Université Paris Sud 2008-2009
IFIPS-Info-2

ARCHITECTURE DES ORDINATEURS
Corrigé - Examen Février 2009
2 H – Tous documents autorisés

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

PROGRAMMES ASSEMBLEUR

Soit la fonction en langage assembleur (jeu d'instructions ARM) ci-dessous, dans laquelle les registre R0 et R1 contiennent des entiers non signés.

```
FONC  CMP    R0,R1
      SUBGT  R0,R0,R1
      SUBLT  R1,R1,R0
      BNE    FF
      MOV R15,R14
```

Q 1) Donner le code C correspond à la fonction. Que fait cette fonction ?

```
Unsigned int FF (unsigned int R0, unsigned int R1)
{
While (R0 !=R1)
{
If (R0>R1) R0= R0-R1;
Else R1=R1-R0); }
Return R0
}
```

Calcul du PGCD (algorithme d'Euclide)

Q 2) Ecrire la fonction FONC en utilisant le jeu d'instruction MIPS

```
FONC  SLTU R3, R1,R2 // R3 = 1 si R1<R2
      BGT R3, Suite
      SUB R2,R1,R1
      BNE R1,R2, FONC
      JMP R31
Suite :  SUB R1,R1,R2
      BNE R1,R2, FONC
      JMP R31
```

PIPELINES

On suppose que le processeur utilisé a les pipelines suivants :

Instructions entières : 5 étages
LI LR EX MEM ER
Instructions de chargement flottant (LF) : 5 étages

LI LR EX MEM EF

Instructions flottantes (addition ou multiplication) : 8 étages

LI DI LF EX1 EX2 EX3 EX4 EF

avec la signification suivante :
LI : lecture des instructions dans le cache instructions
DI : décodage des instructions
LR : lecture registres entiers
LF : lecture des registres flottants
EX : exécution UAL pour les entiers, et calcul des adresses (mémoire et branchements)
EXi : phase d'une exécution flottante
MEM : accès au cache données
ER : Écriture registres entiers.
EF : Ecriture registres flottants
Tous les "bypass" nécessaires existent.

Q 3) Donner les latences entre instruction source et instruction destination. La latence est de n si l'instruction source démarre au cycle c et l'instruction destination démarre au cycle c+n.

a) quelle est la latence d'une instruction UAL entière lorsqu'elle est suivie d'une autre instruction UAL ?

LI LR **EX** MEM ER
LI LR **EX** MEM ER
Latence = 1

b) quelle est la latence d'une instruction de chargement de nombre flottant (LF) lorsqu'elle est suivie par une opération flottante (addition ou multiplication) ?

LI LR EX **MEM** EF
LI DI LF **EX1** EX2 EX3 EF
Latence = 1

c) quelle est la latence d'une instruction flottante (addition ou multiplication) lorsqu'elle est suivie par une autre instruction flottante ?

LI DI LF EX1 EX2 EX3 **EX4** EF
LI DI LF **EX1** EX2 EX3 EX4 EF
Latence = 4

d) quelle est la latence des instructions de branchement du type BNEZ ?

LI LR **EX** MEM ER
LI LR EX MEM ER
Latence = 3

e) quelle est la latence des instructions de branchement de type JR ?

LI **LR** EX MEM ER
LI LR EX MEM ER

Latence = 2

TEMPS D'EXECUTION DE BOUCLES

Soit le programme P1

```
        ADDI R2,R1,1000H
Boucle: LF F2, 0(R1)
        LF F3, 1000H (R1)
        FMUL F2,F2,F0
        FMUL F3,F3,F1
        FADD F3,F3,F2
        SF F3, 2000H(R1)
        ADDI R1,R1,4
        BNE  R1,R2,Boucle
```

Q 4) Quel le temps d'exécution, en nombre de cycles d'horloge, de la boucle de P1 optimisé (mais sans déroulage de boucle)

1 boucle	LF **F2**, 0(R1)
2	LF **F3**, 1000_H (R1)
3	FMUL F2,**F2**,F0
4	FMUL ***F3***,**F3**,F1
5	ADDI R1,R1,4
6	
7	
8	FADD **F3**,*F3*,F2
9	
10	
11	
12	SF **F3**, 2000_H(R1)
13	BNE R1,R2,Boucle

13 cycles/itération

Q 5) Quel est le temps d'exécution de la boucle de P1 optimisé avec un déroulage de boucle d'ordre 4 (4 itérations de la boucle initiale par corps de boucle) ?

1 boucle	LF **F2**, 0(R1)
2	LF F4, 4(R1)
3	LF F6, 8(R1)
4	LF F8, C_H(R1)
5	LF **F3**, 1000_H (R1)

6	LF **F5**, 1004_H (R1)
7	LF **F7**, 1008_H (R1)
8	LF **F9**, $100C_H$ (R1)
9	FMUL F2,**F2**,F0
10	FMUL F4,**F4**,F0
11	FMUL F6,**F6**,F0
12	FMUL F8,**F8**,F0
13	FMUL **F3**,**F3**,F1
14	FMUL **F5**,**F5**,F1
15	FMUL **F7**,**F7**,F1
16	FMUL **F9**,**F9**,F1
17	FADD F3,F3,F2
18	FADD F5,F5,F2
19	FADD F7,F7,F2
20	FADD F9,F9,F2
21	SF **F3**, 2000_H(R1)
22	SF **F5**, 2004_H(R1)
23	SF **F7**, 2008_H(R1)
24	SF **F9**, $200C_H$(R1)
25	ADDI R1,R1,16_{10}
26	BNE R1,R2,Boucle

26 cycles/4 itérations soit 6,5 cycles par itérations

Q 6) Le processeur ayant 32 registres flottants, quel est le facteur maximal de déroulage et quel est alors le nombre de cycles par itération de la boucle initiale ?

Les registres F0 et F1 sont communs à toutes les itérations. Chaque itération utilisant deux registres, le facteur de déroulage maximal est donc 15
Chaque itération utilise 3 instructions mémoire (2LF et 1SF), 3 opérations flottantes (2 FMUL et FADD. Il y a deux instructions de gestion de boucle (ADDI et BNE)
Le temps d'exécution est donc 15 x 6 + 2 = 92 cycles/15 itérations = 6,13 cycles/itération.

CACHES.

On suppose que le processeur utilisé a un cache données de 32 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.

Soit le programme C suivant qui correspond au programme P1.

X[0] est rangé à l'adresse $F0000000_H$

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

Q 7) Dans le cas d'un cache à correspondance directe, quels sont les numéros de bloc dans lesquels sont rangés les éléments X[0], Y[0] et Z[0] ? Quel est le nombre de défauts de cache par itération ?

&X[0] = F000 0000 &Y[0] = F000 1000 &Z[0] = F000 2000
64 octets par bloc : 6 bits.
Le cache a 32K/64 = 512 blocs

&X[0] = 1111 0000 0000 0000 0 [000 0000 00 | 00 0000
&Y[0] = 1111 0000 0000 0000 0 [001 0000 00 | 00 0000
&Z[0] = 1111 0000 0000 0000 0 [010 0000 00 | 00 0000

X[0] va dans le bloc 0, Y[0] va dans le bloc 64 et Z[0] va dans le bloc 128
Chaque bloc contient 16 floats.
Il y a donc 3 échecs toutes les 16 itérations, soit 3/16 échecs par itération

Q 8) Pour quelle valeur de N (puissance de 2) aurait-on 3 défauts par itération avec ce cache ?

Il y aurait 3 défauts par itération si X[0], Y[0] et Z[0] allaient tous dans le même bloc 0.
Ce serait le cas si &Y[0] = &Y[0] = 1111 0000 0000 0000 1 [000 0000 00 | 00 0000 = F000 8000_H
Soit N= 8192 = 2^{13}

SIMD

On utilise les #define suivants pour le jeu d'instructions SIMD IA-32

#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 16 x 8 bits non signés
#define minbu(a,b)	_mm_min_epu8(a,b)	min 16 x 8 bits non signés
#define addh(a,b)	_mm_add_epi16(a,b)	Addition 8 x 16 bits signée
#define addhu(a,b)	_mm_add_epu16(a,b)	addition 8 x 16 bits non signée
#define subh(a,b)	_mm_sub_epi16(a,b)	Soustraction 8 x 16 bits signée
#define subbu (a,b)	_mm_subs_epu8(a,b)	Soustraction 8 bits non signés avec saturation

Q 9) donner le programme C correspondant au programme IA-32 suivant

```
_m128i a, b, c, d ;
int i ;
   for(i=0; i<32; i++) {
      a = ld16(&XS[i]);
      b = ld16(&YS[i]);
      c = subbu (a,b);
      d = subbu (b,a);
      c= maxbu (c,d);
      st16(&A[i], c)}
```

```
unsigned char X[N], Y[N];
int i;
for (i=0; i<512;i++)
      if (X[i]>Y[i]
            A[i] = X[i] - Y[i];
            Else A[i] =Y[i] - X[i];
```

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)

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_{7-0} ← MEM8 [rs + SIMM] ; Rt 31-8 ← extension de signe
LBU	2	LBU rt, IMM. (rs)	Rt_{7-0} ← 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	4	FADD fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
FMUL	4	FMUL fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)
FSUB	4	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)