Architecture, Système, Réseau 2017-2018
{alice.pellet___mary,fabrice.mouhartem}@ens-lyon.fr, florent.de-dinechin@insa-lyon.fr

TD n° 9 : Pipeline et parallélisme au niveau instruction

1 Échauffement

On considère le morceau de programme patassembleur suivant :

```
1:      R3 = R1 * R1;
2:      R4 = R0 * R2;
3:      R4 = R4 * 4;
4:      R5 = R0 * 2;
5:      R0 = R3 - R4;
6:      R2 = SQRT R0;
7:      R3 = -R1 - R2;
8:      R4 = R2 - R1;
9:      R3 = R3 / R5;
10:     R4 = R4 / R5;
```

Question 1 Quelle est la sémantique de ce morceau de code ?

Question 2 Si vous ne l'avez pas encore fait, donnez toutes les vraies dépendances entre les variables (*read after write* ou *RAW*).

Question 3 On appelle graphe acyclique dirigé (DAG en grand-breton) d'un morceau de code le graphe dont
— les noeuds sont soit les opérations du code, soit des valeurs en entrée (par exemple dans des registres)
— les arcs sont les vraies dépendances entre les opérations ou les feuilles.
Dessinez le DAG correspondant au programme précédent. Pourquoi est-il acyclique ?

Question 4 Comment voit-on dans le DAG qu'on peut changer l'ordre de deux instructions ou les exécuter en parallèle ?

Question 5 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 6 (pour ceux qui vont vite) Dessinez les fausses dépendances (WAW et WAR). Empêchent-elles la parallélisation précédente ?

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
```

2.1 Un processeur pipeliné comme il y a dix ans

On dispose d'un processeur pipeliné sur cinq étages : *instruction fetch (IF), instruction decode (ID), operand load (OL), execute/mem (EX), writeback (WB)*. 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 n'a pas de court-circuit dans le pipeline : une dépendance RAW est une dépendance de l'étage WB vers l'étage OL.

Question 7 Dessiner sommairement ce processeur (ALU, banc de registres et registres de pipeline).

2.2 Chronogramme du pipeline

On s'intéresse pour l'instant à la première itération.

Question 8 Dessinez le chronogramme vu en cours pour un pipeline (en x, les cycles ; en y, les étages du pipeline ; dans chaque case, le numéro de l'instruction traitée à cet instant dans cet étage). Prenez garde aux dépendances de données (il n'est pas interdit de faire comme dans la première partie).

Question 9 Étudiez les réordonnancements que peut faire le compilateur. Quelle est la latence (nombre de cycles) d'une itération dans le meilleur cas ?

2.3 Le problème des boucles

En arrivant au `BNZ`, le processeur peut buller.

Un meilleur choix est de *spéculer* : on fait un pari sur la branche qui va être prise, on remplit le pipeline avec... et parfois on se trompe. Dans ce cas il faudra annuler des instructions lancées à tort, c'est-à-dire les remplacer par des bulles dans le pipeline.

Question 10 Il faut donc faire un pari (si vous voulez creuser cette question, cela s'appelle la *prédiction de branchement*). Au vu de notre boucle, quelle prédiction implanteriez-vous dans le processeur ?

Question 11 Compléter le chronogramme de la question précédente en y incluant le branchement et l'itération suivante.

Question 12 Y-a-t il des dépendances d'une itération sur l'autre ? Peut-t-on faire de nouveaux réordonnancements encore plus meilleurs ?

2.4 Un cas de conscience pour le compilateur optimiseur

Question 13 Déroulez la boucle en dupliquant une fois le corps de boucle. On supposera que $nnnn$ est connu à la compilation et pair (pourquoi ?).

Question 14 Comparez les tailles de code, et le temps d'exécution global, et discutez.