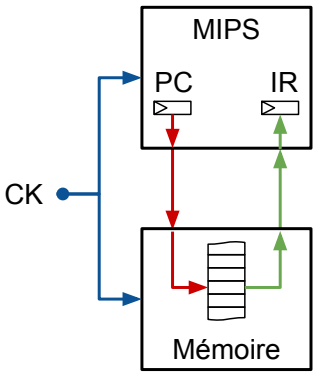
Cache L1 - architecture

LU3IN031 Architecture des ordinateurs - 2
Matériel et Logiciel
B3

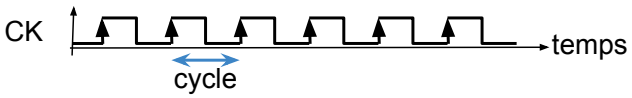
franck.wajsburt@lip6.fr
V3

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).



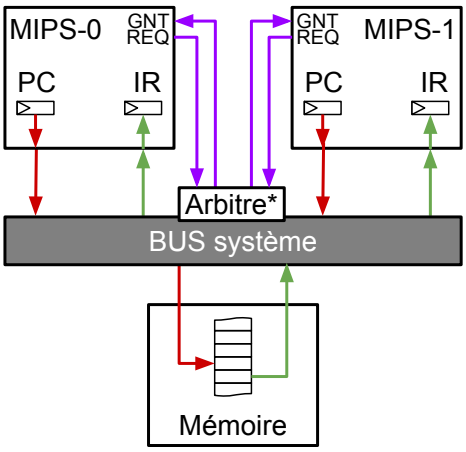
- Un cycle d'horloge (une période entre 2 fronts actifs) est le temps d'un calcul élémentaire dans le MIPS.
- Le MIPS demande une nouvelle instruction à chaque cycle (L'adresse est dans **PC** et l'instruction va s'inscrire dans **IR** *Instruction Register*).
- Il faut donc que la mémoire réponde en 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

En réalité
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

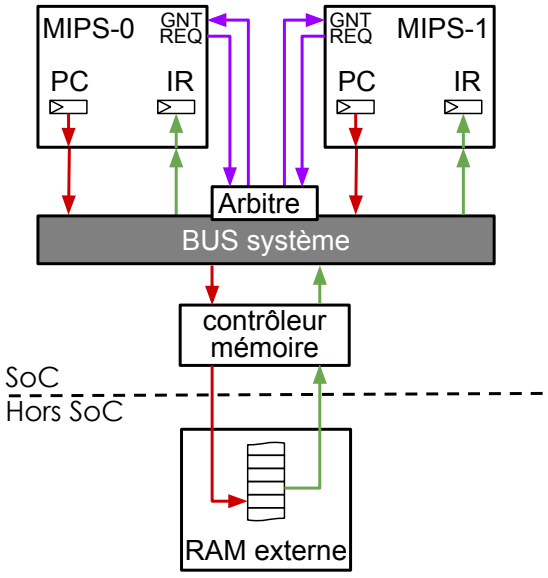


* *L'Arbitre est aussi nommé BCU (Bus Control Unit)*

- Dans un système réel, il peut y avoir plusieurs MIPS utilisant le même espace d'adressage et par conséquent le même composant 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 à la fois (une lecture ou une écriture)
- Un arbitre 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 autorisé à communiquer augmente encore le temps de cycle ou le nombre de cycles de gel des MIPS (arrêt).

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

La mémoire est placée dans des boîtiers externes au SoC



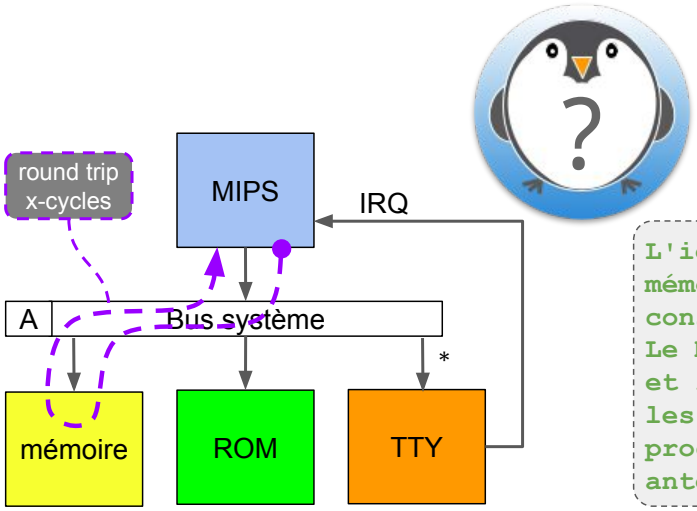
- Dans le cas général, la mémoire est externe, il faut passer par un **contrôleur de mémoire** dont le rôle principal est de gérer l'accès aux barrettes de RAM (protocole complexe).
- Le temps nécessaire pour lire la mémoire externe peut être de plusieurs dizaines de cycles du MIPS
- Avec cette architecture, 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 !

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, ça ne va pas !

S'il faut attendre 10 cycles pour lire une instruction ou une donnée lue et qu'il y a 20% d'instruction de lecture de donnée, alors 100 instructions = 1200 cycles !

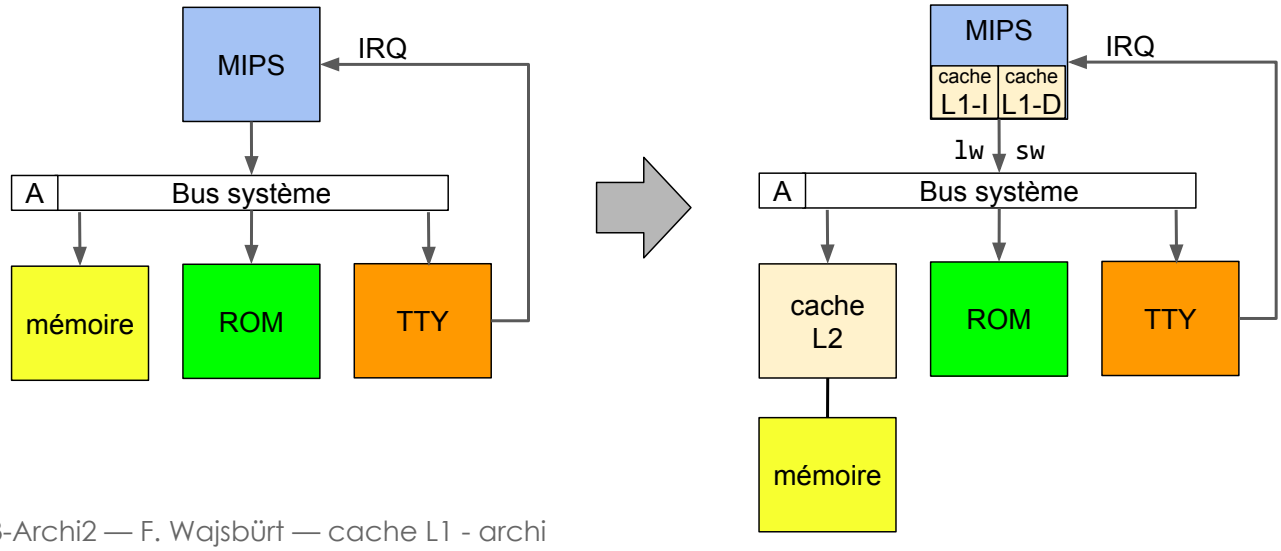


L'idée c'est de ne pas utiliser la mémoire pour le travail et de la considérer comme une zone de stockage. Le MIPS doit aller chercher les données et les instructions dont il a besoin et les écrire dans une petite mémoire très proche accessible en 1 cycle, dans une antémémoire (anté=avant)ou memory cache

* sur ce schéma, on ne fait pas apparaître les données

Principe de la solution

La solution est donc d'utiliser des caches qui sont des petites mémoires qui vont contenir les dernières instructions et les dernières données utilisées.



mémoire cache
Principe de fonctionnement

Idée : exploiter les localités

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

1. **La propriété de localité temporelle**

   Lorsqu'une instruction ou une donnée est lue,

   la probabilité qu'elle le soit à nouveau rapidement est élevée.

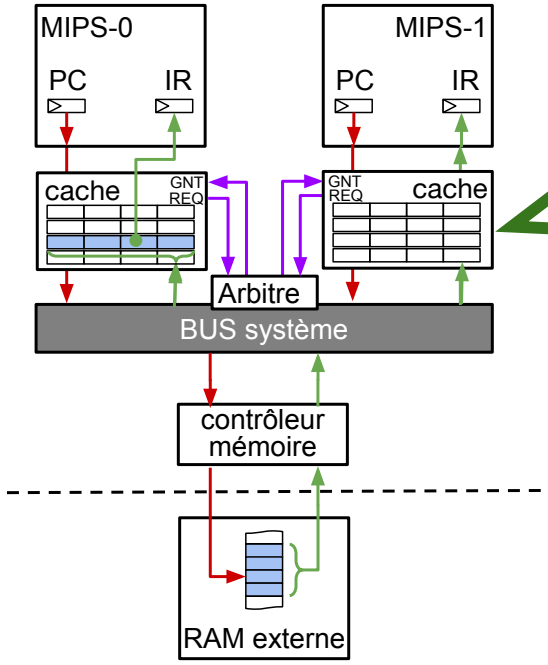
2. **La propriété de localité spatiale**

   Lorsqu'une instruction ou une donnée est lue,

   la probabilité que ces voisines en mémoire le soit aussi est 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



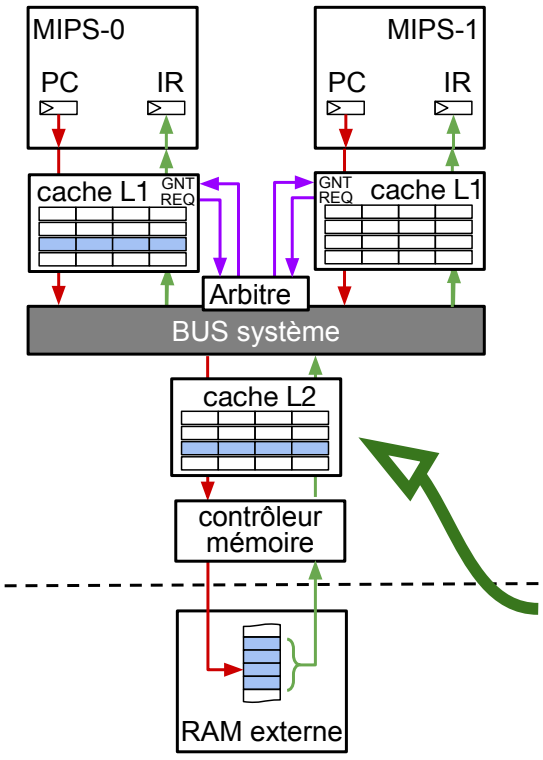
Les MIPS ne "voient" plus la mémoire

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

- Lorsque le cœur lit une donnée ou une instruction pour la première fois. Il la demande à son cache.
- Puisque le cache est vide au départ, il 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 le cache contient déjà toutes les instructions, on peut démarrer une instruction par cycle !

Il peut y avoir plusieurs niveaux de caches



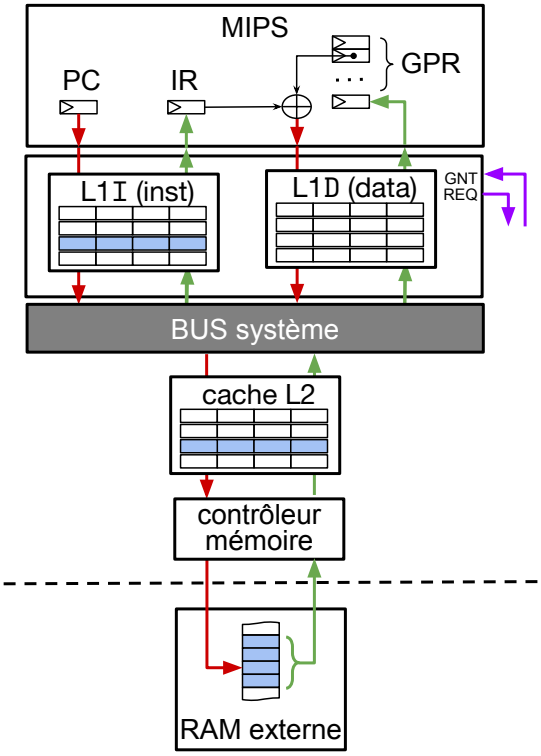
La mémoire cache connectée directement au MIPS doit être très rapide (lue en un seul cycle), mais elle ne peut pas être très grande pour des raisons technologiques (procédés de fabrication)

En conséquence :

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

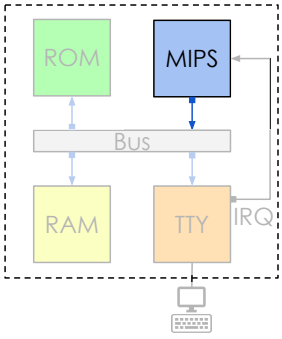
L'idée c'est d'ajouter un second niveau (L2) de cache contenant lui aussi un extrait de la mémoire. Ce second niveau est plus lent que le (L1) premier niveau, entre autres parce qu'il est derrière le bus, mais il est plus grand et, bien sûr, il est toujours beaucoup plus rapide que la mémoire externe.

Le MIPS dispose de deux caches L1 séparés.



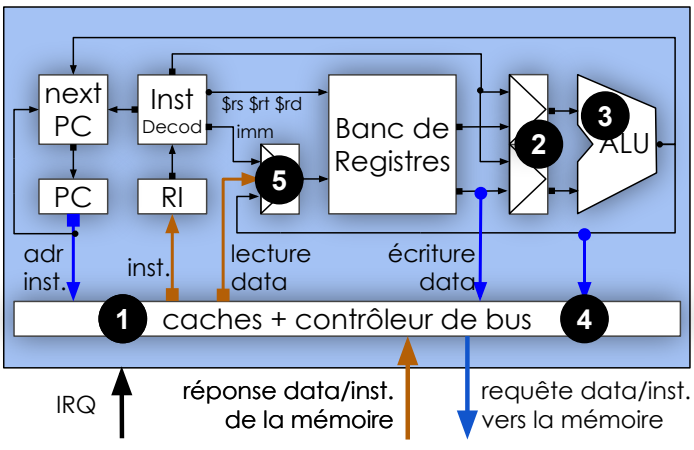
Nous avons regardé l'accès aux instructions,
mais le MIPS lit aussi des données.

- 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 mémoire
  on parle de caches L1 séparés
  - `L1 I` Cache d'instructions
  - `L1 D` Cache de données
- Par contre, le cache L2 est unifié,
  c.-à-d. que données et instructions sont mélangées



Le cœur MIPS32

Le MIPS32 est un processeur RISC qui exécute environ 1 instruction par cycle.
Les calculs sont faits dans les registres internes. La mémoire est juste lue/écrite via un contrôleur de bus qui arbitre entre les accès data et les accès instructions.



Le MIPS exécute les instructions `(add, lw, beq, ...)` en plusieurs étapes

1. Lecture de l'instruction à l'adresse du PC et rangement dans le registre instruction (RI)
2. Décodage de l'instruction et calcul des opérandes à partir de RI et des registres
3. exécution de l'opération dans l'ALU
4. Si c'est une instruction load/store (lw/sw) Lecture ou écriture mémoire
5. Écriture du résultat dans le banc de registre

Le MIPS peut exécuter l'étape 1 sur une instruction
en même temps que l'étape 2 sur la précédente
en même temps que l'étape 3 sur celle encore avant
etc.

→ En apparence, le MIPS exécute 1 instruction / cycle

Répartition des types d'instructions

Un programme type exécute
- 40% Arithmétiques et logiques (add, or, etc.)
- 30% Branchements (bne, jal, j, etc.)
- 20% Loads (lw, lh, lb)
- 10% Stores (sw, sh, sb)

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 dites "postées"

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

Répartition des types d'instructions

Un programme type exécute
- 40% Arithmétiques et logiques (add, or, etc.)
- 30% Branchements (bne, jal, j, etc.)
- 20% Loads (lw, lh, lb)
- 10% Stores (sw, sh, sb)

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 dites "postées"

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

  réponse : 100 lectures d'instructions
  + 20 lectures de donnée
  + 10 écritures de donnée

Taux de Miss

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

Quels sont les Taux "normaux" ?

- Pour le cache d'instruction le taux de MISS de cache < 2% (2 MISS et 98 HIT)
- Pour le cache de données le taux de MISS de cache < 5% (5 MISS et 95 HIT)

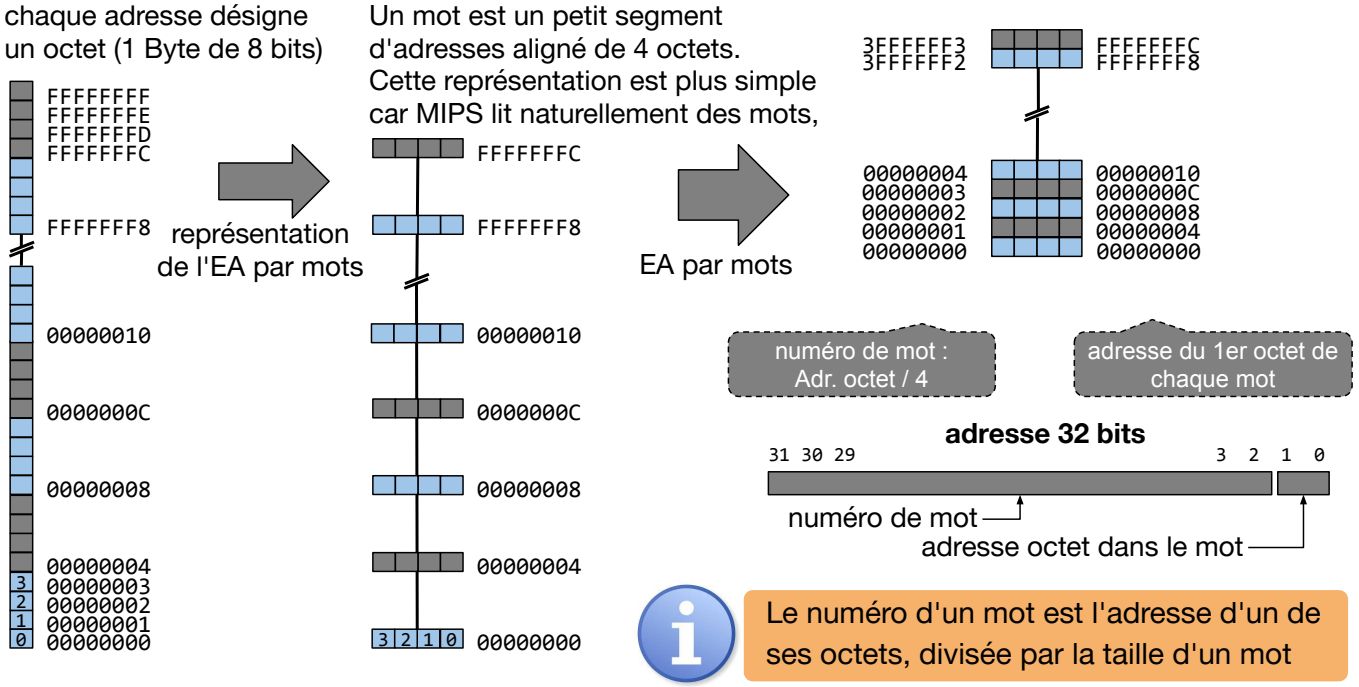
Octets et Mots

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 aligné de 4 octets.
Cette représentation est plus simple car MIPS lit naturellement des mots,

Représentation compacte par mots

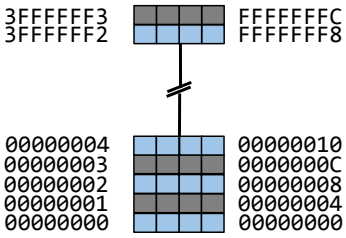


Le numéro d'un mot est l'adresse d'un de ses octets, divisée par la taille d'un mot

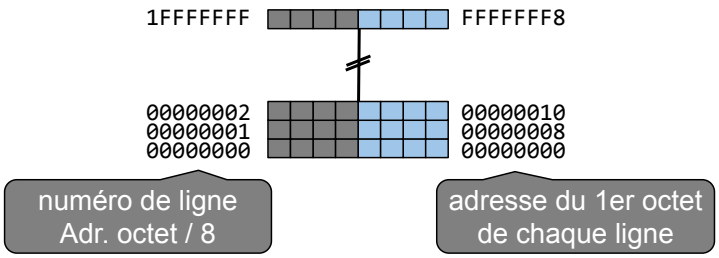
Ligne de cache

- 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, alors il lit une ligne de cache entière (à l'instar du MIPS qui lit toujours des 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 peut donc être de 8, 16, 32, 64 ou 128 octets.
- La taille d'une ligne de cache dépend de l'architecture du cache.

rappel de la
Représentation de l'EA par mots

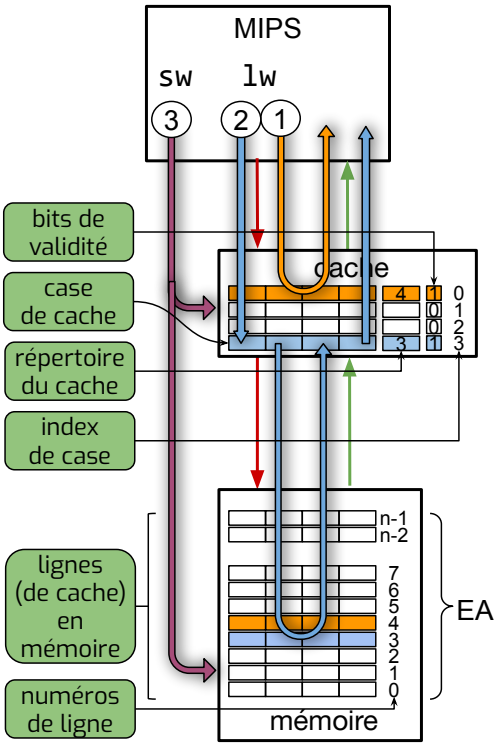


Représentation de l'EA par lignes
Dans cet exemple, on suppose
qu'une ligne fait 8 octets, c'est-à-dire 2 mots



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 de la ligne.

Principe de fonctionnement



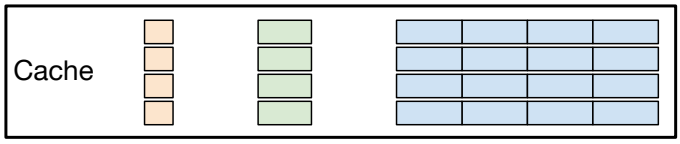
① Lecture d'une donnée déjà présente dans le cache
- Le MIPS exécute une instruction `lw`
- Le cache détermine le numéro de ligne où se trouve la donnée à partir de son adresse
- Et la ligne est déjà dans le cache
- La donnée est lue dans le cache et rendue au MIPS

② Lecture d'une donnée absente du cache
- Le MIPS exécute une instruction `lw`
- Le cache détermine le numéro de ligne
- Mais la ligne est absente du cache
- Le MIPS est gelé
- La ligne est demandée à la mémoire
- La ligne est rangée dans le cache
- La donnée est lue dans le cache et rendue au MIPS

③ Écriture d'une donnée : politique ***write through*** *cf.* slide 27
- Le MIPS exécute une instruction `sw`
- Le cache détermine le numéro de ligne où se trouve la donnée à partir de son adresse
- Si la ligne est déjà dans le cache, elle est mise à jour
- 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.
Il choisit une 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 de la victime 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** : à chaque numéro de ligne correspond un seul numéro de case
- Cache ***N-way-set-associative*** (cache partiellement associatif)
  - C'est un mélange des deux 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 un algorithme de type LRU

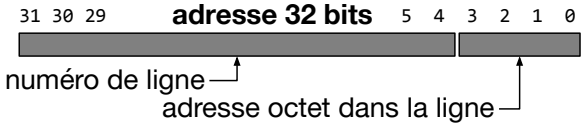
Nous allons voir le cache à correspondance directe, c'est le plus simple, mais nous allons voir qu'il n'est pas optimal parce qu'une case victime peut contenir une ligne qui vient juste d'être chargée.

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

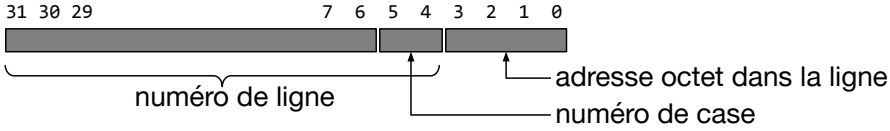
Par exemple,
Pour des lignes de 16 octets



Dans un cache à **correspondance directe**,
le numéro de **la case victime est déterminée 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 == 2`) → 4 cases, etc.

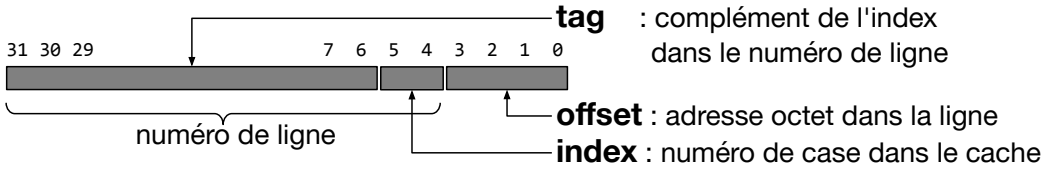
Par exemple, pour (`X == 2`)



Calcul du nombre de bits : tag, index et offset

Vocabulaire

- L'adresse d'un octet dans une ligne s'appelle l'**offset**
- Le numéro de la case du cache utilisée par la ligne s'appelle l'**index**
- Le complément de l'index dans le numéro de ligne s'appelle le **tag** (étiquette)



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 = log2(L)`
- Le nombre de bits d'index est `nbits_index = log2(C/L)`
- Le nombre de bits du tag est `nbits_tag = 32 - nbits_offset - nbits_index`

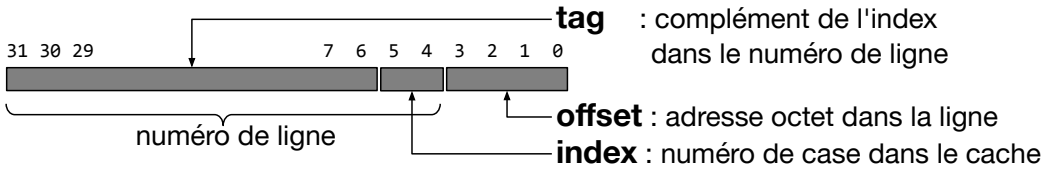
Par exemple un cache de `8kiB` avec des lignes de cache de `8B`

nombre de cases du cache =
`nbits_offset =` `; nbits_index =` `; nbits_tag =`

Calcul du nombre de bits : tag, index et offset

Vocabulaire

- L'adresse d'un octet dans une ligne s'appelle l'**offset**
- Le numéro de la case du cache utilisée par la ligne s'appelle l'**index**
- Le complément de l'index dans le numéro de ligne s'appelle le **tag** (étiquette)



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

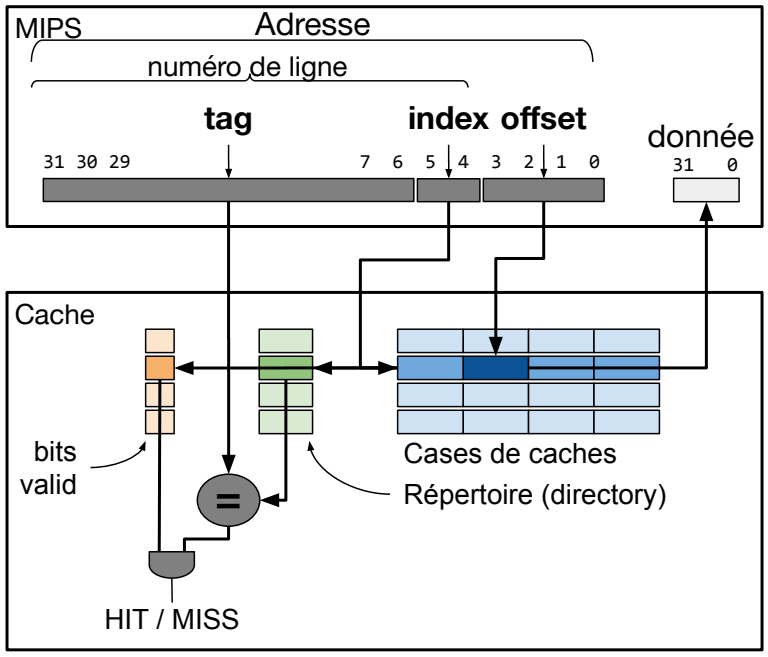
- Le nombre de cases de cache est `C / L`
- Le nombre de bits de l'offset est `nbits_offset = log2(L)`
- Le nombre de bits d'index est `nbits_index = log2(C/L)`
- Le nombre de bits du tag est `nbits_tag = 32 - nbits_offset - nbits_index`

Par exemple un cache de `8kiB` avec des lignes de cache de `8B`

nombre de cases du cache = `8*1024 / 8 = 1024 cases`

`nbits_offset = log2(8) = 3 ; nbits_index = log2(1024) = 10 ; nbits_tag = 32-2-10 = 20`

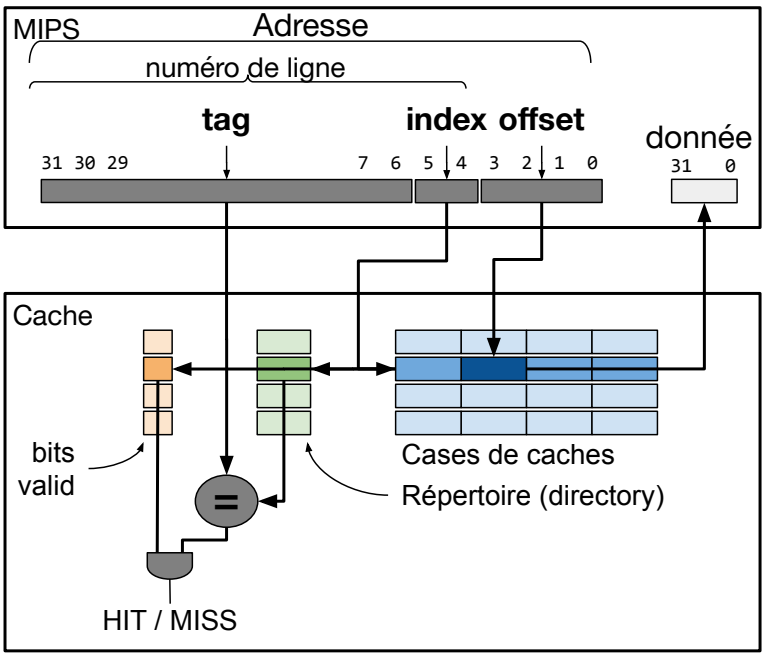
Schéma de principe du cache



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'entrées 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'entrées 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.

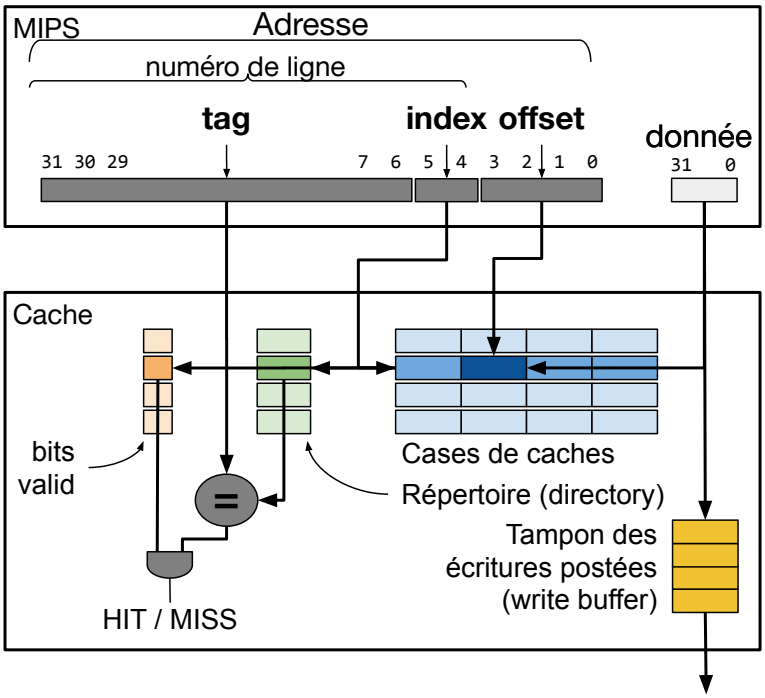
Principe de fonctionnement en lecture



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 la 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, le cache va chercher la ligne en mémoire et il la range dans la case désignée par l'index, il met à jour le bit de validité et le répertoire pour cette case.
- 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 la 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 qui viennent juste écrite mais qui ne sont pas encore dans la mémoire.

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 va créer des "collisions" entre les lignes qui sont en compétition pour une case.

Par exemple dans le cache vu précédemment : 64 octets en 4 case de 16 octets

`nbits_offset = log2(16) = 4`
`nbits_index  = log2(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

Évincement de ligne

- Lors d'un MISS, le cache doit aller chercher la ligne manquante dans la mémoire pour la copier dans une case du cache
- Dans le cas d'un cache à correspondance directe, le numéro de la case choisie est imposé (*on a vu que ce sont les n-bits de poids faible du numéro de ligne*)
- Dans un cache set-associatif ou full-associatif, le cache doit choisir une case avec soin. Il doit éviter de choisir une case contenant une ligne très utilisée.
- Pour savoir si une ligne est très utilisée, il faut mettre des compteurs d'usage dans le répertoire, permettant d'exécuter des algorithmes de type *Last Recently Used* (LRU). Nous n'allons pas détailler ce type d'algorithme.
- La case choisie par l'algorithme LRU est nommée "*case victime*" et la ligne présente dans cette case est dite "*évincée*".
- On a un "*évincement de ligne*" à chaque fois qu'une ligne est retirée du cache.

Comportement du cache pour les écritures

Il existe deux types de comportement pour les écritures

1. Cache Write-Through (WT) *(c'est celui que nous utilisons)*

   Les écritures se font toujours dans la mémoire,
   et si la ligne est déjà présente dans le cache, alors elle est mise à jour.

   Ainsi la mémoire contient toujours la copie la plus à jour des données.

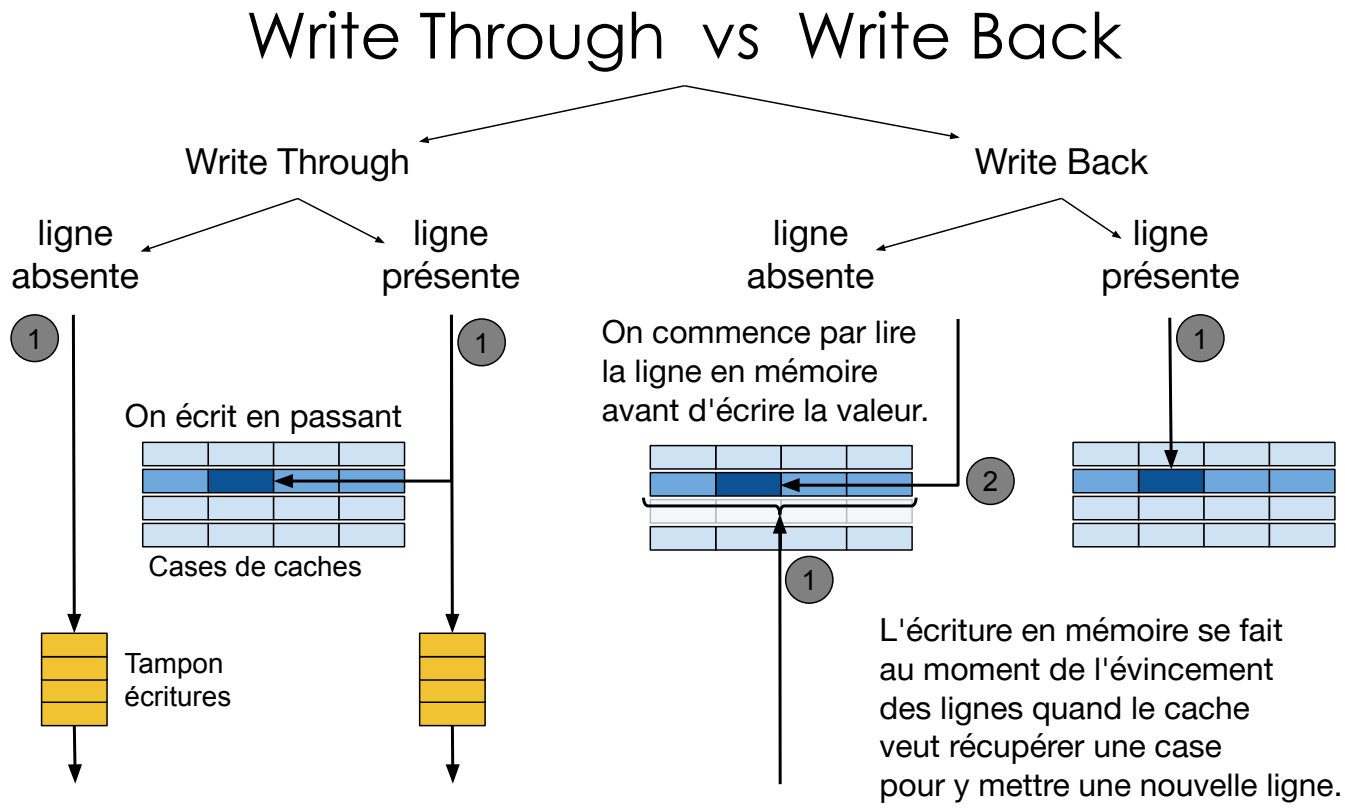
2. Cache Write-Back (WB)

   Lorsqu'on écrit dans une ligne, si elle est absente du cache, elle est chargée depuis la mémoire, puis les écritures se font dans le cache.

   Ainsi, c'est le cache qui contient la copie la plus à jour des données.

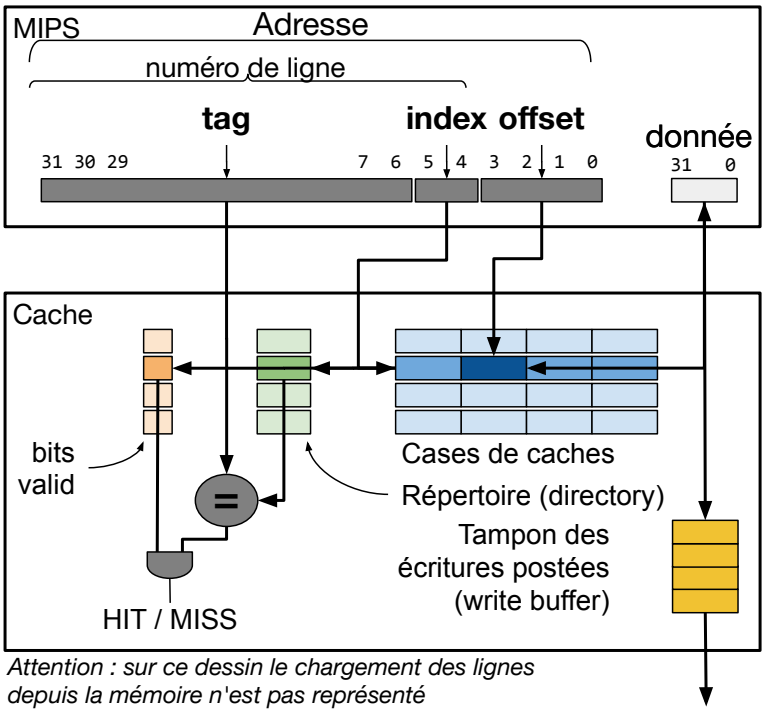
   Les lignes modifiées sont copiées en mémoire uniquement au moment de leur évincement et uniquement si elles ont été modifiées (on dit *dirty*).

Les caches WB réduisent beaucoup le nombre d'écritures sur le bus,
mais ils sont plus complexes, car il y a plus d'états à gérer pour chaque ligne.

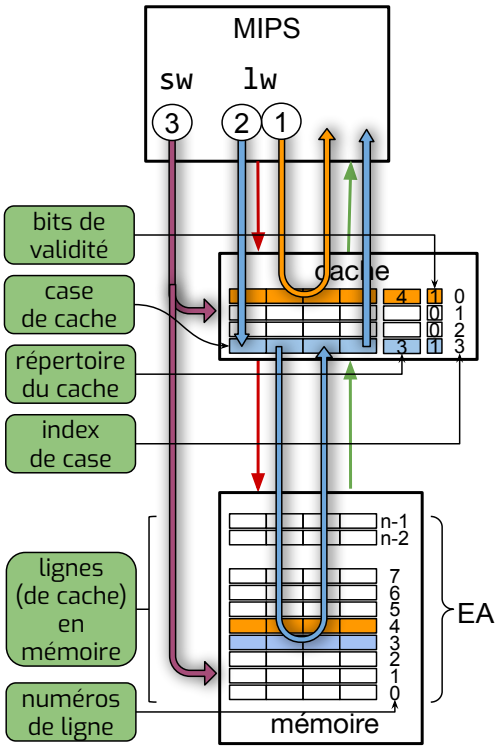


Conclusion

Les 2 schémas que vous devez comprendre



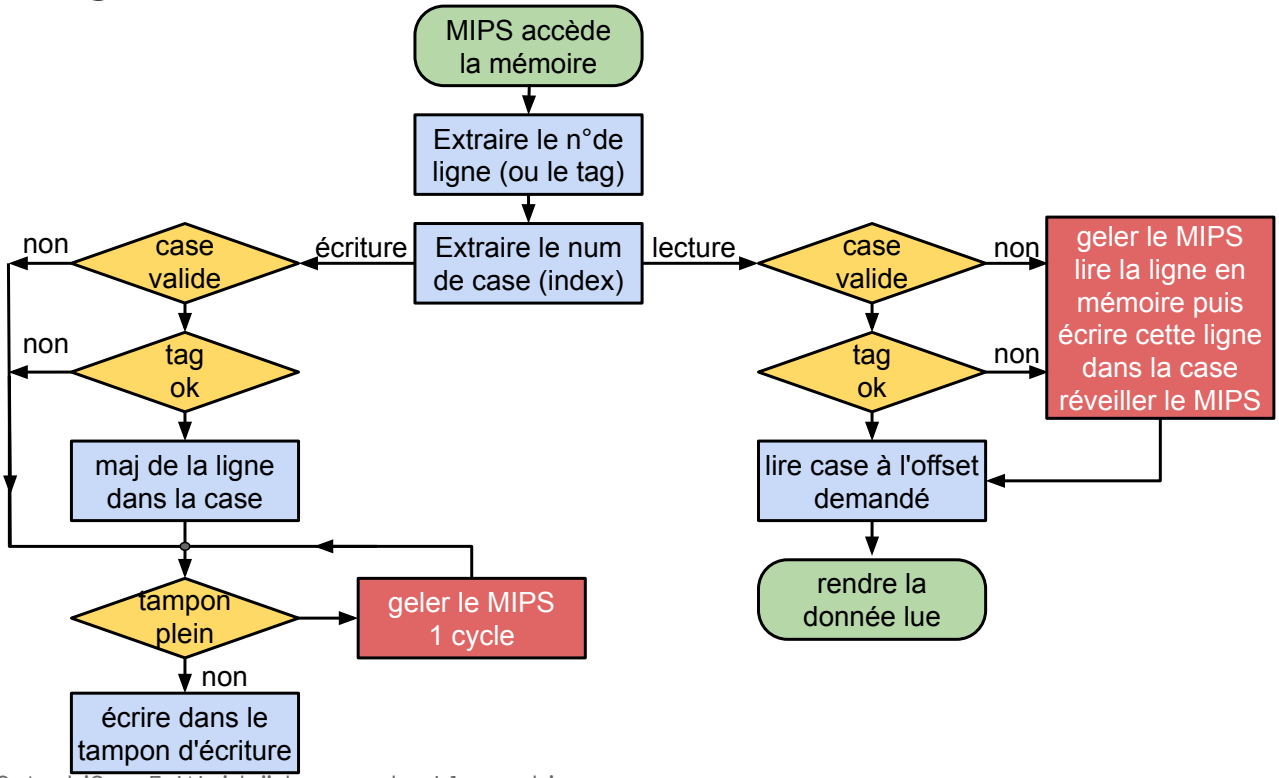
Attention : sur ce dessin le chargement des lignes depuis la mémoire n'est pas représenté



Vocabulaire à retenir

- Ligne de cache
  - Une ligne de cache est un segment d'adresses de 2^n octets dans l'espace d'adressage (8, 16, 32, 64 ou 128 octets, cela dépend du cache).
    C'est la quantité minimale de données ou d'instructions lues par un cache.
- Case de cache
  - Un cache contient des copies de lignes de cache venant de la mémoire.
  - Ces lignes de cache sont rangées dans des cases de cache.
  - Une ligne de cache est un contenu, alors qu'une case de cache est un contenant
- Correspondance directe
  - Le numéro de case (index) utilisé pour ranger une ligne est déterminé en prenant les k bits de poids faible du numéro de ligne, il y a donc 2^k cases dans le cache.
- Répertoire
  - Le cache range dans un répertoire les numéros de ligne de cache qu'il possède.
  - Il y a autant d'entrées dans le répertoire qu'il y a de cases dans le cache.
  - Pour un cache à correspondance directe, on peut sauver seulement le tag de ligne
  - Le tag de ligne est égal au numéro de ligne divisé par l'index (shift right de k bits)
- 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.

Algorithme de comportement du cache



En TME

1. Étude 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. Calcul du taux de miss du cache instruction lors de l'exécution d'une boucle d'instructions
3. Calcul du taux de miss du cache de données
   - de l'usage de donnée par les instructions
   - de leur position en mémoire