Κώστας Κυριακολόπουλος (Bouyos)
mail: kyriakou@mhl.tuc.gr

1.
```
int fib(int n,int a,int b){
        if(n==1) return a;
        if(n==2) return b;
        return fib(n-1,b,a+b);
}
```

Μεταφράστε το διπλανό κομμάτι κώδικα σε εντολές assembly ακολουθώντας τις συμβάσεις της αρχιτεκτονικής συνόλου εντολών MIPS. Δικαιολογήστε τυχών επιλογές σας. Τα ορίσματα της συνάρτησης βρίσκονται στη στοίβα και η τιμή που επιστρέφεται πρέπει να τοποθετηθεί σε αυτή. Δείξτε καθαρά τη χρήση της στοίβας κατά την εκτέλεση του προγράμματος.

2.
```
int min,max,temp;
int A[128],B[128];

ScanArrays(int * A, int * B, int N){
      temp=0;

      for(i=0; i<N; i++){
            if( A[i] > B[i] )
                  A[i]=B[i];
            temp+=A[i];
      }
}
```

Μεταφράστε το ακόλουθο κομμάτι κώδικα C σε εντολές assembly του MIPS. Δηλώστε τις μεταβλητές και τους πίνακες και δώστε τους ετικέτες (labels) με το ίδιο όνομα όπως και στον κώδικα C. Χρησιμοποιήστε συμβολικά ονόματα για τους καταχωρητές και χρησιμοποιήστε τις συμβάσεις χρήσης καταχωρητών του MIPS. Παρατήρηση: όλες οι μεταβλητές αρχικά βρίσκονται στη μνήμη.

3.
```
int swap_r (int * A, int * B, int index)
{
        int tmp = 0, changed = 0;
        if (A[index] < B[index] ) {
              changed = 1;
        }
        index --;
        if (index > 0 )
            changed = changed +
                  swap_r (A, B, index);
        return changed;
}
```

Μεταφράστε το διπλανό κομμάτι κώδικα σε εντολές assembly ακολουθώντας τις συμβάσεις της αρχιτεκτονικής συνόλου εντολών MIPS. Δικαιολογήστε τυχών επιλογές σας. Οι μεταβλητές tmp και changed είναι τοπικές ενώ τα ορίσματα της συνάρτησης βρίσκονται στη στοίβα και η τιμή που επιστρέφεται πρέπει να τοποθετηθεί σε αυτή. Χρησιμοποιήστε καταχωρητές όπου μπορείτε και δώστε ένα

πίνακα με τη χρήση τους. Δείξτε καθαρά τη χρήση της στοίβας κατά την εκτέλεση του προγράμματος.

4. Θεωρείστε τη δομή, σε γλώσσα προγραμματισμού C, που ορίζεται παρακάτω. Θεωρείστε ότι ο compiler είναι αρκετά έξυπνος ώστε να τοποθετεί τα πεδία της δομής με τέτοια σειρά που να μειώνει όσο γίνεται το χώρο της μνήμης που θα ξοδεύεται.

```
struct node {
      int value;
      struct node *leftChild;
      struct node *rightChild;
      char day[5];
      int years[2];
}
```

Μεταφράστε το ακόλουθο κομμάτι κώδικα σε εντολές assembly ακολουθώντας τις συμβάσεις της αρχιτεκτονικής συνόλου εντολών MIPS. Δικαιολογήστε τυχών επιλογές σας.

```
while ((p->value < guard) && (p->years[1] !=1000))
{
        if (p->value > 100)
                p = p-> leftChild;
        else
                p = p-> rightChild;
}
```

O Pointer p είναι pointer σε structure node. Έχει ήδη αρχικοποιηθεί να δείχνει στο πρώτο στοιχείο ενός δέντρου από structure nodes και βρίσκεται στο \$18. Το guard είναι γενική (global) μεταβλητή, τύπου int, αποθηκευμένη στη διεύθυνση μνήμης 4224. Δίπλα σε κάθε εντολή assembly βάλτε μια σύντομη εξήγηση του τι κάνει αυτή.

Οι καταχωρητές που δεν μπορούμε να χρησιμοποιήσουμε:

\$0, \$31, \$1

(\$31) εκτός και αν μπορούμε να το κάνουμε σωστό και δεν μας ζητάτε να ακολουθήσουμε τις συμβάσεις

(\$1) Χρησιμοποιείται από ψευδοεντολές

```
fib:
#####################################################PROLOGOS
    lw      $t1, 8($sp)     #$t1=n
    lw      $t2, 4($sp)     #$t2=a
    lw      $t3, 0($sp)     #$t3=b
    add     $sp, $sp, 12
#########################################################

if1:    bne     $t1, 1, if2
#####################################return $t2
    add     $sp, $sp, -4
    sw      $t2, 0($sp)
    jr      $ra
#########################################

if2:    bne     $t1, 2, anadr
#####################################return $t3
    add     $sp, $sp, -4
    sw      $t3, 0($sp)
    jr      $ra
#########################################

anadr:  add     $sp, $sp, -4
        sw      $ra, 0($sp)
        addi    $t0, $t1, -1
        add     $sp, $sp, -4
        sw      $t0, 0($sp)
        add     $sp, $sp, -4
        sw      $t3, 0($sp)
        add     $t0, $t2, $t3
        add     $sp, $sp, -4
        sw      $t0, 0($sp)

        jal     fib
        lw      $t0, 0($sp)     #$t0=fib(n-1,b,a+b)
        add     $sp, $sp, 4

        lw      $ra, 0($sp)
        add     $sp, $sp, 4
#############################################return $t0
    add     $sp, $sp, -4
    sw      $t0, 0($sp)
    jr      $ra
#############################################
```

pop × 3

σε αυτό (*) το push (το δικό μου)

σε αυτό (*) το push (το δικό μου)

(**)

push

push

push

εδώ γίνεται αυτό (*) το pop και πέρνει το αποτέλεσμα

pop (**)

```
.data

        .globl min
        .globl max
        .globl temp
        .globl A
        .globl B

min:    .space 4
max:    .space 4
temp:   .space 4
A:      .space 512
B:      .space 512

        .text

ScanArrays:
        la      $t0, temp       #$t0=* temp
        lw      $t1, 0($t0)     #$t1=temp
        li      $t1, 0          #temp=0

        li      $t2, 0          #$t2=i=0
for:    bge     $t2, $a2, end_for
##############################################Ypologizw A[i] kai B[i]
        sll     $t3, $t2, 2     #$t3=i*4
        add     $t4, $a0, $t3   #$t4=A+i*4
        add     $t5, $a1, $t3   #$t5=B+i*4
        lw      $t6, 0($t4)     #$t6=A[i]
        lw      $t7, 0($t5)     #$t7=B[i]

        ble     $t6, $t7, next
        move    $t6, $t7 #A[i]=B[i]
        sw      $t6, 0($t4)     #A[i]=B[i]
next:   add     $t1, $t1, $t6   #temp=temp+1;
        addi    $t2, $t2, 1
        j       for

end_for:
        sw      $t1, 0($t0)
        jr      $ra
```

```
swap_r:
#########################################################PROLOGOS
        lw      $t2, 0($sp)     #$t2=index
        add     $sp, $sp, 4     #1st pop
        lw      $t1, 0($sp)     #$t1=* B
        add     $sp, $sp, 4     #2nd pop
        lw      $t0, 0($sp)     #$t0=* A
        add     $sp, $sp, 4     #3rd pop
        #push $ra
        add     $sp, $sp, -4
        sw      $ra, 0($sp)
#################################################################
        add     $sp, $sp, -4    #push an integer->push tmp
        sw      $0, 0($sp)      #temp=0

        add     $sp, $sp, -4    #push an integer->push changed
        sw      $0, 0($sp)      #changed=0
################Ypologizw A[index] kai B[index]
        sll     $t3, $t2, 2     #$t3=4*index
        add     $t4, $t0, $t3   #$t4=A+4*index
        add     $t5, $t1, $t3   #$t5=B+4*index
        lw      $t6, 0($t4)     #$t6=A[index]
        lw      $t7, 0($t4)     #$t7=B[index]


        bge     $t6, $t7, endif1
        li      $t8, 1
        sw      $t8, 0($sp)     #changed=1
endif1: addi    $t2, $t2, -1    #index--
        ble     $t2, $0, endif2 #if(index>0)

#####################################klisi swap_r
        #push A
        add     $sp, $sp, -4
        sw      $t0, 0($sp)
        #push B
        add     $sp, $sp, -4
        sw      $t1, 0($sp)
        #push index
        add     $sp, $sp, -4
        sw      $t2, 0($sp)
        jal     swap_r
        #pop apotelesma
        lw      $t4, 0($sp)     #$t4=apotelesma
        add     $sp, $sp, 4
#####################################telos klisis
```

```
        lw      $t5, 0($sp)     #$t5=changed
        add     $t5, $t5, $t4   #changed-changed+1
        sw      $t5, 0($sp)     #enimerwsi changed
endif2: lw      $t3, 0($sp)     #$t3=changed
        add     $sp, $sp, 8     #2 pops(changed kai tmp)
##########################################epilogos
        #pop $ra
        lw      $ra, 0($sp)
        add     $sp, $sp, 4
        #push changed
        add     $sp, $sp, -4
        sw      $t3, 0($sp)
        jr      $ra
```

```
while:  lw      $t0, 0($18)     #$t0=p->value
        li      $t1, 4224
        lw      $t2, 0($t1)     #$t2=guard
        bge     $t0, $t2, contin
        lw      $t3, 24($18)    #$t3=p->years[1]
        beq     $t3, 1000, contin

if:     ble     $t0, 100, else
        lw      $t4, 4($18)     #$t4=p->leftChild
        move    $18, $t4        #p=p->leftChild
        j       endif
else:   lw      $t5, 4($18)     #$t5=p->rightChild
        move    $18, $t5        #p=p->rightChild
endif:
        j       while
contin:
```