

TD 4 CORIGE. Les Processeurs actuels. Pipelining

1. Pipelining

1.

Appelons X l'instruction après le LOAD. L'instruction X reste bloquée à l'étage 3 tant que l'accès cache du LOAD n'est pas terminé, c'est-à-dire tant que le LOAD n'a pas dépassé l'étage 6.

Juste avant que le LOAD rentre dans l'étage 7, le résultat du LOAD est envoyé sur le réseau de *bypass* et l'instruction X va pouvoir rentrer dans l'étage 4. A cet instant précis, les étages 4 et 5 ne contiennent aucune instruction. 2 bulles ont donc été introduites.

2.

La boucle itère un grand nombre de fois parce que la valeur initiale de R4 est grande. Donc on peut raisonner comme si la boucle itérait un nombre infini de fois et négliger le temps de remplissage du pipeline. Le corps de la boucle comporte 5 instructions. La 2^{ème} instruction dépend de la 1^{ère}, qui est un LOAD. 2 bulles sont donc générées toutes les 5 instructions.

Le débit d'exécution vaut : $\frac{5}{5+2} = \frac{5}{7} \approx 0.71$ instructions / cycle.

3.

Dans le programme initial, l'instruction 2 qui suit immédiatement un LOAD, utilise le résultat du LOAD, ce qui introduit 2 bulles dans le pipeline, bulles qu'il faut supprimer pour ramener le débit d'exécution à 1 instruction / cycle.

Pour supprimer les 2 bulles, il faut placer 2 instructions indépendantes du LOAD juste après le LOAD, comme ceci :

```

1:   boucle: R2 = LOAD R1+0
2:           R1 = R1 ADD 4
3:           R4 = R4 SUB 1
4:           R3 = R3 ADD R2
5:           BNZ R4, boucle

```

4.

a) Pour supprimer les 2 bulles introduites par un LOAD, il faut faire suivre ce LOAD de 2 instructions indépendantes du résultat du LOAD. Ici, le LOAD n'est suivi que d'1 seule instruction indépendante du résultat du LOAD, ce qui introduit 1 *bulle* dans le pipeline.

Le débit d'exécution du programme, comportant 4 instructions et coûtant 5 cycles, est donc de :

$\frac{4}{4+1} = \frac{4}{5}$ instructions / cycle.

b) Pour comparer 2 programmes différents effectuant le même travail, il ne faut pas regarder le débit en instructions par cycle mais le temps total pour effectuer le travail.

Ici, on peut se contenter de comparer le nombre de cycles par itération, puisque chaque itération traite un élément du tableau.

Avec le programme de la question 2, chaque itération coûte 7 cycles.

Le nouveau programme ne comporte que 4 instructions au lieu de 5. 1 bulle est générée à chaque itération. Chaque itération coûte donc 5 cycles. Le nouveau programme est plus performant que celui de la question 2 et aussi performant que celui de la question 3 qui compte 5 cycles aussi mais sans *bulle*.

5.

Rappel :

<i>étape</i>	<i>pipeline P1</i>
1	lit cache d'instructions
2	decode instruction
3	lit registres
4	execute / calcul adresse
5	acces cache de donnees
6	acces cache de donnees
7	ecrit registre

<i>étape</i>	<i>pipeline P2</i>
1	lit cache d'instructions
2	decode instruction
3	lit registres
4	calcul adresse
5	acces cache de donnees
6	execute / acces cache de donnees
7	ecrit registre

. **Pipeline initial (P1) :** 2 bulles sont introduites si le résultat de l'exécution d'une instruction LOAD/STORE dépend de l'instruction précédente.

. **Nouveau Pipeline (P2) :** 2 bulles sont introduites avec le calcul d'adresse pour chaque opérande d'une instruction LOAD/STORE dépendant de l'instruction précédente.

Si les 2 instructions précédant un LOAD/STORE sont indépendantes du calcul d'adresse de ou des opérandes de l'instruction LOAD/STORE, aucune bulle n'est introduite.

Dans le cas du programme de la question 4, le calcul d'adresse du LOAD dépend de R1 et de R4. Seul R4 est mis à jour dans la boucle par l'instruction SUB. Comme il y a 2 instructions intercalées entre le SUB et le LOAD, aucune bulle n'est introduite ici. Chaque itération coûte 4 cycles sur le nouveau pipeline (P2), au lieu de 5 cycles sur l'ancien pipeline (P1).
