TD 6

Exécution spéculative
Résumé des épisodes précédents

1 Exécution spéculative par le tampon de remise en ordre

On s'est jusque là débarassé des dépendances de données vraies puis fausses. Reste plus que les dépendances de contrôle : devant un branchement conditionnel, on ne veut pas devoir buller en attendant de savoir quelle branche il faut prendre. On veut

- soit prédire une des branches (on verra comment un jour) et se lancer dedans, mais être capable en cas d'erreur de défaire tout ce qu'on a fait depuis le branchement.
- soit même, dans le doute, se lancer dans les deux branches, et pouvoir annuler par la suite l'une des deux.

Il paraît clair que si on se donne la mécanique pour faire cela, cela nous aidera aussi à assurer les exceptions précises.

Or il suffit pour cela d'ajouter un petit gadget à notre architecture Tomasulo : le tampon de réordonnancement (ROB pour *ReOrdering Buffer*). L'idée centrale derrière le ROB est qu'on se permet d'exécuter les calculs dans le désordre, mais qu'on écrira le résultat de chaque instruction (en registre ou en mémoire) dans l'ordre du programme. Et on ne le fera que lorsqu'on sera certain que l'instruction ne sera pas annulée par une interruption ou un branchement planté.

Le ROB se place entre le bus commun et les registres, et contient une entrée par instruction *dans l'ordre du programme*. Chaque entrée stockera la valeur calculée par cette instruction entre le moment où elle sort de son unité de calcul et le moment où on peut la ranger en registre ou en mémoire avec la conscience tranquille : nous voilà avec une étape de plus dans la vie d'une instruction, la terminaison *(commit)* qui consiste à écrire la donnée en registre ou en mémoire.

Question 1-1 Refaites le dessin du processeur du TD précédent, avec un ROB. Définissez précisément les structures de données utilisées. En principe, tous les pointeurs qu'on avait vers les stations de réservation sont remplacés par des pointeurs vers des entrées du ROB.

Question 1-2 Quand et comment *committe*-t-on une instruction ?

Question 1-3 Comment détecte-t-on qu'on s'est trompé de branche ? Que fait-on alors ?

Question 1-4 Cette mécanique permet-elle d'exécuter spéculativement les deux alternatives d'un branchement ?

Question 1-5 Comment se passe une écriture en mémoire ?

Question 1-6 Montrez que cela permet de maintenir les exceptions précises.

2 On récapépète

On part de la photocopie du PPC 620 tirée du CAQA. On va détailler l'architecture et le fonctionnement de ce type de processeur, incluant

- le décodage et le lancement de jusqu'à 4 instructions de 32 bits par cycle,
- le renommage des registres (pas comme on a vu),
- l'exécution dans le désordre avec cohérence séquentielle (forte).

Il y a 6 unités fonctionnelles, chacune avec ses stations de réservation. Les caractéristiques de chaque unité sont les suivantes.

- Les deux UAL entières (32 bits) simples XSU (+,- et opérations logiques) ont une latence d'un cycle.
- L'UAL entière complexe MCFSU réalise soit des multiplications, entièrement pipelinées mais à latence variable (entre 3 et 20 cycles suivant les opérandes), soit une division non pipelinée en 20 cycles.
- La FPU pipelinée a une latence de 2 cycles pour multiplication, addition ou addition/multiplication, et une latence de 31 cycles pour une division. Les opérandes flottants sont en double précision (64 bits).
- L'unité d'accès mémoire réalise un chargement entier en un cycle et un chargement FP en deux cycles pipelinés si la donnée est présente dans le cache. Sinon elle contient des tampons de chargement qui conservent l'adresse jusqu'à ce que la donnée arrive. Pour les écritures, on passe pour le moment.

La gestion du renommage se fait en étendant la file des registres physiques avec un certain nombre de registres de renommage. L'avantage est que l'accès à un registre est uniforme (donc simple et rapide). De plus cela évite d'avoir à interroger le ROB au lancement d'une instruction dont un opérande a déjà été calculé mais pas committé. Les registres de renommage doivent être recopiés dans les registres architecturaux lors du *commit*. Le P4 fait cela aussi, au fait.

Voici les différentes phases de l'exécution.

- Fetch, puis decode de (jusqu'à) 4 instructions en parallèle.
- Émission par la *dispatch unit*. Lors de l'émission, chaque intruction est envoyée dans la station de réservation correspondante, un registre de renommage est créé, et une entrée est créée dans le ROB.
- Lancement : l'instruction libère son rayon au lancement. À la fin de son exécution elle écrit son résultat sur le bus résultat, où les SR peuvent le lire.
- Commit : jusqu'à quatre par cycle. Les registres de renommage sont recopiés dans les registres standards et libérés

Question 2-1 Les tailles des différents bus manquent cruellement au schéma qui vous est fourni. Essayez de les placer (le crayon à papier s'impose) et discutez de l'arbitrage des différentes connexions.

Question 2-2 Discutez la taille des différentes stations de réservation.

Question 2-3 Pourquoi le renommage ? Discutez le nombre des registres de renommages.

Question 2-4 Discutez la taille du ROB.

Question 2-5 Détaillez les structures de données maintenues par les stations de réservation, les files de registres, la ROB, etc. Faites transiter toutes ces informations sur les différents bus.