

Exe 1 : Cache Direct-Mapped en Write-through avec tampon d'écriture postée

On étudie un système contenant un cache d'instruction et un cache de donnée à correspondance directe, ils sont tout deux indépendants.

Les adresses sont sur 32 bits. La stratégie d'écriture en mémoire est de type Write Through : toute requête d'écriture émise par le processeur se traduit par une écriture en mémoire et le cache n'est mis à jour que si le bloc est présent dans le cache (cas de Hit).

Un tampon d'écriture postée permet d'éviter de geler le processeur lors des instructions d'écriture tant qu'il n'y a pas un trop grand nombre d'écritures consécutives.

G/ Structure des adresses : on considère des caches Direct Mapped ayant une capacité de 8 lignes de 4 mots.

a/ Donner la structure de l'adresse physique vue par le cache en précisant le nombre de bits dans chaque champ. Quel est le nombre total de lignes de cache dans l'espace d'adressage ?

Notre cache a une capacité de 8 lignes où chaque ligne correspond à $N = 4 \text{ mots} = 16 \text{ octets}$.

Le nombre de bits nécessaires pour décrire un déplacement dans une ligne/bloc est donc égal à $\log(N)$

donc 4 bits. De plus, comme nous sommes sur un cache Direct Mapped, son degré d'associativité est donc $\# \text{deg-Ass} = 1 \text{ Expl/Ens}$, donc on a

$$\# \text{Ens} = \frac{\# \text{Empl}}{\# \text{deg-Ass}} = \frac{8}{1} = 8 \text{ ensembles, il faut donc } \log(\# \text{Ens}) = \log(8) = \underline{3 \text{ bits}}$$

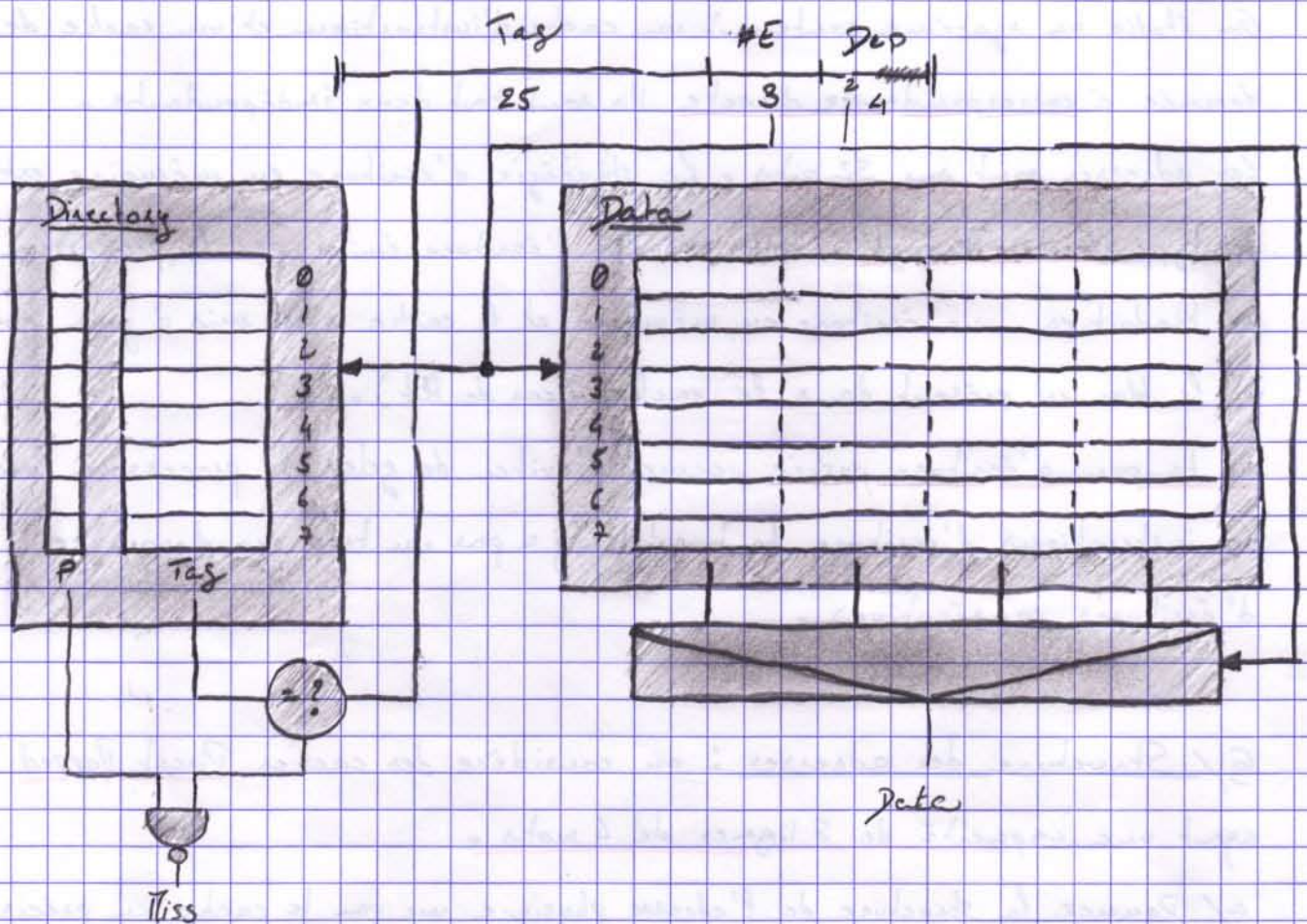
pour décrire le numéro d'ensemble d'un bloc. Pour le Tag, il suffit de compléter : $32 - \log(N) - \log(\# \text{Ens}) = \underline{25 \text{ bits}}$.

mot 0	mot 1	mot 2	mot 3	
				0
				1
				2
				3
				4
				5
				6
				7
0	4	8	12	

1 ligne = 1 Bloc

b/ Donnez un schéma du cache mettant en évidence l'utilisation de chaque champ.

Schéma de l'utilisation de l'adresse par le cache



Q2/ Analyse d'une application : Soit le programme assembleur sur le sujet qui réalise l'addition vectorielle de deux vecteurs : $C = A + B$. Les 3 vecteurs A, B et C possèdent 20 composantes chacun et sont représentés par des tableaux d'entiers stockés dans le segment "data". Avant d'entrer dans la boucle, on charge l'adresse de l'élément A[0] dans le registre R8, et l'adresse de la 1^{ère} case mémoire libre après le tableau A dans le registre R9.

```

.data
    A: .align 2
        .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        .word 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
        .space 48
    B: .word 101, 102, 103, 104, 105, 106, 107, 108, 109, 110
        .word 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
        .space 48
    C: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
    .text
        .align 2
        .global main
        .ent main
    
```

```

.set mwordlen

main:
    lui    R8, 0x1000
    addiu  R9, R8, 80

loop:
    lw     R10, 0(R8)
    lw     R11, 128(R8)
    addi   R8, R8, 4
    add    R10, R10, R11
    bne    R8, R9, loop
    sw     R10, 256(R8)

exit:
    ori    R2, R0, 10
    syscall
    
```


c/ L'adresse de base du segment "text" est $0x0040\ 0000$. La 1^{ère} instruction rangée à cette adresse est donc l'instruction associée à l'étiquette "main".

Quelle est l'adresse de la 1^{ère} instruction de la boucle (instruction associée à l'étiquette "loop") ?

D'après la description en assembleur du segment "text", on a le tableau :

Etiquette	Instruction	Adresse
main	lui	$0x0040\ 0000$
	addiu	$0x0040\ 0004$
loop	lw	$0x\ \dots\ 0008$
	lw	$0x\ \dots\ 000C$
	addi	$0x\ \dots\ 0010$
	add	$0x\ \dots\ 0014$
	bne	$0x\ \dots\ 0018$
	sw	$0x\ \dots\ 001C$
	ori	$0x\ \dots\ 0020$
exit	syscall	$0x\ \dots\ 0024$

Les instructions sont codées

sur 32 bits \Leftrightarrow 4 octets,

donc il suffit d'incrémenter

de 4 l'adresse d'une instr.

pour passer à la suivante.

L'adresse de la 1^{ère} instr.

de la boucle est $0x0040\ 0008$.

d/ L'adresse de base du segment "data" est $0x1000\ 0000$. Quelles sont les adresses des éléments $A[0]$, $B[0]$ et $C[0]$ des trois tableaux ?

D'après la description en assembleur du segment "data", on a le tableau :

Etiquette	Data (Little-Endian) @ +3 +2 +1 +0	Adresse
A	$0x\ 00\ 00\ 00\ 01$	$0x1000\ 0000$
	$0x\ 00\ 00\ 00\ 02$	$0x1000\ 0004$

	$0x\ 00\ 00\ 00\ 14$	$0x1000\ 0050$
	$0x\ \emptyset\ \emptyset\ \emptyset\ \emptyset$	$0x1000\ 0054$

	$0x\ \emptyset\ \emptyset\ \emptyset\ \emptyset$	$0x1000\ 007C$
	$0x\ 00\ 00\ 00\ 65$	$0x1000\ 0080$
B	$0x\ 00\ 00\ 00\ 66$	$0x1000\ 0084$

	$0x\ 00\ 00\ 00\ 78$	$0x1000\ 00D0$
	$0x\ \emptyset\ \emptyset\ \emptyset\ \emptyset$	$0x1000\ 00D4$

	$0x\ \emptyset\ \emptyset\ \emptyset\ \emptyset$	$0x1000\ 00FC$
	$0x\ 00\ 00\ 00\ 00$	$0x1000\ 0100$
	$0x\ 00\ 00\ 00\ 00$	$0x1000\ 0104$
C
	$0x\ 00\ 00\ 00\ 00$	$0x1000\ 0150$

Notre boutisme est dit

Little-Endian, c'est à dire

que l'adresse désigne l'octet de poids faible et on progresse vers les octets de poids fort.

Nos données sont bien alignées

sur des adresses multiples de

la taille de la donnée ; nous

avons donc :

@ $A[0]$ = $0x1000\ 0000$

@ $B[0]$ = $0x1000\ 0080$

@ $C[0]$ = $0x1000\ 0100$

G3/ Analyse du fonctionnement du cache d'instructions :

g/ Représenter dans le tableau ci-dessous l'état du cache d'instructions à la fin de la 1^{ère} itération de la boucle en supposant que le cache est vide avant l'exécution de la 1^{ère} instruction du programme.

Tag (25 bits)	V	Port 3	Port 2	Port 1	Port 0	N°Empl
0040000	1	Lw	Lw	Addiu	Lui	0
0040000	1	Sw	Bne	Add	Addi	1
0040000	1	?	?	Syscall	Ori	2
	0					3
	0					4
	0					5
	0					6
	0					7

Pour remplir le cache, il faut se baser sur la structure de l'adresse physique définie de l'instruction que l'on veut charger.

f/ Quelles instructions ont déclenché un Miss sur le cache d'instructions pour atteindre cet état ? Quel est l'état du cache à la fin de la 2^{ème} itération ? Qu'en est-il à la fin de la 20^{ème} itération ?

Lors de la 1^{ère} itération, nous avons 3 Miss compulsifs qui nous remplissent le cache, il s'agit des instructions de la colonne "Port 0", à savoir : Lui, Addi et Ori. De plus, comme nous n'avons aucun Miss de conflit lors des 1^{ère} et 2^{ème} itérations, nous aurons que des Hits jusqu'à la fin de la 20^{ème} itération.

g/ Calculez le taux de Miss.

Sur un Mips pipeline normal, on a $\#IFC = 2 + 20 \times 6 + 2$.

On a donc $\text{Taux Miss} = \frac{3}{124} \approx 0,024 \Leftrightarrow 2,4\%$ de Miss

$$\text{Taux Miss} = \frac{\# \text{Miss}}{\# \text{Accès instr.}}$$

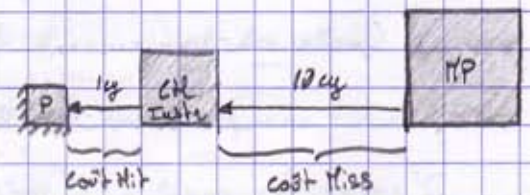
avec $\# \text{Accès} = \# \text{IFC}$

h/ Le coût d'un Miss instruction (nbre de cycles de gel du processeur) est de 10 cycles, en déduire la durée d'exécution du programme (entre le branchement à la 1^{re} instruction du programme et la fin de la dernière instruction), si l'on néglige le coût des Miss sur le cache de données.

Le coût d'un Hit instruction est de 1 cycle, donc à chaque IFC nous avons au moins 1 cycle. Le coût d'un Miss instruction est la durée d'une transaction pour réaliser les transferts de la Mem. Prime vers le cache.

D'après les questions précédentes, on a eu

au total 3 Miss instr, donc la durée d'exécution du programme est de $124 + 3 \times 10 = 154$ cycles.



Q4/ Analyse du fonctionnement du cache de données :

i.sj/ Représentez dans le tableau l'état du cache de données à la fin de la 1^{re} itération de la boucle, en précisant quelles instructions entraînent un Miss sur le cache de données et donc un gel du processeur. Faire de même pour la 20^{ème} itération.

Tag (25 bits)	V	Mot 3	Mot 2	Mot 1	Mot 0	N° Empl	
1 0 0 0 0 0	1	68	67	66	65	0	iter 1 à 4
1 0 0 0 0 0	1	6C	6B	6A	69	1	iter 5 à 8
1 0 0 0 0 0	1	70	6F	6E	6D	2	iter 9 à 12
1 0 0 0 0 0	1	74	73	72	71	3	iter 13 à 16
1 0 0 0 0 0	1	78	77	76	75	4	iter 17 à 20
	0					5	
	0					6	
	0					7	

Pour chaque emplacement : hw R10, 0(R8) provoque un Miss compulsif mais hw R11, 128(R8) provoque un miss de conflit qui écrase la valeur du load précédent. En Winograd les Sw ne font pas de Miss.

k/ Combien y a-t-il de Miss par itération ? En déduire le nombre total de cycles de gel du processeur pour l'exécution complète du programme, en tenant compte du fait qu'un Miss de donnée coûte 2 cycles de plus qu'un Miss d'instruction.

D'après les questions précédentes, un Miss d'instruction a un coût de 10 cycles, donc un Miss de donnée coûte 12 cycles.

A chaque itération, nous avons exactement 2 Miss provoqués par les lw, on a donc un total de $20 \times 2 = 40$ Miss pour 40 accès mémoires.

On a donc un Taux Miss $\text{Data} = \frac{\# \text{Miss}}{\# \text{Accès}} = \frac{40}{40} = 100\%$ de Miss !

l/ Quelle est finalement la durée totale d'exécution du programme, en prenant compte tous les Miss sur les deux caches ? Calculez le CPI.

D'après les questions précédentes, on a 124 IFC sur un Mips normal pipeliné, 3 Miss d'instructions et 40 Miss de données sur tout le programme ; comme un Miss d'instruction coûte 10 cycles et un Miss de donnée en coûte 12, on a :
 $\# \text{Cycles} = 124 + 3 \times 10 + 40 \times 12 = 124 + 510 = 634$ cycles au total.

Donc $\text{CPI} = \frac{\# \text{Cycles}}{\# \text{IFC}} = \frac{634}{124} \approx 5,1$ cycles/instructions.

g6/ Optimisation du fonctionnement du cache de donnée :

m/ Pour cette application, quel pourcentage du temps d'exécution les cycles de gel du processeur du au Miss de conflit sur le cache de données représentent-ils ?

Sur un total de 634 cycles, nous en avons $(40-1) \times 12 = 470$ dus au conflits, on a donc $\frac{470}{634} \times 100 \approx 74,13\%$.

n/ Comment réduire le taux de Miss sur le cache de donnée sans modifier l'architecture matérielle, ni la structure du programme ?

Il suffit de déplacer la structure B d'un bloc vers le haut, c'est à dire faire commencer le tableau B à l'adresse $0 \times 1000 0070$.

o/ Recalculez la durée moyenne du programme avec la proposition faite :

Avec cette proposition, on a 2 Miss de données toutes les 4 itérations donc 10 Miss de données pour toute la boucle $\Rightarrow \#C = 124 + 3 \times 10 + 10 \times 12 = 274$ cy.

Le CPI est donc de $\frac{274}{124} \approx 2,2$ cycles/instructions.