ΕΠΛ605 Εργασία 1 Ημερομηνία Παράδοσης 12/9/2018 στην αρχή του μαθήματος

Ε.1 Σας δίνεται ο πιο κάτω κώδικας. Ξαναγράψτε τον ώστε να μειωθεί ο αριθμός των εντολών του αλλά διατηρώντας την ίδια λειτουργιά (συμπληρώστε την τελική σας απάντηση μέσα στο διπλανό πλαίσιο μια γραμμή κάθε εντολή):

```
add     $sp, $sp, -4
sw      $ra,0($sp)
add     $t0,$0,$a0
addi    $t0, $t0, 1
add     $v0,$t0,$0
lw      $ra,0($sp)
add     $sp,$sp,4
jr      $ra
```

addi $v0,$a0,1
jr $ra

Ε.2 Συμπληρώστε στον πίνακα τις κωδικοποιήσεις στο δεκαδικό των πιο κάτω εντολών MIPS:

	add $t0,$0,$0	R,opcode:0, function:32	**0**	**0**	**0**	**8**	**0**	**32**
L1:	slt $t1,$t0,$a1	R,opcode:0, function:42	**0**	**8**	**5**	**9**	**0**	**42**
	beq $t1,$0, L2	I, opcode: 4	**4**	**9**	**0**	**2**		
	addi $t0,$t0, 4	I, opcode: 8						
	beq $0,$0, L1	I, opcode: 4	**4**	**0**	**0**	**-4**		
L2:								

Ε.3 Δεδομένης της πιο κάτω αρχικής κατάστασης, δείξετε τι είναι το τελικό περιεχόμενο των διαφόρων καταχωρήτων μετά την εκτέλεση των πιο κάτω εντολών MIPS

```
add   $a0,$a0,1
slt   $t0, $a0,$a1
```

Καταχώρητης	Αρχική Κατάσταση	Τελική Κατάσταση
$a0	0x0000 000d	**0x0e**
$a1	0x8000 000f	**Same (this is a negative for slt)**
$t0	0x0000 0040	**0x0**
$t1	0x0000 0020	**same**
PC	0x0080 ffc0	**0x080 ffc8**

Ε.4 Ποια τιμή στο δεκαδικό πρέπει να περιέχεται στο $a1 ώστε ο πιο κάτω κώδικας να προκαλέσει την εκτέλεση 105 εντολών; Επίσης, γράψετε διπλά από κάθε εντολή, στην μικρή γραμμή που σας δίνεται, πόσες φορές εκτελείται η κάθε εντολή για την συγκεκριμένη τιμή του $a1.

$a1=25

```
addu   $v0,$0,$0          1_____

beq    $a1,$0, END        1_____

sll    $t0,$a1,2          1_____

addu   $t0,$a0,$t0        1_____

LOOP:

lw     $t1,0($a0)         25____

addu   $v0,$v0,$t1        25____

addiu  $a0,$a0,4          25____

bne    $a0,$t0,LOOP       25____

END:

jr     $ra          // return    1_____
```

Ε.5 Γράψετε σε γλώσσα ψηλού επιπέδου (πχ C, Java) μια συνάρτηση/μέθοδο που είναι ισοδύναμη με τον συμβολικό κώδικα της Ε.4

```
int foo(int a[], int size){
        int s= 0,i;
        for(i=0;i<size;++i)
                s+=a[i];
        return s;
}
```

Ε.6
Σας δίνετε το πιο κάτω τμήμα συμβολικού προγράμματος.

```
        add     $t0,$0,$0           1_____
L1:
        andi    $t1,$a0,0x00ff      4_____

        add     $t0,$t0,$t1         4_____

        srl     $a0,$a0,8           4_____

        bne     $a0,$0,L1           4_____
```

Εάν η αρχική τιμή του καταχωρητή $a0 είναι το 0x03057fbc,

α) Δείξετε διπλά από την κάθε εντολή στην μικρή γραμμή που σας δίνεται πόσες φορές εκτελείται η κάθε εντολή;

β) Τι θα είναι τα περιεχόμενα των καταχωρήτων t0,t1,a0 μετά την εκτέλεση του πιο πάνω κώδικα.

t0	t1	a0
0x0000 0143	0x0000 0003	0x0000 0000

Ε7) Υπολογίστε την αναπαράσταση σε IEEE754 μονή ακριβείας του αριθμού 6.

Πρώτα εκφράστε τον αριθμό στην μορφή $(-1)^{Sign}$ x (1+mantissa) x $2^{exponent}$

Η mantissa είναι ένας πραγματικός αριθμός μικρότερος του 1.

Τι είναι η τιμή του πρόσημου (sign);

0

Τι είναι η τιμή στο δεκαδικό του εκθέτη (exponent);

2

Τι είναι η τιμή στο δεκαδικό της mantissas (significant);

0.5

Μετατρέψετε το exponent σε biased exponent, τι είναι η τιμή του στο δεκαδικό;

129

<u>Η τελική σας απάντηση να δοθεί σε δεκαεξαδική μορφή</u>.

0x40c0 0000

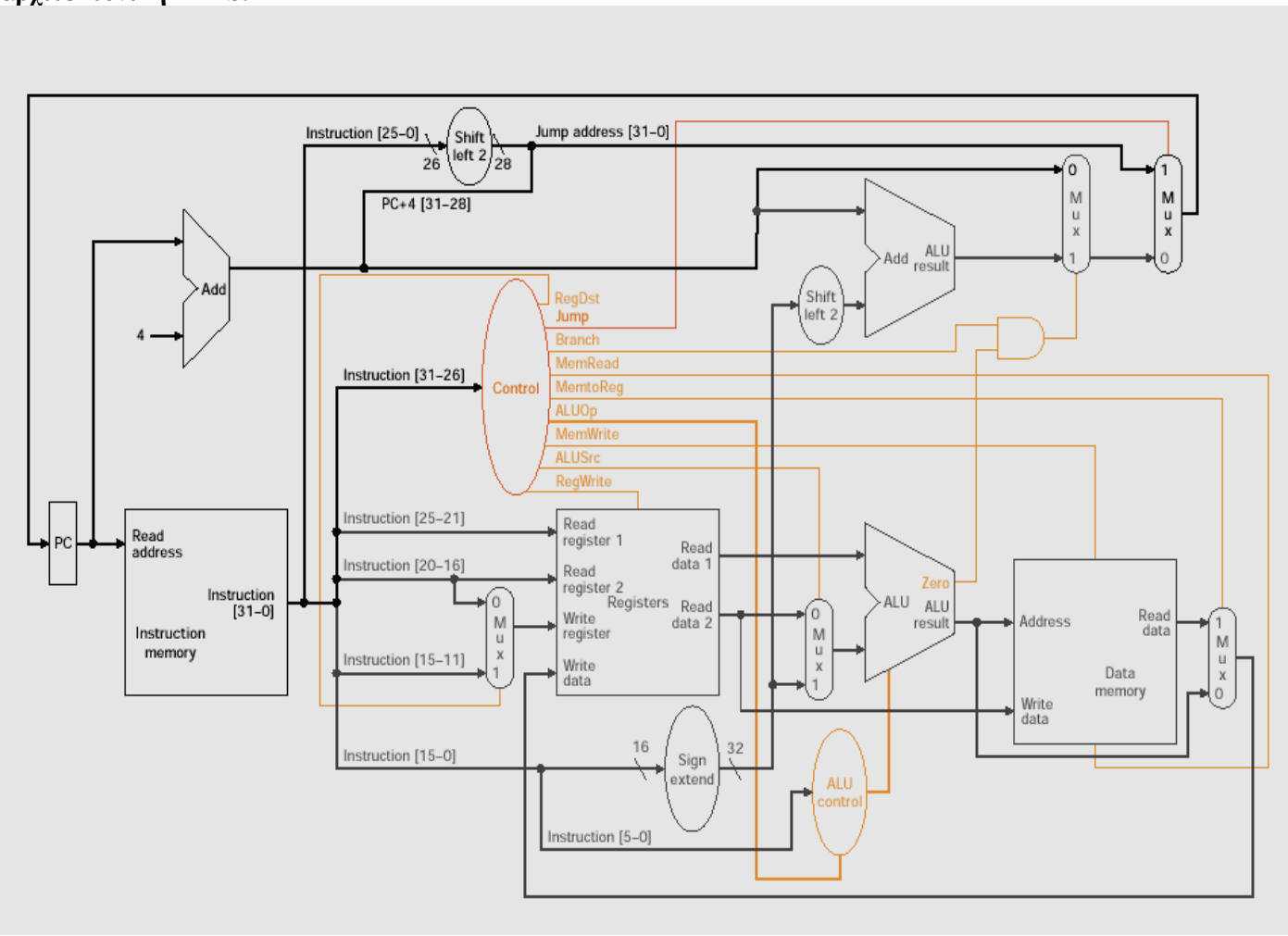
Ε.8 Τι είναι το ΠΡΟΣΗΜΟ (θετικό ή αρνητικό) του αποτελέσματος της αφαίρεσης των πιο κάτω πραγματικών τιμών σε αναπαράσταση IEEE754 μονής ακριβείας; Εξηγήστε την απάντηση σας (HINT: δεν χρειάζεται να υπολογίσετε το τελικό αποτέλεσμα της αφαίρεσης)

0x3ED0A00E =>	0 0111 1101 101 0000 1010 0000 0000 1110
- 0xBEE0B00F	- 1 0111 1101 110 0000 1011 0000 0000 1111

Use sign bit

positive - (negative) => positive

Ε.9 Σας δίνεται ο πιο κάτω διάδρομος δεδομένων που υλοποιεί ένα υποσύνολο των εντολών της αρχιτεκτονική MIPS.



Επισης σας δινεται πιο κάτω πινακας που οριζέι την συμπεριφορά της ALU.

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control Input
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
Branch equal	01	branch equal	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set-on less-than	101010	set-on-less-than	111

Συμπληρώστε στον πιο κάτω πίνακα τις τιμές των σημάτων ελέγχου (1 ή 0) ώστε να εκτελείται μια εντολή φόρτωσης μιας τιμής από την μνήμη σε ένα καταχωρητή, πχ lw $6,16($10)

RegDst	ALUSrc	Mem toReg	Reg Write	Mem Read	MemWrite	Branch	ALU Op1	ALU Op0	Jump

0 1 1 1 1 0 0 0 0 0

Ε.10 Γράψετε την συνάρτηση asci_parity στην συμβολική γλώσσα MIPS που παίρνει μια ακέραια παράμετρο που περιέχει μέσα ένα χαρακτήρα ASCII (7 bit) στα least significant bits της και επιστρέφει μια ακέραια τιμή που περιέχει τον χαρακτήρα ASCII στα 7-least-significant-bits και το odd parity του στην θέση (bit-position) 7. Πχ εάν η παράμετρος της συνάρτησης είναι η τιμή 96_{10} (0x60, 0110 0000_2) η συνάρτηση θα επέστρεφε την τιμή 224_{10} (0xE0, 1110 0000_2), ενώ εάν η τιμή της παραμέτρου ήταν το 22_{10} (0x16, 0001 0110_2) η συνάρτηση θα επέστρεφε την τιμή 22_{10} (0x16, 0001 0110_2). Στην πρώτη περίπτωση το parity είναι 1 για να γίνει περιττός ο αριθμός των 1 ενώ στην δεύτερη περίπτωση το parity είναι 0 γιατί ο αριθμός των 1 είναι ήδη περιττός. .Η συνάρτηση θα πρέπει να ακολουθεί την τυποποίηση MIPS, δεν επιτρέπεται η χρήση της στοίβας, <u>το πολύ επιτρέπεται η χρήση 14 εντολών (λύνεται και με πιο λίγες)</u>. Πολύπλοκες λύσεις η λύσεις που δεν ακολουθούν την απαιτούμενη τυποποίηση θα παίρνουν 0 βαθμούς.

ΤΕΛΙΚΗ ΑΠΑΝΤΗΣΗ:

```
add $v0,$a0,$0
addi $t0,$0,1
LOOP:
beq $a0,$0, END
andi $t1,$a0,1
xor $t0,$t0,$t1
srl $a0,$a0,1
j LOOP
END:
sll $t0,$t0,7
or $v0,$v0,$t0
jr $ra
```

Ε.11 Σας δίνεται η συνάρτηση expand_table_values **σε C και η μερική υλοποίηση της σε συμβολική γλώσσα MIPS. Υποθέστε πως η συνάρτηση** expand_value**, που παίρνει δύο ακέραιες παραμέτρους και επιστρέφει μια ακέραια τιμή, είναι ήδη υλοποιημένη. Οι δύο συναρτήσεις ακολουθούν την τυποποίηση MIPS (MIPS convention). Συμπληρώστε την expand_table_values οπού υπάρχει γραμμή. Δεν πρέπει να υλοποιήσετε την** expand_value**. HINT: οι πρώτες 4 παράμετροι μιας συνάρτησης στην MIPS περνιόνται στην σειρά που παρουσιάζονται στην C στους καταχωρητές $a0-$a3.**

```
void expand_table_values(int table[], int size, int expansion){
        int i;
        for(i=0;i<size; ++i)
                table[i] = expand_value(table[i], expansion);
}
```

```
expand_table_values:
# save in stack saved registers and return address
        add   $sp,$sp,-16____
        sw    _$ra__,__0_($sp)
        sw    _$s0__,__4_($sp)
        sw    _$s1__,__8_($sp)
        sw    _$s2__,_12__($sp)

# save parameters in saved registers and determine table end address
        add   $s0,__$a0_,$0           # table starting address
        sll   $s1,_$a1__,2            # multiply size by 4
        add   $s1,_$s0 ,_$s1_   # address at the end of table (start address + size x 4)
        add   $s2,__$a2_,$0           # expansion value

# the function main loop
LOOP: beq$s0,__$s1__,__END__ # reached end of table?
        lw    $a0,0(_$s0___)  # load table entry into first argument register
        add   __$a1__,$s2,$0  # move expansion value into second argument register
        jal   expand_value    # call function
        sw    __$v0__,0($s0)  # store return value into table
        add   _$s0_,$s0,_4_   # increment address to read next table entry
        j     ___LOOP_

# restore saved registers and return address and return
END:lw    _$ra__,_0__($sp)
        lw    _$s0__,__4_($sp)
        lw    _$s1__,__8_($sp)
        lw    _$s2__,__12_($sp)
        add   $sp,$sp,_16___
jr      $ra
```