***ΕΡΓΑΣΤΉΡΙΟ ΑΡΧΙΤΕΚΤΟΝΙΚΉ Η/Υ***

**2Η ΕΡΓΑΣΊΑ ΕΡΓΑΣΤΗΡΊΟΥ**

**x32 MIPS ASSEMBLY**

Ορφέας- Άγγελος Νικολάου

ΑΜ: 2792

E-MAIL: int02792@uoi.gr

Άρτα, 2023 / 11 / 11

**Table of contents**

[Άσκηση 1 – MIPS 3](#__RefHeading___Toc777_1122253754)

[Περίληψη 3](#__RefHeading___Toc779_1122253754)

[Υλοποίηση 3](#__RefHeading___Toc781_1122253754)

[Κώδικας 3](#__RefHeading___Toc783_1122253754)

[Λειτουργία του κώδικα 5](#__RefHeading___Toc785_1122253754)

[2η Άσκηση – MIPS (μέρος 1ο) 6](#__RefHeading___Toc787_1122253754)

[Περίληψη 6](#__RefHeading___Toc789_1122253754)

[Υλοποίηση 6](#__RefHeading___Toc791_1122253754)

[2η Άσκηση – MIPS (μέρος 2ο) 7](#__RefHeading___Toc1190_1122253754)

[Περίληψη 7](#__RefHeading___Toc1192_1122253754)

[Υλοποίηση 7](#__RefHeading___Toc1195_1122253754)

[Κώδικας μηχανής δεκαδικό 7](#__RefHeading___Toc1197_1122253754)

[Περιγραφή πίνακα 8](#__RefHeading___Toc1202_1122253754)

[Κώδικάς μηχανής δυαδικό 9](#__RefHeading___Toc1204_1122253754)

[2η Άσκηση – MIPS (μέρος 3ο) 10](#__RefHeading___Toc787_1122253754_Copia_1)

[Περίληψη 10](#__RefHeading___Toc1206_1122253754)

[Υλοποίηση 10](#__RefHeading___Toc1211_1122253754)

[Ερώτηση 1 10](#__RefHeading___Toc689_1189963475)

[Ερώτηση 2 10](#__RefHeading___Toc689_1189963475_Copy_1)

[Ερώτηση 3 10](#__RefHeading___Toc689_1189963475_Copy_2)

[Ερώτηση 4 10](#__RefHeading___Toc689_1189963475_Copy_3)

[Ερώτηση 5 11](#__RefHeading___Toc689_1189963475_Copy_4)

[Ερώτηση 6 11](#__RefHeading___Toc689_1189963475_Copy_5)

[Ερώτηση 7 11](#__RefHeading___Toc689_1189963475_Copy_6)

# Άσκηση 1 – MIPS

## Περίληψη

Σε αυτή την άσκηση ζητιέται η υλοποίηση ενός κώδικα σε x32 MIPS asm όπου θα εκτυπώνει το ονοματεπώνυμο, ΑΜ, και εξάμηνο ενός μαθητή (το στοιχεία αυτά θα είναι στο .data και θα είναι ήδη αρχικοποιημένα).

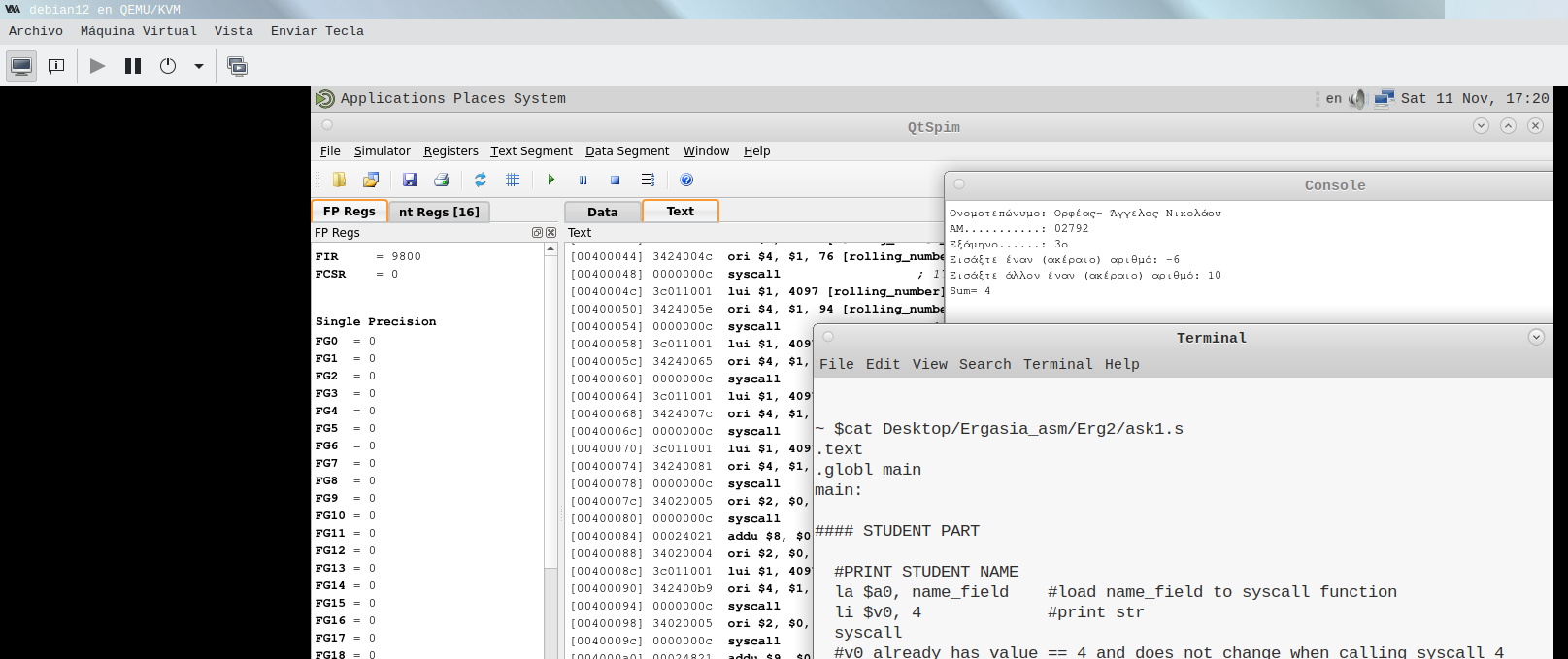
Μετά από αυτό, scanf() δύο ακέραιους αριθμούς με κατάλληλα μηνύματα και να κάνει printf() το άθροισμά τους.

## Υλοποίηση

### Κώδικας

Ο κώδικας βρίσκεται στα αρχεία της εργασίας στον κατάλληλο φάκελο (Ασκ1), και στο ακόλουθο pastebin και screenshot.

[](https://pastebin.com/dukwQxqm)https://pastebin.com/dukwQxqm

Το qtSPIM μόνο έχει .deb πακέτο, άρα έπρεπε να το βάλω σε VM debian,

Και δεν ήθελα να το κάνω compile.

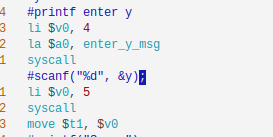
### Λειτουργία του κώδικα

Στο .data έθεσα τους αλφαριθμητικούς σε κάθε tag του. Χρησιμοποιώ asciiz για να έχει το ‘\0’ στο τέλος (δοκίμασα με ascii και βάλω εγώ το ‘\0’, δυστυχώς δεν δούλεψε).

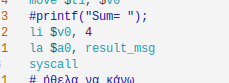
 Θέτω σε parameter για syscall το name\_field (από το .data), και λέω ότι θέλω να καλέσω την ‘4’ (ισοδυναμεί με puts() στην C);

Μέχρι Line30 δεν χρειάζεται να εξηγήσω κάτι αφού είναι η ίδια διαδικασία, απλά με διαφορετικά tags.

 Syscall 5 είναι scanf () int, και κάνει return στο $v0, άρα το θέτω στο $t0.



Ίδια λογική με τον δεύτερο ακέραιο.

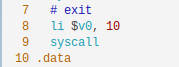


puts (“Sum= “) διότι δεν υπάρχουν variadic functions όπως την printf() στην asm ώστε να μπορώ να κάνω printf (“Sum= %d”, sum (k, y) ).



Θέτω $a0 (για παράμετρο για syscall) το άθροισμα μεταξύ τον δύο αριθμών που διαβάστηκαν.

Καλώ syscall 1 (printf (“%d”, some\_int) ).

 Syscall 10 είναι για exit το πρόγραμμα (αλλιώς συνεχίζει και διαβάζει διευθύνσεις μνήμης σαν να ήταν operations και δεν γίνεται implicitly όπως στην C ή C++).

# 2η Άσκηση – MIPS (μέρος 1ο)

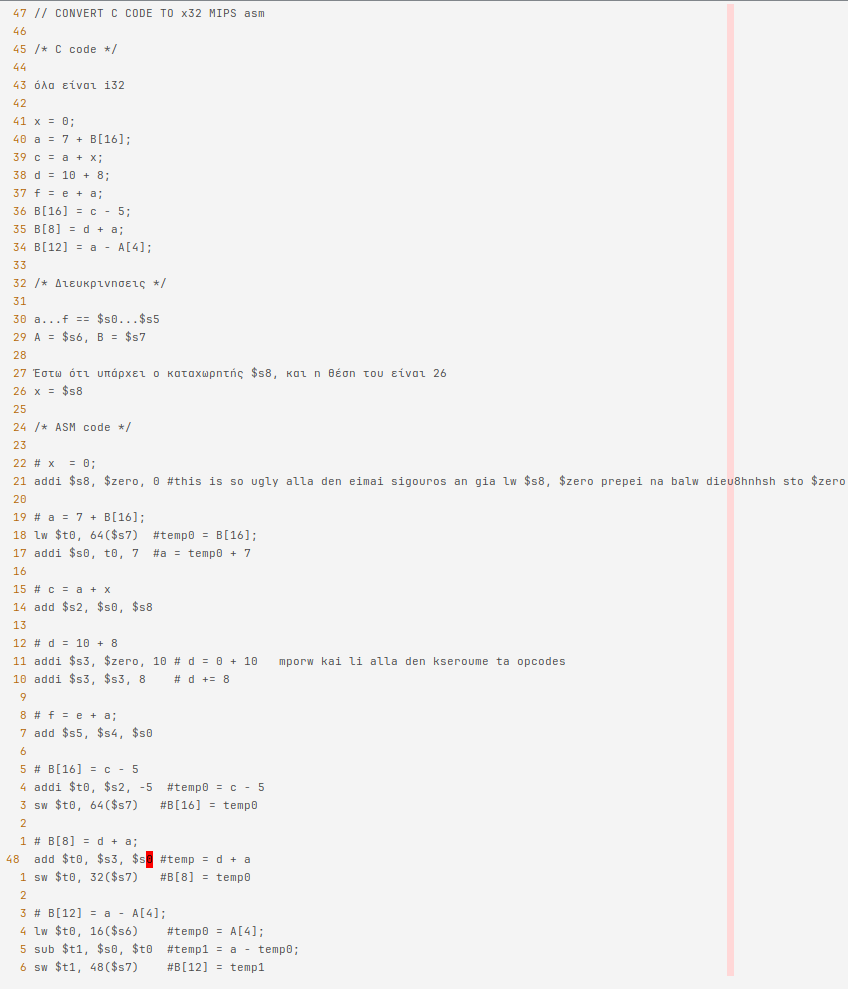
## Περίληψη

Ζητιέται η μετατροπή από κώδικα C σε κώδικα x32 MIPS assembly.

## Υλοποίηση

Το κάθε “block” εντολών στην δίπλα εικόνα αντιστοιχεί σε μία εντολή σε κώδικα της C.

Έστω ότι υπάρχει ο καταχωρητής $s8 και το register είναι 26.



# 2η Άσκηση – MIPS (μέρος 2ο)

## Περίληψη

Θέσαμε πριν στον καταχωρητή $s8 το register 26.

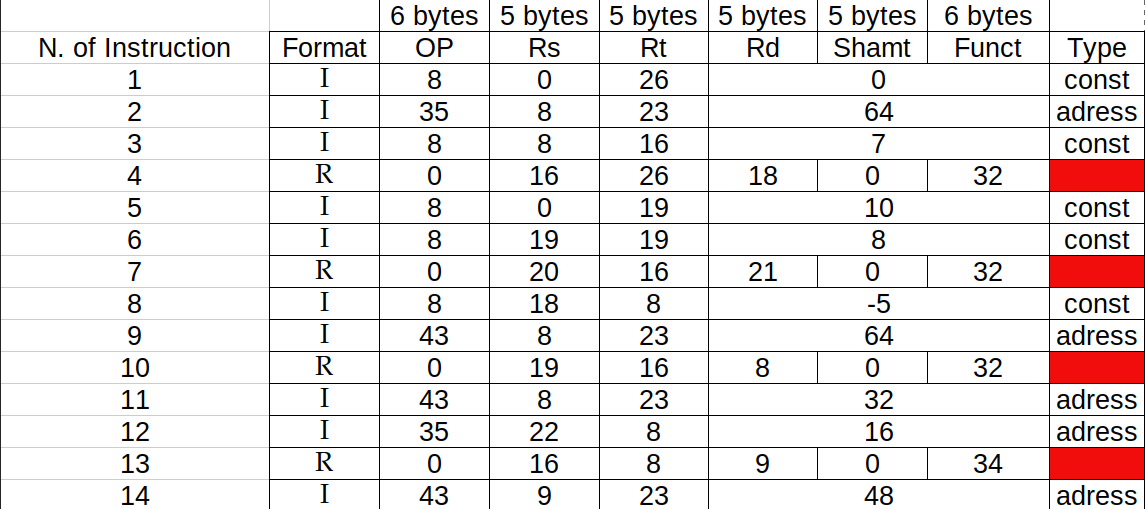
X32 MIPS asm → Machine code (base 10) → Περιγραφή πίνακα →

→ Machine code (base 2).

## Υλοποίηση

### Κώδικας μηχανής δεκαδικό

Στις σειρές με Format R το type είναι κόκκινο διότι δεν υπάρχει λόγος να συμπληρωθεί.



### Περιγραφή πίνακα

Για συντόμευση, εξηγώ το I και R από τώρα:

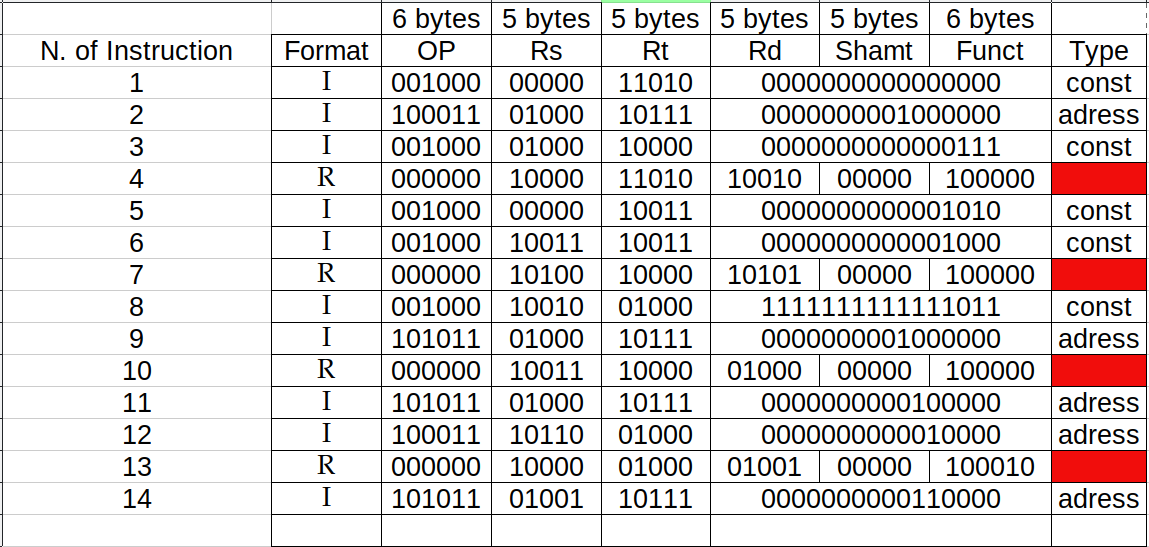
- add, sub πάντα έχουν format R.

- οι άλλες εντολές που ξέρουμε (εντός της εμβέλειας της εργασίας) είναι format I.

Οι αριθμοί στη ακόλουθη λίστα αντιστοιχούν στο N. of Instruction.

1. Rt είναι το destination για τις εντολές μορφής Ι, και αφού έχουμε addi, τότε είναι ο καταχωρητής $s8, ο οποίος δεν υπάρχει και για την εργασία θέσαμε τον αριθμό 26. Η σταθερά είναι 0 και το Rs 0 επειδή και τα δύο είναι $zero ή ‘0’.
2. Κάνω lw άρα το lhs είναι source και το rhs είναι destination, άρα ουσιαστικά είναι στην σειρά. 64 adress επειδή έχω M[$s7 + 64] (η διεύθυνση στις εντολές lw και sw είναι rhs μόνο).
3. Διεύθυνση προορισμού είναι $s0/16 άρα μπαίνει στο Rt. Ίδια λογική με Ν. 1.
4. Αφού είναι εντολή τύπου R τότε OP == 0 και το constant/adress διαιρείται σε 3 κομμάτια, Rd, Shamt, Funct. Τα δύο τελευταία είναι προκαθορισμένα από την add. add x, y, z. Rd → x, Rs → y, Rt → z.
5. καταχωρητής προορισμού $s3 άρα το βάζω στο Rt. Rs τον $zero άρα πρακτικά κάνω li $s3, 10.
6. $s3 += 8.
7. Καταχωρητής $s5 είναι 21 και προορισμού, άρα το βάζω στο Rd αφού η εντολή είναι μορφή R. Τα δύο source registers μπαίνουν Rs, Rt αριστερό δεξί αντίστοιχα.
8. Στο $t0 αφαιρώ το $s3 με το -5. Αφού είναι εντολή Ι, ο προορισμός πάλι είναι Rt (8).
9. Αυτή την φορά έχουμε sw (43) και ευτυχώς για την ψυχολογία μου είναι μια από τις λίγες εντολές που lhs → Rs και rhs → Rt, άρα τους καταχωρητές τους βάζω στην σειρά έτσι όπως είναι. Αφού είναι sw, ο 16bit αριθμός είναι adress.
10. Πάλη add, lhs → Rd, rhs → Rt, middle -→Rs.
11. sw άρα 23/$s7 Rt αφού είναι rhs και 8/$t0 Rs αφού είναι lhs.
12. lw άρα αντίθετα με το sw, Rt → 8/$t0/destination/lhs και Rs → 22/$s6/source/rhs. Adress αυτή την φορά είναι 16 αφού θέλω την 4η θέση.
13. Επιτέλους κάτι διαφορετικό, έχουμε sub. Δουλεύει (στα OP codes) ακριβώς σαν το add (δηλαδή Rd→ destination κτλ) αλλά το Funct είναι 34 αντί για 32.
14. sw άρα η διεύθυνση προορισμού είναι αυτή που είναι δεξιά, δηλαδή ο Β[12] ( 48($s7) ).

### Κώδικάς μηχανής δυαδικό



Στην 8. το const είναι τόσο μεγάλο επειδή είναι το -5. Η αρχιτεκτονική x32 MIPS τα ints έχει ως αρνητικά με συμπλήρωμα του δύο. (Κάνουμε bit flip όλα τα bits και αθροίζουμε μία μονάδα ή από δεξιά προς αριστερά κοιτάμε για έναν άσο, και όταν/άμα τον βρούμε κάνουμε bit flip όλα τα ψηφία στην αριστερή μεριά του). Δεν υπάρχει κάτι άλλο παράξενο, απλά μετατροπή από base10 σε base2.

Παρατηρείται ότι η κάθε γραμμή περιέχει 32bits.

# 2η Άσκηση – MIPS (μέρος 3ο)

## Περίληψη

Απαντήσεις ερωτήσεων.

## Υλοποίηση

### Ερώτηση 1

Πέρα από τα διαφορετικά OP codes (το οποίο είναι λογικό, αφού είναι διαφορετικές εντολές format I), το lw διαβάζει τιμή από την διεύθυνσή μνήμης και την θέτει στον καταχωρητή (το δεξί βάζει τιμή στο αριστερό, και το δεξί μπορούμε να βάλουμε adress). Το sw διαβάζει από έναν καταχωρητή (πάλη αριστερά) και τον θέτει σε μια διεύθυνση μνήμης (δεξιά) (το δεξί παίρνει τιμή από το αριστερό, και το δεξί μπορούμε να βάλουμε adress).

### Ερώτηση 2

Με τον καταχωρητή $zero ή με οποιονδήποτε άλλον που να έχει την ίδια τιμή .

### Ερώτηση 3

Με την εντολή addi, και είναι ως εξής:

addi $some\_register, $some\_register, 10.

### Ερώτηση 4

R format: περιέχει τα fields rd, shamt, funct. 5bit, 5bit, 6bit. Για κάθε πράξη το destination register είναι το rd, το shamt ορίζει πόσο left/right bit shift, και funct ως replacement για OP (αφού αυτό πρέπει να είναι 0 για να οριστεί το R format). Ο καταχωρητής πιο αριστερά είναι ο rd (προορισμού).

I format: τα 3 fields που ανέφερα πριν είναι 1 στο I format, άρα είναι και 16 bit. Αναλόγως την πράξη είναι constant ή adress αυτή η τιμή, και το destination register ΠΆΝΤΑ είναι Rt, όμως ανάλογα με την εντολή αυτός μπορεί να είναι ο rhs ή lhs καταχωρητής.

### Ερώτηση 5

Εντολές R format:

add, sub, sll, slr, or, and, nor

Εντολές I format:

lw, sw, addi, la, li, ori, andi, not, move

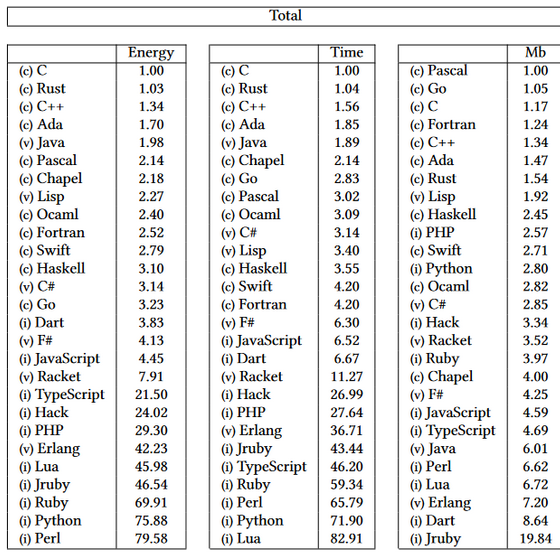
### Ερώτηση 6

Ναι, πχ. το $v0 για να ορίσουμε ποια syscall θέλουμε (κάποιες φορές είναι και τιμή return της syscall). $a0...$a3 για ορίσματα syscall. $t0...$t9 για temp (είναι συνήθεια να χρησιμοποιούνται αν και μόνο αν χρειάζονται για μια πράξη).

Υπάρχουν και $f0, $f12 όπου είναι για parameter και return αντίστοιχα για syscalls με float.

### Ερώτηση 7

Ένα από τα πλεονεκτήματα είναι ότι το πρόγραμμα μπορεί να τρέξει στην ίδια αρχιτεκτονική (πχ. X86\_64/amd64) με λίγες η μπορεί και καμία αλλαγή.

Μαζί με αυτό, το πρόγραμμα είναι πολύ πιο γρήγορο, διότι δεν μέσα από έναν interpreter ή ένα runtime virtual machine.

C/C++/Rust → Compiled

Java →Runtime VM

Python/Lua/JS →interpreted

[ΠΗΓΉ](https://thenewstack.io/which-programming-languages-use-the-least-electricity/)