ΣΗΜΕΙΩΣΕΙΣ ΣΕ ΟΛΑ ΤΑ ΚΕΦΑΛΑΙΑ

COMPUTER ARCHITECTURE 1

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

1 ΑΦΗΡΗΜΕΝΕΣ ΕΝΝΟΙΕΣ

COMPUTER ARCHITECTURE 1

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

📌 **Εισαγωγή – Η Επανάσταση των Υπολογιστών (§1.1)**

Η ραγδαία πρόοδος των υπολογιστών έχει επιφέρει επανάσταση σε πλήθος τομέων της ζωής και της τεχνολογίας. Αυτή η πρόοδος:

- **Στηρίζεται στους εξειδικευμένους επιταχυντές** (accelerators), δηλαδή σε εξαρτήματα σχεδιασμένα να εκτελούν συγκεκριμένες εργασίες **πολύ πιο γρήγορα**.
- Επιτρέπει τη δημιουργία **καινοτόμων εφαρμογών** όπως:
  - ✅ Υπολογιστικά συστήματα στα **αυτοκίνητα**
  - ✅ **Κινητά τηλέφωνα**
  - ✅ Το **ερευνητικό πρόγραμμα του ανθρωπίνου γονιδιώματος** (Human Genome Project)
  - ✅ Ο **Παγκόσμιος Ιστός (World Wide Web)**
  - ✅ Οι **μηχανές αναζήτησης** (π.χ. Google)

📍 **Συμπέρασμα**: Η χρήση των υπολογιστών **εξαπλώνεται παντού**.

💻 **Κατηγορίες Υπολογιστών**

1Προσωπικοί Υπολογιστές (PCs)

- **Γενικής χρήσης** – μπορούν να εκτελέσουν πολλές διαφορετικές εργασίες.
- Χρησιμοποιούνται με **ποικιλία λογισμικού** (π.χ. επεξεργασία κειμένου, προγράμματα σχεδίασης, internet).
- Γίνονται **συμβιβασμοί** μεταξύ:
  - Κόστους 💰
  - Απόδοσης ⚡

2Διακομιστές (Servers)

- **Δικτυακά συστήματα** που εξυπηρετούν άλλους υπολογιστές ή συσκευές.
- Χαρακτηρίζονται από:
  - **Υψηλή παραγωγική δυναμικότητα**
  - Μεγάλη **χωρητικότητα** (σε δεδομένα)
  - **Αξιοπιστία** και **υψηλές επιδόσεις**
- Μπορούν να έχουν **διάφορα μεγέθη**:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

    - Από έναν μικρό υπολογιστή...
    - Μέχρι **ολόκληρα κτίρια** με χιλιάδες διακομιστές (π.χ. data centers)

3Υπερυπολογιστές (Supercomputers)

- Τεχνικά θεωρούνται **διακομιστές**, αλλά:
    - Προορίζονται για **υψηλής ακρίβειας επιστημονικούς και τεχνικούς υπολογισμούς**
    - Είναι **οι ισχυρότεροι υπολογιστές στον κόσμο** ως προς επεξεργαστική ισχύ
    - Παρά τη δύναμή τους, **αντιπροσωπεύουν μικρό μέρος της αγοράς**

🧠 Παραδείγματα χρήσης:

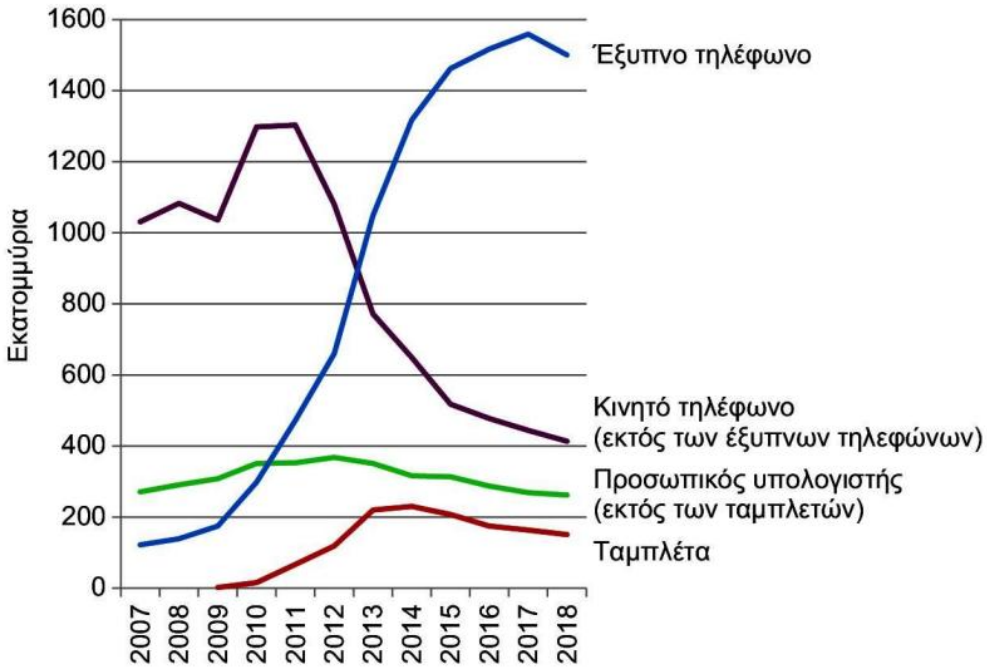
- Προσομοιώσεις κλιματικών μοντέλων
- Προβλέψεις καιρού
- Προσομοίωση ατομικών εκρήξεων

4Ενσωματωμένοι Υπολογιστές (Embedded Computers)

- **"Αόρατοι" ή κρυμμένοι** σε συσκευές:
    - Αυτοκίνητα 🚗
    - Οικιακές συσκευές 🧺
    - Ιατρικός εξοπλισμός 🏥
    - Βιομηχανικά συστήματα 🏭
- Έχουν αυστηρούς περιορισμούς:
    - **Ισχύος (κατανάλωση ενέργειας)**
    - **Απόδοσης**
    - **Κόστους**

📌 **Σημαντικό**: Οι ενσωματωμένοι υπολογιστές είναι **πολύ πιο πολυάριθμοι** από τους προσωπικούς ή υπερυπολογιστές!

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



📊 **Τι δείχνει το διάγραμμα:**

Ο **κατακόρυφος άξονας (y)** δείχνει τις **πωλήσεις συσκευών σε εκατομμύρια μονάδες**, ενώ ο **οριζόντιος άξονας (x)** τα **έτη από το 2007 έως το 2018**.

📈 **Ανάλυση των καμπυλών:**

1. **Έξυπνο τηλέφωνο (μπλε καμπύλη)**
   - Αρχικά χαμηλές πωλήσεις (~100 εκ. το 2007)
   - Εκρηκτική άνοδος από το 2010 και μετά
   - Ξεπέρασε τα **1400 εκατομμύρια μονάδες** γύρω στο 2016
   - Από το 2016 και μετά υπάρχει μια ελαφριά κάμψη
2. **Κινητό τηλέφωνο εκτός των έξυπνων (μοβ καμπύλη)**
   - Ξεκινάει με πάνω από **1000 εκατομμύρια μονάδες**
   - Κορυφώνεται γύρω στο 2011-2012
   - Έπειτα **κατακόρυφη πτώση**, κάτω από 400 εκατομμύρια το 2018
3. **Προσωπικός υπολογιστής (εκτός των ταμπλετών) – πράσινη καμπύλη**
   - Ήπια άνοδος μέχρι περίπου το 2011 (~350 εκατ.)
   - Σταθεροποίηση και **ελαφρά μείωση** τα επόμενα χρόνια
   - Πτώση κοντά στα 250 εκατ. το 2018

ΕΛΕΝΗ ΖΑΦΕΙ

4. **Ταμπλέτα (κόκκινη καμπύλη)**
    - Εμφανίζεται γύρω στο 2010
    - Απότομη άνοδος μέχρι το 2014 (~250 εκατ.)
    - Έπειτα **σταδιακή πτώση**

📌 **Συμπεράσματα:**

- Η **κυριαρχία των έξυπνων τηλεφώνων** είναι ξεκάθαρη. Από το 2012 και μετά ξεπερνούν όλες τις άλλες κατηγορίες.
- Η **αγορά των «παραδοσιακών» υπολογιστών και κινητών** παρουσιάζει **πτώση** ή στασιμότητα.
- Η **τάση της εποχής** μετατοπίζεται ξεκάθαρα προς τις **φορητές, έξυπνες, πάντα συνδεδεμένες συσκευές**.
- Ο όρος **«εποχή μετά τον PC»** αντικατοπτρίζει ακριβώς αυτή την **μετάβαση της υπολογιστικής ισχύος και χρήσης** από τα σταθερά ή φορητά PC προς τα **smartphones** και άλλες φορητές συσκευές.

📱 **Η «εποχή μετά τον PC»**

Προσωπική φορητή συσκευή (Personal Mobile Device – PMD)

(Κεφάλαιο 1 – Αφηρημένες έννοιες και τεχνολογία υπολογιστών – Διαφάνεια 6)

- **Λειτουργούν με μπαταρία**
- **Συνδέονται στο Διαδίκτυο** μέσω WiFi ή κινητών δικτύων
- **Κοστίζουν εκατοντάδες ευρώ** ανάλογα με τις δυνατότητες
- Παραδείγματα:
    - Έξυπνα τηλέφωνα (smartphones)
    - Υπολογιστές-ταμπλέτες (tablets)
    - Ηλεκτρονικά γυαλιά (π.χ. Google Glass)

☁️ **Υπολογιστική νέφους (Cloud Computing)**

- **Υπολογιστές κλίμακας αποθήκης** (Warehouse-Scale Computers – WSC):
    - Τεράστια κέντρα δεδομένων με χιλιάδες servers
- **Λογισμικό ως υπηρεσία** (Software as a Service – SaaS):

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

    - Το λογισμικό εκτελείται μερικώς στο Νέφος και μερικώς στη συσκευή του χρήστη
- Παραδείγματα:
    - Amazon Web Services (AWS)
    - Google Cloud Platform (GCP)

💡 *Παρατήρηση*: Το SaaS επιτρέπει εφαρμογές υψηλής απόδοσης ακόμα και σε φορητές συσκευές χαμηλής ισχύος, καθώς μέρος της επεξεργασίας γίνεται στο σύννεφο.

📘 **Τι θα μάθετε**

- Πώς τα **προγράμματα μεταφράζονται** από γλώσσες υψηλού επιπέδου σε **γλώσσα μηχανής**
- Πώς αυτά **εκτελούνται από το υλικό**
- Πώς αλληλεπιδρούν **υλικό και λογισμικό**
- Ποιοι **παράγοντες επηρεάζουν την απόδοση** ενός προγράμματος
- Πώς μπορεί αυτή να **βελτιωθεί**
- Πώς οι **σχεδιαστές υλικού** (π.χ. επεξεργαστών) αυξάνουν την ταχύτητα
- Τι είναι η **παράλληλη επεξεργασία** και πώς επιταχύνει τις εφαρμογές

⏱️ **Κατανόηση της απόδοσης**

Για να καταλάβουμε την απόδοση ενός υπολογιστικού συστήματος, πρέπει να εξετάσουμε:

1. Αλγόριθμος

- Καθορίζει **πόσες λειτουργίες** (π.χ. προσθέσεις, αναζητήσεις) θα γίνουν.

2. Γλώσσα προγραμματισμού – Μεταγλωττιστής – Αρχιτεκτονική

- Επηρεάζουν **πόσες εντολές γλώσσας μηχανής** απαιτούνται για κάθε λειτουργία.

3. Επεξεργαστής και Σύστημα Μνήμης

- Καθορίζουν **πόσο γρήγορα εκτελούνται** οι εντολές.

4. Σύστημα Εισόδου/Εξόδου (I/O)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Μαζί με το λειτουργικό σύστημα, καθορίζει **την ταχύτητα των λειτουργιών εισόδου/εξόδου**, όπως η ανάγνωση αρχείων ή η επικοινωνία με δίκτυα.

§1.2 – Επτά σπουδαίες ιδέες στην αρχιτεκτονική υπολογιστών

1. **Χρήση αφαιρέσεων για την απλοποίηση της σχεδίασης**
   - Η αφαίρεση επιτρέπει στους σχεδιαστές να εργάζονται σε υψηλότερο επίπεδο, χωρίς να ασχολούνται με όλες τις λεπτομέρειες του υλικού.
2. **Επιτάχυνση της πιο κοινής (συνηθισμένης) περίπτωσης**
   - Οι συχνότερες λειτουργίες βελτιστοποιούνται για καλύτερη απόδοση (π.χ. εντολές, αναζητήσεις στη μνήμη).
3. **Απόδοση μέσω παραλληλίας**
   - Πολλές εργασίες εκτελούνται ταυτόχρονα (π.χ. πολυπύρηνοι επεξεργαστές).
4. **Απόδοση μέσω διοχέτευσης (pipelining)**
   - Οι εντολές εκτελούνται σε «στάδια», όπως σε γραμμή παραγωγής, επιτρέποντας πολλές να βρίσκονται ταυτόχρονα υπό επεξεργασία.
5. **Απόδοση μέσω πρόβλεψης**
   - Ο επεξεργαστής προβλέπει ποια εντολή θα ακολουθήσει (π.χ. σε διακλαδώσεις), και την εκτελεί προκαταβολικά.
6. **Ιεραρχία μνημών**
   - Γρήγορες αλλά μικρές (cache) και αργές αλλά μεγάλες (RAM/δίσκος) μνήμες συνεργάζονται για ταχύτερη πρόσβαση.
7. **Φερεγγυότητα μέσω πλεονασμού (reliability via redundancy)**
   - Εφεδρικά υποσυστήματα ή έλεγχοι σφαλμάτων εξασφαλίζουν την αξιοπιστία των υπολογιστικών συστημάτων.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔷 **§1.3 – Κάτω από το πρόγραμμά σας**

📌 **Τι υπάρχει «κάτω» από τον κώδικα που γράφεις:**

1. **Λογισμικό εφαρμογών**
   - Π.χ. προγράμματα σε C, Java, Python. Γραμμένα σε γλώσσα υψηλού επιπέδου.
2. **Λογισμικό συστημάτων**
   - **Μεταγλωττιστής**: Μεταφράζει τον πηγαίο κώδικα σε εντολές κατανοητές από το υλικό (γλώσσα μηχανής).
   - **Λειτουργικό σύστημα (OS)**: Παρέχει υπηρεσίες για:
     - Είσοδο/έξοδο
     - Διαχείριση μνήμης
     - Διαμοιρασμό πόρων
     - Προγραμματισμό και εκτέλεση πολλών διεργασιών
3. **Υλικό (Hardware)**
   - **Επεξεργαστής (CPU)**: Εκτελεί εντολές
   - **Μνήμη (RAM, cache)**

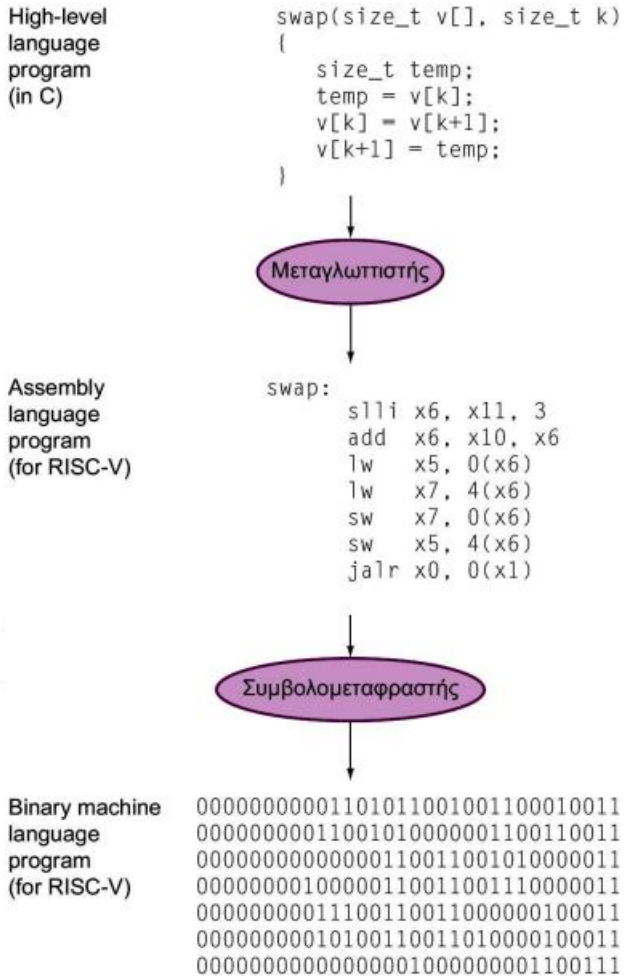
ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Ελεγκτές Εισόδου/Εξόδου (I/O controllers)**

Επίπεδα κώδικα προγραμμάτων

1. **Γλώσσα υψηλού επιπέδου**
   - Πιο κοντά στη γλώσσα του ανθρώπου
   - Παραγωγική και φορητή (π.χ. C, Java, Python)
2. **Συμβολική γλώσσα (Assembly language)**
   - Κείμενο που περιγράφει εντολές της γλώσσας μηχανής
   - Εξαρτάται από την αρχιτεκτονική του επεξεργαστή
3. **Γλώσσα μηχανής (Binary)**
   - Η πραγματική αναπαράσταση του προγράμματος στο υλικό
   - Δυαδικές ακολουθίες από **0 και 1**, που εκτελούνται από τον επεξεργαστή

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



🧩 §1.4 – Κάτω από το κάλυμμα

🔧 Τα συστατικά στοιχεία του υπολογιστή

🖥️ **Όλοι οι υπολογιστές, ανεξαρτήτως τύπου, περιλαμβάνουν τα ίδια βασικά συστατικά:**

- **Επιτραπέζιοι υπολογιστές (PCs)**
- **Διακομιστές (Servers)**
- **Ενσωματωμένοι υπολογιστές (Embedded systems)**
  (π.χ. υπολογιστές σε αυτοκίνητα, κινητά, έξυπνες συσκευές κ.λπ.)

🔄 **Είσοδος / Έξοδος (I/O)**

Η είσοδος/έξοδος περιλαμβάνει όλα τα στοιχεία που επιτρέπουν στον υπολογιστή να επικοινωνεί με το εξωτερικό περιβάλλον (χρήστες ή άλλα συστήματα):

🧍 **Συσκευές διασύνδεσης με τον χρήστη:**

- **Οθόνη**
- **Πληκτρολόγιο**
- **Ποντίκι**

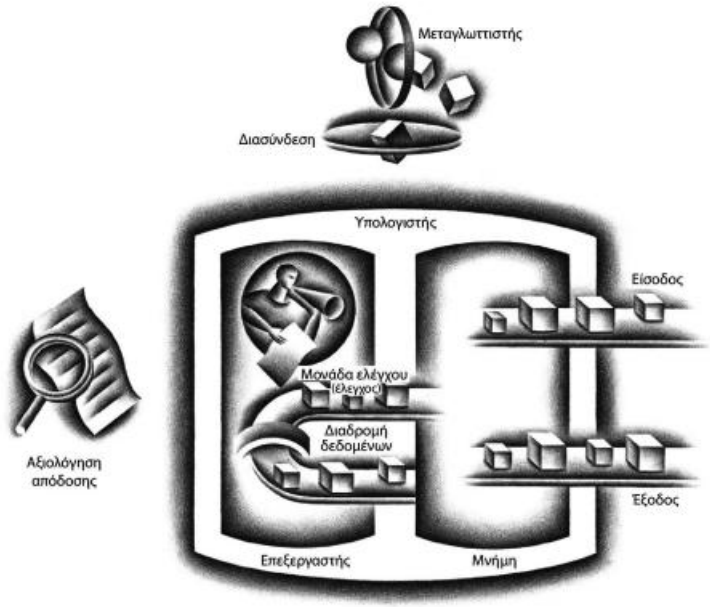
💾 **Συσκευές αποθήκευσης:**

- **Σκληροί δίσκοι (HDD)**
- **CD/DVD**
- **Μονάδες flash (USB, SSD κ.λπ.)**

🌐 **Προσαρμογείς δικτύου:**

- Εξασφαλίζουν **επικοινωνία με άλλους υπολογιστές**
- Ενσύρματοι (Ethernet) ή ασύρματοι (Wi-Fi)

📌 *Συμπέρασμα*:
Ανεξάρτητα από το είδος ή το μέγεθος του υπολογιστή, τα βασικά υποσυστήματα (επεξεργαστής, μνήμη, είσοδος/έξοδος) είναι τα ίδια — απλώς διαφέρουν σε **κλίμακα**, **ταχύτητα**, και **σκοπό χρήσης**.



ΗΛΖΑΦΕΙΡΗ

❓ **Τι είναι η οθόνη αφής;**
Η οθόνη αφής είναι μια συσκευή της «εποχής μετά τον PC» που επιτρέπει στον χρήστη να αλληλεπιδρά απευθείας με την οθόνη, **αντικαθιστώντας το πληκτρολόγιο και το ποντίκι**.

🧩 **Ποιοι είναι οι βασικοί τύποι τεχνολογίας στις οθόνες αφής;**
Υπάρχουν δύο βασικοί τύποι:

- **Τεχνολογία πίεσης**
- **Τεχνολογία αίσθησης χωρητικότητας**

📱 **Ποια τεχνολογία χρησιμοποιείται κυρίως σε smartphones και tablets;**
Η **τεχνολογία αίσθησης χωρητικότητας** είναι αυτή που χρησιμοποιείται περισσότερο σε έξυπνα τηλέφωνα και υπολογιστές-ταμπλέτες.

👆 **Ποιο είναι το πλεονέκτημα της τεχνολογίας αίσθησης χωρητικότητας;**
Η συγκεκριμένη τεχνολογία **επιτρέπει πολλά αγγίγματα ταυτόχρονα (multi-touch)**, διευκολύνοντας τη χρήση χειρονομιών όπως μεγέθυνση, σμίκρυνση και περιστροφή.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

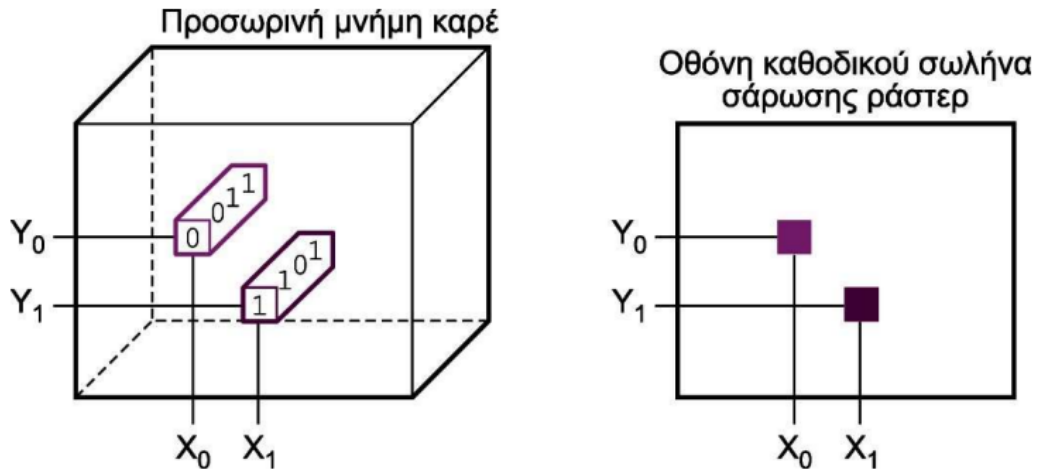
🖥️ **Τι είναι η οθόνη LCD και ποια είναι η βασική μονάδα απεικόνισης της;**
Η οθόνη LCD (Liquid Crystal Display) αποτελείται από **εικονοστοιχεία (pixels)** — αγγλιστί: *picture elements*.

🔁 **Πώς δημιουργείται η εικόνα στην οθόνη LCD;**
Η εικόνα που προβάλλεται στην οθόνη **αντικατοπτρίζει το περιεχόμενο της προσωρινής μνήμης καρέ (frame buffer)**. Δηλαδή, κάθε pixel στην οθόνη αντιστοιχεί σε δεδομένα που είναι αποθηκευμένα σε συγκεκριμένη θέση της μνήμης.

📊 **Τι δείχνει το διάγραμμα με την «Προσωρινή μνήμη καρέ» και την «Οθόνη καθοδικού σωλήνα σάρωσης ράστερ»;**
Το διάγραμμα δείχνει τη συσχέτιση μεταξύ των θέσεων στη μνήμη (π.χ. X0,Y0X_0, Y_0X0,Y0) και των pixel που εμφανίζονται στην οθόνη.

- Στην **προσωρινή μνήμη καρέ**, κάθε θέση αποθηκεύει μια δυαδική τιμή που καθορίζει το χρώμα ή την ένταση ενός pixel.
- Στην **οθόνη καθοδικού σωλήνα (CRT)** ή LCD, αυτές οι τιμές προβάλλονται ως οπτικά pixel σε αντίστοιχες θέσεις.

📌 **Συμπέρασμα:**
Η LCD οθόνη **λειτουργεί σαν καθρέφτης** των δεδομένων που είναι αποθηκευμένα σε μια ειδική περιοχή μνήμης, επιτρέποντας την απεικόνιση γραφικών και εικόνας μέσω της διαχείρισης pixel.



ΗΛΙΑΣ ΖΑΦΕΙΡΗ

📦 **Τι σημαίνει «άνοιγμα του κουτιού» σε τεχνολογικές συσκευές;**
Αναφέρεται στη διαδικασία **αποσυναρμολόγησης μιας συσκευής**, προκειμένου να εξεταστούν τα εσωτερικά της εξαρτήματα και κυκλώματα.

🔧 **Ποια βασικά εξαρτήματα περιέχει το εσωτερικό ενός σύγχρονου smartphone;**
Η εικόνα δείχνει διάφορα εξαρτήματα όπως:

- **Οθόνη**
- **Μπαταρία**
- **Μητρική πλακέτα**
- **Κάμερες**
- **Καλώδια σύνδεσης (flex cables)**

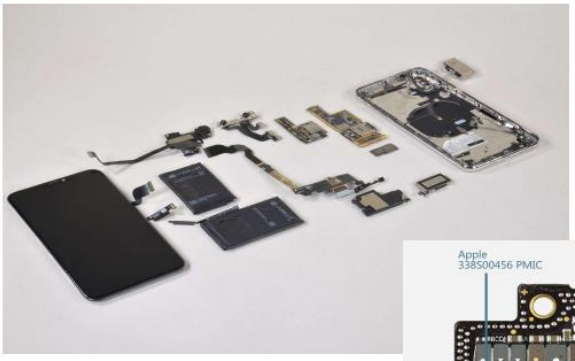
💻 **Τι περιλαμβάνει η μητρική πλακέτα (logic board);**
Η μητρική πλακέτα είναι το βασικό κύκλωμα της συσκευής και περιλαμβάνει:

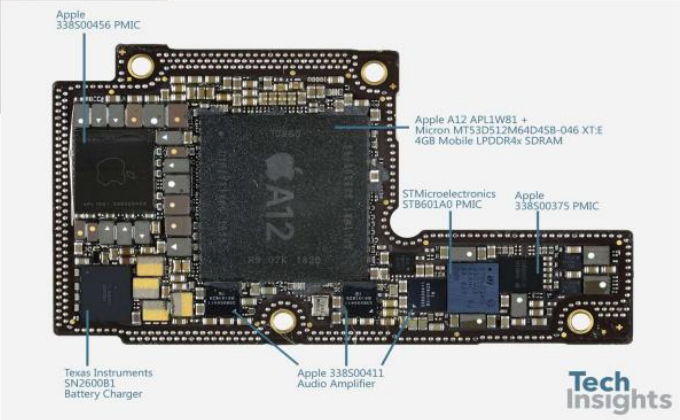
🔹 **Apple A12 (AP1L1W81)** – ο κύριος επεξεργαστής (SoC)
🔹 **Micron MT53D512M64D4SB-046 XT:E** – μνήμη LPDDR4x SDRAM (4GB)
🔹 **Apple 338S00456 PMIC** – κύκλωμα διαχείρισης ισχύος
🔹 **STMicroelectronics STB601A0 PMIC** – επιπλέον μονάδα διαχείρισης ισχύος
🔹 **Texas Instruments SN2600B1** – φορτιστής μπαταρίας
🔹 **Apple 338S00411** – ενισχυτής ήχου (Audio Amplifier)
🔹 **Apple 338S00375 PMIC** – ακόμη ένα εξάρτημα διαχείρισης ενέργειας

🔍 **Ποια είναι η σημασία αυτής της ανάλυσης;**
Το "teardown" (άνοιγμα) επιτρέπει στους ειδικούς:

- να κατανοήσουν τη σύνθεση της συσκευής,
- να αξιολογήσουν την ποιότητα και τις τεχνολογίες,
- να κάνουν έρευνα κόστους ή επισκευών,
- να συγκρίνουν με άλλες συσκευές.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ





🧠 **Τι περιέχει το εσωτερικό ενός επεξεργαστή (CPU);**
Ο επεξεργαστής περιλαμβάνει βασικά υποσυστήματα που συνεργάζονται για την εκτέλεση εντολών και επεξεργασία δεδομένων:

📊 **Τι είναι η διαδρομή δεδομένων (datapath);**
Η διαδρομή δεδομένων είναι υπεύθυνη για την **εκτέλεση λειτουργιών πάνω στα δεδομένα**. Περιλαμβάνει αριθμητικές μονάδες, καταχωρητές και κυκλώματα ελέγχου δεδομένων.

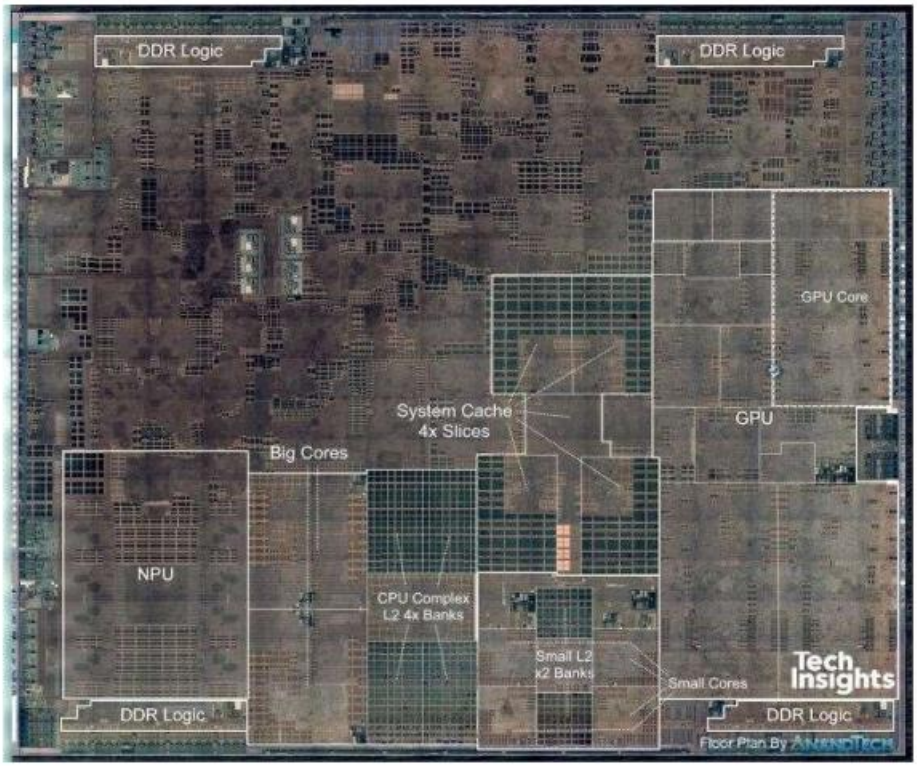
🔢 **Τι κάνει η μονάδα ελέγχου (control unit);**
Η μονάδα ελέγχου καθορίζει **την αλληλουχία λειτουργίας** του επεξεργαστή, δίνοντας σήματα για το πώς και πότε να λειτουργούν η διαδρομή δεδομένων, η μνήμη και άλλα τμήματα.

⚡ **Τι είναι η κρυφή μνήμη (cache memory);**
Η κρυφή μνήμη είναι **μικρή αλλά γρήγορη μνήμη τύπου SRAM**, που επιτρέπει την **άμεση προσπέλαση δεδομένων** που χρησιμοποιούνται συχνά, μειώνοντας την ανάγκη πρόσβασης στη βασική μνήμη RAM.

🧬 **Παράδειγμα σύγχρονου επεξεργαστή: A12**
Ο **Apple A12** είναι ένας σύγχρονος επεξεργαστής που ενσωματώνει όλα τα

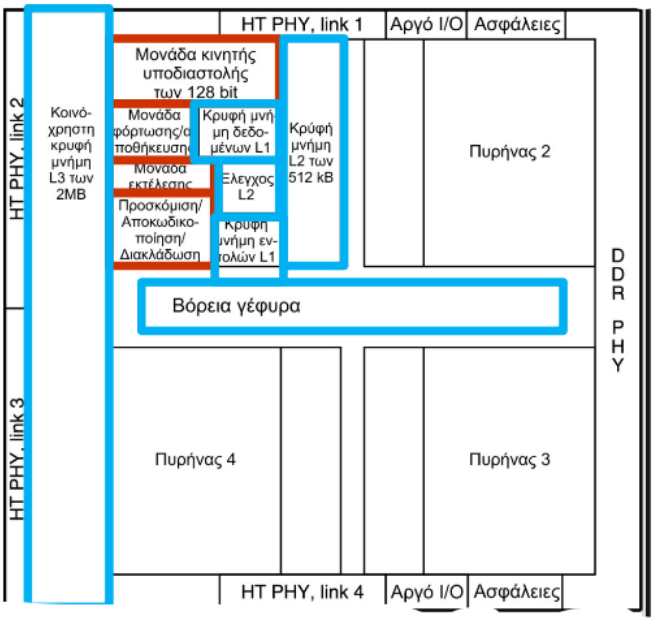
ΕΛΕΝΗ ΖΑΦΕΙ

παραπάνω στοιχεία (datapath, control, cache) και περισσότερα, προσφέροντας υψηλή απόδοση και ενεργειακή αποδοτικότητα σε φορητές συσκευές.



Αρχιτεκτονική I και II

- **Αρχιτεκτονική I**
- σύνολο εντολών, επεξεργαστής
- **Αρχιτεκτονική II**
- προηγμένες τεχνικές επεξεργαστή, μνήμες και είσοδος/έξοδος



🔍 **Τι είναι η αφαίρεση (abstraction) και ποιος ο ρόλος της στην τεχνολογία υπολογιστών;**

Η αφαίρεση είναι μια έννοια που μας επιτρέπει να **αντιμετωπίζουμε την πολυπλοκότητα** κρύβοντας τις χαμηλού επιπέδου λεπτομέρειες. Αυτό μας βοηθά να σχεδιάζουμε και να διαχειριζόμαστε πολύπλοκα συστήματα με μεγαλύτερη ευκολία.

🧱 **Ποιες μορφές αφαίρεσης υπάρχουν σε ένα υπολογιστικό σύστημα;**

- **Αρχιτεκτονική συνόλου εντολών (ISA)**
  ➤ Η διασύνδεση ανάμεσα στο υλικό (hardware) και το λογισμικό (software).
  ➤ Καθορίζει ποιες εντολές μπορεί να εκτελέσει ο επεξεργαστής.
- **Δυαδική διασύνδεση εφαρμογής (ABI)**
  ➤ Η ISA μαζί με τη διασύνδεση του συστήματος λογισμικού, επιτρέποντας την **εκτέλεση δυαδικών προγραμμάτων σε συγκεκριμένο υλικό**.

🛠️ **Τι είναι η υλοποίηση (implementation) σε αυτό το πλαίσιο;**

Η υλοποίηση αναφέρεται στις **χαμηλότερου επιπέδου λεπτομέρειες** του συστήματος και στον τρόπο που πραγματοποιείται πρακτικά η λειτουργία της αρχιτεκτονικής — δηλαδή, **το "πώς" λειτουργεί πίσω από τη σκηνή**.

💾 **Πού αποθηκεύονται τα δεδομένα σε έναν υπολογιστή;**

Υπάρχουν δύο βασικοί τύποι αποθήκευσης:

⚡ **Τι είναι η πτητική (volatile) κύρια μνήμη;**

Η πτητική μνήμη (όπως η RAM):

- Αποθηκεύει εντολές και δεδομένα προσωρινά
- Χάνει το περιεχόμενό της όταν διακοπεί η τροφοδοσία ρεύματος

💾 **Τι είναι η μη πτητική (non-volatile) δευτερεύουσα μνήμη;**

Η μη πτητική μνήμη **διατηρεί τα δεδομένα ακόμα και χωρίς ρεύμα**. Περιλαμβάνει:

- **Μαγνητικούς δίσκους (HDDs)**
- **Μνήμη φλας (Flash memory)**
- **Οπτικά μέσα** όπως CD-ROM και DVD

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🌐 **Τι είναι τα δίκτυα υπολογιστών και ποιος είναι ο σκοπός τους;**

Τα δίκτυα επιτρέπουν:

- **Επικοινωνία** μεταξύ συσκευών
- **Κοινή χρήση πόρων** (όπως εκτυπωτές ή αρχεία)
- **Πρόσβαση σε απομακρυσμένους πόρους** (non-local access)

🏠 **Τι είναι το τοπικό δίκτυο (LAN);**

Ένα **τοπικό δίκτυο (Local Area Network)** συνδέει υπολογιστές και συσκευές σε έναν περιορισμένο χώρο (π.χ. σπίτι ή γραφείο).
➡️ Τυπικό παράδειγμα: **Ethernet**

🌍 **Τι είναι το δίκτυο ευρείας περιοχής (WAN);**

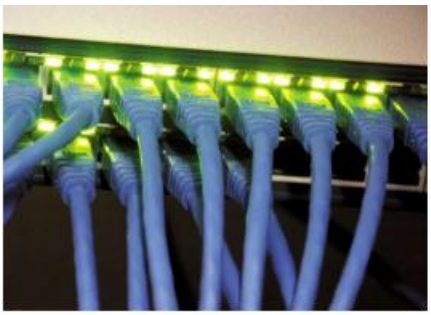
Το **WAN (Wide Area Network)** συνδέει υπολογιστές σε πολύ μεγάλες αποστάσεις.
➡️ Κύριο παράδειγμα: **Το Διαδίκτυο (Internet)**

📡 **Τι είναι τα ασύρματα δίκτυα;**

Επιτρέπουν σύνδεση χωρίς καλώδια. Παραδείγματα:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **WiFi**: για ασύρματη πρόσβαση στο διαδίκτυο
- **Bluetooth**: για σύνδεση συσκευών μικρής εμβέλειας (π.χ. ακουστικά, πληκτρολόγια)



📈 Τάσεις της τεχνολογίας υπολογιστών

🚀 Πώς εξελίσσεται η τεχνολογία των ηλεκτρονικών;

Η τεχνολογία προχωρά συνεχώς, με:

- **Αύξηση χωρητικότητας** (π.χ. μνήμης, υπολογιστικής ισχύος)
- **Αύξηση απόδοσης** (ταχύτητα, ενέργεια)
- **Μείωση κόστους**

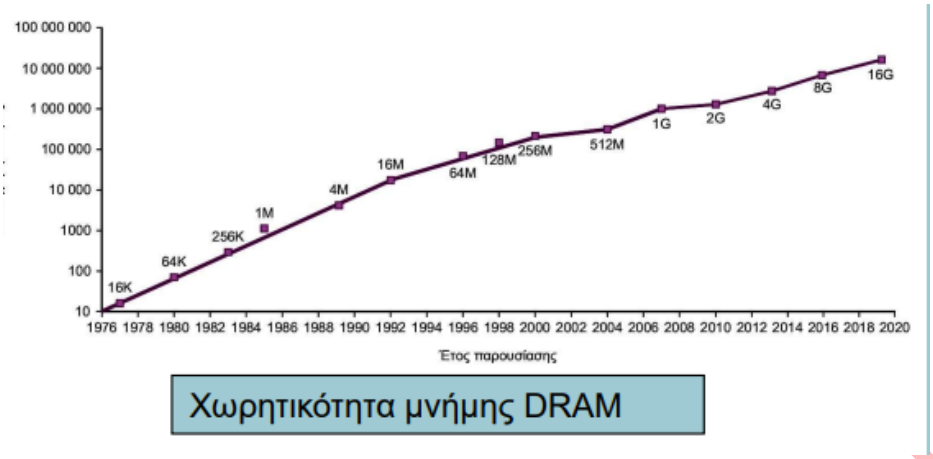
⚙️ Ποια είναι η ιστορική εξέλιξη των τεχνολογιών υπολογιστών;

Τεχνολογία	Έτος	Σχέση απόδοσης-κόστους
Λυχνία κενού	1951	1
Τρανζίστορ	1965	35
Ολοκληρωμένο κύκλωμα	1975	900
Πολύ μεγάλης κλίμακας ολοκλ. κύκλωμα (VLSI)	1995	2.400.000
Υπερυψηλής κλίμακας ολοκλ. κύκλωμα (ULSI)	2013	250.000.000.000

💾 Τι παρατηρείται στη χωρητικότητα της μνήμης DRAM;

Η χωρητικότητα της **DRAM (Dynamic RAM)** αυξάνεται εκθετικά με τα χρόνια, υποστηρίζοντας τις συνεχώς αυξανόμενες απαιτήσεις των εφαρμογών

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Χωρητικότητα μνήμης DRAM

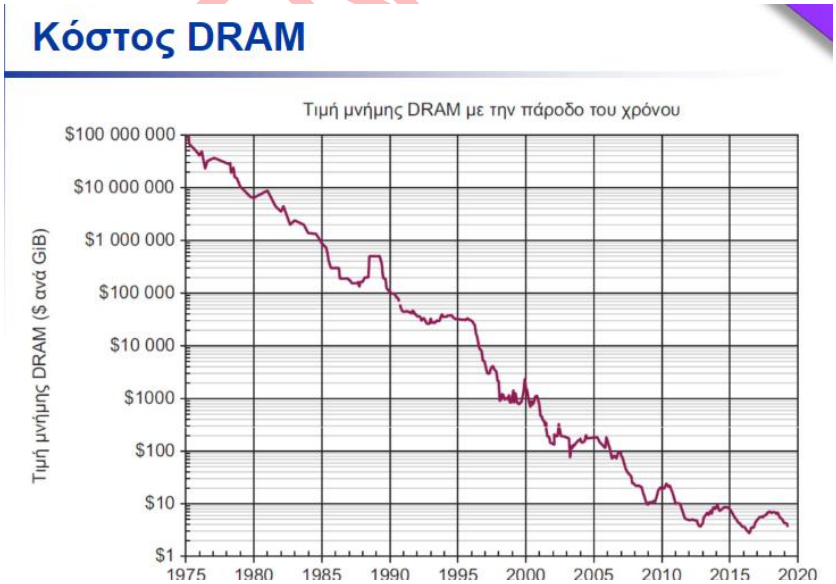
📊 **Πώς έχει εξελιχθεί η χωρητικότητα της μνήμης (π.χ. DRAM) με την πάροδο των ετών;**

🕰️ **Εξέλιξη επί 20 χρόνια:**

- Η χωρητικότητα **τετραπλασιαζόταν (4x) κάθε 3 χρόνια**
- Αυτό αντιστοιχεί σε **αύξηση περίπου 60% κάθε χρόνο**
- Συνολικά μέσα σε αυτή την περίοδο, η αύξηση ήταν **πάνω από 16.000 φορές (16.000x)**

📉 **Τα τελευταία χρόνια:**

- Ο ρυθμός αύξησης έχει επιβραδυνθεί
- Παρατηρείται **διπλασιασμός (2x) κάθε 2–3 χρόνια**



ΝΗ ΖΑΦΕΙΡΗ

💸 **Πώς εξελίχθηκε το κόστος της μνήμης DRAM με την πάροδο του χρόνου;**

Το διάγραμμα δείχνει τη **ραγδαία μείωση** της τιμής της μνήμης DRAM από το 1975 έως το 2020.

📉 **Πόσο σημαντική είναι η μείωση του κόστους σε αριθμούς;**

- Το 1975: το κόστος ήταν περίπου **$100.000 ανά GiB**
- Το 2020: το κόστος είχε πέσει σε επίπεδα **κάτω από $10 ανά GiB**
- Δηλαδή, σε λιγότερο από 50 χρόνια, η τιμή μειώθηκε **κατά περισσότερο από 10 εκατομμύρια φορές**!

⏳ **Ποια είναι η τάση;**

- Η μείωση δεν ήταν γραμμική — υπήρξαν **περίοδοι σταθερότητας** ή **προσωρινής αύξησης**
- Παρ' όλα αυτά, η **γενική τάση** δείχνει **σταθερή πτώση** στο κόστος λόγω της προόδου της τεχνολογίας

🔧 **Γιατί μειώνεται τόσο πολύ η τιμή της DRAM;**

- Βελτιώσεις στη **μικροηλεκτρονική τεχνολογία**
- Μαζική παραγωγή και **οικονομίες κλίμακας**
- Συνεχής **σμίκρυνση τρανζίστορ** και αύξηση χωρητικότητας ανά μονάδα επιφάνειας

🧪 **Τι είναι το πυρίτιο και γιατί είναι σημαντικό στην τεχνολογία;**

Το **πυρίτιο (silicon)** είναι ένα χημικό στοιχείο που ανήκει στην κατηγορία των **ημιαγωγών** — υλικά που μπορούν να συμπεριφέρονται είτε ως αγωγοί είτε ως μονωτές, ανάλογα με τις συνθήκες.

🌍 **Ποια είναι η αφθονία του πυριτίου στη Γη;**

Σύμφωνα με τις χημικές αναλύσεις της επιφάνειας της Γης:

1. **Οξυγόνο** – 49,5%
2. **Πυρίτιο** – 28%
3. **Αλουμίνιο** – 8%

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Το πυρίτιο είναι το **δεύτερο πιο άφθονο στοιχείο** στον φλοιό της Γης!

⚙️ **Πώς αλλάζουν οι ιδιότητες των υλικών στην τεχνολογία ημιαγωγών;**

Μέσω ειδικών τεχνικών προσθήκης υλικών (doping), το πυρίτιο μπορεί να λειτουργήσει ως:

- **Αγωγός του ηλεκτρισμού**
- **Μονωτής του ηλεκτρισμού**
- **Μεταγωγέας (διακόπτης)** — βασική λειτουργία στα τρανζίστορ

📈 **Τι είναι ο Νόμος του Moore και ποια η σημασία του;**

Ο **Νόμος του Moore**, διατυπωμένος το 1965 από τον **Gordon Moore** (συνιδρυτής της Intel), προβλέπει ότι:

"Η χωρητικότητα των μικροεπεξεργαστών σε τρανζίστορ θα **διπλασιάζεται κάθε 18–24 μήνες.**"

⏳ **Πόσο ισχυρός αποδείχθηκε ο νόμος του Moore;**

- Παρέμεινε **αξιοσημείωτα ακριβής για περίπου 40 χρόνια**
- Η συνεχής πρόοδος απαιτούσε **τεράστιες καινοτομίες** σε τεχνικές κατασκευής, όπως:
    - λιθογραφία
    - σμίκρυνση διαστάσεων τρανζίστορ
    - νέες αρχιτεκτονικές

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🏭 Ποια είναι τα βασικά στάδια παραγωγής ολοκληρωμένων κυκλωμάτων (ICs);

Η διαδικασία περιλαμβάνει πολλά διαδοχικά βήματα υψηλής ακρίβειας:

🔹 **1. Πρώτη ύλη: Ράβδος πυριτίου (silicon ingot)**

Η διαδικασία ξεκινά με **μια καθαρή ράβδο πυριτίου**, που αποτελεί τη βάση των ημιαγωγών.

🔹 **2. Τεμαχισμός: Συσκευή τεμαχισμού**

Η ράβδος κόβεται σε **λεπτές πλάκες πυριτίου** (wafers) γνωστές ως **κενά πλακίδια**.

🔹 **3. Επεξεργασία πλακιδίων**

Τα πλακίδια περνούν από **20 έως 40 στάδια επεξεργασίας** για να δημιουργηθούν τα ολοκληρωμένα κυκλώματα.

🔹 **4. Έλεγχος και κοπή (testing & dicing)**

- Δοκιμάζονται (test) για λειτουργικότητα
- Κόβονται σε **μεμονωμένα chips (τοπς)** με ειδικά κοπτικά εργαλεία
- Τα καλά chips διαχωρίζονται για περαιτέρω χρήση

🔹 **5. Συσκευασία (packaging)**

Τα λειτουργικά chips τοποθετούνται σε **κύβους συσκευασίας** για προστασία και σύνδεση με εξωτερικά κυκλώματα.

🔹 **6. Τελικές δοκιμές και αποστολή**

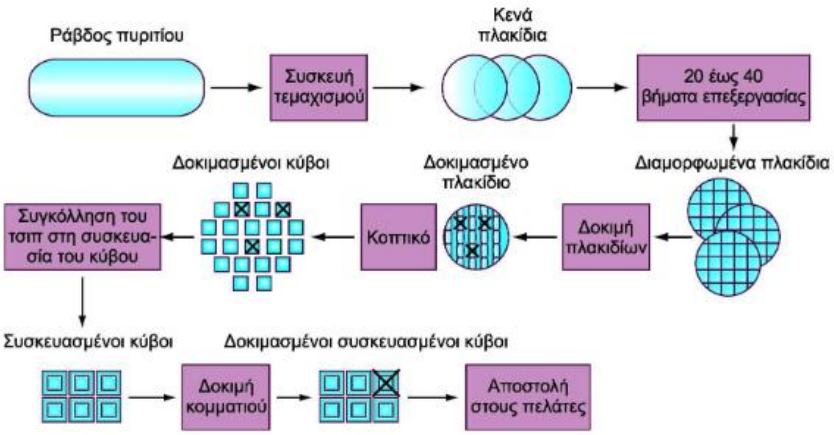
- Οι συσκευασμένοι κύβοι δοκιμάζονται ξανά
- Όσα περνούν τον ποιοτικό έλεγχο αποστέλλονται στους πελάτες

📈 Τι είναι η «εσοδεία» (yield);

Εσοδεία είναι το **ποσοστό των καλών (λειτουργικών) chips** σε σχέση με το **σύνολο των chips** που προέρχονται από ένα πλακίδιο.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Παράδειγμα: Αν από 100 τοπς λειτουργούν τα 80, η εσοδεία είναι **80%**.



💻 Τι περιέχει ένα πλακίδιο επεξεργαστών Intel® Core 10ης γενιάς;

Το πλακίδιο αποτελεί τη βάση πάνω στην οποία κατασκευάζονται δεκάδες ολοκληρωμένα κυκλώματα (chips/τοπς).

📏 Ποια είναι τα βασικά χαρακτηριστικά ενός τέτοιου πλακιδίου;

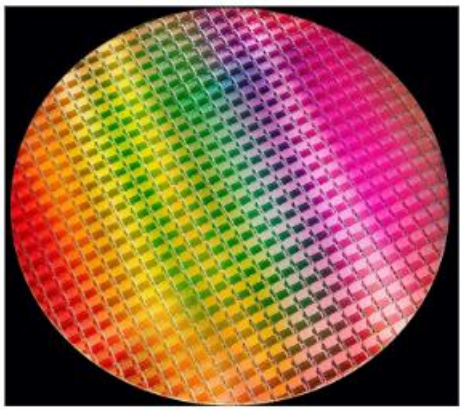
- **Διάμετρος πλακιδίου**: 300 mm
- **Συνολικός αριθμός τοπς (chips)**: 506
- **Τεχνολογία κατασκευής**: 10 nm (νανομέτρα), που δηλώνει την ακρίβεια στη χάραξη τρανζίστορ

🔬 Ποιες είναι οι διαστάσεις κάθε κύβου (chip);

Κάθε τοπ (chip) έχει διαστάσεις:
📐 **11.4 mm × 10.7 mm**

Αυτό σημαίνει πως **το κάθε chip είναι πολύ μικρό σε μέγεθος**, αλλά περιέχει **εκατομμύρια έως δισεκατομμύρια τρανζίστορ**!

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



💰 Πώς υπολογίζεται το κόστος ενός ολοκληρωμένου κυκλώματος (chip);

Ο τύπος είναι:

$$\text{Κόστος ανά κύβο} = \frac{\text{Κόστος ανά πλακίδιο}}{\text{Κύβοι ανά πλακίδιο} \times \text{Εσοδεία}}$$

📐 Πώς υπολογίζονται οι "κύβοι ανά πλακίδιο";

$$\text{Κύβοι ανά πλακίδιο} = \frac{\text{Επιφάνεια πλακιδίου}}{\text{Επιφάνεια κύβου}}$$

⚠️ Τι είναι η εσοδεία (yield) και πώς επηρεάζει το κόστος;

$$\text{Εσοδεία} = \frac{1}{(1 + (\text{Ατέλειες ανά περιοχή}) \times \text{Επιφάνεια κύβου})^n}$$

📉 Είναι γραμμική η σχέση κόστους – επιφάνειας – ατελειών;

Όχι! Η σχέση είναι **μη γραμμική**, επειδή:

- Το **κόστος και η επιφάνεια των πλακιδίων** είναι σταθερά
- Το **ποσοστό ατελειών** εξαρτάται από τη **διαδικασία κατασκευής**
- Η **επιφάνεια κάθε κύβου (chip)** εξαρτάται από τον **σχεδιασμό** και την **αρχιτεκτονική** του κυκλώματος

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✈️ Τι είναι η απόδοση στον τομέα των αεροπλάνων;

Απόδοση σε αυτή την περίπτωση σημαίνει **η ικανότητα ενός αεροπλάνου να μεταφέρει επιβάτες σε μεγάλες αποστάσεις**.
Μετριέται ως:
👉 **Διεκπεραιωτική ικανότητα επιβατών = Χωρητικότητα × Αυτονομία**

📊 Ποια είναι τα χαρακτηριστικά των αεροπλάνων που συγκρίνονται;

Αεροπλάνο	Χωρητικότητα	Αυτονομία (μίλια)	Ταχύτητα (μίλια/ώρα)	Διεκπεραιωτική ικανότητα
Boeing 737	240	3.000	564	135.360
BAC/Sud Concorde	132	4.000	1.350	178.200
Boeing 777-200LR	301	9.395	554	166.761
Airbus A380-800	**853**	**8.477**	**587**	**500.711**

🥇 Ποιο έχει την καλύτερη απόδοση;

➡️ **Το Airbus A380-800** έχει τη **μεγαλύτερη διεκπεραιωτική ικανότητα επιβατών**:
📈 **500.711 επιβάτες × μίλια/ώρα**,
που σημαίνει ότι είναι το **πιο αποδοτικό** από τα παραπάνω αεροσκάφη για μεταφορά μεγάλου αριθμού ανθρώπων σε μεγάλη απόσταση.

⏱️ Τι είναι ο χρόνος απόκρισης (response time);

Είναι ο **χρόνος που απαιτείται για να ολοκληρωθεί μία εργασία** από τη στιγμή που ξεκινά.
➤ Εκφράζει πόσο γρήγορα ανταποκρίνεται το σύστημα σε μία αίτηση ή εντολή.

📦 Τι είναι η διεκπεραιωτική ικανότητα (throughput);

Αναφέρεται στη **συνολική ποσότητα εργασίας που μπορεί να ολοκληρώσει το σύστημα σε συγκεκριμένο χρονικό διάστημα**, π.χ.
➡️ **εργασίες/ώρα**, **συναλλαγές/δευτερόλεπτο**, κ.λπ.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔁 **Πώς επηρεάζονται ο χρόνος απόκρισης και η διεκπεραιωτική ικανότητα;**

- **Με ταχύτερο επεξεργαστή**:
  ➤ **Μειώνεται ο χρόνος απόκρισης**
  ➤ **Αυξάνεται η διεκπεραιωτική ικανότητα**
- **Με προσθήκη περισσότερων επεξεργαστών**:
  ➤ **Αυξάνεται σημαντικά η διεκπεραιωτική ικανότητα**
  ➤ Μπορεί ή όχι να μειωθεί ο χρόνος απόκρισης, ανάλογα με την εργασία

🔍 *Σε αυτή τη φάση επικεντρωνόμαστε στον χρόνο απόκρισης.*

⚖️ **Ποια είναι η σχέση ανάμεσα σε απόδοση και χρόνο εκτέλεσης;**

$$\text{Απόδοση (performance)} = \frac{1}{\text{Χρόνος εκτέλεσης}}$$

🧮 **Παράδειγμα σύγκρισης συστημάτων με βάση την απόδοση:**

Ένα πρόγραμμα εκτελείται:

- Σε σύστημα **A** σε 10 δευτερόλεπτα
- Σε σύστημα **B** σε 15 δευτερόλεπτα

$$\frac{\text{Χρόνος B}}{\text{Χρόνος A}} = \frac{15}{10} = 1{,}5 \Rightarrow \text{Ο A είναι 1,5 φορές ταχύτερος από τον B}$$

🕓 **Πώς μετράμε τον χρόνο εκτέλεσης μιας εργασίας;**

📍 **Παρελθών χρόνος (Elapsed time):**

Ο συνολικός χρόνος που μεσολαβεί από την έναρξη μέχρι την ολοκλήρωση, συμπεριλαμβάνοντας:

- Επεξεργασία (CPU)
- Εισόδους/εξόδους (I/O)
- Λειτουργικό σύστημα
- Καθυστέρηση από άλλες διεργασίες

⚙️ **Χρόνος CPU (CPU time):**

Ο χρόνος που ξοδεύεται **μόνο στην επεξεργασία** της συγκεκριμένης εργασίας από την CPU.
Αποτελείται από:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Χρόνο CPU χρήστη (user CPU time)**
- **Χρόνο CPU συστήματος (system CPU time)**

⏱️ Τι είναι ο χρονισμός της CPU και γιατί είναι σημαντικός;

Η λειτουργία του ψηφιακού υλικού — όπως οι επεξεργαστές — **διέπεται από ένα ρολόι σταθερού ρυθμού**, που συγχρονίζει τις πράξεις επεξεργασίας και μεταφοράς δεδομένων.

🔁 Τι είναι η περίοδος ρολογιού (clock period);

Η **περίοδος ρολογιού** είναι η **διάρκεια ενός πλήρους κύκλου του ρολογιού**.
➤ Παράδειγμα:

- Αν η περίοδος είναι **250 ps**, τότε:

$$250 \text{ ps} = 0.25 \text{ ns} = 250 \times 10^{-12} \text{ sec}$$

📉 Τι σημαίνει μικρότερη περίοδος ρολογιού;

Όσο μικρότερη είναι η περίοδος, τόσο **γρηγορότερα** γίνονται οι πράξεις, άρα **υψηλότερη απόδοση** (αν και όχι απαραίτητα καλύτερη ενεργειακή συμπεριφορά ή σταθερότητα).

📈 Τι είναι η συχνότητα ρολογιού (clock frequency);

Η συχνότητα είναι το **πλήθος κύκλων ανά δευτερόλεπτο**.

Παράδειγμα:

- 4.0 GHz = 4000 MHz =

$$4.0 \times 10^9 \text{ Hz} = 4 \delta\iota\sigma\varepsilon\kappa\alpha\tau o\mu\mu\acute{\upsilon}\rho\iota\alpha\kappa\acute{\upsilon}\kappa\lambda o\iota\alpha\nu\acute{\alpha}\delta\varepsilon\upsilon\tau\varepsilon\rho\acute{o}\lambda\varepsilon\pi\tau o$$

🔁 Πώς σχετίζεται η συχνότητα με την ταχύτητα της CPU;

- **Υψηλότερη συχνότητα ⇒ περισσότερες πράξεις ανά δευτερόλεπτο**
- Όμως, η απόδοση δεν εξαρτάται μόνο από αυτό (παίζουν ρόλο και άλλα στοιχεία όπως η αρχιτεκτονική, η μνήμη cache κ.ά.)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🕓 Τι είναι ο χρόνος CPU (CPU time);

Ο **χρόνος εκτέλεσης CPU για ένα πρόγραμμα** υπολογίζεται με τον τύπο:

$$\text{Χρόνος CPU} = \text{Κύκλοι ρολογιού CPU} \times \text{Χρόνος κύκλου ρολογιού}$$

Ή εναλλακτικά:

$$\text{Χρόνος CPU} = \frac{\text{Κύκλοι ρολογιού CPU}}{\text{Ρυθμός ρολογιού (συχνότητα)}}$$

⚡ Πώς μπορεί να βελτιωθεί η απόδοση της CPU;

Η απόδοση βελτιώνεται:

- ✅ Με τη **μείωση των κύκλων του ρολογιού** (λιγότεροι κύκλοι για την ίδια εργασία)
- ✅ Με την **αύξηση της συχνότητας του ρολογιού** (περισσότεροι κύκλοι ανά δευτερόλεπτο)
- ⚠️ Ο σχεδιαστής πρέπει να **εξισορροπήσει** τους δύο αυτούς παράγοντες για μέγιστο αποτέλεσμα.

🖥️ Δεδομένα:

- **Υπολογιστής Α**:
  - Ρολόι: 2 GHz
  - Χρόνος CPU: 10 δευτερόλεπτα

🎯 Στόχος για Υπολογιστή Β:

- Θέλουμε ο **χρόνος CPU** να γίνει **6 δευτερόλεπτα**.
- Όμως, γνωρίζουμε ότι οι **κύκλοι ρολογιού** στον υπολογιστή Β θα είναι **1.2 φορές περισσότεροι** από του Α (πιθανότατα λόγω πιο σύνθετων εντολών, pipeline, κ.λπ.).

❓ Ερώτηση:

Πόσο **γρήγορο** πρέπει να είναι το **ρολόι του υπολογιστή Β** ώστε να πετύχει αυτόν τον στόχο;

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔢 **Τύπος για τον χρόνο CPU:**

$$\text{Χρόνος}_{CPU} = \frac{\text{Κύκλοι ρολογιού}}{\text{Ρυθμός ρολογιού}}$$

$$\text{Χρόνος}_{CPU_B} = \frac{1.2 \times \text{Κύκλοι}_{CPU_A}}{\text{Ρυθμός}_B}$$

📌 **Αντικαθιστώντας τιμές:**

- Κύκλοι CPU_A = $10 \text{ sec} \times 2 \times 10^9 \text{ κύκλοι/sec} = 20 \times 10^9 \text{ κύκλοι}$
- Θέλουμε: $\text{Χρόνος}_{CPU_B} = 6 \text{ sec}$

Οπότε:

$$6 = \frac{1.2 \times 20 \times 10^9}{\text{Ρυθμός}_B} \Rightarrow \text{Ρυθμός}_B = \frac{1.2 \times 20 \times 10^9}{6} = 4 \times 10^9 = \boxed{4 \text{ GHz}}$$

✅ **Συμπέρασμα:**

Για να επιτευχθεί χρόνος CPU 6 δευτερόλεπτα, παρόλο που ο υπολογιστής B χρειάζεται **20% περισσότερους κύκλους ρολογιού**, το **ρολόι του πρέπει να είναι 4 GHz** (διπλάσιο από το ρολόι του A, που είναι 2 GHz).

Αυτό δείχνει τη **σχέση trade-off** μεταξύ:

- κύκλων ρολογιού
- συχνότητας ρολογιού
- και χρόνου CPU.

ΕΛΕΝΗ ΖΑΚ

🧠 **Κύριες Έννοιες:**

🔹 **Τύπος κύκλων ρολογιού:**

$$\text{Κύκλοι ρολογιού CPU} = \text{Εντολές για ένα πρόγραμμα} \times \text{Μέσος αριθμός κύκλων ανά εντολή}$$

Αυτός ο τύπος δείχνει πόσοι κύκλοι ρολογιού χρειάζονται συνολικά για να εκτελεστεί ένα πρόγραμμα.

🔹 **Τύπος για χρόνο CPU:**

$$\text{Χρόνος CPU} = \text{Πλήθος εντολών} \times \text{CPI} \times \text{Χρόνος κύκλου ρολογιού}$$

$$\text{Χρόνος CPU} = \frac{\text{Πλήθος εντολών} \times \text{CPI}}{\text{Ρυθμός ρολογιού}}$$

🔍 **Τι επηρεάζει αυτούς τους παράγοντες:**

📌 **Πλήθος Εντολών:**

- Καθορίζεται **από το πρόγραμμα**.
- Επηρεάζεται από τη **γλώσσα μηχανής (ISA)** και τον **μεταγλωττιστή**.
- Όσο πιο αποδοτικός ο κώδικας, τόσο λιγότερες εντολές.

📌 **CPI (Cycles Per Instruction):**

- Καθορίζεται **από το υλικό της CPU**.
- Επηρεάζεται από:
  - Το αν υπάρχει pipeline, superscalar, out-of-order execution, κ.λπ.
  - Το **είδος των εντολών**: π.χ., μια πρόσθεση ίσως χρειάζεται 1 κύκλο, ενώ ένα memory load 4 κύκλους.
- Οπότε παίρνουμε **μέση τιμή CPI** (π.χ., 1.5 κύκλοι/εντολή).

🎯 **Τι σημαίνει αυτό πρακτικά;**

Αν θέλεις να **βελτιώσεις τον χρόνο CPU**, μπορείς να επέμβεις σε:

1. **Αριθμό εντολών**: καλύτερη μεταγλώττιση, αλλαγή αλγορίθμου.
2. **CPI**: καλύτερη αρχιτεκτονική CPU.
3. **Συχνότητα ρολογιού**: υψηλότερη συχνότητα (π.χ. από 2 GHz → 4 GHz).

Όμως:

- Αυτές οι βελτιώσεις συχνά έχουν **αντίκτυπο μεταξύ τους** (π.χ., μεγαλύτερη συχνότητα → περισσότερα σφάλματα → πιο αργή αρχιτεκτονική → μεγαλύτερο CPI).

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

📌 **Από τι εξαρτάται το πλήθος εντολών και το CPI;**

- Το **πλήθος εντολών** εξαρτάται από:
  - το πρόγραμμα,
  - την ISA (Instruction Set Architecture),
  - και τον μεταγλωττιστή.
- Το **CPI** εξαρτάται από:
  - το **υλικό της CPU**,
  - τον **τύπο εντολών** (κάθε εντολή μπορεί να έχει διαφορετικό CPI),
  - τον **μέσο όρο CPI** ανά πρόγραμμα.

📈 **Παράδειγμα σύγκρισης δύο CPU ως προς το CPI**

- **Υπολογιστής A**:
  - Χρόνος κύκλου: 250 ps
  - CPI: 2.0
- **Υπολογιστής B**:
  - Χρόνος κύκλου: 500 ps
  - CPI: 1.2

Ποιος είναι ταχύτερος;

Υπολογισμός:

$$\text{Χρόνος CPU}_A = I \times 2.0 \times 250\ ps = 500 \times I\ ps$$

$$\text{Χρόνος CPU}_B = I \times 1.2 \times 500\ ps = 600 \times I\ ps$$

➡️ Ο **A είναι ταχύτερος**, γιατί έχει μικρότερο συνολικό χρόνο CPU.

Πόσο ταχύτερος;

$$\text{Απόδοση A}/\text{Απόδοση B} = \frac{600 \times I}{500 \times I} = 1.2$$

➡️ Ο **A είναι 1.2 φορές ταχύτερος από τον B**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

📌 1. Γενικός Υπολογισμός Κύκλων Ρολογιού:

$$\text{Κύκλοι ρολογιού CPU} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

Όπου:

- CPI_i: οι κύκλοι ανά εντολή για την κατηγορία εντολών i
- C_i: ο αριθμός των εντολών της κατηγορίας i

Δηλαδή: για κάθε τύπο εντολής, πολλαπλασιάζουμε πόσους κύκλους χρειάζεται επί πόσες φορές εμφανίζεται, και αθροίζουμε όλα.

📌 2. Σταθμισμένη Μέση Τιμή CPI:

$$\text{CPI} = \frac{\text{Κύκλοι ρολογιού}}{\text{Σύνολο εντολών}} = \sum_{i=0}^{n} \left(CPI_i \times \frac{\text{Πλήθος εντολών}_i}{\text{Πλήθος εντολών}} \right)$$

Αυτό είναι **σταθμισμένος μέσος όρος**, δηλαδή:

- Κάθε CPI_i σταθμίζεται με βάση τη **σχετική συχνότητα** της αντίστοιχης κατηγορίας εντολών (δηλαδή πόσο συχνά χρησιμοποιείται).

🧠 Πρακτικό Παράδειγμα:

Έστω:

- Εντολές τύπου A: 1 κύκλος, 50% των εντολών → $CPI_1 = 1$, $f_1 = 0.5$
- Εντολές τύπου B: 3 κύκλοι, 30% → $CPI_2 = 3$, $f_2 = 0.3$
- Εντολές τύπου C: 5 κύκλοι, 20% → $CPI_3 = 5$, $f_3 = 0.2$

Τότε:

$$\text{CPI}_{avg} = (1 \times 0.5) + (3 \times 0.3) + (5 \times 0.2) = 0.5 + 0.9 + 1 = \boxed{2.4}$$

✅ Συμπέρασμα:

- Το **CPI** δεν είναι απλά ένας αριθμός — εξαρτάται από το **μίγμα εντολών** και το **υλικό της CPU**.
- Για να μειώσουμε το μέσο CPI:
  - Μειώνουμε τους κύκλους των πιο συχνών εντολών.
  - Ή ελαττώνουμε τη χρήση των πιο "βαριών" εντολών.

Κατηγορία	CPI	Εντολές Ακολουθίας 1	Εντολές Ακολουθίας 2
Α	1	2	4
Β	2	1	1
Γ	3	2	1

📘 Ανάλυση Ακολουθίας 1

- **Αριθμός εντολών (IC)** = 2 (Α) + 1 (Β) + 2 (Γ) = **5**
- **Κύκλοι ρολογιού**:
  - 2×1+1×2+2×3=2+2+6=10
- **Μέσος CPI**:

$$\text{CPI} = \frac{10}{5} = \boxed{2.0}$$

📕 Ανάλυση Ακολουθίας 2

- **Αριθμός εντολών (IC)** = 4 (Α) + 1 (Β) + 1 (Γ) = **6**
- **Κύκλοι ρολογιού**:
  - 4×1+1×2+1×3=4+2+3=9
- **Μέσος CPI**:

$$\text{CPI} = \frac{9}{6} = \boxed{1.5}$$

🎯 Συμπέρασμα

- Παρόλο που η **Ακολουθία 2** έχει **περισσότερες εντολές**, έχει **χαμηλότερο CPI** και συνολικά **λιγότερους κύκλους ρολογιού (9 αντί για 10)**.
- Άρα, **το συνολικό κόστος εκτέλεσης μπορεί να είναι μικρότερο**, ακόμη και αν έχεις περισσότερες εντολές, αρκεί αυτές να είναι **αποδοτικότερες**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🧠 **Ο Γενικός Τύπος Χρόνου Εκτέλεσης (CPU Time):**

$$\text{Χρόνος} = \frac{\text{Εντολές}}{\text{Πρόγραμμα}} \times \frac{\text{Κύκλοι ρολογιού}}{\text{Εντολή}} \times \frac{\text{Δευτερόλεπτα}}{\text{Κύκλο ρολογιού}}$$

ή πιο απλά:

$$\text{Χρόνος CPU} = \text{IC} \times \text{CPI} \times T_c$$

- **IC (Instruction Count)**: αριθμός εντολών
- **CPI**: κύκλοι ανά εντολή
- **Tc**: χρόνος κάθε κύκλου (δηλ. 1 / συχνότητα ρολογιού)

📌 **Τι επηρεάζει την απόδοση:**

1. **Αλγόριθμος**:
   - Επηρεάζει τον **αριθμό εντολών (IC)**.
   - Ίσως και το **CPI**, αν επιλέγεις πιο αποδοτικές/απλές λειτουργίες.
2. **Γλώσσα προγραμματισμού**:
   - Μπορεί να παράγει **περισσότερο ή λιγότερο αποδοτικό κώδικα**, επηρεάζοντας IC και CPI.
3. **Μεταγλωττιστής**:
   - Ένας "έξυπνος" μεταγλωττιστής **βελτιστοποιεί** τον κώδικα μειώνοντας IC ή/και CPI.
4. **Αρχιτεκτονική ISA (Instruction Set Architecture)**:
   - Επηρεάζει τα πάντα:
     - **Πόσες εντολές** χρειάζονται για μια πράξη
     - **Πόσο κοστίζει κάθε εντολή (CPI)**
     - **Τι συχνότητες μπορεί να υποστηρίξει το hardware (Tc)**

🎯 **Συμπέρασμα:**

Η **απόδοση του συστήματος** εξαρτάται από τον **συνδυασμό λογισμικού και υλικού**. Για καλύτερη απόδοση, χρειάζεται **συνεργασία όλων των επιπέδων**: από τον αλγόριθμο και τη γλώσσα, μέχρι τη μικροαρχιτεκτονική και τη συχνότητα CPU.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

⚙️ **CPI και IPC – Τι σημαίνουν;**

✅ **CPI (Cycles Per Instruction)**

Δείχνει **πόσοι κύκλοι ρολογιού χρειάζονται κατά μέσο όρο** για την εκτέλεση **μίας εντολής**.

- Όσο **μικρότερο** το CPI, τόσο **καλύτερη η απόδοση**.

✅ **IPC (Instructions Per Cycle)**

Δείχνει **πόσες εντολές εκτελούνται κατά μέσο όρο σε κάθε κύκλο ρολογιού**.

- Όσο **μεγαλύτερο** το IPC, τόσο **καλύτερη η απόδοση**.
- Είναι **το αντίστροφο του CPI**, όταν οι εντολές εκτελούνται με πλήρη παράλληλη αξιοποίηση:

$$IPC = \frac{1}{CPI} \quad \text{και} \quad CPI = \frac{1}{IPC}$$

📌 **Παράδειγμα:**

Ένας απλός επεξεργαστής χωρίς παράλληλη εκτέλεση:

- Εκτελεί 1 εντολή κάθε 1 κύκλο →

$$IPC = 1, \quad CPI = 1$$

Ένας σύγχρονος **superscalar επεξεργαστής**:

- Εκτελεί 4 εντολές ταυτόχρονα σε κάθε κύκλο →

$$IPC = 4, \quad CPI = \frac{1}{4} = 0.25$$

Αυτό σημαίνει ότι **για κάθε εντολή απαιτείται μόνο 0.25 κύκλος** (δηλαδή, 4 εντολές/κύκλο).

🔍 **Πότε έχουμε CPI < 1;**

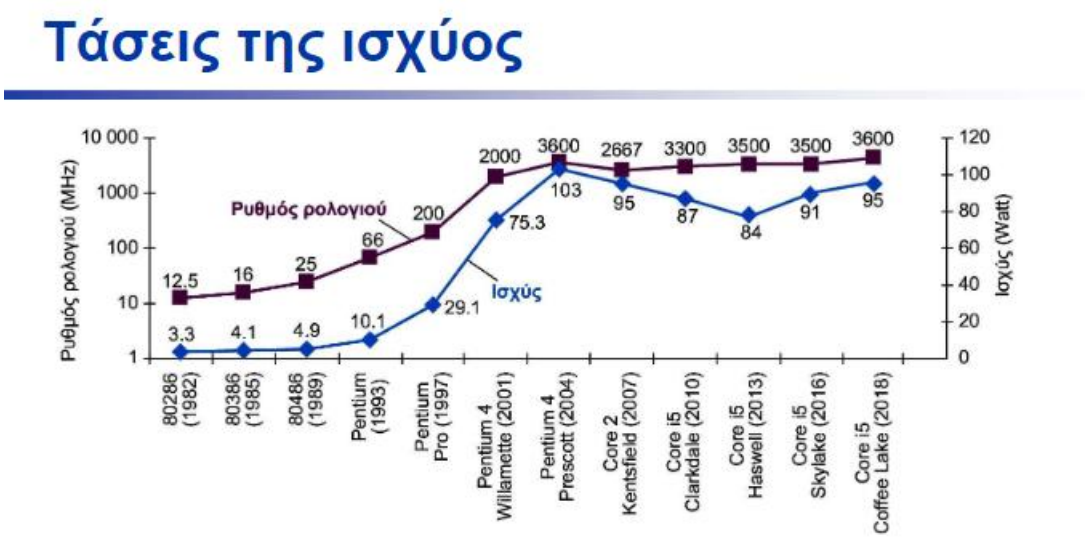
- Όταν η CPU μπορεί να **εκτελεί περισσότερες από 1 εντολή ανά κύκλο**.
- Αυτό γίνεται σε αρχιτεκτονικές όπως:
  - **Superscalar**
  - **Out-of-order execution**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Pipelining**
- **SIMD (Single Instruction Multiple Data)**

✅ **Συμπέρασμα:**

- **CPI → "Πόσο ακριβή" είναι μια εντολή** σε χρόνο.
- **IPC → "Πόση δουλειά κάνουμε" σε κάθε κύκλο.**
- Στη σύγχρονη αρχιτεκτονική, το IPC είναι βασικός δείκτης **εκμετάλλευσης του hardware**.



🔷 **1. Διάγραμμα: Ρυθμός ρολογιού & Ισχύς**

- **Γραφική παράσταση (πάνω):**
  - Ο **ρυθμός ρολογιού** (σε MHz/GHz) αυξάνεται συνεχώς μέχρι περίπου το 2005–2006.
  - Μετά πέφτει ή σταθεροποιείται γύρω στα 3–4 GHz.
  - Η **ισχύς** (σε Watt) φτάνει κορυφή περίπου με το Prescott (~100W) και παραμένει περίπου σταθερή (~85–95W) μετά.

⚙️ **2. Θεμελιώδης Τύπος Ισχύος σε CMOS:**

$$\text{Ισχύς} = C \times V^2 \times f$$

Όπου:

- **C**: χωρητικότητα φορτίου (load capacitance)
- **V**: τάση τροφοδοσίας
- **f**: συχνότητα μεταγωγής (δηλαδή ρυθμός ρολογιού)

📉 **Τάσεις με την πάροδο του χρόνου:**

- **Η χωρητικότητα (C)** αυξήθηκε ~**30 φορές**
- **Η τάση** μειώθηκε από **5V σε 1V** (~**x25** μείωση στην ισχύ μέσω V^2)
- **Η συχνότητα** αυξήθηκε ~**1000 φορές**

➡️ Χωρίς μείωση τάσης, η ισχύς θα αυξανόταν **εκθετικά**.

🔍 **Τι σημαίνει αυτό:**

- Η **κατανάλωση ενέργειας** έγινε **προβληματική** όταν συνέχισε να αυξάνεται η συχνότητα (Prescott 2004 → thermal wall).
- Από εκεί και μετά:
  - Σταθερή ή ήπια αύξηση ρολογιού
  - Εστίαση σε **πολυπύρηνες αρχιτεκτονικές** και **βελτιώσεις IPC**
  - Χρήση **ενεργειακά αποδοτικών πυρήνων** (π.χ. big.LITTLE)

✅ **Συμπέρασμα:**

Η κατανάλωση ισχύος σε CMOS κυκλώματα **εξαρτάται κυρίως από την τάση (V^2)** και τη συχνότητα.
Γι' αυτό η μείωση της τάσης (από 5V → 1V) ήταν **κρίσιμη τεχνολογική εξέλιξη** για να μπορέσουμε να αυξήσουμε τη συχνότητα **χωρίς να υπερθερμάνουμε το chip**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔻 Παράδειγμα Μείωσης Ισχύος

Έστω ότι σχεδιάζεται μια νέα CPU με τις εξής βελτιώσεις:

- 85% του φορτίου χωρητικότητας (C) του παλαιότερου μοντέλου
- 15% μείωση στην τάση (V) → δηλ. $V_{\text{νέου}} = 0.85 \cdot V_{\text{παλιού}}$
- 15% μείωση στη συχνότητα (f) → $f_{\text{νέου}} = 0.85 \cdot f_{\text{παλιού}}$

Χρησιμοποιούμε τον τύπο για την ισχύ CMOS:

$$\text{Ισχύς} = C \cdot V^2 \cdot f$$

➤ **Υπολογισμός:**

$$\frac{\text{Ισχύς}_{\text{νέου}}}{\text{Ισχύς}_{\text{παλιού}}} = (0.85) \cdot (0.85)^2 \cdot (0.85) = 0.85^4 = \boxed{0.52}$$

👉 Δηλαδή, 48% μείωση στην ισχύ.

⚠️ Το «Τείχος της Ισχύος» (Power Wall)

Μετά από κάποιο σημείο, **δεν μπορούμε να μειώσουμε άλλο**:

- **Τάση (V):** Η μείωση επηρεάζει τη σταθερότητα και την ταχύτητα των τρανζίστορ.
- **Θερμότητα:** Δεν μπορούμε να απομακρύνουμε παραπάνω θερμότητα από μικρότερους χώρους (θερμικός περιορισμός).

Άρα, φτάνουμε σε τεχνολογικό όριο.

❓ Πώς μπορούμε να βελτιώσουμε την απόδοση χωρίς αύξηση ισχύος;

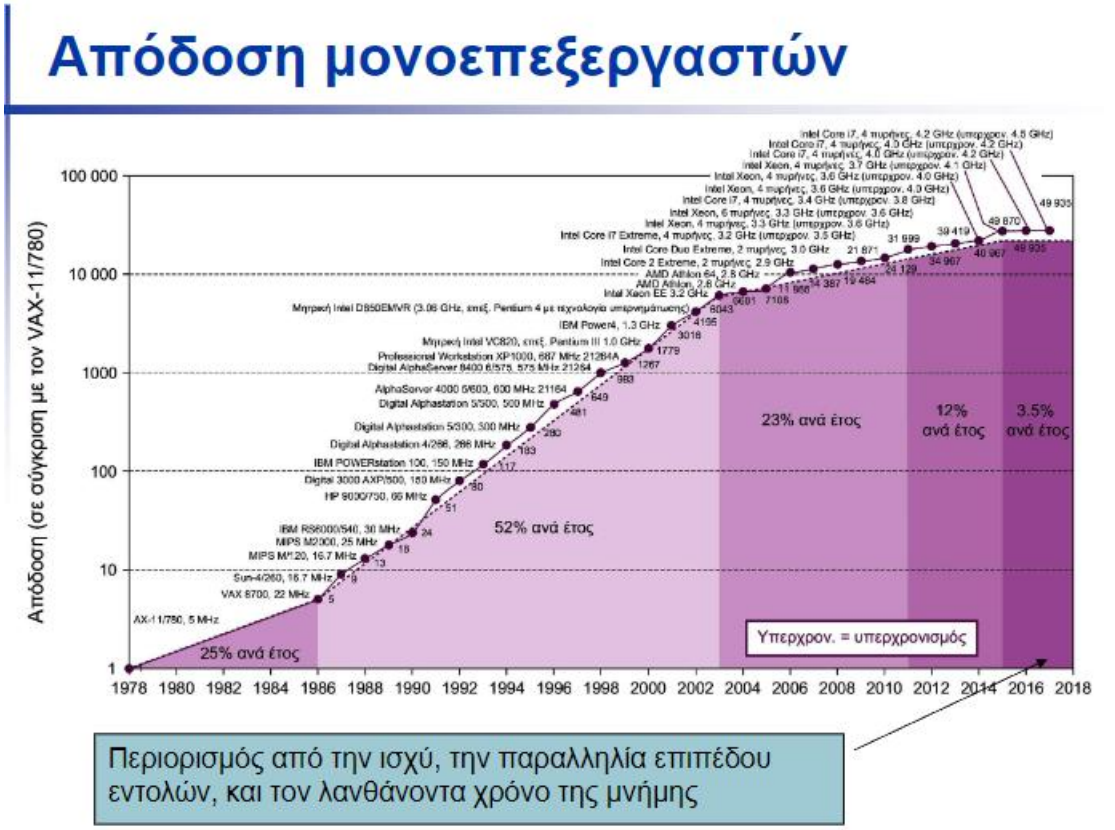
1. **Πολυπύρηνοι επεξεργαστές** (multicore)
2. **Βελτίωση IPC** (περισσότερες εντολές ανά κύκλο)
3. **Εξειδικευμένα κυκλώματα** (π.χ. GPU, NPU)
4. **Ενεργειακά αποδοτικός σχεδιασμός** (π.χ. δυναμική ρύθμιση τάσης και συχνότητας – DVFS)
5. **Παράλληλος αλγόριθμος / λογισμικό** για αποδοτικότερη χρήση πόρων

✅ Συμπέρασμα:

- Η ισχύς είναι **πολυπαραγοντικό πρόβλημα**.
- Υπάρχει **φυσικό όριο** στη μείωση της ισχύος μόνο μέσω υλικού.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Η **συνεργασία hardware + software** είναι απαραίτητη για **συνέχιση της αύξησης απόδοσης**.



📈 **Άξονες του γραφήματος:**

- **Κατακόρυφος (Y):** Απόδοση (σε λογαριθμική κλίμακα), συγκριτικά με τον VAX-11/780.
- **Οριζόντιος (X):** Χρονολογία (1978–2018).
- Οι αριθμοί μέσα στο γράφημα είναι **μοντέλα επεξεργαστών** από Intel, AMD, IBM, κ.ά.

📊 **Ρυθμός αύξησης απόδοσης ανά εποχή:**

- **1980–1986:** ~25% αύξηση ανά έτος
- **1986–2002:** εντυπωσιακή αύξηση ~**52% ανά έτος**
    - Χάρη σε αυξανόμενο ρυθμό ρολογιού, pipeline, superscalar αρχιτεκτονικές
- **2002–2006:** πτώση σε ~23%/έτος
- **2006–2011:** ~12%/έτος

- **2011–2018:** μόλις ~**3.5% ανά έτος**

🧱 **Πού οφείλεται η επιβράδυνση;**

Στο κάτω μέρος της εικόνας αναφέρεται:

"Περιορισμός από την ισχύ, την παραλληλία επιπέδου εντολών (ILP), και τον λανθάνοντα χρόνο μνήμης"

Ας τα αναλύσουμε:

1. ⚡ **Ισχύς (Power Wall):**
   - Δεν μπορούμε να αυξήσουμε περαιτέρω συχνότητα λόγω **υπερθέρμανσης και κατανάλωσης**.
2. 🔁 **ILP (Instruction-Level Parallelism):**
   - Έχουμε εξαντλήσει το **πόσες εντολές μπορούμε να εκτελέσουμε ταυτόχρονα** μέσα σε έναν πυρήνα (π.χ. με pipelining, superscalar κ.λπ.).
3. 🧠 **Μνήμη (Memory Wall):**
   - Η **ταχύτητα της μνήμης δεν αυξάνεται** με τον ίδιο ρυθμό με την CPU → προκαλεί καθυστερήσεις λόγω cache misses.

❗ **Συμπέρασμα:**

Από περίπου το 2005 και μετά, η **βελτίωση στην απόδοση μονοπύρηνων CPUs σταμάτησε να ακολουθεί τον ρυθμό του Moore's Law**.

➡️ Έτσι, μεταβήκαμε σε:

- **Πολυπύρηνες αρχιτεκτονικές (multicore)**
- **Εξειδικευμένα συστήματα** (π.χ. GPU, AI accelerators)
- **Βελτιώσεις ενεργειακής αποδοτικότητας (performance per watt)**

✅ **Πολυεπεξεργαστές & Πολυπύρηνα συστήματα**

🔹 **Πολυπύρηνοι (Multicore) μικροεπεξεργαστές**

- Περιέχουν **περισσότερους του ενός πυρήνες** (processors) στο ίδιο chip.
- Αυτοί οι πυρήνες μπορούν να εκτελούν **παράλληλα διαφορετικά νήματα ή προγράμματα**.

🔹 **Απαίτηση: Ρητά Παράλληλος Προγραμματισμός**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Δηλαδή, ο **προγραμματιστής πρέπει να σχεδιάσει τον κώδικα με παράλληλη εκτέλεση** (multithreading).
- Σε αντίθεση με την **παραλληλία επιπέδου εντολών (ILP)**, που είναι **αυτόματη στο υλικό**, εδώ απαιτείται **σχεδιασμός σε επίπεδο λογισμικού**.

🔹 **Προκλήσεις:**

- Δημιουργία αποδοτικού, παράλληλου κώδικα.
- **Εξισορρόπηση φορτίου** μεταξύ πυρήνων (ώστε όλοι να εργάζονται ισότιμα).
- **Βελτιστοποίηση συγχρονισμού και επικοινωνίας** (για αποφυγή καθυστερήσεων/συγκρούσεων).

🧩 Μορφές Παραλληλίας

Τύπος Παραλληλίας	Περιγραφή
ILP (Instruction-Level Parallelism)	Παράλληλη εκτέλεση εντολών **του ίδιου προγράμματος** μέσω pipelining, superscalar, κ.λπ. (π.χ. 4 εντολές ανά κύκλο).
DLP (Data-Level Parallelism)	Εκτέλεση της **ίδιας λειτουργίας** σε **πολλαπλά δεδομένα** (π.χ. SIMD – π.χ. vector addition).
TLP (Thread-Level Parallelism)	Εκτέλεση **πολλαπλών νημάτων** (threads) **ταυτόχρονα** – είτε στον ίδιο είτε σε διαφορετικούς πυρήνες.

📊 Μετροπρογράμματα SPEC (π.χ. SPEC CPU2006)

Τι είναι:

- **Σύνολα προγραμμάτων/δοκιμών** που μετρούν **την απόδοση** συστημάτων (κυρίως CPU).
- Ανήκουν στον οργανισμό **SPEC (Standard Performance Evaluation Corporation)**.

Χαρακτηριστικά:

- Προσομοιώνουν **ρεαλιστικά προγράμματα**.
- **Ελάχιστη Ε/Ε (I/O)** ώστε να μετριέται καθαρά η **CPU**.
- **Κανονικοποίηση** των αποτελεσμάτων σε σχέση με έναν **υπολογιστή αναφοράς**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Η απόδοση συνοψίζεται με **γεωμετρικό μέσο** των μεμονωμένων αποτελεσμάτων.

Παραδείγματα:

- **CINT2006**: για **ακέραιες** πράξεις.
- **CFP2006**: για **κινητής υποδιαστολής** (floating point) πράξεις.

📌 **Συμπέρασμα**

- Η **μετάβαση σε πολυπύρηνους επεξεργαστές** απαιτεί **νέα στρατηγική ανάπτυξης λογισμικού**.
- Η απόδοση εξαρτάται πλέον όχι μόνο από το hardware αλλά και από το **πόσο αποδοτικά γράφεται ο κώδικας**.
- Οι **μετροπρογράμματα SPEC** είναι απαραίτητα για **τυποποιημένη και αντικειμενική μέτρηση απόδοσης**.
-

Τα μετροπρογράμματα SPECspeed 2017 Integer όταν εκτελούνται σε έναν Intel Xeon E5-2650L στα 1.8 GHz

Περιγραφή	Όνομα	Πλήθος εντολών × 10^9	CPI	Χρόνος κύκλου ρολογιού (sec × 10^{-9})	Χρόνος εκτέλεσης (sec)	Χρόνος αναφοράς (sec)	SPECratio
Διερμηνευτής Perl	perlbench	2684	0.42	0.556	627	1774	2.83
Μεταγλωττιστής C της GNU	gcc	2322	0.67	0.556	863	3976	4.61
Σχεδίαση διαδρομής (route planning)	mcf	1786	1.22	0.556	1215	4721	3.89
Προσομοίωση διακριτών συμβάντων – δίκτυο υπολογιστών	omnetpp	1107	0.82	0.556	507	1630	3.21
Μετατροπή XML σε HTML μέσω XSLT	xalancbmk	1314	0.75	0.556	549	1417	2.58
Συμπίεση βίντεο	x264	4488	0.32	0.556	813	1763	2.17
Τεχνητή νοημοσύνη: Αναζήτηση δέντρου alpha-beta (παιχνίδι σκακιού)	deepsjeng	2216	0.57	0.556	698	1432	2.05
Τεχνητή νοημοσύνη: Αναζήτηση δέντρου MonteCarlo (παιχνίδι Go)	leela	2236	0.79	0.556	987	1703	1.73
Τεχνητή νοημοσύνη: Γεννήτρια αναδρομικής λύσης (παιχνίδι Sudoku)	exchange2	6683	0.46	0.556	1718	2939	1.71
Γενική συμπίεση δεδομένων	xz	8533	1.32	0.556	6290	6182	0.98
Γεωμετρικός μέσος							2.36

Εξήγηση πίνακα

Περιγραφή	Όνομα	Πλήθος εντολών × 10^9	CPI	Χρόνος κύκλου ρολογιού (sec × 10^{-9})	Χρόνος εκτέλεσης (sec)	Χρόνος αναφοράς (sec)	SPECratio
Διερμηνευτής Perl	perlbench	2684	0.42	0.556	627	1774	2.83
Μεταγλωττιστής C της GNU	gcc	2322	0.67	0.556	863	3976	4.61
Τεχνητή νοημοσύνη: Γεννήτρια αναδρομικής λύσης (παιχνίδι Sudoku)	exchange2	6683	0.46	0.556	1718	2939	1.71
Γενική συμπίεση δεδομένων	xz	8533	1.32	0.556	6290	6182	0.98
Γεωμετρικός μέσος							2.36

Χρόνος εκτέλεσης = ICx10^9 x CPI x Tc

Χρ. εκτέλεσης στον υπολογιστή αναφοράς

Χρ.αναφοράς / Χρ. εκτέλεσης

Τύπος:

$$\left(\prod_{i=1}^{n} \text{Λόγος χρόνου εκτέλεσης}_i\right)^{\frac{1}{n}}$$

ή όπως φαίνεται στην εικόνα:

$$\sqrt[n]{\prod_{i=1}^{n} \text{Λόγος χρόνου εκτέλεσης}_i}$$

Τι σημαίνει:

- Ο **λόγος χρόνου εκτέλεσης** για κάθε πρόγραμμα είναι συνήθως:

$$\frac{\text{Χρόνος υπολογιστή αναφοράς}}{\text{Χρόνος συστήματος δοκιμής}} = \text{SPECratio}_i$$

- Ο γεωμετρικός μέσος αυτού του λόγου για όλα τα benchmarks είναι:
  - **Πιο δίκαιος** από τον αριθμητικό μέσο (δεν επηρεάζεται δυσανάλογα από ακραίες τιμές).
  - Χρησιμοποιείται ευρέως σε **μετρήσεις απόδοσης**, π.χ. **SPEC CPU benchmarks**.

⚙️ Τι μετρά το SPEC Power;

Το SPEC Power μετρά δύο βασικά μεγέθη:

1. **Απόδοση**:
   - Μονάδα: ssj_ops/sec
   - Αντιπροσωπεύει τον αριθμό συναλλαγών (server-side Java operations) που εκτελούνται ανά δευτερόλεπτο.
2. **Κατανάλωση ισχύος**:
   - Μονάδα: Watt (Joule/sec)
   - Πόση ηλεκτρική ισχύ καταναλώνεται σε κάθε επίπεδο φορτίου.

📐 Τύπος για συνολική ενεργειακή απόδοση:

$$\text{Ολικό } \frac{\text{ssj_ops}}{\text{Watt}} = \frac{\sum_{i=0}^{10} \text{ssj_ops}_i}{\sum_{i=0}^{10} \text{power}_i}$$

- Δηλαδή:
  Το **άθροισμα των συναλλαγών ανά δευτερόλεπτο** σε όλα τα επίπεδα φορτίου, διαιρεμένο με το **άθροισμα της αντίστοιχης ισχύος** που καταναλώθηκε.

🎯 Τι σημαίνει το αποτέλεσμα:

- Όσο **υψηλότερο** το αποτέλεσμα (ssj_ops/Watt), τόσο **καλύτερη ενεργειακή απόδοση** έχει ο server.
- Είναι κρίσιμο για:
  - **Data centers** όπου η ενέργεια κοστίζει ακριβά.
  - **Περιβαλλοντικά ευαίσθητους σχεδιασμούς**.
  - **Συγκρίσεις μεταξύ επεξεργαστών/διακομιστών** όχι μόνο με βάση την ταχύτητα αλλά και την **ενεργειακή αποδοτικότητα**.

✅ Συμπέρασμα:

Το **SPEC Power** προσφέρει έναν **ισορροπημένο δείκτη** μεταξύ **απόδοσης** και **ισχύος**, και είναι πολύ σημαντικός σήμερα όπου η **ενεργειακή αποδοτικότητα** είναι εξίσου κρίσιμη με την **υψηλή ταχύτητα**.

Αν θέλεις, μπορώ να σου δείξω παράδειγμα με αριθμούς ή να συγκρίνουμε δύο server βάσει του SPEC Power δείκτη.

Το μετροπρόγραμμα SPECpower_ssj2008 στον Xeon E5-2650L

Στόχος φορτίου %	Απόδοση (ssj_ops)	Μέση ισχύς (Watt)
100%	4 864 136	347
90%	4 389 196	312
80%	3 905 724	278
70%	3 418 737	241
60%	2 925 811	212
50%	2 439 017	183
40%	1 951 394	160
30%	1 461 411	141
20%	974 045	128
10%	485 973	115
0%	0	48
Σύνολο	26 815 444	2 165
Σssj_ops / Σpower =		12 385

🧠 **Πλάνες (Fallacies)**

- Είναι **παρανοήσεις**, **λανθασμένες αντιλήψεις** που **φαίνονται λογικές**, αλλά **δεν ισχύουν** στην πράξη.
- Προκαλούν σύγχυση γιατί στηρίζονται σε **φαινομενικά ορθά επιχειρήματα**.
- Το βιβλίο τις καταρρίπτει με **αντιπαραδείγματα**.

🔍 **Παράδειγμα πλάνης:**

"Η υψηλότερη συχνότητα ρολογιού σημαίνει πάντα καλύτερη απόδοση."

📉 **Αντιπαράδειγμα:** Μια CPU με υψηλό CPI μπορεί να είναι πιο αργή από μία χαμηλότερης συχνότητας αλλά χαμηλού CPI.

⚠️ **Παγίδες (Pitfalls)**

- Είναι **συχνά λάθη** που κάνουν μαθητές, μηχανικοί ή προγραμματιστές επειδή **εφαρμόζουν σωστές αρχές σε λάθος πλαίσιο**.
- Είναι παραπλανητικές **γενικεύσεις**.

🔍 **Παράδειγμα παγίδας:**

"Αν αυξήσουμε τον αριθμό των πυρήνων, θα αυξηθεί γραμμικά η απόδοση."

📉 Αυτό **ισχύει μόνο** εάν:

- το πρόγραμμα είναι πλήρως παραλληλίσιμο,
- δεν υπάρχουν bottlenecks (π.χ. μνήμη),

ΕΛΕΝΗ ΖΑΦΕΙ

- και η επικοινωνία μεταξύ πυρήνων είναι αμελητέα.

🎯 **Συμπέρασμα:**

- Οι **πλάνες** μάς προστατεύουν από **λανθασμένα συμπεράσματα**.
- Οι **παγίδες** μάς υπενθυμίζουν ότι **σωστές αρχές χρειάζονται προσοχή στο πλαίσιο εφαρμογής**.

⚠️ Παγίδα: "**Αν επιταχύνω ένα κομμάτι του προγράμματος, θα αυξηθεί ανάλογα και η συνολική απόδοση**"

Αυτό δεν ισχύει πάντα, και γι' αυτό είναι παγίδα.

📐 **Νόμος του Amdahl** – Τύπος:

$$\text{Χρόνος μετά τη βελτίωση} = \frac{\text{Χρόνος που βελτιώνεται}}{\text{Ποσοστό βελτίωσης}} + \text{Χρόνος που δεν επηρεάζεται}$$

🔍 Παράδειγμα:

- Ο πολλαπλασιασμός καταναλώνει 80 sec από τα συνολικά 100 sec.
- Θέλουμε το πρόγραμμα να γίνει 5 φορές πιο γρήγορο:

$$\frac{100}{5} = 20 \text{ sec}$$

Άρα:

$$\frac{80}{n} + 20 = 20 \Rightarrow \frac{80}{n} = 0 \Rightarrow n = \infty$$

👉 Αδύνατο! Δηλαδή, ακόμα κι αν κάναμε τον πολλαπλασιασμό άπειρα γρήγορο, το ελάχιστο δυνατό συνολικό χρόνο είναι 20 sec (το μέρος που δεν βελτιώνεται).

✅ **Συμπέρασμα – Τι μας διδάσκει ο Νόμος του Amdahl:**

- Η **μέγιστη επιτάχυνση** περιορίζεται από **το ποσοστό του προγράμματος που δεν επιταχύνεται**.
- Πρέπει να **επικεντρωνόμαστε στα σημεία που καταναλώνουν τον περισσότερο χρόνο** (συνήθως τα πιο "κοινά" ή συχνά επαναλαμβανόμενα μέρη).
- Η **ολική επιτάχυνση δεν είναι γραμμική** ούτε "δίκαια κατανεμημένη".

💡 **Μήνυμα:**

Μη σπαταλάς χρόνο βελτιστοποιώντας κάτι που τρέχει σπάνια.

❌ **Πλάνη: "Ο υπολογιστής καταναλώνει πολύ λίγη ισχύ όταν είναι αδρανής"**

Αυτό **δεν ισχύει**, και το παράδειγμα με τον **Intel Core i7** το αποδεικνύει:

🔍 **Παράδειγμα μέτρησης:**

Φορτίο CPU	Ισχύς	Ποσοστό σε σχέση με μέγιστο
100%	258W	100%
50%	170W	66%
10%	121W	47%

📌 **Συμπέρασμα**:
Η κατανάλωση **δεν μειώνεται γραμμικά** με τη χρήση.
Ακόμα και με **10% χρήση**, καταναλώνει **σχεδόν τη μισή ισχύ**!

🏢 **Παράδειγμα από Google Data Centers:**

- Οι servers λειτουργούν συνήθως στο **10% – 50%** του φορτίου.
- Στο **100% φορτίο** λειτουργούν **λιγότερο από 1% του χρόνου**.

👉 Άρα, **η περισσότερη ενέργεια καταναλώνεται σε μερική χρήση**, όχι στο peak.

🎯 **Μήνυμα – Τι πρέπει να γίνει:**

- Οι επεξεργαστές πρέπει να σχεδιάζονται έτσι ώστε η **ισχύς να κλιμακώνεται ορθά με το φορτίο**.
- Σημαντικό για:
  - **Ενεργειακή απόδοση**
  - **Μείωση κόστους λειτουργίας (π.χ. σε data centers)**
  - **Περιβαλλοντικούς στόχους (green computing)**

✅ **Συμπέρασμα:**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Η ιδέα ότι ένας υπολογιστής που "δεν κάνει τίποτα" καταναλώνει σχεδόν μηδενική ισχύ είναι **πλάνη**.

Χρειάζεται **βελτιστοποίηση κατανάλωσης** σε όλα τα επίπεδα χρήσης, όχι μόνο στο 100% φορτίο.

Αν θες, μπορώ να δείξω πώς λειτουργεί η τεχνολογία DVFS (Dynamic Voltage and Frequency Scaling) για να πετυχαίνεται αυτό.

❌ **Πλάνη: "Όσο μεγαλύτερο το MIPS, τόσο καλύτερη η CPU"**

Αυτό **δεν ισχύει**, για διάφορους λόγους.

📐 **Ορισμός MIPS:**

📐 Ορισμός MIPS:

$$\text{MIPS} = \frac{\text{Πλήθος εντολών}}{\text{Χρόνος εκτέλεσης} \times 10^6}$$

Αντικαθιστώντας τον χρόνο εκτέλεσης:

$$\text{Χρόνος εκτέλεσης} = \frac{\text{Πλήθος εντολών} \times \text{CPI}}{\text{Ρυθμός ρολογιού}}$$

$\Rightarrow$

$$\text{MIPS} = \frac{\text{Ρυθμός ρολογιού}}{\text{CPI} \times 10^6}$$

📌 Άρα, το MIPS **δεν εξαρτάται από το πρόγραμμα**, αλλά από:

- τη συχνότητα (ρολόι),
- το CPI (κύκλοι ανά εντολή).

⚠️ **Γιατί το MIPS δεν είναι καλός δείκτης απόδοσης;**

1. **Παραβλέπει τη διαφορά στην πολυπλοκότητα των εντολών**:
   - Μια ISA μπορεί να έχει απλές εντολές (RISC) και άλλη σύνθετες (CISC).
   - Άρα: **περισσότερες εντολές ≠ περισσότερη δουλειά**.
2. **Διαφορετικά προγράμματα → διαφορετικά CPI**, ακόμα στην ίδια CPU.
3. **Δύο CPU με ίδιο MIPS δεν έχουν απαραίτητα ίση απόδοση**.

ΕΛΕΝΗ ΤΙΡΗ

✅ **Ποιος δείκτης είναι καλύτερος;**

- Ο **χρόνος εκτέλεσης (execution time)** είναι **το πιο αξιόπιστο μέτρο** απόδοσης.
- Χρησιμοποιούνται benchmarks όπως:
  - **SPEC CPU**
  - **GEM5 simulation**
  - **Real workload timing**

🎯 **Συμπέρασμα:**

Μην εμπιστεύεσαι το MIPS ως καθολικό μέτρο απόδοσης.
Χρησιμοποίησε **χρόνο εκτέλεσης** και **συγκριτικά benchmarks**.

✅ **1. Η σχέση κόστους-απόδοσης συνεχώς βελτιώνεται**

- Χάρη στην **τεχνολογική εξέλιξη** (π.χ. μικρότερα τρανζίστορ, καλύτερες διεργασίες παραγωγής).
- Πιο **ισχυροί υπολογιστές** με **χαμηλότερο κόστος ανά μονάδα απόδοσης**.

✅ **2. Ιεραρχία στρωμάτων αφαίρεσης**

- Οι υπολογιστές σχεδιάζονται **σε επίπεδα** (hardware → microarchitecture → ISA → OS → compiler → application).
- Αυτή η ιεραρχία επιτρέπει:
  - **Απλοποίηση σχεδιασμού**
  - **Ανεξαρτησία απόδοσης και υλοποίησης**

✅ **3. Αρχιτεκτονική συνόλου εντολών (ISA)**

- Είναι το **σημείο επαφής μεταξύ υλικού και λογισμικού**.
- Καθορίζει **ποιες εντολές βλέπει ο μεταγλωττιστής** και **πώς μεταφράζεται ο κώδικας σε εκτελέσιμη μορφή**.

✅ **4. Χρόνος εκτέλεσης: το καλύτερο μέτρο απόδοσης**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Πιο αξιόπιστο από MIPS, CPI ή απλές θεωρητικές μετρήσεις.
- Μετρά **την πραγματική απόδοση** σε συγκεκριμένο έργο (benchmark).

✅ **5. Η ισχύς είναι περιοριστικός παράγοντας**

- Δεν μπορούμε πλέον να αυξάνουμε την απόδοση μόνο αυξάνοντας συχνότητα (λόγω του **power wall**).
- Η **ενεργειακή αποδοτικότητα** είναι κρίσιμη (ειδικά σε mobile ή data centers).

✅ **6. Η παραλληλία είναι το κλειδί για μελλοντική απόδοση**

- Με τη **στασιμότητα της μονοπύρηνης απόδοσης**, η **παράλληλη επεξεργασία** (multicore, SIMD, GPU, threads) είναι απαραίτητη.
- Χρειάζεται και **παράλληλος προγραμματισμός** για να αξιοποιηθεί.

🧠 **Συνολικό μήνυμα:**

Η κατανόηση της αρχιτεκτονικής υπολογιστών απαιτεί **διεπιστημονική προσέγγιση**: τεχνολογία, υλικό, λογισμικό, απόδοση και ενεργειακοί περιορισμοί δουλεύουν **μαζί**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

2 Η ΓΛΩΣΣΑ ΤΟΥ ΥΠΟΛΟΓΙΣΤΗ

COMPUTER ARCHITECTURE 1

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Τι είναι το "Σύνολο Εντολών";**
  - Είναι όλες οι εντολές που καταλαβαίνει και μπορεί να εκτελέσει ένας υπολογιστής.
  - Κάθε υπολογιστής έχει το δικό του μοναδικό σύνολο εντολών.
  - Παρόλο που διαφέρουν, πολλές εντολές μοιάζουν μεταξύ των υπολογιστών.
  - Οι πρώτοι υπολογιστές είχαν πολύ απλές εντολές, γιατί ήταν πιο εύκολο να φτιαχτούν έτσι.
  - Ακόμα και σήμερα, πολλοί σύγχρονοι υπολογιστές χρησιμοποιούν απλά σύνολα εντολών, γιατί είναι πιο γρήγορα και αποδοτικά.

- **Η Ιδέα του «Αποθηκευμένου Προγράμματος»**
  - Είναι η ιδέα ότι:
    - Οι εντολές (π.χ. πρόσθεσε, αποθήκευσε, σύγκρινε)
    - Και τα δεδομένα (π.χ. αριθμοί, χαρακτήρες)
    - Μπορούν να αποθηκεύονται στη μνήμη σαν απλοί αριθμοί.
  - Αυτή η ιδέα επέτρεψε τη δημιουργία των σημερινών υπολογιστών, που αποθηκεύουν και εκτελούν προγράμματα από τη μνήμη τους.
  - Λέγεται "stored program computer".

- **Το Σύνολο Εντολών του RISC-V**
  - Το βιβλίο χρησιμοποιεί το RISC-V ως παράδειγμα για να εξηγήσει πώς λειτουργούν τα σύνολα εντολών.
  - Το RISC-V:
    - Φτιάχτηκε στο Πανεπιστήμιο Berkeley.
    - Είναι ανοιχτό (δωρεάν, χωρίς περιορισμούς χρήσης).
    - Το διαχειρίζεται το Ίδρυμα RISC-V (riscv.org).
  - Είναι αντιπροσωπευτικό μοντέλο για πολλές σύγχρονες αρχιτεκτονικές.
  - Στο βιβλίο υπάρχει “Κάρτα Αναφοράς” με τις βασικές εντολές RISC-V.
  - Παρόμοια σύνολα εντολών (σαν το RISC-V) χρησιμοποιούνται σε πολλές συσκευές όπως:
    - Κινητά τηλέφωνα

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Δρομολογητές δικτύου (routers)
- Ψηφιακές φωτογραφικές μηχανές
- Εκτυπωτές
- Άλλες «έξυπνες» ηλεκτρονικές συσκευές

🔹 §2.2 Λειτουργίες του υλικού των υπολογιστών

❓ Ποιες είναι οι βασικές αριθμητικές πράξεις που υποστηρίζονται από το υλικό;

✅ Πρόσθεση και αφαίρεση, με τρεις τελεστέους:

- Δύο για τους αριθμούς της πράξης
- Ένας για το αποτέλεσμα

❓ Πώς γράφεται η πρόσθεση σε γλώσσα μηχανής (RISC-V);

✅ Παράδειγμα εντολής:

add a, b, c

➡️ Αυτό σημαίνει: η τιμή b + c αποθηκεύεται στο a

❓ Ποια μορφή έχουν όλες οι αριθμητικές εντολές στον RISC-V;

✅ Όλες οι αριθμητικές εντολές ακολουθούν τη μορφή με τρεις τελεστέους:

- Πηγή 1, Πηγή 2, Προορισμός (result)

❓ Ποια σχεδιαστική αρχή ακολουθείται για τις αριθμητικές εντολές;

✅ Σχεδιαστική Αρχή 1:

"Η απλότητα ευνοεί την κανονικότητα"

- Η κανονικότητα κάνει πιο εύκολη την υλοποίηση
- Η απλότητα επιτρέπει υψηλότερη απόδοση με χαμηλότερο κόστος

❓ Πώς μεταφράζεται μια σύνθετη αριθμητική έκφραση από C σε RISC-V;

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ Παράδειγμα σε γλώσσα C:

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

✅ Αντίστοιχος κώδικας RISC-V:

```
add t0, g, h   // t0 = g + h
add t1, i, j   // t1 = i + j
sub f, t0, t1  // f = t0 - t1
```

🔹 §2.3 Τελεστέοι του υλικού των υπολογιστών

❓ Τι χρησιμοποιείται ως τελεστέοι στις αριθμητικές εντολές;

✅ Οι τελεστέοι (δηλαδή οι αριθμοί που συμμετέχουν στις πράξεις) είναι καταχωρητές.

❓ Πόσους καταχωρητές έχει ο RISC-V και τι μέγεθος έχουν;

✅ Ο RISC-V διαθέτει 32 καταχωρητές των 32 bit για λειτουργίες με λέξεις (word)
✅ Επίσης, έχει 32 καταχωρητές γενικού σκοπού των 64 bit (doubleword) — από x0 έως x31

❓ Γιατί χρησιμοποιούνται οι καταχωρητές;

✅ Επειδή κρατούν δεδομένα που προσπελάζονται συχνά και είναι πολύ πιο γρήγοροι από την κύρια μνήμη.

❓ Ποια είναι η Σχεδιαστική Αρχή 2;

✅ "Το μικρότερο είναι ταχύτερο"
➡️ Οι καταχωρητές είναι λίγοι σε αριθμό, αλλά πολύ γρήγοροι.
➡️ Αντίθετα, η κύρια μνήμη έχει εκατομμύρια θέσεων, αλλά είναι πολύ πιο αργή.

❓ Ποιοι είναι οι ρόλοι συγκεκριμένων καταχωρητών στον RISC-V;

Καταχωρητής	Ρόλος
x0	Πάντα έχει την τιμή 0

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Καταχωρητής	Ρόλος
x1	Διεύθυνση επιστροφής από συνάρτηση
x2	Δείκτης στοίβας (stack pointer)
x3	Καθολικός δείκτης (global pointer)
x4	Δείκτης νημάτων (thread pointer)
x5 – x7, x28 – x31	Καταχωρητές προσωρινών τιμών
x8	Δείκτης πλαισίου (frame pointer)
x9, x18 – x27	Αποθηκευμένοι καταχωρητές (saved registers)
x10 – x11	Ορίσματα και αποτελέσματα συναρτήσεων
x12 – x17	Επιπλέον ορίσματα συναρτήσεων

❓ Παράδειγμα με τελεστέους καταχωρητές: Πώς γράφεται το f = (g + h) - (i + j);;

✅ Αν οι μεταβλητές βρίσκονται στους εξής καταχωρητές:

- f → x19
- g → x20
- h → x21
- i → x22
- j → x23

✅ Ο μεταγλωττισμένος RISC-V κώδικας είναι:

```
add x5, x20, x21   // x5 = g + h
add x6, x22, x23   // x6 = i + j
sub x19, x5, x6    // x19 = (g + h) - (i + j)
```

🔹 Τελεστέοι Μνήμης – Διευθυνσιοδότηση – Ευθυγράμμιση – Endianess

---

❓ Πότε χρησιμοποιούμε την κύρια μνήμη στους υπολογιστές;

✅ Όταν τα δεδομένα είναι μεγαλύτερα ή πιο σύνθετα, όπως:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Πίνακες
- Δομές (structs)
- Δυναμικά δεδομένα (π.χ. στον σωρό/heap)

❓ Πώς γίνονται αριθμητικές πράξεις όταν τα δεδομένα είναι στη μνήμη;

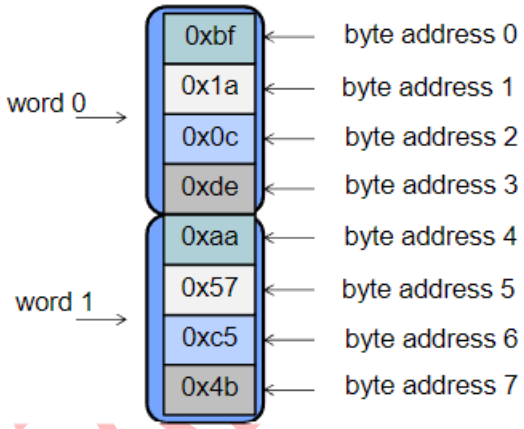
✅ Τα βήματα είναι:

1. Φορτώνουμε τις τιμές από τη μνήμη σε καταχωρητές
2. Κάνουμε την πράξη στους καταχωρητές
3. Αν χρειάζεται, αποθηκεύουμε το αποτέλεσμα πίσω στη μνήμη

❓ Πώς είναι οργανωμένη η μνήμη στον RISC-V;

✅ Η μνήμη είναι διευθυνσιοδοτημένη κατά byte

➡️ Δηλαδή, κάθε διεύθυνση δείχνει 1 byte (8 bit)



❓ Πώς αποθηκεύονται οι λέξεις (word) των 32 bit στη μνήμη;

✅ Κάθε λέξη 32 bit αποτελείται από 4 byte

➡️ Τα 4 bytes καταλαμβάνουν 4 συνεχόμενες διευθύνσεις

❓ Τι σημαίνει ότι ο RISC-V είναι αρχιτεκτονική “little endian”;

✅ Σε little endian αρχιτεκτονίες:

- Η διεύθυνση της λέξης δείχνει στο λιγότερο σημαντικό byte (LSB).
- Τα bytes μιας λέξης τοποθετούνται από το λιγότερο προς το περισσότερο σημαντικό.

➡️ Παράδειγμα λέξης: 0x33221100
Little endian διάταξη στη μνήμη (από τη μικρότερη διεύθυνση):

0x00 → διεύθυνση K

0x11 → K+1

0x22 → K+2

0x33 → K+3

🧠 Παράδειγμα μνήμης (διευθύνσεις & περιεχόμενο)

Παράδειγμα περιεχομένου μνήμης (σε little endian):

Διεύθυνση: 0 1 2 3 4 5 6 7

Τιμή (byte): 0xbf 0x1a 0xde 0x0c 0xaa 0x57 0x4b 0xc5

- Word 0 (από byte 0 ως 3): 0x0cde1abf
- Word 1 (από byte 4 ως 7): 0xc54b57aa

❓ Τι είναι η “big endian” αρχιτεκτονική και πώς διαφέρει;

✅ Σε big endian:

- Η διεύθυνση της λέξης δείχνει στο περισσότερο σημαντικό byte (MSB)

➡️ Για την ίδια λέξη 0x33221100, η διάταξη θα ήταν:

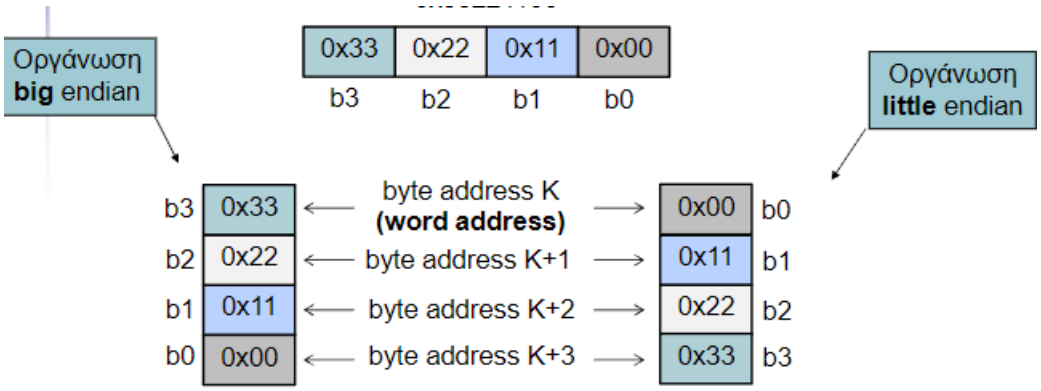
0x33 → διεύθυνση K

0x22 → K+1

0x11 → K+2

0x00 → K+3

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



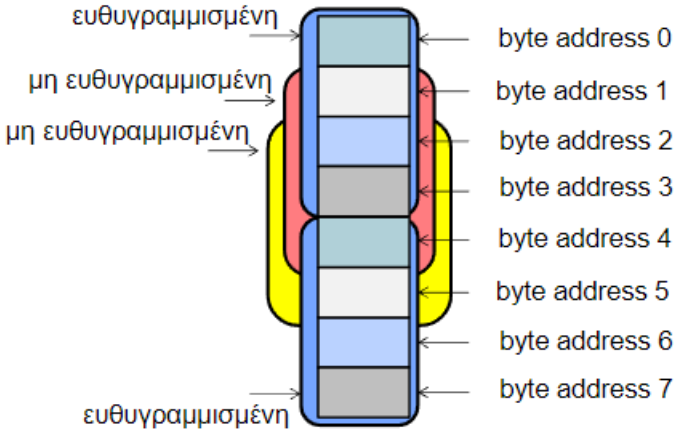
❓ Τι σημαίνει "ευθυγραμμισμένη" και "μη ευθυγραμμισμένη" λέξη;

✅ Ευθυγραμμισμένη λέξη:

- Η λέξη ξεκινά από διεύθυνση πολλαπλάσια του μεγέθους της (π.χ. για 4 bytes → από διεύθυνση 0, 4, 8...)

✅ Μη ευθυγραμμισμένη λέξη:

- Η λέξη ξεκινά από τυχαία διεύθυνση, όχι πολλαπλάσια (π.χ. από διεύθυνση 1, 2, 5...)



❓ Επιτρέπει η αρχιτεκτονική RISC-V μη ευθυγραμμισμένες λέξεις;

✅ Ναι, σε αντίθεση με άλλες αρχιτεκτονικές που απαιτούν ευθυγράμμιση, η RISC-V επιτρέπει και μη ευθυγραμμισμένη πρόσβαση στη μνήμη.

🔹 Παράδειγμα με τελεστέους μνήμης – Καταχωρητές vs. Μνήμη – Άμεσοι τελεστέοι

❓ Παράδειγμα C: Πώς μεταφράζεται η εντολή A[12] = h + A[8]; σε RISC-V;

✅ Υποθέσεις:

- Η μεταβλητή h βρίσκεται στον καταχωρητή x21
- Η διεύθυνση βάσης του πίνακα A βρίσκεται στον x22
- Κάθε στοιχείο του πίνακα A έχει μέγεθος 4 bytes

✅ Επεξήγηση αριθμοδείκτη:

- A[8] → 8 × 4 = 32
- A[12] → 12 × 4 = 48

✅ Μεταγλωττισμένος κώδικας RISC-V:

```
lw x9, 32(x22)      // Φόρτωσε την τιμή A[8] από τη μνήμη στον x9
add x9, x21, x9     // Πρόσθεσε την τιμή του h (x21) με το A[8] και αποθήκευσέ το στον x9
sw x9, 48(x22)      // Αποθήκευσε το αποτέλεσμα στη θέση A[12]
```

❓ Γιατί προτιμούμε τους καταχωρητές αντί για τη μνήμη;

✅ Γιατί οι καταχωρητές είναι πολύ πιο γρήγοροι από την προσπέλαση της μνήμης.

- Όταν τα δεδομένα βρίσκονται στη μνήμη, χρειάζονται επιπλέον εντολές για φόρτωση (load) και αποθήκευση (store).
- Η εκτέλεση πράξεων απευθείας στη μνήμη δεν επιτρέπεται – πάντα μέσω καταχωρητών.

❓ Τι κάνει ο μεταγλωττιστής για να βελτιστοποιήσει την απόδοση;

✅ Ο μεταγλωττιστής:

- Χρησιμοποιεί τους καταχωρητές όσο το δυνατόν περισσότερο.
- Στέλνει δεδομένα στη μνήμη μόνο όταν είναι απαραίτητο (π.χ. αν μια μεταβλητή χρησιμοποιείται σπάνια).
- Στόχος: ελαχιστοποίηση των φορτώσεων και αποθηκεύσεων

➡️ Η βελτιστοποίηση της χρήσης των καταχωρητών είναι πολύ σημαντική για την απόδοση του προγράμματος.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ Τι είναι οι άμεσοι τελεστέοι (immediate operands);

✅ Είναι σταθερές τιμές που γράφονται απευθείας μέσα στην εντολή.

✅ Παράδειγμα:

addi x22, x22, 4

➡️ Πρόσθεσε την τιμή 4 απευθείας στον x22 και αποθήκευσε το αποτέλεσμα στον x22

❓ Ποια είναι τα πλεονεκτήματα των άμεσων τελεστέων;

✅ Πλεονεκτήματα:

- Επιταχύνουν συχνές περιπτώσεις (π.χ. πρόσθεση μικρής σταθερής τιμής)
- Αποφεύγουν την ανάγκη για ξεχωριστή εντολή φόρτωσης της σταθεράς από τη μνήμη

✅ Ιδανικοί για σταθερές μικρής τιμής, που είναι πολύ συχνές σε προγράμματα.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

§2.4 Προσημασμένοι και απρόσημοι (μη προσημασμένοι) ακέραιοι αριθμοί

❓ **Τι είναι οι μη προσημασμένοι (unsigned) δυαδικοί ακέραιοι;**

✅ Είναι **θετικοί αριθμοί** που αναπαρίστανται με **n bits**, χωρίς πρόσημο.

❓ **Ποιος είναι ο τύπος αναπαράστασης ενός μη προσημασμένου n-bit αριθμού;**

✅

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

❓ **Ποιο είναι το εύρος τιμών για μη προσημασμένους αριθμούς;**

✅

- Από **0** έως **2n−12^n - 12n−1**

✔️ Παραδείγματα:

- **Με 32 bit**: 0 έως **4.294.967.295**
- **Με 64 bit**: 0 έως **18.446.744.073.709.551.615**

❓ **Παράδειγμα μη προσημασμένου αριθμού:**

✅ Δυαδικό: $...00001011_2$

Υπολογισμός:

8+0+2+1=118 + 0 + 2 + 1 = 118+0+2+1=11

❓ **Τι είναι οι προσημασμένοι (signed) ακέραιοι σε μορφή συμπληρώματος ως προς 2;**

✅ Είναι αριθμοί με πρόσημο, αναπαριστώνται σε n bits με:

- Το **τελευταίο (MSB)** bit να λειτουργεί ως **bit προσήμου**
  - **1 → αρνητικός αριθμός**
  - **0 → θετικός ή μηδέν**

❓ **Ποιος είναι ο τύπος αναπαράστασης προσημασμένων αριθμών σε συμπλήρωμα 2;**

✅

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

❓ **Ποιο είναι το εύρος τιμών για προσημασμένους αριθμούς;**

✅

- Από **−2n−1-2^{n-1}−2n−1** έως **2n−1−12^{n-1} - 12n−1−1**

✔ Παραδείγματα:

- **Με 32 bit**: −2.147.483.648 έως **+2.147.483.647**
- **Με 64 bit**: −9.223.372.036.854.775.808 έως **+9.223.372.036.854.775.807**

❓ **Παράδειγμα προσημασμένου αριθμού (32-bit):**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ Δυαδικό: $111...1100_2$

Υπολογισμός:

−231+(230+…+22+0+0)=−2.147.483.648+2.147.483.644=−4-2^{31} + (2^{30} + \ldots + 2^2 + 0 + 0) = -2.147.483.648 + 2.147.483.644 = -4−231+(230+…+22+0+0)=−2.147.483.648+2.147.483.644=−4

❓ **Ποιοι αριθμοί έχουν ίδια αναπαράσταση σε signed και unsigned μορφή;**

✅ Όλοι οι **μη αρνητικοί αριθμοί** (δηλ. αυτοί που ξεκινούν με bit 0)

❓ **Ποια είναι κάποια σημαντικά παραδείγματα προσημασμένων 32-bit αριθμών;**

Δεκαδικός	**Δυαδική μορφή (συμπλήρωμα ως προς 2)**
0	0000 0000 ... 0000
–1	1111 1111 ... 1111
Πιο αρνητικός	1000 0000 ... 0000 (-2^{31} σε 32 bit)
Πιο θετικός	0111 1111 ... 1111 ($+2^{31} - 1$ σε 32 bit)

🔹 **Υπολογισμός Αντιθέτου – Επέκταση Προσήμου – Κωδικοποίηση Εντολών**

❓ **Πώς υπολογίζουμε τον αντίθετο ενός προσημασμένου αριθμού σε μορφή συμπληρώματος ως προς 2;**

✅ Βήματα:

1. **Αντιστροφή όλων των bits** (0 → 1 και 1 → 0)
2. **Προσθήκη 1** στο αποτέλεσμα

❓ **Παράδειγμα: Πώς βρίσκουμε το –2;**

✅

- +2 → 0000 0000 ... 0010_2
- Αντιστροφή: 1111 1111 ... 1101_2

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Προσθέτουμε 1 → 1111 1111 ... 1110_2
  ➡️ Άρα: **–2 = 1111 1111 ... 1110_2**

❓ **Τι είναι η επέκταση προσήμου (sign extension);**

✅ Είναι η διαδικασία κατά την οποία ένας προσημασμένος αριθμός **μεγαλώνει σε bit, χωρίς να αλλάξει η τιμή του**.

❓ **Πώς γίνεται η επέκταση προσήμου;**

✅ Αντιγράφεται το **bit προσήμου (MSB)** στα νέα αριστερά bits:

- Αν το MSB είναι 1 → **συμπληρώνουμε με 1**
- Αν το MSB είναι 0 → **συμπληρώνουμε με 0**

❓ **Πώς διαφέρει η επέκταση σε μη προσημασμένους αριθμούς;**

✅ Σε **μη προσημασμένους (unsigned)** αριθμούς, **πάντα προσθέτουμε μηδενικά** στα αριστερά (zero extension).

❓ **Παραδείγματα επέκτασης από 8 σε 16 bit:**

Αριθμός 8 bit	**Επέκταση σε 16 bit**
+2	0000 0010 0000 0000 0000 0010
–2	1111 1110 1111 1111 1111 1110

❓ **Ποιες εντολές RISC-V σχετίζονται με την επέκταση προσήμου;**

✅

- **lb** (load byte): φορτώνει byte και κάνει **επέκταση προσήμου**
- **lbu** (load byte unsigned): φορτώνει byte και κάνει **επέκταση μηδενικού**

❓ **Πώς αναπαρίστανται οι εντολές σε έναν υπολογιστή;**

✅ Οι εντολές κωδικοποιούνται σε **δυαδική μορφή**, δηλαδή σε **κώδικα μηχανής** (machine code)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Πώς κωδικοποιούνται οι εντολές RISC-V;**

✅ Κάθε εντολή RISC-V:

- Έχει μήκος **32 bits**
- Περιλαμβάνει **operation code (opcode)**, **καταχωρητές**, και **άλλες πληροφορίες**

➡️ Η RISC-V χρησιμοποιεί **λίγες σταθερές μορφές εντολών**, κάτι που κάνει την αρχιτεκτονική **κανονική και απλή**

🔹 **Δεκαεξαδική αναπαράσταση – Μορφές εντολών R και I – Αρχές σχεδίασης**

❓ **Τι είναι η δεκαεξαδική (hex) αναπαράσταση;**

✅ Είναι μια **συμπυκνωμένη μορφή** γραφής σειρών από **δυαδικά ψηφία**, χρησιμοποιώντας τη **βάση 16**.

❓ **Πόσα δυαδικά ψηφία (bits) αντιστοιχούν σε ένα δεκαεξαδικό ψηφίο;**

✅ **4 bits ανά 1 δεκαεξαδικό ψηφίο**

❓ **Παράδειγμα δεκαεξαδικού αριθμού:**

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

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

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Ποια είναι η μορφή R εντολών στον RISC-V;**

✅ Χρησιμοποιείται για εντολές όπως η **add** και περιλαμβάνει τα εξής πεδία:

Πεδίο	Περιγραφή
opcode	Κωδικός λειτουργίας
rd	Καταχωρητής προορισμού (destination)
funct3	Πρόσθετος 3-bit κωδικός λειτουργίας
rs1	1ος καταχωρητής προέλευσης (source 1)
rs2	2ος καταχωρητής προέλευσης (source 2)
funct7	Πρόσθετος 7-bit κωδικός λειτουργίας

📋 **RISC-V Opcodes**

Opcode (bin)	Hex	Decimal	Τύπος	Περιγραφή
0110011	0x33	51	R	Αριθμητικές/Λογικές εντολές μεταξύ καταχωρητών
0010011	0x13	19	I	Αριθμητικές/Λογικές εντολές με άμεσες τιμές
0000011	0x03	3	I	Φόρτωση από μνήμη (Load)
0100011	0x23	35	S	Αποθήκευση σε μνήμη (Store)
1100011	0x63	99	B	Διακλαδώσεις υπό συνθήκη
1101111	0x6F	111	J	Άλμα χωρίς συνθήκη (JAL)
1100111	0x67	103	I	JALR (Register-based jump)
0110111	0x37	55	U	Φόρτωση άμεσης τιμής (LUI)
0010111	0x17	23	U	Πρόσθεση άμεσης σε PC (AUIPC)
0001111	0x0F	15	I	FENCE εντολές (memory barriers)
1110011	0x73	115	I	Σύστημα: ECALL, EBREAK, CSR εντολές

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Πώς οργανώνεται η μορφή R στα bits;**

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

❓ **Παράδειγμα R εντολής: add x9, x20, x21**

0	21	20	0	9	51
0000000	10101	10100	000	01001	0110011

✅ Δυαδική μορφή:

0000000 10101 10100 01001 000 0110011

✅ Δεκαεξαδική αναπαράσταση (machine code):

0x015A04B3

❓ **Ποια είναι η μορφή I εντολών στον RISC-V;**

✅ Χρησιμοποιείται για:

- **Άμεσες αριθμητικές πράξεις** (π.χ. addi)
- **Εντολές φόρτωσης** (π.χ. lw, lb)

❓ **Τι περιλαμβάνει η μορφή I;**

Πεδίο	**Περιγραφή**
immediate	Άμεσος τελεστέος (σε συμπλήρωμα ως προς 2)
rs1	Καταχωρητής προέλευσης ή βάση διεύθυνσης
rd	Καταχωρητής προορισμού
funct3	Κωδικός λειτουργίας
opcode	Κωδικός εντολής

✅ Το immediate επεκτείνεται με **επέκταση προσήμου**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

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

❓ **Ποια σχεδιαστική αρχή σχετίζεται με τις μορφές εντολών;**

✅ **Σχεδιαστική αρχή 3**:

"Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς"

- Αν και οι **πολλαπλές μορφές εντολών** δυσκολεύουν την **αποκωδικοποίηση**,
- Εξασφαλίζουν **ομοιομορφία**: όλες οι εντολές είναι **32-bit**, και έχουν **παρόμοια δομή**.

🔹 **Εντολές Μορφής S στον RISC-V**

❓ **Τι είναι οι εντολές μορφής S στον RISC-V;**

✅ Είναι οι **εντολές αποθήκευσης** (store instructions), όπως η sw, sb, sh, που αποθηκεύουν δεδομένα **από καταχωρητές στη μνήμη**.

❓ **Ποια είναι η βασική διαφορά στη μορφή τους σε σχέση με άλλες μορφές (π.χ. I);**

✅ Χρησιμοποιούν **διαφορετική μορφή του άμεσου τελεστέου (immediate)** — για να επιτευχθεί **σταθερή θέση** των πεδίων rs1 και rs2 μέσα στο encoding.

❓ **Ποια πεδία περιλαμβάνει η μορφή S;**

Πεδίο	**Περιγραφή**
rs1	Καταχωρητής με τη **διεύθυνση βάσης** (base register)
rs2	Καταχωρητής που περιέχει την **τιμή προς αποθήκευση**
immediate	Άμεσος τελεστέος = **σχετική απόσταση** (offset) που προστίθεται στη βάση
funct3	Τύπος αποθήκευσης (π.χ. byte, word)
opcode	Κωδικός λειτουργίας

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Ποια είναι η διάταξη (μορφή) bits σε μια S εντολή;**

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bit	5 bit	5 bit	3 bit	5 bit	7 bit

📌 Έτσι εξασφαλίζεται ότι τα **rs1 και rs2 βρίσκονται πάντα στην ίδια θέση**, όπως και σε άλλες μορφές (π.χ. R), για **ευκολότερη αποκωδικοποίηση**.

🔹 **Υπολογιστές Αποθηκευμένων Προγραμμάτων**

❓ **Πώς αναπαρίστανται οι εντολές στους σύγχρονους υπολογιστές;**

✅ Οι **εντολές** αναπαρίστανται σε **δυαδική μορφή**, όπως και τα **δεδομένα**.
➡️ Δηλαδή, είναι **απλοί αριθμοί** αποθηκευμένοι στη μνήμη.

❓ **Πού αποθηκεύονται οι εντολές και τα δεδομένα;**

✅ Και οι **εντολές** και τα **δεδομένα** αποθηκεύονται **στη μνήμη του υπολογιστή**.
➡️ Αυτό είναι το βασικό χαρακτηριστικό ενός **υπολογιστή αποθηκευμένου προγράμματος (stored program computer)**.

❓ **Τι σημαίνει ότι τα προγράμματα μπορούν να επενεργούν σε άλλα προγράμματα;**

✅ Σημαίνει ότι ένα πρόγραμμα μπορεί να:

- **Παράγει**, **τροποποιεί** ή **χειρίζεται** άλλο πρόγραμμα.

➡️ Παραδείγματα τέτοιων προγραμμάτων:

- **Μεταγλωττιστές** (compilers)
- **Προγράμματα σύνδεσης** (linkers)

❓ **Ποιο είναι το πλεονέκτημα της χρήσης κοινής δυαδικής μορφής για εντολές και δεδομένα;**

✅ Χάρη στη **συμβατότητα** της δυαδικής αναπαράστασης:

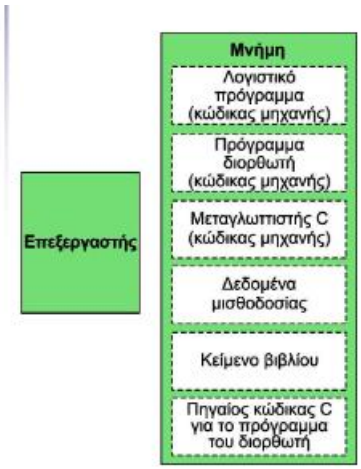
- Ένα **μεταγλωττισμένο πρόγραμμα** μπορεί να χρησιμοποιείται **σε διαφορετικούς υπολογιστές**, **εφόσον αυτοί μοιράζονται την ίδια αρχιτεκτονική**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Τι βοηθάει στην κοινή χρήση προγραμμάτων μεταξύ διαφορετικών υπολογιστών;**

✅ Η ύπαρξη **τυποποιημένων αρχιτεκτονικών συνόλου εντολών (ISA – Instruction Set Architecture)**

➡️ Αυτές εξασφαλίζουν ότι διαφορετικά συστήματα μπορούν να «καταλαβαίνουν» και να εκτελούν τον ίδιο δυαδικό κώδικα.



🔹 **Λογικές Λειτουργίες (Πράξεις) – Bitwise χειρισμοί**

❓ **Τι είναι οι λογικές λειτουργίες στον υπολογιστή;**

✅ Είναι **πράξεις που εκτελούνται bit προς bit**, δηλαδή **σε κάθε bit ξεχωριστά** μεταξύ δύο (ή ενός) αριθμών.

❓ **Ποιες βασικές λογικές λειτουργίες υποστηρίζονται στο RISC-V;**

RISC-V	Java C		Λειτουργία / Πράξη
slli	<<	<<	**Αριστερή ολίσθηση** (shift left)
srli	>>	>>	**Δεξιά ολίσθηση** (shift right)
and, andi	&&	&	**Λογικό AND** (bit προς bit)
or, ori	`	`	
xor, xori	^^	^	**Λογικό XOR** (bit προς bit)
—	~	~	**Λογικό NOT** (αντιστροφή bit)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Ποια είναι η διαφορά των εντολών and, or, xor με τις andi, ori, xori;**

✅ Οι εκδόσεις που **τελειώνουν σε -i** χρησιμοποιούν **άμεσο τελεστέο** (immediate), δηλαδή ένα **σταθερό αριθμό** αντί για δεύτερο καταχωρητή.

❓ **Πού είναι χρήσιμες οι λογικές λειτουργίες;**

✅ Είναι χρήσιμες όταν θέλουμε να:

- **Εξάγουμε** συγκεκριμένα bits από μια λέξη (π.χ. με AND και μάσκα)
- **Εισάγουμε / τροποποιήσουμε** bits σε μια λέξη
- **Μετακινούμε bits** (με ολίσθηση – shift)
- **Κάνουμε κρυπτογραφία, επεξεργασία εικόνας, χειρισμό flags**, κ.ά.

🔹 **Ολίσθηση & Λογικές Λειτουργίες (AND / OR)**

❓ **Τι κάνει μια εντολή ολίσθησης (shift) στον RISC-V;**

✅ Μετακινεί τα bits ενός αριθμού προς τα **αριστερά ή δεξιά**, προσθέτοντας **μηδενικά** στις κενές θέσεις.

❓ **Τι σημαίνει το πεδίο immed σε εντολές ολίσθησης;**

✅ Δείχνει **κατά πόσες θέσεις** θα γίνει η ολίσθηση. (π.χ. slli x5, x6, 3 → ολίσθηση κατά 3 bits)

❓ **Τι είναι η αριστερή λογική ολίσθηση (Shift Left Logical);**

✅ Εντολή: slli

- **Μετακινεί τα bits προς τα αριστερά**
- **Συμπληρώνει με μηδενικά** στα δεξιά
- Ισοδυναμεί με **πολλαπλασιασμό με 2^i**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Τι είναι η δεξιά λογική ολίσθηση (Shift Right Logical);**

✅ Εντολή: srli

- **Μετακινεί τα bits προς τα δεξιά**
- **Συμπληρώνει με μηδενικά** στα αριστερά
- Ισοδυναμεί με **διαίρεση με 2^i** (μόνο για **μη προσημασμένους** αριθμούς)

❓ **Πώς κωδικοποιούνται οι εντολές ολίσθησης σε RISC-V;**

✅ Μορφή:

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

❓ **Τι κάνει η λογική λειτουργία AND στον RISC-V;**

✅ Εντολή: and x9, x10, x11

➡️ Εκτελεί **bit προς bit λογικό AND**: μόνο όταν **και τα δύο bits είναι 1**, το αποτέλεσμα είναι 1.

❓ **Πού είναι χρήσιμη η πράξη AND;**

✅ Για **μάσκες bit**:

- Κρατάμε **μόνο συγκεκριμένα bits**
- Όλα τα υπόλοιπα γίνονται **0**

Η **μάσκα bit (bitmask)** είναι ένας **δυαδικός αριθμός** που χρησιμοποιείται για να **ενεργοποιήσεις, απενεργοποιήσεις ή ελέγξεις συγκεκριμένα bit** μέσα σε έναν άλλο δυαδικό αριθμό, μέσω λογικών πράξεων (bitwise operations).

🔧 **Πού χρησιμοποιείται η bitmask;**

Μια bitmask χρησιμοποιείται κυρίως:

- Για να **ελέγξεις αν συγκεκριμένα bit είναι ενεργοποιημένα** (δηλ. είναι 1)
- Για να **θέσεις συγκεκριμένα bit σε 1**
- Για να **καθαρίσεις (μηδενίσεις) συγκεκριμένα bit**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Για να **αντιστρέψεις** συγκεκριμένα bit

🧠 **Βασικές λογικές πράξεις με bitmask**

Πράξη	Συνάρτηση	Παράδειγμα
AND (&)	Έλεγχος bit	val & mask: κρατά μόνο τα bit που είναι 1 και στα δύο
OR (`	`)	Ενεργοποίηση bit
XOR (^)	Αντιστροφή bit	val ^ mask: αλλάζει όσα bit είναι 1 στη μάσκα
NOT + AND	Καθαρισμός bit	val & ~mask: θέτει σε 0 τα bit που είναι 1 στη μάσκα

🎯 **Τι σημαίνει αυτό σε συστήματα/υλικό;**

Σε αρχιτεκτονικές όπως η **RISC-V**, οι bitmasks χρησιμοποιούνται για:

- Έλεγχο καταχωρητών κατάστασης
- Διαχείριση flags (π.χ. zero flag, interrupt enable)
- Επιλογή πεδίων εντολών (π.χ. opcode, funct3, κ.λπ.)

❓ **Παράδειγμα πράξης AND:**

Καταχωρητής Δυαδικά bits

x10	00000000 00000000 00001101 11000000
x11 (μάσκα)	00000000 00000000 00111100 00000000
x9 (AND)	00000000 00000000 00001100 00000000

❓ **Τι κάνει η λογική λειτουργία OR στον RISC-V;**

✅ Εντολή: or x9, x10, x11
➡️ Εκτελεί **bit προς bit λογικό OR**: αν **τουλάχιστον ένα bit είναι 1**, το αποτέλεσμα είναι 1.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Πού είναι χρήσιμη η πράξη OR;**

✅ Για **ενεργοποίηση (συμπερίληψη)** συγκεκριμένων bits:

- Ορισμένα bits γίνονται **σίγουρα 1**
- Τα υπόλοιπα **παραμένουν ως έχουν**

❓ **Παράδειγμα πράξης OR:**

Καταχωρητής	**Δυαδικά bits**
x10	00000000 00000000 00001101 11000000
x11 (μάσκα)	00000000 00000000 00111100 00000000
x9 (OR)	00000000 00000000 00111101 11000000

🔹 **Λογική Λειτουργία XOR**

❓ **Τι είναι η λογική λειτουργία XOR (Exclusive OR);**

✅ Η XOR είναι μια **bit προς bit** λογική πράξη που λειτουργεί ως **διαφορά**:

- Επιστρέφει **1 μόνο όταν τα δύο bits είναι διαφορετικά**
- Αν τα bits είναι ίδια (0-0 ή 1-1), το αποτέλεσμα είναι **0**

❓ **Ποια είναι η εντολή XOR στον RISC-V;**

✅ Εντολή:

xor x9, x10, x12

➡️ Εκτελεί XOR ανάμεσα στο περιεχόμενο των καταχωρητών **x10** και **x12**, και αποθηκεύει το αποτέλεσμα στον **x9**.

❓ **Τι μπορούμε να πετύχουμε με XOR;**

✅ Μπορούμε να **αντιστρέψουμε επιλεγμένα bits** — σαν **επιλεκτική NOT**:

- Όπου το ένα από τα δύο έχει 1 → γίνεται αντιστροφή

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Όπου είναι 0 → το αρχικό bit παραμένει

❓ **Παράδειγμα λειτουργίας XOR:**

Καταχωρητής Δυαδικά bits

x10	00000000 00000000 00001101 11000000
x12 (μάσκα)	11111111 11111111 11111111 11111111
x9 (XOR)	11111111 11111111 11110010 00111111

➡️ Παρατηρούμε ότι το αποτέλεσμα στον x9 είναι το **αντίθετο του x10**, άρα η μάσκα όλων των 1 **προκαλεί πλήρη αντιστροφή** (σαν NOT).

🔹 **Λειτουργίες που εκτελούνται υπό συνθήκη (διακλαδώσεις)**

❓ **Τι σημαίνει ότι μια λειτουργία εκτελείται "υπό συνθήκη";**

✅ Εκτελείται **μόνο αν ισχύει μια συγκεκριμένη συνθήκη** (π.χ. ισότητα ή ανισότητα μεταξύ δύο καταχωρητών).
➡️ Αν η συνθήκη **είναι αληθής**, τότε γίνεται **διακλάδωση** (δηλαδή άλμα) σε κάποια **συγκεκριμένη εντολή** με **ετικέτα (label)**.
➡️ Αν **δεν είναι αληθής**, συνεχίζεται η **κανονική, σειριακή εκτέλεση** των εντολών.

❓ **Ποια είναι η εντολή beq στον RISC-V;**

✅

beq rs1, rs2, L1

➡️ Αν οι τιμές των καταχωρητών **rs1** και **rs2** είναι **ίσες** (==), τότε η εκτέλεση **πηγαίνει στην εντολή με την ετικέτα L1**.

❓ **Ποια είναι η εντολή bne στον RISC-V;**

✅

bne rs1, rs2, L1

➡️ Αν οι τιμές των καταχωρητών **rs1** και **rs2** είναι **διαφορετικές** (!=), τότε γίνεται **διακλάδωση στην ετικέτα L1**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Τι είναι μια ετικέτα (label) στον κώδικα;**

✅ Είναι ένα **σημείο στον κώδικα** που μπορεί να ονομαστεί και να χρησιμοποιηθεί ως **στόχος άλματος ή διακλάδωσης**.

📌 Παράδειγμα:

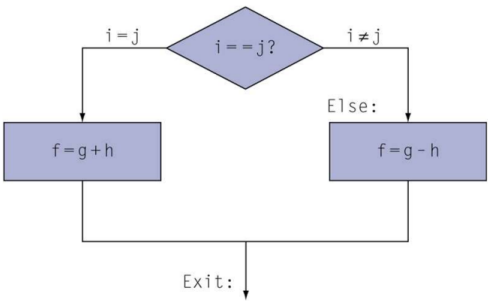
```
beq x5, x6, equal
...
equal:
  add x1, x1, x2
```

🔹 **Μεταγλώττιση εντολών if σε RISC-V**

❓ **Πώς μετατρέπεται μια if-else εντολή από C σε RISC-V;**

✅ Παράδειγμα σε C:

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



ΕΝΗ ΖΑΦΕΙΡΗ

❓ **Ποιοι καταχωρητές αντιστοιχούν στις μεταβλητές;**

✅

- f → x19
- g → x20
- h → x21
- i → x22
- j → x23

❓ **Ποιος είναι ο αντίστοιχος μεταγλωττισμένος κώδικας σε RISC-V;**

✅

bne x22, x23, Else // αν i ≠ j, πήγαινε στο Else

add x19, x20, x21 // f = g + h (αν i == j)

beq x0, x0, Exit // άλμα χωρίς συνθήκη προς Exit

Else: sub x19, x20, x21 // f = g - h

Exit: ... // συνέχεια του προγράμματος

❓ **Τι ρόλο παίζουν οι ετικέτες Else και Exit;**

✅

- **Else:** δείχνει το σημείο όπου θα εκτελεστεί η εναλλακτική εντολή (f = g - h)
- **Exit:** δείχνει το τέλος του μπλοκ if-else, όπου συνεχίζεται το υπόλοιπο πρόγραμμα

❓ **Γιατί χρησιμοποιείται beq x0, x0, Exit; για άλμα χωρίς συνθήκη;**

✅ Επειδή το **x0 περιέχει πάντα την τιμή 0**, η συνθήκη x0 == x0 είναι **πάντα αληθής**, άρα το άλμα εκτελείται **πάντα**.

❓ **Ποιος υπολογίζει τις διευθύνσεις των ετικετών (Else, Exit);**

✅ Ο **συμβολομεταφραστής** (assembler) είναι υπεύθυνος για να **αντικαταστήσει τις ετικέτες** με τις **αντίστοιχες διευθύνσεις** κατά τη μεταγλώττιση.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔹 **Άλλες Λειτουργίες που Εκτελούνται Υπό Συνθήκη**

❓ **Τι κάνει η εντολή blt στον RISC-V;**

✅

blt rs1, rs2, L1

➡️ Αν **rs1 < rs2**, γίνεται **διακλάδωση** στην εντολή με ετικέτα **L1**

❓ **Τι κάνει η εντολή bge στον RISC-V;**

✅

bge rs1, rs2, L1

➡️ Αν **rs1 ≥ rs2**, τότε γίνεται **διακλάδωση** στην ετικέτα **L1**

❓ **Παράδειγμα σε C: if (a > b) a += 1; — Πώς μεταφράζεται σε RISC-V;**

✅ Αντιστοίχιση μεταβλητών:

- a στον **x22**
- b στον **x23**

✅ Κώδικας RISC-V:

bge x23, x22, Exit // Αν b ≥ a, παράλειψε την εντολή a += 1

addi x22, x22, 1 // Διαφορετικά, πρόσθεσε 1 στην a

Exit:

➡️ Η bge x23, x22, Exit είναι **ισοδύναμη με "if (a ≤ b)" → πήγαινε στο Exit**, δηλαδή **δεν εκτελείς την αύξηση** αν η συνθήκη δεν ισχύει.

🔹 **Μεταγλώττιση εντολών με βρόχους**

1.while

Το παράδειγμα που ακολουθεί είναι λίγο δύσκολο, και είναι αυτό που αναγράφεται στις διαφάνειες του μαθήματος. Προτείνω να τσεκάρετε το παραδειγμα πιο κάτω για αρχη :)))

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Παράδειγμα C: Τι κάνει ο παρακάτω βρόχος;**

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

✅ Όσο το **i-οστό στοιχείο** του πίνακα save είναι **ίσο με k**, το i αυξάνεται κατά 1.

❓ **Ποια είναι η αντιστοίχιση των μεταβλητών σε καταχωρητές;**

Μεταβλητή	**Καταχωρητής**
i	x22
k	x24
save	x25 (διεύθυνση βάσης του πίνακα)

❓ **Ποιος είναι ο μεταγλωττισμένος κώδικας RISC-V για το while;**

✅

```
Loop:
slli x10, x22, 2      // x10 = i * 4 (υπολογισμός byte offset)
add x10, x10, x25     // x10 = διεύθυνση save[i]
lw x9, 0(x10)         // x9 = save[i]
bne x9, x24, Exit     // Αν save[i] != k → έξοδος από τον βρόχο
addi x22, x22, 1      // i += 1
beq x0, x0, Loop      // άλμα πίσω στην αρχή του βρόχου
Exit:
```

❓ **Τι κάνει κάθε εντολή βήμα-βήμα;**

✅

- slli x10, x22, 2: πολλαπλασιάζει το i επί 4 (για byte προσπέλαση)
- add x10, x10, x25: υπολογίζει τη **διεύθυνση του save[i]**
- lw x9, 0(x10): φορτώνει το στοιχείο save[i] στη x9

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- bne x9, x24, Exit: αν το στοιχείο **δεν είναι ίσο με k**, βγαίνουμε από τον βρόχο
- addi x22, x22, 1: αυξάνει το i κατά 1
- beq x0, x0, Loop: **χωρίς συνθήκη**, γυρίζει πίσω στην αρχή του βρόχου

Ένα πιο απλό παράδειγμα

❓ **Παράδειγμα C:**

```
while (i < 10)
  i += 1;
```

🔁 **Περιγραφή:**

- Ελέγχει **πρώτα τη συνθήκη**
- Αν είναι αληθής, εκτελεί το σώμα του βρόχου

🛠️ **Αντιστοίχιση:**

- i → x5
- Σταθερά 10 → χρησιμοποιείται άμεσα (immediate)

🔧 **RISC-V κώδικας:**

```
Loop:
  li x6, 10          # x6 = 10
  bge x5, x6, Exit   # αν i >= 10 → Exit
  addi x5, x5, 1     # i += 1
  beq x0, x0, Loop   # άλμα πίσω στην αρχή
Exit:
```

✅ **2. for βρόχος**

❓ **Παράδειγμα C:**

```
for (i = 0; i < 5; i++)
  sum += i;
```

🔁 **Περιγραφή:**

- Αρχικοποίηση → Έλεγχος → Εκτέλεση → Ενημέρωση

🛠️ **Αντιστοίχιση:**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- i → x5
- sum → x6

🔧 **RISC-V κώδικας:**

```
  li x5, 0          # i = 0
  li x6, 0          # sum = 0
Loop:
  li x7, 5          # όριο = 5
  bge x5, x7, Exit    # αν i >= 5 → Exit
  add x6, x6, x5      # sum += i
  addi x5, x5, 1      # i += 1
  beq x0, x0, Loop    # άλμα πίσω
Exit:
```

✅ **3. do-while βρόχος**

❓ **Παράδειγμα C:**

```
do {
  i += 1;
} while (i < 3);
```

🔁 **Περιγραφή:**

- Εκτελεί **τουλάχιστον μία φορά** το σώμα
- Έπειτα ελέγχει τη συνθήκη

🛠️ **Αντιστοίχιση:**

- i → x5

🔧 **RISC-V κώδικας:**

```
Loop:
  addi x5, x5, 1      # i += 1
  li x6, 3          # όριο = 3
  blt x5, x6, Loop    # αν i < 3 → επανάληψη
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Βασικά Μπλοκ (Basic Blocks)

❓ **Τι είναι ένα βασικό μπλοκ (basic block);**

✅ Είναι μια **ακολουθία εντολών** που εκτελείται **διαδοχικά**, **χωρίς διακοπές**, δηλαδή:

- **Δεν περιέχει διακλαδώσεις, εκτός ίσως από την τελευταία εντολή**
- **Δεν αποτελεί στόχο διακλάδωσης, εκτός πιθανώς από την πρώτη εντολή**

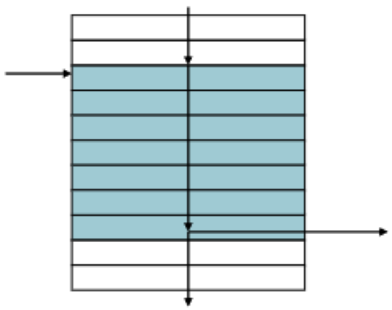
❓ **Ποιος είναι ο ρόλος του μεταγλωττιστή σχετικά με τα βασικά μπλοκ;**

✅ Ο μεταγλωττιστής **αναγνωρίζει** τα βασικά μπλοκ στον κώδικα ώστε να μπορεί να τα **βελτιστοποιήσει** πιο εύκολα (π.χ. να βελτιώσει την απόδοση ή να κάνει αναδιατάξεις εντολών).

❓ **Γιατί είναι σημαντικά τα βασικά μπλοκ για τον επεξεργαστή;**

✅ Οι **προηγμένοι επεξεργαστές** μπορούν να **εκτελούν βασικά μπλοκ γρήγορα και αποδοτικά**, γιατί:

- Δεν χρειάζεται να ελέγχουν για διακλαδώσεις μέσα στο μπλοκ
- Επιτρέπουν **βελτιστοποίηση της ροής εντολών** (π.χ. pipelining, parallelism)



🔹 **Προσημασμένοι και μη προσημασμένοι (απρόσημοι) αριθμοί – Σύγκριση**

❓ **Ποια εντολή χρησιμοποιείται για σύγκριση προσημασμένων (signed) αριθμών στον RISC-V;**

✅

- **blt** → "branch if less than" (αν μικρότερος)
- **bge** → "branch if greater or equal" (αν μεγαλύτερος ή ίσος)

➡️ Χρησιμοποιούνται όταν οι αριθμοί θεωρούνται **με πρόσημο** (signed).

❓ **Ποια εντολή χρησιμοποιείται για σύγκριση μη προσημασμένων (unsigned) αριθμών;**

✅

- **bltu** → "branch if less than (unsigned)"
- **bgeu** → "branch if greater or equal (unsigned)"

➡️ Χρησιμοποιούνται όταν οι αριθμοί θεωρούνται **χωρίς πρόσημο** (unsigned).

❓ **Παράδειγμα με τους καταχωρητές x22 και x23:**

Καταχωρητής	**Δυαδική Τιμή**
x22	1111 1111 ... 1111 → (32 bits = -1 signed, ή 4.294.967.295 unsigned)
x23	0000 0000 ... 0001 → (1 signed και unsigned)

❓ **Ποια είναι η σύγκριση με πρόσημο (signed);**

✅ x22 < x23

➡️ Δηλαδή: **–1 < +1** → ✅ **Αληθές**

❓ **Ποια είναι η σύγκριση χωρίς πρόσημο (unsigned);**

✅ x22 > x23

➡️ Δηλαδή: **4.294.967.295 > 1** → ✅ **Αληθές**

📌 **Συμπέρασμα**:
Η **ίδια δυαδική τιμή** μπορεί να σημαίνει **πολύ διαφορετικό αριθμό** ανάλογα με το αν την ερμηνεύουμε **με ή χωρίς πρόσημο** — και γι' αυτό υπάρχουν **ξεχωριστές εντολές σύγκρισης**!

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔹 §2.8 Κλήσεις Διαδικασιών (Function Calls)

❓ **Ποια είναι τα βασικά βήματα για την κλήση μιας διαδικασίας στον RISC-V;**

✅

1. **Τοποθέτηση παραμέτρων** στους καταχωρητές **x10 έως x17**
2. **Μεταφορά ελέγχου** στη διαδικασία (δηλαδή, άλμα στον κώδικα της διαδικασίας)
3. **Δέσμευση πόρων αποθήκευσης** (π.χ. χώρος στη στοίβα)
4. **Εκτέλεση της διαδικασίας** (λειτουργίες, υπολογισμοί κ.λπ.)
5. **Τοποθέτηση αποτελέσματος** σε καταχωρητή προσβάσιμο από το κύριο πρόγραμμα
6. **Επιστροφή** στο σημείο από όπου κλήθηκε η διαδικασία (χρησιμοποιώντας τη διεύθυνση στον **x1**)

❓ **Ποια εντολή χρησιμοποιείται για κλήση διαδικασίας στον RISC-V;**

✅

jal x1, ProcedureLabel

➡️ **JAL (Jump and Link)**:

- **Αποθηκεύει τη διεύθυνση της επόμενης εντολής στον x1**
- **Μεταβαίνει (κάνει άλμα)** στην εντολή που έχει την ετικέτα ProcedureLabel

---

❓ **Τι συμβαίνει αν χρησιμοποιήσουμε x0 ως καταχωρητή προορισμού στην jal;**

✅

jal x0, Label

➡️ Πρόκειται για **άπλο άλμα χωρίς επιστροφή** — Ισοδύναμο με **goto** στη γλώσσα C — Δεν αποθηκεύει διεύθυνση επιστροφής, γιατί **ο x0 δεν μπορεί να τροποποιηθεί**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Πώς γίνεται η επιστροφή από μια διαδικασία;**

jalr x0, 0(x1)

➡️ **JALR (Jump and Link Register)**:

- Κάνει άλμα στη **διεύθυνση που είναι αποθηκευμένη στον x1**
- Ο x1 είχε πάρει αυτή τη διεύθυνση από την προηγούμενη jal

❓ **Για ποιον άλλο σκοπό μπορεί να χρησιμοποιηθεί η jalr;**

✅ Η jalr μπορεί να χρησιμοποιηθεί και για:

- **Άλματα σε υπολογισμένες διευθύνσεις**, όπως π.χ. σε **εντολές switch/case**

🔹 **Παράδειγμα με μια διαδικασία-φύλλο (leaf procedure)**

❓ **Τι είναι μια διαδικασία-φύλλο (leaf procedure);**

✅ Είναι μια διαδικασία (συνάρτηση) που **δεν καλεί άλλες διαδικασίες**. ➡️ Δεν χρειάζεται να αποθηκεύσει τη διεύθυνση επιστροφής άλλων συναρτήσεων, αλλά **χρησιμοποιεί προσωρινούς καταχωρητές** που **πρέπει να διασωθούν**.

❓ **Παράδειγμα C συνάρτησης:**

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

❓ **Αντιστοίχιση μεταβλητών με καταχωρητές:**

Μεταβλητή	**Καταχωρητής**
g	x10
h	x11

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Μεταβλητή Καταχωρητής

i	x12
j	x13
f	x20
προσωρινά	x5, x6

❓ **Ποιοι καταχωρητές πρέπει να αποθηκευτούν στη στοίβα;**

✅ Οι **x5, x6, x20** είναι προσωρινοί/τοπικοί και **πρέπει να αποθηκευτούν** και να επαναφερθούν από τη **στοίβα**, ώστε να μην αλλοιώσουν τιμές του καλούντος.

❓ **RISC-V κώδικας της leaf διαδικασίας:**

```
leaf_example:
  addi sp, sp, -12     # Δέσμευση 12 bytes στη στοίβα
  sw x5, 8(sp)         # Αποθήκευση x5
  sw x6, 4(sp)         # Αποθήκευση x6
  sw x20, 0(sp)        # Αποθήκευση x20

  add x5, x10, x11     # x5 = g + h
  add x6, x12, x13     # x6 = i + j
  sub x20, x5, x6      # f = x5 - x6
  addi x10, x20, 0     # Αντιγραφή του f στον x10 (καταχωρητής επιστροφής)

  lw x20, 0(sp)        # Επαναφορά x20
  lw x6, 4(sp)         # Επαναφορά x6
  lw x5, 8(sp)         # Επαναφορά x5
  addi sp, sp, 12      # Αποδέσμευση 12 bytes από τη στοίβα

  jalr x0, 0(x1)       # Επιστροφή στον καλούντα
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Σύντομη επεξήγηση βημάτων:**

Βήμα	**Ενέργεια**
addi sp, sp, -12	Δέσμευση χώρου για 3 καταχωρητές (3×4 bytes)
sw εντολές	Αποθήκευση προσωρινών τιμών στη στοίβα
add, sub	Υπολογισμός (g + h) - (i + j)
addi x10, x20, 0	Επιστροφή τιμής μέσω του x10
lw εντολές	Επαναφορά τιμών καταχωρητών από τη στοίβα
addi sp, sp, 12	Αποδέσμευση της στοίβας
jalr x0, 0(x1)	Επιστροφή στη διεύθυνση που είχε ο x1

🔹 **Τοπικά Δεδομένα στη Στοίβα**

❓ **Τι είναι τα τοπικά δεδομένα σε μια διαδικασία;**

✅ Είναι **μεταβλητές** που:

- Ορίζονται **μέσα στη διαδικασία** (και μόνο εκεί ισχύουν)
- Δεν διατηρούνται **μετά το τέλος** της εκτέλεσης της διαδικασίας
- Δεν είναι κοινές με άλλες συναρτήσεις

❓ **Πού αποθηκεύονται τα τοπικά δεδομένα;**

✅ Αποθηκεύονται στη **στοίβα** (stack) του προγράμματος, μαζί με:

- Τιμές προσωρινών καταχωρητών
- Παλαιές τιμές καταχωρητών (π.χ. x1, x5, x20...)
- Ενδεχομένως **δομές ή πίνακες μικρού μεγέθους**

❓ **Γιατί χρησιμοποιείται η στοίβα για τα τοπικά δεδομένα;**

✅ Επειδή:

- Είναι **εύκολη στη διαχείριση με LIFO λογική**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Κάθε κλήση συνάρτησης έχει **δικό της χώρο** (stack frame)
- Η χρήση της στοίβας επιτρέπει **πολλαπλές και αναδρομικές** κλήσεις χωρίς σύγκρουση δεδομένων

❓ **Πώς γίνεται η αποθήκευση και ανάκτηση τοπικών δεδομένων;**

✅ Με εντολές:

- **addi sp, sp, -N** → δέσμευση χώρου στη στοίβα
- **sw xX, offset(sp)** → αποθήκευση δεδομένου στη στοίβα
- **lw xX, offset(sp)** → ανάκτηση δεδομένου από τη στοίβα
- **addi sp, sp, +N** → αποδέσμευση χώρου από τη στοίβα

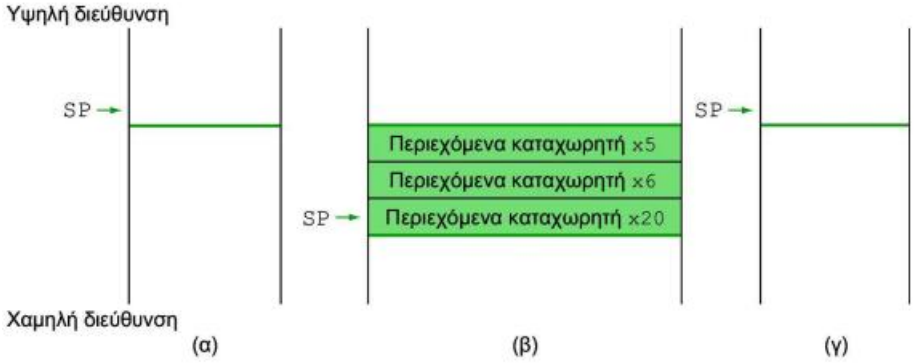
❓ **Παράδειγμα χρήσης τοπικών δεδομένων:**

```
int add_and_double(int x, int y) {
   int sum;
   int result;
   sum = x + y;
   result = sum * 2;
   return result;
}
```

➡️ Σε RISC-V, οι sum και result πιθανόν να αποθηκευτούν στη **στοίβα**, αν δεν επαρκούν οι προσωρινοί καταχωρητές (π.χ. x5, x6).

📌 **Συνοπτικά**:
Η στοίβα **φιλοξενεί** τοπικά δεδομένα για κάθε συνάρτηση, βοηθώντας στη **σωστή απομόνωση, προσωρινή αποθήκευση και ανάκτηση** των τιμών.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



🔹 Χρήση Καταχωρητών – Διαδικασίες Φύλλου και Μη Φύλλου

❓ Ποιοι καταχωρητές στον RISC-V είναι προσωρινοί (temporaries);

- x5 έως x7
- x28 έως x31

📌 Δεν διατηρούνται από την **καλούμενη** διαδικασία. Αν χρησιμοποιηθούν, **δεν χρειάζεται** να αποθηκευτούν από τη διαδικασία.

❓ Ποιοι καταχωρητές είναι αποθηκευμένοι (saved registers);

- x8, x9
- x18 έως x27

📌 Αν μια διαδικασία τους χρησιμοποιήσει, **οφείλει να τους αποθηκεύσει και να τους επαναφέρει** πριν επιστρέψει.

🔹 Διαδικασίες που δεν είναι φύλλο (non-leaf)

❓ Τι είναι μια non-leaf procedure;

✅ Μια διαδικασία που **καλεί άλλες διαδικασίες**.

❓ Τι επιπλέον αποθηκεύει μια non-leaf procedure στη στοίβα;

- Τη **διεύθυνση επιστροφής** (συνήθως στον x1)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Τις **τιμές των ορισμάτων** ή **προσωρινών καταχωρητών** που θα χρειαστεί **μετά την κλήση**

✅ **Παράδειγμα: Υπολογισμός παραγοντικού (factorial)**

❓ **Κώδικας C:**

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

➡️ Αναδρομική συνάρτηση — **δεν είναι φύλλο**.

❓ **Καταχωρητές:**

- n → x10
- Τιμή επιστροφής → x10
- Πρόχειροι καταχωρητές → x5, x6
- x1 → διεύθυνση επιστροφής

❓ **RISC-V Κώδικας:**

```
fact:
  addi sp, sp, -8      # Δέσμευση χώρου στη στοίβα
  sw x1, 4(sp)         # Αποθήκευση διεύθυνσης επιστροφής
  sw x10, 0(sp)        # Αποθήκευση του n

  addi x5, x10, -1     # x5 = n - 1
  bgt x5, x0, L1       # αν n > 0 → πήγαινε στο L1

  addi x10, x0, 1      # επιστροφή 1 αν n < 1
  addi sp, sp, 8       # αποδέσμευση στοίβας
  jalr x0, 0(x1)       # επιστροφή
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

```
L1:
    addi x10, x10, -1     # n = n - 1
    jal x1, fact          # αναδρομική κλήση fact(n-1)
    addi x6, x10, 0       # αντιγραφή αποτελέσματος σε x6

    lw x10, 0(sp)         # επαναφορά n του καλούντα
    lw x1, 4(sp)          # επαναφορά διεύθυνσης επιστροφής
    addi sp, sp, 8        # αποδέσμευση στοίβας

    mul x10, x10, x6      # n * fact(n - 1)
    jalr x0, 0(x1)        # επιστροφή αποτελέσματος
```

❓ **Τι κάνει ο παραπάνω κώδικας βήμα-βήμα;**

Βήμα	**Περιγραφή**
Αποθήκευση x1 & x10	Διατήρηση διεύθυνσης επιστροφής & παραμέτρου n
Έλεγχος n < 1	Αν ναι, επιστροφή 1
Αναδρομική κλήση fact(n - 1)	Αν όχι, καλείται ξανά
Επαναφορά τιμών	Από τη στοίβα
Υπολογισμός n * fact(n-1)	Πραγματοποιείται τελικός πολλαπλασιασμός
Επιστροφή στο κύριο πρόγραμμα	Μέσω jalr

🔹 **Διάταξη της Μνήμης (Memory Layout)**

❓ **Ποια είναι τα βασικά τμήματα της μνήμης ενός προγράμματος;**

✅ Η μνήμη ενός προγράμματος χωρίζεται σε τέσσερις κύριες περιοχές:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

1. **Κείμενο** **(Text)**
   ➤ Περιέχει τον **εκτελέσιμο κώδικα του προγράμματος**
2. **Στατικά δεδομένα (Static Data)**
   ➤ Περιέχει **καθολικές (global)** και **στατικές μεταβλητές**
   ➤ Περιλαμβάνει πίνακες και **σταθερές συμβολοσειρές** (π.χ. "Hello" στη C)
3. **Δυναμικά δεδομένα (Heap)**
   ➤ Χρησιμοποιείται για **δυναμική δέσμευση μνήμης**
   ➤ Παράδειγμα: malloc() στη **C**, new στη **Java**
4. **Στοίβα** **(Stack)**
   ➤ Χρησιμοποιείται για **τοπικές μεταβλητές**, **διευθύνσεις επιστροφής** και **ενεργές κλήσεις διαδικασιών**

❓ **Ποιος καταχωρητής χρησιμοποιείται για την προσπέλαση στατικών δεδομένων;**

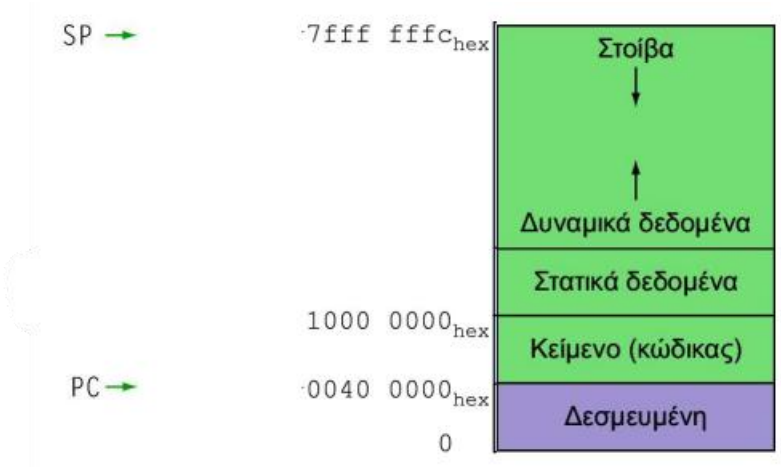
✅ Ο **x3** είναι ο **καθολικός δείκτης (global pointer, gp)**

📌 Ο x3 δείχνει **μέσα στην περιοχή των στατικών δεδομένων**
➤ Από εκεί μπορεί να προσπελαστεί **οποιαδήποτε άλλη διεύθυνση στατικής μνήμης**, είτε με **θετική** είτε με **αρνητική σχετική διεύθυνση** (offset).

❓ **Ποια λειτουργία έχουν οι περιοχές heap και stack στη μνήμη;**

Τμήμα μνήμης	**Χρήση**	**Παράδειγμα κλήσης**
Heap	Για **δυναμικά δεδομένα** που ζουν όσο χρειάζεται	malloc, new
Stack	Για **τοπικές μεταβλητές**, **κλήσεις συναρτήσεων**, **επιστροφές**	Αυτόματη

ΕΑΕΝΗ ΖΑΦΕΙΡΗ



Και για να ξέρουμε για ποια μνήμη μιλάμε....

📍 **Τι δείχνει ο sp (stack pointer);**

Ο **sp δείχνει στην τελευταία χρησιμοποιημένη θέση** (ή την επόμενη διαθέσιμη θέση, ανάλογα με την αρχιτεκτονική).

✔️ **Παράδειγμα στη RISC-V (όπως και στη MIPS):**

- Η στοίβα **μεγαλώνει προς τις μικρότερες διευθύνσεις**.
- Άρα, όταν "σπρώχνουμε" δεδομένα στη στοίβα (π.χ. με addi sp, sp, -16), ο δείκτης μετακινείται **χαμηλότερα**.
- Όταν τα "βγάζουμε", ο sp ανεβαίνει (addi sp, sp, 16).

Η **διάταξη της μνήμης (memory layout)** παραπάνω αφορά τη **μνήμη RAM (Κύρια Μνήμη)** του υπολογιστή ή του μικροεπεξεργαστή **κατά την εκτέλεση** ενός προγράμματος.

🔹 **Ναι, πρόκειται για την Κύρια Μνήμη (RAM)**

- Όταν λέμε *text*, *data*, *heap*, *stack*, μιλάμε για περιοχές μέσα στη **RAM**.
- Όλες αυτές οι περιοχές δημιουργούνται **όταν το πρόγραμμα φορτώνεται από το λειτουργικό σύστημα** (ή στον μικροελεγκτή, από το bootloader).

🔍 **Πού ακριβώς βρίσκεται αυτή η μνήμη;**

Εξαρτάται από το **σύστημα** (π.χ. PC, embedded RISC-V, κ.λπ.), αλλά σε γενικές γραμμές:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Σε συστήματα όπως PC:

- Η RAM είναι φυσική μνήμη που τοποθετείται στη μητρική πλακέτα.
- Το λειτουργικό σύστημα τη διαχειρίζεται και δημιουργεί για κάθε πρόγραμμα μια **εικονική μνήμη** (virtual memory layout).
- Οι περιοχές *stack*, *heap*, *text* κ.λπ. είναι **μέρη του χώρου διευθύνσεων του προγράμματος**, που αντιστοιχούν τελικά σε περιοχές της RAM.

Σε μικροελεγκτές (π.χ. RISC-V χωρίς λειτουργικό):

- Η μνήμη είναι απλή RAM που βρίσκεται χαρτογραφημένη σε συγκεκριμένες διευθύνσεις (π.χ. από 0x80000000 και πάνω).
- Η διάταξη μνήμης ορίζεται από το linker script και τις συμβάσεις της πλατφόρμας.

🔹 **Τοπικά Δεδομένα στη Στοίβα & Πλαίσιο Διαδικασίας**

❓ **Τι είναι τα τοπικά δεδομένα που αποθηκεύονται στη στοίβα;**

✅ Είναι δεδομένα που:

- **Δημιουργούνται μέσα στη διαδικασία**
- Δεν είναι ορατά εκτός της διαδικασίας
- Συνήθως είναι **αυτόματες μεταβλητές** στη **C**, δηλαδή μεταβλητές που δηλώνονται **τοπικά μέσα σε μια συνάρτηση**

📌 Αυτά τα δεδομένα καταχωρούνται **στη στοίβα από την καλούμενη διαδικασία** (δηλαδή την ίδια τη συνάρτηση που τα χρησιμοποιεί).

❓ **Τι είναι το πλαίσιο διαδικασίας (activation record ή frame);**

✅ Το **πλαίσιο διαδικασίας** είναι:

- Η **περιοχή της στοίβας** που δεσμεύεται **κάθε φορά που καλείται μια διαδικασία**
- Περιλαμβάνει όλα όσα χρειάζεται η διαδικασία για να εκτελεστεί σωστά, όπως:
    - **Τοπικά δεδομένα**
    - **Αποθηκευμένους καταχωρητές** (π.χ. x1 για διεύθυνση επιστροφής, x20, x5 κ.λπ.)

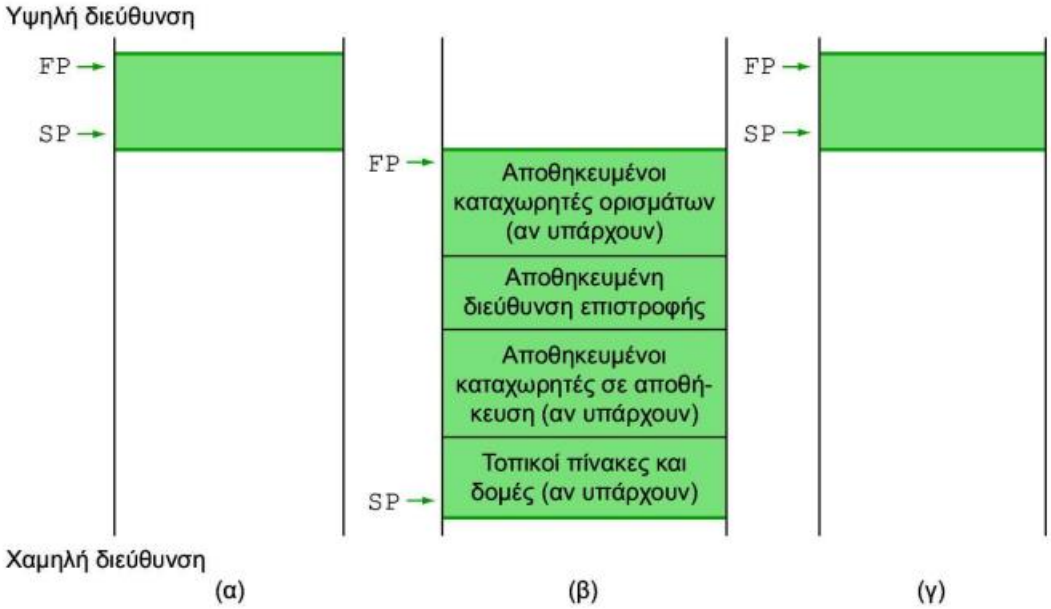
ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Παραμέτρους** (αν χρειάζεται)
- Πιθανόν **δείκτη στο προηγούμενο πλαίσιο** (frame pointer)

❓ **Ποιοι μεταγλωττιστές χρησιμοποιούν πλαίσια διαδικασίας;**

✅ Πολλοί **σύγχρονοι μεταγλωττιστές** χρησιμοποιούν τη δομή του **activation record** για:

- Να **οργανώνουν καλύτερα τη στοίβα**
- Να διαχειρίζονται **με συνέπεια τις κλήσεις** και την επιστροφή από διαδικασίες
- Να υποστηρίζουν **αναδρομή και ένθετες κλήσεις** με ασφάλεια



ΕΛΕΥΘΗ

🔹 Δεδομένα Χαρακτήρων – Byte/Λέξεις – Αντιγραφή Συμβολοσειράς

❓ **Ποια σύνολα χαρακτήρων χρησιμοποιούνται στους υπολογιστές;**

✅ Υπάρχουν διαφορετικές **κωδικοποιήσεις χαρακτήρων**:

Κωδικοποίηση	Πλήθος χαρακτήρων	Περιγραφή
ASCII	128	95 γραφικοί + 33 χαρακτήρες ελέγχου

Κωδικοποίηση	Πλήθος χαρακτήρων	Περιγραφή
Latin-1	256	ASCII + 96 πρόσθετοι γραφικοί
Unicode	έως 2^{32} χαρακτήρες	Περιλαμβάνει **σχεδόν όλα τα ανθρώπινα αλφάβητα** και **σύμβολα**

❓ **Ποιες μορφές Unicode χρησιμοποιούνται;**

✅

- **UTF-8**, **UTF-16**: Κωδικοποιήσεις **μεταβλητού μήκους**, όπου κάθε χαρακτήρας μπορεί να έχει **1–4 bytes**

❓ **Ποιες εντολές RISC-V χρησιμοποιούνται για φόρτωση χαρακτήρων ή λέξεων;**

Εντολή	Λειτουργία	Επέκταση
lb	Φόρτωση **byte** με επέκταση **προσήμου**	
lh	Φόρτωση **ημιλέξης** (16 bit) με πρόσημο	
lw	Φόρτωση **λέξης** (32 bit) με πρόσημο	
lbu	Φόρτωση byte **χωρίς πρόσημο**	
lhu	Φόρτωση ημιλέξης χωρίς πρόσημο	
lwu	Φόρτωση λέξης χωρίς πρόσημο (για 64bit CPUs)	

❓ **Ποιες εντολές RISC-V χρησιμοποιούνται για αποθήκευση;**

Εντολή	Αποθήκευση τύπου
sb	Αποθήκευση **byte**
sh	Αποθήκευση **ημιλέξης (16-bit)**
sw	Αποθήκευση **λέξης (32-bit)**

📌 Αυτές αποθηκεύουν **μόνο τα δεξιά (λιγότερο σημαντικά) bits** του καταχωρητή.

✅ **Παράδειγμα: Αντιγραφή συμβολοσειράς (C → RISC-V)**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Κώδικας C:**

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

➡️ Η συμβολοσειρά **τερματίζεται με null χαρακτήρα** ('\0' δηλαδή byte 0)

❓ **Αντιστοίχιση:**

- x → βάση στον x10
- y → βάση επίσης στον x10 (μάλλον πρόκειται για λάθος: θα έπρεπε να είναι x11)
- i → προσωρινά στον x19
- x5, x6, x7 → ενδιάμεσοι καταχωρητές για διευθύνσεις και χαρακτήρες

❓ **Κώδικας RISC-V:**

```
strcpy:
  addi sp, sp, -4      # Χώρος για 1 λέξη στη στοίβα
  sw x19, 0(sp)        # Αποθήκευση του x19
  add x19, x0, x0      # i = 0

L1:
  add x5, x19, x10     # x5 = y + i
  lbu x6, 0(x5)        # x6 = y[i]
  add x7, x19, x10     # x7 = x + i
  sb x6, 0(x7)         # x[i] = y[i]
  beq x6, x0, L2       # Αν y[i] == '\0' → τέλος
  addi x19, x19, 1     # i++
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

```
    jal x0, L1          # Επανάληψη

L2:
    lw x19, 0(sp)       # Επαναφορά του x19
    addi sp, sp, 4      # Αποδέσμευση στοίβας
    jalr x0, 0(x1)      # Επιστροφή
```

📌 **Σημείωση**: Στην παραπάνω μεταγλώττιση, πιθανόν **η ίδια διεύθυνση (x10) χρησιμοποιείται για x και y**. Στην πράξη, θα έπρεπε x10 = x[], x11 = y[].

🔹 §2.10 — Σταθερές 32-bit & Διευθυνσιοδότηση με μεγάλους άμεσους τελεστέους

❓ **Τι συμβαίνει όταν μια σταθερά έχει μικρή τιμή;**

✅ Αν η σταθερά **χωράει σε 12 bits**, τότε μπορεί να χρησιμοποιηθεί **ως άμεσος τελεστέος** απευθείας σε εντολές όπως addi, lw, sw κ.λπ.

✅ Πρόβλημα: Φόρτωση μεγάλης σταθεράς 32-bit

Στη RISC-V, **οι περισσότερες εντολές** (όπως addi, lw, sw) μπορούν να δουλέψουν μόνο με **άμεσες τιμές 12 bit**. Δηλαδή, μπορείς να βάλεις τιμές από περίπου **–2048 έως +2047** απευθείας.

📌 Αλλά τι γίνεται αν θέλεις να χρησιμοποιήσεις μια **μεγάλη τιμή**, π.χ. **0x003D500** (= **251.392 δεκαδικό**);

Δεν χωράει σε 12 bit → πρέπει να τη "σπάσεις" σε **δύο μέρη**.

✅ Λύση: Χρήση lui και addi

Η RISC-V προσφέρει την εντολή:

🔹 **lui rd, constant**

- **L**oad **U**pper **I**mmediate
- Βάζει μια σταθερά **20 bits** στις **ψηλές θέσεις** (bit 31 έως 12) του καταχωρητή
- Τα **χαμηλά 12 bits (bit 11 έως 0)** γίνονται **0**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Παράδειγμα:

lui x19, 976

✳️ Τι κάνει αυτό;

- Παίρνει το **976** (που είναι 0x003D σε hex)
- Το μετατοπίζει αριστερά κατά 12 bit (δηλ. επί 4096)
- Αποτέλεσμα:
  x19 = 0x003D000 (δηλαδή 976 × 4096 = 250.880)

✅ **Συμπλήρωμα: addi για τα "χαμηλά bits"**

Η lui έβαλε τα **ψηλά bits**.

Για να προσθέσεις τα **χαμηλά bits**, χρησιμοποιείς:

addi x19, x19, 1280 # δηλαδή προσθέτει 0x500

➕ Τώρα έχεις:

x19 = 0x003D000 (από πριν)

+ 0x500

= 0x003D500

📌 **Τελικό αποτέλεσμα: έχεις "φορτώσει" ολόκληρη 32-bit σταθερά στο x19!**

Όλα μαζί:

lui x19, 976 # x19 = 0x003D000

addi x19, x19, 1280 # x19 = x19 + 0x500 = 0x003D500

✅ **Γιατί γίνεται έτσι;**

- **Οικονομία χώρου**: Οι απλές εντολές της RISC-V είναι **32-bit** και θέλουν **μικρούς άμεσους τελεστέους**
- Για **μεγάλες τιμές**, το κόλπο είναι να τις "χτίζεις" με lui + addi
- Είναι **γρήγορο** και **απλό**, γιατί αποφεύγει αναφορές στη μνήμη

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔹 **§2.10 — Διευθυνσιοδότηση Διακλαδώσεων & Άλματος**

❓ **Τι σημαίνει "διευθυνσιοδότηση διακλάδωσης";**

✅ Είναι η διαδικασία με την οποία η εντολή **διαλέγει πού να πάει** (ποια εντολή να εκτελέσει στη συνέχεια), **αν ικανοποιείται μια συνθήκη** (όπως beq, bne, blt, κ.λπ.).

❓ **Τι περιλαμβάνει μια εντολή διακλάδωσης (branch);**

✅ Κάθε τέτοια εντολή περιλαμβάνει:

- Τον **opcode** (τον τύπο της εντολής)
- Δύο **καταχωρητές** (rs1, rs2)
- Έναν **άμεσο τελεστέο** που δηλώνει **πού να πάει** (τη διεύθυνση προορισμού)

❓ **Πώς υπολογίζεται η διεύθυνση προορισμού σε διακλαδώσεις;**

✅ Η διακλάδωση είναι **σχετική με τον μετρητή προγράμματος (PC)**:

Διεύθυνση προορισμού = PC + (άμεσο × 4)

➕ Άρα ο υπολογισμός **δεν βασίζεται σε απόλυτη διεύθυνση**, αλλά **σε απόσταση από την τρέχουσα εντολή** (προς τα μπροστά ή πίσω).

❓ **Ποια μορφή έχει η εντολή διακλάδωσης;**

✅ Είναι της **μορφής τύπου SB** (SB-type format), και τα bits της χωρίζονται ως εξής:

imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode

📌 Ο άμεσος τελεστέος σχηματίζεται από διάσπαρτα bits και **συνδυάζεται** κατά τη μεταγλώττιση.

❓ **Τι γίνεται με τις εντολές άλματος όπως jal (Jump and Link);**

✅ Το jal είναι για **μεγαλύτερα άλματα** και έχει **μορφή τύπου UJ**.

📌 Όπως και στις διακλαδώσεις, το άλμα γίνεται **σχετικά προς το PC**:

Διεύθυνση προορισμού = PC + (άμεσο × 4)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

❓ **Πόσα bits έχει ο άμεσος στην jal και τι επιτρέπει;**

✅ Ο άμεσος είναι **20 bits**, που επιτρέπει **μεγάλη απόσταση άλματος** (προς τα μπροστά ή πίσω).

❓ **Πώς γίνεται άλμα σε πολύ μακρινή διεύθυνση (32-bit);**

✅ Τότε χρησιμοποιούμε:

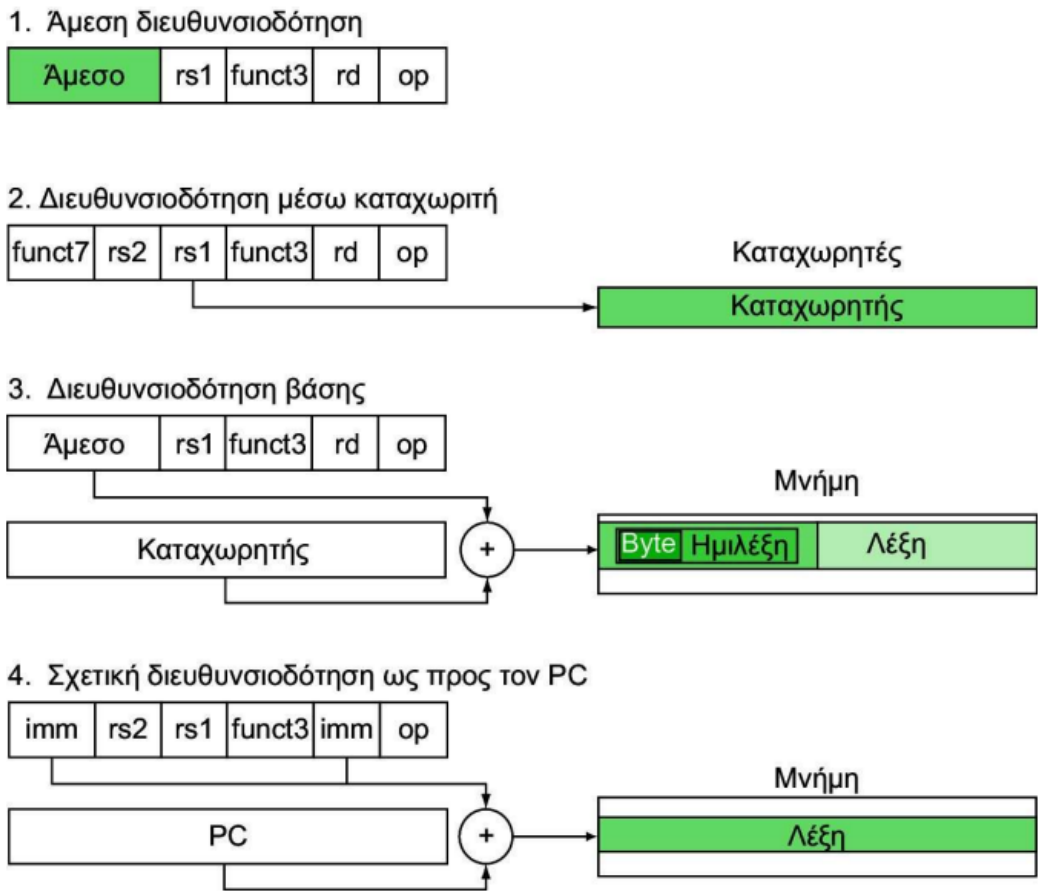
1. **lui**: φορτώνει τα **20 υψηλά bits** της διεύθυνσης (bits 31:12) σε καταχωρητή
2. **jalr**: προσθέτει τα **χαμηλά 12 bits** και κάνει το άλμα (μέσω καταχωρητή)

📌 Έτσι υποστηρίζεται και άλμα σε **οποιαδήποτε 32-bit διεύθυνση**

🔁 **Σύγκριση μορφών:**

Εντολή	Μορφή	Άλμα προς...	Τύπος διευθυνσιοδότησης
beq, bne, blt	SB	**κοντινή διεύθυνση**	**PC-relative με άμεσο 12-bit**
jal	UJ	**μακρινή διεύθυνση**	**PC-relative με άμεσο 20-bit**
jalr	I-type	**διεύθυνση από καταχωρητή**	Άλμα μέσω υπολογισμένης διεύθυνσης

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



✅ **1. Άμεση διευθυνσιοδότηση (Immediate Addressing)**

- **Πού χρησιμοποιείται:** π.χ. σε addi, andi, ori
- **Τι συμβαίνει:** Η εντολή χρησιμοποιεί **μια σταθερά τιμή (άμεσος)**, **όχι μνήμη**.

📦 **Παράδειγμα:**

addi x5, x6, 10 # x5 = x6 + 10

🧠 Ο υπολογισμός γίνεται **μέσα στους καταχωρητές**, με **σταθερό αριθμό** (10 εδώ).

✅ **2. Διευθυνσιοδότηση μέσω καταχωρητή (Register Addressing)**

- **Πού χρησιμοποιείται:** π.χ. σε add x1, x2, x3
- **Τι συμβαίνει:** Και οι **δύο τιμές προέρχονται από καταχωρητές** — δεν υπάρχει πρόσβαση στη μνήμη.

📦 **Παράδειγμα:**

add x5, x6, x7 # x5 = x6 + x7

🧠 Όλη η πράξη γίνεται **μέσα στην CPU**, μεταξύ καταχωρητών.

✅ **3. Διευθυνσιοδότηση βάσης (Base + Offset Addressing)**

🔹 **Πού χρησιμοποιείται:** σε εντολές **προσπέλασης μνήμης** όπως lw, sw, lb, sb

🔹 **Τι συμβαίνει:**

- Η μνήμη προσπελαύνεται μέσω **μιας διεύθυνσης = καταχωρητής + άμεσος**
- Δηλαδή: χρησιμοποιούμε έναν **καταχωρητή ως βάση** + μια **σταθερά απόσταση (offset)**

📦 **Παράδειγμα:**

lw x5, 4(x6) # x5 = λέξη από μνήμη στη διεύθυνση x6 + 4

🧠 Έτσι γίνεται η ανάγνωση π.χ. ενός στοιχείου από πίνακα array[i].

✅ **4. Σχετική διευθυνσιοδότηση ως προς το PC (PC-relative Addressing)**

🔹 **Πού χρησιμοποιείται:** σε **εντολές διακλάδωσης** (beq, bne, jal, jalr)

🔹 **Τι συμβαίνει:**

- Η διεύθυνση υπολογίζεται **σε σχέση με την τρέχουσα εντολή** (δηλ. το **PC**, Program Counter)
- Ο άμεσος λέει **πόσο μπροστά ή πίσω** να πάει

📦 **Παράδειγμα:**

beq x1, x2, Label # Άλμα στο Label αν x1 == x2

🧠 Το Label υπολογίζεται σαν:

νέα διεύθυνση = PC + (άμεσο × 4)

🔄 **Συνοψίζοντας:**

Τύπος	Χρήση	Παράδειγμα
Άμεση (Immediate)	Πράξεις με σταθερές τιμές	addi x5, x6, 10
Μέσω καταχωρητή	Πράξεις μεταξύ καταχωρητών	add x5, x6, x7
Βάσης + offset	Πρόσβαση σε μνήμη (load/store)	lw x5, 4(x6)
Σχετική με το PC	Διακλαδώσεις & άλματα	beq x1, x2, Label

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

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

✅ Γενική Δομή Εντολών στη RISC-V

Κάθε εντολή στη RISC-V έχει **μήκος 32 bits**, αλλά χωρίζεται σε **διάφορα πεδία**, ανάλογα με τον **τύπο (format)** της εντολής.

🔷 Τύπος R – *Μορφή αριθμητικής εντολής*

Πεδία	**Bits**	**Περιγραφή**
funct7	7	Επέκταση του κωδικού λειτουργίας
rs2, rs1	5, 5	Πηγαίοι καταχωρητές (δεδομένα)
funct3	3	Τύπος πράξης
rd	5	Καταχωρητής προορισμού
opcode	7	Βασικός κωδικός λειτουργίας

📌 Χρήση: add, sub, and, or, sll, slt, κ.ά.

🔷 Τύπος I – *Φορτώσεις και αριθμητικές με άμεσο*

Πεδία	**Bits**
immediate[11:0]	12-bit άμεσος τελεστέος
rs1, rd	5, 5
funct3, opcode	3, 7

📌 Χρήση: addi, lw, jalr, andi, ori, lb, lh, κ.ά.

🔷 Τύπος S – *Αποθηκεύσεις*

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Πεδία	Bits
immediate[11:5]	7
rs2, rs1	5, 5
funct3	3
immediate[4:0]	5
opcode	7

📌 Χρήση: sw, sb, sh (store εντολές)

🔷 **Τύπος SB – *Διακλαδώσεις υπό συνθήκη***

Πεδία	Bits
immed[12], [10:5]	1 + 6 = 7
rs2, rs1	5, 5
funct3	3
immed[4:1], immed[11]	4 + 1 = 5
opcode	7

📌 Χρήση: beq, bne, blt, bge, κ.ά.

📌 Ο άμεσος σχηματίζεται **από διάσπαρτα bits**!

🔷 **Τύπος UJ – *Άλματα χωρίς συνθήκη***

Πεδία	Bits
immediate[20,10:1,11,19:12]	20-bit διεύθυνση (συγκεκριμένη σειρά)
rd, opcode	5, 7

📌 Χρήση: jal

📌 Για **μεγάλα άλματα** (μέχρι ±1 MB περίπου)

🔷 **Τύπος U – *Άμεσος άνω μέρους***

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Πεδία	Bits
immediate[31:12]	20-bit
rd, opcode	5, 7

📌 Χρήση: lui, auipc

📌 **Παρατηρήσεις:**

- Οι **φορτώσεις και άλματα με άμεσο** έχουν **άμεσο τελεστέο 12 ή 20 bits**
- Οι **διακλαδώσεις** χρησιμοποιούν **σχετική διεύθυνση** σε μορφή SB (με bits σκορπισμένα)
- Οι **εντολές τύπου R** είναι **καθαρές πράξεις μεταξύ καταχωρητών**

🔹 §2.11 Παραλληλία και Εντολές

❓ **Τι είναι ο συγχρονισμός μεταξύ επεξεργαστών;**

✅ Όταν δύο ή περισσότεροι επεξεργαστές (π.χ. P1 και P2) **μοιράζονται την ίδια μνήμη**, πρέπει να **συνεργάζονται σωστά**:

- Αν ο ένας **γράφει** και ο άλλος **διαβάζει** ταυτόχρονα χωρίς συντονισμό → προκύπτει **σύγκρουση (data race)**
- Το **τελικό αποτέλεσμα εξαρτάται** από τη σειρά εκτέλεσης → απρόβλεπτη συμπεριφορά

❓ **Τι χρειάζεται για να αποφευχθεί ο συναγωνισμός δεδομένων;**

✅ Υποστήριξη από το **υλικό (hardware)** για:

- **Αδιαίρετη λειτουργία**: ανάγνωση και έπειτα εγγραφή **χωρίς παρεμβολή**
- Μπορεί να γίνει με:
  - **Μία μόνο εντολή**
  - Ή **ζεύγος ειδικών εντολών** που «κλειδώνουν» τη μνήμη προσωρινά

✅ **Συγχρονισμός στη RISC-V**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Η RISC-V χρησιμοποιεί ένα **ζεύγος εντολών**:

🔷 **1. lr.w — Load Reserved**

lr.w rd, (rs1)

- Διαβάζει τη λέξη από τη διεύθυνση rs1 και τη βάζει στο rd
- Ταυτόχρονα **"κρατά" τη διεύθυνση** ως δεσμευμένη (reserved)

🔷 **2. sc.w — Store Conditional**

sc.w rd, rs2, (rs1)

- Προσπαθεί να **αποθηκεύσει** τη τιμή του rs2 στη διεύθυνση rs1
- Επιτυχία **μόνο αν** η μνήμη **δεν τροποποιήθηκε** από άλλο thread μετά το lr.w
- Επιστρέφει:
  - **0 στο rd** αν η αποθήκευση **πέτυχε**
  - **Μη μηδενική τιμή** αν **απέτυχε**

🧪 **ΠΑΡΑΔΕΙΓΜΑΤΑ**

🟩 **Παράδειγμα 1: Ατομική αντιμετάθεση**

```
again:
  lr.w x10, (x20)        # Διάβασε τιμή από x20
  sc.w x11, x23, (x20)   # Απόπειρα να γράψουμε x23 στη μνήμη
  bne x11, x0, again     # Αν απέτυχε (x11 ≠ 0), ξαναπροσπάθησε
  addi x23, x10, 0       # Αν πέτυχε, αποθήκευσε την παλιά τιμή σε x23
```

📌 Χρησιμοποιείται για **ανταλλαγή τιμών** με έλεγχο αν κάποιος «πήγε ενδιάμεσα».

🟧 **Παράδειγμα 2: Κλείδωμα**

```
addi x12, x0, 1        # Τιμή που δηλώνει "κλειδωμένο"


again:
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

```
lr.w x10, (x20)        # Διάβασε την τιμή κλειδώματος
bne x10, x0, again     # Αν είναι ήδη κλειδωμένο, επανάλαβε
sc.w x11, x12, (x20)   # Απόπειρα κλειδώματος
bne x11, x0, again     # Αν απέτυχε, επανάλαβε
```

📌 Αυτή είναι η **τυπική τεχνική "spinlock"** — περιμένουμε μέχρι να κλειδώσουμε επιτυχώς.

🟦 **Ξεκλείδωμα (Unlock)**

```
sw x0, 0(x20)          # Βάζει 0 στο κλείδωμα → το απελευθερώνει
```

📌 Με αυτό δηλώνουμε ότι **άλλος επεξεργαστής μπορεί πλέον να μπει στην κρίσιμη περιοχή**.

🔁 **Συμπερασματικά:**

Λειτουργία	Εντολή	Περιγραφή
Κράτηση τιμής	lr.w	Φόρτωση και «κλείδωμα» διεύθυνσης
Συνθήκη εγγραφής	sc.w	Γράφει μόνο αν δεν άλλαξε η διεύθυνση
Κλείδωμα	lr.w + sc.w	Με επανάληψη μέχρι επιτυχίας
Ξεκλείδωμα	sw x0, ...	Εγγραφή 0 → απελευθέρωση

🔹 **Αποκωδικοποίηση Εντολών RISC-V**

🧩 **Αποκωδικοποίηση εντολών RISC-V (1)**

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

ΑντιγραφήΕπεξεργασία

0x0044a283

Ερώτηση:
➡️ Ποια είναι η συμβολική μορφή της εντολής RISC-V που αντιστοιχεί σε αυτή τη δυαδική λέξη;

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🛠️ **Βήμα 1: Μετατροπή της λέξης σε δυαδικό**

0x0044a283 → 0000 0000 0100 0100 1010 0010 1000 0011

🔎 Αυτή είναι η δυαδική μορφή της εντολής, συνολικά 32 bits.

🛠️ **Βήμα 2: Εντοπισμός opcode και funct3**

Εντοπίζουμε τα σημαντικά πεδία μέσα στο 32-bit μοτίβο:

0000 0000 0100 0100 1010 0010 1000 0011

- **opcode** (bit 6:0): 0000011 → αντιστοιχεί σε **εντολή φόρτωσης**
- **funct3** (bit 14:12): 010

📌 Αυτή η τιμή του funct3 αντιστοιχεί στην εντολή **lw** (load word), σύμφωνα με την **κάρτα αναφοράς RISC-V**.

🧩 **Αποκωδικοποίηση εντολών RISC-V (2)**

Πρόκειται για εντολή lw, άρα είναι εντολή **μορφής I** (I-type). Τώρα μπορούμε να εντοπίσουμε και τα υπόλοιπα πεδία.

🛠️ **Βήμα 3: Ανάλυση πεδίων I-type**

Διαχωρίζουμε τα πεδία με βάση τη μορφή I:

Πεδία	Bits	Τιμή από τη λέξη
imm[11:0]	bits 31–20	000000000100 → **4**
rs1	bits 19–15	01001 → **x9**
funct3	bits 14–12	010
rd	bits 11–7	00101 → **x5**
opcode	bits 6–0	0000011

📜 **Τελικό αποτέλεσμα**

Άρα η εντολή είναι:

lw x5, 4(x9)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ Δηλαδή: **Φόρτωσε μία λέξη (32-bit)** από τη **διεύθυνση x9 + 4** και αποθήκευσέ την στον **καταχωρητή x5**.

📌 **Πρόσθετη επεξήγηση (για κατανόηση):**

- Η lw είναι εντολή **φόρτωσης (load)**: μεταφέρει δεδομένα **από τη μνήμη σε καταχωρητή**
- Χρησιμοποιεί **διευθυνσιοδότηση βάσης + offset**, δηλαδή:
  - Η **βάση** είναι ο rs1 (εδώ x9)
  - Το **offset** είναι η σταθερά 4
- Έτσι, αν ο x9 έχει τη διεύθυνση 1000, τότε διαβάζουμε τη λέξη από τη διεύθυνση 1004

Μερικές επιπλέον ασκήσεις από εμένα (πρέπει να έχετε ανοιχτή την κάρτα αναφοράς)

✅ **Επίπεδο 1 – Εύκολο**

🔸 **Παράδειγμα 1:**

Δεκαεξαδικό: 0x00128293

📌 **Ζητούμενο:** Να αποκωδικοποιηθεί σε συμβολική μορφή.

Μετατροπή: 00000000 00010000 10000010 10010011

- opcode = 0010011 → addi
- rd = x5
- rs1 = x5
- imm = 1

➡️ **Απάντηση:** addi x5, x5, 1

</details>

✅ **Επίπεδο 2 – Μέτριο**

🔸 **Παράδειγμα 2:**

Δεκαεξαδικό: 0xfe010113

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

📌 **Ζητούμενο:** Να αναγνωριστεί η εντολή και να εξηγηθεί τι κάνει.

Δυαδικό: 11111110 00000001 00000001 00010011

- opcode = 0010011 → addi
- rd = x2 (stack pointer)
- rs1 = x2
- imm = -32 (0xfe0 → -32 σε 12-bit 2's comp)

➡️ **Απάντηση:** addi sp, sp, -32
📌 Δηλαδή, **δέσμευση 32 bytes στον σωρό (stack)**.

✅ **Επίπεδο 3 – Δύσκολο**

🔸 **Παράδειγμα 3:**

Δεκαεξαδικό: 0x00c12083

📌 **Ζητούμενο:** Ποια είναι η συμβολική μορφή; Τι σημαίνει;

Δυαδικό: 00000000 11000001 00100000 10000011

- opcode = 0000011 → lw
- rd = x1 (return address)
- rs1 = x2 (sp)
- imm = 12

➡️ **Απάντηση:** lw ra, 12(sp)
📌 Δηλαδή, **φόρτωση διεύθυνσης επιστροφής από το stack**.

🧠 **Άσκηση για εξάσκηση (δοκίμασε μόνος σου):**

🔸 **Παράδειγμα 4:**

Δεκαεξαδικό: 0x00430313

📌 Μπορείς να βρεις ποια εντολή είναι; (Χρήση καρτέλας εντολών: πιθανό addi)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

📘 Μετάφραση και Εκκίνηση Προγράμματος

🔹 1. Πρόγραμμα C → Μεταγλωττιστής

- Ξεκινάμε από ένα **πρόγραμμα γραμμένο σε C**
- Ο **μεταγλωττιστής (compiler)** μεταφράζει το πρόγραμμα σε:

➤ **Πρόγραμμα συμβολικής γλώσσας (assembly)**
➤ Ή ακόμη και **απευθείας σε αντικειμενική μορφή** (π.χ. .o αρχεία)

📌 *Σημείωση:* Μερικοί μεταγλωττιστές **παράγουν απευθείας αντικειμενικές μονάδες**, χωρίς να μεσολαβεί συμβολομετάφραση.

🔹 2. Πρόγραμμα συμβολικής γλώσσας → Συμβολομεταφραστής

- Ο **συμβολομεταφραστής (assembler)** μεταφράζει τον assembly κώδικα σε:

✅ **Αντικείμενο (object file): υπομονάδα γλώσσας μηχανής**

🔹 3. Αντικείμενα → Πρόγραμμα σύνδεσης (Linker)

- Το **πρόγραμμα σύνδεσης (linker)** ενώνει:
  - Το αντικείμενο του προγράμματος (από τον χρήστη)
  - Τις ρουτίνες βιβλιοθήκης (π.χ. printf, scanf)

📌 Η διαδικασία αυτή λέγεται **στατική σύνδεση**, γιατί όλα τα κομμάτια του προγράμματος ενώνονται **πριν την εκτέλεση**.

📎 Προϊόν: **Εκτελέσιμο αρχείο (πρόγραμμα γλώσσας μηχανής)**

🔹 4. Εκτέλεση → Φορτωτής (Loader)

- Το **εκτελέσιμο αρχείο** φορτώνεται στη **μνήμη** από τον **φορτωτή (loader)**
- Από εδώ ξεκινά η **εκτέλεση του προγράμματος**

🧭 Διαδρομή μετατροπής:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Πρόγραμμα C

↓ (Μεταγλωττιστής)

Assembly

↓ (Συμβολομεταφραστής)

Αντικείμενο

↓ (Σύνδεση)

Εκτελέσιμο

↓ (Φόρτωση)

Μνήμη → Εκτέλεση

📌 **Συμπληρωματική επεξήγηση:**

- **Μεταγλωττιστής (Compiler):** C → Assembly / Object Code
- **Συμβολομεταφραστής (Assembler):** Assembly → Machine Code (object file)
- **Συνδέτης (Linker):** Ενώνει τα object files και τις βιβλιοθήκες
- **Φορτωτής (Loader):** Βάζει το εκτελέσιμο στη μνήμη και ξεκινά

🔹 **Μετάφραση, Σύνδεση και Εκτέλεση Προγράμματος**

✅ **1. Παραγωγή Αντικειμενικής Υπομονάδας**

- Ο **συμβολομεταφραστής** (ή ο **μεταγλωττιστής**) μεταφράζει το πρόγραμμα σε **εντολές γλώσσας μηχανής**.
- Παρέχει **όλα τα απαραίτητα στοιχεία** για να δημιουργηθεί το τελικό πρόγραμμα από διαφορετικά τμήματα.

📦 **Περιεχόμενα της αντικειμενικής υπομονάδας:**

- **Επικεφαλίδα:** περιγραφή των περιεχομένων του αρχείου
- **Τμήμα κειμένου:** οι μεταφρασμένες εντολές του προγράμματος
- **Τμήμα στατικών δεδομένων:** δεδομένα που υπάρχουν σε όλη τη διάρκεια εκτέλεσης
- **Πληροφορίες επανατοποθέτησης:** αναφορές που εξαρτώνται από την τελική θέση στη μνήμη

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Πίνακας συμβόλων:** καθολικά ονόματα και εξωτερικές αναφορές (π.χ. κλήσεις σε printf)
- **Πληροφορίες αποσφαλμάτωσης:** για αντιστοίχιση με τον πηγαίο κώδικα (χρήσιμο σε debugging)

✅ **2. Σύνδεση Αντικειμενικών Υπομονάδων**

➡️ Παράγεται **ένα εκτελέσιμο αρχείο**, μέσω **στατικής σύνδεσης**.

📌 **Βήματα της διαδικασίας:**

1. **Συγχώνευση** των τμημάτων (εντολές + δεδομένα)
2. **Προσδιορισμός ετικετών**: οι διευθύνσεις τους στο τελικό πρόγραμμα
3. **Επιδιόρθωση αναφορών** (εσωτερικών & εξωτερικών):
    - Π.χ. main καλεί printf, που είναι από εξωτερική βιβλιοθήκη

📌 Αν υπάρχουν εξαρτήσεις από τη θέση του προγράμματος, αυτές:

- **Μπορούν να διορθωθούν από φορτωτή επανατοποθέτησης**
- **Ή παρακάμπτονται πλήρως**, αν χρησιμοποιείται **εικονική μνήμη** (σε αυτή την περίπτωση το πρόγραμμα φορτώνεται σε **συγκεκριμένη σταθερή διεύθυνση**)

✅ **3. Φόρτωση του Προγράμματος (Loader)**

Ο **φορτωτής** διαβάζει το εκτελέσιμο από τον δίσκο και το φέρνει στη **μνήμη**.

📌 **Βήματα του φορτωτή:**

1. **Διαβάζει την επικεφαλίδα** για να καταλάβει πόσο χώρο χρειάζονται τα τμήματα
2. **Δημιουργεί τον χώρο εικονικών διευθύνσεων**
3. **Αντιγράφει τις εντολές και τα δεδομένα** στη μνήμη (ή ρυθμίζει τον πίνακα σελίδων ώστε να γίνει μόνο αν προκύψει ανάγκη - lazy loading)
4. **Αντιγράφει παραμέτρους** στη **στοίβα**
5. **Αρχικοποιεί καταχωρητές** (π.χ. sp για stack pointer, fp, gp)
6. **Μεταβαίνει στη ρουτίνα εκκίνησης (startup)**:
    - Αντιγράφει τα ορίσματα στους καταχωρητές x10, x11, …

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Καλεί τη main
- Όταν η main επιστρέψει, γίνεται κλήση exit

✅ 4. Δυναμική Σύνδεση (Dynamic Linking)

📌 Αντί να συνδεθούν όλες οι βιβλιοθήκες **πριν την εκτέλεση**, όπως στη στατική σύνδεση, εδώ:

- Οι **βιβλιοθήκες φορτώνονται και συνδέονται μόνο όταν κληθούν**
- Ο **κώδικας τους μπορεί να μεταφερθεί** οπουδήποτε στη μνήμη
- Το εκτελέσιμο είναι **πιο μικρό**
- Οι βιβλιοθήκες μπορούν να **ενημερώνονται ανεξάρτητα**, και το πρόγραμμα **να χρησιμοποιεί πάντα την τελευταία έκδοση**

🧭 Ανακεφαλαίωση Ροής:

Πρόγραμμα C

↓ μεταγλώττιση

Assembly ή Object Code (.o)

↓ σύνδεση

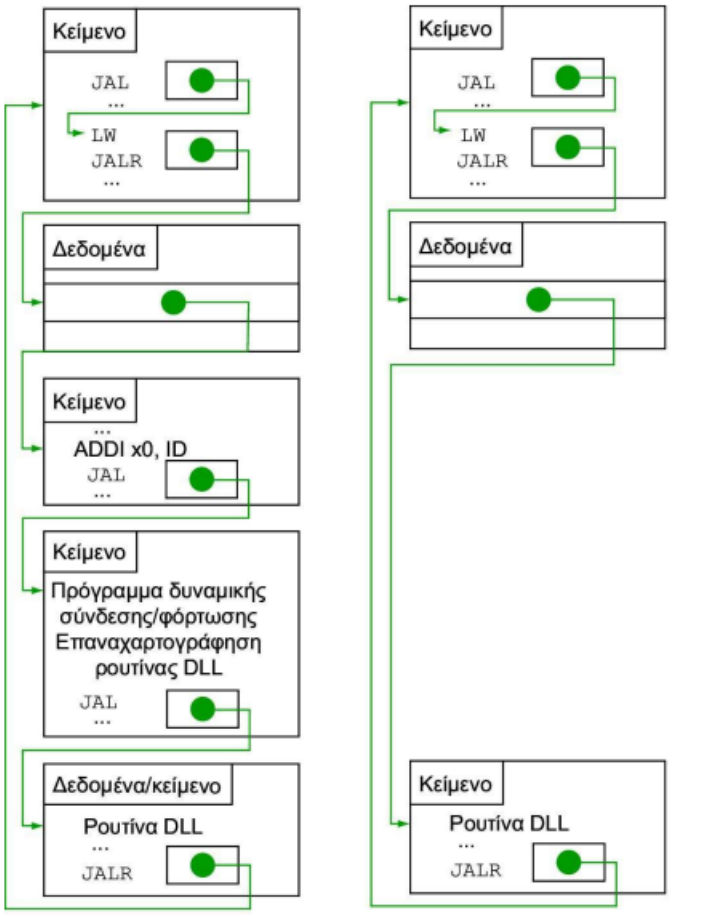
Εκτελέσιμο (.exe)

↓ φόρτωση

Μνήμη → Εκτέλεση → Τερματισμός

📘 Ράθυμη σύνδεση (Lazy Linking)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



(α) Πρώτη κλήση ρουτίνας DLL (β) Επόμενες κλήσεις ρουτίνας DLL

Η «ράθυμη» ή **τεμπέλικη** σύνδεση είναι μια τεχνική που επιτρέπει στο πρόγραμμα να **μη συνδέεται αμέσως με όλες τις εξωτερικές ρουτίνες**, αλλά **μόνο όταν αυτές ζητηθούν**.

Κύρια Στοιχεία Εικόνας:

Ορισμός	Περιγραφή
Πίνακας έμμεσότητας	Περιέχει διευθύνσεις ή δείκτες που αρχικά οδηγούν στον μηχανισμό φόρτωσης, αλλά αλλάζουν μετά την πρώτη χρήση
Στέλεχος	Τμήμα κώδικα που αναγνωρίζει ποια ρουτίνα ζητήθηκε και καλεί το κατάλληλο loader
Κώδικας σύνδεσης-φόρτωσης	Εκτελεί τη δυναμική σύνδεση (π.χ. βρίσκει τη διεύθυνση μιας ρουτίνας από DLL)

Ορισμός	Περιγραφή
Κώδικας που χαρτογραφείται δυναμικά	Δημιουργείται ή συνδέεται **μόνο όταν χρειαστεί** (δηλ. κατά την πρώτη κλήση)

🔁 **Τι δείχνει η εικόνα:**

Η εικόνα χωρίζεται σε **δύο στήλες**:

🟥 **(α) Πρώτη κλήση ρουτίνας DLL**

- Ο κώδικας JAL προσπαθεί να καλέσει τη ρουτίνα
- Η διεύθυνση που χρησιμοποιείται (μέσω πίνακα έμμεσότητας) **δεν δείχνει ακόμα τη ρουτίνα**
- Αντί γι' αυτό, οδηγεί σε:
  - ➤ Μια **ενδιάμεση ρουτίνα** (στέλεχος)
  - ➤ Που κάνει **JALR** στον μηχανισμό **δυναμικής σύνδεσης**
  - ➤ Ο δυναμικός σύνδεσμος (π.χ. loader του OS) **φορτώνει τη σωστή DLL ρουτίνα**
  - ➤ **Ενημερώνει τον πίνακα έμμεσότητας** ώστε **επόμενες κλήσεις** να πηγαίνουν κατευθείαν στη σωστή διεύθυνση

🟩 **(β) Επόμενες κλήσεις της ίδιας ρουτίνας**

- Τώρα, ο πίνακας έμμεσότητας έχει την **πραγματική διεύθυνση** της ρουτίνας
- Οι κλήσεις JAL οδηγούν **απευθείας** στη σωστή ρουτίνα της DLL

✅ **Πλεονεκτήματα "Ράθυμης" σύνδεσης**

- Δεν φορτώνονται όλες οι βιβλιοθήκες στην αρχή → **εξοικονόμηση χρόνου και μνήμης**
- Οι βιβλιοθήκες φορτώνονται **μόνο όταν ζητηθούν**
- Υποστηρίζει **δυναμική αντικατάσταση βιβλιοθηκών**

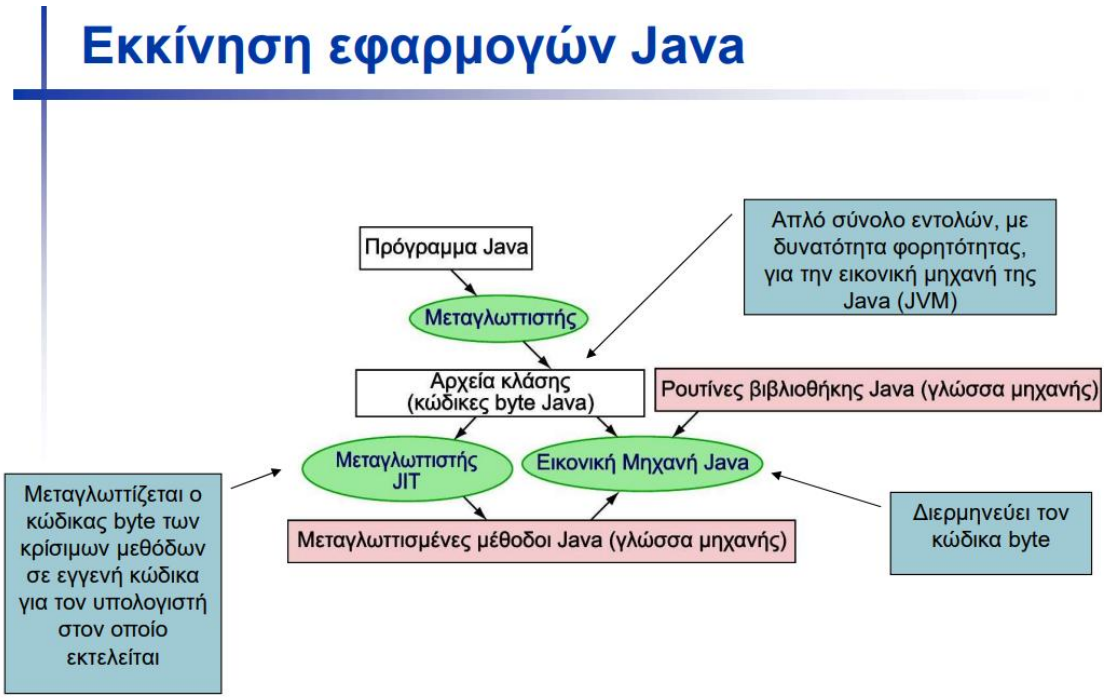
ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🧭 **Σχηματική ροή:**

JAL → Πίνακας έμμεσότητας → Ενδιάμεση ρουτίνα

→ Δυναμικός σύνδεσμος → Πραγματική ρουτίνα DLL

→ Ενημέρωση πίνακα → Επόμενες κλήσεις πάνε απευθείας



Βήμα-βήμα εκτέλεση εφαρμογής Java

✅ **1. Πρόγραμμα Java**

- Ο προγραμματιστής γράφει τον κώδικα σε γλώσσα **Java**
- Παράδειγμα: HelloWorld.java

✅ **2. Μεταγλωττιστής Java (javac)**

- Μεταγλωττίζει τον πηγαίο κώδικα σε:

Αρχεία κλάσης → Κώδικας byte Java (.class)

- Αυτός ο κώδικας **δεν είναι μηχανικός** (για επεξεργαστή), αλλά **εικονικός** για την JVM

📌 **Ο κώδικας byte Java είναι απλός και φορητός**, δηλαδή:

Μπορεί να εκτελεστεί σε οποιοδήποτε σύστημα που διαθέτει την Java Virtual Machine (JVM)

✅ 3. Εικονική Μηχανή Java (JVM)

Η JVM μπορεί να:

🔸 ➤ **Διερμηνεύσει τον bytecode:**

- **Εκτελεί** εντολές bytecode **μία-μία**
- Παράγει **προβλέψιμη αλλά όχι ταχύτατη** απόδοση

🔸 ➤ **Χρησιμοποιήσει JIT μεταγλωττιστή (Just-In-Time):**

- Ανιχνεύει **κρίσιμες μεθόδους** που εκτελούνται συχνά
- Τις μεταγλωττίζει **σε μηχανικό κώδικα (native)** για το σύστημα που εκτελείται
- Αυτές οι μέθοδοι **τρέχουν πολύ πιο γρήγορα** από τον bytecode

📌 Έτσι εξισορροπεί:

✅ **Φορητότητα** με τη JVM
✅ **Απόδοση** με JIT μεταγλώττιση

✅ 4. Ρουτίνες βιβλιοθήκης Java

- Κατά την εκτέλεση, το πρόγραμμα Java χρησιμοποιεί:
  - **Κλάσεις βιβλιοθηκών** της Java (π.χ. System.out, ArrayList, κ.λπ.)
  - Αυτές μεταφράζονται επίσης σε bytecode και εκτελούνται με τον ίδιο τρόπο

📌 **Επιπλέον πληροφορίες από την εικόνα:**

Στοιχείο στην εικόνα	Επεξήγηση
🟦 «Απλό σύνολο εντολών, με φορητότητα»	Ο κώδικας byte Java είναι **ανεξάρτητος πλατφόρμας**
🟦 «Διερμηνεύει τον κώδικα byte»	Η JVM μπορεί να **τρέχει bytecode απευθείας**
🟦 «Μεταγλωττίζεται σε εγγενή κώδικα»	Ο **JIT compiler** βελτιώνει την ταχύτητα

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Στοιχείο στην εικόνα	Επεξήγηση
🟪 «Ρουτίνες βιβλιοθήκης Java»	Πρόσθετες εντολές/βοηθητικές κλάσεις που χρησιμοποιούνται στο πρόγραμμα

✅ **Πλεονεκτήματα της προσέγγισης Java:**

- **Φορητότητα**: Εκτελείται σε κάθε υπολογιστή με JVM
- **Απόδοση**: Το JIT μεταγλωττίζει "καυτά σημεία" του κώδικα σε εγγενή μορφή
- **Ευκολία**: Ο προγραμματιστής γράφει μία φορά — τρέχει παντού

📘 §2.13 — Ένα Παράδειγμα Ταξινόμησης στη C

(που τα συνδυάζει όλα)

✅ **Περιγραφή:**

Το παράδειγμα παρουσιάζει μια **ταξινόμηση φυσαλίδας (bubble sort)** υλοποιημένη στη C και μεταφρασμένη σε **εντολές RISC-V**. Περιλαμβάνει και μια **βοηθητική συνάρτηση swap**, που είναι διαδικασία-φύλλο (δεν καλεί άλλες διαδικασίες).

🔁 **Συνάρτηση swap στη C:**

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

📌 **Καταχωρητές:**

- v αποθηκεύεται στον x10
- k στον x11
- temp χρησιμοποιεί τον x5 προσωρινά

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

💻 **Μεταγλωττισμένος κώδικας RISC-V:**

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

🔁 **Συνάρτηση sort στη C:**

Η κύρια διαδικασία που **καλεί τη swap** για να ταξινομήσει τον πίνακα.

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

📌 **Καταχωρητές:**

- v στον x10
- n στον x11
- i στον x19
- j στον x20

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

📌 **Τι συνδυάζει αυτό το παράδειγμα:**

- Κλήσεις διαδικασιών (swap)
- Δομές ελέγχου (for, if, >=, &&)
- Χρήση πίνακα (με μετατροπή v[k] σε διευθύνσεις)
- Αριθμητικές/λογικές πράξεις
- Δυναμική πρόσβαση στη μνήμη
- Συσχέτιση C → RISC-V κώδικα

🔹 **Ο εξωτερικός και ο εσωτερικός βρόχος του sort**

🔁 **Εξωτερικός βρόχος**

🔸 **Κώδικας σε C:**

```
for (i = 0; i < n; i++) {
  ...
}
```

🔸 **RISC-V Assembly:**

```
li x19, 0          // i = 0
for1tst:
bge x19, x11, exit1 // αν i ≥ n, έξοδος από τον εξωτερικό βρόχο
  ...              // (σώμα του εξωτερικού βρόχου)
addi x19, x19, 1   // i++
j for1tst          // επανέλεγχος συνθήκης (loop)
exit1:
```

📌 **Σημειώσεις:**

- Ο καταχωρητής x19 χρησιμοποιείται για τη μεταβλητή i
- Ο καταχωρητής x11 περιέχει την τιμή n (μέγεθος πίνακα)

🔁 **Εσωτερικός βρόχος**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔸 **Κώδικας σε C:**

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

🔸 **RISC-V Assembly:**

```
addi x20, x19, -1     // j = i - 1
for2tst:
blt x20, x0, exit2    // αν j < 0 → έξοδος από τον εσωτερικό βρόχο

slli x5, x20, 2       // x5 = j * 4 (για byte offset)
add x5, x10, x5       // x5 = διεύθυνση v[j]
lw x6, 0(x5)          // x6 = v[j]
lw x7, 4(x5)          // x7 = v[j + 1]
ble x6, x7, exit2     // αν v[j] ≤ v[j + 1] → έξοδος

mv x21, x10           // αποθήκευση της v στον x21
mv x22, x11           // αποθήκευση της n στον x22 (ασφαλής επιστροφή)

mv x10, x21           // παράμετρος 1 για swap: v
mv x11, x20           // παράμετρος 2 για swap: j
jal x1, swap          // κλήση της swap

addi x20, x20, -1     // j--
j for2tst             // επανέλεγχος συνθήκης εσωτερικού βρόχου
exit2:
```

📌 **Επεξηγήσεις:**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Καταχωρητής Ρόλος

x10	v[] – δείκτης στον πίνακα
x11	n – μέγεθος πίνακα
x19	i – μεταβλητή εξωτερικού βρόχου
x20	j – μεταβλητή εσωτερικού βρόχου
x5, x6, x7	βοηθητικοί για διευθύνσεις και τιμές στοιχείων του πίνακα
x21, x22	προσωρινή αποθήκευση τιμών που θα χαθούν στην swap
x1	καταχωρητής επιστροφής από τη swap

✅ **Τι δείχνει αυτός ο κώδικας:**

- Χρήση **δομών επανάληψης** με bge, blt, ble
- Υπολογισμός **διευθύνσεων πίνακα**
- Προσεκτική **προετοιμασία παραμέτρων** για κλήση συνάρτησης
- Χρήση **καταχωρητών γενικού σκοπού** ως τοπικές μεταβλητές

📘 **Διατήρηση και Επαναφορά Καταχωρητών**

(πριν και μετά την εκτέλεση διαδικασίας)

✅ **1. Διατήρηση (Αποθήκευση) των αποθηκευμένων καταχωρητών**

addi sp, sp, -20 // Δημιουργία χώρου στη στοίβα για 5 καταχωρητές (5 × 4 bytes = 20 bytes)

sw x1, 16(sp) // Αποθήκευση του καταχωρητή x1 (διεύθυνση επιστροφής)

sw x22, 12(sp) // Αποθήκευση του x22

sw x21, 8(sp) // Αποθήκευση του x21

sw x20, 4(sp) // Αποθήκευση του x20

sw x19, 0(sp) // Αποθήκευση του x19

📌 **Σκοπός:** Πριν καλέσουμε υπορουτίνα (π.χ. swap ή sort), αποθηκεύουμε κρίσιμους καταχωρητές ώστε να τους επαναφέρουμε αργότερα.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ **2. Επαναφορά (Ανάκτηση) των αποθηκευμένων καταχωρητών**

```
exit1:
lw x19, 0(sp)      // Επαναφορά του x19 από τη στοίβα
lw x20, 4(sp)      // Επαναφορά του x20
lw x21, 8(sp)      // Επαναφορά του x21
lw x22, 12(sp)     // Επαναφορά του x22
lw x1, 16(sp)      // Επαναφορά της διεύθυνσης επιστροφής (x1)

addi sp, sp, 20    // Επαναφορά του stack pointer (sp) στην προηγούμενη τιμή

jalr x0, 0(x1)     // Επιστροφή στην καλούσα ρουτίνα
```

📌 **Τι πετυχαίνουμε με αυτόν τον χειρισμό;**

- **Προστατεύουμε τις τιμές** των αποθηκευμένων καταχωρητών, ώστε να **μην αλλοιωθούν** κατά την εκτέλεση υπορουτινών.
- Επιτρέπουμε **ασφαλή χρήση της στοίβας**, σύμφωνα με το **κάλεσμα-επιστροφή** (call-return) μοντέλο.
- Διατηρούμε τη **συμβατότητα με τα συμβόλαια κλήσης** της RISC-V (calling conventions).

📝 **Υπενθύμιση:**

Καταχωρητής	Περιγραφή
x1 (ra)	Διεύθυνση επιστροφής
x19–x22	Συνήθως αποθηκευμένοι καταχωρητές (saved registers)
sp	Stack pointer (δείχνει την κορυφή της στοίβας)

📘 **Επιδράσεις βελτιστοποίησης του gcc**

📌 Ο gcc (GNU Compiler Collection) προσφέρει διάφορα επίπεδα βελτιστοποίησης κατά τη μεταγλώττιση:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- -O0: χωρίς καμία βελτιστοποίηση
- -O1: μέση βελτιστοποίηση
- -O2: πλήρης βελτιστοποίηση
- -O3: ενσωματώνει επιπλέον τεχνικές, όπως **ενσωμάτωση διαδικασιών (inline functions)**

📊 **Πίνακας Αποτελεσμάτων:**

Βελτιστοποίηση gcc	Σχετική απόδοση	Κύκλοι ρολογιού (εκατομμύρια)	Αριθμός εντολών (εκατομμύρια)	CPI (κύκλοι/εντολή)
καμία (O0)	1.00	158,615	114,938	1.38
-O1 (μέση)	2.37	66,990	37,470	1.79
-O2 (πλήρης)	2.38	66,521	39,993	1.66
-O3 (ενσωμάτωση)	2.41	65,747	44,993	1.46

🔍 **Ανάλυση των αποτελεσμάτων:**

✅ **Σχετική απόδοση:**

- Όσο μεγαλύτερη η τιμή, τόσο **γρηγορότερη** είναι η εκτέλεση σε σχέση με τη βασική (O0)
- Με -O3, το πρόγραμμα εκτελείται **2.41 φορές γρηγορότερα** από ό,τι χωρίς καμία βελτιστοποίηση!

✅ **Κύκλοι ρολογιού:**

- Ο **συνολικός χρόνος εκτέλεσης** μειώνεται σημαντικά:
  - Από **158,615 εκατομμύρια** (O0) σε **65,747 εκατομμύρια** (O3)
- Δηλαδή, **μικρότερος χρόνος επεξεργασίας**

✅ **Αριθμός εντολών:**

- Οι βελτιστοποιήσεις **μειώνουν τον αριθμό των εντολών** που εκτελούνται — το πρόγραμμα γίνεται πιο «σφιχτό»

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Το -O3 έχει **λίγο περισσότερες εντολές από το -O2**, επειδή κάνει περισσότερη «επεκτατική δουλειά» (όπως inline)

✅ **CPI (κύκλοι ανά εντολή):**

- **Μικρότερο CPI σημαίνει πιο αποτελεσματική εκτέλεση**
- Το -O0 έχει σχετικά χαμηλό CPI (1.38) γιατί κάνει απλά πράγματα
- Το -O3 αυξάνει τον αριθμό των εντολών, αλλά μειώνει τον χρόνο → **βελτιώνει την ταχύτητα με έξυπνες τεχνικές**

📌 **Συμπεράσματα:**

	Χωρίς Βελτιστοποίηση	**Με Βελτιστοποίηση**
🔻 Εκτέλεση	Πιο αργή	Πολύ πιο γρήγορη
🔻 Αξιοποίηση CPU	Κακή	Καλύτερη
🔺 Αριθμός εντολών	Περισσότερες	Λιγότερες/βελτιωμένες
🔺 Απόδοση	Χαμηλή	Υψηλή (έως 2.4x)

📘 **Πίνακας: Επίδραση της γλώσσας και του αλγορίθμου**

Γλώσσα	**Μέθοδος εκτέλεσης**	**Βελτιστοποίηση**	**Σχ. απόδοση ταξινόμησης φυσαλίδας**	**Σχ. απόδοση γρήγορης ταξινόμησης**	**Επιτάχυνση γρ. ταξινόμησης σε σχέση με φυσαλίδα**
C	μεταγλωττιστής	καμία	1.00	1.00	2468
	μεταγλωττιστής	O1	2.37	1.50	1562
	μεταγλωττιστής	O2	2.38	1.50	1555
	μεταγλωττιστής	O3	2.41	1.91	1955
Java	διερμηνευτής	—	0.12	0.05	1050
	μεταγλωττιστής JIT	—	2.13	0.29	338

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔍 **Τι δείχνουν οι στήλες:**

- **Σχετική απόδοση ταξινόμησης φυσαλίδας (bubble sort):** Πόσο γρήγορα εκτελείται ο αργός αλγόριθμος (μεταβλητή βάση 1.00 = C χωρίς βελτιστοποίηση)
- **Σχετική απόδοση γρήγορης ταξινόμησης (quick sort):** Πόσο καλύτερα εκτελείται ο αποδοτικός αλγόριθμος
- **Επιτάχυνση γρήγορης ταξινόμησης σε σχέση με φυσαλίδα:** Πόσες φορές γρηγορότερη είναι η quick sort σε σχέση με bubble sort στην ίδια περίπτωση

✅ **Παρατηρήσεις:**

🔸 **Γλώσσα C:**

- **Χωρίς βελτιστοποίηση**, έχει χαμηλή απόδοση, αλλά **μέγιστη επιτάχυνση** από quick sort: → **2468x πιο γρήγορη**
- Με -O3, η απόδοση και των δύο αλγορίθμων βελτιώνεται, όμως η **σχετική επιτάχυνση μειώνεται** (1955x) γιατί βελτιώνεται και η φυσαλίδα.

🔸 **Γλώσσα Java:**

- Ο **διερμηνευτής είναι πολύ αργός** → bubble: 0.12, quick sort: 0.05 (σε σχέση με τη βάση C O0)
- Ο **JIT compiler** αυξάνει την απόδοση του bubble σε **2.13**, αλλά δεν καταφέρνει να αποδώσει τόσο καλά στην quick sort (μόνο 0.29) → **μικρή επιτάχυνση (338x)**

📌 **Συμπεράσματα:**

Συμπέρασμα	Τι σημαίνει
✅ Ο αλγόριθμος έχει μεγαλύτερη επίδραση από τη γλώσσα	π.χ. quick sort είναι έως και **2.400x γρηγορότερη** από bubble sort
✅ Η βελτιστοποίηση του μεταγλωττιστή αυξάνει την απόδοση	π.χ. O3 → καλύτερο CPI, λιγότεροι κύκλοι
✅ Ο JIT κάνει τη Java ανταγωνιστική για απλούς αλγορίθμους	Αλλά ακόμα έχει χαμηλότερη απόδοση σε πιο σύνθετους αλγόριθμους

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Συμπέρασμα	Τι σημαίνει
✅ Η C παραμένει κορυφαία σε απόδοση	Ειδικά με -O2 ή -O3

📘 **Τι διαπιστώνουμε από τις προηγούμενες συγκρίσεις**

✅ **1. Το πλήθος εντολών και το CPI δεν αρκούν για να εκτιμήσουμε την απόδοση**

- Ένα πρόγραμμα μπορεί να εκτελεί **λίγες εντολές** ή να έχει **χαμηλό CPI** (κύκλοι ανά εντολή), αλλά αυτό **δεν σημαίνει απαραίτητα ότι είναι γρήγορο**.
- Πρέπει να εξετάζουμε **ολιστικά**: αριθμό εντολών, χρονισμό, μεταγλωττιστή, και αλγόριθμο.

✅ **2. Η επίδραση του αλγορίθμου είναι κρίσιμη**

- Η **αποτελεσματικότητα των βελτιστοποιήσεων** εξαρτάται **από το είδος του αλγορίθμου**.
- Παράδειγμα: Δεν έχει νόημα να βελτιστοποιείς τον bubble sort — παραμένει αργός συγκριτικά με quick sort.

✅ **3. Η Java με JIT είναι πολύ πιο γρήγορη από την διερμηνευόμενη Java**

- Ο **διερμηνευτής** της Java εκτελεί bytecode **γραμμή-γραμμή** → πολύ αργός.
- Ο **JIT μεταγλωττιστής** μετατρέπει κρίσιμα κομμάτια του bytecode σε μηχανικό κώδικα → **πολύ πιο αποδοτικός**.

✅ **4. Η Java με JIT μπορεί να πλησιάσει την C με βελτιστοποίηση**

- Αν και ο C compiler με -O3 αποδίδει καλύτερα, ο JIT **κλείνει την ψαλίδα** σε πολλές περιπτώσεις.
- Ιδιαίτερα σε εφαρμογές με **καλή αρχιτεκτονική αλγορίθμων**, η απόδοση μπορεί να είναι **συγκρίσιμη**.

✅ **5. Ο "χαζός" αλγόριθμος δεν διορθώνεται με μεταγλωττιστή**

✳️ "Αν ένας αλγόριθμος είναι χαζός, δεν γίνεται να διασωθεί!"

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Δηλαδή: **η επιλογή του αλγορίθμου είναι σημαντικότερη από τη βελτιστοποίηση**.
- Ένας καλός αλγόριθμος χωρίς βελτιστοποίηση συνήθως αποδίδει καλύτερα από έναν κακό αλγόριθμο με τέλεια μεταγλώττιση.

📘 **Πίνακες ή Δείκτες;**

✅ **1. Πώς γίνεται η δεικτοδότηση πίνακα**

Για να έχεις πρόσβαση στο στοιχείο v[i]:

- Υπολογίζεις τη **διεύθυνση**:

v + i * sizeof(στοιχείο)

- Αυτό σημαίνει ότι γίνεται:
  - **πολλαπλασιασμός**
  - **πρόσθεση στη διεύθυνση βάσης**

✅ **2. Οι δείκτες απλοποιούν τα πράγματα**

- Ένας δείκτης **είναι ήδη διεύθυνση** στη μνήμη.
- Με αυτόν, **παρακάμπτεις τον μηχανισμό δεικτοδότησης**.
- Ειδικά σε assembly ή χαμηλού επιπέδου γλώσσες, η χρήση δείκτη είναι **πιο άμεση και αποδοτική**.

🔍 **Τι προτιμάται;**

- **Δείκτες** για βελτίωση απόδοσης σε χαμηλού επιπέδου κώδικα (π.χ. RISC-V).
- **Πίνακες** για απλότητα σε υψηλού επιπέδου γλώσσες (π.χ. C, Java).

Παράδειγμα: Μηδενισμός ενός πίνακα

🟩 **Μέθοδος 1: Με χρήση δεικτών πίνακα (array[i])**

```
clear1(int array[], int size) {
  int i;
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

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

🔧 **Αντίστοιχος κώδικας RISC-V:**

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

📝 **Σχόλια:**

- x10 περιέχει τη βάση του πίνακα array
- x11 περιέχει το size
- Κάθε στοιχείο είναι 4 bytes (εξού και i * 4)

🟦 **Μέθοδος 2: Με χρήση δεικτών (*p = 0)**

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

🔧 **Αντίστοιχος κώδικας RISC-V:**

```
mv x5, x10          // p = διεύθυνση array[0]
slli x6, x11, 2     // x6 = size * 4
add x7, x10, x6     // x7 = διεύθυνση array[size]
loop2:
sw x0, 0(x5)        // *p = 0
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

```
addi x5, x5, 4       // p++
bltu x5, x7, loop2   // αν p < &array[size] → επανάληψη
```

📝 **Σχόλια:**

- Εδώ ο δείκτης p (στον x5) προχωρά ανά 4 bytes
- Η συνθήκη p < &array[size] υλοποιείται με bltu (unsigned)

🔍 **Συγκριτική ανάλυση:**

Χαρακτηριστικό	**clear1 (με πίνακα)**	**clear2 (με δείκτη)**
Τύπος προσπέλασης	Δεικτοδότηση array[i]	Προσπέλαση με δείκτη *p
Αριθμητικές πράξεις	Πολλαπλασιασμός, πρόσθεση	Μόνο πρόσθεση
Κώδικας	Λίγο πιο περίπλοκος	Πιο ευθύς σε assembly
Απόδοση	Ελαφρώς λιγότερη συνήθως	Πιο άμεση, πιο αποτελεσματική

✅ **Συμπέρασμα:**

- Η χρήση **δεικτών** είναι πιο άμεση και γενικά **αποδοτικότερη** σε assembly, καθώς **αποφεύγει τον πολλαπλασιασμό** και επιτρέπει **συνεχή πρόσβαση** στη μνήμη.
- Σε γλώσσες υψηλού επιπέδου, όμως, η χρήση array[i] είναι **ευκολότερη στην ανάγνωση** και **ασφαλέστερη**.

📘 **Σύγκριση: Πίνακας vs Δείκτες – Βελτιστοποίηση «Μείωσης της Δύναμης»**

✅ **Τι είναι η βελτιστοποίηση "μείωσης της δύναμης";**

- Είναι η **αντικατάσταση μιας βαριάς αριθμητικής πράξης** με μια **απλούστερη**.
  - Π.χ. αντί να κάνουμε i * 4 ➜ κάνουμε i << 2 (ολίσθηση αριστερά κατά 2 bit).

🔹 **Εκδοχή με πίνακα:**

```
array[i] = 0;
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Η **ολίσθηση (shift)** πρέπει να γίνεται **μέσα στον βρόχο**, επειδή το i αυξάνεται κάθε φορά και το offset πρέπει να επανυπολογίζεται.
- Αυτό σημαίνει περισσότερη δουλειά στη CPU ανά επανάληψη.

🔹 **Εκδοχή με δείκτες:**

*p = 0; p++;

- Δεν χρειάζεται ολίσθηση ή πολλαπλασιασμός, απλώς **αυξάνουμε τη διεύθυνση** με p = p + 1 ➜ πολύ πιο **αποδοτικό**.

✅ **Παρατηρήσεις:**

- Ο μεταγλωττιστής μπορεί συχνά να **εφαρμόσει αυτόματα** τέτοιες βελτιστοποιήσεις όταν βλέπει πίνακες.
- Αν όμως ο προγραμματιστής χρησιμοποιήσει **δείκτες ρητά**, βοηθά τον μεταγλωττιστή να **αποδώσει καλύτερα**.

🧠 **Εξάλειψη επαγωγικής μεταβλητής:**

- Αντί να διατηρείς μεταβλητή i **μόνο για να υπολογίζεις διεύθυνση**, είναι προτιμότερο να χρησιμοποιήσεις **δείκτη**.
- Αυτό μειώνει την πολυπλοκότητα, κάνει τον κώδικα **πιο καθαρό, πιο αξιόπιστο, πιο βελτιστοποιήσιμο**.

📘 **Εντολές του MIPS (σε σύγκριση με RISC-V)**

✅ **Γενικά χαρακτηριστικά MIPS:**

- Προκάτοχος του RISC-V, εμπορικά διαδεδομένος.
- Παρόμοιο με RISC-V ως προς:
  - **Σύνολο εντολών 32-bit**
  - **32 γενικοί καταχωρητές** (όπου x0 = 0)
  - **Προσπέλαση μνήμης μόνο με load/store**
  - **Απλός και κανονικός τρόπος διευθυνσιοδότησης**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ **Καταχωρητές MIPS:**

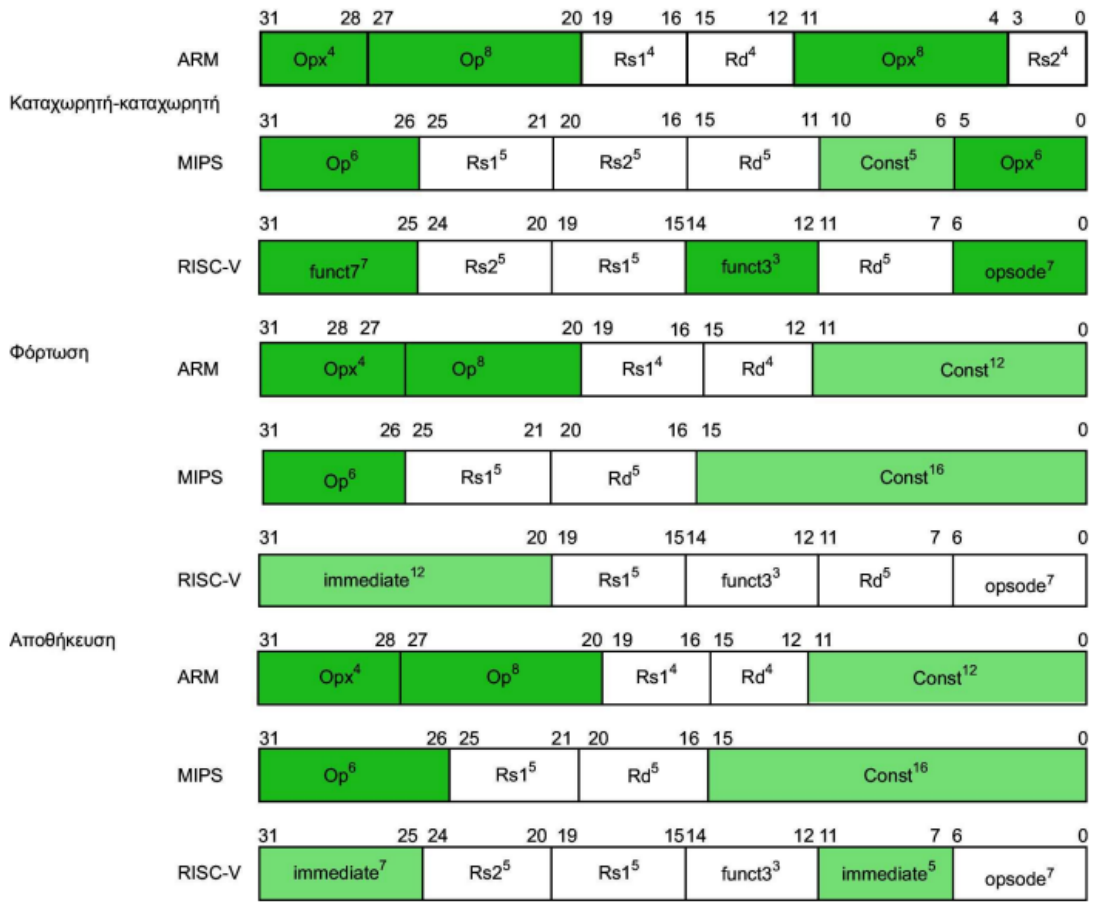
- 32 γενικής χρήσης
- 32 κινητής υποδιαστολής

✅ **Διακλαδώσεις: Διαφορές MIPS vs RISC-V**

RISC-V	**MIPS**
blt, bge, bltu, bgeu	Δεν υπάρχουν ως ξεχωριστές
Χρησιμοποιεί **απευθείας** εντολές σύγκρισης και διακλάδωσης	Χρησιμοποιεί slt, sltu + beq / bne
Π.χ. if (x < y) ➜ blt x, y, label	Π.χ. slt $t0, $x, $y → bne $t0, $zero, label

ΔΕΙΡΗ

Κωδικοποίηση εντολών



🧩 **Τύποι Εντολών:**

1. **Καταχωρητή-καταχωρητή (Register-to-register)**
2. **Φόρτωσης (Immediate format)**
3. **Αποθήκευσης (Store format)**

📘 **1. Εντολές Καταχωρητή-Καταχωρητή**

🔹 **Τι περιέχουν:**

- **Rd**: καταχωρητής προορισμού
- **Rs1, Rs2**: καταχωρητές προέλευσης
- **Opcode**: τύπος εντολής
- **Λειτουργίες (func, op2, etc.)**: προσδιορίζουν ειδικότερη λειτουργία

🧠 **Παρατηρήσεις:**

Αρχιτεκτονική	Σημαντικό στοιχείο
ARM	Δίνει βάση στα πεδία op1 και op2 για λεπτομερή έλεγχο
MIPS	Χρησιμοποιεί func + op συνδυαστικά για να προσδιορίσει την εντολή
RISC-V	Χρησιμοποιεί πολύ απλή και **κανονική** μορφή: funct7, funct3, rs1, rs2, rd, opcode

---

📘 **2. Εντολές Φόρτωσης / Άμεσου Τελεστέου (Immediate)**

🔹 **Τι περιέχουν:**

- **Άμεση τιμή (Constant / Immediate)**: τιμή σταθερά ενσωματωμένη στον κώδικα
- **Καταχωρητής προορισμού και βάσης**
- **Opcode & funct3** (κυρίως σε RISC-V)

🧠 **Παρατηρήσεις:**

Αρχιτεκτονική	Ιδιαιτερότητες
ARM	Άμεσος 12-bit Const
MIPS	Άμεσος 16-bit

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Αρχιτεκτονική Ιδιαιτερότητες

RISC-V	Άμεσος 12-bit σε εντολές τύπου I, τοποθετημένος πριν το rs1

📘 3. Εντολές Αποθήκευσης (Store)

🔹 **Τι διαφέρουν σε σχέση με φόρτωση:**

- Αντί για rd, έχουν **rs2 (πηγή)** και **rs1 (διεύθυνση βάσης)**
- Χρήση άμεσου offset για τη μνήμη

🧠 **Παρατηρήσεις:**

Αρχιτεκτονική Σημειώσεις

ARM	Παρόμοια μορφή με load
MIPS	Πολύ απλή τοποθέτηση: offset(rs1)
RISC-V	Το immediate **διαιρείται** (π.χ. imm[11:5] και imm[4:0]) σε εντολές τύπου S και ανασυντίθεται κατά την αποκωδικοποίηση

📌 **Συγκριτικό Συμπέρασμα:**

Κριτήριο	ARM	MIPS	RISC-V
Σαφήνεια	Μέτρια – πολυπλοκότητα	Καλή	Άριστη – πλήρης κανονικότητα
Επεκτασιμότητα	Περιορισμένη	Περιορισμένη	Υψηλή – ιδανική για RISC
Επεξεργασία Immediate	Συμπαγής μορφής	Απλή	Κανονική αλλά πιο αναλυτική

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔧 Η Αρχιτεκτονική x86 της Intel

📌 **Γενική περιγραφή:**

Η x86 είναι μια αρχιτεκτονική που εξελίχθηκε με **συνεχή συμβατότητα** με τις προηγούμενες εκδόσεις της, γεγονός που της επέτρεψε να κυριαρχήσει στην αγορά προσωπικών υπολογιστών.

🕰️ Ιστορική εξέλιξη των επεξεργαστών x86

- **8080 (1974)**
  - **Μικροεπεξεργαστής 8 bit**
  - Αρχιτεκτονική **συσσωρευτή (accumulator)**, μαζί με **3 ζεύγη δεικτών-καταχωρητών**
    - ➤ Έναρξη των βάσεων για την οικογένεια x86.

- **8086 (1978)**
  - Επέκταση του 8080 στα **16 bit**
  - Εισήγαγε ένα **πολύπλοκο σύνολο εντολών** (CISC – Complex Instruction Set Computer)
    - ➤ Πολλές εντολές με περίπλοκη λειτουργικότητα σε χαμηλό επίπεδο.

- **8087 (1980)**
  - **Συνεπεξεργαστής κινητής υποδιαστολής (FPU)**
  - Προσθήκη:
    - Εντολών κινητής υποδιαστολής (floating point)
    - **Στοίβα καταχωρητών** για αριθμητικές πράξεις

- **80286 (1982)**
  - Υποστήριξη διευθύνσεων **24 bit**
  - Εισάγει τη μονάδα διαχείρισης μνήμης (**MMU**)
  - Προσφέρει:
    - **Τμηματοποιημένη χαρτογράφηση (segmentation)**
    - **Προστασία μνήμης**

- **80386 (1985)**
  - Μεγάλη αλλαγή: **32 bit υποστήριξη (IA-32)**
  - Νέοι τρόποι διευθυνσιοδότησης και επιπλέον λειτουργίες

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Προστέθηκε:
    - **Χαρτογράφηση μνήμης με σελιδοποίηση (paging)**
    - Υποστήριξη για **τμήματα (segments)**

🧱 **Περαιτέρω εξέλιξη**

🔹 **i486 (1989)**

- **Διοχέτευση (pipelining)** για βελτίωση απόδοσης
- **Κρυφές μνήμες** στο εσωτερικό του chip
- Ενσωμάτωση της FPU
- ➤ Εμφανίζονται **ανταγωνιστές** όπως AMD, Cyrix

🔹 **Pentium (1993)**

- **Υπερβαθμωτός επεξεργαστής (superscalar)** ➤ ταυτόχρονη εκτέλεση πολλών εντολών
- **64-bit διαδρομή δεδομένων**
- Προστέθηκαν:
    - Εντολές **MMX (MultiMedia eXtension)** για πολυμέσα
- Γνωστός για:
    - Το διαβόητο σφάλμα **FDIV** (σφάλμα διαίρεσης κινητής υποδιαστολής)

🔹 **Pentium Pro (1995), Pentium II (1997)**

- Εισαγωγή **νέας μικροαρχιτεκτονικής**
- ➤ Αναφορά: *Colwell – The Pentium Chronicles*

🔹 **Pentium III (1999)**

- Προσθήκη:
    - Επεκτάσεων **SSE (Streaming SIMD Extensions)** για επεξεργασία δεδομένων σε πακέτα
    - Σχετικοί καταχωρητές

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔹 **Pentium 4 (2001)**

- Νέα μικροαρχιτεκτονική
- Εντολές **SSE2** για περισσότερες SIMD δυνατότητες

🚀 **Μετάβαση στα 64 bit και πιο πέρα**

🔹 **AMD64 (2003)**

- Επέκταση της x86 αρχιτεκτονικής στα **64 bit**
- ➤ Γνωστή και ως **x86-64**

🔹 **EM64T (2004)**

- Η Intel υιοθετεί την AMD64 με δικές της **βελτιώσεις**
- ➤ Το όνομα EM64T σημαίνει: **Extended Memory 64 Technology**
- Προστίθενται εντολές **SSE3**

🔹 **Intel Core (2006)**

- Προστίθενται:
  - Εντολές **SSE4**
  - Υποστήριξη για **εικονικές μηχανές (virtualization)**

🔹 **AMD64 (2007)**

- Ανακοίνωση εντολών **SSE5**
  - ➤ Η Intel **δεν τις ακολούθησε**. Αντ' αυτού...

🔹 **Advanced Vector Extensions – AVX (2008)**

- Η απάντηση της Intel:
  - **Μεγαλύτεροι καταχωρητές SSE**
  - **Περισσότερες εντολές SIMD**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- ➤ Στόχος: **Υψηλές επιδόσεις** σε multimedia, scientific, AI κ.ά.

📌 **Συμπέρασμα:**

❝ Αν η Intel δεν έκανε επεκτάσεις έχοντας κατά νου τη συμβατότητα, θα το έκαναν οι ανταγωνιστές της! ❞

- Η **τεχνική αρτιότητα** δεν είναι πάντα αρκετή.
- Η **συμβατότητα, το οικοσύστημα και η επιχειρηματική στρατηγική** καθορίζουν την **εμπορική επιτυχία**.

🔹 **Βασικοί τρόποι διευθυνσιοδότησης της x86**

🔸 **Δύο τελεστέοι ανά εντολή**

Η x86 αρχιτεκτονική επιτρέπει **δύο τελεστέους (operands)** σε κάθε εντολή — δηλαδή δύο "συστατικά" που παίρνουν μέρος στην πράξη (π.χ. καταχωρητές, τιμές, ή διευθύνσεις μνήμης).

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

📝 **Παρατήρηση**: Πάντα ο ένας από τους δύο τελεστέους μπορεί να είναι **καταχωρητής ή μνήμη**, και σπανιότερα άμεσος (immediate).

🔸 **Τρόποι διευθυνσιοδότησης μνήμης (Addressing Modes)**

Οι εντολές που προσπελαύνουν δεδομένα στη μνήμη, το κάνουν μέσω **σύνθετων διευθύνσεων**, οι οποίες μπορούν να υπολογιστούν με τους παρακάτω τρόπους:

✅ **1. Απλή διευθυνσιοδότηση με καταχωρητή**

- **Η διεύθυνση** βρίσκεται **σε έναν καταχωρητή**.
  - Παράδειγμα: mov eax, [ebx] ➜ χρησιμοποιεί τη διεύθυνση που έχει ο ebx.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ **2. Με μετατόπιση (displacement)**

- **Διεύθυνση = ΚΤΧβάσης + μετατόπιση**
  - Παράδειγμα: mov eax, [ebx+4]

✅ **3. Με αναφορά μέσω δείκτη και κλίμακας (scaled index)**

- **Διεύθυνση = ΚΤΧβάσης + 2^κλίμακα × ΚΤΧδείκτη**
  - Όπου κλίμακα = 0, 1, 2 ή 3 (δηλαδή επί 1, 2, 4 ή 8)
  - Παράδειγμα: mov eax, [ebx + esi*4]

✅ **4. Συνδυαστική (base + index × scale + offset)**

- **Διεύθυνση = ΚΤΧβάσης + 2^κλίμακα × ΚΤΧδείκτη + μετατόπιση**
  - Παράδειγμα: mov eax, [ebx + esi*4 + 8]
  - Χρήσιμη για πρόσβαση σε πίνακες: base + index × sizeof(element) + offset

---

💡 **Συμπέρασμα**

Η x86 υποστηρίζει **πολύ ευέλικτους τρόπους διευθυνσιοδότησης**, πράγμα που της επιτρέπει:

- Να υποστηρίζει πολύπλοκες προσβάσεις μνήμης σε μία εντολή,
- Αλλά με κόστος σε **απλότητα υλοποίησης**, ειδικά σε σύγκριση με αρχιτεκτονικές όπως η RISC-V που έχουν πιο **κανονική** και **προβλέψιμη** μορφή εντολών.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

α. JE EIP + μετατόπιση

4	4	8
JE	Συνθήκη	Μετατόπιση

β. CALL

8	32
CALL	Σχετική απόσταση

γ. MOV EBX, [EDI + 45]

6	1	1	8	8
MOV	d	w	Επιθεματικό byte r/m	Μετατόπιση

δ. PUSH ESI

5	3
PUSH	Καταχωρητής

ε. ADD EAX, #6765

4	3	1	32
ADD	Καταχωρητής	w	Άμεσο

στ. TEST EDX, #42

7	1	8	32
TEST	w	Επιθεματικό byte	Άμεσο

Κωδικοποίηση εντολών της x86

Μεταβλητό μήκος εντολών

Σε αντίθεση με αρχιτεκτονικές όπως η RISC-V, όπου κάθε εντολή έχει **σταθερό μήκος (32 bit)**, η x86 έχει **εντολές μεταβλητού μήκους**:

- Οι εντολές μπορεί να είναι από **1 έως 15 bytes**.
- Το τελικό μήκος εξαρτάται από:
  - την πράξη (π.χ. ADD, MOV, JE)
  - τους τελεστέους (operands)
  - τους τρόπους διευθυνσιοδότησης
  - την ύπαρξη άμεσων τιμών ή μετατοπίσεων

Τύποι bytes στην κωδικοποίηση

1. **Επιθεματικά bytes (ModR/M, SIB, displacement)**
   ➤ Καθορίζουν τον τρόπο προσπέλασης (διευθυνσιοδότησης):
   - Π.χ. αν αφορά καταχωρητή ή μνήμη, αν έχει μετατόπιση κ.λπ.

ΕΛΕΝΗ ΖΑΓΟΡΙΗ

2. **Προθεματικά bytes (Prefixes)**
   ➤ Τροποποιούν τη συμπεριφορά της εντολής:
   - Π.χ. αλλαγή μεγέθους τελεστέου (word/dword), επαναλήψεις (REP), κλείδωμα (LOCK)

📄 **Παραδείγματα από την εικόνα**

✔ **JE EIP + μετατόπιση**

- JE = εντολή διακλάδωσης εάν ίσο (jump if equal)
- Μήκος εντολής: **2 bytes**
  - Byte 1: opcode
  - Byte 2: **μετατόπιση**

✔ **CALL**

- Κλήση υπορουτίνας
- Έχει **1 byte opcode** και **4 bytes σχετικής απόστασης** → συνολικά 5 bytes

✔ **MOV EBX, [EDI + 4]**

- Αντιγραφή τιμής από μνήμη (στη θέση EDI + 4) στον EBX
- Εντολή **6 bytes**
  - 1 byte: opcode
  - 1 byte: ModR/M
  - 1 byte: SIB
  - 1 byte: μετατόπιση (displacement)

✔ **PUSH ESI**

- 1 byte εντολή που προωθεί τον ESI στη στοίβα

✔ **ADD EAX, #7656**

- Πρόσθεση άμεσης τιμής στον EAX
- Εντολή **6 bytes**
  - 1 byte: opcode
  - 1 byte: τροποποιητής (W – word size)
  - 4 bytes: άμεσος τελεστέος

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✔️ **TEST EDX, #42**

- Λογικός έλεγχος bit μεταξύ του EDX και της τιμής 42
- Περιλαμβάνει επιθεματικά + άμεση τιμή → συνολικό μήκος **3 bytes**

📌 **Συμπέρασμα**

Η x86 είναι μια **CISC αρχιτεκτονική** με **πολύπλοκη και εύκαμπτη κωδικοποίηση**:

- **Πλεονεκτήματα:**
  - Μπορεί να εκφράσει σύνθετες εντολές με λίγα bytes
  - Υποστηρίζει πολύπλοκους τρόπους διευθυνσιοδότησης
- **Μειονεκτήματα:**
  - Περισσότερη δυσκολία στην αποκωδικοποίηση για τον επεξεργαστή
  - Πολύπλοκη υλοποίηση της υποστήριξης στο hardware

🧠 **Υλοποίηση της IA-32 (Intel Architecture 32-bit)**

Η αρχιτεκτονική IA-32 (γνωστή και ως x86) έχει σημαντικές ιδιαιτερότητες που επηρεάζουν την **υλοποίηση στο υλικό (hardware)**:

🔹 **Σύνθετο Σύνολο Εντολών (CISC)**

- Το **πολύπλοκο σύνολο εντολών** της IA-32 δυσχεραίνει την υλοποίηση σε επίπεδο υλικού.
- Σε αντίθεση με τις αρχιτεκτονικές τύπου **RISC (όπως η RISC-V)** που έχουν απλές και προβλέψιμες εντολές, η IA-32 υποστηρίζει:
  - πολλές διαφορετικές μορφές εντολών
  - πολύπλοκους τρόπους διευθυνσιοδότησης
  - μεταβλητό μήκος εντολών

🔸 **Μετατροπή σε μικρολειτουργίες**

- Το **υλικό (ο επεξεργαστής)** μεταφράζει τις σύνθετες εντολές της IA-32 σε πιο **απλές εσωτερικές μικρολειτουργίες (micro-operations ή micro-ops)**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Τύπος εντολής	Μετατροπή σε μικρολειτουργίες
Απλή εντολή	1 προς 1 (μία εντολή → μία μικρολειτουργία)
Σύνθετη εντολή	1 προς πολλές (μία εντολή → πολλές μικρολειτουργίες)

📝 **Παράδειγμα**:

- Η εντολή MOV [EBX+4], EAX μπορεί να μετατραπεί σε:
    1. Υπολογισμός διεύθυνσης
    2. Ανάγνωση τιμής από καταχωρητή
    3. Εγγραφή τιμής στη μνήμη

🔹 **Χρήση μικρομηχανής τύπου RISC**

- Εσωτερικά, η CPU χρησιμοποιεί μια **μικρομηχανή τύπου RISC**, δηλαδή ένα **απλούστερο υποσύστημα** που υλοποιεί τις μικρολειτουργίες.
- Η προσέγγιση αυτή συνδυάζει:
    - τη **συμβατότητα** με την IA-32 (backward compatibility)
    - την **αποδοτικότητα** της RISC υλοποίησης

💸 **Βιωσιμότητα λόγω αγοράς**

- Αν και η IA-32 είναι πιο πολύπλοκη στην υλοποίηση:
    - Έχει **τεράστιο μερίδιο αγοράς**
    - Άρα είναι **οικονομικά βιώσιμη** (αξίζει το κόστος σχεδίασης)

⚖️ **Απόδοση και μεταγλωττιστές**

- Η τελική **απόδοση** είναι **συγκρίσιμη με εκείνη των RISC αρχιτεκτονικών**.
- Ωστόσο:
    - Οι **σύγχρονοι μεταγλωττιστές αποφεύγουν** τη χρήση των πιο σύνθετων εντολών της IA-32
    - Αντίθετα, προτιμούν να παράγουν απλούστερες εντολές, πιο φιλικές προς τη μικρομηχανή

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔷 Άλλες Εντολές του RISC-V

📌 Βασικές εντολές για ακέραιους (RV32I)

Οι ακόλουθες εντολές **προστίθενται επιπλέον** από όσες περιγράφηκαν προηγουμένως. Είναι πολύ χρήσιμες για αριθμητικές συγκρίσεις και υπολογισμούς διευθύνσεων.

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

🔍 Τι κάνουν αυτές οι εντολές;

- **slt, sltu, slti, sltiu**:
  - Επιστρέφουν 1 αν η συνθήκη ισχύει, αλλιώς 0.
  - Είναι χρήσιμες για **συνθήκες**, **διακλαδώσεις**, **λογικούς ελέγχους**.
- **auipc**:
  - Πολύ χρήσιμη για **υπολογισμό διευθύνσεων**, ειδικά όταν έχουμε **μεγάλες άμεσες τιμές**.
  - Συνήθως **συνδυάζεται με jalr** για άλματα σε απομακρυσμένες διευθύνσεις.

🔸 Παραλλαγή 64 bit: RV64I

Το RISC-V υποστηρίζει και 64μπιτη παραλλαγή με τα εξής χαρακτηριστικά:

- Οι **καταχωρητές** έχουν εύρος **64 bit**
- Όλες οι **λειτουργίες/πράξεις** λειτουργούν επίσης σε 64 bit

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Οι εντολές είναι ίδιες, αλλά εκτελούνται σε **μεγαλύτερο εύρος τιμών**

✅ **Συμπλήρωση για καλύτερη κατανόηση**:

RV32I	RV64I
Επεξεργάζεται 32-bit τιμές	Επεξεργάζεται 64-bit τιμές
Κατάλληλο για ενσωματωμένα συστήματα, μικρές εφαρμογές	Ιδανικό για λειτουργικά συστήματα, μεγάλες εφαρμογές, servers

🧩 Επεκτάσεις Συνόλου Εντολών του RISC-V

Το RISC-V έχει **αρθρωτή σχεδίαση**, που σημαίνει ότι μπορεί να επεκταθεί με επιπλέον ομάδες εντολών. Αυτές ονομάζονται **επεκτάσεις ISA (Instruction Set Architecture)** και καλύπτουν διαφορετικές ανάγκες.

📌 Βασικές και Προαιρετικές Επεκτάσεις

Επέκταση	Περιγραφή	Πλήθος εντολών
I	Βασικό σύνολο ακεραίων εντολών	51 εντολές
M	Πολλαπλασιασμός, διαίρεση και υπόλοιπο με ακέραιους αριθμούς	13 εντολές
A	Ατομικές (αδιαίρετες) λειτουργίες μνήμης	22 εντολές
F	Αριθμοί κινητής υποδιαστολής απλής ακρίβειας (32-bit)	30 εντολές
D	Αριθμοί κινητής υποδιαστολής διπλής ακρίβειας (64-bit)	32 εντολές
C	Συμπιεσμένες (compressed) εντολές – μικρότερη κωδικοποίηση (16-bit)	36 εντολές

✏️ Επεξηγήσεις για κάθε επέκταση:

- **I**: Υποχρεωτική. Περιλαμβάνει τις πιο βασικές αριθμητικές και λογικές εντολές, όπως add, sub, and, lw, sw.
- **M**: Αφορά αριθμητικές πράξεις υψηλότερης πολυπλοκότητας, όπως mul, div, rem.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **A**: Παρέχει ατομικές πράξεις πρόσβασης στη μνήμη. Είναι **χρήσιμες για πολυπρογραμματισμό** και συγχρονισμό (π.χ. lr, sc, atomic swaps).
- **F**: Επιτρέπει πράξεις με **κινητή υποδιαστολή (float)**, π.χ. fadd.s, fsub.s.
- **D**: Επεκτείνει την F προσθέτοντας υποστήριξη για **double precision (64-bit floating point)**.
- **C**: Εντολές 16-bit που **αντικαθιστούν συχνές 32-bit εντολές**, εξοικονομώντας χώρο στη μνήμη – ιδιαίτερα χρήσιμες για **ενσωματωμένα συστήματα**.

✅ **Παράδειγμα χρήσης:**

- Ένας μικροεπεξεργαστής που υλοποιεί **RV32IMAC**:
  - Υποστηρίζει: Integer, Multiply, Atomics, Compressed.
  - Χρήσιμος σε ενσωματωμένες συσκευές.

🧠 **Πλάνες – Λανθασμένες αντιλήψεις σχετικά με τις εντολές των επεξεργαστών**

🔁 **Αναδρομική συμβατότητα**

- **Τι σημαίνει;**
  Ότι **δεν αλλάζει το σύνολο εντολών** (instruction set) όταν εξελίσσεται η αρχιτεκτονική.
- **Τι συνέπεια έχει;**
  ➕ Προστατεύει τον υπάρχοντα κώδικα
  ➖ Αλλά **οι εντολές αυξάνονται συνεχώς** για λόγους επέκτασης

📈 Η **γραφική παράσταση** δείχνει αυτή τη συνεχή αύξηση του πλήθους των εντολών της **x86**, από το 1978 έως το 2018. Από ~100 εντολές φτάνουμε σε **πάνω από 1400**!

❌ **Πλάνη 1: «Πιο ισχυρές εντολές = Υψηλότερη απόδοση»**

- **Γιατί φαίνεται λογικό;**
  Οι ισχυρές εντολές κάνουν περισσότερα → άρα χρειάζονται **λιγότερες εντολές** για να ολοκληρωθεί μια εργασία.
- **Τι συμβαίνει στην πράξη;**
  - Οι **πολύπλοκες εντολές** είναι **δυσκολότερες στην υλοποίηση**
  - Μπορεί να **επιβραδύνουν συνολικά** την εκτέλεση του προγράμματος, **ακόμη και των απλών εντολών**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ **Σημερινή Πραγματικότητα:**

- Οι **μεταγλωττιστές** (compilers) είναι **πολύ καλοί στο να βγάζουν αποδοτικό κώδικα** από **απλές εντολές**
- **Σύγχρονοι μεταγλωττιστές** → καλύτερη απόδοση από ό,τι αν γράφαμε χειροκίνητα συμβολικό κώδικα (assembly)

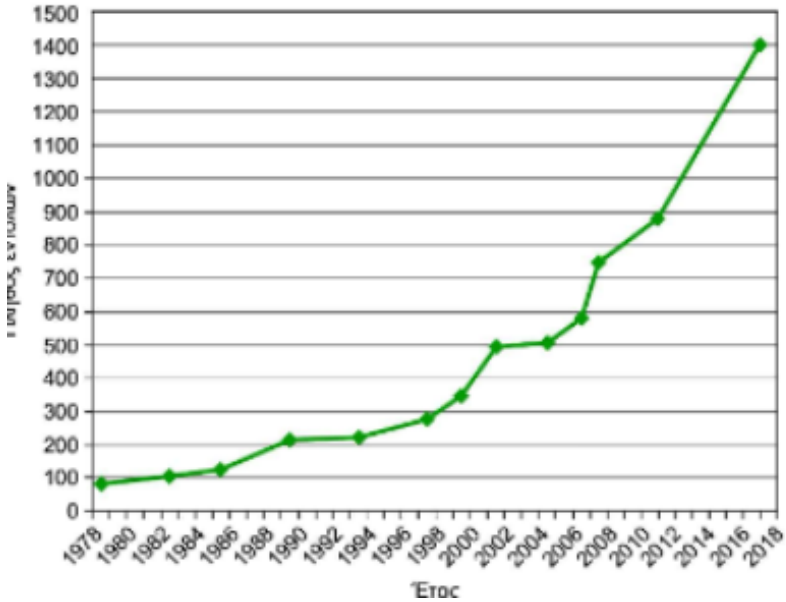
❌ **Πλάνη 2: «Για υψηλή απόδοση, γράφε συμβολική γλώσσα»**

- Ισχύει μόνο σε **σπάνιες περιπτώσεις**
- **Σήμερα** οι μεταγλωττιστές:
  - βελτιστοποιούν καλύτερα
  - εκμεταλλεύονται πιο αποδοτικά τις δυνατότητες των **σύγχρονων επεξεργαστών**

⚠️ **Προσοχή:**

- **Περισσότερες γραμμές κώδικα**:
  - ➕ Περισσότερη πολυπλοκότητα
  - ➖ Περισσότερα σφάλματα
  - ➖ Λιγότερη παραγωγικότητα

ΝΗΖΑΦΕΙΡΗ



Σύνολο εντολών της x86

⚠️ **Παγίδες (Pitfalls)**

❗ **Παγίδα 1: Πρόσβαση σε διαδοχικές λέξεις μνήμης**

- **Λάθος υπόθεση**: Οι διαδοχικές λέξεις δεδομένων βρίσκονται σε διαδοχικές διευθύνσεις μνήμης.
- **Σωστό**:
  ➤ Οι **λέξεις** έχουν μέγεθος **4 bytes**.
  ➤ Άρα οι διευθύνσεις **αυξάνονται κατά 4** κάθε φορά, **όχι κατά 1**.
- ✏️ Παράδειγμα:
  Αν μια λέξη βρίσκεται στη θέση 1000, η επόμενη θα είναι στη θέση **1004**, όχι 1001.

❗ **Παγίδα 2: Χρήση δείκτη μετά την επιστροφή από διαδικασία**

- Συχνό λάθος στη C:
  - Επιστρέφεται **δείκτης σε τοπική (αυτόματη) μεταβλητή** μέσω ορίσματος.
- Τι συμβαίνει;
  ➤ Όταν η διαδικασία επιστρέψει, η στοίβα «καθαρίζεται»
  ➤ Ο δείκτης πλέον **δείχνει σε άκυρη θέση**
- 📌 Ο δείκτης παύει να είναι **έγκυρος**, γιατί η μεταβλητή του **δεν υπάρχει πια στη μνήμη**.

✅ **Συμπερασματικές Παρατηρήσεις**

📌 **Σχεδιαστικές Αρχές**

1. **Η απλότητα ευνοεί την κανονικότητα**
   ➤ Όσο πιο ομοιόμορφες και απλές είναι οι εντολές, τόσο ευκολότερη η υλοποίησή τους.
2. **Το μικρότερο είναι ταχύτερο**
   ➤ Λιγότερα εξαρτήματα, μικρότερες μονάδες = ταχύτερη πρόσβαση & εκτέλεση.
3. **Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς**
   ➤ Πρέπει να εξισορροπηθούν απόδοση, κόστος και πολυπλοκότητα.

📈 **Επιτάχυνση της πιο κοινής περίπτωσης**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Ο σχεδιασμός υλικού και εντολών γίνεται με στόχο την **ταχύτερη εκτέλεση των πιο συνηθισμένων περιπτώσεων**.
- Παράδειγμα: Χρήση **άμεσων τελεστέων**, απλών πράξεων, εντολών με 3 καταχωρητές.

🔁 **Επίπεδα λογισμικού και υλικού**

- Υπάρχουν **πολλά επίπεδα** που συνεργάζονται:
  - **Μεταγλωττιστής** (Compiler)
  - **Συμβολομεταφραστής** (Assembler)
  - **Υλικό** (Hardware)

📚 **Αρχιτεκτονική RISC-V**

- **Αντιπροσωπευτικό παράδειγμα RISC αρχιτεκτονικής**
  - RISC = Reduced Instruction Set Computer (Απλοποιημένο Σύνολο Εντολών)
- 📌 Αντίθεση με **x86**, που είναι CISC (Complex Instruction Set Computer)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

3 ΑΡΙΘΜΗΤΙΚΗ ΓΙΑ ΥΠΟΛΟΓΙΣΤΕΣ

COMPUTER ARCHITECTURE 1

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

1.Αριθμητική για υπολογιστές

Η **αριθμητική για υπολογιστές** είναι η μελέτη των βασικών αριθμητικών συναρτήσεων και των τεχνικών αναπαράστασης και επεξεργασίας των αριθμών στους υπολογιστές. Περιλαμβάνει αρκετές θεμελιώδεις περιοχές:

1. Πράξεις με Ακέραιους Αριθμούς

Οι βασικές αριθμητικές πράξεις περιλαμβάνουν:

- **Πρόσθεση**: Συνδυάζει δύο ή περισσότερους ακεραίους για να παραγάγουνε ένα νέο ακέραιο. Αν το αποτέλεσμα υπερβαίνει το μέγιστο μέγεθος που μπορεί να αναπαρασταθεί, εμφανίζεται **υπερχείλιση**.
- **Αφαίρεση**: Αφαίρεση ενός αριθμού από έναν άλλο. Υπολογίζεται μέσω της πρόσθεσης του αντιθέτου του δεύτερου τελεστή.
- **Πολλαπλασιασμός**: Αυξάνει τον έναν αριθμό με το προϊόν του άλλου. Εδώ, ένα θεμελιώδες θέμα είναι ο υπολογισμός του μήκους του γινομένου, το οποίο είναι το άθροισμα των μηκών των αριθμών.
- **Διαίρεση**: Διαχωρίζει έναν αριθμό σε ίσα μέρη. Ειδικά στις υπολογιστικές εφαρμογές, απαιτείται προσοχή για τον έλεγχο διαίρεσης με το μηδέν, καθώς θα προκαλέσει σφάλμα.

2. Υπερχείλιση

Η υπερχείλιση συμβαίνει όταν το αποτέλεσμα μιας αριθμητικής πράξης δεν μπορεί να αναπαρασταθεί με το διαθέσιμο αριθμητικό σύστημα.

Για παράδειγμα, κατά την πρόσθεση δύο μεγάλων θετικών αριθμών, αν το αποτέλεσμα είναι αρνητικό, αυτό σηματοδοτεί υπερχείλιση (το σύστημα δεν έχει αρκετά bits για να αναπαραστήσει το αποτέλεσμα).

3. Πραγματικοί Αριθμοί Κινητής Υποδιαστολής

Οι πραγματικοί αριθμοί κινητής υποδιαστολής είναι αριθμοί που μπορούν να έχουν ακριβείς αναπαραστάσεις σε διαφορετικά κλίματα μεγέθους. Η αναπαράσταση και οι πράξεις σε πραγματικούς αριθμούς απαιτούν ειδική προσοχή για την ακριβή διαχείριση της ακρίβειας και της στρογγυλοποίησης.

4. Αναπαράσταση και Πράξεις

Η αναπαράσταση ακεραίων και πραγματικών αριθμών στον υπολογιστή επηρεάζει άμεσα την ικανότητα του συστήματος να διενεργεί υπολογισμούς. Η ακριβής αναπαράσταση και οι σωστές τεχνικές για την εκτέλεση αριθμητικών πράξεων είναι κρίσιμες για την αποφυγή σφαλμάτων και την εξασφάλιση της αποτελεσματικότητας στον υπολογισμό.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

2.Πρόσθεση ακεραίων

Η υπερχείλιση είναι ένα σημαντικό φαινόμενο στην αριθμητική με ακέραιους αριθμούς, ιδιαίτερα όταν οι αριθμοί αυτού του τύπου πραγματοποιούν πράξεις που διακυβεύουν την αναπαράσταση τους λόγω περιορισμένου εύρους.

Ας αναλύσουμε καλύτερα τις περιπτώσεις που αναφέρονται:

1. **Πρόσθεση Θετικού και Αρνητικού Τελεστέου**

- Σε αυτή την περίπτωση, όταν προσθέτουμε έναν θετικό και έναν αρνητικό αριθμό, **δεν παρατηρείται υπερχείλιση**.
- Αυτό συμβαίνει γιατί οι δύο αριθμοί τείνουν να εξισορροπούν ο ένας τον άλλον. Για παράδειγμα, στην πρόσθεση του 55 και του −33, το αποτέλεσμα θα είναι 22, το οποίο είναι αποδεκτό από την αναπαράσταση των ακέραιων.

2. **Πρόσθεση Δύο Θετικών Τελεστέων**

- Εδώ, η υπερχείλιση μπορεί να συμβεί εάν το αποτέλεσμα υπερβεί το μέγιστο εύρος της αναπαράστασης των ακεραίων.
- Στην περίπτωση αυτή, **υπερχείλιση παρατηρείται αν το πρόσημο του αποτελέσματος είναι 1** (δηλαδή αν το αποτέλεσμα είναι θετικό).
- Για παράδειγμα, αν προσθέσουμε τους 1,000,0001,000,000 και 1,000,0001,000,000, το αποτέλεσμα είναι 2,000,0002,000,000. Εάν η αναπαράσταση μας μπορεί να αναπαραστήσει μόνο μέχρι 1,500,0001,500,000, θα συμβεί υπερχείλιση.

3. **Πρόσθεση Δύο Αρνητικών Τελεστέων**

- Σε αυτή την περίπτωση, αν προσθέσουμε δύο αρνητικούς αριθμούς, το αποτέλεσμα θα είναι επίσης αρνητικό.
- **Υπερχείλιση παρατηρείται εάν το πρόσημο του αποτελέσματος είναι 0** (δηλαδή εάν το αποτέλεσμα είναι θετικό).
- Για παράδειγμα, αν έχουμε −1,000,000−1,000,000 και −1,000,000−1,000,000, το αποτέλεσμα είναι −2,000,000−2,000,000. Ωστόσο, αν η αναπαράσταση μπορεί να υποστηρίξει μόνο θετικά νούμερα, τότε το αποτέλεσμα αυτό δεν θα μπορεί να αναπαρασταθεί και θα συμβεί υπερχείλιση.

Συμπέρασμα

Η υπερχείλιση είναι μια κρίσιμη έννοια στην υπολογιστική αριθμητική, που συμβαίνει όταν η πραγματική αξία ενός υπολογισμού ξεπερνά τα όρια της υπολογιστικής αναπαράστασης. Οι περιορισμοί στα δυαδικά συστήματα αναπαράστασης απαιτούν

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

να γνωρίζουμε τις προϋποθέσεις που μπορούν να οδηγήσουν σε υπερχείλιση, καθώς αυτό μπορεί να έχει σοβαρές συνέπειες στην εγκυρότητα των υπολογισμών.

3.Ανάλυση της Αφαίρεσης Ακεραίων

Στην υπολογιστική αναπαράσταση ακεραίων, η αφαίρεση μπορεί να θεωρηθεί ως πρόσθεση του αντιθέτου του δεύτερου τελεστέου. Αυτό σημαίνει ότι για να υπολογίσουμε το αποτέλεσμα της αφαίρεσης ενός αριθμού από έναν άλλο, προσθέτουμε τον αρνητικό του δεύτερου αριθμού.

Παράδειγμα

Λαμβάνοντας το παράδειγμα:

7−6=7+(−6)7−6=7+(−6)

Στην δυαδική αναπαράσταση, οι αριθμοί εμφανίζονται ως εξής:

- +7: 0000 0000 ... 0000 0111
- -6: 1111 1111 ... 1111 1010 (Η αναπαράσταση του -6 είναι το συμπλήρωμα του 6.)

Στη συνέχεια, η πρόσθεση του -6 στον +7 θα πραγματοποιηθεί, όπως φαίνεται παρακάτω:

00000000...00000111 (+7)

11111111...11111010 (−6)

00000000...00000001 (+1)

Υπερχείλιση

- **Υπερχείλιση**: Αν το αποτέλεσμα της αφαίρεσης είναι εκτός του εύρους των αναπαραστάσεων των ακεραίων, τότε παρατηρείται υπερχείλιση.
- **Στην περίπτωση της αφαίρεσης**:
  - Όταν αφαιρούμε δύο θετικούς ή δύο αρνητικούς τελεστέους, δεν αναμένεται υπερχείλιση.
  - Όταν αφαιρούμε έναν θετικό από έναν αρνητικό, μπορεί να υπάρχει υπερχείλιση αν το πρόσημο του αποτελέσματος είναι 0 (δηλαδή αν το αποτέλεσμα είναι θετικό).
  - Αν αφαιρούμε έναν αρνητικό από έναν θετικό, τότε μπορεί να υπάρχει υπερχείλιση αν το πρόσημο του αποτελέσματος είναι 1 (δηλαδή αν το αποτέλεσμα είναι αρνητικό).

Συμπέρασμα

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Η αφαίρεση ακεραίων είναι ένα σημαντικό ζήτημα στην αριθμητική των υπολογιστών, καθώς έρχεται σε αντίθεση με την πρόσθεση και έχει πολύπλοκες επιπτώσεις από άποψη υπερχείλισης και αναπαράστασης. Οι κανόνες αυτοί είναι σημαντικοί για την κατανόηση της λειτουργίας των υπολογιστών και των ενεργειών οι οποίες εκτελούνται κατά τη διάρκεια των αριθμητικών υπολογισμών.

4.Αριθμητική για Πολυμέσα

Η αρίθμηση για πολυμέσα αφορά τη διαδικασία και τις τεχνικές που χρησιμοποιούνται για την επεξεργασία γραφικών και άλλων πολυμεσικών δεδομένων, όπως ήχου και βίντεο. Ακολουθούν οι βασικές πτυχές που σχετίζονται με την αριθμητική αυτή, όπως τις περιγράφει το κείμενο:

- **Διανύσματα Δεδομένων**:
  - Τα πολυμέσα συχνά επεξεργάζονται σε διανύσματα δεδομένων των 8 bit και 16 bit. Αυτά τα διανύσματα αποτελούνται από σειρά ψηφίων (bits) που ανάγουν σε ψηφιακή αναπαράσταση πληροφοριών όπως εικόνες και ήχοι.
- **Χρήση Αθροιστών 64 bit**:
  - Οι αθροιστές 64 bit χρησιμοποιούνται για την εκτέλεση προσθέσεων ή άλλων μαθηματικών πράξεων. Η διαχείριση αυτών των αλυσίδων εξασφαλίζει ότι οι πράξεις εκτελούνται γρήγορα και αποδοτικά, κυρίως όταν η ποσότητα δεδομένων είναι μεγάλη.
- **Πράξεις σε Διανύσματα**:
  - Οι πράξεις μπορούν να γίνονται σε διανύσματα 8×8 bit (64 bit), 4×16 bit (64 bit) ή 2×32 bit (64 bit). Αυτός ο πολλαπλασιασμός δεδομένων ταυτόχρονα βελτιώνει την απόδοση και επιταχύνει την επεξεργασία.
- **SIMD (Single Instruction, Multiple Data)**:
  - Ο όρος SIMD αναφέρεται σε μια τεχνική που επιτρέπει την εκτέλεση μιας εντολής σε πολλά δεδομένα ταυτόχρονα. Αυτό επιτρέπει την αποτελεσματική εκτέλεση υπολογισμών, μειώνοντας τον χρόνο εκτέλεσης των πολυμεσικών εφαρμογών.
- **Λειτουργίες Κορεσμού**:
  - Στην αριθμητική, σε καταστάσεις υπερχείλισης, το αποτέλεσμα είναι η μεγαλύτερη τιμή που μπορεί να αναπαρασταθεί. Η κορεσμός συμβαίνει όταν οι τιμές που υπολογίζονται ξεπερνούν το δυνατό εύρος αναπαράστασης (overflow), και το αποτέλεσμα "κόβεται" ή ρυθμίζεται στην υψηλότερη αποδεκτή τιμή.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Αριθμητική Modulo Συμπληρώματος ως προς Δύο**:
    - Η αναφορά στην αριθμητική modulo συμπληρώματος ως προς δύο σχετίζεται με τον τρόπο που η υπερχείλιση αντιμετωπίζεται μέσω αναπαράστασης, επιτρέποντας συγκεκριμένες πράξεις για την αποφυγή λαθών στην αριθμητική.
- **Πρακτικά Παραδείγματα**:
    - Ένα παράδειγμα της χρήσης της αριθμητικής για πολυμέσα είναι η περικοπή ηχητικών αποσπασμάτων και ο κορεσμός σε βίντεο, όπου είναι ζωτικής σημασίας η σωστή και γρήγορη εκτέλεση υπολογισμών για την ποιότητα και την απόδοση των πολυμεσικών εφαρμογών.

Σύνοψη

Η αριθμητική για πολυμέσα παίζει κρίσιμο ρόλο στην επεξεργασία και απόδοση γραφικών και ήχου, χρησιμοποιώντας βελτιστοποιημένες τεχνικές που βασίζονται σε διανύσματα και SIMD, διασφαλίζοντας την αποτελεσματικότητα και στήριξη στους απαιτητικούς υπολογισμούς που σχετίζονται με αυτά τα δεδομένα.

5.Πολλαπλασιασμός

Διαδικασία Πολλαπλασιασμού:

Στην παραδοσιακή μέθοδο του πολλαπλασιασμού, οι αριθμοί τοποθετούνται ο ένας κάτω από τον άλλον. Για παράδειγμα:

1000 (Πολλαπλασιαστέος)

× 1001 (Πολλαπλασιαστής)

1001000 (Γινόμενο)

Όμως, στον κόσμο των υπολογιστών, ο πολλαπλασιασμός δεν πραγματοποιείται με αυτόν τον τρόπο. Για παράδειγμα , έστω ότι έχουμε τους παρακάτω αριθμούς.

Πολλαπλασιαστής : 0101 (5)

Πολλαπλασιαστέος: 0011(3)

1η επανάληψη

Βήμα 1

Αρχικοποιούμε το γινόμενο στο 0.

Βήμα 2

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Ελέγχουμε το LSB του πολλαπλασιαστή. Αν είναι 1 , τότε προσθέτουμε στο γινόμενο τον πολλαπλασιαστέο. Αν είναι 0 δεν κάνουμε τίποτα και συνεχίζουμε στο βήμα 3. Στο παράδειγμά μας, αφού το LSB του πολλαπλασιαστή είναι 1(0011) , τότε έχουμε : γινόμενο = γινόμενο + πολλαπλασιαστέος ⇔ γινόμενο = 0 + 0011 = 0011(5)

Βήμα 3

Πραγματοποιούμε ολίσθηση **προς τα αριστερά** του πολλαπλασιαστέου :

0011 -> ~~0~~0110 -> 0110

Βήμα 4

Πραγματοποιούμε ολίσθηση **προς τα δεξιά** του πολλαπλασιαστή :

0101 -> 00101 ~~1 ->~~ 0010

2η επανάληψη

Ξαναπάμε στο βήμα 2 και ελέγχουμε το LSB του πολλαπλασιαστή. Είναι 0 . Αν είναι 0, δεν προσθέτουμε τον πολλαπλασιαστέο στο γινόμενο, και συνεχίζουμε απλώς με τις ολισθήσεις μας.

Ολίσθηση του πολλαπλασιαστέου **προς τα αριστερά** : 0110 -> 1100

Ολίσθηση του πολλαπλασιαστή **προς τα δεξιά**: 0010 -> 000**1** LSB

3η επανάληψη

Ξαναπάμε στο βήμα 2 και ελέγχουμε το LSB του καινούργιου πολλαπλασιαστή. Είναι 1. Άρα προσθέτουμε στο γινόμενο τον πολλαπλασιαστέο. Έτσι έχουμε :

Γινόμενο = γινόμενο + πολλαπλασιαστέος ⇔ Γινόμενο = 0011 + 1100 = 1111

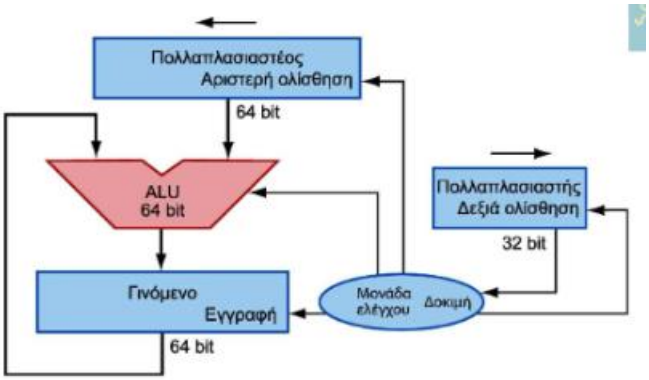
Ολίσθηση του πολλαπλασιαστέου προς τα αριστερά : 1100 -> 1000

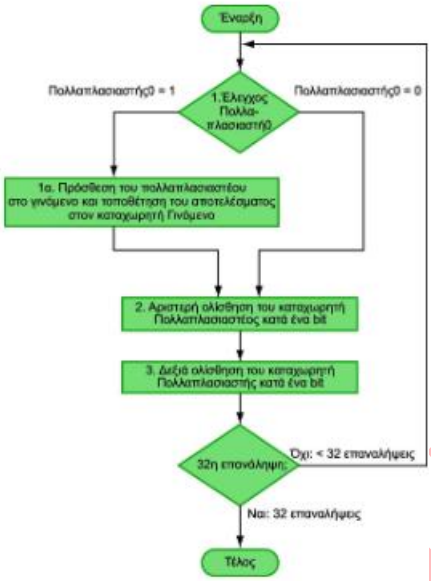
Ολίσθηση πολλαπλασιαστή δεξιά : 0001 -> 0000

Όταν ο πολλαπλασιαστής γίνει όλος με μηδενικά, τότε σταματάμε , και το τελικό γινόμενο είναι το γινόμενο που είχαμε σχηματίσει μέχρι στιγμής. Πράγματι , 3 * 5 = 15 και 1111 = 15 στο δυαδικό σύστημα.

Παρακάτω φαίνεται το διάγραμμα ροής αυτής της διαδικασίας καθώς και μια εικόνα της από τα κυκλώματα του υπολογιστή.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ





ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Ο παραπάνω προγραμματισμένος τρόπος εκτέλεσης του πολλαπλασιασμού οδηγεί στο τελικό γινόμενο, το οποίο είναι το άθροισμα των επιμέρους γινομένων που δημιουργήθηκαν κατά τη διάρκεια της διαδικασίας .Ουσιαστικά, **το μήκος του γινομένου είναι το άθροισμα των μηκών των αρχικών αριθμών.**

Για παράδειγμα, αν πολλαπλασιάσουμε έναν αριθμό 3 ψηφίων με έναν αριθμό 4 ψηφίων, το αποτέλεσμα μπορεί να έχει μέχρι 7 ψηφία.

Βελτιστοποιημένος πολλαπλασιασμός

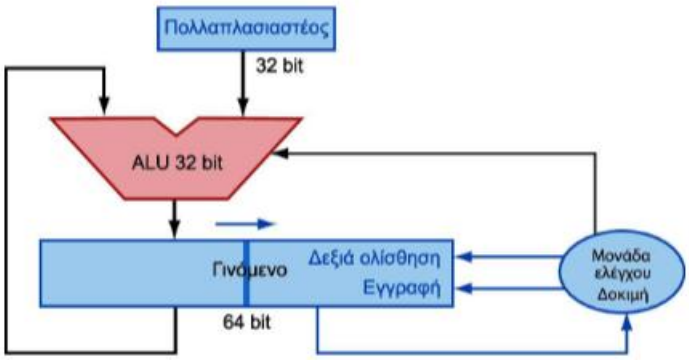
Ο βελτιστοποιημένος πολλαπλασιασμός αφορά τη διαδικασία εκτέλεσης πράξεων πολλαπλασιασμού με μεγαλύτερη αποδοτικότητα, που είναι ιδιαίτερα σημαντική για υπολογιστές και ψηφιακά συστήματα.

Βασικά Στοιχεία:

- **Παράλληλη Εκτέλεση Βημάτων:**

- Η διαδικασία πολλαπλασιασμού μπορεί να διαχωριστεί σε διαφορετικά βήματα, όπως πρόσθεση και ολίσθηση (shift).
- Κατά τη διάρκεια του πολλαπλασιασμού, αντί να περιμένουμε την ολοκλήρωση κάθε ξεχωριστής πράξης, οι υπολογιστές μπορούν να εκτελούν ταυτόχρονα κάποια από αυτά τα βήματα. Αυτή η παράλληλη εκτέλεση μειώνει το συνολικό χρόνο που απαιτείται για να ολοκληρωθεί η πράξη.

- **Ένας Κύκλος για την Πρόσθεση Κάθε Μερικού Γινομένου:**
  - Σε κάθε κύκλο ρολογιού, ο υπολογιστής ολοκληρώνει την πρόσθεση ενός μερικού γινομένου. Τα μερικά γινόμενα είναι τα ενδιάμεσα αποτελέσματα που προκύπτουν από την πολλαπλασιαστική διαδικασία.
  - Αυτό σημαίνει ότι ο υπολογιστής δεν χρειάζεται να περιμένει όλες τις προσθέσεις να ολοκληρωθούν πριν προχωρήσει στο επόμενο βήμα του πολλαπλασιασμού.
- **Αποδεκτό Αν Γίνονται Λίγες Πράξεις Πολλαπλασιασμού:**
  - Ο βελτιστοποιημένος πολλαπλασιασμός είναι αποδεκτός, ειδικά σε σενάρια όπου οι πράξεις πολλαπλασιασμού είναι περιορισμένες και όχι πολύ ακριβές.
  - Αυτό σημαίνει ότι αν οι πολλαπλασιασμοί είναι σπάνιοι ή λιγότερο απαιτητικοί από άποψη υπολογιστικής ισχύος, τότε η βελτιστοποίηση αυτών των διαδικασιών μπορεί να έχει καλύτερα αποτελέσματα σε σχέση με τις παραδοσιακές μεθόδους.



Ταχύτερος Πολλαπλασιασμός

Ο ταχύτερος πολλαπλασιασμός αναφέρεται σε μεθόδους που αποσκοπούν στη βελτίωση της ταχύτητας εκτέλεσης των πράξεων πολλαπλασιασμού, ειδικότερα στην υπολογιστική αρχιτεκτονική. Ακολουθούν οι βασικές πτυχές αυτού του concept:

ΕΝΗΖΑΦΕΙΡΗ

Χρήση Περισσότερων του Ένος Αθροιστών

Αντί να χρησιμοποιείται ένας μόνο αθροιστής για τις πράξεις, η ιδέα είναι να χρησιμοποιηθούν **πολλοί αθροιστές** ταυτόχρονα. Με αυτόν τον τρόπο, μπορεί να επιταχυνθεί η διαδικασία στην οποία οι μερικοί πολλαπλασιασμοί ή αθροίσματα υπολογίζονται. Έτσι, ο συνολικός χρόνος εκτέλεσης μειώνεται.

Συμβιβασμός Ανάμεσα σε Κόστος και Απόδοση

Η χρήση περισσότερων αθροιστών συνεπάγεται ένα **κόστος** για το σχεδιασμό και την κατασκευή επιπλέον υλικού. Αυτή η προσθήκη μπορεί να οδηγήσει σε αύξηση του κόστους των κυκλωμάτων ή της πολυπλοκότητάς τους. Παρ' όλα αυτά, η **απόδοση** μπορεί να βελτιωθεί σημαντικά, και οι σχεδιαστές πρέπει να βρουν μία **ισορροπία** μεταξύ του κόστους και της επιθυμητής απόδοσης του συστήματος.

Υλοποίηση και με Διοχέτευση

Η **διοχέτευση** είναι μια τεχνική στα κυκλώματα υπολογιστών που επιτρέπει τη διαίρεση μιας διαδικασίας σε μικρότερες ανεξάρτητες εργασίες, οι οποίες μπορούν να εκτελούνται παράλληλα. Έτσι, η υλοποίηση του ταχύτερου πολλαπλασιασμού μπορεί επίσης να γίνει μέσω των τεχνικών διοχέτευσης, που ενισχύει την αποδοτικότητα και την ταχύτητα των υπολογισμών.

Παράλληλη Εκτέλεση Πολλών Πράξεων Πολλαπλασιασμού

Η χρήση **παραλληλίας** επιτρέπει τη ταυτόχρονη εκτέλεση πολλών πράξεων πολλαπλασιασμού. Καθώς σε πολλές εφαρμογές, όπως η επεξεργασία πολυμέσων ή η επιστήμη των υπολογιστών, χρειάζονται οι ταυτόχρονες υπολογιστικές διεργασίες, αυτή η προσέγγιση υπηρέτησε στην ταχύτητα των υπολογισμών, ειδικά στα γραφικά και στην αλγοριθμική επεξεργασία.

Πολλαπλασιασμός RISC-V

Η αρχιτεκτονική RISC-V έχει υλοποιήσει τέσσερις εξειδικευμένες εντολές για τον πολλαπλασιασμό που επιτρέπουν τη διαχείριση των σχετικών χαρακτηριστικών των αριθμών που πολλαπλασιάζονται, όπως η υπογραφή τους.

Αναλυτική Εξήγηση των Εντολών

- **mul (multiply):**
    - Αυτή η εντολή πολλαπλασιάζει δύο προσημασμένους ακέραιους αριθμούς και επιστρέφει **τα 32 χαμηλότερα bit του γινομένου**. Η χρήση αυτής της εντολής είναι η πιο απλή περίπτωση, αλλά δεν παρέχει το πλήρες γινόμενο, το οποίο μπορεί να είναι μεγαλύτερο από 32 bit.
- **mulh (multiply high):**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Αυτή η εντολή επιστρέφει **τα 32 υψηλότερα bit του γινομένου** όταν οι δύο τελεστέοι είναι προσημασμένοι. Είναι χρήσιμη όταν η ανάγκη είναι να ελέγξει κανείς μόνο το ανώτερο τμήμα του γινομένου, που συχνά χρειάζεται για να διαπιστωθεί αν έχει γίνει υπερχείλιση.

- **mulhu (multiply high unsigned):**
  - Εδώ, η εντολή επιστρέφει επίσης τα 32 υψηλότερα bit του γινομένου, αλλά για τα μη προσημασμένα (unsigned) νούμερα. Αυτή η εντολή είναι σημαντική όταν δεν χρειάζεται να ανησυχεί κανείς για τα πρόσημα των αριθμών κατά τη διάρκεια της πολλαπλασιαστικής διαδικασίας.

- **mulhsu (multiply high signed/unsigned):**
  - Αυτή η εντολή επιστρέφει τα 32 υψηλότερα bit του γινομένου, με τον ένα τελεστέο να είναι προσημασμένος και τον άλλο μη προσημασμένος. Αυτή η ευελιξία είναι χρήσιμη για διάφορες περιπτώσεις εργασιών όπου μπορεί να συμβαίνει αυτός ο συνδυασμός.

Επαλήθευση Υπερχείλισης

- **Χρήση του αποτελέσματος της mulh:**
  - Η RISC-V σχεδίασε τη χρήση του αποτελέσματος από την εντολή mulh για να ελέγξει <u>αν υπάρχει υπερχείλιση κατά τον πολλαπλασιασμό των 32 bit</u>. Αν τυχόν το αποτέλεσμα της mulh υποδεικνύει ότι τα 32 υψηλότερα bit έχουν τιμή που ενδέχεται να δημιουργήσει πρόβλημα, τότε ακολουθούν περαιτέρω χειρισμοί στην εκτέλεση.

<u>Σημασία</u>

Η διαφορετική επιλογή εντολών και η δύναμη ελέγχου της υπερχείλισης είναι κρίσιμη για την αποδοτικότητα των υπολογισμών στους υπολογιστές, αφού επιτρέπει στους προγραμματιστές να ελέγχουν τη διαχείριση των αριθμών με βάση τις ειδικές ανάγκες της εφαρμογής τους.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Παράδειγμα Κώδικα RISC-V

Edit | Execute

multiply.asm*

```
.text
.global main
main:
    # Φόρτωση των αριθμών στους καταχωρητές
    lw      t0, num1                # t0 = num1
    lw      t1, num2                # t1 = num2

    # Πολλαπλασιασμός (χαμηλό γινόμενο)
    mul     t2, t0, t1              # t2 = t0 * t1 (χαμηλό γινόμενο)

    # Υπολογισμός του υψηλού γινομένου με πρόσημο
    mulh    t3, t0, t1              # t3 = υψηλό γινόμενο (προσημασμένο)

    # Υπολογισμός του υψηλού γινομένου χωρίς πρόσημο
    mulhu   t4, t0, t1              # t4 = υψηλό γινόμενο (χωρίς πρόσημο)

    # Αποθήκευση των αποτελεσμάτων στη μνήμη
    la      t5, result_low
    sw      t2, 0(t5)

    la      t5, result_high
    sw      t3, 0(t5)

    la      t5, result_high_unsigned
    sw      t4, 0(t5)

    # Τερματισμός του προγράμματος
    li      a0, 10                  # Κωδικός τερματισμού
    ecall
.data
    num1:   .word  15                # 0 πρώτος αριθμός
    num2:   .word -3                # 0 δεύτερος αριθμός
    result_low:            .word 0            # Χαμηλό γινόμενο (32 LSB bits)
    result_high:           .word 0            # Υψηλό γινόμενο (32 MSB bits - signed)
    result_high_unsigned:  .word 0            # Υψηλό γινόμενο (unsigned)
```

6.Διαίρεση σε hardware

Η διαδικασία διαίρεσης ακεραίων είναι αρκετά σύνθετη, ειδικά όταν πρόκειται για την υλοποίηση της σε επίπεδο υλικού (hardware). Ακολουθεί μια λεπτομερής ανάλυση των βημάτων που σχετίζονται με τη διαίρεση αριθμών.

Έστω ότι έχουμε τους αριθμούς :

Διαιρετέος : 1101 (13)

Διαιρέτης: 0011 (3)

Βήμα 1.

Έλεγχος για Διαίρεση με το Μηδέν. Πριν ξεκινήσει η διαίρεση, είναι απαραίτητο να ελέγξει κανείς αν ο διαιρέτης είναι μηδέν. Η διαίρεση με το μηδέν είναι αδύνατη και οδηγεί σε σφάλμα. Στο παράδειγμά μας, ο διαιρέτης είναι διάφορος του 0 άρα μπορούμε να συνεχίσουμε κανονικά.

Βήμα 2

Θέτουμε το υπόλοιπο ίσο με 0. Υπόλοιπο = 0000 (R)

Βήμα 3

Θέτουμε το πηλίκο ίσο με τον διαιρετέο. Πηλίκο = 1101 (Q)

Βήμα 4

Γράφουμε Δίπλα δίπλα το υπόλοιπο και το πηλίκο -> R : Q = 0000 1101

Βήμα 5

Πραγματοποιούμε ολίσθηση προς τα αριστερά : 00001101 -> 0001 1010

Άρα το νέο υπόλοιπο είναι R = 0001 και το νέο πηλίκο Q = 1010 LSB

Βήμα 5

Αφαιρούμε τον διαιρέτη από το υπόλοιπο : υπόλοιπο = υπόλοιπο – διαιρέτης ⇔ υπόλοιπο = 0001 – 0011 = 0001 + (-0011) = 0001 + 1101 (συμπλήρωμα ως προς 2) = 1110 (ξεκινάει με 1 άρα το αποτέλεσμα είναι αρνητικό)

Επειδή είναι αρνητικό, επαναφέρουμε το υπόλοιπο όπως ήταν πριν την αφαίρεση, και κάνουμε το LSB του πηλίκου 0. Δηλαδή έχουμε :

R = 0001 Q = 1010 (Το LSB είναι ήδη 0 άρα δεν αλλάζουμε κάτι)

Ξαναπάμε στο βήμα 4. Και έχουμε R : Q = 0001 1010

Μετά από ολίσθηση προς τα αριστερά προκύπτει : 0011 0100

Άρα το νέο υπόλοιπο είναι 0011 και το νέο πηλίκο είναι 0100

Κάνουμε την αφαίρεση μας , υπόλοιπο = υπόλοιπο – διαιρέτης ⇔ υπόλοιπο = 0011 – 0011 = 0000 (θετικό άρα δεν επαναφέρουμε το προηγούμενο υπόλοιπο)

Άρα το υπόλοιπο είναι 0000 και το πηλίκο 0101

Ξαναπάμε στο βήμα 4. Και έχουμε R : Q = 0000 0101

Μετά από ολίσθηση προς τα αριστερά έχουμε : 0000 1010

Άρα υπόλοιπο = 0000 και πηλίκο = 1010

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Υπόλοιπο = υπόλοιπο – διαιρέτης ⇔ υπόλοιπο = 0000 – 0011 = 1101 (αρνητικό)

Άρα το υπόλοιπο γίνεται R = 0000 (δηλαδή το προηγοηγούμενο, απλώς τώρα τυχαίνει ναναι τα ίδια) και το LSB του πηλίκου (Q) γίνεται 0. Άρα το νέο πηλίκο είναι 1010 (δεν άλλαξε)

Ξαναπάμε στο βήμα 4. Και έχουμε R : Q = 0000 1010

Μετά από ολίσθηση προς τα αριστερά έχουμε : 0001 0100

Άρα υπόλοιπο = 0001 και πηλίκο = 0100

Υπόλοιπο = υπόλοιπο – διαιρέτης ⇔ υπόλοιπο = 0001 – 0011 = 1110 (αρνητικό)

Άρα υπόλοιπο = 0001 και πηλίκο = 0100

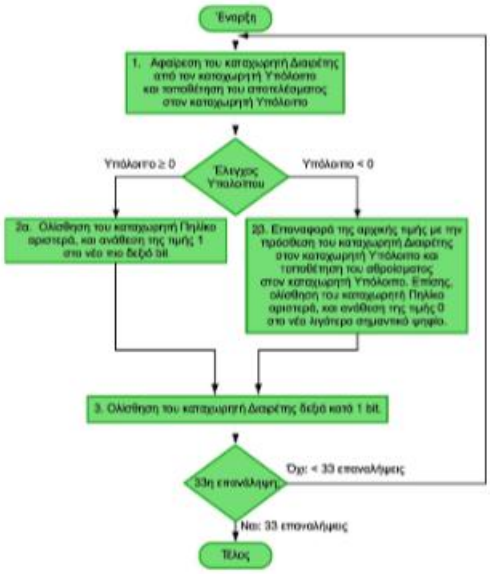
Σταματάμε στις 4 επαναλήψεις ! (επειδή ο διαιρετέος είναι 4 bit)

Q = 4

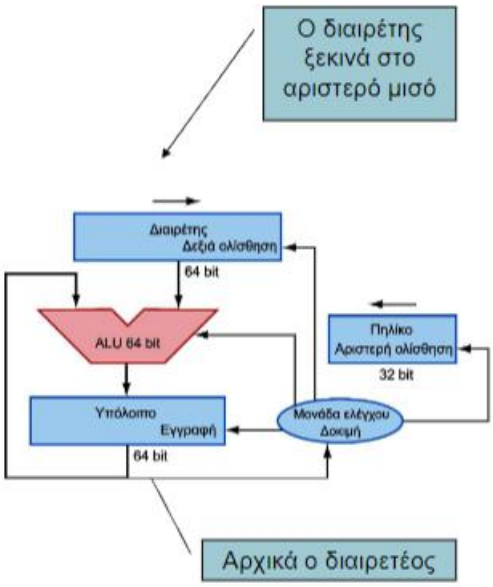
R = 1

Συμπέρασμα

Η διαδικασία διαίρεσης ακεραίων, ειδικά σε επίπεδο υλικού, μπορεί να είναι περίπλοκη και απαιτεί τη σωστή διαχείριση προσημάτων και την παρακολούθηση των υπολοίπων κατά τη διάρκεια της διαδικασίας. Οι παραπάνω αρχές είναι θεμελιώδεις για την κατανόηση των αλγορίθμων διαίρεσης και την εκτέλεση τους.



ΣΗΛΖΑΦΕΙΡΗ



Βελτιστοποιημένη Διαίρεση

Η βελτιστοποιημένη διαίρεση αναφέρεται σε τεχνικές που επιταχύνουν τη διαδικασία διαίρεσης, μειώνοντας τον αριθμό των βημάτων που απαιτούνται. Ειδικότερα:

- **Ένας κύκλος για την αφαίρεση κάθε μερικού υπολοίπου:** Αυτό σημαίνει ότι σε κάθε βήμα της διαδικασίας διαίρεσης, η υπολογιστική μονάδα αφαιρεί το αντίστοιχο μερικό υπόλοιπο από τον διαιρετέο. Η διαδικασία αυτή είναι οργανωμένη έτσι ώστε να διαρκεί μόνο ένα κύκλο ρολογιού για το κάθε μερικό υπόλοιπο.
- **Μοιάζει πολύ με πολλαπλασιασμό:** Η προσέγγιση για την υλοποίηση της διαίρεσης είναι συγγενική με αυτή του πολλαπλασιασμού, υποδηλώνοντας ότι οι ίδιες υπολογιστικές αρχές και τεχνικές μπορούν να εφαρμοστούν. Αυτό είναι χρήσιμο, καθώς επιτρέπει την εξοικονόμηση χρόνου και πόρων.
- **Μπορεί να χρησιμοποιηθεί το ίδιο υλικό τόσο για πολλαπλασιασμό όσο και για διαίρεση:** Αυτό σημαίνει ότι η ίδια αρχιτεκτονική ή κυκλώματα που χρησιμοποιούνται για τη διαδικασία του πολλαπλασιασμού μπορούν να προσαρμοστούν ή να επαναχρησιμοποιηθούν και για τη διαδικασία της διαίρεσης, πάνω σε συνθήκες και υπολογιστικές απαιτήσεις. Αυτή η επαναχρησιμοποίηση του υλικού μειώνει το κόστος και την πολυπλοκότητα του σχεδιασμού των υπολογιστών.

Ταχύτερη Διαίρεση

Η πιο γρήγορη εκτέλεση διαίρεσης περιλαμβάνει αρκετές σημαντικές διαδικασίες:

- **Δεν μπορεί να χρησιμοποιηθεί υλικό παράλληλης εκτέλεσης όπως στον πολλαπλασιασμό:** Σε αντίθεση με τον πολλαπλασιασμό, where διαφορετικές

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

υπολογιστικές μονάδες μπορούν να εκτελέσουν παράλληλες πράξεις, οι διαδικασίες διαίρεσης περιορίζονται έτσι ώστε να μην χρησιμοποιούν παράλληλο υλικό, καθιστώντας τις λιγότερο αποδοτικές.

- **Η αφαίρεση εξαρτάται από το πρόσημο του υπολοίπου:** Όταν εκτελούνται διαίρεσεις, η φύση του υπολοίπου (θετικό ή αρνητικό) επηρεάζει τη διαδικασία. Αυτό σημαίνει ότι οι υπολογιστικές ροές πρέπει να προσαρμοστούν με βάση το πρόσημο, οδηγώντας σε πρόσθετη πολυπλοκότητα.
- **Στις ταχύτερες τεχνικές διαίρεσης (π.χ. διαίρεση SRT) παράγονται περισσότερα από ένα bit του πηλίκου σε κάθε βήμα:** Οι γρηγορότερες μέθοδοι διαίρεσης, όπως η SRT (Sweeney, Robertson, Tocher), επιτρέπουν την παραγωγή περισσότερων από ένα bitπληροφορία για το πηλίκο κάθε φορά που εκτελείται ένα βήμα. Αυτό μπορεί να επιταχύνει τη διαδικασία αλλά εξακολουθεί να απαιτεί περισσότερα από ένα βήματα για να ολοκληρωθεί η διαίρεση.

Διαίρεση RISC-V

Στην αρχιτεκτονική RISC-V, η διαδικασία διαίρεσης περιλαμβάνει συγκεκριμένες εντολές που διαχειρίζονται τις διάφορες κατηγορίες ακεραίων.

- **Τέσσερις εντολές:** Η RISC-V ορίζει τις εξής τέσσερις εντολές διαίρεσης:
    - div: διαίρεση προσημασμένων ακεραίων με υπόλοιπο.
    - rem: επιστροφή του υπολοίπου από τη διαίρεση προσημασμένων ακεραίων.
    - divu: διαίρεση μη προσημασμένων ακεραίων με υπόλοιπο.
    - remu: επιστροφή του υπολοίπου από τη διαίρεση μη προσημασμένων ακεραίων.
- **Η υπερχείλιση και η διαίρεση με το μηδέν δεν προκαλούν σφάλμα:** Αυτό σημαίνει ότι οι σχετικές εντολές έχουν σχεδιαστεί ώστε να διαχειρίζονται καλύτερα την περίπτωση διαίρεσης με το μηδέν και την υπερχείλιση, επιστρέφοντας καθορισμένα αποτελέσματα αντί να προκαλούν σφάλματα, γεγονός που συμβάλλει στη σταθερότητα και την αξιοπιστία των προγραμμάτων.
- **Ταχύτερη εκτέλεση για την πιο κοινή περίπτωση της διαίρεσης χωρίς σφάλμα:** Η RISC-V επιδιώκει να επιταχύνει τις διαδικασίες διαίρεσης, ειδικά για περισσότερες συνηθισμένες περιπτώσεις - καθιστώντας τη διαδικασία της διαίρεσης πιο αποτελεσματική για τις περισσότερες εφαρμογές.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Αυτή η κατασκευή προσφέρει εξαιρετική ευελιξία και απόδοση στη RISC-V αρχιτεκτονική διαχείρισης αριθμητικών υπολογισμών.

RISC-V Assembly Κώδικας

Edit | Execute

multiply.asm* | διαιρεση.asm*

```
.text
    .globl main
main:
    # Προσημασμένη διαίρεση
    lw t0, a_signed          # t0 = a
    lw t1, b_signed          # t1 = b
    div t2, t0, t1           # t2 = a_signed / b_signed
    rem t3, t0, t1           # t3 = a_signed % b_signed

    la t4, result_signed     # Φόρτωσε τη διεύθυνση της μεταβλητής
    sw t2, 0(t4)             # Αποθήκευσε το πηλίκο
    la t4, remainder_signed
    sw t3, 0(t4)             # Αποθήκευσε το υπόλοιπο
    # Μη προσημασμένη διαίρεση
    lw t5, a_unsigned        # t5 = a_unsigned
    lw t6, b_unsigned        # t6 = b_unsigned
    divu t0, t5, t6          # t7 = a_unsigned / b_unsigned
    remu t1, t5, t6          # t8 = a_unsigned % b_unsigned

    la t2, result_unsigned
    sw t3, 0(t2)             # Αποθήκευσε το πηλίκο
    la t4, remainder_unsigned
    sw t5, 0(t2)             # Αποθήκευσε το υπόλοιπο
    # Τερματισμός του προγράμματος
    li a7, 10
    ecall
.data
    a_signed:           .word -10     # Προσημασμένος ακέραιος
    b_signed:           .word 3
    a_unsigned:         .word 14      # Μη προσημασμένος ακέραιος
    b_unsigned:         .word 5
    result_signed:      .word 0       # Αποθήκευση του πηλίκου (signed)
    remainder_signed:   .word 0       # Αποθήκευση υπολοίπου (signed)
    result_unsigned:    .word 0       # Αποθήκευση του πηλίκου (unsigned)
    remainder_unsigned: .word 0       # Αποθήκευση υπολοίπου (unsigned)
```

7.Κινητή Υποδιαστολή

Η έννοια της κινητής υποδιαστολής αφορά την αναπαράσταση αριθμών που περιέχουν κλασματικά μέρη, καλύπτοντας έτσι ένα ευρύ φάσμα, από πολύ μικρούς έως πολύ μεγάλους αριθμούς. Αυτή η αναπαράσταση είναι ζωτικής σημασίας στην υπολογιστική, καθώς επιτρέπει στους υπολογιστές να χειρίζονται αριθμούς που ξεπερνούν τα όρια των σταθερών δεδομένων, όπως οι ακέραιοι.

Στοιχεία της Κινητής Υποδιαστολής

- **Επιστημονική Σημειογραφία**:
  - Οι αριθμοί στην κινητή υποδιαστολή συνήθως αναπαρίστανται με έναν τρόπο που μοιάζει με την επιστημονική σημειογραφία. Για παράδειγμα:
    - -2.34×10^{56}
    - $+0.002 \times 10^{-4}$
    - $+987.02 \times 10^{9}$
- **Δυαδική Αναπαράσταση**:
  - Στην υπολογιστική, η κινητή υποδιαστολή εκφράζεται σε δυαδική μορφή ως: $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
  - Αυτό σημαίνει ότι ο αριθμός αποτελείται από ένα σημαντικό μέρος (mantissa) και έναν εκθέτη (exponent).
- **Κανονικοποιημένοι και Μη Κανονικοποιημένοι Αριθμοί**:
  - **Κανονικοποιημένοι Αριθμοί**: Αυτοί είναι οι αριθμοί που βρίσκονται στη "σωστή" μορφή, όπου το σημαντικό μέρος είναι κανονικοποιημένο (δηλαδή σταθερά μεταξύ 1.0 και 2.0 για την δυαδική αναπαράσταση).
  - **Μη Κανονικοποιημένοι Αριθμοί**: Αυτοί οι αριθμοί μπορεί να έχουν πολύ μικρές ή πολύ μεγάλες τιμές, με το δεξιά δυαδικό κλάσμα να είναι καλυμμένο με μηδενικά (π.χ. 0.000...10.000...1) ή να μην εισάγουν το "1" στο σημαντικό μέρος.

Τύποι Γλώσσας C

Στη γλώσσα προγραμματισμού C, οι τύποι κινητής υποδιαστολής περιλαμβάνουν τον τύπο float (για απλή ακρίβεια) και τον τύπο double (για διπλή ακρίβεια). Αυτό σημαίνει ότι μπορούμε να αναπαραστήσουμε αριθμούς με μεγαλύτερη ακρίβεια χρησιμοποιώντας περισσότερα bit, φέρνοντας έτσι μεγαλύτερες δυνατότητες υπολογισμού.

Το Πρότυπο της Κινητής Υποδιαστολής

Το πρότυπο κινητής υποδιαστολής IEEE 754-1985 καθορίζει τους κανόνες και τη διαδικασία αναπαράστασης αριθμών κινητής υποδιαστολής στους υπολογιστές. Η σημαντικότητα αυτού του προτύπου έγκειται στις παρακάτω πτυχές:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

1. **Καθολικότητα και Ενοποίηση**

Το πρότυπο αναπτύχθηκε για να ενοποιήσει τις διαφορετικές μεθόδους αναπαράστασης αριθμών κινητής υποδιαστολής που υπήρχαν πριν από αυτό, επιλύοντας προβλήματα φορητότητας κώδικα κατά την εκτέλεση επιστημονικών εφαρμογών σε διαφορετικά συστήματα.

2. **Αναπαράσταση αριθμών**

- **Υψηλή ακρίβεια**: Ορίζει δύο ποιότητες αναπαράστασης:
  - **Απλής ακρίβειας (32 bit)**: Χρησιμοποιεί 32 bits για την αναπαράσταση ενός αριθμού κινητής υποδιαστολής, το οποίο περιλαμβάνει ένα bit για το πρόσημο, 8 bits για τον εκθέτη, και 23 bits για το κλάσμα.
  - **Διπλής ακρίβειας (64 bit)**: Εδώ χρησιμοποιούνται 64 bits για την αναπαράσταση, παρέχοντας μεγαλύτερη ακρίβεια και μεγαλύτερο εύρος, με 1 bit για το πρόσημο, 11 bits για τον εκθέτη και 52 bits για το κλάσμα.

Μορφή Κινητής Υποδιαστολής κατά IEEE

Η μορφή κινητής υποδιαστολής σύμφωνα με το πρότυπο IEEE 754 καθορίζει πώς αναπαρίστανται οι αριθμοί με κλασματικά μέρη σε υπολογιστές. Ακολουθούν οι βασικές πτυχές αυτού του προτύπου:

1. Σημασία Bits

- **S (bit προσήμου)**:
  - Το πρώτο bit στην αναπαράσταση:
    - 0 σημαίνει μη αρνητικό αριθμό
    - 1 σημαίνει αρνητικό αριθμό.

2. Κανονικοποίηση Σημαντικού

- **Σημαντικό (Mantissa)**:
  - Οι αριθμοί κινητής υποδιαστολής κανονικοποιούνται ώστε να διασφαλίζεται ότι το σημαντικό βρίσκεται στην κλίμακα μεταξύ 1.0 και 2.0
  - δηλαδή: 1.0≤|σημαντικό΄|<2.0
  - Το αρχικό 1 bit του σημαντικού υπονοείται και δεν αποθηκεύεται ρητά. Αυτό ονομάζεται **κρυφό bit** και βοηθά στη μείωση της απαιτούμενης μνήμης.

3. Εκθέτης

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Η αναπαράσταση του εκθέτη γίνεται με μια **πόλωση** που το καθιστά μη προσημασμένο. Πραγματικός εκθέτης που προκύπτει από τον αποθηκευμένο εκθέτη ισούται με:

  Πραγματικός εκθέτης=Αποθηκευμένος εκθέτης−Πόλωση

4. Πόλωση

- Η πόλωση διευκολύνει την αναπαράσταση των εκθετών, με τις τυπικές τιμές:
  - **Απλή Ακρίβεια (Single Precision)**: πόλωση = 127
  - **Διπλή Ακρίβεια (Double Precision)**: πόλωση = 1023

5. Δομή της Αναπαράστασης

Αναλυτικά, η δομή του αριθμού σε δυαδική μορφή σύμφωνα με το πρότυπο IEEE περιλαμβάνει τα εξής μέρη:

- **S**: 1 bit
- **Εκθέτης**:
  - Απλή ακρίβεια: 8 bits
  - Διπλή ακρίβεια: 11 bits
- **Κλάσμα**:
  - Απλή ακρίβεια: 23 bits
  - Διπλή ακρίβεια: 52 bits

6. Παράδειγμα

Ένας αριθμός στην κινητή υποδιαστολή θα έχει την εξής μορφή:

S | Εκθέτης | Κλάσμα

Αυτή η δομή εξασφαλίζει την ακρίβεια των αριθμών, όπως και την κατάλληλη αναπαράσταση τους τόσο για πολύ μικρούς όσο και για πολύ μεγάλους αριθμούς, γεγονός το οποίο είναι κρίσιμο για επιστημονικούς υπολογισμούς.

Εύρος Τιμών Απλής Ακρίβειας

Η αναπαράσταση των αριθμών κινητής υποδιαστολής απλής ακρίβειας (32 bit) ακολουθεί το πρότυπο IEEE 754, το οποίο ορίζει πώς οι αριθμοί αποθηκεύονται και απεικονίζονται στον υπολογιστή. Ακολουθεί μια αναλυτική εξήγηση των μερών που καθορίζουν το εύρος τιμών στην απλή ακρίβεια:

Δομή της Αναπαράστασης

- **Δομή Bit**:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **S (Sign Bit)**: Το πρώτο bit είναι το bit προσήμου. Έχει την τιμή 0 για μη αρνητικούς αριθμούς (θετικούς) και 1 για αρνητικούς.
- **Εκθέτης**: Ακολουθούν 8 bits που αντιπροσωπεύουν τον εκθέτη.
- **Κλάσμα (Significand)**: Τα τελευταία 23 bits αναπαριστούν το κλάσμα.

- **Δεσμευμένα Bits**:
  - Οι εκθέτες 00000000 και 11111111 είναι δεσμευμένοι και δεν χρησιμοποιούνται για την αναπαράσταση κανονικών αριθμών.

Υπολογισμός Μικρότερων και Μεγαλύτερων Τιμών

- **Μικρότερη Τιμή**:

  Εκθέτης: 00000001. Ο πραγματικός εκθέτης υπολογίζεται ως:

  Πραγματικός Εκθέτης=1−127=−126

  Κλάσμα: Όταν το κλάσμα είναι 000...00, ο σημαντικός αριθμός είναι: Σημαντικός = 1.0

  - **Τελική Ανάθεση**: $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$

- **Μεγαλύτερη Τιμή**:
  - **Εκθέτης:** 11111110. Ο πραγματικός εκθέτης υπολογίζεται ως:

  Πραγματικός Εκθέτης=254−127=+127

  - **Κλάσμα:** Στην περίπτωση που το κλάσμα είναι 111...11, ο σημαντικός αριθμός πλησιάζει το άπειρο: Σημαντικος≈2.0
  - **Τελική Ανάθεση**: $\pm 2.0 \times 2^{127} \approx \pm 3.4 \times 10^{38}$

Εύρος Τιμών Διπλής Ακρίβειας

Η διπλή ακρίβεια (double precision) αναφέρεται στην αναπαράσταση αριθμών κινητής υποδιαστολής με 64 bit, και είναι πιο ακριβής από την απλή ακρίβεια (single precision) που χρησιμοποιεί 32 bit. Το πρότυπο IEEE 754 ορίζει πώς γίνεται αυτή η αναπαράσταση και καθορίζει το εύρος τιμών που μπορούν να αναπαρασταθούν.

Δεσμευμένοι Εκθέτες

- Οι εκθέτες 0000...00 και 1111...1111 είναι δεσμευμένοι και δεν χρησιμοποιούνται για αναπαράσταση αριθμών. Αυτοί οι εκθέτες χρησιμοποιούνται για ειδικές περιπτώσεις όπως το μηδέν και το άπειρο.

Μικρότερη Τιμή

- **Εκθέτης:** 00000000001

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Ο πραγματικός εκθέτης εδώ υπολογίζεται ως:

Πραγματικός εκθέτης=εκθέτης – πόλωση 1−1023= −1022

- **Κλάσμα:** 000...00
- Το σημαντικό εδώ είναι 1.0.
- **Τιμή:**$\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$

Αυτό σημαίνει ότι οι πολύ μικρές αριθμητικές τιμές μπορούν να αναπαρασταθούν, όπως αρνητικοί και θετικοί αριθμοί κοντά στο μηδέν.

Μεγαλύτερη Τιμή

- **Εκθέτης:** 11111111110
- Ο πραγματικός εκθέτης εδώ υπολογίζεται ως:

Πραγματικός εκθέτης=2046−1023=+

Κλάσμα: 111...11

- Το σημαντικό είναι περίπου 2.0.
- **Τιμή:**$\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{308}$

Αυτό υποδεικνύει ότι οι μεγαλύτερες δυνατές αριθμητικές τιμές που μπορούν να αναπαρασταθούν σε διπλή ακρίβεια είναι πολύ μεγάλες, ικανές να καλύψουν ευρύ φάσμα επιστημονικών και μαθηματικών αναγκών.

Ο λόγος που ο αποθηκευμένος εκθέτης στην αναπαράσταση διπλής ακρίβειας (double precision) είναι 2046 είναι λόγω του τρόπου που αναπαρίσταται ο εκθέτης σύμφωνα με το πρότυπο IEEE 754. Ας αναλύσουμε τη διαδικασία:

Διαδικασία Υπολογισμού του Αποθηκευμένου Εκθέτη

- **Αριθμός Μπιτών για τον Εκθέτη**:
  - Στην αναπαράσταση διπλής ακρίβειας, ο εκθέτης έχει μήκος 11 bit.
- **Διαφορετικά Εύρη Εκθετών**:
  - Στο πρότυπο IEEE 754, οι εκθέτες έχουν μία "πόλωση" (bias), που είναι ένας σταθερός αριθμός που προστίθεται στον πραγματικό εκθέτη προκειμένου να επιτευχθεί η αναπαράσταση των θετικών και αρνητικών εκθετών.
- **Πόλωση**:
  - Για τη διπλή ακρίβεια, η πόλωση είναι 1023. Αυτό σημαίνει ότι για να αναπαρασταθεί ένας εκθέτης, προστίθεται το 1023 στον πραγματικό εκθέτη (exponent):

Αποθηκευμένος Εκθέτης=Πραγματικός Εκθέτης+1023

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Υπολογισμός Αποθηκευμένου Εκθέτη**:
  - Ένας πραγματικός εκθέτης της μέγιστης τιμής, που είναι 1023, αποθηκεύεται ως: 1023+1023=2046
  - Ένας πραγματικός εκθέτης της ελάχιστης θετικής τιμής, που είναι -1022, αποθηκεύεται ως: −1022+1023=1
  - Οι ειδικές περιπτώσεις όπου ο εκθέτης είναι 0 ή 2047 είναι δεσμευμένες για τον μηδενικό αριθμό και το άπειρο.

Συμπέρασμα

Ο αποθηκευμένος εκθέτης 2046 σχετίζεται με τη μέγιστη θετική αριθμητική τιμή που μπορεί να αναπαρασταθεί με την κανονικοποιημένη μορφή διπλής ακρίβειας και καθορίζεται από την πόλωση που εφαρμόζεται στην αναπαράσταση. Αυτό δίνει τη δυνατότητα να διατηρούμε μη προσημασμένες αναπαραστάσεις για τους εκθέτες, επιτρέποντας την αναπαράσταση τόσο πολύ μεγάλων όσο και πολύ μικρών αριθμών.

Ακρίβεια Τιμών Κινητής Υποδιαστολής

Σχετική Ακρίβεια

- **Όλα τα κλασματικά bit είναι σημαντικά**: Αυτό σημαίνει ότι κάθε bit που συμμετέχει στην αναπαράσταση του κλάσματος έχει αξία και επηρεάζει την ακρίβεια της αναπαράστασης.

Απλής Ακρίβειας

Η ακρίβεια των αριθμών που αναπαρίστανται ως απλής ακρίβειας υπολογίζεται περίπου ως 2^{-23}.

  - Αυτό ισοδυναμεί με:

$23 \times \log_{10} 2 \approx 23 \times 0.3 \approx 6$

Άρα, οι αριθμοί που αναπαρίστανται σε αυτούς τους τύπους έχουν περίπου ακρίβεια **6 δεκαδικών ψηφίων**.

Διπλής Ακρίβειας

Η ακρίβεια των αριθμών σε διπλή ακρίβεια υπολογίζεται περίπου ως 2^{-52}

  - Αυτό επίσης ισοδυναμεί με:

$52 \times \log_{10} 2 \approx 52 \times 0.3 \approx 16$

Έτσι, οι αριθμοί που αναπαρίστανται ως διπλής ακρίβειας έχουν περίπου ακρίβεια **16 δεκαδικών ψηφίων**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Παράδειγμα Αναπαράστασης Κινητής Υποδιαστολής

Η αναπαράσταση του αριθμού -0.75 στην κινητή υποδιαστολή είναι ένα παράδειγμα του πώς οι υπολογιστές χειρίζονται αριθμούς με κλασματικά μέρη. Ας αναλύσουμε αυτό το παράδειγμα και τους υπολογισμούς του:

1. Αναπαράσταση -0.75

Ο αριθμός -0.75 μπορεί να αναπαρασταθεί με τον εξής τρόπο:

$-0.75=(-1)^1\times 1.1_2\times 2^{-1}$

- Το $(-1)^1$ δείχνει ότι ο αριθμός είναι αρνητικός (bit προσήμου S = 1).
- Το 1.1 αντιπροσωπεύει το κλάσμα (η δυαδική αναπαράσταση του 0.75).
- Ο εκθέτης είναι -1, που υποδεικνύει τη θέση της υποδιαστολής.

2. Σημαντικό

Το σημαντικό (fraction) μετά τη διαδικασία κανονικοποίησης είναι το κλάσμα 1.1_2. Στη μορφή κινητής υποδιαστολής, το "1" υπονοείται, επομένως δεν αναπαρίσταται ρητά.

3. Υπολογισμός Εκθέτη

Ο εκθέτης για την αναπαράσταση κινητής υποδιαστολής υπολογίζεται ως εξής:

Για απλή ακρίβεια (32 bit):

- Η πόλωση είναι 127 (defining bias).
- Για τον εκθέτη:

$-1+127=126$ (σε δυαδική: 011111102)

Για διπλή ακρίβεια (64 bit):

- Η πόλωση είναι 1023.
- Για τον ίδιο εκθέτη

- 1+1023=1022(σε δυαδική: 01111111110)

4. Τελική Αναπαράσταση

Η τελική αναπαράσταση του αριθμού για τις δύο διαφορετικές ακρίβειες είναι:

- **Απλή ακρίβεια (32 bit)**:
  - **S** = 1
  - **Εκθέτης** = 01111110
  - **Κλάσμα** = 10111110 (συμπεριλαμβανομένου του κρυφού bit)

Καθώς μπορούμε να ολοκληρώσουμε:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Διπλή ακρίβεια (64 bit)**:
  - **S** = 1
  - **Εκθέτης** = 01111111110
  - **Κλάσμα** = 101111111110 (συμπεριλαμβάνοντας περισσότερα bits).

Αυτή η διαδικασία επισφραγίζει την αναπαράσταση του -0.75 με τη χρήση της κανονικοποιημένης μορφής κινητής υποδιαστολής.

Αριθμοί Κινητής Υποδιαστολής: Παράδειγμα και Επεξήγηση

Ο αριθμός κινητής υποδιαστολής απλής ακρίβειας αναπαρίσταται με τρία βασικά στοιχεία: ένα bit για το πρόσημο, ένα κλάσμα (ή σημαντικό) και έναν εκθέτη.

Στο παραδείγμα μας, η αναπαράσταση είναι:

11000000101000...00

1. Στοιχεία της Αναπαράστασης

- **S (Bit Πρόσημου)**:
  - Το πρώτο bit 1 δηλώνει ότι ο αριθμός είναι αρνητικός. Δηλαδή, S=1 υποδηλώνει ότι θα υπολογίσουμε τον αριθμό ως $(-1)^1$
- **Κλάσμα (Σημαντικό)**:
  - Ο αριθμός 01000...00 δηλώνει το κλάσμα που υπολογίζεται ως 1+0.25=1.25 (η δυαδική μετατροπή του 01000...00 είναι 0.25).
  - Δηλαδή, το σημαντικό =1+0.25 υπολογίζεται με την κανονικοποίηση.
- **Εκθέτης**:
  - Ο δυαδικός αριθμός 10000001 ισοδυναμεί με 129. Στην κινητή υποδιαστολή, οι εκθέτες συνήθως υποβάλλονται σε "πόλωση" (bias). Για την απλή ακρίβεια, η πόλωση είναι 127. Άρα, ο πραγματικός εκθέτης είναι:

129−127=2

2. Υπολογισμός του Δεκαδικού Αριθμού

Ο αριθμός κινητής υποδιαστολής υπολογίζεται με τον τύπο:

$x=(-1)^S\times(1+\text{κλασμα})\times 2^{(\text{εκθέτης- πόλωση})}$

Οπότε, για το συγκεκριμένο παράδειγμα έχουμε:

$x=(-1)^1\times(1+0.25)\times 2^2 = -5.0$

3. Συμπεράσματα

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Η διαδικασία αναπαράστασης αριθμών κινητής υποδιαστολής είναι κρίσιμη για πολλές εφαρμογές υπολογιστών, ειδικότερα σε επιστημονικούς υπολογισμούς, όπου η ακρίβεια των πραγματικών αριθμών είναι σημαντική. Η κινητή υποδιαστολή επιτρέπει την αναπαράσταση ενός ευρέος εύρους αριθμών με περιορισμένη ακρίβεια, προσφέροντας την δυνατότητα να εκφράσουν αρνητικούς ή πολύ μικρούς/πολύ μεγάλους αριθμούς.

Εξήγηση των Μη Κανονικοποιημένων Αριθμών στην Κινητή Υποδιαστολή

1. **Βασική Ιδέα της Κινητής Υποδιαστολής**

Η κινητή υποδιαστολή είναι μια μέθοδος αναπαράστασης πραγματικών αριθμών στους υπολογιστές, η οποία επιτρέπει την αναπαράσταση τόσο πολύ μικρών όσο και πολύ μεγάλων αριθμών. Η αναπαράσταση αυτή περιλαμβάνει τρία κύρια στοιχεία:

- **Σημαντικό (Significand)**: Απαρτίζεται από τα ψηφία του αριθμού.
- **Εκθέτης (Exponent)**: Καθορίζει τη θέση της υποδιαστολής.
- **Bit Προσήμου (Sign Bit)**: Δηλώνει αν ο αριθμός είναι θετικός ή αρνητικός.

2. **Μη Κανονικοποιημένοι Αριθμοί**

Μη κανονικοποιημένοι αριθμοί είναι οι αριθμοί που δεν μπορούν να αναπαρασταθούν με κανονικούς κανόνες και έχουν ειδικές συνθήκες:

- **Εκθέτης = 000...0**: Στη περίπτωση αυτή, ο εκθέτης είναι 0, πράγμα που υποδεικνύει ότι δεν υπάρχει κανονικοποιημένη μορφή για αυτόν τον αριθμό.
- **Κρυφό Bit ίσο με 0**: Στους μη κανονικοποιημένους αριθμούς, το “κρυφό bit” είναι 0, δηλαδή δεν υπονοείται το 1 μπροστά από το κλάσμα όπως συμβαίνει στους κανονικούς αριθμούς.

3. **Χαρακτηριστικά Μη Κανονικοποιημένων Αριθμών**

- **Μικρότεροι από τους Κανονικούς Αριθμούς**: Η αναπαράσταση των μη κανονικοποιημένων αριθμών επιτρέπει την ύπαρξη αριθμών που είναι μικρότεροι από τους ελάχιστους κανονικούς. Αυτό είναι χρήσιμο, για παράδειγμα, όταν αντιπροσωπεύουμε πολύ μικρές τιμές που δεν μπορούν να εκφραστούν με κανονικές μορφές.
- **Βαθμιαία Ανεπάρκεια με Φθίνουσα Ακρίβεια**: Όταν χρησιμοποιούμε μη κανονικοποιημένους αριθμούς, η ακρίβεια της αναπαράστασης μειώνεται καθώς οι τιμές γίνονται πιο μικρές. Αυτό σημαίνει ότι η ακρίβεια των υπολογισμών μπορεί να μειωθεί όταν χρησιμοποιούνται αυτές οι αναπαραστάσεις.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Δύο Αναπαραστάσεις για το 0.0**: Υπάρχουν δύο διαφορετικές αναπαραστάσεις για το 0.0 στους μη κανονικοποιημένους αριθμούς. Αυτό οφείλεται στο γεγονός ότι το κρυφό bit μπορεί να είναι 0 και το κλάσμα μπορεί επίσης να είναι 0.

4. **Σημασία στην Αριθμητική Υπολογιστών**

Η χρήση του μη κανονικοποιημένου αριθμού είναι σημαντική σε εφαρμογές όπου η αναπαράσταση πολύ μικρών αριθμών είναι κρίσιμη, όπως στην επιστημονική υπολογιστική και τη μελέτη των αριθμητικών σφαλμάτων. Οι υπολογιστές πρέπει να είναι σε θέση να διαχειρίζονται με ακρίβεια αυτά τα ειδικά σενάρια.

Θετικό/Αρνητικό Άπειρο και NaN

Στην αναπαράσταση κινητής υποδιαστολής, υπάρχουν ειδικές τιμές που χρησιμοποιούνται για να υποδηλώνουν άπειρα και μη έγκυρα αριθμητικά αποτελέσματα. Ακολουθεί μια αναλυτική εξήγηση αυτού του φαινομένου:

1. Άπειρο

- **Εκθέτης = 111...1, Κλάσμα = 000...0**:
  - Αυτή η ειδική περίπτωση ορίζει τις τιμές άπειρο.
  - Το "θετικό άπειρο" αναπαρίσταται με εκθέτη ίσο με 111...1 και κλάσμα 000...0, ενώ το "αρνητικό άπειρο" αναπαρίσταται με τον ίδιο εκθέτη αλλά υπάρχει διαφορά στο πρόσημο.
- **Χρήση στους υπολογισμούς**:
  - Το άπειρο μπορεί να χρησιμοποιηθεί σε επόμενους υπολογισμούς, καθώς η αριθμητική συμπεριφορά του είναι προκαθορισμένη.
  - Δε χρειάζεται έλεγχος υπερχείλισης, καθώς οι αριθμητικές πράξεις που περιλαμβάνουν άπειρο συνήθως εξάγουν άπειρο ως αποτέλεσμα χωρίς προβλήματα υπολογισμού.

2. Not-a-Number (NaN)

- **Εκθέτης = 111...1, Κλάσμα ≠ 000...0**:
  - Αυτή η περίπτωση αντιπροσωπεύει το NaN, το οποίο υποδεικνύει ένα μη έγκυρο ή μη ορισμένο αποτέλεσμα.
  - Ένα κοινό παράδειγμα είναι η διαίρεση 0.0 / 0.0, η οποία δεν έχει έγκυρο αποτέλεσμα.
- **Χρήση των NaN**:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Το NaN μπορεί επίσης να χρησιμοποιηθεί σε επόμενους υπολογισμούς.
- Αυτή η δυνατότητα εξασφαλίζει ότι οι υπολογισμοί δεν σταματούν ή δεν προκαλούν σφάλματα όταν προκύπτει ένα NaN. Αντίθετα, το NaN «διαταράσσει» τους υπολογισμούς και το αποτέλεσμα παραμένει NaN σε όλες τις ενέργειες που το περιλαμβάνουν.

Συμπέρασμα

Αυτές οι ειδικές τιμές (άπειρο και NaN) συμβάλλουν στην επιτυχία των υπολογισμών κινητής υποδιαστολής, παρέχοντας μια υγιή μέθοδο για την αντιμετώπιση αριθμητικών περιπτώσεων που αλλιώς θα μπορούσαν να οδηγήσουν σε ανακρίβειες ή σφάλματα κατά τη διάρκεια της εκτέλεσης.

Εύρος & Ακρίβεια Κινητής Υποδιαστολής

Η αναπαράσταση των αριθμών κινητής υποδιαστολής είναι ένα από τα πιο σημαντικά θέματα στην υπολογιστική αριθμητική. Αυτή η αναπαράσταση επιτρέπει την αποθήκευση και την επεξεργασία πολύ μεγάλων και πολύ μικρών αριθμών, χρησιμοποιώντας έναν περιορισμένο αριθμό bits. Ο IEEE 754 προσδιορίζει πώς γίνεται αυτή η αναπαράσταση.

Εύρος Αναπαράστασης

- **Αριθμός Bits**: Η αναπαράσταση κινητής υποδιαστολής μπορεί να χρησιμοποιεί περιορισμένο αριθμό bits. Για παράδειγμα, η απλή ακρίβεια (single precision) χρησιμοποιεί 32 bits, ενώ η διπλή ακρίβεια (double precision) χρησιμοποιεί 64 bits.
- **Εύρος Τιμών**: Το εύρος των τιμών που μπορούν να αναπαρασταθούν εξαρτάται από τον αριθμό των bits που χρησιμοποιούνται. Στην απλή ακρίβεια, το εύρος κυμαίνεται περίπου από -3.4×10^{38}
- **Απρόσημοι Ακέραιοι** και **Προσημασμένοι Ακέραιοι**: Σε σύγκριση με τους ακέραιους, οι αριθμοί κινητής υποδιαστολής έχουν ευρύτερο εύρος αλλά λιγότερη ακρίβεια.

Ακρίβεια Κινητής Υποδιαστολής

- **Σημαντικά Bits**: Η ακρίβεια της αναπαράστασης κινητής υποδιαστολής αναφέρεται στη δυνατότητα να περιγράψει μικρές διαφορές μεταξύ αριθμών. Η απλή ακρίβεια έχει περίπου 6-7 δεκαδικά ψηφία ακριβείας, ενώ η διπλή ακρίβεια έχει περίπου 15-16 δεκαδικά ψηφία.
- **Συγκρίσεις με Ακέραιους**: Ενώ οι ακέραιοι αριθμοί έχουν σταθερή ακρίβεια, οι αριθμοί κινητής υποδιαστολής ενδέχεται να έχουν σφάλματα στρογγυλοποίησης λόγω της περιορισμένης χωρητικότητάς τους. Αυτό

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

σημαίνει ότι μπορεί να προκύψουν προβλήματα ακριβείας κυρίως σε υπολογισμούς που περιλαμβάνουν μεγάλη διαφορά μεταξύ των εκθετών ή μικρές μεταβολές.

Εύρος Αναπαράστασης Αριθμών με 32 bit

Στο πλαίσιο της αριθμητικής των υπολογιστών, η δυνατότητα να αναπαρίστανται διαφορετικοί τύποι αριθμών είναι κρίσιμη. Στη συγκεκριμένη περίπτωση, εξετάζουμε την αναπαράσταση αριθμών σε 32 bit, η οποία επιτρέπει την αποθήκευση ακεραίων και πραγματικών αριθμών με διαφορετικές αναπαραστάσεις.

Μορφές Αριθμών

- **Απρόσημοι Ακέραιοι**:
  - Αυτοί οι αριθμοί είναι μη αρνητικοί και έχουν εύρος αναπαράστασης από 00 έως 4,294,967,2954,294,967,295
  - Το συνολικό εύρος μπορεί να εκφραστεί ως σχεδόν 4 δισεκατομμύρια.
  - Ο απρόσημος ακέραιος αποθηκεύει μόνο θετικό αριθμούς, ξεκινώντας από το μηδέν.
- **Προσημασμένοι Ακέραιοι**:
  - Χρησιμοποιούν το χαρακτηριστικό του "συμπληρώματος προς 2" για την αναπαράσταση θετικών και αρνητικών αριθμών.
  - Το εύρος τους κυμαίνεται από −2,147,483,648−2,147,483,648 έως +2,147,483,647+2,147,483,647
  - Κοινά, οι προσημασμένοι ακέραιοι επιτρέπουν τη διαχείριση και των θετικών και των αρνητικών αριθμών με μεγαλύτερη ευκολία.
- **Πραγματικοί Αριθμοί (Απλή Ακρίβεια)**:
  - Αυτοί οι αριθμοί αναπαρίστανται με μορφή κινητής υποδιαστολής, που περιλαμβάνει ακέραια και κλασματικά μέρη.
  - Μπορούν να αναπαρασταθούν αξίες από περίπου:
    - ±340,282,346,638,528,859,811,704,183,484,516,925,440±340,282,346,638,528,859,811,704,183,484,516,925,440 (μέγιστο).
    - Ελάχιστο: 0.00000000000000000000000000000000000235098856151472858345576598207153302664571798551798085536592623685000612993034607711706485133618116378784179687 5

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Αναπαράσταση σε 64 bits

Πραγματικοί Αριθμοί Διπλής Ακρίβειας

- **Διπλή ακρίβεια**:
  - Μέγιστη τιμή: περίπου

  $1.79769313486231570814527423732 \times 10^{308}$

  - Ελάχιστη τιμή: περίπου

  $4.45014771701440227211481959342 \times 10^{-308}$

Απόσταση Μεταξύ Αριθμών στην Κινητή Υποδιαστολή

Η αναπαράσταση κινητής υποδιαστολής (floating-point representation) είναι μια κοινή μέθοδος που χρησιμοποιείται στους υπολογιστές για την αναπαράσταση πραγματικών αριθμών. Στην κινητή υποδιαστολή, οι αριθμοί αναπαρίστανται σε μια μορφή που περιλαμβάνει ένα προσήμα (sign), έναν εκθέτη (exponent) και ένα κλάσμα (fraction ή significand).

Απόσταση Μεταξύ Διαδοχικών Αριθμών

Στην κινητή υποδιαστολή, **η απόσταση μεταξύ διαδοχικών πραγματικών αριθμών εξαρτάται από τον εκθέτη τους**. Όσο μεγαλύτερος είναι ο εκθέτης, τόσο μεγαλύτερη είναι η απόσταση μεταξύ των διαδοχικών αριθμών στην αναπαράσταση.

Παράδειγμα

Προκειμένου να κατανοηθεί αυτή η έννοια, εξετάζουμε τα παρακάτω παραδείγματα:

- **Αριθμός 1:**

442 721 857 769 029 238 784 | 0 11000011 10000000000000000000000

(S=1(+), Exponent = 195 - 127 = 68)

Κλάσμα: 0.1(σε δυαδική μορφή =0.5) Έχω παράρτημα για το πώς μετατρέπουμε στο τέλος)

Significant: 1.1 (σε δυαδική μορφή =1.5)

- **Αριθμός 2:**

442 721 892 953 401 327 616 | 0 11000011 10000000000000000000001

(S=1, Exponent = 195 - 127 = 68)

Κλάσμα: $0.10000000000000000000001_2$ (δεκαδική) μορφή =0.5000001192092896

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Significant: 1.10000000000000000000001

(σε δεκαδική μορφή =1.5000001192092896)

Η "απόσταση" μεταξύ αυτών των δύο αριθμών είναι:

44272189295340132761 6 −4427218557769029238784 0\

≈35,184,372,088,832

Αυτό δείχνει ότι η απόσταση μεταξύ του πρώτου και του δεύτερου αριθμού είναι **35 τρισεκατομμύρια**.

Συμπέρασμα

Η παραπάνω ανάλυση δείχνει πώς οι αριθμοί θεωρούνται στην κυριολεκτική τους αναπαράσταση στον υπολογιστή και πόσο επηρεάζει ο εκθέτης την ακρίβεια και την απόσταση μεταξύ των διαδοχικών αριθμών. Έτσι, η κατανόηση της απλής και διπλής ακρίβειας στην αναπαράσταση κινητής υποδιαστολής είναι κρίσιμη για τις μαθηματικές πράξεις και τη διαχείριση των πραγματικών αριθμών στην υπολογιστική επιστήμη και την αριθμητική.

Αντίθετα, όταν οι αριθμοί έχουν χαμηλό εκθέτη, η απόσταση μεταξύ τους περιορίζεται και πιο πολλοί αριθμοί μπορούν να αναπαρασταθούν.

Παράδειγμα

Από το κείμενο, έχουμε τα παρακάτω παραδείγματα:

Μεγάλος Εκθέτης

- **Αριθμός**: 442,721,857,769,029,238,784
  - **S** = 1 (+)
  - **Exponent**: 195 – 127 = 68
  - **Fraction**: 0.1
  - **Απόσταση**: Μπορεί να είναι τάξης 35 τρισεκατομμυρίων!

Μικρός Εκθέτης

- **Αριθμός**: 6,442,450,944
  - **S** = 1 (+)
  - **Exponent**: 159 – 127 = 32
  - **Fraction**: 0.1
  - **Απόσταση**: Μόνο λίγες εκατοντάδες!

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Αυτό δείχνει πώς η αυξομείωση του εκθέτη επηρεάζει την απόσταση μεταξύ των διαδοχικών αριθμών στην αναπαράσταση κινητής υποδιαστολής. Το συμπέρασμα είναι ότι οι αριθμοί με υψηλούς εκθέτες είναι πιο αραιά κατανεμημένοι, ενώ οι αριθμοί με χαμηλούς εκθέτες είναι πιο πυκνοί, γεγονός που επηρεάζει την ακρίβεια και την ικανότητα αναπαράστασης των αριθμών σε υπολογιστές.

Πρόσθεση Κινητής Υποδιαστολής

Η πρόσθεση αριθμών κινητής υποδιαστολής απαιτεί μια σειρά βημάτων που είναι κρίσιμα για την ακριβή εκτέλεση των υπολογισμών.

<u>ΠΑΡΑΔΕΙΓΜΑ 1</u>

Ακολουθεί μια ανάλυση της διαδικασίας προσθέσεως, χρησιμοποιώντας το αναφερόμενο παράδειγμα:

$9.999\times10^{1}+1.610\times10^{-1}$

Βήματα Πρόσθεσης

Βήμα 1.Ευθυγράμμιση Υποδιαστολών

Ο σκοπός αυτού του βήματος είναι να κάνουμε τους δύο αριθμούς συγκρίσιμους. Αυτό επιτυγχάνεται με την ολίσθηση του αριθμού με μικρότερο εκθέτη, για να εναρμονιστούν οι εκθέτες.

Συγκεκριμένα:

$1.610 \times 10^{-1} \rightarrow 0.016\times10^{1}$

Τώρα το παράδειγμα γίνεται:

$9.999\times10^{1}+0.016\times10^{1}$

Βήμα 2.Πρόσθεση των Σημαντικών

Αφού εναρμονιστούν οι εκθέτες, προσθέτουμε τους σημαντικούς αριθμούς.

$9.999+0.016=10.015$

Έτσι, το αποτέλεσμα είναι: 10.015×10^{1}

Βήμα 3.Κανονικοποίηση αποτελέσματος

Σε αυτό το βήμα κανονικοποιούμε το αποτέλεσμα για να διασφαλίσουμε ότι βρίσκεται σε αποδεκτή μορφή.

Η κανονικοποίηση συνεπάγεται την μετατροπή του αριθμού έτσι ώστε το σημαντικό να είναι της μορφής 1.xxxxx

$10.015\times10^{1}=1.0015\times10^{2}$

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Βήμα 4.Στρογγυλοποίηση και Ανακανονικοποίηση

Τέλος, εάν είναι απαραίτητο, μπορεί να γίνει στρογγυλοποίηση του αριθμού:

$1.0015\times10^2 \rightarrow 1.002\times10^2$

Συμπέρασμα

Η διαδικασία πρόσθεσης κινητής υποδιαστολής περιλαμβάνει την ευθυγράμμιση των εκθετών, την πρόσθεση των σημαντικών, την κανονικοποίηση του αποτελέσματος και, όταν χρειάζεται, τη στρογγυλοποίηση του τελικού αποτελέσματος. Κάθε βήμα είναι κρίσιμο για την ακριβή αναπαράσταση των αριθμών και τη διασφάλιση του σωστού αποτελέσματος.

ΠΑΡΑΔΕΙΓΜΑ 2

- **Αριθμοί που προστίθενται**:

$1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)

Βήμα 1ο .Ολίσθηση Εκθετών: Το πρώτο βήμα στην πρόσθεση είναι η ευθυγράμμιση των υποδιαιρέσεων. Αυτό γίνεται ολισθαίνοντας τον αριθμό με τον μικρότερο εκθέτη ώστε οι βάσεις (το κλάσμα) να έχουν το ίδιο εκθετικό μέρος. Στο παράδειγμά μας:

$1.0 \times 2^{-1} + (-0.111)\times 2^{-1}$

Βήμα 2ο.Πρόσθεση Σημαντικών: Αφού έχουμε ευθυγραμμίσει τους εκθέτες, προσθέτουμε τα σημαντικά μέρη των αριθμών:

$1.000+(-0.111)=0.001\times 2^{-1}$

Αυτό το βήμα υποδηλώνει ότι η απλή πρόσθεση των δυαδικών αριθμών έχει γίνει.

Βήμα 3ο.Κανονικοποίηση και Έλεγχος: Αφού γίνει η πρόσθεση των σημαντικών, ελέγχουμε αν υπάρχει υπερχείλιση ή ανεπάρκεια. Στην περίπτωσή μας, το αποτέλεσμα κανονικοποιείται σε:

1.0002×2^{-4}

Δεν υπάρχει υπερχείλιση ή ανεπάρκεια εδώ, πράγμα που σημαίνει ότι το αποτέλεσμα είναι έγκυρο εντός του αναπιεστικού εύρους.

Βήμα 4ο.Στρογγυλοποίηση και Επανκανονικοποίηση: Τέλος, εάν χρειάζεται, γίνεται στρογγυλοποίηση του αποτελέσματος και, αν είναι απαραίτητο, εκ νέου κανονικοποίηση. Στην περίπτωση αυτή δεν φαίνεται να υπάρχει καμία αλλαγή στο αποτέλεσμα, το οποίο είναι 0.0625.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Υλικό για την Πρόσθεση Κινητής Υποδιαστολής

Η πρόσθεση αριθμών κινητής υποδιαστολής (floating point addition) είναι μια διαδικασία που απαιτεί περισσότερο περίπλοκο υλικό σε σύγκριση με την πρόσθεση ακέραιων αριθμών. Αυτό οφείλεται σε διάφορους λόγους:

1. Σύνθετη Μηχανική Δομή

Η πρόσθεση κινητής υποδιαστολής περιλαμβάνει διαδικασίες όπως είναι η ευθυγράμμιση των εκθετών και η κανονικοποίηση του αποτελέσματος. Αυτές οι διαδικασίες προσθέτουν στον υπολογισμό περισσότερους μετασχηματισμούς και συνεπώς απαιτούν πιο σύνθετο υλικό:

- **Ευθυγράμμιση Υποδιαστολών:** Πριν από την πρόσθεση, οι δύο αριθμοί πρέπει να ευθυγραμμιστούν, δηλαδή να έχουν τον ίδιο εκθέτη. Αυτό απαιτεί ολίσθηση του αριθμού με τον μικρότερο εκθέτη προς τα δεξιά.
- **Κανονικοποίηση:** Μετά την προσθήκη, το αποτέλεσμα πρέπει να κανονικοποιηθεί για να διασφαλιστεί ότι παραμένει η σωστή μορφή κινητής υποδιαστολής.

2. Χρόνος Εκτέλεσης

Η διαδικασία της προσθήκης κινητής υποδιαστολής συνήθως διαρκεί περισσότερους κύκλους ρολογιού σε σχέση με τις πράξεις ακέραιων αριθμών. Αυτό συμβαίνει γιατί:

- Οι διαδικασίες είναι πιο χρονοβόρες.
- Ενδέχεται να απαιτηθούν πολλαπλές μετατροπές και στρογγυλοποιήσεις.

3. Εξάρτηση από τον Ρόλο του Ρολογιού

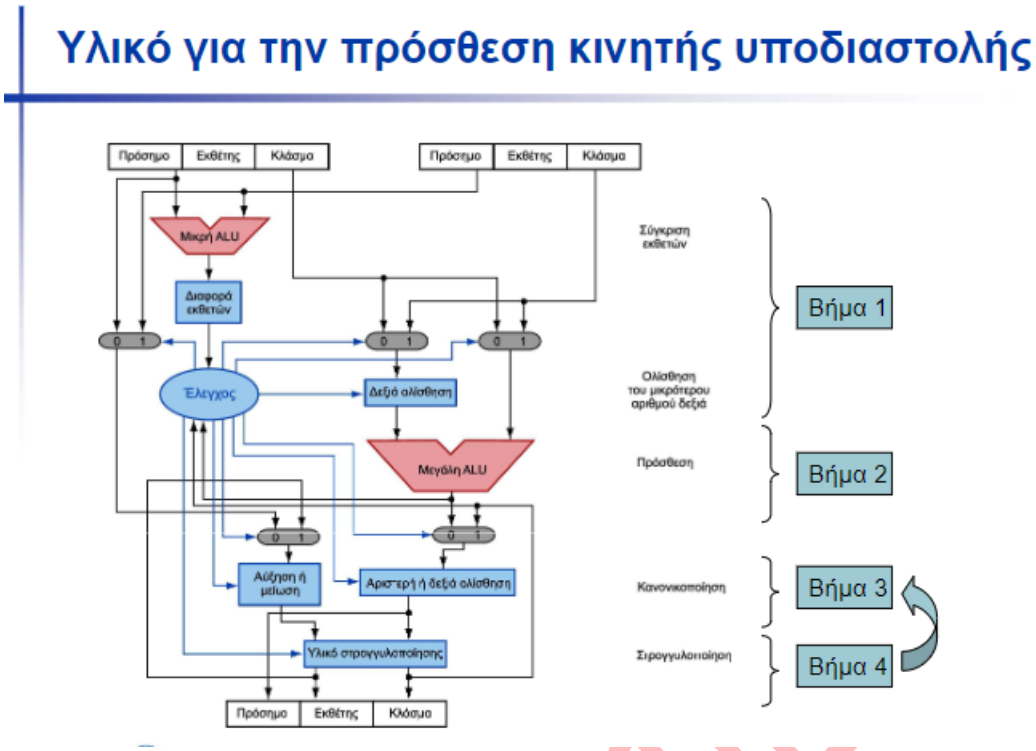
Συνήθως, η προσθήκη κινητής υποδιαστολής επιβαρύνει όλες τις εντολές καθώς συνδέεται με τον ρυθμό του ρολογιού. Όταν η ταχύτητα του ρολογιού είναι χαμηλή, οι εντολές αυτές θα χρειάζονται σημαντικά περισσότερους χρόνους εκτέλεσης, επιβαρύνοντας την γενική απόδοση του συστήματος.

4. Διοχέτευση

Στην υλοποίηση της πρόσθεσης κινητής υποδιαστολής, χρησιμοποιείται και η τεχνική της διοχέτευσης (pipelining), η οποία επιτρέπει την εκτέλεση πολλών βημάτων σε διάφορους κύκλους ρολογιού ταυτόχρονα. Αυτό σημαίνει ότι διαφορετικές φάσεις του υπολογιστικού κύκλου μπορούν να εκτελούνται παράλληλα, βελτιώνοντας την απόδοση.

Στα κυκλώματα του υπολογιστή

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Ανάλυση Βημάτων:

Βήμα 1 - Σύγκριση εκθετών

- Η **Μικρή ALU** υπολογίζει τη διαφορά των εκθετών των δύο αριθμών κινητής υποδιαστολής.
- Το αποτέλεσμα καθορίζει ποιος αριθμός έχει μικρότερο εκθέτη.

Βήμα 2 - Ολίσθηση μικρότερου αριθμού

- Ο αριθμός με τον μικρότερο εκθέτη μετακινείται προς τα δεξιά μέσω ολίσθησης (λογική δεξιά ολίσθηση) ώστε οι εκθέτες να ευθυγραμμιστούν.
- Αυτό επιτυγχάνεται με τη χρήση του **Δεξιού ολισθητή** και της **Μεγάλης ALU**.

Βήμα 3 - Πρόσθεση

- Αφού οι εκθέτες είναι ίσοι, η **Μεγάλη ALU** προσθέτει τα προσημασμένα σημαντικά (mantissas).
- Αν χρειαστεί, γίνεται **αύξηση ή μείωση** του εκθέτη λόγω υπερχείλισης ή μεταφοράς.

Βήμα 4 - Κανονικοποίηση & Στρογγυλοποίηση

- Αν το αποτέλεσμα δεν είναι κανονικοποιημένο (δηλαδή δεν ξεκινά με το πιο σημαντικό bit), γίνεται κατάλληλη **ολίσθηση και προσαρμογή του εκθέτη**.
- Η **μονάδα στρογγυλοποίησης** εφαρμόζει το κατάλληλο σχήμα στρογγυλοποίησης.

ΕΛΕΝΗ ΖΑΦΕ

Πολλαπλασιασμός κινητής υποδιαστολής

Ο πολλαπλασιασμός αριθμών κινητής υποδιαστολής περιλαμβάνει αρκετά βήματα που απαιτούν προσεκτική εκτέλεση, ώστε να διασφαλιστεί η ακρίβεια των αποτελεσμάτων. Παρακάτω αναλύονται τα βήματα του πολλαπλασιασμού και οι σχετικές διαδικασίες.

ΠΑΡΑΔΕΙΓΜΑ 1

Δίνεται το παράδειγμα:

$1.110\times10^{10}\times9.200\times10^{-5}$

1. Πρόσθεση εκθετών

Η παράμετρος που αφορά τους εκθέτες περιλαμβάνει τη διαδικασία για την αναγνώριση των πολωμένων εκθετών.

- Οι πολωμένοι εκθέτες έχουν μια προκαθορισμένη τιμή που ονομάζεται πόλωση.
- Σε αυτή τη διαδικασία, η πόλωση αφαιρείται από το άθροισμα των εκθετών:

[Νέος \ εκθέτης = 10 + (-5) = 5]

2. Πολλαπλασιασμός των σημαντικών

Αφού αποφασίσουμε τους εκθέτες, προχωρούμε στην πολλαπλασίαση των σημαντικών (mantissa):

$1.110 \times 9.200=10.212$

Μετά την πολλαπλασίαση, ο πολλαπλασιασμένος αριθμός αναπαρίσταται με τον νέο εκθέτη:

10.212×10^{5}

3. Κανονικοποίηση

Στη συνέχεια, ο αριθμός κανονικοποιείται ώστε να τηρεί τις προδιαγραφές για τους αριθμούς κινητής υποδιαστολής. Κανονικοποίηση σημαίνει ότι ο σημαντικός αναπαρίσταται με τρόπο που να είναι $1.0 \leq |\text{σημαντικό}| < 2.0$.

- Αν η κανονικοποίηση απαιτείται, το αποτέλεσμα μπορεί να αναγραφεί ως:

1.0212×10^{6}

4. Στρογγυλοποίηση

Εάν απαιτείται, εκτελείται στρογγυλοποίηση για να διασφαλιστεί ότι ο αριθμός συμμορφώνεται με τη μορφή κινητής υποδιαστολής. Σε αυτό το βήμα, το αποτέλεσμα μπορεί να είναι:

1.021×1061.021×106

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

5. Προσδιορισμός του προσήμου

Τέλος, ο προσδιορισμός του προσήμου του αποτελέσματος διαφαίνεται από τα πρόσημα των αρχικών τελεστέων. Εφόσον και οι δύο αριθμοί είναι θετικοί, το τελικό αποτέλεσμα υιοθετεί και αυτό θετικό πρόσημο.

Τελικό αποτέλεσμα:

$+1.021\times10^{6}$

<u>ΠΑΡΑΔΕΙΓΜΑ 2</u>

Αν εξετάσουμε το παράδειγμα:

- 1.000×2^{-1} (ισοδύναμο με 0.5)
- -1.110×2^{-2} (ισοδύναμο με -0.4375)

Διαδικασία Πολλαπλασιασμού

Βήμα 1.Πρόσθεση Εκθετών

Όταν πολλαπλασιάζουμε, οι εκθέτες των αριθμών προστίθενται.

Για τους μη πολωμένους εκθέτες: $-1+-2=-3$

Για τους πολωμένους εκθέτες: $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$

Εδώ αφαιρούμε την πόλωση (127) πριν εφαρμόσουμε την πρόσθεση.

Βήμα 2.Πολλαπλασιασμός Σημαντικών

Τα σημαντικά μέρη των αριθμών πολλαπλασιάζονται: $1.000\times1.110=1.110$

Το αποτέλεσμα τώρα αναπαρίσταται ως: 1.110×2^{-3}

Βήμα 3.Κανονικοποίηση Αποτελέσματος

Αφού γίνουν οι υπολογισμοί, πρέπει να ελέγξουμε αν υπάρχει ανάγκη για κανονικοποίηση και αν έχει προκύψει υπερχείλιση ή ανεπάρκεια.

Εδώ, το αποτέλεσμα είναι: 1.110×2^{-3}

Δεν υπάρχει υπερχείλιση ή ανεπάρκεια, οπότε διατηρούμε το αποτέλεσμα.

Βήμα 4.Στρογγυλοποίηση

- Αν χρειαστεί, η στρογγυλοποίηση εφαρμοζεται στο αποτέλεσμα για να το φέρει σε κανονική μορφή.
- Στην περίπτωση μας, δεν χρειάζεται να αλλάξει το αποτέλεσμα, παραμένει: $1.1102\times2-31.1102\times2-3$

Βήμα5.Προσδιορισμός Προσήμου

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Το τελευταίος βήμα είναι να καθορίσουμε το πρόσημο του αποτελέσματος. Όταν πολλαπλασιάζουμε έναν θετικό με έναν αρνητικό αριθμό το αποτέλεσμα είναι αρνητικό:

$(-1)\times(1.1102)\times 2^{-3}=-1.1102\times 2^{-3}$

Υλικό Αριθμητικής Κινητής Υποδιαστολής

Η υλοποίηση των αριθμητικών πράξεων κινητής υποδιαστολής είναι μία πολύπλοκη διαδικασία που απαιτεί ειδικά σχεδιασμένο υλικό. Παρακάτω αναλύονται οι κυριότερες πτυχές αυτού του υλικού.

Πολυπλοκότητα Πολλαπλασιασμού και Πρόσθεσης

- Ο **πολλαπλασιασμός κινητής υποδιαστολής** χαρακτηρίζεται από την ίδια πολυπλοκότητα με την **πρόσθεση κινητής υποδιαστολής**. Ενώ ο πολλαπλασιασμός απαιτεί τον υπολογισμό του γινομένου των σημαντικών, η πρόσθεση επικεντρώνεται στην αθροιστική λειτουργία αυτών.

Αναγκαίες Λειτουργίες

Κατά κανόνα, το υλικό αριθμητικής κινητής υποδιαστολής εκτελεί τις εξής βασικές λειτουργίες:

- **Πρόσθεση**: Συνδυάζει δύο ή περισσότερους αριθμούς κινητής υποδιαστολής.
- **Αφαίρεση**: Υπολογίζει τη διαφορά ανάμεσα σε δύο αριθμούς.
- **Πολλαπλασιασμός**: Υπολογίζει το γινόμενο των σημαντικών.
- **Διαίρεση**: Υπολογίζει το αποτέλεσμα μεροληπτικά.
- **Υπολογισμός Αντιστρόφων**: Εξάγει το αντίστροφο ενός αριθμού.
- **Υπολογισμός Τετραγωνικών Ριζών**: Υπολογίζει την τετραγωνική ρίζα ενός αριθμού.

Μετατροπές

- Η **μετατροπή** μεταξύ κινητής υποδιαστολής και ακέραιου αριθμού απαιτεί ειδικές διαδικασίες, αφού οι δύο αναπαραστάσεις είναι διαφορετικές. Οι ακέραιοι αριθμοί περιορίζονται σε γραμμικά διαστήματα και δεν ενσωματώνουν την έννοια της υποδιαστολής.

Απαιτήσεις Χρόνου

- Συνήθως, οι πράξεις κινητής υποδιαστολής απαιτούν **αρκετούς κύκλους ρολογιού** για την ολοκλήρωσή τους. Αυτό υποδεικνύει ότι οι πράξεις κινητής υποδιαστολής είναι πιο χρονοβόρες σε σύγκριση με τις αντίστοιχες πράξεις με ακέραιους αριθμούς.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Διοχέτευση

- Η **υλοποίηση και η διοχέτευση** είναι μέθοδοι που χρησιμοποιούνται για τη βελτίωση της αποδοτικότητας κατά την εκτέλεση πολλών πράξεων στον ίδιο κύκλο ρολογιού. Η διοχέτευση επιτρέπει σε διάφορες λειτουργίες να εκτελούνται ταυτόχρονα ή με ελαφριά επεξεργασία, μειώνοντας έτσι το συνολικό χρόνο εκτέλεσης.

Σύνοψη

Γενικά, οι διάφορες αυτές πτυχές υποδεικνύουν ότι η αριθμητική κινητής υποδιαστοχής είναι ένα σύνθετο σύστημα, το οποίο απαιτεί εξειδικευμένο υλικό ώστε να μπορεί να διαχειριστεί τις απαιτήσεις των επιστημονικών και υπολογιστικών εφαρμογών.

Εντολές Κινητής Υποδιαστολής στον RISC-V

Η αρχιτεκτονική RISC-V ενσωματώνει συγκεκριμένες εντολές για την διαχείριση αριθμών κινητής υποδιαστολής, οι οποίες είναι ιδιαίτερες και διαφέρουν από τις εντολές που χρησιμοποιούνται για τους ακέραιους. Ας δούμε αναλυτικά τα σημεία που αναφέρονται:

1. Ξεχωριστοί Καταχωρητές Κινητής Υποδιαστολής

- **Καταχωρητές:** Ο RISC-V διαθέτει 32 καταχωρητές κινητής υποδιαστολής (f0 έως f31). Αυτοί οι καταχωρητές είναι αποκλειστικά αφιερωμένοι για την αποθήκευση και την επεξεργασία δεδομένων κινητής υποδιαστολής.
- **Διπλής Ακρίβειας:** Οι καταχωρητές αυτοί υποστηρίζουν τόσο την απλή (32 bit) όσο και τη διπλή (64 bit) ακρίβεια. Οι τιμές απλής ακρίβειας καταλαμβάνουν τα 32 χαμηλότερα bit ενός καταχωρητή που μπορεί να αποθηκεύει και διπλής ακρίβειας δεδομένα.

2. Εντολές Κινητής Υποδιαστολής

- **Ειδικές Εντολές:** Οι εντολές που σχετίζονται με την κινητή υποδιαστολή επενεργούν αποκλειστικά στους καταχωρητές κινητής υποδιαστολής, πράγμα που σημαίνει ότι οι εντολές αυτές **δεν** μπορούν να εκτελούν πράξεις σε ακέραιους και αντίστροφα. Αυτό διασφαλίζει την ακεραιότητα και την ακρίβεια των υπολογισμών.

3. Φόρτωση και Αποθήκευση Δεδομένων

- **Εντολές Φόρτωσης:** Η αρχιτεκτονική RISC-V χρησιμοποιεί συγκεκριμένες εντολές για να φορτώσει (load) ή να αποθηκεύσει (store) δεδομένα κινητής υποδιαστολής στους καταχωρητές.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- flw: Φόρτωμα 32-bit αριθμού κινητής υποδιαστολής από τη μνήμη σε καταχωρητή κινητής υποδιαστολής.
- fld: Φόρτωμα 64-bit αριθμού κινητής υποδιαστολής από τη μνήμη σε καταχωρητή κινητής υποδιαστολής.
- fsw: Αποθήκευση 32-bit αριθμού κινητής υποδιαστολής από τον καταχωρητή στη μνήμη.
- fsd: Αποθήκευση 64-bit αριθμού κινητής υποδιαστολής από τον καταχωρητή στη μνήμη.

4. Επίπτωση στο Μέγεθος Κώδικα

- **Βελτιστοποίηση:** Η χρήση περισσότερων καταχωρητών κινητής υποδιαστολής έχει ελάχιστη επίπτωση στο συνολικό μέγεθος του κώδικα, ενισχύοντας την απόδοση και την αποδοτικότητα της εκτέλεσης των υπολογισμών κινητής υποδιαστολής.

5. Αριθμητικές Πράξεις

- **Πράξεις Απλής Ακρίβειας:**
  - Η RISC-V διαθέτει εντολές για την εκτέλεση απλών αριθμητικών πράξεων, οι οποίες περιλαμβάνουν:
    - fadd.s: Πρόσθεση
    - fsub.s: Αφαίρεση
    - fmul.s: Πολλαπλασιασμός
    - fdiv.s: Διαίρεση
    - fsqrt.s: Υπολογισμός τετραγωνικής ρίζας
  - Παράδειγμα εντολής: fadds.s f2, f4, f6 προσθέτει τις τιμές των f4 και f6 και αποθηκεύει το αποτέλεσμα στο f2.
- **Πράξεις Διπλής Ακρίβειας:**
  - Για τις διπλές ακριβείς πράξεις, οι αντίστοιχες εντολές περιλαμβάνουν:
    - fadd.d, fsub.d, fmul.d, fdiv.d, fsqrt.d
  - Παράδειγμα: fadd.d f2, f4, f6 προσθέτει τις τιμές των f4 και f6 και αποθηκεύει το αποτέλεσμα στο f2.

4. Σύγκριση Πράξεων

- Οι εντολές σύγκρισης κινητής υποδιαστολής περιλαμβάνουν:
  - feq.s, flt.s, fle.s (για απλή ακρίβεια)
  - feq.d, flt.d, fle.d (για διπλή ακρίβεια)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Η έξοδος από αυτές τις συγκριτικές εντολές είναι είτε 0 είτε 1, που αντιπροσωπεύει την ψευδή ή αληθή κατάσταση των συγκρίσεων.

5. Διακλάδωση με βάση Σύγκριση

- Χρησιμοποιώντας τις εντολές beq (branch if equal) και bne (branch if not equal), είναι δυνατή η διακλάδωση στον κώδικα με βάση το αποτέλεσμα των συγκρίσεων κινητής υποδιαστολής.
- Με αυτόν τον τρόπο, μπορείτε να προγραμματίσετε λογικές αποφάσεις στο πρόγραμμα σας, επιλέγοντας διαφορετικές διαδρομές εκτέλεσης ανάλογα με τα αποτελέσματα των συγκριτικών ελέγχων.

ΠΑΡΑΔΕΙΓΜΑ 1 -Μετατροπή Βαθμών °F σε Βαθμούς °C

Η διαδικασία μετατροπής θερμοκρασιών από Φαρενάιτ (°F) σε Κελσίου (°C) μπορεί να αποτυπωθεί με τη χρήση της αριθμητικής κινητής υποδιαστολής και είναι πολύ χρήσιμη σε επιστημονικά και τεχνικά περιβάλλοντα. Ας εξετάσουμε πώς πραγματοποιείται αυτή η μετατροπή χρησιμοποιώντας τον παρακάτω κώδικα.

Κώδικας C

```
1  float f2c (float fahr) {
2      return ((5.0 / 9.0) * (fahr - 32.0));
3  }
```

Ανάλυση Κώδικα:

- **Αφορά**: Συνάρτηση που δέχεται μια θερμοκρασία σε βαθμούς Φαρενάιτ (fahr).
- **Λογική**: Υπολογίζει τη θερμοκρασία σε κελσίου με βάση τον τύπο: °C=59·(°F−32)°C=95·(°F−32)

Μεταγλωττισμένος Κώδικας RISC-V

Η αντίστοιχη μεταγλώττιση σε RISC-V, που είναι μια αρχιτεκτονική υπολογιστών, είναι ως εξής:

```
f2c:
    flw f0,const5(x3)    // f0 = 5.0f
    flw f1,const9(x3)    // f1 = 9.0f
    fdiv.s f0, f0, f1    // f0 = 5.0f / 9.0f
    flw f1,const32(x3)    // f1 = 32.0f
    fsub.s f10,f10,f1     // f10 = fahr - 32.0
    fmul.s f10,f0,f10     // f10 = (5.0f / 9.0f) * (fahr - 32.0f)
    jalr x0,0(x1)         // επιστροφή
```

Ανάλυση Κώδικα RISC-V:

- **flw**: Η εντολή αυτή φορτώνει μια αριθμητική κινητής υποδιαστολής (float) από τη μνήμη στον αντίστοιχο καταχωρητή (f0, f1).

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **fdiv.s**: Εδώ πραγματοποιείται ο πολλαπλασιασμός του 5.0 με το αντίστροφο του 9.0 (διαιρεί 5.0 διά 9.0).
- **fsub.s**: Εδώ γίνεται η αφαίρεση 32 από την θερμοκρασία που παράγεται στην έξοδο (fahr).
- **fmul.s**: Πολλαπλασιάζει το αποτέλεσμα fahr - 32 με το αποτέλεσμα της προηγούμενης διαίρεσης.
- **jalr x0,0(x1)**: Η εντολή jalr χρησιμοποιείται για την επιστροφή στην εκτέλεση του προγράμματος, μετά τον υπολογισμό.

ΠΑΡΑΔΕΙΓΜΑ 2-Πολλαπλασιασμός Πινάκων στην RISC-V

Γενική Περιγραφή

Ο κώδικας RISC-V που παρουσιάζεται αναφέρεται στην υλοποίηση του αλγορίθμου πολλαπλασιασμού δύο πινάκων (matrix multiplication) χρησιμοποιώντας αριθμούς κινητής υποδιαστολής. Ο αλγόριθμος υπολογίζει το αποτέλεσμα ενός πίνακα C το οποίο είναι το αποτέλεσμα του πολλαπλασιασμού του πίνακα A με τον πίνακα B, σύμφωνα με την εξίσωση:

C=C+A×B$C=C+A\times B$

Όλα τα στοιχεία των πινάκων είναι 32x32 με διπλή ακρίβεια (64 bits).

Κώδικας σε RISC-V

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

```
L1:                     # Βρόχος για i
  li x6, 0              # x6 = j = 0 (μεσαίος βρόχος)
L2:                     # Βρόχος για j
  li x7, 0              # x7 = k = 0 (εσωτερικός βρόχος)
  # Υπολογισμός διεύθυνσης c[i][j]
  slli x30, x5, 5       # x30 = i * 32
  add x30, x30, x6      # x30 = i * 32 + j
  slli x30, x30, 3      # x30 = byte offset για c[i][j] (8 bytes ανά στοιχείο)
  add x30, x10, x30     # x30 = διεύθυνση c[i][j]
  fld f0, 0(x30)        # f0 = c[i][j]

L3:                     # Βρόχος για k
  # Υπολογισμός b[k][j]
  slli x29, x7, 5       # x29 = k * 32
  add x29, x29, x6      # x29 = k * 32 + j
  slli x29, x29, 3      # x29 = byte offset για b[k][j]
  add x29, x12, x29     # x29 = διεύθυνση b[k][j]
  fld f1, 0(x29)        # f1 = b[k][j]

  # Υπολογισμός a[i][k]
  slli x29, x5, 5       # x29 = i * 32
  add x29, x29, x7      # x29 = i * 32 + k
  slli x29, x29, 3      # x29 = byte offset για a[i][k]
  add x29, x11, x29     # x29 = διεύθυνση a[i][k]
  fld f2, 0(x29)        # f2 = a[i][k]

  fmul.d f1, f2, f1     # f1 = a[i][k] * b[k][j]
  fadd.d f0, f0, f1     # f0 = f0 + f1 (c[i][j] += a[i][k] * b[k][j])

  addi x7, x7, 1        # k++
  bltu x7, x28, L3      # αν k < 32 πήγαινε στο L3

fsd f0, 0(x30)          # αποθήκευσε f0 πίσω στο c[i][j]

addi x6, x6, 1          # j++
bltu x6, x28, L2        # αν j < 32 πήγαινε στο L2

addi x5, x5, 1          # i++
bltu x5, x28, L1        # αν i < 32 πήγαινε στο L1
```

Λειτουργίες του Κώδικα

- **Αρχικοποίηση Βρόχων**:
  - **li x28,32**: Ορίζει το μέγεθος του πίνακα (32) που θα χρησιμοποιηθεί για τις βρόχους.
  - **li x5,0**: Αρχικοποιεί τον μετρητή i που αναφέρεται στις γραμμές του πίνακα C.
  - **L1, L2, L3**: Είναι ετικέτες που χρησιμοποιούνται για τους βρόχους του αλγορίθμου.
- **Υπολογισμός Διευθύνσεων**:
  - Οι εντολές slli και add χρησιμοποιούνται για τον υπολογισμό των διευθύνσεων των στοιχείων στους πίνακες C, A, και B. Οι υπολογισμοί περιλαμβάνουν την προσαρμογή για τις σχετικές αποστάσεις και την αύξηση των μετρητών i, j, και k.

- **Διεύθυνση Πίνακα C**:
    - **slli x30,x5,5**: Υπολογίζει την γραμμή i του πίνακα C (μετατόπιση 5 bit ισοδύναμη με πολλαπλασιασμό με 32).
    - **add x30,x30,x6**: Προσθέτει την στήλη j για να βρει την ακριβή διεύθυνση του c[i][j].

- **Φόρτωμα Στοιχείων**:
    - **fld f0,0(x30)**: Φορτώνει την τιμή του c[i][j] στον καταχωρητή f0.
    - Αντίστοιχα, οι διευθύνσεις για τους πίνακες A και B υπολογίζονται με παρόμοιο τρόπο και τα στοιχεία φορτώνονται στους καταχωρητές f1 και f2.

- **Πραγματοποίηση Πράξεων**:
    - **fmul.d f1, f2, f1**: Πολλαπλασιάζει το στοιχείο a[i][k] με το b[k][j].
    - **fadd.d f0, f0, f1**: Προσθέτει το αποτέλεσμα του πολλαπλασιασμού στο στοιχείο c[i][j].

- **Επαναλήψεις**:
    - Τα addi x7,x7,1, addi x6,x6,1, addi x5,x5,1 αυξάνουν τους μετρητές k, j και i, αντίστοιχα, για την επανάληψη των βρόχων. Η συνθήκη bltu ελέγχει αν ο μετρητής είναι μικρότερος από το 32 για να επαναλάβει τα βήματα.

- **Αποθήκευση του Αποτελέσματος**:
    - **fsd f0,0(x30)**: Αποθηκεύει το τελικό αποτέλεσμα στο c[i][j].

Ακριβής Αριθμητική και Πρότυπο IEEE 754

Η ακριβής αριθμητική σε υπολογιστές είναι καθοριστική για τη διαχείριση αριθμητικών υπολογισμών, ειδικά όταν πρόκειται για αριθμούς κινητής υποδιαστολής. Το πρότυπο IEEE 754 είναι ένα από τα πιο ευρέως χρησιμοποιούμενα πρότυπα για την αναπαράσταση και τις πράξεις με αριθμούς κινητής υποδιαστολής. Ακολουθούν τα κύρια σημεία που απορρέουν από την παράγραφο που παραθέσατε:

1. Έλεγχοι Στρογγυλοποίησης

Στο πρότυπο IEEE 754 παρέχονται πρόσθετοι έλεγχοι στρογγυλοποίησης, που περιλαμβάνουν:

- **Bit-φρουρός (Guard Bit)**: Ένα επιπλέον bit που χρησιμοποιείται για να αποθηκεύει την ακρίβεια του τύπου κινητής υποδιαστολής κατά τη διάρκεια μιας αριθμητικής πράξης.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Bit Στρογγύλευσης (Round Bit)**: Χρησιμοποιείται για να καθορίζει πώς πρέπει να γίνει η στρογγυλοποίηση του τελικού αποτελέσματος.
- **Επίμονο Bit (Sticky Bit)**: Χρησιμοποιείται για να δηλώνει εάν υπάρχουν ή όχι μη μηδενικά bits που έχουν χάσει κατά τη διάρκεια μιας στρογγυλοποίησης.

2. Επιλογές Στρογγυλοποίησης

Οι προγραμματιστές έχουν τη δυνατότητα να επιλέξουν τον τρόπο στρογγυλοποίησης που θέλουν να εφαρμόσουν. Αυτές οι επιλογές επιτρέπουν στους προγραμματιστές να ελέγχουν την αριθμητική συμπεριφορά σύμφωνα με τις ανάγκες του υπολογισμού.

3. Υλοποίηση και Περιορισμοί

- **Εφαρμογή και Υλοποίηση**: Δεν υλοποιούνται όλες οι επιλογές στρογγυλοποίησης σε όλες τις μονάδες κινητής υποδιαστολής. Ορισμένες υλοποιήσεις ενδέχεται να παρέχουν μόνο τις βασικές επιλογές.
- **Προκαθορισμένες Επιλογές**: Στις περισσότερες γλώσσες προγραμματισμού και βιβλιοθήκες κινητής υποδιαστολής, χρησιμοποιούνται προκαθορισμένες επιλογές για τη στρογγυλοποίηση, προκειμένου να επιτευχθεί μια ισορροπία μεταξύ ακρίβειας και απόδοσης.

4. Ισορροπία Πολυπλοκότητας και Απόδοσης

Η ανάγκη για ακριβείς υπολογισμούς και η αυξανόμενη πολυπλοκότητα των υπολογιστικών διαδικασιών απαιτεί μια ισορροπία μεταξύ της αριθμητικής ακρίβειας και της απόδοσης του υλικού. Οι σχεδιαστές των υπολογιστικών συστημάτων λαμβάνουν υπόψη αυτές τις απαιτήσεις για να παραδώσουν ευέλικτους και αποτελεσματικούς υπολογιστικούς μηχανισμούς.

Παραλληλία και Αριθμητική Υπολογιστών

Η παραλληλία επιπέδου δεδομένων (Data-Level Parallelism) και η παραλληλία διανυσμάτων (Vector Parallelism) αναφέρονται στη δυνατότητα εκτέλεσης πολλαπλών λειτουργιών ταυτόχρονα σε συλλογές δεδομένων. Αυτή η προσέγγιση είναι κρίσιμη για την απόδοση στις εφαρμογές ήχου και γραφικών.

Βασικές έννοιες

- **Ταυτόχρονη Εκτέλεση**: Οι εφαρμογές ήχου και γραφικών, οι οποίες ασχολούνται με την επεξεργασία μεγάλων ποσοτήτων δεδομένων, μπορούν να επωφεληθούν από την ταυτόχρονη εκτέλεση εργασιών σε πολλά δεδομένα. Για παράδειγμα, μπορεί να χρειαστεί να επεξεργαστούν πολλές τιμές χρώματος ή ηχητικά σήματα ταυτόχρονα.
- **Αθροιστής 128 bit**:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Ένας αθροιστής που μπορεί να επεξεργαστεί 128 bit μπορεί να έχει πολλές διατάξεις, όπως:
    - **Δεκαέξι τελεστέοι των 8 bit**.
    - **Οκτώ τελεστέοι των 16 bit**.
    - **Τέσσερις τελεστέοι των 32 bit**.
- Αυτός ο αθροιστής μπορεί να συγκεντρώνει δεδομένα γρήγορα και αποδοτικά, με αποτέλεσμα την ενίσχυση της απόδοσης.

SIMD (Single Instruction, Multiple Data)

- Μια επαναστατική τεχνολογία που επιτρέπει σε έναν υπολογιστή να εκτελεί την ίδια εντολή σε πολλαπλά σύνολα δεδομένων ταυτόχρονα.
- Αυτό μπορεί να οδηγήσει σε σημαντική βελτίωση της απόδοσης, ιδιαίτερα για εργασίες που περιλαμβάνουν μεγάλα σύνολα ομοίων υπολογισμών, όπως:
    - **Επεξεργασία εικόνας**: Εφαρμογές που απαιτούν την εφαρμογή του ίδιου φίλτρου σε κάθε pixel.
    - **Αναπαραγωγή ήχου**: Επεξεργασία της ίδιας ηχητικής εντολής σε διαφορετικούς καναλιούς.

Σημασία στην Υπολογιστική Αριθμητική

Η παραλληλία επιπέδου δεδομένων είναι καθοριστική για την αύξηση της απόδοσης στις εφαρμογές και την αξιοποίηση της αρχιτεκτονικής των σύγχρονων επεξεργαστών, οι οποίοι είναι σχεδιασμένοι για να εκμεταλλεύονται αυτές τις δυνατότητες μέσω εξειδικευμένων εντολών και αρχιτεκτονικών.

Αυτή η προσέγγιση είναι αναγκαία καθώς η υπολογιστική δύναμη συνεχίζει να αυξάνεται, επιτρέποντας στους προγραμματιστές να δημιουργούν πιο σύνθετες και απαιτητικές εφαρμογές ωφελούμενες από τη δυνατότητα στρατηγικής επεξεργασίας δεδομένων.

Αρχιτεκτονική x86 κινητής υποδιαστολής

Η αρχιτεκτονική x86 κινητής υποδιαστολής αναφέρεται στον τρόπο που σχεδιάστηκαν και υλοποιήθηκαν οι υπολογιστές με βάση την οικογένεια αρχιτεκτονικής x86, που περιλαμβάνει πολλές εκδόσεις και βελτιώσεις της CPU.

Βασικά Στοιχεία

- **Συνεπεξεργαστής 8087**:
    - Η αρχιτεκτονική αυτή βασίστηκε αρχικά στον συνεπεξεργαστή κινητής υποδιαστολής 8087, ο οποίος σχεδιάστηκε για να εκτελεί υπολογισμούς με αριθμούς κινητής υποδιαστολής, επιτρέποντας

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

μεγαλύτερη ακρίβεια και περισσότερες αριθμητικές λειτουργίες από ότι οι υπάρχουσες CPU.

- **Καταχωρητές Επεκταμένης Ακρίβειας**:
  - Διαθέτει 8 καταχωρητές επεκταμένης ακρίβειας των 80 bit (π.χ., ST(0) έως ST(7)), που χρησιμοποιούνται για την αποθήκευση αριθμών κινητής υποδιαστολής. Αυτοί οι καταχωρητές είναι πιο μεγάλοι σε μέγεθος από τους τυπικούς (32 bit ή 64 bit) και επιτρέπουν την εκτέλεση πιο σύνθετων πράξεων.
- **Στοίβα Καταχωρητών**:
  - Οι καταχωρητές οργανώνονται σε στοίβα που επεκτείνεται προς τα κάτω, παρέχοντας έναν ευέλικτο τρόπο για τη διαχείριση των δεδομένων.
- **Δεικτοδότηση από TOS**:
  - Οι καταχωρητές δεικτοδοτούνται με βάση την κορυφή της στοίβας (Top Of Stack - TOS), διευκολύνοντας την πρόσβαση σε αυτούς ενώ επιτρέπουν την εκτέλεση πρόσθετων λειτουργιών.
- **Τιμές στη Μνήμη**:
  - Οι αριθμοί κινητής υποδιαστολής συνήθως αποθηκεύονται ως 32 bit ή 64 bit στη μνήμη, και η μετατροπή συμβαίνει κατά τη διαδικασία φόρτωσης και αποθήκευσης τους.
- **Μετατροπή Ακεραίων και Δυσκολίες Κωδικοποίησης**:
  - Στη διαδικασία αυτή, οι ακέραιοι τελεστέοι μπορούν επίσης να μετατρέπονται κατά την αποθήκευση και φόρτωση. Αυτό προσθέτει μια επιπλέον πολυπλοκότητα στην διαχείριση και υλοποίηση του κώδικα κινητής υποδιαστολής.
- **Προβλήματα Απόδοσης**:
  - Παρά την ευελιξία και την ισχυρή αρχιτεκτονική, η απόδοση του συστήματος μπορεί να είναι κακή. Η πολυπλοκότητα των λειτουργιών και η ανάγκη για μετατροπές μπορούν να οδηγήσουν σε καθυστερήσεις και λιγότερη αποδοτικότητα στον κώδικα.

Συμπεράσματα

Η αρχιτεκτονική x86 για κινητή υποδιαστολή προσφέρει πολλές δυνατότητες, όμως η πολυπλοκότητα της μπορεί να προκαλέσει προκλήσεις στην απόδοση. Οι συγκεκριμένες επιλογές για την οργάνωση των καταχωρητών, οι μετατροπές και η διαχείριση της μνήμης καθιστούν τις εφαρμογές κινητής υποδιαστολής πιο περίπλοκες αλλά και πιο ισχυρές.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Εντολές κινητής υποδιαστολής στην αρχιτεκτονική x86

Μεταφορά δεδομένων	Αριθμητικές πράξεις	Σύγκριση	Υπερβατικές συναρτήσεις
FILD mem/ST(i) FISTP mem/ST(i) FLDPI FLD1 FLDZ	FIADDP mem/ST(i) FISUBRP mem/ST(i) FIMULP mem/ST(i) FIDIVRP mem/ST(i) FSQRT FABS FRNDINT	FICOMP FIUCOMP FSTSW AX/mem	FPATAN F2XMI FCOS FPTAN FPREM FPSIN FYL2X

- Προαιρετικές παραλλαγές
  - I: ακέραιος τελεστέος
  - P: εξαγωγή τελεστέου από τη στοίβα
  - R: αντίστροφη σειρά τελεστέων
  - Δεν επιτρέπονται όλοι οι συνδυασμοί

Η εικόνα που παρέχετε δείχνει πίνακα με **εντολές κινητής υποδιαστολής (floating point)** στην αρχιτεκτονική **x86**. Οι εντολές αυτές χρησιμοποιούνται από τον **συνεπεξεργαστή κινητής υποδιαστολής** (FPU – Floating Point Unit) της x86 για πράξεις με αριθμούς κινητής υποδιαστολής.

Ας κάνουμε μια ανάλυση του περιεχομένου:

◆ **Κατηγορίες εντολών:**

1. Μεταφορά δεδομένων

- FILD mem/ST(i) – Φόρτωση ακέραιου από μνήμη
- FISTP mem/ST(i) – Αποθήκευση ακέραιου στη μνήμη και αφαίρεση από στοίβα
- FLDPI, FLD1, FLDZ – Φορτώνουν σταθερές (π, 1, 0)

2. Αριθμητικές πράξεις

- FIADD, FISUB, FIMUL, FIDIV – Ακέραιες πράξεις με δεδομένα από τη μνήμη ή ST(i)
- FSQRT – Τετραγωνική ρίζα
- FABS – Απόλυτη τιμή
- FRNDINT – Στρογγυλοποίηση στον πλησιέστερο ακέραιο

ΕΛΕΝΗ ΖΑΦΕ

3. Σύγκριση

- FICOMP, FIUCOMP – Σύγκριση ακέραιων
- FCOM, FCOMP – Σύγκριση κινητής υποδιαστολής
- FSTSW AX/mem – Αποθήκευση της κατάστασης σύγκρισης

4. Υπερβατικές συναρτήσεις

- FPATAN, F2XM1, FCOS, FSIN, FPTAN, FPREM, FPSIN, FYL2X – Διάφορες μαθηματικές συναρτήσεις, όπως τριγωνομετρικές και λογαριθμικές.

🔹 **Προαιρετικές παραλλαγές (σημειώσεις):**

- I: Ο τελεστέος είναι **ακέραιος**
- P: Ο τελεστέος **εξάγεται** από τη στοίβα μετά τη χρήση (π.χ. FISTP)
- R: **Αντιστροφή** της σειράς των τελεστέων (σημαντικό για πράξεις όπου η σειρά μετράει, όπως η αφαίρεση ή διαίρεση)
- ⚠️ Δεν επιτρέπονται όλοι οι συνδυασμοί αυτών των παραλλαγών

Επεκτάσεις Streaming SIMD Extension 2 (SSE2)

Βασική Αρχή

Η τεχνολογία Streaming SIMD Extension 2 (SSE2) είναι μια αρχιτεκτονική επεξεργαστή που επιτρέπει την εκτέλεση πολλαπλών λειτουργιών με μία μόνο εντολή. Αξιοποιεί την παραλληλία σε επίπεδο δεδομένων, επιτρέποντας στους επεξεργαστές να εκτελούν τις ίδιες ή παρόμοιες πράξεις ταυτόχρονα σε διάφορα δεδομένα.

Χαρακτηριστικά

- **128-bit Καταχωρητές**: Η SSE2 ενσωματώνει 128-bit καταχωρητές, οι οποίοι μπορούν να συλλέγουν και να επεξεργάζονται πολλαπλά δεδομένα ταυτόχρονα. Στην αρχιτεκτονική AMD64/EM64T, το πλήθος των καταχωρητών επεκτείνεται στους 8, παρέχοντας μεγαλύτερη ευελιξία και απόδοση.
- **Πολλαπλά Δεδομένα**:
  - **Διπλή Ακρίβεια**: Καθιστά εφικτό τον χειρισμό 2 τελεστέων κινητής υποδιαστολής διπλής ακρίβειας (64 bit) σε έναν καταχωρητή των 128 bit.
  - **Απλή Ακρίβεια**: Επιτρέπει την επεξεργασία 4 τελεστέων κινητής υποδιαστολής απλής ακρίβειας (32 bit) στον ίδιο καταχωρητή.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Λειτουργίες

Οι εντολές SSE2 επιτρέπουν την εκτέλεση πολλών επιμέρους πράξεων (όπως πρόσθεση, πολλαπλασιασμός κ.λπ.) ταυτόχρονα. Αυτό μειώνει τον αριθμό των κύκλων ρολογιού που απαιτούνται για την εκτέλεση εργασιών που διαφορετικά θα απαιτούσαν ξεχωριστές εντολές για κάθε τελεστή.

Πλεονεκτήματα

- **Αυξημένη Απόδοση**: Η δυνατότητα εκτέλεσης πολλαπλών διαδικασιών με μία ενοποιημένη εντολή επιταχύνει τις υπολογιστικές διαδικασίες, ειδικά σε εφαρμογές που απαιτούν έντονη αριθμητική επεξεργασία όπως γραφικά, ήχος και επιστημονικούς υπολογισμούς.
- **Βελτιστοποίηση Πόρων**: Η χρήση περισσότερων κατωτέρων καταχωρητών περιορίζει τη ζήτηση για μνήμη και επιταχύνει την προσπέλαση δεδομένων.

Συμπέρασμα

Η SSE2 είναι μια ισχυρή τεχνολογία που ενσωματώνει την παραλληλία σε επίπεδο δεδομένων και έχει εφαρμόσει σημαντικές βελτιώσεις στην ταχύτητα και την αποδοτικότητα των υπολογιστικών μονάδων, ιδίως σε εφαρμογές που απαιτούν έντονη χρήση αριθμητικών υπολογισμών.

Μη βελτιστοποιημένος κώδικας

```
void dgemm(int n, double* A, double* B, double* C) {
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) {
            double cij = C[i + j * n];  // C[i][j]
            for (int k = 0; k < n; ++k)
                cij += A[i + k * n] * B[k + j * n];  // A[i][k] *
            C[i + j * n] = cij;  // C[i][j] = cij
        }
}
```

ΕΛΕΥΖΑΦΕΙΡΗ

Βελτιστοποιημένος κώδικας

```
#include <x86intrin.h>

void dgemm(int n, double* A, double* B, double* C) {
    for (int i = 0; i < n; i += 8) {
        for (int j = 0; j < n; ++j) {
            __m512d c0 = _mm512_load_pd(C + i + j * n);  // C[i][j]..C[i+7][j]
            for (int k = 0; k < n; ++k) {
                // Φόρτωση B[k][j] και αναπαραγωγή του σε όλα τα lanes
                __m512d bb = _mm512_broadcastsd_pd(_mm_load_sd(B + k + j * n));

                // Φόρτωση 8 στοιχείων A[i..i+7][k] και υπολογισμός
                __m512d aa = _mm512_load_pd(A + i + k * n);
                c0 = _mm512_fmadd_pd(aa, bb, c0);  // c0 += aa * bb
            }
            _mm512_store_pd(C + i + j * n, c0);  // Αποθήκευση πίσω στο C
        }
    }
}
```

Βελτιστοποιημένος κώδικας συμβολικής γλώσσας για αρχιτεκτονική x86:

```
multiply.asm*   διαιρεση.asm*   ri
vmovapd (%r11),%zmm1
mov %rbx,%rcx
xor %eax,%eax
vbroadcastsd (%rax,%r8,8),%zmm0
add $0x8,%rax
vfmadd231pd (%rcx),%zmm0,%zmm1

add %r9,%rcx
cmp %r10,%rax
jne 50 <dgemm+0x50>
add $0x1, %esi
vmovapd %zmm1, (%r11
```

Δεξιά Ολίσθηση και Διαίρεση

Η έννοια της ολίσθησης (shift) σε δυαδικούς αριθμούς μπορεί να χρησιμοποιηθεί για τη διευκόλυνση των αριθμητικών υπολογισμών στον υπολογιστή, όπως η πολλαπλασία και η διαίρεση με δύναμη του 2.

Αριστερή Ολίσθηση

- Μια εντολή αριστερής ολίσθησης (left shift) σε έναν ακέραιο αριθμό ισοδυναμεί με πολλαπλασιασμό του αριθμού αυτού με 2^n, όπου n είναι ο αριθμός των θέσεων ολίσθησης.
- **Παράδειγμα:** Αν έχουμε τον αριθμό 33 (σε δυαδική αναπαράσταση: 00000011) και το ολισθήσουμε αριστερά δυο θέσεις (3 << 2), το αποτέλεσμα θα είναι 00001100, που ισοδυναμεί με 12 (δηλαδή 3×4).

Δεξιά Ολίσθηση

- Μια εντολή δεξιάς ολίσθησης (right shift) είναι αντίστοιχα μια μορφή διαίρεσης με δύναμη του 2, αλλά αυτό ισχύει μόνο για **μη προσημασμένους** ακέραιους.
- Για παράδειγμα, αν έχουμε τον αριθμό 88 (σε δυαδική: 00001000) και το ολίσθησουμε δεξιά μια θέση (8 >> 1), το αποτέλεσμα θα είναι 00000100, που ισοδυναμεί με 4 (δηλαδή 8÷2).

Διαχείριση Προσημασμένων Ακεραίων

- Για τους **προσημασμένους** ακεραίους, η δεξιά ολίσθηση μπορεί να πραγματοποιηθεί με δύο διαφορετικούς τρόπους:
  - **Αριθμητική (sign-extending) δεξιά ολίσθηση:** Σε αυτή την περίπτωση, το bit που βρίσκεται στην πιο αριστερή θέση (το "σημειό" bit) αναπαράγεται, ώστε να διατηρείται το πρόσημο του αριθμού. Έτσι, όταν ο αριθμός είναι αρνητικός, οι θέσεις που προστίθενται από αριστερά γίνονται 1.
  - **Λογική δεξιά ολίσθηση:** Διενεργεί μια απλή μετακίνηση των bits, χωρίς αναπαραγωγή του sign bit. Τα νέα bits που προστίθενται στα αριστερά γίνονται πάντα 0.
- **Παράδειγμα**:
  - Για τον αριθμό -5 (σε δυαδική αναπαράσταση: 11111011):
    - −5 >> 2 → 11111110 (αριθμητική ολίσθηση), το αποτέλεσμα είναι −2.
    - −5 >>> 2→ 00111110 (λογική ολίσθηση), το αποτέλεσμα είναι 62.

✅ Πίνακας σύνοψης:

Τύπος ολίσθησης	Προς τα δεξιά	Τι μπαίνει μπροστά (αριστερά);	Διατηρεί το πρόσημο;
Λογική (`>>>`)	✅ Ναι	Πάντα 0	❌ Όχι (αλλάζει πρόσημο)
Αριθμητική (`>>`)	✅ Ναι	Ίδιο με το bit του πρόσημου (MSB)	✅ Ναι

Συμπέρασμα

Η ολίσθηση με επιλεγμένα πρόσημα επιτρέπει στους υπολογιστές να εκτελούν αποτελεσματικές αριθμητικές πράξεις και είναι χρήσιμη για τη κατανόηση των βημάτων επεξεργασίας στην αρχιτεκτονική υπολογιστών.

ΑΕΝΗ ΖΑΦΕΙΡΗ

Προσεταιριστικότητα και Παράλληλη Εκτέλεση

Η προσεταιριστικότητα αναφέρεται στην ιδιότητα των αριθμητικών πράξεων ότι η σειρά με την οποία εκτελούνται οι πράξεις δεν πρέπει να επηρεάζει το τελικό αποτέλεσμα. Αυτό σημαίνει ότι για όλες τις πράξεις α + β + γ, το αποτέλεσμα θα πρέπει να είναι το ίδιο, ανεξαρτήτως της σειράς που θα επιλεγεί (π.χ. (α + β) + γ = α + (β + γ)).

Προβλήματα στην Παράλληλη Εκτέλεση

- **Διαπλέκοντας Λειτουργίες**: Όταν προγράμματα σχεδιάζονται για παράλληλη εκτέλεση, οι διαφορετικές διαδικασίες μπορεί να εκτελούνται ταυτόχρονα. Αυτό μπορεί να οδηγήσει σε απρόβλεπτη σειρά εκτέλεσης λειτουργιών, ενδεχομένως να παραβιάσει την προσεταιριστικότητα.
- **Συγκρούσεις και Υπολογιστικές Αταξίες**: Επιπλέον, αν γίνουν αλλαγές στους τιμές των μεταβλητών κατά τη διάρκεια των υπολογισμών, τα αποτελέσματα μπορεί να μην είναι αναμενόμενα. Για παράδειγμα, αν χρησιμοποιηθούν τα παραδείγματα με τις τιμές:
    - x = -1.50E+38
    - y = 1.50E+38
    - z = 1.0

Η επεξεργασία μπορεί να παράξει διαφορετικά αποτελέσματα ανάλογα με τη σειρά των πράξεων.

Παράδειγμα Επικοινωνίας

Με την πρόσθεση:

- Αν υπολογίσουμε πρώτα ((x + y) + z), το αποτέλεσμα θα είναι

  ((−1.50E+38 + 1.50E+38) + 1.0 = 0.0 + 1.0 = 1.0).
- Αν υπολογίσουμε (x + (y + z)), το αποτέλεσμα θα είναι (−1.50E+38 + (1.50E+38 + 1.0) = −1.50E+38 + 1.50E+38 = 0.0).

Στην πράξη, αυτό σημαίνει ότι **η σειρά εκτέλεσης επηρεάζει τη διαχείριση της ακρίβειας και την απόδοση**, γεγονός που μπορεί να οδηγήσει σε σημαντικά λάθη, ιδιαίτερα σε υπολογισμούς που απαιτούν αυξημένη ακρίβεια.

Συμπέρασμα

Οι προγραμματιστές που εργάζονται με παράλληλες διαδικασίες πρέπει να είναι προσεκτικοί για να διασφαλίσουν ότι οι υπολογισμοί τους παραλαμβάνουν την αναμενόμενη σειρά των εντολών και να επικυρώνουν τα προγράμματά τους ώστε να αποφεύγονται ενδεχόμενες ανακολουθίες στην αλγοριθμική λογική λόγω παράλληλης εκτέλεσης.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Ποιοι ενδιαφέρονται για την ακρίβεια των πράξεων κινητής υποδιαστολής;

Η ακριβής αριθμητική στις πράξεις κινητής υποδιαστολής αποτελεί κρίσιμη παράμετρο στον τομέα των υπολογιστών, ιδιαίτερα για επιστημονικές και τεχνικές εφαρμογές. Ωστόσο, η σημασία της μπορεί να διαφέρει ανάλογα με την εφαρμογή και το περιστατικό:

Σημαντικότητα της Ακρίβειας:

- **Επιστημονικές Εφαρμογές:**
  - Στις επιστημονικές υπολογιστικές εφαρμογές, η ακρίβεια των αριθμητικών υπολογισμών είναι απαραίτητη. Σφάλματα ακόμα και σε μικρότερα επίπεδα μπορεί να έχουν σωρευτικά αποτελέσματα που οδηγούν σε σοβαρές ανακρίβειες στα τελικά αποτελέσματα.
  - Η ανάγκη για υψηλή στήριξη ακριβείας είναι ιδιαίτερα σημαντική σε τομείς όπως η φυσική, η μηχανική, και η χρηματοοικονομική ανάλυση.
- **Καθημερινές Εφαρμογές:**
  - Για καθημερινές εφαρμογές και χρήσεις όπως η διαχείριση λογαριασμών ή οι επιχειρηματικές εφαρμογές, η ακρίβεια είναι επίσης σημαντική, αλλά μπορεί να μη θεωρείται το ίδιο κρίσιμη. Μικρές μεταβολές μπορεί να μην έχουν τόσο αντίκτυπο στις περισσότερο πρακτικές εφαρμογές.
  - Ένα εικονιστικό παράδειγμα είναι το:

«Το υπόλοιπο του τραπεζικού μου λογαριασμού διαφέρει κατά 0.0002 σεντ!». Αυτό δείχνει ότι ενώ υπάρχει διαφορά, ίσως να μην είναι κρίσιμη για τον χρήστη.

Ιστορικό Σφάλματος:

- Το **σφάλμα FDIV** στους επεξεργαστές Pentium της Intel είναι ένα χαρακτηριστικό παράδειγμα που δείχνει πόσο ευάλωτοι μπορεί να είναι οι υπολογιστές σε αριθμητικά προβλήματα. Ο συγκεκριμένος επεξεργαστής είχε προβλήματα με την ελάχιστη ακρίβεια κατά την εκτέλεση διαίρεσης των κινητών αριθμών, προκαλώντας αμφισβήτηση της αξιοπιστίας του.

Καταναλωτική Πήξη:

- Στην καταναλωτική αγορά, η ακρίβεια στις πράξεις κινητής υποδιαστολής θεωρείται δεδομένη, και οι χρήστες συχνά δεν είναι ενήμεροι για τα πιθανά προβλήματα ακρίβειας. Αυτό μπορεί να έχει αντίκτυπο σε μια σειρά εφαρμογών, εάν οι μηχανές δεν έχουν ανθεκτικότητα στον τομέα αυτό.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Συμπερασματικές Παρατηρήσεις σχετικά με τους Υπολογιστές

Αυτές οι παρατηρήσεις παρέχουν βασικές πληροφορίες σχετικά με την αριθμητική και την αναπαράσταση των δεδομένων στους υπολογιστές. Ακολουθεί η ανάλυση τους:

1. Οι Σειρές των Bit δεν έχον Εσωτερικό Νόημα

- Οι σειρές των μπιτ (0s και 1s) που χρησιμοποιούνται για την αποθήκευση και επεξεργασία πληροφοριών δεν έχουν αυτόνομο νόημα από μόνες τους. Η σημασία τους προκύπτει μόνο μέσω της ερμηνείας τους από τις εντολές που εκτελούνται.

2. Η ερμηνεία εξαρτάται από τις Εντολές

- Η ερμηνεία των δεδομένων εξαρτάται πλήρως από το πώς αυτές οι σειρές των bit γίνονται αντιληπτές κατά την εκτέλεση συγκεκριμένων εντολών σε ένα πρόγραμμα. Για παράδειγμα, η ίδια σειρά μπορεί να ερμηνευθεί ως ακέραιος αριθμός ή πραγματικός αριθμός κινητής υποδιαστολής, ανάλογα με την εντολή που χρησιμοποιείται.

3. Αναπαράστασεις Αριθμών στους Υπολογιστές

- Οι αριθμοί στους υπολογιστές αναπαριστώνται μέσω συγκεκριμένων προτύπων (π.χ., δισδιάστατες αναπαραστάσεις, σημείο κινητής υποδιαστολής). Αυτές οι αναπαραστάσεις καθορίζουν πώς αποθηκεύονται και επεξεργάζονται οι αριθμητικές τιμές, και είναι κρίσιμες για τη λειτουργία των υπολογιστικών συστημάτων.

4. Πεπερασμένο Εύρος και Πεπερασμένη Ακρίβεια

- Οι υπολογιστές διαθέτουν πεπερασμένα εύρη αναπαραστάσεων, που σημαίνει ότι η ποσότητα των αριθμών που μπορούν να αντιπροσωπεύσουν είναι περιορισμένη. Αυτή η περιορισμένη αναπαράσταση έχει άμεσες συνέπειες στην ακρίβεια των υπολογισμών, καθώς μπορεί να προκύψουν σφάλματα λόγω στρογγυλοποιήσεων ή του ότι κάποιες τιμές δεν μπορούν να αναπαρασταθούν με ακρίβεια.

5. Σημασία για την Προγραμματιστική Ανάπτυξη

- Αυτές οι παρατηρήσεις είναι κρίσιμες για τους προγραμματιστές, καθώς πρέπει να είναι ενήμεροι για τους περιορισμούς και τις δυνατότητες του υλικού τους κατά την ανάπτυξη λογισμικού. Η ακρίβεια των υπολογισμών και η σωστή ερμηνεία των δεδομένων είναι ζωτικής σημασίας για την επιχειρησιακή και επιστημονική χρήση.

Συμπέρασμα

Η κατανόηση της φύσης των δεδομένων και της αναπαράστασής τους στους υπολογιστές είναι θεμελιώδης για την ανάπτυξη αξιόπιστου και αποδοτικού

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

λογισμικού. Οι προγραμματιστές και οι μηχανικοί συστημάτων πρέπει να λαμβάνουν υπόψη αυτούς τους παράγοντες για την αποφυγή σφαλμάτων και την εξασφάλιση ορθών υπολογιστικών διαδικασιών.

ΜΕΤΑΤΡΟΠΕΣ ΑΡΙΘΜΩΝ

1️⃣ Από **Δεκαδικό** (βάση 10) → σε **Δυαδικό** (βάση 2)

Διαδικασία:

- Διαιρείς τον αριθμό με το 2.
- Κρατάς το υπόλοιπο (0 ή 1).
- Διαιρείς το πηλίκο ξανά με το 2 και συνεχίζεις μέχρι να φτάσεις πηλίκο 0.
- Διαβάζεις τα υπόλοιπα **από κάτω προς τα πάνω**.

Παράδειγμα: Να μετατρέψουμε το **13** σε δυαδικό:

Διαίρεση	Πηλίκο	Υπόλοιπο
13 ÷ 2	6	1
6 ÷ 2	3	0
3 ÷ 2	1	1
1 ÷ 2	0	1

Διαβάζουμε ανάποδα:
→ $13_{10} = 1101_2$

ΕΛΕΝΗ ΛΙΡΗ

2️⃣ Από **Δυαδικό** (βάση 2) → σε **Δεκαδικό** (βάση 10)

Διαδικασία:

- Πολλαπλασιάζεις κάθε ψηφίο του δυαδικού με 2^n, όπου n η θέση του ψηφίου από δεξιά προς αριστερά (ξεκινώντας από 0).
- Προσθέτεις όλα τα αποτελέσματα.

Παράδειγμα: Να μετατρέψουμε το **1101_2** σε δεκαδικό:

Ψηφίο	Θέση (n)	Υπολογισμός
1	3	$1 \times 2^3 = 8$
1	2	$1 \times 2^2 = 4$
0	1	$0 \times 2^1 = 0$
1	0	$1 \times 2^0 = 1$

Άθροισμα:
→ 8 + 4 + 0 + 1 = **13**

→ Άρα **$1101_2 = 13_{10}$**

3️⃣ Από **Δεκαδικό** (βάση 10) → σε **Δεκαεξαδικό** (βάση 16)

Διαδικασία:

- Διαιρείς τον αριθμό με το 16.
- Κρατάς το υπόλοιπο (0–15, όπου 10=A, 11=B, 12=C, 13=D, 14=E, 15=F).
- Διαιρείς το πηλίκο ξανά με 16 μέχρι να φτάσεις πηλίκο 0.
- Διαβάζεις τα υπόλοιπα από κάτω προς τα πάνω.

Παράδειγμα: Να μετατρέψουμε το **254** σε δεκαεξαδικό:

Διαίρεση	Πηλίκο	Υπόλοιπο
254 ÷ 16	15	14 (E)
15 ÷ 16	0	15 (F)

Διαβάζουμε ανάποδα:
→ **$254_{10} = FE_{16}$**

4️⃣ Από **Δεκαεξαδικό** (βάση 16) → σε **Δεκαδικό** (βάση 10)

Διαδικασία:

- Μετατρέπεις κάθε ψηφίο σε αριθμό.
- Πολλαπλασιάζεις κάθε ψηφίο με 16^n, όπου n η θέση από δεξιά προς αριστερά (ξεκινώντας από 0).
- Προσθέτεις όλα τα αποτελέσματα.

Παράδειγμα: Να μετατρέψουμε το **FE_{16}** σε δεκαδικό:

Ψηφίο	Θέση (n)	Υπολογισμός
F=15	1	$15 \times 16^1 = 240$
E=14	0	$14 \times 16^0 = 14$

Άθροισμα:
→ 240 + 14 = **254**

→ Άρα **FE_{16} = 254_{10}**

5️⃣ Από **Δυαδικό** → σε **Δεκαεξαδικό**

Διαδικασία:

- Χωρίζεις το δυαδικό ψηφίο σε ομάδες των **4 bits** από δεξιά προς τα αριστερά.
- Κάθε ομάδα 4 bits αντιστοιχεί σε ένα δεκαεξαδικό ψηφίο.

Παράδειγμα: Να μετατρέψουμε το **11011110_2** σε δεκαεξαδικό:

- Ομάδες 4 bits: (0001) (1011) (1110)
  (συμπληρώνουμε μηδενικά μπροστά αν χρειάζεται)

Ομάδα 4 bits	Αντιστοιχία δεκαεξαδική
0001	1
1011	B
1110	E

→ **11011110_2 = $1BE_{16}$**

6 Από Δεκαεξαδικό → σε Δυαδικό

Διαδικασία:

- Κάθε δεκαεξαδικό ψηφίο μετατρέπεται **σε 4 δυαδικά ψηφία**.

Παράδειγμα: Να μετατρέψουμε το **$1BE_{16}$** σε δυαδικό:

Δεκαεξαδικό	Δυαδικό
1	0001
B=11	1011
E=14	1110

→ **$1BE_{16} = 000110111110_2$**

(Αν δεν θέλεις τα αρχικά μηδενικά, μπορείς να τα παραλείψεις)

📝 Συνοπτικά:

Μετατροπή	Βήματα
Δεκαδικό → Δυαδικό	Διαίρεση με 2
Δυαδικό → Δεκαδικό	Πολλαπλασιασμός με 2^n
Δεκαδικό → Δεκαεξαδικό	Διαίρεση με 16
Δεκαεξαδικό → Δεκαδικό	Πολλαπλασιασμός με 16^n
Δυαδικό → Δεκαεξαδικό	Ομαδοποίηση ανά 4 bits
Δεκαεξαδικό → Δυαδικό	Κάθε ψηφίο → 4 bits

–ΑΠΟ ΚΥ ΣΕ ΔΥΑΔΙΚΟ

🔢 Βήμα 1: Διάσπαση σε ακέραιο και κλασματικό μέρος

Έστω ότι έχεις τον αριθμό **N = 5.75**:

- Ακέραιο μέρος: 5
- Κλασματικό μέρος: 0.75

🔹 Βήμα 2: Μετατροπή του ακέραιου μέρους σε δυαδικό

Διαίρεσε επαναληπτικά με το 2, και κράτα τα υπόλοιπα (από κάτω προς τα πάνω):

- 5 ÷ 2 = 2 υπόλ. 1
- 2 ÷ 2 = 1 υπόλ. 0
- 1 ÷ 2 = 0 υπόλ. 1
  ⇒ **5 = 101_2**

🔁 Βήμα 3: Μετατροπή του κλασματικού μέρους

Πολλαπλασίασε διαδοχικά επί 2 και κράτα το **ακέραιο μέρος κάθε φορά**:

- 0.75 × 2 = 1.5 → κρατάμε 1, μένει 0.5
- 0.5 × 2 = 1.0 → κρατάμε 1, μένει 0 → σταματάμε
  ⇒ **0.75 = $.11_2$**

🧩 Βήμα 4: Συνδυασμός

Συνδυάζεις τα δύο:

$$5.75_{10} = 101.11_2$$

Ας μετατρέψουμε **το 6.2** σε **δυαδική μορφή** βήμα-βήμα:

🔹 Βήμα 1: Διάσπαση

- Ακέραιο μέρος: **6**
- Κλασματικό μέρος: **0.2**

🔹 Βήμα 2: Δυαδικό του ακέραιου μέρους (6)

Διαίρεση με το 2:

- 6 ÷ 2 = 3, υπόλοιπο **0**
- 3 ÷ 2 = 1, υπόλοιπο **1**
- 1 ÷ 2 = 0, υπόλοιπο **1**

Άρα:

$$6_{10} = 110_2$$

🔹 Βήμα 3: Δυαδικό του κλασματικού μέρους (0.2)

Πολλαπλασιασμοί επί 2:

1. 0.2 × 2 = 0.4 → **0**
2. 0.4 × 2 = 0.8 → **0**
3. 0.8 × 2 = 1.6 → **1**
4. 0.6 × 2 = 1.2 → **1**
5. 0.2 × 2 = 0.4 → **0** (επαναλαμβάνεται)

➤ Επαναλαμβάνεται κυκλικά: **0011**

Άρα:

$$0.2_{10} \approx 0.001100110011\ldots\ (_2)$$

🔹 Βήμα 4: Συνδυασμός

$$6.2_{10} \approx 110.001100110011\ldots\ (_2)$$

Ας αναπαραστήσουμε τον αριθμό **−0.75** σε μορφή **IEEE 754 (32-bit, single precision)** βήμα-βήμα:

🔹 Βήμα 1: Πρόσημο

Ο αριθμός είναι **αρνητικός**, άρα:
Sign bit = 1

🔹 Βήμα 2: Δυαδική μορφή του 0.75

$$0.75_{10} = 0.11_2$$

🔹 Βήμα 3: Κανονικοποίηση σε μορφή $1.xxxx \times 2^e$

$$0.11_2 = 1.1_2 \times 2^{-1}$$

➤ Άρα:

- **Mantissa**: τα bits μετά το δεκαδικό: **100000...**
- **Exponent**: -1

🔹 Βήμα 4: Εκθέτης με bias

Για IEEE 754 single precision, το **bias είναι 127**.

$$-1 + 127 = 126 \Rightarrow \text{Exponent} = 01111110_2$$

🔹 Βήμα 5: Mantissa (23 bits)

Μετά το "1." από την κανονικοποίηση έχουμε:

$$.1 \Rightarrow mantissa = 10000000000000000000000$$

✅ Τελική μορφή (32 bits)

Sign	Exponent	Mantissa
1	01111110	10000000000000000000000

Άρα:

$$\boxed{1\ 01111110\ 10000000000000000000000}$$

(σε δυαδικό IEEE 754 -32 bit)

S	Εκθέτης	Κλάσμα

$$(-1)^{\text{πρόσημο}} \times (1 + \text{κλάσμα}) \times 2^{(\text{εκθέτης} - \text{πόλωση})}$$

ΔΦΕΙΡΗ

ΑΠΟ ΔΥΑΔΙΚΟ ΣΕ IEEE 754

🔢 Δοσμένος δυαδικός αριθμός:

Αντιγραφή Επεξεργασία

```
11000000101000...00
```

Αυτός είναι ένας αριθμός 32-bit. Τον χωρίζουμε σε 3 μέρη:

Πεδίο	Bits	Ερμηνεία
S (sign)	`1`	Αρνητικός αριθμός
Exponent	`10000001`	Δυαδικό για το **129**
Mantissa	`01000...00`	Το κλασματικό μέρος (23 bits)

🧮 Βήμα 1: Ερμηνεία των πεδίων

- Sign bit S = 1 → αρνητικός αριθμός
- Exponent = 129, αλλά χρησιμοποιούμε bias 127, άρα:

$$E = 129 - 127 = 2$$

- Mantissa = 1 + 0.25 = 1.25
  (Το `01000...` σημαίνει ότι μόνο το 2^{-2} = 0.25 είναι ενεργό)

📐 Υπολογισμός αριθμού

Η γενική μορφή είναι:

$$x = (-1)^S \times (1 + mantissa) \times 2^E$$

Άρα:

$$x = (-1)^1 \times 1.25 \times 2^2 = -1 \times 1.25 \times 4 = -5.0$$

✅ **Απάντηση: –5.0**

ΣΥΜΠΛΗΡΩΜΑ ΩΣ ΠΡΟΣ ΔΥΟ

1. Τι είναι το συμπλήρωμα ως προς 2;

Είναι **ένας τρόπος αναπαράστασης αρνητικών αριθμών** στο δυαδικό σύστημα.

- Οι **θετικοί αριθμοί** γράφονται κανονικά σε δυαδικό.
- Οι **αρνητικοί αριθμοί** γράφονται χρησιμοποιώντας μια ειδική τεχνική: το **συμπλήρωμα ως προς 2**.

Έτσι, οι υπολογιστές μπορούν να κάνουν **εύκολα πρόσθεση και αφαίρεση** με τον ίδιο μηχανισμό.

---

2. Γιατί χρησιμοποιούμε συμπλήρωμα ως προς 2;

- **Για να υποστηρίζουμε αρνητικούς αριθμούς** στα bits.
- **Για να κάνουμε πρόσθεση και αφαίρεση** πιο απλή: δεν χρειάζεται ξεχωριστό κύκλωμα για το μείον (-).
- Το πιο σημαντικό: με το συμπλήρωμα ως προς 2, **η αφαίρεση γίνεται σαν πρόσθεση**!

---

3. Πώς φτιάχνουμε το συμπλήρωμα ως προς 2;

Για να βρεις το συμπλήρωμα ως προς 2 ενός αριθμού:

1. **Βρες το κανονικό δυαδικό** του αριθμού (χωρίς πρόσημο).
2. **Κάνε αντιστροφή των bits** (0 → 1 και 1 → 0).
3. **Πρόσθεσε 1** στο αποτέλεσμα.

---

4. Παράδειγμα: Βρες το συμπλήρωμα ως προς 2 του -5 σε 8 bits

Βήμα 1: Δυαδική μορφή του 5

5_{10} = 00000101_2 (σε 8 bits)

Βήμα 2: Αντιστροφή όλων των bits

0 → 1, 1 → 0

00000101 → 11111010

Βήμα 3: Πρόσθεση 1

11111010 + 1 = 11111011

Άρα το -5 σε συμπλήρωμα ως προς 2 είναι:
→ 11111011_2

5. Σημαντικές Παρατηρήσεις

- **Το πιο αριστερό bit (MSB)** είναι το **bit πρόσημου**:
  - **0** σημαίνει θετικός αριθμός
  - **1** σημαίνει αρνητικός αριθμός
- **Εύρος τιμών σε N bits**:
  - Θετικοί: από 0 έως 2^{n-1} - 1
  - Αρνητικοί: από -2^{n-1} έως -1

Παράδειγμα για 8 bits:

- Θετικοί: 0 έως 127
- Αρνητικοί: -128 έως -1

6. Πώς κάνω πρόσθεση με συμπλήρωμα ως προς 2

- Αν προσθέσεις δυο αριθμούς (θετικούς ή αρνητικούς) **κανονικά**, χωρίς ειδικούς κανόνες.
- **Αν βγει επιπλέον κρατούμενο**, το αγνοείς.

Παράδειγμα: Να υπολογίσουμε **5 + (-5)**:

- 5: 00000101
- -5: 11111011

Άθροισμα:

markdown Αντιγραφή Επεξεργασία

```
00000101
+11111011
------------
100000000  (το 9ο bit το αγνοούμε)
```

Μένει 00000000 → δηλαδή 0.

7. Πώς βρίσκω τον αριθμό αν μου δώσουν το δυαδικό σε συμπλήρωμα προς 2

Αν το πρώτο bit είναι:

- **0** ➔ είναι θετικός ➔ διαβάζεις το δυαδικό κανονικά.
- **1** ➔ είναι αρνητικός ➔ κάνεις:
  1. Αντίστροφα τα bits
  2. Πρόσθεσε 1
  3. Βρες το δεκαδικό και βάζεις πρόσημο "-".

---

8. Παράδειγμα αναγνώρισης:

Δίνεται το δυαδικό **11111011** σε 8 bits.

1. Το πρώτο bit είναι 1 → αρνητικός αριθμός.
2. Αντίστροφα bits: 00000100
3. Πρόσθεση 1: 00000101 → 5
4. Άρα ο αριθμός είναι **-5**.

🎯 ΣΥΝΟΨΗ:

Ενέργεια	Περιγραφή
Για να βρεις αρνητικό	Αντιστροφή + 1
Πρόσημο από MSB	0 = θετικός, 1 = αρνητικός
Πώς προσθέτουμε	Κανονικά, αγνοώντας υπερχείλιση
Περιοχές τιμών σε N bits	$[-2^{n-1}, 2^{n-1}-1]$

📝 Ένα χρήσιμο ΤΙΠ:

Όταν βλέπεις συμπλήρωμα προς 2 και έχεις **8 bits**, να θυμάσαι ότι το -1 είναι:

Αντιγραφή Επεξεργασία

```
11111111₂
```

και το -128 είναι:

Αντιγραφή Επεξεργασία

```
10000000₂
```

ΕΛΕΝΗ

📝 Άσκηση:

Να βρεις το συμπλήρωμα ως προς 2 του αριθμού -45 σε 8 bits.

Λύση βήμα-βήμα:

Βήμα 1: Γράψε το +45 σε δυαδικό (8 bits)

- 45_{10} σε δυαδικό:
  ➔ 45 ÷ 2 → 22 υπόλ. 1
  ➔ 22 ÷ 2 → 11 υπόλ. 0
  ➔ 11 ÷ 2 → 5 υπόλ. 1
  ➔ 5 ÷ 2 → 2 υπόλ. 1
  ➔ 2 ÷ 2 → 1 υπόλ. 0
  ➔ 1 ÷ 2 → 0 υπόλ. 1

Αντιστρέφοντας τα υπόλοιπα:
➔ **101101_2**

Και σε 8 bits:
➔ **00101101_2**

Βήμα 2: Κάνε **αντιστροφή** των bits (0→1, 1→0)

- 00101101 → **11010010**

Βήμα 3: Πρόσθεσε 1

markdown Αντιγραφή Επεξεργασία

```
 11010010
+        1
---------
 11010011
```

✅ Άρα:

Το -45 σε 8 bits συμπλήρωμα ως προς 2 είναι:
→ 11010011_2

🎯 Τελικό αποτέλεσμα:

$$-45_{10} = 11010011_2 \quad (\text{σε 8 bits, συμπλήρωμα ως προς 2})$$

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

4 Ο ΕΠΕΞΕΡΓΑΣΤΗΣ

COMPUTER ARCHITECTURE 1

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Εισαγωγή στην Απόδοση της CPU

Η απόδοση της κεντρικής μονάδας επεξεργασίας (CPU) εξαρτάται από διάφορους παράγοντες, οι οποίοι σχετίζονται με την αρχιτεκτονική και την υλοποίηση του επεξεργαστή. Ακολουθούν οι κύριες έννοιες που αναφέρονται στην εισαγωγή του κεφαλαίου:

1. Παράγοντες απόδοσης της CPU

Οι παράγοντες που επηρεάζουν την απόδοση της CPU περιλαμβάνουν:

1. **Πλήθος εντολών**: Το πλήθος των εντολών που είναι διαθέσιμες για εκτέλεση. Η ποικιλία αυτή καθορίζεται από την αρχιτεκτονική συνόλου εντολών (ISA) και τον μεταγλωττιστή. Η ISA καθορίζει τις εντολές που υποστηρίζονται από τη CPU και τον τρόπο που μπορούμε να τις χρησιμοποιήσουμε.
2. **Κύκλοι ανά εντολή (CPI)**: Ο αριθμός των ρολογιών που απαιτούνται για την εκτέλεση μιας εντολής. Ο δείκτης αυτός, γνωστός ως CPI, είναι κρίσιμος για την εκτίμηση της απόδοσης, καθώς όσο χαμηλότερος είναι, τόσο καλύτερη απόδοση αναμένουμε.
3. **Χρόνος κύκλου ρολογιού**: Αυτός ο χρόνος καθορίζεται από το υλικό της CPU και περιγράφει τον χρόνο που απαιτείται για έναν πλήρη κύκλο του ρολογιού. Οι ταχύτεροι χρόνοι κύκλων ρολογιού συνήθως οδηγούν σε καλύτερη απόδοση.

2. Υλοποιήσεις RISC-V

Στο πλαίσιο του κεφαλαίου, θα εξεταστούν δύο υλοποιήσεις της αρχιτεκτονικής RISC-V:

- **Απλουστευμένη εκδοχή**: Μια πιο βασική εκδοχή που εστιάζει στις πιο συνηθισμένες και βασικές λειτουργίες χωρίς επιπλέον πολύπλοκες δυνατότητες.
- **Ρεαλιστική υλοποίηση με διοχέτευση**: Μια πιο σύνθετη έκδοση που περιλαμβάνει τη διαδικασία διοχέτευσης (pipelining), επιτρέποντας την εκτέλεση πολλών εντολών ταυτόχρονα σε διαφορετικά στάδια της διαδικασίας.

3. Τύποι Εντολών

Στην εξεταση των ενοτήτων, οι εντολές που θα αναλυθούν περιλαμβάνουν:

- **Εντολές αναφοράς μνήμης**: Όπως lw (load word) και sw (store word), που αφορούν τη φόρτωση και αποθήκευση δεδομένων από και προς τη μνήμη.
- **Αριθμητικές-λογικές εντολές**: Όπως οι add, sub, and, or, οι οποίες εκτελούν αριθμητικές ή λογικές επιχειρήσεις.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Εντολές μεταφοράς του ελέγχου**: Όπως η beq (branch if equal), που χρησιμοποιείται για τη ρύθμιση της ροής ελέγχου ανάλογα με τις συνθήκες.

Εκτέλεση Εντολών

Η εκτέλεση εντολών είναι μια κρίσιμη διαδικασία στη λειτουργία ενός επεξεργαστή και περιλαμβάνει διάφορα στάδια. Ακολουθούν οι βασικές πτυχές της, όπως περιγράφεται στο κείμενο:

1. **Μετρητής Προγράμματος (PC)**

   - **Ρόλος**: Ο Μετρητής Προγράμματος (Program Counter - PC) είναι υπεύθυνος για την διαχείριση της διεύθυνσης της τρέχουσας εντολής που εκτελείται.
   - **Λειτουργία**: Καταγράφει τη διεύθυνση της επόμενης εντολής που θα προσκομιστεί από τη μνήμη. Όταν εκτελείται μια εντολή, ο PC αυξάνεται κατά 4 (ή αλλιώς γεγονός ότι οι περισσότερες εντολές είναι 32-bit ή 4-byte), ώστε να εξετάσει την επόμενη.

2. **Ανάγνωση των Καταχωρητών**

   - **Καταχωρητές**: Οι αριθμοί καταχωρητών (register numbers) προέρχονται από ένα αρχείο καταχωρητών. Αυτοί οι καταχωρητές μπορεί να περιέχουν δεδομένα ή διευθύνσεις μνήμης.
   - **Διαδικασία**: Ο επεξεργαστής διαβάζει τους απαιτούμενους καταχωρητές, ανάλογα με την κατηγορία της εντολής που εκτελείται.

3. **Μονάδα Λογισμικού ALU**

   - **Λειτουργία**: Η Αριθμητική-Λογική Μονάδα (ALU) χρησιμοποιείται για να υπολογίσει:
     - **Αριθμητικά Αποτελέσματα**: Όπως το αποτέλεσμα προσθέσεων ή αφαιρέσεων.
     - **Διευθύνσεις Μνήμης**: Χρησιμοποιείται ώστε να υπολογιστούν οι διευθύνσεις μνήμης όταν πρόκειται για φόρτωση (load) ή αποθήκευση (store) δεδομένων.

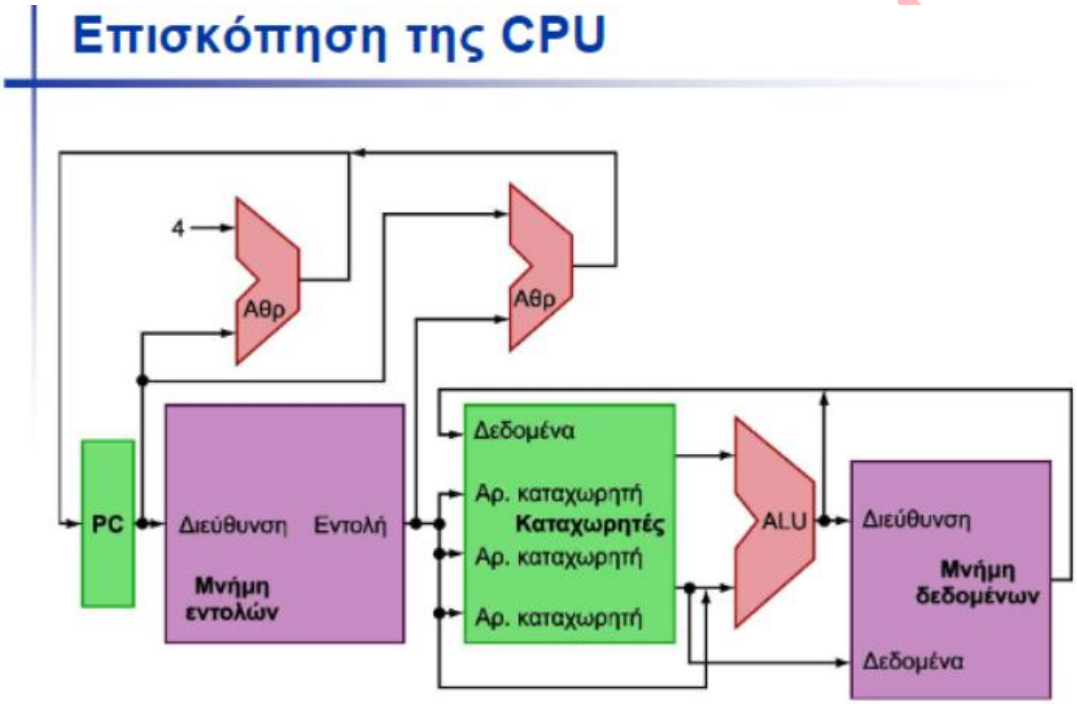
4. **Διακλάδωση (Branching)**

   - **Σύγκριση Διακλάδωσης**: Ο επεξεργαστής συγκρίνει τους καταχωρητές (ή τις τιμές τους) όταν έχει να κάνει με εντολές διακλάδωσης (π.χ. beq, which stands for branch if equal). Αυτή η διαδικασία χρειάζεται για να καθορίσει αν η ροή του προγράμματος θα συνεχιστεί κανονικά ή θα μεταβεί σε άλλη διεύθυνση.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

5. **Προσπέλαση Μνήμης Δεδομένων**

- **Λειτουργία**: Όταν πρόκειται για φόρτωση ή αποθήκευση δεδομένων, ο επεξεργαστής προσεγγίζει τη μνήμη για να πάρει ή να αποθηκεύσει τις αντίστοιχες τιμές.
- **Διεργασία**: Καθώς ολοκληρώνονται αυτοί οι υπολογισμοί, ο PC ενημερώνεται είτε με τη νέα διεύθυνση προορισμού είτε με το PC + 4.



Το διάγραμμα δείχνει τα βασικά μέρη μιας Κεντρικής Μονάδας Επεξεργασίας (CPU) και πώς αυτά συνδέονται μεταξύ τους για την εκτέλεση εντολών.

Συστατικά:

1. **PC (Καταχωρητής Εντολών / Program Counter)**
   Δείχνει τη διεύθυνση της επόμενης εντολής που θα εκτελεστεί.
2. **Μνήμη Εντολών (Instruction Memory)**
   - **Διεύθυνση**: Λαμβάνει τη διεύθυνση από το PC.
   - **Εντολή**: Η εντολή που ανακτάται από τη μνήμη.
3. **Αθρ (Αθροιστής)**
   - Ο πρώτος αθροιστής προσθέτει 4 στο PC για να δείξει στην επόμενη εντολή.

- Ο δεύτερος αθροιστής συμμετέχει σε υπολογισμούς διευθύνσεων για διακλαδώσεις.

4. **Καταχωρητές (Registers)**

Περιέχουν δεδομένα και διευθύνσεις που χρησιμοποιούνται για τους υπολογισμούς:

- **Αρ. καταχωρητή**: Οι αριθμοί των καταχωρητών που χρησιμοποιούνται.
- **Δεδομένα**: Οι τιμές που διαβάζονται ή γράφονται.

5. **ALU (Μονάδα Αριθμητικής και Λογικής)**

Εκτελεί αριθμητικές και λογικές πράξεις στα δεδομένα των καταχωρητών.

6. **Μνήμη Δεδομένων (Data Memory)**

- **Διεύθυνση**: Η θέση στη μνήμη όπου θα διαβαστούν ή θα γραφούν δεδομένα.
- **Δεδομένα**: Τα δεδομένα που διαβάζονται ή γράφονται στη μνήμη.

Ροή λειτουργίας:

Ο κύκλος ξεκινά από το **PC**, το οποίο δείχνει στη **Μνήμη Εντολών**. Η εντολή που ανακτάται καθορίζει ποιοι **Καταχωρητές** θα χρησιμοποιηθούν και τι θα κάνει η **ALU**. Εάν χρειάζεται, η **Μνήμη Δεδομένων** προσπελάζεται. Τέλος, το PC ενημερώνεται για την επόμενη εντολή.

Η ΕΝΤΟΛΗ ADD

🔧 **Παράδειγμα εντολής:**

add R1, R2, R3

Αυτό σημαίνει: **R1 = R2 + R3**

(Πρόσθεσε τα περιεχόμενα του καταχωρητή R2 και R3 και βάλε το αποτέλεσμα στον R1)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Εκτέλεση βήμα-βήμα (βάσει διαγράμματος):

1. Ανάκτηση εντολής (Instruction Fetch)

- Ο **PC (Program Counter)** δείχνει τη διεύθυνση της εντολής.
- Η **Μνήμη Εντολών** διαβάζει την εντολή add R1, R2, R3.

2. Ανάλυση εντολής (Instruction Decode)

- Από την εντολή εντοπίζονται:
  - Οι αριθμοί καταχωρητών πηγής: **R2 και R3**
  - Ο καταχωρητής προορισμού: **R1**

3. Ανάγνωση καταχωρητών (Register Read)

- Διαβάζονται τα περιεχόμενα των **R2 και R3** από το αρχείο καταχωρητών (πράσινο πλαίσιο).

4. Υπολογισμός (ALU Operation)

- Οι τιμές των R2 και R3 πάνε στην **ALU**.
- Η **ALU** εκτελεί την πράξη **πρόσθεσης**.
- Παράγει το αποτέλεσμα.

5. Εγγραφή αποτελέσματος (Write Back)

- Το αποτέλεσμα από την ALU **γράφεται πίσω στον καταχωρητή R1**.

Δεν εμπλέκεται:

- Η **Μνήμη Δεδομένων ΔΕΝ χρησιμοποιείται** εδώ γιατί δεν έχουμε πρόσβαση σε μνήμη – είναι καθαρά υπολογιστική εντολή.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Η ΕΝΤΟΛΗ LOAD

🔧 **Παράδειγμα εντολής:**

load R1, 0(R2)

Αυτό σημαίνει:

Φόρτωσε στη R1 τα δεδομένα από τη μνήμη στη διεύθυνση που δείχνει η R2 + 0

Εκτέλεση βήμα-βήμα (βάσει διαγράμματος):

1. Ανάκτηση εντολής (Instruction Fetch)

- Ο **PC** δείχνει στη **Μνήμη Εντολών**, η οποία ανακτά την εντολή load R1, 0(R2).

2. Ανάλυση εντολής (Instruction Decode)

- Εντοπίζονται:
  - Καταχωρητής **βάσης (R2)**
  - Καταχωρητής **προορισμού (R1)**
  - Άμεση τιμή offset (0)

3. Ανάγνωση καταχωρητή (Register Read)

- Διαβάζεται η τιμή του **R2** από το αρχείο καταχωρητών (πράσινο πλαίσιο).

4. Υπολογισμός διεύθυνσης (Address Calculation)

- Η **ALU** προσθέτει:
  - Την τιμή του R2 **+** το offset (0)
- Το αποτέλεσμα είναι η **διεύθυνση στη μνήμη δεδομένων**.

5. Ανάγνωση από Μνήμη (Memory Read)

- Η **Μνήμη Δεδομένων** (δεξιά μωβ πλαίσιο) διαβάζει τα δεδομένα από αυτή τη διεύθυνση.

6. Εγγραφή στο Καταχωρητή (Write Back)

- Τα δεδομένα που διαβάστηκαν γράφονται στον **καταχωρητή R1**.

Παρατήρηση:

Η load είναι εντολή **πρόσβασης στη μνήμη**, άρα **συνδυάζει** υπολογισμό (μέσω ALU) **και** ανάγνωση από **μνήμη δεδομένων**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Η ΕΝΤΟΛΗ STORE

Παράδειγμα εντολής:

store R1, 0(R2)

Αυτό σημαίνει:
Πάρε τα δεδομένα από τον καταχωρητή R1 και αποθήκευσέ τα στη μνήμη στη διεύθυνση R2 + 0

Εκτέλεση βήμα-βήμα:

1. Ανάκτηση εντολής (Instruction Fetch)

- Ο **PC** δείχνει σε μια διεύθυνση στη **Μνήμη Εντολών**, η οποία επιστρέφει την εντολή store R1, 0(R2).

2. Ανάλυση εντολής (Instruction Decode)

- Καταχωρητές:
  - **R2** = διεύθυνση βάσης (base address)
  - **R1** = δεδομένα προς αποθήκευση
  - **offset** = 0

3. Ανάγνωση καταχωρητών (Register Read)

- Διαβάζονται οι τιμές των **R1** (δεδομένα) και **R2** (διεύθυνση βάσης).

4. Υπολογισμός διεύθυνσης (Address Calculation)

- Η **ALU** προσθέτει: R2 + 0
- Υπολογίζεται η **τελική διεύθυνση μνήμης**.

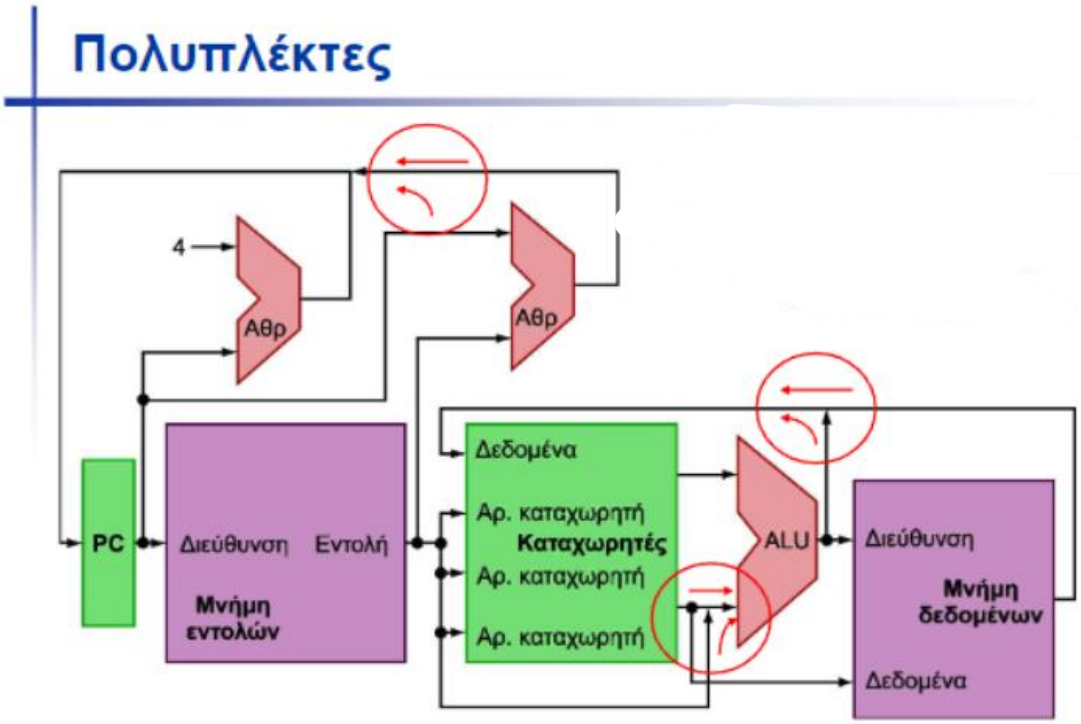
5. Εγγραφή στη Μνήμη (Memory Write)

- Η **τιμή του R1** στέλνεται ως **δεδομένα εισόδου** στη **Μνήμη Δεδομένων**.
- Η μνήμη γράφει τα δεδομένα στη διεύθυνση που υπολόγισε η ALU.

Παρατήρηση:

- **Η εντολή store γράφει στη μνήμη, δεν τροποποιεί καταχωρητές**.
- Δεν υπάρχει φάση "write back" στον καταχωρητή όπως με την add ή load.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



"Δεν γίνεται απλώς να ενωθούν σύρματα — Χρησιμοποιούνται πολυπλέκτες"

Τι είναι ο πολυπλέκτης (multiplexer ή MUX);

Ο **πολυπλέκτης** είναι ένα **ηλεκτρονικό κύκλωμα επιλογής**:

- Παίρνει **πολλαπλές εισόδους** αλλά **βγάζει μόνο μία έξοδο**, ανάλογα με ένα **σήμα επιλογής** (control signal).
- Δηλαδή επιλέγει **ποιο σήμα περνάει** παρακάτω, όταν υπάρχουν **εναλλακτικές διαδρομές**.

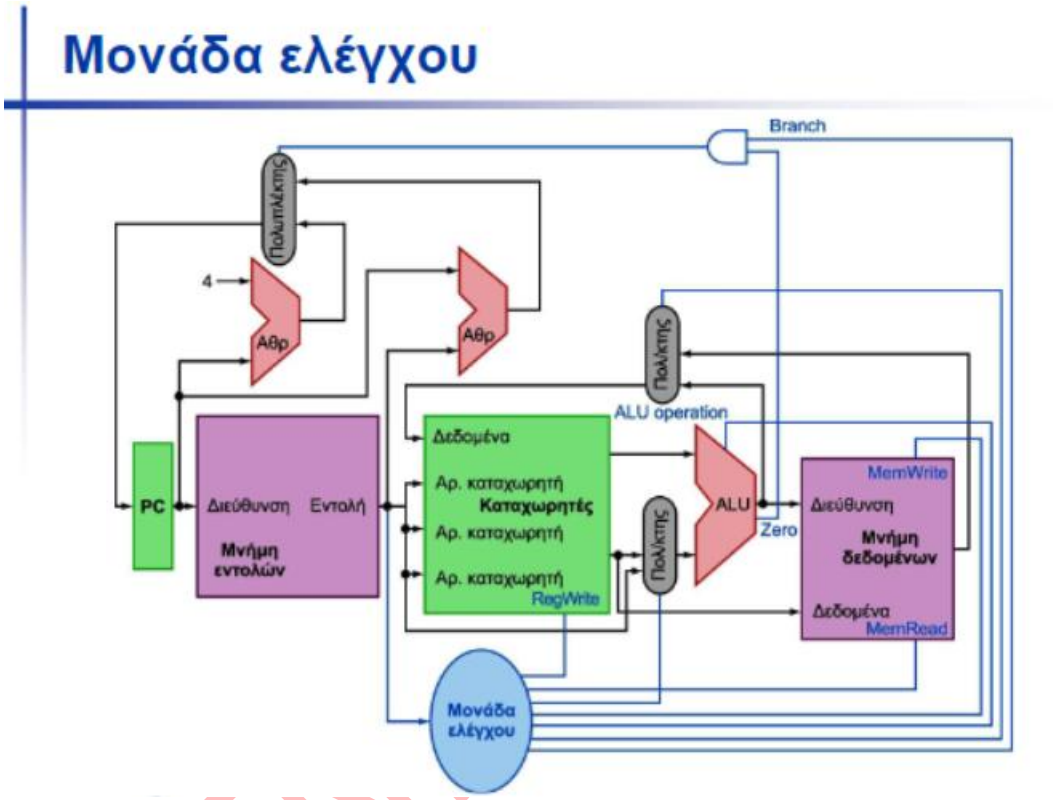
Τι δείχνουν οι κόκκινοι κύκλοι στην εικόνα:

1. **Πάνω αριστερά πολυπλέκτης (μετά τον PC):**
   - Επιλέγει αν το PC θα αυξηθεί κατά 4 (επόμενη εντολή) ή αν θα πάρει **άλλη διεύθυνση** (π.χ. από διακλάδωση).
2. **Δεξιά πολυπλέκτης πριν την ALU:**
   - Επιλέγει **ποιο δεδομένο θα πάει στην ALU**, από διαφορετική πηγή (καταχωρητής ή άμεση τιμή).
3. **Πολυπλέκτης μετά την ALU:**
   - Επιλέγει **πού θα γράψουμε το αποτέλεσμα**: σε καταχωρητή ή στη μνήμη.

ΕΛΕΝΗ ΖΑΦΕ

Γιατί είναι απαραίτητοι;

Χωρίς πολυπλέκτες, **δεν θα μπορούσαμε να ελέγξουμε ποια διαδρομή παίρνει το σήμα** – και τα κυκλώματα θα μπλέκονταν (π.χ. δύο σήματα σε ένα καλώδιο = σύγκρουση).



Αυτή η διαφάνεια δείχνει την **Μονάδα Ελέγχου (Control Unit)** της CPU και τον τρόπο με τον οποίο **καθοδηγεί τη ροή δεδομένων** με σήματα ελέγχου. Είναι η "νοημοσύνη" του επεξεργαστή που καθορίζει **τι να κάνει κάθε υποσύστημα**, ανάλογα με την εντολή που εκτελείται.

Κύρια στοιχεία που προστέθηκαν σε σχέση με τα προηγούμενα διαγράμματα:

Μονάδα Ελέγχου (Control Unit) (μπλε κύκλος κάτω)

- Αναλύει την **εντολή** που έρχεται από τη **Μνήμη Εντολών**.
- Στέλνει σήματα ελέγχου σε όλο το κύκλωμα:
    - RegWrite: αν θα γίνει εγγραφή σε καταχωρητή
    - ALU operation: ποια πράξη να κάνει η ALU

- MemRead / MemWrite: αν θα διαβάσουμε/γράψουμε μνήμη
- Branch: αν πρόκειται για εντολή διακλάδωσης

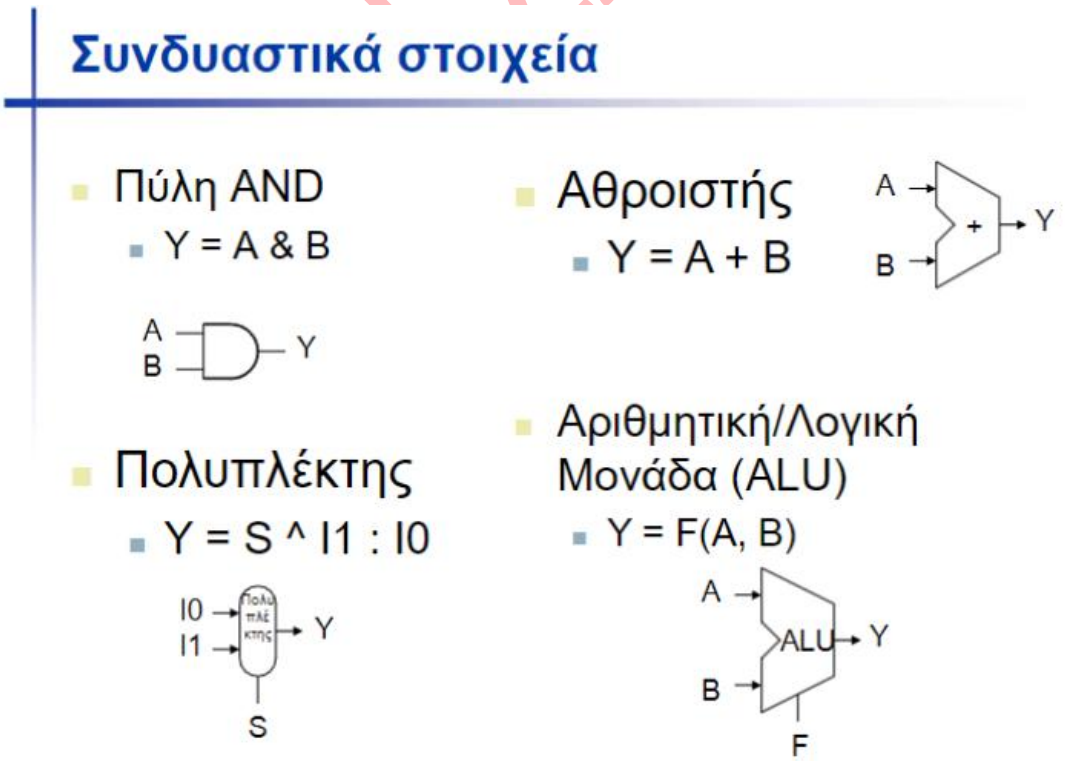
Πώς δουλεύει σε μια εντολή:

Ας δούμε με παράδειγμα **εντολής διακλάδωσης** (beq R1, R2, label):

1. **Η μονάδα ελέγχου** αναγνωρίζει ότι είναι beq → στέλνει Branch = 1.
2. Η **ALU συγκρίνει** R1 και R2 → παράγει σήμα Zero (αν είναι ίσα).
3. Το σήμα Zero ΚΑΙ το Branch περνούν από **AND πύλη** (πάνω δεξιά).
   - Αν είναι και τα δύο 1 → γίνεται διακλάδωση (αλλάζει το PC).
4. Οι **πολυπλέκτες** καθοδηγούν το PC είτε να πάει στην επόμενη εντολή είτε να πηδήξει στη νέα διεύθυνση (label).

Πλεονέκτημα:

Η μονάδα ελέγχου **επιτρέπει στον επεξεργαστή να εκτελεί πολλές διαφορετικές εντολές**, επειδή μπορεί δυναμικά να ρυθμίζει **πώς λειτουργούν τα υποσυστήματα**.



Στη λογική σχεδίαση, τα **συνδυαστικά στοιχεία** είναι δομικά στοιχεία που επεξεργάζονται δεδομένα χωρίς να διατηρούν κατάσταση. Αυτό σημαίνει ότι η

έξοδος τους εξαρτάται αποκλειστικά από τις τρέχουσες εισόδους τους. Ακολουθούν παραδείγματα από βασικά συνδυαστικά στοιχεία:

1**. Πύλη AND**

Η πύλη AND είναι ένα απλό λογικό στοιχείο που έχει δύο εισόδους (A και B) και παράγει μία έξοδο (Y). Η έξοδος είναι 1 μόνο όταν και οι δύο είσοδοι είναι 1.

- **Λειτουργία**: Y=A&B*Y=A&B*

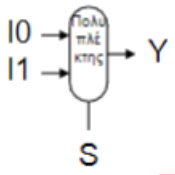
Διάγραμμα:

2. Πολυπλέκτης

Ο πολυπλέκτης είναι ένα στοιχείο που συνδυάζει πολλές εισόδους σε μία έξοδο με βάση μια επιλογή (S). Επιτρέπει την επιλογή μιας από τις πολλές εισόδους και την εκχώρηση της στην έξοδο.

- **Λειτουργία**: Y=S ? I1:I0*Y=S?I1:I0*

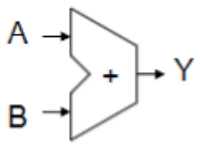
Διάγραμμα:



3. Αθροιστής

Ο αθροιστής είναι ένα στοιχείο που προσθέτει δύο αριθμούς A και B για να παράξει ένα αποτέλεσμα Y. Ή αλλιώς, είναι η βασική μονάδα για την εκτέλεση αριθμητικών πράξεων.

- **Λειτουργία**: Y=A+B*Y=A+B*

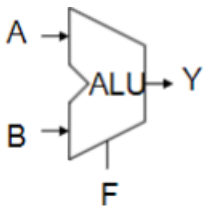


ΕΛΕΝΗ ΖΑΦΕΙΡΗ

4. Αριθμητική/Λογική Μονάδα (ALU)

Η ALU είναι ένα κεντρικό στοιχείο της CPU που εκτελεί όλες τις αριθμητικές και λογικές λειτουργίες. Η επιλογή της λειτουργίας καθορίζεται από τη γραμμή ελέγχου F.

- **Λειτουργία**: Y=F(A,B)$Y=F(A,B)$



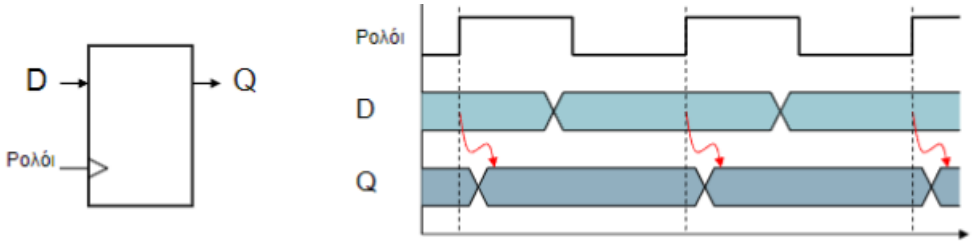
Ακολουθιακά Στοιχεία

Τα ακολουθιακά στοιχεία αποτελούν βασικά δομικά μέρη στα ψηφιακά κυκλώματα και είναι υπεύθυνα για την αποθήκευση και την επεξεργασία πληροφοριών με τον χρόνο. Ακολουθούν οι κύριες έννοιες και λειτουργίες του καταχωρητή, ενός από τα πιο κοινά ακολουθιακά στοιχεία.

Καταχωρητής

- **Ορισμός**:
  - Ο καταχωρητής είναι ένα ακολουθιακό στοιχείο που αποθηκεύει δεδομένα στην ψηφιακή λογική. Χρησιμοποιείται για να κρατά μια λογική τιμή ή ένα διπλό αριθμό για μελλοντική χρήση στην επεξεργασία.
- **Χρήση σήματος ρολογιού**:
  - Ο καταχωρητής χρησιμοποιεί το σήμα ρολογιού (clock signal) για να καθορίσει πότε θα ενημερωθεί η αποθηκευμένη τιμή. Το ρολόι λειτουργεί ως ρυθμιστής του χρόνου για την εκτέλεση των λογικών και ακολουθιακών διαδικασιών.
- **Ενημέρωση τιμής**:
  - Η ενημέρωση της αποθηκευμένης τιμής συμβαίνει όταν το σήμα του ρολογιού μεταβάλλεται από 0 σε 1. Αυτή είναι η αρχή της αυτοπυροδότησης (self-triggering), που σημαίνει ότι η ενημέρωση γίνεται αυτομάτως με βάση την αλλαγή του ρολογιού.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Στο παραπάνω διάγραμμα:

- **D** είναι η είσοδος του καταχωρητή (data input).
- **Ρολόι** είναι το σήμα ρολογιού που ελέγχει την ενημέρωση.
- **Q** είναι η έξοδος του καταχωρητή (data output) που αποθηκεύει την τελευταία αποθηκευμένη τιμή.

Σημασία στην αρχιτεκτονική υπολογιστών

- Τα ακολουθιακά στοιχεία, όπως οι καταχωρητές, είναι κρίσιμα για την υλοποίηση της μνήμης και της λογικής λειτουργίας σε επεξεργαστές και άλλα ψηφιακά κυκλώματα. Παρέχουν την ικανότητα αποθήκευσης πληροφοριών και χρόνος για την επεξεργασία αυτών των πληροφοριών, που είναι απαραίτητος για την εκτέλεση προγραμμάτων και τη διαχείριση δεδομένων.

Καταχωρητής με Έλεγχο Εγγραφής

Ένας **καταχωρητής με έλεγχο εγγραφής** (ή **write-back register** στα αγγλικά) είναι ένας τύπος καταχωρητή στη **λογική μονάδα επεξεργασίας (CPU)** ενός υπολογιστή που χρησιμοποιείται για την αποθήκευση δεδομένων που προορίζονται να γραφούν σε μνήμη. Ο όρος "με έλεγχο εγγραφής" υποδηλώνει ότι η εγγραφή στα δεδομένα που είναι αποθηκευμένα στον καταχωρητή μπορεί να ελεγχθεί ή να αναβληθεί, με σκοπό να διασφαλιστεί ότι τα δεδομένα θα εγγραφούν στη μνήμη σε μια καθορισμένη στιγμή ή υπό ορισμένες συνθήκες.

<u>Λειτουργία</u>

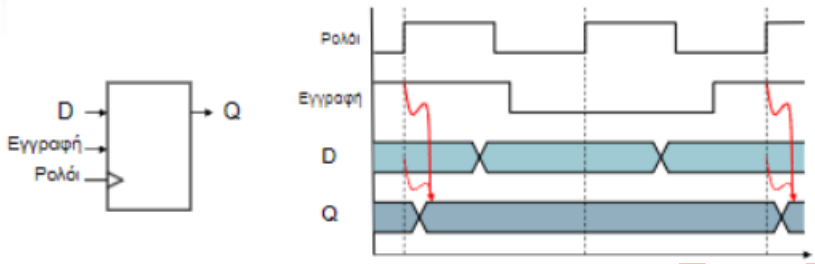
- **Ενημέρωση σε Ακμή Ρολογιού**: Ο καταχωρητής ενημερώνει την αποθηκευμένη τιμή του μόνο κατά την ακμή (rising edge) του σήματος ρολογιού. Αυτό σημαίνει ότι η αλλαγή της αποθηκευμένης τιμής συμβαίνει σε συγκεκριμένες χρονικές στιγμές, βελτιώνοντας τη συγχρονικότητα και την ακρίβεια των λειτουργιών του.
- **Σημασία Σηματοδότη Εγγραφής**: Το σήμα ελέγχου εγγραφής, το οποίο συνήθως έχει τιμή 1 όταν είναι ενεργοποιημένο, καθορίζει πότε γίνεται η ενημέρωση της αποθηκευμένης τιμής. Όταν το σήμα είναι μη ενεργό (τιμή 0), ο καταχωρητής διατηρεί την τρέχουσα τιμή του.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Χρήση

Η χρήση ενός καταχωρητή με έλεγχο εγγραφής είναι συχνή σε περιπτώσεις όπου οι αποθηκευμένες τιμές χρειάζονται για μελλοντικές εργασίες εντός του κυκλώματος. Για παράδειγμα, σε μία διαδικασία, μπορεί να απαιτείται η αποθηκευμένη τιμή να χρησιμοποιηθεί αργότερα για υπολογισμούς ή για σύγκριση με άλλες τιμές.

Σχήμα



Μεθοδολογία Χρονισμού

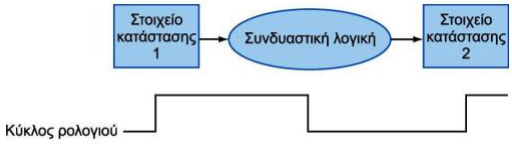
Εισαγωγή

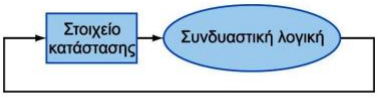
Η **μεθοδολογία χρονισμού** αναφέρεται στον τρόπο που οργανώνονται και συγχρονίζονται οι διαδικασίες εντός μιας ψηφιακής αρχιτεκτονικής κατά τη διάρκεια του χρόνου. Ειδικότερα, αφορά τη διαδικασία μετασχηματισμού δεδομένων μέσω των διακυμάνσεων που παρέχονται από το ρολόι.

Λειτουργία

- **Δεδομένα και Συνδυαστική Λογική**: Τα δεδομένα μετασχηματίζονται μέσω συνδυαστικής λογικής, που αναφέρεται σε λογικά κυκλώματα, τα αποτελέσματα των οποίων εξαρτώνται μόνο από τις τρέχουσες εισόδους τους. Αυτό συμβαίνει **κατά τη διάρκεια των κύκλων του ρολογιού**, όπου συλλέγονται και επεξεργάζονται οι τιμές.
- **Είσοδος και Έξοδος**: Υπάρχει είσοδος από στοιχεία κατάστασης (registers) και έξοδος που οδηγείται σε άλλα στοιχεία κατάστασης. Αυτό επιτρέπει την αλληλεπίδραση μεταξύ διαφορετικών μερών της CPU κατά τη διάρκεια της λειτουργίας της.
- **Περίοδος του Ρολογιού**: Η **μεγαλύτερη καθυστέρηση** στην όλη διαδρομή της επεξεργασίας δεδομένων καθορίζει την περίοδο του ρολογιού, δηλαδή το χρονικό διάστημα μεταξύ των κύκλων του ρολογιού. Αυτό σημαίνει ότι η λειτουργία της CPU δεν μπορεί να είναι γρηγορότερη από το απαιτούμενο μέγιστο διάστημα για την ολοκλήρωση της πιο χρονοβόρας διαδικασίας.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ





Κατασκευή μιας Διαδρομής Δεδομένων

Η "διαδρομή δεδομένων" αναφέρεται στο σύνολο των στοιχείων που είναι υπεύθυνα για την επεξεργασία των δεδομένων και των διευθύνσεων στο εσωτερικό της Κεντρικής Μονάδας Επεξεργασίας (CPU). Αυτή η έννοια παίζει κρίσιμο ρόλο στη σχεδίαση και την απόδοση ενός υπολογιστικού συστήματος.

Κύρια Στοιχεία της Διαδρομής Δεδομένων

- **Καταχωρητές**: Αυτά τα στοιχεία αποθηκεύουν προσωρινά τα δεδομένα που χρειάζονται κατά τη διάρκεια της επεξεργασίας. Ουσιαστικά, οι καταχωρητές είναι μικροί και γρήγοροι χώροι αποθήκευσης που χρησιμοποιούνται για τη διαχείριση των ενδιάμεσων αποτελεσμάτων.
- **Μονάδες ALU (Αριθμητική και Λογική Μονάδα)**: Αυτές οι μονάδες εκτελούν αριθμητικές και λογικές πράξεις, όπως πρόσθεση, αφαίρεση, AND και OR. Η ALU είναι σημαντική για την εκτέλεση κάθε εντολής που απαιτεί υπολογισμό.
- **Πολυπλέκτες**: Αυτά τα στοιχεία επιλέγουν μία από πολλές εισόδους και τις προωθούν στην έξοδο, ανάλογα με το απαιτούμενο σήμα ελέγχου. Είναι κρίσιμα στην αποτελεσματική διαχείριση των δεδομένων που ρέουν μέσα και έξω από τη CPU.
- **Μνήμες**: Οι μνήμες χρησιμοποιούνται για την αποθήκευση δεδομένων και κώδικα. Οι διάφοροι τύποι μνημών (π.χ. RAM, ROM) εξυπηρετούν διαφορετικές ανάγκες σε σχέση με τα δεδομένα που χρησιμοποιούνται και τη διάρκεια αποθήκευσής τους.

<u>Διαδικασία Κατασκευής της Διαδρομής Δεδομένων RISC-V</u>

Η διαδικασία περιλαμβάνει τη βήμα-βήμα ανάπτυξη μονάδων για την υλοποίηση της CPU σύμφωνα με την αρχιτεκτονική RISC-V. Αυτό σημαίνει τη βελτιστοποίηση της διαδρομής δεδομένων για καλύτερη απόδοση, με στόχο τη μείωση της καθυστέρησης και τη μεγιστοποίηση της ταχύτητας εκτέλεσης των εντολών.

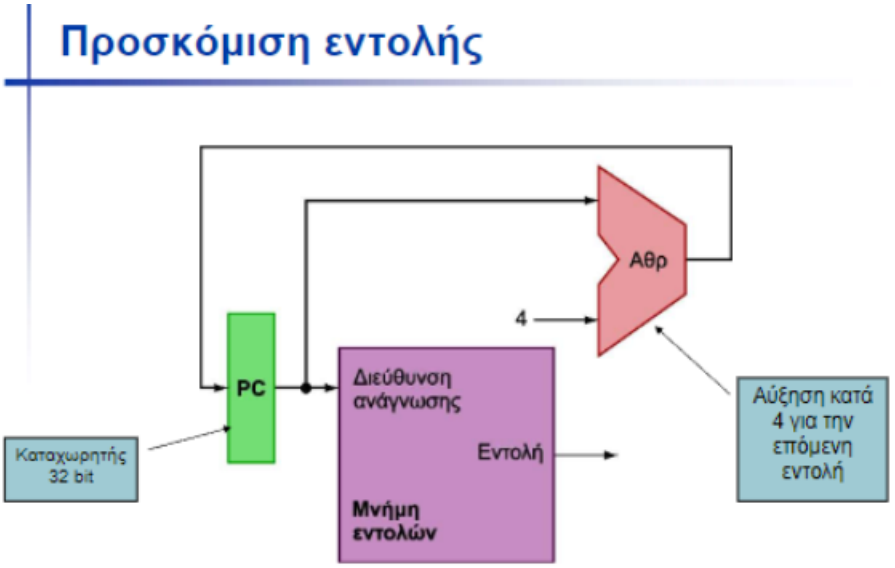
Βελτίωση της Σχεδίασης

Η βελτίωση της γενικότερης σχεδίασης της διαδρομής δεδομένων περιλαμβάνει:

- Ανάπτυξη πιο αποδοτικών μηχανισμών διαχείρισης των δεδομένων.
- Εξαφάνιση ή μείωση των λανθάνοντων χρόνων κατά την εκτέλεση των εντολών.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Δημιουργία μιας σταθερής, συνεκτικής και αποδοτικής ροής εργασίας μέσα στην CPU, που να επιτρέπει γρηγορότερες και πιο αξιόπιστες εκτελέσεις των προγραμμάτων.



Ανάλυση διαγράμματος:

- **PC (Program Counter)**: Καταχωρητής 32-bit που περιέχει τη διεύθυνση της επόμενης εντολής προς εκτέλεση.
- **Μνήμη εντολών (Instruction Memory)**:
  - Δέχεται τη διεύθυνση από τον PC.
  - Παρέχει την αντίστοιχη εντολή από αυτή τη διεύθυνση.
- **Αθροιστής (Adder)**:
  - Προσθέτει την τιμή 4 στη διεύθυνση του PC (PC + 4), ώστε να μεταβεί στην επόμενη εντολή (υποθέτοντας ότι κάθε εντολή έχει μήκος 4 bytes).
- **Ροή δεδομένων**:

1. Ο PC δίνει τη διεύθυνση στην μνήμη εντολών.
2. Η εντολή διαβάζεται.
3. Ο PC αυξάνεται κατά 4 μέσω του αθροιστή.
4. Η νέα τιμή του PC επανατροφοδοτείται για την επόμενη κύκλο.

Εντολές Μορφής R

Οι εντολές μορφής R (Register format instructions) είναι μία κατηγορία εντολών στην αρχιτεκτονική RISC-V που σχετίζονται με την εκτέλεση αριθμητικών και λογικών πράξεων. Ακολουθούν συγκεκριμένα βήματα κατά την εκτέλεσή τους.

Βασικά Χαρακτηριστικά

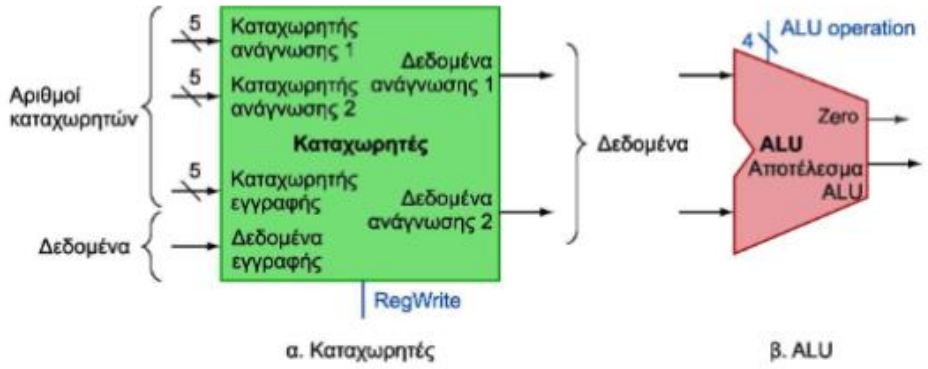
- **Ανάγνωση δύο τελεστέων καταχωρητών:**
  - Κατά την εκτέλεση μιας εντολής μορφής R, η CPU διαβάζει δεδομένα από δύο καταχωρητές. Αυτοί οι καταχωρητές περιέχουν τους τελεστές (operand) που θα χρησιμοποιηθούν στη σχετική πράξη.
  - Αυτή η διαδικασία είναι κρίσιμη καθώς οι καταχωρητές παρέχουν την απαραίτητη πληροφορία για την εκτρεπή πράξη.
- **Εκτέλεση αριθμητικής/λογικής πράξης:**
  - Μετά την ανάγνωση των δεδομένων, η Αριθμητική/Λογική Μονάδα (ALU) αναλαμβάνει την εκτέλεση της συγκεκριμένης αριθμητικής ή λογικής πράξης, όπως προσθήκη, αφαίρεση, AND, OR κ.λπ.
  - Η επιλογή της πράξης βασίζεται στον κωδικό λειτουργίας (opcode) της εντολής, ο οποίος καθορίζει ποια λειτουργία θα εκτελεστεί.
- **Εγγραφή του αποτελέσματος στον καταχωρητή:**
  - Αφού ολοκληρωθεί η πράξη, το αποτέλεσμα αποθηκεύεται σε έναν καταχωρητή. Αυτός ο καταχωρητής αναφέρεται ως "destination register".
  - Έτσι, το αποτέλεσμα είναι διαθέσιμο για περαιτέρω επεξεργασία ή χρησιμοποιείται σε άλλες εντολές που έπονται.

<u>Σημαντικότητα των Εντολών Μορφής R</u>

- **Αποτελεσματικότητα:** Οι εντολές μορφής R επιτρέπουν την εκτέλεση πολύπλοκων υπολογισμών με ελάχιστο εύρος μνήμης, καθώς οι καταχωρητές είναι ταχύτεροι από τη μνήμη.
- **Ευελιξία:** Συνδυάζοντας διαφορετικές λογικές και αριθμητικές πράξεις, η CPU μπορεί να εκτελεί διάφορους αλγορίθμους γρηγορότερα και πιο αποδοτικά.
- **Βασικές δομές:** Αυτές οι εντολές είναι το θεμέλιο της εκτέλεσης υπολογιστών στις αρχιτεκτονικές που βασίζονται σε RISC, όπως το RISC-V και δημιουργούν τις βάσεις για την κατανόηση πιο σύνθετων εντολών ή δομών.

Αυτή η οπτική γωνία στον τρόπο που η CPU διαχειρίζεται και εκτελεί εντολές μορφής R είναι κρίσιμη για τον σχεδιασμό και την κατανόηση της αρχιτεκτονικής υπολογιστών.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



- **Επεξήγηση του διαγράμματος:**

(α) Καταχωρητής:

- Εισόδους:
  - **rs, rt, rd**: διευθύνσεις των καταχωρητών.
  - **RegWrite**: σήμα ελέγχου για εγγραφή.
- Εξόδους:
  - **Δεδομένα ανάγνωσης 1 και 2**: τιμές από rs και rt.
  - **Δεδομένα εγγραφής**: προορίζονται για τον rd.

(β) ALU:

- Δέχεται τα δύο δεδομένα από τους καταχωρητές.
- Εκτελεί την πράξη που υποδεικνύεται από το σήμα **ALU operation**.
- Επιστρέφει το αποτέλεσμα και το σήμα **Zero** (αν το αποτέλεσμα είναι 0).

Εντολή Φόρτωσης/Αποθήκευσης

Η εντολή φόρτωσης/αποθήκευσης (Load/Store Instruction) είναι καθοριστική για τη διαχείριση των δεδομένων στη μνήμη σε μια αρχιτεκτονική υπολογιστών όπως το RISC-V. Ακολουθεί μια αναλυτική εξήγηση των βασικών στοιχείων αυτού του τύπου εντολής:

1. Ανάγνωση Τελεστέων Καταχωρητών

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Αυτή η διαδικασία περιλαμβάνει την ανάγνωση δεδομένων από τους καταχωρητές που συμμετέχουν στη διαδικασία φόρτωσης ή αποθήκευσης. Οι τελεστέοι είναι οι καταχωρητές που κρατούν τις τιμές που συμμετέχουν στις OPERATIONS (λογιστικές π.χ. πρόσθεση, αφαίρεση) ή διευθύνσεις μνήμης.

2. Υπολογισμός της Διεύθυνσης με Σχετική Απόσταση 12 bit

- Σε αυτό το βήμα, υπολογίζεται η διεύθυνση μνήμης από την οποία θα γίνει η φόρτωση ή στην οποία θα γίνει η αποθήκευση. Η "σχετική απόσταση 12 bit" αναφέρεται στο ότι μπορεί να γίνει αναφορά σε διευθύνσεις που είναι σχετικά κοντά στη θέση του τρέχοντος προγράμματος. Αυτό επιτρέπει ταχύτερες διευθύνσεις μνήμης και βελτιστοποιημένη απόδοση.

3. Χρήση της ALU Δεδομένων με Επέκταση Προσήμου

- Η ALU (Αριθμητική/Λογική Μονάδα) χρησιμοποιείται για να υπολογίσει τη σωστή διεύθυνση μνήμης. Η "εξάπλωση προσήμου (σχετική απόσταση)" σημαίνει ότι τα δεδομένα της διεύθυνσης επεκτείνονται για να διατηρηθούν σωστά οι θετικές και αρνητικές τιμές, καθώς οι μνήμες μπορεί να είναι και σε αρνητικές διευθύνσεις.

4. Φόρτωση: Ανάγνωση από Μνήμη και Ενημέρωση Καταχωρητή

- Όταν μια εντολή φόρτωσης εκτελείται, τα δεδομένα διαβάζονται από τη μνήμη και αποθηκεύονται σε έναν καταχωρητή. Αυτή η διαδικασία επιτρέπει την πρόσβαση σε δεδομένα που βρίσκονται στη μνήμη από την CPU.

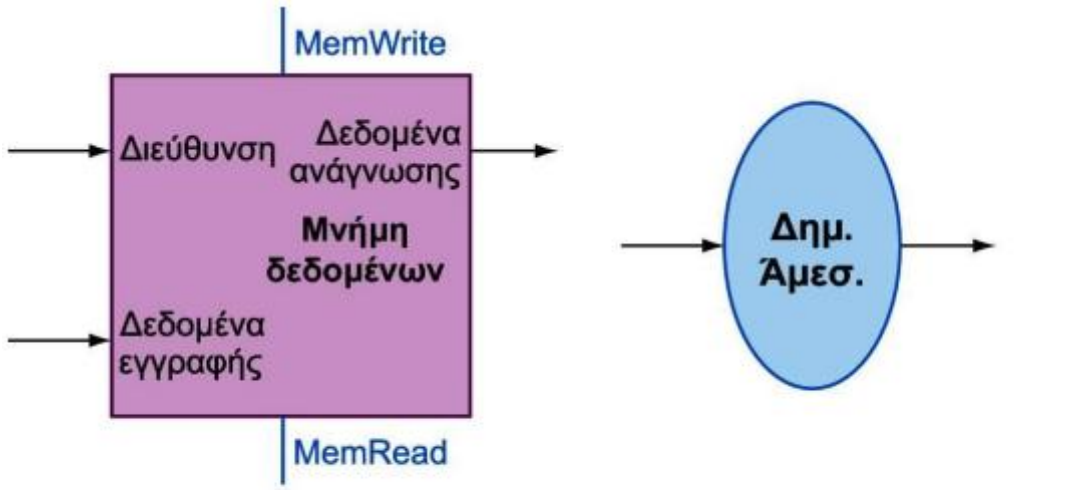
5. Αποθήκευση: Εγγραφή Τιμής Καταχωρητή στη Μνήμη

- Αντίθετα με την εντολή φόρτωσης, η εντολή αποθήκευσης επιτρέπει την εγγραφή τιμών από έναν καταχωρητή πίσω στη μνήμη. Έτσι, τα δεδομένα που έχουν υπολογιστεί ή που έχουν προκύψει από προηγούμενες διεργασίες μπορούν να αποθηκευτούν για μελλοντική χρήση.

<u>Σημασία:</u>

Αυτές οι διαδικασίες είναι κρίσιμες για τη διαχείριση και την προσβασιμότητα των δεδομένων στη CPU, και για την αποτελεσματική και γρήγορη λειτουργία ενός υπολογιστή. Η σωστή κατανόηση των λειτουργιών τους είναι απαραίτητη για την αποδοτική σχεδίαση και υλοποίηση συστημάτων υπολογιστών.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



α. Μονάδα μνήμης δεδομένων β. Μονάδα δημιουργίας άμεσου

Η εικόνα παρουσιάζει δύο επιμέρους μονάδες που χρησιμοποιούνται κατά την εκτέλεση εντολών μορφής **I** (Immediate) ή φορτώματος/αποθήκευσης σε μια αρχιτεκτονική όπως η **MIPS**:

- **(α) Μονάδα μνήμης δεδομένων (Data Memory Unit)**

Αυτή είναι η μονάδα που επιτρέπει την **ανάγνωση ή εγγραφή δεδομένων** από/προς τη **μνήμη δεδομένων** (RAM).

Περιλαμβάνει:

- **Διεύθυνση**: Πού θέλουμε να γράψουμε ή να διαβάσουμε.
- **Δεδομένα εγγραφής**: Τι θέλουμε να αποθηκεύσουμε (μόνο αν είναι ενεργό το MemWrite).
- **Δεδομένα ανάγνωσης**: Τι διαβάζουμε από τη διεύθυνση (αν είναι ενεργό το MemRead).
- **Σήματα ελέγχου**:
    - MemRead: Αν είναι ενεργό, τότε διαβάζει δεδομένα από μνήμη.
    - MemWrite: Αν είναι ενεργό, τότε γράφει δεδομένα στη μνήμη.

Αυτή η μονάδα χρησιμοποιείται σε εντολές όπως:

- lw (load word)
- sw (store word)

- **(β) Μονάδα δημιουργίας άμεσου (Immediate Generator)**

ΕΛΕΝΗ ΖΑΦΕΙΡ

Η Μονάδα Δημιουργίας Άμεσου (Immediate Generator ή ImmGen) **επεξεργάζεται το "immediate" πεδίο** της εντολής (δηλαδή μια σταθερά που περιέχεται μέσα στην εντολή), και το μετατρέπει σε **κατάλληλης μορφής και μήκους αριθμό** για χρήση από την ALU ή από τη μνήμη.

Π.χ. στην εντολή:

assembly

addi $t1, $t2, 10

ο αριθμός 10 περνά από τη μονάδα δημιουργίας άμεσου και δίνεται στην ALU.

- **Πότε χρησιμοποιούνται μαζί:**

Συνδυασμένα, αυτές οι δύο μονάδες χρησιμοποιούνται όταν:

- Η ALU υπολογίζει μια διεύθυνση χρησιμοποιώντας έναν **καταχωρητή και έναν άμεσο αριθμό**.
- Το αποτέλεσμα αυτό δίνεται στη **μονάδα μνήμης δεδομένων** για ανάγνωση ή εγγραφή.

Παρατήρηση

Η **μνήμη** (RAM) **δεν είναι ενσωματωμένη** μέσα στον **επεξεργαστή** στις περισσότερες κλασικές αρχιτεκτονικές. Αντίθετα, είναι **ξεχωριστό υποσύστημα** με το οποίο ο επεξεργαστής επικοινωνεί μέσω διαύλων (buses).

Ωστόσο, υπάρχουν **κάποιες εξαιρέσεις** και διαφορές ανάλογα με τον τύπο της μνήμης:

- **Είδη μνήμης σε σχέση με τον επεξεργαστή:**

1. **Κύρια μνήμη (Main memory / RAM)**:
   - Είναι **εξωτερική**.
   - Ο επεξεργαστής στέλνει αιτήματα ανάγνωσης/εγγραφής μέσω του **memory bus**.
   - Περιλαμβάνει τον χώρο των εντολών και των δεδομένων.
2. **Cache μνήμη** (L1, L2, L3):
   - Είναι **ενσωματωμένη** ή πολύ κοντά στον επεξεργαστή.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Σκοπός: πιο γρήγορη πρόσβαση σε δεδομένα/εντολές που χρησιμοποιούνται συχνά.
- Η L1 συνήθως βρίσκεται **μέσα στον πυρήνα του επεξεργαστή**.

3. **Καταχωρητές (Registers)**:
   - Είναι **μέσα στον επεξεργαστή**.
   - Πολύ μικρός αριθμός, αλλά **ταχύτερη μνήμη** απ' όλες.

Ο **διάυλος** (ή **bus** στα αγγλικά) είναι ένα **σύνολο από καλώδια ή γραμμές** που επιτρέπουν τη **μεταφορά δεδομένων, διευθύνσεων και σημάτων ελέγχου** ανάμεσα στα διάφορα μέρη ενός υπολογιστή, όπως ο επεξεργαστής, η μνήμη και οι περιφερειακές συσκευές.

- **Είδη διαύλων:**

1. **Δίαυλος δεδομένων (Data Bus)**
   - Μεταφέρει τα **δεδομένα** μεταξύ CPU, RAM, και άλλων συσκευών.
   - Π.χ. όταν ο επεξεργαστής ζητάει έναν αριθμό από τη μνήμη, τα δεδομένα περνούν από εδώ.
2. **Δίαυλος διευθύνσεων (Address Bus)**
   - Μεταφέρει τις **διευθύνσεις μνήμης** που χρησιμοποιούνται για να εντοπιστούν τα δεδομένα.
   - Π.χ. η CPU λέει στη RAM: "δώσε μου ό,τι έχεις στη διεύθυνση 0x00400004".
3. **Δίαυλος ελέγχου (Control Bus)**
   - Μεταφέρει **σήματα ελέγχου** (π.χ. "διάβασε", "γράψε", "επιλογή μνήμης").
   - Ορίζει **ποια λειτουργία** θα εκτελεστεί.

- **Παράδειγμα λειτουργίας:**

Όταν η CPU θέλει να διαβάσει μια εντολή από τη μνήμη:

1. Στέλνει τη **διεύθυνση** μέσω του **address bus**.
2. Στέλνει σήμα **"read"** μέσω του **control bus**.
3. Η **μνήμη** απαντά στέλνοντας τα **δεδομένα** μέσω του **data bus**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Στοιχεία Εκτέλεσης Μιας Εντολής Διακλάδωσης

1. Ανάγνωση Τελεστέων Καταχωρητών

Η εντολή διακλάδωσης ξεκινά με την ανάγνωση των περιεχομένων δύο καταχωρητών. Οι τιμές αυτές είναι οι τελεστέοι που θα συγκριθούν.

2. Σύγκριση Τελεστέων

Η σύγκριση γίνεται από την **Αριθμητική Λογική Μονάδα (ALU)**. Αυτή υπολογίζει τη διαφορά των δύο τιμών (π.χ. A - B) και ελέγχει αν το αποτέλεσμα είναι **μηδέν**. Αν είναι, αυτό σημαίνει ότι οι δύο τελεστέοι είναι ίσοι.

3. Χρήση της ALU

Η ALU εκτελεί την αφαίρεση των δύο τελεστέων και αξιολογεί το **Zero flag**, για να διαπιστωθεί αν πρέπει να γίνει διακλάδωση ή όχι.

4. Υπολογισμός Διεύθυνσης Προορισμού

Αν η συνθήκη της διακλάδωσης ισχύει, πρέπει να υπολογιστεί η νέα διεύθυνση στην οποία θα μεταβεί η εκτέλεση. Η νέα διεύθυνση προκύπτει προσθέτοντας μια σχετική μετατόπιση (offset) στην τρέχουσα τιμή του Μετρητή Προγράμματος (PC).

5. Επέκταση Προσήμου

Η μετατόπιση είναι συνήθως κωδικοποιημένη ως 12-bit αριθμός. Επειδή μπορεί να είναι θετική ή αρνητική, απαιτείται **επέκταση προσήμου** (sign extension), ώστε να διατηρηθεί η σωστή αριθμητική σημασία κατά τον υπολογισμό της νέας διεύθυνσης.

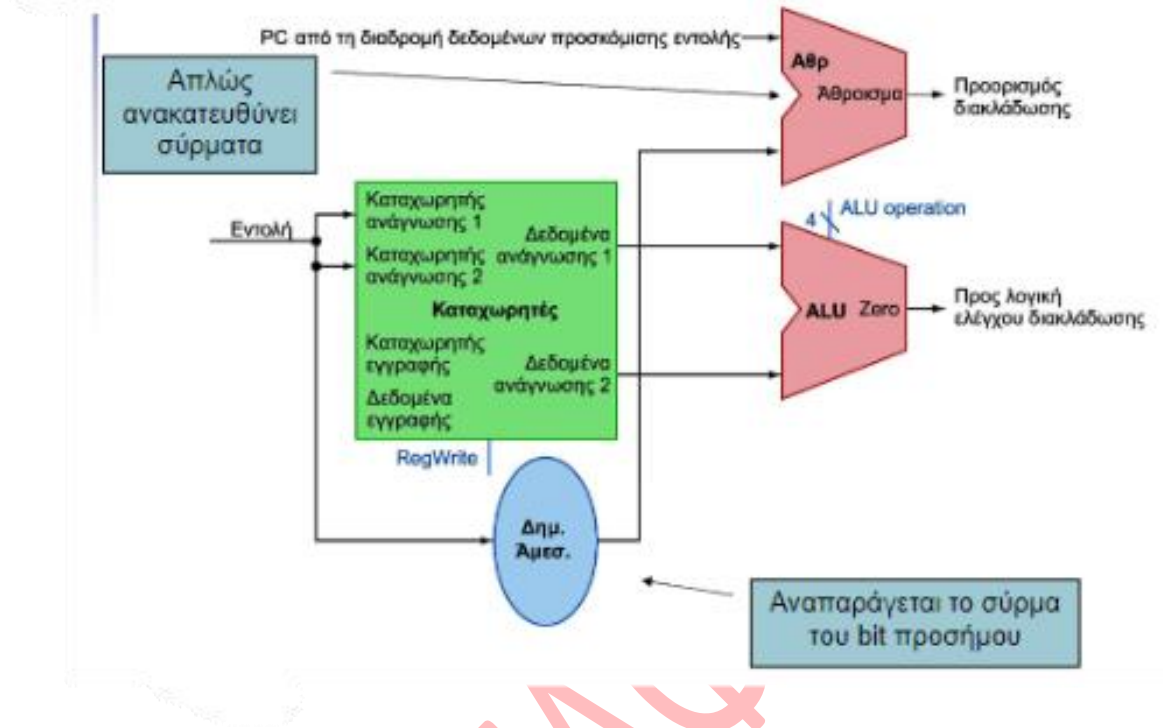
6. Αριστερή Ολίσθηση και Πρόσθεση

Η επεκταθείσα μετατόπιση μετατοπίζεται αριστερά κατά μία θέση (shift left by 1), επειδή οι εντολές στο RISC-V είναι ευθυγραμμισμένες σε λέξεις των 32-bit (δηλαδή, πολλαπλάσια των 4 bytes).
Έπειτα, το αποτέλεσμα προστίθεται στο PC για να προσδιοριστεί η τελική διεύθυνση διακλάδωσης.

Συμπέρασμα

Οι εντολές διακλάδωσης είναι κρίσιμες για τον έλεγχο της εκτέλεσης σε κάθε σύγχρονο επεξεργαστή. Εισάγουν δυναμική συμπεριφορά στην εκτέλεση, επιτρέποντας την υλοποίηση δομών όπως οι συνθήκες (if), οι βρόχοι (while, for), και άλλες μορφές ελέγχου ροής, που είναι απαραίτητες για την ευελιξία και ισχύ των προγραμμάτων.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Σύνθεση των στοιχείων

Η έννοια της "σύνθεσης των στοιχείων" αναφέρεται στον τρόπο που λειτουργεί μια διαδρομή δεδομένων σε έναν επεξεργαστή και είναι κρίσιμη για την κατανόηση της αρχιτεκτονικής των υπολογιστών, ειδικά στο πλαίσιο του RISC-V. Ακολουθεί μια αναλυτική εξήγηση των κυριότερων πτυχών της:

1. Πρωτόλεια Διαδρομή Δεδομένων

- **Εκτέλεση Εντολών:** Η πρωτόλεια διαδρομή δεδομένων εκτελεί μία εντολή σε έναν μόνο κύκλο ρολογιού. Αυτό σημαίνει ότι όλα τα απαραίτητα βήματα της εντολής ξεκινούν και ολοκληρώνονται κατά τη διάρκεια ενός συγκεκριμένου χρονικού διαστήματος.

2. Επεξεργασία Εντολών

- **Μοναδική Λειτουργία Αλληλεπίδρασης:** Κάθε στοιχείο της διαδρομής δεδομένων (όπως οι καταχωρητές, οι μονάδες ALU κ.λπ.) μπορεί να εκτελεί μόνο μία λειτουργία την φορά. Αυτή η προσέγγιση επιτρέπει την οργάνωση των λειτουργικών μονάδων με απλό και κατανοητό τρόπο.

3. Ξεχωριστές Μνήμες

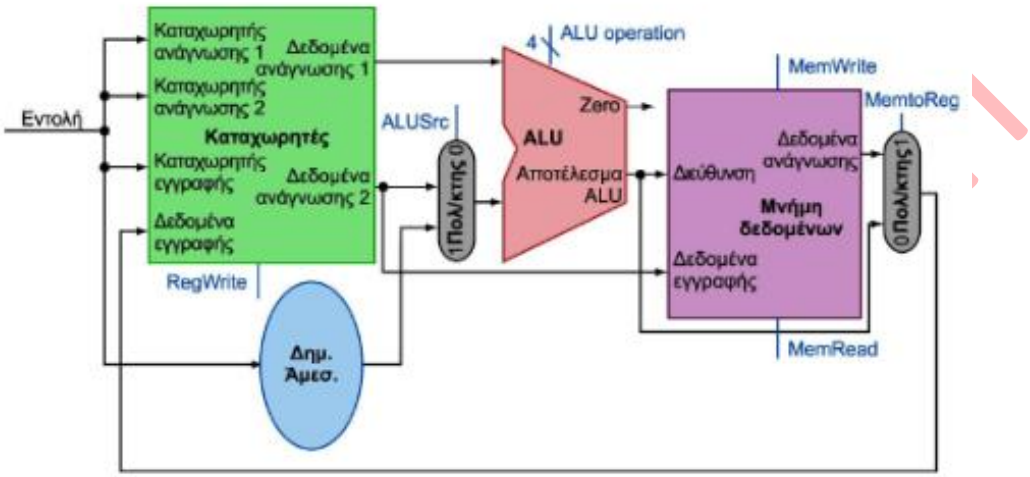
- **Αναγκαία Χωριστά Συστήματα Μνήμης:** Για να αποφευχθούν οι συγκρούσεις και οι καθυστερήσεις κατά την εκτέλεση εντολών, είναι απαραίτητο να υπάρχουν ξεχωριστές μνήμες για τις εντολές και τα δεδομένα. Με αυτό τον

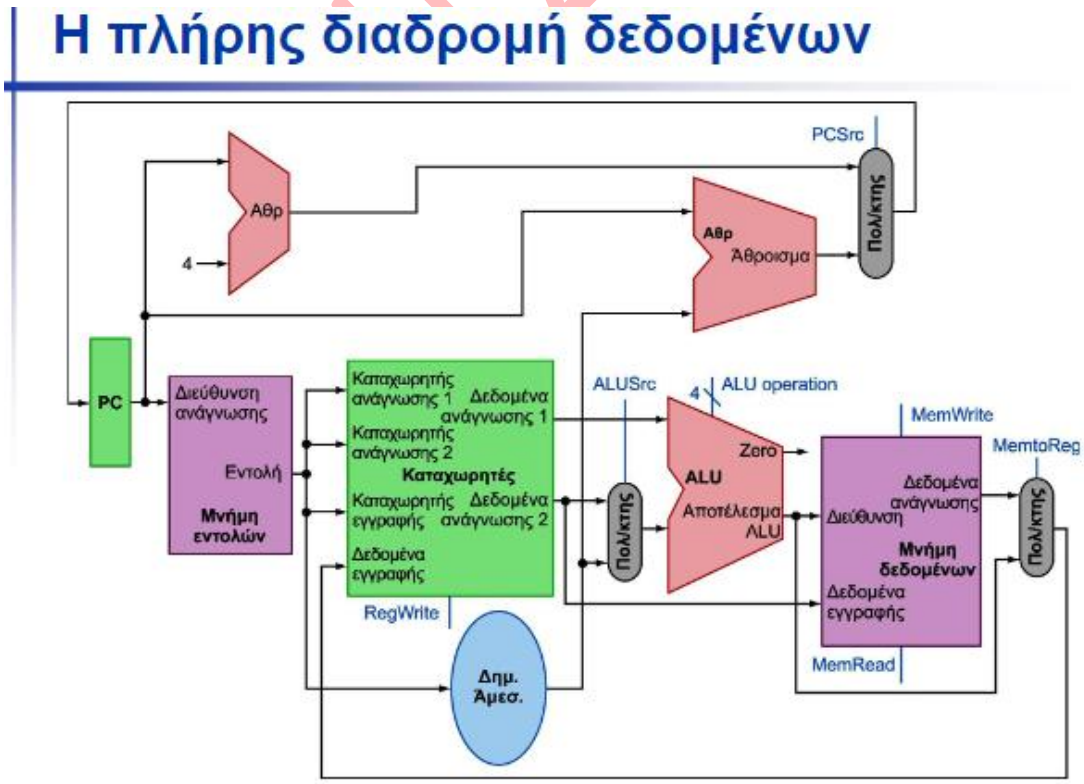
ΕΛΕΝΗ ΖΑΚ

τρόπο, οι εντολές μπορούν να προσκομίζονται χωρίς καθυστερήσεις από την μνήμη δεδομένων.

4. Χρήση Πολυπλεκτών

- **Διαχείριση Πολυάριθμων Πηγών Δεδομένων:** Στην περίπτωση που υπάρχουν εναλλακτικές πηγές προέλευσης δεδομένων για κάθε εντολή, οι πολυπλέκτες είναι αναγκαίοι. Επιτρέπουν την επιλογή της κατάλληλης πηγής δεδομένων που θα χρησιμοποιηθεί σε κάθε εκτέλεση εντολής, με βάση τις απαιτήσεις της εκάστοτε εντολής.





Γενική Επισκόπηση

Το διάγραμμα απεικονίζει την αρχιτεκτονική ενός απλοποιημένου επεξεργαστή,.Δείχνει πώς τα δεδομένα και οι εντολές μετακινούνται μεταξύ των διαφόρων λειτουργικών μονάδων για την εκτέλεση ενός προγράμματος. Οι βασικές μονάδες που διακρίνονται είναι:

- **PC (Program Counter):** Μετρητής Προγράμματος. Κρατά την διεύθυνση της επόμενης εντολής που θα εκτελεστεί.
- **Μνήμη Εντολών:** Αποθηκεύει τις εντολές του προγράμματος.
- **Καταχωρητές:** Ένα σύνολο μικρών, γρήγορων θέσεων μνήμης που χρησιμοποιούνται για την προσωρινή αποθήκευση δεδομένων που χρησιμοποιούνται συχνά.
- **ALU (Arithmetic Logic Unit):** Αριθμητική και Λογική Μονάδα. Εκτελεί αριθμητικές και λογικές πράξεις.
- **Μνήμη Δεδομένων:** Αποθηκεύει δεδομένα που χρησιμοποιούνται από το πρόγραμμα.
- **Μονάδα Ελέγχου (δεν απεικονίζεται ρητά):** Παράγει τα σήματα ελέγχου που καθοδηγούν τη λειτουργία των άλλων μονάδων. Τα σήματα ελέγχου όπως RegWrite, MemRead, MemWrite, MemtoReg, ALUSrc, ALU operation, και PCSrc υποδηλώνουν την ύπαρξή της.
- **Πολυπλέκτες (Πολ/κτης):** Επιλέγουν μία από τις πολλές εισόδους τους και την προωθούν στην έξοδο, βάσει ενός σήματος ελέγχου.
- **Μονάδα Άθροισης (+4):** Χρησιμοποιείται για την αύξηση της τιμής του PC κατά 4 (συνήθως το μέγεθος μιας εντολής σε bytes) για την προσέγγιση της επόμενης εντολής στη σειρά.
- **Μονάδα Δημιουργίας Άμεσης Τιμής (Δημ. Άμεσ.):** Παράγει μια άμεση τιμή (constant) που χρησιμοποιείται από την τρέχουσα εντολή.

Λεπτομερής Ανάλυση της Ροής Δεδομένων

Ας παρακολουθήσουμε τα βήματα εκτέλεσης μιας εντολής, όπως αυτά αντικατοπτρίζονται στο διάγραμμα:

1. **Ανάκτηση Εντολής (Instruction Fetch):**
    - Η διεύθυνση της επόμενης εντολής που πρέπει να εκτελεστεί βρίσκεται στον **PC**.
    - Η τιμή του **PC** χρησιμοποιείται ως διεύθυνση για την ανάγνωση της εντολής από τη **Μνήμη Εντολών**.
    - Η ανακτημένη **Εντολή** μεταφέρεται στα επόμενα στάδια.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Παράλληλα, η τιμή του **PC** περνάει από μια μονάδα **άθροισης (+4)**. Αυτό υπολογίζει τη διεύθυνση της επόμενης εντολής, υποθέτοντας ότι οι εντολές είναι διαδοχικές στη μνήμη. Η νέα αυτή διεύθυνση μπορεί να χρησιμοποιηθεί για την επόμενη ανάκτηση εντολής.

2. **Αποκωδικοποίηση Εντολής και Ανάγνωση Καταχωρητών (Instruction Decode and Register Read):**
   - Η ανακτημένη **Εντολή** αποκωδικοποιείται για να καθοριστεί η λειτουργία που πρέπει να εκτελεστεί και οι τελεστέοι (operands) που εμπλέκονται.
   - Μέρος της εντολής χρησιμοποιείται για τον προσδιορισμό των καταχωρητών που πρέπει να διαβαστούν από το μπλοκ των **Καταχωρητών**. Συνήθως, οι εντολές καθορίζουν έως και δύο καταχωρητές πηγής (Καταχωρητής Ανάγνωσης 1, Καταχωρητής Ανάγνωσης 2).
   - Οι τιμές που βρίσκονται στους καθορισμένους καταχωρητές εξάγονται ως **Δεδομένα Ανάγνωσης 1** και **Δεδομένα Ανάγνωσης 2**.
   - Ένα άλλο τμήμα της εντολής μπορεί να καθορίζει έναν καταχωρητή προορισμού (Καταχωρητής Εγγραφής), όπου θα αποθηκευτεί το αποτέλεσμα της λειτουργίας.

3. **Εκτέλεση (Execute):**
   - Η **ALU** εκτελεί την αριθμητική ή λογική πράξη.
   - Οι είσοδοι στην **ALU** προέρχονται από δύο πηγές, οι οποίες επιλέγονται από έναν **πολυπλέκτη (ALUSrc)**.
     - Η μία είσοδος είναι τα **Δεδομένα Ανάγνωσης 2** από τους καταχωρητές.
     - Η άλλη είσοδος είναι η **Έξοδος Δημ. Άμεσ.** (η άμεση τιμή που δημιουργήθηκε από την εντολή).
   - Η λειτουργία που εκτελεί η **ALU** καθορίζεται από το σήμα ελέγχου **ALU operation**, το οποίο παράγεται από τη μονάδα ελέγχου με βάση την αποκωδικοποιημένη εντολή.
   - Το αποτέλεσμα της λειτουργίας της **ALU** είναι το **Αποτέλεσμα ALU**.
   - Η **ALU** μπορεί επίσης να παράγει ένα σήμα **Zero**, το οποίο υποδεικνύει αν το αποτέλεσμα της λειτουργίας ήταν μηδέν (χρησιμοποιείται για εντολές διακλάδωσης).

4. **Πρόσβαση στη Μνήμη (Memory Access):**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Αυτό το στάδιο χρησιμοποιείται μόνο για εντολές που φορτώνουν δεδομένα από τη μνήμη (load) ή αποθηκεύουν δεδομένα στη μνήμη (store).
- Για εντολές **load**:
  - Η **Διεύθυνση Μνήμης** υπολογίζεται συνήθως από την **ALU** (π.χ., προσθέτοντας μια βάση καταχωρητή με μια μετατόπιση).
  - Η **Διεύθυνση Μνήμης** χρησιμοποιείται για την ανάγνωση δεδομένων από τη **Μνήμη Δεδομένων**.
  - Τα αναγνωσμένα **Δεδομένα Μνήμης (ανάγνωσης)** μεταφέρονται στο επόμενο στάδιο.
  - Το σήμα ελέγχου **MemRead** είναι ενεργό.
- Για εντολές **store**:
  - Η **Διεύθυνση Μνήμης** υπολογίζεται επίσης από την **ALU**.
  - Τα **Δεδομένα Ανάγνωσης 2** (η τιμή που πρέπει να αποθηκευτεί) χρησιμοποιούνται ως **Δεδομένα εγγραφής** για τη **Μνήμη Δεδομένων**.
  - Το σήμα ελέγχου **MemWrite** είναι ενεργό.

5. **Εγγραφή Αποτελέσματος (Write Back):**
   - Το αποτέλεσμα της εκτέλεσης γράφεται πίσω σε έναν καταχωρητή.
   - Η πηγή των δεδομένων που θα γραφτούν πίσω επιλέγεται από έναν **πολυπλέκτη (MemtoReg)**.
     - Η μία είσοδος είναι το **Αποτέλεσμα ALU** (για αριθμητικές/λογικές εντολές).
     - Η άλλη είσοδος είναι τα **Δεδομένα Μνήμης (ανάγνωσης)** (για εντολές load).
   - Το επιλεγμένο αποτέλεσμα (**Δεδομένα εγγραφής**) γράφεται στον καταχωρητή που καθορίζεται από το πεδίο **Καταχωρητής Εγγραφής** της αρχικής εντολής.
   - Το σήμα ελέγχου **RegWrite** είναι ενεργό.

Έλεγχος Ροής (Control Flow)

Εκτός από την επεξεργασία δεδομένων, ο επεξεργαστής πρέπει επίσης να μπορεί να αλλάξει τη ροή εκτέλεσης του προγράμματος (π.χ., με εντολές διακλάδωσης ή άλματος).

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Για τις διακλαδώσεις (branches), η διεύθυνση προορισμού υπολογίζεται συνήθως προσθέτοντας μια μετατόπιση (που εξάγεται από την εντολή) στην τρέχουσα τιμή του **PC** (ή στην **Άθροισμα** PC+4). Αυτός ο υπολογισμός γίνεται στην **Άθροισμα** που βρίσκεται κάτω από το αρχικό PC+4.
- Ένας **πολυπλέκτης (PCSrc)** επιλέγει την επόμενη τιμή που θα φορτωθεί στον **PC**.
    - Η μία είσοδος είναι η διεύθυνση της επόμενης εντολής στη σειρά (**Άθροισμα** PC+4).
    - Η άλλη είσοδος είναι η διεύθυνση προορισμού της διακλάδωσης (υπολογίζεται από την **Άθροισμα** κάτω).
    - Η επιλογή εξαρτάται από το σήμα ελέγχου **PCSrc**, το οποίο με τη σειρά του εξαρτάται από το είδος της τρέχουσας εντολής (αν είναι διακλάδωση) και το αποτέλεσμα της σύγκρισης (π.χ., το σήμα **Zero** από την ALU).

Σημαντικά Στοιχεία

- Τα σήματα ελέγχου (όπως RegWrite, MemRead, MemWrite, MemtoReg, ALUSrc, ALU operation, PCSrc) είναι κρίσιμα για τον συντονισμό των λειτουργιών των διαφόρων μονάδων. Παράγονται από τη μονάδα ελέγχου με βάση την αποκωδικοποιημένη εντολή.
- Οι πολυπλέκτες παίζουν καθοριστικό ρόλο στην επιλογή της σωστής πηγής δεδομένων για κάθε λειτουργική μονάδα.
- Η διαδρομή δεδομένων δείχνει την κίνηση των δεδομένων για την εκτέλεση μιας εντολής. Διαφορετικές εντολές μπορεί να χρησιμοποιούν διαφορετικά τμήματα της διαδρομής δεδομένων.

Έλεγχος ALU

Η Αριθμητική Λογική Μονάδα (ALU) είναι ένα κεντρικό εξάρτημα σε κάθε CPU, υπεύθυνο για την εκτέλεση αριθμητικών και λογικών πράξεων. Στην περίπτωση της ALU RISC-V, το κύριο χαρακτηριστικό της είναι η ικανότητα να προσαρμόζει τις λειτουργίες της ανάλογα με τις εντολές που εκτελούνται. Ας δούμε τα βασικά σημεία της χρήσης και ελέγχου της ALU:

Λειτουργίες της ALU

- **Φόρτωση/Αποθήκευση:**
    - Κατά τη διάρκεια των λειτουργιών φόρτωσης (load) και αποθήκευσης (store), η ALU εκτελεί την πράξη προσθήκης (add). Αυτό συμβαίνει γιατί χρειάζεται να υπολογίσει τη διεύθυνση μνήμης όπου θα γίνει η φόρτωση ή η αποθήκευση δεδομένων.
- **Διακλάδωση:**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Κατά τη διαδικασία της διακλάδωσης, η ALU χρησιμοποιεί τη λειτουργία αφαίρεσης (subtract). Η ALU συγκρίνει δύο τιμές, και αν είναι ίσες, υπολογίζει τη νέα διεύθυνση εκτέλεσης εντολής.

- **Εντολή μορφής R:**
  - Οι εντολές μορφής R, όπως είναι οι πράξεις αθροίσματος, υπολόγισαν την τιμή της εξόδου (F) με βάση τον κωδικό λειτουργίας (opcode) που σχετίζεται με την εντολή.

Γραμμές Ελέγχου της ALU

Η ALU έχει έναν ελεγκτή που καθορίζει τη λειτουργία της με βάση πολύ συγκεκριμένες γραμμές ελέγχου. Αυτές οι γραμμές ελέγχου επιτρέπουν στην ALU να γνωρίζει ποια λειτουργία να εκτελέσει:

Γραμμή ελέγχου ALU	Λειτουργία
0000	AND
0001	OR
0010	add
0110	subtract

Αυτή η αντιστοίχηση επιτρέπει τη γρήγορη είσοδο και έξοδο δεδομένων για διάφορες πράξεις, καθιστώντας την ALU αποδοτική σε περιβάλλοντα με συνεχείς υπολογισμούς.

Για να κατανοήσουμε πώς ελέγχεται η ALU, πρέπει να εξετάσουμε το πεδίο ελέγχου 2 bit που ονομάζεται ALUOp. Οι εντολές που φθάνουν στην ALU περιέχουν το κατάλληλο πεδίο κωδικοποίησης για αυτήν. Ακολουθούν παραδείγματα γραμμών ελέγχου και των αντίστοιχων λειτουργιών:

opcode	ALUOp	Λειτουργία	Πεδίο κωδ. λειτουργίας	Λειτουργία ALU	Γραμμή ελέγχου ALU
lw	00	load register	XXXXXXXXXXX	add	0010
sw	00	store register	XXXXXXXXXXX	add	0010
beq	01	branch on equal	XXXXXXXXXXX	subtract	0110
Μορφής R	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001

Σημεία Κλειδιά

- **ALUOp**: Το πεδίο ελέγχου 2 bit που καθορίζει τη λειτουργία της ALU.

- **opcode**: Ο κωδικός που χρησιμοποιείται για να καθοριστεί η εντολή και ποια γραμμή ελέγχου πρέπει να ενεργοποιηθεί στην ALU.
- Η ALU μπορεί να εκτελέσει πολλές διαφορετικές λειτουργίες σύμφωνα με τις διαφορετικές εντολές που δέχεται.

Αυτή η προσέγγιση διασφαλίζει ότι, ανάλογα με το τύπο της εντολής που διαβάζεται, η ALU θα εκτελέσει τις σωστές αριθμητικές ή λογικές πράξεις, παρέχοντας έτσι τα απαιτούμενα δεδομένα για τις επόμενες φάσεις επεξεργασίας.

Η κύρια μονάδα ελέγχου

- Τα σήματα ελέγχου συνάγονται από την εντολή

Όνομα (Θέση bit)	Πεδία 31:25	24:20	19:15	14:12	11:7	6:0
(α) Τύπου R	funct7	rs2	rs1	funct3	rd	opcode
(β) Τύπου I	immediate[11:0]		rs1	funct3	rd	opcode
(γ) Τύπου S	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(δ) Τύπου SB	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

ALUOp		Πεδίο funct7							Πεδίο funct3			
ALUOp1	ALUOp1	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Λειτουργία
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

Η
εικόνα απεικονίζει την κύρια μονάδα ελέγχου (control unit) ενός επεξεργαστή, με έμφαση στον τρόπο που τα σήματα ελέγχου δημιουργούνται από την εντολή (instru ction). Παρακάτω εξηγούνται τα κύρια μέρη της εικόνας:

Πίνακας "Όνομα (θέση bit)"

Αυτός ο πίνακας παρουσιάζει τα διαφορετικά είδη εντολών που υποστηρίζει η μονά δα ελέγχου. Κάθε είδος εντολής έχει διαφορετική δομή και απαιτεί διαφορετικά σή ματα ελέγχου:

1. **Τύπου R**: Χρησιμοποιείται για εντολές που εκτελούν αριθμητικές/λογικές λει τουργίες. Περιλαμβάνει τα πεδία funct7, rs2, rs1, funct3, rd, και opcode.

2. **Τύπου I**: Χρησιμοποιείται για εντολές που περιλαμβάνουν άμεση τιμή (imme diate). Περιλαμβάνει τα πεδία immediate[11:0], rs1, funct3, rd, και opcode.
3. **Τύπου S**: Χρησιμοποιείται για εντολές αποθήκευσης. Περιλαμβάνει τα πεδία immed[11:5], rs2, rs1, funct3, και immed[4:0].
4. **Τύπου SB**: Χρησιμοποιείται για εντολές διακλάδωσης (branch). Περιλαμβάνε ι τα πεδία immed[12,10:5], rs2, rs1, funct3, και immed[4,1:0].

Πίνακας "ALUOp και Λειτουργία"

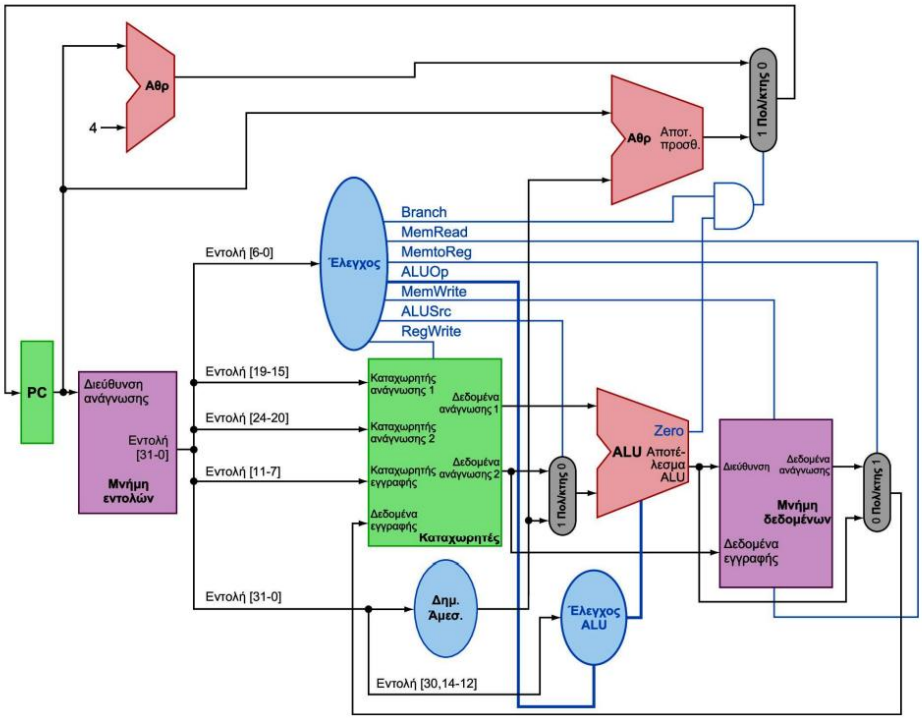
Αυτός ο πίνακας δείχνει το πώς οι τιμές των σημάτων ALUOp, funct7, και funct3 καθ ορίζουν τη λειτουργία της ALU (Arithmetic Logic Unit).

- Τα πεδία funct7 και funct3 είναι μέρος του opcode και παρέχουν επιπλέον πλ ηροφορίες για την εντολή που εκτελείται.
- Οι τιμές του ALUOp καθορίζουν τη λειτουργία της ALU:
    - **00**: Πρόσθεση (π.χ., για εντολές φορτώματος/αποθήκευσης).
    - **01**: Υπολογισμός για διακλαδώσεις (Branch).
    - **10**: Χρήση των πεδίων funct7 και funct3 για πιο εξειδικευμένες λειτου ργίες (π.χ., AND, OR, SUB).
    - **X**: Μη καθορισμένο (don't care).

Βασική Λειτουργία

Η μονάδα ελέγχου χρησιμοποιεί τα διάφορα πεδία της εντολής (π.χ., opcode, funct3 , funct7) για να διαμορφώσει τα σήματα ελέγχου που απαιτούνται για τη σωστή λειτ ουργία της ALU, της μνήμης, και των άλλων υποσυστημάτων του επεξεργαστή.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Η εικόνα παρουσιάζει τη "Διαδρομή Δεδομένων με τη Μονάδα Ελέγχου" ενός επεξε ργαστή. Είναι ένα διάγραμμα που απεικονίζει τη ροή δεδομένων και τον έλεγχο μέσ α στον επεξεργαστή, περιλαμβάνοντας τα κύρια μέρη του και τη συνεργασία τους. Α κολουθεί μια περιγραφή των βασικών στοιχείων της εικόνας:

1. Κύρια Στοιχεία της Διαδρομής Δεδομένων

1. **PC (Program Counter):**
   - Ο μετρητής προγράμματος που κρατάει τη διεύθυνση της τρέχουσας εντολής.
   - Ενημερώνεται με την επόμενη διεύθυνση εντολής (PC + 4 ή Branch).
2. **Μνήμη Εντολών (Instruction Memory):**
   - Περιέχει τις εντολές που πρέπει να εκτελεστούν.
   - Η εντολή που αντιστοιχεί στη διεύθυνση του PC ανακτάται από εδώ.
3. **Αποκωδικοποίηση Εντολής:**
   - Η εντολή χωρίζεται στα διάφορα πεδία της (π.χ., opcode, rs1, rs2, im mediate) για περαιτέρω επεξεργασία.
4. **Καταχωρητές (Registers):**
   - Ένα σύνολο από καταχωρητές που αποθηκεύουν δεδομένα και ενδιά μεσα αποτελέσματα.

ΕΛΕΝΗ ΖΑΦΕ

- Οι τιμές για τα rs1 και rs2 ανακτώνται από εδώ.

5. **ALU (Arithmetic Logic Unit):**
   - Εκτελεί αριθμητικές και λογικές πράξεις.
   - Επεξεργάζεται τιμές από τους καταχωρητές ή άμεσες τιμές (immediate).
6. **Μνήμη Δεδομένων (Data Memory):**
   - Χρησιμοποιείται για εντολές φορτώματος (load) και αποθήκευσης (store).
   - Η διεύθυνση και τα δεδομένα καθορίζονται από την ALU.

---

2. Σήματα Ελέγχου και Παραγωγή Αποφάσεων

1. **Μονάδα Ελέγχου (Control Unit):**
   - Παράγει σήματα ελέγχου βασισμένα στο opcode της εντολής.
   - Τα σήματα αυτά καθορίζουν τη λειτουργία της ALU, της μνήμης, και των καταχωρητών.
2. **Branch & Jump:**
   - Η απόφαση για διακλάδωση (branch) ή άλμα (jump) βασίζεται στην έξοδο της ALU και στα δεδομένα της εντολής.
3. **Multiplexers (MUX):**
   - Χρησιμοποιούνται για την επιλογή μεταξύ πολλών πιθανών εισόδων (π.χ., PC + 4 ή διεύθυνση branch).

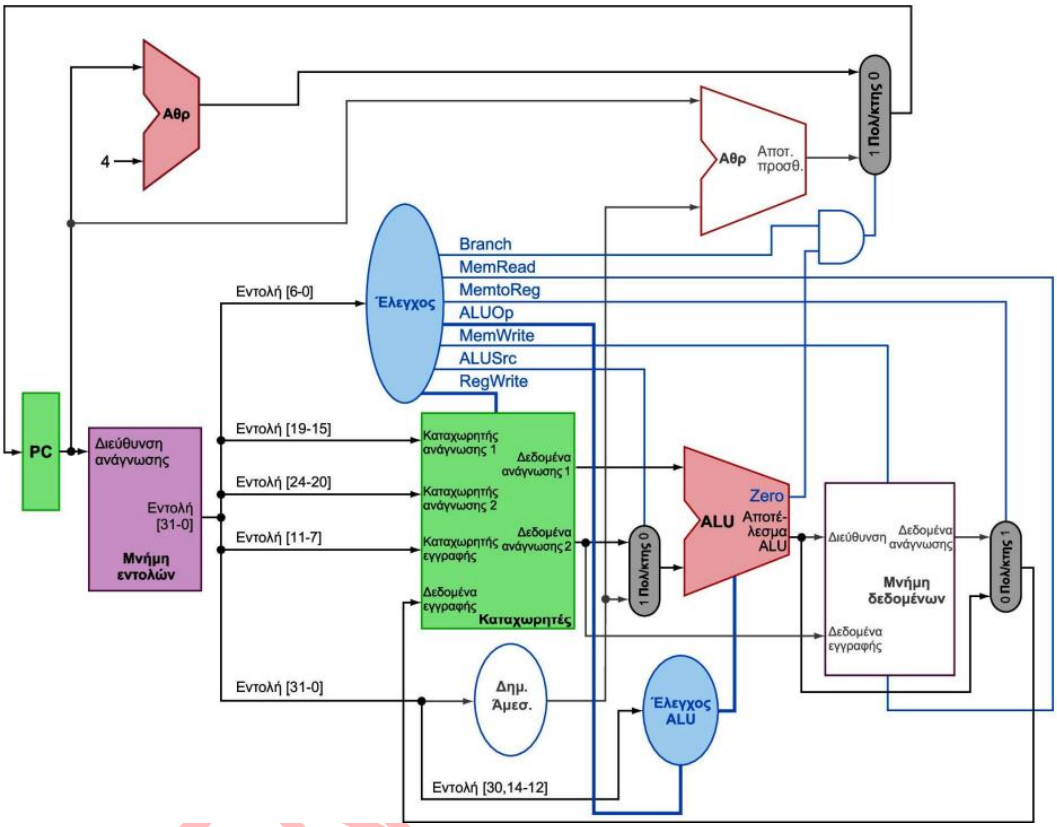
---

3. Ροή Δεδομένων

1. **Αρχή:**
   - Η διαδικασία ξεκινά με την ανάγνωση του PC και την ανάκτηση της εντολής από τη μνήμη εντολών.
2. **Επεξεργασία:**
   - Η εντολή αποκωδικοποιείται και τα δεδομένα φορτώνονται από τους καταχωρητές ή τη μνήμη.
3. **Υπολογισμός:**
   - Η ALU εκτελεί την πράξη που καθορίζεται από την εντολή (π.χ., πρόσθεση, AND, SUB).

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

4. **Αποθήκευση ή Αποφάσεις:**
    - Το αποτέλεσμα αποθηκεύεται στους καταχωρητές ή στη μνήμη.
    - Εναλλακτικά, γίνεται διακλάδωση ή άλμα με βάση τη συνθήκη.



Η εικόνα παρουσιάζει τη διαδρομή δεδομένων (data path) του επεξεργαστή για την εκτέλεση εντολών **μορφής R** (R-
type instructions). Οι εντολές αυτού του τύπου χρησιμοποιούνται για αριθμητικές κ αι λογικές πράξεις μεταξύ καταχωρητών. Ακολουθεί αναλυτική εξήγηση των βασικώ ν στοιχείων της εικόνας:

---

1. Μορφή Εντολής R

Οι εντολές μορφής R περιλαμβάνουν:

- **Rs1 (source register 1):** Ο πρώτος καταχωρητής εισόδου.
- **Rs2 (source register 2):** Ο δεύτερος καταχωρητής εισόδου.
- **Rd (destination register):** Ο καταχωρητής όπου αποθηκεύεται το αποτέλεσμ α.

- **Funct3 και Funct7:** Καθορίζουν τη συγκεκριμένη λειτουργία που θα εκτελέσε ι η ALU.
- **Opcode:** Καθορίζει την κατηγορία της εντολής.

---

2. Ροή Δεδομένων

Η διαδικασία εκτέλεσης μιας εντολής μορφής R περιλαμβάνει τα εξής βήματα:

1. **Ανάγνωση Εντολής από τη Μνήμη Εντολών:**
   - Η διεύθυνση εντολής (PC) χρησιμοποιείται για την ανάγνωση της εντο λής από τη μνήμη εντολών.
   - Η εντολή αποκωδικοποιείται και τα πεδία της (Rs1, Rs2, Rd, Funct3, F unct7) αποσπώνται.
2. **Ανάγνωση Δεδομένων από Καταχωρητές:**
   - Τα πεδία Rs1 και Rs2 χρησιμοποιούνται για την ανάγνωση των αντίστ οιχων τιμών από τους καταχωρητές.
   - Οι τιμές αυτές αποτελούν τις εισόδους για την ALU.
3. **Λειτουργία ALU:**
   - Η ALU εκτελεί την αριθμητική ή λογική πράξη που καθορίζεται από τα πεδία Funct3 και Funct7.
   - Η ALU λαμβάνει τις τιμές από τους Rs1 και Rs2 ως εισόδους και επιστ ρέφει το αποτέλεσμα.
4. **Αποθήκευση Αποτελέσματος:**
   - Το αποτέλεσμα της ALU αποθηκεύεται στον καταχωρητή Rd.
5. **Ενημέρωση του PC:**
   - Το PC ενημερώνεται ώστε να δείχνει στην επόμενη εντολή (PC + 4).

---

3. Σήματα Ελέγχου

Η μονάδα ελέγχου δημιουργεί τα απαραίτητα σήματα:

- **ALUOp:** Καθορίζει τη λειτουργία της ALU.
- **RegWrite:** Ενεργοποιεί την εγγραφή του αποτελέσματος στον καταχωρητή Rd .
- **Branch, MemRead, MemWrite:** Αυτά τα σήματα δεν χρησιμοποιούνται στις ε ντολές R, καθώς δεν περιλαμβάνουν φόρτωση από μνήμη ή διακλάδωση.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

4. Χαρακτηριστικά Εντολών R

Οι εντολές R είναι ιδιαίτερα αποδοτικές για πράξεις που απαιτούν μόνο καταχωρητέ ς και όχι άμεσες τιμές ή πρόσβαση στη μνήμη. Παραδείγματα τέτοιων εντολών περι λαμβάνουν:

- **ADD:** Πρόσθεση
- **SUB:** Αφαίρεση
- **AND/OR:** Λογικές πράξεις
- **SLT:** Σύγκριση (Set Less Than)

◆ Τι περιέχει μια R-type εντολή;

Μια εντολή μορφής R αποτελείται από τα εξής πεδία (32 bits συνολικά):

Πεδία	Μέγεθος	Περιγραφή
opcode	6 bits	Κωδικός λειτουργίας (συνήθως 000000 για R)
rs	5 bits	Πρώτος καταχωρητής πηγής (source 1)
rt	5 bits	Δεύτερος καταχωρητής πηγής (source 2)
rd	5 bits	Καταχωρητής προορισμού (destination)
shamt	5 bits	Αριθμός μετατόπισης (για shift εντολές)
funct	6 bits	Ορίζει την ακριβή πράξη (π.χ. add, sub)

◆ Παράδειγμα R-type εντολής:

add t0, t1, t2

Αυτό σημαίνει:

- Πρόσθεσε το περιεχόμενο των t1 και t2.
- Αποθήκευσε το αποτέλεσμα στο t0.

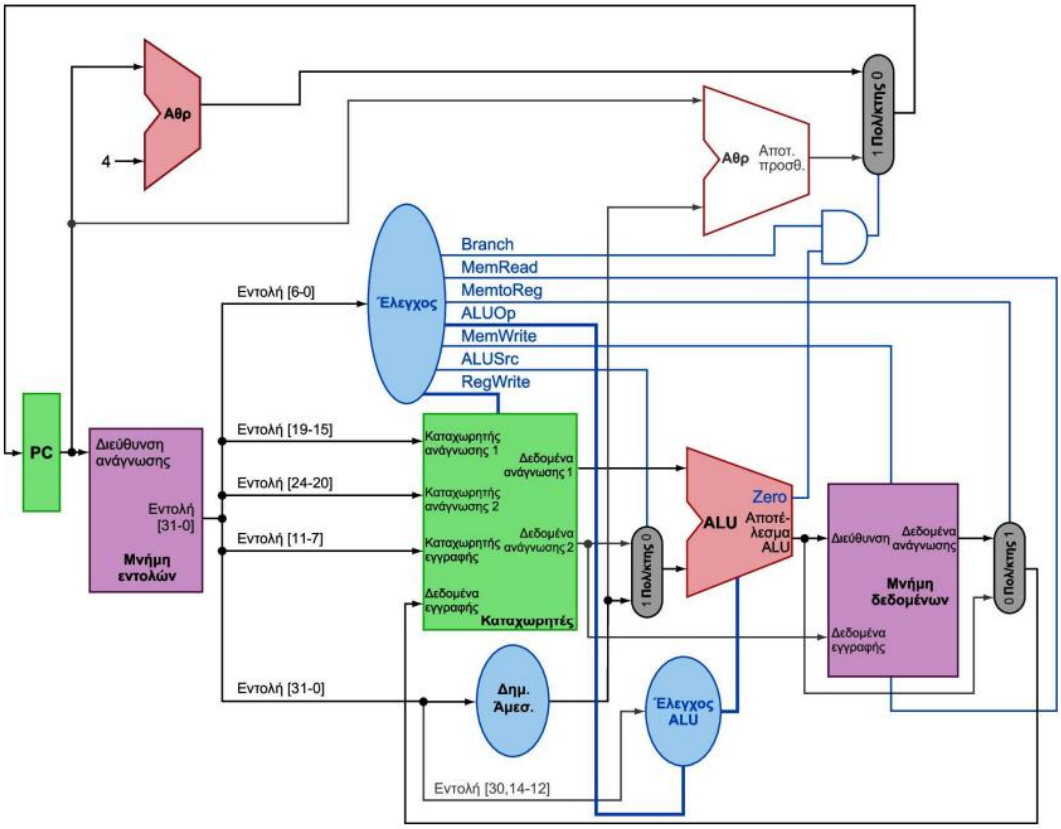
Αντιστοίχιση πεδίων:

- opcode: 000000
- rs: t1
- rt: t2
- rd: t0
- shamt: 00000 (δεν υπάρχει μετατόπιση)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- funct: 100000 (κωδικός για add)
- **Συνήθεις R-type εντολές:**

Εντολή	**Περιγραφή**
add	Πρόσθεση
sub	Αφαίρεση
and	Bitwise AND
or	Bitwise OR
slt	Set on less than
sll	Shift left logical
srl	Shift right logical



Η εικόνα παρουσιάζει τη διαδρομή δεδομένων (data path) και τη ροή της εκτέλεσης μιας εντολής **φόρτωσης δεδομένων** (load instruction) από τη μνήμη στον επεξεργαστή. Αυτές οι εντολές χρησιμοποιούνται για τη μεταφορά δεδομένων από τη μνήμη σε έναν καταχωρητή. Ακολουθεί αναλυτική περιγραφή:

---

1. Μορφή Εντολής Load

Οι εντολές **load** περιλαμβάνουν τα εξής πεδία:

- **Rs1 (source register 1):** Καταχωρητής που περιέχει τη βάση της διεύθυνσης μ νήμης.
- **Immediate:** Άμεση τιμή που προστίθεται στη βάση διεύθυνσης για τον υπολ ογισμό της τελικής διεύθυνσης.
- **Rd (destination register):** Ο καταχωρητής στον οποίο θα αποθηκευτούν τα δε δομένα που φορτώνονται από τη μνήμη.
- **Opcode:** Καθορίζει ότι η εντολή είναι τύπου load.
- **Funct3:** Καθορίζει τον τύπο της εντολής (π.χ., byte, word, κ.λπ.).

---

2. Ροή Δεδομένων

Η διαδικασία εκτέλεσης μιας εντολής **load** περιλαμβάνει τα εξής βήματα:

1. **Ανάγνωση Εντολής από τη Μνήμη Εντολών:**
    - Η διεύθυνση της εντολής (PC) χρησιμοποιείται για την ανάγνωση της εντολής από τη μνήμη εντολών.
    - Η εντολή αποκωδικοποιείται και τα πεδία της (Rs1, Immediate, Rd) απ οσπώνται.
2. **Ανάγνωση της Τιμής από τον Καταχωρητή Rs1:**
    - Η τιμή του καταχωρητή Rs1 ανακτάται από τον πίνακα καταχωρητών και χρησιμοποιείται ως βάση για τον υπολογισμό της διεύθυνσης μνή μης.
3. **Υπολογισμός Διεύθυνσης Μνήμης:**
    - Η ALU χρησιμοποιείται για να υπολογίσει τη διεύθυνση μνήμης προσ θέτοντας την τιμή του Rs1 με την άμεση τιμή (Immediate).
4. **Ανάγνωση Δεδομένων από τη Μνήμη:**
    - Η διεύθυνση που υπολογίστηκε από την ALU χρησιμοποιείται για την ανάγνωση δεδομένων από τη μνήμη.
5. **Αποθήκευση Δεδομένων στον Καταχωρητή Rd:**
    - Τα δεδομένα που ανακτήθηκαν από τη μνήμη αποθηκεύονται στον κ αταχωρητή Rd.
6. **Ενημέρωση του PC:**
    - Το PC ενημερώνεται ώστε να δείχνει στην επόμενη εντολή (PC + 4).

---

3. Σήματα Ελέγχου

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Η μονάδα ελέγχου δημιουργεί τα απαραίτητα σήματα:

- **MemRead:** Ενεργοποιεί την ανάγνωση από τη μνήμη.
- **RegWrite:** Ενεργοποιεί την εγγραφή στον καταχωρητή Rd.
- **ALUOp:** Καθορίζει ότι η ALU θα εκτελέσει πρόσθεση για τον υπολογισμό της διεύθυνσης.
- **Branch, MemWrite:** Αυτά τα σήματα δεν είναι ενεργά για τις εντολές load.

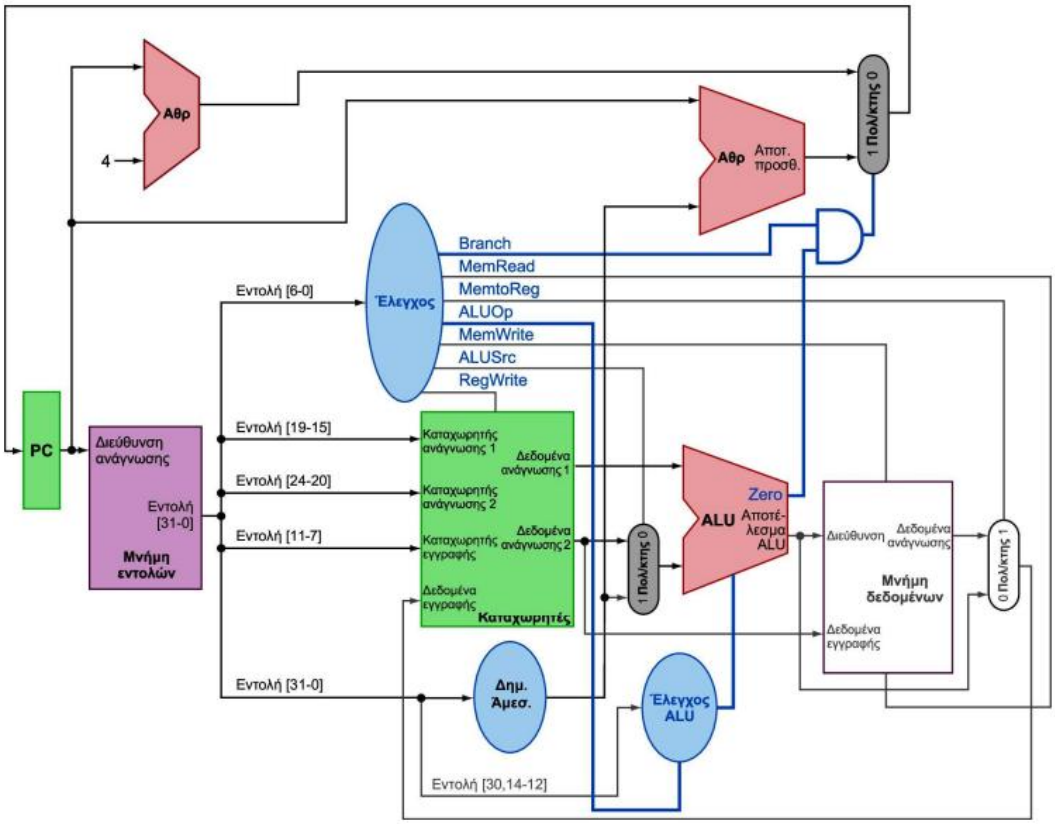
---

4. Χαρακτηριστικά Εντολών Load

Οι εντολές load είναι απαραίτητες για την επικοινωνία του επεξεργαστή με τη μνήμη και υποστηρίζουν διάφορα μεγέθη δεδομένων:

- **LB, LBU:** Φόρτωση byte με ή χωρίς πρόσημο.
- **LH, LHU:** Φόρτωση half-word με ή χωρίς πρόσημο.
- **LW:** Φόρτωση word.

ΦΕΙΡΗ



Η εικόνα παρουσιάζει τη διαδρομή δεδομένων (data path) και τη ροή της εκτέλεσης μιας εντολής **beq (Branch-on-Equal)**. Η εντολή αυτή χρησιμοποιείται για διακλάδωση (branching) όταν δύο καταχ ωρητές έχουν ίσες τιμές. Ακολουθεί αναλυτική περιγραφή:

1. Μορφή Εντολής Branch (beq)

Οι εντολές διακλάδωσης περιλαμβάνουν:

- **Rs1 (source register 1):** Ο πρώτος καταχωρητής που θα συγκριθεί.
- **Rs2 (source register 2):** Ο δεύτερος καταχωρητής που θα συγκριθεί.
- **Immediate:** Άμεση τιμή που καθορίζει την απόσταση (offset) για την αλλαγή στο PC.
- **Opcode:** Καθορίζει ότι η εντολή είναι τύπου branch.
- **Funct3:** Καθορίζει την υποκατηγορία της εντολής (π.χ., beq, bne).

2. Ροή Δεδομένων

Η εκτέλεση μιας εντολής **beq** περιλαμβάνει τα εξής βήματα:

1. **Ανάγνωση Εντολής από τη Μνήμη Εντολών:**
   - Η διεύθυνση της εντολής (PC) χρησιμοποιείται για την ανάγνωση της εντολής από τη μνήμη εντολών.
   - Η εντολή αποκωδικοποιείται και τα πεδία της (Rs1, Rs2, Immediate) α ποσπώνται.
2. **Ανάγνωση Τιμών από Καταχωρητές:**
   - Οι τιμές των καταχωρητών Rs1 και Rs2 ανακτώνται από τον πίνακα κα ταχωρητών.
3. **Σύγκριση Τιμών:**
   - Η ALU συγκρίνει τις τιμές των Rs1 και Rs2.
   - Η έξοδος της ALU είναι το σήμα **Zero**, το οποίο είναι ενεργό (1) αν οι δ ύο τιμές είναι ίσες.
4. **Υπολογισμός Νέας Διεύθυνσης (Branch Target):**
   - Αν το **Zero** είναι ενεργό, το Immediate (offset) προστίθεται στο PC (τρ έχουσα διεύθυνση) για να υπολογιστεί η νέα διεύθυνση.
   - Αν το **Zero** δεν είναι ενεργό, το PC απλώς αυξάνεται κατά 4 (PC + 4) γι α να δείξει στην επόμενη εντολή.
5. **Ενημέρωση του PC:**
   - Το PC ενημερώνεται με τη νέα διεύθυνση που υπολογίστηκε.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

3. Σήματα Ελέγχου

Η μονάδα ελέγχου δημιουργεί τα απαραίτητα σήματα:

- **Branch:** Ενεργοποιείται για εντολές διακλάδωσης.
- **ALUOp:** Καθορίζει ότι η ALU θα εκτελέσει σύγκριση (subtraction) για να παρά γει το σήμα **Zero**.
- **MemRead, MemWrite, RegWrite:** Αυτά τα σήματα δεν είναι ενεργά για τις ε ντολές beq, καθώς δεν περιλαμβάνουν πρόσβαση στη μνήμη ή εγγραφή σε κ αταχωρητή.

---

4. Χαρακτηριστικά Εντολών beq

Οι εντολές **beq** χρησιμοποιούνται για την υλοποίηση διακλαδώσεων σε προγράμματ α, όταν απαιτείται αλλαγή της ροής εκτέλεσης (π.χ., βρόχοι, συνθήκες). Αν το αποτέ λεσμα της σύγκρισης είναι ίσο, τότε η διακλάδωση πραγματοποιείται.

Η εικόνα δείχνει πώς συνεργάζονται η μονάδα ελέγχου, οι καταχωρητές, η ALU, και τ ο PC για την εκτέλεση της εντολής beq

Ζητήματα Απόδοσης στην ALU

Η απόδοση των υπολογιστικών μονάδων, και συγκεκριμένα της Αριθμητικής Λογικής Μονάδας (ALU), επηρεάζεται από διάφορους παράγοντες. Ακολουθούν οι βασικές έννοιες που εκτίθενται στο παραπάνω απόσπασμα:

1. Καθορισμός της Περιόδου Ρολογιού

Η **μεγαλύτερη καθυστέρηση** στην εκτέλεση των εντολών καθορίζει την συνολική **περίοδο του ρολογιού**. Αυτό σημαίνει ότι η διάρκεια που χρειάζεται για την ολοκλήρωση μιας εντολής πριν η CPU προχωρήσει στην επόμενη εντολή, εξαρτάται από το πιο «αργό» στοιχείο στην αλυσίδα εκτέλεσης.

2. Κρίσιμη Διαδρομή

Η **κρίσιμη διαδρομή** είναι η ασφαλής ακολουθία των στοιχείων που απαιτούνται για την εκτέλεση μιας εντολής. Στην προκειμένη περίπτωση, η κρίσιμη διαδρομή αναφέρεται στην εντολή load, η οποία περιλαμβάνει τα εξής βήματα:

- **Μνήμη εντολών:** Προσκόμιση της εντολής από τη μνήμη.
- **Αρχείο καταχωρητών:** Ανάγνωση και εγγραφή τιμών.
- **ALU:** Υπολογισμός της απαιτούμενης ενέργειας (π.χ., πρόσθεση, αφαίρεση).
- **Μνήμη δεδομένων:** Επίσκεψη στη μνήμη για φόρτωση/αποθήκευση δεδομένων.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Αρχείο καταχωρητών:** Ενημέρωση των καταχωρητών με τα αποτελέσματα.

3. Περιορισμοί στην Εκτέλεση

Η περιορισμένη σχεδίαση αναφέρει ότι **δεν μπορεί να διαφοροποιηθεί η περίοδος** για κάθε εντολή. Κάθε εντολή πρέπει να περιμένει να ολοκληρωθεί η πιο αργή εντολή, επομένως οι διαφορετικές εντολές δεν μπορούν να εκτελούνται ανά παράλληλους χρόνους.

4. Επιτάχυνση & Βελτίωση Απόδοσης

Η παραβίαση της σχεδιαστικής αρχής μπορεί να επιφέρει μειωμένη απόδοση, και για αυτό τον λόγο αποζητάται η βελτίωση της απόδοσης μέσω της **διοχέτευσης** (pipelining). Με τον σχεδιασμό μιας διπλής διαδρομής δεδομένων και την παράλληλη εκτέλεση διαφορετικών φάσεων μιας εντολής, μπορεί να προκύψει μεγαλύτερη αποτελεσματικότητα και επιτάχυνση στην εκτέλεση εντολών.

Διοχέτευση (Pipelining) σε Επεξεργαστές

Η εικόνα εξηγεί την έννοια της **διοχέτευσης (pipelining)** σε έναν επεξεργαστή, παρουσιάζοντας τη **βελτίωση απόδοσης** που προσφέρει μέσω της **επικάλυψης των σταδίων εκτέλεσης εντολών**.

🔹 1. Τι είναι η Διοχέτευση;

Η **διοχέτευση** είναι μια τεχνική κατά την οποία η εκτέλεση μιας εντολής **χωρίζεται σε διαδοχικά στάδια** (π.χ. Fetch, Decode, Execute κ.ά.), και αυτά τα στάδια εκτελούνται **παράλληλα** για διαφορετικές εντολές.
Με αυτόν τον τρόπο, **πολλαπλές εντολές βρίσκονται σε διαφορετικά στάδια εκτέλεσης ταυτόχρονα**, αυξάνοντας την απόδοση του επεξεργαστή.

🔹 2. Ανάλυση της Εικόνας

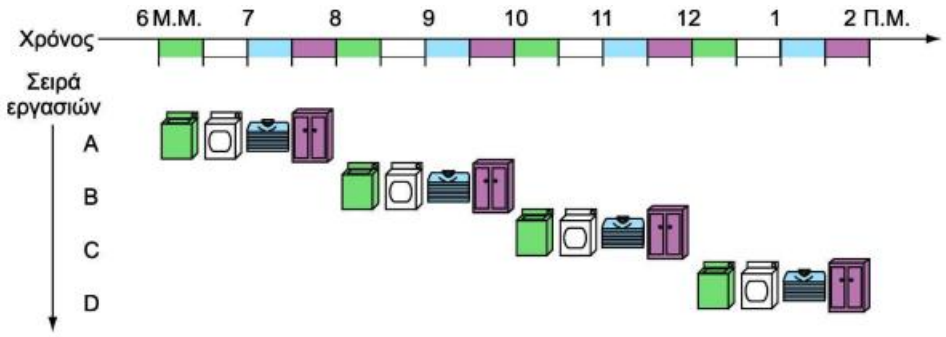
Α. Πάνω Διάγραμμα – Χωρίς Διοχέτευση (Non-Pipelined)

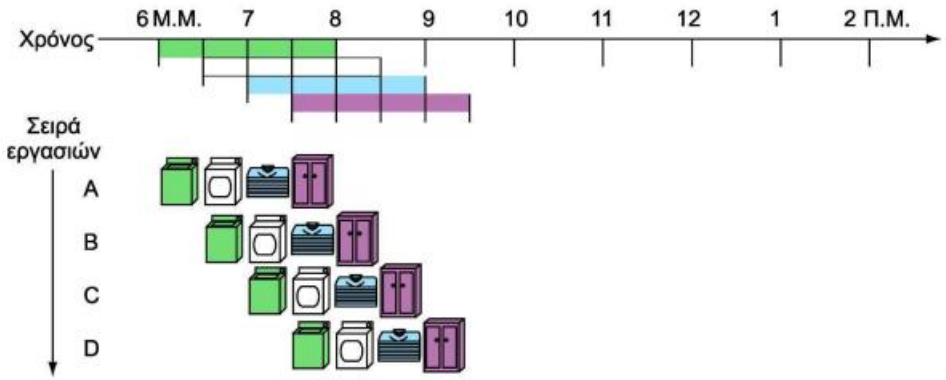
- Κάθε εργασία/εντολή **ολοκληρώνεται πλήρως** πριν ξεκινήσει η επόμενη.
- Υπάρχουν **νεκροί χρόνοι** (idle times) μεταξύ των σταδίων.
- **Απόδοση χαμηλότερη**, λιγότερη παραλληλία.
- Παράδειγμα: Σειριακή εκτέλεση εντολών σε έναν παλιό επεξεργαστή.

Β. Κάτω Διάγραμμα – Με Διοχέτευση (Pipelined)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Οι εντολές εκτελούνται με **επικάλυψη σταδίων** (π.χ. μια εντολή εκτελείται ενώ άλλη αποκωδικοποιείται).
- Κάθε στάδιο του pipeline δουλεύει σε **διαφορετική εντολή ταυτόχρονα**.
- Το σύστημα είναι **πιο αποδοτικό**, καθώς εκτελούνται περισσότερες εντολές ανά μονάδα χρόνου.
- Παράδειγμα: Εκτέλεση εντολών σε έναν μοντέρνο pipelined επεξεργαστή





🔹 3. Υπολογισμός Επιδόσεων

Με διοχέτευση:

Για παράδειγμα, αν εκτελούνται 4 εντολές:

📌 **Επιτάχυνση** =

8/3.5≈2.3

Δηλαδή, ο επεξεργαστής λειτουργεί περίπου **2.3 φορές γρηγορότερα** σε σχέση με εκτέλεση χωρίς διοχέτευση.

Θεωρητικά – Χωρίς καθυστερήσεις (stall-free):

Όταν δεν υπάρχουν εξαρτήσεις ή stalls:

📌 **Επιτάχυνση** ≈

2n/0.5n = 4

🔹 **4. Πλεονεκτήματα της Διοχέτευσης**

- **Αυξημένη απόδοση** μέσω ταυτόχρονης επεξεργασίας πολλών εντολών.
- **Αποτελεσματική χρήση πόρων** του επεξεργαστή.
- **Μειωμένος χρόνος εκτέλεσης** για πλήθος εντολών.

🔹 **5. Περιορισμοί της Διοχέτευσης**

- **Καθυστερήσεις (Stalls)** λόγω:
  - Εξαρτήσεων μεταξύ εντολών (data hazards).
  - Διακλαδώσεων (branch hazards).
- ⚠️ **Πολυπλοκότητα σχεδίασης** σε επίπεδο υλικού και ελέγχου (control logic).

Διοχέτευση RISC-V

Η διοχέτευση (ή pipelining) είναι μια τεχνική που χρησιμοποιείται στους επεξεργαστές για τη βελτίωση της απόδοσης καταμερίζοντας τη διαδικασία εκτέλεσης εντολών σε ξεχωριστά στάδια. Στην αρχιτεκτονική RISC-V, η διαδικασία εκτέλεσης μιας εντολής χωρίζεται σε πέντε διακριτά στάδια:

1. IF: Προσκόμιση Εντολής (Instruction Fetch)

Σε αυτό το στάδιο, ο επεξεργαστής προσκομίζει την επόμενη εντολή από τη μνήμη. Ο Μετρητής Προγράμματος (PC) καθορίζει τη διεύθυνση της εντολής που πρέπει να ανακτηθεί. Μετά την προσκόμιση, το PC προσαρμόζεται για να δείξει την επόμενη εντολή.

2. ID: Αποκωδικοποίηση Εντολής και Ανάγνωση Αρχείου Καταχωρητών (Instruction Decode and Register Read)

Αφού η εντολή προσκομιστεί, αποκωδικοποιείται για να καθοριστεί ο τύπος της εντολής και οι κατάλογοι που θα χρησιμοποιηθούν. Εδώ, ο επεξεργαστής διαβάζει τους απαραίτητους καταχωρητές που θα χρειαστούν για την εκτέλεση της εντολής.

3. EX: Εκτέλεση Λειτουργίας ή Υπολογισμός Διεύθυνσης (Execution or Address Calculation)

Σε αυτό το στάδιο, η Αριθμητική Λογική Μονάδα (ALU) εκτελεί τον υπολογισμό που απαιτείται από την εντολή. Αυτό περιλαμβάνει τη διεξαγωγή αριθμητικών ή λογικών πράξεων ή τον υπολογισμό της διεύθυνσης μνήμης για φόρτωση/αποθήκευση.

4. MEM: Προσπέλαση Μνήμης (Memory Access)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Αν η εντολή περιλαμβάνει φόρτωση ή αποθήκευση, αυτό το στάδιο είναι κατάλληλο για να προσπελάσει τη μνήμη δεδομένων. Αφού ολοκληρωθεί η προσπέλαση, οι τιμές που διαβάζονται ή αποθηκεύονται επιβεβαιώνονται.

5. WB: Εγγραφή του Αποτελέσματος (Write Back)

Σε αυτό το τελικό στάδιο, το αποτέλεσμα της εκτέλεσης ή της προσπέλασης μνήμης επιστρέφεται στον κατάλογο, ενημερώνοντας τη γνώση του επεξεργαστή για την κατάσταση των καταχωρητών.

Υποθέσεις:

Το κείμενο στην κορυφή θέτει τις εξής υποθέσεις για τον χρόνο εκτέλεσης κάθε σταδίου:

- **100 ps (picoseconds) για ανάγνωση/εγγραφή καταχωρητών:** Αυτό αναφέρεται στα στάδια όπου γίνεται πρόσβαση στο σύνολο των καταχωρητών για ανάγνωση των τελεστέων ή για εγγραφή του αποτελέσματος.
- **200 ps για τα άλλα στάδια:** Αυτό αφορά τα υπόλοιπα στάδια της διοχέτευσης, όπως η προσκόμιση της εντολής (Instruction Fetch), η λειτουργία της ALU (Arithmetic Logic Unit), και η προσπέλαση της μνήμης (Memory Access).

Πίνακας Σύγκρισης:

Ο πίνακας παρουσιάζει τέσσερις διαφορετικούς τύπους εντολών και τον χρόνο που απαιτείται για κάθε στάδιο της διοχέτευσης, καθώς και τον συνολικό χρόνο εκτέλεσης για κάθε εντολή.

Εντολή	Προσκόμιση εντολής	Ανάγνωση καταχωρητών	Λειτουργία ALU	Προσπέλαση μνήμης	Εγγραφή καταχωρητών	Συνολικός χρόνος
lw	200 ps	100 ps	200 ps	200 ps	100 ps	**800 ps**
sw	200 ps	100 ps	200 ps	200 ps		**700 ps**
μορφή R	200 ps	100 ps	200 ps		100 ps	**600 ps**
beq	200 ps	100 ps	200 ps			**500 ps**

Ας εξετάσουμε κάθε εντολή ξεχωριστά:

- **lw (load word):** Αυτή η εντολή φορτώνει μια λέξη δεδομένων από τη μνήμη σε έναν καταχωρητή. Περνάει από όλα τα πέντε στάδια της διοχέτευσης:
    - Προσκόμιση εντολής (200 ps)

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Ανάγνωση καταχωρητών (100 ps) - για τον υπολογισμό της διεύθυνσης μνήμης
- Λειτουργία ALU (200 ps) - υπολογισμός της τελικής διεύθυνσης μνήμης
- Προσπέλαση μνήμης (200 ps) - ανάγνωση των δεδομένων από τη μνήμη
- Εγγραφή καταχωρητών (100 ps) - εγγραφή των αναγνωσμένων δεδομένων στον προορισμό καταχωρητή
- **Συνολικός χρόνος: 800 ps**

- **sw (store word):** Αυτή η εντολή αποθηκεύει μια λέξη δεδομένων από έναν καταχωρητή στη μνήμη. Δεν χρειάζεται το στάδιο της εγγραφής καταχωρητών στο τέλος.
  - Προσκόμιση εντολής (200 ps)
  - Ανάγνωση καταχωρητών (100 ps) - για την τιμή που θα αποθηκευτεί και τη διεύθυνση μνήμης
  - Λειτουργία ALU (200 ps) - υπολογισμός της διεύθυνσης μνήμης
  - Προσπέλαση μνήμης (200 ps) - εγγραφή των δεδομένων στη μνήμη
  - **Συνολικός χρόνος: 700 ps**
- **μορφή R (R-type):** Αυτές είναι αριθμητικές ή λογικές εντολές που λειτουργούν απευθείας σε καταχωρητές (π.χ., πρόσθεση, αφαίρεση). Δεν χρειάζονται πρόσβαση στη μνήμη.
  - Προσκόμιση εντολής (200 ps)
  - Ανάγνωση καταχωρητών (100 ps) - ανάγνωση των τελεστέων από τους καταχωρητές
  - Λειτουργία ALU (200 ps) - εκτέλεση της αριθμητικής/λογικής πράξης
  - Εγγραφή καταχωρητών (100 ps) - εγγραφή του αποτελέσματος σε έναν καταχωρητή
  - **Συνολικός χρόνος: 600 ps**
- **beq (branch on equal):** Αυτή είναι μια εντολή διακλάδωσης που ελέγχει αν δύο καταχωρητές είναι ίσοι και, αν ναι, αλλάζει τη ροή εκτέλεσης του προγράμματος. Δεν χρειάζεται πρόσβαση στη μνήμη ή εγγραφή σε καταχωρητή (με την έννοια του αποτελέσματος της σύγκρισης).
  - Προσκόμιση εντολής (200 ps)
  - Ανάγνωση καταχωρητών (100 ps) - ανάγνωση των δύο καταχωρητών που θα συγκριθούν

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Λειτουργία ALU (200 ps) - εκτέλεση της σύγκρισης (συνήθως αφαίρεση για έλεγχο ισότητας)
- **Συνολικός χρόνος: 500 ps**

Σημασία για την Απόδοση:

Παρόλο που κάθε εντολή εξακολουθεί να χρειάζεται έναν συγκεκριμένο συνολικό χρόνο για να εκτελεστεί, η διοχέτευση αυξάνει την **throughput** (ρυθμό διεκπεραίωσης) του επεξεργαστή. Σε έναν ιδανικό σενάριο όπου οι εντολές μπορούν να περνούν ομαλά από τη διοχέτευση χωρίς καθυστερήσεις (hazards), μια νέα εντολή μπορεί να ξεκινάει σε κάθε κύκλο ρολογιού.

Ο χρόνος ενός κύκλου ρολογιού σε μια διοχετευμένη αρχιτεκτονική καθορίζεται από το πιο αργό στάδιο. Στην περίπτωση αυτή, το πιο αργό στάδιο είναι 200 ps (προσκόμιση εντολής, λειτουργία ALU, προσπέλαση μνήμης). Επομένως, ο θεωρητικός χρόνος ενός κύκλου ρολογιού είναι 200 ps.

Σε μια διοχέτευση πέντε σταδίων, μετά την αρχική πλήρωση της διοχέτευσης (που θα πάρει 5 κύκλους), μια εντολή θα ολοκληρώνεται (θεωρητικά) σε κάθε κύκλο ρολογιού.

Σύγκριση με Εκτέλεση Ενός Κύκλου:

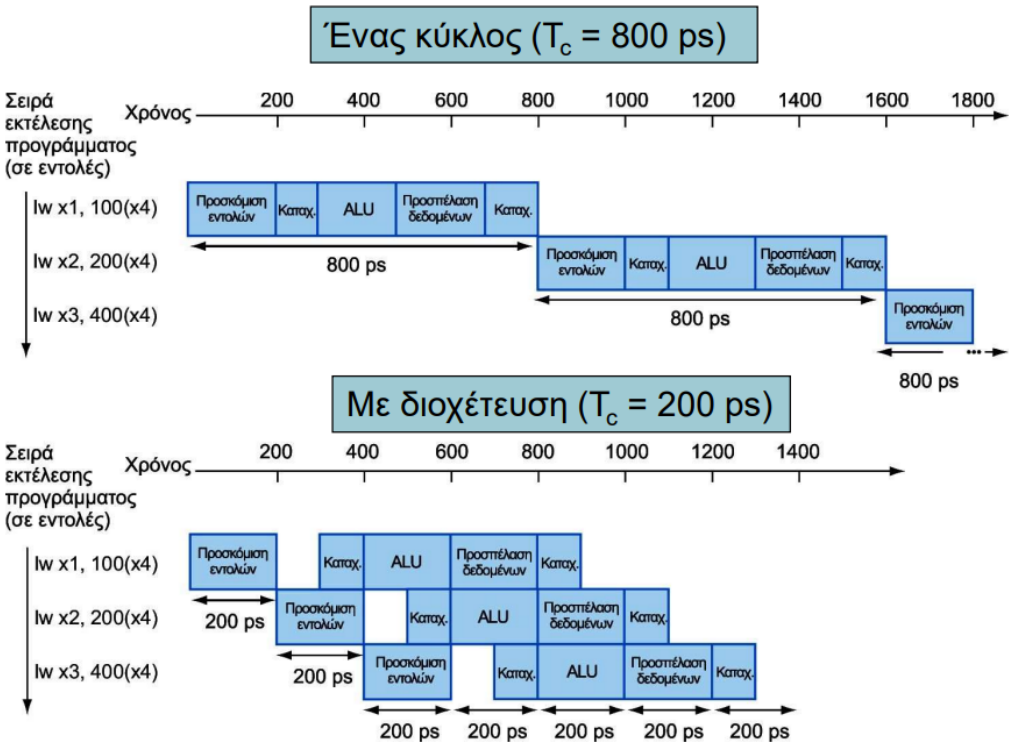
Σε έναν επεξεργαστή που εκτελεί μια εντολή σε έναν μόνο κύκλο, ο χρόνος του κύκλου θα έπρεπε να είναι αρκετά μεγάλος ώστε να χωρέσει το άθροισμα των χρόνων όλων των απαραίτητων σταδίων για την πιο σύνθετη εντολή (στην περίπτωσή μας, την lw που χρειάζεται 800 ps). Έτσι, κάθε εντολή θα χρειαζόταν 800 ps για να εκτελεστεί, ακόμα και οι πιο απλές.

Η διοχέτευση επιτρέπει να έχουμε έναν μικρότερο χρόνο κύκλου ρολογιού (200 ps στην περίπτωσή μας) και να επιτυγχάνουμε υψηλότερο ρυθμό ολοκλήρωσης εντολών στην πάροδο του χρόνου, παρά το γεγονός ότι η καθυστέρηση (latency) για την εκτέλεση μιας μεμονωμένης εντολής (όπως η lw) μπορεί να είναι μεγαλύτερη από τον χρόνο ενός κύκλου.

Συμπέρασμα:

Ο πίνακας δείχνει τον χρόνο που καταναλώνει κάθε στάδιο της διοχέτευσης για διαφορετικούς τύπους εντολών. Η διοχέτευση βελτιώνει την απόδοση επιτρέποντας την επικάλυψη της εκτέλεσης πολλαπλών εντολών, οδηγώντας σε υψηλότερο ρυθμό διεκπεραίωσης εντολών σε σύγκριση με την εκτέλεση μιας εντολής σε κάθε κύκλο.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Ας αναλύσουμε την εικόνα που συγκρίνει την εκτέλεση εντολών χωρίς και με διοχέτευση (pipeline), εστιάζοντας στην απόδοση και τον χρόνο εκτέλεσης.

Επάνω Διάγραμμα: Εκτέλεση Ενός Κύκλου (T_c = 800 ps)

- **Άξονας Χρόνου:** Οριζόντιος άξονας που δείχνει την πρόοδο του χρόνου σε picoseconds (ps).
- **Σειρά Εκτέλεσης Προγράμματος (σε εντολές):** Κάθε οριζόντια γραμμή αντιπροσωπεύει την εκτέλεση μιας εντολής lw (load word). Υπάρχουν τρεις διαδοχικές εντολές lw: lw x1, lw x2, και lw x3. Υποδεικνύεται ότι κάθε εντολή επαναλαμβάνεται 4 φορές, αλλά το διάγραμμα δείχνει την εκτέλεση των πρώτων εμφανίσεων.
- **Χρονική Διάρκεια Εντολής:** Κάθε εντολή lw χρειάζεται **800 ps** για να ολοκληρωθεί. Αυτό αντιπροσωπεύεται από το οριζόντιο μπλοκ που καλύπτει αυτό το χρονικό διάστημα.
- **Στάδια Εκτέλεσης:** Μέσα σε κάθε μπλοκ των 800 ps, φαίνονται τα διαδοχικά στάδια που περνάει η εντολή:
    - **Προσκόμιση εντολής (Προσκόμιση εντολών):** Το πρώτο τμήμα του μπλοκ.
    - **Ανάγνωση καταχωρητών (Καταχ.):** Το δεύτερο τμήμα.
    - **Λειτουργία ALU (ALU):** Το τρίτο τμήμα.

- **Προσπέλαση δεδομένων (Προσπέλαση δεδομένων):** Το τέταρτο τμήμα.
- **Εγγραφή καταχωρητών (Καταχ.):** Το πέμπτο τμήμα.

- **Παρατήρηση:** Κάθε εντολή ξεκινά την εκτέλεσή της μόνο αφού η προηγούμενη έχει ολοκληρωθεί πλήρως. Αυτό σημαίνει ότι υπάρχει διαδοχική εκτέλεση.

Κάτω Διάγραμμα: Με Διοχέτευση (T_c = 200 ps)

- **Άξονας Χρόνου:** Όμοιος με το επάνω διάγραμμα.
- **Σειρά Εκτέλεσης Προγράμματος (σε εντολές):** Εμφανίζονται οι ίδιες τρεις εντολές lw.
- **Χρονική Διάρκεια Σταδίου:** Ο χρόνος ενός κύκλου ρολογιού είναι **200 ps**. Κάθε στάδιο της διοχέτευσης διαρκεί έναν κύκλο ρολογιού.
- **Επικάλυψη Σταδίων:** Παρατηρούμε ότι ενώ η πρώτη εντολή (lw x1) βρίσκεται στο στάδιο της "Ανάγνωσης καταχωρητών" (μετά από 200 ps), η δεύτερη εντολή (lw x2) ξεκινά το στάδιο της "Προσκόμισης εντολής". Αυτό δείχνει την επικάλυψη της εκτέλεσης διαφορετικών εντολών σε διαφορετικά στάδια.
- **Στάδια Διοχέτευσης:** Κάθε εντολή περνάει από τα ίδια πέντε στάδια, αλλά κάθε στάδιο εκτελείται σε έναν ξεχωριστό κύκλο ρολογιού.
- **Παρατήρηση:** Μετά την αρχική "πλήρωση" της διοχέτευσης (όπου οι πρώτες εντολές καταλαμβάνουν τα διαδοχικά στάδια), μια νέα εντολή ολοκληρώνεται περίπου κάθε κύκλο ρολογιού.

Σύγκριση Απόδοσης:

- **Χωρίς Διοχέτευση:** Για να εκτελεστούν τρεις εντολές lw, χρειάζονται 3 * 800 ps = **2400 ps**. Η συχνότητα ρολογιού θα ήταν περιορισμένη από τον χρόνο εκτέλεσης ολόκληρης της εντολής (800 ps ανά εντολή).
- **Με Διοχέτευση:**
  - Η πρώτη εντολή χρειάζεται 5 κύκλους * 200 ps/κύκλο = 1000 ps για να ολοκληρωθεί (latency).
  - Ωστόσο, μετά την αρχική καθυστέρηση, κάθε επόμενη εντολή ολοκληρώνεται σε κάθε κύκλο ρολογιού (throughput).
  - Για να ολοκληρωθούν οι τρεις εντολές, χρειάζονται 5 (για την πρώτη) + 2 (οι υπόλοιπες δύο που ολοκληρώνονται σε κάθε κύκλο) = 7 κύκλοι * 200 ps/κύκλο = **1400 ps**.

Σημαντικά Σημεία:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Η διοχέτευση δεν μειώνει τον χρόνο που χρειάζεται μια μεμονωμένη εντολή για να ολοκληρωθεί (latency). Στην πραγματικότητα, μπορεί να τον αυξήσει ελαφρώς λόγω της προσθήκης latch μεταξύ των σταδίων.
- Η διοχέτευση αυξάνει σημαντικά την **throughput** του επεξεργαστή, δηλαδή τον αριθμό των εντολών που μπορούν να εκτελεστούν ανά μονάδα χρόνου. Στο παράδειγμα, ενώ χωρίς διοχέτευση μια εντολή ολοκληρώνεται κάθε 800 ps, με διοχέτευση, μετά την αρχική καθυστέρηση, μια εντολή ολοκληρώνεται κάθε 200 ps.
- Ο χρόνος του κύκλου ρολογιού (T_c) στην διοχετευμένη αρχιτεκτονική καθορίζεται από τον χρόνο του πιο αργού σταδίου. Στην περίπτωση αυτή, υποτίθεται ότι όλα τα στάδια διαρκούν 200 ps.

Συμπέρασμα:

Το διάγραμμα απεικονίζει ξεκάθαρα πώς η διοχέτευση βελτιώνει την απόδοση ενός επεξεργαστή επιτρέποντας την ταυτόχρονη εκτέλεση διαφορετικών σταδίων πολλαπλών εντολών. Αυτό οδηγεί σε σημαντικά υψηλότερο ρυθμό ολοκλήρωσης εντολών σε σύγκριση με την διαδοχική εκτέλεση εντολών σε έναν κύκλο.

Εξήγηση της Επιτάχυνσης με Διοχέτευση

1. Ισορροπημένα Στάδια

- Αν όλα τα στάδια διαρκούν τον ίδιο χρονικό διάστημα, τότε επιτυγχάνεται η μέγιστη απόδοση.
- Ο τύπος για να υπολογίσει την επιτάχυνση είναι:

Χρόνος μμεταξύ εντολών με διοχέτευση= Χρόνος μμεταξύ εντολών χωρίς διοχέτευση/Αριθμός σταδίων

Αυτό σημαίνει ότι αν ο χρόνος εκτέλεσης για μια εντολή μπορεί να διαχωριστεί σε 5 στάδια που διαρκούν το ίδιο χρονικό διάστημα, η συχνότητα εκτέλεσης των εντολών θα αυξηθεί.

2. Μη Ισορροπημένα Στάδια

- Όταν τα στάδια δεν είναι ισορροπημένα (κάποια διαρκούν περισσότερο από άλλα), τότε η συνολική επιτάχυνση που θα μπορούσε να επιτευχθεί θα είναι μειωμένη.
- Η επιτάχυνση είναι αποτέλεσμα της αυξημένης διεκπεραιωτικής ικανότητας, δηλαδή της ικανότητας του συστήματος να εκτελεί περισσότερες εντολές σε πιο σύντομο χρονικό διάστημα.

3. . Λανθάνων Χρόνος

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Σημαντικό είναι να κατανοήσουμε ότι η επιτάχυνση μέσω διοχέτευσης δεν επηρεάζει τον λανθάνοντα χρόνο (ο χρόνος που απαιτείται για να ολοκληρωθεί μία εντολή).
- Ειδικότερα, η χρήση της διοχέτευσης αυξάνει την παραγωγικότητα της εκτέλεσης εντολών, αλλά ο χρόνος που απαιτείται για να ολοκληρωθεί κάθε μεμονωμένη εντολή παραμένει ο ίδιος.

Διοχέτευση και Σχεδίαση ISA

Στη συγκεκριμένη περίπτωση, αναφερόμαστε στο RISC-V, μια αρχιτεκτονική που έχει σχεδιαστεί για να αξιοποιεί στο έπακρο τις δυνατότητες διοχέτευσης.

Βασικά Σημεία

- **Σχέδιο Εντολών 32-bit**:
  - Όλες οι εντολές του RISC-V έχουν μήκος 32 bits, κάτι που απλοποιεί τη διαδικασία προσκόμισης (fetching) και αποκωδικοποίησης (decoding), καθώς ο επεξεργαστής μπορεί να διαχειρίζεται όλες τις εντολές με τον ίδιο τρόπο και σε έναν μόνο κύκλο.
- **Σύγκριση με άλλες Αρχιτεκτονικές**:
  - Σε αντίθεση με άλλες αρχιτεκτονικές, όπως η x86, όπου οι εντολές μπορεί να έχουν μήκος από 1 έως 17 bytes, η ομοιομορφία των 32-bit εντολών στο RISC-V καθιστά την σχεδίαση και εκτέλεση λιγότερο περίπλοκες.
- **Περιορισμένος Αριθμός Μορφών Εντολών**:
  - Οι λίγες κανονικές μορφές εντολών διευκολύνουν την αποκωδικοποίηση και επιτρέπουν την αποτελεσματικότερη χρήση των πόρων του επεξεργαστή.
- **Διαδικασία Αποκωδικοποίησης και Ανάγνωσης**:
  - Στη διοχέτευση του RISC-V, είναι εφικτή η αποκωδικοποίηση και η ανάγνωση των καταχωρητών να πραγματοποιούνται σε ένα στάδιο (cycle), ελαχιστοποιώντας την καθυστέρηση που προκαλεί συνήθως η ανάγκη για περισσότερα βήματα.
- **Διευθυνσιοδότηση Φόρτωσης/Αποθήκευσης**:
  - Ο RISC-V επιτρέπει την υπολογιστική διεύθυνση κατά τη διάρκεια του τρίτου σταδίου και την προσπέλαση της μνήμης στο τέταρτο, κάτι που ενισχύει περαιτέρω την ταχύτητα απόδοσης και την αποδοτικότητα.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Κίνδυνοι στην Υλοποίηση Διοχέτευσης

Στην υλοποίηση ενός επεξεργαστή με διοχέτευση, υπάρχουν διάφοροι κίνδυνοι που μπορούν να επηρεάσουν την απόδοση και την ομαλή εκτέλεση των εντολών. Αυτοί οι κίνδυνοι διακρίνονται σε τρεις βασικές κατηγορίες:

1. Δομικοί Κίνδυνοι

Αυτοί οι κίνδυνοι προκύπτουν όταν υπάρχει διαμάχη για τη χρήση ενός πόρου. Σε ένα επεξεργαστή που χρησιμοποιεί διοχέτευση, αυτό μπορεί να συμβεί όταν περισσότερες από μία εντολές χρειάζονται πρόσβαση στον ίδιο φυσικό πόρο ταυτόχρονα, όπως η μνήμη ή η ALU (Αριθμητική/Λογική Μονάδα). Όταν αυτό συμβαίνει, μπορεί να προκύψει καθυστέρηση στην εκτέλεση των εντολών, καθώς οι εντολές δεν μπορούν να εκτελούνται ταυτόχρονα.

2. Κίνδυνοι Δεδομένων

Αυτοί οι κίνδυνοι εμφανίζονται όταν μια εντολή εξαρτάται από το αποτέλεσμα μιας προηγούμενης εντολής που δεν έχει ολοκληρώσει την εκτέλεσή της. Για παράδειγμα, αν μια εντολή φορτώσει δεδομένα από τη μνήμη και μια επόμενη εντολή προσπαθήσει να χρησιμοποιήσει αυτά τα δεδομένα πριν ολοκληρωθεί η φόρτωση, η επόμενη εντολή θα πρέπει να περιμένει. Αυτές οι καθυστερήσεις μπορούν να προκληθούν από την απουσία των δεδομένων που χρειάζονται για να εκτελεστεί η επόμενη εντολή, οδηγώντας σε χαμηλή απόδοση.

3. Κίνδυνοι Ελέγχου

Αυτοί οι κίνδυνοι σχετίζονται με τις εντολές διακλάδωσης, όπου η ροή του προγράμματος μπορεί να αλλάξει ανάλογα με την εκτέλεση μιας προηγούμενης εντολής. Η επιλογή της επόμενης εντολής να εκτελεστεί εξαρτάται από το αποτέλεσμα της διακλάδωσης. Εάν δεν γνωρίζουμε εκ των προτέρων ποια πρέπει να είναι η επόμενη εντολή λόγω της ελέγχου που εξαρτάται από την προηγούμενη, τότε η επιτυχής πρόβλεψη μπορεί να αποτρέψει καθυστερήσεις.

Δομικοί Κίνδυνοι στη Διοχέτευση RISC-V

Στη σχεδίαση επεξεργαστών με χρήση διοχέτευσης, είναι σημαντικό να κατανοήσουμε τους κινδύνους που προκύπτουν από περιορισμούς στους διαθέσιμους πόρους, οι οποίοι μπορούν να επηρεάσουν την εκτέλεση των εντολών. Αυτό αναφέρεται στους **δομικούς κινδύνους**.

Τι είναι οι Δομικοί Κίνδυνοι;

Δομικοί κίνδυνοι συμβαίνουν όταν δύο ή περισσότερες εντολές απαιτούν τη χρήση του ίδιου πόρου ταυτόχρονα. Στην περίπτωση των επεξεργαστών RISC-V, ένας κοινός πόρος είναι η μνήμη. Όταν προσπαθούν να εκτελούν εντολές φόρτωσης ή αποθήκευσης (load/store), μπορεί να προκύψει διένεξη για την πρόσβαση στη μνήμη.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Συγκεκριμένα Σενάρια

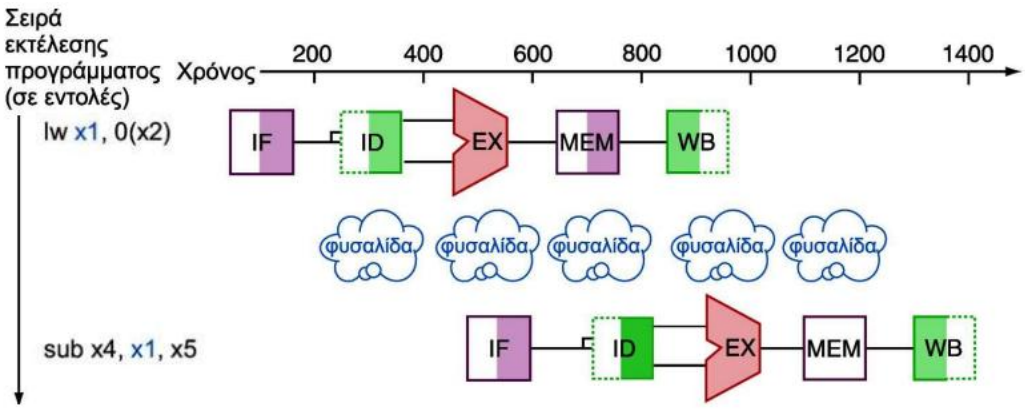
- **Χρήση Μνήμης:**
  - Κάθε εντολή χρειάζεται πρόσβαση σε μνήμη για να φορτώσει ή να αποθηκεύσει δεδομένα. Οι εντολές προσκόμισης (fetching) απαιτούν πρόσβαση στη μνήμη εντολών.
  - Όταν μια εντολή αναμένει να αποκτήσει δεδομένα από τη μνήμη (π.χ., μια εντολή φόρτωσης δεδομένων), αλλά μια άλλη εντολή προσπαθεί επίσης να χρησιμοποιήσει τη μνήμη τη δεδομένη στιγμή (π.χ., για να προσκομίσει κώδικα), αυτό προκαλεί καθυστερήσεις.
- **Καθυστέρηση στον Εκτελώσεις:**
  - Όταν η προσκόμιση εντολής καθυστερεί λόγω της ανάγκης πρόσβασης στη μνήμη, αυτό έχει ως αποτέλεσμα μια "φυσαλίδα" στη διαδρομή δεδομένων. Δηλαδή, οι επόμενες εντολές δεν μπορούν να εκτελούνται στο προγραμματισμένο τους χρονικό πλαίσιο.
  - Αυτές οι καθυστερήσεις έχουν αρνητικό αντίκτυπο στην γενική απόδοση του συστήματος, καθώς θα υπάρξει αυξημένος χρόνος αναμονής.

Λύσεις στους Δομικούς Κινδύνους

Για να αποφευχθούν αυτού του είδους οι δομικοί κίνδυνοι, προτείνεται η χρήση τριών στρατηγικών:

- **Ξεχωριστές Μνήμες:**
  - Δημιουργία ξεχωριστών μνημών για τις εντολές και τα δεδομένα. Αυτό μειώνει τις πιθανότητες να συμβεί διένεξη πόρων, καθώς οι εντολές και τα δεδομένα αποκτούν αποκλειστική πρόσβαση στους αντίστοιχους πόρους.
- **Κρυφές Μνήμες:**
  - Χρήση ξεχωριστών κρυφών μνημών για εντολές και δεδομένα. Αυτό επιτρέπει τα δεδομένα και οι εντολές να προσπελάζονται ταυτόχρονα, καταστώντας τη διαδικασία πιο αποδοτική.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Η εικόνα απεικονίζει έναν **κίνδυνο δεδομένων** σε έναν επεξεργαστή με διοχέτευση (pipeline). Ένας κίνδυνος δεδομένων συμβαίνει όταν μια εντολή εξαρτάται από το αποτέλεσμα μιας προηγούμενης εντολής που βρίσκεται ακόμα σε κάποιο στάδιο της διοχέτευσης. Αυτή η εξάρτηση μπορεί να οδηγήσει τη εξαρτώμενη εντολή να χρησιμοποιήσει λανθασμένα ή παλιά δεδομένα, προκαλώντας εσφαλμένη εκτέλεση του προγράμματος.

1. **Το επάνω μέρος της εικόνας** παρέχει ένα παράδειγμα σε κώδικα που μοιάζει με assembly:

add x19, x0, x1

sub x2, x19, x3

Εδώ, η εντολή sub χρειάζεται το αποτέλεσμα της εντολής add (την τιμή που θα γραφτεί στον καταχωρητή x19). Ωστόσο, όταν η εντολή sub φτάσει στο στάδιο της εκτέλεσης, η εντολή add μπορεί να μην έχει ολοκληρώσει ακόμη το στάδιο της εγγραφής αποτελέσματος (write-back), πράγμα που σημαίνει ότι η ενημερωμένη τιμή του x19 δεν είναι ακόμη διαθέσιμη.

2. **Το κάτω μέρος της εικόνας** δείχνει ένα χρονικό διάγραμμα δύο εντολών σε μια διοχέτευση 5 σταδίων (IF - Ανάκτηση Εντολής, ID - Αποκωδικοποίηση Εντολής, EX - Εκτέλεση, MEM - Πρόσβαση Μνήμης, WB - Εγγραφή Αποτελέσματος):

1. **lw x1, 0(x2):** Αυτή είναι μια εντολή φόρτωσης λέξης. Φορτώνει δεδομένα από τη μνήμη και τα γράφει στον καταχωρητή x1. Το διάγραμμα δείχνει την πρόοδό της μέσα από τα στάδια της διοχέτευσης.

2. **sub x4, x1, x5:** Αυτή είναι μια εντολή αφαίρεσης. Αφαιρεί την τιμή στον καταχωρητή x5 από την τιμή στον καταχωρητή x1 και γράφει το αποτέλεσμα στον καταχωρητή x4.

Παρατηρήστε την εξάρτηση: η εντολή sub χρειάζεται την τιμή που η εντολή lw φορτώνει στον καταχωρητή x1.

Το διάγραμμα δείχνει ότι όταν η εντολή sub φτάνει στο στάδιο **EX (Εκτέλεση)** (περίπου στο χρόνο 600), η εντολή lw βρίσκεται ακόμα στο στάδιο **MEM (Πρόσβαση**

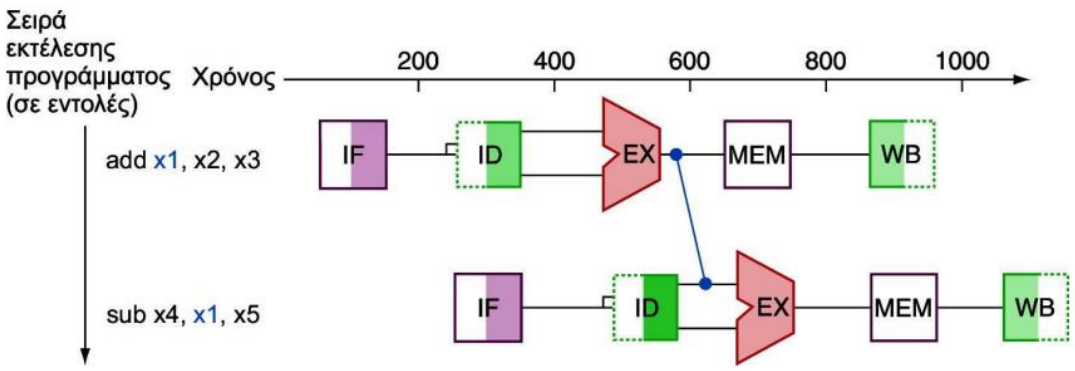
ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Μνήμης). Η τιμή που φορτώνεται από την lw στον x1 δεν είναι ακόμη διαθέσιμη στο αρχείο καταχωρητών.

Οι "(φυσσαλίδα)" που εισάγονται στη ροή της διοχέτευσης για την εντολή sub αντιπροσωπεύουν **αναμονές της διοχέτευσης** ή **φουσκάλες**. Αυτές εισάγονται για να καθυστερήσουν την εντολή sub μέχρι να ολοκληρώσει η εντολή lw το στάδιο **WB (Εγγραφή Αποτελέσματος)** και η σωστή τιμή να είναι διαθέσιμη στον καταχωρητή x1.

Συνοπτικά, η εικόνα καταδεικνύει:

- Μια **εξάρτηση δεδομένων**: Η εντολή sub εξαρτάται από το αποτέλεσμα της εντολής lw.
- Έναν **κίνδυνο διοχέτευσης**: Αυτή η εξάρτηση προκαλεί έναν κίνδυνο δεδομένων επειδή η εξαρτώμενη εντολή προσπαθεί να χρησιμοποιήσει τα δεδομένα πριν αυτά είναι έτοιμα.
- Μια **λύση**: Εισάγονται αναμονές στη διοχέτευση (φουσκάλες) για να επιλυθεί ο κίνδυνος δεδομένων καθυστερώντας την εξαρτώμενη εντολή μέχρι να γίνουν διαθέσιμα τα απαιτούμενα δεδομένα. Αυτό εξασφαλίζει τη σωστή εκτέλεση αλλά μειώνει τη συνολική αποδοτικότητα της διοχέτευσης.



Η εικόνα παρουσιάζει μια βασική τεχνική που χρησιμοποιείται στους σύγχρονους επεξεργαστές με διοχέτευση – την **προώθηση** ή αλλιώς **παράκαμψη**. Πρόκειται για έναν έξυπνο τρόπο να αντιμετωπίζουμε τις **εξαρτήσεις δεδομένων** ανάμεσα σε εντολές που εκτελούνται ταυτόχρονα σε διαφορετικά στάδια της διοχέτευσης.

Ας δούμε πρώτα τι σημαίνει αυτό με ένα παράδειγμα από την εικόνα.

Έχουμε δύο εντολές:

- add x1, x2, x3 – προσθέτει τις τιμές των καταχωρητών x2 και x3 και αποθηκεύει το αποτέλεσμα στον x1.
- sub x4, x1, x5 – αφαιρεί την τιμή του x5 από τον x1 και αποθηκεύει το αποτέλεσμα στον x4.

Παρατηρείς ότι η δεύτερη εντολή χρειάζεται τον **x1**, αλλά αυτός ενημερώνεται μόλις από την πρώτη εντολή. Αν περιμέναμε να περάσει η add απ' όλα τα στάδια, να γράψει το αποτέλεσμα στον x1 (στο στάδιο WB), και **μετά** να διαβάσει η sub, τότε θα χανόταν πολύς χρόνος — η διοχέτευση θα καθυστερούσε.

Γι' αυτό υπάρχει η **προώθηση**.

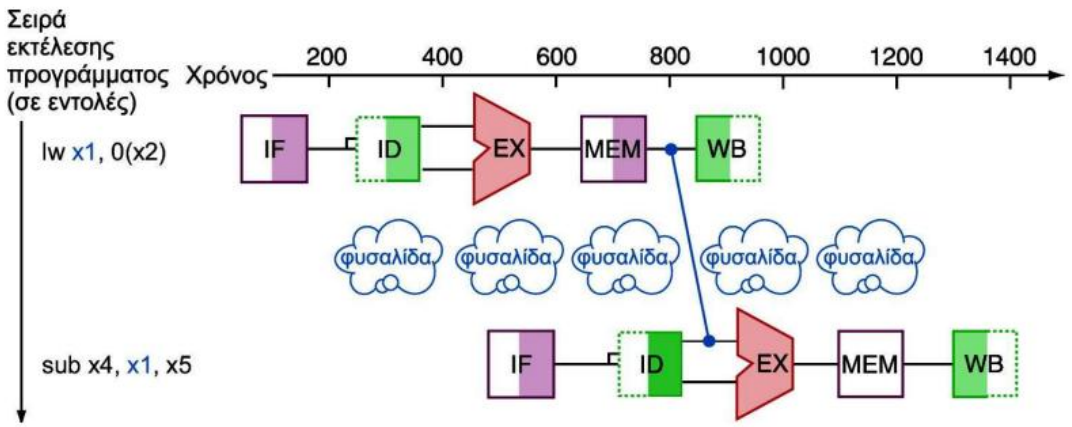
Κοίτα την μπλε γραμμή στο διάγραμμα: δείχνει πώς το αποτέλεσμα της add (που παράγεται στο στάδιο **EX**) **δεν περιμένει να γραφτεί** πρώτα στον καταχωρητή. Αντίθετα, το αποτέλεσμα **προωθείται απευθείας** στη sub, η οποία μπαίνει κι αυτή στο στάδιο EX την ίδια στιγμή.

Με αυτόν τον τρόπο, αποφεύγεται η καθυστέρηση, η sub παίρνει αμέσως τη σωστή τιμή και η ροή στην διοχέτευση παραμένει ομαλή, **χωρίς stalls**.

Για να δουλέψει αυτή η τεχνική, χρειάζονται κάποιες **επιπλέον συνδέσεις** στη διαδρομή των δεδομένων. Είναι αυτές που επιτρέπουν στο αποτέλεσμα από την ALU ή από τη μνήμη να πάει **κατευθείαν** πίσω στην είσοδο της ALU, παρακάμπτοντας τον καταχωρητή. Γι' αυτό λέγεται και "παράκαμψη".

Με λίγα λόγια, η προώθηση είναι σαν να λέμε στον επεξεργαστή: *"Μην περιμένεις να γράψεις το αποτέλεσμα και μετά να το ξαναδιαβάσεις – απλώς δώσ' το αμέσως σε όποιον το χρειάζεται!"*

Έτσι αποφεύγεται η αναμονή, οι εντολές εκτελούνται πιο γρήγορα και η απόδοση του συστήματος βελτιώνεται σημαντικά.



Η εικόνα μάς δείχνει μια χαρακτηριστική περίπτωση όπου, παρότι υπάρχει **μηχανισμός προώθησης (forwarding)**, αυτός **δεν αρκεί** για να αποφευχθούν καθυστερήσεις στη διοχέτευση. Πρόκειται για έναν **κίνδυνο φόρτωσης/χρήσης** (load-use hazard), και συμβαίνει όταν μια εντολή χρειάζεται ένα δεδομένο πριν αυτό να είναι διαθέσιμο.

Πιο συγκεκριμένα, δες τις δύο εντολές που απεικονίζονται:

- Η πρώτη είναι lw x1, 0(x2) – φορτώνει μια τιμή από τη μνήμη και την αποθηκεύει στον καταχωρητή x1.
- Η δεύτερη είναι sub x4, x1, x5 – αφαιρεί την τιμή του x5 από τον x1. Όμως ο x1 μόλις πρόκειται να ενημερωθεί από την lw.

Στην εικόνα βλέπουμε ότι η sub φτάνει στο στάδιο **EX** και είναι έτοιμη να εκτελεστεί. Το πρόβλημα είναι ότι η τιμή του x1 **δεν έχει έρθει ακόμα από τη μνήμη**. Η εντολή lw βρίσκεται στο στάδιο **MEM**, δηλαδή εκεί όπου διαβάζει από τη μνήμη. Τα δεδομένα δεν έχουν φτάσει ούτε καν στο στάδιο WB, και άρα **δεν μπορούν ακόμα να προωθηθούν**.

Κι εδώ είναι το κρίσιμο σημείο: **Η προώθηση δουλεύει μόνο όταν το δεδομένο έχει ήδη παραχθεί**, π.χ. από την ALU σε μια πράξη πρόσθεσης. Όταν όμως το δεδομένο έρχεται από τη μνήμη, όπως σε αυτή την περίπτωση, **απλώς δεν είναι διαθέσιμο εγκαίρως**.

Έτσι, η sub δεν έχει άλλη επιλογή: πρέπει να **περιμένει**. Γι' αυτό και στην εικόνα εισάγονται **φουσκάλες (stalls)** — δηλαδή άδειοι κύκλοι ρολογιού — που καθυστερούν την sub μέχρι να είναι έτοιμο το αποτέλεσμα της lw.

Μόλις η τιμή από τη μνήμη φτάσει στο τέλος του σταδίου **MEM**, μπορεί τελικά να προωθηθεί μέσω της διακεκομμένης μπλε γραμμής προς την ALU της sub, στο στάδιο EX. Αλλά αυτό συμβαίνει **μετά** την καθυστέρηση.

Αυτό το παράδειγμα δείχνει καθαρά ότι:

- Η προώθηση **δεν μπορεί να λύσει όλους τους κινδύνους** δεδομένων.
- Όταν η εξάρτηση αφορά σε δεδομένο που **έρχεται από τη μνήμη**, η διοχέτευση δεν έχει άλλη επιλογή από το να εισάγει αναμονή.
- Ο μηχανισμός forwarding λειτουργεί μόνο **από τη στιγμή που το αποτέλεσμα έχει παραχθεί** — δεν μπορεί να "δει στο μέλλον" ή να επιταχύνει την ανάκτηση από τη μνήμη.

Γι' αυτό λέμε και στην εικόνα: **"Δεν γίνεται προώθηση προς τα πίσω στον χρόνο!"**

Αναδιάταξη Κώδικα για Αποφυγή Καθυστερήσεων

Η αναδιάταξη κώδικα είναι μια τεχνική που χρησιμοποιείται στην προγραμματισμένη εκτέλεση εντολών, ώστε να βελτιωθεί η απόδοση και να αποφευχθούν οι καθυστερήσεις που μπορεί να προκύψουν από εξαρτήσεις μεταξύ εντολών.

Κύριες Ιδέες

- **Κίνδυνος Καθυστέρησης**:
  - Όταν μία εντολή (π.χ. μια εντολή φόρτωσης) εξαρτάται από τα δεδομένα που παρέχονται από μία προηγούμενη εντολή, η εκτέλεση της μπορεί να καθυστερήσει. Αυτό συμβαίνει συχνά σε περιπτώσεις

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

όπου η προηγούμενη εντολή δεν έχει ολοκληρωθεί πριν η επόμενη εντολή προσπαθήσει να χρησιμοποιήσει τα αποτελέσματά της.

- **Παράδειγμα Κώδικα**:
  - Αρχικός κώδικας:
  - lw x1, 0(x0) // Φόρτωση από τη μνήμη στο x1
  - lw x2, 8(x0) // Φόρτωση από τη μνήμη στο x2
  - add x3, x1, x2 // Χρησιμοποιεί το x1 από την προηγούμενη εντολή
  - sw x3, 24(x0) // Αποθήκευση του x3 πίσω στη μνήμη
  - Προβλεπόμενη Καθυστέρηση:
    - Υπάρχει καθυστέρηση, καθώς η τρίτη εντολή (add) περιμένει την ολοκλήρωση της πρώτης εντολής (lw x1). Εάν το x1 δεν είναι έτοιμο, η επόμενη εντολή δεν μπορεί να εκτελεστεί.
- **Αναδιάταξη του Κώδικα**:
  - Οι εντολές μπορούν να αναδιαταχθούν έτσι ώστε να ελαχιστοποιηθούν οι καθυστερήσεις.
  - Θα μπορούσε να γίνει κάτι τέτοιο:
  - lw x1, 0(x0)
  - lw x2, 8(x0)
  - lw x4, 16(x0) // Νέα εντολή προστίθεται εδώ για να γεμίσει το κενό χρόνου
  - add x3, x1, x2
  - sw x3, 24(x0)
  - Στην παραπάνω αναδιάταξη, προσθέσαμε μία εντολή φόρτωσης (lw x4, 16(x0)) μεταξύ των δύο ενεργειών φόρτωσης για να "γεμίσουμε" τον χρόνο αναμονής.

Συμπεράσματα

- Η αναδιάταξη κώδικα βελτιώνει την απόδοση μειώνοντας τις αναμονές κατά την εκτέλεση εντολών.
- Παρέχει μια μέθοδο για την αποφυγή προβλημάτων εξαρτήσεων και καθυστερήσεων στην εκτέλεση προγραμμάτων, βελτιώνοντας συνολικά τη ροή των εντολών.

Είναι σημαντικό για τους προγραμματιστές και τους σχεδιαστές συστημάτων να κατανοήσουν πώς η διάταξη του κώδικα μπορεί να επηρεάσει την εκτελεστική τους

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

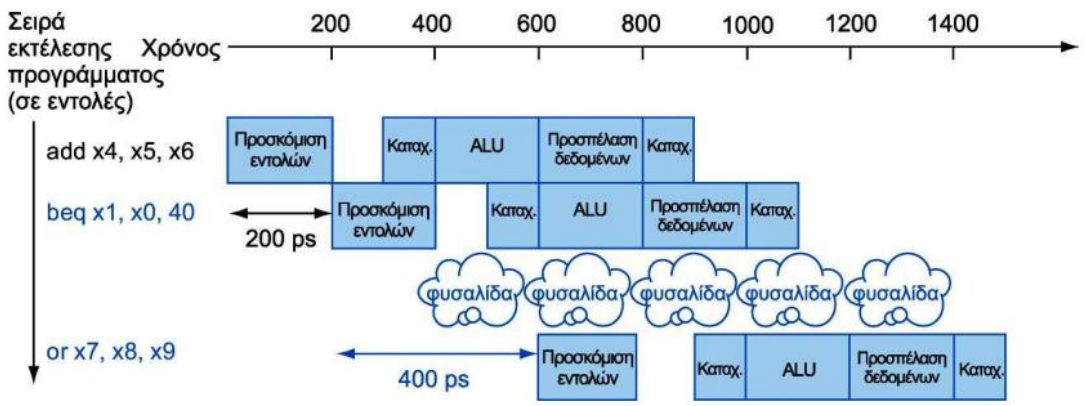
απόδοση, και η αναδιάταξη είναι μία από τις τεχνικές που μπορεί να χρησιμοποιηθεί για την βελτίωση της αποδοτικότητας.

Κίνδυνοι Ελέγχου

Οι κίνδυνοι ελέγχου προκύπτουν κατά την εκτέλεση εντολών που περιλαμβάνουν διακλάδωση, ειδικά σε αρχιτεκτονικές CPU με διοχέτευση, όπως αυτή του RISC-V. Παρακάτω αναλύεται το κείμενο:

- **Ροή Ελέγχου**: Η διακλάδωση (branching) αποφασίζει ποια εντολή θα εκτελεστεί επόμενη, συνεπώς επηρεάζει άμεσα τη ροή του προγράμματος. Όταν υπάρχει διακλάδωση, η επόμενη εντολή που θα πρέπει να εκτελεστεί εξαρτάται από το αποτέλεσμα που θα παραχθεί από τη διακλάδωση.
- **Καθυστέρηση Στην Προσκόμιση Εντολής**: Εάν η CPU δεν γνωρίζει το αποτέλεσμα της διακλάδωσης την κατάλληλη στιγμή, μπορεί να προσκομίσει μια λανθασμένη εντολή. Αυτό συμβαίνει επειδή οι εντολές αναμφίβολα προσκομίζονται σε προεπιλεγμένη ακολουθία, και η σωστή αλλαγή ροής πρέπει να περιμένει να υπολογιστεί το αποτέλεσμα της διακλάδωσης.
- **Στάδιο ID και Συγκρίσεις**: Για να ελαχιστοποιηθεί η πιθανότητα καθυστερήσεων από την διακλάδωση, είναι απαραίτητο να γίνονται οι συγκρίσεις των καταχωρητών και ο υπολογισμός του προορισμού νωρίς κατά τη διάρκεια της διοχέτευσης. Αυτό απαιτεί πρόσθεση επιπλέον υλικού ή λογισμικού για την εκτέλεση των συγκρίσεων νωρίς μέσα στο στάδιο ID (Instruction Decode).
- **Κίνδυνος Προβλέψεων**: Η αποτελεσματική διαχείριση των κινδύνων ελέγχου απαιτεί συχνά στρατηγικές προβλέψεων, οι οποίες επιτρέπουν την πρόβλεψη του αποτελέσματος μιας διακλάδωσης και την πρότερη προσκόμιση της επόμενης εντολής.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



Τι πρόβλημα δείχνει αυτή η εικόνα;
Η εικόνα δείχνει την **καθυστέρηση διακλάδωσης (branch delay)** σε έναν επεξεργαστή με **διοχέτευση (pipeline)** — δηλαδή, τι συμβαίνει όταν ο επεξεργαστής συναντά μια εντολή διακλάδωσης, όπως η beq.

Γιατί δημιουργείται καθυστέρηση στις διακλαδώσεις;
Επειδή ο επεξεργαστής **δεν γνωρίζει αμέσως** αν η διακλάδωση θα γίνει ή όχι. Πρέπει πρώτα να **υπολογίσει τη συνθήκη** (π.χ. αν δύο καταχωρητές είναι ίσοι). Μέχρι τότε, **δεν ξέρει ποια είναι η σωστή επόμενη εντολή** για προσκόμιση.

Τι δείχνει η πάνω πλευρά της εικόνας;
Εξηγεί ακριβώς αυτό:

«Αναμονή ώσπου να προσδιοριστεί το αποτέλεσμα της διακλάδωσης, και προσκόμιση της επόμενης εντολής μετά.»
Δηλαδή, **καθυστέρηση** στη ροή, επειδή ο επεξεργαστής *περιμένει* να βεβαιωθεί για τη διακλάδωση.

Ποια παραδείγματα εντολών βλέπουμε στο κάτω μέρος της εικόνας;

1. add x4, x5, x6 – μια απλή αριθμητική πράξη που εκτελείται κανονικά.
2. beq x1, x0, 40 – η εντολή διακλάδωσης που ελέγχει αν θα αλλάξει η ροή.
3. or x7, x8, x9 – η επόμενη εντολή στην κανονική ροή του προγράμματος.

Τι συμβαίνει χρονικά με αυτές τις εντολές;

- Η beq φτάνει στο στάδιο ALU γύρω στον κύκλο 600, όπου ελέγχεται η συνθήκη.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Η or ξεκινά να προσκομίζεται **πριν ο επεξεργαστής μάθει** αν η beq θα αλλάξει τη ροή.
- Άρα υπάρχει **ρίσκο** να προσκομιστεί η *λάθος* εντολή.

Τι γίνεται αν τελικά η διακλάδωση ΛΑΒΕΤΑΙ;
Η εντολή or, που έχει ήδη αρχίσει, **είναι λάθος** και πρέπει να **ακυρωθεί**, και να προσκομιστεί η σωστή από τη νέα διεύθυνση.

Και αν ΔΕΝ ληφθεί η διακλάδωση;
Η or ήταν τελικά η σωστή εντολή, αλλά ο επεξεργαστής **δεν μπορούσε να το ξέρει εγκαίρως**. Γι' αυτό και **εισάγει φουσκάλες (αναμονές)** μετά την beq — για να μην πάρει ρίσκο.

Ποιο είναι το πρόβλημα συνολικά;
Αυτή η αναμονή **μειώνει την απόδοση** του επεξεργαστή. Οι διακλαδώσεις, ακόμα κι όταν δεν λαμβάνονται, προκαλούν **καθυστέρηση** γιατί "παγώνουν" τη ροή εντολών.

Υπάρχει λύση για αυτήν την καθυστέρηση;
Ναι! Μια κοινή τεχνική είναι η **πρόβλεψη διακλάδωσης (branch prediction)**. Ο επεξεργαστής "μαντεύει" αν θα γίνει η διακλάδωση, προσκομίζει τη σωστή εντολή **προληπτικά**, και αν κάνει λάθος… τότε ακυρώνει και διορθώνει.

Πρόβλεψη Διακλάδωσης

Η πρόβλεψη διακλάδωσης αναφέρεται στην τεχνική που χρησιμοποιείται για να διαχειριστεί τις καθυστερήσεις που προκαλούνται από τις διακλαδώσεις (branch instructions) σε έναν επεξεργαστή με διοχέτευση, όπως η αρχιτεκτονική RISC-V. Ακολουθεί μια πιο αναλυτική περιγραφή αυτής της τεχνικής:

1. **Δυσκολία στην Πρόβλεψη Διακλαδώσεων**

   - **Πρόβλημα Ταχύτητας:** Οι μεγαλύτερες διοχετεύσεις δυσκολεύονται να προσδιορίσουν γρήγορα το αποτέλεσμα μιας διακλάδωσης. Αυτό σημαίνει ότι η επόμενη εντολή δεν μπορεί να προσκομιστεί εγκαίρως, γεγονός που δημιουργεί καθυστερήσεις στην εκτέλεση.
   - **Καθυστέρηση:** Αυτή η καθυστέρηση στη διακλάδωση μπορεί να οδηγήσει σε επιβάρυνση, όπου η ροή των εντολών διακόπτεται, με αποτέλεσμα την απώλεια αποδοτικότητας.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

2. **Πρόβλεψη του Αποτελέσματος**

- **Μηχανισμός Πρόβλεψης:** Η πρόβλεψη των διακλαδώσεων είναι η διαδικασία εκτίμησης του αν μια διακλάδωση θα είναι "ληφθείσα" ή "μη ληφθείσα" πριν να είναι διαθέσιμο το αποτέλεσμα. Αυτή η πρόβλεψη επιτρέπει στον επεξεργαστή να αρχίσει να εκτελεί την επόμενη εντολή χωρίς να περιμένει την ολοκλήρωση της διακλάδωσης.

3. **Συνέπειες Λανθασμένης Πρόβλεψης**

- **Καθυστέρηση:** Αν η πρόβλεψη είναι λανθασμένη, τότε προκαλείται καθυστέρηση, καθώς οι εντολές που εκτελέστηκαν μετά τη διακλάδωση πρέπει να αναιρεθούν (rollback) και να προσκομιστούν οι σωστές εντολές.

4. **Εφαρμογή στην RISC-V**

- **Πρόβλεψη των Μη Ακολουθησών Διακλαδώσεων:** Στη RISC-V έχει υλοποιηθεί ένα σύστημα για να προβλέπονται οι μη ακολουθούμενες διακλαδώσεις, δηλαδή οι διακλαδώσεις που δεν εκτελούνται.
- **Αγνόηση Καθυστερήσεων:** Αν η πρόβλεψη είναι σωστή, τότε οι εντολές μπορούν να συνεχίζουν χωρίς καθυστέρηση, επιτρέποντας την έγκαιρη εκτέλεση και αυξάνοντας την απόδοση της CPU.

Πρόβλεψη Διακλάδωσης

Η πρόβλεψη διακλάδωσης είναι μια τεχνική που χρησιμοποιείται στους επεξεργαστές για να βελτιώσει την απόδοση της εκτέλεσης εντολών. Καθώς οι εντολές διακλάδωσης μπορούν να αλλάξουν την ροή εκτέλεσης του προγράμματος, η σωστή πρόβλεψη της κατεύθυνσης αυτών των διακλαδώσεων μπορεί να μειώσει τις καθυστερήσεις που προκύπτουν από την αναμονή της εκτέλεσης των επόμενων εντολών.

Η πρόβλεψη διακλάδωσης χωρίζεται σε δύο βασικές κατηγορίες:

<u>Στατική Πρόβλεψη Διακλάδωσης</u>

- **Συμπεριφορά:** Αυτή η μέθοδος βασίζεται στη συνήθη συμπεριφορά μιας διακλάδωσης, όπως οι βρόχοι και οι εντολές if.
- **Πρόβλεψη:**
    - **Ληφθείσα Διακλάδωση:** Προβλέπεται ότι θα ληφθεί μια διακλάδωση, ειδικά στις περιπτώσεις βρόχων.
    - **Μη Ληφθείσα Διακλάδωση:** Προβλέπεται ότι η διακλάδωση δεν θα ληφθεί, συνήθως για άλλες εντολές.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Δυναμική Πρόβλεψη Διακλάδωσης

- **Μετρώντας Συμπεριφορά:** Αυτή η μέθοδος παρακολουθεί τη πραγματική συμπεριφορά των διακλαδώσεων μέσω υλικού, όπως με την καταγραφή πρόσφατων ιστορικών διακλάδωσης.
- **Υπόθεση Συνεχιζόμενης Τάσης:** Υποθέτει ότι η συμπεριφορά που παρατηρήθηκε στην πρόσφατη ιστορία θα συνεχιστεί στο άμεσο μέλλον.
- **Διαχείριση Σφαλμάτων:** Σε περίπτωση λάθους, απαιτείται καθυστέρηση για να εκ νέου προσκομισθούν οι σωστές εντολές και ενημέρωση του ιστορικού.

Αυτές οι τεχνικές επιτρέπουν στους επεξεργαστές να εκτελούν αποτελεσματικά πολλές εντολές ταυτόχρονα, μειώνοντας τον χρόνο εκτέλεσης και βελτιώνοντας την **απόδοση**

Η ΓΕΝΙΚΗ ΕΙΚΟΝΑ-ΣΥΜΠΕΡΑΣΜΑΤΑ ΓΙΑ ΤΗΝ ΔΙΟΧΕΤΕΥΣΗ

Εξήγηση της Διοχέτευσης

Η διοχέτευση (Pipelining) είναι ένα σημαντικό σχεδιαστικό χαρακτηριστικό που χρησιμοποιείται στους επεξεργαστές για να αυξήσει την απόδοση και την αποδοτικότητα των εντολών. Ας αναλύσουμε πώς λειτουργεί και ποια είναι τα οφέλη και οι προκλήσεις της.

1. Βελτίωση Απόδοσης

Η διοχέτευση "βελτιώνει την απόδοση αυξάνοντας τη διεκπεραιωτική ικανότητα των εντολών". Αυτό σημαίνει ότι ο επεξεργαστής μπορεί να εκτελεί περισσότερες εντολές σε ένα δεδομένο χρονικό διάστημα, γεγονός που οδηγεί σε αυξημένες επιδόσεις.

2. Ταυτόχρονη Εκτέλεση Εντολών

Μία από τις βασικές ιδιότητες της διοχέτευσης είναι ότι "εκτελεί περισσότερες από μία εντολές ταυτόχρονα". Αυτό επιτυγχάνεται σπάζοντας τη διαδικασία εκτέλεσης των εντολών σε βήματα ή στάδια. Κάθε βήμα μπορεί να εκτελείται ανεξάρτητα, επιτρέποντας στον επεξεργαστή να κινείται από την εκτέλεση μιας εντολής στην επόμενη χωρίς να περιμένει να ολοκληρωθεί η προηγούμενη.

3. Λανθάνων Χρόνος

"Κάθε εντολή έχει τον ίδιο λανθάνοντα χρόνο", πράγμα που σημαίνει ότι όλα τα στάδια της εκτέλεσης έχουν σχεδιαστεί να διαρκούν τον ίδιο χρόνο. Αυτό βοηθά στη διατήρηση του ρυθμού που οι εντολές εκτελούνται και βελτιώνει τη συνολική αποδοτικότητα του επεξεργαστή.

4. Κίνδυνοι

Ωστόσο, η διοχέτευση "υπόκειται σε κινδύνους", οι οποίοι περιλαμβάνουν:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **Δομικούς κινδύνους**: Αυτοί συμβαίνουν όταν δύο τμήματα (ή στάδια) της διαδικασίας εκτέλεσης χρειάζονται πρόσβαση στον ίδιο πόρο (π.χ. μνήμη).
- **Κίνδυνοι δεδομένων**: Αυτοί συμβαίνουν όταν μια εντολή εξαρτάται από τα αποτελέσματα μιας προηγούμενης εντολής που δεν έχει ολοκληρωθεί ακόμα.
- **Κίνδυνοι ελέγχου**: Αυτοί συμβαίνουν όταν οι διακλαδώσεις (branches) μπορούν να επηρεάσουν τη ροή των εντολών και να απαιτούν μια καθυστέρηση μέχρι να επιβεβαιωθεί το αποτέλεσμα της διακλάδωσης.

5. Σχεδίαση Συνόλου Εντολών

Η σχεδίαση του συνόλου εντολών (ISA) επηρεάζει την πολυπλοκότητα της υλοποίησης της διοχέτευσης. Μοναδικές και ποικιλόμορφες εντολές με διαφορετικά μήκη και απαιτήσεις μπορούν να καταστήσουν πιο περίπλοκο το σχεδιασμό μιας αποτελεσματικής διοχέτευσης.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

ΠΑΡΑΡΤΗΜΑ

COMPUTER ARCHITECTURE 1

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

A1.Τύποι Δεδομένων στον Επεξεργαστή RISC-V

Ο επεξεργαστής RISC-V υποστηρίζει διάφορους τύπους δεδομένων, που είναι βασικοί για την αποθήκευση και επεξεργασία πληροφοριών. Οι κύριοι τύποι δεδομένων περιλαμβάνουν τους ακεραίους, τους αριθμούς κινητής υποδιαστολής και τους χαρακτήρες.

Επεξήγηση Τύπων Δεδομένων

- **Byte (8-bit ακέραιος)**: Μπορεί να αποθηκεύσει ακέραιους αριθμούς από -128 έως 127. Είναι ο πιο μικρός τύπος δεδομένων που χρησιμοποιείται συνήθως για την αποθήκευση χαρακτήρων και άλλων μικρών δεδομένων.
- **Halfword (16-bit ακέραιος)**: Έχει ικανότητα αποθήκευσης μεγαλύτερων ακέραιων αριθμών, με εύρος από -32,768 έως 32,767. Συχνά χρησιμοποιείται για την αποθήκευση μικρότερων αριθμών ή δεδομένων.
- **Word (32-bit ακέραιος)**: Ένας από τους πιο κοινώς χρησιμοποιούμενους τύπους δεδομένων, με εύρος από −231−231 έως 231−1231−1. Είναι ταυτόχρονα ο πιο συνηθισμένος τύπος για την αποθήκευση αριθμών και pointers.
- **Float (32-bit αριθμός κινητής υποδιαστολής)**: Χρησιμοποιείται για αριθμούς με δεκαδικά ψηφία. Μπορεί να αποθηκεύσει επαρκή αριθμό για πολλές εφαρμογές που απαιτούν αριθμητικά αποτελέσματα με μεγαλύτερη ακρίβεια.
- **Double (64-bit αριθμός κινητής υποδιαστολής)**: Παρέχει δύο φορές περισσότερη ακρίβεια συγκριτικά με το float, επιτρέποντας ακόμα μεγαλύτερους αριθμούς και μικρότερες σφαλ(mathematical) στο καταμέτρηση.

Συμπερασματικά

Ο τύπος **"halfword"** αναφέρεται σε μια διαμόρφωση 16-bit, που σημαίνει ότι είναι "μισό" από το μέγεθος μιας λέξης (word), που είναι 32-bit. Αυτό τονίζει την ικανότητα αποθήκευσης δεδομένων σε διαφορετικά επίπεδα ακρίβειας και εύρους, ανάλογα με τις απαιτήσεις του προγράμματος ή της εφαρμογής.

Αυτοί οι τύποι δεδομένων είναι κρίσιμοι για την αποτελεσματική λειτουργία του επεξεργαστή RISC-V και την απόδοση διαφόρων υπολογιστικών εργασιών.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

🔹 Α΄.2 Οργάνωση της Μνήμης

✅ 1. Δομή Μνήμης

- Η μνήμη (memory) μπορεί να αναπαρασταθεί ως **σειρά από bytes**.
- **Κάθε byte** έχει τη **δική του διεύθυνση** (byte addressable memory).
- Κάθε διεύθυνση μνήμης αποθηκεύει **1 byte** πληροφορίας.

✅ 2. Αποθήκευση Λέξης (Word)

- Μια λέξη (word) έχει **32 bits**, δηλαδή **4 bytes**.
- Για την αποθήκευσή της απαιτούνται **4 συνεχόμενες διευθύνσεις** μνήμης.

✅ 3. Σειρά Αποθήκευσης Bytes (Endianness)

Όταν αποθηκεύεται μια ποσότητα δεδομένων με πολλά bytes, καθορίζεται η σειρά των bytes:

🔸 **Big-endian**

- Το **περισσότερο σημαντικό byte (Most Significant Byte – MSB)** αποθηκεύεται στη **μικρότερη διεύθυνση μνήμης**.

🔸 **Little-endian**

- Το **λιγότερο σημαντικό byte (Least Significant Byte – LSB)** αποθηκεύεται στη **μικρότερη διεύθυνση μνήμης**.

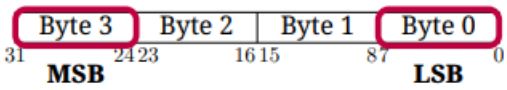
✅ 4. Τρόπος Οργάνωσης του RISC-V

- Ο επεξεργαστής **RISC-V** ακολουθεί την **οργάνωση μνήμης τύπου little-endian**.

✅ 5. Διάταξη Bytes σε Λέξη 32 bits

- Σε μια λέξη των 32 bits:
  - Το **περισσότερο σημαντικό byte** είναι στα **αριστερά**.
  - Το **λιγότερο σημαντικό byte** είναι στα **δεξιά**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ



✅ 6. Παράδειγμα Αποθήκευσης Λέξεων

Αποθηκεύουμε τις λέξεις (32-bit η κάθε μία):

1. 0x11223344
2. 0x5566
3. 58
4. 3.500.000

Μετατροπές:

- 0x5566 → 0x00005566
- 58_{10} → 0x0000003A
- 3500000_{10} → 0x003567E0

✅ 7. Εικόνα 26 – Αποθήκευση ανά Endianness

(α') Little-Endian – Οι τιμές στη μνήμη:

Διεύθυνση	**Περιεχόμενο**
0x10010000	44 33 22 11
0x10010004	66 55 00 00
0x10010008	3A 00 00 00
0x1001000C	E0 67 35 00

(β') Big-Endian – Οι τιμές στη μνήμη:

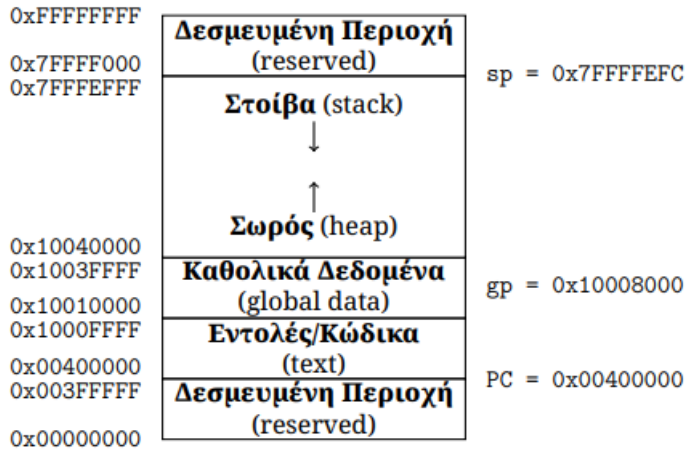
Διεύθυνση	**Περιεχόμενο**
0x10010000	11 22 33 44
0x10010004	00 00 55 66

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Διεύθυνση	Περιεχόμενο
0x10010008	00 00 00 3A
0x1001000C	00 35 67 E0

Α΄.3 Διάταξη Μνήμης (Memory Layout)

Η **μνήμη ενός προγράμματος** στον επεξεργαστή **RISC-V** χωρίζεται σε συγκεκριμένες περιοχές, καθεμία με διαφορετικό σκοπό. Η διάταξη αυτή φαίνεται στην **Εικόνα 27** (Memory Map).



✅ 1. Βασικές Περιοχές της Μνήμης

Από την **ψηλότερη προς τη χαμηλότερη διεύθυνση**:

Περιοχή	Σκοπός
🔒 **Δεσμευμένη περιοχή (reserved)**	Χρησιμοποιείται από το σύστημα – **όχι προσβάσιμη** από προγράμματα χρήστη.
📚 **Στοίβα (stack)**	Αποθηκεύει **τοπικές μεταβλητές συναρτήσεων** και πληροφορίες κλήσεων. Ξεκινά από την **υψηλή μνήμη** και **μεγαλώνει προς τα κάτω**.
📦 **Σωρός (heap)**	Περιέχει **δυναμικά δεσμευμένα δεδομένα** (π.χ. με malloc). **Μεγαλώνει προς τα πάνω**.

Περιοχή	Σκοπός
🌐 **Καθολικά δεδομένα (global data)**	Περιέχει μεταβλητές που είναι **ορατές σε όλο το πρόγραμμα**. Ορίζονται πριν ξεκινήσει η εκτέλεση.
⚙️ **Εντολές / Κώδικας (text)**	Περιέχει τον **κώδικα του προγράμματος** (σε μορφή μηχανής). Συνήθως είναι μόνο-ανάγνωσης.

✅ 2. Αντιστοιχία Περιοχών σε Διευθύνσεις (Ενδεικτικά)

Περιοχή	Διευθύνσεις
Reserved (πάνω)	0xFFFFFFFF ↓
Stack	0x7FFFF000 ↓
Heap	0x10040000 ↑
Global Data	0x10010000 – 0x1003FFFF
Text (code)	0x00400000 – 0x1000FFFF
Reserved (κάτω)	0x00000000

✅ 3. Σημαντικοί Καταχωρητές (Registers)

Καταχωρητής	Ρόλος	Τιμή
sp (stack pointer)	Δείχνει την **τρέχουσα κορυφή της στοίβας**	0x7FFFFEFC
gp (global pointer)	Δείχνει **μέση των καθολικών δεδομένων**	0x10008000
PC (program counter)	Δείχνει την **τρέχουσα εντολή** του προγράμματος	0x00400000

✅ 4. Χρήση του Global Pointer (gp)

- Ο **gp παραμένει σταθερός** κατά την εκτέλεση.
- Οι **καθολικές μεταβλητές προσπελαύνονται** μέσω **offsets (π.χ. +8, –12)** από τη θέση του gp.
- Αυτό βελτιώνει την **αποδοτικότητα πρόσβασης** στα δεδομένα.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ **5. Συμπεριφορά της Στοίβας**

- Η στοίβα ξεκινά από **ψηλές διευθύνσεις** και **μεγαλώνει προς χαμηλότερες**.
- Κάθε νέα κλήση συνάρτησης ή τοπική μεταβλητή, προσθέτει νέα δεδομένα στην στοίβα.
- Όταν τελειώσει μια συνάρτηση, τα δεδομένα **αφαιρούνται (pop)** από τη στοίβα.

🧠 **Γενικές Παρατηρήσεις**

- Η περιοχή **text** μπορεί να φιλοξενήσει μέχρι **252 MB κώδικα**.
- Η περιοχή **global data** μπορεί να αποθηκεύσει έως **192 KB καθολικών μεταβλητών**.
- Η χρήση αυτής της **διάταξης βοηθά στον σαφή διαχωρισμό** μεταξύ εντολών, στατικών και δυναμικών δεδομένων, κάτι που διευκολύνει τόσο την αποδοτική λειτουργία όσο και την ασφάλεια.

🔹 Α΄.4 Οι Καταχωρητές του Επεξεργαστή RISC-V

Οι **καταχωρητές (registers)** είναι **θέσεις αποθήκευσης** μέσα στον **πυρήνα του επεξεργαστή** — πιο γρήγορες από τη μνήμη. Ο προγραμματιστής τους **βλέπει και χειρίζεται απευθείας**, μέσω των εντολών της Assembly.

✅ **1. Τι είναι καταχωρητής;**

- Είναι μια **γρήγορη μονάδα αποθήκευσης** ενσωματωμένη στον επεξεργαστή.
- Κάθε καταχωρητής έχει **συγκεκριμένο όνομα** (π.χ. x0, x1, ..., x31).
- Οι **αριθμητικές και λογικές πράξεις** σε RISC-V γίνονται **μόνο μεταξύ καταχωρητών – ποτέ απευθείας με τη μνήμη**.

✅ **2. Είδη Καταχωρητών σε RISC-V**

🔸 **Ακέραιοι Καταχωρητές (Integer Registers)**

- **Πλήθος**: 32 καταχωρητές → x0 έως x31
- **Μέγεθος**:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- **32-bit** στην έκδοση **RV32I**
- **64-bit** στην έκδοση **RV64I**

🔸 **Καταχωρητές Κινητής Υποδιαστολής (Floating-point Registers)**

- **Πλήθος**: 32 → f0 έως f31
- **Μέγεθος**: 32-bit ή 64-bit, ανάλογα με την επέκταση:
  - F: single precision (32-bit)
  - D: double precision (64-bit)
  - Q: quad precision (128-bit)

✅ Αυτοί υπάρχουν **μόνο αν υποστηρίζεται η επέκταση αριθμών κινητής υποδιαστολής** (π.χ., RV32IF, RV32ID, κ.λπ.).

✅ **3. Ειδικοί Καταχωρητές**

Κάποιοι από τους x καταχωρητές έχουν **προκαθορισμένο ρόλο**:

Όνομα	Αριθμός	Ρόλος
sp	x2	**Stack Pointer** – δείχνει την κορυφή της στοίβας
gp	x3	**Global Pointer** – δείχνει στα καθολικά δεδομένα
ra	x1	**Return Address** – αποθηκεύει τη διεύθυνση επιστροφής μετά από jal
x0	zero	Πάντα **μηδέν** – δεν μπορεί να αλλάξει

✅ **4. Μετρητής Προγράμματος (Program Counter - pc)**

- Ο **pc** είναι ειδικός καταχωρητής, όχι x ή f.
- Περιέχει τη **διεύθυνση της επόμενης εντολής** που θα εκτελεστεί.
- **Ενημερώνεται αυτόματα** από τον επεξεργαστή μετά από κάθε εντολή.
- **Δεν μπορεί να τροποποιηθεί απευθείας** από τον προγραμματιστή.

📌 **Σύνοψη**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Κατηγορία	Πλήθος	Μέγεθος	Παραδείγματα
Integer Registers	32	32 ή 64 bit	x0–x31
Floating-point Registers	32	32, 64, 128 bit	f0–f31
Special Purpose	3+	32 ή 64 bit	sp, gp, pc

🔹 **Πίνακας 8: Καταχωρητές του RISC-V**

✅ **Α. Ακέραιοι Καταχωρητές (Integer Registers)**

Καταχωρητής	Όνομα	Ρόλος / Περιγραφή
x0	zero	Πάντα έχει την τιμή 0 (δεν αλλάζει ποτέ)
x1	ra	Return Address – Διεύθυνση επιστροφής από υπορουτίνα
x2	sp	Stack Pointer – Δείχνει την κορυφή της στοίβας
x3	gp	Global Pointer – Δείχνει στη μέση των καθολικών δεδομένων
x4	tp	Thread Pointer – Χρήσιμο σε πολυνηματικά προγράμματα
x5–x7	t0–t2	Προσωρινοί καταχωρητές (temporary)
x8	s0 / fp	Saved Register / Frame Pointer – αποθήκευση πλαισίου
x9	s1	Saved Register – διατηρεί τιμές μεταξύ συναρτήσεων
x10–x11	a0–a1	Ορίσματα συναρτήσεων / τιμές επιστροφής
x12–x17	a2–a7	Ορίσματα συναρτήσεων
x18–x27	s2–s11	Saved Registers – αποθηκευμένοι για χρήση σε κλήσεις
x28–x31	t3–t6	Temporary – επιπλέον προσωρινοί καταχωρητές

🔸 Όλοι αυτοί είναι **καταχωρητές γενικού σκοπού**, εκτός από κάποιους με ειδική συμβατική χρήση (π.χ. sp, ra, fp).

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

✅ Β. Ειδικός Καταχωρητής

Καταχωρητής	Περιγραφή
pc	**Program Counter** – Περιέχει τη διεύθυνση της επόμενης προς εκτέλεση εντολής. Δεν είναι προσπελάσιμος απευθείας από προγράμματα.

✅ Γ. Καταχωρητές Κινητής Υποδιαστολής (Floating-Point Registers)

(υπάρχουν μόνο αν η αρχιτεκτονική περιλαμβάνει υποστήριξη για float: π.χ. RV32IF, RV64IF, κ.λπ.)

Καταχωρητής	Όνομα	Ρόλος / Περιγραφή
f0–f7	ft0–ft7	Temporary FP – προσωρινοί καταχωρητές ΚΥ
f8–f9	fs0–fs1	Saved FP – αποθηκευμένοι καταχωρητές ΚΥ
f10–f11	fa0–fa1	Floating-point Arguments / Return values
f12–f17	fa2–fa7	Floating-point Arguments
f18–f27	fs2–fs11	Floating-point Saved Registers
f28–f31	ft8–ft11	Floating-point Temporaries

📌 Χρήσιμες Παρατηρήσεις

- Οι καταχωρητές s* **διατηρούν την τιμή τους μεταξύ κλήσεων συναρτήσεων**, ενώ οι t* **χάνονται**.
- Οι a* χρησιμοποιούνται για **πέρασμα παραμέτρων και επιστροφή τιμών**.
- Οι f* καταχωρητές χρησιμοποιούνται αντίστοιχα για αριθμούς κινητής υποδιαστολής.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Α΄.5 Μορφή και Βασικά Χαρακτηριστικά Προγραμμάτων (Assembly RISC-V)

Τα προγράμματα στη γλώσσα **Assembly του RISC-V** πρέπει να ακολουθούν συγκεκριμένη **δομή και μορφοποίηση** για να εκτελούνται σωστά από τον συμβολομεταφραστή (assembler).

✅ Α. Δομή Αρχείου Assembly

Ένα πρόγραμμα assembly **χωρίζεται σε δύο βασικά τμήματα**:

1. **Τμήμα Δεδομένων** (.data) – Δηλώσεις μεταβλητών και σταθερών.
2. **Τμήμα Κειμένου/Κώδικα** (.text) – Ο κύριος κώδικας με τις εντολές.

✅ Α΄.5.1 Σχόλια

- Ο χαρακτήρας **#** δηλώνει ότι η υπόλοιπη γραμμή είναι **σχόλιο**.
- Οι **κενές γραμμές** επιτρέπονται χωρίς ειδικό σύμβολο.
- 🔎 Σημείωση: Ο συμβολισμός **#** χρησιμοποιείται στον **προσομοιωτή RARS**.

✅ Α΄.5.2 Οδηγίες προς τον Συμβολομεταφραστή (Directives)

Οι **οδηγίες**:

- Ξεκινούν με τελεία (.)
- **Δεν είναι εκτελέσιμες εντολές**, αλλά **πληροφορίες για τη μετάφραση**
- Ορίζουν πού ξεκινά/τελειώνει κάθε τμήμα, ποιες ετικέτες είναι καθολικές, κλπ.

📘 Πίνακας 9: Βασικές Οδηγίες

Οδηγία	Περιγραφή
.data	Δηλώσεις μεταβλητών – τμήμα δεδομένων
.text	Εντολές προγράμματος – τμήμα εντολών
.globl Label	Δηλώνει μια ετικέτα ως καθολική, π.χ. main

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Οδηγία	Περιγραφή
.align N	Ευθυγράμμιση δεδομένων (0=byte, 1=halfword, 2=word, 3=doubleword)
.eqv A B	Δημιουργεί **μακροεντολή**, αντικαθιστά το B με το A (όπως #define)

✅ Α΄.5.3 Δηλώσεις Δεδομένων

- Τοποθετούνται **κάτω από την οδηγία .data**
- Αποτελούνται από:
  - **Όνομα μεταβλητής** (ή ετικέτα) – πρέπει να **τελειώνει με :**
  - **Τύπος δεδομένων** (π.χ., .word, .byte)
  - **Τιμή**

📌 **Γενική μορφή:**

όνομα_μεταβλητής: .τύπος_δεδομένου τιμή

📘 **Πίνακας 10: Τύποι Δεδομένων**

Οδηγία / Τύπος	Περιγραφή
.byte	Μεταβλητή 8-bit
.half	Μεταβλητή 16-bit
.word	Μεταβλητή 32-bit
.float	Αριθμός κινητής υποδιαστολής 32-bit
.double	Αριθμός κινητής υποδιαστολής 64-bit
.ascii	Ακολουθία ASCII χαρακτήρων (χωρίς \0)
.asciz	Ακολουθία ASCII χαρακτήρων με τερματισμό \0
.space n	Δέσμευση n bytes χωρίς αρχικοποίηση

🧠 **Συνοπτικά:**

- Τα προγράμματα RISC-V πρέπει να **διαχωρίζουν εντολές από δεδομένα**.

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Τα **σχόλια** βοηθούν στη **τεκμηρίωση** και αγνοούνται από τον μεταφραστή.
- Οι **οδηγίες (.data, .text, .globl, κ.λπ.)** είναι απαραίτητες για τη σωστή μετάφραση.
- Οι **μεταβλητές** δηλώνονται με **συγκεκριμένη μορφή** και τύπους δεδομένων.

🔹 Α΄.5 Μορφή και Βασικά Χαρακτηριστικά Προγραμμάτων Assembly (RISC-V)

✅ Γενικά Χαρακτηριστικά

Τα προγράμματα σε **γλώσσα Assembly RISC-V** χωρίζονται σε δύο βασικά μέρη:

1. **Τμήμα Δεδομένων (.data)**
   Περιέχει σταθερές και μεταβλητές.
2. **Τμήμα Κειμένου / Κώδικα (.text)**
   Περιέχει τις εντολές (εκτελέσιμος κώδικας).

✅ Α΄.5.1 Σχόλια

- Τα σχόλια ξεκινούν με τον χαρακτήρα **#** (όχι // όπως σε άλλα συστήματα).
- Τα πάντα μετά από το # αγνοούνται από τον συμβολομεταφραστή.
- Κενές γραμμές επιτρέπονται χωρίς πρόβλημα.

✅ Α΄.5.2 Οδηγίες προς τον Συμβολομεταφραστή (Directives)

Οι **οδηγίες**:

- Ξεκινούν με τελεία (.).
- ΔΕΝ είναι εντολές, αλλά **πληροφορίες για τον assembler**.
- Ελέγχουν την **δομή και ερμηνεία του προγράμματος**.

📘 Πίνακας 9 – Κύριες Οδηγίες

Οδηγία	Περιγραφή
.data	Ξεκινά το τμήμα δεδομένων
.text	Ξεκινά το τμήμα εντολών

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Οδηγία	Περιγραφή
.globl Label	Δηλώνει την Label ως **καθολική**, π.χ. main
.align N	Ευθυγράμμιση στη μνήμη (0=byte, 1=half, 2=word, 3=doubleword)
.eqv A B	Ορίζει σύμβολο – όπως το #define

✅ Α΄.5.3 Δηλώσεις Δεδομένων

Οι μεταβλητές δηλώνονται **κάτω από το .data**, με τη μορφή:

<όνομα_μεταβλητής>: .<τύπος_δεδομένων> <τιμή>

📘 Πίνακας 10 – Τύποι Δεδομένων

Τύπος Οδηγίας	Περιγραφή
.byte	8-bit ακέραιος
.half	16-bit ακέραιος
.word	32-bit ακέραιος
.ascii	ASCII συμβολοσειρά (χωρίς \0)
.asciz	ASCII συμβολοσειρά (με \0)
.float	32-bit κινητής υποδιαστολής
.double	64-bit κινητής υποδιαστολής
.space n	Δέσμευση n bytes χωρίς αρχικοποίηση

✅ Α΄.5.3.1 Δήλωση Ακέραιων

Χρησιμοποιούνται οι .word, .half, και .byte.

- **Αρνητικοί αριθμοί** αποθηκεύονται με **συμπλήρωμα ως προς δύο**.

📌 Παράδειγμα – Δηλώσεις 32-bit λέξεων

```
var1w: .word 350000      # Θετικός αριθμός
var2w: .word 0x3f45a     # Δεκαεξαδικός
var3w: .word -434        # Αρνητικός αριθμός
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

📌 **Παράδειγμα – Δηλώσεις 8-bit bytes**

var1b: .byte 19 # Θετικός αριθμός

var2b: .byte -53 # Αρνητικός αριθμός

🧠 **Ανακεφαλαίωση**

Στοιχείο	**Περιγραφή**
#	Σχόλια
.data / .text	Δομή αρχείου
.globl main	Δήλωση κύριας ρουτίνας
.word, .byte, κ.λπ.	Δηλώσεις μεταβλητών
<όνομα>: .τύπος τιμή	Μορφή δήλωσης

Α΄.5 Μορφή και Χαρακτηριστικά Προγραμμάτων Assembly RISC-V (συνέχεια)

✅ **Δηλώσεις Συμβολοσειρών**

- Οι **συμβολοσειρές** είναι ακολουθίες χαρακτήρων τύπου byte.
- Συνήθως **τερματίζονται με NULL (0x00)**, σύμφωνα με τη C.
- Ορίζονται με:
  - .ascii → χωρίς τερματισμό NULL
  - .asciz → με αυτόματο τερματισμό NULL

📌 **Παράδειγμα**:

msg: .asciz "Hello World\n"

- Οι χαρακτήρες \n (νέα γραμμή) και \t (tab) επιτρέπονται.
- Η κωδικοποίηση είναι **ASCII**.

✅ **Δηλώσεις Αριθμών Κινητής Υποδιαστολής**

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

- Χρησιμοποιούνται για **πραγματικούς αριθμούς** (float/double).
- Οδηγίες:
  - .float → 32-bit (single precision)
  - .double → 64-bit (double precision)
  - .word → χειροκίνητη αναπαράσταση σε IEEE-754

📌 **Παράδειγμα**:

```
pi: .float 3.1415
dfp: .double 9.257841
f:  .word 0x3fc00000   # 1.5 σε IEEE-754
```

✅ **Ο Κώδικας (.text)**

- Ο **εκτελέσιμος κώδικας** αρχίζει μετά από την οδηγία .text.
- Η **κύρια διαδικασία** του προγράμματος δηλώνεται με:

```
.globl main
main:
```

✅ **Ετικέτες (Labels)**

- Ονόματα που αντιστοιχούν σε θέσεις στη μνήμη (π.χ. συναρτήσεις, μεταβλητές).
- Κανόνες:
  - Ξεκινά με γράμμα ή _
  - Μπορεί να περιλαμβάνει γράμματα, αριθμούς, _
  - Πρέπει να **τελειώνει με :**
  - Δεν μπορεί να επαναχρησιμοποιηθεί (μοναδική)

📌 **Παραδείγματα ετικετών**:

```
main:
```

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

Loop:

_BAR:

var12F:

⚠️ Οι ετικέτες είναι **case-sensitive** → Loop: ≠ loop:

✅ Πρότυπο Προγράμματος (Template)

Ένα βασικό πρότυπο για όλα τα προγράμματα Assembly:

```
# Name and general description of the program

# ---------------------- text segment ---------------------#

.text # Program code goes in this section
.globl __start

__start : # Execution starts here


# Your program code goes here...


li a7 , 10 # Done , terminate program
ecall # au revoir...


# ---------------------- data segment ---------------------#

.data # Data declarations go in this section


# ----------------------- End of File ---------------------#
```

📌 Το πρότυπο βρίσκεται και στο eClass με όνομα template.s
Προτείνεται να χρησιμοποιείται για όλες τις ασκήσεις.

🔹 Παράρτημα Β' – Κλήσεις Συστήματος στο RISC-V (RARS)

✅ Τι είναι οι Κλήσεις Συστήματος;

Οι **system calls** επιτρέπουν σε ένα πρόγραμμα να ζητήσει **υπηρεσίες από το λειτουργικό σύστημα**, όπως:

- **Είσοδος/Εξοδος**
- **Δέσμευση μνήμης**
- **Τερματισμός προγράμματος**

➡️ Στο RARS, οι κλήσεις γίνονται με την εντολή:

ecall

✅ **Πώς χρησιμοποιούνται;**

1. Φόρτωσε **τον κωδικό της λειτουργίας** στον **a7**
2. Αν χρειάζεται, βάλε **τιμές/ορίσματα** στους καταχωρητές a0–a6 (ή fa0 για αριθμούς κινητής υποδιαστολής)
3. Κάνε ecall

📘 **Βασικές Κλήσεις Συστήματος (Πίνακας 12)**

Περιγραφή	Κωδικός (a7)	Είσοδος	Έξοδος
Εκτύπωση ακεραίου	1	a0 = αριθμός	-
Εκτύπωση float	2	fa0 = αριθμός	-
Εκτύπωση double	3	fa0 = αριθμός	-
Εκτύπωση συμβολοσειράς	4	a0 = διεύθυνση	-
Ανάγνωση ακεραίου	5	-	a0 = αριθμός
Ανάγνωση float	6	-	fa0 = αριθμός
Ανάγνωση double	7	-	fa0 = αριθμός
Ανάγνωση συμβολοσειράς	8	a0 = διεύθυνση buffer a1 = μέγεθος	-
Δέσμευση μνήμης (heap)	9	a0 = bytes	a0 = διεύθυνση
Τερματισμός προγράμματος	10	-	-
Εκτύπωση χαρακτήρα	11	a0 = χαρακτήρας	-
Ανάγνωση χαρακτήρα	12	-	a0 = χαρακτήρας
Εκτύπωση ακεραίου σε hex	34	a0 = αριθμός	-
Εκτύπωση unsigned ακεραίου	36	a0 = αριθμός	-

✅ **Παράδειγμα Κώδικα (Κώδικας 9)**

Το παρακάτω πρόγραμμα εμφανίζει:

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

1. Μια συμβολοσειρά
2. Έναν ακέραιο αριθμό

```
.text
.globl __start

__start:
    la a0, msg          # διεύθυνση string
    li a7, 4            # PrintString
    ecall

    li a7, 1            # PrintInt
    la t0, number
    lw a0, 0(t0)        # φόρτωσε τον αριθμό
    ecall

    li a7, 10           # Exit
    ecall

.data
msg: .ascii "Example\n"
      .asciz "This is a sample message. Integer is: "
number: .word 37
```

🧠 **Παρατηρήσεις**

- Χρησιμοποίησε **.asciz** για συμβολοσειρές με τερματικό NULL (\0)
- Για εκτύπωση string, χρειάζεται **η διεύθυνση της μεταβλητής** στο a0
- Κάθε ecall κάνει **μια και μόνο πράξη**: να έχεις έτοιμο τον a7 πριν την καλέσεις

ΕΛΕΝΗ ΖΑΦΕΙΡΗ

ΨΕΥΔΟΕΝΤΟΛΕΣ ③

Μνημονικό	Όνομα	Περιγραφή	Χρήσεις
beqz	Branch = zero	if(R[rs1]==0) PC=PC+{imm,1b'0}	beq
bnez	Branch ≠ zero	if(R[rs1]!=0) PC=PC+{imm,1b'0}	bne
fabs.s,fabs.d	Absolute Value	F[rd] = (F[rs1]< 0) ? –F[rs1] : F[rs1]	fsgnx
fmv.s,fmv.d	FP Move	F[rd] = F[rs1]	fsgnj
fneg.s,fneg.d	FP negate	F[rd] = –F[rs1]	fsgnjn
j	Jump	PC = {imm,1b'0}	jal
jr	Jump register	PC = R[rs1]	jalr
la	Load address	R[rd] = address	auipc
li	Load imm	R[rd] = imm	addi
mv	Move	R[rd] = R[rs1]	addi
neg	Negate	R[rd] = –R[rs1]	sub
nop	No operation	R[0] = R[0]	addi
not	Not	R[rd] = ~R[rs1]	xori
ret	Return	PC = R[1]	jalr
seqz	Set = zero	R[rd] = (R[rs1]== 0) ? 1 : 0	sltiu
snez	Set ≠ zero	R[rd] = (R[rs1]!= 0) ? 1 : 0	sltu

ΚΩΔΙΚΟΙ ΛΕΙΤΟΥΡΓΙΑΣ (OPCODES) ΣΕ ΑΡΙΘΜΗΤΙΚΗ ΣΕΙΡΑ

Μνημονικό	Μορφή	OPCODE	Λειτουργία3	Λειτουργία7 ή IMM	Δεκαεξαδικά
lb	I	0000011	000		03/0
lh	I	0000011	001		03/1
lw	I	0000011	010		03/2
lbu	I	0000011	100		03/4
lhu	I	0000011	101		03/5
fence	I	0001111	000		0F/0
fence.i	I	0001111	001		0F/1
addi	I	0010011	000		13/0
slli	I	0010011	001	0000000	13/1/00
slti	I	0010011	010		13/2
sltiu	I	0010011	011		13/3
xori	I	0010011	100		13/4
srli	I	0010011	101	0000000	13/5/00
srai	I	0010011	101	0100000	13/5/20
ori	I	0010011	110		13/6
andi	I	0010011	111		13/7
auipc	U	0010111			17
sb	S	0100011	000		23/0
sh	S	0100011	001		23/1
sw	S	0100011	010		23/2
add	R	0110011	000	0000000	33/0/00
sub	R	0110011	000	0100000	33/0/20
sll	R	0110011	001	0000000	33/1/00
slt	R	0110011	010	0000000	33/2/00
sltu	R	0110011	011	0000000	33/3/00
xor	R	0110011	100	0000000	33/4/00
srl	R	0110011	101	0000000	33/5/00
sra	R	0110011	101	0100000	33/5/20
or	R	0110011	110	0000000	33/6/00
and	R	0110011	111	0000000	33/7/00
lui	U	0110111			37
beq	SB	1100011	000		63/0
bne	SB	1100011	001		63/1
blt	SB	1100011	100		63/4
bge	SB	1100011	101		63/5
bltu	SB	1100011	110		63/6
bgeu	SB	1100011	111		63/7
jalr	I	1100111	000		67/0
jal	UJ	1101111			6F
ecall	I	1110011	000	000000000000	73/0/000
ebreak	I	1110011	000	000000000001	73/0/001
CSRRW	I	1110011	001		73/1
CSRRS	I	1110011	010		73/2
CSRRC	I	1110011	011		73/3
CSRRWI	I	1110011	101		73/5
CSRRSI	I	1110011	110		73/6
CSRRCI	I	1110011	111		73/7

ΟΝΟΜΑ, ΧΡΗΣΗ, ΣΥΜΒΑΣΗ ΚΛΗΣΗΣ ΚΑΤΑΧΩΡΗΤΩΝ ④

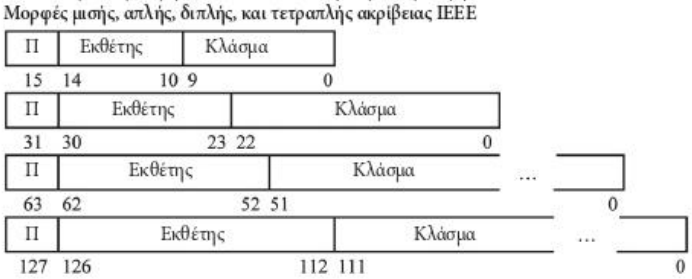
ΚΑΤΑΧΩΡΗΤΗΣ	ΟΝΟΜΑ	ΧΡΗΣΗ	ΑΠΟΘΗΚΕΥΕΤΑΙ ΑΠΟ
x0	zero	Η σταθερή τιμή 0	Δ/Ε
x1	ra	Διεύθυνση επιστροφής	Καλούσα συνάρ./λειτ.
x2	sp	Δείκτης στοίβας	Καλούμενη συνάρ./λειτ.
x3	gp	Καθολικός δείκτης	--
x4	tp	Δείκτης νήματος	--
x5-x7	t0-t2	Προσωρινοί καταχωρητές	Καλούσα συνάρ./λειτ.
x8	s0/fp	Αποθηκευμένος καταχωρητής/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

ΠΡΟΤΥΠΟ ΚΙΝΗΤΗΣ ΥΠΟΔΙΑΣΤΟΛΗΣ IEEE 754

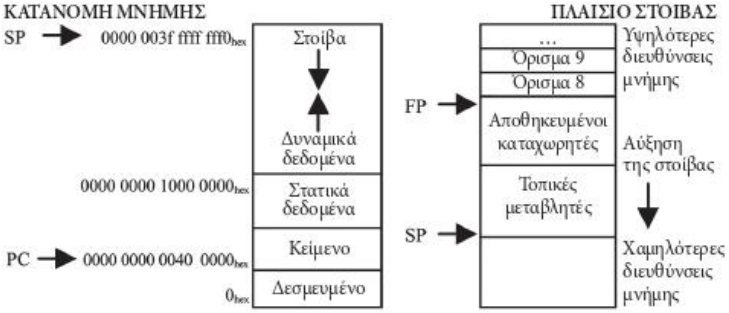
$(-1)^{\Pi} \times (1 + \text{Κλάσμα}) \times 2^{(\text{Εκθέτης} - \text{Πόλωση})}$

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

Μορφές μισής, απλής, διπλής, και τετραπλής ακρίβειας IEEE



ΚΑΤΑΝΟΜΗ ΜΝΗΜΗΣ



ΠΡΟΘΕΜΑΤΑ ΜΕΓΕΘΟΥΣ ΚΑΙ ΣΥΜΒΟΛΑ

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

Κάρτα δεδομένων αναφοράς RISC-V 1. Τραβήξτε κατά μήκος τη διάτρηση για να αφαιρέσετε την κάρτα 2. Διπλώστε την κάρτα κατακόρυφα στη μέση

RISC-V Δεδομένα αναφοράς

①

Βασικές ακέραιες εντολές του RV32I, σε αλφαβητική σειρά

Μνημονικό	Μορφή	Όνομα	Περιγραφή (σε Verilog)	Σημείωση
add	R	ADD	R[rd] = R[rs1] + R[rs2]	
addi	I	ADD Immediate	R[rd] = R[rs1] + imm	
and	R	AND	R[rd] = R[rs1] & R[rs2]	
andi	I	AND Immediate	R[rd] = R[rs1] & imm	
auipc	U	Add Upper Immediate to PC	R[rd] = PC + {imm, 12'b0}	
beq	SB	Branch EQual	if(R[rs1]==R[rs2] PC=PC+{imm,1b'0}	
bge	SB	Branch Greater than or Equal	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}	
bgeu	SB	Branch ≥ Unsigned	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}	2)
blt	SB	Branch Less Than	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}	
bltu	SB	Branch Less Than Unsigned	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}	2)
bne	SB	Branch Not Equal	if(R[rs1]!=R[rs2) PC=PC+{imm,1b'0}	
csrrc	I	Cont./Stat.RegRead&Clear	R[rd] = CSR;CSR = CSR & ~R[rs1]	
csrrci	I	Cont./Stat.RegRead&Clear Imm	R[rd] = CSR;CSR = CSR & ~imm	
csrrs	I	Cont./Stat.RegRead&Set	R[rd] = CSR; CSR = CSR | R[rs1]	
csrrsi	I	Cont./Stat.RegRead&Set Imm	R[rd] = CSR; CSR = CSR | imm	
csrrw	I	Cont./Stat.RegRead&Write	R[rd] = CSR; CSR = R[rs1]	
csrrwi	I	Cont./Stat.Reg Read&Write Imm	R[rd] = CSR; CSR = imm	
ebreak	I	Environment BREAK	Transfer control to debugger	
ecall	I	Environment CALL	Transfer control to operating system	
fence	I	Synch thread	Synchronizes threads	
fence.i	I	Synch Instr & Data	Synchronizes writes to instruction stream	
jal	UJ	Jump & Link	R[rd] = PC+4; PC = PC + {imm,1b'0}	
jalr	I	Jump & Link Register	R[rd] = PC+4; PC = R[rs1]+imm	
lb	I	Load Byte	R[rd] = {24'bM[](7),M[R[rs1]+imm](7:0)}	3) 4)
lbu	I	Load Byte Unsigned	R[rd] = {24'b0,M[R[rs1]+imm](7:0)}	
lh	I	Load Halfword	R[rd] = {16'bM[](15),M[R[rs1]+imm](15:0)}	
lhu	I	Load Halfword Unsigned	R[rd] = {16'b0,M[R[rs1]+imm](15:0)}	4)
lui	U	Load Upper Immediate	R[rd] = {imm, 12'b0}	
lw	I	Load Word	R[rd] = {M[R[rs1]+imm](31:0)}	
or	R	OR	R[rd] = R[rs1] | R[rs2]	
ori	I	OR Immediate	R[rd] = R[rs1] | imm	4)
sb	S	Store Byte	M[R[rs1]+imm](7:0) = R[rs2](7:0)	
sh	S	Store Halfword	M[R[rs1]+imm](15:0) = R[rs2](15:0)	
sll	R	Shift Left	R[rd] = R[rs1] << R[rs2]	
slli	I	Shift Left Immediate	R[rd] = R[rs1] << imm	
slt	R	Set Less Than	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	
slti	I	Set Less Than Immediate	R[rd] = (R[rs1] < imm) ? 1 : 0	
sltiu	I	Set < Immediate Unsigned	R[rd] = (R[rs1] < imm) ? 1 : 0	
sltu	R	Set Less Than Unsigned	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	
sra	R	Shift Right Arithmetic	R[rd] = R[rs1] >> R[rs2]	
srai	I	Shift Right Arith Imm	R[rd] = R[rs1] >> imm	2)
srl	R	Shift Right (Word)	R[rd] = R[rs1] >> R[rs2]	2)
srli	I	Shift Right Immediate	R[rd] = R[rs1] >> imm	5)
sub,subw	R	SUBtract (Word)	R[rd] = R[rs1] – R[rs2]	5)
sw	S	Store Word	M[R[rs1]+imm](31:0) = R[rs2](31:0)	
xor	R	XOR	R[rd] = R[rs1] ^ R[rs2]	
xori	I	XOR Immediate	R[rd] = R[rs1] ^ imm	

Σημειώσεις: 1) Η λειτουργία εκτελείται εξ ορισμού με απρόσημους ακεραίους (αντί για συμπλήρωμα ως προς 2)

2) Το λιγότερο σημαντικό bit της διεύθυνσης διακλάδωσης της jalr παίρνει τιμή 0

3) Οι (προσημασμένες) εντολές load επεκτείνουν το bit προσήμου των δεδομένων ώστε να γεμίσει ο καταχωρητής 32 bit

4) Με επανάληψη του bit προσήμου για να συμπληρωθούν τα αριστερότερα bit του αποτελέσματος κατά τη δεξιά ολίσθηση

5) Πολλαπλασιασμός με έναν τελεστέο προσημασμένο και έναν απρόσημο

6) Η έκδοση single εκτελεί λειτουργία απλής ακρίβειας χρησιμοποιώντας τα 32 δεξιότερα bit ενός καταχωρητή F των 64 bit

7) Κατηγοριοποίηση των εγγραφών με μια μάσκα 10 bit που δείχνει ποιες ιδιότητες ισχύουν (π.χ. –άπειρο, –0, +0, +άπειρο, μη κανονικοποιημένος, …)

8) Ατομική λειτουργία μνήμης: Τίποτε άλλο δεν μπορεί να παρεμβληθεί μεταξύ της ανάγνωσης και της εγγραφής της θέσης μνήμης

Το άμεσο πεδίο υφίσταται επέκταση προσήμου στη RISC-V

ΣΥΝΟΛΟ ΕΝΤΟΛΩΝ ΑΡΙΘΜΗΤΙΚΟΥ ΠΥΡΗΝΑ

②

Επέκταση Πολλαπλασιασμού RV64M

Μνημονικό	Μορφή	Όνομα	Περιγραφή (σε Verilog)	Σημείωση
mul	R	MULtiply	R[rd] = (R[rs1] * R[rs2])(63:0)	
mulh	R	MULtiply High	R[rd] = (R[rs1] * R[rs2])(127:64)	
mulhsu	R	MULtiply High Unsigned	R[rd] = (R[rs1] * R[rs2])(127:64)	2)
mulhu	R	MULtiply upper Half Unsigned	R[rd] = (R[rs1] * R[rs2])(127:64)	6)
div	R	DIVide	R[rd] = (R[rs1] / R[rs2])	
divu	R	DIVide Unsigned	R[rd] = (R[rs1] / R[rs2])	2)
rem	R	REMainder	R[rd] = (R[rs1] % R[rs2])	
remu	R	REMainder Unsigned	R[rd] = (R[rs1] % R[rs2])	2)

Επεκτάσεις κινητής υποδιαστολής RV64F και RV64D

fld,flw	I	Load (Word)	F[rd] = M[R[rs1]+imm]	
fsd,fsw	S	Store (Word)	M[R[rs1]+imm] = F[rd]	
fadd.s,fadd.d	R	ADD	F[rd] = F[rs1] + F[rs2]	7)
fsub.s,fsub.d	R	SUBtract	F[rd] = F[rs1] – F[rs2]	7)
fmul.s,fmul.d	R	MULtiply	F[rd] = F[rs1] * F[rs2]	7)
fdiv.s,fdiv.d	R	DIVide	F[rd] = F[rs1] / F[rs2]	7)
fsqrt.s,fsqrt.d	R	SQuare RooT	F[rd] = sqrt(F[rs1])	7)
fmadd.s,fmadd.d	R	Multiply-ADD	F[rd] = F[rs1] * F[rs2] + F[rs3]	7)
fmsub.s,fmsub.d	R	Multiply-SUBtract	F[rd] = F[rs1] * F[rs2] - F[rs3]	7)
fmnsub.s,fmnsub.d	R	Negative Multiply-ADD	F[rd] = –(F[rs1] * F[rs2] – F[rs3])	7)
fmnadd.s,fmnadd.d	R	Negative Multiply-SUBtract	F[rd] = –(F[rs1] * F[rs2] + F[rs3])	7)
fsgnj.s,fsgnj.d	R	SiGN source	F[rd] = { F[rs2]<63>,F[rs1]<62:0>}	7)
fsgnjn.s,fsgnjn.d	R	Negative SiGN source	F[rd] = { (~F[rs2]<63>), F[rs1]<62:0>}	7)
fsgnjx.s,fsgnjx.d	R	Xor SiGN source	F[rd] = {F[rs2]<63>^F[rs1]<63>, F[rs1]<62:0>}	7)
fmin.s,fmin.d	R	MINimum	F[rd] = (F[rs1] < F[rs2]) ? F[rs1] : F[rs2]	7)
fmax.s,fmax.d	R	MAXimum	F[rd] = (F[rs1] > F[rs2]) ? F[rs1] : F[rs2]	7)
feq.s,feq.d	R	Compare Float EQual	R[rd] = (F[rs1]== F[rs2]) ? 1 : 0	7)
flt.s,flt.d	R	Compare Float Less Than	R[rd] = (F[rs1]< F[rs2]) ? 1 : 0	7)
fle.s,fle.d	R	Compare Float Less than or =	R[rd] = (F[rs1]<= F[rs2]) ? 1 : 0	7)
fclass.s,fclass.d	R	Classify Type	R[rd] = class(F[rs1])	7,8)
fmv.s.x,fmv.d.x	R	Move from Integer	F[rd] = R[rs1]	7)
fmv.x.s,fmv.x.d	R	Move to Integer	R[rd] = F[rs1]	7)
fcvt.d.s	R	Convert from SP to DP	F[rd] = single(F[rs1])	
fcvt.s.d	R	Convert from DP to SP	F[rd] = double(F[rs1])	
fcvt.s.w,fcvt.d.w	R	Convert from 32b Integer	F[rd] = float(R[rs1](31:0))	7)
fcvt.s.l,fcvt.d.l	R	Convert from 64b Integer	F[rd] = float(R[rs1](63:0))	7)
fcvt.s.wu,fcvt.d.wu	R	Convert from 32b Int Unsigned	F[rd] = float(R[rs1](31:0))	2,7)
fcvt.s.lu,fcvt.d.lu	R	Convert from 64b Int Unsigned	F[rd] = float(R[rs1](63:0))	2,7)
fcvt.w.s,fcvt.w.d	R	Convert to 32b Integer	R[rd](31:0) = integer(F[rs1])	7)
fcvt.l.s,fcvt.l.d	R	Convert to 64b Integer	R[rd](63:0) = integer(F[rs1])	7)
fcvt.wu.s,fcvt.wu.d	R	Convert to 32b Int Unsigned	R[rd](31:0) = integer(F[rs1])	2,7)
fcvt.lu.s,fcvt.lu.d	R	Convert to 64b Int Unsigned	R[rd](63:0) = integer(F[rs1])	2,7)

Επέκταση ατομικών (αδιαίρετων) εντολών RV64A

amoadd.w,amoadd.d	R	ADD	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] + R[rs2]	9)
amoand.w,amoand.d	R	AND	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] & R[rs2]	9)
amomax.w,amomax.d	R	MAXimum	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amomaxu.w,amomaxu.d	R	MAXimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amomin.w,amomin.d	R	MINimum	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amominu.w,amominu.d	R	MINimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amoor.w,amoor.d	R	OR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] | R[rs2]	9)
amoswap.w,amoswap.d	R	SWAP	R[rd] = M[R[rs1]], M[R[rs1]] = R[rs2]	9)
amoxor.w,amoxor.d	R	XOR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] ^ R[rs2]	9)
lr.w,lr.d	R	Load Reserved	R[rd] = M[R[rs1]], reservation on M[R[rs1]]	
sc.w,sc.d	R	Store Conditional	if reserved, M[R[rs1]] = R[rs2], R[rd] = 0; else R[rd] = 1	

ΜΟΡΦΕΣ ΕΝΤΟΛΩΝ ΠΥΡΗΝΑ

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