TD 4

Nous sombrons dans le superscalaire

On parle de processeur superscalaire lorsque
- le jeu d'instruction a une sémantique séquentielle (par opposition, par exemple, au VLIW) ;
- l'architecture dispose de plusieurs unités fonctionnelles (UF) qui peuvent fonctionner en parallèle ;
- une mécanique compliquée fait son possible pour exploiter ce parallélisme au maximum tout en maintenant l'illusion que la sémantique séquentielle est respectée *(cohérence séquentielle)*.

Dans ce TD nous allons essayer d'inventer les structures de donnée et la mécanique nécéssaires au fonctionnement d'un processeur superscalaire.

Sur la fin ce sera nécessairement un peu confus et plein de ratures. Dans le TD suivant, on repartira sur des bases saines avec une mécanique parachutée, et on se contentera de prouver qu'elle fonctionne (ou de la bricoler jusqu'à ce qu'on puisse prouver qu'elle fonctionne).

1 Pipeline simple, lancement dans l'ordre

Pour que les exemples soient les mêmes que dans le CAQA, on considère ici qu'on a une unité de chargement mémoire pipelinée **load** (en deux cycles quand tout va bien), une unité d'addition pipelinée (en deux cycles) **add**, une unité de multiplication pipelinée (en 10 cycles) **mul**, et une unité de division (en 40 cycles) **div**.

Unité	latence	pipeline
FPadd	2	oui
FPmul	10	oui
FPdiv	40	non
LD/STR	2	oui

Au vu de ce tableau, il est clair que les résultats peuvent arriver dans le désordre même si les opérations ont été lancées dans l'ordre. Que faut-il faire pour que les exceptions soient précises [1] ?

Proposez une architecture qui lance les instructions dans l'ordre, et gère les dépendances pour maintenir la cohérence séquentielle si elles terminent dans le désordre. Intéressez-vous à la complexité de cette architecture en fonction de la taille du pipeline, du nombre de registres, etc.

Désordre pour désordre, on veut à présent permettre le *lancement* des instructions dans le désordre. Que faut-il surveiller en plus du cas précédent ?

Dans les processeurs comme à l'Éducation Nationale, il y a deux approches pour gérer le désordre : une approche centralisée, et une approche décentralisée.

[1] Et au fait qu'est-ce qu'une exception précise ?

2 Rayonnage centralisé (table de réservation, *scoreboard*, ...)

L'exécution d'une instruction passe par les étapes suivantes, sous la surveillance d'un contrôle centralisé (*scoreboard* ou "" ou "rayonnage centralisé" ou ...).

- *Instruction fetch* : les instructions flottantes sont placées dans un tampon d'instructions, dans l'ordre dans lequel elles arrivent. Elles en sortent dans l'ordre.
- *Émission* : les instructions flottantes sont placées dans un rayonnage (un genre de table) où elles attendront de pouvoir s'exécuter. Qu'attendent-t-elles au juste ?
- *Lancement (issue)* : le rayon envoie les instructions qui peuvent s'exécuter dans les unités de calcul.
- *Exécution* : l'unité fonctionnelle fait son boulot. Quand le résultat est prêt elle en informe le rayon.
- *Écriture du résultat* : le rayon vérifie l'absence de dépendance (lesquelles ?) et bloque l'écriture s'il le faut.

Lecture des opérandes

Un opérande (registre) est disponible si aucune instruction lancée précédemment ne va l'écrire, ou si ce registre est en cours d'écriture par une unité fonctionnelle.

Vaut-il mieux faire la lecture des opérandes à l'émission ou au lancement ? Discutez.

On pourra considérer le bout de code suivant :

```
1            F0  = FDIV F2, F4
2            F10 = FADD F0, F8
3            F8  = FSUB F8, F14
```

Architecture sommaire

Dessinez les blocs fonctionnels pour cette unité flottante. Discutez des tailles des bus, etc.

Structures de données de la table de réservation

Définissez les structures de données nécessaires au rayonnage et au banc de registres pour faire leur travail.

Complexité de l'architecture

Étudiez la taille des différents comparateurs, bus, etc.

Fausses dépendances

Que se passe-t-il en cas de fausse dépendance de donnée ?

Fausse dépendance ? tilt ! renommage !

Proposez une technique de renommage en utilisant plus de registres matériels que n'en possède le jeu d'instruction. En quoi cela modifie votre architecture ?

3 Gestion décentralisée (Tomasulo, stations de réservation)

Cet algorithme présente quelques importantes différences avec le précédent :
- la gestion des dépendances est décentralisée dans des stations de réservation associées aux unité fonctionnelle ;
- Un bus commun relie les entrées et sorties des stations de réservation et celles du banc de registre.
- ces stations de réservation réalisent également le *renommage* des registres pour supprimer les fausses dépendances : elles contiennent d'abord les numéros de registres des opérandes, puis les opérandes eux-mêmes. En d'autres termes les registres de renommage sont à présent contenus dans les stations de réservation.

Les étapes d'exécution de chaque instruction sont alors les suivantes :
- *Lancement (issue)* : prendre une instruction dans la queue d'instructions FP. S'il y a une station de réservation libre, la placer dedans. C'est dans cette étape que les registres sont renommés. Remarquez qu'on peut lancer plusieurs instructions par cycle.
- *Exécution* : tant qu'il manque un opérande, surveiller le bus de données commun jusqu'à le voir passer. Lorsque les deux opérandes sont présents, exécuter. C'est dans cette étape que les aléas RAW sont gérés.
- *Écriture du résultat* : lorsque le résultat est disponible, l'écrire sur le bus de données commun.

Reprendre les questions de la partie précédente.