Corrigé Partiel Architecture des Ordinateurs - 22 Octobre 2014

Tous documents autorisés. Calculatrices autorisées.
Durée 2h

Une justification **concise** sera donnée lorsque c'est nécessaire. Lorsqu'un résultat est demandé en notation hexadécimale, la notation binaire ne sera pas acceptée.

1. Représentation des entiers sur 8 bits [2 pts]

Q1. On note # l'opération effectuée par un additionneur. Pour les opération suivantes, donner le résultat en notation **hexadécimale**, donner la retenue, et indiquer si le résultat est égal à celui de l'opération arithmétique d'addition lorsque les opérandes sont interprété en naturels et en relatifs (complément à deux), suivant le format de la table.

Opération	Résultat	Retenue (0/1)	Correct en naturels (Oui/Non)	Correct en relatifs (Oui/Non)
0x24 # 0xA3	0xC7	0	Oui	Oui
0x24 # 0x48	0x6C	0	Oui	Oui
0xE4 # 0xAF	0x93	1	Non	Oui
0x72 # 0x37	0xA9	0	Oui	Non

Justification :

- Correct en naturels = Retenue à 0
- Correct en relatifs

Opération	
0x24 # 0xA3	Opérandes de signe contraire, toujours correct
0x24 # 0x48	Opérandes de même signe, bit de signe = retenue
0xE4 # 0xAF	Même signe, bit de signe = retenue
0x72 # 0x37	Même signe, bit de signe <> retenue

2. Jeu d'instructions MIPS [8 pts]

Pour Q2, Q3, Q4, les résultats seront donnés en notation **hexadécimale**.

Q2. Donner le codage des instructions :

a) ADD R13, R1, R4 0x00246820

b) ADDI R1, R1, 7 0x20210007

c) SW R7, 8(R1) 0xAC270008

d) LUI R4, 0x1289 0x3C041289

Q3. L'état initial des registres est : R2 = 0x81234567 R3 = 0x0000000F

Donner le contenu du registre R1 après l'exécution des instructions :

a) ADD R1, R2, R3 R1= 0x81234576

b) ADDI R1, R2, 0xFFFF R1 = 0x81234566

c) AND R1, R2, R3 R1 = 0x00000007

d) SRL R1, R2, 4 R1 = 0x08123456 (décalage logique, extension à 0)

e) SRA R1, R2, 8 R1 = 0xFF812345 (décalage arithmétique, extension de signe)

f) LUI R1, 0x4567 R1 = 0x45670000

g) SLT R1, R2, R3 R1 = 0x00000001 (0x81234567 < 0x0000000F en signé)

h) SLTU R1, R2, R3 R1 = 0x00000000 (0x81234567 > 0x0000000F en non signé)

Q4. La mémoire est organisée en Big Endian. L'état de la mémoire est donné par la table 1 (au verso). Le registre R1 contient 0x10000000. Donner le contenu du registre R2 après l'exécution des instructions :

a) LW R2, 0(R1) R2 = 0x74992130

b) LW R2 , 4 (R1) R2 = 0x20142215

c) LB R2, 1(R1) R2 = 0xFFFFFF99 (extension de signe)

d) LBU R2, 0(R1) R2 = 0x00000074

Q5. Si R1 contient initialement 3, quelle est la valeur (notation **décimale**) de R2 après l'exécution du fragment de code :

```
SLL R2, 4, R1          # R2 = 16*R1
ADD R2, R2, R1         # R2 = 16*R1 + R1 = 17*3 = 51
```

3. Conditionnelles et Boucles [8 pts]

On utilise le jeu d'instruction MIPS.

Le tableau d'entiers X est implanté à partir de l'adresse 0x10000000, et contient {-1, 2, -3, 4}. L'état initial est : R1 = 0x10000000 R3 = 0x00000000 R4 = 0x00000000

Prog1	*Prog2*

ADDI R5, R1, 16
Deb : LW R2, 0(R1)
BGEZ R2, Plus
ADD R3, R3, R2
B Suite
Plus : ADD R4, R4, R2
Suite : ADDI R1, R1, 4
BNE R1, R5, Deb
	ADDI R5, R1, 16
	Deb : LW R2, 0(R1)
	SLTI R7, R2, 0
	MOVN R7, R6, R2
	MOVZ R7, R6, R0
	ADD R3, R3, R6
	ADDI R1, R1, 4
	BNE R1, R5, Deb

Prog1 commenté :

```
        ADDI    R5, R1, 16      # R5 <- première adresse après la fin du tableau
Deb :   LW      R2, 0(R1)       # R2 <- X[i]
        BGEZ    R2, Plus        # Si X[i] >=0, aller à Plus
        ADD     R3, R3, R2      # Si X[i] <0, R3 <- R3 + X[i]
        B Suite                 # Branchement inconditionnel
Plus :  ADD     R4, R4, R2      # Si X[i] >= 0, R3 <- R3 + X[i]
Suite : ADDI    R1, R1, 4       # i++ pour X
```

```
        BNE     R1, R5, Deb         # Si R1 <> R5, retour boucle
```

Donc le programme somme dans R4 les élements positifs de X et dans R3 les éléments négatifs.

Q6. Donner la valeur (notation **décimale**) des registres R3 et R4 après la première itération de *Prog1*.

R3 = -1, R4 = 0

Q7. Donner la valeur (notation **décimale**) des registres R3 et R4 après l'exécution de *Prog1*.

R3 = -4, R4 = 6

Prog2 commenté :

```
        ADDI    R5, R1, 16          # R5 <- première adresse après la fin du tableau
Deb :   LW      R2, 0(R1)           # R2 <- X[i]
        SLTI    R7, R2, 0           # R7 <- (X[i] <0)
        MOVN    R7, R6, R2          # Si R7 <>0, R6 <- R2
        MOVZ    R7, R6, R0          # Si R7 == 0, R6 <- 0
        ADD     R3, R3, R6          # R3 <-R3 + X[i]
        ADDI    R1, R1, 4           # i++ pour X
        BNE     R1, R5, Deb         # Si R1 <> R5, retour boucle
```

Donc le programme additionne les éléments négatifs de X.

Q8. Donner la valeur (notation **décimale**) des registres R6 et R3 après la première itération de *Prog2*.

R6 = -1

Q9. Donner la valeur (notation **décimale**) du registre R3 après l'exécution de *Prog2*.

R3 = -4

4. Procédures [2 pts]

On considère le fragment de code

```
Foo :   ADDI R10, R31, 4
        JR R10
Prog :  MOVE R1, R0
        JAL Foo
        ADD R1, R1, -1
        ADD R1, R1, 100
```

Q10. Donner la valeur (notation **décimale**) du registre R1 après l'exécution de ce programme.

```
Foo :   ADDI R10, R31, 4    # R10 <- R31 + 4
        JR R10              # PC <-R10
Prog :  MOVE R1, R0         # R1 <-0
        JAL Foo             # R31 <- @instruction suivante, PC <- @Foo
        ADDI R1, R1, -1     # R1 <- -1
        ADDI R1, R1, 100    # R1 <- 100
```

Donc, Foo ne revient pas à l'instruction qui suit l'appel de procédure, mais à l'instruction suivante et R1 = 100.

5. Mémoire [2 pts]

On considère la déclaration C suivante :

```
char c;
int y[2];
short x[3];
char* z;
```

Q11. Le place est aligné et en big endian. Les variables sont allouées dans l'ordre à partir de l'adresse 0x00001000. Donner l'adresse de la variable y[0], de la variable x[1] et de la variable z.

```
adresse de y[0] = 0x00001004
adresse de x[1] = 0x0000100E
adresse de z = 0x00001014
```

Adresse	Valeur
0x10000000	0x74
0x10000001	0x99
0x10000002	0x21
0x10000003	0x30
0x10000004	0x20
0x10000005	0x14
0x10000006	0x22
0x10000007	0x15

Table 1 – Plan mémoire de la question 4