Ψηφιακοί Υπολογιστές - Αναφορά Project

Ομάδα LAB20138010 13.12.2018

- Μιχαλόπουλος Χρήστος : 2016030088
- Ανδρεαδάκης Αντώνης : 2013030059

Προεργάσια

Για την υλοποίηση του Project του εργαστηρίου απαραίτητες ήταν πολύ καλές γνώσεις πάνω στην γλώσσα προγραμματισμού C, αλλά και στην Assembly καθώς χρειάστηκαν γνώσεις πάνω σε περίπλοκες γνώσεις, όπως η αναδρομή και ο συνεχόμενος έλεγχος διαφόρων περιορισμών.

Ζητούμενα Άσκησης

Η διάσχιση λαβυρίνθου αποτελεί χαρακτηριστικό παράδειγμα αναδρομικής διαδικασίας. Το πρόγραμμα που κληθήκαμε να ολοκληρώσουμε σε C και στην συνέχεια σε Assembly, παρέχει δύο λειτουργικότητες. Η πρώτη λειτουργικότητα του project είναι η υλοποίηση του παιχνιδιού του λαβύρινθου. Όταν πατηθεί το κατάλληλο πλήκτρο, εντοπίζεται η βέλτιστη διαδρομή από την είσοδο προς την έξοδο για οποιοδήποτε λαβύρινθο. Η επίλυση του λαβύρινθου, αφορά τη δεύτερη λειτουργικότητα που πρέπει να υλοποιηθεί. Δίνονται οι αντίστοιχες μεταβλητές που περιγράφουν το πλάτος, το ύψος και το σημείο εκκίνησης του λαβυρίνθου. Κάθε λαβύρινθος αποτελείται από ένα σημείο εισόδου, ένα σημείο εξόδου, τοίχους και διαδρόμους. Στην περίπτωσή της εργασίας μας, οι διάδρομοι είναι τα κελιά του πίνακα με περιεχόμενο '.'(τελεία), ενώ οι τοίχοι είναι τα κελιά με περιεχόμενο 'I'(κεφαλαίο “i”). Η διάνυση, καθώς και η επίλυση του λαβυρίνθου προϋποθέτει κινήσεις σε γειτονικούς διαδρόμους (πάνω, κάτω, αριστερά ή δεξιά) και όχι σε τοίχους. Επίσης, η μετακίνηση πραγματοποιείται κατά ένα στοιχείο κάθε φορά. Ο παίκτης κινείται στον λαβύρινθο, μέχρι να φτάσει στην έξοδό του, στην οποία ο πίνακας περιέχει το χαρακτήρα '@'. Χρησιμοποιεί τα πλήκτρα “W”, “S”, “A”, “D”, ώστε να κινηθεί πάνω, κάτω, αριστερά και δεξιά αντίστοιχα σε διαδρόμους. Η τρέχουσα θέση του χρήστη, περιγράφεται από το γράμμα 'P' στο λαβύρινθο. Παράλληλα μπορεί ο χρήστης να δει το μονοπάτι του, καθώς αφήνεται αυτόματα ένα ίχνος (*) πάνω από τις θέσεις που έχει κινηθεί. Οι κινήσεις πάνω σε τοίχο δεν επιτρέπονται. Όταν ο παίκτης φτάσει στη έξοδο του λαβυρίνθου, του εμφανίζει το μήνυμα “Winner Winner Chicken Dinner!”, ενώ αλλάζει το @ σε %. Εναλλακτικά,

όταν ο χρήστης πατάει το πλήκτρο “E” από το πληκτρολόγιο, εμφανίζεται το βέλτιστο μονοπάτι καθώς το πρόγραμμα λύνεται αυτόματα.

Αρχικά θα αναφερθούμε στο κομμάτι της C.

Εκτέλεση Άσκησης

Η άσκηση υλοποιήθηκε με την χρήση των εξής συναρτήσεων: **main**, **printLabyrinth**, **makeMove**, **leep.** Η λειτουργικότητα τους περιγράφεται παρακάτω.

- **PrintLabyrinth**: η συνάρτηση δημιουργεί και εκτυπώνει τον λαβύρινθο. Με δύο επαναληπτικές διαδικασίες γίνεται η δημιουργία των σειρών και των στηλών. Στην συνέχεια ελέγχεται αν η πρώτη θέση του παίκτη είναι “μαρκαρισμένη” με το γράμμα P και στην συνέχεια γίνεται αυτόματα η εκτύπωση των άλλων στοιχείων του λαβυρίνθου.

```
void printLabyrinth(void)
{
    int i, j, k=0;
    leep(20000000);
    printf("Labyrinth: \n");
    for (i=0; i<H; i++)
    {
        for (j=0; j<W; j++)
        {
            if(k == PlayerPos)
                temp[j] = 'P';
            else
            {
                temp[j]=map[k];
            }
            k++;
        }
        temp[j+1]='\0';
        printf("%s\n", temp);
    }
    printf("\n");
}
```

- **makeMove**: η συνάρτηση είναι αναδρομική και με αυτόν τον τρόπο κάνει συνεχόμενα ελέγχους για το που ακριβώς βρίσκεται ο παίκτης στον λαβύρινθο. Συγκεκριμένα ελέγχει τις περιπτώσεις αν η κίνηση είναι προς τα πάνω: -W (όπου W το πλάτος του λαβυρίνθου), προς τα κάτω: +W, προς τα αριστερά: -1 και προς τα δεξιά +1. Σε κάθε περίπτωση γίνεται αντικατάσταση της θέσης που βρίσκεται ο χρήστης με το #. Τέλος γίνεται έλεγχος και για το αν το στοιχείο που βρίσκεται ο χρήστης είναι το @. Σε αυτή την περίπτωση αντικαθίστεται με το %.

```
int makeMove(int user_index)
{

    if(user_index<0 || user_index>=TotalElements)
    {
        return 0;
    }

    if(map[user_index] == '.')
    {   map[user_index] ='*';

        printLabyrinth();
        if(makeMove(user_index+1)== 1)
        {
            map[user_index]='#';
            return 1;
        }
        if(makeMove(user_index+W)==1)
        {   map[user_index]='#';
            return 1;
        }
        if(makeMove(user_index-1) == 1)
        {
            map[user_index]='#';
            return 1;
        }
        if(makeMove(user_index-W)==1)
        {
            map[user_index]='#';
            return 1;
        }
    }
    else if (map[user_index] =='@')
    {   map[user_index] = '%';

        return 1;
    }

    return 0;
}
```

- **Leep:** Η συνάρτηση leep έχει απλά την λειτουργικότητα που χρειάζεται για να υπάρξει η επιθυμητή καθυστέρηση για την αυτόματη λύση του λαβυρίνθου.

```
void leep(int time)
{
    int i;
    for(i=0;i<time;i++)
    {
    }
}
```

- **Main**: Στην main γίνεται όλη η διαδικασία κίνησης του παίκτη μέσα στον λαβύρινθο. Αρχικά αφού το πρόγραμμα καλωσορίσει τον παίκτη στο παιχνίδι και του εκτυπώσει τα κατάλληλα μηνύματα, που του μαθαίνουν πως να παίξει, ξεκινάνε με επαναληπτικές διαδικασίες οι διαδοχικές εισαγωγές των κινήσεων του παίκτη. Αφού γίνει ο έλεγχος οτι ο αριθμός στοιχείων είναι ακριβώς όσα έχει ο λαβύρινθος, ξεκινάει η διαδικασία χειροκίνητης επίλυσής του. Σε κάθε περίπτωση (W, S, A, D) γίνεται, με την χρήση **if**, αρχικά ο έλεγχος αν είναι έγκυρη η κίνηση, δηλαδή αν δεν υπάρχει τοίχος στο σημείο που θα πάει ο παίκτης. Αν υπάρχει, εκτυπώνεται το μήνυμα "**Ineligible Move**". Σε κάθε άλλη περίπτωση, μέσω της εντολής **index = index + X (όπου X: W (S), -W (W), 1 (D), -1 (A), ανάλογα με την κάθε περίπτωση)** κινείται ο παίκτης πάνω στις (.) και αφήνει ίχνος (*) από το μονοπάτι που δημιούργησε. Τέλος σε κάθε περίπτωση γίνεται ο έλεγχος αν το επόμενο στοιχείο ειναι @, οπότε και θα γίνει η αλλαγη σε % και εκτυπώνεται το μήνυμα "**Winner Winner Chicken Dinner**". Τέλος στην περίπτωση που ο παίκτης πατήσει **E** ξεκινάει η αυτόματη λύση του λαβυρίνθου μετακινώντας και έχοντας ως όρισμα της printLabyrinth το startX.

1

```
int main(){
    int choice;
    int index = startX;
    printf("HELLO !!! \nWelcome to my game!  \n");
    printf("Would you like to play?     \n");
    do {
        printf("->Press 1 for YES \n");
        printf("->Press 2 for NO \n");
        printf("==> Your choice: ");
        scanf("%d", &choice);
        if (choice == 2)
        {
            printf("\nGoodbye... :( \n");
            return 0;
        }
        else if (choice == 1)
        {
            printf("\nWELCOME AGAIN. \nLet's start ! \n");
            printf("\n");
            printf("-You have 4 eligible moves!- \n");
            printf("•PRESS -W- TO MOVE UP. \n");
            printf("•PRESS -S- TO MOVE DOWN. \n");
            printf("•PREES -A- TO MOVE LEFT. \n");
            printf("•PRESS -D- TO MOVE RIGHT. \n");
            printf("•PRESS -E- TO SHOW SOLUTION. \n");
            printf("\n");
            while (index!='@')
            {
                if(index<0 || index >= TotalElements)
                    return 0;

                PlayerPos = index;
                printLabyrinth();
                printf("Give me your move: ");
                scanf("%s", &userInput);
```

2

```
if(map[index] != '@')
{

    if (userInput == 'W')
    {
        if(map[index-W] == 'I')
        {
            printf("==>INELIGIBLE MOVE!\n==>TRY AGAIN !!! \n");
        }
        else {
            map[index]='*';
            index = index-W;
            if(map[index] == '@')
            {
                map[index] = '%';
                printLabyrinth();
                printf("\nWINNER WINNER CHICKEN DINNER\n");
                return 0;
            }else
            {
                map[index]='P';
            }
            printLabyrinth();
        }
    }

    if (userInput == 'S')
    {
        if (map[index+W] == 'I')
        {
            printf("\n==>INELIGIBLE MOVE!\n==>TRY AGAIN !!! \n");

        } else {
            map[index]='*';
            index = index+W;
            if(map[index] == '@')
            {
                map[index] = '%';
                printLabyrinth();
                printf("\nWINNER WINNER CHICKEN DINNER\n");
                return 0;
            }else
            {
                map[index]='P';
            }
            printLabyrinth();
        }
    }
```

```
            if (userInput == 'A')
            {
                if (map[index-1] == 'I')
                {
                    printf("\n==>INELIGIBLE MOVE! \n==>TRY AGAIN !!! \n");
                } else {
                    map[index]='*';
                    index = index-1;
                    if(map[index] == '@')
                    {
                        map[index] = '%';
                        printLabyrinth();
                        printf("\nWINNER WINNER CHICKEN DINNER\n");
                        return 0;
                    }else
                    {
                        map[index]='P';
                    }
                    printLabyrinth();
                }
            }

            if (userInput == 'D')
            {
                if (map[index+1] == 'I')
                {
                    printf("\n==>INELIGIBLE MOVE! \n==>TRY AGAIN !!! \n");
                } else {
                    map[index]='*';
                    index = index+1;
                    if(map[index] == '@')
                    {
                        map[index] = '%';
                        printLabyrinth();
                        printf("\nWINNER WINNER CHICKEN DINNER\n");
                        return 0;
                    }else
                    {
                        map[index]='P';
                    }
                    printLabyrinth();
                }
            }
            if (userInput == 'E')
            {
                index = startX;
                makeMove(index);
                printLabyrinth();
                printf("Pitty you have given up. It was very easy... \n");
                return 0;
            }
        }
    }


    }
}while (choice!=1 || choice != 2);

return 0;
```

Ακολουθεί η επίλυση του project σε Assembly

Εκτέλεση Άσκησης

Και στην Assembly χρησιμοποιήθηκαν οι ίδιες συναρτήσεις (**main**, **printLabyrinth**, **makeMove**, **leep**) με την C. Αρχικά δηλώθηκαν στο .data όλα τα μηνύματα που θα εκτυπώνονταν με την εντολή **addi $v0, $zero, 4**.

Να σημειωθεί οτι ο λαβύρινθος γράφτηκε ως maze: .byte και κάθε στοιχείο του δηλώθηκε σαν byte μέσα σε ‘’, όπως φαίνεται και παρακάτω.

```
maze: .byte 'I','.','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I'
      .byte 'I','.','.','.','.','.','I','.','.','.','.','.','I','.','.','.','.','.','.','.','I','.','I'
      .byte 'I','I','I','.','I','I','I','I','I','I','.','I','.','I','.','I','I','I','.','I','.','I'
      .byte 'I','.','I','.','.','.','.','.','.','I','.','.','.','I','.','.','I','.','.','.','.','.','I'
      .byte 'I','.','I','.','I','I','I','.','I','I','.','.','.','I','I','.','I','.','I','I','I'
      .byte 'I','.','.','.','.','I','.','.','.','I','I','I','.','I','.','.','.','I','.','.','.','.','I'
      .byte 'I','I','I','I','I','I','.','I','I','I','I','I','.','I','I','I','.','I','I','I','.','I'
      .byte 'I','.','.','.','.','.','.','.','.','.','.','.','.','.','.','.','.','I','.','I','.','.','.','I'
      .byte 'I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','.','I','.','I','I','I'
      .byte '@','.','.','.','.','.','.','.','.','.','.','.','.','.','.','.','.','I','.','.','I','I'
      .byte 'I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I','I'
      .align 1
```

Main:

Μέσα στην do_while_label ρωτάται ο χρήστης αν θέλει να παίξει το παιχνίδι επαναληπτικά μέχρι να δώσει έγκυρη τιμή (1 για ναι, 2 για οχι). Στην συνέχεια καλωσορίζεται παλι ο παίκτης και καθοδηγείται στις κινήσεις που μπορεί να κάνει. Πριν ξεκινήσει η επαναληπτική διαδικασία του Polling δηλώνεται οτι το startX είναι το γράμμα P. Στην keyboard_check εκτελείται η επαναληπτική διαδικασία του Polling. Αρχικά γίνεται load word στον Receiver Control. Συγκεκριμένα εκχωρείται στον καταχωρητή **$t0** το **0xffff0000** και στην συνέχεια μέσω της εντολής and περνάει μέσα από την μάσκα “**0x00000001**”. Στην συνέχεια γίνεται ο έλεγχος αν το **$t0** είναι 0, στην οποία περίπτωση θα ξαναγίνει αυτή η διαδικασία. Αυτό θα ολοκληρωθεί όταν ο χρήστης εισάγει κάποιο γράμμα και επομένως αποθηκεύεται στον **$s5** ο **0xffff004.** Ακολούθως καταχωρείται στον **$t0** ο “**0xffff0008**” και προστίθεται με το **“0x00000001”**. Τέλος αποθηκεύεται το γράμμα που θα εισάγει ο χαρακτήρας στο **0xffff000c** και μέσω μιας **addi $v0, $zero, 4** ακολοθούμενη από το **la $a0, print** και μια **syscall** εκτυπώνεται στην οθόνη.

```
keyboard_check:
  lw $t0, 0xffff0000
  add $t1, $zero, 0x00000001
  and $t2, $t0, $t1
  beq $t0, $zero, keyboard_check

  lw $s5, 0xffff0004

  lw $t0, 0xffff0008
  add $t0, $t0, $t1


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

  sw $s5, 0xffff000c

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

Στην συνέχεια αφού γίνει πάλι ο κατάλληλος έλεγχος αν τα στοιχεία είναι ακριβώς 231, με την χρήση εντολών: **slti** και **bne**, ελέγχεται αν το στοιχείο στο οποίο είναι ο χρήστης είναι το @.

Το πρόγραμμα συνεχίζει, ξεκινώντας τις διαδικασίες με τις οποίες ελέγχεται ποιο γράμμα εισήγαγε ο χρήστης. Για τα γράμματα W, S, A, D, η διαδικασία είναι η ίδια. Αυτό που αλλάζει είναι η αντιστοιχία του κάθε γράμματος με την δεκαδική τιμή του στον πίνακα ASCII. Συγκεκριμένα το W = 87, το S = 83, το A = 65 και το D = 68. Επίσης αυτό που αλλάζει είναι ο αριθμός που προσθαφαιρειται κάθε φορά για να γίνει η αντίστοιχη κίνηση. Συγκεκριμένα όταν είναι W αφαιρείται το 21, όταν είναι S προστίθεται το 21, όταν είναι A αφαιρείται το 1 και όταν είναι D προστίθεται το 1. Επομένως για κάθε περίπτωση αρχικά γίνεται έλεγχος αν μπορεί να γίνει η κίνηση, αν δεν συναντάει τοίχο δηλαδή. Ελέγχεται αν το maze($t1) είναι διάφορο του I (όπου $t1 η πράξη κάθε κίνησης (index-W, index+W, index-1, index+1). Αν η κίνηση είναι πάνω σε τοίχο εκτυπώνεται το αντίστοιχο μήνυμα και γίνεται jump πάλι πάνω να ξαναγίνει η διαδικασία Polling. Αλλιώς συνεχίζεται η διαδικασία εκτυπώνοντας τον χαρακτήρα (*), δηλαδη τον αριθμό 42 σε ASCII, και μετακινώντας τον χρήστη ανάλογα με το τι έχει επιλέξει. Μετά γίνεται ο έλεγχος αν με την κάθε κίνηση έχει φτάσει στο τέλος του λαβυρίνθου (χαρακτήρας @ (ASCII: 64)). Αν έχει γίνει αυτό εκτυπώνεται πάλι ο λαβύρινθος με % (ASCII: 37) μαζί με το μήνυμα “**Winner Winner Chicken Dinner**”. Μετά το πέρας αυτής της διαδικασίας και αν δεν είναι άκυρη η κίνηση και δεν έχει φτάσει ούτε στο τέλος του λαβυρίνθου μετακινείται και το ‘P’. Αρχικά φορτώνεται στον καταχωρητή $t3 το startX, στον $t4 το 80 (ASCII: P) και στον $t5 το 46 (ASCII: .). Στην συνέχεια καταχωρέιται ο $t5 στην θέση του startX του λαβυρίνθου και γίνεται η κατάλληλη πράξη με το startX ανάλογα με το κάθε γράμμα. Τέλος μετακινείται το ‘P’ και εκτυπώνεται ο λαβύρινθος με το ‘P’ στην νέα θέση του.

```
while_label_1:
  addi $t0, $zero, 64
  beq $s1, $t0, after_while

if_label_6_1:
  slti $t1, $s2, 0
  bne $t1, $zero, after_if_6

  slti $t3, $s2, 232
  beq $t3, $zero, after_if_6


  jal printLabyrinth

if_label_7:
  addi $t4, $s1, -1

  lb $t5, maze($t4)
  beq $t5, $t0, after_if_7


after_if_7:


if_label_8:                          # -- W CASE --
  addi $t9, $zero, 87
  bne $s5, $t9, after_if_8

if_label_8_1:
  addi $t1, $s1, -21
  lb $t2, maze($t1)
  bne $t2, 73, after_if_8_1

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

  j while_loop

after_if_8_1:
  lb $t1, maze($s1)
  addi $t1, $zero, 42
  addi $s1, $s1, -21

else_lbl_1:
  lb $t2, maze($s1)
  bne $t2, 64, after_lbl_1


  addi $t2, $zero, 37
  jal printLabyrinth
  addi $v0, $zero, 4
  la $a0, wwcd
  syscall

after_lbl_1:

  addi $t4, $zero, 80

  addi $t5, $zero, 46
  sb $t5, maze($s6)

  addi $s6, $s6, -21

  addi $v0, $zero, 1
  move $a0, $s6
  syscall

  sb $t4, maze($s6)

  jal printLabyrinth

  j while_loop
```

Ακολουθούν στιγμότυπα της main για την διαδικασία που περιγράφηκε.

```
after_if_8:
if_label_9:                          # -- S CASE --
  addi $t9, $zero, 83
  bne $s5, $t9, after_if_9
  addi $v0, $zero, 4
  la $a0, debug
  syscall
if_label_9_1:
  addi $t1, $s1, 21
  lb $t2, maze($t1)
  beq $t2, 73, after_if_9_1

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

  j while_loop

after_if_9_1:
  lb $t1, maze($s1)
  addi $t1, $zero, 42
  addi $s1, $s1, 21



else_lbl_2:
  lb $t2, maze($s1)
  bne $t2, 64, after_lbl_2

  addi $t2, $zero, 37
  #jal printLabyrinth

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

after_lbl_2:

addi $t4, $zero, 80

addi $t5, $zero, 46
sb $t5, maze($s6)

addi $s6, $s6, 21

addi $v0, $zero, 1
move $a0, $s6
syscall

sb $t4, maze($s6)

jal printLabyrinth

j while_loop
```

```
after_if_9:
if_label_10:                         # -- A CASE --
  addi $t9, $zero, 65
  bne $s5, $t9, after_if_10

if_label_10_1:
  addi $t1, $s1, -1
  lb $t2, maze($t1)
  beq $t2, 73, after_if_10_1

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

  j while_loop

after_if_10_1:
  lb $t1, maze($s1)
  addi $t1, $zero, 42
  addi $s1, $s1, -1

else_lbl_3:
  lb $t2, maze($s1)
  bne $t2, 64, after_lbl_3
  addi $t2, $zero, 37
  jal printLabyrinth
  addi $v0, $zero, 4
  la $a0, wwcd
  syscall

after_lbl_3:

addi $t4, $zero, 80

addi $t5, $zero, 46
sb $t5, maze($s6)

addi $s6, $s6, -1

addi $v0, $zero, 1
move $a0, $s6
syscall

sb $t4, maze($s6)

jal printLabyrinth
 j while_loop
```

```
after_if_10:
if_label_11:                         # -- D CASE --
  addi $t9, $zero, 68
  bne $s5, $t9, after_if_11

if_label_11_1:
  addi $t1, $s1, 1
  lb $t2, maze($t1)
  beq $t2, 73, after_if_11_1

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

  j while_loop

after_if_11_1:
  lb $t1, maze($s1)
  addi $t1, $zero, 42
  addi $s1, $s1, 1

else_lbl_4:
  lb $t2, maze($s1)
  bne $t2, 64, after_lbl_4
  addi $t2, $zero, 37
  jal printLabyrinth
  addi $v0, $zero, 4
  la $a0, wwcd
  syscall

after_lbl_4:
addi $t4, $zero, 80


addi $t5, $zero, 46
sb $t5, maze($s6)

addi $s6, $s6, 1

addi $v0, $zero, 1
move $a0, $s6
syscall

sb $t4, maze($s6)

jal printLabyrinth

j while_loop

after_if_11:
addi $t9, $zero, 69
bne $s5, $t9, after_if_12

add $s2, $zero, $s1

#jal makeMove
jal printLabyrinth
addi $v0, $zero, 4
la $a0, pitty
syscall
```

- Στην printLabyrinth έγινε η επαναληπτική διαδικασία δημιουργίας και εκτύπωσης του λαβυρίνθου. Αρχικά αρχικοποιούνται τα $t0, $t1, $t2 στο 0 (**addi $t1, $zero, 0**) και στην συνέχεια μέσα στην loop γίνονται οι επαναλήψεις για τις σειρες και τις στήλες του λαβυρίνθου. Συγκεκριμένα το **beq $t1, 11, after_loop** κάνει τον έλεγχο για τις σειρες, δηλαδή ελέγχει το πότε οι σειρές θα φτάσουν τις 11. Στην **loop_1** γίνεται ο αντίστοιχος έλεγχος για τις στήλες, 21 φορές και μετά εκτυπώνονται ένα, ένα τα byte του λαβυρίνθου. Τέλος αυξάνονται κατα 1 οι καταχωρητές $t0, $t1, $t2 και ξαναγίνεται η επανάληψη.

```
printLabyrinth:
  add $t0, $zero, $zero
  addi $t1, $zero, 0
  addi $t2, $zero, 0

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

  loop:
    beq $t1, 11, after_loop
    loop_1:
      beq $t2, 21, after_loop_1
      lb $t4, maze($t0)
      addi $v0, $zero, 11
      add $a0, $zero, $t4
      syscall

      add $t2, $t2, 1
      addi $t0, $t0, 1
      j loop_1

    after_loop_1:
      add $t2, $zero, $zero

      addi $t1, $t1, 1

      addi $v0, $zero, 4
      la $a0, nextline
      syscall
      j loop
  after_loop:
    jr $ra
```

Συμπεράσματα

Ολοκληρώνοντας το project του μαθήματος παρατηρούμε οτι υπήρχαν αρκετές δυσκολείες στην σύνταξη του κώδικα σε Assembly καθώς οι πολλές διαφορετικές επαναληπτικές διαδικασίες δημιουργούσαν προβλήματα σε όλη την διάρκεια της επίλυσης του προβλήματος. Ο στόχος όμως που ήταν να μπορούμε να προγραμματίζουμε σε Assembly επιτεύχθει και ολοκληρώνοντας αυτά τα εργαστήρια αποκτήσαμε γνώσεις που θα μας χρησιμεύσουν και σε επόμενα μαθήματα.