ALMO

Caches

Problème d'accès à la mémoire

Le MIPS lit jusqu'à une instruction par cycle mais l'accès à la mémoire peut prendre plusieurs dizaines de cycles, c'est incompatible.
La solution est d'utiliser des caches ...

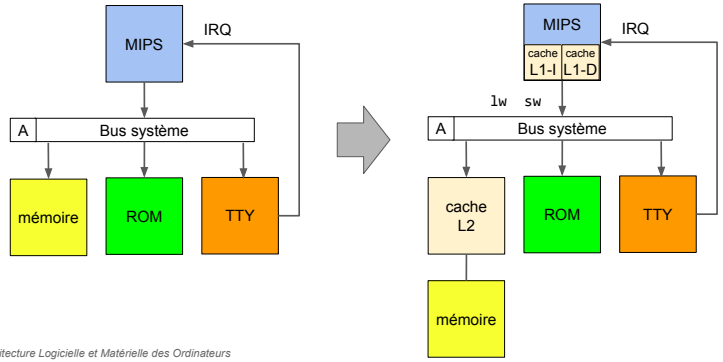
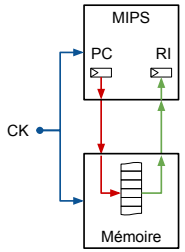
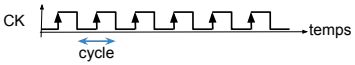


Schéma simplifié de connexion MIPS-mémoire

Lecture de l'instruction suivante par le MIPS dans la mémoire



- Le MIPS et sa mémoire sont cadencés par une horloge unique (un signal périodique).



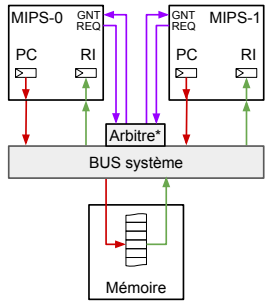
- Un cycle d'horloge (une période entre 2 fronts actifs) est le temps d'un calcul élémentaire.
- Le MIPS demande une nouvelle instruction à chaque cycle (*sur le schéma, on ne fait pas apparaître les accès de données*).
- Il faut donc que la mémoire réponde en un cycle,
- C'est possible mais avec avec une horloge lente, sinon il faut geler (stopper) le MIPS le temps que l'instruction arrive de la mémoire et il est alors impossible de démarrer une instruction par cycle

Dans le schéma réel
Le MIPS et sa mémoire ne sont pas seuls.

- Il y a plusieurs composants mappant des adresses de l'espace d'adressage
- Il peut y avoir plusieurs cœurs se partageant le même espace d'adressage.

Le passage par le bus système augmente le délai

2 MIPS en concurrence pour lire les instructions

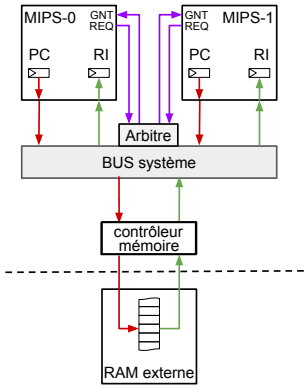


* *Arbitre = BCU (Bus Control Unit)*

- Dans un système réel, il peut y avoir plusieurs MIPS utilisant le même espace d'adressage et par conséquent la même mémoire.
- Les accès des MIPS vers la mémoire passent par un BUS système qui les relie à la mémoire.
- Le BUS est une ressource partagée qui ne peut être utilisée que pour une seule transaction (lecture ou écriture) à la fois.
- Un arbitre qui reçoit les demandes des MIPS (REQuest) pour utiliser le BUS et il l'attribue (GraNT) en garantissant l'absence de famine.
- La présence de plusieurs MIPS et la nécessité de demander à un arbitre pour être autoriser à communiquer augmente encore le temps de cycle ou le nombre de cycle de gel des MIPS.

Si la mémoire est à l'extérieur, c'est pire !

La mémoire est placée dans des boîtiers externes



- Dans le cas général, la mémoire est externe, il faut passer par un contrôleur de mémoire
- Le temps nécessaire pour lire la mémoire externe peut être de plusieurs dizaines de cycles du MIPS
- Avec cette architecture, les MIPS vont exécuter au mieux, une instruction tous les 100 cycles !
- Résumé des étapes
  - Faire une requête à l'arbitre (REQuest)
  - Attendre l'autorisation (GraNT)
  - Envoyer l'adresse,
  - Traverser le contrôleur de mémoire
  - Aller chercher la donnée en RAM externe
  - Acheminer la donnée en réponse

➡ Ça ne peut pas fonctionner ainsi !

Idée

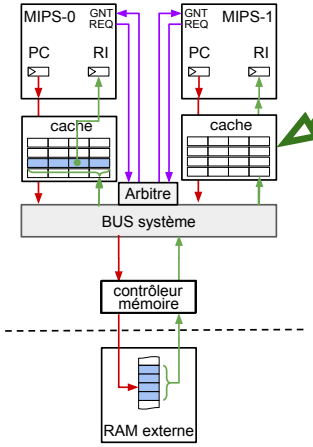
Les accès mémoire d'un programme qui s'exécute ont deux propriétés :

1. **Propriété de localité temporelle**
   Lorsqu'une instruction ou une donnée est lue,
   la probabilité qu'elle le soit à nouveau est très élevée.
2. **Propriété de localité spatiale**
   Lorsqu'une instruction ou une donnée est lue,
   la probabilité que ces voisines le soit aussi est très élevée.

➡ L'idée consiste à profiter de ces deux propriétés pour réduire le temps d'accès aux instructions et aux données.

L'ajout de mémoire cache résout le problème

Les MIPS ne voient plus la mémoire

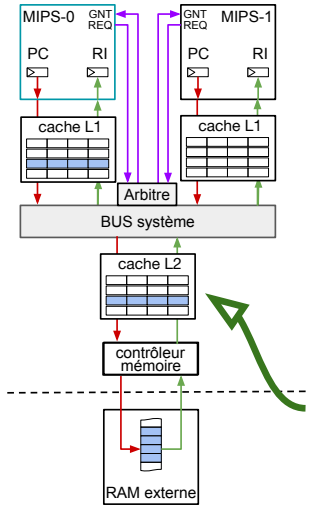


On ajoute une petite mémoire directement reliée au cœur pour stocker une copie des données et des instructions lues en mémoire.

- Lorsque le cœur lit une données ou une instruction pour la première fois. Il la demande à son cache.
- Puisque le cache est vide au départ, le cache va chercher l'instruction ou la donnée dans la mémoire
  - Le cache la stocke pour un appel futur (→ pour profiter de la localité temporelle)
  - Le cache en profite pour lire les données ou les instructions voisines (→ pour profiter de la localité spatiale)
- Si le cache possède la donnée ou l'instruction demandée, il répond dans le cycle

➡ Quand tout se passe bien, on peut démarrer une instruction par cycle.

Il peut y avoir plusieurs niveaux de caches

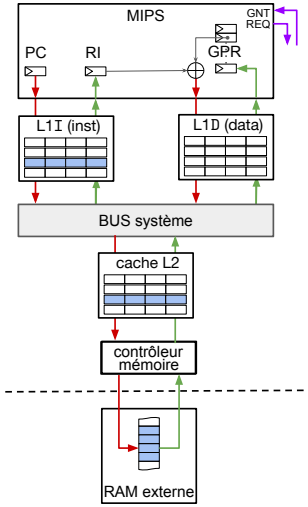


La mémoire cache connectée directement au MIPS doit être très rapide (lue en un seul cycle), en conséquence, elle ne peut pas être grande.

- La mémoire cache ne contient qu'un extrait des instructions et des données.
- Quand un cache ne dispose pas des données ou des instructions demandées, il doit aller les chercher dans la mémoire externe, c'est lent.

L'idée c'est d'ajouter un second niveau (L2) de cache contenant aussi un extrait de la mémoire. Ce second niveau est plus lent que le premier (L1) niveau, entre autre parce qu'il est derrière le bus, mais il est plus rapide que la mémoire externe.

Le MIPS dispose de deux caches séparés.



Nous avons regardé l'accès aux instructions, mais le MIPS lit aussi des données.

- Le MIPS a une architecture en pipeline, c'est-à-dire qu'il lit une nouvelle instruction pendant qu'il termine l'exécution de l'instruction précédente (la durée réelle d'exécution d'1 instruction est de 5 cycles).
- La conséquence de ce fonctionnement en pipeline est que le MIPS peut faire deux accès mémoire au même cycle.
  - La lecture d'une nouvelle instruction
  - La lecture ou l'écriture d'une donnée d'une instruction en cours d'exécution (lue précédemment)
- Nous allons placer un cache par accès
  - `L1I` Cache d'instructions
  - `L1D` Cache de données

Répartition des types d'instructions

Un programme type exécute

- 20% loads (lw, lh, lb)
- 10% stores (sw, sh, sb)
- 40% instructions arithmétiques et logiques (add, or, etc.)
- 30% branchs (bne, jal, j, etc.)

Comportement

- Les lectures sont toujours bloquantes
  → Si la donnée ou l'instruction n'est pas disponible il y a un gel du cœur de processeur
- Les écritures ne sont pas bloquantes
  → Le cœur envoie une donnée vers la mémoire et il continue l'exécution du programme. Les écritures sont postées

Question

Lorsque le cœur de processeur exécute 100 instructions.
➢ Combien de lectures et d'écritures fait-il ?

Taux de Miss

Lorsque le cœur demande une instruction ou une donnée,
si le cache correspondant possède l'instruction ou la donnée demandée,
c'est un succès, sinon c'est un échec, c'est un MISS de cache.

$$\text{Le Taux de MISS de cache} = \frac{\text{Nombre de MISS}}{\text{Nombre total d'accès-mémoire}}$$

Le taux de MISS de cache dépend

- de la taille du cache — il diminue si la taille du cache augmente
- de l'usage de la mémoire — il diminue s'il y a beaucoup de boucles et de tableaux

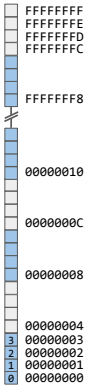
Taux "normaux"

- Pour le cache d'instruction le taux de MISS de cache < 2%
- Pour le cache de données le taux de MISS de cache < 5%

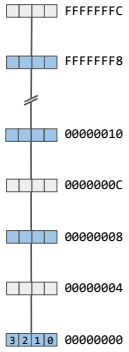
Octets et Mots

Le MIPS32 à un espace d'adressage (EA) de 4GiB : adresses sur 32 bits → 2^{32} adresses

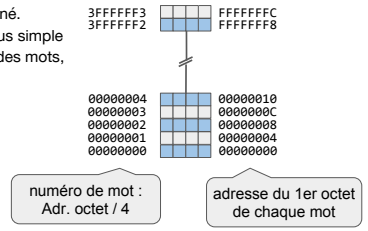
Représentation par octets
chaque adresse désigne un octet (1 Byte de 8 bits)



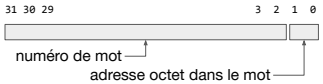
Représentation par mots
Un mot est un petit segment d'adresses de 4 octets aligné.
Cette représentation est plus simple car MIPS lit naturellement des mots,



Représentation compacte par mots



adresse 32 bits

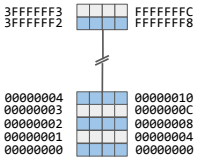


Le numéro de mot est l'adresse octet divisée par la taille d'un mot

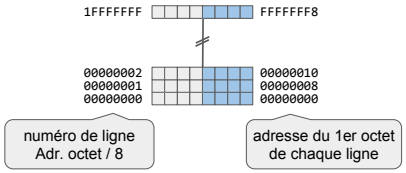
Ligne de cache

- Une ligne de cache est l'unité d'échange en lecture entre la mémoire et un cache, c'est à dire que lorsqu'un cache lit la mémoire, il lit une ligne de cache entière
  (à l'instar du MIPS qui lit toujours de mots même quand on exécute `lb` ou `lh`)
- Une ligne de cache est un segment de 2^n octets, n est compris en 3 et 7.
- La taille d'une ligne de cache n'est pas un standard, elle peut être : 8, 16, 32, 64 ou 128

rappel de la
Représentation de l'EA par mots

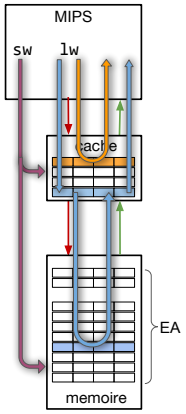


Représentation de l'EA par lignes
Dans cet exemple, on suppose
qu'une ligne fait 8 octets, c'est-à-dire 2 mots



Le numéro de ligne est l'adresse octet divisée par la taille d'une ligne, donc pour savoir dans quelle ligne se trouve un octet, il suffit de diviser son adresse par la taille d'une ligne.

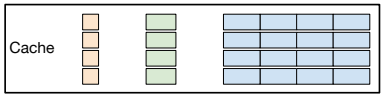
Principe de fonctionnement



- Lecture d'une donnée déjà présente dans le cache
  1. Le MIPS exécute une instruction `lw`
  2. Le cache détermine le numéro de ligne où se trouve la donnée à partir de son adresse
  3. La ligne est déjà dans le cache
  4. La donnée est lue dans le cache et rendue au MIPS
- Lecture d'une donnée absente du cache
  3. La ligne est absente du cache
  4. Le MIPS est gelé
  5. La ligne est demandée à la mémoire
  6. La ligne est rangée dans le cache
  7. La donnée est lue dans le cache et rendue au MIPS
- Écriture d'une donnée (politique ***write through***)
  1. Le MIPS exécute une instruction `sw`
  2. Le cache détermine le numéro de ligne où se trouve la donnée à partir de son adresse
  3. Si la ligne est déjà dans le cache, elle est mise à jour
  4. La donnée est envoyée vers la mémoire

Vocabulaire

- Case de cache 
  - Un cache contient des copies de lignes de cache venant de la mémoire.
  - Ces lignes de cache sont copiées dans des cases de cache.
  - Une ligne de cache est un contenu, alors qu'une case de cache est un contenant
  - **Les lignes de cache sont rangées dans des cases de cache.**
- Répertoire 
  - Le cache doit savoir quelles sont les lignes de cache présentes dans ses cases.
  - Le cache range dans un répertoire les numéros de ligne de cache qu'il possède.
  - Il y a autant d'entrée dans le répertoire qu'il y a de cases dans le cache.
- Bit de validité 
  - Au début les cases de cache sont toutes vides.
  - Elles se remplissent avec des lignes de cache au fur et à mesure des lectures.
  - Il y a un bit de validité par case de cache indiquant si la case contient une ligne.



Types de cache

Lorsque le cache veut ranger une ligne de cache, il doit faire le choix d'une case. C'est le choix de la victime car dans le cas général toutes les cases contiennent déjà des lignes. Il existe plusieurs types de cache.

Cache *full associative* (cache totalement associatif)
Le numéro de la case victime est quelconque.
Une ligne peut être rangée dans n'importe quelle case.
Le choix se fonde sur un algorithme de type LRU (*Last Recently Used*)

Cache *direct mapped* (cache à correspondance directe)
Le numéro de la case victime est déterminé à partir du numéro de la ligne.
En fait, il n'y a pas de choix.

Cache *N-associative* (cache partiellement associatif)
C'est un mélange des types précédents.
Le cache détermine un sous ensemble de N cases à partir du numéro de la ligne.
Puis, le cache choisit la case victime dans ce sous-ensemble par algorithme type LRU

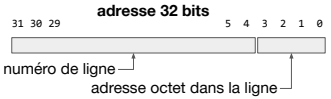
Nous allons présenter le cache à correspondance directe, c'est le plus simple,
mais nous allons voir qu'il n'est pas optimal parce qu'une victime peut contenir une ligne qui vient juste d'être chargée.

Cache à correspondance directe

Rappel : Si les lignes de cache font 2^n octets, l'adresse d'un octet quelconque dans l'espace d'adressage se décompose en deux parties :

- les n bits de poids faible désigne la place de l'octet dans la ligne
- les (32 - n) bits de poids fort désigne le numéro de la ligne

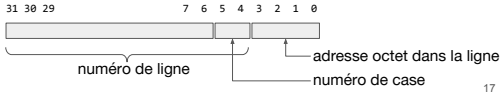
Par exemple,
Pour des lignes de 16 octets



Dans un cache à correspondance directe,
le numéro de la case victime est déterminé à partir du numéro de la ligne.

- Pour déterminer de manière simple un numéro de case à partir d'un numéro ligne, on choisit d'utiliser les X bits de poids faible du numéro de ligne.
- X dépend du nombre de cases et ce nombre est nécessairement une puissance de 2. Si `X==0` alors le cache n'a qu'une case, si `X==1` → 2 cases, si `X==1` → 4 cases, etc.

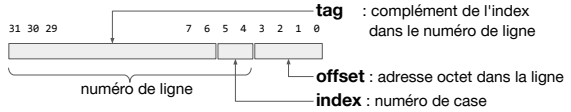
Par exemple, pour X = 2



Calcul du nombre de bits : tag, index et offset

Vocabulaire

- L'adresse octet dans une ligne s'appelle offset
- Le numéro de la case utilisé par la ligne s'appelle index
- Le complément de l'index dans le numéro de ligne s'appelle l'étiquette (tag)

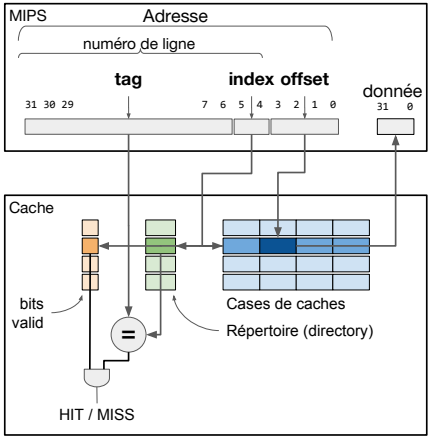


Soit un cache de C octets et des lignes L octets (C et L sont toujours des puissances de 2)

- Le nombre de cases de cache est C / L
- Le nombre de bits de l'offset est nbits-offset = $\log_2(L)$
- Le nombre de bits d'index est nbits-index = $\log_2(C/L)$
- Le nombre de bits d'étiquette est nbits-tag = 32 - nbits-offset - nbits-index

Par exemple un cache de 8kB avec des lignes de cache de 8B
nombre de cases de cache = 8*1024 / 8 = 1024 cases
nbits-offset = $\log_2(8)$ = 3 nbits-index = $\log_2(1024)$ = 10 nbits-tabg = 32 - 2 - 10 = 20

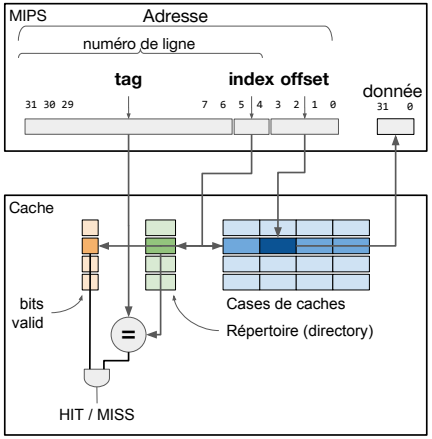
Schéma de principe du cache



Il y a trois tableaux dans le cache :

1. Les cases de cache (en bleu).
   Ces cases contiennent les lignes de cache.
2. Le répertoire (en vert)
   Ce répertoire contient autant d'entrée que de cases dans le cache (il y a une entrée par case). Chaque entrée contient le numéro de la ligne contenu dans la case du cache correspondante. Dans le cas d'un cache à correspondance directe, on peut n'enregistrer que le tag du numéro de la ligne), c'est une optimisation.
3. Les bits de validités (en orange)
   Ce répertoire contient autant d'entrée que de cases dans le cache (il y a une entrée par case). Lorsqu'une entrée est à 1, c'est que la case correspondante contient une ligne, sinon elle n'en contient pas.

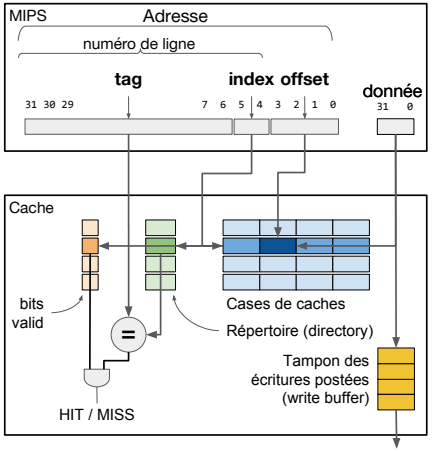
Principe de fonctionnement en lecture



Le numéro de la case de cache est directement désignée par l'index.

- Le cache lit le répertoire et le bit de validité en utilisant l'index
- Si le bit de validité est à 1 et que le répertoire contient le tag de ligne recherchée (celui présent dans l'adresse) alors la case contient la bonne ligne c'est un HIT sinon c'est un MISS
- En cas de MISS, le cache gèle le MIPS, va chercher la ligne en mémoire et la range dans la case désignée par l'index, il met à jour le bit de validité et le répertoire.
- Enfin, la donnée est lue dans la case à la position de l'offset et rendue au MIPS.

Principe de fonctionnement en écriture



Le numéro de la case de cache est directement désignée par l'index.

- Le cache lit le répertoire et le bit de validité en utilisant l'index
- Si le bit de validité est à 1 et que le répertoire contient le tag de ligne recherchée (celui présent dans l'adresse) alors la case contient la bonne ligne, c'est un HIT sinon c'est un MISS
- En cas de HIT le cache met la ligne à jour.
- Dans tous les cas la donnée est mise dans le tampon des écritures pour être écrite en mémoire plus tard.
- Noter que ce tampon est aussi lu lors des lectures pour relire une donnée juste écrite

Inconvénient des caches à correspondance directe

Dès lors que le numéro de la case est déterminé par le numéro de ligne et qu'il est unique, cela peut (va) créer des collisions.

Par exemple dans le cache juste précédent : 64 octets en 4 case de 16 octets

nbits-offset = $\log_2(16) = 4$
nbits-index = $\log_2(64/16) = 2$
nbits-tag = 32 - 4 - 2 = 26

Soit les tableaux globaux

```
int X [16];      // X est a l'adresse 0x10000000
int Y [5];       // Y est à l'adresse 0x10000040
```

dans une fonction

```
Z = X[2] + Y[3];
```

`X[2]` est à l'adresse `0x10000008` ⇒ num ligne `0x1000000` ⇒ num case `0`
`Y[3]` est à l'adresse `0x1000004C` ⇒ num ligne `0x1000004` ⇒ num case `0`

Le cache charge la ligne `0x1000000`, la place dans case 0 et rend X[2] au MIPS puis le cache charge la ligne `0x1000004`, la place aussi dans la case 0 et écrase la ligne de X

En TME

Vous allez

1. étudier la manière dont les caches se remplissent
   - en fonction de la position des instructions et des données en mémoire
   - et en fonction de la taille des caches.
2. Calculer le taux de miss du cache instruction lors de l'exécution d'une boucle d'instructions
3. Calculer le taux de miss du cache de données
   - de l'usage de donnée par les instructions
   - de leur position en mémoire

La semaine prochaine

Nous continuerons sur les caches

- en mesurant leur impact sur les performances
- en introduisant le problème de la cohérence des données copiées dans plusieurs caches