Architecture des ordinateurs et systèmes d'exploitation

Corrigé du TD 7: Pipeline

Marcel Bosc Christophe Dehlinger Arnaud Giersch Mathieu Haefele
Benoît Meister Nicolas Passat

1. **Pipelining égal**
   Un processeur non pipeliné possède un temps de cycle de 10 ns. Quels seront les temps de cycle des versions pipelinées du processeur avec un pipeline de 2, 4, 8 et 16 étages, si la logique de chemin de données est répartie de manière égale entre les étages du pipeline (on considère que le temps de stabilisation après le passage dans chaque étage est de 0,5 ns) ? En outre quel est le temps d'exécution d'une instruction complète pour chacune des versions pipelinées ?

   Correction :

   $$\textit{Temps de cycle}_{\textit{Pipeliné}} = \frac{\textit{Temps de cycle}_{\textit{Non pipeliné}}}{\textit{Nombre d'étages du pipeline}} + \textit{Temps de stabilisation} \quad (1)$$

   En appliquant cette formule, on obtient des temps de cycle de 5,5, 3, 1,75 et 1,125 ns. Pour calculer le temps d'exécution d'une instruction complète, il suffit de multiplier le temps de cycle par le nombre d'étages du pipeline, ce qui donne 11, 12, 14 et 18 ns.

2. **Pipeline et fréquence d'horloge**
   Pour le processeur de l'exercice précédent, combien d'étages de pipeline sont requis pour atteindre un temps de cycle de 2 ns ? Combien en faut-il pour un temps de cycle de 1 ns ?

   Correction : *En manipulant l'équation 1, il vient :*

   $$\textit{Nombre d'étages du pipeline} = \frac{\textit{Temps de cycle}_{\textit{Non pipeliné}}}{\textit{Temps de cycle}_{\textit{Pipeliné}} - \textit{Temps de stabilisation}}$$

   L'application de cette formule nous indique que 6,67 étages de pipeline sont nécessaires pour atteindre un temps de cycle de 2 ns. Ce résultat est arrondi à 7 dans la mesure où un nombre entier d'étages doit être implémenté. La même formule indique que 20 étages sont nécessaires pour atteindre un cycle d'horloge de 1 ns.

3. **Pipelining inégal**
   Supposons qu'un processeur non pipeliné possède un temps de cycle de 25 ns et que le chemin de données est constitué de modules dont les temps d'exécution sont respectivement 2, 3, 4, 7, 3, 2 et 4 ns (dans cet ordre). Il est alors possible de mettre en place un pipeline à 7 étages correspondant à ces 7 modules. On suppose que le temps de stabilisation après le passage dans un étage est de 1 ns.
   – Quel est le temps de cycle minimal qui peut être atteint en implémentant le pipeline sur ce processeur ?

     Correction : *Le temps de cycle minimal est déterminé par le temps d'exécution de l'étage le plus lent, auquel il convient d'ajouter le temps de stabilisation. Ici, l'étage 4 est le plus lent (7 ns), le temps de cycle est donc* $7+1=8$ *ns.*

   – Si le processeur est divisé en un plus petit nombre d'étages qui lui permettent toutefois d'atteindre le temps de cycle de la question précédente, quel sera le temps d'exécution d'une instruction complète ?

     Correction : *On peut regrouper dans un unique étage n'importe quel ensemble de modules adjacents qui offriraient un temps d'exécution total inférieur ou égal à 7 ns. Ceci nous permet de créer un pipeline à 5 étages. Le temps d'exécution d'une instruction complète est alors* $8 \times 5 = 40$ *ns. (cycle d'horloge* $\times$ *nombre d'étages)*

– Si le pipeline est limité à deux étages, quel est le temps de cycle minimal ?

Correction : *Pour pouvoir obtenir un temps de cycle minimal, il faut diviser les modules en étages ayant des temps d'exécution aussi égaux que possible. Pour deux étages, on obtient des temps de 16 et 9 ns (ou l'inverse). Le temps de cycle minimal est alors de 17 ns. (*$16\ ns + 1\ ns$ *de stabilisation).*

– Quel est alors le temps d'exécution d'une instruction complète ?

Correction : *Le temps d'exécution d'une instruction complète est alors* $2 \times 17 = 34$ *ns.*

4. **Aléas d'instruction (dépendances d'instructions)**
Soit la séquence suivante (en pseudo-assembleur) :

```
DIV r2, r5, r8   ! r2 <- r5 / r8
SUB r9, r2, r7   ! r9 <- r2 - r7
AND r5, r14, r6  ! r5 <- r14 & r6
MUL r11, r9, r5  ! r11 <- r9 * r5
BEQ r10, #0, r12 ! si (r10=0) goto r12
OR r8, r15, r2   ! r8 <- r15 | r2
```

– Identifiez tous les aléas LAE (Lecture Après Écriture).

Correction : *Aléas LAE entre DIV et SUB (r2), AND et MUL (r5), SUB et MUL (r9) et DIV et OR (r2).*

– Identifiez tous les aléas EAL (Écriture Après Lecture).

Correction : *Aléas EAL entre DIV et AND (r5) et DIV et OR (r8).*

– Identifiez tous les aléas EAE (Écriture Après Écriture).

Correction : *Pas d'aléa EAE.*

– Identifiez tous les aléas de contrôle.

Correction : *Il n'y a qu'un aléa de contrôle entre BEQ et OR.*

5. **Exécution pipelinée**
En utilisant le pipeline donné dans le cours (5 étages), donnez le diagramme d'exécution du fragment de code suivant :

```
ADD r1, r2, r3     ! r1 <- r2 + r3
SUB r4, r5, r6     ! r4 <- r5 - r6
MUL r8, r9, r10    ! r8 <- r9 * r10
DIV r12, r13, r14  ! r12 <- r13 / r14
```

Correction : *Il n'existe aucun aléa dans ce code, aussi, les instructions traversent le pipeline à la cadence d'un étage par cycle.*

	Cycles							
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
EI	*ADD*	*SUB*	*MUL*	*DIV*				
DI		*ADD*	*SUB*	*MUL*	*DIV*			
LR			*ADD*	*SUB*	*MUL*	*DIV*		
EX				*ADD*	*SUB*	*MUL*	*DIV*	
ER					*ADD*	*SUB*	*MUL*	*DIV*

6. **Exécution pipelinée avec aléas**
En utilisant le pipeline donné dans le cours (5 étages), donnez le diagramme d'exécution du fragment de code suivant :

```
ADD r1, r2, r3     ! r1 <- r2 + r3
SUB r4, r5, r6     ! r4 <- r5 - r6
MUL r8, r9, r4     ! r8 <- r9 * r4
DIV r12, r13, r14  ! r12 <- r13 / r14
```

Correction : *Cette fois, le programme comporte un aléa LAE, entre l'instruction SUB, qui écrit r4 et l'instruction MUL qui le lit. L'instruction MUL ne sera donc pas capable de lire ses registres d'entrée avant que l'instruction SUB n'ait terminé l'étage ER, ce qui crée un blocage de pipeline.*

	Cycles									
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
EI	*ADD*	*SUB*	*MUL*	*DIV*						
DI		*ADD*	*SUB*	*MUL*	*DIV*	*DIV*	*DIV*			
LR			*ADD*	*SUB*	*MUL*	*MUL*	*MUL*	*DIV*		
EX				*ADD*	*SUB*	*nop*	*nop*	*MUL*	*DIV*	
ER					*ADD*	*SUB*	*nop*	*nop*	*MUL*	*DIV*