

Μάθημα:
Αρχιτεκτονική Υπολογιστών (2^ο εξάμηνο)

Διδάσκων:
Δημήτρης Γκιζόπουλος

Εργασία:
Υλοποίηση ενός προγράμματος
επεξεργασίας 20 ακεραίων αριθμών σε
assembly για επεξεργαστή MIPS και
ολιγοσέλιδη τεκμηρίωση της δομής του

Όνομα: Κουρεμένος Νίκος
Α.Μ.: Π / 02161

email: nkour@rainbow.cs.unipi.gr
σταθερό: 210 8815306
κινητό: 6942253282

Εισαγωγή

Σαν αρχή θα ήθελα να αναφέρω πως η assembly είναι αρκετά όμορφη γλώσσα, δύσκολα όμως διαβάζεται από τον τρίτο. Για αυτό τον σκοπό γέμισα τον κώδικα μου με σχόλια, τα οποία μου πήραν τον ίδιο χρόνο όσο και η σύνταξη της εντολής που τα περιέχει! Όμως πραγματικά δεν θα μπορούσε να γίνει διαφορετικά, καθώς πρόκειται για γλώσσα χαμηλού επιπέδου που είναι αρκετά κωδικοποιημένη.

Κατά την υλοποίηση του προγράμματος χρησιμοποίησα βέβαια pseudo-instructions, όμως προσπαθώ να αποφεύγω την πολύ απλοποιημένη γραφή (έτσι γράφω `addi $t0,$t0,1` και όχι `addi $t0,1` ή `add $t0,1`). Επίσης επιδίωξα το πρόγραμμα να είναι όσο γίνεται πιο φιλικό προς τον χρήστη (βέβαια για έναν χρήστη που είναι συνηθισμένος μόνο σε GUI, ο τρόπος χρήσης παραμένει δύσκολος, παρά τις προσπάθειες μου!).

Ειδική μνεία για τον αριθμό 0

Ο αριθμός μηδέν είναι ένας αριθμός που ίσως δώσει ο χρήστης. Παράλληλα είναι όμως και ο null χαρακτήρας. Στη διαδικασία της εκσφαλμάτωσης (debugging) ήταν αρκετά χρήσιμο από τους 20 αριθμούς που δίνει ο χρήστης, να μην είναι κανείς 0, για να αποφεύγεται η σύγχυση μεταξύ πραγματικού 0 και null.

Αν βρεθούμε (λόγω λογικού λάθους) σε λαθεμένη διεύθυνση μνήμης και φορτώσουμε έναν αριθμό υπάρχει περίπτωση να περιμένουμε το «δικό μας» 0 (το έχουμε δώσει για παράδειγμα 8 φορές στους 20 αριθμούς) και να πάρουμε ένα 0 αλλά όχι το «δικό μας» αλλά null. Η αποφυγή του 0 ως ένας από τους 20 αριθμούς με βοήθησε αρκετά στο debugging. Φυσικά, το πρόγραμμα χειρίζεται άψογα τον 0 όπως και κάθε άλλο αριθμό!

Ανάλυση της δομής του προγράμματος

Στη συνέχεια παραθέτω αποσπάσματα κώδικα στα οποία εξηγώ την λειτουργία τους στο συνολικό πρόγραμμα. Ολόκληρος ο κώδικας δίνεται χωριστά μιας και στα αποσπάσματα έχουν αφαιρεθεί πολλές γραμμές κώδικα. Ο χρωματισμένος με κόκκινο χρώμα κώδικας (όπου υπάρχει) επισημαίνει τις βασικότερες λειτουργίες που επιτελούνται στο εκάστοτε υποπρόγραμμα που εξετάζεται.

```
.text
.globl __start
__start:
    la $a0,intro_txt           # επεξηγηματικά σχόλια προς τον χρήστη
    li $v0,4
    syscall

    li $v0,5
    syscall

    move $fp,$sp              # δημιουργώ stack frame
```

Προτιμώ να μην πειράξω τον stack pointer και έτσι δημιουργώ ένα frame stack στο οποίο θα μπουν οι 20 αριθμοί (λέξεις) που δίνει ο χρήστης.

```
main_program:
    li $t0,1                   # $t0 είναι ο μετρητής
    addi $fp,$fp,-80           # 20 words = 80 bytes!

    la $a0,newline
    li $v0,4
```

```
syscall
```

Στην συνέχεια η **ask_numbers** χρησιμοποιεί ένα τέχνασμα ώστε ο χρήστης να βλέπει «Δώσε τον 1ο αριθμό» «Δώσε τον 2ο αριθμό» κλπ. Ο χρήστης δίνει τον κάθε αριθμό ο οποίος αποθηκεύεται στο **frame stack** και στην συνέχεια αυξάνεται κατά μία λέξη ο frame pointer (\$fp) ώστε να δείχνει την θέση που θα καταχωρηθεί το επόμενο στοιχείο. Η συνθήκη της επανάληψης αυτής, αληθεύει 20 φορές. Την 21 πρώτη φορά το κατάλληλο branch στέλνει τον Program Counter (PC) στην διεύθυνση (address) της **done**.

```
ask_numbers:
    beq $t0,21,done          # όσο $t0<21 κάνε loop (20
επαναλήψεις!)

    la $a0,ask_no1          # ερώτηση αριθμού μέρος A
    li $v0,4
    syscall

    move $a0,$t0            # κάνω μεταφορά για να εμφανίσω τον
μετρητή
    li $v0,1                # εμφανίζω τον μετρητή
    syscall

    la $a0,ask_no2          # ερώτηση αριθμού μέρος B
    li $v0,4
    syscall

    li $v0,5                # syscall 5 διαβάζει έναν ακέραιο
    syscall

    sw $v0,0($fp)           # τον αποθηκεύω στο stack frame
    addi $fp,$fp,4          # αυξάνω τον frame pointer ώστε μετά να
αποθηκευτεί ο επόμενος αριθμός
    addi $t0,$t0,1          # αυξάνω κατά ένα τον μετρητή των
επαναλήψεων

    j ask_numbers
```

Η **done** εμφανίζει και ζητάει από τον χρήστη να επιλέξει ποια λειτουργία θέλει να εκτελέσει πάνω στους 20 αριθμούς που μόλις έχει δώσει μέσω της **ask_numbers**

```
done:
```

```
...
...

    beq $v0,1,bubble_sort   # επιλογή 1: ταξινόμηση
    beq $v0,2,amount       # επιλογή 2: εύρεση πλήθους εμφανίσεων
    beq $v0,3,find_max     # επιλογή 3: εύρεση μέγιστου αριθμού
    beq $v0,4,find_min     # επιλογή 4: εύρεση ελάχιστου αριθμού
    beq $v0,5,avg          # επιλογή 5: εύρεση μέσου όρου των
απολύτων τιμών των αριθμών
    beq $v0,6,print        # επιλογή 6: εκτύπωση των αριθμών με -
ενδιάμεσα
    beq $v0,7,exit         # επιλογή 7: έξοδος από το πρόγραμμα

    beq $v0,666,easter_egg # easter egg
```

Στην προσπάθεια μου, το πρόγραμμα να γίνει περισσότερο φιλικό προς τον χρήστη (αφού οι command lines φοβίζουν!), έγραψα ένα σύνολο από branches τα οποία ελέγχουν το νούμερο που αντιστοιχεί στην λειτουργία που θέλει ο χρήστης να

εκτελέσει. Οι λειτουργίες είναι 7. Αν ο \$v0 ο οποίος κρατάει την επιλογή του χρήστη είναι διάφορος του 1,2,3,4,5,6,7 τότε ο PC (program counter) συνεχίζει κανονικά και πηγαίνει στην **attention1** η οποία εμφανίζει το σχετικό μήνυμα για το εύρος των επιλογών του χρήστη, και γυρίζει στην **done** όπου ξαναζητάει επιλογή από τον χρήστη...

Η **bubble_sort** εκτελείται σε περίπτωση που ο χρήστης δώσει ως επιλογή τον αριθμό 1. Για την ταξινόμηση των αριθμών σε αύξουσα σειρά, χρησιμοποιήσα την τεχνική της **φουσαλίδας** και κάνω χρήση ενός **flag** το οποίο σε περίπτωση που βρεθεί ο πίνακας ταξινομημένος σταματάει το loop. Το flag αρχικοποιείται σε 0. Η συνθήκη επανάληψης του εξωτερικού βρόχου (outside_loop) είναι η εξής:

```
bne $t5,0,show_sort
```

και ουσιαστικά εξασφαλίζει ότι ο εξωτερικός βρόχος θα επαναλαμβάνεται **μόνο** αν το flag=0. Αμέσως μετά υπάρχει η εντολή flag=1. Και βέβαια (στο αμέσως επόμενο branch) ελέγχεται και η συνθήκη $i \leq 20$.

```
bgt $s1,20,show_sort
```

Μέσα στο εσωτερικό loop, αν υπάρξει έστω και μια αντιμετάθεση τότε το flag θα πάρει και πάλι την τιμή 0 και επομένως θα συνεχιστεί και το outside loop. Αν δεν υπάρξει καμία αντιμετάθεση, τότε αυτό θα σημαίνει πως ο πίνακας είναι ταξινομημένος και επομένως δεν υπάρχει λόγος για άσκοπα loops, το flag θα παραμείνει με την τιμή 1 και ο εξωτερικός βρόχος (και άρα και ο εσωτερικός) θα τερματιστούν πριν τις 20 επαναλήψεις που είναι και οι μέγιστες. Ακολουθεί αποσπασματικός κώδικας της **bubble_sort**. Με **κόκκινο** χρώμα οι εντολές στις οποίες αναφέρομαι.

```
bubble_sort:
```

```
copy_ok:
```

```
li $s1,2           # i=2 (ώστε να αρχίσω από το 2ο αριθμό)
li $t5,0          # flag=0
la $s0,array      # δείχνει την αρχή του array
```

```
outside_loop:
```

```
bne $t5,0,show_sort # αν flag!=0 show_sort
li $t5,1            # αλλιώς, flag=1
bgt $s1,20,show_sort # όταν i>20 θα έχει γίνει ΣΙΓΟΥΡΑ
```

```
ταξινόμηση
```

```
li $s2,20          # αρχικοποίηση j=20 (θα μετράει ανάποδα)
move $t6,$s0       # κάθε φορά ο $t6 δείχνει την αρχή του
```

```
array
```

```
inside_loop:
```

```
bge $s2,$s1,check_swap # όσο j>=i ελέγχω για αντιμετάθεση
addi $s1,$s1,1         # αλλιώς i=i+1
```

```
j outside_loop        # και ξανά από την αρχή
```

```
check_swap:
```

```
lw $t0,0($t6)         # $t0 = a[j] αριθμός
lw $t1,4($t6)         # $t1 = a[j+1] αριθμός
bgt $t0,$t1,swap_now  # αν a[j]>a[j+1] τότε swap_now, αλλιώς:
```

```
j increment
```

```
swap_now:
```

```

        sw $t1,0($t6)           # τους καταχωρώ
        sw $t0,4($t6)           # όπου θα βρίσκονται ταξινομημένοι
        li $t5,0                 # flag=0, αφού έγινε αντιμετάθεση δεν είναι
ταξινομημένος ο πίνακας! (οπότε συνέχισε το loop)

increment:
        addi $t6,$t6,4           # αυξάνω κατά μία λέξη τον $t6 ώστε να
δείχνει τον επόμενο αριθμό
        addi $s2,$s2,-1         # j=j-1 (μειώνω τον μετρητή των μέσα
επαναλήψεων)

        j inside_loop

```

Η **amount** εκτελείται σε περίπτωση που ο χρήστης δώσει ως επιλογή τον αριθμό 2. Η λογική της amount είναι αρκετά απλή. Στην αρχή, ζητάει από τον χρήστη τον αριθμό για τον οποίο θα ψάξει το πλήθος των εμφανίσεων. Στην συνέχεια στην count ελέγχο τον αριθμό αυτόν με κάθε στοιχείο του frame stack (εδώ βρίσκονται οι 20 αριθμοί). Αν υπάρχει ισότητα αυξάνω τον μετρητή εμφανίσεων κατά ένα. Διαφορετικά απλά πηγαίνω στο επόμενο στοιχείο και συγκρίνω εκ νέου. Η count επαναλαμβάνεται 20 φορές (όσο και το πλήθος των αριθμών).

Για λόγους καλαισθησίας και ευχαρίστησης του χρήστη, έγραψα κάποιες γραμμές κώδικα παραπάνω οι οποίες αποφεύγουν να δώσουν ως απάντηση «ο αριθμός x βρέθηκε **0** φορές!» ή ακόμη χειρότερα «ο αριθμός x βρέθηκε **1** φορές!»

Έτσι χρησιμοποιώ δύο **branches** που ελέγχουν αν ο μετρητής εμφανίσεων (\$t1) ισούται με 0 ή 1. Αν ισούται με 0 υπάρχει σχετικό μήνυμα «ο αριθμός x δεν βρέθηκε καμία φορά!» και αν ισούται με το 1 τότε «ο αριθμός x βρέθηκε μόνο μία φορά!».

Τέλος χρησιμοποιείται πάλι το ίδιο τέχνασμα με αυτό της **ask_numbers** (στην αρχή του προγράμματος) όπου παρεμβάλλονται ASCII με καταχωρητές. Αυτό γίνεται ώστε το εποπτικό αποτέλεσμα να είναι στην **ask_numbers** «Δώσε τον i αριθμό» και εδώ στην amount «ο αριθμός x εμφανίζεται λ φορές!»

Η **find_max (αντίστοιχα find_min)** εκτελείται σε περίπτωση που ο χρήστης δώσει ως επιλογή τον αριθμό 3 (αντίστοιχα 4). Η λογική τους είναι παρόμοια και διαφέρουν μόνο στον τελεστή σύγκρισης (< ή >). Το max και min δεν αρχικοποιούνται με την τιμή 0, αφού κάτι τέτοιο είναι ένα σημαντικό λογικό λάθος μιας και το πρόγραμμα επεξεργάζεται και **αρνητικούς** αριθμούς. Έτσι καταχωρείται ο πρώτος αριθμός που διαβάζεται από τον χρήστη, και το πρόγραμμα βρίσκει απόλυτα σωστά τον μέγιστο ή τον ελάχιστο από τους 20 ακέραιους αριθμούς.

```

#-----
# find_max - εύρεση μέγιστου αριθμού (επιλογή 3)
#   t0 - ο κάθε αριθμός διαδοχικά
#   t1 - μετράει τις επαναλήψεις
#   t2 - max (μέγιστος αριθμός)
#-----

find_max:
        addi $fp,$fp,-80         # γυρίζω τον frame pointer ξανά στην
αρχή
        lw $t2,0($fp)           # αρχικά, καταχωρώ ως max τον πρώτο
αριθμό
        li $t1,0                 # μηδενίζω τον $t1

loop_max:
        addi $fp,$fp,4           # (4 bytes = 1 word)
        beq $t1,19,print_max     # στις 20 επαναλήψεις το πρόγραμμα
γνωρίζει τον μέγιστο

```

```

    addi $t1,$t1,1          # αυξάνω κατά 1 τον μετρητή
    lw $t0,0($fp)         # $t0 κρατάει προσωρινά τον i αριθμό
    bgt $t0,$t2,set_max   # αν a[i] > max τότε set_max

    j loop_max

set_max:
    move $t2,$t0          # εκχωρώ το νέο max

    j loop_max

print_max:

    ...
    ...

    j ask_after

```

Η **avg (average)** εκτελείται σε περίπτωση που ο χρήστης δώσει ως επιλογή τον αριθμό 5. Τα βήματα του avg είναι πολύ απλά. Το άθροισμα (\$t2) αρχικοποιείται σε 0, και στη συνέχεια προσθέτουμε τους αριθμούς τον έναν μετά τον άλλον στο άθροισμα. Και επειδή θέλουμε τις απόλυτες τιμές των αριθμών, έγραψα ένα **branch** το οποίο ελέγχει αν ο κάθε αριθμός διαδοχικά είναι μεγαλύτερος ή όχι του μηδενός. Εφόσον βρεθεί μικρότερος του μηδενός, τότε χρησιμοποιώντας την *neg* καταχωρώ στον ίδιο καταχωρητή την θετική τιμή του και έπειτα τον προσθέτω στο άθροισμα. Θα μπορούσα να χρησιμοποιήσω την *abs* κατευθείαν (χωρίς καν το branch) και έτσι θα αδιαφορούσα για το αν θα ήταν θετικός ή όχι και θα είχα πάντα την θετική τιμή. Όμως προτίμησα την σύγκριση και έπειτα την *neg* που καταχωρεί τον αντίθετο αριθμό.

```

#-----
# avg - εύρεση μέσου όρου των απολύτων τιμών των αριθμών (επιλογή 5)
#   t0 - ο κάθε αριθμός διαδοχικά
#   t1 - μετράει τις επαναλήψεις
#   t2 - sum (το άθροισμα των αριθμών)
#   t3 - μέσος όρος
#-----
avg:  addi $fp,$fp,-80 # γυρίζω τον frame pointer ξανά στην αρχή
      li $t1,0        # μηδενίζω τον $t1
      li $t2,0        # μηδενίζω τον $t2

loop_sum:
  lw $t0,0($fp)      # καταχωρώ προσωρινά τον πρώτο αριθμό
  στον $t0
  bgtz $t0,continue_sum # αν t0<0
  neg $t0,$t0        # τότε θέσε στον $t0 την απόλυτη τιμή του $t0
continue_sum:        # αλλιώς συνέχισε κανονικά
  beq $t1,20,print_avg # στις 20 επαναλήψεις το πρόγραμμα
  γνωρίζει τον μέσο όρο
  addi $fp,$fp,4     # (4 bytes = 1 word)
  addi $t1,$t1,1     # αυξάνω κατά 1 τον μετρητή
  add $t2,$t2,$t0    # προσθέτω στο sum, τον i αριθμό

  j loop_sum

```

Η **print** εκτελείται σε περίπτωση που ο χρήστης δώσει ως επιλογή τον αριθμό 6. Αφήνω στην αρχή μια γραμμή για λόγους καλαισθησίας. Στην συνέχεια στο `loop_print` (όπου εμφανίζω τους αριθμούς χωρισμένους με -) δεν δίνεται πουθενά ο χαρακτήρας `\n` ή κάτι άλλο. Επομένως ο SPIM εμφανίζει κάθε αποτέλεσμα στην ίδια γραμμή. Εμφανίσω τον αριθμό που μόλις έχει φορτωθεί από τα `frame stack` και στην συνέχεια εμφανίζω και τον χαρακτήρα της παύλας. Όμως για να αποφύγω μετά τον τελευταίο αριθμό να υπάρχει – χρησιμοποιώ ένα ακόμη **branch** (πέρα αυτό του βρόχου) το οποίο την 20 φορά δεν θα εμφανίσει μετά τον αριθμό τον χαρακτήρα της παύλας.

```

#-----
# print - εκτύπωση αριθμών χωρισμένων μεταξύ τους με παύλα '-'
# (επιλογή 6)
# t0 - μετράει τις επαναλήψεις
#-----
print:
    addi $fp,$fp,-80 # γυρίζω τον frame pointer ξανά στην αρχή
    li $t0,0 # μηδενίζω τον $t0 που τον χρησιμοποιώ ως
μετρητή

    la $a0,newline # νέα γραμμή
    li $v0,4
    syscall

loop_print:
    lw $a0,($fp) # φορτώνω διαδοχικά τους αριθμούς
    li $v0,1 # syscall 1 εμφανίζει ακέραιο
    syscall

    addi $t0,$t0,1 # αυξάνω κατά ένα τον μετρητή
    addi $fp,$fp,4 # 4 bytes = 1 word (αύξηση κατά μία λέξη)

    beq $t0,20,ask_after # την 20ή φορά θα εμφανίσει τον αριθμό,
αλλά όχι την παύλα
    la $a0,dash # εμφανίζει την -
    li $v0,4 # syscall 4 εμφανίζει string
    syscall

    j loop_print

```

Η **ask_after** καλείται μετά το πέρας των εντολών όλων των επιλογών (πλην της ΕΞΟΔΟΥ). Σε κάθε περίπτωση ο χρήστης καλείται να επιλέξει επιστροφή στο μενού επιλογών, ΕΞΟΔΟ ή να ξανααδώσει 20 νέους αριθμούς. Καθαρά για user-friendly λόγους έγραψα (όπως και στην `done`) λίγες παραπάνω γραμμές κώδικα που σε περίπτωση λάθους επιλογής από τον χρήστη οδηγούν στην **attention2**. Η **attention2** εμφανίζει μήνυμα για το εύρος των επιλογών στην **ask_after** και στη συνέχεια ο SPIM περιμένει από τον χρήστη σωστή επιλογή.

Το σύνολο του πηγαίου κώδικα δίνεται χωριστά.

Παραδείγματα εκτέλεσης του προγράμματος στον SPIM

Στη συνέχεια θα παραθέσω δύο παραδείγματα εκτέλεσης με χαρακτηριστικές τιμές. Κάποιες δοκιμαστικές τιμές που επιτρέπουν να διαπιστώσουμε την σωστή λειτουργία του προγράμματος, είναι:

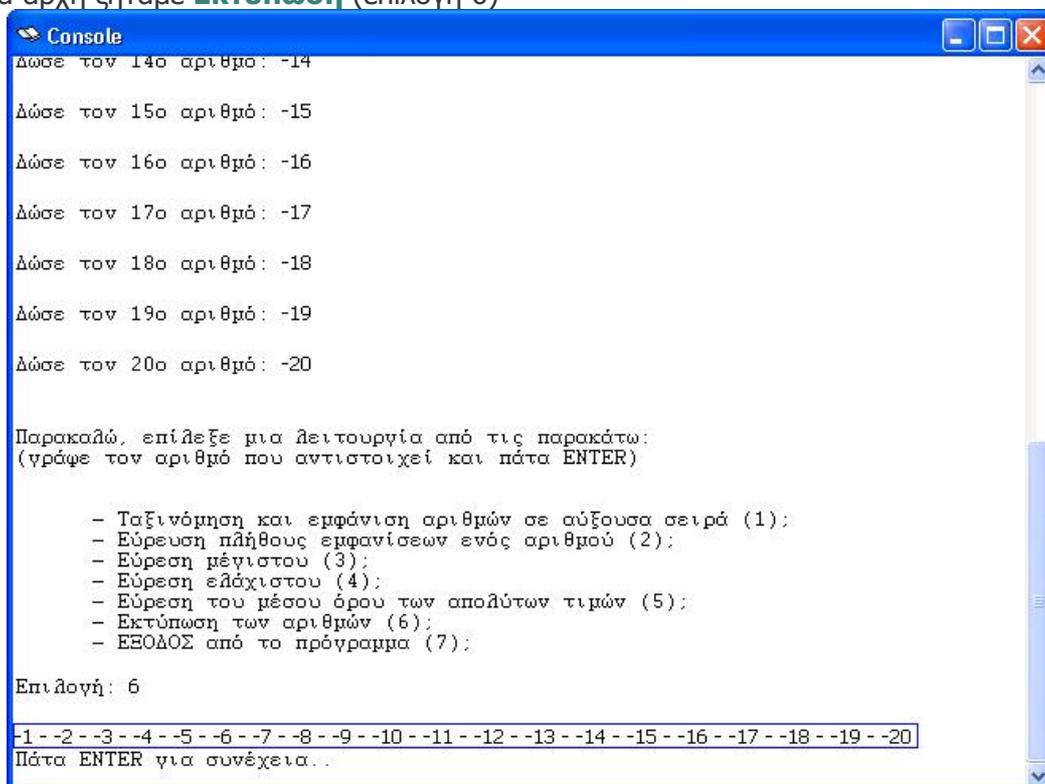
- α) όλες αρνητικές για εξακρίβωση του **max, min, του sort**, αλλά και του **avg**
- β) θετικοί και αρνητικοί αριθμοί.

α) ΠΡΩΤΗ ΔΟΚΙΜΑΣΤΙΚΗ ΕΚΤΕΛΕΣΗ

Στον PCSPIM δίνουμε τους εξής αριθμούς:

-1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20

Για αρχή ζητάμε **Εκτύπωση** (επιλογή 6)



```
Console
Δώσε τον 14ο αριθμό: -14
Δώσε τον 15ο αριθμό: -15
Δώσε τον 16ο αριθμό: -16
Δώσε τον 17ο αριθμό: -17
Δώσε τον 18ο αριθμό: -18
Δώσε τον 19ο αριθμό: -19
Δώσε τον 20ο αριθμό: -20

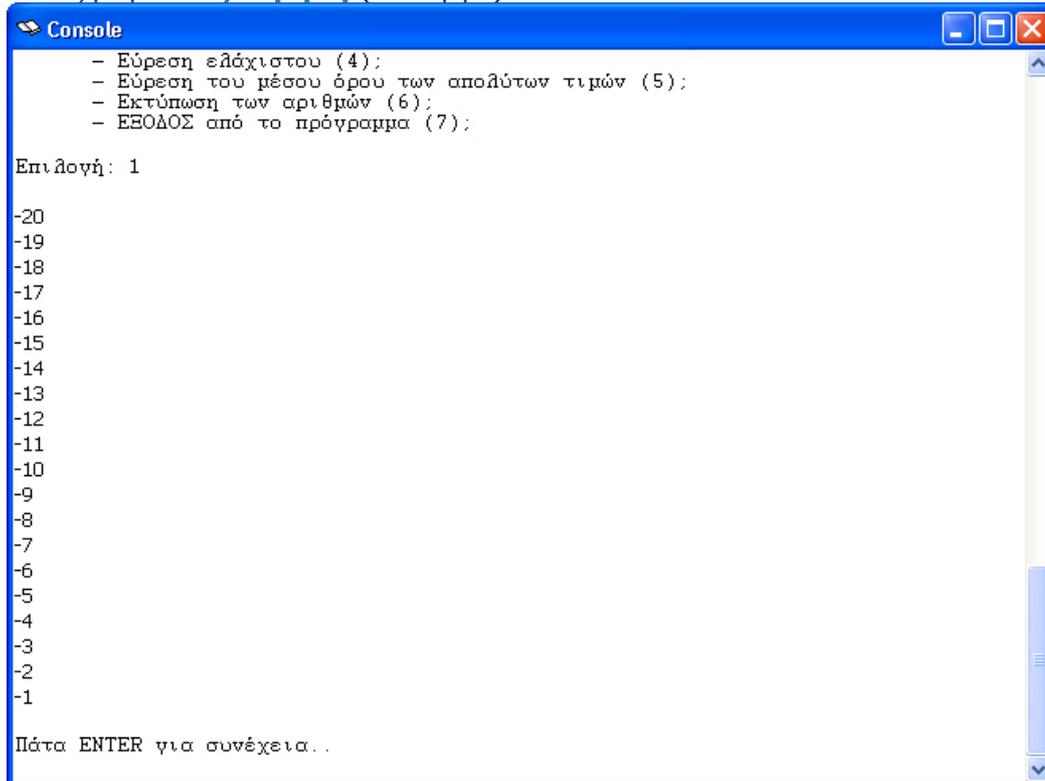
Παρακαλώ, επίλεξε μια λειτουργία από τις παρακάτω:
(γράψε τον αριθμό που αντιστοιχεί και πάτα ENTER)

- Ταξινόμηση και εμφάνιση αριθμών σε αύξουσα σειρά (1);
- Εύρεση πλήθους εμφανίσεων ενός αριθμού (2);
- Εύρεση μέγιστου (3);
- Εύρεση ελάχιστου (4);
- Εύρεση του μέσου όρου των απολύτων τιμών (5);
- Εκτύπωση των αριθμών (6);
- ΕΞΟΔΟΣ από το πρόγραμμα (7);

Επιλογή: 6
-1 - -2 - -3 - -4 - -5 - -6 - -7 - -8 - -9 - -10 - -11 - -12 - -13 - -14 - -15 - -16 - -17 - -18 - -19 - -20
Πάτα ENTER για συνέχεια..
```

Μεταξύ των αριθμών υπάρχει μία παύλα (η δεύτερη, όπου υπάρχει, είναι το αρνητικό πρόσημο). Παρατηρούμε ότι ο τελευταίος αριθμός δεν έχει παύλα στα δεξιά του.

Έπειτα ζητάμε **Ταξινόμηση** (επιλογή 1)



```
Console
- Εύρεση ελάχιστου (4);
- Εύρεση του μέσου όρου των απολύτων τιμών (5);
- Εκτύπωση των αριθμών (6);
- ΕΞΟΔΟΣ από το πρόγραμμα (7);

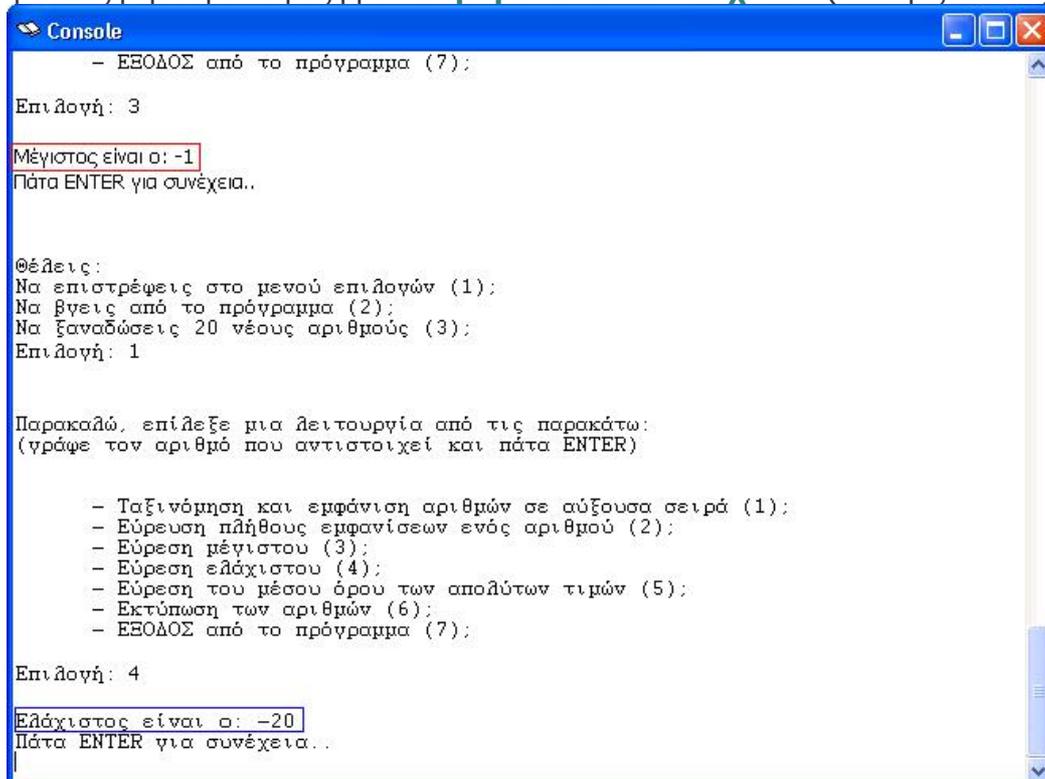
Επιλογή: 1

-20
-19
-18
-17
-16
-15
-14
-13
-12
-11
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1

Πάτα ENTER για συνέχεια..
```

διαπιστώνουμε πως η ταξινόμηση έχει γίνει σωστά. Κρατάμε ότι ο μέγιστος είναι ο -1 ενώ ο ελάχιστος είναι ο -20.

Τώρα θα ζητήσουμε να μας βρει τον **μέγιστο** και τον **ελάχιστο**. (επιλογές 3 και 4)



```
Console
- ΕΞΟΔΟΣ από το πρόγραμμα (7);

Επιλογή: 3
Μέγιστος είναι ο: -1
Πάτα ENTER για συνέχεια..

@έλεος:
Να επιστρέφεις στο μενού επιλογών (1);
Να βγεις από το πρόγραμμα (2);
Να ξαναδώσεις 20 νέους αριθμούς (3);
Επιλογή: 1

Παρακαλώ, επίλεξε μια λειτουργία από τις παρακάτω:
(γράψε τον αριθμό που αντιστοιχεί και πάτα ENTER)

- Ταξινόμηση και εμφάνιση αριθμών σε αύξουσα σειρά (1);
- Εύρεση πλήθους εμφανίσεων ενός αριθμού (2);
- Εύρεση μέγιστου (3);
- Εύρεση ελάχιστου (4);
- Εύρεση του μέσου όρου των απολύτων τιμών (5);
- Εκτύπωση των αριθμών (6);
- ΕΞΟΔΟΣ από το πρόγραμμα (7);

Επιλογή: 4
Ελάχιστος είναι ο: -20
Πάτα ENTER για συνέχεια..
```

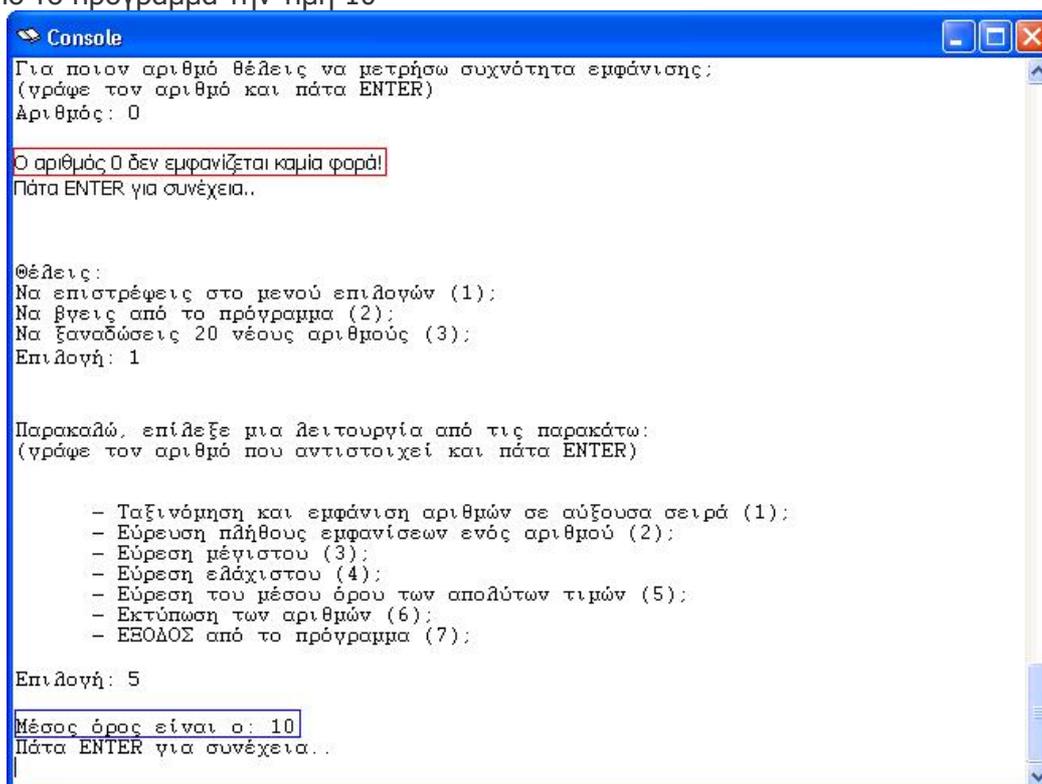
Σωστά αποτελέσματα!

Στη συνέχεια ζητάμε από το πρόγραμμα να μας δείξει **πόσες φορές εμφανίζεται** το 0 (εξηγώ παραπάνω γιατί διαλέγω το 0). Περιμένουμε (μετά τις user-friendly προσπάθειες μου) να μας εμφανίσει «Ο αριθμός 0 δεν εμφανίζεται καμία φορά!».

Έπειτα ζητάμε θα ζητήσουμε να μας δώσει **τον μέσο όρο των απολύτων τιμών**. Πρώτα κάνουμε κάποιους γρήγορους υπολογισμούς στο χαρτί:

$$1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20=210$$
$$210/20=10,5$$

Το ακέραιο μέρος του 10,5 είναι το **10**. Επομένως περιμένουμε ως ορθή απάντηση από το πρόγραμμα την τιμή 10



```
Console
Για ποιον αριθμό θέλεις να μετρήσω συχνότητα εμφάνισης;
(γράψε τον αριθμό και πάτα ENTER)
Αριθμός: 0

Ο αριθμός 0 δεν εμφανίζεται καμία φορά!
Πάτα ENTER για συνέχεια..

Θέλεις:
Να επιστρέφεις στο μενού επιλογών (1);
Να βγεις από το πρόγραμμα (2);
Να ξαναδώσεις 20 νέους αριθμούς (3);
Επιλογή: 1

Παρακαλώ, επίλεξε μια λειτουργία από τις παρακάτω:
(γράψε τον αριθμό που αντιστοιχεί και πάτα ENTER)

- Ταξινόμηση και εμφάνιση αριθμών σε αύξουσα σειρά (1);
- Εύρεση πλήθους εμφανίσεων ενός αριθμού (2);
- Εύρεση μέγιστου (3);
- Εύρεση ελάχιστου (4);
- Εύρεση του μέσου όρου των απολύτων τιμών (5);
- Εκτύπωση των αριθμών (6);
- ΕΞΟΔΟΣ από το πρόγραμμα (7);

Επιλογή: 5

Μέσος όρος είναι ο: 10
Πάτα ENTER για συνέχεια..
```

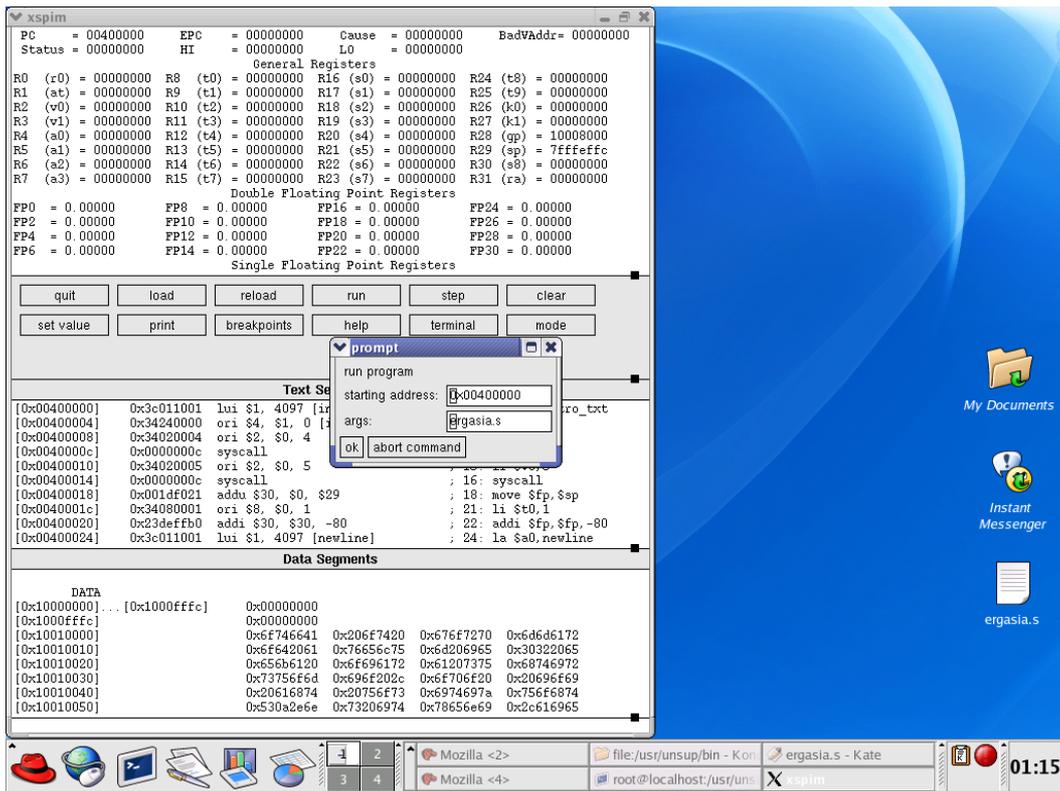
Πράγματι!

Σημείωση

Η σειρά που εκτελέστηκαν οι λειτουργίες ήταν τυχαία. Μπορούν να εκτελεσθούν και με διαφορετική σειρά και θα δώσουν σωστά αποτελέσματα. Επίσης μπορούμε να ζητήσουμε να ξαναδώσουμε 20 νέους αριθμούς και πάλι θα πάνε όλα κατ' ευχή! :)

β) ΔΕΥΤΕΡΗ ΔΟΚΙΜΑΣΤΙΚΗ ΕΚΤΕΛΕΣΗ

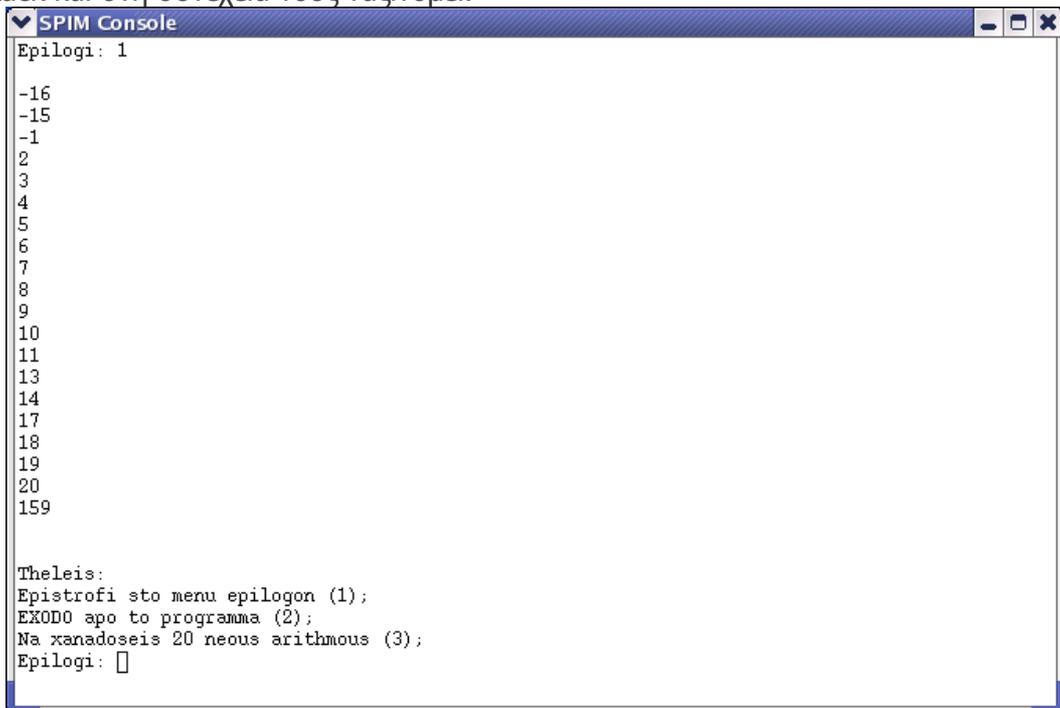
Θα δώσουμε τόσο αρνητικούς όσο και θετικούς αριθμούς. Εκτελούμε το πρόγραμμα στον XSPIM. Έχει προηγηθεί compile του XSPIM στο Redhat Linux. Επίσης προηγήθηκε μετατροπή τους προγράμματος από την ελληνική γλώσσα σε greeklish για λόγους **μη** υποστήριξης από την default γραμματοσειρά του XSPIM των ελληνικών χαρακτήρων (ISO 8859-7). Το παραθυρικό περιβάλλον του XSPIM (έχω ήδη φορτώσει το assembly αρχείο και ετοιμάζομαι για εκτέλεση) φαίνεται παρακάτω:



Δίνουμε του αριθμούς με την εξής σειρά:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 18, 19, 14, 17, -16, -1, -15, 159, 20.

Ζητάμε **ταξινόμηση & εμφάνιση** των αριθμών και εμφανίζει τους αριθμούς που μόλις δώσαμε ταξινομημένους σε αύξουσα σειρά. Το πρόγραμμα χρησιμοποιεί ένα array 20 λέξεων (words) όπου αρχικά αντιγράφει τους 20 αριθμούς από το frame stack και στη συνέχεια τους ταξινομεί.



Μεγέθη

Το **Text Segment** είναι 1132 λέξεις, αφού εκτείνεται σε δεκαεξαδική (δεκαδική) μορφή από 0x00400000 (4194304) έως 0x0040046c (4195436).
Πράγματι:

$$1132 = 4195436 - 4194304$$

Επίσης το **Data Segment** μετά την εκτέλεση του προγράμματος είναι 1572 λέξεις. Αφού εκτείνεται σε δεκαεξαδική (δεκαδική) μορφή από 0x10010000 (268500992) έως 0x10010624 (268502564). Πράγματι:

$$1572 = 268502564 - 268500992$$

Πριν την εκτέλεση του προγράμματος οι λέξεις στο Data Segment είναι ίδιες, όμως επειδή οι 20 αριθμοί (20 λέξεις) είναι όλοι τους 0 ο SPIM δεν τους διακρίνει μέσα στα υπόλοιπα μηδέν! Έτσι δημιουργείται η εντύπωση πως το πρόγραμμα πριν εκτελεστεί έχει 20 λιγότερες λέξεις!

Τέλος η στοίβα (**Stack**) πριν την εκτέλεση του προγράμματος οφείλει να μην περιέχει στοιχεία. Το πρόγραμμα χρησιμοποιεί την στοίβα, αφού δημιουργεί και frame stack. Συνεπώς μετά το τέλος του προγράμματος οι λέξεις που περιέχει είναι 1164. Εκτείνεται σε δεκαεξαδική (δεκαδική) μορφή από 0x7fffeb70 (2147478384) έως 0x7ffffeffc (2147479548). Πράγματι:

$$1164 = 2147479548 - 2147478384$$

Συμπεράσματα - σχόλια

Η συμβολική γλώσσα του MIPS είναι μια γλώσσα που εντυπωσιάζει με την λιτότητα της, αλλά και με τις δυνατότητες της. Όμως τα πάντα είναι θέμα συμβάσεων και συμφωνιών της επιστημονικής κοινότητας. Από αυτήν γεννήθηκε εξάλλου ο πρώτος υπολογιστής...

Μετά την ενασχόληση μου με την assembly για MIPS συνειδητοποίησα ακόμη περισσότερο πως οι άνθρωποι είναι αυτοί που «δημιουργούν» τον υπολογιστή. Μερικές φορές ο υπολογιστής μας εντυπωσιάζει με τις τεράστιες ταχύτητες με τις οποίες εκτελεί τις πράξεις. Δεν γνωρίζω αν και πότε η τεχνητή νοημοσύνη θα φτάσει σε επίπεδο ανθρώπινης κατανόησης και λογικής, πάντως νομίζω πως αν είναι open-source ο κώδικας (λίγο δύσκολο βέβαια..) και σβήσουμε ένα γράμμα ή κάνουμε ένα συντακτικό λάθος σε μία εντολή τίποτα δεν πρόκειται να δουλέψει σωστά.

Τελικά μάλλον δεν υπάρχει χαζός ή έξυπνος υπολογιστής, αλλά προσεκτικός ή απρόσεκτος προγραμματιστής. Αυτό γίνεται περισσότερο εμφανές στην assembly αφού είναι μια γλώσσα που καταδεικνύει την έννοια της βηματικής προσέγγισης και επίλυσης ενός προβλήματος, εξ ου και το όνομα της που σημαίνει «συναρμολόγηση», ενώ παράλληλα δεν «κρύβει» την λιτότητα του υλικού και της αρχιτεκτονικής του επεξεργαστή.