# ΗΥ 232
# Οργάνωση και Σχεδίαση Υπολογιστών

## Διάλεξη 15
# Απόδοση της Ιεραρχίας Μνήμης
# Βελτιστοποίηση της απόδοσης

Νίκος Μπέλλας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

# Πόσο μεγάλη είναι μια μνήμη cache;

Έστω ότι έχουμε μια μνήμη cache σε ένα 16-bit σύστημα διευθύνσεων με τα κάτωθι χαρακτηριστικά:

- Η cache είναι direct-mapped
- Κάθε cache block έχει ένα byte
- Το block index είναι τα 4 τελευταία bit της 16-bit διεύθυνσης.

Πόσο μεγάλη είναι η cache (σε bytes);

Η μνήμη cache αποτελείται από τον πίνακα των δεδομένων, από τον πίνακα των tags και το valid bit.

Data:

4-bit index -> $2^4$ = 16 blocks

Μέγεθος του tag = 12 bits (16 bit διεύθυνση- 4 bit index)

(12 tag bits + 1 valid bit + 1 dirty bit + 8 data bits) x 16 blocks = 22 bits x 16 = 352 bits

# Μέτρηση της απόδοσης του συστήματος

Θυμάστε από προηγούμενο μάθημα:

*Χρόνος Εκτέλεσης Προγράμματος = Κύκλοι Εκτέλεσης * Περίοδος Ρολογιού*

- Οι κύκλοι εκτέλεσης περιλαμβάνουν:
  - Τους κύκλους εκτέλεσης στην CPU
    - Θεωρώντας 100% caches hit rate
    - Περιλαμβάνει τον χρόνο του cache hit
  - Τους κύκλους καθυστέρησης στις μνήμες cache από cache misses (stalls)

*Χρόνος Εκτέλεσης Προγράμματος =*
*(Κύκλοι Εκτέλεσης CPU + κύκλοι καθυστέρησης στις μνήμες) * Περίοδος Ρολογιού*

όπου

*Κύκλοι καθυστέρησης στις μνήμες = Προσπελάσεις στις μνήμες/Πρόγραμμα * Ρυθμός Αστοχίας * Ποινή αστοχίας σε κύκλους =*
*Αριθμός Αστοχιών /Πρόγραμμα * Ποινή αστοχίας σε κύκλους =*
*Εντολές /Πρόγραμμα * Αστοχία/Εντολή * Ποινή αστοχίας σε κύκλους*

Μία εντολή μπορεί να έχει maximum 2 misses (I + D-Cache)

# Μέτρηση της απόδοσης του συστήματος

Συνεπώς:

*Χρόνος Εκτέλεσης Προγράμματος =*
*(Κύκλοι Εκτέλεσης CPU + Εντολές /Πρόγραμμα * Αστοχία/Εντολή * Ποινή αστοχίας σε κύκλους )*
** Περίοδος Ρολογιού*

*Χρόνος Εκτέλεσης Προγράμματος = Εντολές /Πρόγραμμα **
*(CPIex + Αστοχία/Εντολή * Ποινή αστοχίας σε κύκλους ) * Περίοδος Ρολογιού*

# Παράδειγμα απόδοσης του συστήματος

- Έστω ένας επεξεργαστής *A* για τον οποίο ισχύουν τα εξής:
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Ποινή αστοχίας= 100 κύκλοι
  - $CPI_{ex}$ = 2
  - 36% των εκτελούμενων εντολών είναι load & stores
- Πόσο αργότερος είναι ο παραπάνω επεξεργαστής σε σχέση με κάποιον επεξεργαστή *B* που έχει τέλεια cache και τρέχει το ίδιο πρόγραμμα;
- *Χρόνος Εκτέλεσης Προγράμματος A = Εντολές /Πρόγραμμα **
  *(CPIex + Αστοχία/Εντολή * Ποινή αστοχίας σε κύκλους ) * Περίοδος Ρολογιού*
- *Χρόνος Εκτέλεσης Προγράμματος B = Εντολές /Πρόγραμμα **
  *(CPIex) * Περίοδος Ρολογιού*

Επειδή ο αριθμός των εντολών και η περίοδος του ρολογιού είναι οι ίδιες, συγκρίνουμε μόνο τα CPI.

Για τον A: 2 + (0.02 + 0.04*0.36)*100 = 5.44

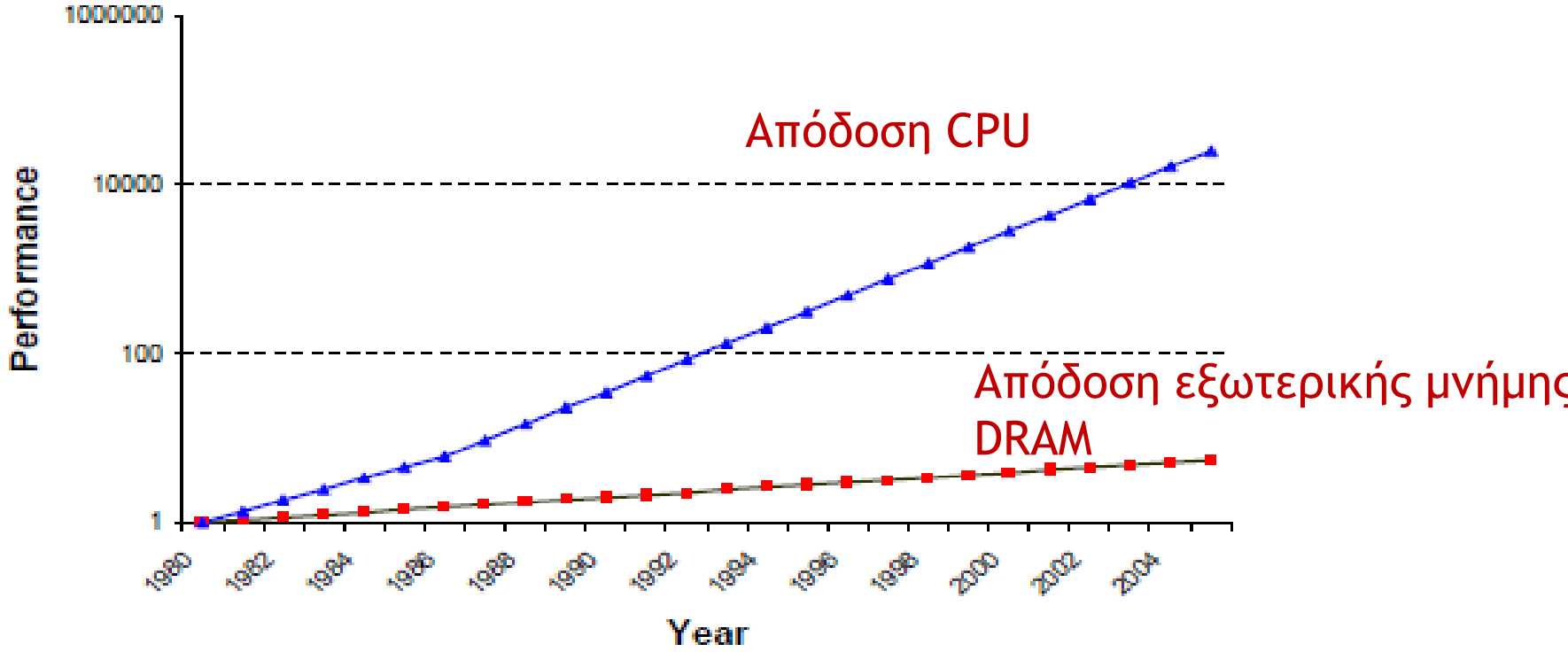
Για τον B: 2

Συνεπώς ο B είναι 5.44 / 2 = 2.72 φορές πιο γρήγορος από τον A !

# Μέσος Χρόνος Προσπέλασης Μνήμης Average Memory Access Time

- Μια σημαντική μετρική για το σύστημα ιεραρχίας μνήμης είναι ο μέσος χρόνος προσπέλασης δεδομένων στην μνήμη (average memory access time, AMAT).
  - Μέσος χρόνος προσπέλασης στην μνήμη : Average memory access time
  - Χρόνος Ευστοχίας : Hit Time
  - Ρυθμός αστοχίας: Miss Rate
  - Ποινή αστοχίας : Miss Penalty
  - *AMAT = Χρόνος ευστοχίας + Ρυθμός Αστοχίας * Ποινή Αστοχίας σε Κύκλους = Hit Time (L1) + Miss Rate * Miss Penalty*
- Παράδειγμα:
  - CPU με περίοδο ρολογιού 1ns clock, hit time = 1 κύκλος, miss penalty = 20 κύκλοι, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 κύκλοι χρειάζονται ανά εντολή ΜΟΝΟ για να διαβάσουμε την εντολή από την μνήμη

# Συμπεράσματα από την μελέτη της απόδοσης της μνήμης




- Αν και η απόδοση της κύριας μνήμης DRAM αυξάνεται με τα χρόνια, εν' τούτοις αυξάνεται πολύ πιο αργά σε σχέση με την απόδοση του επεξεργαστή.
- Αυτό σημαίνει ότι η ποινή αστοχίας από μερικούς κύκλους την δεκαετία του '80, έχει φτάσει να είναι εκατοντάδες κύκλους τα τελευταία χρόνια.
- Η εξίσωση για το AMAT, μας δείχνει την σημασία να μειωθεί ο *Ρυθμός Αστοχίας* και *Ποινή Αστοχίας σε Κύκλους*

# Οργάνωση της μνήμης cache

- Από εδώ και πέρα θα εξετάσουμε τρόπους μείωσης αυτών των δύο παραγόντων (miss rate, miss penalty) μέσω διαφορετικών τεχνικών οργάνωσης της cache

- Μέχρι στιγμής έχουμε εξετάσει μια απλή οργάνωση της cache, δηλ. θεωρούμε cache άμεσης απεικόνισης (direct mapped) και με μέγεθος του cache block ίσο με ένα byte.
- Ας δούμε σήμερα πως μπορούμε να επεκτείνουμε την αρχιτεκτονική της μνήμης cache για να βελτιώσουμε την απόδοσή της (να μειώσουμε τον ρυθμό αστοχίας, miss rate).
- Υπάρχουν 2 βασικοί τρόποι
  - Βελτίωση της χωρικής τοπικότητας
  - Βελτίωση της χρονικής τοπικότητας

# Βελτίωση της χωρικής τοπικότητας

- Η μνήμη cache με 1 byte block δεν εκμεταλλεύεται την χωρική τοπικότητα (spatial locality) των δεδομένων.
- Όχι μόνο αυτό το byte, αλλά και τα άλλα δίπλα του μπορεί να προσπελαστούν σύντομα από την CPU

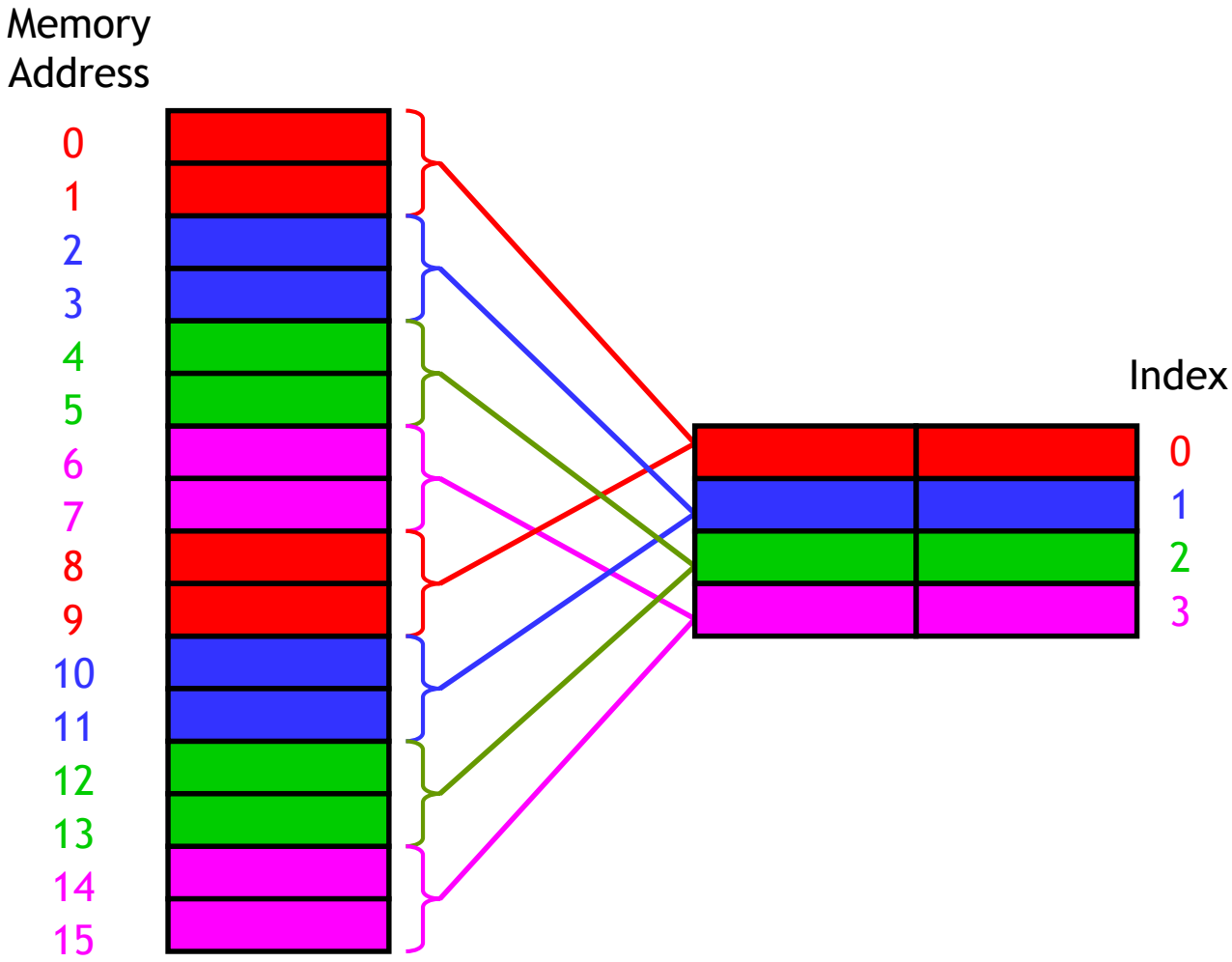
# Βελτίωση της χωρικής τοπικότητας

- Ας κάνουμε το cache block να είναι μεγαλύτερο του ενός byte.

- Παράδειγμα block των 2 bytes

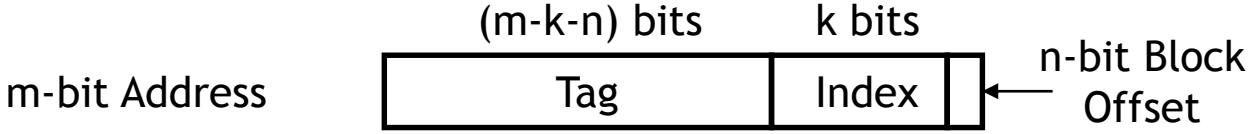
- Εάν θέλουμε να προσπελάσουμε την διεύθυνση 12 της μνήμης, η cache θα διαβάσει από την μνήμη την 12 και την 13

- Αυτός είναι και ο ορισμός των cache blocks: το μέγεθος των δεδομένων που μεταφέρονται από την κύρια μνήμη όταν έχουμε cache miss.



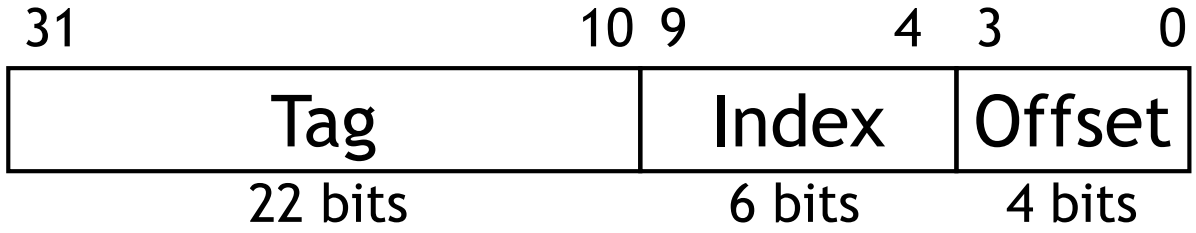
# Πως βρίσκουμε διευθύνσεις στην cache

- Θεωρείστε μια μνήμη cache direct-mapped με $2^k$ blocks, και κάθε block να έχει $2^n$ bytes. Θεωρείστε επίσης μια *m-bit* διεύθυνση.
- Μπορούμε να καθορίσουμε που τοποθετείται κάθε byte της κυρίας μνήμης στην cache εξετάζοντας την διεύθυνση και τα πεδία από τα οποία αποτελείται.
  - *k* bits από την διεύθυνση επιλέγουν ένα από τα $2^k$ cache blocks.
  - Τα *n* LS bits είναι το block offset και καθορίζει ποιο από τα $2^n$ bytes στο cache block θα προσπελασθούν.
- Τα υπόλοιπα *(m-k-n)* bits της m-bit διεύθυνσης είναι το tag.

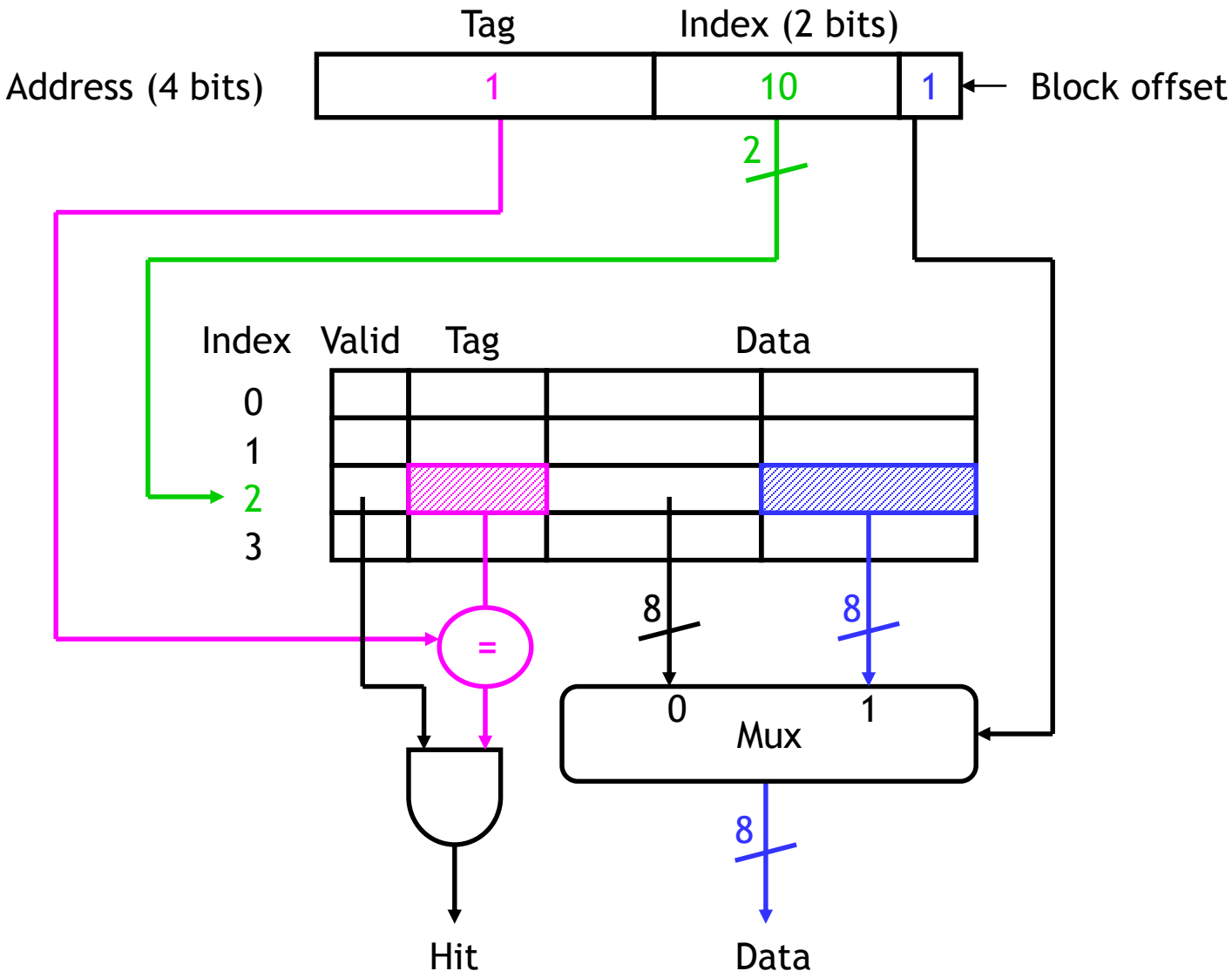


# Παράδειγμα 1

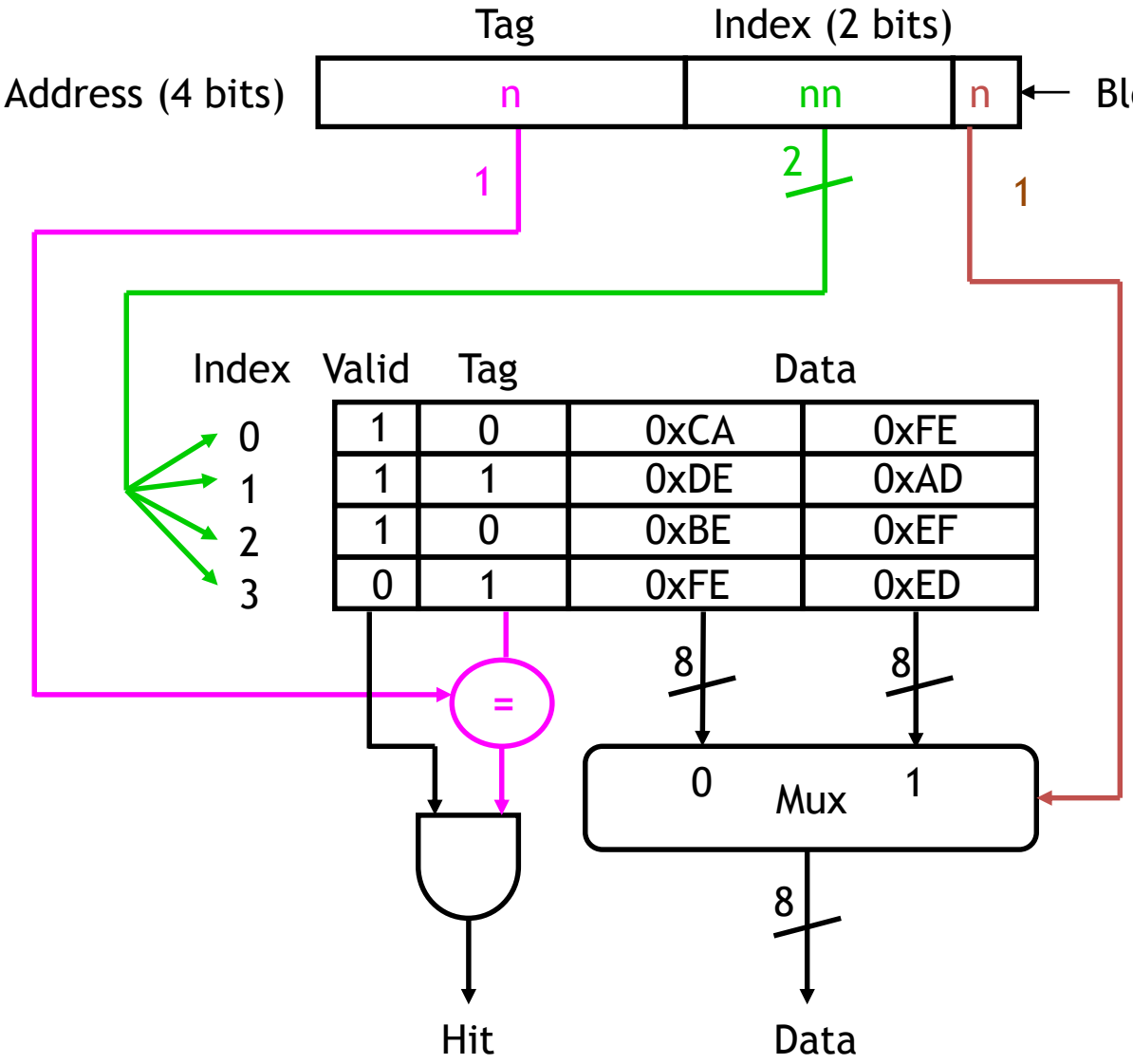
- Έστω μια direct-mapped μνήμη cache με 64 blocks, 16 bytes/block
  - Σε ποιο block θα αποθηκευθούν τα δεδομένα της διεύθυνσης 1200?
- 1200 = **00..001 001011 0000**
- Block number = 11
- Block offset = 0



# Μικρο-αρχιτεκτονική cache



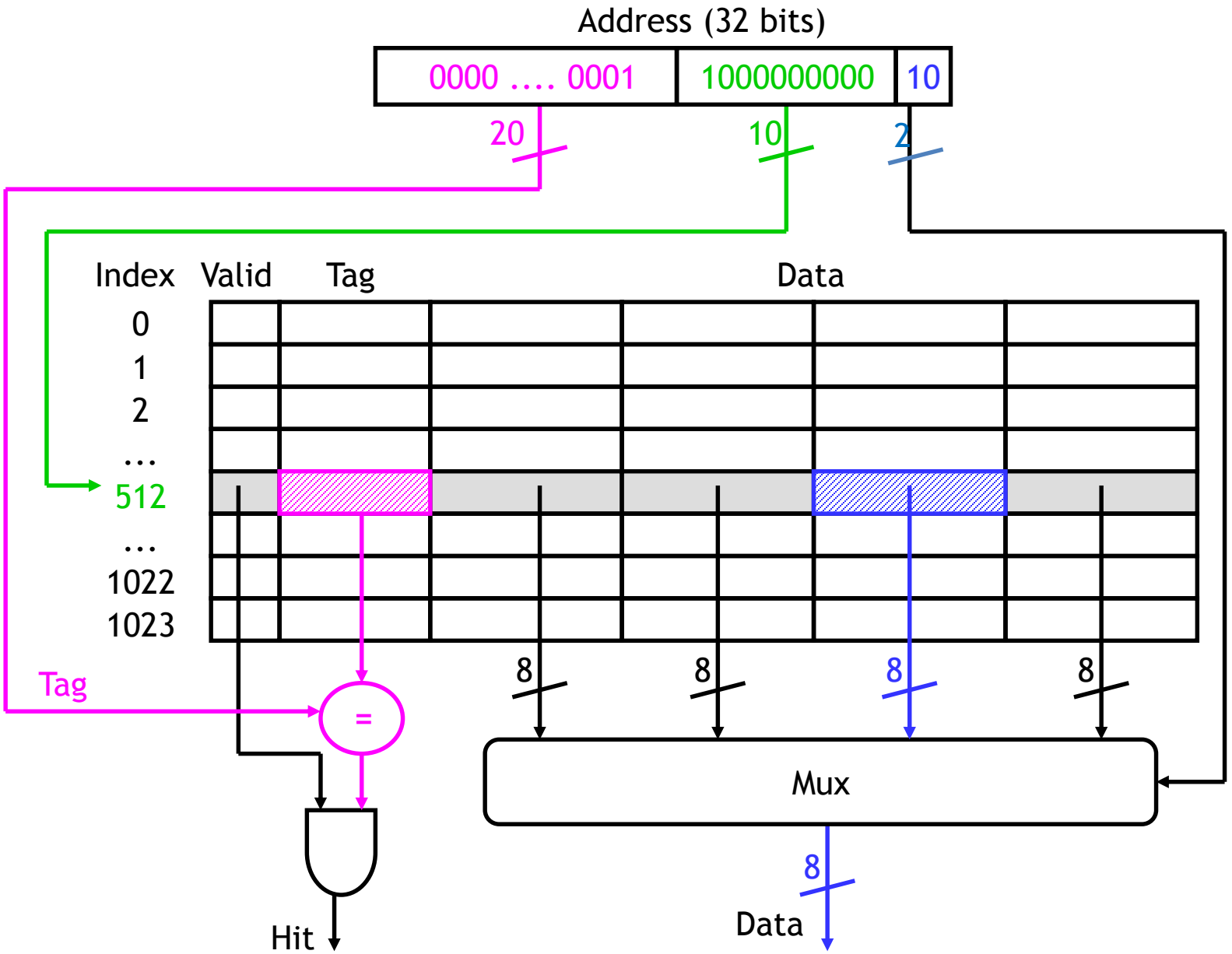
# Παράδειγμα 2




Ποιο byte διαβάζεται για κάθε μία από τις παρακάτω διευθύνσεις;

- 1010 (0xDE)
- 1110 (miss, invalid)
- 0001 (0xFE)
- 1101 (miss, tag mismatch

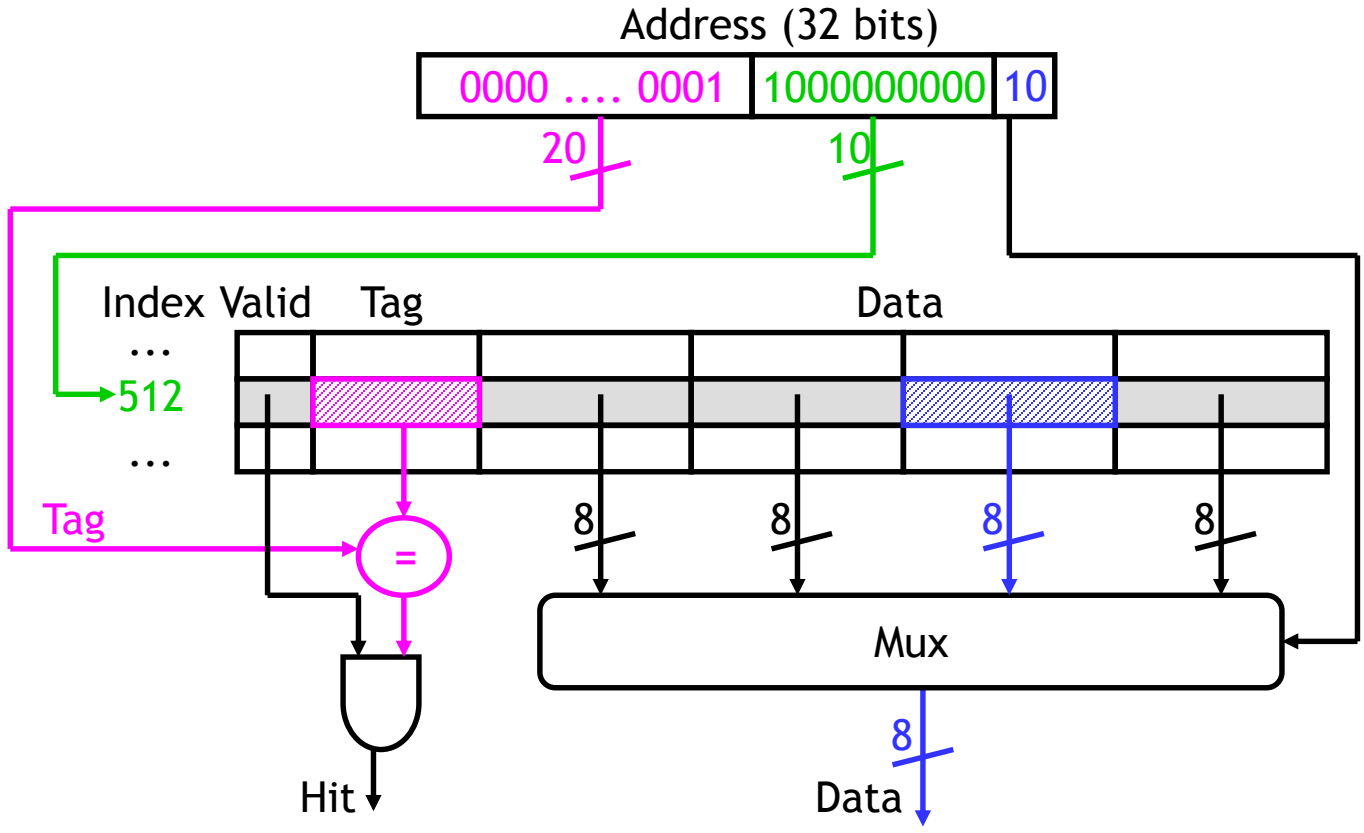
# Παράδειγμα 3



# Λειτουργία cache block

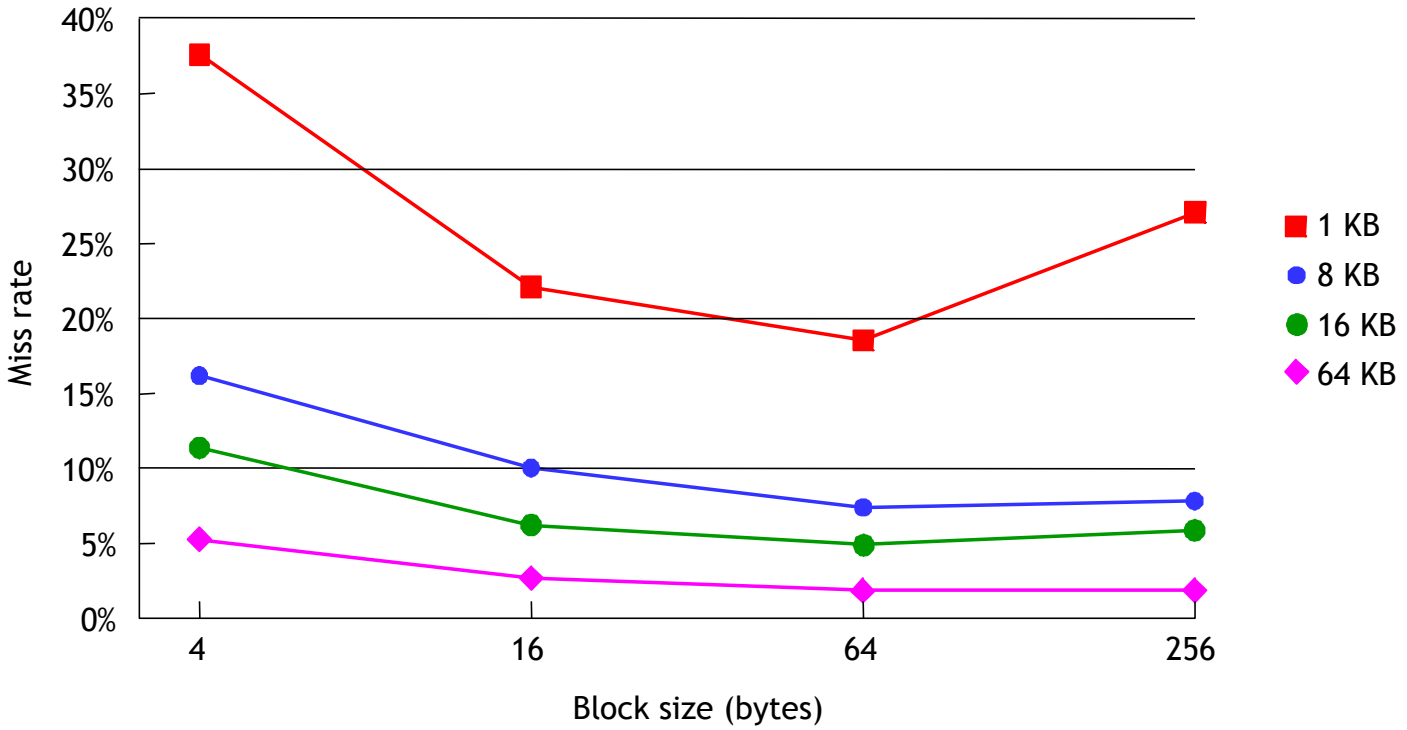
- Τα υπόλοιπα 3 bytes του cache block προέρχονται από τις γειτονικές διευθύνσεις. Δηλ. τις 3 διευθύνσεις με το ίδιο index (1000000000) και το ίδιο tag (00…01).

- Στο παράδειγμά μας, h CPU ζητάει να προσπελάσει την διεύθυνση 6146 (00..01 1000000000 10).
- Ο cache controller θα φέρει στην cache όλο το block (6144-6147)



# Μέγεθος του block και ρυθμός αστοχίας

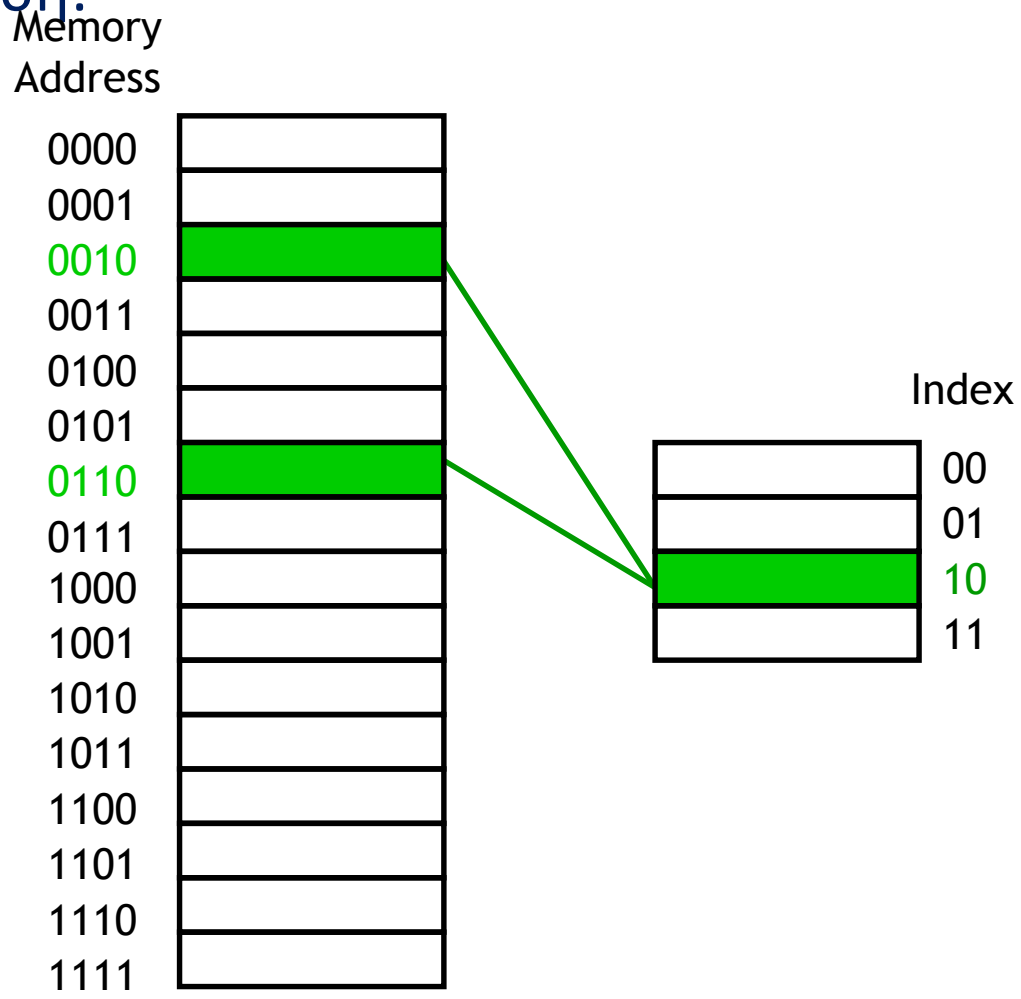
- Γραφική παράσταση που δείχνει τον ρυθμό αστοχίας συναρτήσει του μεγέθους του block για μνήμες cache διαφορετικών μεγεθών
  - Μικρά blocks δεν εκμεταλλεύονται την χωρική τοπικότητα όπως έχουμε πει.
  - Αλλά και πολύ μεγάλα blocks δημιουργούν το πρόβλημα ότι δεν υπάρχουν πολλά blocks στην cache, με αποτέλεσμα να δημιουργούνται επιπλέον αστοχίες.



# Βελτίωση της χρονικής τοπικότητας

- Οι direct-mapped caches είναι εύκολες να υλοποιηθούν σε hardware, και είναι απλό να υπολογίσουμε το (μοναδικό) block που αντιστοιχεί σε κάθε διεύθυνση.

- Πλην όμως είναι περιοριστικές στην λειτουργία τους γιατί κάθε διεύθυνση της μνήμης μπορεί να αποθηκευθεί σε ένα και μοναδικό σημείο της cache.
- Για παράδειγμα, εάν ένα πρόγραμμα προσπελαύνει τις διευθύνσεις 2, 6, 2, 6, 2, ..., κάθε προσπέλαση θα είναι miss, αν και η cache είναι κατά τα άλλα άδεια!
- Θα πρέπει λοιπόν να κάνουμε κάτι για να αλλάξουμε τον περιορισμό της μοναδικής απεικόνισης της direct-mapped cache.

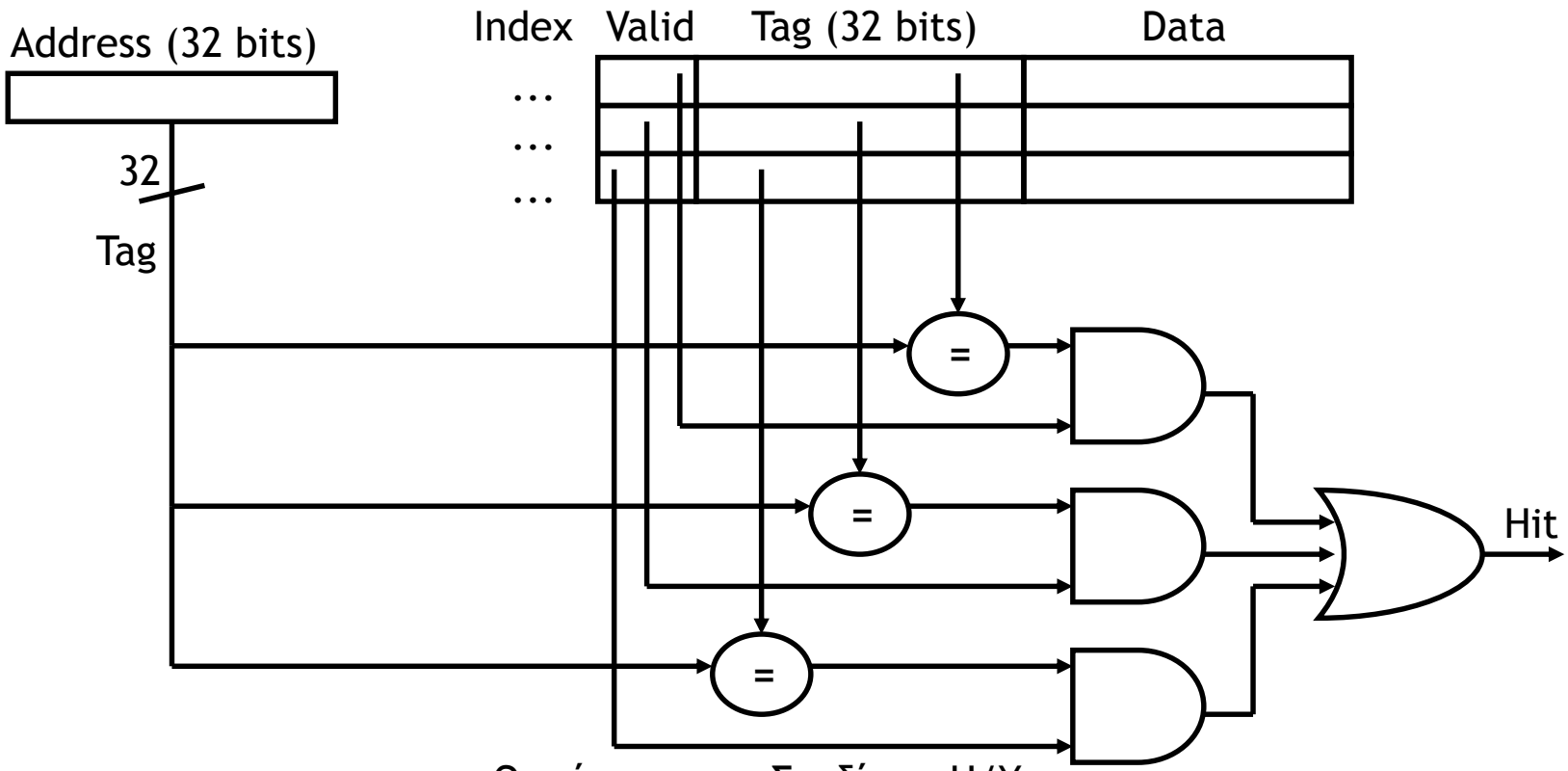


# Πλήρως συσχετιστική μνήμη cache (fully-associative cache)

- Η πλήρως συσχετιστική (fully-associative) cache ακολουθεί την ακριβώς αντίθετη στρατηγική: τα δεδομένα από την μνήμη επιτρέπεται να αποθηκευθούν σε ΟΠΟΙΟΔΗΠΟΤΕ block της cache.
  - Με αυτόν τον τρόπο δεν θα υπάρχει ποτέ σύγκρουση μεταξύ δύο η περισσότερων διευθύνσεων που αποθηκεύονται στο ίδιο cache block.
- Στο προηγούμενο παράδειγμα, η διεύθυνση 2 μπορεί να τοποθετηθεί στο block 2, και η διεύθυνση 6 στο block 3. Από εκεί και πέρα θα έχουμε συνέχεια hits και όχι misses.
- Το ενδιαφέρον ερώτημα που θα δούμε σύντομα είναι ποιο block αντικαθιστούμε όταν γεμίσει η cache.

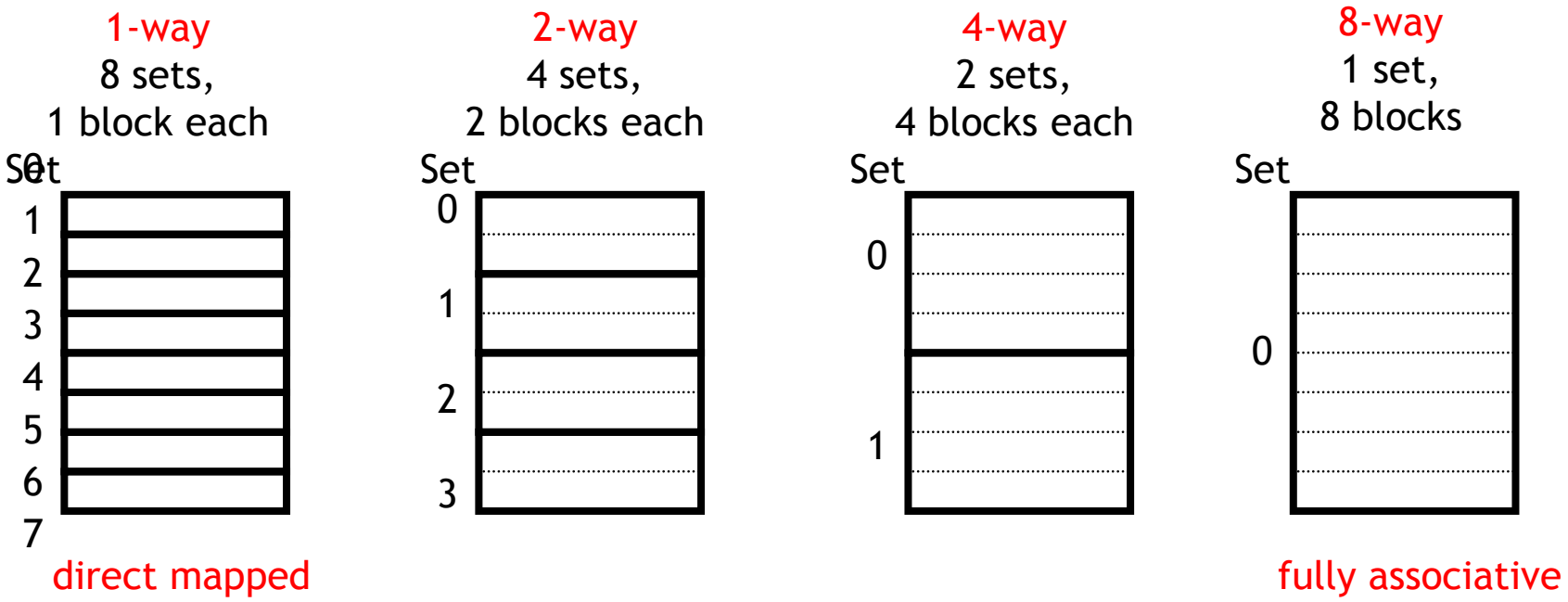
# Fully-associative cache

- Οι fully-associative caches είναι όμως ακριβές γιατί πιάνει πολύ χρόνο το να διαπιστώσουμε εάν έχουμε Hit OR Miss.
  - Δεν υπάρχει πεδίο index στην διεύθυνση
  - Τα δεδομένα μπορούν να είναι παντού στην cache, το οποίο σημαίνει ότι θα πρέπει να ελέγξουμε τα tags κάθε block για να δούμε εάν δεδομένα είναι στην cache.
  - Αυτό χρειάζεται πολλούς συγκριτές και παίρνει πολύ χρόνο.

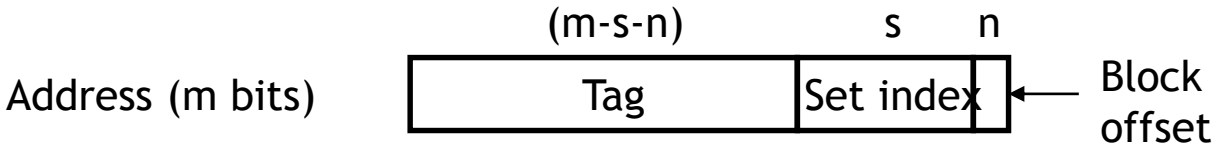


# Συσχετιστική μνήμη συνόλου (set-associative cache)

- Μια ενδιάμεση λύση είναι η συσχετιστική μνήμη συνόλου (set-associative cache).
  - Η cache χωρίζεται σε groups από blocks, που ονομάζονται σύνολα (sets).
  - Κάθε διεύθυνση μνήμης μπορεί να τοποθετηθεί σε ένα και μόνο σύνολο στην cache, αλλά σε οποιοδήποτε block μέσα στο σύνολο.
  - Εάν ένα σύνολο αποτελείται από X blocks, η cache ονομάζεται συσχετιστική μνήμη συνόλου X δρόμων (X-way set associative cache).



# Πως βρίσκουμε διευθύνσεις στην set associative cache




- Διαίρεση διεύθυνσης μνήμης με $2^s$ σύνολα, και μέγεθος block $2^n$ bytes

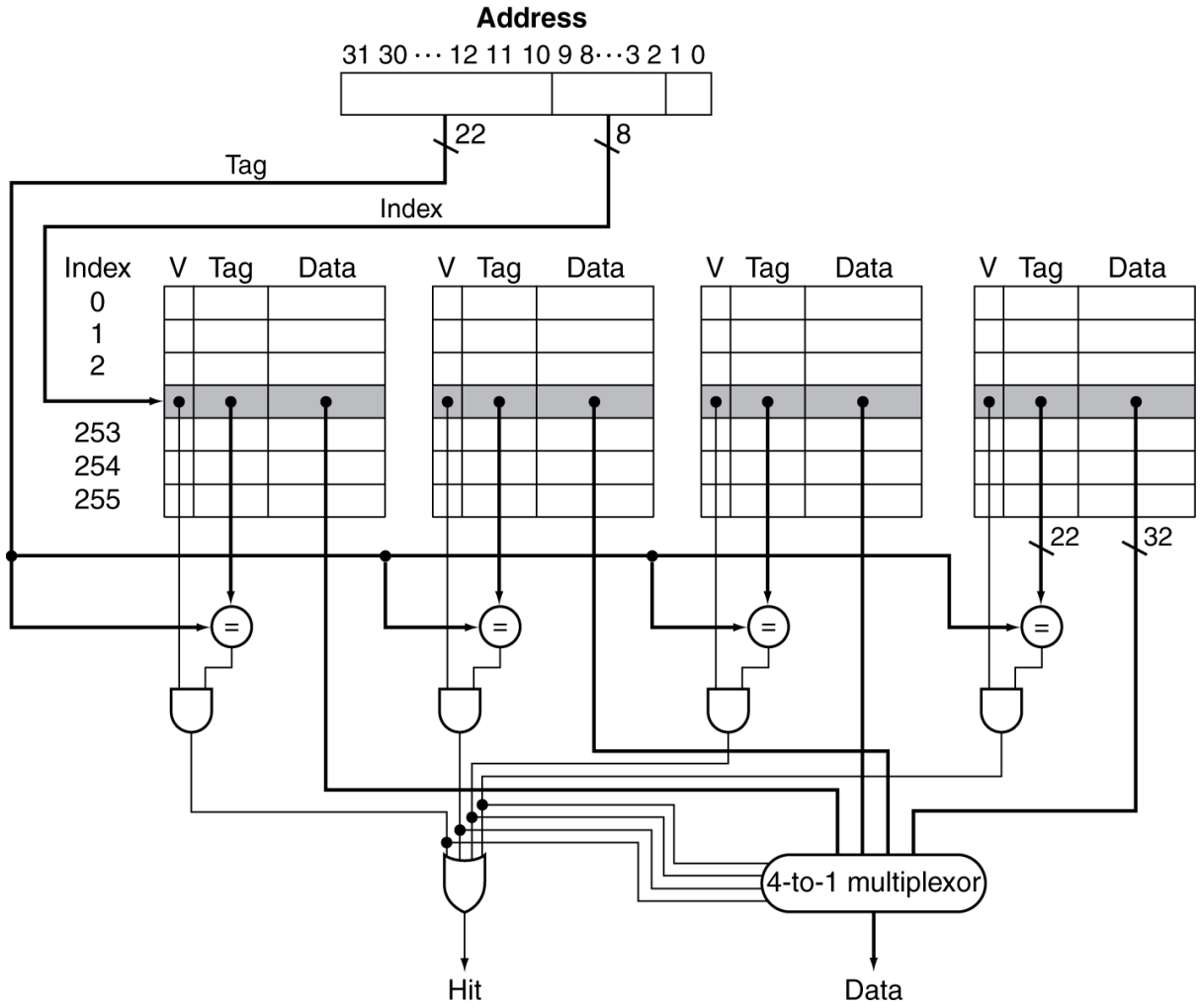
Block Offset = Memory Address mod $2^n$

Block Address = Memory Address / $2^n$

Set Index = Block Address mod $2^s$

# Υλοποίηση συσχετιστική μνήμη συνόλου

4-way set associative
256 σύνολα
1 cache block έχει 4 bytes
Μέγεθος cache 256*4*4 = 4KB



# Παράδειγμα Associativity

- Έστω μια μνήμη cache με 4 blocks
  - Direct mapped, 2-way set associative, fully associative
  - Αλληλουχία διευθύνσεων που προσπελαύνουμε: 0, 8, 0, 6, 8
- Direct mapped

| Block address | Cache index | Hit/miss | Cache content after access | | | |
|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 |
| 0 | 0 | miss | **Mem[0]** | | | |
| 8 | 0 | miss | **Mem[8]** | | | |
| 0 | 0 | miss | **Mem[0]** | | | |
| 6 | 2 | miss | Mem[0] | | **Mem[6]** | |
| 8 | 0 | miss | **Mem[8]** | | Mem[6] | |

# Παράδειγμα Associativity

- 2-way set associative

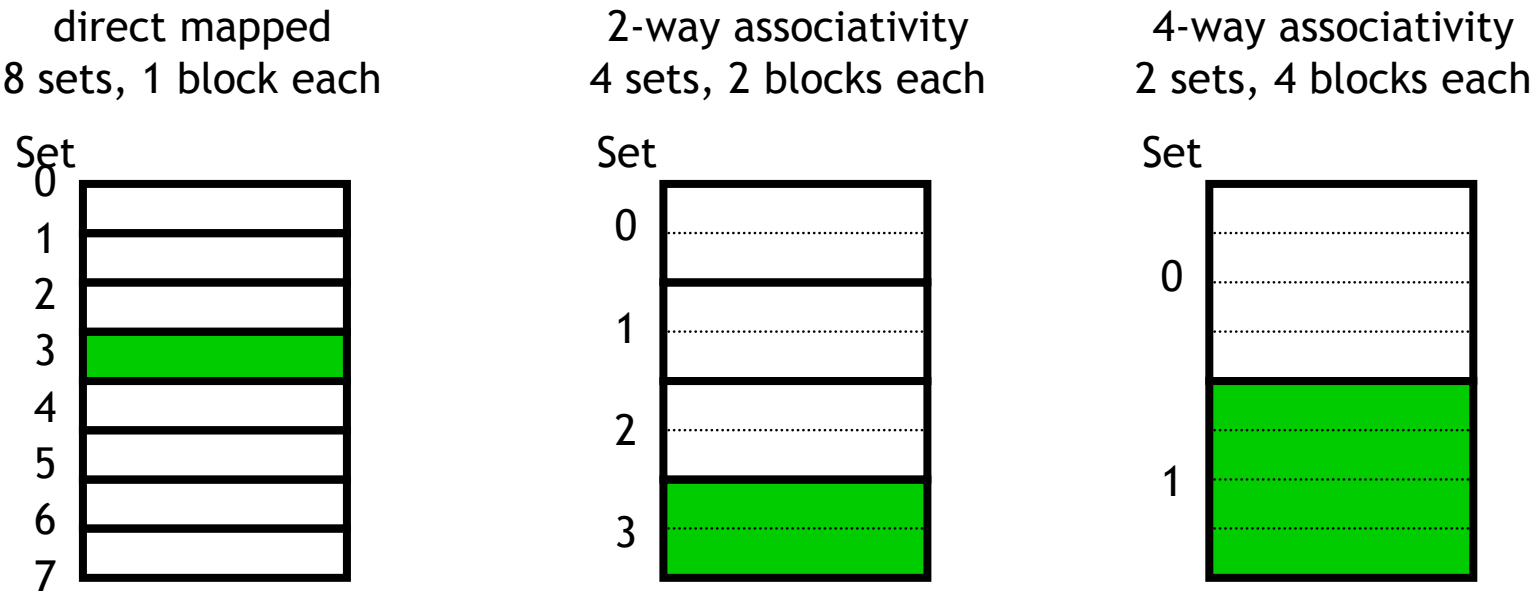
| Block address | Cache index | Hit/miss | Cache content after access | | | |
|---|---|---|---|---|---|---|
| | | | Set 0 | | Set 1 | |
| 0 | 0 | miss | **Mem[0]** | | | |
| 8 | 0 | miss | Mem[0] | **Mem[8]** | | |
| 0 | 0 | hit | **Mem[0]** | Mem[8] | | |
| 6 | 0 | miss | Mem[0] | **Mem[6]** | | |
| 8 | 0 | miss | **Mem[8]** | Mem[6] | | |

- Fully associative

| Block address | | Hit/miss | Cache content after access | | | |
|---|---|---|---|---|---|---|
| 0 | | miss | **Mem[0]** | | | |
| 8 | | miss | Mem[0] | **Mem[8]** | | |
| 0 | | hit | **Mem[0]** | Mem[8] | | |
| 6 | | miss | Mem[0] | Mem[8] | **Mem[6]** | |
| 8 | | hit | Mem[0] | **Mem[8]** | Mem[6] | |

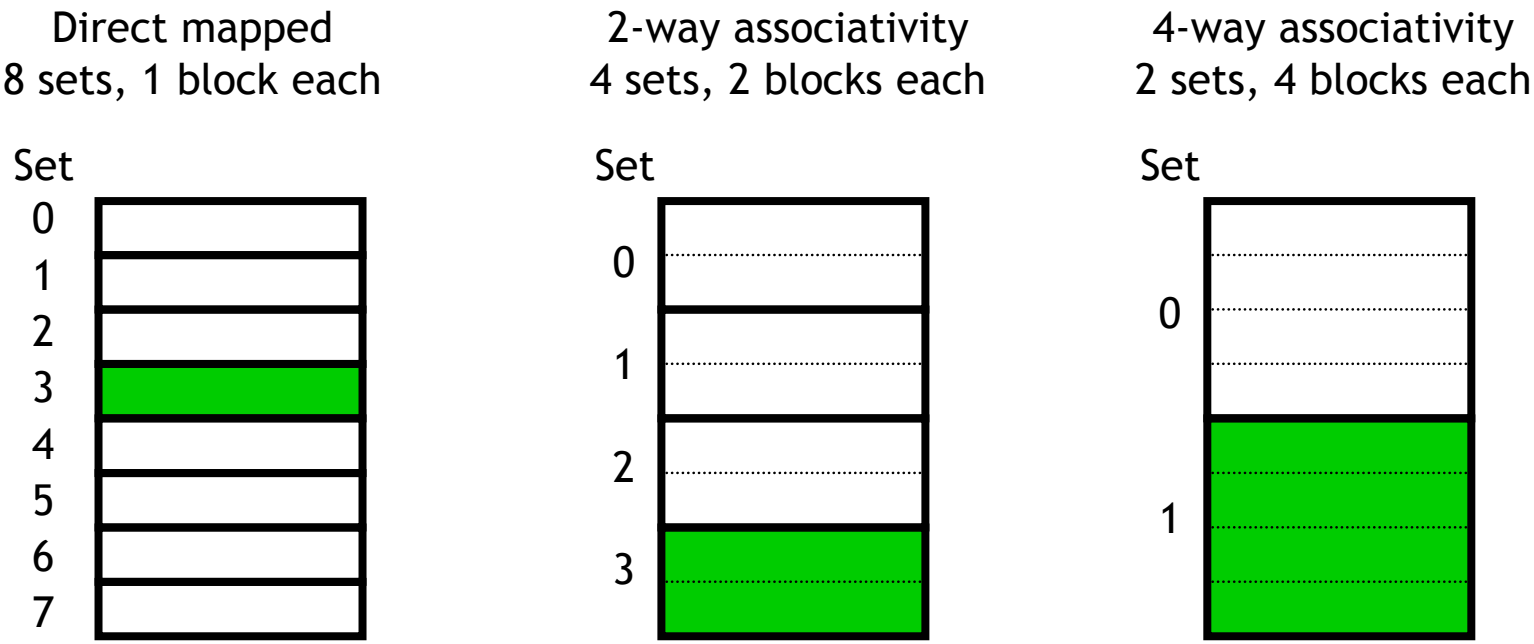
# Παράδειγμα Associativity

- Υποθέστε ότι έχουμε μια μνήμη cache με 8 blocks, 16 bytes/blocks. Σε ποιο cache block θα τοποθετηθούν τα δεδομένα που βρίσκονται στην διεύθυνση 6195, για κάθε μία από τις 3 περιπτώσεις οργάνωσης της cache;
- Το 6195 στο δυαδικό σύστημα είναι 00...01100000110011.
- Κάθε block έχει 16 bytes, άρα τα 4 LSB είναι το Block offset.
- Η direct mapped cache έχει 8 sets, και άρα τα επόμενα 3 bits (011) είναι ο set index.
  Η 2-way set associative cache έχει 4 sets, και άρα τα επόμενα 2 bits (11) είναι ο set index.
  Η 4-way set associative cache έχει 2 sets, και άρα το επόμενα 1 bit (1) είναι ο set index.
- Τα δεδομένα μπορούν να μπουν σε κάθε block από τα πράσινα στην cache.



# Αντικατάσταση blocks

- Εάν δεν υπάρχουν άδεια blocks σε ένα σύνολο (set) όταν προσπαθούμε να φέρουμε ένα νέο block στην cache, ο ελεγκτής της cache (cache controller) θα αντικαταστήσει το block που έχει να χρησιμοποιηθεί τον περισσότερο χρόνο (Least Recently Used, LRU).
  - Αυτός είναι ο πιο συνηθισμένος αλγόριθμος, αλλά δεν είναι και ο μόνος

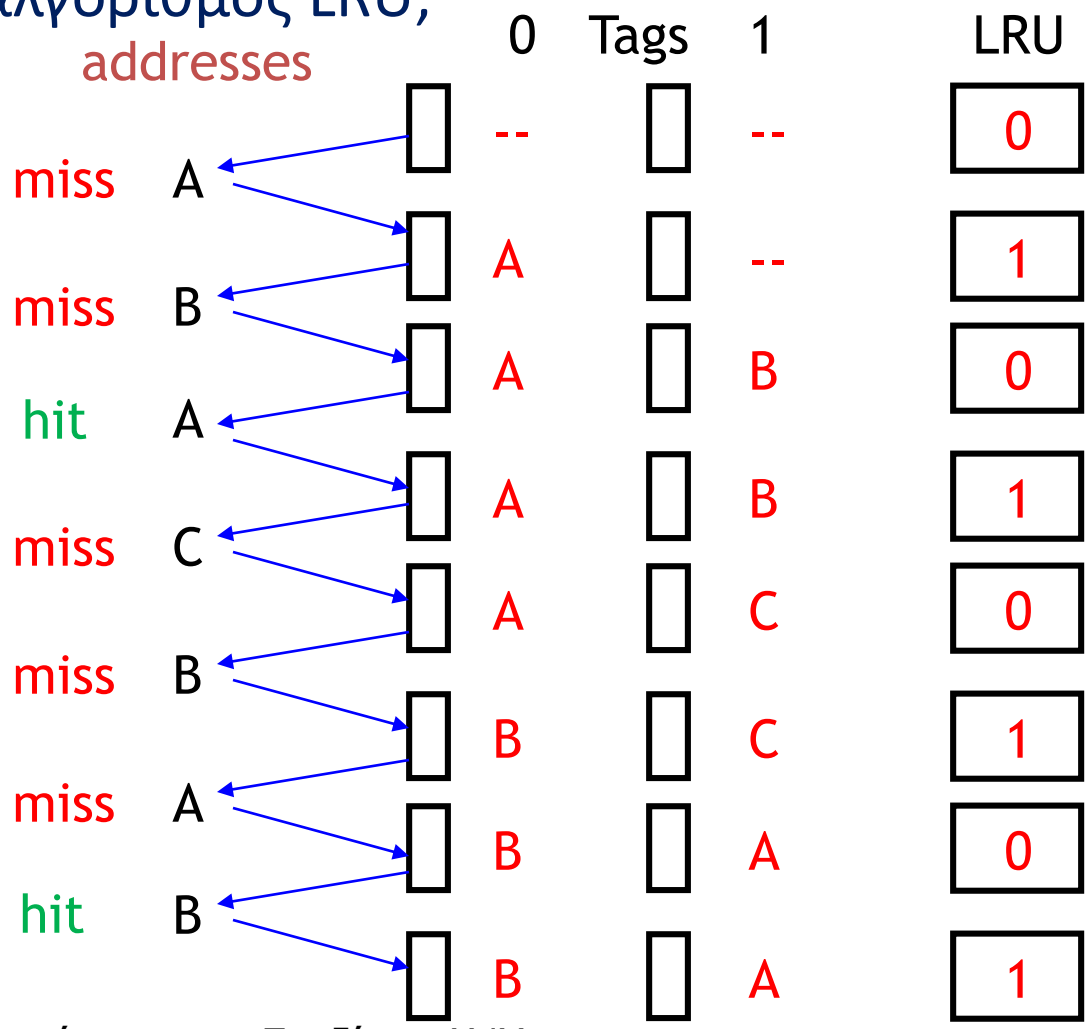


# Παράδειγμα αλγόριθμου LRU

- Υποθέστε ότι έχουμε μια fully-associative cache με μόνο 2 blocks, και μια σειρά από διευθύνσεις από την CPU
  - Πως λειτουργεί ο αλγόριθμος LRU;

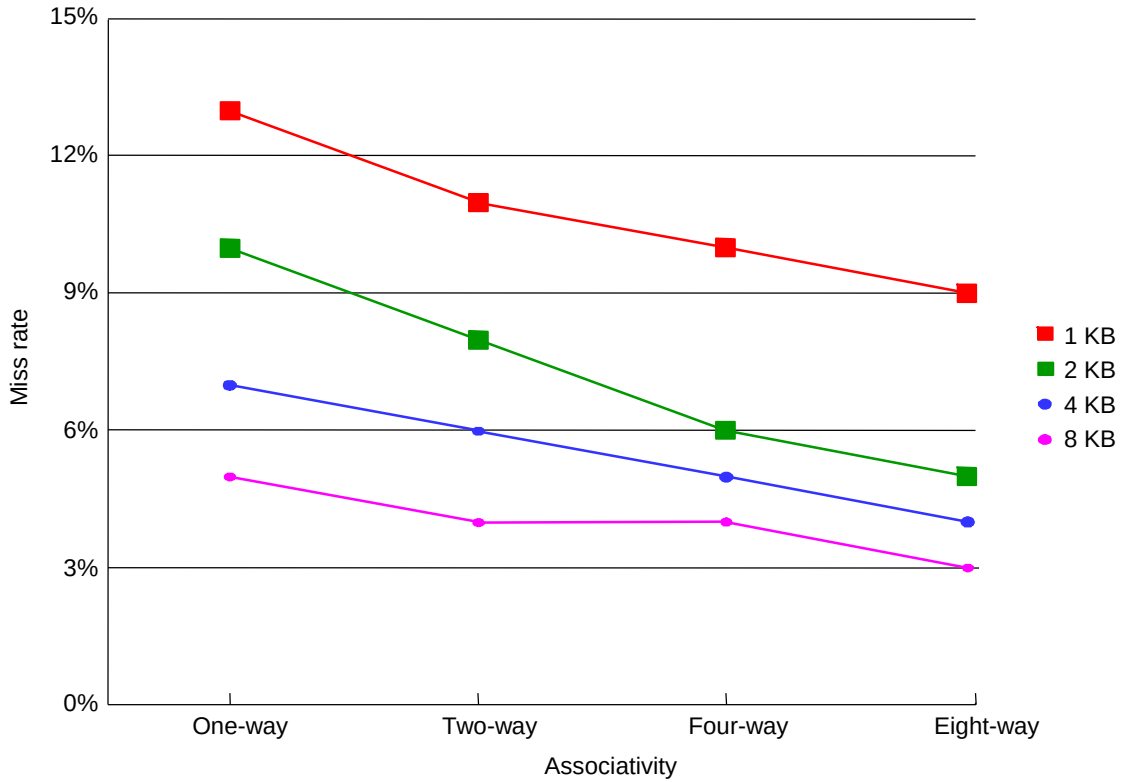
Αντικαθιστούμε το LRU block, όταν υπάρχει miss.

Ανανεώνουμε το LRU, όταν υπάρχει hit.



# Associativity και ρυθμός αστοχίας

- Μεγαλύτερη cache και μεγαλύτερος βαθμός associativity βελτιώνουν την ευστοχία της cache(δηλ. μειώνουν το miss rate)
- Όμως μεγαλύτερος βαθμός associativity απαιτεί και μεγαλύτερο χρόνο εκτέλεσης. Άρα είναι πιθανόν να μειώνει και το Hit Time (=bad!!)



# Cache organizations (II)

- Είδαμε δύο βασικές τεχνικές μείωσης του ρυθμού αστοχίας
- Πως μπορούμε να μειώσουμε την ποινή αστοχίας (miss penalty);

# Μνήμες cache πολλαπλών επιπέδων

- Μπορούμε να μειώσουμε την ποινή αστοχίας με το τοποθετήσουμε ακόμα μία μνήμη cache για να εξυπηρετεί όλες τις αστοχίες που προέρχονται από την αρχική μας cache.
  - Η αρχική cache είναι μικρή (L1 Cache) και γρήγορη
  - Μία L1 cache για εντολές (I-Cache), και μία για δεδομένα (D-Cache)
- Η L2 Cache είναι μεγαλύτερη και πιο αργή από την L1 Cache.
  - Αλλά είναι πολύ πιο γρήγορη από την κύρια μνήμη
- Τα περισσότερα επεξεργαστικά συστήματα έχουν και τρίτο επίπεδο μνήμης cache (L3 Cache)
  - Η L3 Cache εξυπηρετεί όλες τις αστοχίες της L2 Cache
  - Οι αστοχίες της L3 Cache εξυπηρετούνται από την κύρια μνήμη
  - Δεν υπάρχει L4 Cache (!) σε κανένα σύστημα από ότι ξέρω.
- Η L2 και L3 Caches είναι unified, δηλ. αποθηκεύουν και εντολές και δεδομένα στον ίδιο χώρο.
  - Επίσης έχουν μεγαλύτερο βαθμό συσχετιστικότητας (associativity)

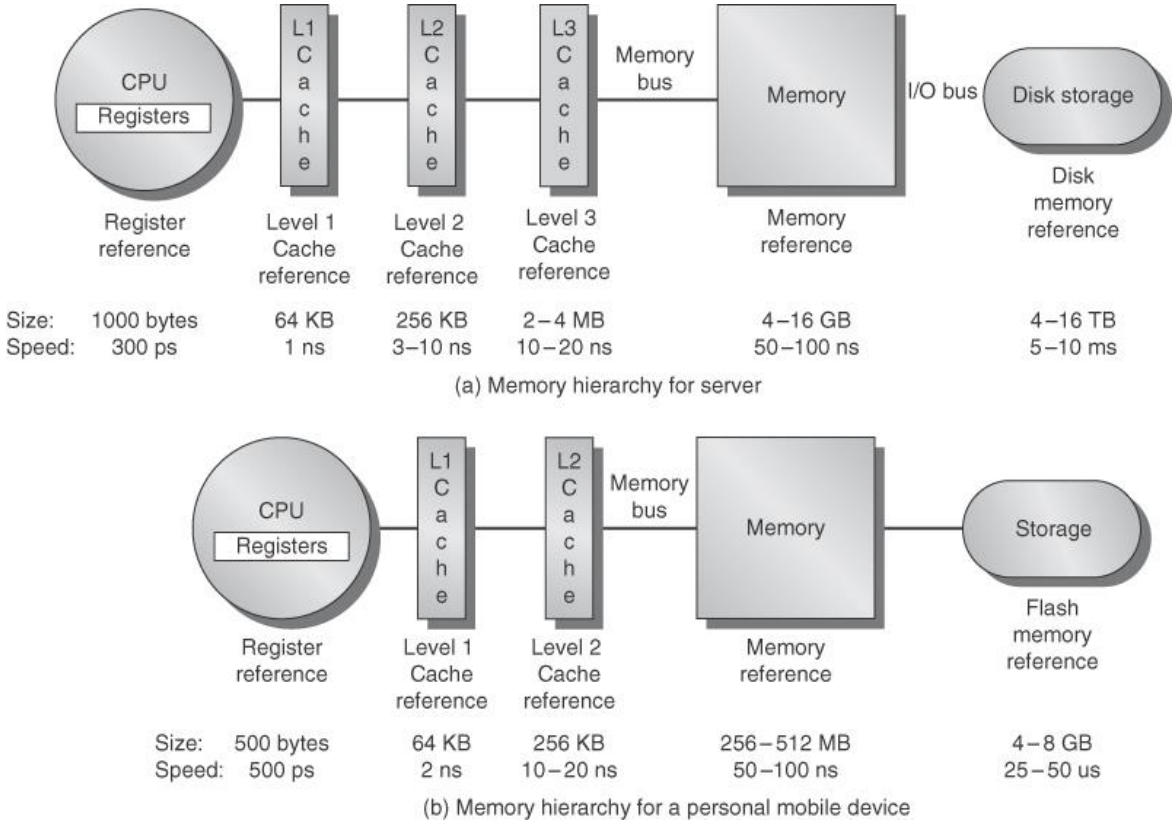
# Μνήμες cache πολλαπλών επιπέδων

Για την L1 Cache το σημαντικό είναι το χαμηλό Hit time. Πρέπει δηλ. να μπορούμε να τις προσπελαύνουμε σε 1-2 κύκλους ρολογιού

- Έχουν μικρό μέγεθος (16-64 KB) και είναι συνήθως direct-mapped.

Για τις L2 και L3 Cache το σημαντικό είναι να έχουν χαμηλό miss rate, για να αποφύγουμε προσπέλαση στην κύρια μνήμη

- Έχουν μεγάλο μέγεθος (L2 Cache ~ 256KB, L3 Cache 2-3 MB)
- Έχουν μεγάλα cache blocks
- Έχουν υψηλό associativity.



# Παράδειγμα μνημών cache πολλαπλών επιπέδων (I)

- *AMAT = Average Memory Access Time =*

  *Hit Time (L1) + Miss Rate (L1) * Miss Penalty (L1->L2) +*

  *Miss Rate (L1) * Local Miss Rate (L2) * Miss Penalty (L2-> Main Mem)*

*Local Miss Rate (L2)* : ο ρυθμός αστοχίας της L2 Cache από τις προσπελάσεις που έρχονται από την L1 Cache.

*Global Miss Rate (L2) = Miss Rate (L1) * Local Miss Rate (L2):* ο ρυθμός αστοχίας της L2 Cache από τις συνολικές προσπελάσεις που έρχονται από την CPU

# Παράδειγμα μνημών cache πολλαπλών επιπέδων (Ι)

- Έστω ένα σύστημα με τις παρακάτω προδιαγραφές
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Χρόνος προσπέλασης κύριας μνήμης = 100ns
- Με μόνο L1 Cache:
  - Ποινή αστοχίας σε κύκλους μηχανής = 100ns/0.25ns = 400 cycles
  - CPI = 1 + 0.02 × 400 = 9

# Παράδειγμα μνημών cache πολλαπλών επιπέδων (II)

- Έστω ότι βάζουμε μια L2 Cache στο σύστημά μας
  - Χρόνος προσπέλασης της L2 Cache = 5ns
  - L2 Global miss rate = 0.5%

- Τα 5ns αντιστοιχούν σε 5ns/0.25ns = 20 cycles. Κάθε φορά που υπάρχει miss στην L1 Cache, πληρώνουμε 20 κύκλους να προσπελάσουμε την L2 Cache.
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

# Γιατί είναι σημαντική η ιεραρχία της μνήμης

- Η ιεραρχία της μνήμης είναι πάρα πολύ σημαντική στους σύγχρονους επεξεργαστές με πολλούς πυρήνες.
  - Συνολικό maximum bandwidth που απαιτείται από την CPU.
    - Ο Intel Core i7 μπορεί να εκτελέσει μέχρι 2 εντολές load/store σε κάθε κύκλο μηχανής
    - Έχει 4 πυρήνες με ρολόι στα 3.2 GHz
      - 2*4*3,2 = 25,6 δισ. 64-bit προσπελάσεις δεδομένων/second + 12.8 δισ. 128-bit προσπελάσεις εντολών/second = 409.6 GB/s!
  - Το bandwidth της DRAM είναι μόλις 25GB/s (6% αυτού που απαιτείται!!). Δεν μπορεί να ικανοποιήσει από μόνη της την CPU.
- Η ιεραρχία της μνήμης λοιπόν είναι πολύ σημαντική.

# Που οφείλονται οι αστοχίες της cache;

- Μοντέλο των 3C
- Υποχρεωτικές αστοχίες (Compulsory or Cold misses)
  - Πρώτη πρόσβαση σε ένα block. Δεν μπορεί να αποφευχθεί.
- Αστοχίες Χωρητικότητας(Capacity misses)
  - Οφείλονται στο ότι η cache δεν είναι αρκετά μεγάλη για να περιέχει όλα τα blocks που χρειάζονται από την CPU σε κάθε χρονική περίοδο.
  - Ένα block αντικαθίσταται και μετά το επαναφέρουμε στην cache
  - Είναι ο σημαντικότερος λόγος για αστοχίες cache εκτός εάν η cache είναι πολύ μεγάλη
- Αστοχίες διένεξης (Conflict misses)
  - Συμβαίνουν σε μνήμες cache που δεν είναι fully-associative
  - Πολλά blocks ανταγωνίζονται να μπουν στο ίδιο σύνολο της cache
  - Δεν θα συνέβαιναν εάν η cache ήταν fully-associative