

Οργάνωση και Σχεδίαση Υπολογιστών (ECE 219)
Χειμερινό Εξάμηνο 2020-2021

Εργαστήριο 1

8% του συνολικού εργαστηριακού βαθμού

Στόχοι του εργαστηρίου

- Εντολές εισόδου-εξόδου
- Χρήση συνθηκών σε δομές επιλογής
- Χρήση συνθηκών σε δομές επανάληψης
- Εντολές προσπέλασης της μνήμης
- Χρήση πινάκων

Εντολές εισόδου-εξόδου

Εντολές για είσοδο ακεραίων αριθμών από την κονσόλα:

li \$v0, 5	# κωδικός 5 από τα syscall στον \$v0
syscall	
move \$t0, \$v0	# καταχώριση της τιμής που εισάγαμε, στον # register \$t0

Εντολές για εμφάνιση ακεραίου στην κονσόλα:

li \$v0, 1	# pseudo-instruction για addi \$v0, \$0, 1
	# κωδικός 1 από τα syscall στον \$v0
move \$a0, \$t0	# ο \$t0 έχει την τιμή που θέλουμε να # εμφανίσουμε και το φορτώνουμε στον \$a0
	# σαν παράμετρο
syscall	

Εντολές για εμφάνιση μηνυμάτων στην κονσόλα:

Για να τυπώσετε ένα string στην οθόνη θα πρέπει να το δηλώσετε πρώτα στον τμήμα .data του προγράμματός σας ως εξής:

Αρχικά πρέπει να ορίσετε το string στην περιοχή της μνήμης με ένα label (στην περίπτωση του παραδείγματος String_name1) για να μπορείτε να έχετε πρόσβαση με χρήση της διεύθυνσης του.

```
.data
String_name1: .asciiz  "msg"
```

Σε περίπτωση που θέλετε να αλλάξετε γραμμή (για να είναι πιο ευανάγνωστη η εκτέλεση) πρέπει να ορίσετε ένα string αλλαγής γραμμής.

```
String_name2: .asciiz  "\n"
```

Για να τυπώσετε το string στην κονσόλα πρέπει να γράψετε τις εξής εντολές.

```
li $v0, 4          # κωδικός 4 από τα syscall στον $v0
la $a0, String_name1 # φορτώνουμε στον $a0 την διεύθυνση
                    # του String που θα εκτυπώσουμε
syscall
```

Οι εντολές syscall είναι κλήσεις λειτουργιών του συστήματος. Με αυτές μπορείτε εκτός από I/O λειτουργίες να τερματίσετε ένα πρόγραμμα (syscall με παράμετρο \$v0=10) ή να ζητήσετε δυναμικά μνήμη (αντίστοιχη εντολή malloc της C). Μπορείτε να βρείτε όλες τις λειτουργίες στο Help menu του MARS.

Υλοποίηση δομών ελέγχου ροής προγράμματος

Εντολές ελέγχου ροής είναι το σύνολο των εντολών το οποίο αλλάζουν την ροή εκτέλεσης προγράμματος. Κανονικά η σειρά εκτέλεσης των εντολών είναι μετά από κάθε εντολή να εκτελείται η αμέσως από κάτω. Υπάρχουν όμως εντολές οι οποίες αλλάζουν είτε στατικά (χωρίς συνθήκη) είτε δυναμικά (με συνθήκη) την ροή εκτέλεσης των εντολών.

Παράδειγμα ελέγχου ροής, μετατροπή από C σε assembly:

Παράδειγμα 1 - if

```
if( x == 0) {
    ...      // Case true
}
else
{
    ...      // Case false.
}
...         // Next instruction
```

Αντιστοιχίζουμε την μεταβλητή x στον καταχωρητή \$t3

```
bnez $t3, else
...                # case true
j endif

else:
...                # case false

endif:
...                # next instruction
```

Παράδειγμα 2 - for

```
for (i = 0; i < 10; i++ )
{
...                // code
}
...                // next instruction
```

Αντιστοιχίζουμε την μεταβλητή i στον καταχωρητή \$t3 και αποθηκεύουμε το όριο του loop (εδώ 10) στον καταχωρητή \$t4.

```
li $t3,0           # add $t3, $0, $0
li $t4, 10          # addi $t4,$0,10

for:
bge $t3, $t4,endfor

...                # code

addi $t3,$t3,1
j for

endfor:
...                # next instruction
```

Παράδειγμα 3 - multiple if

```
if (x > 0 && x < 10)
{
...                // code
}
...                // next instruction
```

Αντιστοιχίζουμε την μεταβλητή x στον καταχωρητή \$t3 και αποθηκεύουμε την σταθερά με την οποία θα συγκρίνουμε (εδώ 10) στον καταχωρητή \$t4.

```
li $t4,10
blez $t3, endif
```

```

bge $t3,$t4, endif

...      # code

endif :
...      # next instruction

```

β' τρόπος:

```

addi $t4,$0,10
bgtz $t3, cond2      # if (x > 0)
j endif

cond2:
blt $t3,$t4,is_true  # if (x < 10)
j endif

is_true:

...      #code

endif:
...      # next instruction

```

Χρήση μνήμης

Τα δεδομένα αποθηκεύονται σε ξεχωριστό τμήμα μνήμης το οποίο χρησιμοποιείται αποκλειστικά για αυτό το σκοπό. Γι' αυτό τον λόγο η δήλωση του πίνακα, που θα χρησιμοποιήσετε, πρέπει να γίνει στο τμήμα **.data**. Ο τρόπος δήλωσης του πίνακα και η σύνταξη των εντολών load (lw) και store (sw), οι οποίες χρησιμοποιούνται για την μεταφορά δεδομένων μεταξύ μνήμη και καταχωρητή, εξηγούνται παρακάτω.

Δήλωση πίνακα

```

.data      # στο τμήμα .data πρέπει να δηλωθεί ο πίνακας

Name_array: .space N    # Name_array: δώστε ότι όνομα
                  # θέλετε στον πίνακα
                  # .space N: δηλώνει ότι θέλουμε να
                  # κρατήσουμε χώρο ίσο με N bytes για
                  # τον πίνακα Name_array

```

Εντολή load (lw)

Η εντολή αυτή αποθηκεύει δεδομένα σε καταχωρητή, τα οποία έχει πάρει από συγκεκριμένη διεύθυνση της μνήμης.

Σύνταξη: **lw Rt, Address(Rs)**

Σημασία: **Rt=Memory[Address+Rs]**

Προσοχή: όπου Rt, Rs είναι καταχωρητές και όπου Address είναι το όνομα του πίνακα (label) που δώσατε στο τμήμα .data.

Με βάση την παραπάνω δήλωση (στο τμήμα .data) θα έπρεπε να γράψουμε:

lw Rt, Name_array(Rs)

Από την σύνταξη της εντολής αυτής γίνεται κατανοητό ότι ζητάμε τα δεδομένα της μνήμης στην διεύθυνση [Name_array+Rs] και τα οποία θα αποθηκευτούν στον καταχωρητή Rt.

Εντολή store (sw)

Η εντολή αυτή αποθηκεύει δεδομένα από έναν καταχωρητή σε συγκεκριμένη διεύθυνση στη μνήμη.

Σύνταξη: sw Rt, Address(Rs)	Σημασία: Memory[Address+Rs]=Rt
------------------------------------	---------------------------------------

Ο τρόπος λειτουργίας είναι παρόμοιος με την εντολή **lw(load word)**. Η λειτουργία που υλοποιεί αυτή η εντολή είναι: Αποθήκευσε τα περιεχόμενα του καταχωρητή Rt στην μνήμη και συγκεκριμένα στην διεύθυνση [Address+Rs].

Η διάφορα με την πρώτη εντολή είναι ότι η sw αποθηκεύει δεδομένα στην μνήμη ενώ η lw διαβάζει τα δεδομένα της μνήμης.

Ευθυγράμμιση

Όπως γνωρίζετε από την θεωρία ο MIPS είναι ένας 32 bit επεξεργαστής. Αυτό συνεπάγεται ότι όλοι οι καταχωρητές του έχουν μέγεθος 32 bits ή αλλιώς 4 bytes. Ορίζουμε σαν **word** τα δεδομένα μεγέθους 4 bytes τα οποία χωράνε σε έναν register.

Όταν ορίζουμε έναν πίνακα στην μνήμη με την εντολή **.space** καθορίζουμε το μέγεθος του ίσο με έναν αριθμό από **bytes** όπως παρουσιάστηκε πριν. Εφόσον όμως τα στοιχεία που αποθηκεύονται σε αυτόν θα είναι μεγέθους 4 bytes το καθένα, πρέπει να καθορίσετε το μέγεθος του πίνακα να είναι πολλαπλάσιο του 4. Έτσι το πρώτο στοιχείο του πίνακα βρίσκεται στα bytes 0 έως 3. Αν θέλετε να το κάνετε load θα πρέπει να συντάξετε την εντολή **lw Rt, Address(Rs)** με τον Rs να έχει περιεχόμενο 0. Αν τώρα θέλετε το δεύτερο στοιχείο του πίνακα αυτό βρίσκεται στα bytes 4 έως 7. Για να το κάνετε load θα πρέπει να γράψετε την ίδια εντολή αλλά ο Rs θα πρέπει να έχει περιεχόμενο τον αριθμό 4.

Σημείωση: Όταν ορίζετε έναν πίνακα πρέπει να δηλώνετε και τι μέγεθος θα έχουν τα δεδομένα που θα αποθηκεύονται σε αυτόν. Αυτό γίνεται με την εντολή **.align n** όπου **n** έναν ακέραιος που θα δώσετε εσείς. Η εντολή αυτή σημαίνει ότι τα στοιχεία του πίνακα έχουν μέγεθος **2ⁿ**. Έτσι για έναν πίνακα 10 θέσεων με λέξεις 4 bytes πρέπει να συντάξετε την εντολή:

.align 2	#μέγεθος στοιχείου 4 bytes
label: .space 40	#μέγεθος πίνακα 40 bytes = 10 στοιχεία

Ένα παράδειγμα χρήσης μνήμης

```

.data
.align 2          # λέξεις 4 bytes
vector: .space 24  # πίνακας 6 θέσεων

.text

# ...

li $t1,4          # ένας τρόπος πρόσβασης στο 2ο στοιχείο
lw $t0,vector($t1) # του πίνακα

# ...

la $t2,vector      # ακόμα ένας τρόπος για πρόσβαση στο 2ο
lw $t3,4($t2)       # στοιχείο του πίνακα

# ...

la $t2, vector     # ακόμα ένας τρόπος για πρόσβαση στο 2ο
addi $t2, $t2, 4    # στοιχείο του πίνακα
lw $t0,0($t2)

```

Το πλήθος των εντολών είναι σχετικά μεγάλο αλλά η λειτουργία τους είναι πολύ απλή. Συνιστάται κατά τον προγραμματισμό σε assembly να κάνετε χρήση του instruction set. Μπορείτε επίσης να χρησιμοποιήσετε και ψευδο-εντολές σαν να ήταν κανονικές εντολές (για παράδειγμα *li*, *la*, *bgez*, *blt* κοκ). Η αποστήθιση όλων των εντολών δεν είναι ζητούμενο σε αυτό το εργαστήριο ωστόσο πρέπει να είστε σε θέση να μπορείτε να χρησιμοποιήσετε τις εντολές που αναγράφονται σε αυτό το αρχείο.

Για να πάρετε άριστα στην εξέταση, η λύση σας θα πρέπει να είναι σωστή αλλά και αποδοτική. Θα πρέπει να χρησιμοποιείτε τον ελάχιστο δυνατό αριθμό εντολών στην λύση σας, αλλά και η λύση σας να είναι ευανάγνωστη και καλά σχολιασμένη.

Για το επόμενο εργαστήριο έχετε να υλοποιήσετε τις παρακάτω εργαστηριακές ασκήσεις. Την ώρα του εργαστηρίου θα εξετασθείτε προφορικά πάνω στους κώδικες που θα παραδώσετε.

Άσκηση 1 – Byte manipulation (3 μονάδες)

Να υλοποιήσετε πρόγραμμα σε MIPS assembly το οποίο θα διαβάζει έναν θετικό ακέραιο αριθμό *num* και να εμφανίζει αν είναι δύναμη του 2 ή όχι. Το πρόγραμμά σας θα πρέπει να γράφει στην κονσόλα “*Number is a Power of 2*” εάν ο αριθμός *num* είναι δύναμη του 2 (πχ 2, 32, 4096, κοκ), και “*Number is a Not a Power of 2*” εάν ο αριθμός *num* δεν είναι δύναμη του 2.

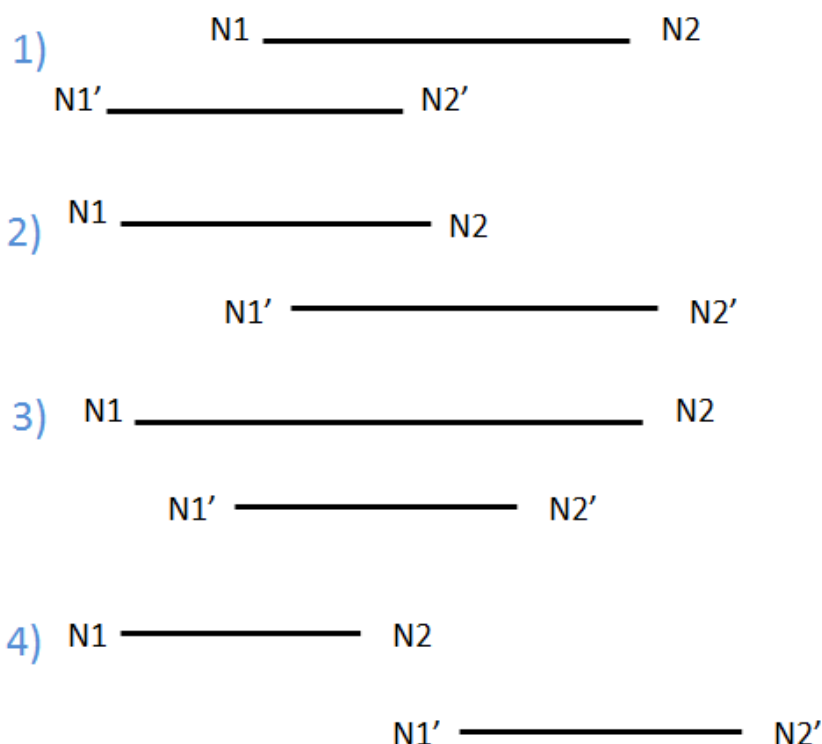
Άσκηση 2-Εύρεση επικαλυπτόμενων διαστημάτων (7 μονάδες)

Να υλοποιήσετε σε assembly ένα πρόγραμμα το οποίο θα ζητά από τον χρήστη να εισάγει επαναληπτικά δυο ακέραιους αριθμούς $N1$ και $N2$, οι οποίοι αντιπροσωπεύουν ένα διάστημα $[N1, N2]$. Αν ο χρήστης εισάγει αρνητικό $N1 < 0$ το πρόγραμμα τερματίζει.

Το πρόγραμμα θα πρέπει να βρίσκει το μέγιστο διάστημα που προκύπτει είτε από αλληλοεπικάλυψη διαστημάτων είτε όχι. Για την εύρεση του μέγιστου διαστήματος κάθε φορά, θα πρέπει να ελέγχονται τέσσερις περιπτώσεις που απεικονίζονται στην Εικόνα 1. Το πρόγραμμα θα πρέπει να κρατάει σε κάθε επανάληψη το αριστερό άκρο και το δεξί άκρο του τρέχοντος μέγιστου διαστήματος (στην Εικόνα 2 απεικονίζονται ως \min και \max). Στην τελευταία επανάληψη, τα \min και \max περιέχουν το τελικό μέγιστο διάστημα.

Πιο συγκεκριμένα, ο χρήστης, στην επανάληψη 0, εισάγει ένα διάστημα $[N1, N2]$, οπότε στην αρχή $\min = N1$ και $\max = N2$. Στην επανάληψη 1, ο χρήστης εισάγει το επόμενο διάστημα $[N1', N2']$ και τα \min , \max ανανεώνονται ανάλογα με τον έλεγχο των περιπτώσεων της Εικόνα 1 μεταξύ του αποτελέσματος της προηγούμενης επανάληψης και του εισαχθέντος διαστήματος κ.ο.κ. Αν δυο διαστήματα δεν αλληλεπικαλύπτονται (περίπτωση 4 της Εικόνα 1), τότε το αποτέλεσμα θα πρέπει να είναι το μέγιστο διάστημα από αυτά τα δύο διαστήματα. Στο τέλος του προγράμματος εκτυπώνεται το τελικό μέγιστο διάστημα.

Για να βρεθεί το μέγιστο των διαστημάτων, υπάρχουν τέσσερις περιπτώσεις αλληλοεπικάλυψης διαστημάτων:



Εικόνα 1

Περίπτωση 1: Το μέγιστο διάστημα της προηγούμενης επανάληψης βρίσκεται πιο δεξιά, και υπάρχει αλληλοεπικάλυψη με το αποτέλεσμα της τρέχουσας επανάληψης. Οπότε το μέγιστο διάστημα είναι $[N1', N2]$.

Περίπτωση 2: Το μέγιστο διάστημα της προηγούμενης επανάληψης βρίσκεται πιο αριστερά, και υπάρχει αλληλοεπικάλυψη με το διάστημα της τρέχουσας επανάληψης. Οπότε το μέγιστο διάστημα είναι $[N1, N2']$.

Περίπτωση 3: Το εισαχθέν διάστημα της τρέχουσας επανάληψης συμπεριλαμβάνεται στο μέγιστο διάστημα της προηγούμενης επανάληψης. Οπότε το μέγιστο διάστημα είναι $[N1, N2]$.

Περίπτωση 4 : Τα διαστήματα της προηγούμενης και της τρέχουσα επανάληψης, δεν επικαλύπτονται, οπότε το αποτέλεσμα είναι το μέγιστο διάστημα από τα δύο διαστήματα.

Παράδειγμα Εκτέλεσης (αναλυτικά βήματα κατά τη διάρκεια των επαναλήψεων στην Εικόνα 2):

Iteration 0

Please give N1:

**** user input : 1

Please give N2:

**** user input : 3

Iteration 1

Please give N1:

**** user input :0

Please give N2:

**** user input :2

Iteration 2

Please give N1:

**** user input :1

Please give N2:

**** user input :3

Iteration 3

Please give N1:

**** user input :1

Please give N2:

**** user input :4

Iteration 4

Please give N1:

**** user input :5

Please give N2:

**** user input :6

Iteration 5

Please give N1:

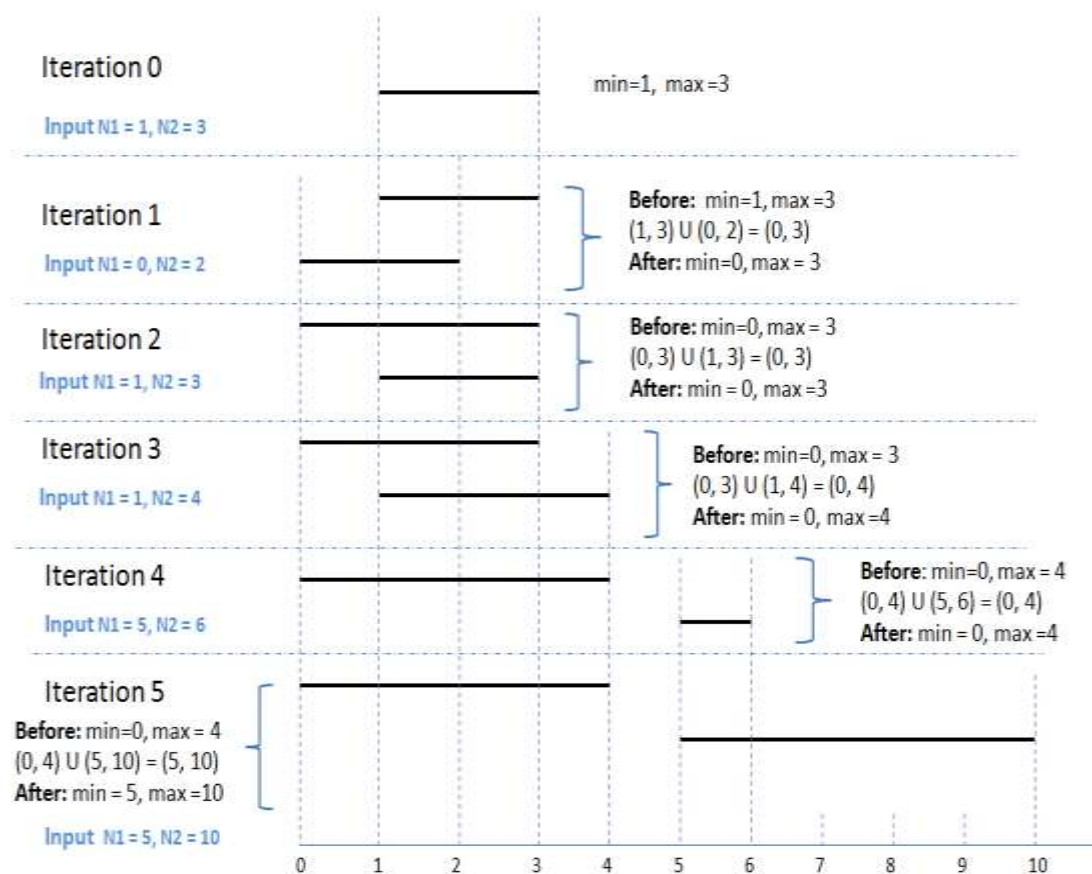
**** user input :5

Please give N2:

**** user input :10

The max final union of ranges is $[5,10]$.

-- program is finished running --



Εικόνα 2

Σε κάθε επανάληψη, το τρέχον μέγιστο διάστημα αποθηκεύεται στις μεταβλητές min και max. Στην εικόνα απεικονίζονται οι επαναλήψεις ενός παραδείγματος εκτέλεσης. Σε κάθε επανάληψη, με την ετικέτα before απεικονίζεται το μέγιστο διάστημα της προηγούμενης επανάληψης, ενώ με την ετικέτα after απεικονίζεται το μέγιστο διάστημα που υπολογίστηκε από το αποτέλεσμα της προηγούμενης επανάληψης και του εισαχθέντος διαστήματος (Input).

Οι εργασίες θα πρέπει να γίνονται upload στο MS teams αμέσως μετά την εξέταση σας στο εργαστήριο.