TP assembleur MIPS

David Delahaye

David.Delahaye@lirmm.fr

Faculté des Sciences

Master M1 2019-2020

Programmation en MIPS

Outils

- Utilisation d'un émulateur ;
- Plusieurs émulateurs : MARS (celui qu'on utilisera), SPIM ;
- Télécharger MARS (lien sous Moodle) ;
- Exécuter l'émulateur : `java -jar Mars4_5.jar`.

Principe

- Écriture de son programme dans l'éditeur ;
- Assemblage du programme (génération du binaire) ;
- Exécution du programme (possibilité de faire du pas à pas) ;
- Vue sur l'état de la mémoire et des registres.

À vous de jouer !

Exercices

- Demander la saisie d'un entier et rendre la valeur absolue de cet entier (afficher le résultat) ;
- Initialiser un tableau de 3 entiers (sans le saisir) et permuter les éléments de ce tableau ;
- Afficher les n premiers entiers (en partant de 1), où l'entier n sera demandé à l'utilisateur ;
- Demander la saisie d'un entier et dire si cet entier est pair ou non (afficher le résultat).

Correction

Valeur absolue

```
.data
msg:    .asciiz "Enter an integer: "

.text
main:   li $v0, 4
        la $a0, msg
        syscall
        li $v0, 5
        syscall
        li $t0, 0
        bge $v0, $t0, disp
        neg $v0, $v0
disp:   move $a0, $v0
        li $v0, 1
        syscall
```

Correction

Permutation des éléments d'un tableau

```
.data
array:   .space 12

.text
main:    la $t0, array
         li $t1, 1
         sw $t1, ($t0)
         li $t1, 2
         sw $t1, 4($t0)
         li $t1, 3
         sw $t1, 8($t0)
         lw $t1, ($t0)
         lw $t2, 8($t0)
         sw $t2, ($t0)
         sw $t1, 8($t0)
```

Correction

Affichage des *n* premiers entiers

```
.data
msg:    .asciiz "Enter an integer: "

.text
main:   li $v0, 4
        la $a0, msg
        syscall
        li $v0, 5
        syscall
        li $t0, 1
        move $t1, $v0
for:    bgt $t0, $t1, end
        move $a0, $t0
        li $v0, 1
        syscall
        addi $t0, $t0, 1
        j for
end:
```

Correction

Parité

```
.data
msg:    .asciiz "Enter an integer: "
meven:  .asciiz "Even"
modd:   .asciiz "Odd"

.text
main:   li $v0, 4
        la $a0, msg
        syscall
        li $v0, 5
        syscall
        li $t0, 2
        div $v0, $t0
        mfhi $t0
        bnez $t0, odd
        la $a0, meven
        j disp
odd:    la $a0, modd
disp:   li $v0, 4
        syscall
```

À vous de jouer !

Exercices

- Écrire une routine qui permute le contenu de deux variables entières de la zone de données avec une variable locale pour effectuer la permutation ;
- Écrire le code assembleur correspondant au code C suivant :

```
int sqr (int x) {
  return x * x;
}

int sum (int x, int y) {
  return sqr(x) + sqr(y);
}
```

Correction

Permutation de deux variables

```
.data
x:      .word 1
y:      .word 1

.text
main:   li $t0, 1       # Initialization of the variables
        la $t1, x
        sw $t0, ($t1)
        li $t0, 2
        la $t1, y
        sw $t0, ($t1)

        li $v0, 1       # Display of the variables
        la $t0, x
        lw $a0, ($t0)
        syscall
        li $v0, 1
        la $t0, y
        lw $a0, ($t0)
        syscall
```

Correction

Permutation de deux variables

```
        la $a0, x       # Call of the swap routine
        la $a1, y
        jal swap

        la $t0, x       # Display of the variables (after swapping)
        lw $a0, ($t0)
        syscall
        li $v0, 1
        la $t0, y
        lw $a0, ($t0)
        syscall
        li $v0, 10      # Exit of the program
        syscall
```

Correction

Permutation de deux variables

```
swap:    sub $sp, $sp, 4
         lw $t0, ($a0)
         sw $t0, ($sp)
         lw $t0, ($a1)
         sw $t0, ($a0)
         lw $t0, ($sp)
         sw $t0, ($a1)
         add $sp, $sp, 4
         jr $ra
```

Correction

Appels imbriqués

```
.text
main:    li $a0, 1
         li $a1, 2
         jal sum
         move $a0, $v0
         li $v0, 1
         syscall
         li $v0, 10
         syscall
```

Correction

Appels imbriqués

```
sum:    addiu $sp, $sp, -8
        sw $ra, 4($sp)
        sw $s0, ($sp)
        move $s0, $a1
        jal sqr
        move $a0, $s0
        move $s0, $v0
        jal sqr
        add $v0, $s0, $v0
        lw $ra, 4($sp)
        lw $s0, ($sp)
        addiu $sp, $sp, 8
        jr $ra

sqr:    mul $t0, $a0, $a0
        mflo $v0
        jr $ra
```

À vous de jouer !

Exercices

- Écrire une routine qui effectue récursivement la somme des n premiers entiers, où n est un entier passé en argument ;
- Écrire une routine qui implante la suite de Fibonacci :

$$fib(n) = \begin{cases} n, \text{ si } n = 0, 1 \\ fib(n-1) + fib(n-2), \text{ sinon} \end{cases}$$

Correction

Somme des n premiers entiers

```
.text
main:    li $v0, 5
         syscall
         move $a0, $v0
         jal sum
         move $a0, $v0
         li $v0, 1
         syscall
         li $v0, 10
         syscall
```

Correction

Somme des n premiers entiers

```
sum:    addiu $sp, $sp, -8
        sw $ra, 4($sp)
        sw $s0, ($sp)
        move $s0, $a0
        blez $s0, basis
        addiu $a0, $s0, -1
        jal sum
        add $v0, $s0, $v0
        j exit
basis:  li $v0, 0
exit:   lw $ra, 4($sp)
        lw $s0, ($sp)
        addiu $sp, $sp, 8
        jr $ra
```

Correction

Suite de Fibonacci

```
.text
main:    li $v0, 5
         syscall
         move $a0, $v0
         jal fib
         move $a0, $v0
         li $v0, 1
         syscall
         li $v0, 10
         syscall
```

Correction

Suite de Fibonacci

```
fib:    addiu $sp, $sp, -12
        sw $ra, 8($sp)
        sw $s0, 4($sp)
        sw $s1, ($sp)
        move $s0, $a0
        li $t0, 1
        ble $s0, $t0, basis
        addiu $a0, $s0, -1
        jal fib
        move $s1, $v0
        addiu $a0, $s0, -2
        jal fib
        add $v0, $s1, $v0
        j exit
basis:  move $v0, $s0
exit:   lw $ra, 8($sp)
        lw $s0, 4($sp)
        lw $s1, ($sp)
        addiu $sp, $sp, 12
        jr $ra
```