HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2025

Τμ. Επ. Υπολογιστών

Ασκήσεις 1: Εισαγωγή, Καταχωρητές, add/sub, addi, Γλώσσα Assembly και ο Προσομοίωτης RARS

Από 1η για 2η εβδομάδα του Εξαμήνου

Βιβλίο: Διαβάστε την αρχή του κεφαλαίου 2 (§2.1, το μεγαλύτερο μέρος της §2.2, και την αρχή της §2.3): σελ. 102-108. (Στις διαλέξεις (διαφάνεια 15 της σειράς 01a και διαφάνειες 9-11 της 05b) θα αναφερθούμε λίγο και στον Assembler, Linker και συναφή· αυτά υπάρχουν στην §2.12, σελίδες 169 και πέρα, σε αυξανόμενο βάθος, μεγαλύτερο απ' όσο αντιστοιχεί σε αυτό εδώ το μάθημα).

1.1 Βασική Λειτουργία του Υπολογιστή:

Οι υπολογιστές αποτελούνται από:

- **Μονάδες Εισόδου/Εξόδου (I/O - Input/Output):** είναι οι "περιφερειακές" συσκευές, μέσω των οποίων ο υπολογιστής επικοινωνεί με τον έξω κόσμο –πληκτρολόγιο, ποντίκι, μικρόφωνο, κάμερα, οθόνη, μεγάφωνα, δίσκοι (μαγνητικοί/SSD/flash), διεπαφές δικτύου (network interfaces), κλπ.
- **Κεντρική Μνήμη (Main Memory):** δισεκατομμύρια στοιχεία αποθήκευσης πληροφορίας –κάτι σαν flip-flops αλλά μικρότερα στην κατασκευή– οργανωμένα σαν πολλές (συχνά δισεκατομμύρια) "λέξεις" (words) των κάμποσων (π.χ. 32, 64) bits καθεμία. Μπορούμε να την φανταστούμε σαν έναν πολύ μεγάλο πίνακα (array) από bytes ή από λέξεις.
- **Επεξεργαστής (Processor)** ή Κεντρική Μονάδα Επεξεργασίας (Central Processing Unit - CPU): περιέχει, πρώτον, τα κυκλώματα *ελέγχου (control)* που καθοδηγούν την εκτέλεση των λειτουργιών που θα πούμε πιό κάτω. Δεύτερον, περιέχει τους "*δρόμους δεδομένων*" *(datapath)* μέσα από τους οποίους περνάνε και οι οποίοι επεξεργάζονται τις πληροφορίες που ανταλλάσσει ο επεξεργαστής με τη μνήμη και τις περιφερειακές συσκευές. Τρίτον, περιέχει τους "*καταχωρητές γενικού σκοπού*" *(general-purpose registers)* (καθώς και κάμποσους καταχωρητές ειδικού σκοπού). Οι καταχωρητές γενικού σκοπού είναι μια πολύ μικρή μνήμη, μεγέθους συνήθως κάμποσων δεκάδων λέξεων (π.χ. 32 λέξεων). Ένα ηλεκτρονικό κύκλωμα, όσο πιο μικρό τόσο πιο γρήγορο είναι, γι' αυτό και το σύνολο αυτών των καταχωρητών ("register file") είναι πολύ μικρό για να είναι πολύ γρήγορο (και για οικονομία στα bits της εντολής, όπως θα δούμε αργότερα).

Μιά απόφαση πρωταρχικής σημασίας στην οργάνωση των υπολογιστών είναι ότι στη μνήμη αποθηκεύουμε **και τα δεδομένα** (αριθμούς, πληροφορίες), **και τις "εντολές"** (instructions) –τις οδηγίες δηλαδή προς τον υπολογιστή για το τι είδους πράξεις και επεξεργασίες αυτός πρέπει να κάνει πάνω στα δεδομένα. Από φιλοσοφική άποψη, με το να αποθηκεύονται και τα δεδομένα και οι εντολές στην ίδια μνήμη, κωδικοποιημένα και τα δύο με παρόμοιους τρόπους (σαν δυαδικές λέξεις π.χ. των 32 ή 64 bits καθεμία), ανοίγει ο δρόμος στο να μπορεί να δει και να επεξεργαστεί ο υπολογιστής τις ίδιες του τις εντολές σαν δεδομένα, αν και σπανιότατα το κάνουν αυτό τα προγράμματα γιά τον εαυτό τους: ο μεταφραστής (compiler) είναι το βασικό πρόγραμμα υπολογιστή που γεννά εντολές –αλλά γιά ένα άλλο πρόγραμμα· από την άλλη, τα "αυτομεταβαλόμενα προγράμματα" (self-modifying code) αποτελούν εφιάλτη γιά το debuuging, και γι' αυτό τα αποφεύγουμε. Από πρακτική άποψη, το να είναι μαζί τα δεδομένα και το

πρόγραμμα στην ίδια μνήμη επιτρέπει την πλήρη αξιοποίηση όλου του χώρου μνήμης, χωρίς να μένουν αχρησιμοποίητα κενά, π.χ. όταν έχουμε μικρό πρόγραμμα με μεγάλα δεδομένα, ή μεγάλο πρόγραμμα με μικρά δεδομένα.

Ο υπολογιστής λειτουργεί με τον εξής βασικό **επαναληπτικό** τρόπο. Ο επεξεργαστής *διαβάζει μια εντολή* από τη μνήμη. Στη συνέχεια την αποκωδικοποιεί για να καταλάβει τι λέει, και κάνει τις δουλειές που αυτή λέει, δηλαδή την *εκτελεί*. Οι δουλειές αυτές είναι συνήθως απλές, όπως π.χ. μεταφορά δεδομένων από ή προς τη μνήμη, ή από ή προς περιφερειακές συσκευές, ή αριθμητικές πράξεις πάνω σε δεδομένα, ή αποφάσεις "αλλαγής πορείας". Μετά, μόλις τελειώσει η εκτέλεση της εντολής, ο επεξεργαστής διαβάζει από τη μνήμη και εκτελεί την "επόμενη" εντολή, και ούτω καθ' εξής επ' αόριστο. Η "επόμενη" εντολή συνήθως είναι η εντολή που βρίσκεται αποθηκευμένη στην επόμενη λέξη μνήμης, εκτός αν η εκτέλεση της προηγούμενης εντολής πει στον επεξεργαστή να "αλλάξει πορεία"....

Γιά να ξέρει ο επεξεργαστής ποια είναι η "επόμενη" εντολή που πρέπει να διαβάσει και εκτελέσει, υπάρχει μέσα του ένας ειδικός καταχωρητής, ο **"Μετρητής Προγράμματος"** (program counter - PC) –ίσως μιά σωστότερη ονομασία να ήταν "δείκτης προγράμματος" (program pointer) ή "δείκτης εντολών" (instruction pointer), αλλά έχει μείνει από παλιά η ονομασία "PC". Στο τέλος της εκτέλεσης μιας εντολής, ο PC περιέχει τη **διεύθυνση μνήμης** της επόμενης εντολής που πρέπει να διαβαστεί από τη μνήμη και να εκτελεστεί. Αν φανταστούμε τη μνήμη σαν έναν μεγάλο πίνακα (array), *M[ ]*, τότε η "διεύθυνση μνήμης" είναι ο δείκτης (index), *i*, που μας λέει να διαβάσουμε την επόμενη εντολή από το *M[i]*. Με όρους της γλώσσας C, η διεύθυνση μνήμης της επόμενης εντολής είναι ένας pointer στην επόμενη εντολή.

1.2 Γλώσσες Μηχανής:

Οι επεξεργαστές καταλαβαίνουν και εκτελούν εντολές από ένα ρεπερτόριο πολύ μικρότερο και απλούστερο από τις γλώσσες υψηλού επιπέδου (High Level Languages - HLL). Οι εντολές που δέχεται και εκτελεί το hardware είναι αναγκαστικά κωδικοποιημένες σαν δυαδικά σύμβολα, και λέγονται *Γλώσσα Μηχανής* (Machine Language). Κάθε οικογένεια επεξεργαστών που είναι μεταξύ τους "binary compatible" έχει την ίδια γλώσσα μηχανής, που είναι διαφορετική από τη γλώσσα μηχανής άλλων οικογενειών. Ένα δημοφιλές σήμερα στυλ γλωσσών μηχανής ("αρχιτεκτονικών") είναι αυτές που έχουν σχετικά λίγες και απλές μόνο εντολές, και που γι' αυτό περιγράφονται σαν *Υπολογιστές Ελαττωμένου Ρεπερτορίου Εντολών* (Reduced Instruction Set Computers - *RISC*). Ενα υποσύνολο ενός τέτοιου ρεπερτορίου εντολών, του ονομαζόμενου *RISC-V* (RISC five, RISC πέντε, μερικές φορές "RV"), θα χρησιμοποιήσουμε σαν παράδειγμα σε αυτό το μάθημα. Αλλες γλώσσες μηχανής στυλ RISC είναι αυτές των ARM, MIPS, PowerPC, SPARC, Alpha. Η σειρά x86 της Intel έχει πιο πολύπλοκη γλώσσα μηχανής, όπως επίσης και οι ιστορικού ενδιαφέροντος υπολογιστές VAX της δεκαετίας του '80 που θα τους αναφέρω κατά καιρούς σαν αντιπαράδειγμα. Στο μάθημα αυτό, μέχρι και το 2018, χρησιμποποιούσαμε σαν παράδειγμα το ρεπερτόριο εντολών του MIPS αντί τώρα του RISC-V.

Το ρεπερτόριο εντολών RISC-V δημιουργήθηκε τη δεκαετία του '10, είναι το πρώτο *ανοικτό* (open) ρεπερτόριο εντολών, κερδίζει συνεχώς σε δημοτικότητα τα τελευταία χρόνια, και υπάρχουν σημαντικές ενδείξεις ότι μπορεί να εξελιχθεί σε δημοφιλέστατο διεθνές standard. Η ιστοσελίδα γιά το RISC-V είναι η riscv.org και η βασική αναφορά γιά το ρεπερτόριο εντολών είναι στη σελίδα riscv.org/specifications/ όπου μπορείτε να διαβάσετε ή "κατεβάσετε" τη σχετική προδιαγραφή (PDF, 670 σελίδες, 4.5 MBytes). Ένα πρόσθετο ενδιαφέρον γιά εμάς είναι ότι το μεγάλο τρέχον Ευρωπαϊκό εγχείρημα *European Processor Initiative (EPI)* (δείτε π.χ. ένα video εδώ) (στο οποίο συμμετέχει και το ΙΤΕ-ΙΠ εδώ στο Ηράκλειο της Κρήτης) περιλαμβάνει και δραστηριότητες βασισμένες στο RISC-V. Για την εξαετία 2024-2029, η Ευρωπαϊκή Ένωση

χρηματοδοτεί ένα έργο ("DARE") μεγάλης κλίμακας (προϋπολογισμού 240 M€) για Ευρωπαϊκούς Επεξεργαστές RISC-V Υψηλών Επιδόσεων στο οποίο συμμετέχει και το ΙΤΕ-ΙΠ.

Κάθε εντολή γλώσσας μηχανής αποτελείται από έναν **κώδικα πράξης** (operation code - **opcode**), και από **τελεστέους (operands)** που περιγράφουν πάνω σε τι θα γίνει η πράξη. Οι τελεστέοι των εντολών αριθμητικών πράξεων του RISC-V είναι πάντα καταχωρητές γενικού σκοπού (registers) του επεξεργαστή, ή σταθερές ποσότητες, αλλά όχι θέσεις μνήμης. Ο βασικός RISC-V έχει 32 καταχωρητές των 32 bits καθένας: 32 bits είναι το μέγεθος λέξης (word) αυτού του βασικού RISC-V. Πιο σύγχρονα μοντέλα επεξεργαστών, σήμερα, έχουν μέγεθος λέξης 64 bits. Το βιβλίο στηρίζεται σε μιά τέτοια πιό σύγχρονη έκδοση του RISC-V που είναι 64-μπίτη, δηλαδή έχει (32) καταχωρητές των 64 bits καθένας. Οι εντολές αριθμητικών πράξεων του RISC-V έχουν πάντα τρεις (3) τελεστέους, για λόγους ομοιομορφίας. Η ομοιομορφία μεταφράζεται σε απλότητα του hardware, πράγμα που ευνοεί την υψηλότερη ταχύτητα.

1.3 Η Γλώσσα Assembly του RISC-V:

Για να γίνει η γλώσσα μηχανής λίγο πιο φιλική προς τον άνθρωπο, χρησιμοποιούμε ένα συμβολικό όνομα για κάθε επιτρεπτό opcode, ένα δεκαδικό ή δεκαεξαδικό αριθμό με μερικά απλά σύμβολα για κάθε τελεστέο, και γράφουμε αυτά τα στοιχεία της κάθε εντολής σε μία χωριστή γραμμή. Αυτή είναι η γλώσσα *Assembly*, η οποία μπορεί να μεταφραστεί σε γλώσσα μηχανής από ένα σχετικά απλό πρόγραμμα υπολογιστή –τον λεγόμενο *Assembler*. Οι γλώσσες υψηλού επιπέδου (HLL) (όπως η C) μεταφράζονται σε Assembly από ένα σημαντικά πολυπλοκότερο πρόγραμμα, τον *Compiler*.

- Η βασική εντολή αριθμητικής πράξης του RISC-V είναι η **`add`** που κάνει πρόσθεση ακεραίων. Η εντολή Assembly "`add x23, x12, x14`" διαβάζει τα περιεχόμενα των καταχωρητών υπ'αριθμόν 12 και 14, τα προσθέτει, και γράφει το αποτέλεσμα στον καταχωρητή υπ'αριθμόν 23, δηλαδή x23 := x12 + x14.
- Η εντολή **`sub`** (subtract) είναι παρόμοια αλλά, αντί να προσθέτει, αφαιρεί τον τρίτο τελεστέο από το δεύτερο: η "`sub x23, x12, x14`" γράφει στον καταχωρητή υπ'αριθμόν 23 το αποτέλεσμα της αφαίρεσης x12 μείον x14.
- Η εντολή **`addi`** (add immediate) είναι παρόμοια της add (πρόσθεση), αλλά ο τρίτος τελεστέος της είναι σταθερός αριθμός αντί καταχωρητή: η "`addi x23, x12, 157`" διαβάζει το περιεχόμενο του καταχωρητή 12, προσθέτει τον αριθμό 157 σε αυτό, και γράφει το αποτέλεσμα στον καταχωρητή 23. Εάν η μεταβλητή *i* βρίσκεται στον καταχωρητή 7, τότε η εκχώρηση *i=i+1* μεταφράζεται σε "`addi x7, x7, 1`".
- Ο **καταχωρητής x0** του RISC-V περιέχει πάντα την σταθερή ποσότητα μηδέν, ανεξαρτήτως του τι γράφει κανείς σε αυτόν (οι εγγραφές σε αυτόν αγνοούνται από το hardware). Ετσι, εάν η μεταβλητή *i* βρίσκεται στον καταχωρητή 7 όπως παραπάνω, τότε η αρχικοποίηση *i=1* μπορεί να γίνει: "`addi x7, x0, 1`".

Γιά έναν υπολογισμό συνθετότερης αριθμητικής έκφρασης, δείτε το πρώτο παράδειγμα στην §2.3 του βιβλίου. Εκτός από τα ονόματα x0, x1, ..., x31 γιά τους 32 καταχωρητές του RISC-V, χρησιμοποιούνται και τα λίγο πιό αφηρημένα ονόματα t0, t1, ... (temporary), s0, s1, ... (saved), a0, a1, ... (argument), κ.α., ανάλογα με τη χρήση που επιφυλάσσουν σε καθένα τους οι συμβάσεις καλέσματος διαδικασιών (procedure calling conventions), όπως θα δούμε στις διαλέξεις 6.

Γιά να εκτελεστεί ένα πρόγραμμα, οι εντολές του γράφονται στην κεντρική μνήμη η μία "κάτω" από την άλλη, δηλαδή σε συνεχόμενες θέσεις (διευθύνσεις) μνήμης. Μετά την ανάγνωση και εκτέλεση μιάς εντολής, ο επεξεργαστής αυξάνει τον PC κατά το μέγεθος της εντολής που εκτελέστηκε, οπότε αυτός (ο PC) δείχνει στην επόμενη (την "από κάτω") εντολή. Η σειριακή αυτή

εκτέλεση εντολών διακόπτεται όταν εκτελείται μιά εντολή **μεταφοράς ελέγχου** (control transfer instruction - CTI). Τέτοιες, όπως θα δούμε, είναι, μεταξύ άλλων, οι *διακλαδώσεις (branch)* και τα *άλματα (jump)*. Η ψεύδοεντολη "`j label`" (*jump* to label, ή παλαιότερα *goto* label, που ο Assembler την μεταφράζει σε άλλη, γενικότερη εντολή άλματος) κάνει ώστε η επόμενη εντολή που θα εκτελεστεί να είναι η εντολή στη διεύθυνση μνήμης `label`, αντί να είναι η "από κάτω" εντολή. Με άλλα λόγια, η (ψεύδο)εντολή "`j label`" φορτώνει τη διεύθυνση `label` στον καταχωρητή PC.

1.4 Ο Προσομοιωτής RARS:

Προγράμματα γραμμένα σε γλώσσα Assembly του RISC-V μπορεί να τα δοκιμάσει κανείς και να παρακολουθήσει πώς τρέχουν χρησιμοποιώντας τον *προσομοιωτή* **RARS** (RISC-V Assembler and Runtime Simulator), γραμμένο σε Java, διαθέσιμο δημόσια μέσω GitHub: github.com/TheThirdOne/rars . Οι προσομοιωτές είναι προγράμματα υπολογιστή που προσπαθούν να συμπεριφέρονται όσο πιο παρόμοια γίνεται, από ορισμένες απόψεις, με ένα φυσικό σύστημα. Εν προκειμένω, ο RARS συμπεριφέρεται σαν ένα τσιπάκι RISC-V από την άποψη των περιεχομένων των καταχωρητών και της μνήμης μετά την εκτέλεση κάθε εντολής· από την άλλη μεριά, πάντως, δεν δίνει καμία πληροφορία, π.χ., για το χρόνο εκτέλεσης της κάθε εντολής (ποιες εντολές εκτελούνται γρήγορα και ποιες αργά), όπως και για άλλες απόψεις του φυσικού συστήματος.

Οδηγίες: κατεβάστε τη νεότερη σταθερή έκδοση, 1.6, από το directory github.com/TheThirdOne/rars/releases/tag/v1.6 –δηλαδή το αρχείο `rars1_6.jar` που βρίσκεται στα assets, από το σύνδεσμο → github.com/TheThirdOne/rars/releases/download/v1.6/rars1_6.jar

- Θα χρειαστεί να έχετε εγκαστεστημένη τη Java στο μηχάνημα σας. Αν δεν την έχετε ήδη, για να την εγκαταστήσετε, δείτε εδώ: www.java.com/en/download/manual.jsp –επιλέξτε ανάλογα το λειτουργικό σας.
- Γιά τους χρήστες **Windows, μόνον:** Αφού εγκασταστήσετε την Java, πρέπει να την προσθέσετε στη μεταβλητή συστήματος `PATH` –δείτε εδώ: www.mkyong.com/java/how-to-set-java_home-on-windows-10/
- Για να τρέξετε το αρχείο, ανοίγετε ένα terminal (windows command line)· αν το `rars.jar` είναι π.χ. στον φάκελο `225/rars/` εκτελέστε την εντολή: `cd 225/rars` και στην συνέχεια εκτελείτε την εντολή: `java -jar rars.jar`

Αν το πρόγραμμα τρέξει κανονικά, θα δείτε την αρχική οθόνη με τα 3 βασικά παράθυρα, τους Registers δεξιά, και αριστερά 2 tabs, το Edit και το Execute, και κάτω άλλα 2 tabs, το Messages και το Run I/O. Μπορεί τα παράθυρα να είναι minimized ή να μην φαίνονται καλά· αν υπάρχει τέτοιο πρόβλημα προσπαθήστε να κλικάρετε μερικές φορές.

- Πολύ χρήσιμο θα σας φανεί το Help (ανοίγει πατώντας Help→Help), όπου θα βρείτε το πλήρες ρεπερτόριο εντολών του προσομοιωτή, καθώς και τον τρόπο σύνταξης της κάθε μιάς απο αυτές. Επίσης, γιά τις επόμενες σειρές ασκήσεων, θα βρείτε τις Οδηγίες προς τον Assembler (Directives) και τις κλήσεις (λειτουργικού) συστήματος.
- Πατώντας το File→Open (πάνω αριστερά), μπορείτε να φορτώσετε ένα αρχείο Assembly (σύμβαση: το όνομα του αρχείου να τελειώνει σε `.asm`). Το αρχείο που φορτώσατε θα το δείτε στο Edit tab.
- Εν συνεχεία, πατώντας Run→Assemble θα δείτε ότι άνοιξε το Execute tab, ακόμα όμως το πρόγραμμα δεν τρέχει.
- Μπορείτε να δείτε τα Text και Data segments του προγραμματος σας, και σε ποιες διεθύνσεις έχουν φορτωθεί οι εντολές σας: Το **Text Segment** είναι η περιοχή της μνήμης του υπό

προσομοίωση υπολογιστή όπου βρίσκονται οι εκτελέσιμες **εντολές**, σε αντίθεση με τα δεδομένα στη μνήμη που βρίσκονται στο **Data Segment** –εμείς δεν θα ασχοληθούμε με δεδομένα στη μνήμη σε αυτή την άσκηση. Η διεύθυνση της κάθε εντολής διαφέρει κατά 4 από αυτήν της προηγούμενης διότι οι εντολές του βασικού RISC-V έχουν μέγεθος 4 Bytes καθεμία.

- Για να τρέξει το πρόγραμμα σας πατήστε Run→Go ή το πράσινο κουμπί με το χαρακτηριστικό σύμβολο του "Play". Μπορείτε επίσης να τρέξετε "βήμα βήμα" το πρόγραμμα σας, πατώντας Run→Step ή το άλλο πράσινο κουμπί που έχει το σύμβολο Play με το 1.
- **32-μπιτος vs. 64-μπιτος** RISC-V: ο RARS, by default, προσομοιώνει 32-μπιτο RISC-V. Όμως, από την έκδοση 1.5 μπορεί να προσομοιώσει και 64-μπιτο RISC-V, αρκεί να το επιλέξτε στα Settings→64 bit.
- Κάτω-κάτω στο παράθυρο των καταχωρητών (Registers - δεξιά), σαν "ειδικός" 33ος "καταχωρητής", φαίνεται ο **μετρητής προγράμματος (PC)**. Όποτε ο RARS είναι σταματημένος –δηλαδή δεν "τρέχει" το πρόγραμμά μας αλλά περιμένει– το περιεχόμενο του PC είναι η διεύθυνση της επόμενης εντολής που πρόκειται να εκτελεστεί μόλις ξαναξεκινήσουμε την προσομοίωση, η οποία εντολή φαίνεται με κίτρινο, δηλαδή η εντολή αυτή δεν έχει "εκτελεστεί" ακόμα.
- Αν το πρόγραμμα σας τυπώνει στην "κονσόλα" (σε επόμενες σειρές ασκήσεων), θα το δείτε στο Run I/O tab ακριβώς απο κάτω. Επίσης, αν δέχεται και είσοδο από το "πληκτρολόγιο", χρησιμοποιείτε το Run I/O tab για την δώσετε. (Επιπλέον, αν θέλετε μπορείτε να δίνετε τιμές από το πληκτρολόγιο σε ένα pop-up window αντί να γράφετε στο Run I/O tab· για να το ενεργοποιήσετε αυτό πατήστε Settings και κάντε check το "Popup dialog for input syscalls(5,6,7,8,12)").
- Τέλος, μπορείτε να σταματήσετε το πρόγραμμα σας πατώντας το STOP, που είναι το πράσινο κουμπί με το λευκό τετράγωνο, ή ακόμα και να το "παγώσετε" πατώντας το PAUSE, που είναι το πράσινο κουμπί με 2 λευκές γραμμές.
- Για να κάνετε Reset το πρόγραμμα σας, δηλαδή να καθαρίσετε τα περιεχόμενα των καταχωρητών και της μνήμης καθώς και να επαναφέρετε τον PC, πατήστε Run→Reset, ή το πράσινο κουμπί με τα δύο αριστερά βελάκια.
- Χρήσιμο μπορεί να σας φανεί, τώρα ή και περισσότερο σε επόμενες ασκήσεις, και το παράδειγμα "hello world" σε Assembly του RISC-V, με επεξήγηση του τι κάνει η κάθε γραμμή, από τη σελίδα: github.com/TheThirdOne/rars/wiki/Creating-Hello-World

Άσκηση 1.5: Κώδικας Γνωριμίας με τον RARS

Αντικείμενο της παρούσας άσκησης είναι να γνωριστείτε με τη γλώσσα Assembly του RISC-V και με τη χρήση του RARS. Για το σκοπό αυτό, **μελετήστε και αντιγράψτε** σε ένα αρχείο (π.χ. "`ex01.asm`") τον παρακάτω κώδικα –ή διάφορες παραλλαγές του που προτιμάτε– και τρέξτε τον στον RARS.

```
     .text  # Directive ".text": put the following into program memory
            # Register use:  x6: variable "i"; x7: variable "k";
main:                       # label "main" = address for "j" to jump to
     addi   x6, x0,  10     # init. i=10; (x0==0 always)
     addi   x7, x0,  64     # init. k=64; (64 decimal = 40 hex)
     add    x28, x6, x7     # x28 := i+k = 74 dec = 4a hex
     add    x28, x28, x28   # x28 := 74+74=148 dec = 94 hex
     add    x28, x28, x7    # x28 := 148+64=212 dec = d4 hex
     addi   x7, x7, -1      # k := k-1 = 63 dec = 3f hex
     sub    x7, x7, x6      # k := k-i = 53 dec = 35 hex
     j      main            # jump back to main (infinite loop)
```

- Το κομμάτι κάθε γραμμής μετά το # είναι σχόλια (στο βιβλίο, τα σχόλια σημειώνονται με "//" αντί "#").
- Οι γραμμές μετά το `.text` είναι εκτελέσιμος κώδικας (σε αντίθεση με αριθμητικά δεδομένα στη μνήμη, που εδώ δεν έχουμε).
- Η τρίτη γραμμή ορίζει την ετικέτα `main` ίση με τη διεύθυνση μνήμης όπου θα τοποθετηθεί αυτό που ακολουθεί ακριβώς μετά –στην περίπτωσή μας η πρώτη εντολή (`addi`) του προγράμματός μας (0x00400000 (δεκαεξαδικό) στον RARS).
- Μετά το `main:` ακολουθούν οκτώ εντολές στη γλώσσα Assembly του RISC-V: έξι εντολές πρόσθεσης (`add` ή `addi`), μία εντολή αφαίρεσης (`sub`), και μία (ψεύδο)εντολή άλματος (`j`).
- Η (ψεύδο)εντολή άλματος `j main`, που είναι η τελευταία εντολή του προγράμματός μας, λέει στον επεξεργαστή να εκτελέσει σαν επόμενη εντολή την εντολή που βρίσκεται στη διεύθυνση μνήμης `main`, δηλαδή την πρώτη εντολή (`addi`) του προγράμματός μας. Αυτό δημιουργεί έναν άπειρο βρόχο: οι 8 αυτές εντολές του προγράμματός μας θα εκτελούνται συνεχώς, επ' άπειρο· ένα κανονικό πρόγραμμα, φυσικά, δεν πρέπει να κάνει κάτι τέτοιο, αλλά εδώ δεν πρόκειται για κανονικό πρόγραμμα....

Ξεκινήστε τον **RARS** με τον τρόπο που είπαμε παραπάνω, στην §1.4. Χρησιμοποιήστε τη δυνατότητα "**single step**" για να εκτελούνται μια-μια οι εντολές και να τις βλέπετε. Παρακολουθήστε ότι μετά από κάθε τέτοια προσομοίωση ο PC έχει αυξηθεί κατά 4 (επειδή οι εντολές του RISC-V έχουν μέγεθος 4 Bytes καθεμία), και ότι αλλάζουν οι τιμές (περιεχόμενα) των καταχωρητών όπου γράφει η κάθε εντολή. Τα περιεχόμενα αυτά ο RARS τα δείχνει στο δεκαεξαδικό by default, αν και αυτό μπορείτε να το αλλάξετε από τα "Settings". Όταν η εκτέλεση φτάσει στην (ψεύδο)εντολή `j main`, παρατηρήστε ότι ο PC παίρνει ξανά την τιμή 00400000 (δεκαεξαδικό), και ξαναρχίζει η εκτέλεση του ίδιου προγράμματός μας, από την αρχή.

1.6 Ο εφεδρικός προσομοιωτής Venus:

Εάν δεν καταφέρετε να εγκαταστήσετε την Java και τον RARS στον υπολογιστή σας, μπορείτε εναλλακτικά να χρησιμοποιήσετε έναν παρόμοιο προσομοιωτή, ονόματι Venus, που βρίσκεται προεγκατεστημένος και τρέχει έτοιμος μέσω του web browser σας στη διεύθυνση: www.kvakil.me/venus/
Κάντε copy-paste το πρόγραμμά σας στην καρτέλα "Editor", και αυτόματα, μόλις πάτε στην άλλη καρτέλα, "Simulator", το πρόγραμμά σας είναι έτοιμο assembled και μπορείτε να το τρέξετε βήμα-βήμα ("Step") (το "Run" δεν δείχνει τίποτα ιδιαίτερο, λόγω του άπειρου βρόχου). Μερικές οδηγίες χρήσης, εάν χρειαστείτε, υπάρχουν στο: github.com/kvakil/venus/wiki . Πάντως, ο προσομοιωτής Venus έχει λιγότερες δυνατότητες από τον RARS –ιδιαίτερα *δεν* έχει environment (system) calls γιά να διαβάζει είσοδο από το πληκτρολόγιο– επομένως δεν θα μπορέσει να σας εξυπηρετήσει σε επόμενες ασκήσεις που θα χρειάζονται τέτοια είσοδο.

Τρόπος Παράδοσης:
Κάντε αυτή την άσκηση γιά να μάθετε τη χρήση του RARS, τον οποίο θα χρειαστείτε στις επόμενες ασκήσεις. Φέτος δεν θα βαθμολογηθεί αυτή η πρώτη σειρά ασκήσεων, άρα δεν έχετε να παραδώσετε τίποτα (αλλά **μην καθυστερήσετε** να την κάνετε, διότι ακολουθούν οι άλλες...).
