```
# Arquitectura de Computadores I
# Ano lectivo 2011/12
#
# Correcção do teste prático 1 (19/11/2011)
#
# ******************************************************************************
# Questão 2
# ******************************************************************************
#
        .data
array:  .space  40

        .text
        .globl  main

# argc:         $a0
# p:            $s0
# argv:         $a1 -> $s1
# res:          $s2
# array+argc:   $s3
#
main:   subu    $sp, $sp, 20
        sw      $ra, 0($sp)
        sw      $s0, 4($sp)
        sw      $s1, 8($sp)
        sw      $s2, 12($sp)
        sw      $s3, 16($sp)

        move    $s1, $a1        # save argv
        li      $s2, 0          # res = 0
if:     bge     $a0, 10, then   # if(argc >= 10
        blt     $a0, 2, then    #   || argc < 2)
        j       endif           #
then:   li      $v0, -1         #    return -1
        j       stack_restore
endif:  la      $s3, array      #
        sll     $a0, $a0, 2     #
        addu    $s3, $s3, $a0   # $s3=array+argc
        la      $s0, array      #
for:    bgeu    $s0, $s3, endfor# while( p < (array + argc) ) {
        lw      $a0, 0($s1)     #    $a0 = *argv
        jal     proc_string     #
        sw      $v0, 0($s0)     #    *p = proc_string( *argv )
        add     $s2, $s2, $v0   #    res += *p
        addu    $s1, $s1, 4     #    argv++
        addu    $s0, $s0, 4     #    p++
        j       for             # }
endfor: move    $a0, $s2        #
        li      $v0, 1          #
        syscall                 # print_int( res )
        li      $v0, 0          # return 0

stack_restore:
        lw      $ra, 0($sp)
        lw      $s0, 4($sp)
        lw      $s1, 8($sp)
        lw      $s2, 12($sp)
        lw      $s3, 16($sp)
        addu    $sp, $sp, 20
        jr      $ra

# a estrutura condicional pode ser implementada de uma forma mais optimizada

if:     bge     $a0, 10, then   # if(argc >= 10
        bge     $a0, 2, endif   #   || argc < 2)
then:   li      $v0, -1         #    return -1
        j       stack_restore
endif:  la      $s3, array      #
```

```
# ********************************************************************************
# Questão 3
# ********************************************************************************
#
# p:    $a0 -> $s0
# c:    $a1 -> $s1
# n:    $a2 -> $s2
# res:  $s3
#
proc1:  subu    $sp, $sp, 20    #
        sw      $ra, 0($sp)     #
        sw      $s0, 4($sp)     #
        sw      $s1, 8($sp)     #
        sw      $s2, 12($sp)    #
        sw      $s3, 16($sp)    #

        move    $s0, $a0        # save p
        move    $s1, $a1        # save c
        move    $s2, $a2        # save n
        li      $s3, 0          # res=0
do:                             # do {
if:     lb      $t0, 0($s0)     #   $t0 = *p
        bne     $t0, $s1, else  #   if(*p == c)
        move    $a0, $s0        #
        add     $a1, $s1, $t0   #
        jal     proc2           #
        add     $s3, $s3, $v0   #        res += proc2(p, c + *p)
        j       endif           #
else:   add     $t0, $t0, 'A'   #    else
        sub     $t0, $t0, 'a'   #
        sb      $t0, 0($s0)     #        *p = *p + 'A' - 'a'
endif:  sub     $s2, $s2, 1     #    n--
        bgt     $s2, 0, do      # } while(n > 0)
        move    $v0, $s3        # return res

        lw      $ra, 0($sp)     #
        lw      $s0, 4($sp)     #
        lw      $s1, 8($sp)     #
        lw      $s2, 12($sp)    #
        lw      $s3, 16($sp)    #
        addu    $sp, $sp, 20    #
        jr      $ra             #
```

```
# ******************************************************************************
# Questão 4
# ******************************************************************************
#
# list:  $a0
# num:   $a1
# count: $a2
# i:     $t0
# j:     $t1
# k:     $t2
#
array_proc:
        sub     $t1, $a1, 1     # j = num - 1
        li      $t2, 0          # k = 0
        li      $t0, 0          # i = 0
for:    bge     $t0, $a1, endfor# while(i < num) {
if:     sll     $t3, $t0, 2     #
        addu    $t3, $t3, $a0   #    $t3 = &(list[i])
        sll     $t4, $t1, 2     #
        addu    $t4, $t4, $a0   #    $t4 = &(list[j])
        lw      $t5, 0($t3)     #    $t5 = list[i]
        lw      $t6, 0($t4)     #    $t6 = list[j]
        blt     $t5, 200, then  #    if(list[i] < 200
        bge     $t5, $t6, then  #       || list[i] >= list[j])
        j       endif           #    {
then:   xor     $t6, $t5, $t6   #
        sw      $t6, 0($t4)     #       list[j] = list[i] ^ list[j]
        addi    $t2, $t2, 1     #       k++
                                #    }
endif:  addi    $t1, $t1, -1    #    j--
        addi    $t0, $t0, 1     #    i++
        j       for             # }
endfor: sw      $t2, 0($a2)     # *count = k
        move    $v0, $a0        # return list
        jr      $ra

# a estrutura condicional pode ser implementada de uma forma mais optimizada

if:     ...

        blt     $t5, 200, then  #    if(list[i] < 200
        blt     $t5, $t6, endif #       || list[i] >= list[j])
then:   xor     $t6, $t5, $t6   #
        sw      $t6, 0($t4)     #       list[j] = list[i] ^ list[j]
        addi    $t2, $t2, 1     #       k++
                                #    }
endif:  addi    $t1, $t1, -1    #    j--
        addi    $t0, $t0, 1     #    i++
        j       for             # }
```