***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)

# 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).

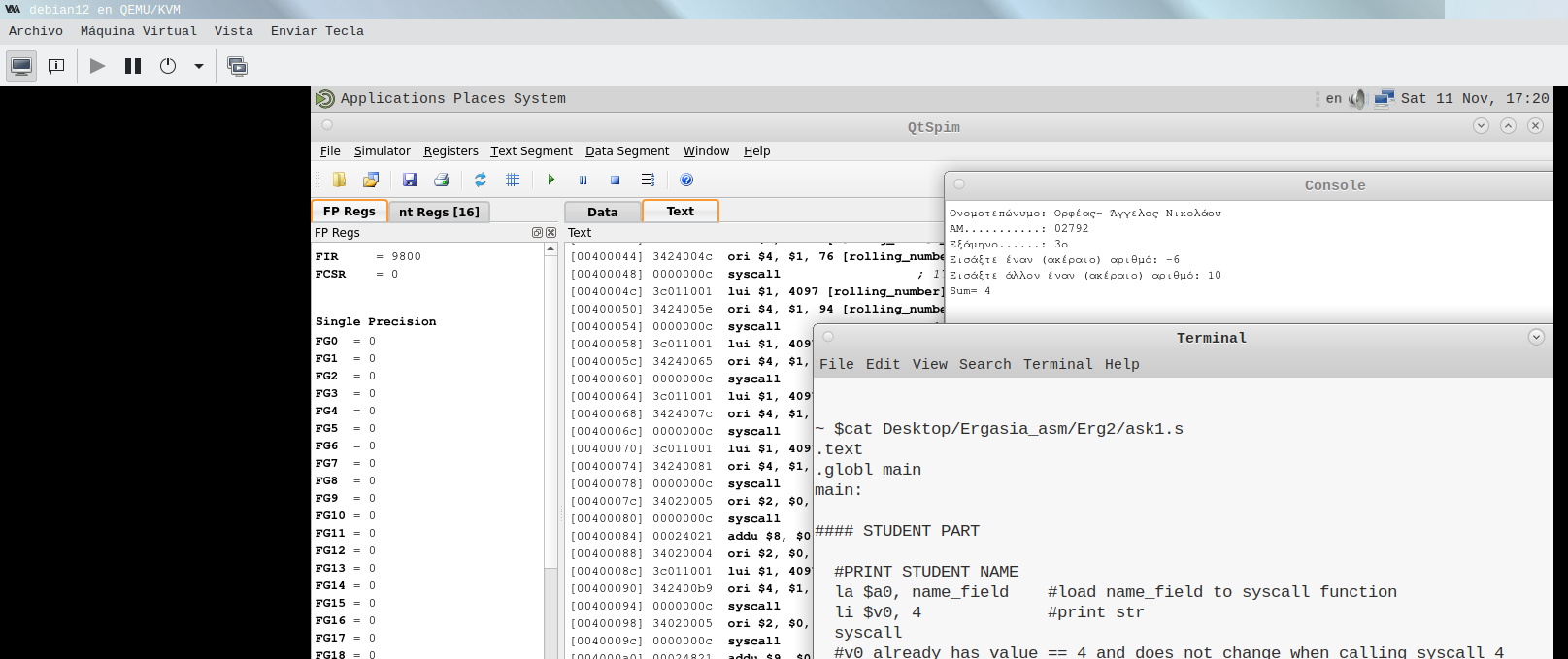
Μετά από αυτό, scanf() δύο ακέραιους αριθμούς με κατάλληλα μηνύματα και να κάνει printf() το άθροισμά τους.

## 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.

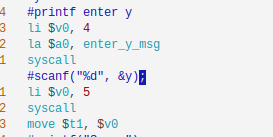
### Λειτουργία του κώδικα

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).

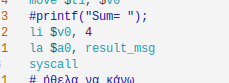
 Θέτω σε parameter για syscall το name\_field (από το .data), και λέω ότι θέλω να καλέσω την ‘4’ (ισοδυναμεί με puts() στην C);

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.

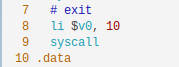


puts (“Sum= “) διότι δεν υπάρχουν variadic functions όπως την printf() στην asm ώστε να μπορώ να κάνω printf (“Sum= %d”, sum (k, y) ).



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)

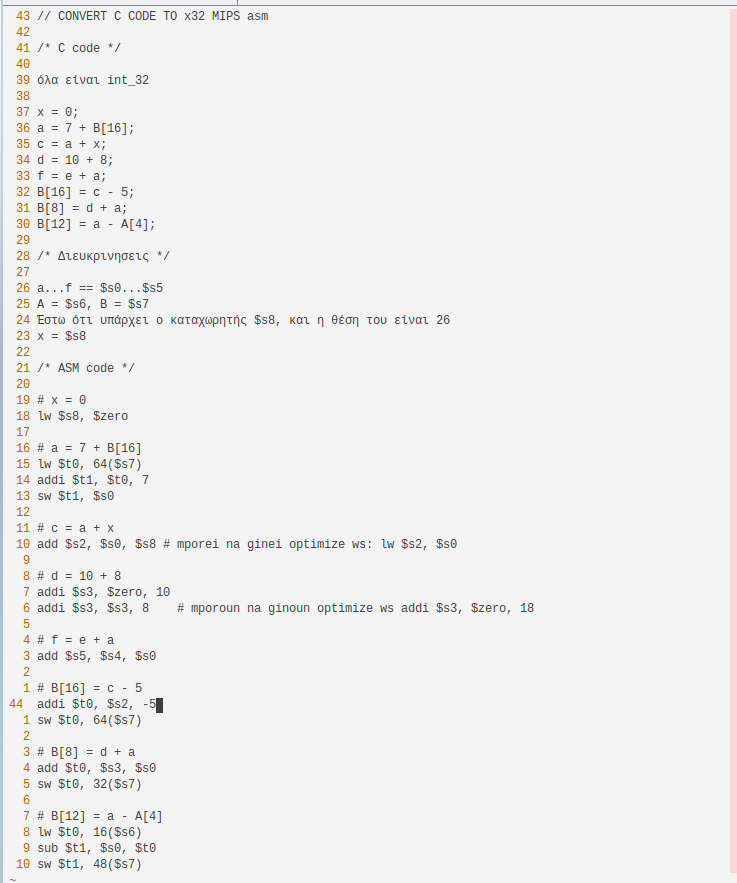
## Περίληψη

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.

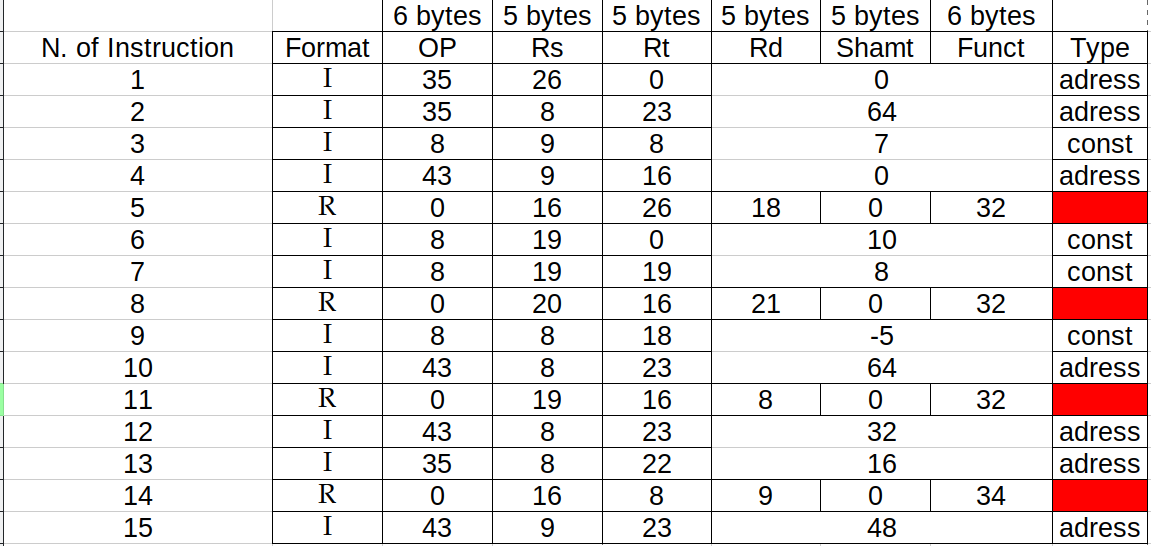
X32 MIPS asm → Machine code (base 10) → Περιγραφή πίνακα →

→ 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:

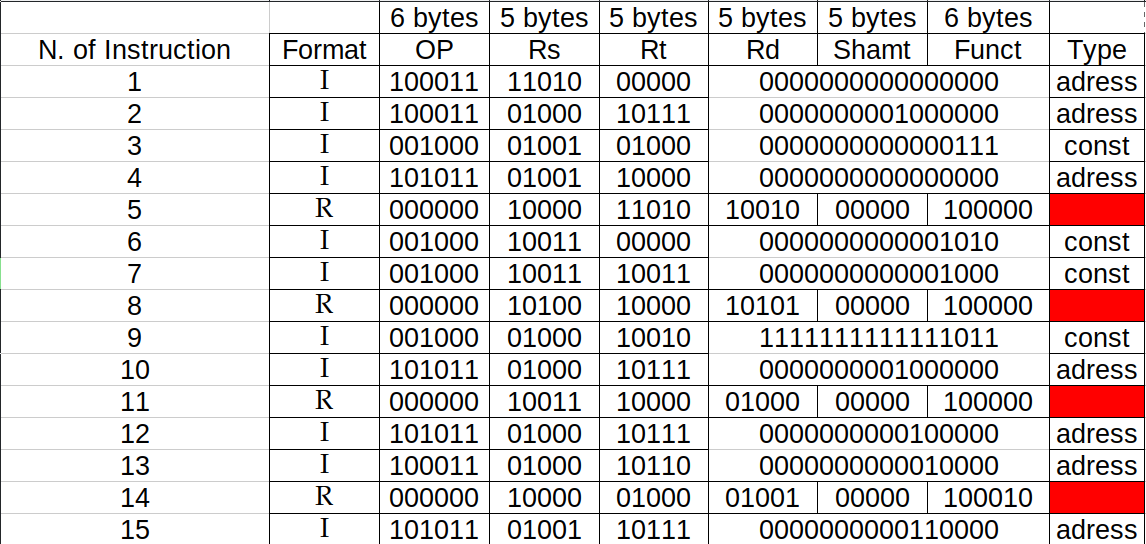
- add, sub πάντα έχουν format R.

- οι άλλες εντολές που ξέρουμε είναι format I.

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

1. Η εντολή lw έχει OP code 35. Rs είναι το αριστερό στοιχείο στις εντολές με format I και Rt το δεξί στοιχείο. Και αφού έχουμε lhs($s8) rhs($zero) τότε τα register είναι 26, 0. Η εντολή lw έχει το type ως adress, αλλά δεν έχουμε πίνακα άρα το 16bit παραμένει 0.
2. $t0 == 8, $s7 == 23. Ίδια λογική με Ν. 1. Offset (η τιμή που είναι 16bit με format I) είναι 64, αφού δουλεύουμε με int. 64/4 == 16η θέση. (Β[16]).
3. Addi OP == 8. Πάλη το destination είναι το Rs αφού είναι Format I. $t1(9) = $t0(8) + 7. Αφού έχουμε addi τότε το type είναι constant, και αυτή την φορά το θέλουμε να είναι 7, αφού αθροίζουμε $t0 με 7.
4. sw έχει ως OP code το 43, και είναι το αντίστροφο με την lw. Αφού αυτή θέτει στην lhs τιμή την rhs τιμή, η sw θέτει στην rhs την lhs.
5. Οι εντολές με format R (σε αυτή την περίπτωση η add) είναι σχετικά παράξενες. Αρχικά, (τουλάχιστον με αυτές που ξέρουμε) το OP code είναι 0. Δεύτερων, το “offset”, “const”, “adress” (βασικά το field με 16bits), χωρίστηκε στα 3. rd 5 bit, shamnt 5 bit, funct 6 bit. Η funct δουλεύει σαν το OP code στις εντολές με format I. πχ. η add είναι πάντα 43 funct. Το shamnt ορίζει το bit shift amount, δεν μας αφορά για τώρα και είναι 0. Τώρα, το destination register πάντα θα είναι το rd. add reg1, reg2, reg3. Το reg1 είναι rd, και reg2 reg3 rs rt αντίστοιχα όπως πριν.
6. Πρακτικά θέτει την τιμή 10 στον καταχωρητή $s3 (reg 19) αφού $zero (τιμή και καταχωρητή 0) + 10 == 10.
7. Το ίδιο με C some\_var += 8.
8. Add στο f (register 21, $s5) το sum a (register 16, $s0) e (register 20, $s4). rd είναι ο προορισμός όταν έχουμε εντολές format R.
9. Διότι δεν ξέρουμε τα Op codes του subi (αν υπάρχει) στο const του addi έβαλα -5, και κάνει την ίδια δουλειά.
10. Save word $t0 (από την προηγούμενη εντολή) προς Β[16] (shift amount 64/sizeof(int\_32)).
11. Άθροισμα των a ($s0) d ($s3) στο $t0 ( reg 8 ).
12. Save word $t0 στο B[8] (shift amount 32/4). (B είναι $s7).
13. Load word από A[4] στο $t0 (δεν το χρειαζόμαστε άλλο έτσι και αλλιώς).
14. a – A[4] θέτω σε ένα temp. ($t1)
15. Save word $t1 (reg 9) στο shitfted with value 48 register $s7 (aka B[12]).

### Κώδικάς μηχανής δυαδικό



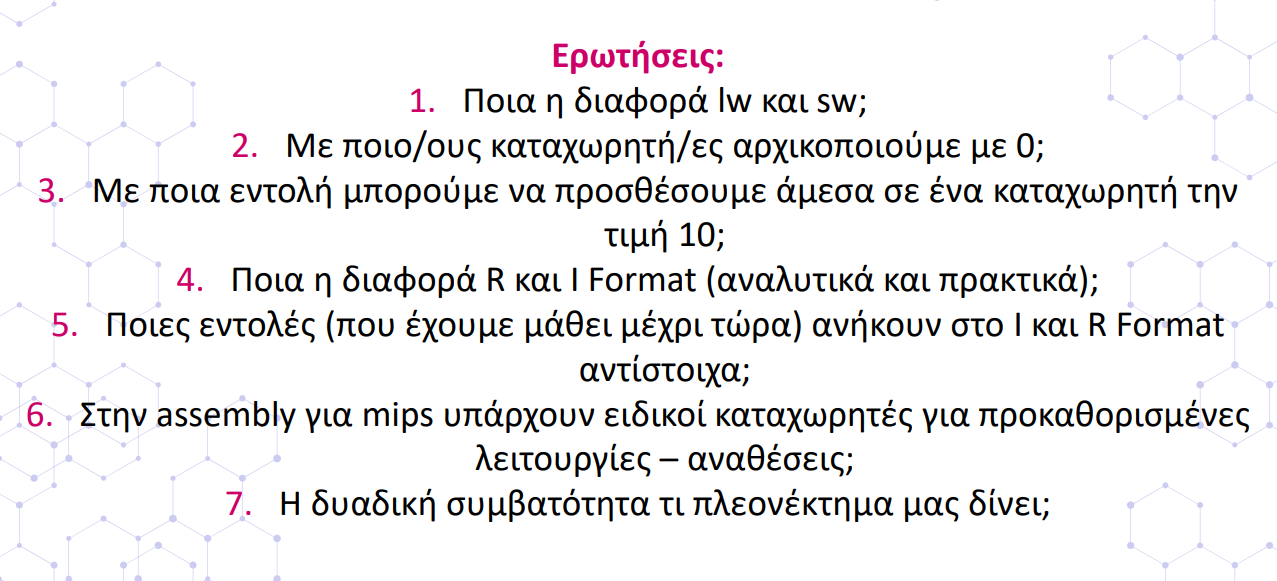
Στην 9. το const είναι τόσο μεγάλο επειδή είναι το -5. Η αρχιτεκτονική x32 MIPS τα ints έχει ως αρνητικά με συμπλήρωμα του δύο. (Κάνουμε bit flip όλα τα bits και αθροίζουμε μία μονάδα). Δεν υπάρχει κάτι άλλο παράξενο, απλά μετατροπή από base10 σε base2.

# 2η Άσκηση – MIPS (μέρος 3ο)

## Περίληψη

Απαντήσεις ερωτήσεων.

## Υλοποίηση



1. Πέρα από τα διαφορετικά OP codes (το οποίο είναι λογικό, αφού είναι διαφορετικές εντολές format I), το lw θέτει την τιμή του rhs register στο lhs register. Το sw θέτει την τιμή του lhs register στο rhs register.
2. Ο ποιος εύκολος τρόπος με καταχωρητές είναι lw/sw και θέτουμε τον καταχωρητή $zero στον καταχωρητή που θέλουμε να μηδενίσουμε. Το ίδιο μπορούμε να κάνουμε με οποιονδήποτε καταχωρητή έχει την τιμή 0, αλλά γενικά με τον $zero είναι ο προτεινόμενος τρόπος.
3. Με την εντολή addi, και είναι ως εξής:

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

1. 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). lhs rhs ποτέ δεν είναι destination registers.

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 εξαρτάται από την πράξη αν είναι lhs ή rhs.

1. Εντολές R format:

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

Εντολές I format:

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

1. 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). Ή κάποιο άλλο σημαντικό το $sp (stack pointer) αν και δεν τον έχουμε κάνει ακόμα.
2. Το πλεονέκτημα ότι το πρόγραμμα μπορεί να τρέξει σε διάφορα λειτουργικά συστήματα είτε με μία μεταγλώττιση ή εγκαθιστώντας ένα runtime VM όπως έχει η Java (μπορεί και διαφορετικές αρχιτεκτονικές, αν υπάρχει compiler για αυτή την γλώσσα και ο προγραμματιστής έχει γράψει portable κώδικα).