

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Αναφορά 2^{ου} Εργαστηρίου RISC-V Μάθημα: Εργαστήριο Μικροϋπολογιστών

Ομάδα: Α3

Ονοματεπώνυμο: Μάρκος Γκέργκες

Αριθμός Μητρώου: 03117870

Ακαδημαϊκό Έτος 2020-21

Ερώτημα 1

Τα στοιχεία των πινάκων είναι αποθηκευμένα σε συνεχόμενη μνήμη ανά 4 bytes, αφού έχουμε ορίσει σαν "τύπο" στοιχείων τα words (4 bytes). Αυξάνοντας έναν κατάλληλο δείκτη (διεύθυνση) κατά 4, μπορούμε να λαμβάνουμε το επόμενο στοιχείο του πίνακα.

Η εντολή `negate(neg RegD, Reg2)` χρησιμοποιείται στην εύρεση της απόλυτης τιμής και έχει ως αποτέλεσμα `RegD = -Reg2`. Έχει την ίδια μεταγλώττιση με την `SUB RegD,x0,Reg2`.

Μέσω του debugger-whisper, μπορούμε εφόσον θέσουμε (στον κώδικα που ακολουθεί) τον `t5` στη διεύθυνση του πίνακα `C`, από τον πίνακα των καταχωρητών να διαβάσουμε την διεύθυνση που βρίσκεται στη μνήμη. Στην εικόνα που παρατίθεται βρέθηκε πως ο `t5` αρχικά είχε τιμή `0x2198`. Επιλέχθηκε από το πεδίο `Memory` να διαβαστούν 40 bytes στην διεύθυνση αυτή, δηλαδή 10 αριθμοί. Στην τελευταία επανάληψη επιβεβαιώνεται η ορθότητα των αποτελεσμάτων που αποθηκεύονται, ενώ ο `t5` έχει πλέον αυξηθεί αφού δείχνει στο τελευταίο στοιχείο του πίνακα `C`.

The screenshot shows the Visual Studio Code interface with the following components:

- Editor:** Displays assembly code for `main.S`. The code includes labels `lab2_1` and `main`, with instructions like `la t1,B`, `add t1,t1,4*(N-1)`, `la t2,A`, `add t2,t2,4*(N-1)`, `la t5,C`, and a `for_loop` section with `bgt t0,t2,end_label`, `lw t3,0(t0)`, `lw t4,0(t1)`, `add t3,t3,t4`, `bge t3,zero,skip_reverse`, `neg t3,t3`, `skip_reverse:`, `sw t3,0(t5)`, `addi t0,t0,4`, `addi t1,t1,-4`, `addi t5,t5,4`, and `end_label:`.
- Left Sidebar:** Contains panels for **VARIABLES** (showing `t1=0x00000000`, `t2=0x00000000`, `t3=0x00000000`, `t4=0x00000000`, `t5=0x00000000`, `pc=0x00000000`), **WATCH**, **CALL STACK** (showing `_libc_init_array@0x00000000`), **BREAKPOINTS**, **PERIPHERALS**, and **REGISTERS** (showing `s11=0x00000000`, `t3=2`, `t4=0x00000000`, `t5=0x00000000`, `pc=0x00000000`).
- Right Sidebar:** Contains the **Memory** panel showing a dump of 40 bytes starting at offset `0x2198`. The dump shows a sequence of zeros, followed by a sequence of ones, and ends with a sequence of zeros.
- Bottom Panel:** Contains the **PROBLEMS** panel showing a `Temporary breakpoint 1, main () at src/main.S:17` and a `la t0,A` instruction.

Κώδικας Assembly

```
1  # με το directive .globl, μπορούμε να αναφερθούμε στην main και από άλλα αρχεία
2  .globl main
3
4  # Ορίζουμε μια σταθερά, το N, ίση με το πλήθος των στοιχείων
5  .equ N, 10
6
7  # Σε αυτό το section ορίζουμε global δεδομένα που αποθηκεύονται στη μνήμη
8  .data          # Συγκεκριμένα πίνακες απο words, δηλαδή 4 bytes
9  A: .word 0,1,2,7,-8,4,5,-12,11,-2
10 B: .word 0,1,2,7,-8,4,5,12,-11,-2
11
12 .bss
13 # Σε αυτό το section μπορούμε να δεσμεύσουμε χώρο στη μνήμη, τα δεδομένα αρχικοποιούνται σε 0
14 C: .space 4*N
15 # ο C θα είναι πίνακας που χωράει 10 words
16
17 .text
18 main:
19     la t0,A          # φορτώνουμε τη διεύθυνση του πίνακα A
20     la t1,B          # φορτώνουμε τη διεύθυνση του πίνακα B
21     add t1,t1,4*(N-1) # αυξάνοντας την τιμή κατά 4*(N-1), θα δείχνει στο τελευταίο στοιχείο
22     la t2,A          # θα βάλουμε σαν άνω όριο του βρόγχου τη διεύθυνση του A[N-1]
23     add t2,t2,4*(N-1) # αυξάνουμε κατά 4*(N-1) για να την λάβουμε
24     la t5,C          # φορτώνουμε τη διεύθυνση του πίνακα C,
25 for_loop:          # όπου θα αποθηκεύσουμε τα αποτελέσματα
26     bgt t0,t2,end_label # τέλος όταν t0 δείχνει σε μεγαλύτερη διεύθυνση απ'το τελευταίο στοιχείο
27     lw t3, 0(t0)      # φορτώνουμε το A[i] στον καταχωρητή t3
28     lw t4, 0(t1)      # αντίστοιχα φορτώνουμε το B[N-1-i] στον καταχωρητή t4
29     add t3, t3, t4     # προσθέτουμε τις 2 τιμές
30     bge t3,zero,skip_reverse #αν είναι ≥0, τότε ισούται με απόλυτη τιμή
31     neg t3,t3         # αν είναι αρνητικό παίρνουμε το 2's complement με την neg
32 skip_reverse:
33     sw t3,0(t5)       # αποθηκεύουμε το αποτέλεσμα στον πίνακα C
34     addi t0, t0, 4     # αυξάνουμε τον δείκτη του πίνακα A ώστε να δείχνει στο επόμενο στοιχείο
35     addi t1, t1, -4    # μειώνουμε τον δείκτη του πίνακα B
36     addi t5, t5, 4     # αυξάνουμε τον δείκτη του πίνακα C
37     j for_loop        # συνεχίζουμε τον βρόγχο
38 end_label:
39
40 .end
```

Ερώτημα 2

Για να θέσουμε τα leds ως έξοδο, θέτουμε όλα τα πρώτα 16 bits της διεύθυνσης 0x80001408 ίσα με '1'. Η υλοποίηση αποτελείται από ένα διπλό βρόγχο που πραγματοποιεί σταδιακά την ενεργοποίηση όλων των leds, και έπειτα άλλο ένα διπλό βρόγχο που πραγματοποιεί την απενεργοποίηση τους. Οι εξωτερικοί βρόγχοι "τρέχουν" 16 φορές, όσες και ο συνολικός αριθμός των leds, ενώ οι εσωτερικοί βρόγχοι πραγματοποιούν την ολίσθηση ενός bit, όσες φορές χρειάζεται κάθε φορά. Η κατάσταση των leds, αποτελείται από ένα σταθερό τμήμα bits που δεν αλλάζει, αφού έχει αλλάξει σε προηγούμενη φάση, και από ένα κινητό τμήμα bits στα οποία παρατηρείται η ολίσθηση του bit (αναμμένου ή μη αντίστοιχα). Με τη χρήση κατάλληλης μάσκας και της εντολής AND διαχωρίζουμε τα αντίστοιχα τμήματα κάθε φορά.

Κώδικας Assembly

```
1  .globl main
2
3  .text
4  main:
5  #GPIO_INOUT 0x80001408
6  #GPIO_LEDS 0x80001404 ---2lsb
7      lui t0, 0x80001
8      lui t1, 0x10          # t1=0x00010000
9      addi t1, t1, -1       # t1=0xFFFF, η τιμή που θα βάλουμε στην GPIO_INOUT διεύθυνση
10     sw t1, 1032(t0)       # 1032 = 0x408, με offset αλλάζουμε το περιεχόμενο της διεύθυνσης 0x80001408
11     sw zero, 1028(t0)     # αρχικά όλα τα leds είναι σβηστά
12     li s1, 15             # στον s1, θα κρατάμε πόσες ολισθήσεις χρειάζεται το bit-0 κάθε φορά
13     li s2, 0xFFFF        # ψευδοεντολή που αντιστοιχεί σε lui και addi -1
14 next_bit:
15     blt s1, zero, all_on  # θα ολοκληρώσουμε μόλις ο s1 γίνει αρνητικός
16     mv s3, s1             # ψευδοεντολή που αντιστοιχεί σε add rd, rs, zero, κρατάμε αντίσγραφο του s1
17     li t1, 1              # σταθερά για να ανάψουμε το bit0
18     lw t3, 1028(t0)       # φορτώνουμε στον t3 τα τρέχοντα leds
19     or t3, t3, t1         # με χρήση της or θέτουμε το lsb = 1
20     sw t3, 1028(t0)       # ανάβουμε τα leds που είχαμε συν το lsb
21 shift_loop:
22     beq zero, s3, shift_done # if s3 == 0, άλμα στο shift_done
23     lw t3, 1028(t0)       # φορτώνουμε στον t3 την τιμή των leds
24     xori t4, s2, -1       # παίρνουμε το One's complement του mask, αντίτιχο της NOT
25     and t4, t3, t4        # ώστε να κρατήσουμε τα leds που δεν θα ολισθηθούν
26     and t3, t3, s2        # κρατάμε τα leds που θα ολισθηθούν χρησιμοποιώντας τη μάσκα (s2)
27     slli t3, t3, 1        # τα ολισθαίνουμε κατά 1 bit αριστερά
28     or t3, t4, t3         # συνδυάζουμε την ολισθημένη τιμή με τα σταθερά leds
29     sw t3, 1028(t0)       # ανάβουμε τα leds
30     addi s3, s3, -1       # μειώνουμε τον μετρητή ολισθήσεων
31     j shift_loop
32
33 shift_done:
34     addi s1, s1, -1       # εδώ το bit έχει ολισθήσει μέχρι την τελική του θέση, πλέον σταθερό
35     srli s2, s2, 1        # ενημερώνουμε την μάσκα η οποία μειώνεται κατά 1 bit
36     j next_bit           # συνεχίζουμε με το επόμενο bit (θα αρχίσει από lsb)
37 all_on:
38     # τώρα όλα τα 16 bits των leds έχουν ανάψει
39     li s1, 15             # αντίστοιχες αρχικοποιήσεις με πριν, τώρα σβήνουμε όμως
40     li s2, 0xFFFF        # bit mask
41 next_bit_2:
42     blt s1, zero, all_off  # τέλος όταν ο s1(ολισθήσεις που θέλει κάθε bit) γίνει αρνητικός
43     mv s3, s1             # κρατάμε ένα αντίγραφο των ολισθήσεων που θέλει το bit(ψευδοεντολή)
44     li t1, 0x7FFF         # αρχικά σβήνει το msb μόνο, σαν ψευδοεντολή γιατί η σταθερά είναι μεγάλη
45     lw t3, 1028(t0)       # φορτώνουμε την κατάσταση των leds
46     and t3, t3, t1        # κρατάμε την κατάσταση των leds σβήνωντας το msb
47     sw t3, 1028(t0)       # με αποθήκευση στη διεύθυνση των leds, τα ενημερώνουμε
```

```

48 shift_loop_2:
49     beq zero, s3, shift_done_2 # if s3 ==0, τότε άλμα σε shift_done_2
50     lw t3, 1028(t0)             # αποθηκεύουμε στον t3, την κατάσταση των leds
51     xori t4, s2, -1             # παίρνουμε το One's complement του mask, αντίτιχο της NOT
52     and t4, t3, t4             # κρατάμε τα leds που δεν θα ολισθηθούν
53     srli t3, t3, 1             # ολισθαίνουμε κατά 1 bit δεξιά τα υπόλοιπα
54     lui t5, 0x8                # ανάβουμε το msb, γιατί γίνεται 0 από την δεξιά ολίσθηση
55     or t3, t3, t5               # το msb θα σβήσει οριστικά από τον εξωτερικό βρόγχο, στην τελευταία επανάληψη
56     and t3, t3, s2             # κόβουμε τον ls άσσο της ολίσθησης που θα επηρέαζε τα σταθερά bits [στην or]
57     or t3, t4, t3              # συναδυάζουμε την ολισθημένη τιμή με τα σταθερά bits
58     sw t3, 1028(t0)           # απεικονίζουμε στα leds την νέα τιμή
59     addi s3, s3, -1           # μείωση του μετρητή ολίσθησης
60     j shift_loop_2
61
62 shift_done_2:
63     addi s1, s1, -1
64     slli s2, s2, 1             # μείωση της μάσκας κατά 1 bit από τα δεξιά
65     lui t1, 0x10               # t1=0x00010000
66     addi t1, t1, -1            # t1=0xFFFF
67     and s2, s2, t1             # σβήνουμε τον αριστερά άσσο που είναι εκτός ορίων
68     j next_bit_2              # και ολοκληρώνεται η μείωση της μάσκας, μετά άλμα
69 all_off:
70
71
72 .end

```