ARCHITECTURE DES ORDINATEURS
Examen Décembre 2005 (CORRIGÉ)
3H – Tous documents autorisés

Exécution d'instructions

On utilise le jeu d'instructions MIPS décrit en annexe.

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

a) Mettre à zéro le contenu du registre R1
b) Mettre 00008000_H dans le registre R2
c) Mettre $F000A000_H$ dans le registre R3
d) Diviser par 2 le contenu du registre R4
e) Multiplier par 17 le contenu du registre R5
f) Multiplier par 7 le contenu du registre R6
g) Multiplier par 119 le contenu du registre R7
h) Mettre à zéro les cases mémoire entre les adresses F0000000 et $F000FFFF_H$

```
a)      XOR R1,R1,R1 ou ADD R1, R0, R0, ou etc
b)      ORI R2, R0, 0x8000      // Résultat faux si ADDI (extension de signe)
c)      LUI R3, 0xF000
        ORI R3,R3, 0xA000       // Résultat faux si ADDI (extension de signe)
d)      SRA R4, R4, 1
e)      SLL R6, R5, 4    // 4 R5
        ADD R5, R6, R5 //4R5+R5 = 5R5
f)      SLL R7, R6, 3    //8R6
        SUB R6, R7, R6 //8R6-R6 = 7R6
g)      119=17*7
        SLL R8, R7, 4
        ADD R7, R8, R7          // 17R7
        SLL R8, R7, 3
        SUB R7, R8, R7
   Autre solution : 119= 128 -8-1
        SLL R8,R7, 7 // 128R7
        SLL R9,R7,3 // 8R7
        SUB R8,R8,R7
        SUB R7, R8, R7
h)      LUI R1, 0x F000H //R1= F0000000H
        LUI R2, 0xF001H  // R2 =F0010000H (adresse suivant F000FFFFH)
L1 :    SW R0, (R1)
        ADDI R1, R1, 4
        BNEQ R1, R2, L1
```

Implantation mémoire

Soit la déclaration de variables C suivante

```
int a, b;
float x[16], y[16], t[64][64];
short i, j, k;
double z[4] ;
char nom[8];
```

Q 2 : Si l'on suppose que la variable a est à l'adresse 1000 0000H, donnez les adresses hexadécimales des variables x[0], y[0], t[0][0], t[1][0], i , z[0], nom[0] et nom[7]

Variable	Adresse en décimal	Adresse En hexa
a	0	0
b	4	4
x0	8	8
y0	72	48
t0	136	88
t1	392	188
i	16520	4088
j	16522	408A
k	16524	408C
Z0	16528	4090
nom0	16560	40B0
nom7	16567	40B7

x[0]	1000 0008
y[0]	1000 0048
t[0][0]	1000 0088
t[1][0]	1000 0188
i	1000 4088
z[0]	1000 4090
nom[0]	1000 40B0
nom[7]	1000 40B7

MICROARCHITECTURES ET TEMPS D'EXECUTION DE PROGRAMMES.

Soient les processeurs non pipelinés (figure 1) et pipelinés (figure 2).

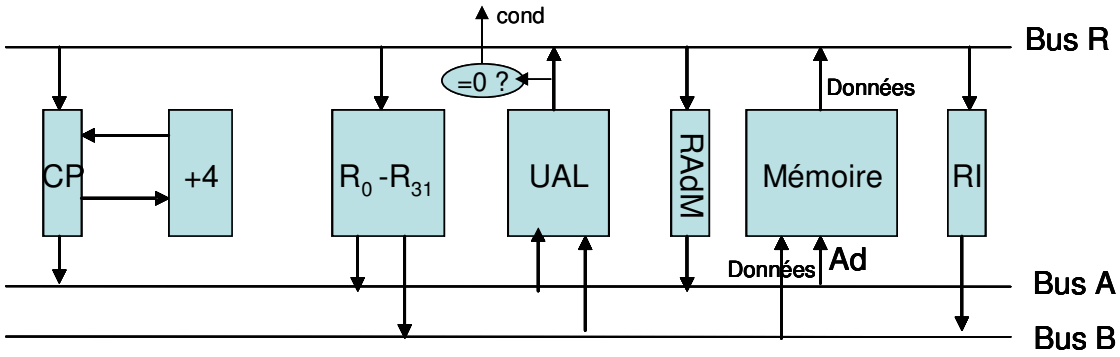

Figure 1 : microarchitecture non pipelinée.

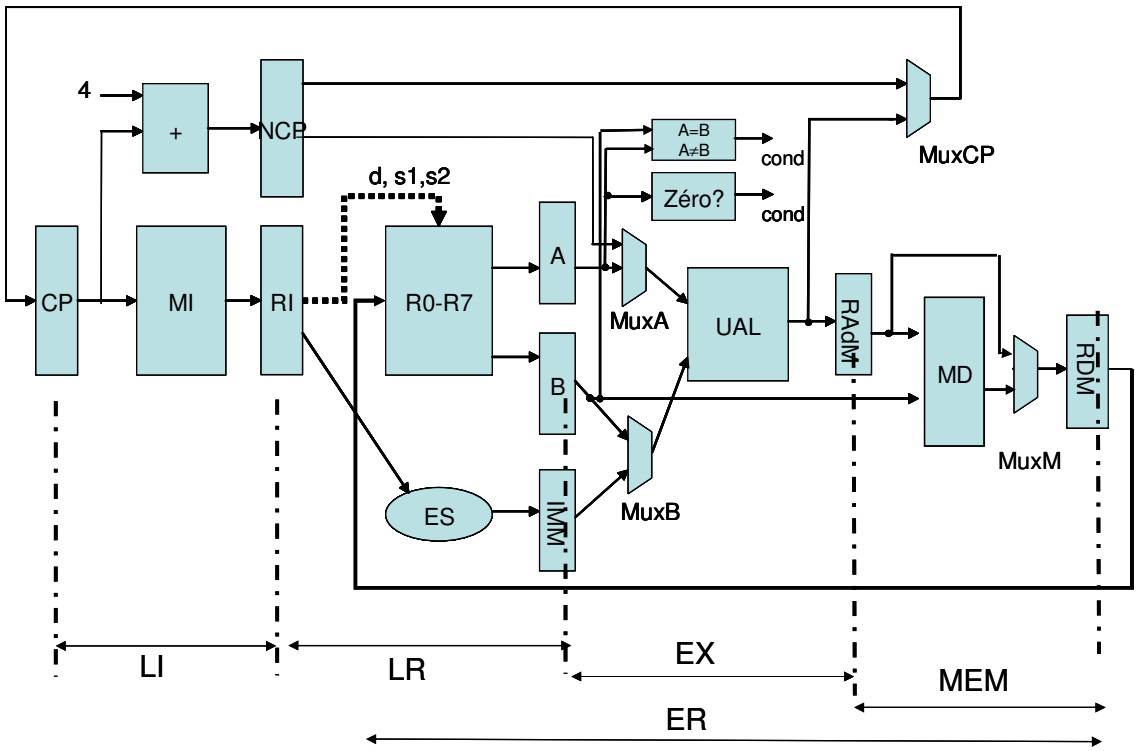

Figure 2 : Microarchitecture pipelinée

Q 3 : Avec le processeur non pipeliné de la Erreur ! Source du renvoi introuvable.**, donner le temps d'exécution de chacune des instructions suivantes :**

(a) ADD Rd,Ra,Rb
(b) ADDI Rd,Ra,Imm
(c) SW Rd, Imm(Ra)
(d) LW Rd, Imm(Ra)
(e) BA Imm où BA Imm signifie : CP <- NCP + ES(Imm**)**

(f) BNEQ Ra, Rb, Imm, dans les cas où Ra=Rb et dans le cas Ra≠Rb

	1	2	3		Total
ADD Rd,Ra,Rb	**RI ← M(CP)** **CP ← CP+4**	**Rd ←Ra+Rb**			**2**
ADDI Rd,Ra,Imm	**RI ← M(CP)** **CP ← CP+4**	**Rd ← Ra+SIMM**			**2**
SW Rd, Imm(Ra)	**RI ← M(CP)** **CP ← CP+4**	**RAdM ← Ra + SIMM**	**Mem (RAdM) ← Rb**		**3**
LW Rd, Imm(Ra)	**RI ← M(CP)** **CP ← CP+4**	**RAdM ← Ra + SIMM**	**Rb ← Mem (RAdM)**		**3**
BA Imm	**RI ← M(CP)** **CP ← CP+4**	**CP ← Ra + SIMM**			**2**
BNEQ Ra, Rb, Imm Ra=Rb	**RI ← M(CP)** **CP ← CP+4**	**Ra - Rb**			**2**
BNEQ Ra, Rb, Imm Ra≠Rb	**RI ← M(CP)** **CP ← CP+4**	**Ra - Rb**	**CP ← Ra + SIMM**		**3**

Q 4 : Même question avec le processeur pipeliné de la Erreur ! Source du renvoi introuvable.**, en supposant qu'aucune anticipation (bypass) n'est implantée. Les instructions après un branchement et qui ont commencé à s'exécuter quand le branchement est pris sont annulées.**

Dans le cas d'un processeur pipeliné, la latence des instructions est de 1, sauf dans le cas où il y a des dépendances de données ou de contrôle.
Sans bypass, les dépendances de données interviennent pour
- ADD et ADDI suivies d'une instruction qui utilise leur résultat

LI LR EX MEM ER
LI LR

Latence : 4 cycles

- Il n'y a pas de dépendances de données sur SW
- LW suivie d'une instruction qui utilise le résultat

LI LR EX MEM ER
LI LR

Latence : 4 cycles

Sans bypass, il y a dépendance de contrôle pour les instructions BA et BNEQ. L'adresse de branchement est calculée dans EX. Les deux instructions qui suivent le branchement sont annulées et la latence est alors de 3 cycles.

Q 5 : Même question avec le processeur pipeliné de la Erreur ! Source du renvoi introuvable.**, en supposant que toutes les anticipations possibles (bypass) sont implantées. Les instructions après un branchement et qui ont commencé à s'exécuter quand le branchement est pris sont annulées.**

Dans le cas d'un processeur pipeliné, la latence des instructions est de 1, sauf dans le cas où il y a des dépendances de données ou de contrôle.

Avec bypass, les dépendances de données interviennent pour
- ADD et ADDI suivies d'une instruction qui utilise leur résultat

LI LR EX MEM ER
LI LR EX MEM ER

Latence : 1 cycle
- Il n'y a pas de dépendances de données sur SW
- LW suivie d'une instruction qui utilise le résultat

LI LR EX MEM ER
LI LR EX MEM ER

Latence : 2 cycles

Sans bypass, il y a dépendance de contrôle pour les instructions BA et BNEQ. L'adresse de branchement est calculée dans EX. Les deux instructions qui suivent le branchement sont annulées et la latence est alors de 3 cycles.

Q 6 : Quel est le nombre de cycles par itération pour le programme assembleur MIPS ci-dessous avec :

(a) le processeur non pipeliné de la Figure 1 ;
(b) le processeur pipeliné de la Figure 2, en supposant que toutes les anticipations (bypass) possibles sont implantées et le même comportement des branchements qu'en Q4 et Q5.

Pour le processeur non pipeliné, le temps d'exécution du programme est la somme du temps d'exécution des instructions individuelles
Pour le processeur pipeliné, le temps d'exécution dépend des dépendances de données et de contrôle. L'exécution pipelinée est indiquée dans la table ci-dessous

	1	2	3	4	5	6	7	8	9	10	11			
LW R11, (R1)	LI	LR	EX	MEM	ER									
LW R12, (R2)		LI	LR	EX	MEM	ER								
ADD R11,R11,R12			LI	LR	-------	EX	MEM	ER						
SW R11, (R3)					LI	LR	EX	MEM	ER					
ADDI R1,R1,4						LI	LR	EX	MEM	ER				
ADDI R2,R2,4							LI	LR	EX	MEM	ER			
ADDI R3,R3,4								LI	LR	EX	MEM	ER		
BNEQ R1,R4, Boucle									LI	LR	EX	MEM	ER	
Annulé										~~LI~~	~~LR~~	~~EX~~	~~MEM~~	~~ER~~
Annulé											~~LI~~	~~LR~~	~~EX~~	~~MEM~~
LW R11, (R1)												LI	LR	EX

Programme	Processeur non pipeliné	Processeur pipeliné avec bypass
Boucle :LW R11, (R1)	3	1
LW **R12**, (R2)	3	2 (dependance de données)
ADD **R11**,R11,**R12**	2	1
SW **R11**, (R3)	3	1
ADDI **R1**,R1,4	2	1
ADDI R2,R2,4	2	1
ADDI R3,R3,4	2	1
BNEQ **R1**,R4, Boucle	3	3
Total	20	11

Optimisation de programmes flottants.

Le processeur utilisé a le jeu d'instructions MIPS (avec branchements non retardés) donné en annexe. Tous les branchements sont parfaitement prédits (s'exécutent en 1 cycle)
La latence des instructions est définie de la manière suivante : une instruction i a une latence de n si l'instruction suivante peut commencer au cycle i+n ; une latence de 1 signifie que l'instruction suivante peut commencer au cycle suivant.
Les instructions flottantes ont les latences suivantes :

Instructions	Latence	Pipelinée ?
LF	2	OUI
FADD, FSUB	3	OUI
FMUL	5	OUI
FDIV	15	NON

Soient les programmes P1, P2 et P3 :

P1	P2	P3
float X[N], Y[N], Z[N] ; int i ; for (i=0 ; i<N ; i++) Z[i]= X[i] + Y[i] ;	float X[N], Y[N], Z[N], a ; int i ; for (i=0 ; i<N ; i++) Z[i]= X[i] /a + Y[i] ;	int i ; float X[N], Y[N], Z[N], a, c ; c =1.0/a ; for (i=0 ; i<N ; i++) Z[i]= X[i] *c + Y[i] ;

Q 7 : Ecrire la version optimisée sans déroulage de boucle du programme P1. Quel est le nombre de cycles par itération et le nombre de cycles total du programme si N=100 ?

Q 8 : Ecrire la version optimisée du programme P1 avec un déroulage de boucle d'ordre 2. Quel est le nombre de cycles par itération et le nombre de cycles total si N=100 ?

Q 9 : Quelle est la version la plus rapide entre le programme P2 et P3 (justifier). Pour la version la plus rapide,

a) écrire la version optimisée sans déroulage de boucle , donner le nombre de cycles par itération et le nombre de cycles total si N=100

b) écrire la version optimisée avec déroulage de boucle d'ordre 2, le nombre de cycles par itération et le nombre de cycles total si N=100.

P1 optimisé sans déroulage de boucle	P1 optimisé avec déroulage de boucle
ADDI R4, R1,400	ADDI R4, R1,400
Boucle : LF F1, (R1)	Boucle : LF F1, (R1)
LF **F2**, (R2)	LF F3, (R1+4)
ADDI R1,R1,4	LF **F2**, (R2)
FADD **F1**, F1, **F2**	LF **F4**, (R2+4)
ADDI R2, R2, 4	FADD **F1**,F1,**F2**
ADDI R3, R3, 4	FADD F3,F3, **F4**
SF **F1**, (R3-4)	ADDI R1, R1, 8
BNEQ R1, R4, Boucle	SF **F1**, (R3)
	SF **F3**, (R3+4)
	ADDI R2, R2,8
	ADDI R3, R3,8
	BNEQ R1,R3, Boucle

Programme P1 sans déroulage de boucle :
8 cycles/itération. Total : 801 cycles si N=100

Programme P1 avec déroulage de boucle :
12 cycles/2 itérations soit 6 cycles/itérations. Total : 601 cycles si N=100

La version P3 est plus rapide que la version P2 car elle utilise une multiplication **pipelinée** ayant une latence de 5 cycles au lieu d'une division **non pipelinée** ayant une latence de 15 cycles

Programme P2 :
12 cycles /itération. Total 1200 + 21 cycles = 1221 cycles

P3 sans déroulage de boucle	P3
LF F0, un	LF F0, un
LF F10, a	LF F10, a
ADDI R4, R1,400	ADDI R4, R1,400
FDIV **F10**,F0,F10 // délai 15 cycles	FDIV **F10**,F0,F10 // délai 15 cycles
Boucle : LF F1, (R1)	Boucle : LF F1, (R1)
LF F2, (R2)	LF F3, (R1+4)
FMUL **F1**, **F10**,F1	FMUL F1,F1, **F10**
ADDI R1,R1,4	FMUL F3,F3,F10
ADDI R2, R2, 4	LF F2, (R2)
	LF F4, (R2+4)

	ADDI R1, R1, 8
FADD F2, **F1**, F2	FADD F1, F1, F2
ADDI R3, R3, 4	FADD F3, F3, F4
	ADDI R2, R2,8
SF F2, (R3-4)	SF F1, (R3)
BNEQ R1, R4, Boucle	SF F3, (R3+4)
	ADDI R3, R3,8
	BNEQ R1,R3, Boucle

Programme P3 non déroulé :
12 cycles par itération
Si N= 100, alors
12 x 100 +16 cycles (initial) = 1216 cycles

Programme P3 déroulé :
14 cycles pour 2 itérations soit 7 cycles par itération :
7 x 100 + 16 cycles (initial) = 716 cycles.

Caches.

On suppose que le processeur utilisé a un cache données de 8 Ko, avec des blocs de 64 octets. Il utilise l'écriture simultanée (write through) non allouée.
On suppose que les tableaux X[N], Y[N], Z[N] et le scalaire a sont rangés à partir de l'adresse hexadécimale 1000 0000H.

Q 10 : Quelles sont les adresses de X[0], Y[0] et Z[0] ? Quel est le nombre total de défaut de caches lors de l'exécution du programme P1 quand N=100 pour les deux cas suivants : a) correspondance directe, b) associativité deux voies (deux blocs par ensemble) ?

X[0] = 1000 0000 H
Y[0] = 1000 0190 H
Z[0] = 1000 0320 H
Les blocs ont 64 octets (6 bits). Il y a 8 Ko/64 = 128 blocs

On a donc 6 bits (adresse dans le bloc). En correspondance directe, il y a 7 bits d'index et 19 bits d'étiquettes.
En correspondance directe, X[0] va dans le bloc 0 et Y[0] va dans le bloc 6. 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 14. (2 * (100/16 +1)
En associatif 2 voies, il n'y a jamais de conflit. 14 défauts de cache.

Q 11 : Quel est le nombre total de défauts de cache pour le programme P1 si N=2048 pour les deux cas : a) correspondance directe, b) associativité 2 voies.

Si N=512
Alors X[0]= 1000 0000H
Y[0]= 1000 2000H

En correspondance directe, les deux vont dans le même bloc et il y a 2 défauts de caches par itération. Soit un total de 4096 défauts
En associatif deux voies, il y a 2 échecs toutes les 16 itérations (1/8) soit un total de 256 défauts.

ENTREES-SORTIES

On dispose d'un clavier et d'un écran.
Dans le contrôleur clavier, le mot d'état est à l'adresse ETATIN et le mot de données à l'adresse DATAIN. Le bit « caractère prêt » est b2.
Dans le contrôleur écran, le mot d'état est à l'adresse ETATOUT et le mot de données à l'adresse DATAOUT. Le bit « prêt à émettre » est b1.

Q 12 : Que fait le programme suivant ?

```
L1 :    LB R1, ETATIN
        ANDI R1,R1, 4
        BEQ R1,R0, L2
        LB R1, DATAIN
        SB R1, (R2)
        ADDI R2,R2,1
L2      LB R3, ETATOUT
        ANDI R3,R3,2
        BEQ R3,R0,L2
        SB R1, DATAOUT
        XORI R4, R1, 0xD // caractère ASCII du retour chariot
        BNEQ R4, R0, L1
```

Le programme lit un caractère tapé au clavier, le range en mémoire à partir de l'adresse contenue initialement dans R2 et imprime le caractère à l'écran. Le processus continue jusqu'à la frappe d'un retour chariot.

ADD	R	ADD rd, rs, rt	rd ← rs + rt (signé)
ADDI	I	ADDI rt, rs, IMM	rt ← rs + SIMM (signé)
ADDIU	I	ADDIU rt, rs, IMM	rt ← rs + SIMM (le contenu des registres est non signé)
ADDU	R	ADDU rd, rs, rt	rd ← rs + rt (le contenu des registres est non signé)
AND	R	AND rd, rs, rt	rd ← rs and rt
ANDI	I	ANDI rt, rs, IMM	rt ← rs and ZIMM
BEQ	I	BEQ rs,rt, déplac.	si rs = rt, branche à ADBRANCH
BGEZ	I	BGEZ rs,déplac.	si rs ≥ 0, branche à ADBRANH
BGEZAL	I	BGEZAL rs, déplac.	adresse de l'instruction suivante dans R31 si rs ≥ 0, branche à ADBRANH
BGTZ	I	BGTZ rs,déplac.	si rs > 0, branche à ADBRANH
BLEZ	I	BLEZ rs,déplac.	si rs ≤ 0, branche à ADBRANH
BLTZ	I	BLTZ rs,déplac.	si rs < 0, branche à ADBRANH
BLTZAL	I	BLTZAL rs, déplac.	adresse de l'instruction suivante dans R31. si rs < 0, branche à ADBRANH
BNEQ	I	BNEQ rs,rt, déplac.	si rs ≠ rt, branche à ADBRANCH
J	J	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	J	JAL destination	Même action que J . Range adresse instruction suivante dans R31
JALR	R	JALR rs, rd	Saute à l'adresse dans rs. Range adresse instruction suivante dans rd
JR	R	JR rs	Saute à l'adresse dans rs
LUI	I	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
LW	I	LW rt, déplac.(rs)	rt ← MEM [rs + SIMM]
OR	R	AND rd, rs, rt	rd ← rs or rt
ORI	I	ANDI rt, rs, IMM	rt ← rs or ZIMM
SLL	R	SLL rd, rt, nb	Décale rt à gauche de nb bits et range dans rd
SLT	R	SLT rd, rs, rt	rd ← 1 si rs < rt avec rs signé et 0 autrement
SLTI	I	SLTI rt, rs, IMM	rt ← 1 si rs < SIMM avec rs signé et 0 autrement
SLTIU	I	SLTIU rt, rs, IMM	rt ← 1 si rs < ZIMM avec rs non signé et 0 autrement
SLTU	R	SLTU rt, rs, rt	rd ← 1 si rs < rt avec rs et rt non signés et 0 autrement
SRA	R	SRA rd, rt, nb	Décaler (arithmétique) rt à droite de nb bits et ranger dans rd
SRL	R	SRL rd, rt, nb	Décaler (logique) rt à droite de nb bits et ranger dans rd.
SUB	R	SUB rd, rs, rt	rd ← rs - rt (signé)
SUBU	R	SUBU rd rs, rt	rd ← rs - rt(non signé)
SW	I	SW rt, déplac.(rs)	rt => MEM [rs + IMM]
XOR	R	XOR rd, rs, rt	rd ← rs xor rt
XORI	I	XORI rt, rs, IMM	rt ← rs xor ZIMM

Figure 3 : Instructions entières MIPS utilisées (NB : les branchements ne sont pas retardés)

LF	I	LF ft, déplac(rs)	rt ← MEM [rs +SIMM]
SF	I	SF ft, déplac.(rs)	ft → MEM [rs + SIMM]
FADD	R	FADD fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
FMUL	R	FMUL fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)
FSUB	R	FSUB fd, fs,ft	fd ← fs - ft (soustraction flottante simple précision)
FDIV	R	FDIV fd,fs,ft	fd ← fs / ft (division flottante simple précision)

Figure 4 : Instructions flottantes ajoutées pour le TD (Ce ne sont pas les instructions MIPS)