Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Μάθημα

# Αρχιτεκτονική Υπολογιστών Ι

## Εργαστηριακές Ασκήσεις

**Δ. Γκιζόπουλος, Καθηγητής**

# Εργαστήριο Αρχιτεκτονικής Υπολογιστών Ι
# Προσομοιωτής QtSpim του Μικροεπεξεργαστή MIPS

## Εργαστηριακή Άσκηση 1
Εισαγωγή – Καταχωρητές – Μνήμη – Κλήσεις συστήματος

### Εισαγωγή

Σκοπός του πρώτου εργαστηριακού μαθήματος είναι η εξοικείωση με το περιβάλλον του προσομοιωτή (simulator) και συμβολομεταφραστή (assembler) QtSpim ο οποίος προσομοιώνει τη λειτουργία του μικροεπεξεργαστή MIPS. Επίσης θα γίνει χρήση βασικών εντολών της συμβολικής γλώσσας του MIPS για μετακίνηση δεδομένων μεταξύ μνήμης και καταχωρητών και την εκτέλεση λογικών πράξεων. Θα χρησιμοποιηθούν βασικές οδηγίες (directives) προς τον προσομοιωτή, καθώς και οι κλήσεις συστήματος (system calls) που υποστηρίζει ο προσομοιωτής. Επίσης, θα χρησιμοποιηθεί η δυνατότητα προσομοίωσης με βηματικό τρόπο (single step) και με σημεία διακοπής (breakpoints).

Ο προσομοιωτής είναι ήδη εγκατεστημένος στο εργαστήριο σε λειτουργικό σύστημα Windows, μπορείτε όμως για ιδιωτική χρήση να τον αντιγράψετε από την ιστοσελίδα του μαθήματος http://eclass.di.uoa.gr/courses/D19 (υπάρχουν διαθέσιμες οι εκδόσεις για Windows, Linux, και Mac). Ο προσομοιωτής διατίθεται από τον σχεδιαστή του James Larus στη σελίδα http://sourceforge.net/projects/spimsimulator/files/ και διανέμεται δωρεάν με βάση το σχήμα δωρεάν αδειοδότησης λογισμικού BSD (Berkeley Software Distribution – BSD license).

### Μέρος 1ο
### Εκτέλεση, τμήμα κώδικα και δεδομένων, μεταφορά δεδομένων

#### Άσκηση 1.1

Εκκινήστε τον QtSpim και φορτώστε (File → Load) το πρόγραμμα `lab1_1.s` που βρίσκεται στην ιστοσελίδα του μαθήματος (ο κώδικας δίνεται και παρακάτω). Ανοίξτε με έναν διορθωτή κειμένου (text editor) όπως το Notepad το αρχείο του παραδείγματος.

```
#################################################
#                                               #
# lab1_1.s                              #
# Pseudoinstruction examples - System calls          #
# t0 - holds each byte from string in turn           #
# t1 - contains count of characters            #
# t2 - points to the string                    #
#                                               #
```

```
##################################################

##################################################
#                                                #
#           text segment                         #
#                                                #
##################################################

      .text
            .globl __start
__start:
      la $t2,str        # t2 points to the string
      li $t1,0          # t1 holds the count
      nextCh: lb $t0,($t2)    # get a byte from string
      beqz $t0,strEnd   # zero means end of string
      add $t1,$t1,1           # increment count
      add $t2,1         # move pointer one character
      j nextCh          # go round the loop again
      strEnd: la $a0,ans      # system call to print
      li $v0,4          # out a message
      syscall
      move $a0,$t1            # system call to print
      li $v0,1          # out the length worked out
      syscall
      la $a0,endl             # system call to print
      li $v0,4          # out a newline
      syscall
      li $v0,10
      syscall           # au revoir...


################################################
#                                              #
#                 data segment                         #
#                                              #
################################################

.data
str:  .asciiz     "hello world"
ans:  .asciiz     "Length is "
endl:       .asciiz     "\n"

################################################
#                                              #
# End of File                                        #
#                                              #
################################################
```

Παρατηρήστε στον QtSpim, στο παράθυρο του κώδικα (text) και των δεδομένων (data), τις διευθύνσεις μνήμης που έχουν καταχωρηθεί ο κώδικας και τα δεδομένα του προγράμματος. Αντιπαραβάλλοντας τα περιεχόμενα του παραθύρου του κώδικα και του Notepad διαπιστώστε ποιες από τις εντολές του παραδείγματος `lab1_1.s` είναι ψευδοεντολές (δηλαδή αναλύονται σε περισσότερες από μία πραγματικές εντολές του επεξεργαστή MIPS ή αντιστοιχούν σε διαφορετικό μνημονικό πραγματικής εντολής MIPS).

Το παράθυρο του κώδικα φαίνεται παρακάτω και δείχνει τη διεύθυνση κάθε εντολής (μέσα σε [ ]). Οι εντολές απέχουν μεταξύ τους 4 byte. Στη δεύτερη στήλη, δεξιά από τη στήλη που περιέχει τις διευθύνσεις, βρίσκεται η

στήλη με το περιεχόμενο της κάθε λέξης εντολής (σε δεκαεξαδικό), ακολουθεί η πραγματική εντολή μηχανής MIPS, και μετά το ελληνικό ερωτηματικό φαίνεται η αντίστοιχη εντολή από τον πηγαίο κώδικα του αρχείου assembly.

Παρατηρήστε ότι η ψευδοεντολή la (load address) αναλύεται σε δύο πραγματικές εντολές (τις lui και ori) οι οποίες μαζί έχουν σαν αποτέλεσμα να φορτωθεί στον καταχωρητή $t2 η διεύθυνση της συμβολοσειράς str. Η διεύθυνση αυτή είναι η δεκαεξαδική 0x10010000 στην οποία αρχίζουν τα δεδομένα του χρήστη (User Data).

Παρατηρήστε την ψευδοεντολή move η οποία μετατρέπεται σε addu.

Επίσης παρατηρήστε την ψευδοεντολή add $t2,1 η οποία μετατρέπεται από τον assembler στην πραγματική εντολή MIPS addi $t2, $t2, 1 (αντί για $t2 βλέπετε $10 και το 10 είναι ο αριθμός του καταχωρητή $t2). Τόσο το μνημονικό όνομα της εντολής αλλάζει (αντί addi χρησιμοποιούμε στον πηγαίο κώδικα το add), όσο και το πλήθος των τελεστέων: ενώ οι πραγματικές εντολές απαιτούν τρεις, εδώ στον πηγαίο κώδικα χρησιμοποιούμε μόνο δύο).

Data | Text

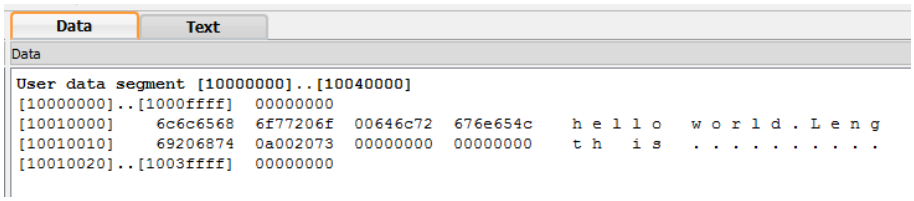
Text

```
                        User Text Segment [00400000]..[00440000]
[00400000] 3c011001  lui $1, 4097 [str]       ; 19: la $t2,str # t2 points to the string
[00400004] 342a0000  ori $10, $1, 0 [str]
[00400008] 34090000  ori $9, $0, 0            ; 20: li $t1,0 # t1 holds the count
[0040000c] 81480000  lb $8, 0($10)            ; 21: lb $t0,($t2) # get a byte from string
[00400010] 11000004  beq $8, $0, 16 [strEnd-0x00400010]
[00400014] 21290001  addi $9, $9, 1           ; 23: add $t1,$t1,1 # increment count
[00400018] 214a0001  addi $10, $10, 1         ; 24: add $t2,1 # move pointer one character
[0040001c] 08100003  j 0x0040000c [nextCh]    ; 25: j nextCh # go round the loop again
[00400020] 3c011001  lui $1, 4097 [ans]       ; 26: la $a0,ans # system call to print
[00400024] 3424000c  ori $4, $1, 12 [ans]
[00400028] 34020004  ori $2, $0, 4            ; 27: li $v0,4 # out a message
[0040002c] 0000000c  syscall                  ; 28: syscall
[00400030] 00092021  addu $4, $0, $9          ; 29: move $a0,$t1 # system call to print
[00400034] 34020001  ori $2, $0, 1            ; 30: li $v0,1 # out the length worked out
[00400038] 0000000c  syscall                  ; 31: syscall
[0040003c] 3c011001  lui $1, 4097 [endl]      ; 32: la $a0,endl # system call to print
[00400040] 34240017  ori $4, $1, 23 [endl]
[00400044] 34020004  ori $2, $0, 4            ; 33: li $v0,4 # out a newline
[00400048] 0000000c  syscall                  ; 34: syscall
[0040004c] 3402000a  ori $2, $0, 10           ; 35: li $v0,10
[00400050] 0000000c  syscall                  ; 36: syscall # au revoir...

                        Kernel Text Segment [80000000]..[80010000]
```

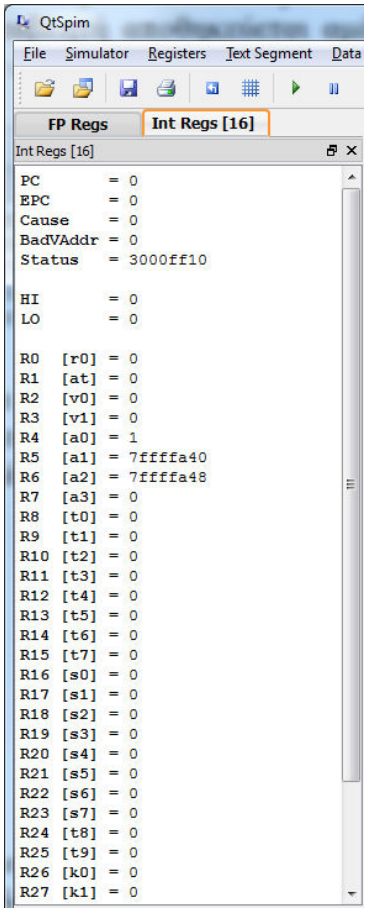
Το παράθυρο δεδομένων φαίνεται παρακάτω. Τα δεδομένα χρήστη αρχίζουν στη δεκαεξαδική διεύθυνση 0x10010000 και κάθε γραμμή που ακολουθεί μια διεύθυνση δείχνει το περιεχόμενο 16 διαδοχικών byte της μνήμης. Για κάθε byte δίνεται αρχικά το δεκαεξαδικό του περιεχόμενο (π.χ. το byte στη διεύθυνση 0x10010000 είναι ίσο με 0x6c δηλαδή δυαδικά ίσο με 01101100). Μετά τη δεκαεξαδική αναπαράσταση του περιεχομένου του κάθε byte δίνεται ο ASCII χαρακτήρας που αυτό παριστάνει. Το 0x6c είναι ίσο με 108 στο δεκαδικό και το 108 είναι ο ASCII κωδικός για το πεζό γράμμα της αγγλικής αλφαβήτου “h”.

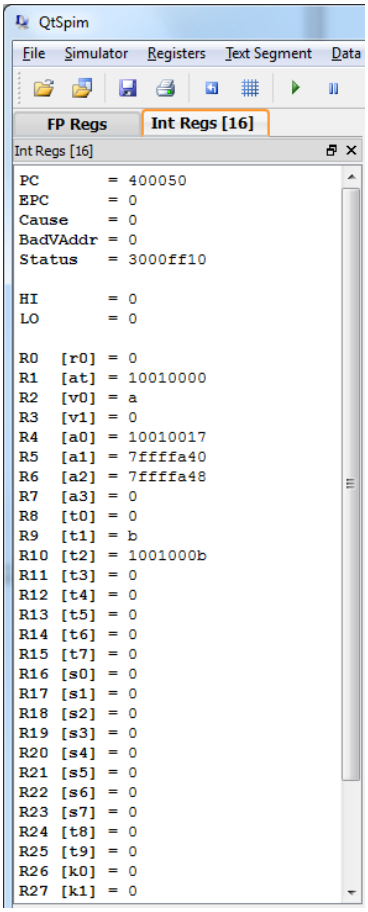
Βλέπετε ολόκληρη τη συμβολοσειρά “hello world” σε διαδοχικά byte της μνήμης. Όταν ένα byte αντιστοιχεί σε μη εκτυπώσιμο χαρακτήρα του ASCII κώδικα τότε βλέπουμε μια τελεία. Η τελεία που ακολουθεί το “hello world” είναι ο μηδενικός χαρακτήρας (null character) που στη C δηλώνει το τέλος μιας συμβολοσειράς. Μετά το “hello world” ακολουθεί η επόμενη συμβολοσειρά του προγράμματος “Length is ”.

Όταν μια μεταβλητή οποιουδήποτε τύπου δηλώνεται στο τμήμα δεδομένων του προγράμματος (Data Segment) μετά την οδηγία .data, τότε η μεταβλητή αποθηκεύεται αμέσως μετά την προηγούμενη μεταβλητή σε διαδοχικές θέσεις μνήμης. Καταλαμβάνει τόσα byte όσο είναι το μέγεθος του τύπου της.




Εκτελέστε το πρόγραμμα με βηματικό τρόπο (Simulator → Single Step ή F10) και παρατηρήστε τις αλλαγές των σχετικών καταχωρητών που μετέχουν στο παράδειγμα σε κάθε βήμα εκτέλεσης. Τα περιεχόμενα των καταχωρητών μπορείτε να τα βλέπετε στο σχετικό παράθυρο του QtSpim. Η εικόνα που ακολουθεί δείχνει το αρχικό περιεχόμενο των καταχωρητών πριν την εκτέλεση του προηγούμενου προγράμματος (αριστερά) και το τελικό περιεχόμενό τους μετά την εκτέλεσή του (δεξιά). Προσέξτε, για παράδειγμα, την τελική τιμή του καταχωρητή $t1 ο οποίος μετρά το πλήθος των χαρακτήρων της συμβολοσειράς. Είναι ίση με το δεκαεξαδικό b δηλαδή 11 στο δεκαδικό σύστημα αρίθμησης που είναι και το σωστό αποτέλεσμα του προγράμματος αφού το μήκος της συμβολοσειράς είναι 11 χαρακτήρες.

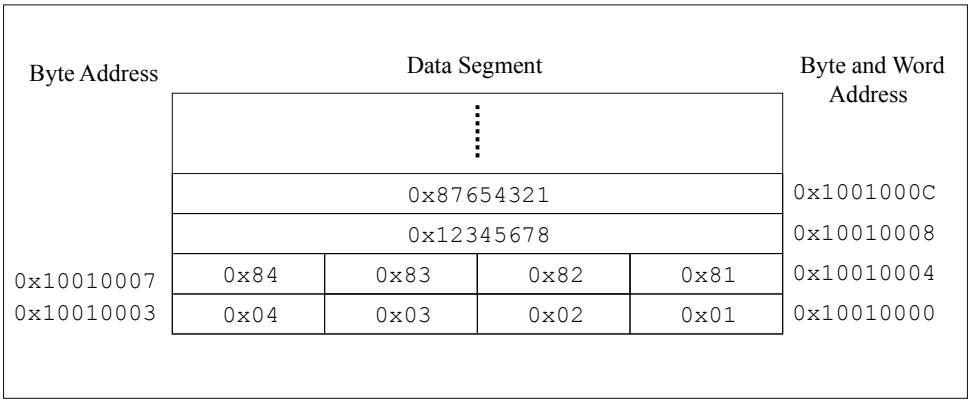


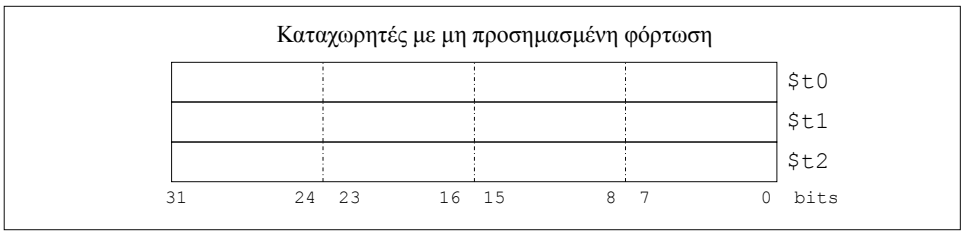
**Άσκηση 1.2**

Στην άσκηση αυτή θα φανούν οι διάφορες κατηγορίες εντολών φόρτωσης (load) και αποθήκευσης (store) καταχωρητών από/προς τη μνήμη που διαθέτει ο QtSpim.

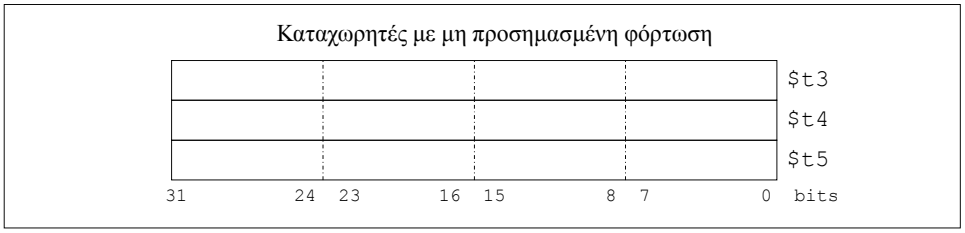
(α) Δημιουργήστε στο τμήμα δεδομένων (data segment) την ακόλουθη διάταξη, χρησιμοποιώντας τις οδηγίες (directives) `.data`, `.byte` και `.word`. Αυτές δεσμεύουν bytes και words αντίστοιχα στο τμήμα δεδομένων της μνήμης.




(β) Στη συνέχεια, στο κυρίως πρόγραμμα χρησιμοποιήστε τις εντολές μη προσημασμένης (ή απρόσημης – unsigned) φόρτωσης `lbu` (load byte unsigned) και `lhu` (load halfword unsigned) καθώς και την εντολή `lw` (load word) για να φορτώσετε στους καταχωρητές `$t0`, `$t1` και `$t2` τα bytes, half words και words, αντίστοιχα, που ξεκινούν από τη διεύθυνση `0x10010000` (αυτά που έχουν τιμές `0x01`, `0x0201` και `0x04030201`, αντίστοιχα). Συμπληρώστε τον πίνακα που ακολουθεί.

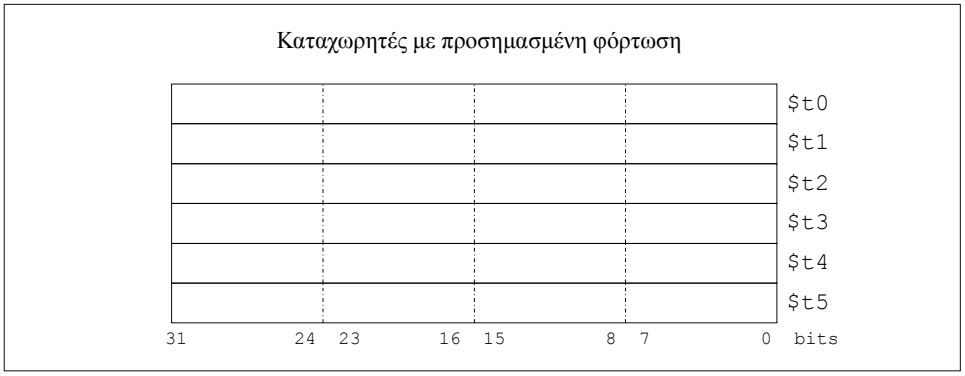



(γ) Κάντε το ίδιο (μη προσημασμένη μεταφορά) αλλά αυτή τη φορά στους καταχωρητές `$t3` `$t4` και `$t5` για τα bytes, half words και words που ξεκινούν από τη διεύθυνση `0x10010004` (αυτά που έχουν τιμές `0x81`, `0x8281` και `0x84838281`, αντίστοιχα). Συμπληρώστε τον πίνακα που ακολουθεί.

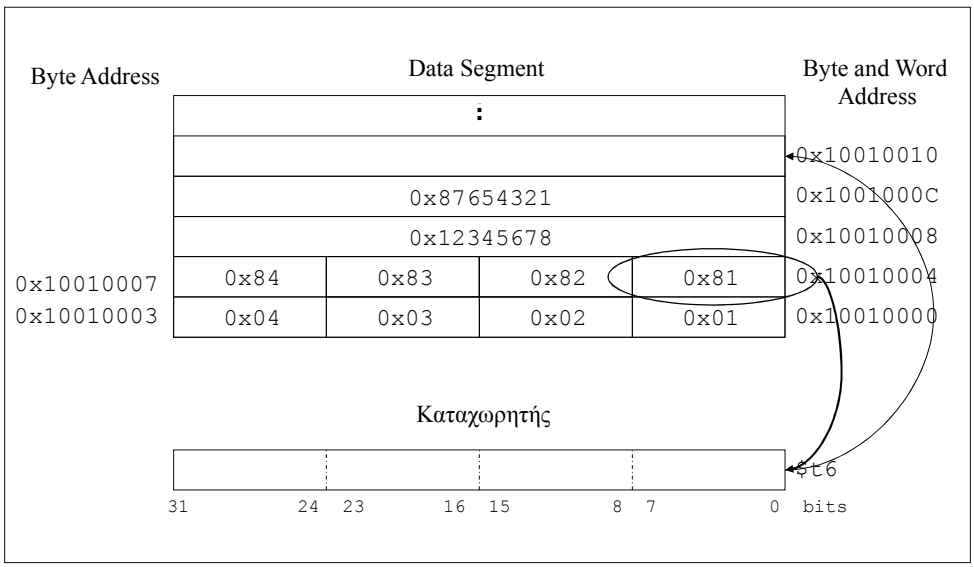


(δ) Εμπλουτίστε το πρόγραμμα επαναλαμβάνοντας τις προηγούμενες μεταφορές χρησιμοποιώντας αυτή τη φορά τις εντολές προσημασμένης (signed) φόρτωσης `lb` (load byte), `lh` (load half) και `lw` (η load word είναι η ίδια τόσο για τη προσημασμένη όσο και την απρόσημη μεταφορά). Χρησιμοποιήστε τους ίδιους καταχωρητές (`$t0, $t1, $t2, $t3, $t4, $t5`), βάζοντας ένα σημείο διακοπής/breakpoint (με δεξί κλικ επάνω στην εντολή που θέλετε να βάλετε το breakpoint και επιλογή Set Breakpoint από το μενού που θα εμφανιστεί) εκεί που αρχίζετε να τους ξαναχρησιμοποιείτε, έτσι ώστε να δείτε τα διαδοχικά περιεχόμενά τους κατά τις δύο φάσεις εκτέλεσης (πριν και μετά το breakpoint). Παρατηρήστε την επέκταση προσήμου που συμβαίνει κατά τις προσημασμένες φορτώσεις.

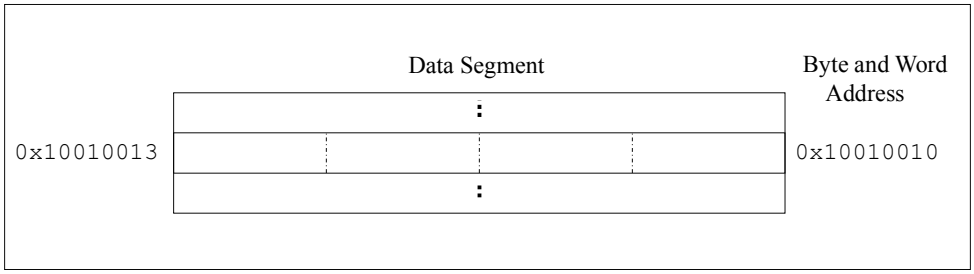
Συμπληρώστε τον παρακάτω πίνακα με τις τιμές των καταχωρητών.




(ε) Φορτώστε στον καταχωρητή `$t6` με μη προσημασμένο τρόπο το byte της διεύθυνσης `0x10010004` και κατόπιν αποθηκεύστε το χαμηλό byte του καταχωρητή (με την εντολή `sb` – store byte) στην πρώτη διεύθυνση που είναι ελεύθερη (μετά τις πρώτες τέσσερις λέξεις που είναι ήδη αποθηκευμένες στη μνήμη). Ελέγξτε αν σε αυτήν την περίπτωση έχουμε επέκταση προσήμου. Η εικόνα που ακολουθεί δείχνει την επιθυμητή λειτουργία. Συμπληρώστε το περιεχόμενο της λέξης που ξεκινά από τη διεύθυνση 0x10010010.



(στ) Ένας πραγματικός επεξεργαστής MIPS μπορεί να ρυθμιστεί κατά τη διάρκεια του hardware reset αν θα λειτουργεί βλέποντας τη μνήμη σαν big-endian ή little-endian (αυτή η ρύθμιση ισχύει και σε αρκετούς άλλους επεξεργαστές). Ο προσομοιωτής QtSpim όμως χρησιμοποιεί τον τρόπο οργάνωσης που χρησιμοποιεί ο επεξεργαστής στον οποίο τρέχει η προσομοίωση. Στην περίπτωση επεξεργαστών Intel το μοντέλο είναι little-endian. Πως θα αποθηκευόταν ο `$t6` στη μνήμη αν ο επεξεργαστής ακολουθούσε το μοντέλο big-endian; Συμπληρώστε τον πίνακα.

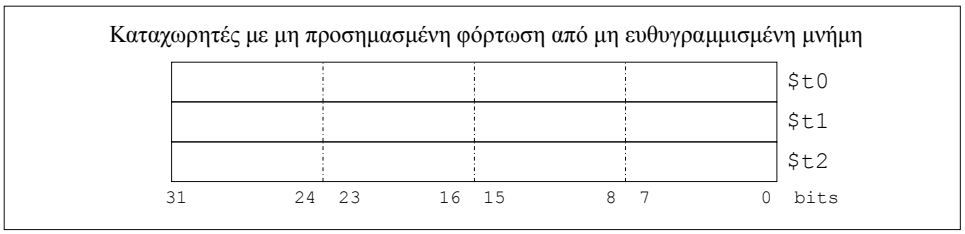



**Άσκηση 1.3**

Στην άσκηση αυτή θα διευκρινιστεί η έννοια της ευθυγράμμισης (alignment) στη μνήμη. Μία λέξη των 4 bytes είναι ευθυγραμμισμένη όταν αποθηκεύεται σε διεύθυνση που είναι πολλαπλάσιο του 4.

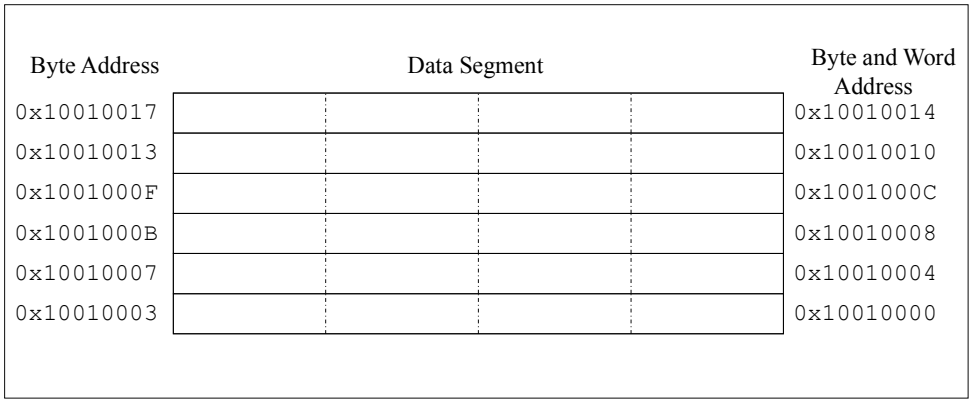
(α) Αλλάξτε το πρόγραμμα της προηγούμενης άσκησης διατηρώντας τη δομή του τμήματος δεδομένων (data segment) και φορτώστε διαδοχικά στους καταχωρητές `$t0`, `$t1` και `$t2` τα byte (χωρίς επέκταση προσήμου), από τις διευθύνσεις `0x1001000A`, `0x1001000B` (μη ευθυγραμμισμένες) και `0x1001000C` (ευθυγραμμισμένη).

Συμπληρώστε τον πίνακα που ακολουθεί.




(β) Προσθέστε στο τμήμα δεδομένων ένα επί πλέον byte με τιμή `0x05` και μισή λέξη (οδηγία `.half` ) με τιμή `0x6677` ανάμεσα στο byte με τιμή `0x04` και το byte με τιμή `0x81`. Κατόπιν φορτώστε σε έναν καταχωρητή τη λέξη με περιεχόμενο `0x12345678` από τη μνήμη. Φορτώστε το πρόγραμμα στον προσομοιωτή ώστε να παρατηρήσετε την αυτόματη ευθυγράμμιση στη διάταξη των δεδομένων στη μνήμη του προσομοιωτή.

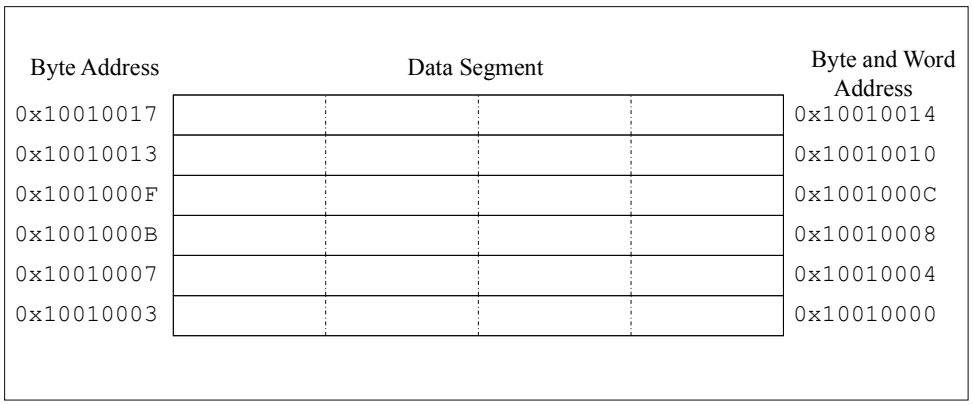
Συμπληρώστε τον πίνακα που ακολουθεί.




Κατόπιν γράψτε τον κώδικα έτσι ώστε στον καταχωρητή `$t0` να φορτωθεί η λέξη με περιεχόμενο `0x12345678` και στον `$t1` η μισή λέξη με περιεχόμενο `0x6677` και εκτελέστε το πρόγραμμα.

(γ) Αλλάξτε το προηγούμενο πρόγραμμα έτσι ώστε τα δεδομένα να μην ευθυγραμμίζονται αυτόματα. Αυτό μπορεί να γίνει αν συμπεριληφθεί η οδηγία `.align 0` αμέσως μετά την `.data`. Προσπαθήστε να κάνετε ότι και πριν. Επειδή τώρα τα δεδομένα (εκτός από τα bytes) δεν είναι ευθυγραμμισμένα, εμφανίζεται μήνυμα λάθους "`Exception occurred at PC=0x00400008 Unaligned address in inst/data fetch: 0x1001000b`".

Παρ' όλα αυτά στο σύνολο εντολών του MIPS υπάρχουν οι ψευδοεντολές `ulw` (unaligned load word) και `ulh` (unaligned load half word) που επιτρέπουν την προσπέλαση σε μη ευθυγραμμισμένες λέξεις και μισές λέξεις αντίστοιχα. Χρησιμοποιήστε τις εντολές αυτές και δείτε πως αναλύονται σε πραγματικές εντολές MIPS.




## Μέρος 2ο

## Κλήσεις συστήματος

### Άσκηση 1.4

Γράψτε ένα πρόγραμμα που θα διαβάζει μία συμβολοσειρά (string) που θα δίνει ο χρήστης. Το περιεχόμενο της συμβολοσειράς φαίνεται στο παράθυρο της κονσόλας. Κατόπιν θα εμφανίζει στο ίδιο παράθυρο μόνο τους πέντε πρώτους χαρακτήρες από τη συμβολοσειρά.

Χρησιμοποιήστε τις κλήσεις συστήματος `read_string` και `print_string` για αυτό το σκοπό. Τα περιεχόμενα της κονσόλας μετά την εκτέλεση του προγράμματος πρέπει να είναι όπως δείχνει η εικόνα που ακολουθεί.


**Άσκηση 1.5**

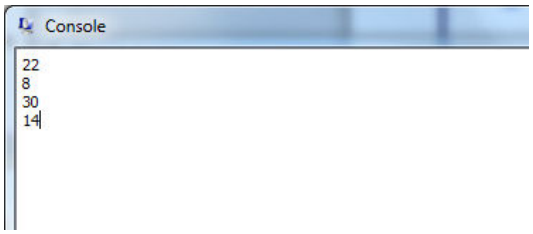
Γράψτε ένα πρόγραμμα που θα διαβάζει από το παράθυρο της κονσόλας έναν ακέραιο που θα δίνει ο χρήστης. Κατόπιν θα εμφανίζει στο ίδιο παράθυρο τον ίδιο ακέραιο.

Χρησιμοποιήστε τις κλήσεις συστήματος `read_int` και `print_int` για αυτό το σκοπό.
Πειραματιστείτε εισάγοντας μεταξύ άλλων και τους ακεραίους 2,147,483,648 (maxint+1) και –2,147,483,649 (–maxint – 1). Τι αποτέλεσμα σας δίνει η εκτέλεση του προγράμματος στις περιπτώσεις αυτές;

**Άσκηση 1.6**

Γράψτε ένα πρόγραμμα που θα διαβάζει από το παράθυρο της κονσόλας δύο ακεραίους που θα δίνει ο χρήστης. Κατόπιν θα εμφανίζει στο ίδιο παράθυρο το άθροισμα των δύο ακεραίων και τη διαφορά τους.

Ένα παράδειγμα εκτέλεσης του προγράμματος φαίνεται παρακάτω.



# Εργαστήριο Αρχιτεκτονικής Υπολογιστών Ι
# Προσομοιωτής QtSpim του Μικροεπεξεργαστή MIPS

## Εργαστηριακή Άσκηση 2
### Αριθμητικές και Λογικές Πράξεις

**Εισαγωγή**

Σε αυτό το εργαστήριο μελετώνται οι εντολές λογικών πράξεων (πρώτο μέρος) μεταξύ λέξεων και αριθμητικών πράξεων (δεύτερο μέρος) ακεραίων αριθμών του MIPS. Ειδικά για τις αριθμητικές πράξεις πρόσθεσης-αφαίρεσης θα δειχθεί και η συμπεριφορά τους σε ακραίες περιπτώσεις όπως στην περίπτωση της υπερχείλισης (overflow). Επειδή ο MIPS δε διαθέτει ειδικό καταχωρητή κατάστασης (status register) που να περιέχει ψηφία κρατουμένου και υπερχείλισης για άμεση χρήση από εντολές, όπως άλλοι μικροεπεξεργαστές, μέρος των ασκήσεων είναι η προσπάθεια δημιουργίας τους με προγραμματιστικό τρόπο. Στο τρίτο μέρος της άσκησης θα μελετηθούν οι πράξεις του πολλαπλασιασμού και της διαίρεσης μεταξύ ακεραίων αριθμών.

**Μέρος 1ο**
**Λογικές Πράξεις**

**Άσκηση 2.1**

Γράψτε ένα πρόγραμμα που να μετατρέπει έναν δυαδικό αριθμό των 32-bit στον αντίστοιχό του σε κωδικοποίηση Gray. Ο αριθμός θα εισάγεται απευθείας σε ένα καταχωρητή. Αυτό γίνεται με δεξί κλικ στο όνομα του καταχωρητή στο παράθυρο των καταχωρητών και επιλογή Change Register Contents από το μενού που εμφανίζεται. Φροντίστε οι καταχωρητές να εμφανίζονται σε δεκαεξαδική μορφή ώστε να είναι εύκολη η μετατροπή τους σε δυαδικό σύστημα (επιλογή Registers → Hex).

Ο αλγόριθμος μετατροπής ενός δυαδικού αριθμού σε αναπαράσταση κώδικα Gray δίνεται παρακάτω, όπου $b_i$ είναι το bit *i*-στής τάξης του δυαδικού αριθμού και $g_i$ το bit *i*-στής τάξης του αντίστοιχου Gray αριθμού (με το *i* να λαμβάνει τιμές από 0 μέχρι και $n - 1$.

$g_n = b_n$
$g_i = b_{i+1}$ XOR $b_i$ , για $i = 0 \ldots n - 1$

Χρησιμοποιήστε μόνο λογικές πράξεις και ολισθήσεις για να υλοποιήσετε τον παραπάνω αλγόριθμο (δε χρειάζεται βρόχος επανάληψης – τους βρόχους θα τους μελετήσουμε αργότερα).

| Binary | Hexadecimal | Gray |
|---|---|---|
| 0000 | 0 | 0000 |
| 0001 | 1 | 0001 |
| 0010 | 2 | 0011 |
| 0011 | 3 | 0010 |
| 0100 | 4 | 0110 |
| 0101 | 5 | 0111 |
| 0110 | 6 | 0101 |
| 0111 | 7 | 0100 |
| 1000 | 8 | 1100 |
| 1001 | 9 | 1101 |
| 1010 | A | 1111 |
| 1011 | B | 1110 |
| 1100 | C | 1010 |
| 1101 | D | 1011 |
| 1110 | E | 1001 |
| 1111 | F | 1000 |

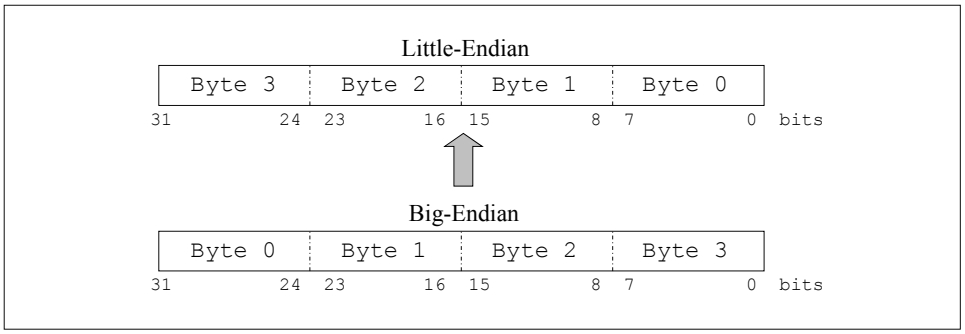
Πειραματιστείτε με μικρούς αριθμούς και συμβουλευτείτε τον παραπάνω πίνακα αντιστοίχισης για τον κώδικα Gray των 4 bit για να επιβεβαιώσετε το αποτέλεσμα, παρατηρώντας τον Gray κωδικοποιημένο αριθμό που θα βρίσκεται σε έναν καταχωρητή της επιλογής σας.

**Άσκηση 2.2**

Υποθέστε ότι ένας επεξεργαστής MIPS λειτουργεί χρησιμοποιώντας το μοντέλο μνήμης little-endian (μικρού άκρου) για την οργάνωση των byte (όπως ο QtSpim σε PC) και επικοινωνεί με διάφορες άλλες μηχανές κάποιες από τις οποίες χρησιμοποιούν το μοντέλο μνήμης big-endian (μεγάλου άκρου) για την οργάνωση των byte. Τα δεδομένα που προέρχονται από τις άλλες μηχανές είναι φυσικά λέξεις των 32 bit και εισάγονται σε έναν καταχωρητή όχι με τη μορφή που τα χρειάζεται ο MIPS.

Θέλουμε να μετατρέψουμε τα δεδομένα από τη «λανθασμένη» μορφή big-endian που λαμβάνονται στη μορφή little-endian σε έναν άλλο καταχωρητή, όπως στο παρακάτω σχήμα. Να σημειωθεί ότι μέσα σε κάθε byte η σειρά των bit είναι η σωστή (δηλαδή το byte 0 αποθηκεύει τα bit 7-0, το byte 1 αποθηκεύει τα bit 15-8, κ.λπ.)

Γράψτε το σχετικό πρόγραμμα μετατροπής, χωρίς να χρησιμοποιήσετε καθόλου προσπέλαση στη μνήμη, παρά μόνο λογικές πράξεις μεταξύ καταχωρητών και αυτό για λόγους ταχύτητας επεξεργασίας.



## Μέρος 2ο

## Πρόσθεση - Αφαίρεση - Υπερχείλιση - Κρατούμενο

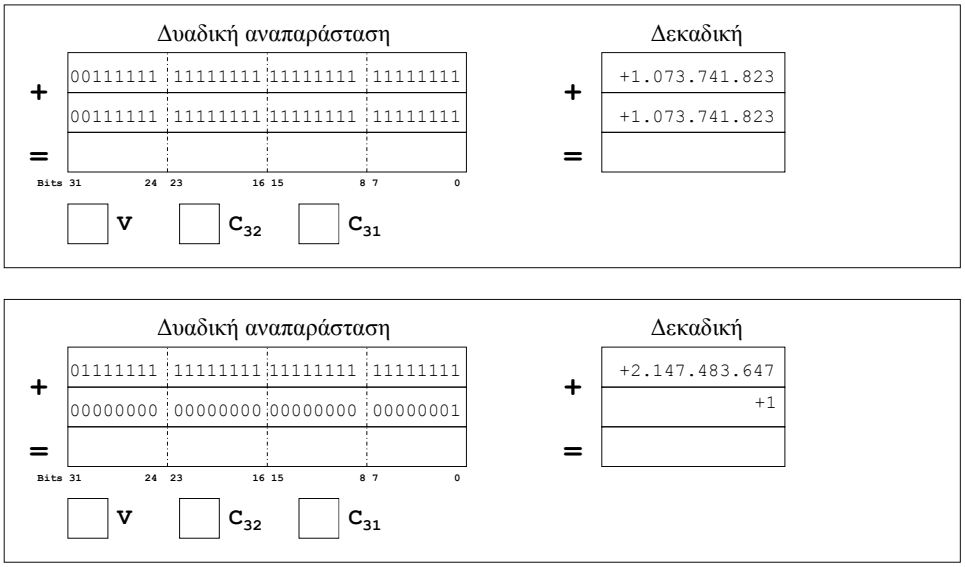
### Άσκηση 2.3

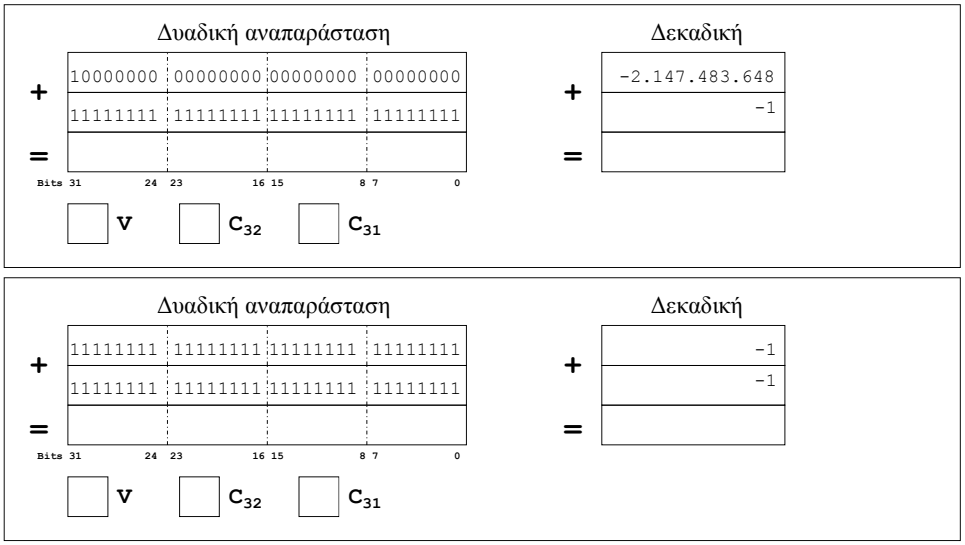
(α) Θεωρήστε τα παρακάτω ζευγάρια προσημασμένων δυαδικών αριθμών των 32-bit σε αναπαράσταση συμπληρώματος ως προς 2. Κάντε τις προσθέσεις που φαίνονται σε δυαδικό σύστημα και συμπληρώστε το αποτέλεσμα. Επίσης συμπληρώστε το σχετικό bit `V` της υπερχείλισης (1 αν έχει συμβεί υπερχείλιση και 0 αν δεν έχει συμβεί). Επιπλέον συμπληρώστε τα κρατούμενα που δίνει η πρόσθεση στα bit τελευταίας και προτελευταίας τάξης, δηλαδή τα $C_{32}$ και $C_{31}$ , αντίστοιχα. (Προσοχή, τα bit $C_{32}$, $C_{31}$ και `V` δεν υπάρχουν στον MIPS αλλά είναι εδώ απλά συμβολισμός της άσκησης).

(β) Κάντε την αντίστοιχη πράξη σε δεκαδικό σύστημα και ελέγξτε κατά πόσον το αποτέλεσμα είναι ίδιο με αυτό της δυαδικής αναπαράστασης.

(γ) Εκτελέστε τις ίδιες πράξεις στο περιβάλλον προσομοίωσης QtSpim χρησιμοποιώντας τις εντολές του MIPS `add` (προσημασμένη πρόσθεση) και `addu` (add unsigned – απρόσημη πρόσθεση). Τα δεδομένα να τα εισάγετε στο data segment (δηλαδή μετά την οδηγία `.data`), τα αποτελέσματα να τα παρατηρείτε ως δεκαεξαδικούς αριθμούς στα περιεχόμενα των καταχωρητών, αλλά και σαν δεκαδικούς στο παράθυρο της κονσόλας χρησιμοποιώντας την κλήση συστήματος `print_int` για την εκτύπωσή τους.

Ποιες διαφορές έχουν οι δύο αυτές εντολές `add` και `addu`; Που φαίνεται αν έχει συμβεί υπερχείλιση;






**Άσκηση 2.4**

(α) Ο MIPS δεν προσφέρει κάποιο bit ανίχνευσης υπερχείλισης όπως άλλοι μικροεπεξεργαστές. Ο τρόπος ανίχνευσης είναι με τη δημιουργία εσωτερικής διακοπής ή εξαίρεσης (exception). Στο βιβλίο του μαθήματος (ενότητα 3.2) υπάρχει ένα παράδειγμα ανίχνευσης υπερχείλισης που αποφεύγει τη δημιουργία εσωτερικής διακοπής. Γράψτε ένα ανάλογο πρόγραμμα που μπορεί να ανιχνεύει υπερχειλίσεις δύο προσημασμένων αριθμών που πρόκειται να προστεθούν χωρίς όμως να χρησιμοποιήσετε υπό συνθήκη άλματα (conditional branches – θα τα δούμε αργότερα) όπως στο παράδειγμα του βιβλίου, παρά μόνο λογικές πράξεις μεταξύ καταχωρητών. Το bit υπερχείλισης να αποθηκεύεται σε ένα bit κάποιου καταχωρητή.

(β) Σε άλλους μικροεπεξεργαστές η σημαία υπερχείλισης παράγεται από τα κρατούμενα τελευταίας και προτελευταίας τάξης που παράγει ο αθροιστής τους. Παρατηρήστε τα αποτελέσματα που πήρατε στην προηγούμενη άσκηση και βρείτε ποια λογική συνάρτηση δίνει την υπερχείλιση.

**Άσκηση 2.5**

Το ζητούμενο αυτής της άσκησης είναι ένας τρόπος παραγωγής κρατουμένου από τον αθροιστή 32 bit του MIPS, επειδή αυτό δεν προσφέρεται από τον συγκεκριμένο μικροεπεξεργαστή. Υποθέτουμε ότι ο αθροιστής της ALU του MIPS αποτελείται από 32 πλήρεις αθροιστές (full adders) του ενός bit συνδεδεμένους μεταξύ τους με τα κρατούμενα εισόδου και εξόδου. Ο πίνακας αλήθειας ενός πλήρους αθροιστή για το bit *i* δίνεται παρακάτω.

| **Είσοδοι** | | | **Έξοδοι** | |
|---|---|---|---|---|
| $C_i$ | $A_i$ | $B_i$ | $C_{i+1}$ | $S_i$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |

$A_i$ και $B_i$ είναι τα δύο bit *i*-τάξης που πρόκειται να προστεθούν, $C_i$ είναι το κρατούμενο που δίνει ο αθροιστής που προσθέτει τα $A_{i-1}$ και $B_{i-1}$. Το $S_i$ είναι το άθροισμα και $C_{i+1}$ το κρατούμενο που παράγει ο αθροιστής που προσθέτει τα $A_i$ και $B_i$. Βρείτε τον πίνακα αλήθειας του $C_{i+1}$ όταν είναι γνωστά τα $A_i$, $B_i$ και $S_i$ (αντί για το $C_i$). Ποια είναι η προϋπόθεση για να υπάρχει τέτοιος πίνακας αλήθειας;

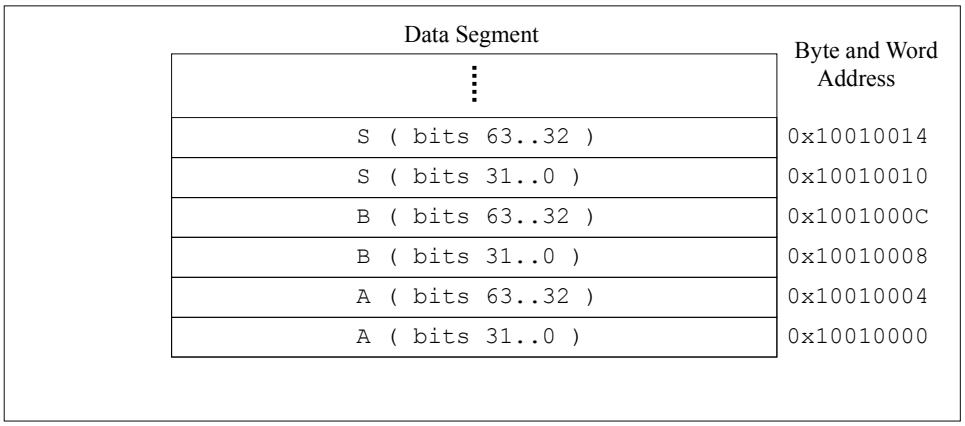
| **Είσοδοι** | | | **Έξοδος** |
|---|---|---|---|
| $S_i$ | $A_i$ | $B_i$ | $C_{i+1}$ |
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

Κατόπιν γράψτε ένα πρόγραμμα στον QtSpim που να υπολογίζει το τελικό κρατούμενο (δηλαδή το 33ο bit) μίας πρόσθεσης ($C_{32}$) μεταξύ δύο καταχωρητών και να το αποθηκεύει σε ένα bit κάποιου άλλου καταχωρητή. Οι αριθμοί που θα προστεθούν να εισάγονται μέσω του data segment (δηλαδή μετά την οδηγία `.data`).

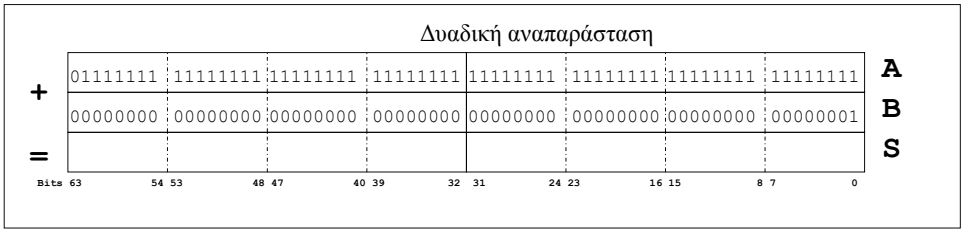
**Άσκηση 2.6**

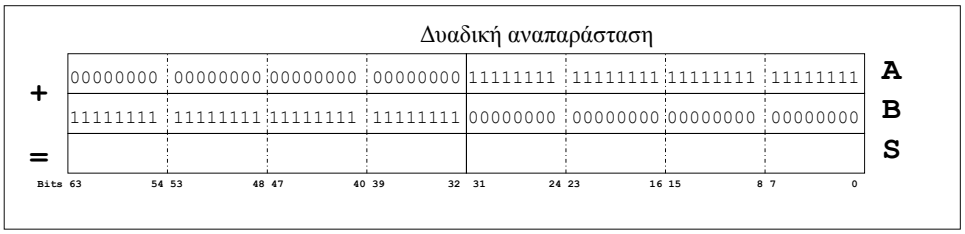
Στην άσκηση αυτή θα δημιουργήσετε ένα πρόγραμμα που θα μπορεί να κάνει πρόσθεση δύο προσημασμένων αριθμών A και B των 64 bit ο καθένας. Αρχικά οι αριθμοί θα βρίσκονται στο data segment (δηλαδή μετά την οδηγία `.data` στον πηγαίο κώδικά σας) και η διάταξή τους θα ακολουθεί το μοντέλο little–endian, δηλαδή η λέξη με τα λιγότερο σημαντικά bit θα βρίσκεται σε χαμηλότερη διεύθυνση όπως φαίνεται και στο σχήμα. Το αποτέλεσμα της άθροισης S θα αποθηκεύεται και αυτό στη μνήμη όπως δείχνει το σχήμα.

Εκμεταλλευτείτε την προηγούμενη άσκηση ώστε το πρόγραμμά σας να χρησιμοποιεί το κρατούμενο $C_{32}$ στην πρόσθεση των bit στη θέση 32 (δηλαδή να «μεταφερθεί» το κρατούμενο από τη λέξη που αποθηκεύει τα 32 χαμηλότερα bit στη λέξη που αποθηκεύει τα 32 υψηλότερα bit).



Πειραματιστείτε με τα παρακάτω ζευγάρια δυαδικών αριθμών και αφού συμπληρώσετε τα αθροίσματα συγκρίνετέ τα με τα αποτελέσματα που δίνει το πρόγραμμα.




## Μέρος 3ο

## Πολλαπλασιασμός και Διαίρεση

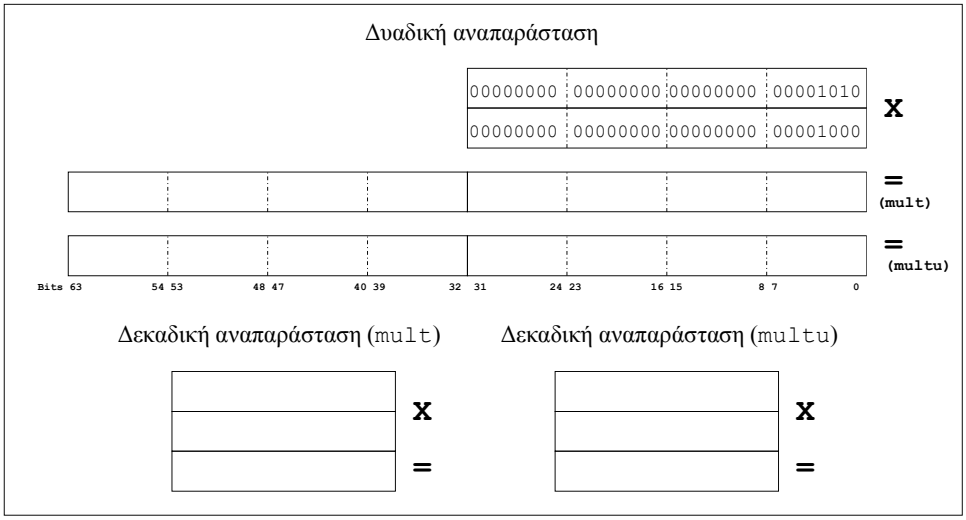
### Άσκηση 2.7

Σε αυτή την άσκηση θα φανεί η διαφοροποίηση των δύο πραγματικών εντολών πολλαπλασιασμού ακεραίων `mult` (multiply with sign) και `multu` (multiply unsigned) που διαθέτει ο επεξεργαστής MIPS. Η διαφορά τους βρίσκεται στο διαφορετικό τρόπο με τον οποίο μεταφράζουν τους τελεστέους της πράξης του πολλαπλασιασμού αλλά και η διαφορά στο αποτέλεσμα που δίνουν οι δύο εντολές.

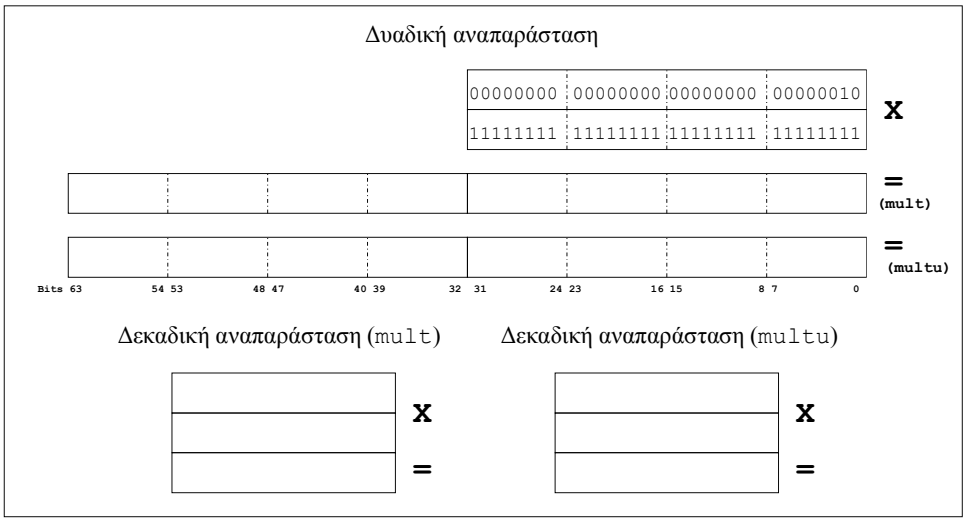
Δοκιμάστε τις δύο εντολές για κάθε ένα από τα παρακάτω ζευγάρια αριθμών και συμπληρώστε το αποτέλεσμα που δίνει το πρόγραμμα σαν δυαδικό αριθμό. Περάστε τους αριθμούς με την οδηγία `.word` στη μνήμη του επεξεργαστή σαν δεκαεξαδικούς αριθμούς. Χρησιμοποιήστε τις εντολές `mflo` (move from lo) και `mfhi` (move from hi) για να γράψετε τα αποτελέσματα σε έναν καταχωρητή γενικού σκοπού και κατόπιν στη μνήμη.

Κατόπιν συμπληρώστε τους πίνακες με την δεκαδική αναπαράσταση για κάθε μία από τις δύο περιπτώσεις πολλαπλασιασμού για να διαπιστώσετε ποιους δεκαδικούς αριθμούς πολλαπλασίασε το πρόγραμμα και τι δεκαδικό αποτέλεσμα έδωσε.

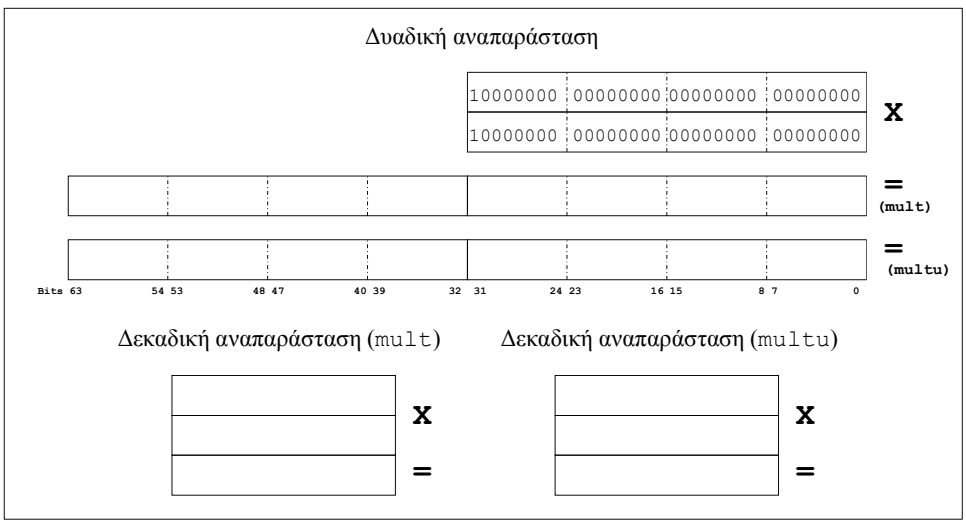
(α)


Δυαδική αναπαράσταση
00000000 00000000 00000000 00001010
x
00000000 00000000 00000000 00001000
=
(mult)
=
(multu)
Bits 63 54 53 48 47 40 39 32 31 24 23 16 15 8 7 0
Δεκαδική αναπαράσταση (mult)
Δεκαδική αναπαράσταση (multu)
x
=
x
=

(β)


Δυαδική αναπαράσταση
00000000 00000000 00000000 00000010
x
11111111 11111111 11111111 11111111
=
(mult)
=
(multu)
Bits 63 54 53 48 47 40 39 32 31 24 23 16 15 8 7 0
Δεκαδική αναπαράσταση (mult)
Δεκαδική αναπαράσταση (multu)
x
=
x
=

(γ)


Δυαδική αναπαράσταση
10000000 00000000 00000000 00000000
x
10000000 00000000 00000000 00000000
=
(mult)
=
(multu)
Bits 63 54 53 48 47 40 39 32 31 24 23 16 15 8 7 0
Δεκαδική αναπαράσταση (mult)
Δεκαδική αναπαράσταση (multu)
x
=
x
=

**Άσκηση 2.8**

Γράψτε ένα πρόγραμμα που κάθε φορά θα δέχεται από το χρήστη μέσω του παραθύρου της κονσόλας δύο ακεραίους προσημασμένους αριθμούς (χρησιμοποιήστε την κλήση `read_int`) και θα τους διαιρεί με την εντολή προσημασμένης διαίρεσης `div`. Τα αποτελέσματα (πηλίκο και υπόλοιπο) να εμφανίζονται στο ίδιο παράθυρο (κλήση συστήματος `print_int`). Χρησιμοποιήστε σαν πρότυπο και συμπληρώστε το πρόγραμμα `lab2_8a.s` που βρίσκεται στην ιστοσελίδα του μαθήματος και περιέχει τις απαραίτητες κλήσεις εισόδου και εξόδου. Το πρόγραμμα είναι το ακόλουθο.

```
################################################
#                                              #
# lab2_8a.s                                    #
# 32 / 32 bit division                         #
#                                              #
################################################

################################################
#                                              #
#           text segment                             #
#                                              #
################################################

      .text
            .globl __start
__start:
# Input
      la $a0,dividend
      li $v0,4
      syscall                 # prompt for dividend
      li $v0,5
      syscall                 # read dividend
      move $t0,$v0            # dividend in $t0

      la $a0,divisor
      li $v0,4
      syscall                 # prompt for divisor
      li $v0,5
      syscall                 # read divisor
      move $t1,$v0            # divisor in $t1

########
########    Calculations
########

#Output
      la $a0,quotient
      li $v0,4
      syscall                 # display "quotient is :"
      move $a0,$t4
      li $v0,1
      syscall                 # display quotient
      la $a0,endl
      li $v0,4
      syscall                 # newline
```

```
        la $a0,remainder
        li $v0,4
        syscall                 # display "remainder is :"
        move $a0,$t5
        li $v0,1
        syscall                 # display remainder
        la $a0,endl
        li $v0,4
        syscall                 # newline

        li $v0,10
        syscall                 # exit


################################################
#                                              #
#                  data segment                          #
#                                              #
################################################

            .data
dividend:   .asciiz "Enter dividend:"
divisor:    .asciiz "Enter divisor:"
endl:       .asciiz "\n"
quotient:   .asciiz "quotient is :"
remainder:  .asciiz "remainder is :"

################################################
#                                              #
# End of File                                        #
#                                              #
################################################
```

Πειραματιστείτε με τους παρακάτω δεκαδικούς αριθμούς, ελέγξτε την ορθότητα των αποτελεσμάτων και διαπιστώστε αν λαμβάνετε κάποιο μήνυμα σφάλματος από τον προσομοιωτή ή δημιουργείται κάποια εσωτερική διακοπή.

| Διαιρετέος | Διαιρέτης | Πηλίκο | Υπόλοιπο |
|---:|---:| | |
| 13 | 3 | | |
| 13 | -3 | | |
| -13 | 3 | | |
| -13 | -3 | | |
| 13 | 0 | | |
| -2147483648 | 1 | | |
| -2147483648 | 1000 | | |

# Εργαστήριο Αρχιτεκτονικής Υπολογιστών Ι
# Προσομοιωτής QtSpim του Μικροεπεξεργαστή MIPS

## Εργαστηριακή Άσκηση 3
### Έλεγχος Ροής Προγράμματος

**Εισαγωγή**

Αντικείμενο αυτού του εργαστηρίου είναι η χρήση των εντολών αποφάσεων που διαθέτει ο MIPS σε συνδυασμό με εντολές διακλαδώσεων υπό συνθήκη (conditional branches) και απλών διακλαδώσεων, προκειμένου να επιτευχθούν γνωστές δομές ελέγχου ροής των γλωσσών προγραμματισμού υψηλού επιπέδου, όπως διακλαδώσεις if – then – else, case, βρόχοι for, while, until κ.λπ.

**Μέρος 1ο**

**Διακλαδώσεις Υπό Συνθήκη**

**Άσκηση 3.1**

Σε αυτή την άσκηση θα δημιουργήσετε μία δομή switch-case. Το ζητούμενο είναι η εισαγωγή ενός ακεραίου από το πληκτρολόγιο και η εμφάνιση μηνυμάτων στο παράθυρο console που θα διαχωρίζουν τις ακόλουθες περιπτώσεις: αν ο ακέραιος που δόθηκε είναι άρτιος (διαιρείται με το 2), διαιρείται με το 3, διαιρείται με το 5 ή τίποτε από τα προηγούμενα. Θα υλοποιήσετε δύο παραλλαγές:

(α) Θα εμφανίζονται μηνύματα για οποιαδήποτε περίπτωση διαιρετότητας, π.χ. αν δόθηκε ο 10 θα πρέπει να εμφανίζεται ότι διαιρείται με το 2 αλλά και με το 5. Εδώ ουσιαστικά πρόκειται για τη δομή switch-case της γλώσσας C χωρίς χρήση break ανάμεσα στους κλάδους. Δηλαδή είναι δυνατόν να εκτελεστούν παραπάνω από ένας κλάδοι κάθε φορά.

Υπόδειξη: Χρησιμοποιήστε την εντολή `bne`.

(β) Θα εμφανίζεται μήνυμα μόνο για τον μικρότερο αριθμό από τους 2, 3 ή 5 που ισχύει η διαιρετότητα, π.χ. για τον αριθμό 10 θα εμφανίζεται μόνο το μήνυμα ότι διαιρείται με το 2. Τώρα πρόκειται για τη δομή switch-case της γλώσσας C με χρήση break ανάμεσα στους κλάδους. Δηλαδή μόνο ένας κλάδος εκτελείται κάθε φορά.

Υπόδειξη: Χρησιμοποιήστε τις εντολές `bne` και `jr`.

## Μέρος 2ο

## Βρόχοι

### Άσκηση 3.2

Η απόσταση Hamming (Hamming distance) μεταξύ δύο δυαδικών λέξεων είναι ο συνολικός αριθμός των θέσεων που διαφέρουν τα bit τους. Για παράδειγμα η απόσταση Hamming των λέξεων (των 8 bit) 11001010 και 11010010 είναι 2 γιατί οι δύο λέξεις έχουν διαφορετικά bit στις θέσεις 3 και 4 (ξεκινώντας την αρίθμηση των bit με 0 από τα δεξιά).

Γράψτε ένα πρόγραμμα που να υπολογίζει τη Hamming απόσταση μεταξύ δύο λέξεων των 32 bit που είναι αποθηκευμένες στη μνήμη.

Υπόδειξη: Χρησιμοποιήστε τις εντολές `xor`, `srl` και μία εντολή για διακλάδωση.

### Άσκηση 3.3

Δίνεται το παρακάτω πρόγραμμα σε συμβολική γλώσσα MIPS (Το πρόγραμμα θα το βρείτε έτοιμο στην ιστοσελίδα του μαθήματος με όνομα `lab3_3.s`). Η λειτουργία του είναι να διαβάζει τη συμβολοσειρά που είναι αποθηκευμένη στη μνήμη (οδηγία `.asciiz`) και να την αντιγράφει σε μία άλλη θέση μνήμης (οδηγία `.space 80`) από την αρχή της μέχρι την πρώτη εμφάνιση ενός χαρακτήρα που φαίνεται στη γραμμή `li $s0,'t'`. Ουσιαστικά πρόκειται για ένα βρόχο τύπου while με συνθήκη «*while (ο τρέχων χαρακτήρας που διαβάζεται δεν είναι ο x)*», όπου στο παράδειγμα ο x είναι το πεζό αγγλικό γράμμα `'t'`.

Το ζητούμενο είναι να κάνετε δύο μετατροπές του προγράμματος:

(α) Ο βρόχος while να μετατραπεί σε: «*while (ο τρέχων χαρακτήρας που διαβάζεται δεν είναι ο x **ΚΑΙ** δεν έχουν διαβαστεί πάνω από ν χαρακτήρες )*». Το ν είναι μία σταθερά που θα αποθηκεύεται σε έναν καταχωρητή.

(β) Ο βρόχος while να μετατραπεί σε: «*while (ο τρέχων χαρακτήρας που διαβάζεται δεν είναι ο x **Ή** δεν έχουν διαβαστεί πάνω από ν χαρακτήρες )*». Εδώ ν είναι πάλι μία σταθερά που θα αποθηκεύεται σε έναν καταχωρητή.

Υπόδειξη: Μπορείτε να χρησιμοποιήσετε ψευδοεντολές για τα υπό συνθήκη άλματα που θα χρειαστείτε, όπως για παράδειγμα `bge`, `bgt`, `ble`, `blt` κ.λπ.

```
################################################
# lab3_3.s                                     #
################################################
      .text
      .globl __start
__start:
      li $t1,0                # counter for string
      li $s0,'t'              # character to end copy
while:lbu $t0,string($t1)           # load a character
      beq $t0,$s0,end         # if character to end copy then exit loop
      sb $t0,copy($t1)        # copy character
```

```
      addi $t1,$t1,1          # increment counter
      j while                 # repeat while loop
end:  li $t2,0
      sb $t2,copy($t1)        # append end character to copied string
      la $a0,copy             # display copy
      li $v0,4
      syscall
      li $v0,10               # exit
      syscall

      .data
string:           .asciiz "Mary had a little lamb"
copy:       .space 80
################################################
#end of program
################################################
```

**Άσκηση 3.4**

Γράψτε ένα πρόγραμμα που θα δέχεται σαν είσοδο από το χρήστη μία ακολουθία χαρακτήρων και θα τυπώνει στην έξοδο την ίδια ακολουθία με κεφαλαία γράμματα (όπου υπάρχουν χαρακτήρες ‘a’-‘z’ θα αντικαθίστανται με τους ‘A’-‘Z’). Οι χαρακτήρες ‘a’-‘z’ έχουν κωδικούς ASCII 97-122 σε δεκαδικό σύστημα και οι ‘A’-‘Z’ έχουν κωδικούς 65-90.

Υπόδειξη : Χρησιμοποιήστε ένα βρόχο while που θα είναι ενεργός όσο ο χαρακτήρας που διαβάζεται από το string δεν έχει κωδικό 0, δηλαδή τον κωδικό που δηλώνει το τέλος του string.

# Εργαστήριο Αρχιτεκτονικής Υπολογιστών Ι
# Προσομοιωτής QtSpim του Μικροεπεξεργαστή MIPS

## Εργαστηριακή Άσκηση 4
Διαδικασίες και στοίβα

### Εισαγωγή

Σε αυτές τις εργαστηριακές ασκήσεις θα δειχθεί η χρήση της στοίβας για τη δημιουργία διαδικασιών (procedures) και θα αναπτυχθεί βήμα-βήμα ένα πρόγραμμα υπολογισμού της ακολουθίας Fibonacci με απλή αναδρομή.

### Μέρος 1ο
### Απλή κλήση και συμβάσεις καταχωρητών

#### Άσκηση 4.1

Δημιουργήστε μία διαδικασία που θα λαμβάνει τέσσερις ακεραίους μέσω των καταχωρητών `$a0`, `$a1`, `$a2` και `$a3` και θα επιστρέφει τον μικρότερο μέσω του καταχωρητή `$v0` και τον μεγαλύτερο μέσω του καταχωρητή `$v1`. Χρησιμοποιήστε για τυχόν ενδιάμεσα αποτελέσματα μόνο τους καταχωρητές `$s` καθώς και τη σύμβαση προγραμματισμού του MIPS που θέλει τους `$s` καταχωρητές να αποθηκεύονται στη στοίβα σε περίπτωση που γίνεται χρήση τους.

Στο κυρίως πρόγραμμα φορτώστε τους καταχωρητές `$a0` … `$a3` με τους τέσσερις αριθμούς από τη μνήμη, και επίσης φορτώστε τους `$s` που θα χρησιμοποιήσετε με κάποιες τυχαίες τιμές έτσι ώστε να ελέγξετε ότι όντως διατηρούνται τα περιεχόμενά τους και μετά την επιστροφή από την κλήση της διαδικασίας.

Τρέξτε το πρόγραμμα με βηματικό τρόπο για να παρακολουθείτε τη στοίβα να μεγαλώνει και να μικραίνει (δηλαδή να προστίθενται σε αυτή και να αφαιρούνται από αυτή στοιχεία). Παρακολουθήστε τα περιεχόμενα του καταχωρητή δείκτη στοίβας `$sp`.

Βασιστείτε στο παρακάτω πρόγραμμα με ονομασία `lab4_1a.s` που θα το βρείτε στην ιστοσελίδα του μαθήματος και συμπληρώστε το. Αυτό είναι το κυρίως πρόγραμμα που αρχικοποιεί τις παραμέτρους που θα περαστούν στη διαδικασία και εμφανίζει στην οθόνη τα αποτελέσματά της. Συμπληρώστε τις απαραίτητες γραμμές κώδικα για την διαδικασία και την κλήση της.

```
################################################
#                                              #
# lab4_1a.s                                    #
# stack exercise 1      (to be completed)      #
#                                              #
################################################
      .text
            .globl __start
__start:

# start of main program
      li $a0,-10        # Initialize variables
      li $a1,-30        #
      li $a2,120        #
      li $a3,200        #

      move $t0,$v0
      move $t1,$v1
      la $a0,max
      li $v0,4
      syscall                 # display "Max is :"
      move $a0,$t0
      li $v0,1
      syscall                 # display max
      la $a0,endl
      li $v0,4
      syscall                 # display end of line
      la $a0,min
      li $v0,4
      syscall                 # display "Min is :"
      move $a0,$t1
      li $v0,1
      syscall                 # display min
      la $a0,endl
      li $v0,4
      syscall                 # display end of line
      li $v0,10
      syscall                 # exit
# end of main program

# start of procedure
#
# end of procedure

            .data
max:        .asciiz "Max is : "
min:        .asciiz "Min is : "
endl:       .asciiz "\n"
################################################
#                                              #
# End of program                               #
#                                              #
################################################
```

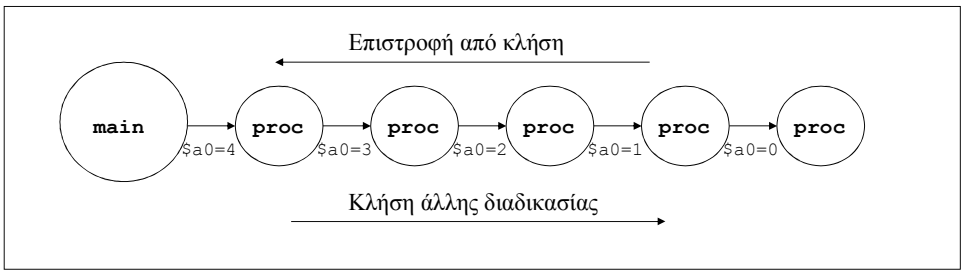
## Μέρος 2ο

## Αναδρομικές Κλήσεις

### Άσκηση 4.2

Γράψτε ένα πρόγραμμα που θα αποτελείται από δύο τμήματα, το κυρίως πρόγραμμα και μία διαδικασία που θα καλεί αναδρομικά τον εαυτό της. Το κυρίως πρόγραμμα θα ζητάει από το χρήστη έναν ακέραιο αριθμό και κατόπιν θα καλεί μία διαδικασία περνώντας της αυτόν τον αριθμό. Η διαδικασία θα δέχεται ένα όρισμα (τον ακέραιο που περνάει κατά την πρώτη κλήση) θα εξετάζει την τιμή και αν είναι μηδέν θα επιστρέφει. Αν δεν είναι μηδέν, θα τον μειώνει κατά ένα και θα καλεί πάλι τον εαυτό της περνώντας της σαν όρισμα τον μειωμένο κατά 1 ακέραιο μέσω του καταχωρητή `$a0`. Σε κάθε περίπτωση (μηδέν ή όχι) θα εμφανίζει στην οθόνη την τιμή του αριθμού που έχει παραλάβει. Σχηματικά η διαδοχή των κλήσεων είναι η παρακάτω.

Είναι απαραίτητο η παράμετρος `$a0` να αποθηκεύεται στη στοίβα; Δώστε εξήγηση.




Βασιστείτε στο παρακάτω πρόγραμμα με ονομασία `lab4_2a.s` που θα το βρείτε στην ιστοσελίδα του μαθήματος. Σε αυτό συμπληρώστε τις απαραίτητες γραμμές κώδικα για την διαδικασία και την κλήση της.

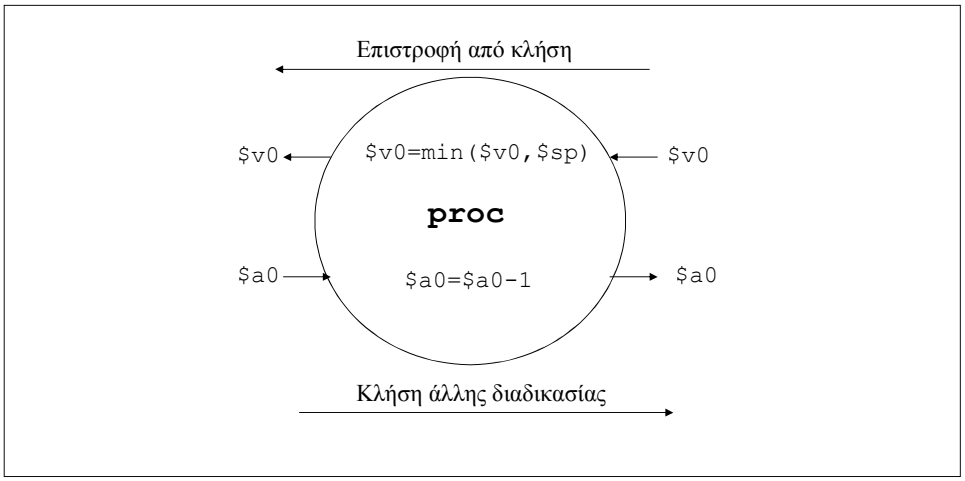
```
################################################
#                                              #
# lab4_2a.s                                    #
# stack exercise 2      (to be completed)      #
#                                              #
################################################
      .text
            .globl __start
__start:

# start of main program

      la $a0,prompt
      li $v0,4
      syscall                 # display "Enter integer number :"
      li $v0,5
      syscall                 # read integer
      move $t0,$v0
      la $a0,endl
      li $v0,4
      syscall                 # display end of line
      move $a0,$t0
```

```
      li $v0,10
      syscall                 # exit
# end of main program

# start of procedure
#
# end of procedure

            .data
prompt:           .asciiz "Enter integer number :"
endl:       .asciiz "\n"
###############################################
#                                             #
# End of program                              #
#                                             #
###############################################
```

**Άσκηση 4.3**

Η προηγούμενη άσκηση περιείχε μία διαδικασία που δεχόταν μία παράμετρο (`$a0`), χωρίς όμως να επιστρέφει κανένα αποτέλεσμα. Τώρα θα εμπλουτίσετε το πρόγραμμά σας έτσι ώστε η διαδικασία να επιστρέφει έναν ακέραιο αριθμό που θα δηλώνει την ελάχιστη διεύθυνση του δείκτη στοίβας που έχει χρησιμοποιήσει η διαδικασία ή κάποια άλλη που κλήθηκε από αυτήν. Ακολουθήστε τη σύμβαση που έχει ο MIPS για τη χρήση καταχωρητών επιστροφής αποτελεσμάτων, π.χ. χρησιμοποιήστε τον `$v0`. Σχηματικά η διαδικασία θα εκτελεί τη λειτουργία που φαίνεται στο παρακάτω σχήμα.

Το κυρίως πρόγραμμα θα παραλαμβάνει το αποτέλεσμα από τις διαδοχικές κλήσεις, δηλαδή πόσο «χαμηλά» έφτασε η στοίβα και θα εμφανίζει το ποσό τη μνήμης που χρησιμοποιήθηκε για τη στοίβα, γνωρίζοντας την αρχική τιμή του `$sp` πριν να γίνει οποιαδήποτε κλήση διαδικασίας.




**Άσκηση 4.4**

Το επόμενο βήμα είναι ο εμπλουτισμός των προηγούμενων ασκήσεων έτσι ώστε να γίνεται υπολογισμός του *n*-οστού όρου της ακολουθίας Fibonacci. Η ακολουθία Fibonacci είναι αναδρομική, κάτι που κάνει τον υπολογισμό της να ταιριάζει με τη χρήση αναδρομικών κλήσεων διαδικασιών. Η ακολουθία Fibonacci δίνεται από τον ακόλουθο τύπο :

$$f(n)=\begin{cases} f(n-1)+f(n-2) & , \quad n>1 \\ 1 & , \quad n=1 \\ 0 & , \quad n=0 \end{cases}$$

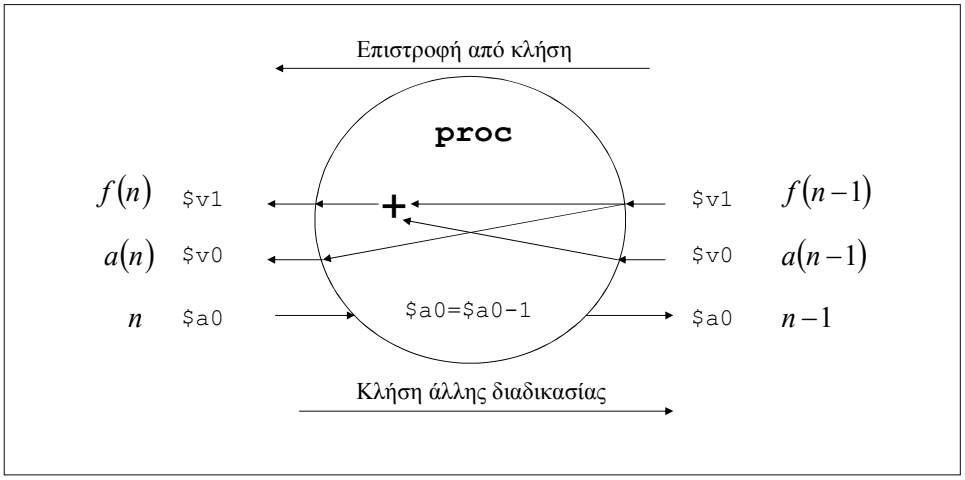
Αν χρησιμοποιηθεί ο παραπάνω τύπος, μέσα σε μία διαδικασία θα πρέπει να υπάρχουν δύο κλήσεις προς τον εαυτό της, μία για *n*–1 και άλλη μία για *n*–2, κάτι που δημιουργεί ένα δεντρικό σχήμα στα διάφορα στιγμιότυπα των διαδικασιών. Για να διατηρήσουμε το γραμμικό σχήμα που ήδη έχουμε ξεκινήσει από την πρώτη άσκηση και για λόγους καλύτερης αποδοτικότητας θα χρησιμοποιήσουμε τους ισοδύναμους τύπους για τον υπολογισμό της ακολουθίας Fibonacci που δίνονται παρακάτω:

$$f(n)=\begin{cases} f(n-1)+a(n-1) & , \quad n>1 \\ 1 & , \quad n=1 \\ 0 & , \quad n=0 \end{cases} \quad \text{και} \quad a(n)=\begin{cases} f(n-1) & , \quad n>1 \\ 1 & , \quad n=1 \\ 1 & , \quad n=0 \end{cases}$$

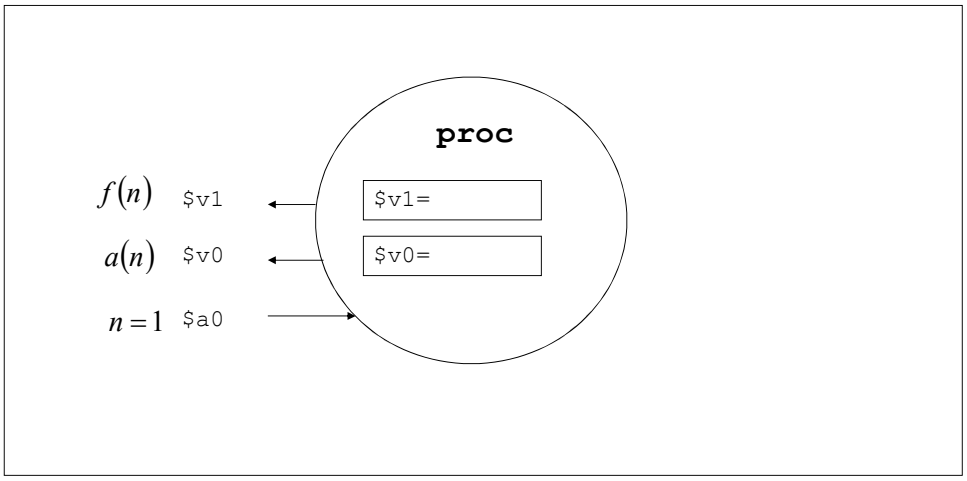
Έτσι τώρα μία διαδικασία καλεί μόνο μία φορά τον εαυτό της (για *n*–1). Οι παράμετροι εισόδου, τα αποτελέσματα και οι λειτουργίες της φαίνονται στο παρακάτω σχήμα (με την εξαίρεση οριακών συνθηκών για *n*=1 και *n*=0).

Χρησιμοποιήστε την παρακάτω μορφή διαδικασίας για να υπολογίσετε τον *n*-οστό όρο της ακολουθίας και κάντε εμφάνιση του αποτελέσματος από το κυρίως πρόγραμμα.

Στη σύμβαση προγραμματισμού που χρησιμοποιείται από τον QtSpim για την έξοδο αποτελεσμάτων από μία διαδικασία παρέχονται μόνο δύο καταχωρητές (οι `$v0` και `$v1`). Χρησιμοποιήστε τους για να την έξοδο των $f(n)$ και $a(n)$.




Πριν γράψετε το πρόγραμμα συμπληρώστε στο παρακάτω σχήμα τη λειτουργία της οριακής κλήσης της διαδικασίας (δηλαδή για $n$=1).



# Εργαστήριο Αρχιτεκτονικής Υπολογιστών Ι
# Προσομοιωτής QtSpim του Μικροεπεξεργαστή MIPS

## Εργαστηριακή Άσκηση 5
### Αριθμητική Κινητής Υποδιαστολής

**Εισαγωγή**

Στο πρώτο μέρος της άσκησης αυτής θα εξεταστεί ο τρόπος εισόδου και εξόδου αριθμών κινητής υποδιαστολής απλής ακρίβειας, οι εντολές μετακίνησης των αριθμών, ο τρόπος αναπαράστασης και οι περιορισμοί που υπάρχουν ως προς την ακρίβεια. Επίσης θα γίνουν κάποιοι πειραματισμοί με πράξεις μεταξύ αριθμών κινητής υποδιαστολής για ειδικές περιπτώσεις, και θα φανεί η ανάγκη για μετατροπή ακεραίων σε αριθμούς κινητής υποδιαστολής. Στο δεύτερο μέρος θα αναπτυχθεί ένα πιο σύνθετο πρόβλημα υπολογισμού της συνάρτησης του εκθετικού $e^x$.

**Μέρος 1ο**

**Αναπαράσταση - Ακρίβεια – Πράξεις – Μετατροπές**

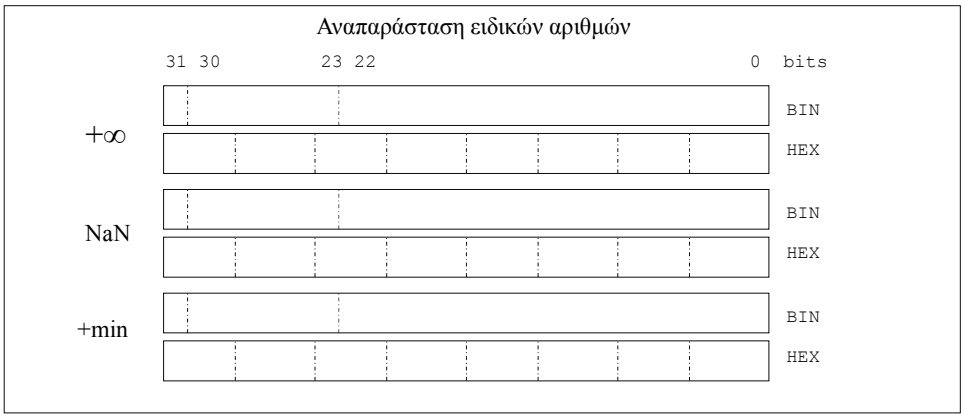
**Άσκηση 5.1**

Γράψτε ένα πρόγραμμα που θα διαβάζει από το παράθυρο της κονσόλας έναν αριθμό απλής ακρίβειας με τη χρήση της κλήσης συστήματος `read_float`. Κατόπιν χρησιμοποιήστε την κλήση `print_float` για να τον εμφανίσετε στην οθόνη. Θα χρειαστείτε την εντολή `mov.s` μεταφοράς αριθμών κινητής υποδιαστολής γιατί η κάθε κλήση χρησιμοποιεί διαφορετικό καταχωρητή για να δώσει και να πάρει το αποτέλεσμα.

(α) Πειραματιστείτε με αριθμούς που είναι δυνάμεις ή αθροίσματα δυνάμεων του 2, π.χ. -0.75, 1.0, 1.5, 0.0000152587890625 ($=1/2^{16}$) κ.λπ.

(β) Πειραματιστείτε με τυχαίους αριθμούς, π.χ. 0.775, 0.1, 1.23456789 κ.λπ.

(γ) Μετατρέψτε το πρόγραμμά σας, έτσι ώστε να εισάγετε απ' ευθείας σε έναν καταχωρητή το περιεχόμενό του (δεξί κλικ επάνω στον καταχωρητή και επιλογή → Change Register Contents από το μενού που εμφανίζεται). Βρείτε την αναπαράσταση των παρακάτω αριθμών σε μορφή κινητής υποδιαστολής και εισάγετέ τους ως δεκαεξαδικούς αριθμούς. Για την περίπτωση NaN, επιλέξτε έναν οποιοδήποτε συνδυασμό ο οποίος να δίνει έναν μη έγκυρο αριθμό της αναπαράστασης. Η περίπτωση +min είναι ο μικρότερος θετικός μη κανονικοποιημένος αριθμός.




**Άσκηση 5.2**

Γράψτε ένα πρόγραμμα που να ορίζει με τη χρήση του τμήματος δεδομένων έξι σταθερές : 0.0, +∞, -∞, +NaN, και δύο αριθμούς, έναν θετικό x και έναν αρνητικό y. Το πρόγραμμα θα πρέπει να εκτελεί τις πράξεις που φαίνονται στον παρακάτω πίνακα και να εμφανίζει τα αποτελέσματά τους. Συμπληρώστε τον παρακάτω πίνακα σύμφωνα με τα αποτελέσματα που θα πάρετε.

| **Πράξης** | **Αποτέλεσμα** |
|---|---|
| x · y | |
| x · (-∞) | |
| y / 0 | |
| 0 / 0 | |
| 0 · (+∞) | |
| (+∞) / (-∞) | |
| (+∞) + (-∞) | |
| x + NaN | |

**Άσκηση 5.3**

Πολλές φορές χρειάζεται να γίνουν πράξεις μεταξύ ακεραίων και πραγματικών αριθμών ή για κάποιο λόγο οι ακέραιοι να αναπαρασταθούν σαν πραγματικοί αριθμοί. Ένα παράδειγμα είναι ο υπολογισμός της συνάρτησης του παραγοντικού. Αν γίνει ο υπολογισμός με πράξεις ακεραίων πολύ γρήγορα φτάνουμε στα όρια της αναπαράστασης ενός ακεραίου 32 bit. Για παράδειγμα, το 13! = 6.227.020.800 δεν μπορεί να αναπαρασταθεί σε 32 bit.

(α) Το πρόγραμμα που δίνεται παρακάτω υπολογίζει με χρήση ακεραίων και εμφανίζει όλα τα παραγοντικά μέχρι το *n*! Θα το βρείτε έτοιμο στην ιστοσελίδα του μαθήματος με όνομα `lab5_3a.s`. Εκτελέστε το και παρατηρήστε τα αποτελέσματα για *n*>=13.

```
################################################
#                                              #
# lab5_3a.s                                    #
# Integer factorial                            #
#                                              #
```

```
################################################
      .text
            .globl __start
__start:
      la $a0,n
      lw $t0,0($a0)      # $t0 = n
      li $t2,1           # $t2 index i=1..n
      li $t1,1           # $t1 contains i!
loop: mul $t1,$t1,$t2
      move $a0,$t2
      li $v0,1
      syscall            # display i
      la $a0,msg1
      li $v0,4
      syscall            # display "! is :"
      move $a0,$t1
      li $v0,1
      syscall            # display i!
      la $a0,endl
      li $v0,4
      syscall            # print end of line
      addi $t2,$t2,1     # i=i+1
      ble $t2,$t0,loop   # repeat if i<=n
      li $v0,10          # exit
      syscall
            .data
n:          .word 25
msg1:       .asciiz "! is :"
endl:       .asciiz "\n"

################################################
#                                              #
# End of program                               #
#                                              #
################################################
```

(β) Αλλάξτε το προηγούμενο πρόγραμμα έτσι ώστε ο υπολογισμός να γίνεται με αριθμητική κινητής υποδιαστολής απλής ακρίβειας. Θα χρειαστείτε πιθανόν τις εξής εντολές: `mtc1` (move to co-processor 1) για μεταφορά περιεχομένου από καταχωρητή γενικής χρήσης σε καταχωρητή κινητής υποδιαστολής, `cvt.s.w` για μετατροπή από αναπαράσταση ακεραίου σε αναπαράσταση κινητής υποδιαστολής και `mul.s` για πολλαπλασιασμό αριθμών κινητής υποδιαστολής.

Εκτελέστε το πρόγραμμα για $n$>20 και παρατηρήστε τα αποτελέσματα. Συγκρίνετε τα αποτελέσματα για μεγάλα $n$ με αυτά που δίνει η αριθμομηχανή (Calculator) των Windows.

## Μέρος 2ο

## Σύνθετοι Υπολογισμοί

### Άσκηση 5.4

Ο υπολογισμός της εκθετικής συνάρτησης $e^x$ μπορεί να γίνει με προσέγγιση χρησιμοποιώντας τη σειρά Taylor:

$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ , $\forall x \in \Re$. Γράψτε ένα πρόγραμμα υπολογισμού της εκθετικής συνάρτησης προσεγγίζοντάς την με χρήση των $k$ πρώτων όρων της παραπάνω σειράς. Ο αριθμός $k$ θα είναι μία σταθερά μέσα στο πρόγραμμα (π.χ. $k$=8, δηλαδή θα χρησιμοποιηθούν οι 8 πρώτοι όροι) και ο $x$ θα δίνεται από τον χρήστη μέσα στο παράθυρο της κονσόλας.

Χρησιμοποιήστε αριθμητική κινητής υποδιαστολής απλής ακρίβειας, όπως και στην Άσκηση 5.3. Θα χρειαστείτε επί πλέον τις εντολές `mov.s` για μετακίνηση περιεχομένων μεταξύ καταχωρητών κινητής υποδιαστολής και `add.s` και `div.s` για πρόσθεση και διαίρεση, αντίστοιχα.