Οργάνωση και Σχεδίαση Υπολογιστών (HY232)
Χειμερινό Εξάμηνο 2019-2020

Εργαστήριο 1

Στόχοι του εργαστηρίου

- Εντολές εισόδου-εξόδου
- Χρήση συνθηκών σε δομές επιλογής
- Χρήση συνθηκών σε δομές επανάληψης
- Εντολές προσπέλασης της μνήμης
- Χρήση πινάκων

Εντολές εισόδου-εξόδου

Εντολές για είσοδο ακεραίων αριθμών από την κονσόλα:

```
li $v0, 5              # κωδικός 5 από τα syscall στον $v0
syscall
move $t0, $v0          # καταχώριση της τιμής που εισάγαμε, στον
                       # register $t0
```

Εντολές για εμφάνιση ακεραίου στην κονσόλα:

```
li $v0, 1              # pseudo-instruction για addi $v0, $0, 1
                       # κωδικός 1 από τα syscall στον $v0
move $a0, $t0          # ο $t0 έχει την τιμή που θέλουμε να     # εμφανίσουμε και
                       το φορτώνουμε στον $a0
                       # σαν παράμετρο
syscall
```

Εντολές για εμφάνιση μηνυμάτων στην κονσόλα:

Για να τυπώσετε ένα string στην οθόνη θα πρέπει να το δηλώσετε πρώτα στον τμήμα .data του προγράμματός σας ως εξής:

Αρχικά πρέπει να ορίσετε το string στην περιοχή της μνήμης με ένα label (στην περίπτωση του παραδείγματος String_name1) για να μπορείτε να έχετε πρόσβαση με χρήση της διεύθυνσης του.

```
.data
String_name1:   .asciiz    "msg"
```

Σε περίπτωση που θέλετε να αλλάξετε γραμμή (για να είναι πιο ευανάγνωστη η εκτέλεση) πρέπει να ορίσετε ένα string αλλαγής γραμμής.

```
String_name2:   .asciiz   "\n"
```

Για να τυπώσετε το string στην κονσόλα πρέπει να γράψετε τις εξής εντολές.

```
li  $v0, 4                      # κωδικός 4 από τα syscall στον $v0
la  $a0, String_name1    # φορτώνουμε στον $a0 την διεύθυνση
                                # του String που θα εκτυπώσουμε
syscall
```

Οι εντολές syscall είναι κλήσεις λειτουργιών του συστήματος. Με αυτές μπορείτε εκτός από I/O λειτουργίες να τερματίσετε ένα πρόγραμμα (syscall με παράμετρο $v0=10) ή να ζητήσετε δυναμικά μνήμη (αντίστοιχη εντολή malloc της C). Μπορείτε να βρείτε όλες τις λειτουργίες στο Help menu του MARS.

Υλοποίηση δομών ελέγχου ροής προγράμματος

Εντολές ελέγχου ροής είναι το σύνολο των εντολών το οποίο αλλάζουν την ροή εκτέλεσης προγράμματος. Κανονικά η σειρά εκτέλεσης των εντολών είναι μετά από κάθε εντολή να εκτελείται η αμέσως από κάτω. Υπάρχουν όμως εντολές οι οποίες αλλάζουν είτε στατικά (χωρίς συνθήκη) είτε δυναμικά (με συνθήκη) την ροή εκτέλεσης των εντολών.

Παράδειγμα ελέγχου ροής, μετατροπή από C σε assembly:

Παράδειγμα 1 - if

```
if( x = = 0){
        …       // Case true
}
else
{
        …       // Case false.
}
…               // Next instruction
```

Αντιστοιχίζουμε την μεταβλητή x στον καταχωρητή $t3

```
bnez $t3,else
…                       # case true
j endif

else:
…                       # case false

endif:
…                       # next instruction
```

Παράδειγμα 2 - for

```
for( i = 0 ; i < 10 ; i++ )
{
        …               // code
}
…                       // next instruction
```

Αντιστοιχίζουμε την μεταβλητή i στον καταχωρητή $t3 και αποθηκεύουμε το όριο του loop (εδώ 10) στον καταχωρητή $t4.

```
li $t3,0          # add $t3, $0, $0
li $t4, 10        # addi $t4,$0,10

for:
bge $t3,$t4,endfor

…                 # code

addi $t3,$t3,1
j for

endfor:
…                 # next instruction
```

Παράδειγμα 3 - multiple if

```
if( x > 0 && x < 10)
{
        …         // code
}
…                 // next instruction
```

Αντιστοιχίζουμε την μεταβλητή x στον καταχωρητή $t3 και αποθηκεύουμε την σταθερά με την οποία θα συγκρίνουμε (εδώ 10) στον καταχωρητή $t4.

```
li  $t4,10
blez $t3, endif
```

```
bge $t3,$t4, endif

…       # code

endif :
…       # next instruction
```

β’ τρόπος:

```
addi $t4,$0,10
bgtz $t3, cond2          # if(x > 0)
j endif
cond2:
blt $t3,$t4,is_true      # if(x < 10)
j endif
is_true:

…       #code

endif:
…       # next instruction
```

Χρήση μνήμης

Τα δεδομένα αποθηκεύονται σε ξεχωριστό τμήμα μνήμης το οποίο χρησιμοποιείται αποκλειστικά για αυτό το σκοπό. Γι’ αυτό τον λόγο η δήλωση του πίνακα, που θα χρησιμοποιήσετε, πρέπει να γίνει στο τμήμα **.data.** Ο τρόπος δήλωσης του πίνακα και η σύνταξη των εντολών load (lw) και store (sw), οι οποίες χρησιμοποιούνται για την μεταφορά δεδομένων μεταξύ μνήμη και καταχωρητή, εξηγούνται παρακάτω.

Δήλωση πίνακα

```
.data              # στο τμήμα .data πρέπει να δηλωθεί ο πίνακας

Name_array: .space N       # Name_array: δώστε ότι όνομα
                  # θέλετε στον πίνακα
                  # .space N:  δηλώνει ότι θέλουμε να
                  # κρατήσουμε χώρο ίσο με N bytes για
                  # τον πίνακα Name_array
```

Εντολή load (lw)

Η εντολή αυτή αποθηκεύει δεδομένα σε καταχωρητή, τα οποία έχει πάρει από συγκεκριμένη διεύθυνση της μνήμης.

Σύνταξη: **lw Rt, Address(Rs)**	Σημασία**: Rt=Memory[Address+Rs]**

Προσοχή: όπου Rt, Rs είναι καταχωρητές και όπου Address είναι το όνομα του πίνακα (label) που δώσατε στο τμήμα .data.
Με βάση την παραπάνω δήλωση (στο τμήμα .data) θα έπρεπε να γράψουμε:

lw Rt, Name_array(Rs)

Από την σύνταξη της εντολής αυτής γίνεται κατανοητό ότι ζητάμε τα δεδομένα της μνήμης στην διεύθυνση [Name_array+Rs] και τα οποία θα αποθηκευτούν στον καταχωρητή Rt.

Εντολή store (sw)

Η εντολή αυτή αποθηκεύει δεδομένα από έναν καταχωρητή σε συγκεκριμένη διεύθυνση στη μνήμη.

Σύνταξη**: sw Rt, Address(Rs)**	Σημασία: **Memory[Address+Rs]=Rt**

Ο τρόπος λειτουργίας είναι παρόμοιος με την εντολή **lw(load word).** Η λειτουργία που υλοποιεί αυτή η εντολή είναι: Αποθήκευσε τα περιεχόμενα του καταχωρητή Rt στην μνήμη και συγκεκριμένα στην διεύθυνση **[Address+Rs].**

Η διάφορα με την πρώτη εντολή είναι ότι η sw αποθηκεύει δεδομένα στην μνήμη ενώ η lw διαβάζει τα δεδομένα της μνήμης.

Ευθυγράμμιση

Όπως γνωρίζετε από την θεωρία ο MIPS είναι ένας 32 bit επεξεργαστής. Αυτό συνεπάγεται ότι όλοι οι καταχωρητές του έχουν μέγεθος 32 bits ή αλλιώς 4 bytes. Ορίζουμε σαν **word** τα δεδομένα μεγέθους 4 bytes τα οποία χωράνε σε έναν register.

Όταν ορίζουμε έναν πίνακα στην μνήμη με την εντολή **.space** καθορίζουμε το μέγεθος του ίσο με έναν αριθμό από **bytes** όπως παρουσιάστηκε πριν. Εφόσον όμως τα στοιχεία που αποθηκεύονται σε αυτόν θα είναι μεγέθους 4 bytes το καθένα, πρέπει να καθορίσετε το μέγεθος του πίνακα να είναι πολλαπλάσιο του 4. Έτσι το πρώτο στοιχείο του πίνακα βρίσκεται στα bytes 0 έως 3. Αν θέλετε να το κάνετε load θα πρέπει να συντάξετε την εντολή *lw Rt, Address(Rs)* με τον Rs να έχει περιεχόμενο 0. Αν τώρα θέλετε το δεύτερο στοιχείο του πίνακα αυτό βρίσκεται στα bytes 4 έως 7. Για να το κάνετε load θα πρέπει να γράψετε την ίδια εντολή αλλά ο Rs θα πρέπει να έχει περιεχόμενο τον αριθμό 4.

Σημείωση: Όταν ορίζετε έναν πίνακα πρέπει να δηλώνετε και τι μέγεθος θα έχουν τα δεδομένα που θα αποθηκεύονται σε αυτόν. Αυτό γίνεται με την εντολή **.align n** όπου **n** έναν ακέραιος που θα δώσετε εσείς. Η εντολή αυτή σημαίνει ότι τα στοιχεία του πίνακα έχουν μέγεθος $\mathbf{2^n}$. Έτσι για έναν πίνακα 10 θέσεων με λέξεις 4 bytes πρέπει να συντάξετε την εντολή:

```
.align 2          #μέγεθος στοιχείου 4 bytes
label: .space 40  #μέγεθος πίνακα 40 bytes = 10 στοιχεία
```

Ένα παράδειγμα χρήσης μνήμης

```
.data
.align 2          # λέξεις 4 bytes
vector: .space 24 # πίνακας 6 θέσεων
```

```
.text

# …


li $t1,4                    # ένας τρόπος πρόσβασης στο 2ο στοιχείο
lw $t0,vector($t1)          # του πίνακα


# …


la $t2,vector                       # ακόμα ένας τρόπος για πρόσβαση στο 2ο
lw $t3,4($t2)               # στοιχείο του πίνακα


# …


la $t2, vector                      # ακόμα ένας τρόπος για πρόσβαση στο 2ο
addi $t2, $t2, 4                    # στοιχείο του πίνακα
lw $t0,0($t2)
```

Το πλήθος των εντολών είναι σχετικά μεγάλο αλλά η λειτουργία τους είναι πολύ απλή. Συνιστάται κατά τον προγραμματισμό σε assembly να κάνετε χρήση του instruction set. Μπορείτε επίσης να χρησιμοποιήσετε και ψευδο-εντολές σαν να ήταν κανονικές εντολές (για παράδειγμα *li, la, bgez, blt* κοκ). Η αποστήθιση όλων των εντολών δεν είναι ζητούμενο σε αυτό το εργαστήριο ωστόσο πρέπει να είσθε σε θέση να μπορείτε να χρησιμοποιήσετε τις εντολές που αναγράφονται σε αυτό το αρχείο.

Για να πάρετε άριστα στην εξέταση, η λύση σας θα πρέπει να είναι σωστή αλλά και αποδοτική. Θα πρέπει να χρησιμοποιείτε τον ελάχιστο δυνατό αριθμό εντολών στην λύση σας, αλλά και η λύση σας να είναι ευανάγνωστη και καλά σχολιασμένη.

Για το επόμενο εργαστήριο έχετε να υλοποιήσετε τις παρακάτω εργαστηριακές ασκήσεις. Την ώρα του εργαστηρίου θα εξετασθείτε προφορικά πάνω στους κώδικες που θα παραδώσετε.

Άσκηση 1 – Byte manipulation (5 μονάδες)

Να γραφεί κώδικας σε MIPS assembly ο οποίος να επιστρέφει στον καταχωρητή $s0 έναν συνδυασμό των περιεχομένων του καταχωρητή $t0 ως εξής:

- Το λιγότερο σημαντικό *byte* του $t0 να γράφεται στο λιγότερο σημαντικό *byte* του $s0.
- Το δεύτερο λιγότερο σημαντικό *byte* του $t0 να γράφεται στο δεύτερο λιγότερο σημαντικό *byte* του $s0, *με τα bits του ανεστραμμένα*.
- Τα δύο πιο σημαντικά bytes του $s0 κρατούν την αρχική τους τιμή.

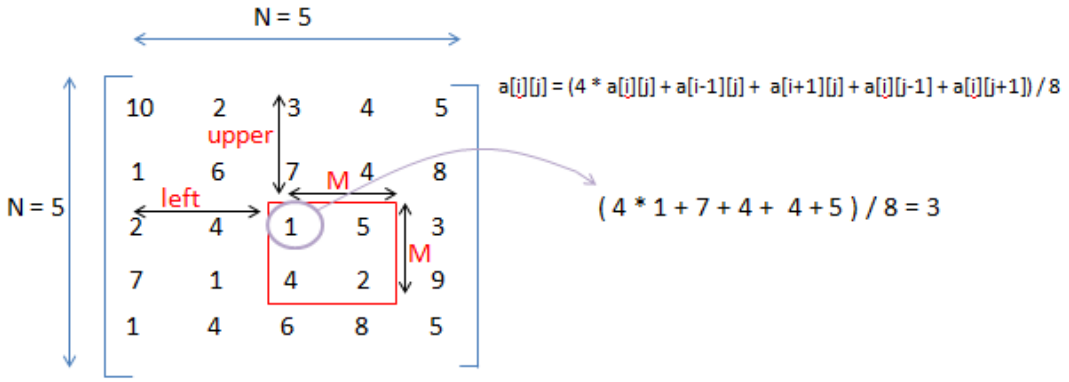
Για παράδειγμα, εάν $t0 = 0x1023B818, και $s0 = 0x00112233, τότε η τελική τιμή του $s0 είναι:

$s0 = 0x00111D18
Ο κώδικας δεν θα πρέπει να χρησιμοποιεί την μνήμη για αποθήκευση δεδομένων, δηλ. δεν θα πρέπει να χρησιμοποιήσετε καθόλου εντολές load και store. Η είσοδος στον $t0 θα πρέπει να δίδεται μέσω της εντολής li (πχ li $t0, 0x1023B818), ενώ το τελικό αποτέλεσμα θα πρέπει να εκτυπώνεται σαν δεκαεξαδικός αριθμός στην κονσόλα μέσω της εντολής syscall.

Άσκηση 2 – Προσπέλαση πινάκων (10 μονάδες)

Να αναπτύξετε ένα πρόγραμμα που θα ζητάει αρχικά από τον χρήστη το μέγεθος (N) ενός τετραγωνικού πίνακα NxN και στη συνέχεια τα στοιχεία που θα αποθηκευτούν στις N^2 θέσεις αυτού του πίνακα. Ζητείται έπειτα το μέγεθος (M) ενός δεύτερου τετραγωνικού πίνακα MxM, ο οποίος είναι υποπίνακας εντός του αρχικού πίνακα. Ο χρήστης θα πρέπει επιπλέον να εισάγει την απόσταση του πρώτου στοιχείου του υποπίνακα από το αριστερό, αλλά και από το άνω άκρο του αρχικού πίνακα, ώστε να μπορεί να προσδιοριστεί η ακριβής θέση του υποπίνακα. Το πρόγραμμα θα αντιγράφει τον αρχικό πίνακα σε έναν νέο πίνακα, με τη διαφορά ότι στη θέση του υποπίνακα θα τοποθετεί έναν νέο υποπίνακα όπου κάθε στοιχείο του υπολογίζεται από την **Error! Reference source not found.**. Πιο συγκεκριμένα, κάθε στοιχείο σύμφωνα με την **Error! Reference source not found.** αντικαθίσταται με το άθροισμα των τεσσάρων γειτονικών του στοιχείων (οριζόντια και κάθετα) συν το ίδιο το στοιχείο πολλαπλασιασμένο με 4 και όλο το άθροισμα διαιρεμένο με οκτώ (βλέπε παράδειγμα Εικόνα 1). Υποθέτουμε ότι ο εσωτερικός υποπίνακας δεν εφάπτεται στα όρια του μεγάλου πίνακα, ώστε να μην λάβουμε υπόψην τις περιπτώσεις που δεν υπάρχουν γείτονες σε κάποιες κατευθύνσεις. Τέλος, θα εκτυπώνεται ο τελικός πίνακας στην οθόνη. Η εμφάνιση πρέπει να γίνει έτσι, ώστε σε κάθε σειρά να εκτυπώνεται και μία γραμμή του πίνακα.

a[i][j] = (4 * a[i][j] + a[i-1][j] + a[i+1][j] + a[i][j-1] + a[i][j+1]) / 8 (1)

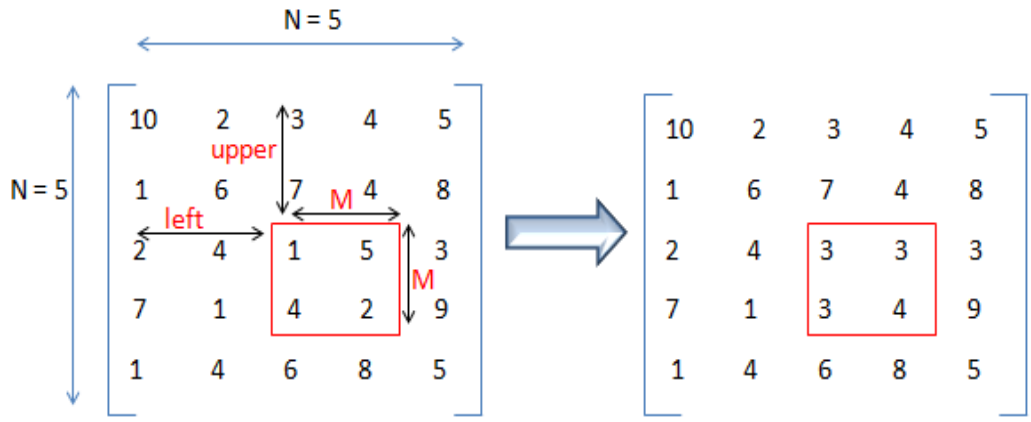


Εικόνα 1

Έστω ένας πίνακας 5x5 (N=5) και ένας υποπίνακας 2x2 (M=2) στο εσωτερικό του. Για να προσδιοριστεί η θέση του υποπίνακα σε σχέση με τον πίνακα θα πρέπει ο χρήστης να προσδιορίσει τις τιμές left και upper όπου εδώ είναι 2 και 2 αντίστοιχα.
Απεικονίζεται ο τύπος που χρησιμοποιείται για την αντικατάσταση του κάθε στοιχείου του υποπίνακα. Για παράδειγμα το στοιχείο 1 του υποπίνακα θα αντικατασταθεί από (4 * 1 + 7 + 4 + 4 + 5) / 8.

Σχηματικά:

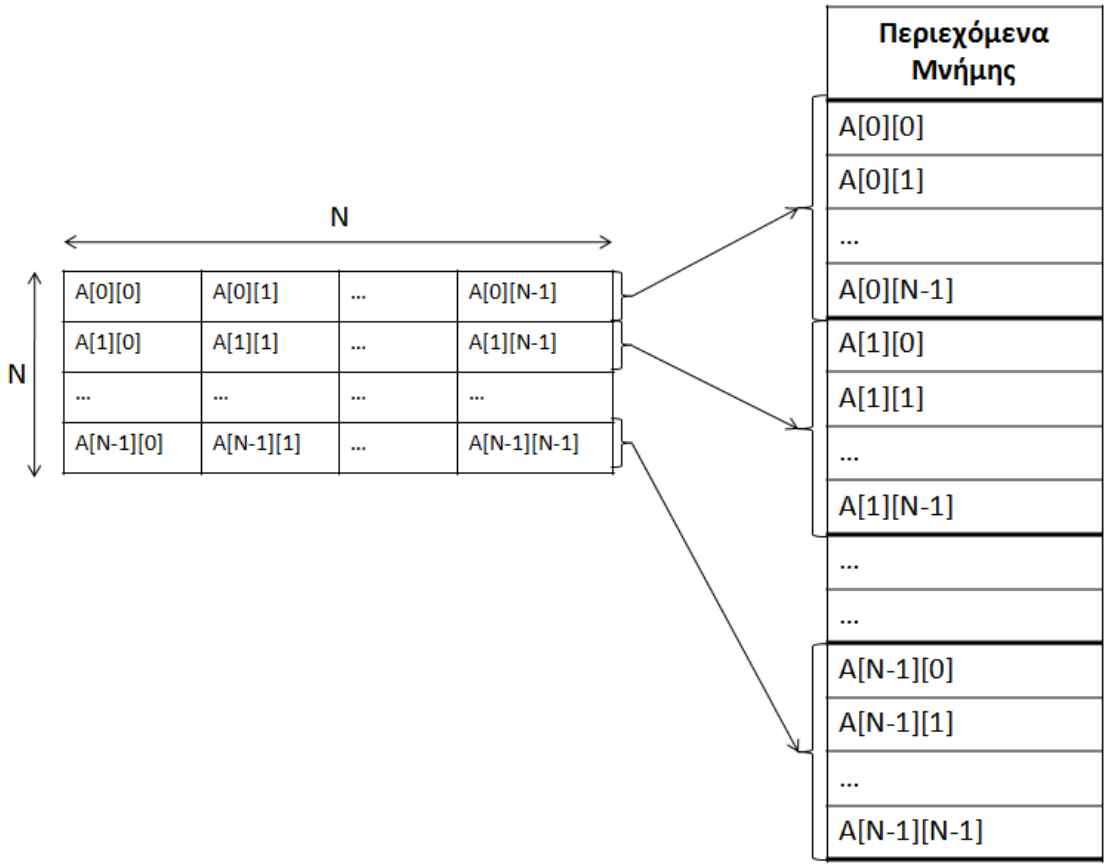
Έστω ένας πίνακας 5x5 (N=5) και ένας υποπίνακας 2x2 (M=2) στο εσωτερικό του. Για να προσδιοριστεί η θέση του υποπίνακα σε σχέση με τον πίνακα θα πρέπει ο χρήστης να προσδιορίσει τις τιμές left και upper όπου εδώ είναι 2 και 2 αντίστοιχα. Το πρόγραμμα θα πρέπει να δημιουργεί και να εκτυπώνει έναν καινούριο πίνακα ο οποίος είναι αντίγραφο του αρχικού, αλλά με τον υποπίνακα τα στοιχεία του οποίου έχουν υπολογιστεί με **Error! Reference source not found.** στην θέση του αρχικού υποπίνακα.



Υπόδειξη:

Οι μονοδιάστατοι πίνακες αποθηκεύονται στην μνήμη έτσι ώστε το στοιχείο A[i] να βρίσκεται αμέσως μετά το στοιχείο A[i-1] του πίνακα. Η αποθήκευση ενός πίνακα Α δύο διαστάσεων γίνεται με το να αποθηκεύεται η γραμμή i-1 αμέσως πριν την γραμμή i.

Παράδειγμα πίνακα δύο διαστάσεων στην μνήμη(NxN):



Παράδειγμα εκτέλεσης:

Please define the dimensions of the matrix:
**** user input : 5
Now, you have to fill the matrix!
Please give an integer :
**** user input : 10
Please give an integer :
**** user input : 2
Please give an integer :
**** user input : 3
Please give an integer :
**** user input : 4
Please give an integer :
**** user input : 5
.
.
.
Please define the dimensions of the submatrix:

**** user input : 2
Please define the left distance:
**** user input : 2
Please define the upper distance:
**** user input : 2
The new matrix is:
10 2 3 4 5
1 6 7 4 8
2 4 3 3 3
7 1 3 4 9
1 4 6 8 5

Άσκηση 3 – Επεξεργασία Strings (10 μονάδες)

Να αναπτύξετε κώδικα σε MIPS assembly που θα διαβάζει από την κονσόλα δύο strings str1 και str2, και θα ψάχνει εαν το str1 μπορεί να αναπαραχθεί με οποιαδήποτε μετάθεση των χαρακτήρων του str2. Σε περίπτωση που αυτό συμβαίνει, το πρόγραμμά σας θα πρέπει να εκτυπώνει 1 (true), αλλιώς θα πρέπει να εκτυπώνει 0 (false). Για ευκολία, θεωρείστε ότι μόνο μικροί χαρακτήρες (lower characters) θα υπάρχουν στο string.

Για παράδειγμα:

str1: abcdef
str2: dfacbe
result: 1

str1: abcdef
str2: abdef
result:0

str1: aaa
str2: aa
result:0

Οι εργασίες θα πρέπει να γίνονται upload στο eclass αμέσως μετά την εξέταση σας στο εργαστήριο. Σε περίπτωση που θέλετε να κάνετε upload πολλαπλά αρχεία, θα πρέπει να τα κάνετε πρώτα zip.