Λύσεις Διαγωνίσματος 4

1Α.

Περιγράψτε τις πιο κάτω εντολές, όπως επίσης και το τελικό αποτέλεσμα που θα επιτευχθεί με την εκτέλεση του πιο κάτω κώδικα. **(2 μονάδες)**

```
la $a0, input    # a0=file descriptor,  περιέχει την διεύθυνση του ονόματος του αρχείου

li $a1, 0        # a1 = άνοιγμα μόνο για διάβασμα

li $a2,0740      # χρήστης όλα τα δικαιώματα, group μόνο για διάβασμα

li $v0, 13       # κλήση συστήματος 13 για άνοιγμα αρχείου

syscall # Οι πιο πάνω εντολές ανοίγουν ένα αρχείο για διάβασμα από τον χρήστη και το group
```

1Β.

Περιγράψτε τις πιο κάτω εντολές, αλλά και το τελικό αποτέλεσμα του κώδικα. **(2 μονάδες)**

```
move $a0, $v0          # a0=file descriptor, ο οποίος επιστράφηκε από το άνοιγμα του αρχείου
la $a1, 0X10008000     # ο $a1 δείχνει στην διεύθυνση μνήμης που θα αποθηκευτεί ο χαρακτήρας
li $a2, 1              # ο $a2 = 1, άρα θα διαβάζουμε ένα – ένα τους χαρακτήρες από το αρχείο
move $s5, $a1          # μετακίνηση του $a1 στον $s5
li $v0, 14             # κλήση συστήματος 14 για διάβασμα αρχείου
syscall

lb $s2, 0($s5) # Ο $s2 θα περιέχει τον ascii κώδικα του χαρακτήρα που διαβάστηκε από το αρχείο
```

1C.

Με τη βοήθεια του Παραρτήματος Α, περιγράψτε τις ενέργειες που χρειάζεται να γίνουν (ποια ορίσματα πρέπει να δοθούν) πριν την κλήση συστήματος για άνοιγμα ενός αρχείου με δικαίωμα διαβάσματος για τον τοπικό χρήστη (local user) και το group στο οποίο ανήκει ο υπολογιστής.

```
la $a0, input    # a0=file descriptor,  περιέχει την διεύθυνση του ονόματος του αρχείου

li $a1, 0        # a1 = άνοιγμα μόνο για διάβασμα

li $a2,0440      # χρήστης και  group έχουν δικαιώματα μόνο για διάβασμα

li $v0, 13       # κλήση συστήματος 13 για άνοιγμα αρχείου

syscall # Οι πιο πάνω εντολές ανοίγουν ένα αρχείο για διάβασμα από τον χρήστη και το group
```

2A.

Με τη βοήθεια του παραρτήματος Α, περιγράψτε τις ενέργειες που χρειάζεται να γίνουν πριν την κλήση συστήματος για διάβασμα ενός αρχείου, χαρακτήρα προς χαρακτήρα. **(2 μονάδες)**

```
move $a0, $v0 # a0=file descriptor, ο οποίος επιστράφηκε από το άνοιγμα του αρχείου
la $a1, 0X10008000 # ο $a1 δείχνει στην διεύθυνση μνήμης που θα αποθηκευτεί ο 1ος χαρακτήρας
li $a2, 1          # ο $a2 = 1, άρα θα διαβάζουμε ένα – ένα τους χαρακτήρες από το αρχείο
li $v0, 14         # κλήση συστήματος 14 για διάβασμα αρχείου
syscall
```

2B.

Με ποιο τρόπο μπορεί να αναγνωριστεί το τέλος ενός αρχείου (.txt) το οποίο διαβάζει ο spim και πως κλείνουμε το αρχείο αυτό; (Γράψτε και τις (3) εντολές σε assembly στην απάντηση σας). **(2 μονάδες)**

Το τέλος ενός αρχείου μπορεί να περιέχει τον ascii κώδικα ενός ειδικού χαρακτήρα που όταν διαβαστεί από το προγραμμα, να τερματίζει το διαβασμα και να κλείνει το αρχείο. Αν για παράδειγμα έχουμε «;» στο τέλος, μπορούμε να γράψουμε:

```
                beq $t2, 59, close_file   # έλεγχος αν ο χαρακτήρας είναι «;»
close file:     li $v0, 16        # κλήση συστήματος για κλείσιμο αρχείου
                syscall
```

2C.

Στις θέσεις μνήμης 0X50000000 ως 0X50000010 υπάρχουν αποθηκευμένοι διάφοροι χαρακτήρες ascii. Γράψτε ένα πολύ σύντομο πρόγραμμα που θα ελέγχει αυτές τις θέσεις μνήμης (μία προς μία) και θα μετρά τις φορές που εντόπισε τον ascii κώδικα του πλήκτρου 3. **(3 μονάδες)**

```
                la $t0, 0X50000000          # initial address
                la $t1, 0X50000010          # end address
                li $t3, 0                   # counter
loop:           lb $s1, ($t0)               # load character
                bne $s1, 0x33, skip_counter # check if character = 0X33
                addi $t3, $t3, 1            # increase counter if it was 0x33
skip_counter:   addi $t0, $t0, 1            # increase pointer to next byte
                bne $t0, $t1, loop          # branch to loop until $t0 = $t1
```

3A.

Γράψτε ένα σύντομο πρόγραμμα το οποίο θα υπολογίζει το 5 παραγοντικό με αναδρομική κλήση (η οποία βασικά θα υπολογίζει: 5*4*3*2*1) χρησιμοποιώντας την εντολή mul. Το πρόγραμμά σας θα αρχίζει με την 1η εντολή jal και θα τερματίζει με την σωστή επιστροφή κάτω από το 1ο jal. (Για το σκοπό της άσκησης, δεν χρειάζεται να γίνονται οι έλεγχοι για τη χρησιμοποίηση της Στοίβας.)
Εάν ΔΕΝ γνωρίζετε πώς να λύσετε αυτή την άσκηση σε assembly, μπορείτε (για τις μισές μονάδες) να περιγράψετε (με βήματα ή ψευδοεντολές) τη διαδικασία που πρέπει να ακολουθηθεί ώστε να τρέχει σωστά ένα πρόγραμμα με αναδρομική κλήση η οποία καλείται αρχικά από το κύριο μέρος του προγράμματος και στη συνέχεια, αφού εκτελεστεί 4 φορές, επιστρέφει κάτω από το 1ο jal (όπως το δείξαμε και στο προηγούμενο εργαστήριο) **(6 μονάδες)**

```
                li $a1, 5
                move $s2, $a1
                jal paragontiko
paragontiko:    bne $a1, 5, skip_push
                addi    $sp, $sp, -4
                sw $ra, ($sp)
skip_push:      addi $a1, $a1, -1
                mul $s2, $s2, $a1
                beqz $a1, exit_jal
                jal paragontiko
exit_jal:       lw $ra, ($sp)
                addi $sp, $sp, 4
                jr $ra
```

3Β.

Γράψτε ένα σύντομο πρόγραμμα το οποίο θα εκτυπώνει στην οθόνη τους αριθμούς 4, 3, 2, και 1 με αναδρομική κλήση. Το πρόγραμμά σας θα αρχίζει με την 1η εντολή jal και θα τερματίζει με την σωστή επιστροφή κάτω από το 1ο jal. (Για το σκοπό της άσκησης, δεν χρειάζεται να γίνονται οι έλεγχοι για τη χρησιμοποίηση της Στοίβας.)

Εάν ΔΕΝ γνωρίζετε πώς να λύσετε αυτή την άσκηση σε assembly, μπορείτε (για τις μισές μονάδες) να περιγράψετε (με βήματα ή ψευδοεντολές) τη διαδικασία που πρέπει να ακολουθηθεί ώστε να τρέχει σωστά ένα πρόγραμμα με αναδρομική κλήση η οποία καλείται αρχικά από το κύριο μέρος του προγράμματος και στη συνέχεια, αφού εκτελεστεί 4 φορές, επιστρέφει κάτω από το 1ο jal (όπως το δείξαμε και στο προηγούμενο εργαστήριο) **(6 μονάδες)**

```
                li $a1, 4
                jal print_num
print_num:      bne $a1, 4, skip_push
                addi    $sp, $sp, -4
                sw $ra, ($sp)
skip_push:      li $v0, 1
                move $a0, $a1
                syscall
                addi $a1, $a1, -1
                beqz $a1, exit_print
                jal print_num
exit_print:     lw $ra, ($sp)
                addi $sp, $sp, 4
                jr $ra
```

3C.

Γράψτε ένα σύντομο πρόγραμμα το οποίο θα υπολογίζει το 2^n (όπου n < 10) με αναδρομική κλήση (η οποία βασικά θα υπολογίζει: 2*2*2...*2) χρησιμοποιώντας την εντολή mul ή την sll $t0, $t0, 1. Το πρόγραμμά σας θα αρχίζει με την $1^η$ εντολή jal και θα τερματίζει με τη σωστή επιστροφή κάτω από το $1^ο$ jal. (Για το σκοπό της άσκησης, δεν χρειάζεται να γίνονται οι έλεγχοι για τη χρησιμοποίηση της Στοίβας.)

Εάν ΔΕΝ γνωρίζετε πώς να λύσετε αυτή την άσκηση σε assembly, μπορείτε (για τις μισές μονάδες) να περιγράψετε (με βήματα ή ψευδοεντολές) τη διαδικασία που πρέπει να ακολουθηθεί ώστε να τρέχει σωστά ένα πρόγραμμα με αναδρομική κλήση η οποία καλείται αρχικά από το κύριο μέρος του προγράμματος και στη συνέχεια, αφού εκτελεστεί τις απαιτούμενες φορές, επιστρέφει κάτω από το $1^ο$ jal (όπως το δείξαμε και στο προηγούμενο εργαστήριο) **(4 μονάδες)**

```
                li $a1, 5               # $a1 holds n
                move $s1, $a1           # copy n in $s1
                jal ypologismos
ypologismos:    bne $a1, $t1, skip_push
                addi $sp, $sp, -4
                sw $ra, ($sp)
skip_push:      addi $a1, $a1, -1
                sll $s1, $s1, 1
                beqz $a1, exit_jal
                jal ypologismos
exit_jal:       lw $ra, ($sp)
                addi $sp, $sp, 4
                jr $ra
```