UNIVERSITÉ PARIS SACLAY
Master E3A – SETI

Examen Architecture T1
Décembre 2020

3h – Documents autorisés pour la deuxième partie de l'examen.

Partie 2

1 Optimisation de boucles

On utilise le processeur superscalaire défini dans l'annexe 1, ou sa version scalaire.

Soit la boucle suivante écrite en assembleur, qui travaille sur des tableaux de `floats` `X[256]` et `Y[256]`. Au démarrage `R1` contient l'adresse de `X[0]` et `R2` contient l'adresse de `Y[0]`. `R4` contient la valeur 256. `F0` contient la valeur 0.5.

```
1     ADDI R4,R4, -2
2  Boucle:
3     LF   F1, 0(R1)
4     LF   F2, 8(R1)
5     FADD F1,F1,F2
6     FMUL F1,F1,F0
7     SF   F1, 4(R2)
8     ADDI R1, R1,4
9     ADDI R2, R2,4
10    ADDI R4, R4,-1
11    BNE  R4, R0, Boucle
```

1.1 Donner le programme C équivalent.

1.2 Indiquer les dépendances du programme et l'optimiser.

Donner l'exécution cycle par cycle de la boucle avant et après optimisation dans la version scalaire du processeur. Quel est, en nombre de cycles, le temps d'exécution par itération de la boucle ?

1.3 On considère maintenant la version surperscalaire du processeur. Donner l'exécution cycle par cycle de la boucle optimisée en plaçant les instructions dans les différents pipelines E0, E1, FA et FM comme sur le tableau ci-dessous. Quel est, en nombre de cycles, le temps d'exécution par itération de la boucle ?

cycles	E0	E1	FA	FM
1				
2				
...				

1.4 Toujours dans la version superscalaire, réaliser un déroulage de boucle d'ordre 4 sur le programme précédent. Donner l'exécution cycle par cycle de la boucle en indiquant l'occupation des différents pipelines.

2 Caches

On suppose qu'un processeur a un cache données de 64 Ko, avec des lignes de 64 octets. Le cache utilise la réécriture avec écriture allouée (il y a des défauts de cache en écriture) Le processeur a des adresses sur 32 bits.

2.1 Quel est pour ce cache le nombre de bits pour le déplacement dans la ligne, le nombre de bits d'index et le nombre de bits d'étiquette pour un cache à correspondance directe ?

2.2 `X` est un vecteur de `floats`. Sachant que `X[0]` est à l'adresse $1000\ 0000_H$, dans quelles lignes du cache vont les flottants `X[0]` et `X[2048]` ?

2.3 Quel est le nombre total de défauts de cache lors de l'exécution des trois boucles ci-dessous pour le cache à correspondance directe ?

```
float X[4096] ;
/*(A)*/
for (i=0 ; i<2048 ; i++)
   S+= X[i];
/*(B)*/
for (i=0 ; i<2032 ; i++)
   S+= X[i] + X[i+16];
/*(C)*/
for (i=0 ; i<2048 ; i++)
   S+= X[i] + X[i+2048];
```

2.4 On suppose maintenant un cache de 8 Ko à correspondance directe avec des lignes de 64 octets et la boucle suivante :

```
float X[4096] ;
for (i=0 ; i<2048 ; i++)
   S+= X[i] + X[i+2048];
```

Quel est maintenant le nombre de défauts de cache lors de l'exécution de la boucle ?

3 Prédiction de branchement

Soit le programme C suivant :

```
int array[1000] = { /* valeurs quelconques */ };
int sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;
for (i = 0; i < 1000; i ++) // BR0 : branchement de boucle
{
  if (i % 4 == 0)     // BR1 : IF CONDITION 1
    sum1 += array[i]; // Pris
  else
    sum2 += array[i]; // Non pris

  if (i % 2 == 0)     // BR2 : IF CONDITION 2
    sum3 += array[i]; // Pris
  else
    sum4 += array[i]; // Non pris
}
```

On rappelle qu'en C, `x % y` renvoie le reste de la division entière de `x` par `y`.

Le but de l'exercice est de déterminer la qualité de la prédiction (% de prédictions correctes) pour les branchements BR0 (pris dans la boucle et non pris en sortie de boucle), et les branchements BR1 et BR2 à l'intérieur de la boucle, pour différents types de prédicteurs

3.1 Quelle est la qualité de la prédiction de chacun des branchements BR0, BR1 et BR2 pour un prédicteur 1 bit (pris/non pris ou P/N). On supposera que tous les prédicteurs sont initialisés à "non pris" (N).

3.2 Quelle est la qualité de prédiction de chacun des branchements BR0, BR1 et BR2 pour un prédicteur 2 bits (fortement pris P/faiblement pris p/faiblement non pris n/fortement non pris N). On supposera que tous les prédicteurs sont initialisés à "fortement non pris".

3.3 Même question quand les prédicteurs sont initialisés à “faiblement non pris”.

3.4 On suppose maintenant que l'on utilise un prédicteur avec historique avec 2 bits d'historique (prise en compte des deux derniers branchements exécutés) et 2 bits de prédiction comme ci dessus. Il y aura ainsi un prédicteur différent pour chacun des historiques des deux banchements précédents. Comme BR0 est (presque) toujours pris, en pratique, pour BR1, il y aura un prédicteur différent suivant que BR2 (à l'étape $i-1$) a été pris ou non pris. De même pour BR2, il y aura deux prédicteurs suivant que BR1 (à l'étape i) a été pris ou non pris.

On suppose tous les prédicteurs initialisés à “fortement non pris” et tout l'historique des branchements intialisé à “non pris”.

Dans ces conditions, déterminer la précision de la prédiction pour BR1 et BR2.

3.5 Montrer qu'un déroulage de boucle permet d'éviter d'effectuer des branchements. Donner une version en C de ce programme optimisé.

4 Mémoire virtuelle et TLB

Soit le programme suivant de transposition de matrices :

```
float a[1024][1024], b[1024][1024];
...
int i, j;
for(i = 0; i < 1024; i++)
  for(j = 0; j < 1024; j++){
     b[j][i]=a[i][j];
```

4.1 On considère un processeur dont les pages font 4ko et on supposera que les tableaux sont disposé successivement en mémoire et qu'ils ont leur élément 0 en début d'une page. On supposera que le TLB comprend 8 entrées disponibles pour les données et qu'il est totalement associatif avec un gestion des remplacements LRU.

Calculer le nombre de défauts de TLB lors de l'exécution de ce programme.

4.2 Même question si le TLB comprend 4 ensembles de 2 voies. On pourra supposer que l'adresse de `a[0][0]` correspond à l'ensemble 0 du TLB.

4.3 Même question avec un TLB à correspondance directe.

5 Caches et associativité

On considère un cache à 4 entrées, avec des lignes de 1 octet. On veut comparer 3 politiques différentes : cache totalement associatif (4 voies) (TA), cache partiellement associatif (2 ensemble de 2 voies) (PA), cache à correspondance directe (CD). Pour les caches associatifs, on supposera que la politique de remplacement est LRU. Dans ces différents cas, indiquer le contenu des différentes lignes du cache avec l'adresse associée à chaque ligne pour le motif d'accès 0, 1, 2, 3, 4, 0, 4, 0, 2, 1. On remplira les tableaux joints. Pour les caches associatifs, on mettra la dernière ligne accédée dans chaque ensemble en haut. Dans tous les cas, on indiquera si la dernière ligne accédée a conduit à un **succès** ou un **échec**.

ANNEXE 1

Soit un processeur superscalaire à ordonnancement statique qui a les caractéristiques suivantes :

- les instructions sont de longueur fixe (32 bits)
- Il a 32 registres entiers (dont R0=0) de 32 bits et 32 registres flottants (de F0 à F31) de 32 bits.
- Il peut lire et exécuter 4 instructions par cycle.
- L'unité entière contient deux pipelines d'exécution entière sur 32 bits, soit deux additionneurs, deux décaleurs. Tous les court-circuits possibles sont implantés.
- L'unité flottante contient un pipeline flottant pour l'addition et un pipeline flottant pour la multiplication.
- L'unité Load/Store peut exécuter jusqu'à deux chargements par cycle, mais ne peut effectuer qu'un *load* et un `store` simultanément. Elle ne peut effectuer qu'un seul `store` par cycle.
- Le processeur dispose d'un mécanisme de prédiction de branchement qui permet de *brancher* en 1 cycle si la prédiction est correcte. Les sauts et branchements ne sont pas retardés.

La Table 1 donne les instructions disponibles et le pipeline qu'elles utilisent : E0 et E1 sont les deux pipelines entiers, FA est le pipeline flottant de l'addition et FM le pipeline flottant de la multiplication. Les instructions peuvent être exécutées simultanément si elles utilisent chacune un pipeline séparé. L'addition et la multiplication flottante sont pipelinées. La division flottante n'est pas pipelinée (une division ne peut commencer que lorsque la division précédente est terminée). L'ordonnancement est statique.

JEU D'INSTRUCTIONS (extrait)

Code	Instruction	Latence	Pipeline	Signification
LF	LF Fi, dép.(Ra)	2	E0 ou E1	Fi ← M (Ra + dépl.16 bits signé)
SF	SF Fi, dép.(Ra)	0	E0	Fi → M (Ra + dépl.16 bits signé)
ADD	ADD Rd,Ra, Rb	1	E0 ou E1	Rd ← Ra + Rb
ADDI	ADDI Rd, Ra, IMM	1	E0 ou E1	Rd ← Ra + IMM (16 bits signé)
SUB	SUB Rd,Ra, Rb	1	E0 ou E1	Rd ← Ra - Rb
FADD	FADD Fd, Fa, Fb	3	FA	Fd ← Fa + Fb
FMUL	FMUL Fd, Fa, Fb	3	FM	Fd ← Fa × Fb
BEQ	BEQ Ri, dépl	1	E1	si Ri=0 alors CP ← NCP + depl
BNE	BNE Ri, dépl	1	E1	si Ri≠ 0 alors CP ← NCP + depl