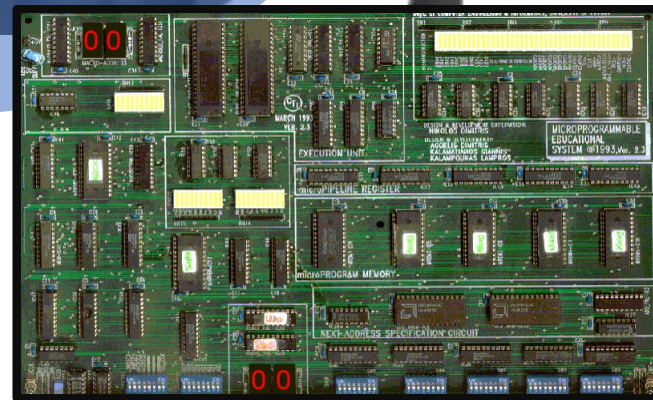
Εργαστήριο Αρχιτεκτονικής Η/Υ 2023-24

Δ. Νικολός

Β. Παπαϊωάννου

Β. Φερεντίνος



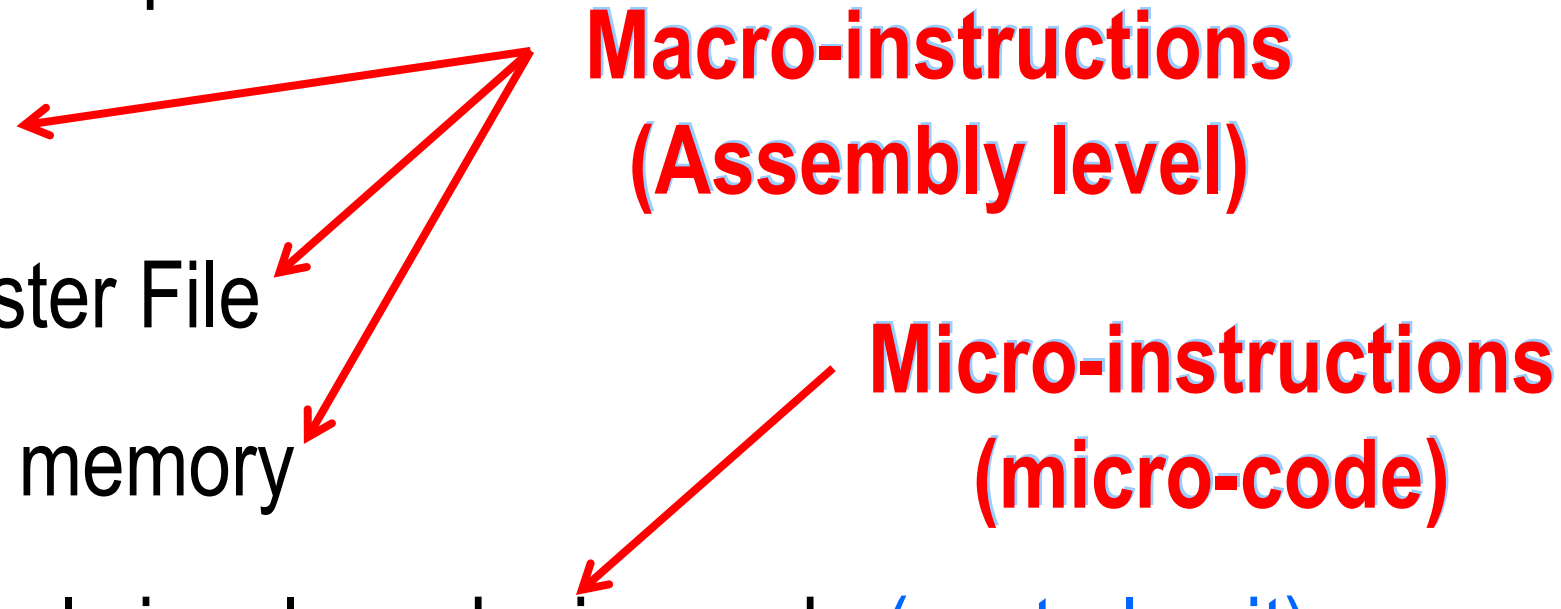
Επέτειος της Επανάστασης του 1821 (25η Μαρτίου)

- Διακοπές του Πάσχα (από το Σάββατο του Λαζάρου μέχρι της Κυριακής του Θωμά)
- Πρωτομαγιά
- Φοιτητικές Εκλογές
- Αγίου Πνεύματος

Εβδομάδα 1	**19/2/2024**	-	**24/2/2024**	
Εβδομάδα 2	26/2/2024	-	2/3/2024	
Εβδομάδα 3	4/3/2024	**ARM 1**	9/3/2024	
Εβδομάδα 4	11/3/2024	**ARM 2**	16/3/2024	
Εβδομάδα 5	18/3/2024	**ARM 3**	23/3/2024	ΚΑΘΑΡΗ ΔΕΥΤΕΡΑ
Εβδομάδα 6	25/3/2024	**ARM 4**	30/3/2024	ΔΕ 25 ΜΑΡΤΙΟΥ
Εβδομάδα 7	1/4/2024	**MICRO 1**	6/4/2024	
Εβδομάδα 8	8/4/2024	**MICRO 2**	13/4/2024	
Εβδομάδα 9	15/4/2024	**MICRO 3**	20/4/2024	
Εβδομάδα 10	22/4/2024	**MICRO 4**	27/4/2024	
	29/4/2024	-	4/5/2024	
ΠΑΣΧΑ	6/5/2024	-	11/5/2024	
Εβδομάδα 11	13/5/2024	**ΑΝΑΠΛΗΡΩΣΗ**	18/5/2024	
Εβδομάδα 12	20/5/2024	**ΕΞΕΤΑΣΕΙΣ**	25/5/2024	
Εβδομάδα 13	27/5/2024	-	1/6/2024	Λήξη εξαμήνου 31/5
	3/6/2024	-	8/6/2024	

Ας ξεκινήσουμε

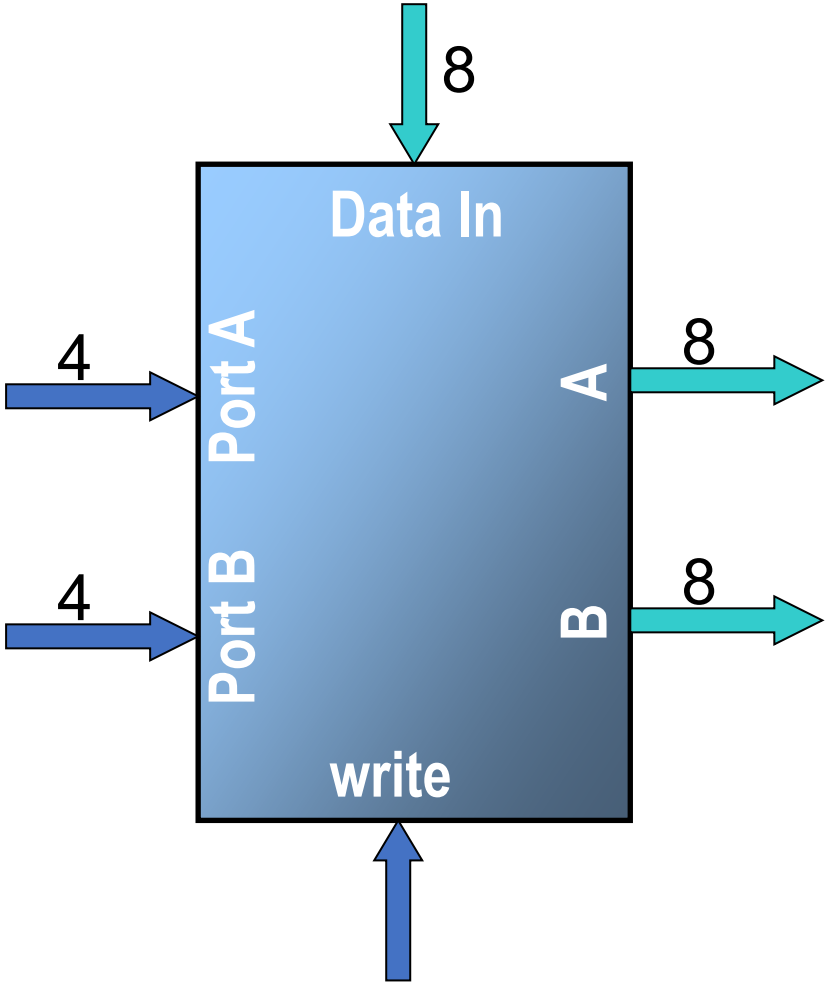
- Τι χρειαζόμαστε για να υλοποιήσουμε έναν επεξεργαστή?
  - ALU
  - Register File
  - Main memory
  - Control signals and microcode (control unit)



Βασικές Έννοιες

Register File (Καταχωρητές)

Καταχωρητής : Βασική μονάδα αποθήκευσης για την εκτέλεση πράξεων



16 x 8 bit Register File : 16 καταχωρητές των 8 bits

Ένα read-only port (**PortA**)

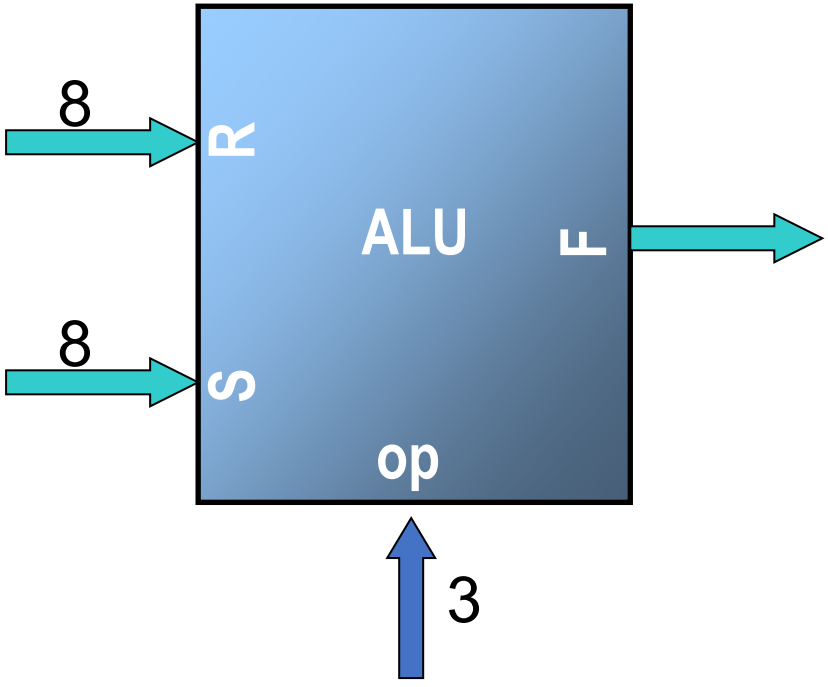
Ένα read/write port (**PortB** και **PortB**)

Σε κάθε κύκλο, η έξοδος A ή B δίνει τα περιεχόμενα του καταχωρητή που διευθυνσιοδοτείται από την είσοδο (**PortA** ή **PortB)**

Αν επιπλέον, είναι ενεργοποιημένο το σήμα **write**, τα δεδομένα εισόδου **Data In**, εγγράφονται στον καταχωρητή που διευθυνσιοδοτείται από το **PortB**

Αριθμητική και Λογική Μονάδα (ALU)

Η καρδιά του data-path



Εκτελεί αριθμητικές και λογικές πράξεις, ανάλογα με την τιμή της εισόδου **op**.

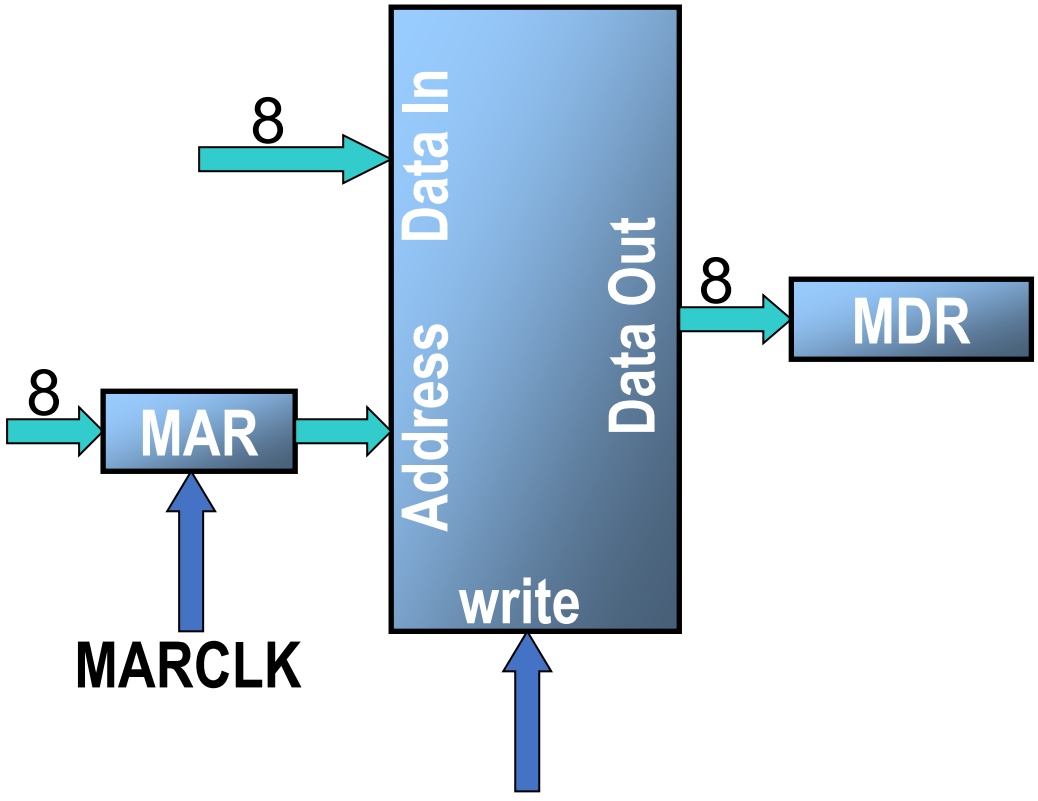
Το αποτέλεσμα της πράξης δίνεται ως έξοδος από το **F**

Π.χ., αν **op = 000,** τότε θα γίνει η πράξη

F = S + R

Η αντιστοίχιση κωδικών της εισόδου **op** σε πράξεις υπάρχει σε μορφή πίνακα στις σημειώσεις του εργαστηρίου.

Κύρια Μνήμη



write : Η τιμή του σήματος αυτού καθορίζει αν θα γίνει εγγραφή ή ανάγνωση στη μνήμη

MAR : *Memory Address Register (8 bit Καταχωρητής)*

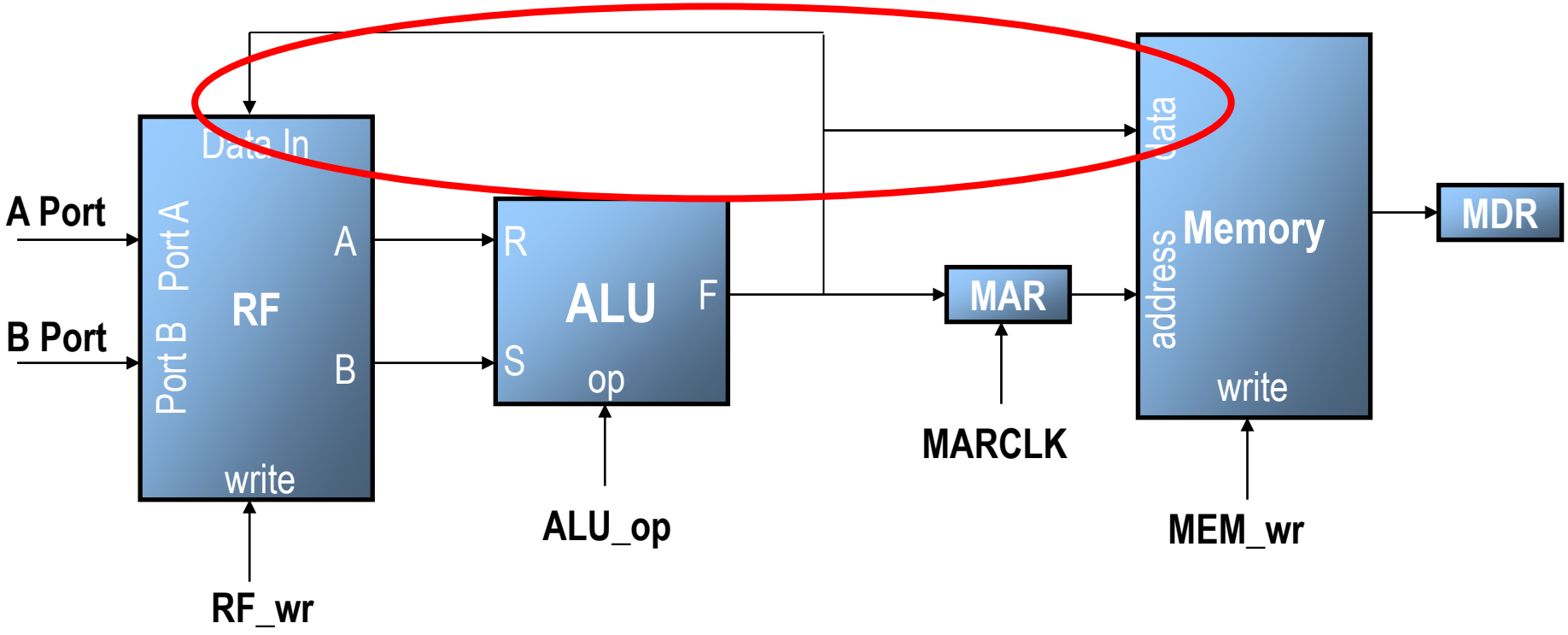
Διευθυνσιοδοτεί τη μνήμη. Κάθε φορά που θέλουμε να αλλάξουμε διεύθυνση, ενεργοποιούμε το σήμα **MARCLK.** Με τον τρόπο αυτό η νέα διεύθυνση **κλειδώνεται** στον MAR.

MDR : *Memory Data Register (8 bit Καταχωρητής)*

Περιέχει τα δεδομένα της θέσης που διευθυνσιοδοτείται από τον **MAR**.

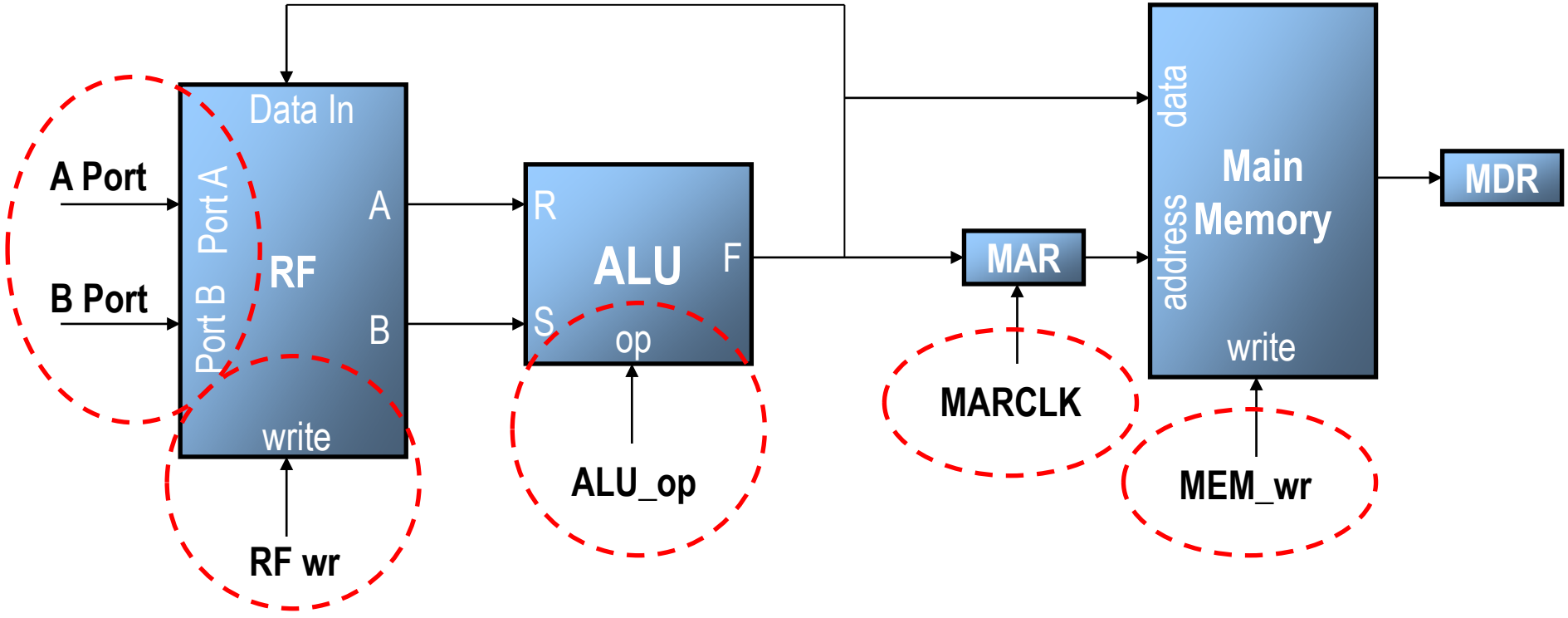
Ένα στοιχειώδες Σύστημα (Υπεραπλουστευμένο)

ΜΟΝΑΔΑ ΕΠΕΞΕΡΓΑΣΙΑΣ



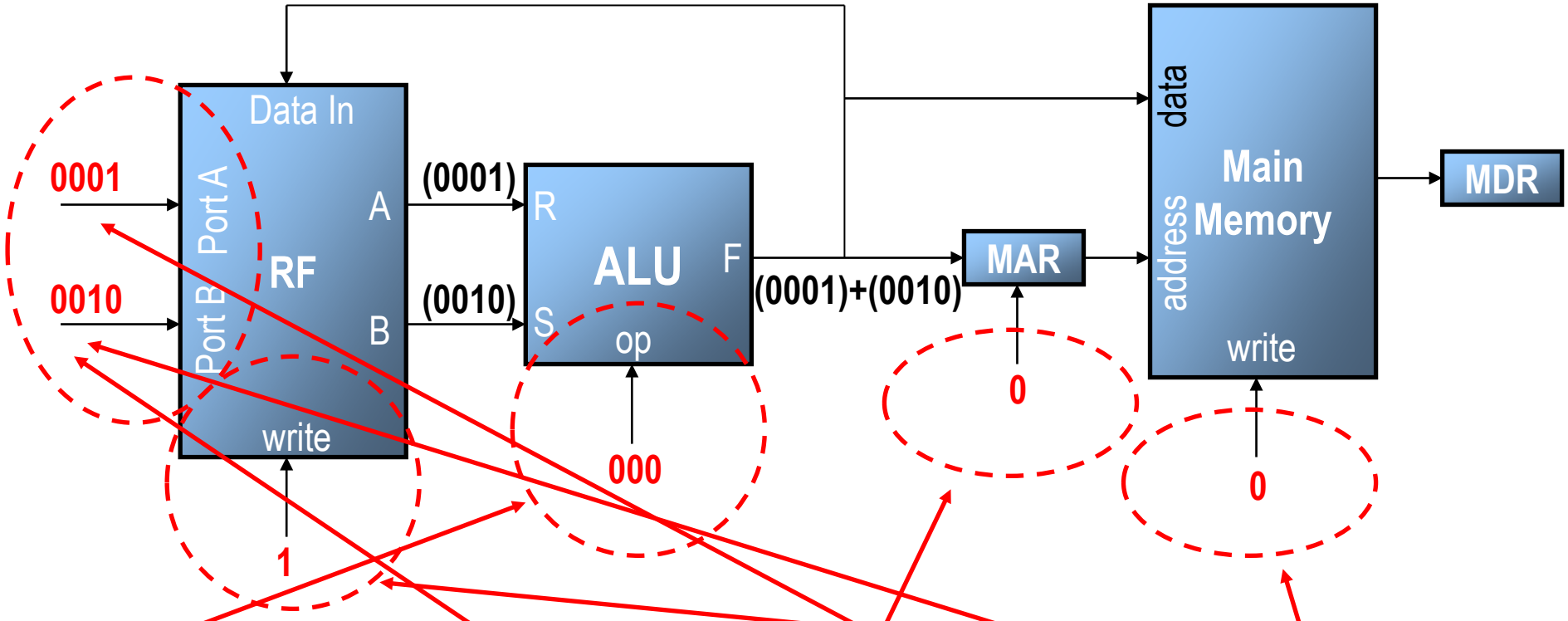
Υπενθύμιση : Οι δύο βασικές υπομονάδες ενός συστήματος είναι η **μονάδα επεξεργασίας** και η **μονάδα ελέγχου**. Στο σχήμα δίνουμε μόνο τη μονάδα επεξεργασίας

Ένα στοιχειώδες Σύστημα (Υπεραπλουστευμένο)



Για να καθορίσουμε τη λειτουργία του συστήματος σε μία χρονική στιγμή, θα πρέπει να θέσουμε τιμές σε **όλα** τα control σήματα.

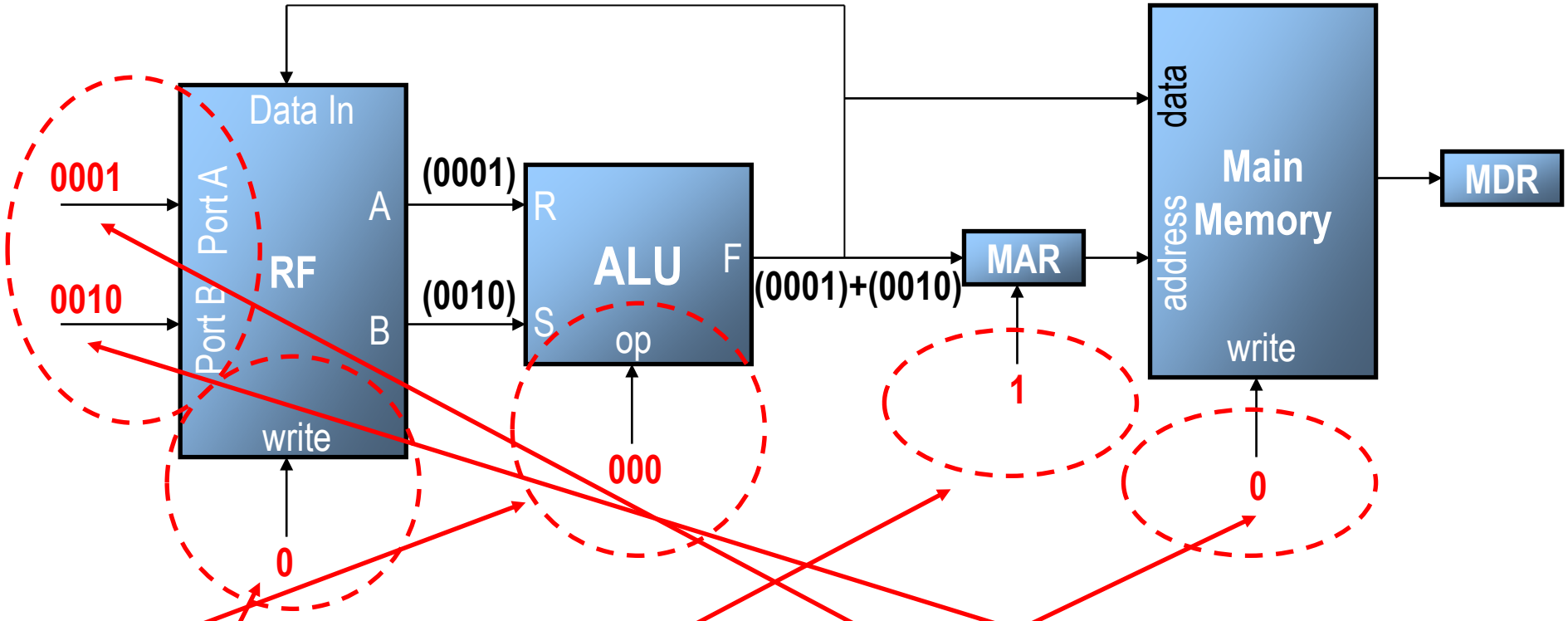
Παράδειγμα 1



Πρόσθεσε τα περιεχόμενα των καταχωρητών 0001 και 0010 και γράψε το αποτέλεσμα στον καταχωρητή 0010.

Δεν επηρεάζεται καθόλου η διεύθυνση της κύριας μνήμης και τα περιεχόμενά της

Παράδειγμα 2



Πρόσθεσε τα περιεχόμενα των καταχωρητών 0001 και 0010.

Μην επηρεάσεις το Register File

Διευθυνσιοδότησε την κύρια μνήμη με το αποτέλεσμα της πράξης

Μην επηρεάσεις το περιεχόμενο της κύριας μνήμης

Σύνθετες Πράξεις

Για να μπορέσουμε να εκτελέσουμε μία σύνθετη πράξη, ‘σπάμε’ την πράξη αυτή σε στοιχειώδεις υποπράξεις.

Για κάθε μία από τις στοιχειώδεις πράξεις θα πρέπει να προσδιορίσουμε όλα τα control σήματα (μικροεντολή).

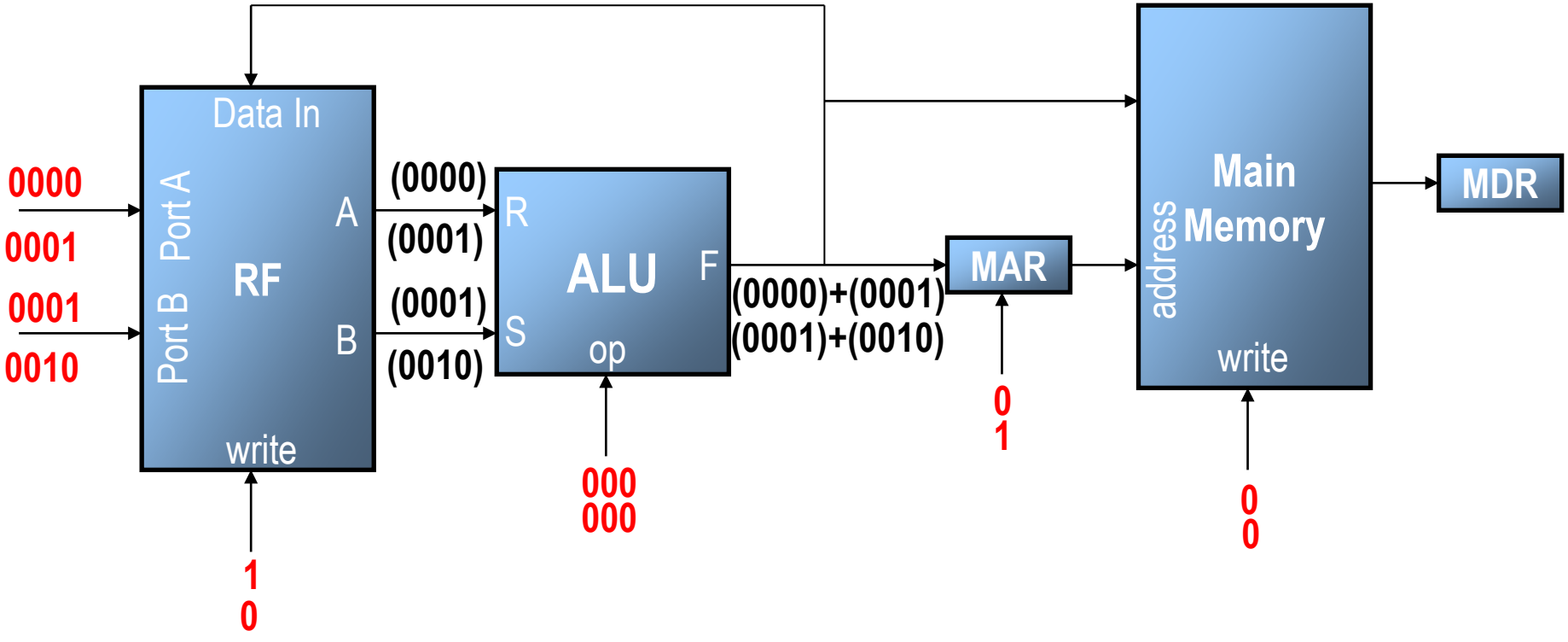
Για να εκτελέσουμε τη σύνθετη πράξη, απλώς εισάγουμε τα control σήματα κάθε υποπράξης διαδοχικά.

Παράδειγμα 3

Ζητούμενο : Διευθυνσιοδότησε την Κύρια μνήμη με την τιμή (0000)+(0001)+(0010) (όπου (X)=περιεχόμενο καταχωρητή X)

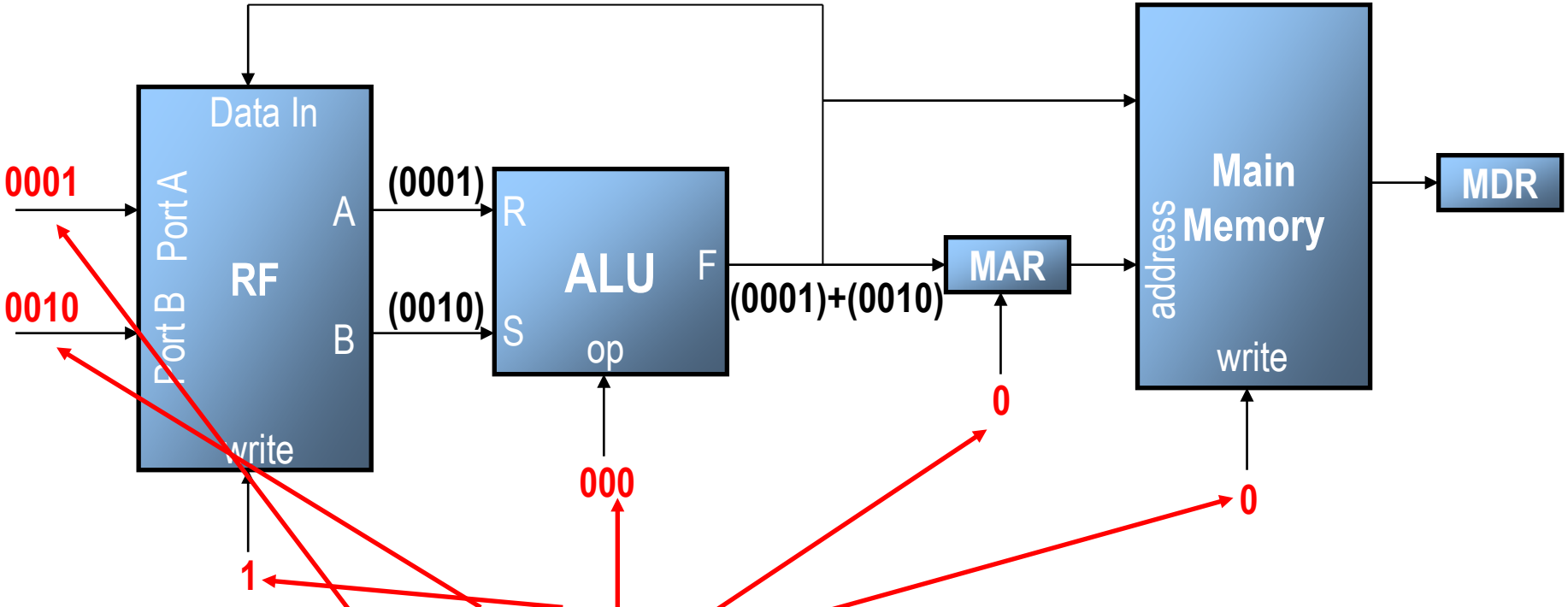
Βήμα 1 : (0000)+(0001) -> (0001)

Βήμα 2 : (0001)+(0010) -> MAR



Μικροεντολές και Μακροεντολές

Μικροεντολή : Ενα bitstream που καθορίζει πλήρως τη λειτουργία του συστήματος σε μία χρονική στιγμή. Η μικροεντολή περιέχει τις τιμές για **όλα τα control σήματα** του συστήματος



Μικροεντολή : 0001 0010 1 000 0 0

Μήκος Μικροεντολής για το απλοϊκό σύστημα : 14 bits (40, 190)

Μικροεντολές και Μακροεντολές

Μακροεντολή (επίπεδο assembly): Μία σύνθετη πράξη

Μια μακροεντολή αποτελείται από μία αλληλουχία μικροεντολών οι οποίες προσδιορίζουν τις στοιχειώδεις πράξεις που θα πρέπει να εκτελεστούν

Παράδειγμα

Μακροεντολή **:** Διευθυνσιοδότησε την Κύρια μνήμη με την τιμή (0000)+(0001) +(0010) (όπου (X)=περιεχόμενο καταχωρητή X)

Υλοποίηση

Μικροεντολή 1 : 0000 0001 1 000 0 0 [(0000)+(0001) -> (0001)]

Μικροεντολή 2 : 0001 0010 0 000 1 0 [(0001)+(0010) -> MAR]

Μικροπρόγραμμα

Instruction Set Architecture (Assembly)

Σύνολο εντολών : Το σύνολο των **μακροεντολών (γλώσσα μηχανής)** που έχουμε υλοποιήσει

Βρίσκουμε ένα σύνολο λειτουργιών τις οποίες θεωρούμε θεμελιώδεις για την υλοποίηση οποιουδήποτε προγράμματος.

Υλοποιούμε τις λειτουργίες αυτές ως γλώσσα μηχανής (μέσω μικροεντολών)

Ο προγραμματιστής του συστήματος χρησιμοποιεί **μόνο** τη γλώσσα μηχανής που ορίσαμε για να υλοποιεί τα προγράμματά του (ο compiler παράγει μακροεντολές)

Ο προγραμματιστής σε καμία περίπτωση δεν γράφει μικροεντολές (αν και θα μπορούσε) : Δεν ασχολείται καθόλου με τη χαμηλού επιπέδου αρχιτεκτονική και το συγχρονισμό των control σημάτων

Προφανώς : Το σύνολο εντολών που ορίζουμε θα πρέπει να είναι αρκετά γενικό ούτως ώστε ο προγραμματιστής να μπορεί να υλοποιήσει ό,τι πρόγραμμα θέλει.

Instruction Set Architecture (Assembly)

Ορίζουμε Ν μακροεντολές (opcode, έντελα, τρόποι διευθυνσιοδότησης …) :

Εντολή1, Εντολή2, … , ΕντολήΝ

Κάθε Εντολή, υλοποιείται μέσω πολλών μικροεντολών

Ο προγραμματιστής γράφει τα προγράμματά του χρησιμοποιώντας το ρεπερτόριο εντολών που ορίσαμε. Π.χ.

Πρόγραμμα :

Εντολή5
Εντολή10
Εντολή4
Εντολή5
Εντολή4

Μακροπρόγραμμα
Το μακροπρόγραμμα βρίσκεται αποθηκευμένο στην **Κύρια Μνήμη**

Μικρομνήμη

Πρόβλημα : Με κάποιον τρόπο θα πρέπει να είμαστε σε θέση ανά πάσα στιγμή να τροφοδοτούμε τις control εισόδους του συστήματος με τις τιμές που υπαγορεύει η κάθε μικροεντολή

Λύση : Αποθηκεύουμε όλες τις μικροεντολές σε μία μνήμη. Η μνήμη αυτή καλείται **μικρομνήμη**. Η έξοδος της μικρομνήμης δίνεται ως είσοδος στις control εισόδους των υπομονάδων που βρίσκονται στο σύστημα.

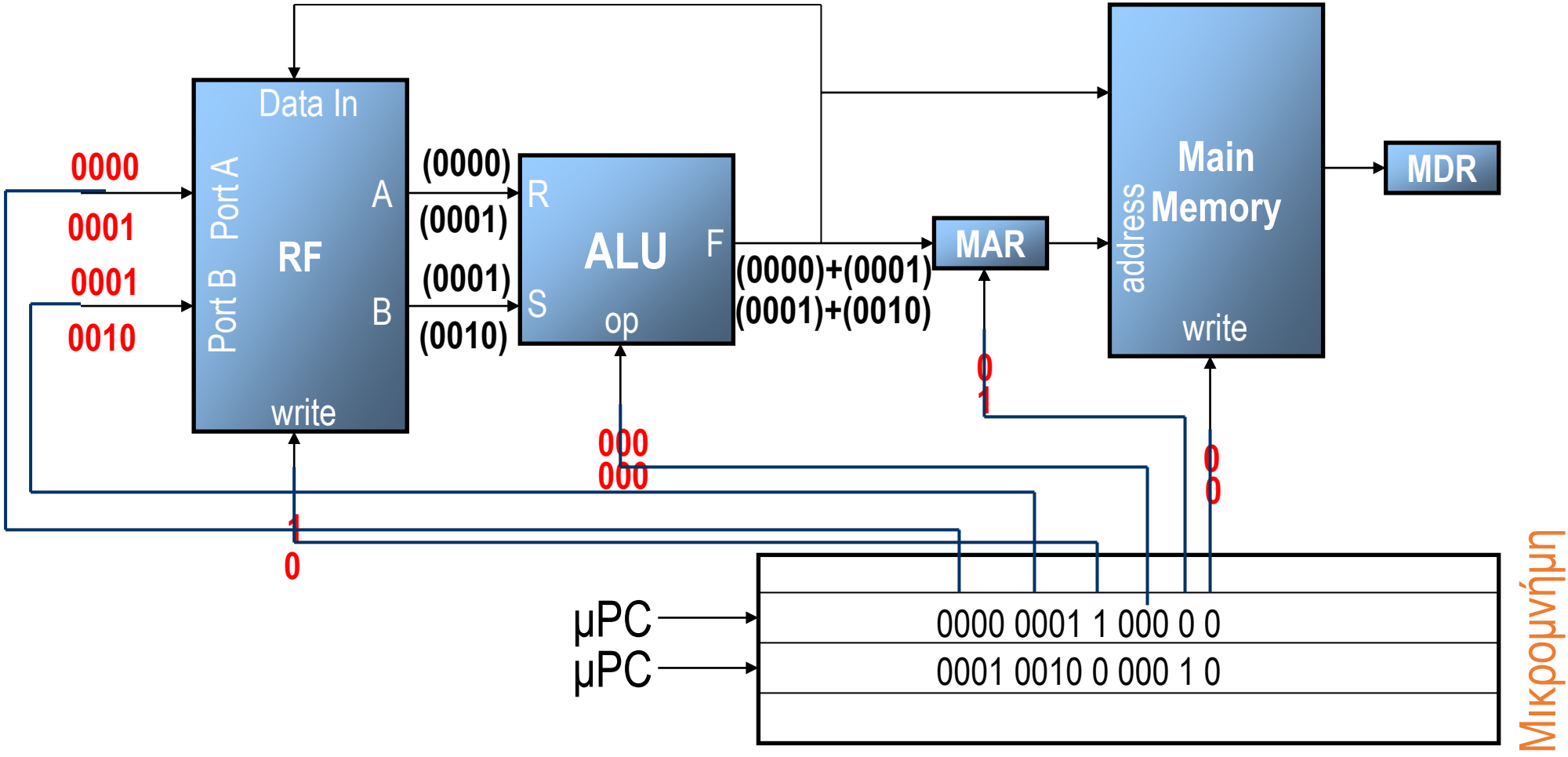
Όπως κάθε μνήμη, έτσι και η μικρομνήμη θα πρέπει να διευθυνσιοδοτείται με κάποιον τρόπο. Για το λόγο αυτό υπάρχει στο hardware ένας μετρητής ο οποίος ανά πάσα στιγμή δείχνει σε κάποια θέση της μικρομνήμης

Ο μετρητής αυτός καλείται **μετρητής μικροπρογράμματος (microPC ή μPC)**

Μικρομνήμη

Βήμα 1 : (0000)+(0001) -> (0001)

Βήμα 2 : (0001)+(0010) -> MAR



Μικρομνήμη

Μπορούμε να αποθηκεύουμε μικροπρογράμματα σε οποιεσδήποτε συνεχόμενες θέσεις της μικρομνήμης θέλουμε.

Υπενθύμιση : Κάθε μικροπρόγραμμα αντιστοιχεί σε μία μάκροεντολή (υπάρχουν και συναρτήσεις με μικροεντολές)

Μνήμη Αντιστοίχισης (Mapper)

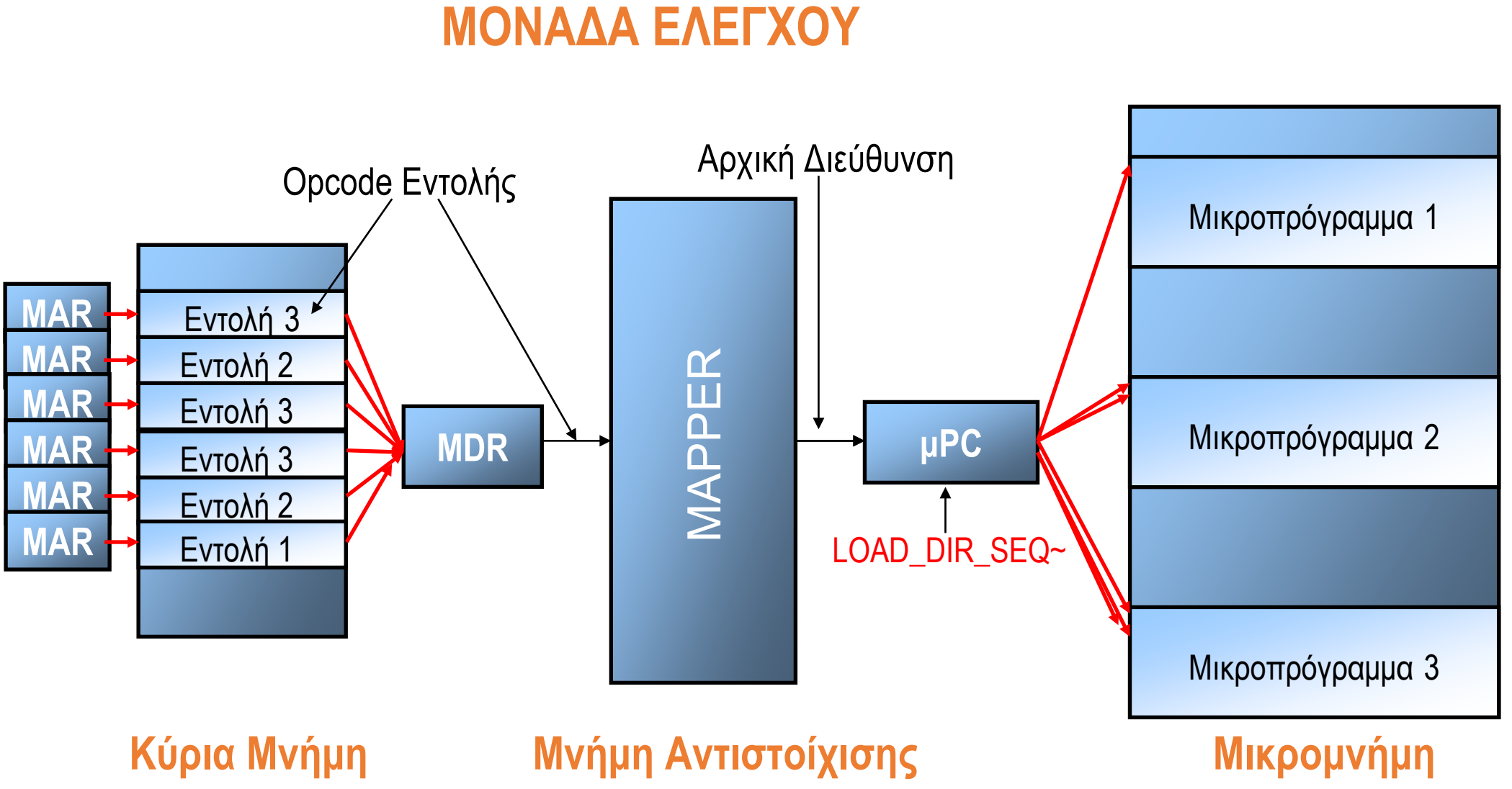
Πρόβλημα : Το μακροπρόγραμμα περιέχει μία ακολουθία μακροεντολών. Το μικροπρόγραμμα για κάθε μακροεντολή μπορεί να βρίσκεται οπουδήποτε μέσα στη μικρομνήμη (βλ. προηγούμενη διαφάνεια). Θα πρέπει να υπάρχει κάποιος τρόπος ούτως ώστε κάθε φορά που συναντάμε μία μακροεντολή, ο μPC να διευθυνσιοδοτεί τη σωστή θέση της μικρομνήμης

Υπενθύμιση : Το μακροπρόγραμμα βρίσκεται αποθηκευμένο στην κύρια μνήμη. Τα μικροπρογράμματα βρίσκονται αποθηκευμένα στην μικρομνήμη.

Λύση : Κάθε μακροεντολή έχει ένα μοναδικό Opcode (κωδικός εντολής). Χρησιμοποιούμε μία μνήμη αντιστοίχισης (mapper). Η μνήμη αυτή εκτελεί την αντιστοίχιση :

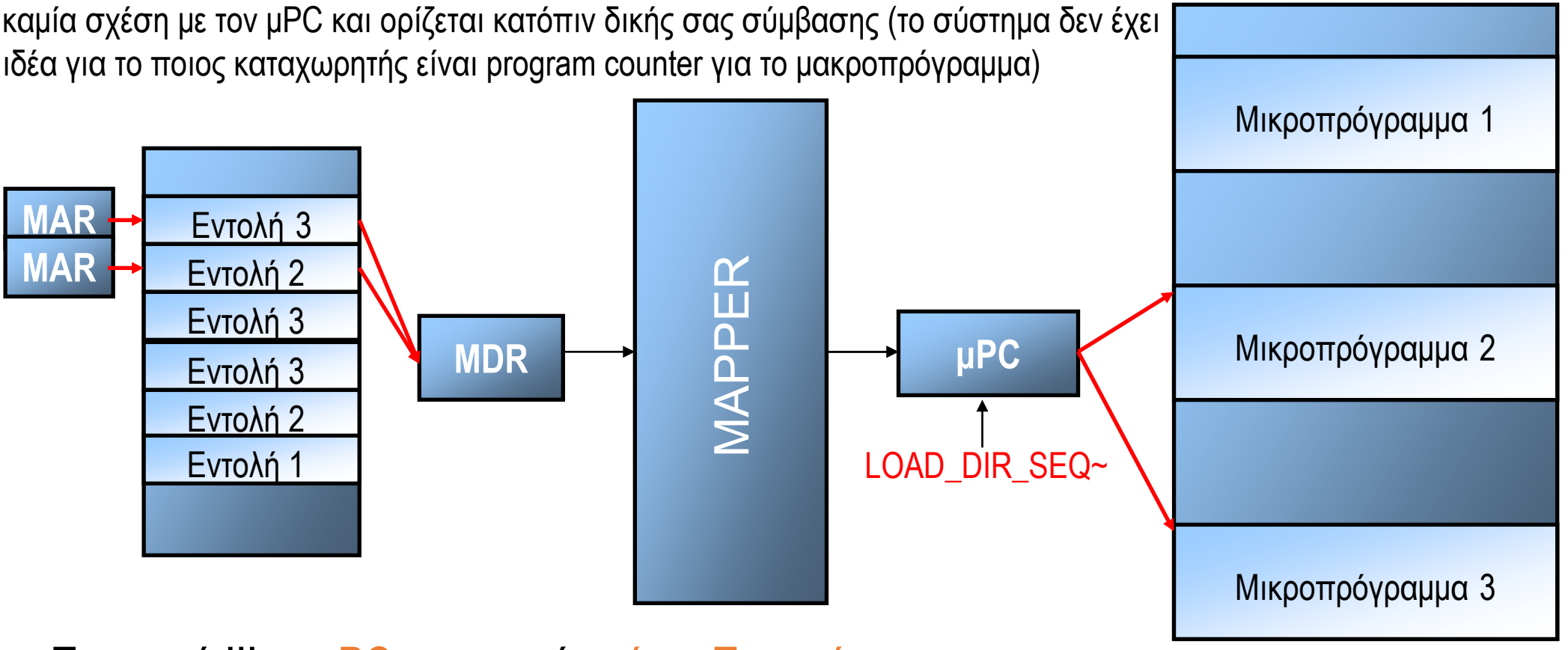
Opcode Μακροεντολής ➔ Αρχική διεύθυνση μικρομνήμης

Μνήμη Αντιστοίχισης (Mapper)



2 Βασικά Σημεία

1. Ο **MAR**, ελέγχεται σε επίπεδο **μικροπρογράμματος**. Είναι δική σας μέριμνα η αύξησή του ώστε να δείχνει στην επόμενη εντολή ! Πρακτικά, αυτό επιτυγχάνεται ορίζοντας έναν από τους καταχωρητές του συστήματος ως **PC** (program counter). Ο PC αυτός δεν έχει καμία σχέση με τον μPC και ορίζεται κατόπιν δικής σας σύμβασης (το σύστημα δεν έχει ιδέα για το ποιος καταχωρητής είναι program counter για το μακροπρόγραμμα)

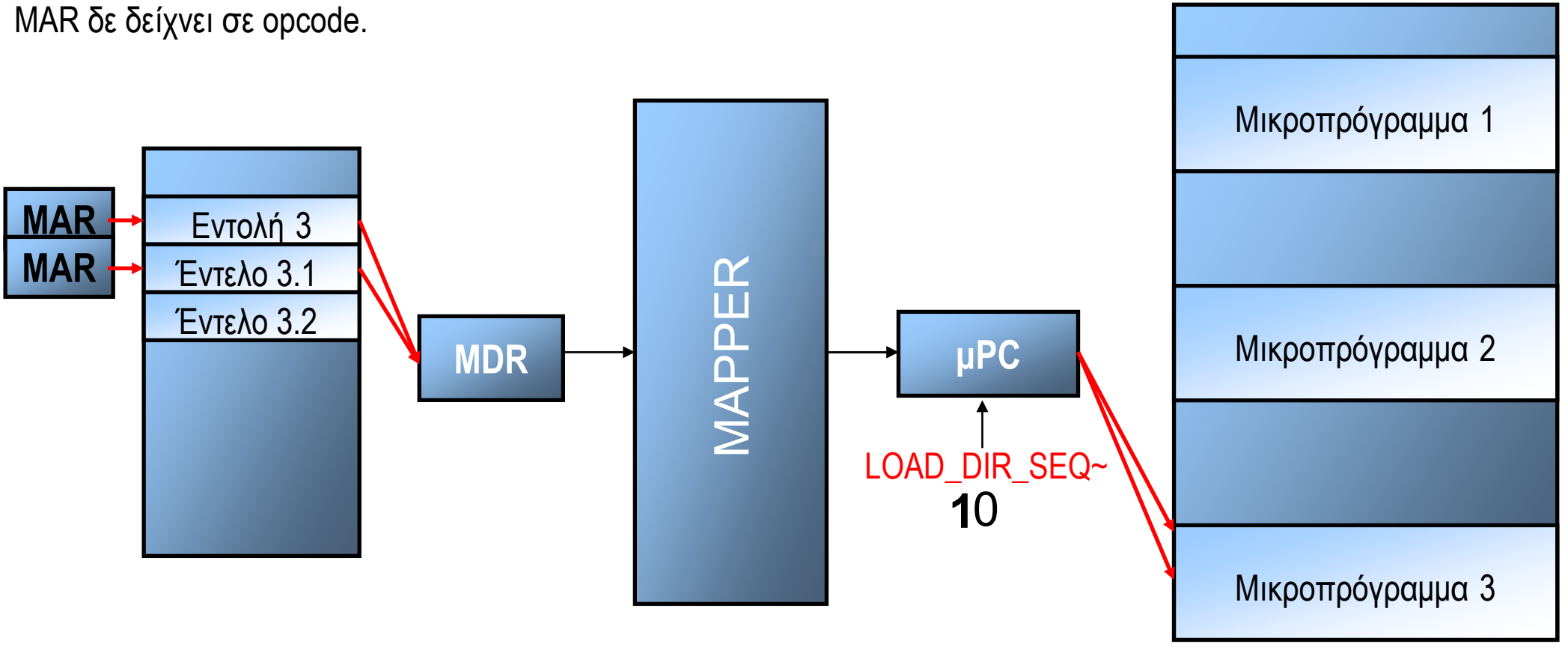


Προσοχή !!! PC = μετρητής μάκροΠρογράμματος

μPC = μετρητής μίκροΠρογράμματος

2 Βασικά Σημεία

2. Γιατί χρειαζόμαστε το σήμα **LOAD_DIR_SEQ~** ; Εκτός από opcodes η κύρια μνήμη μπορεί να περιέχει και άλλα δεδομένα (π.χ. operands, δεδομένα γενικής χρήσης). Το σήμα LOAD_DIR_SEQ~ χρησιμοποιείται για να **μην κλειδώνονται** δεδομένα στον **μPC**, όταν ο MAR δε δείχνει σε opcode.



Υπενθύμιση : Το ~ στο όνομα ενός σήματος υποδηλώνει σήμα αρνητικής λογικής. Το σήμα είναι ενεργοποιημένο όταν έχει τιμή 0

Παράδειγμα

Έστω ότι ορίζουμε το εξής σύνολο εντολών :

LDA $K : Φόρτωσε τον Accumulator (ένας οποιοσδήποτε καταχωρητής τον οποίο βαπτίζουμε Accumulator) με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

ADD $K : Πρόσθεσε στον Accumulator το περιεχόμενο της διεύθυνσης K

STA $K : Αποθήκευσε το περιεχόμενο τον Accumulator στη θέση μνήμης με διεύθυνση K

Παράδειγμα

Εντελώς αυθαίρετα, ορίζουμε ένα μοναδικό **opcode** για κάθε μία από τις μακροεντολές :

LDA $K : 00_H

ADD $K : 01_H

STA $K : 02_H

Το μικροπρόγραμμα για κάθε μακροεντολή ξεκινάει από τη διεύθυνση (μικρομνήμης) :

LDA $K : 10_H

ADD $K : 16_H

STA $K : $1C_H$

Περιεχόμενα Mapper

ΔΙΕΥΘΥΝΣΗ	ΔΕΔΟΜΕΝΑ
00000000	00010000
00000001	00010110
00000010	00011100

Παράδειγμα

Ο προγραμματιστής γράφει στην κύρια μνήμη το εξής μακροπρόγραμμα

LDA 80_H

ADD 81_H

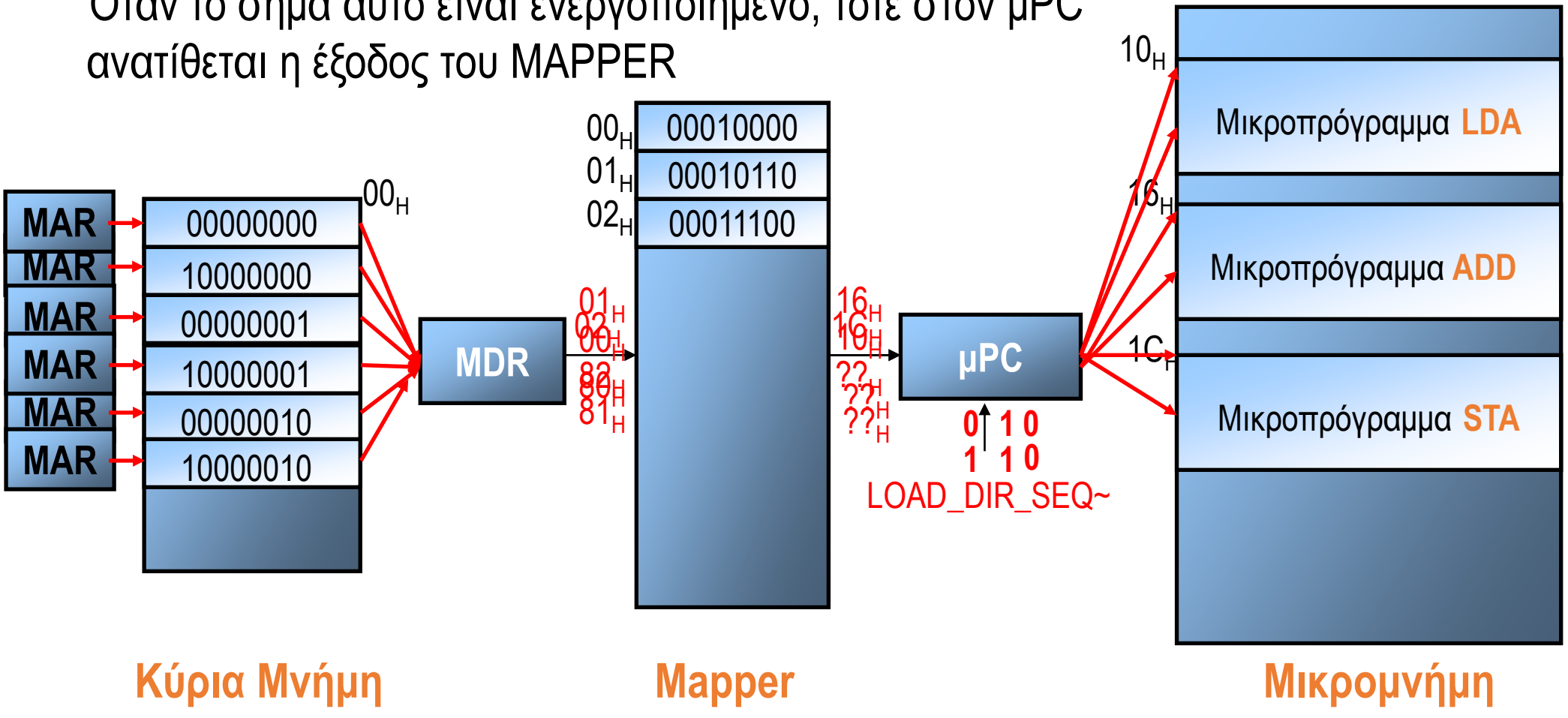
STA 82_H

Περιεχόμενα Κύριας Μνήμης

Διεύθυνση Κύριας Μνήμης	Περιεχόμενα	Εντολή (συμβολική γλώσσα)
00000000	00000000	opcode LDA
00000001	10000000	έντελο 80_H
00000010	00000001	opcode ADD
00000011	10000001	έντελο 81_H
00000100	00000010	opcode STA
00000101	10000010	έντελο 82_H

Παράδειγμα

Σημείωση : Όταν LOAD_DIR_SEQ~ είναι απενεργοποιημένο (δηλ. 1), ο μPC αυξάνεται κατά ένα (σαν κανονικός μετρητής). Όταν το σήμα αυτό είναι ενεργοποιημένο, τότε στον μPC ανατίθεται η έξοδος του MAPPER



Μικροπρογραμματιζόμενος Εκπαιδευτικός Υπολογιστής

Μικροπρογραμματιζόμενος Εκπαιδευτικός Υπολογιστής

Ίδια φιλοσοφία με το απλοϊκό σύστημα που περιγράψαμε

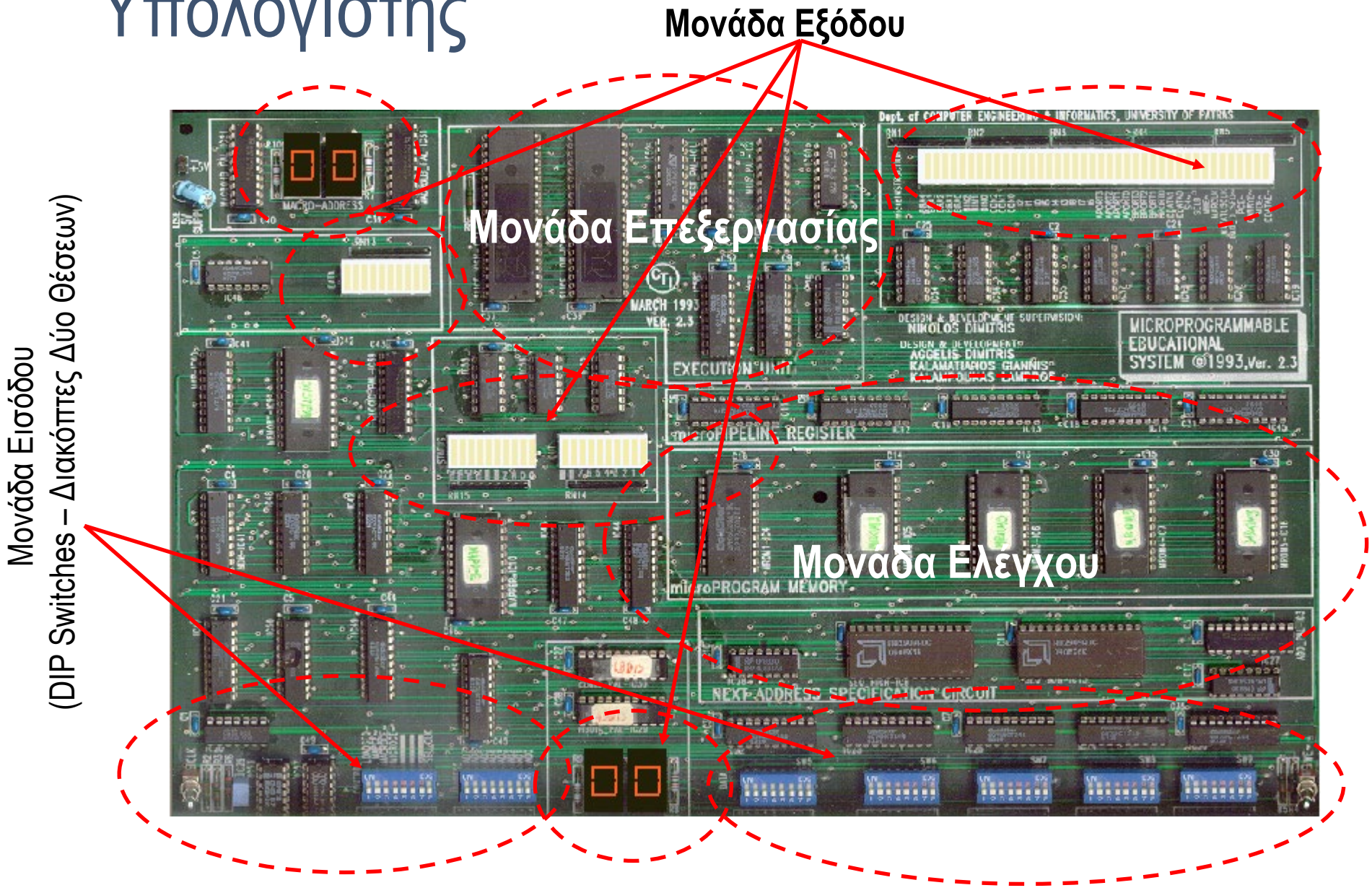
Μονάδα **ελέγχου** και μονάδα **επεξεργασίας**

Και οι δύο μονάδες έχουν **περισσότερες δυνατότητες** σε σχέση με το απλοϊκό σύστημα που περιγράψαμε

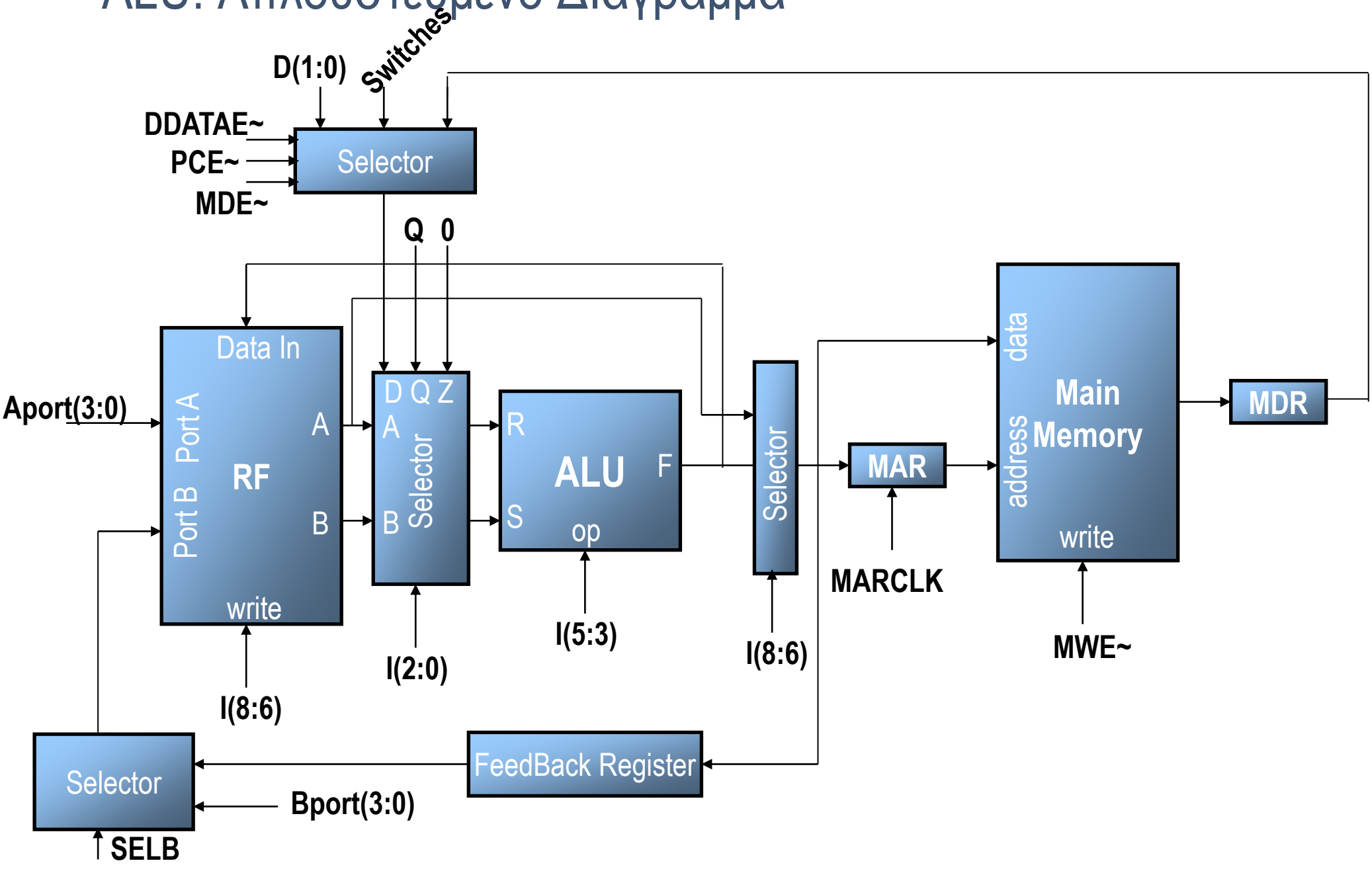
Αποτέλεσμα: περισσότερα σήματα ελέγχου

Μήκος μικροεντολής 40 bit

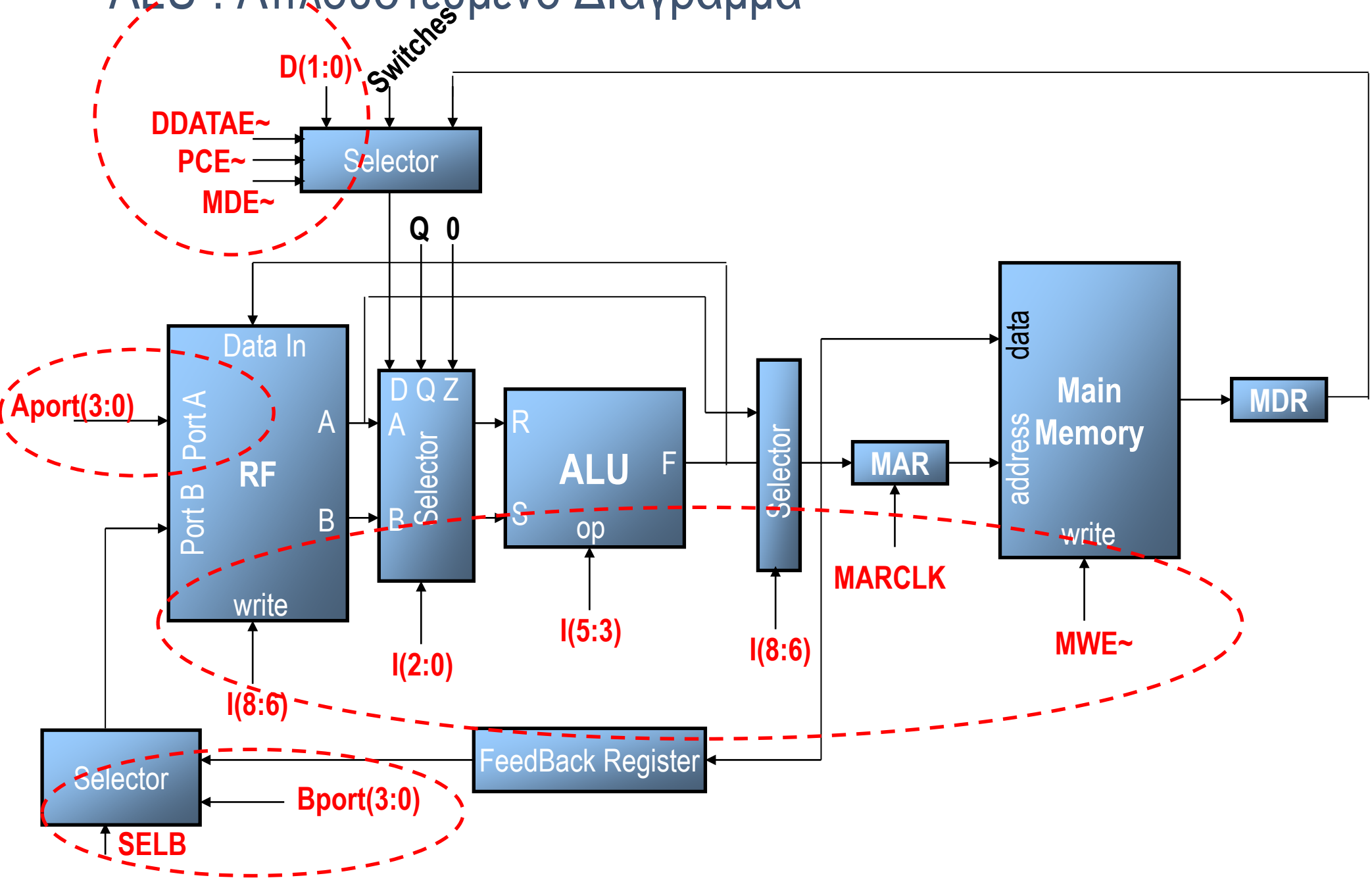
Μικροπρογραμματιζόμενος Εκπαιδευτικός Υπολογιστής



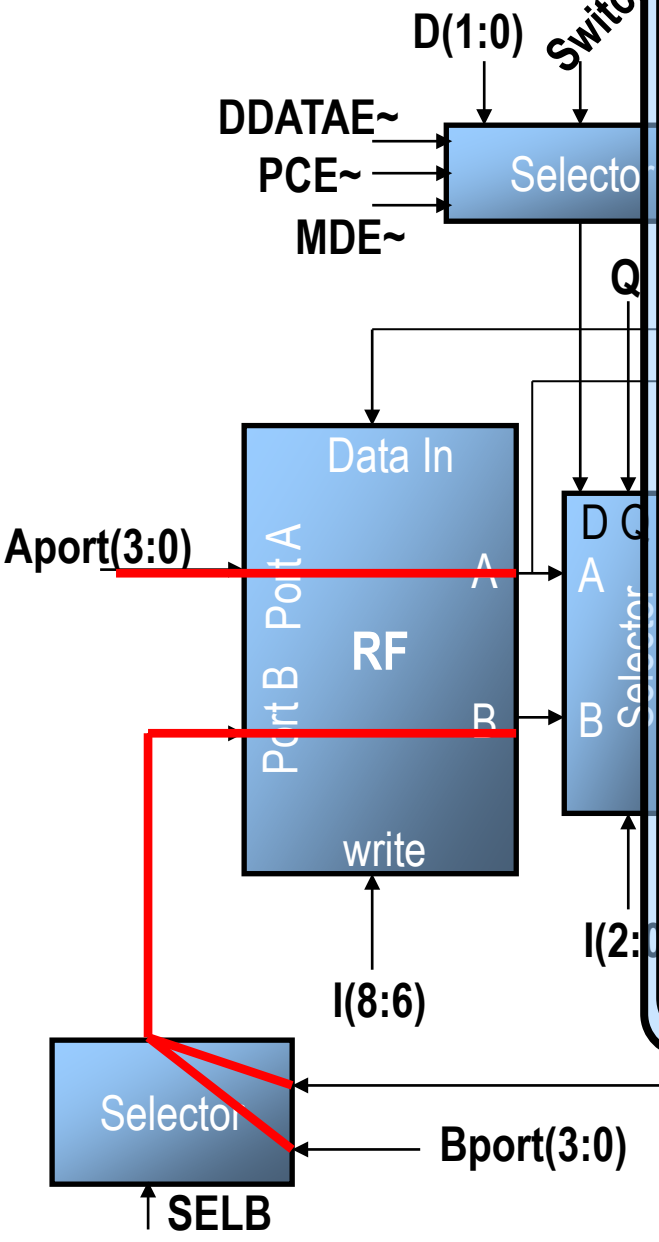
ALU: Απλουστευμένο Διάγραμμα



ALU : Απλουστευμένο Διάγραμμα



Μονάδα Επεξερ



Σήματα **Aport(3:0)** και **Bport(3:0)**, για διευθυνσιοδότηση του Register File

Τι ρόλο παίζει το σήμα **SELB**;

Αν **SELB=1** , τότε το Port B διευθυνσιοδοτείται με την τιμή του **Bport(3:0).**

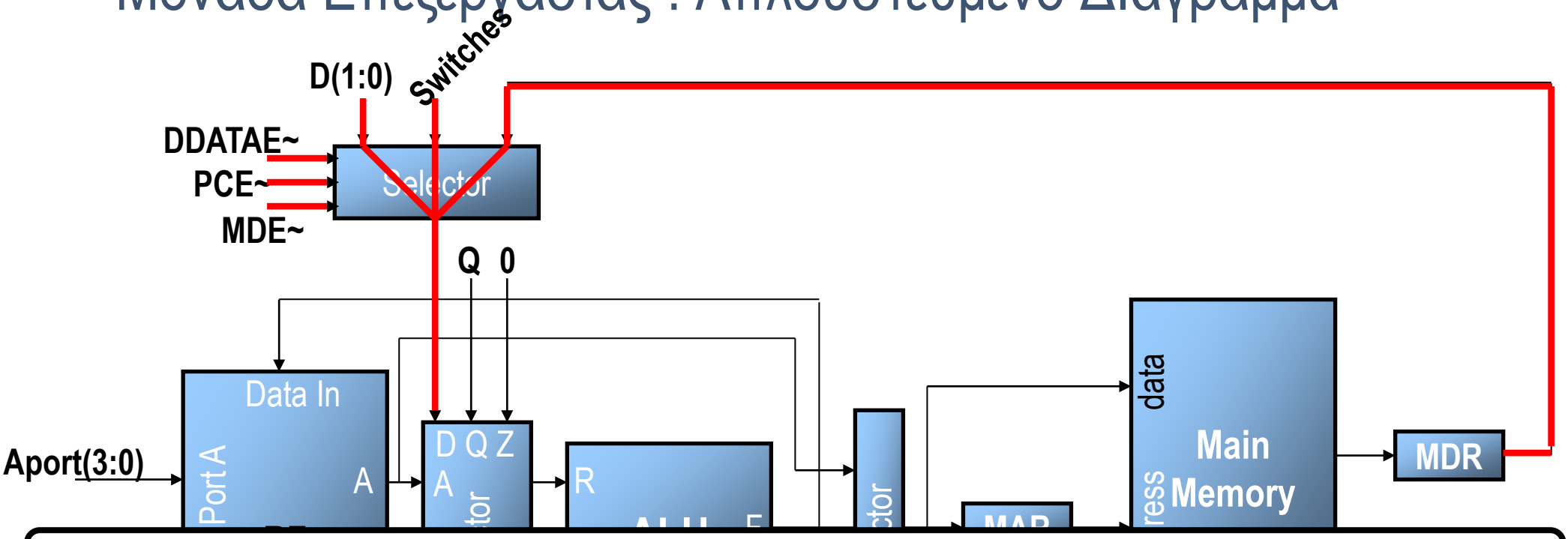
Αν **SELB=0**, τότε το Port B διευθυνσιοδοτείται με την τιμή του **Feedback Register**, ΔΗΛΑΔΗ με το αποτέλεσμα της προηγούμενης πράξης

TIP : Για τις περισσότερες περιπτώσεις αρκεί SELB = 1

Η τιμή SELB=0, είναι πολύ χρήσιμη για την υλοποίηση μακροεντολών του τύπου

ADD RX,$k

Μονάδα Επεξεργασίας : Απλουστευμένο Διάγραμμα

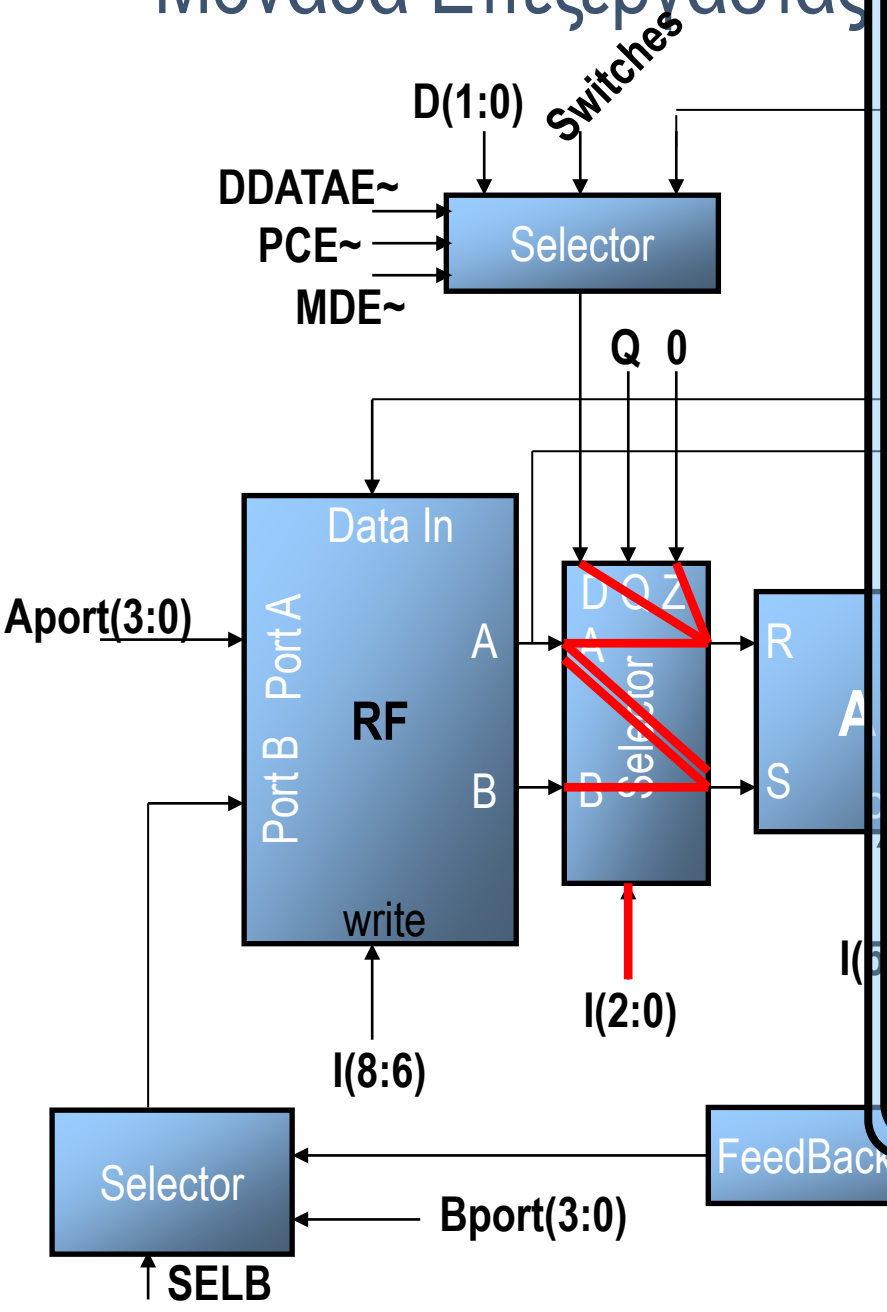


Αν **DDATAE~ = 0** , τότε απ' τον selector περνάει μία σταθερά των 2bit **D(1:0)**. Η τιμή **D(1:0)** καθορίζεται από εμάς (ξεχωριστό πεδίο στη μικροεντολή)

Αν **PCE~ = 0** , τότε απ' τον selector περνάει η τιμή που έχουν 8 **DIP Switches** της πλακέτας

Αν **MDE~ = 0** , τότε απ' τον selector περνάει η τιμή του **MDR**

Μονάδα Επεξεργασίας



I(2:0) : Επιλέγει 2 από τις 5 εισόδους του selector και τροφοδοτεί τις εισόδους R και S της ALU

Βρίσκετε την τιμή που θέλετε από τον πίνακα 1.5 (σελ. 14 του manual)

Προσοχή : Δεν υπάρχουν όλοι οι δυνατοί συνδυασμοί διαθέσιμοι !!! Π.χ. αν και υπάρχει επιλογή D και A, δεν υπάρχει επιλογή D και B

TIP : Αρκετά συνηθισμένες τιμές :

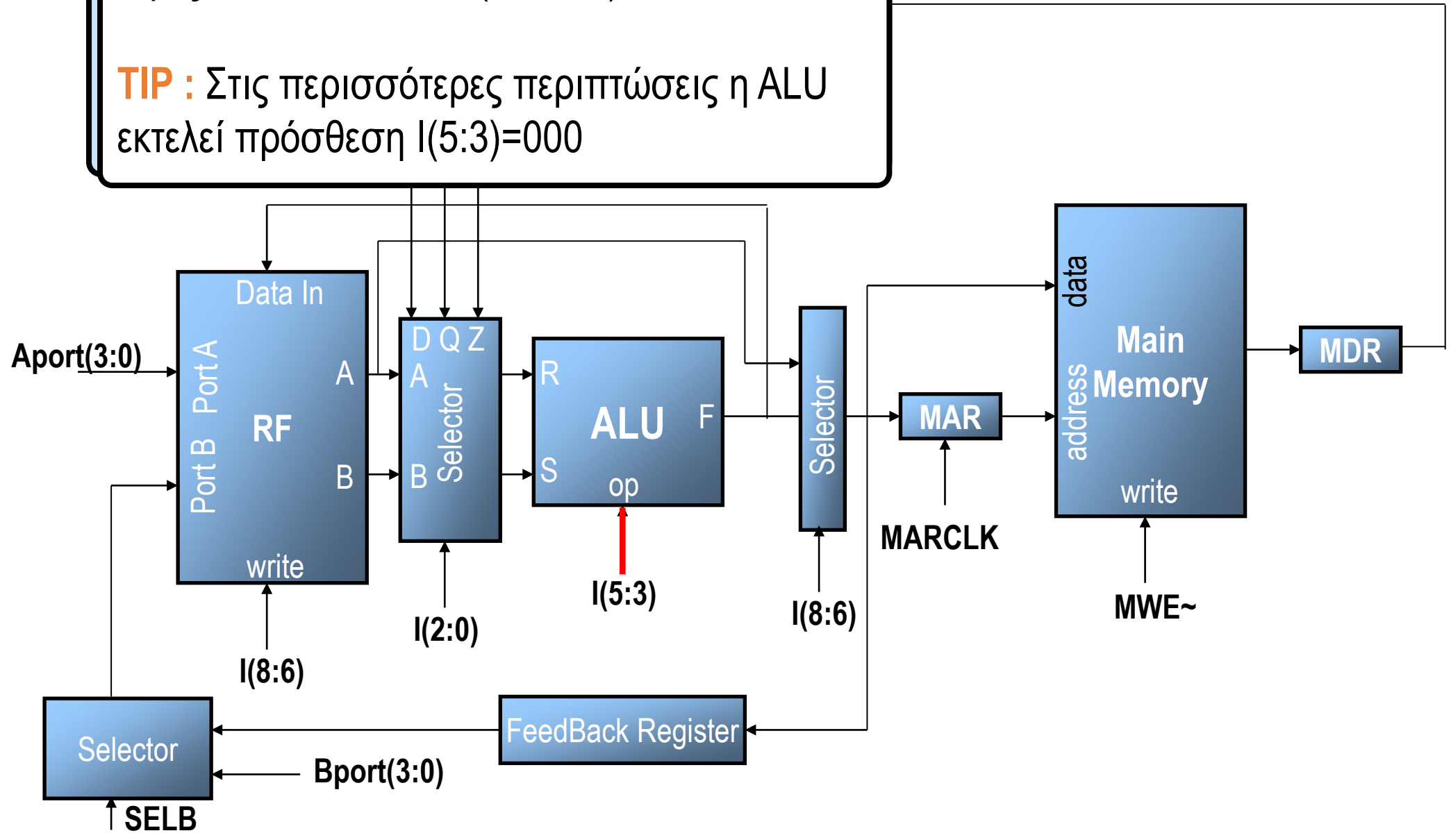
AB	001
ZA	100
DA	101
DZ	111

I(5:3) : 3 bit σήμα που καθορίζει την πράξη που θα εκτελέσει η ALU (8 επιλογές)

Τιμές από πίνακα 1.6 (σελ. 14)

TIP : Στις περισσότερες περιπτώσεις η ALU εκτελεί πρόσθεση I(5:3)=000

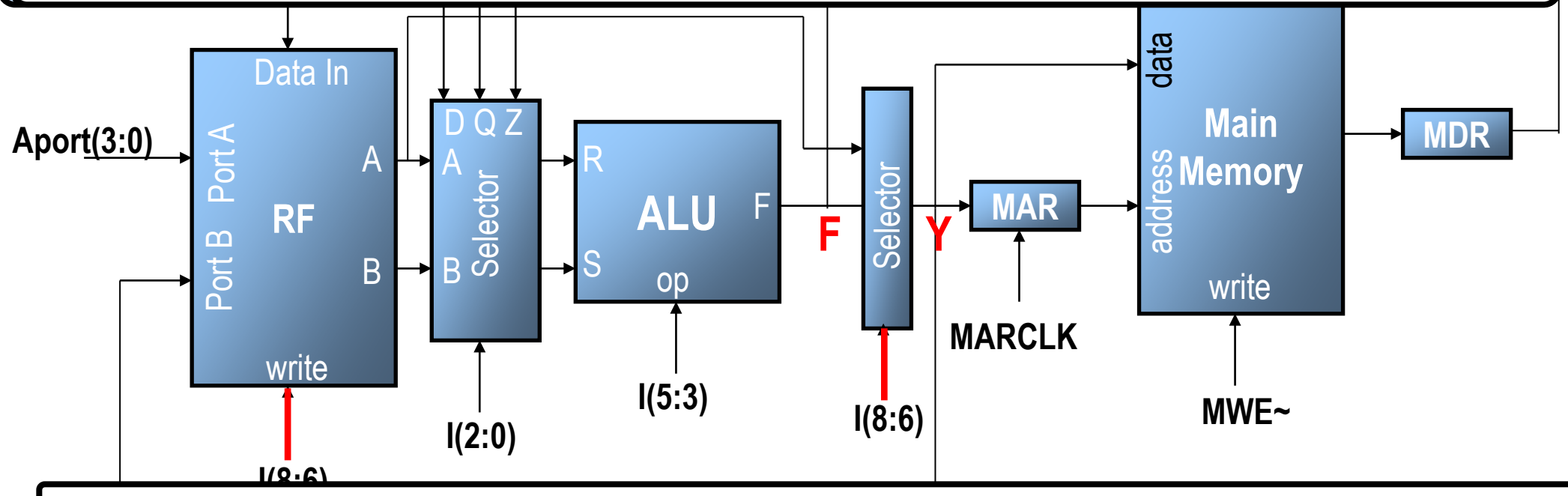
μένο Διάγραμμα



I(8:6) : 3 bit σήμα που καθορίζει (α.) Τι θα περάσει στο σημείο **Y**
(β.) Αν θα γίνει εγγραφή στο Register File
(γ.) Αν θα γίνει εγγραφή στον Q register
(δ.) Αν θα εφαρμοστεί shift πριν την αποθήκευση

Τιμές από πίνακα 1.7 (σελ. 16)

TIP : Για τις περισσότερες των περιπτώσεων επιλέγετε μία από τις
NOP (001) : Καμία εγγραφή, κανένα shift , ο selector επιλέγει το F για την έξοδο Y
RAMF (011) : Γράψε το F στο register file, όχι εγγραφή στον Q, κανένα shift, το F περνάει στο Y

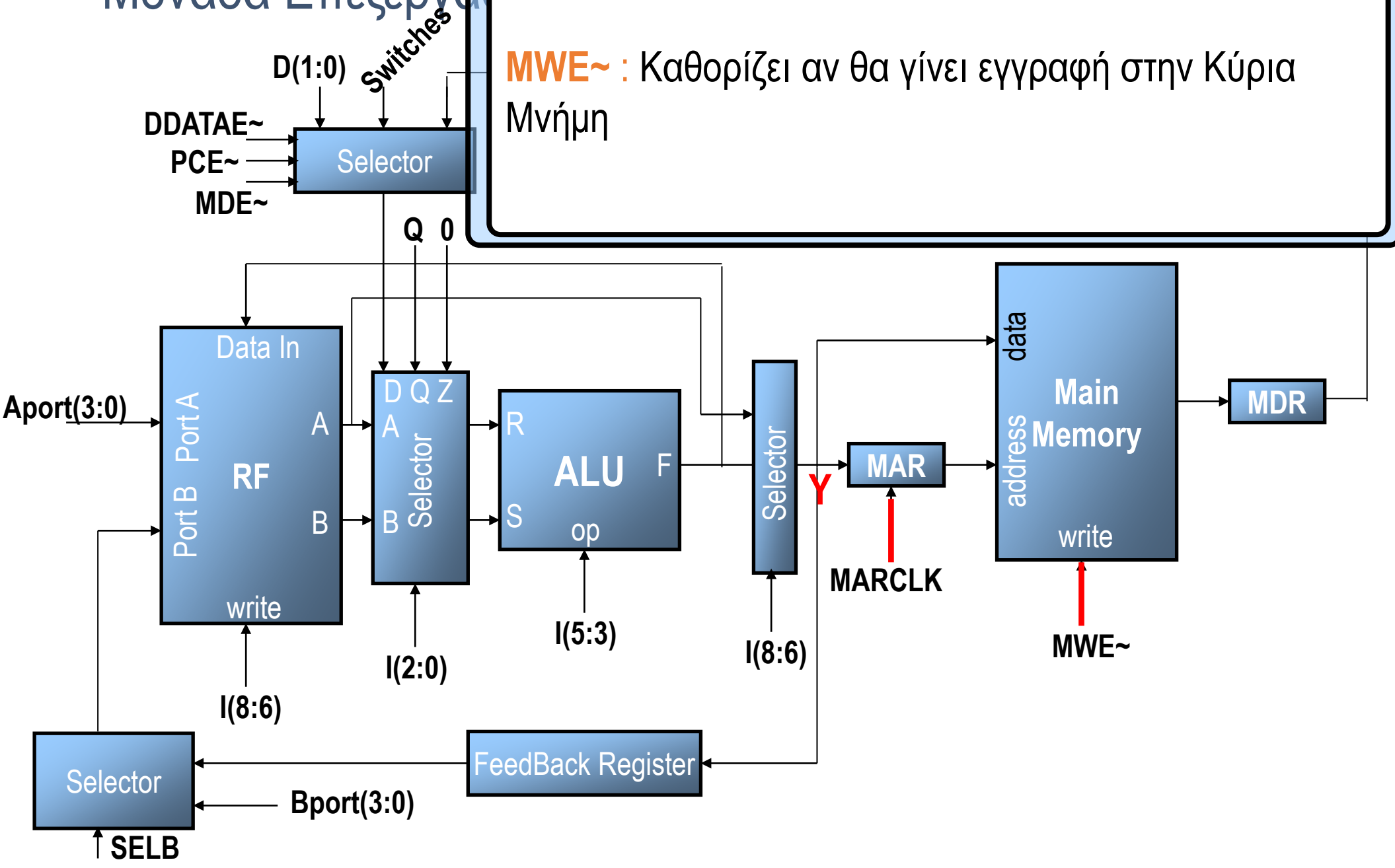


Προσοχή : Στον πίνακα 1.7 (σελ. 16), όπου αναφέρεται **RAM**, υπονοείται το **Register File** και όχι η **Κύρια Μνήμη**. Ο συμβολισμός F->B, σημαίνει ότι το F γράφεται στη θέση του Register File που διευθυνσιοδοτείται από το Port B

Μονάδα Επεξεργα

MARCLK : Καθορίζει αν το Y θα ‘κλειδωθεί’ στον MAR

MWE~ : Καθορίζει αν θα γίνει εγγραφή στην Κύρια Μνήμη



Μονάδα Επεξεργασίας

Άλλα σήματα ελέγχου που σχετίζονται με τη **μονάδα επεξεργασίας**

SH~ : Σε συνδυασμό με το **I(8:6)**, καθορίζει αν θα γίνει απλή ή κυκλική ολίσθηση στα αποτελέσματα (Πίνακας 1.9 σελ. 21, Σχήματα 1.7 και 1.8 σελ. 19)

TIP : Στις περισσότερες περιπτώσεις δεν χρειάζεστε ολίσθηση οπότε SH~=1

CARRYE~ (carry enable): Καθορίζει αν η ALU θα εκτελέσει πράξη με κρατούμενο εισόδου (Πίνακας 1.8 – σελ. 20)

TIP : Στις περισσότερες περιπτώσεις δεν χρειάζεται κρατούμενο εισόδου οπότε CARRYE~ = 1

Μονάδα Επεξεργασίας

MSTATUSCLK : Η ALU μετά από κάθε πράξη που εκτελεί, ενημερώνει και 4 flags : microC (microCarry), microO (microOverflow), microS (microSign) και microZ (microZero). Τα micro-flags, ενημερώνονται σε **κάθε πράξη**.

Εκτός από τα micro-flags, το σύστημα διαθέτει και macro-flags (macroC, macroO, macroS και macroZ). Αν ενεργοποιήσετε το σήμα **MSTATUSCLK** , οι τιμές των micro-flags 'κλειδώνονται' στα macro-flags.

Άρα :

Τα micro-flags ενημερώνονται πάντα

Τα macro-flags ενημερώνονται μόνο όταν θέλετε (και προφανώς σε κάποια χρονική στιγμή που έχει ιδιαίτερο νόημα για το μακροπρόγραμμα.)

TIP : Στις περισσότερες περιπτώσεις δεν απαιτείται η καταγραφή των flags , οπότε MSTATUSCLK = 0.

Μονάδα Ελέγχου

Σε σχέση με τη μονάδα ελέγχου του απλοϊκού συστήματος, υπάρχουν οι εξής βελτιώσεις :

1. Δυνατότητα **διακλαδώσεων** στο μικροπρόγραμμα **με ή χωρίς συνθήκη**

2. Δυνατότητα **κλήσης / επιστροφής από ρουτίνα** (με τη βοήθεια μίας hardware στοίβας)

Κατά τα άλλα το σχήμα **κύρια μνήμη → Mapper → μPC → μικρομνήμη** , έχει διατηρηθεί.

Τα **control σήματα** είναι :

1. **LOAD_DIR_SEQ~** (το γνωστό)
2. **BRA(4:0)** , **BIN(2:0)** , **CON(2:0)**

Σήματα ελέγχου Μονάδας Ελέγχου !

LOAD_DIR_SEQ~ : Με την ενεργοποίηση του σήματος αυτού η έξοδος του mapper περνάει στον μPC (και διευθυνσιοδοτεί τη μικρομνήμη).

TIP: Ενεργοποιούμε το σήμα αυτό μόνο όταν ισχύουν τα δύο παρακάτω (και τα δύο) :

α. όλες οι λειτουργίες του τρέχοντος μικροπρογράμματος έχουν ολοκληρωθεί

β. ο MAR δείχνει στο opcode της επόμενης προς εκτέλεση μακροεντολής

Μετά την ενεργοποίηση ο έλεγχος θα περάσει στο μικροπρόγραμμα που αντιστοιχεί στην επόμενη μακροεντολή (του μακροπρογράμματος)

Πρακτικά : Στις περισσότερες περιπτώσεις η τελευταία εντολή κάθε μικροπρογράμματος, θα έχει ενεργοποιημένο το LOAD_DIR_SEQ~ και δεν θα εκτελεί καμία πράξη στη μονάδα επεξεργασίας.

Σήματα ελέγχου Μονάδας Ελέγχου !

BIN(2:0) : (σελ.34. Πίνακας 2.2) Καθορίζει αν η μικροεντολή είναι

α. Εντολή διακλάδωσ

β. Εντολή διακλάδωσ

γ. Κλήση υπορουτίνα

δ. Επιστροφή από υπ

ε. Σειριακή μικροεντολ

TIP : Συνήθως οι μικροεντολές που γράφετε θα είναι σειριακές οπότε οι τιμές των πεδίων αυτών είναι

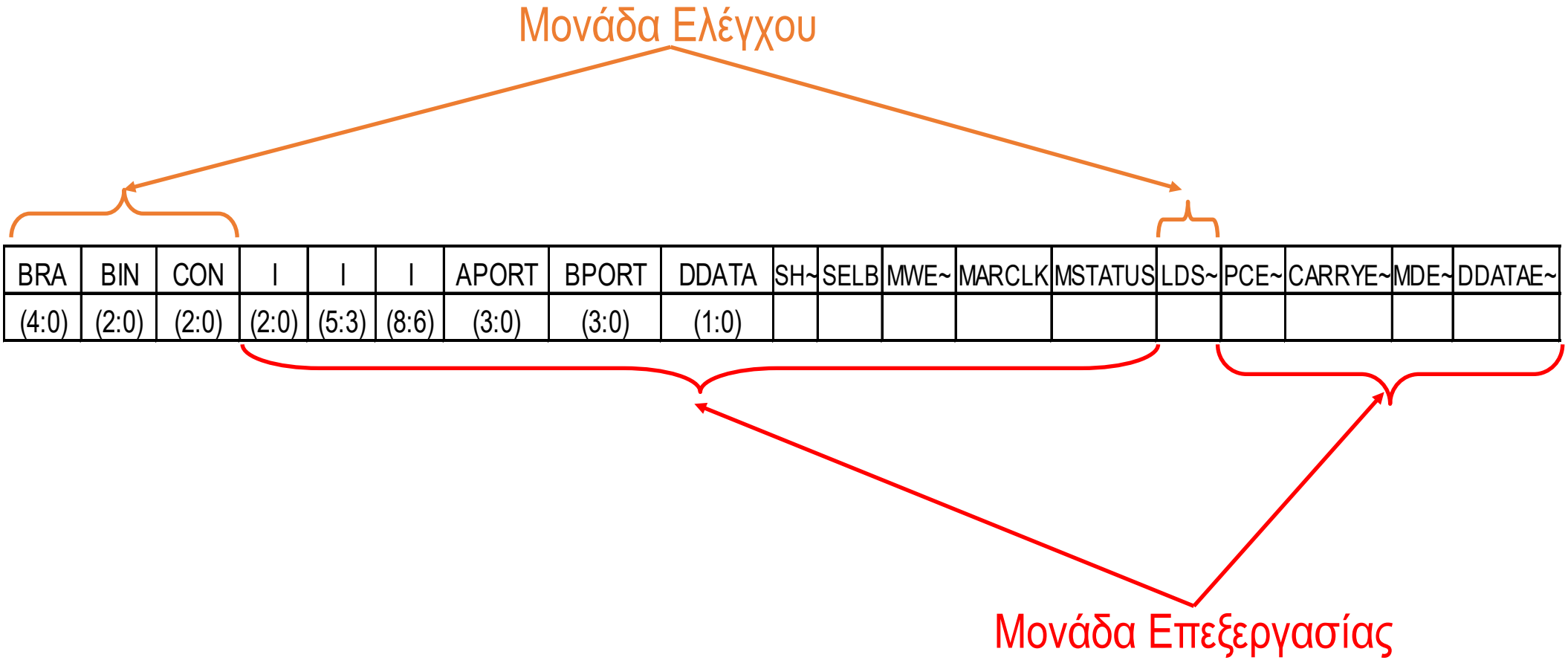
BRA(4:0) : XXXXX
BIN(2:0) : 000
CON(2:0): XXX

CON(2:0) : (σελ. 35, Πίνακας 2.3) Καθορίζει ποια είναι η συνθήκη για conditional εντολές (Π.χ Branch if macroCarry ή Branch if microCarry)

BRA(4:0) : 5-bit πεδίο το οποίο προστίθεται (2's complement, εύρος [-16,15)) στη διεύθυνση της τρέχουσας μικροεντολής για την εύρεση της διεύθυνσης της επόμενης μικροεντολής (σημ: μόνο αν πρόκειται για εντολή που αλλάζει τη ροή π.χ. JMP, taken branch κλπ.)

Οργάνωση Μικροεντολής

Η μικροεντολή περιέχει **όλα** τα πεδία για τα οποία μιλήσαμε μέχρι τώρα



Πώς γράφουμε μικροπρογράμματα ;

Πώς γράφουμε μικροπρογράμματα ;

Βήμα 1 : Σπάμε την πράξη σε στοιχειώδεις λειτουργίες που να μπορούν να υλοποιηθούν από τη μονάδα επεξεργασίας που διαθέτουμε.

Βήμα 2 : Γράφουμε τη μικροεντολή για κάθε στοιχειώδη λειτουργία (δηλ. γράφουμε τη 40άδα)

TIP : Καλό είναι να γράφετε αρχικά τις στοιχειώδεις εντολές σε ψευδογλώσσα. Π.χ. μερικές στοιχειώδεις λειτουργίες μπορεί να συμβολιστούν,

(0001)+(0010)→(0010)
(0001)+(0010)→NOP, MAR
PC + 1 → PC , MAR // σημ.: Θα πρέπει πρώτα να έχουμε πει ποιος είναι ο PC
MDR + ACC → ACC // >> >> >> ACC
ACC + 0 → NOP, MWE

Παράδειγμα (LDA $K – Βήμα 1)

LDA $K : Φόρτωσε τον Accumulator (ένας οποιοσδήποτε καταχωρητής τον οποίο βαπτίζουμε Accumulator) με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

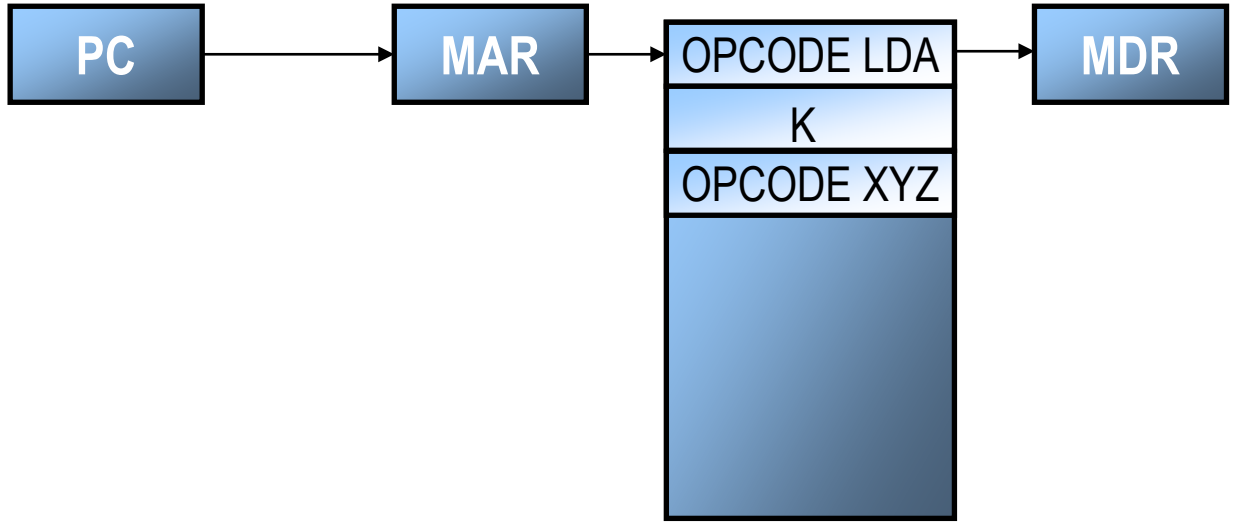
Ορίζουμε αυθαίρετα έναν καταχωρητή ως PC (τον PC τον χρειαζόμαστε πάντα) και έναν ως ACC (επειδή ζητείται από τη σημασιολογία της εντολής)

Έστω λοιπόν : **PC = 0100** και **ACC = 1000**

Παράδειγμα (LDA $K – Βήμα 1)

LDA $K : Φόρτωσε τον Accumulator με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

Με το που καλείται μία LDA $K από το μακροπρόγραμμα, η κατάσταση στην κύρια μνήμη θα έχει ως εξής :

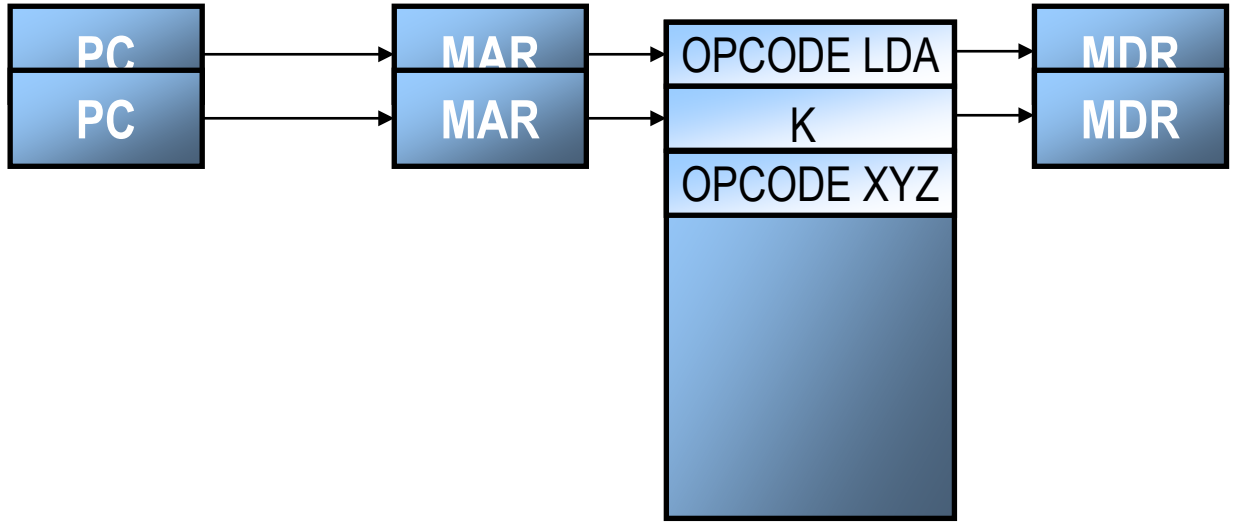


Έτσι όπως έχει η κατάσταση δεν μπορούμε με κανέναν τρόπο να ‘δούμε’ το έντελο K. Τι πρέπει να κάνουμε ;;;;;

Παράδειγμα (LDA $K – Βήμα 1)

LDA $K : Φόρτωσε τον Accumulator με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

PC + 1 → PC , MAR

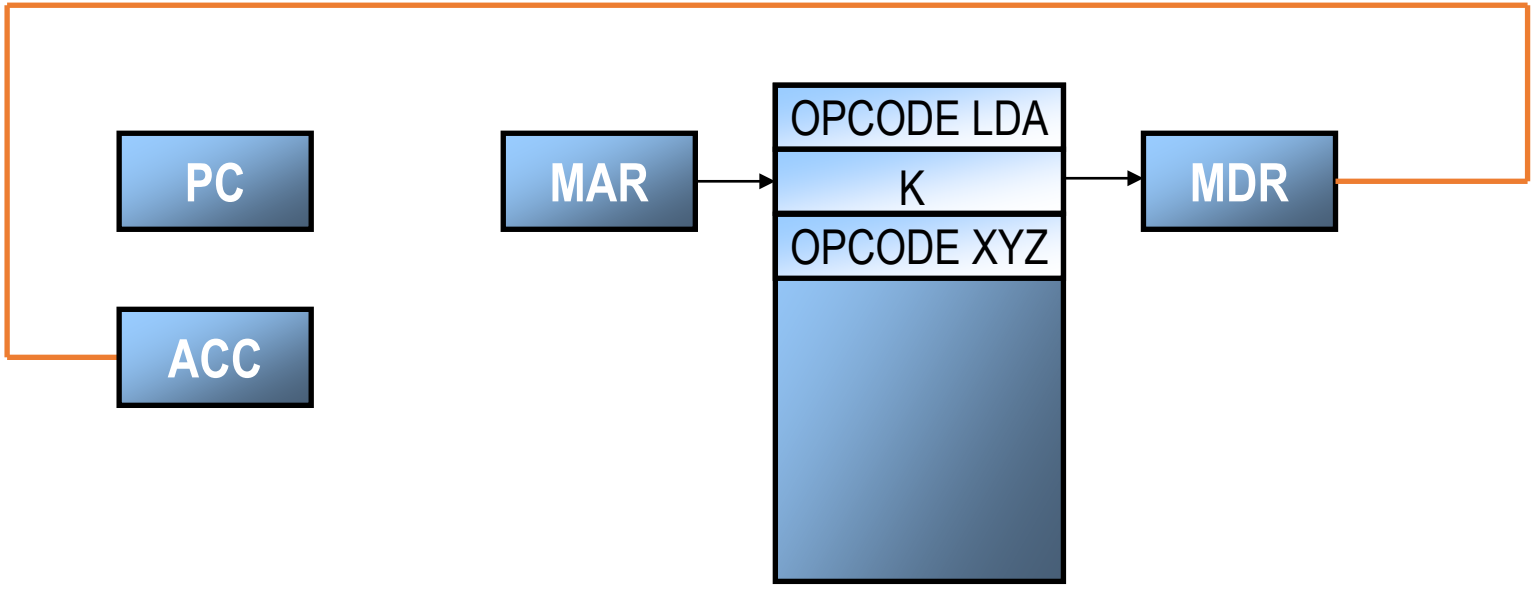


OK !!! Τώρα έχουμε την τιμή K στον MDR. Προφανώς, για να διευθυνσιοδοτήσουμε τη θέση μνήμης K (όπως απαιτεί ο ορισμός της LDA) , θα πρέπει η τιμή K με κάποιον τρόπο να έρθει στον MAR. Πώς ;;;;

Παράδειγμα (LDA $K – Βήμα 1)

LDA $K : Φόρτωσε τον Accumulator με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

MDR + 0 → ACC

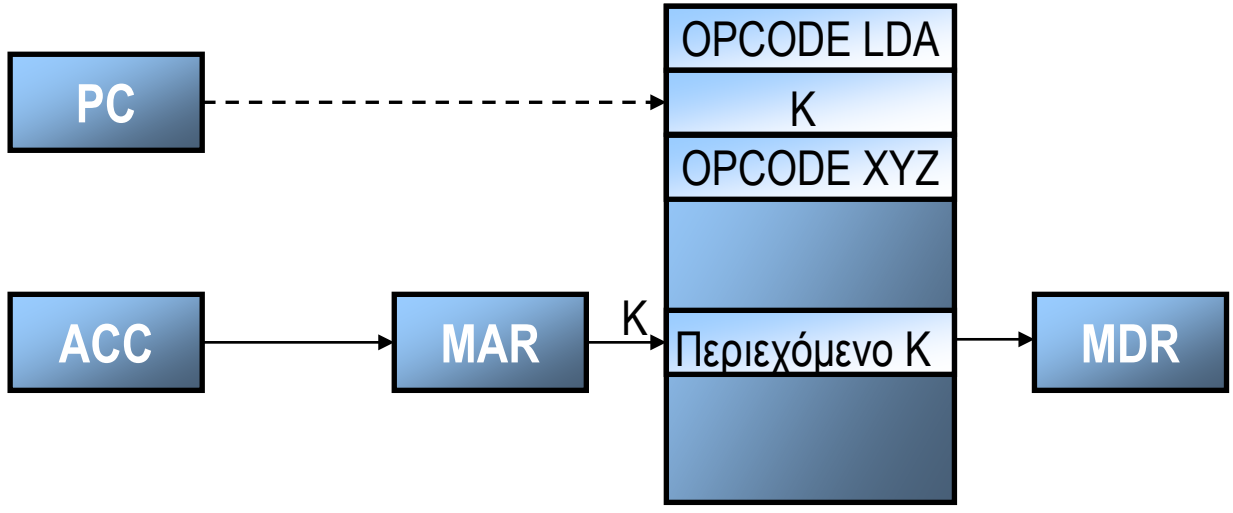


OK !!! Τώρα το K ήρθε στον accumulator. Εγώ όμως το θέλω στον MAR !!!

Παράδειγμα (LDA $K – Βήμα 1)

LDA $K : Φόρτωσε τον Accumulator με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

ACC + 0 → NOP, MAR

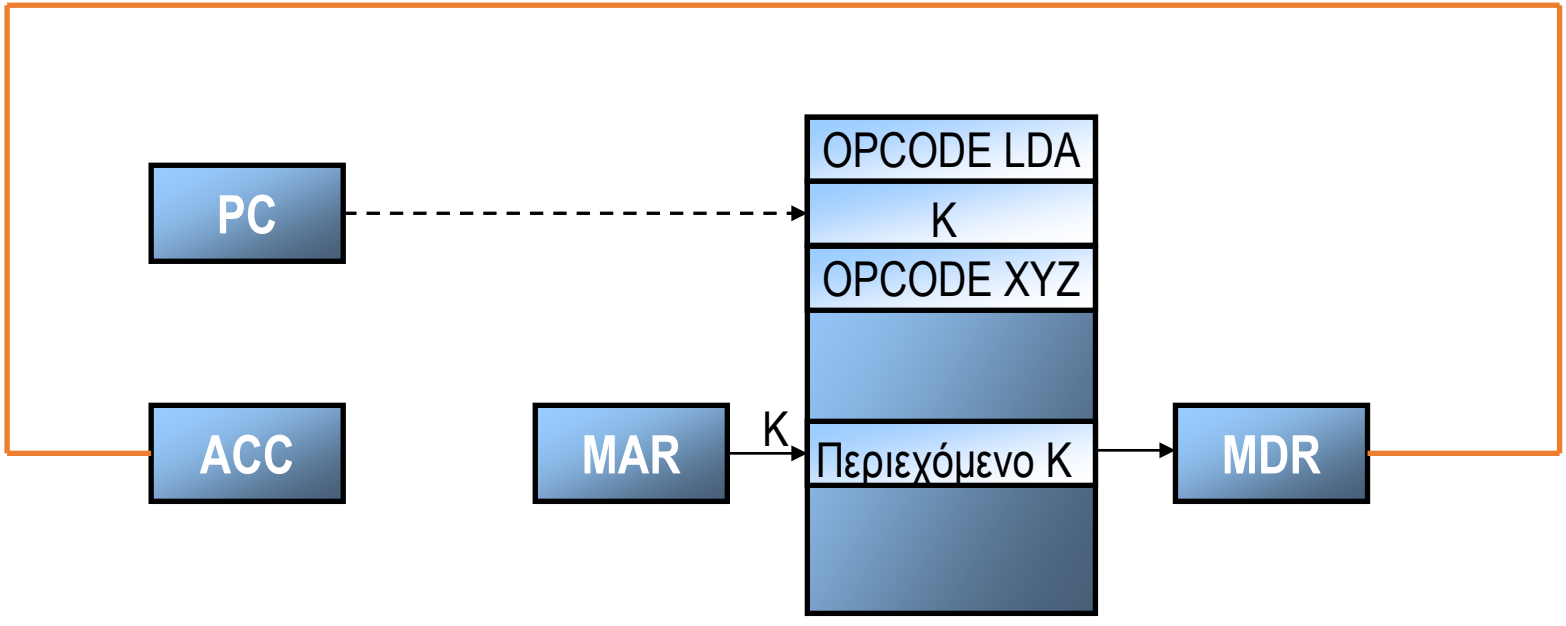


Τώρα η σωστή τιμή βρίσκεται στον MDR. Το μόνο που μένει είναι να την αποθηκεύσω στον accumulator. Πώς ;;;

Παράδειγμα (LDA $K – Βήμα 1)

LDA $K : Φόρτωσε τον Accumulator με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

MDR + 0 → ACC



Τώρα η λειτουργία της LDA $K έχει ολοκληρωθεί. Το περιεχόμενο της K ήρθε στον accumulator. Τελειώσαμε ;;;;;

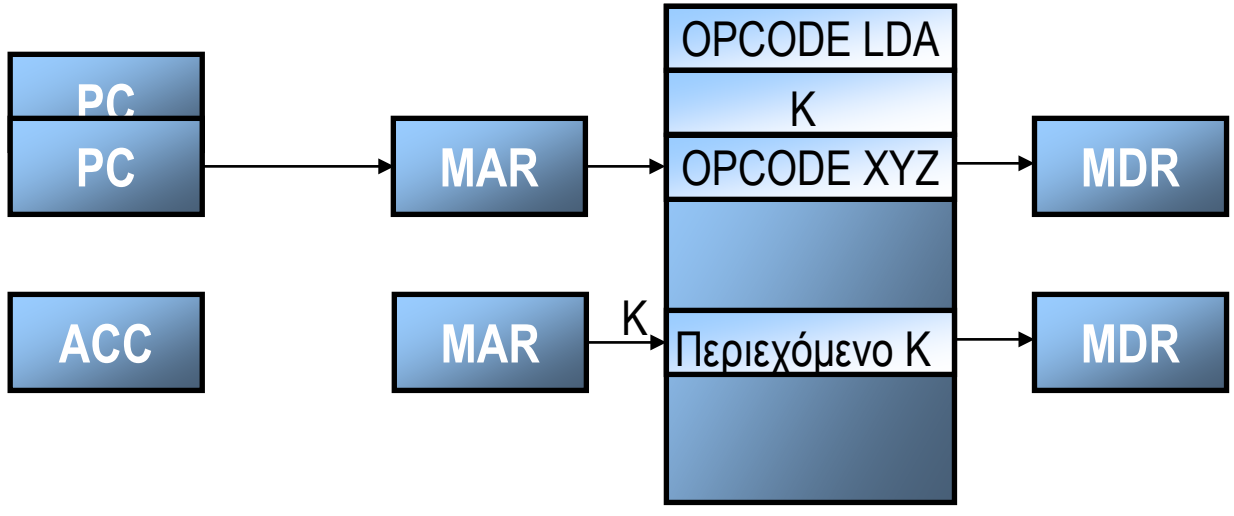
Παράδειγμα (LDA $K – Βήμα 1)

LDA $K : Φόρτωσε τον Accumulator με το περιεχόμενο της διεύθυνσης K (της κύριας μνήμης)

Θα πρέπει να δώσουμε τον έλεγχο στο μικροπρόγραμμα της επόμενης μακροεντολής (XYZ στην περίπτωσή μας). Ο PC δείχνει ακόμα στο έντελο K, οπότε κάνουμε :

PC + 1 → PC , MAR

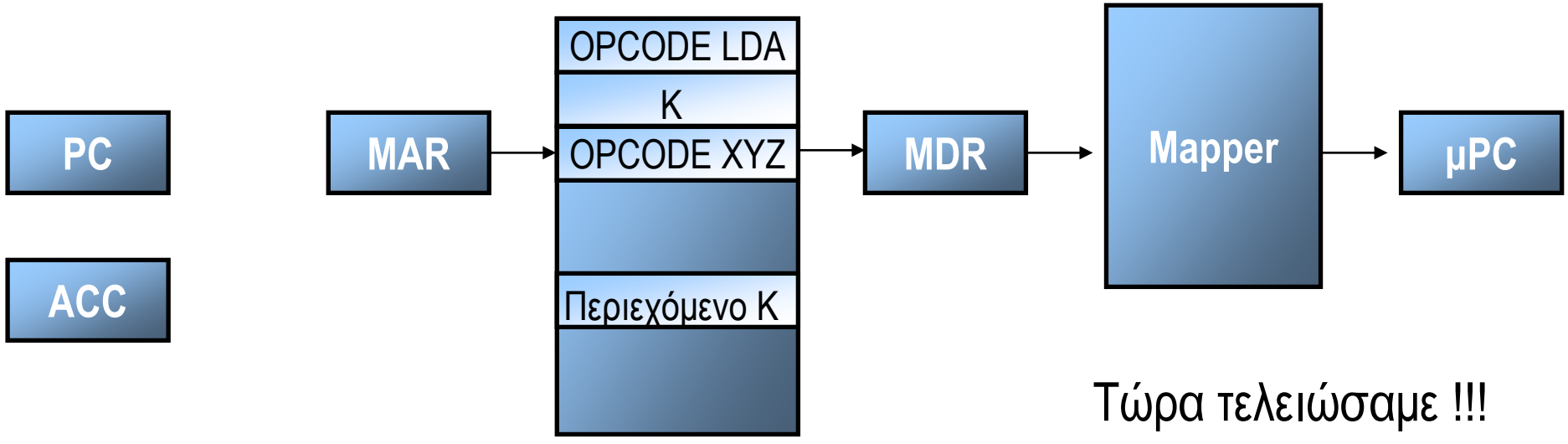
Τελειώσαμε;;;



Παράδειγμα (LDA $K – Βήμα 1)

Το μόνο που απομένει είναι να περάσουμε το opcode της XYZ στον mapper και από εκεί στον μPC. Με τον τρόπο αυτό θα συνεχίσουμε την εκτέλεση από το κατάλληλο μικροπρόγραμμα. Αυτό γίνεται μέσω του σήματος LOAD_DIR_SEQ~ :

NEXT(PC) : Συμβολικό όνομα για μία εντολή που έχει LOAD_DIR_SEQ~ = 0 και δεν κάνει τίποτα άλλο



Τώρα τελειώσαμε !!!

Παράδειγμα (LDA $K – Βήμα 1)

Συνοψίζοντας, το μικροπρόγραμμα σε συμβολική γλώσσα θα είναι

LDA $K :

1. PC + 1 → PC , MAR // Το K θα πάει στον MDR
2. MDR + 0 → ACC // Το K στον accumulator
3. ACC + 0 → NOP, MAR // Το K στον MAR, άρα το περιεχόμενο της
   // K στον MDR
4. MDR + 0 → ACC // Ο MDR (δηλ. το περιεχ. της K) στον ACC
5. PC + 1 → PC, MAR // Ο MAR θα δείξει στο opcode της επόμ. μακροεντολής
6. NEXT(PC) // LOAD_DIR_SEQ~ για να πάμε στο μικροπρόγραμμα της
   // επόμενης μακροεντολής του μακροπρογράμματος

Παράδειγμα (ADD $K – Βήμα 1)

ADD $K : Πρόσθεσε στον Accumulator το περιεχόμενο της διεύθυνσης K

LDA $K :

1. PC + 1 → PC , MAR
2. MDR + 0 → ACC
3. ACC + 0 → NOP,MAR
4. MDR + 0 → ACC
5. PC + 1 → PC, MAR
6. NEXT(PC)

ADD $K :

1. PC + 1 → PC , MAR
2. MDR + 0 → **X**
3. **X** + 0 → NOP, MAR
4. MDR + ACC → ACC
5. PC + 1 → PC , MAR
6. NEXT(PC)

Σημ: Ο X είναι ένας έξτρα καταχωρητής τον οποίο χρειάζομαι.
Η μικροεντολή 2 της ADD δεν μπορεί να γίνει ως MDR + 0 → ACC , γιατί θα καταστραφούν τα προηγούμενα δεδομένα του ACC

Παράδειγμα (STA $K – Βήμα 1)

STA $K : Αποθήκευσε το περιεχόμενο του Accumulator στη θέση μνήμης με διεύθυνση K

LDA $K :

1. PC + 1 → PC , MAR
2. MDR + 0 → ACC
3. ACC + 0 → NOP, MAR
4. MDR + 0 → ACC
5. PC + 1 → PC, MAR
6. NEXT(PC)

STA $K :

1. PC + 1 → PC , MAR
2. MDR + 0 → **X**
3. **X** + 0 → NOP,MAR
4. ACC + 0 → NOP, MWE~
5. PC + 1 → PC , MAR
6. NEXT(PC)

Σημ: Ο X είναι ένας έξτρα καταχωρητής τον οποίο χρειάζομαι.
Η μικροεντολή 2 της ADD δεν μπορεί να γίνει ως MDR + 0 → ACC , γιατί θα καταστραφούν τα προηγούμενα δεδομένα του ACC

Γενικά : Για μία εντολή που δέχεται Κ ορίσματα, θα πρέπει να κάνουμε

Standard τρόπος διευθυνσιοδότησης μνήμης, με το περιεχόμενο του MDR :

MDR + 0 → X
X + 0 → NOP, MAR

Μπορούμε να απαλείψουμε τον X ;;;;;
ΝΑΙ : MDR + 0 → NOP , MAR

Η παραπάνω εντολή είναι αποδεκτή (γλιτώνουμε έναν καταχωρητή και μία μικροεντολή)

ΠΡΟΣΟΧΗ : Δεν υποστηρίζεται εντολή του στυλ :
MDR + 0 → PC, MAR

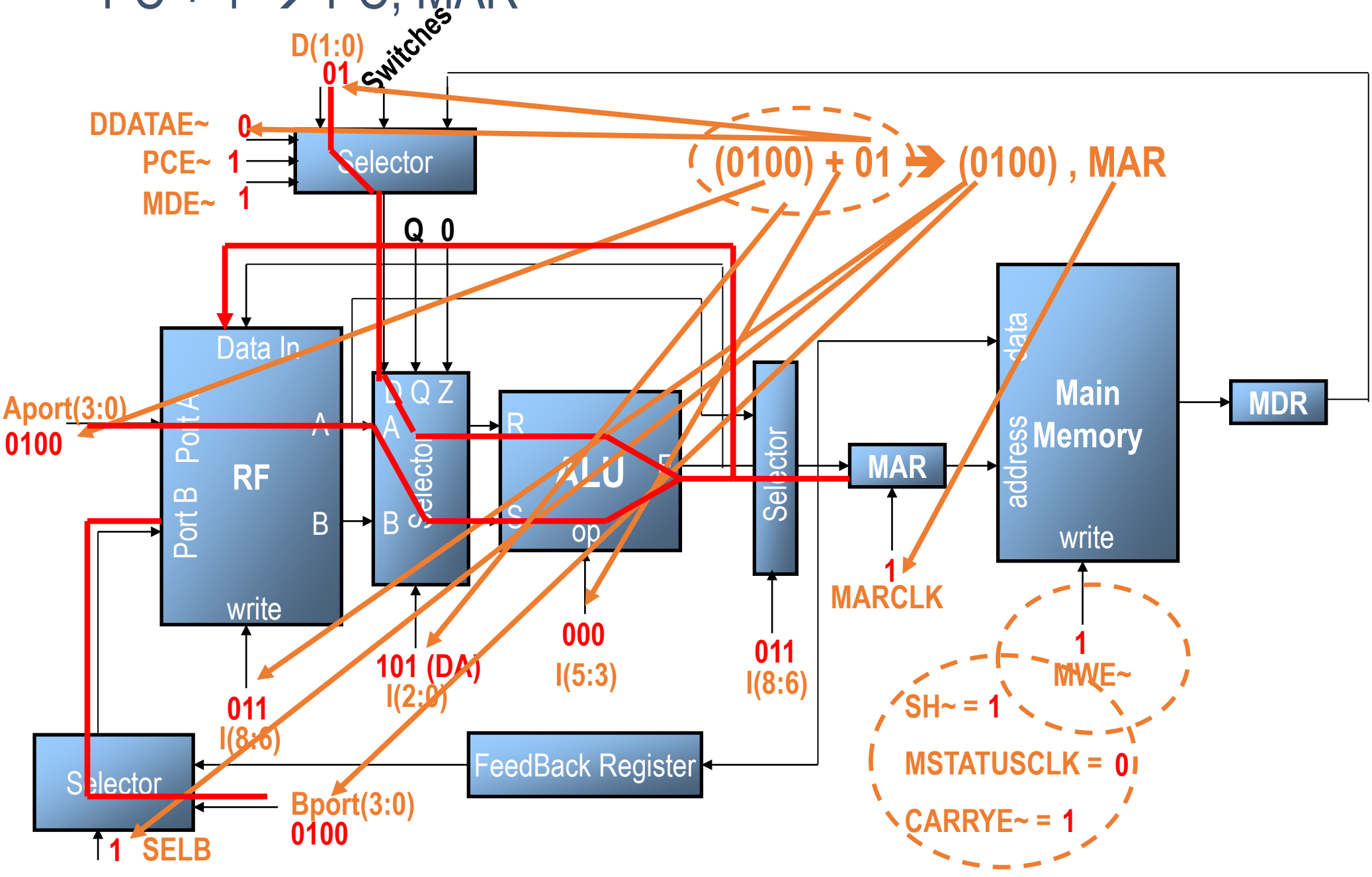
Βήμα 2 : Συγγραφή Μικροεντολών

PC = 0100 και **ACC = 1000**

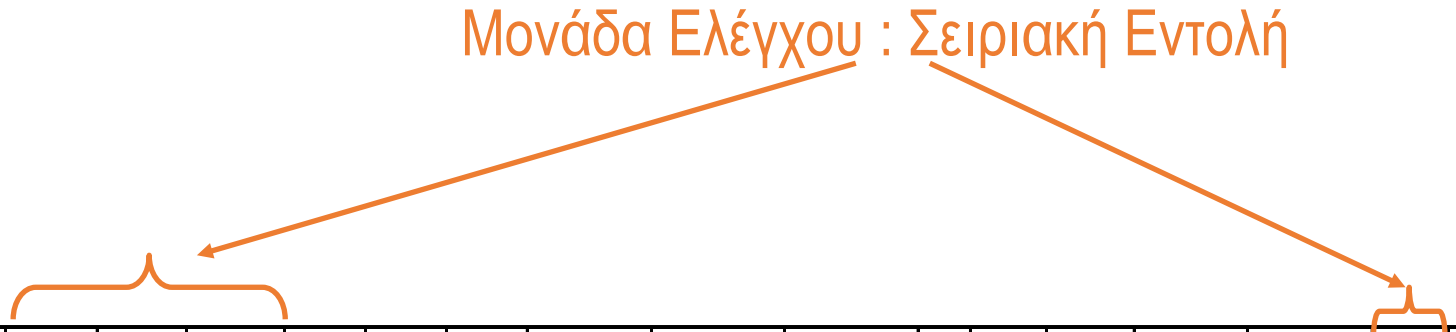
LDA $K :

1. PC + 1 → PC , MAR // Το K θα πάει στον MDR
2. MDR + 0 → ACC // Το K στον accumulator
3. ACC + 0 → NOP, MAR // Το K στον MAR, άρα το περιεχόμενο της
   // K στον MDR
4. MDR + 0 → ACC // Ο MDR (δηλ. το περιεχ. της K) στον ACC
5. PC + 1 → PC, MAR // Ο MAR θα δείξει στο opcode της επόμ. μακροεντολής
6. NEXT(PC) // LOAD_DIR_SEQ~ για να πάμε στο μικροπρόγραμμα της
   // επόμενης μακροεντολής του μακροπρογράμματος

PC + 1 → PC, MAR


D(1:0)
01
Switches
DDATAE~ 0
PCE~ 1
MDE~ 1
Selector
(0100) + 01 → (0100) , MAR
Q 0
Data In
Aport(3:0)
0100
Port A
Port B
RF
A
B
write
D Q Z
A
B
Selector
R
ALU
S
op
Selector
MAR
1
MARCLK
data
address
Main Memory
write
MDR
1
MWE~
SH~ = 1
MSTATUSCLK = 0
CARRYE~ = 1
000
I(5:3)
011
I(8:6)
101 (DA)
I(2:0)
011
I(8:6)
Selector
FeedBack Register
Bport(3:0)
0100
1 SELB

PC + 1 → PC, MAR



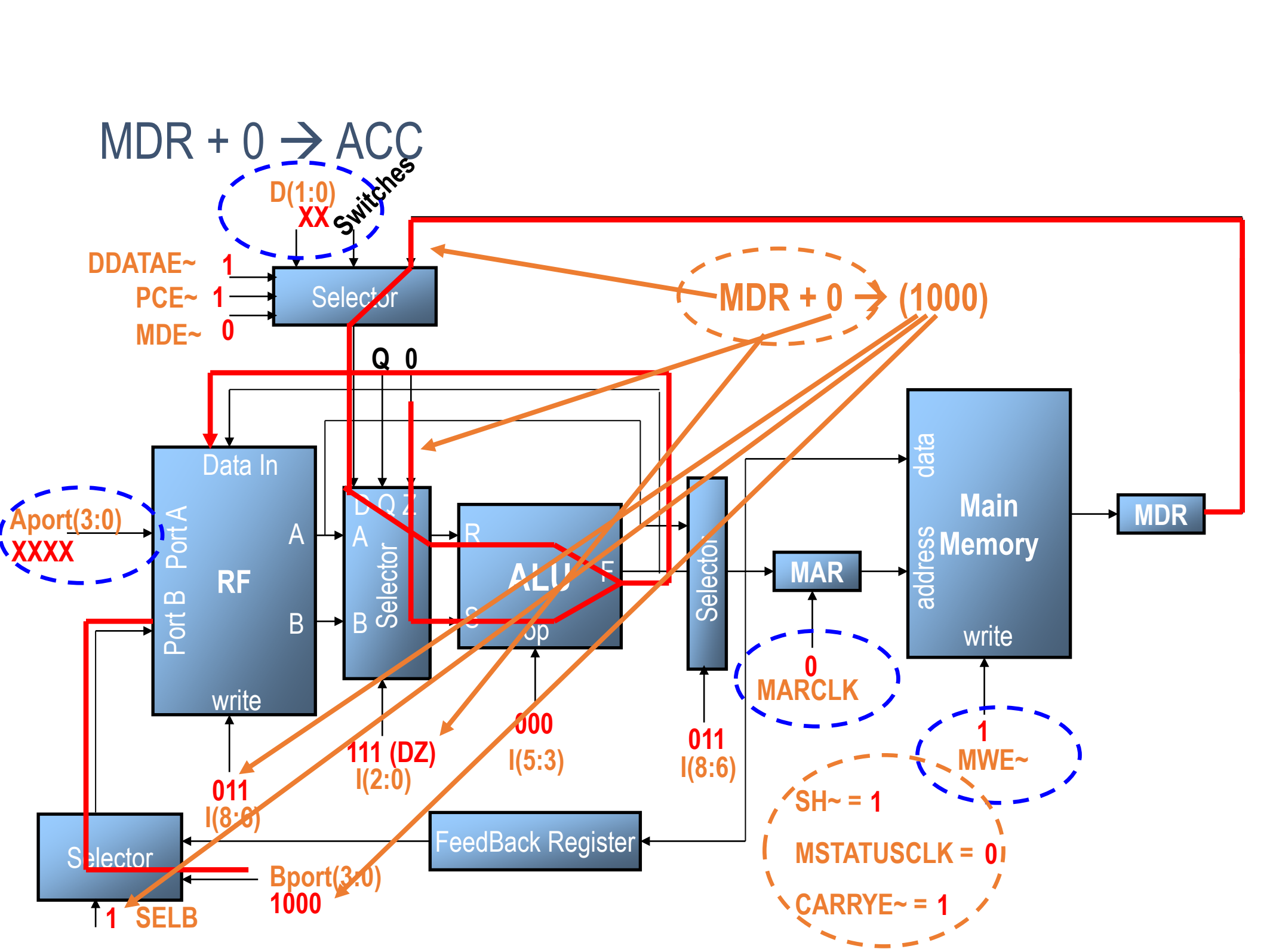
	BRA	BIN	CON	I	I	I	APORT	BPORT	DDATA	SH~	SELB	MWE~	MARCLK	MSTATUS	LDS~	PCE~	CARRYE~	MDE~	DDATAE~
	(4:0)	(2:0)	(2:0)	(2:0)	(5:3)	(8:6)	(3:0)	(3:0)	(1:0)										
PC+1->PC,MAR	xxxxx	000	xxx	101	000	011	0100	0100	01	1	1	1	1	0	1	1	1	1	0

Βήμα 2 : Συγγραφή Μικροεντολών

PC = 0100 και **ACC = 1000**

LDA $K :

1. PC + 1 → PC , MAR // Το K θα πάει στον MDR
2. MDR + 0 → ACC // Το K στον accumulator
3. ACC + 0 → MAR // Το K στον MAR, άρα το περιεχόμενο της K στον MDR
4. MDR + 0 → ACC // Ο MDR (δηλ. το περιεχ. της K) στον ACC
5. PC + 1 → PC, MAR // Ο MAR θα δείξει στο opcode της επόμ. μακροεντολής
6. NEXT(PC) // LOAD_DIR_SEQ~ για να πάμε στο μικροπρόγραμμα της
   // επόμενης μακροεντολής του μακροπρογράμματος


MDR + 0 → ACC
D(1:0)
XX
Switches
DDATAE~ 1
PCE~ 1
MDE~ 0
Selector
MDR + 0 → (1000)
Q 0
Data In
Aport(3:0)
XXXX
Port A
Port B
RF
A
B
write
Selector
A
B
R
ALU
F
op
Selector
MAR
address
data
Main Memory
write
MDR
0
MARCLK
1
MWE~
000
I(5:3)
111 (DZ)
I(2:0)
011
I(8:6)
011
I(8:6)
SH~ = 1
MSTATUSCLK = 0
CARRYE~ = 1
Selector
FeedBack Register
Bport(3:0)
1000
1 SELB

MDR + 0 → ACC

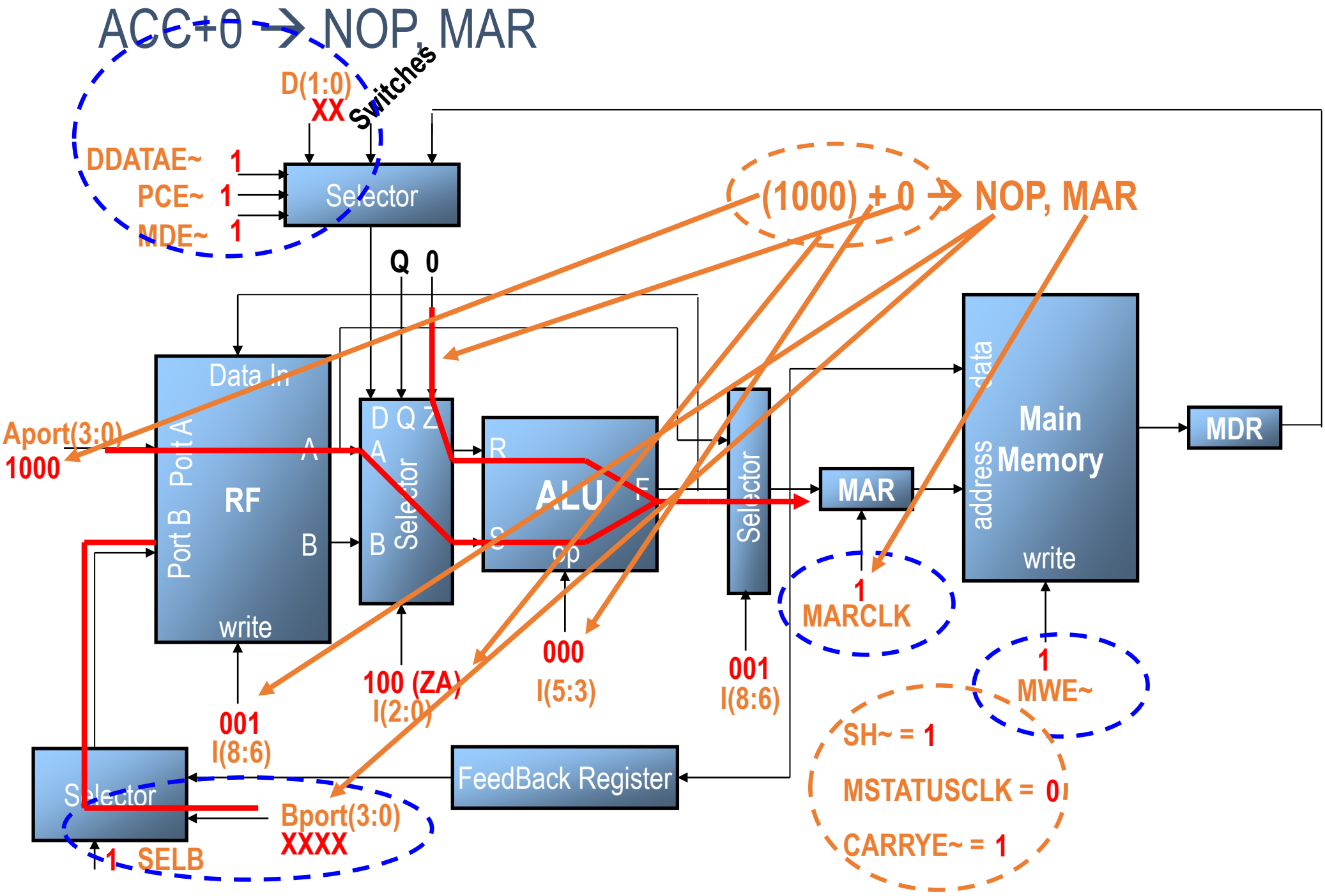
	BRA	BIN	CON	I	I	I	APORT	BPORT	DDATA	SH~	SELB	MWE~	MARCLK	MSTATUS	LDS~	PCE~	CARRYE~	MDE~	DDATAE~
	(4:0)	(2:0)	(2:0)	(2:0)	(5:3)	(8:6)	(3:0)	(3:0)	(1:0)										
MDR+0->ACC	xxxxx	000	xxx	111	000	011	xxxx	1000	xx	1	1	1	0	0	1	1	1	0	1

Βήμα 2 : Συγγραφή Μικροεντολών

PC = 0100 και **ACC = 1000**

LDA $K :

1. PC + 1 → PC , MAR // Το K θα πάει στον MDR
2. MDR + 0 → ACC // Το K στον accumulator
3. ACC + 0 → MAR // Το K στον MAR, άρα το περιεχόμενο της K στον MDR
4. MDR + 0 → ACC // Ο MDR (δηλ. το περιεχ. της K) στον ACC
5. PC + 1 → PC, MAR // Ο MAR θα δείξει στο opcode της επόμ. μακροεντολής
6. NEXT(PC) // LOAD_DIR_SEQ~ για να πάμε στο μικροπρόγραμμα της
   // επόμενης μακροεντολής του μακροπρογράμματος


ACC+0 → NOP, MAR
D(1:0)
XX
Switches
DDATAE~ 1
PCE~ 1
MDE~ 1
Selector
(1000) + 0 → NOP, MAR
Q 0
Data In
Aport(3:0)
1000
Port A
Port B
RF
A
B
write
D Q Z
A
B
Selector
R
ALU
S
cp
F
Selector
MAR
1
MARCLK
data
address
Main Memory
write
MDR
000
I(5:3)
100 (ZA)
I(2:0)
001
I(8:6)
001
I(8:6)
1
MWE~
SH~ = 1
MSTATUSCLK = 0
CARRYE~ = 1
Selector
Bport(3:0)
XXXX
1 SELB
FeedBack Register

ACC+0 → NOP, MAR

	BRA	BIN	CON	I	I	I	APORT	BPORT	DDATA	SH~	SELB	MWE~	MARCLK	MSTATUS	LDS~	PCE~	CARRYE~	MDE~	DDATAE~
	(4:0)	(2:0)	(2:0)	(2:0)	(5:3)	(8:6)	(3:0)	(3:0)	(1:0)										
ACC+0->NOP,MAR	xxxxx	000	xxx	100	000	001	1000	xxxx	xx	1	1	1	1	0	1	1	1	1	1

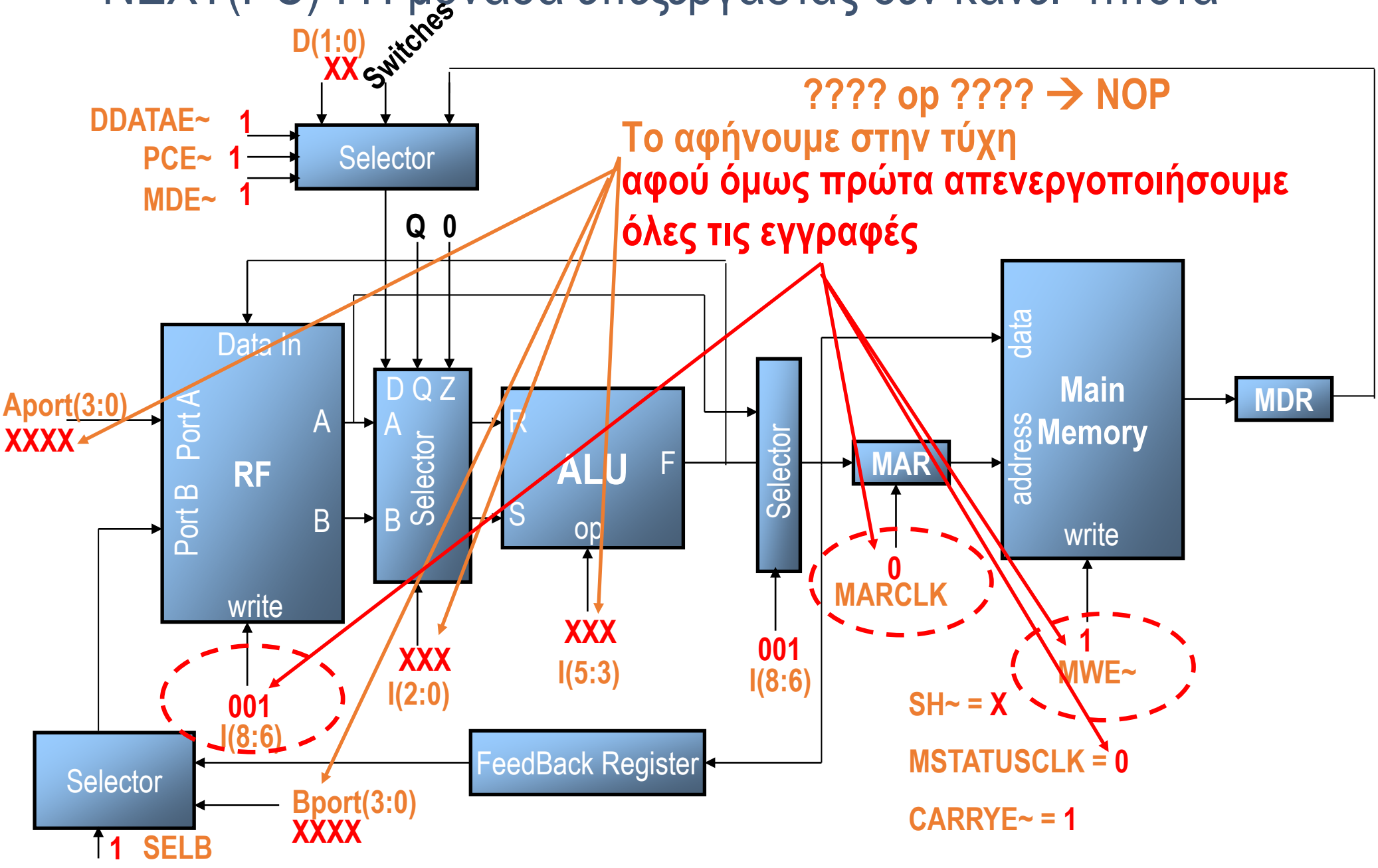
Βήμα 2 : Συγγραφή Μικροεντολών

PC = 0100 και **ACC = 1000**

LDA $K :

1. PC + 1 → PC , MAR // Το K θα πάει στον MDR
2. MDR + 0 → ACC // Το K στον accumulator
3. ACC + 0 → MAR // Το K στον MAR, άρα το περιεχόμενο της K στον MDR
4. MDR + 0 → ACC // Ο MDR (δηλ. το περιεχ. της K) στον ACC
5. PC + 1 → PC, MAR // Ο MAR θα δείξει στο opcode της επόμ. μακροεντολής
6. NEXT(PC) // LOAD_DIR_SEQ~ για να πάμε στο μικροπρόγραμμα της επόμενης μακροεντολής του μακροπρογράμματος

NEXT(PC) : Η μονάδα επεξεργασίας δεν κάνει 'τίποτα'



NEXT(PC) : Η μονάδα επεξεργασίας δεν κάνει ‘τίποτα’

	BRA	BIN	CON	I	I	I	APORT	BPORT	DDATA	SH~	SELB	MWE~	MARCLK	MSTATUS	LDS~	PCE~	CARRYE~	MDE~	DDATAE~
	(4:0)	(2:0)	(2:0)	(2:0)	(5:3)	(8:6)	(3:0)	(3:0)	(1:0)										
NEXT(PC)	xxxxx	000	xxx	xxx	xxx	001	xxxx	xxxx	xx	x	1	1	0	0	0	1	1	1	1

Βήμα 2 : ADD $K , STA $K

ADD $K :

1. PC + 1 → PC , MAR
2. MDR + 0 → **X**
3. **X** + 0 → NOP, MAR
4. MDR + ACC → ACC
5. PC + 1 → PC , MAR
6. NEXT(PC)

STA $K :

1. PC + 1 → PC , MAR
2. MDR + 0 → **X**
3. **X** + 0 → NOP,MAR
4. ACC + 0 → NOP, MWE~
5. PC + 1 → PC , MAR
6. NEXT(PC)

Να τις έχετε έτοιμες για το πρώτο εργαστήριο.
Παρατηρήστε ότι οι περισσότερες εντολές είναι ίδιες με αυτές που περιγράψαμε

Το Bootstrap μικροπρόγραμμα

Με τον τρόπο που γράφουμε τα μικροπρογράμματα εξασφαλίζουμε ότι μετά το τέλος ενός μικροπρογράμματος, ο έλεγχος περνάει στο μικροπρόγραμμα της επόμενης μακροεντολής (PC+1→PC, MAR και NEXT(PC))

Πώς όμως θα ξεκινήσει το μικροπρόγραμμα της πρώτης μακροεντολής, αφού δεν υπάρχει κάποιο άλλο μικροπρόγραμμα που να του δώσει τον έλεγχο ;;;;

Bootstrap μικροπρόγραμμα : **Αρχικοποιεί τον PC** με τη διεύθυνση της κύριας μνήμης όπου έχουμε αποθηκευμένη την πρώτη μακροεντολή του μακροπρογράμματος. Στη συνέχεια εκτελεί τη **NEXT(PC)**

Το Bootstrap μικροπρόγραμμα

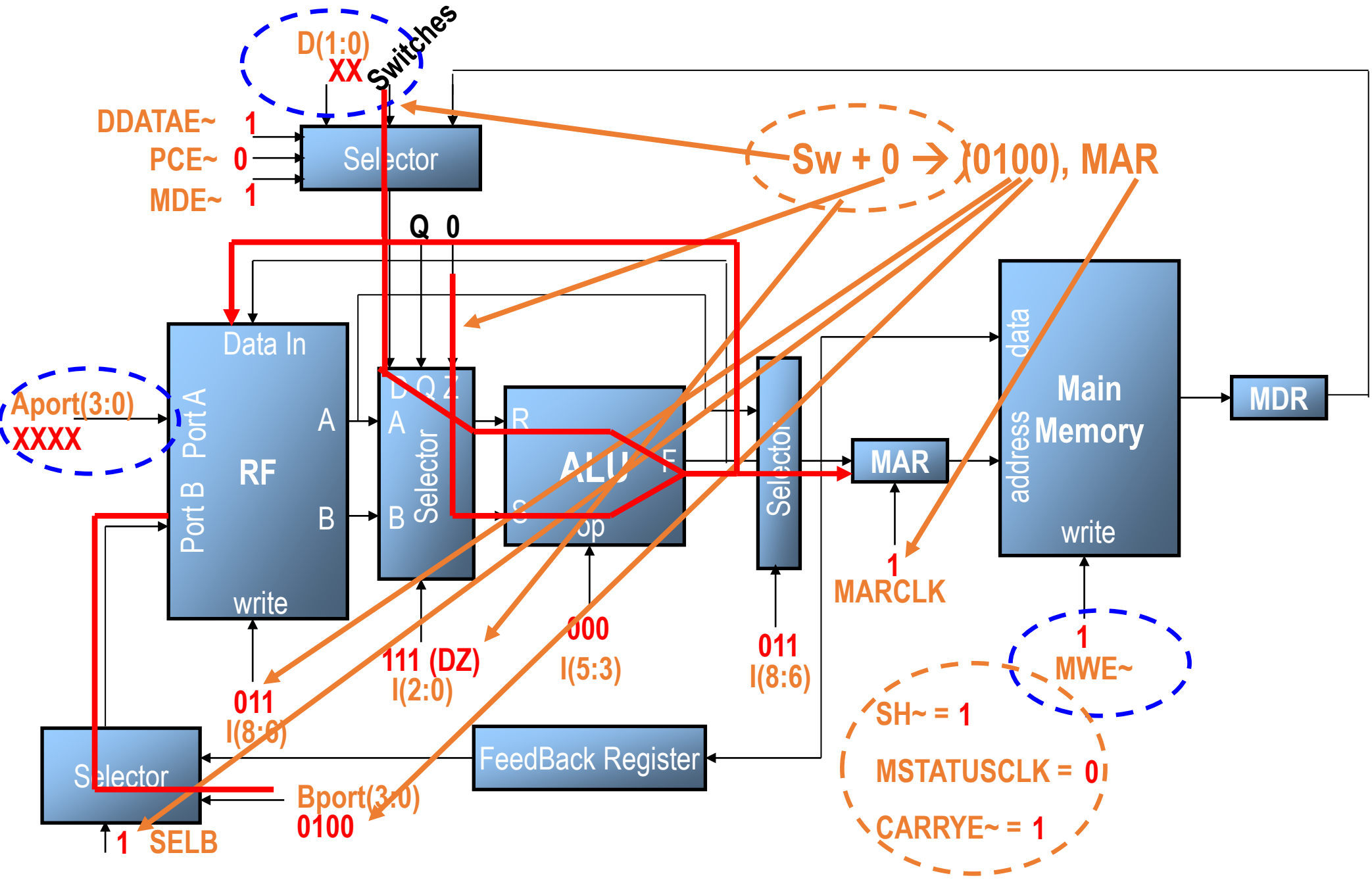
Bootstrap :

switches + 0 → PC , MAR

NEXT(PC)

Η αρχικοποίηση του PC, γίνεται ‘χειροκίνητα’ μέσω των DIP SWITCHES της πλακέτας !

Το bootstrap μικροπρόγραμμα : Switches + 0 →PC, MAR



Αναφορά

Στην αναφορά του εργαστηρίου, θα πρέπει:

1. Να παρουσιάζετε τα μικροπρογράμματα σε ‘συμβολική’ γλώσσα

2. Να διευκρινίζετε ποιους καταχωρητές χρησιμοποιείτε (π.χ. ο PC είναι ο 0100, ο ACC είναι ο 1000, ο X είναι …, αριθμός των regs θα δίνεται στην αρχή κάθε εργαστηρίου)

3. Να διευκρινίζετε ποια opcodes έχετε αναθέσει για κάθε μακροεντολή

4. Να παρουσιάζετε τα περιεχόμενα του mapper

5. Να παρουσιάζετε τα περιεχόμενα της κύριας μνήμης (π.χ. δοκιμαστικό μακροπρόγραμμα)

6. Να παρουσιάζετε τα μικροπρογράμματα (τις 40-άδες) και σε ποια θέση της μικρομνήμης αποθηκεύεται η κάθε μικροεντολή (κατά προτίμηση σε πίνακα του excell ή του word). Να σημειώνετε τα don’t cares των 40-αδων

Σε κάθε εργαστήριο …

- Θα πρέπει να παραδίδετε **την αναφορά της προηγούμενης άσκησης**
- Θα σας επιστρέφετε **η αναφορά της παραπροηγούμενης άσκησης**
- Θα πρέπει να έχετε **προετοιμάσει την τρέχουσα άσκηση. Έλλειψη προετοιμασίας -> απουσία**
- Θα γίνεται **προφορική εξέταση όλων των ομάδων.**

Τέλος