ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Εισαγωγή στους Υπολογιστές

Ενότητα 12: Οργάνωση Κεντρικής Μονάδας Επεξεργασίας (CPU)

Βασίλης Παλιουράς

Πολυτεχνική Σχολή

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Σκοποί ενότητας

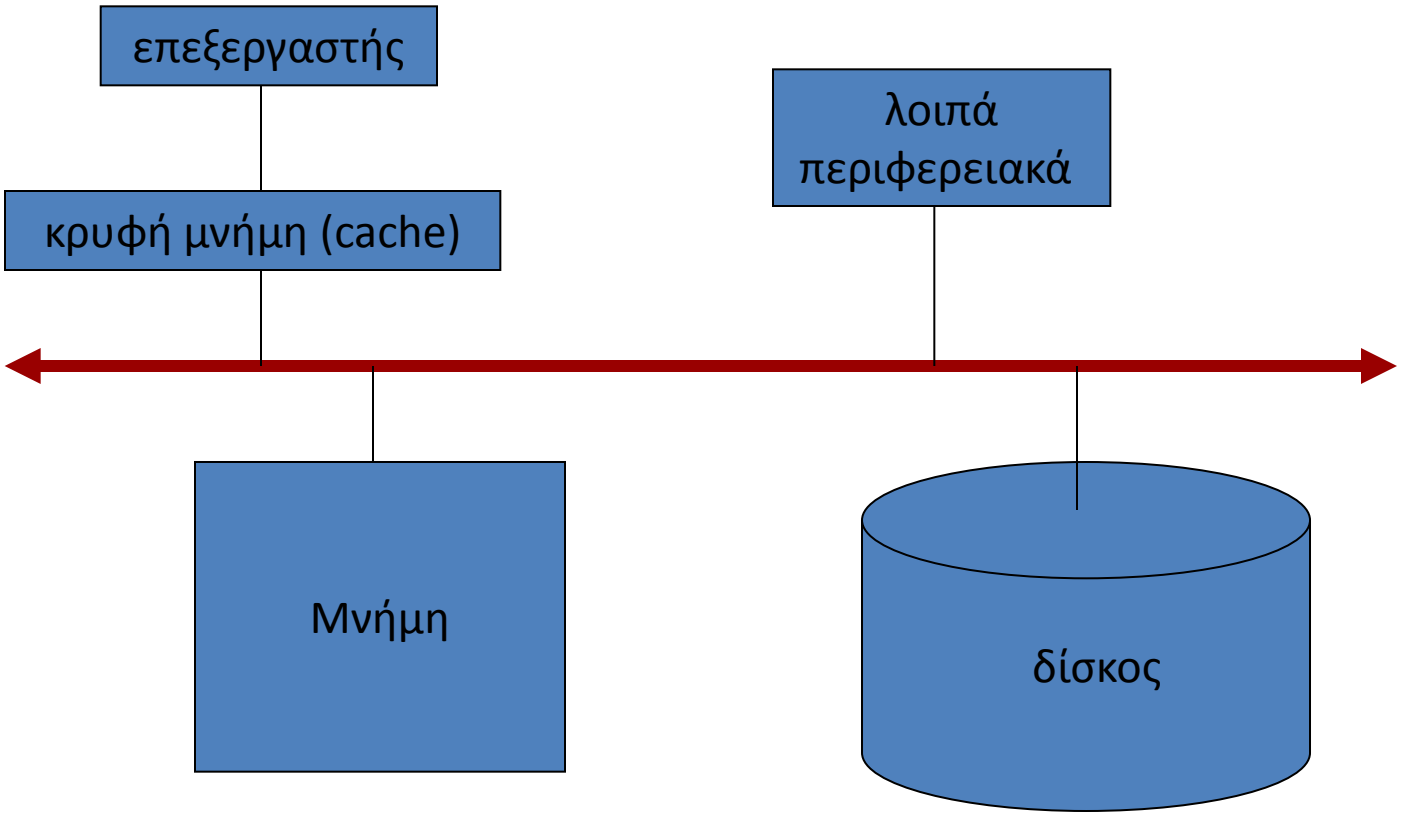
- Οργάνωση Κεντρικής Μονάδας Επεξεργασίας (CPU)
- Στοιχειώδης οργάνωση ΚΜΕ
- Γλώσσας μετάδοσης καταχωρητών, (Register Transfer Language)
- Κώδικας μηχανής

Περιεχόμενα ενότητας

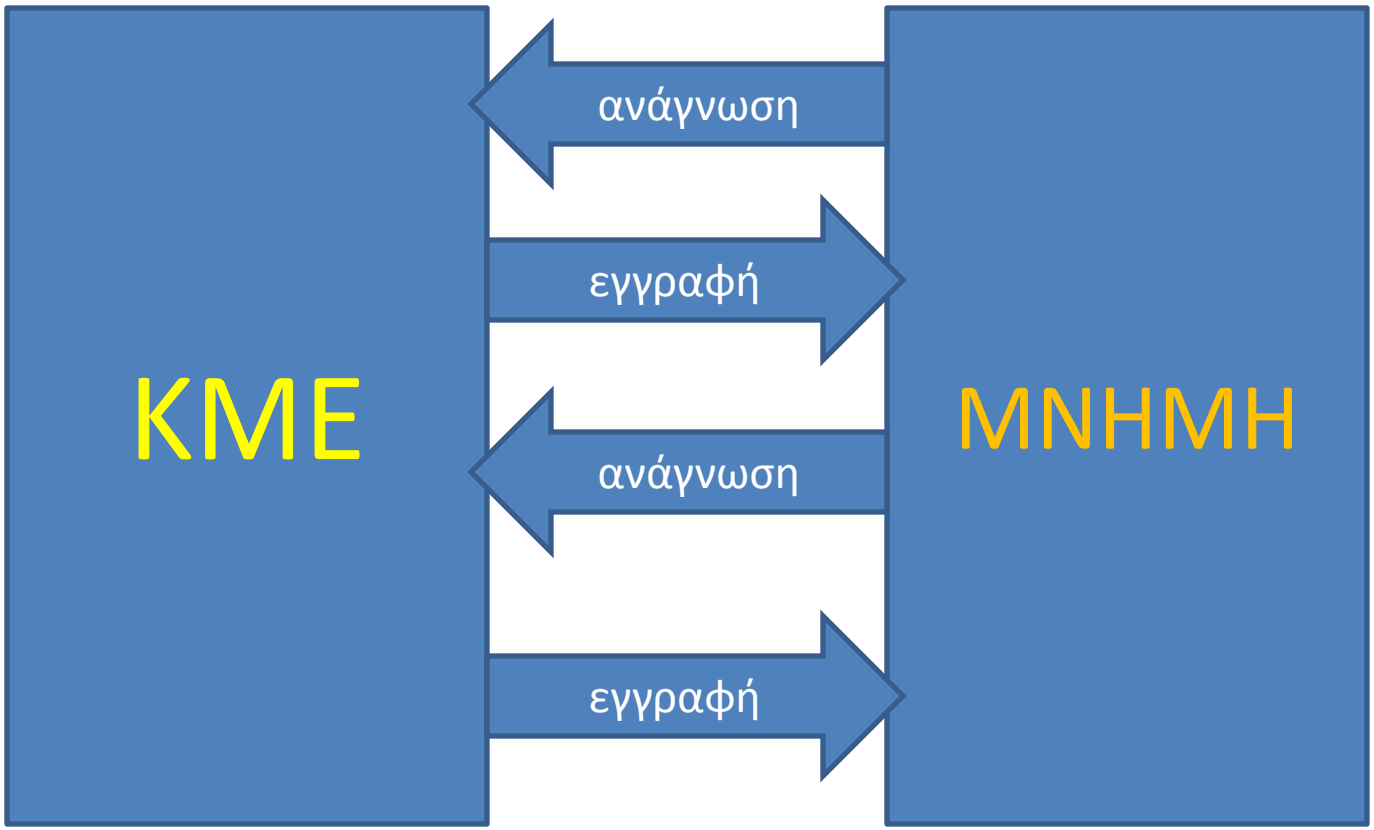
- Οργάνωση Κεντρικής Μονάδας Επεξεργασίας (CPU)
- Στοιχειώδης οργάνωση ΚΜΕ
- Γλώσσας μετάδοσης καταχωρητών, (Register Transfer Language)
- Κώδικας μηχανής

Οργάνωση Κεντρικής Μονάδας Επεξεργασίας (CPU)

Στοιχειώδης οργάνωση υπολογιστή



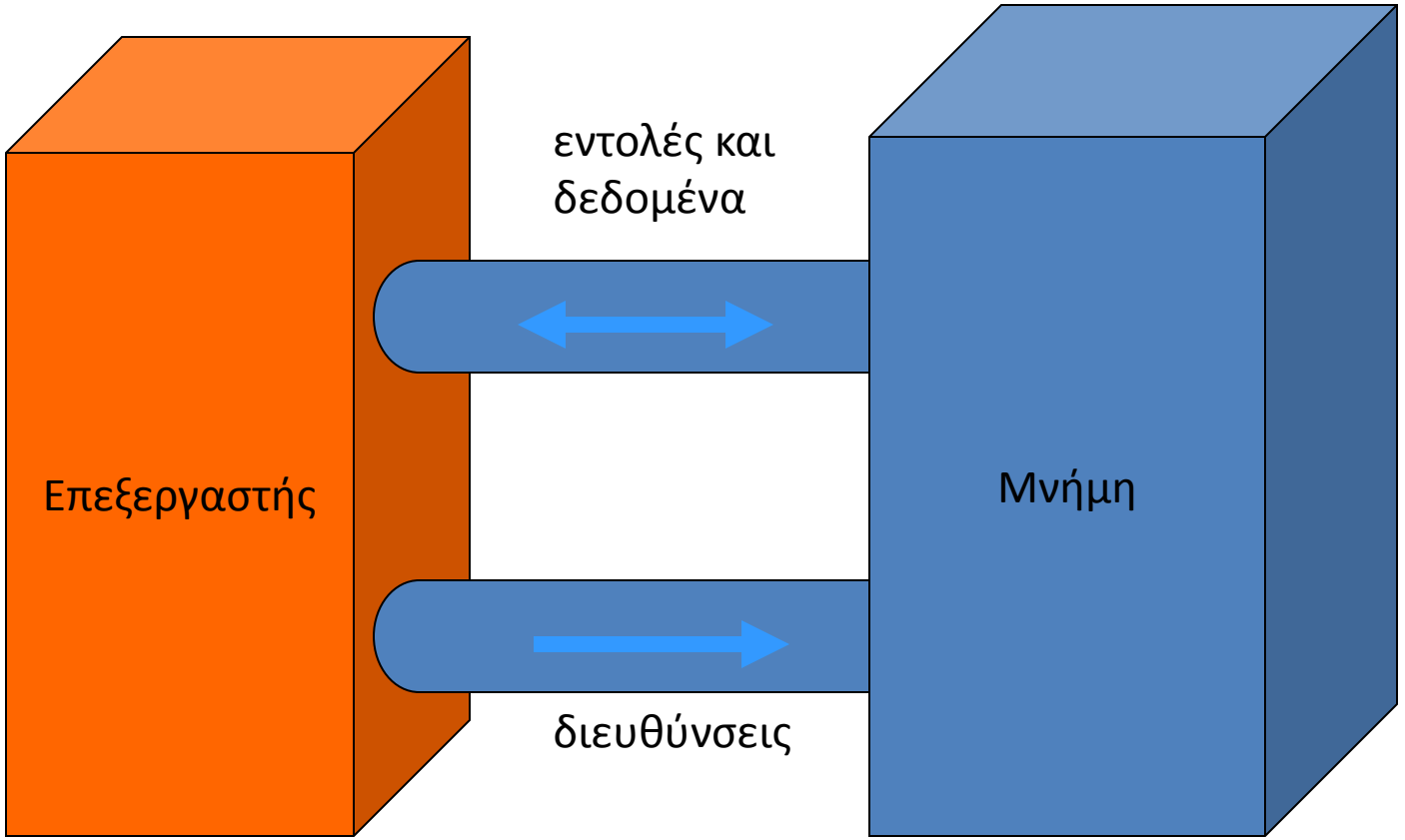
ΡΟΛΟΣ ΚΜΕ
(CPU, Central Processing Unit)



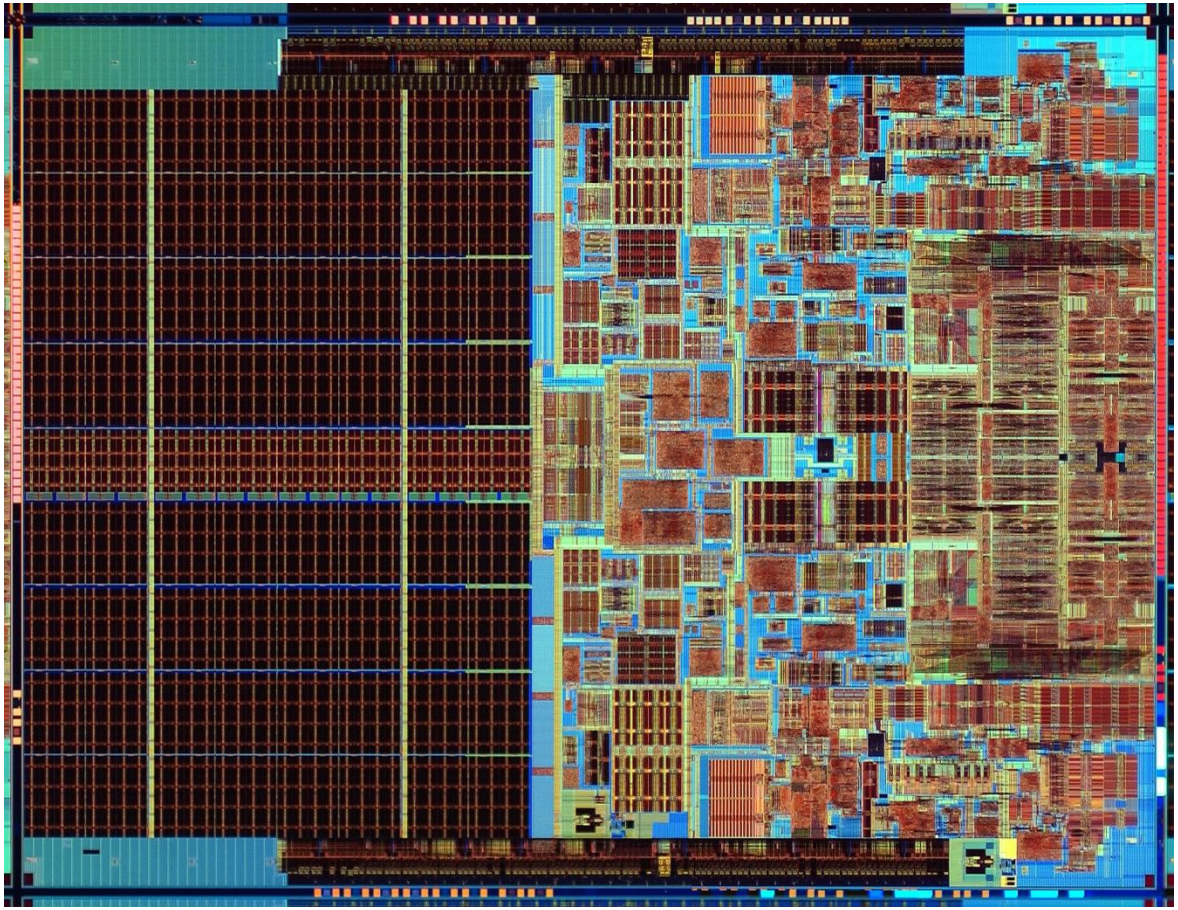
Λογική και πράξεις

Πρόγραμμα και δεδομένα

Το προγραμματιστικό μοντέλο



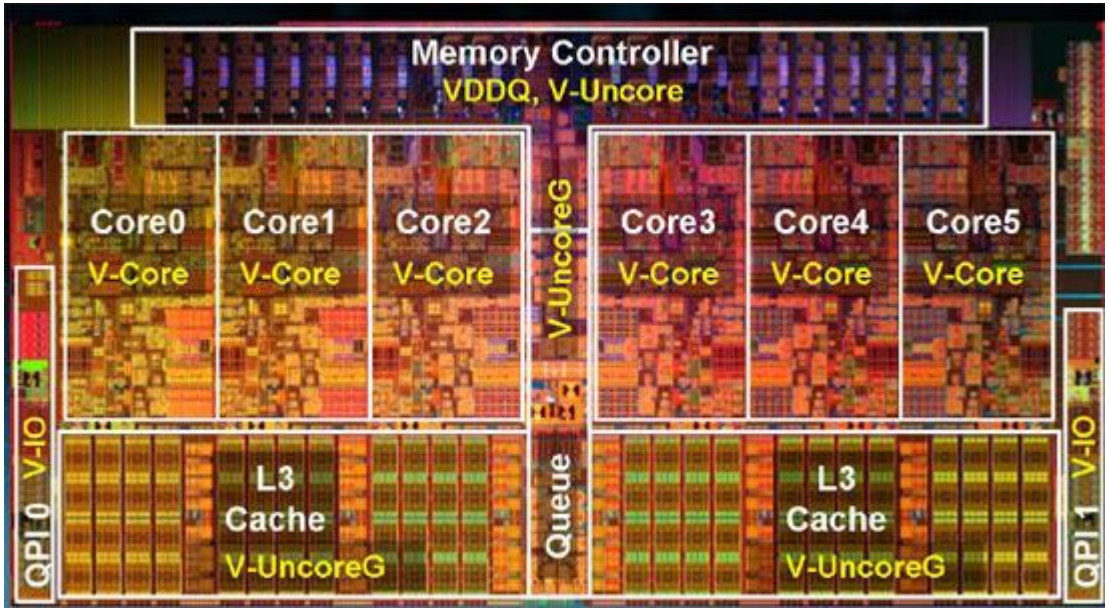
dual core μPs



πηγή: intel.com (tm)

1.17 billion transistors, ~240mm sq.

- Ένα chip 6 επεξεργαστικοί πυρήνες, μνήμες cache:



Intel, Westmere 6C (κωδικό Gulftown)

Μοντέλο μνήμης

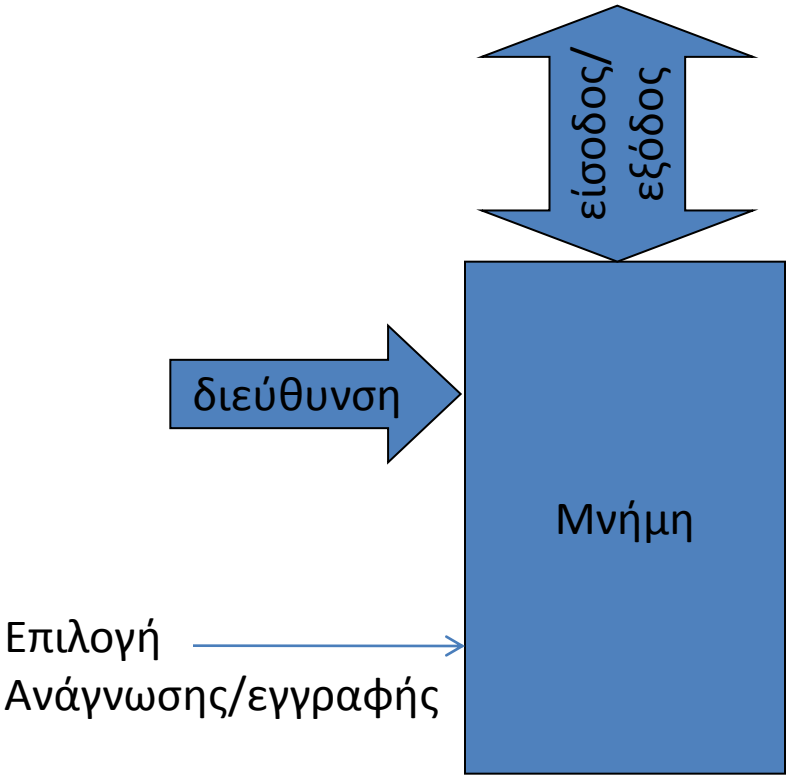
ΜΝΗΜΗ

- Πρόγραμμα
- Δεδομένα
  - σταθερές, μεταβλητές, είσοδοι, έξοδοι

ΜΝΗΜΗ

διεύθυνση	δεδομένα
0	00100
1	00001
2	00100
…	…
N -1	00000

Η μνήμη RAM (Random Access Memory)

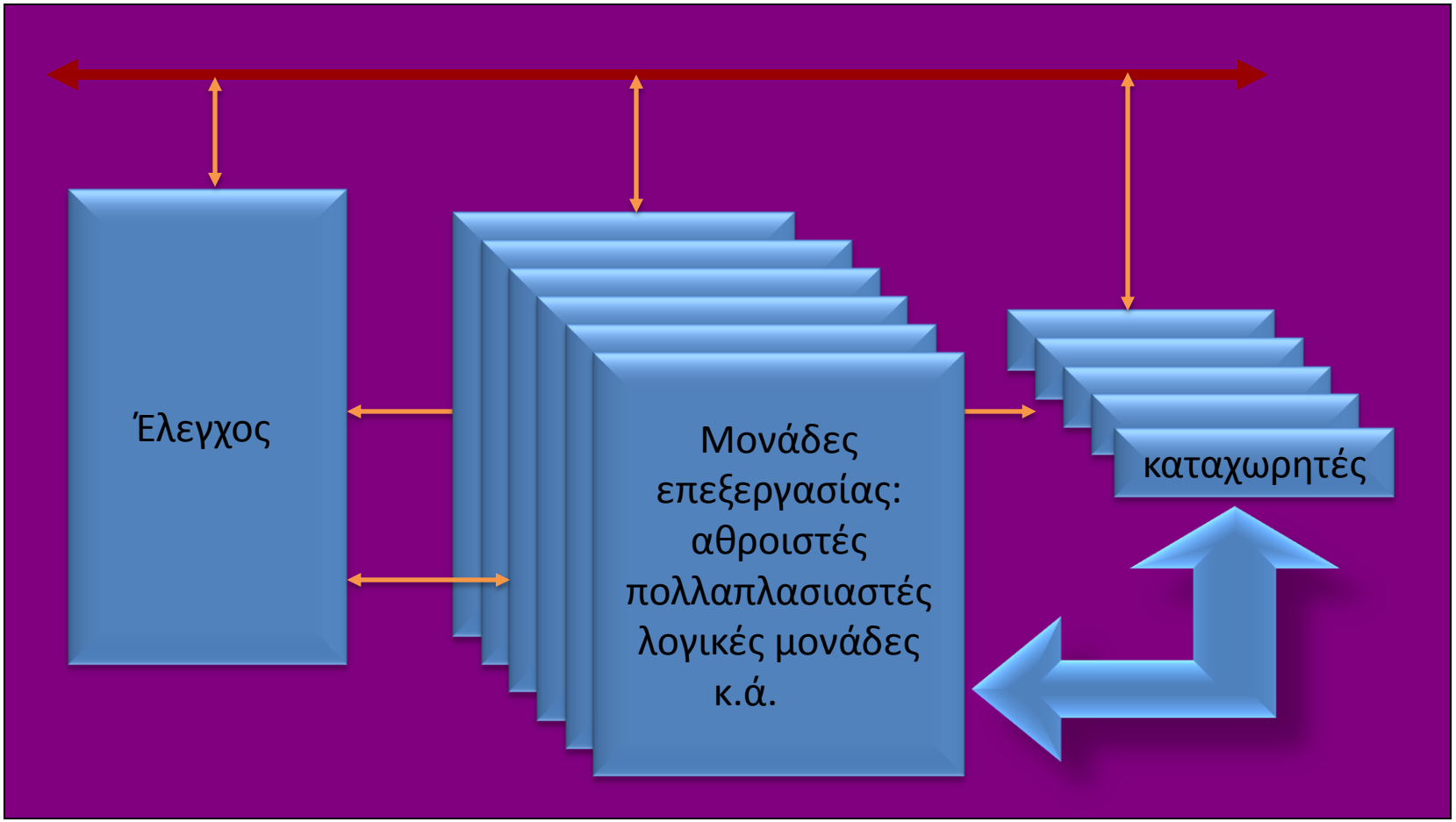


Διεύθυνση (address)	Περιεχόμενα
0x0000	00111111
0x0001	10110101
0x0002	11011110
0x0003	10111100
…	
0x00FF	00111101
0x0100	10101110
…	

Πόσα bytes διαθέτονται για κάθε μεταβλητή;

Περιεχόμενα:
Δεδομένα και *εντολές*

Στοιχειώδης Οργάνωση ΚΜΕ



Καταχωρητές και ΚΜΕ

- Χρήση: Αποθήκευση πληροφοριών στην ΚΜΕ
- Διαχωρίζονται ανάλογα με το ρόλο τους
  - Ειδικού σκοπού: Υποστήριξη θεμελιωδών λειτουργιών
    - Program Counter (Απαριθμητής Προγράμματος)
    - Status Register (Καταχωρητής Κατάστασης)
    - Stack Pointer (Δείκτης Σωρού)
  - Γενικού σκοπού
    - Συσσωρευτής,
    - Καταχωρητές δεδομένων,
- Διασύνδεση αναλόγως αρχιτεκτονικής

Εκτέλεση Προγραμμάτων

Οι γλώσσες υψηλού επιπέδου ως αφαίρεση

- Οι γλώσσες υψηλού επιπέδου είναι μια μορφή ***αφαίρεσης***.
  - Η πολυπλοκότητα του **κώδικα μηχανής** αποκρύπτεται από τον προγραμματιστή.
- Ο μεταγλωττιστής (compiler) αντιστοιχίζει στον κώδικα σύνολα εντολών κώδικα μηχανής
- Παρόμοια η δουλειά του interpreter

Βρόχος επανάληψης **for**

python

```
sum = 0
for i in range(10):
    sum = sum + i
print sum
```

C

```
#include <stdio.h>
main () {
    register int i, sum = 0;
    for ( i = 0; i < 10; i++)
        sum =  sum + i;
    printf ("\ti: %d\n", sum);
}
```

Το ίδιο πρόγραμμα σε *assembly* (για επεξεργαστή Pentium)

```
_main:
        pushl %ebp
        movl %esp,%ebp
        subl $16,%esp
        pushl %esi
        pushl %ebx
        call ___main
        xorl %esi,%esi
        xorl %ebx,%ebx
L11:
        cmpl $9,%ebx
        jle L14
        jmp L12
L14:
        addl %ebx,%esi
L13:
        incl %ebx
        jmp L11
```

```
L12:
        addl $-8,%esp
        pushl %esi
        pushl $LC0
        call _printf
        addl $16,%esp
L10:
        leal -24(%ebp),%esp
        popl %ebx
        popl %esi
        movl %ebp,%esp
        popl %ebp
        ret
```

Αντιστοιχίζοντας κώδικα assembly σε κώδικα μηχανής Παράδειγμα η εντολή ADD

04 ib	ADD AL, imm8	Add imm8 to AL
05 iw	ADD AX, imm16	Add imm16 to AX
05 id	ADD EAX, imm32	Add imm32 to EAX
80 /0 ib	ADD r/m8,imm8	Add imm8 to r/m8
81 /0 iw	ADD r/m16,imm16	Add imm16 to r/m16
81 /0 id	ADD r/m32,imm32	Add imm32 to r/m32
83 /0 ib	ADD r/m16,imm8	Add sign-extended imm8 to r/m16
83 /0 ib	ADD r/m32,imm8	Add sign-extended imm8 to r/m32
00 / r	ADD r/m8,r8	Add r8 to r/m8
01 / r	ADD r/m16,r16	Add r16 to r/m16
01 / r	ADD r/m32,r32	Add r32 to r/m32
02 / r	ADD r8,r/m8	Add r/m8 to r8
03 / r	ADD r16,r/m16	Add r/m16 to r16
03 / r	ADD r32,r/m32	Add r/m32 to r32

...κώδικας μηχανής (ενδεικτικά)

1011001110010101 0

0111010101010111 0

0101010101010101 1

1101010111011100 1

0001010101110111 0

1111110101010101 0

Χαρακτηριστικά γλωσσών assembly

- Οι επεξεργαστές έχουν χαρακτηριστικό σύνολο εντολών
- Τα προγράμματα είναι «μεγαλύτερα»
  - τι σημαίνει μεγαλύτερα; μονάδα μέτρησης οι «Γραμμές κώδικα»
  - περισσότερος χρόνος για να γραφούν
  - ευαίσθητα σε λάθη, δυσνόητα
- έλλειψη *δομής* (structure)
  - ακόμα και οι απλούστεροι βρόχοι, υπό συνθήκη διακλαδώσεις κτλ δομούνται με jumps
  - ιστορική εξέλιξη ⇒ δομημένος προγραμματισμός (structured programming)
- Πλεονεκτήματα
  - ταχύτητα, μικρό μέγεθος κώδικα σε bytes...

pyCPU: Μοντέλο ενός υποθετικού επεξεργαστή

```
instr = [ 'load',      # 0
          'store',     # 1
          'add',       # 2
          'sub',       # 3
          'loadd',     # 4
          'halt',      # 5
          'addd',      # 6
          'subd',      # 7
          'jmp',       # 8
          'jmp_N',     # 9
          'jmp_Z',     # 10
          'jmp_O',     # 11
          'jmp_C',     # 12
          'reserved1', # 13
          'reserved2', # 14
          'reserved3'  # 15
          ]
```

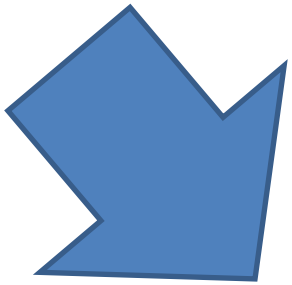
Παράδειγμα

```
def asm (opcode, argument) :
    return (instr.index(opcode) << 12) + ( argument & 0xFFF )
```

Από εντολή σε λέξη (αριθμητική) για αποθήκευση στη μνήμη

Παράδειγμα 2 από pyCPU

```
# example 2: count down loop
M = [asm('load',6),    #   D0 = M[6]
    asm('subd',1),    # 1:D0 = D0 - 1
    asm('jmp_Z',4),   #   if D0 == 0 goto 4
    asm('jmp',1),     #   goto 1
    asm('store',6),   # 4:M[6] = D0
    asm('halt',0),    #   halt
    10]
CPU(0)
```

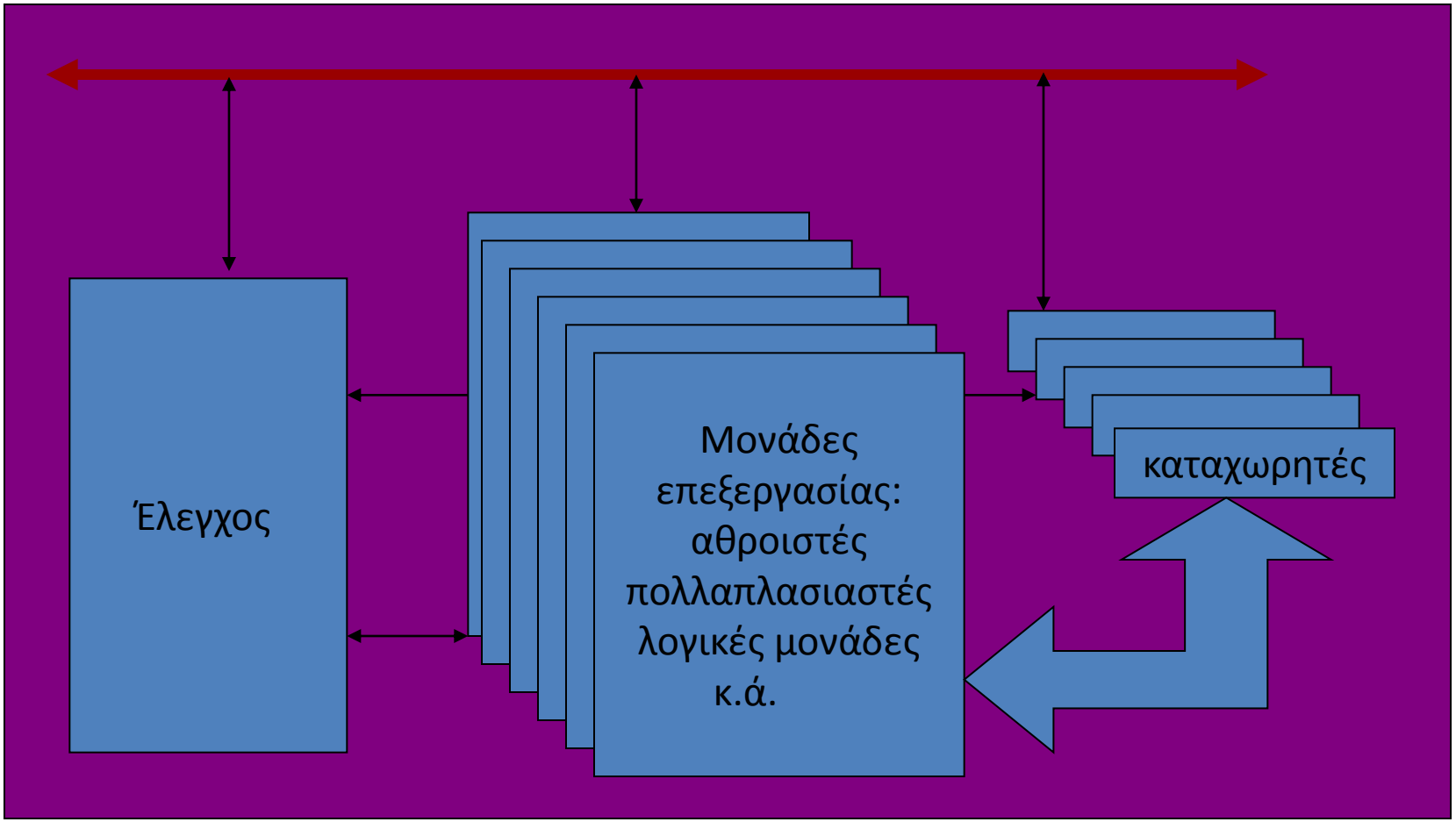


M:
[6, 28673, 40964, 32769, 4102, 20480, 0]

Παράδειγμα 2 από pyCPU

Γλώσσα	assembly	Περιεχόμενα μνήμης	
D0 = M[6]	0: load 6	6	0000 0000 0000 0110
1: D0 = D0 – 1	1: subd 1	28673	0111 0000 0000 0001
if D0 == 0 goto 4	2:Jmp_z 4	40964	1010 0000 0000 0100
goto 1	3: Jmp 1	32769	1010 0000 0000 0100
4: M[6] = D0	4: Store 6	4102	0001 0000 0000 0110
halt	5: halt	20480	0101 0000 0000 0000

Στοιχειώδης Οργάνωση ΚΜΕ



Γλώσσα Μετάδοσης Καταχωρητών (Register Transfer Language)

- Περιγράφει τις διαδικασίες που εκτελεί η κεντρική μονάδα, ως μεταφορά δεδομένων μεταξύ των καταχωρητών και της μνήμης
- Ενδεικτικά:

  [ΚΔΜ] ← [ΑΠ]

  [ΑΠ] ← [ΑΠ] + 1

  [ΑΠ] ← 1026

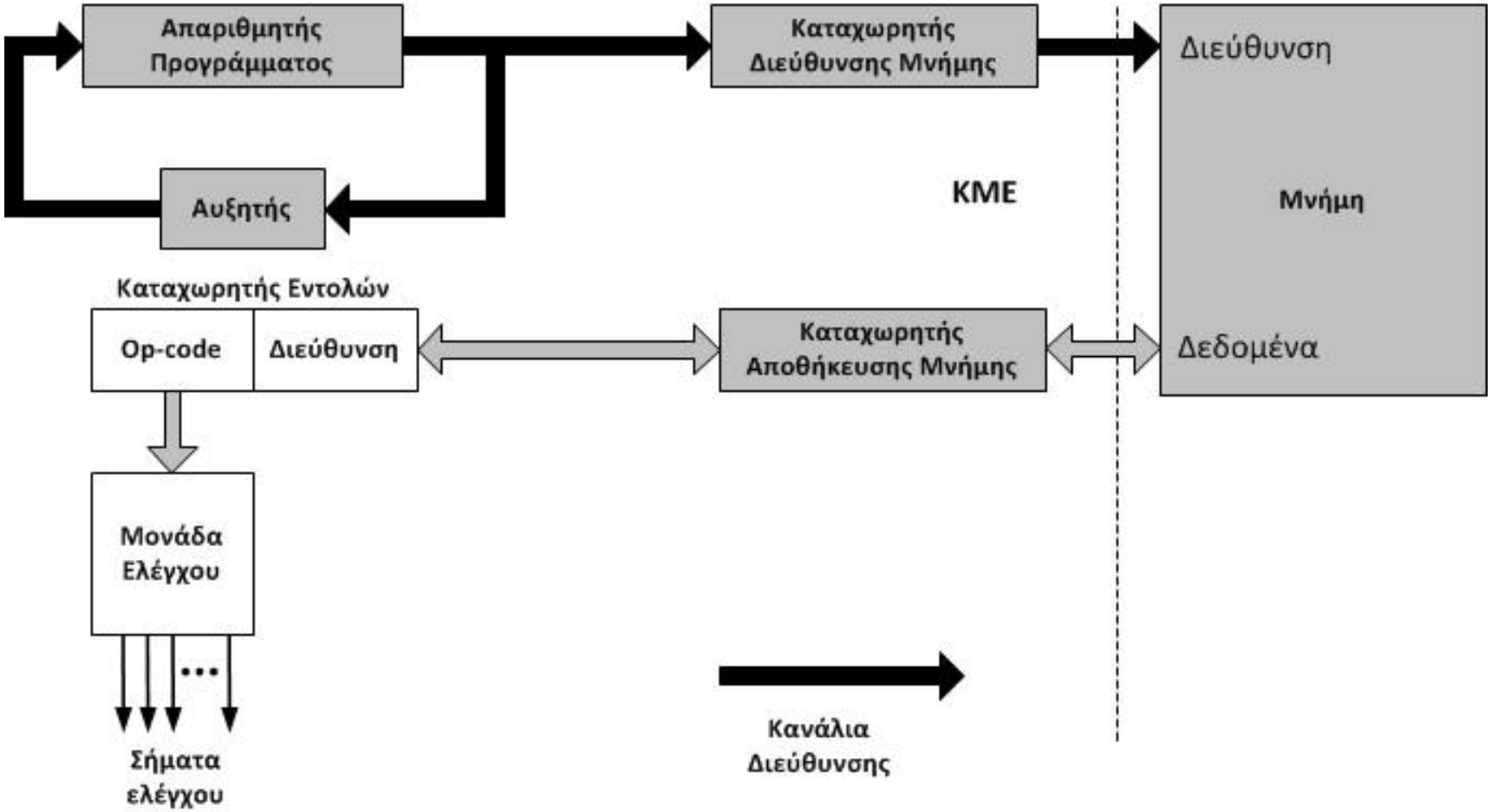
  [Σ] ←[Σ] + 5

Γλώσσα Μετάδοσης Καταχωρητών (Register Transfer Language)

- [M(20)] = 6 Η θέση 20 της μνήμηςλαμβάνει την τιμή 6

- [M(20)] ← [M(6)] + 5

Κανάλι Διεύθυνσης



Κύκλος *εκτέλεσης* εντολής

- Προσκόμιση εντολής από τη μνήμη (FETCH)
- Αποκωδικοποίηση εντολής (DECODE)
- Προσκόμιση επιπλέον ορισμάτων (READ_OP)
  - Μόνο αν χρειάζεται
  - Υπολογισμός αποτελέσματος
- Εκτέλεση εντολής (EXECUTE)
- Αποθήκευση αποτελέσματος (WRITE_BACK)

Εκτέλεση εντολής στο pyCPU

```
def CPU (start_address):
    global run, IP
    IP = start_address
    while run:
        FETCH()
        DECODE()
        READ_OP()
        EXECUTE()
        WRITE_BACK()
```

Προσκόμιση εντολής (instruction fetch)

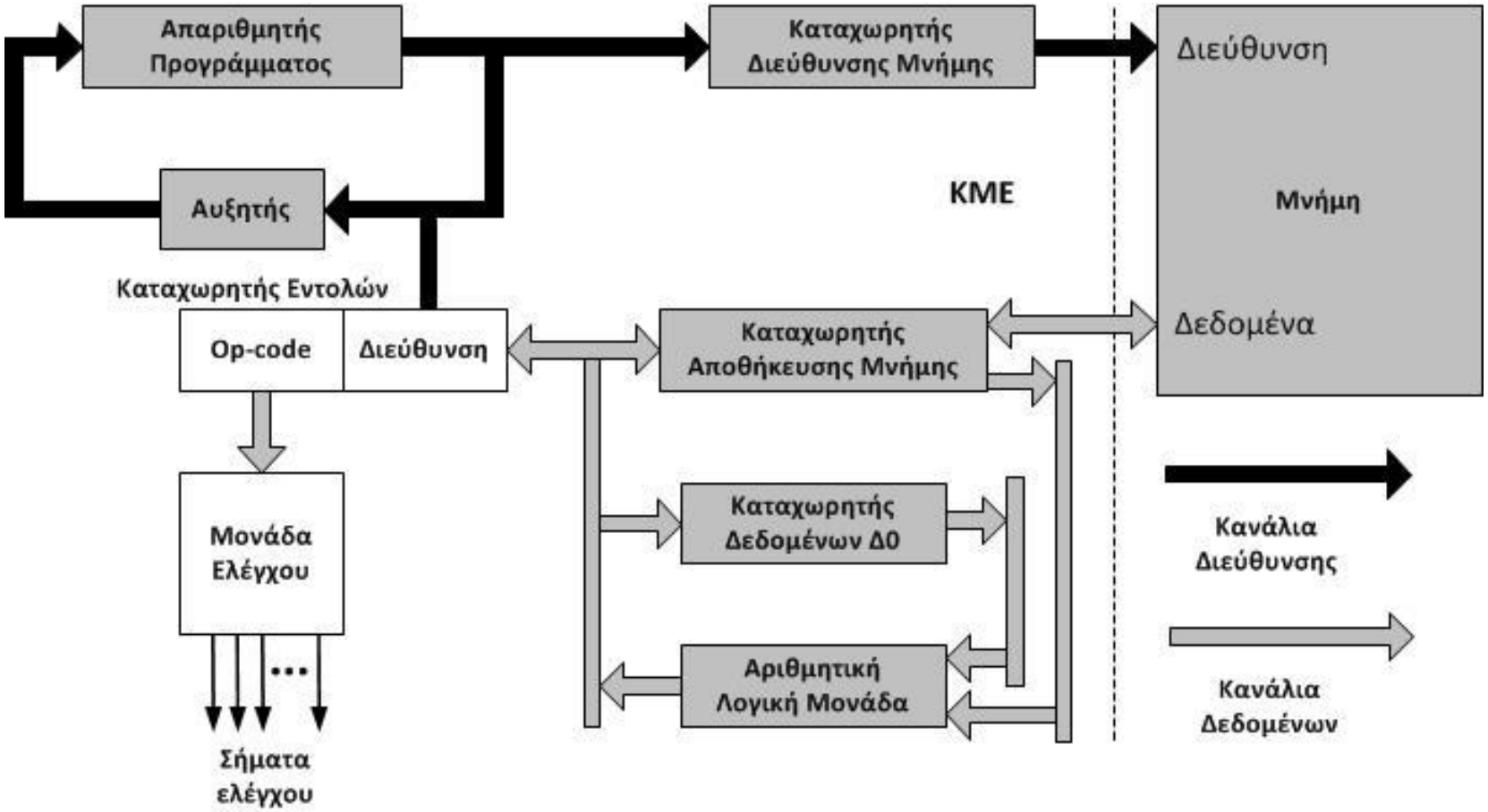
[ΚΔΜ] ← [ΑΠ]	Αντιγραφή περιεχομένου ΑΠ στο ΚΔΜ
[ΑΠ] ← [ΑΠ] + 1	Αύξηση του περιεχομένου του ΑΠ
[ΚΑΜ] ← [Μ([ΚΔΜ])]	Ανάγνωση εντολής από τη μνήμη
[ΚΕ] ← [ΚΑΜ]	Μεταφορά της εντολής στον ΚΕ
ΜΕ ← [ΚΕ(op-code)]	Μετάδοση του op-code στη ΜΕ

ΚΕ	Καταχωρητής Εντολής
ΜΕ	Μονάδα Ελέγχου

Παράδειγμα

```
def FETCH ():
    global MAR, IP, MDR, M, IR
    MAR = IP
    IP = IP + 1
    MDR = M[MAR]
```

Κανάλια δεδομένων



Παραδείγματα Αριθμητικών και Λογικών Πράξεων

Πράξη	Είδος	Μνημονικό
Άθροιση	Αριθμητική	ADD
Αφαίρεση	Αριθμητική	SUB
Πολλαπλασιασμός	Αριθμητική	MULT
Διαίρεση	Αριθμητική	DIV
Διαίρεση με 2	Αριθμητική	ASR
Πολλαπλασιασμός με 2	Αριθμητική	ASL
Σύζευξη	Λογική	AND
Διάζευξη	Λογική	OR
Άρνηση	Λογική	NOT
Αριστερή ολίσθηση	Λογική	LSL
Δεξιά ολίσθηση	Λογική	LSR

Παράδειγμα: ο 8085

Mnemonic	Op	ZAPC	~s	Description	Notes
ACI n	CE	*****	7	Add with Carry Immediate	A=A+n+CY
ADC r	8F	*****	4	Add with Carry	A=A+r+CY(21X)
ADC M	8E	*****	7	Add with Carry to Memory	A=A+[HL]+CY
ADD r	87	*****	4	Add	A=A+r (20X)
ADD M	86	*****	7	Add to Memory	A=A+[HL]
ADI n	C6	*****	7	Add Immediate	A=A+n
ANA r	A7	****0	4	AND Accumulator	A=A&r (24X)
ANA M	A6	****0	7	AND Accumulator and Memory	A=A&[HL]
ANI n	E6	**0*0	7	AND Immediate	A=A&n
CALL a	CD	-----	18	Call unconditional	-[SP]=PC,PC=a
CC a	DC	-----	9	Call on Carry	If CY=1(18~s)
CM a	FC	-----	9	Call on Minus	If S=1 (18~s)
CMA	2F	-----	4	Complement Accumulator	A=~A
CMC	3F	----*	4	Complement Carry	CY=~CY
CMP r	BF	*****	4	Compare	A-r (27X)
CMP M	BF	*****	7	Compare with Memory	A-[HL]
CNC a	D4	-----	9	Call on No Carry	If CY=0(18~s)
CNZ a	C4	-----	9	Call on No Zero	If Z=0 (18~s)
CP a	F4	-----	9	Call on Plus	If S=0 (18~s)
CPE a	EC	-----	9	Call on Parity Even	If P=1 (18~s)
CPI n	FE	*****	7	Compare Immediate	A-n
CPO a	E4	-----	9	Call on Parity Odd	If P=0 (18~s)
CZ a	CC	-----	9	Call on Zero	If Z=1 (18~s)
DAA	27	*****	4	Decimal Adjust Accumulator	A=BCD format
DAD B	09	----*	10	Double Add BC to HL	HL=HL+BC
DAD D	19	----*	10	Double Add DE to HL	HL=HL+DE
DAD H	29	----*	10	Double Add HL to HL	HL=HL+HL
DAD SP	39	----*	10	Double Add SP to HL	HL=HL+SP
DCR r	3D	****-	4	Decrement	r=r-1 (0X5)
DCR M	35	****-	10	Decrement Memory	[HL]=[HL]-1
DCX B	0B	-----	6	Decrement BC	BC=BC-1
DCX D	1B	-----	6	Decrement DE	DE=DE-1
DCX H	2B	-----	6	Decrement HL	HL=HL-1
DCX SP	3B	-----	6	Decrement Stack Pointer	SP=SP-1
DI	F3	-----	4	Disable Interrupts	
EI	FB	-----	4	Enable Interrupts	
HLT	76	-----	5	Halt	
IN p	DB	-----	10	Input	A=[p]
INR r	3C	****-	4	Increment	r=r+1 (0X4)
INR M	3C	****-	10	Increment Memory	[HL]=[HL]+1
INX B	03	-----	6	Increment BC	BC=BC+1
INX D	13	-----	6	Increment DE	DE=DE+1
INX H	23	-----	6	Increment HL	HL=HL+1
INX SP	33	-----	6	Increment Stack Pointer	SP=SP+1
JMP a	C3	-----	7	Jump unconditional	PC=a
JC a	DA	-----	7	Jump on Carry	If CY=1(10~s)
JM a	FA	-----	7	Jump on Minus	If S=1 (10~s)
JNC a	D2	-----	7	Jump on No Carry	If CY=0(10~s)
JNZ a	C2	-----	7	Jump on No Zero	If Z=0 (10~s)
JP a	F2	-----	7	Jump on Plus	If S=0 (10~s)
JPE a	EA	-----	7	Jump on Parity Even	If P=1 (10~s)
JPO a	E2	-----	7	Jump on Parity Odd	If P=0 (10~s)
JZ a	CA	-----	7	Jump on Zero	If Z=1 (10~s)
LDA a	3A	-----	13	Load Accumulator direct	A=[a]
LDAX B	0A	-----	7	Load Accumulator indirect	A=[BC]
LDAX D	1A	-----	7	Load Accumulator indirect	A=[DE]
LHLD a	2A	-----	16	Load HL Direct	HL=[a]
LXI B,nn	01	-----	10	Load Immediate BC	BC=nn
LXI D,nn	11	-----	10	Load Immediate DE	DE=nn
LXI H,nn	21	-----	10	Load Immediate HL	HL=nn
LXI SP,nn	31	-----	10	Load Immediate Stack Ptr	SP=nn
MOV r1,r2	7F	-----	4	Move register to register	r1=r2 (1XX)
MOV M,r	77	-----	7	Move register to Memory	[HL]=r (16X)
MOV r,M	7E	-----	7	Move Memory to register	r=[HL] (1X6)
MVI r,n	3E	-----	7	Move Immediate	r=n (0X6)
MVI M,n	36	-----	10	Move Immediate to Memory	[HL]=n
NOP	00	-----	4	No Operation	
ORA r	B7	**0*0	4	Inclusive OR Accumulator	A=Avr (26X)
ORA M	B6	**0*0	7	Inclusive OR Accumulator	A=Av[HL]
ORI n	F6	**0*0	7	Inclusive OR Immediate	A=Avn
OUT p	D3	-----	10	Output	[p]=A
PCHL	E9	-----	6	Jump HL indirect	PC=[HL]
POP B	C1	-----	10	Pop BC	BC=[SP]+
POP D	D1	-----	10	Pop DE	DE=[SP]+
POP H	E1	-----	10	Pop HL	HL=[SP]+
POP PSW	F1	-----	10	Pop Processor Status Word	{PSW,A}=[SP]+

Παράδειγμα: Π = Ρ + Θ

- MOVE Θ, Δ0
- ADD Ρ, Δ0
- MOVE Δ0, Π

Αν επιτρέπει απευθείας πράξεις με τη μνήμη

- MOVE Θ, Δ1
- MOVE Ρ, Δ2
- ADD Δ1,Δ2,Δ0
- MOVE Δ0, Π

Μηχανή Καταχωρητή - Καταχωρήτη

- LDA Θ
- ADD Ρ
- STA Π

Μηχανή ενός *συσσωρευτή*

- ADD Θ, Ρ, Π

Μηχανή τριών διευθύνσεων

Κύκλος ανάκλησης - εκτέλεσης

FETCH	[ΚΔΜ] ← [ΑΠ]	Μετάφερε το περιεχόμενο του ΑΠ στον ΚΔΜ
	[ΑΠ] ← [ΑΠ] + 1	Αύξησε το περιεχόμενο του ΑΠ
	[ΚΑΜ] ← [Μ([ΚΔΜ])]	Διάβασε την εντολή από την μνήμη
	[ΚΕ] ← [ΚΑΜ]	Μετάφερε το περιεχόμενο του ΚΑΜ στον ΚΕ
	[ΜΕ] ← [ΚΕ (op-code)]	Μετάφερε το op-code από τον ΚΕ στη ΜΕ
ADD	[ΚΔΜ] ← [ΚΕ (διεύθυνση)]	Μετάφερε τη διεύθυνση του τελεστέου στον ΚΔΜ
	[ΚΑΜ] ← [Μ([ΚΔΜ])]	Διάβασε τα δεδομένα από την μνήμη
	ΑΛΜ ← [ΚΑΜ], ΑΛΜ ← [Δ0]	Εκτέλεσε την πρόσθεση
	[Δ0] ← ΑΛΜ	Μετάφερε την έξοδο της ΑΛΜ στον καταχωρητή δεδομένων

Αποφάσεις

- Συνθήκες καθορίζουν τη ροή του προγράμματος
- Προγραμματιστική δομή:
  - IF (condition) THEN statements1 ELSE statements2
- Βρόχοι επανάληψης το χρησιμοποιούν

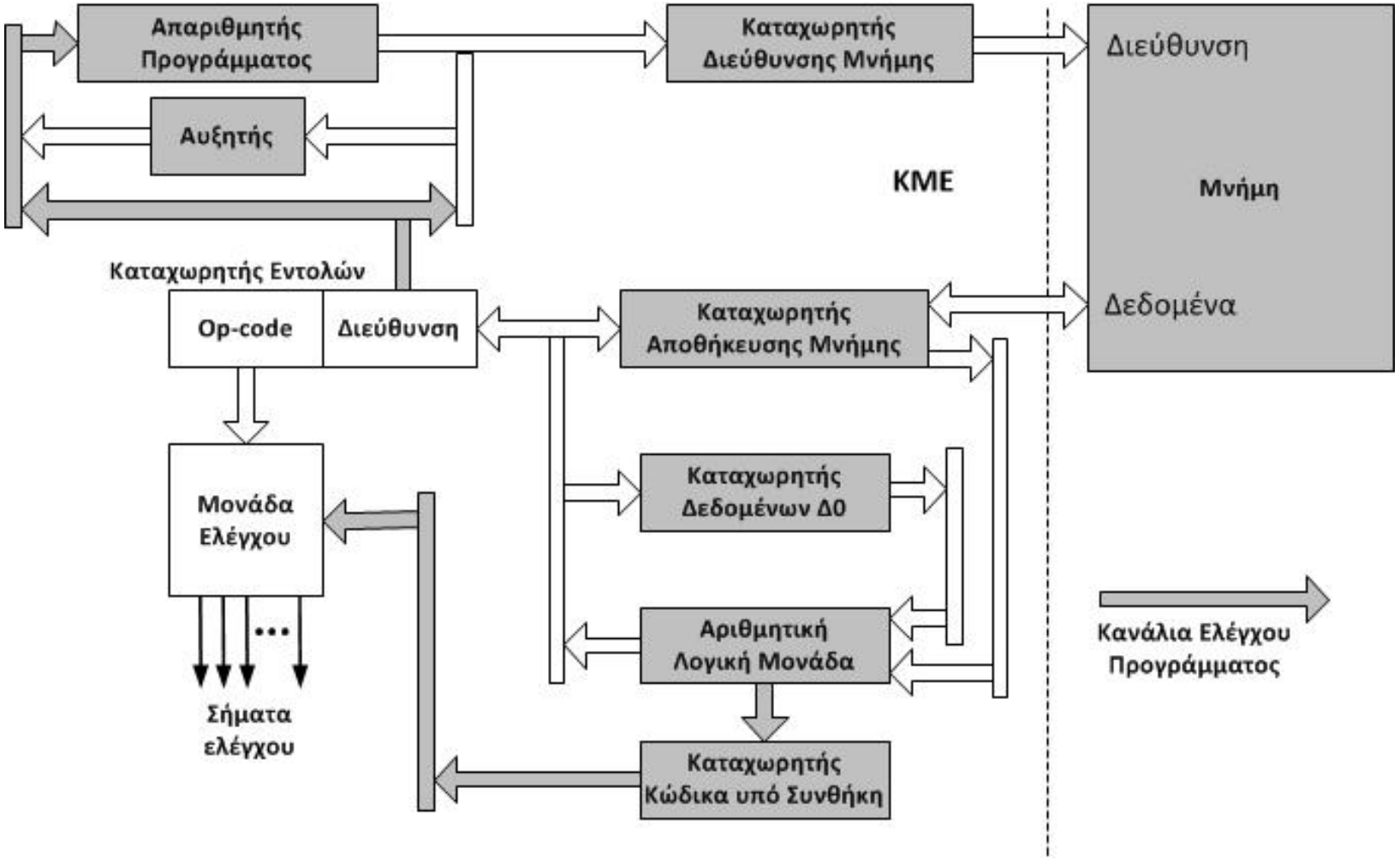
Status Register

Κ = κρατούμενο	**Τίθεται αν Κρατούμενο = 1 στην τελευταία πράξη**
Μ = μηδενικό	**Τίθεται αν αποτέλεσμα = 0 στην τελευταία πράξη**
Α = αρνητικό	**Τίθεται αν το αποτέλεσμα βγήκε αρνητικό**
Υ = υπερχείλιση	**Τίθεται αν το αποτέλεσμα είναι εκτός αναπαραστάσιμης περιοχής**

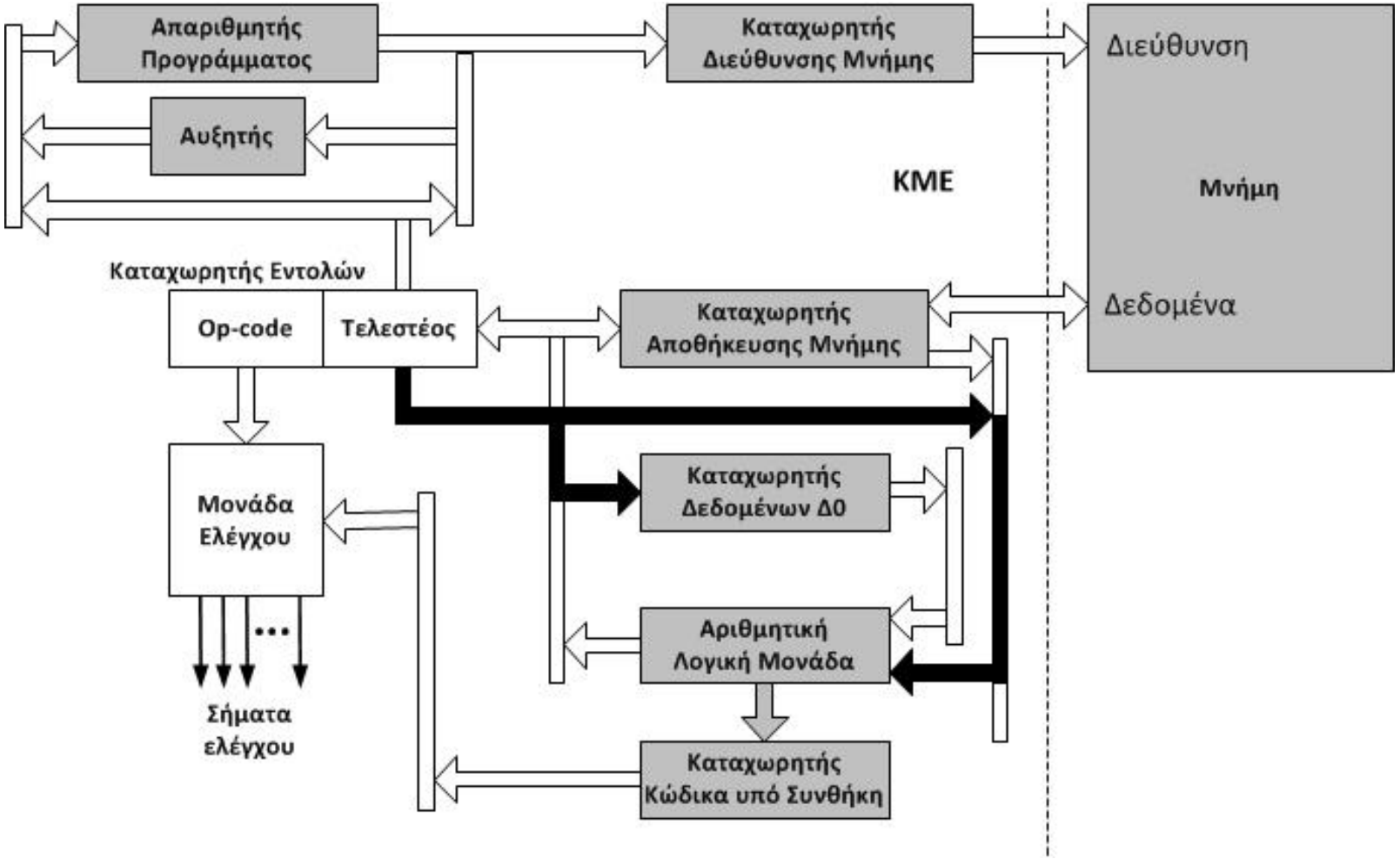
Παράδειγμα

1ος Τελεστέος		2ος Τελεστέος		Αποτέλεσμα		Δυαδικά Ψηφία του ΚΚΣ
00000011	+	00000100	=	00000111		K=0, M=0, A=0, Y=0
11111111	+	00000001	=	00000000		K=1, M=1, A=0, Y=0
01100110	+	00110010	=	10011000		K=0, M=0, A=1, Y=1
11001001	+	10100000	=	01101001		K=1, M=0, A=0, Y=1

Εκτέλεση εντολών με συνθήκη



Απευθείας τελεστέοι



Τέλος Ενότητας

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σημειώματα

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

Έχουν προηγηθεί οι κάτωθι εκδόσεις:

- Έκδοση 1.0 διαθέσιμη εδώ.

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

Εικόνες/Σχήματα/Διαγράμματα/Φωτογραφίες

Διαφάνεια 8: http://www.intel.com/pressroom/kits/core2duo/

Διαφάνεια 9: http://techreport.com/r.x/2010q1/westmere-2c-6c.jpg