ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

1η ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ ΣΤΗΝ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2014-2015, 5ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

(τμήμα Λ – Ω)

Ημερομηνία Παράδοσης: 5/12/2014

Απορίες στο: ca2014-2015-tmima2@cslab.ece.ntua.gr

Άσκηση 1η

Δίνεται η ακόλουθη συνάρτηση γραμμένη σε C, καθώς και η αντίστοιχη μετάφρασή της σε assembly MIPS. Συμπληρώστε τα κενά. Θυμίζουμε ότι ο καταχωρητής $zero απεικονίζει μόνο την τιμή 0 και δεν μπορεί να αλλάξει.

```
int count (char * x, char key){
     int counter=0;
     while (*x != `\0'){
          if (*x==key)
             counter++;
     }
     x++;
     return counter;
}
```

```
count:
     addi $sp, $sp, -4
     sw $s0, 0($sp)
     add $t3, $zero, $zero #$t3='\0'
     add $s0, $zero, $zero  #counter=0
loop:
     lb $t1, 0($a0)          #load x[i]
     beq $t1, $t3, exit      #x[i]=='\0'?
     bne $t1, $a1, goon      #*x==key?
     addi $s0, $s0, 1        #counter++
goon:
     addi $a0, $a0, 1        #x+=1
     j loop
```

```
exit:
    add $v0, $zero, $s0 #return counter
    lw $s0, 0($sp)      #stack recovery
    addi $sp, $sp, 4
    jr $ra              #return to caller
```

Άσκηση 2η

Η ακολουθία F_n των αριθμών Fibonacci ορίζεται από τον ακόλουθο αναδρομικό τύπο:

$$F_n = F_{n-1} + F_{n-2} \quad με \quad F_0 = 0, \qquad F_1 = 1$$

Δίνεται ο αναδρομικός και ο επαναληπτικός αλγόριθμος υπολογισμού του ν-οστού όρου της ακολουθίας υλοποιημένος σε γλώσσα C.

```
int fib(n){
    if (n<2)
        return n;
     else
       return fib(n-2) + fib(n-
1);
}
```

```
int fib(int n)
 {
   int i = 1, j = 0, k, t;
   for (k = 1; k <= n; k++)
   {
     t = i + j;
     i = j;
     j = t;
   }
   return j;
 }
```

Να υλοποιηθούν οι αντίστοιχες διαδικασίες σε assembly MIPS. Υποθέστε ότι οι παραπάνω διαδικασίες περιορίζονται στην επιστροφή 32bit λέξεων, αγνοείστε περιπτώσεις υπερχείλισης και θεωρείστε ορθές εισόδους.

```
#the following is the iterative Fibonacci procedure
fib:    addiu $t1, $zero, 1     # i=1
        addu  $t2, $zero, $zero  # j=0
        addiu $t3, $zero, 1     # k=1
loop:   bgt   $t3, $a0, exit    #k>n?
        addu  $t4, $t2, $t1     #t=i+j
        addu  $t1, $t2, $zero   #i=j
        addu  $t2, $t4, $zero   #j=t
        addiu $t3, $t3, 1       #k+=1
        j loop
exit:   addu $v0, $zero, $t2
        jr $ra
```

```
#the following is the recursive Fibonacci procedure
fib2:       addi $t1, $zero, 2
            bge  $a0, $t1, calcPrevious
            add  $v0, $a0, $zero # return
            jr $ra
calcPrevious:   addi $sp, $sp, -12
                sw $ra, 4($sp)          # return -> stack
                sw $s1, 8($sp)
                sw $a0, 12($sp)
                addi $a0, $a0, -1       # n-1
                jal fib2                # fib(n-1)
                add $s1, $v0, $zero     # temp store
                addi $a0, $a0, -1 # fib(n-2)
                jal fib2
                add $v0, $v0, $s1       # return fib(n-1)+fib(n-2)
                lw $a0, 12($sp)
                lw $s1, 8($sp)
                lw $ra, 4($sp)  # restore the return value
                addi $sp, $sp, 12
                jr $ra
```

Άσκηση 3η

Οι παρακάτω συναρτήσεις (υλοποιημένες σε C) προσφέρουν ακριβώς την ίδια λειτουργικότητα. Ποια είναι η λειτουργικότητα αυτή και ποια η διαφορά στην κάθε υλοποίηση; Δώστε τις αντίστοιχες υλοποιήσεις σε assembly MIPS. Επισημαίνεται ότι ένας πίνακας void αντιμετωπίζεται ως πίνακας από byte.

```
void name1(char array[],
unsigned int size, char
array2[])
{
    int i=0;
    for ( i=0; i<size; i++ )
       array2[i] = array[i]
    return;
}
```

```
void name2(char * array, unsigned int
size, char * array2)
{
    char *p;
    char *f;
    for (p=array, f=array2; p<(array +
size) ; p++, f++)
        *f = *p
   return;}
```

Η λειτουργικότητα που υλοποιούν οι δύο παραπάνω διαδικασίες είναι η αντιγραφή μέρους της μνήμης, με μέγεθος size bytes, από τη θέση array στη θέση array2. Όπως φαίνεται η αντιμετώπιση των περιεχομένων της μνήμης είναι ανεξάρτητη από το είδος αυτών καθότι γίνεται byte προς byte. Η διαφορά των δύο υλοποιήσεων έγκειται στο ότι η πρώτη χρησιμοποιεί αριθμοδεικτοδότηση πινάκων (array indexing) ενώ η δεύτερη αριθμητική απευθείας δεικτοδότησης στη μνήμη. Ακολουθούν snippets με τους αντίστοιχους κώδικες σε assembly MIPS.

```
#the arguments are: ($a0 --> array[], $a1 --> size,
$a2 --> array2[])
name1:
        add $t1, $zero, $zero #int i=0
loop:
        bge $t1, $a1, exit  #i<size ?
        add $t2, $a0, $t1  #array[i]
        add $t3, $a2, $t1 #array2[i]
        lb $t4, 0($t2) #array2[i]=array[i]
        sb $t4, 0($t3)
        addi $t1, $t1, 1
        j loop
exit:   jr $ra    #return
```

```
name2:
        add $t0, $a0, $zero # p = & array[0]
        add $t1, $a2, $zero      # f= & array2[0]
        add $t2, $t0, $a1   # p+size
loop2:
        bge $t0, $t2, exit2 # p<p+size?
        lb      $t4, 0($t0)      # *p
        sb      $t4, 0($t1)      # *f=*p
        addi $t0, $t0, 1   # p++
        addi $t1, $t1, 1  # f++
        j loop2
exit2:  jr $ra              # return
```

Σημείωση: Για τη διευκόλυνσή σας, προτρέπουμε να χρησιμοποιήσετε το προσομοιωτή Qtspim (http://spimsimulator.sourceforge.net/)

Παραδοτέο της άσκησης θα είναι **ηλεκτρονικό κείμενο** (pdf, docs ή odt) που θα περιέχει τις απαντήσεις των τριών μερών. Το έγγραφο πρέπει να φέρει τα στοιχεία σας (όνομα, επώνυμο και αριθμό μητρώου).

Οι κώδικες που θα παραδοθούν οφείλουν να είναι σε ευανάγνωστη μορφή και να φέρουν επαρκή σχόλια.