Αρχιτεκτονική Υπολογιστών

Ενότητα 5: Datapath x86.

Παράδειγμα λειτουργίας υποθετικής αρχιτεκτονικής ΤΟΥ86

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.ece.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη 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.

Σκοπός ενότητας

- Η παρουσίαση της 8086 αρχιτεκτονικής.
- Η κατανόηση του τρόπου λειτουργίας του μονοπατιού δεδομένων σε έναν επεξεργαστή.
- Η παρουσίαση ενός απλοποιημένου παραδείγματος μονοπατιού δεδομένων σε έναν επεξεργαστή.

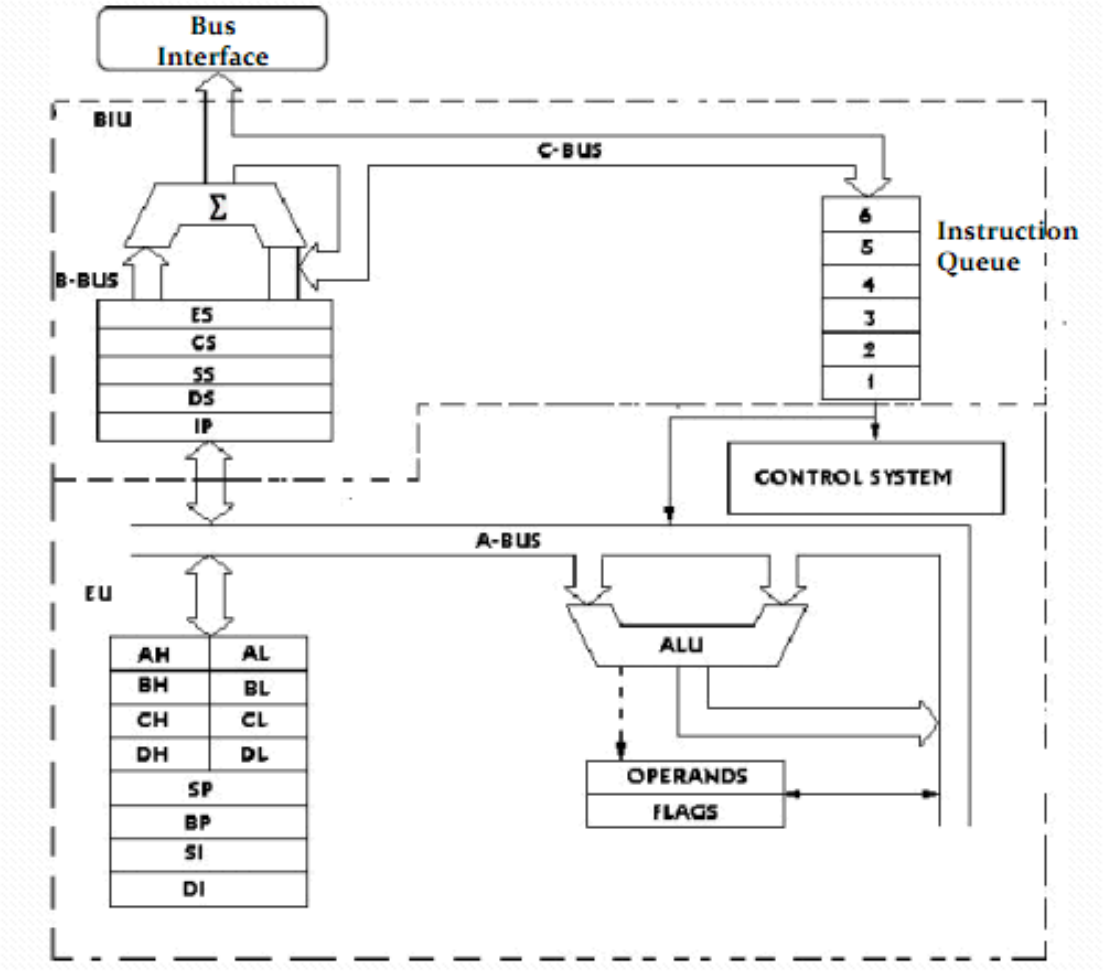
Συγκρίσεις μικρο-επεξεργαστών της Intel

Processor	Ext.DBUS Witdth (bits)	Architecture	Instructions /Clock Cycle	Max Clock Speed(MHZ)
8086	16	Separate bus & execution units; 6-byte queue	1:4	10
386	32	Read & protected mode with 32-bit memory management	1:2	33
486	32	5-stage pipeline and on-board 8K cache & floating point unit	1:1	100
Pentium	64	Two 5-stg. Pipelines with branch prediction. Redesign FPU. Separate 8K code & data cashes	2:1	200
Pentium Pro	64	Three instruction decoders with 12-stg pipeline. Out of order execution. On-bd Level 1&2 cache	2:7:1	200
Pentium II	64	Pent. Pro features with MMX technology and improved 16-bit performance	2:7:1	450
Pentium III	64	Improved Pentium ll core, now including Streaming SIMD Extensions.	2:7:1	1.400
Pentium 4	64	NetBust microarchitecture; HyperThreadingTechnology; improved branch prediction; Increased integer instruction pipeline stages	3:1	3.800

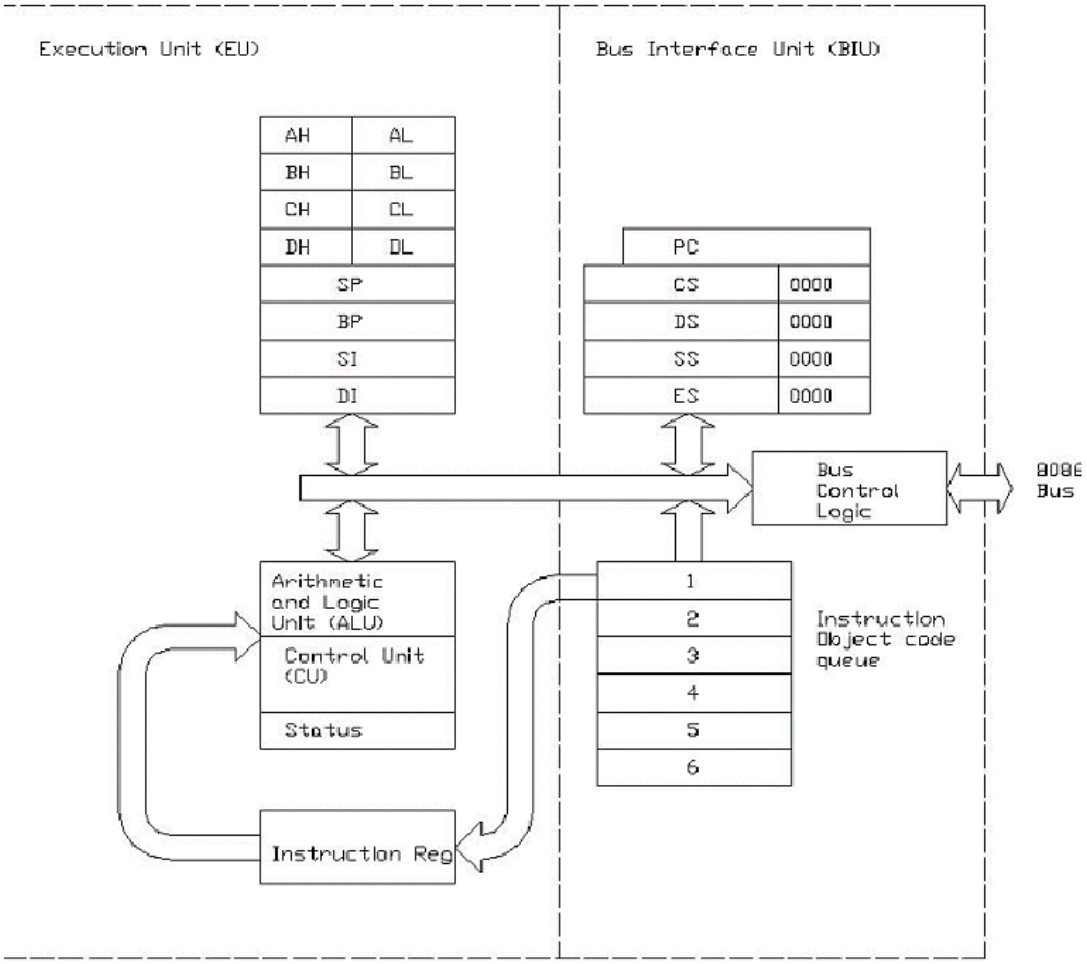
Ο επεξεργαστής 8086

- Υποστηρίζει πλήρως 16bit λειτουργίες.
- Υποστηρίζει πραγματική λειτουργία μόνο.
- Διασωληνωμένη εσωτερική αρχιτεκτονική.
- Ξεχωριστή μονάδα εκτέλεσης και μονάδα διασύνδεσης διαύλου.
- Ουρά εντολών 6-byte.
- Τμηματοποίηση με παραγωγή διευθύνσεων 16-bit έως 20-bit.
- Διευθυνσιοδοτεί 1 Mbyte (20-bit).

Το λειτουργικό διάγραμμα του 8086



Η διαδρομή δεδομένων του 8086



Η διαδρομή δεδομένων στο 8086 αποτελείται από 2 τμήματα

- Υπάρχει το Execution Unit.
- Υπάρχει το Bus Interface Unit.

Γιατί υπάρχουν 2 τμήματα;

Η ύπαρξη των δυο τμημάτων επιτρέπει τη διασωλήνωση 2 βαθμίδων. Έτσι, όσο σε μια βαθμίδα έρχεται η επόμενη εντολή (bus interface unit) στην επόμενη βαθμίδα εκτελείται η προηγούμενη εντολή. Με αυτόν τον τρόπο χρησιμοποιούνται συνεχώς και οι 2 βαθμίδες.

Η χρησιμότητα της μονάδας διασύνδεσης διαύλου (BIU)

- Χωρίς διασωλήνωση (χωρίς διαχωρισμό)- Απαιτούνται 6 κύκλοι για 3 εντολές):

TIME	OPERATION	
1	FETCH INSTRUCTION	1
2	EXECUTE INSTRUCTION	1
3	FETCH INSTRUCTION	2
4	EXECUTE INSTRUCTION	2
5	FETCH INSTRUCTION	3
6	EXECUTE INSTRUCTION	3

- Με διασωλήνωση (EU + BIU) – Απαιτούνται 6 κύκλοι για 5 εντολές:

TIME	OPERATIONS	
1	FETCH INSTRUCTION 1	
2	FETCH INSTRUCTION 2	EXECUTE INSTRUCTION 1
3	FETCH INSTRUCTION 3	EXECUTE INSTRUCTION 2
4	FETCH INSTRUCTION 4	EXECUTE INSTRUCTION 3
5	FETCH INSTRUCTION 5	EXECUTE INSTRUCTION 4

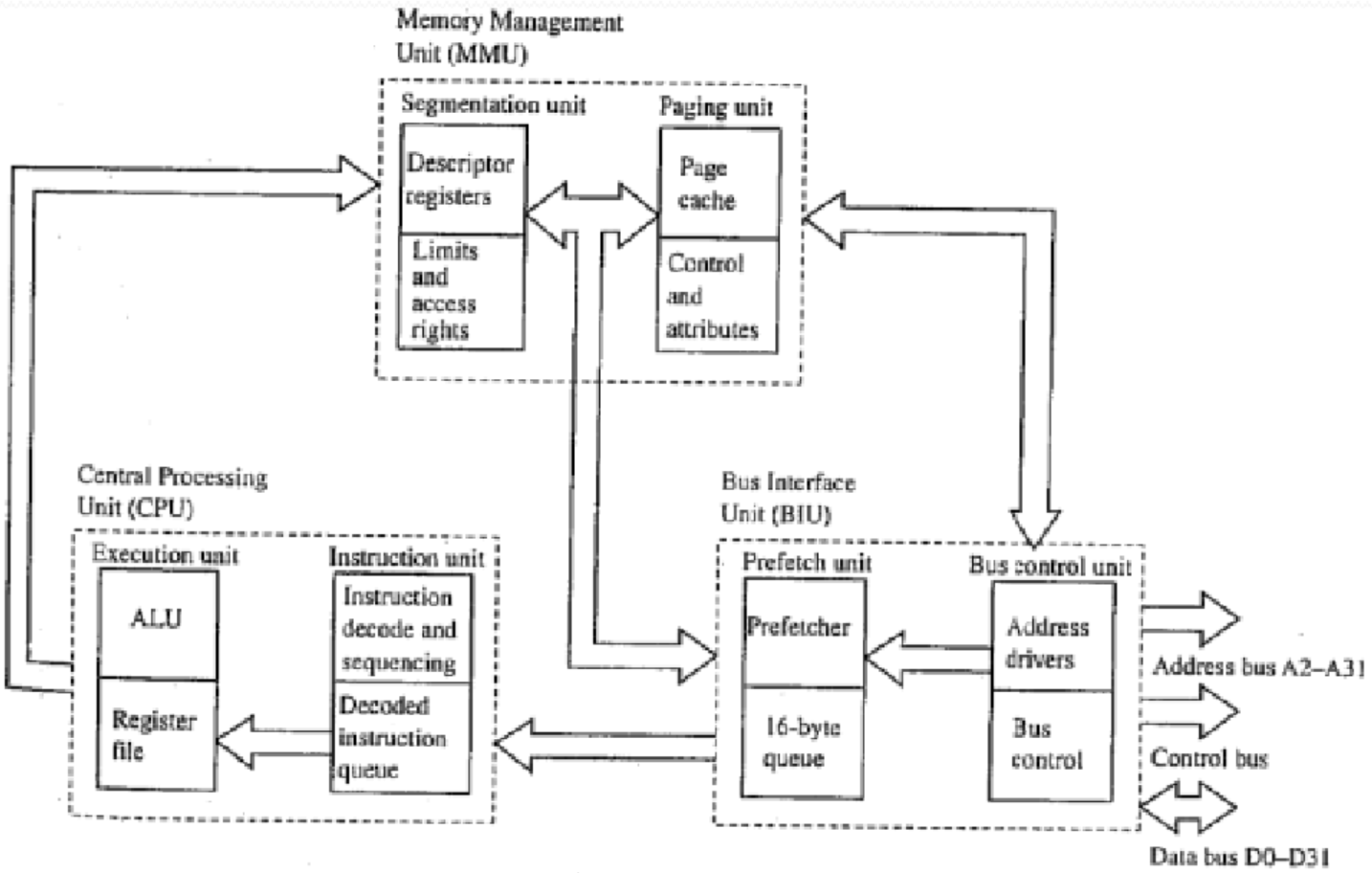
Ουρά εντολών 6 Bytes

- Ο επεξεργαστής 8086 είχε μια μνήμη για την προσωρινή αποθήκευση 6 Byte machine code.
- Ονομάζοταν Instruction Queue ή Instruction buffer.
- Η μνήμη αυτή βρίσκονταν στο τμήμα BUI.
- Αυτό οφείλονταν στο γεγονός ότι οι εντολές του 8086 ήταν από 1 Byte έως 6 Byte.
- Με αυτή την τεχνική κάθε φορά υπήρχε η επόμενη εντολή προς εκτέλεση μέσα στο IC, και δεν υπήρχε καθυστέρηση κατά την εκτέλεση λόγω πρόσβασης εκτός ολοκληρωμένου.
- Αν υπήρχε αλλαγή ροής εκτέλεσης, τότε έπρεπε να αδειάσει η ουρά εντολών και να τοποθετηθούν τα νέα 6 Byte, από το CS:IP.

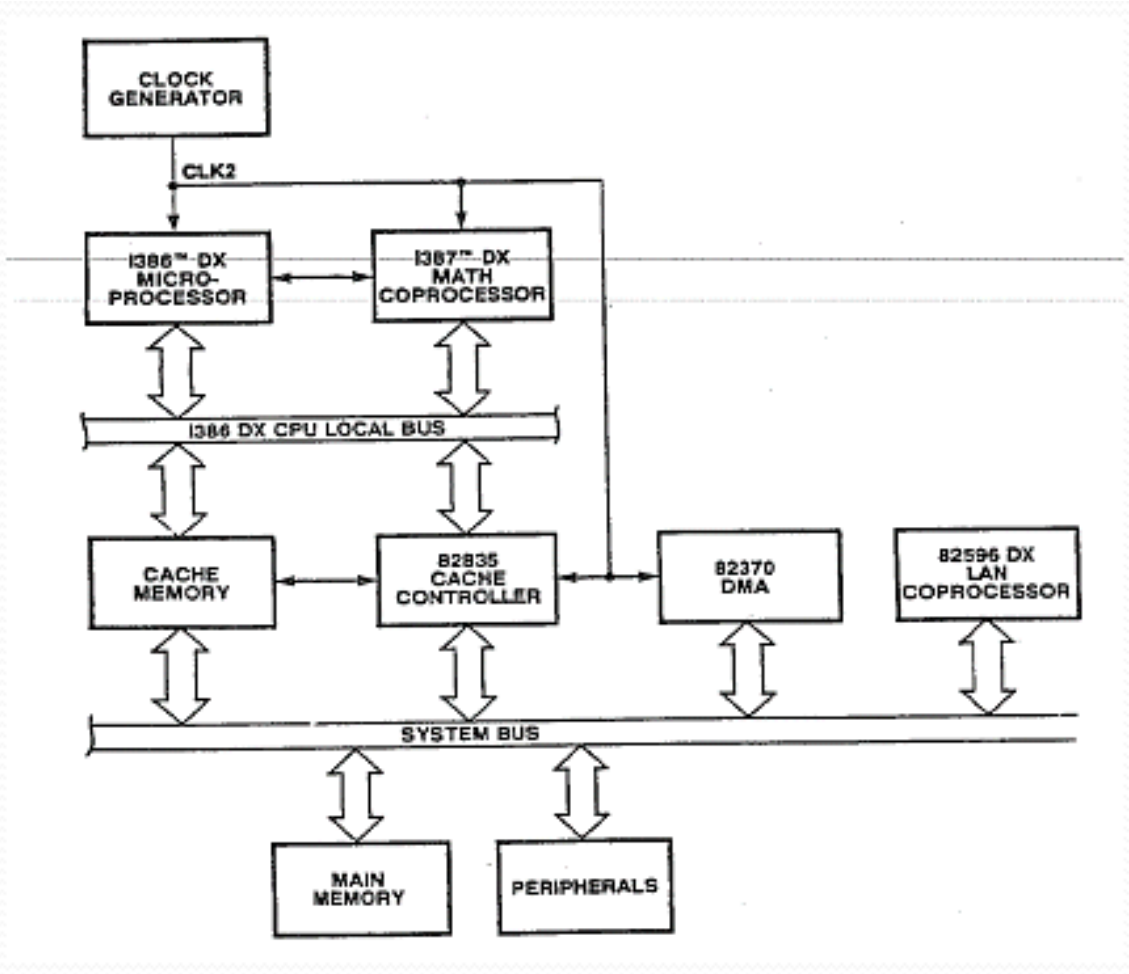
Στοιχεία της 386 αρχιτεκτονικής

- Η πρώτη 32bit αρχιτεκτονική.
- Ουρά εντολών 16-byte.
- Αρχιτεκτονική IA-32.
- Τρεις καταστάσεις λειτουργίας: real, protected and virtual.
- Η κατάσταση λειτουργίας 8086 virtual, επέτρεπε την εκτέλεση ενός ή παραπάνω προγράμματος σε προστατευμένο περιβάλλον.
- Χρησιμοποιεί 32-bit flat memory model (δε χρησιμοποιεί καταχωρητές τμημάτων, οπότε δεν υπάρχει περιορισμός μεγέθους τμήματος).
- Διευθυνσιοδότηση έως 4 GB μνήμης.
- Χρησιμοποιεί μονάδα μετάφρασης σελίδων (εικονικές διευθύνσεις σε φυσικές διευθύνσεις μνήμης).

Το λειτουργικό διάγραμμα του 386



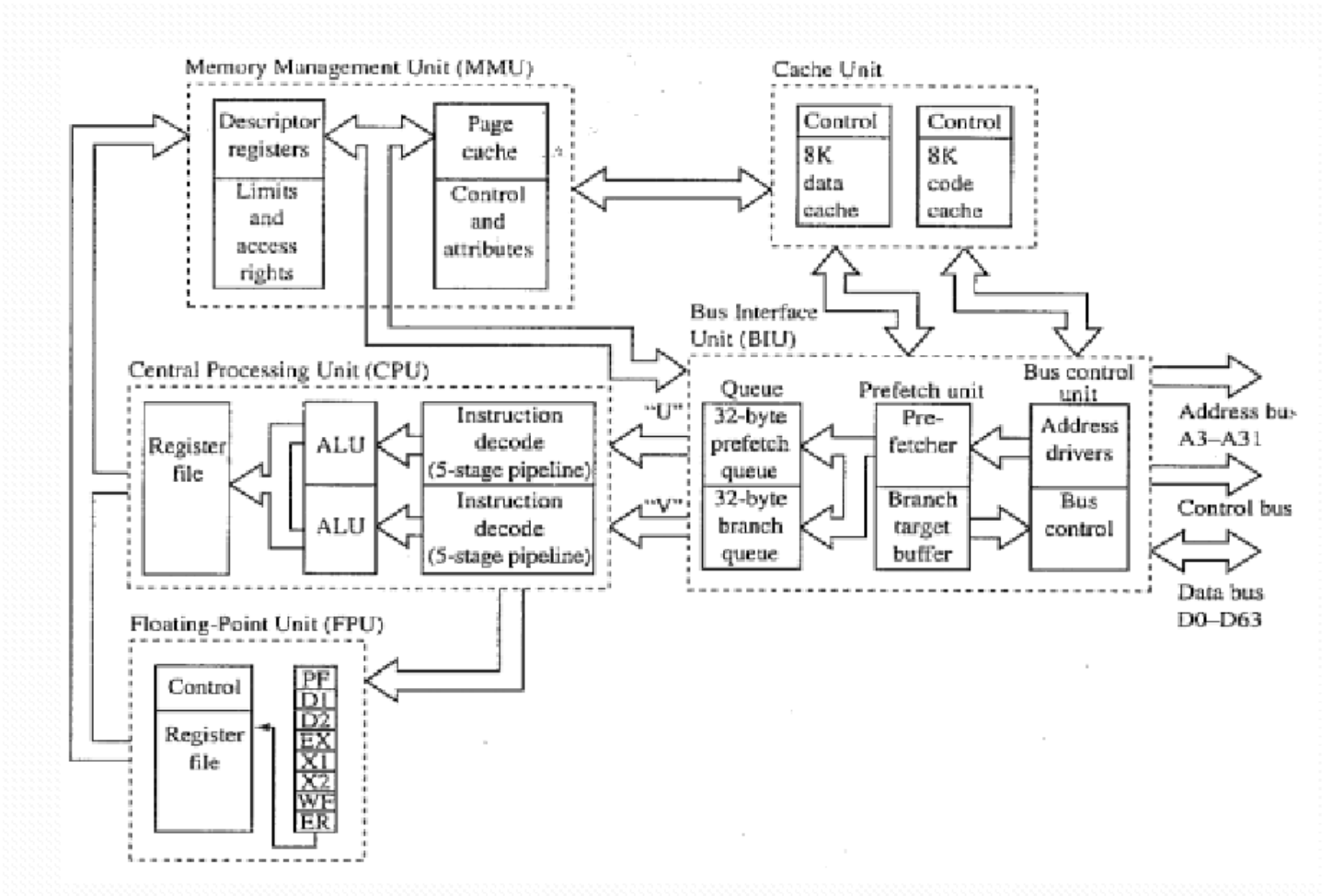
Αρχιτεκτονική 386 μαζί με περιφερειακά



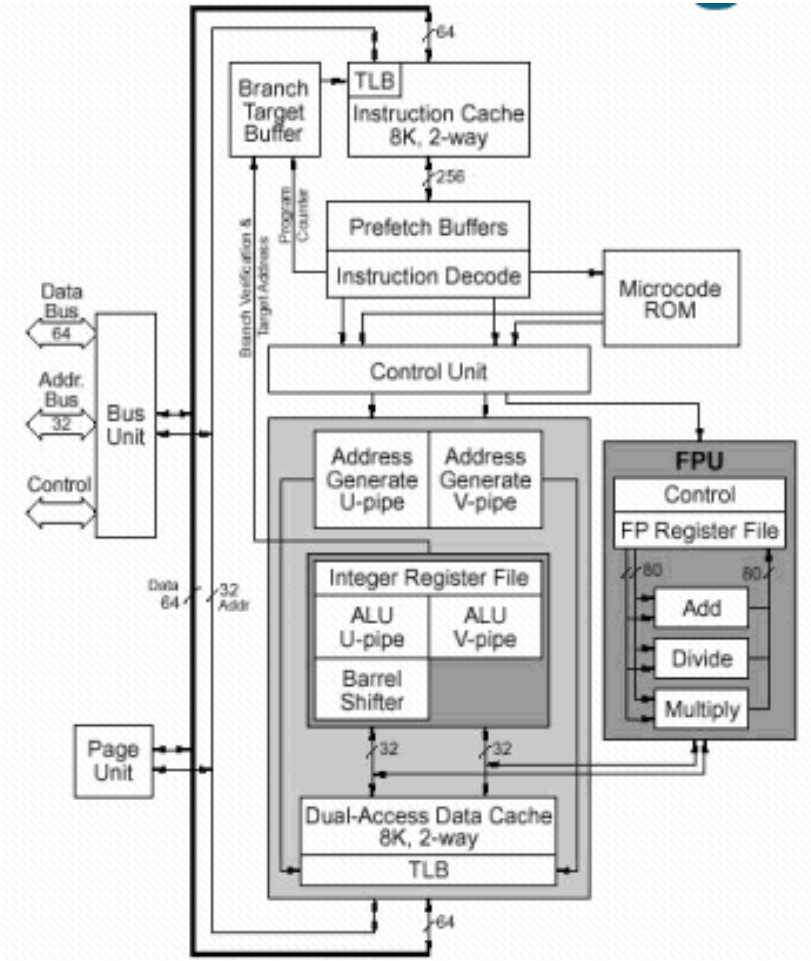
Η αρχιτεκτονική Pentium (586)

- Τρεις διαδρομές δεδομένων: δυο ακέραιες μονάδες και μια μονάδα πραγματικών αριθμών.
- 64-bit δίαυλος δεδομένων με υποστήριξη μεταφοράς ριπής (burst), για γέμισμα κρυφής μνήμης.
- Δυναμική πρόγνωση διακλαδώσεων.
- Ξεχωριστές κρυφές μνήμες 8KB κώδικα & δεδομένων.
- Μονάδα διαχείρισης μνήμης (Memory management unit-MMU).
- Στοιχεία διαχείρισης ενέργειας πάνω στο IC.
- Παρακολούθηση απόδοσης για βελτιστοποίηση κώδικα.
- Υποστήριξη αποσφαλμάτωσης με υλικό, μέσω ειδικών pin.

Το λειτουργικό διάγραμμα του Pentium



Το block διάγραμμα του Pentium



Μια απλοποιημένη διαδρομή δεδομένων (datapath).

Παράδειγμα

- Στις παρακάτω διαφάνειες θα αναλύσουμε τη λειτουργία μιας απλοποιημένης αρχιτεκτονικής επεξεργαστή.
- Όλοι οι προγραμματιζόμενοι επεξεργαστές βασίζονται σε αυτό το μοντέλο, αλλά ασφαλώς έχουν πολλές παραπάνω λειτουργίες.
- Οι επόμενες διαφάνειες θα ασχοληθούν:
- Παρουσίαση της υποθετικής αρχιτεκτονικής TOY86.
- ISA της αρχιτεκτονικής TOY86.
- Παράδειγμα λειτουργίας TOY86.

Όλοι οι επεξεργαστές εκτελούν ένα αέναο βρόχο

- 1/7 (**Fetch**) Μεταφορά στον επεξεργαστή της εντολής (από την εξωτερική μνήμη).
- 2/7 Αύξηση του μετρητή προγράμματος.
- 3/7 (**Decode**) Αποκωδικοποίηση της εντολής.
- 4/7 Μεταφορά των παραμέτρων.
- 5/7 (**Execute**) Εκτέλεση της λειτουργίας που ορίζεται.
- 6/7 (**Write**) Αποθήκευση των αποτελεσμάτων.
- 7/7 Επανάληψη από την αρχή.

1/7 (Fetch) Μεταφορά στον επεξεργαστή της εντολής (από την εξωτερική μνήμη)

Ερωτήματα που συνδέονται με αυτό το στάδιο:

- Τι είναι μια εντολή; (σειρά από Byte).
- Που είναι αποθηκευμένη μια εντολή; (off-chip memory).
- Γιατί πρέπει να μετακινηθεί μέσα στον επεξεργαστή; (δε μπορεί να εκτελεστεί μια εντολή από τη μνήμη, πρέπει να αποκωδικοποιηθεί μέσα στον επεξεργαστή).
- Πως ο επεξεργαστής γνωρίζει κάθε φορά πια εντολή να φέρει; (χρησιμοποιεί ειδικό καταχωρητή για μετρητή προγράμματος).
- Που τοποθετείται η εντολή μέσα στον επεξεργαστή; (σε ειδικό καταχωρητή εντολών).

2/7 Αύξηση του μετρητή προγράμματος

Ερωτήματα που συνδέονται με αυτό το στάδιο:

- Τι είναι ο μετρητής προγράμματος; (ειδικός καταχωρητής που κάθε φορά δείχνει στην επόμενη προς εκτέλεση εντολή).
- Τι μετράει ο μετρητής προγράμματος; (διευθύνσεις μνήμης).
- Πόσο αυξάνεται; (αυξάνει κάθε φορά κατά μέγεθος ίσο με την εντολή που έχει εισαχθεί στο επόμενο στάδιο, για να δείξει στην επόμενη εντολή).
- Μετά την αύξηση, η νέα τιμή που δείχνει; (στην αμέσως επόμενη εντολή προς εκτέλεση).
- Ο μετρητής μόνο αυξάνει; (όχι, αν υπάρχει αλλαγή ροής εκτέλεσης (εντολές **j**??), ο μετρητής τροποποιείται για να δείχνει στη νέα εντολή, που βρίσκεται είτε πιο πριν είτε μετά).

3/7 (Decode)
Αποκωδικοποίηση της εντολής

Ερωτήματα που συνδέονται με αυτό το στάδιο:

- Γιατί πρέπει να αποκωδικοποιηθεί η εντολή; (η εντολή είναι μια σειρά από Byte, που αντιστοιχούν σε σαφώς ορισμένες λειτουργίες, αντιστοίχηση 1-προς-1. Απαιτείται η αποκωδικοποίηση για να βρεθούν οι λειτουργίες).
- Πως αποκωδικοποιείται; (Η θέση κάθε bit μέσα στο machine code, έχει οριστεί από την εταιρία κατασκευής στο τι αντιστοιχεί. Με κατάλληλα κυκλώματα της μονάδας ελέγχου, εξετάζονται όλα τα bit και δημιουργούνται τα κατάλληλα σήματα ελέγχου προς κάθε μονάδα μέσα στον επεξεργαστή).
- Πως δημιουργούνται τα σήματα ελέγχου; (τα σήματα μπορούν να δημιουργηθούν είτε με μικροπρόγραμμα -microprogrammed-, είτε με καλωδιομένο τρόπο -hardwired-).

4/7 Μεταφορά των παραμέτρων

Ερωτήματα που συνδέονται με αυτό το στάδιο:

- Τι είναι παράμετροι; (είναι οι αριθμοί που συνοδεύουν κάποιες εντολές. Υπάρχουν εντολές που δε δέχονται παραμέτρους, όπως και εντολές που δέχονται παραμέτρους).
- Τι σημαίνει η μεταφορά των παραμέτρων; (είναι η μεταφορά από την εξωτερική μνήμη μέσα στον επεξεργαστή σε κατάλληλους καταχωρητές).
- Σε τι διαφέρει αυτή η μεταφορά από το βήμα 1/7; (στο πρώτο βήμα μεταφέρεται η εντολή, σε αυτό το βήμα μεταφέρονται οι παράμετροι. Οι παράμετροι είναι σε άλλο σημείο της μνήμης και όχι μέσα στην ίδια την εντολή).
- Πόσοι είναι οι παράμετροι; (αναλόγως την εντολή, μπορεί να μην έχει καμία παράμετρο ή μπορεί να έχει 1 ή 2).

5/7 (Execute) Εκτέλεση της λειτουργίας που ορίζεται

Ερωτήματα που συνδέονται με αυτό το στάδιο:

- Πως γίνεται η εκτέλεση; (κατά την αποκωδικοποίηση της εντολής στο βήμα 3, δημιουργούνται κατάλληλα σήματα ελέγχου προς όλες τις μονάδες, ακόμη και προς αυτές που δε θα χρησιμοποιηθούν σε αυτή την εντολή. Με βάση τα σήματα αυτά κάποια μονάδα θα χρησιμοποιήσει ή όχι τα δεδομένα, και θα τροποποιήσει ή όχι συγκεκριμένους καταχωρητές).
- Ποιο τμήμα του επεξεργαστή χειρίζεται την εκτέλεση; (η διαδρομή δεδομένων -datapath-).

6/7 (Write) Αποθήκευση των αποτελεσμάτων

Ερωτήματα που συνδέονται με αυτό το στάδιο:

- Ποια αποτελέσματα; (οι περισσότερες εντολές δημιουργούν αποτελέσματα, ακόμη και αν δεν είναι άμεσα αντιληπτό. Τα αποτελέσματα είναι τιμές που γράφονται σε καταχωρητές)
- Που αποθηκεύονται τα αποτελέσματα; (αρχικά αποθηκεύονται σε καταχωρητές. Αν στην εντολή υπάρχει οδηγία για αποθήκευση των αποτελεσμάτων στη μνήμη ή κάπου αλλού--όπως I/O-- τότε υλοποιείται η μεταφορά σε αυτό το στάδιο.

7/7 Επανάληψη από την αρχή

Ερωτήματα που συνδέονται με αυτό το στάδιο:

- Τι επαναλαμβάνεται; (επαναλαμβάνεται συνεχώς η λειτουργία των σταδίων 1 έως 6).
- Από ποιο σημείο ξεκινάει η επανάληψη; (από το πρώτο σημείο της μεταφοράς στον επεξεργαστή της πρώτης εντολής).
- Είναι ένας αέναος βρόχος; (ναι, η διαδικασία επαναλαμβάνεται, ακόμη και αν δεν υπάρχουν επόμενες εντολές προς εκτέλεση, όπου σε αυτή την περίπτωση ο επεξεργαστής εκτελεί μια nop (no operation)).

Κωδικοποίηση εντολών

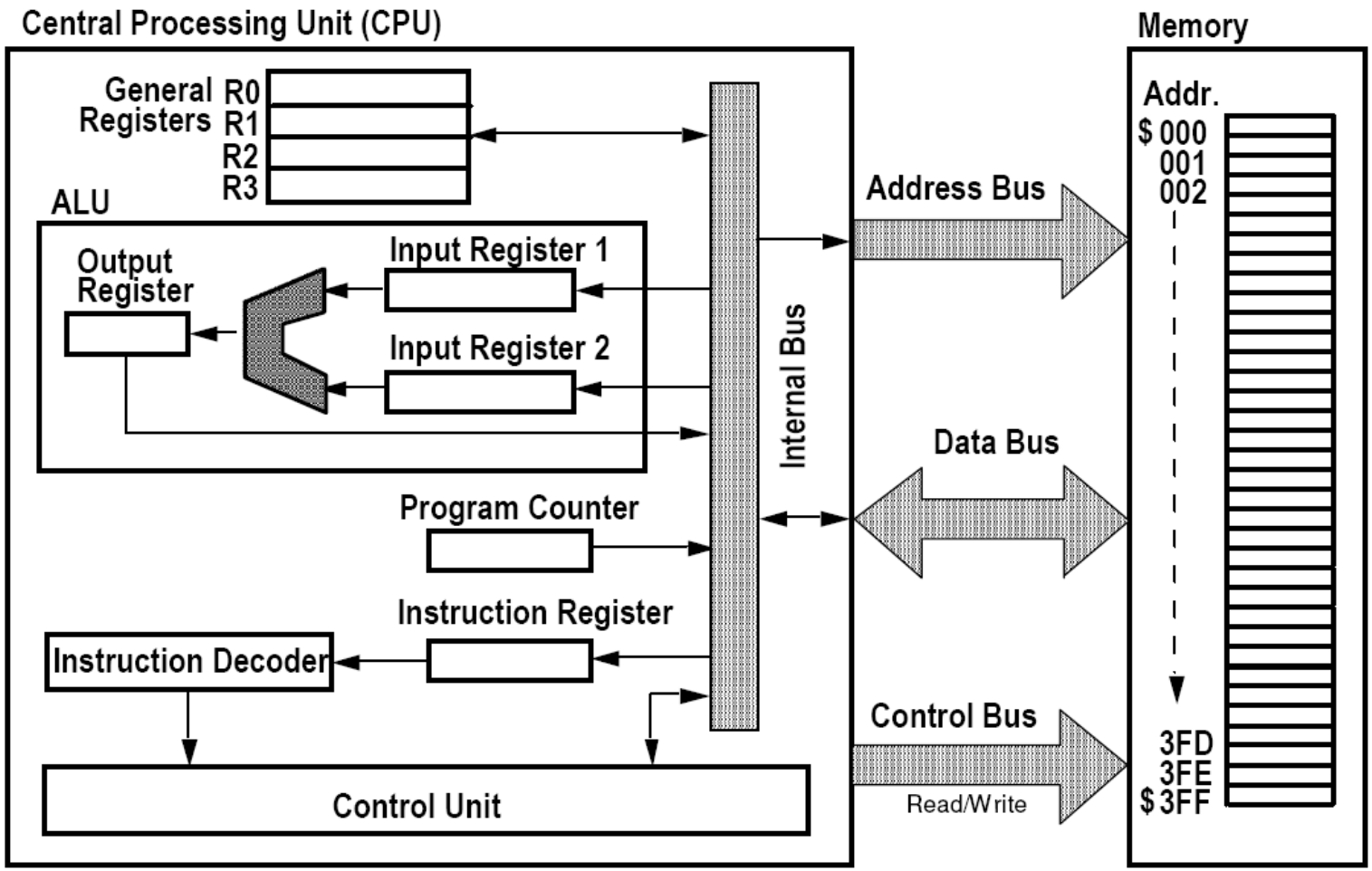
- Οι εντολές κωδικοποιούνται σε αριθμούς στο δυαδικό σύστημα. Κάθε εντολή χωρίζεται σε **δύο τουλάχιστον τμήματα**:
- **Opcode** (operation code): Προσδιορίζει τη λειτουργία.
- **Operands**: Προσδιορίζει τις παραμέτρους.

Παρουσίαση της υποθετικής αρχιτεκτονικής ΤΟΥ86

Η υποθετική αρχιτεκτονική ΤΟΥ86

- Υποθέτουμε ότι έχουμε την παρακάτω αρχιτεκτονική:
  - 4 γενικοί καταχωρητές (16bit): AX,BX,CX,DX (απαιτούνται 2 bit).
  - 16 εντολές (MOV,ADD,SUB...) (απαιτούνται 4 bit).
  - Αναπαράσταση ακεραίων με συμπλήρωμα ως προς 2
  - Η μνήμη διευθυνσιοδοτείται σε λέξεις.
  - Η κάθε λέξη έχει 2 Byte.
  - Υπάρχουν 1024 λέξεις (των 16bit) (απαιτούνται 10 bit).
  - Όλες οι εντολές είναι 16 bit σταθερές (4bit opcode, 2bit register, 10bit address).
  - Μετά από reset/power-on PC=080h.

Λειτουργικό Διάγραμμα της αρχιτεκτονικής ΤΟΥ86 (1/2)



Καταχωρητές

Διακρίνονται οι εξής καταχωρητές:

- **PC (program counter)**: καταχωρητής που φέρει τη διεύθυνση της επόμενης προς εκτέλεση εντολής. Είναι τόσα bit όσο το address bus. Στη ΤΟΥ86 είναι 10bit.
- **IR (instruction register)**: καταχωρητής στον οποίο μεταφέρεται από την εξωτερική μνήμη η επόμενη εντολή. Είναι τόσα bit όσα τα bit της μεγαλύτερης εντολής. Στη ΤΟΥ86 είναι 16bit.
- **ALU Input Registers 1 & 2**: καταχωρητές που φέρουν τις τιμές που θα μεταφερθούν στην ALU.
- **ALU Output Register**: καταχωρητής που φέρει την τιμή που υπολογίζεται από την ALU.
- **General Registers**: καταχωρητές γενικού σκοπού, που χρησιμοποιούνται από τον προγραμματιστή.
- **MAR (memory address register)**: καταχωρητής που φέρει τη διεύθυνση μνήμης που θα τοποθετηθεί στο δίαυλο διευθύνσεων.
- **MDR (memory data register)** : καταχωρητής που φέρει τα δεδομένα που θα τοποθετηθούν στο δίαυλο δεδομένων για τη μνήμη ή φέρει τα δεδομένα που μόλις έχουν μεταφερθεί από τη μνήμη.

Λειτουργικές μονάδες της ΤΟΥ86

Διακρίνονται οι εξής λειτουργικές μονάδες:

- **Control Unit (Μονάδα ελέγχου)**: συγχρονίζει και κατευθύνει όλες τις άλλες μονάδες με σήματα ελέγχου. Συνδέεται με κάθε μονάδα και έχει ένα πολύ εξελιγμένο κύκλωμα χρονισμού.
- **ALU (μονάδα ακέραιων αριθμητικών και λογικών πράξεων)**: Χρησιμοποιείται για τις πράξεις: πρόσθεση, αφαίρεση, σύγκριση, AND, OR, NOT, κ.α.
- **Busses (δίαυλοι)**: στοιχεία επικοινωνίας που μεταφέρουν συνήθως περισσότερα από 1 bit παράλληλα. Είναι 3 τύποι: δίαυλοι διευθύνσεων ή δεδομένων ή ελέγχου.

ISA της αρχιτεκτονικής TOY86

Οι εντολές που υποστηρίζονται (1/2)

- Υποθέτουμε ότι υποστηρίζονται μόνο οι παρακάτω εντολές:
  - Μετακίνησης από τη μνήμη σε καταχωρητή (LOAD).
  - Μετακίνησης από καταχωρητή σε μνήμη (STORE).
  - Πρόσθεσης.
  - Αφαίρεσης.
- Υπάρχει πλήθος αχρησιμοποίητων opcodes.

Οι εντολές που υποστηρίζονται (2/2)

- MOV <Register> , <Memory Address> (LOAD)

<Register> := Memory [<Memory Address>]

- MOV <Memory Address >, <Register> (STORE)

Memory [<Memory Address>] := <Register>

- ADD <Register> , <Memory Address>

<Register> := <Register> + Memory [<Memory Address>]

- SUB <Register> , <Memory Address>

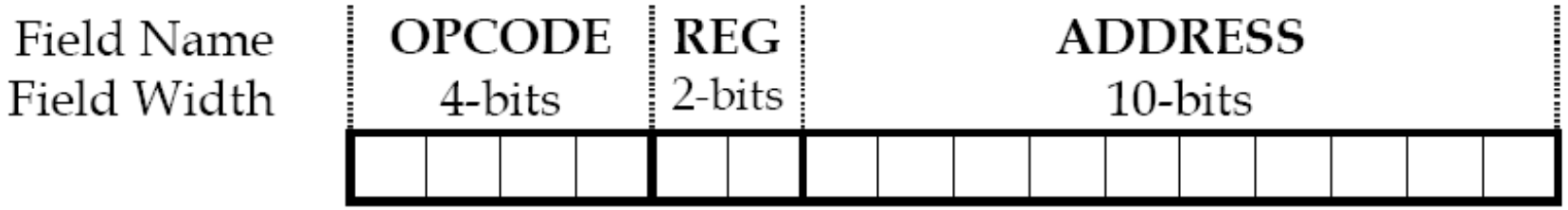
<Register> := <Register> - Memory [<Memory Address>]

Κωδικοποίηση εντολών ΤΟΥ86 (1/2)

- Ενδεικτικοί Opcodes (απαιτούν 4bit κωδικοποίηση, μόνο 4 εντολές χρησιμοποιούνται από τις 16).
  - MOV **0001** (LOAD)
  - MOV **0010** (STORE)
  - ADD **0011**
  - SUB **0100**
- Καταχωρητές (απαιτούν 2bit κωδικοποίηση).
  - AX **00**
  - BX **01**
  - CX **10**
  - DX **11**
- 10bit διευθύνσεις μνήμης.

Κωδικοποίηση εντολών ΤΟΥ86 (2/2)

- Οι εντολές του ΤΟΥ86 είναι 16bit:



- Διακρίνονται τα 3 τμήματα που χωρίζονται τα 16bit.

Παράδειγμα αναπαράστασης προγράμματος

- Υποθέτουμε ότι το πρόγραμμά μας σε υψηλού επιπέδου γλώσσα είναι το εξής:

  B=6;

  C=9;

  A=B+C;

- Κάθε πολύπλοκη λειτουργία που εκτελείται σε έναν επεξεργαστή, αναλύεται και απλοποιείται σε μια σειρά εντολών συμβολικής γλώσσας.
- Για κάθε εντολή συμβολικής γλώσσας, δημιουργείται ο αντίστοιχος κώδικας μηχανής (machine code).
- Η μετάφραση από μια υψηλού επιπέδου γλώσσα σε γλώσσα μηχανής, επιτυγχάνεται με το **compiler**.

Παράδειγμα: Μετάφραση σε συμβολική γλώσσα

- Ο compiler διαβάζει μια προς μια τις εντολές της υψηλής γλώσσας, και για κάθε μια από αυτές δημιουργεί μια ή παραπάνω εντολές συμβολικής γλώσσας.
- Ο compiler αποφασίζει σε ποιες διευθύνσεις θα τοποθετηθούν τα δεδομένα και σε ποιες οι εντολές.
- Αναλόγως της αρχιτεκτονικής και του compiler, οι διευθύνσεις είναι σχετικές ως προς την αρχή του προγράμματος (δηλαδή ορίζονται ως μετατόπιση), ή απόλυτες (δηλαδή, ορίζονται πάντα από την αρχή).

Παράδειγμα λειτουργία του compile – Τοποθέτηση Δεδομένων στη μνήμη

- Η πρώτη εντολή (**B=6**), σημαίνει ότι σε μια θέση μνήμης, θα αποθηκευτεί η τιμή 6.
  - Έστω ότι ο compiler τοποθετεί την τιμή 6 στη διεύθυνση μνήμης 201h.
- Η δεύτερη εντολή (**C=9**), σημαίνει ότι σε μια θέση μνήμης, θα αποθηκευτεί η τιμή 9.
  - Έστω ότι ο compiler τοποθετεί την τιμή 9 στη διεύθυνση μνήμης 202h.
- Η τρίτη εντολή (**A=B+C**), σημαίνει ότι θα χρησιμοποιηθεί η πράξη της πρόσθεσης, και ότι η τιμή που θα υπολογιστεί, θα τοποθετηθεί σε μια θέση μνήμης.
  - Έστω ότι ο compiler τοποθετεί το αποτέλεσμα της άθροισης στη διεύθυνση μνήμης 200h.

Παράδειγμα λειτουργία του compile – Δημιουργία συμβολικών εντολών

- Ο compiler γνωρίζει το ISA του επεξεργαστή στόχου, καθώς και πως να μεταφράζει εντολές υψηλού επιπέδου σε εντολές ISA.
- Στην περίπτωσή μας (TOY86) γνωρίζει ότι η πρόσθεση μπορεί να γίνει με δυο τρόπους:
  - **ADD register1, register2**

  και
  - **ADD register1, [memory_address]**
- Σε κάθε περίπτωση θα πρέπει:
  - Να μεταφερθούν οι αριθμητικές τιμές από την εξωτερική μνήμη σε καταχωρητές.
  - Να γίνει η πράξη της πρόσθεσης.
  - Να μεταφερθεί το αποτέλεσμα πίσω στην εξωτερική μνήμη.

Παράδειγμα λειτουργία του compile – Διαφορετικές υλοποιήσεις της ADD

- Κώδικας για την περίπτωση ADD register1,register2:
  MOV CX,[201h]
  MOV BX,[202h]
  ADD CX,BX
  MOV [200h],CX
- Κώδικας για την περίπτωση ADD register1, [memory_address]:
  MOV CX,[201h]
  ADD CX,[202h]
  MOV [200h],CX
- Η τελευταία υλοποίηση είναι καλύτερη (και αυτή που επιλέγεται) γιατί:
  - Χρησιμοποιείται μόνο ένας καταχωρητής.
  - Χρησιμοποιείται μια εντολή λιγότερη.
- Και στις 2 υλοποιήσεις ο αριθμός προσβάσεων στην εξωτερική μνήμη είναι ο ίδιος.

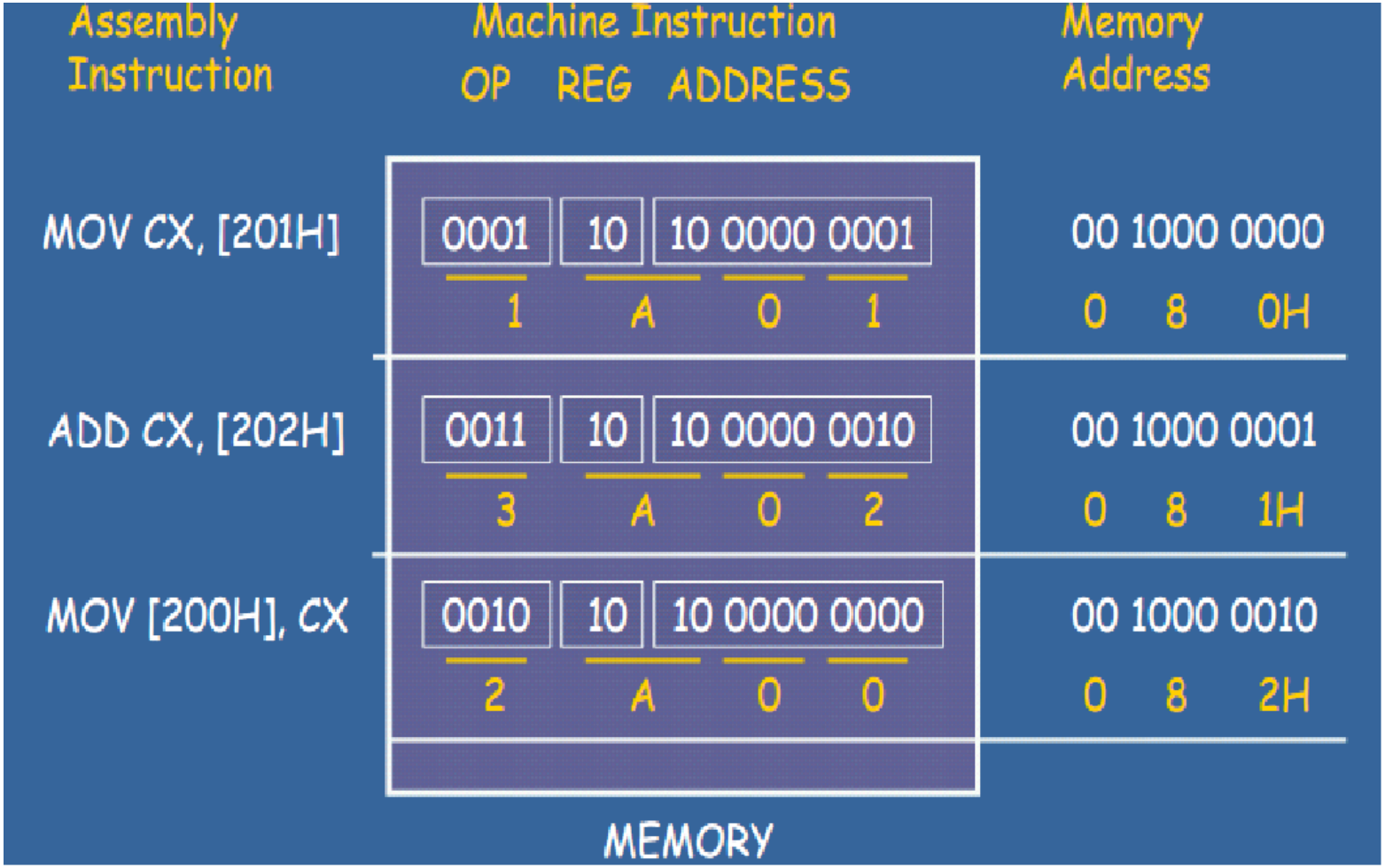
Ακολουθεί ο assembler

- Μετά την παραγωγή του ενδιάμεσου αρχείου με τις συμβολικές εντολές, ακολουθεί ο assembler.
- Ο assembler, δέχεται ως είσοδο εντολές assembly και δημιουργεί τον κώδικα μηχανής (machine code).
- Επίσης, εκτός από τον κώδικα μηχανής, τοποθετείται και κάποιο κομμάτι κώδικα που εξαρτάται από το λειτουργικό σύστημα, προκειμένου να μπορεί να εκτελεστεί ως αυτόνομη εφαρμογή (στοιχεία εκτελέσιμου, ιδιότητες, απαιτήσεις σε μνήμη, κ.ο.κ.) --**εμάς δε θα μας απασχολήσει αυτό το κομμάτι κώδικα**.

Παράδειγμα λειτουργίας του assembler - Τμήμα κώδικα

- Θέλουμε να κωδικοποιήσουμε τις παρακάτω 3 εντολές σε Machine Code για ΤΟΥ86:
  - **MOV CX,[201h]**
  - **ADD CX,[202h]**
  - **MOV [200H],CX**
- Οι εντολές αυτές θα τις τοποθετήσουμε στη διεύθυνση μνήμης που ξεκινάει ο επεξεργαστής να εκτελεί τις εντολές (τιμή που έχει ο PC όταν αρχικοποιείται ο επεξεργαστής).
  - Η πρώτη εντολή θα τοποθετηθεί στη **080H**.
  - Η δεύτερη εντολή θα τοποθετηθεί στη 081H.
  - Η τρίτη εντολή θα τοποθετηθεί στη 082H.

Παράδειγμα Assembly –Τμήμα κώδικα (1/2)



Παράδειγμα Assembly –Τμήμα κώδικα (2/2)

- Σύμφωνα με την κωδικοποίηση λοιπόν έχουμε:
  - MOV CX,[201h]

  1A01H (στη θέση μνήμης **080H** ή **00 1000 0000**)
  - ADD CX,[202h]

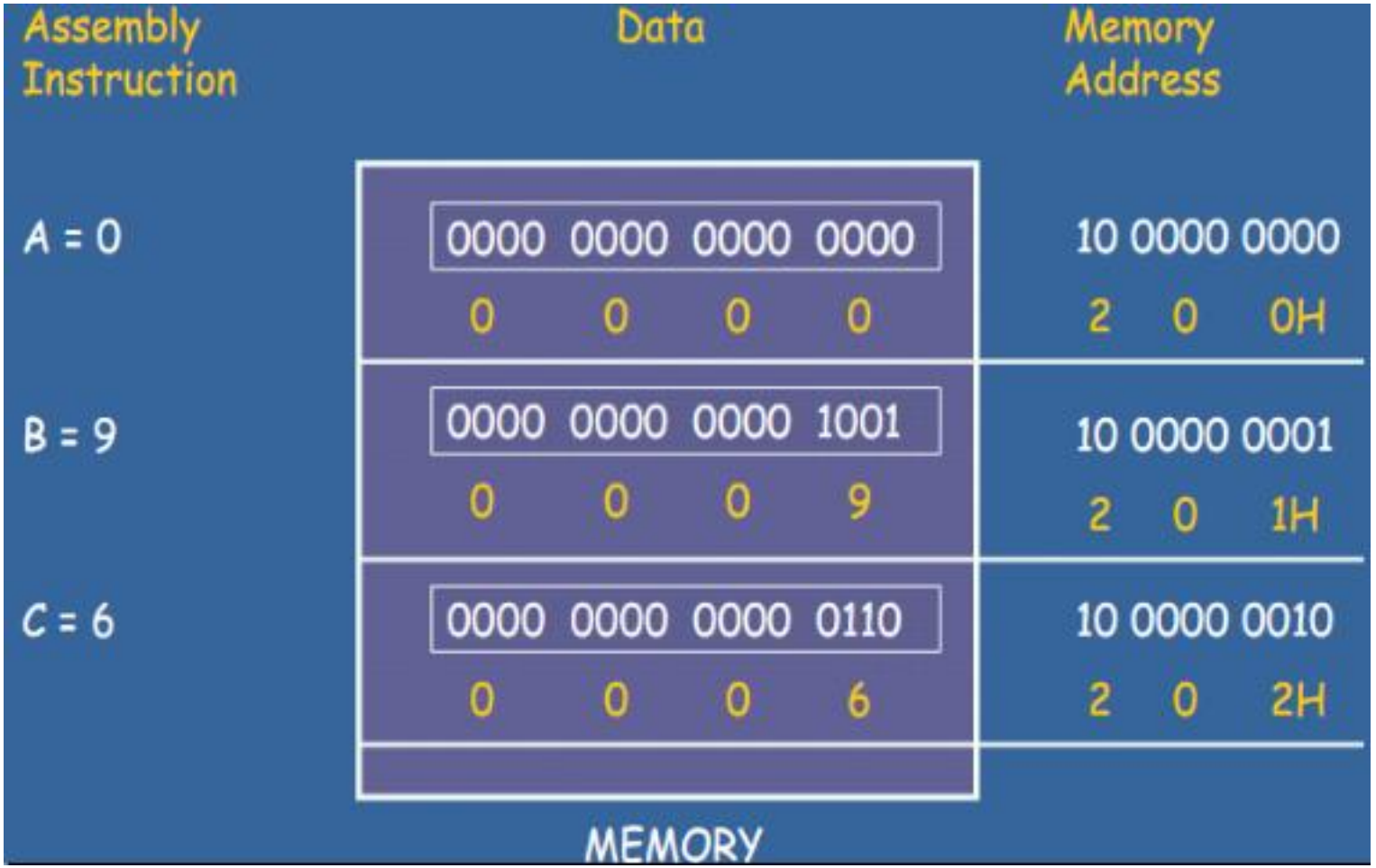
  3A02H (στη θέση μνήμης **081H** ή **00 1000 0001**)
  - MOV [200H],CX

  2A00H (στη θέση μνήμης **082H** ή **00 1000 0010**)

Παράδειγμα Assembly – Τμήμα Δεδομένων

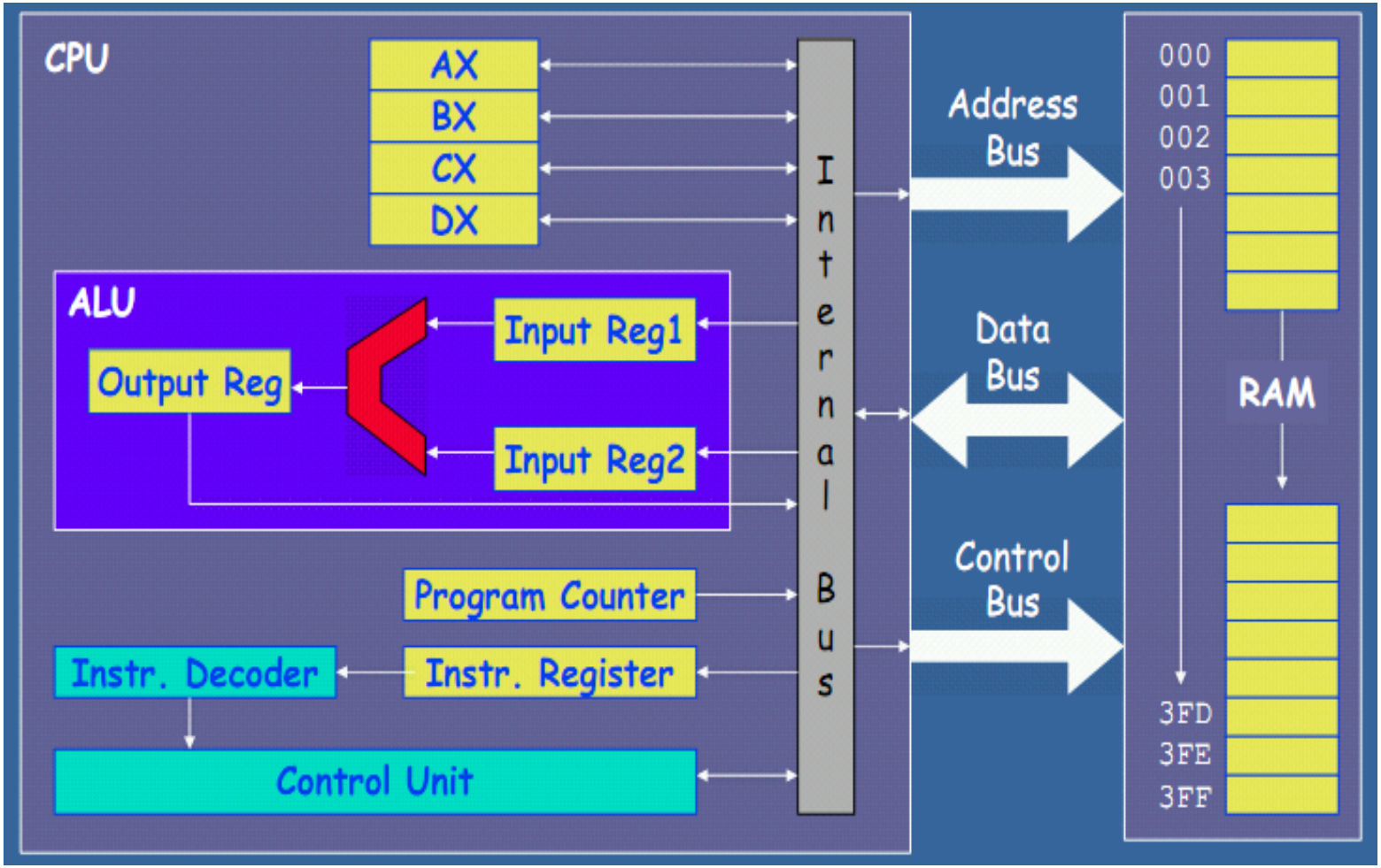
- Στο τμήμα δεδομένων έχουν οριστεί οι τιμές:
  - **A dw 0** (A=0)
  - **B dw 9** (B=9)
  - **C dw 6** (C=6)
- Ο compiler επέλεξε ότι το τμήμα δεδομένων θα ξεκινάει από τη διεύθυνση 200H.
- Οπότε:
  - στη θέση μνήμης 200H θα υπάρχει η τιμή 0.
  - στη θέση μνήμης 201H θα υπάρχει η τιμή 9.
  - στη θέση μνήμης 202H θα υπάρχει η τιμή 6.

Στο τμήμα δεδομένων έχουμε τις παρακάτω τιμές



Παράδειγμα λειτουργίας ΤΟΥ86

Λειτουργικό Διάγραμμα της αρχιτεκτονικής TOY86 (2/2)

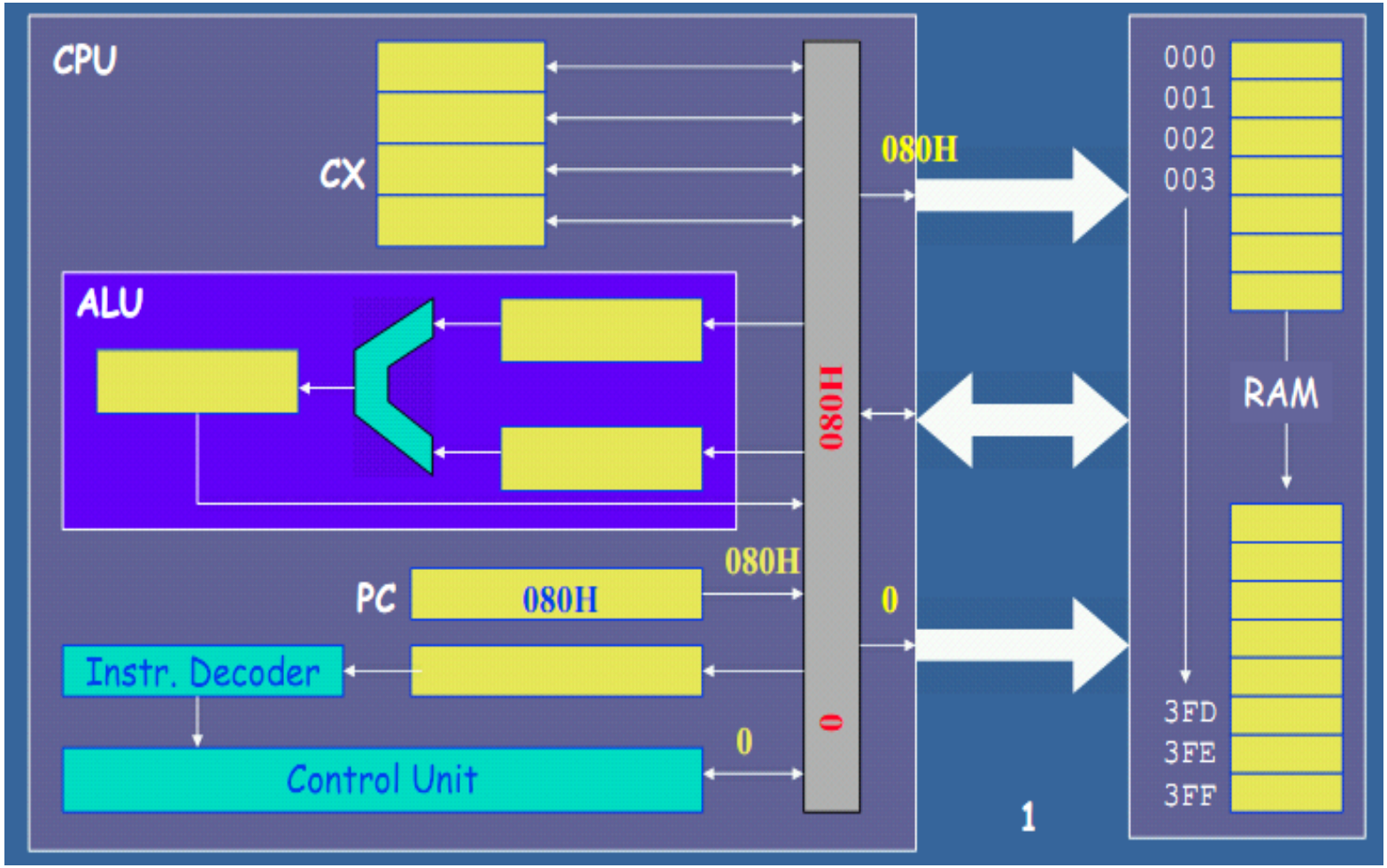


Κύκλος ρολογιού 1

- Ο μετρητής προγράμματος PC αρχικοποιείται στην τιμή **080H** (εργοστασιακή ρύθμιση ύστερα από reset/power on).

- Τοποθετείται αυτή η διεύθυνση στο δίαυλο διευθύνσεων και ζητείται από τη μνήμη να διαβάσει αυτό το κελί (0 στο δίαυλο ελέγχου).

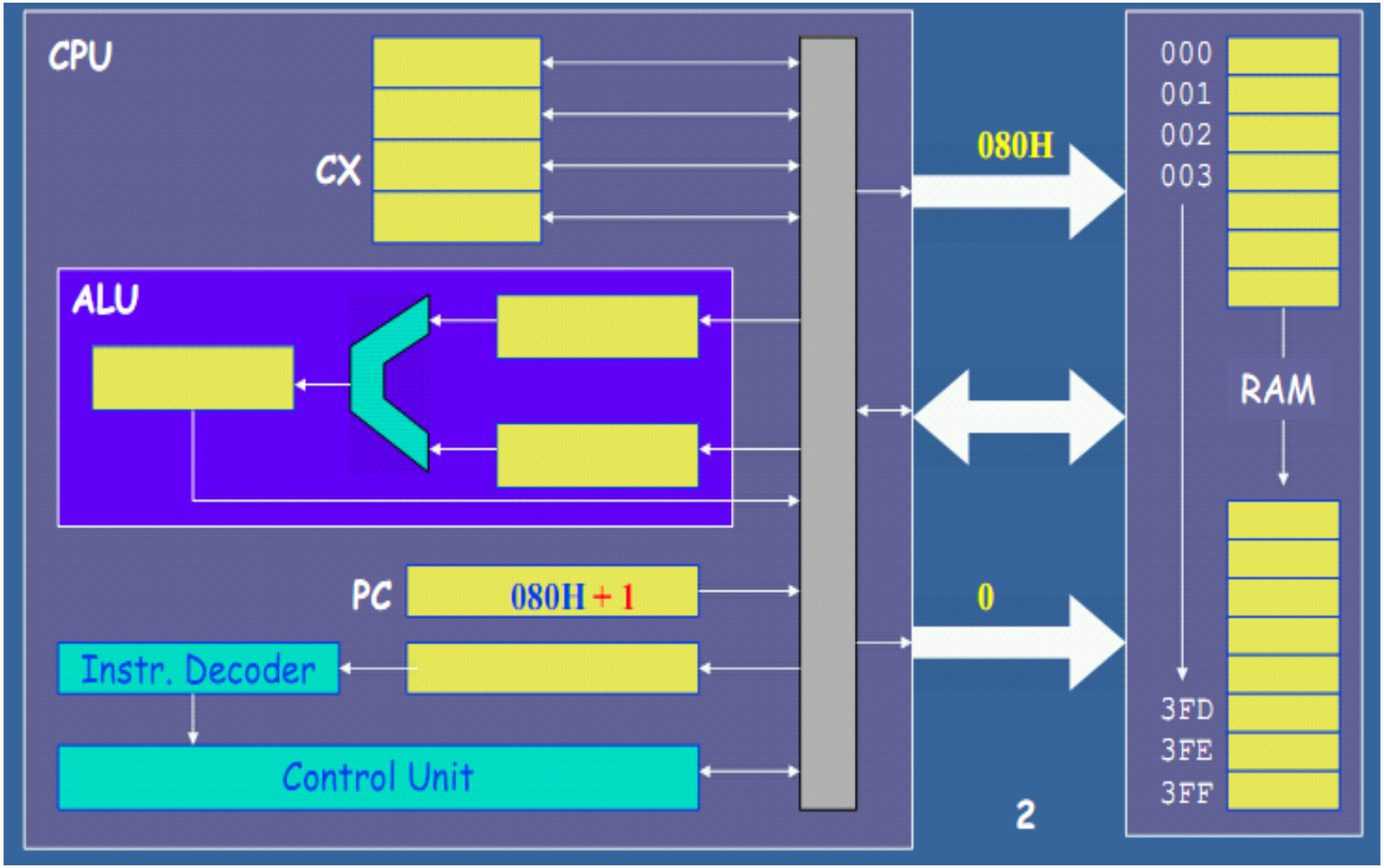
Ο PC έχει τιμή 080H, ζητάει από μνήμη την πρώτη εντολή



Κύκλος ρολογιού 2

Αυξάνεται ο μετρητής προγράμματος κατά 1.

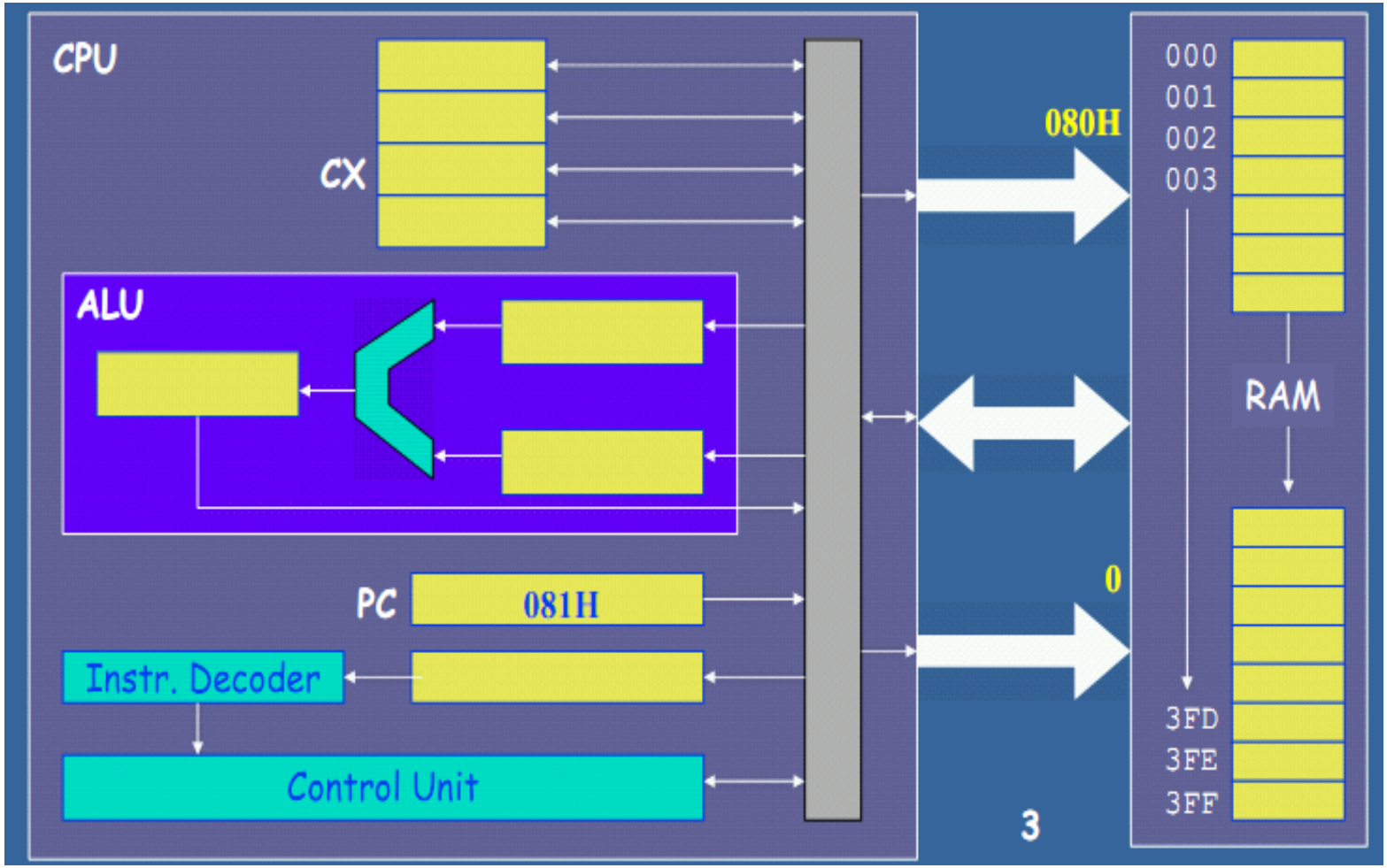
Το PC αυξάνει κατά 1



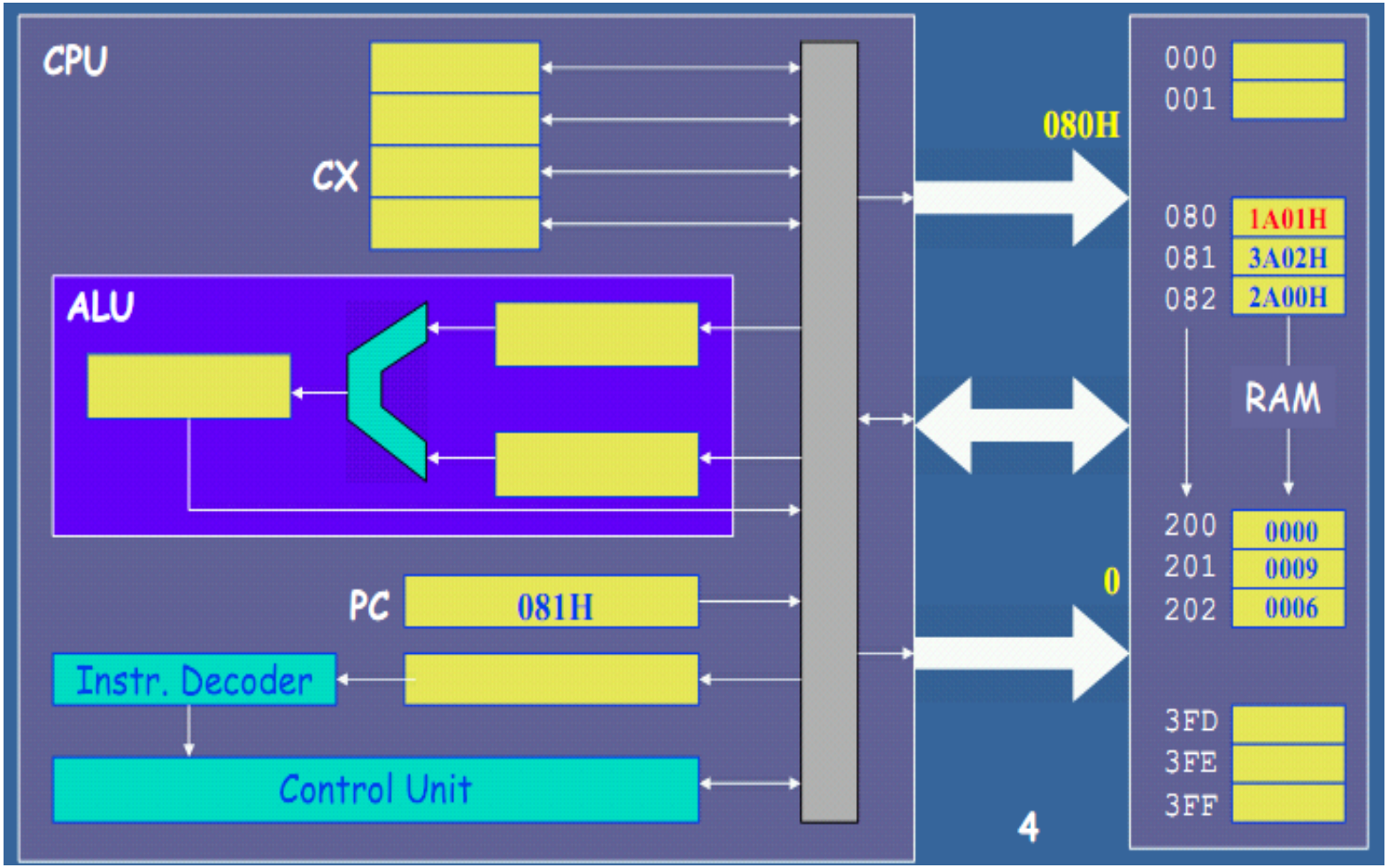
Κύκλοι ρολογιού 3,4,5

- Η πρόσβαση στην εξωτερική μνήμη είναι πολύ πιο αργή από ότι στους καταχωρητές.
- Τα αποτελέσματα έχουν έρθει από το **Data Bus** μέσα στον επεξεργαστή στον κύκλο 5 και έχουν τοποθετηθεί μέσα στον Instruction Register.
- Ο Instruction Register έχει μεταφέρει τα αποτελέσματα στον αποκωδικοποιητή και έχει δώσει τις κατάλληλες οδηγίες στη μονάδα ελέγχου (control unit).
- Η μονάδα ελέγχου παίρνει τις τιμές 1 (πράξη), 2 (καταχωρητής), 201H (δ/νση μνήμης).
- Δίνεται εντολή να διαβαστεί η δ/νση μνήμης 201H.

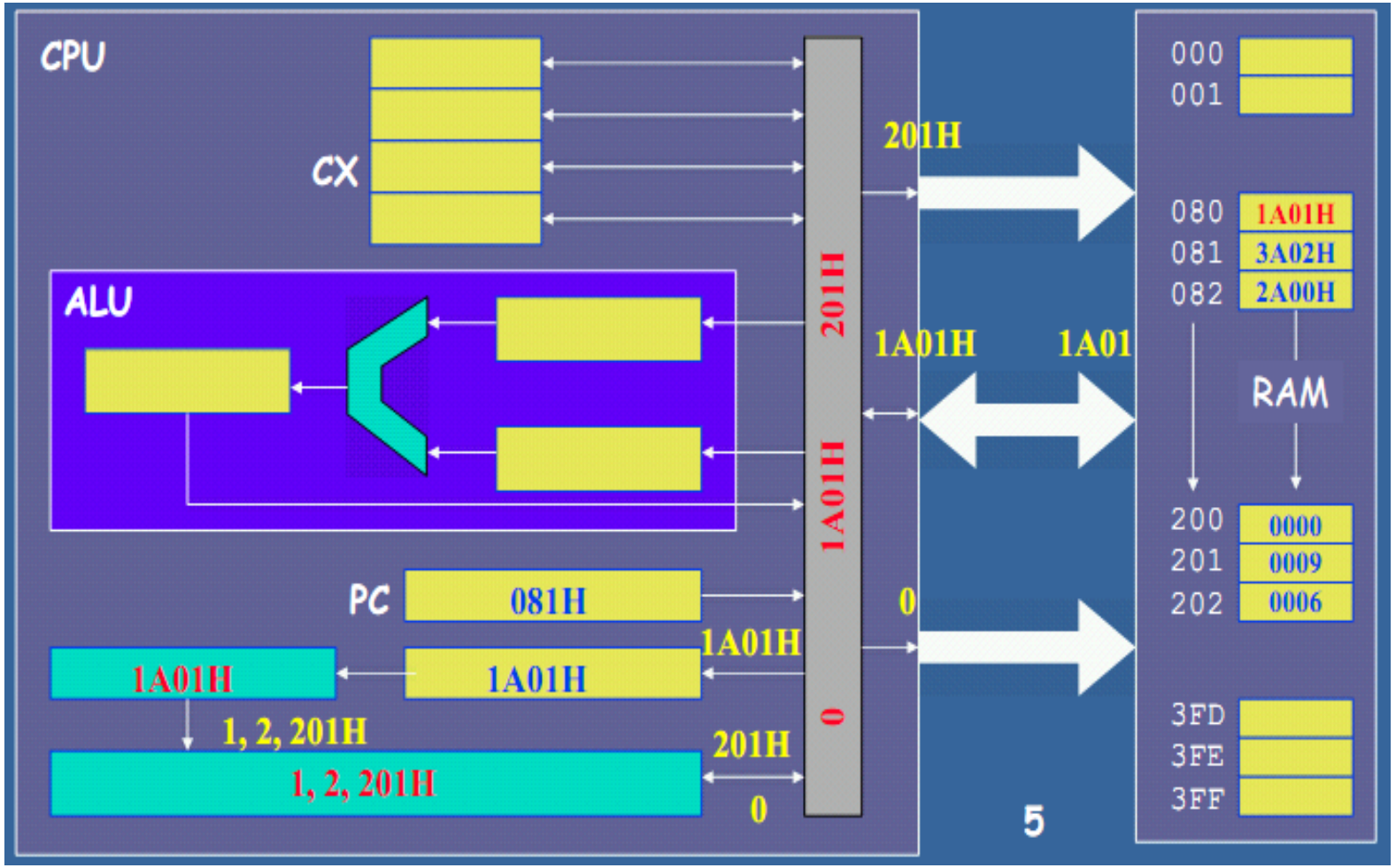
Κύκλος 3. Αναζήτηση δ/νσης μνήμης



Κύκλος 4: Ανάγνωση περιεχομένου μνήμης



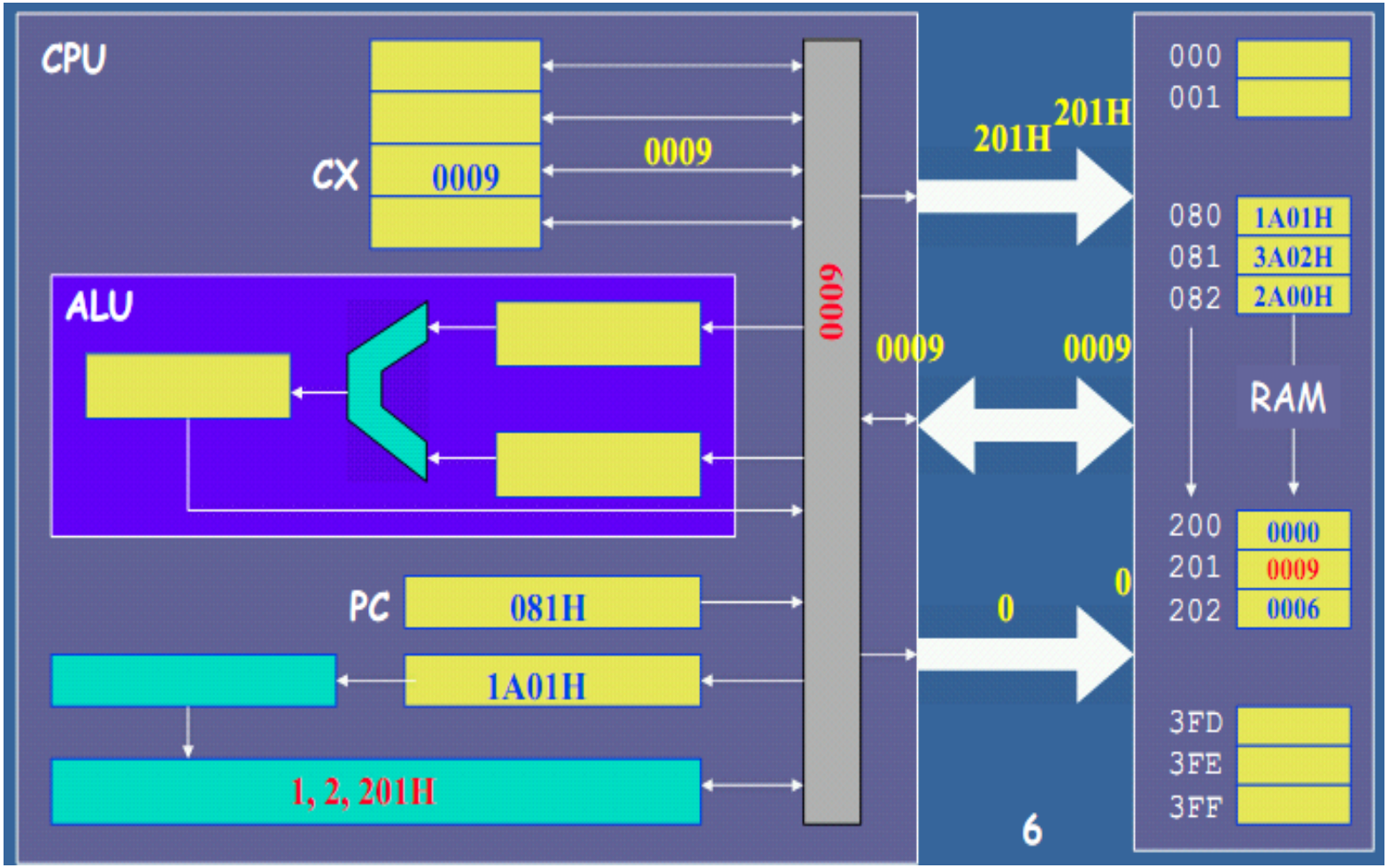
Κύκλος 5: Μεταφορά περιεχομένου στο IR & αποκωδικ.



Κύκλος ρολογιού 6

- Διαβάζεται η θέση μνήμης 200H και το νούμερο τοποθετείται στον καταχωρητή που έχει προσδιοριστεί.

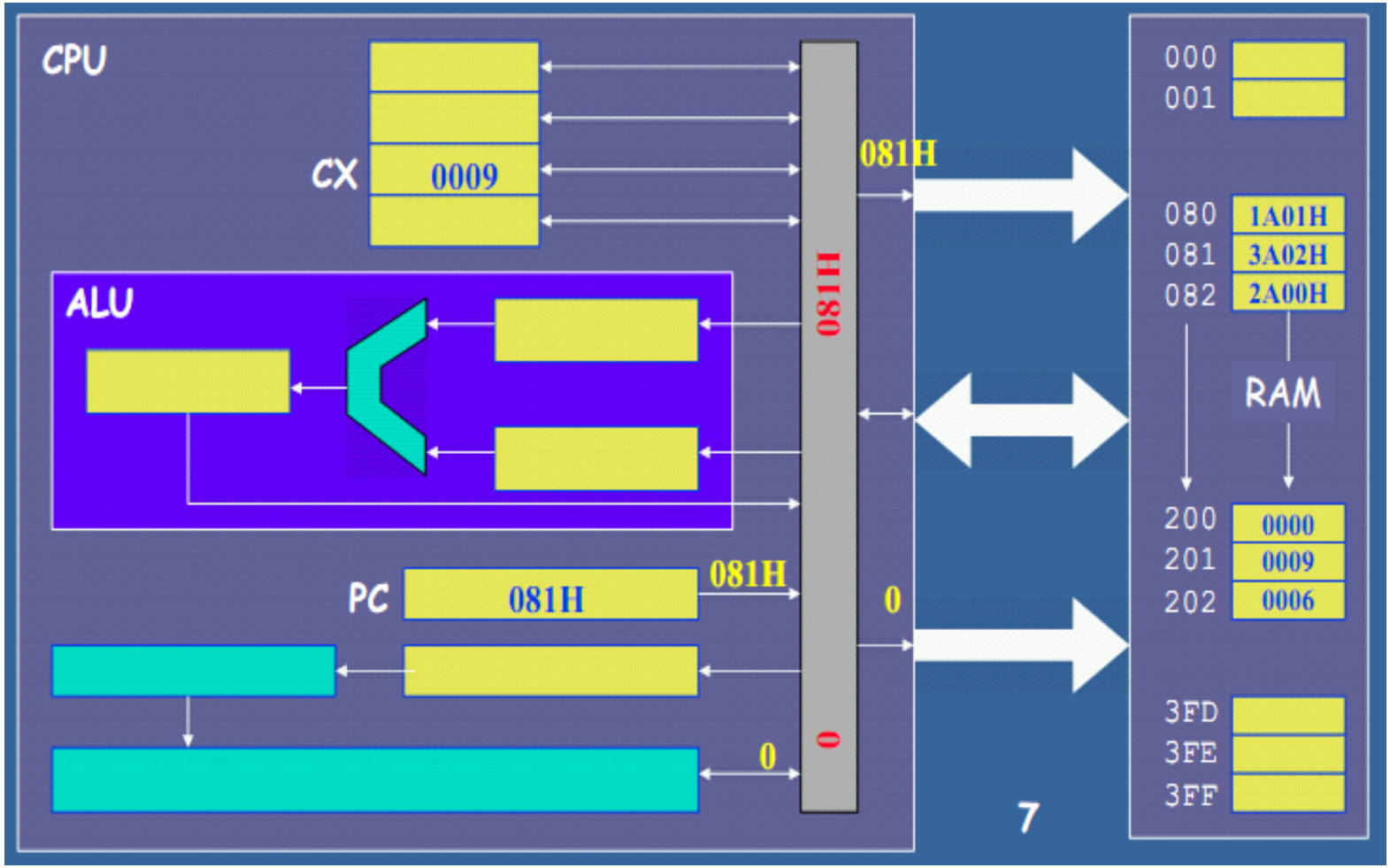
Κύκλος 6: Ανάγνωση θέσης 201H στον καταχωρητή CX



Κύκλος ρολογιού 7

- Χρησιμοποιείται η τιμή του μετρητή προγράμματος για να προσδιορίσει την επόμενη εντολή που θα διαβαστεί από τη μνήμη.
- Ο PC έχει την τιμή 081H οπότε ζητείται από τη μνήμη να διαβαστεί η τιμή του κελιού που βρίσκεται στη διεύθυνση μνήμης 081H.

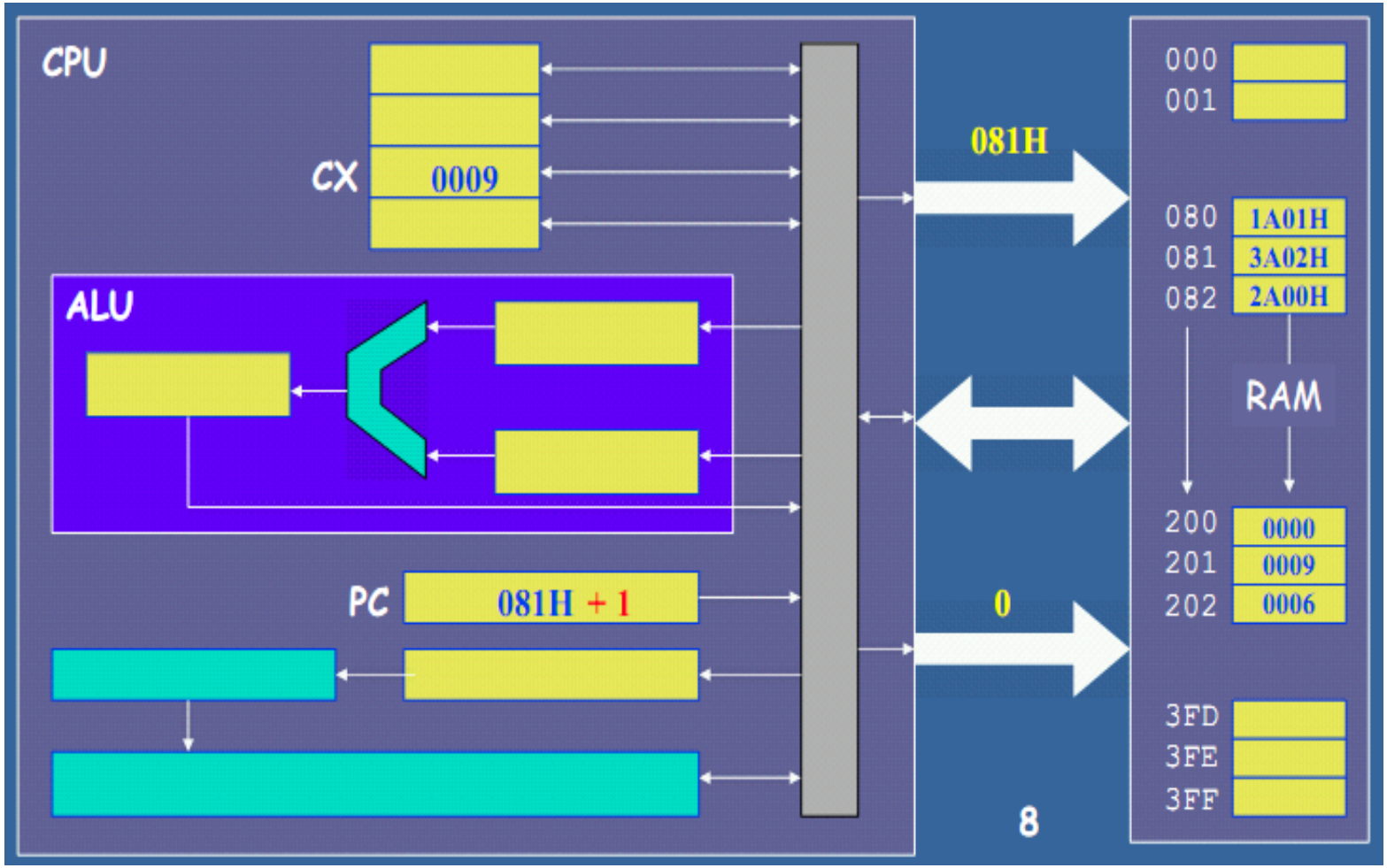
Κύκλος 7: Αναζήτηση δ/νση επόμενης εντολής



Κύκλος ρολογιού 8

- Αυξάνεται ο μετρητής προγράμματος κατά 1.

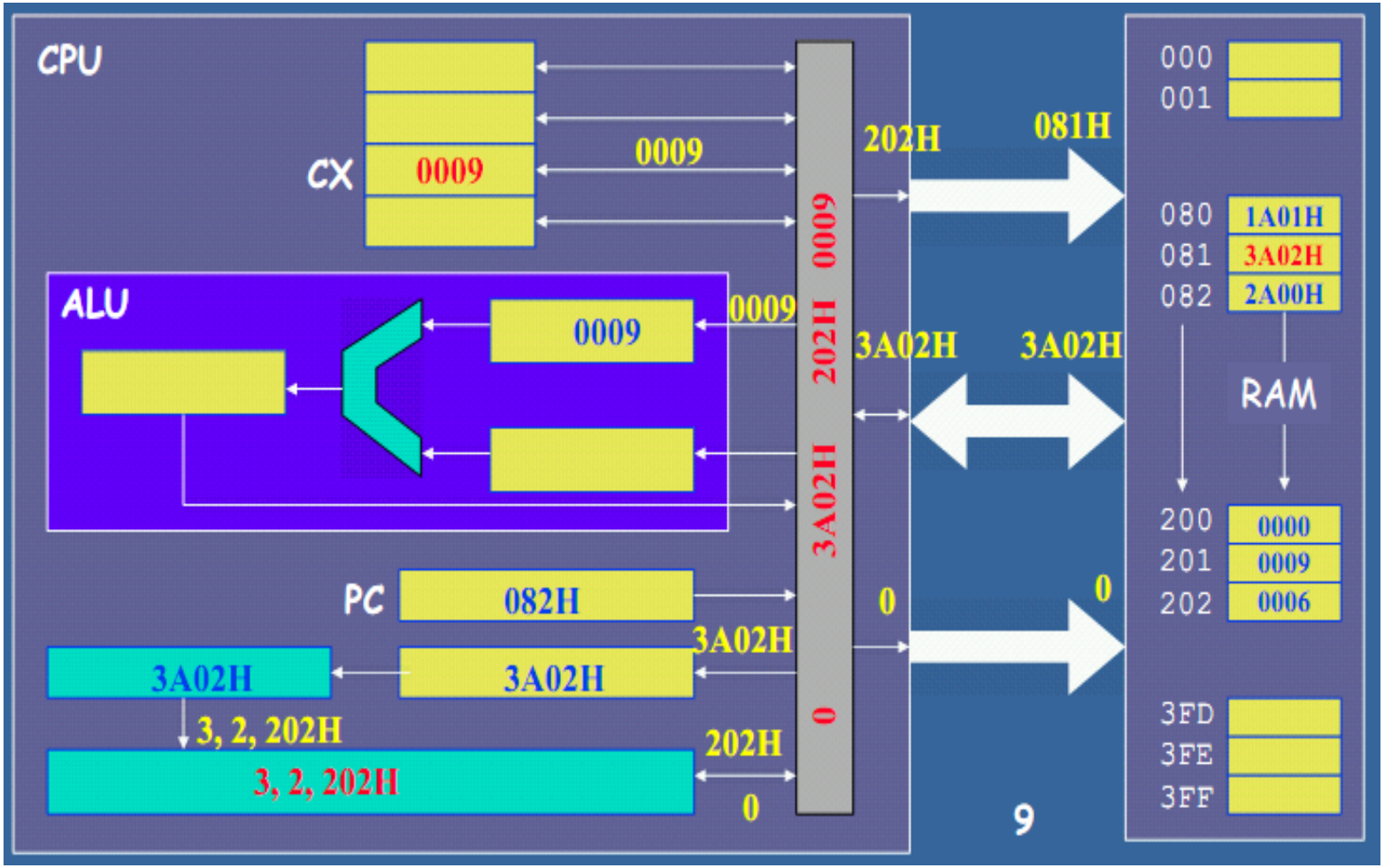
Κύκλος 8: Αύξηση PC κατά 1



Κύκλος ρολογιού 9

- Διαβάζεται η τιμή 3A02H, τοποθετείται στον Instruction Register, αποκωδικοποιείται και ρυθμίζεται κατάλληλα η μονάδα ελέγχου.
- Η εντολή αποκωδικοποιείται σε '3' (πράξη λειτουργίας πρόσθεση), '2' (χρήση του καταχωρητή CX) και 202H (η δ/νση του άλλο προσθετέου).
- Τοποθετείται στον πρώτο καταχωρητή της ALU η τιμή που έρχεται από το CX (0009).
- Δίνεται εντολή στην εξωτερική μνήμη να φέρει τα δεδομένα του κελιού 202H.

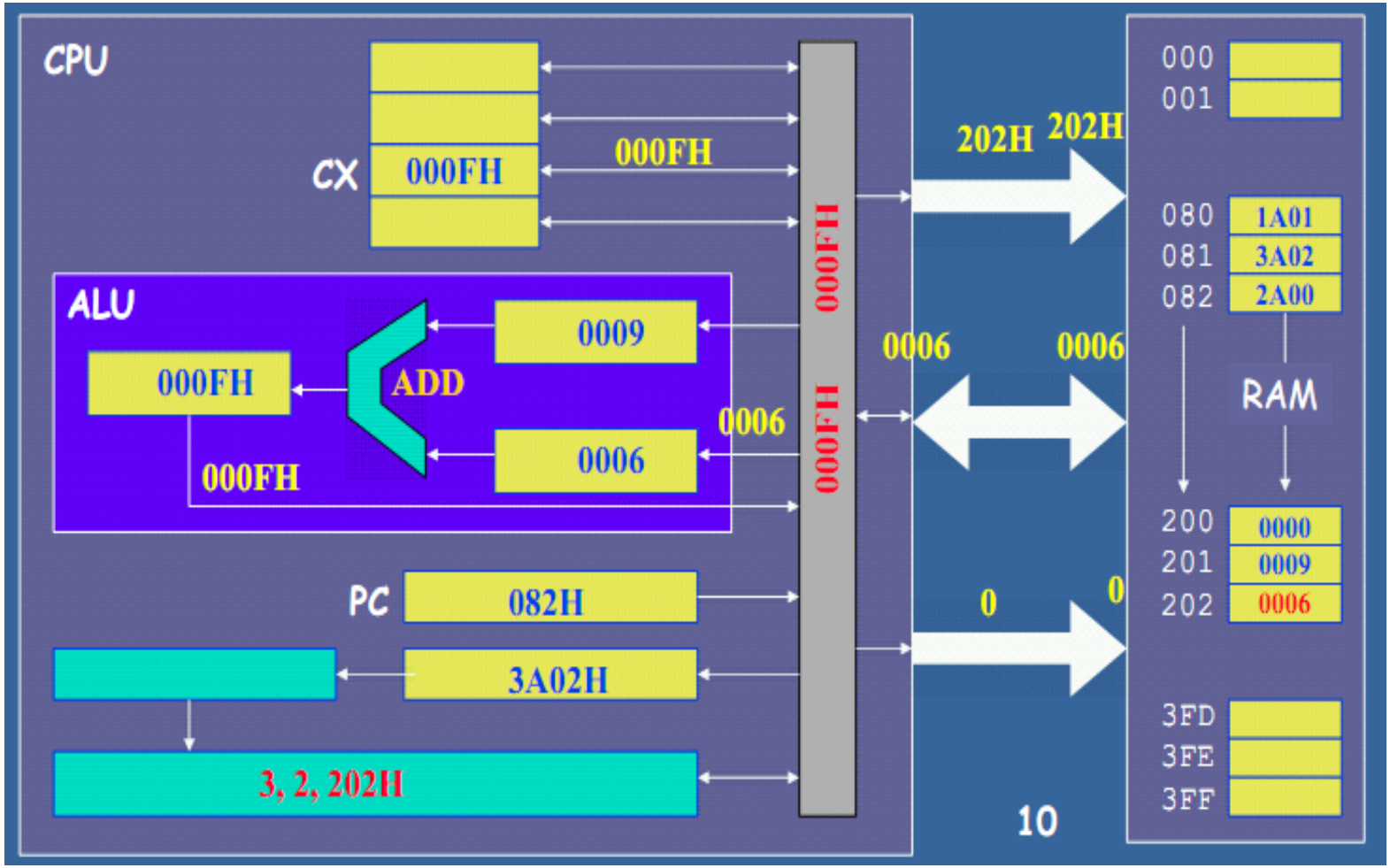
Κύκλος 9: Μεταφορά εντολής στον IR, αποκωδικοποίηση, CX



Κύκλος ρολογιού 10

- Έρχονται τα δεδομένα από την εξωτερική μνήμη.
- Τοποθετούνται στο 2ο καταχωρητή της ALU.
- Από τη μονάδα ελέγχου προσδιορίζεται ότι θα γίνει η πράξη της άθροισης.
- Βρίσκεται το αποτέλεσμα 000FH το οποίο τοποθετείται στον καταχωρητή αποτελέσματος της ALU.
- Μεταφέρεται το αποτέλεσμα από τον καταχωρητή αποτελέσματος της ALU στον καταχωρητή προορισμού CX.

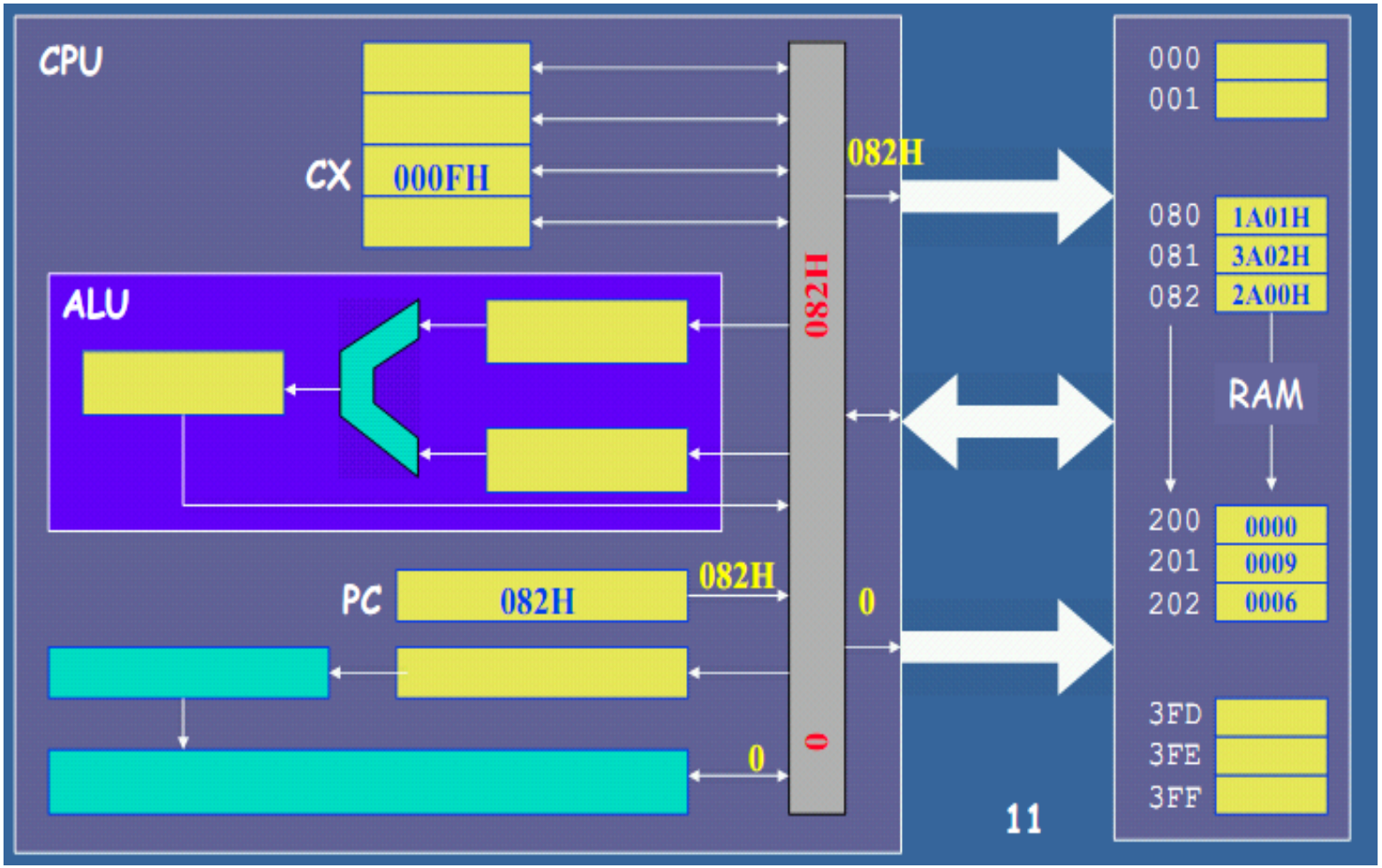
Κύκλος 10: Μεταφορά 202h στο ALU_IN2, άθροισης, μεταφορά ALU_OUT στο CX



Κύκλος ρολογιού 11

- Ο μετρητής προγράμματος έχει την τιμή 082H, οπότε θα πρέπει να μεταφερθεί το περιεχόμενο της θέσης μνήμης 082H στον καταχωρητή εντολών για να αποκωδικοποιηθεί.

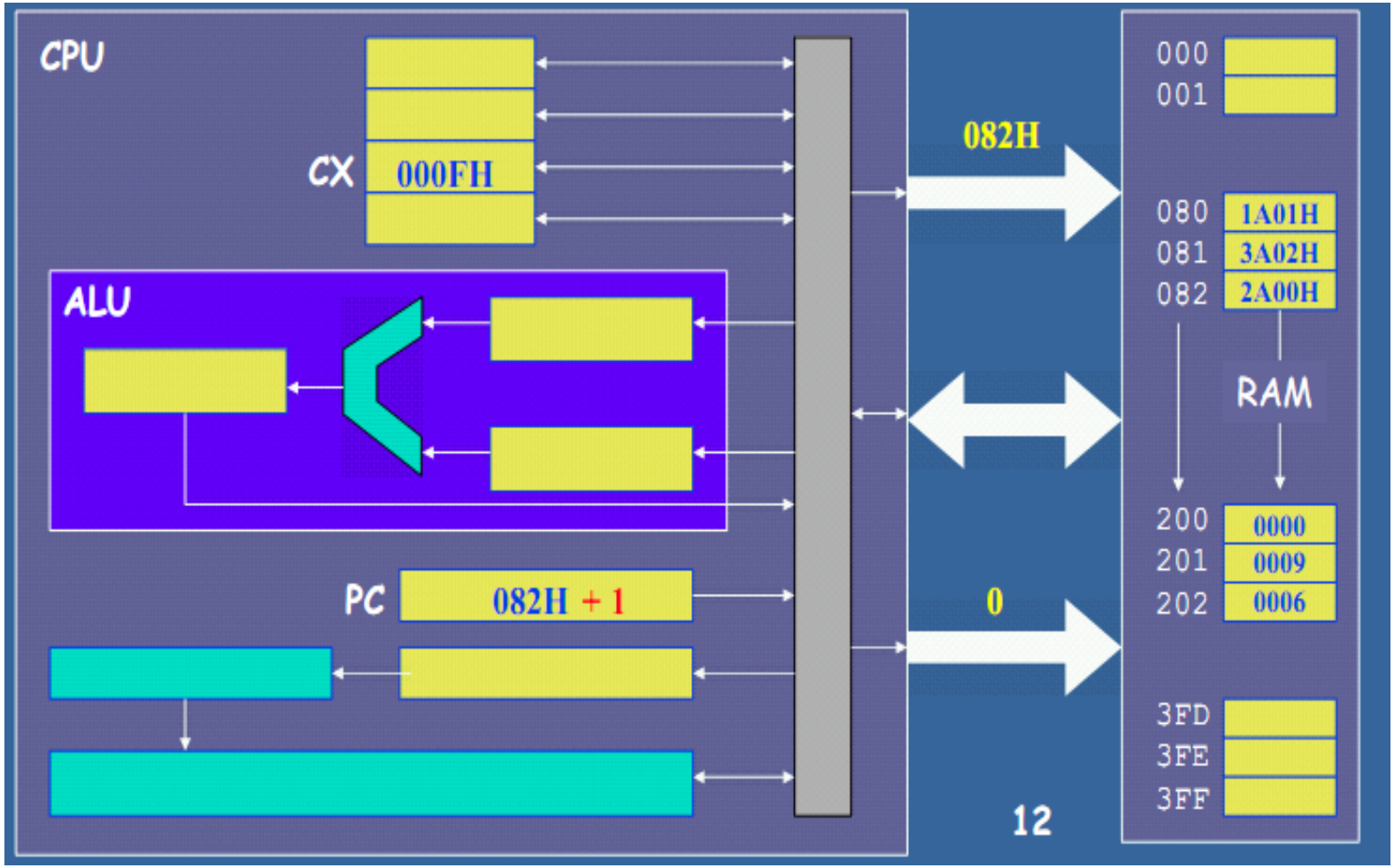
Κύκλος 11: Νέα εντολή από τη δ/νση 082H



Κύκλος ρολογιού 12

- Αυξάνεται κατά 1 ο μετρητής προγράμματος.
- Η νέα τιμή του μετρητή προγράμματος είναι 083H.

Κύκλος 12: Αυξάνεται κατά 1 ο PC

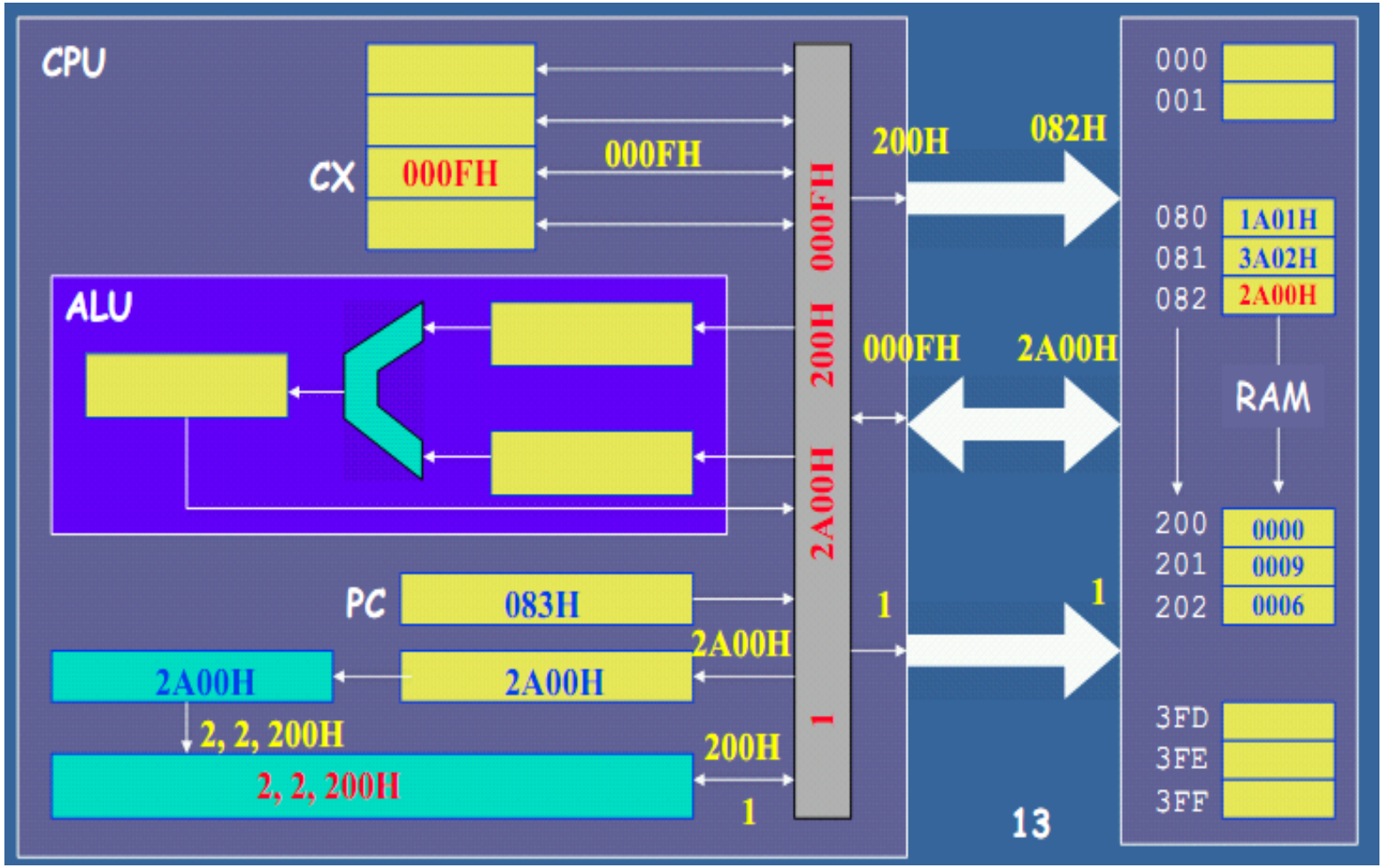


Κύκλος ρολογιού 13

- Η εντολή που έρχεται στον καταχωρητή εντολών είναι η 2A00H.

- Αποκωδικοποιείται και ρυθμίζεται η μονάδα ελέγχου με τις τιμές 2 (πράξη λειτουργίας μετακίνησης σε θέση μνήμης), 2 (καταχωρητής CX), 200H (μετακίνηση σε αυτή τη θέση μνήμης).

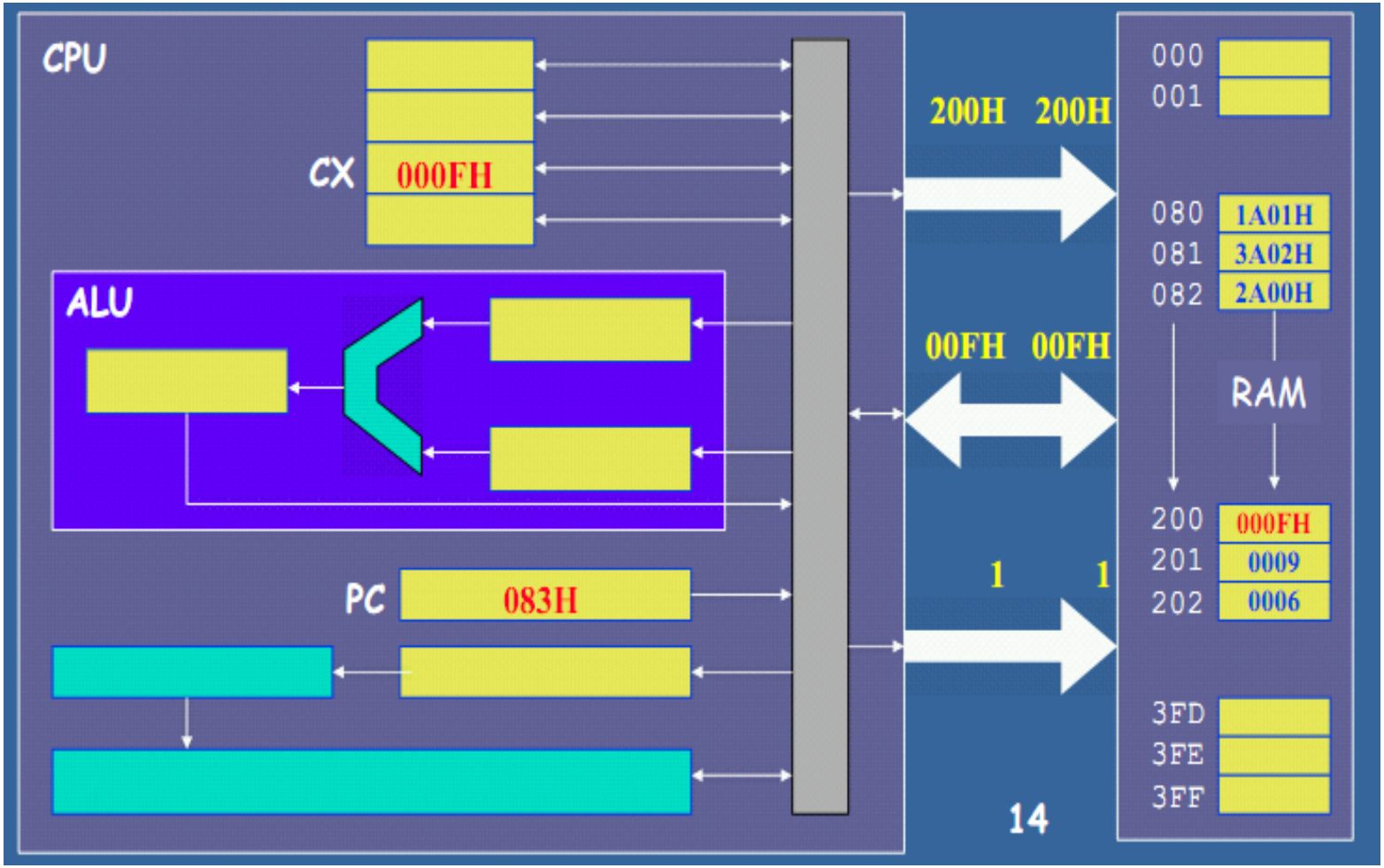
Κύκλος 13: Μεταφορά νέας εντολής, αποκωδικοποίηση



Κύκλος ρολογιού 14

- Τοποθετείται η τιμή του CX (000Fh) στη θέση μνήμης 200H.

Κύκλος 14: Μεταφορά CX στην αντίστοιχη διεύθυνση μνήμης



Το παράδειγμα που είδαμε ήταν ένας επεξεργαστής **Single-Instruction**

- Το παράδειγμα που αναλύσαμε εκτελούσε μια εντολή κάθε φορά.
- Δηλαδή μια εντολής έρχονταν (fetch), αποκωδικοποιούνταν (decode), εκτελούνταν (execute) και αποθηκεύονταν τα δεδομένα (write).
- Υπήρχε λοιπόν ΜΟΝΟ ΜΙΑ εντολή στον επεξεργαστή οποιαδήποτε χρονική στιγμή.
- Όταν η εντολή εκτελούνταν, τότε τα υπόλοιπα τμήματα (fetch, decode, write) ήταν ανενεργά.

Βελτίωση του single-instruction CPU

- Μπορούμε να αυξήσουμε τις επιδόσεις με το να εκμεταλλευόμαστε όλες τις μονάδες τη δεδομένη στιγμή.

- Δηλαδή ενώ γράφουμε (**Write**) τα αποτελέσματα μιας εντολής (0), η αμέσως επόμενη (1 εντολή πιο κάτω) εκτελείται (**Execute**), η επόμενη (2 εντολές πιο κάτω) αποκωδικοποιείται (**Decode**) και η επόμενη (3 εντολές πιο κάτω) εισέρχεται μέσα στον επεξεργαστή (**Fetch**).

- Αυτή η τεχνική ονομάζεται διασωλήνωση ή διοχέτευση (pipeline),

.....και αναλύεται σε επόμενη διάλεξη.

Τέλος Ενότητας