

NTUA DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING



# Άσκηση 1η: Assembly MIPS

ΜΑΘΗΜΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Μαρίνα Φραγκούλη el22429  
18/11/2024

Τμήμα 3 (ΠΑΕ-Ω)

1. ΜΕΡΟΣ Α.....	2
2. ΜΕΡΟΣ Β.....	3
3. ΜΕΡΟΣ Γ.....	4
1) Η συνάρτηση CREATE_LEAVES.....	4
2) Η συνάρτηση CREATE_MERKLE_TREE .....	6
4. ΜΕΡΟΣ Δ.....	9
1) Το ζητούμενο string.....	9
2) Ο κώδικας σε c .....	9
3) Ο Κώδικας σε Assembly: .....	10
4) ΕΙΚΟΝΕΣ.....	15
5. Παράρτημα 1 ο συνολικός κώδικας Assembly .....	18
6. Παράρτημα 2 Δοκιμές .....	27

## 1. ΜΕΡΟΣ Α

```
int sum_min_arrays(int *x, int *y, int n)
{
    int min1 = x[0];
    int min2 = y[0];
    for(int i = 0; i < n-1; i++) {
        x++;
        y++;
        if (*x < min1)
            min1 = *x;
        if (*y < min2)
            min2 = *y;
    }
    return min1+min2;
}
```

Με ΓΚΡΙ: Σχόλια

Με ΜΠΑΕ: Συμπλήρωση κενών

```
# Prologue

addi $sp, $sp, -8 # Adjust stack pointer for 3 registers (3 * 4 bytes)
sw $s1, 4($sp) # Save $s1 at position 8($sp)
sw $s0, 0($sp) # Save $s0 at position 4($sp)

# Main Body

lw $s0, 0($a0) # Load min1 = x[0] into $s0
lw $s1, 0($a1) # Load min2 = y[0] into $s1
add $t3, $zero, $zero # Initialize i = 0 in $t3
subi $a2, $a2, 1 # Compute n - 1 and store in $t1

LOOP:
slt $t4, $t3, $a2 # if (i < n - 1)

beq $t4, $zero, END # if not, exit loop

addi $a0, $a0, 4 # x++ (wordlength 4)
addi $a1, $a1, 4 # y++ (wordlength 4)
lw $t0, 0($a0) # Load *x into $t0
lw $t1, 0($a1) # Load *y into $t1
slt $t4, $t0, $s0 # if (*x < min1)
beq $t4, $zero, CHECK2 # if not, skip updating min1
add $s0, $t0, $zero # min1 = *x

CHECK2:
slt $t4, $t1, $s1 # if (*y < min2)

beq $t4, $zero, NEXT # if not, skip updating min2

add $s1, $t1, $zero # min2 = *y
NEXT:
addi $t3, $t3, 1 # i++
j LOOP

END:
add $v0, $s0, $s1 # return min1 + min2

# Epilogue

lw $s0, 0($sp) # Restore $s0 from position 0($sp)
lw $s1, 4($sp) # Restore $s1 from position 4($sp)
addi $sp, $sp, 8 # Restore stack pointer
jr $ra # Return from function
```

## 2. ΜΕΡΟΣ Β

Στο αριστερά πλαίσιο φαίνεται η αντίστοιχη συνάρτηση σε C. Στο δεξιά φαίνεται ποιοι καταχωρητές χρησιμοποιήθηκαν για να αποθηκεύσουμε τις διάφορες τιμές του προγράμματος.

```
unsigned int cslab_hash(unsigned int input){  
    unsigned int hash = 5381;  
  
    int c;  
  
    while (input != 0) {  
        c = (input & 0xFF);  
        //bitwise and bit by bit comparison  
        hash = ((hash << 4) + hash) + c;  
        input = input >> 8;  
    }  
  
    return hash;  
}
```

```
$a0: input  
$t0: hash  
$t1: c  
$t2: (hash<<4 )  
$t3: ((hash << 4) + hash)  
$v0<-hash
```

CSLAB\_HASH:

```
addi $t0, $zero, 5381 #t0<-hash=5381
```

```
# $t1<-c
```

WHILE\_HASH:

```
beq $a0, $zero, END_HASH # while (input != 0)
```

```
andi $t1, $a0, 0xFF #t1<-c=(input & 0xFF)
```

```
sll $t2, $t0, 4 #t2<-(hash<<4 )
```

```
add $t2, $t2, $t1 #t2<-((hash << 4) + hash!!
```

```
add $t0, $t2, $t1 #t0<-hash=((hash << 4) + hash) + c;
```

```
srl $a0, $a0, 8 # input = input >> 8;
```

```
j WHILE_HASH
```

END\_HASH:

```
add $v0, $t0, $zero # return hash;
```

```
jr $ra # Return from function
```

### 3. ΜΕΡΟΣ Γ

#### 1) Η ΣΥΝΑΡΤΗΣΗ CREATE\_LEAVES

Στο αριστερά πλαίσιο φαίνεται η αντίστοιχη συνάρτηση σε C. Στο δεξιά φαίνεται ποιοι καταχωρητές χρησιμοποιήθηκαν για να αποθηκεύσουμε τις διάφορες τιμές του προγράμματος.

```
void create_leaves(unsigned int* data_array, int array_size, int* tree){  
  
    int i = 0;  
  
    while(i < array_size){  
  
        *tree = cslab_hash(*data_array);  
  
        data_array++;  
  
        tree++;  
  
        i++;  
  
    }  
  
    return;  
  
}
```

```
$a0: *data array  
$a1: array_size  
$a2: *tree  
$s0: i  
$t0: (i<array_size)
```

CREATE\_LEAVES:

addi \$sp, \$sp, -8 #κάνω χώρο (για \$ra) το \$sp γίνεται \$sp'

sw \$ra, 4(\$sp) #saves \$ra at position 4(\$sp')

sw \$s0, 0(\$sp) #saves \$s0 at position 0(\$sp')

addi \$s0, \$zero, 0 #s0<-i μετρητής

WHILE\_LEAVES:

slt \$t0, \$s0, \$a1 # (i < array\_size)

beq \$t0, \$zero, END\_LEAVES # while(i < array\_size)

addi \$sp, \$sp, -4

sw \$a0, 0(\$sp)

#Κλήση CSLAB\_HASH

lw \$a0, 0(\$a0)

jal CSLAB\_HASH

lw \$a0, 0(\$sp)

addi \$sp, \$sp, 4

#πλέον στο \$v0 <-hash

sw \$v0, 0(\$a2) # \*tree = cslab\_hash(\*data\_array);

addi \$a0, \$a0, 4 # data\_array++

addi \$a2, \$a2, 4 # tree++

addi \$s0, \$s0, 1 # i++

j WHILE\_LEAVES

END\_LEAVES:

lw \$ra, 4(\$sp) #loads \$ra from position 4(\$sp')

lw \$s0, 0(\$sp) #loads \$s0 from position 0(\$sp')

addi \$sp, \$sp, 8 #ελευθερώνω χώρο (για \$ra) το \$sp' γίνεται \$sp

jr \$ra # Return from function

## 2) Η ΣΥΝΑΡΤΗΣΗ CREATE\_MERKLE\_TREE

Στο αριστερά πλαίσιο φαίνεται η αντίστοιχη συνάρτηση σε C. Στο δεξιά φαίνεται ποιοι καταχωρητές χρησιμοποιήθηκαν για να αποθηκεύσουμε τις διάφορες τιμές του προγράμματος.

```
unsigned int create_Merkle_Tree(unsigned int* tree, int num_of_leaves){  
  
    int level_ops = num_of_leaves / 2;  
  
    unsigned int * tail = &tree[num_of_leaves];  
  
    int i = 0;  
  
    while (level_ops > 0){  
  
        int j = 0;  
  
        while (j < level_ops){  
  
            unsigned int xored_val = tree[i] ^ tree[i+1];  
  
            *tail = cslab_hash(xored_val);  
  
            i += 2;  
  
            j++;  
  
            tail++;  
  
        }  
  
        level_ops /= 2;  
  
    }  
  
    return *(tail-1);  
  
}
```

```
$a0 : *tree  
$a1: no_of_leaves  
$s0: i  
$s1: j  
$s2: no_of_leaves/2=level_ops  
$s3: tail=&tree[no_of_leaves*4]  
$t0: (i<level_ops)  
$t1: (level_ops>0)  
$t2: *(tree[i])  
$t3: *(tree[i+1])  
$t4: no_of_leaves*4  
$t6: tree[i]  
$t7: tree[i+1]  
$t9: i*4  
$v0: ROOT HASH
```

CREATE\_MERKLE\_TREE:

addi \$sp, \$sp, -20 #κάνω χώρο (για \$ra) το \$sp γίνεται \$sp'

sw \$ra, 16(\$sp) #saves \$ra at position 16(\$sp')

sw \$s0, 12(\$sp) #saves \$s0 at position 12(\$sp')

sw \$s1, 8(\$sp) #saves \$s1 at position 8(\$sp')

sw \$s2, 4(\$sp) #saves \$s2 at position 4(\$sp')

sw \$s3, 0(\$sp) #saves \$s3 at position 0(\$sp')

srl \$s2, \$a1, 1 #num\_of\_leaves / 2

sll \$t4, \$a1, 2 # num\_of\_leaves \*4 για να δείχνει διεύθυνση στοιχείου του πίνακα αφού wordlength 4

add \$s3, \$a0, \$t4# διεύθυνση του tree[num\_of\_leaves]

addi \$s0, \$zero, 0 # αρχικοποίηση i=0

WHILE\_I:

slt \$t1, \$zero, \$s2 #(level\_ops > 0)

beq \$t1, \$zero, END\_TREE

addi \$s1, \$zero, 0 #int j = 0;

j WHILE\_J

DONT\_FORGET:

srl \$s2, \$s2, 1 #level\_ops /= 2;

j WHILE\_I

WHILE\_J:

slt \$t0, \$s1, \$s2

beq \$t0, 0, DONT\_FORGET

sll \$t9, \$s0, 2 #i\*4

add \$t2, \$a0, \$t9 # \*tree[i]

addi \$t3, \$t2, 4 # \*tree[i+1]

lw \$t6, (\$t2) # tree[i]

lw \$t7, (\$t3) # tree[i+1]

xor \$t8, \$t6, \$t7 # xored\_val = tree[i] ^ tree[i+1]

addi \$sp, \$sp, -8 # Κάνε χώρο στη στοίβα



```

sw $ra, 4($sp)    # Αποθήκευσε $ra
sw $a0, 0($sp)    # Αποθήκευσε $a0
addi $a0, $t8, 0   # Δώσε το xored_val ως είσοδο
jal CSLAB_HASH     # Κάλεσε τη συνάρτηση hash

```

```

sw $v0, 0($s3)    # Αποθήκευσε το hash στο tail

```

```

addi $s3, $s3, 4   # tail++

```

```

lw $ra, 4($sp)     # Επαναφορά $ra
lw $a0, 0($sp)     # Επαναφορά $a0
addi $sp, $sp, 8    # Επαναφορά στοίβας

```

```

addi $s0, $s0, 2    # i += 2

```

```

addi $s1, $s1, 1    # j++

```

```

j WHILE_J

```

END\_TREE:

```

add $v0, $s3, -4    # Επιστροφή *(tail - 1)

```

```

lw $v0, ($v0)

```

```

lw $ra, 16($sp) #loads $ra from position 16($sp')

```

```

lw $s0, 12($sp) #loads $s0 from position 12($sp')

```

```

lw $s1, 8($sp) #loads $s1 from position 8($sp')

```

```

lw $s2, 4($sp) #loads $s2 from position 4($sp')

```

```

lw $s3, 0($sp) #loads $s3 from position 0($sp')

```

```

addi $sp, $sp, 20 #απελευθερώνω χώρο το $sp' γίνεται $sp

```

## 4. ΜΕΡΟΣ Δ

---

### 1) ΤΟ ΖΗΤΟΥΜΕΝΟ STRING

XXX NTUA ECE CA 2024 DDMM YYYY MIPS

429 NTUA ECE CA 2024 0610 2003 MIPS

όπου XXX τα 3 τελευταία ψηφία του αριθμού μητρώου σας (AM), DDMM η ημέρα και ο μήνας γέννησης σας και YYYY το έτος γέννησης σας.

---

### 2) Ο ΚΩΔΙΚΑΣ ΣΕ C

Ακολουθούν μόνο οι εντολές του προεπεξεργαστή και της main (οι συναρτήσεις που δίνονται βρίσκονται στα αντίστοιχα ερωτήματα):

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main(){
    unsigned int *data ;
    data = malloc(8 * sizeof(int));

    //MIPS is big-endian = most significant character in MSB
    data[0] = ('4' << 16) | ('2' << 8) | '9'; //3 last digits of AM (e.g. 193)
    data[1] = ('N' << 24) | ('T' << 16) | ('U' << 8) | 'A';
    data[2] = ('E' << 16) | ('C' << 8) | 'E' ;
    data[3] = ('C' << 8) | 'A' ;
    data[4] = ('2' << 24) | ('0' << 16) | ('2' << 8) | '4';
    data[5] = ('0' << 24) | ('6' << 16) | ('1' << 8) | '0'; //DDMM of date of birth
    data[6] = ('2' << 24) | ('0' << 16) | ('0' << 8) | '3'; //YYYY of date of birth
    data[7] = ('M' << 24) | ('I' << 16) | ('P' << 8) | 'S';
    unsigned int* tree = malloc(15 * sizeof(unsigned int));
    create_leaves(data, 8, tree);
    printf("Root hash: %d\n", create_Merkle_Tree(tree, 8));
}
```

```
    return 0;
}
```

---

### 3) Ο ΚΩΔΙΚΑΣ ΣΕ ASSEMBLY:

Ακολουθεί ο κώδικας σε ASSEMBLY παραλείπονται οι συναρτήσεις που φτιάχτηκαν στα προηγούμενα ερωτήματα. Τον κώδικα μαζεμένο με όλες τις συναρτήσεις μπορείτε να τον βρείτε στο ΠΑΡΑΡΤΗΜΑ 1:

```
.data
data: .space 32 #Χώρος για 8 κόμβους
no_of_leaves: .word 8 #Αποθηκεύω τον αριθμό των φύλλων
tree: .space 60 #Χώρος για 15 κόμβους (8 φύλλα + 4 ενδιάμεσοι 1ου επιπέδου + 2 ενδιάμεσοι 2ου επιπέδου + root) *4
wordlength

message: .ascii " Root hash: "

.text

.globl MAIN

MAIN:

    #Κλήση της CREATE_DATA
    la $a0, data    # Διεύθυνση του πίνακα data

    addi $sp, $sp, -4
    sw $ra, 0($sp)
    jal CREATE_DATA
    lw $ra, 0($sp)
    addi $sp, $sp, 4

    # Κλήση CREATE_LEAVES
    la $a0, data    # Διεύθυνση του πίνακα data
    lw $a1, no_of_leaves# Μέγεθος του πίνακα data
    la $a2, tree    # Διεύθυνση του πίνακα tree (φύλλα ξεκινούν από εδώ)

    addi $sp, $sp, -4
    sw $ra, 0($sp)
```

```

jal CREATE_LEAVES    # Κλήση της void CREATE_LEAVES

lw $ra, 0($sp)

addi $sp, $sp, 4


# Κλήση CREATE_MERKLE_TREE

la $a0, tree    # Διεύθυνση του πίνακα tree (φύλλα ξεκινούν από εδώ)

lw $a1, no_of_leaves# Μέγεθος του πίνακα data


addi $sp, $sp, -4

sw $ra, 0($sp)

jal CREATE_MERKLE_TREE    # Κλήση της int CREATE_LEAVES ($v0 <- ROOT)

lw $ra, 0($sp)

addi $sp, $sp, 4


#Εκτύπωση μηνύματος

addi $t0, $v0, 0


la $a0, message    # Load the address of the string into $a0

li $v0, 4    # Syscall code for printing a string

syscall    # Perform the syscall


la $a0, ($t0)    # Load the address of the ROOT saved in $t0

li $v0, 1    # Syscall code for printing a string

syscall    # Perform the syscall


# Τερματισμός

li $v0, 10    # Σύστημα κλήσης: τερματισμός

syscall

```

CREATE\_DATA:

```

# Υπολογισμός ('4' << 16) | ('2' << 8) | '9'

li $t0, '4'    # Φόρτωσε τον χαρακτήρα '4' γίνεται και με addi $t0, $zero, '4'

```

```

sll $t0, $t0, 16    # Μετακίνησε 16 bits αριστερά

li $t1, '2'        # Φόρτωσε τον χαρακτήρα '2'

sll $t1, $t1, 8     # Μετακίνησε 8 bits αριστερά

or $t0, $t0, $t1    # Συνδυασμός '4' και '2'

li $t1, '9'        # Φόρτωσε τον χαρακτήρα '9'

or $t0, $t0, $t1    # Συνδυασμός με '9'


sw $t0, ($a0)       # Αποθήκευση στο data[0]

addi $a0, $a0, 4


# Υπολογισμός ('N' << 24) | ('T' << 16) | ('U' << 8) | 'A'

li $t0, 'N'

sll $t0, $t0, 24

li $t1, 'T'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, 'U'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, 'A'

or $t0, $t0, $t1


sw $t0, ($a0)       # Αποθήκευση στο data[1]

addi $a0, $a0, 4


# Υπολογισμός ('E' << 16) | ('C' << 8) | 'E'

li $t0, 'E'

sll $t0, $t0, 16

li $t1, 'C'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, 'E'

```

```
or $t0, $t0, $t1
```

```
sw $t0, ($a0)    # Αποθήκευση στο data[2]
```

```
addi $a0, $a0, 4
```

```
# Υπολογισμός ('C' << 8) | 'A'
```

```
li $t0, 'C'
```

```
sll $t0, $t0, 8
```

```
li $t1, 'A'
```

```
or $t0, $t0, $t1
```

```
sw $t0, ($a0)    # Αποθήκευση στο data[3]
```

```
addi $a0, $a0, 4
```

```
# Υπολογισμός ('2' << 24) | ('0' << 16) | ('2' << 8) | '4'
```

```
li $t0, '2'
```

```
sll $t0, $t0, 24
```

```
li $t1, '0'
```

```
sll $t1, $t1, 16
```

```
or $t0, $t0, $t1
```

```
li $t1, '2'
```

```
sll $t1, $t1, 8
```

```
or $t0, $t0, $t1
```

```
li $t1, '4'
```

```
or $t0, $t0, $t1
```

```
sw $t0, ($a0)    # Αποθήκευση στο data[4]
```

```
addi $a0, $a0, 4
```

```
# Υπολογισμός ('0' << 24) | ('6' << 16) | ('1' << 8) | '0'
```

```
li $t0, '0'
```

```
sll $t0, $t0, 24
```

```

li $t1, '6'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, '1'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, '0'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[5]

addi $a0, $a0, 4


# Υπολογισμός ('2' << 24) | ('0' << 16) | ('0' << 8) | '3'

li $t0, '2'

sll $t0, $t0, 24

li $t1, '0'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, '0'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, '3'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[6]

addi $a0, $a0, 4


# Υπολογισμός ('M' << 24) | ('I' << 16) | ('P' << 8) | 'S'

li $t0, 'M'

sll $t0, $t0, 24

li $t1, 'I'

sll $t1, $t1, 16

```

or \$t0, \$t0, \$t1

li \$t1, 'P'

sll \$t1, \$t1, 8

or \$t0, \$t0, \$t1

li \$t1, 'S'

or \$t0, \$t0, \$t1

sw \$t0, (\$a0) # Αποθήκευση στο data[7]

addi \$a0, \$a0, 4

jr \$ra

---

#### 4) ΕΙΚΟΝΕΣ

Εικόνα Κώδικα σε C:



```

main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  unsigned int cslab_hash(unsigned int input){
5      unsigned int hash = 5381;
6      int c;
7      while (input != 0) {
8          c = (input & 0xFF);
9          //bitwise and bit by bit comparison
10         hash = ((hash << 4) + hash) + c;
11         input = input >> 8;
12     }
13     return hash;
14 }
15
16 void create_leaves(unsigned int* data_array, int array_size, int* tree){
17     int i = 0;
18     while(i < array_size){
19         *tree = cslab_hash(*data_array);
20         data_array++;
21         tree++;
22         i++;
23     }
24     return;
25 }
26
27 unsigned int create_Merkle_Tree(unsigned int* tree, int num_of_leaves){
28     int level_ops = num_of_leaves / 2;
29     unsigned int * tail = &tree[num_of_leaves];
30     int i = 0;
31     while (level_ops > 0){
32         int j = 0;
33         while (j < level_ops){
34             unsigned int xored_val = tree[i] ^ tree[i+1];
35             *tail = cslab_hash(xored_val);
36             i += 2;
37             j++;
38             tail++;
39         }
40         level_ops /= 2;
41     }
42     return *(tail-1);
43 }
44
45 int main(){
46     unsigned int *data ;
47     data = malloc(8 * sizeof(int));
48
49     //MIPS is big-endian = most significant character in MSB
50     data[0] = ('4' << 16) | ('2' << 8) | '9'; //3 last digits of AM (e.g. 193)
51     data[1] = ('N' << 24) | ('T' << 16) | ('U' << 8) | 'A';
52     data[2] = ('E' << 16) | ('C' << 8) | 'E' ;
53     data[3] = ('C' << 8) | 'A' ;
54     data[4] = ('2' << 24) | ('0' << 16) | ('2' << 8) | '4';
55     data[5] = ('0' << 24) | ('6' << 16) | ('1' << 8) | '0'; //DDMM of date of birth
56     data[6] = ('2' << 24) | ('0' << 16) | ('0' << 8) | '3'; //YYYY of date of birth
57     data[7] = ('M' << 24) | ('I' << 16) | ('P' << 8) | 'S';
58     unsigned int* tree = malloc(15 * sizeof(unsigned int));
59     create_leaves(data, 8, tree);
60     printf("Root hash: %d\n", create_Merkle_Tree(tree, 8));
61     return 0;
62 }
63
64

```

Input

Root hash: 450509173

...Program finished with exit code 0  
Press ENTER to exit console.

Εικόνα κώδικα σε ASSEMBLY MIPS (η προσομοίωση έγινε σε MARS 4.5):

C:\Users\User\Documents\electrical\_engineering\5th semester\Apptextovikr\lets\_play - MARS 4.5

File Edit Run Settings Tools Help

Run speed 30 inst/sec

**Text Segment**

Bkpt	Address	Code	Basic
<input type="checkbox"/>	0x00400064	0x8fbf0000	lw \$31,0(\$29)
<input type="checkbox"/>	0x00400068	0x23bd0004	addi \$29,\$29,4
<input type="checkbox"/>	0x0040006c	0x20480000	addi \$8,\$2,0
<input type="checkbox"/>	0x00400070	0x3c011001	lui \$1,4097
<input type="checkbox"/>	0x00400074	0x34240060	ori \$4,\$1,96
<input type="checkbox"/>	0x00400078	0x24020004	addiu \$2,\$0,4
<input type="checkbox"/>	0x0040007c	0x0000000c	syscall
<input type="checkbox"/>	0x00400080	0x21040000	addi \$4,\$8,0
<input type="checkbox"/>	0x00400084	0x24020001	addiu \$2,\$0,1
<input type="checkbox"/>	0x00400088	0x0000000c	syscall
<input type="checkbox"/>	0x0040008c	0x2402000a	addiu \$2,\$0,10
<input type="checkbox"/>	0x00400090	0x0000000c	syscall

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	3420729	1314149697	4539205	17217
0x10010020	8	26454228	449771917	26458002
0x10010040	449858718	449888775	450271181	1557396
0x10010060	1869566496	1634214004	540698739	0
0x10010080	0	0	0	0
0x100100a0	0	0	0	0
0x100100c0	0	0	0	0
0x100100e0	0	0	0	0
0x10010100	0	0	0	0
0x10010120	0	0	0	0

**Registers** Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	450509173
\$a1	5	8
\$a2	6	268501060
\$a3	7	0
\$t0	8	450509173
\$t1	9	0
\$t2	10	450509146
\$t3	11	268501080
\$t4	12	32
\$t5	13	0
\$t6	14	26496682
\$t7	15	450059633
\$a0	16	0
\$a1	17	0
\$a2	18	0
\$a3	19	14
\$a4	20	1
\$a5	21	0
\$a6	22	268501088
\$a7	23	0
\$t8	24	457642971
\$t9	25	48
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479546
\$fp	30	0
\$ra	31	0
pc		4194452
hi		0
lo		0

**Mars Messages** Run I/O

```
Root hash: 450509173
-- program is finished running --
```

Clear

**Mars Messages** Run I/O

```
Root hash: 450509173
-- program is finished running --
```

Clear

## 5. ΠΑΡΑΡΤΗΜΑ 1 Ο ΣΥΝΟΛΙΚΟΣ ΚΩΔΙΚΑΣ ASSEMBLY

```
.data

data: .space 32 #Χώρος για 8 κόμβους

no_of_leaves: .word 8 #Αποθηκεύω τον αριθμό των φύλλων

tree: .space 60 #Χώρος για 15 κόμβους (8 φύλλα + 4 ενδιάμεσοι 1ου επιπέδου + 2 ενδιάμεσοι 2ου επιπέδου + root) *4
wordlength

message: .asciiz " Root hash: "

.text

.globl MAIN

MAIN:

    #Κλήση της CREATE_DATA

    la $a0, data    # Διεύθυνση του πίνακα data

    addi $sp, $sp, -4

    sw $ra, 0($sp)

    jal CREATE_DATA

    lw $ra, 0($sp)

    addi $sp, $sp, 4

    # Κλήση CREATE_LEAVES

    la $a0, data    # Διεύθυνση του πίνακα data

    lw $a1, no_of_leaves# Μέγεθος του πίνακα data

    la $a2, tree    # Διεύθυνση του πίνακα tree (φύλλα ξεκινούν από εδώ)

    addi $sp, $sp, -4

    sw $ra, 0($sp)

    jal CREATE_LEAVES    # Κλήση της void CREATE_LEAVES

    lw $ra, 0($sp)

    addi $sp, $sp, 4
```

```

# Κλήση CREATE_MERKLE_TREE

la $a0, tree    # Διεύθυνση του πίνακα tree (φύλλα ξεκινούν από εδώ)

lw $a1, no_of_leaves# Μέγεθος του πίνακα data


addi $sp, $sp, -4

sw $ra, 0($sp)

jal CREATE_MERKLE_TREE    # Κλήση της int CREATE_LEAVES ($v0 <- ROOT)

lw $ra, 0($sp)

addi $sp, $sp, 4


#Εκτύπωση μηνύματος

addi $t0, $v0, 0


la $a0, message    # Load the address of the string into $a0

li $v0, 4    # Syscall code for printing a string

syscall    # Perform the syscall


la $a0, ($t0)    # Load the address of the ROOT saved in $t0

li $v0, 1    # Syscall code for printing a string

syscall    # Perform the syscall


# Τερματισμός

li $v0, 10    # Σύστημα κλήσης: τερματισμός

syscall

```

#### CREATE\_DATA:

```

# Υπολογισμός ('4' << 16) | ('2' << 8) | '9'

li $t0, '4'    # Φόρτωσε τον χαρακτήρα '4' γίνεται και με addi $t0, $zero, '4'

sll $t0, $t0, 16    # Μετακίνησε 16 bits αριστερά

li $t1, '2'    # Φόρτωσε τον χαρακτήρα '2'

sll $t1, $t1, 8    # Μετακίνησε 8 bits αριστερά

or $t0, $t0, $t1    # Συνδυασμός '4' και '2'

```

```

li $t1, '9'      # Φόρτωσε τον χαρακτήρα '9'

or $t0, $t0, $t1  # Συνδυασμός με '9'


sw $t0, ($a0)    # Αποθήκευση στο data[0]

addi $a0, $a0, 4


# Υπολογισμός ('N' << 24) | ('T' << 16) | ('U' << 8) | 'A'

li $t0, 'N'

sll $t0, $t0, 24

li $t1, 'T'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, 'U'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, 'A'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[1]

addi $a0, $a0, 4


# Υπολογισμός ('E' << 16) | ('C' << 8) | 'E'

li $t0, 'E'

sll $t0, $t0, 16

li $t1, 'C'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, 'E'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[2]

addi $a0, $a0, 4

```

```

# Υπολογισμός ('C' << 8) | 'A'

li $t0, 'C'

sll $t0, $t0, 8

li $t1, 'A'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[3]

addi $a0, $a0, 4


# Υπολογισμός ('2' << 24) | ('0' << 16) | ('2' << 8) | '4'

li $t0, '2'

sll $t0, $t0, 24

li $t1, '0'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, '2'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, '4'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[4]

addi $a0, $a0, 4


# Υπολογισμός ('0' << 24) | ('6' << 16) | ('1' << 8) | '0'

li $t0, '0'

sll $t0, $t0, 24

li $t1, '6'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, '1'

```

```

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, '0'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[5]

addi $a0, $a0, 4


# Υπολογισμός ('2' << 24) | ('0' << 16) | ('0' << 8) | '3'

li $t0, '2'

sll $t0, $t0, 24

li $t1, '0'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, '0'

sll $t1, $t1, 8

or $t0, $t0, $t1

li $t1, '3'

or $t0, $t0, $t1


sw $t0, ($a0)    # Αποθήκευση στο data[6]

addi $a0, $a0, 4


# Υπολογισμός ('M' << 24) | ('I' << 16) | ('P' << 8) | 'S'

li $t0, 'M'

sll $t0, $t0, 24

li $t1, 'I'

sll $t1, $t1, 16

or $t0, $t0, $t1

li $t1, 'P'

sll $t1, $t1, 8

or $t0, $t0, $t1

```

```
li $t1, 'S'
```

```
or $t0, $t0, $t1
```

```
sw $t0, ($a0) # Αποθήκευση στο data[7]
```

```
addi $a0, $a0, 4
```

```
jr $ra
```

CSLAB\_HASH:

```
addi $t0, $zero, 5381 #t0<-hash=5381
```

```
# $t1<-c
```

WHILE\_HASH:

```
beq $a0, $zero, END_HASH # while (input != 0)
```

```
andi $t1, $a0, 0xFF #t1<-c=(input & 0xFF)
```

```
sll $t2, $t0, 4 #t2<-(hash<<4)
```

```
add $t2, $t2, $t1 #t2<-((hash << 4) + hash!!
```

```
add $t0, $t2, $t1 #t0<-hash=((hash << 4) + hash) + c;
```

```
srl $a0, $a0, 8 # input = input >> 8;
```

```
j WHILE_HASH
```

END\_HASH:

```
add $v0, $t0, $zero # return hash;
```

```
jr $ra # Return from function
```

CREATE\_LEAVES:

```
addi $sp, $sp, -8 #κάνω χώρο (για $ra) το $sp γίνεται $sp'
```

```
sw $ra, 4($sp) #saves $ra at position 4($sp')
```

```
sw $s0, 0($sp) #saves $s0 at position 0($sp')
```

```
addi $s0, $zero, 0 #s0<-i μετρητής
```

WHILE\_LEAVES:

```
slt $t0, $s0, $a1 # (i < array_size)
```

```
beq $t0, $zero, END_LEAVES # while(i < array_size)
```



```
addi $sp,$sp,-4
```

```
sw $a0, 0($sp)
```

```
#Κλήση CSLAB_HASH
```

```
lw $a0, 0($a0)
```

```
jal CSLAB_HASH
```

```
lw $a0, 0($sp)
```

```
addi $sp,$sp,4
```

```
#πλέον στο $v0 <-hash
```

```
sw $v0, 0($a2) # *tree = cslab_hash(*data_array);
```

```
addi $a0, $a0, 4 # data_array++
```

```
addi $a2, $a2, 4 # tree++
```

```
addi $s0, $s0, 1 # i++
```

```
j WHILE_LEAVES
```

```
END_LEAVES:
```

```
lw $ra, 4($sp) #loads $ra from position 4($sp')
```

```
lw $s0, 0($sp) #loads $s0 from position 0($sp')
```

```
addi $sp, $sp, 8 #ελευθερώνω χώρο (για $ra) το $sp' γίνεται $sp
```

```
jr $ra # Return from function
```

```
CREATE_MERKLE_TREE:
```

```
addi $sp, $sp, -20 #κάνω χώρο (για $ra) το $sp γίνεται $sp'
```

```
sw $ra, 16($sp) #saves $ra at position 16($sp')
```

```
sw $s0, 12($sp) #saves $s0 at position 12($sp')
```

```
sw $s1, 8($sp) #saves $s1 at position 8($sp')
```

```
sw $s2, 4($sp) #saves $s2 at position 4($sp')
```

```
sw $s3, 0($sp) #saves $s3 at position 0($sp')
```

```
srl $s2, $a1, 1 #num_of_leaves / 2
```

```
sll $t4, $a1, 2 # num_of_leaves *4 για να δείχνει διεύθυνση στοιχείου του πίνακα αφού wordlength 4
```

```
add $s3, $a0, $t4# διεύθυνση του tree[num_of_leaves]
```

```

addi $s0, $zero, 0 # αρχικοποίηση i=0

WHILE_I:

    slt $t1, $zero, $s2 #(level_ops > 0)

    beq $t1, $zero, END_TREE

    addi $s1, $zero, 0 #int j = 0;

    j WHILE_J

DONT_FORGET:

    srl $s2, $s2, 1 #level_ops /= 2;

    j WHILE_I

WHILE_J:

    slt $t0, $s1, $s2

    beq $t0, 0, DONT_FORGET


    sll $t9, $s0, 2 #i*4

    add $t2, $a0, $t9    # *tree[i]

    addi $t3, $t2, 4     # *tree[i+1]

    lw $t6, ($t2) # tree[i]

    lw $t7, ($t3) # tree[i+1]

    xor $t8, $t6, $t7    # xored_val = tree[i] ^ tree[i+1]


    addi $sp, $sp, -8    # Κάνε χώρο στη στοίβα

    sw $ra, 4($sp)      # Αποθήκευσε $ra

    sw $a0, 0($sp)      # Αποθήκευσε $a0

    addi $a0, $t8, 0    # Δώσε το xored_val ως είσοδο

    jal CSLAB_HASH      # Κάλεσε τη συνάρτηση hash


    sw $v0, 0($s3)      # Αποθήκευσε το hash στο tail


    addi $s3, $s3, 4    # tail++


    lw $ra, 4($sp)      # Επαναφορά $ra

    lw $a0, 0($sp)      # Επαναφορά $a0

```

```

addi $sp, $sp, 8    # Επαναφορά στοίβας
addi $s0, $s0, 2    # i += 2
addi $s1, $s1, 1    # j++
j WHILE_J

```

END\_TREE:

```

add $v0, $s3, -4    # Επιστροφή *(tail - 1)
lw $v0, ($v0)

lw $ra, 16($sp) #loads $ra from position 16($sp')
lw $s0, 12($sp) #loads $s0 from position 12($sp')
lw $s1, 8($sp) #loads $s1 from position 8($sp')
lw $s2, 4($sp) #loads $s2 from position 4($sp')
lw $s3, 0($sp) #loads $s3 from position 0($sp')

addi $sp, $sp, 20 #απελευθερώνω χώρο το $sp' γίνεται $sp

jr $ra            # Επιστροφή

```

## 6. ΠΑΡΑΡΤΗΜΑ 2 ΔΟΚΙΜΕΣ

Για δοκιμές χρησιμοποιήθηκε η παρακάτω main:

```
#include <stdlib.h>
```

```
//... οι συναρτήσεις της εκφώνησης...//
```

```
int main(){
```

```
    unsigned int * data;
```

```
    data = malloc(8*sizeof(int));
```

```
    data[0] = (0x12345678);
```

```
    data[1] = (0x87654321);
```

```
    data[2] = (0x11111111);
```

```
    data[3] = (0x22222222);
```

```
    unsigned int* tree = malloc(7 * sizeof(unsigned int)); //αντι για 15 βαζω 7 αφού το δέντρο θα έχει 4 φύλλα θα εχει 2  
    κόμβους και μια ρίζα
```

```
    create_leaves(data, 4, tree);
```

```
    printf(" hash of 0x12345678: %d\n", cslab_hash(0x12345678));
```

```
    printf(" hash of 0x87654321: %d\n", cslab_hash(0x87654321));
```

```
    printf(" hash of 0x11111111: %d\n", cslab_hash(0x11111111));
```

```
    printf(" hash of 0x22222222: %d\n", cslab_hash(0x22222222));
```

```
    printf("Root hash: %d\n", create_Merkle_Tree(tree, 4));
```

```
    // Εκτύπωση όλων των στοιχείων του tree
```

```
    printf("\nElements of the tree:\n");
```

```
    for (int i = 0; i < 7; i++) {
```

```
        printf("tree[%d] = %d\n", i, tree[i]);
```

```
    }
```

```
free(data);  
  
free(tree);  
  
return 0;  
}
```

Που εκτύπωσε τα παρακάτω αποτελέσματα τα οποία σχεδιάστηκαν και σε δέντρο:

```

graph TD
  A["ROOT  
tree[6]  
26510249"] --> B
  A --> C
  B["1ο επίπεδο κόμβων:  
tree[4]  
26489159"] --> Diagram3["Φύλλα:  
tree[0]  
450041817"]
  B --> Diagram4["Φύλλα:  
tree[1]  
449609845"]
  C["1ο επίπεδο κόμβων:  
tree[5]  
26467511"] --> Diagram5["Φύλλα:  
tree[2]  
449515241"]
  C --> Diagram6["Φύλλα:  
tree[3]  
449603981"]

```

Elements of the tree:

tree[0] = 450041817

tree[1] = 449609845

tree[2] = 449515241

tree[3] = 449603981

tree[4] = 26489159

tree[5] = 26467511

tree[6] = 26510249

Root hash: 26510249

