CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Ecole Thématique : " Conception d'Architectures de Systèmes Informatiques dédiés à des applications spécifiques de type 'enfoui' "
Seix (Ariège), du 20 au 24 novembre 2000

HIERARCHIE MEMOIRE : PROTECTION / MEMOIRE VIRTUELLE / CACHES

Daniel Litaize
IRIT Université Paul Sabatier
TOULOUSE
05 61 55 64 16 litaize@irit.fr

INTRODUCTION

Les systèmes enfouis couvrent une très large gamme d'applications et donc recoupent des architectures variées. Système enfoui évoque système de petite taille, fermé.
De même que les premiers microprocesseurs simples ont fini par intégrer les mécanismes traditionnels (les simples existent toujours et sont toujours utilisés), les systèmes enfouis vont finir par intégrer les possibilités des systèmes actuels pour les raisons suivantes :

- L'ouverture sur le monde extérieur (intranet/Internet) va entraîner la nécessité de protections renforcées.
- La prise en compte d'applications multiples et variées par un même processeur, de façon statique ou dynamique (code mobile), va entraîner une gestion complexe de l'information et là aussi nécessiter une protection adaptée.

Le besoin de puissance de traitement va se traduire au niveau du processeur mais aussi au niveau de la hiérarchie mémoire.

Les mécanismes traditionnels seront-ils adaptés au contexte des systèmes enfouis ?

CACHE / MEMOIRE VIRTUELLE / PROTECTION

Raison essentielle de la présence d'une hiérarchie de mémoires cache : pallier au manque de performance en temps d'accès de la mémoire centrale en technologie DRAM ou ROM.
Pose un problème avec le temps réel.
Utiliser de la SRAM partout. Gérer le cache soi-même comme une mémoire locale ?

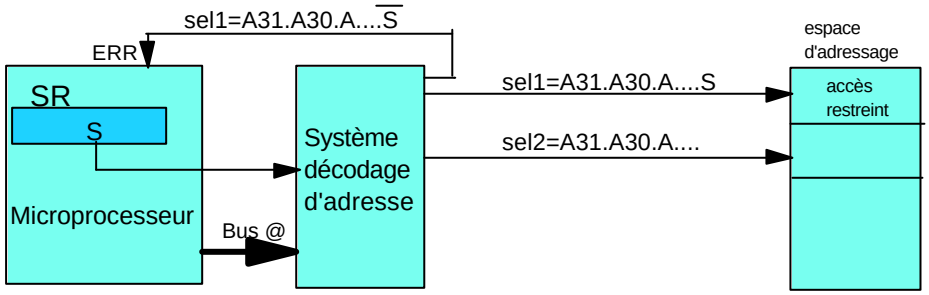
Raison essentielle de la présence d'une mémoire virtuelle : gestion automatique de la mémoire dans un environnement dynamique. Toujours utile ?

La protection : pas forcément utile dans un système fermé et statique. Souhaitable dans un environnement fermé et dynamique. Indispensable dans un système ouvert.

Concepts connus depuis longtemps, mais dont l'utilité et la mise en oeuvre évoluent avec le temps.

Objectif de la présentation : faire le point dans un contexte de système enfoui.

PROTECTION SYSTEME / UTILISATEUR(S) : MODE MAITRE/ESCLAVE



Caractéristiques :
Bit S de la zone SR accessible qu'en mode maître.
Passage mode esclave mode maître : instruction TRAP/SysCall. Provoque:

- sauvegarde de SR et PC (pile ou registres de sauvegarde)
- mise à 1 de S
- PC<- adresse donnée.

Zones mémoire protégées => accès possible que en mode maître, y compris et surtout la pile.
Possibilité de déroutement en cas de tentative d'accès (ERR).
Protection nécessaire et suffisante (?)

MEMOIRE VIRTUELLE SEGMENTEE / MV PAGINEE

But : gestion automatique de l'espace mémoire centrale de capacité insuffisante due à la présence de plusieurs utilisateurs actifs simultanément sur le processeur. Conséquences : mécanismes de protection et de partage nécessaires. Ces deux mécanismes peuvent être exploités sans utiliser l'aspect mémoire virtuelle, qui voit la mémoire centrale comme un cache du disque.

Protection et partage du point de vue de l'utilisateur conduit au concept de mémoire segmentée. Un segment = un objet (code, données) que l'on partage avec des droits d'accès différents.
Contrainte : le segment doit être en mémoire, mais une fois en mémoire peut être utilisé sans interruption contrairement au dispositif paginé.

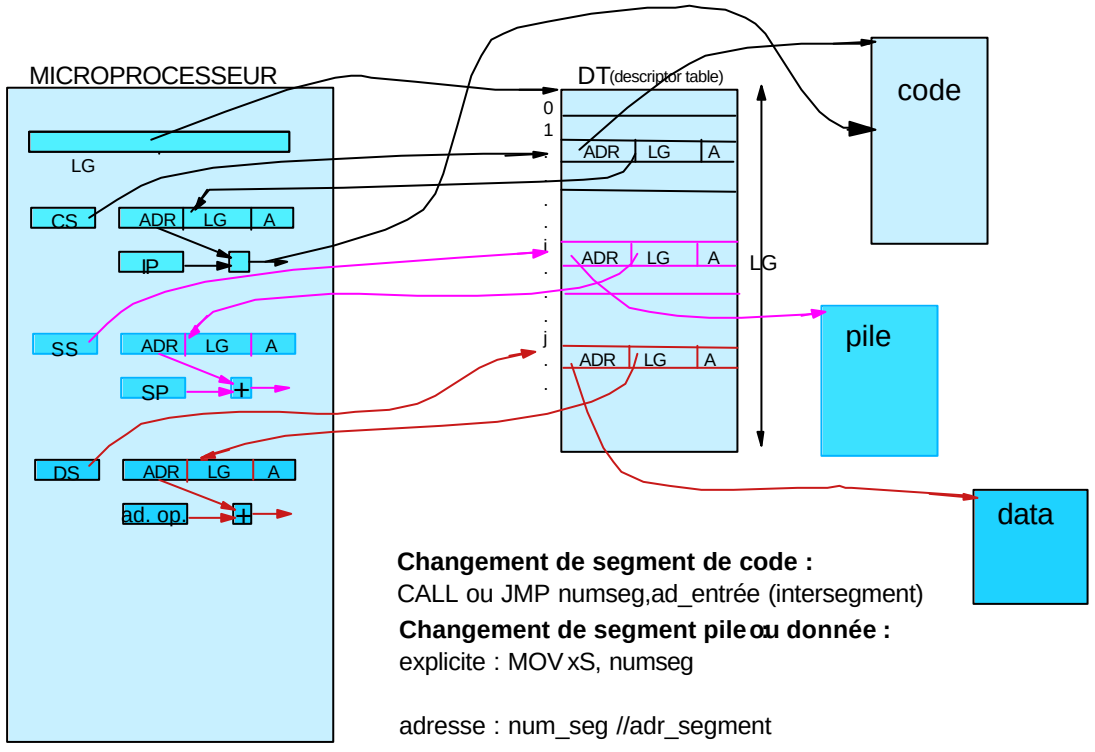
Protection et partage du point de vue de l'optimisation de la ressource mémoire conduisent au concept de mémoire paginée. Fait usage de la localité de l'information. Partage et protection se font au niveau de la page. Un segment est "virtuellement" un ensemble de pages.
Contrainte : pages chargées au fur et à mesure des besoins.

Combinaison des deux : paginer les segments. N'est ce pas redondant ?

Ces deux mécanismes font usages de tables. Des caches de ces tables sont nécessaires. Quelles sont les interférences possibles entre ces tables et les répertoires des caches ?

Quels sont les problèmes posés par l'environnement multiprocesseurs.

PRINCIPE DE L'ADRESSAGE SEGMENTE INTEL



Changement de segment de code :
CALL ou JMP numseg,ad_entrée (intersegment)
Changement de segment pile ou donnée :
explicite : MOV xS, numseg

adresse : num_seg //adr_segment

CARACTERISTIQUES D'ENSEMBLE

NOMBRE DE SEGMENTS :

le nombre de segments est limité par la taille des registres segments (CS,SS,DS,...)

14 bits chez INTEL => 2^{14} entrées de tables possibles.

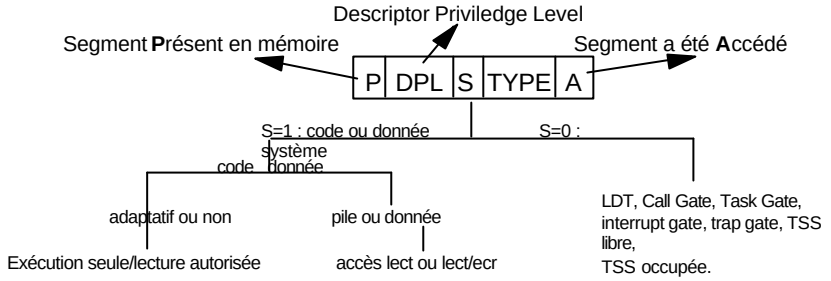
TAILLE DU SEGMENT :

limitée par la capacité des registres d'adresse du micro : taille du PC pour un segment de code, taille de la pile pour le segment pile, taille de l'adresse effective calculée pour les données. 32 bits chez INTEL.

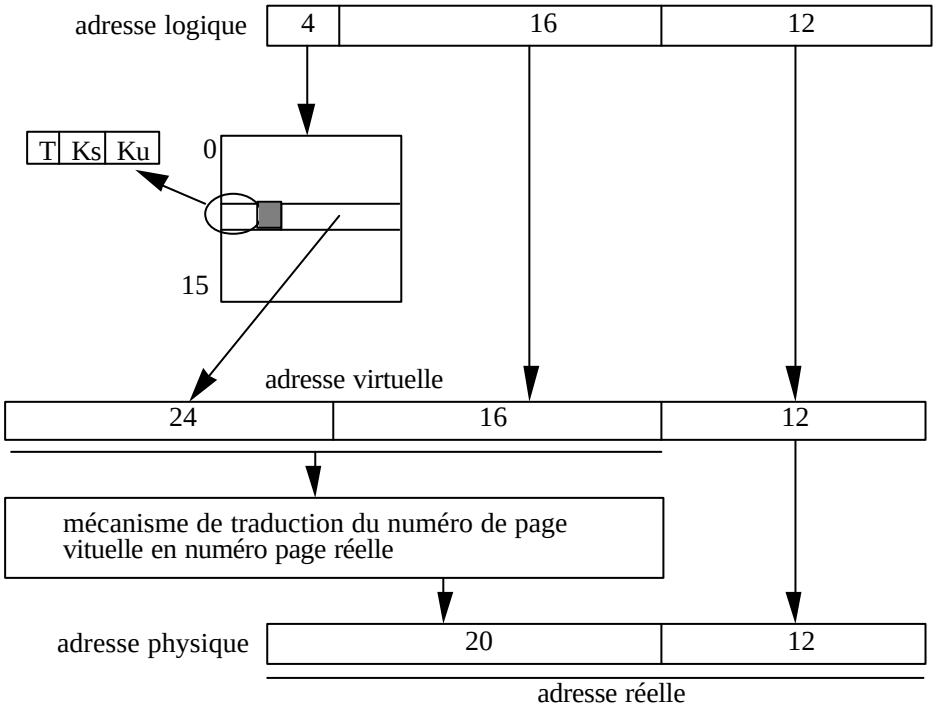
TAILLE MAXIMUM DE LA MEMOIRE PHYSIQUE :

limitée par le nombre de bits du bus adresse, nombre qui se retrouve dans le champ adresse d'un descripteur de segment. 32 bits chez INTEL. On pourrait imaginer un bus adresse sur 40 bits et des segments de taille max 2^{32}.

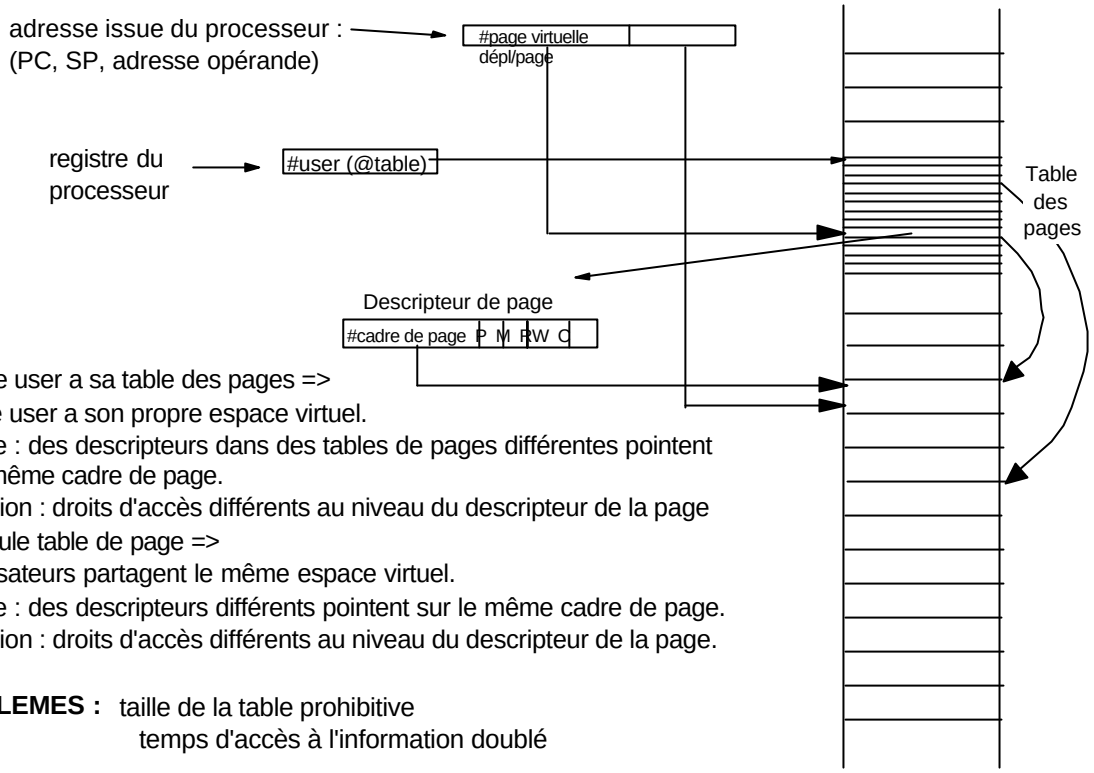
DEFINITION DES CHAMPS DE L'OCTET ATTRIBUT DU SEGMENT (INTEL):



PRINCIPE DE L'ADRESSAGE SEGMENTE IBM/PowerPC



PAGINATION



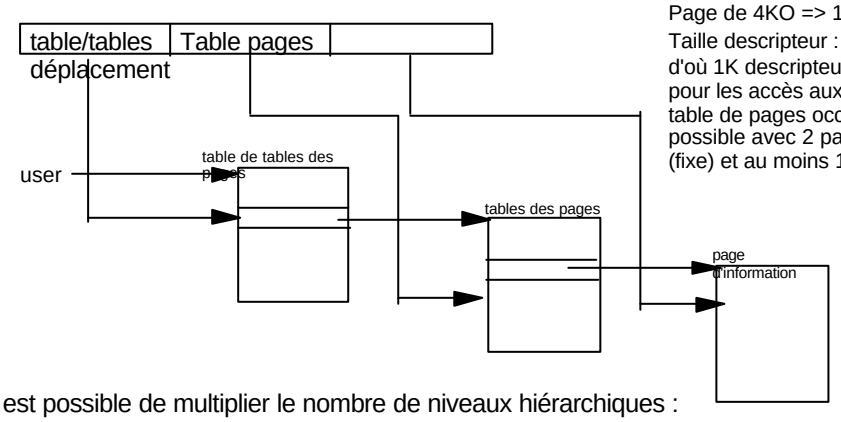
Chaque user a sa table des pages =>
chaque user a son propre espace virtuel.
Partage : des descripteurs dans des tables de pages différentes pointent sur le même cadre de page.
Protection : droits d'accès différents au niveau du descripteur de la page

Une seule table de page =>
les utilisateurs partagent le même espace virtuel.
Partage : des descripteurs différents pointent sur le même cadre de page.
Protection : droits d'accès différents au niveau du descripteur de la page.

PROBLEMES : taille de la table prohibitive
temps d'accès à l'information doublé

REDUIRE LA TAILLE DE LA TABLE : PAGINER LA TABLE

Cas bus adresse 32 bits :



Page de 4KO => 12 bits pour l'adressage octet.
Taille descripteur : #cadre+bits états => 4 octets
d'où 1K descripteurs par page et donc 10 bits pour les accès aux tables => page de table de table de pages occupée à 100%. Fonctionnement possible avec 2 pages : table de tables de pages (fixe) et au moins 1 table de pages (dynamique)

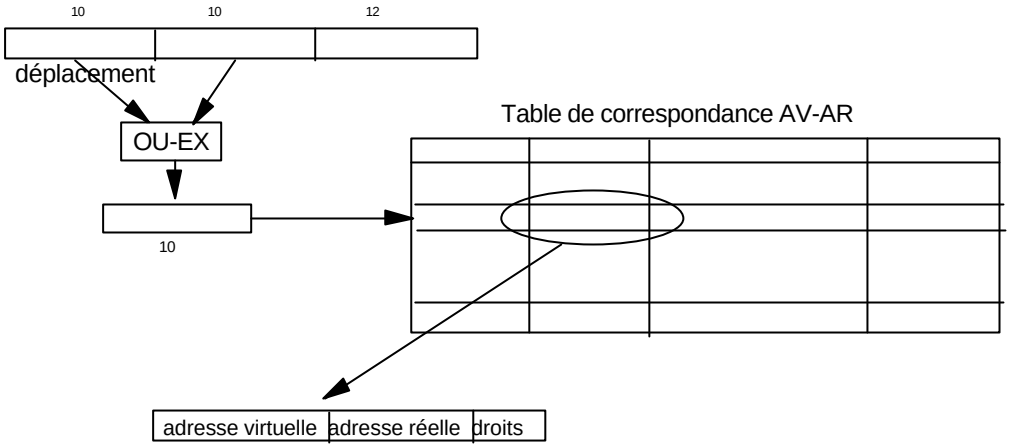
Il est possible de multiplier le nombre de niveaux hiérarchiques :

1. soit pour assurer un partage de pages, de tables de pages,...Le descripteur de niveau directement supérieur définissant les droits d'accès => donne l'ordre des inclusions de droits d'accès et protections.
2. soit pour agrandir la taille de l'espace mémoire physique (suppose des manipulations d'adresses de plus de 32 bits)

Problèmes :

- chaque niveau supplémentaire ajoute un accès de plus à la mémoire pour consulter la table correspondante.
- une table n'occupe plus la page à 100%. => Introduire des pages de taille différentes multiples d'une taille de base ?

REDUIRE LA TAILLE DE LA TABLE : HASHCODER L'ADRESSE



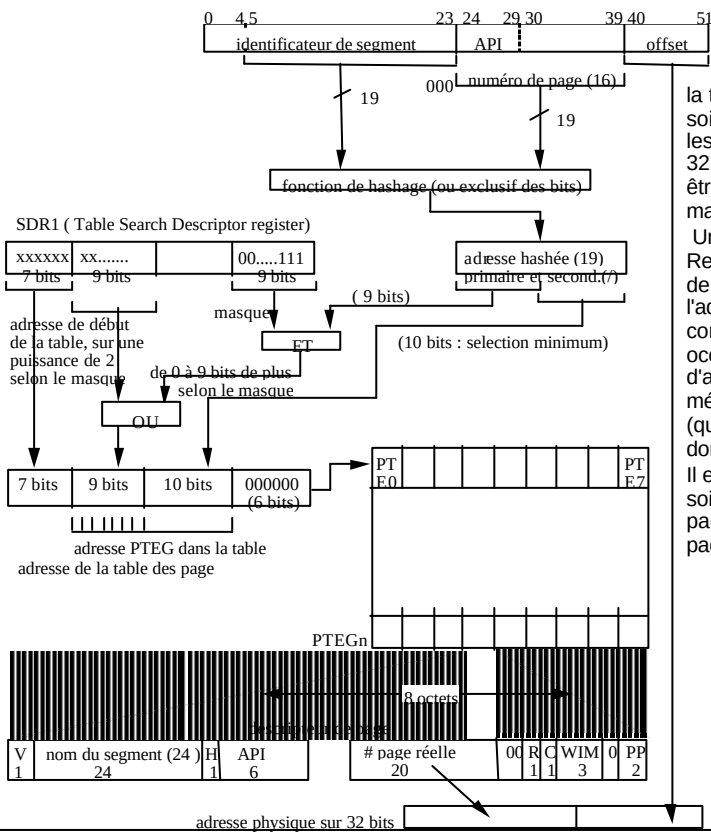
On passe d'une table de 2^{20} entrées à une tables à 2^{10} entrées (1M à 1K !) mais comme plusieurs combinaisons d'adresses conduisent à la même entrée, il faut trouver la bonne correspondance. La table est limitée en nombre de couples AV+AR : 8 par exemple. Une bonne fonction de hashage étale les adresses compressées sur la table.

Que faire si 8 entrées ne suffisent pas ?

Se produit lorsque l'on introduit une nouvelle page => il faut virer une page de la ligne de la table.

REDUIRE LA TAILLE DE LA TABLE :HASHCODER L'ADRESSE.

SOLUTION PowerPC

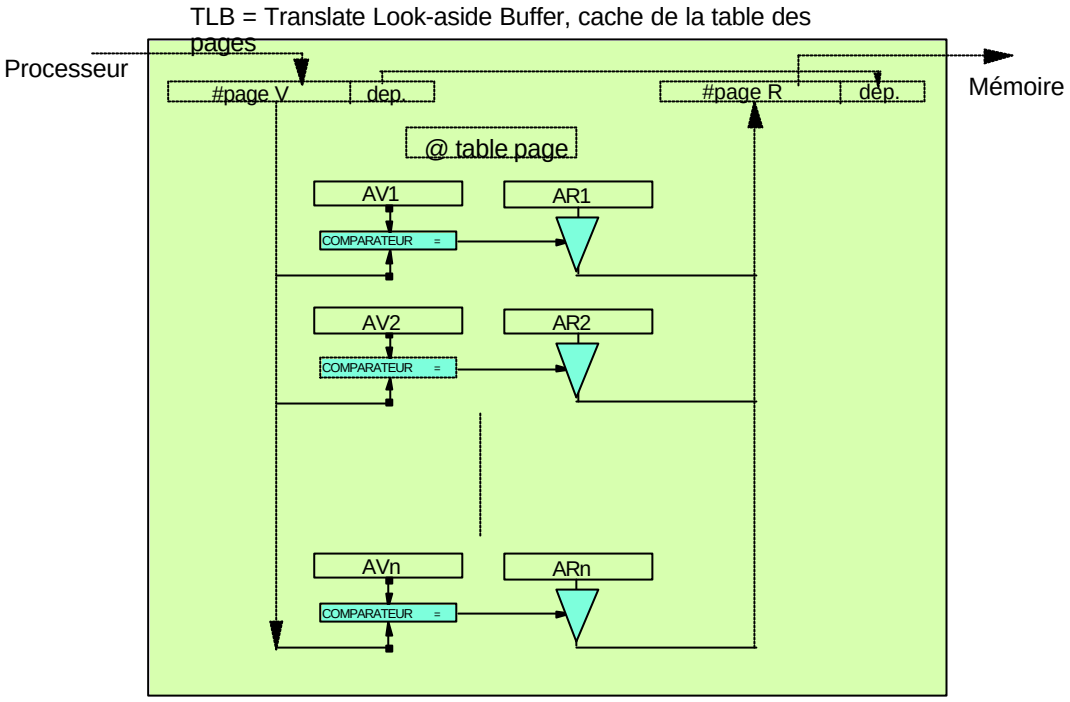


la taille de la table doit être calculée avec soin et est fonction de la taille mémoire. Si les 19 bits sont significatifs, la table occupe 32 MO et cette taille ne doit raisonnablement être utilisée que dans la cas de la taille maximum de 4GO.

Un registre SDR (Table Search Description Register) permet d'indiquer, par un masque de 9 bits, le nombre de bits à utiliser dans l'adresse hashée : de 10 (minimum, correspondant à une table à 2^{10} entrée et occupant 64 KO) à 19. Pour éviter un calcul d'adresse, cette table doit être placée en mémoire à une adresse multiple de sa taille (qui est une puissance de 2), adresse donnée dans SDR.

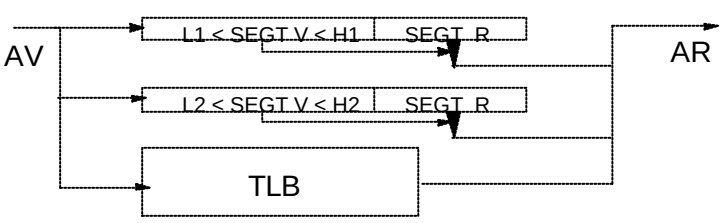
Il est recommandé que le nombre de PTEG soit au moins égal à la moitié du nombre de pages réelles (soit 4 fois plus de PTE que de pages réelles).

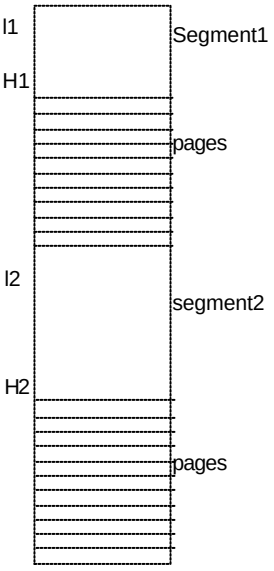
REDUIRE LE TEMPS DE CONVERSION AV/AR : UTILISER UN TLB



AUGMENTER LA PROBABILITE DE PRESENCE DANS LE TLB

1 entrée TLB = 1 descripteur de page => 1 entrée localise 4KO (cas usuel taille de page)
64 entrées => Pp équivalente à celle de celle de la taille d'un cache de 256 KO.
TLB partagé par toutes les applications. Optimisation possible pour les applications résidentes : le système par exemple

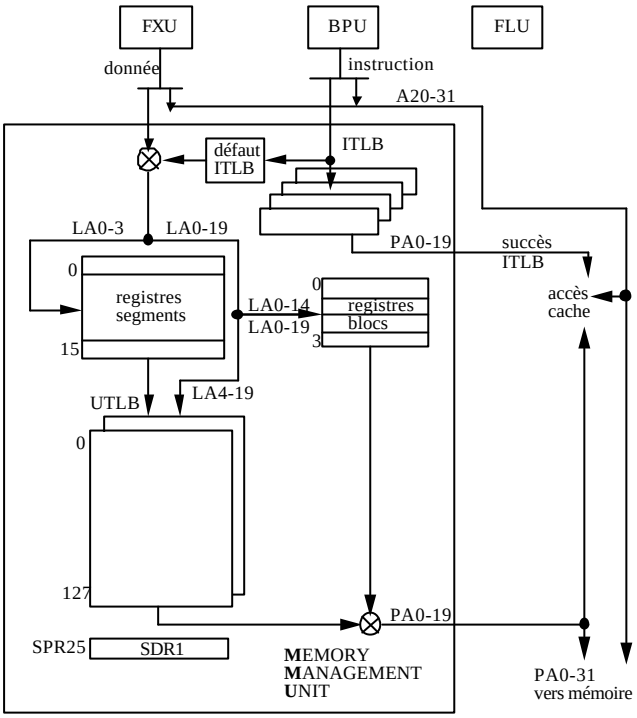




Un segment peut être créé dynamiquement.
Les registres segments donnent un résultat prioritaire sur le TLB => les pages du segments ne seront pas gérées par le TLB.

Les segments doivent avoir une taille égale à une puissante de 2 (au moins égale à la taille d'une page....) **ET** doivent être placés sur une adresse multiple de leur taille afin que la comparaison soit simple et rapide.

TLB REGISTRES SEGMENT DU PowerPC 601



Le 601 met en oeuvre deux TLBs :

- un UTLB (U pour unified, c'est à dire utilisé indifféremment pour adresse et données) qui contient 256 descripteurs de pages organisé en cache d'associativité d'ensemble d'ordre 2.

- un ITLB (I pour instruction) qui contient 4 descripteurs de page ou de blocs organisé en cache d'associativité totale et qui est utilisé pour les instructions seulement. Il faut noter que la consultation du UTLB se fait en cas d'échec dur le ITLB.

Remarque : si l'accès aux registres BATs se traduit par un succès, la translation d'adresse n'est pas poursuivie dans le UTLB.

HIERARCHIE MEMOIRE / ESPRIT CISC vs. RISC

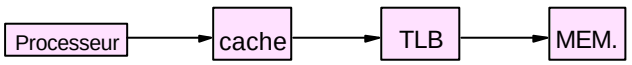
La famille x86 est l'archétype du concept CISC à tous les niveaux :
- jeu d'instruction et modes d'adressages (critère de définition CISC/RISC)
- système de protection par "anneaux" avec "4 niveaux de privilèges" au lieu du simple système M/S
- gestion mémoire à 2 niveaux : segmenté/paginé
- changement contexte câblé
- passage entre niveaux via des "guichets" avec une pile par niveau et copie automatique des paramètres

A l'opposé le contexte RISC a entraîné l'abandon de tous les mécanismes câblés qui pouvaient être pris en charge par le logiciel :
- plus de pile système par exemple
- gestion mémoire paginée uniquement. Le segment est logique : c'est un ensemble de pages ou un niveau dans la hiérarchie des tables* de pages.
- gestion du TLB par matériel et/ou logiciel.

La mémoire virtuelle est vue comme un cache du disque.
Dans des applications spécialisées type système enfoui la vision de la MV pourrait être différente, plus liée à des aspects protections => notion de segment mieux adaptée ?

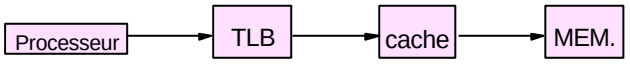
COHABITATION MEMOIRE VIRTUELLE / CACHE

PROCESSEUR -> MV -> CACHE ou PROCESSEUR -> CACHE -> MV ?



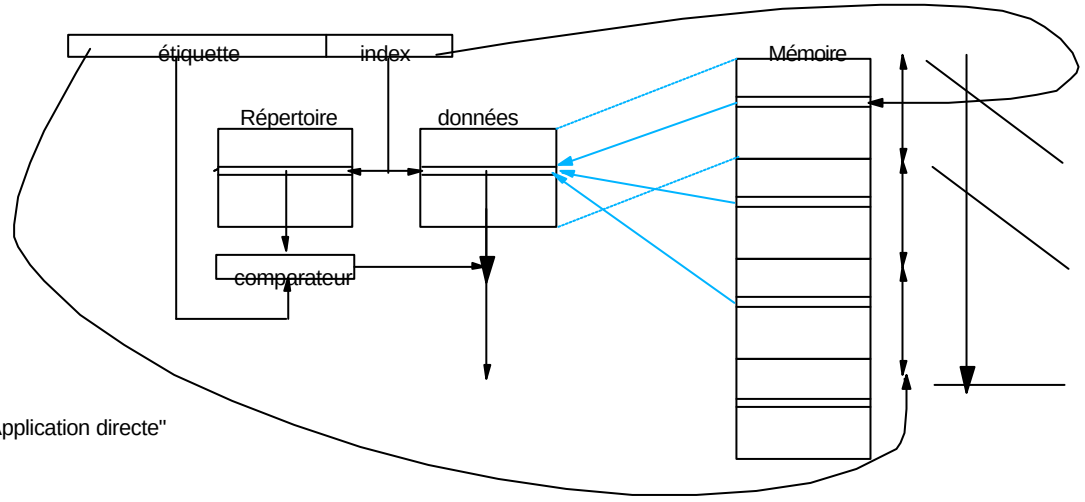
Cache en espace virtuel

Evite la consultation du TLB mais nécessite la purge du cache lors d'un changement d'espace virtuel à moins de préfixer les descripteurs de blocs de cache, nécessaire au moins pour le mode M/S



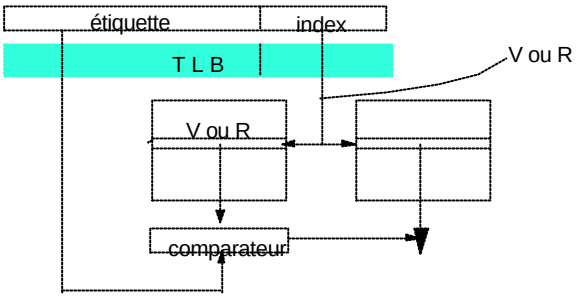
Cache en espace réel

Nécessite une consultation du TLB pour chaque accès mais aucune purge du cache ni préfixe de descripteurs de blocs de cache n'est nécessaire même pas pour le mode M/S (fait dans le TLB)



"Application directe"

PROCESSEUR -> MV -> CACHE ou PROCESSEUR -> CACHE -> MV ?

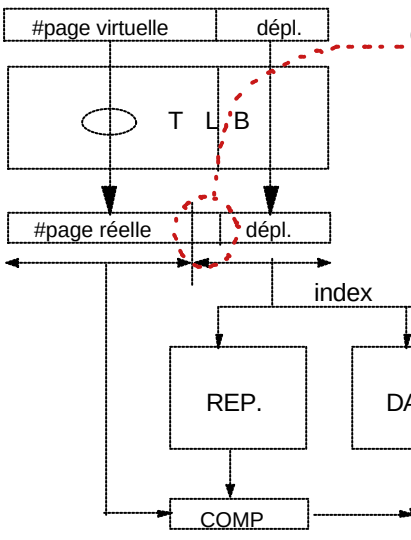


Mais un cache est composé d'un répertoire contenant des étiquettes qui identifient les blocs contenus dans la partie donnée du cache => l'accès au répertoire peut se faire avec une AV ou une AR
et les étiquettes identifier un bloc en EV ou en ER d'où 4 combinaisons :

type	indexation du cache	étiquettes du répertoire	utilisation
R/R	adresse réelle	adresse réelle	IBM
V/V	adresse virtuelle	adresse virtuelle	MIPS
V/R	adresse virtuelle	adresse réelle	IBM/HP
R/V	adresse réelle	adresse virtuelle	aucune

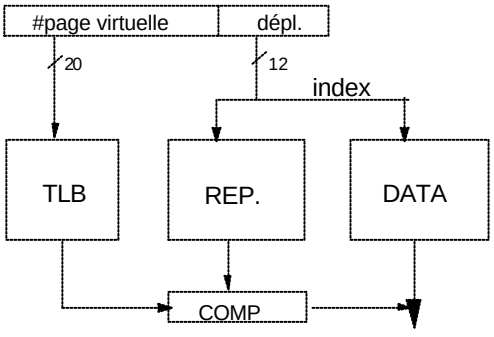
Avec une hiérarchie de caches on peut encore panacher les types de caches !
exemple : L1 V/V et L2 R/R => TLB placé entre L1 et L2

CACHE TYPE R/R



Dans la conversion d'adresse, seuls ces bits sont translatés. Le champ déplacement ne subit pas de modification => si on restreint la taille du cache à la taille de la page, on peut indexer le cache et translater l'adresse en parallèle.

Exemple :adresse 32 bits, page 4KO, cache de 4KO

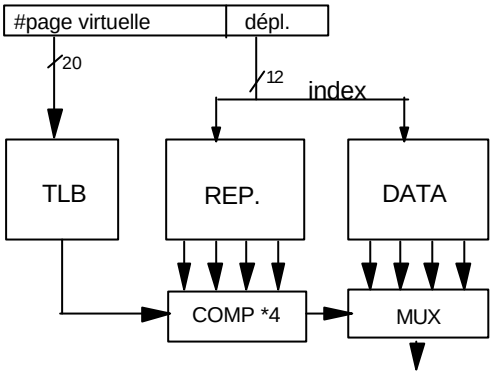


Comment augmenter la taille du cache tout en conservant ce mécanisme d'accès parallèle aux deux caches ?

CACHE TYPE R/R: conserver le // d'accès

Solution 1 : augmenter la taille de la page. Pas d'actualité en espace 32 bits.

Solution 2 : il est possible de conserver les 12 bits d'index en passant d'un cache à application directe à un cache à application associative :

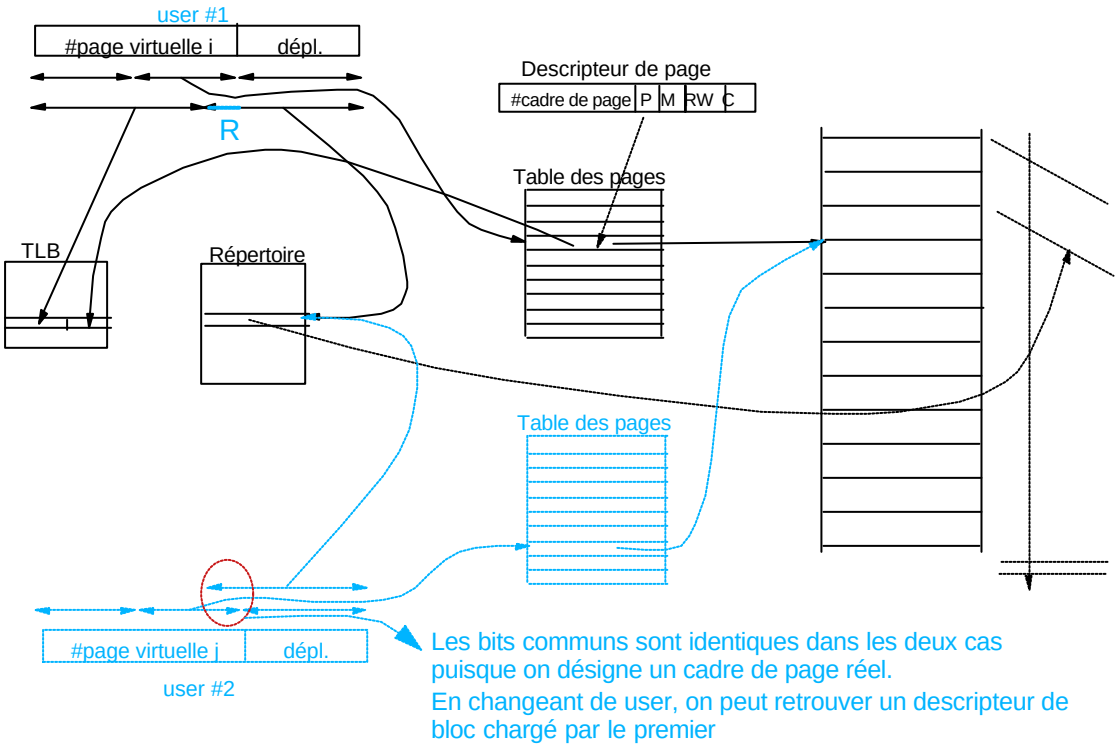


L'associativité rend le fonctionnement plus complexe : à l'ordre 4 il faut comparer 4 étiquettes en parallèle et accéder à 4 lignes de caches si on veut conserver le parallélisme d'accès répertoire et données. Explique pourquoi le PowerPc a une associativité d'ordre 8 (On a eu 16 chez IBM).
La Pp est meilleure en associatif qu'en direct, mais se stabilise à partir de 4.

Solution 3 : les bits communs sont en petit nombre : 3 par exemple pour un cache de 64 KO géré en associativité d'ensemble d'ordre 4. => on peut intercaler une petite table d'historique sur ces 3 bits qui font une conversion spéculative ultra-rapide.

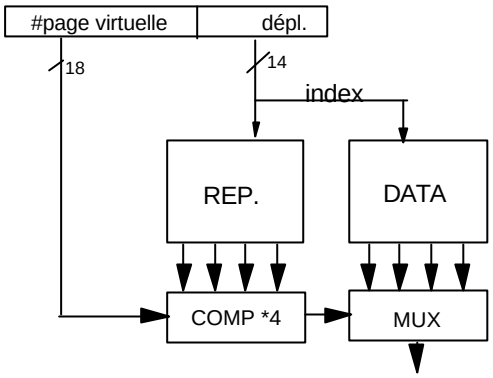
Solution 4 (logicielle) :allouer les pages de façon à ce que les bits communs soient identiques dans les deux espaces

DONNEES PARTAGEES EN R/R



Les bits communs sont identiques dans les deux cas puisque on désigne un cadre de page réel.
En changeant de user, on peut retrouver un descripteur de bloc chargé par le premier

CACHE TYPE V/ V



Exemple :
adresse 32 bits,
page 4KO,
cache de 64KO,
associativité d'ensemble d'ordre 4

Solution performante mais qui nécessite une purge du cache en cas de changement de contexte. La purge peut être faite "en douceur" en associant un identificateur d'espace à chaque descripteur de ligne de cache.

Principal problème : les synonymes en cas de partage, c'est à dire le cas ou deux ou plusieurs adresses virtuelles pointent sur la même adresse réelle.

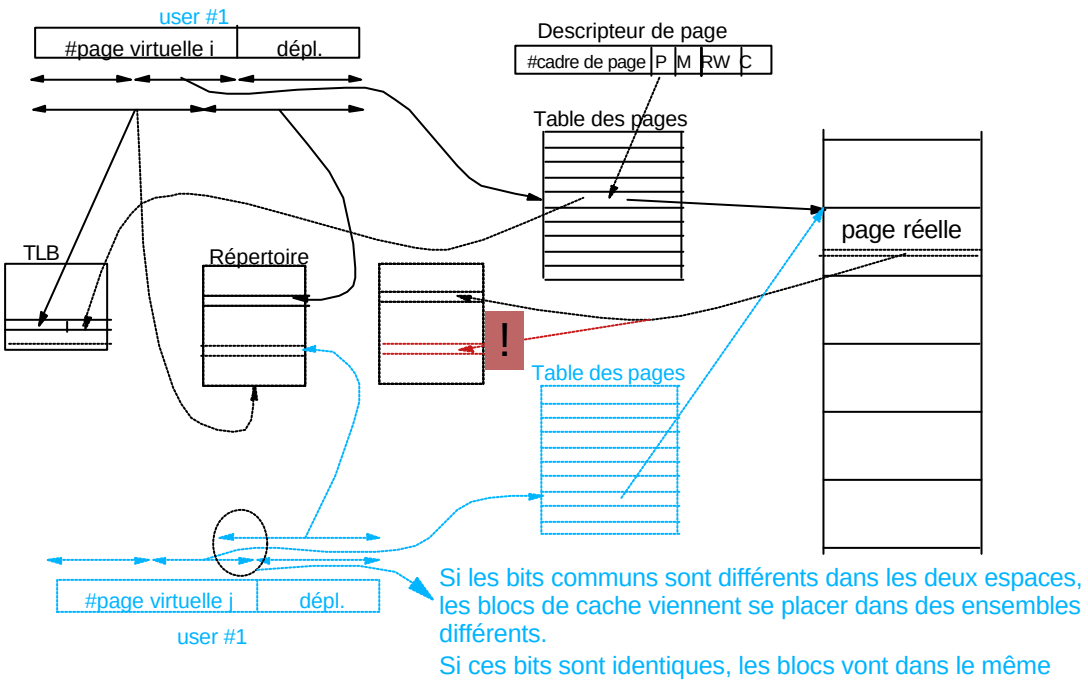
Seule façon de s'en rendre compte : repasser par l'adresse réelle et voir si d'autres adresses virtuelles donnent la même adresse réelle.

La détection d'un synonyme entraîne le déplacement du bloc d'un ensemble à l'autre, éventuellement en repassant par le niveau supérieur (on recherche le synonyme et on le met dans le bon ensemble) =>

Un synonyme entraîne donc un défaut cache bien que la donnée soit présente dans le cache.

La synonymie augmente avec la taille du cache et de l'associativité et donc réduit la Pp, sauf si on implémente un mécanisme qui rend les copies de blocs cohérentes entre elles.

DONNEES PARTAGEES EN V/ V

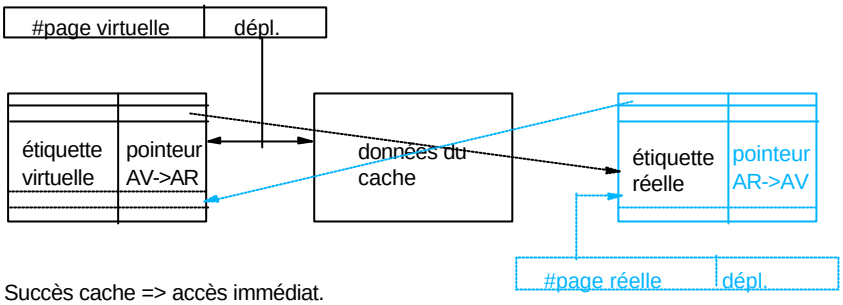


Si les bits communs sont différents dans les deux espaces, les blocs de cache viennent se placer dans des ensembles différents.

Si ces bits sont identiques, les blocs vont dans le même ensemble, mais peuvent occuper des places différentes dans l'ensemble en cas d'associativité.

DONNEES PARTAGEES EN V/ V : LES SYNONYMES

Solution 1 : conserver la trace de la correspondance entre espaces

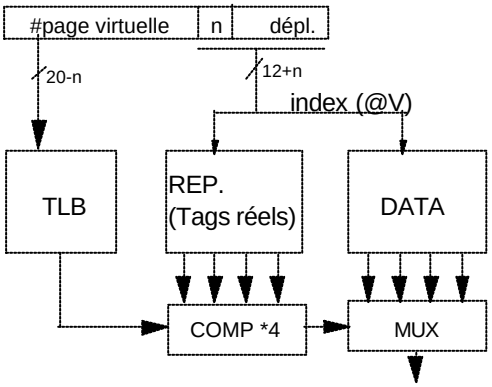


Succès cache => accès immédiat.
Défaut cache => consultation TLB et éventuellement table des pages, en cas de défaut TLB, pour trouver le # page réelle associé.
Vérifie que cette page réelle n'est pas dans le cache à un autre endroit. Si non : défaut normal.
Si oui : retrouve l'étiquette virtuelle associé et il faut déplacer le bloc d'un emplacement à l'autre.

Solution 2 (logicielle) :allouer les pages de façon à ce que les bits communs soient identiques dans les deux espaces.
Attention : ceci garantit que les deux blocs ne peuvent plus aller dans des ensembles différents, mais en cas de cache associatif, les deux blocs peuvent occuper deux emplacements différents. Il faut donc toujours vérifier les correspondances d'adresses, mais on a plus besoin de changer le bloc d'ensemble.

Solution 3 (logicielle) :utiliser les mêmes adresses partagées de façon à ce que tous les bits soient identiques dans les deux espaces.

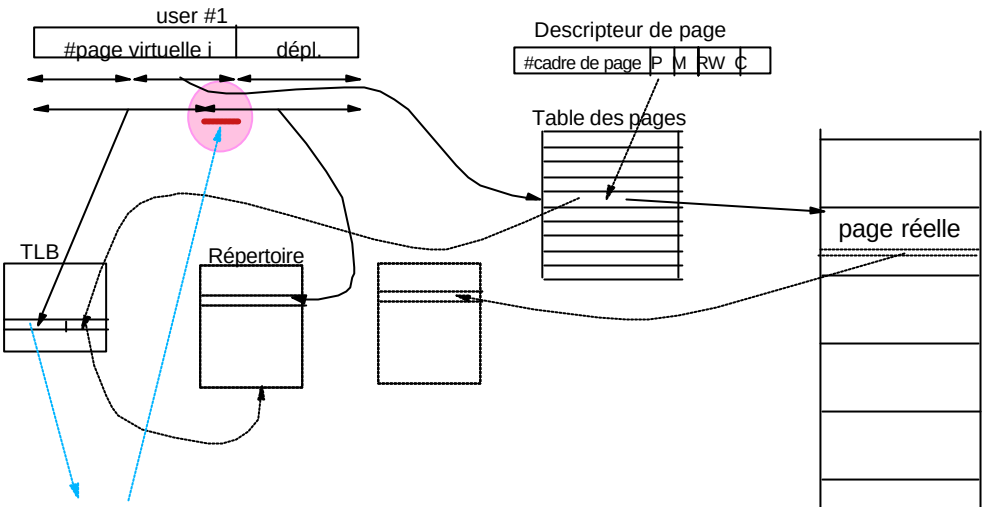
CACHE TYPE V/R



Exemple (cas n=2):
adresse 32 bits,
page 4KO,
cache de 64KO,
associativité d'ensemble d'ordre 4

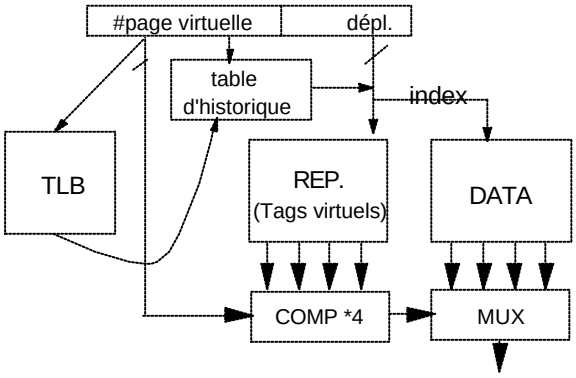
On a bien sûr toujours le problème des synonymes.
Allouer les pages de façon à ce que les bits communs soient identiques dans les deux espaces n'est pas une contrainte forte dans la mesure ou elle ne porte que sur quelques bits (2 dans l'exemple). Ceci garantit que les synonymes éventuels ne peuvent se trouver que dans le même ensemble.
Mais comme on compare toutes les étiquettes REELLES pour accéder aux données du cache, on identifie immédiatement le synonyme s'il existe : il n'y a aucun mouvement de données dans ce cas.

DONNEES PARTAGEES EN V/ R



Sous réserve que les bits communs soient identiques, les adresses virtuelles partagée conduisent dans le TLB à extraire la même étiquette réelle utilisée dans la comparaison.

DONNEES PARTAGEES EN R/V



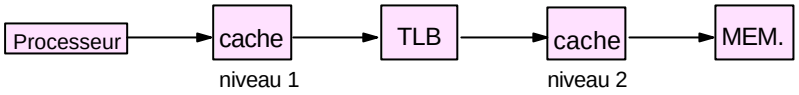
Solution qui présente peu d'intérêt :

- les tags virtuels entraînent des problèmes de synonymie dans l'ensemble de même nature que dans le cas V/V.

- il faut d'abord faire la correspondance AV -> AR avant de pouvoir accéder au cache

HIERARCHIE DE MEMOIRES CACHE A DEUX NIVEAUX

Semble une solution complexe, en réalité solution plus simple et performante avec les bons choix :
Cache de niveau 1 en espace virtuel avec mise à jour immédiate et inclusion stricte, et cache de niveau 2 en espace réel.



Niveau 1 :

- espace virtuel => accès immédiat en cas de succès.
- mise à jour immédiate => purge du cache instantanée.

Impact de la MAJ immédiate négligeable du fait du faible écart de performance entre les 2 niveaux de cache (4 à 10). Nécessite à chaque fois une consultation du TLB.

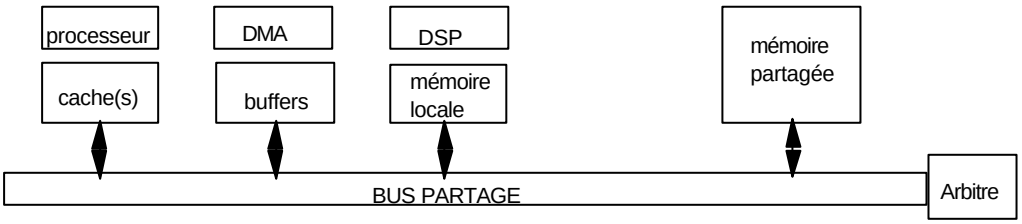
- défaut cache => consultation du TLB avant accès au cache de niveau 2.

Niveau 2 :

- pas de problème de synonymie puisque en espace réel.
- inclusion stricte (tout bloc dans le niveau 1 est dans le niveau 2) règle de façon simple la synonymie du niveau 1 : le niveau 2 a la trace des blocs présents dans le niveau 1. En cas de demande d'un bloc détecte si le bloc est déjà présent en niveau 1 mais à une adresse synonyme => force une purge avant de satisfaire le défaut.

CACHE EN ENVIRONNEMENT MULTIPROCESSEUR

Architectures multiprocesseurs à bus partagé : LA solution pour les petits systèmes.



Problème de la cohérence des données partagées et dupliquées dans les diverses mémoires :
peut être traité de différentes façon selon l'architecture et les applications considérées:
1 seul processeur+cache avec DMA => purge partielle du cache avant activation du DMA
1 seul processeur+ mémoire locale : dépend du fait que la mémoire locale soit accessible ou non, cacheable ou non.
plusieurs processeurs avec caches : zones de mémoire non cacheable pour les données partagées
ou mécanisme de maintien de la cohérence matériel, comme le protocole MESI

CONCLUSION / CACHE VS. MEMOIRE LOCALE

Dans les processeurs conventionnels les caches constituent un niveau de mémoire locale géré de façon automatique et quasi-transparent au logiciel.
Dans les DSPs on ne trouve pas de cache (sauf pour certains mais uniquement pour les instructions), mais de la mémoire locale gérée par logiciel.

Il a été quelquefois suggéré dans le monde des processeurs conventionnels qu'une possibilité de choix de gestion cache/locale apporterait une amélioration pour certaines applications (celles des DSPs ?)

Une gestion cache de la mémoire locale des DSPs avec des mécanismes de préfetch évolués déchargerait le programmeur de la gestion de la mémoire locale. Est-ce envisageable ?

L'approche VLIW qui semble se dessiner pour tous les types de processeurs performants (généralistes comme spécialisés type DSP) pose le problème de leur programmation efficace en langage évolué. Le même problème se pose pour l'accès efficace aux données si on dispose de mécanismes câblés d'accès aux données type DMA évolués.

Il semble que l'analyse du choix cache vs. mémoire locale dans ce contexte n'a pas encore été exploré.