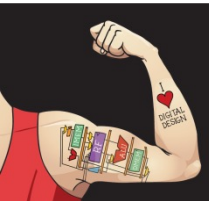
Κεφάλαιο 6

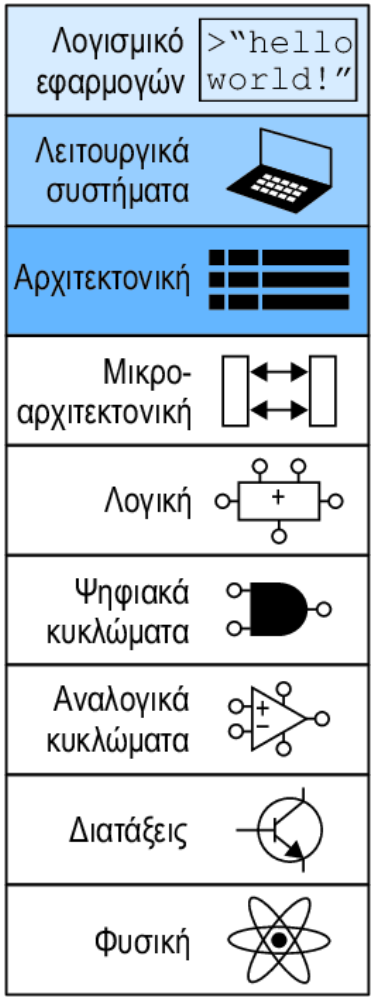
***Ψηφιακή σχεδίαση και αρχιτεκτονική υπολογιστών*: Έκδοση ARM®**

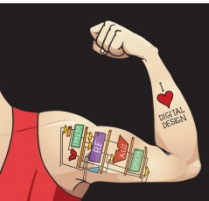
Sarah L. Harris και David Money Harris



Κεφάλαιο 6 :: Θέματα

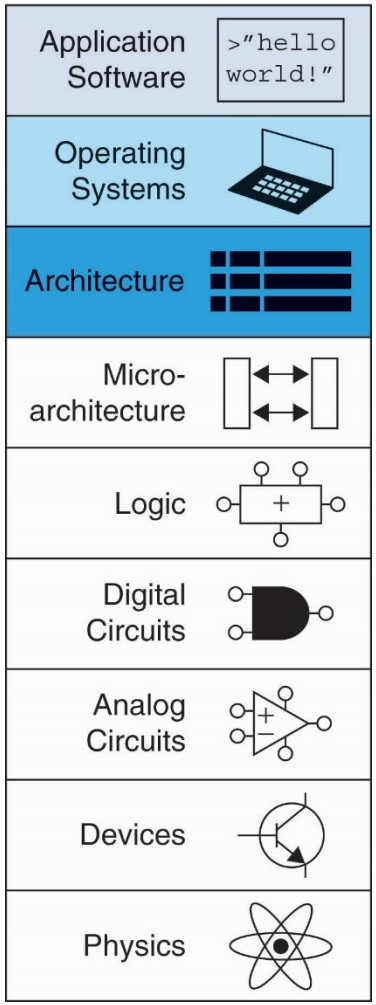
- **Εισαγωγή**
- **Συμβολική γλώσσα**
- **Γλώσσα μηχανής**
- **Προγραμματισμός**
- **Τρόποι διευθυνσιοδότησης**

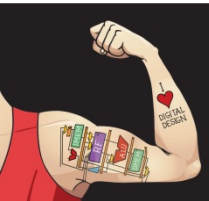




Εισαγωγή

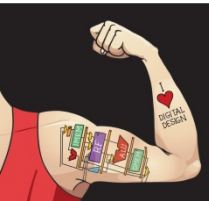
- **Θα ανέβουμε μερικά επίπεδα «αφαίρεσης»**
  - **Αρχιτεκτονική:** Η θεώρηση ενός υπολογιστή από τη σκοπιά του προγραμματιστή
    - Ορίζεται από τις **εντολές** και τις **θέσεις των τελεστέων**
  - **Μικροαρχιτεκτονική:** Το πώς υλοποιείται μια αρχιτεκτονική σε υλικό (δείτε το Κεφάλαιο 7)





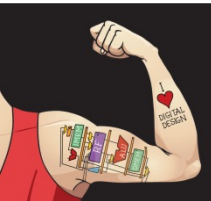
Εντολές

- **Διαταγές στη γλώσσα ενός υπολογιστή**
  - **Συμβολική γλώσσα:** μορφή εντολών που μπορεί να διαβάζεται από ανθρώπους
  - **Γλώσσα μηχανής:** μορφή που μπορεί να διαβάζεται από υπολογιστές (1 και 0)



Αρχιτεκτονική ARM

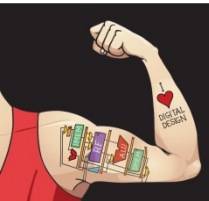
- Αναπτύχθηκε στη δεκαετία του 1980 από την εταιρεία Advanced RISC Machines –γνωστή σήμερα ως ARM Holdings
- Κάθε χρόνο πωλούνται περισσότεροι από 10 δισεκατομμύρια επεξεργαστές ARM
- Σχεδόν όλα τα κινητά τηλέφωνα και οι υπολογιστές tablet περιέχουν περισσότερους από έναν επεξεργαστές ARM
- Πάνω από το 75% των ανθρώπων χρησιμοποιούν προϊόντα με επεξεργαστές ARM
- Χρησιμοποιούνται σε διακομιστές, φωτογραφικές μηχανές, ρομπότ, αυτοκίνητα, φλιπεράκια κ.λπ.



Αρχιτεκτονική ARM

- Αναπτύχθηκε στη δεκαετία του 1980 από την εταιρεία Advanced RISC Machines –γνωστή σήμερα ως ARM Holdings
- Κάθε χρόνο πωλούνται περισσότεροι από 10 δισεκατομμύρια επεξεργαστές ARM
- Σχεδόν όλα τα κινητά τηλέφωνα και οι υπολογιστές tablet περιέχουν περισσότερους από έναν επεξεργαστές ARM
- Πάνω από το 75% των ανθρώπων χρησιμοποιούν προϊόντα με επεξεργαστές ARM
- Χρησιμοποιούνται σε διακομιστές, φωτογραφικές μηχανές, ρομπότ, αυτοκίνητα, φλιπεράκια κ.λπ.

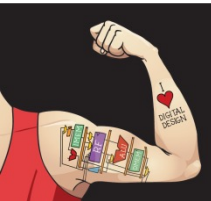
Αν μάθετε μία αρχιτεκτονική, είναι πιο εύκολο να μάθετε και άλλες



Αρχές σχεδίασης αρχιτεκτονικών

Οι υποκείμενες σχεδιαστικές αρχές, όπως διατυπώθηκαν από τους Hennessy and Patterson:

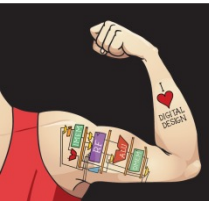
1. **Η κανονικότητα ενισχύει την απλότητα**
2. **Οτιδήποτε είναι σύνηθες και χρησιμοποιείται συχνά πρέπει να είναι γρήγορο**
3. **Το μικρότερο μέγεθος συνεπάγεται πιο γρήγορη εκτέλεση**
4. **Η καλή σχεδίαση απαιτεί και καλούς συμβιβασμούς**



Εντολή: Πρόσθεση

Κώδικας γλώσσας C	Κώδικας συμβολικής γλώσσας της ARM
`a = b + c;`	`ADD a, b, c`

- **ADD:** μνημονικό –υποδεικνύει την πράξη που θα εκτελεστεί
- **b, c:** τελεστέοι προέλευσης (source operands)
- **a:** τελεστέος προορισμού (destination operand)

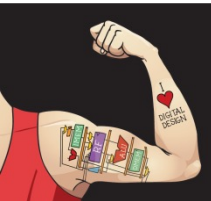


Εντολή : Αφαίρεση

Παρόμοια με την πρόσθεση –μόνο το μνημονικό αλλάζει

Κώδικας γλώσσας C	Κώδικας συμβολικής γλώσσας της ARM
`a = b - c;`	`SUB a, b, c`

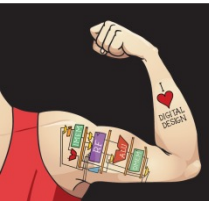
- **SUB:** μνημονικό
- **b, c:** τελεστέοι προέλευσης
- **a:** τελεστέος προορισμού



Σχεδιαστική αρχή 1

Η κανονικότητα ενισχύει την απλότητα

- Η μορφή την εντολών εμφανίζει συνέπεια
- Ίδιο πλήθος τελεστέων (δύο τελεστέοι προέλευσης και ένας τελεστέος προορισμού)
- Ευκολία κωδικοποίησης και χειρισμού στο υλικό



Περισσότερες από μία εντολές

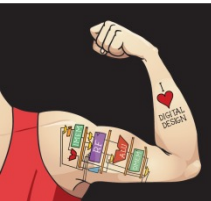
Ο χειρισμός πιο περίπλοκου κώδικα γίνεται με περισσότερες από μία εντολές ARM

Κώδικας γλώσσας C

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

Κώδικας συμβολικής γλώσσας της ARM

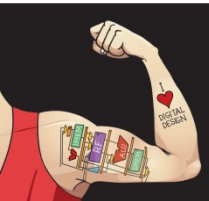
```
ADD t, b, c  ; t = b + c
SUB a, t, d  ; a = t - d
```



Σχεδιαστική αρχή 2

Οτιδήποτε είναι σύνηθες και χρησιμοποιείται συχνά πρέπει να είναι γρήγορο

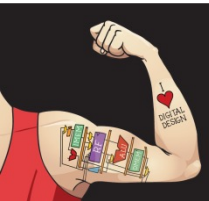
- Η αρχιτεκτονική ARM περιλαμβάνει μόνο απλές εντολές που χρησιμοποιούνται συχνά
- Το υλικό για την αποκωδικοποίηση και την εκτέλεση εντολών μπορεί να είναι απλό, μικρό και γρήγορο
- Για πιο πολύπλοκες εντολές (που δεν εκτελούνται τόσο συχνά) χρησιμοποιούνται πολλές απλές εντολές



Σχεδιαστική αρχή 2

Οτιδήποτε είναι σύνηθες και χρησιμοποιείται συχνά πρέπει να είναι γρήγορο

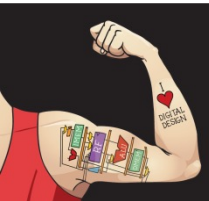
- Η αρχιτεκτονική ARM είναι αρχιτεκτονική τύπου **RISC (Reduced Instruction Set Computer, υπολογιστής περιορισμένου συνόλου εντολών)**, με μικρό αριθμό απλών εντολών
- Άλλες αρχιτεκτονικές, όπως η αρχιτεκτονική x86 της εταιρείας Intel, είναι τύπου **CISC (Complex Instruction Set Computers, υπολογιστές με σύνθετο σύνολο εντολών)**



Θέση τελεστέων

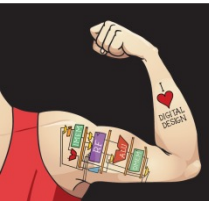
Φυσική θέση στον υπολογιστή

- Καταχωρητές
- Σταθερές (ονομάζεται επίσης *άμεσοι τελεστέοι*)
- Μνήμη



Τελεστέοι: Καταχωρητές

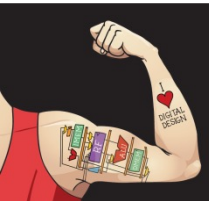
- Η αρχιτεκτονική ARM χρησιμοποιεί 16 καταχωρητές
- Οι καταχωρητές είναι πιο γρήγοροι από τη μνήμη
- Κάθε καταχωρητής έχει μέγεθος 32 bit
- Η αρχιτεκτονική ARM είναι γνωστή ως «αρχιτεκτονική των 32 bit» ακριβώς επειδή εκτελεί πράξεις με δεδομένα που έχουν μέγεθος 32 bit [η έκδοση 8 (ARMv8) έχει επεκταθεί στα 64 bit]



Σχεδιαστική αρχή 3

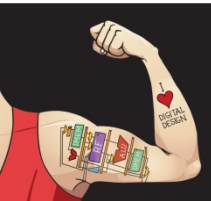
Το μικρότερο μέγεθος συνεπάγεται πιο γρήγορη εκτέλεση

- Η αρχιτεκτονική ARM χρησιμοποιεί μόνο έναν μικρό αριθμό καταχωρητών



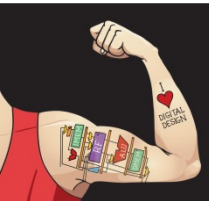
Σύνολο (ή αρχείο) καταχωρητών της αρχιτεκτονικής ARM

Όνομα	Χρήση
R0	Όρισμα / επιστρεφόμενη τιμή / προσωρινή μεταβλητή
R1-R3	Όρισμα / προσωρινές μεταβλητές
R4-R11	Αποθηκευμένες μεταβλητές
R12	Προσωρινή μεταβλητή
R13 (SP)	Δείκτης στοίβας (stack pointer)
R14 (LR)	Καταχωρητής σύνδεσης (link register)
R15 (PC)	Μετρητής προγράμματος (program counter)



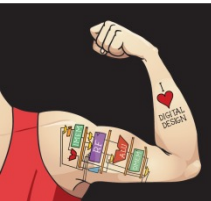
Τελεστέοι: Καταχωρητές

- **Καταχωρητές:**
  - Γράμμα R πριν τον αριθμό, όλα κεφαλαία
  - Παράδειγμα: «R0», ή «καταχωρητής μηδέν», ή «καταχωρητής R0»



Τελεστέοι: Καταχωρητές

- **Καταχωρητές που χρησιμοποιούνται για συγκεκριμένους σκοπούς:**
  - **Αποθηκευμένοι καταχωρητές:** Στους καταχωρητές R4-R11 αποθηκεύονται μεταβλητές
  - **Προσωρινοί καταχωρητές:** Στους καταχωρητές R0-R3 και R12 αποθηκεύονται ενδιάμεσες τιμές
  - Θα αναφερθούμε στους υπόλοιπους αργότερα



Εντολές με καταχωρητές

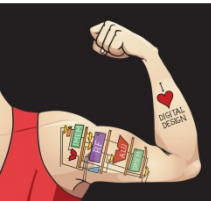
Εντολή ADD (νέα εκδοχή)

Κώδικας γλώσσας C

```
a = b + c
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = a, R1 = b, R2 = c

ADD R0, R1, R2
```



Τελεστέοι: Σταθερές\άμεσοι τελεστέοι

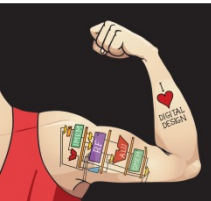
- Πολλές εντολές μπορούν να χρησιμοποιούν σταθερές ή *άμεσους τελεστέους*
- Για παράδειγμα: ADD και SUB
- Η τιμή είναι *άμεσα* διαθέσιμη από την εντολή

Κώδικας γλώσσας C

```
a = a + 4;
b = a - 12;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = a, R1 = b
ADD R0, R0, #4
SUB R1, R0, #12
```



Παραγωγή σταθερών

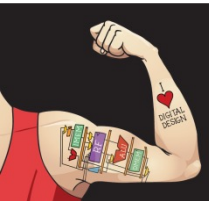
Παραγωγή μικρών σταθερών με την εντολή move (MOV):

Κώδικας γλώσσας C

```
// int: προσημασμένη λέξη
// των 32 bit
int a = 23;
int b = 0x45;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = a, R1 = b
MOV R0, #23
MOV R1, #0x45
```



Παραγωγή σταθερών

Παραγωγή μικρών σταθερών με την εντολή move (MOV):

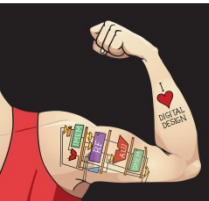
Κώδικας γλώσσας C

```
// int: προσημασμένη λέξη
// των 32 bit
int a = 23;
int b = 0x45;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = a, R1 = b
MOV R0, #23
MOV R1, #0x45
```

Η σταθερά πρέπει να έχει < 8 bit ακρίβειας



Παραγωγή σταθερών

Παραγωγή μικρών σταθερών με την εντολή move (MOV):

Κώδικας γλώσσας C

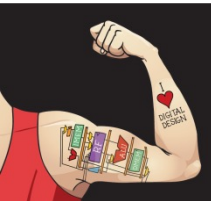
```
// int: προσημασμένη λέξη
// των 32 bit
int a = 23;
int b = 0x45;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = a, R1 = b
MOV R0, #23
MOV R1, #0x45
```

Η σταθερά πρέπει να έχει < 8 bit ακρίβειας

Σημείωση: Η MOV μπορεί επίσης να έχει 2 καταχωρητές: `MOV R7, R9`



Παραγωγή σταθερών

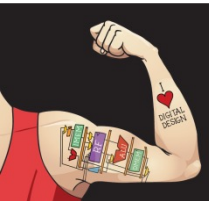
Παραγωγή μεγαλύτερων σταθερών με τις εντολές move (`MOV`) και or (`ORR`):

Κώδικας γλώσσας C

```
int a = 0x7EDC8765;
```

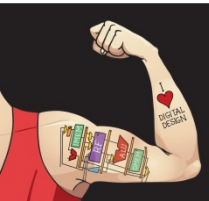
Κώδικας συμβολικής γλώσσας της ARM

```
# R0 = a
MOV R0, #0x7E000000
ORR R0, R0, #0xDC0000
ORR R0, R0, #0x8700
ORR R0, R0, #0x65
```



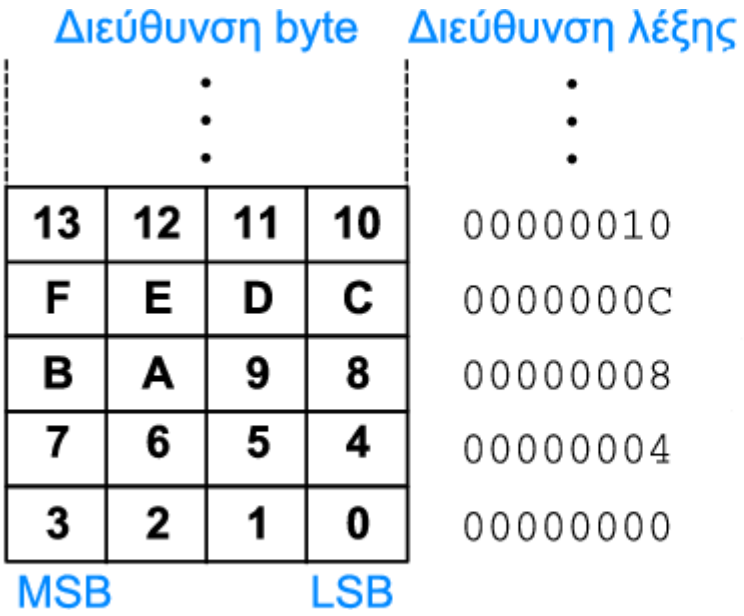
Τελεστέοι: Μνήμη

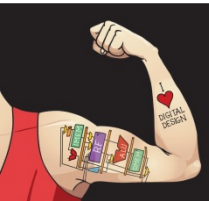
- Οι 16 καταχωρητές δεν αρκούν για μεγάλες ποσότητες δεδομένων
- Περισσότερα δεδομένα μπορούν να αποθηκευτούν στη μνήμη
- Η μνήμη είναι μεγάλη, αλλά αργή
- Οι μεταβλητές που χρησιμοποιούνται συχνά εξακολουθούν να αποθηκεύονται στους καταχωρητές



Μνήμη προσπελάσιμη κατά byte

- Κάθε **byte** δεδομένων έχει μοναδική διεύθυνση
- Λέξη των 32 bit = 4 byte, άρα η διεύθυνση κάθε λέξης είναι πολλαπλάσιο του 4

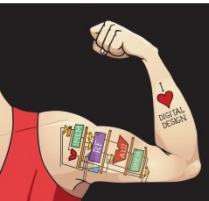




Ανάγνωση μνήμης

- Η ανάγνωση μνήμης ονομάζεται ***φόρτωση***
- **Μνημονικό:** *load register* (`LDR`)
- **Μορφή:**

```
LDR R0, [R1, #12]
```



Ανάγνωση μνήμης

- Η ανάγνωση μνήμης ονομάζεται ***φόρτωση***
- **Μνημονικό:** *load register* (`LDR`)
- **Μορφή:**

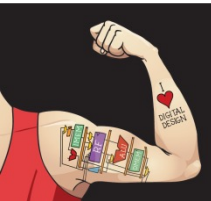
```
LDR R0, [R1, #12]
```

Υπολογισμός διεύθυνσης:

- Πρόσθεση της *διεύθυνσης βάσης* (R1) στη *σχετική απόσταση* (12)
- διεύθυνση = (R1 + 12)

Αποτέλεσμα:

- Ο καταχωρητής R0 διατηρεί τα δεδομένα στη διεύθυνση μνήμης (R1 + 12)



Ανάγνωση μνήμης

- Η ανάγνωση μνήμης ονομάζεται ***φόρτωση***
- **Μνημονικό:** *load register* (`LDR`)
- **Μορφή:**

```
LDR R0, [R1, #12]
```

Υπολογισμός διεύθυνσης:

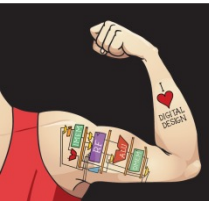
– Πρόσθεση της *διεύθυνσης βάσης* (R1) στη *σχετική απόσταση* (12)

– διεύθυνση = (R1 + 12)

Αποτέλεσμα:

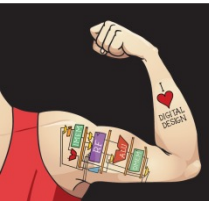
– Ο καταχωρητής R0 διατηρεί τα δεδομένα στη διεύθυνση μνήμης (R1 + 12)

Οποιοσδήποτε καταχωρητής μπορεί να χρησιμοποιηθεί ως διεύθυνση βάσης



Ανάγνωση μνήμης

- **Παράδειγμα:** Φόρτωσε (διάβασε) μια λέξη δεδομένων από τη διεύθυνση μνήμης 8 στον καταχωρητή R3

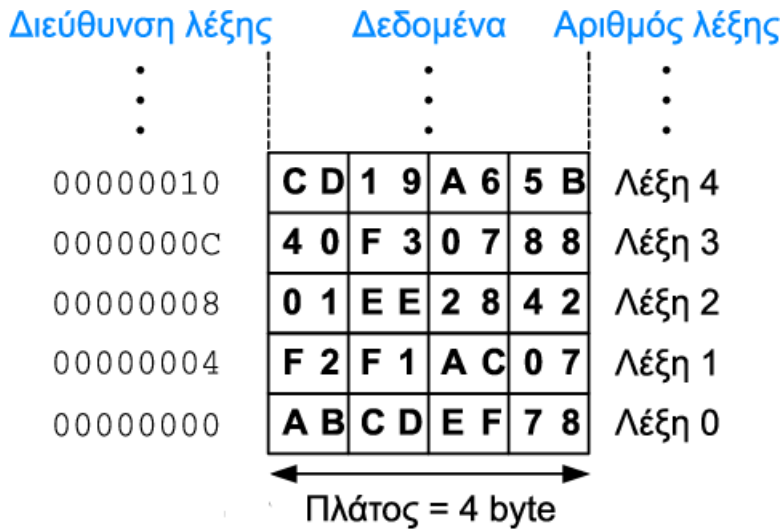


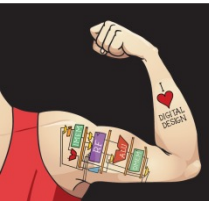
Ανάγνωση μνήμης

- **Παράδειγμα:** Φόρτωσε (διάβασε) μια λέξη δεδομένων από τη διεύθυνση μνήμης 8 στον καταχωρητή R3
  - Διεύθυνση = (R2 + 8) = 8
  - R3 = 0x01EE2842 μετά τη φόρτωση

Κώδικας συμβολικής γλώσσας της ARM

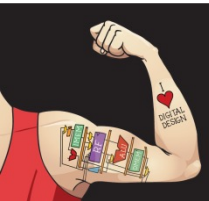
```
MOV R2, #0
LDR R3, [R2, #8]
```





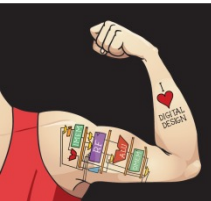
Εγγραφή μνήμης

- Η εγγραφή μνήμης ονομάζεται ***αποθήκευση***
- **Μνημονικό:** *store register* (`STR`)



Εγγραφή μνήμης

- **Παράδειγμα:** Αποθήκευσε (έγγραψε) την τιμή που περιέχει ο καταχωρητής R7 στη λέξη 21 της μνήμης

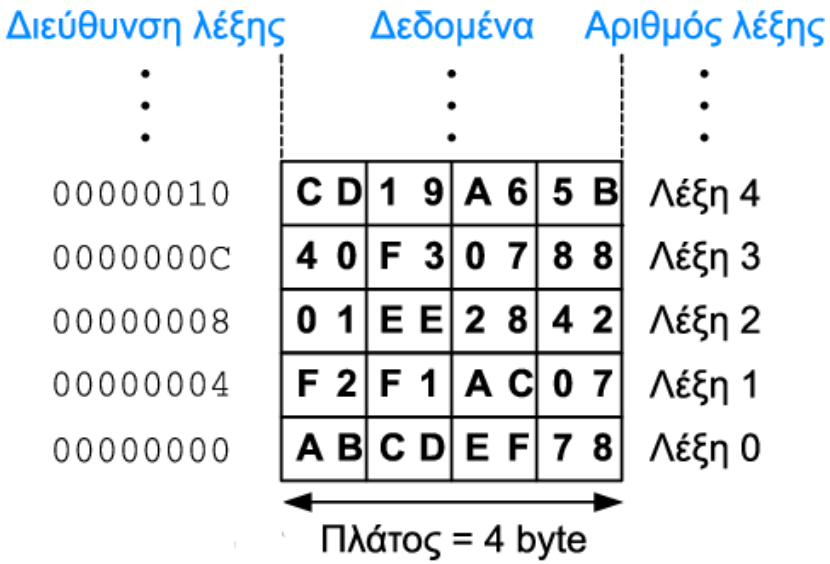


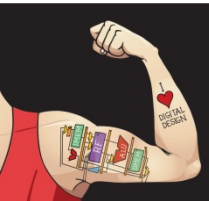
Εγγραφή μνήμης

- **Παράδειγμα:** Αποθήκευσε (έγγραψε) την τιμή που περιέχει ο καταχωρητής R7 στη λέξη 21 της μνήμης
- Διεύθυνση μνήμης = 4 x 21 = 84 = 0x54

Κώδικας συμβολικής γλώσσας της ARM

```
MOV R5, #0
STR R7, [R5, #0x54]
```





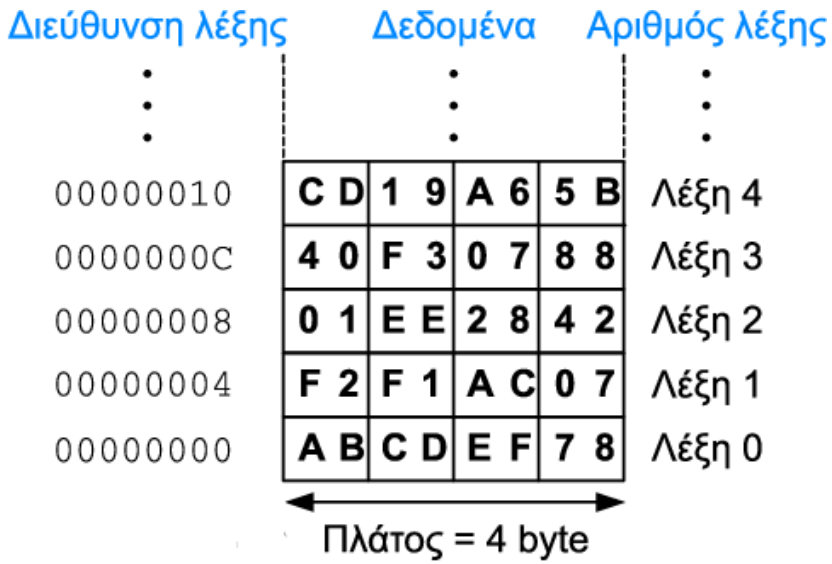
Εγγραφή μνήμης

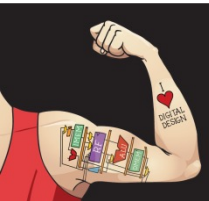
- **Παράδειγμα:** Αποθήκευσε (έγγραψε) την τιμή που περιέχει ο καταχωρητής R7 στη λέξη 21 της μνήμης
- Διεύθυνση μνήμης = 4 x 21 = 84 = 0x54

Κώδικας συμβολικής γλώσσας της ARM

```
MOV R5, #0
STR R7, [R5, #0x54]
```

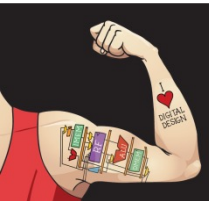
Η σχετική απόσταση μπορεί να είναι γραμμένη σε δεκαδική ή δεκαεξαδική μορφή





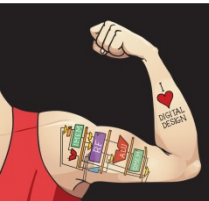
Ανακεφαλαίωση: Προσπέλαση μνήμης

- Η διεύθυνση μιας **λέξης** μνήμης πρέπει να είναι πολλαπλάσιο του 4
- **Παραδείγματα:**
  - Διεύθυνση της λέξης 2 = 2 × 4 = 8
  - Διεύθυνση της λέξης 10 = 10 × 4 = 40



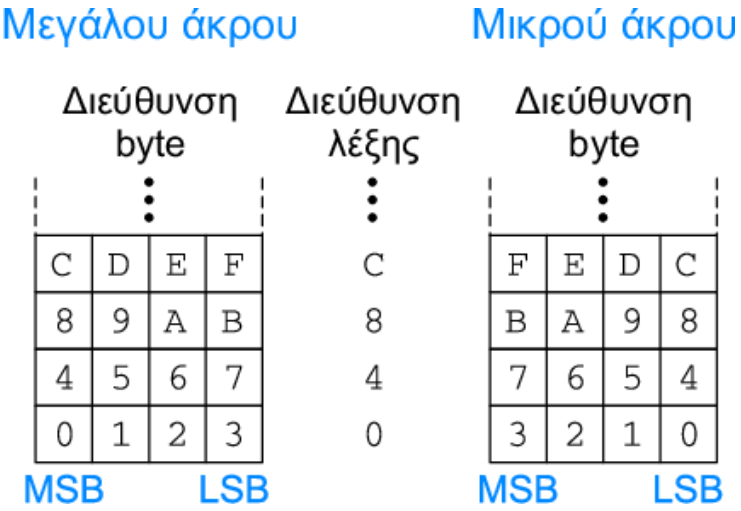
Μνήμη μεγάλου άκρου και μικρού άκρου

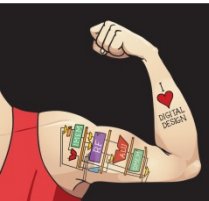
- **Πώς αριθμούνται τα byte εντός μιας λέξης;**



Μνήμη μεγάλου άκρου και μικρού άκρου

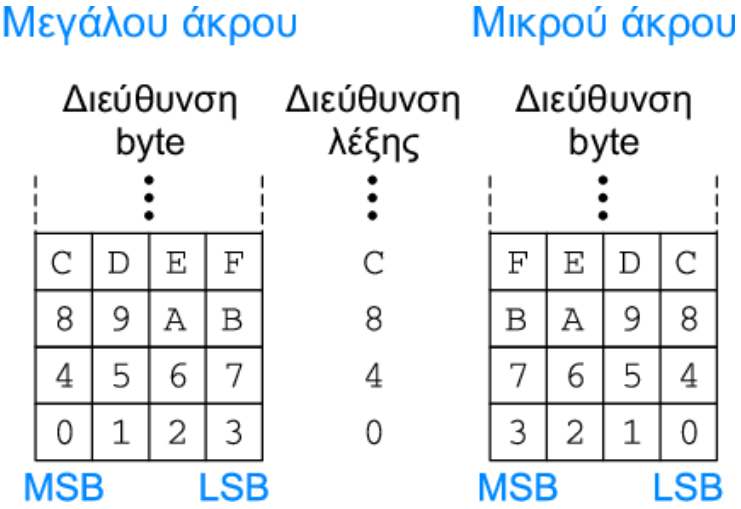
- **Πώς αριθμούνται τα byte εντός μιας λέξης;**
  - **Μορφή μικρού άκρου:** Η αρίθμηση των byte ξεκινάει από το **μικρό** (λιγότερο σημαντικό) άκρο
  - **Μορφή μεγάλου άκρου:** Η αρίθμηση των byte ξεκινάει από το **μεγάλο** (περισσότερο σημαντικό) άκρο

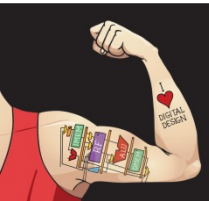




Μνήμη μεγάλου άκρου και μικρού άκρου

- ***Ταξίδια του Γκιούλιβερ* (Jonathan Swift):** Οι «Μικροακρίτες» έσπαζαν τα αυγά τους από τη μικρή πλευρά (μικρό άκρο), ενώ οι «Μεγαλοακρίτες» από τη μεγάλη πλευρά (μεγάλο άκρο)
- **Δεν παίζει ουσιαστικό ρόλο** το ποιος τύπος διευθυνσιοδότησης χρησιμοποιείται –**με εξαίρεση** όταν δύο συστήματα **διαμοιράζονται δεδομένα**



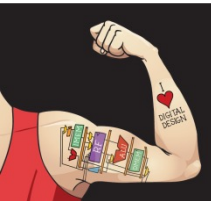


Μνήμη μεγάλου άκρου και μικρού άκρου

Έστω ότι οι καταχωρητές R2 και R5 περιέχουν τις τιμές 8 και 0x23456789

- Αφού εκτελεστεί ο παρακάτω κώδικας σε ένα σύστημα μεγάλου άκρου, ποια τιμή περιέχει ο `R7`;
- Σε ένα σύστημα μικρού άκρου;

```
STR  R5, [R2, #0]
LDRB R7, [R2, #1]
```

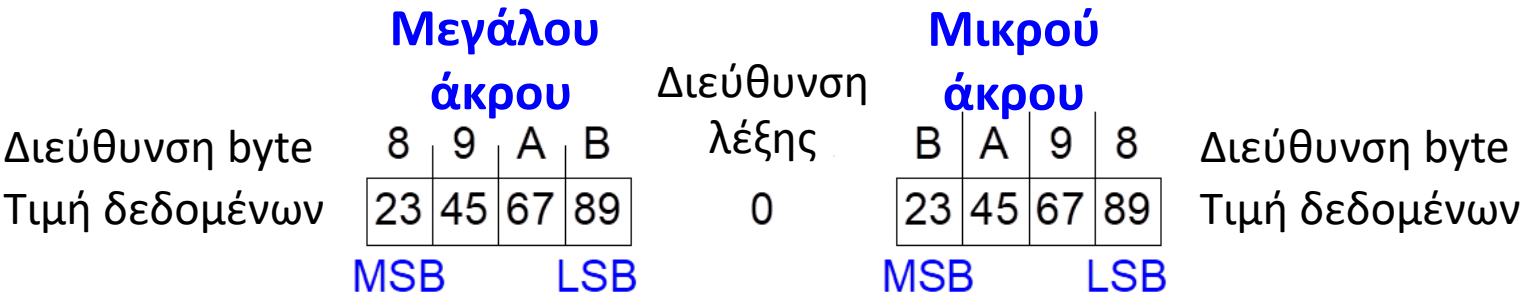


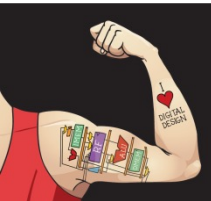
Μνήμη μεγάλου άκρου και μικρού άκρου: Παράδειγμα

Έστω ότι οι καταχωρητές R2 και R5 περιέχουν τις τιμές 8 και 0x23456789

- Αφού εκτελεστεί ο παρακάτω κώδικας σε ένα σύστημα μεγάλου άκρου, ποια τιμή περιέχει ο `R7`;
- Σε ένα σύστημα μικρού άκρου;

```
STR  R5, [R2, #0]
LDRB R7, [R2, #1]
```





Μνήμη μεγάλου άκρου και μικρού άκρου: Παράδειγμα

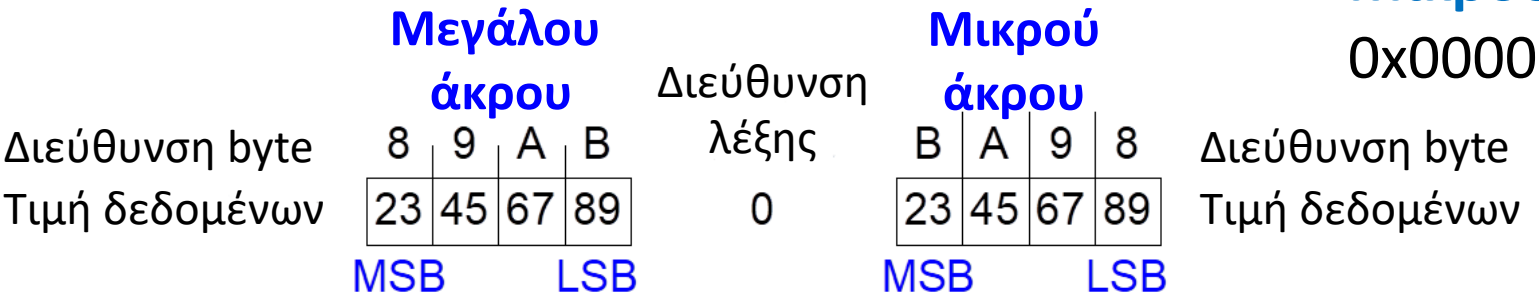
Έστω ότι οι καταχωρητές R2 και R5 περιέχουν τις τιμές 8 και 0x23456789

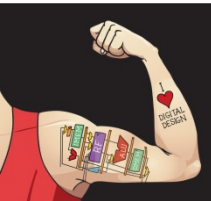
- Αφού εκτελεστεί ο παρακάτω κώδικας σε ένα σύστημα μεγάλου άκρου, ποια τιμή περιέχει ο `R7`;
- Σε ένα σύστημα μικρού άκρου;

```
STR  R5, [R2, #0]
LDRB R7, [R2, #1]
```

Μεγάλου άκρου:
0x00000045

Μικρού άκρου:
0x00000067

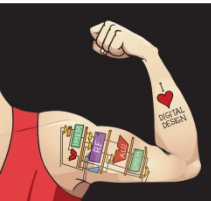




Προγραμματισμός

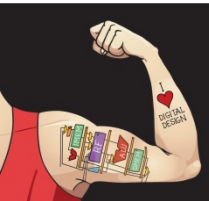
Γλώσσες προγραμματισμού υψηλού επιπέδου:

- Π.χ. C, Java, Python
- Γραμμένες σε υψηλότερο επίπεδο «αφαίρεσης»



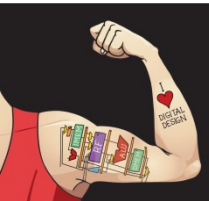
Ada Lovelace, 1815-1852

- Βρετανίδα μαθηματικός
- Έγραψε το πρώτο πρόγραμμα υπολογιστή
- Το πρόγραμμά της υπολόγιζε τους αριθμούς Bernoulli χρησιμοποιώντας την Αναλυτική Μηχανή (Analytical Engine) του Charles Babbage
- Ήταν κόρη του ποιητή Λόρδου Βύρωνα



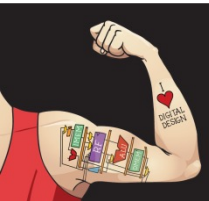
Δομικά στοιχεία προγραμματισμού

- **Εντολές επεξεργασίας δεδομένων**
- **Εκτέλεση υπό συνθήκη**
- **Διακλαδώσεις**
- **Δομές υψηλού επιπέδου:**
  - Εντολές if/else
  - Βρόχοι for
  - Βρόχοι while
  - Πίνακες
  - Κλήσεις συναρτήσεων



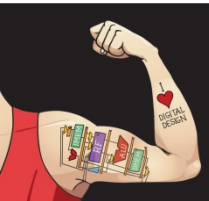
Δομικά στοιχεία προγραμματισμού

- **Εντολές επεξεργασίας δεδομένων**
- **Εκτέλεση υπό συνθήκη**
- **Διακλαδώσεις**
- **Δομές υψηλού επιπέδου:**
  - Εντολές if/else
  - Βρόχοι for
  - Βρόχοι while
  - Πίνακες
  - Κλήσεις συναρτήσεων



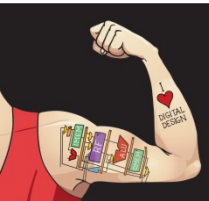
Εντολές επεξεργασίας δεδομένων

- Λογικές πράξεις
- Ολίσθηση / περιστροφή
- Πολλαπλασιασμός



Λογικές εντολές

- `AND`
- `ORR`
- `EOR` **(XOR)**
- `BIC` **(Bit Clear, επαναφορά bit στο 0)**
- `MVN` **(MoVe και NOT)**



Λογικές εντολές: Παραδείγματα

Καταχωρητές προέλευσης

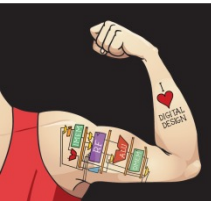
R1	0100 0110	1010 0001	1111 0001	1011 0111
R2	1111 1111	1111 1111	0000 0000	0000 0000

Κώδικας συμβολικής γλώσσας

```
AND  R3, R1, R2
ORR  R4, R1, R2
EOR  R5, R1, R2
BIC  R6, R1, R2
MVN  R7, R2
```

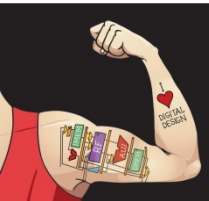
Αποτέλεσμα

R3	0100 0110	1010 0001	0000 0000	0000 0000
R4	1111 1111	1111 1111	1111 0001	1011 0111
R5	1011 1001	0101 1110	1111 0001	1011 0111
R6	0000 0000	0000 0000	1111 0001	1011 0111
R7	0000 0000	0000 0000	1111 1111	1111 1111



Λογικές εντολές: Χρήσεις

- `AND` ή `BIC`: χρήσιμη για την **εφαρμογή μάσκας** στα bit



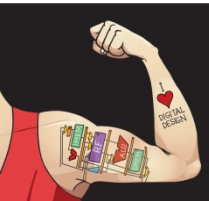
Λογικές εντολές: Χρήσεις

- `AND` ή `BIC`: χρήσιμη για την **εφαρμογή μάσκας** στα bit

  Παράδειγμα: Εφαρμογή μάσκας σε όλα τα byte μιας τιμής εκτός από το λιγότερο σημαντικό

  0xF234012F `AND` 0x000000FF = 0x0000002F

  0xF234012F `BIC` 0xFFFFFF00 = 0x0000002F



Λογικές εντολές: Χρήσεις

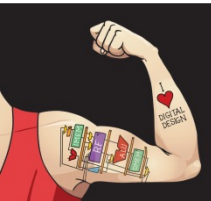
- `AND` ή `BIC`: χρήσιμη για την **εφαρμογή μάσκας** στα bit

  Παράδειγμα: Εφαρμογή μάσκας σε όλα τα byte μιας τιμής εκτός από το λιγότερο σημαντικό

  0xF234012F `AND` 0x000000FF = 0x0000002F

  0xF234012F `BIC` 0xFFFFFF00 = 0x0000002F

- `ORR`: χρήσιμη για την **ανάμειξη** πεδίων bit



Λογικές εντολές: Χρήσεις

- `AND` ή `BIC`: χρήσιμη για την **εφαρμογή μάσκας** στα bit

  Παράδειγμα: Εφαρμογή μάσκας σε όλα τα byte μιας τιμής εκτός από το λιγότερο σημαντικό

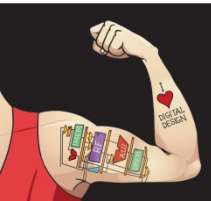
  0xF234012F `AND` 0x000000FF = 0x0000002F

  0xF234012F `BIC` 0xFFFFFF00 = 0x0000002F

- `ORR`: χρήσιμη για την **ανάμειξη** πεδίων bit

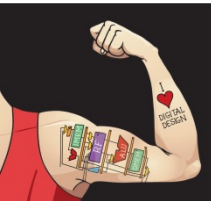
  Παράδειγμα: Ανάμειξη 0xF2340000 και 0x000012BC:

  0xF2340000 `ORR` 0x000012BC = 0xF23412BC



Εντολές ολίσθησης

- `LSL`: logical shift left (λογική ολίσθηση προς τα αριστερά)
- `LSR`: logical shift right (λογική ολίσθηση προς τα δεξιά)
- `ASR`: arithmetic shift right (αριθμητική ολίσθηση προς τα δεξιά)
- `ROR`: rotate right (περιστροφή προς τα δεξιά)



Εντολές ολίσθησης

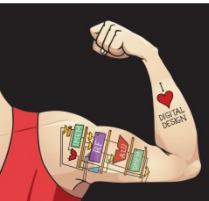
- `LSL`: λογική ολίσθηση προς τα αριστερά

  Παράδειγμα: `LSL R0, R7, #5  ; R0=R7 << 5`

- `LSR`: λογική ολίσθηση προς τα δεξιά

- `ASR`: αριθμητική ολίσθηση προς τα δεξιά

- `ROR`: περιστροφή προς τα δεξιά



Εντολές ολίσθησης

- `LSL`: λογική ολίσθηση προς τα αριστερά

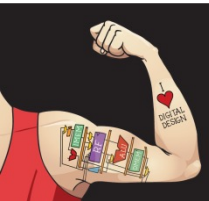
  Παράδειγμα: `LSL R0, R7, #5  ; R0=R7 << 5`

- `LSR`: λογική ολίσθηση προς τα δεξιά

  Παράδειγμα: `LSR R3, R2, #31 ; R3=R2 >> 31`

- `ASR`: αριθμητική ολίσθηση προς τα δεξιά

- `ROR`: περιστροφή προς τα δεξιά



Εντολές ολίσθησης

- `LSL`: λογική ολίσθηση προς τα αριστερά

  Παράδειγμα: `LSL R0, R7, #5  ; R0=R7 << 5`

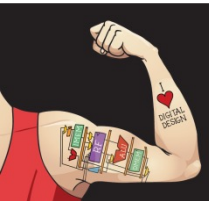
- `LSR`: λογική ολίσθηση προς τα δεξιά

  Παράδειγμα: `LSR R3, R2, #31 ; R3=R2 >> 31`

- `ASR`: αριθμητική ολίσθηση προς τα δεξιά

  Παράδειγμα: `ASR R9, R11, R4 ; R9 = R11 >>> R4`$_{7:0}$

- `ROR`: περιστροφή προς τα δεξιά



Εντολές ολίσθησης

- `LSL`: λογική ολίσθηση προς τα αριστερά

  Παράδειγμα: `LSL R0, R7, #5 ; R0=R7 << 5`

- `LSR`: λογική ολίσθηση προς τα δεξιά

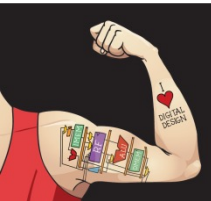
  Παράδειγμα: `LSR R3, R2, #31 ; R3=R2 >> 31`

- `ASR`: αριθμητική ολίσθηση προς τα δεξιά

  Παράδειγμα: `ASR R9, R11, R4 ; R9 = R11 >>>` $R4_{7:0}$

- `ROR`: περιστροφή προς τα δεξιά

  Παράδειγμα: `ROR R8, R1, #3 ; R8=R1 ROR 3`

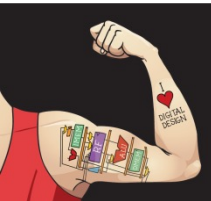


Εντολές ολίσθησης: Παράδειγμα 1

- **Άμεση** ποσότητα ολίσθησης (άμεσος τελεστής των 5 bit)
- Ποσότητα ολίσθησης: 0-31

	Καταχωρητής προέλευσης			
R5	1111 1111	0001 1100	0001 0000	1110 0111

Κώδικας συμβολικής γλώσσας		Αποτέλεσμα			
`LSL R0, R5, #7`	R0	1000 1110	0000 1000	0111 0011	1000 0000
`LSR R1, R5, #17`	R1	0000 0000	0000 0000	0111 1111	1000 1110
`ASR R2, R5, #3`	R2	1111 1111	1110 0011	1000 0010	0001 1100
`ROR R3, R5, #21`	R3	1110 0000	1000 0111	0011 1111	1111 1000



Εντολές ολίσθησης: Παράδειγμα 2

Καταχωρισμένη ποσότητα ολίσθησης (χρησιμοποιεί τα 8 λιγότερο σημαντικά bit του καταχωρητή)

- Ποσότητα ολίσθησης: 0-255

Καταχωρητές προέλευσης

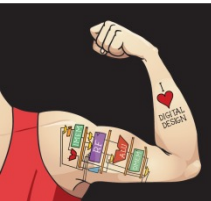
R8	0000 1000	0001 1100	0001 0110	1110 0111
R6	0000 0000	0000 0000	0000 0000	0001 0100

Κώδικας συμβολικής γλώσσας

```
LSL R4, R8, R6
ROR R5, R8, R6
```

Αποτέλεσμα

R4	0110 1110	0111 0000	0000 0000	0000 0000
R5	1100 0001	0110 1110	0111 0000	1000 0001

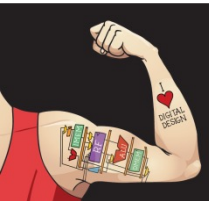


Πολλαπλασιασμός

- **`MUL`:** πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 32 bit

- **`UMULL`** (unsigned multiply long): πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit

- **`SMULL`** (signed multiply long): πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit



Πολλαπλασιασμός

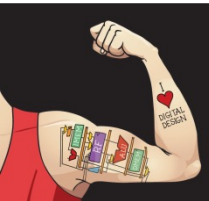
- **MUL:** πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 32 bit

  ```
  MUL R1, R2, R3
  ```

  Αποτέλεσμα: $R1 = (R2 \times R3)_{31:0}$

- **UMULL** (unsigned multiply long): πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit

- **SMULL** (signed multiply long): πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit



Πολλαπλασιασμός

- **MUL:** πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 32 bit

```
MUL R1, R2, R3
```

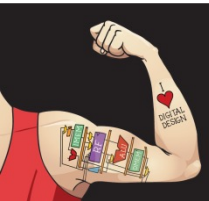
Αποτέλεσμα: $R1 = (R2 \times R3)_{31:0}$

- **UMULL:** unsigned multiply long: πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit

```
UMULL R1, R2, R3, R4
```

Αποτέλεσμα: {R1,R4} = R2 x R3 (R2, R3 μη προσημασμένοι)

- **SMULL:** signed multiply long: πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit



Πολλαπλασιασμός

- **MUL:** πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 32 bit

  `MUL R1, R2, R3`

  Αποτέλεσμα: $R1 = (R2 \times R3)_{31:0}$

- **UMULL:** unsigned multiply long: πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit

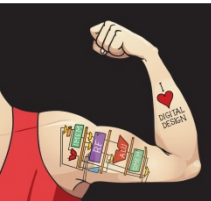
  `UMULL R1, R2, R3, R4`

  Αποτέλεσμα: {R1,R4} = R2 x R3 (R2, R3 μη προσημασμένοι)

- **SMULL:** signed multiply long: πολλαπλασιασμός 32 × 32, αποτέλεσμα με μέγεθος 64 bit

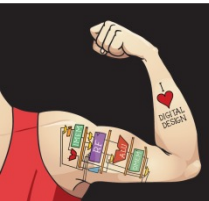
  `SMULL R1, R2, R3, R4`

  Αποτέλεσμα: {R1,R4} = R2 x R3 (R2, R3 προσημασμένοι)



Δομικά στοιχεία προγραμματισμού

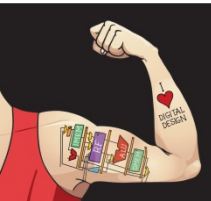
- **Εντολές επεξεργασίας δεδομένων**
- **Εκτέλεση υπό συνθήκη**
- **Διακλαδώσεις**
- **Δομές υψηλού επιπέδου:**
  - Εντολές if/else
  - Βρόχοι for
  - Βρόχοι while
  - Πίνακες
  - Κλήσεις συναρτήσεων



Εκτέλεση υπό συνθήκη

Δεν θέλουμε πάντα να εκτελούμε κώδικα ακολουθιακά

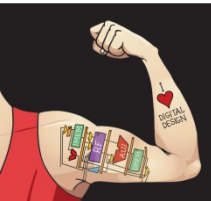
- Για παράδειγμα:
  - Εντολές if/else, βρόχοι while κ.λπ.: Θέλουμε να εκτελείται ο κώδικας μόνο *αν* η συνθήκη είναι αληθής
  - Διακλάδωση: μετάβαση σε άλλο σημείο του κώδικα *αν* η συνθήκη είναι αληθής



Εκτέλεση υπό συνθήκη

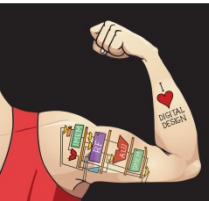
Δεν θέλουμε πάντα να εκτελούμε κώδικα ακολουθιακά

- Για παράδειγμα:
  - Εντολές if/else, βρόχοι while κ.λπ.: Θέλουμε να εκτελείται ο κώδικας μόνο *αν* η συνθήκη είναι αληθής
  - Διακλάδωση: μετάβαση σε άλλο σημείο του κώδικα *αν* η συνθήκη είναι αληθής
- Η αρχιτεκτονική ARM περιλαμβάνει **σημαίες συνθήκης** που μπορούν:
  - Να ενεργοποιηθούν από μια εντολή
  - Να χρησιμοποιούνται για την υπό συνθήκη εκτέλεση μιας εντολής



Αρχιτεκτονική ARM: Σημαίες συνθήκης

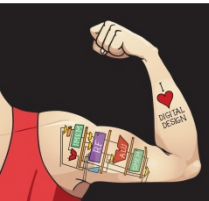
Σημαία	Όνομα	Περιγραφή
N	**N**egative (αρνητικό)	Το αποτέλεσμα της εντολής είναι αρνητικό
Z	**Z**ero (μηδέν)	Το αποτέλεσμα της εντολής είναι ίσο με το μηδέν
C	**C**arry (κρατούμενο)	Η εντολή παράγει μη προσημασμένο κρατούμενο εξόδου
V	o**V**erflow (υπερχείλιση)	Η εντολή προκαλεί υπερχείλιση



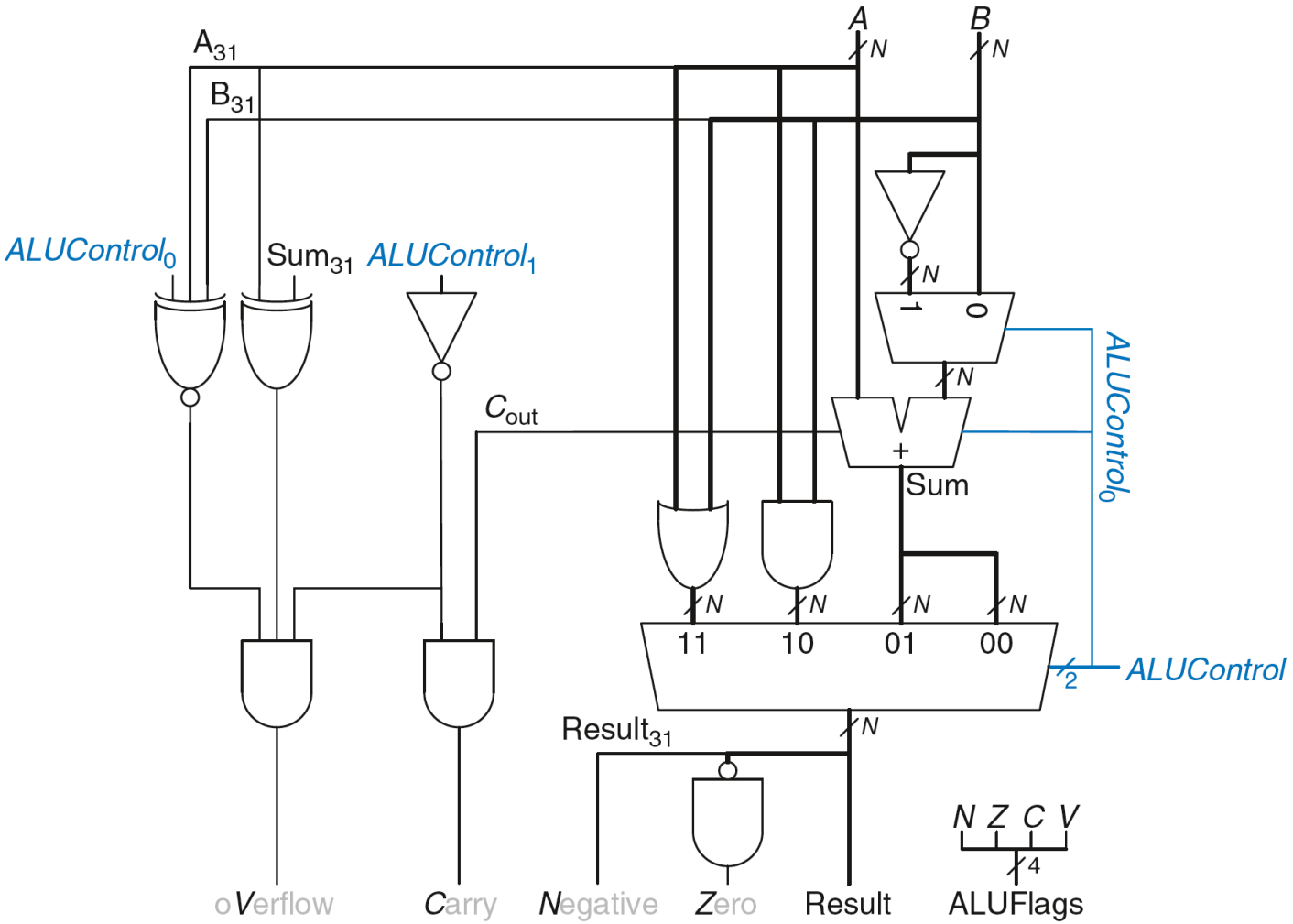
Αρχιτεκτονική ARM: Σημαίες συνθήκης

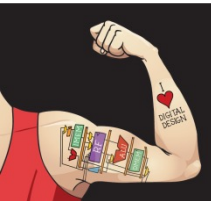
Σημαία	Όνομα	Περιγραφή
N	**N**egative (αρνητικό)	Το αποτέλεσμα της εντολής είναι αρνητικό
Z	**Z**ero (μηδέν)	Το αποτέλεσμα της εντολής είναι ίσο με το μηδέν
C	**C**arry (κρατούμενο)	Η εντολή παράγει μη προσημασμένο κρατούμενο εξόδου
V	o**V**erflow (υπερχείλιση)	Η εντολή προκαλεί υπερχείλιση

- Ενεργοποιούνται από τη μονάδα ALU (Κεφάλαιο 5)
- Είναι αποθηκευμένες στον Καταχωρητή Κατάστασης Τρέχοντος Προγράμματος (Current Program Status Register, CPSR)



Επανάληψη: ARM ALU



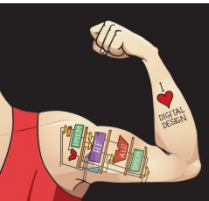


Ενεργοποίηση των σημαιών συνθήκης: *NZCV*

- **Μέθοδος 1:** Εντολή σύγκρισης, `CMP`

  Παράδειγμα: `CMP R5, R6`

  - Εκτελεί: R5 – R6
  - Δεν αποθηκεύει το αποτέλεσμα
  - Ενεργοποιεί σημαίες

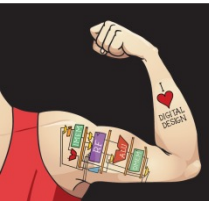


Ενεργοποίηση των σημαιών συνθήκης: *NZCV*

- **Μέθοδος 1:** Εντολή σύγκρισης, `CMP`

  Παράδειγμα: `CMP R5, R6`

  - Εκτελεί: R5 – R6
  - Δεν αποθηκεύει το αποτέλεσμα
  - Ενεργοποιεί σημαίες
    - Δίνει αποτέλεσμα 0, *Z*=1
    - Δίνει αποτέλεσμα αρνητικό, *N*=1
    - Παράγει κρατούμενο εξόδου, *C*=1
    - Προσημασμένη υπερχείλιση, *V*=1

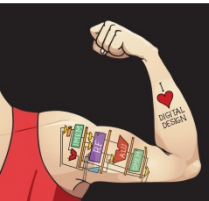


Ενεργοποίηση των σημαιών συνθήκης: *NZCV*

- **Μέθοδος 1:** Εντολή σύγκρισης, `CMP`

  Παράδειγμα: `CMP R5, R6`

  - Εκτελεί: R5–R6
  - Δεν αποθηκεύει το αποτέλεσμα
  - Ενεργοποιεί σημαίες
- **Μέθοδος 2:** Προσάρτηση του χαρακτήρα **`S`** μετά το τέλος του μνημονικού της εντολής



Ενεργοποίηση των σημαιών συνθήκης: *NZCV*

- **Μέθοδος 1:** Εντολή σύγκρισης, `CMP`

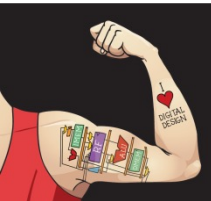
  Παράδειγμα: `CMP R5, R6`

  - Εκτελεί: R5 – R6
  - Δεν αποθηκεύει το αποτέλεσμα
  - Ενεργοποιεί σημαίες

- **Μέθοδος 2:** Προσάρτηση του χαρακτήρα **S** μετά το τέλος του μνημονικού της εντολής

  Παράδειγμα : `ADD`**`S`** `R1, R2, R3`

  - Εκτελεί: R2 + R3
  - Ενεργοποιεί σημαίες: Αν το αποτέλεσμα είναι 0 (Z = 1), αρνητικό (N = 1) κ.λπ.
  - Αποθηκεύει το αποτέλεσμα στον R1



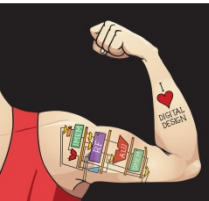
Μνημονικά συνθήκης

- Η εντολή μπορεί να ***εκτελείται υπό συνθήκη*** με βάση τις σημαίες συνθήκης
- Η συνθήκη για την εκτέλεση κωδικοποιείται ως ένα ***μνημονικό συνθήκης*** προσαρτημένο στο μνημονική της εντολής

Παράδειγμα:

```
CMP   R1, R2
SUBNE R3, R5, R8
```

  - **NE:** μνημονικό συνθήκης
  - Η `SUB` θα εκτελεστεί μόνο αν ισχύει R1 ≠ R2 (δηλαδή Z = 0)



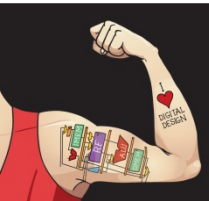
Μνημονικά συνθήκης

cond	Μνημονικό	Όνομα	CondEx
0000	EQ	Equal (ίσοι)	Z
0001	NE	Not equal (μη ίσοι)	$\bar{Z}$
0010	CS / HS	Carry set / Unsigned higher or same (καθορισμός κρατουμένου / μη προσημασμένος υψηλότερος ή ίδιος)	C
0011	CC / LO	Carry clear / Unsigned lower (επαναφορά κρατουμένου / μη προσημασμένος χαμηλότερος)	$\bar{C}$
0100	MI	Minus / Negative (μείον / αρνητικό)	N
0101	PL	Plus / Positive of zero	$\bar{N}$
0110	VS	Overflow / Overflow set (υπερχείλιση / ενεργοποίηση υπερχείλισης)	V
0111	VC	No overflow / Overflow clear (μη υπερχείλιση / επαναφορά υπερχείλισης)	$\bar{V}$
1000	HI	Unsigned higher (μη προσημασμένος υψηλότερος)	$\bar{Z}C$
1001	LS	Unsigned lower or same (μη προσημασμένος χαμηλότερος ή ίδιος)	$Z\ OR\ \bar{C}$
1010	GE	Signed greater than or equal (προσημασμένος μεγαλύτερος ή ίσος)	$\overline{N \oplus V}$
1011	LT	Signed less than (προσημασμένος μικρότερος)	$N \oplus V$
1100	GT	Signed greater than (προσημασμένος μεγαλύτερος)	$\bar{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal (προσημασμένος μικρότερος ή ίσος)	$Z\ OR\ (N \oplus V)$
1110	AL (ή none)	Always / unconditional (πάντα / χωρίς συνθήκη)	Αγνοείται

Εκτέλεση υπό συνθήκη

Παράδειγμα:

```
CMP   R5, R9          ; Εκτελεί την πράξη R5 - R9
                      ; Ενεργοποιεί σημαίες συνθήκης

SUBEQ R1, R2, R3      ; Εκτελείται αν R5==R9 (Z = 1)
ORRMI R4, R0, R9      ; Εκτελείται αν το R5 - R9
                      ; είναι αρνητικό (N = 1)
```



Εκτέλεση υπό συνθήκη

Παράδειγμα:

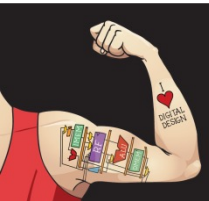
```
CMP   R5, R9          ; Εκτελεί την πράξη R5 - R9
                      ; Ενεργοποιεί σημαίες συνθήκης

SUBEQ R1, R2, R3      ; Εκτελείται αν R5==R9 (Z = 1)
ORRMI R4, R0, R9      ; Εκτελείται αν το R5 - R9
                      ; είναι αρνητικό (N = 1)
```

Έστω ότι R5 = 17, R9 = 23:

Η `CMP` εκτελεί: 17 – 23 = –6 (Ενεργοποιεί σημαίες: $N = 1$, $Z = 0$, $C = 0$, $V = 0$)

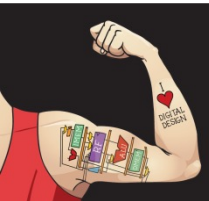
Η `SUBEQ` **δεν εκτελείται** (δεν είναι ίσα: $Z = 0$)

Η `ORRMI` **εκτελείται** επειδή το αποτέλεσμα ήταν αρνητικό ($N = 1$)



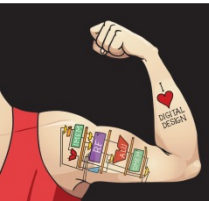
Δομικά στοιχεία προγραμματισμού

- **Εντολές επεξεργασίας δεδομένων**
- **Εκτέλεση υπό συνθήκη**
- **Διακλαδώσεις**
- **Δομές υψηλού επιπέδου:**
  - Εντολές if/else
  - Βρόχοι for
  - Βρόχοι while
  - Πίνακες
  - Κλήσεις συναρτήσεων



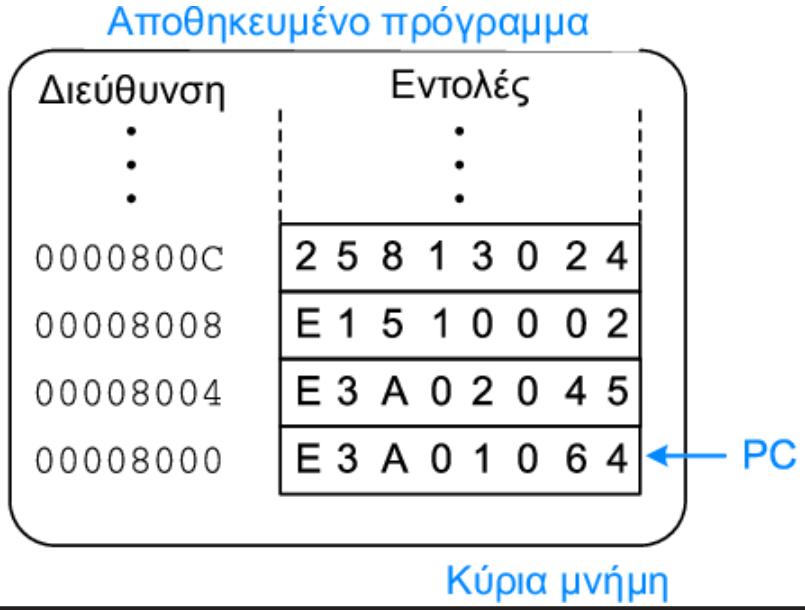
Διακλάδωση

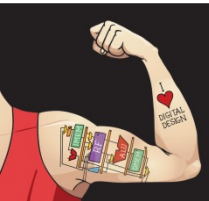
- Οι διακλαδώσεις επιτρέπουν την εκτέλεση εντολών όχι τη μία μετά την άλλη
- Τύποι διακλαδώσεων:
  - **Branch (`B`)**: απλή διακλάδωση
    - Μεταβαίνει σε άλλη εντολή
  - **Branch and link (`BL`)**: διακλάδωση και σύνδεση
    - Θα την περιγράψουμε αργότερα
- Και οι δύο μπορεί να εκτελούνται υπό συνθήκη ή χωρίς συνθήκη



Η έννοια του αποθηκευμένου προγράμματος

Κώδικας συμβολικής γλώσσας	Κώδικας γλώσσας μηχανής
`MOV R1, #100`	`0xE3A01064`
`MOV R2, #69`	`0xE3A02045`
`CMP R1, R2`	`0xE1510002`
`STRHS R3, [R1, #0x24]`	`0x25813024`

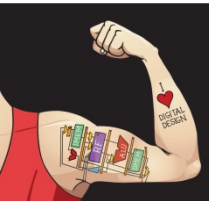




Διακλάδωση χωρίς συνθήκη (B)

Κώδικας συμβολικής γλώσσας της ARM

```
    MOV R2, #17          ; R2 = 17
    B   TARGET           ; Άλμα στον στόχο (TARGET)
    ORR R1, R1, #0x4     ; Δεν εκτελείται

TARGET
        SUB R1, R1, #78      ; R1 = R1 + 78
```

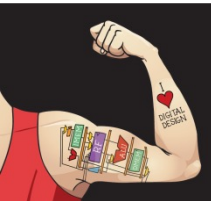


Διακλάδωση χωρίς συνθήκη (B)

Κώδικας συμβολικής γλώσσας της ARM

```
    MOV R2, #17          ; R2 = 17
    B   TARGET           ; Άλμα στον στόχο (TARGET)
    ORR R1, R1, #0x4     ; Δεν εκτελείται

TARGET
        SUB R1, R1, #78      ; R1 = R1 + 78
```

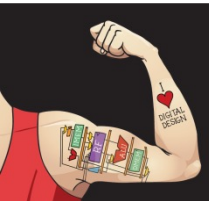
Οι **ετικέτες** (π.χ. TARGET) υποδεικνύουν τη θέση της εντολής. Δεν μπορούν να είναι δεσμευμένες λέξεις (π.χ. ADD, ORR κ.λπ.).



Η διακλάδωση που δεν ακολουθείται

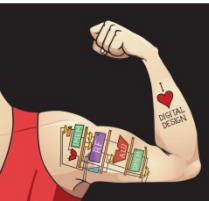
Κώδικας συμβολικής γλώσσας της ARM

```
  MOV   R0, #4          ; R0 = 4
  ADD   R1, R0, R0      ; R1 = R0 + R0 = 8
  CMP   R0, R1          ; Ενεργοποίηση σημαιών με
                        ; με βάση το R0 - R1
  BEQ   THERE           ; Δεν ακολουθείται
                        ; η διακλάδωση(Z = 0)
  ORR   R1, R1, #1      ; R1 = R1 OR R1 = 9
THERE
  ADD R1, R1, 78        ; R1 = R1 + 78 = 87
```



Δομικά στοιχεία προγραμματισμού

- Εντολές επεξεργασίας δεδομένων
- Εκτέλεση υπό συνθήκη
- Διακλαδώσεις
- **Δομές υψηλού επιπέδου:**
  - **Εντολές if/else**
  - **Βρόχοι for**
  - **Βρόχοι while**
  - Πίνακες
  - Κλήσεις συναρτήσεων

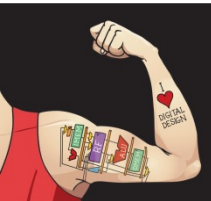


Εντολή if

Κώδικας γλώσσας C

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

```
f = f - i;
```



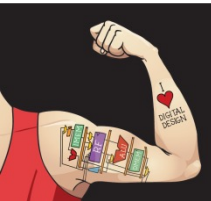
Εντολή if

Κώδικας γλώσσας C

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

f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

  CMP R3, R4        ; Ενεργοποίηση σημαιών
                    ; με βάση το R3 - R4
  BNE L1            ; Αν i!=j, παραλείπεται
                    ; το τμήμα if
  ADD R0, R1, R2    ; f = g + h

L1
  SUB R0, R0, R2    ; f = f - i
```



Εντολή if

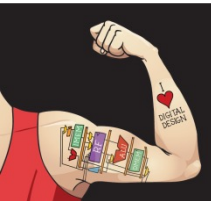
Κώδικας γλώσσας C

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

f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

  CMP R3, R4        ; Ενεργοποίηση σημαιών
                    ; με βάση το R3 - R4
  BNE L1            ; Αν i!=j, παραλείπεται
                    ; το τμήμα if
  ADD R0, R1, R2    ; f = g + h

L1
  SUB R0, R0, R2    ; f = f - i
```

Ο κώδικας συμβολικής γλώσσας ελέγχει την αντίθετη περίπτωση (`i != j`) από ό,τι ο κώδικας υψηλού επιπέδου (`i == j`)



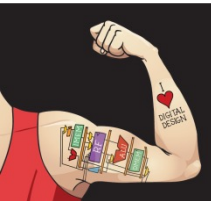
Εντολή if: Εναλλακτικός κώδικας

Κώδικας γλώσσας C

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

Κώδικας συμβολικής γλώσσας της ARM

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

  CMP   R3, R4       ; Ενεργοποίηση σημαιών
                     ; με βάση το R3 - R4
  ADDEQ R0, R1, R2   ; if (i==j) f = g + h
  SUB   R0, R0, R2   ; f = f - i
```



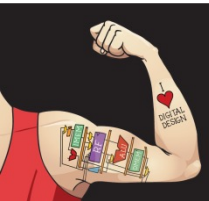
Εντολή if: Εναλλακτικός κώδικας

Αρχικός κώδικας

```
 CMP R3, R4
 BNE L1
 ADD R0, R1, R2
L1
 SUB R0, R0, R2
```

Εναλλακτικός κώδικας συμβολικής γλώσσας

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

 CMP   R3, R4       ; Ενεργοποίηση σημαιών
                    ; με βάση το R3 - R4
 ADDEQ R0, R1, R2   ; if (i==j) f = g + h
 SUB   R0, R0, R2   ; f = f - i
```



Εντολή if: Εναλλακτικός κώδικας

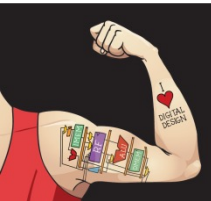
Αρχικός κώδικας

```
 CMP R3, R4
 BNE L1
 ADD R0, R1, R2
L1
 SUB R0, R0, R2
```

Εναλλακτικός κώδικας συμβολικής γλώσσας

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

 CMP   R3, R4       ; Ενεργοποίηση σημαιών
                    ; με βάση το R3 - R4
 ADDEQ R0, R1, R2   ; if (i==j) f = g + h
 SUB   R0, R0, R2   ; f = f - i
```

Χρήσιμος για **μικρά** τμήματα κώδικα που εκτελούνται υπό συνθήκη

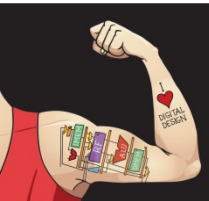


Εντολή if/else

Κώδικας γλώσσας C

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


else
  f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM



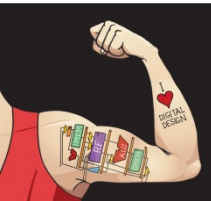
Εντολή if/else

Κώδικας γλώσσας C

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

else
  f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

 CMP R3, R4       ; Ενεργοποίηση σημαιών
                  ; με βάση το R3 - R4
 BNE L1           ; if i!=j, παραλείπεται
                  ; το τμήμα if
 ADD R0, R1, R2   ; f = g + h
 B   L2           ; Άλμα διακλάδωσης μετά
                  ; το τμήμα else
L1
 SUB R0, R0, R2   ; f = f - i
L2
```



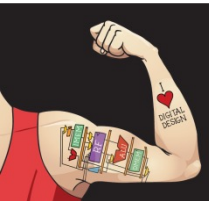
Εντολή if/else: Εναλλακτικός κώδικας

Κώδικας γλώσσας C

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

Κώδικας συμβολικής γλώσσας της ARM

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

 CMP R3, R4          ; Ενεργοποίηση σημαιών
                     ; με βάση το R3 - R4
 ADDEQ R0, R1, R2    ; if (i==j) f = g + h

 SUBNE R0, R0, R2    ; else f = f - i
```



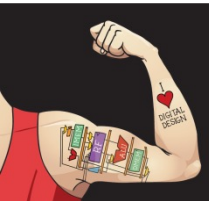
Εντολή if/else: Εναλλακτικός κώδικας

Αρχικός κώδικας

```
 CMP R3, R4
 BNE L1
 ADD R0, R1, R2
 B   L2
L1
 SUB R0, R0, R2
L2
```

Εναλλακτικός κώδικας συμβολικής γλώσσας

```
;R0 = f, R1 = g, R2 = h, R3 = i, R4 = j

 CMP R3, R4          ; Ενεργοποίηση σημαιών
                     ; με βάση το R3 - R4
 ADDEQ R0, R1, R2    ; if (i==j) f = g + h
 SUBNE R0, R0, R2    ; else f = f - i
```

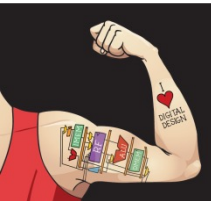


Βρόχοι while

Κώδικας γλώσσας C

```
// Υπολογίζει τη δύναμη
// του x ώστε 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

Κώδικας συμβολικής γλώσσας της ARM



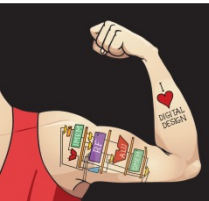
Βρόχοι while

Κώδικας γλώσσας C

```
// Υπολογίζει τη δύναμη
// του x ώστε 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = pow, R1 = x
  MOV   R0, #1          ; pow = 1
  MOV   R1, #0          ; x = 0

WHILE
  CMP R0, #128          ; R0-128
  BEQ DONE              ; if (pow==128)
                        ; έξοδος από τον βρόχο
  LSL R0, R0, #1        ; pow=pow*2
  ADD R1, R1, #1        ; x=x+1
  B   WHILE             ; Επανάληψη βρόχου

DONE
```



Βρόχοι while

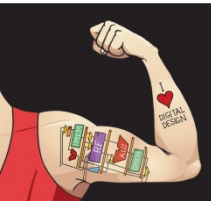
Κώδικας γλώσσας C

```
// Υπολογίζει τη δύναμη
// του x ώστε 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = pow, R1 = x
  MOV   R0, #1          ; pow = 1
  MOV   R1, #0          ; x = 0

WHILE
  CMP R0, #128          ; R0-128
  BEQ DONE              ; if (pow==128)
                        ; έξοδος από τον βρόχο
  LSL R0, R0, #1        ; pow=pow*2
  ADD R1, R1, #1        ; x=x+1
  B   WHILE             ; Επανάληψη βρόχου

DONE
```

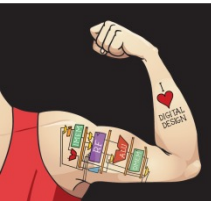
Ο κώδικας συμβολικής γλώσσας ελέγχει την αντίθετη περίπτωση (pow == 128) από ό,τι ο κώδικας της γλώσσας C (pow != 128)



Βρόχοι for

```
for (καθορισμός αρχικής τιμής μεταβλητής;
     συνθήκη; τροποποίηση μεταβλητής)
  εντολή
```

- **`καθορισμός αρχικής τιμής μεταβλητής:`** εκτελείται πριν από την έναρξη του βρόχου
- **`συνθήκη:`** ελέγχεται στο ξεκίνημα κάθε επανάληψης
- **`τροποποίηση μεταβλητής:`** εκτελείται στο τέλος κάθε επανάληψης
- **`εντολή:`** εκτελείται κάθε φορά που ικανοποιείται η συνθήκη

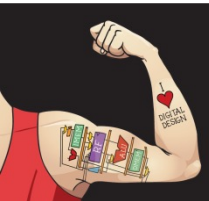


Βρόχοι for

Κώδικας γλώσσας C

```
// Προσθέτει τους
// αριθμούς 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

Κώδικας συμβολικής γλώσσας της ARM



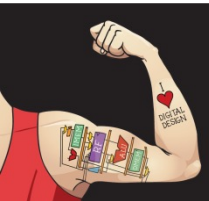
Βρόχοι for

Κώδικας γλώσσας C

```
// Προσθέτει τους
// αριθμούς 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

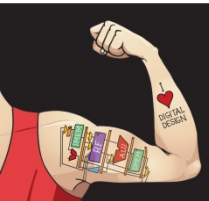
Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = i, R1 = sum
  MOV   R0, #1          ; i = 1
  MOV   R1, #0          ; sum = 0

FOR
  CMP R0, #10           ; R0-10
  BEQ DONE              ; if (i==10)
                        ; έξοδος από τον βρόχο
  ADD R1, R1, R0        ; sum=sum + i
  ADD R0, R0, #1        ; i = i + 1
  B   FOR               ; Επανάληψη βρόχου

DONE
```



Βρόχοι for: Μειούμενη μεταβλητή βρόχου

Στην αρχιτεκτονική ARM, οι μειούμενες μεταβλητές βρόχου είναι πιο αποδοτικές



Βρόχοι for: Μειούμενη μεταβλητή βρόχου

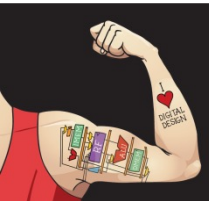
Στην αρχιτεκτονική ARM, οι μειούμενες μεταβλητές βρόχου είναι πιο αποδοτικές

Κώδικας γλώσσας C

```
// Προσθέτει τους
// αριθμούς 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = i, R1 = sum
  MOV   R0, #9          ; i = 9
  MOV   R1, #0          ; sum = 0

FOR
  ADD   R1, R1, R0      ; sum=sum + i
  SUBS  R0, R0, #1      ; i = i - 1
                        ; και ενεργοποίηση
                        ; σημαιών
  BNE   FOR             ; if (i!=0)
                        ; επανάληψη βρόχου
```



Βρόχοι for: Μειούμενη μεταβλητή βρόχου

Στην αρχιτεκτονική ARM, οι μειούμενες μεταβλητές βρόχου είναι πιο αποδοτικές

Κώδικας γλώσσας C

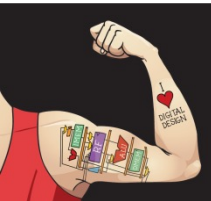
```
// προσθέτει τους
// αριθμούς 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = i, R1 = sum
  MOV   R0, #9          ; i = 9
  MOV   R1, #0          ; sum = 0

FOR
  ADD   R1, R1, R0      ; sum=sum + i
  SUBS  R0, R0, #1      ; i = i - 1
                        ; και ενεργοποίηση
                        ; σημαιών
  BNE   FOR             ; if (i!=0)
                        ; Επανάληψη βρόχου
```

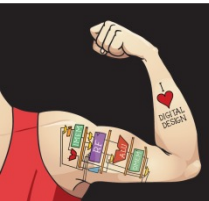
Εξοικονομεί 2 εντολές ανά επανάληψη:

- Μείωση κατά 1 της μεταβλητής βρόχου και σύγκριση: `SUBS R0, R0, #1`
- Μόνο 1 διακλάδωση (αντί για 2)



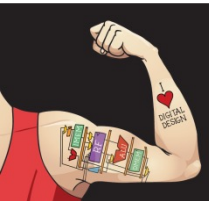
Δομικά στοιχεία προγραμματισμού

- **Εντολές επεξεργασίας δεδομένων**
- **Εκτέλεση υπό συνθήκη**
- **Διακλαδώσεις**
- **Δομές υψηλού επιπέδου:**
  - Εντολές if/else
  - Βρόχοι for
  - Βρόχοι while
  - **Πίνακες**
  - Κλήσεις συναρτήσεων



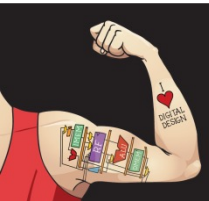
Πίνακες

- Μας επιτρέπουν να προσπελάσουμε μεγάλες ποσότητες παρόμοιων δεδομένων
  - **Αριθμοδείκτης:** προσπέλαση κάθε στοιχείου
  - **Μέγεθος ή μήκος:** πλήθος στοιχείων



Πίνακες

- Πίνακας με 5 στοιχεία
  - **Διεύθυνση βάσης** = 0x14000000 (διεύθυνση του πρώτου στοιχείου, scores[0])
  - Τα στοιχεία προσπελάζονται σε σχέση με τη διεύθυνση βάσης



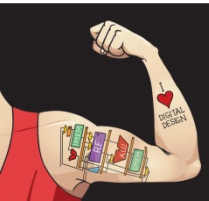
Προσπέλαση πινάκων

Κώδικας γλώσσας C

```
int array[5];
array[0] = array[0] * 8;
array[1] = array[1] * 8;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = διεύθυνση βάσης πίνακα
```



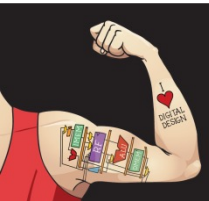
Προσπέλαση πινάκων

Κώδικας γλώσσας C

```
int array[5];
array[0] = array[0] * 8;
array[1] = array[1] * 8;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = διεύθυνση βάσης πίνακα
  MOV R0, #0x60000000          ; R0 = 0x60000000

  LDR R1, [R0]                 ; R1 = array[0]
  LSL R1, R1, 3                ; R1 = R1 << 3 = R1*8
  STR R1, [R0]                 ; array[0] = R1

  LDR R1, [R0, #4]             ; R1 = array[1]
  LSL R1, R1, 3                ; R1 = R1 << 3 = R1*8
  STR R1, [R0, #4]             ; array[1] = R1
```



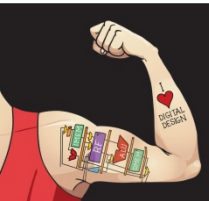
Πίνακες με τη χρήση βρόχων for

Κώδικας γλώσσας C

```
int array[200];
int i;

for (i=199; i >= 0; i = i - 1)
     array[i] = array[i] * 8;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = διεύθυνση βάσης πίνακα, R1 = i
```



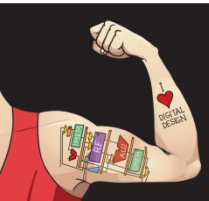
Πίνακες με τη χρήση βρόχων for

Κώδικας γλώσσας C

```
int array[200];
int i;

for (i=199; i >= 0; i = i - 1)
     array[i] = array[i] * 8;
```

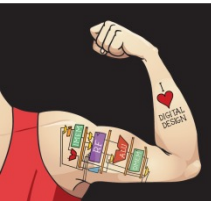
Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = διεύθυνση βάσης πίνακα, R1 = i
  MOV R0, 0x60000000
  MOV R1, #199

FOR
  LDR  R2, [R0, R1, LSL #2]     ; R2 = array(i)
  LSL  R2, R2, #3               ; R2 = R2<<3 = R3*8
  STR  R2, [R0, R1, LSL #2]     ; array(i) = R2
  SUBS R0, R0, #1               ; i = i - 1
                                ; και ενεργοποίηση σημαιών
  BPL  FOR                      ; if (i>=0) επανάληψη βρόχου
```



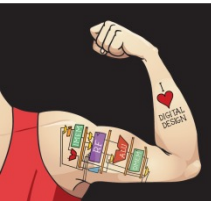
Κώδικας ASCII

- Αμερικανικός Πρότυπος Κώδικας για την Ανταλλαγή Πληροφοριών (American Standard Code for Information Interchange)
- Σε κάθε χαρακτήρα κειμένου αντιστοιχίζεται μια μοναδική τιμή byte
  - Για παράδειγμα, S = 0x53, a = 0x61, A = 0x41
  - Τα πεζά και τα κεφαλαία γράμματα διαφέρουν κατά 0x20 (32)



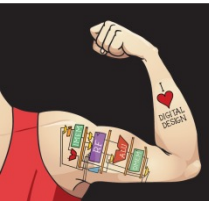
Αντιστοίχιση χαρακτήρων

#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
20	space	30	0	40	@	50	P	60	`	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	‘	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	–	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o		



Δομικά στοιχεία προγραμματισμού

- **Εντολές επεξεργασίας δεδομένων**
- **Εκτέλεση υπό συνθήκη**
- **Διακλαδώσεις**
- **Δομές υψηλού επιπέδου:**
  - Εντολές if/else
  - Βρόχοι for
  - Βρόχοι while
  - Πίνακες
  - **Κλήσεις συναρτήσεων**

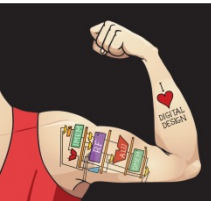


Κλήσεις συναρτήσεων

- **Καλούσα συνάρτηση** (εδώ η `main`)
- **Καλούμενη συνάρτηση** (εδώ η `sum`)

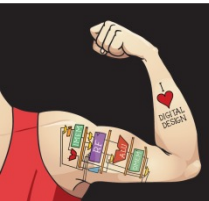
Κώδικας γλώσσας C

```
void main()
{
  int y;
  y = sum(42, 7);
  ...
}

int sum(int a, int b)
{
  return (a + b);
}
```



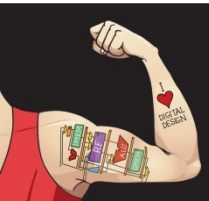
Συμβάσεις για συναρτήσεις

- **Καλούσα συνάρτηση:**
  - Μεταβιβάζει **ορίσματα** στην καλούμενη συνάρτηση
  - Μεταβαίνει (κάνει άλμα) στην καλούμενη συνάρτηση



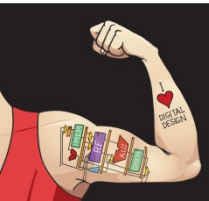
Συμβάσεις για συναρτήσεις

- **Καλούσα συνάρτηση:**
  - Μεταβιβάζει **ορίσματα** στην καλούμενη συνάρτηση
  - Μεταβαίνει (κάνει άλμα) στην καλούμενη συνάρτηση
- **Καλούμενη συνάρτηση:**
  - **Εκτελεί** τη συνάρτηση
  - **Επιστρέφει** το αποτέλεσμα στην καλούσα συνάρτηση
  - **Επιστρέφει** στο σημείο της κλήσης
  - **Δεν πρέπει να υπερεγγράφει** καταχωρητές ή μνήμη που χρειάζεται η καλούσα συνάρτηση



Συμβάσεις για συναρτήσεις: Αρχιτεκτονική ARM

- **Κλήση συνάρτησης:** branch and link `BL`
- **Επιστροφή** από συνάρτηση: Τα περιεχόμενα του καταχωρητή σύνδεσης αντιγράφονται στον μετρητή PC: `MOV PC, LR`
- **Ορίσματα:** `R0-R3`
- **Επιστρεφόμενη τιμή:** `R0`



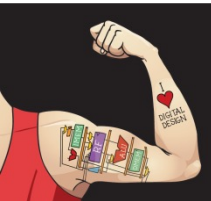
Κλήσεις συναρτήσεων

Κώδικας γλώσσας C

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

Κώδικας συμβολικής γλώσσας της ARM

```
0x00000200 MAIN     BL  SIMPLE
0x00000204          ADD R4, R5, R6
...

0x00401020 SIMPLE   MOV PC, LR
```



Κλήσεις συναρτήσεων

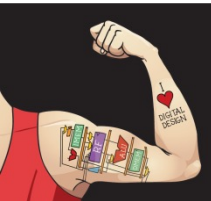
Κώδικας γλώσσας C

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

Κώδικας συμβολικής γλώσσας της ARM

```
0x00000200 MAIN     BL   SIMPLE
0x00000204          ADD R4, R5, R6
...

0x00401020 SIMPLE   MOV PC, LR
```

Το `void` σημαίνει ότι η συνάρτηση `simple` δεν επιστρέφει κάποια τιμή



Κλήσεις συναρτήσεων

Κώδικας γλώσσας C

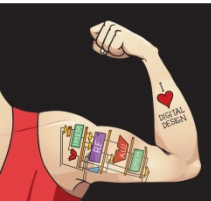
```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

Κώδικας συμβολικής γλώσσας της ARM

```
0x00000200 MAIN     BL   SIMPLE
0x00000204          ADD R4, R5, R6
...

0x00401020 SIMPLE   MOV PC, LR
```

BL διακλαδώνεται στο SIMPLE
LR = PC + 4 = 0x00000204

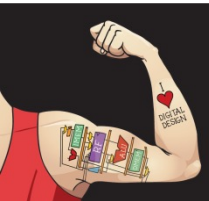
MOV PC, LR θέτει PC = LR
(η επόμενη εντολή που εκτελείται βρίσκεται στη διεύθυνση 0x00000200)



Ορίσματα εισόδου και επιστρεφόμενη τιμή

Συμβάσεις της αρχιτεκτονικής ARM:

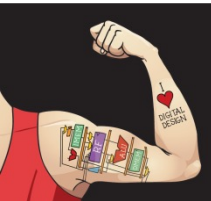
- Τιμές ορισμάτων: Καταχωρητές `R0` - `R3`
- Επιστρεφόμενη τιμή: Καταχωρητής `R0`



Ορίσματα εισόδου και επιστρεφόμενη τιμή

Κώδικας γλώσσας C

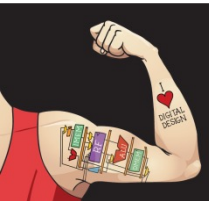
```
int main()
{
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 ορίσματα
  ...
}

int diffofsums(int f, int g, int h, int i)
{
  int result;
  result = (f + g) - (h + i);
  return result;                // επιστρεφόμενη τιμή
}
```



Ορίσματα εισόδου και επιστρεφόμενη τιμή

Κώδικας συμβολικής γλώσσας της ARM

```
; R4 = y

MAIN
  ...
  MOV R0, #2          ; Όρισμα 0 = 2
  MOV R1, #3          ; Όρισμα 1 = 3
  MOV R2, #4          ; Όρισμα 2 = 4
  MOV R3, #5          ; Όρισμα 3 = 5
  BL DIFFOFSUMS       ; Κλήση συνάρτησης
  MOV R4, R0          ; y = επιστρεφόμενη τιμή
  ...
; R4 = result
DIFFOFSUMS
  ADD R8, R0, R1      ; R8 = f + g
  ADD R9, R2, R3      ; R9 = h + i
  SUB R4, R8, R9      ; result = (f + g) - (h + i)
  MOV R0, R4          ; Τοποθέτηση επιστρεφόμενης
                      ; τιμής στον καταχωρητή R0
  MOV PC, LR          ; Επιστροφή στην καλούσα συνάρτηση
```

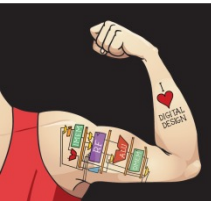


Ορίσματα εισόδου και επιστρεφόμενη τιμή

Κώδικας συμβολικής γλώσσας της ARM

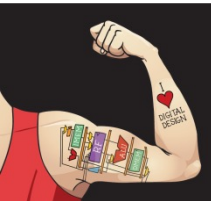
```
; R4 = result
DIFFOFSUMS
  ADD R8, R0, R1      ; R8 = f + g
  ADD R9, R2, R3      ; R9 = h + i
  SUB R4, R8, R9      ; result = (f + g) - (h + i)
  MOV R0, R4          ; Τοποθέτηση επιστρεφόμενης
                      ; τιμής στον καταχωρητή R0
  MOV PC, LR          ; Επιστροφή στην καλούσα συνάρτηση
```

- Η `diffofsums` υπερέγγραψε 3 καταχωρητές: `R4`, `R8`, `R9`
- Η `diffofsums` μπορεί να χρησιμοποιεί τη *στοίβα* για την προσωρινή αποθήκευση των περιεχομένων των καταχωρητών



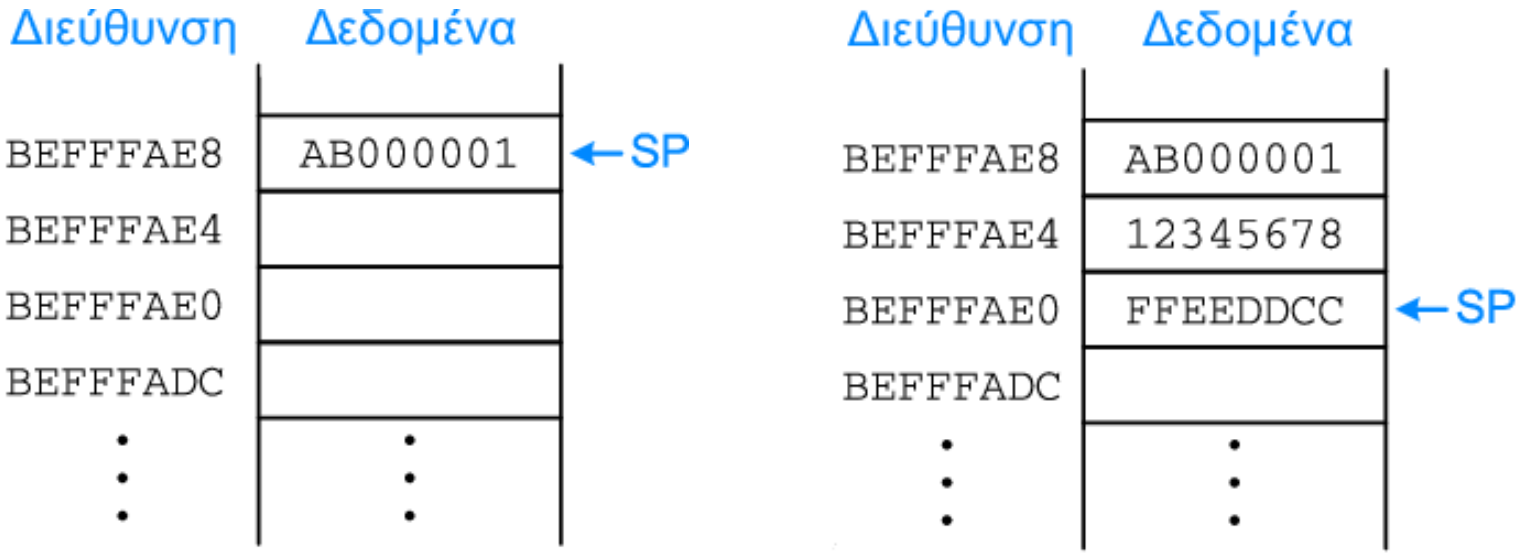
Η στοίβα

- Τμήμα της μνήμης που χρησιμοποιείται για την αποθήκευση προσωρινών μεταβλητών
- Σαν τις στοίβες με πιάτα: ουρά LIFO (last-in-first-out, «τελευταίο μέσα-πρώτο έξω»)
- ***Επεκτείνεται*:** Χρησιμοποιεί περισσότερη μνήμη όταν απαιτείται περισσότερος χώρος
- ***Συρρικνώνεται*:** Χρησιμοποιεί λιγότερη μνήμη όταν δεν απαιτείται πλέον ο χώρος

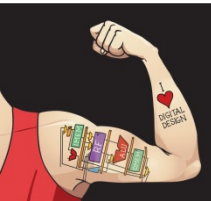


Η στοίβα

- Ο χώρος που καταλαμβάνει αυξάνεται προς τα κάτω (από υψηλότερες προς χαμηλότερες διευθύνσεις μνήμης)
- Δείκτης στοίβας (stack pointer, SP): Δείχνει στην κορυφή της στοίβας



Η στοίβα επεκτείνεται κατά 2 λέξεις

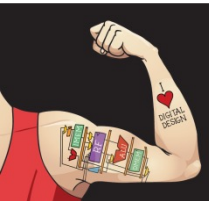


Πώς οι συναρτήσεις χρησιμοποιούν τη στοίβα

- Οι καλούμενες συναρτήσεις δεν πρέπει να έχουν ανεπιθύμητες παρενέργειες
- Όμως η `diffofsums` υπερεγγράφει τα περιεχόμενα 3 καταχωρητών: `R4`, `R8`, `R9`

Κώδικας συμβολικής γλώσσας της ARM

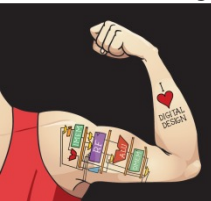
```
; R4 = result
DIFFOFSUMS
  ADD R8, R0, R1      ; R8 = f + g
  ADD R9, R2, R3      ; R9 = h + i
  SUB R4, R8, R9      ; result = (f + g) - (h + i)
  MOV R0, R4          ; Τοποθέτηση επιστρεφόμενης
                      ; τιμής στον καταχωρητή R0
  MOV PC, LR          ; Επιστροφή στην καλούσα συνάρτηση
```



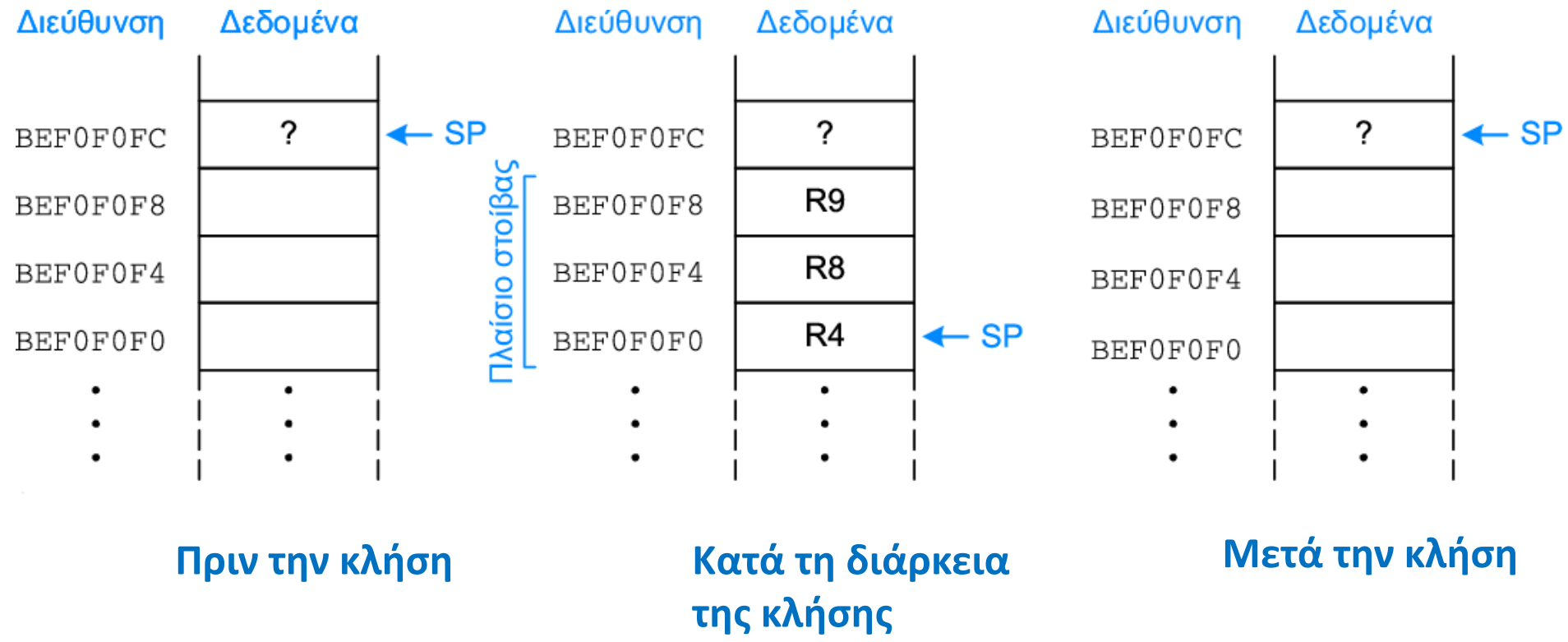
Αποθήκευση τιμών από καταχωρητές στη στοίβα

Κώδικας συμβολικής γλώσσας της ARM

```
; R2 = result
DIFFOFSUMS
  SUB SP, SP, #12     ; Δημιουργία χώρου στη στοίβα
                      ; για 3 καταχωρητές
  STR R4, [SP, #-8]   ; Αποθήκευση του R4 στη στοίβα
  STR R8, [SP, #-4]   ; Αποθήκευση του R8 στη στοίβα
  STR R9, [SP]        ; Αποθήκευση του R9 στη στοίβα
  ADD R8, R0, R1      ; R8 = f + g
  ADD R9, R2, R3      ; R9 = h + i
  SUB R4, R8, R9      ; result = (f + g) - (h + i)
  MOV R0, R4          ; Τοποθέτηση επιστρεφόμενης
                      ; τιμής στον καταχωρητή R0
  LDR R9, [SP]        ; Ανάκτηση του R9 από τη στοίβα
  LDR R8, [SP, #-4]   ; Ανάκτηση του R8 από τη στοίβα
  LDR R4, [SP, #-8]   ; Ανάκτηση του R4 από τη στοίβα
  ADD SP, SP, #12     ; Αποδέσμευση του σχετικού χώρου
                      ; στη στοίβα
  MOV PC, LR          ; Επιστροφή στην καλούσα συνάρτηση
```

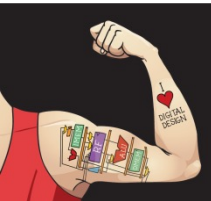


Η στοίβα κατά τη διάρκεια της κλήσης της `diffofsums`



Καταχωρητές

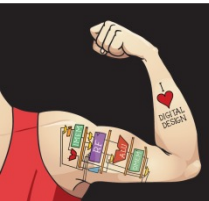
Διατηρούμενοι *Αποθηκευόμενοι από την καλούσα συνάρτηση*	**Μη διατηρούμενοι** *Αποθηκευόμενοι από την καλούμενη συνάρτηση*
R4-R11	**R12**
R14 (LR)	**R0-R3**
R13 (SP)	**CPSR**
Στοίβα πάνω από τον δείκτη SP	**Στοίβα κάτω από τον δείκτη SP**



Αποθήκευση περιεχομένων αποθηκευμένων καταχωρητών μόνο στη στοίβα

Κώδικας συμβολικής γλώσσας της ARM

```
; R2 = result
DIFFOFSUMS
  STR R4, [SP, #-4]! ; Αποθήκευση του R4 στη στοίβα
  ADD R8, R0, R1     ; R8 = f + g
  ADD R9, R2, R3     ; R9 = h + i
  SUB R4, R8, R9     ; result = (f + g) - (h + i)
  MOV R0, R4         ; Τοποθέτηση επιστρεφόμενης τιμής
                     ; στον R0
  LDR R4, [SP], #4   ; Ανάκτηση του R4 από τη στοίβα
  MOV PC, LR         ; Επιστροφή στην καλούσα συνάρτηση
```

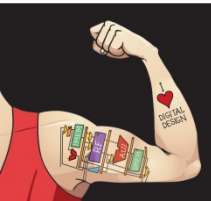


Αποθήκευση περιεχομένων αποθηκευμένων καταχωρητών μόνο στη στοίβα

Κώδικας συμβολικής γλώσσας της ARM

```
; R2 = result
DIFFOFSUMS
  STR R4, [SP, #-4]! ; Αποθήκευση του R4 στη στοίβα
  ADD R8, R0, R1     ; R8 = f + g
  ADD R9, R2, R3     ; R9 = h + i
  SUB R4, R8, R9     ; result = (f + g) - (h + i)
  MOV R0, R4         ; Τοποθέτηση επιστρεφόμενης τιμής
                     ; στον R0
  LDR R4, [SP], #4   ; Ανάκτηση του R4 από τη στοίβα
  MOV PC, LR         ; Επιστροφή στην καλούσα συνάρτηση
```

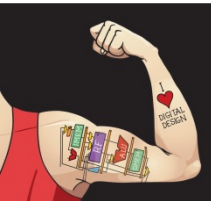
Παρατηρήστε τη βελτιστοποίηση του κώδικα για την επέκταση/συρρίκνωση της στοίβας



Μη τερματική συνάρτηση (ή συνάρτηση/φύλλο)

Κώδικας συμβολικής γλώσσας της ARM

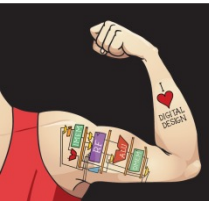
```
STR LR, [SP, #-4]!      ; Αποθήκευση του LR στη στοίβα
BL  PROC2               ; Κλήση άλλης συνάρτησης
...
LDR LR, [SP], #4        ; Ανάκτηση του LR από τη στοίβα
jr  $ra                 ; Επιστροφή στην καλούσα συνάρτηση
```



Μη τερματική συνάρτηση: Παράδειγμα

Κώδικας γλώσσας C

```
int f1(int a, int b) {
  int i, x;
  x = (a + b)*(a - b);
  for (i=0; i<a; i++)
    x = x + f2(b+i);
  return x;
}
int f2(int p) {
  int r;
  r = p + 5;
  return r + p;
}
```



Μη τερματική συνάρτηση: Παράδειγμα

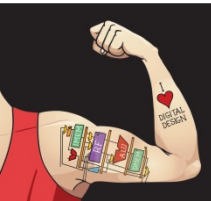
Κώδικας γλώσσας C

```
int f1(int a, int b) {
  int i, x;
  x = (a + b)*(a - b);
  for (i=0; i<a; i++)
    x = x + f2(b+i);
  return x;
}
int f2(int p) {
  int r;
  r = p + 5;
  return r + p;
}
```

Κώδικας συμβ/κής γλώσσας της ARM

```
; R0=a, R1=b, R4=i, R5=x
F1
  PUSH {R4,  R5, LR}
  ADD   R5,  R0, R1
  SUB   R12, R0, R1
  MUL   R5,  R5, R12
  MOV   R4,  #0
FOR
  CMP   R4, R0
  BGE   RETURN
  PUSH {R0, R1}
  ADD   R0, R1, R4
  BL    F2
  ADD   R5, R5, R0
  POP  {R0, R1}
  ADD   R4, R4, #1
  B     FOR
RETURN
  MOV   R0, R5
  POP  {R4, R5, LR}
  MOV   PC, LR
```

```
; R0=p, R4=r
F2
  PUSH {R4}
  ADD   R4, R0, 5
  ADD   R0, R4, R0
  POP  {R4}
  MOV   PC, LR
```

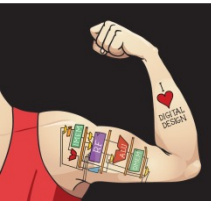


Μη τερματική συνάρτηση: Παράδειγμα

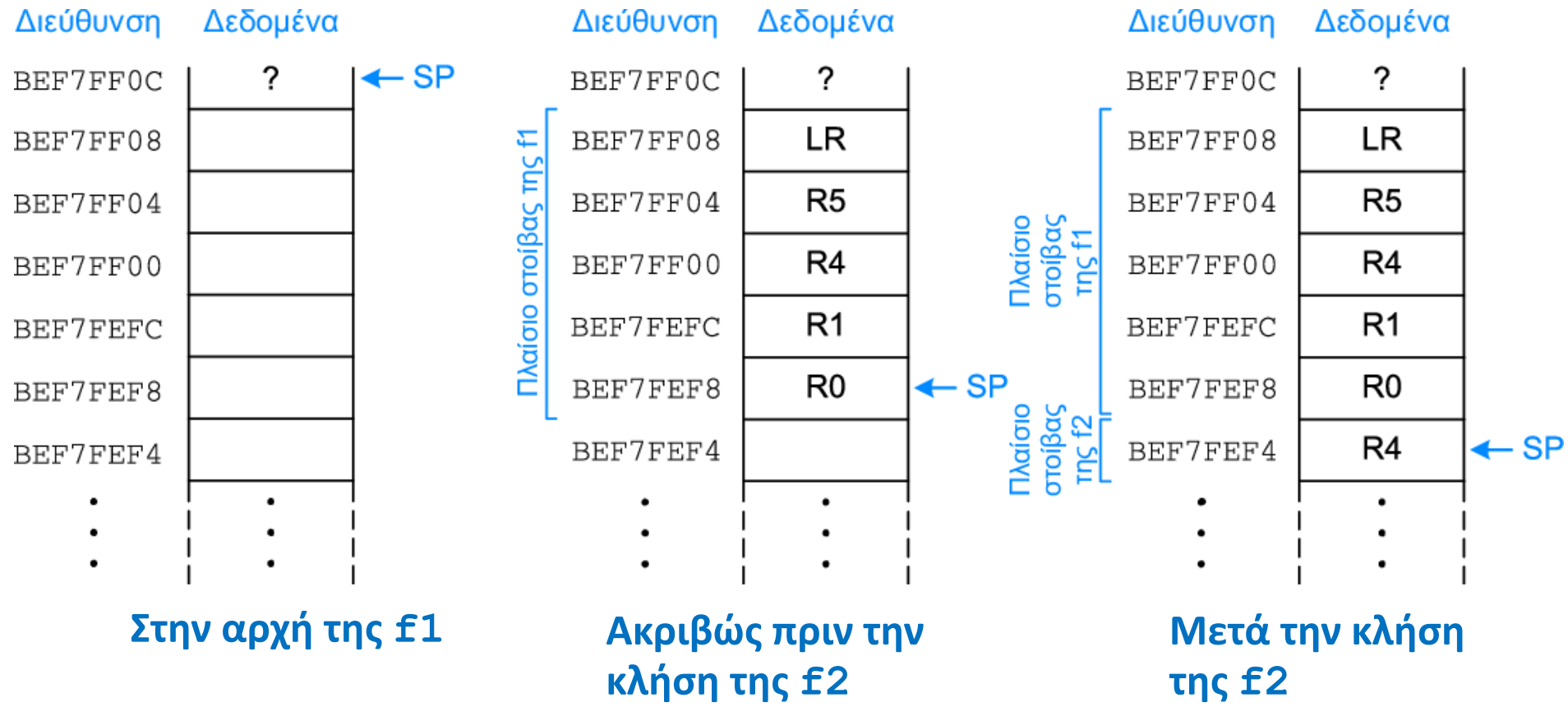
Κώδικας συμβολικής γλώσσας της ARM

```
; R0=a, R1=b, R4=i, R5=x
F1
  PUSH {R4,  R5, LR} ; Αποθήκευση
                     ; καταχωρητών
  ADD   R5,  R0, R1  ; x = (a+b)
  SUB   R12, R0, R1  ; temp = (a-b)
  MUL   R5,  R5, R12 ; x = x*temp
  MOV   R4,  #0      ; i = 0
FOR
  CMP   R4, R0       ; i < a?
  BGE   RETURN       ; Όχι: τερματισμός
                     ; βρόχου
  PUSH {R0, R1}      ; Αποθήκευση
                     ; καταχωρητών
  ADD   R0, R1, R4   ; Το όρισμα της f2 είναι b+i
  BL    F2           ; Κλήση f2(b+i)
  ADD   R5, R5, R0   ; x = x+f2(b+i)
  POP  {R0, R1}      ; Ανάκτηση καταχωρητών
                     ; από τη στοίβα
  ADD   R4, R4, #1   ; i++
  B     FOR          ; Επανάληψη βρόχου
RETURN
  MOV   R0, R5       ; Η επιστρεφόμενη τιμή της f1
                     ; είναι το x
  POP  {R4, R5, LR}  ; Ανάκτηση καταχωρητών
                     ; από τη στοίβα
  MOV   PC, LR       ; Επιστροφή
```

```
; R0=p, R4=r
F2
  PUSH {R4}          ; Αποθήκευση
                     ; καταχωρητών
  ADD   R4, R0, 5    ; r = p + 5
  ADD   R0, R4, R0   ; Επιστροφή
                     ; του r + p
  POP  {R4}          ; Ανάκτηση
                     ; καταχωρητών
  MOV   PC, LR       ; Επιστροφή
```



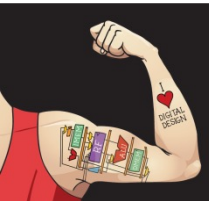
Η στοίβα κατά τη διάρκεια μιας μη τερματικής συνάρτησης



Κλήση αναδρομικής συνάρτησης

Κώδικας γλώσσας C

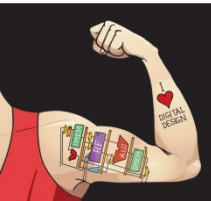
```
int factorial(int n) {
  if (n <= 1)
    return 1;
  else
    return (n * factorial(n-1));
}
```



Κλήση αναδρομικής συνάρτησης

Κώδικας συμβολικής γλώσσας της ARM

```
0x94  FACTORIAL   STR R0, [SP, #-4]!    ; Αποθήκευση του R0 στη στοίβα
0x98              STR LR, [SP, #-4]!    ; Αποθήκευση του LR στη στοίβα
0x9C              CMP R0, #2            ; Ενεργοποίηση των σημαιών με
                                        ; βάση το R0 - 2
0xA0              BHS ELSE              ; if (r0>=2) διακλάδωση
                                        ; στο else
0xA4              MOV R0, #1            ; Διαφορετικά, επιστροφή του 1
0xA8              ADD SP, SP, #8        ; Επαναφορά του SP
0xAC              MOV PC, LR            ; Επιστροφή
0xB0  ELSE        SUB R0, R0, #1        ; n = n - 1
0xB4              BL  FACTORIAL         ; Αναδρομική κλήση
0xB8              LDR LR, [SP], #4      ; Ανάκτηση του LR
0xBC              LDR R1, [SP], #4      ; Ανάκτηση και τοποθέτηση του
                                        ; R0 (n) στον R1
0xC0              MUL R0, R1, R0        ; R0 = n*factorial(n-1)
0xC4              MOV PC, LR            ; Επιστροφή
```



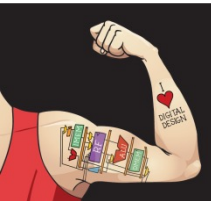
Κλήση αναδρομικής συνάρτησης

Κώδικας γλώσσας C

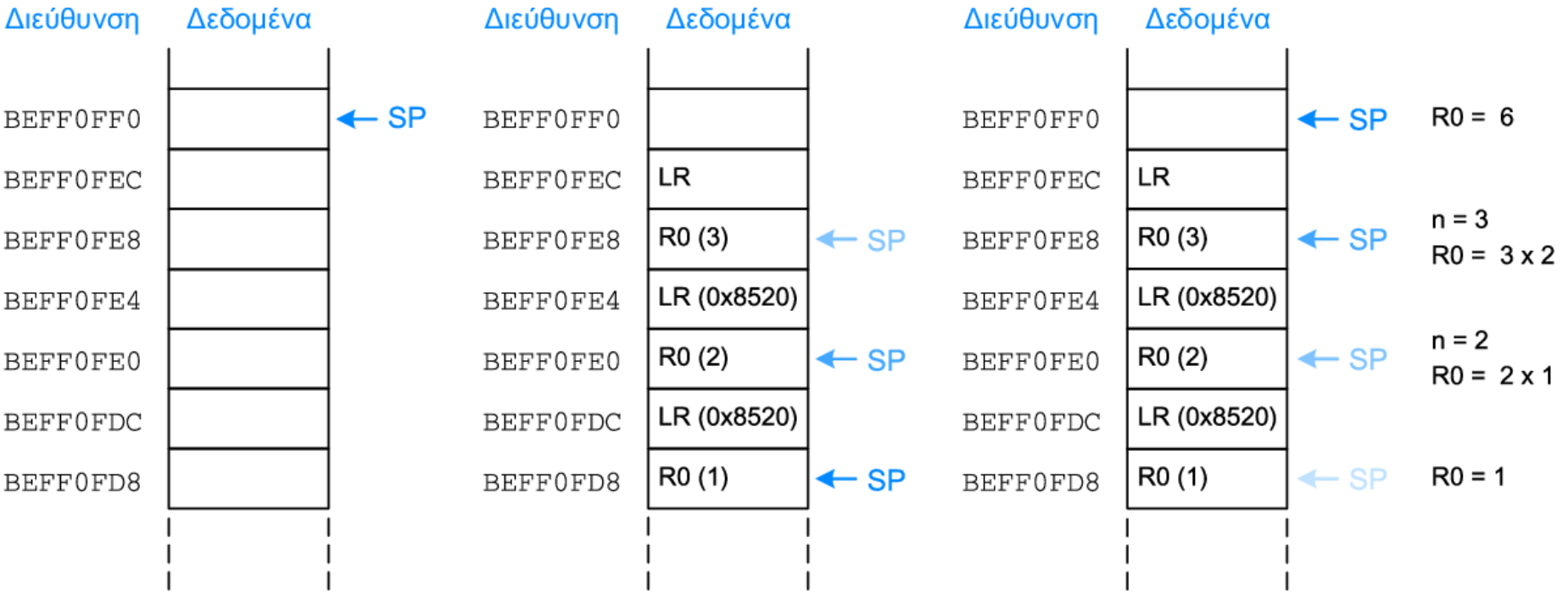
```
int factorial(int n) {

  if (n <= 1)
    return 1;



  else
    return (n * factorial(n-1));
}
```

Κώδικας συμβολικής γλώσσας της ARM

```
0x94 FACTORIAL  STR R0, [SP, #-4]!
0x98            STR LR, [SP, #-4]!
0x9C            CMP R0, #2
0xA0            BHS ELSE
0xA4            MOV R0, #1
0xA8            ADD SP, SP, #8
0xAC            MOV PC, LR
0xB0 ELSE       SUB R0, R0, #1
0xB4            BL  FACTORIAL
0xB8            LDR LR, [SP], #4
0xBC            LDR R1, [SP], #4
0xC0            MUL R0, R1, R0
0xC4            MOV PC, LR
```



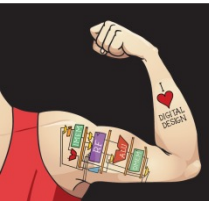
Η στοίβα κατά τη διάρκεια μιας αναδρομικής κλήσης



Πριν την κλήση

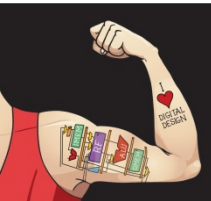
Κατά τη διάρκεια της κλήσης

Μετά την κλήση

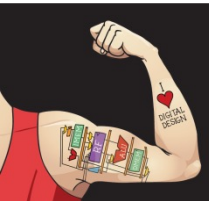


Κλήσεις συναρτήσεων: Σύνοψη

- **Καλούσα συνάρτηση**
  - Τοποθετεί ορίσματα στους `R0-R3`
  - Αποθηκεύει περιεχόμενα τυχόν απαιτούμενων καταχωρητών (`LR`, ενδεχομένως `R0-R3, R8-R12`)
  - Καλεί τη συνάρτηση: `BL CALLEE`
  - Ανακτά τους καταχωρητές
  - Ψάχνει για το αποτέλεσμα στον `R0`
- **Καλούμενη συνάρτηση**
  - Αποθηκεύει καταχωρητές που μπορεί να «πειραχτούν» (`R4-R7`)
  - Εκτελεί τις εντολές της συνάρτησης
  - Τοποθετεί το αποτέλεσμα στον `R0`
  - Ανακτά τους καταχωρητές
  - Επιστρέφει: `MOV PC, LR`

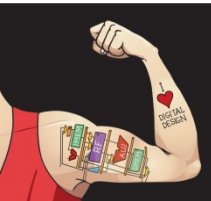


Πώς κωδικοποιούμε εντολές;



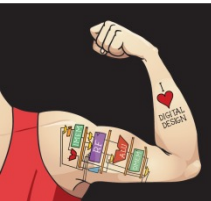
Πώς κωδικοποιούμε εντολές;

- **Σχεδιαστική αρχή 1: Η κανονικότητα ενισχύει την απλότητα στη σχεδίαση**
  - Δεδομένα των 32 bit, εντολές των 32 bit
  - Για λόγους απλότητας στη σχεδίαση, θα προτιμούσαμε μία μόνο μορφή εντολών αλλά…



Πώς κωδικοποιούμε εντολές;

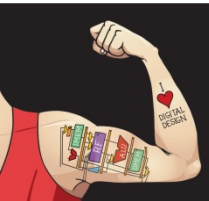
- **Σχεδιαστική αρχή 1: Η κανονικότητα ενισχύει την απλότητα στη σχεδίαση**
  - Δεδομένα των 32 bit, εντολές των 32 bit
  - Για λόγους απλότητας στη σχεδίαση, θα προτιμούσαμε μία μόνο μορφή εντολών αλλά…
  - …οι εντολές έχουν διαφορετικές ανάγκες



Σχεδιαστική αρχή 4

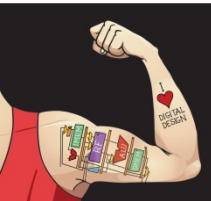
Η καλή σχεδίαση απαιτεί και καλούς συμβιβασμούς

- Οι πολλές μορφές εντολών επιτρέπουν ευελιξία
  - `ADD`, `SUB`: Χρησιμοποιούν 3 καταχωρητές-τελεστέους
  - `LDR`, `STR`: Χρησιμοποιούν 2 καταχωρητές-τελεστέους και μια σταθερά (άμεσο τελεστέο)
- Το πλήθος των μορφών εντολών διατηρείται μικρό
  - Συμμόρφωση με τις σχεδιαστικές αρχές 1 και 3 (Η κανονικότητα ενισχύει την απλότητα στη σχεδίαση και Το μικρότερο μέγεθος συνεπάγεται πιο γρήγορη εκτέλεση)



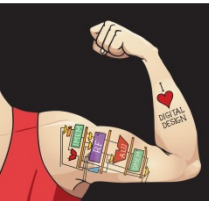
Γλώσσα μηχανής

- **Δυαδική αναπαράσταση των εντολών**
- Οι υπολογιστές κατανοούν μόνο τις τιμές **1 και 0**
- **Εντολές των 32 bit**
  - Η απλότητα ευνοεί την κανονικότητα: Δεδομένα και εντολές των 32 bit
- **3 μορφές εντολών:**
  - Επεξεργασίας δεδομένων
  - Μνήμης
  - Διακλάδωσης



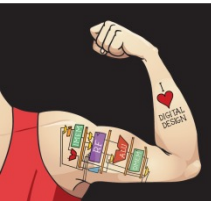
Μορφές εντολών

- **Επεξεργασίας δεδομένων**
- Μνήμης
- Διακλάδωσης



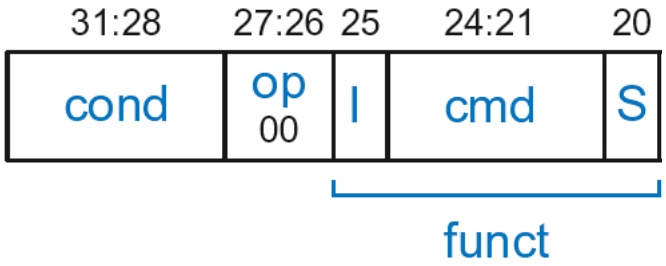
Μορφή εντολής επεξεργασίας δεδομένων

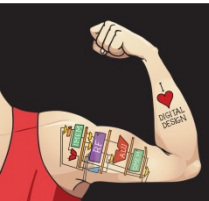
- **Τελεστέοι:**
  - ***Rn***: 1ος καταχωρητής προέλευσης
  - ***Src2***: 2ος καταχωρητής (ή άμεσος τελεστέος) προέλευσης
  - ***Rd***: Καταχωρητής προορισμού
- **Πεδία ελέγχου:**
  - ***cond***: Καθορίζει την εκτέλεση υπό συνθήκη
  - ***op***: *operation code* (κωδικός πράξης) ή *opcode*
  - ***funct***: η *λειτουργία*/πράξη προς εκτέλεση



Πεδία ελέγχου επεξεργασίας δεδομένων

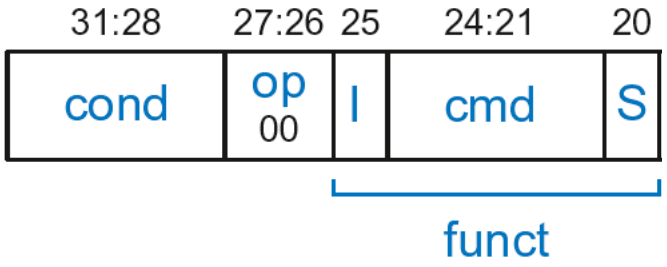
- ***op*** **= 00_2** για εντολές επεξεργασίας δεδομένων (data-processing, DP)
- Το πεδίο ***funct*** αποτελείται από τα υποπεδία ***cmd***, ***I*** και ***S***

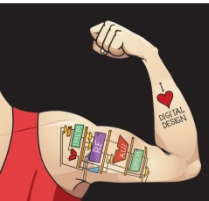




Πεδία ελέγχου επεξεργασίας δεδομένων

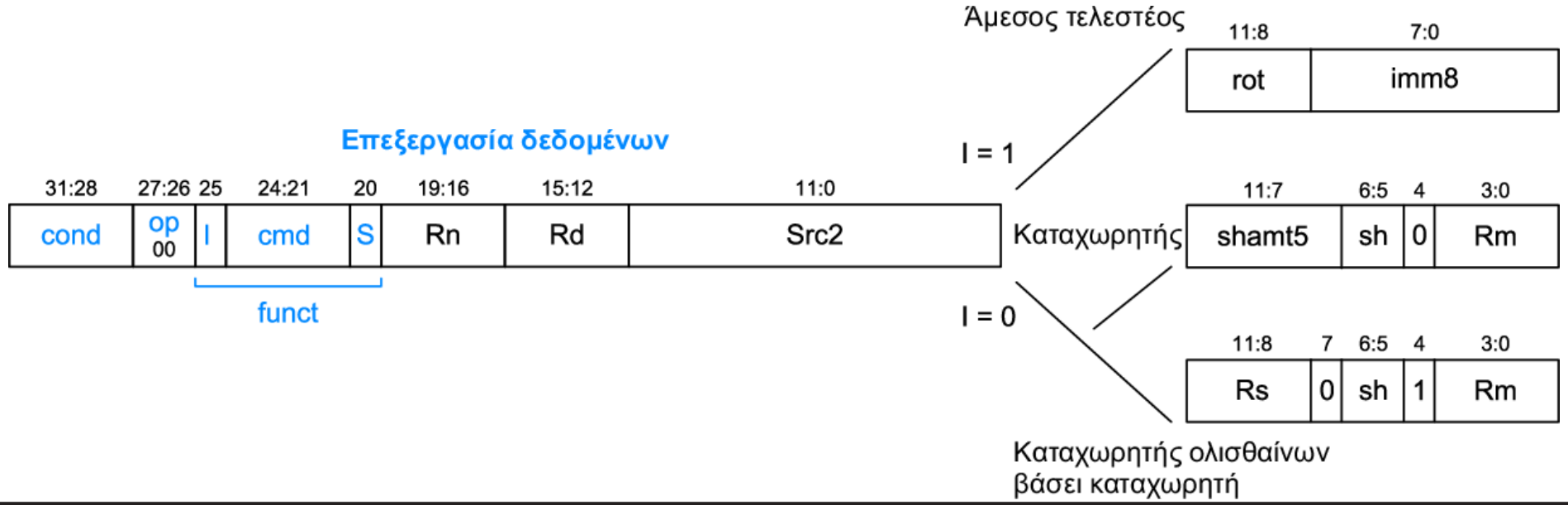
- ***op*** **= 00_2** για εντολές επεξεργασίας δεδομένων (data-processing, DP)
- Το πεδίο ***funct*** αποτελείται από τα υποπεδία ***cmd***, ***I*** και ***S***
  - ***cmd:*** Καθορίζει τη συγκεκριμένη εντολή επεξεργασίας δεδομένων. Π.χ.
    - ***cmd*** = 0100_2 για την `ADD`
    - ***cmd*** = 0010_2 για τη `SUB`
  - ***I*** (bit)
    - ***I*** = 0: Το *Src2* είναι καταχωρητής
    - ***I*** = 1: Το *Src2* είναι άμεσος τελεστέος
  - ***S*** (bit): 1 αν η εντολή ενεργοποιεί σημαίες συνθήκης
    - ***S*** = 0: `SUB  R0, R5, R7`
    - ***S*** = 1: `ADDS R8, R2, R4` ή `CMP R3, #10`

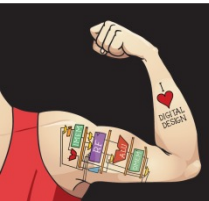




Εντολές επεξεργασίας δεδομένων: *Src2*

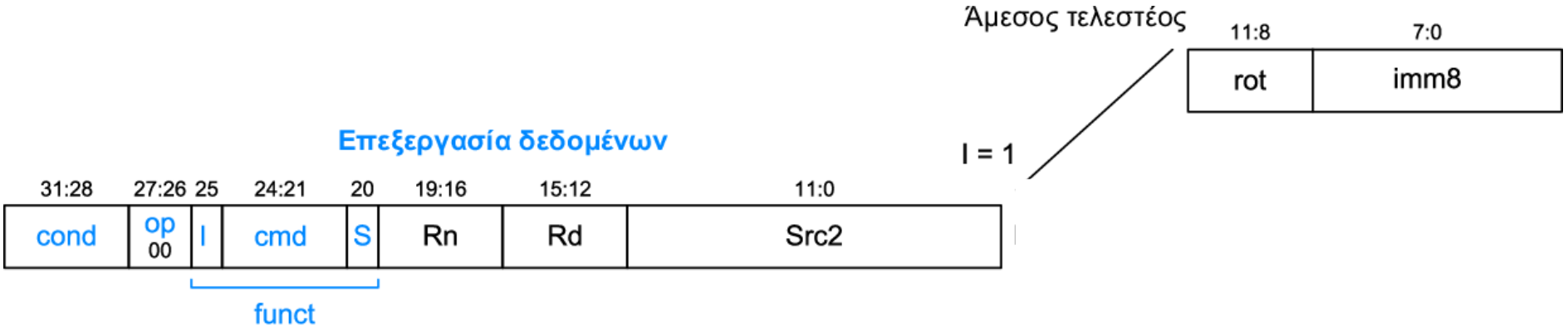
- Το *Src2* μπορεί να είναι:
  - Άμεσος τελεστέος
  - Καταχωρητής
  - Καταχωρητής ολισθαίνων βάσει καταχωρητή

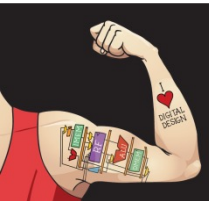




Εντολές επεξεργασίας δεδομένων: *Src2*

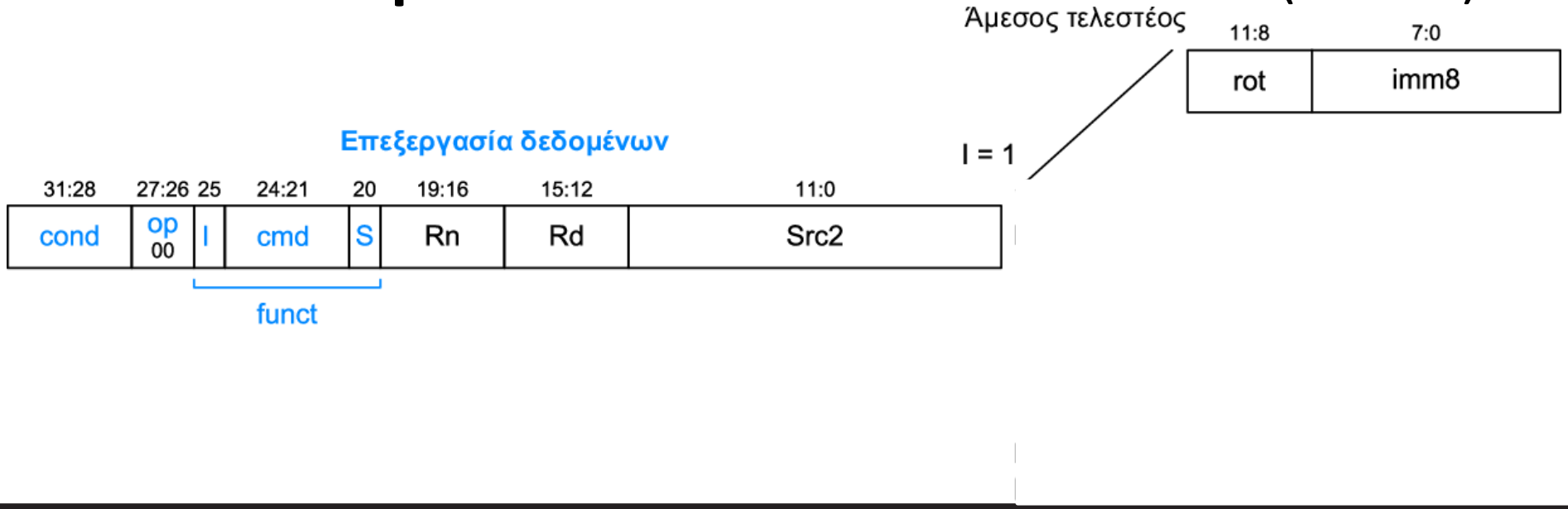
- Το *Src2* μπορεί να είναι:
  - **Άμεσος τελεστέος**
  - Καταχωρητής
  - Καταχωρητής ολισθαίνων βάσει καταχωρητή

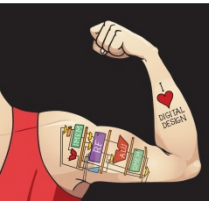




Το *Src2* ως άμεσος τελεστέος

- **Ο άμεσος τελεστέος κωδικοποιείται ως:**
  - *imm8*: μη προσημασμένος άμεσος τελεστέος των 8 bit
  - *rot*: τομή περιστροφής (4 bit)
- **Η σταθερά των 32 bit είναι:** *imm8* **ROR** (*rot* × 2)

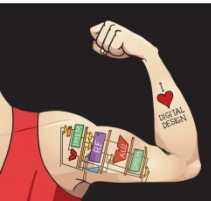




Το *Src2* ως άμεσος τελεστέος

- **Ο άμεσος τελεστέος κωδικοποιείται ως:**
  - *imm8*: μη προσημασμένος άμεσος τελεστέος των 8 bit
  - *rot*: τομή περιστροφής (4 bit)
- **Η σταθερά των 32 bit είναι:** *imm8* **ROR** (*rot* × 2)
- **Παράδειγμα:** *imm8* = abcdefgh

rot	Σταθερά (32 bit)
0000	0000 0000 0000 0000 0000 0000 abcd efgh
0001	gh00 0000 0000 0000 0000 0000 00ab cdef
…	…
1111	0000 0000 0000 0000 0000 00ab cdef gh00



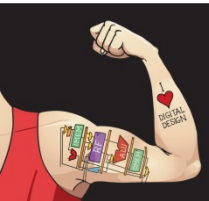
Το *Src2* ως άμεσος τελεστέος

- **Ο άμεσος τελεστέος κωδικοποιείται ως:**
  - *imm8*: μη προσημασμένος άμεσος τελεστέος των 8 bit
  - *rot*: τομή περιστροφής (4 bit)

ROR κατά X = ROL κατά (32–X)
Παρ: ROR κατά 30 = ROL κατά 2

- **Η σταθερά των 32 bit είναι:** *imm8* **ROR** (*rot* × 2)
- **Παράδειγμα:** *imm8* = abcdefgh

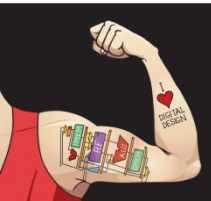
rot	Σταθερά (32 bit)
0000	0000 0000 0000 0000 0000 0000 abcd efgh
0001	gh00 0000 0000 0000 0000 0000 00ab cdef
…	…
1111	0000 0000 0000 0000 0000 00ab cdef gh00



Εντολή DP με άμεσο τελεστέο *Src2*

```
ADD R0, R1, #42
```

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0100_2 (4) για την εντολή ADD
- Το ***Src2*** είναι άμεσος τελεστέος, άρα ***I*** = 1
- ***Rd*** = 0, ***Rn*** = 1
- ***imm8*** = 42, ***rot*** = 0



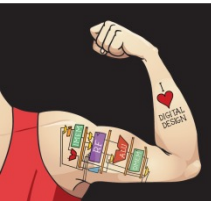
Εντολή DP με άμεσο τελεστέο *Src2*

```
ADD R0, R1, #42
```

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0100_2 (4) για την εντολή ADD
- Το ***Src2*** είναι άμεσος τελεστέος, άρα ***I*** = 1
- ***Rd*** = 0, ***Rn*** = 1
- ***imm8*** = 42, ***rot*** = 0

Τιμές πεδίων

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0100_2	0	1	0	0	42
cond	op	I	cmd	S	Rn	Rd	shamt5	sh Rm
1110	00	1	0100	0	0001	0000	0000	00101010



Εντολή DP με άμεσο τελεστέο *Src2*

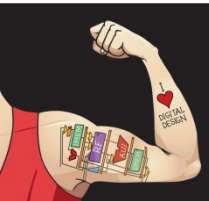
```
ADD R0, R1, #42
```

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0100_2 (4) για την εντολή ADD
- Το ***Src2*** είναι άμεσος τελεστέος, άρα ***I*** = 1
- ***Rd*** = 0, ***Rn*** = 1
- ***imm8*** = 42, ***rot*** = 0

Τιμές πεδίων

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0100_2	0	1	0	0	42
cond	op	I	cmd	S	Rn	Rd	shamt5	sh Rm
1110	00	1	0100	0	0001	0000	0000	00101010

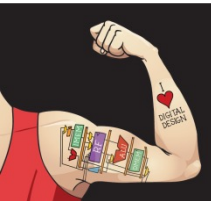
0xE281002A



Εντολή DP με άμεσο τελεστέο *Src2*

```
SUB R2, R3, #0xFF0
```

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0010_2 (2) για την εντολή SUB
- Το ***Src2*** είναι άμεσος τελεστέος, άρα ***I*** = 1
- ***Rd*** = 2, ***Rn*** = 3
- ***imm8*** = 0xFF
- Το ***imm8*** πρέπει να περιστραφεί προς τα δεξιά κατά 28 για να παραχθεί η τιμή 0xFF0, άρα ***rot*** = 14

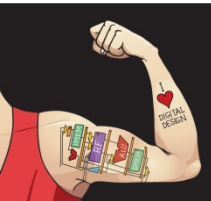


Εντολή DP με άμεσο τελεστέο *Src2*

```
SUB R2, R3, #0xFF0
```

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0010_2 (2) για την εντολή SUB
- Το ***Src2*** είναι άμεσος τελεστέος, άρα ***I*** = 1
- ***Rd*** = 2, ***Rn*** = 3
- ***imm8*** = 0xFF
- Το ***imm8*** πρέπει να περιστραφεί προς τα δεξιά κατά 28 για να παραχθεί η τιμή 0xFF0, άρα ***rot*** = 14

ROR κατά 28 =
ROL κατά (32–28) = 4



Εντολή DP με άμεσο τελεστέο *Src2*

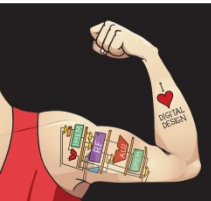
```
SUB R2, R3, #0xFF0
```

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0010_2 (2) για την εντολή SUB
- Το ***Src2*** είναι άμεσος τελεστέος, άρα ***I*** = 1
- ***Rd*** = 2, ***Rn*** = 3
- ***imm8*** = 0xFF
- Το ***imm8*** πρέπει να περιστραφεί προς τα δεξιά κατά 28 για να παραχθεί η τιμή 0xFF0, άρα ***rot*** = 14

ROR κατά 28 =
ROL κατά (32–28) = 4

Τιμές πεδίων

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0010_2	0	3	2	14	255
cond	op	I	cmd	S	Rn	Rd	rot	imm8
1110	00	1	0010	0	0011	0010	1110	11111111



Εντολή DP με άμεσο τελεστέο *Src2*

```
SUB R2, R3, #0xFF0
```

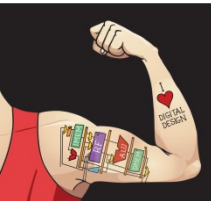
- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0010_2 (2) για την εντολή SUB
- Το ***Src2*** είναι άμεσος τελεστέος, άρα ***I*** = 1
- ***Rd*** = 2, ***Rn*** = 3
- ***imm8*** = 0xFF
- Το ***imm8*** πρέπει να περιστραφεί προς τα δεξιά κατά 28 για να παραχθεί η τιμή 0xFF0, άρα ***rot*** = 14

ROR κατά 28 =
ROL κατά (32–28) = 4

Τιμές πεδίων

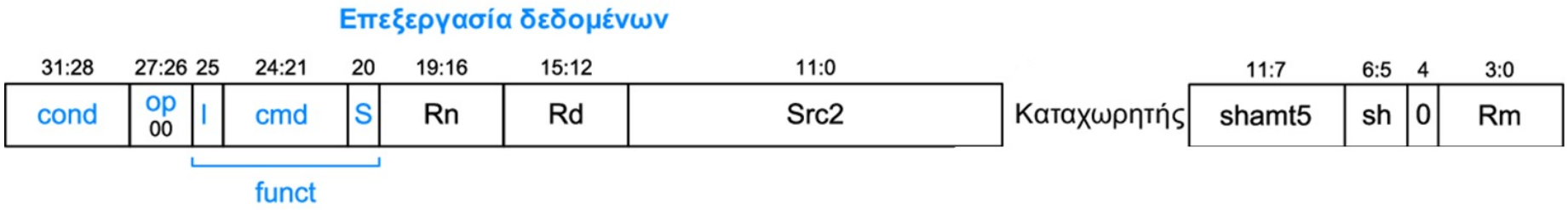
31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0010_2	0	3	2	14	255
cond	op	I	cmd	S	Rn	Rd	rot	imm8
1110	00	1	0010	0	0011	0010	1110	11111111

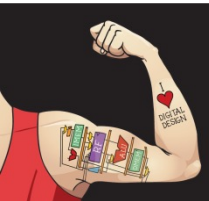
0xE2432EFF



Εντολή DP με καταχωρητή *Src2*

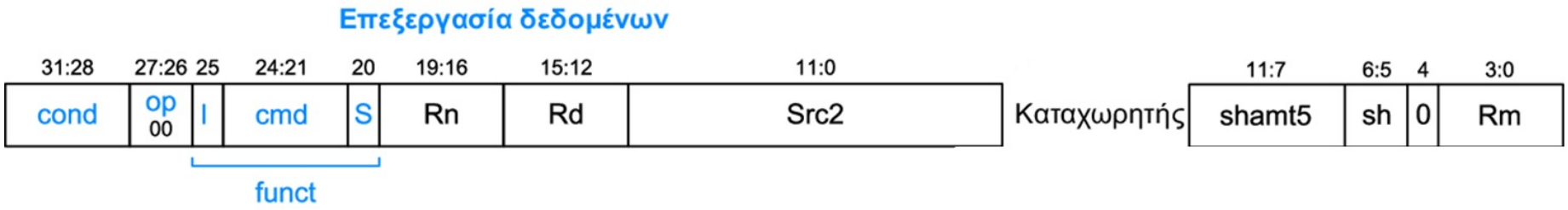
- Το *Src2* μπορεί να είναι:
  - Άμεσος τελεστέος
  - **Καταχωρητής**
  - Καταχωρητής ολισθαίνων βάσει καταχωρητή

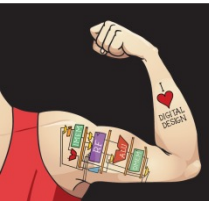




Εντολή DP με καταχωρητή *Src2*

- ***Rm***: Ο δεύτερος τελεστέος προέλευσης
- ***shamt5***: Η ποσότητα ολίσθησης του Rm
- ***sh***: Ο τύπος ολίσθησης (>>, <<, >>>, ROR)

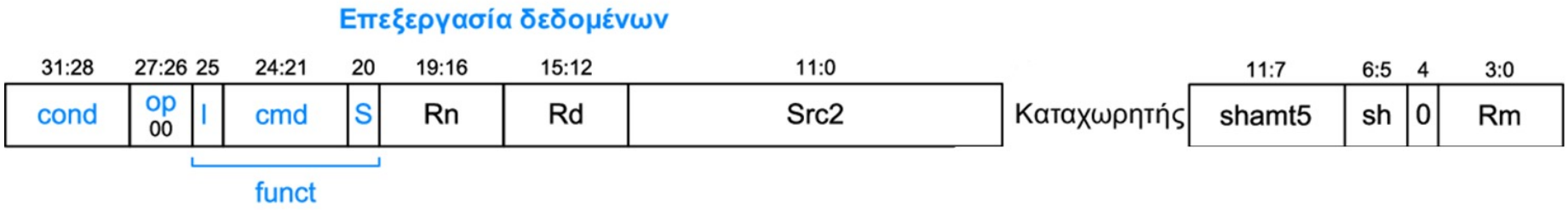


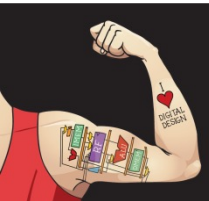


Εντολή DP με καταχωρητή *Src2*

- ***Rm***: Ο δεύτερος τελεστέος προέλευσης
- ***shamt5***: Η ποσότητα ολίσθησης του Rm
- ***sh***: Ο τύπος ολίσθησης (>>, <<, >>>, ROR)

Πρώτα θα εξετάσουμε μη ολισθαίνουσες εκδοχές του *Rm*

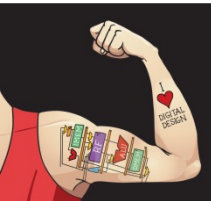




Εντολή DP με καταχωρητή *Src2*

`ADD R5, R6, R7`

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0100_2 (4) για την εντολή `ADD`
- Το ***Src2*** είναι καταχωρητής, άρα ***I*** = 0
- ***Rd*** = 5, ***Rn*** = 6, ***Rm*** = 7
- ***shamt*** = 0, ***sh*** = 0



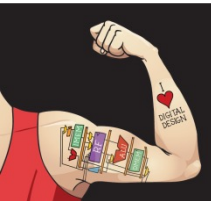
Εντολή DP με καταχωρητή *Src2*

`ADD R5, R6, R7`

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0100_2 (4) για την εντολή `ADD`
- Το ***Src2*** είναι καταχωρητής, άρα ***I*** = 0
- ***Rd*** = 5, ***Rn*** = 6, ***Rm*** = 7
- ***shamt*** = 0, ***sh*** = 0

Τιμές πεδίων

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110_2	00_2	0	0100_2	0	6	5	0	0	0	7
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm
1110	00	0	0100	0	0110	0101	00000	00	0	0111



Εντολή DP με καταχωρητή *Src2*

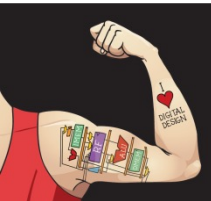
`ADD R5, R6, R7`

- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0100_2 (4) για την εντολή `ADD`
- Το ***Src2*** είναι καταχωρητής, άρα ***I*** = 0
- ***Rd*** = 5, ***Rn*** = 6, ***Rm*** = 7
- ***shamt*** = 0, ***sh*** = 0

Τιμές πεδίων

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110_2	00_2	0	0100_2	0	6	5	0	0	0	7
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm
1110	00	0	0100	0	0110	0101	00000	00	0	0111

0xE0865007

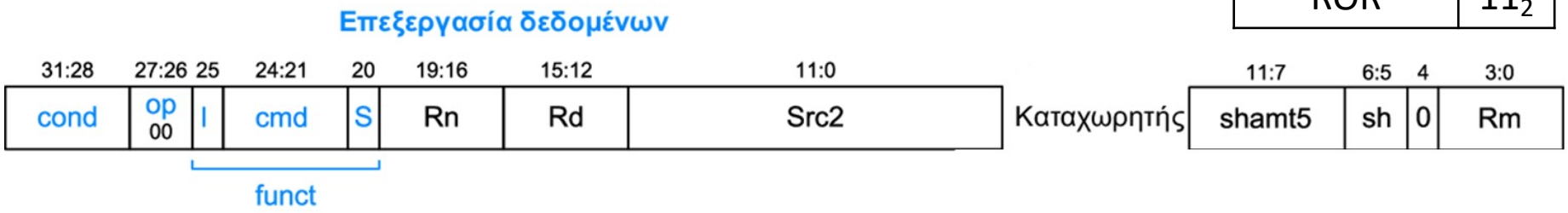


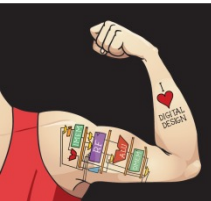
Εντολή DP με καταχωρητή *Src2*

- ***Rm***: Ο δεύτερος τελεστέος προέλευσης
- ***shamt5***: Η ποσότητα ολίσθησης του Rm
- ***sh***: Ο τύπος ολίσθησης

Τύπος ολίσθησης	*sh*
LSL	00_2
LSR	01_2
ASR	10_2
ROR	11_2

Τώρα θα εξετάσουμε τις ολισθαίνουσες εκδοχές



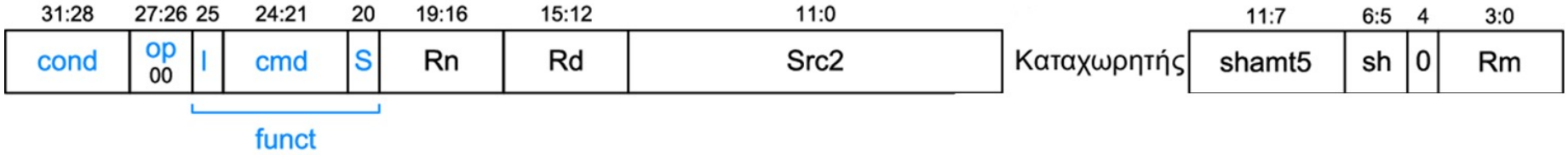


Εντολή DP με καταχωρητή *Src2*

`ORR R9, R5, R3,` **`LSR #2`**

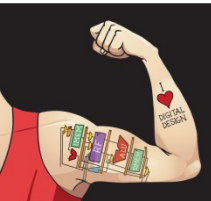
- **Πράξη:** R9 = R5 OR (R3 >> 2)
- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 1100_2 (12) για την εντολή `ORR`
- Το ***Src2*** είναι καταχωρητής, άρα ***I*** = 0
- ***Rd*** = 9, ***Rn*** = 5, ***Rm*** = 3
- ***shamt5*** = 2, ***sh*** = 01_2 (LSR)

Επεξεργασία δεδομένων



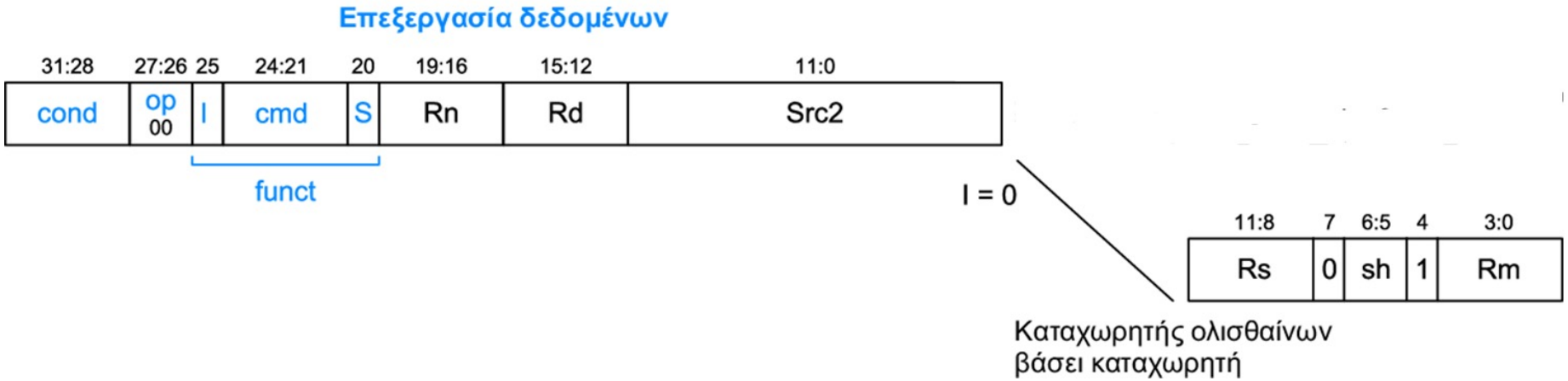
1110 00 0 1100 0 0101 1001 00010 01 0 0011

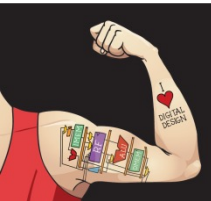
0xE1859123



Εντολή DP με καταχωρητή ολισθαίνων βάσει καταχωρητή (*Src2*)

- Το *Src2* μπορεί να είναι:
  - Άμεσος τελεστέος
  - Καταχωρητής
  - **Καταχωρητής ολισθαίνων βάσει καταχωρητή**

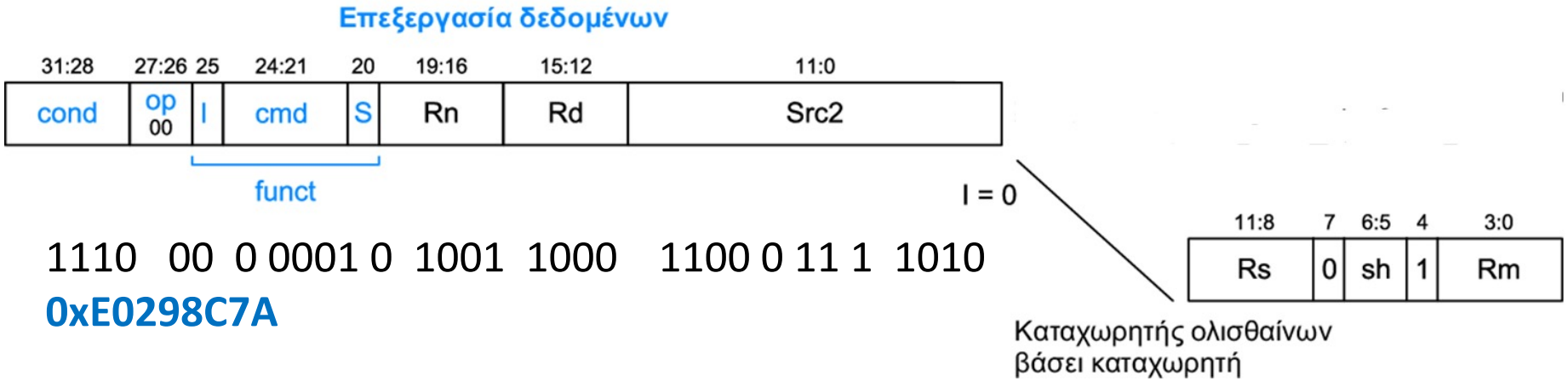




Εντολή DP με καταχωρητή ολισθαίνων βάσει καταχωρητή (*Src2*)

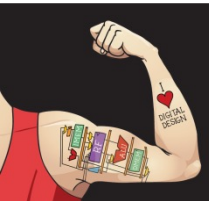
```
EOR R8, R9, R10, ROR R12
```

- **Πράξη:** R8 = R9 XOR (R10 ROR R12)
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 0001_2 (1) για την εντολή `EOR`
- Το ***Src2*** είναι καταχωρητής, άρα ***I*** = 0
- ***Rd*** = 8, ***Rn*** = 9, ***Rm*** = 10, ***Rs*** = 12
- ***sh*** = 11_2 (ROR)



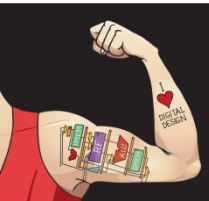
1110 00 0 0001 0 1001 1000 1100 0 11 1 1010

0xE0298C7A



Κωδικοποίηση εντολών ολίσθησης

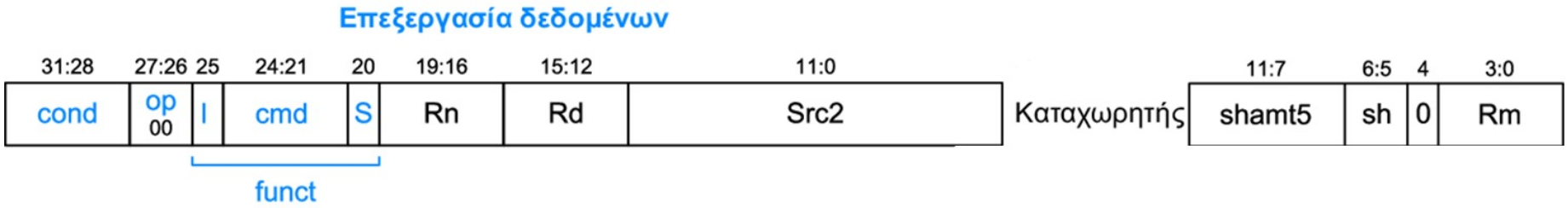
Τύπος ολίσθησης	*sh*
LSL	00_2
LSR	01_2
ASR	10_2
ROR	11_2



Εντολές ολίσθησης: Άμεσος τελεστέος *shamt*

```
ROR R1, R2, #23
```

- **Πράξη:** R1 = R2 ROR 23
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 1101_2 (13) για όλες τις ολισθήσεις (LSL, LSR, ASR και ROR)
- Το ***Src2*** είναι καταχωρητής ολισθαίνων βάσει άμεσου τελεστέου, άρα ***I*** = 0
- ***Rd*** = 1, ***Rn*** = 0, ***Rm*** = 2
- ***shamt5*** = 23, ***sh*** = 11_2 (ROR)



1110 00 0 1101 0 0000 0001 10111 11 0 0010

0xE1A01BE2

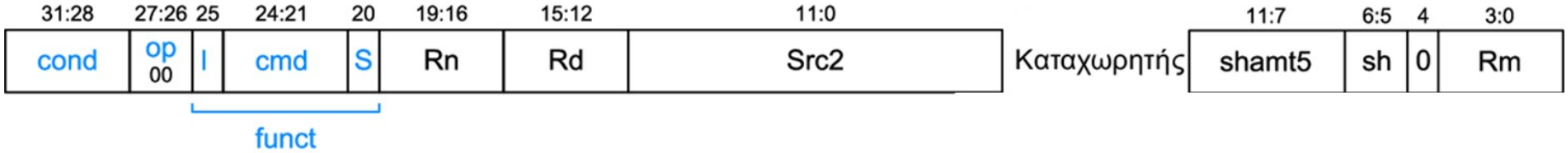
Εντολές ολίσθησης: Άμεσος τελεστέος *shamt*

```
ROR R1, R2, #23
```

- **Πράξη:** R1 = R2 ROR 23
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 1101_2 (13) για όλες τις ολισθήσεις (LSL, LSR, ASR και ROR)
- Το ***Src2*** είναι καταχωρητής ολισθαίνων βάσει άμεσου τελεστέου, άρα ***I*** = 0
- ***Rd*** = 1, ***Rn*** = 0, ***Rm*** = 2
- ***shamt5*** = 23, ***sh*** = 11_2 (ROR)

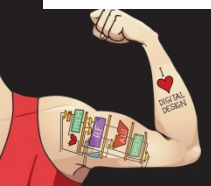
Χρησιμοποιεί κωδικοποίηση καταχωρητή (ολισθαίνοντος βάσει άμεσου τελεστέου) για το *Src2*

Επεξεργασία δεδομένων



1110 00 0 1101 0 0000 0001 10111 11 0 0010

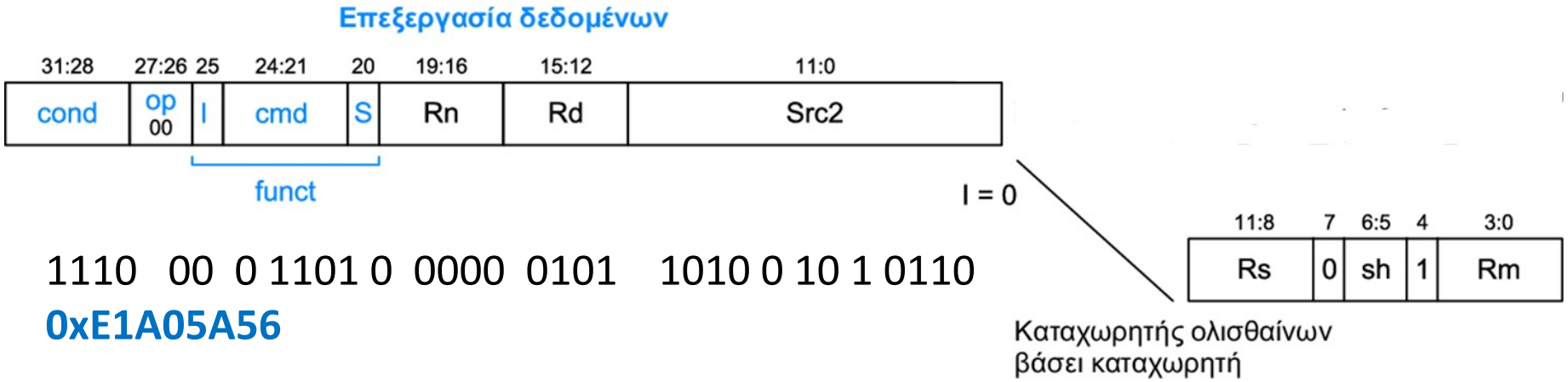
0xE1A01BE2



Εντολές ολίσθησης: Καταχωρητής *shamt*

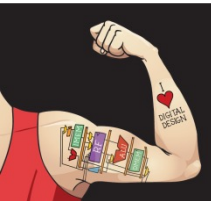
```
ASR R5, R6, R10
```

- **Πράξη:** R5 = R6 >>> $R10_{7:0}$
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 1101_2 (13) για όλες τις ολισθήσεις (LSL, LSR, ASR, and ROR)
- Το ***Src2*** είναι καταχωρητής, άρα ***I*** = 0
- ***Rd*** = 5, ***Rn*** = 0, ***Rm*** = 6, ***Rs*** = 10
- ***sh*** = 10_2 (ASR)



1110 00 0 1101 0 0000 0101 1010 0 10 1 0110

0xE1A05A56



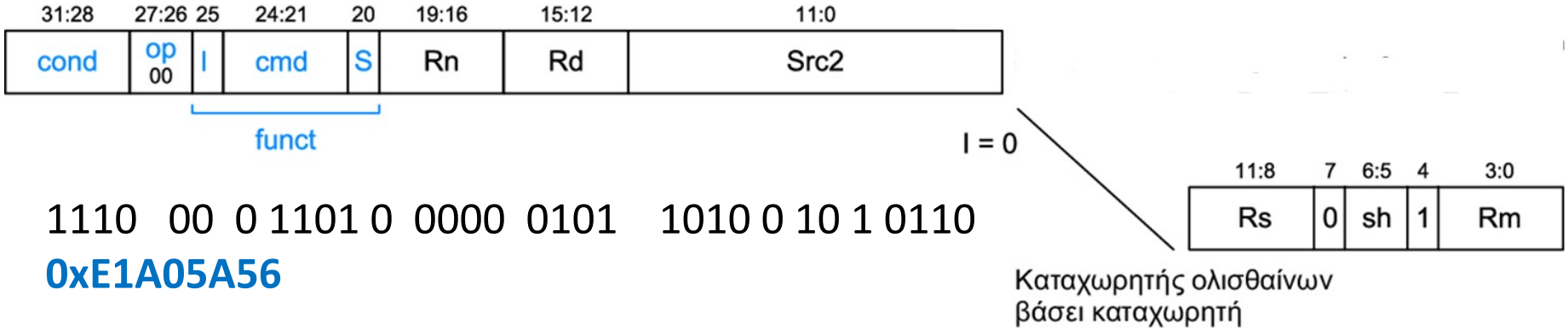
Εντολές ολίσθησης: Καταχωρητής shamt

```
ASR R5, R6, R10
```

- **Πράξη:** R5 = R6 >>> $R10_{7:0}$
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 00_2 (0) για εντολές επεξεργασίας δεδομένων
- ***cmd*** = 1101_2 (13) για όλες τις ολισθήσεις (LSL, LSR, ASR, and ROR)
- Το ***Src2*** είναι καταχωρητής, άρα ***I*** = 0
- ***Rd*** = 5, ***Rn*** = 0, ***Rm*** = 6, ***Rs*** = 10
- ***sh*** = 10_2 (ASR)

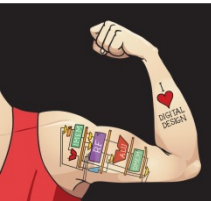
Χρησιμοποιεί κωδικοποίηση καταχωρητή ολισθαίνοντος βάσει καταχωρητή για το *Src2*

Επεξεργασία δεδομένων



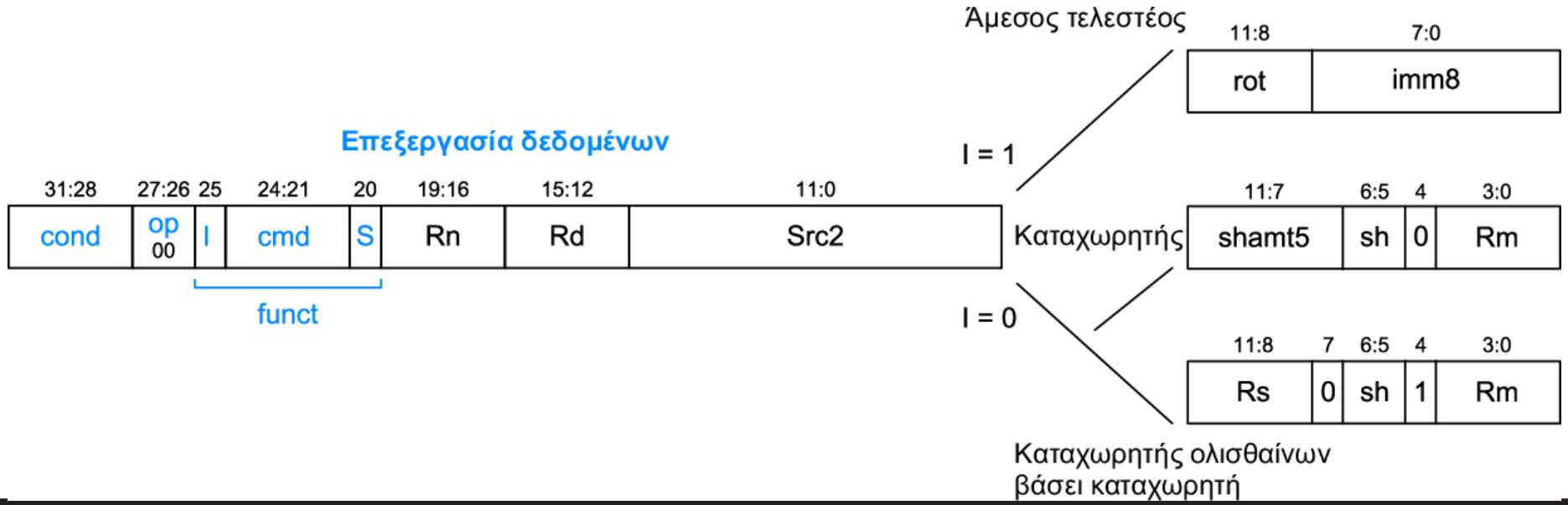
1110 00 0 1101 0 0000 0101 1010 0 10 1 0110

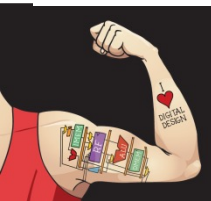
0xE1A05A56



Επανάληψη: Μορφή επεξεργασίας δεδομένων

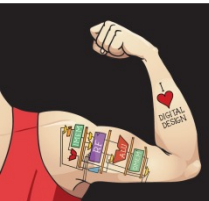
- Το *Src2* μπορεί να είναι:
  - Άμεσος τελεστέος
  - Καταχωρητής
  - Καταχωρητής ολισθαίνων βάσει καταχωρητή





Μορφές εντολών

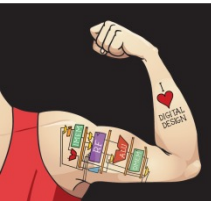
- Επεξεργασίας δεδομένων
- **Μνήμης**
- Διακλάδωσης



Μορφή εντολής μνήμης

Κωδικοποιεί: `LDR`, `STR`, `LDRB`, `STRB`

- ***op*** = 01_2
- ***Rn*** = καταχωρητής βάσης
- ***Rd*** = προορισμός (φόρτωση), προέλευση (αποθήκευση)
- ***Src2*** = σχετική απόσταση
- ***funct*** = 6 bit ελέγχου



Σχετική απόσταση: Επιλογές

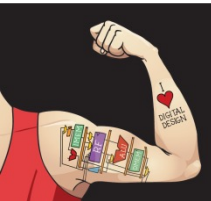
Θυμηθείτε: διεύθυνση = διεύθυνση βάσης + σχετική απόσταση

Παράδειγμα: `LDR R1, [R2, #4]`

Διεύθυνση βάσης = R2, σχετική απόσταση = 4

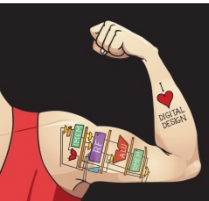
Διεύθυνση = (R2 + 4)

- Η διεύθυνση βάσης είναι αποθηκευμένη πάντα σε καταχωρητή
- Η σχετική απόσταση μπορεί να είναι:
  - άμεσος τελεστέος
  - καταχωρητής ή
  - προσαρμοσμένος (ολισθαίνων) καταχωρητής



Παραδείγματα σχετικής απόστασης

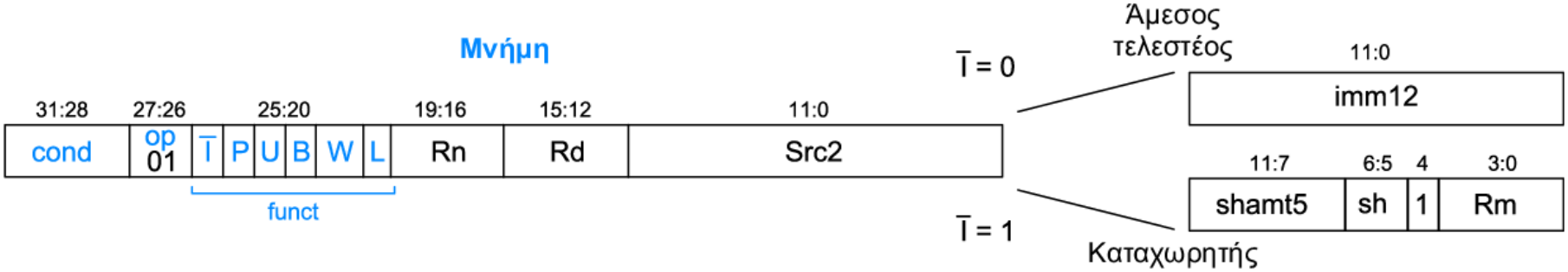
Κώδικας συμβολικής γλώσσας της ARM	Διεύθυνση μνήμης
`LDR R0, [R3, #4]`	R3 + 4
`LDR R0, [R5, #-16]`	R5 – 16
`LDR R1, [R6, R7]`	R6 + R7
`LDR R2, [R8, -R9]`	R8 – R9
`LDR R3, [R10, R11, LSL #2]`	R10 + (R11 << 2)
`LDR R4, [R1, -R12, ASR #4]`	R1 – (R12 >>> 4)
`LDR R0, [R9]`	R9

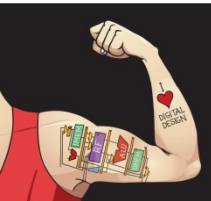


Μορφή εντολής μνήμης

Κωδικοποιεί: `LDR`, `STR`, `LDRB`, `STRB`

- ***op*** = 01_2
- ***Rn*** = καταχωρητής βάσης
- ***Rd*** = προορισμός (φόρτωση), προέλευση (αποθήκευση)
- ***Src2*** = **σχετική απόσταση: καταχωρητής (προαιρετικά ολισθαίνων) ή άμεσος τελεστέος**
- ***funct*** = 6 bit ελέγχου



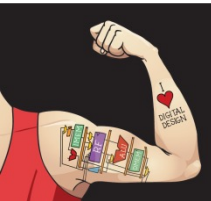


Τρόποι διευθυνσιοδότησης

Τρόπος	Διεύθυνση	Ενημέρωση καταχωρητή βάσης
Σχετική απόσταση	Καταχωρητής βάσης ± σχετική απόσταση	Χωρίς μεταβολή
Προ-αριθμο-δεικτοδότηση	Καταχωρητής βάσης ± σχετική απόσταση	Καταχωρητής βάσης ± σχετική απόσταση
Μετα-αριθμο-δεικτοδότηση	Καταχωρητής βάσης	Καταχωρητής βάσης ± σχετική απόσταση

Παραδείγματα

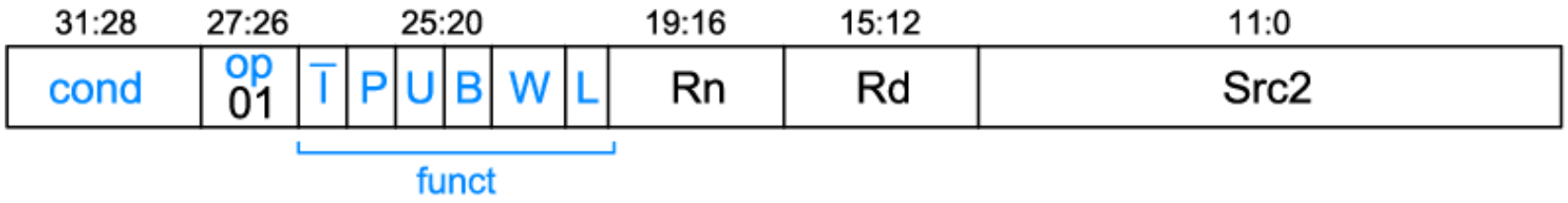
- **Σχετική απόσταση:**
```
LDR R1, [R2, #4]      ; R1 = mem[R2+4]
```
- **Προ-αριθμο-δεικτοδότηση :**
```
LDR R3, [R5, #16]!    ; R3 = mem[R5+16]
                      ; R5 = R5 + 16
```
- **Μετα-αριθμο-δεικτοδότηση:**
```
LDR R8, [R1], #8      ; R8 = mem[R1]
                      ; R1 = R1 + 8
```



Μορφή εντολής μνήμης

- ***funct***:
  - $\overline{I}$: immediate (άμεσος)
  - ***P***: προ-αριθμοδεικτοδότηση
  - ***U***: add (πρόσθεση)
  - ***B***: byte
  - ***W***: writeback (ετεροχρονισμένη εγγραφή)
  - ***L***: load (φόρτωση)

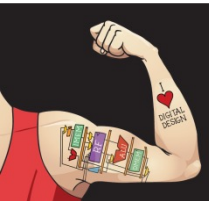
Μνήμη



Μορφή εντολής μνήμης: Κωδικοποιήσεις του *funct*

Τύπος πράξης

L	*B*	Εντολή
0	0	`STR`
0	1	`STRB`
1	0	`LDR`
1	1	`LDRB`



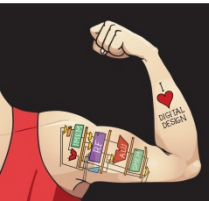
Μορφή εντολής μνήμης: Κωδικοποιήσεις του *funct*

Τύπος πράξης

L	*B*	Εντολή
0	0	`STR`
0	1	`STRB`
1	0	`LDR`
1	1	`LDRB`

Τρόπος διευθυνσιοδότησης

P	*W*	Τρόπος διευθυνσιοδότησης
0	1	Δεν υποστηρίζεται
0	0	Μετα-αριθμοδεικτοδότηση
1	0	Σχετική απόσταση
1	1	Προ-αριθμοδεικτοδότηση



Μορφή εντολής μνήμης: Κωδικοποιήσεις του *funct*

Τύπος πράξης

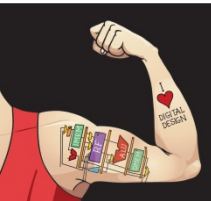
L	*B*	Εντολή
0	0	STR
0	1	STRB
1	0	LDR
1	1	LDRB

Τρόπος διευθυνσιοδότησης

P	*W*	Τρόπος διευθυνσιοδότησης
0	1	Δεν υποστηρίζεται
0	0	Μετα-αριθμοδεικτοδότηση
1	0	Σχετική απόσταση
1	1	Προ-αριθμοδεικτοδότηση

Σχετική απόσταση καταχωρητή/άμεσου τελεστέου για πρόσθεση/αφαίρεση

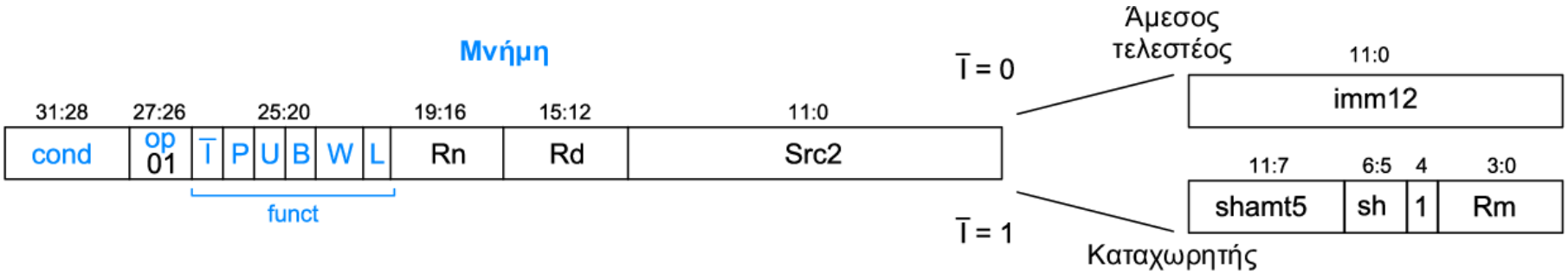
Τιμή	*I*	*U*
0	Σχετική απόσταση **άμεσου τελεστέου** στο *Src2*	**Αφαίρεση** σχετικής απόστασης από τη βάση
1	Σχετική απόσταση **καταχωρητή** στο *Src2*	**Πρόσθεση** σχετικής απόστασης στη βάση

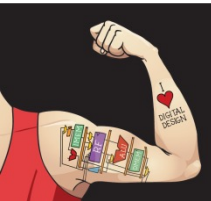


Μορφή εντολής μνήμης

Κωδικοποιεί: `LDR`, `STR`, `LDRB`, `STRB`

- ***op*** = 01_2
- ***Rn*** = καταχωρητής βάσης
- ***Rd*** = προορισμός (φόρτωση), προέλευση (αποθήκευση)
- ***Src2*** = σχετική απόσταση: άμεσος τελεστέος ή καταχωρητής (προαιρετικά ολισθαίνων)
- ***funct*** = $\overline{I}$ (άμεσος), *P* (προ-αριθμοδεικτοδότηση), *U* (πρόσθεση), *B* (byte), *W* (ετεροχρονισμένη εγγραφή), *L* (φόρτωση)



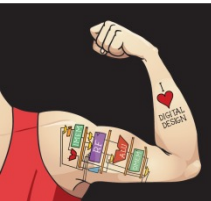


Εντολή μνήμης με άμεσο τελεστέο *Src2*

```
STR R11, [R5], #-26
```

- **Πράξη:** mem[R5] <= R11; R5 = R5 - 26
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 01_2 (1) για εντολή μνήμης
- ***funct*** = 0000000_2 (0)

  I = 0 (άμεσος τελεστέος για σχετική απόσταση), ***P*** = 0 (μετα-αριθμοδεικτοδότηση), ***U*** = 0 (αφαίρεση), ***B*** = 0 (αποθήκευση λέξης), ***W*** = 0 (μετα-αριθμοδεικτοδότηση, ***L*** = 0 (αποθήκευση)
- ***Rd*** = 11, ***Rn*** = 5, ***imm12*** = 26



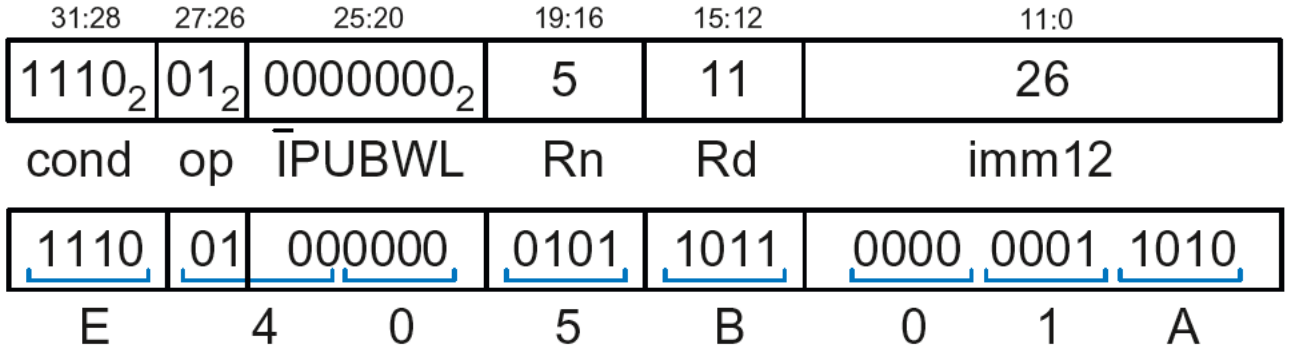
Εντολή μνήμης με άμεσο τελεστέο *Src2*

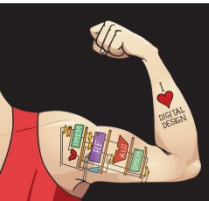
```
STR R11, [R5], #-26
```

- **Πράξη:** mem[R5] <= R11; R5 = R5 - 26
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 01_2 (1) για εντολή μνήμης
- ***funct*** = 0000000_2 (0)

  I = 0 (άμεσος τελεστέος για σχετική απόσταση), ***P*** = 0 (μετα-αριθμοδεικτοδότηση), ***U*** = 0 (αφαίρεση), ***B*** = 0 (αποθήκευση λέξης), ***W*** = 0 (μετα-αριθμοδεικτοδότηση, ***L*** = 0 (αποθήκευση)
- ***Rd*** = 11, ***Rn*** = 5, ***imm12*** = 26

Τιμές πεδίων



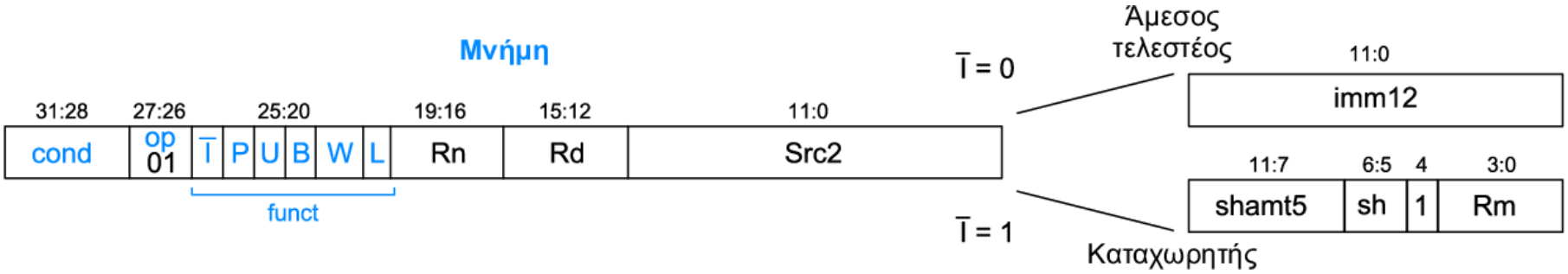


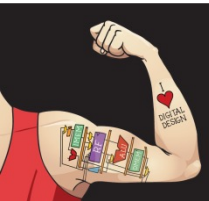
Εντολή μνήμης με καταχωρητή *Src2*

```
LDR R3, [R4, R5]
```

- **Πράξη:** R3 <= mem[R4 + R5]
- ***cond*** = 1110_2 (14) για συνθήκη χωρίς εκτέλεση
- ***op*** = 01_2 (1) για εντολή μνήμης
- ***funct*** = 111001_2 (57)
  I = 1 (καταχ̄ωρητής για σχετική απόσταση), ***P*** = 1 (διευθυνσιοδότηση με σχετική απόσταση), ***U*** = 1 (πρόσθεση), ***B*** = 0 (φόρτωση **λέξης**), ***W*** = 0 (διευθυνσιοδότηση με σχετική απόσταση), ***L*** = 1 (φόρτωση)
- ***Rd*** = 3, ***Rn*** = 4, ***Rm*** = 5 (***shamt5*** = 0, ***sh*** = 0)

1110 01 111001 0100 0011 00000 00 0 0101 = **0xE7943005**





Εντολή μνήμης με προσαρμοσμένο καταχωρητή *Src2*

```
STR R9, [R1, R3, LSL #2]
```

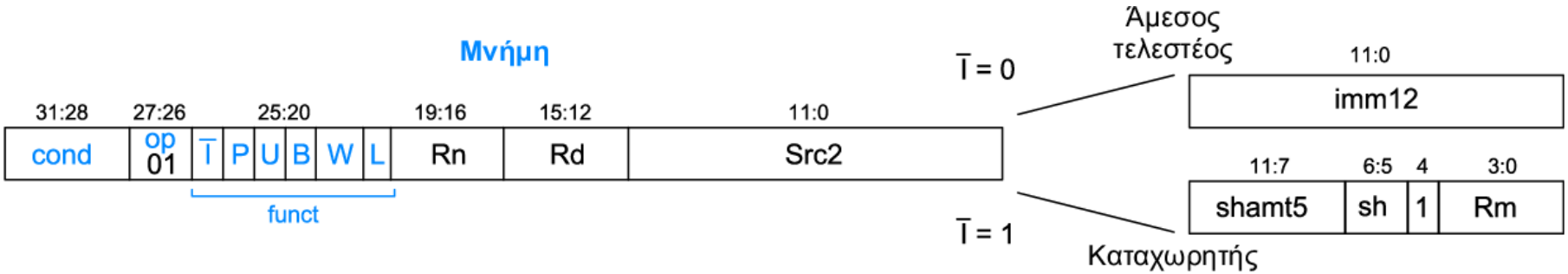
- **Πράξη:** mem[R1 + (R3 << 2)] <= R9
- ***cond*** = 1110_2 (14) για εκτέλεση χωρίς συνθήκη
- ***op*** = 01_2 (1) για εντολή μνήμης
- ***funct*** = 111000_2 (0)

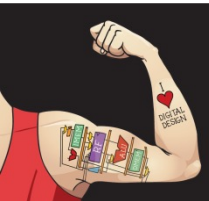
  I = 1 (καταχωρητής για σχετική απόσταση), ***P*** = 1 (διευθυνσιοδότηση με σχετική απόσταση),

  U = 1 (πρόσθεση), ***B*** = 0 (αποθήκευση **λέξης**), ***W*** = 0 (διευθυνσιοδότηση με σχετική απόσταση),

  L = 0 (αποθήκευση)
- ***Rd*** = 9, ***Rn*** = 1, ***Rm*** = 3, ***shamt*** = 2, ***sh*** = 00_2 (LSL)

1110 01 111000 0001 1001 00010 00 0 0011 = **0xE7819103**

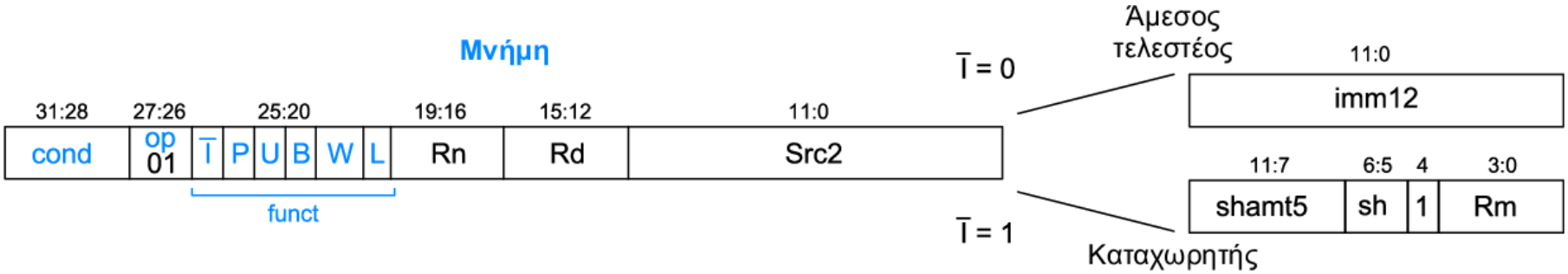


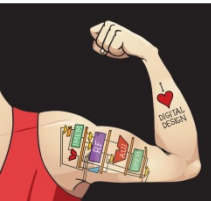


Επανάληψη: Μορφή εντολής μνήμης

Κωδικοποιεί: `LDR`, `STR`, `LDRB`, `STRB`

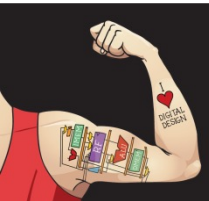
- ***op*** = 01_2
- ***Rn*** = καταχωρητής βάσης
- ***Rd*** = προορισμός (φόρτωση), προέλευση (αποθήκευση)
- ***Src2*** = σχετική απόσταση: καταχωρητής (προαιρετικά ολισθαίνων) ή άμεσος τελεστέος
- ***funct*** = $\overline{I}$ (άμεσος), *P* (προαριθμοδεικτοδότηση), *U* (πρόσθεση), *B* (byte), *W* (ετεροχρονισμένη εγγραφή), *L* (φόρτωση)





Μορφές εντολών

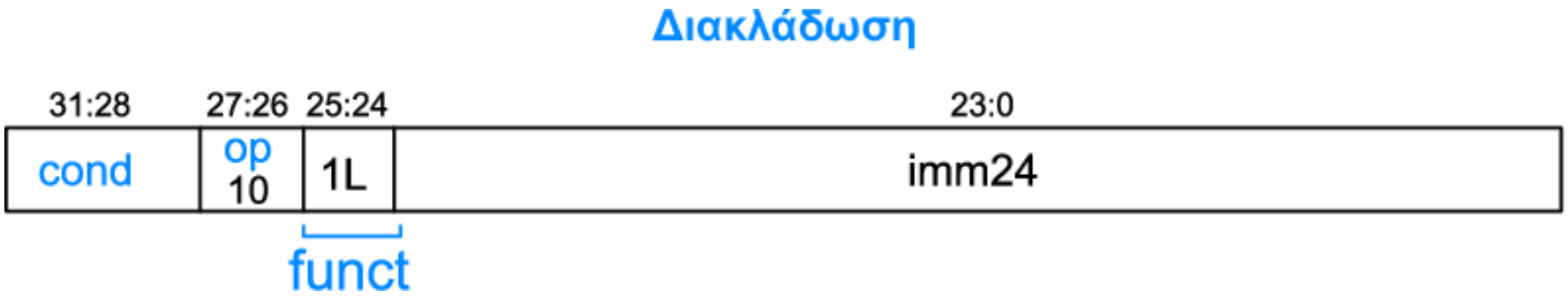
- Επεξεργασίας δεδομένων
- Μνήμης
- **Διακλάδωσης**

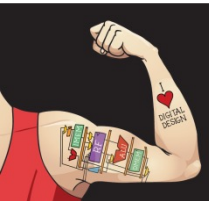


Μορφή εντολής διακλάδωσης

Κωδικοποιεί τις εντολές `B` και `BL`

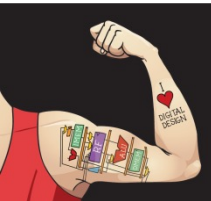
- ***op*** = 10_2
- ***imm24*****:** Άμεσος τελεστέος των 24 bit
- ***funct*** = $1L_2$: ***L*** = 1 για την `BL`, ***L*** = 0 για την `B`





Κωδικοποίηση διεύθυνσης-στόχου διακλάδωσης

- ***Διεύθυνση-στόχος της διακλάδωσης (Branch Target Address,* BTA)*:*** Το επόμενο PC αν ακολουθηθεί η διακλάδωση
- Η BTA ορίζεται σε σχέση με το τρέχον PC + 8
- Το *imm24* κωδικοποιεί την BTA
- ***imm24*** = πλήθος λέξεων (απόσταση) μεταξύ BTA και PC+8

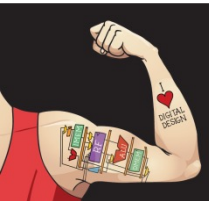


Εντολή διακλάδωσης: Παράδειγμα 1

Κώδικας συμβολικής γλώσσας της ARM

```
0xA0          BLT THERE        ← PC
0xA4          ADD R0, R1, R2
0xA8          SUB R0, R0, R9   ← PC+8
0xAC          ADD SP, SP, #8
0xB0          MOV PC, LR
0xB4  THERE   SUB R0, R0, #1   ← BTA
0xB8          BL  TEST
```

- PC = 0xA0
- PC + 8 = 0xA8
- Η ετικέτα `THERE` βρίσκεται 3 εντολές μετά το PC+8
- Άρα, *imm24* = 3



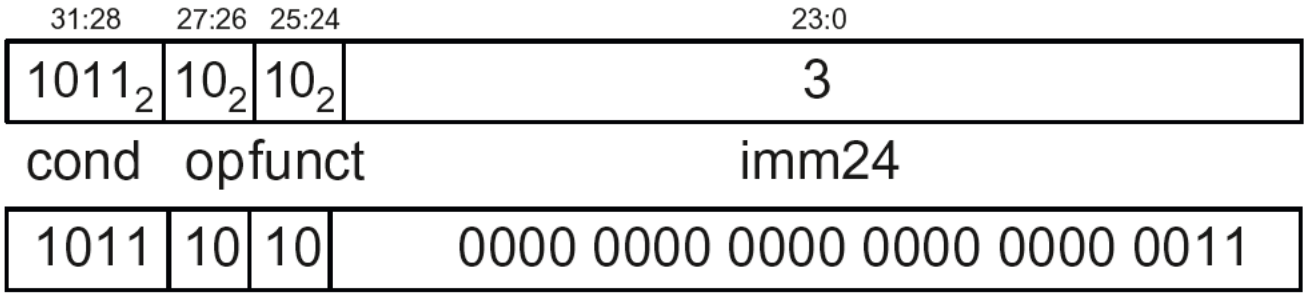
Εντολή διακλάδωσης: Παράδειγμα 1

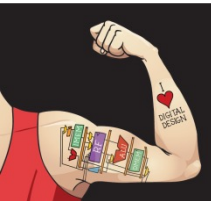
Κώδικας συμβολικής γλώσσας της ARM

```
0xA0          BLT  THERE        ← PC
0xA4          ADD  R0, R1, R2
0xA8          SUB  R0, R0, R9   ← PC+8
0xAC          ADD  SP, SP, #8
0xB0          MOV  PC, LR
0xB4  THERE   SUB  R0, R0, #1   ← BTA
0xB8          BL   TEST
```

- PC = 0xA0
- PC + 8 = 0xA8
- Η ετικέτα `THERE` βρίσκεται 3 εντολές μετά το PC+8
- Άρα, *imm24* = 3

Τιμές πεδίων





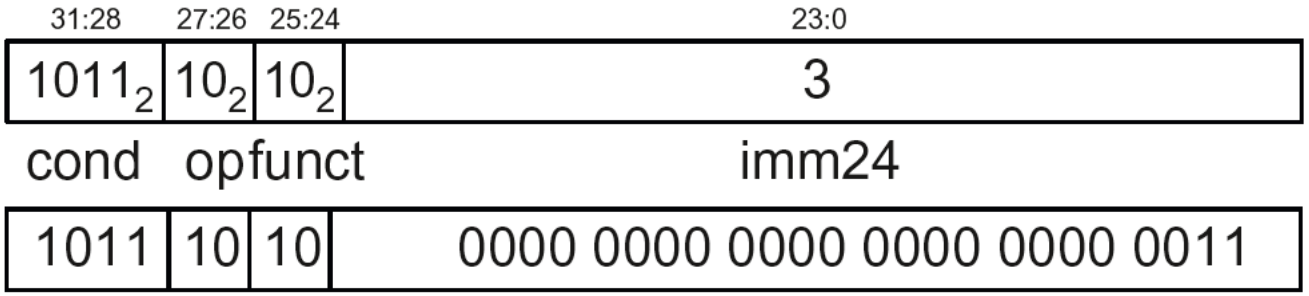
Εντολή διακλάδωσης: Παράδειγμα 1

Κώδικας συμβολικής γλώσσας της ARM

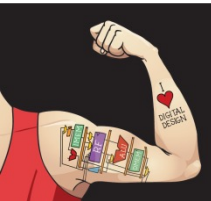
```
0xA0          BLT  THERE        ← PC
0xA4          ADD  R0, R1, R2
0xA8          SUB  R0, R0, R9   ← PC+8
0xAC          ADD  SP, SP, #8
0xB0          MOV  PC, LR
0xB4  THERE   SUB  R0, R0, #1   ← BTA
0xB8          BL   TEST
```

- PC = 0xA0
- PC + 8 = 0xA8
- Η ετικέτα `THERE` βρίσκεται 3 εντολές μετά το PC+8
- Άρα, *imm24* = 3

Τιμές πεδίων



0xBA000003

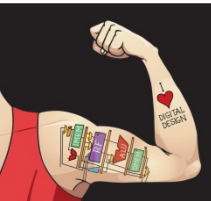


Εντολή διακλάδωσης: Παράδειγμα 2

Κώδικας συμβολικής γλώσσας της ARM

```
0x8040 TEST  LDRB R5, [R0, R3]   ← BTA
0x8044       STRB R5, [R1, R3]
0x8048       ADD  R3, R3, #1
0x8044       MOV  PC, LR
0x8050       BL   TEST           ← PC
0x8054       LDR  R3, [R1], #4
0x8058       SUB  R4, R3, #9     ← PC+8
```

- PC = 0x8050
- PC + 8 = 0x8058
- Η ετικέτα `TEST` βρίσκεται 6 εντολές μετά το PC+8
- Άρα, *imm24* = -6



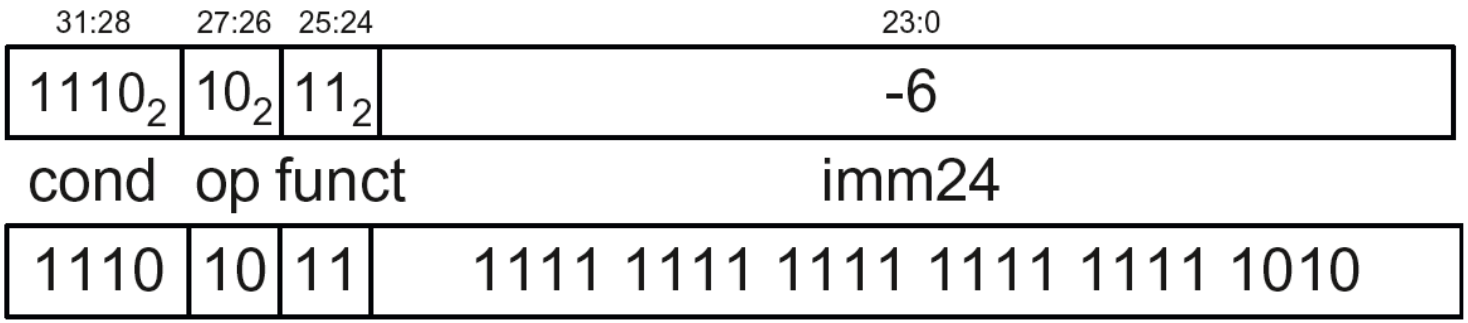
Εντολή διακλάδωσης: Παράδειγμα 2

Κώδικας συμβολικής γλώσσας της ARM

```
0x8040 TEST  LDRB R5, [R0, R3]   ← BTA
0x8044       STRB R5, [R1, R3]
0x8048       ADD  R3, R3, #1
0x8044       MOV  PC, LR
0x8050       BL   TEST           ← PC
0x8054       LDR  R3, [R1], #4
0x8058       SUB  R4, R3, #9    ← PC+8
```

- PC = 0x8050
- PC + 8 = 0x8058
- Η ετικέτα `TEST` βρίσκεται 6 εντολές μετά το PC+8
- Άρα, *imm24* = -6

Τιμές πεδίων



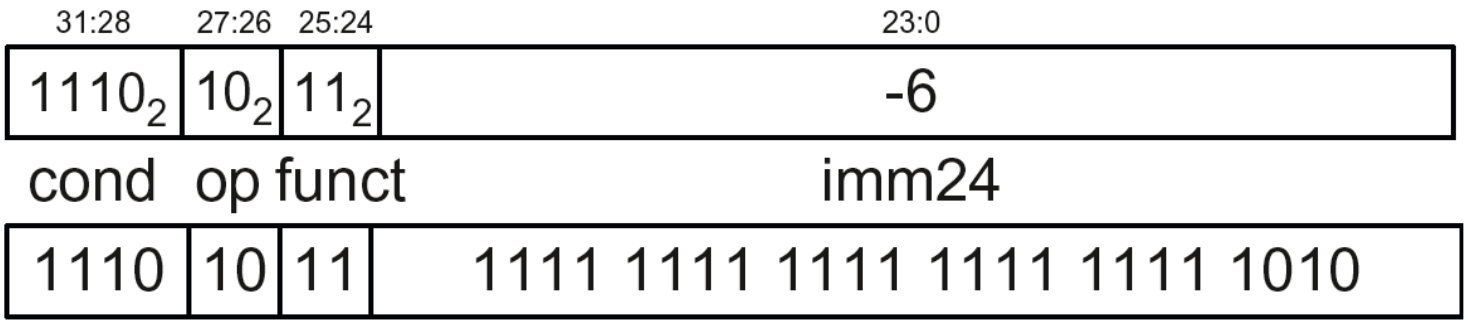
Εντολή διακλάδωσης: Παράδειγμα 2

Κώδικας συμβολικής γλώσσας της ARM

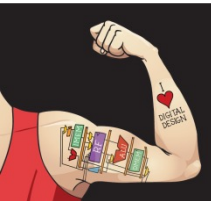
```
0x8040 TEST   LDRB R5, [R0, R3]   ← BTA
0x8044        STRB R5, [R1, R3]
0x8048        ADD  R3, R3, #1
0x8044        MOV  PC, LR
0x8050        BL   TEST           ← PC
0x8054        LDR  R3, [R1], #4
0x8058        SUB  R4, R3, #9     ← PC+8
```

- PC = 0x8050
- PC + 8 = 0x8058
- Η ετικέτα `TEST` βρίσκεται 6 εντολές μετά το PC+8
- Άρα, *imm24* = -6

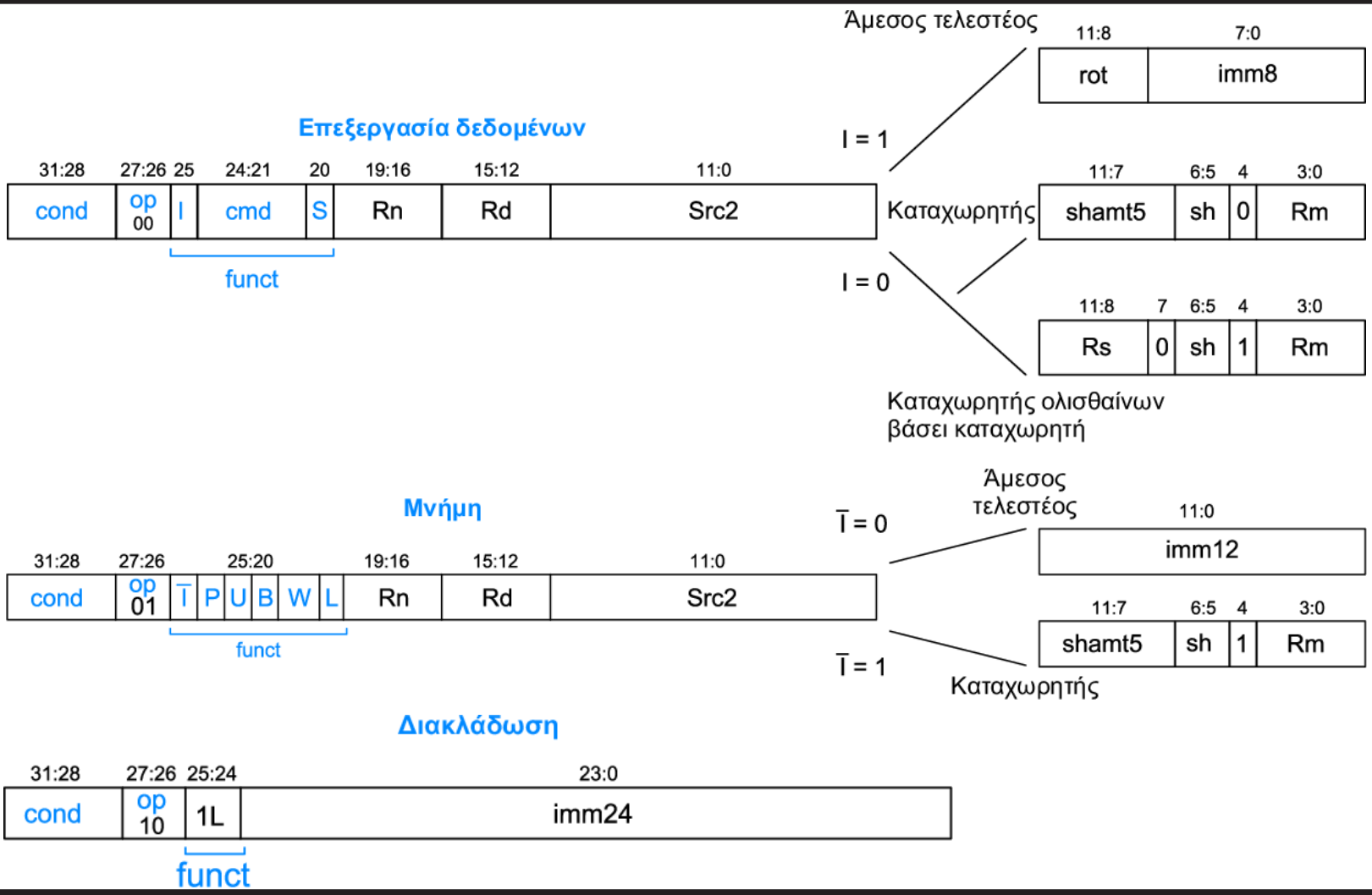
Τιμές πεδίων



0xEBFFFFFA



Επανάληψη: Μορφές εντολών

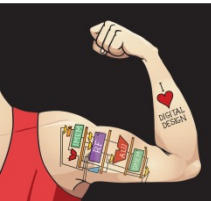


Εκτέλεση υπό συνθήκη

Κωδικοποιείται στα bit του πεδίου *cond* της εντολής μηχανής

Για παράδειγμα,

```
ANDEQ R1, R2, R3      (cond = 0000)
ORRMI R4, R5, #0xF    (cond = 0100)
SUBLT R9, R3, R8      (cond = 1011)
```



Επανάληψη: Μνημονικά συνθήκης

cond	Μνημονικό	Όνομα	CondEx
0000	EQ	Equal (ίσοι)	Z
0001	NE	Not equal (μη ίσοι)	$\bar{Z}$
0010	CS / HS	Carry set / Unsigned higher or same (καθορισμός κρατουμένου / μη προσημασμένος υψηλότερος ή ίδιος)	C
0011	CC / LO	Carry clear / Unsigned lower (επαναφορά κρατουμένου / μη προσημασμένος χαμηλότερος)	$\bar{C}$
0100	MI	Minus / Negative (μείον / αρνητικό)	N
0101	PL	Plus / Positive of zero	$\bar{N}$
0110	VS	Overflow / Overflow set (υπερχείλιση / ενεργοποίηση υπερχείλισης)	V
0111	VC	No overflow / Overflow clear (μη υπερχείλιση / επαναφορά υπερχείλισης)	$\bar{V}$
1000	HI	Unsigned higher (μη προσημασμένος υψηλότερος)	$\bar{Z}C$
1001	LS	Unsigned lower or same (μη προσημασμένος χαμηλότερος ή ίδιος)	$Z\ OR\ \bar{C}$
1010	GE	Signed greater than or equal (προσημασμένος μεγαλύτερος ή ίσος)	$\overline{N \oplus V}$
1011	LT	Signed less than (προσημασμένος μικρότερος)	$N \oplus V$
1100	GT	Signed greater than (προσημασμένος μεγαλύτερος)	$\bar{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal (προσημασμένος μικρότερος ή ίσος)	$Z\ OR\ (N \oplus V)$
1110	AL (ή none)	Always / unconditional (πάντα / χωρίς συνθήκη)	Αγνοείται

Εκτέλεση υπό συνθήκη: Κώδικας μηχανής

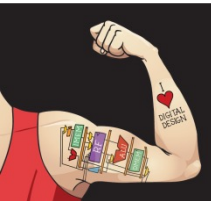
Κώδικας συμβολικής γλώσσας

Τιμές πεδίων

	31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
`SUBS R1, R2, R3`	14	0	0	2	1	2	1	0	0	0	3
`ADDEQ R4, R5, R6`	0	0	0	4	0	5	4	0	0	0	6
`ANDHS R7, R5, R6`	2	0	0	0	0	5	7	0	0	0	6
`ORRMI R8, R5, R6`	4	0	0	12	0	5	8	0	0	0	6
`EORLT R9, R5, R6`	11	0	0	1	0	5	9	0	0	0	6
	cond	op	I	cmd	S	rn	rd	shamt5	sh		rm

Κώδικας γλώσσας μηχανής

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0	
1110	00	0	0010	0	0010	0001	00000	00	0	0011	(0xE0421003)
0000	00	0	0100	0	0101	0100	00000	00	0	0110	(0x00854006)
0010	00	0	0000	0	0101	0111	00000	00	0	0110	(0x20057006)
0100	00	0	1100	0	0101	1000	00000	00	0	0110	(0x41858006)
1011	00	0	0001	0	0101	1001	00000	00	0	0110	(0xB0259006)
cond	op	I	cmd	S	rn	rd	shamt5	sh		rm	

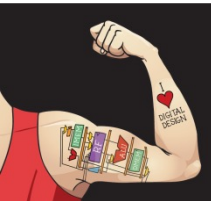


Ερμηνεία κώδικα μηχανής

- **Ξεκινάμε με το πεδίο *op*:** Μας λέει πώς να ερμηνεύσουμε τα υπόλοιπα
  - ***op*** = 00 (επεξεργασία δεδομένων)
  - ***op*** = 01 (μνήμη)
  - ***op*** = 10 (διακλάδωση)
- **Πεδίο *I* (bit):** Μας λέει πώς να ερμηνεύσουμε το ***Src2***
- **Εντολές επεξεργασίας δεδομένων:**

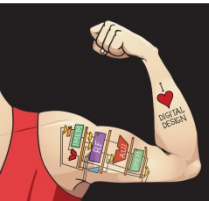
  Αν το bit ***I*** έχει την τιμή 0, το bit 4 καθορίζει αν το ***Src2*** είναι καταχωρητής (bit 4 = 0) ή καταχωρητής ολισθαίνων βάσει καταχωρητή (bit 4 = 1)
- **Εντολές μνήμης:**

  Εξετάζουμε τα bit του πεδίου ***funct*** για τον τρόπο διευθυνσιοδότησης, την εντολή, και τη σχετική απόσταση που πρέπει να προστεθεί ή να αφαιρεθεί



Ερμηνεία κώδικα μηχανής: Παράδειγμα 1

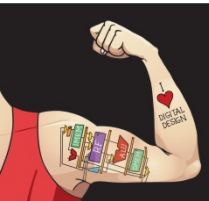
0xE0475001



Ερμηνεία κώδικα μηχανής: Παράδειγμα 1

0xE0475001

- **Ξεκινάμε με το πεδίο *op*:** 00_2, άρα εντολή επεξεργασίας δεδομένων

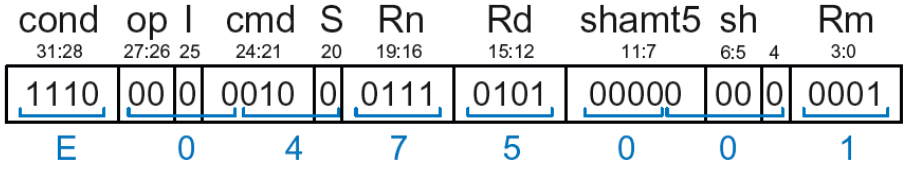


Ερμηνεία κώδικα μηχανής: Παράδειγμα 1

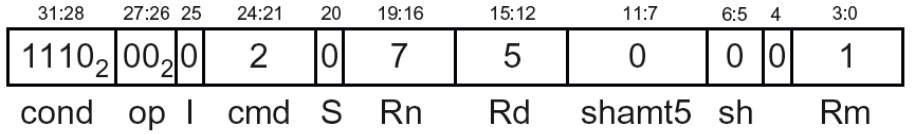
0xE0475001

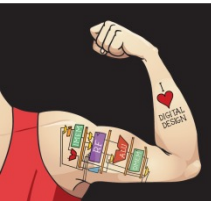
- **Ξεκινάμε με το πεδίο *op*:** 00_2, άρα εντολή επεξεργασίας δεδομένων
- **Bit *I*:** 0, άρα το *Src2* είναι καταχωρητής
- **Bit 4:** 0, άρα το *Src2* είναι καταχωρητής (προαιρετικά ολισθαίνων κατά *shamt5*)

Κώδικας γλώσσας μηχανής



Τιμές πεδίων



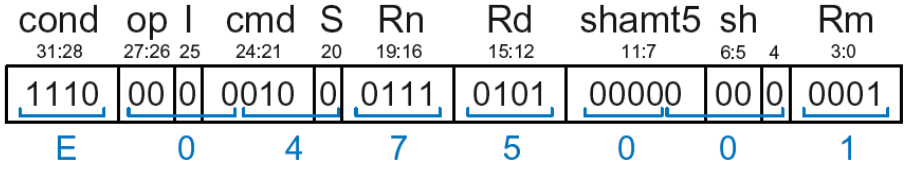


Ερμηνεία κώδικα μηχανής: Παράδειγμα 1

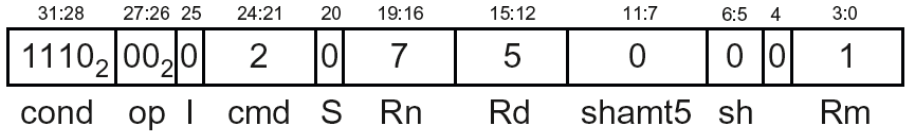
0xE0475001

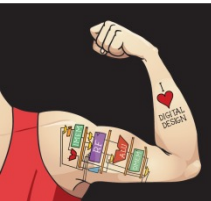
- **Ξεκινάμε με το πεδίο *op*:** 00_2, άρα εντολή επεξεργασίας δεδομένων
- **Bit *I*:** 0, άρα το *Src2* είναι καταχωρητής
- **Bit 4:** 0, άρα το *Src2* είναι καταχωρητής (προαιρετικά ολισθαίνων κατά *shamt5*)
- ***cmd*:** 0010_2 (2), άρα εντολή SUB
- **Rn** = 7, **Rd** = 5, **Rm** = 1, ***shamt5*** = 0, ***sh*** = 0
- Επομένως, η εντολή είναι **`SUB R5,R7,R1`**

Κώδικας γλώσσας μηχανής



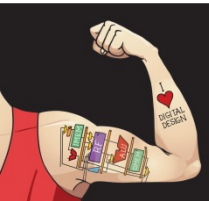
Τιμές πεδίων





Ερμηνεία κώδικα μηχανής: Παράδειγμα 2

0xE5949010



Ερμηνεία κώδικα μηχανής: Παράδειγμα 2

0xE5949010

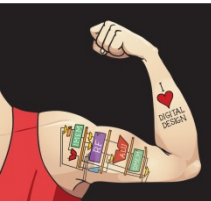
- **Ξεκινάμε με το πεδίο *op*:** 01_2, άρα εντολή μνήμης
- ***funct*:** *B* = 0, *L* = 1, άρα εντολή `LDR`· *P* = 1, *W* = 0, άρα διευθυνσιοδότηση με σχετική απόσταση· *I* = 0, άρα άμεσος τελεστέος για σχετική απόσταση· *U* = 1, άρα πρόσθεση σχετική απόστασης
- **Rn** = 4, **Rd** = 9, ***imm12*** = 16
- Επομένως, η εντολή είναι **`LDR R9,[R4,#16]`**

Κώδικας γλώσσας μηχανής

cond	op	$\bar{I}$PUBWL	Rn	Rd	imm12
31:28	27:26	25:20	19:16	15:12	11:0
1110	01	011001	0100	1001	0000 0001 0000
E	5	9	4	9	0 1 0

Τιμές πεδίων

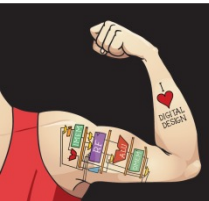
31:28	27:26	25:20	19:16	15:12	11:0
1110_2	01_2	25	4	9	16
cond	op	$\bar{I}$PUBWL	Rn	Rd	imm12



Τρόποι διευθυνσιοδότησης

Πώς διευθυνσιοδοτούμε τελεστέους;

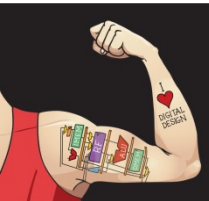
- Βάσει καταχωρητή
- Άμεση
- Βάσης
- Σε σχέση με τον μετρητή PC



Τρόποι διευθυνσιοδότησης

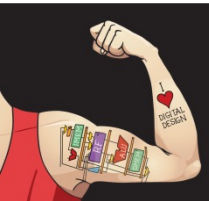
Πώς διευθυνσιοδοτούμε τελεστέους;

- **Μόνο καταχωρητής**
- Άμεση
- Βάσης
- Σε σχέση με τον μετρητή PC



Διευθυνσιοδότηση βάσει καταχωρητή

- Οι τελεστέοι προέλευσης και προορισμού είναι αποθηκευμένοι σε καταχωρητές
- Χρησιμοποιείται από εντολές επεξεργασίας δεδομένων
- **Τρεις υποκατηγορίες:**
  - Μόνο καταχωρητής
  - Καταχωρητής ολισθαίνων βάσει άμεσου τελεστέου
  - Καταχωρητής ολισθαίνων βάσει καταχωρητή



Διευθυνσιοδότηση βάσει καταχωρητή: Παραδείγματα

- **Μόνο καταχωρητής**

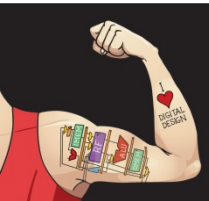
  Παράδειγμα: `ADD R0, R2, R7`

- **Καταχωρητής ολισθαίνων βάσει άμεσου τελεστέου**

  Παράδειγμα: `ORR R5, R1, R3, LSL #1`

- **Καταχωρητής ολισθαίνων βάσει καταχωρητή**

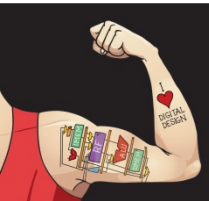
  Παράδειγμα: `SUB R12, R9, R0, ASR R1`



Τρόποι διευθυνσιοδότησης

Πώς διευθυνσιοδοτούμε τελεστέους;

- Μόνο καταχωρητής
- **Άμεση**
- Βάσης
- Σε σχέση με τον μετρητή PC

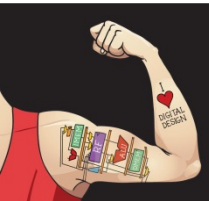


Άμεση διευθυνσιοδότηση

- Οι τελεστέοι προέλευσης και προορισμού είναι αποθηκευμένοι σε καταχωρητές **ή** είναι άμεσοι τελεστέοι

  Παράδειγμα: `ADD R9, R1, #14`

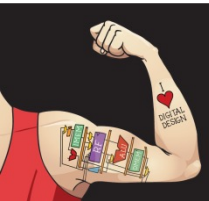
- Χρησιμοποιεί τη μορφή επεξεργασίας δεδομένων με *I* = 1
- Ο άμεσος τελεστές κωδικοποιείται ως
  - Άμεσος τελεστέος των 8 bit (*imm8*)
  - Τιμή περιστροφής των 4 bit (*rot*)
- Άμεσος τελεστέος των 32 bit = *imm8* ROR (*rot* x 2)



Τρόποι διευθυνσιοδότησης

Πώς διευθυνσιοδοτούμε τελεστέους;

- Μόνο καταχωρητής
- Άμεση
- **Βάσης**
- Σε σχέση με τον μετρητή PC

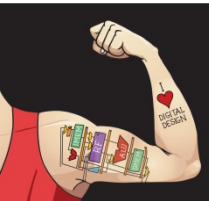


Διευθυνσιοδότηση βάσης

- Η διεύθυνση του τελεστέου είναι:

  καταχωρητής βάσης + σχετική απόσταση

- Η σχετική απόσταση μπορεί να είναι:
  - Άμεσος τελεστέος των 12 bit
  - Καταχωρητής
  - Καταχωρητής ολισθαίνων βάσει άμεσου τελεστέου



Διευθυνσιοδότηση βάσης: Παραδείγματα

- **Σχετική απόσταση άμεσου τελεστέου**

  Παράδειγμα: `LDR R0, [R8, #-11]`

  (R0 = mem[R8 – 11])

- **Σχετική απόσταση καταχωρητή**

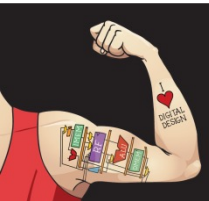
  Παράδειγμα: `LDR R1, [R7, R9]`

  (R1 = mem[R7 + R9])

- **Σχετική απόσταση καταχωρητή ολισθαίνοντος βάσει άμεσου τελεστέου**

  Παράδειγμα: `STR R5, [R3, R2, LSL #4]`

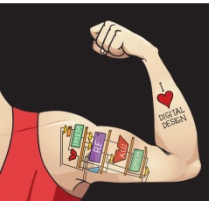
  (R5 = mem[R3 + (R2 << 4)])



Τρόποι διευθυνσιοδότησης

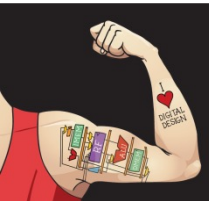
Πώς διευθυνσιοδοτούμε τελεστέους;

- Μόνο καταχωρητής
- Άμεση
- Βάσης
- **Σε σχέση με τον μετρητή PC**



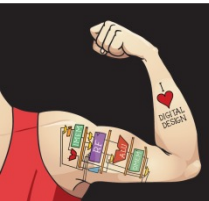
Διευθυνσιοδότηση σε σχέση με τον μετρητή PC

- Χρησιμοποιείται για διακλαδώσεις
- Μορφή εντολής διακλάδωσης:
  - Οι τελεστέοι είναι ο μετρητής PC και ένας προσημασμένος άμεσος τελεστέος των 24 bit (*imm24*)
  - Αλλάζει την τιμή του PC
  - Η νέα τιμή του PC ορίζεται σε σχέση με την προηγούμενη τιμή του PC
  - Το πεδίο *imm24* υποδεικνύει την απόσταση (πλήθος λέξεων) από το PC + 8
- PC = (PC + 8) + (SignExtended(*imm24*) x 4)



Η δύναμη του αποθηκευμένου προγράμματος

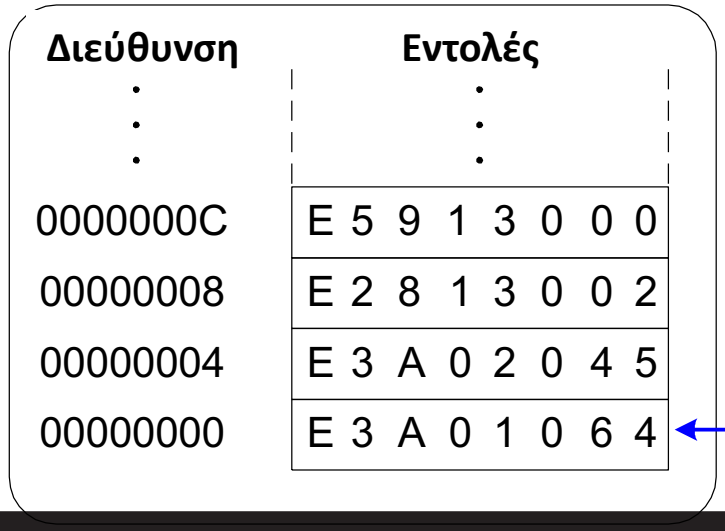
- **Εντολές και δεδομένα των 32 bit** αποθηκεύονται στη μνήμη
- **Ακολουθία εντολών:** η μόνη διαφορά μεταξύ δύο εφαρμογών
- **Για την εκτέλεση ενός νέου προγράμματος:**
  - Δεν απαιτείται επανασύνδεση συρμάτων
  - Το νέο πρόγραμμα απλώς αποθηκεύεται στη μνήμη
- **Εκτέλεση προγράμματος:**
  - Ο επεξεργαστής *προσκομίζει* (διαβάζει) ακολουθιακά εντολές από τη μνήμη
  - Ο επεξεργαστής εκτελεί την καθορισμένη πράξη



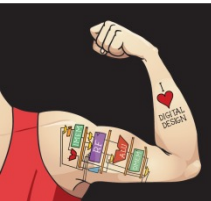
Το αποθηκευμένο πρόγραμμα

Κώδικας συμβολικής γλώσσας	Κώδικας γλώσσας μηχανής
`MOV R1, #100`	`0xE3A01064`
`MOV R2, #69`	`0xE3A02045`
`ADD R3, R1, R2`	`0xE2813002`
`STR R3, [R1]`	`0xE5913000`

Αποθηκευμένο πρόγραμμα



Μετρητής προγράμματος (program counter, PC): Παρακολουθεί την τρέχουσα εντολή



Τι θα μάθουμε στη συνέχεια

Πώς να υλοποιούμε το σύνολο εντολών της αρχιτεκτονικής ARM σε υλικό

Μικροαρχιτεκτονική