CHAP II Microprocesseur MIPS R3000 Architecture externe

Exercice 1:

a. Initialiser le registre $a1 à 0
b. Initialiser le registre $a1 à 0x4567 (valeur 16 bits)
c. Initialiser le registre $a1 à 0x4567ABCD (valeur 32 bits)

Exercice 2:

Programmez en assembleur MIPS la fonction F = 2*X + 2*Y - Z .
On suppose que la variable X se trouve dans le registre $t0, la variable Y dans $t1, la variable Z dans $t2 et que le résultat de la fonction F se trouve dans le registre $v0.

Exercice 3:

1. Ecrire une séquence d'instructions assembleur pour permuter le contenu de 2 registres, On utilisera un registre intermédiaire.
2. Permuter deux registres sans utiliser un registre intermédiaires (penser à XOR).
3. Ecrire une séquence d'instructions assembleur qui permute le contenu de 2 variables 32 bits. Même chose pour deux variables 16 bits, puis 8 bits.

Exercice 4:

Programmez en assembleur MIPS les expressions (variables à 32 bits):

1. $x = y + z - 2 * t$
2. $S = a * x^2 + b * x + c$
3. $S = a^2 + b^2 + c^2$
4. $(x + 5 - y) * 35 / 3$

Exercice 5:

a. Supposez que $a0 = 0x1234 et $a1 = 0x3. Quelle est la valeur du registre $t0 après l'exécution du programme MIPS suivant ?

```
        add     $t0, $zero, $zero
toto:   add     $t0, $t0, $a0
        addi    $a1, $a1, -1
        bne     $a1, $zero, toto
        srl     $t0, $t0, 4
```

b. Supposez le code MIPS suivant, qui reçoit deux entrées dans les registres $t2 et $t3 et produit une sortie dans le registre $s4.

```
        add     $t1, $zero, $zero
lo:     beq     $t3, $zero, fin
        add     $t1, $t1, $t2
        subi    $t3, $t3, 1
        j       lo
fin:    addi    $t1, $t1, 100
        add     $s4, $t1, $zero
```

a) Décrivez en une phrase la fonction du programme.
b) Quelle est la valeur de $s4 à la fin du programme, si $t2 = 4 et $t3 = 6 au début du programme?

Exercice 6: Opérations logiques bit à bit.

Pour chaque question, donner la réponse en langage machine MIPS.

1- Multiplier $t0 par 8.
2- Inverser le bit 4 de $t0.
3- Si $t0 est pair, $t0 = 0, sinon $t0 = 1.
4- Mettre à un le bit 7 de $t0.

Exercice 7:

Programmez en assembleur MIPS les fonctions suivantes:

a) PGCD(A, B) = B si A mod B = 0
= PGCD(B , A mod B) sinon

b) PPCM(A, B) = PPCM(A', B') = A' si A' = B'
= PPCM(A' + A, B') si A' < B'
= PPCM(A', B'+ B) si B' < A'
au départ A' = A , B' = B

c) F_n = 1 si n = 0 ou n = 1
= $F_{n-1} + F_{n-2}$ si n > 1

d) A * B = 0 si B = 0
= (2*A) * (B div 2) si B est pair
= (2*A) * (B div 2) + A si B est impair

e) A^b = 1 si b = 0
= $A^{(b \text{ div } 2)} * A^{(b \text{ div } 2)}$ si b est pair
= $A^{(b \text{ div } 2)} * A^{(b \text{ div } 2)} * A$ si b est impair

Exercice 8:

Ecrire des fonctions en MIPS pour :

1. Vérifier si un nombre est valable dans un système de numération à base B
2. Transformer un nombre entier d'une base B en nombre entier sur 16 bits
3. Ecrire un nombre entier 16 bits dans une base B

Le nombre entier est lu sous forme de chaîne de caractères (pour les bases > 10, on utilise les chiffres A, B, C, D, E, F)

Exercice 9:

Programmez en assembleur MIPS un programme qui lit une phrase terminée par un '.' au clavier et qui et qui affiche (en pascal) le nombre de mots.

Exercice 10:

Programmez en assembleur MIPS un programme qui lit une phrase terminée par un '.' au clavier et qui et qui affiche (en pascal) le nombre de mots qui se termine par la lettre 's'.

Exercice 11 :

Programmer la fonction **copy** une chaine de caractères d'un emplacement mémoire à un autre.

Série N° 02 :

Soit le modèle de Von Neumann simplifier soit :

1/- Si la taille de MC est 2Go et la taille d'un mot mémoire est de 32 bits

Quelle est la taille du :

- Bus d @
- Bus de données
- des Registres

2/ Dérouler les instruction suivants :

- ADD R_0, R_1, R_2
- lw R_0, X // X est une @

1. MC = 2Go

mot = 32 bit

1 bit → 4Ø

nombre de mot = $N = \frac{2G\emptyset}{(32/4)} = \frac{2^{31}}{2^3} = 2^{28}$ mots

taille Bus @ = taille R@M

contenu ordinale → $C\emptyset = \log_2 2^{28} =$ 28 bits

1/2. taille Bus de données = $t_{RIM} = t_{Registre} = t_{ACC} =$ 32 bits

2. ADD R_0, R_1, R_2

($R_0 := R_1 + R_2$)

C Ø ⟶ R@M

lect

RIM ⟶ RI

Analyse (Sequenteur)

R_1 ⟶ RTUAL

R_2 ⟶ ACC

ADD (+)

ACC ⟶ R_0 , CØ ++

Série N°02 :

Exercice N°1 :

a : Init $a1 à 0

$a1 := 0

- Move $a1, $zero
- Sub $a1, $t0, $t0 ($a1 := $t0 - $t0)
- Xor $a1, $a1, $a1

b : $a1 := 0X4567

Move $a1, 0X567

- li $a1, 0X4567 // valeur immédiate
- add $a1, $zero, 0X4567 ($a1 := 0 + 0X4567)

Sub $a1, $zero, -0X4567

... (Sub $a1, 0X4567, $zero) faux

l : Load : Regi ← mémoire

s : store : Regi → mémoire

Exercice N°02 :

```
# Programme Mips qui calcule
# F = 2x + 2y - z
# x --> $t0
# y --> $t1
# z --> $t2
# F --> $v0
.data
    x: .word 10
    y: .word 20
    z: .word -10
    F: .word
.text
main:
    lw  $t0, X
    lw  $t1, y
    lw  $t2, z
    add $a0, $t0, $t2   # x+x --> 2x
    add $a1, $a0, $t1   # 2x+y
    add $a2, $a1, $t1   # 2x+2y
    Sub $v0, $a2, $t2   # 2x+2y-z
    sw  $v0, F
    Move $a0, $v0
    li  $v0, 1
    Syscall.
    li  $v0, 10 # termine le progm
    Syscall.
```

EX003 :

1) Permutation entre 2 Registre

$a0	$a1
5	6
6	5

- text

main :

```
move $t0, $a0
move $a0, $a1
move $a1, $t0
li $v0, 10
syscall
```

EX004 :

2) $S = ax^2 + bx + c$

- data

```
a: .word 10
b: .word -5
c: .word 1
mess1: .asciiz "Donnez a,b,c \n"
mess2: .asciiz "Donnez x \n"
```

('10' → 13)

- text

main :

vars	Allocation Reg
a	$t0
b	$t1
c	$t2
S	$v0
x	$t3

```
la $a0, mess1   // affiche   (// adress)
li $v0, 4       // mess1     (4 : afficher chaine de caractère)
syscall
li $v0, 5       // lire(a)
syscall
move $t0, $v0
li $v0, 5       // lire(b)
syscall
move $t1, $v0
li $v0, 5       // lire(c)
syscall
move $t2, $v0

la $a0, mess2   // affiche
li $v0, 4       // mess2
syscall

li $v0, 5       // lire(x)
syscall
move $t3, $v0

mul $a0, $t3, $t3   // x^2
mul $a0, $a0, $t3
mul $a1, $t1, $t3
add $v0, $a0, $a1
add $v0, $v0, $t2
move $a0, $v0
li $v0, 1
syscall
li $v0, 10
syscall
```

$a0, $v0 → obligatoire pour l'affichage

$v0, 1 → Pour afficher un entier

Exo 05 :

```
add   $t0, $zero, $zero
toto: add  $t0, $t0, $a0
      addi $a1, $a1, -1
      bne  $a1, $zero, toto
      srl  $t0, $t0, 4
```

(تفرع / not equale → bne ; si vrais → toto ; faux → srl)

décallag adroite a 4 bits

$a0 = 0X1234
$a1 = 0X3

$a0	$a1	$t0
0X1234	3	Ø
	2	X1234 + 0X1234 + 0X1234
	1	
	0	0X369C
	0000	0 011 0110 1001 1100 → 4 (srl)

```
$t0 := 0;
Repeat
    $t0 = $t0 + $a0 ;
    $a1 := $a1 - 1 ;
while ($a1 == 0)
```

* le programe multiplier $a0 par $a1 en utiliser l'adition est diviser p 2^4

$t1	$t2	$t3	$s4
~~0~~	4	~~6~~	
~~4~~		~~5~~	
~~8~~		~~4~~	
~~12~~		~~3~~	
~~16~~		~~2~~	
~~20~~		~~1~~	
24 + 100		0	124

programe (2) multiplier $t2 par $t3 puis ajouter 100 au resultad

```
$t1 := 0 ;
while ($t3 ≠ 0)
    $t1 := $t1 + $t2 ;
    $t3 -- ;

$t1 := $t1 + 100
$s4 := $t1
```

Exo 01 :

$$PGCD(A,B) = \begin{cases} B \text{ si } A \bmod B = 0 \\ PGCD(B, A \bmod B) \end{cases}$$

Simon

* PGCD(A, B) ← A mod B

```
Read (A);
Read (B);
while ( A mod B ≠ 0 )
    C = A;
    A = B;
    B = C mod B;
fin
write ("PGCD(A,B)=", B)
Fin
```

A	B	C	A mod B
~~15~~	4		3
4	3	~~1~~	1
3	1	4	0

var	allocation
A	$a0
B	$a1
C	$t0

```
.data
    mess1 : .asciiz "Donne a et b \n"

.text
main : la $a0, mess1
       li $V0, 4
       Syscall
       li $V0, 5      // lire (A)
       Syscall
       move $a0, $V0
       li $V0, 5      // lire (B)
       Syscall
       move $a1, $V0
while : rem $t1, $a0, $a1      (mod)
       // st1 : $a0 mod $a1
       beq $t1, $zero, Endw
       move $t0, $a0
       move $a0, $a1
    # rem $a1, $t0, $a1
       move $a1, $t1
       j // jumpe

endw :
       move $a0, $a1
       li $V0, 1
       Syscall
       li $V0, 10
       Syscall
```

```
# Prgme permettant
# d'inverse les element
# d'un tableau
#
.data
   tab   .word   1, -1, 3, 4
   tab'  .word   0, 0, 0, 0
   en    .space  24
   taille .word  6
.text
   la    $s0, tab
   lw    $t0, taille

   Subi  $t5, $t0, 1     // taille - 1
   mul   $t5, $t5, 4     // octets
   add   $s1, $s0, $t5   // $s1, point le dernier
   li    $t1, 0;            element de $s0
   div   $t2, $t0, 2     //sra
for: bge  $t1, $t2, E For
   lw    $t3, 0($s1)
   lw
```