Travaux Dirigés n°4

Architecture des ordinateurs

—IR 2000—

Parallélisme d'instruction

Cette séance de travaux dirigée est consacré aux trois notions suivantes : réordonnancement de code, déroulage de boucles et CPI (cycle par instruction).

Quand on lance une instruction sur un processeur pipeline, le résultat n'est pas disponible immédiatement.

Définition 1 *On appelle* ***latence d'une instruction*** *(I) le nombre de cycles qu'il faut attendre avant de lancer une nouvelle instruction utilisant le résultat de l'instruction (I).*

Attention : si le pipeline dispose de court-circuit, la latence peut dépendre de l'instruction suivante.

Pour qu'un pipeline soit plein, le parallélisme d'instructions doit être exploité. Il faut trouver des séquences d'instructions indépendantes qui peuvent se recouvrir dans le pipeline. Pour éviter une suspension du pipeline, une instruction dépendante doit être placés à une distance égale de l'instruction source à la latence du pipeline pour réaliser cette instruction.

Définition 2 *On appelle* ***CPI (cycle par instruction)*** *le nombre de cycle moyen mis par un processeur pour exécuter une instruction.*

Le but du pipeline est évidement d'essayer de faire un CPI de 1. Ce T.D. est une introduction à quelques techniques de compilation destinées à faire baisser le CPI.

▸ **Exercice 1. (Ordonnancement).**

1. Dans le modèle de pipeline présenté en cours, quelle est la latence des instructions arithmétiques, logiques, de chargement/rangement si n'y a pas de court-circuit.
2. Même question avec un court circuit.

On veut compiler le code C suivant :

```
a = b + c;
d = e - f;
```

On suppose que les variables sont en mémoire à une adresse que l'on pourra représenter symboliquement par le nom de la variable. Tous les registres sont libres.

3. Écrire le code DLX qui exécute directement le code C ci dessus ;
4. Écrire les suspensions de pipeline correspondant aux temps de latence des différentes instructions sans et avec court-circuit ;
5. Ordonnancer le code pour éviter les suspensions dans le cas d'un pipeline avec court circuit.

▸ **Exercice 2. (Déroulage de boucle).** Pour simplifier on suppose les latences suivantes (on a diminué les latences des instructions flottantes) :

Intruction produisant le résultat	Intruction utilisant le résultat	Latence en cycles
Op UAL entière	tout type	0
Chargement entier	tout type	1
Op UAL flottante	Op UAL flottante	3
Op UAL flottante	Rangement flottant	2
Chargement flottant	Op UAL flottante	1
Chargement flottant	Rangement flottant	0

On veut compiler le code C suivant, qui ajoute un scalaire `s` à tous les éléments d'un vecteur `x`.

```
int i;
float x,s;
for (i=n; i>=0; i++)
      x[i] = x[i] + s;
```

Pour simplifier, on suppose que `R1` contient initialement l'adresse du dernier élément du vecteur `x[n]`, le registre flottant `F2` contient la valeur scalaire `s`, et que l'adresse du premier élément est nulle.

1. Écrire la boucle sans tenir compte du pipeline ;
2. Écrire les suspensions en tenant compte de la latence des opérations et du branchement retardé ; combien de cycle faut-il par élément du vecteur ?
3. Ordonnancer la boucle. Combien de cycle faut-il maintenant par élément du vecteur ?

À chaque itération, on perd 3 cycle pour la boucle. Pour résoudre ce problème, on doit faire plus d'opération dans la boucle par rapport aux opérations de gestion de boucle. Une solution simple à ce problème s'appelle le **déroulage de boucle**. Cette solution consiste à dupliquer le corps de la boucle plusieurs fois puis à adapter le code de terminaison de la boucle.

4. Écrire la boucle déroulée avec 4 copies du corps de la boucle. On suppose qu'a l'initialisation, `R1` est un multiple de 32 ce qui signifie que le nombre d'itérations de la boucle est un multiple de 4. Combien de cycle faut-il pour exécuter une boucle ? Combien de cycle faut-il par élément du vecteur ?
5. Ordonnancer le code précédent. Combien de cycle faut-il pour exécuter une boucle. Combien de cycle faut-il par élément du vecteur ?