CHAPITRE 4

LA MÉMOIRE DE L'ORDINATEUR

1. Les différents types de mémoires

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 ou alors secondaire).

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. Ces mémoires seront vues plus en détail lorsque nous aborderons les périphériques de l'ordinateur.

La mémoire principale, quant à elle, est une mémoire plus rapide, à laquelle 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 au 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 différentes 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 identifié 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. Tant qu'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.

1.1. Les technologies utilisées pour construire des mémoires principales

Comme on l'a mentionné précédemment, plusieurs types de mémoires composent un ordinateur: la mémoire principale, les registres et les mémoires périphériques. Nous allons pour l'instant nous pencher surtout sur la mémoire principale.

Ceci nous amène à faire la distinction entre deux autres types de mémoire: la mémoire vive (Random Acces Memory) et la mémoire morte (Read Only Memory). Par défaut, la mémoire morte, est une mémoire dont le contenu est fixé en permanence. Ce type de mémoire ne peut donc qu'être lu. Par contre, le contenu de la mémoire vive peut être changé à volonté, il s'agit donc d'une mémoire où on peut lire et écrire au besoin.

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 des mémoires vives et les mémoires mortes. Certaines recherches sont faites dans le but de mettre au point des procédés utilisant la perforation optique ou l'effet Jacobson, mais pour l'instant, il ne semble pas que les mémoires à semi-conducteurs soient vraiment menacées de disparaître; les dernières améliorations technologiques ont plutôt consisté à pousser de plus en plus l'intégration des circuits.

1.1.1 Les 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) sont les mêmes sur chaque plan forment des mots.

Chaque tore est traversé par trois fils: un décodeur de rangée et un décodeur de colonne qui permettent de localiser un tore (donc un bit) et finalement un fil détecteur dont la fonction est de détecter le sens de l'aimantation et de le modifier au besoin. Il suffit d'appliquer à un tore une certaine charge de courant pour changer le sens de l'aimantation, et donc, son état.

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.

1.1.1.1 Désavantages:

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.

1.1.2 Les mémoires à semi-conducteurs.

Un semi-conducteur est un matériau dont la conductibilité électrique se situe entre celle des isolants et celle des métaux. Les plus utilisés sont le germanium et surtout, le silicium.

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.

1.1.2.1 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.

On sait donc qu'à chaque combinaison de bits d'adresse en entrée doit correspondre une combinaison de bits de données en sortie. Comme on l'a dit, le contenu d'une mémoire ROM est fixé à la construction et ne peut plus être changé. C'est donc dire, que dès la construction, on connaît les sorties qu'on désire pour chaque entrée. Il suffit alors de construire le transcodeur, un circuit qui permet de "transformer" les bits d'entrée pour produire les sorties désirées.

1.1.2.1.1 Les "PROM" ou ROM programmables.

Cependant, 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.

Il peut être avantageux de pouvoir modifier une PROM. Mais les fusibles détruits lors de la programmation d'une PROM ne peuvent pas être recréés. C'est pourquoi les EPROM ont été mis au point. Située un peu à mi-chemin entre la RAM et la ROM, l'EPROM (Erasable PROM) est un dispositif

dont le contenu peut être effacé lorsqu'il est soumis à un rayonnement ultraviolet, autorisant ainsi une nouvelle programmation.

1.1.2.1.2 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.

1.1.2.2 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.

1.1.2.3 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

1.1.2.4 Classification de la mémoire dynamique.

Chaque technologies a ses avantages et 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 toutes des type de mémoire qui fonctionnent en synchronisation avec l'horloge du CPU.

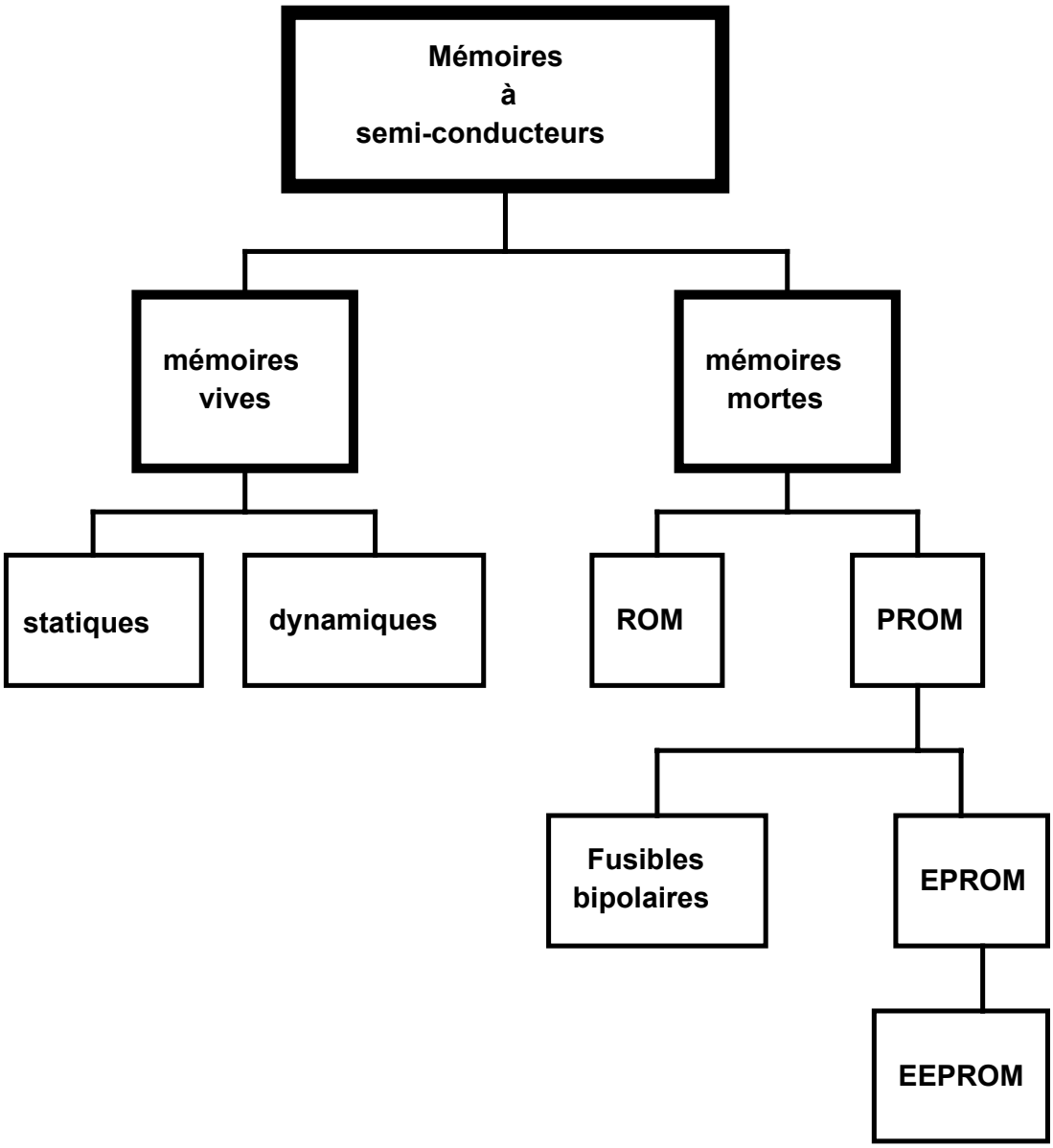
1.1.3 Technologie BIPOLAIRE et MOS.

Que ce soit pour la construction de mémoires ROM ou RAM, les constructeurs ont le choix de deux technologies: les technologies unipolaires MOS ou la technologie bipolaire.

Les principales mémoires bipolaires sont celles de la famille des TTL (Transistor Transistor Logic). On les retrouve surtout sur des ordinateurs de grande taille. Elles permettent d'obtenir des temps d'accès plus rapides , mais leur prix est plus élevé et elles ont tendance à dégager plus de chaleur.

Il existe plusieurs variantes de la gamme des mémoires MOS. Les plus utilisées sont les C-MOS et les H-MOS. On trouve aussi les N-MOS, P-MOS, MOSFET, MOSROM,... Ces mémoires permettent d'atteindre un haut niveau d'intégration. Certaines ont des caractéristiques particulières pour une application, comme les FAMOS (Floating gate Avalanche injection MOS) dont le dispositif à grille flottante permet la réalisation des EPROM.

Figure 1 : Les mémoires semi-conducteurs.

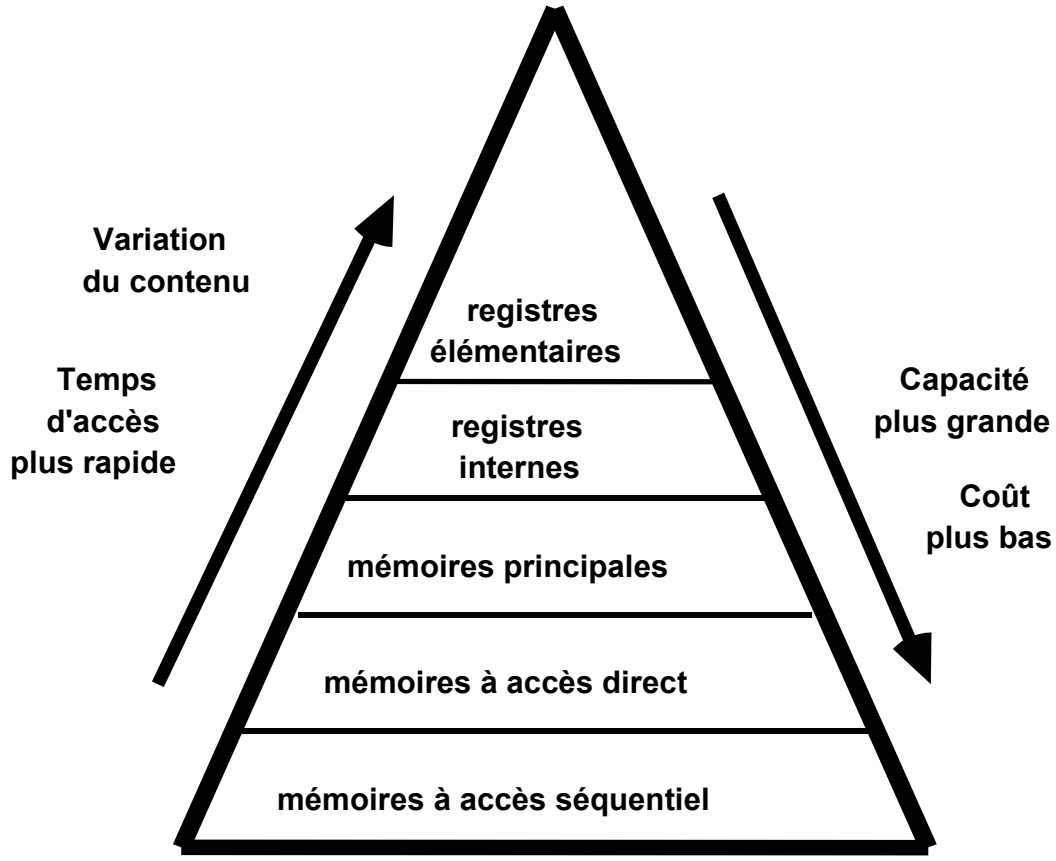


1.2. La hiérarchie des mémoires.

Nous savons qu'un ordinateur, de part sa fonction, doit emmagasiner de l'information, extraite à plus ou moins court échéance par le ou les usagers (information destinée à l'externe) ou encore par des unités mêmes de la machine (information destinée à l'interne). La quantité d'information transférée varie de quelques bits, à quelques caractères, à quelques centaines et milliers de caractères et même jusqu'à quelques milliards de caractères sur de gros systèmes. Toutes ces informations, quel que soit le volume considéré, sont retenues dans des mémoires, dispositifs pouvant conserver des états stables et bien définis. Il existe une certaine hiérarchie ou classification dans les mémoires qui dépend essentiellement des trois critères suivants:

1. le temps d'accès,
2. la capacité de mémorisation,
3. le coût de l'information.

On peut résumer cette classification à l'aide de la figure suivante:



Dans cette pyramide, plus nous allons vers le haut, plus les mémoires qu'on y trouve sont de faibles capacités, coûteuses à produire, ayant des temps d'accès rapides et dont les contenus sont changeants dans le temps. Au sommet, on trouve en fait les registres conçus et réalisés à l'aide des plus hautes technologies. La capacité des ces mémoires se résume à quelques bits ou mots-mémoire. Dans le centre de la pyramide, on trouve la mémoire principale de l'ordinateur dont l'accès est rapide mais pas autant que celui des registres internes dans lesquels les contenus sont susceptibles de changer continuellement. Dans le bas de la pyramide, on trouve les mémoires de masse ou de très forte capacité. Par contre, les temps d'accès sont plus longs à comparer avec ceux de la mémoire centrale et encore plus avec les temps d'accès des registres. On trouve deux principales sous-classes de mémoires de masse: celles à accès direct et celles à accès séquentiel.

Les mémoires à accès séquentiel.
L'accès aux enregistrements se fait de façon séquentielle, c'est-à-dire qu'on lit ou écrit d'abord le premier, puis le second, le troisième et ainsi de suite.. . Les médiums pouvant être utilisés sont: les cartes perforées, le ruban perforé, le ruban magnétique (3000 caractères au centimètre, 100 millions de caractères sur une bobine de ruban de 20 centimètres de rayon) et même le disque magnétique (plus d'un million de caractères sur une disquette de 10 centimètres de rayon pour un coût aussi bas que 1.50$). C'est le type de mémoire le moins coûteux par unité d'information, de temps d'accès le plus long

et dont les capacités sont les plus grandes.

Les mémoires à accès direct.
On trouve dans cette catégorie de mémoires, les disques magnétiques. Le temps d'accès pour lire sur un ruban magnétique est beaucoup plus grand que sur un disque magnétique. Pour comprendre ce phénomène, il nous suffit de faire le parallèle entre le temps pris pour atteindre une chanson qu'on aime sur une cassette comparativement à celui requis pour l'atteindre sur un disque. Par contre, la capacité est moins grande en général et le coût d'un disque est plus élevé que celui d'un ruban magnétique. Il existe plusieurs types de disques, allant des grands disques pour les gros systèmes dont la capacité peut atteindre plusieurs milliards d'octets et de temps d'accès moyen de quelques milli-secondes, des disques moyens (30 centimètres de diamètre) placés en permanence dans un contenant de plastique, jusqu'aux disquettes rigides ou souples que tout le monde connaît pouvant contenir au maximum quelques millions de caractères et de temps d'accès de quelques centaines de milli-secondes. Nous allons maintenant détailler les principales mémoires de masse dans ces deux catégories que sont les disques et les bandes magnétiques.

1.3. Les éléments d'information de la mémoire principale

1.3.1 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**

1.3.2 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.

1.3.3 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 (voir annexe sur la représentation des données sur le VAX).
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.

1.3.4 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".

1.4. 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.

1.4.1 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 ces valeurs extrêmes.

1.4.2 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 ou antémémoires. 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.

1.4.3 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 ms 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

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

1.5. L'organisation de la mémoire principale

La cellule (la plus petite partie adressable d'une mémoire) peut être:
- le bit, on a alors une machine à bits;
- le caractère, on a alors une machine à caractères;
- le mot, on a alors une machine à mots.

Il existe plusieurs façons différentes d'organiser une mémoire. a titre d'exemple, une mémoire formée de 96 bits peut avoir les trois organisations suivantes (en terme d'adressage):

(A) - 6 mots de 16 bits
(B) - 8 mots de 12 bits
(C) - 12 mots de 8 bits,

impliquant respectivement 6, 8 et 12 adresses.

Figure 2 : Mémoire de 96 bits organisée en 6 mots de 16 bits

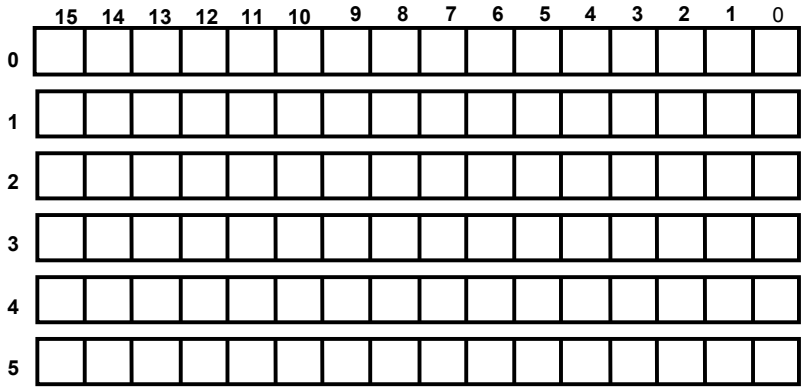


Figure 3 : Mémoire de 96 bits organisée en 8 mots de 12 bits

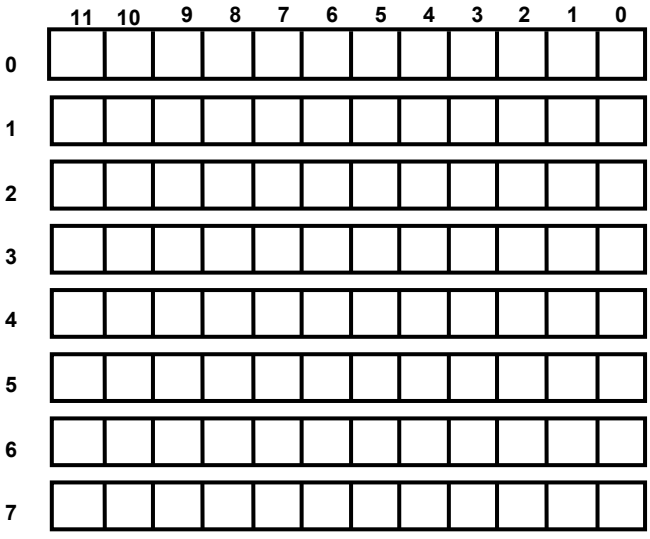
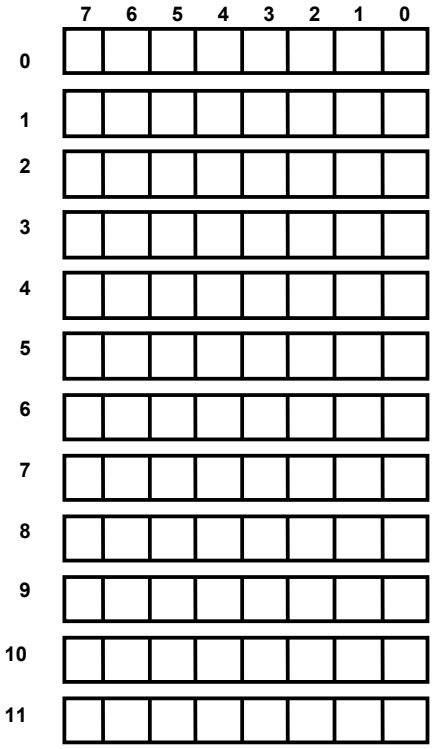
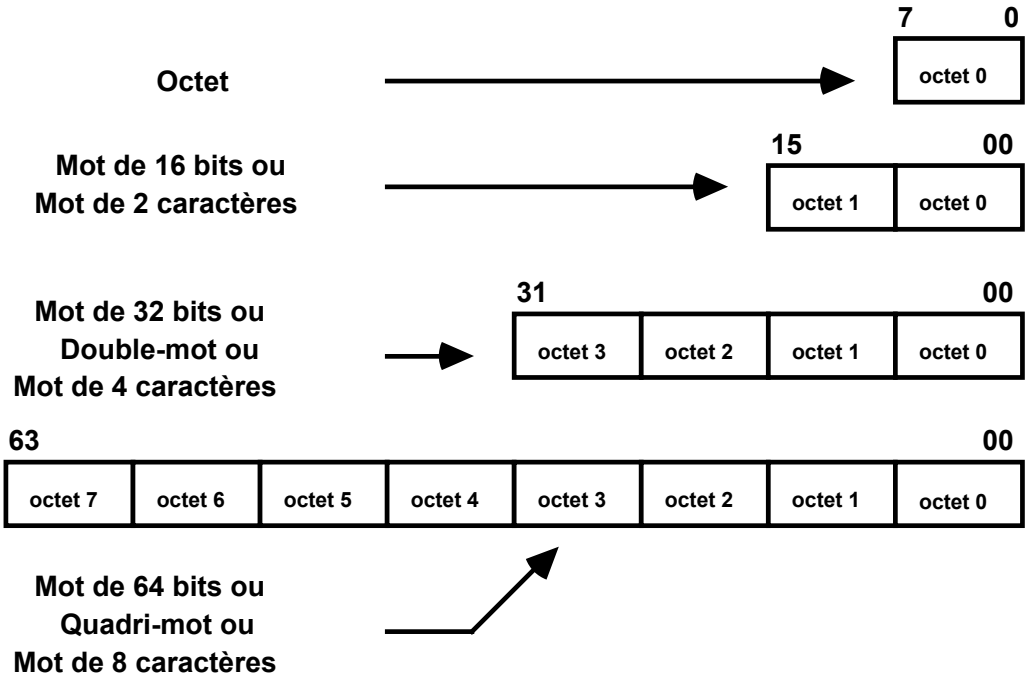


Figure 4 : Mémoire de 96 bits organisée en 12 mots de 8 bits



Lorsque la cellule mémoire comporte plus d'un octet, elle est toujours adressée par rapport à l'octet de plus faible poids. Les octets sont généralement numérotés comme les bits, de droite à gauche.

Figure 5 :Adressage des octets



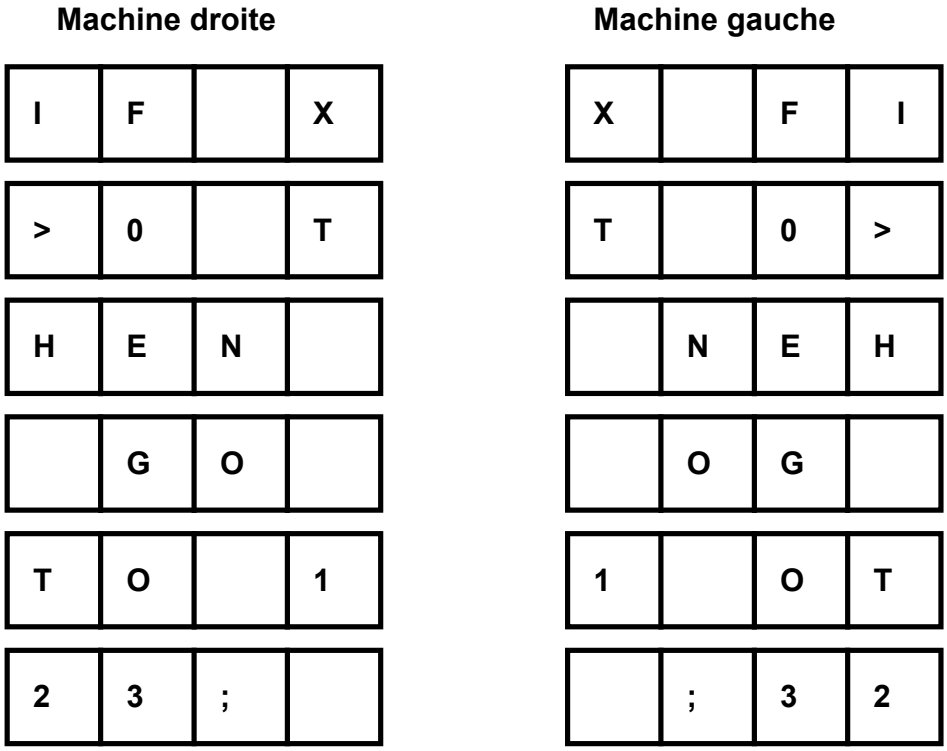
Par contre, le choix d'une organisation de la mémoire a des répercussions sur la performance. Voyons, par exemple, comment une chaîne de caractères, disons:

IF X > 0 THEN GO TO 123;

peut être emmagasinée dans la mémoire.

Si la machine possède une mémoire à 4 octets par mot, alors on trouvera en général 4 caractères par cellule, soit de gauche à droite ou de droite à gauche selon les machines.

Figure 6 : Architecture de la mémoire



Pour tout changement dans la chaîne de caractères demandé par l'usager, le programme éditeur devra faire les déplacements nécessaires à l'insertion de nouveaux caractères, ce qui demande un certain temps.

On peut diminuer ce temps de manipulation si on utilise une autre organisation de la mémoire. Dans cette nouvelle organisation, chaque cellule contient un seul caractère (au lieu de 4), et on y insère un pointeur qui indique où se trouve le prochain caractère.

On remarque cependant que la performance ainsi gagnée se paie par le fait qu'un espace plus grand de mémoire est requis pour mémoriser le même nombre de caractères (dû à la redondance des pointeurs).

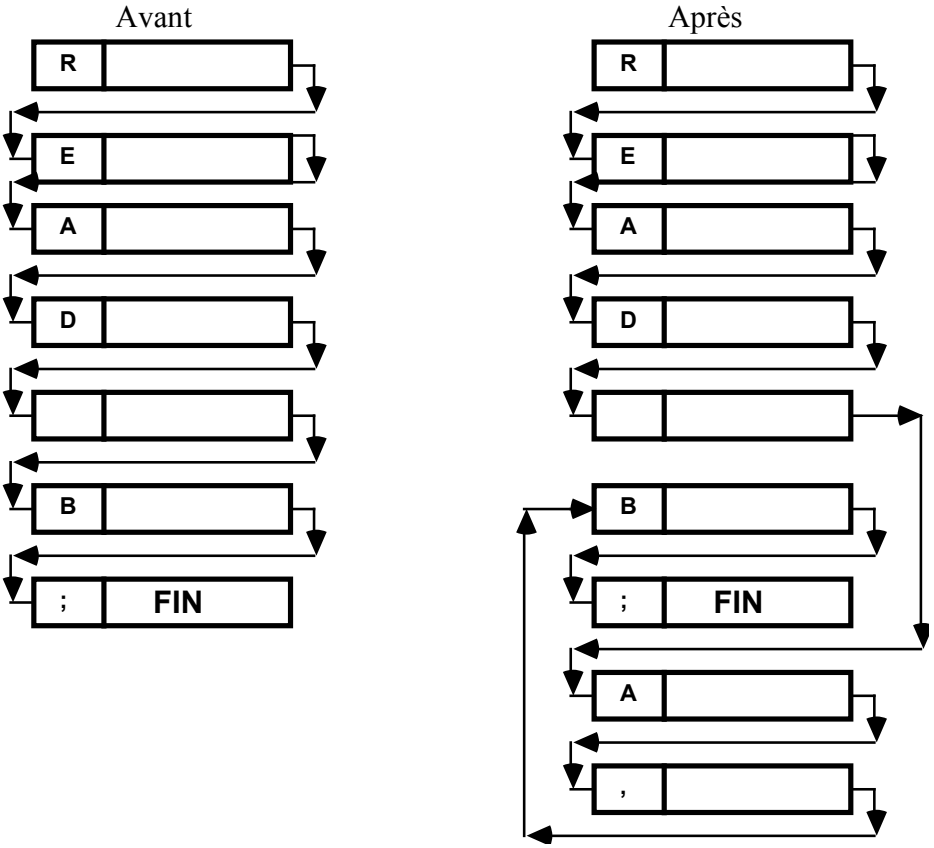
Le schéma suivant montre ce qui se passe lorsqu'on modifie la chaîne originale pour

READ B;

à la chaîne modifiée par l'éditeur

READ A,B;

Figure 7 : Structure par pointeur



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

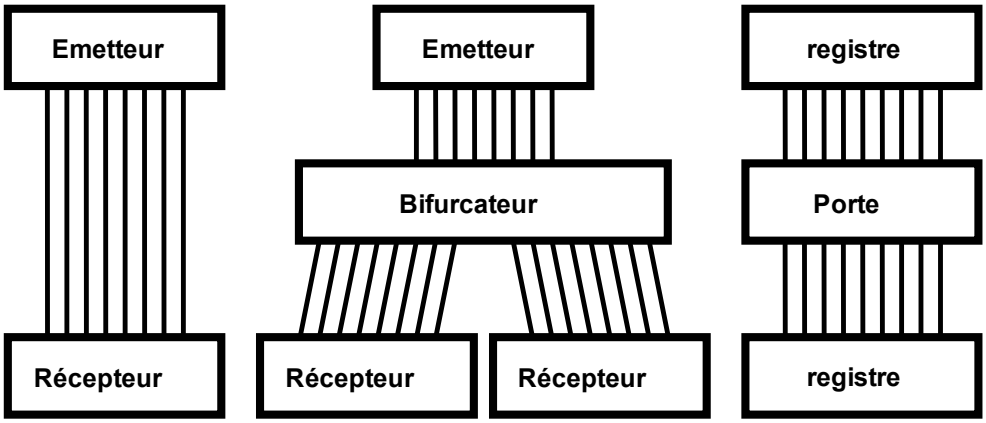
Afin de communiquer entre eux, les différents blocs de l'ordinateur disposent d'organes de liaison que sont les lignes, qu'on appelle souvent bus, les portes et les registres.

1.6.1 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. Il y a toujours une commande spéciale permettant d'ouvrir ou de fermer une porte.

Figure 8 : Bus



1.6.2 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'une 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.

1.7. 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 gigantesque 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: il faut savoir si la mémoire est disponible pour accepter une requête; si la requête en est une de lecture ou une d'écriture; il faut également savoir 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

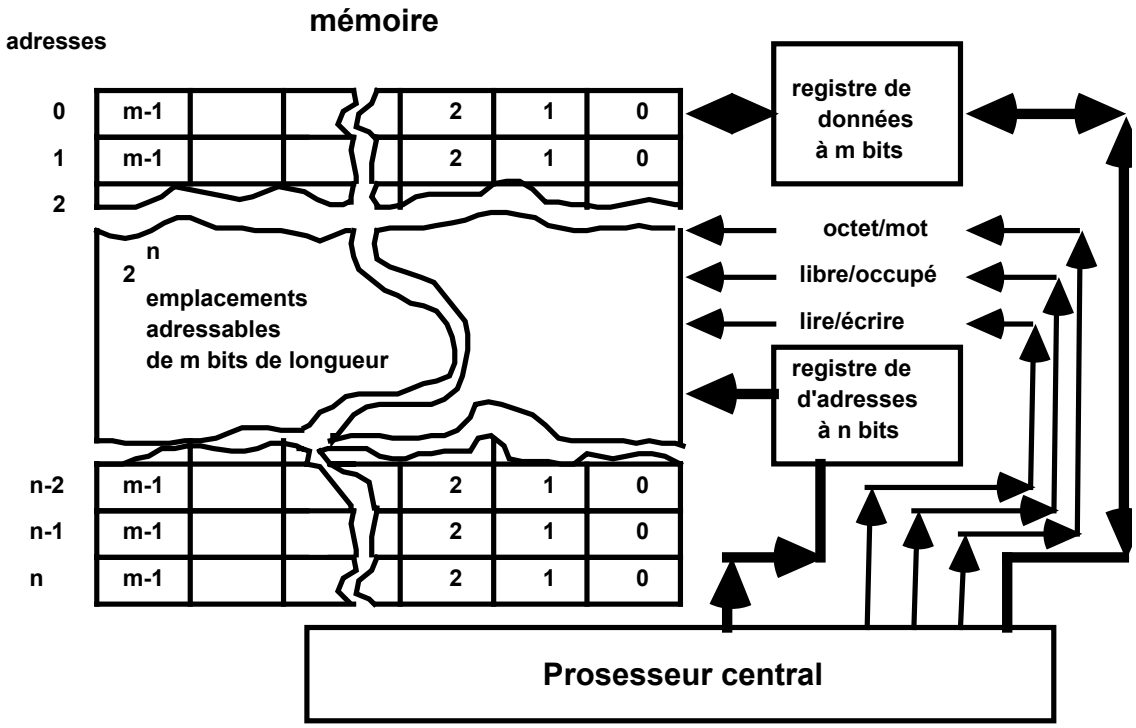
Le registre de données sert à recevoir le mot lu lors d'une lecture ou à garder le mot à écrire avant une écriture. Ce registre doit donc avoir une largeur de bande égale au nombre de bits d'un mot mémoire. S'il s'agit de double mots, la largeur doit être de 16 registres élémentaires (soit deux octets).

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.

1.7.1 Déroulement d'une requête au bloc de contrôle de mémoire

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

Figure 9 : Requête à la mémoire



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

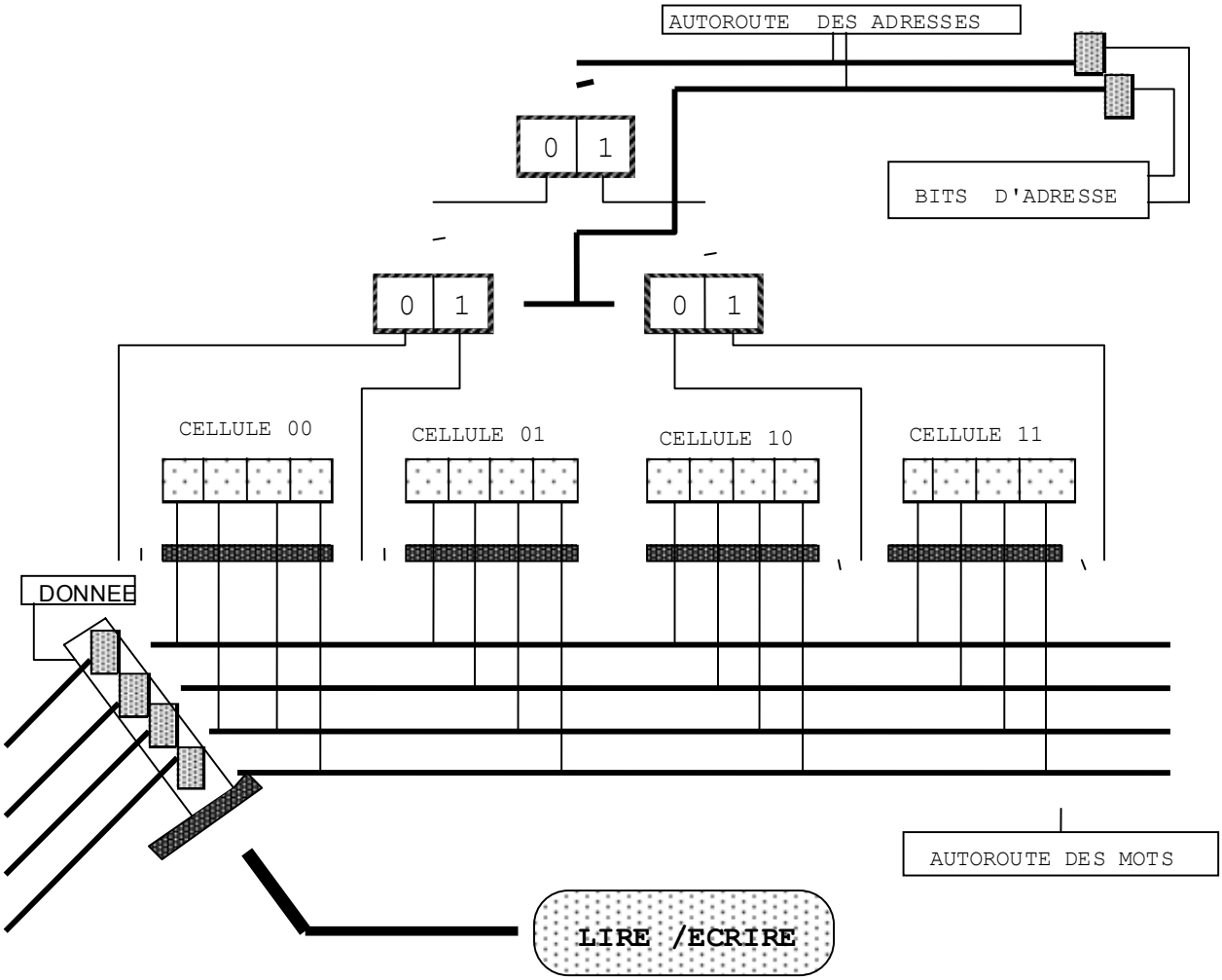
Si le bloc est occupé, attendre
Placer l'adresse désirée dans le registre d'adresses
Envoyer le signal lire
Attendre que le signal libre soit rétabli. Le registre de données contient alors une copie du mot mémoire désiré.

1.7.2 Structure du bloc de contrôle de mémoire

Le bloc de contrôle de la mémoire (ou contrôleur), est en fait une pyramide d'aiguillages faits sur chaque bit de l'adresse. Chaque bit de l'adresse contrôle un étage de la pyramide, de sorte qu'il y a autant d'opérations d'aiguillage qu'il y a de bits dans l'adresse.

Le principe est celui d'un arbre binaire: au premier étage, le premier bit est testé. Selon qu'il s'agit d'un 0 ou d'un 1, on prendra une direction différente (bifurcation). Le deuxième bit est ensuite testé au deuxième niveau, et ainsi de suite, jusqu'à ce qu'on arrive à l'unité d'enregistrement recherchée.

Figure 10 : Câblage interne du contrôleur



1.7.3 Correspondance du bloc de contrôle de mémoire avec l'extérieur

Sur le registre de données est connectée une ligne de données (bus de données) et sur le registre d'adresses est connecté une ligne d'adresses (bus d'adresses).

L'indicateur libre/occupé est connecté à une ligne orientée simple, tout comme l'indicateur d'ordre de lecture/écriture.

Les autres composantes de l'ordinateur, en particulier l'unité centrale et le bloc de contrôle d'entrées/sorties sont branchées sur ces lignes. Plusieurs modules peuvent être connectés au même bloc de contrôle de mémoire grâce à des portes qui évitent le chevauchement des requêtes.

Figure 11 : Une mémoire centrale

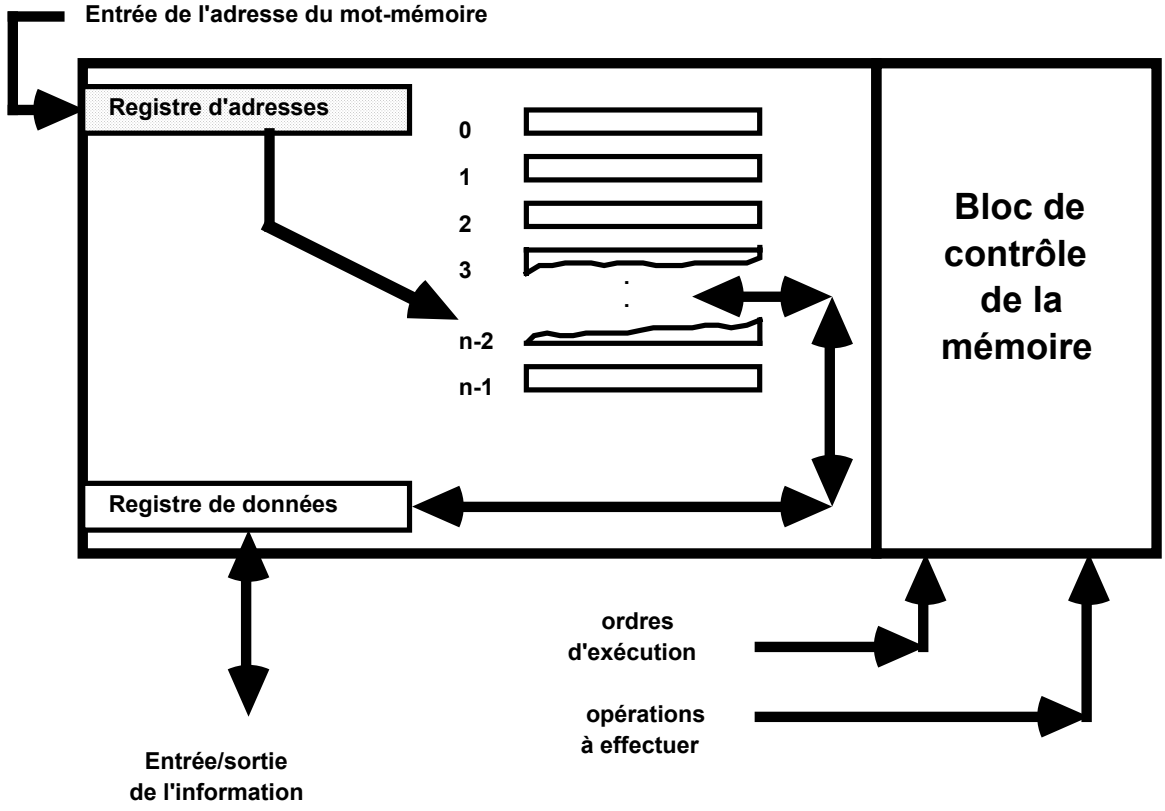
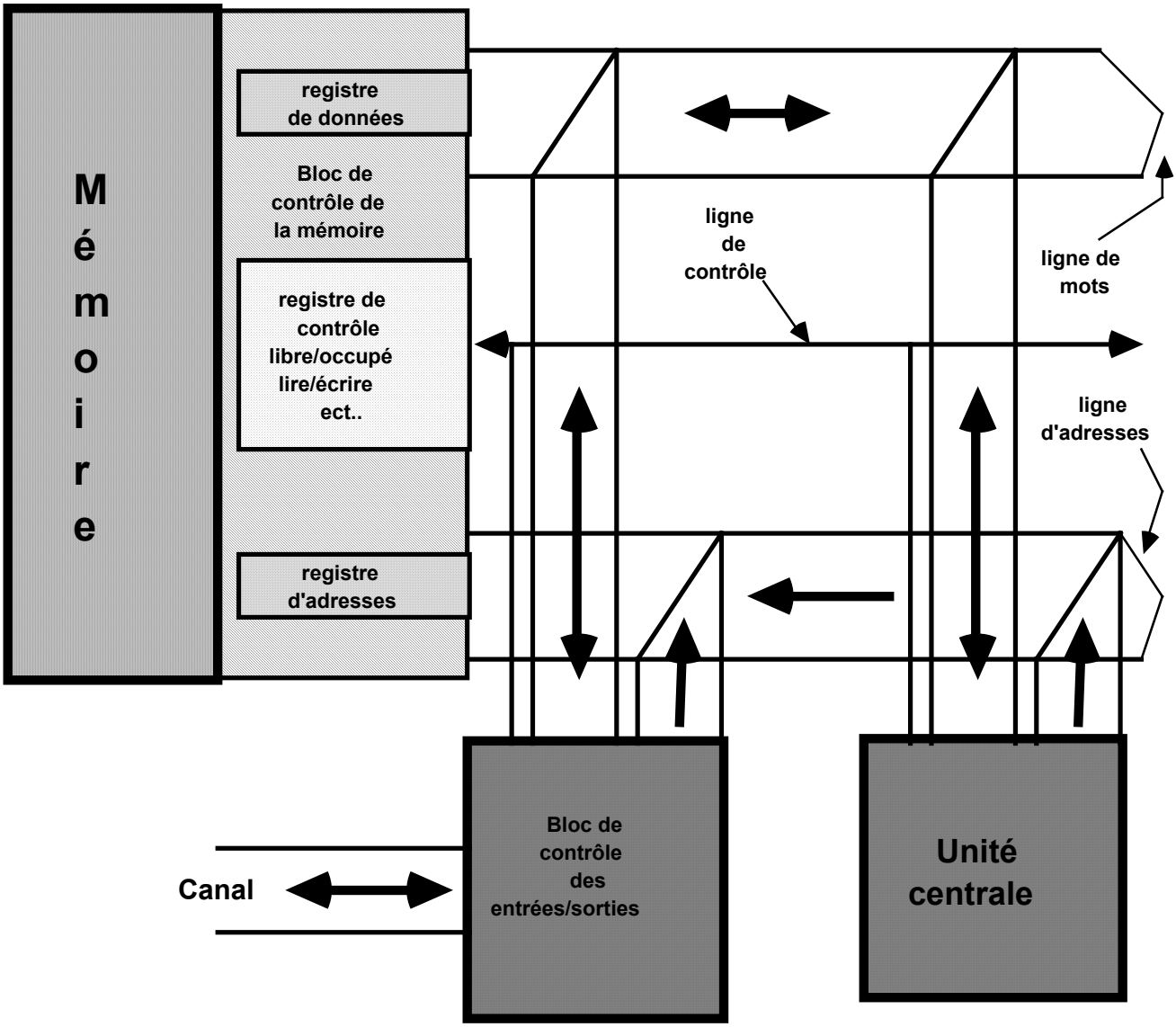


Figure 12


Mémoire
registre de données
Bloc de contrôle de la mémoire
registre de contrôle libre/occupé lire/écrire ect..
registre d'adresses
ligne de contrôle
ligne de mots
ligne d'adresses
Canal
Bloc de contrôle des entrées/sorties
Unité centrale

1.8. GESTION DE LA MÉMOIRE PRINCIPALE

1.8.1 Exigences de la gestion de mémoire

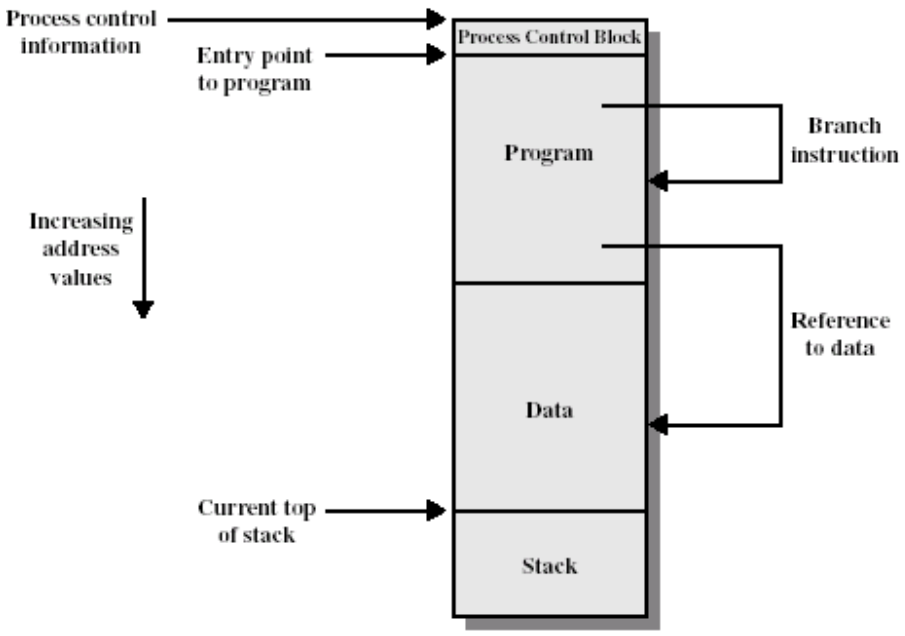
Les mécanisme de gestions de la mémoire doivent être capable de satisfaire à 5 exigences :

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

1.8.1.1 Relocalisation

Dans un systeme qui supporte plusieurs programmes (multiprogramming) 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 I(swap) les différents processus pour pouvoir optimiser l'utilisation de la mémoire et du processeur. Lorsqu'un programme transféré sur un disque doit être recharger 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 sa place initiale. Cette possibilité de relocaliser un programme n'importe où en mémoire nous oblige à gérer l'adressage de notre mémoire

Figure 13 : 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 se trouve. Ainsi lorsque nous aurons

des références aux données ou encore des instructions de branchement à l'intérieur du programme, le système d'exploitation pourra grâce à l'adressage répondre aux demandes.

1.8.1.2 Protection

Chaque programmes 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 mémoire sont vérifiées lors de l'exécution, afin de s'assurer que toutes les références ne sont faites qu'à des espaces réservés à ce programme. Le processeur doit pouvoir arrêter tout programme qui essaye 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 sur toutes les références qu'un programme pourra faire. Il est donc possible de le faire seulement lors de l'exécution de l'instruction.

1.8.1.3 Organisation logique

La mémoire des ordinateurs est généralement organisée de façon linéaire 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 constitués de modules.

Si les systèmes d'exploitation et l'ordinateur peuvent utiliser efficacemen des programmes ou des données sous forme de modules, nous pourrons alors en tirer les avantages suivants :

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

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

1.8.1.4 Organisation Physique

La mémoire est organisé en au moins deux 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 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 primordiale. La responsabilité de ce contrôle pourrait être assigné au programmeur, mais ceci est presque impossible pour deux 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 principale 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 sait pas à l’avance, combien d’espace sera disponible et où elle sera localisée.

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

1.8.2 Partitionnement de la mémoire

La tâche principale de la gestion de la mémoire est de charger des programmes en mémoire pour qu’ils soient exécuté 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 base, **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.

1.8.2.1 Partitionnement fixe (statique)

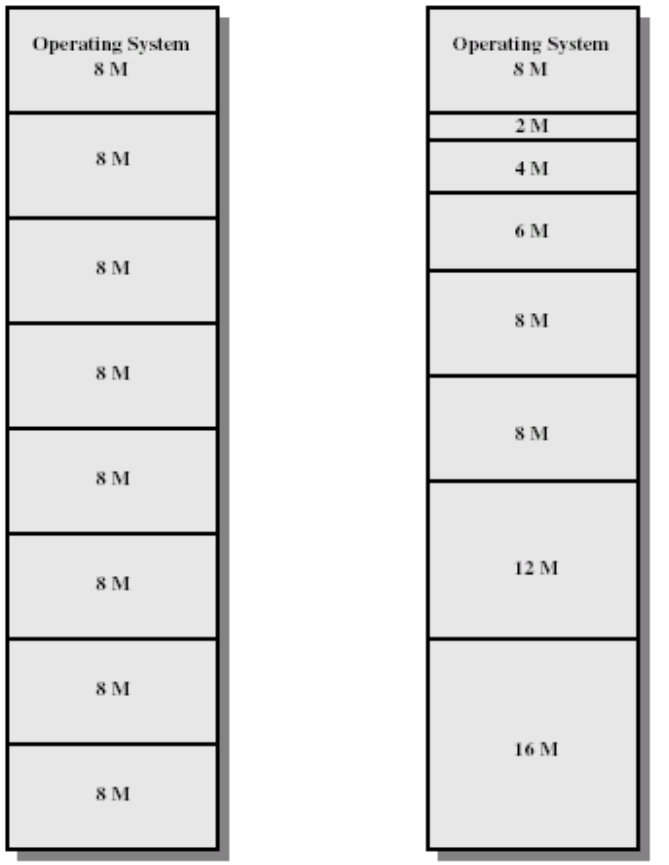
Dans toutes les méthodes de gestion de la mémoire, nous pouvons assumer que les OS occupent une partie fixe de la mémoire principale et que le reste de la mémoire est disponible. Le méthode la plus simpliste est le partitionnement fixe. Il peut existé sous 2 formats, des partitions de même grandeur ou 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 chargé 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 14 nous montre un partitionnement qui vas jusqu'à 16 MB.

Figure 14 : Exemple de partitions fixes



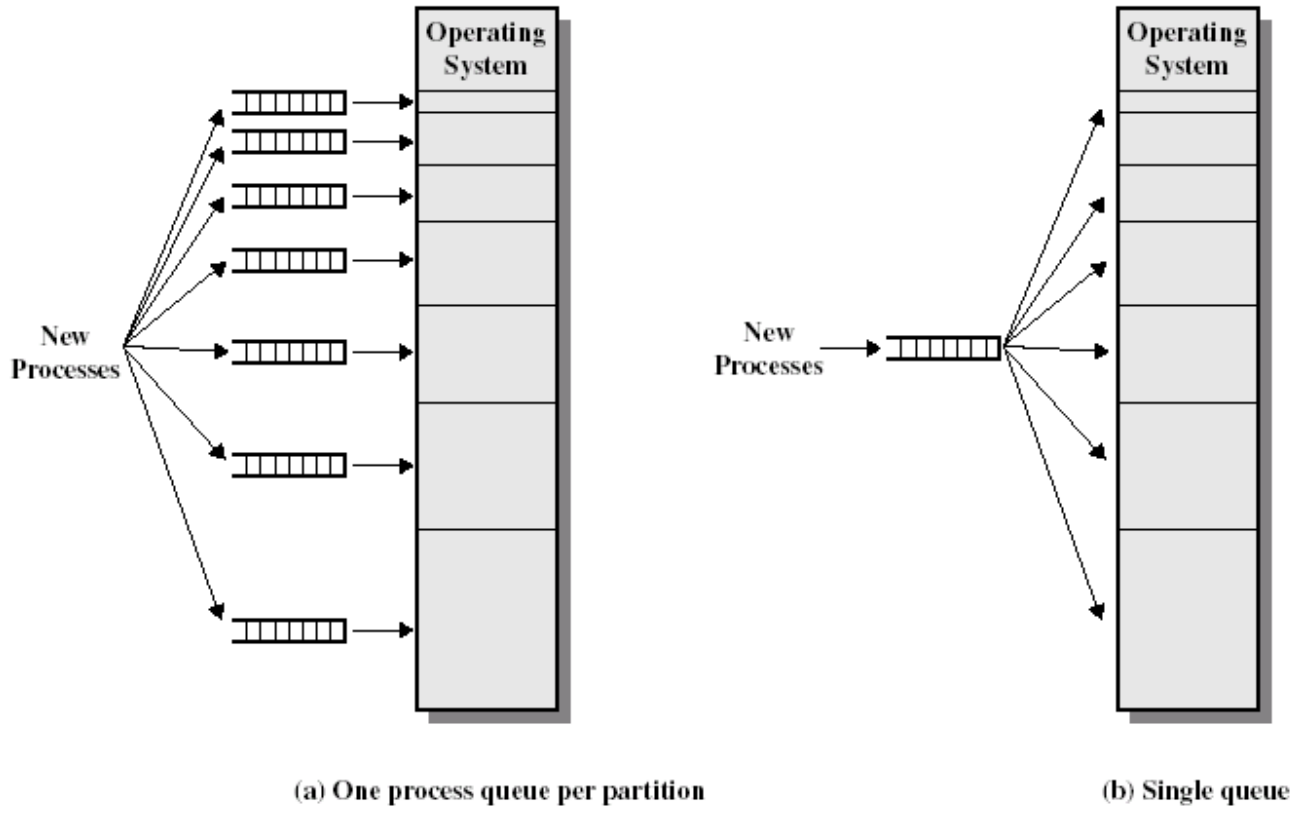
(a) Equal-size partitions (b) Unequal-size partitions

1.8.2.2 Algorithme 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 partitionnement de dimension égale (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 dimensions différentes, plusieurs possibilités s'offrent au système. Nous allons voir que deux méthodes existent : **une file par partition** ou **une file pour tous les partitions**.

Si nous avons une file 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 file de cette partition. Le désavantage est que même s'il reste encore d'autres partitions non utilisées, le système ne va pas les utiliser. Si, en revanche, nous avons une file pour toutes les partitions, le système va essayer de charger le programme dans la partition libre la plus petite. Si toutes les partitions sont prises, une décision sera alors prise sur quel programme devra être déchargé.

Figure 15 : Algorithme de placement, partition fixe

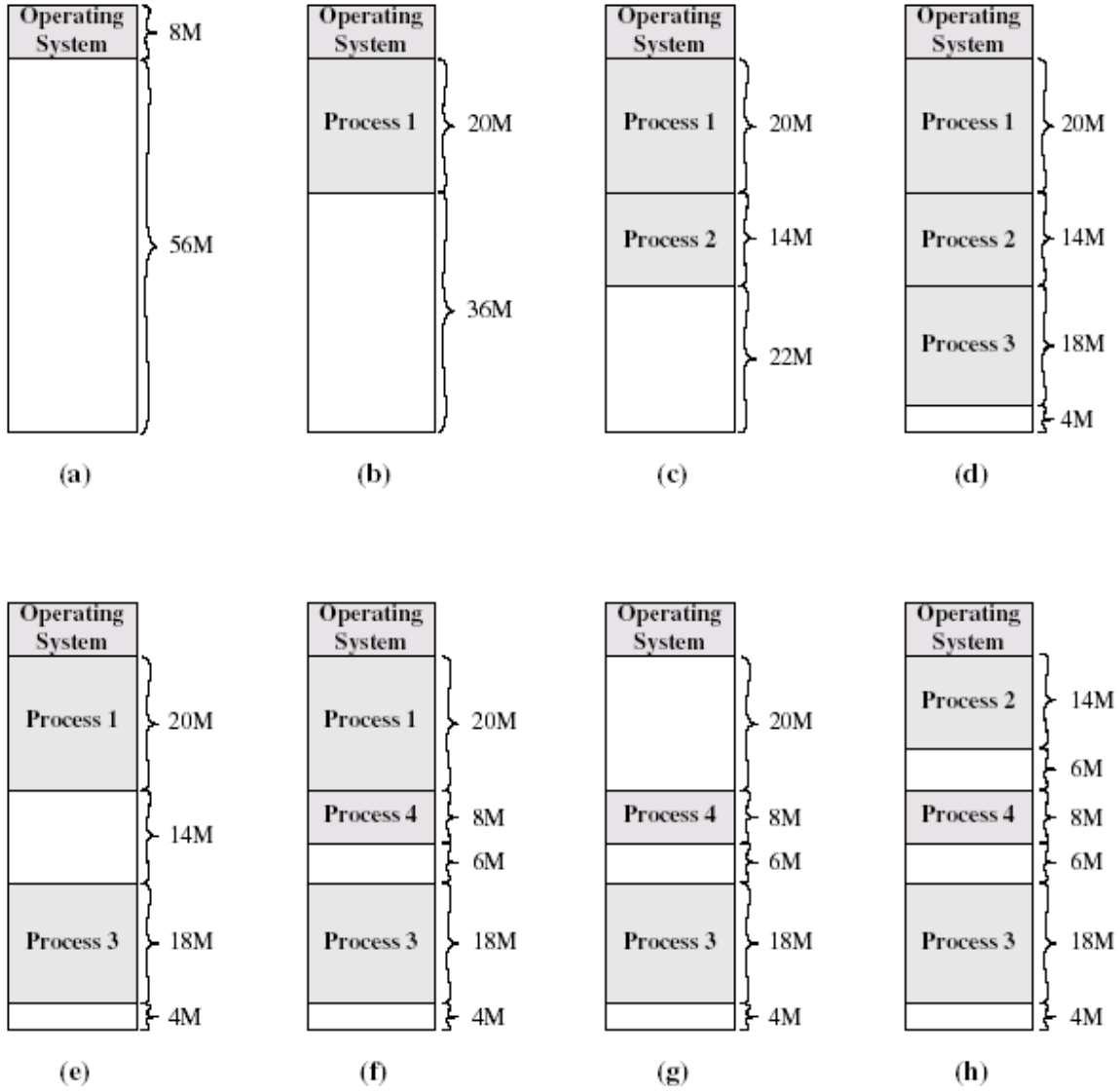


1.8.2.3 Partitionnement dynamique

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

Nous pouvons voir à la figure 16 (ci-dessous), le processus d'allocation dynamique des partitions. Cette technique utilise un nombre variable de partitions de dimensions variables. Quand un programme est chargé en mémoire, on lui alloue l'espace nécessaire. Cette technique n'est plus utilisée de nos jours. Elle a été remplacée par des techniques plus sophistiquées. L'allocation dynamique cause avec le temps ce qu'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**" (algorithme de ramasse-miettes). 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. Notons que la compaction coûte beaucoup de temps processeur.

Figure 16 : Partionnement dynamique



<

1.8.2.4 Algorithme de placement

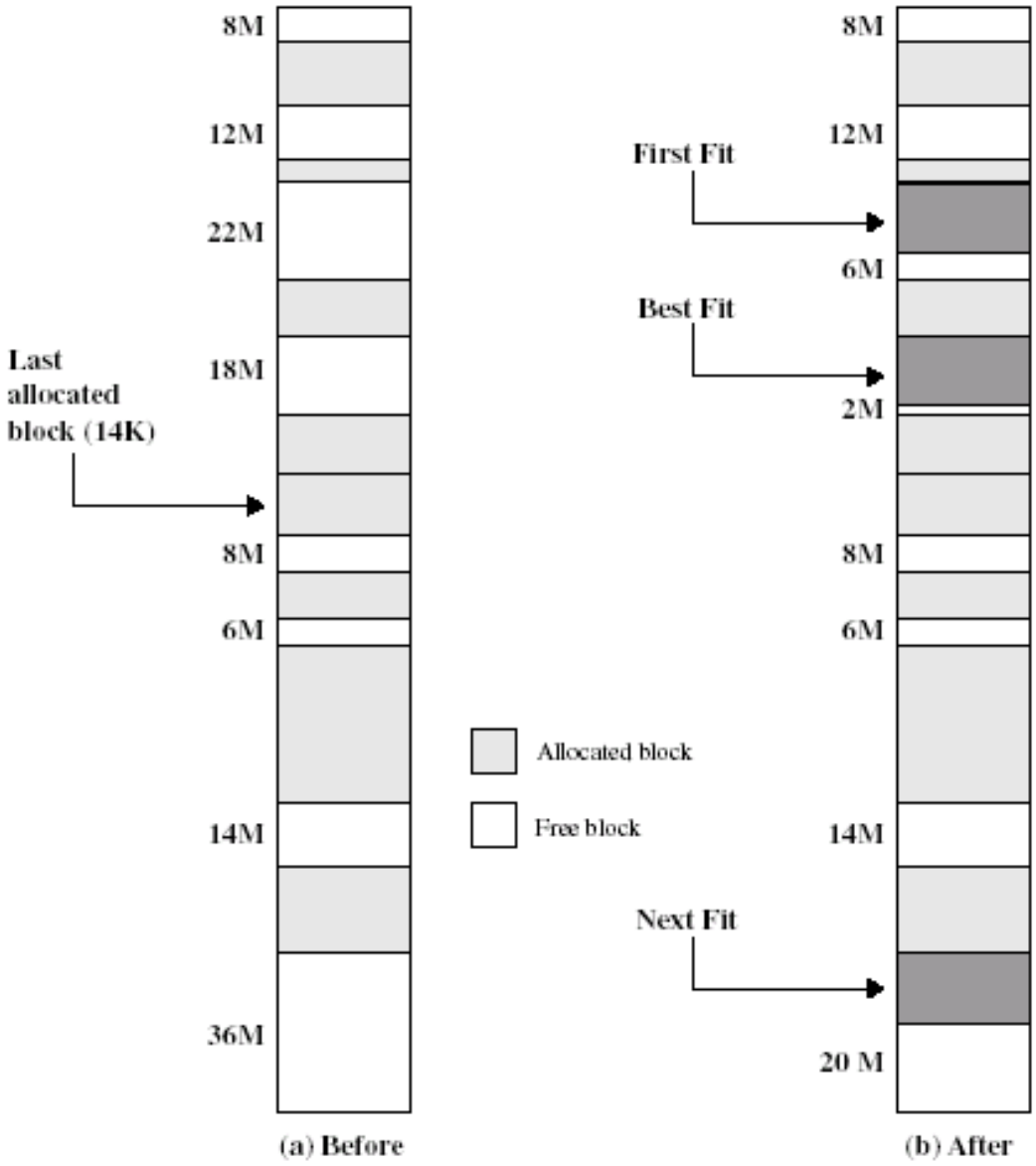
Étant donnée que la compaction de la mémoire nous faire perdre beaucoup de temps CPU, il est donc important d'implanter de bon algorithmes de placement afin de bien remplir les trous. Nous allons voir trois méthodes :

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 3 méthodes.

Figure 17 : Algorithme de placement, partition dynamique



Question : Laquelle de ces 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 fractionnée 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.

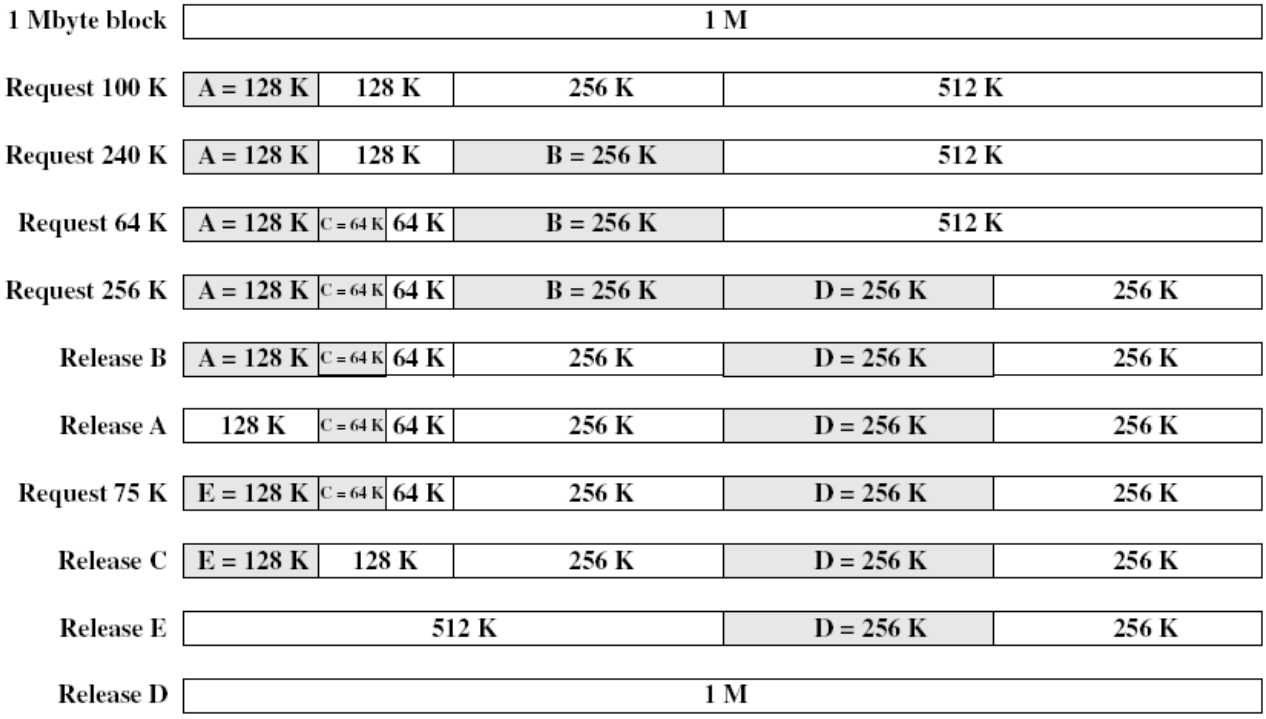
1.8.2.5 Le "Buddy System"

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 vérifie la grandeur et alloue l'espace disponible ou la divise en deux parties égales. Lorsque les blocs se libèrent ils vont être regroupés pour redevenir un seul bloc. L'algorithme est comme suit :

- Débuter avec un seul block de taille 2^U
- Sur une requête pour un block de taille S
    1. si $2^{u-1} < S \leq 2^u$ alors allouer le block entier de taille 2^U
    2. Sinon partager ce block deux compagnons (buddies), chacun de taille 2^{U-1}
    3. Si $2^{u-2} < S \leq 2^{u-1}$ alors allouer un des deux buddies
    4. Sinon diviser diviser un des deux buddies
- Le processus est répété jusqu'à ce que le plus petit block pouvant contenir S soit généré.
- Deux compagnons sont fusionnés lorsqu'ils deviennent tous les deux non allouées.

Voyons cela sur un exemple.

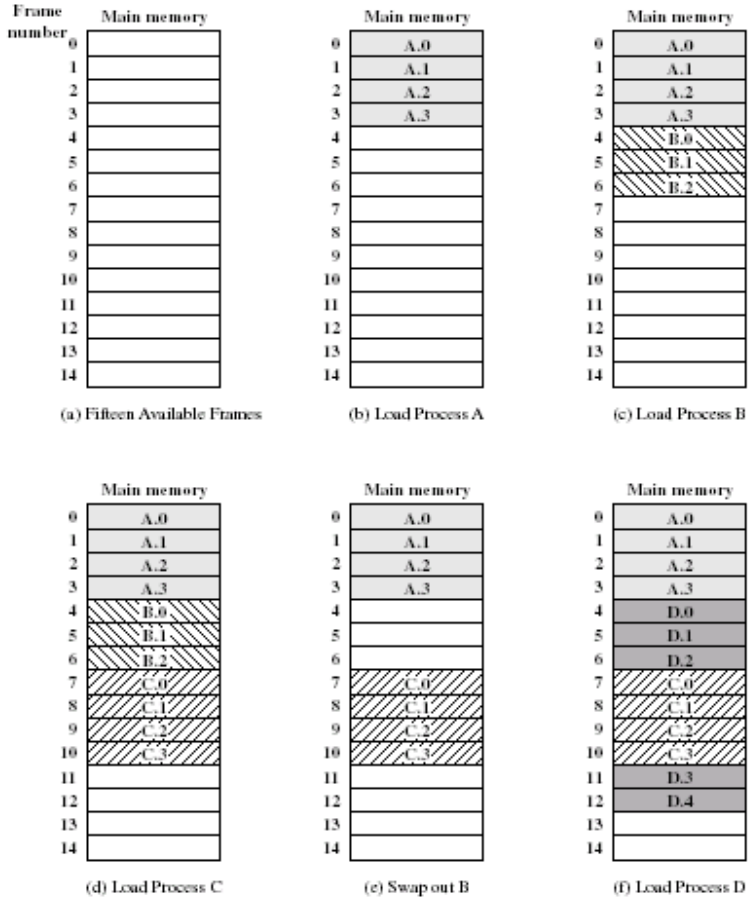
Figure 18 : Buddy System



1.8.3 Pagination

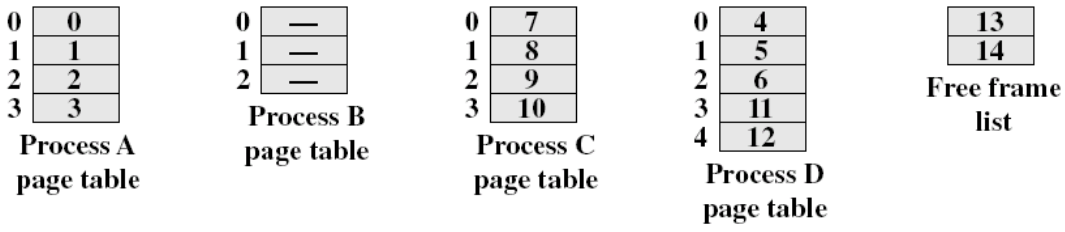
Comme nous venons de voir les techniques de partitionnement fixes ou variables ne gèrent pas efficacement la mémoire, ils causent des problèmes de fragmentation internes ou externes. Nous allons voir maintenant une méthode qui en plus de fractionner la mémoire en petits blocs égales(frames), va fractionner les programmes en petits blocs (pages) de même dimension que les partitions de la mémoire. Le terme frames (cadre en français) est utilisé car un cadre va tenir une page d'information. Nous verrons que grâce à cette méthode, la fragmentation externe est complètement éliminée. La seule fragmentation interne sera causée 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. En utilisant cette méthode nous rendons transparent la pagination aux programmeurs car l'adresse relative est la même que l'adresse absolue.

Figure 19 : 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 ci-dessus que même s'il ne reste pas d'espace continu, 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 20 la table des pages au temps (f). On peut y voir aussi une liste des frames qui sont disponibles.

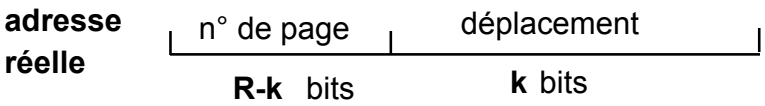
Figure 20 : Tables des pages de la figure 19



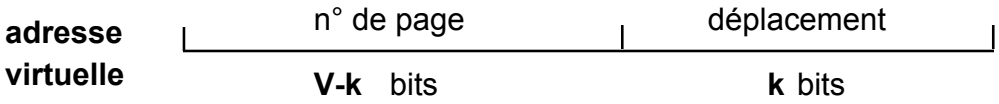
Il existe 2 types de pagination:
- à un niveau
- à plusieurs niveaux (utilisée lorsque l'espace virtuel est très supérieur à l'espace réel).

1. Pagination à 1 niveau

Une adresse (réelle ou virtuelle) peut être scindée en un numéro de **page** et un **déplacement dans la page**. Le déplacement est **invariant** puisqu'il désigne le rang d'un mot par rapport au début de la page (qui est toujours chargée d'un seul tenant). Il n'interviendra donc pas dans la transformation d'adresse.

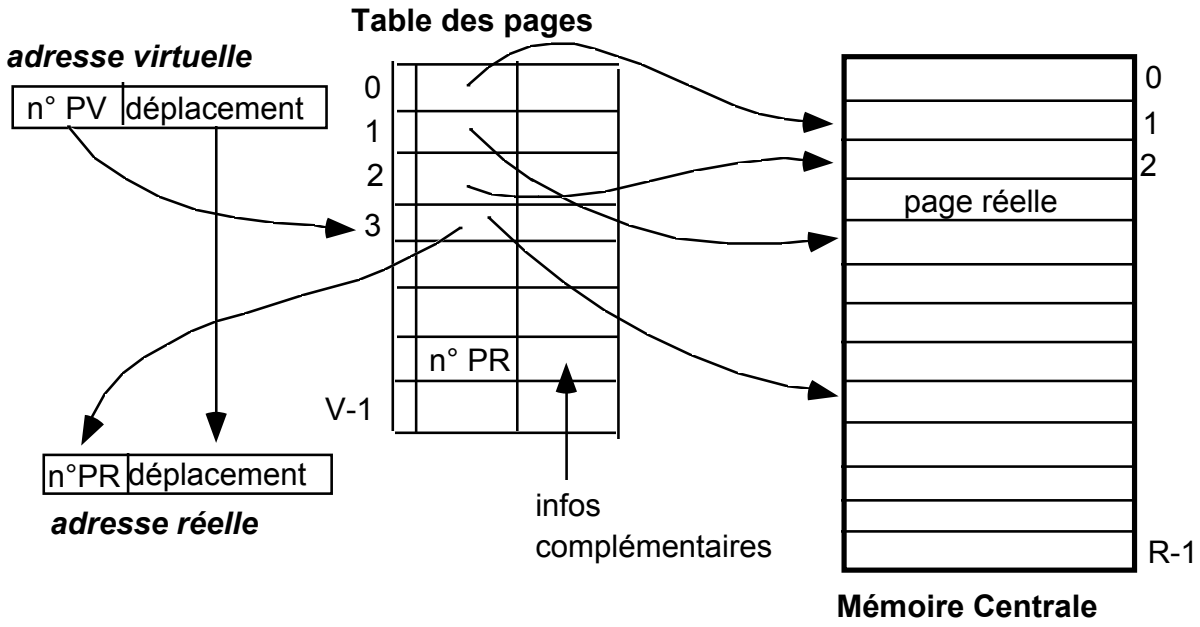


pour un espace réel de 2^R mots et des pages de 2^k mots



pour des espaces virtuels de 2^V mots et des pages de 2^k mots.

Le nombre de pages réelles et virtuelles est connu à la génération du système: la **fonction topographique** sera réalisée par une table appelée **table des pages**. L'implantation de cette table des pages sera faite en MC ou dans des registres rapides car elle sera consultée fréquemment (lors de l'exécution de chaque instruction à référence mémoire).



Informations pouvant se trouver dans une entrée de la table des pages, outre le numéro de page réelle:

- ***bit de présence*** (ou d'invalidité) *I*: positionné, il indique que la page virtuelle référencée n'est pas chargée en MC d'où un déroutement pour **défaut de page**. C'est le premier élément testé par le mécanisme de traduction automatique d'adresse.

- ***bit d'écriture*** *E*: positionné, ce bit indique que la page a été modifiée par une ou plusieurs opérations d'écriture; dans ce cas, la page doit être recopiée sur le disque lors d'un remplacement éventuel.

- ***bit de référence*** *R*: ce bit peut avoir une utilité pour connaître les pages référencées (et donc susceptibles de l'être à nouveau) et celles qui ne l'ont pas été (qu'on peut donc choisir comme victime lors d'un remplacement).

- ***verrou d'accès*** *V*: donne les accès permis dans la page.

exemple: verrou d'accès (2 bits) associé à une page:

00: tout accès permis

01: écriture interdite (lecture et exécution permises)

10: lecture seulement

11: aucun accès permis

La *propriété de localité* doit limiter le nombre de consultations de la table des pages. La propriété de localité est une caractéristique importante de l'exécution des programmes en général. En effet la quasi-totalité des programmes contiennent des boucles d'itérations et/ou utilisent des variables structurées tels

les vecteurs ou les matrices; et si l'on voit un processus comme une suite de références à des instructions ou des données dans un espace adressable linéaire (mémoire virtuelle), on constate que la distribution des références aux pages mémoire *n'est pas uniforme*: pendant un intervalle de temps *T* non négligeable, les références portent sur les mêmes pages.

Cette propriété peut être mise en défaut par certains programmes (plutôt rares) qui n'utilisent que des listes chaînées (dynamiques) dont les éléments sont dispersés dans la mémoire d'allocation dynamique.

La consultation de la table des pages peut provoquer un déroutement pour **défaut de page**. Le traitement de ce déroutement se fera ainsi:

- Chercher une page libre en MC en consultant la **table d'allocation des pages réelles**,
- Si pas de page libre alors faire une réquisition suivant un **algorithme de remplacement**.

2. Traitement des défauts de pages

La consultation de la table des pages peut provoquer un déroutement pour défaut de page. Le traitement de ce déroutement se fait ainsi:

- Chercher une page libre en MC en consultant la table d'allocation des pages réelles
- Si pas de page libre Alors faire une réquisition suivant un algorithme de remplacement
- Mise à jour de la table des pages et des registres associatifs
- Branchement à l'instruction qui a provoqué le déroutement.

3. Les algorithmes de remplacement

Différents critères peuvent être utilisés pour déterminer la page à réquisitionner pour un remplacement; c'est ce qui différencie ces algorithmes.

Random: choisir la page à remplacer (victime) au hasard.

FIFO: remplacer la plus anciennement chargée (nécessité de mémoriser l'ordre de chargement des pages).

NRU (*Not Recently Used*): remplacer une page non récemment référencée; cet algorithme utilise les bits de référence R et d'écriture E de la table des pages. Périodiquement, toutes les 20 ou 30 ms par exemple, le bit de référence de toutes les pages est remis à 0. La victime sera la première page trouvée ayant la plus petite valeur (0 à 3) donnée par les bits RE :

00 = page non récemment référencée et non modifiée,
01 = modifiée et non récemment référencée,
10 = non modifiée mais récemment référencée,
11 = modifiée et récemment référencée.

LRU (*Least Recently Used*): remplacer la moins récemment référencée (nécessité de mémoriser l'instant de la dernière référence pour chaque page réelle). Pratiquement, il serait très coûteux d'associer un registre 32 bits à chaque page et qui serait mis à jour à chaque référence à la page.

LFU (*Least Frequently Used*): remplacer la moins fréquemment référencée (nécessité de mémoriser le nombre de références à chaque page réelle).

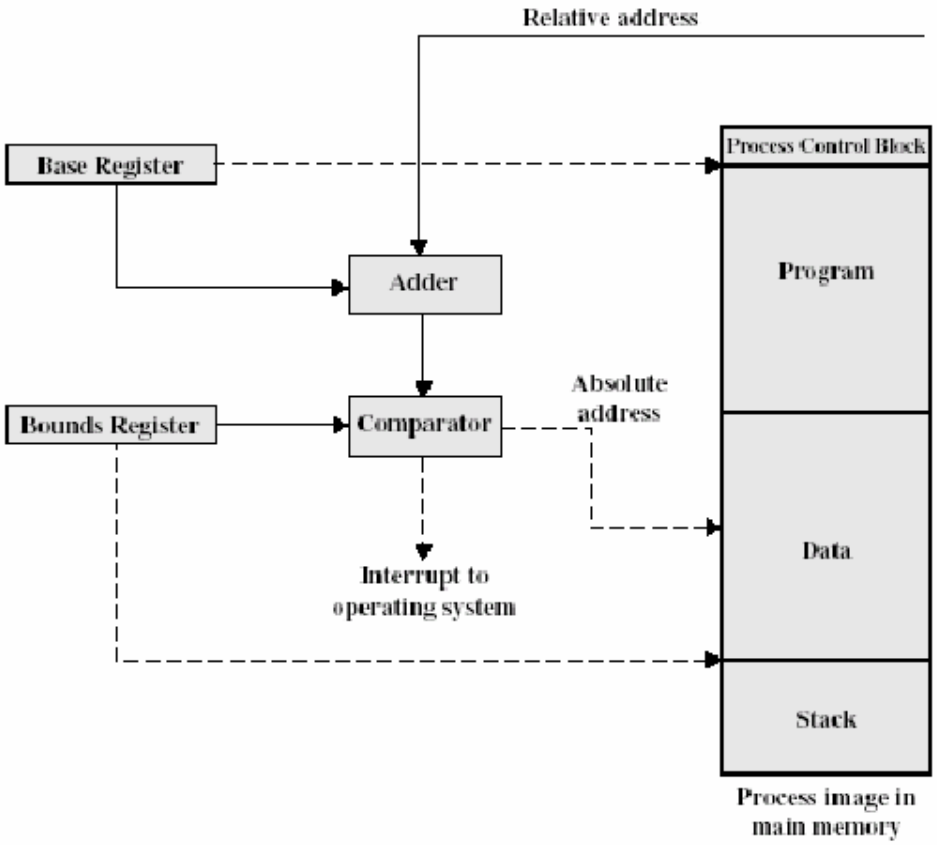
1.9. Les modes d'adressage.

Nous avons vu dans la dernière section que les programmes qui sont chargés 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 saut (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 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 actuelle en mémoire.

1.9.1 Adressage relative

Figure 21 : Adressage Relative.



1.9.1.1 Registre nécessaire à l'adressage relative

Les registres nécessaires pour l'adressage relative sont le registre de base (base register) et le registre de frontière (bounds register). Le registre de base nous donne l'adresse de début où le programme est emmagasiné et le registre de frontière nous donne la fin de l'espace alloué. Ces valeurs sont décidées lorsque le programme est chargé en mémoire.

Pour générer l'adresse, nous additionnons l'adresse contenue dans le registre de base à une adresse relative pour nous donner l'adresse absolue. Cette adresse est maintenant comparée à la valeur limite (frontière) et si celle-ci dépasse la limite fixée par le système un message d'erreur est généré et le programme est interrompu.

1.9.2 Segmentation

Encore une fois, on sépare le programme en petite partie appelée 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 sera constitué de 2 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 une différence majeure entre les 2 : 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 pouvons découper les programmes en petits segments elle devrait être moindre.

1.10. 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. Un autre conséquence des segments inégaux est le manque de relation simple entre l'adresse physique et relative

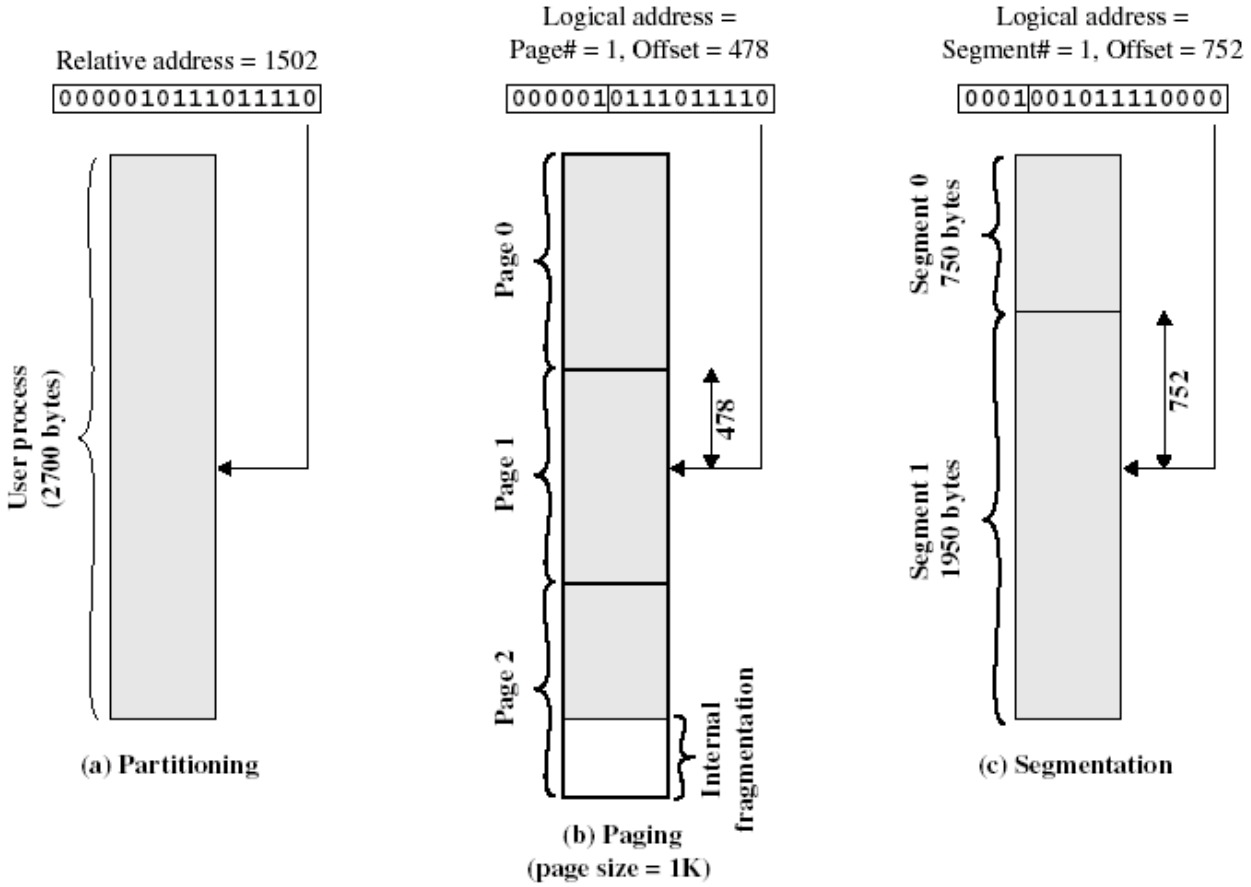
Nous allons voir ici le processus de décodage de la pagination et de la segmentation.

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 nous montre que l'adresse relative de 1502 correspond a 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 22:Adresse logique

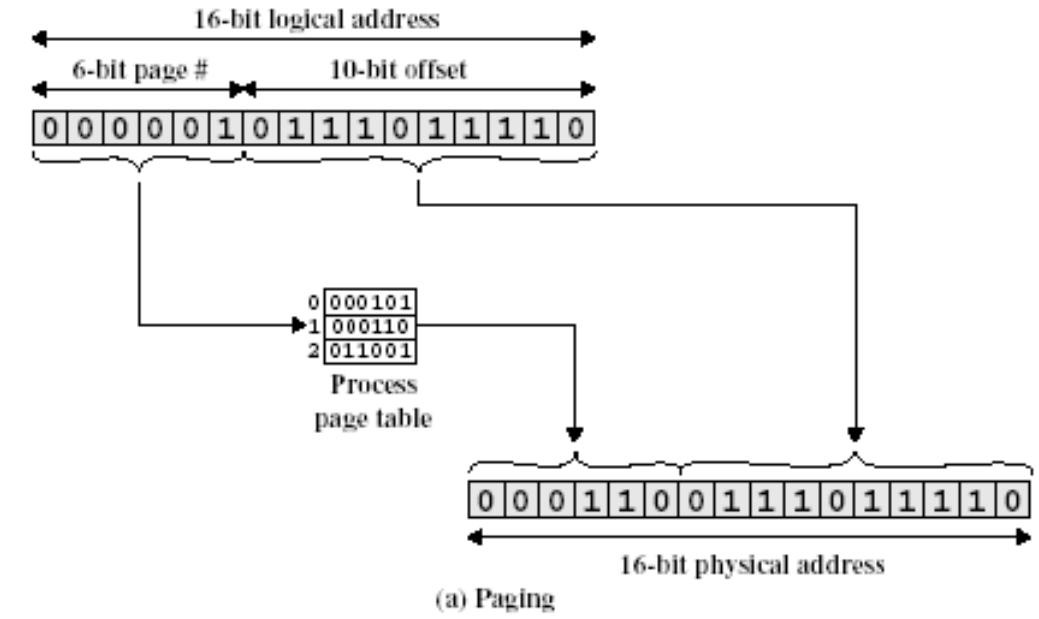


Exemple 50 :

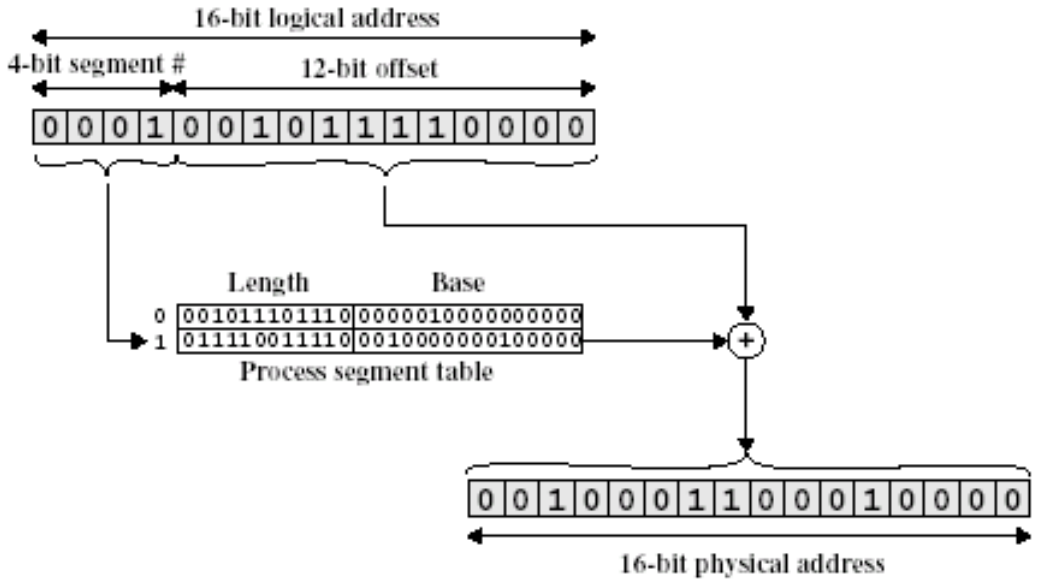
Continuons avec notre exemple. Pour la pagination, si nous savons que la page numéro 1 réside en mémoire au cadre 6 (000110), nous aurons alors physique de 0001100111011110 démontré à la figure 54a.

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 54b.

Figure 23 : Adresse logique vers adresse physique



(a) Paging



(b) Segmentation