UCL Département d'ingénierie informatique

Le niveau zéro

Le niveau zéro concerne le matériel proprement dit (le hardware). On n'étudiera pas, dans ce document, tout le matériel utilisé dans les ordinateurs, mais uniquement quelques concepts et quelques techniques qui ne sont pas apparus dans le présentation de la machine simplifiée mais qui jouent un rôle important dans la performance des ordinateurs et sont trop spécifiques à ceux-ci pour trouver place dans un cours d'électronique.

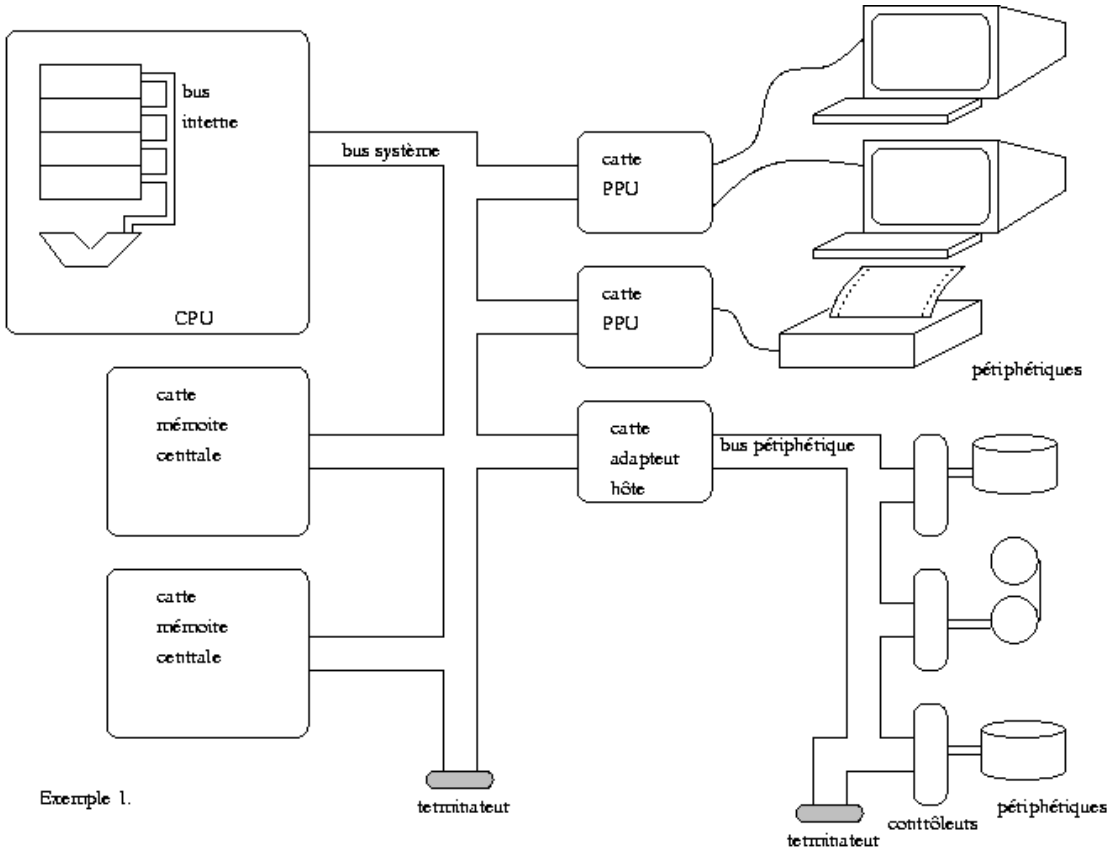
1. Les bus

On trouvera une présentation plus complète des bus dans le cours "Bases électroniques de l'informatique". Seul leur rôle est présenté ici.

La connection électrique entre deux registres s'appelle un **BUS**. Un bus est traditionellement constitué de fils parallèles, un pour chaque bit dans les registres. Deux registres de 16 bits seront connectés par un bus de 16 fils. On dit alors que la largeur de ce bus est 16. Les bits sont transmis en parallèle et simultanément sur ces lignes des bus, appelées lignes de données. Un bus comporte, en général, un certain nombre de lignes supplémentaires pour des informations de contrôle et de synchronisation. On rencontre aussi des bus série où les différents bits sont transmis à la suite les uns les autres sur les (deux) mêmes fils. Les informations de contrôle sont alors des séquences de bits bien connues et aisément identifiables transmises sur ces mêmes fils. Il n'est plus nécessaire de synchroniser entre eux les signaux transmis sur des fils parallèles puisqu'ils sont transmis en série. Le seul problème de synchronisation est alors pour récepteur d'identifier où finit chaque bit et où commence le suisvant.

Les bus, quoique constitués principalement de simples fils électriques auront une importance considérable sur les performances et le comportement de l'ordinateur. Ces bus servent de support physique aux échanges d'informations dans les ordinateurs. Comme toutes les communications, celles qui s'effectuent sur les bus sont régies par des protocoles, c'est à dire des règles qui fixeront qui "parlera", qui "écoutera" et quand. Comme on est au niveau "hardware", il s'agit de protocoles électriques.

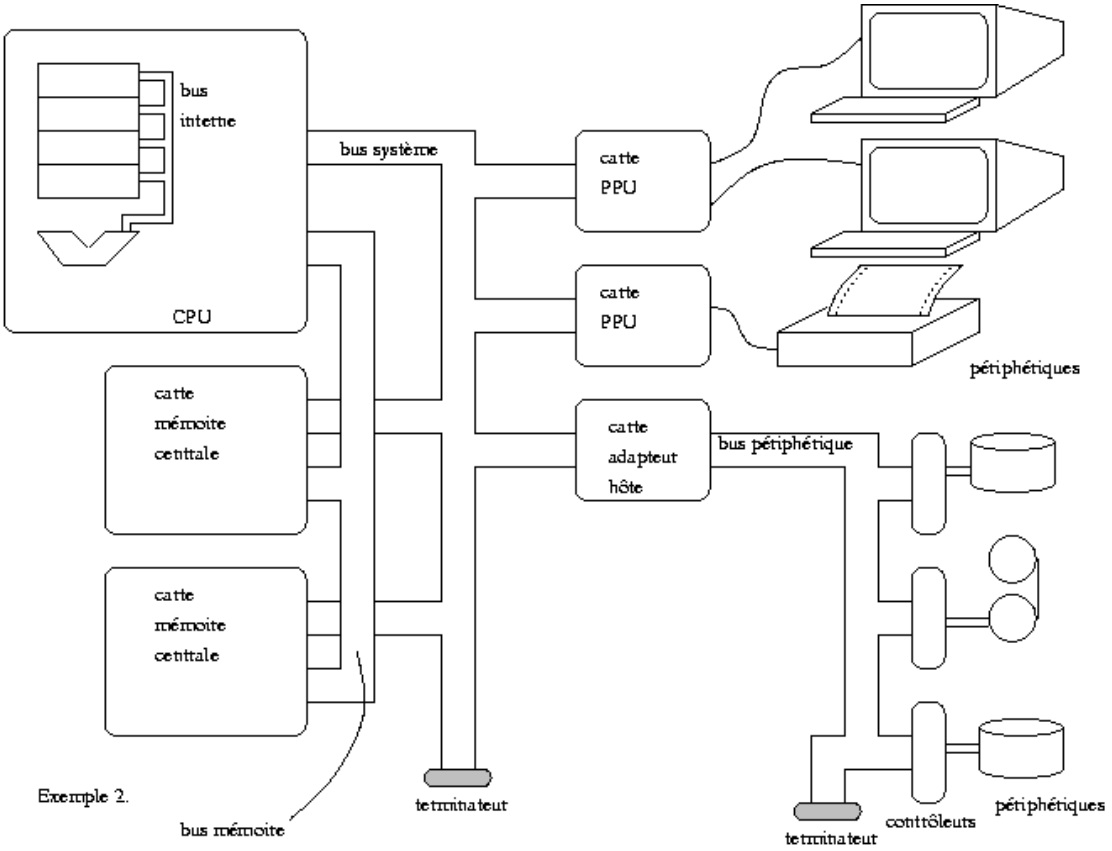
Il existe des bus à l'intérieur du CPU. Ils connectent entre eux registres, `ALU`, etc... dans ce CPU. Il existe aussi des bus qui servent à l'interconnection du CPU avec les autres éléments de la machine auquels le CPU a directement accès (ces éléments font en général partie de l'architecture de la machine de niveau 2): mémoire centrale, objets partagés avec les processeurs périphériques. Enfin, il existe des bus entre les processeurs périphériques et les contrôleurs de ces périphériques proprement dits.



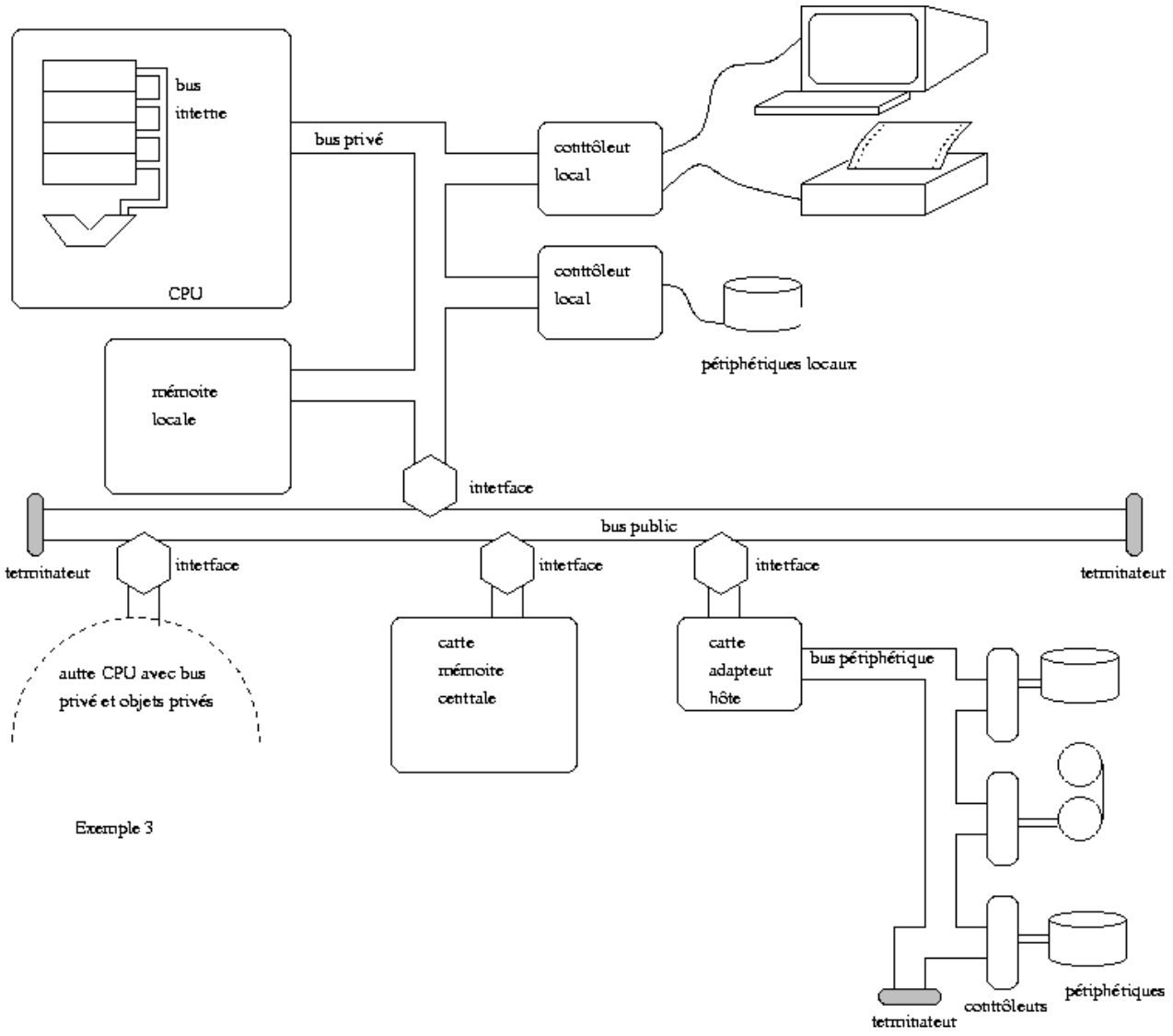
Exemple 1.

exemple 1: les bus dans une machine comme une PDP11 ou un PC-XT: les différents éléments internes à la machine (CPU, mémoire, contr$oleurs de periphériques) sont interconnectés par un bus système unique (UNIBUS ou QBUS pour les PDP,

bus XT pour les PC). Le bus périphérique est par exemple un bus SCSI. (si l'USB avait existé à l'époque c'aurait aussi pu être un USB)



exemple 2: on préfère souvent doter la machine d'un bus séparé pour la mémoire. ce bus est alors plus rapide (et permet un transfert de plus de bits en parallèles) que le bus système pqrtagé avec les contrôleurs de périphériques. Le VAX (successeur de la PDP11 avait une architecture de ce genre, de même que les PC (ceux-ci ont souvent, en plus un bus AGP entre mémoire et contrôleur d'écran). Dans le cas des PC, bus système et bus mémoire n'aboutissent pas au CPU mais au "chipset", lui même connecté au CPU. Les bus ISA et PCI sont des exemple de bus système utilisés dans ce genre de contexte.



Exemple 3

exemple 3:ordinateur de contôle industriel typique: c'est un lultiprocesseur à mémoire partagée en partie. le bus public est, par exemple, un VME

Exemples de bus

- BUS systèmes (pour un seul CPU) : UNIBUS (PDP 11), bus XT, bus ISA (AT), bus PCI
- BUS périphériques : canal IBM, BUS SCSI (variantes 1, 2 et 3), BUS USB
- BUS publics (multi-CPU) : MULTIBUS I et II (INTEL), VME (des dizaines de constructeurs dont MOTOROLA, SUN, etc.)
- BUS mémoire rapides : CMI, SBI (VAX), VSB (utilisé avec VME), VESA (utilisé avec ISA)

2. Techniques d'accélération des transferts entre mémoire centrale

2.1 Mémoires entrelacées

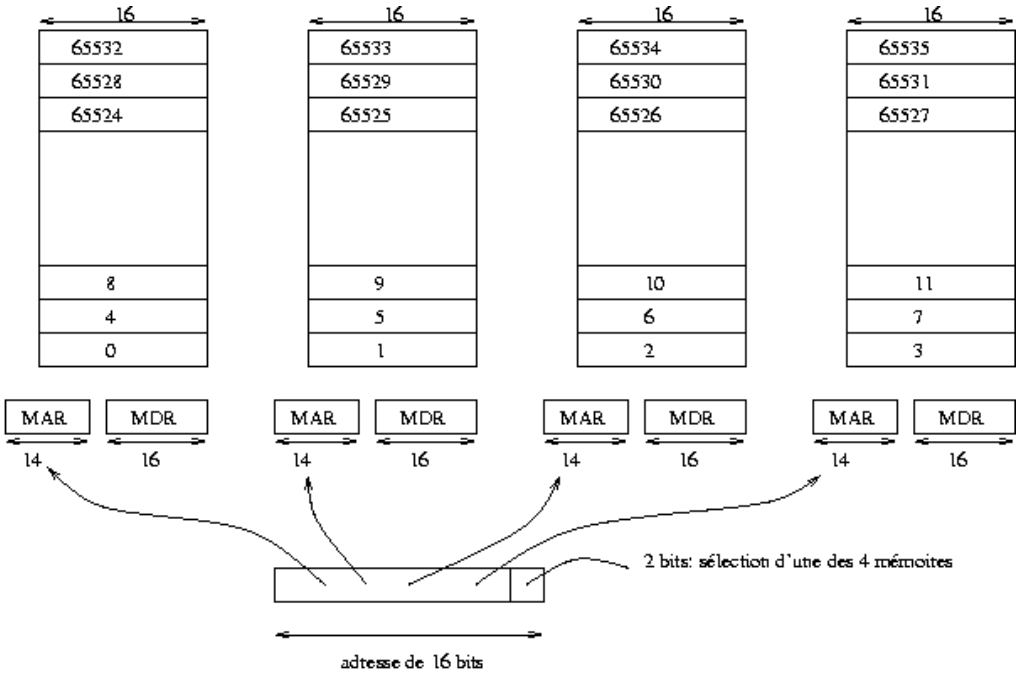
La technique la plus évidente pour augmenter la quantité d'information transférée de la mémoire centrale au CPU chaque seconde (sans utiliser de mémoire centrale plus rapide) est d'utiliser un bus plus large.

Ceci implique cependant une modification fondamentale de l'architecture de la machine car cela revient à modifier la taille du mot.

Une technique alternative est de diviser la mémoire en plusieurs bancs, par exemple 4. On placera dans le premier banc les mots dont l'adresse est multiple de 4, dans le deuxième, les (multiples de 4) plus 1, dans le troisième les (multiples de 4) plus 2, etc. .

On dotera l'ordinateur de 4 MAR et de 4 MDR et chaque fois qu'une lecture sera effectuée dans une case d'un banc, on lira en même temps les cases correspondantes des autres bancs. Si lors de l'accès mémoire suivant, on a besoin d'un de ces mots, il sera déjà prêt dans le MDR de son banc.

On peut remarquer que ces deux solutions: bus plus large ou mémoires entrelacées reviennent toutes deux finalement à la même chose : transférer plus de bits simultanément. Seule l'organisation physique diffère.



les 14 bits de gauche de l'adresse sont envoyés simultanément comme adresse aux 4 mémoires

2.2 "Caches" ou "High Speed Memory Buffers" ou antémémoires

Si la durée d'un cycle d'accès à la mémoire centrale était la même que celle d'un cycle d'accès à un registre du CPU, le processeur ne devrait jamais perdre de temps à attendre la fin d'un cycle mémoire. Il est trop coûteux de réaliser toute la mémoire centrale dans la même technologie que les registres. Cependant, il est possible de réaliser une petite partie de la mémoire en circuits rapides. Si on arrive à s'arranger pour que la plupart des accès mémoires se fassent dans cette petite partie (par exemple par ce que les cases de la mémoire centrale accédées le plus souvent y ont été copiées), cette mémoire rapide cachera au processeur la lenteur de la mémoire centrale. De là vient le nom "\fBcache\fR". Cette mémoire rapide servira de tampon (BUFFER) entre processeur et mémoire centrale.

L'utilisation d'une cache permettra de réduire le nombre d'accès à la mémoire centrale pour les raisons suivantes :

1. Les transferts de la mémoire centrale vers la cache peuvent avoir lieu un bloc (c'est à dire plusieurs mots) à la fois. Si la mémoire est entrelacée avec un facteur d'entrelacement égal au nombre de mots d'un bloc, les transferts des différents mots du bloc peuvent avoir lieu en même temps, en un seul cycle d'accès à la mémoire. Sinon, il est souvent possible d'accélérer les transferts par bloc en n'envoyant à la mémoire que l'adresse du premier mot du bloc. La mémoire répondra en renvoyant l'un après l'autre tous les mots du bloc. Ce mode de travail s'appelle "*streaming*". Pour le demander, il faut une ligne de contrôle supplémentaire sur le bus.
2. Le transfert par bloc est un genre de "prefetching" puisque, lors de la lecture d'une instruction du programme, les instructions suivantes situées dans le même bloc sont ramenées dans la cache. Elles s'y trouveront donc quand on en aura besoin.
3. Souvent, le contenu de la cache sera utilisé plusieurs fois car les programmes contiennent de nombreuses boucles. Ceci réduit encore le nombre d'accès à la mémoire centrale. les variables sont aussi, le plus souvent, accédées plusieurs fois.

Comme il s'agit d'accélérer les accès à la mémoire, la gestion de la cache, c'est à dire le choix de ce qui sera copié de la mémoire centrale vers la cache et vice versa, aura lieu entièrement en hardware.

La cache est divisée en blocs de quelques mots mémoire et les transferts entre mémoire centrale et cache se font toujours par blocs entiers. Lorsque le processeur veut lire le contenu d'un mot mémoire, il le lira dans la cache si ce mot appartient à un des blocs de la cache, sinon, le bloc correspondant sera copié de la mémoire centrale vers la cache: les transferts mémoire vers cache sont fait **à la demande**.

Le problème des transferts de la mémoire vers la cache peut se formuler ainsi: si je veux lire un bloc de la mémoire centrale, existe-t-il une copie de ce bloc dans la cache et, si oui, où dans la cache et, si non, où mettre mon nouveau bloc dans cette cache. Il s'agit d'établir une correspondance entre les blocs de la cache et les blocs de la mémoire centrale.

Une solution serait de donner à chaque bloc de la mémoire centrale une adresse et d'ajouter à chaque bloc de la cache\fI un registre contenant l'adresse du bloc de la mémoire centrale qui y est copié\fR. Ce registre est appelé TAG ou étiquette.

Lors d'un accès à un mot d'un bloc, les contenus de tous les registres étiquettes seraient comparés **simultanément** (il faut donc autant de comparateurs que de registres) avec l'adresse du bloc recherché. L'ensemble de ces registres forme une mémoire dont le choix d'un objet se fait par son contenu et non pour son adresse. On appelle cela une **mémoire associative**. Si une des comparaisons donne un résultat positif, le mot demandé se trouve dans **ce** bloc de la cache, sinon il faut remplacer un des blocs (p. ex. celui qui est dans la cache depuis le temps le plus long sans être utilisé: ceci se dit "Least Recently Used": LRU) par le bloc demandé.

Cette solution est cependant trop coûteuse: la cache peut contenir des centaines de blocs : il faudrait donc des centaines de registres.

Les organisations utilisées en pratique ne sont que des variantes de la précédente. Le but est toujours de réduire le nombre de registres à comparer tout en conservant un comportement aussi proche que possible de cette cache idéale.

Les transferts en sens inverse (cache vers mémoire) des blocs dont le contenu a été modifié par une opération d'écriture peuvent se faire de deux manières:

1. **write back**: on ne réécrit les blocs modifiés que lorsque la place qu'ils occupent dans la cache doit être libérée pour faire place à un autre bloc. Ceci implique, bien sûr, que le hardware "marque" les blocs dont le contenu est modifié, de manière à ne pas recopier en mémoire centrale des blocs non modifiés.
2. **write through**: toute écriture se fait simultanément dans la cache et dans la mémoire centrale. Cette technique est plus simple à implémenter, mais moins efficace puisqu'on ne bénéficiera pas de la rapidité de la cache pour les opérations d'écriture. Comme les opération[Es de lecture sont les plus nombreuse, la cache reste cependant utile.

Le reste de ce paragraphe sera consacré à l'organisation et au fonctionnement des caches.

Les caractéristiques d'une cache sont, en ordre décroissant d'importance:

- méthode de choix de la position où placer dans la cache un bloc en provenance de la mémoire centrale
- taille de la cache
- taille des blocs
- stratégie de réécriture des blocs de la cache
- algorithme de remplacement
-

2.2.1 Organisation par secteurs

La première cache a été implémentée dans l'IBM 360/85. Elle était organisée par secteurs.

L'idée de base de l'organisation par secteurs est la suivante: dans la cache idéale, la manière la plus évidente de réduire le nombre de registres à comparer sans réduire le taille de la cache est d'utiliser des blocs plus grands. Ceci impose soit l'emploi d'un nombre immense de mémoires entrelacées (trop cher) soit le transfert successif des différents mots du bloc vers la cache (trop lent). Cette dernière possibilité ne serait pas grave (du moins dans un système de traitement par lots, où seule compte l'optimisation de la vitesse moyenne de traitement) si on était sûr d'utiliser tôt ou tard tous les mots ainsi transférés de la mémoire vers la cache, mais ceci est très peu probable. On aura donc sacrifié des cycles pour aller chercher des choses inutiles en mémoire. Or le but de la cache est précisément d'économiser des cycles d'accès à la mémoire. Les deux solutions envisagées pour transférer de grands blocs vers la cache sont donc inacceptables.

Dans l'organisation par secteurs, on appelle les grands blocs *secteurs* et on divise ces derniers en petits blocs dont la taille est compatible avec le niveau d'entrelacement de la machine ou le nombre de cycles que l'on accepte de consacrer au transfert de chaque petit bloc vers la cache.

Dans l'organisation par secteurs, tout comme dans la cache idéale, chaque secteur de la mémoire centrale peut se retrouver dans n'importe quel secteur de la cache. La localisation du secteur dans la cache se fait par une recherche associative sur les étiquettes de secteurs ("TAGS") qui contiennent l'adresse du secteur correspondant en mémoire centrale.

Tout comme dans la cache idéale, lorsqu'un secteur doit être ramené vers la cache, il prendra la place d'un secteur qui s'y trouvait déjà. \fILa différence essentielle est que le secteur ne sera pas ramené en entier vers la cache: seul le petit bloc contenant le mot recherché sera ramené. \fRIl sera placé dans le secteur de la cache à une \fIposition relative\fR identique à celle qu'il occupait dans le secteur en mémoire centrale. Par conséquent, les secteurs de la cache ne seront, en général pas complets: certains blocs y seront et d'autres pas. Lorsqu'on cherchera un mot dans la cache, il faudra, non seulement trouver le secteur correspondant par recherche associative, mais encore, si le secteur a été trouvé, vérifier si le bloc correspondant a été ramené dans la cache et, dans le cas contraire, l'y ramener. L'information "un bloc est dans la cache ou non" est binaire: comme la position relative du bloc doit être la même dans le secteur de la mémoire centrale et dans celui de la cache, il n'y aura qu'un bit à vérifier. Ce bit s'appelle **bit de validation du bloc** ("`valid bit`"). Si ce bit est *vrai*, il suffira d'extraire le mot voulu grâce à son adresse relative dans le bloc. Si le bit est faux, il faut d'abord ramener le bloc de la mémoire centrale vers la cache et mettre le "`valid bit`" à *vrai*.

Du point de vue de la cache, les adresses sont donc divisées en trois parties.

- Les bits les plus significatifs indiquent le No de secteur. On les utilise pour la recherche associative.
- Les bits de poids moyen identifient le No de bloc dans le secteur. On les utilise pour localiser le bloc et son "valid bit" dans le secteur.
- Les bits les moins significatifs donnent l'adresse du mot à l'intérieur du bloc.

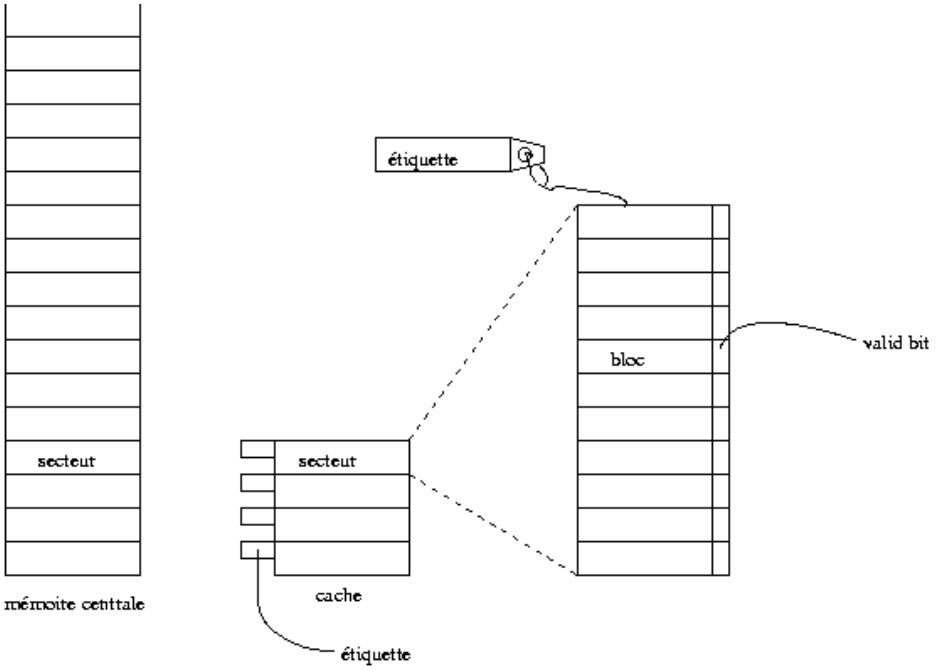
Lorsque le secteur recherché n'est pas dans la cache, on choisit un secteur de la cache pour l'y ramener. On met tous les "valid bits" de ce

secteur de la cache à \flfaux\fR, on met l'adresse du nouveau secteur dans l'étiquette de celui de la cache. On ramène le bloc à accéder vers la cache et on met son "valid bit" à \flvrai\fR.

Pour l'écriture, la cache à organisation par secteurs utilise une politique "write through". Sans cela, on risquerait de devoir réécrire jusqu'à un secteur complet lorsqu'il faut l'éjecter de la cache pour faire place à un autre. Une politique "write back" ne pourrait en effet pas se baser sur l'éjection des blocs car on ne peut retrouver l'adresse de ceux-ci qu'aussi longtemps que le secteur dont il fait partie est dans la cache.

Lorsqu'un secteur est éjecté, un seul de ses blocs est remplacé mais les autres ne sont pas utilisables car il n'est plus possible de déduire leur adresse en mémoire. Il faudrait donc les considérer comme perdus et les recopier en mémoire lors de l'éjection du secteur en cas de politique "write back". La quantité d'information à recopier à ce moment étant trop importante, IBM a préféré la politique "write through" qui est, par ailleurs, plus simple à implémenter.

> La cache de l'IBM 360/85 contenait 16 secteurs de 1K bytes (donc 256 mots), chaque secteur était divisé en 16 blocs de 64 bytes (donc 16 mots). Lorsqu'un bloc est ramené dans la cache, ses 4 premiers mots sont d'abord amenés simultanément (la 360/85 avait une mémoire entrelacée à 4 bancs) et le reste suit lors des cycles suivants. L'algorithme de remplacement est LRU.



2.2.2 Organisation à correspondance directe ("direct mapping")

Dans cette organisation, la cache contient un seul secteur, mais chaque bloc est doté, non d'un "`valid bit`" mais d'une étiquette. *Les différents blocs de la cache peuvent donc appartenir à des secteurs différents de la mémoire centrale*. Leur position relative dans un secteur de mémoire centrale et dans la cache est cependant la même. Si on numérote tous les blocs de la mémoire centrale et si la cache peut contenir N blocs, les blocs k, k+N, k+2N correspondent au bloc k de la cache. Pour retrouver un mot, on comparera d'abord les bits les plus significatifs de son adresse au contenu de l'étiquette du bloc de la cache désigné par les bits de poids moyen. Les bits de poids faible permettent de localiser le mot dans le bloc. Il n'y a donc pas de recherche associative car si un bloc est dans la cache, il ne peut y être qu'à un seul endroit. Il n'y a donc pas d'algorithme de remplacement non plus car on ne peut pas choisir où mettre le bloc dans la cache.

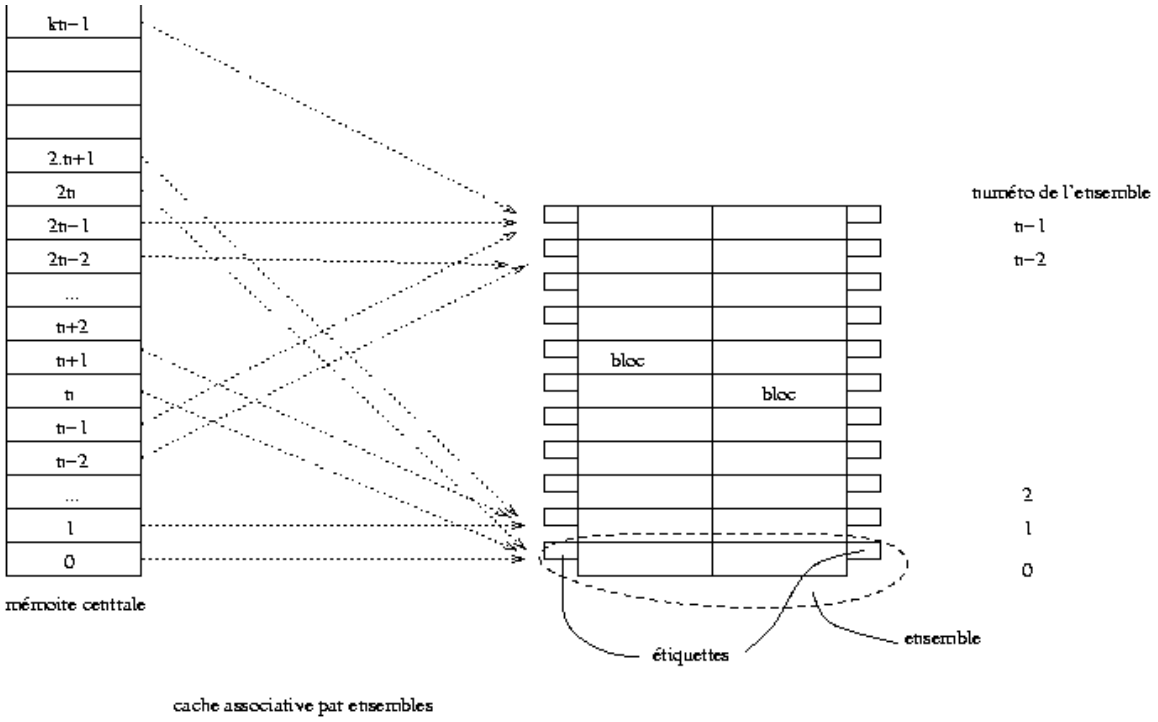
Cette organisation est celle qui peut se réaliser avec le matériel le plus simple. Elle peut cependant poser des problèmes d'efficacité si programme et données sont "mal" placés en mémoire. Ce sera le cas, par exemple, si on veut additionner deux vecteurs dont les adresses en mémoires diffèrent d'un multiple de la taille de la cache : les 2 opérandes de chaque addition voudront toujours venir dans le même bloc de la cache et celle-ci ne servira à rien car ce bloc sera remplacé à chaque accès.

> Cette organisation a été utilisée sur la PDP 11-60 avec des blocs de 1 mot. On la choisira quand on veut une cache bon marché.

2.2.3 Organisation associative par ensembles

On peut aisément faire sauter le défaut de l'organisation directe en n'imposant pas à un bloc de mémoire centrale de venir dans un bloc déterminé de la cache mais en lui laissant le choix parmi les blocs d'un ensemble. La cache ne contiendra alors pas N blocs mais N ensembles de quelques blocs (2 ou 4, par exemple). Le bloc k + (i x N) de la mémoire centrale ne devra donc plus venir vers le bloc k de la cache mais vers 1 des blocs de l'ensemble k. La recherche associative se fera donc sur les étiquettes des blocs d'un ensemble et non sur tous les blocs de la cache. Ceci réduit le nombre de comparateurs à quelques unités (2 ou 4) et préserve la simplicité de la cache à

organisation directe sans son inconvénient.



cache associative par ensembles

Cette organisation associative par ensembles est très populaire.

> Elle a été utilisée, par exemple, dans l'IBM 370/165. La cache comporte 8 Kbytes. Chaque bloc contient 32 bytes et la mémoire est entrelacée à 4 voies, ce qui permet de ramener un bloc en deux accès. La cache contient 256 blocs groupés en 64 ensembles (appelés colonnes) de 4 blocs chacuns. La mémoire centrale est aussi divisée logiquement en 64 colonnes de blocs (bloc 0 dans 1ère colonne, bloc 1 dans la deuxième,..., bloc 64 de nouveau dans la 1ère etc...) chaque bloc de la mémoire centrale peut trouver place dans un des blocs de la même colonne dans la cache. Parmi les blocs de chaque colonne de la cache, on utilise un algorithme de remplacement LRU.

Contrairement à l'organisation par secteurs, les organisations directe et associative par ensembles conviennent bien à une politique d'écriture "write back". Lorsqu'il faut éjecter un bloc de la cache, il est facile de recalculer vers quelle adresse en mémoire il faut le recopier : l'étiquette donne les bits les plus significatifs de l'adresse, et le No d'ordre du bloc dans la cache donne les bits de poids intermédiaire.

2.2.4 Remarque : interférences entre les caches et les dispositifs d'entrées/sorties

1. Transfert DMA de la mémoire centrale vers un périphérique

   Avec une politique d'écriture "write through" il n'y aura pas de problème car la mémoire centrale contient les bonnes valeurs, mais avec une politique "write back" on risque d'envoyer vers le périphérique des zones mémoire qui ne sont pas à jour : le contenu de certains mots modifiés dans la cache peut ne pas encore avoir été recopié en mémoire centrale. Les blocs correspondants doivent donc être recopiés en mémoire centrale avant le transfert, mais cela ralentira le processus en cours d'exécution à ce moment.

   Une autre solution serait de faire passer les transferts vers le périphérique à travers la cache. Si le processeur est sur une carte et la mémoire et les contrôleurs de périphériques sur d'autres cartes connectées au bus système, ceci peut être réalisé en plaçant la cache sur les cartes mémoires plutôt que sur la carte processeur. Cette solution impose le passage à travers le bus système pour arriver à la cache, ce qui ralentit l'accès.

   Si on place la cache près du processeur, cette cache devient beaucoup plus complexe et un mécanisme d'arbitrage des accès à la cache devient nécessaire.
2. Transfert DMA d'un périphérique vers la mémoire centrale

   Il peut y avoir dans la cache des copies de zones de mémoire centrale qui sont remplacées. Les informations de la cache ne sont, dès lors, plus à jour puisque l'original, dont la cache n'est qu'une copie, a été remplacé. Il faut donc que la cache reste à l'écoute des transferts d'entrées/sorties et marque comme invalides les blocs correspondant aux zones de mémoire dont le contenu a changé, ou se mette à jour "au vol".

   Si la machine fonctionne en "memory mapped i/o", il faut, en outre, veiller à ne pas cacher les registres des contrôleurs de périphériques. Les changements des contenus de ceux-ci causés par le contrôleur ne seraient pas répercutés sur leur copie dans la

cache.

2.2.5 Evaluation des performances d'une cache

Les performances d'une cache dépendent de différents paramètres :

- taille de la cache
- taille des blocs
- organisation de la cache

L'impact de l'algorithme de remplacement est très faible et la politique d'écriture n'a pas beaucoup d'influence. L'unité de mesure la plus courante pour évaluer les performances d'une cache est le "hit ratio", **h**:

$$h = \frac{\text{nombre d'accès dans la cache}}{\text{nombre total d'accès en mémoire de niveau 2}}$$

(1-h) est appelé le "miss ratio". Si t_b est le temps d'accès à la cache et et t_c le temps d'accès à la mémoire centrale, on peut calculer le temps d'accès moyen:

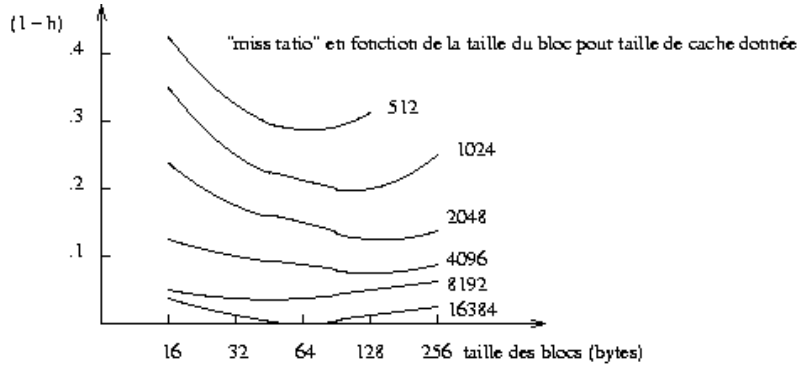
$$t_a = t_b + (1 - h) t_c$$

car on tentera toujours d'abord un accès dans la cache avant d'en faire un en mémoire centrale.

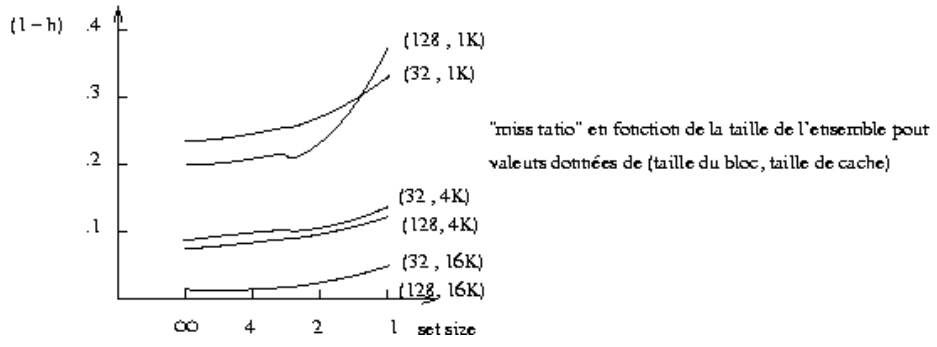
Ce calcul est approximatif car il néglige les interférences avec la mémoire, par exemple en cas de "write back". En outre, t\da\u ne donne pas forcément une idée exacte du temps de cycle moyen de l'ordinateur: l'accès à la cache peut durer de l'ordre de 50 nsec, ce qui est du même ordre que les microcycles de la micromachine et ne correspondra donc qu'à une partie du cycle de la machine de niveau 2.

Pour une taille donnée de la cache, il existe une taille optimale des blocs. Les résultats qui suivent datent des alentours de 1980. L'aspect des courbes est intéressant. Les valeurs numériques étaient valables pour les programmes utilisés à l'époque. Actuellement, ils sont jusqu'à 100 fois plus grands, surtout pour les machines RISC. Les caches nécessaires seront donc également plus grandes.

La première figure ci-dessous montre le "miss ratio" en fonction de la taille du bloc pour diverses tailles de la cache. Il apparaît que les tailles de blocs de 64 à 128 bytes conviennent le mieux mais que la taille du bloc n'est pas un facteur dominant, les courbes étant assez plates. La taille totale de la cache est plus importante.



La figure suivante indique le "miss ratio" en fonction de la taille de l'ensemble pour des caches associatives par ensemble (la taille 1 correspond à une organisation directe). Les courbes correspondent à diverses valeurs de la taille du bloc et de la cache. La seule différence notable apparaît quand on introduit plus d'un bloc dans l'ensemble. On ne gagne pas grand chose en mettant plus de deux blocs dans chaque ensemble.



Par ailleurs, des simulations montrent que des effacements de tout le contenu de la cache à intervales réguliers ont peu d'effet sur les performances de la cache.

En conclusion, une cache de 8 à 16 K avec des blocs de 64 à 128 bytes et une organisation associative par ensemble permettait des "hit ratio" supérieurs à 95%. Une organisation directe permettait des "hit ratio" de 80%.

Conception et réalisation: Marc Lobelle