Corrigé Examen Décembre 2011
Architectures Avancées
3H – Tous documents autorisés

OPTIMISATION DE BOUCLES

Soit les programmes assembleur P1 et P2.

R1 contient initialement l'adresse d'un tableau X[128] de flottants simple précision;
R2 contient initialement l'adresse d'un tableau Y[128] de flottants simple précision;
R3 contient initialement la valeur 126.

```
P1 :
Boucle :    LF F1, 0(R1)
            LF F2, 4(R1)
            LF F3, 8(R1)
            FADD F0,F1,F2
            FADD F0,F0,F3
            SF F0, 4(R2)
            ADDI R1,R1,4
            ADDI R2,R2,4
            ADDI R3,R3,-1
            BGTZ R3, Boucle
```

```
P2
Boucle :    LF F1, 0(R1)
            LF F2, 4(R1)
            LF F3, 8(R1)
            LF F4,12(R1)
            FADD F5, F2,F3
            FADD F6, F1, F5
            FADD F7, F4, F5
            SF F6, 4(R2)
            SF F7, 8(R2)
            ADDI R1,R1,8
            ADDI R2,R2,8
            ADDI R3, R3, -2
            BGTZ R3, Boucle
```

Question 1) Donner le code C correspondant aux programme P1 et P2

```
Programme P1 :
float X[128], Y[128] ;
for (i=1 ; i<127 ; i++)
      Y[i] = X[i-1]+X[i]+X[i+1];

Programme P2 :
float X[128], Y[128] ;
for (i=1 ; i<127 ; i+=2){
      Y[i] = X[i-1]+X[i]+X[i+1];
      Y[i+1] = X[i]+X[i+1]+X[i+2];}
```

Les latences des instructions flottantes sont
- LF : 2 cycles
- FADD : 3 cycles
- SF : 1 cycle

Question 2) Quel le temps d'exécution, en nombre de cycles d'horloge par itération de la boucle de P1 optimisée sur le processeur superscalaire

Cycle	E0	E1	FADD	FMUL
Boucle	`LF F1, 0(R1)`	`LF F2, 4(R1)`		
2	`LF F3, 8(R1)`			
3	`ADDI R1,R1,4`	`ADDI R2,R2,4`	`FADD F0,F1,F2`	
4	`ADDI R3,R3,-1`			
5				
6			`FADD F0,F0,F3`	
7				
8				

9	SF F0, 0(R2)	BGTZ R3, Boucle		

9 cycles/itération

Question 3) Quel est le temps d'exécution de la boucle de P2 optimisée sur le processeur superscalaire en nombre de cycles par itération de la boucle initiale ?

Cycle	E0	E1	FADD	FMUL
Boucle	LF F2, 4(R1)	LF F3, 8(R1)		
2	LF F1, 0(R1)	LF F4,12(R1)		
3	ADDI R1,R1,8	ADDI R2,R2,8	FADD F5, F2,F3	
4	ADDI R3,R3,-3			
5				
6			FADD F6, F1, F5	
7			FADD F7, F4, F5	
8				
9	SF F6, -4(R2)	BGTZ R3, Boucle		
10	SF F7, 0(R2)	BGTZ R3, Boucle		

10 cycles pour 2 itérations soit 5 cycles/itération.

SIMD IA-32 (première partie)

Soit le programme suivant, qui utilise les intrinsics IA-32 fournis en annexe

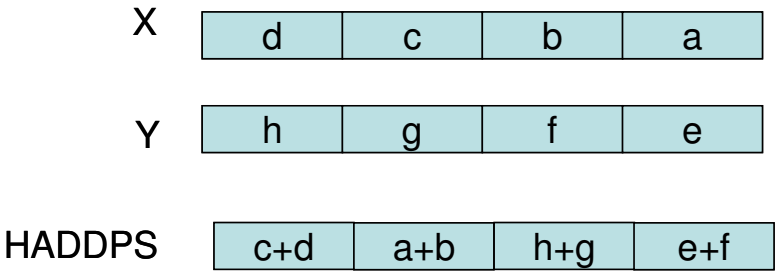
```
_m128 XS[1025] YS[1025] ,A, B, C ; // mots de 128 bits (4 floats)

For (i=0 ; i<1024 ; i++) {
      A= lf4 (&XS[i]) ;
      B= lf4u (&XS[i] +4) ;
      C= lf4u (&XS[i] + 8) ;
      A = addps (A,B) ;
      A = addps (A,C) ;
      sf4u (&YS[i] +4 , A) ;
}
```

Question 4) Donner le code C scalaire correspondant au code SIMD

For (i=1 ; i<4095 ; i++=
    Y[i] = X[i-1] + X[i] + X[i+1] ;

SIMD IA 32 (Deuxième partie)

L'effet de l'instruction HADDPS sur des mots de 4 floats est défini par la figure suivante :



Question 5 : Comment peut on effectuer l'addition horizontale (d+c+b+a) à l'aide de l'instruction HADDPS ?

HAPPS (X,X) ;
HAPPS (X,X)

Question 6 : En utilisant les intrinsics définis en annexe, écrire la version SIMD du programme C suivant.

```
float A[1024][1024], Y[1024], X[1024], SF ;

for (i=0 ; i<1024 ; i++) {
      S=0.0 ;
      for (j=0 ; j<1024 ; j++)
            SF+= A[i][j] * X[j];
      Y[i] = SF; }
```

On utilisera les variables
_mm128 **AS, *X, *Y, SS; // sur des mots de 4 floats.

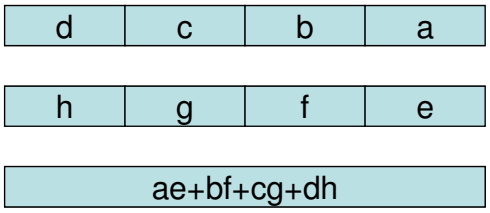
For (i=0 ; i<1024. i++){
SS= setf (0.0) ;
For (j=0 ; j<256 ; j++)
SS= ADDPS (SS, MULPS (AS[i][j], X[j])) ;
SF4(&YS[i], SS);

INSTRUCTIONS SPECIALISEES POUR NIOS II

Question 7 : Définir l'instruction spécialisée NIOS II correspondant au code VHDL ci-dessous :

```
use IEEE.std_logic_unsigned.all;

entity XX is

      port
      (
      dataa :     in    std_logic_vector(31 downto 0);
      datab :     in    std_logic_vector(31 downto 0);
      result:     out   std_logic_vector(31 downto 0)
      );
end XX;

architecture comp of XX is
signal tmp, a, b, c, d, e , f ,g, h : std_logic_vector (15 downto 0);
signal ae, bf, cg, dh: std_logic_vector (31 downto 0);
begin
a <= "00000000"& dataa (7 downto 0);
b <= "00000000"& dataa(15 downto 8);
c <="00000000"&dataa(23 downto 16);
d <= "00000000"&dataa(31 downto 24);
e <= "00000000"& datab (7 downto 0);
f <= "00000000"&datab(15 downto 8);
g <="00000000"&datab(23 downto 16);
h <= "00000000"&datab(31 downto 24);
ae <= a * e;
bf <= b * f;
cg <= c * g;
dh <= d * h;
result <= ae + bf + cg + dh ;
end comp; -- end of architecture
```

Instruction DOTUP (produit scalaire sur des octets non signés et résultat sur 32 bits)



Question 8 : Donner le code VHDL (entité + architecture) pour ajouter au jeu d'instructions NIOS l'instruction Addition horizontale ADDHUB (octets non signés) et résultat sur 32 bits

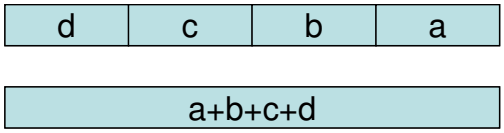


Figure 1 : Instructions ADDHUB (gauche) et DOTUN (droite)

```
-- IEEE Libraries --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--use IEEE.std_logic_signed.all;
library work;
--use work.float_pkg.all;

entity addhu is

      port
      (
      dataa :     in    std_logic_vector(31 downto 0);
      result:     out   std_logic_vector(31 downto 0)
      );
end addhu;

architecture comp of addhu is
signal tmp, a, b, c, d : std_logic_vector (9 downto 0);

begin
a <= "00"& dataa (7 downto 0);
b <= "00"& dataa(15 downto 8);
c <="00"& dataa(23 downto 16);
d <= "00"& dataa(31 downto 24);
tmp <= a+b+c+d;
result <= "00000000000000000000"& tmp ;

end comp; -- end of architecture
```

PIPELINE LOGICIEL TMS 320C6x

Le code assembleur TMS320C62 ci-dessous donne l'itération du pipeline logiciel pour un programme C.

```
LOOP:
            LDW   .D1   *A4++,A2          ; load ai and ai+1
||          LDW   .D2   *B4++,B2          ; load bi and bi+1
||[A0]      ADD   .L1   A6,A7,A7          ; sum0+=(ai*bi)
```

```
||[B0]      ADD   .L2   B6,B7,B7          ; sum1+=(ai+1*bi+1)
||[A1]      B     .S2   LOOP              ; branch to loop

           MPY   .M1X  A2,B2,A6          ; ai*bi
||         MPYH  .M2X  A2,B2,B6          ; ai+1*bi+1
||         CMPLT .L1    0,A6,A0          ; A0 = 1 si A6 >0
||         CMPLT .L2 0,B6,B0             ; B0 = 1 si B6 >0
||[A1]      SUB   .S1   A1,1,A1           ; decrement loop counter

           ADD   .L1X  A7,B7,A4          ; sum=sum0+sum1
```

On rappelle le délai et la latence des instructions entières

Cycle	0	2	4	6	8
D1	LDW	LDW	LDW	LDW	LDW
D2	LDW	LDW	LDW	LDW	LDW
L1					ADD
L2					ADD
S1					
S2			B	B	B
M1					
M2					

Cycle	1	3	5	7	9
D1					
D2					
L1				CMP	CMP
L2				CMP	CMP
S1					
S2		SUB	SUB	SUB	SUB
M1			MPY	MPY	MPY
M2			MPYH	MPYH	MPYH

Instruction Type	Delay Slots
NOP (no operation)	0
Store	0
Single cycle	0
Multiply (16 × 16)	1
Load	4
Branch	5

Tableau 1 : Délai des instructions entières (latence = 1+délai)

Question 9) Donner cycle par cycle le prologue correspondant au pipeline logiciel.

Prologue

```
LDW
||LDW
NOP
LDW
||LDW
```

```
SUB
LDW
||LDW
||B
SUB
||MPY
||MPYH
LDW
||LDW
||B
SUB
||MPY
||MPYH
||COMP
||COMP
```

LOI D'AMDAHL

Un programme séquentiel a 5% de son temps d'exécution qui ne peut être parallélisé. On veut l'accélérer à taille constante.

Question 10) Quel est le nombre de processeurs nécessaire pour obtenir une accélération de 8 ? Quelle est alors l'efficacité parallèle (accélération /nombre de processeurs) ?

$$\text{Acc} = \frac{1}{0{,}05 + \frac{0{,}95}{n}} = 8$$

0,05 + 0,95/n = 0,125
0,95/ n = 0,075
n=0,95/0,075 = 13

Accélération parallèle = 8/13 = 61%

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 (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
- 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. Les chargements ne peuvent pas passer devant les rangements en attente.

JEU D'INSTRUCTIONS (extrait)

LF	LF Fi, dép.(Ra)	2	E0 ou E1	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	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
FADD	FADD Fd, Fa, Fb	3	FA	Fd ←Fa + Fb
FSUB	FSUB Fd, Fa, Fb	3	FA	Fd ← Fa - Fb
FMUL	FMUL Fd, Fa, Fb	3	FM	Fd ←Fa x Fb
BEQ	BEQ Ri, dépl		E1	si Ri=0 alors CP ← NCP + depl
BNE	BNE Ri, dépl		E1	si Ri≠0 alors CP ← NCP + depl

Table 1 : instructions disponibles (avec latence et pipeline utilisé)

ANNEXE 2 : Instructions SIMD IA-32 utilisables

MULPS	_mm_mul_ps(a,b)	Quatre multiplications flottantes 32 bits
ADDPS	_mm_add_ps (a,b)	Quatre additions flottantes 32 bits
LF4	_mm_load_m128 (&x)	Chargement aligné de 128 bits
LF4U	_mm_loadu_m128 (&x)	Chargement non aligné de 128 bits
SF4	_mm_store_m128 (&x,a)	Rangement aligné de 128 bits
SF4U	_mm_storeu_m128 (&x,a)	Rangement non aligné de 128 bits
SF1	_mm_store_ss(float * p, a)	Rangement float de poids faible d'un mot de 128 bits.
SETF	_mm_set_ps1 (float w)	Quatre fois le flottant 32 bits w dans un mot de 128 bits
HADDPS	_mm_hadd_ps(a, b)	Addition horizontale de 4 flottants