ΨΗΦΙΑΚΟΙ ΥΠΟΛΟΓΙΣΤΕΣ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΟΥ 4

ΜΙΧΑΛΗΣ ΓΑΛΑΝΗΣ 2016030036
ΓΙΩΡΓΟΣ ΒΙΡΙΡΑΚΗΣ 2016030035

Προεργασία

Στη 4η εργαστηριακή άσκηση κύριο ζητούμενο ήταν η υλοποίηση σε Assembly του κώδικα που είχαμε κατασκευάσει σε Clang στο προηγούμενο εργαστήριο, συμπεριλαμβανομένου και μιας επιπλέον συνάρτησης, την οποία καλούμασταν να υλοποιήσουμε τόσο σε Clang αλλά και σε Assembly.

Περιγραφή Ζητούμενων

Σκοπός αυτού του εργαστηρίου ήταν η εμβάθυνση στην γλώσσα Assembly και η περαιτέρω εξοικείωση με τις λειτουργίες της.

Το πρόγραμμα εκτελούταν επαναληπτικά και περιείχε 4 συναρτήσεις, οι οποίες περιλάμβαναν την χρήση εντολών ελέγχου, επαναληπτικών βρόγχων, ανάγνωση και εκτύπωση στοιχείων, καθώς και χρήση και επεξεργασία πινάκων και τις διευθύνσεις μνήμης τους.

Περιγραφή της Εκτέλεσης

Το πρόγραμμά μας περιείχε τις συναρτήσεις **main, function_1, function_2, function_3** και **function_4.**

- **CLANG**

  Για τη Clang, το πρόγραμμά μας αυτή τη φορά περιείχε και μια 4η συνάρτηση η οποία σε μια σειρά χαρακτήρων μετέτρεπε τα κεφαλαία γράμματα σε μικρά και αντίστροφα. Οι αριθμοί και τα συμβολα δεν επηρεαζόντουσαν.

```
void function_4(int R4, int R5){
    char c,d;
    R15 = c;
    R16 = d;
    R18 = 0;
    R15 = charArrayIn[R18];

    While_label:
    if (R15 == NULL) goto after_loop; //While
        if (R15<97) goto else_label_1;
        if(R15>122) goto else_label_1;
            R15 = R15 - 32;
            goto after_cond;
    else_label_1:
        if (R15<65) goto else_label_2;
        if(R15>90) goto else_label_2;
            R15 = R15 + 32;
            goto after_cond;
        else_label_2:

        after_cond:
                R16 = R15;
                charArrayOut[R18] = (char) R16;
                R18 = R18 + 1;
                R15 = charArrayIn[R18];
    goto While_label;
    after_loop:
    return;
}
```

- **ASSEMBLY**

Στην Assembly, μας ζητήθηκε αυτή τη φορά να κατασκευάσουμε όλες τις συναρτήσεις. Γενικώς η διαδικασία που ακολουθούσαμε ήταν η εξής:

Αρχικά ζητούσαμε καποιο είδους input απο το χρήστη. Αμέσως μετά αποθηκεύαμε το input σε όσους $a καταχωρητές χρειαζόταν και με την εντολή **jal** καλούσαμε τη συνάρτηση η οποία επέστρεφε (με την εντολή **jr $ra**) με τη σειρά της τον καταχωρητή $v0.

- Στη **main**:

```
li $v0,5 #"READ INTEGER FROM USER"
syscall
move $t0,$v0

move $a1,$t0 #"$a1 is the first parameter"

jal FUNCTION_1 #"Calls Function_1, Sends $a1, returns $v0"
```

- Στη **συνάρτηση**:

```
move $v0,$t1
jr $ra #"End of Function_1"
```

Η πρώτη συνάρτηση αποτελούταν απο μια εμφωλευμένη for loop. Το ιδιαίτερο σημείο είναι ότι για κάθε επανάληψη της εξωτερικής loop, χρειάστηκε να επαναφέρουμε τον counter της εσωτερικής loop στην αρχική του τιμή για να λειτουργήσει σωστά.

```
#"Sets j to 1 every time the Inside Loop Ends"
li $t3,1
```

Στη δεύτερη συνάρτηση για να αποφύγουμε τη χρήση της συνάρτησης modulo έπρεπε ανάλογα με το πρόσημο του αριθμού να προσθέσουμε ή να αφαιρέσουμε το 2 μέχρι να γίνει 0 ή 1. Στην περίπτωση αρνητικού αριθμού, η αντιστροφή του αριθμού έγινε με την εντολή **subu**.

```
FUNCTION_2: #"Input: $a1, Output: $v0"
    move $t0,$a1 #"a1 (number extracted from user) is reserved"

    blt $t0,$zero,else_function2_1 #"Positive Numbers"

        while_label_function2_1:
            ble $t0,1,after_loop_function2_1

                addi $t0,$t0,-2 #"N = N-2"

                j while_label_function2_1
                after_loop_function2_1:
                    move $v0,$t0 #"$t0 copied into return variable $v0"
                    jr $ra #"End of Function_2"

    else_function2_1: #"Negative Numbers"
        while_label_function2_2:
            bge $t0,-1,after_loop_function2_2

                addi $t0,$t0,2 #"N = N+2"

                j while_label_function2_2
                after_loop_function2_2:
                    subu $t0,$zero,$t0 #"t0 becomes -t0, returns -N"
                    move $v0,$t0 #"$t0 copied into return variable $v0"
                    jr $ra #"End of Function_2"
```

Στη 3η συνάρτηση χρησιμοποιήσαμε τους καταχωρητές $a1,$a2 για να προσπελάσουμε στοιχεία του πίνακα.

```
la $a1,intArray1 #"$a1 is the adress of the first integer of intArray1"
la $a2,intArray2 #"$a2 is the adress of the first integer of intArray2"
```

Αρχικά οι δύο πίνακες ήταν κενοί. Στη main γέμιζε ο πρώτος πίνακας απο το χρήστη. Ύστερα η συνάρτηση υπολόγιζε τους εξαπλάσιους αριθμούς και τους τοποθετούσε διαδοχικά στον δεύτερο πίνακα. Τέλος επέστρεφε στη main για να εκτυπωθούν οι δύο πίνακες.

Η ανάγνωση στοιχείων απο τον πίνακα γινόταν με τον εξής τρόπο:

```
lw $t5, ($a1) #"Stores value of each consecutive integer to $t5"
```

Η εγγραφή στοιχείων στον πίνακα γινόταν με τον εξής τρόπο:

```
sw $v0, ($a1) #"Storing N to intArray1"
```

Η μετακίνηση κάθε φορά στο επόμενο στοιχείο του πίνακα γινόταν με πρόσθεση του κατάλληλου **$α** κατά 4 θέσεις (4 επειδη είναι integers).

```
addiu $a1, $a1, 4 #"Move $a1 to next integer"
```

Στην συνάρτηση για να πραγματοποιηθεί ο πολλαπλασιασμός των στοιχείων, χρειαζόταν για κάθε στοιχείο του πίνακα (μια for loop) να προσθέσουμε τον εαυτό του 6 φορές (αλλη μια for loop). Ένα ιδιαίτερο σημείο είναι ότι στη πρόσθεση έπρεπε να κρατήσουμε σταθερό έναν απο τους δύο προσθετέους για να αποφύγουμε την ύψωση σε δύναμη.

```
label_loop_function3_1: #"Outside For Loop (for each integer)"
    bge $t1,5,after_loop_function3_1 #"5 for 5 integers"

        lw $t5, ($a1) #"Stores value of each consecutive integer to $t5"
        lw $t3, ($a1) #"Stores value of each consecutive integer to $t5 again for multiplication and not exponent"

        addiu $a1,$a1,4 #"Move $a1 to next integer"

        label_loop_function3_2:  #"Inside For Loop (multiplication by 6)"
            bge $t2,6,after_loop_function3_2 #"6 for times added"


                add $t5,$t5,$t3 #"$t5 = $t5 + $t3 (multiplication), t3 is constant"

                addi $t2,$t2,1 #"j++"


                j label_loop_function3_2
                after_loop_function3_2:

                    li $t2,1 #"Sets j to 1 every time outside loop occurs"

                    addi $t1,$t1,1 #"i++"
                    sw $t5, ($a2)
                    addiu $a2, $a2, 4 #"Move $a2 to next integer"

                    j label_loop_function3_1
                    after_loop_function3_1:

                        move $v0,$t0 #"$t0 (value of $a2 at start) copied into return variable $v0"
                        jr $ra #"End of Function_3"
```

Στη τέταρτη συνάρτηση, όπως και με τη προηγούμενη, χρειαστήκαμε και πάλι στη main τα **$a1,$a2** για δυο πίνακες char. Η συνάρτηση επεξεργαζόταν το String που εισήγαγε ο χρήστης με τον παρακάτω τρόπο:

```
FUNCTION_4: #"Input: $a1,$a2 Output: $v0"

    lb $t0, ($a1) #"Store value of first char to $t0"

    while_label_function4:
        beq $t0,$zero,after_loop_function4 #"While char!=NULL"
            addi $a1,$a1,1 #"Move $a1 to next char"
            blt $t0,97,else_function4_1 #"Old is Lowercase"
            bgt $t0,122,else_function4_1 #"Old is Lowercase"

                addi $t0,$t0,-32 #"Convert to Uppercase"
                j after_cond_function4

        else_function4_1:
            blt $t0,65,else_function4_2 #"Old is Uppercase"
            bgt $t0,90,else_function4_2#"Old is Uppercase"

                addi $t0,$t0,32 #"Convert to Lowercase"
                j after_cond_function4

            else_function4_2:
        after_cond_function4:
            sb $t0,($a2) #"Store"
            addi $a2,$a2,1 #"Move $a2 to next char"
            lb $t0, ($a1) #"Store value of current char to $t0"

        j while_label_function4
        after_loop_function4:
        jr $ra
```

Ύστερα με απλό τρόπο εκτυπώσαμε τα δυο Strings στην οθόνη.

Bonus: Διαπιστώσαμε ότι εαν ξαναχρησιμοποιήσουμε αυτη τη συνάρτηση, «επανεγγραφόταν» ο πίνακας με αποτέλεσμα εαν τη δεύτερη φορά το String ήταν μικρότερο θα κρατούσε τους παλιούς χαρακτήρες που δεν εγγράφηκαν τη δεύτερη φορά. Άρα η λύση ήταν να «καθαρίζουμε» τους δυο πίνακες με τα String κάθε φορά που εκτελούταν η συνάρτηση.

```
#"Clearing String at the end"
li $t6,0 #"i = 0"
while_label_MAIN_function4:
    bge $t6,100,after_loop_MAIN_4 #"100 For Loops"

        #"Clearing oldString"
        lb $t4,($a1)
        move $t4,$zero
        sb $t4,($a1)

        #"Clearing newString"
        lb $t5,($a2)
        move $t5,$zero
        sb $t5,($a2)

        #"Move $a's to next char"
        addi $a1,$a1,1
        addi $a2,$a2,1

        #"i++"
        addi $t6,$t6,1

        j while_label_MAIN_function4
    after_loop_MAIN_4:
```

Συμπέρασμα

Στο 4ο εργαστήριο επιτεύχθηκε η περαιτέρω εμβάθυνση στην Assembly. Συγκεκριμένα μάθαμε να καλούμε συναρτήσεις, να περνάμε παραμέτρους και να επιστρέφουμε άλλες. Κάποια σημεια ήθελαν ιδιαίτερη προσοχή αλλά πιστεύουμε ότι τα αντιμετωπίσαμε με επιτυχία.