Université de Nantes
Région des Pays de La Loire

Feuille de travaux dirigés n^o 5
Procédures en assembleur

Exercice 5.1

Écrire en assembleur MIPS la fonction `int impair(int x)` retournant `1` si `x` est impair et `0` sinon.

▽ **Correction**

```
.text
.globl _start

_start:
li $a0, 8
jal impair
move $a0, $v0
jal print_int

# exit()
li    $v0, 10
syscall

#  int impair(int x) {
# return x & 1;
# }
impair:
sub $sp, $sp, 32
sw $ra, 20($sp)
sw $fp, 16($sp)
add $fp, $sp, 32

and $v0, $a0, 1

lw $ra, 20($sp)
lw $fp, 16($sp)
add $sp, $sp, 32
jr $ra
```

Exercice 5.2

Traduire en assembleur la fonction C ci-dessous (`sizeT` indique le nombre de cases du tableau `T`).

```
int somme_tab(int T[], int sizeT)
{
int accu = 0;
for (int i=0; i < sizeT; ++i) {
```

```
accu += T[i];
}
return accu;
}
```

▽ **Correction**

```
.text
.globl _start

_start:
la $a0, T
li $a1, 4
jal somme_tab

move $a0, $v0
jal print_int

# exit()
li    $v0, 10
syscall


.data
T: .word 23, -6, 4, 10


.text

# int somme_tab(int T[], int sizeT)
# {
# int accu = 0;
# for (int i=0; i < sizeT; ++i) {
# accu += T[i];
# }
# return accu;
# }
somme_tab:
sub $sp, $sp, 32
sw $ra, 20($sp)
sw $fp, 16($sp)
add $fp, $sp, 32

li $v0, 0  # accu
li $t1, 0  # i
for:
bge $t1, $a1, endfor

sll $t2, $t1, 2   # Calcul de l'offset dans T à partir de i
add $t2, $a0, $t2 # adresse de T[i]
lw $t2, 0($t2)    # t2 <- T[i]
add $v0, $v0, $t2

add $t1, $t1, 1
b for
endfor:

lw $ra, 20($sp)
lw $fp, 16($sp)
```

```
add $sp, $sp, 32
jr $ra
```

Exercice 5.3

Écrire entièrement en assembleur la fonction récursive `int poire(int a, int b)` définie de la façon suivante (les règles s'appliquent dans leur ordre d'apparition) :

$$\text{poire}(x,y) = \begin{cases} y+1 & \text{si } x = 0 \\ \text{poire}(x-1,1) & \text{si } y = 0 \\ \text{poire}(x-1,\text{poire}(x,y-1)) & \text{sinon} \end{cases}$$

On respectera les conventions de passage de paramètres vues en cours. On considérera qu'un `int` est codé sur 32 bits.

▽ Correction

```
.text
.globl _start

_start:
li $a0, 2
li $a1, 7
jal poire
move $a0, $v0
jal print_int

# exit()
li    $v0, 10
syscall


# int poire(int x, int y) {
# if (x == 0) {
# return y+1;
# }
# if (y == 0) {
# return poire(x-1,1);
# }
# return poire(x-1,poire(x,y-1));
# }
poire:
sub $sp, $sp, 32
sw $a1, 28($sp)
sw $a0, 24($sp)
sw $ra, 20($sp)
sw $fp, 16($sp)
add $fp, $sp, 32

if1:
bnez $a0, endif1
move $v0, $a1
add $v0, $v0, 1
b return
endif1:
if2:
```

```
bnez $a1, endif2
sub $a0, $a0, 1
li $a1, 1
jal poire
b return
endif2:
sub $a1, $a1, 1
jal poire   # poire(x,y-1)
move $a1, $v0
sub $a0, $a0, 1
jal poire
return:
lw $a1, 28($sp)
lw $a0, 24($sp)
lw $ra, 20($sp)
lw $fp, 16($sp)
add $sp, $sp, 32
jr $ra
```

Exercice 5.4

Écrire en assembleur la fonction récursive `facto` calculant la factorielle d'un entier sur 32 bits. On utilisera les conventions de passage des paramètres vues en cours. On supposera que le résultat est toujours représentable sur 32 bits.

▽ **Correction**

```
.text
.globl _start

_start:
li $a0, 5
jal facto
move $a0, $v0
jal print_int

# exit()
li    $v0, 10
syscall



# int facto(int n) {
# if (n <= 1) {
# return 1;
# } else {
# return n*facto(n-1);
# }
# }
facto:
sub $sp, $sp, 32
sw $a0, 24($sp)
sw $ra, 20($sp)
sw $fp, 16($sp)
add $fp, $sp, 32

if:
```

```
bgt $a0, 1, else
then:
li $v0, 1
b endif
else:
sub $a0,$a0, 1
jal facto
lw $t0, 24($sp) # Récupération du paramètre original
mul $v0, $t0, $v0
endif:

lw $a0, 24($sp)
lw $ra, 20($sp)
lw $fp, 16($sp)
add $sp, $sp, 32
jr $ra
```

Exercice 5.5

Écrire en assembleur MIPS la procédure `void minmax(int a, int b, int c, int d, int *m, int *M)` prenant en entrée quatre entiers sur 32 bits et retournant dans les deux derniers paramètres passés par adresse le minimum et le maximum de ces quatre nombres.

▽ Correction

```
.text
.globl _start

_start:
sub $sp, $sp, 8 # réservation pour variables locales
li $a0, 5
li $a1, -3
li $a2, 7
li $a3, 16

la $t0, 0($sp) # minimum
la $t1, 4($sp) # maximum
sub $sp, $sp, 12 # réservation pour paramètres 5 et 6 (2*4 + 4 libres)
sw $t0, 4($sp)
sw $t1, 8($sp)
jal minmax

add $sp, $sp, 12 # récupération mémoire des paramètres 5 et 6

la $a0, ministr
jal print_string
lw $a0, 0($sp)
jal print_int
la $a0, cr
jal print_string
la $a0, maxistr
jal print_string
lw $a0, 4($sp)
jal print_int
la $a0, cr
jal print_string
```

```
add $sp, $sp, 8 # Nettoyage de la pile

# exit()
li    $v0, 10
syscall

.data
ministr: .asciiz "Minimum : "
maxistr: .asciiz "Maximum : "
cr: .asciiz "\n"

.text
#  void minmax(int a, int b, int c, int d, int *m, int *M) {
# int mab, Mab;
# int mcd, Mcd;
# int Mm;
# int mM;
# minmax2(a,b,&mab,&Mab);
# minmax2(c,d,&mcd,&Mcd);
# minmax2(mab,mcd,m,&Mm);
# minmax2(Mab,Mcd,&mM,M);
# }
# Stack:
# | M   |   8
#         |_m___|   4
# $fp ->| $a3 |   0
# | $a2 |  -4
# | $a1 |  -8
# | $a0 | -12
# | $ra | -16
# | $fp | -20
# |   | -24
# |_____| -28
# | mab | -32
# | Mab | -36
# | mcd | -40
# | Mcd | -44
# | Mm  | -48
# | mM  | -52
# $sp ->|   |
minmax:
sub $sp, $sp, 32
sw $a3, 32($sp)
sw $a2, 28($sp)
sw $a1, 24($sp)
sw $a0, 20($sp)
sw $ra, 16($sp)
sw $fp, 12($sp)
add $fp, $sp, 32

sub $sp, $sp, 24 # Réservation d'espace pour les variables locales

# minmax2(a,b,&mab,&Mab);
lw $a0, -12($fp)
lw $a1, -8($fp)
la $a2, -32($fp) # mab
la $a3, -36($fp) # Mab
jal minmax2
# minmax2(c,d,&mcd,&Mcd);
lw $a0, -4($fp)
lw $a1, 0($fp)
```

```
la $a2, -40($fp) # mcd
la $a3, -44($fp) # Mcd
jal minmax2
# minmax2(mab,mcd,m,&Mm);
lw $a0, -32($fp)
lw $a1, -40($fp)
lw $a2, 4($fp)
la $a3, -48($fp)
jal minmax2
# minmax2(Mab,Mcd,&mM,M);
lw $a0, -36($fp)
lw $a1, -44($fp)
la $a2, -52($fp)
lw $a3, 8($fp)
jal minmax2

add $sp, $sp, 24

lw $a3, 32($sp)
lw $a2, 28($sp)
lw $a1, 24($sp)
lw $a0, 20($sp)
lw $ra, 16($sp)
lw $fp, 12($sp)
add $sp, $sp, 32
jr $ra



#  void minmax2(int a, int b, int *m, int *M) {
# if (a <= b) {
# *m = a;
# *M = b;
# } else {
# *m = b;
# *M = a;
# }
# }
minmax2:
sub $sp, $sp, 32
sw $ra, 20($sp)
sw $fp, 16($sp)
add $fp, $sp, 32

if:
bgt $a0, $a1, else
then:
sw $a0, ($a2)
sw $a1, ($a3)
b endif
else:
sw $a0, ($a3)
sw $a1, ($a2)
endif:

lw $ra, 20($sp)
lw $fp, 16($sp)
add $sp, $sp, 32
jr $ra
```