



ΟΡΓΑΝΩΣΗ ΚΑΙ ΣΧΕΔΙΑΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

Η διασύνδεση υλικού και λογισμικού



Κεφάλαιο 2

Εντολές: Η γλώσσα του υπολογιστή

Σύνολο εντολών

- Η γκάμα των εντολών ενός υπολογιστή
- Κάθε υπολογιστής έχει διαφορετικό σύνολο εντολών
 - Αλλά με πολλές κοινές πτυχές
- Οι πρώτοι υπολογιστές είχαν πολύ απλά σύνολα εντολών
 - Απλουστευμένη υλοποίηση
- Πολλοί σύγχρονοι υπολογιστές έχουν επίσης απλά σύνολα εντολών

«Αποθηκευμένο πρόγραμμα»

■ “The stored program concept”

- Η ιδέα ότι οι εντολές και τα δεδομένα πολλών τύπων μπορούν να αποθηκευτούν στη μνήμη ως αριθμοί
- Οδήγησε στον υπολογιστή αποθηκευμένου προγράμματος (stored program computer)

Το σύνολο εντολών του RISC-V

- Χρησιμοποιείται ως παράδειγμα σε ολόκληρο το βιβλίο
- Αναπτύχθηκε στο Berkeley ως ανοιχτή αρχιτεκτονική συνόλου εντολών (ISA)
- Πλέον η διαχείρισή του γίνεται από το ίδρυμα RISC-V Foundation (riscv.org)
- Αντιπροσωπευτική πολλών σύγχρονων αρχιτεκτονικών συνόλου εντολών (ISA)
 - Δείτε την Κάρτα Αναφοράς του RISC-V στο βιβλίο
- Παρόμοιες αρχιτεκτονικές συνόλου εντολών έχουν μεγάλο μερίδιο της αγοράς επεξεργαστών με ενσωματωμένους πυρήνες
 - Εφαρμογές σε ηλεκτρονικές συσκευές ευρείας κυκλοφορίας, εξοπλισμό δικτύου/αποθήκευσης, ψηφιακές φωτογραφικές μηχανές, εκτυπωτές, κ.ά.

Αριθμητικές πράξεις (λειτουργίες)

- Πρόσθεση και αφαίρεση, τρεις τελεστέοι
 - Δύο για τους αριθμούς κάθε πράξης και ένας για το αποτέλεσμα
add a, b, c // το άθροισμα των b και c τοποθετείται στην a
- Όλες οι αριθμητικές πράξεις (λειτουργίες) είναι αυτής της μορφής
- **Σχεδιαστική αρχή 1:** Η απλότητα ευνοεί την κανονικότητα
 - Η κανονικότητα απλουστεύει την υλοποίηση
 - Η απλότητα επιτρέπει υψηλότερη απόδοση με χαμηλότερο κόστος

Ένα αριθμητικό παράδειγμα

- Κώδικας C:

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

- Μεταγλωττισμένος κώδικας RISC-V:

```
add t0, g, h    // η προσωρινή μεταβλητή t0 περιέχει το g + h  
add t1, i, j    // η προσωρινή μεταβλητή t1 περιέχει το i + j  
sub f, t0, t1   // το f παίρνει το t0 - t1
```

Πίνακες εντολών στο βιβλίο (1)

Τελεστέοι του RISC-V

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	x0 – x31	Γρήγορες θέσεις για δεδομένα. Στον RISC-V, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις. Ο καταχωρητής x0 έχει πάντα την τιμή 0.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], ..., Memory[4294967292]	Προσπελάζονται μόνο από εντολές μεταφοράς δεδομένων. Ο RISC-V χρησιμοποιεί διευθύνσεις byte, και έτσι οι διευθύνσεις διαδοχικών λέξεων διαφέρουν κατά 4. Η μνήμη κρατάει δομές δεδομένων, πίνακες και διασκορπισμένους ("spilled") καταχωρητές, όπως αυτοί που αποθηκεύονται στις κλήσεις διαδικασιών.

Συμβολική γλώσσα του RISC-V

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	Add	add x5, x6, x7	$x5 = x6 + x7$	Τρεις τελεστέοι καταχωρητές πρόσθεση
	Subtract	sub x5, x6, x7	$x5 = x6 - x7$	Τρεις τελεστέοι καταχωρητές αφαίρεση
	Add immediate	addi x5, x6, 20	$x5 = x6 + 20$	Χρήση για πρόσθεση σταθερών
Μεταφορά δεδομένων	Load word	lw x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Λέξη από τη μνήμη σε καταχωρητή
	Load word, unsigned	lwu x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Απρόσημη λέξη από τη μνήμη σε καταχωρητή
	Store word	sw x5, 40(x6)	$\text{Memory}[x6 + 40] = x5$	Λέξη από καταχωρητή στη μνήμη
	Load halfword	lh x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Ημιλέξη από τη μνήμη σε καταχωρητή
	Load halfword unsigned	lhu x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Απρόσημη ημιλέξη από τη μνήμη σε καταχωρητή
	Store halfword	sh x5, 40(x6)	$\text{Memory}[x6 + 40] = x5$	Ημιλέξη από καταχωρητή στη μνήμη
	Load byte	lb x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Byte από τη μνήμη σε καταχωρητή
	Load byte unsigned	lbu x5, 40(x6)	$x5 = \text{Memory}[x6 + 40]$	Απρόσημο byte από τη μνήμη σε καταχωρητή
	Store byte	sb x5, 40(x6)	$\text{Memory}[x6 + 40] = x5$	Byte από καταχωρητή στη μνήμη
	Load reserved	lr.w x5, (x6)	$x5 = \text{Memory}[x6]$	Φόρτωση: πρώτο μισό μιας αδιαίρετης εναλλαγής
	Store conditional	sc.w x7, x5, (x6)	$\text{Memory}[x6] = x5; x7 = 0/1$	Αποθήκευση: δεύτερο μισό μιας αδιαίρετης εναλλαγής
	Load upper immediate	lui x5, 0x12345000	$x5 = 0x12345000$	Φόρτωση σταθεράς 20 bit ολισθημένης δεξιά κατά 12 bit

Πίνακες εντολών στο βιβλίο (2)

Λογικές πράξεις	And	and x5, x6, x7	$x5 = x6 \& x7$	Ιρεις τελεστέοι καταχωρητές: AND bit προς bit
	Inclusive or	or x5, x6, x8	$x5 = x6 x8$	Τρεις τελεστέοι καταχωρητές: OR bit προς bit
	Exclusive or	xor x5, x6, x9	$x5 = x6 ^ x9$	Τρεις τελεστέοι καταχωρητές: XOR bit προς bit
	And immediate	andi x5, x6, 20	$x5 = x6 \& 20$	AND bit προς bit καταχωρητή με σταθερά
	Inclusive or immediate	ori x5, x6, 20	$x5 = x6 20$	OR bit προς bit καταχωρητή με σταθερά
	Exclusive or immediate	xori x5, x6, 20	$x5 = x6 ^ 20$	XOR bit προς bit καταχωρητή με σταθερά
Ολισθήσεις	Shift left logical	sll x5, x6, x7	$x5 = x6 \& 20$	Αριστερή ολίσθηση με καταχωρητή
	Shift right logical	srl x5, x6, x7	$x5 = x6 20$	Δεξιά ολίσθηση με καταχωρητή
	Shift right arithmetic	sra x5, x6, x7	$x5 = x6 ^ 20$	Αριθμητική δεξιά ολίσθηση με καταχωρητή
	Shift left logical immediate	slli x5, x6, 3	$x5 = x6 << x7$	Αριστερή ολίσθηση με σταθερά
	Shift right logical immediate	srali x5, x6, 3	$x5 = x6 >> x7$	Δεξιά ολίσθηση με σταθερά
	Shift right arithmetic immediate	srai x5, x6, 3	$x5 = x6 >> x7$	Αριθμητική δεξιά ολίσθηση με σταθερά
Διακλάδωση υπό συνθήκη	Branch if equal	beq x5, x6, 100	if ($x5 == x6$) go to PC+100	Σχετική ως προς PC διακλάδωση αν καταχωρητές ίσο
	Branch if not equal	bne x5, x6, 100	if ($x5 != x6$) go to PC+100	Σχετική ως προς PC διακλάδωση αν καταχωρητές άνισο
	Branch if less than	blt x5, x6, 100	if ($x5 < x6$) go to PC+100	Σχετική ως προς PC διακλάδωση αν καταχωρητές μικρότερο
	Branch if greater or equal	bge x5, x6, 100	if ($x5 >= x6$) go to PC+100	Σχετική ως προς PC διακλάδωση αν καταχωρητές μεγαλύτερο ή ίσο
	Branch if less, unsigned	bltu x5, x6, 100	if ($x5 < x6$) go to PC+100	Σχετική ως προς PC διακλάδωση αν καταχωρητές μικρότερο, απρόσημοι
	Branch if greater or equal, unsigned	bgeu x5, x6, 100	if ($x5 >= x6$) go to PC+100	Σχετική ως προς PC διακλάδωση αν καταχωρητές μεγαλύτερο ή ίσο, απρόσημοι
Άλμα χωρίς συνθήκη	Jump and link	jal xl, 100	$xl = PC+4; \text{ go to } PC+100$	Σχετική ως προς PC κλήση διαδικασίας
	Jump and link register	jal r xl, 100(x5)	$xl = PC+4; \text{ go to } x5+100$	Επιστροφή από διαδικασία· έμμεση κλήση

Τελεστέοι καταχωρητές

- Στις αριθμητικές εντολές χρησιμοποιούνται τελεστέοι καταχωρητές
- Ο RISC-V έχει ένα αρχείο 32 καταχωρητών των 32 bit
 - Χρήση για δεδομένα που προσπελάζονται συχνά
 - διπλή λέξη (doubleword): δεδομένα των 64 bit
 - 32 καταχωρητές γενικού σκοπού των 64 bit: οι x0 έως x31
 - λέξη (word): δεδομένα των 32 bit
- Σχεδιαστική αρχή 2: Το μικρότερο είναι ταχύτερο
 - πρβλ. κύρια μνήμη: εκατομμύρια θέσεων

Καταχωρητές του RISC-V

- x0: η σταθερή τιμή 0
- x1: διεύθυνση επιστροφής
- x2: δείκτης στοίβας
- x3: καθολικός δείκτης
- x4: δείκτης νημάτων
- x5 – x7, x28 – x31: καταχωρητές προσωρινών τιμών
- x8: δείκτης πλαισίου
- x9, x18 – x27: αποθηκευμένοι καταχωρητές
- x10 – x11: ορίσματα/αποτελέσματα συναρτήσεων
- x12 – x17: ορίσματα συναρτήσεων

Ένα παράδειγμα με τελεστέους καταχωρητές

- Κώδικας C:

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

- Οι f, ..., j στους x19, x20, ..., x23

- Μεταγλωττισμένος κώδικας RISC-V:

add x5, x20, x21

add x6, x22, x23

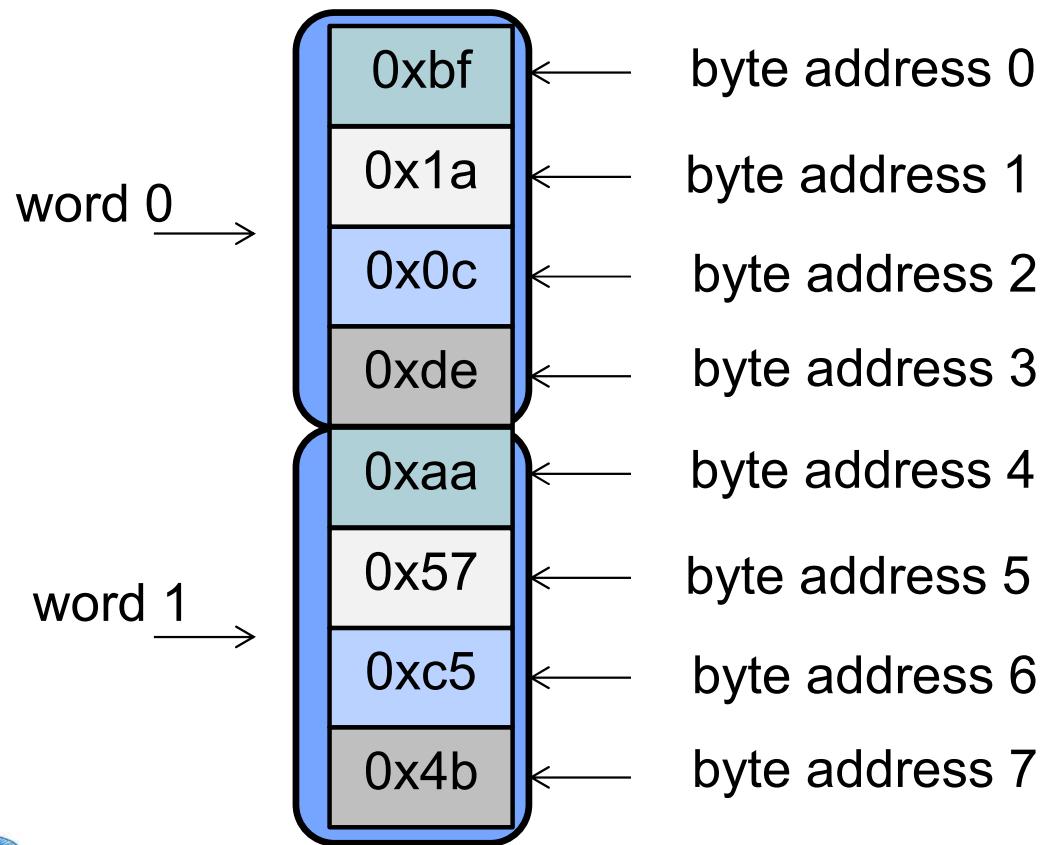
sub x19, x5, x6

Τελεστέοι μνήμης

- Η κύρια μνήμη χρησιμοποιείται για σύνθετα δεδομένα
 - Πίνακες, δομές, δυναμικά δεδομένα
- Για να εκτελεστούν αριθμητικές πράξεις (λειτουργίες)
 - Φορτώνονται τιμές από τη μνήμη στους καταχωρητές
 - Αποθηκεύεται το αποτέλεσμα από τον καταχωρητή στη μνήμη
- Η μνήμη είναι διευθυνσιοδοτημένη κατά byte
 - Κάθε διεύθυνση αντιστοιχεί σε ένα byte των 8 bit
- Η αρχιτεκτονική RISC-V είναι αρχιτεκτονική «μικρού άκρου» (little endian)
 - Διεύθυνση λέξης στο λιγότερο σημαντικό byte (άκρο)
 - πρβλ. Επεξεργαστές μεγάλου άκρου (big endian): Διεύθυνση λέξης στο περισσότερο σημαντικό byte (άκρο)
- Στην αρχιτεκτονική RISC-V δεν είναι απαραίτητο να είναι ευθυγραμμισμένες στη μνήμη
 - Αντίθεση με κάποιες άλλες αρχιτεκτονικές

Διευθύνσεις byte και λέξεων

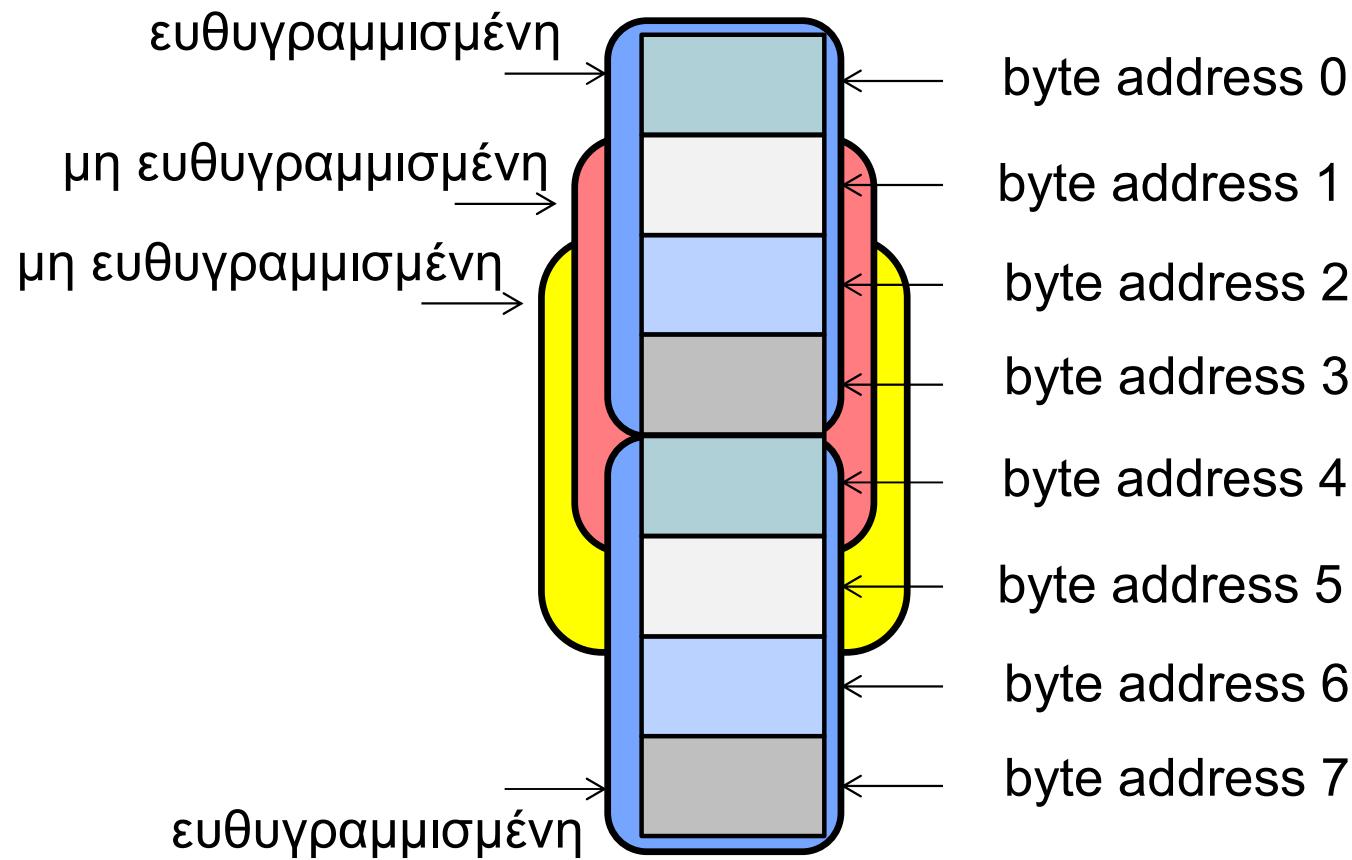
- Κάθε λέξη των 32 bit αποτελείται από 4 byte
 - Κάθε byte έχει δική του διεύθυνση



Κεφάλαιο 2 — Εντολές: η γλώσσα του υπολογιστή — 13

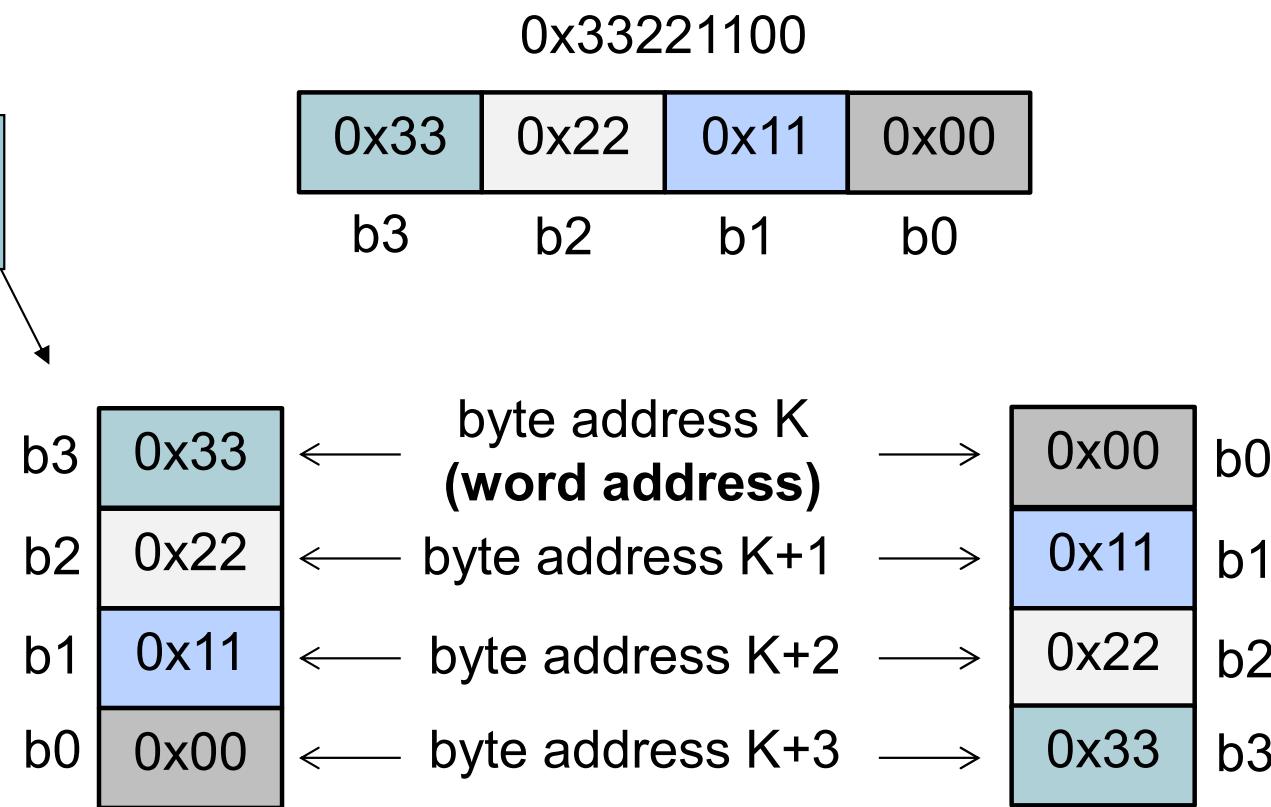
Ευθυγράμμιση

- Ευθυγραμμισμένες (aligned) και μη ευθυγραμμισμένες (unaligned) λέξεις



“Endianess” – big vs. little

- Πως αποθηκεύονται τα byte μιας λέξης



Ένα παράδειγμα με τελεστέους μνήμης

- Κώδικας C:

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

- Η h στον x21, η διεύθυνση βάσης του A στον x22

- Μεταγλωττισμένος κώδικας RISC-V:

- Για τον αριθμοδείκτη 8 απαιτείται σχετική απόσταση 32
 - 4 byte ανά λέξη

lw x9, 32(x22)

add x9, x21, x9

sw x9, 48(x22)

Καταχωρητές ή μνήμη;

- Η προσπέλαση των καταχωρητών είναι ταχύτερη από την προσπέλαση της μνήμης
- Η εκτέλεση πράξεων (λειτουργιών) σε δεδομένα που βρίσκονται στη μνήμη συνεπάγεται εντολές φόρτωσης και αποθήκευσης
 - Πρέπει να εκτελεστούν περισσότερες εντολές
- Ο μεταγλωττιστής πρέπει να χρησιμοποιεί τους καταχωρητές για τις μεταβλητές όσο το δυνατόν περισσότερο
 - Διασκορπίζει στη μνήμη μόνο όσες μεταβλητές χρησιμοποιούνται σπανιότερα
 - Είναι σημαντική η βελτιστοποίηση της χρήσης των καταχωρητών!

Άμεσοι τελεστέοι

- Σταθερές τιμές δεδομένων που καθορίζονται σε μια εντολή
addi x22 , x22 , 4
- Επιτάχυνση της πιο κοινής (συνηθισμένης) περίπτωσης
 - Συχνά εμφανίζονται σταθερές με μικρή τιμή
 - Με έναν άμεσο τελεστέο αποφεύγεται μια εντολή φόρτωσης

Μη προσημασμένοι δυαδικοί ακέραιοι αριθμοί

- Με δεδομένο έναν αριθμό των n bit

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_12^1 + x_02^0$$

- Εύρος τιμών: Από 0 έως $+2^n - 1$
- Παράδειγμα
 - 0000 0000 ... 0000 1011₂
 $= 0 + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
 $= 0 + \dots + 8 + 0 + 2 + 1 = 11_{10}$
- Με 32 bit: 0 ως +4 294 967 295
- Με 64 bit: 0 έως +18 446 774 073 709 551 615

Προσημασμένοι ακέραιοι σε μορφή συμπληρώματος ως προς 2

- Με δεδομένο έναν αριθμό των n bit

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_12^1 + x_02^0$$

- Εύρος τιμών: Από -2^{n-1} έως $+2^{n-1} - 1$
- Παράδειγμα
 - 1111 1111 ... 1111 1100₂
 $= -1 \times 2^{31} + 1 \times 2^{30} + \dots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
 $= -2\ 147\ 483\ 648 + 2\ 147\ 483\ 644 = -4_{10}$
- Με 32 bit: $-2\ 147\ 483\ 648$ ως $+2\ 147\ 483\ 647$
- Με 64 bit: $-9\ 223\ 372\ 036\ 854\ 775\ 808$
έως $9\ 223\ 372\ 036\ 854\ 775\ 807$

Προσημασμένοι ακέραιοι σε μορφή συμπληρώματος ως προς 2

- To bit 31 είναι το bit προσήμου
 - 1 για αρνητικούς αριθμούς
 - 0 για μη αρνητικούς αριθμούς
- Δεν αναπαρίσταται το $-(-2^n - 1)$
- Οι μη αρνητικοί αριθμοί αναπαρίστανται με τον ίδιο τρόπο σε μη προσημασμένη μορφή και σε μορφή συμπληρώματος ως προς 2
- Μερικοί συγκεκριμένοι αριθμοί
 - 0: 0000 0000 ... 0000
 - -1: 1111 1111 ... 1111
 - Ο πιο αρνητικός: 1000 0000 ... 0000
 - Ο πιο θετικός: 0111 1111 ... 1111

Υπολογισμός αντίθέτου

- Αντιστροφή των bit και πρόσθεση του 1
 - Αντιστροφή σημαίνει $1 \rightarrow 0, 0 \rightarrow 1$

$$\begin{aligned}x + \bar{x} &= 1111\dots111_2 = -1 \\ \bar{x} + 1 &= -x\end{aligned}$$

- Παράδειγμα: Βρείτε τον αντίθετο του +2
 - $+2 = 0000\ 0000\dots0010_{\text{two}}$
 - $-2 = 1111\ 1111\dots1101_{\text{two}} + 1$
 $= 1111\ 1111\dots1110_{\text{two}}$

Επέκταση προσήμου

- Αναπαράσταση ενός αριθμού με χρήση περισσότερων bit
 - Διατηρείται η αριθμητική τιμή
- Αναπαραγωγή του bit προσήμου στα αριστερά
 - πρβλ. μη προσημασμένες τιμές: επέκταση με μηδενικά
- Παραδείγματα: 8 bit σε 16 bit
 - +2: 0000 0010 => 0000 0000 0000 0010
 - -2: 1111 1110 => 1111 1111 1111 1110
- Στο σύνολο εντολών του RISC-V
 - 1b: φόρτωση byte με επέκταση προσήμου
 - 1bu: φόρτωση byte με επέκταση μηδενικού

Αναπαράσταση εντολών

- Οι εντολές κωδικοποιούνται σε δυαδική μορφή
 - Λέγονται κώδικας μηχανής
- Εντολές του RISC-V
 - Κωδικοποιούνται ως λέξεις εντολών 32 bit
 - Λίγες μορφές με τις οποίες κωδικοποιούνται κώδικες λειτουργιών (operation codes), αριθμοί καταχωρητών, ...
 - Κανονικότητα!

Δεκαεξαδική αναπαράσταση

■ Βάση το 16

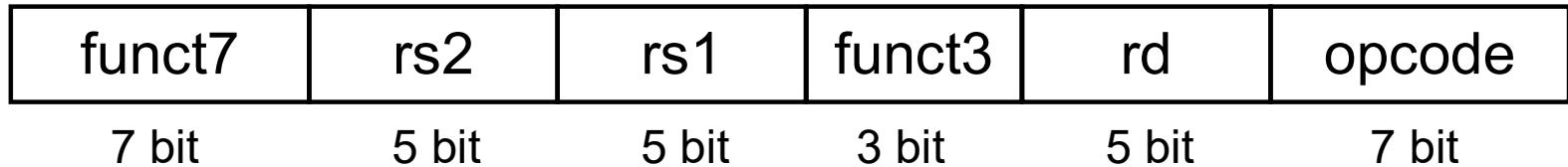
- Συμπυκνωμένη αναπαράσταση σειρών bit
- 4 bit ανά δεκαεξαδικό ψηφίο

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

Εντολές μορφής R του MIPS



Πεδία εντολών

- opcode: κωδικός λειτουργίας
- rd: αριθμός καταχωρητή τελεστέου προορισμού
- funct3: κωδικός συνάρτησης 3 bit (πρόσθετος κωδ. λειτουργίας)
- rs1: ο αριθμός καταχωρητή του πρώτου τελεστέου προέλευσης
- rs2: ο αριθμός καταχωρητή του δεύτερου τελεστέου προέλευσης
- funct7: κωδικός συνάρτησης 7 bit (πρόσθετος κωδ. λειτουργίας)

Ένα παράδειγμα με τη μορφή R

funct7	rs2	rs1	funct3	rd	opcode
7 bit	5 bit	5 bit	3 bit	5 bit	7 bit

add x9, x20, x21

0	21	20	0	9	51
---	----	----	---	---	----

0000000	10101	10100	000	01001	0110011
---------	-------	-------	-----	-------	---------

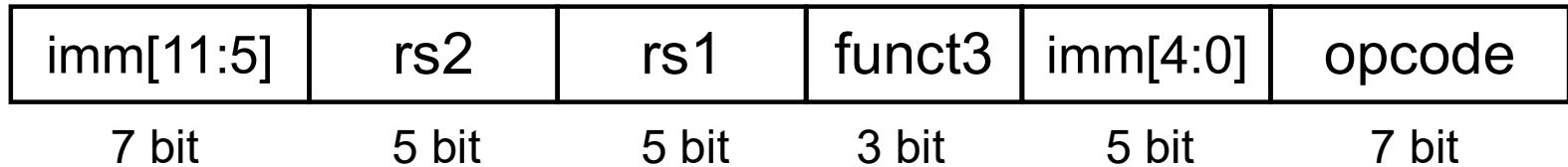
0000 0001 0101 1010 0000 0100 1011 0011_{two} =
015A04B3₁₆

Εντολές μορφής I του RISC-V

άμεσος	rs1	funct3	rd	opcode
12 bit	5 bit	3 bit	5 bit	7 bit

- Άμεσες αριθμητικές εντολές και εντολές φόρτωσης
 - rs1: ο αριθμός καταχωρητή του τελεστέου προέλευσης ή της διεύθυνσης βάσης
 - immediate (άμεσος): σταθερός τελεστέος, ή σχετική απόσταση που προστίθεται στη διεύθυνση βάσης
 - σε μορφή συμπληρώματος ως προς 2, με επέκταση προσήμου
- Σχεδιαστική αρχή 3: Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς
 - Οι διαφορετικές μορφές κάνουν πιο περίπλοκη την αποκωδικοποίηση, αλλά διευκολύνουν την ομοιομορφία των εντολών 32 bit
 - Οι μορφές των εντολών πρέπει να διατηρούνται όσο το δυνατόν πιο παρόμοιες

Εντολές μορφής S του RISC-V



- Διαφορετική μορφή άμεσου τελεστέου για τις εντολές αποθήκευσης
 - rs1: ο αριθμός καταχωρητή της διεύθυνσης βάσης
 - rs2: ο αριθμός καταχωρητή του τελεστέου προέλευσης
 - immediate (άμεσος τελεστέος): σχετική απόσταση που προστίθεται στη διεύθυνση βάσης
 - Ο διαχωρισμός έγινε προκειμένου τα πεδία rs1 και rs2 να βρίσκονται πάντα στην ίδια θέση

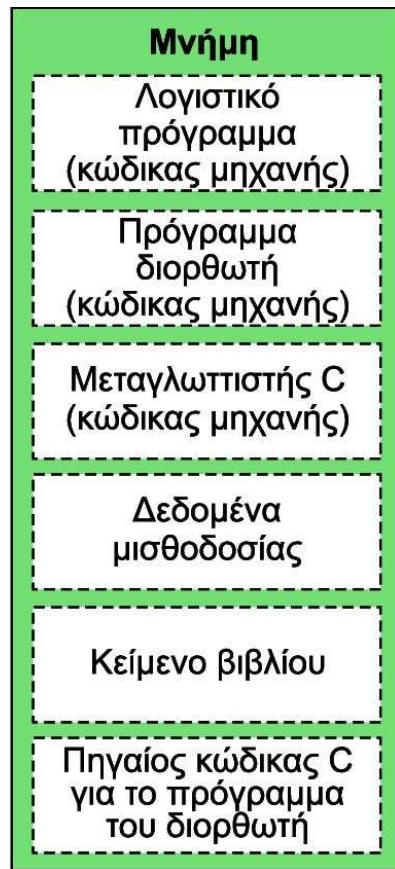
Παραδείγματα εντολών μορφών R, I, S

Εντολή	Μορφή	funct7	rs2	rs1	funct3	rd	opcode
add (add)	R	0000000	reg	reg	000	reg	0010011
sub (sub)	R	0100000	reg	reg	000	reg	0010011
Εντολή	Μορφή	immediate		rs1	funct3	rd	opcode
add (add immediate)	I	σταθερά		reg	000	reg	0010011
lw (load word)	I	διεύθυνση		reg	010	reg	0000011
Εντολή	Μορφή	immediate	rs2	rs1	funct3	immediate	opcode
sw (store word)	S	διεύθυνση	reg	reg	010	διεύθυνση	0100011

Υπολογιστές αποθηκευμένων προγραμμάτων

η ΓΕΝΙΚΗ εικόνα

Επεξεργαστής



- Οι εντολές αναπαρίστανται σε δυαδική μορφή, όπως τα δεδομένα
- Οι εντολές και τα δεδομένα αποθηκεύονται στη μνήμη
- Τα προγράμματα μπορούν να επενεργούν σε άλλα προγράμματα
 - π.χ. μεταγλωτιστές, προγράμματα σύνδεσης, ...
- Λόγω της συμβατότητας των δυαδικών μορφών, τα μεταγλωτισμένα προγράμματα μπορούν να χρησιμοποιούνται σε διαφορετικούς υπολογιστές
 - Τυποποιημένες αρχιτεκτονικές συνόλου εντολών (ISA)

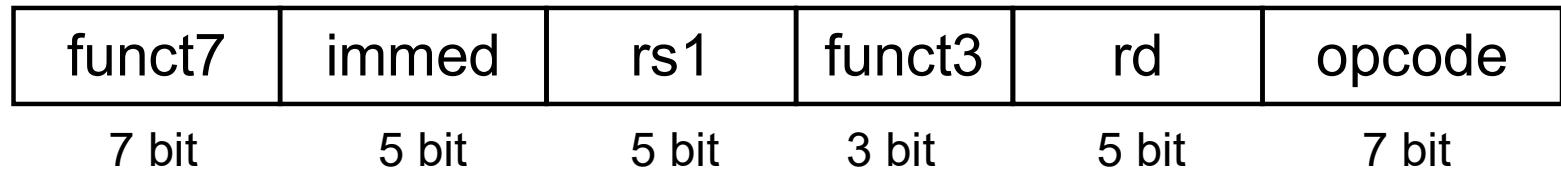
Λογικές λειτουργίες (πράξεις)

- Εντολές για χειρισμούς κατά bit

Λειτουργία/πράξη	C	Java	RISC-V
Αριστερή ολίσθηση (shift left)	<<	<<	slli
Δεξιά ολίσθηση (shift right)	>>	>>>	srlti
AND bit προς bit	&	&	and, andi
OR bit προς bit			or, ori
XOR bit προς bit	^	^	xor, xori
NOT bit προς bit	~	~	

- Χρήσιμο για την εξαγωγή και εισαγωγή ομάδων από bit σε μια λέξη

Λειτουργίες (πράξεις) ολίσθησης



- immed: κατά πόσες θέσεις γίνεται ολίσθηση
- Shift left logical — Αριστερή λογική ολίσθηση
 - Αριστερή ολίσθηση και συμπλήρωση με μηδενικά (0) bit
 - $slli$ i κατά i bit ισοδυναμεί με πολλαπλασιασμό με 2^i
- Shift right logical — Δεξιά λογική ολίσθηση
 - Δεξιά ολίσθηση και συμπλήρωση με μηδενικά (0) bit
 - $srli$ i κατά i bit ισοδυναμεί με διαίρεση με 2^i (μόνο για μη προσημασμένους αριθμούς)

Λειτουργία AND

- Χρήσιμη για την εφαρμογή μάσκας bit σε μια λέξη
 - Επιλογή μερικών bit, τα υπόλοιπα τίθενται ίσα με 0 and x9,x10,x11

x10	00000000	00000000	00001101	11000000
x11	00000000	00000000	00111100	00000000
x9	00000000	00000000	00001100	00000000

Λειτουργία OR

- Χρήσιμη για τη συμπερίληψη bit σε μια λέξη
 - Ορισμός μερικών bit ίσων με 1, τα άλλα παραμένουν αμετάβλητα

or x9, x10, x11

x10	00000000 00000000 00001101 11000000
x11	00000000 00000000 00111100 00000000
x9	00000000 00000000 00111101 11000000

Λειτουργία XOR

- Λειτουργία (πράξη) διαφοράς
 - Ορισμός μερικών bit ίσων με 1, τα άλλα παραμένουν αμετάβλητα

`xor x9,x10,x12 // λειτουργία NOT`

x10	00000000 00000000 00011011 11000000
x12	11111111 11111111 11111111 11111111
x9	11111111 11111111 11110010 00111111

Λειτουργίες που εκτελούνται υπό συνθήκη

- Αν μια συνθήκη είναι αληθής, διακλάδωση σε μια εντολή με μια συγκεκριμένη ετικέτα
 - Ειδάλλως, συνεχίζεται η ακολουθιακή εκτέλεση
- **beq rs1, rs2, L1**
 - αν ($rs1 == rs2$), τότε διακλάδωση στην εντολή με ετικέτα L1
- **bne rs1, rs2, L1**
 - αν ($rs1 != rs2$), τότε διακλάδωση στην εντολή με ετικέτα L1

Μεταγλωττιση εντολών If

Κώδικας C:

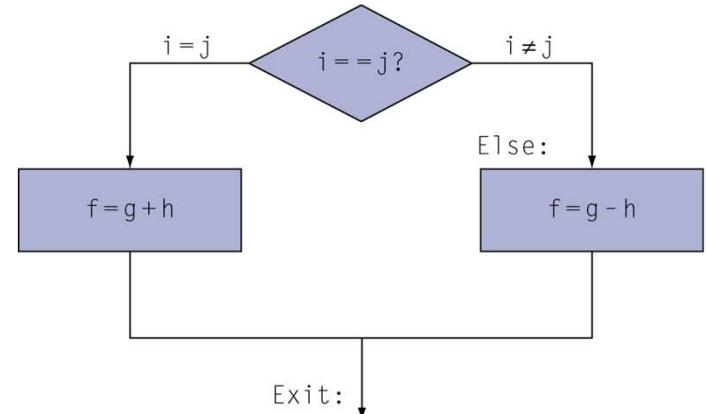
```
if (i==j) f = g+h;  
else f = g-h;
```

- Οι f, g, ... στους x19, x20, ...

Μεταγλωττισμένος κώδικας RISC-V:

```
bne x22, x23, Else  
add x19, x20, x21  
beq x0,x0,Exit // χωρίς συνθήκη  
Else: sub x19, x20, x21  
Exit: ...
```

Ο συμβολομεταφραστής
υπολογίζει τις διευθύνσεις



Μεταγλωττιση εντολών με βρόχους

■ Κώδικας C:

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

- Η i στον x22, η k στον x24, η διεύθυνση βασης του save στον x25

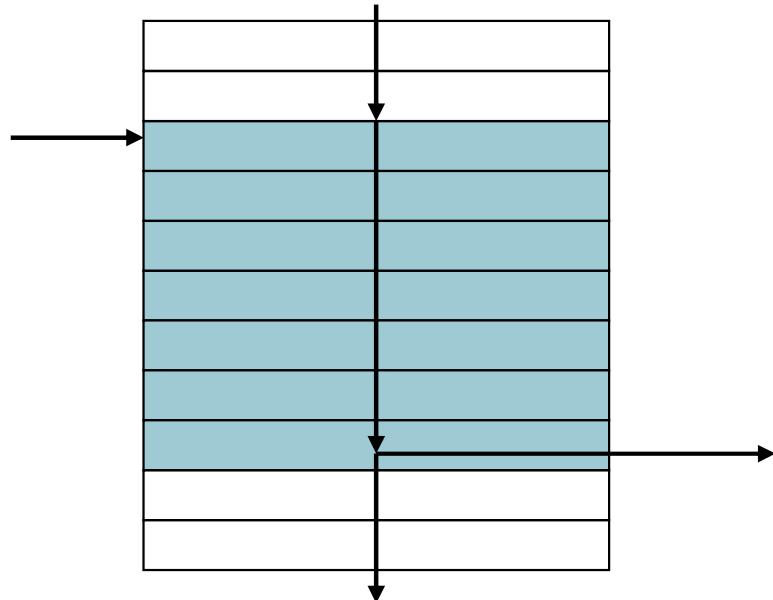
■ Μεταγλωττισμένος κώδικας RISC-V:

```
Loop: slli x10, x22, 2  
       add x10, x10, x25  
       lw x9, 0(x10)  
       bne x9, x24, Exit  
       addi x22, x22, 1  
       beq x0, x0, Loop
```

Exit: ...

Βασικά μπλοκ

- Το βασικό μπλοκ (basic block) είναι μια ακολουθία εντολών
 - χωρίς διακλαδώσεις (εκτός ίσως από το τέλος)
 - χωρίς προορισμούς διακλάδωσης (εκτός πιθανώς από την αρχή)



- Ο μεταγλωττιστής αναγνωρίζει τα βασικά μπλοκ για να τα βελτιστοποιήσει
- Οι προηγμένοι επεξεργαστής μπορούν να επιταχύνουν την εκτέλεση των βασικών μπλοκ

Άλλες λειτουργίες που εκτελούνται υπό συνθήκη

- **blt rs1, rs2, L1**
 - αν ($rs1 < rs2$), τότε διακλάδωση στην εντολή με ετικέτα L1
- **bge rs1, rs2, L1**
 - αν ($rs1 \geq rs2$), τότε διακλάδωση στην εντολή με ετικέτα L1
- Παράδειγμα
 - if ($a > b$) a += 1;
 - Η a στον x22, η b στον x23
bge x23, x22, Exit // διακλάδωση αν $b \geq a$
addi x22, x22, 1

Exit:

Προσημασμένοι και μη προσημασμένοι (απρόσημοι) αριθμοί

- Σύγκριση προσημασμένων αριθμών: blt, bge
- Σύγκριση μη προσημασμένων αριθμών: bltu, bgeu
- Παράδειγμα
 - $x22 = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$
 - $x23 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001$
 - $x22 < x23 // \text{ με πρόσημο}$
 - $-1 < +1$
 - $x22 > x23 // \text{ χωρίς πρόσημο}$
 - $+4\ 294\ 967\ 295 > +1$

Κλήσεις διαδικασιών

■ Βήματα που απαιτούνται

1. Τοποθέτηση των παραμέτρων στους καταχωρητές x10 έως x17
2. Μεταβίβαση του ελέγχου στη διαδικασία
3. Απόκτηση των πόρων αποθήκευσης που χρειάζεται η διαδικασία
4. Εκτέλεση των λειτουργιών της διαδικασίας
5. Τοποθέτηση της τιμής του αποτελέσματος στον καταχωρητή όπου το καλούν πρόγραμμα μπορεί να την προσπελάσει
6. Επιστροφή στο σημείο κλήσης (διεύθυνση στον x1)

Εντολές κλήσεων διαδικασιών

- Κλήση διαδικασίας: Jump and link — Άλμα και σύνδεση
`jal x1, ProcedureLabel`
 - Στον x1 η διεύθυνση της επόμενης εντολής
 - Άλμα στη διεύθυνση προορισμού
 - (αν ο καταχωρητής επιστροφής είναι ο x0 τότε πρόκειται για ένα απλό άλμα χωρίς συνθήκη – ισοδύναμο εντολής goto της C)
- Επιστροφή της διαδικασίας: jump and link register
`jalr x0, 0(x1)`
 - Όπως η jal, αλλά κάνει άλμα στο 0 + διεύθυνση που υπάρχει στον x1
 - Χρήση του x0 ως rd (ο x0 δεν επιτρέπεται να τροποποιηθεί)
 - Μπορεί επίσης να χρησιμοποιηθεί για άλματα σε υπολογισμένους μετρητές
 - π.χ. για εντολές case/switch



Ένα παράδειγμα με μια διαδικασία-φύλλο (leaf procedure)

■ Κώδικας C:

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

- Τα ορίσματα g, ..., j στους x10, ..., x13
- Η f στον x20
- Οι προσωρινές τιμές στους x5, x6
- Οι x5, x6, x20 πρέπει να αποθηκευτούν στη στοίβα

Ένα παράδειγμα με μια διαδικασία-φύλλο (leaf procedure)

■ Κώδικας RISC-V:

leaf_example:

```
addi sp,sp,-12      // Αποθήκευση των x5, x6, x20 στη στοίβα
sw   x5,8(sp)
sw   x6,4(sp)
sw   x20,0(sp)
add  x5,x10,x11    // x5 = g + h
add  x6,x12,x13    // x6 = i + j
sub  x20,x5,x6      // f = x5 - x6
addi x10,x20,0       // αντιγραφή της f στον καταχωρητή επιστροφής
lw   x20,0(sp)        // Επαναφορά των x5, x6, x20 από τη στοίβα
lw   x6,4(sp)
lw   x5,8(sp)
addi sp,sp,12
jalr x0,0(x1)        // Επιστροφή στον καλούντα
```

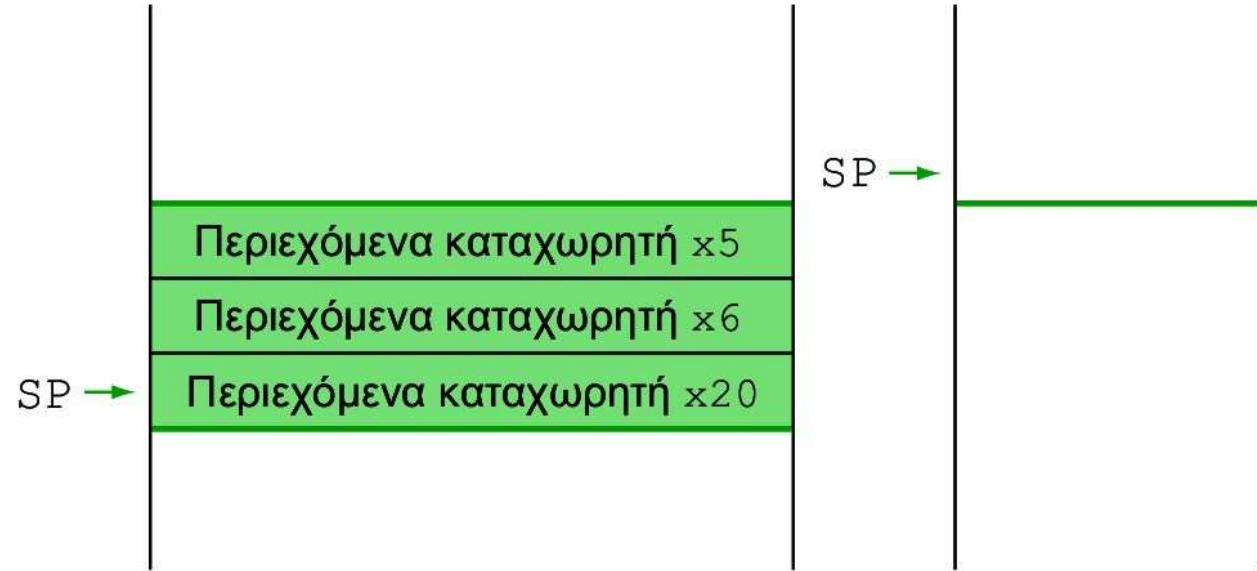
Τοπικά δεδομένα στη στοίβα

Υψηλή διεύθυνση

SP →

Χαμηλή διεύθυνση

(α)



(β)

(γ)

Χρήση καταχωρητών

- x5 – x7, x28 – x31: προσωρινοί (temporary) καταχωρητές
 - Δεν διατηρούνται από την καλούμενη διαδικασία
- x8 – x9, x18 – x27: αποθηκευμένοι καταχωρητές
 - Εφόσον χρησιμοποιηθούν, η καλούμενη διαδικασία τους αποθηκεύει και τους επαναφέρει

Διαδικασίες που δεν είναι διαδικασίες-φύλλα

- Δηλαδή διαδικασίες που καλούν άλλες διαδικασίες
- Για ένθετη κλήση, η καλούσα διαδικασία πρέπει να αποθηκεύσει στη στοίβα:
 - Τη διεύθυνση επιστροφής της
 - Ορίσματα και προσωρινές τιμές που τυχόν της χρειάζονται μετά από την κλήση
- Επαναφορά από τη στοίβα μετά από την κλήση

Ένα παράδειγμα με μια διαδικασία που καλεί μια άλλη (non-leaf procedure)

- Κώδικας C:

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

- Το όρισμα n στον x10
- Το αποτέλεσμα στον x10

Ένα παράδειγμα με μια διαδίκασία που καλεί μια άλλη (non-leaf procedure)

■ Κώδικας RISC-V:

fact:

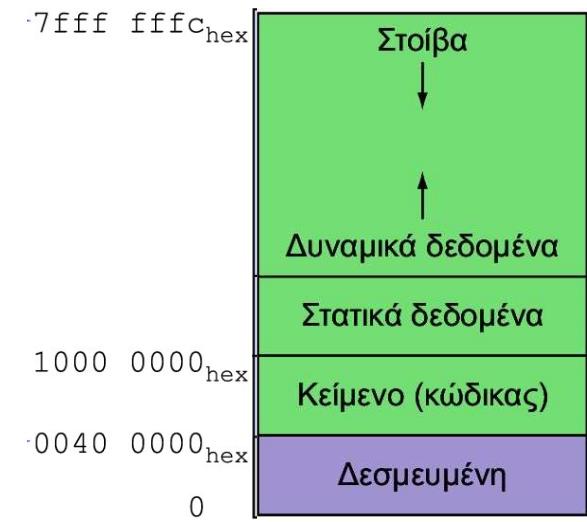
addi sp,sp,-8	Αποθήκευση της διεύθυνσης επιστροφής και της n στη στοίβα
sw x1,4(sp)	
sw x10,0(sp)	
addi x5,x10,-1	$x5 = n - 1$
bgt x5,x0,L1	αν $n \geq 1$, μετάβαση στην L1
addi x10,x0,1	ειδάλλως, η τιμή επιστροφής τίθεται ίση με 1
addi sp,sp,8	Εξαγωγή από τη στοίβα, δεν χρειάζεται επαναφορά των τιμών επιστροφής
jalr x0,0(x1)	
L1: addi x10,x10,-1	$n = n - 1$
jal x1,fact	κλήση της fact($n-1$)
addi x6,x10,0	μετακίνηση του αποτελέσματος της fact($n - 1$) στον x6
lw x10,0(sp)	Επαναφορά της τιμής n του καλούντα
lw x1,4(sp)	Επαναφορά της διεύθυνσης επιστροφής του καλούντα
addi sp,sp,8	Εξαγωγή από τη στοίβα
mul x10,x10,x6	επιστροφή της τιμής n * fact($n-1$)
jalr x0,0(x1)	επιστροφή

Διάταξη της μνήμης

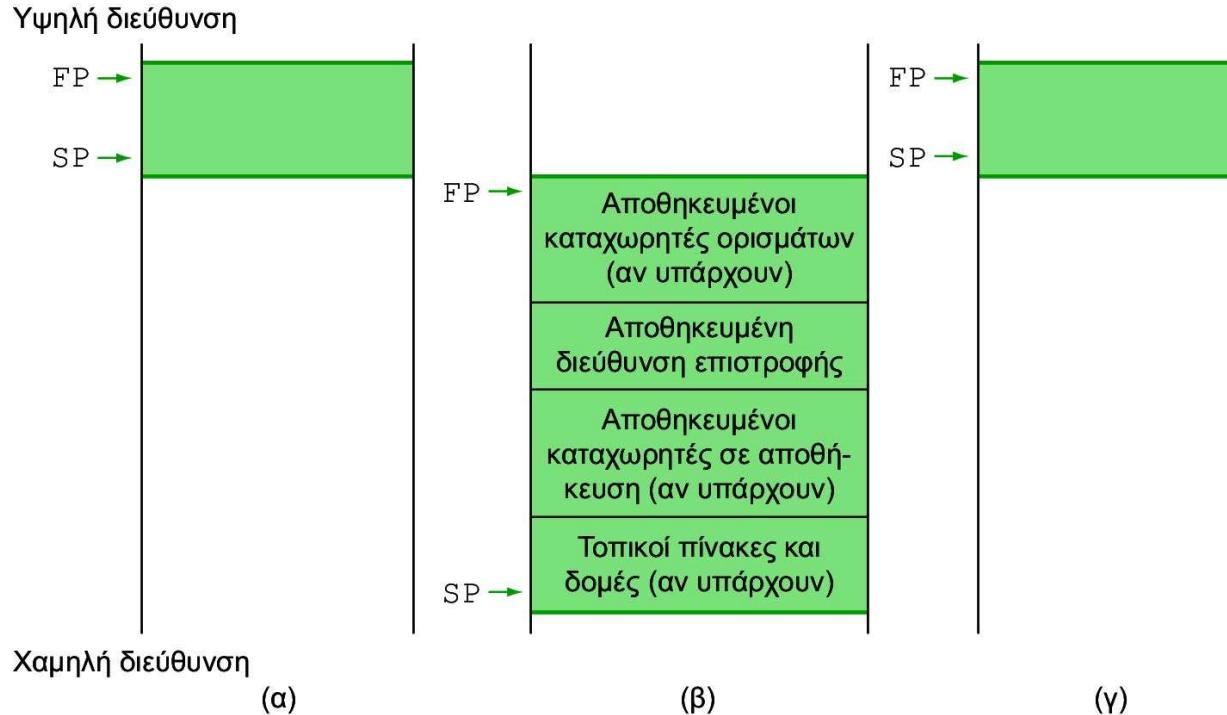
- Κείμενο: κώδικας προγράμματος
- Στατικά δεδομένα: καθολικές μεταβλητές
 - π.χ. στατικές μεταβλητές στη C, πίνακες και συμβολοσειρές από σταθερές
 - Ο x3 (καθολικός δείκτης) αρχικοποιείται σε αυτή τη διεύθυνση ώστε να προσπελάζει τις άλλες διευθύνσεις χρησιμοποιώντας θετικές και αρνητικές σχετικές διευθύνσεις
- Δυναμικά δεδομένα: σωρός
 - π.χ. malloc στη C, new στην Java
- Στοίβα: αυτόματη αποθήκευση

SP →

PC →



Τοπικά δεδομένα στη στοίβα



- Τοπικά δεδομένα που εκχωρούνται από την καλούμενη διαδικασία
 - π.χ. αυτόματες μεταβλητές της C
- Πλαίσιο διαδικασίας (εγγραφή ενεργοποίησης)
 - Χρησιμοποιείται από κάποιους μεταγλωττιστές για τη διαχείριση της αποθήκευσης στη στοίβα

Δεδομένα χαρακτήρων

- Σύνολα χαρακτήρων με κωδικοποίηση byte
 - ASCII: 128 χαρακτήρες
 - 95 χαρακτήρες γραφικών, 33 χαρακτήρες ελέγχου
 - Latin-1: 256 χαρακτήρες
 - ASCII, +96 χαρακτήρες γραφικών
- Unicode: Σύνολο χαρακτήρων 32 bit
 - Χρησιμοποιείται στην Java, στους χαρακτήρες της C++, ...
 - Τα περισσότερα ανθρώπινα αλφάβητα, συν σύμβολα
 - UTF-8, UTF-16: κωδικοποιήσεις μεταβλητού μήκους

Λειτουργίες byte/ημιλέξεων/λέξεων

- Εντολές φόρτωσης/αποθήκευσης byte/ημιλέξεων/λέξεων στον RISC-V
 - Φόρτωση byte/ημιλέξης/λέξης: Επέκταση προσήμου στα 32 bit στον rd
 - 1b rd, offset(rs1)
 - 1h rd, offset(rs1)
 - 1w rd, offset(rs1)
 - Φόρτωση byte/ημιλέξης/λέξης χωρίς πρόσημο: Επέκταση μηδενικού στα 32 bit στον rd
 - 1bu rd, offset(rs1)
 - 1hu rd, offset(rs1)
 - 1wu rd, offset(rs1)
 - Αποθήκευση byte/ημιλέξης/λέξης: Αποθήκευση των δεξιών 8/16/32 bit
 - sb rs2, offset(rs1)
 - sh rs2, offset(rs1)
 - sw rs2, offset(rs1)

Ένα παράδειγμα αντιγραφής μιας συμβολοσειράς

■ Κώδικας C:

- Συμβολοσειρά που τερματίζεται με τον μηδενικό χαρακτήρα (null)

```
void strcpy (char x[], char y[])
{ size_t i;
  i = 0;
  while ((x[i]=y[i]) != '\0')
    i += 1;
}
```

Ένα παράδειγμα αντιγραφής μιας συμβολοσειράς

■ Κώδικας RISC-V:

strcpy:

```
addi sp,sp,-4          // ρύθμιση της στοίβας για  
                       // 1 διπλή λέξη  
sw    x19,0(sp)        // ώθηση του x19  
add   x19,x0,x0        // i = 0  
L1: add   x5,x19,x10  // x5 = διεύθυνση του y[i]  
    lbu   x6,0(x5)        // x6 = y[i]  
    add   x7,x19,x10  // x7 = διεύθυνση του x[i]  
    sb    x6,0(x7)        // x[i] = y[i]  
    beq   x6,x0,L2        // αν y[i] == 0, τότε έξοδος  
    addi  x19,x19,1        // i = i + 1  
    jal   x0,L1          // επόμενη επανάληψη του βρόχου  
L2: lw    x19,0(sp)        // επαναφορά αποθηκευμένου x19  
    addi  sp,sp,4          // εξαγωγή 1 διπλής λέξης από τη  
                           // στοίβα  
    jalr x0,0(x1)        // και επιστροφή
```

Σταθερές των 32 bit

- Οι περισσότερες σταθερές έχουν μικρή τιμή
 - Επαρκούν άμεσοι τελεστέοι των 12 bit
- Για μια σταθερά των 32 bit που ενδέχεται να προκύψει περιστασιακά

lui rd, constant

- Αντιγράφει τη σταθερά των 20 bit στα bit [31:12] του rd
- Θέτει τα bit [11:0] του rd ίσα με 0

lui x19, 976 // 0x003D0

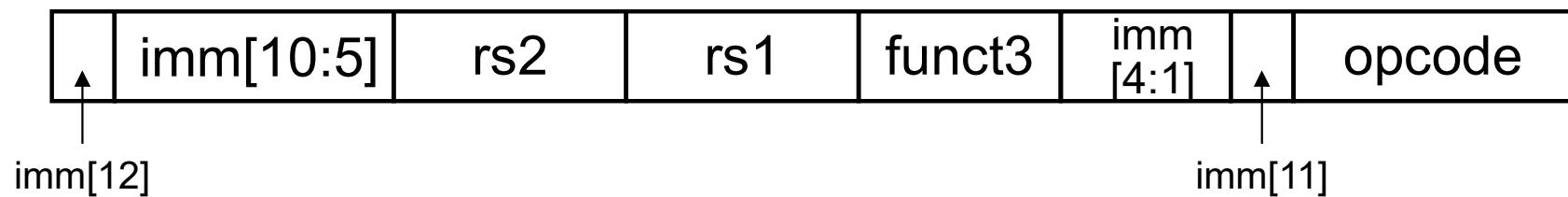
0000 0000 0011 1101 0000	0000 0000 0000
--------------------------	----------------

addi x19, x19, 1280 // 0x500

0000 0000 0011 1101 0000	0101 0000 0000
--------------------------	----------------

Διευθυνσιοδότηση διακλαδώσεων

- Στις εντολές διακλάδωσης προσδιορίζονται
 - ο κωδικός λειτουργίας (opcode), δύο καταχωρητές, η διεύθυνση προορισμού
 - Οι περισσότερες διακλαδώσεις υπό συνθήκη έχουν προορισμό μια κοντινή θέση
 - Είτε προς τα εμπρός είτε προς τα πίσω
 - Μορφή τύπου SB:

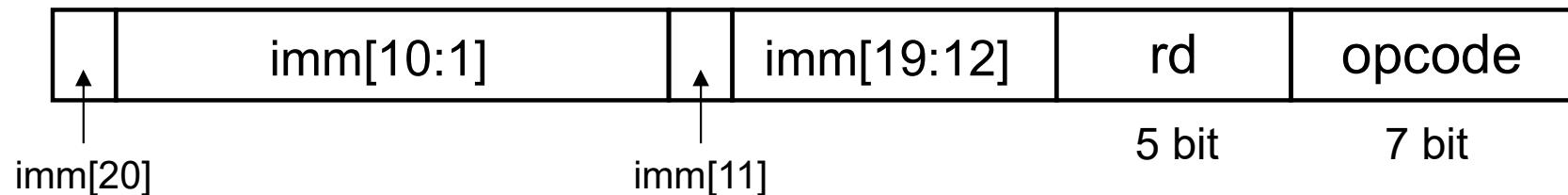


- Διευθυνσιοδότηση σχετική ως προς τον μετρητή προγράμματος (PC-relative addressing)
 - Διεύθυνση προορισμού = PC + άμεσος × 4



Διευθυνσιοδότηση με áλμα

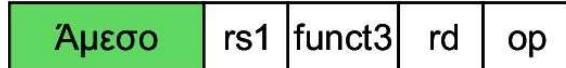
- Ο προορισμός μιας εντολής jump and link (jal) έχει áμεσο τελεστέο των 20 bit για μεγαλύτερο εύρος τιμών
- Μορφή τύπου UJ:



- Κι εδώ διευθυνσιοδότηση σχετική ως προς τον μετρητή προγράμματος (PC-relative addressing)
 - Διεύθυνση προορισμού áλματος = PC + áμεσος × 4
- Για μεγάλα áλματα, π.χ. σε μια διεύθυνση 32 bit
 - lui: Φόρτωση διεύθυνσης [31:12] σε προσωρινό καταχωρητή
 - jalr: πρόσθεση της διεύθυνσης [11:0] και áλμα στην εντολή προορισμού

Σύνοψη των τρόπων διευθυνσιοδότησης στον RISC-V

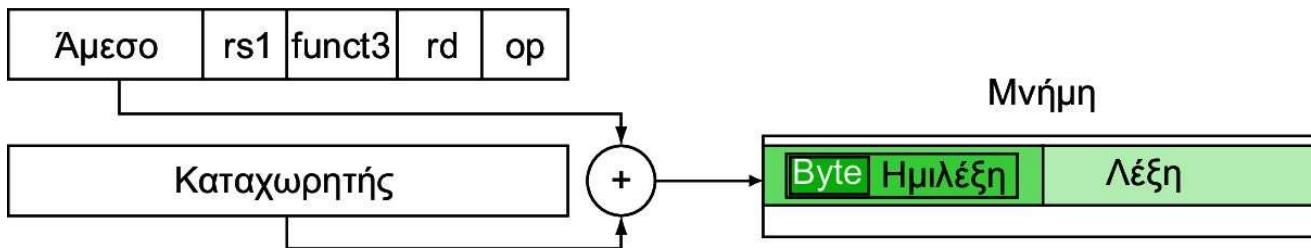
1. Άμεση διευθυνσιοδότηση



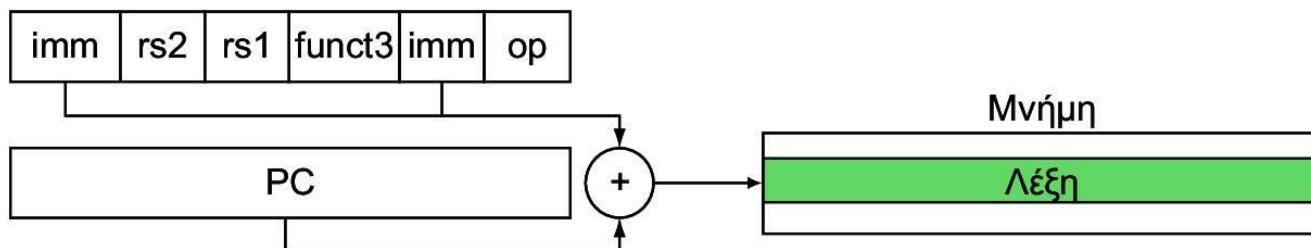
2. Διευθυνσιοδότηση μέσω καταχωριτή



3. Διευθυνσιοδότηση βάσης



4. Σχετική διευθυνσιοδότηση ως προς τον PC



Σύνοψη των τρόπων κωδικοποίησης στον RISC-V

'Όνομα (Μέγεθος πεδίου)	Πεδίο						Σχόλια
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
Τύπου R	funct7	rs2	rs1	funct3	rd	opcode	Μορφή αριθμητικής εντολής
Τύπου I	immediate[11:0]		rs1	funct3	rd	opcode	Φορτώσεις & αριθμητικές με άμεσα
Τύπου S	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Αποθηκεύσεις
Τύπου SB	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Μορφή διακλάδωσης υπό συνθήκη
Τύπου UJ	immediate[20,10:1 11,19:12]				rd	opcode	Μορφή άλματος χωρίς συνθήκη
Τύπου U	immediate[31:12]					opcode	Μορφή άνω άμεσου

Κάρτα αναφοράς RISC-V (από βιβλίο) (1)



① Δεδομένα αναφοράς

Βασικές ακέραιες εντολές του RV32I, σε αλφαριθμητική σειρά

Μνημονικό	Μορφή	Όνομα	Περιγραφή (σε Verilog)	Σημείωση
add	R	ADD	$R[rd] = R[rs1] + R[rs2]$	
addi	I	ADD Immediate	$R[rd] = R[rs1] + imm$	
and	R	AND	$R[rd] = R[rs1] \& R[rs2]$	
andi	I	AND Immediate	$R[rd] = R[rs1] \& imm$	
auipc	U	Add Upper Immediate to PC	$R[rd] = PC + \{imm, 12'b0\}$	
beq	SB	Branch EQual	$if(R[rs1]==R[rs2])$ $PC=PC+\{imm,1b'0\}$	
bge	SB	Branch Greater than or Equal	$if(R[rs1]>=R[rs2])$ $PC=PC+\{imm,1b'0\}$	
bgeu	SB	Branch \geq Unsigned	$if(R[rs1]>=R[rs2])$ $PC=PC+\{imm,1b'0\}$	2)
blt	SB	Branch Less Than	$if(R[rs1]<R[rs2])$ $PC=PC+\{imm,1b'0\}$	
bltu	SB	Branch Less Than Unsigned	$if(R[rs1]<R[rs2])$ $PC=PC+\{imm,1b'0\}$	2)
bne	SB	Branch Not Equal	$if(R[rs1]!=R[rs2])$ $PC=PC+\{imm,1b'0\}$	
csrrc	I	Cont./Stat.RegRead&Clear	$R[rd] = CSR; CSR = CSR \& \sim R[rs1]$	
csrrci	I	Cont./Stat.RegRead&Clear	$R[rd] = CSR; CSR = CSR \& \sim imm$	
		Imm		
csrrs	I	Cont./Stat.RegRead&Set	$R[rd] = CSR; CSR = CSR R[rs1]$	
csrrsi	I	Cont./Stat.RegRead&Set	$R[rd] = CSR; CSR = CSR imm$	
		Imm		
csrrw	I	Cont./Stat.RegRead&Write	$R[rd] = CSR; CSR = R[rs1]$	
csrrwi	I	Cont./Stat.Reg Read&Write	$R[rd] = CSR; CSR = imm$	
		Imm		
ebreak	I	Environment BREAK	Transfer control to debugger	
ecall	I	Environment CALL	Transfer control to operating system	
fence	I	Synch thread	Synchronizes threads	
fence.i	I	Synch Instr & Data	Synchronizes writes to instruction stream	
jal	UJ	Jump & Link	$R[rd] = PC+4; PC = PC + \{imm,1b'0\}$	
jalr	I	Jump & Link Register	$R[rd] = PC+4; PC = R[rs1]+imm$	

②

ΣΥΝΟΛΟ ΕΝΤΟΛΩΝ ΑΡΙΘΜΗΤΙΚΟΥ ΗΥΨΗΝΑ

Επέκταση Πολλαπλασιασμού RV64M

Μνημονικό	Μορφή	Όνομα	Περιγραφή (σε Verilog)	Σημείωση
mul	R	MULtiply	$R[rd] = (R[rs1] * R[rs2])(63:0)$	
mulh	R	MULtiply High	$R[rd] = (R[rs1] * R[rs2])(127:64)$	
mulhsu	R	MULtiply High Unsigned	$R[rd] = (R[rs1] * R[rs2])(127:64)$	2)
mulhu	R	MULtiply upper Half Unsigned	$R[rd] = (R[rs1] * R[rs2])(127:64)$	6)
div	R	DIVide	$R[rd] = (R[rs1] / R[rs2])$	
divu	R	DIVide Unsigned	$R[rd] = (R[rs1] / R[rs2])$	2)
rem	R	REMAinder	$R[rd] = (R[rs1] \% R[rs2])$	
remu	R	REMAinder Unsigned	$R[rd] = (R[rs1] \% R[rs2])$	2)
Επεκτάσεις κινητής υποδιαστολής RV64F και RV64D				
fld, flw	I	Load (Word)	$F[rd] = M[R[rs1]+imm]$	
fsd, fsw	S	Store (Word)	$M[R[rs1]+imm] = F[rd]$	
fadd.s, fadd.d	R	ADD	$F[rd] = F[rs1] + F[rs2]$	7)
fsub.s, fsub.d	R	SUBtract	$F[rd] = F[rs1] - F[rs2]$	7)
fmul.s, fmul.d	R	MULtiply	$F[rd] = F[rs1] * F[rs2]$	7)
fdiv.s, fdiv.d	R	DIVide	$F[rd] = F[rs1] / F[rs2]$	7)
fsqrt.s, fsqrt.d	R	SQuare Root	$F[rd] = sqrt(F[rs1])$	7)
fmadd.s, fmadd.d	R	Multiply-ADD	$F[rd] = F[rs1] * F[rs2] + F[rs3]$	7)
fmsub.s, fmsub.d	R	Multiply-SUBtract	$F[rd] = F[rs1] * F[rs2] - F[rs3]$	7)
fmnsub.s, fmnsub.d	R	Negative Multiply-ADD	$F[rd] = -(F[rs1] * F[rs2] - F[rs3])$	7)
fmnadd.s, fmnadd.d	R	Negative Multiply-SUBtract	$F[rd] = -(F[rs1] * F[rs2] + F[rs3])$	7)
fsgnj.s, fsgnj.d	R	SiGN source	$F[rd] = \{ F[rs2]<63>, F[rs1]<62:0>\}$	7)
fsgnjn.s, fsgnjn.d	R	Negative SiGN source	$F[rd] = \{ (\sim F[rs2]<63>), F[rs1]<62:0>\}$	7)
fsgnjx.s, fsgnjx.d	R	Xor SiGN source	$F[rd] = \{ F[rs2]<63> \wedge F[rs1]<63>, F[rs1]<62:0>\}$	7)
fmin.s, fmin.d	R	MINimum	$F[rd] = (F[rs1] < F[rs2]) ? F[rs1] : F[rs2]$	7)
fmax.s, fmax.d	R	MAXimum	$F[rd] = (F[rs1] > F[rs2]) ? F[rs1] : F[rs2]$	7)
feq.s, feq.d	R	Compare Float Equal	$R[rd] = (F[rs1]==F[rs2]) ? 1 : 0$	7)
flt.s, flt.d	R	Compare Float Less Than	$R[rd] = (F[rs1] < F[rs2]) ? 1 : 0$	7)
fle.s, fle.d	R	Compare Float Less than or =	$R[rd] = (F[rs1] \leq F[rs2]) ? 1 : 0$	7)
fclass.s, fclass.d	R	Classify Type	$R[rd] = class(F[rs1])$	7)



Κάρτα αναφοράς RISC-V (από βιβλίο) (2)

lw	I	Load Word	$R[rd] = \{M[R[rs1]] + imm\}(31:0)$
or	R	OR	$R[rd] = R[rs1] R[rs2]$
ori	I	OR Immediate	$R[rd] = R[rs1] imm$
sb	S	Store Byte	$M[R[rs1]] + imm(7:0) = R[rs2](7:0)$
sh	S	Store Halfword	$M[R[rs1]] + imm(15:0) = R[rs2](15:0)$
sll	R	Shift Left	$R[rd] = R[rs1] << R[rs2]$
slli	I	Shift Left Immediate	$R[rd] = R[rs1] << imm$
slt	R	Set Less Than	$R[rd] = (R[rs1] < R[rs2]) ? 1 : 0$
slti	I	Set Less Than Immediate	$R[rd] = (R[rs1] < imm) ? 1 : 0$
sltiu	I	Set < Immediate Unsigned	$R[rd] = (R[rs1] < imm) ? 1 : 0$
sltu	R	Set Less Than Unsigned	$R[rd] = (R[rs1] < R[rs2]) ? 1 : 0$
sra	R	Shift Right Arithmetic	$R[rd] = R[rs1] >> R[rs2]$
srai	I	Shift Right Arith Imm	$R[rd] = R[rs1] >> imm$
srl	R	Shift Right (Word)	$R[rd] = R[rs1] >> R[rs2]$
srlt	I	Shift Right Immediate	$R[rd] = R[rs1] >> imm$
sub, subw	R	SUBtract (Word)	$R[rd] = R[rs1] - R[rs2]$
sw	S	Store Word	$M[R[rs1]] + imm(31:0) = R[rs2](31:0)$
xor	R	XOR	$R[rd] = R[rs1] ^ R[rs2]$
xori	I	XOR Immediate	$R[rd] = R[rs1] ^ imm$

- Σημειώσεις:
- Η λειτουργία εκτελείται εξ ορισμού με απρόσημους ακεραίους (αντί για συμπλήρωμα ως προς 2)
 - Το λιγότερο σημαντικό bit της διεύθυνσης διακλάδωσης της jalr παίρνει τηνή 0
 - Οι (προσημασμένες) εντολές load επεκτείνουν το bit προσήμου των δεδομένων ώστε να γεμίσει ο καταχωρητής 32 bit
 - Με επανάληψη του bit προσήμου για να συμπληρωθούν τα αριστερότερα bit του αποτελέσματος κατά τη δεξιά ολίσθηση
 - Πολλαπλασιασμός με έναν τελεστέο προσημασμένο και έναν απρόσημο
 - Η έκδοση single εκτελεί λειτουργία απλής ακρίβειας χρησιμοποιώντας τα 32 δεξιότερα bit ενός καταχωρητή F των 64 bit
 - Κατηγοριοποίηση των εγγραφών με μια μάσκα 10 bit που δείχνει ποιες ιδιότητες ισχύουν (π.χ. -άπειρο, -0, +0, +άπειρο, μη κανονικοποιημένος, ...)
 - Ατομική λειτουργία μνήμης: Τίποτε άλλο δεν μπορεί να παρεμβληθεί μεταξύ της ανάγνωσης και της εγγραφής της θέσης μνήμης

To άμεσο πεδίο υφίσταται επέκταση προσήμου στη RISC-V

4)	fcvt.s.wu, fcvt.d.wu	R	Convert from 32b Int Unsigned	$F[rd] = float(R[rs1](31:0))$	2,7)
	fcvt.s.lu, fcvt.d.lu	R	Convert from 64b Int Unsigned	$F[rd] = float(R[rs1](63:0))$	2,7)
	fcvt.w.s, fcvt.w.d	R	Convert to 32b Integer	$R[rd](31:0) = integer(F[rs1])$	7)
	fcvt.l.s, fcvt.l.d	R	Convert to 64b Integer	$R[rd](63:0) = integer(F[rs1])$	7)
	fcvt.wu.s, fcvt.wu.d	R	Convert to 32b Int Unsigned	$R[rd](31:0) = integer(F[rs1])$	2,7)
	fcvt.lu.s, fcvt.lu.d	R	Convert to 64b Int Unsigned	$R[rd](63:0) = integer(F[rs1])$	2,7)
	Επέκταση ατομικών (αδιαίρετων) εντολών RV64A				
	amoadd.w, amoadd.d	R	ADD	$R[rd] = M[R[rs1]],$ $M[R[rs1]] = M[R[rs1]] + R[rs2]$	9)
	amoand.w, amoand.d	R	AND	$R[rd] = M[R[rs1]],$ $M[R[rs1]] = M[R[rs1]] & R[rs2]$	9)
	amomax.w, amomax.d	R	MAXimum	$R[rd] = M[R[rs1]],$ if $(R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]$	9)
	amomaxu.w, amomaxu.d	R	MAXimum Unsigned	$R[rd] = M[R[rs1]],$ if $(R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]$	2,9)
	amomin.w, amomin.d	R	MINimum	$R[rd] = M[R[rs1]],$ if $(R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]$	9)
	amominu.w, amominu.d	R	MINimum Unsigned	$R[rd] = M[R[rs1]],$ if $(R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]$	2,9)
	amoor.w, amoor.d	R	OR	$R[rd] = M[R[rs1]],$ $M[R[rs1]] = M[R[rs1]] R[rs2]$	9)
	amoswap.w, amoswap.d	R	SWAP	$R[rd] = M[R[rs1]], M[R[rs1]] = R[rs2]$	9)
	amoxor.w, amoxor.d	R	XOR	$R[rd] = M[R[rs1]],$ $M[R[rs1]] = M[R[rs1]] ^ R[rs2]$	9)
	lr.w, lr.d	R	Load Reserved	$R[rd] = M[R[rs1]],$ reservation on $M[R[rs1]]$	
	sc.w, sc.d	R	Store Conditional	if reserved, $M[R[rs1]] = R[rs2]$, $R[rd] = 0$; else $R[rd] = 1$	

ΜΟΡΦΕΣ ΕΝΤΟΛΩΝ ΠΥΡΗΝΑ

	31	27	26	25	24	20	19	15	14	12	11	7	6	0
R	funct7		rs2		rs1		funct3		rd		Opcode			
I		imm[11:0]			rs1		funct3		rd		Opcode			
S	imm[11:5]		rs2		rs1		funct3	imm[4:0]		opcode				
SB	imm[12 10:5]		rs2		rs1		funct3	imm[4:1 11]		opcode				
U		imm[31:12]							rd		opcode			
UJ		imm[20 10:1 11 19:12]							rd		opcode			



Κάρτα αναφοράς RISC-V (από βιβλίο) (3)

ΨΕΥΔΟΕΝΤΟΛΕΣ

Mnημονικό	Όνομα	Περιγραφή
beqz	Branch = zero	if(R[rs1]==0) PC=PC+{imm,1b'0}
bnez	Branch ≠ zero	if(R[rs1]!=0) PC=PC+{imm,1b'0}
fabs.s,fabs.d	Absolute Value	F[rd] = (F[rs1]< 0) ? -F[rs1] : F[rs1]
fmv.s, fmv.d	FP Move	F[rd] = F[rs1]
fneg.s, fneg.d	FP negate	F[rd] = -F[rs1]
j	Jump	PC = {imm,1b'0}
jr	Jump register	PC = R[rs1]
la	Load address	R[rd] = address
li	Load imm	R[rd] = imm
mv	Move	R[rd] = R[rs1]
neg	Negate	R[rd] = -R[rs1]
nop	No operation	R[0] = R[0]
not	Not	R[rd] = ~R[rs1]
ret	Return	PC = R[1]
seqz	Set = zero	R[rd] = (R[rs1]== 0) ? 1 : 0
sneqz	Set ≠ zero	R[rd] = (R[rs1]!= 0) ? 1 : 0

ΚΩΔΙΚΟΙ ΛΕΙΤΟΥΡΓΙΑΣ (OPCODES) ΣΕ ΑΡΙΘΜΗΤΙΚΗ ΣΕΙΡΑ

Mnημονικό	Μορφή	OPCODE	Λειτουργία3	Λειτουργία7 ή IMM	Δεκαεξαδικά
lb	I	0000011	000		03/0
lh	I	0000011	001		03/1
lw	I	0000011	010		03/2
lbu	I	0000011	100		03/4
lhu	I	0000011	101		03/5
fence	I	0001111	000		0F/0
fence.i	I	0001111	001		0F/1
addi	I	0010011	000		13/0
slli	I	0010011	001	0000000	13/1/00
slti	I	0010011	010		13/2
sltiu	I	0010011	011		13/3
xori	I	0010011	100		13/4
srli	I	0010011	101	0000000	13/5/00
srai	I	0010011	101	0100000	13/5/20
ori	I	0010011	110		13/6

③

beq	zero
bne	ra
fabs.s,fabs.d	sp
fmv.s, fmv.d	gp
fneg.s, fneg.d	tp
j	t0-t2
jr	s0/fp
la	s1
li	a0-a1
mv	a2-a7
neg	s2-s11
nop	t3-t6
not	ft0-ft7
ret	ft8-fs1
seqz	fa0-fa1
sneqz	fa2-fa7
	fs2-fs11
	ft8-ft11

ΟΝΟΜΑ, ΧΡΗΣΗ, ΣΥΜΒΑΣΗ ΚΛΗΣΗΣ ΚΑΤΑΧΩΡΗΤΩΝ

④

ΚΑΤΑΧΩΡΗΤΗΣ	ΟΝΟΜΑ	ΧΡΗΣΗ	ΑΠΟΘΗΚΕΥΕΤΑΙ ΑΠΟ
x0	zero	Η σταθερή τιμή 0	Δ/Ε
x1	ra	Διεύθυνση επιστροφής	Καλούσα συνάρ./λειτ.
x2	sp	Δείκτης στοίβας	Καλούμενη συνάρ./λειτ.
x3	gp	Καθολικός δείκτης	--
x4	tp	Δείκτης νήματος	--
x5-x7	t0-t2	Προσωρινοί καταχωρητές	Καλούσα συνάρ./λειτ.
x8	s0/fp	Αποθηκευμένος καταχωρητής/ δείκτης πλαισίου	Καλούμενη συνάρ./λειτ.
x9	s1	Αποθηκευμένος καταχωρητής	Καλούμενη συνάρ./λειτ.
x10-x11	a0-a1	Ορίσματα συνάρτησης/ επιστρεφόμενες τιμές	Καλούσα συνάρ./λειτ.
x12-x17	a2-a7	Ορίσματα συνάρτησης	Καλούσα συνάρ./λειτ.
x18-x27	s2-s11	Αποθηκευμένοι καταχωρητές	Καλούμενη συνάρ./λειτ.
x28-x31	t3-t6	Προσωρινοί καταχωρητές	Καλούσα συνάρ./λειτ.
f0-f7	ft0-ft7	Προσωρινοί καταχωρητές κιν. υποδ.	Καλούσα συνάρ./λειτ.
f8-f9	fs0-fs1	Αποθηκευμένοι καταχωρητές κιν. υποδ.	Καλούμενη συνάρ./λειτ.
f10-f11	fa0-fa1	Ορίσματα συνάρτησης/ επιστρεφόμενες τιμές κιν. υποδ.	Καλούσα συνάρ./λειτ.
f12-f17	fa2-fa7	Ορίσματα συνάρτησης κιν. υποδ.	Καλούσα συνάρ./λειτ.
f18-f27	fs2-fs11	Αποθηκευμένοι καταχωρητές κιν. υποδ.	Καλούμενη συνάρ./λειτ.
f28-f31	ft8-ft11	R[rd] = R[rs1] + R[rs2]	Καλούσα συνάρ./λειτ.

ΠΡΟΤΥΠΟ ΚΙΝΗΤΗΣ ΥΠΟΔΙΑΣΤΟΛΗΣ IEEE 754

$$(-1)^{\text{Π}} \times (1 + \text{Κλάσμα}) \times 2^{(\text{Εκθέτης} - \text{Πόλωση})}$$

όπου πόλωση μισής ακρίβειας = 15, πόλωση απλής ακρίβειας = 127, πόλωση διπλής ακρίβειας = 1023, πόλωση τετραπλής ακρίβειας = 16383
Μορφές μισής, απλής, διπλής, και τετραπλής ακρίβειας IEEE

Π	Εκθέτης	Κλάσμα
15	14	10 9 0
Π	Εκθέτης	Κλάσμα
31	30	23 22 0

Κεφάλαιο 2 — Εντολές: η γλώσσα του υπολογιστή — 65



Διαφάνειες διδασκαλίας πρωτότυπου βιβλίου μεταφρασμένες στα ελληνικά (μετάφραση, επιμέλεια, προσθήκες: Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών)

Κάρτα αναφοράς RISC-V (από βιβλίο) (4)

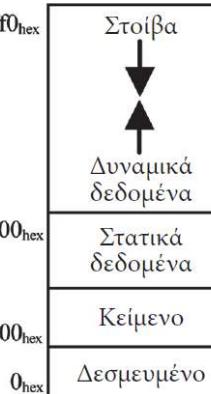
sw	S	0100011	000		23/0
sh	S	0100011	001		23/1
sw	S	0100011	010		23/2
add	R	0110011	000	00000000	33/0/00
sub	R	0110011	000	01000000	33/0/20
sll	R	0110011	001	00000000	33/1/00
slt	R	0110011	010	00000000	33/2/00
sltu	R	0110011	011	00000000	33/3/00
xor	R	0110011	100	00000000	33/4/00
srl	R	0110011	101	00000000	33/5/00
sra	R	0110011	101	01000000	33/5/20
or	R	0110011	110	00000000	33/6/00
and	R	0110011	111	00000000	33/7/00
lui	U	0110111			37

ΚΑΤΑΝΟΜΗ ΜΝΗΜΗΣ

SP → 0000 003f ffff ffff0_{hex}

0000 0000 1000 0000_{hex}

PC → 0000 0000 0040 0000_{hex}



ΠΛΑΙΣΙΟ ΣΤΟΙΒΑΣ

Υψηλότερες διευθύνσεις μνήμης

Αύξηση της στοιβας

Χαμηλότερες διευθύνσεις μνήμης

beq	SB	1100011	000		63/0
bne	SB	1100011	001		63/1
blt	SB	1100011	100		63/4
bge	SB	1100011	101		63/5
bltu	SB	1100011	110		63/6
bgeu	SB	1100011	111		63/7
jalr	I	1100111	000		67/0
jal	UJ	1101111			6F
ecall	I	1110011	000	00000000000000	73/0/000
ebreak	I	1110011	000	00000000000001	73/0/001
CSRRW	I	1110011	001		73/1
CSRRS	I	1110011	010		73/2
CSRRC	I	1110011	011		73/3
CSRRWI	I	1110011	101		73/5
CSRRSI	I	1110011	110		73/6
CSRRCI	I	1110011	111		73/7

ΠΡΟΘΕΜΑΤΑ ΜΕΓΕΘΟΥΣ ΚΑΙ ΣΥΜΒΟΛΑ

ΜΕΓΕΘΟΣ	ΠΡΟΘΕΜΑ	ΣΥΜΒΟΛΟ	ΜΕΓΕΘΟΣ	ΠΡΟΘΕΜΑ	ΣΥΜΒΟΛΟ
1000 ¹	Kilo-	K	2 ¹⁰	Kibi-	Ki
1000 ²	Mega-	M	2 ²⁰	Mebi-	Mi
1000 ³	Giga-	G	2 ³⁰	Gibi-	Gi
1000 ⁴	Tera-	T	2 ⁴⁰	Tebi-	Ti
1000 ⁵	Peta-	P	2 ⁵⁰	Pebi-	Pi
1000 ⁶	Exa-	E	2 ⁶⁰	Exbi-	Ei
1000 ⁷	Zetta-	Z	2 ⁷⁰	Zebi-	Zi
1000 ⁸	Yotta-	Y	2 ⁸⁰	Yobi-	Yi
1000 ⁹	Ronna-	R	2 ⁹⁰	Robi-	Ri
1000 ¹⁰	Quecca-	Q	2 ¹⁰⁰	Quebi-	Qi
1000 ⁻¹	milli-	m	1000 ⁻⁵	femto-	f
1000 ⁻²	micro-	μ	1000 ⁻⁶	atto-	a
1000 ⁻³	nano-	n	1000 ⁻⁷	zepto-	z
1000 ⁻⁴	pico-	p	1000 ⁻⁸	yocto-	y
			1000 ⁻⁹	ronto-	r
			1000 ⁻¹⁰	quecto-	q



Συγχρονισμός (synchronization)

- Δύο επεξεργαστές P1 και P2 χρησιμοποιούν από κοινού μια περιοχή της μνήμης
 - Ο P1 γράφει, και έπειτα ο P2 διαβάζει
 - Αν οι P1 και P2 δεν συγχρονιστούν, τότε γίνεται συναγωνισμός δεδομένων (data race)
 - Το αποτέλεσμα εξαρτάται από τη σειρά των προσπελάσεων
- Απαιτείται υποστήριξη από το υλικό
 - Αδιαίρετη λειτουργία ανάγνωσης-εγγραφής της μνήμης
 - Δεν επιτρέπεται άλλη προσπέλαση της συγκεκριμένης θέσης μνήμης ανάμεσα στην ανάγνωση και εγγραφή
- Μπορεί να πρόκειται για μία και μόνη εντολή
 - π.χ. εναλλαγή καταχωρητών  μνήμης
 - Ή ένα αδιαίρετο ζεύγος εντολών

Συγχρονισμός στον RISC-V

- εντολή load reserved (φόρτωση δεσμευμένης· εδώ: απλής λέξης): `l r.w rd, (rs1)`
 - Φόρτωση από τη διεύθυνση που υπάρχει στον rs1 προς τον rd
 - Κράτηση της διεύθυνσης στη μνήμη
- εντολή store conditional (αποθήκευση υπό συνθήκη· εδώ: απλής λέξης): `sc.w rd, rs2, (rs1)`
 - Αποθήκευση από τον rs2 στη διεύθυνση που υπάρχει στον rs1
 - Εκτελείται με επιτυχία αν η θέση μνήμης δεν έχει αλλάξει μετά την εντολή `l r.w`
 - Επιστρέφει την τιμή 0 στον rd
 - Δεν εκτελείται αν η θέση μνήμης έχει τροποποιηθεί
 - Επιστρέφει μη μηδενική τιμή στον rd

Συγχρονισμός στον RISC-V

- Παράδειγμα 1: ατομική αντιμετάθεση (για έλεγχο/ορισμό μεταβλητής κλειδώματος)

```
again:    lr.w x10,(x20)
          sc.w x11,x23,(x20) // x11 = κατάστασης
          bne x11,x0,again  // διακλάδωση αν αποτύχει
                      // η αποθήκευση
          addi x23,x10,0     // x23 = τιμή που φορτώθηκε
```

- Παράδειγμα 2: κλείδωμα

```
again:    addi x12,x0,1      // αντιγραφή κλειδωμένης τιμής
          lr.w x10,(x20)    // ανάγνωση κλειδώματος
          bne x10,x0,again  // έλεγχος αν είναι ακόμη 0
          sc.w x11,x12,(x20) // απόπειρα για αποθήκευση
          bne x11,x0,again  // διακλάδωση αν αποτύχει
```

- Ξεκλείδωμα:

```
sw x0,0(x20) // αποδέσμευση κλειδώματος
```

Αποκωδικοποίηση εντολών RISC-V (1)

Στη μνήμη ενός επεξεργαστή RISC-V είναι αποθηκευμένη η λέξη (στο δεκαεξαδικό):

0x0044a283

Ποια εντολή συμβολικής γλώσσας RISC-V είναι;

Την μετατρέπουμε στο δυαδικό

0000 0000 0100 0100 1010 0010 1000 0011

Εντοπίζουμε το opcode (bit 6:0) και το funct3 (bit 14:12)

0000 0000 0100 0100 1010 0010 1000 0011

Αποκωδικοποίηση εντολών RISC-V (2)

Εντοπίζουμε το opcode (bit 6:0) και το funct3 (bit 14:12)

0000 0000 0100 0100 1010 0010 1000 0011

Τα αναζητούμε στην κάρτα αναφοράς RISC-V

ΚΩΔΙΚΟΙ ΛΕΙΤΟΥΡΓΙΑΣ (OPCODES) ΣΕ ΑΡΙΘΜΗΤΙΚΗ ΣΕΙΡΑ

Μνημονικό	Μορφή	OPCODE	Λειτουργία3	Λειτουργία7 ή IMM	Δεκαεξαδικά
lb	I	0000011	000		03/0
lh	I	0000011	001		03/1
lw	I	0000011	010		03/2

Πρόκειται για εντολή lw (άρα μορφής I) – τώρα γνωρίζουμε και τα άλλα τρία πεδία.

Αποκωδικοποίηση εντολών RISC-V (3)

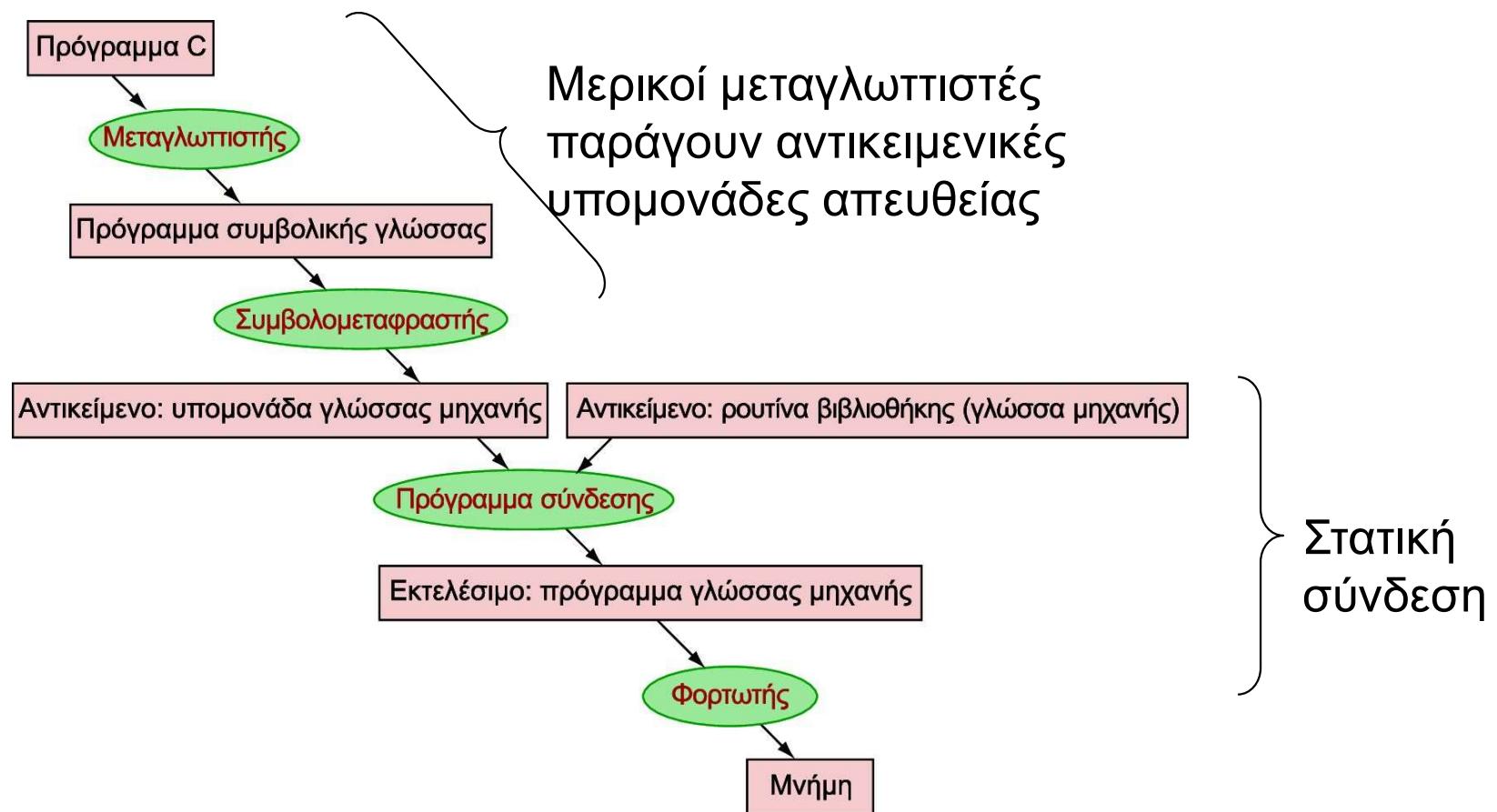
Πρόκειται για εντολή lw (άρα μορφής I) – τώρα γνωρίζουμε και τα άλλα τρία πεδία.

<u>0000</u>	<u>0000</u>	<u>0100</u>	<u>0100</u>	<u>1010</u>	<u>0010</u>	<u>1000</u>	<u>0011</u>
imm[11:0]			rs1	funct3	rd		opcode
=4			=x9		=x5		

Άρα είναι η εντολή:

lw x5, 4(x9)

Μετάφραση και εκκίνηση



Παραγωγή αντικειμενικής υπομονάδας

- Ο συμβολομεταφραστής (ή ο μεταγλωττιστής) μεταφράζει το πρόγραμμα σε εντολές γλώσσας μηχανής
- Παρέχει τα στοιχεία που χρειάζονται για τη δημιουργία του πλήρους προγράμματος από τα επιμέρους τμήματα
 - Επικεφαλίδα: περιγραφή των περιεχομένων της αντικειμενικής υπομονάδας
 - Τμήμα κειμένου: οι μεταφρασμένες εντολές
 - Τμήμα στατικών δεδομένων: δεδομένα που κατανέμονται για όλη τη διάρκεια ζωής του προγράμματος
 - Πληροφορίες επανατοποθέτησης: για περιεχόμενα που εξαρτώνται από την απόλυτη θέση του προγράμματος που φορτώνεται
 - Πίνακας συμβόλων: ορισμοί καθολικής ισχύος και εξωτερικές αναφορές
 - Πληροφορίες αποσφαλμάτωσης: για συσχέτιση με τον πηγαίο κώδικα

Σύνδεση αντικειμενικών υπομονάδων

- Παράγεται ένα εκτελέσιμο αρχείο
 1. Συγχωνεύονται τα επιμέρους τμήματα
 2. Προσδιορίζονται οι ετικέτες (οι διευθύνσεις τους)
 3. Επιδιορθώνονται οι αναφορές: τόσο εξωτερικές και εσωτερικές (όσες εξαρτώνται από θέσεις στο πρόγραμμα)
- Οι εξαρτήσεις θέσεων μπορούν να διορθωθούν από έναν φορτωτή επανατοποθέτησης
 - Όμως με την εικονική μνήμη αυτό είναι περιττό
 - Το πρόγραμμα μπορεί να φορτωθεί σε μια απόλυτη θέση στον χώρο της εικονικής μνήμης

Φόρτωση του προγράμματος

- Ο φορτωτής διαβάζει το αρχείο από τον δίσκο και το φορτώνει στη μνήμη
 - 1. Διαβάζει την επικεφαλίδα για να προσδιορίσει τα μεγέθη των τμημάτων
 - 2. Δημιουργεί έναν χώρο εικονικών διευθύνσεων
 - 3. Αντιγράφει τις εντολές και τα αρχικοποιημένα δεδομένα στη μνήμη
 - Ή ορίζει καταχωρίσεις πίνακα σελίδων για τον χειρισμό των σφαλμάτων
 - 4. Αντιγράφει τις παραμέτρους στη στοίβα
 - 5. Αρχικοποιεί τους καταχωρητές (sp, fp, gp, κ.ά.)
 - 6. Μεταπηδά στη ρουτίνα εκκίνησης
 - Αντιγράφει τα ορίσματα στους καταχωρητές $\times 10$, ... και καλεί τη ρουτίνα main
 - Όταν η κύρια ρουτίνα επιστρέφει, το πρόγραμμα τερματίζεται με μια κλήση για έξοδο (exit)

Δυναμική σύνδεση

- Μια διαδικασία βιβλιοθήκης συνδέεται/φορτώνεται μόνο όταν γίνεται κλήση της
 - Ο κώδικας των διαδικασιών πρέπει να μπορεί να μεταφερθεί σε άλλη θέση
 - Αποφεύγεται η διόγκωση του εκτελέσιμου αρχείου, η οποία προκαλείται από τη στατική σύνδεση όλων των βιβλιοθηκών που αναφέρονται (έστω και παροδικά)
 - Ενημερώνονται αυτόματα στις νέες εκδόσεις τους οι βιβλιοθήκες

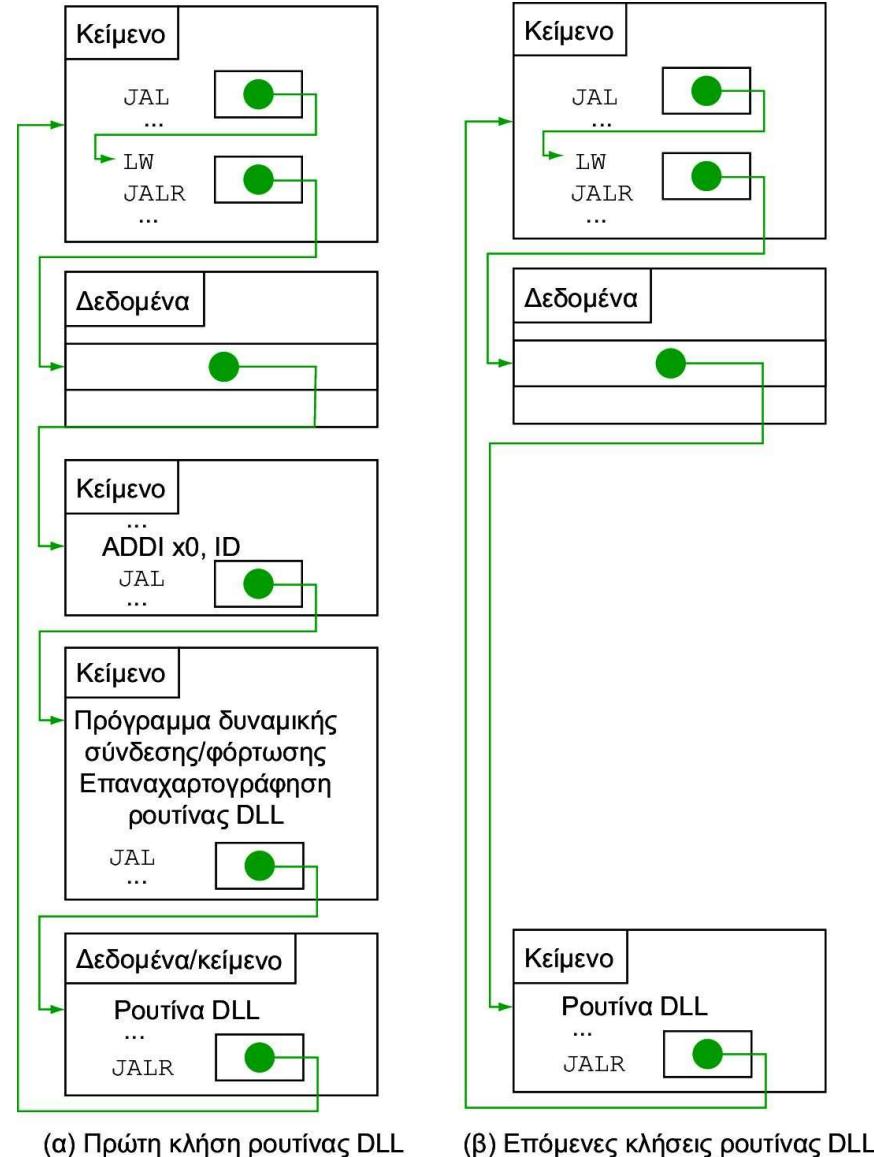
«Ράθυμη» σύνδεση

Πίνακας εμμεσότητας

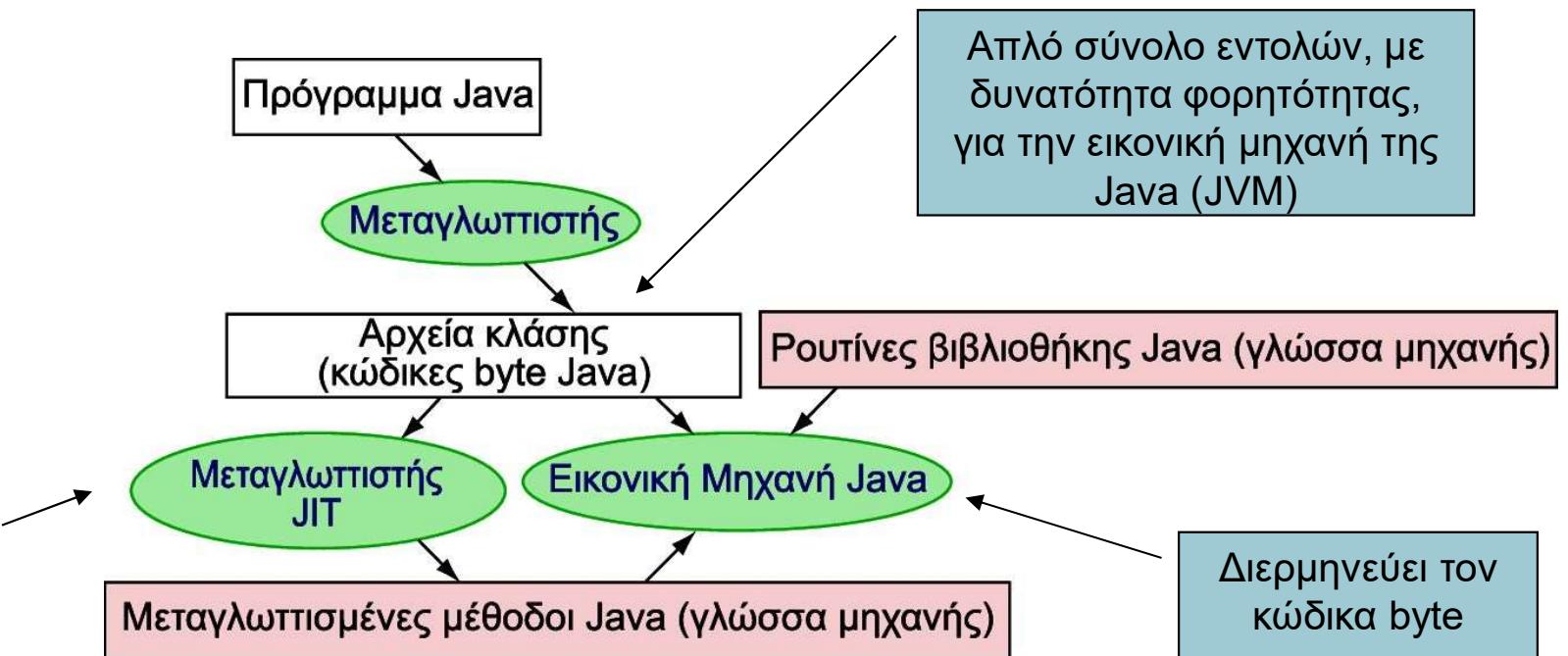
Στέλεχος: Φορτώνει
το αναγνωριστικό της υπορουτίνας,
μετάβαση-άλμα στο
πρόγραμμα σύνδεσης-φόρτωσης

Κώδικας
σύνδεσης-φόρτωσης

Κώδικας που χαρτογραφείται
δυναμικά



Εκκίνηση εφαρμογών Java



Ένα παράδειγμα ταξινόμησης στη C

- Δείχνει τη χρήση των εντολών συμβολικής γλώσσας για μια συνάρτηση ταξινόμησης φυσαλίδας (bubble sort) στη C
- Διαδικασία swap (διαδικασία-φύλλο)

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

- Η v στον x10, η k στον x11, η προσωρινή τιμή στον x5

Η διαδικασία swap

swap:

```
slli x6,x11,2      // reg x6 = k * 4
add  x6,x10,x6     // reg x6 = v + (k * 4)
lw   x5,0(x6)       // reg x5 (temp) = v[k]
lw   x7,4(x6)       // reg x7 = v[k + 1]
sw   x7,0(x6)       // v[k] = reg x7
sw   x5,4(x6)       // v[k+1] = reg x5 (temp)
jalr x0,0(x1)       // επιστροφή στην καλούσα ρουτίνα
```

Η διαδικασία sort στη C

- Διαδικασία που καλεί τη swap

```
void sort (int v[], size_t n)
{
    size_t 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);
        }
    }
}
```

- Η v στον x10, η n στον x11, η i στον x19, η j στον x20

Ο εξωτερικός βρόχος

- Το περίγραμμα του κώδικα του εξωτερικού βρόχου:

- `for (i = 0; i <n; i += 1) {`

```
    li    x19,0          // i = 0
```

for1tst:

```
    bge  x19,x11,exit1   // μετάβαση στην exit1
                           // αν x19 ≥ x11 (i≥n)
```

(σώμα εξωτερικού βρόχου for)

```
    addi x19,x19,1        // i += 1
```

```
    j     for1tst         // διακλάδωση στον έλεγχο
                           // του εξωτερικού βρόχου
```

exit1:

Ο εσωτερικός βρόχος

- Το περίγραμμα του κώδικα του εσωτερικού βρόχου:

```
for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
    addi x20, x19, -1      // j = i - 1
    for2tst:
        blt x20, x0, exit2 // μετάβαση στην exit2 αν x20 < 0 (j < 0)
        slli x5, x20, 2     // reg x5 = j * 4
        add x5, x10, x5      // reg x5 = v + (j * 4)
        lw x6, 0(x5)         // reg x6 = v[j]
        lw x7, 4(x5)         // reg x7 = v[j + 1]
        ble x6, x7, exit2   // μετάβαση στην exit2 αν x6 ≤ x7
        mv x21, x10           // αντιγραφή παραμέτρου x10 στον x21
        mv x22, x11           // αντιγραφή παραμέτρου x11 στον x22
        mv x10, x21           // η πρώτη παράμετρος της swap είναι το v
        mv x11, x20           // η δεύτερη παράμετρος της swap είναι το j
        jal x1, swap          // κλήση της swap
        addi x20, x20, -1      // j -= 1
        j for2tst             // διακλάδωση στον έλεγχο του εσωτερικού βρόχου
exit2:
```

Διατήρηση των καταχωρητών

■ Διατήρηση των αποθηκευμένων καταχωρητών

```
addi sp,sp,-20 // δημιουργία χώρου στη στοίβα για 5 καταχωρητές  
sw x1,16(sp) // αποθήκευση του x1 στη στοίβα  
sw x22,12(sp) // αποθήκευση του x22 στη στοίβα  
sw x21,8(sp) // αποθήκευση του x21 στη στοίβα  
sw x20,4(sp) // αποθήκευση του x20 στη στοίβα  
sw x19,0(sp) // αποθήκευση του x19 στη στοίβα
```

■ Επαναφορά των αποθηκευμένων καταχωρητών

exit1:

```
sw x19,0(sp) // επαναφορά του x19 από τη στοίβα  
sw x20,4(sp) // επαναφορά του x20 από τη στοίβα  
sw x21,8(sp) // επαναφορά του x21 από τη στοίβα  
sw x22,12(sp) // επαναφορά του x22 από τη στοίβα  
sw x1,16(sp) // επαναφορά του x1 από τη στοίβα  
addi sp,sp, 20 // επαναφορά του δείκτη στοίβας  
jalr x0,0(x1)
```

Επίδραση της βελτιστοποίησης του μεταγλωττιστή

Μεταγλώττιση με το gcc για επεξεργαστή Pentium 4 με Linux

Βελτιστοποίηση gcc	Σχετική απόδοση	Κύκλοι ρολογιού (εκατομμύρια)	Αριθμός εντολών (εκατομμύρια)	CPI
καμία	1.00	158 615	114 938	1.38
O1 (μέση)	2.37	66 990	37 470	1.79
O2 (πλήρης)	2.38	66 521	39 993	1.66
O3 (ενσωμάτωση διαδικασίας)	2.41	65 747	44 993	1.46

Η επίδραση της γλώσσας προγραμματισμού και του αλγορίθμου

Γλώσσα	Μέθοδος εκτέλεσης	Βελτιστο-ποίηση	Σχετική απόδοση ταξινόμησης φυσαλίδας	Σχετική απόδοση γρήγορης ταξινόμησης	Επιτάχυνση γρήγορης ταξινόμησης σε σχέση με την ταξινόμηση φυσαλίδας
C	μεταγλωττιστής	καμία	1.00	1.00	2468
	μεταγλωττιστής	O1	2.37	1.50	1562
	μεταγλωττιστής	O2	2.38	1.50	1555
	μεταγλωττιστής	O3	2.41	1.91	1955
Java	διερμηνευτής	—	0.12	0.05	1050
	μεταγλωττιστής JIT	—	2.13	0.29	338

Τι διαπιστώνουμε

- Το πλήθος εντολών και η τιμή CPI δεν είναι από μόνα τους επαρκείς δείκτες απόδοσης
- Οι βελτιστοποιήσεις του μεταγλωττιστή επηρεάζονται από τον αλγόριθμο
- Ο κώδικας Java που μεταγλωτίζεται επιλεκτικά (JIT) είναι σημαντικά ταχύτερος σε σχέση με τον κώδικα που διερμηνεύεται από την JVM
 - Σε κάποιες περιπτώσεις, είναι συγκρίσιμος με βελτιστοποιημένο κώδικα C
- Αν ένας αλγόριθμος είναι «χαζός», δεν γίνεται να διασωθεί!

Πίνακες ή δείκτες;

- Για τη δεικτοδότηση των πινάκων
 - Πολλαπλασιάζουμε τον δείκτη με το μέγεθος του στοιχείου
 - Προσθέσουμε στη διεύθυνση βάσης του πίνακα
- Οι δείκτες αντιστοιχούν απευθείας σε διευθύνσεις της μνήμης
 - Αποφεύγεται η πολυπλοκότητα της δεικτοδότησης

Παράδειγμα: Μηδενισμός ενός πίνακα

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

```
li x5,0           // i = 0  
loop1:  
    slli x6,x5,2   // x6 = i * 4  
    add x7,x10,x6  // x7 = διεύθυνση  
                  // του array[i]  
    sw x0,0(x7)    // array[i] = 0  
    addi x5,x5,1    // i = i + 1  
    blt x5,x11,loop1 // αν (i<size)  
                  // μετάβαση στον  
loop1
```

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

```
mv x5,x10          // p = διεύθυνση  
                  // του array[0]  
    slli x6,x11,2    // x6 = size * 4  
    add x7,x10,x6    // x7 = διεύθυνση  
                  // του array[size]  
loop2:  
    sw x0,0(x5)      // Memory[p] = 0  
    addi x5,x5,4      // p = p + 4  
    bltu x5,x7,loop2 // αν (p<&array[size])  
                  // μετάβαση στον loop2
```

Σύγκριση εκδοχών με πίνακα και με δείκτες

- Βελτιστοποίηση «μείωσης της δύναμης»:
εκτελείται ολίσθηση αντί για πολλαπλασιασμός
- Στην εκδοχή με πίνακα, η ολίσθηση πρέπει να είναι μέσα στον βρόχο
 - Μέρος του υπολογισμού με το αυξημένο i
 - πρβλ. αύξηση της τιμής του δείκτη
- Ο μεταγλωττιστής μπορεί να πετύχει ό,τι μπορεί να γίνει και με μη αυτόματη χρήση δεικτών
 - Εξάλειψη επαγωγικής μεταβλητής
 - Καλύτερα για να είναι το πρόγραμμα πιο σαφές και πιο αξιόπιστο στην εκτέλεση

Εντολές του MIPS

- MIPS: ο προκάτοχος του RISC-V στο εμπόριο
- Παρόμοιο σύνολο βασικών εντολών
 - Εντολές 32 bit
 - 32 καταχωρητές γενικού σκοπού, ο καταχωρητής 0 είναι πάντα ίσος με 0
 - 32 καταχωρητές κινητής υποδιαστολής
 - Προσπέλαση της μνήμης μόνο μέσω εντολών φόρτωσης/αποθήκευσης
 - Συνεπής χρήση τρόπων διευθυνσιοδότησης για όλα τα μεγέθη δεδομένων
- Διαφορετικές διακλαδώσεις για συνθήκες
 - Για <, <=, >, >=
 - RISC-V: blt, bge, bltu, bgeu
 - MIPS: slt, sltu (set less than, αποτέλεσμα 0 ή 1)
 - Και χρήση των beq, bne για να ολοκληρωθεί η διακλάδωση

Κωδικοποίηση εντολών

	31 28 27	20 19	16 15	12 11	4 3	0
Καταχωρητή-καταχωρητή	ARM	Opx ⁴	Op ⁸	Rs1 ⁴	Rd ⁴	Opx ⁸ Rs2 ⁴
MIPS		31 26 25	21 20	16 15	11 10	6 5 0
		Op ⁶	Rs1 ⁵	Rs2 ⁵	Rd ⁵	Const ⁵ Opx ⁶
Φόρτωση	RISC-V	31 25 24	20 19	15 14	12 11	7 6 0
		funct7 ⁷	Rs2 ⁵	Rs1 ⁵	funct3 ³	Rd ⁵ opsode ⁷
Αποθήκευση	ARM	31 28 27	20 19	16 15	12 11	0
		Opx ⁴	Op ⁸	Rs1 ⁴	Rd ⁴	Const ¹²
MIPS		31 26 25	21 20	16 15		0
		Op ⁶	Rs1 ⁵	Rd ⁵	Const ¹⁶	
RISC-V		31	20 19	15 14	12 11	7 6 0
		immediate ¹²		Rs1 ⁵	funct3 ³	Rd ⁵ opsode ⁷
Αποθήκευση	ARM	31 28 27	20 19	16 15	12 11	0
		Opx ⁴	Op ⁸	Rs1 ⁴	Rd ⁴	Const ¹²
MIPS		31 26 25	21 20	16 15		0
		Op ⁶	Rs1 ⁵	Rd ⁵	Const ¹⁶	
RISC-V		31 25 24	20 19	15 14	12 11	7 6 0
		immediate ⁷	Rs2 ⁵	Rs1 ⁵	funct3 ³	immediate ⁵ opsode ⁷

Η αρχιτεκτονική x86 της Intel

- Εξέλιξη με συμβατότητα με προηγούμενες εκδόσεις
 - 8080 (1974): Μικροεπεξεργαστής 8 bit
 - Αρχιτεκτονική συσσωρευτή, συν 3 ζεύγη δεικτών-καταχωρητών
 - 8086 (1978): Επέκταση του 8080 στα 16 bit
 - Πολύπλοκο σύνολο εντολών (Complex Instruction Set, CISC)
 - 8087 (1980): συνεπεξεργαστής κινητής υποδιαστολής
 - Προστίθενται εντολές κινητής υποδιαστολής και στοίβα καταχωρητών
 - 80286 (1982): Διευθύνσεις των 24 bit, MMU
 - Τμηματοποιημένη χαρτογράφηση και προστασία μνήμης
 - 80386 (1985): Επέκταση στα 32 bit (πλέον IA-32)
 - Πρόσθετοι τρόποι διευθυνσιοδότησης και λειτουργίες
 - Χαρτογράφηση μνήμης με σελιδοποίηση, και τμήματα

Η αρχιτεκτονική x86 της Intel

■ Περαιτέρω εξέλιξη...

- i486 (1989): με διοχέτευση, κρυφές μνήμες στα τσιπ, FPU
 - Συμβατοί ανταγωνιστικοί επεξεργαστές: AMD, Cyrix, ...
- Pentium (1993): υπερβαθμωτός, διαδρομή δεδομένων της τάξης των 64 bit
 - Σε επόμενες εκδόσεις προστέθηκαν εντολές MMX (Multi-Media eXtension)
 - Το διαβόητο σφάλμα FDIV
- Pentium Pro (1995), Pentium II (1997)
 - Νέα μικροαρχιτεκτονική (Παραπομπή σε: Colwell, *The Pentium Chronicles*)
- Pentium III (1999)
 - Προστέθηκαν επεκτάσεις SSE (Streaming SIMD Extensions) και σχετικοί καταχωρητές
- Pentium 4 (2001)
 - Νέα μικροαρχιτεκτονική
 - Προστέθηκαν εντολές SSE2

Η αρχιτεκτονική x86 της Intel

- Και ακόμη παραπέρα...
 - AMD64 (2003): επέκταση της αρχιτεκτονικής στα 64 bit
 - EM64T – Extended Memory 64 Technology (2004)
 - Η Intel υιοθέτησε την AMD64 (με βελτιώσεις)
 - Προστέθηκαν εντολές SSE3
 - Intel Core (2006)
 - Προστέθηκαν εντολές SSE4, υποστήριξη για εικονικές μηχανές
 - AMD64 (ανακοινώθηκε το 2007): Εντολές SSE5
 - Η Intel δεν συμβάδισε, αλλά αντ' αυτού...
 - Προηγμένες διανυσματικές επεκτάσεις (Advanced Vector Extension –ανακοινώθηκαν το 2008)
 - Μεγαλύτεροι καταχωρητές SSE, περισσότερες εντολές
- Αν η Intel δεν έκανε επεκτάσεις έχοντας κατά νου τη συμβατότητα, θα το έκαναν οι ανταγωνιστές της!
 - Τεχνική φινέτσα ≠ επιτυχία στην αγορά

Βασικοί καταχωρητές της x86

Όνομα	Χρήση
31	0
EAX	GPR 0
ECX	GPR 1
EDX	GPR 2
EBX	GPR 3
ESP	GPR 4
EBP	GPR 5
ESI	GPR 6
EDI	GPR 7
CS	Δείκτης τμήματος κώδικα
SS	Δείκτης τμήματος στοίβας (κορυφή στοίβας)
DS	Δείκτης τμήματος δεδομένων 0
ES	Δείκτης τμήματος δεδομένων 1
FS	Δείκτης τμήματος δεδομένων 2
GS	Δείκτης τμήματος δεδομένων 3
EIP	Δείκτης εντολής (PC)
EFLAGS	Κωδικοί συνθήκης

Βασικοί τρόποι διευθυνσιοδότησης της x86

■ Δύο τελεστέοι ανά εντολή

Τύπος τελεστέου προέλευσης/προορισμού	Δεύτερος τελεστέος προέλευσης
Καταχωρητής	Καταχωρητής
Καταχωρητής	Άμεσος
Καταχωρητής	Μνήμη
Μνήμη	Καταχωρητής
Μνήμη	Άμεσος

■ Τρόποι διευθυνσιοδότησης μνήμης

- Η διεύθυνση είναι σε έναν καταχωρητή
- Διεύθυνση = $KTX_{\text{βάσης}} + \text{μετατόπιση}$
- Διεύθυνση = $KTX_{\text{βάσης}} + 2^{\text{κλίμακα}} \times KTX_{\text{αρ/δείκτη}}$
(κλίμακα = 0, 1, 2, ή 3)
- Διεύθυνση = $KTX_{\text{βάσης}} + 2^{\text{κλίμακα}} \times KTX_{\text{αρ/δείκτη}} + \text{μετατόπιση}$

Κωδικοποίηση εντολών της x86

α. JE EIP + μετατόπιση

4 4 8



β. CALL

8

32



γ. MOV EBX, [EDI + 45]

6

1

1

8

8



δ. PUSH ESI

5

3



ε. ADD EAX, #6765

4

3

1

32



στ. TEST EDX, #42

7

1

8

32



■ Κωδικοποίηση μεταβλητού μήκους

- Τα επιθεματικά byte καθορίζουν τον τρόπο διευθυνσιοδότησης
- Τα προθεματικά byte τροποποιούν τη λειτουργία/πράξη
 - Μήκος τελεστέου, επανάληψη, κλείδωμα, κ.ά.

Υλοποίηση της IA-32

- Το σύνθετο σύνολο εντολών δυσχεραίνει την υλοποίηση
 - Το υλικό μετατρέπει τις εντολές σε πιο απλές μικρολειτουργίες
 - Απλές εντολές: 1–1
 - Πολύπλοκες εντολές: 1–πολλά
 - Μικρομηχανή παρόμοιο με την αρχιτεκτονική RISC
 - Λόγω μεριδίου στην αγορά, αυτό είναι βιώσιμο από οικονομική άποψη
- Απόδοση συγκρίσιμη με εκείνη της RISC
 - Οι μεταγλωττιστές αποφεύγουν τις πολύπλοκες εντολές

Άλλες εντολές του RISC-V

- Βασικές εντολές για ακεραίους (RV32I)
 - Όσες περιγράψαμε παραπάνω και επιπλέον οι:

Εντολή	Όνομα	Μορφή	Περιγραφή
Add upper immediate to PC	auipc	U	Πρόσθεση του επάνω άμεσου 20 bit στον PC εγγραφή αθροίσματος σε καταχωρητή
Set if less than	slt	R	Σύγκριση καταχωρητών εγγραφή λογικού (Boolean) αποτελέσματος σε καταχωρητή
Set if less than, unsigned	sltu	R	Σύγκριση καταχωρητών εγγραφή λογικού (Boolean) αποτελέσματος σε καταχωρητή
Set if less than, immediate	slti	I	Σύγκριση καταχωρητών εγγραφή λογικού (Boolean) αποτελέσματος σε καταχωρητή
Set if less than immediate, unsigned	sltiu	I	Σύγκριση καταχωρητών εγγραφή λογικού (Boolean) αποτελέσματος σε καταχωρητή

- Παραλλαγή 64 bit: RV64I
 - οι καταχωρητές έχουν εύρος 64 bit, οι λειτουργίες/πράξεις είναι των 64 bit

Επεκτάσεις συνόλου εντολών

- I (βασικό σύνολο ακεραίων εντολών) – **51 εντολές**
- M: πολλαπλασιασμός, διαίρεση και υπόλοιπο με ακέραιους αριθμούς – **13 εντολές**
- A: ατομικές (αδιαίρετες) λειτουργίες μνήμης – **22 εντολές**
- F: για αριθμούς κινητής υποδιαστολής απλής ακρίβειας – **30 εντολές**
- D: για αριθμούς κινητής υποδιαστολής διπλής ακρίβειας – **32 εντολές**
- C: συμπιεσμένες (compressed) εντολές – **36 εντολές**



κωδικοποίηση 16 bit για τις πιο συχνές εντολές

Κεφάλαιο 2 — Εντολές: Η γλώσσα του υπολογιστή — 102

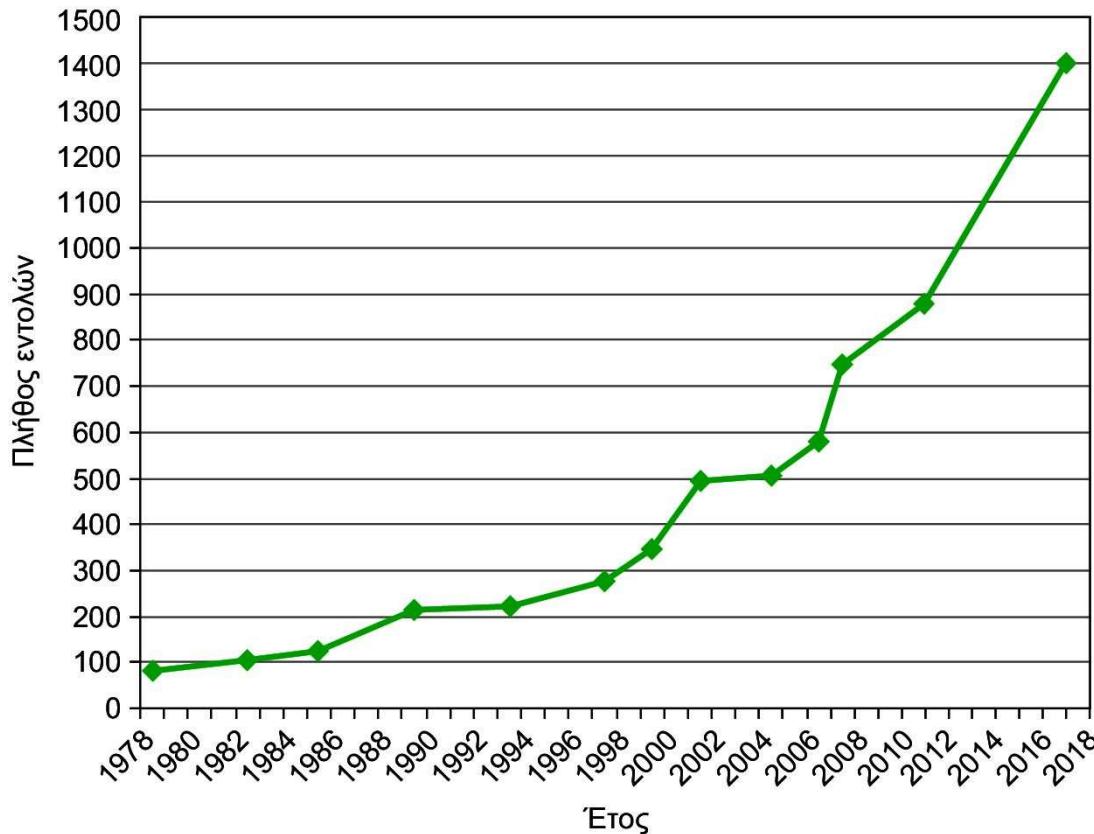
Διαφάνειες διδασκαλίας πρωτότυπου βιβλίου μεταφρασμένες στα ελληνικά (μετάφραση, επιμέλεια, προσθήκες: Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών)

Πλάνες

- Πιο ισχυρές εντολές ⇒ υψηλότερη απόδοση
 - Απαιτούνται λιγότερες εντολές
 - Άλλα οι πολύπλοκες εντολές είναι δύσκολο να υλοποιηθούν με ταχύτητα
 - Ενδέχεται να επιβραδυθούν όλες οι εντολές, ακόμη και οι απλές
 - Οι μεταγλωττιστές διακρίνονται στο να δημιουργούν γρήγορο κώδικα από απλές εντολές
- Για υψηλή απόδοση: χρήση κώδικα συμβολικής γλώσσας
 - Όμως οι σύγχρονοι μεταγλωττιστές χειρίζονται καλύτερα τους σύγχρονους επεξεργαστές
 - Περισσότερες γραμμές κώδικα ⇒ περισσότερα σφάλματα, λιγότερη παραγωγικότητα

Πλάνες

- Αναδρομική συμβατότητα \Rightarrow δεν αλλάζει το σύνολο εντολών
 - Άλλα αυξάνονται οι εντολές



Σύνολο εντολών
της x86

Παγίδες

- Οι διαδοχικές λέξεις δεν βρίσκονται σε διαδοχικές διευθύνσεις
 - Αύξηση ανά 4, όχι ανά 1!
- Η χρήση ενός δείκτη προς μια αυτόματη μεταβλητή αφού γίνει επιστροφή από τη διαδικασία
 - π.χ. επιστροφή του δείκτη με μεταβίβαση μέσω ορίσματος
 - Ο δείκτης παύει να είναι έγκυρος μετά την εξαγωγή από τη στοίβα

Συμπερασματικές παρατηρήσεις

- Σχεδιαστικές αρχές
 1. Η απλότητα ευνοεί την κανονικότητα
 2. Το μικρότερο είναι ταχύτερο
 3. Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς
- Επιτάχυνση της πιο κοινής (συνηθισμένης) περίπτωσης
- Επίπεδα λογισμικού/υλικού
 - Μεταγλωτιστής, συμβολομεταφραστής, υλικό
- Αρχιτεκτονική RISC-V: αντιπροσωπευτική των αρχιτεκτονικών συνόλου εντολών (ISA) τύπου RISC
 - πρβλ. x86