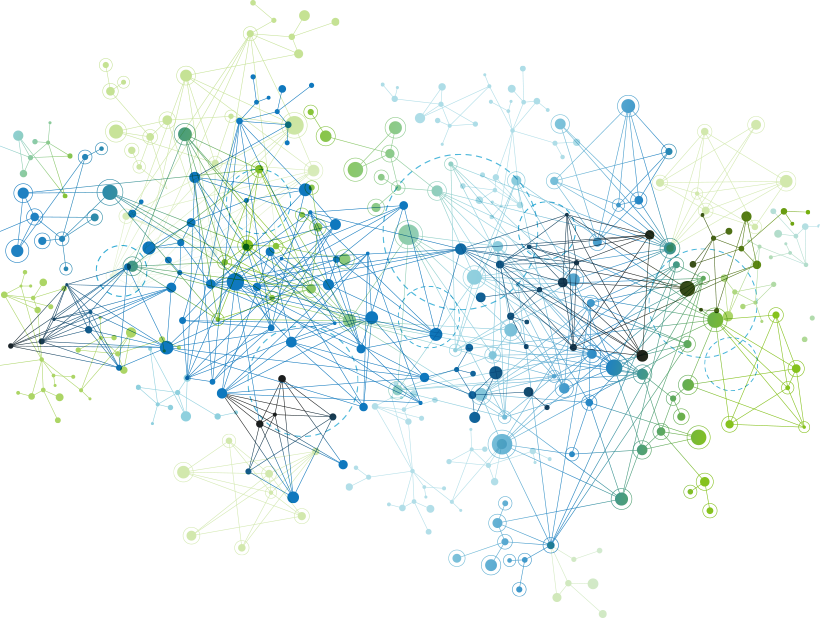
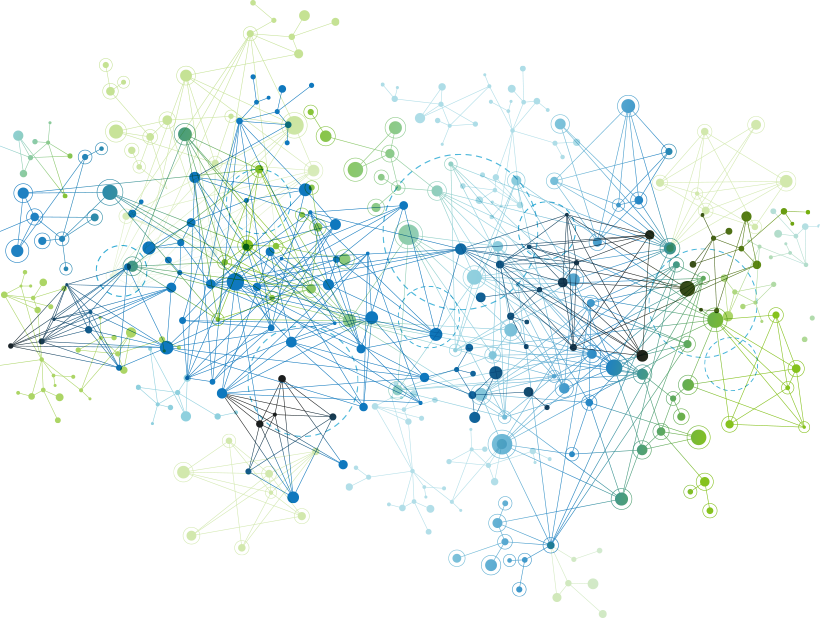
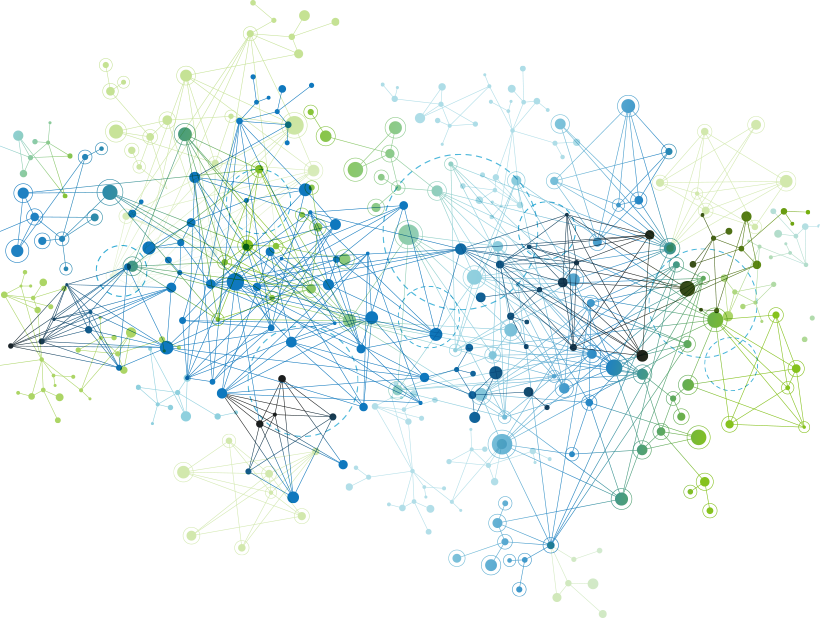
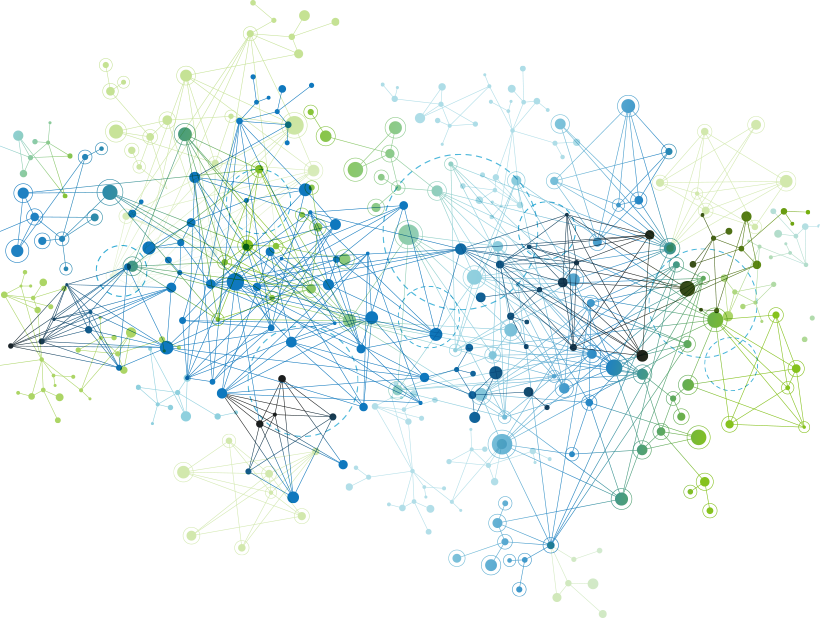
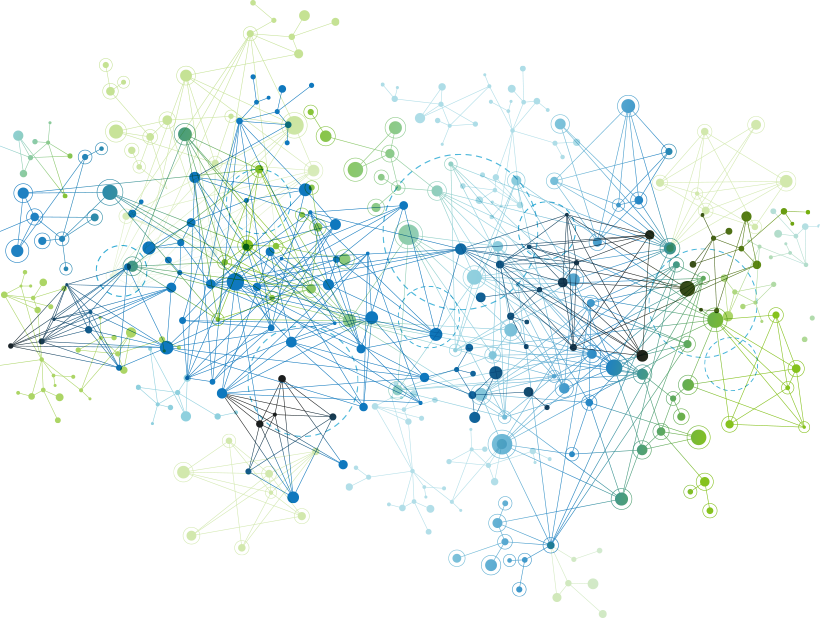
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

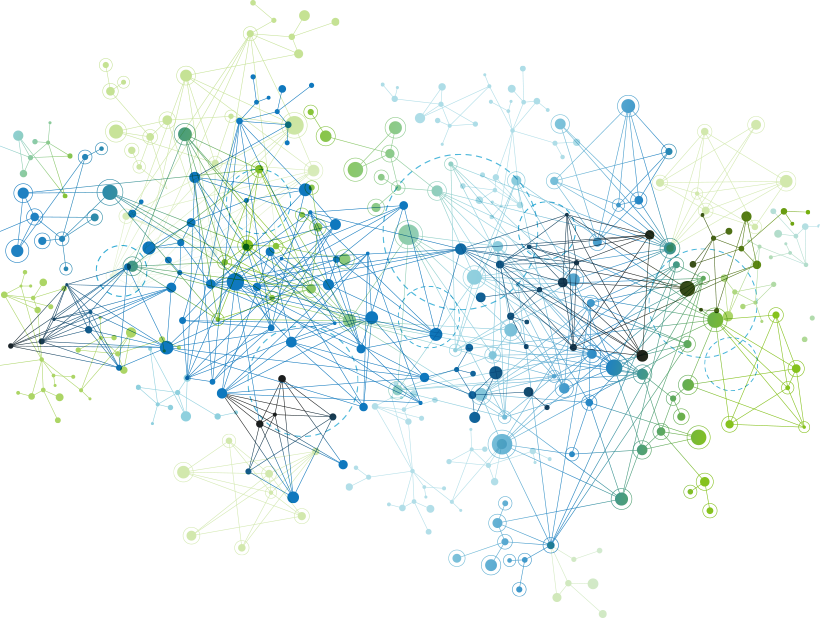


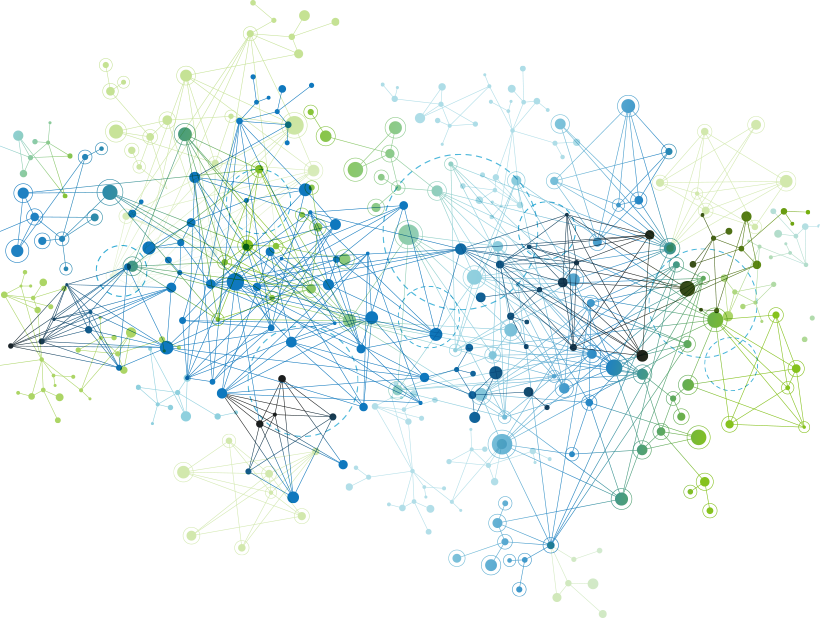
**Εργασία Εξαμήνου**

**Αρχιτεκτονική Υπολογιστών και Ενσωματωμένα Συστήματα**

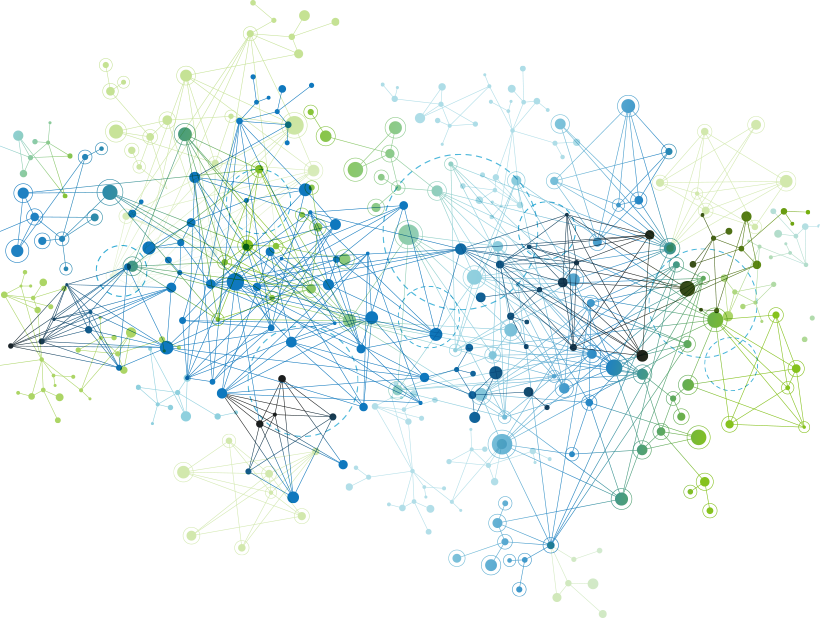
**Κωνσταντίνος Σακκάς   
ΑΜ: 136**

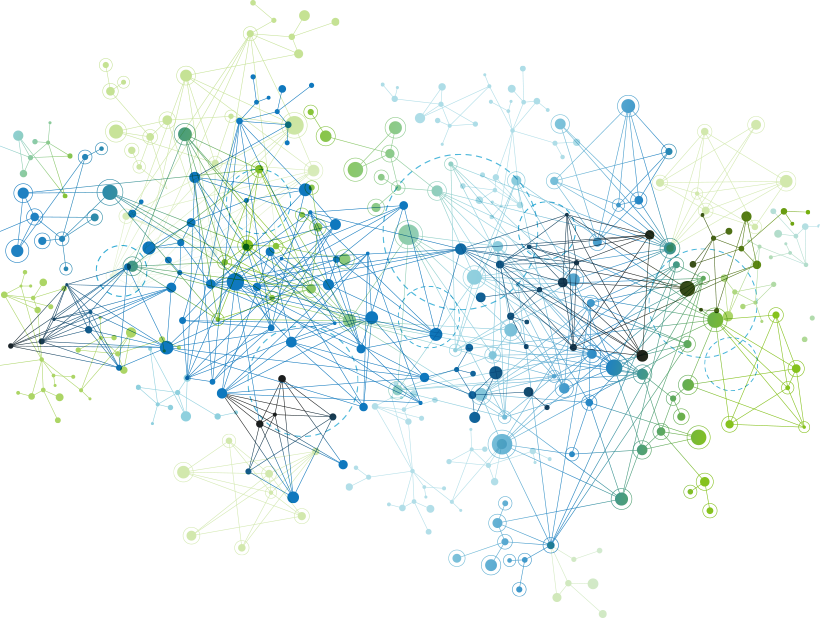
**Άρτα, 2022**

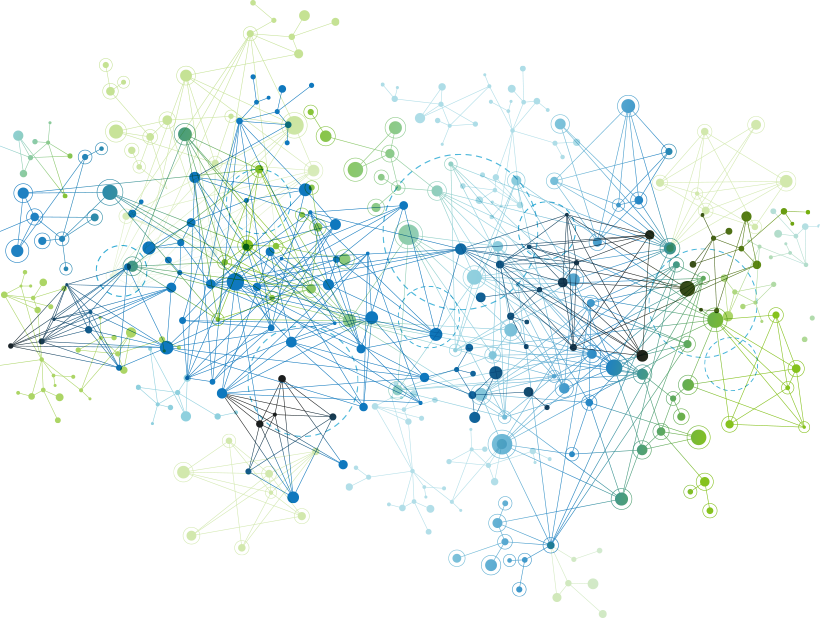












**Περιεχόμενα**

[**Θέμα 1ο** 3](#_Toc123409129)

[**Θέμα 2ο** 5](#_Toc123409130)

[**Θέμα 3ο** 8](#_Toc123409131)

[**Θέμα 4ο** 10](#_Toc123409132)

[**Θέμα 5ο – Bonus** 14](#_Toc123409133)

# **Θέμα 1ο**

Αφού εκτελέσετε επιτυχώς το παρακάτω πρόγραμμα:

# Variables:

# $s1 - iterations counter

# $s4 - accumulator for sum

# $a0 - number of system service

.data

message: .asciiz "Result = "

.text

.globl main

main:

la $4, message # $a0 <- start of welcome message

li $v0, 4 # $v0 <- service #4

syscall # call to system service

nop # not operation

move $20, $zero # $s4 <- 0, initialize accumulator

move $17, $0 # $s1 <- 0, initialize iterations counter

# Next two instructions mean

#"While $s1 < 100 Do"

loop: slti $18, $17, 101 # $s1 < 100 => $s2 <- 1

beq $18, $0, end\_lop # $s2 = 0 => go to end\_loop

add $20, $20, $17 # $s4 <- $s4 + $s1, add number

add $17, $17, 1 # $s1 <- $s1 + 1, update counter of iterations

j loop # go to loop

nop

end\_lop: move $4, $20 # $a0 <- $s4, load result of sum

li $v0,1 # $v0 <- service #1 (data is already in $a0)

syscall # call to system service

\_exit: # main program exit

li $v0,10 # $v0 <- service #10

syscall # call to system service

nop

το οποίο έχει ως αποτέλεσμα τον αριθμό 5050, να απαντήσετε στα παρακάτω θέματα:

1. **Ποιος είναι ο σκοπός του προγράμματος;**

Σκοπός του προγράμματος είναι να αθροίσουμε όλες τις τιμές από 0 μέχρι 100 και η εμφάνιση του αθροίσματος.

Αναλυτικότερα, στο πρόγραμμα μας αρχικά δηλώνουμε τα δεδομένα μας, παρατηρούμε μια ετικέτα message η οποία περιέχει μέσω asciiz ένα κείμενο το οποίο στην συνέχεια εμφανίζεται στην έξοδο του προγράμματος.

Στο κύριο πρόγραμμα έχουμε αρχικά την φόρτωση της διεύθυνσης των δεδομένων της ετικέτας message (δηλαδή το κείμενο: Result =) και την αποθήκευση αυτού στο καταχωρητή $4 ή αλλιώς στον καταχωρητή $a0.

Στην επόμενη εντολή παρατηρούμε άμεση φόρτωση της τιμής 4 στο καταχωρητή $v0.

Με την εντολή syscall λέμε στο πρόγραμμα ότι πρέπει να ετοιμαστεί για να εξάγει δεδομένα μέσω της κονσόλας.

Με την εντολή NOP ορίζουμε μια μηδενική λειτουργία.

Με την εντολή move $20, $zero, αρχικοποιούμε των συσσωρευτή με 0 όπου θα αποθηκεύσουμε στην συνέχεια το άθροισμα, αποθηκεύοντας το στο καταχωρητή $s4 ($20).

Με την εντολή move $17, $0, αρχικοποιούμε των μετρητή με 0 όπου θα μετρά τις επαναλήψεις, αποθηκεύοντας το στο καταχωρητή $s1 ($17).

Στο κομμάτι με την επανάληψη (loop) παρατηρούμε χρήση της εντολής slti (Set on less than immediate) η οποία ελέγχει αν το περιεχόμενο του καταχωρητή $s1 ($17) είναι μικρότερο από το 101 τότε μας επιστρέφει λογικό 1 το οποίο εκχωρείτε στο καταχωρητή $s2 (18)

Στην συνέχεια παρατηρούμε την εντολή beq (Branches if equal) μέσω της οποίας ελέγχουμε αν ο καταχωρητής $s2 είναι ίσος με την τιμή 0, αν είναι τότε πηγαίνουμε στην ετικέτα end\_loop, αλλιώς συνεχίζουμε με την εντολή:

add $20, $20, $17 μέσω της οποίας προσθέτουμε στο καταχωρητή $s4 το $s4 + $s1.

add $17, $17, 1, προσθέτουμε στο καταχωρητή $s1 το $s1 + 1, στην ουσία δηλαδή αυξάνουμε τον μετρητή κατά ένα.

Στην συνέχεια ξεκινάμε από την αρχή του loop ελέγχοντας με τα νέα δεδομένα που έχουμε τώρα.

**Ετικέτα ή αλλιώς ένα είδος ρουτίνας – συνάρτησης end\_lop:**

Στην συγκεκριμένη ετικέτα έχουμε πρόσβαση μέσω της εντολής ελέγχου beq του κυρίου προγράμματος. Στην end\_lop παρατηρούμε ότι:

Αποθηκεύουμε τα δεδομένα του καταχωρητή $s4 στο καταχωρητή $a0

Στην συνέχεια έχουμε απευθείας φόρτωση στο καταχωρητή $v0 την τιμή 1

Ενώ τέλος έχουμε κλήση της syscall για έξοδο των δεδομένων

**\_exit: (έξοδος προγράμματος)**

Η τελευταία εντολή πριν το τέλος του προγράμματος είναι:

li $v0,10, δηλαδή η άμεση φόρτωση στο καταχωρητή $v0 την τιμή 10.

Μετά ακολουθεί η syscall

Και η nop.

1. **Ποια είναι η έννοια της οδηγίας του SPIM “.asciiz”;**

Μέσω του .asciiz αποθηκεύουμε σε μορφή ascii μια συμβολοσειρά στην μνήμη επίσης σε αντίθεση με το .ascii το .asciiz τερματίζει μια συμβολοσειρά με το χαρακτήρα \0.

1. **Ποιος είναι ο σκοπός των οδηγιών του SPIM “.data” και “.text”;**

Το .data το χρησιμοποιούμε για να ορίσουμε τα δεδομένα του προγράμματος, καθώς και για να αρχικοποιήσουμε το τμήμα αυτό ως περιοχή δεδομένων.

Το .text χρησιμοποιείται συνήθως για το κύριο πρόγραμμα και δηλώνει ότι αυτά που ακολουθούν είναι στοιχεία κειμένου ενώ παράλληλα ορίζουμε την περιοχή ως κειμένου, στην πλειονότητα στο κομμάτι αυτό τοποθετούνται οι εντολές του προγράμματος.

1. **Τι δηλώνει μια ετικέτα;**

Μια ετικέτα δηλώνουμε σε ένα σημείο στο κώδικα το οποίο συνήθως εκτελείτε μετά από μια συνθήκη ή βρόγχο. Ο τρόπος δήλωσης της γίνεται με όνομα\_ετικέτας: .

**Παράδειγμα:**

labelpms: .asciiz "Hiii "

# **Θέμα 2ο**

1. **Να αναφέρετε και να εξηγήσετε το “προγραμματιστικό μπλοκ (σύνολο εντολών)” που απαιτείται, για να στείλετε μηνύματα στην κονσόλα.**

Για να εμφανίσουμε κάτι μέσω της κονσόλας θα πρέπει να χρησιμοποιήσουμε την εντολή syscall (System Calls), συγκεκριμένα για τα system calls ακολουθούμε μια μεθοδολογία:

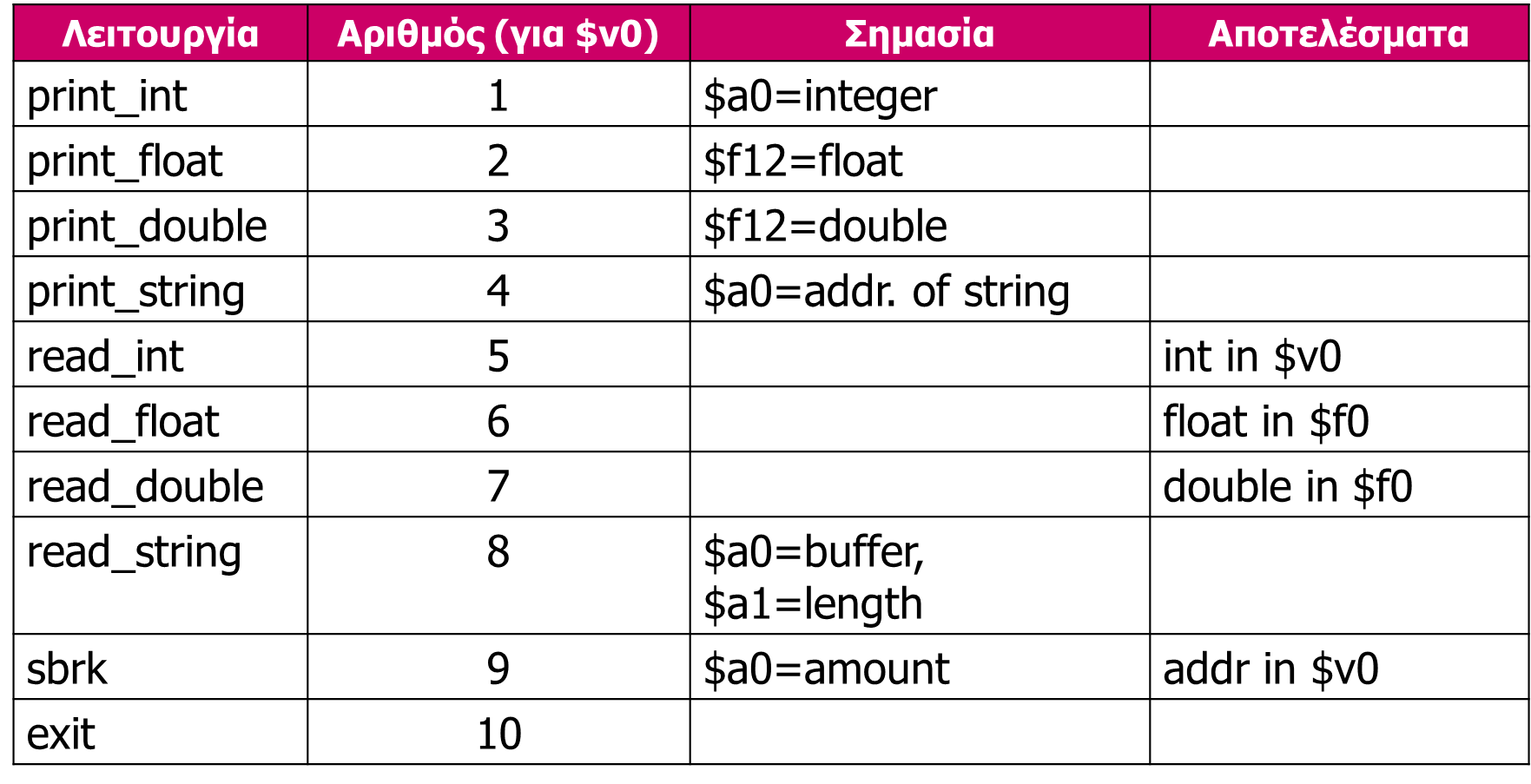
Φορτώνουμε το system call κώδικα (σύμφωνα με τον ακόλουθο κώδικα) στο καταχωρητή $v0.

Φορτώνουμε τα δεδομένα – ορίσματα μας στους καταχωρητές $a0…$a3

Καλούμε την εντολή syscall

Επιστρέφουμε τυχόν δεδομένα

**Πίνακας αντιστοιχίας λειτουργιών με αριθμούς:**



**Παράδειγμα:**

.data

txt: .asciiz "dit"

.text

.globl main

main:

li $v0, 4

la $a0, txt

syscall

1. **Να αναφέρετε το “προγραμματιστικό μπλοκ” που απαιτείται, για να εισάγετε δεδομένα από το πληκτρολόγιο.**

Για να εισάγουμε δεδομένα σύμφωνα με το πίνακα χρειαζόμαστε ένα service read ανάλογα το τύπο του περιεχομένου που θα διαβάσουμε, ας δούμε ένα απλό παράδειγμα όπου διαβάζουμε ένα αριθμό και στην συνέχεια τον εμφανίζουμε.

**Παράδειγμα:**

.data

value: .word 0

.text

.globl main

main:

la $t0, value

li $v0, 5

syscall

sw $v0, 0($t0)

lw $t1, 0($t0)

li $v0, 1

move $a0, $t1

syscall

1. **Μόλις ανακτηθούν τα δεδομένα από το πληκτρολόγιο, που τοποθετούνται;**

Σε καταχωρητές τύπου $v0 ή $f0 και συνήθως τους αποθηκεύουμε στην μνήμη με την εντολή sw ή μεταφέρουμε την τιμή σε κάποιον καταχωρητή μόνιμης αποθήκευσης.

1. **Δημιουργήστε ένα πρόγραμμα για να προσθέσετε δύο ακέραιους αριθμούς που έχουν εισαχθεί από το πληκτρολόγιο και να εμφανίζει το αποτέλεσμα ως εξής: “Sum= αποτέλσμα”.**

.data

        value: .word 0, 0, 0

        msg1:  .asciiz "Sum = "

    .text

    .globl main

main:

    la $t0, value

    li $v0, 5

    syscall

    sw $v0, 0($t0)

    li $v0, 5

    syscall

    sw $v0, 4($t0)

    lw $t1, 0($t0)

    lw $t2, 4($t0)

    add $t3, $t1, $t2

    sw $t3, 8($t0)

   li $v0, 4

   la $a0, msg1

   syscall

   li $v0, 1

   move $a0, $t3

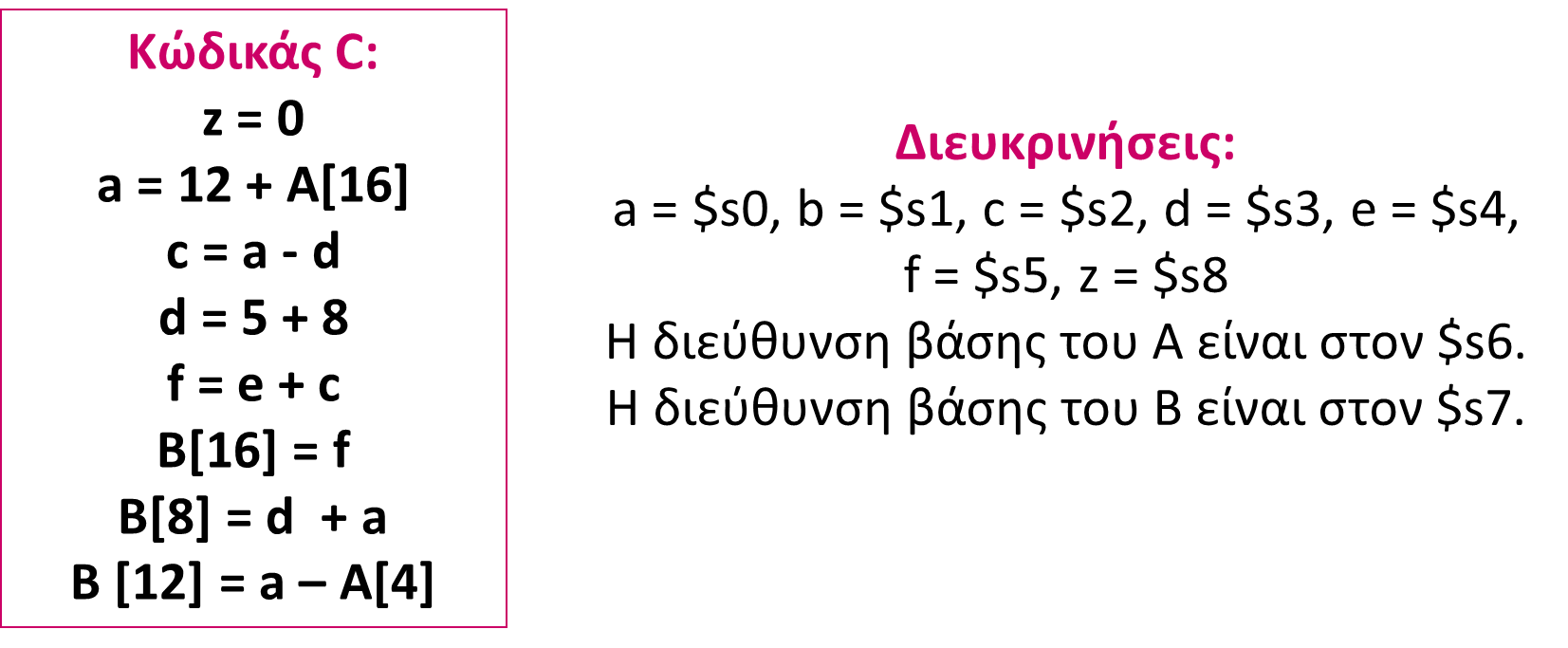
   syscall

   li  $v0, 10

   syscall

# **Θέμα 3ο**

Να μετατρέψετε τον ακόλουθο κώδικα από C σε Assembly.



li $s8, 0

lw $t0, 64($s6)

addi $s0, $t0, 12

sub $s2, $s0, $s3

addi $s3, $s3, 5

addi $s3, $s3, 8

add $s5, $s4, $s2

sw $s5, 64($s7)

add $t1, $s3, $s0

sw $t1, 32($s7)

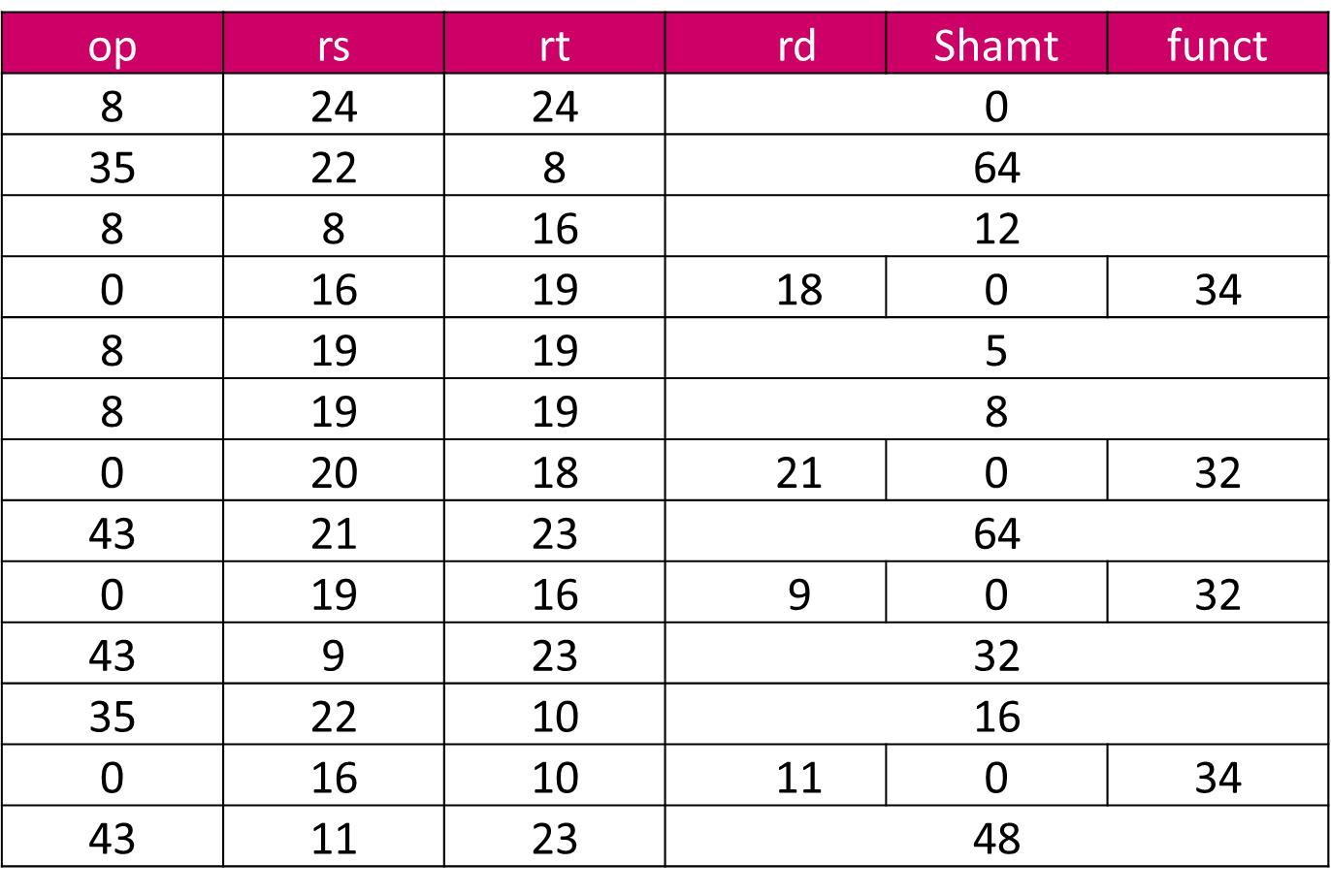
lw $t2, 16($s6)

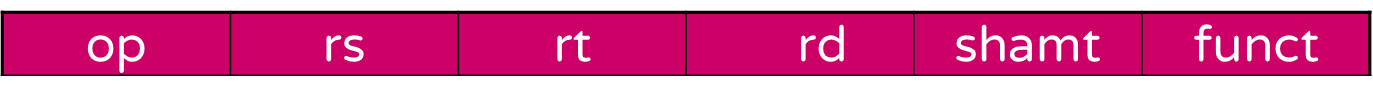
sub $t3, $s0, $t2

sw $t3, 48($s7)

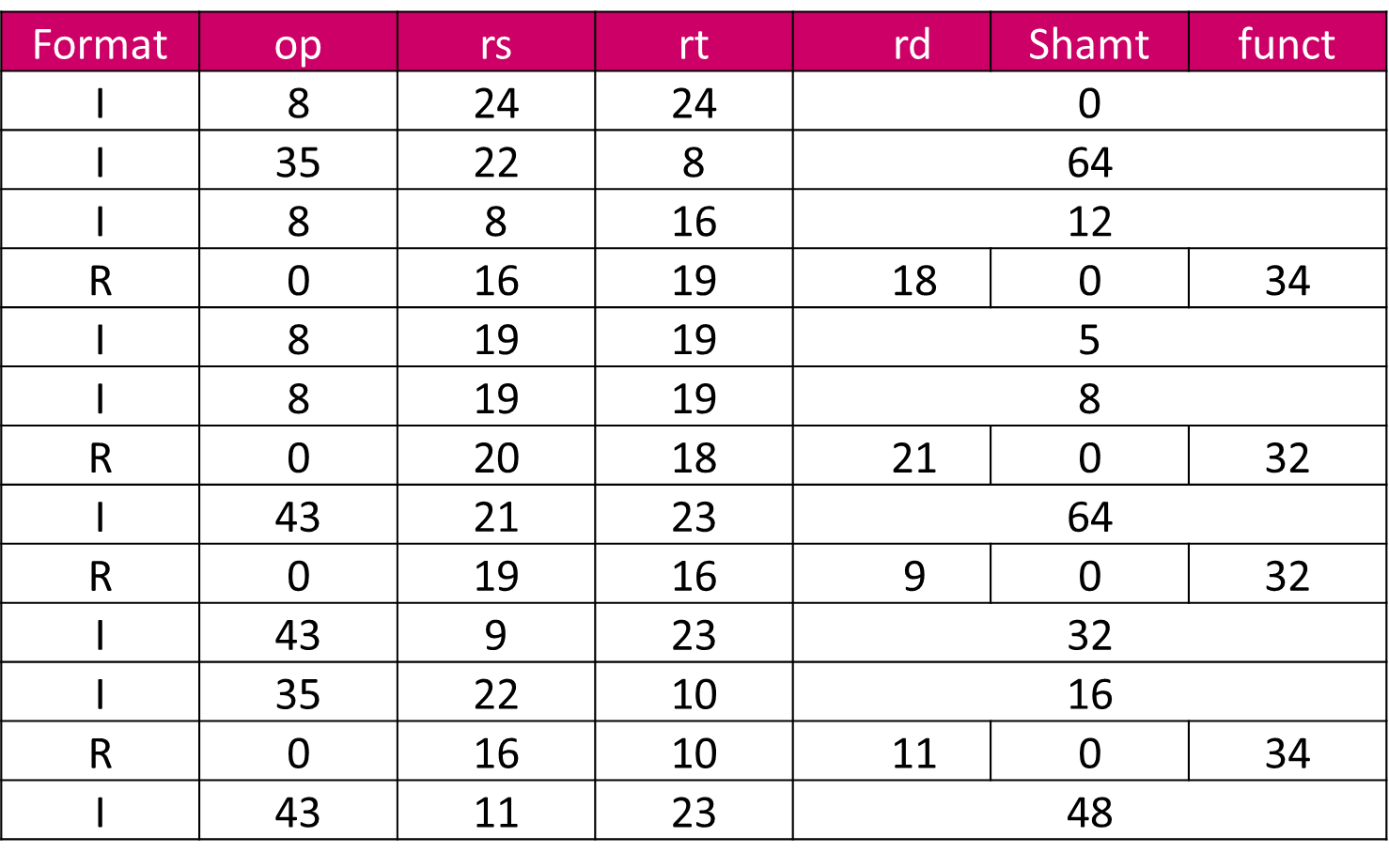
1. Μετατρέψτε τον προηγούμενο κώδικα Assembly σε πίνακα κώδικα μηχανής στο δεκαδικό σύστημα.

Το li είναι μια ψευδο-εντολή που υποστηρίζεται από assemblers MIPS-32, όχι μια "πραγματική" εντολή γιαυτό και δεν έχει αντίστοιχο αριθμό στο πίνακα forma, άρα θα το τοποθετίσουμε με μια αντίστοιχη εντολή όπως: addi $s8, $s8, 0.



1. Για κάθε γραμμή κώδικα θα δηλώνεται το format στο οποίο ανήκει, ο πίνακας θα είναι της μορφής (ανάλογα το format):  
   

Το li είναι μια ψευδο-εντολή που υποστηρίζεται από assemblers MIPS-32, όχι μια "πραγματική" εντολή γιαυτό και δεν έχει αντίστοιχο αριθμό στο πίνακα forma, άρα θα το τοποθετίσουμε με μια αντίστοιχη εντολή όπως: addi $s8, $s8, 0.



# **Θέμα 4ο**

1. **Να υλοποιήσετε ένα μικρό διάλογο, δική σας επιλογής, με το χρήστη.**

.data

value: .word 0, 0, 0

msg1: .asciiz "Γειαα, ποιο είναι το όνομα σου; \n"

msg2: .asciiz "Σε ποιο τμήμα σπουδάζεις; \n"

msg3: .asciiz "Τι εξάμηνο είσαι; \n"

msg4: .asciiz "Ευχαριστούμε για την επικοινωνία \n"

.text # Μέρος text προγράμματος

.globl main # Κύριο πρόγραμμα

main:

la $t0, value # Φόρτωση διεύθυνσης του value για αρχικοποίηση των καταχωρητών

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, msg1 # Φόρτωση της διέυθυνσης του msg1 στο καταχωρητή $a0 (argument to print\_string call)

syscall # Κλήση syscall

li $v0, 8 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 8 (syscall για read\_string)

syscall # Κλήση syscall

sw $v0, 0($t0) # Αποθήκευση του περιεχομένου του καταχωρητή $v0 στην θέση 4 του καταχωρητή $t0

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, msg2 # Φόρτωση της διέυθυνσης του msg2 στο καταχωρητή $a0 (argument to print\_string call)

syscall # Κλήση syscall

li $v0, 8 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 8 (syscall για read\_string)

syscall # Κλήση syscall

sw $v0, 8($t0) # Αποθήκευση του περιεχομένου του καταχωρητή $v0 στην θέση 4 του καταχωρητή $t0

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, msg3 # Φόρτωση της διέυθυνσης του msg2 στο καταχωρητή $a0 (argument to print\_string call)

syscall # Κλήση syscall

li $v0, 8 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 8 (syscall για read\_string)

syscall # Κλήση syscall

sw $v0, 12($t0) # Αποθήκευση του περιεχομένου του καταχωρητή $v0 στην θέση 4 του καταχωρητή $t0

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, msg3 # Φόρτωση της διέυθυνσης του msg2 στο καταχωρητή $a0 (argument to print\_string call)

syscall # Κλήση syscall

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, msg4 # Φόρτωση της διέυθυνσης του msg2 στο καταχωρητή $a0 (argument to print\_string call)

syscall # Κλήση syscall

li $v0, 10 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 10 (syscall για exit)

syscall # Κλήση syscall

1. Τροποποιήστε το πρόγραμμα που δίνεται στην περιγραφή, ώστε να εκτελεί επαναληπτικά την αρχική διαδικασία, αλλά υψώνοντας αυτή τη φορά στο τετράγωνο όλους τους περιττούς αριθμούς.

.data

message: .asciiz "Result for add = "

message2: .asciiz "\nResult for ^2 = "

.text

.globl main

main:

la $4, message # $a0 <- start of welcome message

li $v0, 4 # $v0 <- service #4

syscall # call to system service

nop # not operation

move $20, $zero # $s4 <- 0, initialize accumulator

move $21, $zero # $s5 <- 0, initialize accumulator

move $17, $0 # $s1 <- 0, initialize iterations counter

move $22, $0 # $s6 <- 0, initialize iterations counter

# Next two instructions mean

#"While $s1 < 100 Do"

addi $s6,2

loop: slti $18, $17, 101 # $s1 < 101 => $s2 <- 1

beq $18, $0, end\_lop # $s2 = 0 => go to end\_loop

div $17, $s6

mfhi $t0

beq $t0, $0, if\_else\_label

mul $23, $17, $17 # $s6 <- $s1 \* $s1, ^2 number

add $21, $21, $23 # $s5 <- $s5 + $s6, add number

add $17, $17, 1 # $s1 <- $s1 + 1, update counter of iterations

j if\_end\_label

if\_else\_label:

add $20, $20, $17 # $s4 <- $s4 + $s1, add number

add $17, $17, 1 # $s1 <- $s1 + 1, update counter of iterations

if\_end\_label:

j loop # go to loop

nop

end\_lop:

move $4, $20 # $a0 <- $s4, load result of sum

li $v0,1 # $v0 <- service #1 (data is already in $a0)

syscall # call to system service

la $4, message2 # $a1 <- message2

li $v0, 4 # $v1 <- service #4

syscall # call to system service

move $4, $21 # $a0 <- $s5, load result of ^2

li $v0,1 # $v0 <- service #1 (data is already in $a0)

syscall # call to system service

\_exit: # main program exit

li $v0,10 # $v0 <- service #10

syscall # call to system service

nop

# **Θέμα 5ο – Bonus**

Να γράψετε και να εκτελέσετε ένα πρόγραμμα σε κώδικα assembly MIPS, το οποίο να προσδιορίζει το πλήθος των πρώτων αριθμών, που είναι μικρότεροι από κάποιον θετικό ακέραιο N. Ο αριθμός αυτός συνήθως αναφέρεται ως π(N). Π.χ., π(1) = 0, π(2) = 1, π(3) = 2, π(4) = 2, π(5) = 3, ..., π(10) = 4 κτλ. Θυμηθείτε ότι το ένα (1) δεν είναι πρώτος αριθμός.

Ακολουθήστε τα παρακάτω βήματα:

1. Το πρόγραμμα δέχεται ως είσοδο ένα θετικό ακέραιο αριθμό,
2. Ορίζεται ένας πίνακας ακεραίων από το 2 έως το Ν,
3. Αναγνωρίζεται ο μικρότερος ακέραιος *i* στον πίνακα, ο οποίος δεν έχει επισημανθεί ως κάποιο πολλαπλάσιο των ακεραίων που έχουν ήδη ελεγχθεί (φυσικά το 2 στην αρχή), προσμετράται ως πρώτο αριθμός, και στη συνέχεια επισημαίνονται τα πολλαπλάσια του, δηλαδή, *i, 2i, 3i, κοκ*.
4. Το βήμα 3 επαναλαμβάνεται.
5. Όταν , το πρόγραμμα τερματίζει και τυπώνει το πλήθος των πρώτων αριθμών.

Π.χ., για N = 20, ο πίνακας είναι:

2, 3, 4, 5, 6, 7,8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20

Τίποτα δεν έχει επισημανθεί ακόμη, και ο μικρότερος ακέραιος είναι το 2. Συνεπώς, το 2 υπολογίζεται ως πρώτος αριθμός (εμφανίζεται υπογραμμισμένο), ενώ τα πολλαπλάσιά του επισημαίνονται (εμφανίζονται έντονα):

**2**, 3, **4**, 5, **6**, 7, **8**, 9, **10**, 11, **12**, 13, **14**, 15, **16**, 17, **18**, 19, **20**

Ο μικρότερος ακέραιος αριθμός χωρίς σήμανση είναι τώρα το *i = 3*. To 3 προσμετράται ως πρώτος αριθμός και στη συνέχεια τα πολλαπλάσιά του επισημαίνονται:

**2**, **3**, **4**, 5, **6**, 7, **8**, **9**, **10**, 11, **12**, 13, **14**, **15**, **16**, 17, **18**, 19, **20**

Ένας αριθμός μπορεί να επισημανθεί παραπάνω από 1 φορά, όπως το 6, ωστόσο δεν σημαίνει κάτι διαφορετικό, απλά και πάλι δεν θεωρείται πρώτος αριθμός.

Ο επόμενος μικρότερος ακέραιος αριθμός που δεν έχει σημανθεί είναι το 5. Ωστόσο είναι μεγαλύτερος από την τετραγωνική ρίζα του 20, και το πρόγραμμα τερματίζει αφού συνυπολογιστεί στο πλήθος των πρώτων αριθμών (2 έως τώρα), το πλήθος των ακεραίων του πίνακα που δεν έχουν επισημανθεί, δηλ. οι 6 αριθμοί 5, 7, 11, 13, 17, 19. Έτσι, συνολικά, π(20) = 8.

Συμβουλή: Υλοποιήστε τον πίνακα των ακεραίων *2, .., Ν* ως *bit vector* στην περιοχή δεδομένων του MIPS. Με άλλα λόγια, να ορίσετε ένα διάνυσμα μήκους *N-1,* όπου το 1ο bit αντιστοιχεί στο 2, το 2ο bit στο 3, κοκ. Αρχικά, οι τιμές του διανύσματος θα είναι όλες 1. Συνεπώς, όταν κάποια θέση πρέπει να επισημανθεί, η τιμή της θα αλλάζει από 1 σε 0. Στο τέλος της διαδικασίας, όλα τα bit που έχουν παραμείνει 1, θα αντιστοιχούν σε πρώτους αριθμούς.

Στην εκκίνηση του προγράμματος, η έξοδος της κονσόλας πρέπει να είναι “Enter Integer:” Ένας θετικός ακέραιος αριθμός N θα πρέπει να μπορεί να εισαχθεί ως δεκαδικός αριθμός. Η επόμενη έξοδος (μετά από enter) θα πρέπει να είναι “pi(N) =” ακολουθούμενη από τη σωστή τιμή του π(N), δοσμένη σαν δεκαδικός αριθμός.

Παράδειγμα εκτέλεσης:

1. Enter Integer:
2. 5000
3. pi(5000) = 669

Η εργασία θα υλοποιηθεί από ομάδες των 3 ατόμων. Ο σχηματισμός των ομάδων είναι δική σας ευθύνη.

**Κώδικας:**

.data

lastm: .asciiz "\n Τελική Εργασία Εξαμήνου \n"

firstms: .asciiz "\nΕισάγετε έναν ακέραιο αριθμό:\n"

primes1: .asciiz "\nΟι πρώτοι αριθμοί βρίσκονται ανάμεσα απο το 2 και το "

primes2: .asciiz " και είναι: "

nl: .asciiz "\n"

comma: .asciiz ", "

ms1: .asciiz "p("

ms2: .asciiz ") = "

memory: .word 0

.text

main:

la $t0, memory # Φόρτωση της διευθύνσης του memory στο καταχωρητή

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, lastm # Φόρτωση της διέυθυνσης του lastm στο καταχωρητή $a0 (argument to print\_string call)

syscall

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, firstms # Φόρτωση της διέυθυνσης του firstms στο καταχωρητή $a0 (argument to print\_string call)

syscall

li $v0, 5 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 5 (syscall για read\_int)

syscall

move $s0, $v0 # Μεταφορά της τιμής του $v0 στο $s0

sw $s0, 0($t0) # Αποθήκευση του περιεχομένου του καταχωρητή $s0 στην θέση 0 του καταχωρητή $t0

mul $a0, $s0, 4 # Πολλαπλασιασμός του $s0 με το 4 για να συλλέξουμε τα byte μνήμης

li $v0, 9 # Δυναμική δεσμευση μνήμης για αποθήκευση αποτελέσματων

syscall

move $s1, $v0 # Μετακίνηση στην δυναμική μνήμη

# Αρχικοποίηση δυναμικού πίνακα με 0

li $s2, 0 # Αρχικοποίηση μετρητή με 0

bl\_loop:

sb $zero, 0($s1) # Αποθήκευση με 0 των θέσεων του πίνακα $s1

beq $s0, $s2, endbl # Αν το $s0 είναι ισο του $s2 τοτε πήγαινε στην ετικέτα endbl

addi $s1, $s1, 1 # Αύξηση του μετρητή κατά ένα

addi $s2, $s2, 1 # Αύξηση του μετρητή κατά ένα

j bl\_loop # Μεταπήδηση στην ετικέτα bl\_loop

endbl:

li $s2, 1 # Αρχικοποίηση του $s2 με 1

# Εύρεση πρώτων αριθμών

ot\_loop:

addi $s2, $s2, 1 # Αύξηση του μετρητή κατά ένα

mult $s2, $s2 # Πολλαπλασιασμός $s2 με το $s2

mflo $s3 # Αποθήκευση αποτελέσματος πολλαπλασιασμού

bgt $s3, $s0, exit # Εάν ο μετρητής είναι μεγαλύτερος από την είσοδο του χρήστη

#if prime[counter] == 0

lb $s4, 0($s1) # Φόρτωση του πρώτου byte του $s1

bnez $s4, ot\_loop # Εάν η τιμή δεν είναι 0 -> ot\_loop

mul $s5, $s2, $s2 # Πολλαπλασιασμός μετρητή $s2 με $s2

in\_loop:

# Εάν ο εσωτερικός μετρητής περάσει την είσοδο του χρήστη, πραγματοποιήθηκε βρόχος

bgt $s5, $s0, ot\_loop # Αν το $s5 είναι μεγαλύτερο του $s0 --> ot\_loop

add $s6, $s5, $s1 # Πρόσθεση $s5 + $s1

sb $s2, 0($s6) # Αποθήκευση με $s2 στο 0($s6)

add $s5, $s5, $s2 # Πρόσθεση $s5 + $s2

j in\_loop # Μεταπήδηση στην ετικέτα in\_loop

j ot\_loop # Μεταπήδηση στην ετικέτα ot\_loop

exit:

# Είσοδος χρήστη

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, primes1 # Φόρτωση της διέυθυνσης του primes1 στο καταχωρητή $a0 (argument to print\_string call)

syscall

li $v0, 1 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 1 (syscall για print\_int)

move $a0, $s0 # Φόρτωση στο καταχωρητή $a0 το περιεχόμενο του $s0

syscall

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, primes2 # Φόρτωση της διέυθυνσης του primes2 στο καταχωρητή $a0 (argument to print\_string call)

syscall

# Εκτύπωση όλων των πρώτων αριθμών

li $s2, 1 # Αρχικοποίηση μετρητή

li $s5, 0 # Αρχικοποίηση μετρητή

print:

addi $s2, $s2, 1 # Αύξηση μετρητή κατά 1

bgt $s2, $s0, done # Εάν ο μετρητής είναι μεγαλύτερος από την είσοδο του χρήστη --> exit

add $s3, $s1, $s2 # Πρόσθεση $s1 + $s2

lb $s4, 0($s3) # Φόρτωση του πρώτου byte του $s3

bnez $s4, print # Εάν η τιμή δεν είναι 0 -> ot\_loop

# Αλλιώς εμφάνισε τα δεδομένα - πρώτοι αριθμοί

li $v0, 1 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 1 (syscall για print\_int)

move $a0, $s2 # Φόρτωση στο καταχωρητή $a0 το περιεχόμενο του $s0

syscall

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, comma # Φόρτωση της διέυθυνσης του comma στο καταχωρητή $a0 (argument to print\_string call)

addi $s5, $s5, 1 # Αύξηση μετρητή

syscall

j print # Μεταπήδηση στην ετικέτα print

done:

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, nl # Φόρτωση της διέυθυνσης του nl στο καταχωρητή $a0 (argument to print\_string call)

syscall

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, nl # Φόρτωση της διέυθυνσης του nl στο καταχωρητή $a0 (argument to print\_string call)

syscall

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, ms1 # Φόρτωση της διέυθυνσης του ms1 στο καταχωρητή $a0 (argument to print\_string call)

syscall

lw $t2, 0($t0) # Φόρτωση του 0($t0) στο $t2

li $v0, 1 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 1 (syscall για print\_int)

move $a0, $t2 # Φόρτωση στο καταχωρητή $a0 το περιεχόμενο του $t2

syscall

li $v0, 4 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για print\_string)

la $a0, ms2 # Φόρτωση της διέυθυνσης του ms2 στο καταχωρητή $a0 (argument to print\_string call)

syscall

li $v0, 1 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 1 (syscall για print\_int)

move $a0, $s5 # Φόρτωση στο καταχωρητή $a0 το περιεχόμενο του $t2

syscall

li $v0, 10 # Load exit opcode

syscall # Exit program # Κλήση syscall