ARCHITECTURE DES ORDINATEURS
Corrigé - Examen Juin 2007
3 H – Tous documents autorisés

NOMBRES FLOTTANTS

On représente sur 32 bits des nombres flottants avec les conventions suivantes :

Exposant e sur 8 bits en code excès 127 ; soit E = e + 127

Partie fractionnaire f sur 23 bits.

Le format est donné ci-dessous.

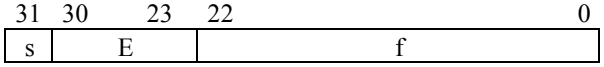
Un nombre flottant normalisé x est donné par

$x = \mathbf{1}.f \times 2^e$ si 0<E<255 quand s = 0 (positif) ; NB : le 1 est implicite

$x = -\mathbf{1}.f \times 2^e$ si 0<E<255 quand s = 1 (négatif) ; NB : le 1 est implicite

x = 0 si E=0, s = 0 et f = 0

$x = \infty$ si E=255 et f = 0 ; x = NaN si E=255 et $f \neq 0$.



Q 1) On considère les nombres normalisés positifs (zéro exclu).

a) quelle est la plus grande différence (écart) entre deux nombres normalisés positifs successifs ?

L'écart entre deux valeurs successives de la mantisse est 2^{-23}

Le plus grand écart intervient pour le plus grand exposant soit E=254 et e = 127

Il est égal à $2^{-23} \times 2^{127} = 2^{104}$

b) quelle est la plus petite différence (écart) entre deux nombres normalisés positifs successifs ?

Le plus petit écart intervient pour le plus petit exposant soit E=1 et e=-126

Il est égal à $2^{-23} \times 2^{-126} = 2^{-149}$

Q 2) Donner la valeur décimale correspondant aux cas suivants (représentation hexadécimale de nombres flottants 32 bits)

a) $4080\ 0000_H$

0 | 100 0000 1 | 000 …… = + 1.0 x $2^{(129-127)}$ = +4

b) $C1C8\ 0000_H$

1 | 100 0001 1 | 100 1000 …..= - (1+1/2+1/16) x $2^{(131-127)}$ = -(25/16) x 16 = -25

c) $7F80\ 0000_H$

0 | 111 1111 1 | 000 0000…. = + ∞

E = 255 f = 0

Q 3) Donner les valeurs hexadécimales correspondant aux résultats des opérations suivantes

a) $4080\ 0000_H + C1C8\ 0000_H = C1A8\ 0000$

4 – 25 = -21 = -(21/16) x 16 = -(1+1/4+1/16) x 2(131-127)

1 | 100 0001 1 | 010 1000….

b) $4080\ 0000_H \times C1C8\ 0000_H$

4 x (-25) = -100 = -(100/64) x 64 = - (1+ ½+1/16) x $2^{(133-127)}$

1 | 100 0010 1 | 100 1 0000…. = C 2C8 0000_H

EXECUTION D'INSTRUCTIONS

On utilise un sous ensemble du jeu d'instructions MIPS décrit en annexe.

Q 4) En partant à chaque fois du contenu de la table 1, donner le contenu des registres et des cases mémoire modifié après exécution des instructions MIPS ou séquences d'instructions. Dans chaque cas, on considérera que l'instruction est à l'adresse 1000 0000_H

```
a)    ADD R8,R3,R4          R8 = 7111B5B4
b)    AND R9,R3,R4          R9 = 84120200
c)    XOR R10,R3,R4         R10 = 68EDB1B4
d)    BGEZ R0, +10 //  10 est un nombre d'instructions  CP =1000 002C
e)    BGEZ R3, +10          CP = 1000 0004
f)    LW R11, (R1+8)        R11 = Mem(1008) = FEDC8765
g)    SRL R12,R3,8          R12= 008432A3
h)    SRA R13, R3,8         R13 = FF8432A3
```

Q 5) Donner l'instruction ou la suite des instructions MIPS pour effectuer les actions suivantes :

a) Mettre 1 dans le registre R1

```
ADDI R1,R0,1
```

b) Mettre à zéro les 4000 octets commençant à l'adresse mémoire $F0000000_H$ (en supposant que le registre R2 contient $F0000000_H$)

```
        ADDI R3,R2,4000
Boucle  SW R0, (R2)
        ADDI R2,R2,4
        BNEQ R2,R3, Boucle
```

Registre	Contenu (hexa)
R0	00000000
R1	00001000
R2	00001016
R3	8432A380
R4	ECDF1234
R6	00000020
R7	00000030

Adresse (hexa)	Contenu (hexa)
00001000	00000001
00001004	00000002
00001008	FEDC8765
0000100C	000000A4
00001010	00000005
00001014	00000006

Table 1 : contenu des registres du processeur (MIPS) et de cases mémoire

CACHES.

On suppose que le processeur utilisé a un cache données de 16 Ko, avec des blocs de 64 octets.
Le cache utilise la réécriture avec écriture allouée (il y a des défauts de cache en écriture)
Le processeur a des adresses sur 32 bits.

On considère le programme suivant, pour lequel les tableaux X et Y sont rangés successivement en mémoire à partir de l'adresse 1000 0000_H (adresse de X[0].)

```
double X[4096], Y[2048];
for (i=0 ; i<2048 ; i++)
      Y[i] = X[i+2048] – X[i] ;
```

Q6) 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 : a) correspondance directe, b) associativité quatre voies (quatre blocs par ensemble).

En correspondance directe, il y a 6 bits pour l'adresse dans le bloc, 8 bits pour l'index (256 blocs de 64 octets) et 18 bits d'étiquette.
En associativité 4 voies, il y a 6 bits pour l'adresse dans le bloc, 6 bits d'index (64 ensembles) et 20 bits d'étiquette.

Q7) Quelles sont les adresses de X[0], X[2048] et Y[0] ?

X[0] est à l'adresse 1000 0000_H
X[2048] est à l'adresse 1000 0000 + 4000 = 1000 4000_H
Y[0] est à l'adresse 1000 0000 + 8000 = 1000 8000_H

Q8) Quel est le nombre total de défauts de caches lors de l'exécution du programme pour les deux cas suivants : a) correspondance directe, b) associativité quatre voies

En correspondance directe,
X[0] est à l'adresse 1000 0000_H soit dans le bloc 0
X[2048] est à l'adresse 1000 4000_H soit dans le bloc 0
Y[0] est à l'adresse 1000 8000_H soit dans le bloc 0
Il y a conflit puisque les trois accès mémoire de chaque itération correspondent au même bloc.
Il y a donc 3 défauts par itération soit un total de 6144 défauts de cache

En associativité quatre voies, il n'y a pas de conflits possibles pour 3 accès par itération. Un bloc de 64 octets contient 8 doubles. Il y a donc 3 défauts de cache toutes les 8 itérations soit un total de 768 défauts de cache.

OPTIMISATIONS DE PROGRAMME

On suppose une version pipelinée du processeur utilisant les instructions MIPS données en annexe. (Les branchements ne sont pas retardés : branchements normaux).
La latence des instructions est donnée dans la deuxième colonne des figures 3 et 4.
On rappelle qu'une latence de **n** signifie que si une instruction **I** démarre au cycle **c**, une instruction qui utilise le résultat de I ne peut démarrer qu'au cycle c+n. (une latence de 1 signifie qu'elle peut démarrer au cycle suivant).
La table 2 présente un programme C et le programme assembleur MIPS correspondant (On suppose que R3 contient au départ l'adresse de X[0]).

Q9) Quel est le temps d'exécution (en cycles par itération) de la boucle de la table 2. Optimiser la boucle et donner le nouveau temps d'exécution.

Table 2 : Programme C et programme assembleur

float X[100], S;
int i ;
for (i=0; i<100; i++)
S+=X[i]*X[i];
	ADDI R5, R3, 400
	FSUB F0, F0, F0
	Boucle :LF F1,(R3)
	FMUL F1,F1,F1
	FADD F0,F0,F1
	ADDI R3,R3,4
	BNEQ R3,R5, Boucle
	SF F0, (adresse de S)

```
          ADDI   R5, R3, 400
          FSUB   F0, F0, F0
1Boucle :LF      F1,(R3)
2
3         FMUL   F1,F1,F2
4
5
6         FADD   F0,F0,F1
7         ADDI   R3,R3,4
8         BNEQ   R3,R5, Boucle
```

Total : 8 cycles par itération de la boucle

Version optimisée

```
          ADDI   R5, R3, 400
          FSUB   F0, F0, F0
1Boucle :LF      F1,(R3)
2
3         FMUL   F1,F1,F2
4         ADDI   R3,R3,4
5
6         FADD   F0,F0,F1
7         BNEQ   R3,R5, Boucle
```

Total : 7 cycles par itération de la boucle

PROGRAMME ASSEMBLEUR

Soit l'extrait de programme C suivant :

```
int x, y ;

 main () {
while (x !=y) {
          if (x > y)  x=x-y ;
```

```
        else  y=y-x ; }
}
```

On suppose que la variable x est dans le registre R1 et la variable y dans le registre R2.

Q10) Ecrire le programme assembleur correspondant en utilisant les instructions MIPS fournies en annexe (branchements non retardés).

```
LOOP :  BEQ R1,R2, FIN          // teste si R1=R2
        SUB R1,R1,R2
        BGEZ R1, LOOP           // on boucle si R1 est >= R2)
        ADD R1, R1,R2           // annule l'effet de SUB
        SUB  R2,R2, R1
        BEQ R0,R0, LOOP         //branchement inconditionnel à LOOP
FIN :
```

ANNEXE

Les figures donnent la liste des instructions disponibles.
La signification des abréviations est la suivante :
IMM correspond aux 16 bits de poids faible d'une instruction.
ZIMM est une constante sur 32 bits, avec 16 zéros suivis de IMM (extension de zéros)
SIMM est une constante sur 32 bits, avec 16 fois le signe de IMM, suivi de IMM (extension de signe)
ADBRANCH est l'adresse de branchement, qui est égale à NCP+ SIMM (NCP est l'adresse de l'instruction qui suit le branchement).
On considère de

ADD	1	ADD rd, rs, rt	rd ← rs + rt (signé)
ADDI	1	ADDI rt, rs, IMM	rt ← rs + SIMM (signé)
ADDIU	1	ADDIU rt, rs, IMM	rt ← rs + SIMM (le contenu des registres est non signé)
ADDU	1	ADDU rd, rs, rt	rd ← rs + rt (le contenu des registres est non signé)
AND	1	AND rd, rs, rt	rd ← rs and rt
ANDI	1	ANDI rt, rs, IMM	rt ← rs and ZIMM
BEQ	1	BEQ rs,rt, IMM.	si rs = rt, branche à ADBRANCH
BGEZ	1	BGEZ rs,IMM.	si rs ≥ 0, branche à ADBRANCH
BGEZAL	1	BGEZAL rs, IMM.	adresse de l'instruction suivante dans R31 si rs ≥ 0, branche à ADBRANCH
BGTZ	1	BGTZ rs,IMM.	si rs > 0, branche à ADBRANCH
BLEZ	1	BLEZ rs,IMM.	si rs ≤ 0, branche à ADBRANCH
BLTZ	1	BLTZ rs,IMM.	si rs < 0, branche à ADBRANCH
BLTZAL	1	BLTZAL rs, IMM.	adresse de l'instruction suivante dans R31. si rs < 0, branche à ADBRANCH
BNEQ	1	BNEQ rs,rt, IMM.	si rs ≠ rt, branche à ADBRANCH
J	1	J destination	Décale l'adresse destination de 2 bits à gauche, concatène aux 4 bits de poids fort de CP et saute à l'adresse obtenue
JAL	1	JAL destination	Même action que J . Range adresse instruction suivante dans R31

JR	1	JR rs	Saute à l'adresse dans rs
LUI	1	LUI rt, IMM	Place IMM dans les 16 bits de poids fort de rt. Met 0 dans les 16 bits de poids faible de rt
LB	2	LB rt, IMM(rs)	Rt $_{7\text{-}0}$ ← MEM8 [rs + SIMM] ; Rt 31-8 ← extension de signe
LBU	2	LBU rt, IMM. (rs)	Rt $_{7\text{-}0}$ ← MEM8 [rs + SIMM] ; Rt 31-8 ← extension de zéros.
LW	2	LW rt, IMM.(rs)	rt ← MEM [rs + SIMM]
OR	1	AND rd, rs, rt	rd ← rs or rt
ORI	1	ANDI rt, rs, IMM	rt ← rs or ZIMM
SLL	1	SLL rd, rt, nb	Décale rt à gauche de nb bits et range dans rd
SLT	1	SLT rd, rs, rt	rd ← 1 si rs < rt avec rs signé et 0 autrement
SLTI	1	SLTI rt, rs, IMM	rt ← 1 si rs < SIMM avec rs signé et 0 autrement
SLTIU	1	SLTIU rt, rs, IMM	rt ← 1 si rs < ZIMM avec rs non signé et 0 autrement
SLTU	1	SLTU rt, rs, rt	rd ← 1 si rs < rt avec rs et rt non signés et 0 autrement
SRA	1	SRA rd, rt, nb	Décaler (arithmétique) rt à droite de nb bits et ranger dans rd
SRL	1	SRL rd, rt, nb	Décaler (logique) rt à droite de nb bits et ranger dans rd.
SUB	1	SUB rd, rs, rt	rd ← rs - rt (signé)
SUBU	1	SUBU rd rs, rt	rd ← rs - rt(non signé)
SW	1	SW rt, IMM.(rs)	rt => MEM [rs + IMM]
XOR	1	XOR rd, rs, rt	rd ← rs xor rt
XORI	1	XORI rt, rs, IMM	rt ← rs xor ZIMM

Figure 1 : Instructions entières MIPS utilisées (NB : les branchements ne sont pas retardés)

LF	2	LF ft, IMM(rs)	rt ← MEM [rs +SIMM]
SF	1	SF ft, IMM.(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)

Figure 2 : Instructions flottantes ajoutées (Ce ne sont pas les instructions MIPS)