ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΜΠ

"Αρχιτεκτονική Υπολογιστών"

Ορφανουδάκης Φίλιππος 03113140

1η Άσκηση

Κώδικας C:

```
int odd_sum(int *a, int size)
 {
            int sum = 0, i, *p;
            for (p = a, i = 0; i < size; p++, i++)
                {
                  if (*p % 2 != 0) {
                          sum += *p;
                       }
                }
return sum;
}
```

Assembly του MIPS :

Σε κόκκινο τα συμπληρωμένα κενά

```
odd_sum:    add $t0, $zero, $zero     # sum = 0
            add $t1, $a0, $zero       # p = a
            add $t2, $zero, $zero     # i = 0
loop:       slt $t3, $t2, $a1         # i < size ?
            beq $t3, $zero, exit      #  if yes then jump to exit
            lw $t4, 0($t1)            # t4 = *p
            div $t4, $s0              # *p / 2
            mfhi $t5                  # modulo sent to t5
            beq $t5, $zero, cont      # if t5 == 0 jump to cont
            add $t0, $t0, $t4         # sum+= *p
cont:       addi $t1, $t1, 4          # t1 is a pointer , in order to point to the next
                                      # block we must add 4 (integer = 4 bytes )
            addi $t2, $t2, 1          # i++
             j loop                   # jump to loop
exit:       add $v0, $zero, $t0       # v0 = t0 = sum , v0 is the return value
             jr $ra                   # return address
```

2η Άσκηση

- Θα πραγματοποιήσουμε in-place αντιστροφή πίνακα , δηλαδή δεν θα χρησιμοποιήσουμε άλλη μνήμη στην αποθήκευση του νεου αντεστραμμένου πίνακα.
- Για να συμβεί αυτό και για την διευκόλυνση μας θα υλοποιήσουμε πρώτα την αντίστοιχη συνάρτηση C και θα την μετατρέψουμε σε κώδικα assembly MIPS .
- Σαν ορίσματα ακολουθούμε την εκφώνηση και έχουμε πρώτα τον πίνακα που σε κώδικα assembly μεταφράζεται στην πρώτη θέση του πίνακα και το μέγεθος του πίνακα.

Κώδικας C :

```
void reverseArray ( int A[], int size)
{

  int i = size - 1;
  int j = 0;
  while(i > j)
  {
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
    i--;
    j++;
  }
}
```

Assembly του MIPS :

```
reverseArray:   add $t0, $zero, $a1       # t0 = i =size
                sll $t0, $t0, 2           # size*4
                addi $t0, $t0, -4         # t0 = i =4*( size -1)
                add $t1, $zero , $zero    # t1 = j  = 0

loop:           slt $t2 , $t1, $t0        # j<i
                bne $t2, $zero, exit      # if not , jump to exit
                add $t3, $a0, $t0         # t3 = a+i
                add $t4, $a0, $t1         # t4 = a+j
                lw $t5, 0($t3)            # t5 = a[i]
                lw $t6, 0($t4)            # t6 = a[j]
                sw $t5 , 0($t4)           # a[j] = a[i]
                sw $t6 , 0($t3)           # a[i] = a[j]
                addi $t0,$t0,-4           # i-- (= i-4 because we have pointers and
                                          #         integers)
                addi $t1,$t1,4            # j++ ( same reason)
                j loop


exit:           jr $ra                    # return address
```

- **Παρατηρούμε πως δεν χρειάζεται η μεταβλητή temp !!**

3η Άσκηση

Μας ζητείται να μετατρέψουμε τις παρακάτω ρουτίνες που ειναι σε C σε Assembly του MIPS, επίσης οι αριθμοί που διαχειριζόμαστε είναι ακέραιοι.

Κώδικας C

```
void swap(int *m1, int *m2) {
         int tmp = *m1;
         *m1 = *m2;
         *m2 = tmp;
}


// To heapify a subtree rooted with node i which is an index in arr[].
// n is the size of heap
void heapify(int arr[], int n, int i) {
          int largest = i; // initialize largest as root
          int l = 2 * i + 1; // left = 2 * i + 1
          int r = 2 * i + 2; // right = 2 * i + 2

         // if left child is larger than root
         if (l < n && arr[l] > arr[largest]) {
               largest = l;
         }

        // if right child is larger than largest so far
        if (r < n && arr[r] > arr[largest]) {
              largest = r;
         }

       // if largest is not root
       if (largest != i) {
             swap(&arr[i], &arr[largest]);
             heapify(arr, n, largest); // recursively heapify the

                                       //affected sub-tree
         }
```

```
}

// main function to implement heapsort
void heapsort(int arr[], int n) {
	int i;

	// build heap (rearrange array)
	for (i = n / 2 - 1; i >= 0; i--) {
	heapify(arr, n, i);
	}

	// one by one extract an element from heap
	for (i = n - 1; i >= 0; i--) {
	swap(&arr[0], &arr[i]); // move current root to end
	heapify(arr, i, 0); // call max heapify on the reduced heap
	}
}
```

Assembly του MIPS :

```
swap:	lw $t0, 0($a0)		# t0 = *m1
	lw $t1, 0(a1)		# t1 = *m2
	sw $t1, 0($a0)		# *m1 = *m2
	sw $t0, 0($a1)		# *m2 = *m1

	jr $ra
```

ΠΑΡΑΤΗΡΗΣΕΙΣ :

- **ΜΕΣΑ ΣΤΗ heapify ΔΙΑΣΦΑΛΙΖΕΤΑΙ Η ΣΥΝΕΠΕΙΑ ΟΛΩΝ ΤΩΝ ΕΥΑΙΣΘΗΤΩΝ ΜΕΤΑΒΛΗΤΩΝ (a0,a1,a2,s0,s1,s2,ra) ΜΕ ΤΗ ΧΡΗΣΗ ΤΗΣ ΣΤΟΙΒΑΣ ΠΡΙΝ ΤΟ ΚΑΛΕΣΜΑ ΚΑΘΕ ΥΠΟΡΟΥΤΙΝΑΣ , ΣΕ ΟΡΙΣΜΕΝΕΣ ΠΕΡΙΠΤΩΣΕΙΣ ΜΠΟΡΕΙ ΝΑ ΜΗΝ ΕΙΝΑΙ ΑΠΑΡΑΙΤΗΤΟ , BETTER BE SAFE THAN SORRY.**
- **ΗΔΗ ΑΠΟ ΤΗΝ heapshort ΕΧΕΙ ΓΙΝΕΙ Η ΜΕΤΑΤΡΟΠΗ ΤΩΝ ΜΕΤΑΒΛΗΤΩΝ ΠΟΥ ΑΝΑΦΕΡΟΝΤΑΙ ΣΕ index ΠΙΝΑΚΩΝ ΣΕ x4 ΕΠΟΜΕΝΩΣ ΔΕΝ ΕΠΑΝΑΛΑΜΒΑΝΕΤΑΙ ΑΛΛΑ ΚΑΘΕ ΑΥΞΟΜΕΙΩΣΗ ΓΙΝΕΤΑΙ ΜΕ +- 4.**

```
heapify:	add $s0, $zero, $a2		# s0 = largest = i
```

```
                sll $s1, $s0, 1          # s1 = 2*i
                addi $s1, $s1, 4         # s1 = 2*i + 4 = l
                sll $s2, $s2, 1          # s2 = 2*i
                addi $s2, $s2, 4         # s2 = 2*i +4 = r

                slt $t0, $s1, $a1        # l<n
                bne $t0,$zero, next      # if yes then next
                j cont                   # if no cont

next:           add $t0, $a0, $s1        # t0 = arr + l
                lw $t3, 0($t0)           # t3 = arr[l]
                add $t1, $a0, $s0        # t1 = arr + largest
                lw $t4, 0($t1)           # t4 = arr[largest]
                slt $t2, $t4, $t3        # arr[largest]<arr[l]
                beq $t2, $zero , cont    # if no go to cont
                add $s0, $zero, $s1      # if yes largest = l

cont:           slt $t0, $s2, $a1        # r<n
                bne $t0, $zero, next1    # if yes then next1
                j cont1                  # if no cont1

next1:          add $t0, $a0, $s2        # t0 = arr + r
                lw $t3, 0($t0)           # t3 = arr[r]
                add $t1, $a0, $s0        # t1 = arr + largest
                lw $t4, 0($t1)           # t4 = arr[largest]
                slt $t2, $t4, $t3        # arr[largest]<arr[r]
                beq $t2, $zero , cont1   # if no go to cont
                add $s0, $zero, $s2      # if yes largest = r

                beq $s0, $a2,end         # largest == i ? , if yes go to end


                addi $sp, $sp, -28       # stack - 28
                sw $a0, 0($sp)           # save a0
                sw $a1, 4($sp)           # save a1
                sw $a2, 8($sp)           # save a2
                sw $s0, 12($sp)          # save s0
                sw $s1, 16($sp)          # save s1
                sw $s2, 20($sp)          # save s2
                sw $ra, 24($sp)          # save ra

                add $a0, $a0, $a2        # a0 = arr + i*4
                add $a1, $a0, $s0        # a1 = arr + largest
                jal swap                 # jump to swap with the right arguments

                lw $a0, 0($sp)           # a0 = arr
```

```
                lw $a1, 4($sp)          # a1 = 4*n
                lw $a2, 8($sp)          # a2 = 4*i
                lw $s0, 12($sp)         # s0 = largest
                lw $s1, 16($sp)         # s1 = l
                lw $s2, 20($sp)         # s2 = r
                lw $ra, 24($sp)         # restore ra
                addi $sp, $sp, 28       # stack +28

                addi $sp, $sp, -28      # stack - 28
                sw $a0, 0($sp)          # save a0
                sw $a1, 4($sp)          # save a1
                sw $a2, 8($sp)          # save a2
                sw $s0, 12($sp)         # save s0
                sw $s1, 16($sp)         # save s1
                sw $s2, 20($sp)         # save s2
                sw $ra, 24($sp)         # save ra

                add $a2, $zero, $s0     # a2 = largest
                jal heapify             # jump to heapify with the right arguments

                lw $a0, 0($sp)          # a0 = arr
                lw $a1, 4($sp)          # a1 = 4*n
                lw $a2, 8($sp)          # a2 = 4*i
                lw $s0, 12($sp)         # s0 = largest
                lw $s1, 16($sp)         # s1 = l
                lw $s2, 20($sp)         # s2 = r
                lw $ra, 24($sp)         # restore ra
                addi $sp, $sp, 28       # stack +28

end:            jr $ra                  # return address
```

ΠΑΡΑΤΗΡΗΣΕΙΣ :

- **ΜΕΣΑ ΣΤΗ heapsort ΔΙΑΣΦΑΛΙΖΕΤΑΙ Η ΣΥΝΕΠΕΙΑ ΟΛΩΝ ΤΩΝ ΕΥΑΙΣΘΗΤΩΝ ΜΕΤΑΒΛΗΤΩΝ (a0,a1,s0,ra) ΜΕ ΤΗ ΧΡΗΣΗ ΤΗΣ ΣΤΟΙΒΑΣ ΠΡΙΝ ΤΟ ΚΑΛΕΣΜΑ ΚΑΘΕ ΥΠΟΡΟΥΤΙΝΑΣ , ΣΕ ΟΡΙΣΜΕΝΕΣ ΠΕΡΙΠΤΩΣΕΙΣ ΜΠΟΡΕΙ ΝΑ ΜΗΝ ΕΙΝΑΙ ΑΠΑΡΑΙΤΗΤΟ , BETTER BE SAFE THAN SORRY.**

- **ΓΙΝΕΤΑΙ Η ΜΕΤΑΤΡΟΠΗ ΤΩΝ ΑΠΑΡΑΙΤΗΤΩΝ ΜΕΤΑΒΛΗΤΩΝ ΣΕ x4 ΓΙΑ ΤΗ ΣΩΣΤΗ ΧΡΗΣΗ ΠΙΝΑΚΩΝ ΚΑΙ ΔΕΝ ΠΡΕΠΕΙ ΝΑ ΕΠΑΝΑΛΗΦΘΕΙ ΜΕΣΑ ΣΤΙΣ ΥΠΟΡΟΥΤΙΝΕΣ**

```
heapsort:   div $a1,2           # n/2
            mflo $t0            # quotient in $LO so we move it to t0
            addi $t0, $t0, -1   # t0 = t0 -1
            sll $t0, $t0, 2     # t0 = (n/2 -1)*4
            add $s0,$zero,$t0   # s0 = i*4
            sll $a1, $a1, 2     # a1 = a1*4 = n*4

loop1:      addi $t0, $zero, -4 # t0 = -4
            slt $t0, $s0, $t0   # 4*i < -1
            bne $t0, $zero, next # if yes next
            addi $sp, $sp, -16  # stack -12
            sw $a0, 0($sp)      # save a0 , in case we change it in heapify
            sw $a1, 4($sp)      # save a1
            sw $s0, 8($sp)      # save s0
            sw $ra, 12($sp)     # save ra
            add $a2, $zero, $s0 # a2 = i*4
            jal heapify         # a0 = arr , a1 = n*4 , a2 = i*4
            lw $a0, 0($sp)      # a0 = arr
            lw $a1, 4($sp)      # a1 = n*4
            lw $s0, 8($sp)      # s0 = i*4
            lw $ra, 12($sp)     # restore ra
            addi $sp, $sp, 16   # stack + 16
            addi $s0, $s0, -4   # a2 = i = i -1
            j loop1             # jump loop1


next:       add $s0, $zero, $a2 # s0 = n
            addi $s0, $s0, -1   # s0 = n-1
            sll $s0, $s0 ,2     # s0 = 4*(n-1) = i

loop2:      addi $t4, $zero, -4 # t4 = -4
            slt $t5 , $s0, $t4  # i*4 < - 4
            bne $t5, $zero, exit # if yes then exit

            addi $sp, $sp , -16 # stack - 16
            sw $a0, 0($sp)      # save a0
            sw $a1, 4($sp)      # save a1
            sw $s0, 8($sp)      # save s0
            sw $jr. 12($sp)             # save ra

            add $t2, $a0 , $s0  # t2 = arr + i
            add $a1, $zero , $t2 # a1 = arr[i]
```

```
                jal swap                # go to swap with the right arguments

                lw $a0, 0($sp)          # a0= arr
                lw $a1, 4($sp)          # a1 = n * 4
                lw $s0, 8($sp)          # s0 = i * 4
                lw $ra, 12($sp)         # restore ra
                addi $sp, $sp, 16       # stack + 16

                addi $sp, $sp, -16      # stack - 16
                sw $a0, 0($sp)          # save a0
                sw $a1, 4($sp)          # save a1
                sw $s0, 8($sp)          # save s0
                sw $ra, 12($sp)         # save ra

                add $a1, $zero, $s0     # a1 = i
                addi $a2, $zero , $zero  # a2 = 0
                jal heapify             # go to heapify with the right arguments

                lw $a0, 0($sp)          # a0 = arr
                lw $a1, 4($sp)          # a1 = n*4
                lw $s0, 8($sp)          # s0 = i*4
                lw $ra, 12($ra)         # restore ra
                addi $sp, $sp, 8        # stack +16

                addi $s0, $s0, -4       # i--
                j loop2

exit:           jr $ra
```