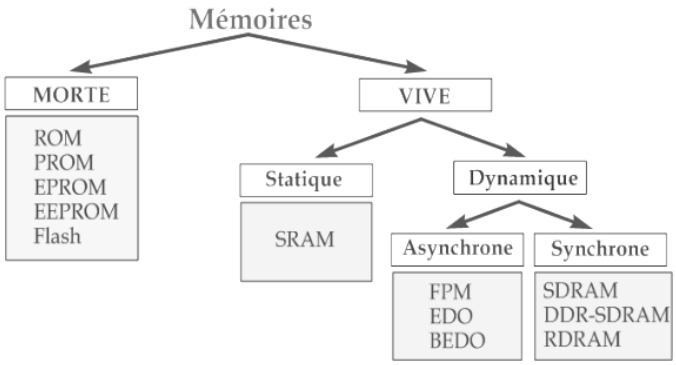
Chapitre 5 La mémoire

Introduction

L'accès à la mémoire est aussi un facteur clef dans la définition de la puissance de calcul réel d'un processeur. En effet, même si on dispose d'un processeur rapide pour exécuter des opérations, les résultats se feront attendre si les données d'entrée ne sont pas fournies à une bonne cadence. Or, les accès à la mémoire RAM (Random Access Memory) sont aujourd'hui bien plus lents que la cadence des processeurs. C'est pourquoi l'augmentation du débit mémoire augmente directement la puissance réelle de traitement du processeur.

Les différentes mémoires peuvent être classées comme indiqué sur la figure.



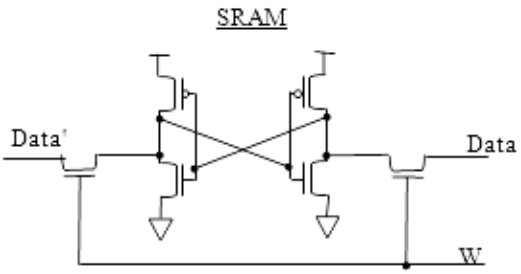
La mémoire vive ou RAM (Random Access Memory) pour mémoire à accès aléatoire, est une mémoire volatile, cela signifie que si l'on coupe l'alimentation, les données qu'elle contient sont perdues. Ses caractéristiques sont les suivantes :
elle sert à stocker les programmes exécutés par le processeur elle est accessible en lecture et en écriture
elle est organisée sous forme matricielle

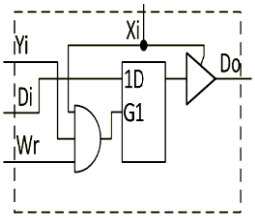
Il existe deux grandes familles de mémoires vives :
les RAM Statiques : SRAM
les RAM Dynamiques : DRAM

Technologie mémoire	Temps d'accès typique	$ par Go 2004	$ par Go 2015
SRAM	0.5 – 5 ns	4K – 10K	150
DRAM	50 – 70 ns	150 – 200	5 – 8
Disque magnétique	5.10^6 – 20.10^6 ns	0.50 – 2	0.05

les mémoires Statiques SRAM:

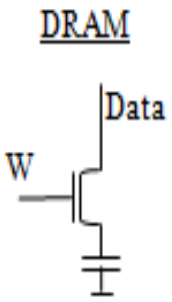
La cellule de mémoire RAM statique utilise une bascule pour stocker le bit
Nécessite 6 transistors et Contient les données tant que l'alimentation est fournie
L'information est mémorisée dans un latch
D'après la figure la mémorisation de Di si Wr • Yi • Xi est actif
La lecture de la cellule si Xi est actif





mémoire dynamique (DRAM)

La cellule mémoire RAM dynamique utilise un transistor MOS et un condensateur pour stocker le bit. Le condensateur détermine la valeur du bit : le bit vaut 1 si le condensateur est chargé, il vaut 0 dans le cas contraire. Le transistor gère l'accès au condensateur. L'Accès est plus lent que la SRAM.
On distingue plusieurs types de DRAM on cite que:



- Mémoire asynchrone : pour ce type de mémoire, l'intervalle de temps entre deux accès mémoire consécutif n'est pas régulier. Le processeur ne sait donc pas quand l'information qu'il attend est disponible et doit attendre (wait-state) que la mémoire lui transmette les données.
- Mémoire synchrone : la cadence de sortie des informations est régulière, on évite ainsi les états d'attente (wait state) du processeur.

Comparaison

Pour la DRAM, la mémorisation d'une charge sur une capacité, 1 transistor par bit donne une grande densité d'intégration le temps d'accès est relativement long: 50-100 ns utilisé comme mémoire principale.

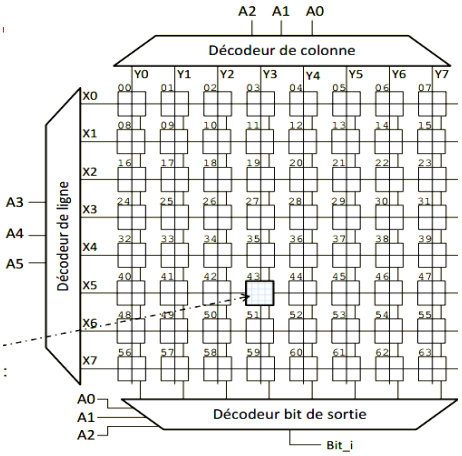
Pour la mémoire statique (SRAM) la mémorisation par bouclage de 2 portes: cher mais temps d'accès court: 5-10 ns; taille relativement petite

DRAM est Plus compact que la SRAM

SRAM facilement intégrée sur la même puce que le processeur, DRAM plus difficile

L'inconvénient des DRAM est que le condensateur possède une tendance naturelle à se décharger. Pour que l'information reste cohérente, on va devoir réaliser un rafraîchissement de la mémoire toutes les quelques millisecondes. Ce rafraîchissement consiste à lire et à réécrire la donnée.

La mémoire est une matrice composée de lignes L et de colonnes C, à chaque intersection (ligne,colonne) on trouve des bascules (SRAM), ou des condensateurs-transistors (DRAM).



La structure matricielle de la mémoire optimise la place occupée dans le circuit intégré

Une cellule est repérée par sa ligne et sa colonne D'où deux décodeurs au lieu d'un seul si la mémoire était sous forme linéaire

Exemple de sélection de cellule: Adresse = "101 011" => cell. 43 (Voir figure)

Noter que:

• Plus la mémoire est rapide, plus elle consomme : Gestion mémoire basse énergie: solutions

• éteindre ce qui ne sert pas--→Mémoire en bancs

• Plus une mémoire est grande, plus son temps d'accès est long

Critères d'une mémoire

Les principaux critères d'une mémoire performante sont liés au type d'application où elle sera intégrée. On note parmi ces critères :

-le **temps d'accès :** est le temps entre l'envoi à la mémoire d'une adresse et la réception en lecture de la donnée correspondante ; temps nécessaire pour sauvegarder un état ou une donnée

– le **temps de cycle:** est le temps s'écoulant entre deux opérations sur la mémoire (toute combinaison de lectures et écritures) ;

– le **débit :** est le nombre d'octets que la mémoire peut fournir par unité de temps. mesure ensuite le flux de données transmises pendant le régime stable c'est-à-dire une fois la latence écoulée, Il est fonction du temps de cycle de la mémoire, de la largeur des mots, exprimé en octets, fournis par la mémoire et du type d'accès (accès à un mot individuel ou accès à une série de mots successifs).

Quand il est question de vitesse d'accès à la mémoire, il faut distinguer la *latence* et le *débit*. La latence est le temps qui s'écoule entre la demande des données et l'arrivée de la première donnée.

-capacité

-Le coût de fabrication

-L'énergie consommée (qui dépend en particulier des tensions et des courants appliqués liée à leur latence)

-La durée de rétention des données

-L'endurance, soit la capacité de maintenir dans le temps un état

-La miniaturisation,

Jusqu'à présent, chacune des technologies citées précédemment est confrontée à des défis technologiques variés. La taille importante des SRAMs, et les courants de fuite des capacités des DRAMs qui réduisent la rétention et qui exigent un rafraîchissement régulier de ces mémoires, restent un vrai challenge. En outre, la limitation majeure de ces mémoires est la volatilité, ce qui augmente potentiellement la consommation statique du système.

Parmi les mémoires non volatiles, la technologie flash, inventée dans les années 1980. En fait, pour des cellules de petites dimensions, des limitations électriques se présentent telles que la réduction du

nombre d'électrons stockés pour sauvegarder l'information ainsi que la fiabilité. Cette dernière diminue lorsque la taille de la cellule diminue. Enfin, la programmation de ces mémoires exige une tension de l'ordre 3.5 à 10 V, ce qui augmente forcément la consommation totale ainsi que la surface, car des blocs spécifiques sont rajoutées afin de générer ces tensions.
Vis- à -vis de ces limitations, changer le mode de stockage de l'information est devenu une préoccupation prépondérante. Le but est d'intégrer ces mémoires alternatives ou « émergentes » dans les systèmes embarqués pour des nœuds technologiques avancés.

Les mémoires émergentes non volatiles

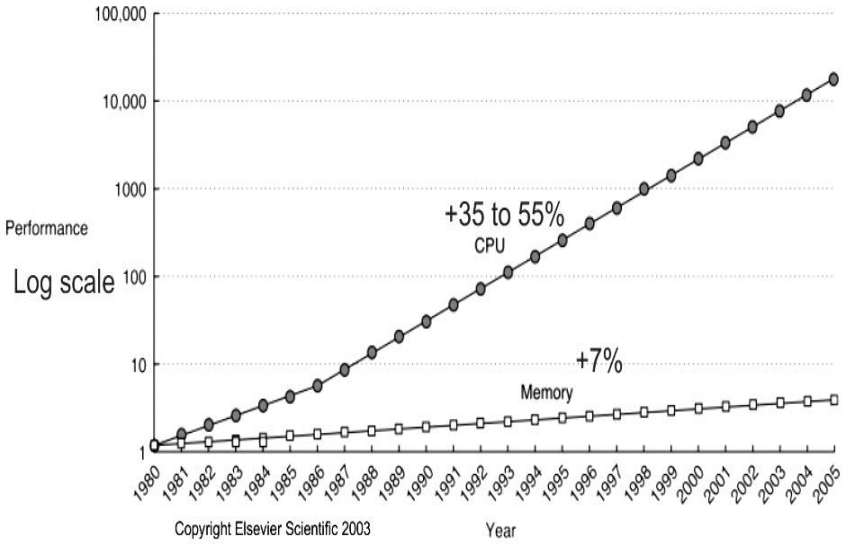
Ces mémoires sont en développement depuis longtemps avec l'objectif de prendre le relais des mémoires actuelles citées ci-dessus.
Le développement de ces mémoires met en jeu différents principes physiques (magnétorésistance, ferroélectricité, ...) qui permettent de stocker l'information selon un état distinct de la résistance de la cellule. Ces nouvelles technologies ne s'appuient plus sur le stockage des charges. Les données sont plutôt représentées sous la forme d'un état de résistance selon le phénomène physique adopté.

Problème du memory wall

La latence mémoire n'est pas seulement constituée par le temps d'accès a la mémoire mais elle prend en considération: la translation d'adresse, la traversée des broches du processeur et le bus externe et le multiplexage si plusieurs bancs mémoires existent.

- Exemple dans le cas ou la Latence mémoire principale (ordre de grandeur) est de 100-200 ns avec un processeur à 1 GHz, ce qui donne un retard de 100 à200 cycles CPU.

En effet, les processeurs deviennent plus rapides plus rapidement que la mémoire (notez l'échelle logarithmique dans la figure)

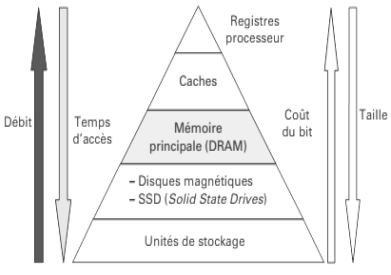


• Amélioration de la vitesse du processeur: 35% à55%
• Amélioration de la latence de la mémoire: 7%

La mémoire devient le frein à l'augmentation des performances des processeurs: problème connu sous le nom de memory wall: Mur mémoire).

Temps de cycle processeur < temps d'accès mémoire
Requête mémoire du processeur =processeur inactif plusieurs cycles

Pour trouver une solution à ce problème et surpasser cet obstacle la notion d'Hiérarchie de mémoire a été introduite.



Hiérarchie de mémoire

Au sein de l'unité de calcul d'un système électronique, le choix technologique des mémoires est un élément clef. Les caractéristiques et les performances des mémoires ont un impact capital sur le fonctionnement intrinsèque du système. Aujourd'hui, la mémoire idéale, autrement dit la mémoire qui combine à la fois une endurance infinie, une densité élevée, une consommation réduite, un coût très faible avec une vitesse assez importante n'existe pas. C'est pourquoi les technologies actuelles reposent sur un compromis entre ces différentes performances au regard de l'application visée. La solution est donc

de ne pas se limiter à une technologie de mémoire mais d'adopter plutôt une hiérarchie de mémoire de façon bien définie. Les programmes standard ont souvent des comportements prédictibles soit temporellement, soit spatialement, de sorte que les accès mémoires réalisés par le processeur pour les exécuter ne se font pas au hasard.

les registres sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et résultats intermédiaires.

Le cache est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.

la mémoire centrale contient les programmes (code + données) et est plus lente que les deux mémoires précédentes.

La mémoire d'appui est l'équivalent de la mémoire cache pour la mémoire de masse.

la mémoire de masse est un support de stockage généralement de grande capacité et sert au stockage et à l'archivage des informations.

Hiérarchie des composants de la mémoire

• Composants supérieurs: Rapide ,Petit, Cher
• Composants inférieurs : Lent, grand, pas cher
• Connecté par des bus qui ont également des problèmes de latence et de bande passante
• Données les plus fréquemment consultées en M1
• Déplacer des données de haut en bas de la hiérarchie
• Optimiser le temps d'accès moyen

Le but: Donner à l'utilisateur l'illusion d'avoir la capacité de la mémoire la moins chère (Disque) et la vitesse de la mémoire la plus rapide (cache interne)

Principe de la mémoire cache

Mémoire très rapide, mais de petite taille, placée entre le processeur et la mémoire principale. Le processeur essaie d'accéder un mot d'abord dans la cache, avant de passer à la mémoire principale. En cas d'échec (miss), le mot est gardé dans la cache pour un accès futur. En cas de succès (hit), la mémoire principale n'est pas accédée. La fréquence des succès (hit rate) dépend de la taille de la cache et de l'algorithme exécuté par le contrôleur de cache.

Généralement conçu avec SRAM, Généralement sur la même puce que le processeur espace limité, tellement plus petit que la mémoire principale hors puce, accès plus rapide (1 cycle vs plusieurs cycles pour la mémoire principale).

On distingue Plusieurs choix de conception selon le mappage de cache, stratégies de remplacement et techniques d'écriture.

Localité des références mémoire

C'est une Propriété empirique des programmes dans le cas d'une application réelle, à quelques exceptions près est c'est la raison pour laquelle que l'hiérarchie de mémoire à trouver son succès. :

En pratique, on observe fréquemment des taux de succès moyens de l'ordre de 80 à 90 %. Cette performance s'explique par les fortes propriétés de localité des programmes. Il existe deux types :

Localité spatiale: probabilité d'accès à une adresse voisine: Plus susceptible de référencer des données à proximité de données récemment référencées.

Proactif: mettez en cache de gros morceaux de données pour inclure les données à proximité.

• instructions : les programmes sont des séquences d'instructions, rompues uniquement par des if, for, etc. : si la 1ère instruction d'une séquence est lue, les suivantes le seront à coup sûr rapidement
• données : les données sont souvent organisées en tableaux, souvent parcourus séquentiellement : si un élément de tableau est accédé, les suivants le seront très probablement rapidement

le code d'un programme s'exécute toujours à l'intérieur de petites zones répétées de mémoire (des blocs correspondant à des boucles ou/et des sous-programmes)

Si l'adresse A est référencée a T , alors forte probabilité de référencer A + a à T + t , avec a petit et t petit

Localité temporelle

Localité temporelle: probabilité d'accès aux mêmes adresses successivement: Les données récemment référencées seront probablement à nouveau référencées prochainement

• Réactif: cache les données récemment utilisées dans une petite mémoire rapide

• instructions : les instructions du corps d'une boucle for, accédées lors de la 1ère itération, vont probablement être ré-accédées rapidement car une boucle effectue généralement plus d'une itération.

• données : toutes les variables de boucle (i, somme, etc.) sont dans le même cas, pour les mêmes raisons ; plus généralement toutes les variables locales de fonctions.

Les blocs s'exécutent en séquences très proches (il y a plus de chances d'accéder à une position de mémoire utilisée il y a 10 cycles qu'à une autre utilisée il y a 10000 cycles)

Si l'adresse A est référencée a T , alors forte probabilité de référencer A à T + t , avec t petit (réutilisation).

localité temporelle : si une donnée située à une adresse A en mémoire est référencée, elle a une forte probabilité d'être référencée à nouveau dans un court intervalle de temps.

localité spatiale : si une donnée située à une adresse A en mémoire est référencée, il y a une forte probabilité de référencer une donnée située à une adresse voisine dans un court intervalle de temps.

Si les accès à la mémoire demandent 3 wait states et la fréquence de succès est de 90%, le nombre moyen de waits states est de 10% x 3 = 0.3 wait state / cycle de mémoire

la chute de la fréquence de succès à 80% fait doubler cette valeur

Si une instruction demande 2 cycles sans wait state et 5 cycles avec, et si la fréquence de succès est de 80%, l'exécution de 10 instructions demande:

(10 x 0.8 x 2) + (10 x (1 - 0.8) x 5) = 26 cycles dont 16 sont faits avec la cache.

C'est-à-dire: le processeur passe 40% du temps (10/26) à chercher 20% du code: pendant ce temps le bus du système est occupé par les transferts entre le processeur et la mémoire

Accès: lecture ou écriture dans le cache

• Hit: données souhaitées trouvées dans le cache (succès)

• Miss: données souhaitées introuvables dans le cache (defaut)(Doit provenir d'un autre composant)

• Remplir: action de mise en cache des données

•(taux-miss)%: c'est le nombre de default divisé par le nombre d'accès

• $t_{accès}$: temps de vérifier le cache. En cas de hit, on termine.

• t_{miss}: temps de lire les données dans le cache

Mesure de performance: temps d'accès moyen= t_{avg}= t_{access}+ (%miss* t_{miss})

$Latence_{avg}$=$latence_{hit}$+ (%miss*$latence_{miss}$)

• La métrique ultime du cache est tavg. La capacité du cache et les circuits déterminent approximativement taccès, alors que les structures de mémoire de niveau inférieur déterminent tmiss

Le moyen le plus simple de réduire le taux du miss c'est d'augmenter la capacité

Cependant taccès augmente, car la Latence est proportionnelle à sqrt (capacité). Donc Compte tenu de la capacité, manipuler le taux de miss en changeant d'organisation du cache.

(On définit ici la notion de Working set: c'est l'ensemble de travail: l'ensemble des instructions et données d'un programme utilisé activement)

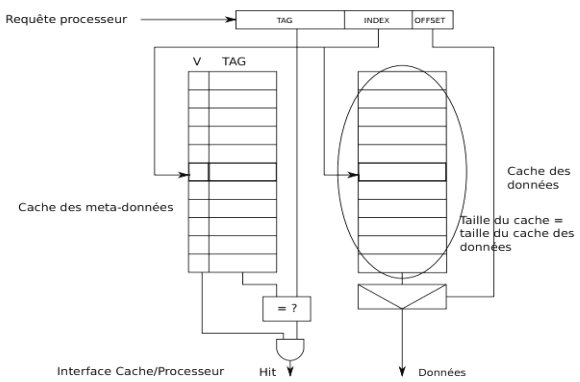
Organisation et fonctionnement

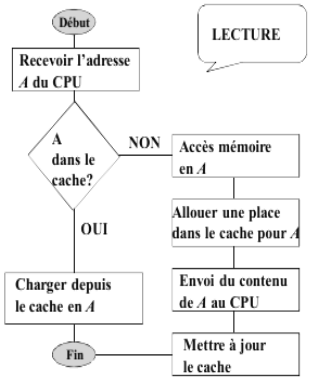
La taille du cache étant nettement inférieure à celle de la mémoire, le cache ne pourra contenir qu'une partie des données de la mémoire. En conséquence, la position d'une donnée dans le cache ne correspond donc plus à son adresse, contrairement à ce qui se passe en mémoire principale. Avec chaque donnée stockée dans le cache, il est donc nécessaire de conserver également son adresse. Aussi, un cache comporte deux sous-composants principaux : la table des étiquettes et la table des données. La table des étiquettes stocke les adresses des données, et la table des données stocke les données elles-mêmes.

Structure générale d'un cache

Contrairement à la mémoire principale, accessible par adresse, le cache est généralement accessible par le contenu; par conséquent, c'est souvent appelée mémoire adressable par contenu CAM content adressable memory).

La correspondance entre les blocs de cache et MM les blocs sont créés par un algorithme de mappage de cache





Le mapping

La mémoire cache ne pouvant contenir toute la mémoire principale,il faut définir une méthode indiquant à quelle adresse de la mémoire cache doit être écrite une ligne de la mémoire principale. Cette méthode s'appelle le mappage ou correspondance (mapping).

Compromis entre la complexité du cache et sa capacité à retenir un maximum de lignes. Il existe trois types de mapping :

1-Cache à correspondance directe (*direct-mapped*)
une ligne n'a qu'un seul emplacement possible dans le cache, conflits entre différentes lignes
2-Cache complètement associatif (*fully-associative*)
une ligne peut être à n'importe quel emplacement dans le cache l'espace du cache est mieux utilisé
3-Associatif par ensembles de N lignes (*N-way set-associative*)
intermédiaire entre direct-mapped et fully-associative

Choix de l'élément à remplacer lors d'un conflit (miss):

Les caches à ensembles associatifs présentent un nouveau choix de conception
En cas de manque de cache, quel bloc de l'ensemble remplacer (expulser)? On parle de Techniques de remplacement:
• FIFO (first-in first-out) ordre d'ancienneté : facile à implémenter en matériel (un index par cache mais correspond le moins bien à la localité temporelle

• LRU (least recently used) S'adapte à la localité temporelle, LRU = le moins susceptible d'être utilisé à l'avenir
D'un point de vue algorithmique, on peut maintenir l'ordre LRU par une liste chaînée chaque ligne accédée est placée en tête de liste la ligne en queue de liste est la moins récemment utilisée plusieurs listes chaînées les lignes qui ont le même index cache sont dans la même liste si cache FA, une seule liste globale
Ajouter un champ LRU à chaque ensemble • Les données LRU sont codées • success (hit)? mettre à jour MRU • Bits LRU mis à jour à chaque accès: chaque ligne de la cache (même ensemble mais voies différentes) possède des bits pour indiquer l'ordre d'utilisation des voies.

Exemple: Pour une cache à 4 voies il y a 24 possibilités (4!): il faudrait 5 bits de codage au minimum. Pour une cache à 16 voies il faudrait 45 bits...
Assez facile à implémenter en matériel (un compteur d'âge par mot), correspond assez bien à la localité temporelle
En pratique, on utilise souvent LRU, qui est un bon compromis.
• Aléatoire: favorise une allocation uniforme ,remplacer une ligne au hasard parmi les N
• LFU (Least Frequently Used) : on remplace le mot le moins fréquemment utilisé dans le cache répond idéalement à la propriété de localité temporelle mais coûteuse à implémenter en matériel (add. + diviseur par mot)

Un bloc entier de données est copié après un échec (miss) car le principe de localité nous dit qu'une fois élément est accédé, il est probable qu'une donnée à proximité de cet élément sera bientôt nécessaire.
Lorsqu'une ligne doit être remplacée ou lors d'une écriture en mémoire le cach e doit assurer la cohérence avec la mémoire principale.
Exemple LRU: Supposons que les lignes A,B,C,D,E aient le même index cache et que initialement le cache contient DBCA
La table résume les différents accès et les blocs à remplacer dans le cas du LRU

Ligne accédée	Cache avant	Hit / miss ?	Ligne évincée	Cache après
A	DBCA	hit		ADBC
B	ADBC	hit		BADC
C	BADC	hit		CBAD
D	CBAD	hit		DCBA
E	DCBA	miss	A	EDCB
A	EDCB	miss	B	AEDC
E	AEDC	hit		EADC
B	EADC	miss	C	BEAD

Paramètres du cache les plus importants en termes de performances:
Taille totale du cache: nombre total d'octets de données que le cache peut contenir, les tag (étiquette), bits valides et autres éléments de ménage non inclus dans le total
Degré d'associativité:
Taille du bloc de données: les caches plus grandes fournissent des taux de miss plus faible mais au prix d'un temps d'accès plus élevé
Exemple
cache a 2 Koctet: taux miss = 15%, prix du hit = 2 cycles, cout miss = 20 cycles
coût moyen d'accès à la mémoire = (0.85 * 2) + (0.15 * 20) = 4.7 cycles
cache 4 Koctet: taux miss = 6.5%, prix du hit = 3 cycles, le coût du miss ne changera pas en moyenne.
coût d'accès à la mémoire = (0.935 * 3) + (0.065 * 20) = 4.105 cycles (amélioration)
cache 8 Kbyte: taux miss = 5.565%, prix du hit = 4 cycles, le coût du miss ne changera pas en moyenne.
coût d'accès à la mémoire = (0.94435 * 4) + (0.05565 * 20) = 4.8904 cycles (pire)
De nombreux systèmes modernes utilisent des caches séparés pour les données et les instructions. C'est ce qu'on appelle une cache de Harvard. La plupart des programmes ont une localité d'instruction plus élevée que la localité de données

Les performances du cache peuvent également être améliorées en ajoutant un petit cache associatif pour contenir les blocs qui ont été expulsés récemment. C'est ce qu'on appelle un cache victime.