Αρχιτεκτονική Υπολογιστών

Κεφάλαιο 2

Εντολές: η γλώσσα του υπολογιστή

[Έχει χρησιμοποιηθεί υλικό από τις διαφάνειες *Computer Organization and Design, 4th Edition*, Patterson & Hennessy, © 2008, MK]

Εισαγωγή

- Σύνολο εντολών (instruction set)
  - Το ρεπερτόριο των εντολών ενός υπολογιστή
- Διαφορετικοί υπολογιστές έχουν διαφορετικά σύνολα εντολών
  - Άλλα με πολλά κοινά στοιχεία
- Οι παλιότεροι υπολογιστές είχαν πολύ απλά σύνολα εντολών
  - Απλοποιημένη υλοποίηση
- Πολλοί σύγχρονοι υπολογιστές έχουν επίσης απλά σύνολα εντολών

Σύνολο εντολών του MIPS

- Στα πλαίσια του μαθήματος θα μελετήσουμε το σύνολο εντολών του επεξεργαστή MIPS
- Σχεδιάστηκε στο Stanford, εκμεταλλεύεται εμπορικά από την MIPS Technologies (www.mips.com)
- Μεγάλο τμήμα της αγοράς ενσωματωμένων επεξεργαστών
  - Σε εφαρμογές όπως ηλεκτρονικά προιόντα, εξοπλισμός δικτύου/αποθήκευσης, ψηφιακές κάμερες, εκτυπωτές, …
- Ομοιότητες με σύνολα εντολών σύγχρονων επεξεργαστών

MIPS

- MIPS: Microprocessor without Interlocked Pipeline Stages
  - Reduced instruction set computer (RISC)
- Ιστορική αναδρομή:
  - 1981: Σχεδιάστηκε από την ομάδα του καθηγητή J.L. Hennessy στο Πανεπιστήμιο Stanford
  - 1984: Δημιουργήθηκε η MIPS Computer Systems, που ανακοίνωσε τον πρώτο MIPS R2000 το 1985.
  - 1990: ένας στους τρεις RISC επεξεργαστές που πωλούνταν βασίζονταν στην αρχιτεκτονική MIPS
  - 2002: κατασκευάστηκαν σχεδόν 100 εκατομμύρια αυτών των δημοφιλών μικροεπεξεργαστών
  - Σήμερα: βρίσκονται σε προϊόντα εταιρειών όπως ATI Technologies, Broadcom, Cisco, NEC, Nintendo, Silicon Graphics, Sony, Texas Instruments, Toshiba, κ.α.

Αριθμητικές λειτουργίες

- Πρόσθεση και αφαίρεση, τρεις τελεστέοι
  - Δύο προέλευσης και ένας προορισμού

```
add a, b, c  # a = b + c
```

- Όλες οι αριθμητικές λειτουργίες έχουν αυτή τη μορφή
- *Σχεδιαστική αρχή 1:* Η απλότητα ευνοεί την κανονικότητα
  - Η κανονικότητα απλοποιεί την υλοποίηση
  - Η απλότητα οδηγεί σε καλύτερη απόδοση με χαμηλότερο κόστος

Πρόσθεση

- Αν θέλουμε να προσθέσουμε τις μεταβλητές b, c, d, και e και να τοποθετήσουμε το αποτέλεσμα στη μεταβλητή a
  - Τότε θα εκτελέσουμε την ακολουθία εντολών

```
add a, b, c     # a = b+c
add a, a, d     # a = b+c+d
add a, a, e     # a = b+c+d+e
```

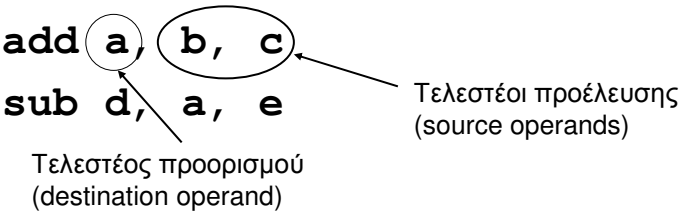
Μεταγλώττιση εντολών της C

- Κώδικας C

```
a = b + c;
d = a - e;
```

- Μεταγλώττιση στη συμβολική γλώσσα MIPS

```
add a, b, c
sub d, a, e
```



Σύνθετη ανάθεση της C

- Κώδικας C :

```
f = (g + h) - (i + j);
```

- Μεταγλωττισμένος κώδικας MIPS:

```
add t0, g, h   # t0 = g + h
add t1, i, j   # t1 = i + j
sub f, t0, t1  # f = t0 - t1
               # f = (g + h)-(i + j)
```

Συμβολική γλώσσα

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	add a, b, c	a = b + c	Πάντα τρεις τελεστέοι
	subtract	sub a, b, c	a = b - c	Πάντα τρεις τελεστέοι

Τελεστέοι του υλικού

- Στις συμβολικές γλώσσες οι τελεστέοι των αριθμητικών εντολών είναι περιορισμένοι
  - Καταχωρητές (registers)
    - Περιορισμένος αριθμός ειδικών θέσεων του υλικού
- Αρχιτεκτονική MIPS :
  - Περιέχει 32 καταχωρητές με μέγεθος 32 bit
- *Σχεδιαστική αρχή 2:* Το μικρότερο είναι γρηγορότερο
  - Αντί για κύρια μνήμη: εκατομμύρια θέσεις μνήμης

Καταχωρητές (registers)

- `$s0, $s1, ...`
  - αποθηκεύουν μεταβλητές
- `$t0, $t1, ...`
  - αποθηκεύουν προσωρινές τιμές
- Επομένως,
  - Οι τελεστέοι των αριθμητικών εντολών πρέπει να επιλέγονται ανάμεσα στους 32 καταχωρητές
- Ορολογία: λέξη (word)
  - Η φυσική μονάδα προσπέλασης σε έναν υπολογιστή, συνήθως μια ομάδα από 32 bit· αντιστοιχεί στο μέγεθος ενός καταχωρητή στην αρχιτεκτονική MIPS

Παράδειγμα

- Κώδικας C:

```
f = (g + h) - (i + j);
```

```
f, g, h, i, j στους $s0, $s1, $s2,
  $s3, $s4
```

- Μεταγλωττισμένος κώδικας MIPS:

```
add $t0, $s1, $s2   # $t0 = g + h
add $t1, $s3, $s4   # $t1 = i + j
sub $s0, $t0, $t1   # f = $t0 - $t1,
                    # f = (g+h)-(i+j)
```

Αποθήκευση δομών δεδομένων

- Πώς μπορεί ένας υπολογιστής να αναπαραστήσει και να προσπελάσει πολύπλοκες δομές δεδομένων;
  - Π.χ. Έναν πίνακα
- Οι αριθμητικές πράξεις γίνονται μόνο μεταξύ καταχωρητών στις εντολές MIPS
  - Ο επεξεργαστής περιέχει περιορισμένο αριθμό καταχωρητών
  - Οι δομές δεδομένων διατηρούνται στη μνήμη
  - Άρα, ο MIPS πρέπει:
    - να διαβάζει δεδομένα από τη μνήμη
    - να τα αποθηκεύει σε καταχωρητές
    - να εκτελεί αριθμητικές πράξεις μεταξύ καταχωρητών
    - να αποθηκεύει το αποτέλεσμα πίσω στη μνήμη

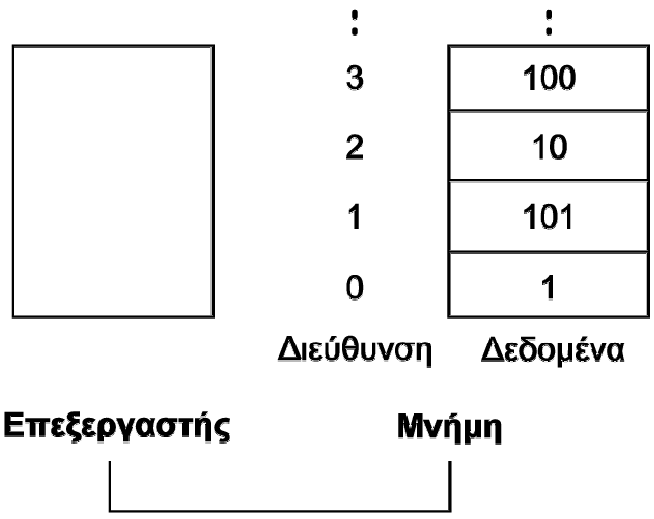
Εντολή μεταφοράς δεδομένων

- Εντολή μεταφοράς δεδομένων
  - Μια εντολή που μεταφέρει δεδομένα μεταξύ μνήμης και καταχωρητών
- Διεύθυνση
  - Μια τιμή που χρησιμοποιείται για να καθορίσει τη θέση ενός συγκεκριμένου στοιχείου δεδομένων μέσα σε έναν πίνακα μνήμης
- Εντολή φόρτωσης δεδομένων:
  - lw (load word)
  - Τι αποτέλεσμα θα έχει η εντολή;

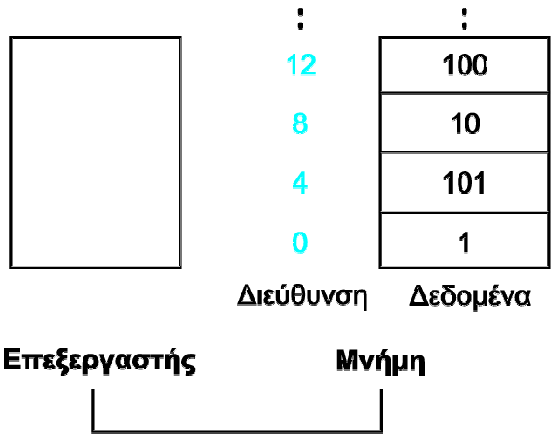
```
lw $t0, 0($s0)
```

- Εντολή αποθήκευσηςδεδομένων:
  - sw (store word)

Διευθύνσεις μνήμης



Πραγματικές διευθύνσεις λέξεων



Οι διευθύνσεις διαδοχικών λέξεων διαφέρουν κατά 4

- Άρα, για να φορτώσουμε το στοιχείο A[8] :

```
lw   $t0, 32($s3)     # offset 4 x 8 = 32
```

Μεταγλώττιση με load/store

- Η μεταβλητή h είναι στον $s2
- Η διεύθυνση βάσης του A είναι στον $s3

```
A[12] = h + A[8];
```

- Μεταγλωττισμένος κώδικας MIPS

```
lw  $t0,32($s3)     # $t0 = A[8]
add $t0,$s2,$t0     # $t0 = h+A[8]
sw  $t0,48($s3)     # A[12] = $t0
```


Καταχωρητές vs. μνήμη

- Η προσπέλαση των καταχωρητών είναι γρηγορότερη από την προσπέλαση της μνήμης
- Όταν οι τελεστέοι είναι στη μνήμη απαιτούνται φορτώσεις και αποθηκεύσεις
  - Εκτελούνται περισσότερες εντολές
- Ο μεταγλωττιστής πρέπει να χρησιμοποιεί όσο το δυνατό περισσότερο τους καταχωρητές για τις μεταβλητές
  - Να διασκορπίζει (spill) στη μνήμη τις μεταβλητές που χρησιμοποιούνται λιγότερο
  - Η βέλτιστη χρήση των καταχωρητών είναι σημαντική!

Σταθερές τιμές

- Τα προγράμματα χρησιμοποιούν σταθερές σε πράξεις
  - Π.χ. η αύξηση του αριθμοδείκτη ενός πίνακα
- Πώς θα υλοποιήσουμε πράξεις με σταθερές, με χρήση των εντολών που έχουμε δει έως τώρα;
- Παράδειγμα:
  - για να προσθέσουμε τη σταθερά 4 στον καταχωρητή $s3

```
lw  $t0,AddrConstant4($s1)
        # $t0 = σταθερά 4
add $s3,$s3,$t0
        # $s3 = $s3 + $t0 ($t0==4)
```


Εντολές με σταθερό τελεστέο

- Εναλλακτική λύση:
  - χρήση αριθμητικών εντολών με σταθερό τελεστέο
- Εντολή άμεσης πρόσθεσης
  - addi (add immediate)

```
addi $s3,$s3,4  # $s3 = $s3 + 4
```

- Δεν υπάρχει άμεση αφαίρεση
  - Απλά χρησιμοποιούμε αρνητική σταθερά

```
addi $s2, $s1, -1
```

- *Σχεδιαστικά αρχή 3:* Κάνε την πιο συνηθισμένη περίπτωση γρήγορη
  - Οι μικρές σταθερές είναι συνηθισμένες
  - Με τον άμεσο τελεστέο αποφεύγουμε τη φόρτωση

Η σταθερά 0

- Ο καταχωρητής 0 του MIPS ($zero) είναι η σταθερά 0
  - Δεν μπορεί να γραφτεί
- Χρήσιμος για συνηθισμένες λειτουργίες
  - Π.χ., μεταφορά τιμής μεταξύ καταχωρητών

```
add $t2, $s1, $zero
```


Τελεστέοι MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	$s0,$s1,....	Γρήγορες θέσεις για δεδομένα. Στο MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . ., Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων στο MIPS. Ο MIPS χρησιμοποιεί διευθύνσεις byte, οπότε διαδοχικές διευθύνσεις λέξης διαφέρουν κατά 4.

Συμβολική γλώσσα MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	Χρησιμοποιείται για να προσθέσει σταθερές
Μεταφορά δεδομένων	load word	lw $s1,100($s2)	$s1 = Memory[$s2+100]	Δεδομένα από τη μνήμη σε καταχωρητή
	store word	sw $s1,100($s2)	Memory[$s2+100] = $s1	Δεδομένα από καταχωρητή στη μνήμη

Απρόσημοι ακέραιοι

- Αριθμός των n-bit

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Εύρος: 0 έως $+2^n - 1$
- Παράδειγμα
  - $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$
- Με 32 bit
  - 0 έως +4,294,967,295

Προσημασμένοι σε συμπλήρωμα ως προς 2

- Αριθμός των n-bit

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Εύρος: -2^{n-1} έως $+2^{n-1} - 1$
- Παράδειγμα
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- Με 32 bit
  - –2,147,483,648 to +2,147,483,647

Προσημασμένοι σε συμπλήρωμα ως προς 2

- Bit 31: πρόσημο
  - 1 για αρνητικούς αριθμούς
  - 0 για μη αρνητικούς
- Οι μη αρνητικοί αριθμοί έχουν την ίδια απρόσημη αναπαράσταση και σε συμπλήρωμα ως προς 2
- Συγκεκριμένοι αριθμοί:
  - 0: 0000 0000 … 0000
  - –1: 1111 1111 … 1111
  - Ο μικρότερος αρνητικός: 1000 0000 … 0000
  - Ο μεγαλύτερος θετικός: 0111 1111 … 1111

Υπολογισμός αντιθέτου

- Συμπλήρωμα και πρόσθεση του 1
  - Συμπλήρωμα σημαίνει 1 → 0, 0 → 1

$$x + \bar{x} = 1111...111_2 = -1$$
$$\bar{x} + 1 = -x$$

- Παράδειγμα: αντίθετο του +2
  - +2 = 0000 0000 … 0010_2
  - –2 = 1111 1111 … 1101_2 + 1
    = 1111 1111 … 1110_2

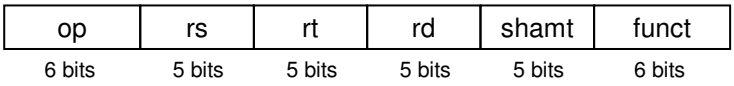
Επέκταση προσήμου

- Αναπαράσταση του αριθμού με περισσότερα bit
  - Διατήρηση της ίδιας αριθμητικής τιμής
- Στο σύνολο εντολών του MIPS
  - `addi`: επέκταση της άμεσης τιμής
  - `lb`, `lh`: επέκταση του byte/halfword που φορτώνει
- Επανάληψη του bit προσήμου στα αριστερά
  - στις απρόσημες τιμές: επεκτείνουμε με 0
- Παραδείγματα: 8-bit σε 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

Αναπαράσταση εντολών

- Οι εντολές κωδικοποιούνται στο δυαδικό
  - Ονομάζεται κώδικας μηχανής (machine code)
- Γλώσσα μηχανής (machine language)
  - Δυαδική αναπαράσταση των εντολών σε έναν υπολογιστή
- Εντολές MIPS
  - Κωδικοποιούνται σαν λέξεις εντολών των 32-bit
  - Μικρός αριθμός μορφών εντολών για την κωδικοποίηση του κώδικα λειτουργίας (opcode), τους αριθμούς των καταχωρητών,
  - Κανονικότητα!
- Αριθμοί καταχωρητών
  - $t0 – $t7: καταχωρητές 8 – 15
  - $t8 – $t9: καταχωρητές 24 – 25
  - $s0 – $s7: καταχωρητές 16 – 23

Εντολή MIPS R-format



- Πεδία εντολής
  - ***op***: opcode (operation code — κώδικας λειτουργίας)
  - ***rs***: Τελεστέος προέλευσης πρώτου καταχωρητή
  - ***rt***: Τελεστέος προέλευσης δεύτερου καταχωρητή
  - ***rd***: Τελεστέος καταχωρητή προορισμού
  - ***shamt***: Ποσότητα ολίσθησης (shift amount)
  - ***funct***: Κώδικας συνάρτησης (function code), επεκτείνει το opcode

Παράδειγμα εντολής R-format

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$00000010001100100100000000100000_2 = 02324020_{16}$

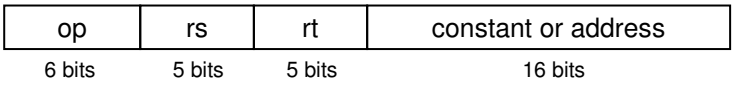
Δεκαεξαδικό

- Βάση 16
  - Πιο εύκολη αναπαράσταση του κώδικα μηχανής
  - 4 bit ανά δεκαεξαδικό ψηφίο

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

- Παράδειγμα: eca8 6420
  - 1110 1100 1010 1000 0110 0100 0010 0000

Εντολή MIPS I-format



- Άμεσες αριθμητικές και εντολές load/store
  - rt: αριθμός καταχωρητή προέλευσης ή προορισμού
  - Σταθερά: -2^{15} έως $+2^{15} - 1$
  - Διεύθυνση: μετατόπιση (offset) που προστίθεται στη διεύθυνση βάσης στον rs
- *Σχεδιαστική αρχή 4:* Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς
  - Οι διαφορετικές μορφές περιπλέκουν την κωδικοποίηση, αλλά επιτρέπουν την ομοιομορφία στις εντολές
  - Διατηρείται τις μορφές όσο το δυνατό παρόμοιες

Κωδικοποίηση εντολών MIPS

Εντολή	Μορφή	Op	Rs	Rt	Rd	Shamt	Funct	Address
add	R	0	Reg	Reg	Reg	0	32_{10}	-
sub (subtract)	R	0	Reg	Reg	Reg	0	34_{10}	-
add immediate	I	8_{10}	Reg	Reg	-	-	-	σταθερά
lw (load word)	I	35_{10}	Reg	Reg	-	-	-	διεύθυνση
sw (store word)	I	43_{10}	Reg	Reg	-	-	-	διεύθυνση

C ➔assembly➔γλώσσα μηχανής

- Η μεταβλητή h είναι στον $s2
- Η διεύθυνση βάσης του A είναι στον $t1

```
A[300] = h + A[300];
```

Μεταγλωττισμένος κώδικας MIPS

```
lw  $t0,1200($t1) # $t0 = A[300]
add $t0,$s2,$t0   # $t0 = h + A[300]
sw  $t0,1200($t1) # A[300] = h + A[300]
```

Κώδικας μηχανής

op	rs	rt	rd	shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

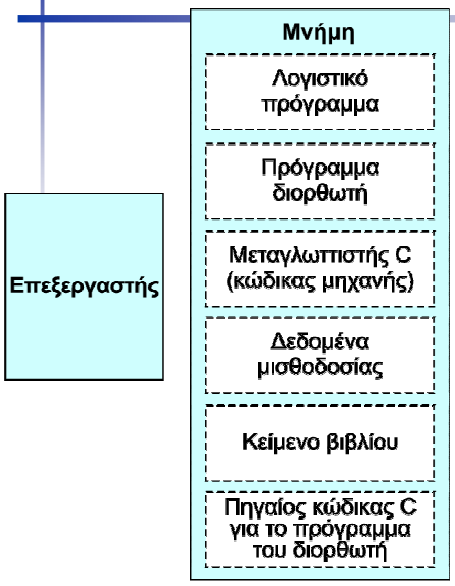
Δυαδική αναπαράσταση εντολών

op	rs	rt	rd	shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

op	rs	rt	rd	shamt	funct
100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Σε τι διαφέρουν οι δυο εντολές lw και sw ;

Η έννοια του αποθηκευμένου προγράμματος



- Οι εντολές αναπαριστάνονται δυαδικά, όπως και τα δεδομένα
- Οι εντολές και τα δεδομένα αποθηκεύονται στη μνήμη
- Τα προγράμματα μπορούν να λειτουργήσουν σε προγράμματα
  - Π.χ. μεταγλωττιστές, …
- Η δυαδική συμβατότητα επιτρέπει στα μεταγλωττισμένα προγράμματα να εκτελούνται σε διαφορετικούς υπολογιστές

Λογικές πράξεις

Λογικές λειτουργίες	Τελεστές C	Τελεστές Java	Εντολές MIPS
Αριστερή ολίσθηση (Shift left)	<<	<<	sll
Δεξιά ολίσθηση (Shift right)	>>	>>>	srl
AND bit προς bit	&	&	and, andi
OR bit προς bit	|	|	or, ori
NOT bit προς bit	~	~	nor

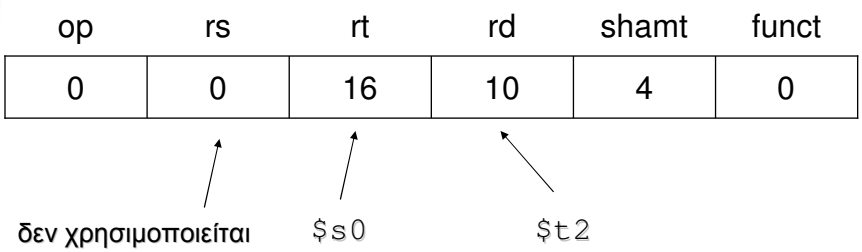
Ολισθήσεις

- Ολίσθηση της 32-bit τιμής:
  - 0000 0000 0000 00000 000 0000 0000 0000 $1001_2 = 9_{10}$
- Αριστερή λογική ολίσθηση (shift left logical)
  - Κατά 1-bit
    - 0000 0000 0000 0000 0000 0000 0000 0001 0010 $_2 = 18_{10}$
  - Κατά 4-bit
    - 0000 0000 0000 0000 0000 0000 0000 1001 0000 $_2 = 144_{10}$
  - Ποια αριθμητική πράξη υλοποιεί η αριστερή ολίσθηση;
- Δεξιά λογική ολίσθηση (shift right logical)
  - Κατά 1-bit
    - 0000 0000 0000 0000 0000 0000 0000 0000 0100 $_2 = 4_{10}$
  - Κατά 2-bit
    - 0000 0000 0000 0000 0000 0000 0000 0000 0010 $_2 = 2_{10}$
  - Ποια αριθμητική πράξη υλοποιεί η δεξιά ολίσθηση;

Εντολή ολίσθησης του MIPS

```
sll  $t2,$s0,4  # $t2 = $s0 << 4 bit
```

- Αριστερή λογική ολίσθηση $s0 κατά 4 θέσεις και αποθήκευση του αποτελέσματος στον $t2



Λογικό AND

`$t2` = 0000 0000 0000 0000 0000 1101 0000 0000_2

`$t1` = 0000 0000 0000 0000 0011 1100 0000 0000_2

- τότε, μετά την εκτέλεση της

```
and $t0,$t1,$t2   # $t0 = $t1 & $t2
```

- ο `$t0` θα είναι:

  0000 0000 0000 0000 0000 1100 0000 0000_2

- Χρήση της AND (masking):
  - Για να μηδενίσουμε (clear) κάποια bit σε μια μεταβλητή

Λογικό OR

`$t2` = 0000 0000 0000 0000 0000 1101 0000 0000_2

`$t1` = 0000 0000 0000 0000 0011 1100 0000 0000_2

- τότε, μετά την εκτέλεση της

```
or $t0,$t1,$t2    # $t0 = $t1 | $t2
```

- ο `$t0` θα είναι:

  0000 0000 0000 0000 0011 1101 0000 0000_2

- Χρήση της OR:
  - Για να θέσουμε την τιμή 1 (set) σε κάποια bit σε μια μεταβλητή

Λογικό NOT (αντιστροφή)

- NOT : Μια λογική πράξη με έναν τελεστέο που αντιστρέφει τα bit

`$t1` = 0000 0000 0000 0000 0011 1100 0000 0000_2

`~$t1` = 1111 1111 1111 1111 1100 0011 1111 1111_2

- Για να διατηρηθεί στον MIPS η μορφή των εντολών με δύο τελεστέους η πράξη NOT υλοποιείται με την εντολή NOR

Λογικό NOR

- NOR : Μια λογική πράξη με δύο τελεστέους που υπολογίζει το NOT του OR των δύο τελεστέων

`$t2` = 0000 0000 0000 0000 0000 1101 0000 0000_2

`$t1` = 0000 0000 0000 0000 0011 1100 0000 0000_2

`~($t1 | $t3)`

= 1111 1111 1111 1111 1100 0010 1111 1111_2

Λογικό NOR και NOT

- Υπάρχει η εντολή NOR

```
nor $t0,$t1,$t3   # $t0=~($t1|$t3)
```

- Αν ο ένας καταχωρητής είναι ίσος με 0 τότε είναι ισοδύναμη με NOT

```
nor $t0,$t1,$zero # $t0=~$t1
```


Συμβολική Γλώσσα MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Λογικές πράξεις	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Τρεις τελεστέοι καταχωρητή. AND bit προς bit
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Τρεις τελεστέοι καταχωρητή. OR bit προς bit
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Τρεις τελεστέοι καταχωρητή. NOR bit προς bit
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	AND bit προς bit καταχωρητή με σταθερά
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	OR bit προς bit καταχωρητή με σταθερά
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Αριστερή ολίσθηση με σταθερά
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Δεξιά ολίσθηση με σταθερά

Εντολές λήψης αποφάσεων

- Στις γλώσσες προγραμματισμού η λήψη αποφάσεων αντιπροσωπεύεται συνήθως με τη χρήση της εντολής if
- Η συμβολική γλώσσα του MIPS περιλαμβάνει δύο εντολές λήψης αποφάσεων:

```
beq register1, register2, L1
bne register1, register2, L1
```

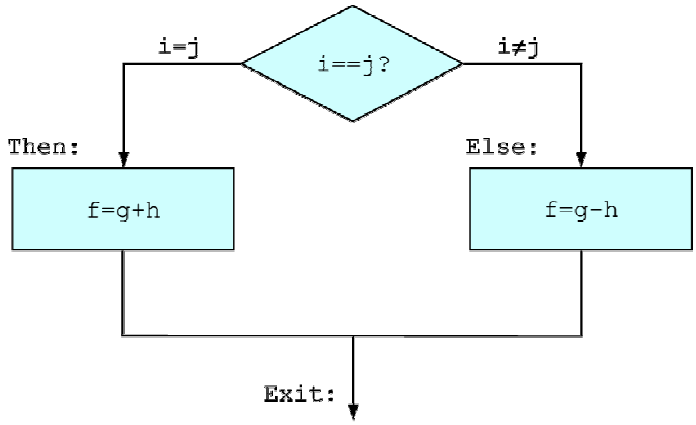
- Αυτές οι δύο εντολές ονομάζονται διακλαδώσεις υπό συνθήκη (conditional branches)

Διακλαδώσεις υπό συνθήκη

- **`beq register1, register2, L1`**
  - Σημαίνει *«πήγαινε στην εντολή με την ετικέτα L1 αν η τιμή στον καταχωρητή register1 είναι ίση με την τιμή στον καταχωρητή register2»*
  - beq : branch if equal (διακλάδωση σε περίπτωση μη ισότητας)
- **`bne register1, register2, L1`**
  - Σημαίνει *«πήγαινε στην εντολή με ετικέτα L1 αν η τιμή στον καταχωρητή register1 δεν είναι ίση με την τιμή του καταχωρητή register2»*
  - bne : branch if not equal (διακλάδωση σε περίπτωση μη ισότητας)

Μεταγλώττιση if-then-else

```
if (i == j) f = g + h;
else f = g - h;
```



Μεταγλώττιση if-then-else

```
if (i == j) f = g + h;
else f = g - h;
```

- f, g, h, i, j αντιστοιχούν στους $s0, $s1, $s2, $s3, $s4

```
     bne $s3,$s4,Else # πήγαινε στο Else αν i ≠ j
     add $s0,$s1,$s2  # f = g + h (παραλείπεται
                      # αν i ≠ j)
     j Exit           # μετάβαση στην Exit
Else:sub $s0,$s1,$s2  # f = g - h
                      # (παραλείπεται αν i = j)
Exit:
```

Ο assembler υπολογίζει τις διευθύνσεις

Μεταγλώττιση while loop

```
while (save[i] == k)
    i += 1;
```

- i=$s3, k=$s5, βάση save=$s6

```
Loop:sll $t1,$s3,2      # $t1 = 4 * i
     add $t1,$t1,$s6    # $t1 = διεύθ. save[i]
     lw  $t0,0($t1)     # $t0 = save[i]
     bne $t0,$s5, Exit  # μετάβαση στο Exit αν
                        # save[i] ≠ k
     addi $s3,$s3,1     # i = i + 1
     j Loop             # πήγαινε στο Loop
Exit:
```


Εντολή set-on-less-than

```
slt $t0, $s3, $s4
```

- ο `$t0` τίθεται ίσος με 1 αν η τιμή στον καταχωρητή `$s3` είναι μικρότερη από την τιμή στον καταχωρητή `$s4`, αλλιώς ο καταχωρητής `$t0` τίθεται ίσος με 0
- Άμεση μορφή (immediate format)

```
slti $t0,$s2,10
        # $t0 = 1 αν $s2 < 10
```

Υλοποίηση όλων των συνθηκών

- Γιατί όχι `blt`, bge, etc;
- Με τις εντολές **`beq, bne, slt, slti`** και τον καταχωρητή **`$zero`** που έχει πάντα την τιμή 0, οι μεταγλωττιστές του MIPS υλοποιούν όλες τις συνθήκες
  - ίσο, διάφορο
  - μικρότερο, μικρότερο ή ίσο
  - μεγαλύτερο, μεγαλύτερο ή ίσο
- Το υλικό για <, ≥, … πιο αργό από το =, ≠
  - Πιο συνδυαστικές διακλαδώσεις απαιτούν περισσότερη λογική σε κάθε εντολή, που οδηγεί σε πιο αργό ρολόι
  - Επηρεάζονται όλες οι εντολές!
- Οι beq και bne είναι η συνηθισμένη περίπτωση
  - Αυτός είναι ένας καλός σχεδιαστικός συμβιβασμός

Εντολή Case ή Switch

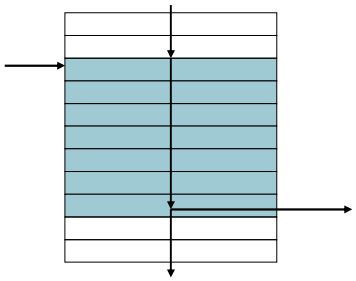
- Μία, από πολλές εναλλακτικές επιλογές κώδικα
  - 1η υλοποίηση: πολλαπλές εντολές if-then-else
  - 2η υλοποίηση: πίνακας διευθύνσεων άλματος (jump address table)
    - σε συνδυασμό με την εντολή jump register
    - **`jr register`**

Προσημασμένοι vs. απρόσημοι

- Προσημασμένη σύγκριση: `slt`, `slti`
- Απρόσημη σύγκριση: `sltu`, `sltui`
- Παράδειγμα
  - $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - `slt  $t0, $s0, $s1  # signed`
    - –1 < +1 ⇒ $t0 = 1
  - `sltu $t0, $s0, $s1  # unsigned`
    - +4,294,967,295 > +1 ⇒ $t0 = 0

Βασικά μπλοκ

- Ένα βασικό μπλοκ (basic block) είναι μια ακολουθία εντολών με καθόλου
  - Ενσωματωμένες διακλαδώσεις (μόνο στο τέλος)
  - Προορισμούς διακλαδώσεων (μόνο στην αρχή)



- Ένας μεταγλωττιστής αναγνωρίζει τα βασικά μπλοκ για βελτιστοποίηση
- Ένας προηγμένος επεξεργαστής μπορεί να επιταχύνει την εκτέλεση των βασικών μπλοκ

Συμβολική γλώσσα MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Διακλάδωση με συνθήκη	branch on equal	beq $s1,$s2,L	αν ($s1 == $s2) πήγαινε στο L	Έλεγχος ισότητας και διακλάδωση
	branch on not equal	bne $s1,$s2,L	αν ($s1 != $s2) πήγαινε στο L	Έλεγχος μη ισότητας και διακλάδωση
	set on less than	slt $s1,$s2,$s3	αν ($s2 < $s3) τότε $s1 = 1· αλλιώς $s1 = 0	Σύγκριση μικρότερο από· χρήση με τις beq, bne
	set on less than immediate	slti $s1,$s2,100	αν ($s2 < 100) τότε $s1 = 1· αλλιώς $s1 = 0	Άμεση σύγκριση μικρότερο από· χρήση με τις beq, bne
Άλμα χωρίς συνθήκη	jump	j L	πήγαινε στο L	Άλμα στη διεύθυνση προορισμού

Κλήση διαδικασίας

- Τα βήματα που απαιτούνται
  1. Τοποθέτηση παραμέτρων σε καταχωρητές
  2. Μεταβίβαση του ελέγχου στη διαδικασία
  3. Λήψη των πόρων αποθήκευσης που χρειάζεται η διαδικασία
  4. Εκτέλεση της επιθυμητής εργασίας
  5. Τοποθέτηση αποτελέσματος σε καταχωρητή για την καλούσα διαδικασία
  6. Επιστροφή στο σημείο κλήσης

Διαδικασίες στο MIPS

- Χρήση των καταχωρητών του MIPS για την κλήση διαδικασιών:
- **`$a0–$a3`**
  - καταχωρητές ορίσματος (argument registers)
  - μεταβιβάζονται οι παράμετροι της διαδικασίας
- **`$v0–$v1`**
  - καταχωρητές τιμής (value registers)
  - επιστρέφονται οι τιμές της διαδικασίας
- **`$ra`**
  - καταχωρητής διεύθυνσης επιστροφής (return address register)
  - για την επιστροφή στη σημείο κλήσης

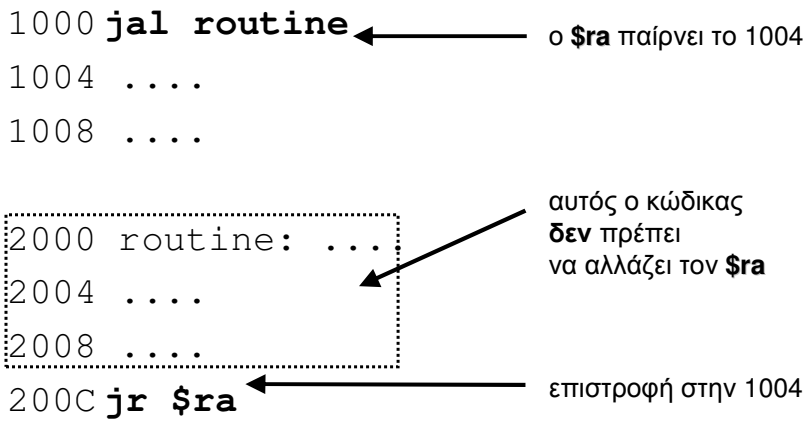
Διαδικασίες στο MIPS

- **Procedure call: εντολή άλματος και σύνδεσης** (jump-and-link instruction)

  `jal ΔιεύθυνσηΔιαδικασίας`
  - Μεταφέρεται σε μια διεύθυνση και
  - Αποθηκεύει τη διεύθυνση της επόμενης εντολής σε έναν καταχωρητή
    - Αποθηκεύει τον **`PC + 4`** στον **`$ra`**
- Μετρητής προγράμματος (program counter - PC)
  - Καταχωρητής που περιέχει τη διεύθυνση της εκτελούμενης εντολής
- Διεύθυνση επιστροφής (return address - RA)
  - Επιτρέπει σε μια διαδικασία να επιστρέψει στην κατάλληλη διεύθυνση
- **Procedure return: jump register (jr):**

  `jr $ra`
  - Άλμα χωρίς συνθήκη στη διεύθυνση που καθορίζεται σε έναν καταχωρητή

Κλήση & επιστροφή



Βήματα κατά την κλήση

- Το πρόγραμμα που καλεί (caller) τοποθετεί τις παραμέτρους στους καταχωρητές **`$a0 – $a3`**
- Χρησιμοποιεί την εντολή **`jal X`** για να κάνει άλμα στη διαδικασία X (καλούμενη - callee)
- Η διαδικασία X εκτελεί τους υπολογισμούς
- Τοποθετεί τα αποτελέσματα στους καταχωρητές **`$v0–$v1`**, και
- Επιστρέφει τον έλεγχο στο σημείο κλήσης μέσω της εντολής **`jr $ra`**

Περισσότεροι καταχωρητές

- Αν απαιτούνται περισσότεροι καταχωρητές για μια διαδικασία (από τους 4 για τα ορίσματα και τους 2 για την επιστροφή τιμής) ;
  - Προσοχή: οι καταχωρητές που χρειάζονται στο πρόγραμμα που καλεί πρέπει να επαναφέρονται στις τιμές που είχαν πριν κληθεί η διαδικασία.
  - ⇨ **Αποθηκεύουμε (διασκορπίζουμε) καταχωρητές στη μνήμη**
- Χρήση στοίβας (stack):
  - Μια δομή δεδομένων για το διασκορπισμό καταχωρητών, οργανωμένη σε μια ουρά «τελευταίο μέσα πρώτο έξω» (LIFO queue).

Στοίβα (stack)

- Δείκτης στοίβας (stack pointer)
  - Δείχνει τη διεύθυνση της στοίβας στην οποία η επόμενη διαδικασία πρέπει να τοποθετήσει τους καταχωρητές ή τη διεύθυνση που βρίσκονται οι παλιές τιμές
  - +/-1 για κάθε καταχωρητή που αποθηκεύεται ή επαναφέρεται
  - **$sp** (stack pointer): καταχωρητής του MIPS
- Συνθηματικές λέξεις για τη μεταφορά δεδομένων
  - **Τοποθέτηση (push)**: προσθήκη δεδομένων
  - **Εξαγωγή (pop)**: αφαίρεση δεδομένων από τη στοίβα

Μεταγλώττιση διαδικασίας

- Διαδικασία που δεν καλεί άλλη
  - Διαδικασία-φύλλο (leaf)

```
int leaf_example
  (int g, int h, int i, int j)
  {int f;
      f = (g + h) - (i + j);
      return f;}
```

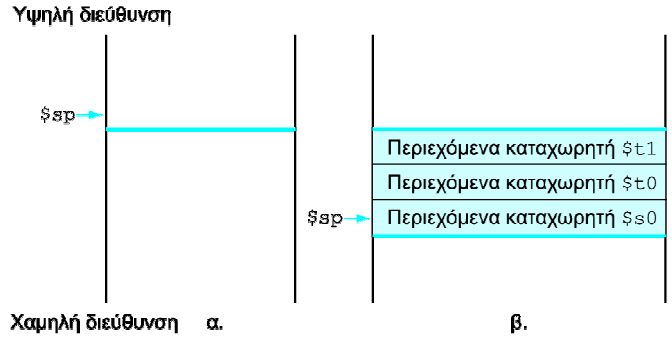
  - Ορίσματα g, …, j στους $a0, …, $a3
  - f στον $s0
  - Αποτέλεσμα στον $v0

Παράδειγμα leaf (1)

- Αποθηκεύουμε τους καταχωρητές (`$s0`, `$t0`, και `$t1`) που θα χρησιμοποιηθούν από τη διαδικασία

```
leaf_example:          # ετικέτα διαδικασίας
  addi $sp,$sp,-12     # χώρος στη στοίβα
                       # για 3 αντικείμενα
  sw $t1, 8($sp)       # αποθήκευση $t1
  sw $t0, 4($sp)       # αποθήκευση $t0
  sw $s0, 0($sp)       # αποθήκευση $s0
```

Εξέλιξη της στοίβας (1)



Παράδειγμα leaf (2)

- Το πέρασμα των παραμέτρων της συνάρτησης `g,h,i,j` γίνεται μέσω των `$a0,$a1,$a2,$a3`
- Για να επιστρέψουμε το αποτέλεσμα της `f($s0)`, το αντιγράφουμε στον καταχωρητή `$v0`

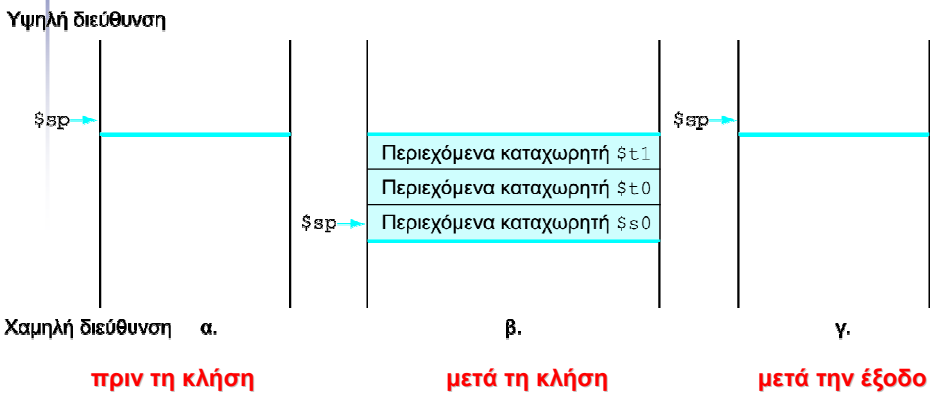
```
add $t0, $a0, $a1      # $t0 περιέχει g+h
add $t1, $a2, $a3      # $t1 περιέχει i+j
sub $s0, $t0, $t1      # f = $t0-$t1
add $v0, $s0, $zero    # επιστρέφει το
                       # f($v0=$s0+0)
```

Παράδειγμα leaf (3)

- Επαναφέρουμε τις παλιές τιμές των καταχωρητών από τη στοίβα
- Η διαδικασία ολοκληρώνεται με μια εντολή jump register

```
lw   $s0, 0($sp)    # επαναφορά $s0
lw   $t0, 4($sp)    # επαναφορά $t0
lw   $t1, 8($sp)    # επαναφορά $t1
addi $sp, $sp,12    # ρύθμιση στοίβας
jr   $ra            # επιστροφή στην καλούσα
                    # ρουτίνα
```


Εξέλιξη της στοίβας (2)



Αποθηκευμένοι & προσωρινοί

- **$t0–$t9**:
  10 προσωρινοί (temporary) καταχωρητές που **δεν διατηρούνται** από την καλούμενη διαδικασία
- **$s0–$s7**:
  8 αποθηκευμένοι (saved) καταχωρητές που **πρέπει να διατηρηθούν** σε μια κλήση διαδικασίας
  - αν χρησιμοποιούνται, η καλούμενη διαδικασία τους αποθηκεύει και τους επαναφέρει
- Άρα, ποιους καταχωρητές πρέπει να τοποθετήσουμε στη στοίβα στο προηγούμενο παράδειγμα;

Κώδικας παραδείγματος leaf

- MIPS code:

```
leaf_example:
  addi $sp, $sp, -4
  sw   $s0, 0($sp)
  add  $t0, $a0, $a1
  add  $t1, $a2, $a3
  sub  $s0, $t0, $t1
  add  $v0, $s0, $zero
  lw   $s0, 0($sp)
  addi $sp, $sp, 4
  jr   $ra
```

Save $s0 on stack

Procedure body

Result

Restore $s0

Return

Ένθετες (nested) διαδικασίες

- Τίθεται θέμα προστασίας του **$ra** και των παραμέτρων (**$a0**, **$a1**, **$a2**, **$a3**)
- Παράδειγμα με αναδρομική (recursive)
  - παραγοντικό (factorial)
  - το ***n*** στον **$a0**

```
int fact (int n)
  {
      if (n < 1) return (1);
         else return (n * fact(n-1));
  }
```


Παράδειγμα fact (1)

- Η μεταβλητή `n` αντιστοιχεί στον καταχωρητή `$a0`
- Αποθηκεύει δύο καταχωρητές στη στοίβα: `$ra` & `$a0`
- Την πρώτη φορά που καλείται η fact, η sw αποθηκεύει μια διεύθυνση στο πρόγραμμα που κάλεσε τη fact

```
fact:
addi  $sp,$sp,-8    # ρύθμιση της στοίβας για 2
sw    $ra, 4($sp)   # αποθήκευση διεύθ. επιστροφής
sw    $a0, 0($sp)   # αποθήκευση του ορίσματος n
```

Παράδειγμα fact (2)

- Ελέγχει αν το n είναι μικρότερο από 1

```
slti $t0,$a0,1          # έλεγχος αν n < 1
beq  $t0,$zero,L1       # αν n >= 1, μετάβαση στην L1
```

- Αν το n είναι μικρότερο από 1, η fact επιστρέφει 1.
- Εξάγει τις δύο αποθηκευμένες τιμές στη στοίβα και μεταπηδά στη διεύθυνση επιστροφής

```
addi $v0,$zero,1        # επιστρέφει 1
addi $sp,$sp,8          # εξάγει 2 τιμές από τη στοίβα
jr   $ra                # επιστροφή
```

- Γιατί πριν από την εξαγωγή (pop) των δύο αντικειμένων από τη στοίβα, δεν φορτώνουμε τους καταχωρητές $a0 και $ra;

Παράδειγμα fact (3)

- Αν το n δεν είναι μικρότερο από 1, το όρισμα n μειώνεται και καλείται πάλι η συνάρτηση fact

```
L1: addi $a0,$a0,-1     # n>=1: το όρισμα παίρνει το (n-1)
    jal fact            # κλήση της fact με (n - 1)
```

- Η επόμενη εντολή είναι εκεί που επιστρέφει η fact.
- Επαναφέρονται η παλιά διεύθυνση επιστροφής και το παλιό όρισμα, μαζί με το δείκτη στοίβας

```
lw   $a0, 0($sp)        # επιστροφή από την jal: επαναφορά
                        # του ορίσματος n
lw   $ra, 4($sp)        # επαναφορά της διεύθυνσης
addi $sp, $sp,8         # ρύθμιση δείκτη στοίβας για
                        # εξαγωγή 2 αντικειμένων
```

Παράδειγμα fact (4)

- Υπολογίζεται το νέο γινόμενο και αποθηκεύεται στον καταχωρητή τιμής $v0
- Άλμα στη διεύθυνση επιστροφής

```
mul  $v0,$a0,$v0     # επιστροφή n * fact (n – 1)
jr   $ra             # επιστροφή στον καλούντα
```


Κώδικας παραδείγματος fact

- MIPS code:

```
fact:
    addi $sp, $sp, -8      # adjust stack for 2 items
    sw   $ra, 4($sp)       # save return address
    sw   $a0, 0($sp)       # save argument
    slti $t0, $a0, 1       # test for n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1     # if so, result is 1
    addi $sp, $sp, 8       #   pop 2 items from stack
    jr   $ra               #   and return
L1: addi $a0, $a0, -1      # else decrement n
    jal  fact              # recursive call
    lw   $a0, 0($sp)       # restore original n
    lw   $ra, 4($sp)       #   and return address
    addi $sp, $sp, 8       # pop 2 items from stack
    mul  $v0, $a0, $v0     # multiply to get result
    jr   $ra               # and return
```

Διατηρούνται ή όχι

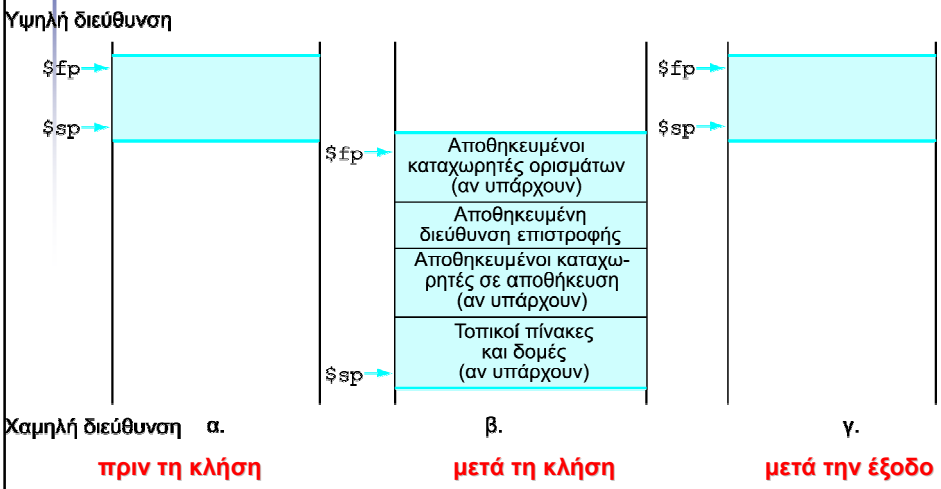
- Κατά τη κλήση μιας διαδικασίας
  - σύμβαση/συμφωνία μεταξύ των προγραμματιστών

Διατηρούνται	Δεν διατηρούνται
Αποθηκευμένοι (saved) καταχωρητές: `$s0–$s7`	Προσωρινοί (temporary) καταχωρητές: `$t0–$t9`
Καταχωρητής δείκτη στοίβας (stack pointer): `$sp`	Καταχωρητές ορίσματος: `$a0–$a3`
Καταχωρητής διεύθυνσης επιστροφής (return address): `$ra`	Καταχωρητές τιμής επιστροφής (return value): `$v0–$v1`
Στοίβα πάνω από το δείκτη στοίβας	Στοίβα κάτω από το δείκτη στοίβας

Κατανομή χώρου στη στοίβα

- Η στοίβα χρησιμοποιείται επίσης για την αποθήκευση μεταβλητών που είναι τοπικές στη διαδικασία και δεν χωρούν σε καταχωρητές, π.χ πίνακες (arrays) και δομές (structures).
- **πλαίσιο διαδικασίας** (procedure frame)
  - Το τμήμα της στοίβας που περιέχει τους αποθηκευμένους καταχωρητές και τις τοπικές μεταβλητές μιας διαδικασίας
- **δείκτης πλαισίου** (frame pointer)
  - Μια τιμή που δείχνει τη θέση των αποθηκευμένων καταχωρητών και των τοπικών μεταβλητών για μια δεδομένη διαδικασία
  - καταχωρητής `$fp` στον MIPS

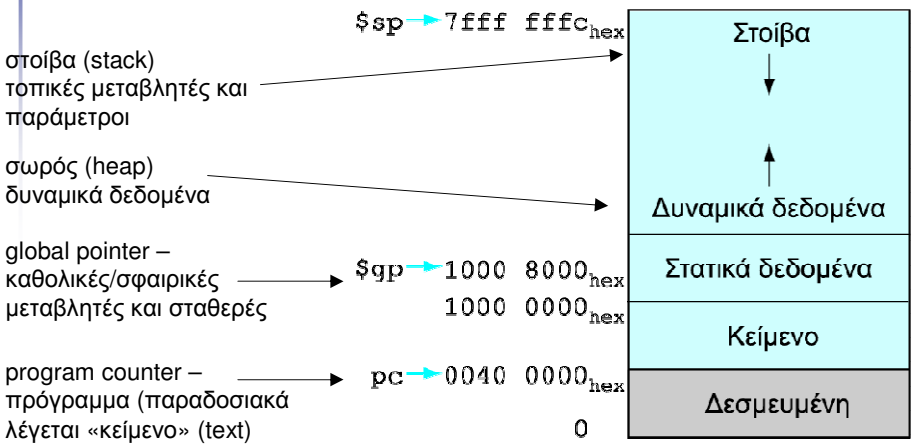
Εξέλιξη στοίβας με fp



Σωρός (heap)

- Εκτός από τις τοπικές μεταβλητές, χρειαζόμαστε χώρο στη μνήμη για τις στατικές μεταβλητές και τις δυναμικές δομές δεδομένων
- Οι στατικές μεταβλητές αποθηκεύονται στο τμήμα στατικών δεδομένων (static data segment)
- Οι δυναμικές δομές δεδομένων (που το μέγεθός τους αυξομειώνεται, π.χ. συνδεδεμένες λίστες) αποθηκεύονται στο **σωρό (heap)**
- Η C κατανέμει και ελευθερώνει χώρο στο σωρό με ρητές συναρτήσεις:
  - Η συνάρτηση `malloc()` κατανέμει χώρο στο σωρό
  - Η συνάρτηση `free()` ελευθερώνει χώρο στο σωρό

Κατανομή μνήμης του MIPS



Σύνοψη καταχωρητών

Όνομα	Αριθμός Καταχωρητή	Χρήση	Διατηρείται κατά την κλήση
$zero	0	η σταθερή τιμή 0	δ.ε.
$v0-$v1	2-3	τιμές αποτελεσμάτων και υπολογισμού εκφράσεων	Όχι
$a0-$a3	4-7	Ορίσματα	Όχι
$t0-$t7	8-15	Προσωρινοί	Όχι
$s0-$s7	16-23	Αποθηκευμένοι	Ναι
$t8-$t9	24-25	περισσότεροι προσωρινοί	Όχι
$gp	28	καθολικός δείκτης	Ναι
$sp	29	δείκτης στοίβας	Ναι
$fp	30	δείκτης πλαισίου	Ναι
$ra	31	διεύθυνση επιστροφής	Ναι

- Λείπουν 3 από τους 32
  - `$at` για ψευδοεντολές από τον συμβολομεταφραστή
  - `$k0, $k1` δεσμεύονται από το λειτουργικό σύστημα

Αναπαράσταση χαρακτήρων

- Οι περισσότεροι υπολογιστές σήμερα χρησιμοποιούν byte των 8 bit για να αναπαραστήσουν χαρακτήρες
- Κώδικας ASCII:
  - Αμερικανικός Πρότυπος Κώδικας Ανταλλαγής Πληροφοριών (American Standard Code for Information Interchange)

Κώδικας ASCII

Τιμή ASCII	Χαρα-κτήρας	Τιμή ASCII	Χαρα-κτήρας	Τιμή ASCII	Χαρα-κτήρας	Τιμή ASCII	Χαρα-κτήρας	Τιμή ASCII	Χαρα-κτήρας	Τιμή ASCII	Χαρα-κτήρας
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

- Τα κεφαλαία και τα πεζά γράμματα διαφέρουν ακριβώς κατά 32
- Οι τιμές που δεν εμφανίζονται περιλαμβάνουν χαρακτήρες μορφοποίησης.
  - Το 8 αναπαριστά το backspace, το 9 το tab, και το 13 το CR
  - Μια άλλη χρήσιμη τιμή είναι το 0 για τον κενό χαρακτήρα (null)

Μεταφορές byte

- Ο MIPS παρέχει εντολές για μεταφορά byte
  - load byte (lb)
    - φορτώνει ένα byte από τη μνήμη, και το τοποθετεί στα δεξιότερα 8 bit ενός καταχωρητή
  - store byte (sb)
    - παίρνει ένα byte από τα δεξιότερα 8 bit ενός καταχωρητή και το γράφει στη μνήμη

```
lb $t0,0($sp)   # Ανάγνωση byte
sb $t0,0($gp)   # Εγγραφή byte
```


Συμβολοσειρές

- Για την αναπαράσταση μιας συμβολοσειράς (string) η γλώσσα προγραμματισμού C χρησιμοποιεί έναν ειδικό χαρακτήρα (ονομάζεται **null** στον κώδικα ASCII) για να σημειώσει το τέλος της συμβολοσειράς
  - Π.χ. η συμβολοσειρά «Cal» αναπαρίσταται στη C από τα παρακάτω 4 byte: 67, 97, 108, 0

Αντιγραφή συμβολοσειράς

- Κώδικας C

```
void strcpy (char x[], char y[])
{
  int i;
   i = 0;
   while ((x[i] = y[i]) != '\0') /*
     αντιγραφή και
      έλεγχος του byte */
   i += 1;
}
```


Κώδικας strcpy (1)

- Οι διευθύνσεις βάσης των πινάκων x και y βρίσκονται στους `$a0` και `$a1`, και το i στον `$s0`
- Αποθηκεύει τον $s0 στη στοίβα

```
strcpy:
    addi   $sp,$sp,-4
    sw     $s0, 0($sp)
    add    $s0,$zero,$zero # i = 0+0
```

Κώδικας strcpy (2)

- Φορτώνουμε το χαρακτήρα του y[i]
- Αποθηκεύουμε το χαρακτήρα στο x[i]
- Ελέγχουμε το χαρακτήρα
  - Αν είναι 0 βγαίνουμε από το βρόχο
  - Αν όχι, αυξάνουμε το i και ο βρόχος επαναλαμβάνεται:

```
L1:add    $t1,$s0,$a1  # δνση y[i] στον $t1
   lb     $t2,0($t1)   # $t2 = y[i]
   add    $t3,$s0,$a0  # δνση x[i] στον $t3
   sb     $t2, 0($t3)  # x[i] = y[i]
   beq    $t2,$zero,L2 # y[i]==0,μετάβαση σε L2
   addi   $s0, $s0,1   # i = i + 1
   j      L1           # μετάβαση σε L1
```


Κώδικας strcpy (3)

- Επαναφέρουμε τον καταχωρητή $s0 και το δείκτη στοίβας και επιστρέφουμε

```
L2:lw     $s0,0($sp)  # y[i]==0:τέλος string
                      # επαναφορά $s0
   addi   $sp,$sp,4   # εξαγωγή λέξης
   jr     $ra         # επιστροφή
```

Παράδειγμα string copy

- MIPS code:

```
strcpy:
    addi $sp, $sp, -4       # adjust stack for 1 item
    sw   $s0, 0($sp)        # save $s0
    add  $s0, $zero, $zero  # i = 0
L1: add  $t1, $s0, $a1      # addr of y[i] in $t1
    lbu  $t2, 0($t1)        # $t2 = y[i]
    add  $t3, $s0, $a0      # addr of x[i] in $t3
    sb   $t2, 0($t3)        # x[i] = y[i]
    beq  $t2, $zero, L2     # exit loop if y[i] == 0
    addi $s0, $s0, 1        # i = i + 1
    j    L1                 # next iteration of loop
L2: lw   $s0, 0($sp)        # restore saved $s0
    addi $sp, $sp, 4        # pop 1 item from stack
    jr   $ra                # and return
```


Χαρακτήρες στη Java – Unicode

- Η Java χρησιμοποιεί την κωδικοποίηση Unicode για τους χαρακτήρες
- Πρότυπο Unicode
  - Παγκόσμια κωδικοποίηση των αλφαβήτων των περισσότερων ανθρώπινων γλωσσών
  - 16 bit (2 byte = 1 halfword, ημιλέξη) για την αναπαράσταση ενός χαρακτήρα
- Ο MIPS παρέχει εντολές για μεταφορά ημιλέξεων
  - load half (lh)
  - store half (sw)

```
lh $t0,0($sp)    # ανάγνωση ημιλέξης
sh $t0,0($gp)    # γραφή ημιλέξης
```

Άμεσοι τελεστέοι

- Οι εντολές με άμεσους τελεστέους, π.χ. `(addi, beq)` έχουν τελεστέους (σταθερές ή διευθύνσεις) των 16 bit
- Ο MIPS παρέχει λύσεις για τελεστέους 32 bit

§2.10 Διευθυνσιοδότηση MIPS για άμεσους τελεστέους

Άμεσοι τελεστέοι των 32 bit

- Ειδική εντολή
  - load upper immediate (lui)
  - **`lui $t0, 255`**

Η έκδοση γλώσσας μηχανής της `lui $t0,255` `# $t0` είναι ο καταχωρητής 8:

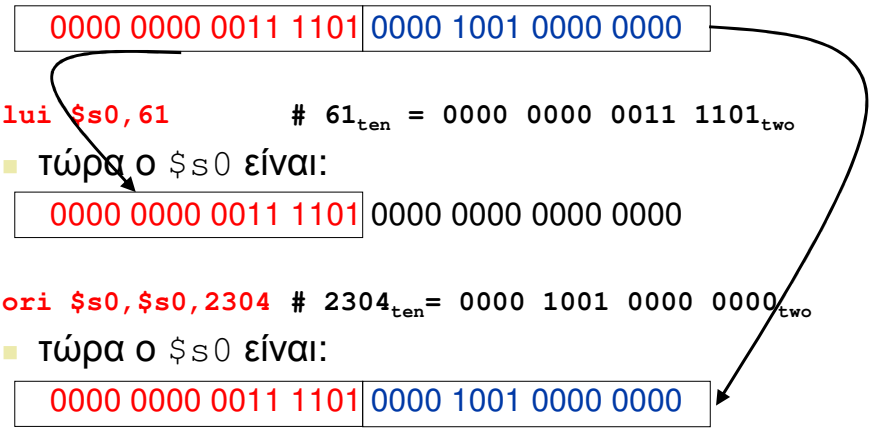
001111	00000	01000	0000 0000 1111 1111

Περιεχόμενα του καταχωρητή `$t0` μετά την εκτέλεση της `lui $t0,255` :

0000 0000 1111 1111	0000 0000 0000 0000

Φόρτωση σταθεράς 32 bit

- Φόρτωση στον `$s0` της τιμής




```
lui $s0,61          # 61ten = 0000 0000 0011 1101two
```

- τώρα ο `$s0` είναι:

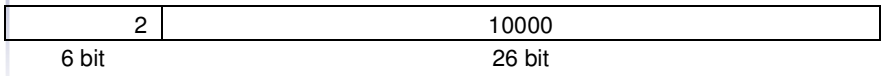

```
ori $s0,$s0,2304 # 2304ten= 0000 1001 0000 0000two
```

- τώρα ο `$s0` είναι:

Διευθύνσεις αλμάτων/διακλαδώσεων

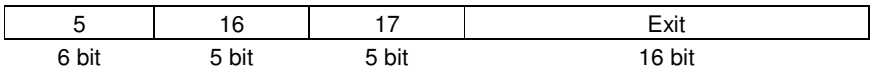
- Άλμα χωρίς συνθήκη

```
j 10000     # μετάβαση στη θέση 10000
```



- Διακλάδωση υπό συνθήκη

```
bne $s0,$s1,Exit    # Exit αν $s0 ≠ $s1
```



Διευθύνσεις αλμάτων/διακλαδώσεων

- **Πρόβλημα:** Αν οι διευθύνσεις του προγράμματος έπρεπε να χωρέσουν σε αυτό το πεδίο 16 bit
  - κανένα πρόγραμμα δεν θα μπορούσε να είναι μεγαλύτερο από 2^{16} !
- **Λύση:** Η εντολή διακλάδωσης να υπολογίζεται με βάση έναν καταχωρητή ως εξής:

Μετρητής προγράμματος (program counter) =
Καταχωρητής + Διεύθυνση διακλάδωσης

- Ποιος καταχωρητής θα χρησιμοποιηθεί ;

Σχετική διευθυνσιοδότηση

- **διευθυνσιοδότηση σχετική ως προς PC** (PC-relative addressing): Ένας τρόπος διευθυνσιοδότησης στον οποίο η διεύθυνση είναι το άθροισμα του μετρητή προγράμματος (PC) και μιας σταθεράς στην εντολή
- Οι εντολές διακλάδωσης υπό συνθήκη χρησιμοποιούν διευθυνσιοδότηση σχετική ως προς PC. Γιατί;
  - Ο προορισμός αυτών των εντολών είναι πιθανό να βρίσκεται κοντά στη διακλάδωση
- Οι εντολές άλματος και σύνδεσης (jump-and-link) χρησιμοποιούν άλλες μορφές διευθυνσιοδότησης. Γιατί;
  - Καλούν διαδικασίες που δεν έχουν λόγο να βρίσκονται κοντά στην κλήση
  - Μεγάλες διευθύνσεις για κλήσεις διαδικασιών χρησιμοποιώντας τη μορφή τύπου J (για τις εντολές jump και jump-and-link)

Σχετική διευθυνσιοδότηση

- Η διεύθυνση του MIPS είναι στην πραγματικότητα σχετική ως προς τη διεύθυνση της επόμενης εντολής (PC + 4), και όχι ως προς την τρέχουσα εντολή (PC)
- Επειδή όλες οι εντολές τού MIPS έχουν μήκος 4 byte:
  - Η απόσταση της διακλάδωσης αναφέρεται στον αριθμό των λέξεων μέχρι την επόμενη εντολή (και όχι στον αριθμό των byte)
  - Παρόμοια, το πεδίο 26 bit στις εντολές άλματος jump είναι επίσης διεύθυνση λέξης
    - αντιπροσωπεύει μια διεύθυνση byte των 28 bit

Αποστάσεις διακλάδωσης

```
Loop: sll  $t1,$s3,2
      add  $t1,$t1,$s6
      lw   $t0,0($t1)
      bne  $t0,$s5, Exit
      addi $s3,$s3,1
      j    Loop
Exit:
```

80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	…					

Διακλάδωση πολύ μακριά

- Η εντολή

```
beq $s0, $s1, L1
```

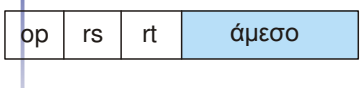
- Γίνεται

```
    bne $s0, $s1, L2
    j   L1
L2:
```

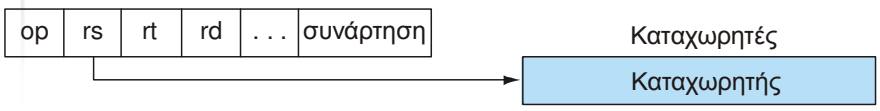
- και έτσι η διακλάδωση γίνεται πολύ πιο μακρινή
  - αντί για τα 16 bit σχετική απόσταση (offset) της beq τώρα έχει τα 26 bit της j

Τρόποι διευθυνσιοδότησης MIPS

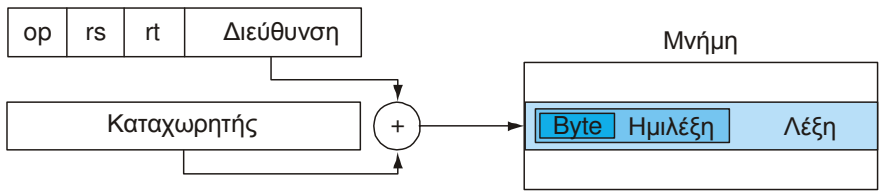
1. Άμεση διευθυνσιοδότηση



2. Διευθυνσιοδότηση μέσω καταχωρητή

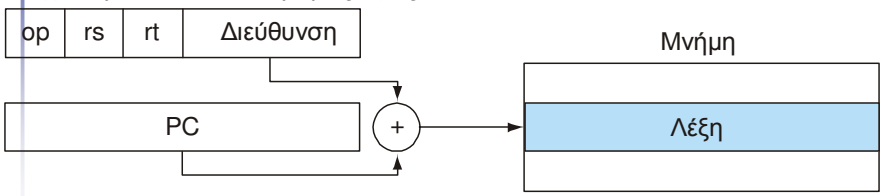


3. Διευθυνσιοδότηση βάσης

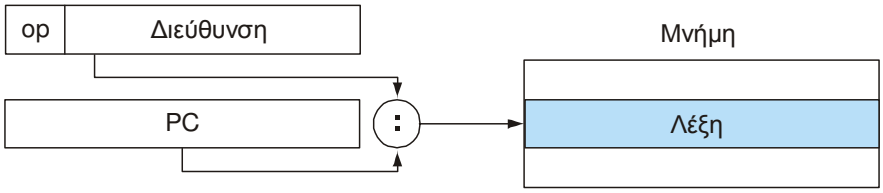


Τρόποι διευθυνσιοδότησης MIPS

4. Σχετική διευθυνσιοδότηση ως προς PC



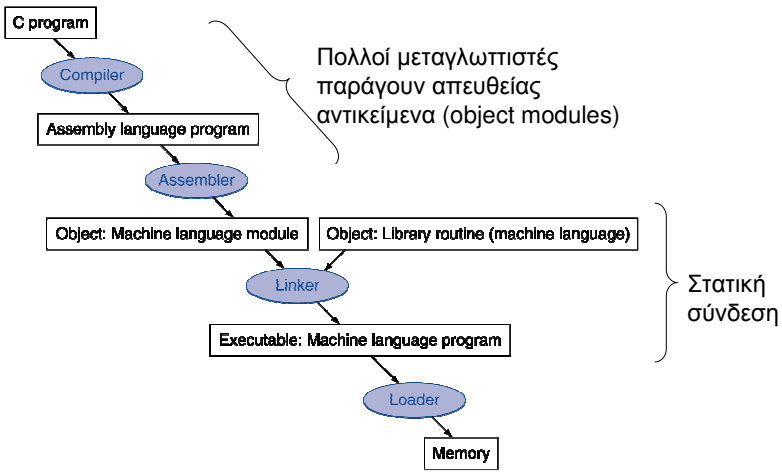
5. Ψευδο-απευθείας διευθυνσιοδότηση



Συμβολική γλώσσα MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Μεταφορά δεδομένων	load word	lw $s1,100($s2)	$s1 = Memory[$s2+100]	Λέξη από τη μνήμη σε καταχωρητή
	store word	sw $s1,100($s2)	Memory[$s2+100] = $s1	Λέξη από καταχωρητή στη μνήμη
	load half	lh $s1,100($s2)	$s1 = Memory[$s2+100]	Ημιλέξη από τη μνήμη σε καταχωρητή
	store half	sh $s1,100($s2)	Memory[$s2+100] = $s1	Ημιλέξη από καταχωρητή στη μνήμη
	load byte	lb $s1,100($s2)	$s1 = Memory[$s2+100]	Byte από τη μνήμη σε καταχωρητή
	store byte	sb $s1,100($s2)	Memory[$s2+100] = $s1	Byte από καταχωρητή στη μνήμη
	loadupper imm.	lui $s1,100	$s1 = 100 * 2^{16}	Φορτώνει σταθερά στα ανώτερα 16 bit
Άλμα χωρίς συνθήκη	jump	j 2500	μετάβαση στο 10000	Άλμα στη διεύθυνση προορισμού
	jump register	jr $ra	μετάβαση στο $ra	Για εναλλαγή και επιστροφή διαδικασίας
	jump and link	jal 2500	$ra = PC + 4· μετάβαση στο 10000	Για κλήση διαδικασίας

Μετάφραση και εκκίνηση προγράμματος



Ψευδοεντολές

- Οι περισσότερες εντολές παριστάνουν εντολές μηχανής μία-προς-μία
- Ψευδοεντολές (pseudoinstructions): δεν υπάρχουν - η χρήση των τους απλοποιεί το έργο του προγραμματισμού σε assembly!

```
move $t0, $t1     → add $t0, $zero, $t1
blt  $t0, $t1, L  → slt $at, $t0, $t1
                    bne $at, $zero, L
```

  - $at (ο καταχωρητής 1): assembler temporary

Παραγωγή object module

- Ο συμβολομεταφραστής (ή ο μεταγλωττιστής) μεταφράζει το πρόγραμμα σε εντολές μηχανής
- Παρέχει πληροφορία για την κατασκευή ενός πλήρους προγράμματος
  - Κεφαλίδα (header): περιγράφει τα περιεχόμενα της αντικειμενικής μονάδας
  - Τμήμα κειμένου (text segment): μεταφρασμένες εντολές
  - Τμήμα στατικών δεδομένων (static data segment): δεδομένα που κατανέμονται για όλη τη διάρκεια ζωής του προγράμματος
  - Πληροφορία επανατοποθέτησης (relocation info): για περιεχόμενα που εξαρτώνται από την απόλυτη θέση του φορτωμένου προγράμματος
  - Πίνακας συμβόλων (symbol table): ορισμοί και εξωτερικές αναφορές
  - Πληροφορία εκσφαλμάτωσης (debug info): για συσχέτιση με τον πηγαίο κώδικα

Σύνδεση object modules

- Παράγει μια εκτελέσιμη εικόνα (executable image)
  1. Ενώνει τα τμήματα
  2. Καθορίζει τις ετικέτες (προσδιορίζει τις διευθύνσεις τους)
  3. Διορθώνει (patching) τις αναφορές που εξαρτώνται από τη θέση και τις εξωτερικές αναφορές
- Θα μπορούσε να αφήσεις τις εξαρτήσεις θέσεων να ρυθμιστούν από έναν φορτωτή επανατοποθέτησης (relocating loader)
  - Αλλά με την εικονική μνήμη, αυτό δεν είναι απαραίτητο
  - Το πρόγραμμα μπορεί να φορτωθεί σε απόλυτη θέση στο χώρο των εικονικών διευθύνσεων

Φόρτωση προγράμματος

- Φόρτωση από το αρχείο εικόνας (image file) του δίσκου στη μνήμη
  1. Ανάγνωση της κεφαλίδας (header) για να καθοριστούν τα μεγέθη των τμημάτων
  2. Δημιουργία χώρου εικονικών διευθύνσεων
  3. Αντιγραφή του κώδικα (text) και των δεδομένων στη μνήμη
     - Ενεργοποίηση των καταχωρίσεων του πίνακα σελίδων ώστε να δημιουργηθούν σφάλματα σελίδας και να μεταφερθούν στη μνήμη
  4. Τοποθέτηση των ορισμάτων στη στοίβα
  5. Αρχικοποίηση καταχωρητών (και οι $sp, $fp, $gp)
  6. Άλμα στη ρουτίνα εκκίνησης
     - Αντιγράφει τα ορίσματα στους $a0, … και καλεί τη main
     - Όταν η main επιστρέψει, εκτελείται η κλήση συστήματος exit

Παράδειγμα

- Πρόγραμμα ταξινόμησης στη C
- Υλοποίηση του προγράμματος σε γλώσσα assembly του MIPS

Πρόγραμμα ταξινόμησης στη C

- Ταξινομεί έναν πίνακα ακεραίων, χρησιμοποιώντας τον αλγόριθμο ταξινόμησης φυσαλίδας (bubble sort)

```
void sort (int v[], int n)
{
  int i, j;
  for (i = 0; i < n; i += 1) {
    for (j = i - 1;
         j >= 0 && v[j] > v[j + 1];
         j -= 1) {
      swap(v,j);
    }
  }
}
```


Διαδικασία swap σε C

```
void swap(int v[], int k)
 {
   int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
 }
```

Διαδικασία swap σε assembly

- Κατανομή καταχωρητών:
  - v στον καταχωρητή $a0
  - k στον καταχωρητή $a1
  - temp στον καταχωρητή $t0

```
swap:
# Υπολογισμός της διεύθυνση του v[k]
sll $t1, $a1, 2          # καταχωρητής $t1 = k * 4
add $t1, $a0, $t1        # καταχωρητής $t1 = v + (k*4)
# Φόρτωση των στοιχείων v[k] και v[k+1]
lw  $t0, 0($t1)          # καταχωρητής $t0 (temp) = v[k]
lw  $t2, 4($t1)          # καταχωρητής $t2 = v[k + 1]
# Αποθήκευση νέων τιμών στα στοιχεία v[k] και v[k+1]
sw  $t2, 0($t1)          # v[k] = καταχωρητής $t2
sw  $t0, 4($t1)          # v[k+1] = καταχωρητής $t0(temp)
jr  $ra                  # επιστροφή στην καλούσα ρουτίνα
```


Πρόγραμμα sort σε assembly

- Κατανομή καταχωρητών:
  - v στον καταχωρητή $a0
  - n στον καταχωρητή $a1
  - i στον καταχωρητή $s0
  - j στον καταχωρητή $s1

Αποθήκευση καταχωρητών

```
Sort:
addi $sp,$sp,-20   # δημιουργία χώρου στη στοίβα για 5
                   # καταχωρητές
sw   $ra,16($sp)   # αποθήκευση του $ra στη στοίβα
sw   $s3,12($sp)   # αποθήκευση του $s3 στη στοίβα
sw   $s2,8($sp)    # αποθήκευση του $s2 στη στοίβα
sw   $s1,4($sp)    # αποθήκευση του $s1 στη στοίβα
sw   $s0,0($sp)    # αποθήκευση του $s0 στη στοίβα
```

Πρόγραμμα sort σε assembly

```
            Μετακίνηση παραμέτρων
move $s2,$a0          # αντιγραφή της παραμέτρου $a0 στον $s2
move $s3,$a1          # αντιγραφή της παραμέτρου $a1 στον $s3
            Εξωτερικός βρόχος
move $s0,$zero        # i = 0
for1tst:
slt  $t0,$s0,$a1      # καταχωρητής $t0=0 αν  $s0≥$a1 (i≥n)
beq  $t0,$zero,exit1  # μετάβαση στην exit1 αν $s0≥$a1 (i≥n)
```


Πρόγραμμα sort σε assembly

```
            Εσωτερικός βρόχος
addi $s1, $s0,-1          # j = i - 1
for2tst:
slti $t0,$s1,0            # καταχωρητής $t0= 1 αν $s1<0 (j<0)
bne  $t0,$zero,exit2      # μετάβαση στην exit2 αν $s1<0 (j<0)
sll  $t1,$s1,2            # καταχωρητής $t1 = j * 4
add  $t2,$a0,$t1          # καταχωρητής $t2 = v + (j * 4)
lw   $t3,0($t2)           # καταχωρητής $t3 = v[j]
lw   $t4,4($t2)           # καταχωρητής $t4 = v[j + 1]
slt  $t0,$t4,$t3          # καταχωρητής $t0 = 0 αν $t4 ≥"$t3
beq  $t0,$zero,exit2      # μετάβαση στην exit2 αν $t4 ≥ $t3
```

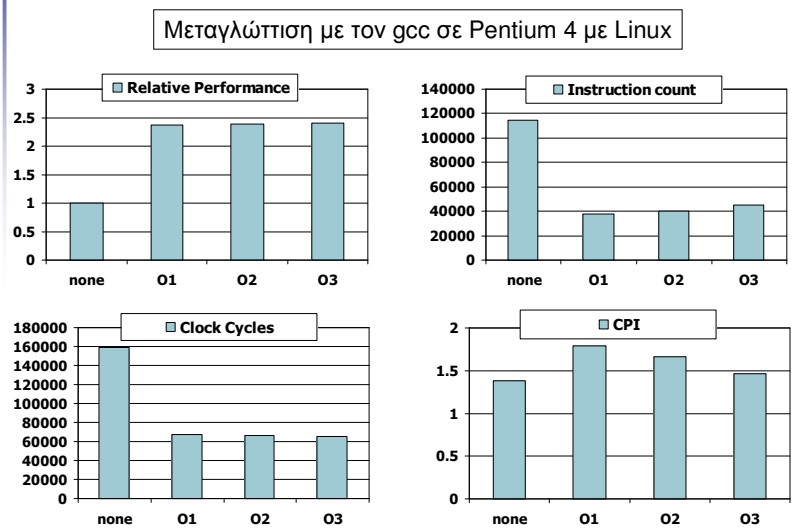
Πρόγραμμα sort σε assembly

```
      Μεταβίβαση παραμέτρων και κλήση της swap
move  $a0,$s2              # η πρώτη παράμετρος της swap είναι η v
move  $a1,$s1              # η δεύτερη παράμετρος είναι η j
jal   swap                 # κλήση της swap
      Εσωτερικός βρόχος
addi  $s1,$s1,-1           # j = j - 1
j     for2tst              # άλμα στον έλεγχο του εσωτερικού βρόχου
      Εξωτερικός βρόχος
exit2:
addi  $s0, $s0, 1          # i = i + 1
j     for1tst              # άλμα στον έλεγχο του εξωτερικού βρόχου
```

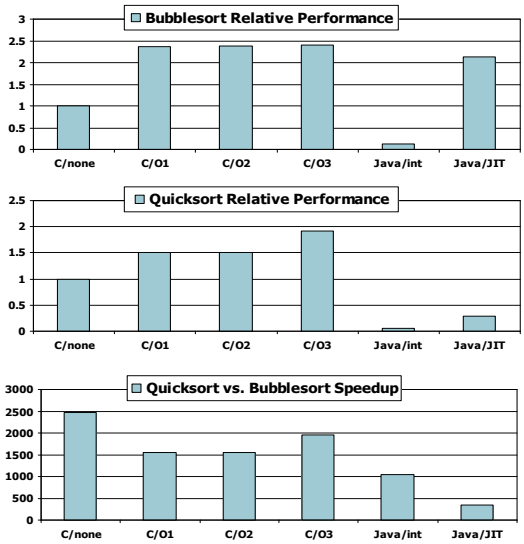

Πρόγραμμα sort σε assembly

```
      Επαναφορά καταχωρητών
exit1:
lw    $s0, 0($sp)   # επαναφορά του $s0 από τη στοίβα
lw    $s1, 4($sp)   # επαναφορά του $s1 από τη στοίβα
lw    $s2, 8($sp)   # επαναφορά του $s2 από τη στοίβα
lw    $s3,12($sp)   # επαναφορά του $s3 από τη στοίβα
lw    $ra,16($sp)   # επαναφορά του $ra από τη στοίβα
addi  $sp,$sp,20    # επαναφορά του δείκτη στοίβας
      Επιστροφή διαδικασίας
jr    $ra #         Επιστροφή στην καλούσα ρουτίνα
```

Βελτιστοποιήσεις του μεταγλωττιστή



Γλώσσα και αλγόριθμος



Πίνακες ή δείκτες

- Η αριθμοδότηση πινάκων (array indexing) περιλαμβάνει
  - Πολλαπλασιασμό του αριθμοδείκτη επί το μέγεθος του στοιχείου του πίνακα
  - Πρόσθεση στη διεύθυνση βάσης του πίνακα
- Οι δείκτες (pointers) αντιστοιχούν απευθείας σε διευθύνσεις μνήμης
  - Μπορεί να αποφύγουν τις δυσκολίες της αριθμοδότησης

Παράδειγμα: καθαρισμός & πίνακες

```
clear1(int array[], int size) {
  int i;
  for (i = 0; i < size; i += 1)
    array[i] = 0;
}
```

```
clear2(int *array, int size) {
  int *p;
  for (p = &array[0]; p < &array[size];
       p = p + 1)
    *p = 0;
}
```

```
       move $t0,$zero   # i = 0
loop1: sll $t1,$t0,2    # $t1 = i * 4
       add $t2,$a0,$t1  # $t2 =
                        #   &array[i]
       sw $zero, 0($t2) # array[i] = 0
       addi $t0,$t0,1   # i = i + 1
       slt $t3,$t0,$a1  # $t3 =
                        #   (i < size)
       bne $t3,$zero,loop1 # if (…)
                           # goto loop1
```

```
       move $t0,$a0     # p = & array[0]
       sll $t1,$a1,2    # $t1 = size * 4
       add $t2,$a0,$t1 # $t2 =
                        #   &array[size]
loop2: sw $zero,0($t0) # Memory[p] = 0
       addi $t0,$t0,4   # p = p + 4
       slt $t3,$t0,$t2 # $t3 =
                        #(p<&array[size])
       bne $t3,$zero,loop2 # if (…)
                           # goto loop2
```

Σύγκριση πινάκων και δεικτών

- Ο πολλαπλασιασμός μετατρέπεται σε ολίσθηση ("strength reduction")
- Η έκδοση με πίνακες απαιτεί η ολίσθηση να είναι μέσα στο βρόχο
  - Μέρος του υπολογισμού αριθμοδείκτη μετά την αύξηση του i
  - Σε αντίθεση με την αύξηση του δείκτη (pointer)
- Ο μεταγλωττιστής μπορεί να πετύχει το ίδιο αποτέλεσμα με την χρήση των δεικτών (pointers) από τον προγραμματιστή
  - Εξάλειψη επαγωγικής μεταβλητής (induction variable elimination)
  - Καλύτερα το πρόγραμμα να είναι πιο απλό και τη δύσκολη δουλειά να την κάνει ο μεταγλωττιστής

Ομοιότητες ARM & MIPS

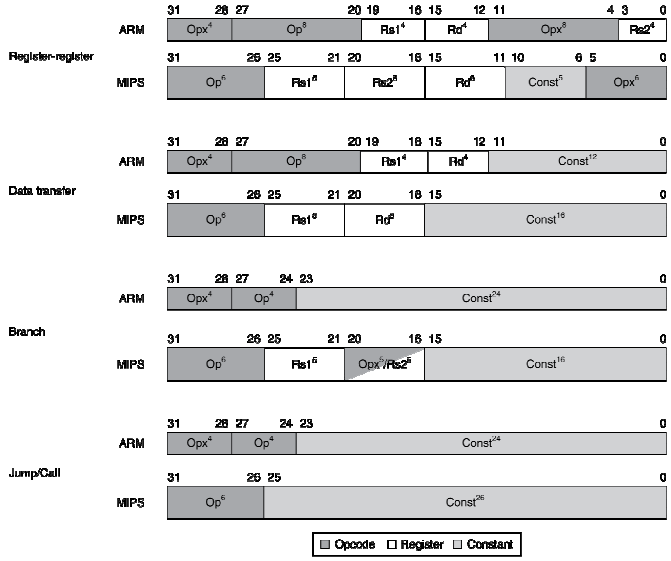
- ARM: ο δημοφιλέστερος ενσωματωμένος πυρήνας
- Παρόμοιο βασικό σύνολο εντολών με τον MIPS

	ARM	MIPS
Έτος ανακοίνωσης	1985	1985
Μέγεθος εντολής	32 bit	32 bit
Χώρος διευθύνσεων	32-bit επίπεδος	32-bit επίπεδος
Ευθυγράμμιση δεδομένων	Ευθυγραμμισμένα	Ευθυγραμμισμένα
Τρόποι διευθυνσιοδότησης δεδομένων	9	3
Καταχωρητές	15 GP × 32-bit	31 GP × 32-bit
Είσοδος/έξοδος	memory mapped	memory mapped

Σύγκριση και διακλάδωση στον ARM

- Χρησιμοποιεί κωδικούς συνθήκης για το αποτέλεσμα μιας αριθμητικής/λογικής εντολής
  - Αρνητικό (negative), μηδέν (zero), κρατούμενο (carry), υπερχείλιση (overflow)
  - Εντολές σύγκρισης (compare) που δίνουν τιμές στους κωδικούς συνθήκης χωρίς να διατηρούν το αποτέλεσμα
- Κάθε εντολή μπορεί να είναι υπό συνθήκη
  - 4 υψηλότερα bit της λέξης εντολής: τιμή της συνθήκης
  - Μπορεί να αποφύγει τις διακλαδώσεις πάνω από μεμονωμένες εντολές

Κωδικοποίηση εντολών

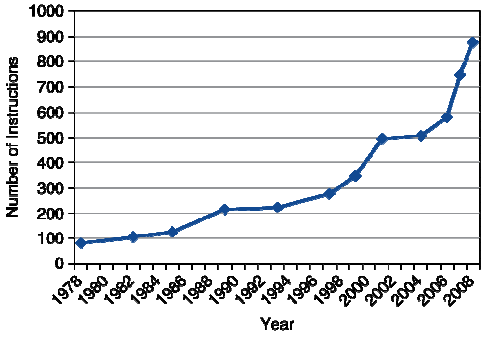


Πλάνες

- Ισχυρή εντολή ⇒ υψηλότερη απόδοση
  - Απαιτούνται λιγότερες εντολές
  - Αλλά οι σύνθετες εντολές είναι δύσκολο να υλοποιηθούν
    - Μπορεί να καθυστερήσουν όλες τις εντολές, ακόμη και τις πιο απλές
  - Οι μεταγλωττιστές είναι καλοί στο να παράγουν γρήγορο κώδικα με απλές εντολές
- Χρήση κώδικα συμβολικής γλώσσας για υψηλή απόδοση
  - Αλλά οι σύγχρονοι μεταγλωττιστές είναι καλύτεροι στο χειρισμό των σύγχρονων επεξεργαστών
  - Περισσότερες γραμμές κώδικα ⇒ περισσότερα σφάλματα και μικρότερη παραγωγικότητα

Πλάνες

- Αναδρομική συμβατότητα (backward compatibility) ⇒ το σύνολο εντολών δεν αλλάζει
  - Αλλά προστίθενται περισσότερες εντολές



σύνολο εντολών x86

Παγίδες

- Οι διαδοχικές λέξεις δεν βρίσκονται σε διαδοχικές διευθύνσεις
  - Αύξηση κατά 4, όχι κατά 1!
- Διατήρηση ενός δείκτη (pointer) προς μια αυτόματη μεταβλητή μετά την επιστροφή της διαδικασίας
  - π.χ., μεταβίβαση του δείκτη μέσω ενός ορίσματος
  - Ο δείκτης γίνεται άκυρος μετά το «άδειασμα» της στοίβας για τη διαδικασία

Σχεδιαστικές αρχές

1. Η απλότητα ευνοεί την κανονικότητα
2. Το μικρότερο είναι ταχύτερο
3. Κάνε τη συνηθισμένη περίπτωση γρήγορη
4. Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς

§2.19 Συμπερασματικές παρατηρήσεις

Συχνότητα εκτέλεσης εντολών

- Μέτρηση εκτελέσεων εντολών MIPS σε μετροπρογράμματα
  - Κάντε τη συνηθισμένη περίπτωση γρήγορη
  - Κάντε συμβιβασμούς

Κατηγορία εντολής	Παραδείγματα MIPS	SPEC2006 Int	SPEC2006 FP
Αριθμητικές	`add, sub, addi`	16%	48%
Μεταφοράς δεδομένων	`lw, sw, lb, lbu, lh, lhu, sb, lui`	35%	36%
Λογικές	`and, or, nor, andi, ori, sll, srl`	12%	4%
Διακλάδωσης υπό συνθήκη	`beq, bne, slt, slti, sltiu`	34%	8%
Άλματος	`j, jr, jal`	2%	0%