Hiérarchie de Mémoire: Principale, Cache, Virtuelle

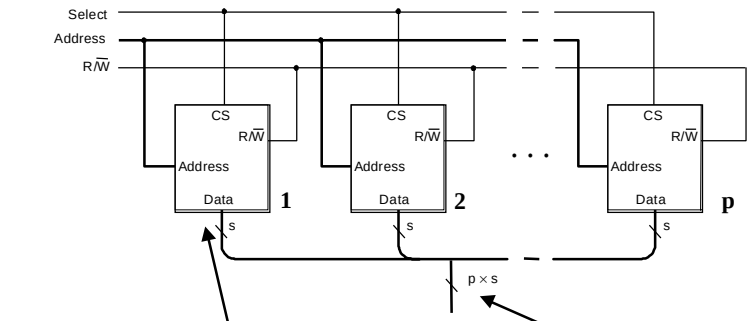
IFT1225

Dép. d'IRO, Université de Montréal
(professeur E. Cerny)

Planches et modules de mémoire

- Plus grandes (nb de mots) et larges (nb de bits/mot) mémoires construites avec les puces de base
- Assemblée sur une planche de circuit imprimé séparée du processeur ou intégrée sur la planche mère du système; bientôt intégrée avec le processeur
- Module ou banque de mémoire est formé de plusieurs planches
- Mémoire système assemblée de plusieurs modules
- Modules
  - satisfait le protocole d'interface avec le processeur
  - ajout de modules permet l'expansion de la mémoire
  - peut contenir le circuit de rafraîchissement de DRAM
  - peuvent être interlacés pour accélérer l'accès aux blocs de mots

Formation de plus large mémoire

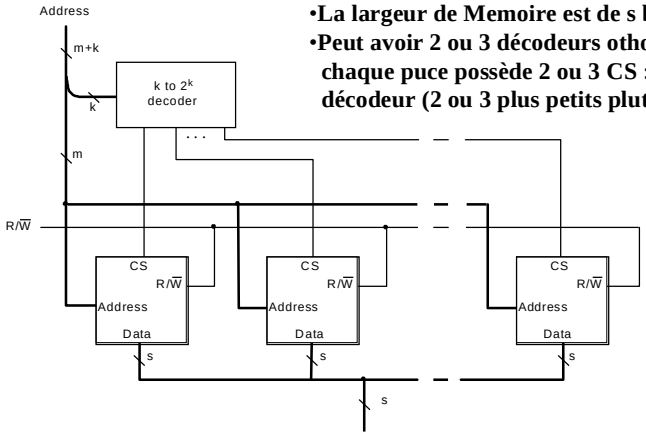


Placés en <u>parallèle</u>

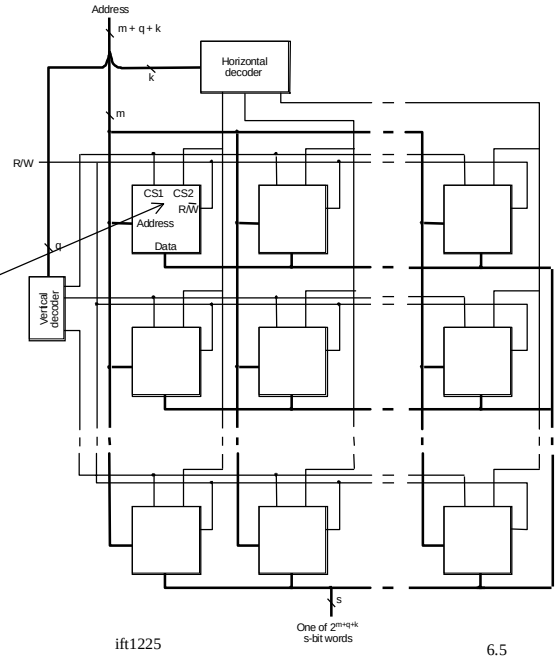
p puces de largeur s bits élargissent la mémoire à largeur de p x s bits

Formation de plus grande mémoire

- **k bits d'adresse sont décodés et sélectionnent puce ou planche ou module**
- **La largeur de Memoire est de s bits**
- **Peut avoir 2 ou 3 décodeurs othogonaux si chaque puce possède 2 ou 3 CS : simplifie le décodeur (2 ou 3 plus petits plutôt qu'un grand)**

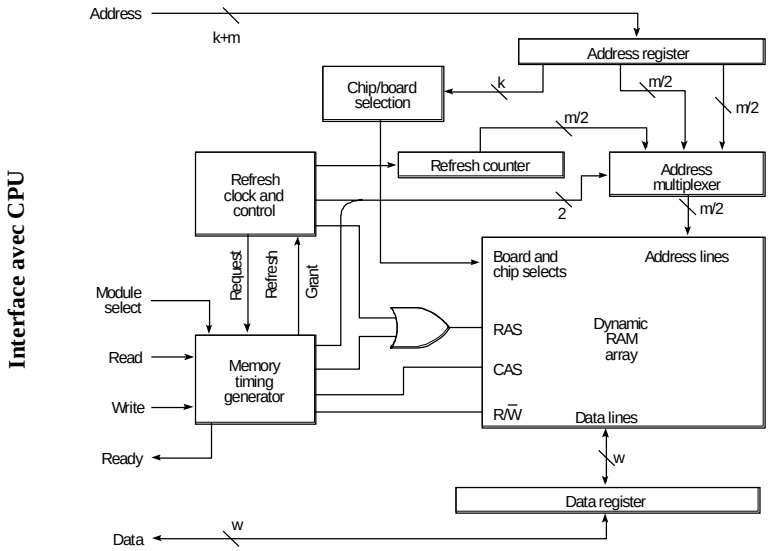


Mémoire à 2-D avec 2 décodeurs

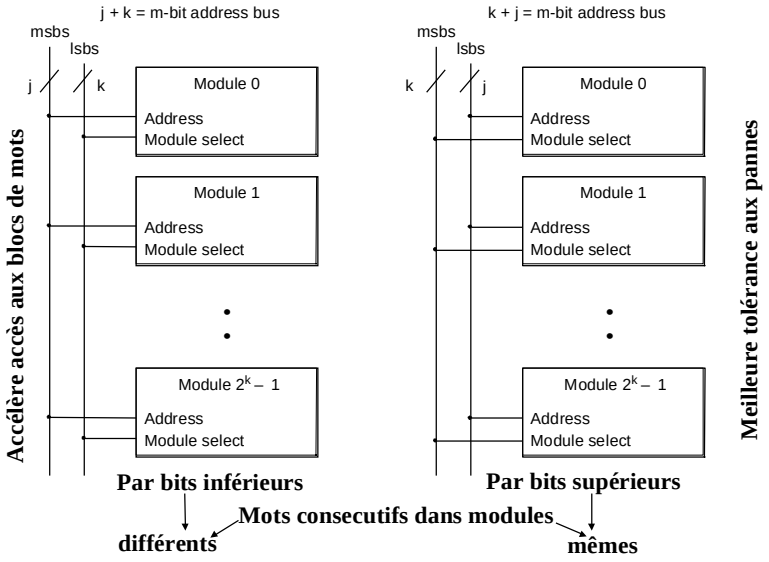


Deux CS combinés par une porte ET à l'intérieur

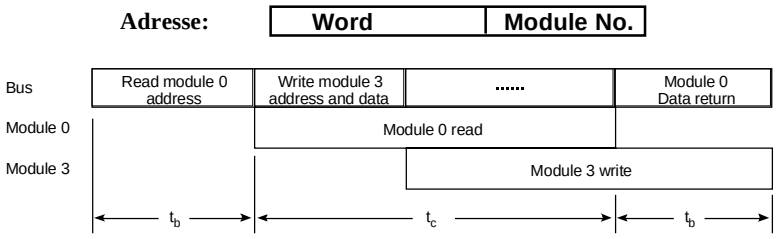
Module DRAM



Entrelacement



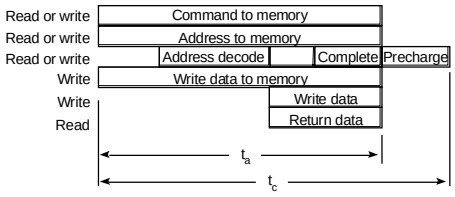
Entrelacement par bits inférieurs



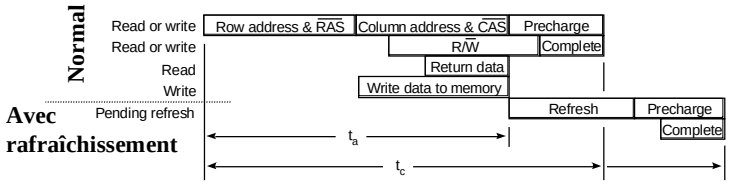
Si $t_{bus} < t_{cycle}$ alors transfert d'un bloc des mots consécutifs peut procéder avec le taux $1/t_{bus}$

Faut équilibrer le nombre des modules $m = 2^k = t_{cycle}/t_{bus}$
où k est le nombre des bits d'adresse inférieurs décodés

Séquencement d'un accès à la mémoire



Mémoire statique



Mémoire dynamique

Exemple de temps d'accès

- **Paramètres approximatifs de SRAM:**
  - **Activation des pilots de bus d'adresse: 40 ns.**
  - **Temps de propagation sur bus + tolérance: 10 ns.**
  - **Sélection de la planche: 20 ns.**
  - **Délai de propagation de sélection: 30 ns.**
  - **Sélection de puce: 20 ns.**

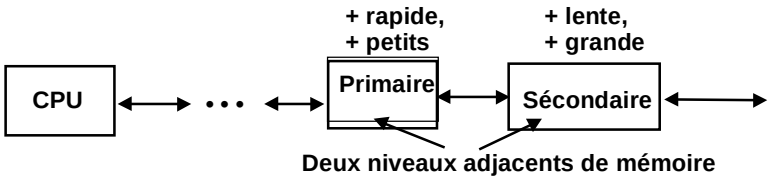
Temps total de propagation d'adresse: 120 ns.

  - **Temps d'accès dans le circuit intégré: 70 ns.**
  - **Délai du CI au bus de données de planche: 30 ns.**
  - **Délai de pilots et du bus même: 50 ns.**

- **Temps d'accès total: 270 ns.**
- **Conclusion: Circuit Intégré (CI) de mémoire de temps d'accès de 70 ns ne donnent pas nécesairement un temps d'accès à la mémoire de 70 ns !**

Hiérarchie de mémoire

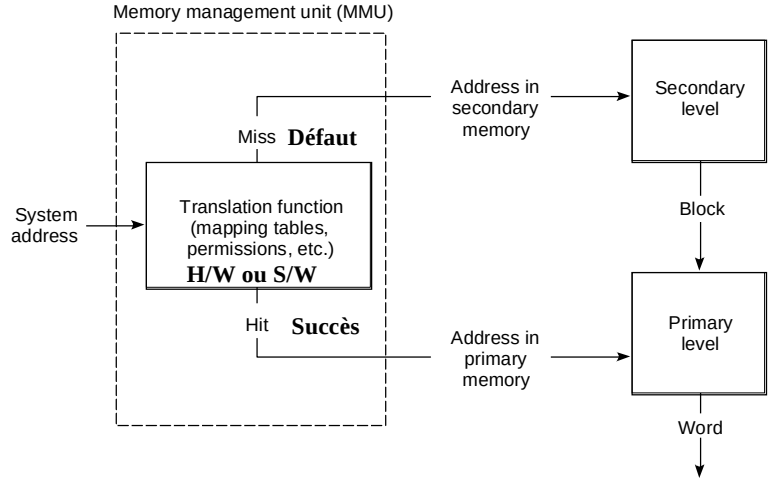
- **Objecif**: Faire semblance d'avoir une mémoire de la taille de la mémoire sécondaire avec la vitesse de la mémoire primaire (la plupart de temps)
  - **Cache** - Memoire principale : équilibrer la vitesse avec le CPU
  - Mémoire principale - Disque : **Mémoire virtuelle** : augmenter la taille apparent de la memoire principale
- Pourquoi ca fonctionne: Localité d'accès temporelle et spaciale
- Notion de *Working set*, ensemble de travail des blocs ou pages accédés en intervale de temps T (fenêtre de temps)… change lentement grâce aux "localités"….



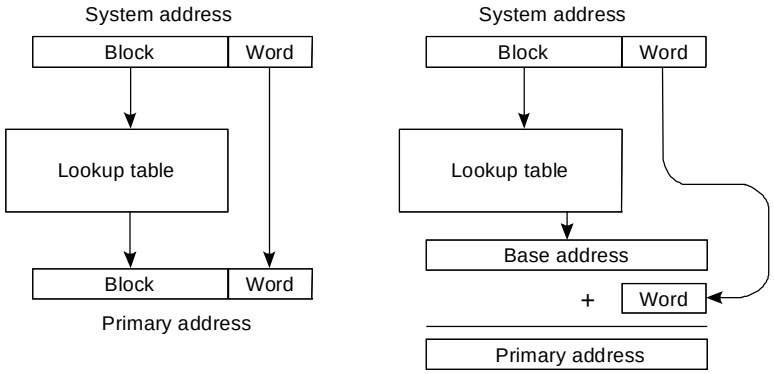
Niveaux primaire et sécondaire

- Transfert entre deux niveaux en terme de **blocs** (de mots, octets)
  - CPU - cache: quelques octets
  - cache - mémoire principal: bloc de 16 à 64 octets
  - mémoire - disque: 1k à 4k octets (taille de page virtuelle)
- Latence: temps pour obtenir le 1er mot
  - cache - mémoire principale: 4 à 50 top d'horloge
  - mémoire principale - disque: 100 000 top d'horloge
- Débit: nombre des mots transmis par seconde entre les 2 niveaux
- Lorsque l'ensemble de travail change, les bloc sont déplacés d'un niveau à l'autre pour satisfaire la demande
- Adresse changent d'un niveau à l'autre - besoin de traduction d'adresse
  P.E.: mémoire principale = entier;
  disque = (surface, piste, secteur, deplacement dans secteur)
  - Adresse primaire vs Adresse sécondaire

Adressage et accès dans 2 niveaux



Calcul d'adresse primaire



Système paginé (blocs de taille fixe)

Système segmenté (blocs de taille variable)

Caractéristiques d'accès, terminologie

Succès d'accès: le mot a été trouvé au niveau où demandé

Défaut d'accès: le mot n'était pas trouvé au niveau où demandé
(Il faut alors rchercher un bloc au niveau supérieur qui contient le bloc demandé)

Taux de succès = h = nombre de succès d'accès / nombre total d'accès

Taux de défauts: 1 - h

t_p = temps d'accès àmém. Primaire; t_s = temps d'accès à mém. sécondaire

Temps d'accès effectif $t_a = h \cdot t_p + (1-h) \cdot t_s$.

Page: habituellement, un bloc de disque Défaut de page: *Page fault*

Pagination sur demande: pages transférées du disk à la mémoire seulement si un mot est demandé par processeur vs. Prépagination

Décisions de placement et de remplacment : faut faire chaque fois un bloc est déplacé

Mémoires Caches

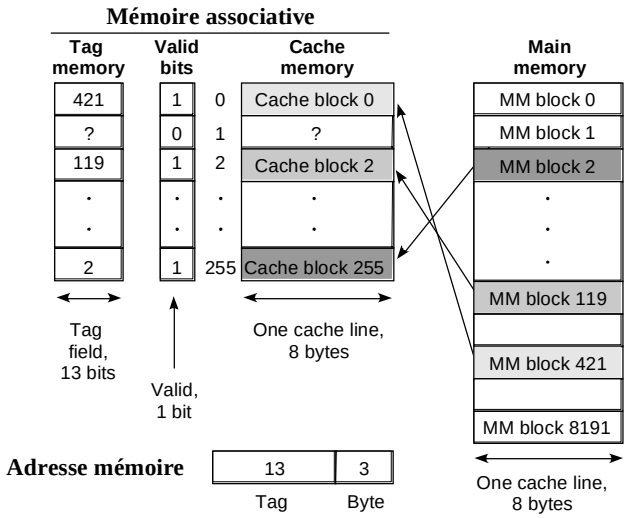
- Cache - mémoire rapide entre le CPU et la mémoire principale
- Agit comme un tampon pour accélérer des accès aux instructions et aux données (cache commun ou 2 caches séparés)

Décisions:

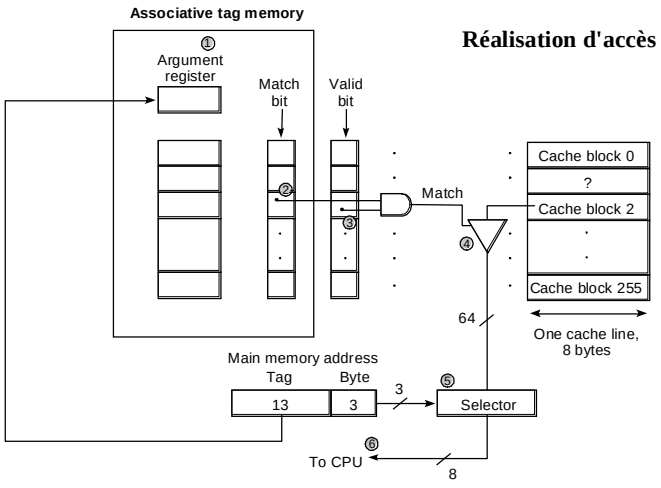
- Procédure de traduction d'adresses
- Taille de bloc
- Placement primaire: direct, associatif ou combinaison (associatif par ensemble de blocs)
- Politique de remplacement (FIFO, LRU, aléatoire, …)
- Accès au travers (direct) à la mémoire au cas de défaut
- Écriture immédiate (*write through*)
- Écriture au remplacement
- Lecture immédiate lors de chargement du bloc au cache

À cause de la vitesse d'opération, les mécanismes sont implantés en matériel

Cache associatif



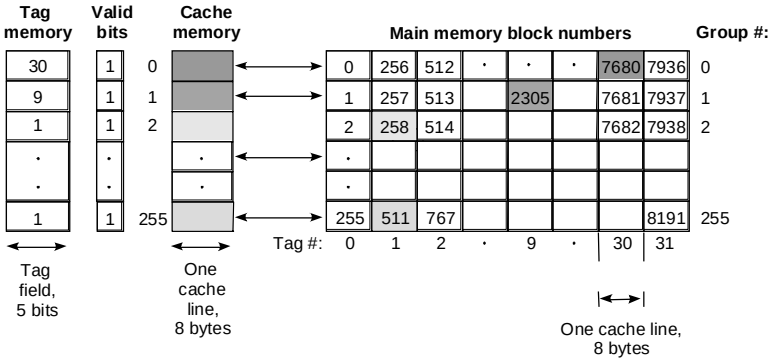
Cache associatif (suite)



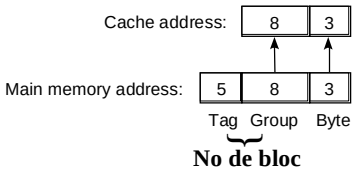
Cache associatif (suite)

- Avantages:
  - Le plus flexible, bloc peut être placé n'import où dan le cache
- Inconvénients:
  - Une grande mémoire associative est chère
  - produit beaucoup chaleur
  - recherche de tout mots en parallèle est compliquée
  - politique de remplacment oeut être un problème
- Autres solutions
  - associativité directe: Chaque bloc de mémoire ne peut parraitre que dans un bloc de cache
  - associativité par ensemble de blocs: Chaque bloc ne peut exister que dans un sousensemble fixe de bloc de cache.

Cache à associativité directe

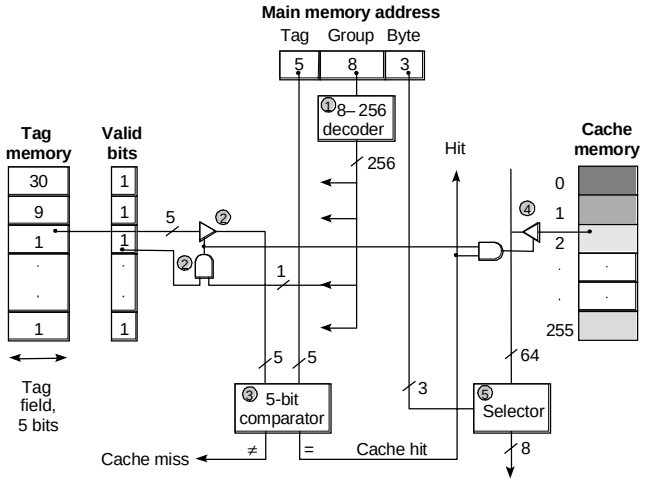


le bloc de mémoire
à l'adresse B reside dans
le bloc de cache
b = B mod (Nb. bloc dans le cache)



Cache à associativité directe - opération

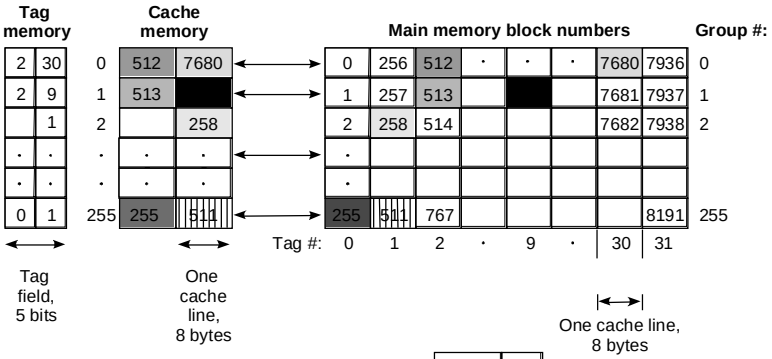
1. Sélectionner groupe de cache
2. Verifier si valide
3. Si oui, comparer Tag
4. Si même, lire le bloc
5. Sélectionner le mot, octet,...



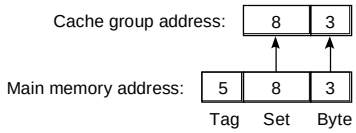
Très restrictif, si deux blocs actifs de mémoire résident dans le meme bloc de cache….

Cache à associativité par ensemble de blocs

Élargir le groupe de cache à associativitée directe à contenir plus d'un bloc, avec une recherche associative dans le groupe sélectionné

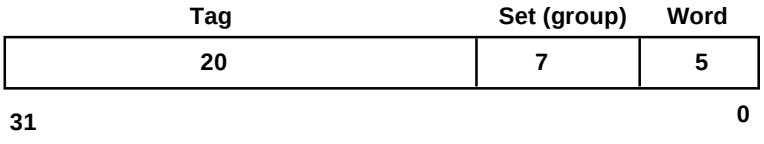


Associative par ensemble de 2 blocs



The Intel Pentium Cache

- 2 caches séparés de niveau 1 (L1)
  - intructions (P.I: 8k, assoc. par ens. de 2 blocs; P.Pro: 8k, assoc. par ens. de 4 blocs; P.II: 16k, assoc. par ens. de 4?)
  - donnees (P.I: 8k, assoc. par ens. de 2 blocs; P.Pro: 8k, assoc. par ens. de 2 blocs; P.II: 16k par ens. de 2 blocs)
- Assoc. par ens. de 2 dans Pentium I:
  - **8 K = 2^{13} octets**
  - **32 = 2^5 octets par ligne (bloc)**
  - **64 or 2^6 bytes per set (= 2 blocs x 32 octets/bloc)**
  - **$2^{13}/2^6 = 2^7$ = 128 groupes**
  - **32 - 5 - 7 = 20 bits pour le tag**



Politiques de lecture et d'écriture

- Sur succès d'accès:
  - écriture immédiate (*write through*) - mise-à-jour de la mémoire et du cache lors d'écriture
  - écriture au remplacement (*write back*) - écriture du bloc lors de l'enlèvement du cache.
    - bit "sale" indique si le bloc a été modifié lors de son séjour dans le cache
- Sur défaut d'accès:
  - Lecture
    - mise-à-jour du bloc de cache avec le transfert du mot au CPU
    - relancement de la lecture après avoir chargé le bloc au complet
  - Écriture
    - Allouer avant écrire (*write allocate*) - emmener le bloc dans cache et écrire
    - écriture sans allocation (*write non-allocate*) - écrire dans la mémoire sans mise-à-jour du cache

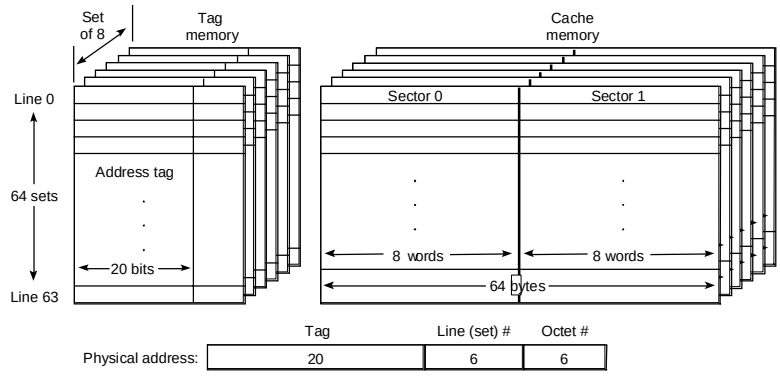
Politique de remplacement

- Associativité directe - pas besoin de politique…
- Associativité complète ou par ensemble
  - LRU (*Least Recently Used*)
    - mesurer l'usage à l'aide d'un compteur pare ensemble
      - à l'accès d'un bloc, remise à zero de son compteur
      - incrémenter par un chaque compteur ayant son compte plus petit que celui du compteur du bloc accédé
      - les autres restent inchangés
    - Lors que l'ensemble est plein et on doit remplacer, alors éliminer le bloc avec le compte le plus grand
    - approximation de un algorithme basé sur une pile qui toujours replace le No du bloc accédé au sommet de la pile; sur remplacement le bloc au fond de la pile est éliminé
  - Remplacement aléatoire fonctionne aussi assez bien

Performance de cache

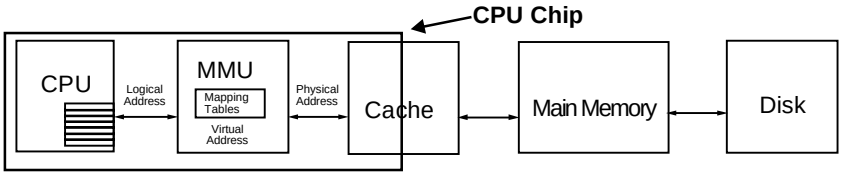
- **Avec le cache: $t_{accès} = h_{succès} \cdot t_{Cache} + (1 - h_{succès}) \cdot t_{Mémoire}$**
- **Faut connaitre les temps d'accès, le temps de remplissage du bloc de cache, et le taux de succès sur des programmes tests et ensuite calculer**
  - **le temps d'accès effectif**
  - **l'accéleration $S = T_{sans\ cache} / T_{avec\ cache}$**

Cache du PowerPC 601



- un cache unifié - instructions et données
- 32k octets, 64 groupes x 8 blocs / groupe associative par ensemble, un bloc = 16 mots de 4 octets chacun, organisés en 2 secteurs de 8 mots chacun pour faciliter la mise-à-jour
- Mise-à-jour en 2 tops d'horloge, 4 mots transférés chaque fois
- écriture sur remplacement, mais peut aussi programmer immédiate ou inhiber le cache complètement

Mémoire virtuelle



- MMU (*Memory Management Unit*) s'occupe de la gestion de l'espace de mémoire et de la traduction d'adresse virtuelle à adresse physique
- Adresse effective calculée par le processeur est une adresse (logique) dans un espace virtuel (plus grand que l'espace physique). Pour accéder à l'information à une adresse virtuelle V, il faut d'abord emmener l'info dans la mémoire physique et lui associer une adresse physique P. Tout accès à V est traduit à un accès à P.
- Cache peut voir soit adresses virtuelles soit adresses physiques (ce qui est plus habituel)

PowerPC 601 - adresses virtuelles / physiques

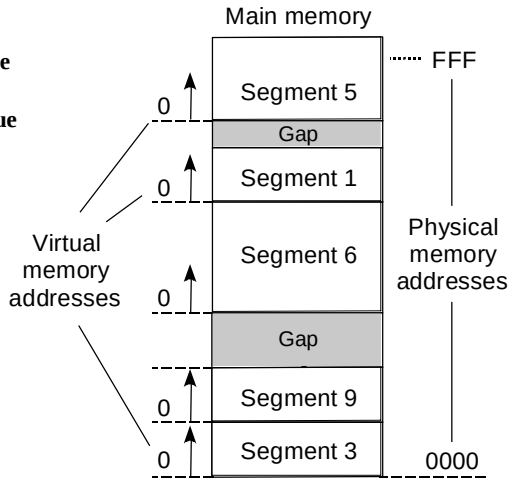
- CPU génère des adresses logiques de 32 bits
- Élargies par le MMU à 52 bits d'adresse virtuelle (pour identifier le processus à qui le programme appartient); cette adresse virtuelle est ensuite traduite à une adresse physique de 32 bits.
- Différents modèles de PPC ont des configurations différentes de taille d'adresses virtuelle et physique, selon les applications visées

Mémoire virtuelle - avantages

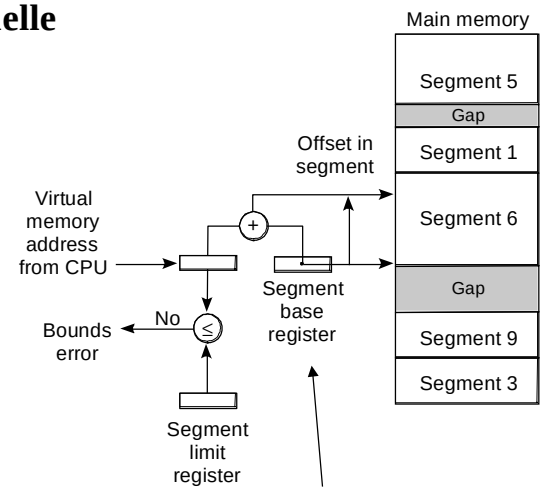
- Écriture des programme simplifiée, espace d'adresses assez grand
- Multiprogrammation améliore l'utilisation du CPU
- Donc pas de fragmentation du programme et recouvrements (*overlays*)
- Mémoire disque moins chère remplace mémoire principale plus couteuse
- Contrôle d'accès au niveau élémentaire des blocs - vérification des privilèges sur lecture, écriture, et exécution
  - Peut protéger contre erreur dans programme (OS, ou autres processus)
  - Peut protéger contre des attaques intentionnels
  - C'est l'origine des messages "*segmentation error*", "*bus error*"
- Inconvenient - pour des programmes avec une pauvre localité d'accès, la performance peut être très mauvaise… pour des programme complexes (simulation, CAO, etc., Faut programmer en tennant compte de la présence de mémoire virtuelle)

Gestion de mémoire par segmentation

Problème:
Fragmentation externe
Solution:
Compaction périodique des segments dans la mémoire



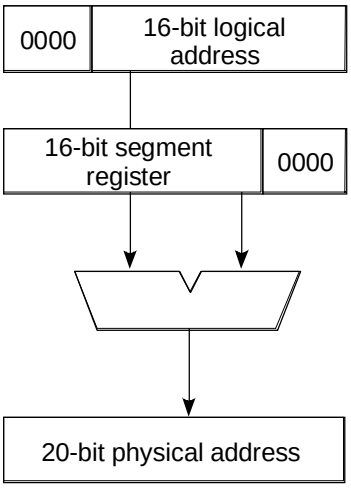
Mémoire virtuelle segmentée



Plusieurs registres de base

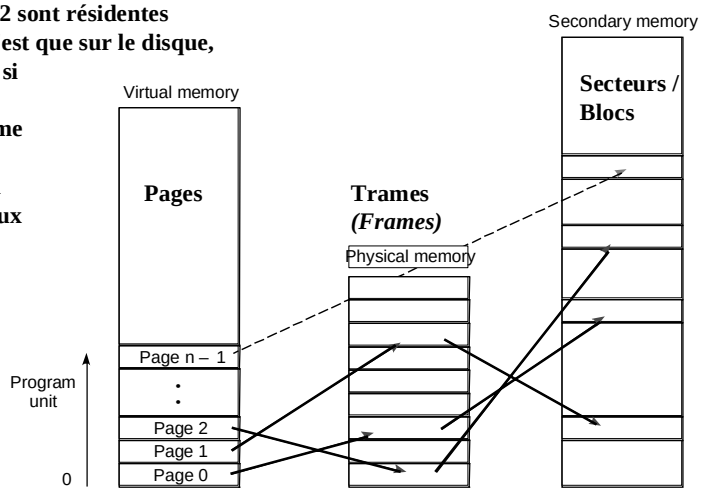
Segmentation dans l'Intel 8086

- Adresse "virtuelle" de 16 bits
- Adresse physique de 20 bits
- 4 registre de base de segments
  - CODE, DATA, STACK et EXTRA
- Utilisé pour augmenter l'espace d'adresses logiques
- Existe dans Pentium, mais aussi avec pagination et des segments plus grands



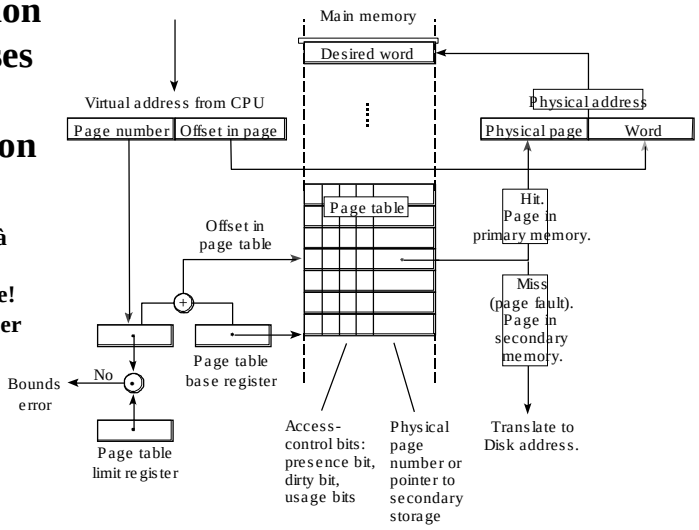
Gestion de mémoire par pagination

•Pages 0, 1, 2 sont résidentes
•Page n-1 n'est que sur le disque, sera charge si référée par le programme
•MMU gère l'affectation de trames aux pages et traduction d'adresses



Traduction d'adresses avec pagination

- **Table de pages à un niveau**
- **Peut être grande!**
- **Une table / usager**



Problème: Fragmentation interne (aux pages)

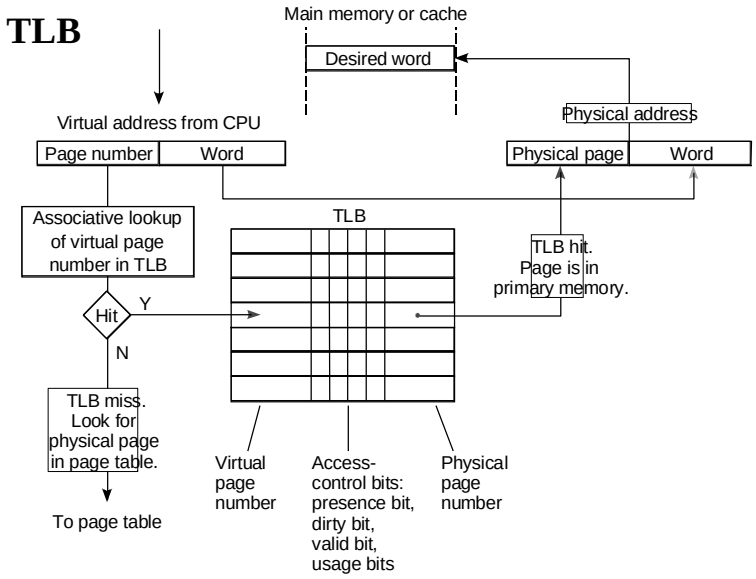
Placement et remplacement de pages

- Tables de pages: correspondance d'adresses directe
- Pages de programe - traduites via la table, résident dans des trames physiques sans ordre quelconque
- Table des pages risque d'être très grande
- Utilisation des tables à deux niveaux ou des tables de hashage qui ne contiennent que des pages référées
- Politique de remplacement généralement LRU (ou une approximation), masi aussi un simple bit de usage à l'intérieurr d'une fenêtre de temps
- Bits Valide et Sale nécessaires avec chaque trame

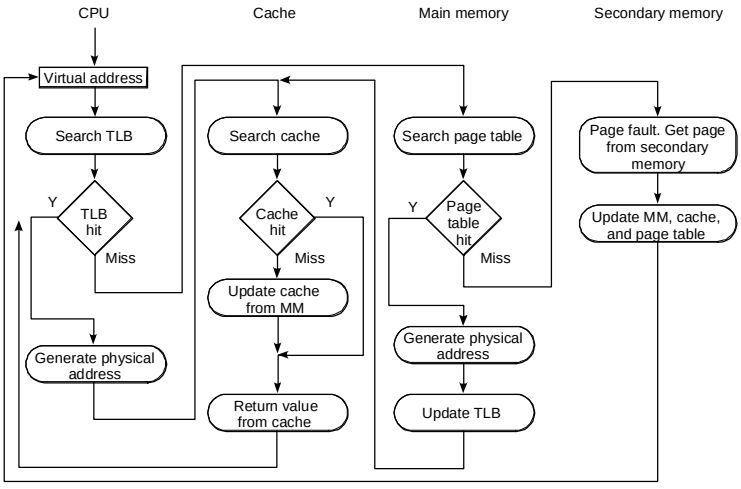
Problème avec la lenteur de la traduction d'adresses

- **Utilisation d'un tampon de traduction (*Translation Lookaside Buffer - TLB*) - une sorte de mémoire cache pour des adresses**
- **Complètement associatif**

TLB

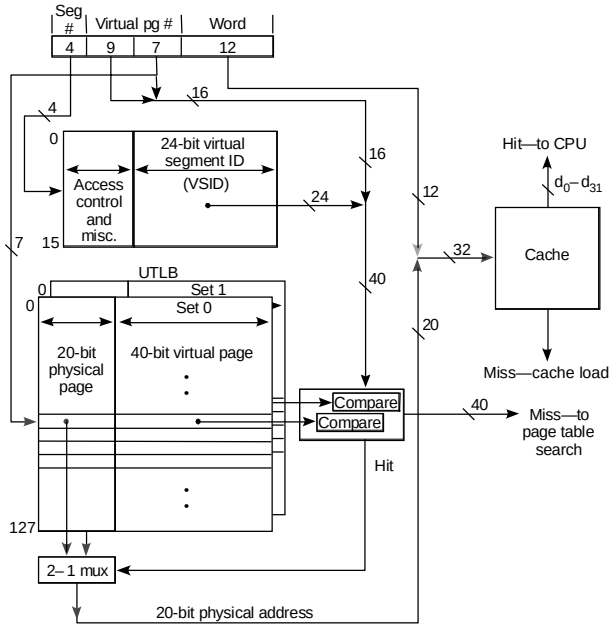


Sommaire de l'opération de la hiérarchie



Power PC MMU

Segments:
Blocs de 256MB!



Cache, mémoire virtuelle et E/S