# Ψηφιακοί Υπολογιστές - Αναφορά Εργαστηρίου 3

**Ομάδα LAB20138010**

- **Χρήστος Μιχαλόπουλος (2016030088)**
- **Αντώνης Ανδρεαδάκης (2013030059)**

## Προεργασία

Για την τρίτη εργαστηριακή άσκηση, σκοπός ήταν η εμβάθυνση στην κατανόηση της λειτουργίας της μνήμης και στη γλώσσα CLang. Επίσης, γνωριμία με την Assembly.

## Ζητούμενα και Εκτέλεση της άσκησης

Γράψαμε ένα πρόγραμμα σε CLang, το οποίο εκτελείται επαναληπτικά δίνοντας 4 επιλογές στον χρήστη. Η πρώτη επιλογή, αφορά την εκτύπωση ενός τριγώνου/πυραμίδας. Δηλαδή ο χρήστης δίνει έναν ακέραιο N και ξεκινώντας απ' το 1, εκτυπώνονται σε N γραμμές όλοι οι αριθμοί από το 1 μέχρι το N που έδωσε ο χρήστης. Η δεύτερη επιλογή, έχει να κάνει με την εκτύπωση ενός μηνύματος για το αν ο αριθμός που θα εισάγει ο χρήστης είναι άρτιος ή περιττός. Η τρίτη και ίσως πιο δύσκολη σε υλοποίηση, είναι η ανάγνωση 5 ακεραίων και εκτύπωση του πενταπλάσιού τους. Δηλαδή ο χρήστης δίνει επαναληπτικά 5 αριθμούς και μετά εμφανίζεται ένα μήνυμα, με τους αριθμούς πολλαπλασιασμένους κατά 5 ο καθένας τους. Τελευταία επιλογή, είναι η έξοδος από το πρόγραμμα. Συγκεκριμένα:

```
int main(void)
{
                int i;

main_while_1:   printf("\nWHAT DO YOU WANT TO DO? \n");
                printf("1. PRINT THE PYRAMID. \n");
                printf("2. CHECK IF A NUMBER IS EVEN OR ODD. \n");
                printf("3. MULTIPLY NUMBERS BY FIVE. \n");
                printf("4. EXIT.\n");
                printf("\nPLEASE ENTER YOUR CHOICE: ");
                scanf("%lld", &R30);

                R9 = 1;
                if (R30 != R9)  goto after_if_1;
                printPyramid(R2);
                goto main_after_if;
after_if_1:     R9=2;
                if (R30 != R9) goto after_if_2;
                checkNumber(R2);
                goto main_after_if;
after_if_2:     R9=3;
                if (R30 != R9) goto after_if_3;
                printf("Give me 5 numbers you want to multiply by 4. \n");
                printf("\n --- INPUT --- \n");

                i = (int)R10;
                R10 = 0;

if_label_2_1:   if(R10 >= 5) goto after_if_2_1;
                printf("Enter the number: ");
                scanf("%d", &A[R10]);
                R10 = R10 + 1;
                goto if_label_2_1;

after_if_2_1:
                R8 = (long long)A;
                R9 = (long long)B;

                multiplyNum(R8,R9);

                printf("\n --- OUTPUT --- \n");
                R10 = 0;
if_label_2_2:   if (R10 >= 5) goto main_after_if;
                printf("The result is: %d \n", B[R10]);
                R10 = R10 +1;
                goto if_label_2_2;



after_if_4:     R9=4;
                if (R30 != R9) goto main_after_if;
                return 0;
main_after_if:  R9=4;
                if(R9 != R30) goto main_while_1;
}
```

Υλοποιήθηκαν 3 συναρτήσεις για το κάθε ερώτημα και 1 main για την εκτέλεση του βασικού menu.

Main: επειδή θέλουμε να εκτελείται επαναληπτικά το πρόγραμμά μας, έως ότου ο χρήστης πατήσει έξοδο, χρησιμοποιήσαμε ένα while label (main_while_1 το ονομάσαμε). Μέσα σε αυτό λοιπόν, τοποθετήσαμε τα κατάλληλα μηνύματα που θέλουμε να "βλέπει" ο χρήστης πριν από κάθε επιλογή του, καθώς και τις κλήσεις των 3 συναρτήσεων που θα πρέπει να εκτελούνται, ανάλογα με την επιλογή του χρήστη. Για τις 2 πρώτες επιλογές, ζητείται ο αριθμός N και γίνεται ανάγνωση του αριθμού αυτού. Αντιθέτως, στην τρίτη επιλογή ζητείται από το χρήστη να δώσει 5 διαδοχικούς αριθμούς, οι οποίοι αποθηκεύονται σε μία αλληλουχία θέσεων μνήμης (πίνακας).

**1η συνάρτηση:** δέχεται ως όρισμα την τιμή N και μέσω κατάλληλων βρόγχων επανάληψης τυπώνεται ένα τρίγωνο που έχει μέγεθος όσο και ο αριθμός N (σε γραμμές) και ξεκινώντας απ' το 1 στην πρώτη γραμμή, φτάνουμε έως το N στην N-οστή γραμμή.
Συγκεκριμένα η πρώτη for (for_label_1), κάνει επανάληψη για τις γραμμές του τριγώνου, τόσες φορές όσο είναι ο N που εισήγαγε ο χρήστης. Στην δεύτερη for (for_loop_2) τυπόνονται επαναληπτικά οι αριθμοί στον πίνακα (οι στήλες) μέχρι να φτάσουν τις N στήλες στις αντίστοιχες N σειρές. Επιπλέον, μετά την εκτύπωση του τριγώνου, εκτυπώνεται και το συνολικό άθροισμα των στοιχείων του χρησιμοποιώντας τον τύπο (N*(N+1)/2).

```
void printPyramid(long long R4)
{
    long long R8=1, R9=1;
    long long R11=0;

do_while_label:     printf("\nEnter the number of the rows you want: \n");
                    scanf("%lld", &R4);
                    if(R4>R11) goto after_label_1;
                    goto do_while_label;
after_label_1:

for_label_1:        if(R8>R4) goto after_for_1;
                    R9=1;
for_label_2:        if(R9>R8) goto after_for_2;
                    printf("%4lld", R9);
                    R9 = R9+1;
                    goto for_label_2;
after_for_2:        printf("\n");
                    R8 = R8+1;
                    goto for_label_1;
after_for_1:        printf("\n");
                    R5 = (R4*(R4+1))/2;
                    printf("The result of all elements added is: %lld \n", R5);

                    return;
}
```

**2η συνάρτηση:** επίσης δέχεται ως όρισμα την τιμή N, ομοίως μέσω κατάλληλων βρόγχων επανάληψης γίνεται η αναγνώριση του αριθμού αν είναι άρτιος ή περιττός. Απ' τα ζητούμενα της άσκησης, περιοριστήκαμε και δεν έγινε χρήση της εντολής % (modulus). Μέσω μιας επανάληψης (if_label_2), αφαιρώντας το 2 κάθε φορά ο αριθμός φθίνει. Αν το τελικό αποτέλεσμα γίνει 0 θα εκτυπωθεί μήνυμα ότι είναι άρτιος, αλλιώς αν γίνει 1 θα εκτυπωθεί ότι είναι περιττός.

```
void checkNumber (long long R4)
{
    long long R11 = 0, R8;

                printf("\nGive the number you want to check: ");
                scanf("%lld", &R4);

if_label_1:     if(R4>=R11) goto after_if_1;
                R4 = R4*(-1);
after_if_1:     R8=R4;
if_label_2:     if (R8<2) goto if_label_3;
                R8 = R8-2;
                goto if_label_2;
if_label_3:     if(R8!=0) goto else_label;
                printf("\nThe number is even. \n");
                goto after_cond;
else_label:     printf("\nThe number is odd. \n");
after_cond:
                return;
}
```

**3η συνάρτηση:** χρησιμοποιεί 2 πίνακες (A και B) που δέχεται ο καθένας 5 ακέραιους. Κι εδώ, υπήρξε περιορισμός απ' τα δεδομένα της άσκησης, να μην χρησιμοποιηθεί η πράξη του πολλαπλασιασμού. Οπότε, με τη χρήση προσθέσεων μέσα σε Loop καταφέραμε να υλοποιήσουμε τη συγκεκριμένη διαδικασία.
Αρχικά, ζητώνται στην main οι τιμές των 5 αριθμών από τον χρήστη και με μια επαναληπτική διαδικασία αποθηκεύονται στον πίνακα A. Στην συνέχεια, και αφου γίνει κλήση της συνάρτησης multiplyNum μέσω της επαναληπτικής διαδικασίας στο if_label_2 προστίθεται 5 φορές ο ίδιος αριθμός και το τελικό αποτέλεσμα αποθηκεύεται στο εκάστοτε στοιχείο στον πίνακα B. Η συνάρτηση επιστρέφει στην main τον πίνακα B. Στην main με μια επαναληπτική διαδικασία εκτυπώνεται το τελικό αποτέλεσμα για κάθε έναν από τους 5 αριθμούς.

```
long long multiplyNum(long long *R6, long long *R7)
{
    int i=0, j=0, input=0 , output=0;
    int *A,*B;

    R10 = (long long)i;
    R11 = (long long)j;
    R12 = (long long )input;
    R13 = (long long )output;
    A  = (long long *)R6;
    B  = (long long *)R7;


    R12 = 0;
if_label_1: if(R10 >= 5) goto end_if_1;
    R13 = 0;
    R12 = A[R10];

if_label_2: if(R11 >= 4) goto end_if_2;
    R13 = R13 + R12;
    R11 = R11 + 1;
    goto if_label_2;
end_if_2:
    B[R10] = (int)R13;
    R10 = R10 +1;
    R11 = 0;
    goto if_label_1;
end_if_1:
    R2 = (long long)B;
    return R2;

}
```

Τέλος, για πρώτη φορά υλοποιήσαμε ένα μέρος του προγράμματός μας σε assembly. Αυτό αφορούσε την main και συγκεκριμένα μόνο η δυνατότητα επιλογής του προγράμματος από το menu. Στη συνέχεια, τυπώνεται η επιλογή που κάνει ο χρήστης και το πρόγραμμα τρέχει επαναληπτικά μέχρι να επιλεγεί η έξοδος. Έγινε χρήση των εντολών **li (load immediate)**, **la (load address)**, **move** καθώς και της **syscall**, η οποία εκτελεί τις εντολές που δώθηκαν. Στην εντολή li ο αριθμός 4, εκτυπώνει την αντίστοιχη συμβολοσειρά, ο αριθμός 1 χρησιμοποιείται για την εκτύπωση του κάθε ακεραίου και ο αριθμός 5 διαβάζει τον ακέραιο που θα εισάγει ο χρήστης.

```
.data
message1: .asciiz "\nWhat do you want to do? \n"
message2: .asciiz "1. Print the pyramid. \n"
message3: .asciiz "2. Check if number is even or odd. \n"
message4: .asciiz "3. Multiply numbers by five. \n"
message5: .asciiz "4. Exit. \n"
message6: .asciiz "Please enter your choice: "
message7: .asciiz "Your choice is "
invalid:  .asciiz "INVALID \n"
terminate: .asciiz "Programm finished. \n"

.text
#print message1

      li $v0, 4
      la $a0, message1
      syscall

#print message2


      li $v0, 4
      la $a0, message2
      syscall

#print message3


      li $v0, 4
      la $a0, message3
      syscall

#print message4


      li $v0, 4
      la $a0, message4
      syscall

#print message5


      li $v0, 4
      la $a0, message5
      syscall
```

```
#print message6

li $v0, 4
la $a0, message6
syscall

#get users option
li $v0, 5
syscall

#store result
move $t0, $v0

#display result

li $v0, 4
la $a0, message7
syscall

#print option
#li $v0, 1
#move $a0, $t0
#syscall

#option = 1
while1:
      bne $t0,1,while2
      li $v0,1
      move $a0, $t0
      syscall
      j while0

#option = 2
while2:
      bne $t0,2,while3
      li $v0, 1
      move $a0, $t0
      syscall
      j while0

#option = 3
while3:
      bne $t0,3,while4
      li $v0, 1
      move $a0, $t0
      syscall
      j while0
```

```
#option = 4
while4:
      bne $t0,4,while5
      li $v0, 1
      move $a0, $t0
      syscall

      li $v0, 10
      syscall



#otherwise
while5:
      li $v0,4
      move $a0, $t0
      la $a0, invalid
      syscall
      j while0
```

## Συμπεράσματα

Ολοκληρώνοντας την τρίτη εργαστηριακή άσκηση κατανοήσαμε καλύτερα την γλώσσα προγραμματισμού CLANG, αφού υλοποιήσαμε πολλές συναρτήσεις σε αυτή την γλώσσα. Επιπλέον κατανοήσαμε πως συνδέεται η CLANG με την Assembly, αφού χρειάστηκε να υλοποιήσουμε σε αυτήν το menu επιλογών.

# Ψηφιακοί Υπολογιστές - Αναφορά Εργαστηρίου 4

**Ομάδα LAB20138010**

- **Χρήστος Μιχαλόπουλος (2016030088)**
- **Αντώνης Ανδρεαδάκης (2013030059)**

## Προεργασία

Για την τέταρτη εργαστηριακή άσκηση, σκοπός ήταν η εξοικείωση με τη χρήση της γλώσσας Assembly. Μας χρειάστηκε η CLang για να κάνουμε τη μετατροπή σε Assembly και οι κώδικες από το 3ο εργαστήριο.

## Ζητούμενα και Εκτέλεση της άσκησης

Γράψαμε ένα πρόγραμμα σε Assembly, το οποίο εκτελείται επαναληπτικά δίνοντας 5 επιλογές στον χρήστη. Στην ουσία επαναλάβαμε το προηγούμενο εργαστήριο, με την προσθήκη 1 επιπλέον συνάρτησης/επιλογής. Διαφοροποίηση υπήρχε στην τρίτη, όπου γίνεται η ανάγνωση 5 ακεραίων και εκτύπωση του τετραπλασιού τους. Δηλαδή ο χρήστης δίνει επαναληπτικά 5 αριθμούς και μετά εμφανίζεται ένα μήνυμα, με τους αριθμούς πολλαπλασιασμένους κατά 4 ο καθένας τους. Στην τέταρτη επιλογή, ο χρήστης δίνει ως είσοδο μια συμβολοσειρά το πολύ 100 χαρακτήρων και πραγματοποιείται εναλλαγή των πεζών και κεφαλαίων γραμμάτων. Τελευταία επιλογή, είναι η έξοδος από το πρόγραμμα. Υλοποιήθηκαν 4 συναρτήσεις για το κάθε ερώτημα και 1 main για την εκτέλεση του βασικού menu. Στο πεδίο data λοιπόν:

```
.data
#data for main
menu:   .asciiz "\n What do you want to do? \n"
menu1:  .asciiz "1. Print pyramid. \n"
menu2:  .asciiz "2. Check if a number is even or odd. \n"
menu3:  .asciiz "3. Multiply numbers by four. \n"
menu4:  .asciiz "4. Change uppers and lowers letters. \n"
choice: .asciiz "Please enter your choice: "
space1: .asciiz "\n"
error:  .asciiz " \n The number you have entered is invalid. Try again. \n"

#data for function 1
numOfRows:     .asciiz "Enter the number of rows you want: "
positiveCheck: .asciiz "Please enter only positive numbers. \n"
totalSum:      .asciiz "The result of all elements added is: \n"
space:         .asciiz "\n"
tab:           .asciiz "\t"

#data for function 2
enterNumber: .asciiz "Give the number you want to check \n"
even:        .asciiz "The number is even \n"
odd:         .asciiz "The number is odd \n"

#data for function 3
arrayA: .space 20
.align 2
arrayB: .space 20
.align 2

giveNums: .asciiz "Give me the 5 numbers you want to multiply by 5. \n"
enterNum: .asciiz "Enter the number: "
input:    .asciiz " --- INPUT --- \n"
output:   .asciiz " --- OUTPUT --- \n"
result:   .asciiz "\n --- The result is: "
newLine:  .asciiz "\n"

#data for function 4
array_a: .space 100
.align 1
array_b: .space 100
.align 1
enterStr:    .asciiz "Please enter the string you want: "
modifiedStr: .asciiz "The modified string is: "
```

***Main:*** επειδή θέλουμε να εκτελείται επαναληπτικά το πρόγραμμά μας, έως ότου ο χρήστης πατήσει έξοδο, χρησιμοποιήσαμε ένα while label ( main_while_1 το ονομάσαμε). Μέσα σε αυτό λοιπόν, τοποθετήσαμε τα κατάλληλα μηνύματα που θέλουμε να "βλέπει" ο χρήστης πριν από κάθε επιλογή του, καθώς και τις κλήσεις των 4 συναρτήσεων που θα πρέπει να εκτελούνται, ανάλογα με την επιλογή του χρήστη. Για τις 2 πρώτες επιλογές, ζητείται ο αριθμός N και γίνεται ανάγνωση του αριθμού αυτού και αποθηκεύεται στον καταχωρητή t0. Αντιθέτως, στην τρίτη επιλογή ζητείται από το χρήστη να δώσει 5 διαδοχικούς αριθμούς, οι οποίοι αποθηκεύονται σε μία αλληλουχία θέσεων μνήμης, δηλαδή σε πίνακες και ο οποίοι αποθηκεύονται στους καταχωρητές s1 και s2 αντίστοιχα. Τέλος, για την τέταρτη επιλογή ζητείται από το χρήστη μια συμβολοσειρά, η

οποία αποθηκεύεται σε συγκεκριμένο χώρο μνήμης, στους καταχωρητές s4 και s5 αντίστοιχα. Να σημειωθεί οτι δεν επιτρεπόταν οι χρήση ψευδοεντολών.

1

```
.text
#MAIN
main:
main_while_1:

        #print Menu
addi $v0, $zero, 4
la $a0, menu
syscall

addi $v0, $zero, 4
la $a0, menu1
syscall

addi $v0, $zero, 4
la $a0, menu2
syscall

addi $v0, $zero, 4
la $a0, menu3
syscall

addi $v0, $zero, 4
la $a0, menu4
syscall

addi $v0, $zero, 4
la $a0, choice
syscall

addi $v0, $zero, 5
syscall
add $t0, $zero, $v0             #store choice to $t0
after_if_1a:
addi $t1, $zero, 1
bne $t0, $t1, after_if_2        #if choice=1, run this part

addi $v0, $zero, 4
la $a0, space1
syscall

jal printPyramid

j main_while_1
```

2

```
after_if_2:
addi $t1, $zero, 2
bne $t0, $t1, after_if_3        #if choice=2, run this part
addi $v0, $zero, 4
la $a0, space1
syscall
jal checkNumber

j main_while_1

after_if_3:
addi $t1, $zero, 3
bne $t0, $t1, after_if_4        #if choice=3, run this part
addi $v0, $zero, 4
la $a0, giveNums
syscall
addi $v0, $zero, 4
la $a0, input
syscall
la $s1, arrayA                  #store array A[0] to s1
la $s2, arrayB                  #store array B[0] to s2

add $t7, $zero, $s1             #temporary store s1 to t7
addi $t8, $zero, 0
addi $t9, $zero, 5
if_label_2_1:                           #while loop to enter 5 numbers and to store them in arrayA
sge $t5, $t8, $t9
bne $t5, $zero, after_if_2_1

addi $v0, $zero, 4
la $a0, enterNum
syscall
addi $v0, $zero, 5
syscall
add $t3, $zero, $v0
sw $t3, 0($t7)
addi $t7, $t7, 4
addi $t8, $t8, 1
j if_label_2_1

after_if_2_1:
la $s2, arrayB

add $t7, $zero, $s1

jal multiplyNum
```

3

```
add $t7, $zero, $s2

addi $v0, $zero, 4
la $a0, output
syscall

addi $t0, $zero, 0
addi $t1, $zero, 5
if_label_2_2:

sge $t5, $t0, $t1
bne $t5, $zero, main_while_1

addi $v0, $zero, 4
la $a0, result
syscall
lw $a0, 0($s2)
addi $v0, $zero, 1
syscall

addi $s2, $s2, 4
addi $t0, $t0, 1

j if_label_2_2

after_if_4:
addi $t1, $zero, 4
bne $t0, $t1, after_if_6        #if choice=4, run this part

la $s4, array_a
la $s5, array_b

addi $v0, $zero, 4
la $a0, enterStr
syscall

addi $v0, $zero, 8
la $a0, array_a
addi $a1, $zero, 100
syscall

add $a0, $zero, $s4
add $a1, $zero, $s5

jal modifyString


addi $v0, $zero, 4
la $a0, modifiedStr
syscall
```

4

```
addi $v0, $zero, 4
la $a0, array_b
syscall

j main_while_1

after_if_5:
addi $t1, $zero, 4
bne $t0, $t1, after_if_6
li $v0, 10
syscall

after_if_6:
addi $t1, $zero, 4
li $v0, 4
la $a0, error
syscall
bgt $t0, $t1, main_while_1
```

***1η συνάρτηση:*** printPyramid και δέχεται ως όρισμα την τιμή N (στην περίπτωσή μας numOfRows), την αποθηκεύει στον καταχωρητή a0 και μέσω κατάλληλων βρόγχων επανάληψης με χρήση της εντολής addi, τυπώνεται ένα τρίγωνο που έχει μέγεθος όσο και ο αριθμός N (σε γραμμές) και ξεκινώντας απ' το 1 στην πρώτη γραμμή, εκτυπώνει τους αριθμούς διαδοχικά έως το N στην N-οστή γραμμή. Στις συνθήκες επανάληψης χρησιμοποιήθηκαν οι εντολές **slt (set less than)** και **bne (branch not equal)** για να ελεγχθεί αν οι συνθήκες ισχύουν. Ο τρόπος λειτουργίας τους είναι αρχικά να ελέγχεται αν δύο καταχωρητές τηρούν την συνθήκη (<). Ο έλεγχος, θα οριστεί σε έναν τρίτο καταχωρητή, ο οποιος με την εντολή bne θα ελεγχθεί αν είναι ίσος με το 0, δηλαδή αν δεν ισχύει. Σε αυτή την περίπτωση θα γίνει jump στο label που θα αναγράφεται, αλλιώς θα συνεχιστεί κανονικά η ροή του προγράμματος.

```
#FUNCTION 1
printPyramid:

do_while_label:
while_1A:
addi $v0, $zero, 4
la $a0, numOfRows
syscall
addi $v0, $zero, 5
syscall
add $s1, $zero, $v0

slt $t8, $zero, $s1
bne $t8, $zero, after_label_1
addi $v0, $zero, 4
la $a0, positiveCheck
syscall
j while_1A

after_label_1:

addi $t0, $zero, 1
addi $t1, $zero, 1
add $t4, $zero, $zero
add $t5, $zero, $zero
add $t2, $zero, $s1
for_label_1:
slt $t8, $t2, $t0
bne $t8, $zero, after_for_1
addi $t1, $zero, 1
for_label_2:

slt $t8, $t0, $t1
bne $t8, $zero, after_for_2
addi $v0, $zero, 1
add $a0, $zero, $t1
syscall
addi $v0, $zero, 4
la $a0, tab
syscall
addi $t1, $t1, 1
j for_label_2
after_for_2:
addi $t0, $t0,1
addi $v0, $zero, 4
la $a0, space
syscall
j for_label_1
```

```
after_for_1:
addi $v0, $zero, 4
la $a0, totalSum
syscall

addi $t1, $s1, 1
mult $t1, $s1
mflo $t1
srl  $t1, $t1, 1

addi $v0, $zero, 1
add $a0, $zero, $t1
syscall

addi $v0, $zero, 4
la $a0, space
syscall
jr $ra
```

***2η συνάρτηση:*** checkNumber που δέχεται ως όρισμα την τιμή N, την αποθηκεύει στον καταχωρητή a0 και ομοίως μέσω κατάλληλων βρόγχων επανάληψης και χρήση της εντολής addi, και του αριθμου 5 γίνεται η ανάγνωση του αριθμού από τον χρήστη. Μέσω μιας επανάληψης, αφαιρώντας το 2 κάθε φορά ο αριθμός φθίνει. Η επανάληψη γίνεται παλι με τις εντολές **blt (branch less than), bne (branch not equal)**, οι οποίες λειτουργούν όπως περιγράφηκαν στην συνάρτηση 1. Όταν γίνει 0 θα εκτυπωθεί μήνυμα ότι είναι άρτιος, αλλιώς αν γίνει 1 θα εκτυπωθεί ότι είναι περιττός.

```
#FUNCTION 2
checkNumber:
while1B:
addi $v0, $zero, 4
la $a0, enterNumber
syscall
addi $v0, $zero, 5
syscall
add $s2, $v0, $zero

if_label_1:
addi $t8,$t8,-1
addi $t9,$zero, 0
sge $t5, $s2, $t9
bne $t5, $zero, after_if_1
mul $s2, $s2, -1

after_if_1:
add $t0, $s2, $zero
```

1

```
if_label_2:
blt $t0, 2, if_label_3
addi $t0, $t0, -2
j if_label_2
if_label_3:
bne  $t0, $zero, else_label
addi $v0, $zero, 4
la $a0, even
syscall
j after_cond
else_label:
addi $v0, $zero, 4
la $a0, odd
syscall

after_cond:
jr $ra
```

2

```
#FUNCTION 3
multiplyNum:

add $t8, $zero, $s1     #move arrayA to register t8
add $t9, $zero, $s2     #move arrayB to register t9

addi $t4, $zero, 0      #output = 0
addi $t0, $zero, 0      #i = 0
addi $t1, $zero, 5
if_label_1c:

sge $t5, $t0, $t1
bne $t5, $zero, end_if_1
addi $t2, $zero, 0
lw $t3, 0($t8)  #input = A[0]

sll  $t3, $t3, 2 #multiply by 2^2

end_if_2:
sw $t3, 0($t9)
addi $t9, $t9, 4
addi $t8, $t8, 4
addi $t0, $t0, 1
addi $t4, $zero, 0
j if_label_1c
end_if_1:

jr $ra
```

***3η συνάρτηση:*** multiplyNum η οποία χρησιμοποιεί 2 πίνακες που δέχεται ο καθένας 5 ακέραιους στους καταχωρητές t8 και t9 αντίστοιχα. Προηγουμένως, έχουμε μετακινήσει τους πίνακες από τους καταχωρητές s1 και s2 στους t8 και t9. Στον 1ο πίνακα αποθηκεύονται οι τιμές που θα εισάγει ο χρήστης στην main. Στην συνέχεια πάλι με την χρήση των εντολών **sge(set greater than)**, **bne (branch not equal)**, γίνεται επαναληπτικά η διαδικασία πολλαπλασιασμού για κάθε έναν από τους 5 ακεραίους. Στην συνέχεια με την χρήση της εντολής **lw (load word)**, εκχωρείται η τιμή του πρώτου στοιχείου του πίνακα A στον καταχωρητή t3. Στην συνέχεια μέσω της εντολής sll (shift left logical) υλοποιείται ο πολλαπλασιασμός του εκάστοτε αριθμού με το 2^2 που ήταν το ζητούμενο. Ουσιαστικά γίνεται αριστερή ολίσθιση κατά 4 bytes, που τελικά μας δίνει το επιθυμητό αποτέλεσμα. Τελικά αυτό αποθηκεύεται στον πίνακα B μέσω της εντολής **sw (store word)**, όπου εκχωρείται η τιμή του πρώτη στοιχείου του πίνακα B, στον καταχωρητή t3. Επιπλέον γίνεται και πρόσθεση +4 στον κάθε καταχωρητή που αντιστοιχεί στους πίνακες A, B για να γίνει η επαναληπτική διαδικασία.

***4η συνάρτηση:*** modifyString έχει να κάνει με την μετατροπή των πεζών σε κεφαλαία και αντίστροφα, σε μια συμβολοσειρά. Χρησιμοποιούμε 2 πίνακες μεγέθους 100 και τους αποθηκεύουμε στους καταχωρητές s4 και s5 αντίστοιχα. Επίσης, μεταφέρουμε το περιεχόμενό τους στους t8 και t9. Μέσω των εντολών **slt(set less than), bne(branch not equal)**, **sge(set greater or equal)**, ελέγχουμε κάθε φορά τους χαρακτήρες αν βρίσκονται στο διάστημα 65 έως 90 που αφορά τους κεφαλαίους χαρακτήρες ή στο 97 έως 122 που αφορά τους πεζούς. Μέσα στο if_label_2d η εντολή **lb (load byte)** φορτώνει τον αριθμό που βρίσκεται στην πρώτη θέση του πίνακα array_a στον καταχωρητή t2. Στην συνέχεια γίνονται οι κατάλληλοι έλεγχοι για το τι είναι ο κάθε χαρακτήρας. Συγκεκριμένα αν ο χαρακτήρας είναι κεφαλαίος προστίθεται το 32, για να φτάσει στο διάστημα των πεζών χαρακτήρων, αν είναι πεζός αφαιρείται το 32 για να φτάσει στο πεδίο των κεφαλαίων και αν είναι κάποιος διαφορετικός χαρακτήρας (ούτε κεφαλαίο, ούτε πεζός) δεν γίνεται τίποτα, δηλαδή επιστρέφεται ο ίδιος χαρακτήρας. Στην συνέχεια, με την χρήση της εντολής **sb (store byte)** εκχωρείται η τιμή του καταχωρητή t2 στην πρώτη θέση του πίνακα array_b. Τελικά προστίθεται 1 σε κάθε πίνακα για να γίνεται η επαναληπτική διαδικασία και επιστρέφεται η αλλαγμένη συμβολοσειρά σην main.

CLANG

```
long long modifyStr(long long *R4, long long *R5) {
            R4 = (long long *)a;
            R5 = (long long *)b;

            R10 = 0;
if_label_1:
            if(R10 > 100) goto end_if;


if_label_2: if(a[R10]<=64) goto else_lbl_1;
            if(a[R10]>=91) goto else_lbl_1;
            b[R10] = a[R10]+32;
            R10 = R10+1;
            goto if_label_1;
else_lbl_1: if(a[R10]<=96) goto else_lbl_2;
            if(a[R10]>=123) goto else_lbl_2;
            b[R10] = a[R10] - 32;
            R10 = R10+1;
            goto if_label_1;
else_lbl_2: b[R10] = a[R10];
            R10 = R10+1;
            goto if_label_1;

end_if:     R2 = (long long)b;
            return R2;
}
```

```
#FUNCTION 4
modifyString:
addi $t0, $zero, 0

if_label_1d:
add $t8, $a0, $t0
add $t9, $a1, $t0

slti $t5, $t0, 100
beq $t5, $zero, end_if
if_label_2d:
lb $t2, 0($t8)
addi $t3, $zero, 64
addi $t4, $zero, 91
slt $t5, $t2, $t3
bne $t5, $zero, else_lbl_1d
slt $t5, $t2, $t4
beq $t5, $zero, else_lbl_1d

addi $t6, $t2, 32
sb $t6, 0($t9)
addi $t0, $t0, 1
j if_label_1d

else_lbl_1d:
addi $t3, $zero, 96
addi $t4, $zero, 123
slt $t5, $t2, $t3
bne $t5, $zero, else_lbl_2d
slt $t5, $t2, $t4
beq $t5, $zero, else_lbl_2d
addi $t6, $t2, -32
sb $t6, 0($t9)
addi $t0, $t0, 1
j if_label_1d

else_lbl_2d:
sb $t2, 0($t9)
addi $t0, $t0, 1
j if_label_1d
end_if:


jr $ra
```

## Συμπεράσματα

Ολοκληρώνοντας την τέταρτη εργαστηριακή άσκηση, εξοικειωθήκαμε με τη χρήση της γλώσσας Assembly, ποιους καταχωρητές πρέπει να χρησιμοποιούμε και γιατί, καθώς επίσης και με τους βρόγχους επανάληψης και τις κλήσεις συναρτήσεων. Επίσης να σημειωθεί οτι δεν επιτρεπόταν η χρήση ψευδοεντολών.