IV. Gestion de la mémoire.

1. Introduction.

Nous avons introduit en première année les concepts de base de la gestion de la mémoire. Ce chapitre s'étend sur les méthodes de gestion des mémoires principales et secondaires dans une configuration, qui ont été introduites au début des années 70.

Comme nous l'avons expliqué au début de ce cours, la mémoire d'un système est composée de différents éléments faisant appel à des technologies diverses: mémoire cache rapide, mémoire RAM standard, disques de pagination,.....

La première technique de gestion de la mémoire était appelée "overlay", cette technique consistait à surcharger en mémoire des routines déjà employées. A charge du programmeur de décider quelle routine ne sera plus utilisée dans un futur proche et donc pourra être écrasée. Cette technique a été reprise dans les premières versions de MS-DOS (linker utility LINK).

Ensuite apparut une solution simple et qui fut en vogue au début des années 60, elle consistait à installer suffisamment de mémoire centrale pour contenir tous les programmes en cours d'exécution ou à exécuter prochainement. Les arguments économiques eurent vite raison de cette solution.

Au même moment, Kilburn de l'université de Manchester proposa une méthode de gestion de la mémoire appelée le 'One-level storage', base de la technique actuelle de pagination, qui devint une méthode standard de gestion d'une hiérarchie des mémoires. Cette méthode de gestion avait pour objectif de faire paraître une mémoire centrale RAM et une mémoire secondaire disque comme un seul niveau de mémoire d'où le nom "One level storage". La mémoire virtuelle était née, puisque cette méthode de gestion donne l'impression à l'utilisateur de disposer d'un espace de mémoire continu, uniforme et important.

La segmentation fut ensuite introduite, elle consiste quant à elle à partager l'espace mémoire en segments de taille inégale. Une donnée peut alors être adressée en spécifiant le segment auquel elle appartient et le déplacement au sein de ce segment. Cette méthode de gestion permet la définition d'objets: piles, fichiers,....

Enfin, la pagination fut appliquée aux segments dans les systèmes à segmentation paginée.

2. Dynamic Address Translation (DAT).

La mémoire d'un système contient deux types d'informations: du code machine et des données

L'adresse des données ou des instructions peut être soit réelle soit virtuelle auquel cas l'adresse générée par l'unité centrale subira une traduction dynamique avant d'être envoyée vers les circuits de la mémoire. La traduction dynamique fait appel à des mécanismes hardware dont nous allons expliquer le fonctionnement. Rappelons brièvement qu'elle est basée comme illustré à la figure ci-dessous sur des tables de pages qui contiennent les correspondances entre page de mémoire virtuelle et frame de mémoire réelle.

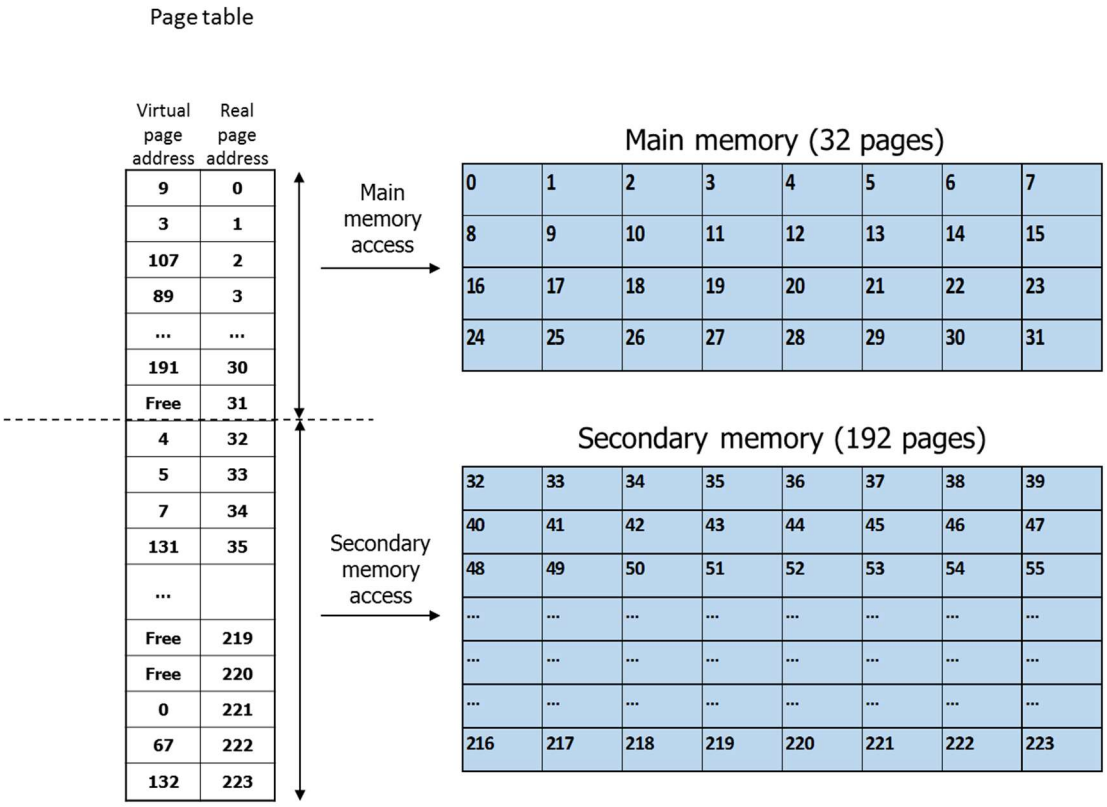


Fig. 41. Correspondance mémoire reélle/mémoire virtuelle.

Si la table des pages est maintenue en mémoire centrale, un accès utile à la mémoire demande deux références ce qui constitue un *overhead* insupportable. Ces tables doivent donc être stockées dans des mémoires rapides qui sont en général chères et dont on ne peut disposer en grande quantité.

Il existe 3 techniques de traduction implémentées via des circuits ou du microcode :

- le mapping direct;
- le mapping associatif;
- le mapping set-associatif;

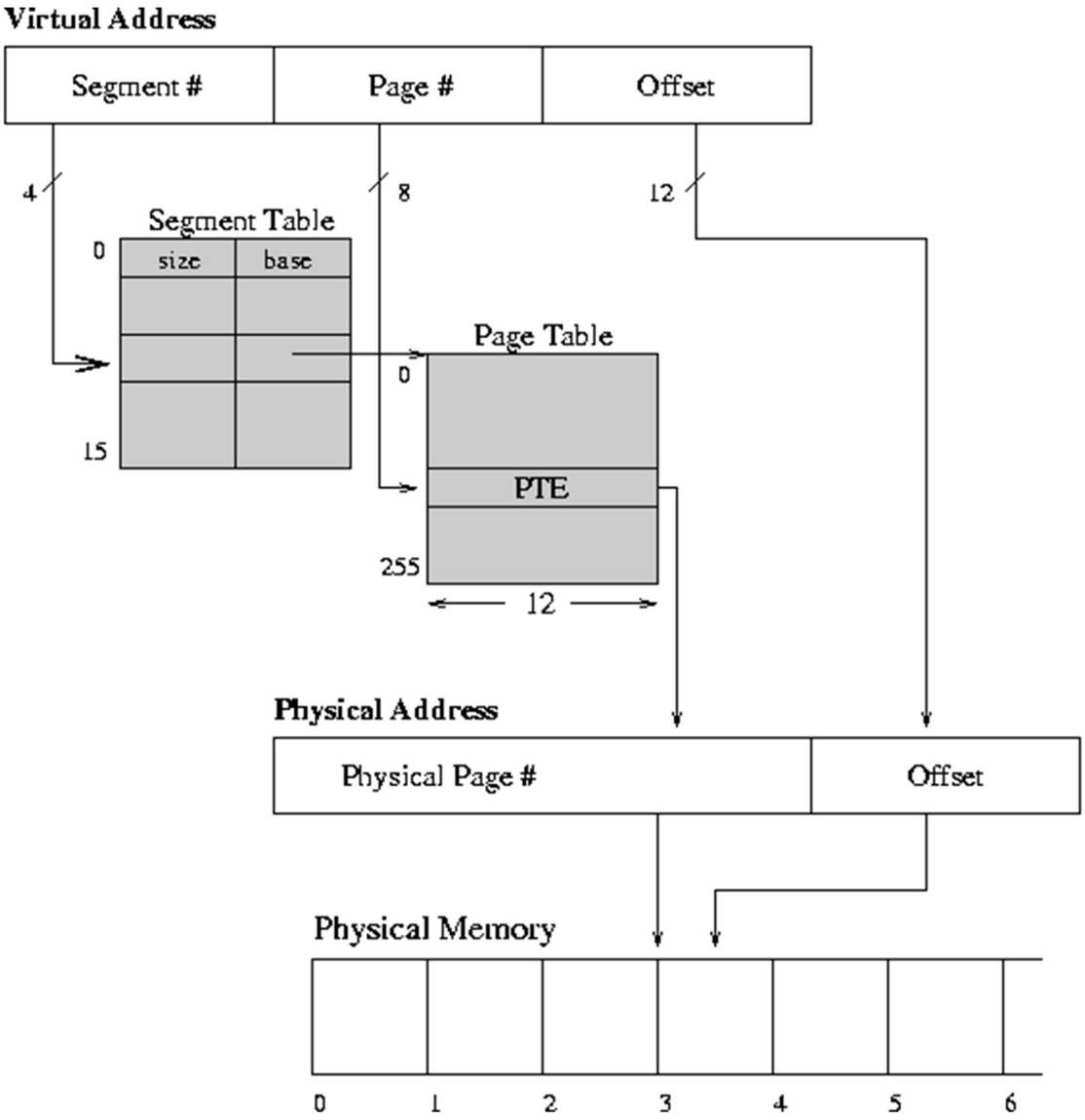


Fig. 42. Dynamic Address Translation.

Rappelons que dans tous les cas, seul le numéro de page est traduit en numéro de bloc ou frame, le déplacement au sein d'une page est égal au déplacement au sein d'un bloc.

A. Mapping direct

Toutes les adresses des pages de la mémoire réelle sont stockées dans une mémoire RAM rapide à un endroit dont l'adresse correspond à la mémoire virtuelle. En conséquence, l'adresse d'un frame de mémoire réelle peut être obtenue directement en consultant cette mémoire à l'adresse correspondant au numéro de la page de mémoire virtuelle.

Cette technique ne peut être utilisée que si le nombre de pages de la mémoire virtuelle est faible, il faut en effet une entrée pour chaque page de la mémoire virtuelle même si cette dernière n'est pas présente en mémoire.

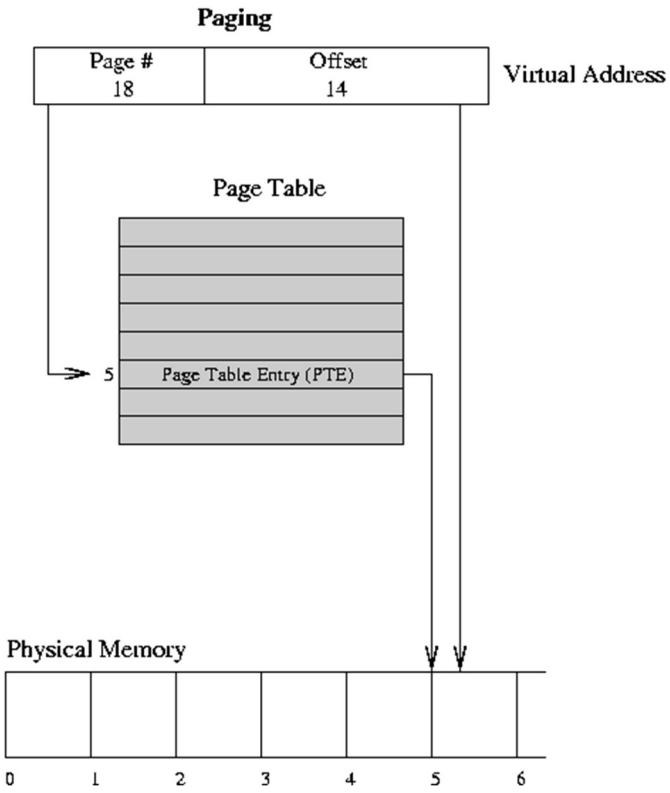


Fig. 43. Mapping direct.

B. Associative mapping

L'adresse virtuelle et l'adresse réelle sont stockées toutes deux dans une mémoire rapide. L'adresse virtuelle est comparée simultanément à toutes les adresses stockées, si une correspondance est trouvée, la page réelle est lue. Chaque entrée demande un comparateur.

Un type particulier de mémoire peut être employé pour cet usage, il s'agit de mémoire CAM (*Content Addressable Memory*), ces mémoires incluent des comparateurs. Un endroit de la mémoire est identifié par son contenu plutôt que par une adresse assignée. Ces mémoires sont associées à de la mémoire RAM classique, la CAM contient l'adresse virtuelle, la RAM contient l'adresse réelle.

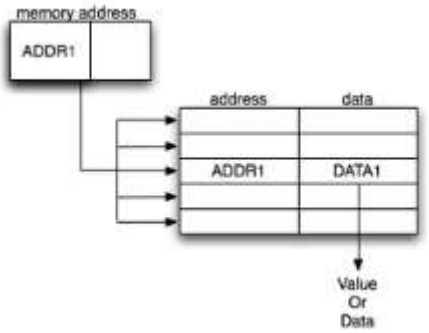


Fig. 44. Mémoire de type CAM.

Lorsqu'une adresse virtuelle est présentée, la table est parcourue en une fois, en utilisant les comparateurs (une comparaison séquentielle serait trop lente). Si une correspondance est trouvée, l'adresse réelle est tirée de la mémoire RAM, sinon la condition dite de *Page Fault* survient et une routine software est activée pour traiter cette condition.

Ces mémoires sont onéreuses et leur degré de complexité croît exponentiellement en fonction de leur taille.

C. Set-Associative mapping.

Cette technique est une combinaison des deux techniques précédentes. Il s'agit d'un artifice qui permet de tirer avantages des deux techniques tout en en limitant les coûts liés au degré de complexité pour les mémoires CAM et à la taille pour les mémoires RAM.

Les adresses sont décomposées en un *Tag,* un index (*set*) et une *line* (*offset*). Le numéro de page correspondant comme dans les autres cas aux deux premiers champs (*index* et *tag*) concaténés.

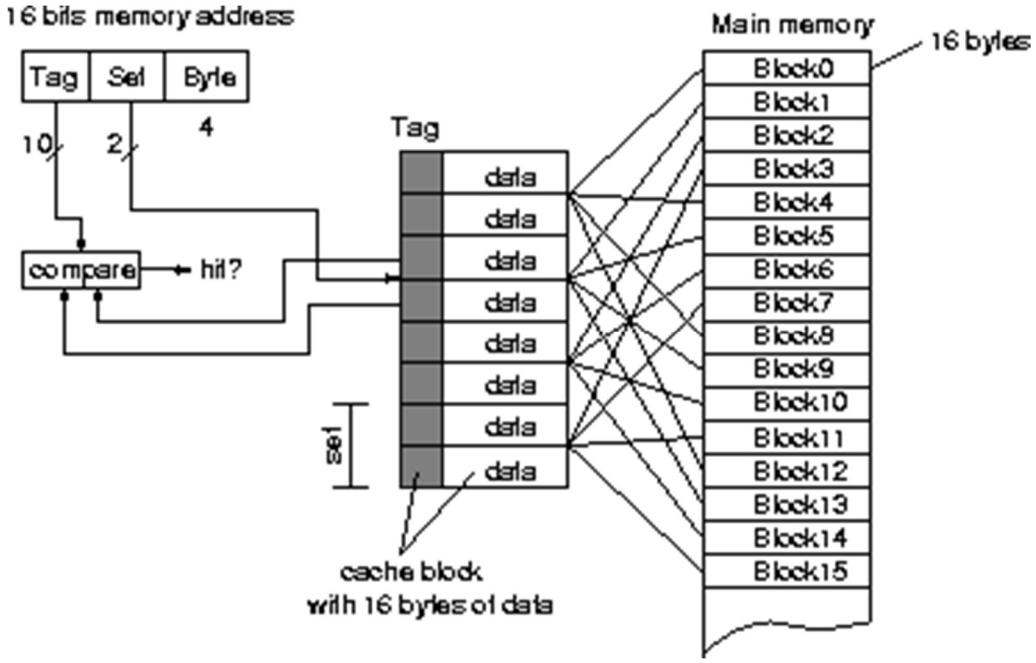


Fig. 45. Set associative mapping (2-way set-associative, 4 sets, 16byte blocks)

La mémoire RAM est organisée en blocs d'une taille de 2**I cellules, avec I le nombre de bit de la partie index. Chacune de ces cellules contient la paire *tag*/adresse réelle. Les blocs de mémoire RAM sont arrangés de telle sorte que les paires appartenant à chacun des blocs sont accédés simultanément. Ces blocs sont adressés suivant le principe du mapping direct.

Le *tag* de la mémoire virtuelle est comparé avec ceux trouvés à l'index indiqué ; si une correspondance est trouvée, l'adresse réelle est transmise. Cette comparaison simultanée fait appel au principe des mémoires CAM. Le nombre d'entrées qui sont comparées s'appelle le "set-size" ou "s-way". Le schéma illustre une technique 2-way.

D. Translation lookaside buffer

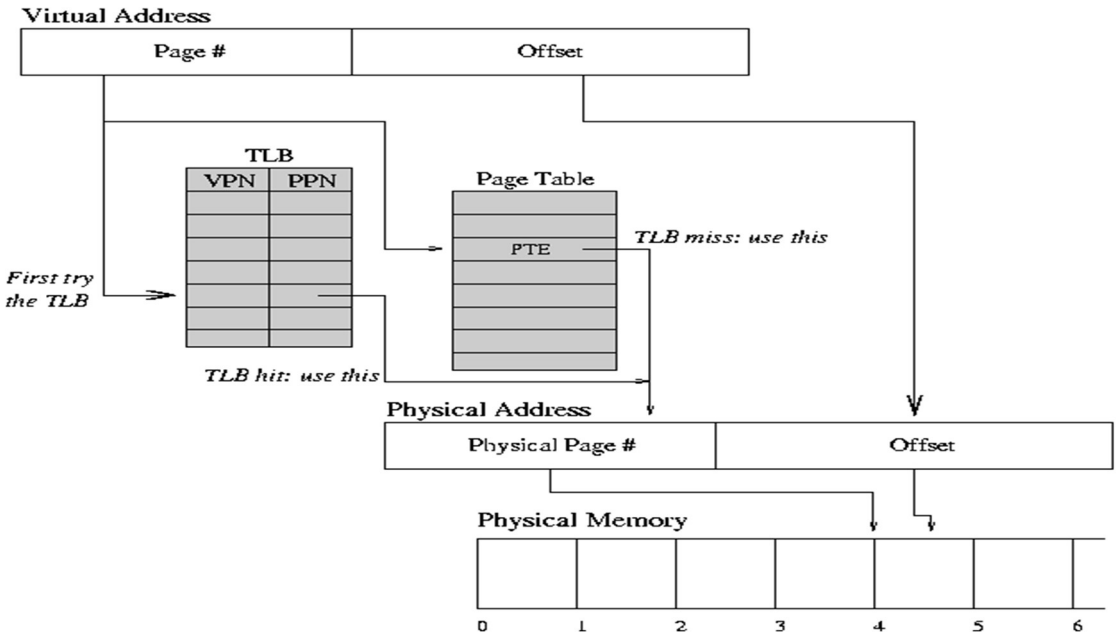


Fig. 46. Translation look-aside buffer.

En pratique, le nombre de pages présentes dans une machine moderne rend les techniques *direct* et *set-associative* difficiles à implémenter totalement grâce au *hardware*. Vu le principe de localisation, un nombre restreint de pages vont être souvent référencées à moins qu'un changement de contexte n'intervienne. Si nous nous reportons à la théorie sur la hiérarchie des mémoires, les adresses des pages les plus souvent accédées peuvent être traduites par le *hardware*, les autres peuvent être traduites par consultation d'une table placée en mémoire centrale sans impacter profondément les performances du système.

Le composant *hardware* contenant les adresses des pages les plus référencées est appelé TLB: *Translation lookaside buffer (directory lookaside*, *address translation cache*). Le TLB ne contient pas l'ensemble des tables de pages, il existe donc un algorithme de remplacement pour les pages non traduites par le TLB activé en cas de *Miss*. Cet algorithme est soit microcodé soit implémenté via les circuits. Certaines parties du TLB peuvent être dédicacées aux pages attribuées à l'O/S afin de privilégier ce dernier.

E. Hashed page tables and inverted page tables

(cité pour mémoire, non développé ici)

3. Algorithmes de remplacement de pages.

A. Introduction.

Une condition de Page Fault survient lorsqu'une page référencée n'est pas présente en mémoire centrale c.à.d. lorsque qu'aucune entrée valide n'est trouvée dans les tables de pages lors du processus de traduction dynamique des adresses virtuelles.

Lorsque cette condition apparaît, la page recherchée doit être localisée en mémoire secondaire et une page de la mémoire centrale doit être choisie afin de faire, si nécessaire, de la place en mémoire centrale. Le système d'exploitation met ici en œuvre un algorithme de remplacement des pages. Les transferts entre mémoire centrale et secondaire peuvent être lents, le processus demandeur de la page peut être suspendu durant le temps du transfert, le système sélectionnera alors un autre processus en attente d'activation.

Deux grands types d'algorithmes existent afin de choisir la ou les pages à remplacer en mémoire centrale, ils sont basés ou non sur la fréquence de référence. Les algorithmes tenant compte de la fréquence de référence des pages peuvent s'appuyer sur un dispositif hardware capable d'enregistrer ces références.

La méthode la plus simple consiste à maintenir un *reference bit*[5] associé à chaque entrée d'une *page table*. Ces bits sont ensuite lus par le système d'exploitation afin de déterminer si la page a été utilisée. Chaque lecture par le système d'exploitation provoque une remise à zéro (*reset)* du bit.

Afin de mesurer l'utilisation de la mémoire, il suffit de parcourir régulièrement les entrées de la table des pages; la fréquence de déclenchement de cet algorithme doit être soigneusement choisie afin de ne pas provoquer un *overhead* trop important. Un dispositif *hardware* peut être introduit pour compter le nombre de références et ainsi limiter la nécessité de scanner les reference bits à intervalles rapprochés.

A côté de ce reference bit, on trouve, la plupart du temps, un bit appelé *changed bit*[6]. Ce bit est positionné si une modification du contenu de la page survient. Il est employé par l'algorithme de pagination afin de décoder s'il est nécessaire de sauver en mémoire secondaire le contenu d'un frame au moyen d'une opération de page-out.

Dans certains systèmes, on trouve un *unused bit* qui indique si la page a été employée depuis qu'elle a été chargée de la mémoire secondaire vers la mémoire centrale afin d'éviter de l'enlever de la mémoire centrale avant qu'elle ne soit utilisée.

[5] ou « used » ou « accessed » bit
[6] ou « modified » ou « written » ou « dirty » bit.

On peut adopter une seconde classification des algorithmes de pagination basée sur l'étude que font ces algorithmes du comportement des processus présents en mémoire. Un algorithme global tient compte de l'ensemble de l'occupation de la mémoire sans tenir compte des caractéristiques des processus occupant les pages. Un algorithme local tient compte uniquement des pages attribuées au processus. Cet ensemble de pages est appelé le *working set*.

En général, les algorithmes locaux sont plus efficaces que les algorithmes globaux dans un environnement en multiprogrammation : ces derniers ignorent le working set des processus et peuvent donc retirer de la mémoire centrale une série de pages associées à des processus qui seront exécutés prochainement. Ces algorithmes peuvent conduire à la condition de *thrashing* (emballement) qui consiste en un transfert excessif de pages dans un environnement en multiprogrammation où la mémoire est fortement sollicitée.

La pagination se définit comme le transfert de pages de mémoire virtuelle entre la mémoire centrale et les mémoires secondaires. Comme nous l'avons expliqué, une page de mémoire virtuelle ne peut être accédée par un processus que si elle est présente en mémoire centrale. Sinon, cette page est transférée d'une des mémoires auxiliaires vers la mémoire centrale. Cette page peut traverser plusieurs niveaux de la hiérarchie des mémoires lors de son trajet vers la mémoire centrale.

Les transferts sont effectués dans les deux sens, des mémoires auxiliaires vers la mémoire centrale (*page-in*), ou de la mémoire centrale vers les mémoires auxiliaires (*page-out*). Ils sont gérés par les composants du système d'exploitation en charge de la gestion des mémoires : le Real Storage Manager pour la gestion de la mémoire réelle et l'Auxiliary storage Manager pour la gestion des mémoires auxiliaires. Ces composants sont activés en réaction à une interruption déclenchée par le hardware lorsqu'il rencontre la condition de Page Fault.

Trois grandes fonctions doivent être assurées lorsque le système d'exploitation traite ce type d'interruption:

1. Déterminer quelles pages présentes en mémoire centrale doivent être transférées vers les mémoires auxiliaires. On parle de *replacement policy*.
2. Déterminer le moment du chargement des pages en mémoire centrale. On parle de *fetch policy*.
3. Déterminer l'endroit où placer ces pages en mémoires centrales, on parle de *placement policy*.

La "*fetch policy*" que l'on rencontre le plus souvent est appelée pagination à la demande (Demand paging). Elle consiste à attendre que la condition de page fault survienne pour démarrer le transfert. Une autre politique consiste à transférer des pages en mémoire avant qu'elles ne soient utilisées, il s'agit ici de pre-fetch. Cette dernière politique peut se révéler plus efficace que d'autres, nous verrons dans la suite de ce cours des éléments nous permettant de juger de l'efficacité de ces politiques.

Le choix d'un endroit en mémoire centrale pour y stocker les pages de mémoire virtuelle entrantes peut paraître évident à partir du moment où des pages à remplacer ont été choisies. Certains systèmes peuvent cependant décider de conserver une certaine quantité d'espace libre contigu dans la mémoire réelle en vue de préparer un partitionnement de cette mémoire[7].

La *replacement policy*, soit la méthode de choix d'une page ou de plusieurs pages à remplacer, constitue cependant la caractéristique principale des algorithmes de pagination. Nous allons passer en revue le fonctionnement de quelques-uns de ces algorithmes puis nous nous attarderons sur leur efficacité.

B. Algorithme de remplacement de pages.

a) Random.

La plus simple, le moins efficace, une page est choisie au hasard, il n'est pas tenu compte d'une quelconque relation entre la page et son utilisation. Cet algorithme ne tient pas compte du principe (bien connu) de localisation des références. La génération des numéros de pages est basée sur un nombre pseudo-aléatoire ou en comptant le nombre d'occurrences d'un certain événement (timer). Le TLB du VAX 11/780 ainsi que le processeur Intel i860 employaient ce type d'algorithme.

b) First-in First-out (FIFO)

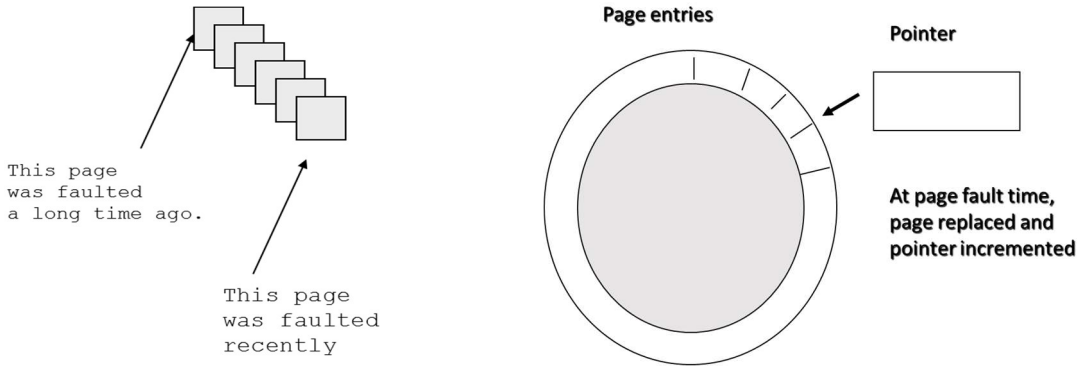


Fig. 47. FIFO : liste de longueur fixe ou liste circulaire.

La page la plus vieille en mémoire centrale est choisie au moment du page fault. Cet algorithme est global, il peut être implémenté comme indiqué à la figure précédente comme un algorithme de gestion de liste circulaire: à chaque page fault, le pointeur avance d'un cran dans la liste.

[7] On partitionne la mémoire d'une machine en vue de faire gérer cette partition par une instance d'un système d'exploitation. Une même machine physique complète (CPU, mémoire et canaux) peut ainsi être partitionnée, chaque partition étant gérée par une instance du système d'exploitation. Les partitions sont indépendantes les unes des autres, il est cependant possible de démarrer un processus de reconfiguration dynamique qui reprend de la mémoire d'une partition pour l'attribuer à une autre, d'où la nécessité de réfléchir à l'occupation de la mémoire réelle.

Aucun dispositif hardware n'est nécessaire pour maintenir l'âge de la page en mémoire centrale, il suffit d'inspecter la liste. Cet algorithme suppose que les processus réclament des pages de manière purement séquentielle, ce qui est loin d'être le cas en réalité.

c) Clock based – Not Recently Used (NRU)

L'algorithme précédent peut être amélioré en évitant de transférer les pages référencées récemment. Cet algorithme est appelé FINUFO pour First In not used First out, ou encore NRU pour Not Recently Used. Il doit inspecter le reference bit d'une page avant de décider de la remplacer. Si une référence est détectée, la page suivante est inspectée.

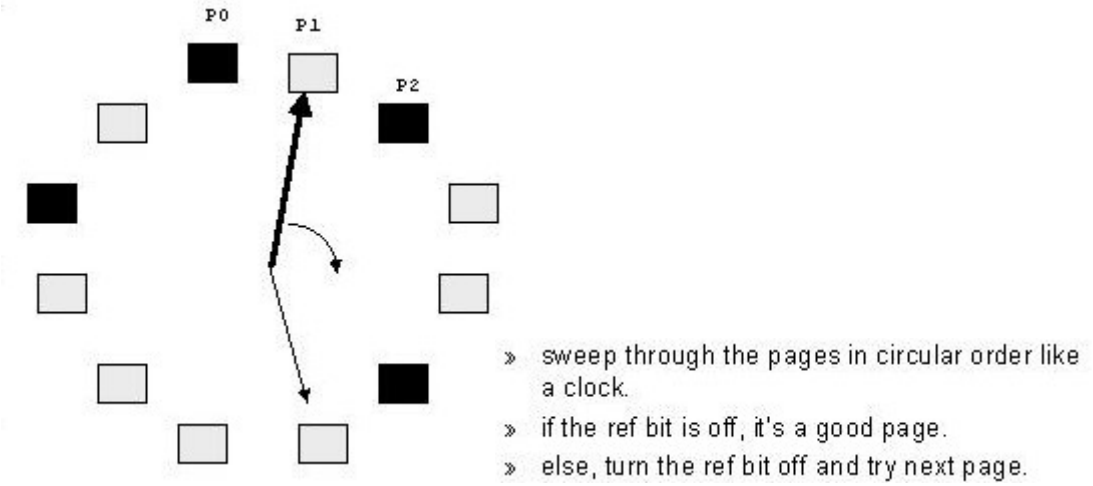


Fig. 48. Clock-based replacement algorithm.

d) Least Recently Used (LRU)

Suivant cet algorithme, la page éligible pour être remplacée au moment du page fault est la page qui a été référencée depuis le plus long temps. Il s'agit d'un algorithme placé dans la catégorie des "Stack Algorithm", nous reviendrons plus tard sur cette notion.

L'algorithme LRU pose des problèmes d'implémentation si un grand nombre de pages sont présentes en mémoire centrale puisqu'il exige l'enregistrement d'une référence à chaque fois qu'un programme accède une page de mémoire. Un compteur peut ainsi être associé à chaque page de mémoire virtuelle et être incrémenté à intervalles réguliers. A chaque référence, le compteur est remis à zéro, ainsi, la page de mémoire virtuelle possédant le compteur le plus élevé sera la page choisie par l'algorithme de remplacement des pages. Cette technique est relativement complexe à mettre en œuvre, des circuits spécifiques doivent maintenir des compteurs alors que l'information essentielle n'est pas la valeur du compteur mais l'ordre des pages: seule la ou les pages en fin de liste sont éligibles pour un remplacement.

Une approximation courante de l'algorithme LRU utilise le reference bit maintenu par le hardware. A intervalles réguliers, tous les reference bits sont inspectés par le système

d'exploitation et remis à zéro. Un enregistrement du nombre de fois que ce bit est positionné pour une page donne une approximation du nombre de références subies par cette page. Cette valeur (appelée Unreferenced Interval Count) permet d'ordonner la liste des pages en fonction de leur fréquence d'usage. Cet algorithme est donc appelé « Not Frequently Used » (NFU) ou « Least Frequently Used » (LFU) selon les details de l'implémentation. Ces derniers algorithmes réagissent cependant mal à un changement total de contexte, et des mécanismes d'oubli (« *aging* ») des plus vieux accès est nécessaire.

e) Working Set.

Le working set w(T,t) à l'instant t se définit comme l'ensemble des pages référencées par un processus durant l'intervalle de temps (t-T, t). Le working set doit inclure l'ensemble des pages nécessaires au fonctionnement d'un processus. Il s'agit d'une fonction croissante avec T.

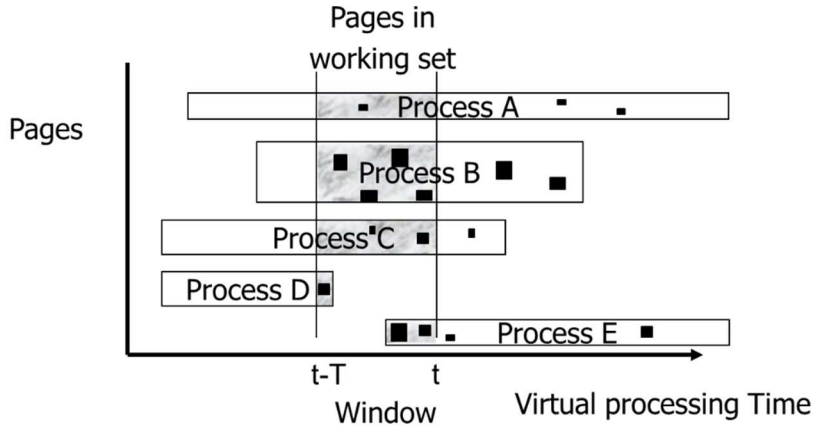


Fig. 49. Le working set

L'algorithme dit Working set algorithm illustré à la figure précédente, remplace les pages qui n'ont pas été référencées par un processus dans l'intervalle T précédent.

Les pages sont ajoutées au working set au fur et à mesure des pages faults. Le nombre de pages maximales référencées dans l'intervalle de temps est appelé "window size" ; il est évidemment variable en fonction du temps. Un processus qui montre une bonne localisation des références aura une window fort étroite.

Cet algorithme est local, il est basé sur le comportement d'un programme, il est très efficace car il permet de ne conserver en mémoire que les pages réellement nécessaires aux besoins d'un processus.

C. Performances et coûts.

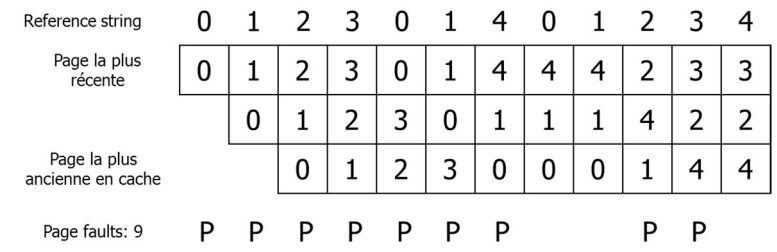
Les performances et le coût (en terme d'instructions exécutées) sont les critères permettant de faire le choix d'un algorithme de remplacement de pages. Il est clair que l'algorithme random ne coûte rien mais qu'il offre de piètres performances. Tout est donc comme à chaque fois une question de compromis.

L'algorithme de remplacement optimal pourrait être défini comme l'algorithme qui produit le nombre minimum de pages faults. Cet algorithme a été décrit en 1966 par Belady et est connu comme OPT (pour optimal) ou MIN (pour minimum page fault). La description de Belady est purement théorique, ces algorithmes ne peuvent être implémentés en pratique au vu de leur coût élevé. Ces descriptions théoriques servent principalement de base de comparaison afin de réaliser des benchmarks d'algorithmes réels.

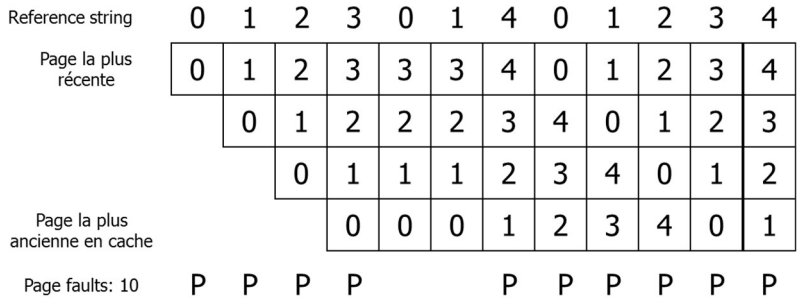
Certains algorithmes fonctionnent très bien dans des circonstances particulières. Par exemple, FIFO fonctionne bien pour des processus référençant des pages suivant de longues séquences, mais offre de piètres performances pour la majorité des autres processus. LRU fonctionne parfaitement pour des références dont la localisation est poussée. Retenons qu'en général, l'algorithme working set est le plus précis et se rapproche à un coût raisonnable de l'efficacité de MIN/OPT.

D. Anomalie de Belady et stack algorithm.

On devrait s'attendre à ce que le nombre de *page faults* diminue ou à tout le moins reste constant si la taille de la mémoire augmente. Cette propriété est vérifiée pour les *Stack algorithm*. L'algorithme FIFO n'est pas un *stack algorithm*, il est sujet à l'anomalie de Belady qui a le premier montré que pour un certain *pattern* d'allocation, une augmentation de la mémoire cause une augmentation du nombre de page fault. En fait, FIFO fonctionne mal si le pattern de référence passe alternativement du mode séquentiel au mode aléatoire.



(a)



(b)

Fig. 50. Anomalie de Belady.

Chaque processus génère une séquence de références à des pages virtuelles, cette séquence est appelée le *Reference String*. Supposons pour la simplicité de l'exposé qu'un seul processus est présent en mémoire. Le nombre de pages est fixé à m, nous disposons d'un algorithme de remplacement des pages et d'un processus référençant ces pages.

Conceptuellement, nous pouvons représenter la mémoire virtuelle comme un tableau de n entrées (autant qu'il y a de pages virtuelles). Les premières m entrées correspondent aux pages présentes en mémoire centrale, les n-m suivantes correspondent aux pages qui ont été sauvées en mémoire secondaire.

Au départ, le tableau est vide et tant que m pages différentes n'ont pas été référencées, nous rencontrons une condition de page fault. Lorsque m entrées sont occupées et que l'O/S rencontre une condition de page fault, l'algorithme de remplacement de page est appelé. Une page est choisie parmi les m premières et est laissée dans les n-m suivantes, la page manquante suit le trajet inverse. Un réarrangement des pages tant en mémoire centrale (les m premières entrées) qu'en mémoire secondaire est possible. L'algorithme LRU est illustré par le schéma.

Un algorithme est classé dans la catégorie des *stack algorithm* si M(m,r) est inclus dans M(m+1,r), par conséquent, le nombre de page faults diminue en fonction de m. On peut rapidement remarquer que FIFO échappe à cette règle et est donc sujet à l'anomalie de Belady.

E. Estimation de l'efficacité

a) Distance string

La chaîne des distances (*distance string)* est un tableau contenant une entrée pour chaque référence d'une page ; la valeur de chaque entrée est égale au nombre de cases séparant la page du début de la liste. Lorsqu'une page n'est pas présente en mémoire ni centrale ni secondaire, la valeur vaut l'infini. La fréquence d'apparition des valeurs permet de mesurer l'efficacité de l'association entre la taille de la mémoire et l'algorithme de pagination puisque la distance dépend des deux éléments.

Reference string	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Page faults	P	P	P	P	P	P	P		P					P			P						P	
Distance string	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	2	3	5	3

Fig. 51. Distance String.

La figure suivante reprend l'évolution de la probabilité d'apparition d'une valeur de d: distance par rapport au sommet de la liste de pages. Plus la probabilité d'apparition de 1 ou des valeurs proches de 1 est grande, plus l'algorithme est efficace.

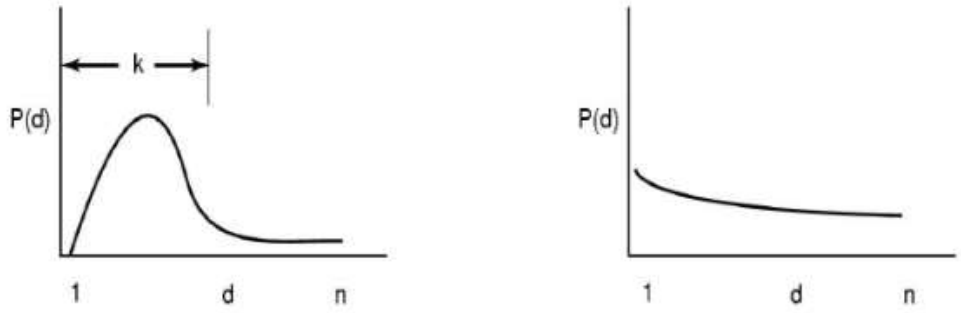


Fig. 52. Efficacité d'un algorithme de pagination.

b) Estimation du nombre de Page Fault

Reprenons dans un tableau la fréquence d'apparition d'une valeur dans le distance string. Cette fréquence correspond à la probabilité d'apparition illustrée par la figure précédente. La valeur de cette entrée dans notre tableau (*Cn*) sera égale au nombre de fois que la valeur *n* apparaît dans le distance string.

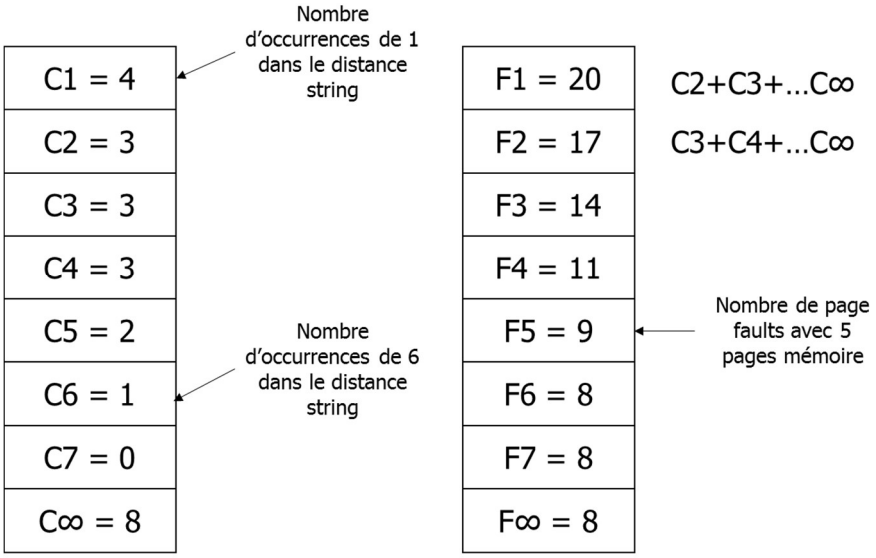


Fig. 53. Estimation du nombre de page faults.

Appelons Fm le nombre de page fault pour une certaine valeur de m (taille de la mémoire centrale) ; ce nombre s'exprime comme la somme de toutes les entrées du tableau précédent (*Cn*) pour m+1 à ∞, soit pour le nombre de références à des pages situées en mémoire secondaire.

4. Segmentation

A. Introduction.

La segmentation a été introduite afin de permettre la définition de plusieurs aires de mémoires virtuelles utilisables par un ou plusieurs processus. Comme nous le verrons dans la suite, un processus peut disposer d'un segment de mémoire virtuelle à partir duquel le processeur « fetche » les instructions, d'un autre segment de mémoire virtuelle dans lequel il stocke des données... Un segment est appelé *address space* ou *data space* en MVS.

Dans un système segmenté, la mémoire n'est pas divisée en pages de tailles égales, mais en blocs contigus appelés segments, ces segments pouvant être de tailles différentes. Chaque adresse générée par le processeur est composée d'un numéro de segment et d'un déplacement dans le segment. Le déplacement est également appelé *Offset*, le segment la base. Un aspect important de la segmentation consiste en une séparation entre le segment et l'offset. Une incrémentation de l'offset ne peut en aucun cas conduire à une incrémentation d'un segment. Une fois l'offset maximum atteint, lui ajouter 1 doit provoquer une erreur ou provoquer un wrap around au sein d'un même segment.

L'adresse des segments est contenue dans une table des segments à raison d'une table de segments par processus. Cette table des segments est adressée via un registre de contrôle.

B. Segmentation et pagination

On peut combiner la pagination et la segmentation, ce qui permet de simplifier l'allocation de la mémoire. La segmentation est appelée segmentation symbolique. Chaque segment est partagé en pages et l'unité de transfert entre mémoire secondaire et mémoire centrale est la page.

L'adresse virtuelle est donc constituée par un numéro de segment, un numéro de page et un déplacement. La traduction de l'adresse se base sur un pointeur adressant une table de segments, le numéro de segment permet d'identifier une *page table*. Le numéro de page permet d'identifier la page à laquelle l'offset est ajouté pour localiser le mot ou la ligne.

Un processus peut donc utiliser des segments au moyen d'instructions particulières. Ces instructions permettent de jouer avec les registres de contrôle afin d'adresser des opérandes localisés dans d'autres segments.

C. Exemples de segmentation.

a) Le 80386

Le microprocesseur 8086 contient 4 registres appelé le CS (*Code Segment*), le DS (*Data Segment*), le SS (*Stack Segment*) et le ES (*Extra Segment*). Une adresse générée par le processeur est une adresse en 16 bit sans numéro de segment.

Les instructions sont *fetchées* sur base du *code segment register* auquel on ajoute l'offset du compteur ordinal (*Instruction pointer*). Les données sont adressées en utilisant le DS, les opérations sur pile utilisent le SS. L'ES est utilisé en pour adresser le résultat d'opérations sur string.

L'emploi de ces registres est étendu sur 286 par l'emploi de *Descriptor cache register*. Ces derniers contiennent une image de blocs de contrôle décrivant les segments. Des instructions particulières permettent de manipuler les registres CS, SS, DS et ES. L'adresse d'un descripteur de segment peut y être chargée, automatiquement le processeur charge le *descriptor cache register* et le segment est employé comme base dans la résolution des adresses. 2**13 segments peuvent être définis
.
Le 80386 génère des adresses en 32 bits, comme nous l'avons expliqué dans ce cas, des pages tables à plusieurs niveaux deviennent nécessaires. Elles sont utilisées si un bit d'un registre de contrôle indique que la pagination est active.

b) MVS : address spaces

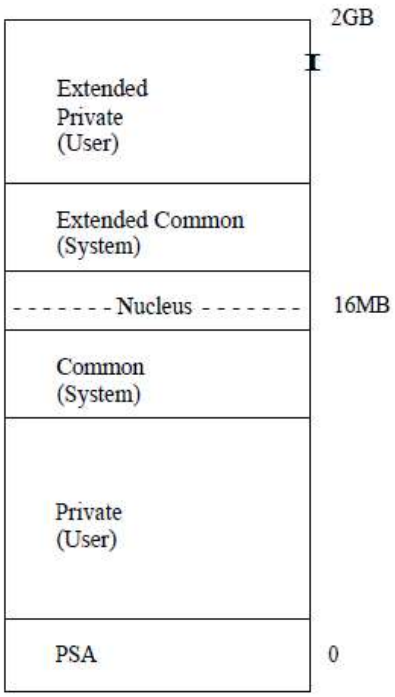


Fig. 54. Address spaces MVS.

En MVS, la mémoire est partagée en segments communs pour tous les utilisateurs et en segments appelé *Private Area* ou *Private Region*. Une pagination est appliquée tant aux segments communs qu'aux segments privés. Chaque processus peut adresser 2GB de mémoire virtuelle. Pour rappel, une page de mémoire appartenant à un segment commun est désignée par un même déplacement dans la table des segments appartenant à tous les processus, ces tables de segments pointent toutes vers une même *page table*.

Les segments communs sont utilisés pour passer des informations ou pour contenir du code exécutable.

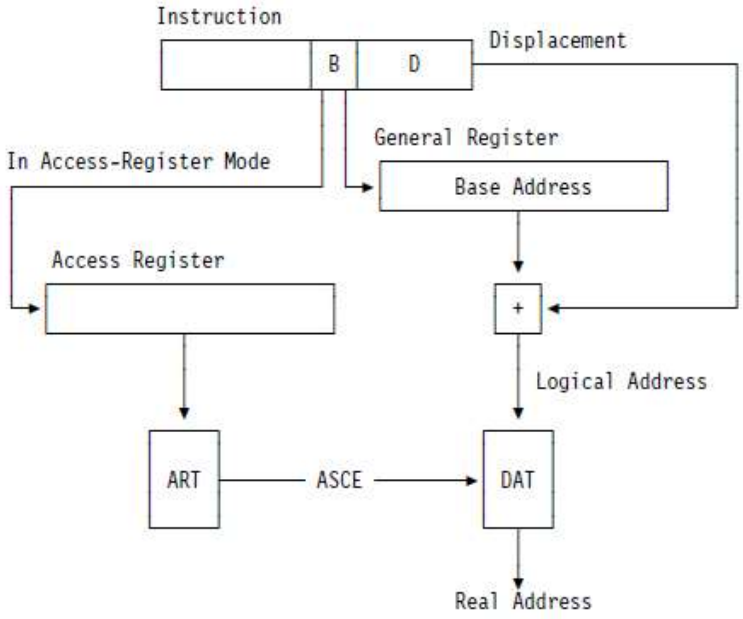


Fig. 55. Data Spaces MVS: résolution des adresses.

Une extension à la segmentation a été apportée par MVS ESA/370. Cette architecture permet à un processus de définir des segments de type *data*. Le mécanisme de résolution des adresses est basé sur des registres complémentaires appelés *access register*. Ce registre contient l'adresse d'une entrée dans une structure de données particulière appelée *access list*. Ce mécanisme appelé *access register translation* permet de retrouver l'adresse de la *segment table*. Cette adresse est transmise à la DAT pour transposition de l'adresse virtuelle en adresse réelle.