ΕΡΓΑΣΤΉΡΙΟ ΑΡΧΙΤΕΚΤΟΝΙΚΉ Η/Υ

2Η ΕΡΓΑΣΊΑ ΕΡΓΑΣΤΗΡΊΟΥ

x32 MIPS ASSEMBLY

Ορφέας- Άγγελος Νικολάου

AM: 2792

E-MAIL: int02792@uoi.gr

Άρτα, 2023 / 11 / 11

Table of contents

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

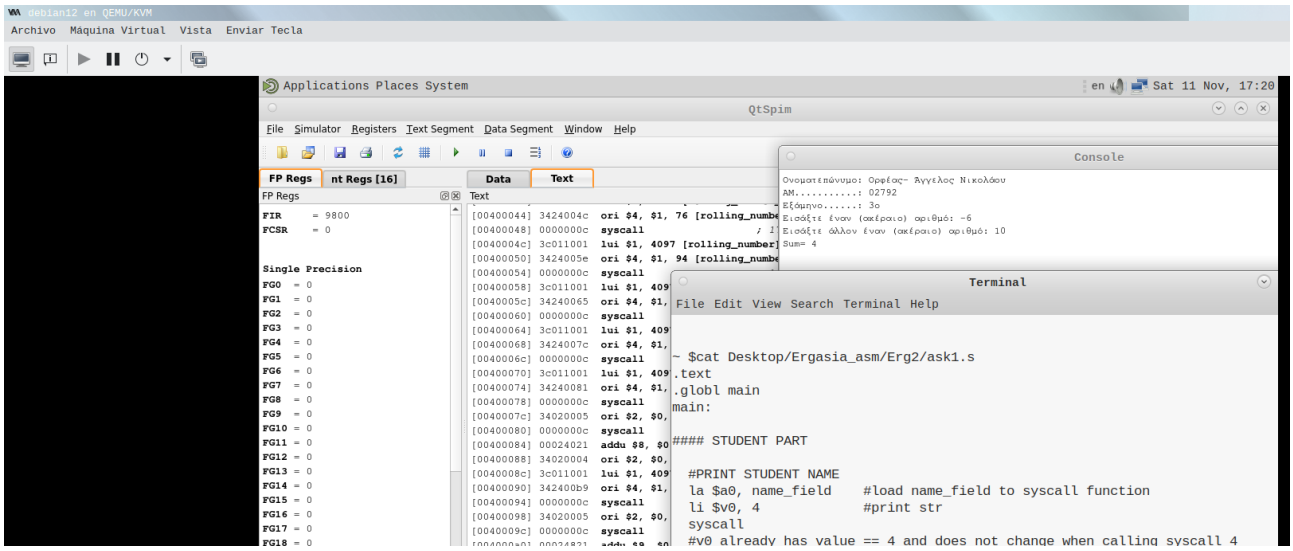
Κώδικας

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

https://pastebin.com/dukwQxqm

```
1   .text
 1 .globl main
 2 main:
 3
 4 #### STUDENT PART
 5
 6   #PRINT STUDENT NAME
 7   la $a0, name_field    #load name_field to syscall function
 8   li $v0, 4             #print str
 9   syscall
10   #v0 already has value == 4 and does not change when calling syscall 4
11   #so no reason to load it again as 4
12   la $a0, name
13   syscall
14   #PRINT STUDENT ROLLING NUMBER
15   la $a0, rolling_number_field
16   syscall
17   la $a0, rolling_number
18   syscall
19   #PRINT STUDENT SEMESTER
20   la $a0, semester_field
21   syscall
22   la $a0, semester
23   syscall
24
25 ####ADD NUMBERS PART
26
27   # x
28   #printf enter x
29   la $a0, enter_x_msg
30   syscall
31   #scanf("%d", &x);
32   li $v0, 5
33   syscall
34   move $t0, $v0 #return value from syscall 5 is $v0
35   #y
36   #printf enter y
37   li $v0, 4
38   la $a0, enter_y_msg
39   syscall
```

```
 22   #printf enter y
 21   li $v0, 4
 20   la $a0, enter_y_msg
 19   syscall
 18   #scanf("%d", &y);
 17   li $v0, 5
 16   syscall
 15   move $t1, $v0
 14   #printf("Sum= ");
 13   li $v0, 4
 12   la $a0, result_msg
 11   syscall
 10   # ήθελα να κάνω
  9   # add $s0, $t0, $t1
  8   # και το s0 --> a0 με lw/sw
  7   # αλλά δεν με άφηνε το interpreter/compiler/whatever
  6   #printf("%d", sum);
  5   li $v0, 1
  4   add $a0, $t0, $t1
  3   syscall
  2   # exit
  1   li $v0, 10
 59   syscall
  1 .data
  2   # για καλύτερο modularity εβαλα 1 για το σταθερο και 1 για το καθε μαθητη
  3   name_field:
  4     .asciiz "Ονοματεπώνυμο: "
  5   name:
  6     .asciiz "Ορφέας- Άγγελος Νικολάου\n"
  7   rolling_number_field:
  8     .asciiz "AM..........: "
  9   rolling_number:
 10     .asciiz "02792\n"
 11   semester_field:
 12     .asciiz "Εξάμηνο.....: "
 13   semester:
 14     .asciiz "3ο\n"
 15   # printf sum part
 16   enter_x_msg:
 17     .asciiz "Εισάξτε έναν (ακέραιο) αριθμό: "
 18   enter_y_msg:
 19     .asciiz "Εισάξτε άλλον έναν (ακέραιο) αριθμό: "
 20   result_msg:
 21     .asciiz "Sum= "
~
```



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).

```
7   la $a0, name_field
8   li $v0, 4
9   syscall
```

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.

```
31  #scanf("%d", &x);
32  li $v0, 5
33  syscall
34  move $t0, $v0 #return value from syscall 5 is $v0
```

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

```
4   #printf enter y
3   li $v0, 4
2   la $a0, enter_y_msg
1   syscall
    #scanf("%d", &y);
1   li $v0, 5
2   syscall
3   move $t1, $v0
```

Ίδια λογική με τον δεύτερο ακέραιο.

```
3   #printf("Sum= ");
2   li $v0, 4
1   la $a0, result_msg
    syscall
1   # ήθελα να κάνω
```

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

```
6   li $v0, 1
7   add $a0, $t0, $t1
8   syscall
```

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

Καλώ syscall 1 (printf (“%d”, some_int)).

```
7   # exit
8   li $v0, 10
9   syscall
10 .data
```

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.

```
// CONVERT C CODE TO x32 MIPS asm

/* C code */

όλα είναι int_32

x = 0;
a = 7 + B[16];
c = a + x;
d = 10 + 8;
f = e + a;
B[16] = c - 5;
B[8] = d + a;
B[12] = a - A[4];

/* Διευκρινησεις */

a...f == $s0...$s5
A = $s6, B = $s7
Έστω ότι υπάρχει ο καταχωρητής $s8, και η θέση του είναι 26
x = $s8

/* ASM code */

# x = 0
lw $s8, $zero

# a = 7 + B[16]
lw $t0, 64($s7)
addi $t1, $t0, 7
sw $t1, $s0

# c = a + x
add $s2, $s0, $s8 # mporei na ginei optimize ws: lw $s2, $s0

# d = 10 + 8
addi $s3, $zero, 10
addi $s3, $s3, 8    # mporoun na ginoun optimize ws addi $s3, $zero, 18

# f = e + a
add $s5, $s4, $s0

# B[16] = c - 5
addi $t0, $s2, -5
sw $t0, 64($s7)

# B[8] = d + a
add $t0, $s3, $s0
sw $t0, 32($s7)

# B[12] = a - A[4]
lw $t0, 16($s6)
sub $t1, $s0, $t0
sw $t1, 48($s7)
```

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.

		6 bytes	5 bytes	5 bytes	5 bytes	5 bytes	6 bytes	
N. of Instruction	Format	OP	Rs	Rt	Rd	Shamt	Funct	Type
1	I	35	26	0		0		adress
2	I	35	8	23		64		adress
3	I	8	9	8		7		const
4	I	43	9	16		0		adress
5	R	0	16	26	18	0	32	
6	I	8	19	0		10		const
7	I	8	19	19		8		const
8	R	0	20	16	21	0	32	
9	I	8	8	18		-5		const
10	I	43	8	23		64		adress
11	R	0	19	16	8	0	32	
12	I	43	8	23		32		adress
13	I	35	8	22		16		adress
14	R	0	16	8	9	0	34	
15	I	43	9	23		48		adress

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 N. 1. Offset (η τιμή που είναι 16bit με format I) είναι 64, αφού δουλεύουμε με int. 64/4 == 16η θέση. (B[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 $t0 (από την προηγούμενη εντολή) προς B[16] (shift amount 64/sizeof(int_32)).
11. Άθροισμα των a ($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

		6 bytes	5 bytes	5 bytes	5 bytes	5 bytes	6 bytes	
N. of Instruction	Format	OP	Rs	Rt	Rd	Shamt	Funct	Type
1	I	100011	11010	00000	0000000000000000			adress
2	I	100011	01000	10111	0000000001000000			adress
3	I	001000	01001	01000	0000000000000111			const
4	I	101011	01001	10000	0000000000000000			adress
5	R	000000	10000	11010	10010	00000	100000	
6	I	001000	10011	00000	0000000000001010			const
7	I	001000	10011	10011	0000000000001000			const
8	R	000000	10100	10000	10101	00000	100000	
9	I	001000	01000	10010	1111111111111011			const
10	I	101011	01000	10111	0000000001000000			adress
11	R	000000	10011	10000	01000	00000	100000	
12	I	101011	01000	10111	0000000000100000			adress
13	I	100011	01000	10110	0000000000010000			adress
14	R	000000	10000	01000	01001	00000	100010	
15	I	101011	01001	10111	0000000000110000			adress

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

Ερωτήσεις:

1. Ποια η διαφορά lw και sw;
2. Με ποιο/ους καταχωρητή/ες αρχικοποιούμε με 0;
3. Με ποια εντολή μπορούμε να προσθέσουμε άμεσα σε ένα καταχωρητή την τιμή 10;
4. Ποια η διαφορά R και I Format (αναλυτικά και πρακτικά);
5. Ποιες εντολές (που έχουμε μάθει μέχρι τώρα) ανήκουν στο I και R Format αντίστοιχα;
6. Στην assembly για mips υπάρχουν ειδικοί καταχωρητές για προκαθορισμένες λειτουργίες – αναθέσεις;
7. Η δυαδική συμβατότητα τι πλεονέκτημα μας δίνει;

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.

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). 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.

5. Εντολές R format:

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

   Εντολές I format:

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

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). Ή κάποιο άλλο σημαντικό το $sp (stack pointer) αν και δεν τον έχουμε κάνει ακόμα.

7. 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).