ΨΗΦΙΑΚΟΙ ΥΠΟΛΟΓΙΣΤΕΣ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΟΥ 5

ΜΙΧΑΛΗΣ ΓΑΛΑΝΗΣ 2016030036
ΓΙΩΡΓΟΣ ΒΙΡΙΡΑΚΗΣ 2016030035

Προεργασία

Στην 5η εργαστηριακή άσκηση μας ζητήθηκε να κατασκευάσουμε ένα Maze Solver. Μας δόθηκε αρχικά πρότυπος κώδικας σε γλώσσα C, τον οποίο καλούμασταν να μετατρέψουμε σε Clang και ύστερα σε Assembly.

Περιγραφή Ζητούμενων

Η 5η εργαστηριακή άσκηση αποσκοπούσε στην εξοικείωσή μας με τη χρήση στοίβας (stack), των αναδρομικών συναρτήσεων και τις συμβάσεις των καταχωρητών σε Assembly MIPS.

Το πρόγραμμα ξεκινούσε από ένα σημείο του χάρτη που ορίζαμε εμείς και «έψαχνε» επαναληπτικά την έξοδο (δηλαδή το **@**), ώστε να μας δείξει ύστερα τη βέλτιστη διαδρομή που εκτελέστηκε.

Στη Clang το πρόγραμμα περιείχε τη main και 2 συναρτήσεις, την **PrintLabyrinth** και την **MakeMove.** Η PrintLabyrinth εκτύπωνε τον Λαβύρινθο στην οθόνη και η MakeMove ήταν υπεύθυνη για την εύρεση της λύσης του.

Στην Assembly εκτός από τις παραπάνω συναρτήσεις, είχαμε και την **Usleep** , η οποία καθυστερούσε την εκτέλεση, ώστε η εμφάνιση του λαβυρίνθου σε κάθε βήμα της αναδρομής να γίνεται πιο ομαλά. Στη C και στη Clang, αυτή η συνάρτηση ήταν ενσωματωμένη στην PrintLabyrinth.

Περιγραφή της Εκτέλεσης

- **C**

Αφού μας δόθηκε το πρόγραμμα σε γλώσσα C, ουσιαστικά το μόνο που έπρεπε να κάνουμε ήταν να καλέσουμε από τη main τη συνάρτηση MakeMove με το κατάλληλο όρισμα (=σημείο εκκίνησης του χάρτη) και να εκτυπώσουμε τον τελικό χάρτη με τη βέλτιστη διαδρομή.

```
int main(){
    makeMove(startX);
    printLabyrinth();
    return 0;
}
```

- **CLANG**

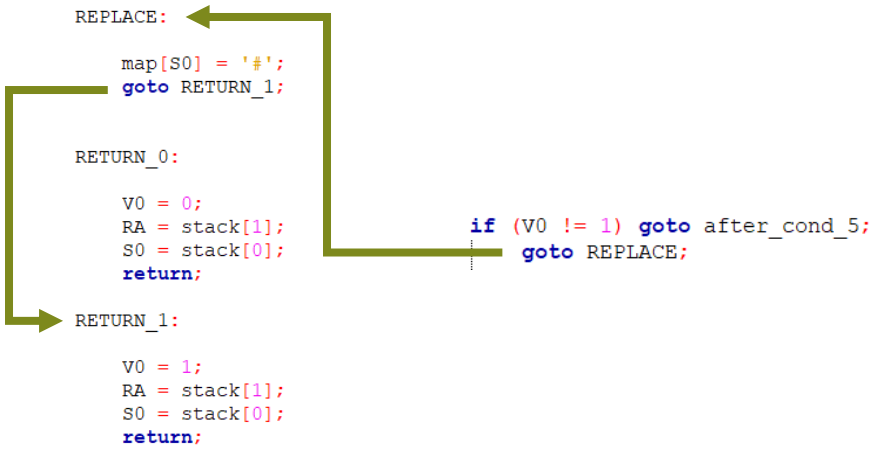
Η μετατροπή στη γλώσσα Clang προϋπέθετε να ακολουθήσουμε τις συμβάσεις καταχωρητών που υπάρχουν στην Assembly. Συγκεκριμένα, χρησιμοποιήσαμε τους καταχωρητές με τα ονόματά τους, αντί για τους αριθμημένους **R0-R31.**

```
//Register variables
int V0;
int A0;
int T0,T1,T2,T3,T4,T5,T6,T7,T8,T9;
int S0,S1;
int RA;
```

`Η μεταβλητή RA στη Clang δεν έχει πρακτικό σκοπό αλλά, για να υπάρξει αντιστοιχία με την Assembly τη χρησιμοποιούμε όπως τον καταχωρητή **$ra** και σε κάθε κλήση συνάρτησης αποθηκεύουμε την τιμή του **RA** στη στοίβα. Στην αρχή κάθε συνάρτησης δεσμεύαμε κατάλληλο χώρο (stack) στον οποίο διατηρούσαμε τους επιθυμητούς καταχωρητές μεταξύ επαναλήψεων της αναδρομής.

```
stack[1] = RA;
printLabyrinth();
RA = stack[1];
```

Στη Clang και στην Assembly, για εξοικονόμηση και αποφυγή αντίγραφου κώδικα χρησιμοποιήσαμε labels που τα καλούσαμε από κάποιο σημείο μιας συνάρτησης



- **Assembly**

Ο χάρτης στην Assembly είναι ένας μονοδιάστατος πίνακας χαρακτήρων και έπρεπε να δηλωθεί με τον παρακάτω τρόπο. Η προσπέλαση στοιχείων σε αυτόν γινόταν με τη φόρτωση της διεύθυνσης του πρώτου στοιχείου του πίνακα και ολίσθηση του σε όποιο στοιχείο θέλουμε.

```
map:
    .ascii  "I.IIIIIIIIIIIIIIIIII"
    .ascii  "I....I....I.......I.I"
    .ascii  "III.IIIII.I.I.III.I.I"
    .ascii  "I.I.....I..I..I.....I"
    .ascii  "I.I.III.II...II.I.III"
    .ascii  "I...I...III.I...I...I"
    .ascii  "IIIII.IIIII.III.III.I"
    .ascii  "I............I.I...I"
    .ascii  "IIIIIIIIIIIIII.I.III"
    .ascii  "@.............I..II"
    .asciiz "IIIIIIIIIIIIIIIIIIII"
```

```
    #"T7 = MAP[T6]
    la      $t7,map
    addu    $t7,$t7,$t6
    lb      $t7,($t7)
```

```
#"USLEEP"
USLEEP:
    li      $t0,0

for_loop:
    bge     $t0,30000,after_loop        #"<30000 for faster and >30000 is slower"
    addi    $t0,$t0,1
    j       for_loop

after_loop:

    jr      $ra                         #"FUNCTION RETURN"
```

Η συνάρτηση **USLEEP** αποτελούνταν από ένα **$t** καταχωρητή ο οποίος αυξανόταν επαναληπτικά μέχρι να φτάσει μια επιθυμητή τιμή για να εξασφαλιστεί η καθυστέρηση.

Η **PrintLabyrinth** αποτελούνταν από 1 εμφωλευμένη for loop που εκτύπωνε το χάρτη γραμμή-γραμμη με τη βοήθεια ενός πίνακα χαρακτήρων temp.

Παρομοίως με τη Clang, στην assembly δεσμεύαμε στην αρχή κάθε συνάρτησης χώρο για να αποθηκεύσουμε τις τιμές των καταχωρητών που μας ενδιαφέρουν. Αυτό επιτεύχθηκε με την εντολή **addi $sp, $sp,-4** για την printLabyrinth και **addi $sp, $sp,-12** για τη MakeMove. Ο αρνητικός αριθμός οφείλεται στο γεγονός ότι στη μνήμη οι υψηλές διευθύνσεις στη μνήμη βρίσκονται στο κάτω μέρος της στοίβας και αντιστρόφως. Για να αποθηκεύσουμε/ανακτήσουμε την τιμή κάποιου καταχωρητή χρησιμοποιούμε τις εντολές **sw** και **lw** αντίστοιχα. Στο τέλος της όμως αποδεσμεύουμε το χώρο που χρησιμοποιήσαμε με την εντολή **addi $sp, $sp, 4** για την printLabyrinth και **addi $sp, $sp, 12** για τη MakeMove.

Τα Labels στην Assembly πραγματοποιήθηκαν με τον ακόλουθο τρόπο:

```
 bne     $v0,1,after_cond_6          #"Checking if MAKE_MOVE returns $v0 = 1"

 j       REPLACE                     #"Replacing * with #"



#"Replace with #"
REPLACE:
    lw      $s1,8($sp)               #"Restoring value of $s1 from stack[2]"

    li      $t0,35                   #"ASCII CODE FOR # IS 35"
    sb      $t0,($s1)                #"Replacing * with #"

    j       RETURN_1                 #"Jumps to RETURN_1 label"


#"Return 0"
RETURN_0:

    lw      $s1, 8($sp)              #"Restoring value of $s1 from stack[2]"
    lw      $s0, 4($sp)              #"Restoring value of $s0 from stack[1]"
    lw      $ra, ($sp)               #"Restoring value of $ra from stack[0]"
    addiu   $sp, $sp, 12             #"Freeing stack memory"

    li      $v0,0
    jr      $ra                      #"FUNCTION RETURN"


#"Return 1"
RETURN_1:

    lw      $s1, 8($sp)              #"Restoring value of $s1 from stack[2]"
    lw      $s0, 4($sp)              #"Restoring value of $s0 from stack[1]"
    lw      $ra, ($sp)               #"Restoring value of $ra from stack[0]"
    addiu   $sp, $sp, 12             #"Freeing stack memory"

    li      $v0,1
    jr      $ra                      #"FUNCTION RETURN"
```

Συμπέρασμα

Στο 4ο εργαστήριο επιτεύχθηκε η περαιτέρω εξοικείωση στην Assembly. Ασχοληθήκαμε με το πως αντιμετωπίζουμε την αναδρομή και μάθαμε να διαχειριζόμαστε τη στοίβα.