Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 & 2

p.06 - Mémoire cache

Classes T-2/I-2 // 2017-2018

Daniel Gachet | HEIA-FR/TIC

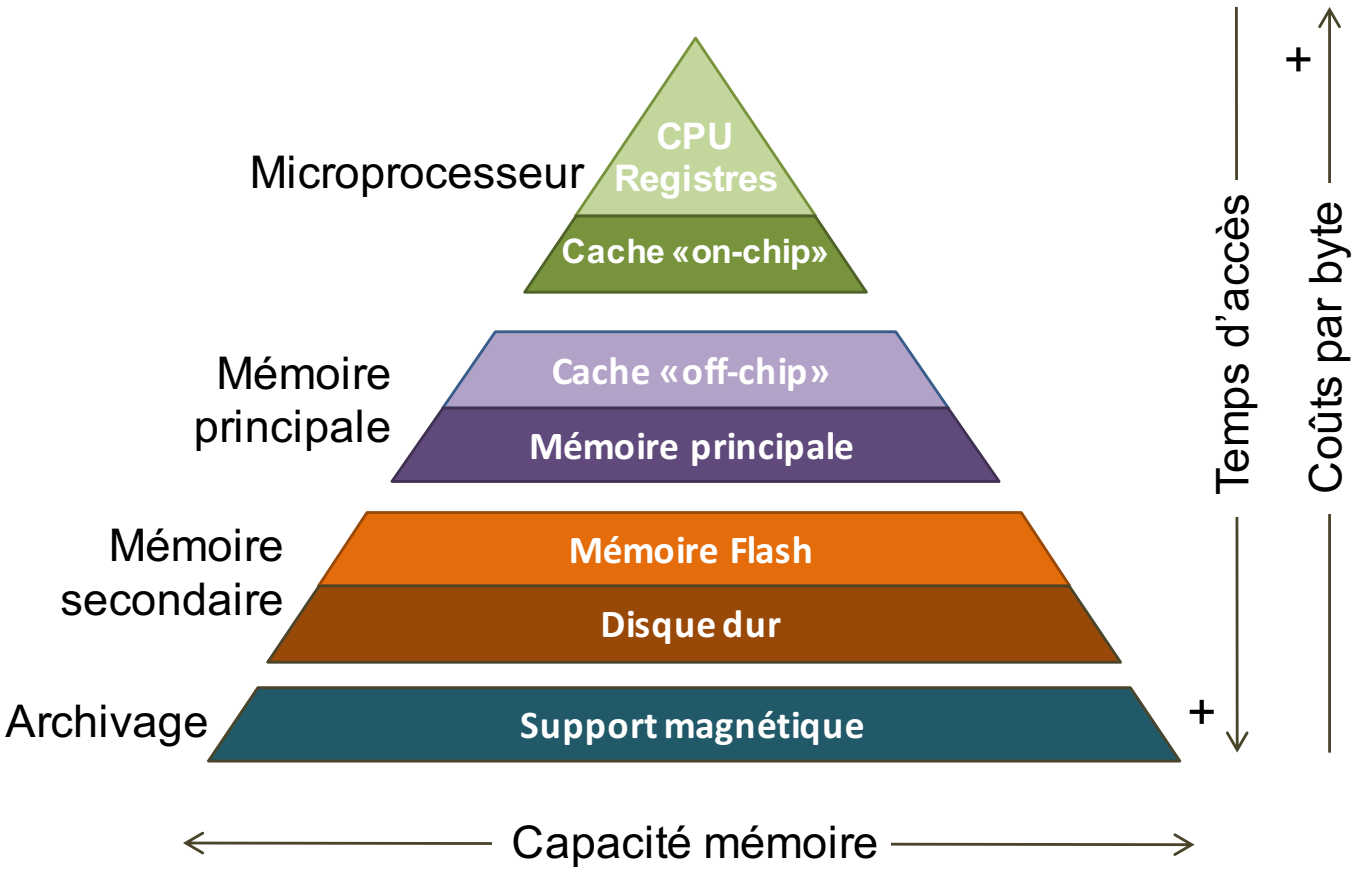
Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz

Contenu

- Hiérarchie de la mémoire
- Organisation de la mémoire cache
- Accès à la mémoire cache
- Mise à jour de la mémoire cache
- Ecriture au travers de la mémoire cache
- Implémentation du TI AM335x / Cortex-A8
- Optimisations

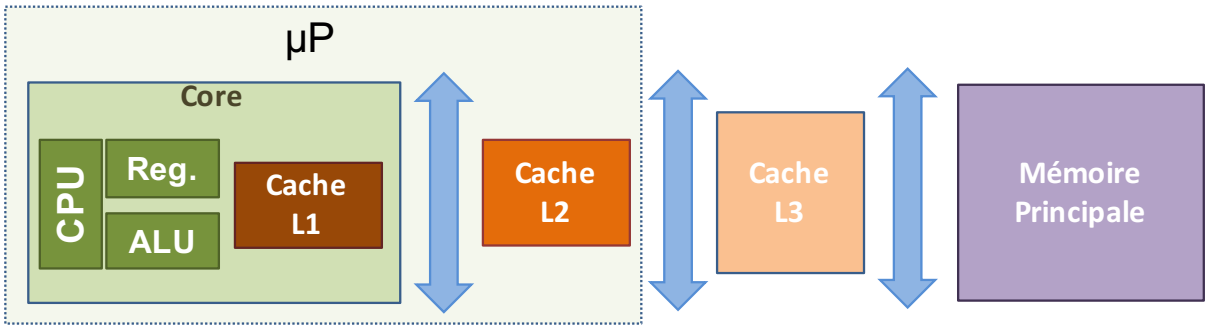
Hiérarchie de la mémoire

- La mémoire d'un processeur est organisée hiérarchiquement, ceci afin de pouvoir satisfaire les aspects performance et les aspects coût



Mémoire cache

- La mémoire cache permet d'augmenter les performances globales d'un µP



- Deux principes sont à l'origine des mémoires caches
  - Le principe de localité spatiale
  - Le principe de localité temporelle

Localité spatiale

- Le principe de localité spatiale suppose que si un emplacement mémoire est référencé à un instant donné, il est alors fort probable qu'un emplacement à proximité soit également référencé peu après

```
uint16_t in[128];
uint16_t out[128];
for (int i=0 ; i<128 ; i++) {
    out[i] = in[i];
}
```

Localité temporelle

- Le principe de localité temporelle suppose que si à un instant donné un emplacement mémoire est référencé, il est alors fort probable que ce même emplacement soit à nouveau référencé peu après

```
uint8_t msg[128];
uint32_t sum=0;
for (int i=0; i<128; i++) {
    sum += msg[i];
}
```

Aspects à résoudre

- Lorsqu'on parle de mémoire cache, 5 aspects principaux doivent être résolus
  1. Comment identifier une ligne de la mémoire principale dans la mémoire cache ?
  2. Où peut-on placer une ligne de la mémoire principale dans la mémoire cache ?
  3. Comment peut-on retrouver une ligne dans la mémoire cache ?
  4. Quelle ligne doit être remplacée dans la mémoire cache ?
  5. Que se passe-t-il lors d'une écriture ?
- Quelles opérations sont nécessaires pour gérer une mémoire cache ?

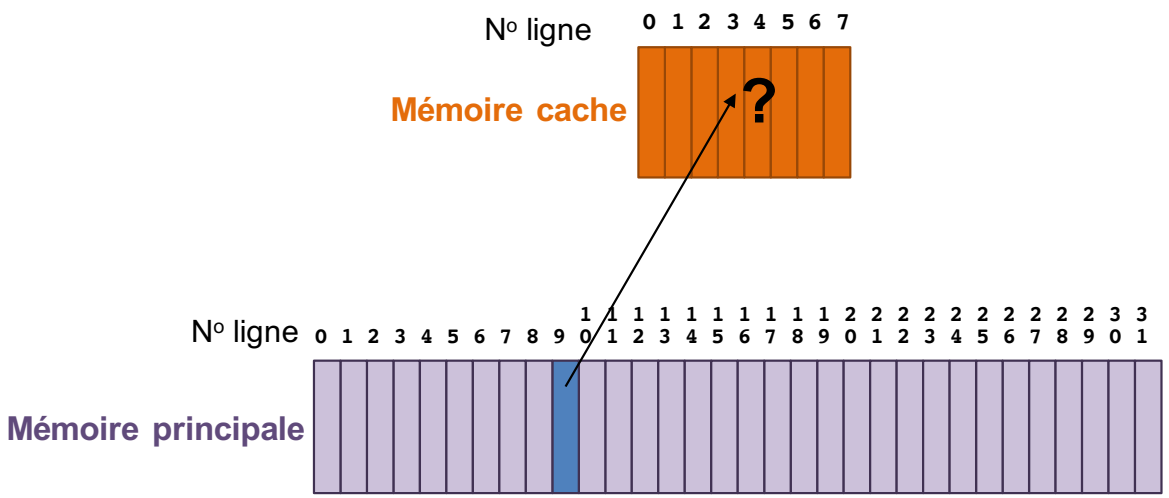
1. Comment identifier une ligne dans la mémoire cache ?

- La mémoire cache est organisée en lignes ou ensembles (set)

- Elle contient
  - Tag : l'étiquette identifie les données stockées dans la mémoire cache
  - V : le bit "valid" indique si la ligne contient des données valides
  - D : le bit "dirty" indique si l'un des mots a été modifié par le µP
  - Word : les mots de données stockées dans la mémoire cache (puissance de 2 octets)

2. Où placer une ligne dans la mémoire cache ?

- Il existe 3 types de mémoire cache
  - Cache à accès direct
  - Cache complètement associative
  - Cache à N-voies associatives

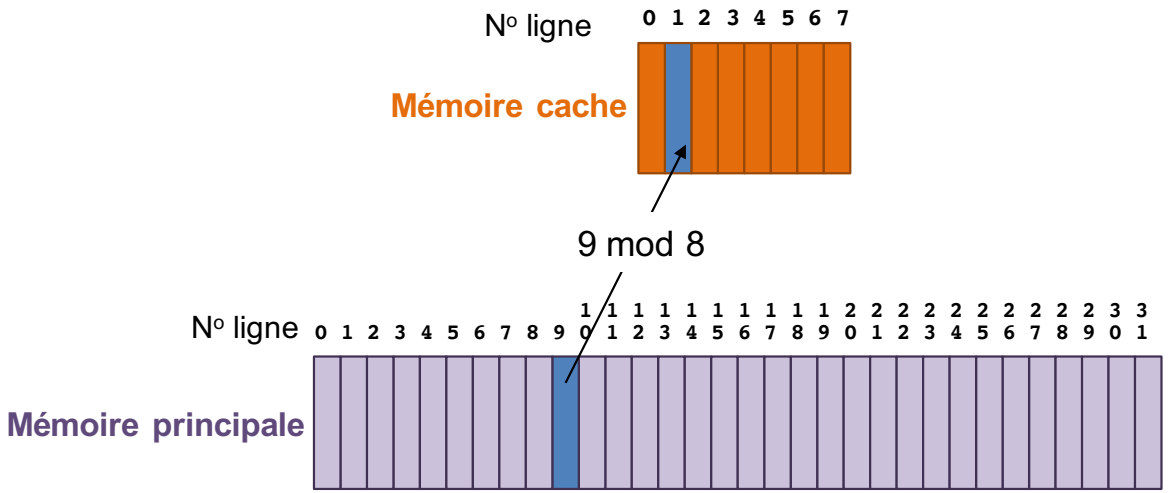


L'exemple ci-dessus représente une mémoire cache de 8 lignes et une mémoire de 32 lignes.
En réalité, un cache a plusieurs centaines de lignes et la mémoire a plusieurs millions de lignes.

Cache à accès direct (direct-mapped cache)

- Une ligne de la mémoire principale ne peut être placée qu'à une seule place dans la mémoire cache
- Le choix de la place est calculé en prenant le modulo de l'adresse de la ligne en mémoire principale avec le nombre de lignes de la mémoire cache

cache_entry = line_address % number_of_lines_in_cache



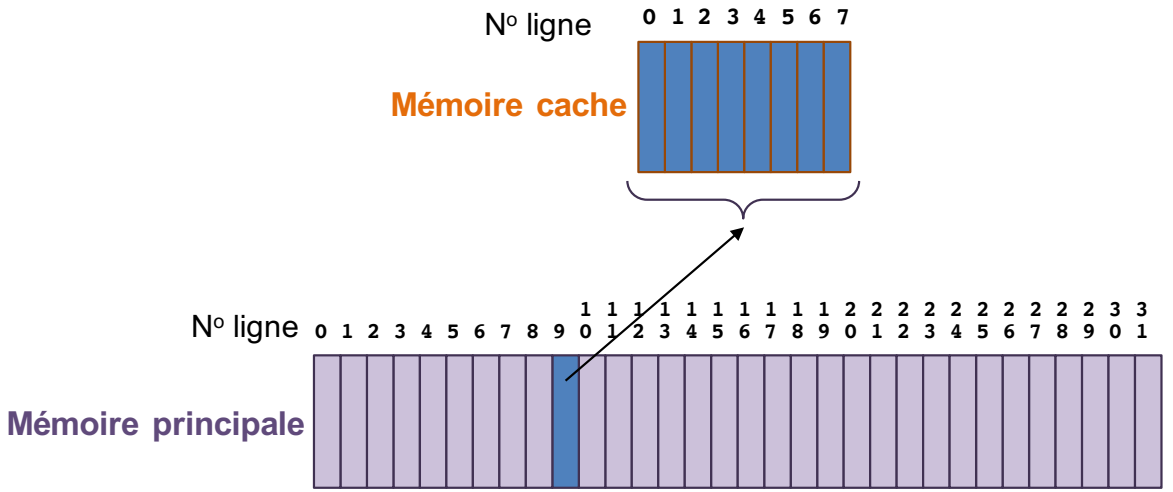
ligne no 9 ne peut aller que dans la ligne no 1 de la cache *(9 mod 8)*

\+ simple et bon marché

\- mauvaises performances

Cache complètement associative (fully associative cache)

- Une ligne de la mémoire principale peut être placée à n'importe quel endroit dans la mémoire cache

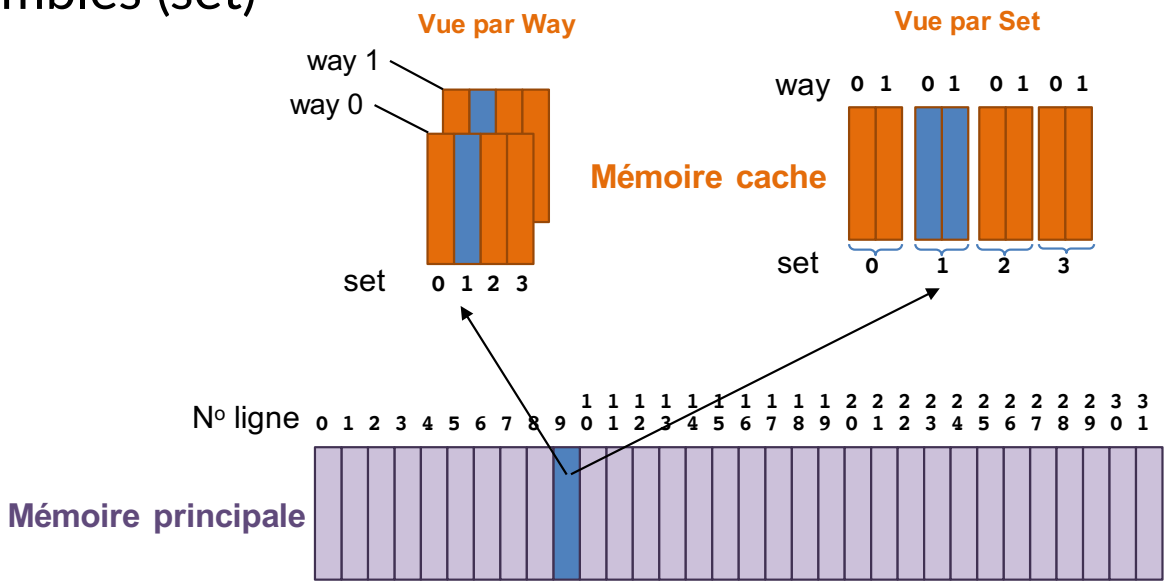


ligne no 9 peut aller dans n'importe quelle ligne de la cache

\+ excellentes performances dues à la banalisation des emplacements
\- coûts très élevés dus à la grande surface de silicium nécessaire

Cache à N-voies associatives (N-way set associative cache)

- Ce type de mémoire cache est un compromis des deux premiers
  - La mémoire cache est divisée en ensembles (set)
  - Chaque ensemble contient plusieurs lignes
  - Une voie (way) regroupe toutes les lignes de la même position des ensembles (set)

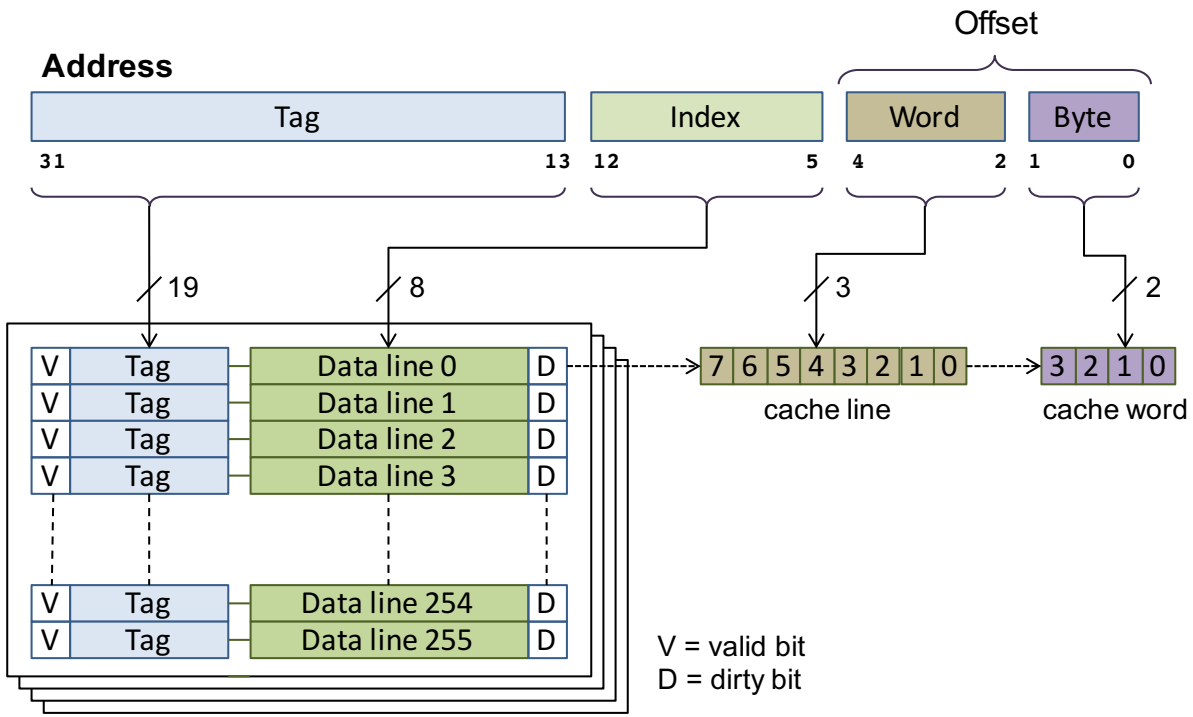


ligne no 9 peut aller soit dans la voie 0 ou 1, mais que dans le set no 1

- L'emplacement dans la mémoire cache est calculé en 2 étapes
  - Accès direct pour le choix de l'ensemble (set)
  - Complètement associatif pour le choix de la voie (way)

3. Comment retrouver une ligne ?

- L'adresse générée par le µP est décomposée en trois parties
  - Tag : partie supérieure de l'adresse stockée dans la mémoire cache
  - Index : numéro de la ligne dans la mémoire cache (set)
  - Offset :
    - Word : offset du mot dans la ligne
    - Byte : offset de l'octet dans le mot



exemple d'une cache de 32KiB à 4 voies associatives avec des lignes de 8 mots de 4 bytes

4. Quelle ligne doit être remplacée dans la mémoire cache ?

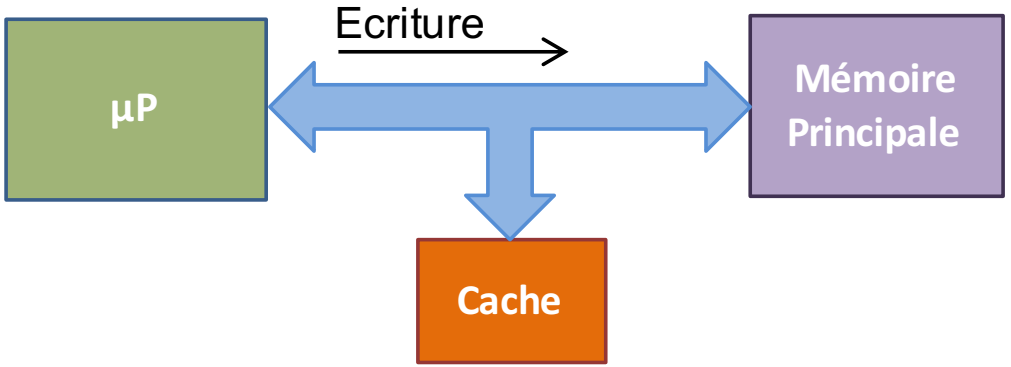
- Il existe différents algorithmes pour remplacer une ligne dans la mémoire cache
  - Aléatoire (Random) :
    le choix de la ligne à remplacer se fait au hasard
  - LRU (Least Recently Used) :
    la ligne la plus ancienne est remplacée
  - FIFO (First-In – First-Out) :
    les lignes sont remplacées dans l'ordre dans lequel elles ont été introduites dans la mémoire cache
  - LFU (Least Frequently Used) :
    la ligne la moins fréquemment utilisée est remplacée

5. Que se passe-t-il lors d'une écriture ?

- Lorsque le µP désire écrire une donnée dans la mémoire principale il doit passer par la mémoire cache
- La cache en fait une copie avant de la mettre dans la mémoire principale selon deux algorithmes à choix
  - Ecriture simultanée (write-through)
  - Ecriture différée (write-back ou copy-back)

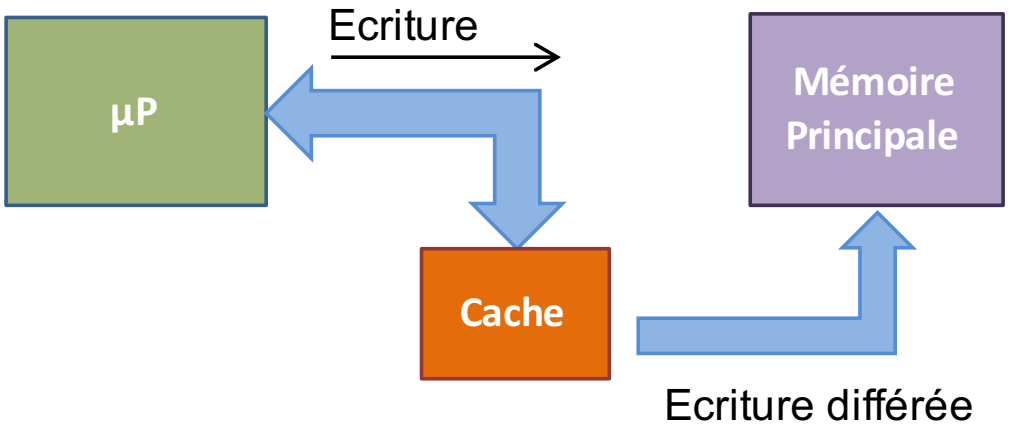
Ecriture simultanée

- La nouvelle valeur de la donnée est écrite simultanément dans la mémoire cache et dans la mémoire principale



Ecriture différée

- La nouvelle valeur de la donnée est écrite seulement dans la mémoire cache
- Cette nouvelle donnée est signalée par le bit D (drity)
- Lorsque la ligne doit être remplacée pour céder la place à une autre ligne, elle est écrite dans la mémoire principale

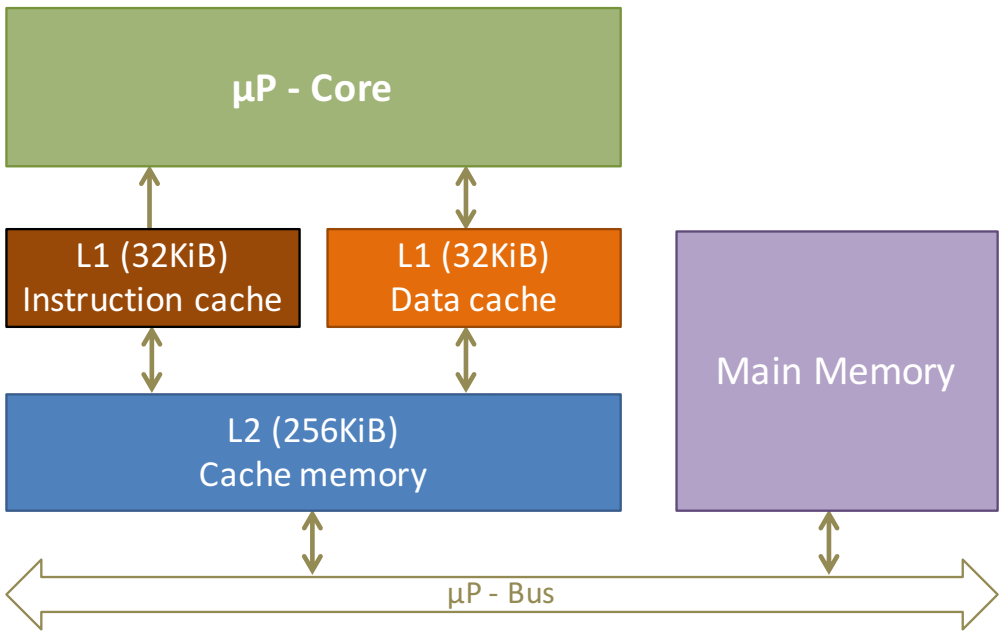


Quelles opérations sont nécessaires pour gérer une cache ?

- L'opération d'une mémoire cache nécessite que son contrôleur fournisse une série d'instructions permettant de la gérer selon les besoins des applications
  - Activer (enable) :
    la mémoire cache est activée
  - Désactiver (disable) :
    la mémoire cache est désactivée
  - Invalider (invalidate) :
    le contenu de toute la mémoire cache est marqué comme non valide
  - Invalider une ligne (invalidate single entry) :
    le contenu d'une ligne de la mémoire cache est marqué comme non valide
  - Nettoyer (clean) :
    le contenu d'une ligne de la mémoire cache est copié dans la mémoire principale si son contenu est valide et a été modifié (dirty)
  - Verrouiller (lockdown) :
    la mémoire cache est verrouillée

Implémentation du TI AM335x

- La famille de µP TI AM335x basé sur ARM Cortex-A8 implémente 2 niveaux de mémoire cache
  - une première mémoire cache L1 sur l'architecture Harvard (I-Cache & D-Cache) de 32KiB à 16 mots/ligne et 4 voies associatives
  - une deuxième mémoire cache L2 sur l'architecture Von Neumann de 256KiB à 16 mots/ligne et 8 voies associatives



Optimisations : matrices/tableaux

- Accès aux données par colonne
  → mauvaise utilisation de la mémoire cache

```
uint32_t x[4096][128];

for (int j=0; j<128; j++)
    for (int i=0; i<4096; i++)
        x[i][j] = x[i][j] * 2;
```

- Accès aux données par ligne
  → utilisation optimale de la mémoire cache

```
uint32_t x[4096][128];

for (int i=0; i<4096; i++)
    for (int j=0; j<128; j++)
        x[i][j] = x[i][j] * 2;
```

Optimisations : regroupement de données

- Attributs séparés
  → mauvaise utilisation de la mémoire cache

```
int data_attr1[1024];
int data_attr2[1024];
```

- Regroupement des attributs dans une structure
  → utilisation optimale de la mémoire cache

```
struct merged {
    int attr1;
    int attr2;
};
struct merged data[1024];
```

Optimisations : fusion de boucles

- Deux boucles séparées

  → mauvaise utilisation de la mémoire cache

```
for (int i=0; i<N; i++)
    for (int j=0; j<M; j++)
        a[i][j] = 1/b[i][j] * c[i][j];

for (int i=0; i<N; i++)
    for (int j=0; j<M; j++)
        d[i][j] = a[i][j] + c[i][j];
```

- Une boucle avec les instructions

  → utilisation optimale de la mémoire cache

```
for (int i=0; i<N; i++) {
    for (int j=0; j<M; j++) {
        a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] =   a[i][j] + c[i][j];
    }
}
```