ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Οργάνωση Υπολογιστών

Ασκήσεις 1: Εισαγωγή, Καταχωρητές, add/sub, addi, η Γλώσσα Assembly και ο Προσομοιωτής SPIM

Μανόλης Γ.Η. Κατεβαίνης

Τμήμα Επιστήμης Υπολογιστών

Άδειες Χρήσης

•Το παρόν εκπαιδευτικό υλικό υπόκειται στην άδεια χρήσης **Creative Commons** και ειδικότερα
Αναφορά – Μη εμπορική Χρήση – Όχι Παράγωγο Έργο 3.0 Ελλάδα
(Attribution – Non Commercial – Non-derivatives 3.0 Greece)

CC BY-NC-ND 3.0 GR

•Εξαιρείται από την ως άνω άδεια υλικό που περιλαμβάνεται στις διαφάνειες του μαθήματος, και υπόκειται σε άλλου τύπου άδεια χρήσης. Η άδεια χρήσης στην οποία υπόκειται το υλικό αυτό αναφέρεται ρητώς.

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Ασκήσεις 1: Εισαγωγή, Καταχωρητές, add/sub, addi, η Γλώσσα Assembly και ο Προσομοιωτής SPIM

Βιβλίο: Διαβάστε την αρχή του κεφαλαίου 2 (§2.1, το μεγαλύτερο μέρος της §2.2, και την αρχή της §2.3): σελίδες 114-116 και 118-121. (Στις διαλέξεις αναφερθήκαμε λίγο και στον linker και συναφή· αυτά υπάρχουν στην §2.12, σελίδες 182-191, αν και το βιβλίο εκεί πηγαίνει σε περισσότερες λεπτομέρειες απ' όσο αντιστοιχεί σε αυτό το μάθημα --περισσότερο προς την ύλη του HY-255...).

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 και οπτικοί), διεπαφές δικτύου (network interfaces), κλπ.
- **Κεντρική Μνήμη (Main Memory):** εκατομμύρια στοιχεία αποθήκευσης πληροφορίας --κάτι σαν flip-flops αλλά απλούστερα στην κατασκευή-- οργανωμένα σαν πολλές (συνήθως εκατομμύρια και πάνω) "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 μιας τέτοιας γλώσσας μηχανής --του επεξεργαστή MIPS-- θα χρησιμοποιήσουμε σαν παράδειγμα σε αυτό το μάθημα. Αλλες γλώσσες μηχανής στυλ RISC είναι αυτές των SPARC, PowerPC, και ARM. Η σειρά x86 και Pentium της Intel έχει πιο πολύπλοκη γλώσσα μηχανής.

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 MIPS είναι πάντα καταχωρητές γενικού σκοπού (registers) του επεξεργαστή, ή σταθερές ποσότητες, αλλά όχι θέσεις μνήμης. Η CPU του MIPS έχει 32 καταχωρητές των 32 bits καθένας --32 bits είναι το μέγεθος λέξης (word) του MIPS. Πιο σύγχρονα μοντέλα επεξεργαστών, σήμερα, έχουν μέγεθος λέξης 64 bits. Οι εντολές αριθμητικών πράξεων του MIPS 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 MIPS:

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 MIPS είναι η **`add`** που κάνει πρόσθεση ακεραίων. Η εντολή Assembly "`add $23, $16, $18`" διαβάζει τα περιεχόμενα των καταχωρητών υπ'αριθμόν 16 και 18, τα προσθέτει, και γράφει το αποτέλεσμα στον καταχωρητή υπ'αριθμόν 23, δηλαδή $23 := $16 + $18.

- Η εντολή **`sub`** (subtract) είναι παρόμοια αλλά, αντί να προσθέτει, αφαιρεί τον τρίτο τελεστέο από το δεύτερο: η "`sub $23, $16, $18`" γράφει στον καταχωρητή υπ'αριθμόν 23 το αποτέλεσμα της αφαίρεσης $16 - $18.
- Η εντολή **`addi`** (add immediate) είναι παρόμοια της add (πρόσθεση), αλλά ο τρίτος τελεστέος της είναι σταθερός αριθμός αντί καταχωρητή: η "`addi $23, $16, 157`" διαβάζει το περιεχόμενο του καταχωρητή 16, προσθέτει τον αριθμό 157 σε αυτό, και γράφει το αποτέλεσμα στον καταχωρητή 23. Εάν η μεταβλητή *i* βρίσκεται στον καταχωρητή 18, τότε η εκχώρηση *i=i+1* μεταφράζεται σε "`addi $18, $18, 1`".
- Ο **καταχωρητής $0** του MIPS περιέχει πάντα την σταθερή ποσότητα μηδέν, ανεξαρτήτως του τι γράφει κανείς σε αυτόν (οι εγγραφές σε αυτόν αγνοούνται από το hardware). Ετσι, η αρχικοποίηση *i=1* μπορεί να γίνει: "`addi $18, $0, 1`".

Γιά έναν υπολογισμό συνθετότερης αριθμητικής έκφρασης, δείτε το παράδειγμα στη σελ. 121 του βιβλίου. Εκτός από τα ονόματα $0, $1, ..., $31 γιά τους 32 καταχωρητές του MIPS, χρησιμοποιούνται και τα λίγο πιό αφηρημένα ονόματα $s0, $s1, ..., $t0, $t1, ... ανάλογα με την κατηγορία χρήσης που συνήθως επιφυλάσσει σε καθένα τους ο compiler, όπως θα δούμε αργότερα.

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 εντολή "`j label`" (*jump* to label, ή παλαιότερα *goto* label) 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 εντολή "`j label`" φορτώνει τη διεύθυνση `label` στον καταχωρητή PC.

1.4 Ο Προσομοιωτής SPIM:

Προγράμματα γραμμένα σε γλώσσα Assembly του MIPS μπορεί να τα δοκιμάσει κανείς και να παρακολουθήσει πώς τρέχουν χρησιμοποιώντας τον *προσομοιωτή* **SPIM**, γραμμένο από τον James Larus στο Πανεπιστήμιο Wisconsin - Madison, http://spimsimulator.sourceforge.net/ . Οι προσομοιωτές είναι προγράμματα υπολογιστή που προσπαθούν να συμπεριφέρονται όσο πιο παρόμοια γίνεται, από ορισμένες απόψεις, με ένα φυσικό σύστημα. Εν προκειμένω, ο SPIM συμπεριφέρεται σαν ένα τσιπάκι MIPS 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.

Η νεότερη έκδοση του SPIM είναι το **QtSpim**.

- Μπορείτε να διαβάστε τα εγχειρίδια χρήσης του "κλασικού" SPIM "Getting Started with xspim" ή "Getting Starting with PCSpim", και "Getting Started with spim" που περιγράφουν όσα από τον SPIM θα χρειαστείτε προς στιγμήν. Επίσης, μπορείτε να ξεφυλλίστε "στα πεταχτά" το εγχειρίδιο "SPIM Command-Line Options" και τις σελίδες A-40 έως A-49 του Παραρτήματος Α του βιβλίου για να δείτε τι άλλο κάνει ο SPIM. Θα τα βρείτε έντυπα στο http://spimsimulator.sourceforge.net/ xspim.pdf, PCSpim.pdf, spim.pdf, SPIM_command-line.pdf, και HP_AppA.pdf.
- Για τον QtSpim, καλό είναι να διαβάσετε τον οδηγό χρήσης που βρίσκεται στό: http://claws.eng.ua.edu/attachments/166_QtSpim%20Tutorial.pdf

Γιά να τρέξετε τη νεότερη έκδοση του προσομοιωτή, QtSpim:

- Μπορείτε να προμηθευτείτε τον QtSpim από το:

http://sourceforge.net/projects/spimsimulator/files/ - Είναι διαθέσιμος για Linux, Windows, και Mac OSX. Αφού επιλέξετε τη σωστή έκδοση, εκτελέστε την εγκατάσταση της εφαρμογής.

- Αντιγράψτε το αρχείο **`~hy225/spim/trap_handler.s`** στο directory όπου θα τρέξετε τον SPIM, με το ίδιο όνομα --ο SPIM περιμένει να το βρει στο παρών directory όταν ξεκινάει. Το αρχείο αυτό περιέχει τον κώδικα χειρισμού των εξαιρέσεων/διακοπών που χρειάζεται για τη σωστή λειτουργία του υπό προσομοίωση υπολογιστή (κάτι σαν την καρδιά της καρδιάς του λειτουργικού συστήματος). Δείτε παρακάτω και γιά το πώς πρέπει να δηλώσετε το αρχείο αυτό στον QtSpim.
- Αν είστε σε υπολογιστή με λειτουργικό σύστημα Linux/Mac OSX, καλέστε τον QtSpim εκτελώντας στη γραμμή εντολών: **`% qtspim`**
- Αν είστε σε υπολογιστή με λειτουργικό σύστημα Windows, εκτελέστε την εφαρμογή από τη συντόμευση που δημιουργήθηκε έπειτα από την ολοκλήρωση της εγκατάστασής της.
- Από το μενού του QtSpim, επιλέξτε διαδοχικά: Simulator -> Settings -> καρτέλα MIPS. Εκεί, ορίστε για Exception Handler το αρχείο trap_handler.s που αντιγράψατε από τη περιοχή του μαθήματος.

Άσκηση 1.5: Κώδικας Γνωριμίας με τον SPIM

Αντικείμενο της παρούσας άσκησης είναι να γνωριστείτε με τη χρήση του SPIM (και με τη γλώσσα Assembly του MIPS). Για το σκοπό αυτό, **μελετήστε και αντιγράψτε** σε ένα αρχείο (π.χ. "`ask1.s`") τον παρακάτω κώδικα --ή διάφορες παραλλαγές του που προτιμάτε-- και τρέξτε τον στον SPIM.

```
     .text               # program memory:
     .globl main             # label "main" must be global;
                             # bootstrap trap.handler calls main.
main:
                             # registers:  $16: i; $17: j;
     addi   $16, $0,  10     # init. i=10; ($0==0 always)
     addi   $17, $0,  64     # init. j=64; (64 decimal = 40 hex)
     add    $18, $16, $17    # $18 <- i+j = 74 dec = 4a hex
     add    $18, $18, $18    # $18 <- 74+74=148 dec = 94 hex
     add    $18, $18, $17    # $18 <-148+64=212 dec = d4 hex
     addi   $17, $17, -1     # j <- j-1 = 63 dec = 3f hex
     sub    $17, $17, $16    # j <- j-i = 53 dec = 35 hex
     j      main             # jump back to main (infinite loop)
```

- Το κομμάτι κάθε γραμμής μετά το `#` είναι σχόλια.
- Οι γραμμές μετά το `.text` είναι εκτελέσιμος κώδικας (σε αντίθεση με αριθμητικά δεδομένα στη μνήμη, που εδώ δεν έχουμε).
- Η γραμμή `.globl main` λέει στον SPIM να βάλει την ετικέτα (label) `main` στον πίνακα καθολικών (global) συμβόλων. Το `main` είναι εκεί που ο trap.handler θα καλέσει τον κώδικά μας, και πρέπει να έχει αυτό ακριβώς το όνομα (όπως και στην C) και να είναι καθολικό σύμβολο, γιά να μπορεί να το βρίσκει ο linker που θα "συνενώσει" τον trap.handler με τον δικό μας κώδικα.
- Η γραμμή `main:` ορίζει την ετικέτα `main` σαν τη διεύθυνση μνήμης όπου θα τοποθετηθεί αυτό που ακολουθεί ακριβώς μετά --στην περίπτωσή μας η πρώτη εντολή (`addi`) του προγράμματός μας.
- Μετά το `main:` ακολουθούν οκτώ εντολές στη γλώσσα Assembly του MIPS: έξι εντολές πρόσθεσης (`add` ή `addi`), μία εντολή αφαίρεσης (`sub`), και μία εντολή άλματος (`j`).
- Η 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....

Άσκηση 1.6: Τρέξιμο του SPIM

Ξεκινήστε το **QtSpim** με τον τρόπο που είπαμε παραπάνω, στην §1.4. Στην καρτέλα **Text** του QtSpim βλέπετε τα περιεχόμενα της μνήμης του υπό προσομοίωση υπολογιστή, ξεκινώντας από τη διεύθυνση 00400000 δεκαεξαδικό. Πρόκειται για την περιοχή εκείνη της μνήμης (text segments) όπου ο SPIM αποθηκεύει τις εκτελέσιμες **εντολές**, σε αντίθεση με τα δεδομένα στη μνήμη (data segments), που φαίνονται στην καρτέλα **Data** --εμείς δεν θα ασχοληθούμε με δεδομένα στη μνήμη σε αυτή την άσκηση.

Στις διευθύνσεις 00400000 έως και 00400020 (δεκαεξαδικό) υπάρχουν 9 "παράξενες" εντολές που προέρχονται από το αρχείο `trap_handler.s` (η διεύθυνση της κάθε εντολής διαφέρει από αυτήν της προηγούμενης κατά 4 διότι οι εντολές του MIPS έχουν μέγεθος 4 bytes καθεμία). Δεν χρειάζεται να καταλάβετε τι κάνουν αυτές οι εντολές --αρκεί να ξέρετε ότι ρόλος τους είναι να καλέσουν τη διαδικασία (procedure) `main`, περνώντας της σαν παραμέτρους τα γνωστά από την C `argc`, `argv`, και `envp`, και όταν η διαδικασία αυτή επιστρέψει να καλέσουν τη διαδικασία `exit` του λειτουργικού συστήματος (system call). Παρακάτω στο ίδιο παράθυρο, ξεκινώντας από τη διεύθυνση 80000180 (δεκαεξαδικό), υπάρχει ο κώδικας του υπό προσομοίωση πυρήνα (kernel) του λειτουργικού συστήματος, που επίσης προέρχεται από το αρχείο trap_handler.s και που επίσης δεν χρειάζεται να καταλάβετε τι κάνει....

Χρησιμοποιήστε το εικονίδιο "**load file**" από το μενού του QtSpim για να ζητήστε να φορτωθεί το αρχείο με το παραπάνω πρόγραμμα που γράψατε (π.χ. "`ask1.s`"). Οι εντολές του προγράμματός σας προστίθενται στη μνήμη (text segments), ξεκινώντας από τη διεύθυνση 00400024 (δεκαεξαδικό), δηλαδή αμέσως κάτω από τις 9 "παράξενες" εντολές (00400000 έως 00400020) που καλούν το `main`.

Πάνω αριστερά φαίνεται το περιεχόμενο του **μετρητή προγράμματος (PC)**. Όποτε ο SPIM 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 (δηλαδή η εντολή αυτή δεν έχει "εκτελεστεί" ακόμα). Ο SPIM αρχικοποιεί τον PC σε 00400000 (δεκαεξαδικό), δηλαδή στην αρχή των 9 "παράξενων" εντολών που καλούν το "main".

Χρησιμοποιήστε το εικονίδιο "**single step**" από το μενού *Simulator* του QtSpim (συντόμευση F10) για να ζητήστε *single-stepping* του προγράμματός σας, δηλαδή να εκτελούνται μια-μια οι εντολές και να τις βλέπετε. Κάθε φορά που πατάτε το κουμπί "single step", ο SPIM προσομοιώνει την εκτέλεση μιας ακόμα εντολής. Παρακολουθήστε ότι μετά από κάθε τέτοια προσομοίωση ο PC έχει αυξηθεί κατά 4 (επειδή οι εντολές του MIPS έχουν μέγεθος 4 bytes καθεμία), δείχνοντας τώρα στην επόμενη εντολή που **θα** προσομοιωθεί, και η επόμενη αυτή εντολή εμφανίζεται τονισμένη με άλλο χρώμα στην καρτέλα **Text** του QtSpim.

Μετά την εκτέλεση της 6ης εντολής από το trap.handler, δηλαδή της εντολής `jal main` (jump and link) από τη διεύθυνση 00400014 (δεκαεξαδικό), η εκτέλεση πηγαίνει στη **διεύθυνση 00400024** (δεκαεξαδικό), δηλαδή στο `main` --στο δικό μας πρόγραμμα. Μετά την εκτέλεση της εντολής `addi $16, $0, 10` από τη διεύθυνση 00400024, δηλαδή όταν ο PC είναι 00400028, παρατηρήστε ότι ο καταχωρητής R16 (καρτέλα **Int Regs**) αλλάζει τιμή και γίνεται **a** δεκαεξαδικό, δηλαδή 10 δεκαδικό, όπως έπρεπε, αφού σε αυτόν τον καταχωρητή έγραψε η εντολή που μόλις εκτελέστηκε το αποτέλεσμα της πρόσθεσης του καταχωρητή $0 (που περιέχει πάντα μηδέν) με τη σταθερή ποσότητα 10 (δεκαδικό).

Καθώς εκτελείτε μια-μια τις εντολές του προγράμματός μας, παρακολουθήστε πώς αλλάζουν τα

περιεχόμενα **των καταχωρητών R16, R17, και R18**. Θυμηθείτε ότι τα περιεχόμενα αυτά ο SPIM τα δείχνει στο δεκαεξαδικό σύστημα. Όταν η εκτέλεση φτάσει στην εντολή `j main`, παρατηρήστε ότι ο PC παίρνει ξανά την τιμή 00400024 (δεκαεξαδικό), και ξαναρχίζει η εκτέλεση του ίδιου προγράμματός μας, από την αρχή. Επειδή το πρόγραμμά μας είναι ένας άπειρος βρόχος που δεν επικοινωνεί καθόλου με τον έξω κόσμο --δεν γράφει ή διαβάζει τίποτα από την "κονσόλα"-- αποφύγετε να πατήστε το εικονίδιο "run/continue" από το μενού, διότι ο προσομοιωτής "κολλάει", δηλαδή μπαίνει κι αυτός σε άπειρο βρόχο, προσομοιώνοντας συνεχώς το δικό μας άπειρο βρόχο, οπότε μοναδική λύση είναι να τον διακόψετε (πατώντας "stop") ή "σκοτώσετε" με control-C.