Architecture Avancée
SLE/ISI
Examen de rattrapage

Ensimag 2A

Année scolaire 2016–2017

— Durée : 2h. Tous documents et calculatrices autorisés ;
— Le barème est donné à titre indicatif ;
— Les exercices sont indépendants et peuvent être traités dans le désordre, et certaines questions au sein du même exercice peuvent aussi être traitées indépendamment ;
—

Ex. 1 : Analyse quantitative (3 pts)

Soit les hypothèses suivantes :
— un processeur exécute une instruction RISC par cycle, et lit et écrit une donnée et lit une instruction en 1 cycle si elle est dans le cache (cas *hit*) ;
— la latence de chargement d'une ligne de cache à partir de la mémoire est de $l = 50$ cycles ;
— pour une application donnée, le pourcentage d'instructions de lecture est de 20%, que nous notons $r = 0.20$, et le pourcentage w d'instruction de lecture est 12%, que nous noterons $w = 0.12$;
— les écritures ont lieu dans un cache *write-back* et font *hit* dans le cache dans 65% des cas : $dwh = 0.65$. Dans le cas ou elles font *miss*, il faut dans 27% des cas (*dirty rate* $dr = 27\%$) écrire la ligne en mémoire (ligne modifiée, *dirty*) avant de rapatrier la ligne désirée (*write allocate*). Les écritures qui font *miss* sont synchronisées : on attend le retour de la ligne pour faire effectivement l'écriture avant de passer à l'instruction suivante ;
— le taux de *miss* en lecture drm est de 8% pour le cache de données, et on considère un taux de *miss* nul sur les instructions.

Question 1 (2,5pt) Donnez la formule générale du CPI(l, r, w, dwh, drm, dr) dans ces conditions. Faites l'application numérique avec les valeurs fournies.

$$\text{CPI} = ((1-(r+w))\times 1 + r\times(drm\times l + (1-drm)\times 1) + w\times(dwh\times 1 + (1-dwh)\times(2\times l\times dr + l\times(1-dr))))$$

On multiplie par 2 la taille du cache des données, ce qui fait que les miss rates diminuent, $dwh = 0.75$ et $drm = 0.06$, mais que le hit rate en écriture augmente, donc $dr = 0.42$. En contrepartie, la fréquence de fonctionnement du deuxième système est légèrement plus basse que celle du premier : $f_{s2} = 0.985 \times f_{s1}$.

Question 2 (0,5pt) Est-il utile de faire ce changement ? Justifiez votre réponse.

L'application numérique, que je n'ai pas faite, le dira !

Ex. 2 : Cohérence en Multiprocesseur (7pts)

L'objectif de cet exercice est de définir un protocole de cohérence sur des systèmes à base de caches *write-through*, et d'en esquisser l'implantation.

Soit un système constitué de 2 processeurs avec des caches de données à correspondance directe (*direct mapping*) de 64 lignes de 4 mots ayant une politique d'écriture *write-through* (sans *write-allocate*) et une mémoire interconnectés sur un bus. Les différentes actions qui peuvent être « vues » par une ligne de cache sont : lecture par le processeur qui est connecté au cache (P_r), écriture par le processeur qui est connecté au cache (P_w), lecture sur le bus, *i.e.* lecture effectuée par l'autre processeur (B_r), écriture sur le bus, *i.e.* écriture effectuée par l'autre processeur (B_w). Attention, ces actions s'entendent ligne par ligne et non pas globalement sur le cache.

Question 1 (2pt)

— Rappelez l'état du contenu de la mémoire par rapport au contenu du cache qui fait une écriture (cache *write-through*).

La mémoire est toujours à jour vis-à-vis de la dernière écriture effectuée par le cache, puisqu'elle a lieu dans le cache et dans la mémoire lors de l'action d'écriture (pas au même cycle cependant).

— Dans ces conditions, est-il nécessaire d'avoir une information d'état associé à chaque « ligne » de mémoire [1] ?

Non, car la ligne qui faire référence est la ligne mémoire, elle est donc toujours correcte et si une autre cache veut accéder à cette ligne, les autres caches n'ont pas besoin d'être avertis.

— Quelle est l'unique action qu'il faut espionner pour garantir la cohérence entre les caches ?

L'unique action qu'il est nécessaire d'espionner est l'*écriture* d'une donnée à une adresse qui se trouve cachée. Ceci correspond à l'action B_w du point de vue d'une ligne.

Comme dans le cas *write-back* vu en cours, on doit associer un état à chaque ligne de chaque cache *write-through*. On suppose que l'on veille faire un protocole avec invalidation (*invalidate*) plutôt que mise à jour (*update*).

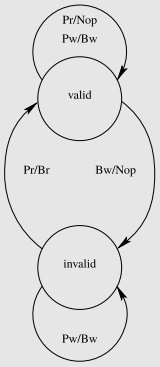
Question 2 (3pt)

— Indiquez les différents états dans lesquels une ligne de cache peut se trouver. Dans un de ces états, le processeur exécute ses instructions normalement : nous appellerons cet état *oisif*.

Les 2 états possibles sont valide ou invalide. Valide signifie que la ligne elle contient la donnée la plus récente, sachant que plusieurs caches peuvent avoir la ligne valide à la différence du *write-back*. La ligne est invalide si elle n'est pas à jour, soit au démarrage, soit à cause d'une transaction B_w due à un autre cache. L'état *oisif* est l'état valide.

1. On appelle « ligne » de mémoire l'ensemble des octets constitutifs d'une ligne de cache qui sont chargés en cache lors d'un *miss*.

— Proposez un protocole, sous la forme d'un automate d'état, qui garantisse la cohérence. Préciser les conditions de transitions entre états, et les actions à effectuer lors de ces transitions. On fait ici explicitement l'hypothèse que les actions issues du processeur et du bus sont mutuellement exclusives, et que, au niveau du protocole, on passe d'un état à un autre en 1 cycle.



On reste en valide tant que le processeur lit, le bus lit ou le processeur écrit. Dans ce dernier cas, on envoie la donnée à écrire qui est visible par les autres caches connectés au bus.
On reste dans l'état invalide tant que le processeur ne fait que des écritures (pas de *write-allocate*) que les autres peuvent espionner. Si le processeur fait une lecture, il lit sa ligne de cache et passe donc de nouveau dans l'état valide.

— Vu l'implantation réel d'un cache, est-il possible de ne passer qu'un cycle dans certains états de votre automate de protocole de cohérence qui conceptuellement ne prendraient qu'un cycle ?

On ne peut passer un unique cycle par état : en valide, on lit le directory pour savoir s'il y a hit ou miss externe. En invalide, on écrit 1 dans le bit de validité pour invalider la ligne. Comme on n'a pas besoin d'échantillonner la sortie de la mémoire pour l'invalidation, on ne perd pas de cycle entre les 2 états. Par contre, pour repasser en valide, il faut recharger la ligne à partir de la mémoire, ce qui prend au moins le nombre de mots de la ligne.

Question 3 (2pt)

— Quel problème se pose si l'automate de cohérence n'est pas dans l'état oisif et qu'une autre action de cohérence a lieu sur le bus et concerne une ligne dans le cache ?

Si l'on est en train d'invalider la ligne en écrivant et qu'un write quelconque passe sur le bus, on a peut être perdu une action nécessitant de la cohérence, et on ne peut donc plus la garantir.

— Peut-on tenter d'éviter d'en tenir compte ? Comment ?

Oui, on peut conserver dans un registre l'adresse de l'invalidation en cours et s'assurer que la nouvelle écriture fait partie de la même ligne, (On ne regarde que les bits de TAG et d'INDEX).
Si c'est le cas, on n'a rien à faire, sinon, il faut lever un drapeau pour le savoir.

— Dans le cas ou l'on ne rentre pas dans le cas ci-dessus, quelle est la *seule* solution pour garantir la cohérence dans ces conditions ? Justifiez.

Il faut invalider la totalité du cache, car on ne sait pas si on a raté une écriture dans le cache ou pas, on n'a pas le choix !

Ex. 3 : Processeur RISC (10 pts)

Le pipeline du processeur Microblaze (de Xilinx, Inc) possède un *pipeline* à 3 étages. C'est ce *pipeline* que nous nous proposons d'étudier dans cet exercice. Le tableau ci-dessous montre le pipeline en fonctionnement optimal.

	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	$t+6$	$t+7$	$t+8$
i	IF	DEC	EXE						
$i+1$		IF	DEC	EXE					
$i+2$			IF	DEC	EXE				
$i+3$				IF	DEC	EXE			
$i+4$					IF	DEC	EXE		
$i+5$						IF	DEC	EXE	

L'étage IF effectue la lecture de l'instruction. L'étage DEC accède le banc de registres et décode l'instruction. L'étage EXE fait le reste, c'est à dire les opérations arithmetiques et logiques, les calculs de branchement, les accès mémoire et l'écriture des résultats dans le banc de registres. Cette stratégie, bien qu'assez peu efficace en terme de fréquence, permet cependant de faire un matériel très compact et d'exécuter une instruction par cycle.

Dans un premier temps, on considère que le jeu d'instruction est identique à celui vu en cours, et on fait l'hypothèse que les ressources ($pc, $ir, $0, $1, ..., $31, etc) sont identiques à celles du R3000.

Question 1 Donnez un schéma bloc (du genre de ceux dessiné en cours) du pipeline idéal. Attention, ce pipeline n'a que 3 étages ! ! !

Question 2 Soit la séquence d'instructions :

```
       ins0
       bne $3,$2, label
       ins2
       ins3
       ins4
label: ins5
```

- **a)** Donnez l'exécution du pipeline dans le cas ou la branche est prise et dans le cas ou la branche n'est pas prise.
- **b)** Qu'en concluez vous par rapport au comportement du MIPS R3000 ?
- **c)** Expliquez en une ligne le pourquoi de ce comportement

Le jeu d'instruction du Microblaze propose 2 types de branchements : le branchement sans *delay slot*, noté `bxx`, ou xx est une condition qui si la branche est prise n'exécute pas d'instrutions après le branchement, et le branchement avec *delay slot*, noté `bxxd`, qui exécute l'instruction qui suit immédiate le branchement avant d'exécuter l'instruction qui est à l'adresse cible du branchement. On suppose par ailleurs que l'on dispose d'un circuit prenant le registre instruction en entrée et générant une sortie br qui vaut 1 sur l'instruction est un branchement et 0 sinon, et une sortie delay qui vaut 1 si on doit exécuter l'instruction qui suit le branchement et 0 sinon.

Question 3 Sachant qu'un code possible de `nop` est `000000000`, ajoutez le matériel nécessaire à la gestion correcte des 2 types de branchements dans votre dessin inital.

Question 4 Soit la séquence d'instructions :

```
add $1,$2,$3
add $6,$4,$1
add $5,$6,$1
```

On sait par ailleurs que l'étage EXE des instructions arithmétique et logique ne prend qu'un cycle.

a) Mettez en évidence, sur la séquence d'instructions présentée ci-dessus, les problèmes de dépendances entre les opérations sur l'ALU.

b) Quelles modifications faut-il apporter au pipeline pour résoudre ces problèmes.

Question 5 Les opérations qui accèdent la mémoire passent 2 cycles dans l'étage EXE. On dispose d'un circuit prenant en entrée le registre `$ir` et donnant un sortie un signal mem qui vaut 1 si l'instruction accède la mémoire et 0 sinon.

a) Proposez une méthode pour que l'exécution des instructions continue à se dérouler comme prévu (pas de perte d'instruction, toujours en séquence, etc).

b) Apportez les modifications au pipeline pour prendre en compte cette latence.

Question 6 Soit la séquence d'instructions :

```
add $1,$2,$3
lw  $3, 0($1)
lw  $4, 0($1)
add $1,$4,$3
```

On rappelle (*c.f.* la question précédente), que les opérations qui accèdent la mémoire passent 2 cycles dans l'étage EXE.

a) Mettez en évidence, sur la séquence d'instructions présentée ci-dessus, les problèmes de dépendances entre les lectures/écritures et les opérations sur l'ALU, et *vice-versa*.

b) Faut-il apporter des modifications au pipeline pour résoudre ces problèmes ? Si oui, lesquelles ?