la dimension la plus proche du programme à charger. Le première place, commence à vérifier la mémoire en partant d'en haut et va placer le programme dans la première espace qui peut recevoir le programme, sans se soucier de l'espace perdu. La prochaine place commence à vérifier la mémoire à partir du dernier programme placé et va allouer la prochaine

espace qui peut recevoir le programme à charger. La Figure 17 nous montre l'application des trois méthodes.

Figure 8 : Algorithme de placement, partition dynamique



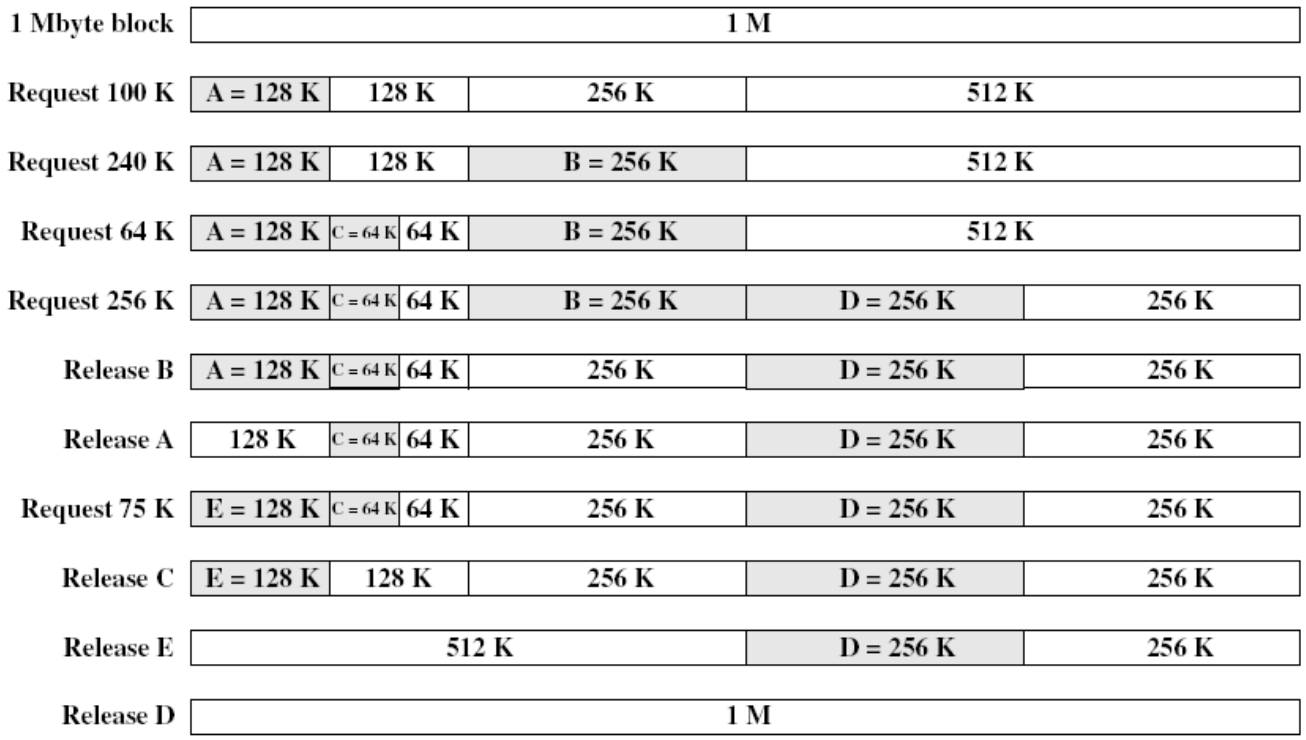
Question : Laquelle des trois méthodes est la meilleure ?

Réponse : Tout dépend de l'ordre dans lequel les programmes sont chargés et déchargés. On peut cependant dire que le First-Fit est non seulement le plus simple à implanter, il est aussi le meilleur et le plus rapide. Le Next-Fit va fréquemment allouer la mémoire à la fin de la mémoire. La plus grosse partie de l'espace mémoire qui est libre est normalement à la fin de la mémoire physique, qui va donc être rapidement fractionner en petites parties, on aura donc besoin de compacter souvent la mémoire. Le Best-fit, contrairement à se que son nom veut dire est la pire des méthodes. En utilisant cette méthode nous allons nous retrouver avec plein de petites espaces, toutes trop petites pour recevoir des programmes et nous devrons compacter la mémoire beaucoup plus souvent.

6.5.5. Le "Buddy System"

Voici un bon compromis entre les avantages et désavantages du partitionnement fixe et dynamique. Cette méthode considère que la mémoire est un bloc d'une grandeur de 2^U. Lorsqu'une requête d'allocation est faite, le système vérifier la grandeur et alloue l'espace disponible ou la divise en 2 partie égale. Lorsque les blocs se libèrent ils vont être regroupé pour redevenir un seul bloc.

Figure 9 : Buddy System



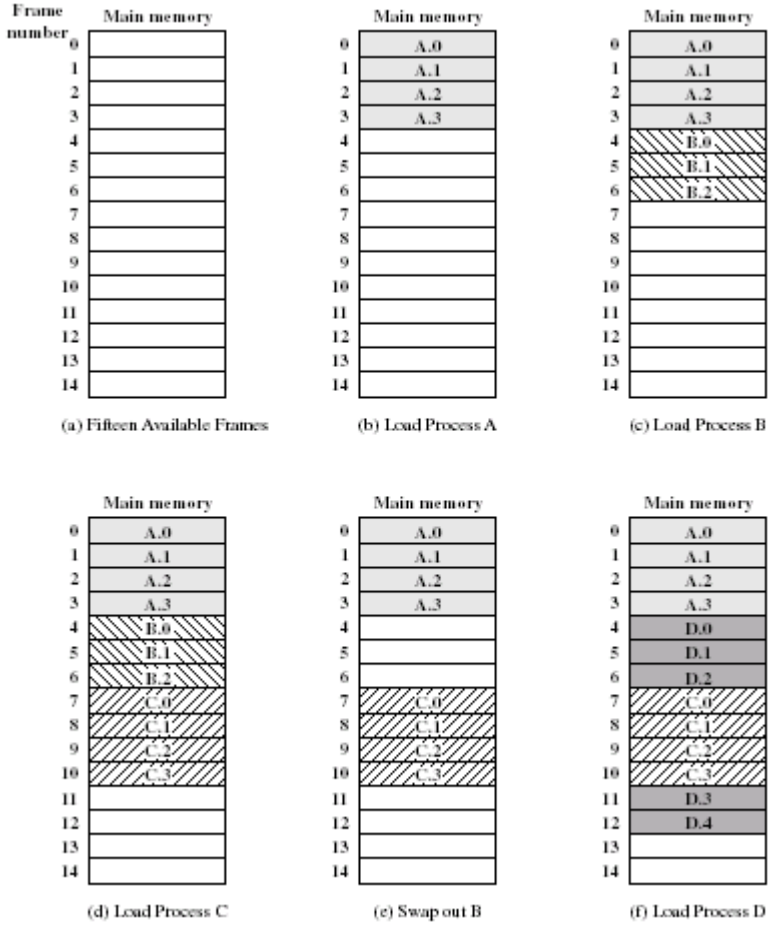
7. La mémoire virtuelle

Le principe de la mémoire virtuelle repose sur l'idée que la taille du programme, des données et de la pile, peut dépasser la mémoire disponible. Le système d'exploitation garde en mémoire les parties du programme qui sont utilisées et stocke le reste dans le disque. Par exemple, on peut exécuter un programme de 1 Mo sur une machine de 256 Ko en choisissant judicieusement les 256 Ko du programme à mettre en mémoire principale à tout instant. Les autres parties effectuant des va-et-vient entre le disque et la mémoire.

7.1. Pagination

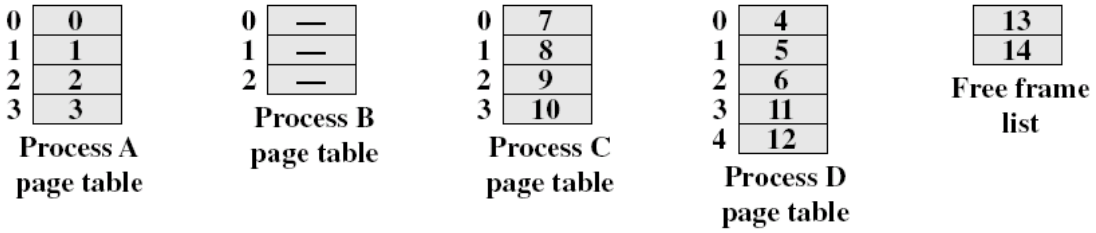
Comme nous venons de voir les techniques de partitionnement fixes ou variables ne gèrent pas efficacement la mémoire, ils causent des problèmes de fragmentation internes ou externes. Nous allons voir maintenant une méthode qui en plus de fractionner la mémoire en petits blocs égaux (frames), va fractionner les programmes en petits blocs (pages) de même dimension que les partitions de la mémoire. Le terme frames ou cadre en français est utilisé car un cadre va tenir une page d'information. Nous verrons que grâce à cette méthode nous éliminons complètement la fragmentation externe, et que la seule fragmentation interne sera causé par la dernière page du programme qui peut être plus petite qu'un cadre. Afin de rendre l'utilisation plus pratique, nous allons aussi fixer la grandeur des pages à un nombre égale à une puissance de 2^X. En utilisant cette méthode nous rendons transparent la pagination aux programmeurs car l'adresse relative est la même que l'adresse absolue.

Figure 10 : Allocation à l'aide de la pagination



Nous pouvons voir dans cet exemple le processus d'allocation de la mémoire. On peut voir à la figure f que même si il ne reste pas d'espace continue on peut charger un programme sans faire de compactage des autres programmes. Nous aurons cependant besoin d'une table pour pouvoir retrouver les adresses de chaque partie de notre programme. On peut voir à la figure 11 la table des pages au temps (f). On peut y voir aussi une liste des frames qui sont disponibles.

Figure 11 : Tables des pages de la figure 10.



7.2. Les modes d'adressage.

Nous avons vu dans la dernière section que les programmes qui sont chargé en mémoire le sont d'une manière un peu aléatoire au niveau de leurs adresses, ce qui rend impossible pour les programmeurs de coder les références à des sauts (if, goto…) ou aux données directement dans les programmes. Pour ce faire le système doit pouvoir gérer l'adressage lui-même. Nous allons faire la distinction entre les différents types d'adressage.

L'**adresse relative** est un type d'adresse logique, l'adresse est exprimée en fonction (relativement) d'un point connu.

L'**adresse physique** ou encore **absolue**, est l'adresse actuel des données en mémoire.

La plupart des systèmes à mémoire virtuelle ont recours à la pagination. Sur tout ordinateur, les programmes peuvent générer un certain nombre d'adresses. Les adresses manipulées par des programmes sont appelées les adresses virtuelles et constituent l'espace d'adresses virtuelles.

Sur les ordinateurs sans mémoire virtuelle, les adresses virtuelles sont directement placées sur le bus de la mémoire et provoquent la lecture ou l'écriture du mot à l'adresse spécifiée. Lorsque la mémoire virtuelle est utilisée, les adresses virtuelles ne sont pas directement placées sur le bus de la mémoire. Elles sont envoyées à l'unité de gestion de la mémoire (MMU) qui traduit les adresses virtuelles en adresses physiques.

L'espace d'adressage virtuel est divisé en petites unités appelées des pages. Les unités correspondantes de la mémoire physique sont les cases mémoires (page frames). Les pages et les cases ont toujours la même taille. Supposons que nous ayons des pages de 4Ko avec 64Ko de mémoire virtuelle et 32Ko de mémoire physique, on a 16 pages virtuelles et 8 cases. Les transferts entre la mémoire et le disque dur se font toujours par pages entières. Quand un programme essaie de lire l'adresse 0, par exemple, l'adresse virtuelle 0 est envoyée au MMU qui constate que cette adresse virtuelle se situe à la page 0 (adresses de 0 à 4095) qui appartient à la case 2 (8192 à 12887). Le MMU transforme l'adresse en 8192 et place cette valeur sur le bus. La carte mémoire ne connaît pas l'existence du MMU. Elle reçoit simplement une demande de lecture de l'adresse 8192. LA MMU a donc fait correspondre les adresses virtuelles comprises entre 0 et 4096 sur les adresses physiques 8192 à 12357.

Cette correspondance au moyen du MMU des 16 pages virtuelles sur n'importe laquelle des huit cas ne résout pas le problème soulevé par le fait que l'espace d'adressage virtuel est plus grand que la mémoire physique. Comme on n'a que huit cas physiques, on ne peut faire correspondre que huit pages virtuelles sur la mémoire physique. Que se passe-t-il si le programme tente d'utiliser une page n'ayant pas de correspondance en mémoire physique, par exemple en effectuant une instruction qui se situe 12 octets après le début de la page 8. Le MMU constate que cette page n'a pas encore de correspondance, et provoque un déroutement : le processeur est restitué au système d'exploitation. Le déroutement est appelé un défaut de page. Le système d'exploitation repère une case peu utilisée et recopie son contenu sur le disque. Il place ensuite la page demandée dans la case qui vient d'être libérée, modifie la correspondance et réexécute l'instruction déroutée.

7.2.1. LA SEGMENTATION

La pagination est une technique qui permet d'implanter un grand espace d'adressage linéaire dans une mémoire physique limitée. Pour certaines applications, il est plus commode d'avoir un espace d'adressage à 2 dimensions. L'idéal serait que chaque processus possède un grand nombre de segments (par exemple 232) formé chacun d'un grand nombre d'octets(232). Les premiers segments (par ex les 64 premiers Ko) sont réservées aux procédures, aux données, à la pile et au tas du programme en cours d'exécution. Les autres segments peuvent chacun contenir un fichier pour que les processus puissent adresser directement leurs fichiers sans avoir à les ouvrir et à utiliser les primitives d'E/S particulières pour les lire et les écrire.

Encore une fois, on sépare le programme en petites parties appelées chacune segment. Ces segments n'on pas à être de la même taille, il existe cependant une taille maximum à la longueur d'un segment.

Comme pour la pagination, une adresse est constituée de deux parties : un numéro de segment et un déplacement (offset). Étant donné que les segments ne sont pas nécessairement de même longueur, la segmentation est semblable au partitionnement dynamique. Il existe toutefois une différence majeure entre les deux. Avec la segmentation, un programme peut occuper plus d'une partition et ces partitions n'ont pas à être continues. La segmentation élimine ainsi la fragmentation interne mais cause de la fragmentation externe, quoique, étant donné que nous pourrions découper les programmes en petits segments, elle devrait être moindre.

7.3. Pagination vs. Segmentation

Contrairement à la pagination qui est invisible pour le programmeur, la segmentation procure un outil pour la gestion des programmes et des données. Normalement le programmeur ou le compilateur va assigner les données et les programmes à des segments différents. Une autre conséquence des segments inégaux est le manque de relation simple entre l'adresse physique et relative

Nous allons voir ici le processus de décodage de la pagination et de la segmentation.

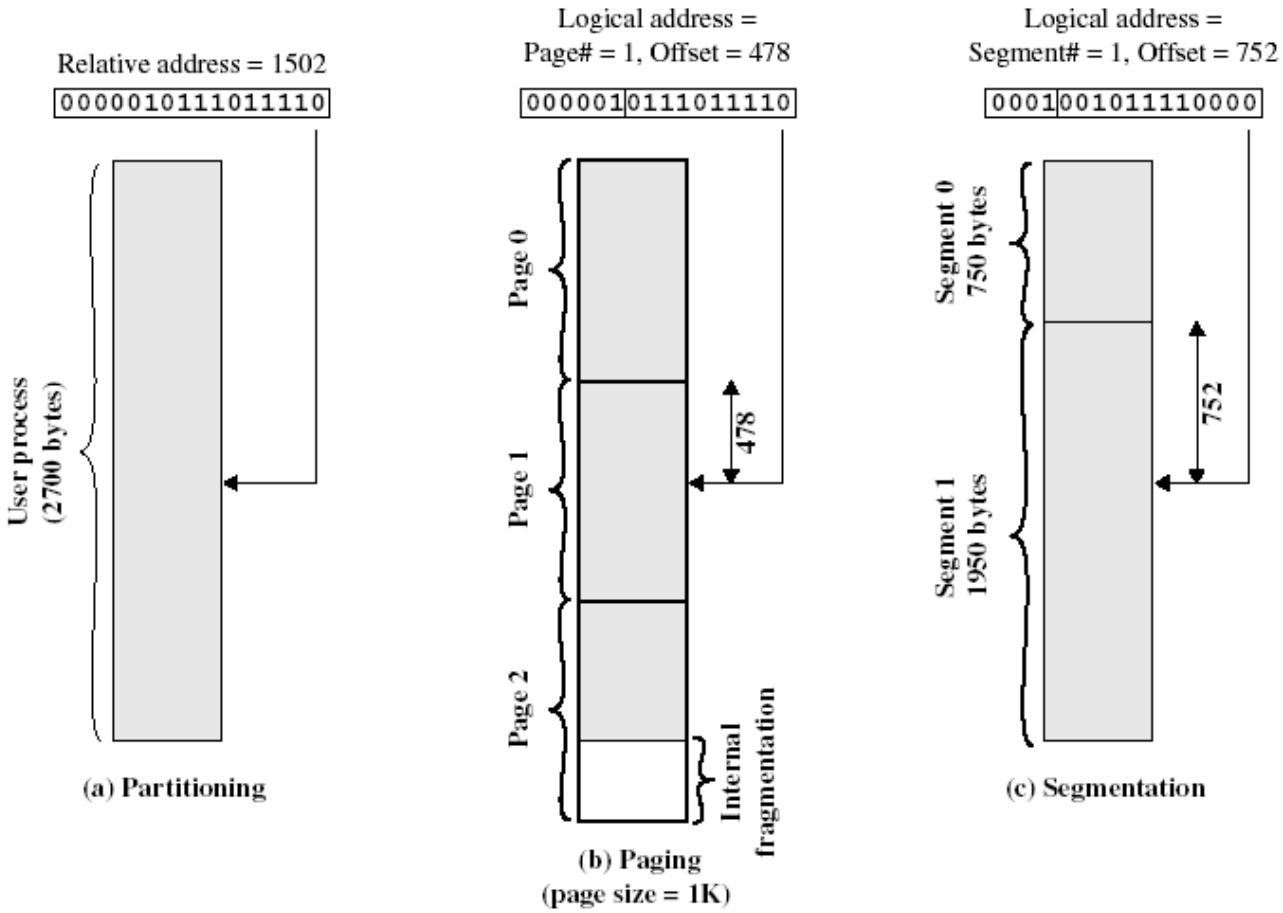
Exemple 49 :

Pour notre exemple nous allons utiliser une adresse de 16 bits et des pages de 1024 bytes (1K). Si l'adresse relative est 1502 ou 0000010111011110 en binaire. Nous avons besoin de 10 bits pour le déplacement avec une grandeur de pages de 1K, ce qui nous laisse 6 bits pour le nombre de pages. Nous pouvons donc avoir 2^6 ou 64 pages de 1K. La figure 53b montre que l'adresse relative de 1502 correspond à un déplacement de 478 (0111011110) à la page 1 (000001). On peut voir ici que

l'utilisation d'une grandeur de page de 1024 K, soit une puissance de 2, nous donne une adresse absolue qui est la même que l'adresse relative.

En se servant du même nombre 1502, si nous utilisons 4 bytes pour le segment et 12 bytes pour le déplacement, nous aurons alors le premier segment avec un déplacement de 752. Cela nous donne aussi une grandeur de segment maximum de $2^{12} = 4096$.

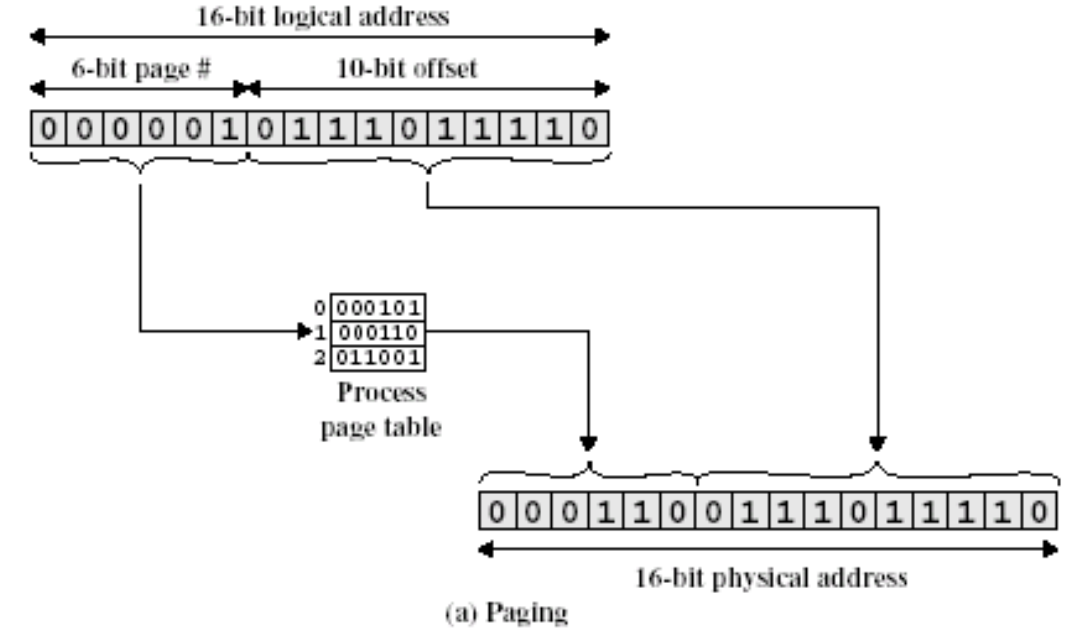
Figure 12:Adresse logique



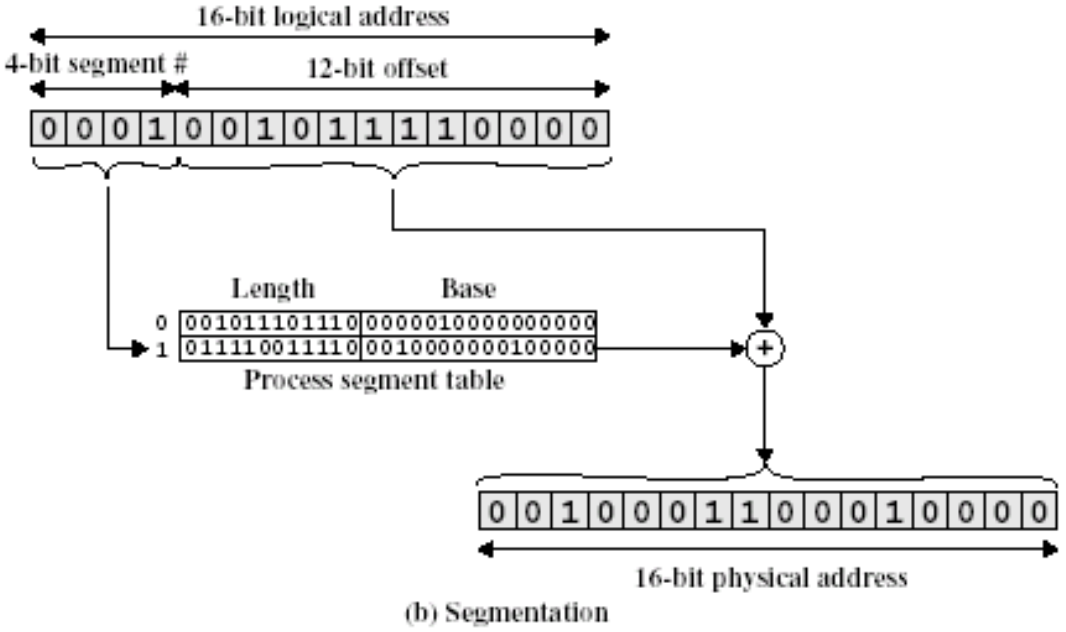
Exemple 50 :

Continuons avec notre exemple. Pour la pagination, si nous savons que la page numéro 1 réside en mémoire au cadre 6 (000110), nous aurons alors physique de 0001100111011110 démontré à la Figure 54a. Alors que pour la segmentation, supposons que le segment 1 réside en mémoire à l'adresse physique 0010000000100000. Alors l'adresse physique sera 0010000000100000 + 001011110000 = 0010001100010000.

Figure 13 : Adresse logique vers adresse physique



(a) Paging



(b) Segmentation