National Technical University of Athens

School of Electrical
and Computer Engineering

Microprocessors Laboratory

Semester 7 - Flow Y

Lab Report 6
RISC-V

Nafsika Abatzi - 031 17 198

Dimitris Dimos - 031 17 165

Athens
January, 2021

1 Ερώτημα 1: Άθροισμα 32-bit Στοιχέιων Πινάκων

Στο πρώτο ερώτημα καλούμαστε να υλοποιήσουμε ένα πρόγραμμα σε assembly. Το πρόγραμμα θα διαθέτει δύο πίνακες A, B τύπου word, δηλαδή με θέσεις 32-bit, και θα πρέπει να τοποθετήσει στις θέσεις ενός τρίτου πίνακα C τις τιμές:

$$C[i] = |A[i] + B[N-1-i]|$$

όπου N είναι το μήκος όλων των πινάκων.
Παραθέτουμε τον κώδικα που υλοποιεί το ζητούμενο και περιέχει επεξηγηματικά σχόλια.

1.1 Assembly Κώδικας

```
.globl main

.equ N,10

.data
A: .word 0,1,2,7,-8,4,5,-12,11,-2
B: .word 0,1,2,7,-8,4,5,12,-11,-2

.bss
C: .space 4*N

.text
main:
    la t0, A     # address of A --> t0 (t0 points at A[0])
    la t1, B     # address of B --> t1 (t1 points at B[0])
    add t1, t1, 4*(N-1)      # now t1 points to B[N-1]
    li t2, 0     # i = 0 (iterator on the C array)
    li t6, 4*N   # t6 = N (when i == t6, program finishes)

process:
    beq t2, t6, end_process # end condition (i == N)
    lw t3, 0(t0)     # t3 <-- A[i]
    lw t4, 0(t1)     # t4 <-- B[N-1-i]

    add t5, t3, t4   # temp = A[i] + B[N-1-i]
    bge t5, zero, sign_ok    # if temp >= 0 then go on
    neg t5, t5       # otherwise reverse sign

sign_ok:
    la t3, C     # t3 <-- address of C (t3 points at C[0])
    add t3, t3, t2   # now t3 points at C[i]
    sw t5, 0(t3)     # t5 --> C[i], meaning C[i] = | A[i] + B[N-1-i] |
    addi t0, t0, 4   # t0 points to the next 4 bytes (1 word): the next A cell
    addi t1, t1, -4 # t1 points to the previous 4 bytes (1 word): the previous B cell
    addi t2, t2, 4   # i++
    j process

end_process:

.end
```

2 Ερώτημα 2: LEDs play

Στο δεύτερο ερώτημα καλούμαστε να υλοποιήσουμε ένα ακόμη πρόγραμμα σε assembly. Το πρόγραμμα θα εκτελεί την παρακάτω διεργασία.

1. Άναμμα λιγότερο σημαντικού LED
2. "Ολίσθηση" ένδειξης προς τα αριστερά (δηλαδή σβήσιμο λιγότερο σημαντικού LED και άναμμα 2ου λιγότερο σημαντικού)
3. Επανάληψη βήματος 2 μέχρις ότου η ένδειξη φτάσει το πιο σημαντικό LED
4. Κρατώντας το πιο σημαντικό LED αναμμένο, επανάληψη βήματος 1
5. Επανάληψη βήματος 2 μέχρις ότου η ένδειξη φτάσει το 2ο π ιο σημαντικό LED
6. Κρατώντας τα 2 πιο σημαντικά LED αναμμένα, επανάληψη βήματος 1

Στη συνέχεια, γίνεται η αντίστροφη διαδικασία:

1. Άναμμα λιγότερο σημαντικού LED
2. "Ολίσθηση" ένδειξης προς τα αριστερά (δηλαδή σβήσιμο λιγότερο σημαντικού LED και άναμμα 2ου λιγότερο σημαντικού)
3. Σβήσιμο λιγότερο σημαντικού LED
4. "Ολίσθηση" μη-ένδειξης προς τα δεξιά (δηλαδή άναμμα σημαντικότερου LED και σβήσιμο 2ου σημαντικότερου)
5. κ.ο.κ.

Παραθέτουμε τον κώδικα που υλοποιεί το ζητούμενο και περιέχει επεξηγηματικά σχόλια.

2.1 Συνοπτική Εξήγηση

Εξηγούμε συνοπτικά την λογική του προγράμματος. Σε κάθε επανάληψη (όπου επανάληψη θεωρούμε μια πλήρη κίνηση του κινούμενου LED από το ένα άκρο στο άλλο), υπάρχουν κάποια LEDs που είναι διαρκώς ON κι ένα LED που "κινείται". Τα LEDs που σε μια επανάληψη είναι διαρκώς ON, τα ονομάζουμε *background*. Το LED που κινείται, το ονομάζουμε $shifter$. Επομένως, σε κάθε επανάληψη έχουμε υπερθεση του *background* και του $shifter$ στην έξοδο.

Όταν η κίνηση του $shifter$ είναι προς τα αριστερά το αποτέλεσμα είναι:

$$result = background + shifter$$

δηλαδή τα LEDs που είναι ON διαρκώς στην επανάληψη αυτή, μαζί με το LED που κινείται. Όταν η κίνηση είναι προς τα δεξιά το αποτέλεσμα είναι:

$$result = background \oplus shifter$$

δηλαδή τα LEDs που είναι ON (δηλαδή 1), "βγάζοντας" το LED που κινείται (που έχει επίσης τιμή 1, οπότε η υπέρθεση προκύπτει με $\oplus$)

Ο $shifter$ κάθε φορά φτάνει μια θέση πριν από αυτή που έφτασε στην προηγούμενη επανάληψη. Οπότε ορίζουμε κι ένα όριο *limit* στο οποίο όταν φτάνει ο $shifter$ σταματάει να κινείται, ενσωματώνεται στο *background* και ξεκινάει νέα επανάληψη.

2.2 Assembly Κώδικας

```
.globl main

.text
main:

    li t0, 0x80001404    # t0 = LEDs address
    li t1, 0x80001408    # t1 = IO address
    li t2, 0xFFFF        # 0xFFFF: setting LEDs to output
    sw t2, 0(t1)

    # t0 will always contain LEDs address
    # t1 holds the value of static LEDs (background)
    # t2 holds the value of the moving LED (shifter)
    # t3 hold the limit value of the shifter

    mv t1,zero  # initially, background LEDs are OFF
    li t3,0x8000     # initial limit is = 1000 0000 0000 0000
    li t5,0xFFFF     # when output is t5 (== 0xFFFF), reverse

left:
    li t2,1      # initialize shifter
rep1:
    add t4,t1,t2     # result = background + shifter
    sw t4, 0(t0)     # output result to LEDs
    beq t4,t5,end_left  # if (LEDs == 1111 1111 1111 1111) then begin reverse
    slli t2,t2,1     # shifter goes left
    bne t2,t3,rep1   # if shifter has not reached limit, repeat

    add t1,t1,t2     # otherwise, update background (the static LEDs)
    sw t1, 0(t0)     # output background
    srli t3,t3,1     # lower the shifter limit for the next round
    j left
end_left:


    mv t1,t4     # update background
    li t5,0      # result == t5 will signify the end of program

# inverse process, now shifter goes right
right:
    li t2,0x8000     # initialize shifter
rep2:
    xor t4,t1,t2     # result = background xor shifter
    sw t4, 0(t0)     # output result to LEDs
    beq t4,t5,end_right # if we output a result of 0x0000 --> end
    srli t2,t2,1     # shifter goes right
    bne t2,t3,rep2   # if shifter has not reached limit, repeat

    xor t1,t1,t2     # otherwise, update background
    sw t1, 0(t0)     # output background
    slli t3,t3,1     # increase shifter limit for the next round
    j right

end_right:

.end
```