ΜΥΥ-402 Αρχιτεκτονική Υπολογιστών
Αναπαράσταση εντολών

Αρης Ευθυμίου

Ανακοινώσεις

- Ασκηση 2 στο ecourse , GitHub
  - Παράδοση: Τρίτη 1η Μάρτη
- Λάθη στη δομή καταλόγων, ονόματα αρχείων:
  - πρέπει να είναι ακριβώς όπως στο labXX_starter

```
GitHubUsername_labs/
        lab01/
            lab01.asm
            test/
                ...
```

  - όχι κεφαλαίο αρχικό γράμμα, tester αντί για test, …
- Βραδινή κλινική για git/GitHub μετά το μάθημα...

Συγκρίσεις ανισότητας

- Μερικές φορές δεν αρκεί έλεγχος ισότητας ή ανισότητας
  - αν και είναι ο πιο συνηθισμένος τύπος ελέγχου
- Χρειάζεται να ελεγθεί αν ένας αριθμός είναι μικρότερος
- Εντολή MIPS slt – set on less than
  - μη την μπερδεύετε με τις εντολές ολίσθησης (sll, srl)
  - Σύνταξη: `slt rdest, rsrc1, rsrc2`

    Αν τιμή του rsc1 < τιμή rsrc2,

    rdest ← 1,

    αλλιώς rdest ← 0

Διακλάδωση με συνθήκη <

Μετατροπή σε assembly:

```
if (g >= h)
   εντολή 1
εντολή 2
```

g→ $s0, h → $s1

```
        slt   $t0, $s0, $s1     # t0 = (g < h)
        bne   $t0, $zero, less  # if (t0==1) goto less
        εντολή 1
less:
        εντολή 2
```

Συνθήκες ανισότητας

- Δεν υπάρχει εντολή set if greater than
  - απλά αλλάζει η σειρά των καταχωρητών

```
if (g < h)
   εντολή 1
εντολή 2
```

1

```
      if (g >= h) goto skip
      εντολή 1
skip:
      εντολή 2
```

2

```
      if not (g < h) goto skip
      εντολή 1
skip:
      εντολή 2
```

3

```
        slt   $t0, $s0, $s1    # t0 = (g < h)
        beq   $t0, $zero, geq  # if (t0==0) goto geq
        εντολή 1
geq:
        εντολή 2
```

Ψευτοεντολές διακλάδωσης

- Υπάρχουν ψευτο-εντολές σύγκρισης-διακλάδωσης
  - blt – branch if less than
  - ble – branch if less or equal
  - …
- Μην τις χρησιμοποιείτε στις ασκήσεις!
  - η παραλλαγή του Mars του μαθήματος, δεν τις επιτρέπει
  - είναι σημαντικό να κατανοήσετε ότι με μία εντολή σύγκρισης και τις δύο παραλλαγές της branch μπορούν να γίνουν τα πάντα
  - επίσης σημαντικό να σκέφτεται κανείς με «αρνητική λογική»

Σταθερές σε συγκρίσεις

- Πολύ συχνά χρειάζονται συγκρίσεις με σταθερές
- Εντολή: `slti rdest, rsrc1, immediate`

- Προσοχή: η σταθερά είναι πάντα η **δεύτερη πηγή**
  - π.χ. `slti $t0, $s0, 5   #  t0 = s0 < 5`

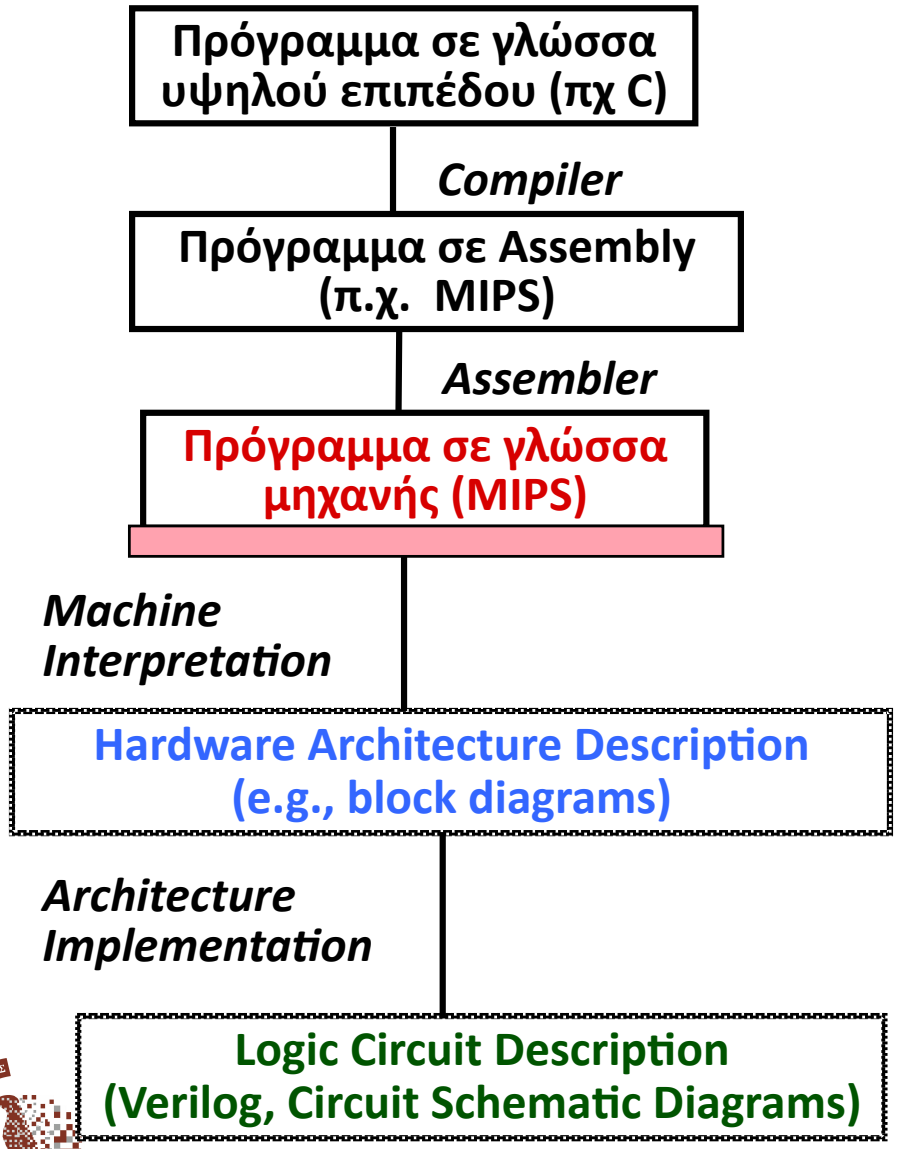
Συγκρίσεις με απρόσημους

- Οι εντολές που είδαμε είναι για ακέραιους με πρόσημο
  - συμπλήρωμα ως προς 2
- Δεν δουλεύουν σωστά με απρόσημους (θετικούς) αριθμούς
  - $\texttt{ffffffff}_{\text{hex}}$ < $\texttt{00000001}_{\text{hex}}$, γιατί -1 < 1
  - αλλά αν είναι θετικοί, $\texttt{ffffffff}_{\text{hex}}$ > $\texttt{00000001}_{\text{hex}}$
- Παραλλαγές εντολών με u στο τέλος:
  - `sltu`, `sltiu`

Το σημερινό μάθημα

- Η ιδέα του αποθηκευμένου προγράμματος
- Η γλώσσα μηχανής του MIPS
- Εντολές μορφής R
  - πεδία, χρήση καθενός
- Εντολές μορφής I
  - πεδία
  - διευθύνσεις για διακλαδώσεις
  - σχηματισμός μεγάλων σταθερών (εντολή lui)
- Εντολές μορφής J

Γλώσσα μηχανής

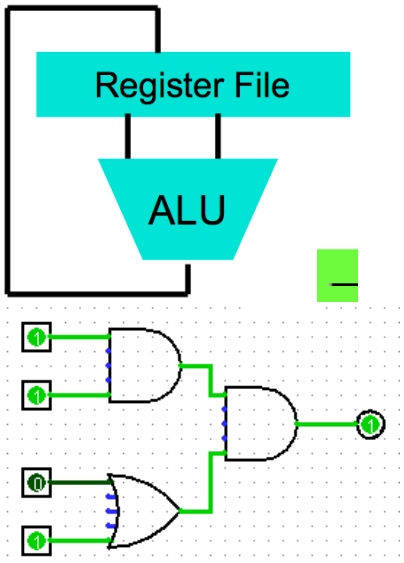



```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

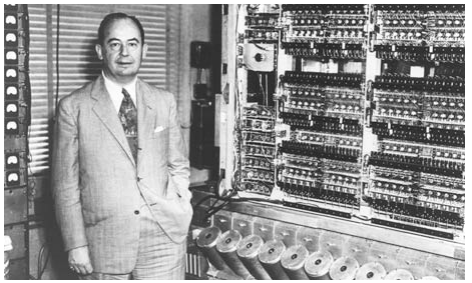
Τα **πάντα** αναπαριστώνται ως αριθμοί, (δεδομένα, εντολές)



Πηγή: CS61C, UCB

Stored-program computer

- Από τις σημαντικότερες ιδέες στην ιστορία των υπολογιστών
  - Πιστώνεται στον John von Neumann
- Οι εντολές αναπαριστώνται ως σειρές από bits
  - μπορεί να τις θεωρήσει κανείς «αριθμούς»
- Τα προγράμματα αποθηκεύονται στη μνήμη
  - διαβάζονται, γράφονται όπως και τα δεδομένα
- Ο υπολογιστής μπορεί να ξαναπρογραμματιστεί σε δευτερόλεπτα
  - παλιότερα έπρεπε να αλλάξει η καλωδίωση. Δουλειά ημερών!



Τα «πάντα» έχουν διεύθυνση

- Πρόγραμμα και δεδομένα αποθηκεύονται στη μνήμη
  - άρα έχουν διευθύνσεις
- Προγράμματα:
  - οι εντολές διακλάδωσης δείχνουν το στόχο δίνοντας τη διεύθυνση της επόμενης εντολής προς εκτέλεση
    - αν η συνθήκη είναι αληθινή
- Δεδομένα:
  - δείκτες στη μνήμη
- Ειδικός καταχωρητής: διεύθυνση τρέχουσας εντολής
  - λέγεται program counter, PC
    - για ιστορικούς λόγους
  - η Intel τον ονομάζει instruction pointer, IP

Binary compatibility

- Διανομή προγραμμάτων σε κώδικα μηχανής
  - δεν χρειάζεται ο χρήστης να κάνει μεταγλώτιση
- Τα προγράμματα δεσμεύονται από την ISA
  - δεν τρέχουν σε διαφορετικό επεξεργαστή
- Καινούριοι επεξεργαστές: πρέπει να τρέχουν παλιά προγράμματα (binaries)
  - αλλά και καινούρια που εκμεταλεύονται νέες εντολές
- Σύνολα εντολών με συμβατότητα προς τα πίσω (backwards compatible instruction sets)
  - εντολές του 8086 (κατασκευή 1981) υποστηρίζονται ακόμη

Εντολές ώς «αριθμοί»

- Στον MIPS οι εντολές αναπαριστώνται ώς λέξεις των 32 bit
  - το ίδιο και στους περισσότερους RISC επεξεργαστές
  - απλουστεύει την υλοποίηση
- Οι λέξεις εντολών χωρίζονται σε πεδία (fields)
  - κάθε πεδίο περιέχει μια πληροφορία σχετική με την εντολή
- Στον MIPS υπάρχουν 3 πρότυπα με τα οποία κωδικοποιούνται οι εντολές (instruction format types)
  - θα μπορούσε κάθε εντολή να έχει διαφορετική μορφή
  - αλλά για απλοποίηση ακολουθούν ένα από τα 3 πρότυπα

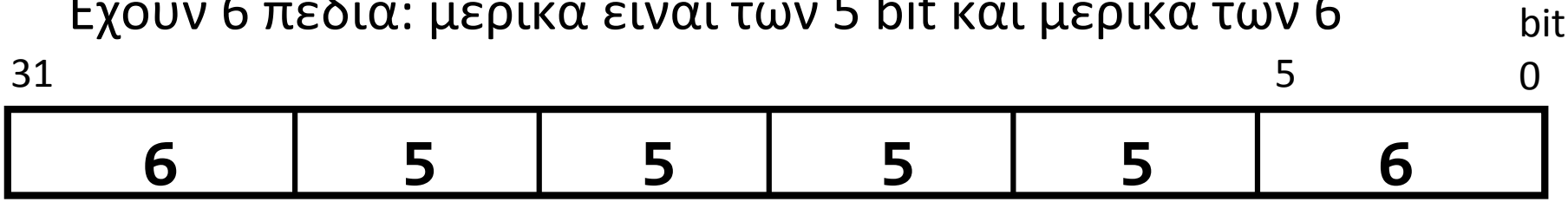
Instruction formats

- I-format: για εντολές με σταθερές (immediates), μεταφορά δεδομένων, και διακλαδώσεις
  - π.χ. **`lw`** and **`sw`** (το offset είναι στην ουσία μια σταθερά)
  - αλλά όχι για τις εντολές ολίσθησης
- J-format: για την εντολή **`j`**
- R-format: για όλες τις υπόλοιπες

- Σε λίγο θα ξεκαθαρίσει γιατί χωρίζονται οι εντολές με αυτό τον τρόπο

Εντολές τύπου R

Εχουν 6 πεδία: μερικά είναι των 5 bit και μερικά των 6



Για την ευκολία μας, τα πεδία έχουν ονόματα:

opcode	rs	rt	rd	shamt	funct

Κάθε πεδίο θα θεωρείται ένας απρόσημος ακέραιος αριθμός των 5, 6 bit

Πεδία 5 bit: δυνατές τιμές 0 – 31

Πεδία 6 bit: δυνατές τιμές 0 - 63

Εντολές τύπου R

- Πεδίο opcode
  - καθορίζει, μερικώς, την εντολή
  - πάντα 0 για εντολές τύπου R
- Πεδίο funct
  - μαζί με το opcode καθορίζει πλήρως την εντολή
  - διαφορετικός κωδικός για κάθε εντολή, π.χ. 36_{ten} (0x24) - and
- Γιατί δεν συνδιάζονται σε ένα πεδίο με περισσότερα bits;
  - θα δούμε σε λίγο...
- Πώς μπορώ να βρώ τους κωδικούς;
  - στην πράσινη κάρτα του βιβλίου
  - υπάρχει link πάνω δεξιά στο ecourse

R-format, πεδία καταχωρητών

- **`rs`** (Source Register)
  - ο αριθμός καταχωρητή του πρώτου τελεσταίου πηγής
- **`rt`** (Target Register)
  - ο αριθμός καταχωρητή του δεύτερου τελεσταίου πηγής
  - το όνομα είναι λίγο παραπλανητικό
- **`rd`** (Destination Register)
  - ο αριθμός καταχωρητή που θα δεχθεί το αποτέλεσμα (καταχωρητής/τελεσταίος προορισμού)
- Τα πεδία είναι 5 bit το καθένα
  - κατάλληλα για να καθορίσουν έναν καταχωρητή MIPS (0-31)
- Υπάρχουν λίγες εντολές που χρησιμοποιούν τα πεδία διαφορετικά

R-format, πεδίο shamt

- **`shamt`** (shift amount)
  - Αριθμός ολίσθησης (σταθερά)
  - Δεν έχει νόημα ολίσθηση μιας 32 bit λέξης για περισσότερες από 31 θέσεις. 5 bit είναι αρκετά
- Είναι πάντα 0 για όλες τις εντολές εκτός από ολισθήσεις

Παράδειγμα 1/2

Κωδικοποίηση: `add $t0,$s1,$s2`

Οι κωδικοί βρίσκονται στην πράσινη κάρτα MIPS

- `opcode` = 0
- `funct` = 32
- `rd` = προορισμός: `$t0` είναι ο καταχωρητής 8
- `rs` = πηγή 1η: `$s1` είναι ο 17
- `rt` = πηγή 2η: `$s2` είναι ο 18
- `shamt` = 0. Δεν είναι ολίσθηση

Παράδειγμα 2/2

```
add $t0,$s1,$s2        add $8,$17,$18
```

στήλη basic στον Mars

Κωδικοποίηση με τα πεδία σε δεκαδικό σύστημα:

0	17	18	8	0	32

Κωδικοποίηση με τα πεδία σε δυαδικό σύστημα:

000000	10001	10010	01000	00000	100000

Ολόκληρη η λέξη εντολής σε δεκαεξαδικό (γλώσσα μηχανής):

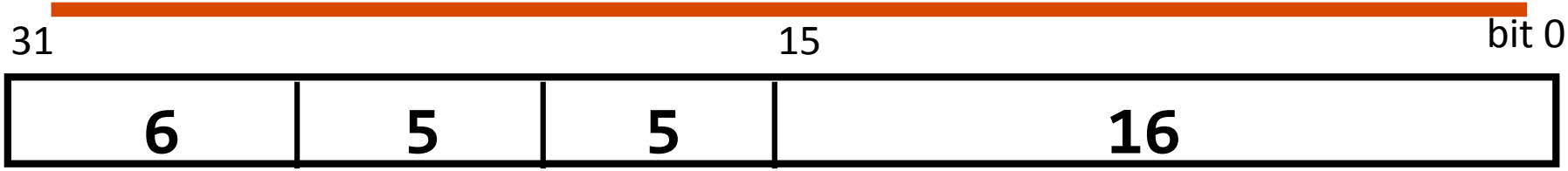
$0232\ 4020_{hex}$

στήλη Code στον Mars

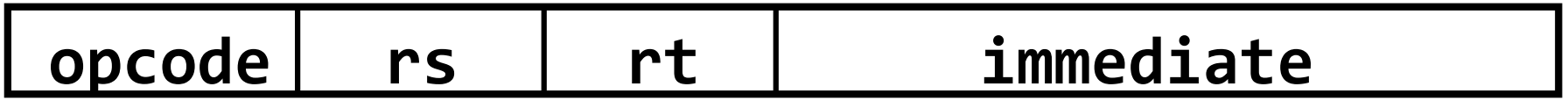
Εντολές τύπου I

- Για εντολές με σταθερές – (I – immediate)
  - Οι εντολες τύπου R έχουν χώρο για μικρές σταθερές (shamt)
  - αλλά τα 5 bit είναι πολύ λίγα για το εύρος τιμών που χρειάζονται
- Συνάδει μερικώς με τον τύπο R
  - κάποια πεδία είναι ίδια
- Παρατήρηση: οι εντολές που έχουν σταθερές χρησιμοποιούν το πολύ 2 καταχωρητές
- Εντολές I-format:
  - addi, beq, lw, sw, …

Πεδία εντολών τύπου I



Πάλι τα πεδία έχουν ονόματα:



Μόνο 1 πεδίο (immediate) είναι διαφορετικό από τη μορφή-R

Το opcode βρίσκεται στην ίδια θέση και έχει ίδιο «πλάτος»

Οι εντολές R, ξεχωρίζουν από το 0 στο opcode!

Με το πεδίο funct (R-type), υπάρχουν περισσότερες επιλογές για κωδικοποίηση εντολών τύπου I

Εντολές τύπου I

- **`rs`**
  - ο αριθμός καταχωρητή του πρώτου τελεσταίου πηγής
- **`rt`**
  - καταχωρητής προορισμού (*target* register)
    - εντολές addi, lw
  - 2ος καταχωρητής πηγής για άλλες εντολές
    - sw, beq,

Πεδίο immediate

- Με 16 bit μπορεί να αναπαρασταθούν ως 2^{16} τιμές
  - αρκετά μεγάλος εύρος για offset σε lw, sw ή για σύγκριση, slti
- Η σταθερά μετατρέπεται σε 32 bit ανάλογα με την εντολή
  - με επέκταση προσήμου
    - για lw, sw, addi, slti, …
  - με επέκταση με μηδενικά (στα αριστερά)
    - για addiu, …
- Αν χρειαζόμαστε μεγαλύτερες σταθερές;
  - αρχή: κάνε τη συνηθισμένη περίπτωση γρήγορη
  - υπάρχουν τρόποι, αλλά χρειάζονται περισσότερες εντολές
    - συνεχίζεται μετά το επόμενο παράδειγμα.

Παράδειγμα 1/2

Κωδικοποίηση: `addi $t0,$s1,-50`

Οι κωδικοί βρίσκονται στην πράσινη κάρτα MIPS

- `opcode` = 8
- `rs` = πηγή 1η: `$s1` είναι ο 17
- `rt` = προορισμός: `$t0` είναι ο 8
- `immediate` = -50_{ten} = $FFCE_{hex}$ συμπλήρωμα ως προς 2

Παράδειγμα 2/2

```
addi $t0,$s1,-50      add $8,$17,-50
```

Κωδικοποίηση με τα πεδία σε δεκαδικό σύστημα:

8	17	8	-50

Κωδικοποίηση με τα πεδία σε δυαδικό σύστημα:

001000	10001	01000	1111	1111	1100	1110

Ολόκληρη η λέξη εντολής σε δεκαεξαδικό (γλώσσα μηχανής):

$2228\ \ FFCE_{hex}$

Μεγάλες σταθερές

- Πώς μπορούμε να έχουμε μια σταθερά 32 bit;
  - π.χ. για μια μάσκα για την εντολή andi που κρατάει το πιο σημαντικό byte μιας λέξης `0xff000000`
- Δεν χωράει στις εντολές του MIPS
  - όλη η εντολή είναι 32bit. Δεν θα υπάρχει χώρος για opcode!
  - δεν θέλουμε εντολές με μεταβλητό μέγεθος (άλλες 32, άλλες 64)
- Λύση: νέα εντολή `lui` – load upper immediate
  - `lui rdst, imm`
  - Γράφει («φορτώνει») τη σταθερά στα 16 περισσότερο σημαντικά bit του καταχωρητή (31-16)
  - και μηδενίζει τα υπόλοιπα bit (15-0)

Παράδειγμα: 32bit σταθερά

- Θέλουμε μια σταθερά: $FF0000FF_{hex}$ για να πάρουμε το 1ο και το τελευταίο byte μιας λέξης
  - εντολή: `andi  $t0, $s0, 0xff0000ff`
  - αλλά δεν γίνεται γιατί η σταθερά είναι πολύ μεγάλη
- Γράφουμε:

```
lui $at, 0xff00
ori $at, $at, 0x00ff  # όχι addi
and $t0, $s0, $at
```

- Νέος καταχωρητής at – assembler temporary
  - μόνο για τον assembler. Μη τον πειράζετε!

Ψευτοεντολή li

- Σύνταξη: `li rdest, σταθερά`
- Γράφει μια σταθερά σε ένα καταχωρητή
  - οποιουδήποτε μεγέθους σταθερά (ως 32 bit)
- Ο assembler την μετατρέπει σε
  - μία εντολή (π.χ. ori) αν η σταθερά είναι μικρή (και στα 16 lsb)
  - σε 2 εντολές (π.χ. lui, ori) αλλιώς
    - γι'αυτό χρειάζεται ο καταχωρητής at
- Το ίδιο κάνει και η `la` που έχουμε ήδη δεί
  - η `la` αντί για σταθερά παίρνει label
- Αυτήν επιτρέπεται να την χρησιμοποιείτε στις ασκήσεις

Κωδικοποίηση εντολών διακλάδωσης

- Εντολές beq, bne
  - τύπου I
  - 2 καταχωρητές για σύγκριση (δεν γράφεται αποτέλεσμα)
  - 1 σταθερά για να δείξει τον στόχο της διακλάδωσης (branch target)
    - από που θα συνεχιστεί η εκτέλεση αν η συνθήκη αληθεύει
- Πώς χρησιμοποιείται η σταθερά για να δείξει τη διεύθυνση;

Χρήση εντολών διακλάδωσης

- Οι εντολές διακλάδωσης χρησιμοποιούνται για
  - if-then, if-then-else, επαναλήψεις: while, for, do-while
- Οι ακολουθίες εντολών σε loops, τμήματα then, else είναι σχετικά μικρές
  - π.χ. οι περισσότεροι βρόγχοι είναι < 50 εντολές
- Συμπέρασμα: ο στόχος της διακλάδωσης βρίσκεται κάπου κοντά στην εντολή διακλάδωσης
  - και η τρέχουσα διεύθυνση κρατιέται στον Program Counter (PC)

PC-relative addressing

- Χρησιμοποιούμε τη σταθερά (immediate) των εντολών διακλάδωσης ως offset του PC
  - μας δείχνει πόσο μακριά (offset) θα πάμε από εδώ που βρισκόμαστε τώρα (PC)
- Το offset μπορεί να είναι θετικός ή αρνητικός αριθμός
  - άρα η σταθερά είναι σε συμπλήρωμα ως προς 2
- Ο στόχος της διακλάδωσης είναι $\pm 2^{15}$ από το PC

Απόσταση διακλάδωσης

- Μπορούμε να αυξήσουμε λίγο παραπάνω την απόσταση που μπορεί να έχει ο στόχος από την εντ. διακλάδωσης
- Ισχύουν τα παρακάτω:
  - Οι διευθύνσεις αναφέρονται σε (χωριστά) bytes
  - Οι εντολές του MIPS είναι πάντα 32bit (4 byte)
  - Ο MIPS απαιτεί οι λέξεις να είναι ευθυγραμμισμένες
- Αρα: ο αριθμός των bytes που πρέπει να προστεθούν στο PC θα είναι πάντα πολλαπλάσιος του 4
  - τα 2 λιγότερο σημαντικά bit του offset θα είναι πάντα 0
- Θεωρούμε ότι η σταθερά αναφέρεται σε απόσταση εντολών (λέξεων) όχι σε bytes
  - μπορούμε να φτάσουμε σε ± 2^{15} εντολές από το PC ή ± 2^{17} bytes

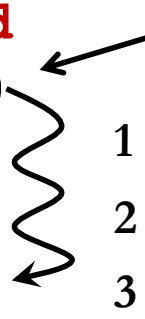
Υπολογισμός στόχου

- Αν δεν ακολουθηθεί η διακλάδωση (branch not taken)
  - η συνθήκη δεν αληθεύει
  - διεύθυνση στόχου: PC + 4
  - η διεύθυνση της επόμενης εντολής
- Αν ακολουθηθεί η διακλάδωση (branch taken)
  - η συνθήκη αληθεύει
  - δεύθυνση στόχου: (PC + 4) + immediate * 4
- Υπενθύμιση
  - το immediate μετράει απόσταση εντολών (λέξεων)
    - μπορεί να είναι θετικό ή αρνητικό
  - η διευθύνση στόχου είναι σε bytes (μνήμη byte addressable)
- Γιατί πάντα PC+4; θέμα υλοποίησης σε υλικό

Παράδειγμα κωδικ. διακλάδωσης

- Κώδικας MIPS:

```
Loop:  beq   $9,$0,End
       add   $8,$8,$10
       addi  $9,$9,-1
       j     Loop
End:
```



Μετράμε από την εντολή μετά την διακλάδωση

- Πεδία I-Format:

`opcode` = 4 (από το πράσινη κάρτα)

`rs` = 9 (πρώτη πηγή)

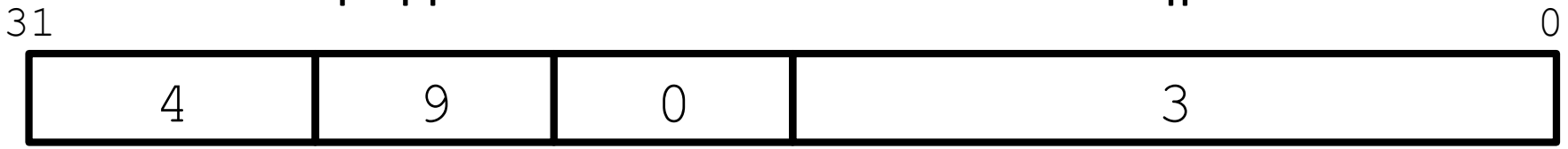
`rt` = 0 (δεύτερη πηγή)

`immediate` = ???

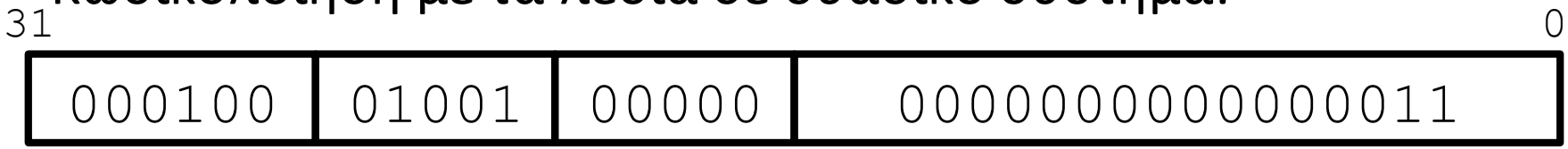
Παράδειγμα 2/2

```
beq $9, $0, 3
```

Κωδικοποίηση με τα πεδία σε δεκαδικό σύστημα:



Κωδικοποίηση με τα πεδία σε δυαδικό σύστημα:

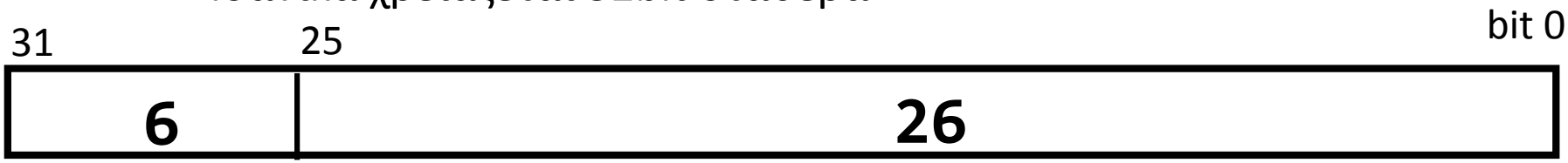


Ολόκληρη η λέξη εντολής σε δεκαεξαδικό (γλώσσα μηχανής):

$1120\ 0003_{hex}$

Εντολές τύπου J

- Οι εντολή άλματος (j) χρειάζεται να μπορεί να πάει οπουδήποτε στο πρόγραμμα
  - Ιδανικά χρειάζεται 32bit σταθερά



Ονόματα πεδίων:

Το opcode βρίσκεται στην ίδια θέση και έχει ίδιο «πλάτος»

Μένουν 26 bit για τη σταθερά

Διευθύνσεις εντολών J

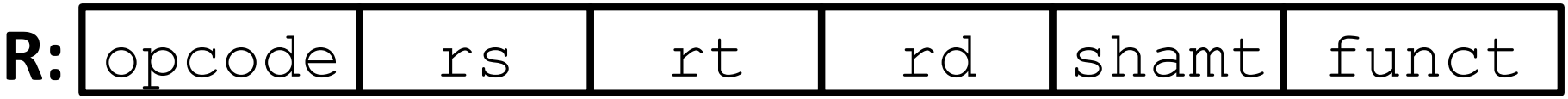
- Χρησιμοποιούμε το ίδιο τρικ με τις διακλαδώσεις
  - η σταθερά αναφέρεται σε λέξεις αντί για bytes
  - έτσι κερδίζουμε 2 bits επιπλέον
- Τα υπόλοιπα 4, τα συμπληρώνουμε από τα περισσότερο σημαντικά bit του PC
  - στόχος = { (PC+4)[31..28], target address, 00_{two} }
  - Το {} σημαίνει ένωσε τα bits (Verilog)
- Μπορεί να φτάσει μέχρι 256MB μακριά

Σύνοψη – γλώσσα μηχανής

- I-Format: Εντολές με σταθερές (π.χ. `addi`), `lw/sw` ,`beq/bne`
  - Αλλά όχι για τις εντολές ολίσθησης (sll, srl, sra)
  - Οι διακλαδώσεις χρησιμοποιούν PC-relative addressing

- J-Format: `j`

- R-Format: όλες οι υπόλοιπες εντολές



Επόμενο μάθημα

MIPS assembly – Φροντιστήριο - επανάληψη

(στείλτε απορίες μέχρι την Πέμπτη το μεσημέρι)