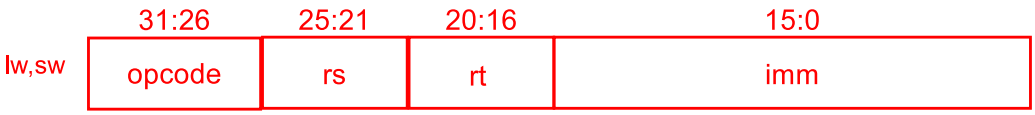
Πανεπιστήμιο Κρήτης	**Δημήτρης Νικολόπουλος, Χριστόφορος Κάχρης**
Τμήμα Επιστήμης Υπολογιστών	**14 Απριλίου 2010, 19:00–12:00**
ΗΥ225: Οργάνωση Υπολογιστών, Άνοιξη 2010	**Η εξέταση γίνεται με κλειστές σημειώσεις**
Εξέταση Προόδου	**Τα θέματα επιστρέφονται**
Ονοματεπώνυμο:	
Αριθμός Μητρώου:	

Ερώτημα 1 (30)

1. Για κάθε μία από τις παρακάτω εντολές του MIPS, εξηγήστε με ποιον τρόπο ο επεξεργαστής διευθυνσιοδοτεί τη μνήμη (δηλαδή βρίσκει τη ζητούμενη από την εντολή διεύθυνση μνήμης, είτε πρόκειται για μνήμη δεδομένων είτε για μνήμη εντολών): (α) `lw, sw`, (β) `jr` (γ) `j`, (δ) `bne, beq` **(12)**

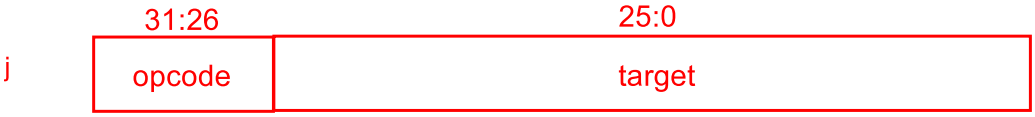
(α)



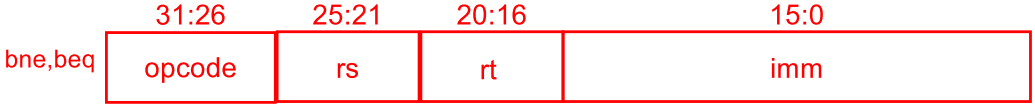
$EA = \$rs + imm$ (immediate sign extended)

(β)

$PC = \$rs$



$PC = (target << 2) | (PC \& f0000000_{hex})$



$PC = PC + 4 + 4 \times imm$ (immediate sign extended)

2. Έστω ότι ο επεξεργαστής MIPS χρησιμοποιεί 64 αντί για 32 καταχωρητές γενικού σκοπού, ενώ το μήκος των εντολών παραμένει 32 bits. Έστω επίσης ότι το opcode των εντολών παραμένει μεγέθους 6 bits. Σε αυτή την περίπτωση ποια θα ήταν η μέγιστη απόσταση σε bytes την οποία θα μπορούσε να διανύσει ένα άλμα στις εντολές `beq, bne`; Εξηγήστε λεπτομερώς την απάντησή σας **(8)**

Εφόσον ο επεξεργαστής έχει τώρα 64 καταχωρητές, για κάθε καταχωρητή χρειαζόμαστε 6 bits για την κωδικοποίησή του ($2^6 = 64$). Άρα οι εντολές `bne, beq` χρειάζονται 12 bits για να κωδικοποιήσουν τους καταχωρητές και 6 bits για το opcode. Κατά συνέπεια διαθέτουν 14 bits για το immediate operand. Η μέγιστη απόσταση σε bytes μπροστά από τον PC την οποία μπορεί να διανύσει ένα άλμα είναι συνεπώς $(2^{13} - 1) \times 4$ ή 32,764 bytes. Η μέγιστη απόσταση σε bytes πίσω από τον PC την οποίοα μπορεί να διανύσει ένα άλμα είναι $2^{13} \times 4$ ή 32,768 bytes. Κατά συνέπεια η μέγιστη απόσταση που μπορεί να διανύσει οποιοδήποτε άλμα είναι 32,768 bytes.

3. Υποθέτουμε πάλι ότι ο επεξεργαστής MIPS έχει 32 καταχωρητές γενικού σκοπού, όπως έχετε διδαχθεί στο μάθημα. Έστω ότι θέλετε να υλοποιήσετε μια διακλάδωση με συνθήκη ελέγχου ισότητας (equal) στον επεξεργαστή για την οποία η διεύθυνση της εντολής προς την οποία θα γίνει το άλμα αν η συνθήκη είναι αληθής βρίσκεται 512 Kilobytes (524,288 bytes) "μπροστά" από τη διεύθυνση της εντολής του άλματος. Πώς θα υλοποιούσατε ένα τέτοιο άλμα; Εξηγήστε λεπτομερώς την απάντησή σας **(10)**

Με τον ίδιο συλλογισμό με τον οποίο λύσαμε την προηγούμενη άσκηση, βρίσκουμε ότι εφόσον ο επεξεργαστής έχει 32 καταχωρητές, έχει 16 bits διαθέσιμα για το άλμα στην `beq`, άρα η μέγιστη απόσταση

ενός άλματος είναι 131,072 bytes. Κατά συνέπεια ο επεξεργαστής δεν μπορεί να κάνει άλμα 524,288 bytes μόνο με μία εντολή beq. Έστω L η ετικέττα της εντολής στην οποία πρέπει να γίνει το άλμα. Η λύση είναι να συνδυάσει μία εντολή beq με μία εντολή j ή μία εντολή bne με μία εντολή j:

```
beq $rs,$rt,L1
...
... #distance up to 128KB
...
L1: j L
...
...
...
L:
```

```
bne $rs,$rt,L2
L1: j L
L2: ...
...
...
L:
```

Ερώτημα 2 (28)

Δίνεται η αρχική κατάσταση μίας περιοχής της μνήμης δεδομένων του MIPS όπως φαίνεται δεξιά στο σχήμα. Δώστε τα περιεχόμενα των καταχωρητών $1 και $2 και των συγκεκριμένων 8 bytes μνήμης μετά την εκτέλεση καθενός από τα 7 τμήματα κώδικα (1–7) που φαίνονται αριστερά στο σχήμα. Θεωρήστε ότι κάθε τμήμα κώδικα εκτελείται μόνο του και ανεξάρτητα από τα άλλα, ότι όλες οι σταθερές στον κώδικα είναι στο δεκαεξαδικό σύστημα, ότι πριν την εκτέλεση κάθε τμήματος οι καταχωρητές περιέχουν την τιμή 0 και πριν την εκτέλεση κάθε τμήματος η μνήμη περιέχει ό,τι δείχνει το σχήμα δεξιά και όλες οι υπόλοιπες θέσεις μνήμης δεδομένων έχουν περιεχόμενο 0. Θεωρήστε τέλος ότι ο κώδικας εκτελείται σε έναν επεξεργαστή MIPS που είναι little-endian. Σε κάθε περίπτωση ελέγξτε εάν οι προσβάσεις στη μνήμη είναι νόμιμες και εάν δεν είναι εξηγήστε το λόγο.

```
1.      lw $1, 1004($0)
        lw $2, 1000($0)
2.      lw $1, 1002($0)
3.      lw $1, 1000($0)
        lw $2, 1004($0)
        add $1, $1, $2
        sw $1, 1000($0)
        sw $1, 1004($0)
4.      lw $1, 1004($0)
        lw $2, 1000($0)
        addi $1, $1, 1
        sw $1, 3($1)
```

```
5.      addi $1, $0, 1004
        sw $0, 1($1)
6.      lb $1, dead($0)
        lb $2, beef($0)
        sw $1, 1004($0)
        sw $2, 1000($0)
7.      addi $1, $0, 1000
        lw $1, 0($1)
        lw $2, 1004($0)
        beq $1, $2, L2
        L1:sw $2, 0($2)
        L2:sw $1, 0($1)
```

Address (Hex)	Data (Hex)
...	...
00001000	04
00001001	10
00001002	00
00001003	00
00001004	00
00001005	10
00001006	00
00001007	00
...	...

1. $\$1 = 00001000_{hex}, \$2 = 00001004_{hex}$, τα περιεχόμενα της μνήμης δεν αλλάζουν.
2. Μη στοιχισμένη πρόσβαση στη μνήμη.
3. $\$1 = 00002004_{hex}, \$2 = 00001000_{hex}$

Address (Hex)	Data (Hex)
...	...
00001000	04
00001001	20
00001002	00
00001003	00
00001004	04
00001005	20
00001006	00
00001007	00
...	...

4. $\$1 = 00001001_{hex}, \$2 = 00001004_{hex}$

Address (Hex)	Data (Hex)
...	...
00001000	04
00001001	10
00001002	00
00001003	00
00001004	01
00001005	10
00001006	00
00001007	00
...	...

5. $\$1 = 00001004_{hex}, \$2 = 00000000_{hex}$, μή στοιχισμένη πρόσβαση στη μνήμη από την `sw`

Address (Hex)	Data (Hex)
...	...
00001000	00
00001001	00
00001002	00
00001003	00
00001004	00
00001005	00
00001006	00
00001007	00
...	...

6. $\$1 = 00000000_{hex}, \$2 = 00000000_{hex}$

Address (Hex)	Data (Hex)
...	...
00001000	00
00001001	10
00001002	00
00001003	00
00001004	04
00001005	10
00001006	00
00001007	00
...	...

7. $\$1 = 00001004_{hex}, \$2 = 00001000_{hex}$

Address (Hex)	Data (Hex)
...	...
00001000	00
00001001	10
00001002	00
00001003	00
00001004	04
00001005	10
00001006	00
00001007	00
...	...

Ερώτημα 3 (24)

1. Εξηγήστε τον ουσιαστικό λόγο χρήσης συμβάσεων διαχωρισμού των καταχωρητών σε caller-save (σώσιμο με ευθύνη της καλούσας διαδικασίας) και callee-save (σώσιμο με ευθύνη της καλούμενης διαδικασίας) **(6)**

Έστω ότι η καλούμενη διαδικασία βρίσκεται σε μία βιβλιοθήκη και καλείται εξωτερικά από πολλές άλλες διαδικασίες σε πολλά προγράμματα που χρησιμοποιούν τη βιβλιοθήκη. Σε αυτές τις περιπτώσεις, η καλούμενη διαδικασία δεν γνωρίζει τους καταχωρητές τους οποίους η καλούσα διαδικασία ενδεχομένως διαβάζει μετά την κλήση θεωρώντας τα περιεχόμενά τους αμετάβλητα από την καλούμενη διαδικασία. Αντίστοιχα, η καλούσα διαδικασία δεν γνωρίζει ποιων καταχωρητών τα περιεχόμενα ενδέχεται να μεταβάλλει η καλούμενη διαδικασία. Η σύμβαση λύνει αυτά τα προβλήματα, ορίζοντας το ποιους καταχωρητές οφείλει να σώσει στη στοίβα η καλούμενη διαδικασία, πριν αλλάξει το περιεχόμενό τους

και ποιους καταχωρητές οφείλει να σώσει στη στοίβα η καλούσα διαδικασία πριν την κλήση (και εφόσον διαβάζει αυτούς τους καταχωρητές μετά την κλήση). Η σύμβαση αυτή επιτρέπει την ελεύθερη χρήση ορισμένων καταχωρητών τόσο από την καλούσα όσο και από την καλούμενη διαδικασία χωρίς φόρτωση και αποθήκευσή τους από/στη στοίβα. Η αποφυγή άσκοπων φορτώσεων και αποθηκεύσεων περιεχομένων καταχωρητών από/προς τη στοίβα βελτιώνει την επίδοση του προγράμματος στον επεξεργαστή.

2. Υλοποιήστε σε Assembly του MIPS την παρακάτω αναδρομική διαδικασία υπολογισμού του μήκους ενός αλφαριθμητικού (string).

```
int strlen_r(char *s)
{
        if (*s=='\0') return 0;
        else return(1 + strlen_r(s+1));
}
```

```
strlen: lb $t0,0($a0)           #fetch next character
        bne $t0,$0,cont         # if (*s=='\0')
        add $v0,$0,$0           # return 0;
        jr $ra
cont:   addi $sp,$sp,-4         #room on stack for 4 bytes
        sw $ra,0($sp)           #save $ra, recursion
        addi $a0,$a0,1          #s+1
        jal strlen              #call strlen(s+1)
        addi $v0,$v0,1          #1 + strlen(s+1)
        lw $ra,0($sp)           #restore $ra
        addi $sp,$sp,4          #restore room on stack
        jr $ra
```

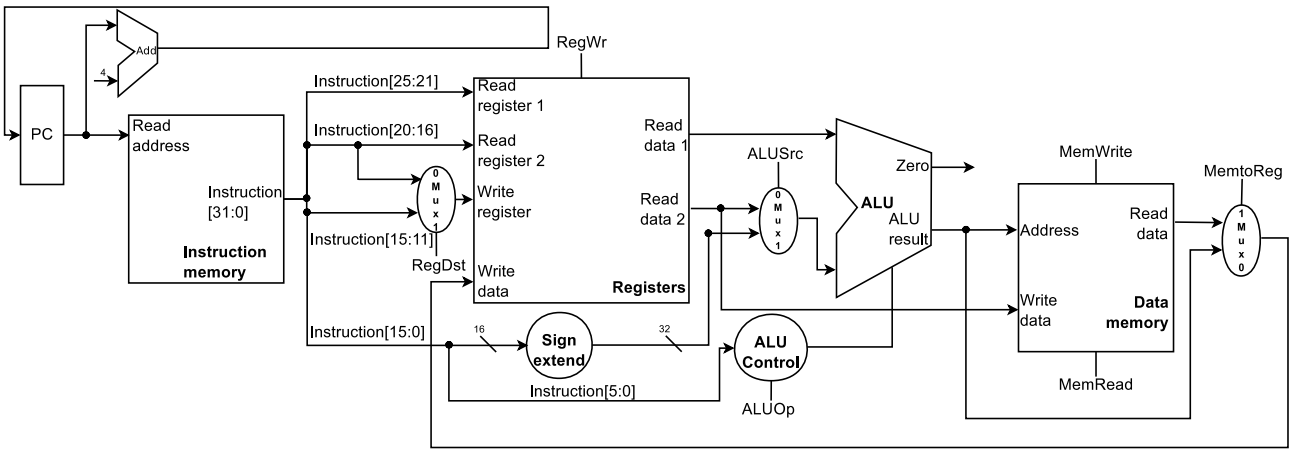
Θεωρήστε ότι η διαδικασία βρίσκεται σε βιβλιοθήκη και μπορεί να κληθεί από εξωτερικά τμήματα κώδικα και χρησιμοποιήστε όποιες συμβάσεις για το σώσιμο των καταχωρητών κρίνετε απαραίτητες. Υπενθυμίζεται ότι η κωδικοποίηση του null character στον πίνακα ASCII είναι το 0. Χρησιμοποιήστε την εντολή `lb $rt, imm($rs)` για να φορτώνετε τους χαρακτήρες του αλφαριθμητικού έναν προς έναν από τη μνήμη. Υποθέστε ότι οι χαρακτήρες του αλφαριθμητικού αποθηκεύονται στη μνήμη σε αύξουσες διευθύνσεις όπως διαβάζουμε το αλφαριθμητικό από αριστερά προς τα δεξιά (την ίδια προσέγγιση ακολουθεί και ο μεταφραστής της C ώστε η ανάγνωση των αλφαριθμητικών να μην επηρεάζεται από το endianness της μηχανής) **(18)**

Ερώτημα 4 (18)

Παρακάτω φαίνεται το datapath ενός επεξεργαστή που εκτελεί εντολές σε έναν κύκλο ρολογιού.

1. Συμπληρώστε τις τιμές που πρέπει να έχουν τα σήματα ελέγχου `RegDst`, `RegWr`, `ALUSrc`, `ALUOp`, `MemWrite`, `MemRead`, `MemtoReg` για να εκτελέσει ο επεξεργαστής εντολές `add` τύπου R-format (`add $rd, $rs, $rt`) και εντολές store (`sw $rt, address($rs)`). Αν η τιμή του σήματος δεν έχει σημασία, σημειώστε το με X. Σας δίνεται το format των εντολών του επεξεργαστή και ο πίνακας των κωδικών εντολών της ALU στα παρακάτω σχήματα **(14)**

Instruction	RegDst	RegWrite	ALUSrc	ALUOp	MemWrite	MemRead	MemtoReg
add	1	1	0	00	0	X ή 0	0
sw	X	0	1	00	1	0	X



add	0	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0
sw	35 or 43	rs	rt	address		
	31:26	25:21	20:16	15:0		
beq	4	rs	rt	address		
	31:26	25:21	20:16	15:0		

ALUOp	Function
00	add
01	subtract
10	AND
11	OR

2. Στη συνέχεια τροποποιήστε το κύκλωμα ώστε να εκτελεί και την εντολή beq στην οποία αν οι καταχωρητές είναι ίδιοι τότε ο PC γίνεται `PC = PC+4+address`. Μπορείτε να χρησιμοποιήσετε ένα σήμα ελέγχου `PCSrc` που θα ελέγχει αν ο PC θα γίνει `PC+4` ή `PC+4+address` **(4)**

Η λύση η οποία ολισθαίνει τη 32-bit έξοδο του sign extend κατά 2 θεωρήθηκε επίσης σωστή.