

Les Mémoires Caches

Rappels et exercices

PAR CH. BENCHERIET

Rappel

Le processeur souhaite accéder à une adresse mémoire.

L'adresse est divisée en plusieurs parties : un tag, un index, et un offset.

Le **TAG** est utilisé pour identifier si un bloc dans la cache correspond à l'adresse demandée.

- Si le **TAG** dans la cache correspond à celui de l'adresse mémoire, alors on vérifie le **bloc** de cache, qui contient la **donnée**.
- Si le **TAG** ne correspond pas, un **miss (défaut cache)** se produit et la donnée est récupérée de la mémoire principale, puis stockée dans la cache.

Ainsi, le **TAG** est essentiel pour localiser la donnée dans la cache, mais c'est le **bloc de cache** qui contient la donnée elle-même.

Rappel

En mémoire cache, il existe plusieurs méthodes d'adressage utilisées pour déterminer comment les données sont stockées et récupérées. Voici les principales formes d'adressage du cache :

1. Adressage direct (Direct-mapped Cache)
2. Adressage associatif (Fully Associative Cache)
3. Adressage associatif par ensembles (Set-Associative Cache)

Rappel

1. Adressage direct (Direct-mapped Cache)

Chaque ligne de la mémoire principale est mappée à une ligne spécifique de la cache. Cela signifie qu'une seule ligne de la cache peut contenir les données provenant d'une adresse mémoire spécifique.

Structure de l'adresse :

- **Tag** : La portion supérieure de l'adresse, utilisée pour identifier un bloc de mémoire particulier.
- **Index** : La portion médiane de l'adresse, utilisée pour sélectionner une ligne spécifique dans la cache.
- **Offset** : La portion inférieure de l'adresse, utilisée pour localiser les données spécifiques au sein d'un bloc de cache.

Tag (adr bloc/mem)	Index (adr ligne/cache)	Offset (adr mots/ligne)
--------------------	-------------------------	-------------------------

Rappel

$$\text{Adresse mémoire} = \text{Tag} + \text{Index} + \text{Offset}$$

Offset : Nombre de bits nécessaires pour adresser une donnée à l'intérieur d'un bloc de cache. Si la taille de chaque bloc de cache est **B**, alors :

$$\text{Nombre de bits pour l'offset} = \log_2(B)$$

Index : Nombre de bits nécessaires pour indexer les différentes lignes de cache. Si le cache contient **N** lignes, alors :

$$\text{Nombre de bits pour l'index} = \log_2(N)$$

Tag : Le reste de l'adresse après le calcul de l'**index** et de l'**offset**. Si l'adresse mémoire est de **A** bits, alors :

$$\text{Nombre de bits pour le tag} = A - (\text{Nombre de bits pour l'index} + \text{Nombre de bits pour l'offset})$$

Rappel

2. Adressage associatif (Fully Associative Cache)

Un bloc de la mémoire principale peut être stocké dans **n'importe quelle ligne de la cache**, ce qui offre plus de flexibilité par rapport à l'adressage direct. Il n'y a pas de relation fixe entre l'adresse mémoire et une ligne spécifique de la cache.

Structure de l'adresse :

Tag : Toute la portion de l'adresse, utilisée pour identifier un bloc de mémoire dans la cache.

Offset : Comme dans les autres méthodes, il identifie la position spécifique des données dans le bloc de cache.

Rappel

Dans un cache totalement associatif, la totalité de l'adresse mémoire est utilisée comme **Tag** pour identifier un bloc de mémoire, et la **cache** n'a pas de notion d'index. Le cache recherche toutes les lignes en parallèle pour trouver une correspondance.

$$\text{Adresse mémoire (A)} = \text{Tag} + \text{Offset}$$

Offset : Comme dans le cache direct-mapped, le nombre de bits nécessaires pour localiser les données à l'intérieur d'un bloc est donné par (**B**: la taille de chaque bloc de cache) :

$$\text{Nombre de bits pour l'offset} = \log_2(B)$$

Tag : Le reste de l'adresse mémoire est le **Tag**, calculé comme :

$$\text{Nombre de bits pour le tag} = A - \text{Nombre de bits pour l'offset}$$

Rappel

3. Adressage associatif par ensembles (Set-Associative Cache)

Combine les deux précédentes : la mémoire cache est divisée en plusieurs **ensembles** (sets), et chaque ensemble contient plusieurs lignes. Une ligne de mémoire peut être mappée à un ensemble spécifique, mais au sein de cet ensemble, elle peut être stockée dans **n'importe quelle ligne**.

Structure de l'adresse :

Tag : La portion supérieure de l'adresse, utilisée pour identifier un bloc de mémoire particulier.

Index : Une portion de l'adresse qui détermine l'ensemble spécifique dans lequel le bloc de mémoire sera stocké.

Offset : Comme dans les autres modèles, il identifie la position spécifique des données dans le bloc de cache.

Exemple : Si la cache est à 4 voies associatives, chaque ensemble contient 4 lignes. Une adresse donnée sera mappée à un ensemble particulier, mais elle peut être stockée dans n'importe quelle ligne de cet ensemble.

Rappel

$$\text{Adresse mémoire} = \text{Tag} + \text{Index} + \text{Offset}$$

Offset : Comme dans les autres types, le nombre de bits nécessaires pour localiser une donnée à l'intérieur d'un bloc est donné par:

$$\text{Nombre de bits pour l'offset} = \log_2(B)$$

Index : Nombre de bits nécessaires pour indexer l'ensemble (set). Si le cache contient **S** ensembles et chaque ensemble contient **E** lignes, alors : Nombre de bits pour

$$\text{l'index} = \log_2(S)$$

Tag : Le reste de l'adresse après le calcul du **index** et du **offset**. Si l'adresse mémoire est de **A** bits, alors :

$$\text{Nombre de bits pour le tag} = A - (\text{Nombre de bits pour l'index} + \text{nombre de bits pour l'offset})$$

Récapitulation

Type de cache	Nombre de bits pour le Tag	Nombre de bits pour l'Index	Nombre de bits pour l'Offset
Direct-mapped	$A - (\log_2(N) + \log_2(B))$	$\log_2(N)$	$\log_2(B)$
Associatif	$A - \log_2(B)$	Aucun (pas d'indexation)	$\log_2(B)$
Set-associatif	$A - (\log_2(S) + \log_2(B))$	$\log_2(S)$	$\log_2(B)$

Enoncé de l'Exercice 1

Mots mémoire : 8 bits

Adresse mémoire (A) : 32 bits.

Taille du bloc de cache (B) : 16 octets.

Taille du cache :

Pour le cache direct-mapped : 64 lignes.

Pour le cache set-associatif : 16 ensembles de 4 lignes (4-way set-associative).

Question: Calculer le nombre de bits d'adressage (Offset? Index? Tag?) pour chaque forme d'adressage.

Solution de l'Exercice 1

Cas 1: Cache Direct-Mapped

Adresse mémoire : 32 bits

Taille du bloc de cache : 16 octets (donc $16 = 2^4$, donc l'offset occupe 4 bits) donc :

Nombre bits Offset = $\log_2(16) = 4$ bits

Cache : 64 lignes (donc $64 = 2^6$, donc l'index occupe 6 bits) donc :

Nombre bits index = $\log_2(64) = 6$ bits

Nombre bits Tag = $32 - (6 + 4) = 22$ bits

Solution de l'Exercice 1

Cas 2: Cache Associatif (Fully Associative)

Nombre bits **Offset** = $\log_2(16) = 4$ bits

Pas d'indexe

Nombre bits **Tag** = $32 - 4 = 28$ bits

Solution de l'Exercice 1

Cas 3: Cache Set-Associatif (4-Way)

Adresse mémoire : 32 bits, **Taille du bloc de cache** : 16 octets, **Cache** : 16 ensembles de 4 lignes

4-Way : 4 lignes par ensemble soit 16 ensembles ($64/4 = 16$ ensembles dans le cache)

Taille du bloc = 16 octets (16 mots) donc **Nombre bits Offset** = $\log_2(16) = 4$ bits

Index : Comme le cache contient 16 ensembles, le nombre de bits nécessaires pour l'index est :

Nombre de bits pour l'index = $\log_2(16) = 4$

Nombre bits Tag = $32 - (4+4) = 24$ bits

Exercise 2

A processor has a 32-bit memory address space (i.e. 32-bit addresses). The memory is broken into blocks of 32 **bytes each**. The computer also has a cache capable of **storing 16K bytes**.

- How many blocks can the cache store?
- Assuming the cache uses **direct-mapping**, how many bits are there in each of the **TAG**, **BLOCK (INDEXE)**, and **BYTE OFFSET** fields of the address. Show your calculations.
- Assuming the cache uses a **4-way set-associative** mapping, how many bits are there in each of the **TAG**, **SET** and **BYTE OFFSET** fields of the address. Show your calculations.

Solution de Exercice 2

A processor has a 32-bit memory address space (i.e. 32-bit addresses). The memory is broken into blocks of 32 **bytes each**. The computer also has a cache capable of **storing 16K bytes**.

- How many blocks can the cache store?

$$16\text{Kbytes} / 32 \text{ bytes} = 2^{14} / 2^5 = 2^9 = 512 \text{ blocks}$$

- Assuming the cache uses **direct-mapping**, how many bits are there in each of the **TAG**, **BLOCK (INDEXE)**, and **BYTE OFFSET** fields of the address. Show your calculations.

Tag	Block	Byte Offset
A31-A14 18-bits	A13-A5 9-bits	A4-A0 5-bits
Remaining Bits	512 blocks = 2^9 => 9 address bits	32 bytes per block

Solution de Exercice 2

c.) Assuming the cache uses a **4-way set-associative mapping**, how many bits are there in each of the **TAG**, **SET** and **BYTE OFFSET** fields of the address. Show your calculations.

Tag	Set	Byte Offset
A31-A12 20-bits	A11-A5 7-bits	A4-A0 5-bits
Remaining Bits	512 blocks / 4-ways= 2 ⁷ blocks in a set ⇒ 7 address bits	32 bytes per block

Politiques de remplacement

LRU (Least Recently Used): Remplacer le bloc qui a été utilisé le plus anciennement.

LFU (Least Frequently Used): Remplacer le bloc qui a été utilisé le moins fréquemment.

Random (Aléatoire): Choisir de manière aléatoire un bloc parmi ceux à évincer.

MRU (Most Recently Used): Remplacer le bloc qui a été le plus récemment utilisé.

FIFO (First In, First Out): Remplacer le bloc qui a été dans le cache le plus longtemps.

Enoncé de l'Exercice

Considérons un système avec un cache de **3 blocs seulement**. Chaque bloc du cache contient **3 adresses successives**. Lorsqu'un bloc doit être ajouté au cache et que celui-ci est plein, un remplacement doit être effectué. La politique de remplacement utilisée est **LFU (Least Frequently Used)**, c'est-à-dire que le bloc ayant la fréquence d'accès la plus faible sera remplacé. En cas d'égalité de fréquence, **on fait un remplacement par FIFO (remplace le bloc qui est arrivé en premier dans le cache)**.

Séquence d'accès mémoire (en décimal) : **0,3,6,4,7,9,3,0,8,1,2,13,6,12,1**

Le cache est vide au début.

Enoncé de l'Exercice 3

Les blocs sont comme suit:

Bloc A : Contient les adresses **0, 1, 2**

Bloc B : Contient les adresses **3, 4, 5**

Bloc C : Contient les adresses **6, 7, 8**

Bloc D : Contient les adresses **9, 10, 11**

Bloc E : Contient les adresses **12, 13, 14**

.... Etc.

Enoncé de l'Exercice

1. Complétez le tableau en indiquant si chaque accès génère un **hit** (**succès cache**) ou un **miss** (**défaut cache**) (bloc chargé depuis la mémoire principale).
2. Calculez les taux de **hit** et de **miss**.

Accès N°	Adresse	Bloc mémoire	Hit/Miss	Cache (Blocs)	Fréquence d'utilisation
1	0	A			
2	3	B			
3	6				
4	4				
5	7				
...					

Solution de l'Exercice

Accès N°	Adresse	Bloc mémoire	Hit/Miss	Cache (Blocs)	Fréquence d'utilisation
1	0	A	Miss	A	A: 1
2	3	B	Miss	A, B	A: 1, B: 1
3	6	C	Miss	A, B, C	A: 1, B: 1, C: 1
4	4	B	Hit	A, B, C	A: 1, B: 2, C: 1
5	7	C	Hit	A, B, C	A: 1, B: 2, C: 2
6	9	D	Miss	B, C, D	B: 2, C: 2, D: 1 (remplacement de A)
7	3	B	Hit	B, C, D	B: 3, C: 2, D: 1
8	0	A	Miss	A, B, C	B: 3, C: 2, A: 1 (remplacement de D)
9	8	C	Hit	A, B, C	B: 3, C: 3, A: 1
10	1	A	Hit	A, B, C	B: 3, C: 3, A: 2
11	2	A	Hit	A, B, C	B: 3, C: 3, A: 3
12	13	E	Miss	C, A, E	C: 3, A: 3, E: 1 (remplacement de B par FIFO car les trois ont la même fréquence)
13	6	C	Hit	C, A, E	C: 4, A: 3, E: 1
14	12	E	Hit	C, A, E	C: 4, A: 3, E: 2
15	1	A	Hit	C, A, E	C: 4, A: 4, E: 2

Solution de l'Exercice

Taux des Hits (succès) = $9/15 \times 100 = 60\%$

Taux des Miss (échec) = $6/15 \times 100 = 40\%$

Le cache est efficace, mais il y a encore une part importante d'accès qui nécessitent un accès à la mémoire principale. Ce taux peut être acceptable dans certaines situations, mais il est toujours possible d'optimiser l'efficacité du cache pour réduire le taux de miss.