Architectures avancées

Examen Décembre 2012
3H – Tous documents autorisés
5 parties indépendantes

1. Optimisation de programmes

On utilise le processeur superscalaire défini en annexe 1 et la boucle SAXPY :

```
float Y[1024], X[1024], A ; int I;
for (i=0 ; i<1024 ;i++)
      Y[i]+=A*X[i] ;
```

Le code assembleur est donné dans la **Erreur ! Source du renvoi introuvable.**.

- La version scalaire utilise les instructions des Figure 8 et Figure 9.
- La version SIMD utilise les instructions des Figure 8 et Figure 10.

Avec l'adresse de X[0] dans 0[R3] et adresse de Y[0] dans 4096[R3]

Version Scalaire	Version SIMD
LSF F0, A // F0← A	LPF X0, A // F0← A
ADDI R4,R3,4096	SETF X0,X0
Boucle : LSF F1,0(R3)	ADDI R4,R3,4096
LSF F2,4096(R3)	Boucle : LPF X1,0(R3)
MULSF F1,F1,F0	LPF X2,4096(R3)
ADDSF F2,F2,F1	MULPF X2,X1,X0
ADDI R3,R3,4	ADDPF X2,X2,X1
SSF F2, 4092(R3)	ADDI R3,R3,16
BNE R3,R4,Boucle	SPF X2, 4080(R3)
	BNE R3,R4,Boucle

Figure 1 : code assembleur SAXPY

Q 1) Donner le temps d'exécution par itération de SAXPY de la version superscalaire après optimisation, mais sans déroulage (montrer le placement des instructions dans les différents pipelines)

Q 2) Donner le temps d'exécution par itération de SAXPY de la version SIMD après optimisation, mais sans déroulage (montrer le placement des instructions dans les différents pipelines)

Q 3) Donner le temps d'exécution par itération SAXPY de la boucle initiale de la version superscalaire avec un déroulage de boucle d'ordre 4. (Il n'est pas nécessaire de fournir le placement cycle par cycle si le résultat est correct, mais le placement cycle par cycle peut permettre de comprendre des erreurs éventuelles).

Q 4) Donner le temps d'exécution par itération SAXPY de la boucle initiale de la version SIMD avec un déroulage de boucle d'ordre 4. (Il n'est pas nécessaire de fournir le placement cycle par cycle si le résultat est correct, mais le placement cycle par cycle peut permettre de comprendre des erreurs éventuelles).

2. Instructions spécialisées NIOS

Q 5) Ecrire le code VHDL d'une instruction spécialisée NIOS qui multiplie le contenu d'un registre R1 contenant un nombre flottant simple précision par 2^N, N étant contenu dans l'octet de poids faible d'un registre R2. Lorsque le résultat n'est pas représentable dans le format flottant simple précision, alors le résultat sera l'infini. (La Figure 2 présente le format flottant simple précision).

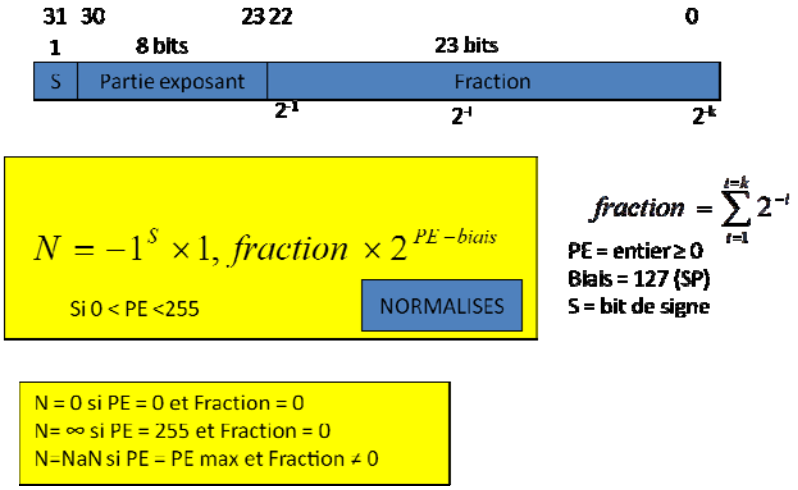

Figure 2 : Représentation flottante simple précision

3. Pipeline logiciel TMS320C67

Soit un pipeline logiciel correspondant à une version flottante de SAXPY.

LOOP	LDW	D1	*A4++,A2	yi
||	LDW	D2	*B5++,B2	xi
||	SUB	S1	A1,1,A1	
||[A1]	B	S2	Loop	
	MPYSP	M2x	A8,B2,B6	a*xi
||	ADDSP	L2x	A2,B6,B7	yi+a*xi
||	STW	D1	*(B4)++,B7	yi

Figure 3 : Pipeline logiciel SAXPY

Q 6) Pourquoi le pipeline logiciel a un intervalle inter-itération de 2 ?

Les latences des instructions sont

LDW : 5
MPYSP et ADDYSP : 4
SUB, B, STW : 1

La Figure 4 présente les deux premiers cycles du prologue et le pipeline logiciel.

Q 7) Compléter le prologue en plaçant les instructions à chaque cycle (du cycle 2 au cycle 11) en utilisant les instructions « partielles » qui apparaissent dans le pipeline logiciel.

	0	2	4	6	8	10	12
D1	LDW a						LDW yi
D2	LDW xi						LDW xi
M1							
M2							
L1							
L2							
S1	MV A4,B4						SUB
S2	MVK 100,A1						B

	1	3	5	7	9	11	13
D1							
D2							STW yi
M1							
M2							MYPSP
L1	NOP						
L2							ADDSP
S1							
S2							

Figure 4 : Deux premiers cycles du prologue et pipeline logiciel

4. SIMD IA-32

Q 8) En utilisant les intrinsics définis dans la Figure 11, donner la suite d'instructions SIMD pour faire la somme des 4 flottants V3V2V1V0 contenus dans une variable 128 bits V.

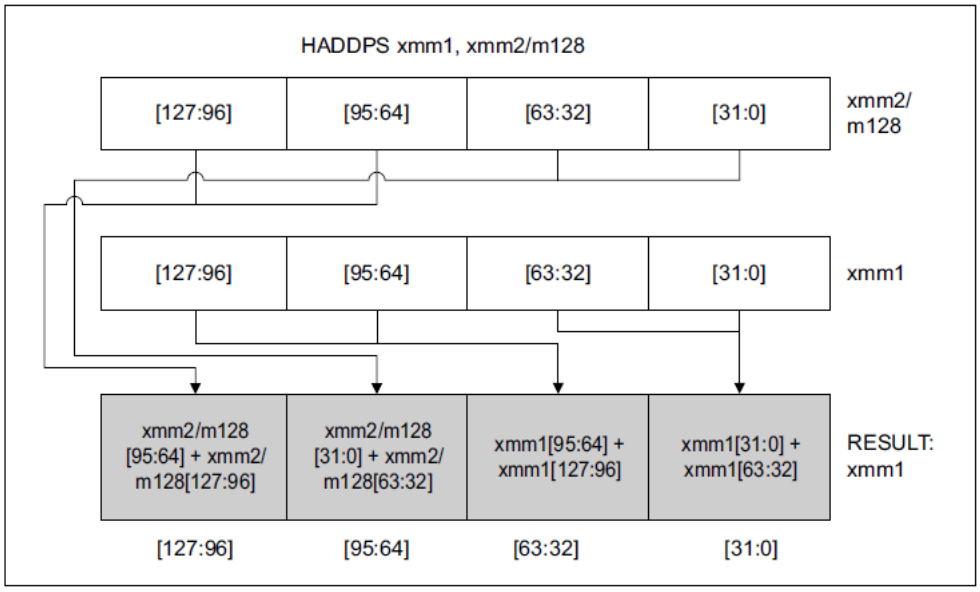

Figure 5 : Instruction HADDPS

Soit le produit scalaire de deux vecteurs X[1024] et Y[1024] contenant des nombres F32 (flottants simple précision)

```
float S=0.0, X[1024], Y[1024], RES;
for (i=0 ; i<1024 ; i++)
      S+= X[i]*Y[i];
RES=S;
```

Q 9) En utilisant les intrinsics définis dans la Figure 11, écrire la version SIMD IA-32 du produit scalaire. Le résultat sera rangé dans la variable RES.

5. OpenMP

Soit une fonction OMP dessous qui calcule la somme des N premiers entiers.

```
int N = 200000000;
#define NB_THREADS 4
void main_OMP ()
{        int i;        double x, pi, sum = 0.0;
         omp_set_num_threads(NB_THREADS);
                starttime=dtime();
#pragma omp parallel for reduction(+:sum)
         for (i=0;i< N; i++)
               sum+=i;
         benchtime=dtime()-starttime ;
 printf (" temps execution OMP4/iteration %f  \n ", (double) (benchtime)/N);
       printf("sum OMP_4 threads= %f \n \n", sum);
}
```

Figure 6 : Code OpenMP pour la somme des N premiers entiers.

La **Erreur ! Source du renvoi introuvable.** présente les résultats de son exécution, en même temps que le résultat de l'exécution de la version séquentielle et de la version calculant directement la somme des N premiers entiers par multiplication et division N(N-1)/2.

```
sum N*(N-1)/2= 19999999900000000.000000

temps execution SEQ/iteration  = 3.030499
sum_sequentiel= 19999999867108864.000000

temps execution OMP4/iteration  = 1.385521
sum OMP_4 threads= 19999999900000000.000000
```

Figure 7 : Résultats d'exécution

Q 10) Expliquer les résultats obtenus.

6. 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 a 16 registres SIMD de 128 bits (X0 à X15).
- 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.

- 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 Figure 8 donne les instructions entières disponibles et le pipeline E0 ou E1 qu'elles. Les table et table donnent deux versions des instructions flottantes et d'accès mémoire, sans et avec SIMD. 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)

Mnémo	Syntaxe	Latence	Pipeline	Effet
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 avec ES
SUB	SUB Rd,Ra, Rb	1	E0 ou E1	Rd ←Ra - Rb
BEQ	BEQ Ri, Rj, dépl	1	E1	si Ri=Rj alors CP ← NCP + depl
BGE	BNE Ri, Rj, dépl	1	E1	si Ri≠Rj alors CP ← NCP + depl

Figure 8 : instructions entières disponibles

Mnémo	Syntaxe	Latence	Pipeline	Effet
LSF	LSF Fi, dép.(Ra)	3	E0 ou E1	Fi ← M (Ra + dépl.16 bits avec ES)
SSF	SSF Fi, dép.(Ra)	1	E0	Fi -> M (Ra + dépl.16 bits avec ES)
ADDSF	ADDF Fd, Fa, Fb	3	FA	Fd ←Fa + Fb
MULSF	MULF Fd, Fa, Fb	5	FM	Fd ←Fa x Fb

Figure 9 : Instructions flottantes – version 1

Mnémo	Syntaxe	Latence	Pipeline	Effet
LPF	LPF Xi, dép.(Ra)	3	E0 ou E1	Xi ← M (Ra + dépl.16 bits avec ES)
SPF	SPF Xi, dép.(Ra)	1	E0	Xi -> M (Ra + dépl.16 bits avec ES)
ADDPF	ADDPF Xd, Xa, Xb	3	FA	Xd ←Xa + Xb (Addition SIMD)
MULPF	MULPF Xd, Xa, Xb	5	FM	Xd ←Xa x Xb (Multiplication SIMD)
SETF	SETF Xd,Xa	2	E0	Xd3=Xd2=Xd1=Xd0 ← Xa0 Float « bas » Xa 4 fois dans Xd

Figure 10 : Instructions flottantes SIMD – version 2

ADDPS (a,b)	_mm_add_ps(a,b)	Quatre additions flottantes 32 bits.
MULPS (a,b)	_mm_mul_ps(a,b)	Quatre multiplications flottantes 32 bits
DIVPS (a,b)	_mm_div_ps (a,b)	Quatre divisions flottantes 32 bits
LF4 (p)	_mm_load_si128 (*p)	Chargement aligné de 128 bits
SF4 (p,a)	_mm_store_si128 (*p,a)	Rangement aligné de 128 bits
SF1 (p,a)	_mm_store_ss(float * p, a)	Rangement float de poids faible d'un mot de 128 bits.
SETF (w)	_mm_set_ps1 (float w)	Quatre fois le flottant 32 bits w dans un mot de 128 bits
HADDPS (a,b)	_mm_hadd_ps(a, b)	Addition horizontale de 4 flottants

Figure 11 : Instructions SIMD utilisables