

# ΜΥΥ-402 Αρχιτεκτονική Υπολογιστών Εισαγωγή στη γλώσσα της μηχανής

---

Αρης Ευθυμίου

*“I speak Spanish to God,  
Italian to women,  
French to men,  
and German to my horse”, Charles V, King of France*

# Ανακοινώσεις

---

- Εργαστήριο Ηλεκτρονικής (ΜΥΥ404), κος Τσιατούχας
  - εγγραφή μέσω της σελίδας του μαθήματος στο ecourse
  - μέχρι την ερχόμενη Δευτέρα
- Κλινική GitHub σήμερα
  - μετά το μάθημα, στο εργαστήριο 1<sup>ου</sup> ορόφου
  - το εργαστήριο κλειδώνει στις 21:00!
- Στο ecourse υπάρχει «Αρχιτεκτονική (Ανοιχτά Μαθήματα)»
  - **Δεν είναι το σωστό μάθημα!**



# Το σημερινό μάθημα

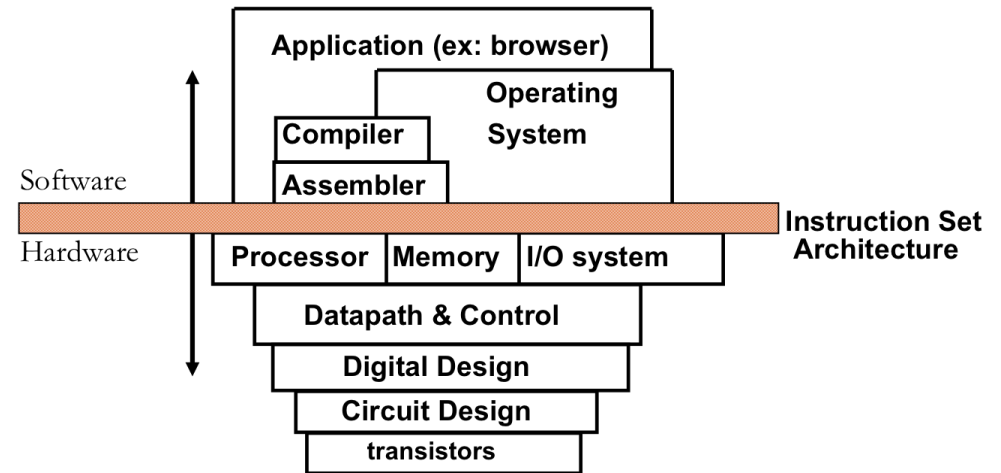
---

- Εισαγωγή στη γλώσσα του υπολογιστή (MIPS)
- Καταχωρητές
  - οι «μεταβλητές» του υπολογιστή
- Εντολές αριθμητικών πράξεων
- Εντολές μεταφοράς δεδομένων
- Τελεσταίοι



# Γλώσσα μηχανής, assembly

- Για να κατευθύνεις το υλικό ενός υπολογιστή, πρέπει να γνωρίζεις τη γλώσσα του
- Οι λέξεις της λέγονται εντολές (**instructions**)
- Το «λεξιλόγιο» ονομάζεται αρχιτεκτονική συνόλου εντολών
  - Instruction Set Architecture (**ISA**)
- Διαφορετικοί επεξεργαστές έχουν διαφορετικά ISAs
  - αλλά με σημαντικές ομοιότητες: μαθαίνοντας μία ISA, εύκολα μαθαίνεις και άλλες



# MIPS

---

- Θα χρησιμοποιήσουμε τον MIPS
  - απλός, τυπικός RISC επεξεργαστής
    - όταν δείτε τον x86 θα καταλάβετε!
  - τον χρησιμοποιεί το βασικό σύγγραμμα!
- Πού βρίσκεται;
  - στο 75% των Blu-ray disc players
  - σε πολλές ψηφιακές τηλεοράσεις και αποκωδικοποιητές
  - στον file server (NAS) του εργαστηρίου υλικού
- Ανήκει πλέον στην Imagination technologies
  - γνωστή για τα PowerVR, GPUs



# Πρόσθεση

---

- Κάθε επεξεργαστής κάνει αριθμητικές πράξεις
  - η πρόσθεση είναι η θεμελιώδης αριθμητική πράξη

- Στον MIPS: σχόλιο

add a, b, c    # a = b + c

- δίνει εντολή να προστεθούν οι μεταβλητές b, c και το αποτέλεσμα να εγγραφεί στην μεταβλητή a



# Σύνταξη εντολών

---

πράξη    προορισμός, 1<sup>η</sup> πηγή, 2<sup>η</sup> πηγή

- Η μορφή των εντολών είναι άκαμπτη (rigid):
  - πάντα μία πράξη, 3 τελεσταίοι (**operands**)
  - 1<sup>ος</sup> τελεσταίος είναι ο προορισμός (**destination** operand)
  - 2<sup>ος</sup> τελεσταίος είναι η 1<sup>η</sup> πηγή (1<sup>st</sup> **source** operand)
  - 3<sup>ος</sup> τελεσταίος είναι η 2<sup>η</sup> πηγή (2<sup>nd</sup> source operand)
  - η σειρά των πηγών έχει σημασία για πράξεις όπως η αφαίρεση
- Γιατί η ακαμψία;
  - η ομοιομορφία απλοποιεί την υλοποίηση



# Πράξεις με πολλές μεταβλητές

---

- Πώς μπορώ να υπολογίσω  $f = a+b+c+d$ ;

```
add  f, a, b      # f = a+b
add  f, f, c      # f = f+c, (a+b+c)
add  f, f, d      # f = f+d, (a+b+c+d)
```

Συνθέτω σύνθετες πράξεις με ακολουθίες απλών εντολών





# ISA και αριθμός τελεστών

- Με τρεις τελεστές υπάρχει ευελιξία για οποιοδήποτε υπολογισμό:
  - χρήση οποιουδήποτε καταχωρητή ως πηγή ή προορισμό
- Μπορώ να κάνω το ίδιο με 2 τελεστές;
  - Ναι. Χρησιμοποιείται σε μερικές άλλες ISA (π.χ. x86)  
π.χ.: `add a, b    # a = a + b`
  - Χάνεται μέρος της ευελιξίας: αλλάζω την τιμή μιας πηγής
  - Όχι στον MIPS!
- Αν όλες οι εντολές είχαν 4 τελεστές
  - κερδίζω: πράξεις με 3 πηγές σε μία εντολή
  - χάνω: δυσκολία στην κωδικοποίηση εντολών (σε επόμενο μάθημα)
  - πόσο χρήσιμο/συχνό είναι: όχι πολύ



# Τελεσταίοι (operands)

---

- Οι εντολές κάνουν πράξεις με δεδομένα αποθηκευμένα σε μεταβλητές
  - οι τελεσταίοι - operands
- Στην assembly χρησιμοποιούνται κυρίως οι καταχωρητές (registers)
  - μικρές μνήμες μέσα στον επεξεργαστή
  - Οι περισσότερες εντολές έχουν **μόνο** καταχωρητές ως τελεσταίους
- Γιατί;
  - εξαιρετικά γρήγοροι (Αρχή #3: τοπικότητα αναφορών)



# Καταχωρητές στον MIPS

---

- Προκαθορισμένος αριθμός καταχωρητών
  - κατασκευασμένοι στο υλικό. Δεν αλλάζουν!
- Ο MIPS έχει 32 καταχωρητές
  - Αρχή σχεδίασης: το μικρό είναι γρήγορο
- Στον MIPS του μαθήματος, κάθε καταχωρητής έχει «πλάτος» 32 bit
  - 32bit ονομάζονται «λέξη» (**word**)
  - το βασικό μέγεθος με το οποίο γίνονται πράξεις απευθείας σε υλικό



# Καταχωρητές

---

- Για να αναφερθούμε σε έναν καταχωρητή χρειάζεται
  - είτε ο αριθμός του, από 0 έως 31
  - είτε το όνομά του
- Στη γλώσσα assembly του MIPS αναφορές σε καταχωρητή γίνονται δίνοντας το σύμβολο \$ και μετά το όνομα (ή τον αριθμό)
  - για να ξεχωρίζουν από τους απλούς αριθμούς
- Παράδειγμα:

```
add $8, $2, $3
```



# Ονόματα καταχωρητών

---

- Προτιμώνται από τους απλούς αριθμούς
- Υπάρχουν κάποιες συμβάσεις στη χρήση των καταχωρητών
  - που αποτυπώνονται στα ονόματά τους
- Για την ώρα:
  - \$t0 - \$t9 προσωρινές τιμές (temporary)
  - \$s0 - \$s7



# Παράδειγμα

---

- Γράψτε πρόγραμμα σε assembly που υπολογίζει
$$f = (g + h) - (i + j)$$
- Αντιστοιχία μεταβλητών-καταχωρητών
  - $g - s0, h - s1, i - s2, j - s3, f - s4$

```
add    $t0, $s0, $s1
add    $t1, $s2, $s3
sub     $s4, $t0, $t1
```

- Νέα αριθμητική εντολή: sub – αφαίρεση



# Σταθερές

---

- Αγγλικό όρος (ειδικά για assembly): **immediate**
  - άμεσα διαθέσιμες τιμές – δεν χρειάζεται καν ανάγνωση καταχωρητή
- Σταθερές χρησιμοποιούνται πολύ συχνά σε προγράμματα
- MIPS: εντολές με **μία** σταθερά αντί για καταχωρητή
  - δεν είναι απαραίτητες. μπορούν να υλοποιηθούν με εντολές
    - π.χ. για υπολογισμό 0: `sub $t0, $t0, $t0`
  - τόσο συχνές που αξίζει να έχουν δική τους εντολή
  - αρχή: «κάνε γρήγορη τη συχνή περίπτωση»
  - αλλιώς θα χρειαζόταν τουλάχιστον 2 εντολές

```
addi $t0, $t0, 1    # t0 = t0 + 1
```



# Σταθερές

---

- Δεν υπάρχει `subi`. Γιατί;
  - μπορεί να γίνει εύκολα με την `addi`. π.χ. `addi $t0, $t0, -1`
  - οι εντολές είναι «πολύτιμες», δεν προσθέτουμε εντολές στην ISA αν δεν υπάρχει σοβαρός λόγος
- Η σταθερά είναι πάντα ο δεύτερος τελεσταίος πηγής
- Σε γλώσσα `assembly` μπορούμε να χρησιμοποιήσουμε δεκαεξαδική αναπαράσταση ή ακόμα και χαρακτήρες
  - `addi $t0, $t0, 0x10`
  - `addi $t0, $t0, 'a'`





# Καταχωρητής μηδέν

---

- Η τιμή μηδέν είναι εξαιρετικά χρήσιμη/συχνή
- Ο MIPS καλωδιώνει τον καταχωρητή μηδέν (\$0, **\$zero**) στην τιμή 0
  - ανάγνωση πάντα 0, εγγραφές δεν τον επηρεάζουν
- Αντιγραφή/μεταφορά καταχωρητή
  - add \$s0, \$s1, \$zero # s0 = s1
- Το παρακάτω δεν κάνει τίποτα!
  - add \$zero, \$s1, \$t1
  - μερικές φορές χρειάζεται! Ονομάζεται **no-op** (no operation)
    - ή nop



# Υπερχείλιση

---

- Οι υπολογιστές χρησιμοποιούν αριθμητική με σταθερό «μήκος» bit
  - όταν το αποτέλεσμα δεν μπορεί να παρασταθεί με τον διαθέσιμο αριθμό bits, συμβαίνει υπερχείλιση (overflow)
- Κάποιες γλώσσες προγραμματισμού την αγνοούν (C), ενώ άλλες την ανιχνεύουν και διακόπτουν το πρόγραμμα
- Ο MIPS έχει 2 είδη αριθμητικών εντολών για τις δύο περιπτώσεις



# Υπερχείλιση στον MIPS

---

- Εντολές που προκαλούν λάθος αν γίνει υπερχείλιση:
  - add
  - addi
  - sub
- Εντολές που **δέν** προκαλούν λάθος (u – unsigned):
  - addu
  - addiu
  - subu
- Ο compiler της κάθε γλώσσας επιλέγει το είδος εντολής που χρειάζεται



# Προσπέλαση μνήμης

---

- Οι καταχωρητές δεν αρκούν
  - είναι λίγοι
  - πίνακες (array), αντικείμενα (object), ...
- Τα περισσότερα δεδομένα είναι αποθηκευμένα στη μνήμη
- Αλλά ο MIPS εκτελεί πράξεις μόνο με καταχωρητές
  - και άλλοι RISC επεξεργαστές
- Εντολές μεταφοράς από/προς τη μνήμη
  - Αγγλικός data transfer instructions



# Μεταφορά δεδομένων

---

- Δύο είδη
  - από τη μνήμη σε καταχωρητή: φόρτωμα **load**
  - από καταχωρητή στη μνήμη: αποθήκευση **store**
- Δύο πληροφορίες χρειάζονται
  - το όνομα/αριθμός του καταχωρητή
  - η διεύθυνση της μνήμης



# Σχηματισμός διεύθυνσης

---

- Στον MIPS η διεύθυνση μνήμης υπολογίζεται από
  - έναν καταχωρητή (η τιμή του ονομάζεται **base address**)
  - μία σταθερά (ονομάζεται **offset**)
  - η διεύθυνση (όνομα **effective address**) είναι το άθροισμα των τιμών των παραπάνω
- Συμβολισμός: σταθερά (όνομα καταχωρητή)
  - π.χ. 8(\$s3)
- Δεν μπορούν να χρησιμοποιηθούν 2 καταχωρητές
  - μπορεί να γίνει με επιπλέον εντολές μόνο



# Σύνταξη εντολών μεταφοράς

πράξη    1<sup>ος</sup> καταχ., σταθερά(καταχ. βάσης)

- Πράξη:
  - lw (load word) – ανάγνωση λέξης 32b από τη μνήμη
  - sw (store word) – εγγραφή λέξης 32b στη μνήμη
- 1<sup>ος</sup> καταχωρητής
  - για load, εκεί θα αποθηκευτεί η τιμή
  - για store, από εκεί θα διαβαστεί η τιμή
  - **διαφορετική σημασία ανάλογα με την εντολή!**
- Παράδειγμα:
  - lw \$s0, 0xc(\$t1)    # s0 = Mem[\$t1+12]



# Διευθύνσεις μνήμης

---

- Η μνήμη θεωρείται ότι είναι ένας τεράστιος πίνακας
- Ο MIPS εσωτερικά δουλεύει με λέξεις (word) 32 bit
- Αλλά χρειαζόμαστε τη δυνατότητα να διαβάζουμε/γράφουμε bytes (8bit)
  - για «μικρά» δεδομένα: χαρακτήρες, boolean, ...
- Συμπεράσματα
  - οι διευθύνσεις μνήμης είναι σε bytes
    - αλλιώς: κάθε byte της μνήμης έχει ξεχωριστή διεύθυνση
  - υπάρχει δυνατότητα απευθείας προσπέλασης και για bytes και για λέξεις





# Μνήμη: προσπέλαση λέξης

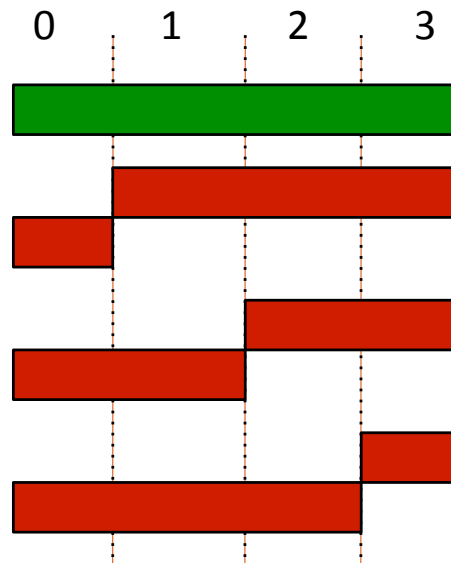
---

- Ποιά είναι η διεύθυνση μιας λέξης στη μνήμη;
  - η διεύθυνση του πρώτου byte της λέξης
- Εστω μία λέξη στη διεύθυνση A. Ποιά είναι η διεύθυνση της επόμενης λέξης;
  - $A + 4$
  - όχι  $A + 1$ ! Πολύ συνηθισμένο λάθος.



# Μνήμη: ευθυγράμμιση

- Μπορώ να βάζω λέξεις σε όποια διεύθυνση μνήμης θέλω;
  - σε μερικούς υπολογιστές, ναι
- Στον MIPS, όχι
  - πρέπει να είναι **ευθυγραμμισμένες (aligned)**
  - οι διευθύνσεις πρέπει να είναι πολλαπλάσιες του 4



# Υπολογισμός διεύθυνσης

---

- Μετάφραστε σε assembly
  - $g = h + a[8]$
  - $g$  - \$s1,  $h$  - \$s2,  $a[]$  array λέξεων με βάση ( $a[0]$ ) στον \$s3
- Χρειάζεται χωριστή εντολή για μεταφορά από μνήμη
- Ποιό είναι το offset;
  - $8 \text{ (θέση πίνακα)} \times 4 \text{ (bytes ανά θέση)} = 32_{\text{ten}}$
  - Το  $a[8]$  βρίσκεται στη θέση  $\$s3 + 32$
- Απάντηση:

```
lw    $t0, 32($s3)    # $t0 = a[8]
add   $s1, $s2, $t0
```



# Δείκτες μνήμης

- Πολύ συχνά χρησιμοποιούμε καταχωρητές που «δείχνουν» σε διευθύνσεις μνήμης
  - λέγονται **δείκτες**, pointers
- Στο προηγούμενο παράδειγμα, αν η εντολή ήταν σε loop
  - $g = g + a[i]$
  - δεν μπορούμε να χρησιμοποιήσουμε σταθερά γιατί το offset μεταβάλλεται

```
add  $t1, $s3, $zero  # $t1 points to a[0]
...
addi $t1, $t1, 4      # $t1 points to next
                        # element of a[]
lw   $s0, 0($t1)      # Note offset is 0
```



# Δείκτες και δεδομένα


---

- Ενας καταχωρητής κρατά οποιαδήποτε τιμή 32 bit
  - μπορεί να είναι δείκτης στη μνήμη
  - μπορεί να είναι ο αριθμός των φοιτητών στην τάξη
- Και στις δύο περιπτώσεις έχουν νόημα αριθμητικές πράξεις
  - διαφορετικό νόημα βέβαια!
- Αλλά προσοχή: μη κάνετε προσπάθεια μνήμης χρησιμοποιώντας ως δείκτη καταχωρητή που περιέχει δεδομένα!
  - Ο `s0` πρέπει να είναι δείκτης: `lw $t0, 0($s0)`
  - δεν υπάρχει κάτι (π.χ. Java interpreter) να πιάσει το λάθος



# Μεταφορά byte

---

- Αποθήκευση ενός byte στη μνήμη. π.χ.
  - `sb $t0, 12($s1) # Mem[$s1+12] = $t0`
  - Αλλά ο `$t0` είναι 32 bit. Τι μεταφέρεται;
  - Τα 8 λιγότερο σημαντικά bits του `$t0` 
- Φόρτωση byte σε καταχωρητή
  - τι κάνουμε με τα 24 περισσότερα σημαντικά bits;
- Δύο παραλλαγές για φόρτωση:
  - `lb`, επέκταση προσήμου (**sign-extension**) στα 24 msb
    - χρήσιμο όταν το byte είναι αριθμός με πρόσημο
  - `lbu`, συμπλήρωση με μηδενικά (zero-fill)
    - `load byte unsigned`



# Επέκταση προσήμου

---



- Το πιο σημαντικό bit του byte (s)
- αντιγράφεται προς τα αριστερά μέχρι να γεμίσει η λέξη



# Ετικέτες, οδηγίες assembler

- Στον προγραμματισμό σε assembly αναφερόμαστε σε δεδομένα (και εντολές) χρησιμοποιώντας ετικέτες (**labels**) αντί για διευθύνσεις
  - βοήθημα για εμάς, η μηχανή χρησιμοποιεί μόνο διευθύνσεις
- Επίσης για βοηθητικές δουλειές όπως να βάλουμε τιμές σε θέσεις μνήμης, δίνουμε οδηγίες (**directives**)
- Βλ. εργαστήριο 1

labels ↗  
↘

```
number: .word 0  
next:   addi $t1, $t1, 8
```

directive ↙





# Ψευτοεντολή la

---

- Όταν θέλουμε να δώσουμε τη διεύθυνση κάποιου δεδομένου σε ένα καταχωρητή πρέπει να μετατρέψουμε την ετικέτα σε διεύθυνση
- `la $reg, label`
  - η ετικέτα δίνεται χωρίς το :
- Αυτή είναι μια **ψευτοεντολή (pseudo-instruction)**
  - μετατρέπεται σε 1-2 πραγματικές εντολές
  - δυστυχώς ονομάζεται load address, **αλλά δεν μεταφέρεται τίποτα από τη μνήμη**
  - συχνά αναφέρονται και ως macro-εντολές
- Υπάρχουν πολλές ψευτοεντολές MIPS
  - γενικά απαγορεύεται να χρησιμοποιηθούν στο εργαστήριο
  - εκτός της `la`



# Σύνοψη – MIPS

---

- Αριθμητικές εντολές, εντολές μεταφοράς δεδομένων
  - add, sub, addu, subu, addi, addiu, lw, sw, lb, lbu, sb
- Τελεσταίοι
  - καταχωρητές, σταθερές (immediates), μνήμη
- Διευθυνσιοδότηση μνήμης
  - ευθυγράμμιση
  - υπολογισμός offset
- Δείκτες μνήμης
- Ετικέτες, οδηγίες assembler
- Ψευτοεντολή la



# Εργαστηριακή άσκηση 1

---

- Η 1<sup>η</sup> εργαστηριακή άσκηση τρέχει (βλ ecourse)
  - παράδοση μέχρι την Τρίτη 23/2
  - ώρα 23.00
- Θα διαβάζω τα αποθετήρια με τις ασκήσεις αυτόματα
  - λίγο μετά την ώρα παράδοσης
  - **δεν θα δεχθώ ότι κάποιος ξέχασε να κάνει push** αλλά έχει εμπρόθεσμο στιγμιότυπο στο τοπικό αποθετήριό του
    - οι ημερομηνίες στο git δεν είναι αξιόπιστες
  - δείτε 1<sup>ο</sup> μάθημα για τα ημερήσια πάσα - παράταση
- Η φόρμα για το GitHub αποσύρεται την Παρασκευή
  - ολοκληρώστε εγγραφή εργαστηρίου άμεσα



# Επόμενο μάθημα

---

MIPS assembly – Συνέχεια

Εντολές αλλαγής ροής προγράμματος,  
διακλαδώσεις, άλματα, επαναλήψεις

