Exécution en ordre partiel

Une fois les instructions renommées, les seules dépendances qui subsistent entre instructions registre-registre sont les dépendances LAE.
L'ordre d'exécution est l'ordre partiel induit par ces dépendances.
Une instruction ne peut démarrer son exécution tant que ses sources ne sont pas prêtes.

Exécution en ordre partiel

Les sources d'une instruction registre-registre **ADD RR0, RR1, RR2** sont produites par les instructions registre-registre antérieures les plus récentes ayant **RR1** et **RR2** pour destination.

Ces instructions peuvent être:

- -> terminées,
- -> en cours de calcul,
- -> pas encore démarrées.

Source prête ou en attente

Si l'instruction produisant la source est terminée, celle-ci se trouve dans le banc de registres de renommage.
Si l'instruction est en cours de calcul, la source se trouve dans le pipeline de l'opérateur.
Si l'instruction n'est pas démarrée, la source sera produite **c** cycles après le démarrage par un opérateur de latence **c** (les chargements de la mémoire ont une latence variable).

Etat des sources en registre

Après son renommage, une instruction ADD RR0, RR1, RR2 lit l'état des registres RR1 et RR2:

-> registre plein: la source est prête,

-> registre vide: la source n'est pas prête.

Une source constante est toujours prête.

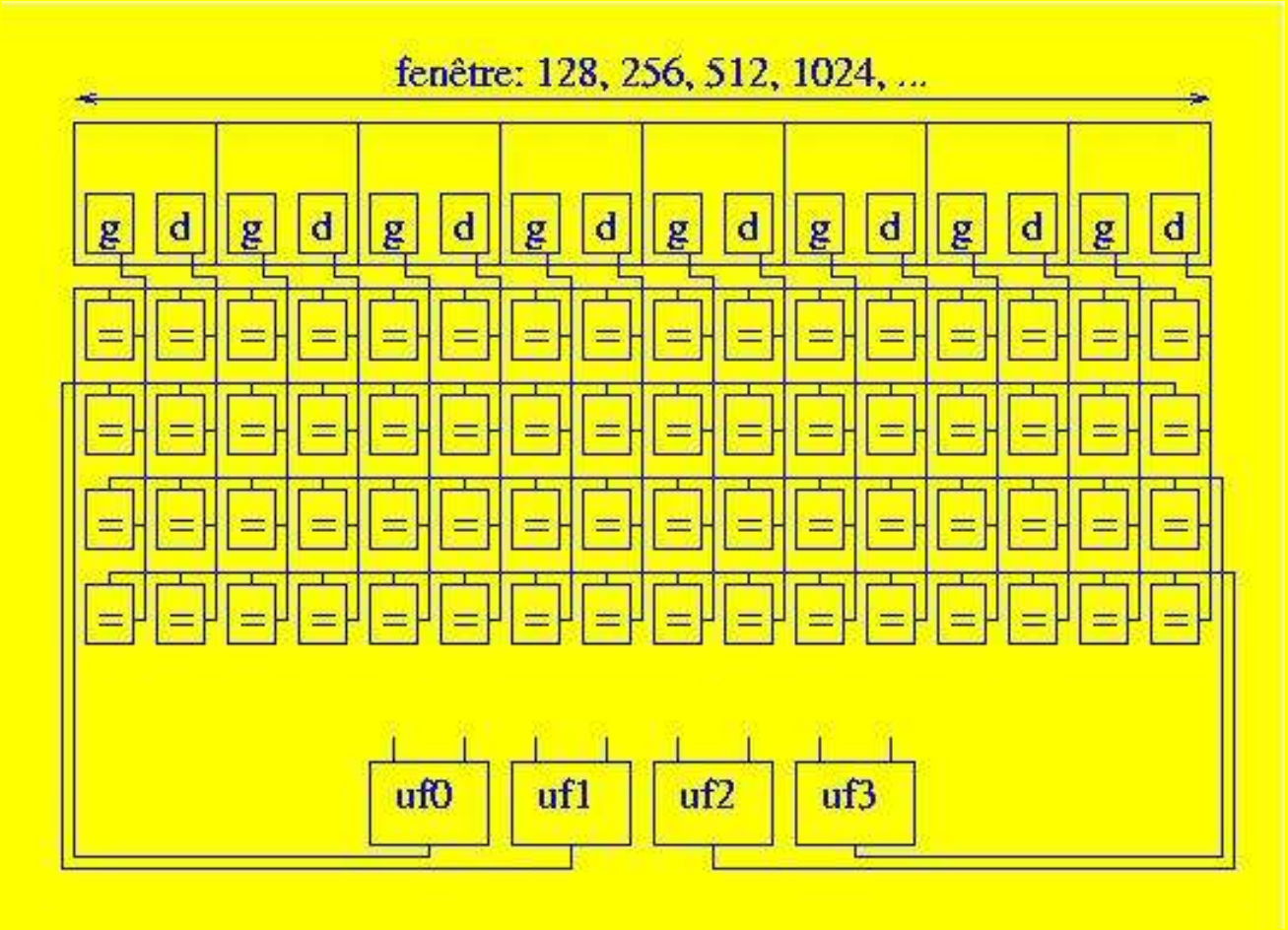
L'instruction est dirigée vers son opérateur où elle est rangée dans une table jusqu'à ce que ses deux sources soient prêtes.

Envoi d'étiquette

Dans sa station d'attente, l'instruction attend, pour chaque source non prête, un signal indiquant que sa valeur est établie. L'opérateur qui produit la valeur de RR1 envoie l'étiquette RR1 à toutes les stations d'attente. Celles qui attendent RR1 peuvent marquer leur source prête.

Le réveil des instructions

O(2*f*u)

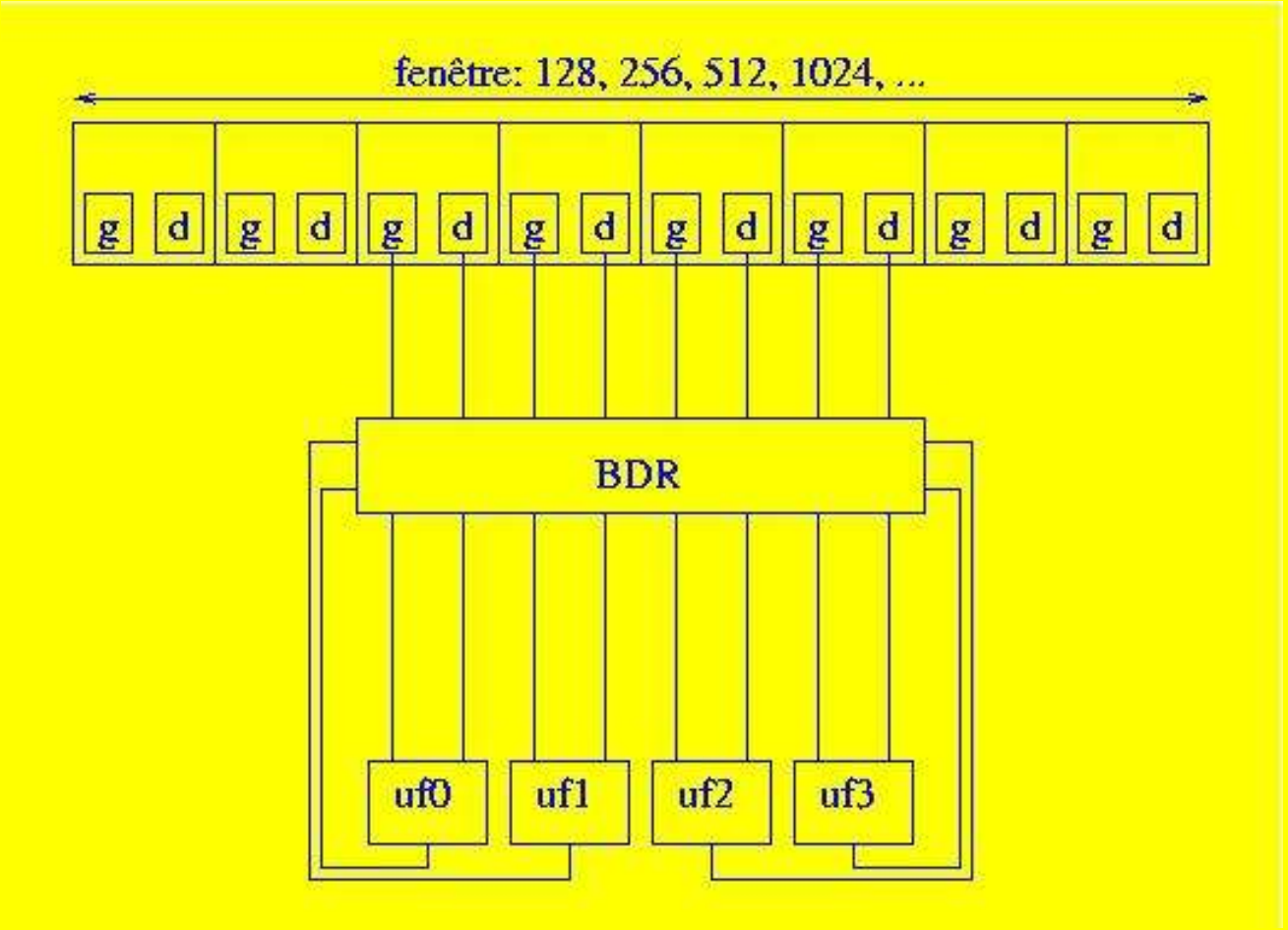


Réveil et démarrage

Quand ses deux sources sont prêtes, l'instruction est candidate au démarrage.
A chaque cycle (pour un opérateur pipeliné), l'opérateur choisit une instruction candidate parmi celles de sa table de stations d'attente.
L'instruction élue lit ses deux sources dans le banc de registres de renommage et entre dans le pipeline de l'opérateur.

La surface du banc de registre

$O(f*(3u)^2)$



Envoi de valeur

Dans le cas suivant:

ADD RR0, RR1, RR2 (1)

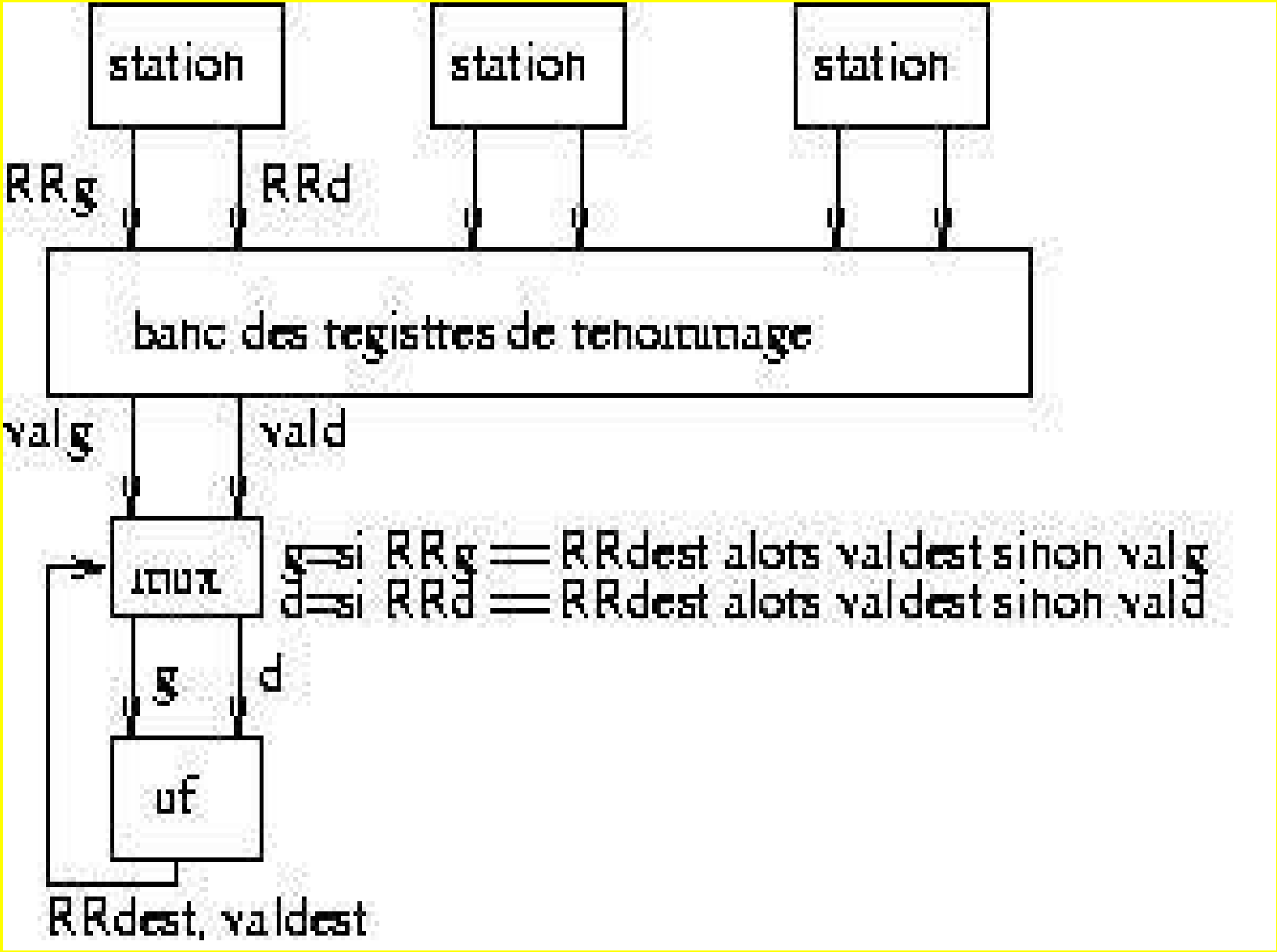
ADD RR3, RR0, RR1 (2)

la valeur de la source RR0 peut être envoyée à l'instruction (2) par l'additionneur. L'instruction (2) et l'instruction (1) partagent le même opérateur (ADD). L'instruction (2) a une dépendance LAE avec l'instruction (1). La latence de l'opérateur est de 1 cycle car il est pipeliné.

Envoi de valeur

Dans ces conditions, l'instruction (2) démarre 1 cycle après l'instruction (1) si son autre source RR1 est prête.
Elle lit RR1 et RR0 dans le banc de registres de renommage (donc une valeur incorrecte pour RR0) et avant d'entrer dans l'opérateur, la nouvelle valeur de RR0, envoyée par l'opérateur, remplace celle lue du banc de registres.

Envoi de valeur



Sauts prédits et exécution spéculative

Les instructions qui suivent un saut prédit sont exécutées spéculativement.
Si la prédiction de saut est incorrecte, pour chaque instruction qui suit le saut il faut:

- **-> défaire le renommage de sa destination,**
- **-> restaurer le contenu antérieur de cette destination,**
- **-> pour une écriture en mémoire (STORE), annuler l'écriture.**

Sauts prédits et renommage spéculatif

```
// calculer R0 = 2*|R1-R2|
    SUB   R0, R1, R2 // RR0 = RR1 - RR2
    BGE   R0, e      // if (RR0>=0) goto e
    SUB   R0, R2, R1 // RR3 = RR2 - RR1
e: ADD   R0, R0, R0 // RR4 = RRx + RRx
```

Si le saut est pris, RRx = RR0, sinon RRx = RR3.

Le renommage dépend du saut. On renomme selon la prédiction.

(RR0 si prédit pris, RR3 sinon)

Sauts prédits et terminaison spéculative

```
// calculer R0 = 2*|R1-R2|
    SUB   R0, R1, R2 // RR0 = RR1 - RR2
    BGE   R0, e      // if (RR0>=0) goto e
    SUB   R0, R2, R1 // RR3 = RR2 - RR1
e: ADD   R0, R0, R0 // RR4 = RRx + RRx
```

En supposant le saut prédit non pris, la soustraction dans RR3 peut se terminer avant le saut. Le résultat est écrit spéculativement dans RR3. Si le saut est pris, il faut restaurer RR3.

Correction à la validation

On peut corriger les effets d'un saut mal prédit à sa validation.
Quand le saut est validé, les instructions qui le précèdent sont toutes validées. Les instructions qui le suivent sont toutes annulées.
Il n'y a donc plus de renommage en cours: les registres détiennent tous une seule valeur qui est leur contenu après le saut.

Correction à la validation: PIII

Tant qu'une instruction n'est pas validée, son calcul reste spéculatif. Le résultat est conservé dans le registre de renommage pointé par la table RAT. A sa validation, une instruction écrit son résultat dans sa destination architecturale.

Pour annuler les instructions qui suivent un saut mal prédit, on vide la table RAT. Les résultats validés sont dans le banc architectural.

Correction à la validation: P4

Tant qu'une instruction n'est pas validée, son calcul reste spéculatif. Le résultat est conservé dans le registre de renommage.
A sa validation, le numéro de registre de renommage est copié dans RATD.
Pour annuler les instructions qui suivent un saut mal prédit, on vide la table RATS. Les résultats validés sont ceux pointés par RATD dans le banc des registres de renommage.

Correction à la terminaison

Corriger dès la terminaison d'un saut permet de reprendre une exécution correcte plus tôt. Quand un saut se termine, des instructions qui le précède peuvent ne pas être validées. Ces instructions doivent être conservées alors que toutes celles qui suivent le saut doivent être annulées. Quand les registres de renommage sont alloués et libérés en file, on peut aisément distinguer la tranche des registres à annuler de celle à conserver.

Correction à la terminaison

```
// calculer R3 = 2*|R1-R2|
    SUB   R3, R1, R2 // RR3 = RR1 - RR2
    BGE   R3, e      // if (RR3>=0) goto e
    SUB   R3, R2, R1 // RR4 = RR2 - RR1
e: ADD   R3, R3, R3 // RR5 = RRx + RRx
```

Le saut est prédit non pris mais est pris. L'instruction SUB R3, R1, R2 n'est pas validée. Il faut annuler RR4 et RR5, mais conserver RR3. On vide RAT (ou RATS) de la tranche commençant en RR4.

Correction à la terminaison: checkpointing

Quand les registres de renommage ne sont pas alloués et libérés en file (cas du p4), on ne peut pas distinguer les registres à annuler de ceux à conserver.

Il est nécessaire de conserver une copie de la table RATS propre à chaque saut prédit (la table RATD n'est plus utile). A chaque saut prédit, une nouvelle table est allouée, initialisée avec l'état de la table courante.

Lors d'une correction, on redémarre en récupérant la table RATS allouée au saut.

Accès mémoire et spéculation

Les accès mémoire peuvent être effectués spéculativement (et en ordre des dépendances LAE). Les chargements postérieurs à un saut mal prédit peuvent être poursuivis. Cela sert de préchargement du cache. Les rangements doivent être annulés. Les rangements étant conservés en ordre dans un store buffer, il suffit d'annuler la tranche des STORE postérieurs au saut. L'annulation peut intervenir dès que le saut mal prédit se termine.

Chargements mémoire spéculatifs

Un chargement mémoire peut être effectué spéculativement en passant avant un rangement dont l'adresse est en cours de calcul.
Si l'adresse calculée révèle un conflit avec le chargement, celui-ci doit être repris (effet identique à la correction d'un saut mal prédit).
On peut reprendre le chargement après la validation du rangement ou plus précocement dès que l'adresse est établie. Un checkpointing est nécessaire lors de chaque chargement.