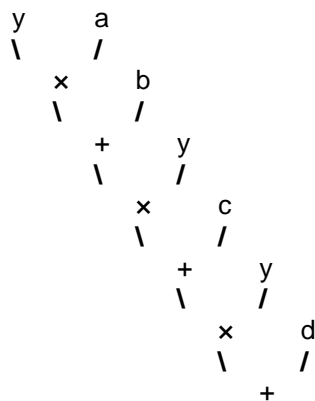


TP 4 CORRIGE. Les Processeurs actuels. Pipelining

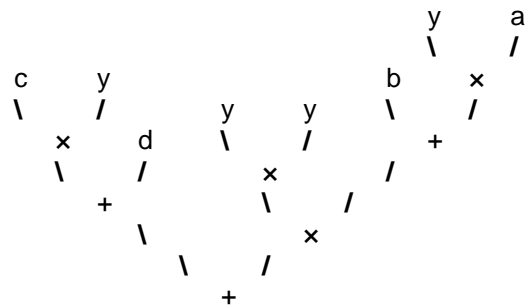
1. Pipelining (1)

Une remarque importante est que le compilateur doit effectuer le calcul exactement comme spécifié par le programme car l'addition et la multiplication en virgule flottante ne sont pas des opérations associatives.

On peut représenter le calcul par un arbre de dépendance, comme représenté ci-dessous :



Méthode 1 (de Horner) : $d + y(c + y(b + ya))$



Méthode 2 : $(d + cy) + ((y \times y) \times (ay + b))$

a) Pseudo-codes assembleur

Rappel : Initialisation :

```
F0 = y
F1 = a
F2 = b
F3 = c
F4 = d
```

Méthode 1 (de Horner) : $d + y(c + y(b + ya))$

Le programme suivant implémente la méthode 1 (de Horner) :

```
1:  F5 = F1 FMUL F0      // F5 = ay
2:  F5 = F5 FADD F2      // F5 = ay + b
3:  F5 = F5 FMUL F0      // F5 = (ay + b)y
4:  F5 = F5 FADD F3      // F5 = (ay + b)y + c
5:  F5 = F5 FMUL F0      // F5 = [(ay + b)y + c]y
6:  F5 = F5 FADD F4      // F5 = [(ay + b)y + c]y + d
```

Méthode 2 : $(d + cy) + ((y \times y) \times (ay + b))$

La méthode 2 peut être implémentée avec le programme suivant :

```
1:  F5 = F0 FMUL F1      // F5 = ya
2:  F6 = F0 FMUL F3      // F6 = yc
3:  F7 = F0 FMUL F0      // F7 = y^2
4:  F5 = F5 FADD F2      // F5 = ya + b
5:  F6 = F6 FADD F4      // F6 = yc + d
6:  F5 = F5 FMUL F7      // F5 = (ya + b)y^2
7:  F5 = F5 FADD F6      // F5 = (ya + b)y^2 + yc + d
```

b) Performances

Méthode 1 (de Horner) :

Chacune des 6 instructions dépend de l'instruction précédente. Ce programme va occuper l'étage de décodage (**decode instruction**) pendant :

$$6 + 5 \times 2 = 16 \text{ cycles.}$$

Méthode 2 :

Les instructions 1, 2, 3 sont indépendantes.

L'instruction 4 dépend de l'instruction 1 mais la pénalité de 2 cycles est couverte par les instructions 2 et 3.

De même, la dépendance entre les instructions 2 et 5 est couverte par les instructions 3 et 4.

La dépendance entre les instructions 4 et 6 n'est que partiellement couverte par l'instruction 5, donc on paye 1 cycle de pénalité.

Quant à la dépendance entre les instructions 6 et 7, elle n'est pas couverte, on paye donc 2 cycles de pénalité.

Au total, le programme occupe l'étage de décodage (**decode instruction**) pendant :

$$7 + 2 + 1 = 10 \text{ cycles.}$$

La méthode 2 est donc plus performante que la méthode 1 (de Horner).

2. Pipelining (2)

1.

Dans un processeur non pipeliné, chaque instruction doit entièrement être exécutée avant que l'exécution de la suivante ne puisse commencer.

Dans un processeur pipeliné, l'exécution de l'instruction est décomposée en plusieurs étapes correspondant aux étages du pipeline.

Dès que le premier étage du pipeline est franchi par une instruction, l'instruction suivante peut entamer son exécution en y accédant à son tour pendant que l'instruction précédente continue son exécution. Ceci augmente la vitesse d'exécution.

Le pipelining divise le chemin de données d'un processeur en étages séparés par des latches (registres).

Dans un processeur non pipeliné, une instruction doit être capable de parcourir la totalité du chemin de données en un seul cycle d'horloge.

Dans un processeur pipeliné, l'instruction doit simplement pouvoir passer à l'étage suivant à chaque cycle, ce qui permet d'avoir un cycle d'horloge beaucoup plus court.

2.

Il existe 2 limites principales.

La 1^{ère} tient au fait que, à mesure que le nombre d'étages du pipeline augmente, la fraction de temps que représente la latence (temps de traitement) du latch de chaque étage devient plus importante.

La 2^{ème} limite provient des dépendances des données et des délais de branchement.

Les instructions qui dépendent des résultats d'autres instructions doivent attendre que ces dernières aient terminé leur exécution et provoquent dans ce cas des blocages dans le pipeline (*bulles*).

Par ailleurs, les instructions qui suivent les branchements doivent attendre que ces branchements se terminent pour pouvoir être insérées dans le pipeline. Ces différents retards conduisent le pipeline à exécuter moins d'1 instruction par cycle en moyenne, ce qui implique qu'une plus grande partie du temps processeur sera consacré à attendre les déblocages du pipeline.

3.1. L'étage le plus long possède une latence de 7 ns. En ajoutant le délai de 1 ns pour le latch de pipeline (latch de chaque étage de pipeline), on obtient un temps de cycle de 8 ns (si on a des étages de différentes latences, on prend toujours la latence de l'étage le plus long).

3.2. Puisqu'il y a 5 étages, la latence totale du pipeline est de : $8 \text{ ns} * 5 \text{ étages} = 40 \text{ ns}$.