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

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2018-2019

Project

Maze σε Assembly MIPS

1. Σκοπός του εργαστηρίου

Σκοπός του project είναι η εξοικείωση του φοιτητή με τη χρήση της στοίβας, των αναδρομικών συναρτήσεων, τις συμβάσεις των καταχωρητών, το polling και τα interrupts 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 assembly MIPS, θα πρέπει να παρέχει δύο λειτουργικότητες. Ο χρήσης θα μπορέσει να διασχίσει το λαβύρινθο με στόχο την έξοδο από αυτόν. **Η πρώτη λειτουργικότητα του project είναι η υλοποίηση του παιχνιδιού του λαβύρινθου.** Εάν πατηθεί το κατάλληλο πλήκτρο, θα πρέπει να 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. 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 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. Το πρόγραμμα θα πρέπει να κινείται στον λαβύρινθο μέχρι να φτάσει στην έξοδό του, στην οποία ο πίνακας περιέχει το χαρακτήρα '@'.

```
Labyrinth:
IPIIIIIIIIIIIIIIIII
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
```

Ο χρήστης θα πρέπει να κινείται μέσα στο λαβύρινθο χρησιμοποιώντας τα πλήκτρα “W”, “S”, “A”, “D”, ώστε να κινηθεί πάνω, κάτω, αριστερά και δεξιά σε διαδρόμους. Η τρέχουσα θέση του χρήστη περιγράφεται από το γράμμα 'P' στο λαβύρινθο. Οι κινήσεις του χρήστη επάνω σε τοίχο δεν επιτρέπονται. Όταν ο παίκτης φτάσει στη έξοδο του λαβυρίνθου θα πρέπει να του εμφανίζει το μήνυμα “Winner Winner Chicken Dinner!”. Εναλλακτικά, όταν ο χρήστης πατάει το πλήκτρο “E” από το πληκτρολόγιο, θα πρέπει να εμφανίζεται το βέλτιστο μονοπάτι.

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. Η θέση του παίκτη θέλουμε να παραμείνει με το γράμμα 'P'.

```
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.P.I...I
IIIII.IIIII.III.III.I
I.............I.I...I
IIIIIIIIIIIII.I.III
@..............I..II
IIIIIIIIIIIIIIIIIII
```

Κατά την εύρεση του βέλτιστου μονοπατιού, 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 '%'. Και πάλι η θέση του παίκτη παραμένει ώς έχει. 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.P#I***I
IIIII#IIIII#III#III*I
I....#######**I#I***I
IIIIIIIIIIIII#I*III
%##############I**II
IIIIIIIIIIIIIIIIIII
```

3. Υλοποίηση του Εργαστηρίου σε C (10%)

Αρχικά, καλείστε να υλοποιήσετε σε C τον αλγόριθμο διάσχισης λαβυρίνθου. Τα δεδομένα που 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 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 κινεί τον παίκτη και 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 επίλυσης του λαβυρίνθου.

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

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

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++){
                        if(k==playerPos)
                                temp[j]='P';
                        else
                        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 Assembly (40%)

1. Μετατρέψτε το πρόγραμμά σας από C σε Assembly χωρίς την κλήση *usleep(200000);* (35%). Το πλήκτρο επιλογής του χρήστη θα γίνεται με syscall.
2. Υλοποιήστε αντίστοιχη λειτουργικότητα με την κλήση *usleep(200000);* σε Assembly (5%).

   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.

5. Polling (25%)

Η χρήση απλών εντολών ανάγνωσης/εγγραφής συγκεκριμένων θέσεων μνήμης είναι μία

τεχνική επικοινωνίας του επεξεργαστή με τις περιφερειακές συσκευές εισόδου/εξόδου. οι καταχωρητές ελέγχου αλλά και μεταφοράς που ανήκουν σε ένα περιφερειακό αποκτούν από μια διεύθυνση μνήμης. Οι απλές συσκευές, όπως το πληκτρολόγιο και η κονσόλα, χρειάζονται μόνο δύο τέτοιες διευθύνσεις, μία για την μεταφορά των δεδομένων (Data) και μια για τον έλεγχο της συσκευής και την παρακολούθηση των μεταφορών (Control).
Στον SPIM, οι καταχωρητές Control και Data έχουν πλάτος 32 bits αν και μόνο τα 8 λιγότερο σημαντικά bits του καταχωρητή Data και τα 2 λιγότερο σημαντικά bits του καταχωρητή Control χρησιμοποιούνται στην πραγματικότητα. Συγκεκριμένα, η χρήση των 8 bits του καταχωρητή Data είναι προφανής. Σε αυτά τα bits αποθηκεύεται κάθε φορά ο χαρακτήρας που μεταφέρεται σε ASCII μορφή. Αντίθετα, η λειτουργία του καταχωρητή Control είναι πιο σύνθετη. Το λιγότερο σημαντικό bit (bit 0) του καταχωρητή Control είναι το Ready bit. Αν αυτό έχει την τιμή Ι σημαίνει ότι το αντίστοιχο περιφερειακό είναι έτοιμο για χρήση διαφορετικά η συσκευή δεν είναι ακόμη έτοιμη για λειτουργία (π.χ. δεν υπάρχει χαρακτήρας για ανάγνωση από το πληκτρολόγιο ή η κονσόλα είναι απασχολημένη και δεν μπορεί να δεχτεί νέο χαρακτήρα). Όταν το συγκεκριμένο bit γίνει 1, τότε για την περίπτωση του πληκτρολογίου μπορούμε να διαβάσουμε τον χαρακτήρα που εισήγαγε ο χρήστης από την αντίστοιχη διεύθυνση Data, ενώ για την περίπτωση της κονσόλας μπορούμε να γράψουμε στην διεύθυνση Data τον επόμενο χαρακτήρα που θα εμφανιστεί στην οθόνη.
Το αμέσως πιο σημαντικό bit (bit 1) του καταχωρητή Control είναι το Interrupt Enable. Αν ο χρήστης επιθυμεί την ενεργοποίηση των αιτήσεων διακοπής από τη συγκεκριμένη συσκευή, θα πρέπει να θέσει το bit 1 του Control καταχωρητή ίσο με 1. Η ενεργοποίηση των αιτήσεων διακοπής αφορά μόνο το τεταρτο μέρος του εργαστηρίου.
Ο επόμενος πίνακας δίνει τις διευθύνσεις των 4 καταχωρητών των δύο αυτών συσκευών στο SPIM.

Όνομα Καταχωρητη	**Διεύθυνση**
Receiver Control	0xFFFF0000
Receiver Data	0xFFFF0004
Transmitter Control	0xFFFF0008
Transmitter Data	0xFFFF000C

Η τεχνική polling στηρίζεται στον διαρκή έλεγχο των περιφερειακών συσκευών έως ότου αυτές είναι έτοιμες για την πράξη που θέλουμε. Για παράδειγμα, εάν θέλουμε να γράψουμε ένα χαρακτήρα στην κονσόλα, θα πρέπει να διαβάζουμε επαναληπτικά τον καταχωρητή Transmitter control από την διεύθυνση 0xffff0008 και να ελέγχουμε τη τιμή του λιγότερου σημαντικού bit (Ready bit). Εάν αυτό είναι 0, τότε επαναλαμβάνουμε τη διαδικασία ανάγνωσης του παραπάνω καταχωρητή. Όταν η τιμή του συγκεκριμένου bit γίνει 1, τότε γράφουμε στον καταχωρητή Transmitter Data, που βρίσκεται στη διεύθυνση 0xffff000c της μνήμης, το byte (χαρακτήρα) που θέλουμε να εμφανίσουμε στην κονσόλα. Αντίστοιχη είναι η εργασία που απαιτείται για την ανάγνωση ενός χαρακτήρα από τον χρήστη με τη χρήση των κατάλληλων θέσεων μνήμης.
Θα πρέπει υλοποιήσετε τη συναρτηση read_ch η οποία, χρησιμοποιώντας την τεχνική polling που περιγράφηκε παραπάνω, θα διαβάζει από το πληκτρολόγιο ένα χαρακτήρα. Μόλις διαβαστεί ο χαρακτήρας αυτός, θα πρέπει ή να κινείται ο παίκτης στον λαβύρινθο, ή να εμφανίζεται η λύση του λαβύρινθου.
Παρατηρηση: Πρέπει να ενεργοποιήσετε τα «memory mapped IO» στον SPIM. Συγκεκριμένα, από το μενού ενεργοποιήστε την επιλογή «Mapped I/O», εάν δεν είναι ήδη ενεργοποιημένη.

6. Interrupts (25%)

Ένας διαφορετικός τρόπος επικοινωνίας επεξεργαστή-περιφερειακών συσκευών είναι η χρήση διακοπών (interrupt). Με τη συγκεκριμένη μέθοδο μια περιφερειακή συσκευή σηματοδοτεί στον επεξεργαστή (χρησιμοποιώντας τον δίαυλο (bus) που τους ενώνει) ότι είναι έτοιμη για μεταφορά δεδομένων και “χρειάζεται την προσοχή του”. Ο επεξεργαστής τότε διακόπτει την κανονική εκτέλεση του προγράμματος και αρχίζει εμβόλιμα την εκτέλεση του interrupt handler. Ο interrupt handler είναι μία «ρουτίνα», η οποία βρίσκει το λόγο για τον οποίο έγινε η διακοπή, ποια συσκευή είναι αυτή που χρειάζεται προσοχή, και μετά αναλαμβάνει την εξυπηρέτησή της.
Επειδή η εκτέλεση του interrupt handler είναι εμβόλιμη στην εκτέλεση του κανονικού προγράμματος, οι καταχωρητές του επεξεργαστή μπορεί να χρησιμοποιούνται από το πρόγραμμα, και συνεπώς οι συμβάσεις κλήσεις υπορουτίνας δεν ισχύουν για τον interrupt handler. Για την διευκόλυνση και επιτάχυνση του όμως ο MIPS δεσμεύει τους καταχωρητές $k0 και $k1 για χρήση από τον πυρήνα (kernel) και κατ' επέκταση και από τον interrupt handler. Για να μπορέσει να χρησιμοποιήσει άλλους καταχωρητές, ο interrupt handler πρέπει να τους σώσει σε ασφαλές μέρος στη μνήμη. Ο επεξεργαστής MIPS μπορεί να ελέγξει ποιες συσκευές επιτρέπεται να δημιουργήσουν interrupt μέσω του καταχωρητή Status (Coprocessor0, καταχωρητής 12). Στον καταχωρητή αυτό, το bit 11 ενεργοποιεί τις διακοπές για το πληκτρολόγιο, και το bit 0 είναι η συνολική ενεργοποίηση των διακοπών για τον επεξεργαστή. Επιπλέον, η κάθε συσκευή έχει δικό της εσωτερικό έλεγχο (μέσω του αντίστοιχου καταχωρητή control) για το αν θα ζητάει interrupt ή αν θα επικοινωνεί με την τεχνική polling. Όπως και στην τεχνική polling, όταν μια συσκευή ζητήσει interrupt, ο καταχωρητής status της συσκευής δείχνει ότι η συσκευή είναι έτοιμη για μεταφορά. Μόλις διαβαστεί (στην περίπτωση εισόδου) ο καταχωρητής data, τα δεδομένα καταναλώνονται και η συσκευή παύει να ζητάει interrupt και ο καταχωρητής control δείχνει ότι η συσκευή δεν είναι έτοιμη για μεταφορά.
Η επικοινωνία του interrupt handler με τον κώδικα του χρήστη θα γίνεται μέσω δύο θέσεων μνήμης, cflag και cdata. Η θέση μνήμης cflag θα χρησιμοποιηθεί για να "δείχνει” στο πρόγραμμα πότε ο interrupt handler έδωσε ένα νέο χαρακτήρα. Ο νέος χαρακτήρας θ’ αποθηκεύεται στη θέση μνήμης cdata. Το πρόγραμμα που πρέπει να φτιάξετε θα πρέπει να λειτουργεί όπως περιγράφεται παρακάτω.
Αρχικά, ενεργοποιήστε τα interrupts στον επεξεργαστή και στο πληκτρολόγιο. Πρέπει να ενεργοποιήσετε όλα τα σημεία ελέγχου που αναφέρονται παραπάνω. Στη συνέχεια, φτιάξτε ένα loop το οποίο θα ελέγχετε την τιμή της θέσης μνήμης cflag. Για όσο η τιμή αυτή είναι 0, θα συνεχίζεται η εκτέλεση του loop. Η αρχικοποίηση του cflag θα είναι 0 και θα πρέπει να μηδενίζεται κάθε φορά που καταναλώνουμε τον εισερχόμενο χαρακτήρα. Όταν η τιμή στο cflag βρεθεί διαφορετική από το μηδέν, θα την μηδενίζετε αφού πήρατε το "μήνυμα” ότι υπάρχει εισερχόμενος χαρακτήρας.
Ο νέος χαρακτήρας θα είναι αποθηκευμένος στη μεταβλητή cdata. Συγκεκριμένα, κατά τη διάρκεια της εκτέλεσης του συνεχούς loop όταν ο χρήστης πατήσει ένα χαρακτήρα θα ενεργοποιηθεί ένα interrupt και θα τρέξει ο κώδικας του interrupt handler (exceptions.s). Εκεί θα βάλετε τον δικό σας κώδικα που θα κάνει την εκχωρηση του εισερχόμενου χαρακτήρα στο cdata. Επίσης, εκεί θα πρέπει να αποθηκεύετε την τιμή 1 στο cflag. Μόλις εκτελεστεί ο κώδικας που βρίσκεται στο αρχείο exceptions.s, η ροή του προγράμματος συνεχίζει από το σημείο που είχε σταματήσει όταν ήρθε το interrupt.
Εξετάστε και κατανοήστε τον κώδικα στο αρχείο exceptions.s που δίδεται με το SPIM. Δημιουργήστε ένα αντίγραφο αυτού του αρχείου ώστε να εισάγετε στο σωστό σημείο τον δικό σας κώδικα. Ο κώδικας αυτός θα «γεμίζει» τις μεταβλητές cflag και cdata (με τι τιμές;) όταν έρθει το interrupt από το πληκτρολόγιο. Ενεργοποιήστε τη χρήση του αλλαγμένου αρχείου στον SPIM από το μενού Simulator -> Settings και επιλέξετε το σωστό αρχείο στην επιλογή «Exception». Τέλος, δώστε ιδιαίτερη προσοχή στο ποιούς καταχωρητές θα χρησιμοποιήσετε εντός του interrupt handler.

Θα πρέπει στη main να ελέγχετε τις μεταβλητές cflag και cdata, ώστε να εντοπιστεί και να

διαβαστεί κάποιος νέος χαρακτήρας. Μόλις διαβαστεί ο χαρακτήρας αυτός, θα πρέπει ή να κινείται ο παίκτης στον λαβύρινθο, ή να εμφανίζεται η λύση του λαβύρινθου.

7. Bonus Bitwise Operations (10%)

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 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. 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....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,     //111011110101011111101101
                    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, μαζί τα αρχεία *.c, *.asm του κώδικά σας.

Απαντήσεις σε πιθανές ερωτήσεις

Γενικά για να δείτε εάν θα τοποθετήσετε μεταβλητές σε saved καταχωρητές, θα πρέπει να δείτε ποιες μεταβλητές χρειάζεται να διατηρούν τις τιμές τους μετά την κλήση συναρτήσεων. Επίσης, κατά τη μετατροπή του χάρτη σε assembly, θα μπορούσατε να χρησιμοποιήσετε το .ascii directive (μιλάμε για την γενική περίπτωση και όχι το bonus).

Ένα σύνηθες λάθος το οποίο συμβαίνει αφορά τη χρήση της στοίβας σε CLANG / Assembly. Για το λόγο αυτό θα πρέπει να γράφουμε στη στοίβα τον `$ra` πρίν την κλήση της αναδρομικής συνάρτησης και να τον επαναφέρουμε αμέσως μετά την επιστροφή της. Επίσης αναφορικά με τη χρήση των SAVED καταχωρητών, η αποθήκευση στη στοίβα θα πρέπει να γίνεται κατά τον πρόλογο της συνάρτησης (στην αρχή). Το σημαντικότερο όμως είναι ότι θα πρέπει να επαναφέρουμε τους SAVED καταχωρητές ΠΡΙΝ ΕΠΙΣΤΡΕΨΕΙ Η ΣΥΝΑΡΤΗΣΗ, δηλαδη πρίν από ΚΑΘΕ return / jr $ra. Ένα σύνηθες ΛΑΘΟΣ είναι η επαναφορά των SAVED καταχωρητών μόνο στο τελευταίο return και όχι στα εμφωλευμένα.

Γενικά, η usleep αυτό που κάνει είναι να προσθέσει καθυστέρηση ώστε να εμφανίζονται ομοιόμορφα (σιγά-σιγά) τα αποτελέσματα στη οθόνη. Αν δε χρησιμοποιήσετε την usleep, απλά το πρόγραμμα εμφανίζει το λαβύρινθο πολύ γρήγορα. Είναι κομμάτι του βασικού κορμού του εργαστηρίου η υλοποίηση κάποιας σχετικής συνάρτησης. Επίσης στην υλοποίηση της usleep σε assembly μπορείτε να θέσετε όση καθυστέρηση θέλετε αρκεί να εμφανίζονται τα αποτελέσματα βήμα-βήμα.

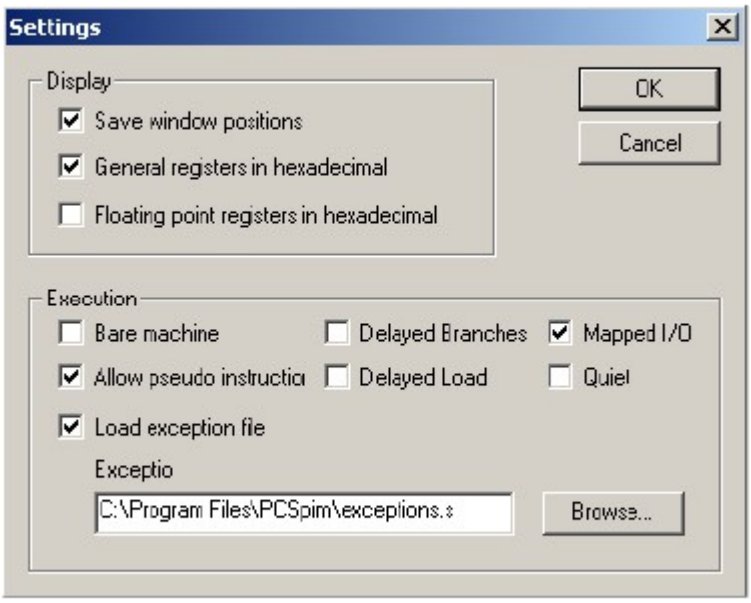
Όταν δεν υπάρχει protection μπορείτε να γράφετε και να διαβάζετε όποια διεύθυνση θέλετε από τη μνήμη. Επειδή τα αλφαριθμητικά (ορισμένα με .asciiz directive) είναι στη μνήμη, μπορείτε να τα γράφετε και να τα διαβάζετε. Πρακτικά το directive .asciiz φτιαχνει ένα πίνακα και τον γεμίζει με το αλφαριθμητικό που βλέπει, όπως στη C έχουμε char temp[ ] = "Whatever";

Η εκφώνηση αναφέρει: "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". Εδώ, δεδομένου ότι κοιτάζουμε τις δυαδικές τιμές του πίνακα, θα πρέπει να καθορίσουμε κάποιο τρόπο που το index του πίνακα των προηγούμενων ερωτημάτων, θα πρέπει να να αντιστοιχιστεί με τη διεύθυνση συγκεκριμένου bit συγκεκριμένου στοιχείου του νέου πίνακα. Η αντιστοίχιση αυτή θα γίνεται ως εξής:
Το στοιχείο 0 του παλιού πίνακα (χάρτη λαβυρίνθου), αντιστοιχεί με το bit 7 του στοιχείου 0 του νέου πίνακα. Το στοιχείο 1 του παλιού πίνακα, αντιστοιχεί με το bit 6 του στοιχείου 0 του νέου πίνακα... Το στοιχείο 7 του παλιού πίνακα, αντιστοιχεί με το bit 0 του στοιχείου 0 του νέου πίνακα. Το στοιχείο 8 του παλιού πίνακα, αντιστοιχεί με το bit 7 του στοιχείου 1 του νέου πίνακα... κλπ.

Γενικά, ο πίνακας του δυαδικού χάρτη για αρχή σας δίδεται στην εκφώνηση. Θα μπορείτε να έχετε ένα βοηθητικό binary πίνακα αντίστοιχο με το χάρτη του λαβυρίνθου (ίδιο μέγεθος) ο οποίος θα κρατάει τις θέσεις από τις οποίες περάσατε. Εκεί, με λογικό 1 θα περιγράφετε τις θέσεις που περάσατε και με λογικό 0 τις θέσεις που δεν έχετε περάσει. Κάθε φορά για κάθε θέση που θα θέλετε να μεταβείτε θα πρέπει να ελέγχετε τα αντίστοιχα bit ώστε το σημείο του χάρτη να μην είναι τοίχος και να μην το το έχετε περάσει. Άρα τα αντίστοιχα bit και στους δύο πίνακες αν είναι 0, τότε το σημείο είναι προσβάσιμο. Στο τέλος, που θα έχει βρεθεί η βέλτιστη διαδρομή, κατά την επιστροφή της αναδρομής μπορείτε να χρησιμοποιήσετε τον πίνακα του λαβυρίνθου ακολούθως. Όταν επιστρέψει η συνάρτηση επιλυσης, τοποθετούμε 1 στα στοιχεία της βέλτιστης διαδρομής. Άρα στο τέλος του προγράμματος ο χάρτης του λαβυρίνθου θα περιέχει 1 εκεί που είναι είτε τοίχος, είτε είναι τμήμα της βέλτιστης διαδρομής. Κατα την τελική εμφάνιση, εάν ένα στοιχείο είναι 1 μόνο στον πίνακα των διαδρομών, τότε εμφανίζετε '*', εάν το στοιχείο καί στους δύο πίνακες είναι 1 εμφανίζετε '#', εάν πάλι είναι 1 μόνο στον πίνακα του λαβυρίνθου, εμφανίζετε 'I', αλλιώς όταν είναι 0 παντού εμφανίζετε '.' . Εναλλακτικά, εάν η επαναχρησιμοποίηση του πίνακα του λαβυρίνθου θεωρείτε ότι είναι πολύ πολύπλοκη, μπορείτε να φτιάξετε ένα τρίτο binary πίνακα μόνο για τη βέλτιστη διαδρομή. Γενικά πάντως το να χρησιμοποιήσετε 2 πίνακες αντί για 3, είναι προτιμότερο.

ΚΑΛΗ ΕΠΙΤΥΧΙΑ

Μενού ρυθμίσεων SPIM για ενεργοποίηση Memory Mapped I/O και καθορισμό αρχείου interrupt handler.



Διευθύνσεις Memory Mapped I/O