TP 3 : Architecture des ordinateurs

.

Durée : 2 semaines

L'objectif de ces TPS est double. A l'issue de cette série des TPs les étudiants seront capables d'écrire des procédures en langage d'assemblage MIPS.

Exercice 1

Dans cet exercice nous introduisons le mécanisme de base de la procédure d'appel dans MIPS. Nous voulons, d'une part, voir en détail ce qui se produit lorsqu'une procédure (*appelante)* appelle une autre procédure (*appelée)*, et d'autre part quand le contrôle et renvoyé à la procédure (appelante) par l'appelée. Nous utilisons les instructions (jal et jr).

Les 04 étapes à suivre pour exécuter une procédure sont :

a. Sauvegarde de l'adresse de retour (qui est l'adresse de l'instruction juste après le point d'appel).
b. Appel de la procédure.
c. Exécution de la procédure.
d. restauration de l'adresse de retour.

L'architecture MIPS fournit deux instructions qui sont utilisées conjointement pour effectuer l'appel tout en en sauvegardant l'adresse de retour.

jal nom_procedure

L'instruction (jal) suavegarde dans le registre $**ra l'adresse de l'instruction** qui la suit puis elle effectue le saut à l'instruction étiquetée par (nom_procédure).

Q 1 : Admettons que la procédure appelée appelles aussi une autre procédure, est ce que l'adresse de retour à la première procédure appelante sera maintenue ?

Q2 : Que peut-on faire dans ce cas ?

Q3 : Ecrire le code suivant :

```
add   $sp, $sp, -4        # initialisation du registre 29
sw    $ra, 4($sp)         # sauvegarde du contenu du registre 31 dans la pile M[4 + $sp]
```

Q4 : Au lieu d'ajouter une valeur immédiate négative à **$sp** on peut soustraire une valeur positive de ce registre. Quelle instruction peut faire la même opération ? Réécrire ce code et refaire son assemblage. Vérifier le contenu des registres $ra et $sp.

Q5 : Son faire l'assemblage et l'exécution de ce fragment de code, calculer l'adresse mémoire 8($sp) et 4($sp).

```
li   $a0, 3
li   $a1, 6
add $sp, $sp, -8          # on veut ajouter deux mots, c-à-d. 8 bytes
sw $a0, 8($sp)            # rangement de $s0
sw $a1, 4($sp)            # rangement de $s1
```

Q6 : Donner l'instruction d'initialisation du registre $sp dans le cas où on veut ranger trois paramètres : $s0, $s1 et $s2 dans la pile.

Q7 : On définit la fonction «*moyenne*» qui fait appel à la fonction «*sum*» avec la contrainte que ($x \leq y$) de la façon suivante :

void main (){ *int x, y ;* *printf("Entrer le premier entier \n") ;* *scanf("%d", &x);* *printf("Entrer le deuxième entier \n ") ;* *scanf(" %d", &y) ;* *sum(x,y) ;* *}*	*Int sum(int p, int q) {* *long s =0;* *while (p≤q) {* *s+ = p;* *p++;* *}* *return s;}*

- Ecrire le code d'assemblage MIPS. Avec l'affectation des registres et la réservation mémoire suivante :

Registres	**Mémoires**
$a1 ← *x* $a2 ← *y* $t0 et $t1 deux registres de travail de la fonction appelée $t3 ← *s*	16($sp) ← $ra 12($sp) ← $a1 8($sp) ← $a2 4($sp) ← $t3

- Vérifier le contenu le contenu de la mémoire à l'adresse 12($sp), 8($sp), 4($sp).

Solution

```
########################## Segment data ######################################
.data
msg1: .asciiz"Entrer le premier entier\n"
msg2: .asciiz"Entrer le deuxieme entier\n"
########################## Segment text ######################################

.text

            ############# Fonction appelante ########
main:

   li $v0, 4                  # Entrer le premier entier
   la $a0, msg1
   syscall
   li $v0, 5
   syscall
   move $a1, $v0              # transfert du contenu de $v0 vers $a1


   li $v0, 4                  # Entrer le deuxième entier
   la $a0, msg2
   syscall
   li $v0, 5
   syscall
   move $a2, $v0            # transfert du contenu de $v0 vers $a2

   add    $sp, $sp,-12        # on veut ranger deux mots dans la pile, c-à-d. 8 bytes
   sw     $a1,8($sp)          # rangement de $a1. 12($sp) est réservée au registre $ra
   sw     $a2,4($sp)          # rangement de $a2
   jal          sum
```

```
    li      $v0,1
    lw      $a0,0($sp)
    syscall
    add     $sp,$sp,12        # ajustement de l'état de la pile

###################### Fin du programme ###################################
    li $v0, 10
    syscall

###################### Fonction appelée ###################################
sum:
    sw $ra,12($sp)
    li $t3,0
    lw  $t0,8($sp)
    lw  $t1,4($sp)
while:
    sub  $t7,$t0,$t1
    bgtz $t7,fin
    add  $t3,$t3,$t0
    addi $t0,$t0,1
    sw   $t3,0($sp)
    j    while

fin:
    lw $ra,12($sp)
    jr $ra
```