HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2017

Τμ. Επ. Υπολογιστών

Σειρά Ασκήσεων 3: Προσπελάσεις Μνήμης στον MIPS

Προθεσμία έως Παρασκευή 24 Φεβρουαρίου 2017, ώρα 23:59 (βδομάδα 3.3) (από βδ. 2.1)

Βιβλίο: Διαβάστε τη συνέχεια της §2.3: σελίδες 121-127.

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*. Ο σταθερός αριθμός `imm` χρησιμοποιείται σαν *προσημασμένος* από το υλικό του MIPS, επομένως η "κίνηση" που αυτός επιβάλει σε σχέση με το πού "δείχνει" ο καταχωρητής `$rx` μπορεί να είναι προς τα "εμπρός" ή προς τα "πίσω".

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.

Οι συνήθεις τρόποι χρήσης του παραπάνω τρόπου "διευθυνσιοδότησης" (addressing mode), δηλαδή "καταχωρητής + σταθερά", θα γίνουν κατανοητοί καθώς το μάθημα θα προχωρά. Ας σημειώσουμε όμως εδώ, γιά μελλοντική αναφορά, ότι η πιό συνηθισμένη χρήση είναι με τον καταχωρητή να περιέχει έναν pointer, και τη σταθερή ποσότητα να είναι μιά "απόκλιση" (offset) από εκεί που δείχνει ο pointer. Αυτό χρησιμοποιείται εξαιρετικά συχνά (α) όταν ο pointer δείχνει σε μιά δομή δεδομένων και η απόκλιση ορίζει ποιό από τα πεδία αυτής της δομής θέλουμε να προσπελάσουμε (π.χ. p->next), (β) όταν ο καταχωρητής είναι ο stack pointer και η απόκλιση ορίζει ποιά από τις τοπικές μεταβλητές της διαδικασίας ζητάμε, και (γ) όταν ο καταχωρητής είναι ο global pointer και η απόκλιση ορίζει μία από τις καθολικές βαθμωτές μεταβλητές. Μιά άλλη χρήση του τρόπου διευθυνσιοδότησης "σταθερά + καταχωρητής" είναι με την σταθερά = διεύθυνση βάσης ενός (στατικά allocated) πίνακα (array) και καταχωρητής = index του στοιχείου του πίνακα πολλαπλασιασμένο (ήδη) επί το μέγεθος του στοιχείου του πίνακα, οπότε προσπελαύνουμε το στοιχείο με εκείνο το index, όπως θα πούμε στην αμέσως επόμενη ενότητα. Αυτή η χρήση είναι σπανιότερη, διότι (α) σπανίζουν οι πίνακες που να ξεκινούν στα πρώτα 32 KBytes του χώρου διευθύνσεων (ώστε η διεύθυνση βάσης τους να χωρά στα 16 bits που έχει η *προσημασμένη* σταθερά), και (β) συχνά οι compilers αλλάζουν την αριθμητική με array indexes σε αριθμητική με array element pointers μέσα σε βρόχους που επεξεργάζονται στοιχεία πινάκων.

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) του ενός Byte καθένας, 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. (Παρατηρήστε ότι π.χ. 64-μπιτες ποσότητες (8 Bytes) αποθηκευμένες σε 32-μπιτη μνήμη (πλάτος μνήμης = 4 Bytes) απαιτούν πάντα 2 προσπελάσεις μνήμης γιά να τις διαβάσουμε ή γράψουμε, είτε αυτές είναι ευθυγραμμισμένες σε διευθύνσεις πολλαπλάσια του 8 είτε σε πολλαπλάσια του 4. Γιατί λοιπόν ο MIPS απαιτεί τέτοιες ποσότητες να έχουν ευθυγράμμιση πολλαπλασίου του 8; Απάντηση: διότι όταν θα βγεί ένα μελλοντικό μοντέλο υπολογιστή με 64-μπιτη μνήμη, τότε με την μεν ευθυγράμμιση πολλαπλασίων του 8 θα έχουμε εξασφαλισμένη τη μία μόνο προσπέλαση πάντα, ενώ με ευθυγράμμιση πολλαπλασίων του 4 αυτό δεν εξασφαλίζεται).

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 κατασκευαστών επεξεργαστών για τη σειρά αυτή, με συνέπεια να υπάρχουν δύο διαφορετικοί τύποι επεξεργαστών σήμερα --οι επονομαζόμενοι "big-endian" και οι επονομαζόμενοι "little-endian".

Big–Endian Machine:

	MS			*LS*
word 12:	byte 12: 00000000	byte 13: 00000000	byte 14: 00000111	byte 15: 11010011
word 16:	byte 16: **k**	byte 17: **a**	byte 18: **t**	byte 19: **e**
word 20:	byte 20: **v**	byte 21: **e**	byte 22: **n**	byte 23: **i**
word 24:	byte 24: **s**	byte 25: **\0**	byte 26:	byte 27:

Little–Endian Machine:

	MS			*LS*
word 12:	byte 15: 00000000	byte 14: 00000000	byte 13: 00000111	byte 12: 11010011
word 16:	byte 19: **e**	byte 18: **t**	byte 17: **a**	byte 16: **k**
word 20:	byte 23: **i**	byte 22: **n**	byte 21: **e**	byte 20: **v**
word 24:	byte 27:	byte 26:	byte 25: **\0**	byte 24: **s**

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. (Η πιό "φημισμένη" αρχιτεκτονική Little-Endian είναι η οικογένεια **x86** της Intel. Σύμφωνα με μιά πρόχειρη συλλογή πληροφοριών (2/2015), η (πιθανόν μόνη) άλλη αρχιτεκτονική Little-Endian ήταν η VAX της DEC, ενώ (μάλλον) όλες οι υπόλοιπες αρχιτεκτονικές είναι Big-Endian. Επίσης, κάμποσες αρχιτεκτονικές είναι configurable να δουλεύουν με όποιο endianness επιλέγει ο χρήστης, χωριστά γιά τα data segments και code segments. Τη δυνατότητα αυτή γιά configurability έχουν (μάλλον) τουλάχιστον οι: ARM versions 3 και νεότερες, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC, SuperH SH-4, και IA-64).

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: Πίνακας Ακεραίων

Γράψτε ένα πρόγραμμα σε Assembly του MIPS που να διαβάζει 8 ακεραίους από την κονσόλα, να τους αποθηκεύει σ' ένα πίνακα (array) στη μνήμη, και στη συνέχεια να τυπώνει τα **εξαπλάσιά** τους και με την **αντίστροφη** σειρά. 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 QtSpim (σελίδες A-

47, A-48 του παραρτήματος του βιβλίου) για να κρατήστε χώρο στη μνήμη δεδομένων (data segment) γιά τον πίνακα "a[ ]", μεγέθους 8 ακεραίων, και σωστά ευθυγραμμισμένο για ακεραίους. Τοποθετήστε κατάλληλα το label "a" ώστε να μπορείτε πιο κάτω να αναφερθείτε στη διεύθυνση όπου αρχίζει ο χώρος που κρατήσατε.

- Στην αρχή του προγράμματός σας, τοποθετήστε τη διεύθυνση όπου αρχίζει ο πίνακας a[ ] σε έναν καταχωρητή. Τη διεύθυνση αυτή την 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 $rd, label`" του Assembler του QtSpim: αυτή λέει στον Assembler να γεννήσει μια ή δύο πραγματικές εντολές που τοποθετούν την πραγματική διεύθυνση του `label` στον καταχωρητή `$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 καλέσματος συστήματος) και θα τον αποθηκεύει στην επόμενη θέση του a[ ]. Επισημάνσεις: *(i)* Οι εντολές beq, bne γιά τη δημιουργία 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 QtSpim (σελίδα A.45 παραρτήματος).
- Αφού βγείτε από τον προηγούμενο βρόχο, τυπώστε μια διαχωριστική γραμμή, και μπείτε σ'έναν άλλο βρόχο, που θα επαναληφθεί και αυτός 8 φορές, και που θα επισκεφθεί τα στοιχεία του πίνακα "a[ ]" αλλά κατ' **αντίστροφη** σειρά. Για κάθε στοιχείο, θα το διαβάζει από τη μνήμη, θα το εξαπλασιάζει (χωρίς να πειράξει την τιμή στη μνήμη), και θα τυπώνει αυτό το εξαπλάσιο, στην ίδια γραμμή αλλά όχι "κολλητά" με το προηγούμενό του. Υπολογίστε το εξαπλάσιο μέσω τριών κατάλληλων εντολών πρόσθεσης (add).
- 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 QtSpim 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 παράγραφο 2.2, και στη συνέχεια μπείτε στον QtSpim και μελετήστε τα περιεχόμενα της μνήμης δεδομένων στην καρτέλα "Data". 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 "xyz" 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 QtSpim: 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 (μ' ένα κάλεσμα συστήματος) σαν ακέραιος. Ο QtSpim συμπεριφέρεται σαν big-endian ή σαν little-endian ανάλογα σε ποιόν υπολογιστή τρέχει! Σε τι υπολογιστή τον τρέξατε τον QtSpim; Τι έβγαλε; Αν τον τρέξτε σε άλλο τύπο υπολογιστή, με επεξεργαστή άλλης εταιρείας, θα βγάλει άλλα; Παραδώστε το πρόγραμμά σας (πηγαίος κώδικας), τα συμπεράσματά σας σ' ένα μικρό κειμενάκι, κι ένα στιγμιότυπο (screen-dump) του τρεξίματος.

Τρόπος Παράδοσης: Παραδώστε μέσω **`turnin ex03@hy225 [directoryName]`** τα εξής:
(1) τον πηγαίο κώδικά σας της άσκησης 3.1, "ex03_1.s"
(2) τον πηγαίο κώδικά σας της άσκησης 3.2(iii), "ex03_2.s"
(3) ένα στιγμιότυπο (screen-dump) του τρεξίματος της άσκησης 3.1, "ex03_1.jpg"
(4) ένα στιγμιότυπο (screen-dump) του τρεξίματος της άσκησης 3.2(iii), "ex03_2.jpg"
(5) ένα κείμενο με τις τρείς απαντήσεις σας στα 3 μέρη της άσκησης 3.2, "ex03_2.pdf" (σε αυτό το μάθημα παρακαλείσθε να δίνετε τις απαντήσεις σας κειμένου σε μορφή **PDF**).
Θα εξεταστείτε ***και προφορικά*** για την Άσκηση 3, από βοηθούς του μαθήματος, με διαδικασία γιά την οποία θα ενημερωθείτε μέσω ηλτά (email) στη λίστα του μαθήματος.