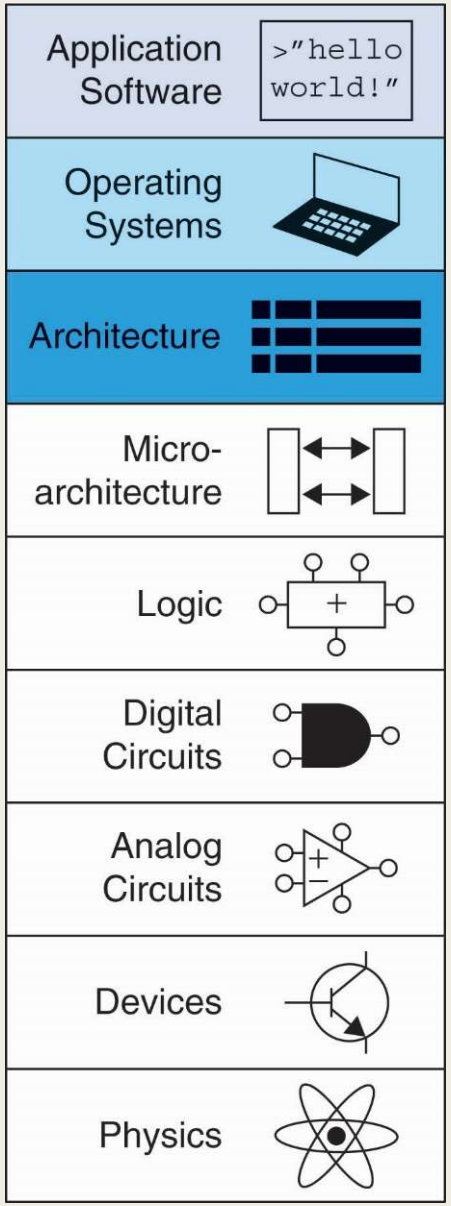
Κεφάλαιο 6

Αρχιτεκτονική

Ιωάννης Σίδερης, Αντώνης Πασχάλης

Περιεχόμενα κεφαλαίου

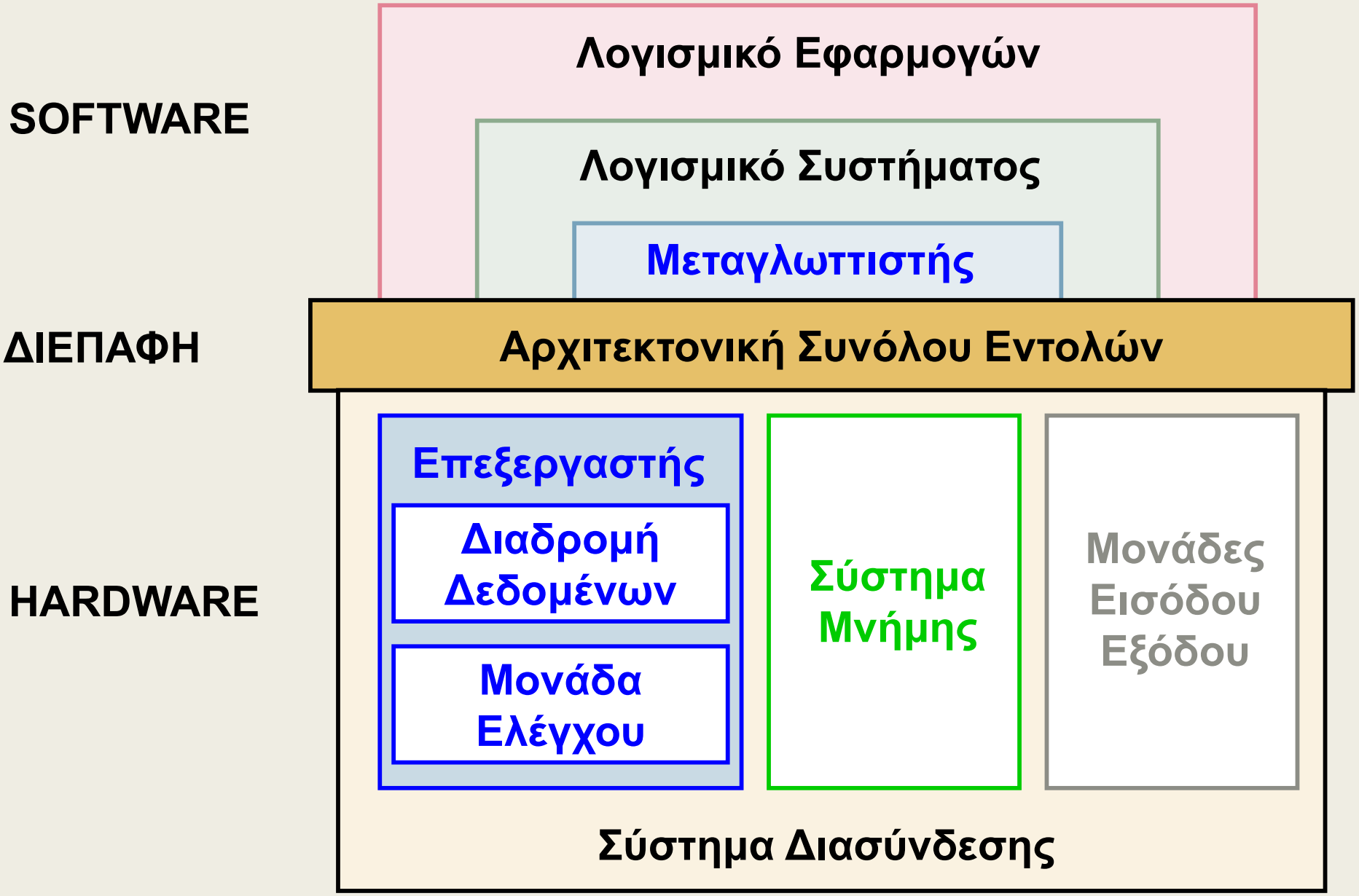
- Εισαγωγή
- Αρχιτεκτονική συνόλου εντολών
- Συμβολική γλώσσα
- Γλώσσα μηχανής
- Προγραμματισμός χαμηλού επιπέδου
- Τρόποι διευθυνσιοδότησης



Εισαγωγή

- Οι λέξεις της γλώσσας ενός υπολογιστή ονομάζονται **εντολές** (*instructions*) και ορίζουν ένα **σύνολο εντολών** (*instruction set*) σε μία συγκεκριμένη **αρχιτεκτονική συνόλου εντολών**
  - *Οποιοδήποτε πρόγραμμα που εκτελείται σε έναν υπολογιστή χρησιμοποιεί το ίδιο σύνολο εντολών*
- Οι εντολές κωδικοποιούνται ως δυαδικοί αριθμοί σε μια μορφή που ονομάζεται **γλώσσα μηχανής** (machine language).
  - *η αναπαράσταση εντολών σε γλώσσα μηχανής είναι δύσχρηστη*
- Προτιμούμε να αναπαριστούμε τις εντολές σε μια συμβολική μορφή που ονομάζεται **συμβολική γλώσσα** (assembly language)
  - *χρησιμοποιείται στον προγραμματισμό χαμηλού επιπέδου*
  - *εάν μάθουμε τη συμβολική γλώσσα ενός υπολογιστή μπορούμε να κατανοήσουμε τη συμβολική γλώσσα ενός άλλου υπολογιστή*
    - τουλάχιστον όσο αφορά στο βασικό σύνολο εντολών

Ο υπολογιστής



Το λογισμικό συστήματος

- Το απαιτούμενο λογισμικό για τη λειτουργία του υπολογιστή ως ένα ολοκληρωμένο σύστημα. Συμπεριλαμβάνει:
  - *το λειτουργικό σύστημα (operating system – OS)*
    - Πρόγραμμα επίβλεψης που διαχειρίζεται τους πόρους ενός υπολογιστή προς όφελος των εκτελούμενων προγραμμάτων
      - *Διαχείριση βασικών λειτουργιών εισόδου και εξόδου*
      - *Κατανομή χώρου αποθήκευσης και μνήμης*
      - *Παροχή δυνατοτήτων κοινής χρήσης του υπολογιστή μεταξύ πολλών εφαρμογών που τον χρησιμοποιούν ταυτόχρονα*
  - *τον μεταγλωττιστή (compiler)*
    - Πρόγραμμα που μεταφράζει εντολές μιας γλώσσας υψηλού επιπέδου (π.χ. C) σε εντολές συμβολικής γλώσσας (assembly language)
  - *τον συμβολομεταφραστή (assembler)*
    - Πρόγραμμα που μεταφράζει μια συμβολική έκδοση των εντολών στη δυαδική έκδοσή τους (σε γλώσσα μηχανής)

Το υλικό του υπολογιστή

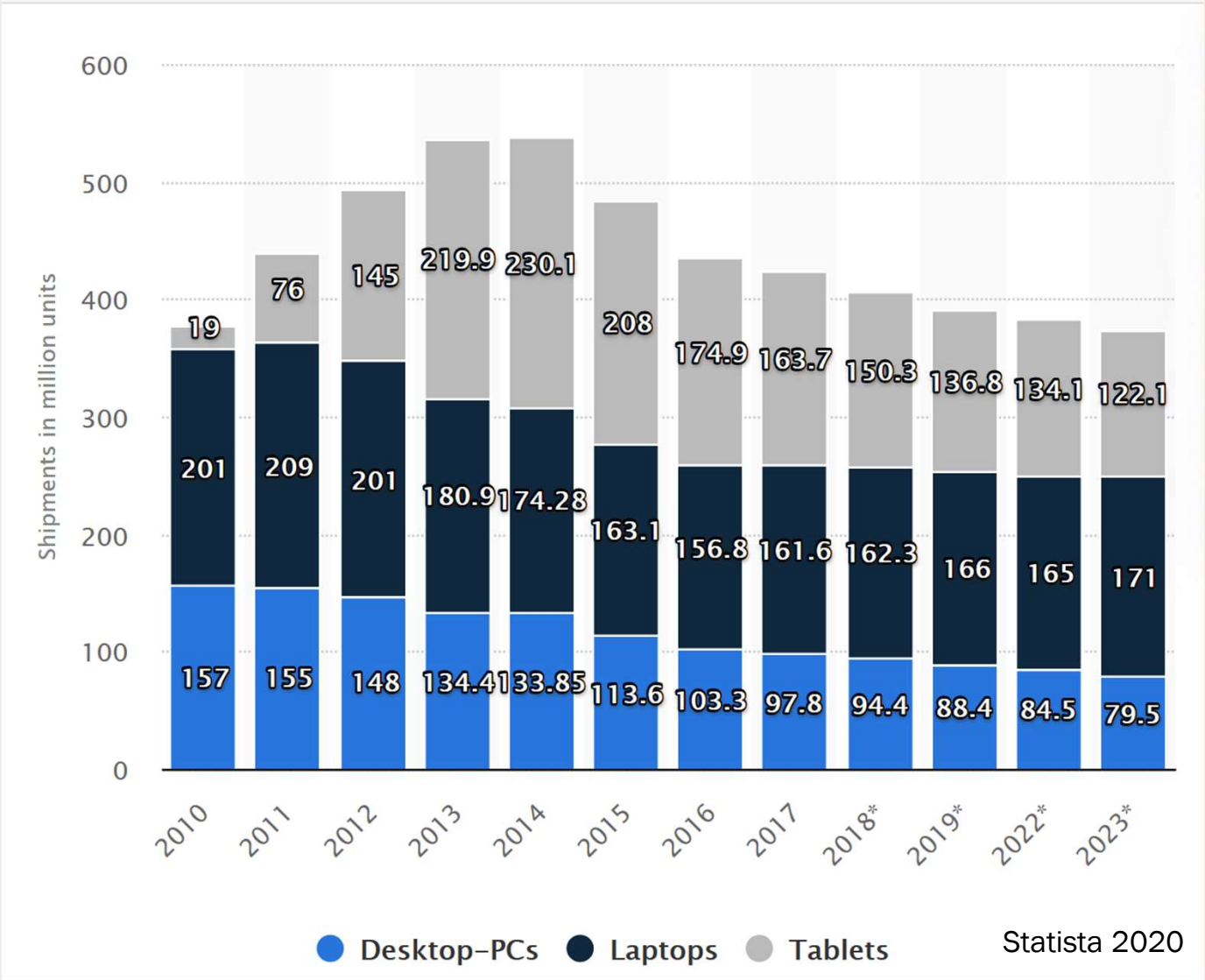
- Διαδρομή δεδομένων (datapath)
  - *το τμήμα τού επεξεργαστή που εκτελεί τις εντολές*
- Μονάδα ελέγχου (control unit)
  - *το τμήμα του επεξεργαστή που ελέγχει τη διαδρομή δεδομένων, τη μνήμη, και τις συσκευές εισόδου/εξόδου σύμφωνα με τις εντολές του προγράμματος*
- Μνήμη (memory)
  - *η περιοχή αποθήκευσης στην οποία διατηρούνται τα προγράμματα όταν εκτελούνται, και η οποία περιέχει τα δεδομένα που χρειάζονται τα προγράμματα αυτά*
- Συσκευή εισόδου (input device)
  - *ένας μηχανισμός μέσω του οποίου ο υπολογιστής τροφοδοτείται με πληροφορίες, όπως είναι το πληκτρολόγιο ή το ποντίκι*
- Συσκευή εξόδου (output device)
  - *Ένας μηχανισμός που μεταφέρει το αποτέλεσμα ενός υπολογισμού στο χρήστη (όπως η οθόνη ή ο εκτυπωτής) ή σε άλλον υπολογιστή*
- Σύστημα διασύνδεσης
  - *Ένας μηχανισμός που εξασφαλίζει την επικοινωνία του επεξεργαστή με τη μνήμη και τις μονάδες εισόδου - εξόδου.*

Βασικές κατηγορίες υπολογιστών

- Προσωπικός υπολογιστής (personal computer – PC)
  - *Ένας υπολογιστής σχεδιασμένος για χρήση από έναν ιδιώτη, που συνήθως περιλαμβάνει μια οθόνη, ένα πληκτρολόγιο και ένα ποντίκι*
  - *Σταθερός (desktop) ή φορητός (laptop)*
- Διακομιστής (server)
  - *Ένας υπολογιστής που χρησιμοποιείται για την εκτέλεση μεγαλύτερων προγραμμάτων για πολλούς χρήστες, συνήθως ταυτόχρονα, και ο οποίος τυπικά είναι προσπελάσιμος μόνο μέσω κάποιου δικτύου*
  - *Είναι η σύγχρονη μορφή αυτού που ήταν κάποτε τα μεγάλα υπολογιστικά συστήματα (mainframes)*
- Ενσωματωμένος υπολογιστής (embedded computer)
  - *Ένας υπολογιστής μέσα σε μια άλλη συσκευή, που χρησιμοποιείται για την εκτέλεση προκαθορισμένων εφαρμογών*
  - *Υπολογιστές σε ένα smartphone/tablet, σε ένα βιντεοπαιχνίδι, σε μια ψηφιακή τηλεόραση, ή σε μία οικιακή συσκευή*
  - *Δίκτυα επεξεργαστών που ελέγχουν ένα σύγχρονο αυτοκίνητο, ένα αεροπλάνο ή ένα εμπορικό πλοίο*
  - *Σήμερα, σχεδιάζονται κυρίως με τη χρήση πολλαπλών πυρήνων επεξεργαστών (processor cores) σε ένα τσιπ*

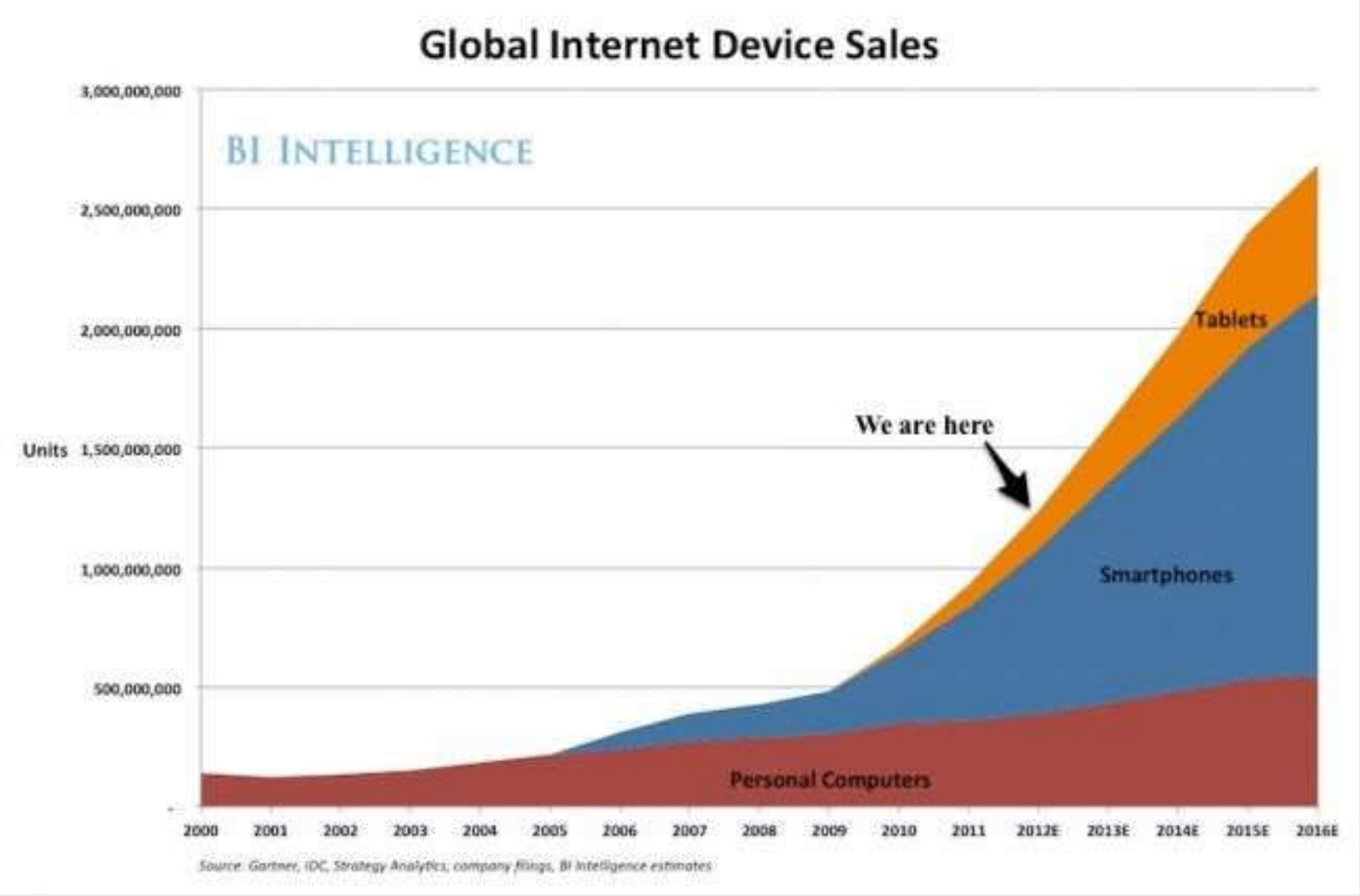
Πωλήσεις ανά κατηγορία υπολογιστών

- Στους προσωπικούς υπολογιστές υποχωρούν οι πωλήσεις των desktops έναντι των laptops και των tablets
  - *Πωλήσεις desktops (+ servers), laptops, tablets*



Πωλήσεις ανά κατηγορία υπολογιστών

- Κυριαρχούν οι ενσωματωμένοι υπολογιστές
  - *Πωλήσεις personal computers, smartphones, tablets*



Αρχιτεκτονική υπολογιστών (computer architecture)

- Προσδιορίζει τη θεώρηση ενός υπολογιστή από τη σκοπιά του προγραμματιστή με βάση την αρχιτεκτονική συνόλου εντολών (instruction set architecture) που είναι η διεπαφή υλικού και λογισμικού
  - *κατηγορίες εντολών – λειτουργιών*
  - *τελεστέοι εντολών (instruction operands)*
    - τα απαιτούμενα δεδομένα που χρησιμοποιούνται κατά την εκτέλεση μίας πράξης καθώς και το αποτέλεσμα της πράξης
    - τύποι και μεγέθη τελεστέων
      - *ακέραιοι των 8, 16, 32, 64 bit*
      - *πραγματικοί κινητής υποδιαστολής (απλής ακρίβειας των 32 bit ή διπλής ακρίβειας των 64 bit)*
      - *χαρακτήρας ASCII ή 2 BCD σε ένα byte (8 bit)*
    - που βρίσκονται αποθηκευμένοι
      - *στους καταχωρητές, στη μνήμη ή σε πεδίο της εντολής*
      - *τρόποι διευθυνσιοδότησης*
  - *μορφή και κωδικοποίηση εντολών*
- Παραδείγματα γνωστών αρχιτεκτονικών (συνόλου εντολών)
  - *ARM, x86, MIPS, SPARC, PowerPC, κλπ.*

Μικροαρχιτεκτονική (microarchitecture)

- Προσδιορίζει πως υλοποιείται η αρχιτεκτονική στο υλικό
  - *Σχεδίαση της διαδρομής δεδομένων (datapath) του επεξεργαστή*
    - χωροταξική διάταξη των καταχωρητών, των μνημών, των μονάδων ALU και των άλλων δομικών στοιχείων που απαρτίζουν τη διαδρομή δεδομένων
  - *Σχεδίαση της μονάδας ελέγχου (control unit) του επεξεργαστή*
    - παραγωγή των κατάλληλων σημάτων ελέγχου για τον χρονισμό του επεξεργαστή και των υπολοίπων διατάξεων
- Συχνά υπάρχουν πολλές διαφορετικές μικροαρχιτεκτονικές για την ίδια αρχιτεκτονική

Βασικές αρχιτεκτονικές συνόλου εντολών
Ανάλογα με το που βρίσκονται αποθηκευμένοι οι τελεστέοι

- Έμμεση αποθήκευση σε στοίβα (stack)
  - *οι τελεστέοι αποθηκεύονται έμμεσα στη στοίβα, πριν και μετά την εκτέλεση της πράξης*
  - *οι εντολές φόρτωσης από τη μνήμη στη στοίβα (PUSH) και αποθήκευσης από τη στοίβα στη μνήμη (POP) προσδιορίζουν:*
    - έναν τελεστέο που αποθηκεύεται άμεσα στη μνήμη και έναν τελεστέο που αποθηκεύεται έμμεσα στη στοίβα
  - *οι εντολές εκτέλεσης πράξεων δεν προσδιορίζουν κανένα τελεστέο*
    - και οι 3 τελεστέοι προσδιορίζονται έμμεσα στη στοίβα
    - Χαρακτηρίζονται ως εντολές μηδενικής διεύθυνσης

Κώδικας γλώσσας υψηλού επιπέδου

```
b = mem[B];
c = mem[C];
a = b + c;
Mem[A] = a;
```

Κώδικας συμβολικής γλώσσας

```
PUSH B  ; το B είναι διεύθυνση
PUSH C  ; το C είναι διεύθυνση
ADD     ; μηδενικής διεύθυνσης
POP A   ; το C είναι διεύθυνση
```

Βασικές αρχιτεκτονικές συνόλου εντολών
Ανάλογα με το που βρίσκονται αποθηκευμένοι οι τελεστέοι

- Έμμεση αποθήκευση σε συσσωρευτή (accumulator)
  - *ο ένας από τους δύο τελεστέους πριν την εκτέλεση της πράξης καθώς και το αποτέλεσμα της πράξης αποθηκεύονται έμμεσα στο συσσωρευτή που είναι ένας καταχωρητής ειδικού σκοπού*
  - *οι εντολές φόρτωσης από τη μνήμη στον συσσωρευτή (LOAD) και αποθήκευσης από τον συσσωρευτή στη μνήμη (STORE) προσδιορίζουν:*
    - έναν τελεστέο που αποθηκεύεται άμεσα στη μνήμη και έναν τελεστέο που αποθηκεύεται έμμεσα στον συσσωρευτή
  - *οι εντολές εκτέλεσης πράξεων προσδιορίζουν:*
    - έναν τελεστέο που αποθηκεύεται άμεσα στη μνήμη και έναν τελεστέο που αποθηκεύεται έμμεσα στον συσσωρευτή

Κώδικας γλώσσας υψηλού επιπέδου

```
b = mem[B];
c = mem[C];
a = b + c;
Mem[A] = a;
```

Κώδικας συμβολικής γλώσσας

```
LOAD B   ; AC = mem[B]
ADD C    ; AC = AC + mem[C]
STORE A  ; mem[A] = AC
```

Βασικές αρχιτεκτονικές συνόλου εντολών
Ανάλογα με το που βρίσκονται αποθηκευμένοι οι τελεστέοι

- Άμεση αποθήκευση σε καταχωρητή και μνήμη
  - *ο ένας από τους δύο τελεστέους πριν την εκτέλεση της πράξης καθώς και το αποτέλεσμα της πράξης αποθηκεύονται άμεσα σε έναν καταχωρητή (έστω R1) του αρχείου καταχωρητών*
  - *οι εντολές φόρτωσης από τη μνήμη στο αρχείο καταχωρητών (LOAD) και αποθήκευσης από το αρχείο καταχωρητών στη μνήμη (STORE) προσδιορίζουν:*
    - έναν τελεστέο που αποθηκεύεται άμεσα στη μνήμη και έναν τελεστέο που αποθηκεύεται άμεσα σε έναν καταχωρητή
  - *οι εντολές εκτέλεσης πράξεων προσδιορίζουν:*
    - έναν τελεστέο που αποθηκεύεται άμεσα στη μνήμη και έναν τελεστέο που αποθηκεύεται άμεσα σε έναν καταχωρητή

Κώδικας γλώσσας υψηλού επιπέδου

`b = mem[B];`
`c = mem[C];`
`a = b + c;`
`Mem[A] = a;`

Κώδικας συμβολικής γλώσσας

`LOAD R1,B ; R1 = mem[B]`
`ADD R1,C ; R1 = R1 + mem[C]`
`STORE A,R1; mem[A] = R1`

Βασικές αρχιτεκτονικές συνόλου εντολών

Ανάλογα με το που βρίσκονται αποθηκευμένοι οι τελεστέοι

- Άμεση αποθήκευση σε πολλούς (2 ή 3) καταχωρητές
  - *οι εντολές φόρτωσης (LOAD) και αποθήκευσης (STORE)* ***διαχωρίζονται*** *από τις εντολές εκτέλεσης πράξεων που χρησιμοποιούν* ***αποκλειστικά*** *το αρχείο καταχωρητών*
  - *οι εντολές LOAD και STORE προσδιορίζουν:*
    - έναν τελεστέο που αποθηκεύεται άμεσα στη μνήμη και έναν τελεστέο που αποθηκεύεται άμεσα σε έναν καταχωρητή
  - *οι εντολές εκτέλεσης πράξεων προσδιορίζουν:*
    - τρεις τελεστέους που αποθηκεύονται άμεσα σε 2 (ή 3) καταχωρητές του αρχείου καταχωρητών
    - στην περίπτωση των 2 καταχωρητών ο ένας από τους δύο τελεστέους πριν την εκτέλεση της πράξης καθώς και το αποτέλεσμα της πράξης αποθηκεύονται άμεσα στον ίδιο καταχωρητή, έστω R1 (**`ADD R1,R2`**)

Κώδικας γλώσσας υψηλού επιπέδου

```
b = mem[B];
c = mem[C];
a = b + c;
Mem[A] = a;
```

Κώδικας συμβολικής γλώσσας

```
LOAD R1,B     ; R1 = mem[B]
LOAD R2,C     ; R2 = mem[C]
ADD R3,R1,R2; R3 = R1 + R2
STORE A,R3    ; mem[A] = R3
```

Βασικές αρχιτεκτονικές συνόλου εντολών
Ανάλογα με το που βρίσκονται αποθηκευμένοι οι τελεστέοι

- Συγκρίσεις
  - *Οι βασικές αρχιτεκτονικές συνόλου εντολών έμμεσης αποθήκευσης είτε σε στοίβα, είτε σε συσσωρευτή στόχευαν στη μείωση του μεγέθους ενός προγράμματος σε γλώσσα μηχανής*
    - εμφανίσθηκαν πριν την έλευση των μικροεπεξεργαστών και των διατάξεων μνήμης σε ολοκληρωμένα κυκλώματα (RAM, ROM), όταν οι ολοκληρώσεις ήταν μικρές (SSI και MSI κυκλώματα)
  - *Οι βασικές αρχιτεκτονικές συνόλου εντολών άμεσης αποθήκευσης είτε σε καταχωρητή και μνήμη, είτε σε πολλούς καταχωρητές στόχευαν στη χρήση του αρχείου καταχωρητών, που αυξάνει την απόδοση του υπολογιστή και παρέχει ευελιξία*
    - η άμεση αποθήκευση σε καταχωρητή και μνήμη εμφανίσθηκε με την έλευση των μικροεπεξεργαστών (ως υπολογιστές CISC),
    - η άμεση αποθήκευση σε πολλούς καταχωρητές εμφανίσθηκε όταν το επέτρεψε η τεχνολογική εξέλιξη στις διατάξεις μνήμης (ως υπολογιστές RISC)

Επιλεγμένες ασκήσεις

- Ποια είναι η ακολουθία των εντολών σε συμβολική γλώσσα που υπολογίζουν τις κάτωθι αριθμητικές παραστάσεις για όλες τις βασικές αρχιτεκτονικές συνόλου εντολών:
  - $Y = A + [10 \times (B - C)]$
  - $Y = (A+10) \times (B-C)^2$

  όπου τα Y, A, B, C είναι διευθύνσεις μνήμης

Υπολογιστές CISC vs RISC

- Οι υπολογιστές CISC (complex instruction set computer) υλοποιούν απλές και σύνθετες εντολές (π.χ. x86)
  - *Το απαιτούμενο υλικό για την αποκωδικοποίηση και εκτέλεση των εντολών είναι ιδιαίτερα σύνθετο, μεγάλο και αργό ακόμα και για τις απλές εντολές*
- Οι υπολογιστές RISC (reduced instruction set computer) υλοποιούν μόνο απλές εντολές
  - *εμφανίσθηκαν στην αρχή της δεκαετίας του 1980 και βασίζονται στην παρατήρηση ότι η μεγάλη πλειοψηφία των εντολών που εμφανίζονται συχνά σε ένα πρόγραμμα είναι απλές εντολές*
  - *το απαιτούμενο υλικό για την αποκωδικοποίηση και εκτέλεση των εντολών διατηρείται απλό, μικρό και γρήγορο*
    - Μια εντολή σε έναν υπολογιστή CISC απαιτεί πολλές, ακόμα και εκατοντάδες, απλές εντολές σε έναν υπολογιστή RISC
    - Τα προγράμματα των υπολογιστών RISC έχουν περισσότερες εντολές και καταλαμβάνουν περισσότερο χώρο στη μνήμη

Υπολογιστές CISC vs RISC

- Η μεγάλη πλειοψηφία των εντολών (>90%) που εμφανίζονται συχνά σε ένα πρόγραμμα είναι απλές εντολές
  - *Οι 10 πιο συχνές απλές εντολές της αρχιτεκτονικής x86 που κυριαρχούν σε 5 προγράμματα SPECint92*

Σειρά	Εντολή	Ποσοστό % των συνολικά εκτελουμένων
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Άθροισμα	96%

Αρχιτεκτονική ARM

- Η αρχιτεκτονική ARM αναπτύχθηκε αρχικά στη δεκαετία του 1980 από την εταιρεία Acorn Computer Group, η οποία αργότερα δημιούργησε την εταιρεία Advanced RISC Machines Ltd (ARM)
- Η ARM δίνει άδεια σε άλλες εταιρείες να κατασκευάζουν τους επεξεργαστές της, συχνά ως πυρήνες επεξεργαστή για ενσωματωμένους υπολογιστές
  - *Κάθε χρόνο πωλούνται πάνω από 10 δισεκατομμύρια επεξεργαστές ARM*
  - *Σχεδόν όλα τα smartphones και τα tablets έχουν πολλαπλούς επεξεργαστές ARM*
  - *Άνω του 75% των ανθρώπων χρησιμοποιούν προϊόντα με επεξεργαστές ARM*

Αρχιτεκτονική ARM

- Η «*αρχιτεκτονική ARM*» που θα περιγράψουμε είναι η έκδοση 4 της αρχιτεκτονικής συνόλου εντολών της ARM (ARMv4).
  - *περιοριζόμαστε στις συνήθεις απλές εντολές της αρχιτεκτονικής συνόλου εντολών της ARM*
  - *διευθυνσιοδότηση της μνήμης στα 32 bit*
  - *λέξεις εντολών και δεδομένων μεγέθους 32 bit*
  - *αρχείο 16 καταχωρητών μεγέθους 32 bit*
    - ο program counter (PC) συμπεριλαμβάνεται ως R15
- Το Εγχειρίδιο Αναφοράς Αρχιτεκτονικής της ARM (*Architecture Reference Manual*, *ARM*), ορίζει επίσημα την αρχιτεκτονική
- Οι λόγοι που επιλέξαμε να εστιάσουμε στην αρχιτεκτονική ARM είναι οι εξής:
  - *αφενός είναι από τις πιο επιτυχημένες εμπορικά αρχιτεκτονικές συνόλου εντολών*
  - *αφετέρου η αρχιτεκτονική της είναι «καθαρή», με ελάχιστα εκκεντρικά χαρακτηριστικά*

Αρχές σχεδίασης αρχιτεκτονικής

1. Η κανονικότητα υποστηρίζει την απλότητα
   - *Η αρχιτεκτονική ARM χρησιμοποιεί εντολές σταθερού μεγέθους των 32 bit, ακόμα και όταν δεν χρησιμοποιούνται όλα τα bit*
2. Οτιδήποτε είναι σύνηθες και χρησιμοποιείται συχνά πρέπει να είναι γρήγορο
   - *Το πλήθος των εντολών της αρχιτεκτονικής ARM διατηρείται εσκεμμένα μικρό (ορίζεται υπολογιστής RISC), έτσι ώστε:*
     - να καλύπτει τις συνήθεις και συχνά εμφανιζόμενες εντολές σε ένα πρόγραμμα, και
     - το απαιτούμενο υλικό να μπορεί να είναι απλό, μικρό και γρήγορο
3. Το μικρότερο μέγεθος του αποθηκευτικού μέσου συνεπάγεται πιο γρήγορη εκτέλεση της εντολής
   - *Η αρχιτεκτονική ARM χρησιμοποιεί αρχείο καταχωρητών με 16 καταχωρητές μεγέθους 32 bit*
4. Η καλή σχεδίαση απαιτεί και καλούς συμβιβασμούς
   - *Η αρχιτεκτονική ARM ορίζει μόνο τρεις κύριες μορφές εντολών: επεξεργασίας δεδομένων, μνήμης, και διακλάδωσης*

Συμβολική γλώσσα – Η εντολή ADD

- Η πιο συνηθισμένη πράξη που εκτελούν οι υπολογιστές είναι η πρόσθεση (addition)

Κώδικας γλώσσας υψηλού επιπέδου
`a = b + c;`

Κώδικας συμβολικής γλώσσας της ARM
`ADD a, b, c`

- Η εντολή σε συμβολική γλώσσα γράφεται σε μία σειρά.
- Το πρώτο τμήμα της εντολής σε συμβολική γλώσσα, το ADD, ονομάζεται μνημονικό (*mnemonic*) και υποδεικνύει την πράξη που θα εκτελεστεί
- Η πράξη εκτελείται με τα b και c που ονομάζονται τελεστέοι προέλευσης (*source operands*) και αποθηκεύονται σε καταχωρητές προέλευσης
- Το αποτέλεσμα εγγράφεται στο a, που ονομάζεται τελεστέος προορισμού (*destination operand*) και αποθηκεύεται σε καταχωρητή προορισμού

Συμβολική γλώσσα – Η εντολή SUB

- Η πράξη της αφαίρεσης (subtraction) είναι παρόμοια με την πράξη της πρόσθεσης

Κώδικας γλώσσας υψηλού επιπέδου
`a = b - c;`

Κώδικας συμβολικής γλώσσας της ARM
`SUB a, b, c`

- Στην αφαίρεση το μνημονικό είναι το SUB
- Το πλήθος των τελεστέων παραμένει ίδιο:
  - *δύο τελεστέοι προέλευσης (source operands) **b**, **c** που αποθηκεύονται σε καταχωρητές προέλευσης*
  - *ένας τελεστέος προορισμού (destination operand) **a** που αποθηκεύεται σε καταχωρητή προορισμού*

Συμβολική γλώσσα – Σχόλια

- Στη συμβολική γλώσσα ARM χρησιμοποιούνται μόνο σχόλια μίας γραμμής
- Τα σχόλια ξεκινούν με ένα ελληνικό ερωτηματικό (;) και συνεχίζονται έως το τέλος της γραμμής

Κώδικας γλώσσας υψηλού επιπέδου

```
a = b + c;     // σχόλιο μίας γραμμής
/* σχόλιο πολλών γραμμών */
```

Κώδικας συμβολικής γλώσσας της ARM

```
ADD a, b, c    ; a = b + c
```

Συμβολική γλώσσα – Πιο σύνθετη εντολή

- Παράδειγμα εκτέλεσης σύνθετης εντολής υψηλού επιπέδου με πολλές εντολές της συμβολικής γλώσσας της αρχιτεκτονικής ARM

Κώδικας γλώσσας υψηλού επιπέδου

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

Κώδικας συμβολικής γλώσσας της ARM

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

Συμβολική γλώσσα – Τελεστέοι

- Στις εντολές προσδιορίζονται οι τελεστέοι ως τα απαιτούμενα δεδομένα που χρησιμοποιούνται κατά την εκτέλεση μίας πράξης καθώς και το αποτέλεσμα της πράξης
- Οι τελεστέοι μπορεί να είναι είτε μεταβλητές που αποθηκεύονται στο αρχείο των 16 καταχωρητών μεγέθους 32 bit ή στη μνήμη, είτε σταθερές που αποθηκεύονται σε κάποιο πεδίο της εντολής
  - *οι σταθερές στην ίδια την εντολή και οι μεταβλητές που αποθηκεύονται στο αρχείο καταχωρητών προσπελάζονται γρήγορα, αλλά διαθέτουν μικρή χωρητικότητα δεδομένων*
  - *Οι μεταβλητές που αποθηκεύονται στη μνήμη προσπελάζονται αργά, αλλά διαθέτουν μεγάλη χωρητικότητα δεδομένων*
  - *Ο χρόνος εκτέλεσης της εντολής εξαρτάται από τον χρόνο προσπέλασης και τη χωρητικότητα δεδομένων*
    - Το ιεραρχικό σύστημα μνήμης με αρχείο καταχωρητών και κρυφές μνήμες επιπέδων L1, L2 και ενδεχομένως L3, τεχνολογίας SRAM, καθώς και κύρια μνήμη τεχνολογίας DRAM στοχεύει στη βελτιστοποίηση του χρόνου προσπέλασης και της χωρητικότητας δεδομένων με κατάλληλους μηχανισμούς μεταφοράς δεδομένων

Συμβολική γλώσσα – Καταχωρητές ως τελεστέοι

- Παράδειγμα της εντολής ADD με καταχωρητές ως τελεστέους:
  - *Τα ονόματα των καταχωρητών στην αρχιτεκτονική ARM ξεκινούν με το αγγλικό γράμμα 'R' (Register)*
  - *Οι μεταβλητές a, b και c τοποθετούνται αυθαίρετα στους καταχωρητές R0, R1 και R2 του αρχείου καταχωρητών*

Κώδικας γλώσσας υψηλού επιπέδου

```
a = b + c;
```

Κώδικας συμβολικής γλώσσας της ARM

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

Συμβολική γλώσσα – Προσωρινοί καταχωρητές

- Παράδειγμα εκτέλεσης εντολών με αποθήκευση αποτελέσματος ενδιάμεσου υπολογισμού
  - *Το αποτέλεσμα της πράξης b + c αποθηκεύεται προσωρινά στον καταχωρητή R4 του αρχείου καταχωρητών*

Κώδικας γλώσσας υψηλού επιπέδου

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

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = a, R1 = b, R2 = c, R3 = d, R4 = t,
ADD R4, R1, R2  ; t = b + c
SUB R0, R4, R3  ; a = t - d
```

Συμβολική γλώσσα – Καταχωρητές ως τελεστέοι

- Παράδειγμα μετάφρασης κώδικα υψηλού επιπέδου σε συμβολική γλώσσα ARM
  - *Χρησιμοποιούνται δύο επιπλέον καταχωρητές R8 και R9 του αρχείου καταχωρητών για αποθήκευση αποτελεσμάτων ενδιάμεσων υπολογισμών*

Κώδικας γλώσσας υψηλού επιπέδου

```
a = b – c;
f = (g + h) – (i + j);
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = a, R1 = b, R2 = c, R3 = f
; R4 = g, R5 = h, R6 = i, R7 = j
SUB R0, R1, R2 ; a = b – c
ADD R8, R4, R5 ; R8 = g + h
ADD R9, R6, R7 ; R9 = i + j
SUB R3, R8, R9 ; f = (g + h) – (i + j)
```

Αρχείο καταχωρητών

- 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)

- Οι καταχωρητές R0–R12 χρησιμοποιούνται ως καταχωρητές δεδομένων για την αποθήκευση μεταβλητών
  - *Οι καταχωρητές R0–R3 έχουν και άλλες ειδικές χρήσεις κατά τη διάρκεια κλήσεων συναρτήσεων/διαδικασιών*
- Ο program counter (PC) συμπεριλαμβάνεται στους καταχωρητές του αρχείου καταχωρητών ως καταχωρητής R15

Συμβολική γλώσσα – Άμεσοι τελεστέοι

- Εκτός από πράξεις με καταχωρητές, οι εντολές ARM μπορούν να χρησιμοποιούν σταθερές ή *άμεσους* τελεστέους
- Αυτές οι σταθερές ονομάζονται άμεσοι τελεστέοι επειδή οι τιμές τους είναι άμεσα αποθηκευμένες στην ίδια την εντολή και δεν απαιτούν την προσπέλαση κάποιου καταχωρητή ή μνήμης
- Επιτρέπεται η χρήση ενός μόνο άμεσου τελεστέου στην εντολή
- Το αποτέλεσμα της πράξης αποθηκεύεται πάντα σε καταχωρητή
- Στη συμβολική γλώσσα, ένας άμεσος τελεστέος ξεκινάει με το σύμβολο # και μπορεί να γράφεται σε δεκαδική μορφή ως έχει ή σε δεκαεξαδική μορφή με πρόθεμα το 0x.
- Οι άμεσοι τελεστέοι είναι μη προσημασμένοι αριθμοί των 8 έως 12 bit και έχουν μία ιδιαίτερη κωδικοποίηση στην αρχιτεκτονική ARM

Συμβολική γλώσσα – Άμεσοι τελεστέοι

- Παραδείγματα των εντολών ADD και SUB με άμεσους τελεστέους
  - *Οι μεταβλητές a και b τοποθετούνται αυθαίρετα στους καταχωρητές R7 και R8 του αρχείου καταχωρητών*

Κώδικας γλώσσας υψηλού επιπέδου

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

Κώδικας συμβολικής γλώσσας της ARM

```
; R7 = a, R8 = b
ADD R7, R7, #4     ; a = a + 4
SUB R8, R7, #0xC   ; b = a - 12
```

Συμβολική γλώσσα – Η εντολή MOV

- Η εντολή μετακίνησης (MOV) αποτελεί έναν χρήσιμο τρόπο για τον καθορισμό των αρχικών τιμών των καταχωρητών.
- Η εντολή MOV δέχεται έναν τελεστέο προέλευσης
  - *σταθερά ως άμεσο τελεστέο*
  - *μεταβλητή ως τελεστέο που αποθηκεύεται σε καταχωρητή προέλευσης*
- Η εντολή MOV δέχεται έναν τελεστέο προορισμού
  - *μεταβλητή ως τελεστέο που αποθηκεύεται σε καταχωρητή προορισμού*

Κώδικας γλώσσας υψηλού επιπέδου

```
i = 0;
x = 4080;
y = x;
```

Κώδικας συμβολικής γλώσσας της ARM

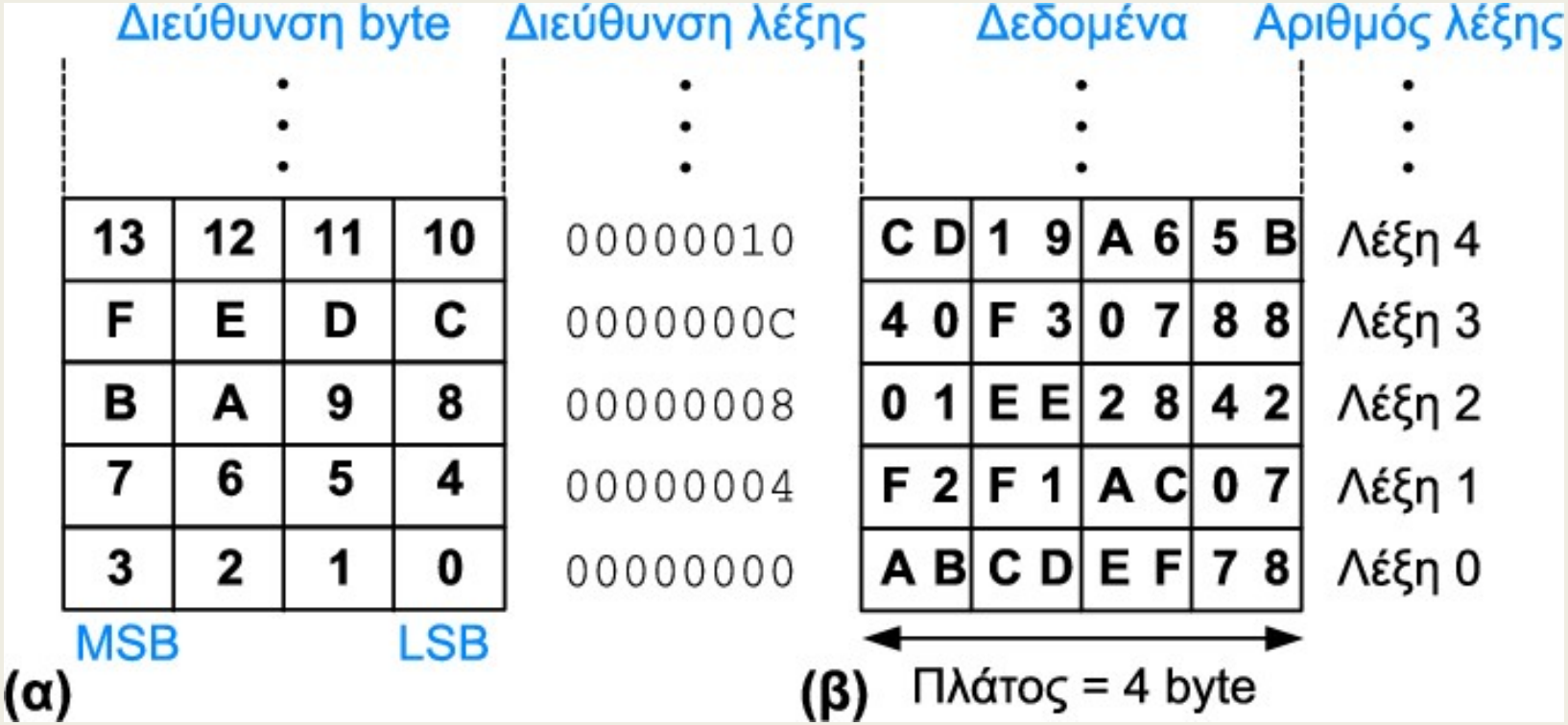
```
; R4 = i, R5 = x, R6 = y
MOV R4, #0       ; i = 0
MOV R5, #0xFF0   ; x = 4080
MOV R6, R5       ; y = x
```

Διαχείριση μνήμης στην αρχιτεκτονική ARM

- Εκτός από τους καταχωρητές, τα δεδομένα μπορούν επίσης να αποθηκεύονται στη μνήμη
  - *η μνήμη είναι πολύ πιο μεγάλη, αλλά και πιο αργή*
- Στην αρχιτεκτονική ARM, οι εντολές εκτελούνται αποκλειστικά με καταχωρητές
  - *τα δεδομένα που είναι αποθηκευμένα στη μνήμη πρέπει να μετακινηθούν πρώτα σε έναν καταχωρητή, ώστε να είναι εφικτή η επεξεργασία τους*
- Η αρχιτεκτονική ARM χρησιμοποιεί το μέγεθος των 32 bit τόσο για τις διευθύνσεις μνήμης όσο και για τις λέξεις δεδομένων
- Στην αρχιτεκτονική ARM η μνήμη είναι προσπελάσιμη κατά byte (byte addressable), με ευθυγραμμισμένες διευθύνσεις, όπου διευθυνσιοδοτείται το μικρό άκρο της λέξης δεδομένων (little endian)
  - *Μια λέξη δεδομένων μεγέθους 32 bit αποτελείται από 4 byte, άρα η διεύθυνση κάθε λέξης είναι ευθυγραμμισμένη σε πολλαπλάσια του 4*
  - *Η διεύθυνση της λέξης δεδομένων ταυτίζεται με τη διεύθυνση του λιγότερου σημαντικού byte (least significant byte – LSB) της λέξης δεδομένων (διευθυνσιοδότηση μικρού άκρου)*

Διαχείριση μνήμης στην αρχιτεκτονική ARM

- Στην αρχιτεκτονική ARM η μνήμη είναι προσπελάσιμη κατά byte με λέξεις δεδομένων μεγέθους 32 bit (4 byte)
- Το *περισσότερο* σημαντικό byte (*most significant byte, MSB*) της λέξης βρίσκεται στα αριστερά, ενώ το λιγότερο σημαντικό byte (*least significant byte, LSB*) της λέξης βρίσκεται στα δεξιά
- Η διεύθυνση της λέξης ταυτίζεται με τη διεύθυνση του LSB της λέξης και είναι ευθυγραμμισμένη (σε πολλαπλάσια του 4)



(a) διευθύνσεις byte και λέξης και (β) δεδομένα μνήμης

Συμβολική γλώσσα – Εντολές μνήμης

- Για τον προσδιορισμό της διεύθυνσης της λέξης δεδομένων στη μνήμη χρησιμοποιείται ο τρόπος διευθυνσιοδότησης μνήμης με σχετική απόσταση (offset addressing) που χρησιμοποιεί:
  - *έναν καταχωρητή βάσης (base register) που περιέχει τη διεύθυνση βάσης, και*
  - *μια σχετική απόσταση (offset) ως μη προσημασμένος ακέραιος των 12 bit που είναι άμεσα αποθηκευμένος στην ίδια την εντολή*
  - *για να σχηματιστεί η διεύθυνση μνήμης, προσθέτουμε ή αφαιρούμε τη σχετική απόσταση (η οποία μπορεί να είναι 0) στη διεύθυνση που είναι αποθηκευμένη στον καταχωρητή βάσης*
  - *Η διεύθυνση της λέξης δεδομένων είναι ευθυγραμμισμένη (πολλαπλάσια του 4 – λήγει σε 0, 4, 8, C), ώστε να υλοποιείται εύκολα στο υλικό η μεταφορά δεδομένων από και προς τη μνήμη*

Συμβολική γλώσσα – Η εντολή LDR

- Η αρχιτεκτονική ARM ορίζει την εντολή φόρτωσης καταχωρητή (*load register, LDR*) με την οποία μια λέξη δεδομένων διαβάζεται από τη μνήμη και φορτώνεται σε έναν καταχωρητή του αρχείου καταχωρητών
  - *Μετά την εκτέλεση της εντολής LDR, στον καταχωρητή R7 υπάρχει η τιμή που είναι αποθηκευμένη στη διεύθυνση 8 της μνήμης*

Κώδικας γλώσσας υψηλού επιπέδου

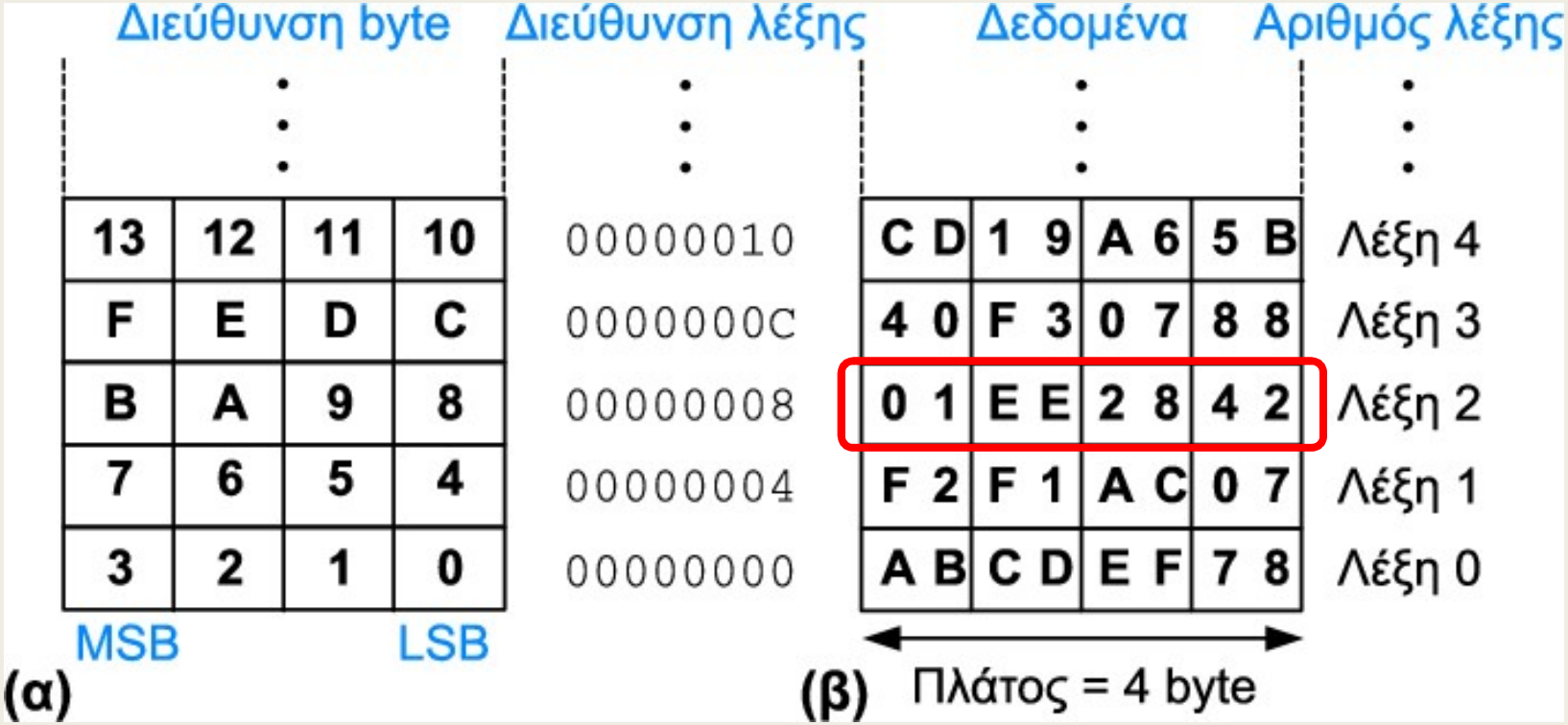
```
a = mem[2];
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R5 = base register, R7 = a
MOV R5, #0          ; R5 = 0
LDR R7, [R5, #8]    ; R7 = δεδομένα από τη
                    ; διεύθυνση μνήμης R5+8
```

Συμβολική γλώσσα – Η εντολή LDR

- Η αρχιτεκτονική ARM ορίζει την εντολή φόρτωσης καταχωρητή (*load register, LDR*) με την οποία μια λέξη δεδομένων διαβάζεται από τη μνήμη και φορτώνεται σε έναν καταχωρητή του αρχείου καταχωρητών
  - *Μετά την εκτέλεση της εντολής LDR, στον καταχωρητή R7 υπάρχει η τιμή που είναι αποθηκευμένη στη διεύθυνση 8 της μνήμης*
  - R7 = 0x01EE2842



Συμβολική γλώσσα – Η εντολή STR

- Η αρχιτεκτονική ARM ορίζει την εντολή αποθήκευσης καταχωρητή (*store register, STR*) με την οποία μια λέξη δεδομένων που είναι αποθηκευμένη σε έναν καταχωρητή του αρχείου καταχωρητών εγγράφεται στη μνήμη
  - *Μετά την εκτέλεση της εντολής STR, στη διεύθυνση 16 της μνήμης υπάρχει η τιμή που είναι αποθηκευμένη στον καταχωρητή R9 = 255*

Κώδικας γλώσσας υψηλού επιπέδου

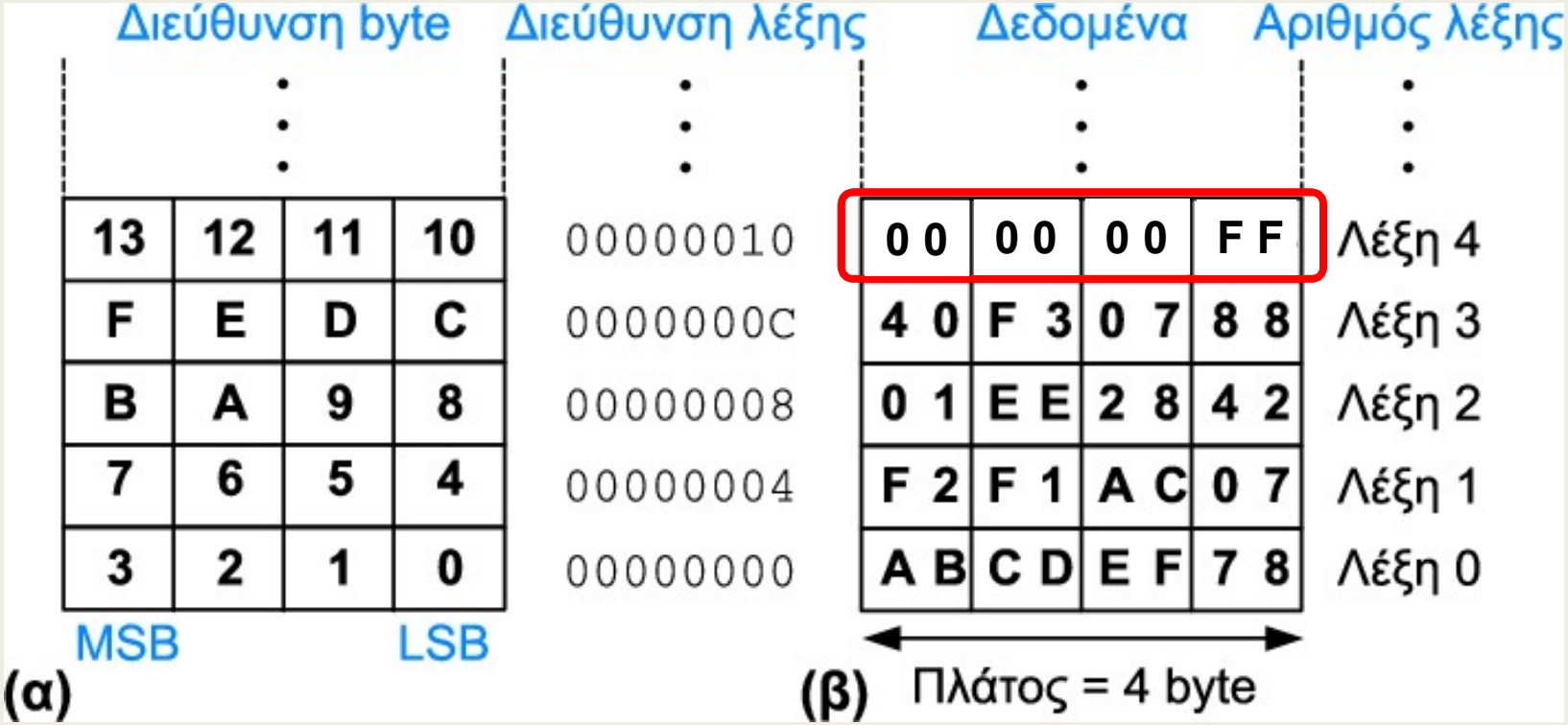
```
mem[4] = 255;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R5 = base register
MOV R5, #0              ; R5 = 0
MOV R9, #0x000000FF     ; R9 = 255
STR R9, [R5, #0x10]     ; στη διεύθυνση μνήμης
                        ; R5+16 αποθηκεύεται
                        ; το 0x000000FF
```

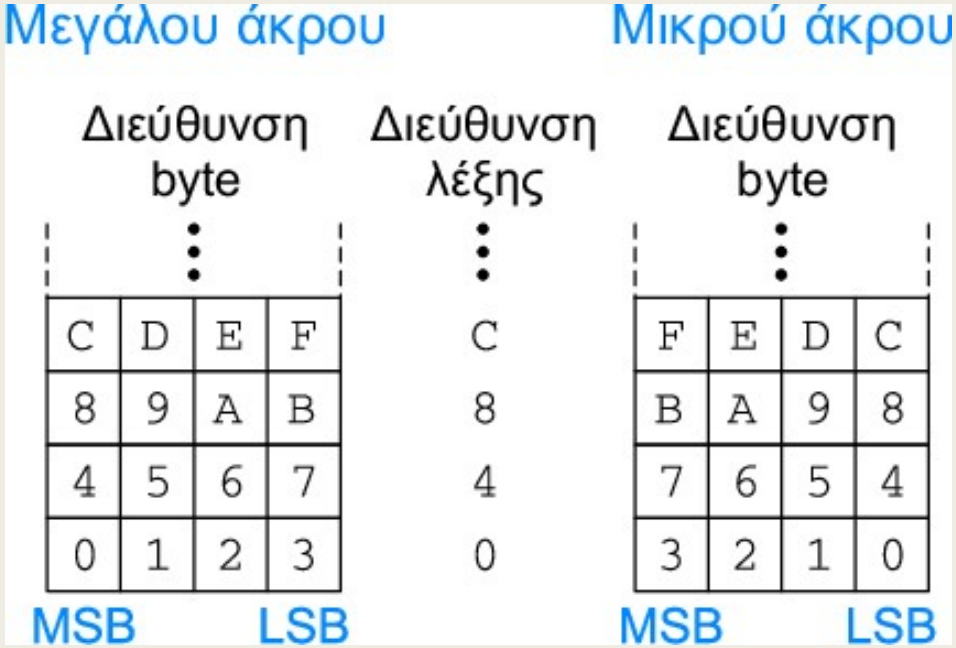
Συμβολική γλώσσα – Η εντολή STR

- Η αρχιτεκτονική ARM ορίζει την εντολή αποθήκευσης καταχωρητή (*store register, STR*) με την οποία μια λέξη δεδομένων που είναι αποθηκευμένη σε έναν καταχωρητή του αρχείου καταχωρητών εγγράφεται στη μνήμη
  - *Μετά την εκτέλεση της εντολής STR, στη διεύθυνση 16 της μνήμης υπάρχει η τιμή που είναι αποθηκευμένη στον καταχωρητή R9 = 255*
  - *Mem[4] = 0x000000FF*



Διευθυνσιοδότηση μικρού και μεγάλου άκρου

- Οι προσπελάσιμες κατά byte μνήμες είναι οργανωμένες με βάση τη μορφή μεγάλου άκρου (*big-endian*) ή μικρού άκρου (*little-endian*)
  - *Στους υπολογιστές μεγάλου άκρου, η αρίθμηση των byte ξεκινάει από το 0 στο MSB της λέξης, δηλαδή στο μεγάλο (περισσότερο σημαντικό) άκρο*
  - *Στους υπολογιστές μικρού άκρου, η αρίθμηση των byte ξεκινάει από το 0 στο LSB της λέξης, δηλαδή στο μικρό (λιγότερο σημαντικό) άκρο*
- Η αρχιτεκτονική ARM υιοθετεί τη διευθυνσιοδότηση μικρού άκρου αλλά υπάρχουν εκδόσεις της που υποστηρίζουν διευθυνσιοδότηση δεδομένων διπλού άκρου (bi-endian), η οποία επιτρέπει φορτώσεις και αποθηκεύσεις δεδομένων σε οποιαδήποτε από τις δύο μορφές



Συμβολική γλώσσα – Λογικές εντολές

- Η αρχιτεκτονική ARM ορίζει τις ακόλουθες λογικές εντολές για την εκτέλεση *λογικών πράξεων* σε επίπεδο bit (*bitwise*):
  - *AND (AND), ORR (OR), EOR (XOR)*
  - *BIC (bit clear), επαναφέρει στο 0 εκείνα τα bit του πρώτου τελεστέου προέλευσης που έχουν την τιμή 1 στον δεύτερο τελεστέο προέλευσης*
    - Εάν ο πρώτος τελεστέος είναι στον R1 και ο δεύτερος τελεστέος στον R2 εκτελεί την πράξη R1 AND (NOT R2)
    - Χρησιμοποιείται για την εφαρμογή μάσκας (masking) στα bit (δηλαδή την αναγκαστική ανάθεση της τιμής 0 στα μη επιθυμητά bit).
  - *MVN (MoVe και Not), αντιστρέφει τα bit του δεύτερου τελεστέου προέλευσης*
- Για την εκτέλεση της λογικής πράξης χρησιμοποιούνται δύο τελεστέοι προέλευσης:
  - *ο πρώτος τελεστέος προέλευσης είναι πάντα αποθηκευμένος σε έναν καταχωρητή προέλευσης (δεν χρησιμοποιείται στην MVN)*
  - *ο δεύτερος τελεστέος προέλευσης μπορεί να είναι είτε άμεσος τελεστέος, είτε τελεστέος αποθηκευμένος σε έναν καταχωρητή προέλευσης*
- Μετά την εκτέλεση της λογικής πράξης προκύπτει ένας τελεστέος προορισμού, που αποθηκεύεται σε έναν καταχωρητή προορισμού

Συμβολική γλώσσα – 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		Αποτέλεσμα			
`AND R3, R1, R2`	R3	0100 0110	1010 0001	0000 0000	0000 0000
`ORR R4, R1, R2`	R4	1111 1111	1111 1111	1111 0001	1011 0111
`EOR R5, R1, R2`	R5	1011 1001	0101 1110	1111 0001	1011 0111
`BIC R6, R1, R2`	R6	0000 0000	0000 0000	1111 0001	1011 0111
`MVN R7, R2`	R7	0000 0000	0000 0000	1111 1111	1111 1111

- Η εντολή **`BIC R6, R1, R2`** είναι ισοδύναμη με τις δύο εντολές:

```
MVN R7, R2
AND R6, R1, R7
```

Συμβολική γλώσσα – Εντολές Ολίσθησης

- Η αρχιτεκτονική ARM ορίζει τις ακόλουθες εντολές ολίσθησης:
  - *LSL (logical shift left, λογική ολίσθηση προς τα αριστερά)*
    - Οι αριστερές ολισθήσεις γεμίζουν τα λιγότερο σημαντικά κενά bit με μηδενικά
  - *LSR (logical shift right, λογική ολίσθηση προς τα δεξιά)*
    - Οι δεξιές λογικές ολισθήσεις γεμίζουν τα περισσότερο σημαντικά κενά bit με μηδενικά
  - *ASR (arithmetic shift right, αριθμητική ολίσθηση προς τα δεξιά)*
    - Οι δεξιές αριθμητικές ολισθήσεις γεμίζουν τα περισσότερο σημαντικά κενά bit με το bit του προσήμου
  - *ROR (rotate right, περιστροφή προς τα δεξιά)*
- Για την εκτέλεση της εντολής ολίσθησης με σταθερή ποσότητα ολίσθησης χρησιμοποιούνται δύο τελεστέοι προέλευσης:
  - *ο πρώτος τελεστέος προέλευσης του οποίου το περιεχόμενο ολισθαίνει είναι πάντα αποθηκευμένος σε έναν καταχωρητή προέλευσης*
  - *ο δεύτερος τελεστέος προέλευσης ορίζει τη σταθερή ποσότητα ολίσθησης και είναι άμεσος τελεστέος με τιμές από 0 μέχρι 31*
- Μετά την εκτέλεση της εντολής ολίσθησης προκύπτει ένας τελεστέος προορισμού, που αποθηκεύεται σε έναν καταχωρητή προορισμού

Συμβολική γλώσσα – Εντολές Ολίσθησης

- Παραδείγματα:

		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

- Το περιεχόμενο του πρώτου καταχωρητή προέλευσης R5 ολισθαίνει κατά την άμεση ποσότητα ολίσθησης και το αποτέλεσμα τοποθετείται στον καταχωρητή προορισμού
- Η ολίσθηση μιας τιμής προς τα αριστερά (δεξιά) κατά N bit είναι ισοδύναμη με τον πολλαπλασιασμό (διαίρεση) της τιμής με το 2^N
- Οι λογικές ολισθήσεις χρησιμοποιούνται επίσης για την εξαγωγή ή τη συνένωση πεδίων bit.

Σημαίες συνθήκης

- Η αρχιτεκτονική ARM υποστηρίζει εντολές που ενεργοποιούν σημαίες συνθήκης (*condition flags*) ανάλογα με το αποτέλεσμα μίας πράξης στη μονάδα ALU

Σημαία	Περιγραφή
N	Αρνητικό αποτέλεσμα (Negative)
Z	Μηδενικό αποτέλεσμα (Zero)
C	Η πράξη παρήγαγε κρατούμενο εξόδου (Carry)
V	Η πράξη υπερχείλισε (oVerflowed)

- Οι εντολές, που έπονται της εντολής που ενεργοποιεί σημαίες συνθήκης, εκτελούνται υπό συνθήκη, ανάλογα με την κατάσταση των σημαιών
- Οι σημαίες συνθήκης αποθηκεύονται στα 4 περισσότερο σημαντικά bit του Καταχωρητή Κατάστασης Τρέχοντος Προγράμματος (*Current Program Status Register - CPSR*) των 32bit

Συμβολική γλώσσα – Η εντολή CMP

- Ο πιο συνήθης τρόπος για την ενεργοποίηση σημαιών συνθήκης είναι με τη χρήση της εντολής σύγκρισης (compare, CMP) της αρχιτεκτονικής ARM
- Η CMP αφαιρεί τον δεύτερο τελεστέο προέλευσης από τον πρώτο τελεστέο προέλευσης και ενεργοποιεί τις σημαίες συνθήκης με βάση το αποτέλεσμα της πράξης
  - *Η εντολή CMP ενημερώνει μόνο τις σημαίες χωρίς να αποθηκεύει το αποτέλεσμα της αφαίρεσης σε έναν καταχωρητή προορισμού*
- Για την εκτέλεση της λογικής πράξης χρησιμοποιούνται δύο τελεστέοι προέλευσης:
  - *ο πρώτος τελεστέος προέλευσης είναι πάντα αποθηκευμένος σε έναν καταχωρητή προέλευσης*
  - *ο δεύτερος τελεστέος προέλευσης μπορεί να είναι είτε άμεσος τελεστέος, είτε τελεστέος αποθηκευμένος σε έναν καταχωρητή προέλευσης*

Συμβολική γλώσσα – Εντολές υπό συνθήκη

- Οι εντολές, που έπονται της εντολής CMP, μπορούν να εκτελούνται υπό συνθήκη ανάλογα με την κατάσταση των σημαιών
- Η αρχιτεκτονική ARM υποστηρίζει **εντολές υπό συνθήκη**
  - *το μνημονικό της εντολής ακολουθείται από ένα **μνημονικό συνθήκης** το οποίο υποδεικνύει τη **συνθήκη (CondEx)** που πρέπει να ικανοποιείται για να εκτελεσθεί η συγκεκριμένη εντολή*
  - *Το μνημονικό συνθήκης αποθηκεύεται στο **πεδίο συνθήκης (cond)** της εντολής μεγέθους 4 bit*
- Τα μνημονικά συνθήκης διαφέρουν όσον αφορά τη σύγκριση μεταξύ προσημασμένων ή μη προσημασμένων αριθμών
  - *για τη σύγκριση «μεγαλύτερο από ή ίσο με», το **HS** (higher or same) χρησιμοποιείται για τους μη προσημασμένους αριθμούς, ενώ το **GE** (greater or equal) για τους προσημασμένους*
  - *Στην περίπτωση των προσημασμένων αριθμών, όταν ισχύει **$A \geq B$** η πράξη A – B θα θέσει στις σημαίες τις τιμές:*
    - N = 0 και V = 0 (όταν δεν υπάρχει υπερχείλιση)
    - N = 1 και V = 1 (όταν υπάρχει υπερχείλιση)
  - *Στην περίπτωση των μη προσημασμένων αριθμών, όταν ισχύει **$A \geq B$** η πράξη A – B θα ενεργοποιήσει το κρατούμενο εξόδου (C = 1)*

Σύγκριση μεταξύ προσημασμένων ή μη προσημασμένων αριθμών

- Οι αριθμοί είναι των 4 bit για να διευκολυνθεί η ερμηνεία

	Unsigned	Signed
$A = 1001_2$	A = 9	A = –7
$B = 0010_2$	B = 2	B = 2

A – B: 1001 *NZCV* = 0011_2
+ 1110 **HS:** TRUE
(a) 10111 **GE:** FALSE

-B = 1101 + 1 = 1110

HS = true, γιατί C = 1
GE = false, γιατί N = 0 και V = 1

	Unsigned	Signed
$A = 0101_2$	A = 5	A = 5
$B = 1101_2$	B = 13	B = –3

A – B: 0101 *NZCV* = 1001_2
+ 0011 **HS:** FALSE
(b) 1000 **GE:** TRUE

-B = 0010 + 1 = 0011

HS = false, γιατί C = 0
GE = true, γιατί N = 1 και V = 1

Μνημονικά Συνθήκης (1/2)

$cond_{3:0}$	Μνημονικό	Όνομα	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\bar{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\bar{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\bar{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\bar{V}$

Οι συνθήκες με $cond_0 = 0$ είναι συμπληρωματικές των συνθηκών με $cond_0 = 1$

Μνημονικά Συνθήκης (2/2)

$cond_{3:0}$	Μνημονικό	Όνομα	CondEx
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z+\bar{C}$
1010	GE	Signed greater or equal	$\overline{N \oplus V}$
1011	LT	Signed less	$N \oplus V$
1100	GT	Signed greater	$\bar{Z}\,\overline{N \oplus V}$
1101	LE	Signed less or equal	$Z+(N \oplus V)$
1110	AL (ή none)	Always / unconditional	1
1111	none	For unconditional instructions	1

Οι συνθήκες με $cond_0 = 0$ είναι συμπληρωματικές των συνθηκών με $cond_0 = 1$
Δεν ισχύει για cond = 1110/1111

Συμβολική γλώσσα – Εντολές υπό συνθήκη

- Παράδειγμα εντολών με υπό συνθήκη εκτέλεση:
  - *Η εντολή CMP εκτελείται χωρίς συνθήκη και ενημερώνει τις σημαίες συνθήκης*
  - *Οι υπόλοιπες εντολές αριθμητικών και λογικών πράξεων:*
    - δεν επηρεάζουν τις σημαίες συνθήκης
    - εκτελούνται υπό συνθήκη, ανάλογα με τις τιμές των σημαιών συνθήκης που ενεργοποίησε η εντολή CMP

Κώδικας συμβολικής γλώσσας της ARM

```
CMP   R2, R3              ; ενημερώνει τις σημαίες
ADDEQ  R4, R5, #78        ; εκτελείται αν R2 = R3
ANDHS  R7, R8, R9         ; εκτελείται αν R2 ≥ R3 (U)
ORRMI  R10, R11, R12      ; εκτελείται αν N = 1
EORLT  R12, R7, R10       ; εκτελείται αν R2 < R3 (S)
```

Η αρχιτεκτονική ARM υποστηρίζει αριθμητικές και λογικές εντολές που ενημερώνουν τις σημαίες συνθήκης, όταν το μνημονικό εντολής ακολουθείται από το “S”, όπως ADDS

Συμβολική γλώσσα – Εντολές υπό συνθήκη

- Παράδειγμα εντολών με υπό συνθήκη εκτέλεση:

Κώδικας συμβολικής γλώσσας της ARM

```
CMP   R2, R3              ; ενημερώνει τις σημαίες
ADDEQ  R4, R5, #78        ; εκτελείται αν R2 = R3
ANDHS  R7, R8, R9         ; εκτελείται αν R2 ≥ R3 (U)
ORRMI  R10, R11, R12      ; εκτελείται αν N = 1
EORLT  R12, R7, R10       ; εκτελείται αν R2 < R3 (S)
```

Ας υποθέσουμε ότι: R2 = 0x80000000 και R3 = 0x00000001

R2 – R3 = 0x80000000 + 0xFFFFFFFF = 0x17FFFFFFF

Μετά την εκτέλεση της εντολής CMP οι σημαίες ενημερώνονται ως εξής:

N = 0, Z = 0, C =1, V = 1

Η εντολή ADDEQ δεν εκτελείται γιατί δεν ισχύει η συνθήκη Z = 1

Η εντολή ANDHS εκτελείται γιατί ισχύει η συνθήκη C = 1

Η εντολή ORRMI δεν εκτελείται γιατί δεν ισχύει η συνθήκη N = 1

Η εντολή EORLT εκτελείται γιατί ισχύει η συνθήκη N ⊕ V = 1

Συμβολική γλώσσα – Εντολές διακλάδωσης

- Η αρχιτεκτονική ARM υποστηρίζει εντολές διακλάδωσης (*branch instructions*), ώστε να αγνοεί τμήματα του κώδικα ή να επαναλαμβάνει τμήματα κώδικα
  - *Οι εντολές διακλάδωσης τροποποιούν το περιεχόμενο του program counter (PC), ώστε η επόμενη προς εκτέλεση εντολή να μην είναι η αμέσως επόμενη στη διεύθυνση PC+4*
- Η ARM περιλαμβάνει δύο τύπους διακλάδωσης:
  - *την απλή διακλάδωση (simple branch - B)*
  - *τη διακλάδωση και σύνδεση (branch and link - BL) για την κλήση συναρτήσεων (ή διαδικασιών)*
- Αμφότερες μπορούν να εκτελεστούν χωρίς συνθήκη ή υπό συνθήκη με βάση τα μνημονικά συνθήκης

Ως σύνδεση ορίζεται η αποθήκευση της διεύθυνσης εντολής PC+4 στον καταχωρητή σύνδεσης (link register – LR, R14) με την κλήση της συνάρτησης, ώστε στο τέλος της εκτέλεσης της συνάρτησης να είναι δυνατή η επιστροφή της εκτέλεσης του προγράμματος στην αμέσως επόμενη εντολή μετά την εντολή BL του καλούσας συνάρτησης

Συμβολική γλώσσα – Εντολές διακλάδωσης

- Ο κώδικας συμβολικής γλώσσας χρησιμοποιεί **ετικέτες** (labels) για να υποδεικνύει τις θέσεις των εντολών μέσα στο πρόγραμμα.
- Όταν ο κώδικας συμβολικής γλώσσας μεταφράζεται σε κώδικα μηχανής, αυτές οι ετικέτες μεταφράζονται σε διευθύνσεις εντολών
- Ο μεταγλωττιστής ARM υιοθετεί την απαίτηση ότι:
  - *οι ετικέτες τοποθετούνται στην αρχή της γραμμής, χωρίς κενό*
  - *πριν από τις εντολές πρέπει να υπάρχει κενό*
- Παράδειγμα διακλάδωσης χωρίς συνθήκη:

Κώδικας συμβολικής γλώσσας της ARM

```
   ADD R1, R2, #17    ; R1 = R2 + 17
   B TARGET           ; διακλάδωση στο TARGET
   ORR R1, R1, R3     ; δεν εκτελείται
   AND R3, R1, #0xFF  ; δεν εκτελείται
TARGET
   SUB R1, R1, #78    ; R1 = R1 – 78
```

Όταν η εκτέλεση του προγράμματος φθάσει στην εντολή B TARGET, η επόμενη εντολή που εκτελείται είναι η εντολή SUB που βρίσκεται αμέσως μετά από την ετικέτα με το όνομα TARGET

Συμβολική γλώσσα – Εντολές διακλάδωσης

- Παράδειγμα διακλάδωσης υπό συνθήκη:

Κώδικας συμβολικής γλώσσας της ARM

```
   MOV R0, #4          ; R0 = 4
   ADD R1, R0, R0      ; R1 = R0 + R0 = 8
   CMP R0, R1          ; ενεργοποίηση σημαιών λόγω CMP
                       ; ισχύει R0 < R1, NZCV = 1000
   BEQ THERE           ; διακλάδωση, εάν Z = 1
   ORR R1, R1, #1      ; R1 = R1 OR 1 = 9
THERE
   ADD R1, R1, #78 ; R1 = R1 + 78 = 87
```

Όταν η εκτέλεση του προγράμματος φθάσει στην εντολή BEQ, η σημαία συνθήκης Z έχει τιμή 0, οπότε δεν ακολουθείται η διακλάδωση και η επόμενη εντολή που εκτελείται είναι η εντολή ORR που βρίσκεται αμέσως μετά από την εντολή BEQ

Προγραμματισμός – Η εντολή if

- Παράδειγμα μεταγλώττισης της εντολής if

Κώδικας γλώσσας υψηλού επιπέδου

```
if (apples == oranges)
    f = i + 1;
f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = apples, R1 = oranges, R2 = f, R3 = i
  CMP R0, R1      ; apples = oranges ?
  BNE L1          ; αν όχι, διακλάδωση στο L1
                  ; και παραλείπεται το τμήμα if
  ADD R2, R3, #1  ; τμήμα if: f = i + 1
L1
  SUB R2, R2, R3  ; f = f - i
```

Παρατηρείστε ότι ελέγχεται η αντίθετη συνθήκη **apples ≠ oranges**

Προγραμματισμός – Η εντολή if

- Παράδειγμα μεταγλώττισης της εντολής if

Κώδικας γλώσσας υψηλού επιπέδου

```
if (apples == oranges)
    f = i + 1;
f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = apples, R1 = oranges, R2 = f, R3 = i
CMP R0, R1          ; apples = oranges ?
ADDEQ R2, R3, #1    ; αν R0 = R1 (Z=1), f = i + 1
SUB R2, R2, R3      ; f = f - i
```

Σε αυτό το παράδειγμα φαίνεται η χρησιμότητα της εκτέλεσης εντολής υπό συνθήκη στην αρχιτεκτονική ARM, που μειώνει το μέγεθος και τον χρόνο εκτέλεσης του προγράμματος

Προγραμματισμός – Η εντολή if/else

- Παράδειγμα μεταγλώττισης της εντολής if/else

Κώδικας γλώσσας υψηλού επιπέδου

```
if (apples == oranges)
   f = i + 1;
else
   f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = apples, R1 = oranges, R2 = f, R3 = i
  CMP R0, R1      ; apples = oranges ?
  BNE L1          ; αν όχι, διακλάδωση στο L1
                  ; και παραλείπεται το τμήμα if
  ADD R2, R3, #1  ; τμήμα if:   f = i + 1
  B L2            ; διακλάδωση στο L2 ώστε να
                  ; παραλείπεται το τμήμα else
L1
  SUB R2, R2, R3  ; τμήμα else: f = f - i
L2
```

Παρατηρείστε ότι ελέγχεται η αντίθετη συνθήκη **apples ≠ oranges**

Προγραμματισμός – Η εντολή if/else

- Παράδειγμα μεταγλώττισης της εντολής if/else

Κώδικας γλώσσας υψηλού επιπέδου

```
if (apples == oranges)
   f = i + 1;
else
   f = f - i;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = apples, R1 = oranges, R2 = f, R3 = i
CMP R0, R1          ; apples = oranges ?
ADDEQ R2, R3, #1    ; αν R0 = R1 (Z=1), f = i + 1
SUBNE R2, R2, R3    ; αν R0 ≠ R1 (Z=0), f = f - i
```

Σε αυτό το παράδειγμα φαίνεται η χρησιμότητα της εκτέλεσης εντολών υπό συνθήκη στην αρχιτεκτονική ARM, που μειώνει το μέγεθος και τον χρόνο εκτέλεσης του προγράμματος

Προγραμματισμός – Η εντολή if/else if/else

- Παράδειγμα μεταγλώττισης της εντολής if/else if/else

Κώδικας γλώσσας υψηλού επιπέδου

```
if      (button == 1) amt = 20;
else if (button == 2) amt = 50;
else if (button == 3) amt = 100;
else                  amt = 0;
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = button, R1 = amt
  CMP R0, #1      ; έχει πατηθεί το κουμπί 1?
  MOVEQ R1, #20   ; αν ναι, τότε amt = 20
  BEQ DONE        ; αν ναι, διακλάδωση στο DONE
  CMP R0, #2      ; έχει πατηθεί το κουμπί 2?
  MOVEQ R1, #50   ; αν ναι, τότε amt = 50
  BEQ DONE        ; αν ναι, διακλάδωση στο DONE
  CMP R0, #3      ; έχει πατηθεί το κουμπί 3?
  MOVEQ R1, #100  ; αν ναι, τότε amt = 100
  BEQ DONE        ; αν ναι, διακλάδωση στο DONE
  MOV R1, #0      ; default τιμή: amt = 0
DONE
```

Προγραμματισμός – Βρόχος while

- Παράδειγμα μεταγλώττισης του βρόχου while
  - *Ο βρόχος while εκτελεί επανειλημμένα ένα τμήμα κώδικα μέχρι να **μην** ικανοποιείται μία συνθήκη*
  - *Στο παράδειγμα βρίσκει την τιμή του x για την οποία ισχύει $2^x = 128$*

Κώδικας γλώσσας υψηλού επιπέδου

```
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    ; pow = 128 ?
  BEQ DONE        ; αν ναι, έξοδος από βρόχο
  LSL R0, R0, #1  ; pow = pow * 2
  ADD R1, R1, #1  ; x = x + 1
  B   WHILE       ; επανάληψη βρόχου
DONE
```

Παρατηρείστε ότι ελέγχεται η αντίθετη συνθήκη **pow = 128**

Προγραμματισμός – Βρόχος for

- Παράδειγμα μεταγλώττισης του βρόχου for
  - *Ο βρόχος for συνδυάζει τον καθορισμό της αρχικής τιμής, τον έλεγχο της συνθήκης και την τροποποίηση της μεταβλητής σε ένα σημείο*
    - Χρησιμοποιείται όταν είναι γνωστό το πλήθος των επαναλήψεων του βρόχου
  - *Στο παράδειγμα προστίθενται οι αριθμοί από το 0 έως το 9*
    - ο βρόχος επαναλαμβάνεται 10 φορές (το i αυξάνεται κατά 1 στο τέλος του βρόχου, ενώ η συνθήκη (i < 10) ελέγχεται στην αρχή του βρόχου

Κώδικας γλώσσας υψηλού επιπέδου

```
int i;
int sum = 0;
for (i = 0; i < 10; i = i + 1) {
    sum = sum + i;
}
```

Κώδικας συμβολικής γλώσσας της ARM

```
; R0 = i, R1 = sum
  MOV R1, #0      ; sum = 0
  MOV R0, #0      ; i = 0
FOR
  CMP R0, #10     ; i ≥ 10 ?
  BGE DONE        ; αν ναι, έξοδος από βρόχο
  ADD R1, R1, R0  ; sum = sum + i
  ADD R0, R0, #1  ; i = i + 1
  B FOR           ; επανάληψη βρόχου
DONE
```

Παρατηρείστε ότι ελέγχεται η αντίθετη συνθήκη `i ≥ 10`

Προγραμματισμός – Προσπέλαση πίνακα

- Για λόγους διευκόλυνσης της αποθήκευσης και της προσπέλασης, παρόμοια ομοειδή δεδομένα (στοιχεία) μπορούν να ομαδοποιούνται σε έναν **πίνακα** (*array*)
  - *Τα περιεχόμενα του πίνακα αποθηκεύονται σε **συνεχόμενες λέξεις δεδομένων** στη μνήμη*
  - *Κάθε στοιχείο του πίνακα προσδιορίζεται από έναν αριθμό που είναι γνωστός ως ο **αριθμοδείκτης** (index) του πίνακα*
  - *Το πλήθος των στοιχείων του πίνακα συνιστά το **μήκος** (length) του*
  - *Παράδειγμα του πίνακα* **scores[200]** *με 200 λέξεις δεδομένων των 32 bit (4 byte) και διεύθυνση βάσης (του πρώτου στοιχείου του πίνακα) την 0x14000000*

Προγραμματισμός – Προσπέλαση πίνακα με βρόχος for

- Παράδειγμα μεταγλώττισης της προσπέλασης του πίνακα με βρόχο for
  - *Στο παράδειγμα προστίθενται 10 μονάδες σε κάθε στοιχείο του πίνακα*
    - Ο βρόχος επαναλαμβάνεται 200 φορές (το i αυξάνεται κατά 1 στο τέλος του βρόχου, ενώ η συνθήκη (i < 200) ελέγχεται στην αρχή του βρόχου
    - Η διεύθυνση του πίνακα αυξάνεται κατά 4 γιατί κάθε στοιχείο του πίνακα έχει μέγεθος 4 byte και η μνήμη είναι προσπελάσιμη κατά byte
    - Ο τρόπος διευθυνσιοδότησης της μνήμης είναι η έμμεση διευθυσιοδότηση καταχωρητή (η σχετική απόσταση (offset) είναι 0)

Κώδικας συμβολικής γλώσσας της ARM	Κώδικας γλώσσας υψηλού επιπέδου

```
; R0 = διεύθυνση πίνακα, R1 = i
; κώδικας καθορισμού αρχικών τιμών
; αρχικά R0 = διεύθυνση βάσης
   MOV R0, #0x14000000
   MOV R1, #0          ; i = 0
LOOP
   CMP R1, #200        ; i ≥ 200?
   BGE DONE            ; αν ναι, έξοδος από βρόχο
   LDR R3, [R0]        ; R3 = scores[i]
   ADD R3, R3, #10     ; R3 = R3 + 10
   STR R3, [R0]        ; scores[i] = R3
   ADD R0, R0, #4      ; R0 = R0 + 4
   ADD R1, R1, #1      ; i = i + 1
   B LOOP              ; επανάληψη βρόχου
DONE
```

```
int i;
int scores[200];
...
for (i = 0; i < 200; i = i + 1)
    scores[i] = scores[i] + 10;
```

Παρατηρείστε ότι ελέγχεται η αντίθετη συνθήκη **i ≥ 200**

Προγραμματισμός – Κλήσεις συναρτήσεων

- Οι γλώσσες προγραμματισμού υποστηρίζουν **συναρτήσεις** (*functions*), καθώς και **διαδικασίες** ή **υπορουτίνες** (*procedures* ή *subroutines*)
  - *με τις συναρτήσεις είναι εφικτή η επαναχρησιμοποίηση κώδικα και η ενίσχυση της αναγνωσιμότητας ενός προγράμματος*
- Οι συναρτήσεις έχουν εισόδους που ονομάζονται **ορίσματα** (*arguments*), και μια έξοδο που ονομάζεται **επιστρεφόμενη τιμή** (*return value*)
  - *οι διαδικασίες έχουν περισσότερες επιστρεφόμενες τιμές*
- Στην αρχιτεκτονική ARM ισχύει οι εξής συμβάσεις:
  - *η **καλούσα συνάρτηση** (caller) τοποθετεί έως και τέσσερα ορίσματα στους καταχωρητές **R0–R3** πριν την κλήση της συνάρτησης*
  - *η **καλούμενη συνάρτηση** (callee) τοποθετεί την επιστρεφόμενη τιμή στον καταχωρητή **R0** πριν ολοκληρωθεί η εκτέλεσή της*
  - *η **καλούσα συνάρτηση** αποθηκεύει τη διεύθυνση επιστροφής (PC+4) στον **καταχωρητή σύνδεσης** (link register – LR, R14) με την κλήση της καλούμενης συνάρτησης με την εντολή διακλάδωσης και σύνδεσης **BL***
  - *η **καλούμενη συνάρτηση** πρέπει να προστατεύσει τους καταχωρητές **R4–R11 και R14 (LR)** (όταν η ίδια καλεί μία άλλη συνάρτηση), καθώς και περιοχές της μνήμης που χρησιμοποιεί η καλούσα συνάρτηση*
    - Επιτυγχάνεται με τη χρήση της **στοίβας** (stack) για προσωρινή αποθήκευση

Προγραμματισμός – Κλήσεις συναρτήσεων

- Παράδειγμα απλής κλήσης και επιστροφής από συνάρτηση χωρίς ανταλλαγή τιμών (η main καλεί τη simple)
  - *χρησιμοποιούνται οι εντολές BL και MOV PC, LR*
  - *η BL στη διεύθυνση PC εκτελεί δύο μεταφορές δεδομένων:*
    - LR = PC+4 (LR=R14, PC=15)
    - PC = 6 msb του PC & διεύθυνση διακλάδωσης των 24 bit & “00”

Κώδικας συμβολικής γλώσσας της ARM

```
0x00008000 MAIN  ...
...              ...
0x00008020   BL SIMPLE; κλήση simple
0x00008024   ...
...
0x0000902C SIMPLE MOV PC, LR; επιστροφή
```

Κώδικας γλώσσας υψηλού επιπέδου

```
int main() {
    ...
    simple();
    ...
}
void simple() {
    return;
}
```

LR = 0x00008024

Προγραμματισμός – Κλήσεις συναρτήσεων

- Παράδειγμα κλήσης και επιστροφής από συνάρτηση με ορίσματα και επιστρεφόμενη τιμή (χωρίς χρήση στοίβας) (η main καλεί τη dif)
  - *χρησιμοποιούνται οι εντολές BL και MOV PC, LR*

Κώδικας συμβολικής γλώσσας της ARM

```
; R4 = y μεταβλητή της MAIN
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 DIF     ; κλήση dif
  MOV R4, R0 ; επιστρ. τιμή
  ...
; R4 = result μεταβλ. της DIF
DIF
  ADD R8, R0, R1 ; R8 = f + g
  ADD R9, R2, R3 ; R9 = h + i
  SUB R4, R8, R9 ; R4 = result
  MOV R0, R4     ; R0 = επιστρ. τιμή
  MOV PC, LR     ; επιστροφή
```

Κώδικας γλώσσας υψηλού επιπέδου

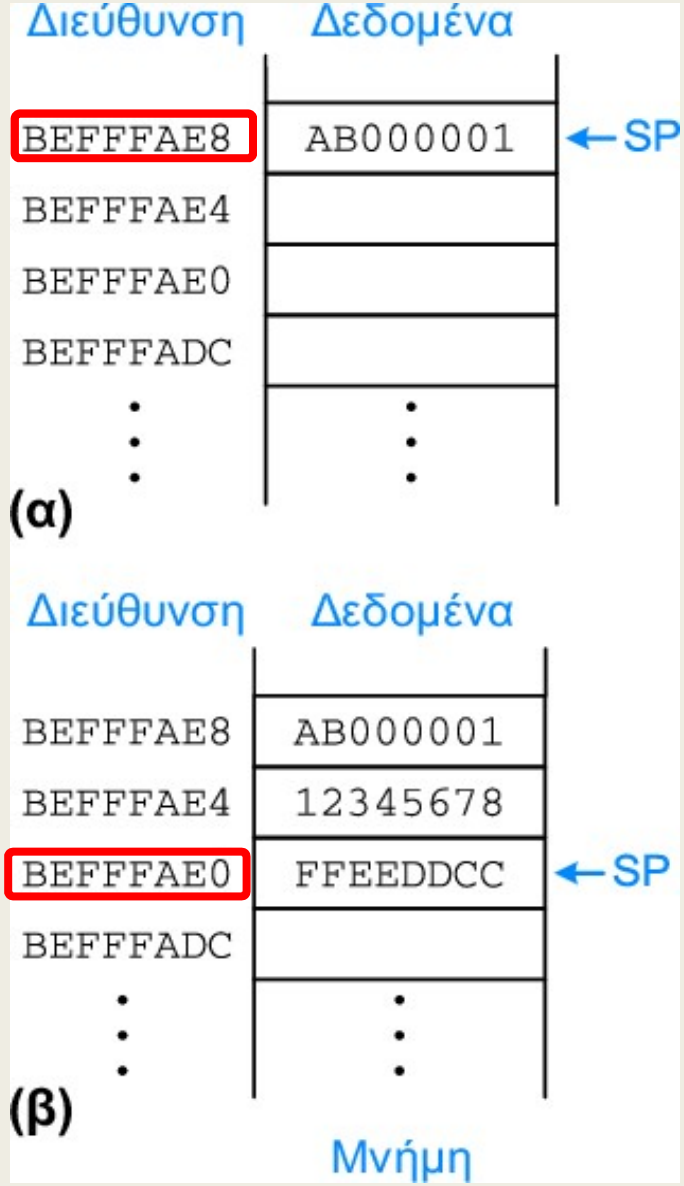
```
int main() {
    int y;
    ...
    y = dif(2, 3, 4, 5);
    ...
}

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

Οι καταχωρητές R4, R8 και P9 δεν προστατεύονται !

Η στοίβα

- Η στοίβα είναι ένα τμήμα μνήμης που χρησιμοποιείται για την προσωρινή αποθήκευση καταχωρητών που πρέπει να προστατευθούν κατά την κλήση και εκτέλεση μιας συνάρτησης
- Η στοίβα είναι μια ουρά LIFO (last-in-first-out):
  - *Το τελευταίο στοιχείο που τοποθετείται (push) πάνω στη στοίβα είναι και το πρώτο που μπορεί να ανακτηθεί (pop).*
- Η *κορυφή της στοίβας* αντιστοιχεί στην πιο πρόσφατα αποθηκευμένη λέξη δεδομένων
- Η αρχιτεκτονική ARM χρησιμοποιεί τον R13 του αρχείου καταχωρητών ως δείκτη στοίβας (stack pointer – SP) που δείχνει στην κορυφή της στοίβας
  - *Ο SP ξεκινάει σε μια υψηλή διεύθυνση μνήμης και μειώνεται κατά 4 κάθε φορά που νέα δεδομένα μπαίνουν στη στοίβα*
    - Η στοίβα (α) πριν από την επέκταση και (β) μετά από επέκταση δύο λέξεων



Προγραμματισμός – Κλήσεις συναρτήσεων

- Η στοίβα είναι ένα τμήμα μνήμης που χρησιμοποιείται για την προσωρινή αποθήκευση καταχωρητών που πρέπει να προστατευθούν κατά την κλήση και εκτέλεση μιας συνάρτησης
  - *Οι συναρτήσεις αποθηκεύουν τα περιεχόμενα των καταχωρητών στη στοίβα προτού τα τροποποιήσουν, και μετά τα ανακτούν από τη στοίβα πριν την επιστροφή*
  - *Οι συναρτήσεις εκτελούν τα ακόλουθα βήματα:*
    1. Δέσμευση κατάλληλου χώρου στη στοίβα που ονομάζεται πλαίσιο στοίβας μειώνοντας κατάλληλα τον SP
    2. Αποθήκευση των τιμών των καταχωρητών στη στοίβα
    3. Χρήση των καταχωρητών εντός της συνάρτησης
    4. Επαναφορά των αρχικών τιμών των καταχωρητών από τη στοίβα
    5. Αποδέσμευση χώρου στη στοίβα αυξάνοντας τον SP
  - *Η αρχή της τμηματικότητας (modularity) ορίζει ότι κάθε συνάρτηση πρέπει να προσπελάζει μόνο το δικό της πλαίσιο στοίβας, και όχι τα πλαίσια που ανήκουν σε άλλες συναρτήσεις*

Προγραμματισμός – Κλήσεις συναρτήσεων

- Παράδειγμα κλήσης και επιστροφής από συνάρτηση με ορίσματα και επιστρεφόμενη τιμή (με χρήση στοίβας) (η main καλεί τη dif)
  - *χρησιμοποιούνται οι εντολές BL και MOV PC, LR*

Κώδικας συμβολικής γλώσσας της ARM	Κώδικας γλώσσας υψηλού επιπέδου

```
; R4 = result μεταβλ. της DIF
DIF
; δέσμευση χώρου & αποθήκευση
  SUB SP, SP, #12
  STR R9, [SP, #8]
  STR R8, [SP, #4]
  STR R4, [SP]
; κώδικας συνάρτησης
  ADD R8, R0, R1 ; R8 = f + g
  ADD R9, R2, R3 ; R9 = h + i
  SUB R4, R8, R9 ; R4 = result
  MOV R0, R4     ; R0 = επιστρ. τιμή
; επαναφορά & αποδέσμευση χώρου
  LDR R4, [SP]
  LDR R8, [SP, #4]
  LDR R9, [SP, #8]
  ADD SP, SP, #12
; επιστροφή
  MOV PC, LR
```

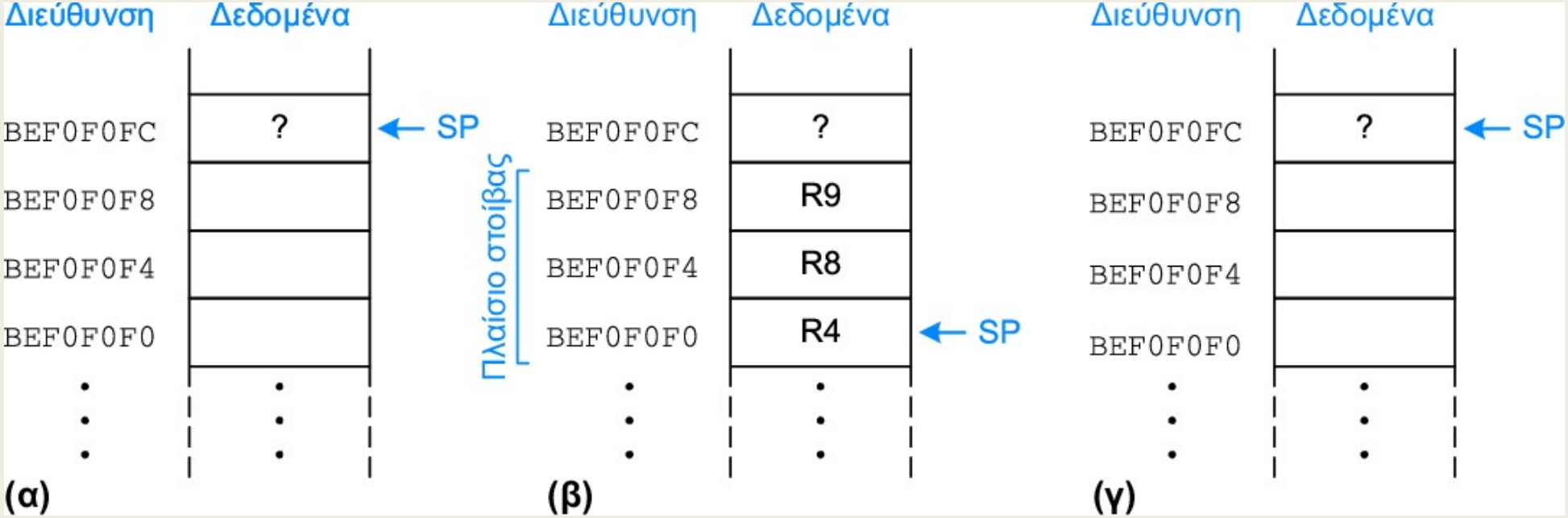
```
int dif(int f, int g, int h, int i) {
    int result;
    result = (f + g) – (h + i);
    return result;
}
```

Οι καταχωρητές R4, R8 και P9 προστατεύονται !

Προσοχή! Οι καταχωρητές επαναφέρονται στις αρχικές τιμές τους με την αντίστροφη σειρά που αποθηκεύονται στη στοίβα

Προγραμματισμός – Κλήσεις συναρτήσεων

- Η στοίβα: (α) πριν από, (β) κατά τη διάρκεια και (γ) μετά από την κλήση της συνάρτησης dif



Εάν μία καλούμενη συνάρτηση καλεί μία άλλη συνάρτηση, τότε και ο καταχωρητής σύνδεσης (link register – LR, R14) πρέπει να αποθηκευτεί στη στοίβα

Προγραμματισμός – Κλήσεις συναρτήσεων

- Η αρχιτεκτονική ARM χωρίζει τους καταχωρητές του αρχείου καταχωρητών σε δύο κατηγορίες, ώστε να αποφεύγονται άσκοπες αποθηκεύσεις στη στοίβα
  - *οι **διατηρούμενοι καταχωρητές** αποθηκεύονται από την **καλούμενη συνάρτηση**, όταν τους χρησιμοποιεί*
  - *οι **μη διατηρούμενοι καταχωρητές** αποθηκεύονται από την **καλούσα συνάρτηση**, όταν τους χρησιμοποιεί*

Διατηρούμενοι	Μη Διατηρούμενοι
Αποθηκευόμενοι καταχωρητές: R4–R11	Προσωρινός καταχωρητής: R12
Δείκτης στοίβας: SP (R13)	Καταχωρητές ορισμάτων: R0–R3
Διεύθυνση επιστροφής: LR (R14)	Καταχωρητής Κατάστασης Τρέχοντος Προγράμματος (CPSR)
Στοίβα πάνω από τον δείκτη στοίβας	Στοίβα κάτω από τον δείκτη στοίβας

Προγραμματισμός – Κλήσεις συναρτήσεων

- Παράδειγμα κλήσης και επιστροφής από συνάρτηση με ορίσματα και επιστρεφόμενη τιμή (με χρήση στοίβας) (η main καλεί τη dif)
  - *Μείωση της χρήσης διατηρούμενων καταχωρητών από την dif*
    - Χρησιμοποιούνται αποκλειστικά οι μη διατηρούμενοι καταχωρητές R0 – R3
  - *Δεν απαιτείται πλέον η χρήση της στοίβας*

Κώδικας γλώσσας υψηλού επιπέδου

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

Κώδικας συμβολικής γλώσσας της ARM

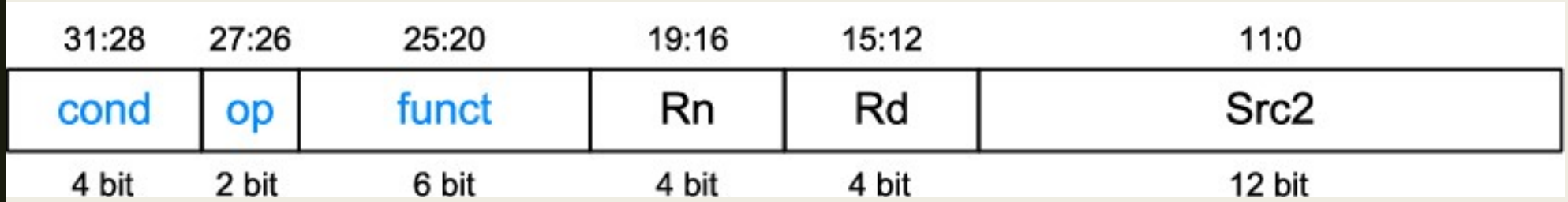
```
DIF
  ADD R1, R0, R1 ; R1 = f + g
  ADD R3, R2, R3 ; R3 = h + i
  SUB R0, R1, R3 ; επιστροφή του (f+g)-(h+i)
  MOV PC, LR     ; επιστροφή
```

Γλώσσα μηχανής

- Η συμβολική γλώσσα είναι πιο ευανάγνωστη για τους ανθρώπους Όμως τα ψηφιακά κυκλώματα κατανοούν μόνο τις τιμές 0 και 1
- Ο συμβολομεταφραστής (assembler) μεταφράζει τις εντολές στη συμβολική γλώσσα στη δυαδική έκδοσή τους (σε γλώσσα μηχανής)
- Η αρχιτεκτονική ARM ορίζει εντολές σε γλώσσα μηχανής σταθερής κωδικοποίησης των 32 bit με τρεις κύριες μορφές εντολών:
  - *εντολές επεξεργασίας δεδομένων*,
  - *εντολές μνήμης*, *και*
  - *εντολές διακλάδωσης*

Γλώσσα μηχανής – Γενική μορφή εντολών

- Γενική μορφή εντολής:



- Πεδία εντολής:
  - *Πεδίο cond (condition, 4 bit, 31:28) για τη δήλωση των μνημονικών συνθήκης*
    - cond = 1110 για τις εντολές χωρίς συνθήκη
    - cond = 0000 – 1101 για τις εντολές υπό συνθήκη
  - *Πεδίο op (opcode, 2 bit, 27:26) για τη δήλωση της μορφής των εντολών*
    - op = 00 για τις εντολές επεξεργασίας δεδομένων
    - op = 01 για τις εντολές μνήμης
    - op = 10 για τις εντολές διακλάδωσης

Γλώσσα μηχανής – Εντολές επεξεργασίας δεδομένων

- Μορφή εντολής:

- Πεδία εντολής:
  - *Πεδίο funct (function, 6 bit, 25:20) για τη δήλωση της λειτουργίας/πράξης*
    - Υποπεδίο I (immediate, 1 bit, 25) για τη δήλωση ύπαρξης άμεσου τελεστέου στο πεδίο src2 (I=1 άμεσος τελεστέος)
    - Υποπεδίο cmd (command, 4 bit, 24:21) για τη δήλωσης της πράξης
      - *ADD = 0100, SUB = 0010, CMP = 1010*
      - *AND = 0000, ORR = 1100, EOR = 0001*
      - *LSL = LSR = ASR = ROR = 1101*
      - *MOV = 1101, MVN = 1111, BIC = 1110*
    - Υποπεδίο S (1 bit, 20) για τη ενημέρωση ή μη των σημαιών συνθήκης
      - *S = 1, η εντολή ενημερώνει τις σημαίες*

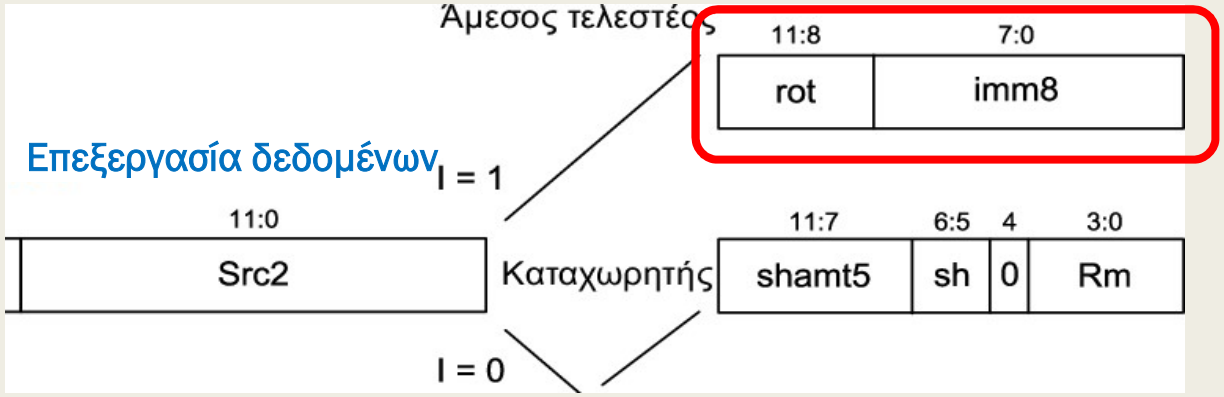
Γλώσσα μηχανής – Εντολές επεξεργασίας δεδομένων

- Μορφή εντολής:

- Πεδία εντολής:
  - *Πεδίο Rn (4 bit, 19:16) για τη δήλωση του καταχωρητή προέλευσης του πρώτου τελεστέου*
  - *Πεδίο Rd (4 bit, 15:12) για τη δήλωση του καταχωρητή προορισμού του αποτελέσματος της πράξης*
  - *Πεδίο Src2 (12 bit, 11:0) για τον προσδιορισμό του δευτέρου τελεστέου προέλευσης*
    - Θα επικεντρώσουμε στη δήλωση:
      - *άμεσου τελεστέου*
      - *δευτέρου τελεστέου σε καταχωρητή προέλευσης του οποίου το περιεχόμενο δεν ολισθαίνει*
      - *δευτέρου τελεστέου σε καταχωρητή προέλευσης του οποίου το περιεχόμενο ολισθαίνει κατά σταθερή ποσότητα ολίσθησης*

Γλώσσα μηχανής – Εντολές επεξεργασίας δεδομένων

- Μορφή εντολής:



- Υποπεδία του πεδίου Src2 της εντολής για δήλωση άμεσου τελεστέου (I=1):
  - *Υποπεδίο imm8 (8 bit, 7:0) για την αποθήκευση ενός μη προσημασμένου άμεσου τελεστέου των 8 bit*
  - *Υποπεδίο rot (4 bit, 11:8) για τη δήλωση της άμεσης ποσότητας περιστροφής προς τα δεξιά του άμεσου τελεστέου των 8 bit*
    - ώστε να μετατραπεί σε **σταθερά των 32 bit** σύμφωνα με την πράξη:

      imm32 = imm8 right shift by (2 x rot)

Γλώσσα μηχανής – Εντολές επεξεργασίας δεδομένων

- Περιστροφές άμεσων τελεστέων και η σταθερά των 32 bit που προκύπτει για imm8 = 0xFF
  - *Εκτελείται η πράξη: imm32 = imm8 right shift by (2 x rot)*

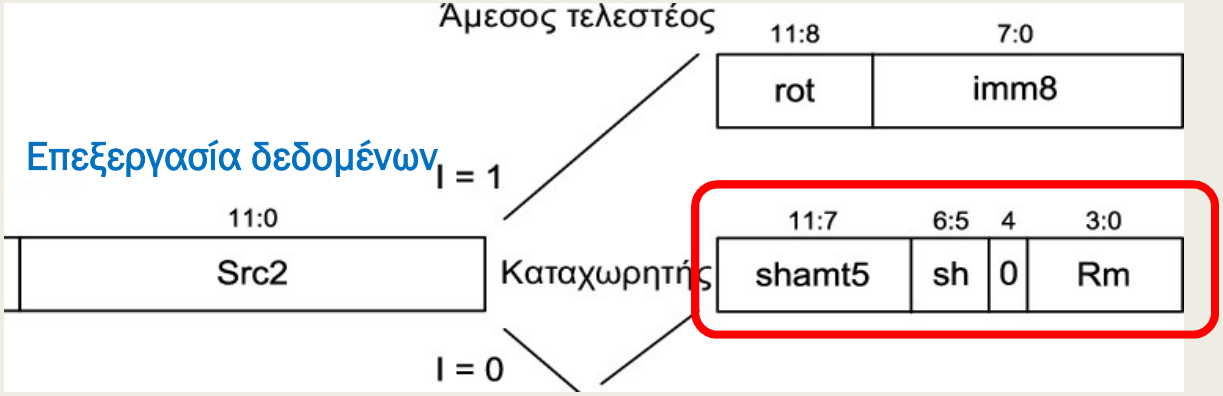
rot	Σταθερά 32 bit
0000	0000 0000 0000 0000 0000 0000 1111 1111
0001	1100 0000 0000 0000 0000 0000 0011 1111
0010	1111 0000 0000 0000 0000 0000 0000 1111
…	…
1111	0000 0000 0000 0000 0000 0011 1111 1100

Η συγκεκριμένη αναπαράσταση είναι πολύτιμη επειδή επιτρέπει σε πολλές χρήσιμες σταθερές, μεταξύ των οποίων και πολλαπλάσια οποιασδήποτε δύναμης του δύο, να αναπαρασταθούν με ένα μικρό πλήθος bit

Η δεξιά ολίσθηση κατά (2 x rot) θέσεις αντιστοιχεί σε αριστερή ολίσθηση κατά 32 – (2 x rot) θέσεις
Π.χ.: #4080 = 0xFF0 → rot = 14, imm8 = 0xFF

Γλώσσα μηχανής – Εντολές επεξεργασίας δεδομένων

- Μορφή εντολής:

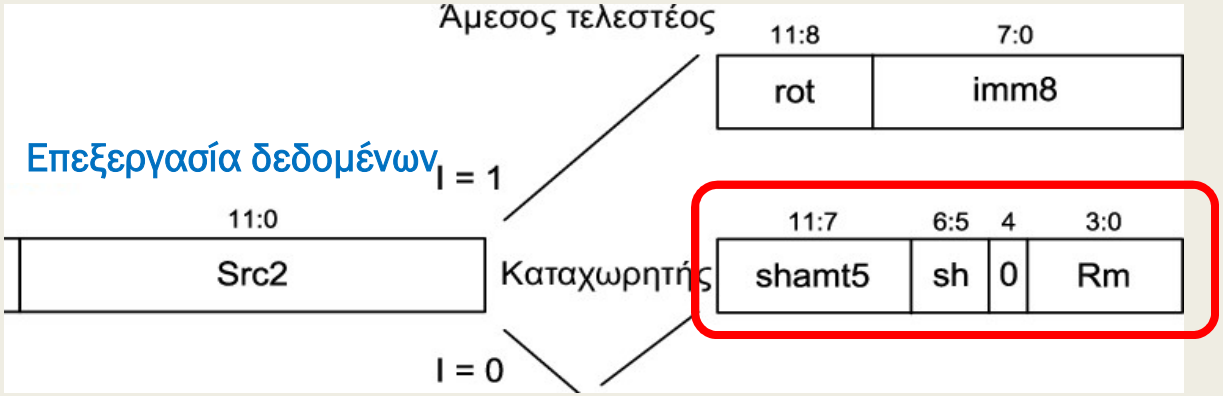


- Υποπεδία του πεδίου Src2 της εντολής για δήλωση δευτέρου τελεστέου σε καταχωρητή προέλευσης του οποίου το περιεχόμενο δεν ολισθαίνει (I=0):
  - *Υποπεδίο Rm (4 bit, 3:0) για τη δήλωση του καταχωρητή προέλευσης του δεύτερου τελεστέου*
    - Υποχρεωτικά: shamt5 = 0, sh = 0

Για τη δήλωση του καταχωρητή προέλευσης του πρώτου τελεστέου χρησιμοποιείται το πεδίο Rn

Γλώσσα μηχανής – Εντολές επεξεργασίας δεδομένων

- Μορφή εντολής:

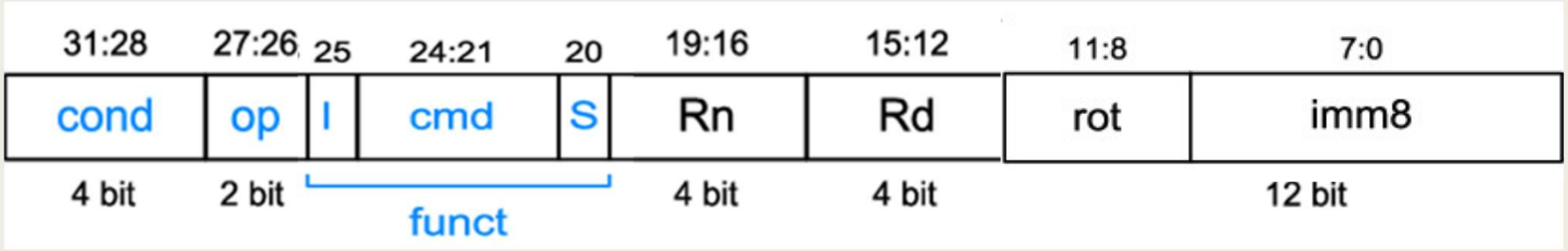


- Υποπεδία του πεδίου Src2 της εντολής για δήλωση δευτέρου τελεστέου σε καταχωρητή προέλευσης του οποίου το περιεχόμενο ολισθαίνει κατά σταθερή ποσότητα ολίσθησης (0-31) (I=0):
  - *Υποπεδίο Rm (4 bit, 3:0) για τη δήλωση του καταχωρητή προέλευσης του δευτέρου τελεστέου*
  - *Υποπεδίο shamt5 (shift amount, 5 bit, 11:7) για τη δήλωση της σταθερής ποσότητας ολίσθησης (0-31)*
  - *Υποπεδίο sh (shift, 2 bit, 6:5) για τη δήλωση του τύπου της ολισθησης (LSL = 00, LSR = 01, ASR = 10, ROR = 11)*

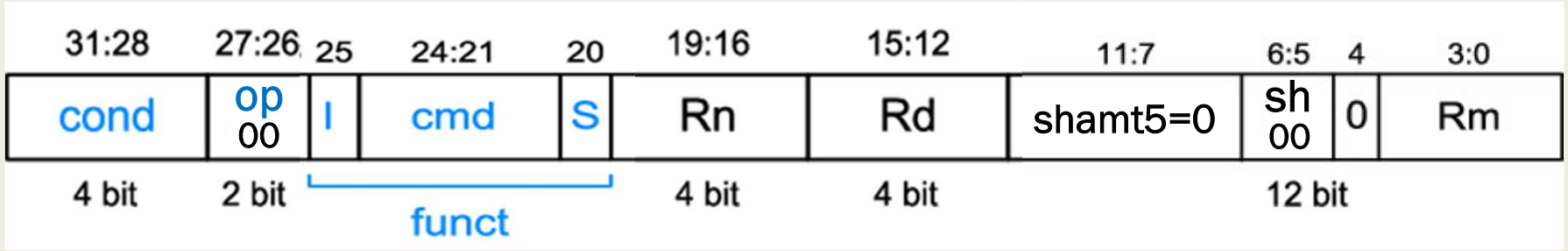
Στην εντολές MOV, MVN και στις εντολές ολίσθησης δεν χρησιμοποιείται ο πρώτος τελεστέος που δηλώνεται στο πεδίο Rn. Υποχρεωτικά Rn = 0. Ο δεύτερος τελεστέος που δηλώνεται στο υποπεδίο Rm εκλαμβάνεται ως πρώτος.

Γλώσσα μηχανής – Η εντολή ADD (cmd = 0100)

- Κώδικας συμβολικής γλώσσας:
  - *ADD Rd, Rn, #imm32 ; Rd = Rn + imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0

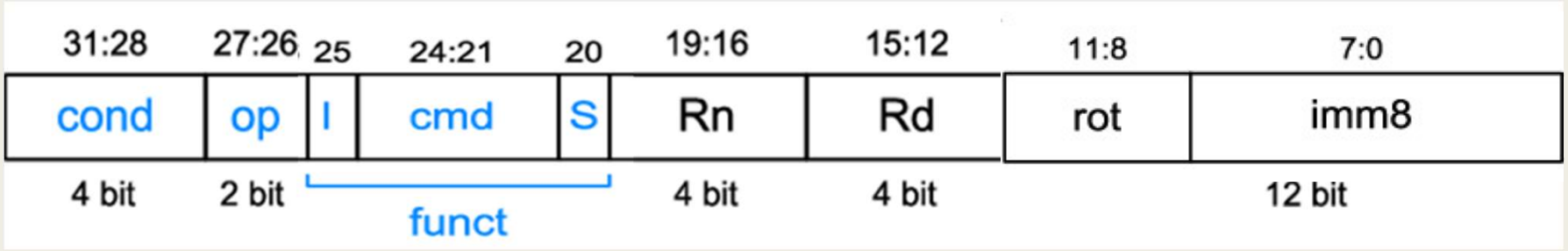


- Κώδικας συμβολικής γλώσσας:
  - *ADD Rd, Rn, Rm; Rd = Rn + Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, shamt5 = 0, sh = 00

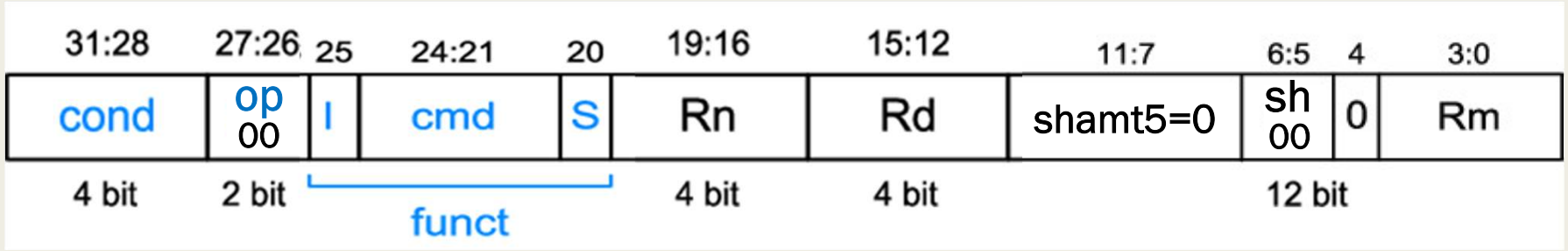


Γλώσσα μηχανής – Η εντολή SUB (cmd = 0010)

- Κώδικας συμβολικής γλώσσας:
  - *SUB Rd, Rn, #imm32 ; Rd = Rn - imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0

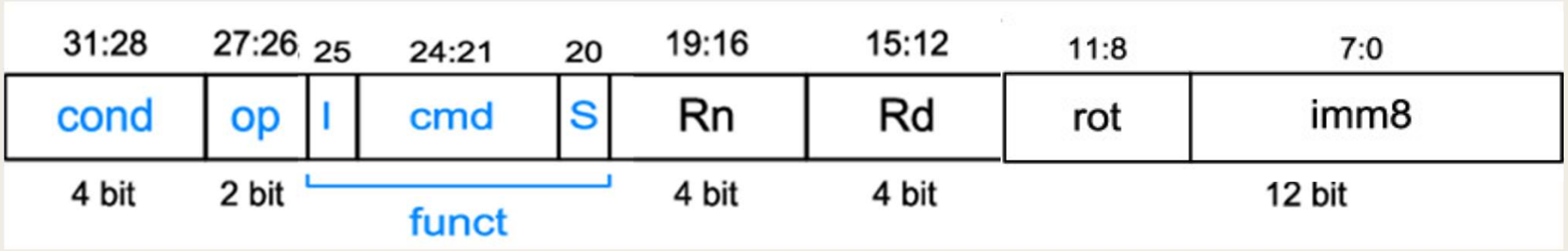


- Κώδικας συμβολικής γλώσσας:
  - *SUB Rd, Rn, Rm; Rd = Rn – Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, shamt5 = 0, sh = 00

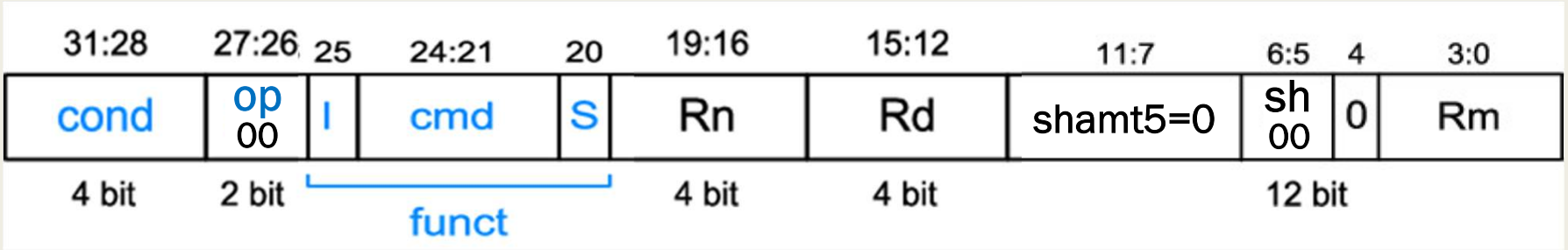


Γλώσσα μηχανής – Η εντολή CMP (cmd = 1010)

- Κώδικας συμβολικής γλώσσας:
  - *CMP Rn, #imm32; ενημέρωση σημαιών για Rn - imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 1, Rd = 0

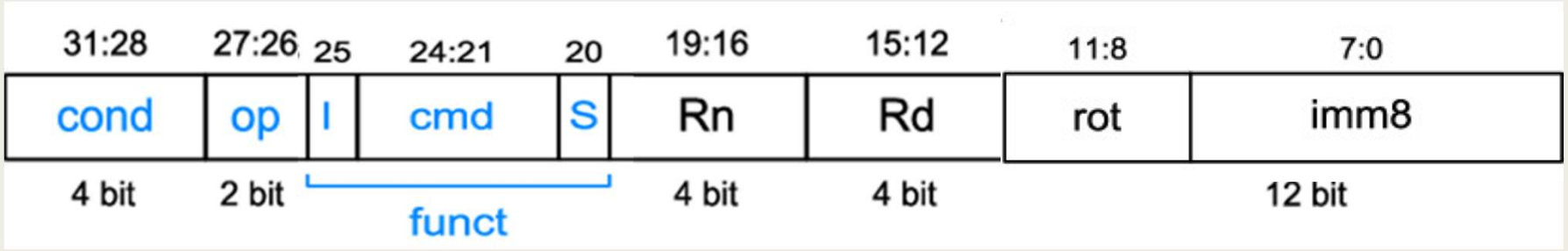


- Κώδικας συμβολικής γλώσσας:
  - *CMP Rn, Rm; ενημέρωση σημαιών για Rn – Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 1, Rd = 0, shamt5 = 0, sh = 00

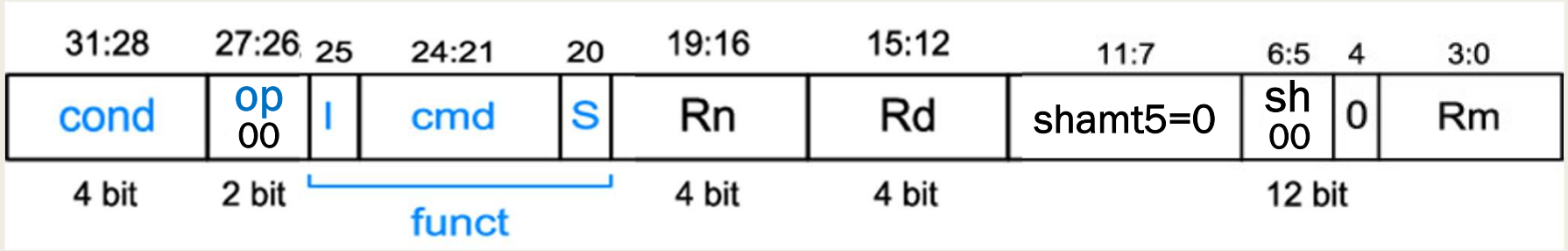


Γλώσσα μηχανής – Η εντολή AND (cmd = 0000)

- Κώδικας συμβολικής γλώσσας:
  - *AND Rd, Rn, #imm32 ; Rd = Rn and imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0

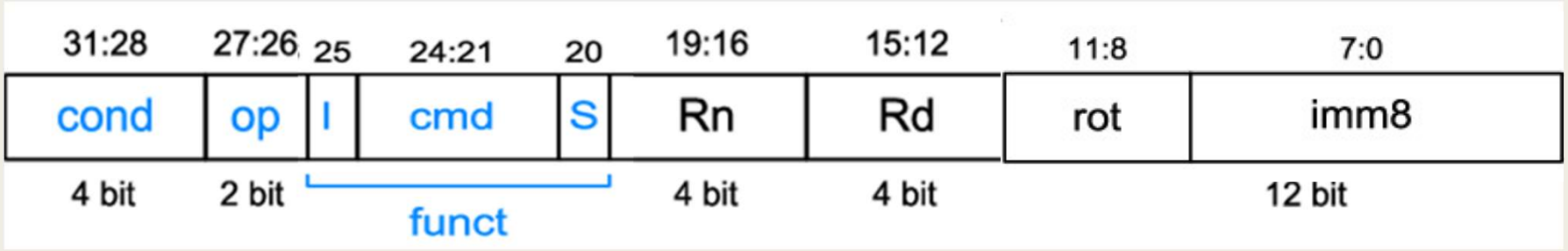


- Κώδικας συμβολικής γλώσσας:
  - *AND Rd, Rn, Rm; Rd = Rn and Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, shamt5 = 0, sh = 00

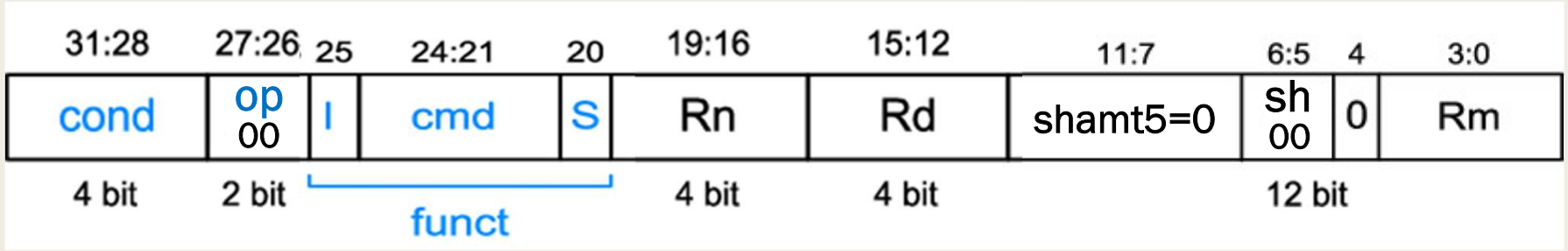


Γλώσσα μηχανής – Η εντολή ORR (cmd = 1100)

- Κώδικας συμβολικής γλώσσας:
  - *ORR Rd, Rn, #imm32 ; Rd = Rn or imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0

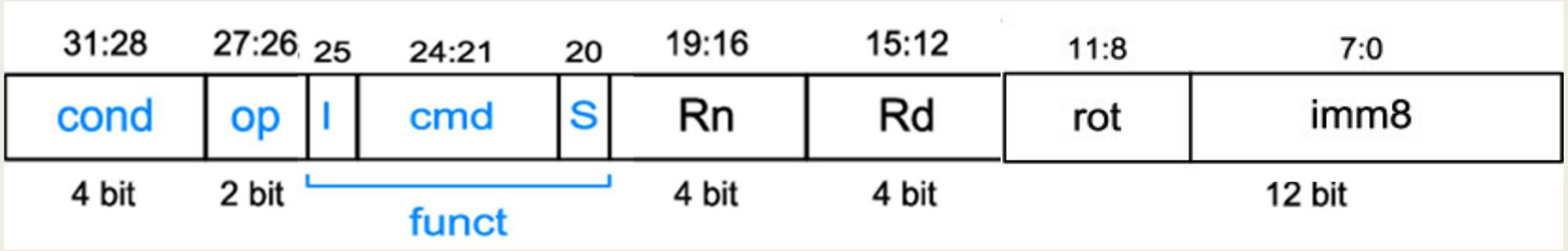


- Κώδικας συμβολικής γλώσσας:
  - *ORR Rd, Rn, Rm; Rd = Rn or Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, shamt5 = 0, sh = 00

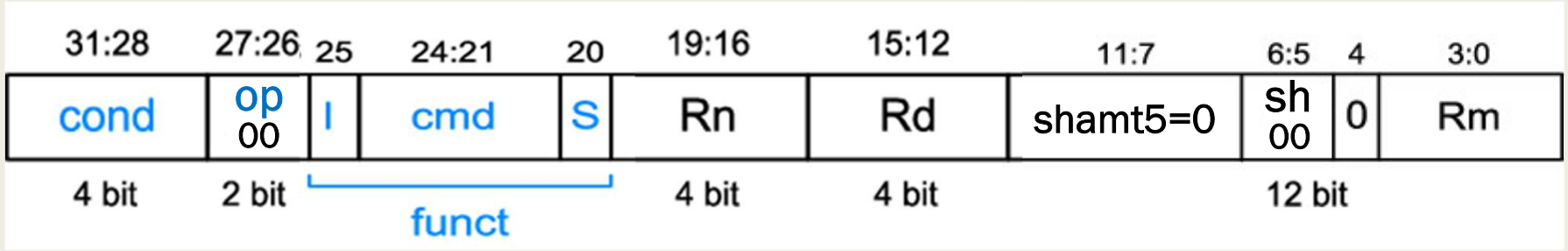


Γλώσσα μηχανής – Η εντολή EOR (cmd = 0001)

- Κώδικας συμβολικής γλώσσας:
  - *EOR Rd, Rn, #imm32 ; Rd = Rn xor imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0

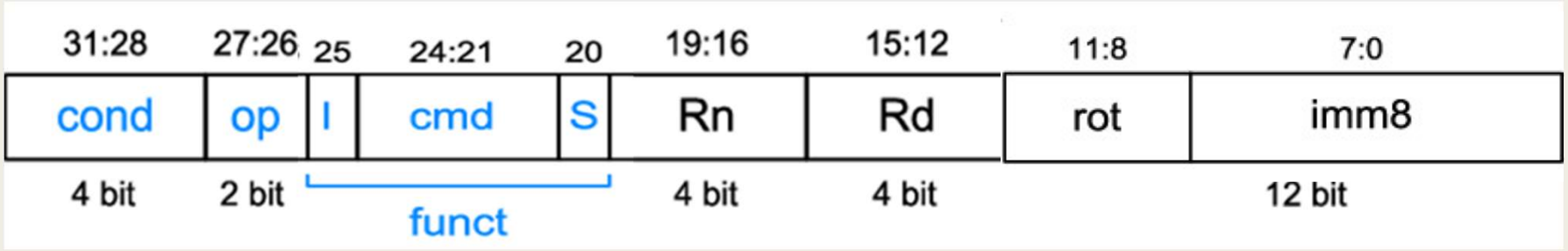


- Κώδικας συμβολικής γλώσσας:
  - *EOR Rd, Rn, Rm; Rd = Rn xor Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, shamt5 = 0, sh = 00

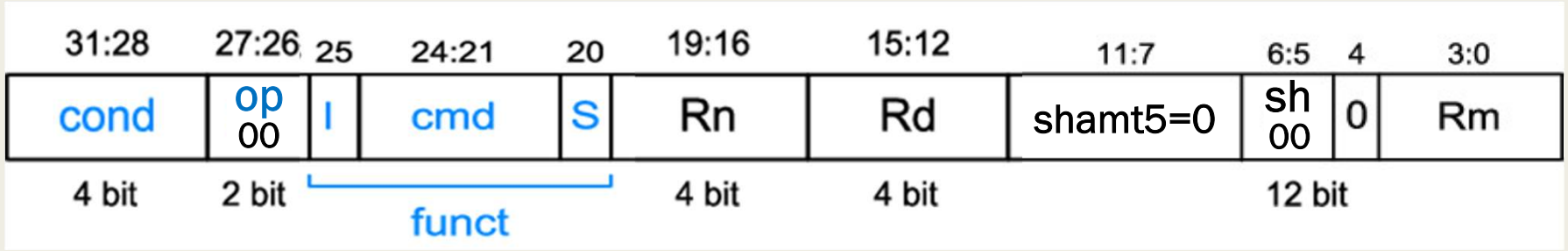


Γλώσσα μηχανής – Η εντολή MOV (cmd = 1101)

- Κώδικας συμβολικής γλώσσας:
  - *MOV Rd, #imm32 ; Rd = imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0, Rn = 0

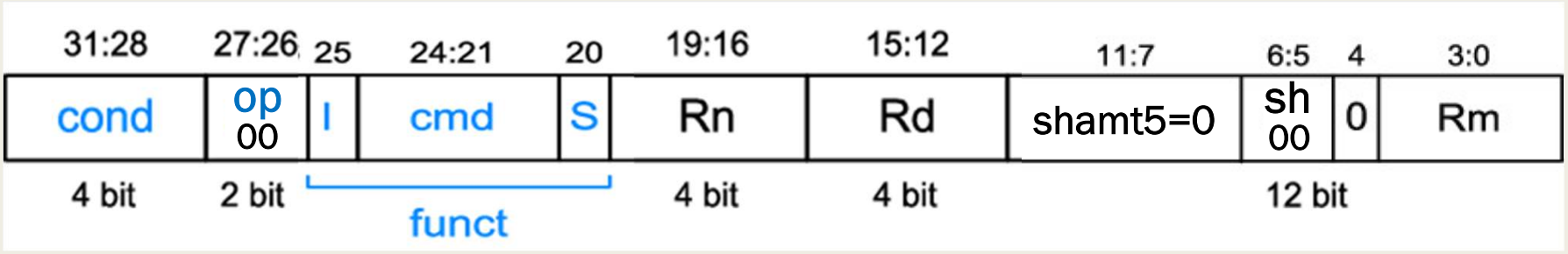


- Κώδικας συμβολικής γλώσσας:
  - *MOV Rd, Rm; Rd = Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, Rn = 0
  shamt5 = 0, sh = 00



Γλώσσα μηχανής – Η εντολή NOP (cmd = 1101)

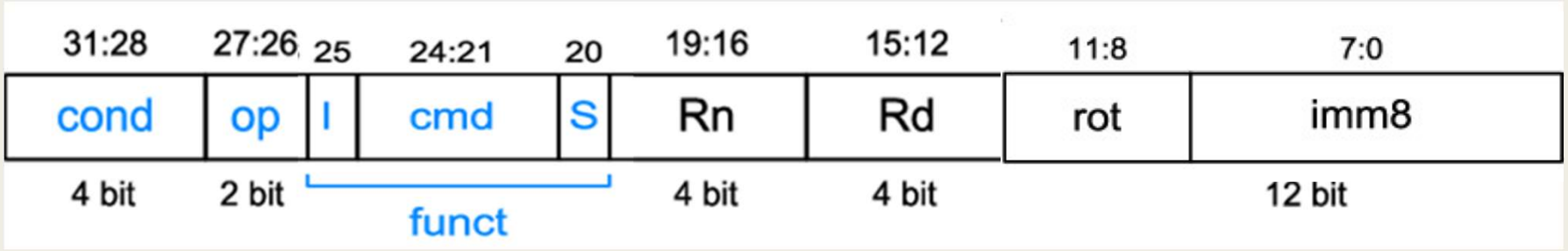
- Είναι ψευδοεντολή, που δεν εκτελεί κάποια πράξη, αλλά απλώς καθυστερεί την εκτέλεση του προγράμματος
- Κώδικας συμβολικής γλώσσας:
  - *NOP*
- Συνήθως αντιστοιχίζεται στην εντολή:
  - *MOV R0, R0; R0 = R0*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, Rn = 0, Rd = 0 shamt5 = 0, sh = 00, Rm =0



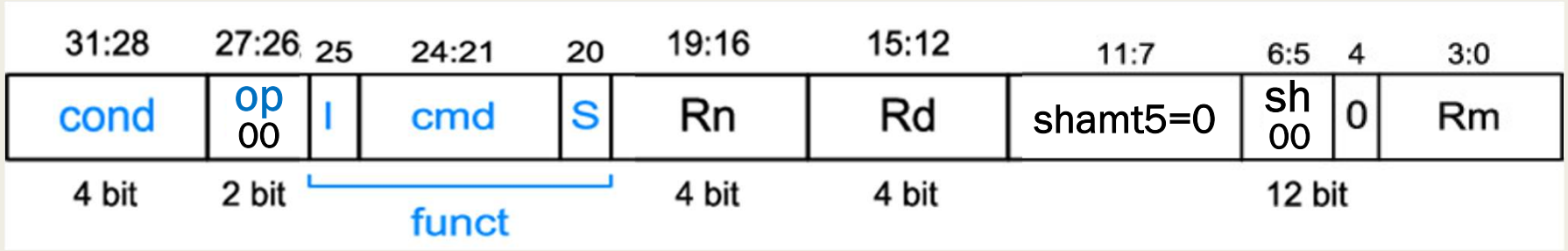
- Κωδικοποίηση:
  - *1110 00 0 1101 0 0000 0000 00000 00 0 0000*
  - *1110 0001 1010 0000 0000 0000 0000 0000 = **0xE1A00000***

Γλώσσα μηχανής – Η εντολή MVN (cmd = 1111)

- Κώδικας συμβολικής γλώσσας:
  - *MVN Rd, #imm32 ; Rd = not imm32→ rot, imm8*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0, Rn = 0

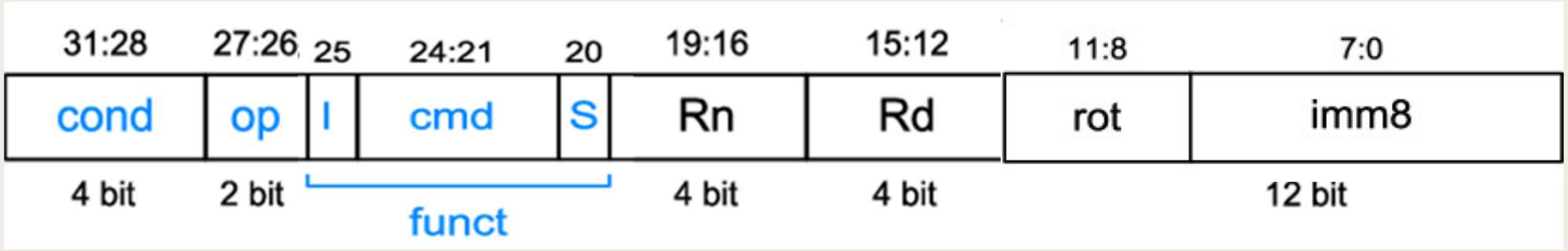


- Κώδικας συμβολικής γλώσσας:
  - *MVN Rd, Rm; Rd = not Rm*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, Rn = 0 shamt5 = 0, sh = 00

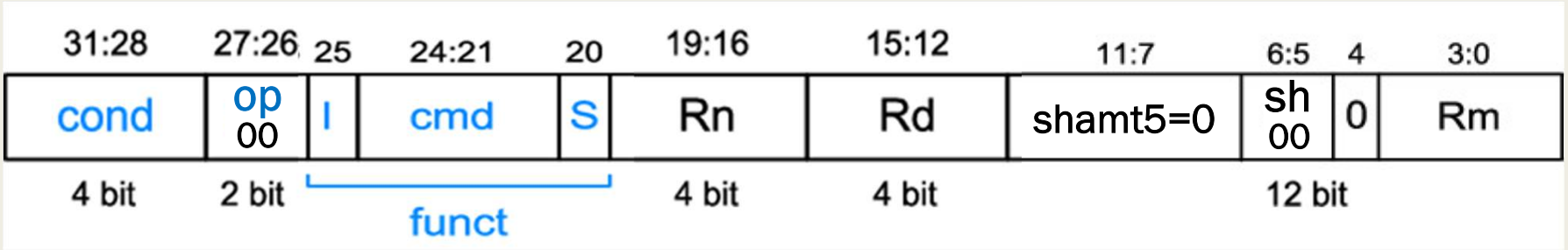


Γλώσσα μηχανής – Η εντολή BIC (cmd = 1110)

- Κώδικας συμβολικής γλώσσας:
  - *BIC Rd, Rn, #imm32 ; Rd = Rn and (not imm32→ rot, imm8)*
- Μορφή εντολής: cond = 1110, op = 00, I = 1, S = 0

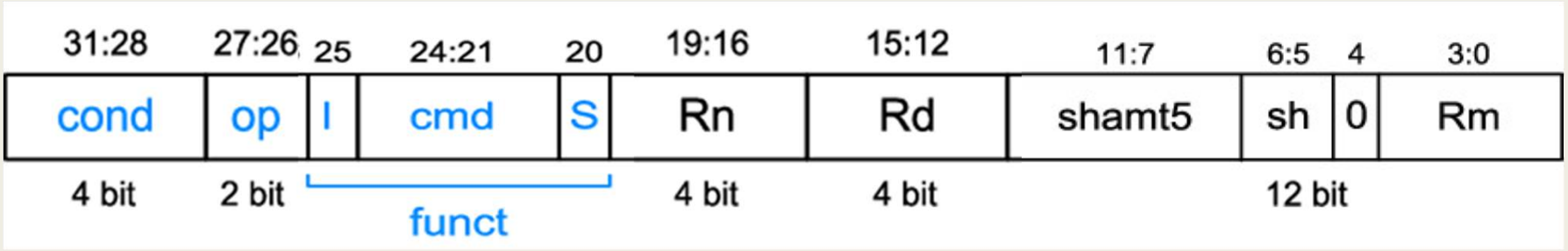


- Κώδικας συμβολικής γλώσσας:
  - *BIC Rd, Rn, Rm; Rd = Rn and (not Rm)*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0
  shamt5 = 0, sh = 00



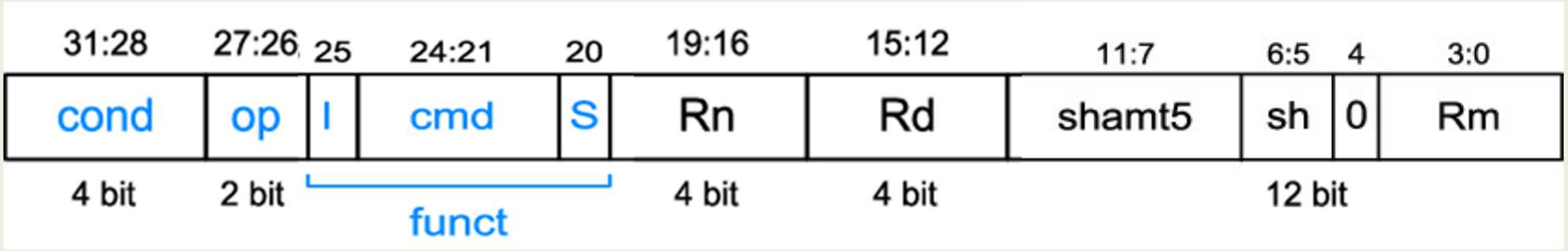
Γλώσσα μηχανής – Η εντολή LSL (cmd = 1101)

- Κώδικας συμβολικής γλώσσας:
  - *LSL Rd, Rm, #shamt5 ; Rd = Rm LSL by #shamt5*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, Rn = 0, sh = 00



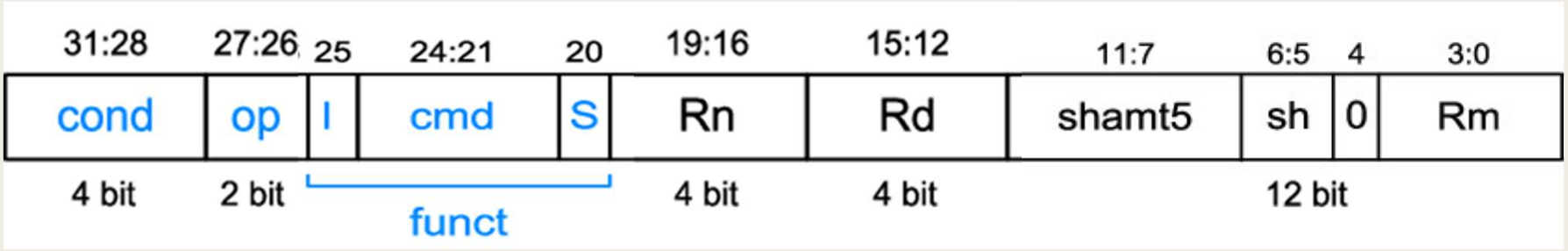
Γλώσσα μηχανής – Η εντολή LSR (cmd = 1101)

- Κώδικας συμβολικής γλώσσας:
  - *LSR Rd, Rm, #shamt5 ; Rd = Rm LSR by #shamt5*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, Rn = 0, sh = 01



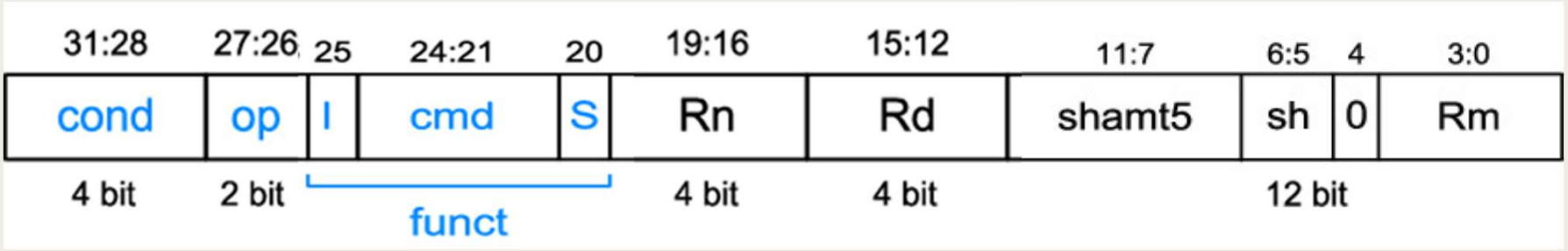
Γλώσσα μηχανής – Η εντολή ASR (cmd = 1101)

- Κώδικας συμβολικής γλώσσας:
  - *ASR Rd, Rm, #shamt5 ; Rd = Rm ASR by #shamt5*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, Rn = 0, sh = 10



Γλώσσα μηχανής – Η εντολή ROR (cmd = 1101)

- Κώδικας συμβολικής γλώσσας:
  - *ROR Rd, Rm, #shamt5 ; Rd = Rm ROR by #shamt5*
- Μορφή εντολής: cond = 1110, op = 00, I = 0, S = 0, Rn = 0, sh = 11



Γλώσσα μηχανής – Εντολές μνήμης

- Μορφή εντολής:

- Πεδία εντολής:
  - *Πεδίο funct (function, 6 bit, 25:20) για τη δήλωση της λειτουργίας/πράξης*
    - Υποπεδίο Ī (inverse immediate, 1 bit, 25) για τη δήλωση ύπαρξης άμεσου τελεστέου στο πεδίο src2 (Ī =0 άμεσος τελεστέος)
    - Υποπεδία P=1 (1 bit, 24) και W=0 (1 bit, 21) για τη δήλωση του τρόπου διευθυνσιοδότησης της μνήμης με σχετική απόσταση (offset addressing mode)
      - *Η αρχιτεκτονική ARM υποστηρίζει και άλλους τρόπους διευθυσιδότησης μνήμης (μετά-αριθμοδεικτοδότηση, πριν-αριθμοδεικτοδότηση)*
    - Υποπεδία B=0 (1 bit, 22) και L=1 (1 bit, 20) για την εντολή LDR
    - Υποπεδία B=0 (1 bit, 22) και L=0 (1 bit, 20) για την εντολή STR
    - Υποπεδίο U (1 bit, 23) για την πρόσθεση (U=1) ή αφαίρεση (U=0) της σχετικής απόστασης

Γλώσσα μηχανής – Εντολές μνήμης

- Μορφή εντολής:

- Πεδία εντολής:
  - *Πεδίο Rn (4 bit, 19:16) για τη δήλωση του καταχωρητή βάσης*
  - *Πεδίο Rd (4 bit, 15:12) για τη δήλωση:*
    - του καταχωρητή προορισμού στην εντολή LDR ή
    - του καταχωρητή προέλευσης στην εντολή STR
  - *Πεδίο imm12 (12 bit, 11:0) για τον προσδιορισμό σχετικής απόστασης ως άμεσος τελεστέος* ($\bar{I}$ =0)
    - Προσδιορίζεται ένας μη προσημασμένος ακέραιος αριθμός των 12 bit που προστίθεται στον ή αφαιρείται από τον καταχωρητή βάσης

Γλώσσα μηχανής – Η εντολή LDR

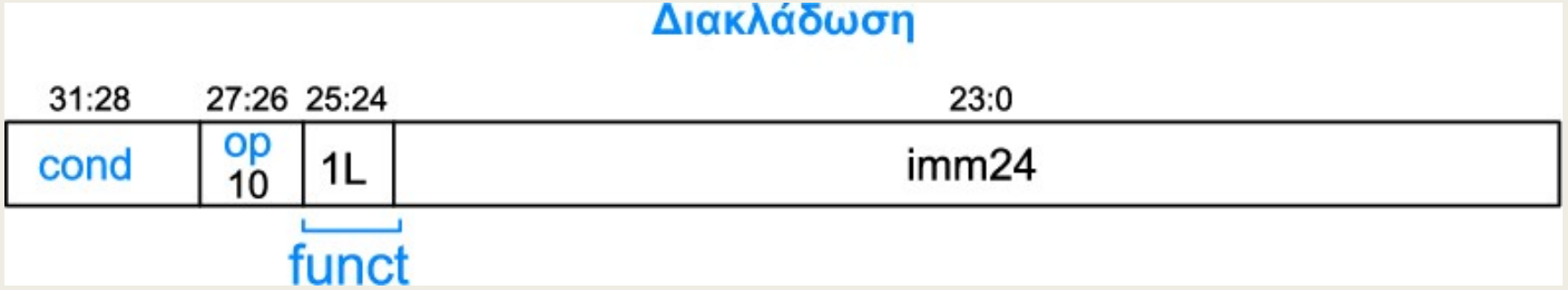
- Κώδικας συμβολικής γλώσσας:
  - *LDR Rd, [Rn, #imm12] ; Rd = mem[Rn + #imm12]*
- Μορφή εντολής: cond = 1110, op = 01, $\overline{I}$ = 0, P = 1, W = 0, B = 0, L = 1, U = 1 (funct = 011001)
- Κώδικας συμβολικής γλώσσας:
  - *LDR Rd, [Rn, #–imm12] ; Rd = mem[Rn – #imm12]*
- Μορφή εντολής: cond = 1110, op = 01, $\overline{I}$ = 0, P = 1, W = 0, B = 0, L = 1, U = 0 (funct = 010001)

Γλώσσα μηχανής – Η εντολή STR

- Κώδικας συμβολικής γλώσσας:
  - *STR Rd, [Rn, #imm12] ; mem[Rn – #imm12] = Rd*
- Μορφή εντολής: cond = 1110, op = 01, $\overline{I}$ = 0, P = 1, W = 0, B = 0, L = 0, U = 1 (funct = 011000)
- Κώδικας συμβολικής γλώσσας:
  - *STR Rd, [Rn, #–imm12] ; mem[Rn – #imm12] = Rd*
- Μορφή εντολής: cond = 1110, op = 01, $\overline{I}$ = 0, P = 1, W = 0, B = 0, L = 0, U = 0 (funct = 010000)

Γλώσσα μηχανής – Εντολές διακλάδωσης

- Μορφή εντολής:

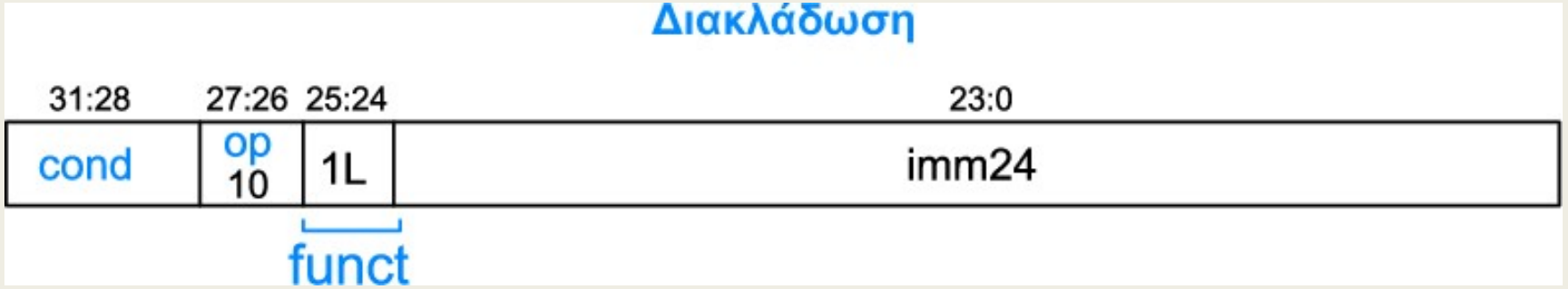


- Πεδία εντολής:
  - *Πεδίο L (link, 1 bit, 24) για τη δήλωση της υποστήριξης της διαδικασίας σύνδεσης (L = 1)*
  - *Πεδίο imm24 (24 bit, 23:0) για τον προσδιορισμό της διεύθυνσης - στόχου της διακλάδωσης (branch target address, BTA) ως άμεσος τελεστέος*
    - Προσδιορίζεται ένας προσημασμένος ακέραιος σε αναπαράσταση συμπληρώματος ως προς δύο των 24 bit που καθορίζει μία διεύθυνση-στόχο σε σχέση με το PC+8
    - Η διεύθυνση - στόχος της διακλάδωσης υπολογίζεται ως εξής:

      BTA = PC + 8 + imm24 x 4 (PC–σχετική διευθυνσιοδότηση)

Γλώσσα μηχανής – Η εντολή B

- Κώδικας συμβολικής γλώσσας:
  - *B label ; PC = BTA→ imm24*
- Μορφή εντολής: cond = 1110, op = 10, L= 0



Γλώσσα μηχανής – Η εντολή B

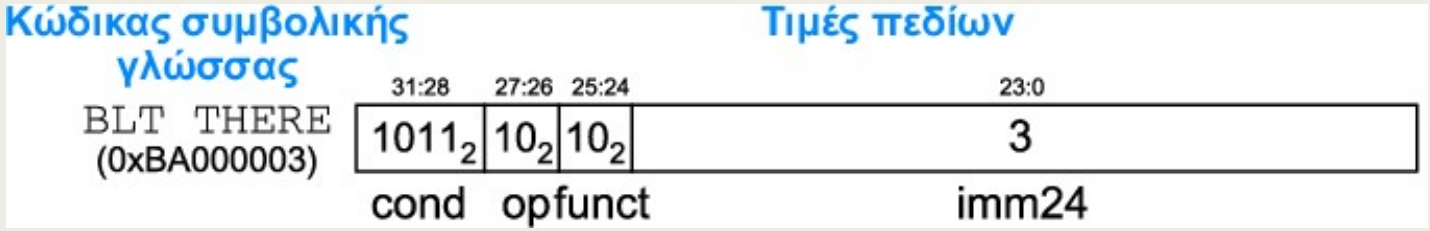
- Υπολογισμός του πεδίου imm24 μίας εντολής απλής διακλάδωσης

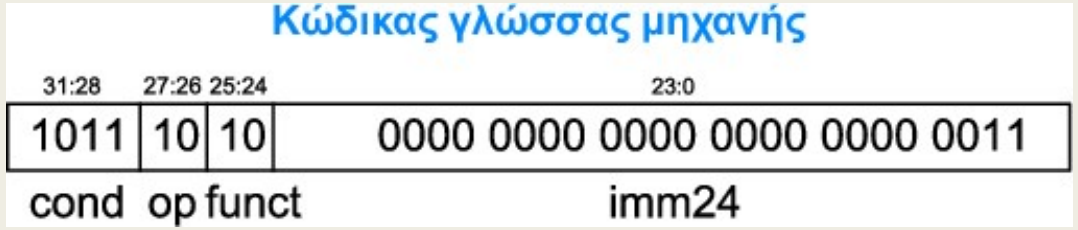
Κώδικας συμβολικής γλώσσας της ARM

```
0x80A0          BLT THERE          ; PC
0x80A4          ADD R0, R1, R2
0x80A8          SUB R0, R0, R9     ; PC + 8
0x80AC          ADD SP, SP, #8
0x80B0          MOV PC, LR
0x80B4 THERE    SUB R0, R0, #1     ; BTA
0x80B8          ADD R3, R3, #0x5
```

+3

- *imm24 = +3*

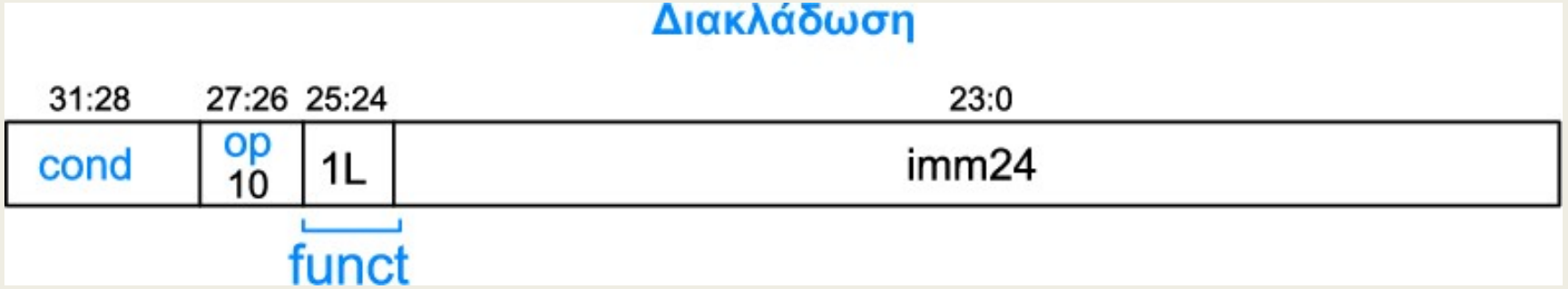




Προσοχή! Στο πεδίο imm24 αποθηκεύεται μία απόσταση μεταξύ εντολών

Γλώσσα μηχανής – Η εντολή BL

- Κώδικας συμβολικής γλώσσας:
  - *BL label ; LR = PC + 4 , PC = BTA → imm24*
- Μορφή εντολής: cond = 1110, op = 10, L= 1



Γλώσσα μηχανής – Η εντολή BL

- Υπολογισμός του πεδίου imm24 μίας εντολής διακλάδωσης με σύνδεση

Κώδικας συμβολικής γλώσσας της ARM

```
0x8040  TEST    LDRB R5, [R0, R3]
0x8044          STRB R5, [R1, R3]
0x8048          ADD  R3, R3, #1
0x804C          MOV  PC, LR
0x8050          BL   TEST
0x8054          LDR  R3, [R1], #4
0x8058          SUB  R4, R3, #9
```

−6

- imm24 = −6

Προσοχή! Στο πεδίο imm24 αποθηκεύεται μία απόσταση μεταξύ εντολών

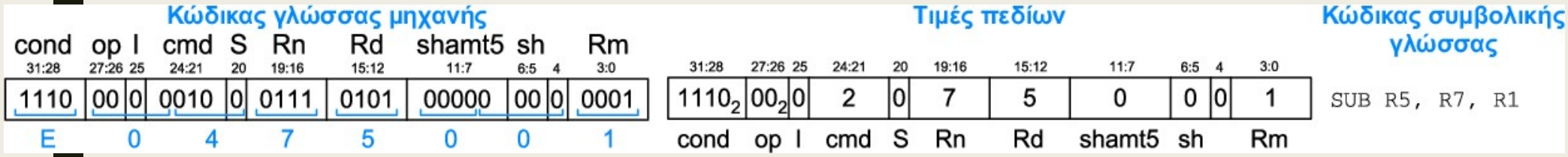
Τρόποι Διευθυνσιοδότησης

- Η αρχιτεκτονική ARM χρησιμοποιεί τέσσερεις κύριους τρόπους διευθυνσιοδότησης:
  1. Διευθυνσιοδότηση καταχωρητή (*register addressing*)
  2. Άμεση διευθυνσιοδότηση (*immediate addressing*)
  3. Διευθυνσιοδότηση βάσης (*base addressing*)
  4. PC–σχετική διευθυνσιοδότηση (*PC – relative addressing*)

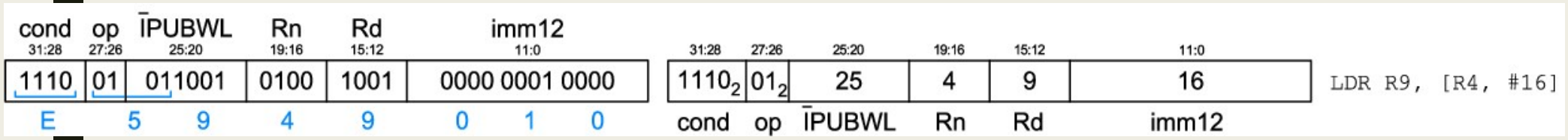
Τρόπος διευθυνσιοδότησης	Παράδειγμα	Περιγραφή
Διευθυνσιοδότηση καταχωρητή	ADD R3, R2, R1	R3 ← R2 + R1
Άμεση διευθυνσιοδότηση	SUB R3, R2, #25	R3 ← R2 – 25
Διευθυνσιοδότηση βάσης με σχετική απόσταση άμεσου τελεστέου	STR R6, [R11, #77]	mem[R11+77] ← R6
PC–σχετική διευθυνσιοδότηση	B LABEL1	Διακλάδωση στο LABEL1

Γλώσσα μηχανής – Ερμηνεία του κώδικα

- Για να ερμηνεύσει κανείς τη γλώσσα μηχανής, πρέπει πρώτα να αποκωδικοποιήσει τα πεδία κάθε λέξης εντολής των 32 bit
- Όλες οι μορφές εντολών ξεκινούν με ένα πεδίο συνθήκης ***cond*** των 4 bit και ένα πεδίο ***op*** των 2 bit.
  - *Πρώτα εξετάζουμε την τιμή του πεδίου op, ώστε να προσδιορισθούν και τα υπόλοιπα πεδία*
- *Παραδείγματα μετάφρασης γλώσσας μηχανής σε συμβολική γλώσσα:*
  - *0xE0475001 → 1110 0000 0100 0111 0101 0000 0000 0001*
    - Εντολή επεξεργασίας δεδομένων



  - *0xE5949010 → 1110 0101 1001 0100 1001 0000 0001 0000*
    - Εντολή μνήμης



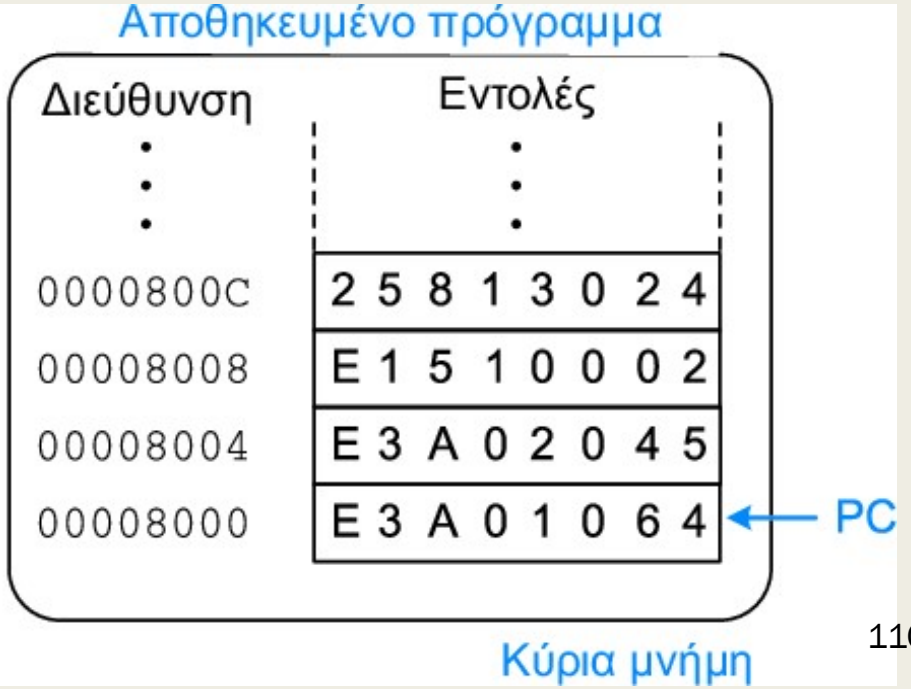
Γλώσσα μηχανής – Αποθηκευμένο πρόγραμμα

- Ένα πρόγραμμα γραμμένο σε γλώσσα μηχανής είναι μια σειρά από αριθμούς μεγέθους 32 bit που αναπαριστούν τις εντολές που μπορούν να αποθηκεύονται στη μνήμη.
  - *η έννοια του **αποθηκευμένου προγράμματος** (stored program)*
- Η εκτέλεση ενός διαφορετικού προγράμματος απαιτεί μονάχα την εγγραφή του νέου προγράμματος στη μνήμη
- Οι εντολές σε ένα αποθηκευμένο πρόγραμμα **προσκομίζονται** από τη μνήμη και εκτελούνται από τον επεξεργαστή
- Η διεύθυνση της τρέχουσας εντολής διατηρείται σε έναν καταχωρητή των 32 bit ο οποίος ονομάζεται **μετρητής προγράμματος** (program counter, PC), που είναι ο καταχωρητής R15
  - *Μια ανάγνωση του μετρητή PC επιστρέφει τη διεύθυνση της τρέχουσας εντολής συν 8 (PC + 8)!*

Γλώσσα μηχανής – Αποθηκευμένο πρόγραμμα

- Για να εκτελεστεί ο κώδικας της Εικόνας, ο μετρητής PC παίρνει αρχικά ως τιμή τη διεύθυνση 0x00008000
- Ο επεξεργαστής προσκομίζει την εντολή στη συγκεκριμένη διεύθυνση μνήμης και εκτελεί την εντολή 0xE3A01064 (MOV R1, #100)
- Κατόπιν αυξάνει τον μετρητή PC κατά 4 στην τιμή 0x00008004
- Ο επεξεργαστής προσκομίζει την εντολή στη συγκεκριμένη διεύθυνση μνήμης και εκτελεί την εντολή 0xE3A02045 (CMP R1, R2)
- Η διαδικασία επαναλαμβάνεται ...

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



Γλώσσα μηχανής – Αποθηκευμένο πρόγραμμα

- Στην αρχιτεκτονική κατάσταση (*architectural state*) ενός επεξεργαστή είναι αποθηκευμένη η κατάσταση ενός προγράμματος
- Στην ARM, η αρχιτεκτονική κατάσταση περιλαμβάνει το αρχείο καταχωρητών και καταχωρητές κατάστασης
  - *Αποθηκεύοντας και επαναφέροντας αργότερα την αρχιτεκτονική κατάσταση ενός προγράμματος, το πρόγραμμα δεν αντιλαμβάνεται ότι διακόπτεται η εκτέλεσή του*

Γλώσσα μηχανής – Ο χάρτης μνήμης

- Ο χώρος διευθύνσεων της αρχιτεκτονικής ARM καταλαμβάνει 2^{32} byte (4 GB) από 0 έως 0xFFFFFFFC
- Χωρίζεται σε 5 τμήματα:
  - *το τμήμα κειμένου, όπου* αποθηκεύεται το πρόγραμμα σε γλώσσα μηχανής
  - *το τμήμα καθολικών δεδομένων για στατικά δεδομένα, που δεσμεύεται στη μνήμη προτού ξεκινήσει η εκτέλεση του προγράμματος και προσπελαύνεται από τον δείκτη στατικής βάσης (static base pointer, SB) (συνήθως είναι ο R9)*
  - *το τμήμα δυναμικών δεδομένων, που δεσμεύονται και αποδεσμεύονται δυναμικά καθ' όλη τη διάρκεια της εκτέλεσης του προγράμματος και απαρτίζεται από τη στοίβα και τον σωρό*
  - *το τμήμα για χειριστές εξαιρέσεων που ξεκινάει από τη διεύθυνση 0x0*
  - *το τμήμα για το λειτουργικό σύστημα και την είσοδο/έξοδο* με απεικόνιση στη μνήμη