Architecture des ordinateurs
Corrigé du projet assembleur SPARC

Arnaud Giersch, Benoît Meister et Frédéric Vivien

1 Fonction modulo

Écrire une fonction calculant le reste de la division entière (fonction *modulo*) de son premier argument par le second. Les arguments sont des entiers positifs, le deuxième est non nul. Vous pourrez utiliser la formule suivante :

$$a \bmod b = a - b \times \left\lfloor \frac{a}{b} \right\rfloor$$

où $\left\lfloor \frac{a}{b} \right\rfloor$ est le résultat de la division entière de a par b.

```
.section        ".text"
        .align 4
        .global modulo

modulo:                         ! modulo (a, b): reste (r) de la division
                                !                entière a / b
                                !       a = b.q + r
                                !  <=>  r = a - b.q
                                ! avec  q = a / b
        save %sp, -64, %sp

        mov %g0, %y             ! le premier opérande de udiv est sur 64 bits
        udiv %i0, %i1, %i2      ! %i2 <- q = a / b
        umul %i1, %i2, %i1      ! %i1 <- b.q
        sub %i0, %i1, %i0       ! %i0 <- r = a - b.q

        ret                     ! return %i0
        restore
```

2 Algorithme d'Euclide

2.1 Version itérative

Écrire un programme prenant en entrée deux nombres entiers positifs a et b, et calculant puis affichant le pgcd de ces deux nombres. Le pgcd sera calculé avec une version itérative de l'algorithme d'Euclide :

tant que b $\neq$ 0
  r := a mod b
  a := b
  b := r
fin tant que
pgcd := a

Vous utiliserez la fonction *modulo* écrite à la question 1.

```
.section        ".rodata"
        .align 8

.INPUT_PRINTF:
        .asciz "a b (a >= b)? "

.INPUT_SCANF:
        .asciz "%u %u"

.OUTPUT_PRINTF:
        .asciz "pgcd(%u, %u) = %u\n"


.section        ".text"
        .align 4
        .global main

main:
        save %sp, -104, %sp     ! réserve de l'espace pour 2 variables locales

        set .INPUT_PRINTF, %o0
        call printf             ! printf (.INPUT_PRINTF)
        nop

        set .INPUT_SCANF, %o0
        add %fp, -4, %o1
        add %fp, -8, %o2
        call scanf              ! scanf (.INPUT_SCANF, %fp-4, ,%fp-8)
        nop

        ld [%fp-4], %l0         ! %l0 <- *(%fp-4)
        ld [%fp-8], %l1         ! %l1 <- *(%fp-8)

.while:                         ! # calcul de %l0 mod %l1
        cmp %l1, %g0            ! while (%l1 != 0) {
        be .end_while           !
        nop                     !

        mov %l0, %o0            !
        mov %l1, %o1            !
        call modulo             !    %o0 <- %l0 mod %l1
        nop                     !

        mov %l1, %l0            !    %l0 <- %l1
        mov %o0, %l1            !    %l1 <- %o0

        ba .while               ! }
        nop

.end_while:                     ! # le résultat est dans %l0

        set .OUTPUT_PRINTF, %o0
        ld [%fp-4], %o1
        ld [%fp-8], %o2
        mov %l0, %o3
        call printf             ! printf (.OUTPUT_PRINTF,
                                !         *(%fp-4), *(%fp-8), %l0)
        nop

        clr %i0
        ret                     ! return 0
        restore
```

2.2 Version récursive

Même question qu'en 2.1, mais avec une version récursive de l'algorithme :

pgcd(a, b) := **si** b = 0 **alors** a **sinon** pgcd(b, a mod b)

```
.section        ".text"
        .align 4
        .global pgcd

pgcd:                           ! pgcd (%i0, %i1):  calcul du pgcd
        save %sp, -96, %sp

        cmp %i1, %g0            ! if (b != 0) {
        be .end_pgcd            !
        nop                     !

        mov %i0, %o0            !
        mov %i1, %o1            !
        call modulo             !       %o0 <- %i0 mod %i1
        nop                     !

        mov %o0, %o1            !
        mov %i1, %o0            !
        call pgcd               !       %o0 <- pgcd (%i1, %o0)
        nop                     !

        mov %o0, %i0            !       %i0 <- %o0
.end_pgcd:                      ! }
        ret                     ! return %i0
        restore


.section        ".rodata"
        .align 8

.INPUT_PRINTF:
        .asciz "a b (a >= b)? "

.INPUT_SCANF:
        .asciz "%u %u"
```

```
.OUTPUT_PRINTF:
        .asciz "pgcd(%u, %u) = %u\n"


.section        ".text"
        .align 4
        .global main

main:
        save %sp, -104, %sp     ! réserve de l'espace pour 2 variables locales

        set .INPUT_PRINTF, %o0
        call printf             ! printf (.INPUT_PRINTF)
        nop

        set .INPUT_SCANF, %o0
        add %fp, -4, %o1
        add %fp, -8, %o2
        call scanf              ! scanf (.INPUT_SCANF, %fp-4, ,%fp-8)
        nop

        ld [%fp-4], %o0
        ld [%fp-8], %o1
        call pgcd               ! %o0 <- pgcd (*(%fp-4), *(%fp-8))
        nop

        mov %o0, %o3
        set .OUTPUT_PRINTF, %o0
        ld [%fp-4], %o1
        ld [%fp-8], %o2
        call printf             ! printf (.OUTPUT_PRINTF, *(%fp-4), *(%fp-8), %o0)
        nop

        clr %i0
        ret                     ! return 0
        restore
```