ΛΟΓΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Δίνεται η F_1= A'B'C' + A'B'C + A'BC' + AB'C' + ABC'

A	B	C	Ελαχιστόρος	Τιμή F1
0	0	0	0	1
0	0	1	1	1
0	1	0	2	1
0	1	1	3	0
1	0	0	4	1
1	0	1	5	0
1	1	0	6	1
1	1	1	7	0

Είσοδοι N σε πλήθος δημιουργούν 2^N συνδυασμούς

Θέλουμε να βρούμε για την F1 ποια είναι η τιμή της όταν A=B=C=0

0'0'0' + 0'0'0 + 0'00' + 00'0' + 000' = 1

Κάθε ελαχιστόρος (συνδυασμός εισόδων) δίνει μία τιμή στην έξοδο.

ΛΟΓΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Δίνεται η F_2= C' + A'B' = F_1

A	B	C	Ελαχιστόρος	Τιμή F2
0	0	0	0	1
0	0	1	1	1
0	1	0	2	1
0	1	1	3	0
1	0	0	4	1
1	0	1	5	0
1	1	0	6	1
1	1	1	7	0

Η F2 για κάθε ελαχιστόρο δίνει ίδια έξοδο με την F1. Ενδεικτικά:

A=B=C=0

F_2=0’ + 0’0’ = 1

ΙΣΟΔΥΝΑΜΕΣ: Δύο λογικές συναρτήσεις που σε κάθε συνδυασμό εισόδων δίνουν ίδια έξοδο.

Η F_1= A’B’C’ + A’B’C + A’BC’ + AB’C’ + ABC’

Αν έπρεπε να υλοποιήσω ένα κύκλωμα που παράγει αυτή τη λογική συνάρτηση, τότε θα έπρεπε να χρησιμοποιήσω:

5 AND τριών εισόδων
1 OR 5 εισόδων

Για την F_2 χρειαζόμαστε 1 αντιστροφέα (C’), μία AND 2 εισόδων και μία OR δύο εισόδων.

ΠΟΛΎ λιγότερο hardware

ΠΩΣ από την F_1 μπορώ να φτάσω στην F_2;

ΛΟΓΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Δίνεται η F_1= A’B’C’ + A’B’C + A’BC’ + AB’C’ + ABC’ στην οποία ξαναγράφω τον όρο A’B’C’:

F_1= A’B’C’ + A’B’C + A’BC’ + **A’B’C’** + AB’C’ + ABC’ = A’B’(C+C’) + A’C’ (B+B’) + AC’ (B+B’) =

= A’B’ +C’(A’+A) = C’ +A’B’

A’B’C’ + A’B’C’ = A’B’C’ επομένως έχω το δικαίωμα να προσθέσω στη συνάρτησή μου το A’B’C’ άλλη μία φορά (με την προϋπόθεση ότι ο όρος γινομένου υπάρχει ήδη στη συνάρτηση).

ΠΟΥ ΤΟ ΞΕΡΩ;;;; -> ΑΠΑΝΤΗΣΗ (όταν δούμε τη μέθοδο του Karnaugh)

A’B’C’ + A’B’C = A’B’ (C’+C) = A’B’ 1 = A’B’
A’BC’ + **A’B’C’** = A’C’ (B + B’) = A’C’
AB’C’ + ABC’ = AC’ (B’+B) = AC’

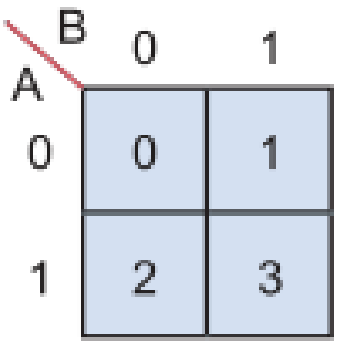
Τελικά έχουμε F_1 = A’B’ + A’C’ + AC’ = A’B’ + C’(A’+A) = A’B’ + C’ =F_2

ΠΑΡΑΤΗΡΗΣΕΙΣ
1. Πως ξέρουμε ποιον όρο πρέπει να προσθέσουμε;
ΑΠ: Από τη μορφή που θα έχει ο χάρτης μπορούμε σίγουρα να το δούμε.

2. Τι θα συμβεί αν έχω 5-6 μεταβλητές και όχι 3;
ΑΠ: Μεγάλο πρόβλημα, δεν είναι εύκολο να χρησιμοποιηθεί άλγεβρα Boole.
Αυτά τα ζητήματα μας οδηγούν σε μία οπτικοποιημένη μέθοδο απλοποίησης που λέγεται

χάρτης Karnaugh

ΧΑΡΤΗΣ ΔΥΟ ΜΕΤΑΒΛΗΤΩΝ



Τον χρησιμοποιούμε όταν έχουμε συνάρτηση με 2 μεταβλητές (στο παράδειγμα Α, Β).

Στις στήλες το Β στις γραμμές το Α.

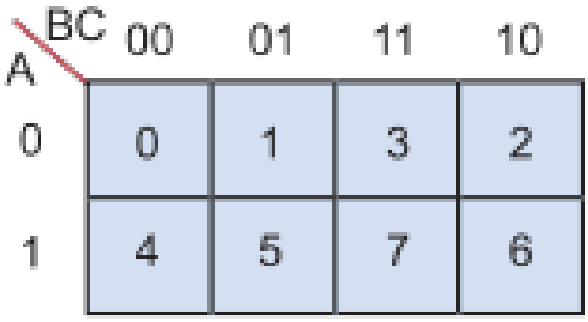
Συνολικά, ο χάρτης δύο μεταβλητών έχει 4 τετράγωνα (γιατί 2 μεταβλητές μπορούν να δημιουργήσουν 2^2=4 ελαχιστόρους). Οι ελαχιστόροι είναι γραμμένοι μέσα στον χάρτη (Για διευκόλυνση, ΔΕΝ τους γράφουμε όταν έχουμε να αναλύσουμε συναρτήσεις)

Πχ ο ελαχιστόρος 0 βρίσκεται από τον συνδυασμό ΑΒ =00 (άνω αριστερά κουτάκι)
ο ελαχιστόρος 1 βρίσκεται από τον συνδυασμό ΑΒ =01 (άνω δεξιά κουτάκι)
ο ελαχιστόρος 2 βρίσκεται από τον συνδυασμό ΑΒ =10 (κάτω αριστερά κουτάκι)
ο ελαχιστόρος 3 βρίσκεται από τον συνδυασμό ΑΒ =11 (κάτω δεξιά κουτάκι)

ΓΕΙΤΟΝΙΚΑ ΤΕΤΡΑΓΩΝΑ: Δύο τετράγωνα λέγονται γειτονικά όταν οι σειρές bit στις οποίες αντιστοιχούν διαφέρουν κατά 1 μόνο bit

Το τετράγωνο 0 (ΑΒ=00) είναι γειτονικό με το ΑΒ=01 (τετράγωνο 1), οι συμβολοσειρές bit 00 και 01 διαφέρουν κατά 1 bit
Το τετράγωνο 0 (ΑΒ=00) είναι γειτονικό με το ΑΒ=10 (τετράγωνο 2), οι συμβολοσειρές bit 00 και 10 διαφέρουν κατά 1 bit
Το τετράγωνο 0 (ΑΒ=00) ΔΕΝ είναι γειτονικό με το ΑΒ=11 (τετράγωνο 3) γιατί οι συμβολοσειρές bit 00 και 11 διαφέρουν κατά 2 bit
Το τετράγωνο 2 (ΑΒ=10) ΔΕΝ είναι γειτονικό με το ΑΒ=01 (τετράγωνο 1) γιατί οι συμβολοσειρές bit 10 και 01 διαφέρουν κατά 2 bit

ΧΑΡΤΗΣ ΤΡΙΩΝ ΜΕΤΑΒΛΗΤΩΝ



A \ BC	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Τρεις μεταβλητές A,B,C στις γραμμές τοποθετείται η A, στις στήλες οι συνδυασμοί των BC

ΣΥΝΔΥΑΣΜΟΙ B,C: κανονικά, θα γράφαμε:

00
01 (μεταξύ των 01 και 10 η διαφορά είναι 2 bit)
10
11

Όμως, στον χάρτη βλέπουμε
00
01
11
10
Αυτό συμβαίνει επειδή έχουμε πει ότι τα γειτονικά τετράγωνα διαφέρουν κατά 1 bit

00 με 01 διαφορά 1 bit
01 με 11 διαφορά 1 bit
11 με 10 διαφορά 1 bit ΚΑΙ για να κλείσει ο κύκλος
10 με 00 διαφορά 1 bit

Το τετράγωνο 0 (000) είναι γειτονικό με το 1(001) με το 4 (100) ΚΑΙ με το 2 (010) γιατί και με τα 3 διαφέρουν κατά 1 bit.
Για την περίπτωση των τετραγώνων 0 και 2 σκεφτείτε τον χάρτη ως κύλινδρο

Έστω τα 4 τετράγωνα 000 -> 010 -> 110 -> 100 -> **000**

Αυτά τα 4 δημιουργούν μία τετράδα γειτονικών κελιών. Γιατί;

000 με το 010 διαφέρουν 1 bit
010 με το 110 διαφέρουν 1 bit
110 με το 100 διαφέρουν 1 bit
100 με το 000 διαφέρουν 1 bit

ΑΝΤΙΠΑΡΑΔΕΙΓΜΑ
Τα τετράγωνα 3,2,5,4 είναι γειτονική τετράδα;
011 με το 010 διαφέρουν κατά 1 bit (ΑΠΌ 3 -> 2)
010 με το 101 διαφέρουν κατά 3 bit (ΑΠΌ 2 -> 5)

ΓΕΙΤΟΝΙΚΕΣ τετράδες θα σχηματιστούν ΜΟΝΟ από διαδοχικά τετράγωνα γειτονικά μεταξύ τους και ο έλεγχος θα πρέπει να συμπεριλαμβάνει το κλείσιμο του κύκλου (να γυρίσουμε στο σημείο εκκίνησης)
0473; ΕΊΝΑΙ γειτονικά; ΟΧΙ!!!!

Οι ομάδες γειτονικών τετραγώνων που θα αναζητούμε περιέχουν πλήθος τετραγώνων ΔΥΝΑΜΗ του 2
Δεν θα ψάξουμε τριάδες ή πεντάδες ή επτάδες γειτονικών τετραγώνων

0 = **0**00
4 = **1**00

011 = 3

ΧΑΡΤΗΣ ΤΕΣΣΑΡΩΝ ΜΕΤΑΒΛΗΤΩΝ

AB \ BC	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

A, B, C, D

Στις γραμμές A,B
Στις στήλες C,D

4 μεταβλητές σημαίνει 16 ελαχιστόροι, 16 τετράγωνα

Ο ελαχιστόρος 12 = 1100 είναι στο αριστερό τετράγωνο της τρίτης γραμμής και όχι της τέταρτης. Αυτό συμβαίνει γιατί και στις γραμμές οι συνδυασμοί ΑΒ γράφονται:
00-> 01 -> 11 -> 10 και όχι 00 -> 01 -> 10 -> 11

ΓΕΙΤΟΝΙΚΕΣ τετράδες: Τα τετράγωνα 0,1,4,5 είναι γειτονικά; ΝΑΙ
4, 12, 7, 15 ΟΧΙ
0,2,10, 8 ΝΑΙ ΣΕ ΔΥΑΔΙΚΗ ΜΟΡΦΗ 0000 -> 0010 -> 1010 -> 1000 -> 0000

2, 10, 8, 12 ΟΧΙ 0010 -> 1010 -> 1000 -> 1100 -> **00**10

5,6,7,14 ΟΧΙ

0,4,12,8,10,14,6,2: ΟΚΤΑΔΑ ΝΑΙ
0,4,12,8,1,5,13,9 ΟΚΤΑΔΑ ΝΑΙ

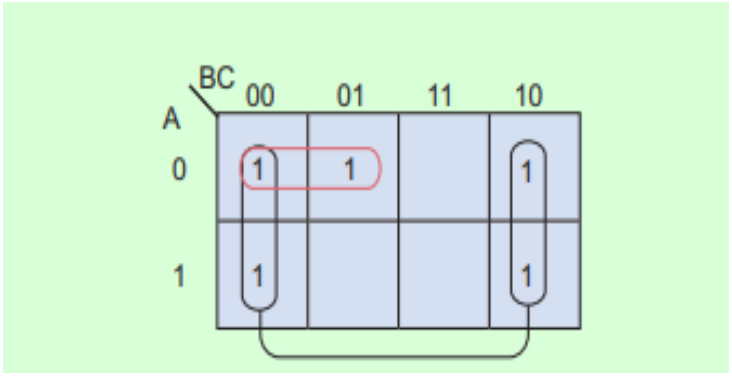
0,4,12,8, 11, 15, 7,3 ΟΧΙ (από το 8 στο 11 η διαφορά δεν είναι 1 bit)
ΔΕΝ παίζει ρόλο η σειρά που γράφω τα τετράγωνα, αλλά παίζει ρόλο η σειρά που θα ακολουθήσω για να πω ότι είναι ή δεν είναι γειτονικά. ΘΕΛΩ να βρίσκω συνεχώς γειτονικά τετράγωνα στην ακολουθία

Έστω τα τετράγωνα 0000 και 0001. Σε ποια γινόμενα αντιστοιχούν;

A'B'C'D' (ελαχιστόρος 0)
A'B'C'D (ελαχιστόρος 1)

Πρόσθεση: A'B'C'D' +A'B'C'D = A'B'C'(D+D') = A'B'C'
Διώξαμε το D γιατί πήραμε ομάδα με 2 τετράγωνα

ΕΠΙΣΤΡΟΦΗ ΣΤΟ ΠΑΡΑΔΕΙΓΜΑ – ΒΗΜΑΤΑ ΤΗΣ ΜΕΘΟΔΟΥ ΤΟΥ ΧΑΡΤΗ KARNAUGH



F_1= A'B'C' + A'B'C + A'BC' + AB'C' + ABC'

Ποιοι ελαχιστόροι βρίσκονται γραμμένοι στην έκφραση της συνάρτησης;
Όταν έχουμε τιμή 0, η μεταβλητή γράφεται συμπληρωματικά, διαφορετικά σε κανονική μορφή.

A'B'C' σημαίνει ότι A=B=C=0 άρα είναι ο ελαχιστόρος 0 (000)
A'B'C A=B=0 και C=1, άρα είναι ο ελαχιστόρος 1 (001)
A'BC' , είναι A=C=0, B=1 είναι ο ελαχιστόρος 2 (010)
AB'C', είναι A=1, B=C=0 άρα είναι ο ελαχιστόρος 4 (100)
ABC', είναι A=B=1, C=0 άρα είναι ο ελαχιστόρος 6 (110)

$F_1 = \Sigma(0,1,2,4,6)$

ΠΡΩΤΟ ΒΗΜΑ: Βάσει του πλήθους μεταβλητών επιλέγουμε τον κατάλληλο χάρτη
ΔΕΥΤΕΡΟ ΒΗΜΑ: Εκφράζουμε την F ως άθροισμα ελαχιστόρων, Σ( ), και τοποθετούμε τις μονάδες στις θέσεις του χάρτη που αντιστοιχούν σε αυτούς τους ελαχιστόρους

ΤΡΙΤΟ ΒΗΜΑ: Δημιουργούμε ομάδες από άσσους οι οποίες:
1) το πλήθος των άσσων σε κάθε ομάδα είναι δύναμη του 2 (1, 2, 4, 8…..)
2) κάθε ομάδα, πρέπει να περιέχει ΜΕΓΙΣΤΟ πλήθος άσσων δηλαδή αν έχω τη δυνατότητα να πάρω μια τετράδα, δεν θα πάρω δύο δυάδες.
3) Πρέπει όλοι οι άσσοι να βρεθούν σε τουλάχιστον μία ομάδα
4) Μπορούμε έναν άσσο να τον συμπεριλάβουμε σε περισσότερες από 1 ομάδες

ΣΥΛΛΟΓΙΣΜΟΣ: Έχω 8 τετράγωνα, μπορώ να βρω οκτάδα από άσσους (8 είναι το μέγιστο).
Αν είχα 8 άσσους η συνάρτηση θα ήταν ίση με 1
Πάω στην αμέσως επόμενη μικρότερη δύναμη του 2 που είναι το 4. Μπορώ να βρω τετράδα από άσσους;
ΝΑΙ μπορώ από τους άσσους στα τετράγωνα 0,4, 2, 6.
Κάλυψα όλους τους άσσους; ΟΧΙ μου μένει ο άσσος στο τετράγωνο 1
ΠΩΣ θα καλύψω τον άσσο στο τετράγωνο 1;
Μπορώ να πάρω τετράδα;
Μπορώ σε δυάδα; Το τετράγωνο μηδέν μπορεί κάνει δυάδα με το τετράγωνο 1.
Παρατηρούμε ότι το τετράγωνο 0 χρησιμοποιήθηκε 2 φορές, όπως ακριβώς ο όρος A'B' C' προστέθηκε 2 φορές στην απλοποίηση με άλγεβρα Boole.

ΤΕΤΑΡΤΟ ΒΗΜΑ
Κάθε ομάδα, δημιουργεί έναν ΑΠΛΟΠΟΙΗΜΕΝΟ όρο γινομένου. Αν έχουμε Κ ομάδες (στο παράδειγμα Κ = 2) θα έχουμε Κ όρους γινομένου, οι οποίοι θα αθροιστούν

Κ1: Ο όρος που προκύπτει από την τετράδα:

Κ1: Εξετάζουμε μία μία τις μεταβλητές εντός της ομάδας. Α: 000, 010, 100, 110 δηλαδή το Α αλλάζει μέσα στην τετράδα επομένως φεύγει.
Β: Αλλάζει 0, 1, 0, 1 άρα φεύγει
C: Το τρίτο bit είναι σταθερά 0. Άρα το C παραμένει σταθερό και επειδή είναι ίσο με 0 θα

γραφτεί C’ (αν παρέμενε σταθερά 1 θα γραφόταν C). Άρα ο πρώτος όρος μου είναι C’ και έχουν φύγει τα A,B

Άλγεβρα Boole: A’B’C’ + A’BC’ + AB’C’ + ABC’ =
A’C’(B+B’) + AC’(B+B’) = A’C’+AC’ = C’

ΠΟΣΟΙ άσσοι συμμετείχαν στο γκρουπ; $2^{\mathbf{2}}$=4
Όταν σχηματίζουμε ομάδα με 2^2=4 άσσους ΠΑΝΤΑ θα απαλείφουμε 2 όρους (B, A).

ΠΑΡΑΤΗΡΗΣΗ: αν υπήρχε οκτάδα, θα είχαμε F2=1 (άρα θα είχαμε διώξει 3 όρους, 2^3=8)
ΜΕ ΔΥΑΔΑ πόσους όρους διώχνουμε; ΈΝΑΝ
Με γκρουπ ενός άσσου; ΜΗΔΕΝ (2^0=1)

K2: 00**0** και 00**1**.
A=0 σταθερό
B= 0 σταθερό
C αλλάζει από 0 σε 1 (ΕΝΤΟΝΑ)
Επειδή τα A,B είναι σταθερά 0, ο όρος K2 γράφεται A’B’
Άλγεβρα Boole: A’B’C’ +A’B’C = A’B’

ΠΟΣΟΙ όροι αφαιρέθηκαν από τον K2; ΕΝΑΣ γιατί έχουμε ομάδα $2^{\mathbf{1}}$=2 άσσων

ΠΕΜΠΤΟ ΒΗΜΑ
Αθροίζουμε τα γινόμενα που παράγονται στο ΒΗΜΑ 4
F1 = K1 +K2 = C’ +A’B’

ΟΠΤΙΚΑ:
Η τετράδα εκτείνεται και στις 2 γραμμές άρα το A φεύγει.
Τα B,C εκτείνονται στις στήλες 1 και 4. Άρα, το B στη στήλη 1 είναι 0 και στη στήλη 4 γίνεται 1 άρα φεύγει ενώ το C παραμένει 0 στις στήλες 1 και 4. Άρα γράφεται C’

ΒΗΜΑΤΑ ΤΗΣ ΜΕΘΟΔΟΥ ΤΟΥ ΧΑΡΤΗ KARNAUGH

1. Αποφασίζουμε ποιος χάρτη είναι ο κατάλληλος
2. Τοποθετούμε τις μονάδες
3. Δημιουργούμε ομάδες άσσων τέτοιες ώστε
    1. Κάθε ομάδα να έχει μέγιστο πλήθος άσσων, όπου το πλήθος αυτό είναι δύναμη του 2 (ΌΧΙ 3, 5, 6, κ.ο.κ)
    2. Ένας άσσος μπορεί να ανήκει σε 1 ή περισσότερες ομάδες
4. Από κάθε ομάδα εξάγουμε έναν όρο γινομένου
5. Αθροίζουμε τους όρους

ΧΑΡΤΗΣ ΠΕΝΤΕ ΜΕΤΑΒΛΗΤΩΝ

A=0

BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

A=1

BC \ DC	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

F(A,B,C,D,E)
5 μεταβλητές: 2 χάρτες όπου τοποθετούνται οι μεταβλητές B,C,D,E. Για τον ένα χάρτη, A=0. Για τον άλλο, A=1.

ΤΟΠΟΘΕΤΗΣΗ ΕΛΑΧΙΣΤΟΡΩΝ: A= 0 σταθερά. Άρα οι ελαχιστόροι που προκύπτουν είναι από 0 – 15. Αυτός ο χάρτης από αριστερά είναι ουσιαστικά ο ίδιος με έναν χάρτη τεσσάρων μεταβλητών. Π.χ. το κάτω αριστερά τετράγωνο αντιστοιχεί στον ελαχιστόρο 01000 = 8. Με 4 μεταβλητές, ήταν το 1000=8.
Δεξιά, το A=1 σταθερά. Αυτό έχει ως συνέπεια, τα τετράγωνα που βρίσκονται σε αντίστοιχες θέσεις στους 2 χάρτες διαφέρουν κατά 16.
Π.χ., το κάτω αριστερά τετράγωνο, είναι 01000=8 στον πρώτο χάρτη και 11000=24 στον δεύτερο.
ΤΕΤΡΑΓΩΝΑ ΠΟΥ ΒΡΙΣΚΟΝΤΑΙ ΣΕ ΑΝΤΙΣΤΟΙΧΕΣ ΘΕΣΕΙΣ ΣΤΟΥΣ 2 ΧΑΡΤΕΣ ΕΊΝΑΙ ΓΕΙΤΟΝΙΚΑ.

Π.χ. **0**1000 και **1**1000

Τελικά, για να εντοπίσουμε γειτονικά τετράγωνα σε χάρτη 5 μεταβλητών εργαζόμαστε όπως και στην περίπτωση με 4 μεταβλητές αλλά επιπλέον ισχύει ότι
ΤΕΤΡΑΓΩΝΑ ΠΟΥ ΒΡΙΣΚΟΝΤΑΙ ΣΕ ΑΝΤΙΣΤΟΙΧΕΣ ΘΕΣΕΙΣ ΣΤΟΥΣ 2 ΧΑΡΤΕΣ ΕΊΝΑΙ ΓΕΙΤΟΝΙΚΑ.

Π.χ., ΤΕΤΡΑΔΑ 0,1,16,17: 0->1->17->16->0

00000 -> 00001 -> 10001 -> 10000 -> 00000

Π.χ., ΟΚΤΑΔΑ 4,5,12,13,20,21,28,29

ΧΑΡΤΗΣ ΕΞΙ ΜΕΤΑΒΛΗΤΩΝ

AB=00

CD \ EF	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

AB=01

CD \ EF	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

AB=10

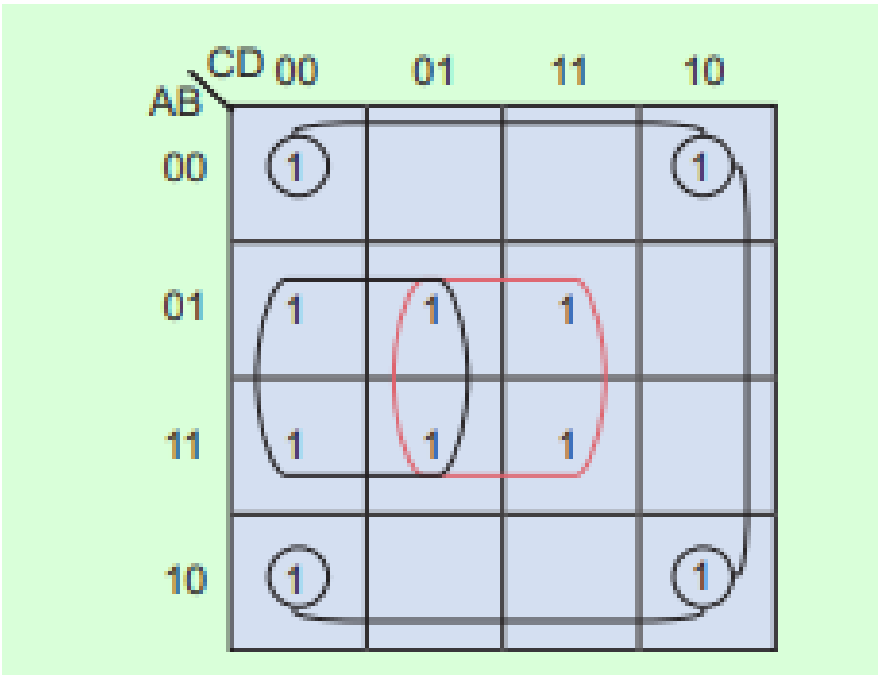
CD \ EF	00	01	11	10
00	32	33	34	35
01	36	37	39	38
11	44	45	47	46
10	40	41	43	42

AB=11

CD \ EF	00	01	11	10
00	48	49	51	50
01	52	53	55	54
11	60	61	63	62
10	56	57	59	58

Επέκταση της περίπτωσης με 5.
Έχουμε μία συνάρτηση F(A,B,C, D, E, F), όπου οι C, D, E, F τοποθετούνται με παρόμοιο τρόπο σε έναν χάρτη 4 μεταβλητών και τα AB μεταβάλλονται από 00→01→11→10

ΠΑΡΑΔΕΙΓΜΑ ΧΑΡΤΗ ΜΕ 4 ΜΕΤΑΒΛΗΤΕΣ



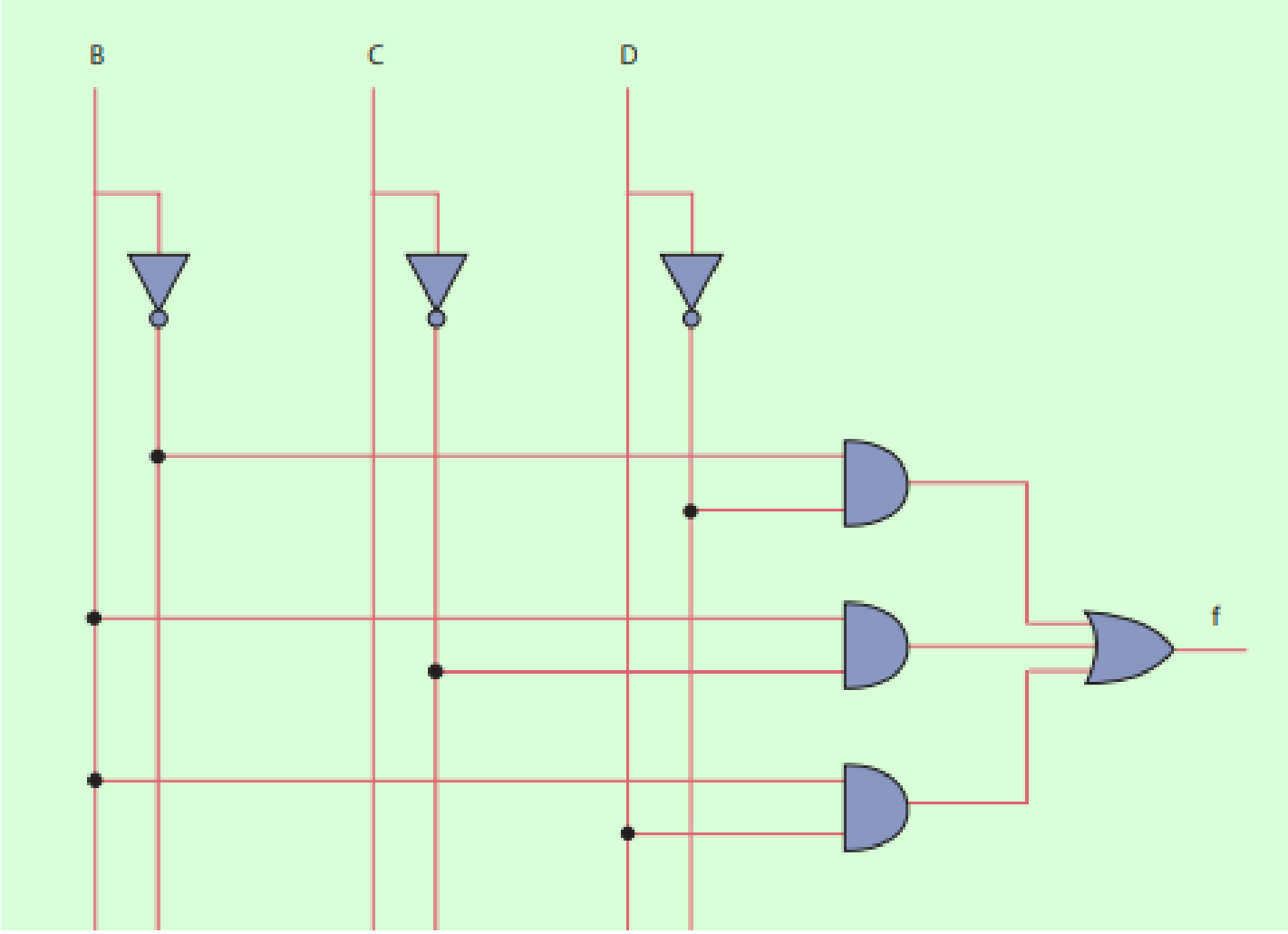
f(A, B, C, D) = Σ (0, 2, 4, 5, 7, 8, 10, 12, 13, 15).

ΛΟΓΙΚΗ: Έχω 16 τετράγωνα; Υπάρχει 16άδα; ΟΧΙ
Υπάρχει 8άδα; ΟΧΙ
ΤΕΤΡΑΔΕΣ ΕΧΟΥΜΕ: 1η τετράδα (γωνίες): Η τετράδα αυτή εκτείνεται στην πρώτη και στην τέταρτη γραμμή. Το Α αλλάζει και φεύγει, το Β=0. Τα C, D εκτείνονται στην πρώτη και τέταρτη στήλη άρα το C από 0 γίνεται 1 άρα φεύγει και το D=0 σταθερά. Το πρώτο γκρουπ δίνει B'D'.

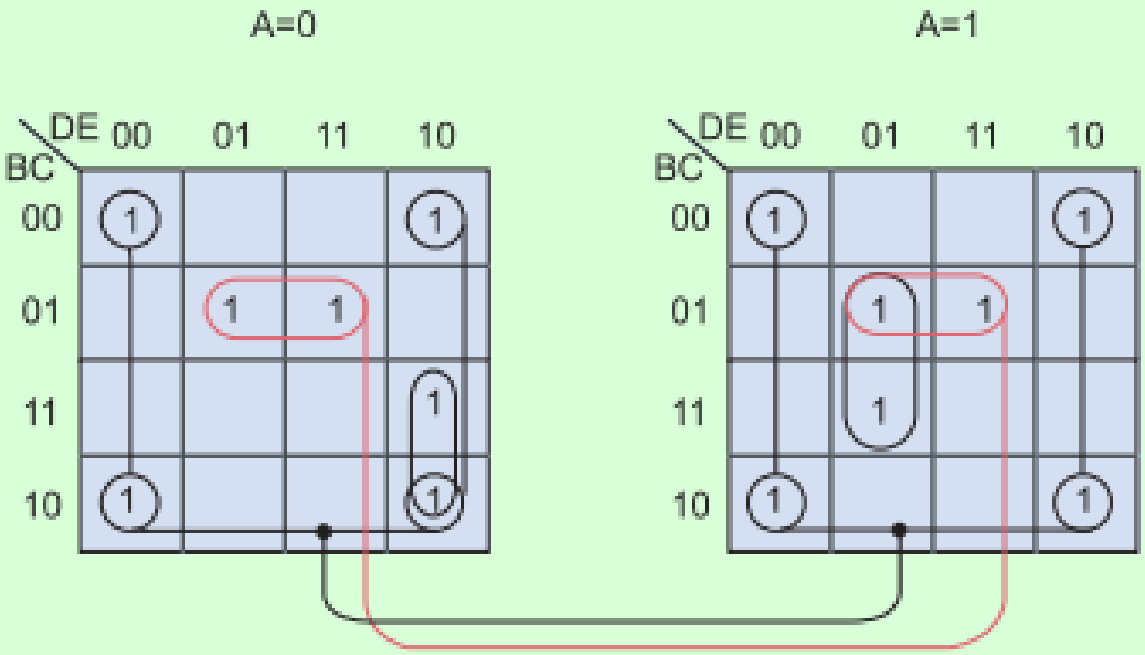
Τετράδα (στις γραμμές 2,3 στήλες 1,2): BC'
Τετράδα (κόκκινη): BD.
Συμπέρασμα: F= B'D' + BC' + BD

Αν πάρω 2 τετράδες και μία δυάδα έχω: B'D' + BC' + BCD (ένας όρος γινομένου θα ήταν λίγο μεγαλύτερος).

ΣΤΟΧΟΣ: Να καλύψουμε όλους τους άσσους σε όσο το δυνατόν μεγαλύτερα γκρουπ (ΔΥΝΑΜΗ ΤΟΥ 2).



ΠΑΡΑΔΕΙΓΜΑ 5 μεταβλητών



f(A, B, C, D, E) = Σ (0, 2, 5, 7, 8, 10, 14, 16, 18, 21, 23, 24, 26, 29).
Να απλοποιηθεί η λογική έκφραση F= Σ() με τη μέθοδο του χάρτη Karnaugh και να σχεδιαστεί το κύκλωμα.
Έχουμε 32 τετράγωνα: Υπάρχει 32άδα;
Υπάρχει 16άδα;
Υπάρχει 8άδα; ΝΑΙ στις 8 άκρες των 2 χαρτών. Α αλλάζει από 0 σε 1 άρα φεύγει. C'E'

(παρατηρήστε ότι με ομάδα 8= $2^{\mathbf{3}}$ άσσων εξαλείψαμε **3** μεταβλητές, A,B,D).

ΑΠΟΜΕΝΟΥΝ 6 άσσοι.
Υπάρχει άλλη 8άδα; ΟΧΙ
Υπάρχει τετράδα; Οι άσσοι που περικλείονται από κόκκινη γραμμή. B'CE (παρατηρήστε ότι με ομάδα 4= $2^{\mathbf{2}}$ άσσων εξαλείψαμε **2** μεταβλητές, A,D).
Έχουν απομείνει 2 άσσοι. Προφανώς καθένας θα πάει σε δυάδα.

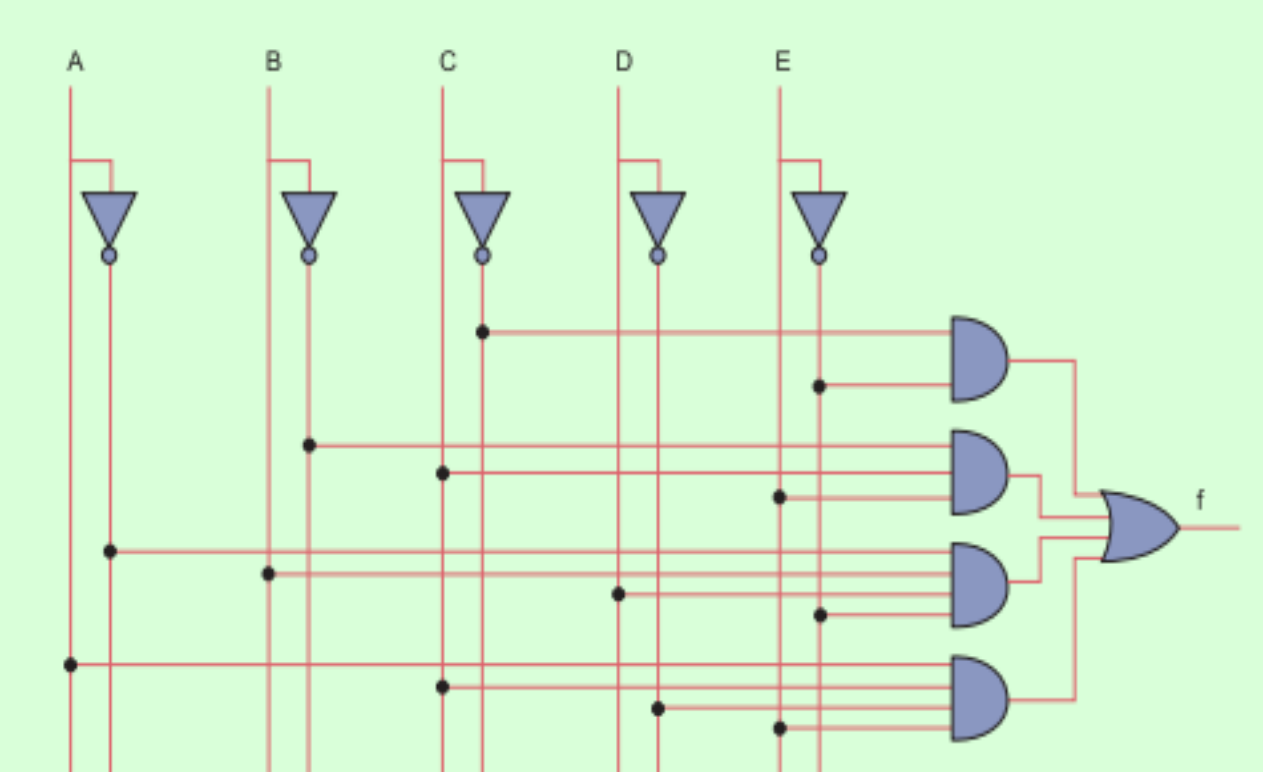
1η δυάδα (αριστερό χάρτη): Α σταθερά 0, A'. A'BDE' (παρατηρήστε ότι με ομάδα 2= $2^{\mathbf{1}}$ άσσων εξαλείψαμε **1** μεταβλητή, C).

2η δυάδα (δεξιό χάρτη): Α είναι σταθερά 1, A: ACD'E

F= C'E' + B'CE + A'BDE' + ACD'E

Έστω ότι στην F δεν υπήρχαν οι ελαχιστόροι 7, 21, 23
Ο ελαχιστόρος 5 έπρεπε να σχηματίσει ομάδα μόνος του.

A'B'CD'E (δηλαδή με ομάδα αποτελούμενη από 1=$2^{\mathbf{0}}$ άσσο εξαλείφουμε **0** όρους). Θα παρέμενε στο τελικό άθροισμα ένα γινόμενο 5 μεταβλητών



ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΕΣ

Ο αποκωδικοποιητής είναι το κύκλωμα που μετατρέπει την κωδικοποιημένη πληροφορία στην αρχική της μορφή. Γενικά, πρόκειται για ένα συνδυαστικό κύκλωμα που λαμβάνει στην είσοδο κωδικοποιημένη πληροφορία μεγέθους n bits, και την μετατρέπει στην αρχική της μορφή μεγέθους m bits, n < m. Συμβολικά, ο κωδικοποιητής αναφέρεται και ως n-σε-m.

Ο δυαδικός αποκωδικοποιητής είναι ένα κύκλωμα, το οποίο λαμβάνει n bits πληροφορίας στις γραμμές εισόδου του, τα οποία μετατρέπει σε αποκωδικοποιημένη πληροφορία μεγέθους 2^n bits, όπου όμως ένα μόνον bit είναι ίσο με τη μονάδα. Η αποκωδικοποιημένη πληροφορία εξάγεται ως αποτέλεσμα στις γραμμές εξόδου του κυκλώματος

ΔΥΑΔΙΚΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΣΗ: Αν για παράδειγμα έχουμε έναν αποκωδικοποιητή με N=3 εισόδους, τότε αυτό το κύκλωμα θα έχει 2^N= 8 εξόδους. Από τις 8 εξόδους, κάθε χρονική στιγμή, μόνο μία θα είναι ίση με 1. Οι άλλες θα είναι 0. Οι αποκωδικοποιητές ονομάζονται με βάση το πλήθος εισόδων και εξόδων. Π.χ., λέμε DEC 3 x 8, DEC 4 x 16
DEC 5 x 32.
Αυτό το κύκλωμα υλοποιείται «διαισθητικά».

ΠΑΡΑ ΠΟΛΛΕΣ ΕΦΑΡΜΟΓΕΣ: Και στη μονάδα ελέγχου της CPU, στην υλοποίηση της RAM, κ.ο.κ.

ΠΙΝΑΚΑΣ ΑΛΗΘΕΙΑΣ DEC 3x8

x	y	z	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

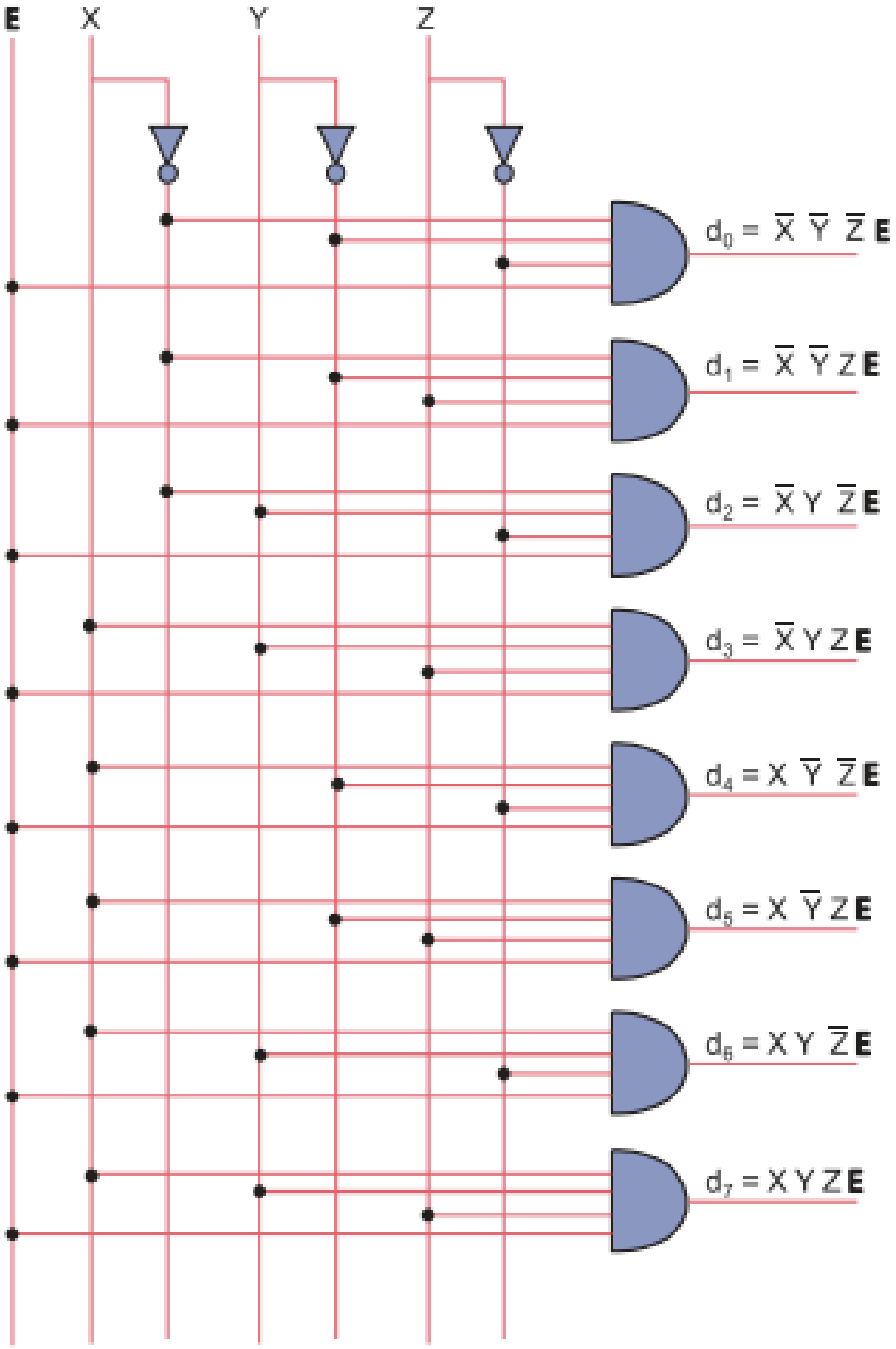
Πίνακα αληθείας αποκωδικοποιητή 3x8.
3 είσοδοι: x, y, z
8 έξοδοι: d7 – d0

ΔΙΑΙΣΘΗΤΙΚΑ: Δεν θα γράψουμε για κάθε συνάρτηση εξόδου μία απλοποιημένη λογική έκφραση. Επειδή κάθε έξοδος γίνεται 1 ΜΟΝΟ για έναν συνδυασμό εισόδων. ΔΕΝ ΜΠΟΡΩ ΝΑ ΚΑΝΩ γκρουπ από άσσους.

Όταν οι είσοδοι x,y,z σχηματίζουν τον αριθμό **0**, δηλαδή είναι 000, τότε η έξοδος **0** λαμβάνει τιμή 1 και όλες οι άλλες είναι 0 (πρώτη γραμμή του πίνακα αληθείας).
Όταν οι είσοδοι x,y,z σχηματίζουν τον αριθμό **1**, δηλαδή είναι 001, τότε η έξοδος **1** λαμβάνει τιμή 1 και όλες οι άλλες είναι 0 (δεύτερη γραμμή του πίνακα αληθείας).
Όταν οι είσοδοι x,y,z σχηματίζουν τον αριθμό **2**, δηλαδή είναι 010, τότε η έξοδος **2** λαμβάνει τιμή 1 και όλες οι άλλες είναι 0 (τρίτη γραμμή του πίνακα αληθείας).
Όταν οι είσοδοι x,y,z σχηματίζουν τον αριθμό **3**, δηλαδή είναι 011, τότε η έξοδος **3** λαμβάνει τιμή 1 και όλες οι άλλες είναι 0 (τέταρτη γραμμή του πίνακα αληθείας).
……

Όταν οι είσοδοι x,y,z σχηματίζουν τον αριθμό **7**, δηλαδή είναι 111, τότε η έξοδος **7** λαμβάνει τιμή 1 και όλες οι άλλες είναι 0 (όγδοη και τελευταία γραμμή του πίνακα αληθείας).

ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ DEC 3x8



Ε είναι ένα σήμα το οποίο όταν 1 επιτρέπει να περάσουν τα σήματα από τις πύλες ΚΑΙ. Το Ε προκύπτει από τη λέξη ENABLE που σημαίνει ΕΠΙΤΡΕΨΗ. ΑΝ Ε=0 όλες οι πύλες ΚΑΙ μηδενίζονται και δεν υπάρχει λειτουργία του κυκλώματος.

ΣΤΟΧΟΣ: Όταν οι είσοδοι x,y,z σχηματίζουν τον αριθμό **0**, δηλαδή είναι 000, τότε η έξοδος **0** λαμβάνει τιμή 1 και όλες οι άλλες είναι 0 (πρώτη γραμμή του πίνακα αληθείας).

Αν x=y=z=0 τότε x'y'z'=1.
Κάθε πύλη ΚΑΙ είναι διαφορετικά συνδεδεμένη με τις εισόδους ώστε να υλοποιεί έναν συνδυασμό.
ΛΟΓΙΚΗ: Οι είσοδοι σχηματίζουν έναν αριθμό. Αν κάποια bit αυτού του αριθμού είναι 0, τότε συνδέονται με αντιστροφείς (για να γίνουν 1). Αν τα bit είναι 1, συνδέονται με το κανονικό σήμα.
Έστω ότι οι είσοδοι είναι κάποια χρονική στιγμή xyz=101 (σχηματίζεται ο αριθμός 5).
X=1 (η σύνδεση της πύλης 5 είναι με την είσοδο X)
Y=0 (η σύνδεση της πύλης 5 είναι με την είσοδο Y΄)
Z=1 (η σύνδεση της πύλης 5 είναι με την είσοδο Z)

Κάθε χρονική στιγμή, ένας μόνο αριθμός μπορεί να σχηματίζεται στις εισόδους. Οι συνδέσεις είναι τέτοιες ώστε σε αυτό το συνδυασμό ΜΟΝΟ μία έξοδος να είναι 1.

ΤΡΟΠΟΣ ΕΠΑΛΗΘΕΥΣΗΣ: Για τους 2^3=8 των τιμών που μπορούν να έχουν οι είσοδοι γενικά (σε κάθε χρονική στιγμή ισχύει ΕΝΑΣ από αυτούς), εξετάστε τις τιμές των εξόδων. Θα διαπιστώσετε ότι ΜΙΑ από αυτές είναι 1 κάθε φορά.

CASCADING

Ένας αποκωδικοποιητής μεγάλου μεγέθους (π.χ. Word select της RAM) σχεδιάζεται από μικρότερα κυκλώματα αποκωδικοποιητών (cascading)

Επιθυμητός αποκωδικοποιητής $N \times 2^N$

Κατασκευή με διαθέσιμο αποκωδικοποιητή $K \times 2^K$

Η διαίρεση N/K θα δώσει το πλήθος των απαιτούμενων επιπέδων που θα χρησιμοποιηθούν

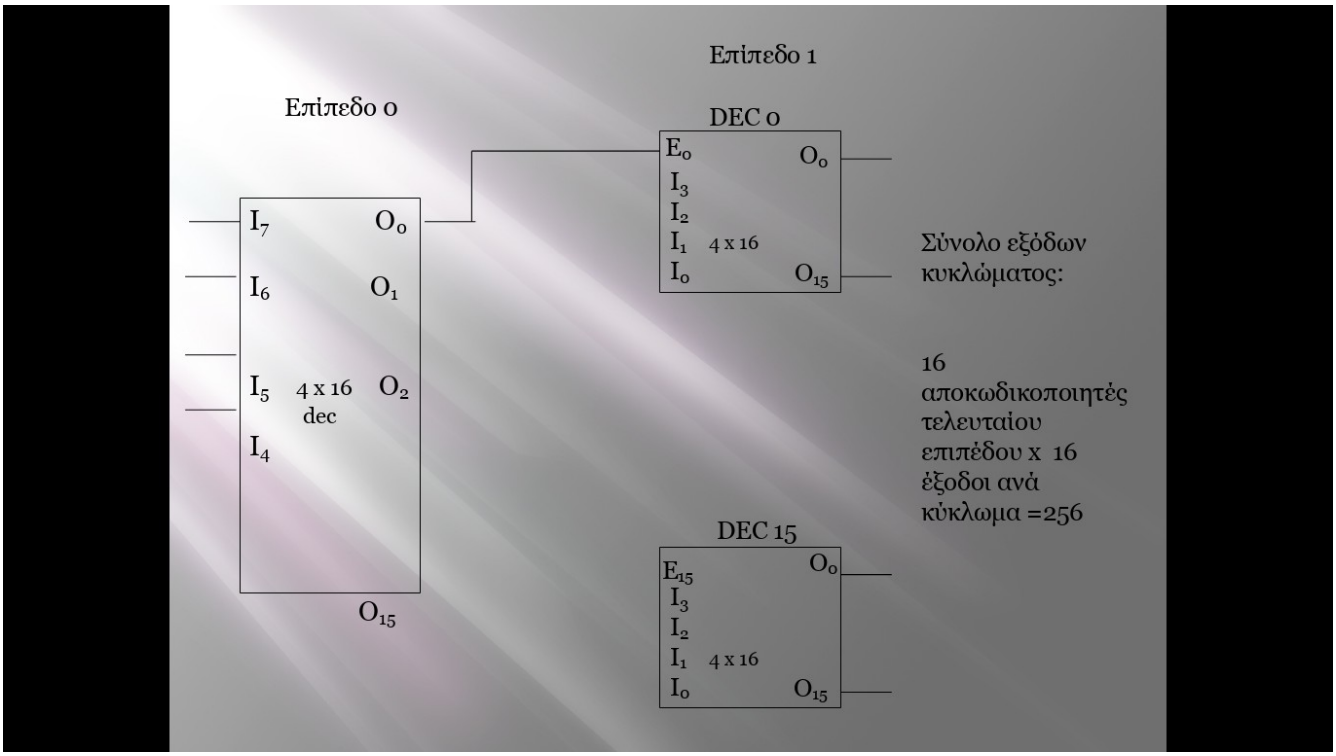
ΠΑΡΑΔΕΙΓΜΑ

Αποκωδικοποιητής 8 x 256 αποτελούμενος από αποκωδικοποιητές 4 x 16

Θα απαιτηθούν 8/4=2 επίπεδα αποκωδικοποιητών 4 x 16

Αποκωδικοποιητής 8 x 256 με 4 x 16. N=8, K=4 (N είσοδοι του επιθυμητού αποκωδικοποιητή, K οι είσοδοι των διαθέσιμων). ΕΔΏ το N διαιρείται ακριβώς από το K.

Επειδή ο αποκωδικοποιητής έχει 256 εξόδους ενώ κάθε αποκωδικοποιητής- εργαλείο περιέχει 16, αυτό σημαίνει ότι θα χρειαστούμε συνολικά 256/16= 16 αποκωδικοποιητές που δίνουν τις τελικές εξόδους.



ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ:

Στο επίπεδο 0 υπάρχει ένας αποκωδικοποιητής 4 x 16. Κάθε χρονική στιγμή, μία από τις εξόδους 0-15 είναι ίση με 1. Κάθε έξοδος τροφοδοτεί το σήμα επίτρεψης Ε ενός άλλου αποκωδικοποιητή 4 x 16.

Η έξοδος Ο0 του αποκ. Του επιπέδου 0 συνδέεται με την επίτρεψη Ε0 του αποκ. 0 του Επιπέδου 1.

Η έξοδος Ο1 του αποκ. Του επιπέδου 0 συνδέεται με την επίτρεψη Ε1 του αποκ. 1 του Επιπέδου 1.

Η έξοδος Ο2 του αποκ. Του επιπέδου 0 συνδέεται με την επίτρεψη Ε2 του αποκ. 2 του Επιπέδου 1.

…….

Η έξοδος Ο15 του αποκ. Του επιπέδου 0 συνδέεται με την επίτρεψη Ε15 του αποκ. 15 του Επιπέδου 1.

ΠΑΡΑΔΕΙΓΜΑ

‘Εστω ότι ο αποκωδικοποιητής 8 x 256 πρέπει να επιλέξει μία από τις 256 λέξεις μίας μνήμης (0-255). Έστω ότι κάποια στιγμή πρέπει να επιλεγεί η λέξη 15. Να δείξετε τις τιμές των εισόδων των αποκωδικοποιητών.

ΛΥΣΗ

Ο αποκωδικοποιητής συνολικά έχει 8 εισόδους Ι7, Ι6, Ι5, …… Ι0.

Οι 4 από αυτές (μεγαλύτερες δυνάμεις του 2), είναι οι είσοδοι του αποκ. Που υπάρχει στο επίπεδο 0. Οι άλλες (Ι3-Ι0) είναι κοινές σε όλους τους αποκωδικοποιητές του επιπέδου 1. Οι είσοδοι Ι7-Ι4 θα έχουν μία τιμή κάθε φορά, βάσει της οποίας μία από τις εξόδους Ο0-Ο15

του αποκ. Να είναι 1. Αυτές οι έξοδοι ΕΠΙΤΡΕΠΟΥΝ κάθε φορά τη λειτουργία ενός αποκ. Του επιπέδου 1.

Επειδή έχουμε 8 εισόδους, ο αριθμός 15 γράφεται με 8 bit: 00001111

I7 I6 I5 I4 I3 I2 I1 I0
0 0 0 0 1 1 1 1

Όταν ο αποκωδικοποιητής 8 x 256 δεχτεί ως είσοδο τον αριθμό 15, τότε τα bit I7, I6, I5, I4 θα είναι ίσα με 0. Άρα, το σήμα E0 του DEC0 Στο επίπεδο 1 θα είναι 1. Όλα τα E για τους υπόλοιπους 15 αποκωδικοποιητές του Επιπέδου 1 (DEC1-DEC15) θα είναι 0 δηλαδή οι DEC1 – DEC15 θα τεθούν εκτός λειτουργίας.
Όταν ο αριθμός 15 δοθεί ως είσοδος στον αποκ. 8 x 256, τα bit 1111 θα τροφοδοτήσουν τις εισόδους I3-I0 όλων των αποκ. Του επιπέδου 1.

Επομένως, με τιμές I3I2I1I0=1111, ο DEC0 θα ενεργοποιήσει την έξοδο O15 (αυτή θα είναι 1)

Άρα, από τις 256 εξόδους επιλέγεται η έξοδος 15 του αποκ. 0 στο επίπεδο 1.
0000 1111

0000: Επιλογή DEC στο επίπεδο 1 (ΔΗΛΑΔΗ ο 0)
1111: Επιλογή σήματος εξόδους του επιλεγμένου DEC του επιπέδου 1.

Ερώτηση: Αν ζητούταν η λέξη 31: Ποιος αποκ. Του επιπέδου 1 θα επιλεγόταν και ποια έξοδός του.

Στο δυαδικό 00011111

Άρα τα bit 0001 τροφοδοτούν τις εισόδους I7 I6 I5 I4 με αποτέλεσμα η επιλεγμένη έξοδος από του DEC του επιπέδου 0 να είναι η 1. Άρα, ο DEC 1 επιλέγεται.

Η E1 του DEC 1 είναι 1 και οι είσοδοι I3-I0 είναι 1111 άρα επιλέγουν την O15

ΕΡΩΤΗΣΗ 128 (ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗΣ ΚΑΙ ΕΞΟΔΟΣ).

1000 0000 =128

I7 I6 I5 I4
1 0 0 0 (επιλέγεται η O8 του επιπέδου 0), άρα το E8=1 και επιλέγεται ο DEC8.
ΠΟΙΑ ΕΞΟΔΟΣ; 0
ΑΠΌ τους 16 αποκ. Του επιπέδου 1 επιλέγεται και δουλεύει ΜΟΝΟ ο 8.

ΠΑΡΑΔΕΙΓΜΑ 2

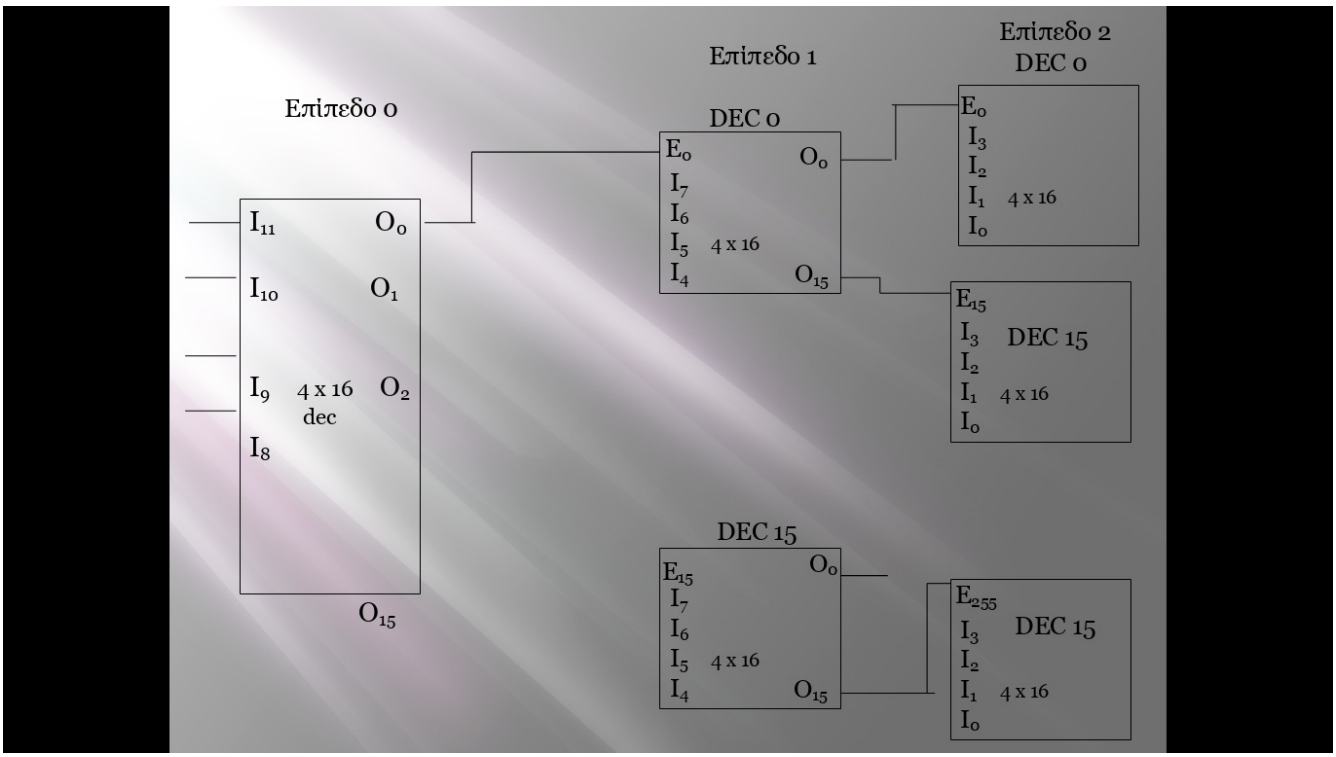
Αποκωδικοποιητής 12x 4096 με αποκωδικοποιητές 4 x 16.

Επειδή N=12, K=4, απαιτούνται 12/4 = 3 επίπεδα (από 0 ως 2)

Οι 4096 έξοδοι εμφανίζονται στο τελευταίο επίπεδο

4096 = 2^{12} έξοδοι
4 x 16.

Αν διαιρέσουμε τις εισόδους του επιθυμητού αποκωδικοποιητή με τις εισόδους του αποκωδικοποιητή που έχουμε στη διάθεσή μας θα βρούμε το σύνολο των επιπέδων.
12/4= 3 επίπεδα (η αρίθμησή τους είναι από (0 ως 2).
Οι 4K έξοδοι (και γενικά οι τελικές έξοδοι) εμφανίζονται στο τελευταίο επίπεδο.
Οι 12 είσοδοι εμφανίζονται ανά 4 σε κάθε επίπεδο (είναι κοινές στους αποκωδικοποιητές κάθε επιπέδου)
4 bit εισόδου στο επίπεδο 0 (υπάρχει 1 αποκωδικοποιητής)
4 bit εισόδου στο επίπεδο 1 (κοινά για όλους τους αποκωδικοποιητές του επιπέδου 1)
4 bit εισόδου στο επίπεδο 2 (κοινά για όλους τους αποκωδικοποιητές του επιπέδου 2)



Ο αποκωδικοποιητής που θέλουμε να κατασκευάσουμε με μέγεθος 12 x 4096 έχει 12 εισόδους (I11-I0)
Στο Επίπεδο 0 υπάρχει ένας αποκωδικοποιητής, με μέγεθος 4 x16. Αυτός δέχεται τα 4 πιο σημαντικά από τα 12 bit εισόδου του αποκωδικοποιητή μας (I11-I8).

Επειδή από τον αποκωδικοποιητή του επιπέδου 0 βγαίνουν 16 έξοδοι, στο επίπεδο 1 υπάρχουν 16 αποκωδικοποιητές μεγέθους 4 x 16. Καθένας από αυτούς έχει ένα σήμα

επίτρεψης λειτουργίας, (E0-E15). Κάθε φορά, ένας μόνο από αυτούς τους αποκωδικοποιητές του επιπέδου 1 επιτρέπεται να λειτουργήσει. ΠΟΙΟΣ; Είναι αυτός που θα ορίσουν τα 4 bit εισόδου I11-I8.

Επειδή από τους 16 αποκωδικοποιητές του επιπέδου 1 βγαίνουν 16 x 16 =256 έξοδοι, στο επίπεδο 2 υπάρχουν 256 αποκωδικοποιητές μεγέθους 4 x 16. Καθένας από αυτούς έχει ένα σήμα επίτρεψης λειτουργίας, (E0-E255). Κάθε φορά, ένας μόνο από αυτούς τους αποκωδικοποιητές του επιπέδου 2 επιτρέπεται να λειτουργήσει. ΠΟΙΟΣ; Είναι αυτός που θα ορίσουν τα 4 bit εισόδου I7-I4. ΠΡΟΣΟΧΗ: Τα bit I7-I4 είναι κοινά σε όλους τους αποκωδικοποιητές του επιπέδου 1. Επειδή στο επίπεδο 2 έχουμε 256 αποκωδικοποιητές με 16 εξόδους ο καθένας, αυτό σημαίνει ότι έχω 256 x 16= 4096 εξόδους συνολικά.

Ερώτηση: Έστω ότι ο αποκωδικοποιητής δέχεται ως είσοδο τον αριθμό 127. Να δείξετε την αποκωδικοποίηση. Ειδικότερα, να δείξετε ποιοι αποκωδικοποιητές ανά επίπεδο θα είναι ενεργοί και σε ποιον από τους αποκωδικοποιητές του τελευταίου επιπέδου βρίσκεται η επιθυμητή έξοδος.

Λύση: Καταρχήν πρέπει να γράψουμε τον αριθμό 127 με 12 bit: 000001111111

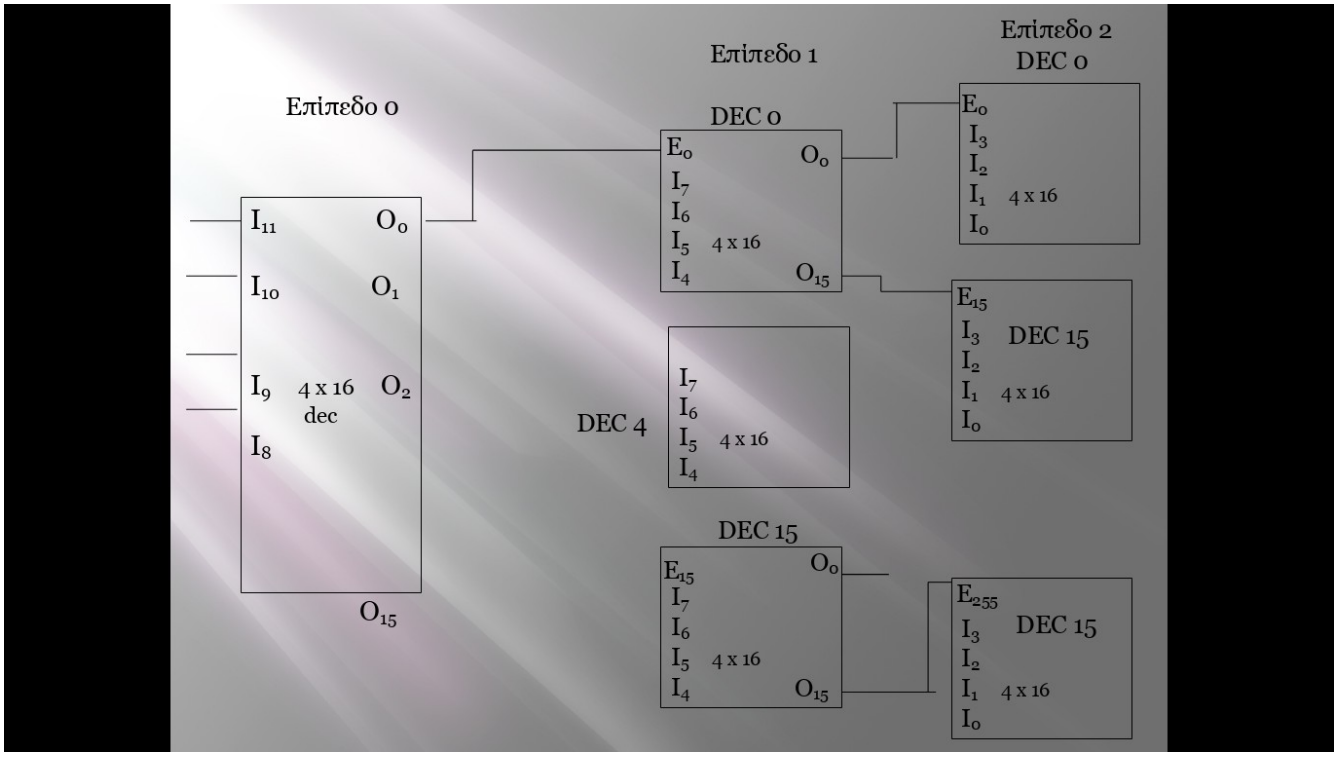
Χωρίζουμε τον αριθμό 000001111111 σε τρία τμήματα από 4 bit.
1ο τμήμα: τα 4 αριστερότερα bit, τα οποία είναι 0000. Αυτά θα τροφοδοτήσουν τις εισόδους I11-I8 του επιπέδου 0. Αυτό θα έχει ως αποτέλεσμα να γίνει 1 η έξοδος O0 του αποκωδικοποιητή του επιπέδου 0. Άρα θα γίνει 1 το σήμα επίτρεψης E0, που ενεργοποιεί τον DEC 0 του επιπέδου 1.

Τα 4 επόμενα bit του αριθμού 0000**0111**1111 είναι αυτά με έντονα γράμματα. Οι είσοδοι I7-I4 για όλους τους αποκωδικοποιητές του επιπέδου 1 (αυτές οι είσοδοι είναι κοινές) θα είναι 0111. Όμως, μόνο ο DEC0 είναι ενεργοποιημένος και επειδή οι είσοδοι I7 ως I4 σχηματίζουν τον αριθμό 7 (0111), η έξοδος O7 του DEC0 του επιπέδου 1 θα είναι ίση με 1. Αυτό σημαίνει ότι από τους 256 αποκωδικοποιητές του επιπέδου 2 θα ενεργοποιηθεί ο DEC7.

Τα 4 τελευταία bit του αριθμού 00000111**1111** είναι αυτά με έντονα γράμματα. Οι είσοδοι I3-I0 είναι κοινές για όλους τους αποκωδικοποιητές του επιπέδου 2 και είναι όλες ίσες με 1 (1111=15).
Όμως, μόνο ο DEC7 του επιπέδου 2 είναι ενεργοποιημένος. Αυτό σημαίνει ότι η έξοδος O15 του DEC7 είναι ίση με 1. Τελικά αυτή είναι και η μοναδική από τις 4096 εξόδους που θα είναι ίση με 1.

Ενεργοποιούνται ο αποκωδικοποιητής του επιπέδου 0, ο οποίος επιλέγει τον αποκωδικοποποιητή DEC0 του επιπέδου 1 (I11-I8). Στη συνέχεια ο DEC0 του επιπέδου 1 ενεργοποιεί τον DEC 7 του επιπέδου 2 (I7-I4). Στο επίπεδο 2 επιλέγεται η τελική έξοδος που είναι η 15.

ΠΑΡΑΔΕΙΓΜΑ 2: 1035.



ΠΑΡΑΔΕΙΓΜΑ 2: 1035 = 1024 + 8+2+1.
Με 12 bit: 010000001011

Χωρίζουμε τον αριθμό σε 3 τετράδες. Η πρώτη τετράδα είναι 0100. Αυτό σημαίνει ότι οι είσοδοι
I11=0
I10=1
I9=0
I8=0.

Αποτέλεσμα, επιλέγεται ο αποκωδικοποιητής DEC4 του επιπέδου 1, δηλαδή το E4 του επιπέδου 1 είναι 1.
Τα επόμενα 4 bit του αριθμού 0100**0000**1011 είναι 0000. Άρα
I7=0
I6=0
I5=0
I4=0.

Επομένως, επιλέγεται ο αποκωδικοποιητής 0 (ο οποίος συνδέεται με την έξοδο O0) του DEC4 του επιπέδου 1.
ΠΑΡΕΝΘΕΣΗ:
Ο DEC 0 του επιπέδου 1 βγάζει τους DEC 0-15
Ο DEC 1 του επιπέδου 1 βγάζει τους DEC 16-31 (Αυτή είναι η ΣΥΝΟΛΙΚΗ τους αρίθμηση, διότι με 4 bit έχουν τοπική αρίθμηση από 0-15)
Ο DEC 2 του επιπέδου 1 βγάζει τους DEC 32-47 (Αυτή είναι η ΣΥΝΟΛΙΚΗ τους αρίθμηση, διότι με 4 bit έχουν τοπική αρίθμηση από 0-15)

Ο DEC 3 του επιπέδου 1 βγάζει τους DEC 48-63 (Αυτή είναι η ΣΥΝΟΛΙΚΗ τους αρίθμηση, διότι με 4 bit έχουν τοπική αρίθμηση από 0-15)
Ο DEC 4 του επιπέδου 1 βγάζει τους DEC 64-79 (Αυτή είναι η ΣΥΝΟΛΙΚΗ τους αρίθμηση, διότι με 4 bit έχουν τοπική αρίθμηση από 0-15)

Τα πρώτα 8 bit του αριθμού 0100**0000** μου δείχνουν τη ΣΥΝΟΛΙΚΗ σειρά του επιλεγμένου αποκωδικοποιητή στο επίπεδο 2.

ΣΧΟΛΙΟ: Καθένας από τους αποκωδικοποιητές στο επίπεδο 1 ενεργοποιεί 16 αποκωδικοποιητές στο επίπεδο 2. Αυτοί οι αποκωδικοποιητές του επιπέδου 2 έχουν μία τοπική αρίθμηση από 0-15 (λογικό με 4 bit εισόδου). Όμως, αυτοί οι αποκωδικοποιητές έχουν και συνεχόμενη αρίθμηση (ή γενική αρίθμηση), η οποία μπορεί να βρεθεί από τα bit του εισερχόμενου αριθμού που αντιστοιχούν στα 2 πρώτα επίπεδα, δηλαδή τα 8 πρώτα bit. Επειδή αυτά είναι 01000000 = 64, ο αποκωδικοποιητής αυτός είναι συνολικά ο 64 στο επίπεδο 2.

Τέλος, επειδή τα τελευταία 4 bit είναι 1011, η έξοδος που επιλέγεται είναι η 11.

ΣΥΝΟΛΙΚΑ: Επιλέχθηκε ο αποκωδικοποιητής DEC4 στο επίπεδο 1. Αυτός επέλεξε τον αποκωδικοποιητή 0 με τον οποίο συνδέεται στο επίπεδο 2 (επειδή τα bit I7-I4) είναι 0000, αλλά αυτός ο αποκωδικοποιητής 0 είναι συνολικά ο αποκωδικοποιητής 01000000 = 64 του επιπέδου 2. Με άλλα λόγια, είναι ΤΟΠΙΚΑ ο αποκωδικοποιητής DEC 0 του επιπέδου 2 που συνδέεται με τον DEC4 του επιπέδου 1, αλλά ΣΥΝΟΛΙΚΑ είναι ο DEC64 (64 από 255). Τελικά επιλέγεται ως έξοδος η 11.

Με άλλα λόγια ο DEC 0 του επιπέδου 2, ο οποίος συνδέεται με την έξοδο O0 του DEC4 του επιπέδου 1 όπως προκύπτει από τα bit I7-I4, είναι ο DEC 64 του επιπέδου 2, όπως προκύπτει από τα bit I11-I4.

Το σήμα εξόδου 11 του DEC 64 που επιλέγει έχει συνολική διεύθυνση 1035 (το 1035 μέσα στο σύνολο σημάτων εξόδου από 0-4095).

Οι DEC 0-15 που ενεργοποιούνται από τον DEC0 του επιπέδου 1 παράγουν τα σήματα 0-255 (256 σήματα): 0**0000**000000000- 0**0000**011111111 (τα 4 πρώτα bit δείχνουν DEC0, επίπεδο 1).
Οι DEC 0-15 που ενεργοποιούνται από τον DEC1 του επιπέδου 1 παράγουν τα σήματα 256-511 (256 σήματα): **0001**00000000- **0001**11111111 (τα 4 πρώτα bit δείχνουν DEC1, επίπεδο 1).
Οι DEC 0-15 που ενεργοποιούνται από τον DEC2 του επιπέδου 1 παράγουν τα σήματα 512-767 (256 σήματα): **0010**00000000- **0010**11111111 (τα 4 πρώτα bit δείχνουν DEC2, επίπεδο 1).
Οι DEC 0-15 που ενεργοποιούνται από τον DEC3 του επιπέδου 1 παράγουν τα σήματα 768-1023 (256 σήματα): **0011**00000000- **0011**11111111 (τα 4 πρώτα bit δείχνουν DEC3, επίπεδο 1).
Οι DEC 0-15 που ενεργοποιούνται από τον **DEC4** του επιπέδου 1 παράγουν τα σήματα 1024-1279 (256 σήματα): **0100**00000000- **0100**11111111 (τα 4 πρώτα bit δείχνουν DEC4, επίπεδο 1).

Επομένως, το σήμα εξόδου 0 που ενεργοποιείται λόγω του **DEC4** του επιπέδου 1 είναι 1024. Άρα το ενδέκατο (επειδή 1011) είναι το 1035.

ΧΡΗΣΗ 2 ΕΙΔΩΝ DEC

Π.χ., να κατασκευαστεί Dec 7 x 128 με Dec 4 x 16 και 3x 8

Βρίσκουμε το κατάλληλο άθροισμα των εισόδων των διαθέσιμων decoders. Εδώ είναι φανερό

3+4=7

Αφού το άθροισμα αποτελείται από 2 αριθμούς, έχουμε 2 επίπεδα

Τοποθετούμε στο επίπεδο 0 έναν dec 3 x 8 και στο επίπεδο 1 έναν dec 4 x 16

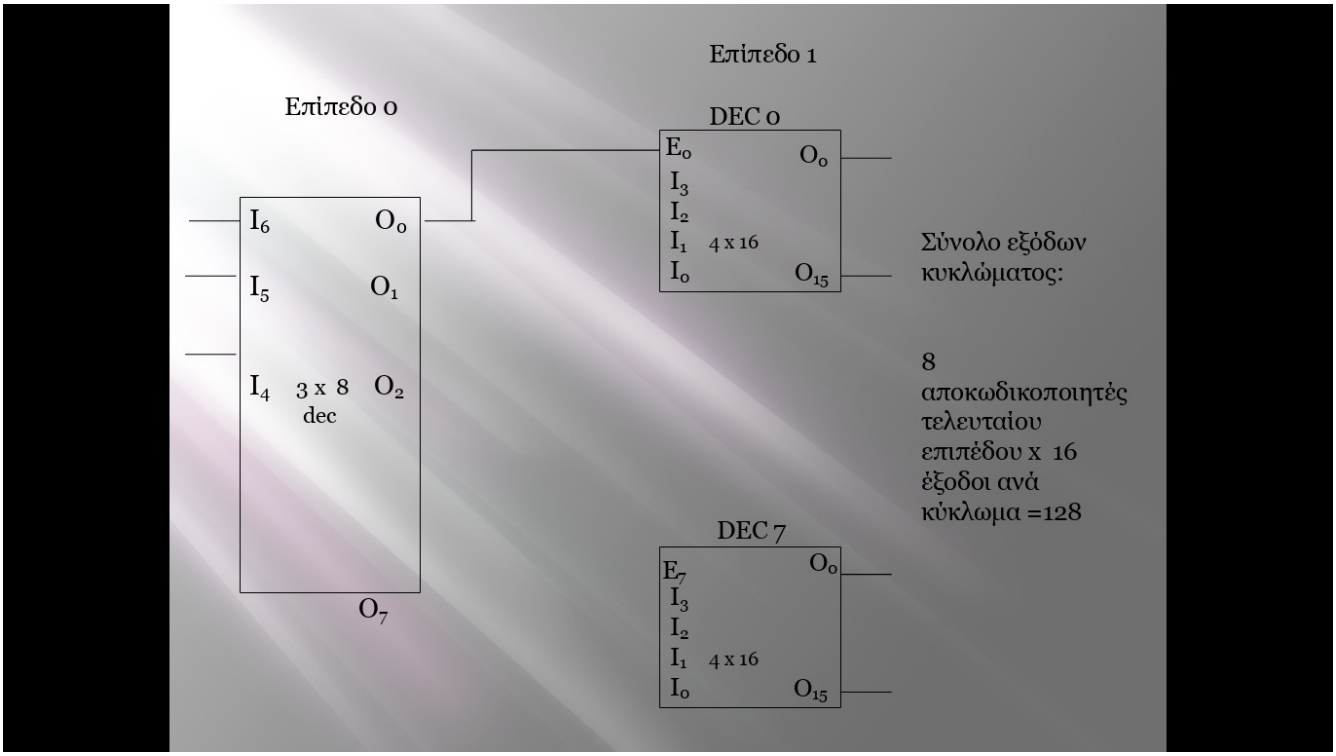
Θέλουμε έναν αποκωδικοποιητή με 7 εισόδους, και το 7 μπορεί να αναλυθεί σε άθροισμα 4+3. Άρα, επειδή έχω 2 αριθμούς σε αυτό το άθροισμα, θα έχω 2 επίπεδα. Στο πρώτο επίπεδο θα έχουμε έναν Dec 3x8 και στο δεύτερο 8 decoders 4 x 16.

Αν θεωρήσουμε ότι έχουμε 2 επίπεδα 0 και 1 και ότι στο επίπεδο 0 τοποθετούμε τον αποκωδικοποιητή 3x8, τότε θα υπάρχουν 8 έξοδοι, καθεμία από τις οποίες ενεργοποιεί έναν αποκωδικοποιητή 4 x 16. Άρα, στο επίπεδο 1 θα υπάρχουν 8 αποκωδικοποιητές, καθένας από τους οποίους θα έχει 16 εξόδους. Συνολικά 16 x 8 =128 έξοδοι.

Αν θεωρήσουμε ότι έχουμε 2 επίπεδα 0 και 1 και ότι στο επίπεδο 0 τοποθετούμε τον αποκωδικοποιητή 4x16, τότε θα υπάρχουν 16 έξοδοι, καθεμία από τις οποίες ενεργοποιεί έναν αποκωδικοποιητή 3 x 8. Άρα, στο επίπεδο 1 θα υπάρχουν 16 αποκωδικοποιητές, καθένας από τους οποίους θα έχει 8 εξόδους. Συνολικά 8 x 16 =128 έξοδοι.

Π.χ. 7x 128. Επειδή το 7 μπορεί να γραφτεί και 2+2+2+1, μπορούμε να έχουμε 4 επίπεδα με αποκωδικοποιητές 2x4(3 επίπεδα) και 1x2 (ένα επίπεδο)

16 x256 . Με 4x16 θέλουμε 4 επίπεδα, με 8 x 64 θέλουμε 2 επίπεδα, με χρήση 4x16 ΚΑΙ 8 x64 θα χρειαστούμε 8+4+4 (άρα 3 επίπεδα).



Παράδειγμα: Να δείξετε την αποκωδικοποίηση και τους ενεργοποιημένους αποκωδικοποιητές, αν η είσοδος του κυκλώματος είναι ο αριθμός 122. Να δώσετε την τοπική και συνολική αρίθμηση αυτών των αποκωδικοποιητών.

Αποκωδικοποιητής με 7 εισόδους και 128 εξόδους.
122=1111010.

Ο αριθμός 122 θα χωριστεί σε 2 τμήματα. Το πρώτο τμήμα είναι 3 bit (I6-I4) και το 2ο τμήμα είναι 4 bit (I3-I0).
Ο αποκωδικοποιητής του επιπέδου 0 δέχεται ως είσοδο 111. Αυτό σημαίνει ότι η έξοδος O7 είναι ίση με 1, άρα ενεργοποιείται ο DEC 7 του επιπέδου 1.
Οι άλλες 4 είσοδοι (I3-I0) είναι ίσες με 1010= 10. Αυτό σημαίνει ότι γίνεται 1 η έξοδος O10 του αποκωδικοποιητή DEC7 του επιπέδου 1.

Ο ενεργοποιημένος DEC είναι ο 7 και η ενεργοποιημένη έξοδος η 10 του DEC 7. Συνολικά, η έξοδος 10 του DEC 7 είναι η 122.

ΠΑΡΑΔΕΙΓΜΑ 2: Έστω ότι είχατε στη διάθεσή σας μόνο αποκωδικοποιητές 2 x 4 και 3 x 8. Να επαναλάβετε για την διεύθυνση 122.

Είσοδοι = 7:
2+2+3: 2x 4 στο επίπεδο 0, 2x 4 στο επίπεδο 1, 3x 8 στο επίπεδο 2,
3+2+2: 3x 8 στο επίπεδο 0, 2x 4 στο επίπεδο 1, 2x 4 στο επίπεδο 2,
2+3+2: 2x 4 στο επίπεδο 0, 3x 8 στο επίπεδο 1, 2x 4 στο επίπεδο 2.

Έστω ότι διαλέγουμε τον 2ο

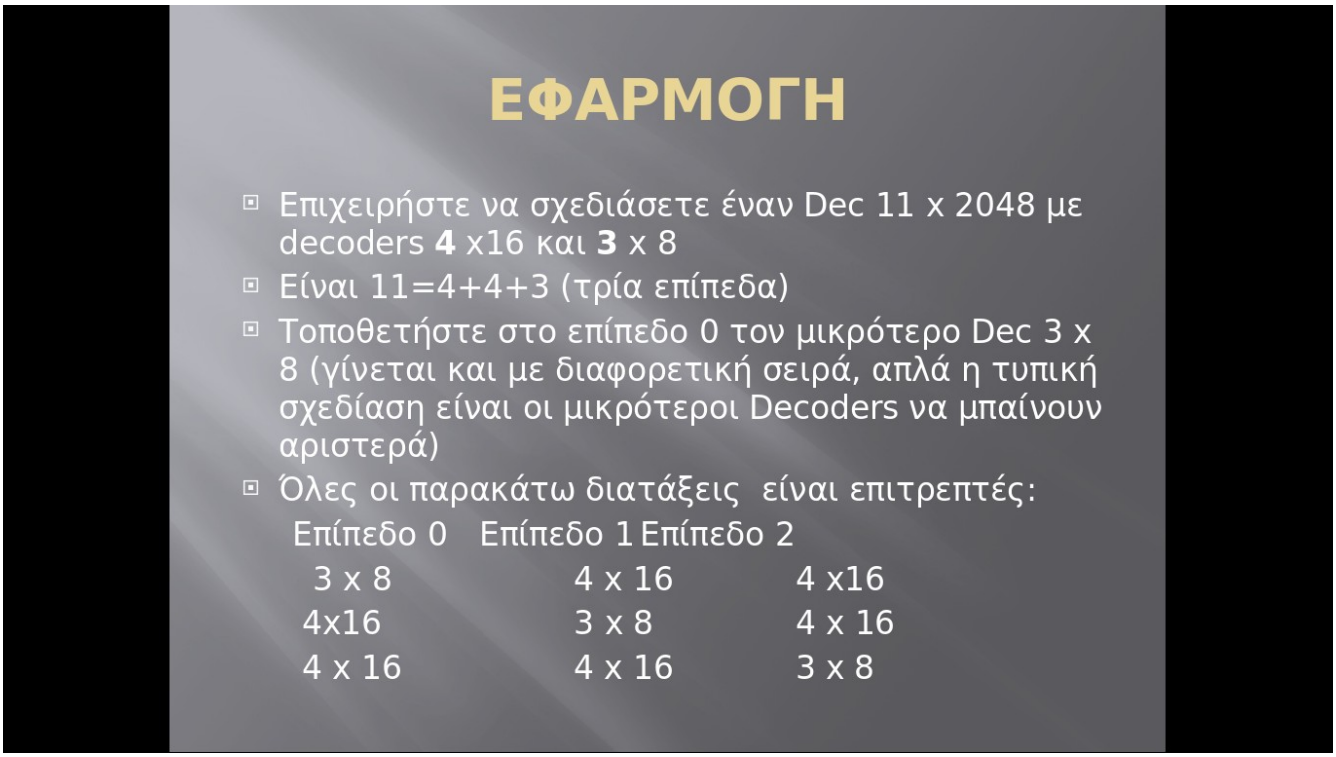
Άρα, το πρώτο επίπεδο είναι 3x8, το δεύτερο 2 x4 και το τρίτο 2x4.

Από το πρώτο επίπεδο όπου έχουμε 1 αποκωδικοποιητή 3 x 8 βγαίνουν 8 έξοδοι, οι οποίες ενεργοποιούν 8 αποκωδικοποιητές 2 x 4. Άρα, το επίπεδο 1 έχει 8 αποκωδικοποιητές 2 x 4 και συνολικά 8 x4 =32 εξόδους. Ο κάθε αποκωδικοποιητής 2x4 του επιπέδου 1, ενεργοποιεί 4 αποκωδικοποιητές 2x4 του επιπέδου 2. Άρα στο επίπεδο 2 υπάρχουν 8 x 4 = 32 αποκωδικοποιητές 2x4 άρα συνολικά 32 x 4 = 128 έξοδοι.

122 = 111 10 10
Χωρίζουμε τον αριθμό σε 3 κομμάτια:

1ο 111. Άρα, ο αποκωδικοποιητής του επιπέδου 0 επιλέγει τον αποκωδικοποιητή 7 του επιπέδου 1.
2ο 10. Αυτό σημαίνει ότι ο αποκωδικοποιητής 7 του επιπέδου 1 δέχεται ως είσοδο 10, που σημαίνει ότι η έξοδος 10=2 (O2) θα έχε τιμή 1. Αυτό σημαίνει ότι επιλέγει τον αποκωδικοποιητή 2 με τον οποίο συνδέεται στο επίπεδο 2 (ΤΟΠΙΚΗ). Συνολικά, ο αποκωδικοποιητής 2 του επιπέδου 2 ο οποίος ενεργοποιείται από τον αποκωδικοποιητή 7 του επιπέδου 1 είναι ο
(30). Άρα πρόκειται για τον αποκωδικοποιητή 30 του επιπέδου 2.
Τέλος, ο αποκωδικοποιητής 30 του επιπέδου 2 θα δώσει έξοδο 10=2. Η τελική έξοδος είναι η 1111010=122.

Η έξοδος 122 βρίσκεται στον αποκωδικοποιητή 30 του επιπέδου 2 και είναι η έξοδος 2 (ΤΟΠΙΚΑ)



DEC 13 x 2^{13} με DEC 5 x 32 και 3 x 8.

ΔΙΑΤΑΞΕΙΣ
5 x 32, 5x 32, 3x8
3x8, 5x32, 5x32
5x32, 3x8, 5x32

2) 20 x 2^{20}, ΜΕ DEC 4x16 και 8 x 256.

4 επίπεδα:
8x 256, 4x16, 4x16, 4x16
4x 16, 8x256, 4x16, 4x16
4x 16, 4x16, 8x256, 4x16
4x 16, 4x16, 4x16, 8x256

3 επίπεδα
8 x 256, 8x256, 4 x 16
8 x 256, 4x16, 8 x 256
4 x 16, 8x256, 8 x 256

ΠΑΡΑΤΗΡΗΣΗ: Αφού ορίσουμε τη διάταξη, γνωρίζουμε πόσα bit θα είναι η είσοδος για το επίπεδο 0 και πόσα τα bit για τις εισόδους στα άλλα επίπεδα.

Π.χ. ένας αποκωδικοποιητής 20x2^{20} με τρία επίπεδα σε διάταξη 4 x 16, 8x256, 8 x 256
Θα έχει:

I19- I16: Επίπεδο 0
I15-I8: Σε όλους τους dec του επιπέδου 1
I7-I0: Σε όλους τους dec του επιπέδου 2

Η είσοδος στον αποκωδικοποιητή 20 x 2^{20} με τιμή 10000000000000001111 θα διασπαστεί σε

1000
00000000
00001111

ΠΟΛΥΠΛΕΞΙΑ

Επιλογή μίας ανάμεσα σε πολλές γραμμές εισόδου ενός ψηφιακού συστήματος

Μέγεθος πολυπλέκτη 2^N σε 1 όπου

- N, γραμμές επιλογής
- 2^N οι γραμμές εισόδου
- Ενσωματωμένος αποκωδικοποιητής N x 2^N
- Επίσης, 2^N πύλες ΚΑΙ και μία Ή

Πολυπλεξία: επιλογή ανάμεσα σε πολλές γραμμές εισόδου και καθοδήγηση στη μοναδική έξοδο.

Μέγεθος 2^N x 1,
N γραμμές επιλογής και 2^N είναι οι γραμμές εισόδου. Το πλήθος γραμμών εισόδου από τις οποίες γίνεται η επιλογή της μοναδικής είναι δύναμη του 2, 2^N.
Επιλογή της γραμμής εισόδου, ουσιαστικά σημαίνει αποκωδικοποίηση.
2^N γραμμές εισόδου, για να επιλεγεί μία από αυτές, απαιτούνται Ν σήματα (γραμμές επιλογής)

Παραδείγματα: 8 x1 (N=3), 16x1 (N=4), 4 x 1(N=2)

Πχ πολυπλέκτης 8 x 1:

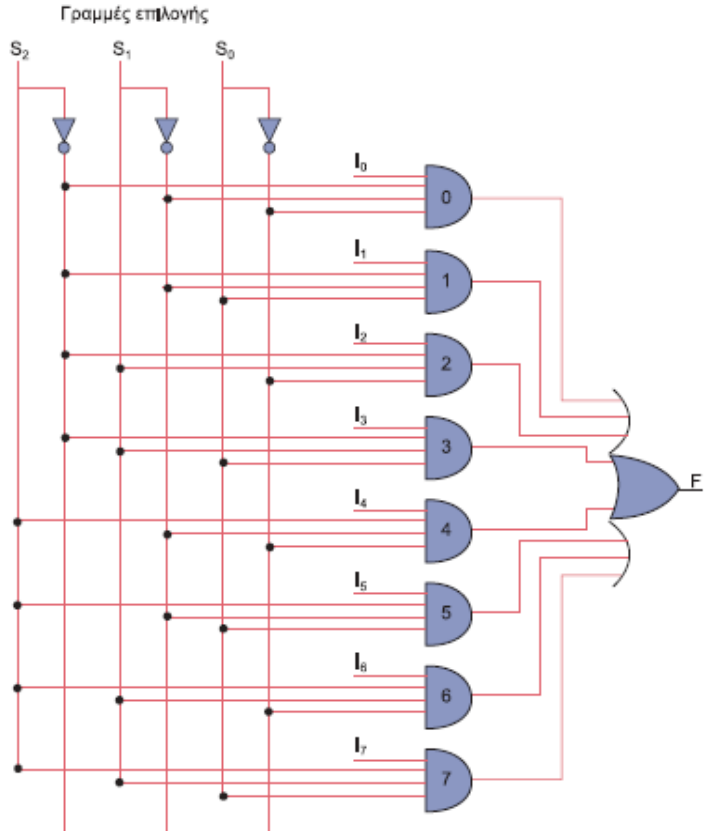
2^3 = 8 γραμμές εισόδου
3 γραμμές επιλογής, οι οποίες χρησιμοποιούνται για να επιλέξουν μία από τις 8 γραμμές εισόδου, δηλαδή εμπεριέχει έναν αποκωδικοποιητή 3 x 8
2^3= 8 πύλες ΚΑΙ, και μία πύλη Ή.

16 x 1 (N=4):

2^4 = 16 γραμμές εισόδου
4 γραμμές επιλογής, οι οποίες επιλέγουν μία από τις 16 γραμμές εισόδου, αποκωδικοποιητής 4 x 16
16 πύλες ΚΑΙ, μία πύλη Ή

ΣΥΣΤΑΤΙΚΑ ΠΟΛΥΠΛΕΚΤΗ
- N γραμμές επιλογής
- 2^N γραμμές εισόδου
- αποκωδικοποιητής N x 2^N
- 2^N πύλες AND
- 1 πύλη OR

ΠΑΡΑΔΕΙΓΜΑ ΠΟΛΥΠΛΕΚΤΗ 8 x 1 (σελ. 176)



S_2	S_1	S_0	F
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Πολυπλέκτης 8 x 1
Τα σήματα εισόδου είναι 8 (N=3). Τα σήματα αυτά είναι τα I0 – I7. Τα σήματα αυτά ΔΕΝ έχουν κάποια διασύνδεση με τις 3 γραμμές επιλογής.
Αν εξαιρέσουμε τα σήματα I0-I7 και την πύλη Ή στο τέλος, μας μένει ο ενσωματωμένος αποκωδικοποιητής 3 x 8.

Έστω ότι οι είσοδοι S2,S1, S0 είναι 000 δηλαδή σχηματίζουν τον αριθμό 0. Τότε, βάσει της λειτουργίας του αποκωδικοποιητή 3 x 8, οι έξοδοι των πυλών 1-7 είναι μηδενισμένες.
Η έξοδος της πύλης 0 θα είναι I0 AND 0’ AND 0’ AND 0’ = I0 AND 1 AND 1 AND 1 = I0. Άρα, η πύλη Ή τροφοδοτείται με 7 μηδενικά (από τις πύλες 1-7) και από το σήμα I0.
Αν I0 =0 τότε F=0
Αν I0 =1 τότε F=1
Τελικά, F=I0 (πρώτη γραμμή του πίνακα αληθείας του σχήματος)

Έστω ότι οι είσοδοι S2,S1, S0 είναι 001 δηλαδή σχηματίζουν τον αριθμό 1. Τότε, βάσει της λειτουργίας του αποκωδικοποιητή 3 x 8, οι έξοδοι των πυλών 0 και 2-7 είναι μηδενισμένες.
Η έξοδος της πύλης 1 θα είναι I1 AND 0’ AND 0’ AND 1 = I1 AND 1 AND 1 AND 1 = I1. Άρα, η πύλη Ή τροφοδοτείται με 7 μηδενικά (από τις πύλες 0 και 2-7) και από το σήμα I1.
Αν I1 =0 τότε F=0
Αν I1 =1 τότε F=1
Τελικά, F=I1 (δεύτερη γραμμή του πίνακα αληθείας του σχήματος)

Με παρόμοιο τρόπο παράγεται ο πίνακας αληθείας στο δεξί μέρος της διαφάνειας

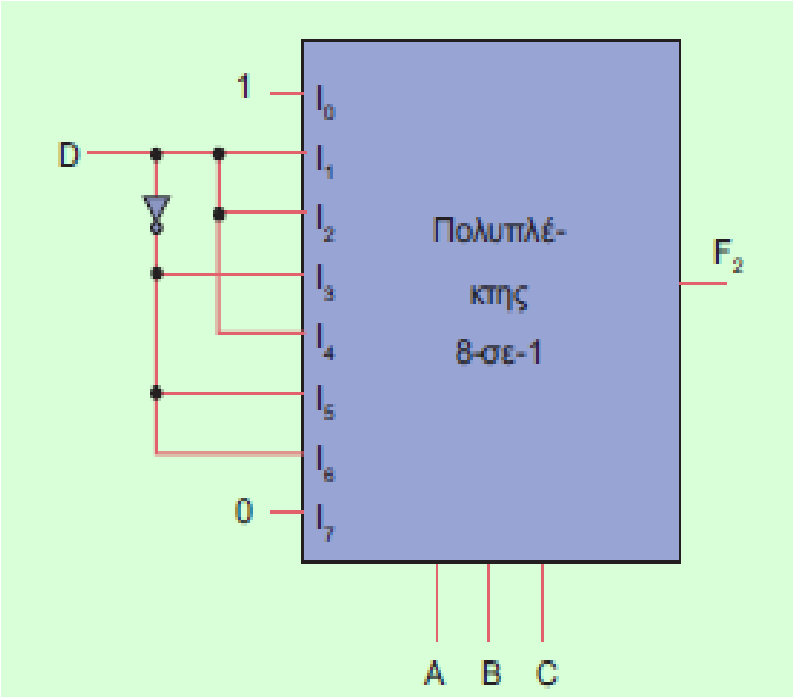
ΣΥΝΔΥΑΣΤΙΚΑ ΚΥΚΛΩΜΑΤΑ ΜΕ ΠΟΛΥΠΛΕΚΤΕΣ

Να υλοποιηθεί η $F_2(A,B,C,D)= \Sigma(0,1,3,5,6,9,10,12)$ με MUX 8 σε 1, όπου τα σήματα A,B,C τοποθετούνται στις γραμμές επιλογής, το D στις γραμμές εισόδου.

Λύση

Από την εκφώνηση: Τα A,B,C θα συνδεθούν στα σήματα επιλογής S_2, S_1, S_0

Το D στις 8 γραμμές εισόδου. ΠΩΣ; Με τέτοιον τρόπο ώστε να υλοποιείται ο πίνακας αληθείας της F_2



A	B	C	D	F_2
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Να υλοποιηθεί η $F_2(A,B,C,D)= \Sigma(0,1,3,5,6,9,10,12)$ με MUX 8 σε 1, όπου τα σήματα A,B,C τοποθετούνται στις γραμμές επιλογής**, το D στις γραμμές εισόδου.**

Υπάρχει απευθείας σύνδεση των σημάτων της συνάρτησης που ορίζονται από την εκφώνηση με τα σήματα επιλογής του πολυπλέκτη. Στο παράδειγμα, τα A,B,C συνδέονται με τα S2, S1, S0 αντίστοιχα.

Μας ενδιαφέρει να δούμε ποιες θα είναι οι συνδέσεις ανάμεσα στο D και τις γραμμές εισόδου I0-I7, έτσι ώστε να υλοποιείται τελικά η F2.

Τι σημαίνει ότι τα A, B, C έχουν συνδεθεί στα S2, S1, S0;

1) Όταν τα S2, S1, S0 δεχτούν ως είσοδο 0,0,0, τότε η έξοδος F2= I0. Στον πίνακα αληθείας, υπάρχουν 2 συνδυασμοί για τους οποίους A, B, C = 0,0,0. Άρα, για τις 2 αυτές περιπτώσεις, το D θα πρέπει να είναι κατάλληλα συνδεδεμένο με το I0. ΠΩΣ θα πετύχουμε την κατάλληλη σύνδεση; Συγκρίνουμε για τις 2 αυτές γραμμές του πίνακα αληθείας το D με το F2. Όταν D=0 ή D=1, είναι F2 =1. Αν συνδέσουμε το I0 με σταθερό σήμα 1, τότε, όταν A, B, C = 0,0,0, η έξοδος F2=1 και έχουμε υλοποιήσει τις 2 πρώτες γραμμές του πίνακα αληθείας.

2)) Όταν τα S2, S1, S0 δεχτούν ως είσοδο 0,0,1, τότε η έξοδος F2= I1. Στον πίνακα αληθείας, υπάρχουν 2 συνδυασμοί για τους οποίους A, B, C = 0,0,1. Άρα, για τις 2 αυτές περιπτώσεις, το D θα πρέπει να είναι κατάλληλα συνδεδεμένο με το I1. ΠΩΣ θα πετύχουμε την κατάλληλη σύνδεση;

Συγκρίνουμε για τις 2 αυτές γραμμές του πίνακα αληθείας το D με το F2. Όταν D=0, είναι F2 =0, D=1, είναι F2 =1. Άρα F2=D. Αν συνδέσουμε το I1 με το D, τότε, όταν A, B, C = 0,0,1, η έξοδος F2=D και έχουμε υλοποιήσει τις γραμμές 3-4 του πίνακα αληθείας.

ΠΑΡΑΤΗΡΗΣΗ: Με βάση τις τιμές των A,B,C, κάθε φορά ένα από τα σήματα εισόδου I θα βγει στην έξοδο, έστω $I_η$ δηλαδή θα είναι $I_η$=F2. Ουσιαστικά συγκρίνουμε το D με το F2 στις γραμμές του πίνακα αληθείας για τις οποίες επιλέγεται το $I_η$, και έπειτα συγκρίνουμε το D με το F2. Ότι έκφραση του D προκύψει από αυτή την εξίσωση, εξισώνεται στη συνέχεια με το $I_η$.

3) Όταν τα S2, S1, S0 δεχτούν ως είσοδο 0,1,0, τότε η έξοδος F2= I2. Στον πίνακα αληθείας, υπάρχουν 2 συνδυασμοί για τους οποίους A, B, C = 0,1,0. Άρα, για τις 2 αυτές περιπτώσεις, το D θα πρέπει να είναι κατάλληλα συνδεδεμένο με το I2. ΠΩΣ θα πετύχουμε την κατάλληλη σύνδεση; Συγκρίνουμε για τις 2 αυτές γραμμές του πίνακα αληθείας το D με το F2. Όταν D=0, είναι F2 =0, D=1, είναι F2 =1. Άρα F2=D. Αν συνδέσουμε το I2 με το D, τότε, όταν A, B, C = 0,1,0, η έξοδος F2=D και έχουμε υλοποιήσει τις γραμμές 5-6 του πίνακα αληθείας.

4) Όταν τα S2, S1, S0 δεχτούν ως είσοδο 0,1,1, τότε η έξοδος F2= I3. Στον πίνακα αληθείας, υπάρχουν 2 συνδυασμοί για τους οποίους A, B, C = 0,1,1. Άρα, για τις 2 αυτές περιπτώσεις, το D θα πρέπει να είναι κατάλληλα συνδεδεμένο με το I3. ΠΩΣ θα πετύχουμε την κατάλληλη σύνδεση; Συγκρίνουμε για τις 2 αυτές γραμμές του πίνακα αληθείας το D με το F2. Όταν D=0, είναι F2 =1, D=1, είναι F2 =0. Άρα F2=D'. Αν συνδέσουμε το I3 με το D', τότε, όταν A, B, C = 0,1,1, η έξοδος F2=D' και έχουμε υλοποιήσει τις γραμμές 7-8 του πίνακα αληθείας.

ΠΑΡΑΛΛΑΓΕΣ ΣΧΕΔΙΑΣΗΣ

Στο ίδιο παράδειγμα, ο πολυπλέκτης να είναι 4 σε 1, με τα A,B στις γραμμές επιλογής και τα C,D στις γραμμές εισόδου.

Λύση

Ο πίνακας αληθείας θα χωριστεί σε 4 τμήματα

A=B =0

A=0, B=1

A=1, B=0

A=B=1

Πίνακας αληθείας του πολυπλέκτη 4 x 1. Ο πολυπλέκτης διαθέτει 4 γραμμές εισόδου I0-I3.

A	B	F2 =
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Για κάθε συνδυασμό των A.B που βρίσκονται στις γραμμές επιλογής, θα πρέπει να εκφράζουμε τη συνάρτηση F2 σε μία απλοποιημένη (χάρτης Karnaugh) έκφραση των σημάτων (C,D) που συνδέονται στις γραμμές εισόδου I0-I3

Έστω A=B=0. Τότε, η έξοδος F2= I0. Άρα, πρέπει να συνδέσουμε την I0 με τα σήματα C, D με κατάλληλο τρόπο, έτσι ώστε να πετύχουμε τις 4 πρώτες γραμμές του πίνακα αληθείας.

Από τις 4 πρώτες γραμμές του πίνακα αληθείας, έχουμε ότι F2 = 1 όταν: (C=D=0) ή (C=0 ΚΑΙ D=1) ή (C=D=1).

Με χάρτη Karnaugh 2 μεταβλητών προκύπτει ότι C' + D. Άρα, το I0 θα πρέπει να συνδεθεί με το C+D

Όταν A=B=0, η F_2 δίνει 1 όταν

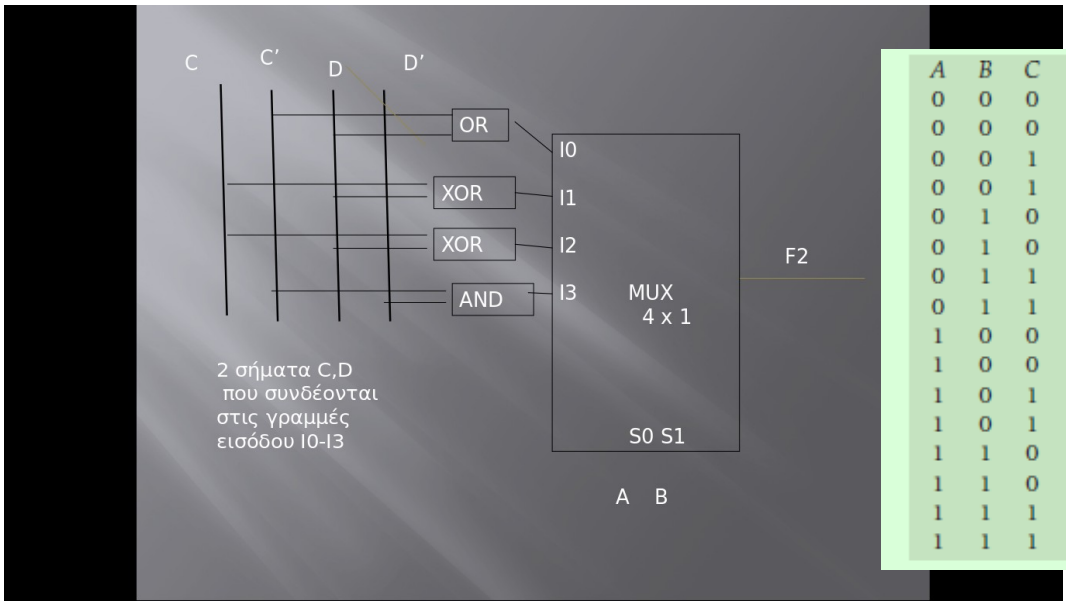
C=D=0,

C=0, D=1

C=D=1

Άρα, στο I_0 θα συνδεθεί το σήμα C' + D

Ομοίως, θα βρεθούν οι συνδέσεις και για τις άλλες εισόδους I_1=I_3



A	B	C	D	F_2
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Πίνακας αληθείας του πολυπλέκτη 4 x 1. Ο πολυπλέκτης διαθέτει 4 γραμμές εισόδου I0-I3.

A	B	F2 =
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Για κάθε συνδυασμό των A.B που βρίσκονται στις γραμμές επιλογής, θα πρέπει να εκφράζουμε τη συνάρτηση F2 σε μία απλοποιημένη (χάρτης Karnaugh) έκφραση των σημάτων (C,D) που συνδέονται στις γραμμές εισόδου I0-I3

	D	
C	1	1
		1

Έστω A=B=0. Τότε, η έξοδος F2= I0. Άρα, πρέπει να συνδέσουμε την I0 με τα σήματα C, D με κατάλληλο τρόπο, έτσι ώστε να πετύχουμε τις 4 πρώτες γραμμές του πίνακα αληθείας.
Από τις 4 πρώτες γραμμές του πίνακα αληθείας, έχουμε ότι F2 = 1 όταν: (C=D=0) ή (C=0 ΚΑΙ D=1) ή (C=D=1).
Με χάρτη Karnaugh 2 μεταβλητών προκύπτει ότι C' + D. Άρα, το I0 θα πρέπει να συνδεθεί με το C+D

Έστω A=0, B=1. Τότε, η έξοδος F2= I1. Άρα, πρέπει να συνδέσουμε την I1 με τα σήματα C, D με κατάλληλο τρόπο, έτσι ώστε να πετύχουμε τις γραμμές 5-8 του πίνακα αληθείας.

Από τις γραμμές 5-8 του πίνακα αληθείας, έχουμε ότι F2 = 1 όταν: (C=0 ΚΑΙ D=1) ή (C=1 ΚΑΙ D=0).
Άρα, F2= C'D+CD' = C XOR D

Έστω A=1, B=0. Τότε, η έξοδος F2= I2. Άρα, πρέπει να συνδέσουμε την I2 με τα σήματα C, D με κατάλληλο τρόπο, έτσι ώστε να πετύχουμε τις γραμμές 9-12 του πίνακα αληθείας.
Από τις γραμμές 9-12 του πίνακα αληθείας, έχουμε ότι F2 = 1 όταν: (C=0 ΚΑΙ D=1) ή (C=1 ΚΑΙ D=0).
Άρα, F2= C'D+CD' = C XOR D

Έστω A=1, B=1. Τότε, η έξοδος F2= I3. Άρα, πρέπει να συνδέσουμε την I3 με τα σήματα C, D με κατάλληλο τρόπο, έτσι ώστε να πετύχουμε τις γραμμές 13-16 του πίνακα αληθείας.
Από τις γραμμές 13-16 του πίνακα αληθείας, έχουμε ότι F2 = 1 όταν: (C=0 ΚΑΙ D=0).
Άρα, F2= C'D'

ΠΑΡΑΛΛΑΓΕΣ ΣΧΕΔΙΑΣΗΣ

Αν τα C,D τοποθετηθούν στις γραμμές επιλογής και τα A,B στις γραμμές εισόδου

Έστω ότι C=D=0 οπότε ως έξοδος επιλέγεται η I_0

Οι γραμμές που μας ενδιαφέρουν είναι οι 1, 5, 9, 13 του πίνακα αληθείας

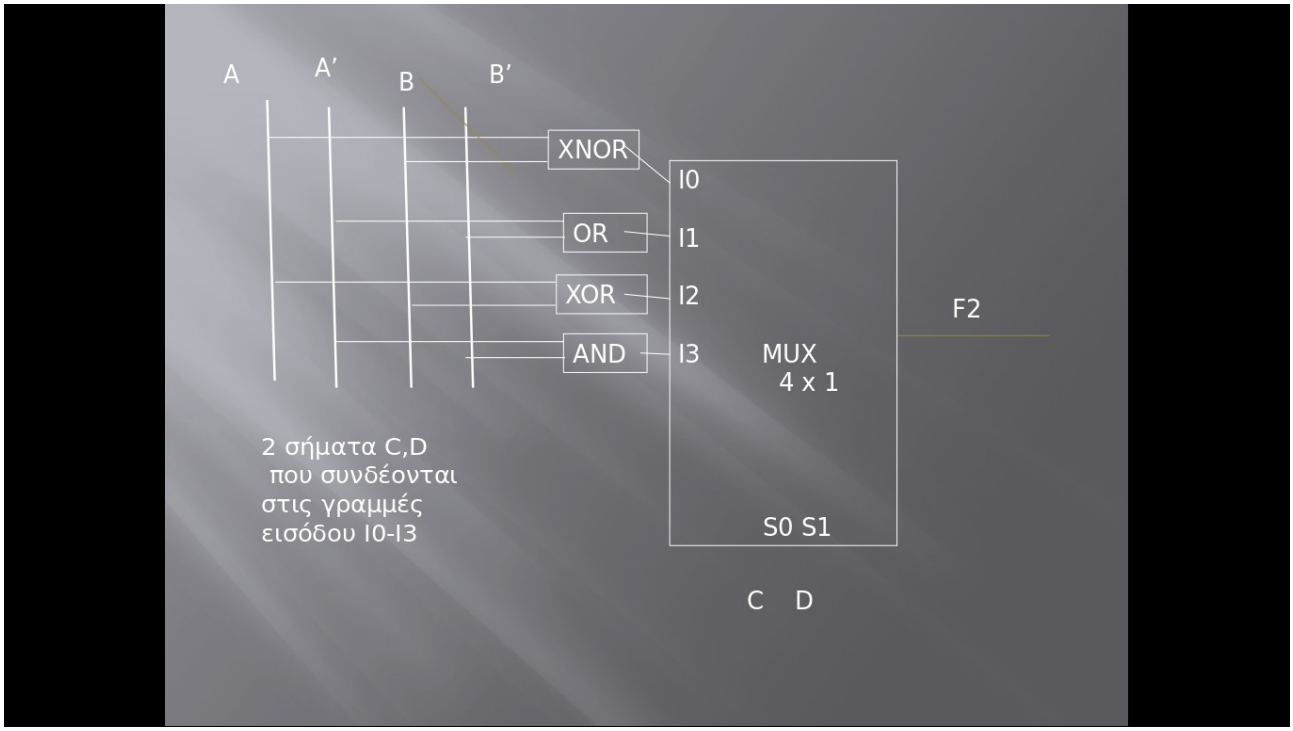
Όταν C=D=0, η F_2 δίνει 1 όταν

A=B=0 (γραμμή 1)

A=B=1 (γραμμή 13)

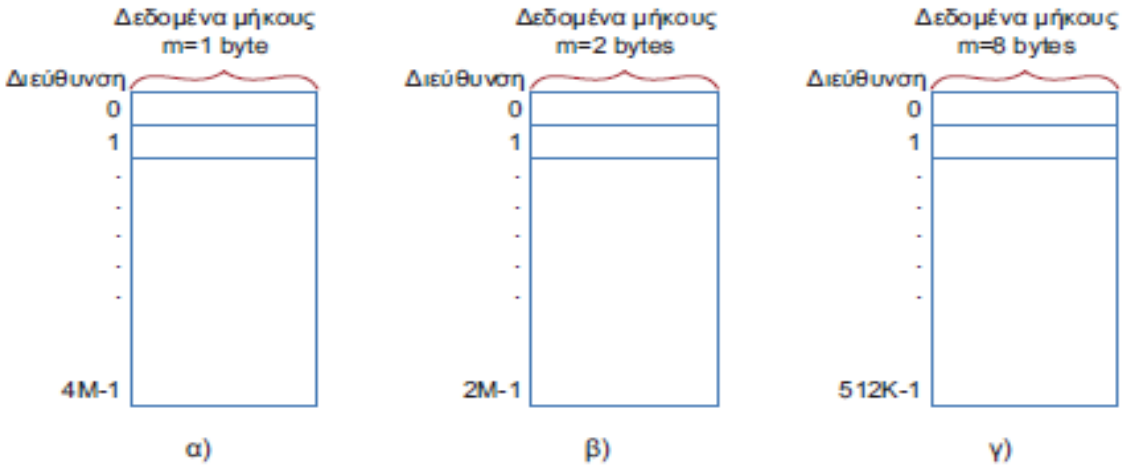
Άρα, η I0 θα συνδεθεί με το A'B' + AB (A XNOR B)

Ομοίως τα άλλα



Χωρητικότητα: Ορίζεται από το πλήθος λέξεων μνήμης και από το μέγεθος της λέξης.

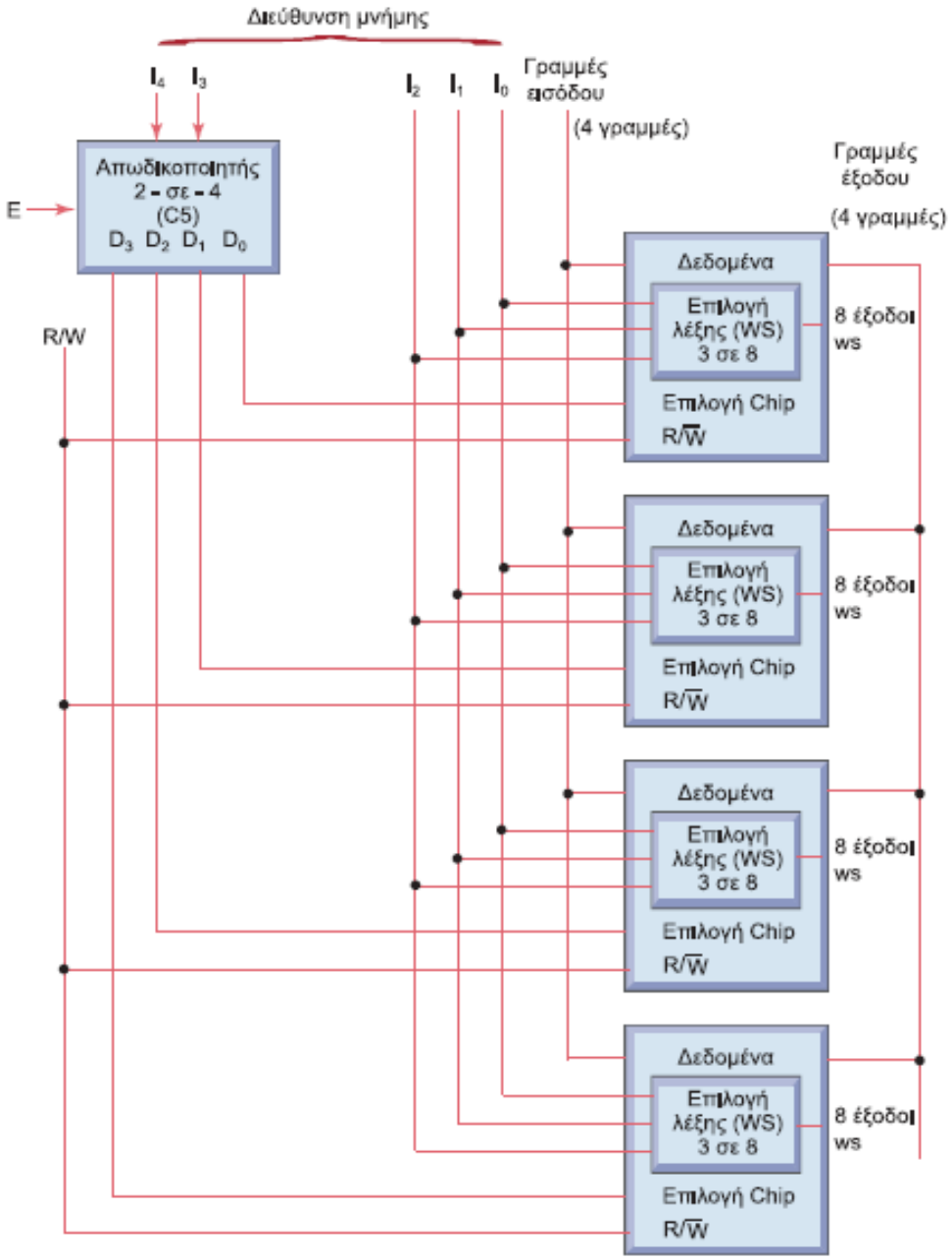
Π.χ., τρόποι υλοποίησης μνήμης 4Mbyte



Πλήθος λέξεων = Πλήθος των διευθύνσεων (Κάθε λέξη έχει τη δική της διεύθυνση)
Α) 4M x 1 = 4 Mbyte
Β) 2M x 2 = 4 Mbyte
Γ) 512K x 8 = 4 Mbyte

Περαιτέρω, μία μνήμη μπορεί να υλοποιηθεί με περισσότερα από 1 τσιπ μνήμης. Πχ 4 τσιπ, 1MB ανά τσιπ. Στην περίπτωση αυτή, για να βρεθεί η ζητούμενη λέξη (από τη CPU) πρέπει να αποκωδικοποιηθεί τόσο το τσιπ μνήμης στο οποίο βρίσκεται η λέξη όσο και η θέση της ζητούμενης λέξης μέσα στο τσιπ. Μιλάμε για αποκωδικοποίηση σε 2 επίπεδα.

ΟΡΓΑΝΩΣΗ ΜΝΗΜΗΣ ΜΕ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΕΣ (σελ. 475)



Διεύθυνση μνήμης.
Στο παράδειγμα έχουμε μία μνήμη μεγέθους 32 bytes. Η μνήμη αυτή είναι χωρισμένη σε 4 τσιπ, άρα κάθε τσιπ έχει μέγεθος 8 bytes.
32 bytes = 2^5.
Αυτό σημαίνει ότι απαιτούνται 5 bit για τη διευθυνσιοδότηση αυτής της μνήμης.
32 λέξεις ενός byte.

Μία μνήμη 32 byte θα μπορούσε να υλοποιηθεί με 16 λέξεις 2 byte. Τότε θα χρειαζόμασταν 4 bit.

00000 = δ 0 ως 11111 = δ.31

4 τσιπ, απαιτούνται 2 bit για την επιλογή τσιπ. Τα bit είναι τα πιο σημαντικά, δηλαδή I4 και I3. Το πλήθος των bit της διεύθυνσης μνήμης που χρησιμοποιούνται για αποκωδικοποίηση τσιπ είναι ίσο με log2 (M), όπου M είναι το πλήθος των τσιπ. Στο παράδειγμα M=4, άρα 2 bit απαιτούνται για επιλογή τσιπ. Επιλογέας τσιπ (chip select). Στο παράδειγμά μας, το CS είναι 2 x 4, και οι έξοδοι D0-D3 συνδέονται με ένα σήμα επιλογής τσιπ που υπάρχει σε καθένα από τα 4 τσιπ.

Τα τρία επόμενα bit της διεύθυνσης (I2-I0) χρησιμοποιούνται για την επιλογή λέξης εντός του τσιπ. Εφόσον έχουμε 8 λέξεις ανά τσιπ, απαιτείται αποκωδικοποιητής 3 x8 για την επιλογή λέξης. Αυτός ο αποκωδικοποιητής λέγεται επιλογέας λέξης (Word Select – WS).
Ενώ το CS Είναι ΕΝΑ για ολόκληρη τη μνήμη, υπάρχει ένα WS σε κάθε τσιπ. Τα τρία bit (I2-I0) είναι κοινά σε όλα τα WS. Αν υπάρχουν N λέξεις σε κάθε τσιπ, απαιτούνται log2(N) bit της διεύθυνσης μνήμης για επιλογή λέξης εντός του τσιπ.

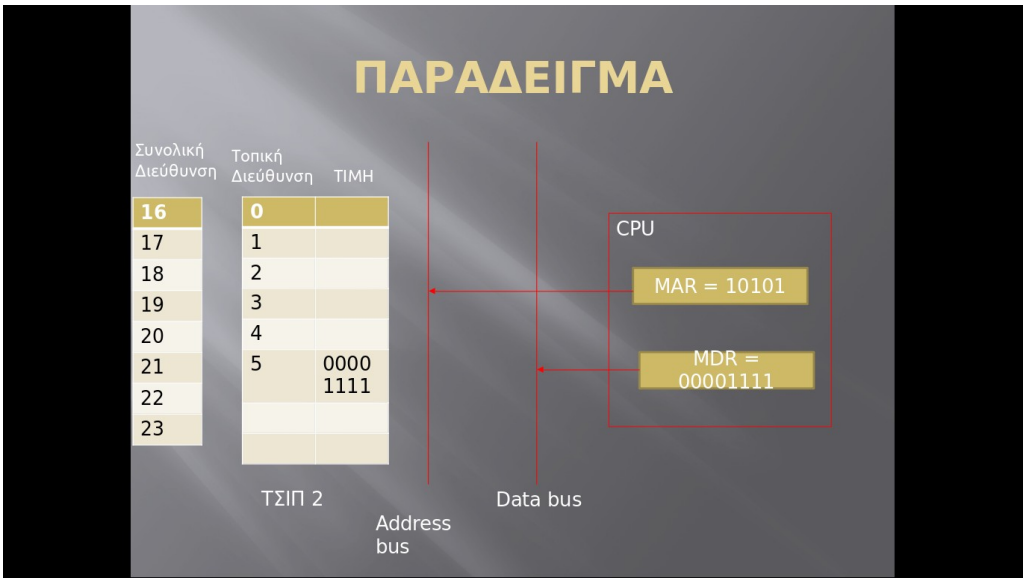
M+N = K, όπου 2^K είναι το πλήθος byte της μνήμης.

Λειτουργία αποκωδικοποίησης της RAM: Τα M πιο σημαντικά bit της εισερχόμενης διεύθυνσης μνήμης χρησιμοποιούνται ως είσοδοι του CS για επιλογή τσιπ. Τα υπόλοιπα N, τοποθετούνται ως είσοδοι σε όλα τα τσιπ WS για την επιλογή λέξης.

R/W: Όταν είναι 1 γίνεται ανάγνωση μνήμης, όταν είναι 0 εγγραφή στη μνήμη
Γραμμές εισόδου: Συνδέονται με τον έξω κόσμο έτσι ώστε η μνήμη να λαμβάνει δεδομένα (εγγραφή στη μνήμη)
Γραμμές εξόδου: Συνδέονται με τον έξω κόσμο έτσι ώστε η μνήμη να στέλνει δεδομένα (ανάγνωση από τη μνήμη)

ΠΑΡΑΔΕΙΓΜΑ: Έστω μία στιγμή η CPU ζητά να γράψει στη δ. 21 της μνήμης που φαίνεται στο παραπάνω σχήμα. Έστω ότι για να γράψουμε στη μνήμη πρέπει το R/W =1. Να δείξετε την αποκωδικοποίηση της μνήμης και τον τρόπο επιλογής της διεύθυνσης

Έχουμε λέξεις μεγέθους 1 byte, το οποίο σημαίνει ότι ο δίαυλος δεδομένων έχει μήκος 8 bit.
Έχουμε 32 λέξεις μνήμης έχουμε άρα 5 bit διευθυνσιοδότηση, άρα ο δίαυλος διευθύνσεων είναι 5 bit.
Η CPU ζητάει να γράψει στη δ. 21



Για να γράψουμε στη δ.21 πρέπει η CPU να ζητήσει τη διεύθυνση 21 για γράψιμο. Για να γίνει αυτό, η τιμή 21 (10101) γράφεται στον MAR (Memory Address Register – Καταχωρητής Διευθύνσεων Μνήμης), ο οποίος «μιλάει» με τον δίαυλο διευθύνσεων και έχει ως σκοπό να ζητάει δ. από τη μνήμη.
Υποθέτουμε ότι η τιμή που θα γραφτεί στη μνήμη είναι ο αριθμός 15 (00001111). Ο αριθμός αυτός τοποθετείται σε έναν ειδικό καταχωρητή (MDR-Memory DATA Register, καταχωρητής δεδομένων μνήμης), ο οποίος έχει σκοπό να μεταφέρει δεδομένα προς/ από τη μνήμη και «μιλάει» με τον δίαυλο δεδομένων.

1) Ο MAR περνάει την 5μπιτη διεύθυνση 21 = 10101 στον δίαυλο διευθύνσεων

Όταν η τιμή 10101 δοθεί από τον MAR στον δίαυλο διευθύνσεων, τα 5 bit περνάνε στα σήματα I4-I0

I4 I3 I2 I1 I0
1 0 1 0 1

Τα bit I4I3 είναι είσοδοι στο μοναδικό CS και η επιλεγμένη έξοδος είναι η D2. Δηλαδή, από 4 τσιπ

Chip 0, Chip 1, Chip 2, Chip 3, επιλέγεται το chip 2.

Chip 0: Λέξεις 0-7 (8 λέξεις)
Chip 1: Λέξεις 8-15 (8 λέξεις)
Chip 2: Λέξεις 16-23 (8 λέξεις)
Chip 3: Λέξεις 24-31 (8 λέξεις)

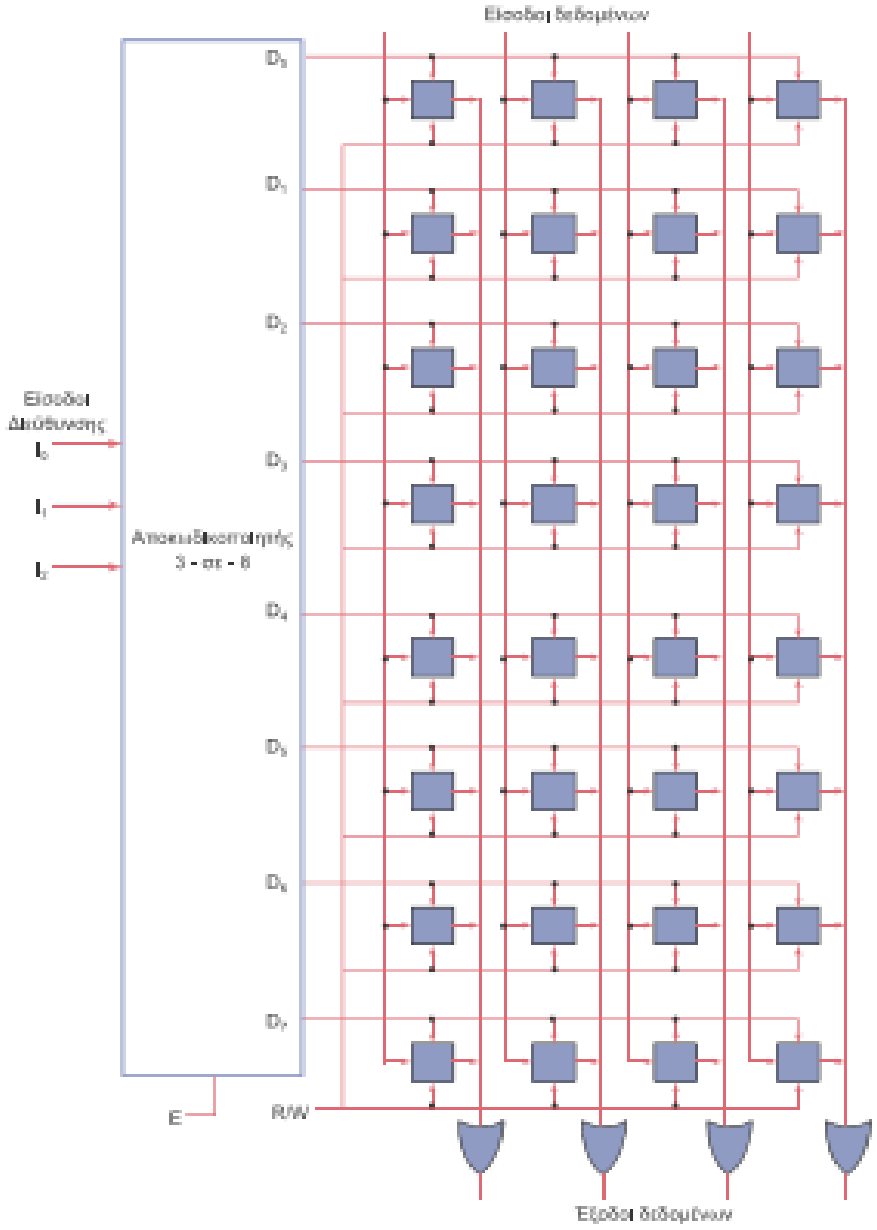
Κάθε τσιπ διαθέτει έναν αποκωδικοποιητή WS 3 x 8. Άρα, τοπικά (μέσα στο τσιπ) οι διευθύνσεις έχουν τιμές 0-7

Τα bit I2,I1,I0 θα δώσουν την τοπική διεύθυνση (εντός του τσιπ) της ζητούμενης λέξης. Επειδή η δ. είναι η 21=10101,
Αυτά τα bit είναι 101 = 5.
Αναζητούμε την λέξη με διεύθυνση 5 στο τσιπ με διεύθυνση 2.
Το σήμα R/W γίνεται 1 (εγγραφή).

Όταν η μνήμη αποκωδικοποιήσει την λέξη δηλαδή αποφασίσει ότι πρόκειται για τη λέξη 5 του τσιπ 2, τότε τα δεδομένα από τη CPU (ειδικότερα από τον MDR) θα πάνε στον δίαυλο δεδομένων.
Τα 8 bit του διαύλου δεδομένων θα περάσουν στη γραμμή εισόδου 8 bit της μνήμης και θα γραφτούν στην θέση 5 του τσιπ 2.

MAR <- PC (το πρόγραμμά μας θα συνεχιστεί από τη θέση μνήμης που ζητάει ο MAR)
MDR <- M[MAR]
M[MAR] <- MDR (Τα δεδομένα του MDR γράφονται στη θέση μνήμης που δείχνει ο MAR)

ΕΣΩΤΕΡΙΚΗ ΟΡΓΑΝΩΣΗ ΜΝΗΜΗΣ



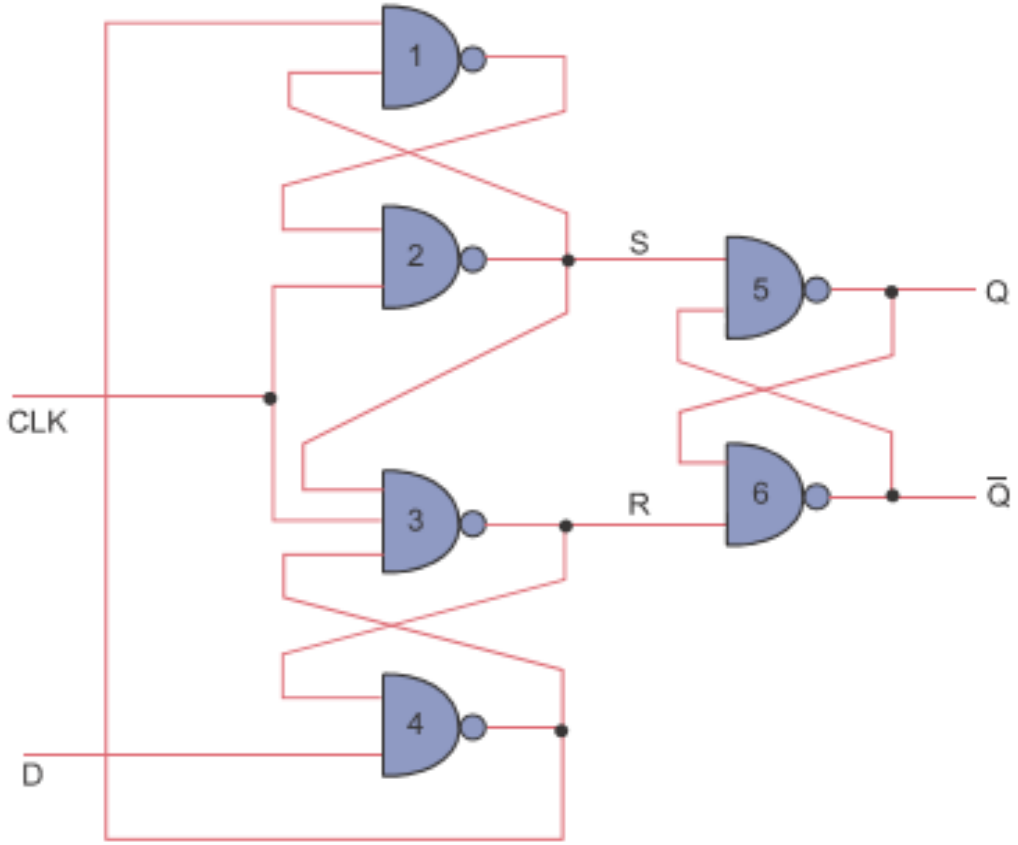
Εσωτερική οργάνωση **ενός τσιπ (1 από τα 4 του παραδείγματος, με τη διαφορά ότι θεωρούμε εδώ ότι έχουμε 4 bit λέξεις).** Έστω ότι η κάθε λέξη μας έχει μήκος 4 bit, και έχουμε 8 λέξεις ανά τσιπ.
Κάθε λέξη είναι μία οριζόντια γραμμή κυττάρων (Memory cells).
Το κάθε κύτταρο μνήμης είναι ένα flip-flop που αποθηκεύει ένα bit πληροφορίας.

Ο αποκωδικοποιητής αυτός είναι το WS. Δέχεται ως είσοδο 3 bit (τα λιγότερο σημαντικά bit της διεύθυνσης I2, I1, I0) και επιλέγει μία από τις γραμμές κυττάρων. Αν η μνήμη πρέπει να γράψει, τότε τα δεδομένα από τον MDR -> δίαυλος δεδομένων -> περνάνε στις γραμμές εισόδου της μνήμης (4 bit). Κάθε κελί μνήμης αποθηκεύει 1 bit. Αν επιλεγεί η λέξη με δ.5 τότε κάθε κύτταρο της γραμμής 5 θα αποθηκεύσει 1 bit πληροφορίας.

Αν η λέξη μνήμης επιλεγεί για ανάγνωση, αυτό σημαίνει ότι η CPU ζητάει τη λέξη για να την

επεξεργαστεί. Τότε τα bit από το επιλεγμένο κύτταρο θα βγουν στην έξοδο (στις γραμμές εξόδου), θα μπουν στον δίαυλο δεδομένων, και θα αποθηκευτούν στον MDR (πουθενά αλλού). Ο MDR είναι εκείνος που θα υποδεχτεί αυτά τα δεδομένα.

D FLIP-FLOP



Το ρολόι παράγει συνεχώς θετικούς και αρνητικούς παλμούς (0 και 1, εναλλάξ) οι οποίοι τυπικά έχουν σταθερό μήκος.
Το κύκλωμα αυτό είναι η βάση για την υλοποίηση των κυττάρων μνήμης.
Η λειτουργία είναι η εξής:
1) Όταν το ρολόι μεταβεί από 0->1 (θετική μετάβαση του ρολογιού), τότε και μόνο τότε, η τιμή D αποθηκεύεται στο κύτταρο. Η τιμή D έρχεται στις γραμμές εισόδου (από τον έξω κόσμο- από τον MDR-> δίαυλο δεδομένων).
2) Όσο το ρολόι είναι θετικό (ίσο με 1), ΚΑΜΙΑ αλλαγή δεν μπορεί να αποθηκευτεί
3) Όσο ο παλμός του ρολογιού είναι 0 ΚΑΜΙΑ αλλαγή στο κύκλωμα.

Η επιθυμητή λειτουργία είναι να συγχρονίζονται όλα τα κύτταρα, έτσι ώστε η αποθήκευση της πληροφορίας που έρχεται στην είσοδο να γίνεται μόνο σε συγκεκριμένη χρονική στιγμή.

Αποθήκευση κατά τη θετική μετάβαση: Θετικά ακμο-πυροδοτούμενα (positive edge-triggered)
Αποθήκευση κατά τη αρνητική μετάβαση: Αρνητικά ακμο-πυροδοτούμενα (negative edge-triggered)

Η αλλαγή της τιμής που αποθηκεύει ένα FF μπορεί να γίνει σε διαφορετικούς χρόνους (έναν από τους 2)

Α) Όταν το ρολόι έχει τιμή 0 και αλλάζει σε 1 (τη στιγμή της μετάβασης του ρολογιού από 0 σε 1, ΘΕΤΙΚΗ ΑΚΜΟΠΥΡΟΔΟΤΗΣΗ)
Β) Όταν το ρολόι έχει τιμή 1 και αλλάζει σε 0 (τη στιγμή της μετάβασης του ρολογιού από 1 σε 0, ΑΡΝΗΤΙΚΗ ΑΚΜΟΠΥΡΟΔΟΤΗΣΗ)

Η τιμή που αποθηκεύεται στη μνήμη είναι η τιμή της εξόδου Q. Τα FF έχουν τη δυνατότητα παραγωγής και της συμπληρωματικής εξόδου.

Το ρολόι εναλλάσσεται σε τιμές 0 και 1.

Έστω ότι αρχικά CLK =0 και το Q=0
CLK=0. Αυτό σημαίνει οι έξοδοι των πυλών 2,3 είναι ίσες με 1, δηλαδή τα σήματα S, R =1.
Αν Q=0, άρα Q'=1.
Άρα, η έξοδος της πύλης 6 θα παραμείνει 1 και επειδή Q'=1, η έξοδος της πύλης 5 θα παραμείνει 0.
1ο συμπέρασμα: Επομένως, αν CLK=0 δεν γίνεται ΚΑΜΙΑ ΜΕΤΑΒΟΛΗ ΣΤΙΣ εξόδους του ff (άρα και στο τι είναι αποθηκευμένο)

Έστω CLK=0 και Q=1, Q'=0
Άρα, η έξοδος της πύλης 6 θα παραμείνει 0 και επειδή Q'=0, η έξοδος της πύλης 5 θα παραμείνει 1.
2ο συμπέρασμα: Επομένως, αν CLK=0 δεν γίνεται ΚΑΜΙΑ ΜΕΤΑΒΟΛΗ ΣΤΙΣ εξόδους του ff (άρα και στο τι είναι αποθηκευμένο)

Όταν CLK =0 ισχύει ότι S=R=1. Αυτό έχει ως συνέπεια τα Q και Q' να διατηρούν τις τιμές τους.

Το ρολόι περνάει από 4 καταστάσεις που πρέπει να μελετηθούν: 1) Παραμένει 0, 2) Να μεταβεί από 0 σε 1, 3) Να παραμείνει 1, 4) Να μεταβεί από 1 σε 0.
Οι περιπτώσεις 1 και 4 αναλύθηκαν, το κύκλωμα διατηρεί την κατάστασή του.

2) Το ρολόι μεταβαίνει από 0 σε 1.
Το ρολόι ήταν 0 άρα κατά τη στιγμή της μετάβασης τα S, R ήταν 1.
Για να δούμε τι θα συμβεί στο κύκλωμα, εξετάζουμε το D.

2.1) D=0 Η πύλη 4 θα έχει έξοδο 1

- Η (4) δίνει έξοδο 1
- Ο άσσος της (4) είναι είσοδος της 1 και μαζί με το S=1 θα κάνουν την έξοδο της 1 ίση με 0
- Η (2) θα έχει μία είσοδο ίση με 0 άρα το S θα παραμείνει 1.

Όμως το CLK =1, S=1, η (4) =1 άρα η (3) =0, δηλαδή το R κάποια στιγμή αναπόφευκτα θα γίνει από 1 ->0
- Άρα Q'=1 και επειδή Q'=1 και S=1, θα είναι **Q=0.**

2.2) D=1, S=R=1

Η έξοδος της (4) θα είναι 0 (επειδή R=D=1)
Επειδή S=1 και (4)=0, η (1) θα έχει έξοδο 1
Άρα η είσοδοι της (2) είναι 1 (από την 1) και 1 από το CLK, άρα η έξοδος της (2) =0 => S=0
S=0 σημαίνει ότι **Q=1**.
Επειδή Q=1, R=1, Q'=0

Συμπέρασμα: Όταν το ρολόι μεταβεί θετικά (από 0 σε 1) τότε Q=D **Δηλαδή ότι τιμή υπάρχει στη γραμμή εισόδου της μνήμης η οποία συνδέεται με το κύτταρο θα γραφτεί στο κύτταρο.**

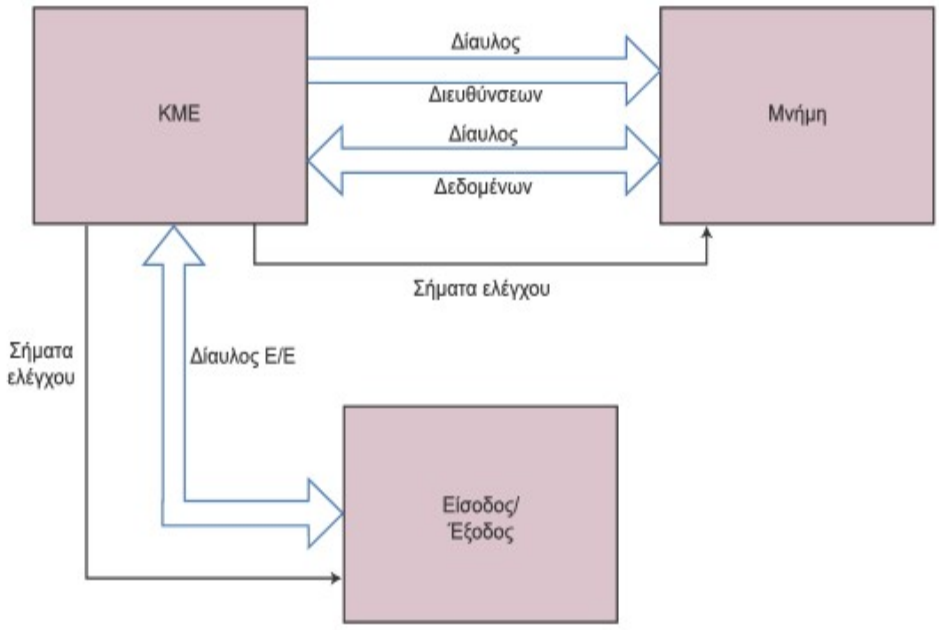
3) Συνέχεια της περίπτωσης 2.2. Έστω ότι Q=1, Q’=0 το CLK παραμένει 1, R=1. S=0. Έστω ότι έρχεται ένα bit από τον έξω κόσμο το οποίο είναι 0 και τροφοδοτεί το D. Άρα, το D αλλάζει. Αυτή η αλλαγή του D θα περάσει στην έξοδο (θα αποθηκευτεί;)

D=0. Άρα η (4) θα δώσει έξοδο 1. Η (1) έχει εισόδους 1 και 0 (από το S) οπότε η (1) = 1
(2) Θα έχει έξοδο 0 (έναν άσσο από την (1) και ένα από το ρολόι). Άρα το S θα παραμείνει 0
Επομένως και το R θα παραμείνει 1 (η πύλη 3 δέχεται ένα μηδενικό από το S).
Τελικά, S=0, R=1 => Q=1, Q’=0 δηλαδή **η τιμή του D δεν αποθηκεύεται όσο το ρολόι είναι 1.**

ΤΙ ΘΑ ΣΥΜΒΕΙ ΜΕΤΑ; Το ρολόι κάποια στιγμή θα πάει στο 0 οπότε το κύκλωμα παραμένει σταθερό. Μετά θα πάει από 0 σε 1 και η οποιαδήποτε τιμή του D θα αποθηκευτεί. Τελικά η λειτουργία είναι η εξής;

- Όταν το ρολόι είναι 0 το κύκλωμα δεν αλλάζει κατάσταση.
- Όταν μεταβεί από 0 σε 1, η τιμή που υπάρχει στις γραμμές εισόδου (άρα τροφοδοτεί το D) θα αποθηκευτεί.
- Όσο το ρολόι είναι 1 δεν γίνεται καμία αλλαγή.
- Όταν το ρολόι γυρίσει στο 0 δεν γίνεται καμία αλλαγή.
- Στην επόμενη θετική μετάβαση η τιμή που υπάρχει στις γραμμές εισόδου (άρα τροφοδοτεί το D) θα αποθηκευτεί.

ΣΧΕΔΙΑΣΗ CPU



Δίαυλος διευθύνσεων, μονής κατεύθυνσης υπό την έννοια ότι η ΚΜΕ ζητά διευθύνσεις από τη μνήμη. Για να ζητηθεί μία διεύθυνση η CPU την τοποθετεί σε έναν ειδικό καταχωρητή MAR (Memory Address Register) Καταχωρητής Διευθύνσεων Μνήμης. Ο MAR μιλάει αποκλειστικά με τον δίαυλο διευθύνσεων.

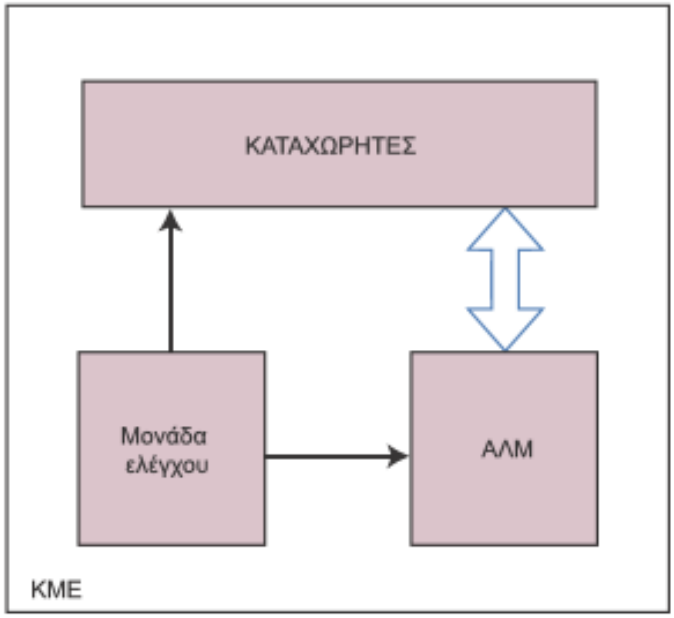
Κάθε φορά που ένα πρόγραμμα πρέπει να εκτελέσει μία εντολή ανάγνωσης ή εγγραφής μνήμης, πρέπει μία διεύθυνση να γραφτεί στον MAR, ο οποίος την αποστέλλει στη μνήμη και ακολουθεί αυτό το οποίο έχετε ακούσει ως αποκωδικοποίηση μνήμης.

Συμβολικά: MAR <- R (Ο MAR διαβάζει μία διεύθυνση, την οποία περιέχει ένας καταχωρητής R). Ο R δίνει μία διεύθυνση στον MAR προκειμένου αυτός να τη ζητήσει από τη μνήμη.

Ο δίαυλος δεδομένων έχει 2 κατευθύνσεις επειδή μπορούμε να στείλουμε δεδομένα από τη ΚΜΕ προς τη μνήμη (εγγραφή), ή από τη μνήμη προς την ΚΜΕ (ανάγνωση)
Ο καταχωρητής της ΚΜΕ ο οποίος αναλαμβάνει τη μεταφορά λέξεων μνήμης προς/από τη μνήμη είναι ο καταχωρητής δεδομένων μνήμης (Memory Data Register – MDR)

2 περιπτώσεις:
- M[MAR] <- MDR Ο MDR βάζει δεδομένα στον δίαυλο δεδομένων τα οποία θα γραφτούν στη μνήμη Μ στη θέση που έχει αποκωδικοποιηθεί. Τη θέση αυτή την δείχνει ο καταχωρητής MAR. Με άλλα λόγια, οποιαδήποτε εγγραφή στη μνήμη συμβολικά στη γλώσσα RTL γράφεται M[MAR] <- MDR.
- MDR <- M[MAR] Η μνήμη τοποθετεί στον δίαυλο δεδομένων τα περιεχόμενα της λέξης μνήμης που υπάρχουν στη θέση που έχει αποκωδικοποιηθεί. Στη θέση αυτή την δείχνει ο καταχωρητής MAR. Με άλλα λόγια, οποιαδήποτε ανάγνωση από τη μνήμη συμβολικά στη γλώσσα RTL γράφεται MDR<- M[MAR]

ΒΑΣΙΚΗ ΔΟΜΗ CPU



1. Δεν αναφερόμαστε σε σύστημα παράλληλης επεξεργασίας (παράλληλοι επεξεργαστές ή GPU)
2. Δεν αναφερόμαστε σε συστήματα που υλοποιούν διασωλήνωση εντολών
3. Αλλά, σε έναν απλό επεξεργαστή

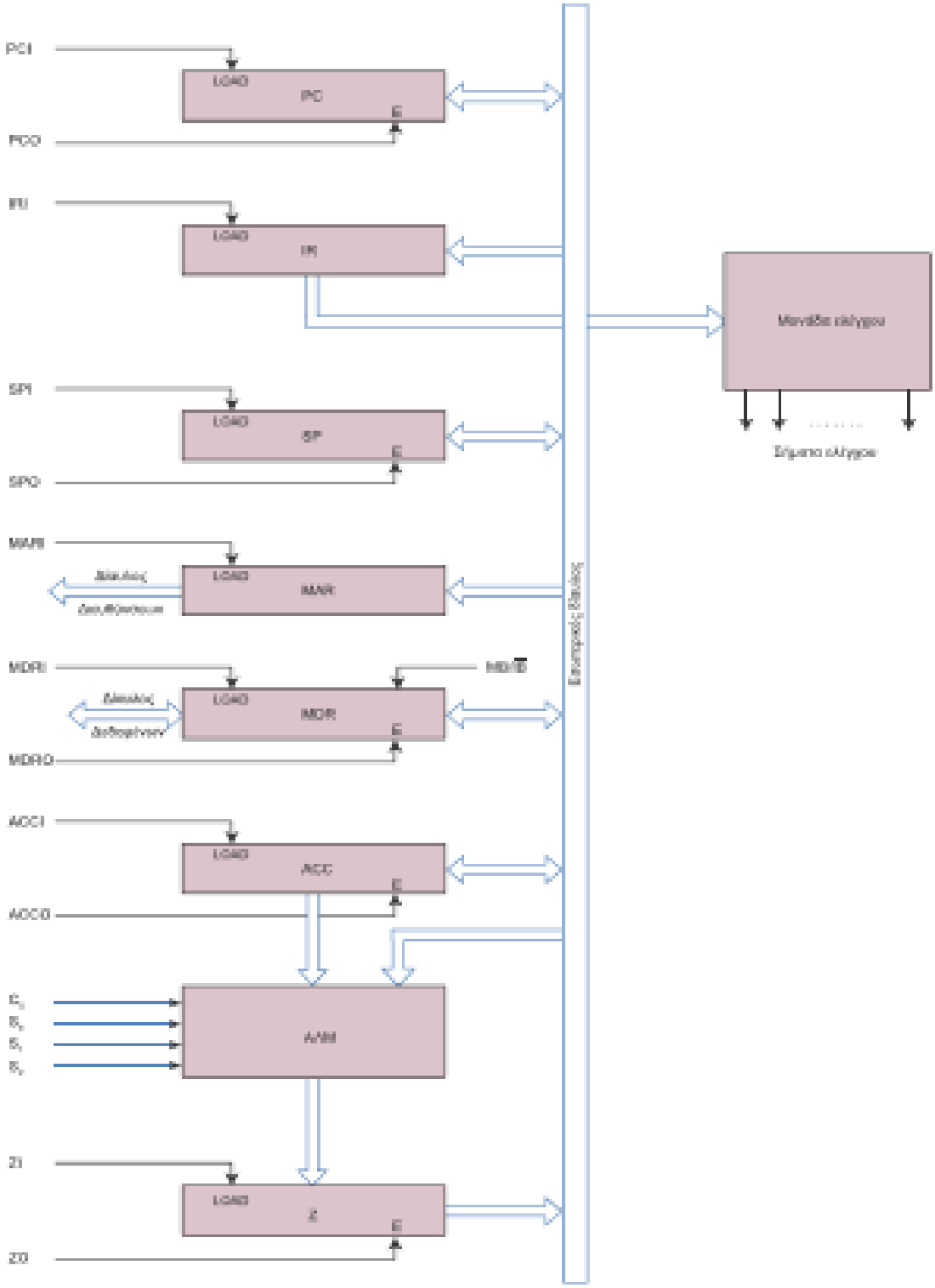
Καταχωρητές: Στοιχεία μνήμης της ΚΜΕ
Μονάδα Ελέγχου: Παράγει σήματα ελέγχου έτσι ώστε να γνωρίζει κάθε στοιχείο της ΚΜΕ τι πρέπει να κάνει και πότε.
ΑΛΜ: Είναι ΣΥΝΔΥΑΣΤΙΚΟ κύκλωμα (δεν διαθέτει στοιχεία μνήμης, δηλαδή δέχεται είσοδο παράγει έξοδο). Σε αυτήν υπάρχουν όλα τα κυκλώματα υλοποίησης των λογικών και αριθμητικών πράξεων. Επειδή δεν υπάρχει μνήμη απαιτείται καταχωρητής ο οποίος θα αποθηκεύσει το αποτέλεσμα.

Έστω ότι έχουμε την πράξη C=A+B όπου οι A,B,C είναι αποθηκευμένοι σε 3 θέσεις μνήμης A1, B1, C1.

1. Ζητούνται για ανάγνωση τα περιεχόμενα της διεύθυνσης A1: Ο MAR = A1, και η δ. A1 αποκωδικοποιείται από τη μνήμη. Η μνήμη θα βάλει τα περιεχόμενα της A1 στον δίαυλο δεδομένων και από εκεί θα πάνε στον MDR, Δηλαδή MDR=M[A1] = A.
2. Ο MDR μιλάει με τον δίαυλο δεδομένων, άρα επειδή πρέπει να φέρει άλλη μία λέξη από τη θέση μνήμης B1, θα πρέπει να δώσει τα περιεχόμενά του σε έναν προσωρινό καταχωρητή, έστω R. Άρα R<-MDR, R = A
3. Ζητούνται για ανάγνωση τα περιεχόμενα της διεύθυνσης B1: Ο MAR = B1, και η δ. B1 αποκωδικοποιείται από τη μνήμη. Η μνήμη θα βάλει τα περιεχόμενα της B1 στον δίαυλο δεδομένων και από εκεί θα πάνε στον MDR, Δηλαδή MDR=M[B1] = B.
4. Ο MDR και ο R έχουν σύνδεση με την ΑΛΜ. Επομένως, οι αριθμοί B και A τοποθετούνται ως είσοδοι στον αθροιστή (Η ΑΛΜ περιέχει πολλά κυκλώματα και έναν αποκωδικοποιητή ο οποίος επιλέγει πράξη).

5. Ο αθροιστής λαμβάνει ως εισόδους τους αριθμούς A και B, παράγει το αποτέλεσμα και το αποθηκεύει σε έναν καταχωρητή έστω Z.
6. Για να γραφτεί στη μνήμη το αποτέλεσμα, πρέπει να περάσει στον MDR (μόνο αυτός μιλάει με τον δίαυλο δεδομένων)
7. Πρέπει να ζητηθεί για εγγραφή η δ, C1, άρα MAR=C1 και η δ. C1 αποκωδικοποιείται από τη μνήμη.
8. M[MAR] <-MDR δηλαδή M[C1] = C (C=A+B)

ΚΑΤΑΧΩΡΗΤΕΣ ΤΗΣ CPU ΚΑΙ ΣΥΝΔΕΣΗ



Εσωτερικός δίαυλος: Είναι ο δίαυλος που χρησιμοποιείται για να μιλάνε μεταξύ τους οι καταχωρητές της ΚΜΕ. Στο παράδειγμα της πρόσθεσης, είδαμε ότι ο MDR μετέφερε τον αριθμό Α σε έναν γενικό καταχωρητή, έστω R. Αυτό έγινε μέσω του εσωτερικού διαύλου. Αν υποθέσουμε ότι ο R είναι απευθείας συνδεδεμένος με την ΑΛΜ (όπως είναι ο ACC στο σχήμα) ή έστω ότι αυτός ο R ήταν ο ACC. Στη συνέχεια, ο MDR διάβασε τον Β. Ο δίαυλος μπορεί να χρησιμοποιείται ΑΠΟΚΛΕΙΣΤΙΚΑ ΑΠΌ ΈΝΑΝ ΚΑΤΑΧΩΡΗΤΗ ΚΆΘΕ ΦΟΡΑ. Άρα ο MDR έγραψε τον αριθμό Β στον δίαυλο και από εκεί ο αριθμός τροφοδότησε την ΑΛΜ (τον αθροιστή). Ο αριθμός Α μεταφέρθηκε στην ΑΛΜ απευθείας από τον ACC.

ΕΡΩΤΗΣΕΙΣ
1. Γιατί δεν χρησιμοποιούν και ο MDR και ο ACC τον εσωτερικό δίαυλο και πηγαίνει ο ACC (R) απευθείας; ΔΕΝ γίνεται 2 καταχωρητές να γράψουν την ίδια στιγμή στον εσωτερικό δίαυλο και από την άλλη τα δεδομένα σε ένα συνδυαστικό κύκλωμα (δεν έχει μνήμη) πρέπει να πάνε ταυτόχρονα.
2. Γιατί δεν χρησιμοποιούμε 2 εσ. Διαύλους. Αυτό γίνεται και θα το δούμε. Προσφέρει κάποια πλεονεκτήματα (ταχύτητα εκτέλεσης εντολών) αλλά η περαιτέρω προσθήκη διαύλων δεν προσφέρει τίποτα
3. Διαχωρίζουμε τον εσ. Δίαυλο από τον δίαυλο δεδομένων και διευθύνσεων που είναι εξωτερικοί και με τους οποίους μιλάνε μόνο 2 καταχωρητές (MDR, MAR αντίστοιχα)
4. Που πάνε τα αποτελέσματα που βγαίνουν από την ΑΛΜ; Στο σχήμα φαίνεται ένα βελάκι που κατευθύνεται από την ΑΛΜ προς έναν καταχωρητή Z. Είναι φρόνιμο η ΑΛΜ να συνδέεται με έναν καταχωρητή ο οποίος λαμβάνει τα αποτελέσματα των πράξεων και να μην περιμένουμε αυτά να περάσουν σε έναν καταχωρητή μέσω διαύλου γιατί δεν μπορούμε να είμαστε σίγουροι ότι ο δίαυλος θα είναι ελεύθερος. Εάν έχουμε περισσότερους του ενός εσ. Διαύλους υπάρχει αυτή η εναλλακτική. Στο παράδειγμά μας με την πρόσθεση, η ΑΛΜ έβγαλε το αποτέλεσμα C το οποίο μεταφέρθηκε απευθείας στον καταχωρητή Z. Στη συνέχεια επειδή το αποτέλεσμα έπρεπε να γραφτεί στη μνήμη, ο Z το έριξε στον εσ. δίαυλο και το διάβασε ο MDR (ΠΩΣ ΗΞΕΡΕ Ο MDR ΟΤΙ ΠΡΕΠΕΙ ΝΑ ΤΟ ΔΙΑΒΑΣΕΙ; ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ δίνει εντολή στον MDR να διαβάσει τον εσ. Δίαυλο τη **συγκεκριμένη στιγμή**).
5. Αυτό σημαίνει ότι η εντολή πρόσθεσης δύο στοιχείων είναι ΠΡΟΚΑΤΑΣΚΕΥΑΣΜΕΝΗ. Οι καταχωρητές που εμπλέκονται ξέρουν από πριν πότε, και τι θα κάνουν. Δηλαδή ο R ξέρει ότι την Χ χρονική στιγμή θα διαβάσει τον δίαυλο για να πάρει τα δεδομένα από τον MDR, ο MDR ξέρει ότι την Υ χρονική στιγμή θα γράψει στον δίαυλο για να διαβάσει ο R, επίσης ο R ξέρει ότι την Κ χρονική στιγμή θα γράψει το αποτέλεσμα στον δίαυλο για τον διαβάσει ο MDR, ο MDR ξέρει ότι την χρονική στιγμή Δ θα διαβάσει από τον δίαυλο το αποτέλεσμα, ο MAR ξέρει πότε θα διαβάσει μέσω του διαύλου τις διευθύνσεις.

ΚΑΤΑΧΩΡΗΤΕΣ

PC (program counter), μετρητής προγράμματος: Αποθηκεύει τη διεύθυνση της **επόμενης εντολής από αυτή που εκτελείται:** Ένα πρόγραμμα εκτελεί τις εντολές τη μία μετά την άλλη. Οι εντολές αυτές είναι γραμμένες σε μία περιοχή της μνήμης, όπου υπάρχουν οι εντολές του κάθε προγράμματος. Όταν εκτελείται μία εντολή, ο PC αποθηκεύει τη διεύθυνση μνήμης όπου βρίσκεται η αμέσως επόμενη εντολή. Δηλαδή αν εκτελούμε μία εντολή ADD που έχει αποθηκευτεί στη θέση μνήμης Χ, τότε ο PC θα αποθηκεύει την τιμή Χ+1.

Παράδειγμα: Ο PC ζητάει την διεύθυνση μνήμης (μέσω του MAR) όταν ολοκληρωθεί η εκτελούμενη εντολή, η εντολή έρχεται στον MDR…… ο MAR Αρχικά ενημερώνεται από τον PC.

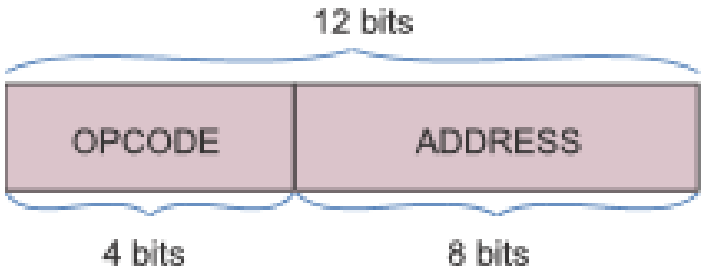
MAR: Να περιέχει και δίνει στη μνήμη της διεύθυνση της ζητούμενης λέξης. ΠΟΥ τη βρίσκει; Του την μεταφέρει ένας άλλος καταχωρητής. ΠΩΣ γίνεται η μεταφορά και γενικώς πως επικοινωνούν μεταξύ τους οι καταχωρητές της ΚΜΕ; **ΕΣΩΤΕΡΙΚΟΣ ΔΙΑΥΛΟΙ**

IR: Instruction Register Ο καταχωρητής περιέχει την εντολή η οποία εκτελείται, ειδικότερα τον κωδικό της εντολής που εκτελείται.
Το **OPCODE (κωδικός εντολής)** το οποίο αποθηκεύει ο IR, θα το στείλει στη **μονάδα ελέγχου**, η οποία παράγει τα σήματα ελέγχου που υλοποιούν την κάθε εντολή (τα σήματα διατάζουν κάθε καταχωρητή να εκτελέσει συγκεκριμένη δουλειά σε συγκεκριμένη στιγμή). Το OPCODE διαβάζεται από τη μνήμη.

ACC: κατεξοχήν καταχωρητής πηγή και προορισμός των πράξεων που εκτελούνται στην ΑΛΜ

Z: Προσωρινή αποθήκευση αποτελεσμάτων

ΜΙΑ ΒΑΣΙΚΗ ΟΡΓΑΝΩΣΗ CPU



Με OPCODE 4 bits, το σύστημα μπορεί να έχει 16 εντολές (2^4)
Με διευθύνσεις 8 bit, η μνήμη μας περιέχει ως 256 λέξεις

Μία εντολή ενός παράγοντα: Σημαίνει ότι αποτελείται από έναν κωδικό εντολής (OPCODE), που δείχνει ποια εντολή εκτελείται και μία διεύθυνση η οποία δείχνει σε ποια διεύθυνση μνήμης θα βρούμε τα δεδομένα για την εντολή.

Ερωτήματα:
1. Πόσο είναι το μήκος του OPCODE: Εξαρτάται από το πλήθος των εντολών. Αν υποθέσουμε ότι το σύστημά μας διαθέτει 16 εντολές άρα απαιτούνται 4 bit OPCODE. Με OPCODE n bit το σύστημά μας υποστηρίζει 2^n εντολές.
2. Το μήκος διεύθυνσης. Αν η διεύθυνση είναι m bits, τότε η μνήμη μας περιέχει 2^m διευθύνσεις. Εδώ έχουμε 256 διευθύνσεις.
3. Που αποθηκεύονται οι εντολές και τα δεδομένα για ένα πρόγραμμα. Αποθηκεύονται σε διαφορετικές περιοχές της μνήμης Ο λόγος είναι ότι οι εντολές πρέπει να είναι μαζεμένες, επειδή τυπικά ένα πρόγραμμα που τρέχει εκτελεί εντολές σειριακά,.
Πολύ γενικά, η εκτέλεση μίας εντολής ακολουθεί τα εξής βήματα: !) Η εντολή προσκομίζεται από τη μνήμη **(ΑΝΑΚΛΗΣΗ),** 2) Τα δεδομένα της εντολής προσκομίζονται στη CPU, 3) γίνεται η επεξεργασία, 4) αποθηκεύονται τα αποτελέσματα

Υπάρχουν και άλλες μορφές εντολών που θα συναντήσουμε:
Εντολές με 0 παράγοντες (έχουν OPCODE μόνο)
Εντολές 2 παραγόντων (υπάρχει OPCODE και 2 διευθύνσεις)
Εντολές 3 παραγόντων (υπάρχει OPCODE και 3 διευθύνσεις)

ΜΕΡΙΚΕΣ ΕΝΤΟΛΕΣ ΕΝΟΣ ΠΑΡΑΓΟΝΤΑ

Συμβολικό όνομα	Opcode	Περιγραφή
NOP	0000	Καμία λειτουργία
LDA	0001	Φόρτωση συσσωρευτή
STA	0010	Αποθήκευση συσσωρευτή
CLR	0011	Μηδενισμός συσσωρευτή
ADD	0100	Πρόσθεση της μνήμης στον συσσωρευτή
INC	0101	Αύξηση κατά 1 του συσσωρευτή
AND	0110	Λογικό AND της μνήμης και του ACC
NOT	0111	Συμπλήρωμα του συσσωρευτή

Με αυτό το «ρεπερτόριο», μία εντολή πρόσθεσης 2 μεταβλητών που θα γραφτεί για αυτό τον επεξεργαστή θα πρέπει να μεταφραστεί στις εξής εντολές (θα δοθεί η εντολή στη συνέχεια)

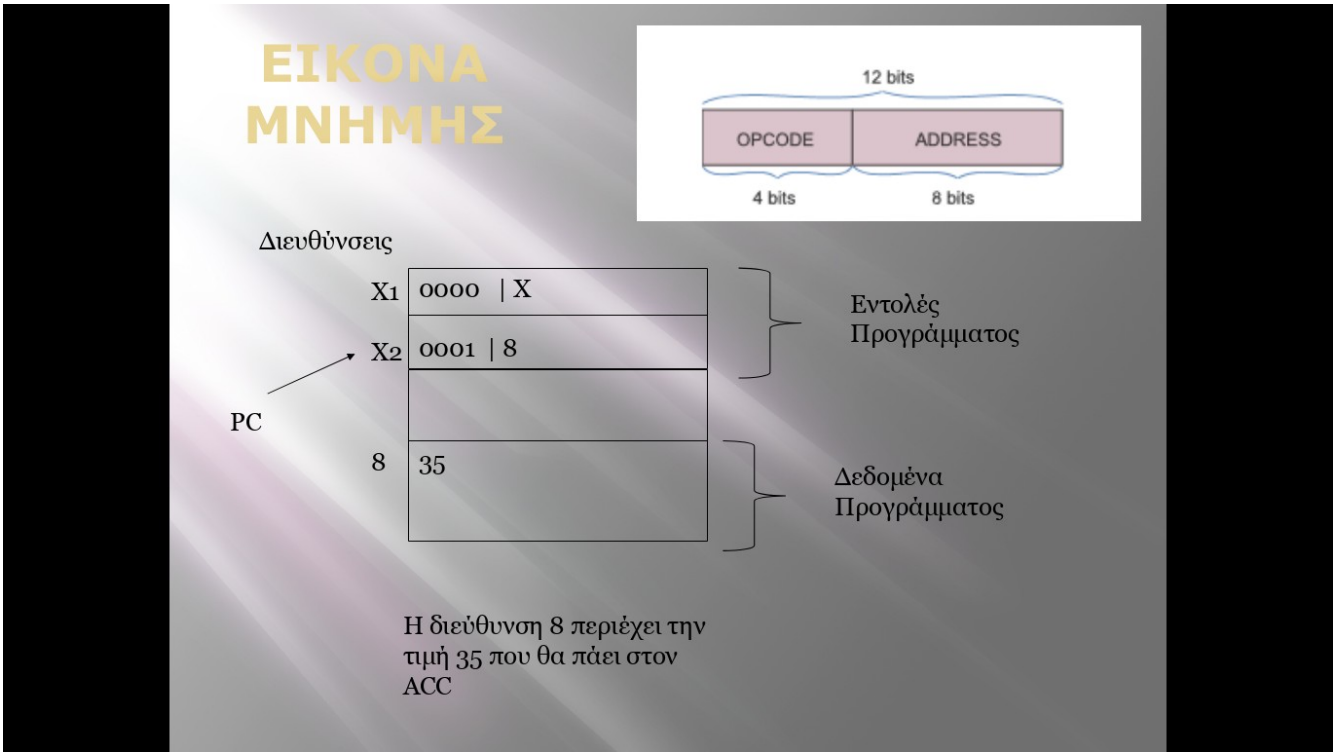
1. Να φορτώσει τη μία τιμή από τη μνήμη στο συσσωρευτή
2. Να φέρει τη δεύτερη τιμή στον MDR και να 3. προσθέσει με τον ACC
4. Να φορτώσει το αποτέλεσμα στον ACC
5. Να αποθηκεύσει το συσσωρευτή στη μνήμη

ΕΚΤΕΛΕΣΗ ΤΗΣ ΕΝΤΟΛΗΣ LDA

Από τη θέση μνήμης Α στον συσσωρευτή

Πολύ σημαντικό να ορίσουμε τις απαιτούμενες επικοινωνίες μεταξύ CPU και μνήμης σε μορφή βημάτων.

LDA : Έστω ότι το OPCODE της είναι το 0001 (δηλαδή είναι η εντολή με κωδικό 1 μέσα από ένα σύνολο 16 εντολών με κωδικούς 0-15)
Τα περιεχόμενα που βρίσκονται στη διεύθυνση που ορίζεται από το πεδίο ADDRESS (ο παράγοντας του οποίου η διεύθυνση δίνεται από το πεδίο Address) θα φορτωθεί από τη μνήμη στον ACC.



RESS (ο παράγοντας του οποίου η διεύθυνση δίνεται από το πεδίο Address) θα φορτωθεί από τη μνήμη στον ALDA : Έστω ότι το OPCODE της είναι το 0001 (δηλαδή είναι η εντολή με κωδικό 1 μέσα από ένα σύνολο 16 εντολών με κωδικούς 0-15)
Τα περιεχόμενα που βρίσκονται στη διεύθυνση που ορίζεται από το πεδίο ADDCC.

Το τμήμα της μνήμης που περιέχει τις εντολές του υποθετικού προγράμματος, περιέχει και την εντολή με OPCODE 0001, η οποία βρίσκεται στη θέση X2 της μνήμης και ακολουθεί την εκτέλεση της εντολής που βρίσκεται στη θέση X1 (μετά την NOP). Ο παράγοντας της εντολής που μας ενδιαφέρει, δηλαδή της LDA είναι 8. Αυτό σημαίνει ότι η εντολή μας έχει τη μορφή

ΕΝΤΟΛΗ της διεύθυνσης X2: 0001 | 00001000

Επίσης:
1) Τα δεδομένα της εντολής LDA βρίσκονται στη θέση μνήμης με διεύθυνση 8. Αυτό σημαίνει ότι στον ACC τελικά θα φορτώσουμε τα περιεχόμενα της θέσης μνήμης 8, δηλαδή την τιμή 35. Άρα, όταν ολοκληρωθεί η εντολή ο ACC←35,
2) Η εντολή όπως είναι, δηλαδή 0001 | 00001000, πρέπει να προσκομιστεί στη CPU

ΑΝΑΚΛΗΣΗ: Μέχρι η εντολή να ανακληθεί και το OPCODE να τοποθετηθεί στον καταχωρητή IR (ο οποίος θα το στείλει στην μονάδα ελέγχου για να παραχθούν τα σήματα ελέγχου της εντολής), η CPU δεν έχει ιδέα ποια εντολή θα εκτελέσει. Στο παράδειγμα, εκτελεί την NOP αλλά δεν ξέρει τι ακολουθεί.
Για τον λόγο αυτό, η ανάκληση είναι ίδια για όλες τις εντολές.

PC: δείχνει τη διεύθυνση της επόμενης προς εκτέλεση εντολής. Δηλαδή, όταν εκτελείται η NOP (X1) ο PC έχει ήδη την τιμή της X2 (την έχει πάρει από την ανάκληση της εντολής που βρίσκεται στην X1).
Άρα, για να ζητηθεί η διεύθυνση μνήμης X2, θα πρέπει ο PC να δώσει τη δ. X2 στον MAR Δηλαδή, ο MAR θα διαβάσει από τον PC τη δ. X2 για να ζητήσει τη λέξη

0001 | 00001000 από τη μνήμη. Στη συνέχεια, ο PC πρέπει να αυξηθεί κατά 1 για να δείξει την επόμενη εντολή που θα εκτελεστεί.

Ο καταχωρητής που θα λάβει τη λέξη που βρίσκεται στη δ. Χ2 είναι ο MDR και στη συνέχεια θα δώσει το OPCODE στον IR

Η ανάκληση και η εκτέλεση υλοποιούνται με μία σειρά από μικρολειτουργίες όπου κάθε μικρολειτουργία είναι η μικρότερη δυνατή λειτουργία που μπορεί να γίνει σε μία περίοδο ρολογιού (το χρονικό διάστημα ώστε το ρολόι από 0 να πάει στο 1 και να γυρίσει στο 0). Οι περίοδοι αυτοί τις συμβολίζουμε με Tį

ΑΝΑΚΛΗΣΗ

T0: MAR <-PC, Z<- PC+1, στο παράδειγμα, MAR=X2, ο Z= X2+1 (ΑΚΟΛΟΥΘΕΙ Η διαδικασία αποκωδικοποίησης μνήμης, ο MAR ζητάει μία διεύθυνση)

T1: MDR<-M[MAR], PC<-Z, στο παράδειγμα MDR= 0001 | 00001000, PC= X2+1 (PC έχει τη διεύθυνση της επόμενης εντολής που θα εκτελεστεί)

T2: IR<- MDR[OPCODE], IR<- 0001 (**μόνο τώρα ξέρει η CPU τι θα εκτελέσει)**

T0: Ο εσωτερικός δίαυλος μπορεί να χρησιμοποιηθεί σε κάθε περίοδο του ρολογιού από έναν καταχωρητή (οπτικά, αυτός είναι ο καταχωρητής που βρίσκεται δεξιά από το βελάκι). Ο PC βάζει την τιμή στον δίαυλο και από τη μονάδα ελέγχου δίνονται τα κατάλληλη σήματα για να διαβαστεί αυτή η τιμή από τον MAR και από την ΑΛΜ (η ΑΛΜ είναι συνδυαστικό κύκλωμα, δεν έχει μνήμη, δεν θα γράφουμε ποτέ ALU, η ALU υπονοείται επειδή έχουμε πρόσθεση). Η ΑΛΜ δεν μπορεί να βγάλει το αποτέλεσμα στον δίαυλο, επειδή αυτός χρησιμοποιείται σε αυτή την περίοδο από τον PC.

T1: Ο MDR λαμβάνει από τον δίαυλο δεδομένων την λέξη 0001 | 00001000 ενώ ο PC ενημερώνεται για την νέα του τιμή από τον Z μέσω του εσ. διαύλου.

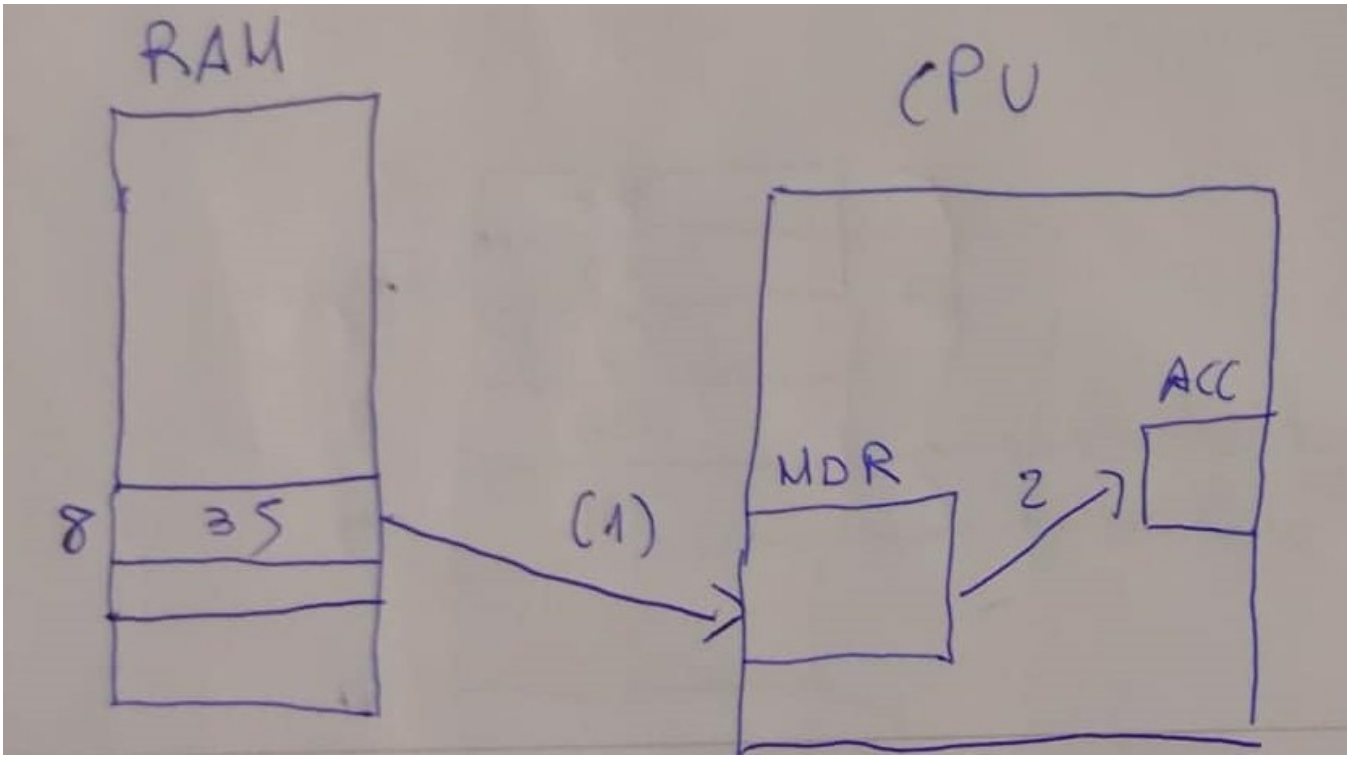
T2: Όταν ο IR διαβάσει το OPCODE τότε πλέον γνωρίζουμε ποια εντολή πρέπει να εκτελεστεί. Άρα, η Μονάδα ελέγχου θα παράγει τα κατάλληλα σήματα για την εκτέλεση της εντολής. ΕΡΩΤΗΣΗ: Πώς ήξεραν οι καταχωρητές ότι πρέπει να διαβάσουν ή να γράψουν στον δίαυλο κατά την ανάκληση; Η ανάκληση είναι κοινή για όλες τις εντολές

Ο IR έχει χωρητικότητα ίση με το OPCODE, στο παράδειγμα 4 bit. Όμως, υπάρχουν υλοποιήσεις στις οποίες έχει όσο μέγεθος έχει και ο MDR. Εδώ, ο MDR είναι 12 bit.

Η ανάκληση είναι κοινή, άρα οι καταχωρητές που εμπλέκονται σε αυτή ξέρουν πότε θα γράψουν στον δίαυλο ή πότε θα διαβάσουν από αυτόν.

Όταν ο IR στείλει το OPCODE στη μονάδα ελέγχου παράγονται τα σήματα για την εκτέλεση, η οποία διαφέρει από εντολή σε εντολή.

ΕΠΙΚΟΙΝΩΝΙΕΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΕΝΤΟΛΗΣ



ΑΝΑΚΛΗΣΗ

T0: MAR <-PC, Z<- PC+1, στο παράδειγμα, MAR=X2, ο Z= X2+1 (ΑΚΟΛΟΥΘΕΙ Η διαδικασία αποκωδικοποίησης μνήμης, ο MAR ζητάει μία διεύθυνση)

T1: MDR<-M[MAR], PC<-Z, στο παράδειγμα MDR= 0001 | 00001000, PC= X2+1 (PC έχει τη διεύθυνση της επόμενης εντολής που θα εκτελεστεί)

T2: IR<- MDR[OPCODE], IR<- 0001 (**μόνο τώρα ξέρει η CPU τι θα εκτελέσει)**

Είναι πολύ σημαντικό να καθορίσουμε τις επικοινωνίες που απαιτεί η κάθε εντολή.
ΒΗΜΑΤΑ: (1) Η τιμή 35 πρέπει να πάει στον MDR, (2) Η τιμή από τον MDR θα πάει στον ACC.

Για να περάσει η τιμή 35 στη CPU, πρέπει η διεύθυνση 8 να ζητηθεί μέσω του MAR. Που υπάρχει καταχωρημένη η διεύθυνση 8; Από την ανάκληση, ο MDR περιέχει τη ζητούμενη διεύθυνση (ΑΥΤΌ ΙΣΧΥΕΙ ΠΑΝΤΑ επειδή η ανάκληση είναι η ίδια!!!!!!!!).

T3: MAR<-MDR[Address], MAR <- 00001000 =8 (ακολουθεί αποκωδικοποίηση)

T4: MDR <- M[MAR] (η επικοινωνία 1 στο σχήμα, αλλά θυμίζουμε ότι για να γίνει η επικοινωνία 1 απαιτείται εμπλοκή του MAR για να ζητηθεί η διεύθυνση), MDR=35

T5: ACC<-MDR, ACC=35

Οι επικοινωνίες των περιόδων T4 και T5, εμπλέκουν τον εξωτερικό και τον εσωτερικό δίαυλο, αντίστοιχα. ΜΠΟΡΟΥΝ αυτά τα 2 βήματα να γίνουν μαζί; ΌΧΙ, διότι πρέπει πρώτα να γραφτεί η τιμή στον MDR και μετά να περάσει στον ACC.

Παράδειγμα: STA, διαβάζει τα περιεχόμενα του ACC και τα τοποθετεί στην διεύθυνση μνήμης του παράγοντα. Έστω ότι η STA έχει OPCODE 2, ο ACC= 19, και η διεύθυνση του παράγοντα είναι η 30 (σε αυτή τη διεύθυνση θα γραφτεί το περιεχόμενο του ACC). Έστω ότι η εντολή είναι αποθηκευμένη στη διεύθυνση μνήμης 11.

ΔΕΔΟΜΕΝΑ
1. Εντολή: 2 | 30 δηλαδή 0010 | 00011110
2. PC = 11 (εννοείται ότι την τιμή 11 την έλαβε κατά την ανάκληση της εντολής που υπήρχε στη διεύθυνση 10)

T0: MAR <-PC, Z<- PC+1 (MAR=11, Z=11+1=12)
T1: MDR<- M[MAR], MDR<-M[11], δηλαδή ο MDR = 0010 | 00011110
T2: IR<- MDR[OPCODE], IR=0010 (το OPCODE στέλνεται στη μονάδα ελέγχου)

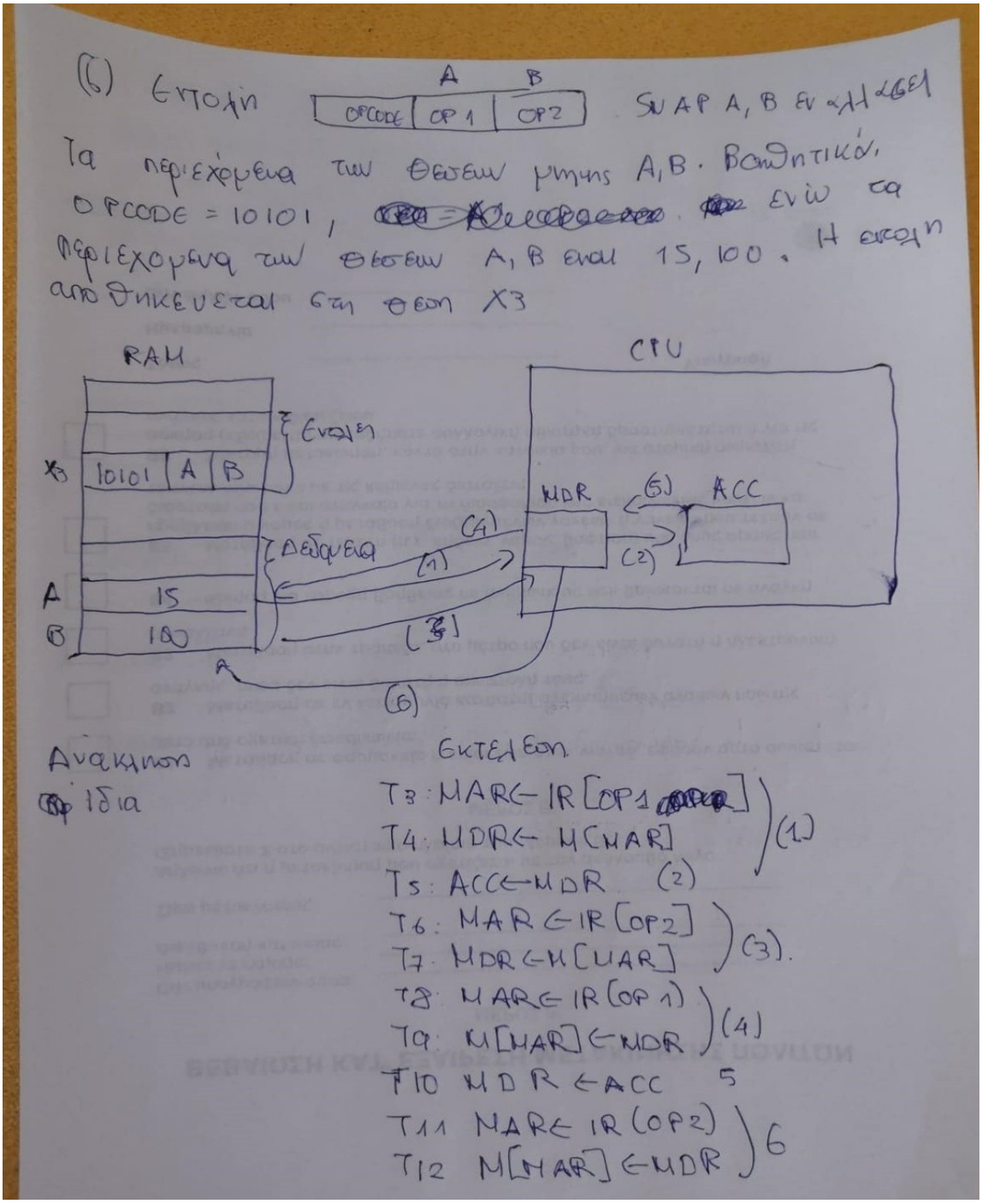
T3: MAR<- MDR[ADDRESS], δηλαδή MAR<-= 0010 | **00011110** (λαμβάνει το έντονο, 30). Ακολουθεί αποκωδικοποίηση
T4: MDR<-ACC (αυτό γίνεται επειδή μόνο ο MDR βλέπει τον δίαυλο δεδομένων), MDR=19
T5: M[MAR]<-MDR, M[30]= 19

ΠΑΡΑΔΕΙΓΜΑ: SWAP(A,B) δύο παραγόντων: OPCODE | A | B |

Ανάκληση:
T0: MAR <-PC, Z<- PC+1
T1: MDR<- M[MAR], MDR = OPCODE | A | B
T2: IR<- MDR[OPCODE]

Εκτέλεση.
Διαβάζουμε τα περιεχόμενα της θέσης Α (να τα φέρουμε στη CPU)
2) Επειδή το βήμα 1 απαιτεί εμπλοκή του MDR, αλλά ο MDR θα φέρει μετά και τη λέξη από τη θέση μνήμης Β, πρέπει η λέξη της θμ Α να αποθηκευτεί σε έναν γενικό καταχωρητή, έστω τον ACC
3) Φέρνουμε τη λέξη από την θέση μνήμης Β, θα παραμείνει στον MDR
4) Ζητάμε την Α, και τοποθετούμε τα περιεχόμενα του MDR (Β)
5) Ζητάμε την Β, και φέρνουμε στον MDR τα περιεχόμενα του ACC
6) τοποθετούμε τα περιεχόμενα του MDR (Α)

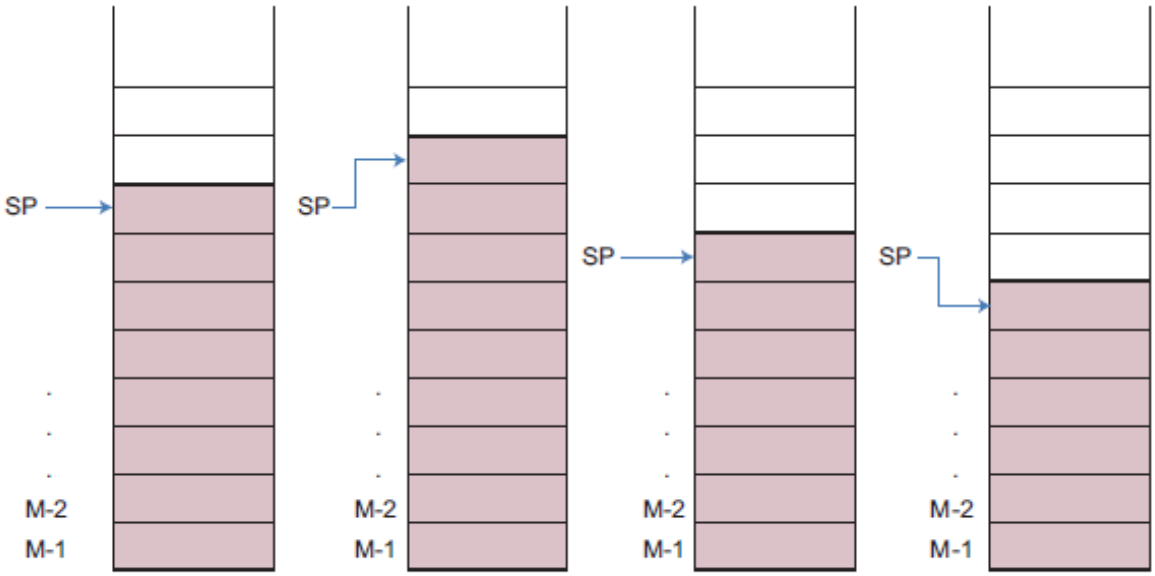
Στη λύση της διαφάνειας 10, ο IR θεωρείται ότι έχει ίδιο μήκος με τον MDR, άρα αποθηκεύει εκτός του OPCODE και τους παράγοντες μετά την ανάκληση και επειδή ο MDR θα φέρει τιμές από τη μνήμη, χρησιμοποιούμε τον IR για να δίνουμε τις ζητούμενες διευθύνσεις στον MAR.



Στη λύση της διαφάνειας 10, ο IR θεωρείται ότι έχει ίδιο μήκος με τον MDR, άρα αποθηκεύει εκτός του OPCODE και τους παράγοντες μετά την ανάκληση και επειδή ο MDR θα φέρει τιμές από τη μνήμη, χρησιμοποιούμε τον IR για να δίνουμε τις ζητούμενες διευθύνσεις στον MAR.

Εναλλακτικά, στην αρχή της εκτέλεσης θα έπρεπε να θέσουμε R<- MDR (Στο T3) και μετά κάθε ζήτηση διεύθυνση να γίνεται με την εντολή MAR<- R[ADDRESS] και όχι MAR<-IR(ADDRESS)

ΑΛΜΑ ΣΕ ΡΟΥΤΙΝΑ
ΣΤΟΙΒΑ



α) β)

SP (Stack Pointer): Δείχνει στην τελευταία θέση όπου υπάρχουν δεδομένα. Έστω ότι ο SP δείχνει στη θέση 7. Αν θελήσουμε να προσθέσουμε ένα στοιχείο στην στοίβα, τότε ο SP θα μειωθεί κατά 1 για να δείχνει τη θέση 6 και η τιμή του SP θα δοθεί στον MAR, ο οποίος θα ζητήσει τη θέση 6 και θα γίνει η εγγραφή. Ο SP δεν έχει πρόσβαση στον δίαυλο διευθύνσεων άρα όποια διεύθυνση ζητηθεί από τη μνήμη πρέπει να ζητηθεί μέσω του MAR.

Στο (β) αν υποτεθεί ότι ο SP δείχνει στη θέση 7 και χρησιμοποιηθεί το στοιχείο που βρίσκεται εκεί, τότε ο SP θα δείχνει στη θέση 8 (αυξάνεται κατά 1). Το επόμενο στοιχείο που θα διαβαστεί είναι εκείνο της θέσης 8, δηλαδή η θέση 7 διαγράφεται με ΛΟΓΙΚΟ τρόπο. Αν χρειαστεί εγγραφή στη στοίβα, τότε τα δεδομένα της θέσης 7 θα διαγραφούν με φυσικό τρόπο και θα αντικατασταθούν από τα νέα.

Χρήση: Αποθήκευση μεταβλητών που δημιουργούνται κατά την εκτέλεση του προγράμματος
Εκτέλεση εντολών 0 παραγόντων (;;;;;)

JSR: Jump to Subroutine

Θέση 4 είναι η τελευταία γεμάτη θέση της στοίβας, δηλαδή η θέση 3 είναι η πρώτη ελεύθερη
Στη θέση 100 δείχνει ο PC μετά από την ανάκληση της προηγούμενης εντολής (πριν από το άλμα).
Η εντολή της θέσης μνήμης 100 είναι μία εντολή άλματος σε ρουτίνα.
Η εντολή άλματος σε ρουτίνα έχει τη μορφή

OPCODE ADDRESS
xxxx 200 (στη θέση 200 βρίσκεται η διεύθυνση εκκίνησης της υπορουτίνας)

Η θέση μνήμης 100 περιέχει την εντολή xxxx 200

Αυτό σημαίνει τα εξής: Η εντολή JSR της γραμμής 100 θα εκτελεστεί και θα οδηγήσει το πρόγραμμα στην γραμμή 200. Όταν γίνει ανάκληση της JSR ο PC θα έχει τιμή 101 (η εντολή που υπάρχει στη γραμμή 101 είναι η εντολή που ακολουθεί την κλήση της υπορουτίνας).
Πως το πρόγραμμα θα κάνει άλμα στη γραμμή 200; Ο PC θα πρέπει να λάβει την τιμή 200 «ξαφνικά» εκεί που η τιμή του ήταν 101. Τι θα γίνει με την τιμή 101, η οποία ΑΠΑΓΟΡΕΥΕΤΑΙ να χαθεί διότι αν χαθεί δεν θα ξέρουμε σε ποια εντολή θα επιστρέψει το πρόγραμμα και αυτό είναι Fatal Error. Αυτό το 101 πρέπει να μεταφερθεί στη στοίβα και όταν το πρόγραμμα πρέπει να επιστρέψει να ξαναγραφτεί στον PC. Αυτό σημαίνει ότι μία εντολή JSR **συνοδεύεται από μία εντολή επιστροφής (RET).**

Ανάκληση
T0: MAR <- PC, Z<- PC +1 (MAR <- 100, Z<- 101)
T1: MDR<- M[MAR], PC<- Z (MDR = xxxx 200)
T2: IR <- MDR[OPCODE]; (IR = xxxx)

Εκτέλεση
1) Ο SP πρέπει να μειωθεί κατά 1 για να γράψουμε εκεί την τρέχουσα τιμή του PC, η οποία θα μας χρειαστεί κατά την επιστροφή.
2) Ο MAR πρέπει να διαβάσει την μειωμένη τιμή του SP (δηλαδή την τιμή 3) επειδή αυτός είναι που θα ζητήσει τη διεύθυνση 3.
3) Ο MDR σίγουρα θα μεταφέρει δεδομένα προς την μνήμη (θα μεταφέρει την τιμή του PC για να γραφτεί στη στοίβα, ΜΟΝΟ ο MDR μπορεί να γράψει δεδομένα στη μνήμη, άρα είναι σίγουρο ότι η τιμή 101 θα περάσει από τον MDR)
4) Όμως, ο MDR έχει αυτή τη στιγμή αποθηκευμένη την εντολή xxxx 200, η οποία πρέπει να αποθηκευτεί σε έναν καταχωρητή της CPU γιατί δεν θέλουμε να χάσουμε την τιμή της δ. μνήμης όπου ξεκινάει η υπορουτίνα. Ο IR ΔΕΝ μπορεί να χρησιμοποιηθεί επειδή θεωρούμε ότι έχει μήκος ίσο με τον OPCODE. Αν δεν είχε θα γράφαμε όλο τον MDR στον IR και θα λυνόταν το πρόβλημα. Θα δουλεύαμε με τον IR.

5) Πρέπει να δώσουμε στον PC την τιμή 200.

T3: Z (ADDRESS)<-SP-1 (εδώ μεσολαβεί η ALU, επειδή έχουμε αφαίρεση, ο εσωτερικός δίαυλος διαβάζει την τιμή του SP, αυτή μειώνεται και αποθηκεύεται στον Z)
T4: SP<-Z(ADDRESS), MAR<-Z[ADDRESS]. Ο Z χρησιμοποιεί τον δίαυλο, δίνει τα περιεχόμενα στον SP και στον MAR. Γράφουμε ADDRESS Επειδή ο Z μπορεί να είναι μεγαλύτερος σε μέγεθος από τους SP και MAR, οι οποίοι είναι σχεδιασμένοι για να κρατάν μόνο διευθύνσεις και όχι ολόκληρες λέξεις μνήμης.

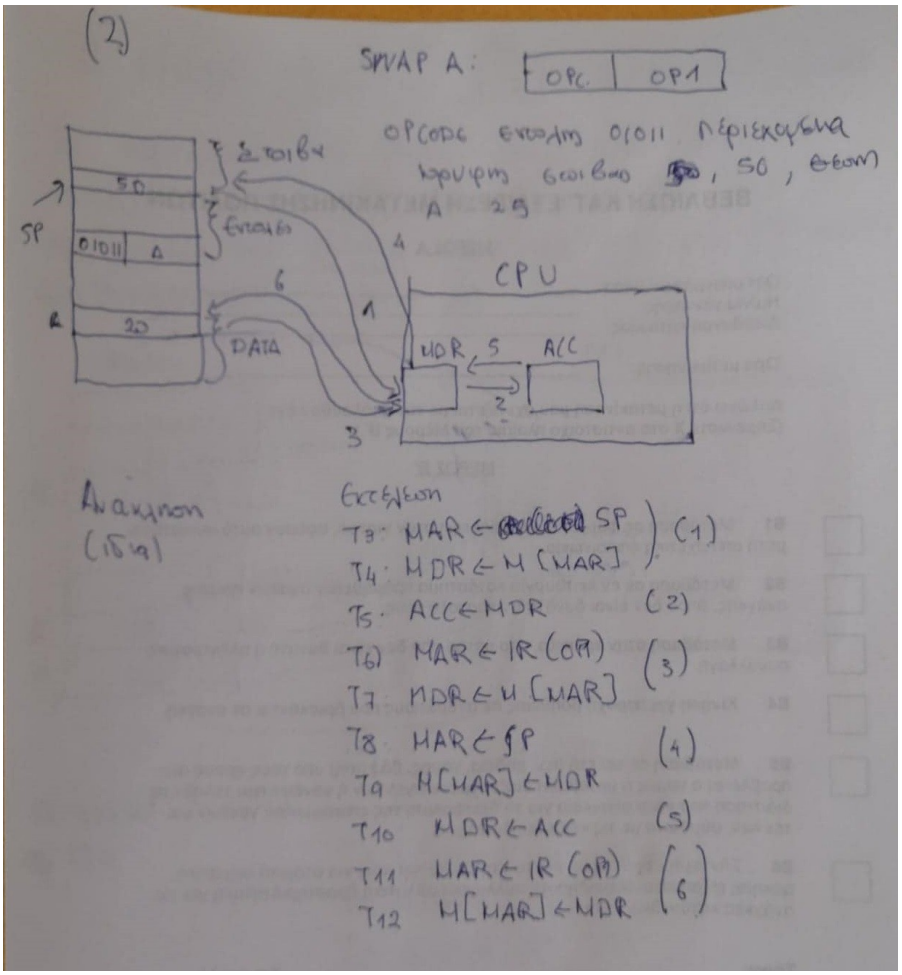
----------> ΠΕΡΑΣΜΑ του PC στην στοίβα.

T5: Z<- MDR (εδώ η εντολή xxxx 200 μεταφέρεται στον Z επειδή ο MDR θα λάβει την τιμή του PC για να την περάσει στη στοίβα)
T6: MDR <- PC
T7: M[MAR] <-MDR, PC<-Z(ADDRESS) (Η πρώτη από τις 2 πράξεις γίνεται με τον εξωτερικό δίαυλο δεδομένων άρα σε καμία περίπτωση δεν επηρεάζει τη δεύτερη πράξη, η οποία γίνεται από τον εσωτερικό δίαυλο. Η θέση μνήμης στην οποία δείχνει ο MAR από το βήμα T4 είναι η πρώτη ελεύθερη θέση της στοίβας. Ο PC λαμβάνει τη διεύθυνση 200 από τον Z, ο οποίος την είχε παραλάβει από τον MDR στο βήμα T5.
Από τη στιγμή που ο PC έχει τιμή 200, είναι φανερό ότι ο έλεγχος του προγράμματος θα περάσει εκεί.
Από εκεί και κάτω, ανακαλείται η εντολή της δ. 200, ο PC <- 201 εκτελείται η εντολή αυτή, έπειτα ανακαλείται η εντολή της δ. 201, ο PC <- 202 εκτελείται η εντολή αυτή, ……..
ανακαλείται η εντολή της δ. 205 (RETURN) και πρέπει να δούμε τι γίνεται τώρα (πρέπει να γυρίσει το πρόγραμμα στην εντολή της δ. μνήμης 101, η οποία υπάρχει στην κορυφή της στοίβας)

RETURN
Η τιμή του PC που έχει αποθηκευτεί στην κορυφή της στοίβας πρέπει να γραφτεί πίσω σε αυτόν
Για να γίνει η πρώτη ενέργεια, θα πρέπει να ζητηθεί ανάγνωση (μόνο μέσω του MAR) της λέξης μνήμης που βρίσκεται στην κορυφή της στοίβας.
Αυτό σημαίνει ότι θα έχουμε και μία λογική διαγραφή δηλαδή ο SP θα αυξηθεί κατά 1.
Πάλι, η τιμή αυτή (101) θα πρέπει να γραφτεί στον MDR με τη διαφορά ότι εδώ δεν έχουμε ανησυχία σχετικά με το αν θα χαθεί κάποια τιμή του MDR.

Ανάκληση της Return
T3: MAR <- SP, Z<-SP+1 (Στο βήμα αυτό, ο SP έχει στον έλεγχό του τον δίαυλο, οπότε γράφει εκεί και η αυξημένη κατά 1 τιμή του μεταφέρεται στο Z)
T4: MDR<- M[MAR], SP<-Z (στο βήμα αυτό ο MDR θα λάβει από την κορυφή της στοίβας την τιμή 101 και ο SP θα δείχνει στη θέση 4 (εκεί που έδειχνε και πριν)
T5: PC <- MDR (ADDRESS) (ο PC θα είναι ίσος ξανά με 101 και επομένως θα πρέπει να ανακληθεί η εντολή που βρίσκεται εκεί και στη συνέχεια ο PC <-102, κ.ο.κ.)

ΠΑΡΑΔΕΙΓΜΑ 1 (ΕΝΑΛΛΑΓΗ)



SWAP A: Εναλλάσσει τα περιεχόμενα της κορυφής της στοίβας με τα περιεχόμενα μίας θέσης μνήμης A

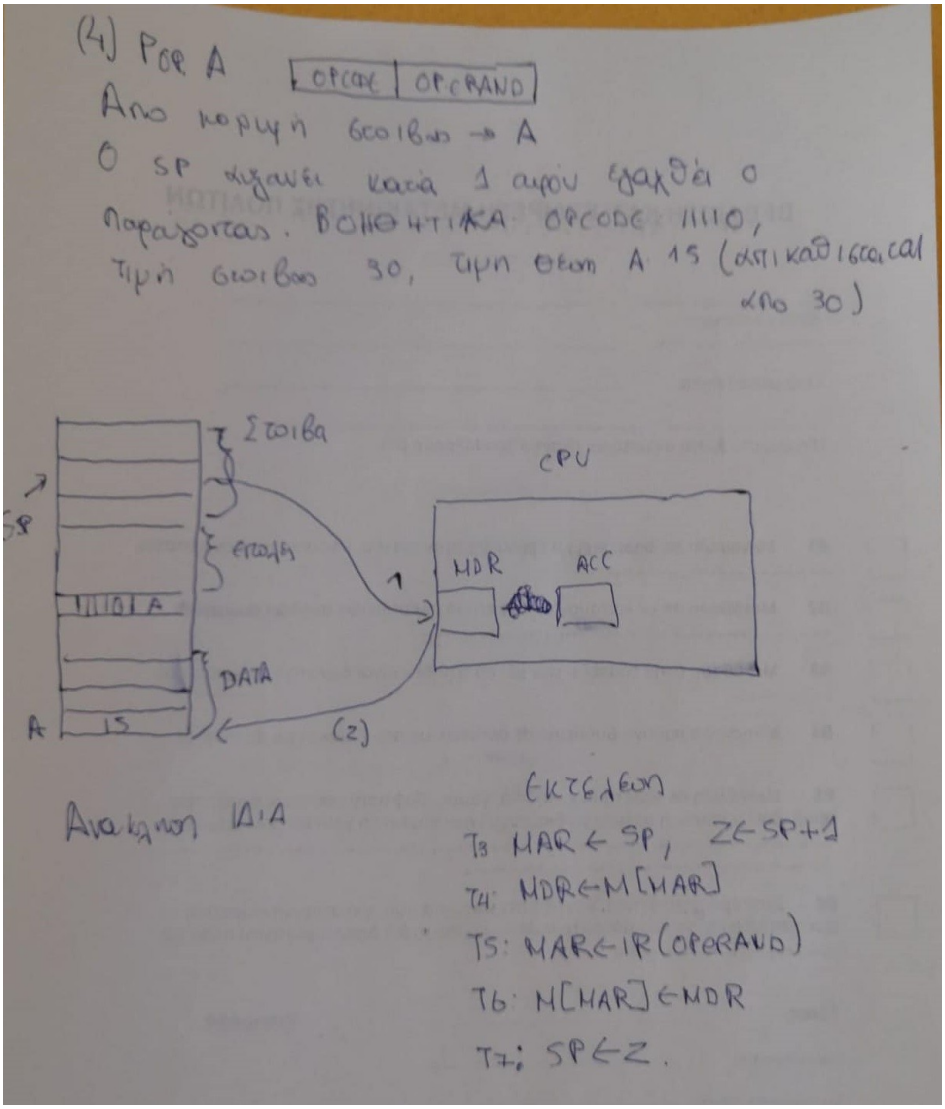
1) Θα φέρουμε από την κορυφή της στοίβας τα περιεχόμενα στην CPU, και συγκεκριμένα στον MDR. Επειδή όμως ο MDR θα χρησιμοποιηθεί και για την δεύτερη εργασία ανάγνωσης, δηλαδή για να περάσουν τα δεδομένα από τη θέση μνήμης A στον MDR, η οποία ακολουθεί, τα περιεχόμενα του MDR από την πρώτη ανάγνωση θα πάνε στον ACC.
2) Από τον MDR τα περιεχόμενα της κορυφής της στοίβας θα μεταφερθούν για προσωρινή αποθήκευση στον ACC
3) Θα φέρουμε από την θέση μνήμης A τα περιεχόμενα στην CPU, και συγκεκριμένα στον MDR. Επειδή ο MDR δεν χρειάζεται να γράψει κάτι άλλο από τη μνήμη θα στείλει απευθείας τα δεδομένα της θέσης μνήμης A στην κορυφή της στοίβας.
4) Αποστολή των δεδομένων στην κορυφή της στοίβας
5) Ο ACC δίνει τα δεδομένα που υπήρχαν αρχικά στην κορυφή της στοίβας στον MDR για να τα περάσει στη θέση μνήμης A
6) Αποστολή των δεδομένων στη θέση μνήμης A

Παρατήρηση: Η λύση υποθέτει αυτή τη φορά ότι ο IR έχει ίδιο μέγεθος με τον MDR, άρα ο IR θα χρησιμοποιείται για τις αναφορές στη θέση μνήμης A.

Ανάκληση
T0: MAR <- PC, Z <-PC+1
T1: MDR <- M[MAR], PC<-Z;
T2: IR <- MDR (Άρα ο IR θα έχει αποθηκευμένο το OPCODE και **τη διεύθυνση μνήμης A,** άρα εδώ δεν έχουμε να ανησυχούμε για το πως θα αντιγράψουμε τον MDR σε κάποιον βοηθητικό καταχωρητή.

Εκτέλεση
T3: MAR <- SP (Ο MAR διαβάζει την τιμή του SP, δηλαδή την κορυφή της στοίβας για να ζητήσει ανάγνωση της συγκεκριμένης διεύθυνσης από τη μνήμη……ακολουθεί η αποκωδικοποίηση μνήμης) **Δεν μεταβλήθηκε η τιμή του SP.**
T4: MDR <- M[MAR] εδώ ο MDR γράφει την τιμή 50.
T5: ACC <- MDR
(X←Y, Z←X, όχι μαζί T4 και T5)
T6: MAR <- IR(ADDRESS) **Προσοχή:** Από την ανάκληση, ο IR περιέχει την διεύθυνση Α. Ο IR είναι μεγαλύτερος του MAR.
T7: MDR <- M[MAR] εδώ ο MDR γράφει την τιμή 25.
(**ο MDR μπορεί την τιμή που έλαβε να την περάσει απευθείας στη στοίβα χωρίς να την αποθηκεύσει σε άλλο καταχωρητή)**
T8: MAR<-SP (για λειτουργίες με τη στοίβα, ο MAR ενημερώνεται από τον SP)
T9: M[MAR] <- MDR
T10: MDR<- ACC
T11: MAR<-IR(OPCODE) (για λειτουργίες με τη υπόλοιπη μνήμη, ο MAR ενημερώνεται από τον IR)
T12: M[MAR] ← MDR

ΠΑΡΑΔΕΙΓΜΑ 2



POP A: Μεταφέρει τα δεδομένα από την κορυφή της στοίβας στη θέση μνήμης A
PUSH A: Μεταφέρονται τα δεδομένα από την θέση μνήμης A στην κορυφή της στοίβας

Μορφή της εντολής OPCODE Address

1) Ζητούμε τη θέση της κορυφής της στοίβας από τον SP και να τη δώσουμε στον MAR προκειμένου να ζητηθεί η λέξη για ανάγνωση
2) Αύξηση του SP κατά 1
3) Πρέπει ο MAR να ενημερωθεί με τη διεύθυνση μνήμης A (ΑΠΌ ΠΟΥ; Η λύση υποθέτει ότι ο IR έχει ίδιο μέγεθος με το μέγεθος της λέξης μνήμης ή με τον MDR) για να τη ζητήσει για εγγραφή
4) Εγγραφή στη θέση μνήμης A

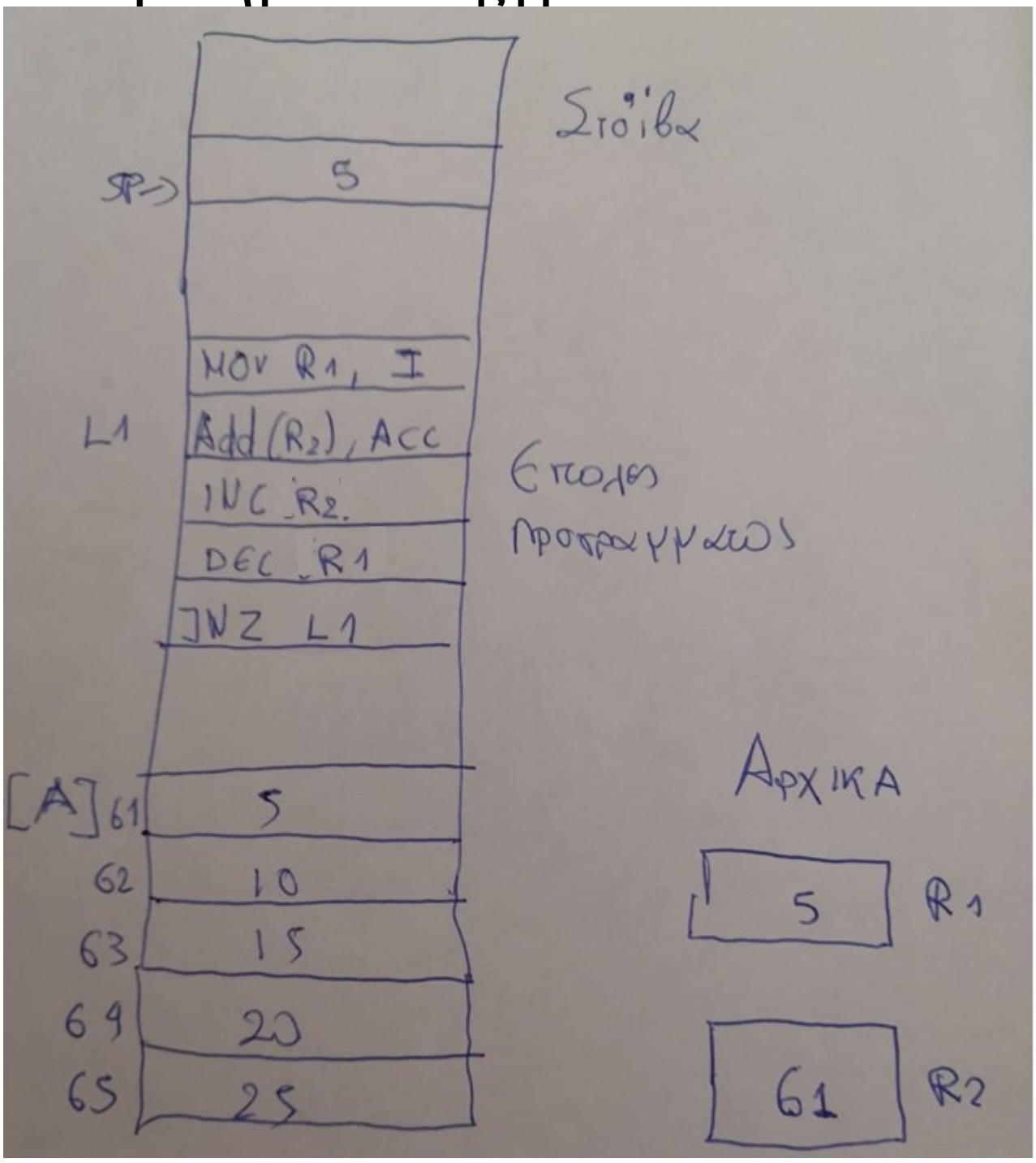
Ανάκληση
T0: MAR <- PC Z<-PC+1
T1: MDR<- M[MAR], PC<-Z
T2: IR<-MDR (ο IR πλέον περιέχει εκτός του OPCODE και τη διεύθυνση A στην οποία θα γραφτεί η κορυφή της στοίβας)

Εκτέλεση

T3: MAR <- SP, Z<- SP+1 (Αυξάνουμε τον SP γιατί η τιμή που θα διαβαστεί δεν θα ξαναχρησιμοποιηθεί δηλαδή θα διαγραφεί λογικά)
T4: MDR <-M[MAR]
T5: MAR<-IR(Address) αν ο IR αποθήκευε μόνο το OPCODE Στο βήμα 3 θα έπρεπε να γράφαμε ACC<-MDR ή γενικά R<-MDR και να ακολουθούσαν τα επόμενα βήματα. Στο τρέχον βήμα, θα γράφαμε MAR<- R[Address] ή MAR<- ACC[Address]. Αυτό το βήμα θα ήταν το T6
T6: M[MAR] <-MDR
T7: SP<-Z (αυτό θα μπορούσε να γίνει και μετά το βήμα 3.

PUSH A (προηγήθηκε ανάκληση)
T3: MAR < - IR(address) (ο MAR διαβάζει τη διεύθυνση A για να την ζητήσει για ανάγνωση από τη μνήμη)
T4: MDR<- m[mar] (Ανάγνωση των περιεχομένων της θέσης μνήμης A)
T5: Z<-SP-1 (αν θεωρήσουμε ότι ο SP δείχνει στην τελευταία ή κορυφαία τιμή της στοίβας)
T6: MAR<-Z . SP<-Z (ο Z χρησιμοποιεί τον δίαυλο)
T7: M[MAR] <- MDR

Ένα παράδειγμα εκτέλεσης βρόχου



MOV R1, 5 -- Στον καταχωρητή R1 μεταφέρεται η 5, η οποία είναι το πλήθος επαναλήψεων του βρόχου

L1: -- ετικέτα η οποία χρησιμοποιείται για να μεταφέρουμε το πρόγραμμα σε μία συγκεκριμένη θέση

<LOOP-BODY> Εντολές, στο παράδειγμα μία πρόσθεση των στοιχείων του πίνακα A

DEC R1 -- Μείωση του πλήθους των επαναλήψεων του βρόχου (άρα R1 = R1-1)

JNZ R1, L1 -- Συγκρίνει το R1 με το 0, αν δεν είναι 0 πήγαινε στη ετικέτα L1

1η Εντολή: Move R1, I: η εντολή αυτή τοποθετεί σε έναν βοηθητικό καταχωρητή R1 την τιμή I, η οποία είναι το πλήθος επαναλήψεων του βρόχου. Αυτό το πλήθος έχει γραφτεί στην κορυφή της στοίβας.

ΜΟΡΦΗ ΕΝΤΟΛΗΣ: Εντολή μηδέν παραγόντων μόνο OPCODE (δεν υπάρχει παράγοντας, η κορυφή της στοίβας ελέγχεται από τον SP)

ΑΝΑΚΛΗΣΗ:
Εκτέλεση:
T3: MAR < - SP
T4: MDR < - M[MAR], MDR=5
T5: R1 <- MDR, R1 = 5

2η εντολή: Add(R2), ACC: Προσθέτει στα περιεχόμενα του ACC τα περιεχόμενα της διεύθυνσης μνήμης, η οποία καταχωρείται στον R2. Το αποτέλεσμα της πρόσθεσης καταχωρείται στον ACC. Έμμεση διευθυνσιοδότηση καταχωρητή (indirect register addressing): Η τιμή που ζητούμε βρίσκονται στη θέση μνήμης που αποθηκεύεται στον καταχωρητή R2.

Μορφή της εντολής: OPCODE, #R (OPCODE, διεύθυνση καταχωρητή). Διεύθυνση ενός παράγοντα, αλλά ο παράγοντας είναι διεύθυνση καταχωρητή.

Ο πίνακας Α αρχίζει στη διεύθυνση 61 =R2.
Διαβάζει την τιμή 61 από τον R2 για να φέρει την τιμή της διεύθυνσης 62 στην CPU

ΑΝΑΚΛΗΣΗ

Εκτέλεση
T3: MAR<-R2
T4: MDR <-M[MAR]. MDR<-M[61]= 5
T5: Z = MDR +ACC (Z=5+0=5)
T6: ACC =Z (ACC=5)

ΣΧΟΛΙΟ: Έμμεση διευθυνσιοδότηση καταχωρητή: δηλώνεται με ( )

3η εντολή: INC R2, αύξησε κατά 1 το περιεχόμενο του R2
Ενός παράγοντα: OPCODE, Διεύθυνση καταχωρητή

T3: Z<- R2+1 , Z= 62
T4: R2<-Z, R2 =62

4η εντολή: DEC R1, μείωσε κατά 1 το περιεχόμενο του R1
Ενός παράγοντα: OPCODE, Διεύθυνση καταχωρητή

T3: Z<- R1-1 , Z= 4
T4: R1<-Z, R1 =4

5η εντολή: JUMP NO ZERO, R1, L1: Εξετάζει αν η τιμή του R1 είναι ίση με 0 και αν όχι επιστρέφει το πρόγραμμα στην τιμή της ετικέτας.

Ενός παράγοντα, OPCODE, διεύθυνση του R1

If R1 >0 then PC<- L1, PC=L1

R1>0 στην ΑΛΜ υπάρχει κύκλωμα συγκριτή μεγέθους ο οποίος ελέγχει την ανισότητα 2 τιμών

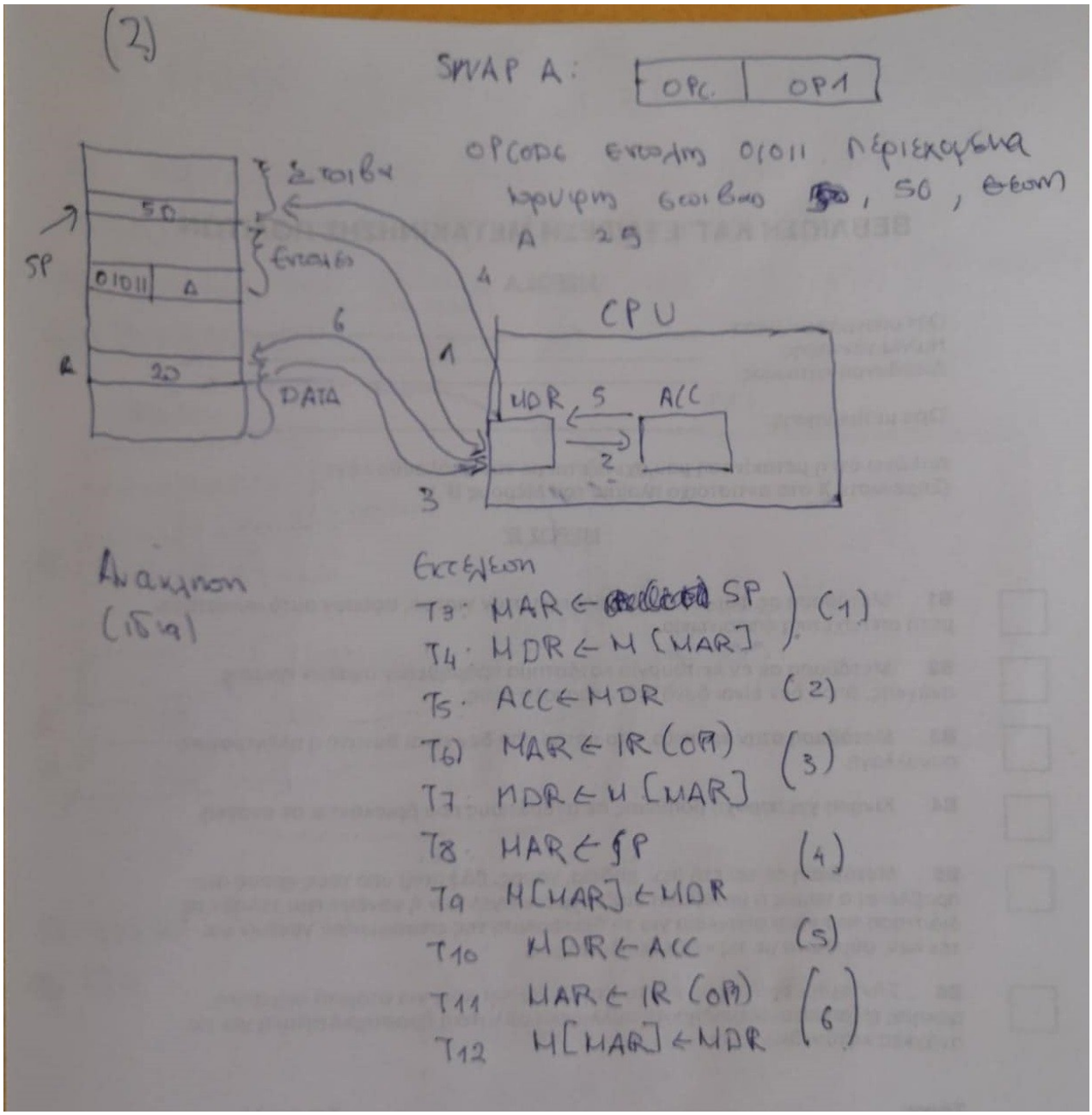
Παράδειγμα εκτέλεσης:

1η επανάληψη: R1=5
ACC = 5
R2 = 62
R1 = 4
PC = L1

2η επανάληψη:
ACC = 5+10=15
R2 = 63
R1 = 3
PC = L1

2η επανάληψη:
ACC = 5+10+15=30
R2 = 64
R1 = 2
PC = L1
…….

ΠΑΡΑΔΕΙΓΜΑ 1



ΥΛΟΠΟΙΗΣΗ 2 ΕΣΩΤΕΡΙΚΩΝ ΔΙΑΥΛΩΝ

T0: MAR<- PC,
T1: MDR<-M[MAR], PC<-PC+1
T2: IR<- MDR;

Η αύξηση του PC γίνεται σε έναν χρόνο χωρίς τη συμμετοχή του Z. Η ALU δέχεται ως είσοδο την τιμή του PC την αυξάνει κατά 1 και του την γυρίζει πίσω μέσω του 2ου εσωτερικού διαύλου. Ως προς την ανάκληση, το πλήθος των βημάτων παραμένει 3 δεν έχουμε ουσιαστικό κέρδος απλά η αύξηση γίνεται σε 1 χρόνο (δεν μεσολαβεί το Z)

ΔΕΝ μπορούμε να κάνουμε ταυτόχρονα αναθέσεις της εξής μορφής

X<-Z

Y<-X δηλαδή ένας καταχωρητής σε μία χρονική στιγμή να βρίσκεται και δεξιά και αριστερά του συμβόλου ανάθεσης τιμής <-. Π.χ., το X δεν μπορεί ταυτόχρονα να λαμβάνει από το Z και να μεταφέρει στο Y γιατί δεν υπάρχει καμία εγγύηση σχετικά με το ποια τιμή του X θα πάει στο Y (το παλιό X ή το Z). Οι χρονικοί παλμοί T είναι μία συγκεκριμένη χρονική περίοδος.
Επιπλέον, η μονάδα ελέγχου αν είχαμε αυτές τις αναθέσεις

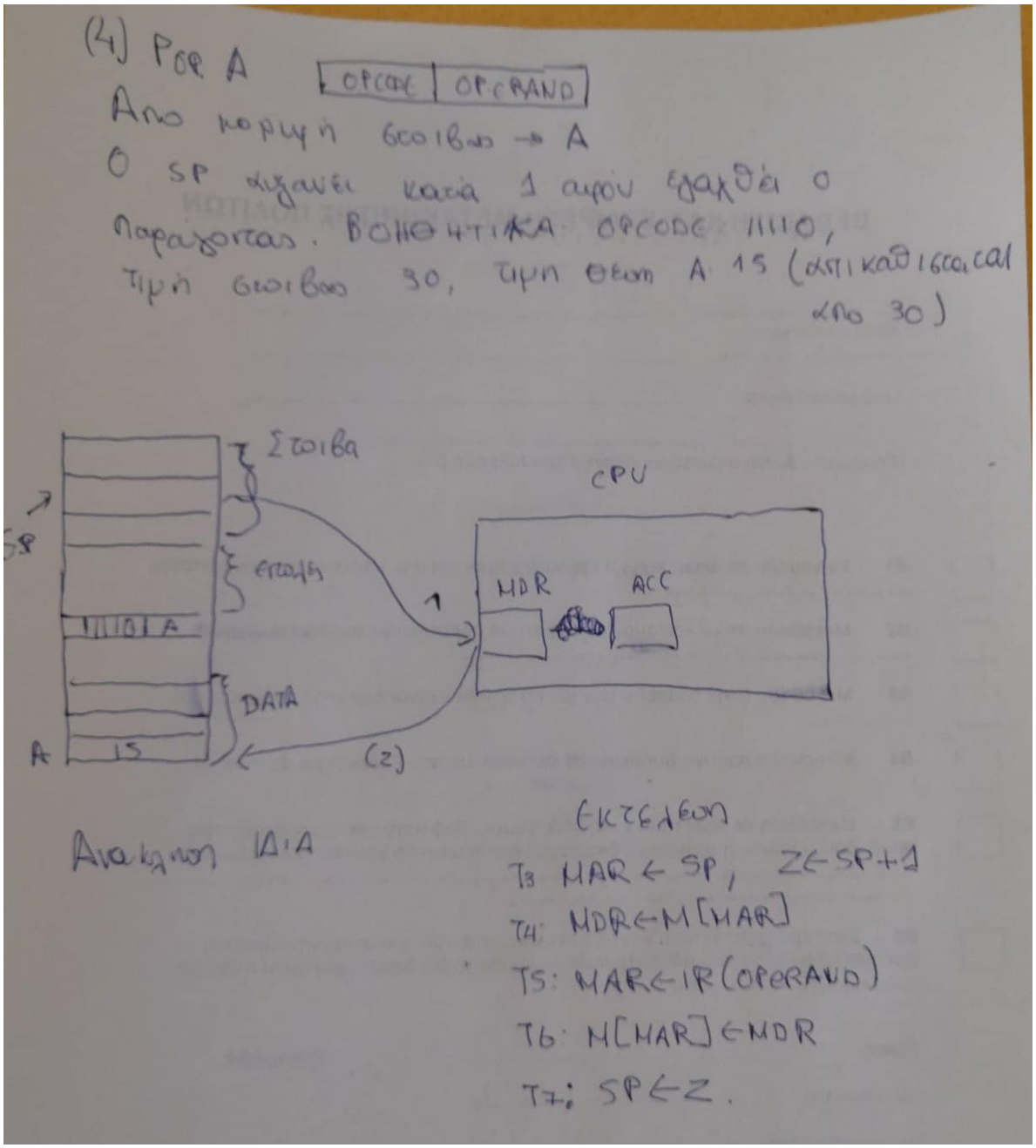
T3: X<-Z, Y<-X **(ΌΧΙ)**

Θα έστελνε στους καταχωρητές X, Y ένα σήμα ανάγνωσης στον χρόνο T3 και στους X, Z ένα σήμα εγγραφής στους διαύλους σε χρόνο T3.

T3: MAR<-SP
T4: MDR<- M[MAR]
T5: ACC<-MDR **(όχι μαζί με το T4 διότι δεν υπάρχει καμία εγγύηση σχετικά με το ποια τιμή θα λάβει ο ACC)**, MAR <- IR(address) (ο MDR χρησιμοποιεί τον πρώτο δίαυλο για να γράψει και ο ACC για να διαβάσει, ενώ ο IR χρησιμοποιεί τον δεύτερο δίαυλο για να γράψει και ο MAR για να διαβάσει)
T6: MDR<-M[MAR]
T7: MAR <- SP
T8: M[MAR]<- MDR
T9: MDR<-ACC, MAR<-IR(Address)
T10: M[MAR]<-MDR

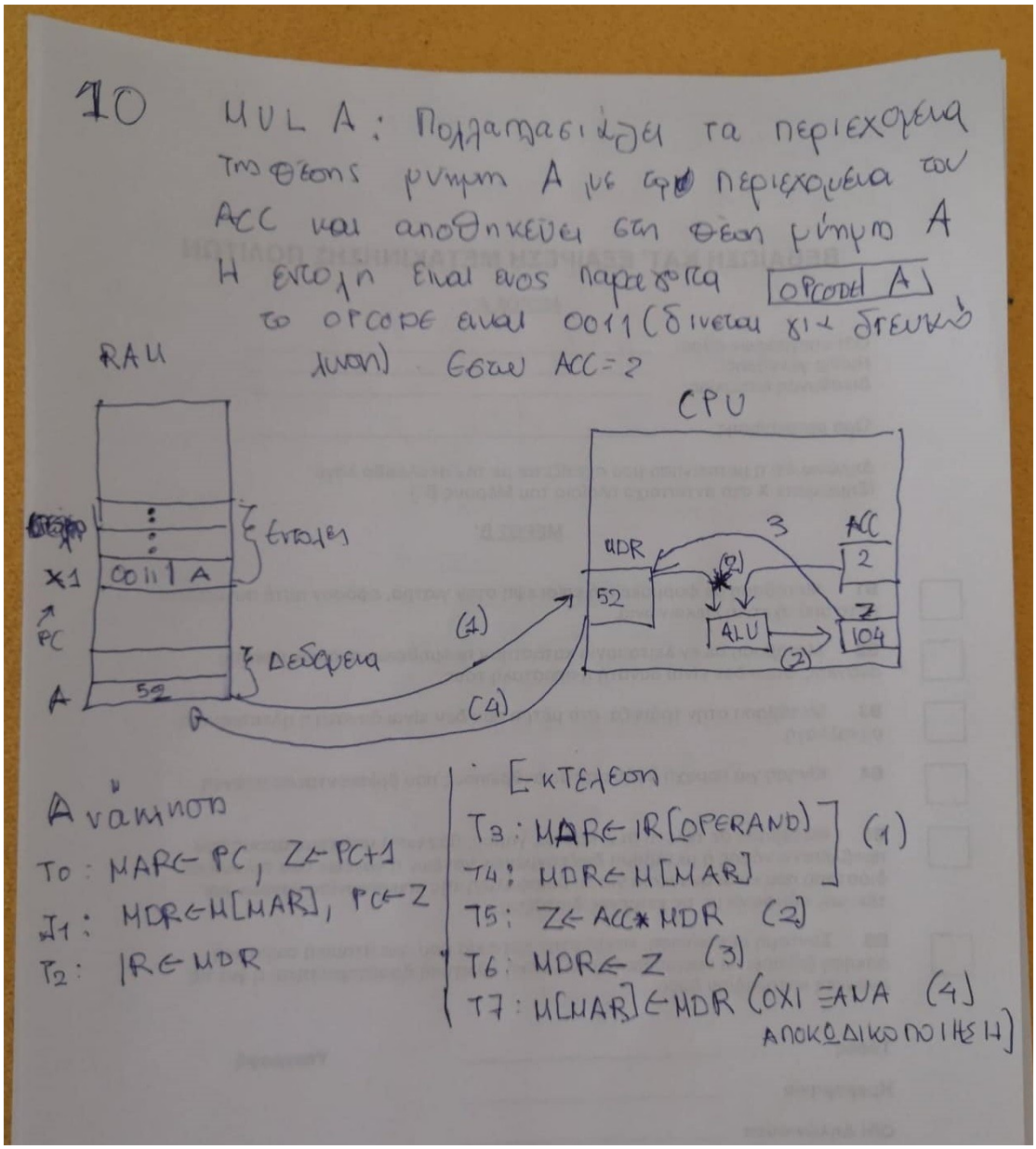
Κανόνας: Υλοποιούμε την εντολή με έναν εσωτερικό δίαυλο. Τοποθετούμε στον ίδιο χρόνο μικρολειτουργίες στις οποίες συμμετέχουν 4 διαφορετικοί καταχωρητές, χωρίς να παραβιάσουμε την σειρά εκτέλεσης των μικρολειτουργιών βάσει των αναγκών της εντολής. Τέλος, αλλάζουμε την ανάκληση.

ΠΑΡΑΔΕΙΓΜΑ 2



ΑΝΑΚΛΗΣΗ: Ίδια όπως στο παράδειγμα 1, δηλαδή η αύξηση του PC γίνεται σε έναν χρόνο
T3: MAR<-SP
T4: SP<-SP+1, MDR<-M[MAR]
T5: MAR<-IR(Address)
T6: M[MAR] <-MDR

ΠΑΡΑΔΕΙΓΜΑ 4



ΑΝΑΚΛΗΣΗ όπως προηγουμένως.

T3: MAR<-IR(Address)
T4: MDR<- M[MAR]
T5: MDR<- ACC * MDR
T6: M[MAR] <-MDR

IF (IEN=0) THEN F←0 ELSE G<-1

Μονάδα ελέγχου είναι το κύκλωμα το οποίο παράγει τα σήματα τα οποία δίνουν εντολή σε έναν καταχωρητή να διαβάσει από τον δίαυλο/ δίαυλους ή να γράψει στον δίαυλο σε μία συγκεκριμένη χρονική στιγμή, T_i

Τα σήματα Ti, παράγονται από ένα κύκλωμα της ΜΕ με τη βοήθεια δύο επιμέρους κυκλωμάτων: (1) Μετρητής: ένα κύκλωμα το οποίο σε κάθε θετικό παλμό του ρολογιού αυξάνεται κατά 1, δηλαδή η τιμή που αποθηκεύει αυξάνεται κάθε φορά κατά 1. Δηλαδή αν αποθηκεύει την τιμή 3, στον επόμενο θετικό παλμό του ρολογιού θα αποθηκεύει το 4, (2) ένας αποκωδικοποιητής (Αυτό το κύκλωμα θα το εξετάσουμε στο επόμενο μάθημα).

Έστω ότι το σύστημά μας διαθέτει 16 εντολές δηλαδή το OPCODE είναι 4 bits.

Το OPCODE αρχικά κατά την ανάκληση κάθε εντολής διαβάζεται από τον IR. Αυτός στέλνει το OPCODE στη ΜΕ, η οποία βάσει του OPCODE δίνει τις «οδηγίες» στους καταχωρητές. Κάθε καταχωρητής ξέρει ανάλογα με την εντολή πότε πρέπει να γράψει ή να διαβάσει.

Υπάρχουν 2 τρόποι υλοποίησης της ΜΕ: Ο πρώτος λέγεται προκατασκευασμένος και βασίζεται σε πύλες. Ο δεύτερος λέγεται μικροπρογραμματιζόμενος και βασίζεται σε μνήμη.

F: το F είναι ένα flip-flop ένα στοιχείο μνήμης, το οποίο αποθηκεύει τιμή ίση με 0 όταν βρισκόμαστε στην ανάκληση μίας εντολής ενώ γίνεται 1 όταν βρισκόμαστε στην εκτέλεση.

G: flip-flop το οποίο σχετίζεται με τον κύκλο διακοπής. Όταν η CPU δεχτεί ένα σήμα διακοπής από το ΛΣ τότε αυτό το flip-flop γίνεται 1 όταν ξεκινήσει ο κύκλος διακοπής διαφορετικά είναι 0. ΠΑΡΑΔΟΧΗ: Θεωρούμε ότι ο έλεγχος διακοπής γίνεται όταν ολοκληρωθεί η εκτέλεση μίας εντολής. Στο τέλος κάθε εντολής γίνεται αυτός ο έλεγχος: IF (IEN=0) THEN F←0 ELSE G<-1

IEN: Όταν το ΛΣ στείλει ένα σήμα διακοπής τότε αυτό το flip – flop γίνεται 1.

ΠΩΣ θα πούμε στον καταχωρητή MAR ΠΟΤΕ θα πρέπει να διαβάσει;

Ο κάθε καταχωρητής διαθέτει 2 βασικά σήματα: Ένα σήμα In το οποίο όταν είναι 1, ο καταχωρητής πρέπει να διαβάσει από τον εσωτερικό δίαυλο. Όταν είναι 0, δεν διαβάζει. Αντίστοιχα, ένα σήμα out το οποίο, όταν είναι 1 ο καταχωρητής πρέπει να γράψει στον εσωτερικό δίαυλο, όταν είναι 0, δεν γράφει.

Στη συγκεκριμένη ερώτηση, μας ενδιαφέρει να εξετάσουμε ΠΟΤΕ το σήμα MARin είναι ίσο με 1. Το πότε, προδιαθέτει για ένα πλήθος περιπτώσεων.

Δηλαδή θα πούμε ότι MARin =1 όταν (ΣΥΝΘΗΚΗ 1) Ή (ΣΥΝΘΗΚΗ 2) Ή ……. (ΣΥΝΘΗΚΗ Ν). Ποιες είναι αυτές οι συνθήκες;

Αυτό καθορίζεται από τις προ-σχεδιασμένες μας εντολές. Ο κάθε καταχωρητής διαβάζει όταν βρίσκεται αριστερά του συμβόλου <-

Οι πύλες του πρώτου επιπέδου (AND, OR, OR, AND) 1-4

Οι πύλες της δεύτερης στήλης (AND, AND) 5-6

Η τελική OR = 7

1) **Ανάκληση:** Στο βήμα Τ0 της ανάκλησης. Για να γνωρίζουμε ότι έχουμε ανάκληση πρέπει το F =0. Άρα (Σχήμα σελ.511) η πύλη 1 θα δώσει έξοδο 1 όταν το F=0 (ανάκληση) και το χρονικό σήμα (χρονική περίοδος) είναι το Τ0 (Κάθε φορά, ένα Τ είναι ίσο με 1). Αν ισχύει ότι Τ0=1 και F=0, επειδή η έξοδος της πύλης 1 πηγαίνει ως είσοδος στην OR με αριθμό 7, θα είναι MARin =1 δηλαδή ο MAR εκείνη τη χρονική στιγμή διατάσσεται να διαβάσει από τον δίαυλο. Επειδή ανάκληση έχουμε όταν f=0 στο σχήμα βάζουμε F’=1(ισοδύναμο του F=0).
2) **Ζήτηση διακοπής:** Για να ζητηθεί διακοπή πρέπει να βρισκόμαστε στην εκτέλεση (θεωρούμε ότι διακοπή γίνεται μόνο στο τέλος της εκτέλεσης κάθε εντολής, άρα πριν την ανάκληση, συνεπώς F=1). Επειδή θα εκτελεστεί ο κύκλος διακοπής, G=1.

   Για να διαβάσει ο MAR κατά την διαδικασία ζήτησης διακοπής, πρέπει να είναι

   (F AND G=1) AND (T1=1 OR T4=1) -> Υπο-κύκλωμα των πυλών 2 και 5.

   Αν βρισκόμαστε στη χρονική στιγμή Τ1 ή Τ4 ΚΑΙ τα F=G=1, τότε το σήμα που προκύπτει στην έξοδο της πύλης AND με αριθμό 5 είναι 1 και τοποθετείται ως είσοδος στην πύλη OR με αριθμό 7, θέτοντας τελικά MARin=1. Άρα ο MAR πρέπει να διαβάσει.

3) **Εκτέλεση:** Η εκτέλεση καταρχήν σημαίνει ότι F=1 ΚΑΙ G=0 (όταν ζητηθεί διακοπή G=1 και F=1). Δηλαδή ΔΕΝ διακόπηκε η CPU και περάσαμε την ανάκληση. FG’=1. Δηλαδή, για τις εντολές, LDA, STA, ADD, AND, IN, OUT και RET πρέπει να διαβάσει στο Τ3. Δηλαδή, όταν βρισκόμαστε στο βήμα Τ3 ΚΑΙ FG’=1 ΚΑΙ το OPCODE που θα έρθει από τον IR αντιστοιχεί σε μία από τις παραπάνω εντολές, MARin=1.
   ΠΑΡΑΤΗΡΗΣΗ: Ομαδοποιούμε όλες τις εντολές (φάση εκτέλεσης) για τις οποίες ένας καταχωρητής διαβάζει σε συγκεκριμένη χρονική στιγμή.

   Ο MAR διαβάζει και όταν Τ4=1 ΚΑΙ F=1 ΚΑΙ G=0 ΚΑΙ το opcode είναι εκείνο της JSR (πύλη AND με αριθμό 4).

   1) Πως παράγονται τα σήματα χρονισμού;

2) Πως από τα opcode παράγονται τα σήματα LDA, STA, ADD. Το OPCODE δεν είναι δίτιμο αλλά τα σήματα LDA, STA, ADD κλπ είναι.

Πύλη 1: Ανάκληση **(F'=1**)

Πύλες 2 και 5: Διακοπή **(FG=1)**

Πύλες 3,4. Και 6: Εκτέλεση **(FG'=1)**

Εντολή AND βήμα T4: MDR <- M[MAR].

ΠΑΡΑΤΗΡΗΣΗ (θα συνδεθεί και με τον μικροπρογραμματιζόμενο έλεγχο): Γενικά, στον προκατασκευασμένο έλεγχο δεν έχει νόημα (είναι και επιζήμιο γιατί προσθέτουμε υλικό) να θέσουμε το MDRin=1. Δεν μιλάμε για επικοινωνία με εσωτερικό δίαυλο αλλά με τη μνήμη. Άρα αν ζητηθεί το MDRin ΔΕΝ θα ληφθούν υπόψη οι αναγνώσεις από τη μνήμη.

ACCout:

1) **Ανάκληση:** Δεν υπάρχει συμμετοχή του ACC.
2) **Διακοπή:** Δεν υπάρχει συμμετοχή.

Αν δει κανείς τις εντολές, ο ACC βρίσκεται στο δεξί μέρος του συμβόλου <- σε πάρα πολλές εντολές, αλλά η υλοποίηση περιέχει ΜΟΝΟ μία πύλη AND με εισόδους t4, F, G' STA. ΠΟΙΟ ΕΙΝΑΙ ΤΟ ΛΑΘΟΣ;

ΑΠΑΝΤΗΣΗ: Δεν υπάρχει κανένα λάθος, σε όλες τις εντολές πλην της STA, ο ACC συμμετέχει ως είσοδος σε αριθμητικές ή λογικές πράξεις. Κατά συνέπεια, τροφοδοτεί απευθείας την CPU και όχι τον εσωτερικό δίαυλο

MDRin: Ανάκληση, δεν υπάρχει κομμάτι υλικού για την παραγωγή του σήματος.

Διακοπή: FGT2. Μία AND τριών εισόδων F, G, T2 της οποίας η έξοδος μπαίνει ως είσοδος στην τελική OR. [ΕΞΟΔΟΣ 0]

STA, T4

IN, T4

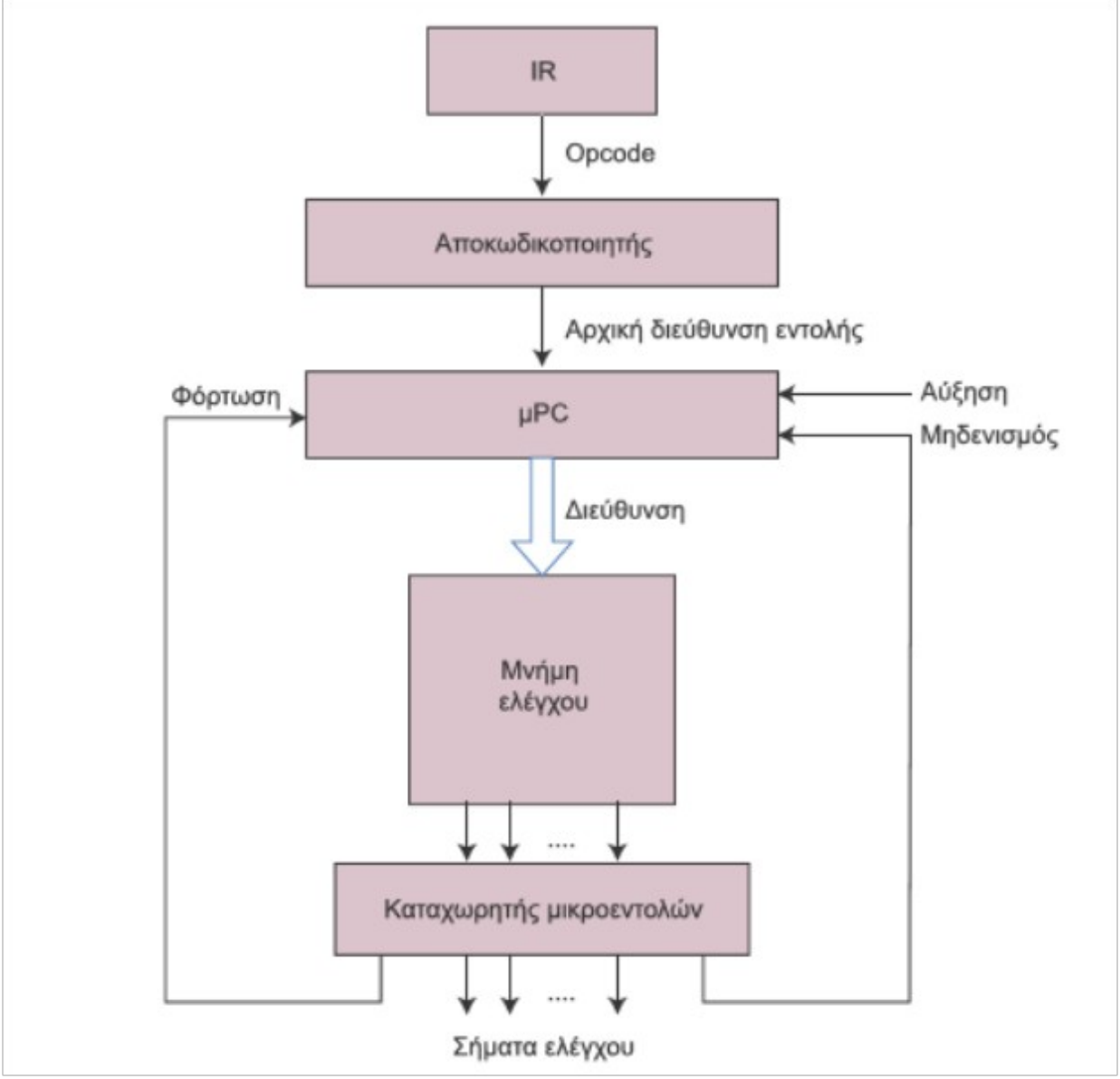
JSR, T6

Μία OR για τα σήματα STA και IN η έξοδος της οποίας θα έμπαινε είσοδος σε μία AND τριών εισόδων T4, F, G’ [ΕΞΟΔΟΣ 1]

Μία AND τεσσάρων εισόδων JSR, T6, F, G’. [ΕΞΟΔΟΣ 2]

Τα σήματα εξόδου αυτών των πυλών θα πήγαιναν σε μία τελική OR. Δηλαδή η τελική OR θα δεχόταν ως εισόδους τις ΕΞΟΔΟΥΣ 0 - 2

ΜΙΚΡΟΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΟΣ ΕΛΕΓΧΟΣ



Έστω ότι ένα πρόγραμμα πρέπει να εκτελέσει με τη σειρά τις εντολές STA, NOP, LDA και μετά ακολουθεί διακοπή.

Υποθέτουμε ότι το σύστημά μας περιέχει τις εντολές NOP, LDA, STA, τον κύκλο διακοπής και τον κύκλο ανάκλησης.

Η στήλη 0 (η διεύθυνση 0, όπου κάθε διεύθυνση περιέχει ένα πλήθος bit το οποίο είναι τόσο όσα και τα σήματα που χρησιμοποιούνται για όλες τις εντολές, εδώ έχουμε 15 σήματα άρα 15 bit) περιέχει όλα τα σήματα που συμμετέχουν στο πρώτο βήμα, το T0, της ανάκλησης..

T0: MAR <- PC, Z<PC +1;

ΑΝΑΛΥΣΗ: MARIn=1, Pcout=1, Zin=1, S2,S1,S0, C0 = 1001

Άρα, οι 5 μονάδες αποθηκεύονται στη διεύθυνση 0 και αντιστοιχούν στους καταχωρητές που πρέπει να κάνουν λειτουργίες Ανάγνωσης ή Εγγραφής κατά το βήμα T0 της ανάκλησης και σε σήματα που δείχνουν την επιθυμητή λειτουργία της ALU.

T1: MDR<-M[MAR], PC<-Z

ΠΡΟΣΟΧΗ: Στον μικροπρογραμματιζόμενο έλεγχο το bit που αντιστοιχεί στο MDRin γίνεται 1 και αντίστοιχα το σήμα MB/IB' γίνεται 1. Αυτό δεν γινόταν στον προκατασκευασμένο έλεγχο, στον οποίο δεν χρειάζεται να προστεθούν πύλες (επιπλέον) όταν γίνεται ανάγνωση από τη μνήμη. Εδώ, το μέγεθος της μνήμης είναι σταθερό και δεν επηρεάζεται από το αν MDRin = 0 ή 1.

Επιπλέον Pcin=1, Zout=1, R/W=1 (ανάγωση από τη μνήμη)

T2: IR<- MDR[OPCODE]. IRin (δεν φαίνεται σε αυτό το πινακάκι)=1, MDRout=1.

ΠΡΟΒΛΗΜΑ

Υποθέτουμε ότι ένα πρόγραμμα που ξεκινάει να εκτελέσει εντολές, στέλνει τη μονάδα ελέγχουν να διαβάσει τη στήλη 0, μετά την 1, μετά τη 2. ΜΕΤΑ τι;

Το πρόγραμμα εκτελεί διαδοχικά εντολές STA, NOP, LDA και μετά ακολουθεί διακοπή.

Άρα για να δώσει η ΜΕ τα κατάλληλα σήματα, θα πρέπει να διαβάσει:

Ανάκληση STA: στήλες 0-2

Εκτέλεση STA: στήλες 14-16

Ανάκληση NOP: στήλες 0-2

Εκτέλεση NOP: στήλες 10

Ανάκληση LDA: 0-2

Εκτέλεση LDA: Στήλες 11-13

Κύκλος διακοπής : στήλες 3-9

ΠΡΕΠΕΙ να υπάρχει ένας δείκτης, ο οποίος ανάλογα με το OPCODE (το OPCODE καθορίζει ποια εντολή θα εκτελεστεί) θα οδηγείται στο σημείο ΕΚΚΙΝΗΣΗ ΤΗΣ ΜΝΗΜΗΣ ΕΛΕΓΧΟΥ ΓΙΑ ΤΑ ΣΗΜΑΤΑ ΠΟΥ ΑΝΤΙΣΤΟΙΧΟΥΝ ΣΤΗΝ ΕΝΤΟΛΗ. Κάθε εντολή έχει μία διεύθυνση εκκίνησης.

Η NOP έχει την 10, η LDA την 11, η STA έχει την 14

Για τα σήματα που ορίζουν τις πράξεις της ΑΛΜ, δείτε σελ.511 πίνακας 14.3

STA

T3: MAR<- MDR(Address), άρα MARin=1, MDRout=1 (στήλη 14)

T4: MDR<-ACC, άρα MDRin=1, ACCout=1

T5: M[MAR] <-MDR,

ΠΩΣ θα ορίσουμε ο δείκτης των θέσεων μνήμης να πηγαίνει κάθε φορά στην κατάλληλη θέση, βάσει του OPCODE.

ΕΠΕΞΕΓΗΣΗ ΣΧΗΜΑΤΟΣ. Ο IR διαβάζει ένα OPCODE, το οποίο περνάει από έναν αποκωδικοποιητή. Τι είδους είναι αυτός ο αποκωδικοποιητής;

ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗΣ

Ο αποκωδικοποιητής είναι ένα κύκλωμα, το οποίο δεχόμενο ένα opcode θα πρέπει να παράγει το σημείο εκκίνησης της αντίστοιχης εντολής.

Έστω ότι τα OPCODE 6 εντολών είναι τα εξής

000 NOP

001 LDA

010 STA

011 AND

100 JSR

101 NOT

Έστω 6 εντολές 3 bit opcode

Ο αποκωδικοποιητής πρέπει διαβάζοντας τα OPCODE να παράγει έξοδο τα σημεία εκκίνησης.

Αν η είσοδος από τον IR=000 (OPCODE της NOP) τότε πρέπει να παράγει ως έξοδο το 10 (σημείο εκκίνησης της NOP)

Αν η είσοδος από τον IR=001 (OPCODE της LDA) τότε πρέπει να παράγει ως έξοδο το 11 (σημείο εκκίνησης της LDA)

Αν η είσοδος από τον IR=010 (OPCODE της STA) τότε πρέπει να παράγει ως έξοδο το 14 (σημείο εκκίνησης της STA)

Αν η είσοδος από τον IR=011 (OPCODE της AND) τότε πρέπει να παράγει ως έξοδο το 17 (σημείο εκκίνησης της AND). Επειδή η AND απαιτεί 4 βήματα (17-20, βλ. σελ 506), η επόμενη εντολή JSR αποθηκεύεται με σημείο εκκίνησης τη στήλη 21.

Αν η είσοδος από τον IR=100 (OPCODE της JSR) τότε πρέπει να παράγει ως έξοδο το 21 (σημείο εκκίνησης της JSR). Επειδή η AND απαιτεί 5 βήματα (21-25, βλ. σελ 507), η επόμενη εντολή JSR αποθηκεύεται με σημείο εκκίνησης τη στήλη 26.

Αν η είσοδος από τον IR=101 (OPCODE της NOT) τότε πρέπει να παράγει ως έξοδο το 26 (σημείο εκκίνησης της NOT)

ΕΡΩΤΗΣΗ: Πόσες εισόδους και πόσες εξόδους έχει αυτό το κύκλωμα αποκωδικοποίησης; Είσοδοι = 3 όσο και το μήκος του opcode και οι έξοδοι είναι τόσες ώστε να είναι δυνατός ο σχηματισμός της μεγαλύτερης υπαρκτής τιμής διεύθυνσης εκκίνησης. Στο παράδειγμά μας αυτή είναι το 26. Άρα χρειάζομαι 5 bits.

ΕΙΣΟΔΟΙ			Δεκ. Τιμή	ΕΞΟΔΟΙ				
I2	I1	I0		F4	F3	F2	F1	F0
0	0	0	10	0	1	0	1	0
0	0	1	11	0	1	0	1	1
0	**1**	**0**	14	0	1	1	1	0
0	1	1	17	1	0	0	0	1
0	**0**		21	1	0	1	0	1
1	0	1	26	1	1	0	1	0

Για κάθε έξοδο ξεχωριστά απλοποιούμε με χάρτη Karnaugh και βρίσκουμε την απλοποιημένη έκφραση την οποία υλοποιούμε με κύκλωμα.

Ενδεικτικά, η λογική έκφραση της F2= I2’ I1 I0’ + I2 I1’ I0’

ΣΧΗΜΑ: ο IR διαβάζει το OPCODE στο τελευταίο βήμα της ανάκλησης και το προωθεί στη μονάδα ελέγχου, η οποία το αποκωδικοποιεί με τον τρόπο που είδαμε και περνάει την τιμή στον μPC (μικρο PC).

ΠΑΡΑΔΕΙΓΜΑ: Ένα πρόγραμμα εκτελεί τις εντολές LDA, STA με OPCODE 1 και 2 αντίστοιχα.

IR= 001 (από το βήμα 2 της ανάκλησης). Η τιμή περνάει ως είσοδος στον αποκωδικοποιητή, ο οποίος επιστρέφει έξοδο 11 = 01011. Η τιμή 01011 πηγαίνει στον mPC, ο οποίος είναι ένας μετρητής. Ο μετρητής είναι ένα κύκλωμα, το οποίο σε κάθε παλμό ρολογιού αυξάνεται κατά 1, ενώ έχει τη δυνατότητα μηδενισμού.

μPC θα είναι ένας καταχωρητής με χωρητικότητα τόσα ώστε να μπορεί να μετρήσει ως την τελευταία στήλη του πίνακα. (Στο παράδειγμά μας ως το 27). Άρα αποτελείται από 5 flip-flop και η αρχική του τιμή θα είναι 01011. Η τιμή 01011 ζητείται από τη μνήμη ελέγχου και η μνήμη ελέγχου διαβάζει τη στήλη 11 η οποία είναι

101000000000000. Καθένα από αυτά τα bit αντιστοιχίζεται σε ένα σήμα in, out ενός καταχωρητή. Ο **καταχωρητής μικροεντολών έχει μέγεθος τόσα bit όσα είναι τα σήματα συνολικά και κάθε έξοδός του συνδέεται με ένα σήμα από αυτά που απαιτείται να ενημερώνονται κάθε χρονική στιγμή δηλαδή ένα σήμα ελέγχου.**

ΠΡΟΚΕΙΤΑΙ για καταχωρητή, στο συγκεκριμένο παράδειγμα 15bit άρα αποτελούμενο από 15 flip-flop. Καθένα από αυτά τα flip – flop έχει μία έξοδο Q. Κάθε έξοδος Q τροφοδοτεί ένα διαφορετικό σήμα ελέγχου.

1 0 1 0 ……

MARin MDRin MDRout ACCin

Π.χ., επειδή MARin =1, ο MAR διαβάζει.

Επειδή MDRin =0, ο MDR δεν κάνει κάτι σε αυτό το βήμα

ΤΕΛΕΙΩΝΕΙ το βήμα T3 της LDA (Στήλη 11). Εδώ, στον πίνακα θα υπάρχουν και τα σήματα F,G. Όταν τελειώσει το βήμα T3 θα συνεχίσει να είναι F=1 (εκτέλεση) και G=0(ΟΧΙ διακοπή). Αυτό σημαίνει ότι συνεχίζεται η εντολή και πρέπει ο μPC να αυξηθεί. Άρα γίνεται 12, ζητείται η στήλη 12, διαβάζεται

010000000110000. Τα bit αυτά αποθηκεύονται στον καταχωρητή μικροεντολών και «διανέμονται» ομοίως στα σήματα

MARin=0

MDRin=1 …..

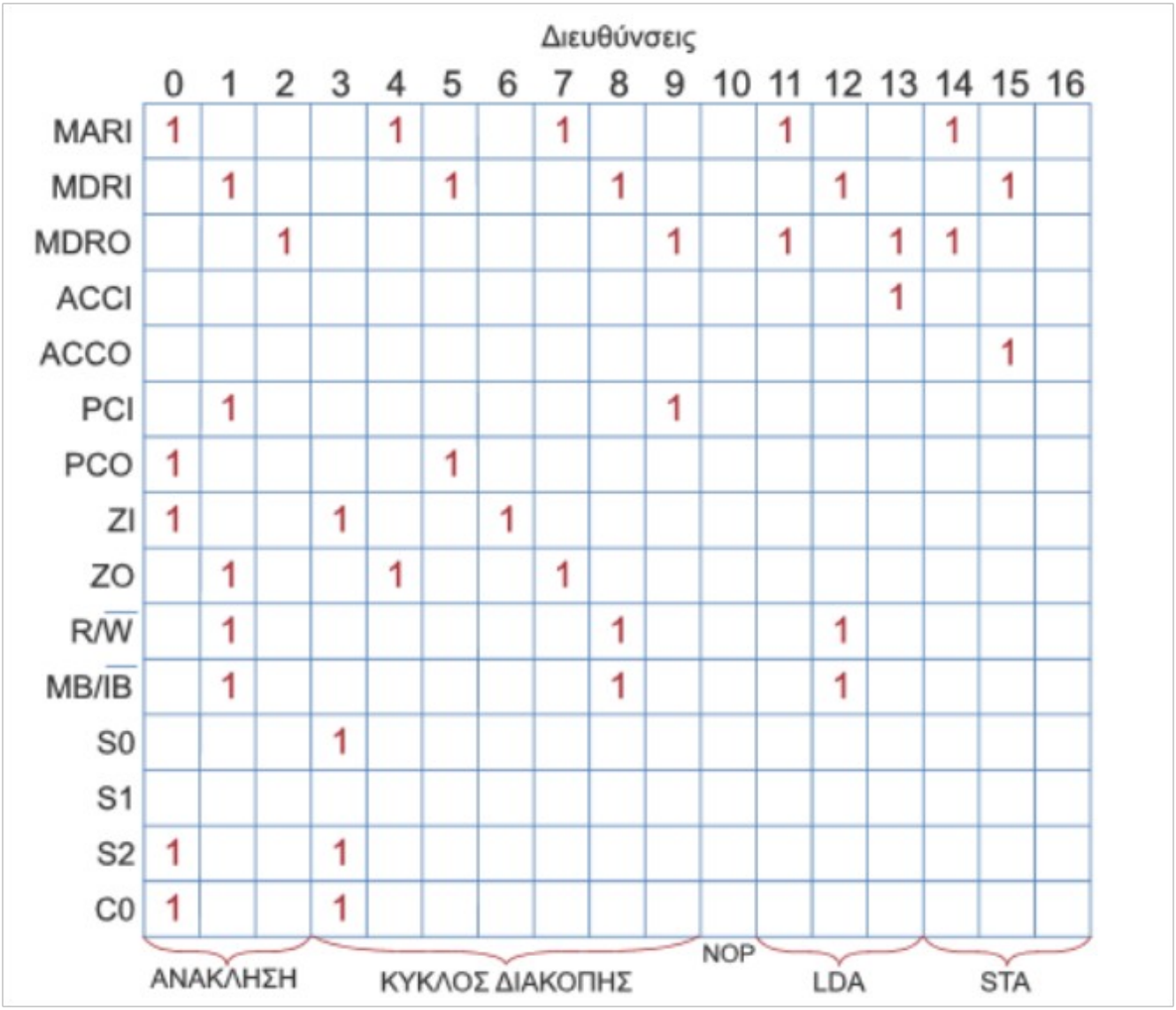
Όσο F=1 και G=0 έχουμε εκτέλεση. Στη στήλη 13 έχουμε το τέλος της εντολής LDA. Εκεί στη στήλη 13 θα υπάρχει η τιμή F=0. Η γραμμή που αντιστοιχεί στο F θα είναι 1 για τις στήλες 11 και 12 (εκτέλεση) και 0 για τη στήλη 13 (ΤΕΛΟΣ ΕΚΤΕΛΕΣΗΣ). Άρα ο μPC μηδενίζεται και ξεκινάμε από την αρχή. Δηλ. ο IR διαβάζει το επόμενο OPCODE κ.ο.κ.

ΜΕΓΕΘΟΣ ΜΝΗΜΗΣ: Πλήθος όλων των σημάτων (γραμμές) Χ άθροισμα όλων των βημάτων όλων των εντολών και των κύκλων ανάκλησης και διακοπής.

Αν προσθέταμε τα σήματα IRin, F, G, θα είχαμε μέγεθος 18x16= 288 bit.

Αν είχαμε 27 στήλες (δηλαδή τις 6 εντολές που αναφέραμε παραπάνω) θα ήταν 27x18

ΜΝΗΜΗ ΕΛΕΓΧΟΥ

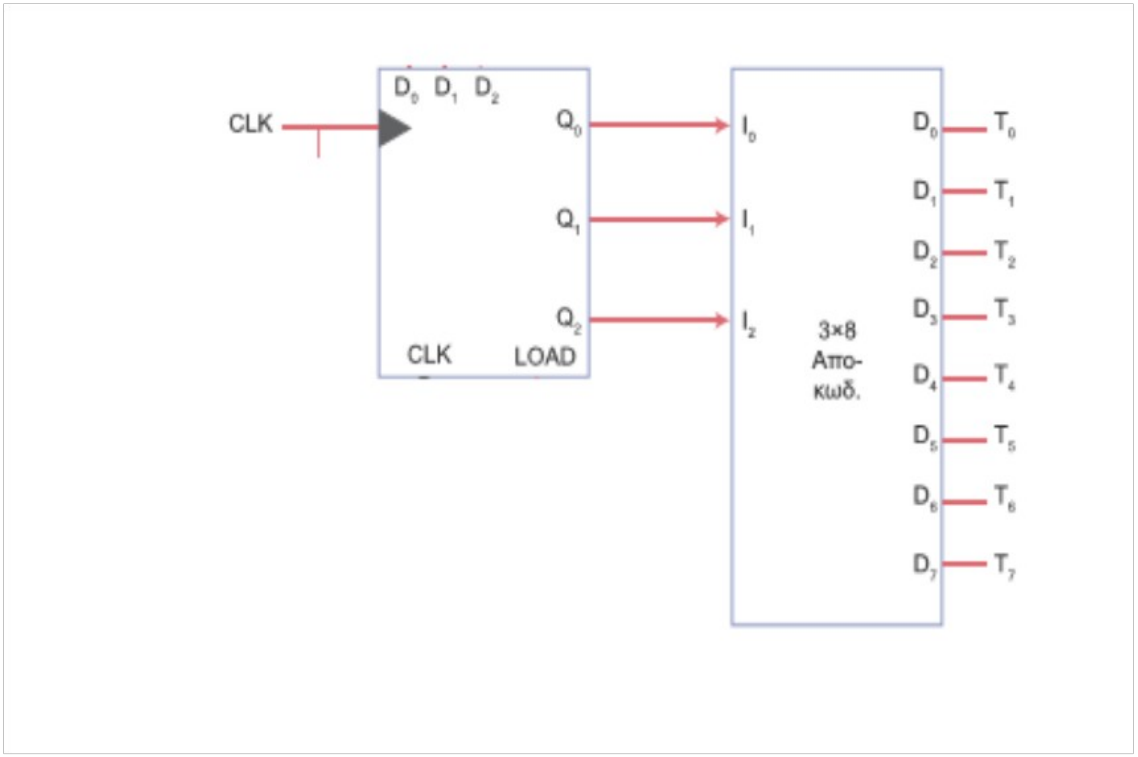


R/W: Σήμα ανάγνωσης –εγγραφής

MB/IB’: Όταν ο MDR διαβάζει από τη μνήμη, το σήμα αυτό γίνεται 1, απομονώνοντας τον MDR από τον εσωτερικό δίαυλο. Αν είναι 0, τότε διαβάζει από τον δίαυλο άρα απομονώνεται από τη μνήμη.

S,C: S0,S1,S2,C0=0011. Ο κωδικός αυτός αποκωδικοποιείται για να επιλεγεί η αντίστοιχη αριθμητική ή λογική λειτουργία της ALU. Η στήλη 0 αντιστοιχεί στο βήμα T0 της ανάκλησης επομένως πρόκειται για μία λειτουργία αύξησης του PC κατά 1.

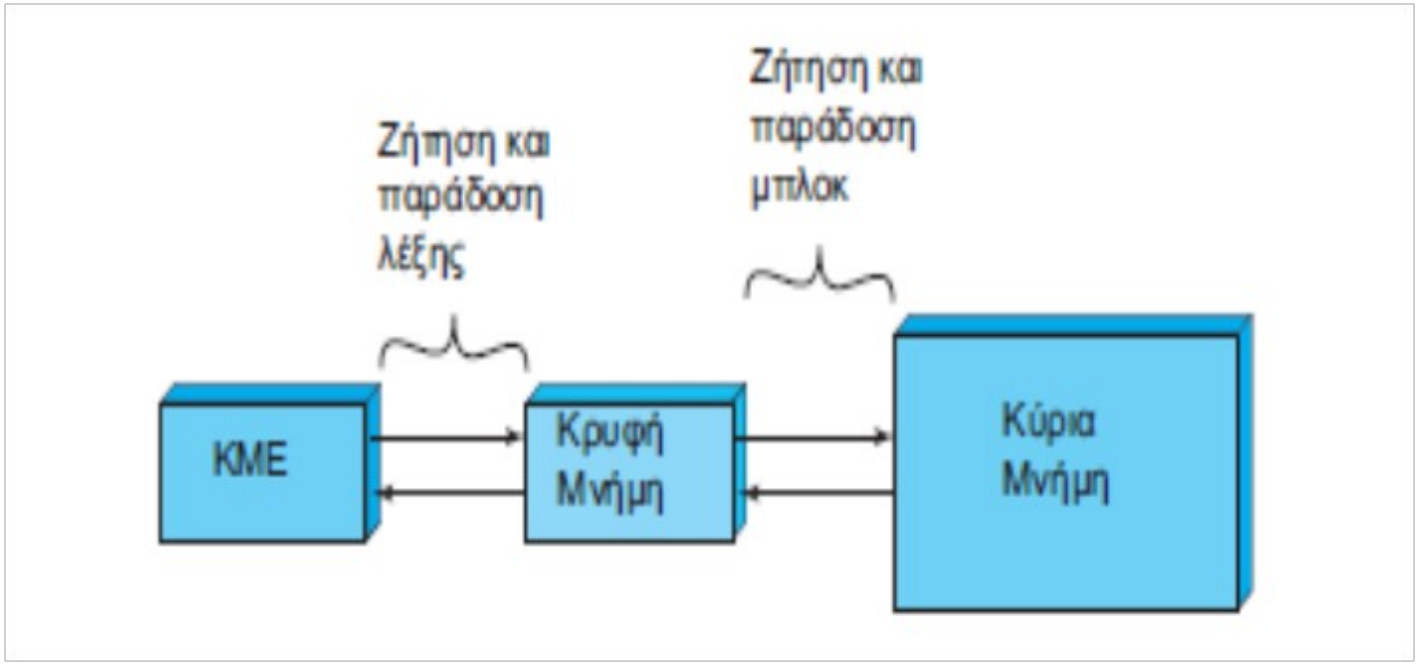
ΠΑΡΑΓΩΓΗ ΒΗΜΑΤΩΝ ΧΡΟΝΙΣΜΟΥ



Έστω ότι η μεγαλύτερη εντολή μας σε πλήθος μικρολειτουργιών απαιτεί 5 βήματα (συν 3 την ανάκληση, σύνολο 8, T0-T7). Αυτό σημαίνει ότι χρειαζόμαστε ένα κύκλωμα, το οποίο θα μπορεί να μετράει από 0 ως 7. Ο μετρητής μετράει από 0-7 και η τιμή που αποθηκεύει καταχωρείται σε 3 flip flop (Τρία flip flop είναι αρκετά για να μετρήσουμε ως το 7.) Οι έξοδοι των flip flop τροφοδοτούν ένα αποκωδικοποιητή. Άρα αν το flip flop αποθηκεύει τις τιμές 0, 1, 1, αυτό σημαίνει ότι οι είσοδοι του

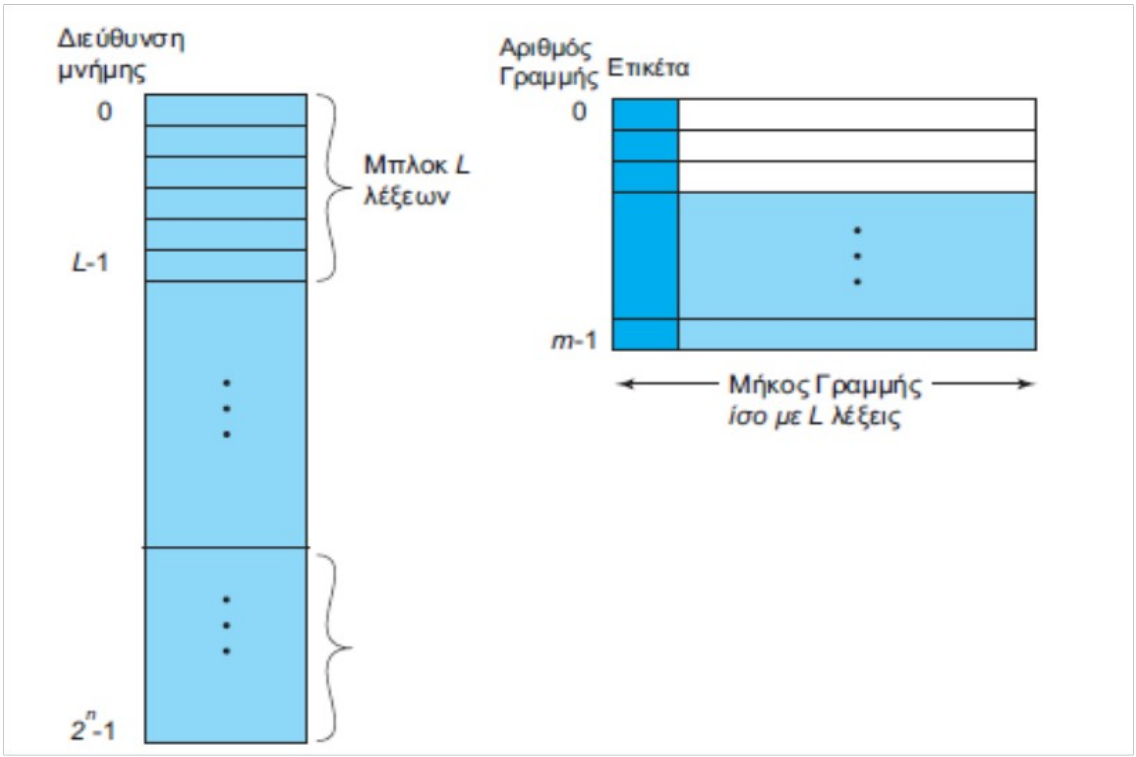
Αποκωδικοποιητή I2,I1,I0 θα είναι 0,1,1 άρα το σήμα εξόδου που θα γίνει 1 είναι το T3. Αυτό το σήμα θα δοθεί στο κύκλωμα της μονάδας ελέγχου που παράγει τα σήματα εισόδου.

ΚΡΥΦΗ ΜΝΗΜΗ



Γραμμές της κρυφής μνήμης και μπλοκ της κύριας. Αυτά τα 2 είναι το ίδιο πράγμα, αλλά χρησιμοποιούμε διαφορετικές ονομασίες για λόγους σαφήνειας.

ΓΕΝΙΚΗ ΔΟΜΗ



RAM η οποία περιέχει 2n-1 γραμμές, καθεμία από αυτές τις γραμμές περιέχει ένα πλήθος από bytes.

Η κρυφή μνήμη η οποία περιέχει m γραμμές κάθε μία από τις οποίες περιέχει ένα πλήθος από byte.

Όταν ένα πρόγραμμα ζητήσει μία γραμμή της RAM, τότε στην κρυφή μνήμη δεν μεταφέρεται ΜΟΝΟ η γραμμή αυτή αλλά ολόκληρο το μπλοκ στο οποίο αυτή ανήκει.

Έστω ότι L=8. Αν ζητηθεί για πρώτη φορά από τη CPU η λέξη 5, τότε όλο το μπλοκ θα μεταφερθεί στην κρυφή μνήμη δηλαδή και οι 8 λέξεις του μπλοκ, οι οποίες θα γραφτούν σε μία γραμμή της κρυφής μνήμης.

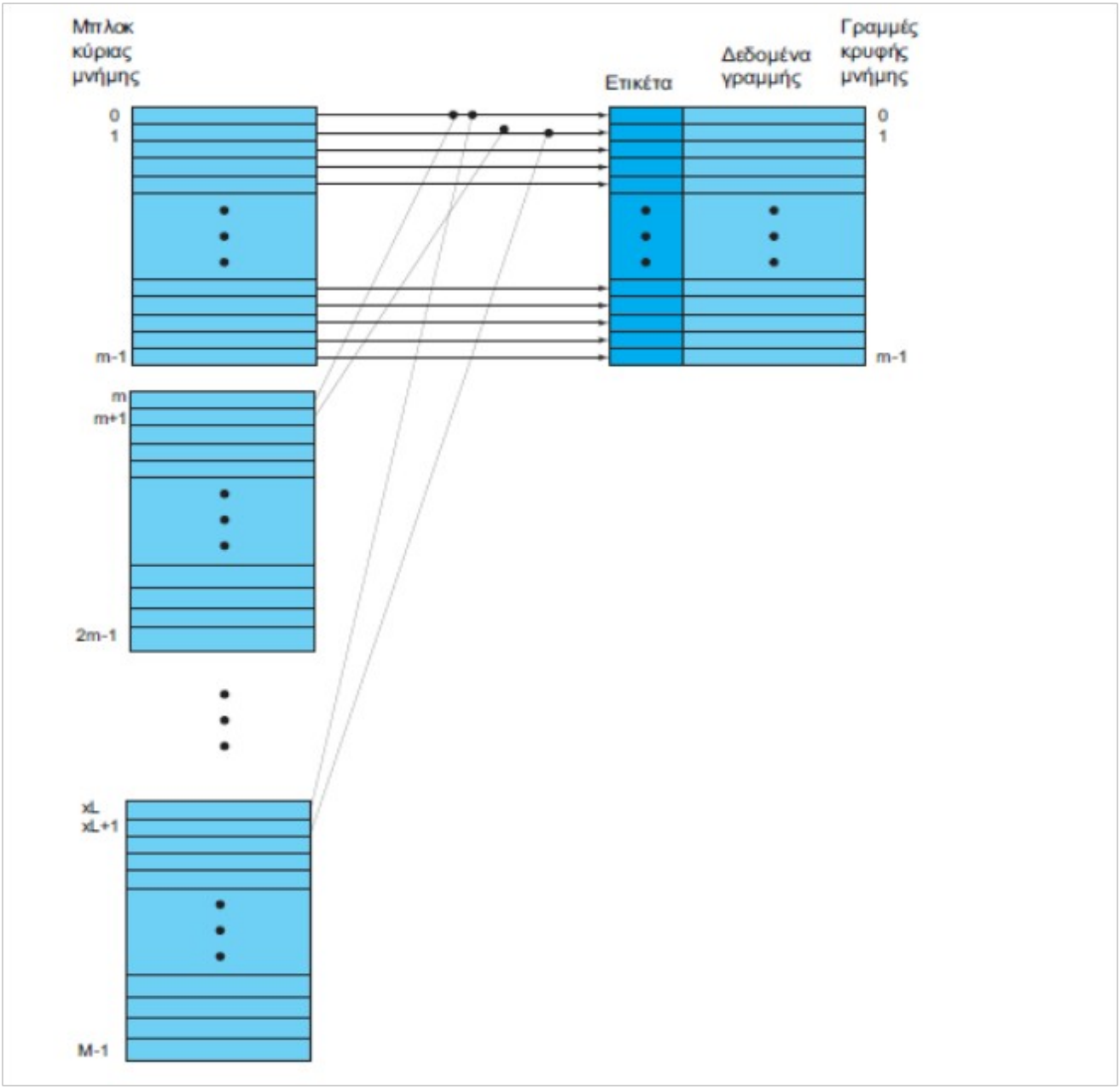
Το μέγεθος γραμμής της κρυφής μνήμης είναι ίσο με το μέγεθος μπλοκ της RAM.

Γιατί η μεταφορά δεν γίνεται ανά λέξη της RAM; Οι υπολογιστές λειτουργούν βάσει της τοπικότητας αναφοράς. Είναι μία αρχή η οποία λέει ότι όταν χρησιμοποιηθούν τα δεδομένα μίας θέσης, το πιθανότερο είναι αμέσως μετά να ζητηθούν τα δεδομένα της επόμενης διεύθυνσης. Για τον λόγο αυτό μεταφέρουμε μπλοκ από την Ram και όχι μία λέξη κάθε φορά.

Ερώτημα: Ποια μπλοκ της RAM αντιστοιχίζονται σε ποιες γραμμές της cache. Μπορεί δηλαδή κάθε μπλοκ να τοποθετείται σε όποια γραμμή θέλουμε ή υπάρχουν κανόνες; Η διεύθυνση της λέξης 0 που ανήκει στο μπλοκ 0 της RAM, όταν ζητηθεί και πρέπει να μεταφερθεί στην κρυφή μνήμη, που θα τοποθετηθεί, δηλαδή ποια γραμμή της κρυφής μνήμης είναι σχεδιασμένη για να υποδεχτεί το μπλοκ 0 της κύριας μνήμης (το οποίο περιέχει τη λέξη 0 και τις λέξεις ως L-1). Οι απαντήσεις που θα δοθούν σε αυτός το ερώτημα προσδιορίζουν και τις μορφές οργάνωσης της κρυφής μνήμης.

Επειδή υπάρχουν 3 βασικές μορφές, εδώ θα τις αναλύσουμε και θα κάνουμε τις συγκρίσεις.

CACHE ΑΜΕΣΗΣ ΣΥΣΧΕΤΙΣΗΣ



Αν η κρυφή μνήμη έχει m γραμμές, τότε το μπλοκ i της κύριας μνήμης αντιστοιχίζεται στη γραμμή j της κρυφής βάσει της σχέσης:

j= i mod m

Έστω ότι η κρυφή μνήμη έχει 8 γραμμές, m=8. Τότε τα μπλοκ 0, 8, 16, 24, 32, κλπ αντιστοιχίζονται στη γραμμή 0 της cache, επειδή 0 mod 8=0, 8 mod 8 =0, 16 mod 8 =0 κλπ

Ομοίως, τα μπλοκ 1, 9, 17, 25, 33, κλπ της RAM αντιστοιχίζονται στη γραμμή 1 επειδή

1 mod 8=1, 9 mod 8,=1 17 mod 8=1 κλπ

Με άλλα λόγια, όποτε ζητηθεί μία διεύθυνση που ανήκει στα μπλοκ 0 ή 8 ή 16 ή 24 κλπ τότε όλο το αντίστοιχο μπλοκ μεταφέρεται στη γραμμή 0 της cache. Αυτό σημαίνει ότι αυτά τα μπλοκ αντιστοιχίζονται στη γραμμή 0 και ΜΟΝΟ σε αυτή. **Δεν μπορούν να βρίσκονται ταυτόχρονα 2 από αυτά στην cache. ΑΜΟΙΒΑΙΑ αποκλειόμενα μπλοκ.**

Επειδή τα μπλοκ (m=8) 0, 8,,16, 24 κ.ο.κ έχουν όλα modulo 0 όταν διαιρούνται με το 8, αυτό σημαίνει ότι 2 εξ αυτών ΔΕΝ μπορούν να βρίσκονται ταυτόχρονα στην κρυφή όταν έχουμε μνήμη άμεσης συσχέτισης.

Δηλαδή όλα τα μπλοκ της RAM τα οποία στη διαίρεσή τους με το πλήθος γραμμών της cache **ΈΧΟΥΝ** ίδιο modulo είναι αμοιβαία αποκλειόμενα δηλαδή δεν μπορούν να βρίσκονται μαζί στην κρυφή μνήμη, όταν έχουμε μνήμη άμεσης συσχέτισης.

ΠΑΡΑΔΕΙΓΜΑ

Μέγεθος λέξης 1 byte

- RAM 128 bytes
- Μέγεθος μπλοκ/γραμμής 4 λέξεις
- Cache 32 bytes

- Α) Ανάλυση διεύθυνσης
- Β) Διευθύνσεις λέξεων που σχετίζονται με τη γραμμή 2 της cache

Παράδειγμα: Μέγεθος λέξης 1 byte, το μέγεθος της RAM είναι 128 bytes, κάθε μπλοκ της RAM ή γραμμή της cache αποτελείται από 4 λέξεις άρα 4 bytes. Η cache είναι 32 bytes.

Να αναλύσετε τη διεύθυνση της κύριας μνήμης

Απάντηση

Η μορφή μίας διεύθυνσης μνήμης RAM όταν χρησιμοποιείται η οργάνωση κρυφής μνήμης άμεσης συσχέτισης είναι η εξής

Ετικέτα (Tag) Γραμμή Λέξη

Η RAM έχει μέγεθος 128 bytes άρα ($128 = 2^7$) άρα απαιτούνται 7 bit διεθυνσιοδότησης, με άλλα λόγια αυτά τα 7 bit είναι που θα μοιραστούν στα 3 πεδία.

Πόσα bit είναι το tag;

Πόσα είναι η γραμμή;

Πόσα είναι η λέξη;

Και τι δείχνουν αυτά τα τρία πεδία.

Λέξη: Χρειαζόμαστε να βρούμε το πλήθος bit που απαιτείται για να διευθυνσιοδοτήσουμε τις λέξεις που υπάρχουν σε κάθε μπλοκ ή γραμμή. Στο παράδειγμα είναι 4 άρα για το πεδίο Λέξη χρειάζονται 2 bit. Η κάθε λέξη που ανήκει σε ένα μπλοκ ή γραμμή θα έχει **τοπική διεύθυνση** 00, 01, 10 ή 11 (0-3).

Γραμμή: Χρειάζεται να βρούμε πόσες γραμμές έχει η cache.

Η cache έχει μέγεθος 32 bytes και κάθε γραμμή είναι 4 byte. Η cache έχει 32/4 = 8 γραμμές, άρα απαιτούνται 3 bit για τη διευθυνσιοδότηση αυτών των γραμμών

Άρα οι 8 γραμμές τις κρυφής μνήμης θα έχουν διεύθυνση 000, 001, 010, 011, 100, 101, 110, 111.

Απαιτούνται 3 bit για τη γραμμή, 2 για τη λέξη και 7-5=2 για το tag. TI δείχνει το tag;

Πεδίο Γραμμή: Δείχνει μία από τις γραμμές της κρυφής μνήμης

Πεδίο λέξη: Δείχνει μία από τις λέξεις

Πεδίο tag; ΔΕΝ είναι σαφές. Προς το παρόν, το tag είναι ότι περισσεύει από τα πεδία Γραμμή και Λέξη. Θα απαντηθεί από το ερώτημα 2.

Tag Line Byte

2 3 2 -> Μορφή της διεύθυνσης της RAM όταν χρησιμοποιείται άμεση συσχέτιση

Ερώτημα Β) Αναδιατυπώνεται ως εξής: Ποιες διευθύνσεις λέξεων της RAM αντιστοιχίζονται με την γραμμή 2 της cache; Ποιες διευθύνσεις λέξεων της RAM, όταν ζητηθούν, θα τοποθετηθούν στη γραμμή 2 της cache;

Απάντηση

Για να βρούμε ποιες διευθύνσεις αντιστοιχίζονται με τη γραμμή 2 της κρυφής μνήμης θα πρέπει να θέσουμε το πεδίο Line =2 (σταθερό) και να βρούμε όλους τους δυνατούς συνδυασμούς των πεδίων tag, byte.

Tag Line Byte

XX **010** YY

1ος συνδυασμός: XX=00, και YY= 00-11

Προκύπτουν οι διευθύνσεις

00 010 00 = 8

00 010 01 = 9

00 010 10 = 10

00 010 11 =11

ΠΑΡΑΤΗΡΗΣΗ: Εφόσον κάθε μπλοκ αποτελείται από 4 λέξεις, το μπλοκ 0 της RAM θα περιέχει τις λέξεις 0-3, το μπλοκ 1 τις λέξεις 4-7 και το μπλοκ 2 τις λέξεις 8-11. Το μπλοκ 3 τις λέξεις 12-15…….

Οι διευθύνσεις 8-11. Σε ποιο μπλοκ ανήκουν αυτές οι διευθύνσεις; Για να βρούμε το μπλοκ, διαιρούμε 8/4=2. 9/4=2. 10/4=2, 11/4 = 2.

Ο αριθμός μπλοκ 2 αν διαιρεθεί με το πλήθος γραμμών m=8 της κρυφής μνήμης θα δώσει υπόλοιπο 2. Άρα το μπλοκ 2 της RAM το οποίο περιέχει τις διευθύνσεις 8-11, αντιστοιχίζεται στην γραμμή 2 της κρυφής μνήμης.

2ος συνδυασμός: XX=01, YY=00-11

Προκύπτουν οι διευθύνσεις

01 010 00 = 40

01 010 01 = 41

01 010 10 = 42

01 010 11 = 43

Οι διευθύνσεις 40-43 ανήκουν στο μπλοκ 40/4=10, 41/4=10, 42/4=10, 43/4=10. Άρα στο μπλοκ 10. Ο αριθμός μπλοκ 10 της RAM αν διαιρεθεί με το πλήθος γραμμών της κρυφής μνήμης που είναι m=8 , θα δώσει modulo 10 mod 8 =2. Άρα, οι διευθύνσεις από 40-43 που ανήκουν στο μπλοκ 10 της RAM επίσης αντιστοιχίζονται στην γραμμή 2 της κρυφής μνήμης.

3ος συνδυασμός: XX=10, YY=00-11

Προκύπτουν οι διευθύνσεις

10 010 00 = 72

10 010 01 = 73

10 010 10 = 74

10 010 11 = 75

Αν διαιρέσουμε 72/4 = 18 άρα αυτές οι 4 διευθύνσεις ανήκουν στο μπλοκ 18 της RAM και αν διαιρέσουμε το 18 με το m=8 θα πάρουμε ξανά Modulo 2. Άρα και οι διευθύνσεις του μπλοκ 18 αντιστοιχίζονται στη γραμμή 2.

4ος συνδυασμός: XX=11, YY=00-11

Προκύπτουν οι διευθύνσεις

11 010 00 = 104

11 010 01 = 105

11 010 10 = 106

11 010 11 = 107

Αν διαιρέσουμε 104/4 = 26 άρα αυτές οι 4 διευθύνσεις ανήκουν στο μπλοκ 26 της RAM και αν διαιρέσουμε το 26 με το m=8 θα πάρουμε ξανά Modulo 2. Άρα και οι διευθύνσεις του μπλοκ 26 αντιστοιχίζονται στη γραμμή 2.

Άρα τα μπλοκ 2, 10, 18, 26 αντιστοιχίζονται στη γραμμή 2.

Τι δείχνει το tag;

Το πλήθος bits του tag δείχνει πόσα block της RAM αντιστοιχίζονται σε κάθε γραμμή. Εδώ επειδή είναι 2 bit, σημαίνει ότι 4 μπλοκ της RAM αντιστοιχίζονται σε κάθε γραμμή της κρυφής μνήμης. ΠΩΣ αλλιώς μπορούμε να επαληθεύσουμε αυτό το γεγονός;

Η RAM έχει μέγεθος 128 bytes άρα αν διαιρέσω με το 4 έχει 32 μπλοκ. Τα 32 μπλοκ αντιστοιχίζονται σε 8 γραμμές της κρυφής μνήμης άρα 4 μπλοκ σε κάθε γραμμή. Προφανώς, το tag έχει μήκος 2 bit (22=4).

Tag

00 Είναι την τοπική διεύθυνση (μπλοκ 0) που αντιστοιχίζεται σε μία γραμμή

01 Είναι την τοπική διεύθυνση (μπλοκ 1) που αντιστοιχίζεται σε μία γραμμή

10 Είναι την τοπική διεύθυνση (μπλοκ 2) που αντιστοιχίζεται σε μία γραμμή

11 Είναι την τοπική διεύθυνση (μπλοκ 3) που αντιστοιχίζεται σε μία γραμμή

Το μπλοκ 2 της RAM είναι το μπλοκ με τοπική διεύθυνση 0 (tag=00) που αντιστοιχίζεται στη γραμμή 2 της κρυφής μνήμης

Το μπλοκ 10 της RAM είναι το μπλοκ με τοπική διεύθυνση 1 (tag=01) που αντιστοιχίζεται στη γραμμή 2 της κρυφής μνήμης

Το μπλοκ 18 της RAM είναι το μπλοκ με τοπική διεύθυνση 2 (tag=10) που αντιστοιχίζεται στη γραμμή 2 της κρυφής μνήμης

Το μπλοκ 26 της RAM είναι το μπλοκ με τοπική διεύθυνση 3 (tag=11) που αντιστοιχίζεται στη γραμμή 2 της κρυφής μνήμης

ΠΑΡΑΔΕΙΓΜΑ

Κρυφή μνήμη με λέξεις 1 byte

- Μέγεθος RAM 32Kbytes
- Cache 4 Kbytes
- Μέγεθος μπλοκ/γραμμής 32 λέξεις

- Α) Ανάλυση διεύθυνσης
- Β) Μεγαλύτερη/μικρότερη διεύθυνση λέξεων που συσχετίζονται με τη γραμμή 64

Μέγεθος λέξης 1 byte, RAM 2^{15}, cache = 2^{12}, μέγεθος μπλοκ είναι 32 λέξεις = 2^5.
Επομένως, η διεύθυνση είναι 15 bit και αναλύεται ως εξής:

Λέξη: 5 bit (32 λέξεις/ μπλοκ)

Γραμμή: $2^{12}/ 2^5 = 2^7$=128, 7 bit για διευθυνσιοδότηση

Tag: 3 bit. Η RAM διαθέτει $2^{15}/ 2^5$ μπλοκ δηλαδή 1024. Υπάρχουν 1024 μπλοκ της RAM που αντιστοιχίζονται σε 128 γραμμές της cache. Αυτό σημαίνει ότι σε κάθε γραμμή της κρυφής μνήμης αντιστοιχίζονται 1024/128=8 μπλοκ. Άρα το μέγεθος του tag είναι 3.

Tag Line Byte

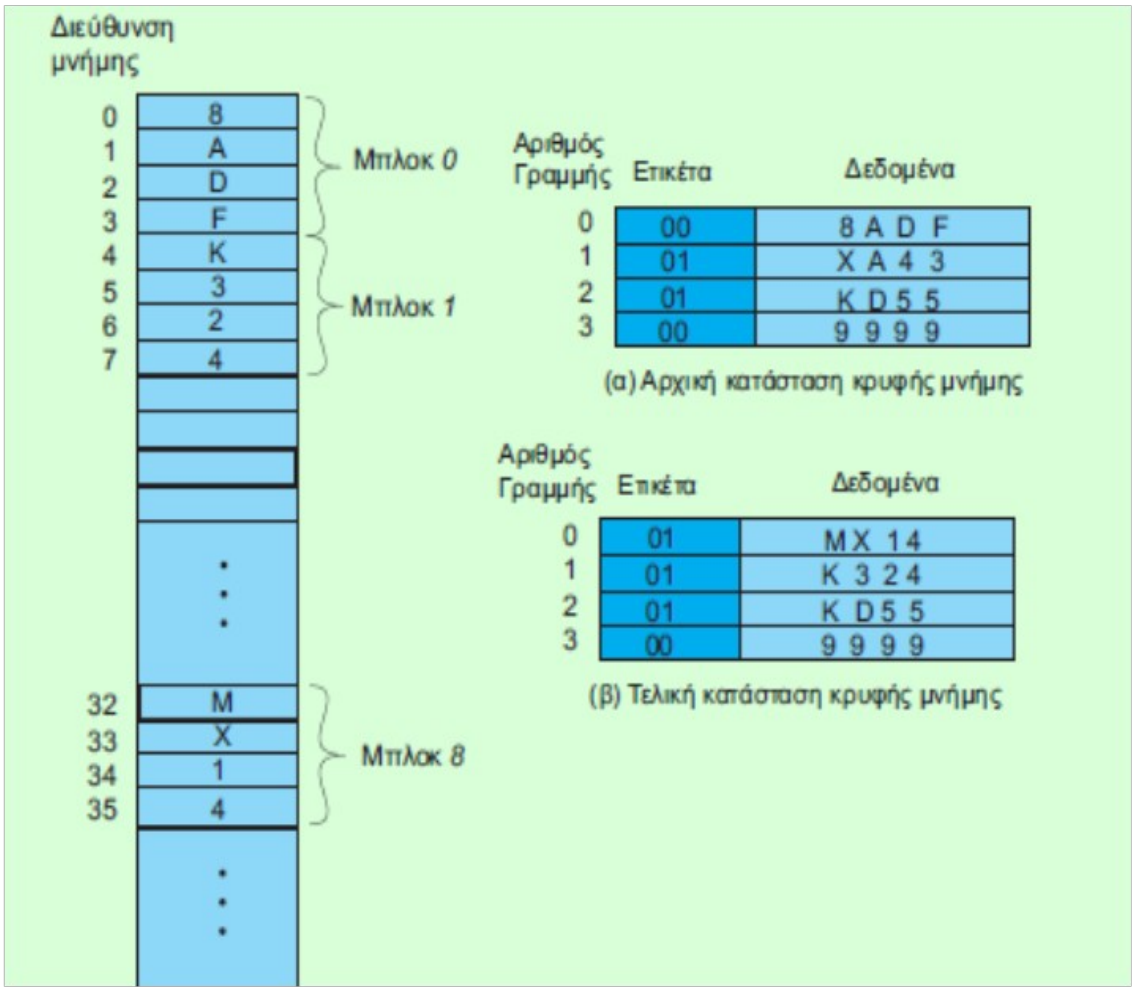
3 7 5

Γράφουμε τον αριθμό 64 με 7 bit 1000000 και το κρατάμε σταθερό. Άρα, η μικρότερη διεύθυνση είναι η

000 1000000 00000 = 2048

Η μεγαλύτερη είναι 111 1000000 11111 = 16384 + 8192 +4096 + 2048 + 31.

ΠΑΡΑΔΕΙΓΜΑ



Έστω μία RAM με μέγεθος 128 Bytes και μέγεθος λέξης/μπλοκ 4 bytes. Η κρυφή μνήμη είναι 32 Bytes. Η αρχική κατάσταση δίνεται στο σχήμα (α) δηλαδή καθώς τρέχει ένα πρόγραμμα έχει ήδη ζητήσει κάποιες διευθύνσεις και τα αντίστοιχα μπλοκ έχουν φορτωθεί. Η CPU συνεχίζει ζητώντας τις διευθύνσεις 2, 18, και 32. Να δείξετε την τελική κατάσταση της μνήμης. Πότε έχουμε hit και πότε miss;

Απάντηση

Ανάλυση διεύθυνσης: RAM 128 bytes άρα 7 bit διεύθυνσης. Επειδή έχουμε 4 λέξεις, το πεδίο byte θα είναι 2 bit. Επειδή η κρυφή μνήμη είναι 32 byte έχουμε 32/4 = 8 γραμμές δηλαδή απαιτούνται 3 bit για το πεδίο της γραμμής άρα το tag είναι 2 bit.

Η RAM έχει 128 byte άρα διαθέτει 128/4= 32 μπλοκ. Αυτά αντιστοιχίζονται σε 8 γραμμές. Άρα σε κάθε γραμμή (8 συνολικά) αντιστοιχίζονται 32/8=4 μπλοκ άρα σωστά το tag=2 bits.

Διεύθυνση 2: 00 000 10

Για να ελέγξει η κρυφή μνήμη αν τα δεδομένα της διεύθυνσης 2 υπάρχουν, κάνει τα εξής:

Εξετάζει την τιμή του πεδίου Line = 000.

Πηγαίνει στο tag directory. Είναι ένας μικρός κατάλογος που περιέχει τα tag των μπλοκ που έχουν αποθηκευτεί στις γραμμές της κρυφής μνήμης. Το αποθηκευμένο tag=**00**.

Συγκρίνει το αποθηκευμένο tag με το tag της ζητούμενης διεύθυνσης, το οποίο είναι 00. Αν τα 2 tag είναι ίδια, έχουμε hit δηλαδή Τα δεδομένα που ζητήθηκαν υπάρχουν ήδη στην κρυφή μνήμη και δεν χρειάζεται να προσκομιστούν από την RAM.

Από την στιγμή που το πεδίο Line της ζητούμενης διεύθυνσης 2 είναι ίσο με 000=0, το σύστημά μας συγκρίνει με το tag της γραμμής 0 γιατί η διεύθυνση 2 ΔΕΝ ΜΠΟΡΕΙ ΝΑ ΑΝΤΙΣΤΟΙΧΙΘΕΙ σε άλλη γραμμή παρά μόνο στη 0. Η διεύθυνση 2 υπάρχει στο μπλοκ 0 το οποίο αντιστοιχίζεται αποκλειστικά στη γραμμή 0.

Αυτό σημαίνει ότι, έχει ήδη ζητηθεί από τη CPU κάποια διεύθυνση του μπλοκ 0 (πιθανότατα η 0) και όλο το μπλοκ προσκομίστηκε στην κρυφή μνήμη για να είναι διαθέσιμες και οι διευθύνσεις 1-3.

Κάθε φορά που ζητείται μία διεύθυνση, γίνεται σύγκριση του tag της με το tag της γραμμής στην οποία αντιστοιχεί το μπλοκ της. Πόσους συγκριτές απαιτεί αυτή η μνήμη (η μνήμη άμεσης συσχέτισης) και γιατί;

Επειδή κάθε ζητούμενη διεύθυνση μπορεί να αντιστοιχίζεται ΜΟΝΟ σε μία γραμμή, η μνήμη άμεσης συσχέτισης απαιτεί 1 συγκριτή.

Διεύθυνση 33:

Η διεύθυνση 33 με 7 bit γράφεται ως: 01 000 00

Η διεύθυνση 33 αντιστοιχίζεται στη γραμμή 000=0. Άρα, για να βρούμε αν υπάρχει στην κρυφή μνήμη θα πρέπει να μεταβούμε στη γραμμή 0 και να συγκρίνουμε το tag που υπάρχει εκεί με το tag της εισερχόμενης διεύθυνσης. Το tag της εισερχόμενης διεύθυνσης 01 είναι διαφορετικό από το tag που αποθηκεύεται στην γραμμή 0. Άρα τα δεδομένα του μπλοκ (33/4=8) δεν υπάρχουν στην cache, άρα MISS. ΤΙ ΚΑΝΟΥΜΕ; Τα δεδομένα που δεν υπάρχουν προσκομίζονται από την RAM, και αντικαθιστούν τα δεδομένα που υπήρχαν στη γραμμή 0. Δηλαδή η ετικέτα γίνεται 01, τα δεδομένα MX14. Με άλλα λόγια, τοποθετήθηκε στην γραμμή 0 το μπλοκ με τοπική διεύθυνση 01. Αυτό είναι το μπλοκ 8, που περιέχει τις διευθύνσεις 32-35.

Διεύθυνση 7: 00 001 11

Η διεύθυνση 7 αντιστοιχίζεται στη γραμμή 1 (βρίσκεται στο μπλοκ 7/4=1 της RAM)

Στην γραμμή 1, η ετικέτα είναι 01. Άρα, έχουμε miss, υπάρχουν άλλα δεδομένα από τα ζητούμενα στη γραμμή 1, άλλο μπλοκ.

Φέρνουμε τα δεδομένα από το μπλοκ 1 και τα περνάμε στη γραμμή 1 της κρυφής μνήμης.

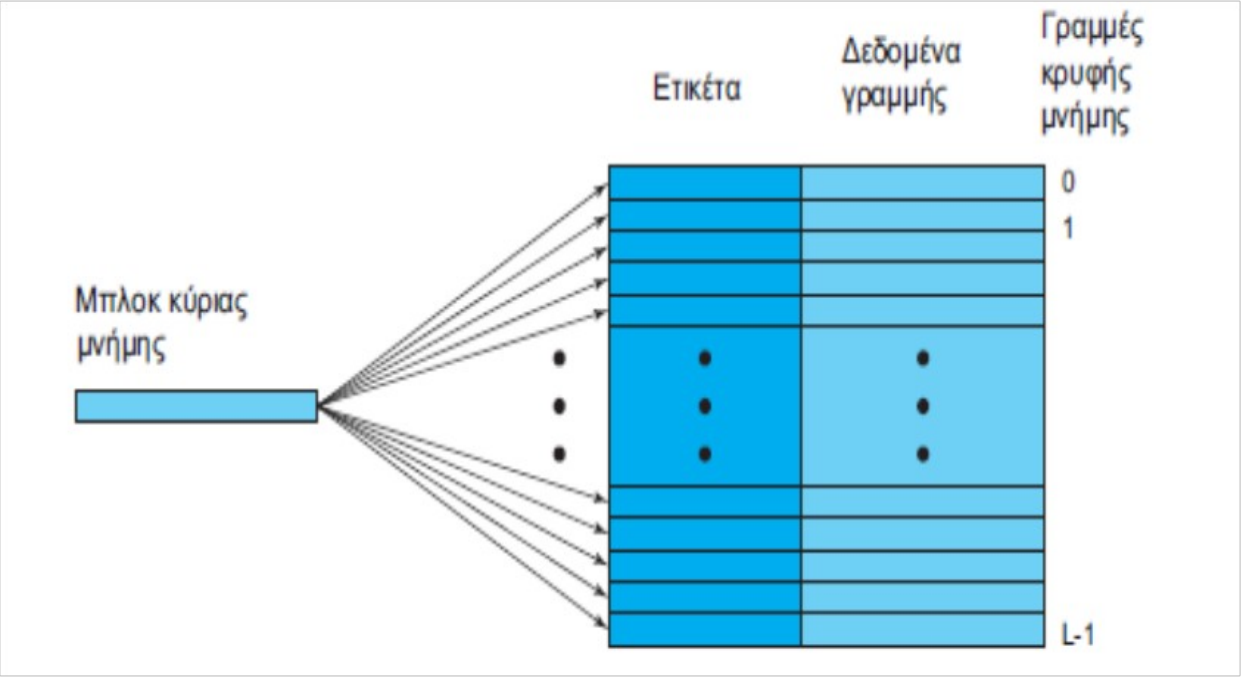
ΥΠΑΡΧΕΙ ΛΑΘΟΣ ΣΤΟ ΣΧΗΜΑ: το tag στην τελική κατάσταση για την γραμμή 1 θα είναι 00.

Αν κάπου παρακάτω ζητηθεί ξανά η διεύθυνση 3.

00 000 11

Αυτή αντιστοιχίζεται στη γραμμή 0. Εκεί το tag είναι 01 ενώ το tag της εισερχόμενης διεύθυνσης είναι 00. Έχουμε MISS Και η εγγραφή όσον αφορά τη γραμμή 0 θα επιστρέψει σε αυτήν που δείχνει το σχήμα (α).

Cache ΠΛΗΡΟΥΣ ΣΥΣΧΕΤΙΣΗΣ



Διαφοροποιήσεις σε σχέση με την κρυφή μνήμη άμεσης συσχέτισης.

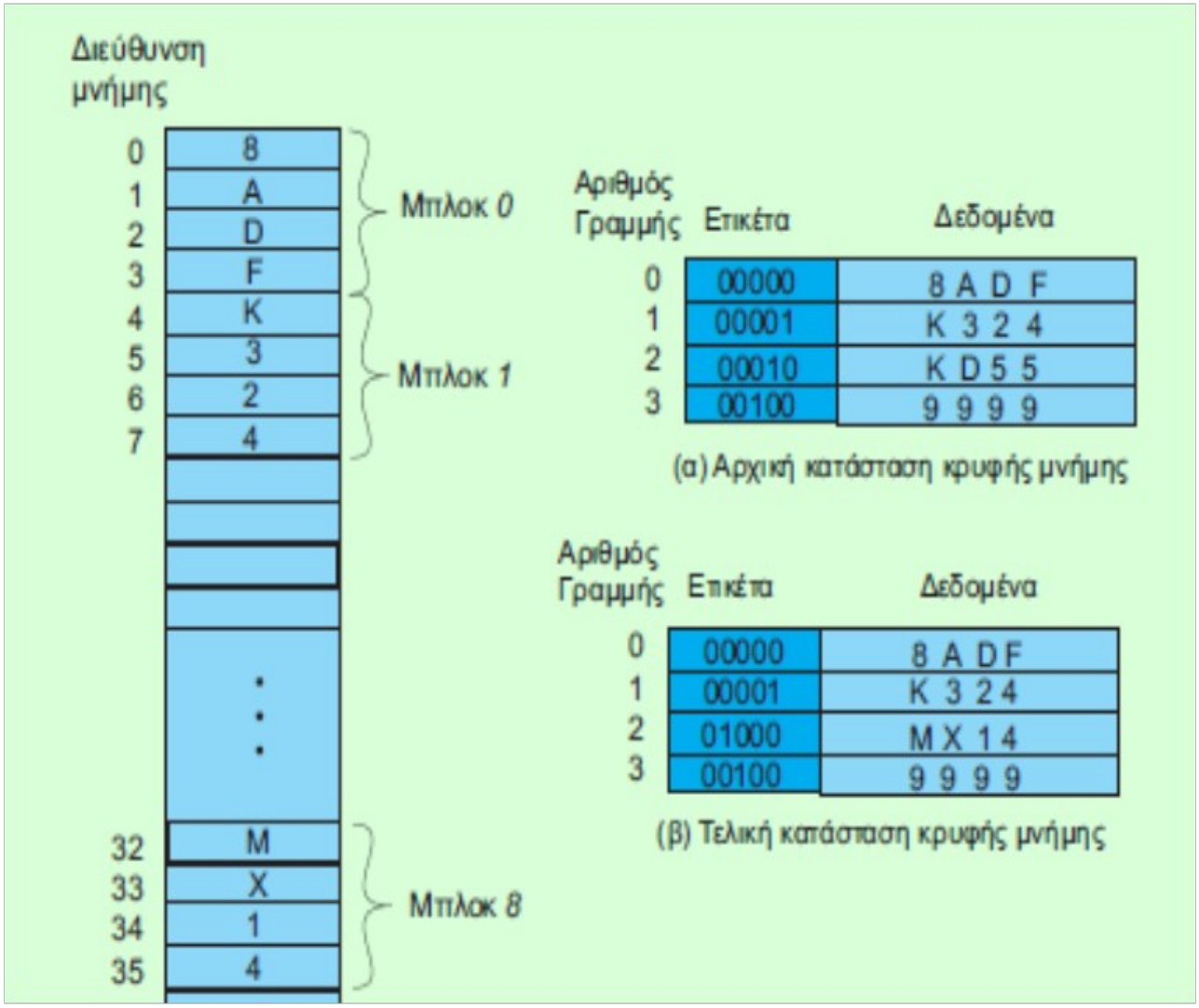
1) Μορφή της διευθυνσιοδότησης. Εδώ δεν υπάρχει το πεδίο Line, επειδή ακριβώς κάθε μπλοκ μπορεί να βρεθεί σε οποιαδήποτε γραμμή.

Tag Byte

2) Όταν εξετάζουμε αν ένα μπλοκ βρίσκεται στην κρυφή μνήμη, δεν ελέγχουμε το tag της ζητούμενης διεύθυνσης σε σχέση με το tag μίας μόνο γραμμής (όπως στην άμεση συσχέτιση), αλλά με όλα τα tags. Επειδή αυτές οι συγκρίσεις πρέπει να γίνουν ΠΑΡΑΛΛΗΛΑ, για να κερδίσουμε χρόνο, απαιτούνται τόσοι συγκριτές όσες είναι οι γραμμές της κρυφής μνήμης (αν m οι γραμμές της κρυφής μνήμης, m Συγκριτές)

Στην πλήρη συσχέτιση έχουμε περισσότερο hardware από θέμα συγκριτών, αλλά από την άλλη μπορούν οποιαδήποτε 2 μπλοκ της RAM να βρίσκονται ταυτόχρονα στην κρυφή μνήμη, δηλ. δεν υπάρχουν αμοιβαία αποκλειόμενα μπλοκ (στην άμεση συσχέτιση, τα μπλοκ με δείκτες που είχαν ίδιο mod m, όπου m το πλήθος γραμμών της cache, ήταν αμοιβαία αποκλειόμενα, δεν μπορούσαν να βρεθούν μαζί στην κρυφή μνήμη).

ΠΑΡΑΔΕΙΓΜΑ



RAM 128 bytes μέγεθος μπλοκ/γραμμής 4 λέξεις, Cache 16 bytes.

Η CPU ζητά τις λέξεις 2, 7, 33. Να δείξετε την αρχική και τελική κατάσταση της cache.

ΑΝΑΛΥΣΗ ΔΙΕΥΘΥΝΣΗΣ: RAM 128 bytes άρα 7 bit διευθυνσιοδότησης

Tag, byte. Έχουμε 4 bytes ανά γραμμή, άρα το μέγεθος του πεδίου byte είναι 2 bit άρα 5 bit είναι το tag. Τι σημαίνει αυτό το 5 bit στο tag;

Η Ram περιέχει 128/4=32 μπλοκ. Τα 32 μπλοκ είναι 25, όπου 5 το μέγεθος του tag. Δηλαδή καθένα από τα 32 μπλοκ μπορεί να πάει σε οποιαδήποτε γραμμή. Τα 5 bit του tag δείχνουν απλά ότι έχω 32 μπλοκ.

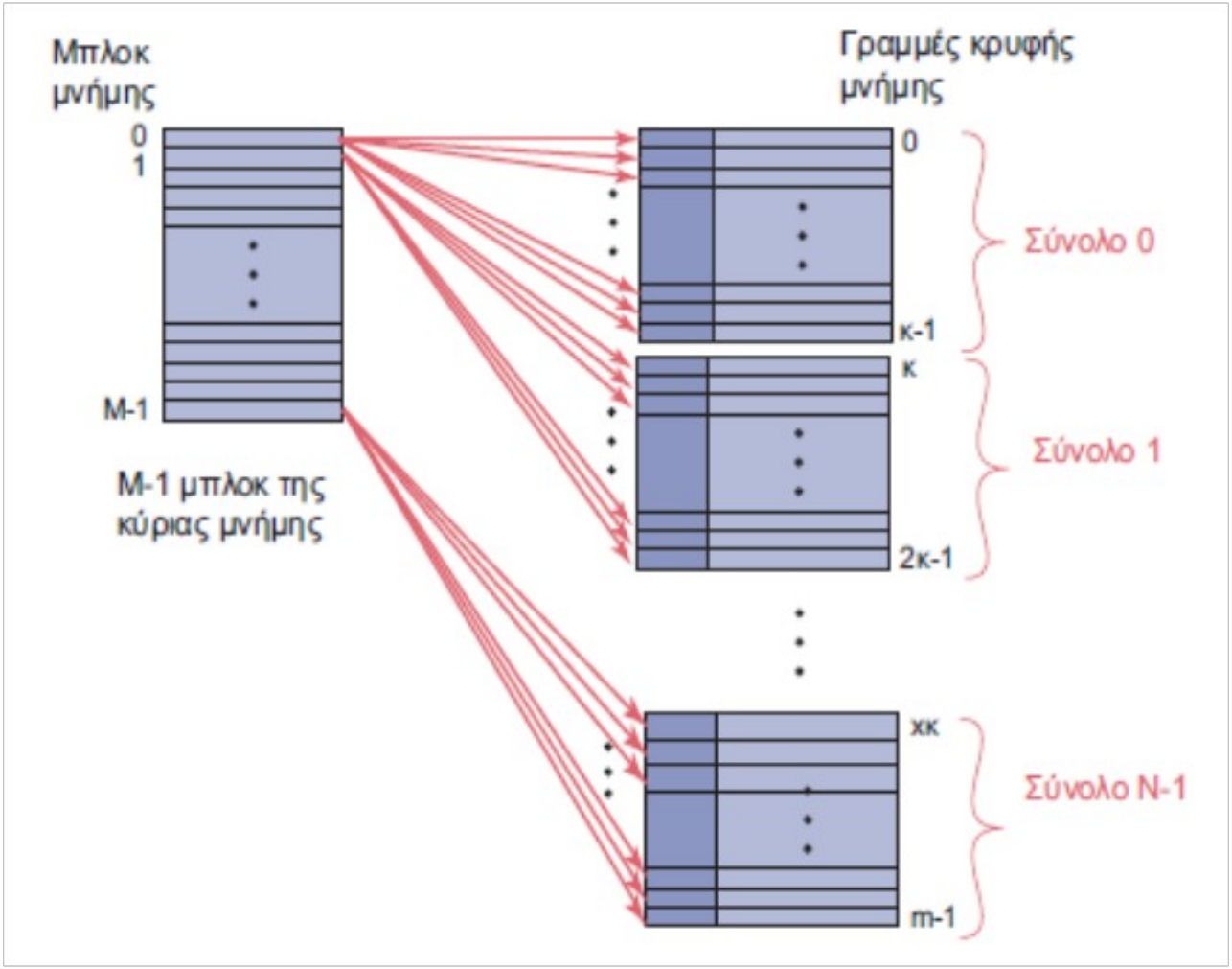
Διεύθυνση 2: 00000 10. Υπάρχει η λέξη με διεύθυνση 2 στην κρυφή μνήμη; Εξετάζουμε το tag της ζητούμενης διεύθυνσης σε σχέση με όλα τα tag της cache. Θα γίνουν 4 συγκρίσεις ταυτόχρονα, όσες οι γραμμές τις κρυφής μνήμης.

Επειδή το tag υπάρχει στη γραμμή 0, έχουμε hit, δηλαδή ολόκληρο το μπλοκ που περιέχει τη λέξη με διεύθυνση 2 είναι στη γραμμή 0 της κρυφής μνήμης. Ψάχναμε το D. Έχει γραφτεί όλο το μπλοκ που ανήκει το D, δηλαδή 8ADF.

Διεύθυνση 7: 00001 11: Εξετάζουμε το tag της ζητούμενης διεύθυνσης σε σχέση με όλα τα tag της cache. Θα γίνουν 4 συγκρίσεις ταυτόχρονα, όσες οι γραμμές τις κρυφής μνήμης. Επειδή το tag υπάρχει στη γραμμή 1, έχουμε hit, δηλαδή ολόκληρο το μπλοκ που περιέχει τη λέξη με διεύθυνση 7 είναι στη γραμμή 1 της κρυφής μνήμης. Ψάχναμε το 4. Έχει γραφτεί όλο το μπλοκ που ανήκει το 4, δηλαδή K324.

Διεύθυνση 33: 01000 01 Το tag δεν υπάρχει. Άρα έχουμε MISS. Το μπλοκ στο οποίο ανήκει η διεύθυνση 33 μπορεί να τοποθετηθεί σε οποιαδήποτε γραμμή της κρυφής μνήμης. Εδώ επιλέχθηκε τυχαία η γραμμή 2. Όμως, μπορεί να ακολουθείται κάποιος αλγόριθμος πολιτικής αντικατάστασης σελίδων. Αν πχ είναι ο FIFO, τότε το πρώτο μπλοκ που μπήκε, αντικαθίσταται, αν είναι LIFO, το τελευταίο που μπήκε αντικαθίσταται.

CACHE ΣΥΣΧΕΤΙΣΗΣ ΣΥΝΟΛΩΝ



Οι γραμμές της κρυφής μνήμης χωρίζονται σε σύνολα. Πχ αν έχω 16 γραμμές, μπορώ να τις χωρίσω σε 4 σύνολα. Κάθε σύνολο μπορεί να υποδεχτεί ένα πλήθος από μπλοκ της RAM.

Η διεύθυνση χωρίζεται σε 3 μέρη: tag Set Byte

Όσα μπλοκ έχουν ίδιο αριθμό Set τοποθετούνται (συσχετίζονται με το ίδιο Set).

Πόσα μπλοκ της RAM με ίδιο αριθμό Set μπορούν να συνυπάρχουν στην κρυφή μνήμη; Απάντηση: Όσες είναι οι γραμμές των συνόλων, ανάλογα με τον χωρισμό της κρυφής μνήμης σε σύνολα. Αν οι γραμμές κάθε συνόλου είναι k τότε, k μπλοκ με ίδιο αριθμό Set μπορούν να συνυπάρχουν. Η κρυφή μνήμη με k γραμμές ανά σύνολο λέγεται κρυφή μνήμη συσχέτισης συνόλων k δρόμων (k-way).

Πόσα μπλοκ της RAM με ίδιο αριθμό Line μπορούν να συνυπάρχουν στην κρυφή μνήμη; Απάντηση: 1

Στο παράδειγμα, στο σύνολο 0 μπορούν να τοποθετηθούν ταυτόχρονα k block. Όλα αυτά τα μπλοκ έχουν το πεδίο Set =0

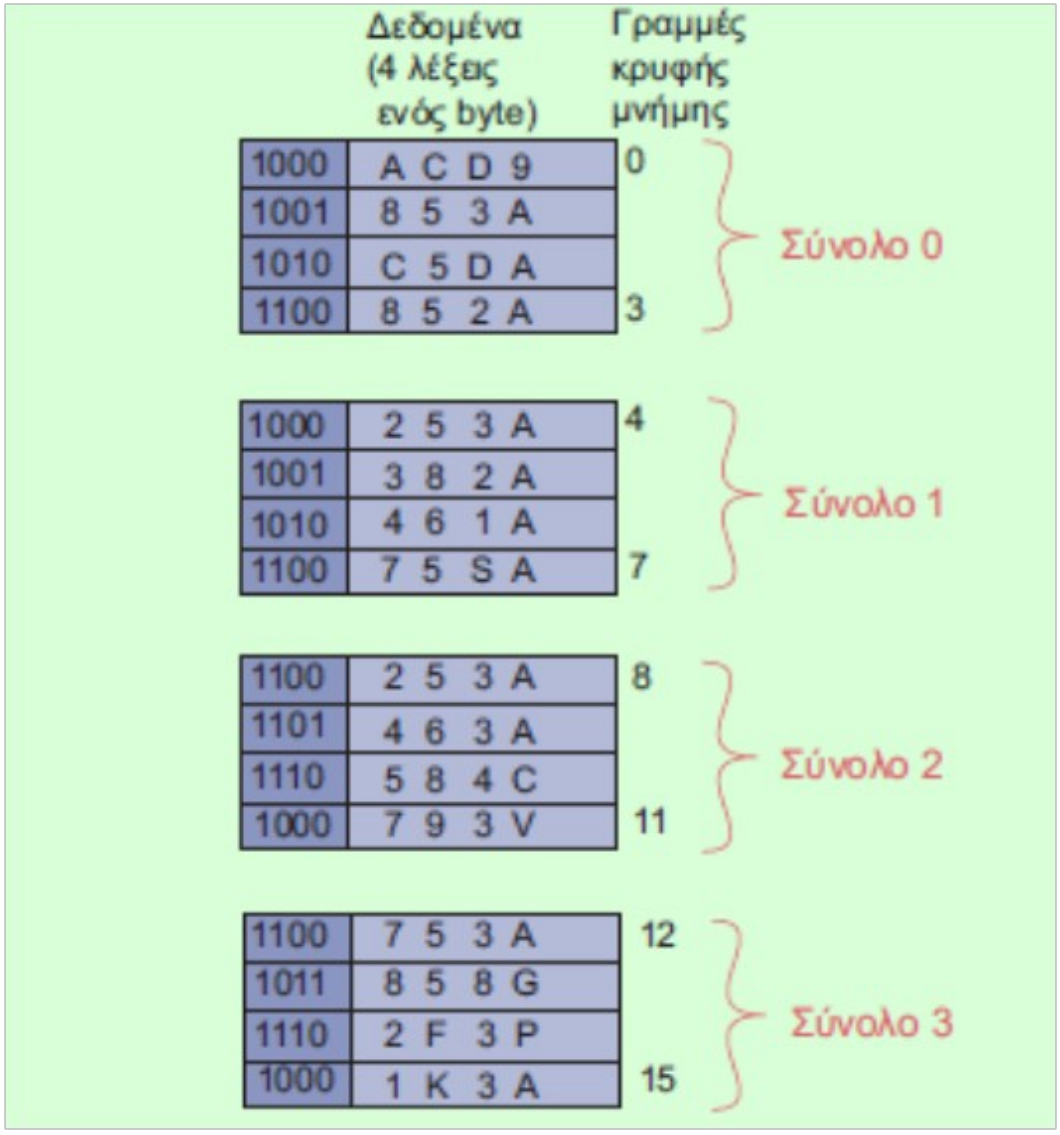
Δηλαδή τα bit του πεδίου Set σχηματίζουν τον αριθμό 0. Ομοίως , στο σύνολο 2 μπορούν να τοποθετηθούν ταυτόχρονα k block. Όλα αυτά τα μπλοκ έχουν το πεδίο Set =2

Δηλαδή τα bit του πεδίου Set σχηματίζουν τον αριθμό 2.

ΣΥΓΚΡΙΣΕΙΣ: Για να δούμε αν ένα μπλοκ βρίσκεται στην κρυφή μνήμη, ελέγχουμε το πεδίο Set. Έπειτα διαβάζουμε το tag της ζητούμενης διεύθυνσης και το συγκρίνουμε με όλα τα tag Που υπάρχουν στο σύνολο. Δηλαδή στο παράδειγμα του σχήματος, αν η τιμή Set ήταν 0, θα κάναμε k συγκρίσεις, ανάμεσα στο tag της ζητούμενης διεύθυνσης και στα k tag που υπάρχουν στο σύνολο 0. ΠΟΣΟΙ ΣΥΓΚΡΙΤΕΣ απαιτούνται για μία μνήμη συσχέτισης συνόλων; k, όπου k το πλήθος δρόμων των συνόλων.

Αν σας πουν ότι έχουμε μία κρυφή μνήμη 4-way, τότε υπάρχουν 4 συγκριτές. Ένα μπλοκ που αντιστοιχεί σε ένα σύνολο έστω Χ, Μπορεί να τοποθετηθεί σε ΟΠΟΙΑΔΗΠΟΤΕ από τις k γραμμές του Χ.

ΠΑΡΑΔΕΙΓΜΑ



RAM 256 bytes cache 64 bytes μπλοκ 4 λέξεις. Σύνολα τεσσάρων δρόμων

Α) Ανάλυση διεύθυνσης

Β) Να δώσετε τη μεγαλύτερη και τη μικρότερη διεύθυνση λέξης που σχετίζεται με το σύνολο 2

Γ) Δίνεται η αρχική κατάσταση του σχήματος.

Γ1) Ποια μπλοκ της RAM είναι αποθηκευμένα στους δρόμους 0 και 1 του συνόλου 0;

Γ2) Να βρείτε τα hit/miss για τις διευθύνσεις 8 και 129

Α) 256= 2^8, άρα διεύθυνση 8 bits.

Byte: Έχουμε 4 byte/ γραμμή ή μπλοκ είναι 2 bit

Set: Η κρυφή μνήμη είναι 4-way. Επειδή η κρυφή μνήμη περιέχει 64 byte έχει 64/4 (bytes ανά γραμμή)=16 γραμμές

Άρα επειδή είναι 4-way η cache έχουμε 16/4 = 4 σύνολα, άρα το πεδίο Set είναι 2 bit.

Δηλαδή, για να βρούμε το πλήθος των bit που απαιτεί το πεδίο SET κάνουμε τους εξής υπολογισμούς

Πλήθος γραμμών ΚΜ = Μέγεθος ΚΜ σε byte/ Πλήθος bytes ανά γραμμή

Πλήθος συνόλων ΚΜ = Πλήθος γραμμών ΚΜ/ Πλήθος δρόμων

Προφανώς το tag είναι ότι περισσεύει 4 bit.

Τι δείχνει το tag 4 bit;

Για να απαντήσω στο ερώτημα, βρίσκω το σύνολο των μπλοκ της RAM.

256/ 4 = 64 μπλοκ.

Η ΚΜ έχει 16 γραμμές και 4 σύνολα. Τα 64 μπλοκ χωρίζονται σε 4 σύνολα της ΚΜ. Άρα κάθε σύνολο της ΚΜ «φιλοξενεί» 16 μπλοκ. Άρα τα 4 bit του tag δείχνουν ότι 16 μπλοκ αντιστοιχίζονται σε κάθε σύνολο. Από τα 16 αυτά μπλοκ κάθε συνόλου, μόνο τα 4 μπορούν να συνυπάρχουν (αφού το σύνολο έχει 4 γραμμές). Π.χ το σύνολο 0, μπορεί να φιλοξενήσει 16 διαφορετικά μπλοκ (16 μπλοκ αντιστοιχούν στο σύνολο 0), αλλά μόνο 4 ταυτόχρονα (4 δρόμοι).

Β) 0000 10 00 =8 Σε ποιο μπλοκ ανήκει; 8/4=2

1111 10 11 =251 Σε ποιο μπλοκ 251/4 = 62

Πόσες διευθύνσεις λέξεων αντιστοιχίζονται στο σύνολο 2;

Το μεσαίο πεδίο είναι σταθερά 2 = 10.

Τα 4 πρώτα bit του tag δημιουργούν 16 συνδυασμούς τιμών από 0000-1111.

Τα 2 τελευταία bit του byte άλλους 4 00-11

ΣΥΝΟΛΟ: 4 x 16 =64 διευθύνσεις λέξεων αντιστοιχίζονται στο σετ 2 (και σε κάθε σετ)

Επειδή όμως αυτές οι 64 διευθύνσεις ανήκουν σε μπλοκ των 4 byte άρα τα μπλοκ αυτά είναι 64/4=16.

Επειδή η ΚΜ είναι 4-way μπορούν να συνυπάρχουν 4 μπλοκ σε κάθε σύνολο, δηλαδή 16 διευθύνσεις λέξεων.

Γ1) Στον δρόμο 0 του συνόλου 0 υπάρχει η ετικέτα 1000 και η τιμή συνόλου είναι 00 αφού αναφερόμαστε στο σύνολο 0.

Άρα, τα δεδομένα ACD9 είναι οι διευθύνσεις

1000 00 00 = 128 για το A

1000 00 01 =129 για το C

1000 00 10 = 130 για το D

1000 00 11 =131 για το 9

Οι διευθύνσεις 128-131 ανήκουν στο μπλοκ 128/4 = 32, 129/4=32 ….

Στον δρόμο 1 του συνόλου 0 υπάρχει η ετικέτα 1001 και η τιμή συνόλου είναι 00 αφού αναφερόμαστε στο σύνολο 0.

Άρα, τα δεδομένα 853A είναι οι διευθύνσεις

1001 00 00 = 144 για το 8

1001 00 01 =145 για το 5

1001 00 10 = 146 για το 3

1001 00 11 =147 για το A

Οι διευθύνσεις 144-147 ανήκουν στο μπλοκ 144/4 = 36, 145/4=36 ….

Γ2) Διεύθυνση 8: 0000 10 00

Η διεύθυνση 8 έχει τιμή Set = 10 δηλαδή το σύνολο είναι το 2. Επομένως, πηγαίνουμε και συγκρίνουμε τα 4 tags που υπάρχουν στο Set 2 με το tag της ζητούμενης διεύθυνσης.

Οι 4 συγκρίσεις που θα γίνουν είναι 0000 = 1100, 0000= 1101, 0000=1110, 0000=1000. Δηλαδή απαιτούνται 4 συγκριτές (4 way). Επειδή όλες οι συγκρίσεις δίνουν αποτέλεσμα

FALSE, έχω MISS. Πρέπει το μπλοκ 2 (η διεύθυνση 8 ανήκει στο μπλοκ 2) να αντικαταστήσει ένα από τα μπλοκ του συνόλου 2. Αν π.χ. η αντικατάσταση ήταν FIFO, θα αλλάζαμε το μπλοκ με tag 1100 και δεδομένα 253A.

Για να γίνει αντικατάσταση σε μία μνήμη συσχέτισης συνόλων πρέπει το σύνολο που δίνεται από το πεδίο SET να είναι γεμάτο και να έχουμε MISS.

129: 1000 00 01 Συγκρίνουμε με τα 4 tags του συνόλου 0.

Οι συγκρίσεις είναι: 1000=1000 (TRUE), 1000=1001, 1000=1010, 1000=1100. Άρα έχουμε hit.