Mémoire cache d'une machine de Von Neumann

Partie 1 : Etude théorique

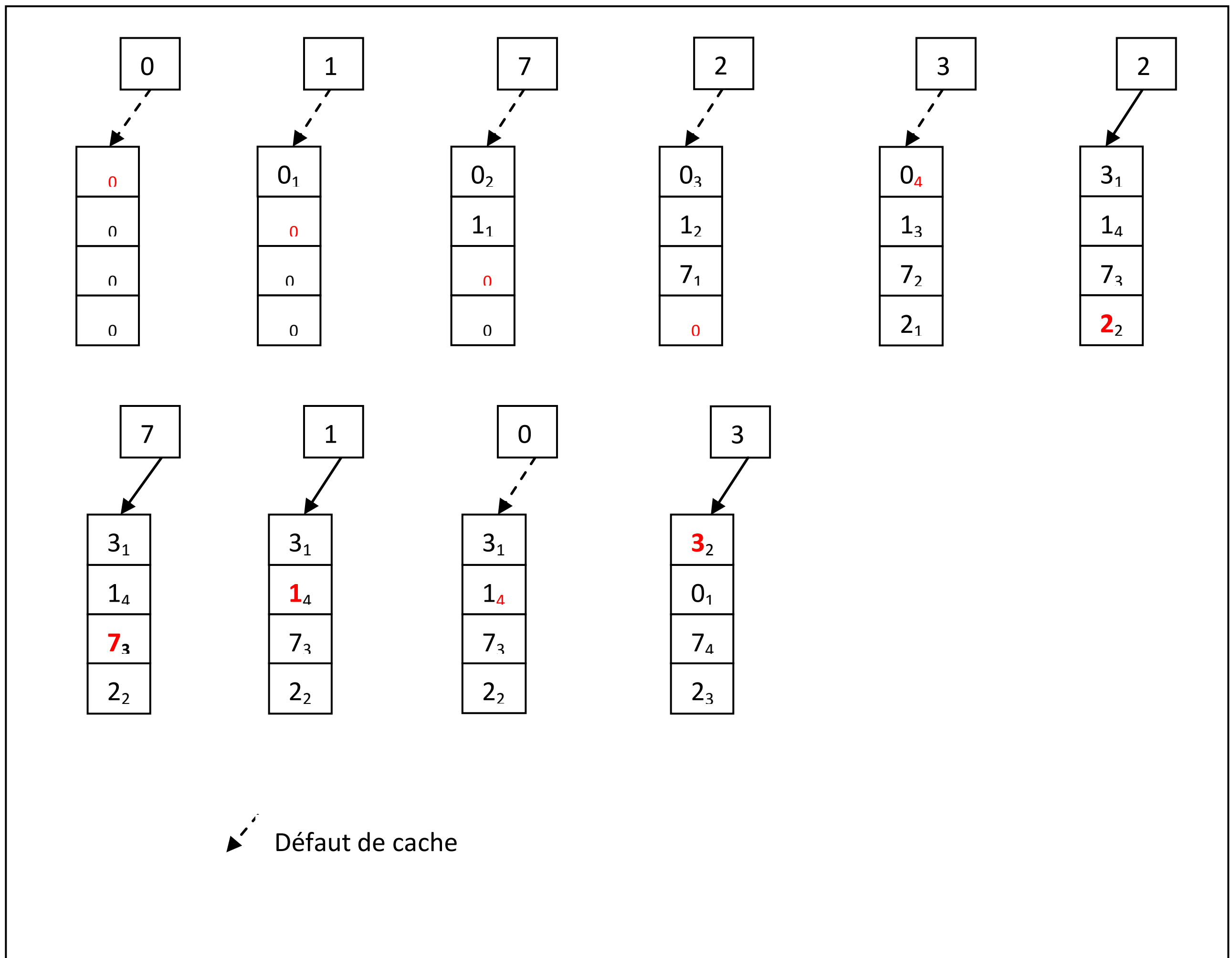
Thibault COUDERT ; Thomas TYGREAT
Architecture Des Ordinateurs

Exercice 1 – Etude des stratégies de remplacement

Stratégie FIFO : 6 défauts de cache

Une stratégie FIFO nécessite un tag pour chaque bloc indiquant à chaque instant la position du bloc dans la file. Ainsi, le bloc dont le tag est '1' est le bloc qui a été ajouté en dernier, donc il sortira en dernier. Le bloc qui a le tag n° '4' est le bloc ajouté en premier, donc il sortira en premier.

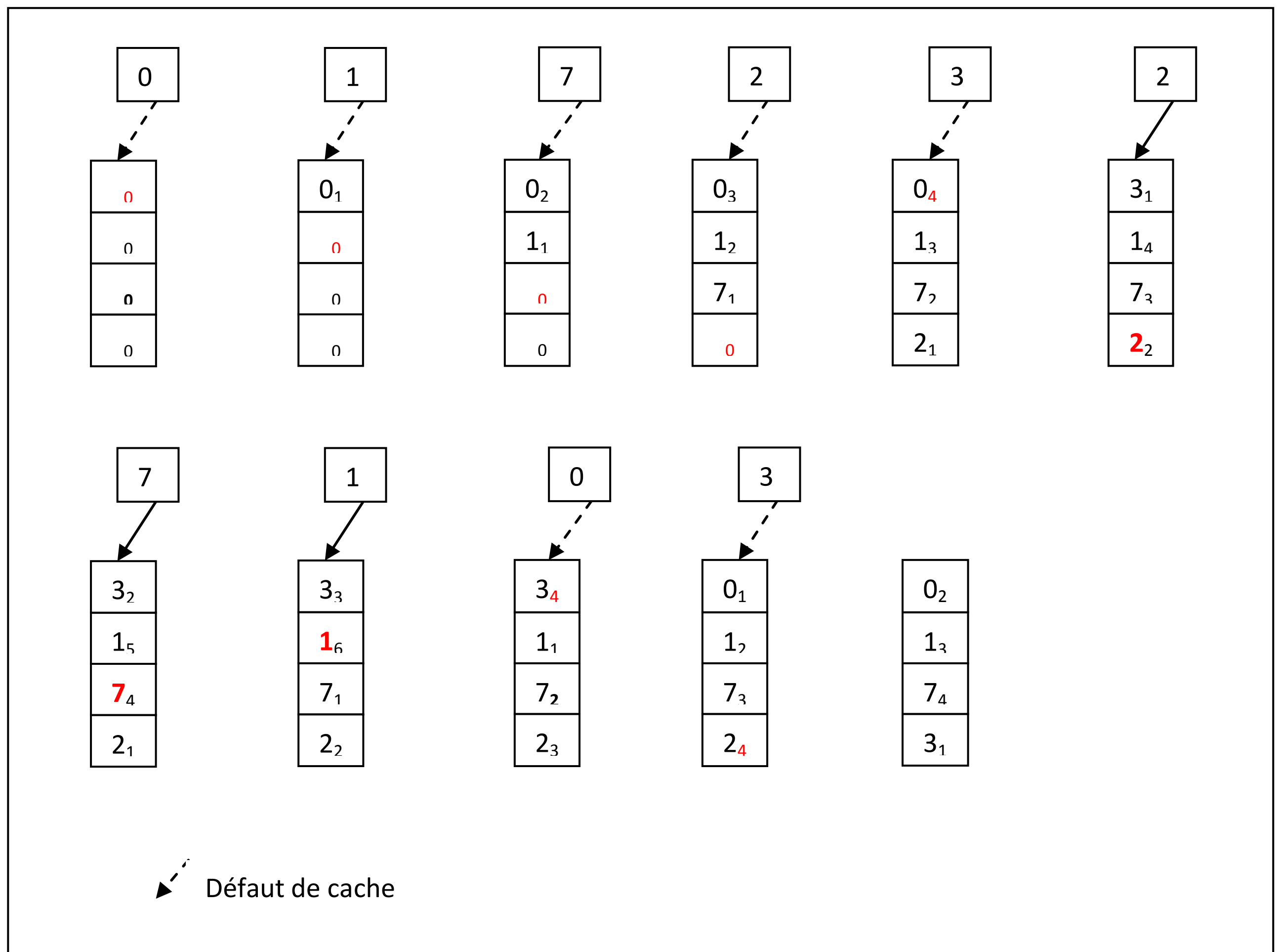
Il est simple de gérer la mise à jour des tags : il suffit d'incrémenter tous les tags quand un bloc est écrasé, et le bloc écrasé prend la valeur '1'.



Stratégie LRU : 7 défauts de cache

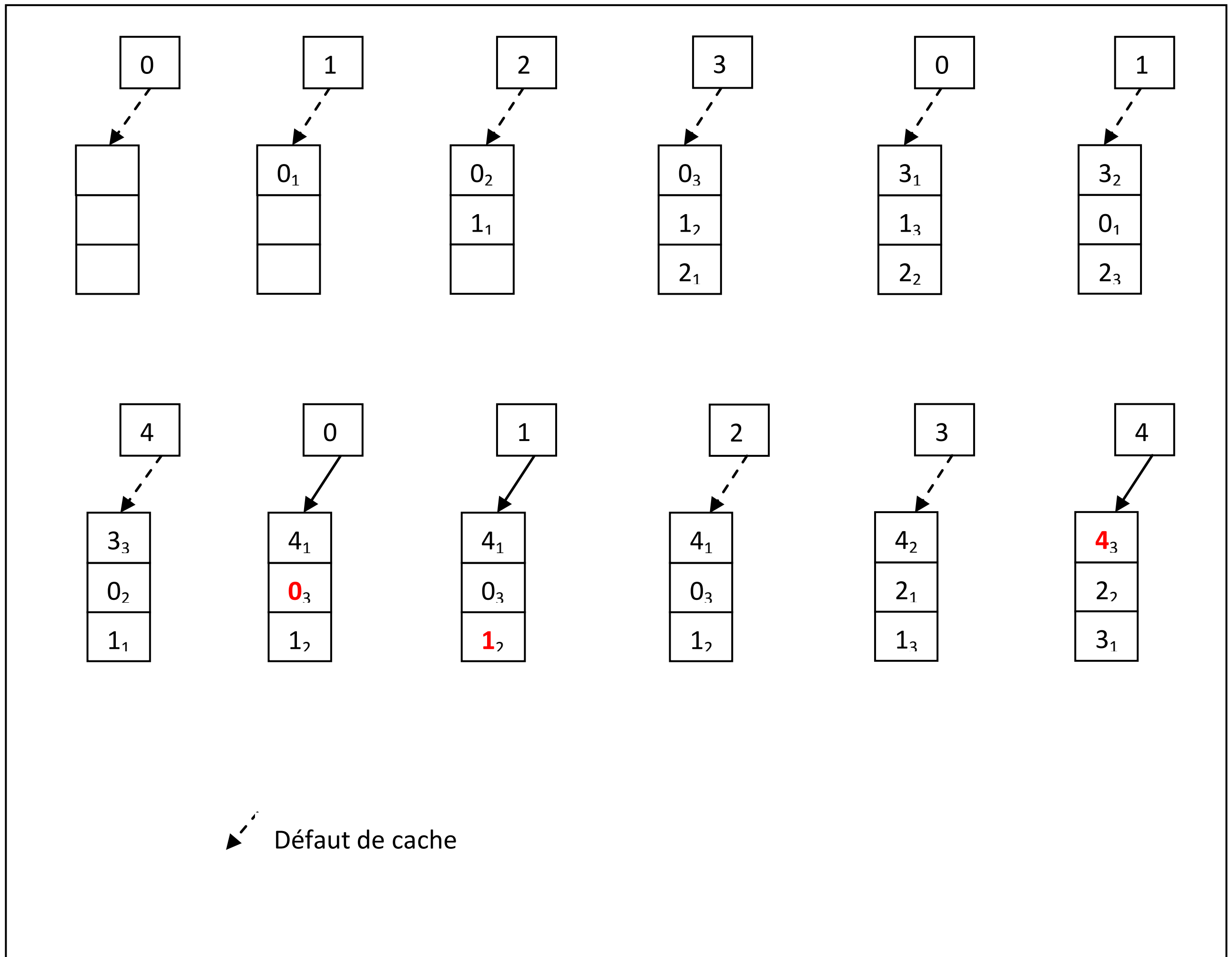
La stratégie LRU (Last Recently Used)nécessite un tag pour chaque bloc qui donne le temps de dernière utilisation. A chaque chaque fois que le cache est consulté, tous les tags sont incrémentés. Le tag du bloc qui est consulté est remis à ‘1’.

S’il y a défaut de cache, on incrémente tous les tags sauf celui du bloc remplacé, qui est réinitialisé à ‘1’. Le bloc remplacé est celui donc le tag est le plus grand (celui qui a été utilisé le plu longtemps).

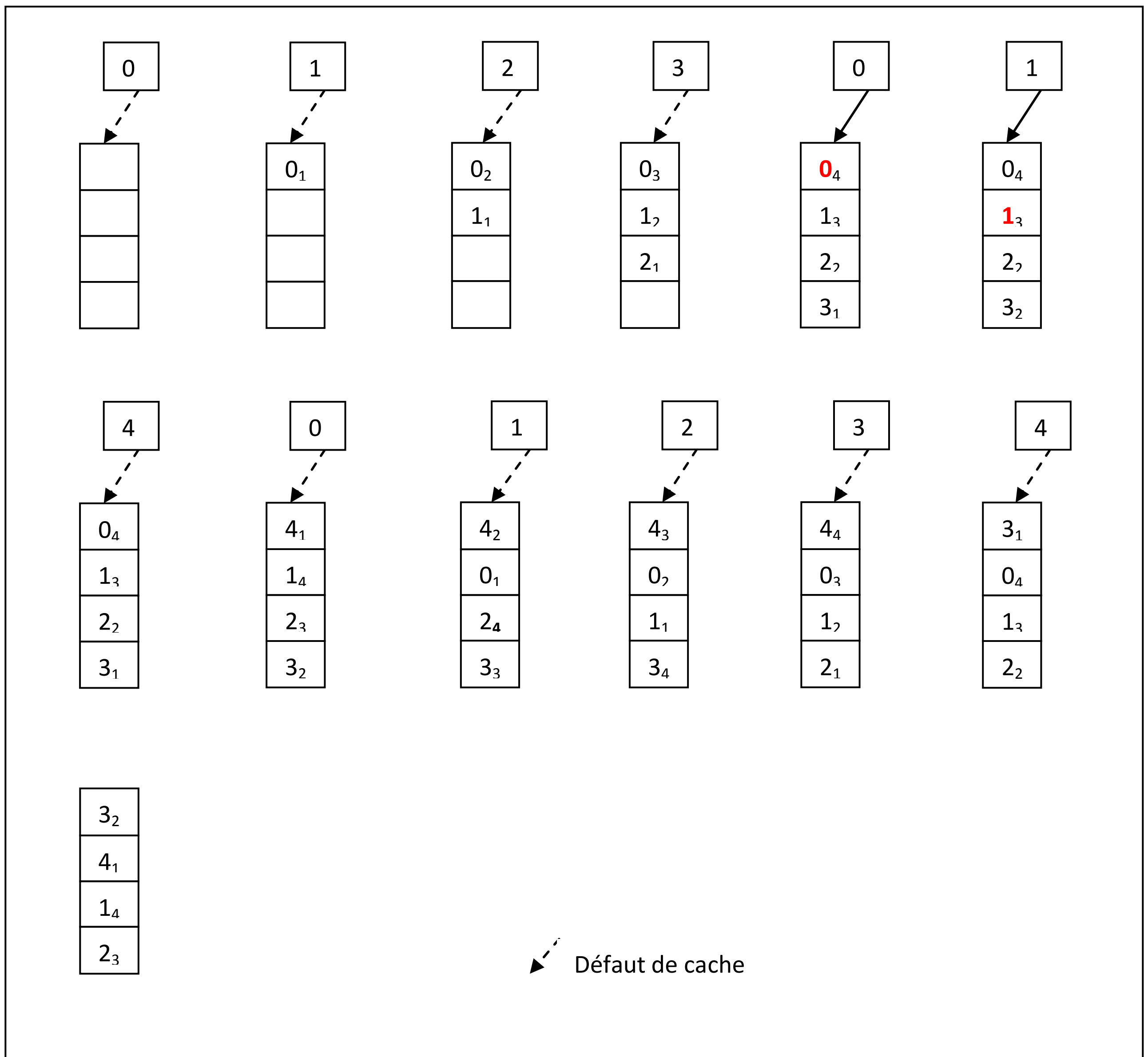


Anomalie de BELADY

Avec 3 blocs : 9 défauts de cache



Avec 4 blocs : 10 défauts de cache

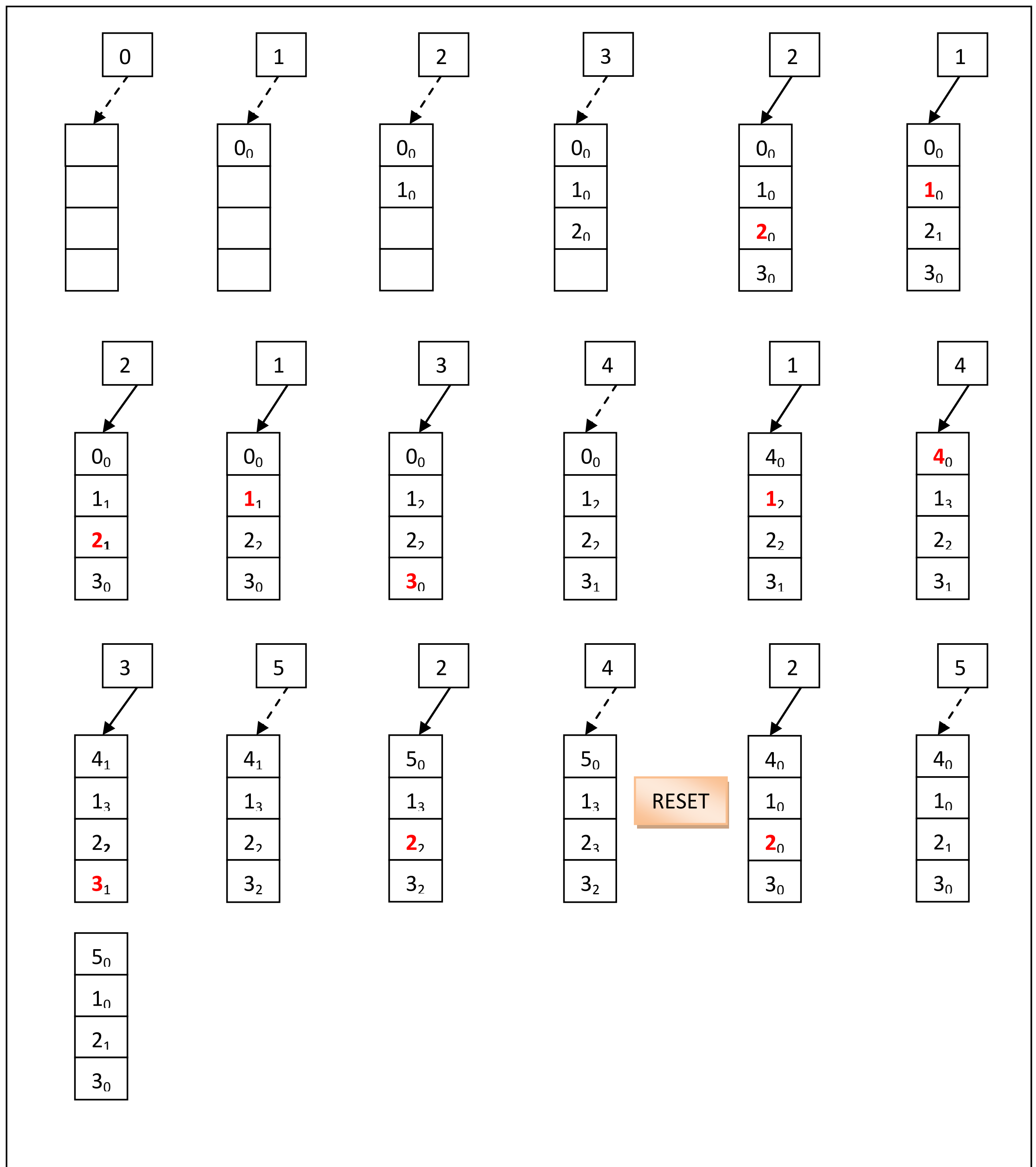


Cette anomalie souligne le fait qu’augmenter la taille du cache ne diminue pas systématiquement le nombre de défauts de cache, contrairement à ce qu’on pourrait penser.

Stratégie LFU sur 4 blocs avec un RESET : 8 défauts de cache

La stratégie LFU consiste à supprimer les blocs de mémoire les moins utilisés (ceux auxquels on accède le moins). Il nous faut donc un tag qui compte les accès à chaque bloc. Il est incrémenté lorsqu'on accède à ce bloc. Lorsqu'il y a défaut de cache, on remplace le bloc dont le tag est le plus petit (si il y a égalité on prend le premier qui vient).

Il faut prévoir en plus un compteur pour savoir quand faire le RESET.



Le but du RESET est d'empêcher un bloc qui a été beaucoup utilisé pendant un moment d'encombrer la mémoire. En effet, il est possible qu'il ait été utile pendant un moment donc son nombre d'utilisation est important et il n'est jamais remplacé, alors qu'il est peut-être devenu inutile.

Exercice 2 – Synchronisation en écriture d'un cache avec la mémoire principale

MP : Mémoire Principale
MC : Mémoire Cache

Etape	Write-Back	Write-Through
1	MP --> MC	MP --> MC
2	MP --> MC	MP --> MC ; **MC --> MP (MàJ du bloc n°2)**
3		**MC --> MP (MàJ du bloc n°2)**
4		**MC --> MP (MàJ du bloc n°2)**
5		**MC --> MP (MàJ du bloc n°2)**
6	MP --> MC	MP --> MC ; **MC --> MP (MàJ du bloc n°3)**
7	MP --> MC	MP --> MC
8	MP --> MC	MP --> MC
9		
10	**MC --> MP (MàJ du bloc n°2)** ; MP --> MC	MP --> MC ; **MC --> MP (MàJ du bloc n°6)**
11	**MC --> MP (MàJ du bloc n°3)** ; MP --> MC	MP --> MC
TOTAL	9 échanges	13 échanges

La gestion du write-through nécessite un tag supplémentaire car on doit savoir si l'on doit mettre à jour le bloc dans la mémoire principale avant de l'écraser en mémoire cache. Il s'agit d'un booléen que l'on met à 'vrai' si l'on effectue une écriture sur ce bloc, et qui est initialisé à 'faux'.

Cependant, le write-back nécessite un nombre d'échanges beaucoup plus important entre la mémoire cache et la mémoire principale, ce qui fait perdre beaucoup de temps.

Exercice 3 - Mesure d'efficacité d'un cache

1. Exprimer N_w

$$N_w = M_{rate} * N_{at} * M_{penalty}$$

N_{at} = nombre de données à récupérer
M_{rate} = pourcentage de données que l'on va chercher en mémoire centrale
$M_{penalty}$ = temps d'accès pour récupérer une donnée en mémoire centrale

2. En déduire l'expression de T_{CPU}

- $N_x = IC * CPI_x$
- $N_w = M_{rate} * N_{at} * M_{penalty}$
- $N_{at} = N_a * IC$

$$T_{CPU} = T_{cycle}.(IC * CPI_x + N_a * IC * M_{rate} * M_{penalty})$$
$$= (T_{cycle} * IC).(CPI_x + N_a * M_{rate} * M_{penalty})$$

3. Application

- *Calcul du temps CPU sans mémoire cache*

$$T_{CPU}^{(1)} = T_{cycle} * (IC * N_a * 2 + IC * CPI_x)$$
$$= (T_{cycle} * IC)(2 * N_a + CPI_x)$$

- *Calcul du temps CPU avec mémoire cache*

$$T_{CPU}^{(2)} = (T_{cycle} * IC)(CPI_x + N_a * M_{rate} * M_{penalty})$$

- ***Application***

$$\frac{T_{CPU}^{(2)}}{T_{CPU}^{(1)}} = \frac{CPI_x + N_a * M_{rate} * M_{penalty}}{2 * N_a + CPI_x}$$

$$\frac{T_{CPU}^{(2)}}{T_{CPU}^{(1)}} = \frac{8,5 + 3,0 * 0,11 * 6}{2 * 3 + 8,5}$$

$$\frac{T_{CPU}^{(2)}}{T_{CPU}^{(1)}} = \frac{10,48}{14,50}$$

$$\frac{T_{CPU}^{(2)}}{T_{CPU}^{(1)}} \sim 72\%$$

Exercice 4 – Cache associatif par ensemble

La taille de l’adresse est de 9 bits :

$16\ blocs = 2^4$
$4\ mots = 2^2$
$8\ ensembles = 2^3$

Soit :

- 4 bits à réserver pour connaître le numéro du bloc (TAG)
- 2 bits à réserver pour connaître la position du mot dans le bloc (offset)
- 3 bits à réserver pour connaître l’ensemble qui contient ce bloc

De plus, pour la stratégie LRU il faut réserver pour chaque bloc un tag donnant l’ordre de plus récente utilisation au sein de son ensemble. Etant donné qu’il y a 16 blocs dans un ensemble, il faut réserver 4 bits en plus pour ce tag.

En stratégie LFU, il faut également 4 bits en plus pour chaque bloc, mais aussi un compteur qui va de 0 à 1023 soit 9 bits (on peut garder le même compteur pour tous les ensembles).

Théorie sur les mémoires caches

I. Introduction

Afin d’améliorer les performances des ordinateurs, les constructeurs d’ordinateurs utilisent de plus en plus de mémoires dites cachées. Le principe de ces mémoires cachées a été développé au début des années 1960 par un ingénieur français. Une mémoire cache est une mémoire intermédiaire qui se situe entre un support de données (une autre mémoire dite principale) et le matériel informatique (souvent le microprocesseur) qui souhaite accéder à ces données. Elle est constituée de copies temporaires de données qui proviennent de la mémoire principale. L’intérêt des mémoires caches est leur vitesse d’accès : en effet, du fait de leur proximité physique avec le microprocesseur et de la qualité des matériaux utilisés, les performances d’accès en lecture ou en écriture aux mémoires caches sont grandement améliorées.
On pourrait se poser la question de savoir pourquoi les constructeurs n’utilisent pas uniquement des mémoires de type cache. Deux raisons à ceci ; la première est financière, puisque les coûts des matériaux utilisés pour une mémoire cache sont très élevés. La deuxième raison est d’ordre physique, si on augmente la taille de la mémoire cache, son efficacité en est ralentie puisque l’on a plus de données à rechercher. Toujours dans un souci de performances et pour garantir la plus grande proximité possible entre le microprocesseur et les données, les mémoires caches sont souvent situées dans les microprocesseurs, et la place est donc très limitée ce qui explique la faible capacité de ces mémoires.

L’efficacité des mémoires caches a été confirmée non seulement de manière empirique, mais aussi de manière théorique. En effet, des études sur les comportements des programmes ont permis de tirer deux conclusions :

1. Le principe de localité spatiale

   Lorsque l’on récupère une donnée située à une adresse i, il y a une grande probabilité pour que la prochaine donnée à laquelle on accède soit située à une adresse j qui est physiquement proche de l’adresse d.

2. Le principe de localité temporelle

   Lorsque l’on récupère une donnée à un moment donné t, il y a une grande probabilité pour que l’on ait à nouveau besoin de cette donnée dans un futur immédiat.

II. Fonctionnement

Voici le fonctionnement du processeur en présence d'une mémoire cache :

a. Le microprocesseur demande une information
b. Si cette information se situe dans le cache, celui la renvoie : succès de cache. Sinon, on parle de défaut de cache, et la mémoire cache demande l'information à la mémoire principale
c. La mémoire principale renvoie l'information à la mémoire cache
d. Une fois enregistrée en mémoire cache (stockée), l'information est transmise au microprocesseur

III. Les différents types de mémoire cache

La mémoire cache étant de petite taille, elle ne peut contenir qu'une petite partie de la mémoire principale. Il faut donc savoir à chaque instant quelle partie de la mémoire principale est stockée dans la mémoire cache. Pour ce faire, on utilise le mapping. Il en existe de trois sortes :

- ***Fully associative cache*** : une donnée quelconque de la mémoire principale peut se situer n'importe où dans la mémoire cache. Efficace au niveau des défauts de cache mais temps de recherche important.
- ***Direct mapped cache*** : la position d'une donnée dans la mémoire cache est entièrement déterminée par sa position dans la mémoire principale. Très efficace pour la recherche, elle est néanmoins très coûteuse en termes de défauts de cache.
- ***N-way set associative cache*** : il s'agit d'un compromis entre les deux versions précédentes, qui assure une bonne efficacité de recherche et un pourcentage de défaut de cache acceptable. C'est cette méthode qui est universellement utilisée de nos jours.

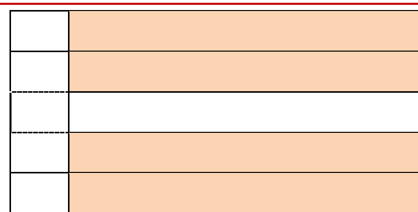
IV. *N-way set associative cache*

L'idée est de diviser la mémoire cache en 2^N ensembles distincts. Chaque donnée de la mémoire principale ne peut être chargée que dans un seul ensemble. Cependant, elle peut prendre n'importe quelle place à l'intérieur de cet ensemble. Ainsi, la position d'une donnée dans la mémoire cache étant limité à quelques adresses, la recherche de celle-ci est rapide. Cependant, le fait que la donnée puisse être stockée à plusieurs adresses différentes, limite le nombre de défaut de caches.
L'adresse d'une donnée dans la mémoire cache est divisée en plusieurs partie : il faut identifier la voie de la mémoire cache dans laquelle la donnée se trouve (TAG), repérer l'ensemble considéré (Index), puis récupérer le mot parmi la ligne(Offset).

Tag

Voie 1

Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3



Tag

Voie 2

Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3

Tag

Voie K

Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3
Mot 0	Mot 1	Mot 2	Mot 3

Dans l'exemple ci-dessus, chaque couleur représente un ensemble. Pour récupérer un mot, il faut connaître la voie dans laquelle il se situe (il y a autant de voies que d'éléments par ensemble), l'ensemble considéré (il y en a 2^N) et enfin le mot parmi la ligne.