Architecture des ordinateurs 2

Pr. Ch. Bencheriet

CORRECTION SÉRIE N°3

EXERCICE 1

Soit une Mémoire principale de 64 Mi mots de 32 bits, nous avons associé à cette mémoire un cache totalement associatif (Fully associative) d'une capacité de 1Mio avec des lignes de 128 octets.

1. Calculer le nombre de lignes du cache.
2. Calculer le nombre de blocs de la MP.
3. Calculer le nombre de bits d'adressage.
4. Calculer l'offset et le Tag et en déduire le format de l'adresse mémoire.
5. Donnez l'adresses du premier mot dans les lignes contenant les adresses suivantes : 0x35617A6, 0x285E832, 0x3B025BA.

EXERCICE 1

1. Calcul du nombre de lignes du cache :

C_{cache} = 1Mo , C_{ligne} = 128 oct

$$Nbre_{ligne} = \frac{1\ Mo}{128\ Octets} = 8\ K\ lignes$$

2. Calcul du nombre de blocs de la MP :

C_{MP} = 256 Mo

$$Nbre_{bloc} = \frac{256\ Mo}{128\ Octets} = 2\ M\ blocs$$

3. Calculer le nombre de bits d'adressage mémoire.

Nombre de bits d'adressage :

C_{MP} = 64 M mots = 2^{26} mots donc **26** bits d'adressage

- Cache totalement associatif donc : Adresse (A) = Tag + Offset

4. Calculer l'offset et le Tag et en déduire le format de l'adresse mémoire.

Calcul de l'offset et du Tag

1 bloc = 128 octets, 1 mot mémoire = 4 octets

Nbre_mot_bloc = $\frac{128}{4} = 32\ Mots\ par\ bloc$

Offset = log (32) = **5** bits (l'adressage des mots du bloc)

Tag = A - offset = 26 – 5 = **21** bits

EXERCICE 1

5. Donnez l'adresses du premier mot dans les lignes contenant les adresses suivantes : 0x35617A6, 0x285E832, 0x3B025BA.

Les 5 bits de poids faible de l'adresse indiquent à quel mot dans la ligne une adresse fait référence. L'adresse du premier mot dans la ligne peut être trouvée en positionnant à 0 les bits spécifiant les octets dans la ligne. Les adresses des premiers mots sont :

0x35617A6 ----- 0x35617A0

0x285E832 ----- 0x285E820

0x3B025BA ----- 0x3B025A0

EXERCICE 2

Un système de cache est utilisé pour optimiser l'accès à la mémoire. Le cache en question a une taille totale de **4 Ko**, et les blocs de données sont de **64 octets**. Pour chaque type de cache effectuer les calculs suivants :

On suppose que le bus d'adresses est de 32 bits et le bus de données est de 8 bits

1. Cache Direct
   - Calculez le **nombre de blocs** dans le cache.
   - Déterminez le **nombre de bits pour l'index**, l'**offset** et le **tag**.
   - Donnez la structure de l'adresse (en bits) et **décomposez l'adresse donnée 0x1F3A7C80** en **Tag**, **Index** et **Offset**.
2. Cache Totalement Associatif
   - Déterminez le **nombre de bits pour l'offset** et le **tag**.
   - Donnez la structure de l'adresse (en bits).
   - **Décomposez** l'adresse donnée 0x1F3A7C80 en **Tag** et **Offset**.
3. Cache Associatif par Ensemble (8 voies)
   - Calculez le **nombre d'ensembles** et le **nombre de voies par ensemble**.
   - Déterminez le **nombre de bits pour l'index**, l'**offset** et le **tag**.
   - Donnez la structure de l'adresse (en bits).
   - **Décomposez** l'adresse donnée 0x1F3A7C80 en **Tag**, **Set Index** et **Offset**.
   - Donner vos conclusions.

EXERCICE 2

1. Cache direct

Taille mot = 8 bits
Taille du cache : 4 Ko = 4096 octets
Taille d'un bloc : 64 octets.
Nombre de blocs dans le cache = Taille totale du cache/Taille d'un bloc = 4096/64 = 64.
Index : $\log_2(64) = 6$ bits.
Nombre de Mots par blocs = Taille bloc/ taille mot = 64 octs/1 oct = 64 mots
Offset : $\log_2(64) = 6$ bits.
Tag : 32– (Index + Offset) = 32– (6+6) = 20 bits.
Structure de l'adresse (32 bits) : | **Tag (20 bits)** | **Index (6 bits)** | **Offset (6 bits)** |
Pour l'adresse 0x1F3A7C80 :
0x1F3A7C80 = 0001 1111 0011 1010 0111 1100 1000 0000 (binaire).
Offset (6 bits, les moins significatifs) : 000000 (0x00).
Index (6 bits suivants) : 110010 (0x32).
Tag (20 bits restants) : 00011111001110100111 (0x1F3A7).

EXERCICE 2

2. Cache totalement associatif

Il n'y a pas d'index, car tout bloc peut être placé dans n'importe quelle ligne.

Offset : $\log_2(64) = 6$ bits

Tag : 32-6 = 26 bits

Structure de l'adresse (32 bits) : | Tag (26 bits) | Offset (6 bits) |

Pour l'adresse 0x1F3A7C80

0001 1111 0011 1010 0111 1100 1000 0000 (binaire)

Offset (6 bits, les moins significatifs) : 000000 (0x00).

Tag (26 bits restants) : 00011111001110100111110010 (0x07CE9F2).

EXERCICE 2

3. Cache associatif par ensemble (8 voies)

Nombre de blocs dans le cache = Taille totale du cache/Taille d'un bloc = 4096/64 = 64

Nombre d'ensembles = Nombre de blocs / Nombre de voies = 64/8 = 8

Set Index : $\log_2(8)=3$ bits.

Nombre de Mots par blocs = Taille bloc/ taille mot = 64 octs/ 1 oct = **64 mots**

Offset : $\log_2(64) = 6$ bits.

Tag : 32 – (Set Index + Offset) = 32–(3+6) = 23 bits.

Structure de l'adresse (32 bits) : | Tag (23 bits) | Set Index (3 bits) | Offset (6 bits) |

Pour l'adresse 0x1F3A7C80 :

Offset (6 bits, les moins significatifs) : 000000 (0x00).

Set Index (3 bits suivants) : 010 (0x2).

Tag (23 bits restants) : 00011111001110100111110 (0x0F9D3E).

EXERCICE 2

Critère	Cache direct	Cache totalement associatif	Cache associatif par ensemble (8 voies)
Placement des blocs	Chaque bloc a une ligne unique	Tout bloc peut aller dans n'importe quelle ligne	Un bloc peut aller dans plusieurs lignes d'un ensemble
Gestion des conflits	Forte probabilité de conflits	Très faible probabilité de conflits	Moins de conflits que le cache direct
Complexité matérielle	Simple	Complexe (recherche dans toutes les lignes)	Complexe (politique de remplacement nécessaire)
Usage	Bon pour des accès réguliers	Idéal pour des accès aléatoires	Bon compromis entre les deux

Conclusion

Cache direct : efficace si les accès mémoire ont peu de conflits (bien répartis).
Cache totalement associatif : idéal pour minimiser les conflits mais coûteux en matériel.
Cache associatif par ensemble : équilibre entre flexibilité et coût matériel, souvent préféré en pratique.

EXERCICE 3

Considérons un système avec un cache de **4 blocs seulement**. 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 ou LRU.**

On donne la séquence suivante d'accès mémoire (en termes de numéros de blocs) : 2,3,2,1,5,2,4,5,3,2,4,6.

1. Effectuez pour chaque politique de remplacement (LFU et LRU) le suivant :
   a. Sur un tableau récapitulatif simuler le comportement du cache a chaque accès effectué :
      En déterminant s'il s'agit d'un **hit** (succès cache) ou d'un **miss** (échec cache).
      En désignant les blocs remplacés si le cache est plein.
   b. Calculez le nombre total et les taux de **hit** et de **miss**.
2. Comparez et commentez les résultats obtenus ?

Remarque

Notons que le cache est vide au début.

Dans la politique LFU, en cas d'égalité de fréquences, utiliser la politique FIFO pour l'arbitrage.

EXERCICE 3

Politique LRU (Least Recently Used)

Étape	Accès	Cache (avant)	Hit/Miss	Cache (après)	Explication
1	2	-	Miss	[2]	Cache vide, ajoute 2.
2	3	[2]	Miss	[2, 3]	Ajoute 3.
3	2	[2, 3]	Hit	[2, 3]	2 existe dans le cache (hit).
4	1	[2, 3]	Miss	[2, 3, 1]	Ajoute 1.
5	5	[2, 3, 1]	Miss	[2, 3, 1, 5]	Ajoute 5, cache plein maintenant.
6	2	[2, 3, 1, 5]	Hit	[2, 3, 1, 5]	2 existe dans le cache (hit).
7	4	[2, 3, 1, 5]	Miss	[2, 4, 1, 5]	Remplace 3 (le moins récemment utilisé).
8	5	[2, 4, 1, 5]	Hit	[2, 4, 1, 5]	5 est déjà présent (hit).
9	3	[2, 4, 1, 5]	Miss	[2, 4, 3, 5]	Remplace 1 (le moins récemment utilisé).
10	2	[2, 4, 3, 5]	Hit	[2, 4, 3, 5]	2 existe dans le cache (hit).
11	4	[2, 4, 3, 5]	Hit	[2, 4, 3, 5]	4 existe dans le cache (hit).
12	6	[2, 4, 3, 5]	Miss	[2, 4, 3, 6]	Remplace 5 (le moins récemment utilisé).

Calcul des taux

Nombre de hits : 5
Nombre de misses : 7
Taux (hits) = (5/12) x 100 = 41,67 %
Taux (miss) = (7/12) x 100 = 58,33 %

EXERCICE 3

Politique LFU (Least Frequently Used)

Pour cette politique, nous devons suivre **le nombre d'accès** à chaque bloc pour décider quel bloc remplacer.

Étape	Accès	Cache (avant)	Fréquences	Hit/Miss	Cache (après)	Explication
1	2	-	-	Miss	[2]	Cache vide, ajout de 2.
2	3	[2]	2:1	Miss	[2, 3]	Ajout de 3.
3	2	[2, 3]	2:1, 3:1	Hit	[2, 3]	2 est déjà présent, donc c'est un hit.
4	1	[2, 3]	2:2, 3:1,	Miss	[2, 3, 1]	Ajout de 1.
5	5	[2, 3, 1]	2:2, 3:1, 1:1	Miss	[2, 3, 1, 5]	Ajout de 5. Cache plein maintenant.
6	2	[2, 3, 1, 5]	2:2, 3:1, 1:1, 5:1	Hit	[2, 3, 1, 5]	2 est déjà présent, donc c'est un hit.
7	4	[2, 3, 1, 5]	2:3, 3:1, 1:1, 5:1	Miss	[2, 4, 1, 5]	Remplacement de **3** (le plus ancien parmi les blocs ayant la même fréquence).
8	5	[2, 4, 1, 5]	2:3, 4:1, 1:1, 5:1	Hit	[2, 4, 1, 5]	5 est déjà présent (hit), donc aucun remplacement.
9	3	[2, 4, 1, 5]	2:3, 4:1, 1:1, 5:2	Miss	[2, 4, 3, 5]	Remplacement de **1** (le plus ancien parmi ceux ayant une fréquence de 1).
10	2	[2, 4, 3, 5]	2:3, 4:1, 3:1, 5:2	Hit	[2, 4, 3, 5]	2 est déjà présent (hit).
11	4	[2, 4, 3, 5]	2:4, 4:1, 3:1, 5:2	Hit	[2, 4, 3, 5]	4 est déjà présent (hit).
12	6	[2, 4, 3, 5]	2:4, 4:2, 3:1, 5:2	Miss	[2, 4, 6, 5]	Remplacement de **3** (le moins fréquemment utilisé)

EXERCICE 3

Méthode	Taux de miss (%)	Taux de hit (%)
LFU	58.33	41.67
LRU	58.33	41.67

Calcul des taux
Nombre de hits : 5
Nombre de misses : 7
Taux (hits) = (5/12) x 100 = 41,67 %
Taux (miss) = (7/12) x 100 = 58,33 %

Comparaison

Les deux politiques de gestion de cache (**LFU** et **LRU**) ont donné des résultats identiques, avec un **taux de hit de 41.67%** et un **taux de miss de 58.33%**. Toutefois, en fonction des caractéristiques d'un système réel et des types d'accès aux données, la différence entre les deux politiques pourrait devenir significative.

- **LFU** peut être plus efficace si les accès sont très réguliers et si certaines données sont utilisées bien plus fréquemment que d'autres.
- **LRU**, quant à lui, est plus simple à implémenter et peut être plus efficace lorsque les données les plus récemment utilisées ont plus de chances d'être utilisées à nouveau.

EXERCICE 4

Nous cherchons à optimiser une mémoire cache ayant une capacité totale de **8 octets**. Trois conceptions de **cache direct** sont possibles selon la taille des blocs :
Les mots mémoires sont de 1 octet
Conception 1 : Cache avec des blocs de **1 octet et temps d'accès : 2 cycles**
Conception 2 : Cache avec des blocs de **2 octets et temps d'accès : 3 cycles**
Conception 3 : Cache avec des blocs de **4 octets et temps d'accès : 5 cycles**
Notons que le temps d'attente en cas de défaut (*miss stall time*) : **25 cycles**
On donne la séquence suivante d'accès mémoire (en terme d'adresse) : **1, 134, 212, 1, 135, 213, 162, 161, 2, 44, 41, 221**.

Remarques

- ✓ Chaque accès à la mémoire cache (un succès ou un défaut) nécessite un temps d'accès
- ✓ Les adresses ont été converties en format binaire pour simplifier la tâche.

Faire pour chaque conception le suivant :

1. Décomposez les adresses en **bits de tag**, **bits d'index**, et **bits d'offset**, selon la taille des blocs.
2. Complétez le tableau (ci-dessous) des accès en identifiant si chaque adresse entraîne un **hit** ou un **miss**.
3. Calculer le taux de défaut (miss rate) et en déduire la configuration qui minimise les défauts ?
4. Calculer le nombre de cycle totale et en déduire la conception de cache optimale.

EXERCICE 4

1. Décomposez les adresses en bits de tag, bits d'index, et bits d'offset?

Conception 1 (1 octet par bloc) :

Offset = 0 bits.
Index = 3 bits ($\log_2$(8 lignes)).
Tag = Bits restant.

Conception 2 (2 octets par bloc) :

Offset = 1 bit.
Index = 2 bits ($\log_2$(8 octets / 2 octets par bloc)).
Tag = Bits restant.

Conception 3 (4 octets par bloc) :

Offset = 2 bits.
Index = 1 bit ($\log_2$(8 octets / 4 octets par bloc)).
Tag = Bits restant.

Cache	Tag	Index	Offset
Conception 1	A_n-A_3	A_2-A_0	aucun
Conception 2	A_n-A_3	A_2-A_1	A_0
Conception 3	A_n-A_3	A_2	A_1-A_0

EXERCICE 4

2. Complétez le tableau (ci-dessous) des accès en identifiant si chaque adresse entraîne un **hit** ou un **miss**.

Conception 1 (1 octet par bloc)

Chaque bloc correspond exactement à 1 octet, donc il y aura toujours **un défaut (miss)** à chaque accès unique à une nouvelle adresse.

Adresse	Adresse (en binaire) 8 bits LS de l'adresse	Index (3 bits)	Hit/Miss
1	0000 0001	001(1)	Miss
134	1000 0110	110(6)	Miss
212	1101 0100	100(4)	Miss
1	0000 0001	001(1)	Hit
135	1000 0111	111(7)	Miss
213	1101 0101	101(5)	Miss
162	1010 0010	010(2)	Miss
161	1010 0001	001(1)	Miss
2	0000 0010	010(2)	Miss
44	0010 1100	100(4)	Miss
41	0010 1001	001(1)	Miss
221	1101 1101	101(5)	Miss

EXERCICE 4

Conception 2 (2 octets par bloc)

Les blocs contiennent **2 adresses**. Une fois un bloc chargé, la deuxième adresse correspondante sera un **hit**.

Adresse	Adresse (en binaire) 8 bits LS de l'adresse	Index (2 bits)	Offset	Hit/Miss
1	0000 0001	00 (0)	1	Miss
134	1000 0110	11 (3)	0	Miss
212	1101 0100	10 (2)	0	Miss
1	0000 0001	00 (0)	1	Hit
135	1000 0111	11 (3)	1	Hit
213	1101 0101	10 (2)	1	Hit
162	1010 0010	01 (1)	0	Miss
161	1010 0001	00 (0)	1	Miss
2	0000 0010	01 (1)	0	Miss
44	0010 1100	10 (2)	0	Miss
41	0010 1001	00 (0)	1	Miss
221	1101 1101	10 (2)	1	Miss

EXERCICE 4

Conception 3 (4 octets par bloc)

Les blocs contiennent **4 adresses**. Une fois un bloc chargé, les trois suivantes seront des **hits**.

Adresse	Adresse(en binaire) 8 bits LS de l'adresse	Index (1 bit)	Offset	Hit/Miss
1	0000 0001	0	01(1)	Miss
134	1000 0110	1	10(2)	Miss
212	1101 0100	1	00(0)	Miss
1	0000 0001	0	01(1)	Hit
135	1000 0111	1	11(3)	Miss
213	1101 0101	1	01(1)	Miss
162	1010 0010	0	10(2)	Miss
161	1010 0001	0	01(1)	Hit
2	0000 0010	0	10(2)	Miss
44	0010 1100	1	00(0)	Miss
41	0010 1001	1	01(1)	Miss
221	1101 1101	1	01(1)	Miss

EXERCICE 4

3. Calculer le taux de défaut (miss rate) et en déduire la configuration qui minimise les défauts ?

Le meilleur design est **C**onception **2**.

	Conception 1	Conception 2	Conception 3
Nombre de Miss	11/12	9/12	10/12
Taux de Miss	92%	75%	83%

4. Calculer le nombre de cycle total et en déduire la conception de cache optimale

Le meilleur design est **C**onception **2**.

	Conception 1	Conception 2	Conception 3
Nombre de cycles	2*12 + 11*25 = 299	3*12+9*25 = 251	5*12+10*25 = 310