ARCHITECTURE DES ORDINATEURS
Examen Janvier 2008
2 H – Tous documents autorisés

OPTIMISATIONS DE PROGRAMME
Superscalaire statique

Un processeur superscalaire à ordonnancement statique 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. Les opérations Load et Store sont pipelinées.
- 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
- 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'ordonnancement est statique. Les chargements ne peuvent pas passer devant les rangements en attente.

Dans la version scalaire du processeur, une seule instruction démarre par cycle d'horloge.

JEU D'INSTRUCTIONS (extrait)

Instruction		Pipeline	Latence	
LF	LF Fi, dép.(Ra)	E0 ou E1	2	Fi <- M (Ra + dépl.16 bits avec ES)
SF	SF 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
FADD	FADD Fd, Fa, Fb	FA	3	Fd <- Fa + Fb
FSUB	FSUB Fd, Fa, Fb	FA	3	Fd <- Fa - Fb
FMUL	FMUL Fd, Fa, Fb	FM	3	Fd <- Fa x Fb
BEQ	BEQ Ri, dépl	E1	1	si Ri=0 alors CP <- CP + depl
BNE	BNE Ri, dépl	E1	1	si Ri≠0 alors CP <- CP + depl

Table 1 : instructions disponibles

La colonne latence de la table 1 donne la latence entre une instruction source et une instruction destination, dans le cas de dépendances de données. La valeur 1 est le cas où les deux instructions peuvent se succéder normalement, d'un cycle i au cycle i+1.

La table 2 présente un programme C et le programme assembleur correspondant
On suppose 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. F10 contient la valeur de A, F11 contient la valeur de B et R4 contient initialement le nombre d'itérations de la boucle.

Table 2 : Programme C et programme assembleur

`float X[1024],Y[1024],Z[1024],A,B;`
`for (i=0 ; i<1024 ; i++)`
`  Z[i] = A*X[i] – B*Y[i] ;`
	ADDI R4, R0, 1024
	Boucle : LF F1,0(R1)
	FMUL F1,F1,F10
	LF F2, 0(R2)
	FMUL F2,F2,F11
	FSUB F1,F1,F2
	SF F1,0(R3)
	ADDI R1,R1,4
	ADDI R2,R2,4
	ADDI R3,R3,4
	ADDI R4,R4,-1
	BNE R4, Boucle

Q 1) Pour la version SCALAIRE du processeur, quel est le temps d'exécution de la boucle, en nombre de cycles d'horloge, telle qu'elle est écrite dans la Table 2 ?

Q 2) Pour la version scalaire, optimiser la boucle de la table 3. Quel est, en nombre de cycles, le temps d'exécution de la boucle optimisée ?

Q 3) Donner cycle par cycle la boucle optimisée pour la version SUPERSCALAIRE du processeur. Quel est le nombre de cycles par itération ?

Q 4) Quel est le temps d'exécution par itération de la boucle initiale avec un déroulage de boucle d'ordre 4 (4 itérations de la boucle initiale par corps de boucle) pour la version superscalaire ?

CACHES.

On suppose qu'un processeur a des adresses mémoire de 32 bits (4 Go adressables) et un cache données de 64 Ko, avec des blocs de 64 octets. Il utilise la réécriture (write back) alloué (sur un défaut de cache en écriture, on traite le défaut de cache avant d'écrire dans le cache).
L'algorithme LRU est utilisé pour les remplacements.
Soit le programme suivant P1

```
float X[256], Y[256], Z[256] ;
for (i=0 ; i<256 ; i++)
      Z[i] = X[i]–Y[i] ;
```

On suppose que les tableaux X[N], Y[N], Z[N] sont rangés à partir de l'adresse hexadécimale 1000 0000H.

Q 5) a) Quelles sont les adresses de X[0], Y[0] et Z[0] ?

b) Quel est pour ce cache le nombre de bits pour l'adresse dans le bloc, le nombre de bits d'index et le nombre de bits d'étiquette dans les trois cas suivants : 1) correspondance directe, 2) associativité deux voies (deux blocs par ensemble), 3) associativité quatre voies (quatre blocs par ensemble)

c) Quel est le nombre total de défauts de caches lors de l'exécution du programme P1 pour les trois cas suivants : 1) correspondance directe, 2) associativité deux voies (deux blocs par ensemble), 3) associativité quatre voies (quatre blocs par ensemble) ?

Q 6) Pour quelle valeur minimale de N étant une puissance de 2 aura-t-on trois défauts de cache par itération avec la correspondance directe ?

SIMD IA-32

Soit une image X[128][128] en niveau de gris. On applique sur cette image un certain filtre qui produit une image Y[128][128].
On utilise le jeu d'instructions IA-32, avec les « define » ci-dessous pour les intrinsics IA-32.

#define byte unsigned char

#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 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

Soit la fonction filtre_simd

```
/* ------------------------------------------------------------- */
void filtre_simd(byte **X, int i0, int i1, int j0, int j1, byte **Y)
/* ------------------------------------------------------------- */
{int i, j;
  __m128i aij, tmp, zero;
```

```
 __m128i **XS, **YS;
 XS=X; YS=Y;
zero = xor(zero,zero);
for(i=i0; i<=i1; i++) {
   for(j=j0; j<=j1/16; j++) {
           aij=ld16(&XS[i][j]);
           tmp=cmpgt8 (zero, aij);
           st16(&YS[i][j], tmp); }}}
```

Q 7) Que fait la fonction filtre_simd ? Donner le corps de boucle de la version scalaire équivalente. Quelle accélération maximale peut-on espérer par rapport à la version scalaire correspondante ?

NB : les valeurs des pixels sont comprises entre 0 et 255 (octets non signés) et l'instruction SIMD correspondant à cmpgt8 porte sur des octets signés.

Pipeline

Un processeur utilise un pipeline à 4 étages : LI, DI, EX, et RR, dont les différentes phases sont les suivantes :

LI : lecture de l'instruction

DI : décodage de l'instruction, lecture des opérandes, calcul de l'adresse de branchement et de l'adresse mémoire

EX : exécution UAL ou opération mémoire (lecture/écriture)

RR : rangement du résultat.

Q 8) En supposant qu'il n'y ait tous les mécanismes d'anticipation (bypass) nécessaires, indiquer le nombre de cycles de suspension dans les cas suivants d'instruction produisant un résultat suivie d'une instruction utilisant le résultat.

Instruction source	Instruction destination	Cycles de suspension
LD **R1**, 0(R2)	ADD R3, **R1**,R4	
LD **R1**, 0(R2)	ST R3, 0(**R1**)	
LD **R1**, 0(R2)	ST **R1**, 0(R4)	
ADD **R1**,R2,R3	ST **R1**, 0(R4)	
ADD **R1**,R2,R3	ST R4, 0(**R1**)	
ADD **R1**,R2,R3	SUB R5, **R1**,R6	

Q 9) Quel est le nombre de cycles de suspension pour les branchements, dans les cas branchement pris et branchement non pris (en supposant que les branchements sont prédits non pris) ?

Efficacité parallèle

On rappelle que l'efficacité parallèle est l'accélération obtenue avec N processeurs divisée par N (nombre de processeurs).

Un programme séquentiel a 10% de son temps d'exécution qui ne peut être parallélisé.

Q 10) Quel est le nombre de processeurs nécessaires pour obtenir une efficacité parallèle de 0,5 ?