Examen Décembre 08 - Architectures Avancées

3H – Tous documents autorisés

OPTIMISATION DE BOUCLES

On utilise le processeur superscalaire décrit dans l'annexe 1. Les instructions ont les latences définies dans l'annexe 1 table 1.

Soit le programme P1 suivant :

```
float x[100], y[100], z[100], a;

main ()
{
int i ;
for (i=0;i<100;i++)
      {
      a = x[i]+ y[i];
      if (a <0.0)
            a=0.0;
      z[i] = a;
      }
```

On supposera que l'adresse de X[0] est initialement dans R1, que l'adresse de Y[0] est initialement dans R2, que l'adresse de Z[0] est initialement dans R3. F0 contient la valeur 0.0, et R4 contient initialement le nombre d'itérations de la boucle.

Q 1) Quel est en nombre de cycles, le temps d'exécution par itération de la boucle originale en supposant que les branchements sont parfaitement prédits

	E0	E1	FA	FM

On considère maintenant une prédiction de branchement statique, les branchements avant étant prédits « non pris » et les branchements arrière étant prédits pris. Une mauvaise prédiction de branchement coûte 4 cycles.

Q 2) Avec la prédiction statique, quel est le nombre minimal et quel est le nombre maximal de cycles par itération, selon les résultats de la comparaison à chaque itération ?

On ajoute maintenant l'instruction FCMOVyy

FCMOVyy	FCMOVyy Fc, Fa, Fb	FA	4	si Fc yy 0, alors Fa ← Fb avec yy = EQ,NE, GT, GE, LT, LE

Q 3) Reprendre les questions 1 et 2 en utilisant l'instruction FCMOV

- nombre de cycles avec prédiction de branchement parfait
- nombre de cycles avec prédiction statique.

Q 4) Donner la liste des instructions SIMD (sur 128 bits) qui seraient utilisées par le programme P1.

SIMD IA-32

Soit le programme suivant

```
unsigned char X[512], Y[512], A[512];
for (i=0 ; i<512 ; i++)
      A[i] = abs (X[i] –Y[i] ) ;// abs est la fonction valeur absolue
```

Q 5) Donner la version SIMD IA-32 du programme ci-dessus, en utilisant les instructions IA-32 données en annexe (on supposera que les vecteurs A, X et Y sont alignés sur des frontières de mots de 128 bits)

Pipeline logiciel avec TMS 320C62x

Soit la boucle de pipeline logiciel suivante, qui travaille sur un tableau T[N] :

```
            CMPGT    .L2X     A3,B4,B0         ; B0 = 1 si A3>B4 ;=0 sinon
||          CMPLT    .L1      A3,A5,A1         ; A1 = 1 si A3<A5 ;=0 sinon
|| [ A2]    B        .S2      $C$L2              ;

   [ B1]    SUB      .S2      B1,1,B1            ;
|| [ A1]    MV       .S1      A3,A5              ;
|| [ B0]    MV       .L2X     A3,B4              ;
|| [ A2]    SUB      .L1      A2,1,A2            ;
|| [ B1]    LDH      .D1T1    *A4++,A3           ; A4 pointe sur T[N]

   [ A1]    SUB      .S1      A4,1,A6              ;
|| [ B0]    SUB       S2X     A4,1,B6              ;
```

Ce pipeline logiciel correspond à un programme C dont le début est

```
Type1 T[N] ; Type2 X, Y, Z, W ;
X = T[0] ;
Y = T[0] ;
Z=0;
W=0 ;
for (i=1 ; i<N ;i++){
      Corps de boucle ; }
```

Q 6) Quel est le type des variables T[N] d'une part, X, Y, Z et W d'autre part. Donner le corps de boucle du programme C original. Que fait le programme.

Q 7) Quel est le nombre de cycles par itération du pipeline logiciel ? Pourquoi ne peut-on diminuer ce nombre de cycles ?

Instructions spécialisées sur processeur NIOS.

On veut implémenter sous forme d'instructions spécialisées les deux instructions flottantes 32 bits suivantes (simple précision)

- Rd = FNEG (Rs) // Rs contient un nombre flottant 32 bits. L'instruction flottante négation FNEG place dans Rd le nombre opposé (si Rs contient 3.5 , alors Rd recevra -3.5)
- Rd = FABS (Rs) // Rs contient un nombre flottant 32 bits. L'instruction flottante valeur absolue FABS place dans Rd la valeur absolue du nombre contenu dans Rs

Q 8) Donner le code VHDL (entité et architecture) pour les instructions FNEG et FABS. Quel nombre de cycles d'horloge processeur utilisera chacune de ces instructions ?

Cohérence des caches dans un multiprocesseur

Soit un biprocesseur symétrique utilisant le protocole MESI (invalidation et « write back »). Les caches sont associatifs 4 voies avec l'écriture allouée (traitement du défaut de cache lors d'un défaut en écriture). Les lignes (blocs) de cache ont 64 octets (16 floats).

Le compilateur parallélise le code suivant

```
Int N, i;
Float A[N], B[N], C[N] ;
for (i =0 ; i<N ; i++)
        C[i]= A[i]+B[i] ;
```

Le processeur P0 exécute le code

```
for (i=0 ; i<N ; i+=2)
        C[i]= A[i]+B[i] ;
```

Le processeur P1 exécute le code

```
for (i =1 ; i<N ; i+=2)
        C[i]= A[i]+B[i] ;
```

Q 9) En supposant que les adresses de A, B et C correspondent à des débuts de ligne de cache, déterminer le nombre de défauts de cache et d'invalidations en fonction de N.

Q 10) Proposer une autre parallélisation (en donnant le code pour P0 et P1) qui minimise invalidations et défauts de caches. Quel est alors le nombre de défauts de cache et d'invalidations en fonction de N ?

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 bypass 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.

- Il 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).

JEU D'INSTRUCTIONS (extrait)

	Assembleur	Pipeline	Latence	Action
LD	LD Fi, dép.(Ra)	E0 ou E1	2	Fi ← M (Ra + dépl.16 bits avec ES)
SD	SD Fi, dép.(Ra)	E0	-	Fi → M (Ra + dépl.16 bits avec ES)
ADD	ADD Rd,Ra, Rb	E0 ou E1	1	Rd ← Ra + Rb
ADDI	ADDI Rd, Ra, IMM	E0 ou E1	1	Rd ← Ra + IMM-16 bits avec ES
SUB	SUB Rd,Ra, Rb	E0 ou E1	1	Rd ← Ra - Rb
SUBI	SUBI Rd, Ra, IMM	E0 ou E1	1	Rd ← Ra - IMM-16 bits avec ES
FADD	FADD Fd, Fa, Fb	FA	4	Fd ← Fa + Fb
CMPyy	CMP Rd, Ra, Rb	E0 ou E1	1	Rd ←1 si Ra yy Rb, Rd ← 0 sinon avec yy = EQ,NE, GT, GE, LT, LE
FMUL	FMUL Fd, Fa, Fb	FM	4	Fd ← Fa x Fb
BEQ	BEQ Ri, dépl	E1	1	si Ri=0 alors CP ← CP + depl
FCMPyy	FCMPyy Fd, Fa, Fb	FA	4	Fd ←1.0 si Fa yy Fb, Fd ← 0.0 sinon avec yy = EQ,NE, GT, GE, LT, LE
FBxx	FBxx Fi, dépl	FM	-	si Fi xx 0.0 alors CP ← NCP + depl avec xx= EQ,NE, GT, GE, LT, LE NCP est l'adresse de l'instruction après FBxx
Bxx	Bxx Ri, dépl	E1	-	si Ri xx 0 alors CP ← NCP + depl avec xx= EQ,NE, GT, GE, LT, LE NCP est l'adresse de l'instruction après Bxx

Table 1 : instructions disponibles

ANNEXE 2 : Instructions SIMD IA-32 utilisables

#define ld16(a)	_mm_load_si128(&a)	chargement aligné
#define st16(a, b)	_mm_store_si128(&a, b)	rangement aligné
#define or(a,b)	_mm_or_si128(a,b)	ou logique
#define xor(a,b)	_mm_xor_si128(a,b)	ou exclusif
#define maxbu(a,b)	_mm_max_epu8(a,b)	max 8 bits non signés
#define minbu(a,b)	_mm_min_epu8(a,b)	min 8 bits non signés
#define addh(a,b)	_mm_add_epi16(a,b)	addition 16 bits signée
#define addhu(a,b)	_mm_add_epu16(a,b)	addition 16 bits non signée
#define subh(a,b)	_mm_sub_epi16(a,b)	soustraction 16 bits signée
#define subbu (a,b)	_mm_subs_epu8(a,b)	Soustraction 8 bits non signés avec saturation
#define b2hl(a,b)	_mm_unpacklo_epi8 (a,b)	4 octets bas entrelacés vers 4 shorts
#define b2hh(a,b)	_mm_unpackhi_epi8 (a,b)	4 octets haut entrelacés vers 4 shorts
#define h2b(a,b)	_mm_packus_epi16(a,b)	8 shorts (a) dans 8 octets bas - 8 shorts (b) dans 8 octets haut
#define srli128(a,v)	_mm_srli_si128(a,v)	décalage droite des 128 bits de a de v octets
#define slli128(a,v)	_mm_slli_si128(a,v)	décalage gauche des 128 bits de a de v octets
#define srai16(a,v)	_mm_srai_epi16(a,v)	déc. droite des8x 16 bits de a de v bits
#define slli16(a,v)	_mm_slli_epi16(a,v)	déc. gauche des 8x16 bits de a de v bits
#define cmpgt8(a,b)	_mm_cmpgt_epi8(a,b)	comparaison octets signés. Si octet (a)> octet(b) octet résultat = FF_H sinon 00_H