ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΥΠΟΛΟΓΙΣΤΩΝ

Φροντιστηριακές Διαλέξεις

ΔΙΔΑΣΚΟΝΤΕΣ

- Δρ. Άννα Κεφάλα, Μέλος ΕΔΙΠ
- Δρ. Χρήστος Κούτσικας, Εντεταλμένος Διδάσκων
- Αναστάσιος Δρακόπουλος, MSc., Αποσπασμένος Καθ. Μ.Ε.

intro.cs.aueb@gmail.com

ΔΙΑΔΙΚΑΣΤΙΚΑ ΘΕΜΑΤΑ ΦΡΟΝΤΙΣΤΗΡΙΑΚΩΝ ΔΙΑΛΕΞΕΩΝ

- **Πέμπτη 17:00-19:00, Αμφ. Γ** (*για <u>όλους</u> τους φοιτητές, κατά περίπτωση μπορεί εναλλακτικά να πραγματοποιηθεί την Τρίτη 17:00-19:00, Αμφ. Δο*)
  - διεξαγωγή φροντιστηριακής διάλεξης
  - ώρες συζήτησης απορίων
- Η διεξαγωγή των φροντιστηριακών διαλέξεων **ανακοινώνεται** μέσω **eClass**

ΠΕΡΙΕΧΟΜΕΝΟ ΦΡΟΝΤΙΣΤΗΡΙΑΚΩΝ ΔΙΑΛΕΞΕΩΝ

- Εισαγωγή στην γλώσσα Python
- Επιλεγμένα θέματα θεωρίας και υλοποίηση με Python
- Συζήτηση απορριών και διευκρινίσεις για τις εργασίες

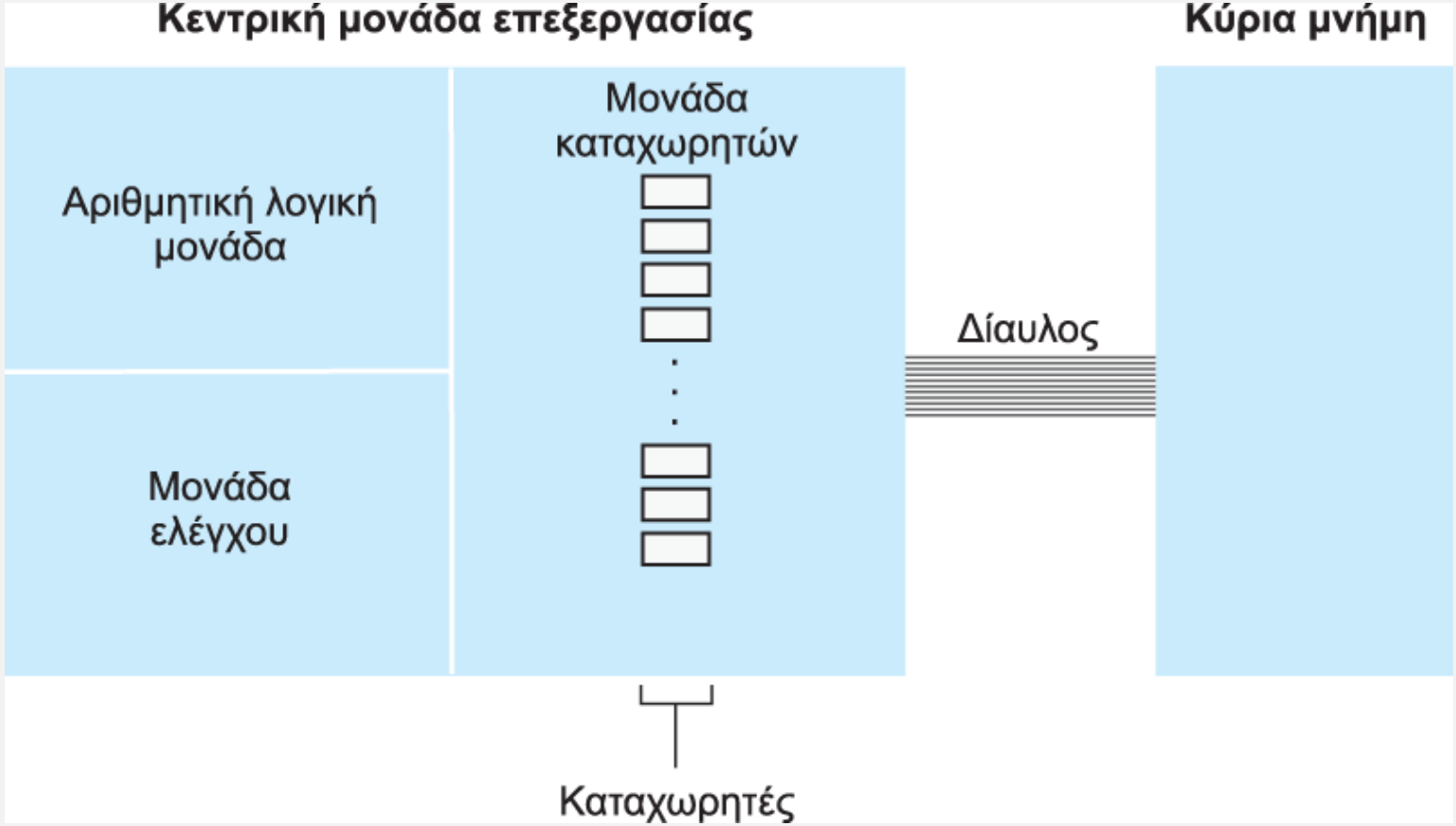
ΕΡΓΑΣΙΕΣ ΚΑΙ ΑΞΙΟΛΟΓΗΣΗ

- **2 ομαδικές εργασίες** (2 ή 3 άτομα ανά ομάδα, ίδια σύσταση ομάδων και για τις δύο εργασίες) → **40%** τελικού βαθμού
- 1 ομαδική εργασία bonus → **+15%** στον τελικό βαθμό (*αν βαθμός εργασίας*>=5)
- Αν δεν έχετε παραδώσει εργασία δεν εξετάζεστε προφορικά για την εν λόγω εργασία, και δε βαθμολογείστε
- Εργασίες χωρίς **προφορική εξέταση** επίσης δε βαθμολογούνται
- Αν δεν υποβάλλετε εργασίες, ο μέγιστος βαθμός που μπορείτε να διεκδικήσετε είναι **7**
- Οι εργασίες συνυπολογίζονται στον τελικό βαθμό μόνο αν βαθμός στην τελική γραπτή εξέταση >= 4
- Αν πάρετε στην τελική εξέταση >= 5, περνάτε το μάθημα, ανεξαρτήτως βαθμού στις εργασίες ή αν έχετε παραδώσει εργασίες.

ΚΥΚΛΟΣ ΜΗΧΑΝΗΣ

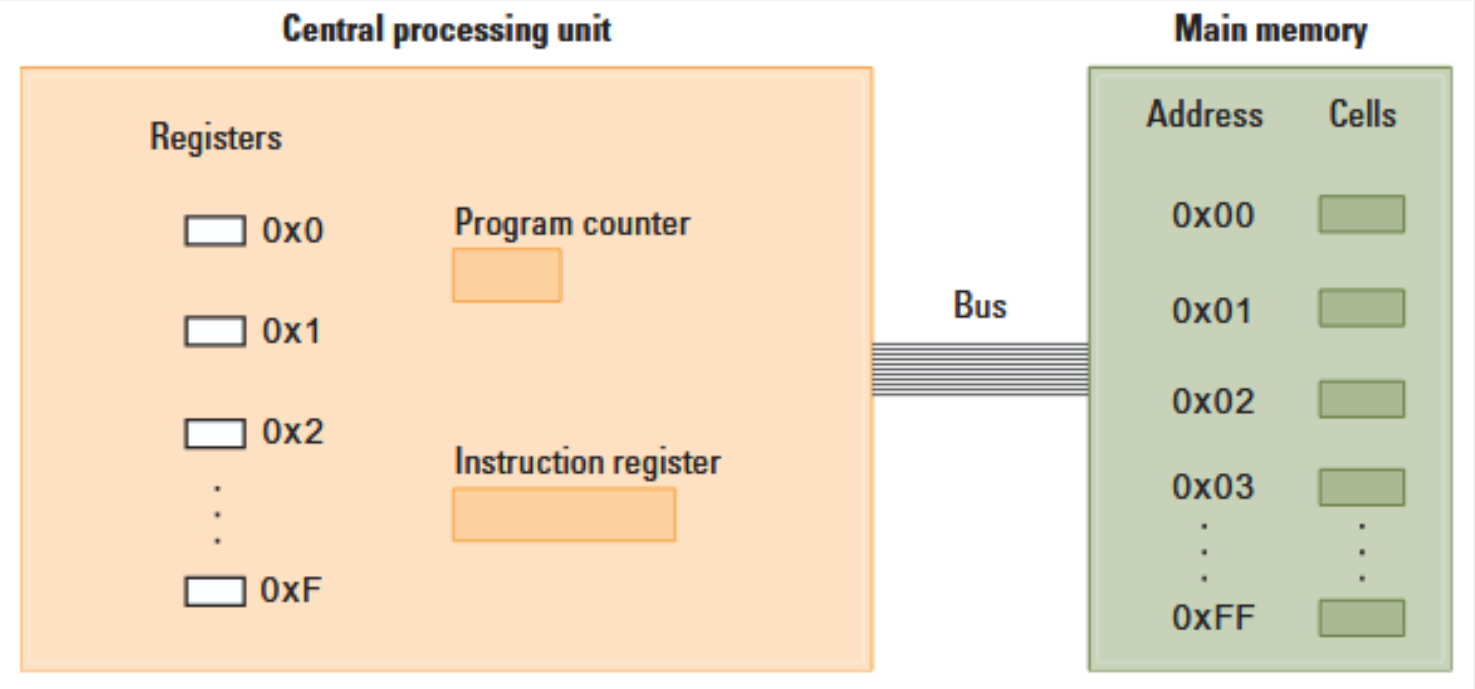
Εκτέλεση εντολής/προγράμματος

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΗ



ΚΥΚΛΟΣ ΜΗΧΑΝΗΣ | ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

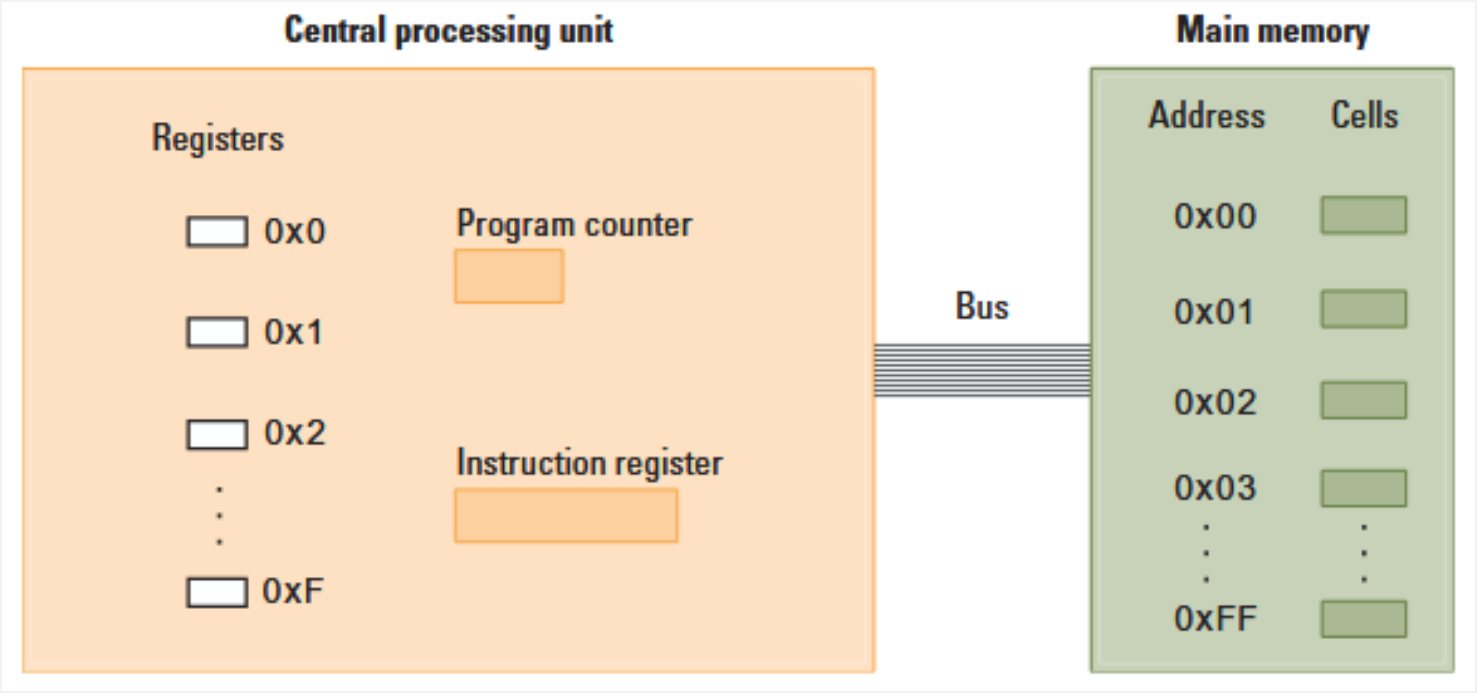
1. **Fetch** (προσκόμισε): ανάκτησε επόμενη εντολή από μνήμη (όπως υποδεικνύεται από Program Counter)
   → Καταχωρητή εντολών & κατόπιν αύξησε το μετρητή προγράμματος
2. Decode
3. Execute
4. Store



Μετρητής προγράμματος (Program Counter, PC) : περιέχει την διεύθυνση μνήμης για την επόμενη εντολή προς εκτέλεση
Καταχωρητής εντολών (Instruction Register, IR): περιέχει την τρέχουσα εντολή

ΚΥΚΛΟΣ ΜΗΧΑΝΗΣ | ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ (2)

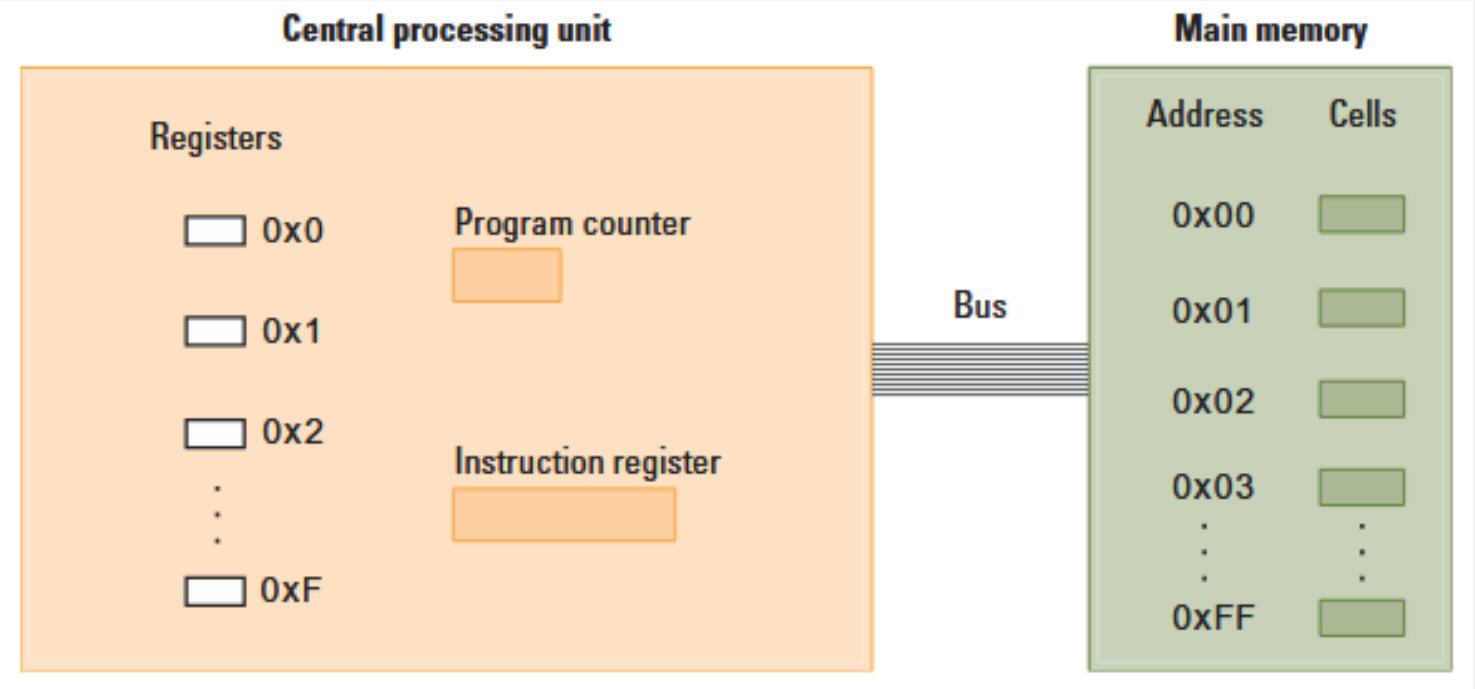
1. Fetch
2. **Decode** (αποκωδικοποίησε): αποκωδικοποίησε το pattern (σχήμα) bit (*εντολή*) στον Καταχωρητή εντολών
3. Execute
4. Store



Μετρητής προγράμματος (Program Counter, PC) : περιέχει την διεύθυνση μνήμης για την επόμενη εντολή προς εκτέλεση
Καταχωρητής εντολών (Instruction Register, IR): περιέχει την τρέχουσα εντολή

ΚΥΚΛΟΣ ΜΗΧΑΝΗΣ | ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ (3)

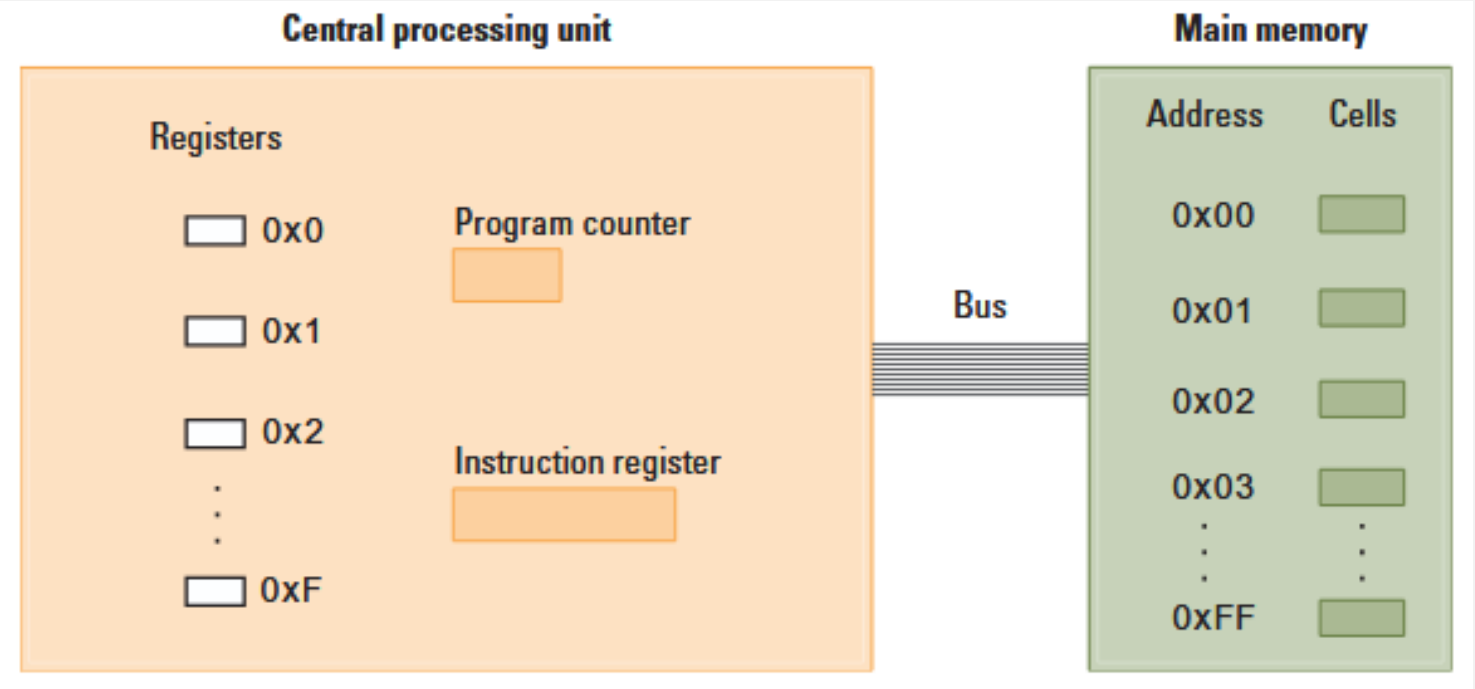
1. Fetch
2. Decode
3. **Execute** (εκτέλεσε): εκτέλεσε την ενέργεια που απαιτείται από την εντολή στον Καταχωρητή εντολών
4. Store



Μετρητής προγράμματος (Program Counter, PC) : περιέχει την διεύθυνση μνήμης για την επόμενη εντολή προς εκτέλεση
Καταχωρητής εντολών (Instruction Register, IR): περιέχει την τρέχουσα εντολή

ΚΥΚΛΟΣ ΜΗΧΑΝΗΣ | ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ (4)

1. Fetch
2. Decode
3. Execute
4. **Store** (αποθήκευσε): αποτέλεσμα εκτέλεσης εντολής αποθηκεύεται σε καταχωρητή ή κύρια μνήμη



Μετρητής προγράμματος (Program Counter, PC) : περιέχει την διεύθυνση μνήμης για την επόμενη εντολή προς εκτέλεση
Καταχωρητής εντολών (Instruction Register, IR): περιέχει την τρέχουσα εντολή

ΚΥΚΛΟΣ ΜΗΧΑΝΗΣ

Προσομοίωση με γλώσσα προγραμματισμού (Python)

ΚΥΚΛΟΣ ΜΗΧΑΝΗΣ & ΚΩΔΙΚΑΣ

- Απλός κώδικας (έστω Python)

A = 5

B = 10

C = A + B

Μέσω διερμηνέα

CPU:
αποκωδικοποιείται σε δεκάδες απλές εντολές μηχανής που εκτελούνται σύμφωνα με τον Κύκλο Μηχανής

ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΜΝΗΜΗΣ & ΚΑΤΑΧΩΡΗΤΩΝ

Στοιχεία αρχιτεκτονικής υπολογιστή	Αναπαράσταση σε Python
Κεντρική μνήμη (RAM)	Μία λίστα (έστω memory) που περιέχει τα δεδομένα και τις εντολές (κώδικα)
Μετρητής προγράμματος (Program Counter)	Μία μεταβλητή που περιέχει τον δείκτη (index) της επόμενης εντολής στη λίστα memory
Καταχωρητές (Registers)	Απλές μεταβλητές ή ένα **λεξικό*** για όλους τους καταχωρητές

* **Dictionary (λεξικό)**: χρησιμοποιείται για αποθήκευση τιμών δεδομένων σε ζεύγη key:value

ΠΡΟΣΟΜΟΙΩΣΗ ΜΝΗΜΗΣ

- Κεντρική μνήμη → Λίστα (list) memory

```
memory = [
    100,        # Εντολή: LOAD 100 (Φόρτωσε την τιμή 100)
    200,        # Εντολή: LOAD 200
    300,        # Εντολή: ADD
    0,          # Διεύθυνση για αποθήκευση αποτελέσματος (R3)
    ...         # χώρος για δεδομένα (μεταβλητές)
]
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΤΑΧΩΡΗΤΩΝ

- Καταχωρητές → Λεξικό (dictionary) registers

```
registers = {
    "R1": 0,
    "R2": 0,
    "R3": 0,
    "IR": 0, # Καταχωρητής Εντολών
    "PC": 0  # Μετρητής Προγράμματος (δείχνει στο 0)
}
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | ΣΥΝΑΡΤΗΣΗ

Κάθε στάδιο Κύκλου Μηχανής → μία λειτουργία ή **function** που ενημερώνει τη μνήμη ή τους καταχωρητές

Συνάρτηση (function)

- περιέχει ένα κομμάτι κώδικα που εκτελεί μια συγκεκριμένη εργασία.
- (μπορεί να) δέχεται κάποια δεδομένα ως είσοδο, ο κώδικας μέσα στη συνάρτηση εκτελεί ενέργειες με τα δεδομένα και επιστρέφεται το αποτέλεσμα.

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | FETCH

1. **Fetch**

```
def fetch():     # ορισμός (definition) της συνάρτησης
    global registers      # έχουν δηλωθεί εκτός της function (κυρίως πρόγραμμα)
        # η μεταβλητή instruction παίρνει την τιμή στη διεύθυνση του PC
    instruction = memory[registers["PC"]]
    registers["IR"] = instruction
        # προετοιμασία για την επόμενη εντολή (αύξησε τον PC)
    registers["PC"] += 1
    print(f"FETCH: Loaded instruction {instruction} to IR. PC updated to
        {registers['PC']}")
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | DECODE

2. Decode

```
def decode():
    opcode = registers["IR"]  # εντολή στο IR, ανάλογα με την τιμή, αποκωδικοποίησε στην αντίστοιχη εντολή
    if opcode == 100:    # αν opcode=100 ...
        operation = "LOAD_R1"
        print("DECODE: Decoded operation is LOAD R1.")
        return operation # επίστρεψε το αποτέλεσμα της function στο κυρίως πρόγραμμα
    elif opcode == 200: # αλλιώς αν opcode=200 ...
        operation = "LOAD_R2"
        print("DECODE: Decoded operation is LOAD R2.")
        return operation
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | DECODE (2)

2. Decode

```
# συνέχεια function decode
  elif opcode == 300:
      operation = "ADD_R3"
      print(f"DECODE: Decoded operation is ADD R1, R2 to R3.")
      return operation
   else:            #  αλλιώς, σε οποιαδήποτε άλλη περίπτωση
      print("DECODE: Unknown instruction.")
      return None
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | EXECUTE

3. Execute

```
def execute(operation):
    global registers
    if operation == "LOAD_R1":
        # απλοποιημένο LOAD: γεμίζουμε τον R1 με μια σταθερή τιμή
        registers["R1"] = 5
        print(f"EXECUTE: Loaded value 5 into R1. R1={registers['R1']}")
    elif operation == "LOAD_R2":
        registers["R2"] = 7
        print(f"EXECUTE: Loaded value 7 into R2. R2={registers['R2']}")
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | EXECUTE (2)

3. Execute

```
# συνέχεια function execute
    elif operation == "ADD_R3":
        result = registers["R1"] + registers["R2"]
        registers["R3"] = result
        print(f"EXECUTE: Added R1+R2. Result {result} stored in R3.")


# η ALU εκτελεί την πράξη...
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | STORE

4. Store

το αποτέλεσμα ($R3$) αποθηκεύεται πίσω στη μνήμη στη διεύθυνση 3

```
def store():
    global memory
        # Αποθηκεύει το αποτέλεσμα του R3 στην 4η θέση της μνήμης (δείκτης 3)
    memory[3] = registers["R3"]
    print(f"STORE: Result {registers['R3']} stored in memory[3].")
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | ΕΚΤΕΛΕΣΗ

Εκτέλεση πλήρους Κύκλου Μηχανής (*κυρίως πρόγραμμα*)

```
# ΚΥΚΛΟΣ 1: LOAD R1
fetch()
op1 = decode()
execute(op1)
# Δεν χρειάζεται Store σε αυτό το βήμα
 # ΚΥΚΛΟΣ 2: LOAD R2
fetch()
op2 = decode()
execute(op2)
```

ΠΡΟΣΟΜΟΙΩΣΗ ΚΥΚΛΟΥ ΜΗΧΑΝΗΣ | ΕΚΤΕΛΕΣΗ (2)

Εκτέλεση πλήρους Κύκλου Μηχανής (*κυρίως πρόγραμμα*) *# συνέχεια*

```
# ΚΥΚΛΟΣ 3: ADD R3
fetch()
op3 = decode()
execute(op3)
# ΚΥΚΛΟΣ 4: STORE A
fetch() # θα φορτώσει το 0, το οποίο το χρησιμοποιούμε ως διεύθυνση αποθήκευσης
store()
print(f"\n FINAL MEMORY STATE:", memory)
```