HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2020

Τμ. Επ. Υπολογιστών

Σειρά Ασκήσεων 3: Προσπελάσεις Μνήμης στον RISC-V

Προθεσμία έως Παρασκευή 21 Φεβ. 2020, ώρα 23:59 (βδ. 3.3) (από βδ. 1.3)

Βιβλίο: Υπόλοιπη §2.3: pp. 68-73 (Αγγλικό), ή σελίδες 121-127 (Ελληνικό).

3.1 Προσπελάσεις Μνήμης: Εντολές load και store

Ο RISC-V, 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 μία διπλή λέξη (double - 64 bits), ή μία λέξη (word - 32 bits), ή μία μισή λέξη (half - 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 συνήθως δεν πετυχαίνουμε ψηλότερη ταχύτητα, ακόμα και όταν μια μόνη εντολή κάνει και την αντιγραφή και την επεξεργασία, διότι, όπως θα δούμε, όταν χρησιμοποιείται ομοχειρία (pipelining), συνήθως ο περιοριστικός παράγοντας (bottleneck) είναι αλλού.

Αντιγραφή μιας 32-μπιτης λέξης από τη μνήμη σ' ένα καταχωρητή ("*φόρτωμα* στον καταχωρητή") γίνεται με την εντολή "**`lw rd, offset(rs1)`**" (**load** word), όπου `rd` είναι ο καταχωρητής προορισμού (destination register), και `rs1` είναι ένας καταχωρητής πηγής (source/index/base register) που περιέχει μια διεύθυνση μνήμης (pointer) στην οποία προστίθεται ο σταθερός αριθμός `offset` (απόσταση/απόκλιση), και το αποτέλεσμα της πρόσθεσης είναι η τελική διεύθυνση μνήμης απ' όπου γίνεται η ανάγνωση και 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, offset(rs1)` προκαλεί ανάγνωση από τη διεύθυνση μνήμης *(offset + rs1)*, δηλαδή διαβάζει το *M[offset + rs1]*, και το γράφει στον καταχωρητή *rd*. Ο σταθερός αριθμός `offset` χρησιμοποιείται σαν *προσημασμένος* από το υλικό του RISC-V, επομένως η "κίνηση" που αυτός επιβάλει σε σχέση με το πού "δείχνει" ο καταχωρητής `rs1` μπορεί να είναι προς τα "εμπρός" ή προς τα "πίσω".

Αντίστροφα, αντιγραφή μιας 32-μπιτης λέξης από έναν καταχωρητή στη μνήμη ("*αποθήκευση* του καταχωρητή") γίνεται με την εντολή "**`sw rs2, offset(rs1)`**" (**store** word), η οποία γράφει στη θέση μνήμης με διεύθυνση *(offset + rs1)*, δηλαδή προκαλεί την αντιγραφή *M[offset + rs1]* ← *rs2*. Εδώ, ο `rs2` 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.

Υπάρχουν και οι παρόμοιες εντολές "**`lb`**" (load byte) και "**`lh`**" (load half) που διαβάζουν αντίστοιχα 1 ή 2 Bytes από τη μνήμη, τα μετατρέπουν σε 32 bits θεωρώντας τα προσημασμένα (signed), και τα γράφουν στον καταχωρητή *rd*. Επίσης οι εντολές "**`lbu`**" (load byte unsigned) και "**`lhu`**" (load half unsigned) κάνουν

την ίδια δουλειά εκτός ότι στη μετατροπή σε 32 bits θεωρούν τα 8 ή 16 bits που διάβασαν από τη μνήμη σαν μη προσημασμένα. Αντίστοιχα οι εντολές **`sb`**" (store byte) και **`sh`**" (store half) γράφουν στη μνήμη τα 8 ή 16 δεξιά (λιγότερο σημαντικά - least significant) bits του καταχωρητή *rs2*.

Στον 64-μπιτο RISC-V (όπως στο Αγγλικό βιβλίο, αλλά όχι όπως στον RARS), οι καταχωρητές είναι 64-μπιτοι, οπότε η **`lw`** διαβάζει πάλι 32 bits από τη μνήμη, όπως και πριν, αλλά τα μετατρέπει σε 64 bits θεωρώντας τα προσημασμένα (signed) γιά να τα γράψει στο 64-μπιτο καταχωρητή *rd*, ενώ η **`lwu`** (load word unsigned) κάνει το ίδιο αλλά στη μετατροπή θεωρεί τα 32 bits που διάβασε σαν unsigned. Αντίστοιχα, η εντολή **`sw`** γράφει τα 32 δεξιά (least significant) bits του καταχωρητή *rs2* στη μνήμη. Επίσης στον 64-μπιτο RISC-V υπάρχει η **`ld`** (load double) που διαβάζει 64 bits (8 bytes) από τη μνήμη, και η **`sd`** (store double) που γράφει 64 bits (8 bytes) στη μνήμη. Στους σημερινούς 64-μπιτους υπολογιστές, οι compilers της C θεωρούν τις μεταβλητές τύπου pointer πάντα σαν 64-μπιτες ποσότητες (προφανώς), ενώ τις μεταβλητές τύπου **`int`** σαν 32-μπιτους ακεραίους και τις μεταβλητές τύπου **`long long int`** σαν 64-μπιτους ακεραίους· τον τύπο **`long int`**, τα μεν Windows τον θεωρούν 32 bits, το δε Linux τον θεωρεί 64 bits.

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 (*sp* = x2) και η απόκλιση ορίζει ποιάν από τις τοπικές μεταβλητές της διαδικασίας ζητάμε· και (γ) όταν ο καταχωρητής είναι ο global pointer (*gp* = x3) και η απόκλιση ορίζει μία από τις καθολικές βαθμωτές μεταβλητές. Μιά άλλη χρήση του τρόπου διευθυνσιοδότησης "σταθερά + καταχωρητής" θα μπορούσε να ήταν με τη σταθερά να είναι η διεύθυνση βάσης ενός (στατικά allocated) πίνακα (array) και ο καταχωρητής να είναι το index του στοιχείου του πίνακα πολλαπλασιασμένο (ήδη) επί το μέγεθος του στοιχείου του πίνακα, οπότε προσπελαύνουμε το στοιχείο με εκείνο το index. Όμως στην πράξη, στον RISC-V, αυτό είναι σχεδόν αδύνατο διότι θα έπρεπε ο πίνακας να ξεκινά στα πρώτα 2 KBytes του χώρου διευθύνσεων, ώστε η διεύθυνση βάση του να χωρά στα μόλις 12 bits που έχει η (προσημασμένη) σταθερά. Ευτυχώς, η χρήση αυτή δεν είναι απαραίτητη, διότι συνήθως οι compilers αλλάζουν την αριθμητική με array indexes σε αριθμητική με array element pointers μέσα στους βρόχους που επεξεργάζονται στοιχεία πινάκων.

3.2 Διευθύνσεις Bytes και η σχέση Ευθυγράμμισης-Ταχύτητας:

Οι διευθύνσεις μνήμης στον RISC-V, όπως και σχεδόν σε όλους τους μοντέρνους επεξεργαστές, αναφέρονται σε **Bytes** στη μνήμη, δηλαδή ο RISC-V είναι "**Byte Addressable**". Έτσι, μια 32-μπιτη λέξη (π.χ. ένας ακέραιος `int` της C) 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 πάντα η διεύθυνση εκείνου από τα 4 Bytes του που έχει τη **μικρότερη** ("πρώτη") από τις 4 διευθύνσεις.

Γιά λόγους ταχύτητας προσπέλασης της μνήμης είναι προτιμότερο οι ποσότητες που προσπελαύνουν οι εντολές load και store να είναι **ευθυγραμμισμένες** (aligned) στα *φυσικά του όρια*, δηλαδή μια ποσότητα μεγέθους *N* Bytes να έχει διεύθυνση που να είναι ακέραιο πολλαπλάσιο του *N*. Αυτό αντιστοιχεί στις διευθύνσεις που θα είχαν τέτοιες ποσότητες εάν γεμίζαμε τη μνήμη, από την αρχή (διεύθυνση 0), με τέτοιες ποσότητες. 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. Όταν ικανοποιείται αυτός ο περιορισμός, τότε 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. Όταν όμως ένα επόμενο μοντέλο υπολογιστή έχει 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. Στον RISC-V μιά τέτοια ευθυγράμμιση είναι έντονα επιθυμητή γιά λόγους ταχύτητας, αλλά πάντως προαιρετική (γιά να εξυπηρετούνται παλαιά προγράμματα) (ενώ στον MIPS η ευθυγράμμιση αυτή είναι υποχρεωτική).

3.3 Αρίθμηση των Bytes: Μηχανές Little-Endian (και Big-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 στο παρελθόν δεν είχε υπάρξει συμφωνία μεταξύ των κατασκευαστών επεξεργαστών για τη σειρά αυτή, με συνέπεια να υπάρχουν δύο διαφορετικοί τύποι επεξεργαστών – οι επονομαζόμενοι "Little-Endian" και οι επονομαζόμενοι "Big-Endian". Σήμερα πάντως, μοιάζει να επικρατούν οι *Little-Endian*, και γι' αυτό ο RISC-V είναι 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 αυτούς τους υπολογιστές, το MS Byte του κάθε ακεραίου έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των Bytes εντός του ακεραίου 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. Η λογική των Big-endians είναι ότι τα character strings διαβάζονται κανονικά από τους ανθρώπους που στις γλώσσες τους γράφουν από τα αριστερά προς τα δεξιά.
- **Little-Endian**: Στους επικρατέστερους σήμερα υπολογιστές, και στον RISC-V, το LS Byte του κάθε ακεραίου έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των Bytes εντός του ακεραίου 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 λογική των Little-endians είναι ότι τα Bytes αριθμώνται προς την ίδια κατεύθυνση προς την οποία αριθμώνται και τα bits ενός ακεραίου, δηλαδή προς την κατεύθυνση που αυξάνουν οι δυνάμεις του 2 - συντελεστές των bits του ακεραίου.

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 little-endian ή big-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).

Άσκηση 3.4: Πίνακας Ακεραίων

Γράψτε ένα πρόγραμμα σε Assembly του (32-μπιτου) RISC-V (γιά τον RARS) που να διαβάζει 8 ακεραίους (`int`) 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" και ".space" του Assembler του RARS για να κρατήστε χώρο στη μνήμη δεδομένων (data segment) γιά τον πίνακα "a[ ]", μεγέθους 8 ακεραίων = 32 Bytes (το όρισμα της οδηγίας .space είναι σε Bytes). Τοποθετήστε κατάλληλα το label "a" ώστε να μπορείτε πιο κάτω να αναφερθείτε στη διεύθυνση όπου αρχίζει ο χώρος που κρατήσατε. (Εάν θέλετε να έχετε και καλά ευθυγραμμισμένους ακεραίαους, χρησιμοποιήστε και την οδηγία .align του RARS – δείτε την καρτέλα Help→RISCV→Directives του RARS).
- Στην αρχή του προγράμματός σας, τοποθετήστε τη διεύθυνση όπου αρχίζει ο πίνακας a[ ] σε έναν καταχωρητή, π.χ. στον x5. 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 στα 12 bits του offset ή όχι (μάλλον όχι...). Σε όλα αυτά έρχεται να σας βοηθήσει η **ψεύδο**εντολή (pseudoinstruction) "`la rd, label`" του Assembler του RARS: 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 12 bits, δύο εντολές αλλιώς). Παράδειγμα χρήσης της ψευδοεντολής "la" θα βρείτε στην § 2.2, εκεί που ετοιμάζαμε τα ορίσματα για τις εκτυπώσεις των 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 περιβάλλοντος) 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 RISC-V είναι "σταθερή ποσότητα (immediate offset) + καταχωρητής" –μην χρησιμοποιήσετε τις **ψεύδο**διευθυνσιοδοτήσεις του RARS (πράσινη καρτέλα Help→RISCV).
- 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) με τον εαυτό του ή με προηγούμενα αποτελέσματα.
- Τέλος, ξαναγυρίστε στην αρχή (όπως και στην άσκηση 2), ώστε η ίδια δουλειά να επαναλαμβάνεται επ' αόριστο.

Άσκηση 3.5: Υπολογιστές Little-Endian και Big-Endian

i. Χρησιμοποιήστε τον RARS 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 RARS και μελετήστε τα περιεχόμενα της μνήμης δεδομένων στην καρτέλα Data Segment. Γράψτε τις διαπιστώσεις σας σε μορφή σχολίων μέσα στον κώδικά σας αυτής της άσκησης, 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 little-endian, και (β) σε μια μηχανή big-endian. Δώστε την απάντησή σας, μαζί με τις αριθμητικές πράξεις που κάνατε (στο δεκαδικό), πάλι σε μορφή σχολίων μέσα στον κώδικά σας.

iii. Επαληθεύστε την απάντησή σας με το πρόγραμμά σας στον RARS: Ζητήστε από τον Assembler να βάλει το string στη μνήμη δεδομένων, και μέσα από το πρόγραμμά σας αντιγράψτε εκείνη τη λέξη (ολόκληρη! - με μία εντολή lw) σ' έναν καταχωρητή και ζητήστε να τυπωθεί (μ' ένα κάλεσμα συστήματος) σαν ακέραιος. (Φαντάζομαι ότι ο RARS θα συμπεριφέρεται σαν little-endian, όπως και ο RISC-V, εκτός και έχει κληρονομήσει τη συμπεριφορά του QtSpim που έλεγε ότι συμπεριφέρονταν το ίδιο με τον υπολογιστή όπου έτρεχε...).

Τρόπος Παράδοσης: Παραδώστε μέσω **`turnin ex03@hy225 [directoryName]`** τα εξής:
(1) τον πηγαίο κώδικά σας της άσκησης 3.4, "ex03_4.asm"
(2) τον πηγαίο κώδικά σας της άσκησης 3.5, "ex03_5.asm"
(3) ένα στιγμιότυπο (screen-dump) του τρεξίματος της άσκησης 3.4, "ex03_4.jpg"
(4) ένα στιγμιότυπο (screen-dump) του τρεξίματος της άσκησης 3.5, "ex03_5.jpg"
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.
