Αρχιτεκτονική Υπολογιστών

Σειρά Ασκήσεων 1

Ονοματεπώνυμο: Λιαροκάπης Αλέξανδρος
Αριθμός Μητρώου: 03114860

Άσκηση 1

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
                                                        #
                .data                                   #
                                                        #
array:          .word   5, 9, 8, 7, 6, 10, 4, 3, 2, 1   #       uint32_t array[10] = {5, 9, 8, 7, 6, 10, 4, 3, 2, 1}
                                                        #
                .text                                   #
                                                        #
                                                        #       void main():
main:           la      $s1,    array                   #               s1 = &array
                addi    $s3,    $0,     10              #               s3 = 10 // n = 10
                add     $t0,    $zero,  $zero           #               t0 = 0
I_LOOP:         beq     $t0,    $s3,    END             #  I_LOOP:      if t0 == s3 goto END // if (t0 == n) ..
                add     $t1,    $zero,  $zero           #               t1 = 0
J_LOOP:         sub     $t2,    $s3,    $t0             #               t2 = s3 - t0
                addi    $t2,    $t2,    -1              #               t2 = t2 - 1
                beq     $t1,    $t2,    NEXT_I          #  J_LOOP:      if t1 == t2 goto NEXT_I // if (t1 == n - t0 - 1) ..
                sll     $t2,    $t1,    2               #               t2 = t1 << 2
                add     $t2,    $t2,    $s1             #               t2 = s1 + t2
                lw      $t3,    0($t2)                  #               t3 = *((uint32_t*)(t2))   // t3 = array[t1]
                lw      $t4,    4($t2)                  #               t4 = *((uint32_t*)(t2+4)) // t4 = array[t1+1]
                slt     $t5,    $t4,    $t3             #               t5 = t4 < t3
                beq     $t5,    $zero,  NEXT_J          #               if (t5 == 0) goto NEXT_J // if !(t4 < t3) ..
                sw      $t4,    0($t2)                  #               *((uint32_t*)t2) = t4 // array[t1] = array[t1+1]
                sw      $t3,    4($t2)                  #               *((uint32_t*)(t2+1)) = t3 // array[t1+1] = array[t1]
NEXT_J:         addi    $t1,    $t1,    1               #  NEXT_J:      ++t1
                j       J_LOOP                          #               goto J_LOOP
NEXT_I:         addi    $t0,    $t0,    1               #  NEXT_I:      ++t0
                j       I_LOOP                          #               goto I_LOOP
END:            addi    $v0,    $0,     10              #  END:         v0 = 10
                syscall                                 #               system call (exit)
                                                        #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

Άσκηση 2

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
                                                               #
                .data   0x10000000                             #       data starts at address 0x10000000
                                                               #
global_x:       .space  4                                      #       uint32_t x
global_y:       .space  4                                      #       uint32_t y
global_z:       .space  4                                      #       uint32_t z
global_w:       .space  4                                      #       uint32_t w
                                                               #
                .text                                          #
                                                               #       void xorshift_init(a0, a1, a2, a3):
xorshift_init:  sw      $a0,            global_x               #               x = a0
                sw      $a1,            global_y               #               y = a1
                sw      $a2,            global_z               #               z = a2
                sw      $a3,            global_w               #               w = a3
                jr      $ra                                    #               return to ra
                                                               #
                                                               #       uint32_t xorshift():
xorshift:       lw      $t0,            global_x               #               t0 = x1
                sll     $t1,            $t0,            11     #               t1 = t0 << 11
                xor     $t0,            $t0,            $t1    #               t0 = t0 ^ t1
                srl     $t1,            $t1,            8      #               t1 = t1 >> 8
                xor     $t0,            $t0,            $t1    #               t0 = t0 ^ t1
                lw      $t1,            global_y               #               t1 = y
                sw      $t1,            global_x               #               x = t1
                lw      $t1,            global_z               #               t1 = z
                sw      $t1,            global_y               #               y = t1
                lw      $t1,            global_w               #               t1 = w
                sw      $t1,            global_z               #               z = t1
                srl     $t2,            $t1,            19     #               t2 = t1 >> 19
                xor     $t1,            $t1,            $t2    #               t1 = t1 ^ t2
                xor     $t1,            $t1,            $t0    #               t1 = t1 ^ t0
                sw      $t1,            global_w               #               w = t1
                addi    $v0,            $t1,            0      #               v0 = t1
                jr      $ra                                    #               return to ra
                                                               #
                                                               #       void main():
main:           addi    $a0,            $0,             1231   #               a0 = 1231
                addi    $a1,            $0,             6456   #               a1 = 6456
                addi    $a2,            $0,             3453   #               a2 = 3453
                addi    $a3,            $0,             8567   #               a3 = 8567
                jal     xorshift_init                          #               call xorshift_init
                                                               #
                jal     xorshift                               #               call xorshift
                jal     xorshift                               #               call xorshift
                jal     xorshift                               #               call xorshift
                                                               #
                addi    $v0,            $0,             10     #               v0 = 10
                syscall                                        #               system call (exit)
                                                               #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

Άσκηση 3

```
##################################################################################################
                                                                #
                .data   0x10000000                              #       data starts at address 0x10000000
                                                                #
global_x:       .space  4                                       #       uint32_t global_x
global_y:       .space  4                                       #       uint32_t global_y
global_z:       .space  4                                       #       uint32_t global_z
global_w:       .space  4                                       #       uint32_t global_w
                                                                #
global_array:   .word    10, 5, 8, 2, 7, 3, 1, 9, 4, 6          #       uint32_t global_array[] = {10, 5, 8, 2, 7, 3, 1, 9, 4, 6}
                                                                #
                .text                                           #
                                                                #       void xorshift_init(a0, a1, a2, a3):
xorshift_init:  sw      $a0,            global_x                #               global_x = a0
                sw      $a1,            global_y                #               global_y = a1
                sw      $a2,            global_z                #               global_z = a2
                sw      $a3,            global_w                #               global_w = a3
                jr      $ra                                     #               return to ra
                                                                #
                                                                #       uint32_t xorshift():
xorshift:       lw      $t0,            global_x                #               t0 = global_x
                sll     $t1,            $t0,            11      #               t1 = t0 << 11
                xor     $t0,            $t0,            $t1     #               t0 = t0 ^ t1
                srl     $t1,            $t1,            8       #               t1 = t1 >> 8
                xor     $t0,            $t0,            $t1     #               t0 = t0 ^ t1
                lw      $t1,            global_y                #               t1 = global_y
                sw      $t1,            global_x                #               global_x = t1
                lw      $t1,            global_z                #               t1 = global_z
                sw      $t1,            global_y                #               global_y = t1
                lw      $t1,            global_w                #               t1 = global_w
                sw      $t1,            global_z                #               global_z = t1
                srl     $t2,            $t1,            19      #               t2 = t1 >> 19
                xor     $t1,            $t1,            $t2     #               t1 = t1 ^ t2
                xor     $t1,            $t1,            $t0     #               t1 = t1 ^ t0
                sw      $t1,            global_w                #               global_w = t1
                addi    $v0,            $t1,            0       #               v0 = t1
                jr      $ra                                     #               return to ra
                                                                #
                                                                #       void swap(a0, a1):
swap:           lw       $t0,           0($a0)                  #               t0 = *((uint32_t*)a0)
                lw       $t1,           0($a1)                  #               t1 = *((uint32_t*)a1)
                sw       $t0,           0($a1)                  #               *((uint32_t*)a1) = t0
                sw       $t1,           0($a0)                  #               *((uint32_t*)a0) = t1
                jr       $ra                                    #               return to ra
                                                                #
                                                                #       uint32_t partition(a0, a1, a2):
partition:      addi     $sp,           $sp,       -20          #               allocate stack space
                sw       $s0,           0($sp)                  #               store s0
                sw       $s1,           4($sp)                  #               store s1
                sw       $s2,           8($sp)                  #               store s2
                sw       $s3,           12($sp)                 #               store s3
                sw       $ra,           16($sp)                 #               store ra
                addi     $s0,           $a0,       0            #               s0 = a0
                addi     $s1,           $a1,       0            #               s1 = a1
                addi     $s2,           $a2,       0            #               s2 = a2
                sll      $s3,           $s1,       2            #               s3 = s1 << 2
                add      $s3,           $s3,       $s0          #               s3 = s3 + s0
                lw       $s3,           0($s3)                  #               s3 = *((uint32_t*)s3) // s3 = ((uint32_t*)s0)[s1]
                addi     $s1,           $s1,       -1           #               --s1
                addi     $s2,           $s2,       1            #               ++s2
partition_L1:   addi     $s1,           $s1,       1            #  part_L1:     ++s1
                sll      $t0,           $s1,       2            #               t0 = s1 << 2
                add      $t0,           $s0,       $t0          #               t0 = s0 + t0
                lw       $t0,           0($t0)                  #               t0 = *((uint32_t*)t0) // t0 = ((uint32_t*)s0)[s1]
                slt      $t0,           $t0,       $s3          #               t0 = t0 < s3
                bne      $t0,           $0,        partition_L1 #               if t0 != 0 goto part_L1 // if ((uint32_t*)s0)[s1] < s3 ..
partition_L2:   addi     $s2,           $s2,       -1           #  part_L2:     --s2
                sll      $t0,           $s2,       2            #               t0 = s2 << 2
                add      $t0,           $s0,       $t0          #               t0 = s0 + t0
                lw       $t0,           0($t0)                  #               t0 = *((uint32_t*)t0)
                sgt      $t0,           $t0,       $s3          #               t0 = t0 > s3
                bne      $t0,           $0,        partition_L2 #               if t0 != 0 goto part_L2 // if ((uint32_t*)s0)[s2] > s3 ..
                sge      $t0,           $s1,       $s2          #               t0 = s1 >= s2
                bne      $t0,           $0,        partition_end #              if t0 != 0 goto part_end // if s1 >= s2 ...
                sll      $a0,           $s1,       2            #               a0 = s1 << 2
                add      $a0,           $s0,       $a0          #               a0 = s0 + a0 // a0 = &(((uint32_t*)s0)[s1])
                sll      $a1,           $s2,       2            #               a1 = s2 << 2
                add      $a1,           $s0,       $a1          #               a1 = s0 + a1 // a1 = &(((uint32_t*)s0)[s2])
                jal      swap                                   #               call swap
```

```
                j       partition_L1                                #               goto part_L1
                                                                    #
partition_end:  addi    $v0,            $s2,        0               #  part_end:    v0 = s2 // return value
                lw      $s0,            0($sp)                      #               restore s0
                lw      $s1,            4($sp)                      #               restore s1
                lw      $s2,            8($sp)                      #               restore s2
                lw      $s3,            12($sp)                     #               restore s3
                lw      $ra,            16($sp)                     #               restore ra
                addi    $sp,            $sp,        20              #               deallocate stack space
                jr      $ra                                         #               return to ra
                                                                    #
                                                                    #       void quicksort(a0, a1, a2):
quicksort:      addi    $sp,            $sp,        -20             #               allocate stack space
                sw      $s0,            0($sp)                      #               store s0
                sw      $s1,            4($sp)                      #               store s1
                sw      $s2,            8($sp)                      #               store s2
                sw      $s3,            12($sp)                     #               store s3
                sw      $ra,            16($sp)                     #               store ra
                addi    $s0,            $a0,        0               #               s0 = a0
                addi    $s1,            $a1,        0               #               s1 = a1
                addi    $s2,            $a2,        0               #               s2 = a2
                sge     $t0,            $s1,        $s2             #               t0 = s1 >= s2
                bne     $t0,            $0,         quicksort_end   #               if t0 != 0 goto quicksort_end // if s1 >= s2 ..
                jal     xorshift                                    #               call xorshift
                sub     $a0,            $s2,        $s1             #               a0 = s2-s1
                divu    $v0,            $a0                         #
                mfhi    $a0                                         #               a0 = v0 % a0
                add     $a0,            $a0,        $s1             #               a0 = a0 + s1
                sll     $a0,            $a0,        2               #               a0 = a0 << 2
                add     $a0,            $s0,        $a0             #               a0 = s0 + a0 // a0 = &(((uint32_t*)s0)[a0])
                sll     $a1,            $s1,        2               #               a1 = s1 << 2
                add     $a1,            $s0,        $a1,            #               a1 = s0 + a1 // a1 = &(((uint32_t*)s0)[a1])
                jal     swap                                        #               call swap
                addi    $a0,            $s0,        0               #               a0 = s0
                addi    $a1,            $s1,        0               #               a1 = s1
                addi    $a2,            $s2,        0               #               a2 = s2
                jal     partition                                   #               call partition
                addi    $s3,            $v0,        0               #               s3 = v0
                addi    $a0,            $s0,        0               #               a0 = s0
                addi    $a1,            $s1,        0               #               a1 = s1
                addi    $a2,            $s3,        0               #               a2 = s3
                jal     quicksort                                   #               call quicksort
                addi    $a0,            $s0,        0               #               a0 = s0
                addi    $a1,            $s3,        1               #               a1 = s3+1
                addi    $a2,            $s2,        0               #               a2 = s2
                jal     quicksort                                   #               call quicksort
quicksort_end:  lw      $s0,            0($sp)                      #               restore s0
                lw      $s1,            4($sp)                      #               restore s1
                lw      $s2,            8($sp)                      #               restore s2
                lw      $s3,            12($sp)                     #               restore s3
                lw      $ra,            16($sp)                     #               restore ra
                addi    $sp,            $sp,        20              #               deallocate stack space
                jr      $ra                                         #               return to ra
                                                                    #
                                                                    #       void main():
main:           addi    $a0,            $0,         1231            #               a0 = 1231
                addi    $a1,            $0,         3453            #               a1 = 3453
                addi    $a2,            $0,         5675            #               a2 = 5675
                addi    $a3,            $0,         7898            #               a3 = 7898
                jal     xorshift_init                               #               call xorshift_init
                la      $a0,            global_array                #               a0 = &global_array
                addi    $a1,            $0,         0               #               a1 = 0
                addi    $a2,            $0,         9               #               a2 = 9
                jal     quicksort                                   #               call quicksort
                addi    $v0,            $0,         10              #               v0 = 10
                syscall                                             #               system call (exit)
                                                                    #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```