Σειρά Ασκήσεων 1

Εισαγωγή, Ο επεξεργαστής και η γλώσσα του.

Σε αυτό το εργαστήριο θα κάνουμε μια εισαγωγή σε έναν απλό επεξεργαστή (RISC), το instruction set του οποίου είναι πολύ απλό, αλλά περιέχει όλα τα απαραίτητα στοιχεία για να μπορεί να λειτουργήσει σε ένα πλήρες σύστημα υπολογιστή. Αυτός ο επεξεργαστής είναι ο MIPS που χρησιμοποιήθηκε μεταξύ άλλων από την NEC, την Nintendo, την Silicon Graphics, και την SONY.
Επιλέξαμε αυτόν τον επεξεργαστή επειδή λόγο της απλότητας του, αλλά και της πληρότητας του σαν επεξεργαστής αποτελεί ένα καλό εκπαιδευτικό εργαλείο. Βασισμένοι σε αυτόν θα μπορέσουμε αργότερα να συζητήσουμε και άλλες αρχιτεκτονικές όπως αυτές των επεξεργαστών των ημερών μας.

Για να αξιοποιήσουμε έναν υπολογιστή πρέπει να μιλήσουμε την γλώσσα του. Οι λέξεις αυτού ονομάζονται εντολές (instructions), και το λεξιλόγιο του *instruction set.* Εδώ θα δούμε αυτό το instruction set ενός πραγματικού υπολογιστή στην μορφή που έχει, από την πλευρά του προγραμματιστή, και από την πλευρά που το αντιλαμβάνεται η μηχανή. Οι γλώσσες των υπολογιστών είναι παρά πολύ πιο απλές από τις γλώσσες των ανθρώπων, και αυτό για λόγους που θα τους συζητήσουμε στο εργαστήριο.
Οι επεξεργαστές καταλαβαίνουν και εκτελούν εντολές από ένα ρεπερτόριο πολύ μικρότερο και απλούστερο από τις γλώσσες υψηλού επιπέδου (HLL --High Level Languages). 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" (*RISC* -- Reduced Instruction Set Computers).
Ένας υπολογιστής βασικά θα πρέπει να μπορεί να εκτελεί αριθμητικές πράξεις. 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, C++) μεταφράζονται σε Assembly από ένα σημαντικά πολυπλοκότερο πρόγραμμα, τον Compiler.
Κάθε εντολή γλώσσας μηχανής αποτελείται από έναν **κώδικα πράξης** (**opcode** --operation code), 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.
Από την γλώσσα assembly του MIPS η εντολή "`add $17, $15, $16`" προσθέτει το περιεχόμενο του καταχωρητή 15, με αυτό του 16, και τοποθετεί το αποτέλεσμα στον καταχωρητή 17. Για λόγους **απλότητας** (Σχεδιαστική αρχή 1: Simplicity favors regularity) ο αριθμός των τελεστών είναι τρεις. Οι δύο τελεστές προσθέτονται και το αποτέλεσμα τοποθετείται σε έναν τρίτο τελεστή. Αντίθετα από τις γλώσσες υψηλού επιπέδου, οι τελεστές των αριθμητικών πράξεων δεν μπορούν να είναι οποιεσδήποτε μεταβλητές. Πρέπει να

είναι από ένα **μικρό** σύνολο από θέσεις μνήμης που καλούνται **registers**. (Σχεδιαστική αρχή 2: Smaller is faster).
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 τελεστής της είναι σταθερός αριθμός αντί καταχωρητή: "`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). Έτσι, η αρχικοποίηση "i=1" μπορεί να γίνει: "`addi $18, $0, 1`".
Για έναν υπολογισμό συνθετότερης αριθμητικής έκφρασης, εκτός από τα ονόματα $0, $1, ..., $31 γιά τους 32 καταχωρητές του MIPS, χρησιμοποιούνται και τα λίγο πιο αφηρημένα ονόματα $s0, $s1, ..., $t0, $t1, ... ανάλογα με την κατηγορία χρήσης που συνήθως επιφυλάσσει σε καθένα τους ο compiler, όπως θα δούμε αργότερα.

Για 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 μία εντολή **μεταφοράς ελέγχου** (CTI - control transfer instruction). 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.

Για τα παραπάνω αλλά και για αυτά που θα ακολουθήσουν θα μιλήσουμε με περισσότερη λεπτομέρεια στο μάθημα της θεωρίας της αρχιτεκτονικής.

Ας δούμε τώρα, αφού κάναμε μια μικρή βουτιά στην καρδιά του υπολογιστή, λίγο πιο σφαιρικά τα πράγματα, από μια πιο μακρινή γωνία.

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 (CPU - Central Processing Unit): περιέχει τα κυκλώματα *ελέγχου (control)* που καθοδηγούν την εκτέλεση των λειτουργιών που θα πούμε πιο κάτω. Επίσης 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)*. Πρόκειται για μια πολύ μικρή μνήμη, μεγέθους συνήθως κάμποσων δεκάδων λέξεων (π.χ. 32 λέξεων).

Μία 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. [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. Ο μεταφραστής (compiler) είναι το πρώτο βασικό πρόγραμμα υπολογιστή που γεννά εντολές υπολογιστή].

Ο υπολογιστής λειτουργεί με τον εξής βασικό **επαναληπτικό** τρόπο. Ο επεξεργαστής διαβάζει μιαν εντολή από τη μνήμη. Στη συνέχεια την αποκωδικοποιεί για να καταλάβει τι λεει, 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 έπ' 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"....

Για να ξέρει ο επεξεργαστής ποια είναι η "επόμενη" εντολή που πρέπει να διαβάσει και εκτελέσει, υπάρχει μέσα του ένας ειδικός καταχωρητής, ο **"Μετρητής Προγράμματος"** (PC - program counter) --ίσως μία σωστότερη ονομασία να ήταν "δείκτης προγράμματος" (program 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.

Ασκήσεις:

Ask1.1.

Μία πράξη που περιέχει τους καταχωρητές x, y, z, στην γλώσσα προγραμματισμού C θα μπορούσε να είναι η εξής:

x = (y + z) ;

Γράψτε το αποτέλεσμα που θα δημιουργούσε ένας C compiler για τον MIPS.

Ask1.2.

Μία πράξη που περιέχει τους καταχωρητές x, y, z, k, στην γλώσσα προγραμματισμού C θα μπορούσε να είναι η εξής:

x = (x + y) - (z + k);

Γράψτε το αποτέλεσμα που θα δημιουργούσε ένας C compiler για τον MIPS.

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://www.cs.wisc.edu/~larus/spim.html . 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.
Για να δείτε τι άλλο κάνει ο spim μπορείτε να ανατρέξετε στο http://www.cs.wisc.edu/~larus/SPIM/cod-appa.pdf

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` λεει 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, για 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, λεει 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, έπ' άπειρο --ένα κανονικό πρόγραμμα, φυσικά, δεν πρέπει να κάνει κάτι τέτοιο, αλλά εδώ δεν πρόκειται για κανονικό πρόγραμμα....

Άσκηση Τρεξίματος του SPIM:

Ξεκινήστε το **`spim`** με τον τρόπο που είπαμε πιο πάνω. Στο **τρίτο** (δηλ. μεσαίο) παράθυρο του SPIM 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 στο τέταρτο παράθυρο --εμείς δεν θα ασχοληθούμε με δεδομένα στη μνήμη σε αυτή την άσκηση. Φαρδύνετε αρκετά το παράθυρο του SPIM για να μπορείτε να βλέπετε ολόκληρες τις εντολές, μαζί με τα σχόλια που τις συνοδεύουν (επίσης, μεγαλώστε κατακόρυφα το τρίτο παράθυρο, κουνώντας το μικρό τετραγωνάκι κάτω δεξιά του).

Στις διευθύνσεις 00400000 έως 0040001c (δεκαεξαδικό) υπάρχουν 8 "παράξενες" εντολές που προέρχονται από το αρχείο trap.handler (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 80000080 (δεκαεξαδικό), υπάρχει ο κώδικας του υπό προσομοίωση πυρήνα (kernel) του λειτουργικού συστήματος, που επίσης προέρχεται από το αρχείο trap.handler και που επίσης δεν χρειάζεται να καταλάβετε τι κάνει....

Χρησιμοποιήστε το κουμπί "**load**" στο δεύτερο παράθυρο του SPIM γιά 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 00400020 (δεκαεξαδικό), δηλαδή αμέσως κάτω από τις 8 "παράξενες" εντολές (00400000 έως 0040001c) που καλούν το "main".

Τρόπος παράδοσης:

Θα παραδώσετε τις ασκήσεις σε χαρτί εκτός αυτήν του spim που θα την παραδώσετε ηλεκτρονικά. Σε αυτήν του Spim θα παραδώσετε ένα στιγμιότυπο της οθόνης καθώς τρέχετε το πρόγραμμα spim, και όταν αυτό βρίσκεται σε ένα 'ενδιαφέρον' ενδιάμεσο σημείο της επιλογής σας. Το στιγμιότυπο θα το πάρετε και θα το παραδώσετε ως εξής:
Αφού βρίσκεστε στο ενδιαφέρον κομμάτι της προσομοίωσης πατήστε το κουμπί 'Print Screen'. Μετά μπορείτε να ανοίξετε το πρόγραμμα Paint των Windows και να κάνετε εκεί 'paste', μετά θα σώστε αυτό το αρχείο με όνομα ask1.jpg.
Έπειτα θα δημιουργήστε ένα φάκελο με όνομα "**tp2108_ασκ1_student_ip**", όπου **student** είναι το επίθετό σας, **ip** είναι ο αριθμός μητρώου σας στη σχολή. Μέσα σε αυτό το φάκελο θα τοποθετήσετε το αρχείο "ask1.jpg".
Την ώρα του επόμενου εργαστηρίου θα ανοίξει ένας ftp server με τον οποίο θα συνδεθείτε και θα μεταφέρετε εκεί το φάκελο με τη δουλειά σας.