**TD 5**

# *Exécution superscalaire*

## 1 Gestion centralisée

### 1.1 Ce qu'on aurait du trouver au TD précédent

Chaque instruction passe par 4 états :

**Lancement** Si une UF est libre pour cette opération, et si aucune instruction déja présente dans la table n'a le même registre de destination (pour éviter les WAW), l'instruction est entrée dans la table. Les instructions sont lancées dans l'ordre : si le lancement n'est pas possible, le tampon d'instructions en attente de lancement se remplit...

**Lecture des opérandes** La table attend que tous les opérandes soient disponibles (dans le banc de registres). Quand c'est le cas, l'exécution est lancée. C'est à cette étape qu'est garanti le respect des dépendances RAW.

**Exécution** L'UF commence son boulot. Lorsqu'elle a terminé (après un nombre de cycles variables) elle prévient la TR.

**Écriture du résultat** La table attend que des éventuelles dépendances WAR soient résolues (par la lecture des opérandes nécessaires) avant de d'écrire le résultat dans le banc de registres.

Les structures de données maintenues par la table de réservation sont les suivantes, pour une opération **Fd = Fa Op Fb** :

**État des instructions**

| Instruction | Lancée | Op. lus | Exécutée | Rés. écrit |
|---|---|---|---|---|
| **F1 = [...]** | | | | |
| **F0 = F1 * F2** | | | | |
| **F4 = F3 - F1** | | | | |
| **F5 = F0 / F3** | | | | |
| **F3 = F4 + F2** | | | | |

Cette table est un tampon circulaire avec un certain nombre de lignes (sur la fin nous discuterons ce certain nombre). Elle est remplie par l'étage *Instruction Fetch*.

**État des unités fonctionnelles**

| nomUF | Busy | Op | d | a | Oa | b | Ob | Ra | Rb |
|---|---|---|---|---|---|---|---|---|---|
| **load** | | | | | | | | | |
| **mul** | | | | | | | | | |
| **add** | | | | | | | | | |
| **div** | | | | | | | | | |

Le champ **O** indique une UF qui va produire ce registre lorsque son calcul est en cours. Le champ **R** est un booléen qui indique que les registres sont prêts mais pas encore lus.

**Va être produit par**

| **n° Reg** | 0 | 1 | 2 | 3 | 4 | 5 | ... | 31 |
|---|---|---|---|---|---|---|---|---|
| **UF** | | | | | | | | |

Cette information est la transposée de la colonne **d** de la table précédente.

### 1.2 Des équations

Donnez, pour une instruction **Fd = Fa Op Fb**, en fonction de son état dans la première table, la condition que la TR attend pour faire passer cette instruction dans l'état suivant, ainsi que les mises à jour à faire dans les différentes tables.

### 1.3 Du boulot d'ordinateur

Simulez sur la feuille jointe l'exécution de la séquence d'instructions présentes dans la table.

### 1.4 Des preuves avec les mains

Montrez que les trois types de dépendances sont respectées par ce fonctionnement. Sinon, recommencez.

### 1.5 Du matériel

Dessinez les blocs de cette architecture, en précisant les tailles des bus.

Vous avez du faire une supposition sur le nombre de registres que l'on peut lire et écrire dans le banc de registre à chaque cycle. Que faire s'il ne peut par cycle sortir que deux registre et en entrer un ?

### 1.6 Limitations

Essayez de voir où sont les difficultés dans une architecture à *scoreboard*, et les limitations causées par les différents paramètres de tailles.

# 2 Tomasulo

Le défaut du *scoreboard*, outre qu'il est centralisé, est de buller en présence de fausses dépendances. Or on se débarasse des fausses dépendances en renommant les registres. Le principe de l'algorithme de Tomasulo est de réaliser l'équivalent du renommage, en remplaçant dans les rayonnages l'information sur les registres opérandes par

– le contenu de ces registres, ou bien
– un pointeur vers qui va les calculer si le calcul est en cours.

Il y a désormais plusieurs rayonnages par unité fonctionnelle, il faudra discuter combien. Et on les appelle des stations de réservation.

## 2.1 Ce qu'on aurait du trouver au TD précédent

Les tables sont donc à présent les suivantes :

**État des instructions**

C'est une table virtuelle, qui est en fait délocalisée dans les stations de réservation.

| Instruction | Lancée | Exécutée | Rés. écrit |
|---|---|---|---|
| **F1 = [...]** | | | |
| **F0 = F1 * F2** | | | |
| **F4 = F3 - F1** | | | |
| **F5 = F0 / F3** | | | |
| **F3 = F4 + F2** | | | |

**Stations de réservation**

| nom | Busy | Op | Va | SRa | Vb | SRb | A |
|---|---|---|---|---|---|---|---|
| **load1** | | | | | | | |
| **load2** | | | | | | | |
| **add1** | | | | | | | |
| **add2** | | | | | | | |
| **add3** | | | | | | | |
| **mul1** | | | | | | | |
| **mul2** | | | | | | | |
| **div** | | | | | | | |

**Produit Par**

| n° Reg | 0 | 1 | 2 | 3 | 4 | 5 | ... | 31 |
|---|---|---|---|---|---|---|---|---|
| **SR** | | | | | | | | |

Notez que cette table n'est plus redondante comme dans le cas précédent.

Un bus commun relie toutes les SR ainsi que le banc de registres, et chaque instruction passe par trois états :

**Lancement** Si une SR est libre pour cette opération, l'instruction est envoyée à cette SR. Si les opérandes sont dans les registres, ils sont envoyés aussi dans le rayon correspondant. Sinon (une autre SR est en train de les calculer), on lit dans la table des registres résultats de quelle SR il s'agit, et on recopie cette information dans le champ **SR** correspondant

**Lecture des opérandes et exécution** Chaque SR surveille ce qui passe sur le bus commun. Lorsqu'elle voit passer une donnée dont elle a besoin, elle la place dans le champ **Va** ou **Vb**. Dès que les deux valeurs sont présentes, l'exécution est lancée.

**Écriture du résultat** L'unité fonctionnelle qui a fini envoie son résultat dans le banc de registres (et tout le monde le voit passer).

### 2.2 Du matériel

Dessinez les blocs de cette architecture, en précisant les tailles des bus. Discutez du traffic sur le bus commun.

### 2.3 Du boulot d'ordinateur

Simulez sur la feuille jointe l'exécution de la séquence d'instructions de la première partie. Qui gagne, et pourquoi (ce n'est pas forcément significatif) ?

### 2.4 Les accès mémoire

Comment s'insèrent les chargements et écritures mémoires dans cette architecture ? Vous en reprendrez bien une louche avec le programme suivant :

```
Loop:  F0   = [R1]
       F4   = F0 * F2
       [R1] = F4
       R1   = R1 - 8
       BNEZ Loop
```

Qu'observez-vous avec stupeur et ravissement ?

### 2.5 Des équations

Donnez, pour une instruction **Fd = Fa Op Fb**, en fonction de son état, la condition que la SR attend pour faire passer cette instruction dans l'état suivant, ainsi que les mises à jour à faire dans les différentes tables.

### 2.6 Des preuves avec les mains

Montrez que les trois types de dépendances sont respectées par ce fonctionnement. Sinon, recommencez la question précédente.