TD 2

Parallélisme au niveau instruction et pipeline

L'objectif de ce TD est de concrétiser les notions de dépendance de donnée (RAW, WAR, WAW, vraie et fausse) et de dépendance de ressource dans le cas d'un processeur pipeliné.

1 Échauffement

On considère le morceau de programme patassembleur suivant :

```
1:      R3 = R1 * R1;
2:      R4 = R0 * R2;
3:      R4 = R4 * 4;
4:      R0 = R3 - R4;
5:      R2 = SQRT R0;
6:      R3 = -R1 - R2;
7:      R4 = R2 - R1;
```

Question 1-1 Quelle est la sémantique de ce morceau de code ? Quelle est la sémantique des lignes 3 à 6 seulement ? Que deviendrait sans doute la première sémantique si l'on considérait le code qui suit la ligne 7 ?

Question 1-2 Si vous ne l'avez pas encore fait, donnez toutes les dépendances entre les variables et précisez leur type. Construisez le DAG pour ce programme. Au fait, pourquoi est-il acyclique ?

Question 1-3 Comment voit-on dans le DAG qu'on peut changer l'ordre de deux instructions ou les exécuter en parallèle ?

Question 1-4 Proposez des parallélisations de ces instructions en supposant un nombre d'additionneurs et de multiplieurs illimité. Vous supposerez que toutes les opérations prennent chacune un cycle.

Question 1-5 Même question avec deux multiplieurs à quatre cycles non pipelinés, deux additionneurs en un cycle, et une boite-à-SQRT en 6 cycles non pipelinés.

Question 1-6 Gagne-t-on sur ce code à pipeliner les multiplieurs ?

Question 1-7 Renommez les registres pour supprimer les fausses dépendances, et reprenez les quelques questions précédentes.

Question 1-8 A votre avis, une fenêtre de code de taille 7 comme dans cet exemple est-elle adaptée pour faire du réordonnancement de code dans un processeur ? Dans un compilateur ?

Les "vraies" réponses seront données plus tard en archi avancées et en compilation...

2 Des choses sérieuses

On considérera dans cette partie le programme suivant :

```
1            R0 = xxxx    // adresse du vecteur X
2            R1 = yyyy    // adresse du vecteur Y
3            R2 = nnnn    // taille des vecteurs
4            R3 = 0

5  .boucle   R10 = [R0]
6            R11 = [R1]
7            R12 = R10 * R11
8            R3  = R3 + R12
9            R0 = R0 + 4
10           R1 = R1 + 4
11           R2  = R2 - 1
12           BNZ boucle
```

Un processeur pipeliné comme il y a dix ans

On dispose d'un processeur pipeliné sur cinq étages : *fetch, decode, operand load, execute/mem, operand writeback*. Ce processeur possède des caches de données et d'instructions séparées (pas de conflit de ressource entre *fetch* et *mem*). Dans ce TD, on suppose que l'ALU fait une addition ou une multiplication en un cycle, on se demande comment.

Question 2-1 Dessiner sommairement ce processeur (ALU, banc de registres et latches).

Table de réservation du pipeline

Question 2-2 Pour le moment, on n'a pas de court-circuit dans le pipeline. Dessinez la table de réservation d'une itération de la boucle ci-dessus (lignes 5 à 11), en prenant garde aux dépendances de données (il n'est pas interdit de faire comme dans la première partie). Étudiez les réordonnancements que peut faire le compilateur. Quelle est la latence d'une itération dans le meilleur cas ?

Le problème des boucles

En arrivant au `BNZ`, le processeur doit décider avec quelle branche il va remplir son pipeline (et parfois il se trompera). Cela s'appelle la prédiction de branchement, et un jour on traitera cette question en détail.

Question 2-3 Au vu de cette boucle, quel choix implantez-vous dans le processeur ?

Question 2-4 Compléter la table de réservation de la question précédente en y incluant le branchement.

Question 2-5 Y-a-t il des dépendances d'une itération sur l'autre ? Peut-t-on faire de nouveaux réordonnancements encore plus meilleurs ?

Question 2-6 Calculez le CPI moyen de N itérations en fonction de N.

Court-circuits

Question 2-7 Quels court-circuits raisonnables pouvez-vous introduire dans le pipeline pour diminuer la latence de cette boucle ? Pouvez-vous estimer leur coût matériel ?

Du boulot de compilateur optimiseur

Question 2-8 Déroulez la boucle une fois en supposant que $nnnn$ est connu à la compilation et pair. Comparez les tailles de code et discutez.

Question 2-9 Reprenez les questions précédentes.

Vraies-fausses dépendances de contrôle

Question 2-10 Vous avez sans doute oublié une dépendance entre les instructions 11 et 12 : la dépendance de donnée sur les drapeaux. Si maintenant notre addition entière est pipelinée (mettons sur 2 cycles), on a une bulle inévitable dans le pipeline. Une solution est de spécifier que les drapeaux utilisés par BNZ sont ceux mis à jour deux instructions auparavant. Pourquoi est-ce inélégant ? Comment modifier plus fondamentalement le jeu d'instruction pour arranger cela ?

Question 2-11 Lors d'une exception (ou interruption, ou faute, ou ...), le pipeline doit être vidé et l'état du processeur sauvé. À quel étage peut se produire une exception ? Décrivez le matériel qui gère le traitement de l'exception.

Un peu de techno

Question 2-12 Quels sont les facteurs qui limitent le nombre d'étages de pipeline ? Lisez les deux papiers distribués, et réjouissez-vous du consensus à ce sujet.