9/12/2017

Αναφορά εργαστηρίου 5

ΤΡΙΜΑΣ ΧΡΗΣΤΟΣ
ΝΙΚΟΥ ΓΕΩΡΓΙΟΣ ΝΕΚΤΑΡΙΟΣ

Προεργασία:

Για την υλοποίηση του 5ου εργαστηρίου ψηφιακών υπολογιστών, ζητήθηκε ως προεργασία η υλοποίηση ενός προγράμματος στη γλώσσα Clang και η μετατροπή σε MIPS-Assembly.

Περιγραφή Ζητούμενων:

Η άσκηση είχε ως σκοπό την εξοικείωση μας με την χρήση αναδρομικών συναρτήσεων. Ως ζητούμενο είχαμε την κατασκευή κατάλληλου προγράμματος για την διάσχιση ενός λαβύρινθου που εισάγεται. Ο λαβύρινθος αποτελείται από τελείες και Ι που αντιστοιχούν στους διαδρόμους και στους τοίχους αντίστοιχα. Επιπλέον δίνονται διάφορα στοιχεία για τον λαβύρινθο όπως μέγεθος, διαστάσεις κλπ. Το πρόγραμμα οφείλει να διατρέχει τους διαδρόμους του λαβύρινθου από την έναρξη, που δίνεται, και να μετατρέπει τα βήματα σε '*' μέχρι να συναντήσει την έξοδο '@'. Τέλος τυπώνει την βέλτιστη διαδρομή προς αυτήν αντικαθιστώντας τις τελείες με την '#' και την έξοδο με '%'.

Περιγραφής της Εκτέλεσης:

Αρχικά κατασκευάσαμε την main με τις κατάλληλες κλήσεις δύο συναρτήσεων, που μας δίνονταν, σε γλώσσα C. Έπειτα κληθήκαμε να υλοποιήσουμε την συνάρτηση σε Clang ,η οποία αντιστοιχίζεται 1-1 με τον κώδικα assembly που υλοποιήθηκε.
Έτσι χρησιμοποιήσαμε ονόματα καταχωρητών του mips ως global ακεραίους(T0-T9, ,S0-S7 κλπ.),οι συναρτήσεις μας ήταν void και τέλος τα ορίσματα και οι επιστροφές συναρτήσεων πραγματοποιούνταν μέσω των ακεραίων A0 και V0 αντίστοιχα.
Τέλος υλοποιήσαμε την assembly έχοντας ως βάση τον κώδικα clang πρώτα χωρίς χρονική καθυστέρηση(usleep) και ύστερα με την χρήση κατάλληλης συνάρτησης-μετρητή που προσομοιώνει την usleep.

Συμπεράσματα:

Με τη επιτυχή ολοκλήρωση του 5^{ου} εργαστηρίου, έγινε κατανοητό, ο τρόπος λειτουργείας της αναδρομής σε γλώσσα MIPS assembly. Συγκεκριμένα, είδαμε έναν πιο ουσιαστικό ρόλο των καταχωρητών a0 και ra, καθώς επίσης έγινε κατανοητή η λειτουργεία του stack για καταχώρηση των επιθυμητών τιμών.

Παράρτημα(κώδικας):

Παρατίθεται κάθε κομμάτι κώδικα σε clang και ύστερα assembly και υπάρχει ο ανάλογος σχολιασμός από κάτω

```
void printLabyrinth(void){
    int i=0,j=0,k=0;
    T0 = i;
    T1 = j;
    T2 = k;
    usleep(200000);
    printf("Labyrinth:\n");

    for_label_1:
    if(T0>=S4) goto exit_label;
    T1 = 0;
    for_label_2:
    if(T1>=S3) goto continue_label;
    tmp[T1] = map[T2];  //προσθηκη μιας σειρας στοιχειων του λαβυρινθου

    T1 = T1 + 1;
    T2 = T2 + 1;
    goto for_label_2;

    continue_label:
    printf("%s\n",tmp); //εκτύπωση κάθε φορά του temp

    T0 = T0 +1;
    goto for_label_1;

    exit_label:
    return;
```

```
printLabyrinth:
        addi $sp,$sp,-8
        sw $a0,0($sp)
        sw $ra,4($sp)


        li $a0,100000
        jal usleep

        li $t0, 0       #i
        li $t1, 0       #j
        li $t2, 0       #k

        li $v0, 4
        la $a0, labmessage
        syscall

        li $v0,4
        la $a0, newline
        syscall

        for_label_1:
        bge $t0,$s4,exit_label
        li $t1,0
        for_label_2:
        bge $t1,$s3,continue_label
        lb $t3, map($t2)
        sb $t3, tmp($t1)        #temp[j]=map[k]

        addi $t1,$t1,1
        addi $t2,$t2,1
        j for_label_2

        continue_label:
        li $v0,4
        la $a0,tmp
        syscall

        li $v0,4
        la $a0,newline
        syscall

        addi $t0,$t0,1
        j for_label_1

        exit_label:
        lw $a0,0($sp)
        lw $ra,4($sp)
        addi $sp,$sp,8
        jr $ra
```

Στη πρώτη συνάρτηση(printLabyrinth) σε έναν βρόγχο καταχωρούμε μια σειρά κάθε φορά του λαβυρίνθου(map) στο temp και κατόπιν το εκτυπώνουμε. Επιπλέον στην assembly σώζουμε στην στοίβα τις μεταβλητές που δεν θέλουμε να χαθούν με την κλήση της usleep.

```
void makeMove(void){
    int stack[2];
    stack[0] = A0;
    stack[1] = RA;
    int S0 = A0; //S0 = index(startX)
    int T1 = 1;

    if(A0>=0) goto scnd;
        goto end_if;
    scnd:
    if(A0<S1) goto if_label;
        goto end_if;

    if_label:
    if(map[A0] != '.') goto else_if;//ελεγχος για το σωστο path
        map[A0] = '*'; //αντικατασταση του με *
        printLabyrinth();//κληση της συναρτησης για εκτυπωση
        A0 = stack[0];
```

```
makeMove:
        addi $sp,$sp,-8
        sw $a0,0($sp)
        sw $ra,4($sp)

        bgez $a0,scnd
        j end_if
        scnd:
        blt $a0,$s1,if_label
        j end_if


        if_label:
        la $t0,map
        add $t1,$t0,$a0
        lb $t0,0($t1)

        bne $t0,46,el_if
        li $t0,42
        sb $t0,0($t1)
        jal printLabyrinth
        lw $a0,0($sp)
```

Σε πρώτη φάση στη συνάρτηση makeMove κάνουμε μια κατοχύρωση(<<εικονική>> για τη clang) των A0 και RA στο stack σύμφωνα με τις απαιτήσεις της εκφώνησης. Αφού γίνει ο έλεγχος του στοιχείου που εξετάζουμε, και ανήκει στον λαβύρινθο, πραγματοποιείται ο έλεγχος για τελεία(πιθανό σωστό path) και τις αντικαθιστά με το σύμβολο ‘*’. Στη συνέχεια εκτυπώνει τον νέο λαβύρινθο.

```
if_1:
     A0= S0+T1;
     makeMove();
     if(V0 != T1) goto if_2;//ελεγχος του δεξιου στοιχειου
          A0 = stack[0];
          map[A0] = '#';
          V0 = 1;
          return;

if_2:
     A0= S0+S3;
      makeMove();

     if(V0 != T1) goto if_3;//ελεγχος του κατω στοιχειου
          A0 = stack[0];
          map[A0] = '#';
          V0 = 1;
          return;

if_3:
     A0= S0-T1;
     makeMove();

     if(V0 != T1) goto if_4;//ελεγχος του αριστερου
          A0 = stack[0];
          map[A0] = '#';
          V0 = 1;
          return;

if_4:
     A0= S0-S3;
     makeMove();

     if(V0 != T1) goto end_if;//ελεγχος του πανω στοιχειου
        A0 = stack[0];
          map[A0] = '#';
          V0 = 1;
          return;
```

```
if_1:
addi $a0,$a0,1                    #index=index+1
jal makeMove
lw $a0,0($sp)

bne $v0,1,if_2                    #if(makeMove(index+1)==1)
la $t3,map
add $t3,$t3,$a0
li $t0,35
sb $t0,0($t3)                     #map[index]='#'
li $v0,1
j ret_label

if_2:
addi $a0,$a0,21
jal makeMove
lw $a0,0($sp)

bne $v0,1,if_3
la $t3,map
add $t3,$t3,$a0
li $t0,35
sb $t0,0($t3)                     #map[index]='#'
li $v0,1
j ret_label

if_3:
addi $a0,$a0,-1
jal makeMove
lw $a0,0($sp)
bne $v0,1,if_4
la $t3,map
add $t3,$t3,$a0
li $t0,35
sb $t0,0($t3)                     #map[index]='#'
li $v0,1
j ret_label

if_4:
addi $a0,$a0,-21
jal makeMove
lw $a0,0($sp)

bne $v0,1,end_if
la $t3,map
add $t3,$t3,$a0
li $t0,35
sb $t0,0($t3)                     #map[index]='#'
li $v0,1
j ret_label
```

Σε αυτό το κομμάτι κώδικα, γίνεται αναδρομική κλήση της συνάρτησης makeMove, και γίνεται έλεγχος των γειτονικών στοιχείων για '.' .Τέλος αντικαθίσταται το συντομότερο δυνατό path για την έξοδο με # και το V0 γίνεται ίσο με 1.

```
else_if:
    if(map[A0] != '@') goto end_if;  //ευρεση του τελους
        map[A0] = '%';               //αντικατασταση με το % συμβολο

        printLabyrinth();
        V0 = 1;
        goto return_label;

end_if:
    V0= 0;

return_label:
A0 = stack[0];
S0 = stack[1];
return;
```

```
el_if:
la $t0,map
add $t1,$t0,$a0
lb $t0,0($t1)

bne $t0,64,end_if
li $t0,37
sb $t0,0($t1)
jal printLabyrinth
li $v0,1
j ret_label

end_if:
li $v0,0

ret_label:
lw $a0,0($sp)
lw $ra,4($sp)
addi $sp,$sp,8
jr $ra
```

Εδώ πραγματοποιείται έλεγχος για την εύρεση του τέλους του map και αντικατάσταση αυτού(@) με το σύμβολο '%'. Τέλος, γίνεται η επαναφορά των αρχικών τιμών από το stack.

```
usleep:
        addi $sp,$sp,-8
        sw $a0,0($sp)
        sw $ra,4($sp)

        li $t0,0

        for_lab:
        bge $t0,$a0,e_label
        addi $t0,$t0,1
        j for_lab

        e_label:
        lw $a0,0($sp)
        lw $ra,4($sp)
        addi $sp,$sp,8
        jr $ra
```

Η συνάρτηση έχει έναν απλό βρόγχο που μετράει έως έναν μεγάλο αριθμό με σκοπό να καθυστερεί αρκετά το πρόγραμμα ώστε να είναι κατανοητή βήμα-βήμα η λειτουργία του.