TD 6 4MMARCA, Ensimag 2A/S6, ISI/SEOC

Réordonnancement d'instructions

Architecture avancée, 2018-2019

Les techniques utilisables pour les instructions des processeurs pipe-line travaillant sur des entiers deviennent vite inopérantes pour des processeurs ayant une partie opérative complexe, comportant de nombreux opérateurs pipe-line, par exemple des opérateurs de calcul flottant. Ces opérateuurs effectuent un calcul en plusieurs périodes d'horloge (4 à 30 périodes) : un simple « bypass » ne permettra pas de résoudre le problème de dépendances de données, car le résultat du calcul précédent n'est pas encore calculé. Ci-dessous un diagramme pour une opération flottante en (seulement) 3 cycles :

```
i1   IF   ID   EX1 EX2   EX3   MEM   WB
     i2   IF   ID  stall stall EX1
```

Quand il y a risque de nombreux cycles d'attente, il est nécessaire de réorganiser le programme pour obtenir des performances intéressantes. Ce réordonnancement peut-être effectué par le compilateur (réordonnancement statique) et est particulièrement intéressant pour les corps de boucle du programme. L'inconvénient est que cette optimisation est effectuée pour une réalisation précise du processeur. Pour une autre réalisation du processeur, le code doit être optimisé à nouveau, donc recompilé. En effet, lorsque l'on améliore les performances du processeur, la fréquence d'horloge augmente, donc la profondeur de pipe-line aussi (à technologie identique), et le nombre de cycles d'attente à éviter augmente, demandant une optimisation plus poussée pour utiliser au mieux les performances de la nouvelle version du processeur.

Ex. 1 : Optimisation de programme pour un processeur pipe-line

formats :

Le processeur utilisé est caractérisé par : registres :

registre, avec déplacement par rapport à un registre pour les accès m

Instructions :

lw ri, imm(rj)	ri ← mém(rj+imm)
lf fi, imm(rj)	fi ← mém(rj+ imm)
sw ri, imm(rj)	mém(rj+ imm) ← ri
sf fi, imm(rj)	mém(rj+ imm) ← fi
op rk,ri,rj	rk ← ri opint rj
opf fk,fi,fj	fk ← fj opflo fj
bcc ri, étiq	branchement à étiq si ri remplit la condition

Soit un programme qui ajoute une valeur flottante fixe (stockée dans F0) à un tableau de n nombres flottants dont l'adresse est dans R1.

```
      add  r2,r0,n
loop: ldl  f1,0(r1)
      addf f2,f0,f1
      stl  f2, 0(r1)
      add  r1,r1,8
      sub  r2,r2,1
      bnz  r2, boucle
```

Le processeur étant pipe-line, une instruction peut être lue et son exécution peut commencer à chaque période d'horloge, en l'absence de conflits et de dépendances. On suppose les conflits matériels résolus ; il reste le problème des dépendances, d'autant plus important que les opérations en flottant s'exécutent en plusieurs cycles, sur des opérateurs qui sont eux-mêmes pipe-line (on peut lancer une opération en flottant à chaque cycle, mais le résultat n'est pas immédiatement disponible). Le nombre de cycles d'attente entre 2 instructions dépendantes est donné dans le tableau suivant :

1ère instruction instruction dépendante	lw	lf	sw	sf	op	opf	bcc
lw	1	-	-	-	0	-	1
lf	1	-	-	-	0	-	1
sw	1	-	-	-	0	-	1
sf	1	1	-	-	0	3	1
opint	1	-	-	-	0	-	1
opflo	-	1	-	-	-	3	1
bcc	0	0	0	0	0	0	1

On voit par exemple :
— qu'après un lf, il faut attendre 1 cycle pour utiliser la valeur lue en mémoire pour une opération flottante ;
— qu'après un branchement, il faut toujours attendre 1 cycle pour commencer à exécuter l'instruction suivante.

Question 1 – Combien de cycles sont nécessaires pour chaque itération, pour la boucle donnée ci-dessus ? Peut-on réorganiser les instructions pour diminuer ce nombre de cycles (en les modifiant un peu éventuellement, mais sans changer le comportement) ?

Question 2 – On sait qu'une façon d'optimiser le temps d'exécution d'un programme est de « dérouler » les boucles. A partir du programme initial, proposer une boucle qui contienne 2 des itérations de la boucle proposée (on suppose que la dimension du tableau est multiple de 2). Utiliser pour cela autant de registres que nécessaire (registres flottant distincts pour chaque itération d'origine) et supprimer les instructions redondantes. Quel est le nombre de cycles par itération d'origine ? Réorganiser le programme obtenu pour minimiser les cycles d'attente et donner le nombre de cycles par itération d'origine. Idem en déroulant la boucle 4 fois.

On ne s'intéresse qu'aux instructions « importantes » de la boucle, lf, addf, sf. Le diagramme ci-dessous met en évidence l'exécution pipe-line du corps de boucle, avec trois phases : prologue, itération et épilogue.

```
lf  addf | sf
     lf  | addf sf
         | lf  addf sf
         |     lf   addf sf
         |          lf   addf sf
         |               lf   addf sf
         |                    lf   addf sf
         |                         lf   addf | sf
         |                              lf   | addf sf
prologue |  n -2 itérations avec pipe-line plein  | épilogue
```

Proposer un programme implémentant ce diagramme. Note : c'est la technique du pipe-line logiciel (software pipe-lining).

Question 3 – On peut modifier le processeur en implémentant un branchement à effet retardé (d'une instruction). Réécrire les programmes des questions 1 et 3 en les optimisant et donner le nombre de cycles par itération.

Question 4 – On décide de modifier le processeur. On constate qu'il dispose d'un opérateur flottant et d'un opérateur entier distincts (ainsi que de jeux de registres entier et flottant distincts), donc d'une partie opérative avec de fortes possibilités de parallélisme qui ne sont pas complètement exploitées par l'exécution séquentielle des instructions. On le modifie pour qu'il puisse exécuter des "super instructions", qui comportent une instruction normale (toute instruction sauf une opération en flottant) et une instruction de flottante. Par exemple, avec le symbole `||` indiquant que l'on exéctute les 2 instructions simultanément :

```
lf fj,imm(ri) || addf fk,fl,fm
```

Réécrire les programmes obtenus en question 4 en utilisant ces super instructions et évaluer le nombre de cycles par itération d'origine. Note : ce type de processeurs s'appelle processeur VLIW (very long instruction word)