ARCHITECTURE DES ORDINATEURS
Examen Décembre 2007 (CORRIGÉ)
3H – Tous documents autorisés

EXECUTION D'INSTRUCTIONS

On utilise le jeu d'instructions NIOS II .

Q 1) Donner l'instruction ou la suite des instructions NIOS II pour effectuer les actions suivantes :

a) Mettre à zéro le registre R1
ADD R1,R0,R0 ou ADDI R1,R0,0 ou XOR R1,R1,R1 ou …
b) Mettre 00008000_H dans le registre R2
ORI R2,R0,8000H
c) Mettre $F000A000_H$ dans le registre R3
ORI R3,R0,A000H
ORHI R3,R3,F000H
d) Diviser par 2 le contenu du registre R4, interprété en signé
SRAI R4,R4 ,1
e) Multiplier par 17 le contenu du registre R5
SLLI R4,R5,4
ADD R5,R4,R5
f) Mettre à zéro les cases mémoire entre les adresses F0000000 et $F000FFFF_H$
ORHI R1,R0,F000
ORI R2,R1,FFFC
Boucle : STW R0,0(R1)
ADDI R1,R1,4
BLE R1,R2, Boucle

PROGRAMMATION ASSEMBLEUR

Q 2) Ecrire une procédure NIOS II qui vérifie si un caractère, contenu dans l'octet de poids faible de R2, est un chiffre, et renvoie 1 dans R2 si vrai et 0 sinon. (On suppose que le caractère a été chargé précédemment dans le registre par une instruction LDBU)

On rappelle que les chiffres sont codés entre 30_H (0) et 39_H (9)

```
Test_chiffre :
        CMPGEUI R3,R2,30h // R3 = 1 si R2 >= 30H
        CMPLEUI R4,R2,39h // R4=1 si R2 <= 39H
        AND R2,R3,R4
        RET
```

PIPELINES

Un processeur a les pipelines suivants pour les trois types d'instructions travaillant sur les entiers

- Instructions arithmétiques simples
  LI DI LR EX ER
- Instructions mémoire
  LI DI LR EX AM ER
- Instructions de multiplication
  LI DI LR M1 M2 M3 ER

Les différents étages ont la signification suivante :

- LI à l'accès au cache instructions (lecture de l'instruction)
- DI : décodage
- LR : Lecture des opérandes dans le banc de registres
- EX : Calculs UAL pour les opérations arithmétiques et logiques et pour les calculs d'adresse mémoire et de branchement. Si un branchement a été mal prédit, toutes les instructions dans les étages précédents du pipeline sont annulées et l'on réinitialise le pipeline à partir de la bonne adresse
- AM : accès au cache données
- M1, M2, M3 : multiplication entière pipelinée
- ER : Ecriture du résultat dans le banc de registres.

Tous les circuits d'anticipation (bypass) existent.

Q 3) Donner les latences entre instruction producteur et instruction consommateur en nombre de cycles.

NB : la valeur n signifie que deux instructions dépendantes peuvent se suivre aux cycles i et i+n. UAL correspond aux instructions arithmétiques et logiques simples. MUL correspond à la multiplication. LDW et STW sont les instructions de chargement et de rangement

	Instruction producteur	Instruction consommateur	Latence
a	UAL Ri, -, -	UAL -, Ri, -	1
b	UAL Ri, -, -	STW Ri, ()	1
c	UAL Ri, -, -	LDW Rj, (Ri)	1
d	LDW Ri, ()	UAL -, Ri, -	2
e	UAL Ri	MUL -,Ri,-	1
f	LDW Ri, ()	MUL -,Ri,-	2
g	MUL Ri,	STW Ri, ()	2
h	LDW Ri,-,-	STW -,Ri,-	1

a)

LI DI LR **EX** ER
LI DI LR **EX** ER

b)

LI DI LR EX **ER**
LI DI LR EX **AM** ER

c)

LI DI LR EX **ER**
LI DI LR **EX** AM ER

d)

LI	DI	LR	EX	**AM**	ER			
	LI	DI	LR		**EX**	ER		
e)								
LI	DI	LR	**EX**	ER				
	LI	DI	LR	**M1**	M2	M3	ER	
f)								
LI	DI	LR	EX	**AM**	ER			
	LI	DI	LR		**M1**	M2	M3	ER
g)								
LI	DI	LR	M1	M2	**M3**	ER		
	LI	DI	LR	EX		**AM**	ER	
h)								
LI	DI	LR	EX	**AM**	ER			
	LI	DI	LR	EX	**AM**	ER		

CACHES.

On suppose que le processeur utilisé a un cache données de 64 Ko, avec des blocs de 64 octets. Il utilise l'écriture simultanée (write through) non allouée. On rappelle qu'avec l'écriture non allouée, il n'y a pas de défauts de cache en écriture

Soit le programme suivant P1

```
#define N 256
float X[N], Y[N], Z[N] ;
for (i=0 ; i<N ; i++)
if (X[i] > Y[i]) Z[i] = X[i]-Y[i] ;
      else Z[i] = 0.0 ;
```

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

Q 4)

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 deux cas suivants : 1) correspondance directe, 2) associativité deux voies (deux blocs par ensemble).

c) Quel est le nombre total de défaut de caches lors de l'exécution du programme P1 pour les deux cas suivants : a) correspondance directe, b) associativité deux voies (deux blocs par ensemble) ?

Un float a 4 octets. Chaque tableau utilise 4 x 256 = 1024 octets
X[0] = 1000 0000 H
Y[0] = 1000 0400 H
Z[0] = 1000 0800 H
Les blocs ont 64 octets (6 bits). Il y a 64 Ko/64 = 1024 blocs. Un bloc contient 16 floats

En correspondance directe, on a 6 bits (adresse dans le bloc), 10 bits d'index et 16 bits d'étiquettes. En associativité deux voies, on a 6 bits (adresse dans le bloc), 9 bits d'index et 17 bits d'étiquettes.

En correspondance directe, X[0] va dans le bloc 0 et Y[0] va dans le bloc 16. Il n'y a pas de conflit. On a donc deux défauts de cache toutes les 16 itérations (1/8). Le nombre total de défauts est de 256/8=32 défauts.
La situation est la même avec l'associativité deux voies.

Q 5 : Pour quelle valeur minimale de N étant une puissance de 2 aura-t-on deux défauts de cache par itération avec la correspondance directe ?

Il y a conflit si les deux adresses de départ correspondent au même bloc du cache. Avec 6 bits d'adresse dans le bloc et 10 bits d'index, il faut que les deux adresses diffèrent à partir du 17ème bit, soit par exemple
X[0] = 1000 0000H
Y[0] = 1001 0000H
Le tableau X a alors pour taille 2^{16} octets soit 2^{14} floats. Ceci correspond à N =2^{14} = 16384

OPTIMISATION DE BOUCLES

On ajoute au jeu d'instructions NIOS II des instructions flottantes simple précision (32 bits) (Figure 2) et 32 registres flottants F0 à F31 (F0 est un registre normal).
Les additions, soustractions et multiplications flottantes sont pipelinées. Une nouvelle instruction peut démarrer à chaque cycle. Les latences sont de 2 cycles pour LF et de 3 cycles pour les instructions flottantes.
L'instruction FCMLT fd, fs, ft est une instruction de transfert conditionnel. Si ft <0.0 alors le contenu de fs est copié dans fd. Sinon, l'instruction ne fait rien (NOP). Dans les deux cas, sa latence est de 3.
Les branchements ne sont pas retardés. On suppose qu'il n'y a pas de pénalité de branchement.

LF	2	LF ft, déplac(rs)	rt ← MEM [rs +SIMM]
SF	1	SF ft, déplac.(rs)	ft → MEM [rs + SIMM]
FADD	3	FADD fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
FMUL	3	FMUL fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)
FSUB	3	FSUB fd, fs,ft	fd ← fs - ft (soustraction flottante simple précision)
FDIV	12	FDIV fd,fs,ft	fd ← fs / ft (division flottante simple précision)
FCMLT	3	FCMOVLT fd, fs, ft	Si ft <0.0 alors fd ← fs

Figure 1 : Instructions flottantes ajoutées (Ce ne sont pas les instructions NIOS)

Soit le programme assembleur P1, qui travaille sur des tableaux de flottants simple précision (float) X[N], Y[N] et Z[N] rangés successivement en mémoire, avec N = 256. L'adresse de X[0] est initialement contenue dans le registre R1.

```
FSUB F0,F0,F0
```

```
        Mov R2, 1024
Boucle :LF F1, 0(R1)
        LF F2, 1024(R1)
        FSUB F1,F1,F2
        FCMLT F1,F0,F1
        SF F1,2048(R1)
        ADDI R1,R1,4
        BLT R1,R2, Boucle
```

Q 6) Donner le code C correspondant au programme P1

```
#define N 256
float X[N], Y[N], Z[N] ;
for (i=0 ; i<N ; i++)
if (X[i] > Y[i]) Z[i] = X[i]-Y[i] ;
      else Z[i] = 0.0 ;
```

Q 7) En montrant l'exécution cycle par cycle du programme assembleur P1 (après optimisation), donner le nombre de cycles par itération de la boucle du programme assembleur P1.

*NB : cette question est faisable même sans avoir répondu à la question Q*1

	FSUB F0,F0,F0
	Mov R2, 1024
1	Boucle :LF F1, 0(R1)
2	LF F2, 1024(R1)
3	
4	FSUB F1,F1,F2
5	ADDI R1,R1,4
6	
7	FCMLT F1,F0,F1
8	
9	
10	SF F1,2044(R1)
11	BLT R1,R2, Boucle

11 cycles par itération.

Q 8) Donner l'exécution optimisée cycle par cycle avec un déroulage de boucle d'ordre 2. Quel est maintenant le nombre de cycles par itération de la boucle initiale ?

	FSUB F0,F0,F0
	Mov R2, 1024
1	Boucle :LF F1, 0(R1)
2	LF F3, 4(R1)
3	LF F2, 1024(R1)
4	LF F4, 1028(R1)

5	FSUB F1,F1,F2
6	FSUB F3,F3,F4
7	ADDI R1,R1,8
8	FCMLT F1,F0,F1
9	FCMLT F3,F0,F3
10	
11	SF F1,2044(R1)
12	SF F3,2044(R1)
13	BLT R1,R2, Boucle

13 cycles/2 itérations soit 6,5 cycles/itération.

Q 9) Donner directement (sans détailler l'exécution cycle par cycle) le nombre de cycles par itération de la boucle initiale pour un déroulage de boucle d'ordre 4

Il y a 4 LF, 2 FSUB, 2 FCMLT et 2 SF supplémentaires soient 10 instructions supplémentaires qui utiliseront 9 cycles de plus par rapport au déroulage par 2
22 cycles pour 4 itérations soit 5,5 cycles/itération.

PROGRAMMATION SIMD IA32

Soient les define pour le jeu d'instructions IA32

```
#define ld16(a)        _mm_load_si128(&a)          //chargement aligné (entiers)
#define st16(a, b)     _mm_store_si128(&a, b)      //rangement aligné (entiers)
#define orh(a,b)       _mm_or_si128(a,b)           //ou logique
#define xorh(a,b)      _mm_xor_si128(a,b)          //ou exclusif
#define maxhs(a,b)     _mm_max_epi16(a,b)          // max  sur 8 x16 bits signés
#define minhs(a,b)     _mm_min_epi16(a,b)          // min sur 8 x 16 bits signés
#define addhs(a,b)     _mm_add_epi16(a,b)          // addition 8 x 16 bits signée
#define subhs(a,b)     _mm_sub_epi16(a,b)          //soustraction 8 x 16 bits signée
```

Soit les variables _m128i a, b, c ;

Q 10) Que font les instructions suivantes

```
b = xorh (a, a) ;
c= subhs (a, a) ;
```

Deux manières différentes de mettre à zéro le registre de 128 bits , soient 8 shorts à 0.

Q 11) Que fait le programme suivant ?

```
#define N 1024
_mi128i X[N/8], A, tmp1, tmp2 ;

      A = xorh (A,A) ;
For (i=0 ; i< N/8 ; i++) {
      ld16 (&X[i], tmp1) ;
      tmp2 = subhs (A,tmp1) ;
      tmp1 = maxhs (tmp1,tmp2)
```

```
      st16 (&X[i], tmp1) ;}
```

Le programme calcule la valeur absolue de chaque « short » du tableau X[N] et la réécrit dans X[N]. (change le signe de tous les shorts négatifs du tableau)

Q 12) Ecrire la version scalaire du programme C équivalent.

```
#define N 1024
short X[N]
for (i=0 ; i<N ; i++){
if (X[i] <0)
      X[i] = -X[i] ;
```