Πανεπιστήμιο Κρήτης	**Δημήτρης Νικολόπουλος, Χριστόφορος Κάχρης**
Τμήμα Επιστήμης Υπολογιστών	**26 Αυγούστου 2010, 09:00–12:00**
ΗΥ225: Οργάνωση Υπολογιστών, Άνοιξη 2010	**Η εξέταση γίνεται με κλειστές σημειώσεις**
Εξέταση Β' Περιόδου	**Τα θέματα επιστρέφονται, ΚΑΛΗ ΕΠΙΤΥΧΙΑ!**
Ονοματεπώνυμο:	
Αριθμός Μητρώου:	

Ερώτημα 1 (10 βαθμοί): Δειγματοληψία ή Διακοπές;

Συγκρίνετε την Είσοδο/Εξοδο με δειγματοληψία με την Είσοδο/Εξοδο με διακοπές ως προς το κόστος εξυπηρέτησης των εισόδων και εξόδων περιφερειακών συσκευών. Ποια είναι τα πλεονεκτήματα και ποια τα μειονεκτήματα κάθε μεθόδου και σε ποιες περιπτώσεις εμφανίζονται αυτά;

Το κόστος διαχείρισης μίας διακοπής είναι πολύ μεγαλύτερο του κόστους δειγματοληψίας. Ωστόσο, ενώ οι διακοπές επιφέρουν κόστος μόνο όταν είναι απαραίτητο (δηλαδή μόνο όταν υπάρχει αίτημα για λειτουργία Ε/Ε από συσκευή), η δειγματοληψία μπορεί να επιφέρει κόστος και όταν δεν υπάρχει αίτημα για λειτουργία Ε/Ε, εφόσον γίνεται περιοδικά. Η δειγματοληψία πλεονεκτεί των διακοπών όταν τα αιτήματα Ε/Ε είναι πυκνά και εμφανίζουν περιοδικότητα, με την προϋπόθεση ότι η περίοδος δειγματοληψίας έχει επιλεχθεί σωστά ώστε να μην χάνονται αιτήματα Ε/Ε και να μην γίνεται δειγματοληψία περισσότερες φορές από όσες χρειάζονται τα περιφερειακά. Οι διακοπές πλεονεκτούν της δειγματοληψίας όταν τα αιτήματα Ε/Ε είναι αραιά (σπάνια).

Ερώτημα 2 (10 βαθμοί): Ομοχειρία και εξαρτήσεις

Έστω επεξεργαστής με ομοχειρία 5 βαθμίδων (IF, ID, EX, MEM, WB) που χρησιμοποιεί προώθηση για την επίλυση εξαρτήσεων μεταξύ εντολών. Έστω ότι RegisterRs και RegisterRt είναι οι δύο καταχωρητές που διαβάζει μία εντολή τύπου R-format και RegisterRd ο καταχωρητής που γράφει μία εντολή τύπου R-format. Έστω επίσης ότι ο επεξεργαστής χρησιμοποιεί δύο σήματα ForwardA και ForwardB για να ενεργοποιήσει την προώθηση αποτελέσματος της ALU στον καταχωρητή RegisterRs και RegisterRt αντίστοιχα.

1. Δώστε τις λογικές εκφράσεις που χρησιμοποιεί η μονάδα ελέγχου για να αποφασίσει αν πρέπει να γίνει προώθηση από τον καταχωρητή μεταξύ των βαθμίδων EX και MEM (EX/MEM) προς τον καταχωρητή RegisterRs ή/και τον καταχωρητή RegisterRt για εντολές τύπου R-format. **(5 βαθμοί)**

```
IF (EX/MEM.RegWrite
and (EX/MEM.RegisterRd <> 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs) ForwardA = 10

IF (EX/MEM.RegWrite
and (EX/MEM.RegisterRd <> 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt) ForwardB = 10
```

2. Δώστε τις λογικές εκφράσεις που χρησιμοποιεί η μονάδα ελέγχου για να αποφασίσει αν πρέπει να γίνει προώθηση από τον καταχωρητή μεταξύ των βαθμίδων MEM και WB (MEM/WB) προς τον καταχωρητή RegisterRs ή/και τον καταχωρητή RegisterRt για εντολές τύπου R-format. **(5 βαθμοί)**

```
IF (MEM/WB.RegWrite
and (MEM/WB.RegisterRd <> 0)
and (EX/MEM.RegisterRd <> ID/EX.RegisterRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs) ForwardA = 01

IF (MEM/WB.RegWrite
and (MEM/WB.RegisterRd <> 0)
and (EX/MEM.RegisterRd <> ID/EX.RegisterRt)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt) ForwardB = 01
```

Ερώτημα 3 (24 βαθμοί): Assembly

Υλοποιήστε σε assembly την αναδρομική διαδικασία δυαδικής αναζήτησης σε πίνακα που δίνεται δεξιά σε γλώσσα C. Θεωρήστε ότι ο πίνακας είναι ταξινομημένος και περιέχει ακεραίους μεγέθους μίας λέξης. Σας δίνεται σαν βοήθημα η εντολή `sra $rd, $rt, shift_amt` η οποία ολισθαίνει τα bits του καταχωρητή `$rt` δεξιά τόσες θέσεις όσο το τελούμενο `shift_amt` και στο most significant bit του καταχωρήτη `$rd` θέτει το bit του προσήμου του καταχωρητή `$rt`.

```c
int binary_search(int A[], int low, int high, int target) {
    if (high < low)
        return -1;
    int middle = (low + high)/2;
    if (target < a[middle])
        return binary_search(a, low, middle-1, target);
    else if (target > a[middle])
        return binary_search(a, middle+1, high, target);
    else if (target == a[middle])
        return middle;
}
```

Δίνεται ο πλήρης κώδικας με μία διαδικασία main που καλεί την binary_search (για να δοκιμάσετε τον κώδικα και στον SPIM εάν θέλετε). Στις απαντήσεις δεν χρειάζεται η main.

```
#a0: address of A, a1: low, a2: high, a3: target
                .data
                .align 2
a:              .word 1,3,5,7,9
                .text
                .globl main
main:           la   $a0, a
                addi $a1, $0, 0
                addi $a2, $0, 4
                addi $a3, $0, 124
                jal  binary_search
                add  $s0, $v0, $0 #result in $s0
                li   $v0, 10
                syscall
binary_search:  addi $sp, $sp, -4 #save space for $ra on stack
                sw   $ra, 4($sp)  #save $ra on stack
                slt  $t0,$a2,$a1  #if high < low, set $t0
                beq  $t0,$0,search#otherwise split and search
                addi $v0,$0,-1    #set return value to -1
                j    finish       #return
search:         add  $t0,$a1,$a2  #$t0 = low + high
                sra  $t0,$t0,1    #$t0 = (low + high)/2
                sll  $t1,$t0,2    #$t1 = offset to a[middle]
                add  $t1,$a0,$t1  #$t1 = address to a[middle]
                lw   $t1,0($t1)   #$t1 = a[middle]
                slt  $t2,$a3,$t1  #if target < a[middle]
                beq  $t2,$0,go_right #target >= a[middle]
go_left:        addi  $a2,$t0,-1  #third argument of binary_search becomes middle-1
                jal binary_search #call binary_search
                j   finish        #necessary to avoid all other paths
go_right:       slt  $t2,$t1,$a3  #if target > a[middle]
                beq  $t2,$0,found #target = a[middle]
                addi  $a1,$t0,1   #second argument of binary_search becomes middle+1
                jal binary_search
                j   finish        #necessary to avoid all other paths
found:          add  $v0,$t0,$0   #return middle
finish:         lw   $ra, 4($sp)  #restore $ra
                addi $sp,$sp,4    #restore $sp
                jr $ra
```

Ερώτημα 4 (16 βαθμοί): Assembly

Έστω κώδικας MIPS assembly στον οποίο έχουμε δηλώσει 4 διαδικασίες, `func0, func1, func2, func3` όπως φαίνεται παρακάτω. Υλοποιήστε σε MIPS assembly έναν πίνακα αλμάτων func_jump_table σε αυτές τις διαδικασίες και δείξτε τη χρήση του γράφοντας μία διαδικασία callfunc η οποία δέχεται σαν όρισμα έναν ακέραιο `i` και εκτελεί άλμα στη διαδικασία `funci` εάν το `i` έχει τιμή 0. . .3.

```
func_jump_table: #create a jump table for the four functions here
...
func0: ...
...
func1: ...
...
func2: ...
...
func3: ...
...
callfunc: #callfunc should take an argument i and call funci using the jump table
...
```

Δίνεται ο πλήρης κώδικας με μία διαδικασία main που καλεί την callfunc (για να δοκιμάσετε τον κώδικα και στον SPIM εάν θέλετε). Στις απαντήσεις δεν χρειάζεται η main.

```
                .data
                .align 2
call_func_table: .word func0,func1,func2,func3
                .text
                .globl main
main:           addi $a0,$0,2 #pick an i
                jal callfunc  #call callfunc
                li   $v0, 10
                syscall
callfunc:       addi $sp,$sp,-4          #need space on stack for $ra due to nested call
                sw   $ra,4($sp)          #save $ra
                add  $t0,$a0,$0
                addi $t1,$0,3
                slt  $t2,$a0,$0          #if arg < 0 return
                bne  $t2,$0, return
                slt  $t2,$t1,$a0         #if arg > 3 return
                bne  $t2,$0, return
                la   $t0,call_func_table #load starting address of jump table
                sll  $t1,$a0,2           #offset of function in jump table
                add  $t1,$t1,$t0         #address of function in jump table
                lw   $t1,0($t1)          #$t1 now has PC of funci
                la   $ra, return         #need to set return address register
                                         #since we can not call funci jusing jal!
                jr   $t1                 #call funci
return:         lw   $ra,4($sp)
                addi $sp,$sp,4
                jr   $ra
func0:          addi $v0,$0,0
                jr $ra
func1:          addi $v0,$0,1
                jr $ra
func2:          addi $v0,$0,2
                jr $ra
func3:          addi $v0,$0,3
                jr $ra
```

Ερώτημα 5 (20 βαθμοί): Κρυφές μνήμες

1. Έστω κρυφή μνήμη μονοσήμαντης απεικόνισης μεγέθους 8 λέξεων, με μέγεθος λέξης 4 bytes και μέγεθος block 1 λέξη. Έστω η ακολουθία προσπελάσεων του επεξεργαστή στη μνήμη που φαίνεται παρακάτω. Οι προσπελάσεις είναι αριθμημένες από 01 έως 26 και η διεύθυνση κάθε προσπέλασης δίνεται στο δεκαεξαδικό. Δείξτε (στην κόλλα σας, όχι πάνω στα θέματα) για κάθε προσπέλαση αν προκαλεί ευστοχία (E) ή αστοχία (A). **(6,5 βαθμοί)**

```
01: 0x10010000
02: 0x10010004
03: 0x10010000
04: 0x10010004
05: 0x10010004
06: 0x10010008
07: 0x10010008
08: 0x1001000c
09: 0x1001000c
10: 0x10010010
11: 0x1001000c
12: 0x10010010
13: 0x10010010
```

```
14: 0x10010014
15: 0x10010014
16: 0x10010018
17: 0x10010018
18: 0x1001001c
19: 0x1001001c
20: 0x10010020
21: 0x10010020
22: 0x10010024
23: 0x10010024
24: 0x10010028
25: 0x10010024
26: 0x10010028
```

A,A,E,E,E,A,E,A,E,A,E,E,E
A,E,A,E,A,E,A,E,A,E,A,E,E

2. Διαθέτουμε κρυφή μνήμη με το ίδιο μέγεθος όπως η παραπάνω (8 λέξεις), με μέγεθος λέξης 4 bytes και μέγεθος block 2 λέξεις. Δείξτε (στην κόλλα σας, όχι πάνω στα θέματα) για κάθε προσπέλαση αν προκαλεί ευστοχία (E) ή αστοχία (A). **(6,5 βαθμοί)**

A,E,E,E,E,A,E,E,E,A,E,E,E
E,E,A,E,E,E,A,E,E,E,A,E,E

3. Για κάθε ένα από τα ερωτήματα **3.1** και **3.2** υπολογίστε το μέσο χρόνο πρόσβασης στη μνήμη με τα παρακάτω δεδομένα: Συχνότητα επεξεργαστή 100 MHz. Χρόνος ευστοχίας 1 κύκλος. Χρόνος αστοχίας 4 κύκλοι για την πρώτη λέξη συν ένας κύκλος για κάθε επιπλέον λέξη. **(7 βαθμοί)**

Ο κύκλος ρολογιού έχει διάρκεια 10 ns. Στην πρώτη περίπτωση ο μέσος χρόνος πρόσβασης είναι

$$AMAT_1 = 1 + \text{miss\_ratio} \times 4 = 1 + (\frac{11}{26}) \times 4 = 2.69 = 26.9ns \quad (1)$$

Στη δεύτερη περίπτωση ο μέσος χρόνος πρόσβασης είναι

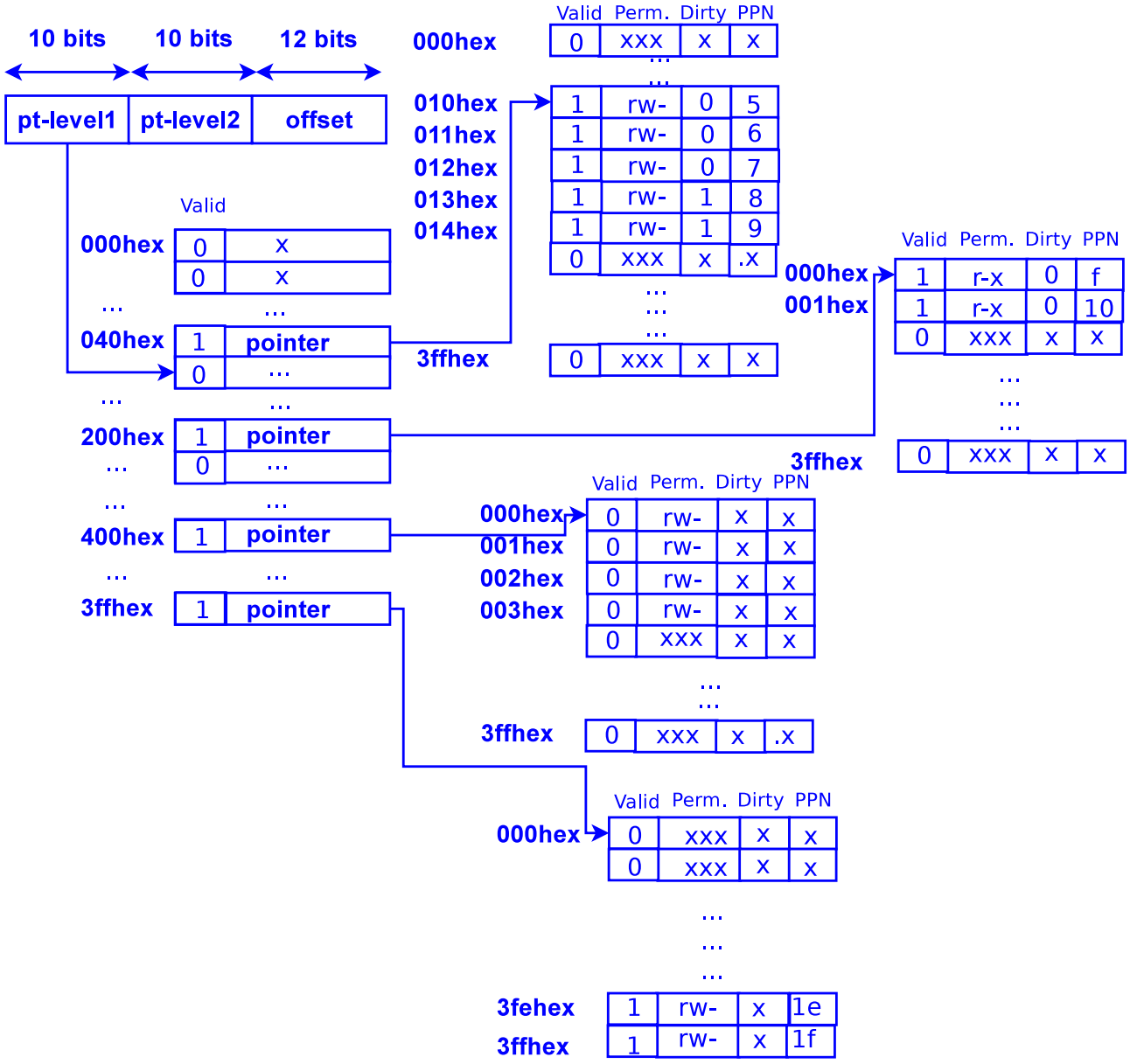
$$AMAT_2 = 1 + \text{miss\_ratio} \times 5 = 1 + (\frac{6}{26}) \times 5 = 2.15 = 21.5ns \quad (2)$$

Ερώτημα 6 (20 βαθμοί): Εικονική μνήμη

Έστω επεξεργαστής που υποστηρίζει εικονική μνήμη, με μέγεθος εικονικής διεύθυνσης 32 bits και μέγεθος σελίδας 4 Kbytes. Η οργάνωση του πίνακα σελίδων είναι σε 2 επίπεδα με ίσο αριθμό γραμμών στους πίνακες κάθε επιπέδου. Έστω πρόγραμμα με τα παρακάτω χαρακτηριστικά (όλες οι διευθύνσεις και αριθμοί σελίδων δίνονται στο δεκαεξαδικό):

- Οι εικονικές σελίδες που περιλαμβάνουν τις διευθύνσεις $0x10010000$ έως $0x10014fff$ περιέχουν δεδομένα με δικαιώματα προσπέλασης read-write, βρίσκονται στη φυσική μνήμη και απεικονίζονται σε συνεχόμενες σελίδες φυσικής μνήμης ξεκινώντας από τη σελίδα 5.
- Οι εικονικές σελίδες που περιλαμβάνουν τις διευθύνσεις από $0x10013000$ έως $0x10014fff$ περιέχουν τιμές που έχουν μεταβληθεί από τη στιγμή που έγινε η απεικόνιση των αντίστοιχων σελίδων στη μνήμη.
- Οι εικονικές σελίδες που περιλαμβάνουν τις διευθύνσεις από $0x40000000$ έως $0x40001200$ περιέχουν κώδικα με δικαιώματα προσπέλασης read-execute, βρίσκονται στη φυσική μνήμη και απεικονίζονται σε συνεχόμενες σελίδες φυσικής μνήμης ξεκινώντας από τη σελίδα f.
- Οι εικονικές σελίδες που περιλαμβάνουν τις διευθύνσεις από $0x80000000$ έως $0x80003e00$ περιλαμβάνουν δυναμικά δεδομένα με δικαιώματα προσπέλασης read-write και είναι απούσες από τη φυσική μνήμη.
- Οι εικονικές σελίδες που περιλαμβάνουν τις διευθύνσεις από $0xfffffe00$ έως $0xffffffff$ περιέχουν δεδομένα στοίβας με δικαιώματα προσπέλασης read-write, βρίσκονται στη φυσική μνήμη και απεικονίζονται σε συνεχόμενες σελίδες στη φυσική μνήμη με τελευταία από αυτές τη σελίδα $1f$.
- Όλες οι άλλες εικονικές σελίδες είναι απούσες από τη φυσική μνήμη.

Σχεδιάστε τους πίνακες απεικόνισης σελίδων στους οποίους να φαίνεται ξεκάθαρα πόσες γραμμές έχει κάθε πίνακας, ποιες γραμμές περιέχουν απεικονίσεις που χρησιμοποιεί το παραπάνω πρόγραμμα (δείξτε τον αριθμό κάθε τέτοιας γραμμής στο δεκαεξαδικό ή το δεκαδικό), πώς γίνεται η απεικόνιση της εικονικής σε φυσική διεύθυνση και τα δικαιώματα πρόσβασης σε κάθε σελίδα του προγράμματος. Επιτρέπεται η χρήση αποσιωπητικών


10 bits
10 bits
12 bits
pt-level1
pt-level2
offset
Valid
000hex
040hex
200hex
400hex
3ffhex
pointer
000hex
010hex
011hex
012hex
013hex
014hex
3ffhex
Valid Perm. Dirty PPN
000hex
001hex
3ffhex
Valid Perm. Dirty PPN
000hex
001hex
002hex
003hex
3ffhex
Valid Perm. Dirty PPN
000hex
3fehex
3ffhex