ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ
ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ ΓΙΑ ΤΟ ΜΑΘΗΜΑ: ΗΡΥ 201 ΨΗΦΙΑΚΟΙ ΥΠΟΛΟΓΙΣΤΕΣ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2017-2018

Εργαστήριο 5:

Maze Solver σε Assembly MIPS

1. Σκοπός του εργαστηρίου

Σκοπός του εργαστηρίου είναι η εξοικείωση του φοιτητή με την χρήση της στοίβας, των αναδρομικών συναρτήσεων και τις συμβάσεις των καταχωρητών σε Assembly MIPS.

2. Περιγραφή προβλήματος

Η διάσχιση λαβυρίνθου αποτελεί χαρακτηριστικό παράδειγμα αναδρομικής διαδικασίας. Το πρόγραμμα που καλείστε να ολοκληρώσετε σε C, και εν συνεχεία να υλοποιήσετε σε CLANG και assembly MIPS, θα πρέπει να εντοπίζει τη βέλτιστη διαδρομή από την είσοδο προς την έξοδο για οποιοδήποτε λαβύρινθο. Ο αρχικός χάρτης του λαβυρίνθου θα είναι αποθηκευμένος στο data segment του προγράμματος ως πίνακας χαρακτήρων. Θα δίδονται επίσης και 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 του εργαστηρίου αυτού, οι διάδρομοι είναι τα κελιά του πίνακα με περιεχόμενο '.'(τελεία), ενώ οι τοίχοι είναι τα κελιά με περιεχόμενο 'I'(κεφαλαίο i). Η επίλυση του λαβυρίνθου προϋποθέτει να κινήσεις σε γειτονικούς διαδρόμους (πάνω, κάτω, αριστερά ή δεξιά), και όχι σε τοίχους. Επίσης η μετακίνηση πραγματοποιείται κατά ένα στοιχείο κάθε φορά. Το πρόγραμμα θα πρέπει να κινείται στον λαβύρινθο μέχρι να φτάσει στην έξοδό του, στην οποία ο πίνακας περιέχει το χαρακτήρα '@'. Στο τέλος της εκτέλεσης του προγράμματος, θα πρέπει να εμφανίζεται το βέλτιστο μονοπάτι από την είσοδο του λαβυρίνθου προς την έξοδο.

```
Labyrinth:
I.IIIIIIIIIIIIIIIII
I....I....I.......I.I
III.IIIII.I.I.III.I.I
I.I.....I..I..I.....I
I.I.III.II...II.I.III
I...I...III.I...I...I
IIIII.IIIII.III.III.I
I............I.I...I
IIIIIIIIIIIII.I.III
@..............I..II
IIIIIIIIIIIIIIIIIII
```

Η διάσχιση του λαβυρίνθου πραγματοποιείται μέσω μίας αναδρομικής συνάρτησης. Κάθε κλήση της αναδρομικής συνάρτησης, το πρόγραμμα κάνει ένα βήμα σε διάδρομο του λαβυρίνθου. Για να εντοπίσουμε το βήμα που έγινε, η συνάρτηση αντικαθιστά το στοιχείο του πίνακα από χαρακτήρα '.'(τελεία) σε χαρακτήρα '*' (αστερίσκος) και συνέχεια εμφανίζει στην οθόνη τον λαβύρινθο.

```
Labyrinth:
I*IIIIIIIIIIIIIIIII
I****I....I.......I.I
III*IIIII.I.I.III.I.I
I.I*****I..I..I.....I
I.I.III*II...II.I.III
I...I...III.I...I...I
IIIII.IIIII.III.III.I
I.............I.I...I
IIIIIIIIIIIII.I.III
@.............I..II
IIIIIIIIIIIIIIIIIII
```

Κατά τον τερματισμό του προγράμματος, όταν η συνάρτηση επισκεφτεί το στοιχείο εξόδου ('@'), θα πρέπει να εμφανίζεται η λύση του λαβυρίνθου. Όσα κελιά διαδρόμου ανήκουν στο σωστό μονοπάτι, θα πρέπει αντί για '*' να περιέχουν '#', καθώς και το στοιχείο εξόδου από '@' θα πρέπει να γίνει '%'. Πριν το τέλος του προγράμματος, θα πρέπει να εμφανίζεται μία τελευταία φορά ο λαβύρινθος με το σωστό μονοπάτι.

```
Labyrinth:
I#IIIIIIIIIIIIIIIII
I###*I....I..#####I*I
III#IIIII.I.I#III#I*I
I.I#####I..I##I####**I
I.I.III#II.##II#I*III
I...I###III#I..#I***I
IIIII#IIIII#III#III*I
I....#######**I#I***I
IIIIIIIIIIIIII#I*III
%##############I**II
IIIIIIIIIIIIIIIIIII
```

3. Υλοποίηση του Εργαστηρίου σε C

Αρχικά, καλείστε να υλοποιήσετε σε C τον αλγόριθμο διάσχισης λαβυρίνθου. Τα δεδομένα που θα έχετε στη διάθεσή σας είναι:

- το πλάτος (W) και το ύψος(H) του λαβυρίνθου
- ο πίνακας χαρακτήρων που απεικονίζει το χάρτη του λαβυρίνθου (map)
- το index του σημείου εισαγωγής στο λαβύρινθο (startX)
- τον συνολικό αριθμό των στοιχείων του πινάκα (TotalElements)

Τα ακόλουθα δεδομένα εισόδου μπορούν να χρησιμοποιηθούν κατά τις πρώτες εκδόσεις του κωδικά σας, αν και είναι πολύ πιθανό να αξιολογηθεί η δουλειά σας με μεγαλύτερους χάρτες, κατά την εξέταση.

```
int W = 21;
int H = 11;
int startX = 1;
int TotalElements = 231;

char map[232] ="I.IIIIIIIIIIIIIIIII"
               "I....I....I.......I.I"
               "III.IIIII.I.I.III.I.I"
               "I.I.....I..I..I.....I"
               "I.I.III.II...II.I.III"
               "I...I...III.I...I...I"
               "IIIII.IIIII.III.III.I"
               "I.............I.I...I"
               "IIIIIIIIIIIIII.I.III"
               "@.............I..II"
               "IIIIIIIIIIIIIIIIIII";
```

Το πρόγραμμα σας σε C θα πρέπει να αποτελείται από τις ακόλουθες συναρτήσεις:

1. main η οποία σαν κύρια λειτουργικότητα θα έχει το να καλεί την αναδρομική συνάρτηση.
2. printLabyrinth η οποία θα εμφανίζει στην έξοδο το χάρτη του λαβυρίνθου στη μορφή που πρέπει.
3. makeMove η αναδρομική συνάρτηση διάσχισης του λαβυρίνθου.

Τα ακόλουθα κομμάτια κώδικα αποτελούν ένα παράδειγμα των κυριότερων συναρτήσεων, τα οποία μπορείτε να χρησιμοποιήσετε.

```
char temp[100]; //Global

void printLabyrinth (void){
    int i,j,k=0;
    usleep(200000);
    printf("Labyrinth:\n");
    for (i=0; i<H; i++){
          for(j=0; j<W; j++){
                temp[j]=map[k];
                k++;
          }
          temp[j+1]='\0';
          printf("%s\n", temp);
    }
}
```

```
int makeMove(int index){
    if(index<0 || index>=TotalElements)return 0;
    if(map[index]=='.'){
          map[index]='*';
          printLabyrinth();
          if(makeMove(index+1)==1){
               map[index]='#';
               return 1;
          }
          if(makeMove(index+W)==1){
               map[index]='#';
               return 1;
          }
          if(makeMove(index-1)==1){
               map[index]='#';
               return 1;
          }
          if(makeMove(index-W)==1){
               map[index]='#';
               return 1;
          }
    }else if (map[index]=='@'){
          map[index]='%';
          printLabyrinth();
          return 1;
    }
    return 0;
}
```

4. Υλοποίηση σε CLANG (30%)

Μετατρέψτε τις συναρτήσεις από C σε CLANG, λαμβάνοντας υπόψιν σας τις συμβάσεις των καταχωρητών. Σε αυτό το εργαστήριο, στη CLANG δεν θα έχουμε global μεταβλητές R0-R31, αλλά οι GLOBAL μεταβλητές που θα έχουμε είναι:

- μεταβλητή ZERO ίση με μηδέν.
- μεταβλητές V0-V1 που περνάμε τις τιμές επιστροφής των συναρτήσεων (κατά αντιστοιχία με τους καταχωρητές $v0-$v1 σε assembly MIPS).
- στις μεταβλητές A0-A4 περνάμε τις παραμέτρους των συναρτήσεων (κατά αντιστοιχία με τους καταχωρητές $a0-$a4 σε assembly MIPS).
- στις μεταβλητές T0-T9 και S0-S7 τοποθετούμε όλες τις υπόλοιπες τιμές (κατά αντιστοιχία με τους καταχωρητές $t0-$t9 και $s0-$s7 σε assembly MIPS).
- τη μεταβλητή RA η οποία δεν θα έχει κάποιο πρακτικό σκοπό, αλλά θα πρέπει να τη διαχειρίζεστε σε CLANG όπως τον $ra σε assembly.

Στη CLANG οι παράμετροι των συναρτήσεων και οι τιμές επιστροφής θα πρέπει να περνάνε στις αντίστοιχες μεταβλητές καθώς όλες οι συναρτήσεις θα πρέπει να είναι τύπου void foo(void). Εξαιρέσεις αποτελούν οι usleep και printf τι οποίες μπορείτε να καλέσετε ως έχουν.

Στη CLANG η στοίβα μπορεί να χρησιμοποιηθεί μέσω τοπικών μεταβλητών. Στην αρχή της συνάρτησης ορίζουμε ένα πίνακα με θέσεις όσες και οι τετράδες που θα δεσμεύαμε στη στοίβα του MIPS. Εκεί βάσει των συμβάσεων για τους Temporary και Saved

Registers τοποθετούμε τις τιμές των T0-T9 και S0-S7 αντίστοιχα. Τέλος, στη CLANG είναι υποχρεωτικό να αποθηκεύετε την τιμή του RA στη στοίβα σε κάθε κλήση συνάρτησης, για να είναι πιο εύκολη η μετάβαση σε assembly MIPS.
Αξίζει να σημειωθεί ότι στη CLANG θα πρέπει να ληφθούν υπόψιν οι συμβάσεις των Temporary και Saved καταχωρητών MIPS. Πρακτικά στον MIPS ισχύει:

- $s0-$s7 saved καταχωρητές. Σε CLANG έχουμε τις μεταβλητές S0-S7 οι οποίες τους προσομοιάζουν.
- $t0-$t9 temporary καταχωρητές. Σε CLANG έχουμε τις μεταβλητές T0-T9 οι οποίες τους προσομοιάζουν.

Πρακτικά μπορείτε να χρησιμοποιήσετε είτε saved είτε temporary καταχωρητές μέσα στο σώμα της κάθε συνάρτησης. Η διαφορά τους είναι ότι κάθε συνάρτηση που καλείται αδιαφορεί για το περιεχόμενο των temporary καταχωρητών. Μπορεί να τους γράφει άφοβα χωρίς να την ενδιαφέρει τι περιέχουν. Αυτό σημαίνει ότι αυτός που κάλεσε τη συνάρτηση τους καταχωρητές αυτούς τους έχει ξεγραμμένους.
Υπάρχει όμως και η περίπτωση που χρειάζεται όσες συναρτήσεις και αν καλέσουμε, οι τιμές των καταχωρητών να μην χάνονται. Αυτή την ιδιότητα έχουν οι saved καταχωρητές. Βέβαια αυτή η ιδιότητα έρχεται με κάποιο κόστος. Για να γράψει κάποια συνάρτηση αυτούς τους καταχωρητές, πρέπει να αποθηκεύσει στη στοίβα τις παλιές τους τιμές στον πρόλογο και να τις επαναφέρει στον επίλογο. Οι καταχωρητές που έχουν αυτή την ιδιότητα (την ευθύνη της αποθήκευσής τους έχει η συνάρτηση που κλήθηκε) λέγονται callee saved.
Υπάρχει όμως και η περίπτωση που οι καταχωρητές αυτοί δε μας φτάνουν, και θέλουμε να διατηρήσουμε περισσότερες τιμές ανάμεσα στις κλήσεις των συναρτήσεων, στους temporary καταχωρητές. Γνωρίζουμε ότι κάθε νέα συνάρτηση αδιαφορεί για τις παλιές τιμές των temporary καταχωρητών. Για αυτό το λόγο, ο καλών πρέπει να αποθηκεύσει τις τιμές αυτές στη στοίβα πρίν καλέσει τη νέα συνάρτηση, και να τις επαναφέρει από τη στοίβα μόλις η νέα συνάρτηση επιστρέψει. Οι καταχωρητές που έχουν αυτή την ιδιότητα (την ευθύνη της αποθήκευσής τους έχει η συνάρτηση που καλεί) λέγονται caller saved.
Παρακάτω παρατίθεται ένα παράδειγμα σε CLANG για την εφαρμογή caller και callee seved καταχωρητών.

```
void foo(void){
   int temp[4];   //we allocate stack space for 16 bytes
   temp[0] = S0; //save $s0
   temp[1] = S1; //save $s1
   /*The code of the function foo body is placed here*/
   temp[2]= T0;
   temp[3]= RA;
   A0 = 5;
   foo2();
   RA = temp[3]; //restore $ra
   T0 = temp[2]; //restore $t0
   /*The code of the function foo body is placed here*/
   S1=temp[1];   // restore $s1
   S0=temp[0];   //restore $s0
   //IN ASSEMBLY WE NEED TO RELEASE STACK SPACE IN THIS LINE
   V0 = T0;       //Return value
}
```

Προειδοποίηση: Η σωστή υλοποίηση σε CLANG είναι κρίσιμη, καθώς η μετατροπή από μία σωστή CLANG σε assembly MIPS είναι μια πολύ απλή διαδικασία που γενικά δουλεύει εύκολα. Κατά την εξέταση του εργαστηρίου, η μη ακριβής αντιστοίχιση CLANG με Assembly MIPS, θα έχει αρνητικό αντίκτυπο στη συνολική βαθμολογία.

5. Υλοποίηση σε Assembly (70%)

1. Μετατρέψτε το πρόγραμμά σας από CLANG σε Assembly χωρίς την κλήση *usleep(200000);* (60%).
2. Υλοποιήστε αντίστοιχη λειτουργικότητα με την κλήση *usleep(200000);* σε Assembly (10%).

   Η υλοποίηση του υποερωτήματος 2 προϋποθέτει την επιτυχή ολοκλήρωση του 1. Σε αυτό το στάδιο θα πρέπει να δημιουργήσετε μία απλή συνάρτηση που απλά να καθυστερεί την εκτέλεση, ώστε η εμφάνιση του λαβυρίνθου σε κάθε βήμα της αναδρομής να γίνεται πιο ομαλά. Η νέα συνάρτηση που θα φτιάξετε μπορεί να ξεκινάει ένα καταχωρητή από το μηδέν και μέσα σε ένα loop να αυξάνεται η τιμή του καταχωρητή κατά 1. Όταν φτάσει ο καταχωρητής σε κάποια τιμή που θα ορίσετε, η συνάρτηση μπορεί να επιστρέψει.

6. Bonus Bitwise Operations (15%)

Κατά την υλοποίηση του bonus καλείστε να αλλάξετε τον κώδικα σε C, CLANG και Assembly, ώστε ο χάρτης να είναι σε compact μορφή. Πιο συγκεκριμένα, αντί για να έχουμε αντιστοίχιση κελιού πίνακα με τοίχο ή διάδρομο, θα υπάρχει αντιστοίχιση σε bit.
Αρχικά σε αυτή την έκδοση θα υπάρχει ο περιορισμός ότι το πλάτος του χάρτη θα είναι πολλαπλάσιο του 8. Εδώ, ο διάδρομος θα αντιστοιχιστεί με λογικό 0 και ο τοίχος με λογικό 1. Η εκτύπωση θα πρέπει να γίνεται αντίστοιχα με χαρακτήρες μηδέν και ένα. Τέλος, θα υπάρχει μεταβλητή endX η οποία θα σημειώνει την έξοδο του λαβυρίνθου, κατά αντιστοιχία με τη startX.
Κατά την υλοποίηση του Bonus, ο αλγόριθμος θα παραμείνει ως έχει, αλλά η ουσιαστική διαφορά είναι ότι για τον εντοπισμό της θέσεως ενός στοιχείου, θα πρέπει να λάβετε υπόψιν σας την τοποθέτηση των bit μέσα στον πίνακα. Για παράδειγμα το κελί 0 της πρώτης γραμμής του αρχικού λαβυρίνθου αντιστοιχεί με το bit 7 του πρώτου χαρακτήρα του binary πίνακα. Αντίστοιχα το κελί 7 της πρώτης γραμμής του αρχικού λαβυρίνθου αντιστοιχεί με το bit 0 του πρώτου χαρακτήρα του binary πίνακα.
Τέλος, για να μπορέσετε να κρατάτε τα βήματα της διάνυσης του λαβυρίνθου, μπορείτε να ορίσετε ένα νεό “binary” πίνακα στον οποίο το θα απεικονίζετε τις κινήσεις. Το μέγεθος αυτού του πίνακα θα ισούται με το μέγεθος του χάρτη για το bonus. Εκεί, οι διάδρομοι που έχουμε επισκεφτεί θα είναι σημειωμένοι με το λογικό 1 και όλα τα υπόλοιπα στοιχεία θα έχουν την τιμή 0. Γενικά το bonus είναι μία καλή ευκαιρία εξάσκησης σε bitwise operations.
Ακολουθεί ένα παράδειγμα δεδομένων εισόδου στην αρχική τους μορφή και στη συνέχεια σε binary.

Initial:

```
int W = 24;
int H = 16;
int startX = 24;
int TotalElements = 384;
char map[385] = "IIIIIIIIIIIIIIIIIIIIIIII"
                ".....I....I.......I....I"
                "III.IIIII.I.I.IIIII.II.I"
                "I.I.....I..I........II.I"
                "I.I.III.II...II.I.IIII.I"
                "I...I...III.I...I...II.I"
                "IIIII.IIIII.III.III.II.I"
                "I............I.I...II.I"
                "IIIIIIIIIIIIII.I.II...I"
                "I..............I..I.I.I"
                "IIIII.I.II.IIIIII.II.I.I"
                "I.II..II.III.II.I....I.I"
                "I...II........I...I.II.I"
                "I.I.II.II.III.IIIIIIII.I"
                "I.I....II...I.........I"
                "I@IIIIIIIIIIIIIIIIIIIII";
```

Binary:

```
Initial:
int W = 24;
int H = 16;
int startX = 24;
int endX = 361;
int TotalElements = 384;
char map[48] ={     0xFF,0xFF,0xFF,     //111111111111111111111111
                    0x04,0x20,0x21,     //000001000010000000100001
                    0xEF,0xAB,0xED,     //111011111010101111101101
                    0xA0,0x90,0x0D,     //101000001001000000001101
                    0xAE,0xC6,0xBD,     //101011101100011010111101
                    0x88,0xE8,0x8D,     //100010001110100010001101
                    0xFB,0xEE,0xED,     //111110111110111011101101
                    0x80,0x02,0x8D,     //100000000000001010001101
                    0xFF,0xFE,0xB1,     //111111111111111010110001
                    0x80,0x00,0x95,     //100000000000000010010101
                    0xFA,0xDF,0xB5,     //111110101101111110110101
                    0xB3,0x76,0x85,     //101100110111011010000101
                    0x8C,0x02,0x2D,     //100011000000001000101101
                    0xAD,0xBB,0xFD,     //101011011011101111111101
                    0xA1,0x88,0x01,     //101000011000100000000001
                    0xBF,0xFF,0xFF};    //101111111111111111111111
```

Παραδοτέα

1. Σύντομη αναφορά σχετικά με σημαντικά σημεία του κώδικα σας.
2. Προβλήματα που αντιμετωπίσατε για μελλοντική βελτίωση του εργαστηρίου.
3. Κατά την παράδοση της αναφοράς καλείστε να παραδώσετε ένα συμπιεσμένο αρχείο (*.zip, *.rar, *.7z, *.tar.bz2...) που θα περιέχει το pdf με το κείμενο της αναφοράς, σύν τα αρχεία *.c, *.asm του κώδικά σας.

ΚΑΛΗ ΕΠΙΤΥΧΙΑ