Σειρά Ασκήσεων 3: Προσπελάσεις Μνήμης στον MIPS

Προθεσμία έως Δευτέρα 1 Μαρτίου 2010, ώρα 23:59 (βδομάδα 4) (από βδομάδα 2.3)

[**Up:** Table of Contents] [printer version, in **PDF**]
[**Prev:** 2. Loops, SPIM I/O] [**Next:** 4. Instruction Formats]

3.1 Προσπελάσεις Μνήμης: Εντολές load και store

Ο MIPS, όπως και οι άλλοι επεξεργαστές τύπου RISC, δεν έχει εντολές που να κάνουν αριθμητικές πράξεις πάνω σε τελεστέους που βρίσκονται στη μνήμη --όλες οι αριθμητικές πράξεις του γίνονται πάνω σε καταχωρητές ή σταθερές ποσότητες (immediate constants). Ο μόνος τρόπος να επεξεργαστούμε τα περιεχόμενα της μνήμης είναι πρώτα να αντιγράψουμε μιά λέξη (32 bits), ή μία μισή λέξη (16 bits), ή ένα byte (8 bits) από τη μνήμη σ' ένα καταχωρητή της CPU, να την επεξεργαστούμε σε καταχωρητές, και τέλος να αντιγράψουμε το αποτέλεσμα από έναν καταχωρητή στη μνήμη. Οι λόγοι είναι (α) γιά απλότητα του hardware, και (β) γιατί δεν θα πετυχαίναμε ψηλότερη ταχύτητα αν μια μόνη εντολή έκανε και την αντιγραφή και την επεξεργασία, όπως διδάσκεται στο μάθημα HY-425 (Αρχιτεκτονική Υπολογιστών).

Αντιγραφή μιας 32-μπιτης λέξης από τη μνήμη σ' ένα καταχωρητή ("*φόρτωμα* στον καταχωρητή") γίνεται με την εντολή "**`lw $rd, imm($rx)`**" (**load** word), όπου `$rd` είναι ο καταχωρητής προορισμού (destination register), και `$rx` είναι ένας καταχωρητής (index register) που περιέχει μια διεύθυνση μνήμης στην οποία προστίθεται ο σταθερός αριθμός `imm` και το αποτέλεσμα της πρόσθεσης είναι η τελική διεύθυνση μνήμης απ' όπου γίνεται η αντιγραφή στον `$rd`. Συχνά, συμβολίζουμε τη μνήμη σαν έναν πίνακα (array) *M[ ]*, και γράφουμε *M[A]* γιά να συμβολίσουμε το περιεχόμενο της θέσης μνήμης με διεύθυνση *A*. Έτσι, η παραπάνω εντολή `lw $rd, imm($rx)` προκαλεί ανάγνωση από τη διεύθυνση μνήμης *(imm + $rx)*, δηλαδή διαβάζει το *M[imm + $rx]*, και το γράφει στον καταχωρητή *$rd*.

Αντίστροφα, αντιγραφή μιας 32-μπιτης λέξης από ένα καταχωρητή στη μνήμη ("*αποθήκευση* του καταχωρητή") γίνεται με την εντολή "**`sw $rs, imm($rx)`**" (**store** word), η οποία γράφει στη θέση μνήμης με διεύθυνση *(imm + $rx)*, δηλαδή προκαλεί την αντιγραφή *M[imm + $rx]* <-- *$rs*. Εδώ, ο `$rs` είναι καταχωρητής πηγής (source register)· προσέξτε ότι σε αυτή την περίπτωση, ο τελεστέος πηγής (source operand) γράφεται αριστερά και ο τελεστέος προορισμού δεξιά μέσα στην εντολή Assembly, αντίθετα δηλαδή από τις εντολές αριθμητικών πράξεων και από την εντολή load.

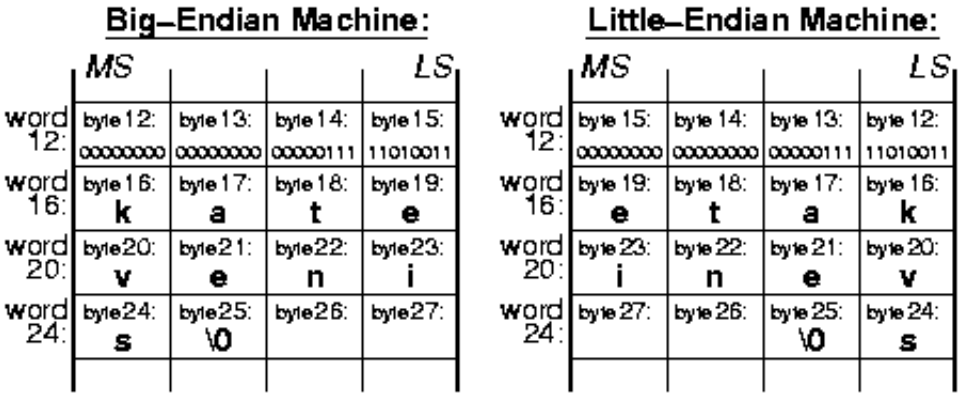
3.2 Διευθύνσεις Bytes και Περιορισμοί Ευθυγράμμισης:

Οι διευθύνσεις μνήμης στον MIPS, όπως και σχεδόν σε όλους τους μοντέρνους επεξεργαστές, αναφέρονται σε **Bytes** στη μνήμη, δηλαδή ο MIPS είναι "**Byte Addressable**". Έτσι, μια 32-μπιτη λέξη (π.χ. ένας ακέραιος) καταλαμβάνει 4 "θέσεις μνήμης" (4 bytes). Κατά συνέπεια, ένας πίνακας (array) μεγέθους 100 ακεραίων "πιάνει" 400 (συνεχόμενες) διευθύνσεις (θέσεις) στη μνήμη. Σ' ένα τέτοιο πίνακα, η διεύθυνση του κάθε ακέραιου διαφέρει από αυτήν του διπλανού του κατά 4. Εαν A_0 είναι η διεύθυνση του "πρώτου" (μηδενικού) στοιχείου, a[0], ενός πίνακα ακεραίων της C, τότε το στοιχείο a[i] του πίνακα αυτού θα βρίσκεται στη διεύθυνση (A_0 + 4*i). Αν ο πίνακας αυτός ήταν πίνακας χαρακτήρων (char), τότε το στοιχείο a[i] θα ήταν στη διεύθυνση (A_0 + i).

Τα 4 bytes που αποτελούν έναν ακέραιο έχουν διευθύνσεις που είναι συνεχόμενοι αριθμοί. Διεύθυνση του ακεραίου είναι η διεύθυνση εκείνου από τα 4 bytes του που έχει τη μικρότερη ("πρώτη") από τις 4 διευθύνσεις. Ο MIPS επιβάλει **περιορισμούς ευθυγράμμισης** (alignment restrictions) στις ποσότητες που προσπελαύνουν οι εντολές load και store στη μνήμη: μια ποσότητα μεγέθους *N* bytes επιβάλλεται να έχει διεύθυνση που να είναι ακέραιο πολλαπλάσιο του *N*. Έτσι, όταν το *N* είναι δύναμη του 2, η διεύθυνση κάθε τέτοιας ποσότητας τελειώνει σ' ένα αντίστοιχο πλήθος μηδενικών, και η διεύθυνση των υπολοίπων bytes της ποσότητας διαφέρει μόνο σε αυτά τα λιγότερο σημαντικά (least significant) bits. Λόγω αυτού του περιορισμού, όταν η φυσική μνήμη έχει πλάτος *N* bytes, αρκεί μία μόνο προσπέλαση σε αυτήν για κάθε πρόσβαση σε ποσότητα μεγέθους *N* bytes.

3.3 Αρίθμηση των Bytes: Μηχανές Big-Endian και Little-Endian

Όταν αποθηκεύεται στη μνήμη ενός υπολογιστή μια ποσότητα αποτελούμενη από πολλαπλά bytes (π.χ. ένας ακέραιος), πρέπει να καθοριστεί με ποια σειρά αριθμούνται (διευθυνσιοδοτούνται) τα επιμέρους bytes μέσα στην ποσότητα αυτή. Δυστυχώς, δεν έχει υπάρξει συμφωνία μεταξύ των μεγάλων εταιρειών κατασκευής επεξεργαστών για τη σειρά αυτή, με συνέπεια να υπάρχουν δύο διαφορετικοί τύποι επεξεργαστών σήμερα -- οι επονομαζόμενοι "big-endian" και οι επονομαζόμενοι "little-endian".



Ας ξεκινήσουμε με μια σύμβαση που αφορά τον τρόπο σχεδιασμού στο χαρτί των ποσοτήτων που αποτελούνται από πολλαπλά bytes: Μέσα σ' έναν ακέραιο αριθμό, τα bits εκείνα που πολλαπλασιάζονται επί τις μεγαλύτερες δυνάμεις του 2 για να μας δώσουν την αριθμητική τιμή του ακεραίου λέγονται "περισσότερο σημαντικά" (MS - most significant) bits, και αυτά που πολλαπλασιάζονται επί τις μικρότερες δυνάμεις του 2 λέγονται "λιγότερο σημαντικά" (LS - least significant) bits. Το byte που περιέχει τα MS bits λέγεται MS byte, και εκείνο που περιέχει τα LS bits λέγεται LS byte. Όποτε σχεδιάζουμε έναν ακέραιο στο χαρτί, οριζόντια, θα βάζουμε πάντα τα MS bits και byte αριστερά, και τα LS bits και byte δεξιά, δηλαδή όπως και στους δεκαδικούς αριθμούς (φυσικά, η σύμβαση αυτή αφορά μόνο τους ανθρώπους --μέσα στον υπολογιστή δεν έχει νόημα να μιλάμε για "αριστερά transistors" και "δεξιά transistors"...). Ακολουθώντας τη σύμβαση αυτή, το σχήμα δείχνει ένα παράδειγμα τεσσάρων (4) λέξεων μνήμης (16 bytes) ενός 32-μπιτου υπολογιστή σε μία μηχανή "big-endian" και σε μία μηχανή "little-endian". Η πρώτη λέξη περιέχει τον ακέραιο αριθμό 2003 (δεκαδικό) = 7D3 (δεκαεξαδικό), ενώ στις επόμενες 3 λέξεις υπάρχει ένας πίνακας χαρακτήρων (array of char) μεγέθους 10 στοιχείων, και περισσεύουν και δύο ελεύθερα bytes· ο πίνακας χαρακτήρων περιέχει το (null-terminated) string "katevenis" (κάθε byte θα περιέχει το δυαδικό κώδικα ASCII ενός χαρακτήρα --π.χ. το πρώτο byte θα περιέχει 01101011, που είναι ο κώδικας του 'k'-- αλλά εμείς, για ευκολία, δείχνουμε το συμβολιζόμενο χαρακτήρα).

- **Big-Endian**: Σε πολλούς υπολογιστές, το MS byte του κάθε ακεραίου έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των bytes του αυξάνουν (προχωρούν) καθώς προχωράμε "δεξιά", προς το LS byte του. Αυτοί οι υπολογιστές λέγονται "big-endian" διότι η αρίθμηση των bytes ξεκινά από το "big end", δηλαδή το MS byte.
- **Little-Endian**: Σε άλλους υπολογιστές, το LS byte του κάθε ακεραίου έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των bytes του αυξάνουν (προχωρούν) καθώς προχωράμε "αριστερά", προς το MS byte του. Αυτοί οι υπολογιστές λέγονται "little-endian" διότι η αρίθμηση των bytes ξεκινά από το "little end", δηλαδή το LS byte.

Παρατηρήστε ότι η διεύθυνση μιας λέξης (π.χ. του ακεραίου 2003 στη θέση 12) είναι η ίδια και στις δύο μηχανές, αφού, όπως είπαμε παραπάνω, είναι πάντα η διεύθυνση εκείνου από τα 4 bytes του που έχει τη μικρότερη ("πρώτη") από τις 4 διευθύνσεις. Επίσης παρατηρήστε ότι οι χαρακτήρες ενός string αποθηκεύονται σε διαδοχικά bytes της μνήμης κατά αύξουσες διευθύνσεις, όπως ακριβώς επιβάλει ο απλός κανόνας που είπαμε και παραπάνω. Το σχήμα αντιστοιχεί στη δήλωση (σε C) "`char buf[10];`", όπου ο πίνακας `buf[]` έχει τοποθετηθεί (π.χ. από τον compiler) στις θέσεις μνήμης με διεύθυνση 16 έως και 25· τότε, το στοιχείο *i* του πίνακα, `buf[i]`, βρίσκεται στη διεύθυνση *16+i*, επειδή 16 είναι η διεύθυνση εκκίνησης του πίνακα (η διεύθυνση του πρώτου του στοιχείου, `buf[0]`), και το μέγεθος του κάθε στοιχείου του πίνακα είναι 1 (byte). Έτσι, ο χαρακτήρας 'k' βρίσκεται στη θέση `buf[0]` δηλαδή στη διεύθυνση 16, ο χαρακτήρας 'a' βρίσκεται στη θέση `buf[1]` δηλαδή στη διεύθυνση 17, κ.ο.κ.

Το "endian-ness" του υπολογιστή, δηλαδή το αν είναι big-endian ή little-endian, δεν μας επηρεάζει όταν εργαζόμαστε σε ένα και μόνο μηχάνημα, και πάντα γράφουμε και διαβάζουμε την κάθε ποσότητα με τον ίδιο τύπο --πράγμα που είναι και το σωστό να κάνει κανείς-- δηλαδή όπου στη μνήμη γράφουμε string διαβάζουμε πάντα string, και όπου γράφουμε integer διαβάζουμε πάντα integer. Το "endian-ness" μας επηρεάζει όταν αλλάζουμε τύπο μεταξύ εγγραφής και ανάγνωσης --πράγμα ανορθόδοξο-- π.χ. γράφουμε κάπου ένα string και μετά το διαβάζουμε σαν integer, ή γράφουμε integer και διαβάζουμε string. Το σημαντικότερο όλων όμως είναι ότι το endian-ness του υπολογιστή πρέπει να λαμβάνεται υπ' όψη όταν

μεταφέρονται δεδομένα μέσω δικτύου μεταξύ υπολογιστών. Συνήθως, τα προγράμματα μεταφοράς δεδομένων (π.χ. ftp) θεωρούν ότι μεταφέρουμε κείμενο (ASCII strings), και τοποθετούν τα bytes με την αντίστοιχη σειρά. Αν όμως μεταφέρουμε άλλες μορφές δεδομένων (π.χ. 32-μπιτους ακεραίους) μεταξύ υπολογιστών με διαφορετικό endian-ness, η σειρά αυτή θα ήταν λάθος: όπως οι χαρακτήρες k, a, t, e μεταφέρονται σαν e, t, a, k στο παραπάνω σχήμα από big-endian σε little-endian, έτσι και ο ακέραιος 2003 θα ερμηνεύονταν σαν 00, 00, 07, D3 (δεκαεξαδικό), και θα μεταφέρονταν σαν D3, 07, 00, 00, δηλαδή 11010011.00000111.00000000.00000000 (δυαδικό), που είναι ο αριθμός -754,515,968 (δεκαδικό, συμπλήρωμα ως προς 2). Για να γίνει σωστά η μεταφορά, πρέπει να δηλωθεί στο πρόγραμμα μεταφοράς ο τύπος των δεδομένων που μεταφέρονται (π.χ. εντολή "type" στο ftp).

§ 3.4 - Άσκηση 3.1: Πίνακες Ακεραίων και Αντιγραφές στη Μνήμη

Γράψτε ένα πρόγραμμα σε Assembly του MIPS που να διαβάζει 8 ακεραίους από την κονσόλα και να τους αποθηκεύει σ' ένα πίνακα (array) "a[ ]" στη μνήμη. Στο ίδιο πρόγραμμα, δεσμεύστε χώρο στη μνήμη για ένα δεύτερο πίνακα "c[ ]" 8 ακεραίων. Ζητείται να υλοποιήσετε μία αντιμετάθεση των στοιχείων του πίνακα "a[ ]" που διαβάσατε από την κονσόλα και να αποθηκεύσετε το αποτέλεσμα που προκύπτει από την αντιμετάθεση στον πίνακα "c[ ]" ως εξής: Τα πρώτα μισά στοιχεία του πίνακα "c[ ]" θα πρέπει να περιέχουν τα στοιχεία του "a[ ]" με ζυγούς δείκτες και σε αύξουσα σειρά δεικτών (δηλαδή τα "a[0],a[2],...") και τα άλλα μισά στοιχεία του πίνακα "c[ ]" θα πρέπει να περιέχουν τα στοιχεία του "a[ ]" με μονούς δείκτες και σε αύξουσα σειρά δεικτών (δηλαδή τα "a[1],a[3],..."). Σας ζητείται να **χρησιμοποιήσετε το ελάχιστο πλήθος εντολών προσπέλασης ακεραίων στη μνήμη στην υλοποίησή σας**. Η προσπέλαση στη μνήμη κοστίζει αρκετά σε χρόνο στους σύγχρονους επεξεργαστές και σκοπός μας είναι να ελαχιστοποιήσουμε τις καθυστερήσεις που επιφέρει. Στο τέλος της εκτέλεσης του κώδικά σας, θα πρέπει να τυπώσετε τα περιεχόμενα τόσο του πίνακα "a[ ]" όσο και του πίνακα "c[ ]". Παραδώστε τον κώδικά σας κι ένα στιγμιότυπο από μια επιτυχημένη εκτέλεσή του, όπως αναφέρεται στο τέλος.

- Ξεκινήστε με όσα μάθατε στις ασκήσεις **2**, αλλά εδώ θα χρειαστεί να χρησιμοποιήσετε και τις νέες εντολές προσπέλασης ακεραίων στη μνήμη "lw" και "sw".
- Χρησιμοποιήστε τις οδηγίες ".data", ".align", και ".space" του Assembler του SPIM (σελίδες A-47, A-48 του παραρτήματος του βιβλίου) για να κρατήστε χώρο στη μνήμη δεδομένων (data segment) για τους πίνακες "a[ ], c[ ]", μεγέθους 8 ακεραίων έκαστος, και σωστά ευθυγραμμισμένο για ακεραίους. Τοποθετήστε κατάλληλα τα labels "a,c" ώστε να μπορείτε πιο κάτω να αναφερθείτε στη διεύθυνση όπου αρχίζει ο χώρος που κρατήσατε.
- Στην αρχή του προγράμματός σας, τοποθετήστε τις διευθύνσεις όπου αρχίζουν οι πίνακας a[ ], c[ ] σε δύο καταχωρητές. Τις διευθύνσεις αυτές τις ξέρει ο Assembler, αλλά εσείς πιθανότατα όχι (εκτός αν την είχατε δώσει σαν argument στην οδηγία .data). Επίσης, δεν ξέρετε αν η διεύθυνση αυτή χωρά σε 16 bits (ένα immediate) ή όχι. Σε όλα αυτά έρχεται να σας βοηθήσει η **ψεύδο**εντολή (pseudoinstruction) "`la $rd, label`" του Assembler του SPIM: αυτή λέει στον Assembler να γεννήσει μια ή δύο πραγματικές εντολές που τοποθετούν την πραγματική διεύθυνση του `label` στον καταχωρητή `$rd` (μία εντολή αν η διεύθυνση χωρά σε 16 bits, δύο εντολές αλλιώς). Παράδειγμα χρήσης της ψευδοεντολής "la" θα βρείτε στην § **2.2**, εκεί που ετοιμάζαμε τα ορίσματα για τις εκτυπώσεις strings.
- Στη συνέχεια, τυπώστε ένα prompt που να ζητά 8 ακεραίους σε 8 γραμμές.
- Μετά, μπείτε σ' ένα βρόχο που θα επαναληφθεί 8 φορές, και που κάθε φορά θα διαβάζει έναν αριθμό (μέσω κλήσης συστήματος) και θα τον αποθηκεύει στην επόμενη θέση του πίνακα a[ ]. Ακολούθως, γράψτε τον κώδικα για την αντιμετάθεση του πίνακα a[ ] και την αποθήκευση του αποτελέσματος στον πίνακα c[ ] χρησιμοποιώντας βρόχο (ή βρόχους) και εντολές προσπέλασης στη μνήμη. Επισημάνσεις: *(i)* Οι εντολές "beq" και "bne" γιά τη δημιουργία του βρόχου δέχονται μόνο καταχωρητές σαν τελεστέους, και όχι σταθερές ποσότητες (immediates). *(ii)* Η μοναδική διευθυνσιοδότηση (addressing mode) του MIPS είναι "σταθερή ποσότητα (immediate) + καταχωρητής" --μην χρησιμοποιήστε τις **ψεύδο**διευθυνσιοδοτήσεις του SPIM (σελίδα A.45).
- Αφού βγείτε από τον προηγούμενο βρόχο, τυπώστε μια διαχωριστική γραμμή, και γράφτε δύο ακόμη βρόχους, που θα επαναληφθούν και αυτοί 8 φορές, που θα τυπώνουν τα στοιχεία των πινάκων a[ ] και c[ ] αντίστοιχα. Ο κώδικας σε κάθε έναν από αυτούς τους βρόχους πρέπει να διαβάζει από τη μνήμη και να τυπώνει κάθε στοιχείο του αντίστοιχου πίνακα στην ίδια γραμμή αλλά όχι "κολλητά" με το προηγούμενό του.
- Τέλος, ξαναγυρίστε στην αρχή (όπως και στην άσκηση 2), ώστε η ίδια δουλειά να επαναλαμβάνεται επ' αόριστο.

§ 3.5 - Άσκηση 3.2: Υπολογιστές Big-Endian και Little-Endian

i. Χρησιμοποιήστε τον SPIM γιά να βρείτε τον δυαδικό (δεκαεξαδικό) κώδικα εσωτερικής αναπαράστασης (κώδικα ASCII) των χαρακτήρων του Λατινικού αλφαβήτου, a, b, ..., z, A, ..., Z, και των αριθμητικών χαρακτήρων, 0, 1, ..., 9. Για να το πετύχετε, ορίστε σταθερές τύπου string όπως στην πρώτη σειρά ασκήσεων, και στη συνέχεια μπείτε στον SPIM και μελετήστε τα περιεχόμενα της μνήμης δεδομένων στο παράθυρο "Data Segments". Δώστε τις διαπιστώσεις σας σ' ένα μικρό κειμενάκι, με 3 στήλες: χαρακτήρας, κώδικας ASCII στο δεκαεξαδικό, κώδικας ASCII στο δυαδικό. Βάλτε αποσιωπητικά και εξηγήστε εκεί που παρατηρείτε κάποια ομοιομορφία....

ii. Έστω ότι αποθηκεύουμε το null-terminated string "dsn" σε μια λέξη ενός 32-μπιτου υπολογιστή, και στη συνέχεια διαβάζουμε αυτή τη λέξη σαν να είναι (32-μπιτος) ακέραιος. Υπολογίστε με αριθμητικές πράξεις ποιόν ακέραιο θα διαβάσουμε (α) σε μια μηχανή big-endian, και (β) σε μια μηχανή little-endian. Δώστε την απάντησή σας, μαζί με τις αριθμητικές πράξεις που κάνατε, σ' ένα μικρό κειμενάκι (στο δεκαδικό).

iii. Επαληθεύστε την απάντησή σας μ' ένα μικρό πρόγραμμα στον SPIM: Ζητήστε από τον Assembler να βάλει το string στη μνήμη δεδομένων, και μέσα από το πρόγραμμά σας αντιγράψτε εκείνη τη λέξη (ολόκληρη!) σ' ένα καταχωρητή και ζητήστε να τυπωθεί (με μία κλήση συστήματος) σαν ακέραιος. Ο SPIM συμπεριφέρεται σαν big-endian ή σαν little-endian ανάλογα σε ποιόν υπολογιστή τρέχει! Σε τι υπολογιστή τρέξατε τον SPIM; Τι έβγαλε; Αν τον τρέξτε σε άλλου τύπου υπολογιστή, με επεξεργαστή άλλης εταιρείας, θα βγάλει άλλα αποτελέσματα; Παραδώστε το πρόγραμμά σας (πηγαίος κώδικας), τα συμπεράσματά σας σ' ένα μικρό κειμενάκι, κι ένα στιγμιότυπο (screen-dump) του τρεξίματος.

Τρόπος Παράδοσης:

Θα παραδώσετε ηλεκτρονικά τα εξής:

1. τον πηγαίο κώδικά σας της άσκησης 3.1, "ask3.1.s"
2. τον πηγαίο κώδικά σας της άσκησης 3.2(iii), "ask3.2.s"
3. ένα στιγμιότυπο (screen-dump) του τρεξίματος της άσκησης 3.1, "ask3.1.jpg"
4. ένα στιγμιότυπο (screen-dump) του τρεξίματος της άσκησης 3.2(iii), "ask3.2.jpg"
5. ένα κείμενο με τις τρεις απαντήσεις σας στα 3 μέρη της άσκησης 3.2, "ask3.2.txt" (ή "ask3.2.pdf", αν και είναι προτιμότερη η μορφή ".txt").

Για να παραδώσετε την άσκηση συνδεθείτε σε ένα μηχάνημα Linux του Τμήματος με το login και το password που χρησιμοποιείτε στο Τμήμα και μέσω της ασφαλούς διαδικασίας σύνδεσης με ssh/tunnel. Δημιουργήστε ένα directory με τα 5 αρχεία που σας ζητάει η άσκηση (ask3.1.s, ask3.2.s, ask3.1.jpg, ask3.2.jpg, ask3.2.txt). Ας υποθέσουμε ότι το όνομα του directory είναι /somepath/mydir. Μετακινηθείτε στο directory somepath και εκτελέστε την εντολή:

```
/usr/local/bin/submit exercise3@hy225 mydir
```

Η διαδικασία submit θα σας ζητήσει να επιβεβαιώσετε την αποστολή των αρχείων. Περισσότερες πληροφορίες και αναλυτικές οδηγίες για τη νέα διαδικασία submit είναι διαθέσιμες εκτελώντας:

```
man submit
```

σε κάποιο από τα μηχανήματα Linux του Τμήματος. Θα εξεταστείτε **και προφορικά** για την Άσκηση 3, από βοηθούς του μαθήματος. **Η προφορική εξέταση είναι υποχρεωτική και κατά τη διάρκειά της θα πρέπει να επιδείξετε εάν ο κώδικας που παραδώσατε εκτελείται σωστά, να απαντήσετε σε ερωτήσεις των βοηθών σχετικές με τον κώδικα που παραδώσατε και να τεκμηριώσετε πλήρως τις απαντήσεις σας. Εάν παραδώσετε σωστό κώδικα χωρίς να δώσετε τεκμηριωμένες απαντήσεις σχετικές με αυτόν η άσκησή σας θα βαθμολογείται με 0. Είναι προτιμότερο να μας παραδώσετε λάθος κώδικα τον οποίο γράψατε εσείς οι ίδιοι και στον οποίο καταλαβαίνετε τι κάνατε (βαθμός > 0), παρά σωστό κώδικα που "δανειστήκατε" από άλλον χωρίς να καταλαβαίνετε τι κάνει (βαθμός = 0).**

Για την προφορική εξέταση θα κλείσετε ραντεβού με έναν βοηθό χρησιμοποιώντας το web-based εργαλείο **Submit-Rendezvous**. Αφού κάνετε login με το όνομα και τον κωδικό σας, επιλέγετε το tab "Rendezvous", κατόπιν "HY225 - Examination of exercise 3" και τέλος "Continue". Θα δείτε μία λίστα με διαστήματα λίγων λεπτών που είναι διαθέσιμα για προφορικές συνεντεύξεις με βοηθούς. Επιλέξτε ένα διάστημα και πατήστε "Book". Οι βοηθοί που θα εξετάσουν την άσκηση είναι οι Μιχάλης Αλβανός (alvanos) και Γιώργος Τσάμης (gtsamis). Η προφορική εξέταση θα γίνει στα γραφεία των μεταπτυχιακών στο υπόγειο των λευκών κτιρίων. **Προσοχή: το web-based εργαλείο θα το χρησιμοποιήσετε μόνο για να κλείσετε ραντεβού και όχι για να υποβάλετε την άσκηση. Για την υποβολή της άσκησης θα χρησιμοποιήσετε το command-line εργαλείο submit και τη**

διαδικασία που περιγράψαμε παραπάνω.

[**Up:** Table of Contents]
[**Prev:** 2. Loops, SPIM I/O]
[printer version, in **PDF**]
[**Next:** 4. Instruction Formats]

Up to the Home Page of CS-225