

La mémoire cache

Points clés

- ◆ La mémoire est organisée de manière hiérarchique. Au niveau supérieur (le plus près du processeur) se trouvent les registres du processeur. Ils sont suivis d'un ou de plusieurs niveau(x) de cache, libellés L1, L2, etc. Vient ensuite la mémoire, généralement composée de RAM dynamique (DRAM, *Dynamic Random-Access Memory*). Ces différents niveaux sont internes au système. La hiérarchie se poursuit par la mémoire externe : disque dur fixe, suivi de quelques niveaux de médias amovibles comme les cartouches ZIP, les disques optiques et les bandes.
- ◆ À mesure que l'on descend dans la hiérarchie, le rapport coût/bit et le temps d'accès diminuent alors que la capacité augmente. Il serait intéressant d'exploiter uniquement la mémoire la plus rapide, mais cette dernière étant la plus coûteuse, on préfère réduire les coûts au détriment du temps d'accès en exploitant davantage la mémoire la plus lente. La solution consiste à organiser les données et les programmes en mémoire de manière que les mots mémoire nécessaires se trouvent dans la mémoire la plus rapide.
- ◆ Le processeur accède essentiellement à des emplacements mémoire qu'il a récemment utilisés et dont le cache conserve automatiquement une copie. Si ce dernier est correctement conçu, le processeur demandera principalement des mots mémoire qui se trouvent déjà dans le cache.

Bien que son concept soit apparemment simple, la mémoire présente plus de types, de technologies, d'organisations, de performances et de coûts que n'importe quelle autre fonctionnalité d'un ordinateur, sans pourtant qu'aucune technologie ne satisfasse pleinement ses exigences. On en arrive par conséquent à un ordinateur équipé d'une hiérarchie de sous-systèmes mémoire, certains internes (accessibles directement par le processeur) et d'autres externes (accessibles au processeur *via* le module d'E/S).

Ce chapitre et le chapitre suivant se concentrent sur les éléments de la mémoire interne, alors que le chapitre 6 est consacré à la mémoire externe. Pour commencer, la première section examine un élément essentiel de tout ordinateur actuel : la mémoire cache.

4.1 Aperçu du système mémoire

Caractéristiques des systèmes mémoire

Pour simplifier le sujet complexe de la mémoire, nous allons classer les différents systèmes en fonction de leurs caractéristiques, dont les plus importantes sont répertoriées dans le tableau 4.1.

Tableau 4.1 • Principales caractéristiques des systèmes mémoire.

Emplacement	Performances
Processeur	Temps d'accès
Interne (principale)	Temps de cycle
Externe (auxiliaire)	Débit de transfert
Capacité	Type physique
Taille du mot	Semi-conducteur
Nombre de mots	Magnétique
Unité de transfert	Optique
Mot	Magnéto-optique
Bloc	Caractéristiques physiques
Méthode d'accès	Volatiles/non volatiles
Séquentielle	Effaçable/non effaçable
Directe	Organisation
Aléatoire	
Associative	

Le terme **emplacement** du tableau 4.1 indique si la mémoire est interne ou externe à l'ordinateur. Si la mémoire interne correspond souvent à la mémoire principale, il en existe d'autres formes. Comme nous le verrons dans les prochains chapitres, le processeur a besoin de sa propre mémoire locale, sous la forme de registres (voir, par

exemple, la figure 2.3), ainsi que sa partie unité de contrôle. Le cache est une autre forme de mémoire interne. La mémoire externe, à laquelle le processeur accède *via* les contrôleurs d'E/S, se compose de périphériques de stockage, comme les disques et les bandes.

La **capacité** de la mémoire interne s'exprime en octets (1 octet = 8 bits) ou en mots. Pour la mémoire interne, les longueurs classiques de mots sont 8, 16 et 32 bits. On se réfère à la capacité de la mémoire externe essentiellement en termes d'octets.

L'**unité de transfert** de la mémoire interne correspond au nombre de lignes de données qui entrent et sortent du module mémoire. Elle peut être égale à la longueur du mot, mais est souvent plus grande (64, 128 ou 256 bits). Pour expliquer ce point, clarifions trois concepts relatifs à la mémoire interne :

- **Mot** : unité « naturelle » de l'organisation de la mémoire. La taille du mot est généralement égale au nombre de bits utilisés pour représenter un nombre et à la longueur de l'instruction. Il existe malheureusement de nombreuses exceptions. Par exemple, la longueur de mot du CRAY C90 est de 64 bits, mais il utilise une représentation entière 46 bits. Le VAX dispose d'une variété surprenante de longueurs d'instruction, exprimées en multiples d'octets et une taille de mot de 32 bits.
- **Unités adressables** : si sur certains systèmes, l'unité adressable est le mot, une grande majorité des systèmes permettent l'adressage au niveau de l'octet. La relation entre la longueur en bits A d'une adresse et le nombre N d'unités adressables est $2^A = N$.
- **Unité de transfert** : pour la mémoire principale, il s'agit du nombre de bits lus ou écrits dans la mémoire en une seule fois. Il n'est pas nécessaire qu'elle soit équivalente à un mot ou à une unité adressable. Pour la mémoire externe, les données sont souvent transférées en unités plus grandes qu'un mot. Dans ce cas, on parle de blocs.

La **méthode d'accès** aux unités de données est une autre distinction des types de mémoire :

- **Accès séquentiel** : la mémoire est organisée en unités de données, appelés enregistrements, auxquels on accède par une séquence linéaire spécifique. On se sert des informations d'adressage stockées pour séparer les enregistrements et assister le processus de recherche. On utilise un mécanisme de lecture/écriture commun qu'il faut déplacer de son emplacement en cours vers l'emplacement souhaité en passant et en rejetant chaque enregistrement intermédiaire. Ainsi, le temps d'accès à un enregistrement arbitraire est variable. Les bandes, dont nous parlerons au chapitre 6, exploitent l'accès séquentiel.
- **Accès direct** : comme pour l'accès séquentiel, l'accès direct implique un mécanisme de lecture/écriture commun. Cependant, les blocs ou enregistrements possèdent une adresse unique associée à un emplacement physique. Pour y accéder, on utilise l'accès direct pour s'en rapprocher et la recherche séquentielle, le comptage ou l'attente pour atteindre l'emplacement final. Le temps d'accès est

également variable. Les disques, que nous étudierons au chapitre 6, utilisent l'accès direct.

- **Accès aléatoire** : chaque emplacement adressable en mémoire possède un mécanisme d'adressage unique physiquement câblé. Le temps d'accès à un emplacement donné est constant et indépendant de la séquence des accès précédents. En conséquence, on peut sélectionner n'importe quel emplacement au hasard, l'adresser et y accéder directement. La mémoire principale et certains systèmes de cache sont à accès aléatoire.
- **Associatif** : ce type de mémoire à accès aléatoire permet de comparer les emplacements de bits d'un mot pour trouver une correspondance spécifique et de le faire simultanément pour tous les mots. Ainsi, on ne se sert pas de l'adresse d'un mot mais d'une partie de son contenu pour le retrouver. Comme pour la mémoire à accès aléatoire, chaque emplacement possède son propre mécanisme d'adressage. En outre, le temps d'accès est constant et indépendant de l'emplacement ou des configurations d'accès précédentes. Les mémoires cache peuvent employer l'accès associatif.

Du point de vue utilisateur, les deux caractéristiques les plus importantes de la mémoire sont la capacité et la **performance**, dont on exploite trois paramètres :

- **Temps d'accès (latence)** : pour l'accès aléatoire, il s'agit du temps nécessaire pour effectuer une opération de lecture ou d'écriture, autrement dit, le temps entre l'instant où on présente une adresse à la mémoire et celui où les données sont stockées ou disponibles. Pour l'accès non aléatoire, le temps d'accès équivaut au temps nécessaire au positionnement du mécanisme de lecture/écriture à l'emplacement approprié.
- **Temps de cycle mémoire** : ce concept s'applique principalement à l'accès aléatoire et correspond au temps d'accès auquel on ajoute le temps nécessaire avant qu'un second accès puisse commencer. Ce temps supplémentaire sert à attendre la fin des transitions sur les lignes de signaux ou à régénérer des données si elles sont lues de manière destructive. Notez que le temps de cycle mémoire est en relation avec le bus système, mais pas avec le processeur.
- **Débit de transfert** : il s'agit du débit auquel les données peuvent être transférées depuis ou vers une unité de mémoire. Pour l'accès aléatoire, il est égal à $1/(\text{durée du cycle})$.

Voici l'équation pour l'accès mémoire non aléatoire :

$$T_N = T_A + \frac{N}{R}$$

où

T_N = Temps moyen de lecture ou d'écriture de N bits

T_A = Temps d'accès moyen

N = Nombre de bits

R = Débit de transfert, en bits par seconde (b/s)

On a utilisé une grande variété de **type de mémoire physiques**, dont les plus courants actuellement sont la mémoire à semi-conducteurs, la mémoire à surface magnétique, utilisée par les disques et les bandes, et la mémoire optique et magnéto-optique.

Plusieurs **caractéristiques physiques** de stockage sont importantes. Dans une mémoire volatile, les informations se détériorent naturellement ou sont perdues en cas de coupure de l'alimentation électrique. Pour la mémoire non volatile, les informations enregistrées sont conservées sans détérioration, à moins d'être délibérément modifiées, sans qu'aucune alimentation électrique ne soit nécessaire à leur conservation. La mémoire à semi-conducteurs entre dans ces deux catégories. Lorsqu'elle n'est pas effaçable, on l'appelle ROM (*Read-Only Memory*, mémoire morte). On ne peut pas altérer la mémoire non effaçable, excepté en détruisant l'unité de stockage. Par la force des choses, une mémoire non effaçable doit également être non volatile.

L'**organisation** de la mémoire à accès aléatoire est un problème essentiel de conception. Par organisation, on entend la disposition physique des bits formant les mots. Comme nous allons le voir à présent, on n'exploite pas toujours l'agencement le plus évident.

Hiérarchie mémoire

Les contraintes de conception d'une mémoire se résument à trois facteurs : son coût, sa vitesse et son prix.

Le problème de la quantité est sans limites : si la capacité existe, les applications seront développées pour l'exploiter. Il est, en un sens, plus facile d'aborder la question de la vitesse. Pour obtenir de meilleures performances, la mémoire doit pouvoir suivre le rythme du processeur. Autrement dit, on ne doit pas attendre les instructions ou opérandes dont il a besoin pour exécuter des instructions. On doit également tenir compte de la dernière question. Le prix de la mémoire doit être raisonnable et en relation avec celui des autres composants.

Comme on peut s'y attendre, il faut faire un compromis entre ces trois caractéristiques : coût, capacité et temps d'accès. À tout moment, dans la gamme de technologies qui implémentent les systèmes mémoire, on doit tenir compte des relations suivantes :

- Plus le temps d'accès est court, plus le prix par bit est élevé.
- Plus la capacité est importante, plus le prix par bit est faible.
- Plus la capacité est importante, plus le temps d'accès est long.

Le concepteur est confronté à un dilemme : il voudrait exploiter les technologies qui proposent une capacité de mémoire importante, d'une part parce qu'il a besoin de cette capacité et d'autre part en raison du faible coût par bit. En revanche, pour répondre aux besoins en matière de performance, le concepteur doit faire appel à des mémoires plus chères, de capacité moins importante et à des temps d'accès plus courts.

La solution consiste à ne pas se limiter à un composant ou à une technologie uniques, mais à employer une **hiérarchie mémoire** (voir figure 4.1). Voici ce que l'on rencontre à mesure que l'on descend dans cette hiérarchie :

- a. Baisse du coût par bit.
- b. Augmentation de la capacité.
- c. Augmentation du temps d'accès.
- d. Baisse de la fréquence d'accès du processeur à la mémoire.

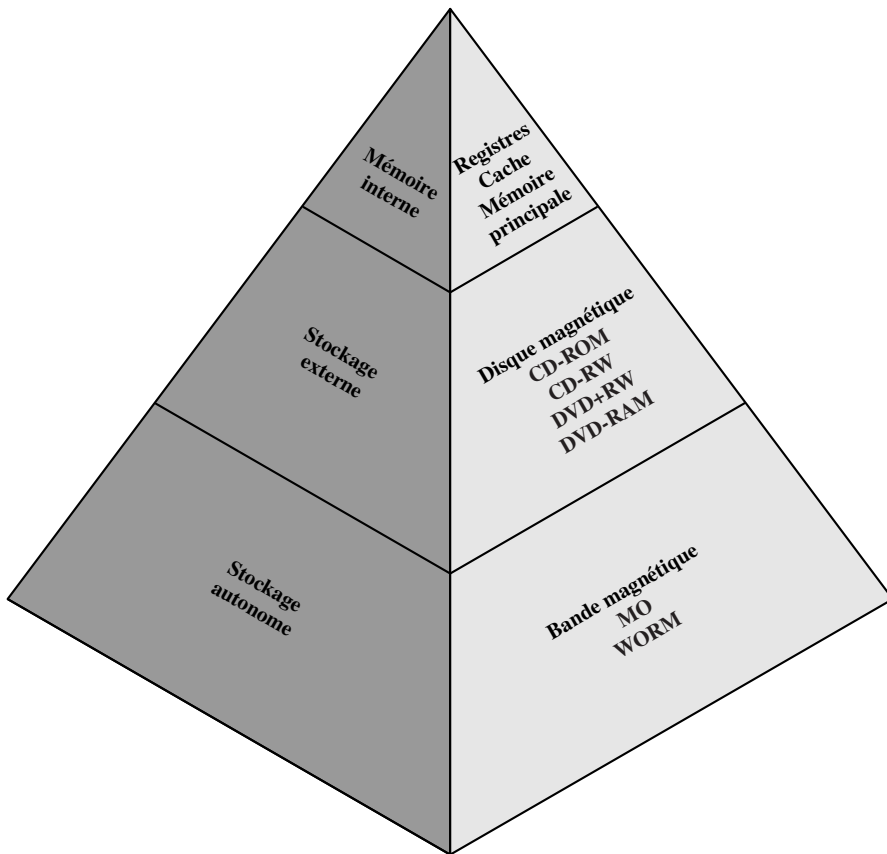


Figure 4.1 • Hiérarchie de la mémoire.

Ainsi, les mémoires plus petites, moins chères et plus rapides sont alors supplantées par les mémoires plus grandes, moins chères et plus lentes. L'élément (d) est la clé du succès de cette organisation : diminuer la fréquence d'accès. Nous détaillerons ce concept lorsque nous étudierons le cache, plus loin dans ce chapitre, et la mémoire virtuelle au chapitre 8, mais voici une brève explication.

Supposons que le processeur ait accès à deux niveaux de mémoire. Le niveau 1 contient 1 000 mots et son temps d'accès est de $0,01 \mu\text{s}$. Le niveau 2 contient 100 000 mots et son temps d'accès est de $0,1 \mu\text{s}$. Le processeur accède directement à un mot du niveau 1. Pour le niveau 2, le mot est transféré au niveau 1 avant que le processeur puisse y accéder. Pour plus de simplicité, nous ignorons le temps dont le processeur a besoin pour déterminer si le mot se trouve au niveau 1 ou 2. La courbe de la figure 4.2 illustre cette situation. Le temps d'accès moyen à une mémoire à deux niveaux y prend la forme d'une fonction du taux de succès S , où

S = fraction de tous les accès mémoire que l'on trouve dans la mémoire la plus rapide (par exemple, le cache).

T_1 = Temps d'accès au niveau 1

T_2 = Temps d'accès au niveau 2

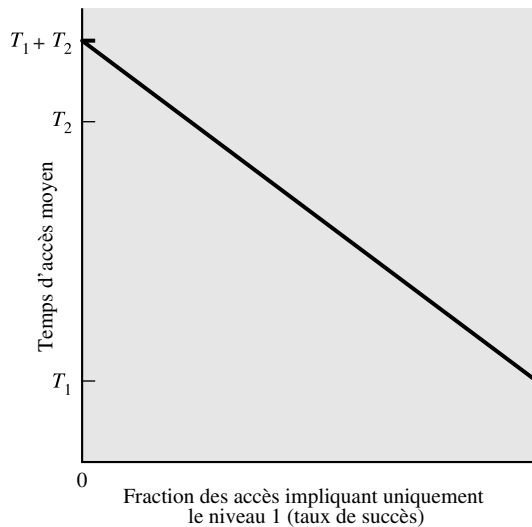


Figure 4.2 • Performance d'une mémoire à deux niveaux.

Comme vous pouvez le constater, pour les pourcentages d'accès au niveau 1 élevés, le temps d'accès total moyen est beaucoup plus proche de celui du niveau 1 que de celui du niveau 2.

Dans notre exemple, supposons que 95 % des accès mémoire se trouvent dans le cache. Le temps d'accès moyen à un mot peut alors s'exprimer comme suit

$$(0,95) (0,01 \mu\text{s}) + (0,05) (0,01 \mu\text{s} + 0,1 \mu\text{s}) = 0,0095 + 0,0055 = 0,015 \mu\text{s}$$

Dans cet exemple, le temps d'accès moyen est bien plus proche de $0,01 \mu\text{s}$ que de $0,1 \mu\text{s}$, comme attendu. L'utilisation d'une mémoire à deux niveaux pour réduire le temps d'accès fonctionne sur le principe, mais uniquement si les conditions (a) à (d) s'appliquent. En employant plusieurs technologies, on trouve une gamme de systèmes

mémoire qui répondent aux conditions (a) à (c). Heureusement, c'est aussi souvent le cas de la condition (d).

Le principe à la base de la condition (d) s'appelle **localité des références**. Pendant l'exécution d'un programme, les références faites à la mémoire par le processeur, tant pour les instructions que pour les données, tendent à se constituer en groupes. Les programmes contiennent un certain nombre de boucles et de sous-routines itératives. Une fois entré dans une boucle ou une sous-routine, le processeur référence de manière répétitive un petit ensemble d'instructions. De la même manière, les opérations sur des tables et des tableaux impliquent l'accès à un petit groupe de mots de données. Sur une longue période, les groupes changent, mais sur une période courte, le processeur travaille essentiellement avec les mêmes groupes de références mémoire. Il est ainsi possible d'organiser les données dans la hiérarchie de sorte que plus on descend dans les niveaux, plus le pourcentage d'accès diminue. Prenons l'exemple de la mémoire à deux niveaux déjà présenté. Disons que le niveau 2 de la mémoire contient toutes les informations de programmes et les données. Les groupes en cours sont placés temporairement au niveau 1. De temps en temps, l'un des groupes du niveau 1 doit basculer à nouveau au niveau 2 pour laisser la place à un nouveau groupe provenant du niveau 1. Mais en moyenne, la plupart des références concerneront des instructions et des données qui se trouvent au niveau 1.

On peut appliquer ce principe à plus de deux niveaux de mémoire, comme le suggère la hiérarchie de la figure 4.1. Le type de mémoire le plus rapide, le plus petit et le plus cher se compose des registres internes au processeur. Ce dernier contient généralement quelques douzaines de ces registres, bien que certaines machines en aient plusieurs centaines. Si l'on descend de deux niveaux, on se retrouve dans la mémoire principale qui représente le système de mémoire interne principal de l'ordinateur. Chaque emplacement de la mémoire principale possède une adresse unique. Pour étendre la mémoire principale, on utilise un cache haute vitesse plus petit. Il est généralement invisible pour le programmeur, comme pour le processeur. Il sert à gérer les mouvements des données entre la mémoire principale et les registres du processeur pour améliorer les performances.

Les trois formes de mémoires que nous venons de décrire sont volatiles et emploient la technologie des semi-conducteurs. L'utilisation de ces trois niveaux exploite le fait qu'il existe une grande variété de mémoires à semi-conducteurs, dont la vitesse et le prix diffèrent. Les données sont stockées de manière plus permanente sur des composants de stockage de masse externes, dont les plus courants sont les disques durs et les supports amovibles, comme les disques, les bandes et les disques optiques. La mémoire externe non volatile prend également le nom de mémoire secondaire ou auxiliaire. Elle sert à stocker des fichiers de programmes ou de données généralement visibles pour le programmeur sous la forme de fichiers ou d'enregistrements, par opposition aux octets ou aux mots. Le disque sert également à fournir une extension de la mémoire principale, appelée mémoire virtuelle, dont nous parlerons au chapitre 8.

On peut trouver d'autres formes de mémoire dans la hiérarchie. Par exemple, de gros ordinateurs IBM sont équipés d'une forme de mémoire interne appelée stockage étendu. Elle utilise une technologie de semi-conducteurs plus lente et moins coûteuse

que la mémoire principale. À proprement parler, cette mémoire n'entre pas dans la hiérarchie mais dans une branche latérale : on peut déplacer les données entre la mémoire principale et le stockage étendu, mais pas entre le stockage étendu et la mémoire externe. Parmi les autres formes de mémoire auxiliaire, on trouve les disques optiques et magnéto-optiques. Pour finir, d'autres niveaux peuvent être ajoutés à la hiérarchie par logiciel. On peut utiliser une partie de la mémoire principale comme tampon pour conserver temporairement les données à lire sur le disque. Une telle technique, que l'on nomme parfois cache de disque¹, améliore les performances de deux manières :

- Les écritures sur le disque sont regroupées. À la place d'un grand nombre de petits transferts de données, on instaure un petit nombre de transferts importants, d'où une amélioration de la performance du disque et une faible implication du processeur.
- Certaines données destinées à être écrites peuvent être référencées par un programme avant le prochain transfert sur le disque. Dans ce cas, les données peuvent être obtenues rapidement depuis le cache logiciel au lieu d'un accès disque lent.

L'annexe 4A examine les implications sur les performances des structures mémoire multiniveaux.

4.2 Principes de la mémoire cache

La mémoire cache est destinée à fournir une vitesse approchant celle des mémoires les plus rapides tout en offrant une taille importante à un prix inférieur de celui des mémoires à semi-conducteurs. La figure 4.3 illustre ce concept. On a associé une mémoire principale lente de taille importante à une mémoire cache plus petite et plus rapide, qui contient une copie de certaines parties de la mémoire principale. Lorsque le processeur tente de lire un mot en mémoire, on vérifie s'il se trouve dans le cache.

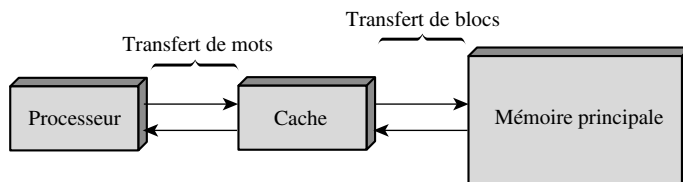


Figure 4.3 • Mémoire cache et mémoire principale.

Si tel est le cas, le mot est livré au processeur. Dans le cas contraire, on lit un bloc de la mémoire principale, constitué d'un nombre fixe de mots, que l'on place dans le cache avant de livrer le mot au processeur. En raison de la localité des références, il est

1. Le cache disque est généralement une technique purement logicielle que nous n'étudierons pas dans ce livre.

fort probable que lorsqu'un bloc de données est placé dans le cache pour répondre à une référence mémoire, d'autres références à venir concerneront le même emplacement mémoire ou d'autres mots du même bloc.

La figure 4.4 illustre la structure du système cache/mémoire principale. Cette dernière contient jusqu'à 2^n mots adressables, chacun possédant une adresse unique de n bits. Pour la correspondance, cette mémoire se compose de blocs d'une longueur fixe de K mots chacun. Autrement dit, il y a $M = 2^n/K$ blocs. Le cache se divise en C lignes de K mots, le nombre de lignes étant nettement inférieur au nombre de blocs de la mémoire principale ($C \ll M$). Il se trouve toujours un sous-ensemble de blocs mémoire dans les lignes du cache. Si on lit un mot d'un bloc de mémoire, on transfère ce bloc dans l'une des lignes du cache. Dans la mesure où il y a plus de blocs que de lignes, celles-ci ne peuvent être dédiées de manière permanente à un bloc particulier. En conséquence, chaque ligne contient une **étiquette**, composée généralement d'une partie de l'adresse de la mémoire principale, qui identifie le bloc stocké. L'étiquette est décrite plus loin.

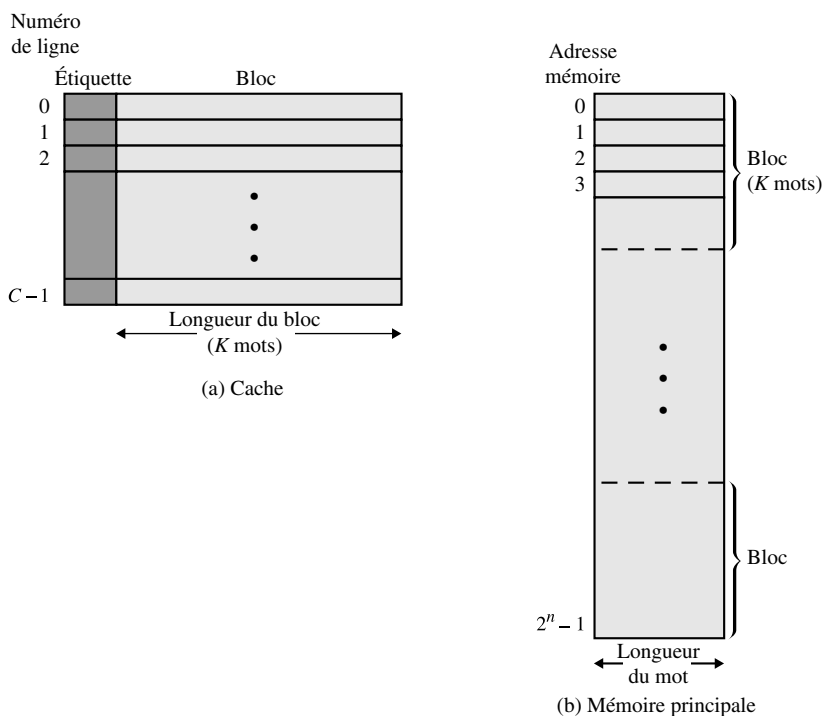


Figure 4.4 • Structure cache/mémoire principale.

La figure 4.5 illustre l'opération de lecture. Le processeur génère l'adresse, AL, d'un mot à lire. Si celui-ci se trouve dans le cache, il est fourni au processeur. Dans le cas contraire, on charge le bloc contenant ce mot dans le cache et on le transmet au processeur.

La figure 4.5 montre les deux dernières opérations qui se produisent parallèlement et reflète l'organisation classique des caches contemporains, illustrée par la figure 4.6. Dans cette organisation, le cache est connecté au processeur *via* les lignes de données, de contrôle et d'adresses. Les lignes de données et d'adresses sont également liées aux tampons de données et d'adresses, qui sont reliés à un bus système à partir duquel on peut atteindre la mémoire principale. En cas de succès (*hit*), on désactive les tampons de données et d'adresses : seule la communication entre le processeur et le cache est active, sans autre forme de trafic sur le bus système. En cas d'échec (*miss*), on charge l'adresse sur le bus système et on retourne les données via le tampon de données au cache et au processeur. Dans d'autres organisations, on interpose physiquement le cache entre le processeur et la mémoire principale pour toutes les lignes de données, d'adresses et de contrôle. Dans ce cas, sur un échec cache, on lit d'abord le mot dans le cache puis on le transfère du cache au processeur.

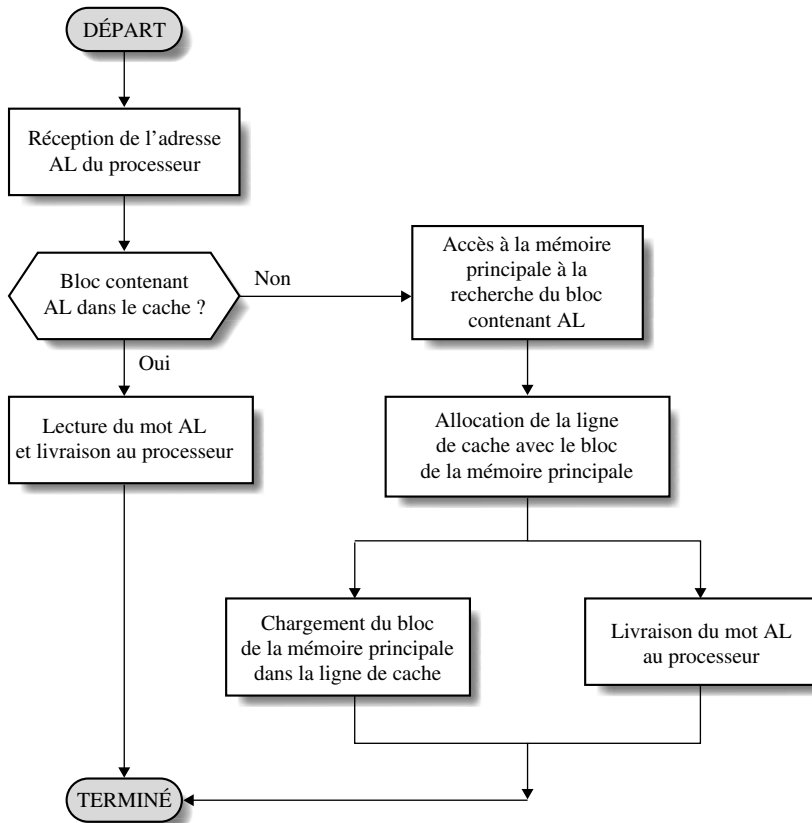


Figure 4.5 • Opération de lecture du cache.

L'annexe 4A contient une discussion sur les paramètres de performance relatifs à l'utilisation du cache.

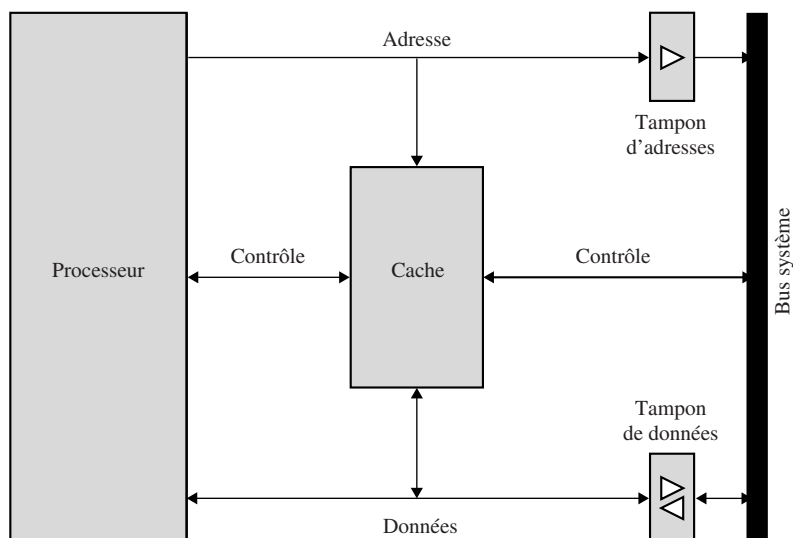


Figure 4.6 • Organisation classique du cache.

4.3 Éléments de la conception du cache

Cette section donne un aperçu des paramètres de conception du cache et fournit quelques résultats classiques. Nous nous référons occasionnellement à l'utilisation des caches dans le calcul haute performance (CHP). Le CHP concerne les superordinateurs et leurs logiciels, en particulier dans les applications scientifiques qui impliquent des grandes quantités de données, des calculs vectoriels et matriciels, et l'utilisation d'algorithmes parallèles. La conception du cache des CHP est différente de celle des autres plates-formes et applications matérielles. En fait, les chercheurs ont découvert que les performances des applications CHP sur les architectures informatiques qui emploient les caches sont médiocres. D'autres chercheurs ont démontré qu'une hiérarchie de cache peut améliorer les performances si le logiciel d'application est optimisé pour utiliser le cache.

Bien qu'il existe un grand nombre d'implémentations de cache, les éléments de conception fondamentaux sont peu nombreux. Ils servent à classer et à différencier les architectures de cache et sont récapitulés dans le tableau 4.2.

Taille du cache

Nous avons déjà parlé du premier élément : la taille du cache doit être suffisamment réduite pour que le coût global moyen par bit soit aussi proche que possible de celui de la mémoire principale seule et suffisamment importante pour que le temps d'accès global soit proche de celui du cache seul. Plusieurs autres motifs poussent à minimiser la taille du cache. Plus il est grand, plus il faut de portes pour son adressage.

Tableau 4.2 • Éléments de la conception du cache.

Taille du cache	Stratégies d'écriture
Fonction de correspondance	Écriture immédiate (<i>Write through</i>)
Directe	Écriture différée (<i>Write back</i>)
Associative	Écriture unique (<i>Write once</i>)
Associative par ensemble	
Algorithme de remplacement	Taille des lignes
LRU (<i>Least recently used</i> , moins récemment utilisé)	Nombre de caches
FIFO (<i>First in first out</i> , premier entré premier sorti)	Un ou deux niveaux
LFU (<i>Least frequently used</i> , moins fréquemment utilisé)	Unifié ou séparé
Aléatoire	

En conséquence, les caches de grande taille tendent à être légèrement plus lents, même lorsque la technologie du circuit intégré et l'emplacement sur la puce et la carte sont identiques. La surface disponible pour ces deux composants limite également la taille du cache. Les performances de ce dernier étant particulièrement sensibles à la nature de la charge de travail, il est impossible de parvenir à une taille de cache optimale unique. Le tableau 4.3 liste les tailles de cache de processeurs actuels et anciens.

Tableau 4.3 • Tailles de cache de certains processeurs.

Processeur	Type	Année de mise en service	Cache L1	Cache L2	Cache L3
IBM 360/85	Gros ordinateur	1968	16 à 32 Ko	—	—
PDP-11/70	Mini-ordinateur	1975	1 Ko	—	—
VAX 11/780	Mini-ordinateur	1978	16 Ko	—	—
IBM 3033	Gros ordinateur	1978	64 Ko	—	—
IBM 3090	Gros ordinateur	1985	128 à 256 Ko	—	—
Intel 80486	PC	1989	8 Ko	—	—
Pentium	PC	1993	8 Ko/8 Ko	256 à 512 Ko	—
PowerPC 601	PC	1993	32 Ko	—	—
PowerPC 620	PC	1996	32 Ko/32 Ko	—	—
PowerPC G4	PC/serveur	1999	32 Ko/32 Ko	256 Ko à 1 Mo	2 Mo
IBM S/390 G4	Gros ordinateur	1997	32 Ko	256 Ko	2 Mo
IBM S/390 G6	Gros ordinateur	1999	256 Ko	8 Mo	—
Pentium 4	PC/serveur	2000	8 Ko/8 Ko	256 Ko	—
IBM SP	Serveur haut de gamme/ superordinateur	2000	64 Ko/32 Ko	8 Mo	—

Tableau 4.3 • Tailles de cache de certains processeurs (*suite*).

Processeur	Type	Année de mise en service	Cache L1	Cache L2	Cache L3
CRAY MTA	PC/serveur	2001	16 Ko/16 Ko	96 Ko	4 Mo
Itanium	PC/serveur	2001	16 Ko/16 Ko	96 Ko	4 Mo
SGI Origin 2001	Serveur haut de gamme	2001	32 Ko/32 Ko	4 Mo	—

Deux valeurs séparées par une barre oblique font référence aux caches d'instructions et de données. Les deux caches sont uniquement des caches d'instructions et non de données.

Fonction de correspondance

Dans la mesure où les lignes de cache sont moins nombreuses que les blocs de mémoire principale, on a besoin d'un algorithme pour la correspondance entre les blocs et les lignes de cache. Il faut, en outre, trouver un moyen de déterminer quel bloc de la mémoire principale occupe la ligne de cache. Le choix de la fonction de correspondance détermine l'organisation du cache. On peut employer trois techniques de correspondance : directe, associative et associative par ensemble. Pour chaque cas, nous allons étudier la structure générale puis un exemple particulier qui tiendra compte des éléments suivants :

- Le cache peut contenir 64 Ko.
- Les données sont transférées entre la mémoire principale et le cache sous forme de blocs de 4 octets. Autrement dit, le cache est organisé en $16\text{ K} = 2^{14}$ lignes de 4 octets.
- La mémoire principale fait 16 Mo, chaque octet étant directement adressable par une adresse 24 bits ($2^{24} = 16\text{ M}$). Ainsi, pour la correspondance, nous pouvons considérer que la mémoire principale se compose de 4 M de blocs de 4 octets chacun.

La technique la plus simple, appelée **correspondance directe**, associe chaque bloc de la mémoire principale à une seule ligne de cache possible. La figure 4.7 en illustre le mécanisme général. On exprime la correspondance sous la forme

$$i = j \text{ modulo } m$$

où

i = numéro de la ligne de cache

j = numéro du bloc de la mémoire principale

m = nombre de lignes dans le cache

On utilise l'adresse pour implémenter la fonction de correspondance. En matière d'accès au cache, on peut diviser chaque adresse de la mémoire principale en trois champs. Les bits w de poids faible identifient un mot ou un octet uniques dans un bloc de mémoire principale. Sur les ordinateurs récents, l'adresse se trouve au niveau de l'octet.

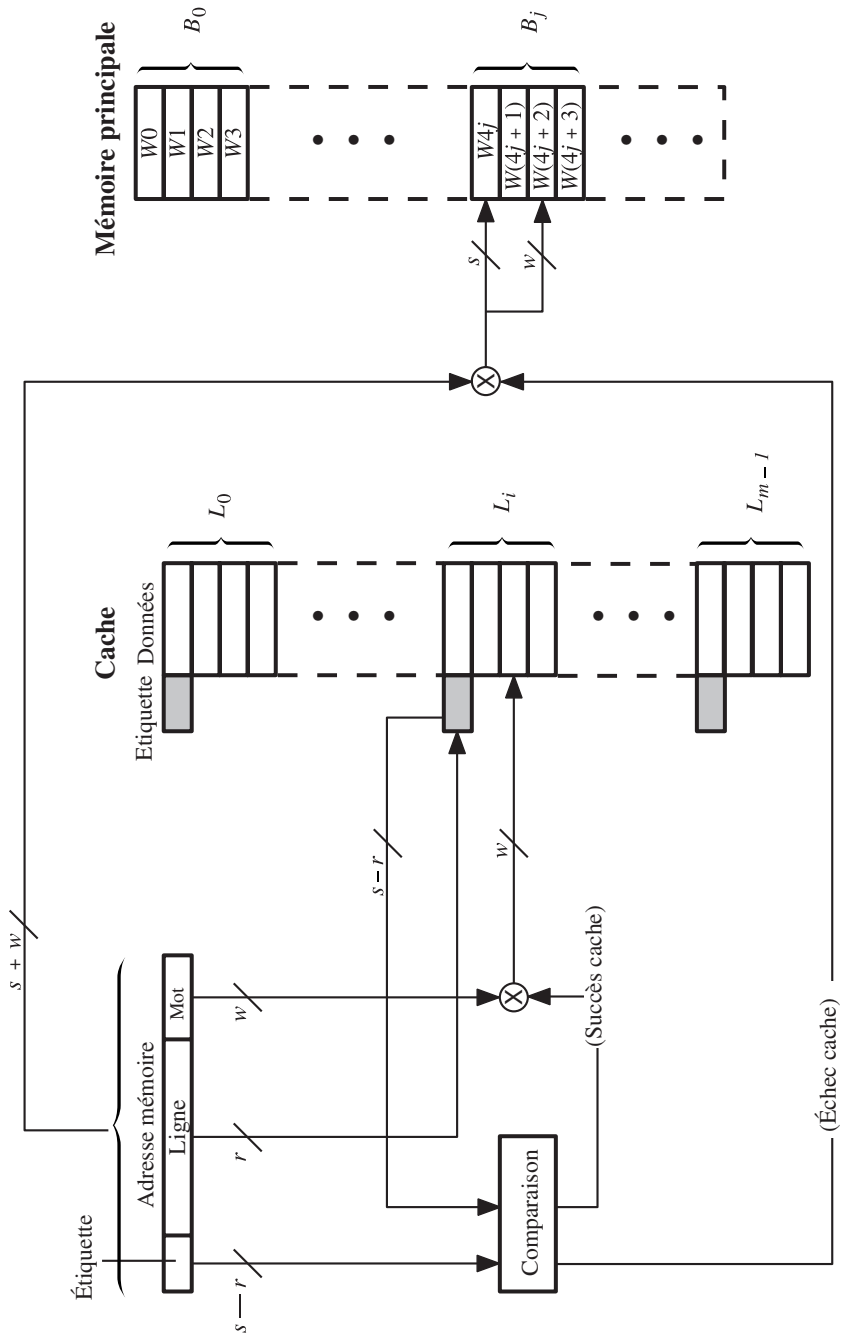


Figure 4.7 • Organisation d'un cache avec correspondance directe.

Les bits restants désignent l'un des 2^s blocs de la mémoire principale. La logique du cache interprète ces s bits comme une étiquette de $s - r$ bits (la partie la plus significative) et un champ de r bits. Ce dernier identifie l'une des $m = 2^r$ lignes de cache. En résumé :

- longueur de l'adresse = $(s + w)$ bits
- nombre d'unités adressables = 2^{s+w} mots ou octets
- taille du bloc = taille de la ligne = 2^w mots ou octets
- nombre de blocs dans la mémoire principale = $\frac{2^{s+w}}{2^w} = 2^s$
- nombre de lignes dans le cache = $m = 2^r$
- taille de l'étiquette = $(s - r)$ bits

Le résultat de cette correspondance est que les blocs de la mémoire principale sont attribués à des lignes du cache de la manière suivante :

Ligne du cache	Blocs de la mémoire principale attribués
0	$0, m, 2m, \dots, 2^s - m$
1	$1, m + 1, 2m + 1, \dots, 2^s - m + 1$
...	...
$m - 1$	$m - 1, 2m - 1, 3m - 1, \dots, 2^s - 1$

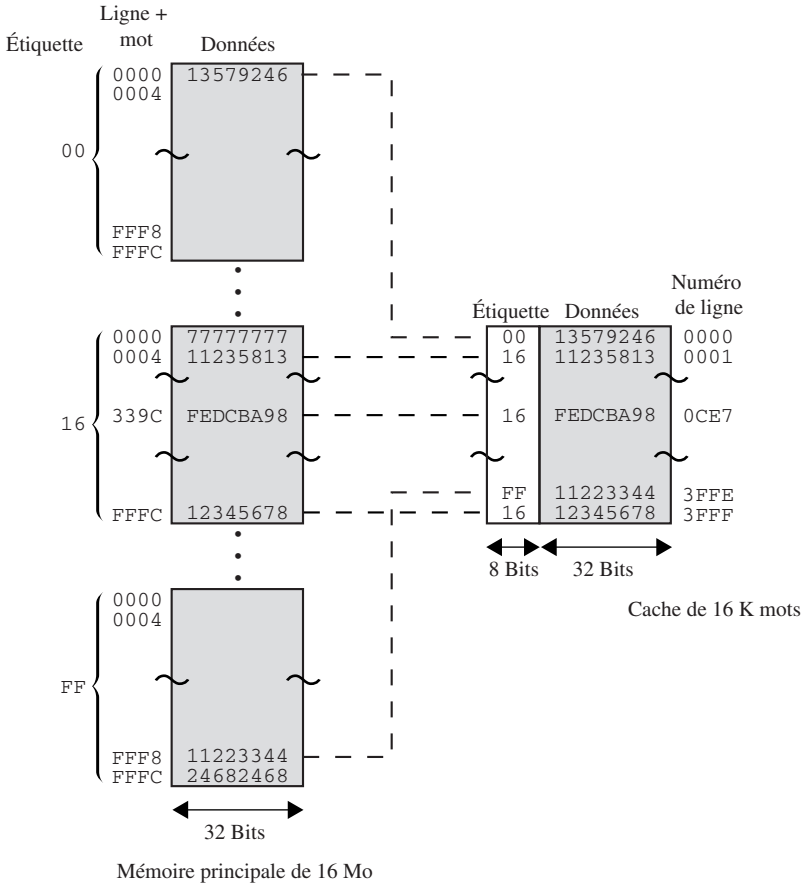
Ainsi, en utilisant une partie de l'adresse comme numéro de ligne, on obtient une correspondance unique de chaque bloc de mémoire principale dans le cache. Lorsqu'on lit ce bloc dans la ligne qui lui est attribuée, on doit marquer les données pour le distinguer des autres blocs qui peuvent être dans cette ligne. Pour ce faire, on fait appel aux $s - r$ bits les plus significatifs.

La figure 4.8 montre notre exemple de système exploitant la correspondance directe¹, dans lequel $m = 16 \text{ K} = 2^{14}$ et $i = j$ modulo 2^{14} . Voici le résultat de cette correspondance :

Ligne de cache	Adresse mémoire de départ des blocs
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
...	...
$2^{14} - 1$	00FFFC, 01FFFC, ..., FFFFFC

1. Dans les figures suivantes, les valeurs d'adresse et mémoire sont représentées en notation hexadécimale. Reportez-vous à l'annexe B pour un rappel des systèmes de numérotation (décimal, binaire, hexadécimal).

Notez que les blocs qui vont dans le même numéro de ligne possèdent la même étiquette. Ainsi, les blocs dont les adresses de départ sont 000000, 010000, ..., FF0000 possèdent respectivement les étiquettes 00, 01, ..., FF.



	Étiquette	Ligne	Mot
Adresse de la mémoire principale =	8	14	2

Figure 4.8 • Exemple de correspondance directe.

Si nous nous référons à nouveau à la figure 4.5, voici comment fonctionne une opération de lecture. On présente une adresse (24 bits) au système cache. Le numéro de ligne (14 bits) sert d'index dans le cache pour accéder à une ligne donnée. Si l'étiquette (8 bits) correspond à l'étiquette stockée dans cette ligne, on utilise le numéro du mot (2 bits) pour sélectionner l'un des quatre octets de cette ligne. Dans le cas contraire, le champ (22 bits) « étiquette + ligne » sert à lire un bloc de la mémoire

principale. L'adresse réelle qui sert à la lecture se compose du champ (22 bits) étiquette + ligne concaténé à deux bits 0 pour lire 4 octets en démarrant sur une frontière de bloc.

La correspondance directe est simple et peu coûteuse à implémenter. Son principal inconvénient est l'emplacement fixe dans le cache de chaque bloc. En conséquence, si un programme référence de manière répétitive des mots de deux blocs différents qui correspondent à la même ligne, ces blocs permutent continuellement dans le cache, et le taux de succès est faible (ce phénomène est appelé *effondrement*).

La **correspondance associative** contourne l'inconvénient de la correspondance directe en permettant de charger les blocs de la mémoire principale dans n'importe quelle ligne du cache. Dans ce cas, la logique de contrôle du cache considère l'adresse mémoire comme un champ étiquette et mot. Le champ étiquette identifie uniquement un bloc de la mémoire principale. Pour déterminer si un bloc se trouve dans le cache, la logique de contrôle du cache doit simultanément examiner chaque étiquette de ligne à la recherche d'une correspondance, comme l'illustre la figure 4.9. Notez qu'aucun champ de l'adresse ne correspond au numéro de la ligne, de manière que le nombre de lignes du cache ne soit pas déterminé par le format de l'adresse. En résumé :

- longueur de l'adresse = $(s + w)$ bits
- nombre d'unités adressables = 2^{s+w} mots ou octets
- taille du bloc = taille de la ligne = 2^w mots ou octets
- nombre de blocs dans la mémoire principale = $\frac{2^{s+w}}{2^w} = 2^s$
- nombre de lignes dans le cache = indéterminé
- taille de l'étiquette = s bits

La figure 4.10 illustre notre exemple utilisant la correspondance associative. L'adresse de la mémoire principale se compose d'une étiquette (22 bits) et d'un numéro d'octet (2 bits). L'étiquette (22 bits) doit être stockée avec les 32 bits de données pour chaque ligne du cache. Notez que ce sont les 22 bits les plus significatifs de l'adresse qui forment l'étiquette¹. Ainsi, l'adresse hexadécimale sur 24 bits 16339C contient l'étiquette (22 bits) 058CE7, ce que montre la notation binaire :

Adresse mémoire	0001	0110	0011	0011	1001	1100	(binaire)
	1	6	3	3	9	C	(hexadécimal)
Étiquette (22 bits les plus significatifs)	00	0101	1000	1100	1110	0111	(binaire)
	0	5	8	C	E	7	(hexadécimal)

1. Dans la figure 4.10, le marqueur 22 bits est représenté par un nombre hexadécimal à 6 chiffres. La longueur du chiffre hexadécimal le plus significatif n'est en fait que de 2 bits.

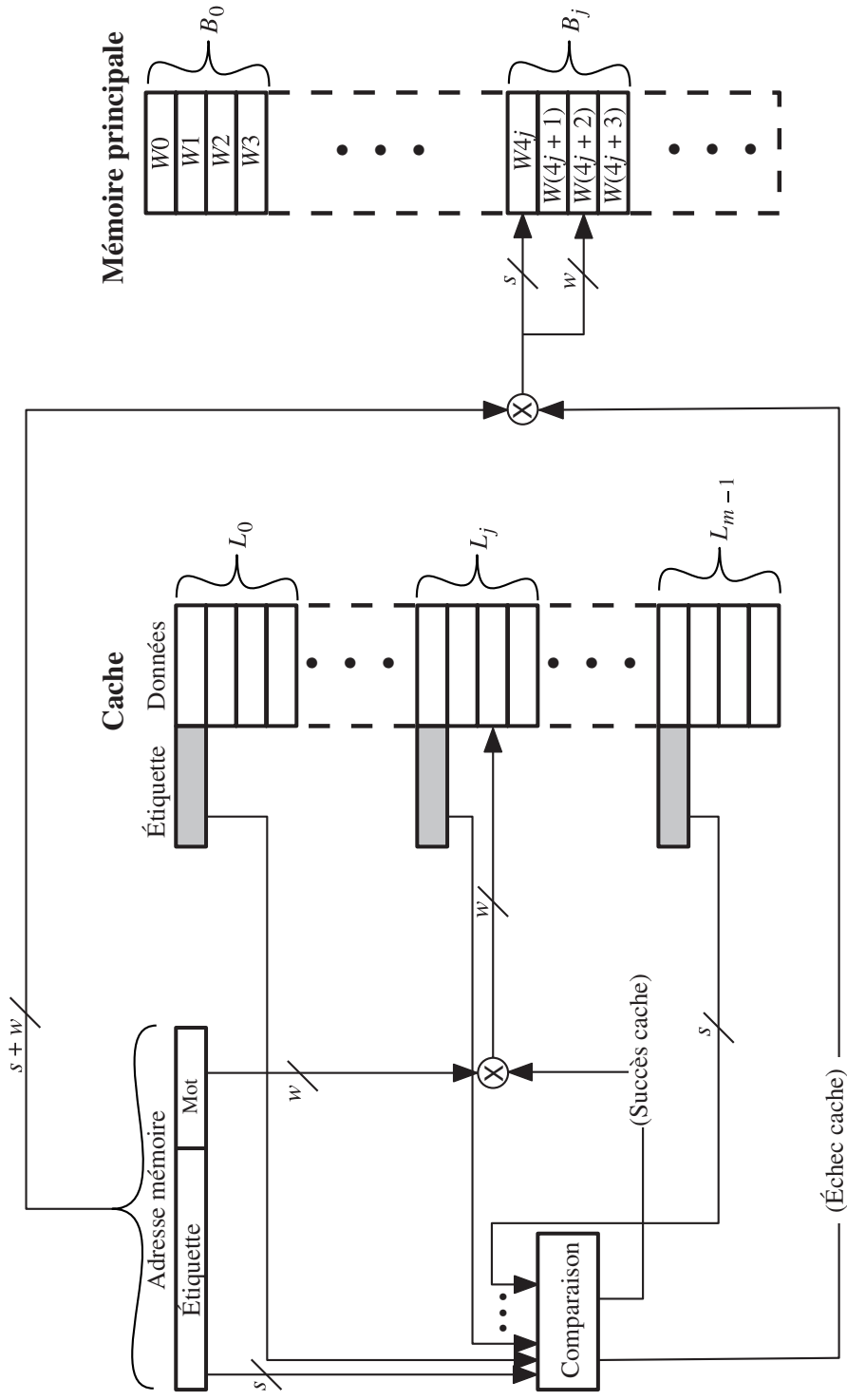


Figure 4.9 • Organisation d'un cache totalement associatif.

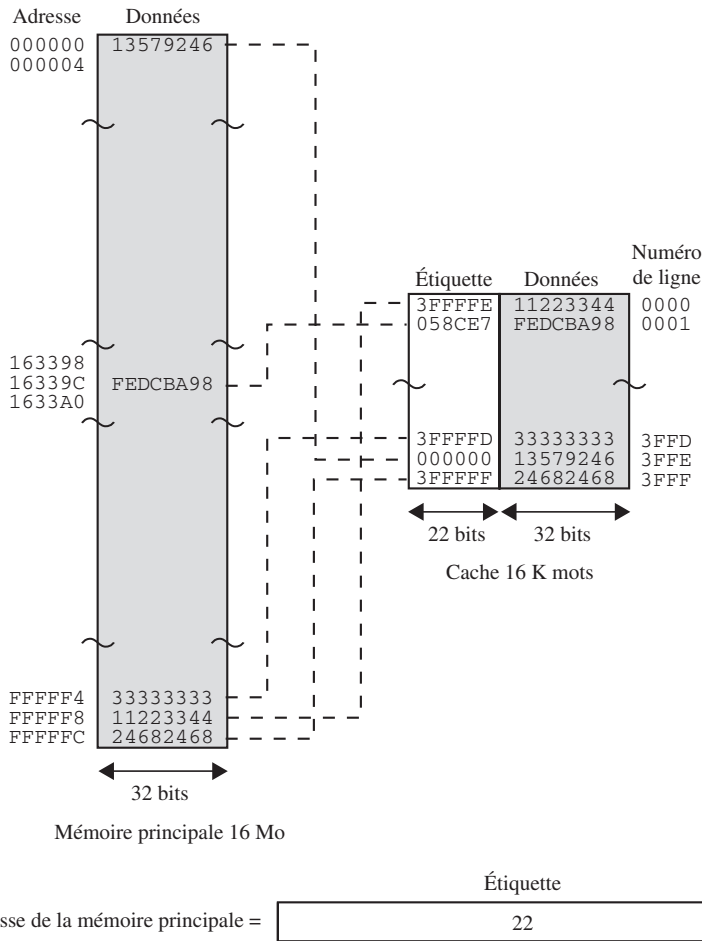


Figure 4.10 • Exemple de correspondance associative.

La correspondance associative est souple quant au bloc à remplacer lorsqu'on en lit un nouveau dans le cache. Les algorithmes de remplacement, dont nous parlerons plus loin dans cette section, sont destinés à augmenter le taux de succès. La complexité des circuits nécessaires pour examiner les étiquettes de toutes les lignes de cache en parallèle représente le principal inconvénient de ce type de correspondance.

La **correspondance associative par ensemble** est un compromis qui profite des avantages des approches directe et associative en en réduisant les inconvénients. Dans ce cas, le cache se divise en v ensembles, composés chacun de k lignes. Les relations sont les suivantes :

$$m = v \times k$$

$$i = j \text{ modulo } v$$

où

- i = numéro d'ensemble du cache
- j = numéro de bloc de la mémoire principale
- m = nombre de lignes dans le cache

C'est ce que l'on appelle la correspondance associative k voies. Avec la correspondance associative par ensemble, le bloc B_j peut être placé dans n'importe quelle ligne de l'ensemble i . Dans ce cas, la logique du contrôle interprète une adresse mémoire en trois champs : étiquette, ensemble et mot. Les bits de l'ensemble d désignent l'un des $v = 2^d$ ensembles. Les bits s de l'étiquette et de l'ensemble représentent l'un des 2^s blocs de la mémoire principale. La figure 4.11 illustre la logique du contrôle de cache.

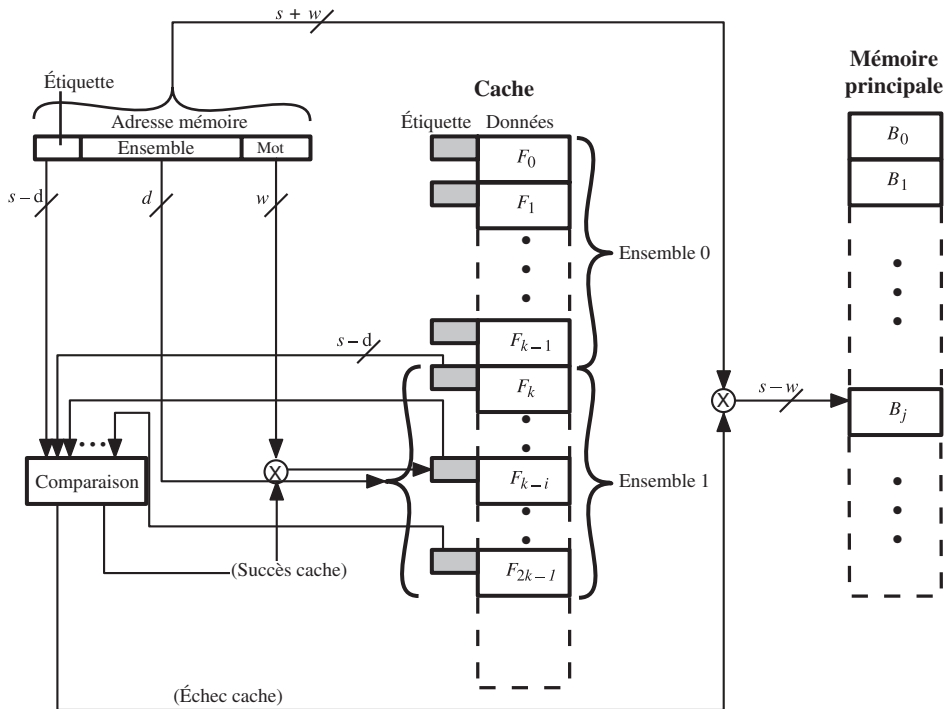


Figure 4.11 • Organisation du cache associatif k voies.

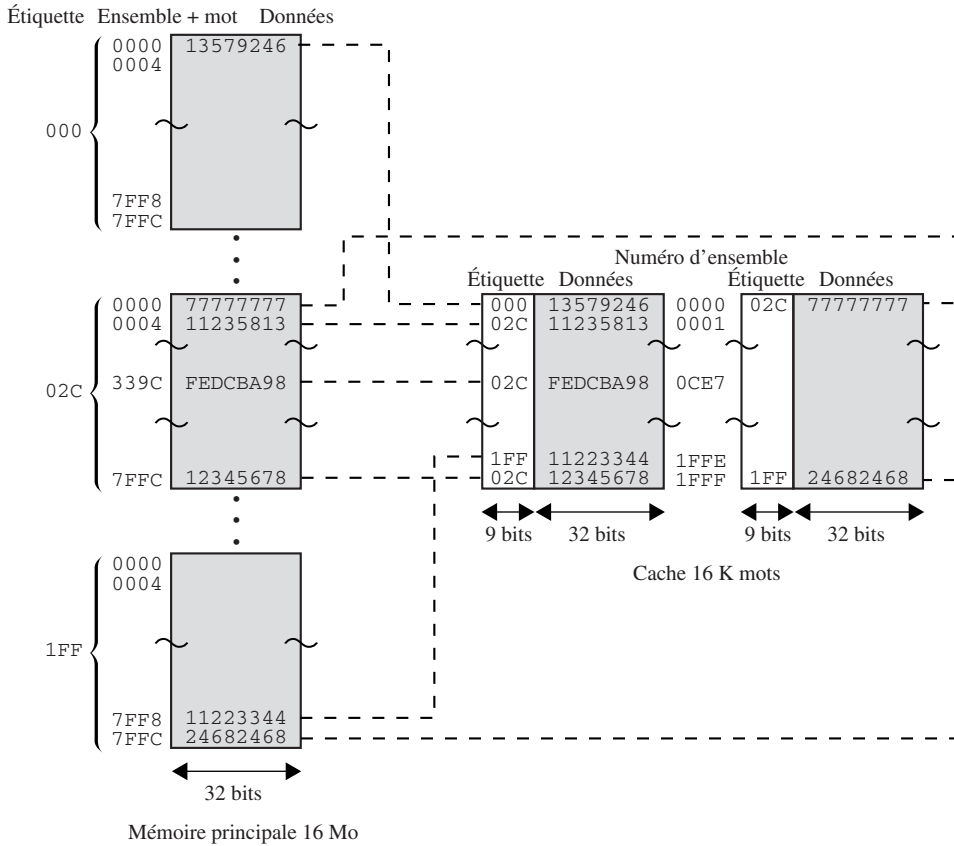
Avec une correspondance totalement associative, l'étiquette qui se trouve dans une adresse mémoire est grande et doit être comparée à l'étiquette de chaque ligne du cache. Avec la correspondance associative k voies, l'étiquette qui se trouve dans une adresse mémoire est beaucoup plus petite et doit uniquement être comparée aux étiquettes k d'un ensemble. En résumé :

- longueur de l'adresse = $(s + w)$ bits
- nombre d'unités adressables = 2^{s+w} mots ou octets
- taille du bloc = taille de la ligne = 2^w mots ou octets
- nombre de blocs dans la mémoire principale = $\frac{2^{s+w}}{2^w} = 2^s$
- nombre de lignes dans l'ensemble = k
- nombre d'ensembles $v = 2^d$
- nombre de lignes dans le cache = $kv = k \times 2^d$
- taille de l'étiquette = $(s + d)$ bits

La figure 4.12 montre un exemple d'utilisation de la correspondance associative par ensemble avec deux lignes par ensemble, que l'on appelle associative 2 voies¹. Le numéro d'ensemble (13 bits) identifie un ensemble unique de deux lignes dans le cache. Il indique également le nombre de blocs qui se trouvent dans la mémoire principale, modulo 2^{13} , ce qui détermine la correspondance des blocs en lignes. Ainsi les blocs 000000, 008000, ..., FF8000 de la mémoire principale vont dans l'ensemble 0 du cache. On peut charger n'importe lequel de ces blocs dans l'une ou l'autre des deux lignes de l'ensemble. Notez que deux blocs qui vont dans le même ensemble de cache possèdent le même numéro d'étiquette. Pour une opération de lecture, le numéro d'ensemble (13 bits) sert à déterminer l'ensemble de deux lignes à examiner à la recherche d'une correspondance avec le numéro d'étiquette de l'adresse à laquelle on doit accéder.

Dans le cas extrême où $v = m$, $k = 1$, la technique associative par ensemble se réduit à la correspondance directe et pour $v = 1$, $k = m$, elle se réduit à la correspondance associative. L'utilisation de deux lignes par ensemble ($v = m/2$, $k = 2$) est l'organisation associative par ensemble la plus courante. Elle améliore significativement le taux de succès (*hit ratio*) par rapport à la correspondance directe. La correspondance associative 4 voies ($v = m/4$, $k = 4$) apporte de modestes améliorations pour un coût relativement faible. Augmenter davantage le nombre de lignes n'a que peu d'effet.

1. Dans la figure 4.12, le marqueur 9 bits est représenté par un nombre hexadécimal à trois chiffres. La longueur du chiffre hexadécimal le plus significatif est en fait de 1 bit.



Adresse de la mémoire principale =	Étiquette	Ensemble	Mot
	9	13	2

Figure 4.12 • Exemple de correspondance associative 2 voies.

Algorithmes de remplacement

Lorsqu'on amène un nouveau bloc dans le cache, il faut remplacer l'un des blocs existants. Pour la correspondance directe, on ne peut disposer que d'une ligne par bloc, sans qu'aucun autre choix soit possible. Avec les techniques associative et associative par ensemble, on a besoin d'un algorithme de remplacement. Pour obtenir une vitesse élevée, un tel algorithme doit être implémenté par voie matérielle. Nous mentionnerons quatre des algorithmes les plus courants, dont le plus efficace est le LRU (*Least recently used*, moins récemment utilisé) qui remplace le bloc de l'ensemble qui se trouve dans le cache depuis le plus longtemps sans avoir été référencé. Cet algorithme est simple à implémenter avec la correspondance associative 2 voies. Chaque ligne contient un bit USE. Lorsque l'on référence une ligne, son bit USE est positionné à 1

et le bit USE de l'autre ligne est positionné à 0. Quand on lit le bloc dans l'ensemble, on utilise la ligne dont le bit USE est positionné à 0. Dans la mesure où l'on suppose que les emplacements mémoire les plus récemment utilisés sont plus susceptibles d'être référencés, le LRU doit obtenir le meilleur taux de succès. L'algorithme FIFO (*First in first out*, premier entré premier sorti) remplace le bloc de l'ensemble qui se trouve dans le cache depuis le plus longtemps. On l'implémente avec la technique de priorité tournante ou celle du tampon circulaire. L'algorithme LFU (*Least frequently used*, moins fréquemment utilisé) est une autre solution. Il remplace le bloc de l'ensemble qui a été le moins référencé. On l'implémente en associant un compteur à chaque ligne. Il existe également une technique qui n'est pas basée sur l'utilisation et qui consiste à choisir une ligne de façon aléatoire. Des simulations ont démontré que le remplacement aléatoire donne des résultats très légèrement inférieurs aux algorithmes basés sur l'utilisation.

Stratégie d'écriture

Avant de pouvoir remplacer un bloc situé dans le cache, on doit vérifier s'il a été modifié à cet endroit et non dans la mémoire principale. Si tel n'est pas le cas, on peut alors écraser le bloc qui réside dans le cache. Dans le cas contraire, cela signifie qu'au moins une opération d'écriture a eu lieu sur un mot de cette ligne du cache et que la mémoire principale doit être mise à jour en conséquence. Il existe différentes stratégies d'écriture qui allient des compromis à la fois économiques et de performance. Deux problèmes se posent. Premièrement, plusieurs composants peuvent devoir accéder à la mémoire principale. Un module d'E/S, par exemple, peut devoir lire/écrire directement en mémoire. Si un mot a uniquement été modifié dans le cache, le mot mémoire correspondant n'est pas valide. Il en va de même pour un mot du cache si le composant d'E/S a modifié la mémoire principale. Le problème se complique lorsque plusieurs processeurs sont connectés au même bus et que chacun possède son propre cache local. Dans ce cas, si on modifie un mot dans l'un des caches, il est possible qu'il invalide un mot dans les autres caches.

La technique d'écriture la plus simple est l'*écriture simultanée* (*write through*). Avec cette technique, toutes les opérations d'écriture se font simultanément dans le cache et la mémoire principale, garantissant ainsi que celle-ci est toujours valide. Tout autre module processeur-cache peut observer le trafic vers la mémoire principale pour conserver la consistance avec son propre cache. Le principal inconvénient de cette technique est qu'elle génère un trafic mémoire important et qu'elle peut être la source de goulets d'étranglement. L'*écriture différée* ou *réécriture* (*write back*) est une technique alternative qui minimise les écritures en mémoire. Dans ce cas, on actualise uniquement le cache ; on définit alors un bit MODIFIÉ associé à la ligne. Ensuite, lorsqu'on remplace un bloc, on l'écrit dans la mémoire principale si et seulement si le bit MODIFIÉ est positionné. Le problème de cette technique est que de grandes portions de la mémoire principale sont invalides et que, par conséquent, les modules d'E/S peuvent uniquement y accéder *via* le cache, d'où des circuits complexes et des goulets d'étranglement potentiels. L'expérience a démontré que le pourcentage des références mémoire qui sont des écritures mémoire est de l'ordre de 15 %.

Cependant, pour les applications CHP, ce chiffre peut approcher les 33 % (multiplication vecteur vecteur) voire atteindre les 50 % (transposition de matrice).

Dans une structure de bus où plusieurs composants (typiquement un processeur) possèdent un cache et où la mémoire principale est partagée, un nouveau problème se présente. Si on modifie les données de l'un des caches, on n'invalide pas uniquement le mot correspondant dans la mémoire principale, mais également ce mot dans les autres caches (si l'un d'eux contient le même mot). Même avec une stratégie d'écriture simultanée, les autres caches peuvent contenir des données invalides. Pour éviter ce problème, on fait appel à un système maintenant la cohérence des caches, dont il existe plusieurs approches :

- **Surveillance du bus avec écriture simultanée** : chaque contrôleur de cache surveille les lignes d'adressage pour détecter les opérations d'écriture en mémoire effectuées par d'autres maîtres du bus. Si un autre maître écrit dans un emplacement de la mémoire partagée qui se trouve également dans le cache, le contrôleur de cache invalide cette entrée. Cette stratégie dépend de l'utilisation de la stratégie d'écriture simultanée par tous les contrôleurs de cache.
- **Transparence matérielle** : on utilise du matériel supplémentaire pour garantir que toutes les actualisations de la mémoire principale via le cache sont reflétées dans tous les caches. Si un processeur modifie un mot dans son cache, on actualise la mémoire principale ainsi que tous les mots correspondants dans les autres caches.
- **Mémoire non cachable** : plusieurs processeurs se partagent une partie seulement de la mémoire principale que l'on nomme non cachable. Dans un tel système, tous les accès à la mémoire partagée sont des échecs cache puisqu'on ne copie jamais la mémoire partagée dans le cache. Pour identifier la mémoire non cachable, on peut faire appel à la logique de sélection des boîtiers ou aux bits d'adresse haute.

La cohérence du cache est un domaine de recherche actif que nous approfondirons au chapitre 18.

Taille de ligne

La taille de ligne est un autre élément de conception. Lorsque l'on accède et place un bloc de données dans le cache, on ne récupère pas uniquement le mot recherché, mais également quelques mots adjacents. À mesure que la taille du bloc augmente, le taux de succès commence par croître en raison du principe de localité qui veut que l'on référence rapidement les mots situés dans un proche voisinage, et que des données utiles se retrouvent dans le cache. Le taux de succès diminue lorsque la probabilité d'exploiter les informations récemment lues devient plus faible que celle d'utiliser les données qui ont dû être remplacées. Deux effets spécifiques entrent en jeu :

- Les blocs plus importants réduisent le nombre total de blocs dans le cache. Chaque lecture de bloc écrase l'ancien contenu du cache. Par conséquent, plus le nombre de blocs diminue, plus on écrase des données récemment lues.

- À mesure qu'un bloc grandit, chaque nouveau mot s'éloigne du mot demandé, ce qui réduit la probabilité qu'il soit exploité dans un avenir proche.

La relation entre la taille du bloc et le taux de succès est complexe. Elle dépend des caractéristiques de localité d'un programme donné et ne permet pas de définir une valeur optimale définitive. Il semble raisonnable de prévoir entre 8 et 32 octets. Sur les systèmes CHP, on emploie plus fréquemment des tailles de lignes de cache situées entre 64 et 128 octets.

Nombre de caches

Si à l'origine, on ne trouvait qu'un cache par système, plus récemment, l'utilisation de plusieurs caches est devenue la norme, ce qui engendre deux problèmes de conception : le nombre de niveaux de caches et l'emploi de caches unifiés ou séparés.

Caches multiniveaux

À mesure que la densité d'intégration augmente, il devient possible d'implanter le cache sur la même puce que le processeur, on obtient ainsi un cache interne. Comparé au cache auquel on accède *via* un bus externe, le cache interne réduit l'activité externe du processeur et accélère, par conséquent, le temps d'exécution et les performances globales du système. Lorsque l'instruction ou les données demandées se trouvent dans le cache interne, on élimine l'accès bus. Les chemins de données dans le processeur étant plus courts que les longueurs de bus, les accès au cache interne s'effectuent plus rapidement que ne le feraient des cycles de bus, même sans attente. En outre, pendant cette période, le bus est libre de prendre en charge d'autres transferts.

Le cache interne laisse en suspens la question suivante : « Faut-il ajouter un cache en dehors de la puce du processeur ou avoir un cache externe ? » La réponse est oui et la plupart des conceptions actuelles sont équipées des deux. L'organisation qui en découle se nomme cache à deux niveaux : le cache interne de niveau 1 (L1) et le cache externe de niveau 2 (L2). Pourquoi ajouter un cache L2 ? En son absence, si le processeur demande à accéder à un emplacement mémoire qui ne se trouve pas dans le cache L1, il doit passer par le bus pour atteindre la DRAM ou la ROM. La lenteur du bus et du temps d'accès mémoire entraînent des performances médiocres. Par contre, si on utilise une SRAM (RAM statique) L2, on pourra rapidement trouver les informations manquantes. Si la SRAM est suffisamment rapide pour coïncider avec la vitesse du bus, on accède aux données par une transaction sans attente, autrement dit par le transfert de bus le plus rapide.

La conception des caches multiniveaux possède deux fonctionnalités intéressantes. D'abord, pour un cache externe L2, on utilise rarement le bus système comme chemin de transfert entre le cache L2 et le processeur. On lui préfère un chemin de données distinct qui permet de réduire la surcharge du bus système. Ensuite, avec le rétrécissement de la taille des composants du processeur, on intègre de plus en plus souvent le cache L2 dans la puce du processeur, améliorant ainsi les performances.

Les économies liées à la présence d'un cache L2 dépendent des taux de succès dans les caches L1 et L2. Plusieurs études ont démontré qu'en règle générale, l'utilisation d'un cache de second niveau améliore vraiment les performances. Cependant, l'emploi de caches multiniveaux complique réellement la conception globale des caches en termes de taille, d'algorithme de remplacement et de stratégie d'écriture.

Caches unifiés ou séparés

Lorsque les premiers caches internes ont fait leur apparition, on trouvait souvent un cache unique pour ranger les références aux données et aux instructions. Plus récemment, il est devenu courant de séparer le cache en deux : l'un consacré aux instructions et l'autre aux données.

Le cache unifié présente deux avantages :

- Pour une taille donnée, le cache unifié possède un taux de succès supérieur à celui des caches séparés. Il équilibre en effet automatiquement la charge entre les lectures d'instructions et de données. Autrement dit, si une configuration d'exécution implique plus de lectures d'instructions que de lectures de données, le cache tend à se remplir d'instructions. Et si une configuration d'exécution implique relativement plus de lectures de données, c'est le contraire qui se produit.
- Il ne faut concevoir et implémenter qu'un seul cache.

En dépit de ces avantages, la tendance va dans le sens des caches séparés, en particulier sur les processeurs superscalaires comme le Pentium et le PowerPC qui favorisent l'exécution d'instructions en parallèle et le préchargement des instructions à venir. Le principal avantage du cache séparé est qu'il élimine les conflits entre l'unité de lecture/décodage d'instructions et l'unité d'exécution : point important pour toute conception basée sur l'exécution pipeline des instructions. Le processeur lit les instructions à l'avance et remplit un tampon, ou pipeline, avec les instructions à exécuter. Supposons maintenant qu'il existe un cache instructions/données unifié. Lorsque l'unité d'exécution accède à la mémoire pour charger et ranger des données, la requête est soumise au cache unifié. Si, au même moment, le préchargement d'instructions émet une requête de lecture vers le cache, celle-ci sera temporairement bloquée pour que le cache puisse d'abord servir l'unité d'exécution. Ce conflit pour le cache peut dégrader les performances, empêchant une utilisation efficace du pipeline d'instructions. Des caches séparés contournent cette difficulté.

4.4 Organisations des caches du Pentium 4 et du PowerPC

Organisation des caches du Pentium 4

L'évolution des microprocesseurs Intel reflète parfaitement celle de l'organisation des caches. Si le 80386 n'est pas équipé d'un cache interne, le 80486 comporte un cache interne de 8 Ko suivant l'organisation associative 4 voies, avec une taille de ligne de 16 octets. Tous les processeurs Pentium sont équipés de deux caches internes L1, un

pour les données et un pour les instructions. Sur le Pentium 4, le cache de données L1 fait 8 Ko, avec une taille de ligne de 64 octets et une organisation associative 4 voies. Nous décrirons plus loin le cache d'instructions du Pentium 4. Ce dernier inclut également un cache L2 qui alimente les deux caches L1. Le cache L2 est associatif 8 voies avec une taille de 256 Ko et des lignes de 128 octets. La figure 4.13 donne un aperçu simplifié de son organisation et met en évidence la disposition des trois caches. Le noyau du processeur se compose de quatre composants majeurs :

- **Unité lecture/décodage** : lit dans l'ordre les instructions du programme à partir du cache L2, les décode en une série de micro-opérations et range le résultat dans le cache d'instructions L1.
- **Logique d'exécution non ordonnée** : planifie l'exécution des micro-opérations sensibles aux dépendances de données et à la disponibilité des ressources. Les micro-opérations peuvent ainsi s'exécuter dans un ordre différent de celui dans lequel elles ont été lues dans le flot d'instructions. Si elle en a le temps, cette unité planifie l'exécution spéculative de micro-opérations à venir.
- **Unités d'exécution** : ces unités exécutent les micro-opérations. Elles lisent les données du cache de données L2 et rangent temporairement le résultat dans les registres.
- **Sous-système mémoire** : cette unité contient le cache L2 et le bus système qui sert à accéder à la mémoire principale lorsque les caches L1 et L2 ont un échec, et à accéder aux ressources d'E/S du système.

Contrairement à l'organisation employée dans tous les modèles Pentium précédents, et dans la majorité des autres processeurs, le cache d'instructions du Pentium 4 se trouve entre la logique de décodage des instructions et l'exécution. Comme nous le verrons plus en détail au chapitre 14, le processus du Pentium décode, ou convertit, les instructions machine Pentium en instructions de type RISC que l'on appelle micro-opérations. Grâce à ces micro-opérations simples de longueur fixe, on peut exploiter le fonctionnement pipeline superscalaire et les techniques d'ordonnement qui améliorent les performances. Les instructions machine du Pentium sont néanmoins complexes à décoder : leur nombre d'octets est variable et elles s'accompagnent de nombreuses options. On peut néanmoins améliorer les performances en effectuant ce décodage indépendamment de la logique d'ordonnement ou du pipeline. Nous reviendrons sur le sujet au chapitre 14.

Le cache de données exploite la stratégie d'écriture différée : les données sont écrites dans la mémoire principale uniquement lorsqu'elles sont supprimées du cache et qu'elles ont été modifiées. On peut configurer dynamiquement le processeur du Pentium 4 pour qu'il utilise la politique d'écriture simultanée.

Le cache de données L1 est contrôlé par deux bits de l'un des registres de contrôle, appelés bits CD (*Cache Disable*, cache désactivé) et NW (*Not Write-Through*, pas d'écriture simultanée) (voir tableau 4.4). On peut également faire appel à deux instructions du Pentium 4 pour contrôler le cache de données : INVD invalide (purge) la mémoire cache interne et demande au cache externe (s'il y en a un) d'invalider. WBINVD utilise l'écriture différée pour invalider le cache interne avant de faire de même avec le cache externe.

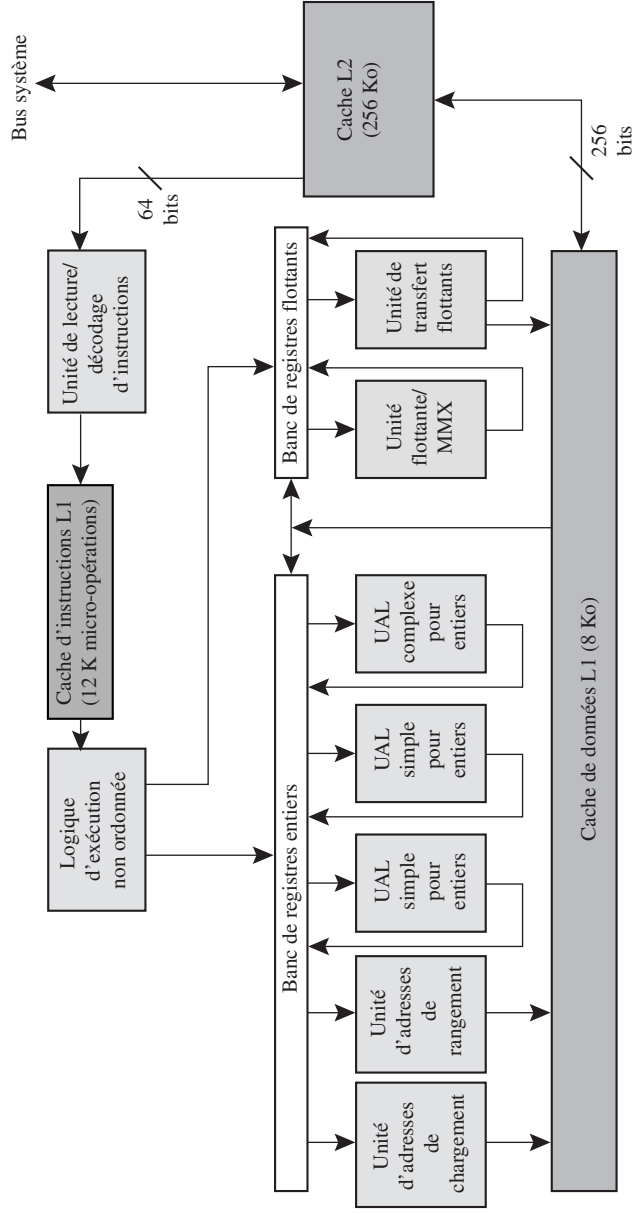


Figure 4.13 • Diagramme en blocs du Pentium 4.

Tableau 4.4 • Modes de fonctionnement du cache du Pentium 4

Bits de contrôle		Mode de fonctionnement		
CD	NW	Remplissage du cache	Écritures simultanées	Invalidations
0	0	Activé	Activé	Activé
1	0	Désactivé	Activé	Activé
1	1	Désactivé	Désactivé	Désactivé

Note : CD = 0, NW = 1 est une combinaison non valide.

Organisation des caches du PowerPC

L'organisation des caches du PowerPC a évolué en même temps que l'architecture globale de la gamme des PowerPC, reflétant la recherche continue de performances qui motive tous les concepteurs de microprocesseurs.

Le tableau 4.5 montre cette évolution. Le modèle d'origine, le 601, ne contient qu'un cache code/données 32 Ko avec l'organisation associative 8 voies. Le 603 emploie une conception RISC plus sophistiquée mais possède un cache de taille moins importante : 16 Ko divisés en deux caches, instructions et données, chacun utilisant l'organisation associative 2 voies. Le résultat est que le 603 obtient approximativement les mêmes performances que le 601 à moindre coût. Les modèles 604 et 620 doublent chacun la taille des caches par rapport au modèle précédent. Les modèles G3 et G4 sont équipés de caches L1 de la même taille que le modèle 620.

Tableau 4.5 • Caches internes du PowerPC.

Modèle	Taille	Octets/ligne	Organisation
PowerPC 601	1 32 Ko	32	associatif 8 voies
PowerPC 603	2 8 Ko	32	associatif 2 voies
PowerPC 604	2 16 Ko	32	associatif 4 voies
PowerPC 620	2 32 Ko	64	associatif 8 voies
PowerPC G3	2 32 Ko	64	associatif 8 voies
PowerPC G4	2 32 Ko	32	associatif 8 voies

La figure 4.14 montre un schéma simplifié de l'organisation du PowerPC et met en évidence les positions des deux caches. Les unités d'exécution se composent de deux unités logiques arithmétiques entières, qui peuvent s'exécuter en parallèle, et d'une unité virgule flottante possédant ses propres registres et composants de multiplication, d'addition et de division. Le cache de données alimente les opérations entières et virgule flottante *via* une unité de chargement/rangement. Le cache d'instructions, qui est en lecture seule, alimente une unité d'instructions, dont nous étudierons le fonctionnement au chapitre 14.

Les caches L1 sont associatifs 8 voies et le cache L2 est associatif 2 voies avec 256 Ko, 512 Ko ou 1 Mo de mémoire.

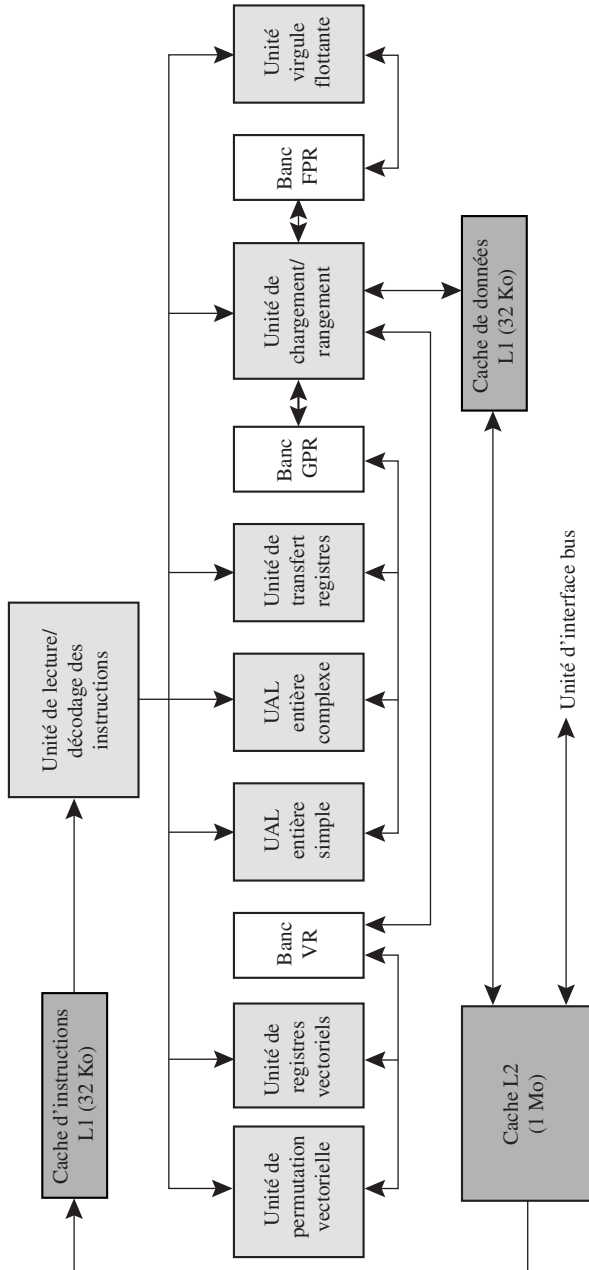


Figure 4.14 • Diagramme en blocs du PowerPC.

Questions et problèmes

Questions

- 4.1 Quelles sont les différences entre l'accès séquentiel, l'accès direct et l'accès aléatoire ?
- 4.2 Quelle est la relation générale entre le temps d'accès, le coût de la mémoire et la capacité ?
- 4.3 Quelle est la relation entre le principe de localité et l'utilisation de plusieurs niveaux de mémoire ?
- 4.4 Quelles sont les différences entre correspondance directe, correspondance associative et correspondance associative par ensemble ?
- 4.5 Pour un cache à correspondance directe, l'adresse de la mémoire principale se compose de trois champs. Listez et définissez ces trois champs.
- 4.6 Pour un cache associatif, l'adresse de la mémoire principale se compose de deux champs. Listez et définissez ces deux champs.
- 4.7 Pour un cache associatif par ensemble, l'adresse de la mémoire principale se compose de trois champs. Listez et définissez ces trois champs.
- 4.8 Quelle est la différence entre la localité spatiale et la localité temporelle ?
- 4.9 En règle générale, quelles sont les stratégies employées pour exploiter la localité spatiale et la localité temporelle ?

Problèmes

- 4.1 Un cache associatif par ensemble se compose de 64 lignes, ou emplacements, divisés en ensembles de quatre lignes. La mémoire principale contient des blocs de 4 Ko de 128 mots chacun. Donnez le format des adresses de la mémoire principale.
- 4.2 Pour les adresses hexadécimales de la mémoire principale 111111, 666666 et BBBBBB, donnez les informations suivantes au format hexadécimal :
 - a. Les valeurs de Étiquette, Ligne et Mot pour un cache à correspondance directe, en vous servant du format de la figure 4.8.
 - b. Les valeurs de Étiquette et Mot dans un cache totalement associatif, en vous servant du format de la figure 4.12.
 - c. Les valeurs de Étiquette, Ensemble et Mot d'un cache associatif 2 voies en vous servant du format de la figure 4.12.

- 4.3** Listez les valeurs suivantes :
- Pour l'exemple de cache à correspondance directe de la figure 4.8 : longueur de l'adresse, nombre d'unités adressables, taille de bloc, nombre de blocs dans la mémoire principale, nombre de lignes dans le cache et taille de l'étiquette.
 - Pour l'exemple de cache associatif de la figure 4.10 : longueur de l'adresse, nombre d'unités adressables, taille de bloc, nombre de blocs dans la mémoire principale, nombre de lignes dans le cache et taille de l'étiquette.
 - Pour l'exemple de cache associatif 2 voies de la figure 4.12 : longueur de l'adresse, nombre d'unités adressables, taille de bloc, nombre de blocs dans la mémoire principale, nombre de lignes dans l'ensemble, nombre d'ensembles, nombre de lignes dans le cache et taille de l'étiquette.
- 4.4** Prenez l'exemple d'un microprocesseur 32 bits équipé d'un cache interne 16 Ko associatif 4 voies. Supposez que la taille de ligne du cache soit de quatre mots de 32 bits. Dessinez un diagramme en blocs de ce cache montrant son organisation et la manière dont les différents champs d'adresses servent à déterminer un succès/échec de cache. Où place-t-on dans le cache le mot d'adresse mémoire ABCDE8F8 ?
- 4.5** Étant donnée les spécifications suivantes d'un cache externe : associatif 4 voies, taille de ligne de deux mots de 16 bits, pouvant contenir 4 K de mots 32 bits provenant de la mémoire principale, utilisée par un processeur 16 bits qui émet des adresses 24 bits, concevez la structure du cache avec toutes les informations pertinentes et montrez comment elle interprète les adresses du processeur.
- 4.6** Le 40486 d'Intel est équipé d'un cache interne unifié. Il contient 8 Ko, utilise l'organisation associative 4 voies et des blocs de quatre mots 32 bits. Le cache est organisé en 128 ensembles. Il y a un seul « bit de ligne valide » et trois bits B0, B1 et B2 (les bits « LRU »), par ligne. Sur un échec cache, le 80486 lit une ligne de 16 octets de la mémoire principale en une rafale de lecture mémoire. Dessinez un diagramme simplifié du cache et montrez comment sont interprétés les différents champs de l'adresse.
- 4.7** Prenez l'exemple d'un ordinateur équipé d'une mémoire principale adressable en octets de 2^{16} octets avec une taille de bloc de 8 octets. Supposez qu'on exploite un cache à correspondance directe composé de 32 lignes sur cet ordinateur :
- Comment une adresse mémoire 16 bits est-elle divisée en étiquette, nombre de lignes et nombre d'octets ?
 - Dans quelle ligne les octets de chacune des adresses suivantes sont-ils rangés ?
 0001 0001 0001 1011
 1100 0011 0011 0100
 1101 0000 0001 1101
 1010 1010 1010 1010

- c. Supposez que l'octet dont l'adresse est 0001 1010 0001 1010 soit rangé dans le cache. Quelles sont les adresses des autres octets rangés avec lui ?
- d. Combien d'octets de mémoire peuvent être rangés dans le cache ?
- e. Pourquoi l'étiquette est-elle également rangée dans le cache ?
- 4.8 Pour son cache interne, le 80486 d'Intel utilise un algorithme de remplacement appelé **pseudo moins récemment utilisé**. Trois bits (B0, B1 et B2) sont associés à chacun des 128 ensembles de quatre lignes (L0, L1, L2 et L3). L'algorithme de remplacement fonctionne de la manière suivante : lorsqu'on doit remplacer une ligne, le cache commence par déterminer si la plus récente utilisation s'est produite en L0 et L1 ou L2 et L3. Il détermine ensuite quelle paire de blocs a été utilisée le moins récemment et la marque pour remplacement. La figure 4.15 illustre cette logique.
- a. Indiquez comment les bits B0, B1 et B2 sont positionnés et décrivez en quelques mots leur utilisation dans l'algorithme de remplacement décrit dans la figure 4.15.
- b. Montrez que l'algorithme 80486 s'approche d'un vrai algorithme LRU. *Astuce* : envisagez le cas de figure où l'ordre d'utilisation le plus récent est L0, L2, L3 et L1.
- c. Démontrez qu'un véritable algorithme LRU demande 6 bits par ensemble.

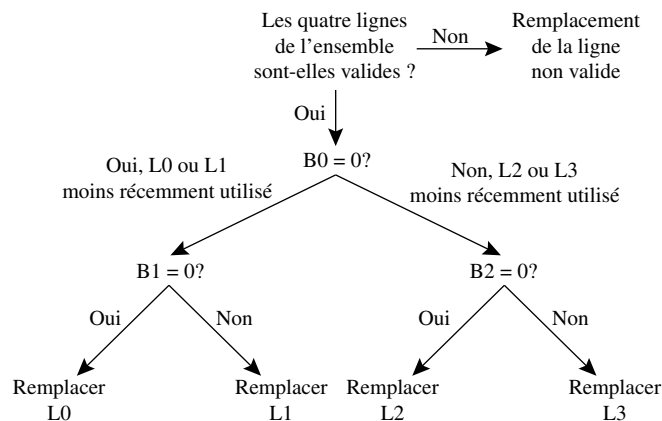


Figure 4.15 • Stratégie de remplacement du cache interne Intel 80486.

- 4.9 Un cache associatif par ensemble avec lignes de quatre mots de 16 bits et ensembles de 2 lignes peut contenir 4 048 mots. La taille de la mémoire principale cachable est de 64 Ko × 32 bits. Concevez la structure du cache et montrez comment sont interprétées les adresses du processeur.
- 4.10 Une mémoire système utilise une adresse 32 bits pour adresser au niveau octets, plus un cache avec lignes de 64 octets.

- a. Imaginez un cache à correspondance directe avec un champ étiquette de 20 bits dans l'adresse. Indiquez le format d'adresse et déterminez les paramètres suivants : nombre d'unités adressables, nombre de blocs dans la mémoire principale, nombre de lignes dans le cache et taille de l'étiquette.
- b. Imaginez un cache totalement associatif. Indiquez le format d'adresse et déterminez les paramètres suivants : nombre d'unités adressables, nombre de blocs dans la mémoire principale, nombre de lignes dans le cache et taille de l'étiquette.
- c. Imaginez un cache associatif 4 voies avec un champ étiquette de 9 bits dans l'adresse. Indiquez le format d'adresse et déterminez les paramètres suivants : nombre d'unités adressables, nombre de blocs dans la mémoire principale, nombre de lignes dans un ensemble, nombre d'ensembles dans le cache, nombre de lignes dans le cache et taille de l'étiquette.

4.11 Décrivez une technique simple pour implémenter un algorithme de remplacement LRU dans un cache associatif 4 voies.

4.12 Pour l'exemple de code suivant :

```

for (i = 0; i < 20; i++)
    for (j = 0; j < 10; j++)
        a[i] = a[i] * j
  
```

- a. Donnez un exemple de localité spatiale dans le code.
 - b. Donnez un exemple de localité temporelle dans le code.
- 4.13 Généralisez les équations (4.1) et (4.2), dans l'annexe 4A, à des hiérarchies de mémoire de niveau N .
- 4.14 Un ordinateur est équipé d'une mémoire principale de 32 K mots de 16 bits, d'un cache 4 K mots divisé en ensembles de quatre lignes avec 64 mots par ligne. Supposez que le cache est initialement vide. Le processeur lit des mots d'adresses 0, 1, 2, ..., 4351 dans cet ordre. Il répète ensuite neuf fois la séquence de lectures. Le cache est dix fois plus rapide que la mémoire principale. Estimez l'amélioration résultant de l'utilisation du cache. Tenez compte d'une stratégie LRU pour les remplacements des blocs.
- 4.15 Un système mémoire possède les paramètres suivants :
- | | |
|-------------------|---------------------|
| $T_c = 100$ ns | $C_c = 0,01$ €/bit |
| $T_m = 1\ 200$ ns | $C_m = 0,001$ €/bit |
- a. Quel est le prix de 1 Mo de mémoire principale ?
 - b. Quel est le prix de 1 Mo de mémoire principale utilisant la technologie de la mémoire cache ?
 - c. Si le temps d'accès utile est supérieur de 10 % à celui du cache, quel est le taux de succès S ?
- 4.16 Un ordinateur est équipé d'un cache, d'une mémoire principale et d'un disque utilisé pour la mémoire virtuelle. Si un mot référencé se trouve dans le cache,

il faut 20 ns pour y accéder. S'il se trouve dans la mémoire principale et non dans le cache, il faut 60 ns pour le charger dans le cache et redémarrer l'accès. Si le mot ne se trouve pas dans la mémoire principale, il faut 12 ms pour l'obtenir du disque, suivies de 60 ns pour le copier dans le cache avant de redémarrer la référence. Le taux de succès du cache est de 0,9 et celui de la mémoire principale est de 0,6. Quel est le temps moyen en ns nécessaire pour accéder à un mot référencé sur ce système ?