ΗΜΥ 314 – Εργαστήριο Αρχιτεκτονικής Υπολογιστών

Εργαστήριο 5

Cache Simulator

Προσομοιωτής Κρυφής Μνήμης

Μαρία Κ. Μιχαήλ
Αναπληρώτρια Καθηγήτρια ΗΜΜΥ

Εργαστηριακή Άσκηση 5

- Η περιγραφή της εργαστηριακής άσκησης γίνεται στο ακόλουθο PDF αρχείο, που είναι αναρτημένο στο Blackboard:

ECE314.Fall.2025.Lab.Exercise.5.pdf

- Αφού μελετήσετε **ενδελεχώς** το πιο πάνω αρχείο, προχωρήστε στα επόμενα slides…

Εργαστηριακή Άσκηση 5

- Ο προσομοιωτής πρέπει να δέχεται σαν είσοδο (input) ένα αρχείο που θα περιέχει **διευθύνσεις μνήμης**.
- Υποστήριξη λειτουργίας οποιουδήποτε μεγέθους κρυφής μνήμης (το ζητούμενο μέγεθος δίνεται σε **αριθμό blocks**) και των ακόλουθων τύπων κρυφής μνήμης:
  - Άμεσης απεικόνισης (**direct-mapped**)
  - Συσχετιστικής συνόλου Ν δρόμων (**N-way set-associative**), και
  - Πλήρως συσχετιστικής (**fully associative**)
- Υποστήριξη λειτουργίας των ακόλουθων **πολιτικών αντικατάστασης block** (**block replacement policies**) για τους τύπους κρυφής μνήμης set-associative και fully associative:
  - **Least-Recently Used (LRU)**, και
  - **Static Re-Reference Interval Prediction (SRRIP)** (βλέπε επιπρόσθετο PDF αρχείο που είναι αναρτημένο στο Blackboard για τις λεπτομέρειες λειτουργίας του μηχανισμού SRRIP)

Εργαστηριακή Άσκηση 5

- Ο προσομοιωτής θα πρέπει να διαβάσει όλες τις διευθύνσεις και – στο τέλος της εκτέλεσης της λειτουργίας του – θα δημιουργήσει ένα **αρχείο εξόδου** (output file) που θα περιέχει τις εξής πληροφορίες:
  - Όλα τα **περιεχόμενα** (διευθύνσεις μνήμης) της κρυφής μνήμης, δηλ., τα περιεχόμενα του κάθε συνόλου (set) και του κάθε δρόμου (way).
  - Τα ακόλουθα **6 στατιστικά στοιχεία**:
    - Total Number of Cache Queries
    - Total Number of Unique Addresses
    - Total Number of Misses
    - Total Number of Hits
    - Miss Rate
    - Hit Rate

Εργαστηριακή Άσκηση 5

Παράδειγμα: **4-way set-associative** κρυφή μνήμη **χωρητικότητας 64ων blocks**
(δηλ., 16 σύνολα/sets των 4ων δρόμων/ways)

Cache Contents: LRU replacement policy;

Set#	64 Block, 4 -way set-associative cache			
0	0	16	-	-
1	1	-	-	-
2	2	18	-	-
3	-	-	-	-
4	-	-	-	-
5	-	-	-	-
6	-	-	-	-
7	-	-	-	-
8	-	-	-	-
9	-	-	-	-
10	10	26	42	-
11	-	-	-	-
12	12	-	-	-
13	-	-	-	-
14	14	-	-	-
15	-	-	-	-

Μερικές από τις πληροφορίες που πρέπει να εκτυπώνονται μέσα σε αρχείο εξόδου μετά την ολοκλήρωση της λειτουργίας του προσομοιωτή.

Εργαστηριακή Άσκηση 5

- Ο πίνακας που θα αναπαριστά συσχετιστική κρυφή μνήμη συνόλου Ν δρόμων θα είναι της μορφής:

(Συνολικός αριθμός συνόλων στην κρυφή μνήμη) ×
(Συνολικός αριθμός δρόμων σε κάθε σύνολο)

- Ο πίνακας που θα αναπαριστά πλήρως συσχετιστική κρυφή μνήμη θα είναι της μορφής:

1 × (Μέγεθος κρυφής μνήμης σε αριθμό blocks)

- Ο πίνακας που θα αναπαριστά κρυφή μνήμη άμεσης απεικόνισης θα είναι της μορφής:

(Μέγεθος κρυφής μνήμης σε αριθμό blocks) × 1

Επανάληψη σχετικής ύλης από ΗΜΥ 212

Λεπτομέρειες λειτουργίας
κρυφής μνήμης (**cache**)

Συστήματα Μνήμης

- Η οργάνωση του συστήματος μνήμης επηρεάζει τη λειτουργία και απόδοση ενός μικροεπεξεργαστή:
  - Διαχείριση μνήμης και περιφερειακών (I/O) από το λειτουργικό σύστημα
  - Παραγωγή κώδικα από μεταγλωττιστές
  - Χρήση μνήμης από εφαρμογές
  - ΣΗΜΑΝΤΙΚΟΣ παράγοντας στον καθορισμό της απόδοσης

- Τις τελευταίες δεκαετίες, οι αρχιτέκτονες υπολογιστών μετέτρεψαν την οργάνωση της μνήμης από επίπεδη μνήμη τυχαίας προσπέλασης σε ***ιεραρχίες μνήμης***, για βελτίωση της απόδοσης
- ***Ιεραρχία Μνήμης (Memory hierarchy)***: χρήση πολλαπλών επιπέδων μνήμης. Το μέγεθος και ο χρόνος προσπέλασης της μνήμης που είναι «κοντινότερα» στον επεξεργαστή (CPU) είναι μικρότερα και μεγαλώνουν όσο η μνήμη απομακρύνεται από το CPU.

→ Δημιουργεί την ψευδαίσθηση ότι υπάρχει απεριόριστο μέγεθος γρήγορης μνήμης

Θεμελιώδεις αρχές

- **Αρχή Τοπικότητας (*Principle of locality*)**
  Σε τυπικά προγράμματα, ένα μικρό μέρος της μνήμης χρειάζεται να προσπελαστεί σε μια δεδομένη στιγμή

  - **Χρονική Τοπικότητα (*Temporal locality*)**
    Εάν μια θέση μνήμης έχει προσπελαστεί πρόσφατα, τότε τείνει να προσπελαστεί ξανά σε σύντομο χρονικό διάστημα

  - **Χωρική Τοπικότητα (*Spatial locality*)**
    Εάν μια θέση μνήμης έχει προσπελαστεί πρόσφατα, τότε κοντινές θέσεις μνήμης (οι γείτονες), τείνουν να προσπελαστούν ξανά σε σύντομο χρονικό διάστημα

  - Θεωρήστε ένα βρόγχο σε ένα πρόγραμμα. Ποια αρχή τοπικότητας θα επιδεικνύουν οι εντολές μέσα στο βρόγχο; Την αρχή της χρονικής τοπικότητας; Τι γίνεται με την αρχή της χωρικής τοπικότητας;

Αξιοποίηση Ιεραρχίας

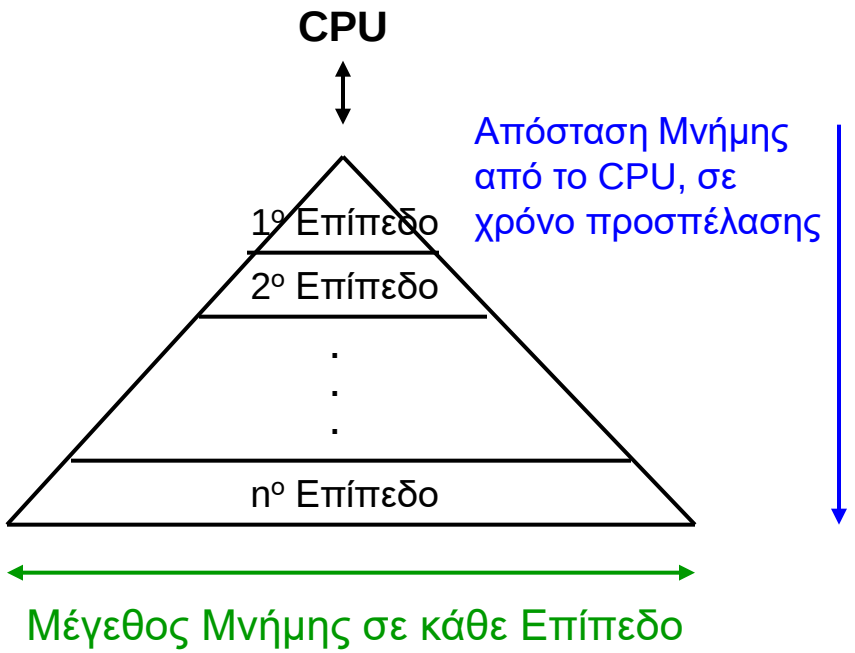
- Οι χρήστες ζητούν διαρκώς μεγαλύτερες (σε χωρητικότητα) και γρηγορότερες μνήμες!

DDR3 DRAM in 2014 → Approx. $10 per GB

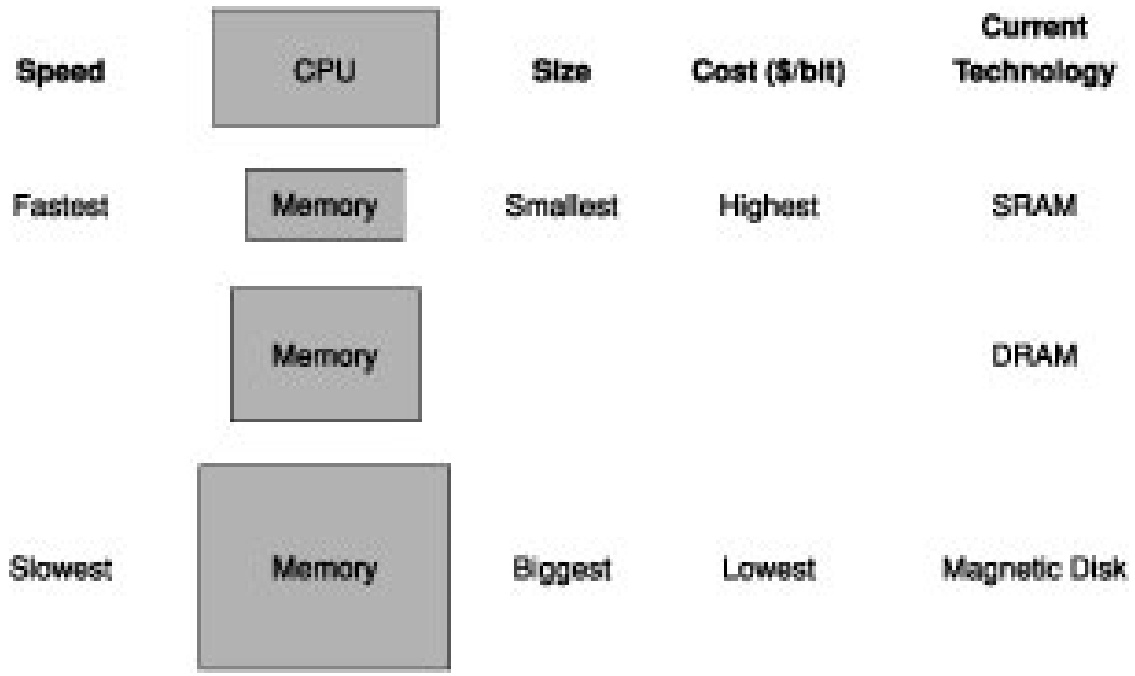
Χρόνος προσπέλασης SRAM μεταξύ 0.5 – 2.5 ns στις $2,000 μέχρι $5,000 για κάθε GB
Χρόνος προσπέλασης DRAM μεταξύ 50 – 70 ns στις $20 μέχρι $75 για κάθε GB
Χρόνος προσπέλασης Δίσκου μεταξύ 5 – 20 ms στις $0.20 μέχρι $2 για κάθε GB

(2008)

- Η βιομηχανία προσπαθεί να αντεπεξέλθει στις απαιτήσεις
  - Κατασκευή μνήμης βάσει ιεραρχίας



Η βασική δομή ιεραρχίας μνήμης



Κρυφή Μνήμη
(cache) [MB]

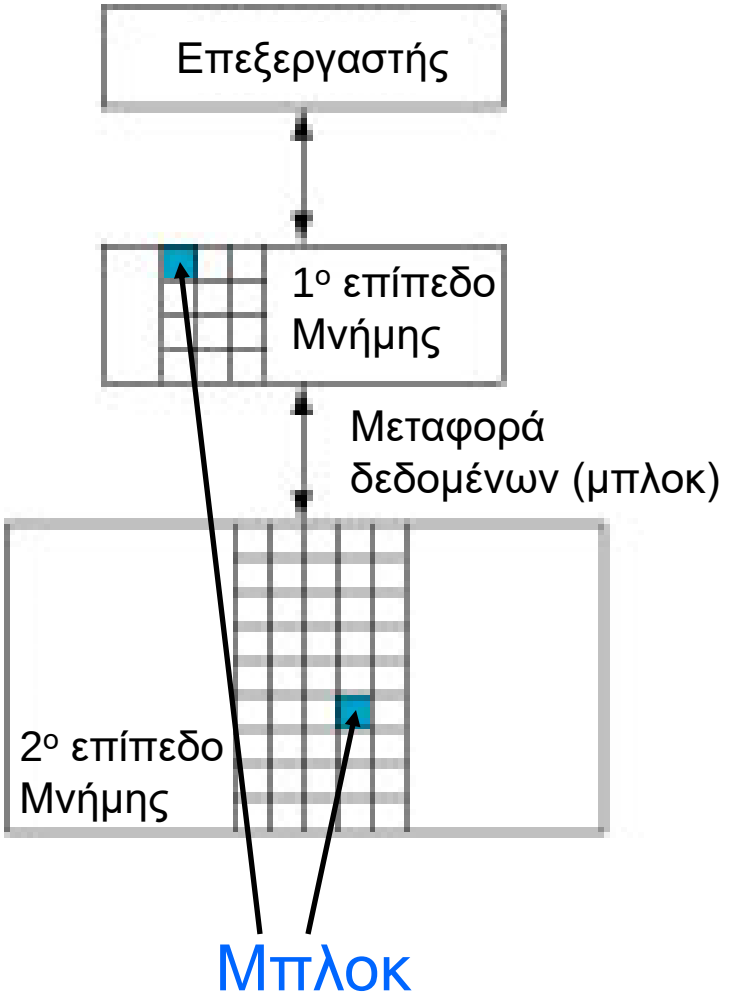
Κυρίως Μνήμη
(main memory) [GB]

Δευτερεύουσα Μνήμη
(secondary memory) [TB]

- Ιεραρχία 3ων επιπέδων. Μπορούμε να έχουμε επιπρόσθετα επίπεδα (ειδικά για κρυφή μνήμη)

Η βασική δομή ιεραρχίας μνήμης (συν.)

- Το επίπεδο που είναι πιο κοντά στο μικροεπεξεργαστή είναι υποσύνολο οποιουδήποτε μεταγενέστερου επιπέδου
- Τα δεδομένα αντιγράφονται μεταξύ δύο γειτονικών επιπέδων
- ***Μπλοκ***: μικρότερη μονάδα πληροφορίας που μπορεί να υπάρχει ή όχι μεταξύ δύο γειτονικών επιπέδων



Βασική Ορολογία – Hit!

- ***Επιτυχής Προσπέλαση (Hit)***: τα δεδομένα που απαιτούνται από τον CPU υπάρχουν στο υψηλότερο επίπεδο της ιεραρχίας της μνήμης

- ***Ρυθμός Επιτυχούς Προσπέλασης (Hit rate)***: ποσοστό προσπελάσεων όπου τα απαιτούμενα δεδομένα βρίσκονται στο υψηλότερο επίπεδο (μετρά την απόδοση της ιεραρχίας)

- ***Χρόνος Επιτυχούς Προσπέλασης (Hit time)***: χρόνος που απαιτείται για την προσπέλαση ενός μπλοκ στο υψηλότερο επίπεδο της ιεραρχίας, συν ο χρόνος που απαιτείται για τον καθορισμό επιτυχούς ή όχι προσπέλασης

Βασική Ορολογία – Miss!

- ***Ανεπιτυχής Προσπέλαση (Miss)***: τα δεδομένα που απαιτούνται από τον CPU ΔΕΝ υπάρχουν στο υψηλότερο επίπεδο μνήμης

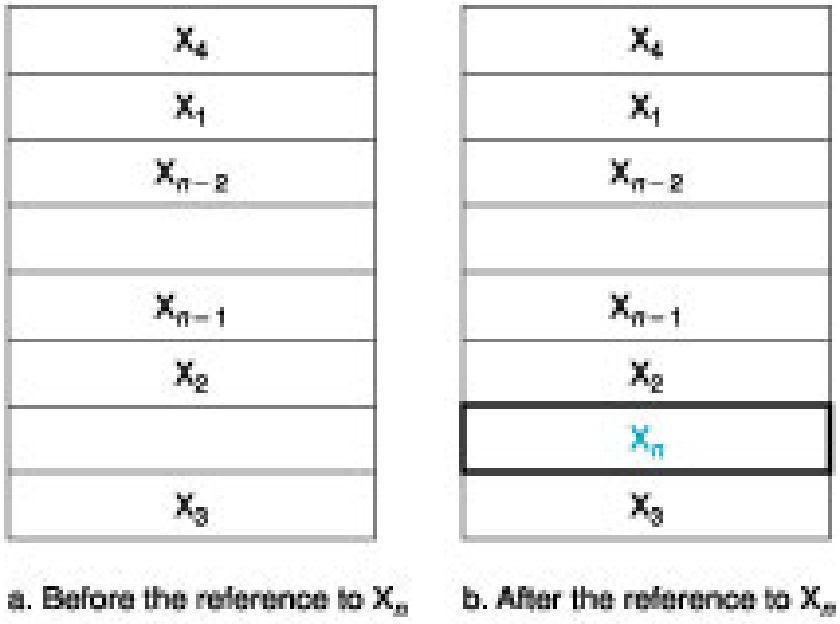
- ***Ρυθμός Ανεπιτυχούς Προσπέλασης (Miss rate)***: ποσοστό προσπελάσεων όπου τα απαιτούμενα δεδομένα ΔΕΝ βρίσκονται στο υψηλότερο επίπεδο (= 1 – hit rate)

- ***Ποινή Ανεπιτυχούς Προσπέλασης (Miss penalty)***: χρόνος που απαιτείται για την μεταφορά ενός μπλοκ από χαμηλότερο σε ψηλότερο επίπεδο της ιεραρχίας, συν ο χρόνος που απαιτείται για τον καθορισμό επιτυχούς ή όχι προσπέλασης

Κρυφή Μνήμη (*Cache memory*)

- **Cache (*κρύπτη, κρυψώνα*):** ασφαλής χώρος για απόκρυψη ή αποθήκευση *-- Webster's New World Dictionary*
- **Σε σχέση με το θέμα μας:** μνήμη που εκμεταλλεύεται την αρχή της τοπικότητας (συνήθως, το(α) επίπεδο(α) μνήμης μεταξύ του επεξεργαστή και της κυρίως μνήμης)

- Παράδειγμα απλής κρυφής μνήμης:
  Θεωρήστε ότι έχουμε αίτημα για τα δεδομένα X_n
  (a) πριν
  (b) μετά



a. Before the reference to X_n b. After the reference to X_n

Κρυφή Μνήμη (συν.)

- Δύο θέματα:
  - Πως μπορούμε να γνωρίζουμε αν τα δεδομένα που χρειαζόμαστε είναι μέσα στην κρυφή μνήμη;
  - Εάν είναι, πως τα βρίσκουμε (σε ποια διεύθυνση);

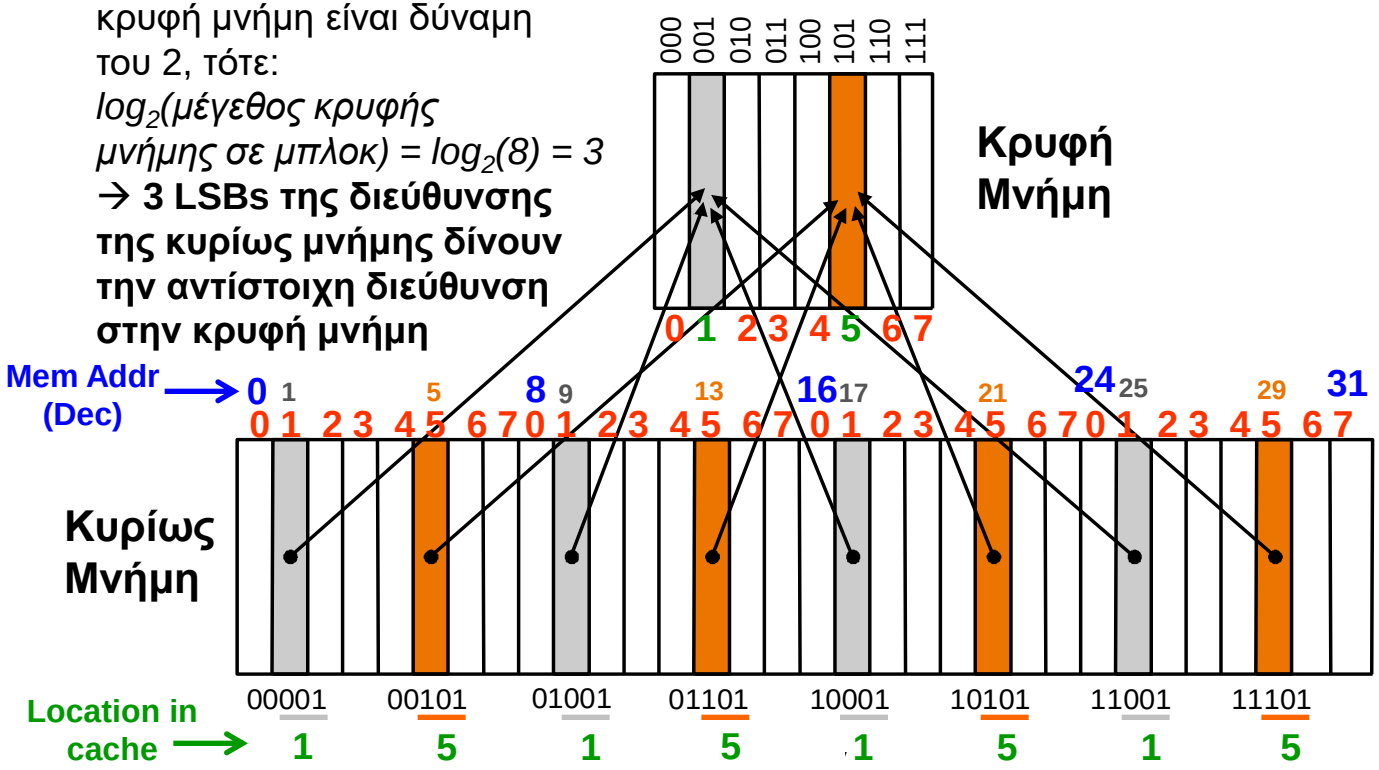
- Πρώτο παράδειγμα:
  - Το μέγεθος του μπλοκ είναι μια λέξη (*single-word block*)
  - Τρόπος οργάνωσης: «direct-mapped» (άμεση απεικόνιση) Κάθε μπλοκ σε ένα χαμηλότερο επίπεδο της ιεραρχίας αντιστοιχεί ακριβώς σε μια τοποθεσία στην κρυφή μνήμη, δηλ., πολλά μπλοκ σε χαμηλότερα επίπεδα μοιράζονται μιαν τοποθεσία στο ψηλότερο επίπεδο μνήμης

Κρυφή Μνήμη Άμεσης Απεικόνισης (*Direct-Mapped Cache*)

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Memory addresses from 0 to 31

- Μέγεθος μπλοκ = 1 word
- Θεωρήστε μια κρυφή μνήμη με 8 μπλοκ (000, …, 111) και μια κυρίως μνήμη με 32 μπλοκ (00000, …, 11111)
- Η διεύθυνση της κρυφής μνήμης που αντιστοιχεί σε μια διεύθυνση στην κυρίως μνήμη: *(διεύθυνση στην κυρίως μνήμη) mod (# μπλοκ στην κρυφή μνήμη)*

Αν ο αρ. των μπλοκ στην κρυφή μνήμη είναι δύναμη του 2, τότε:
log_2(μέγεθος κρυφής μνήμης σε μπλοκ) = log_2(8) = 3
→ 3 LSBs της διεύθυνσης της κυρίως μνήμης δίνουν την αντίστοιχη διεύθυνση στην κρυφή μνήμη



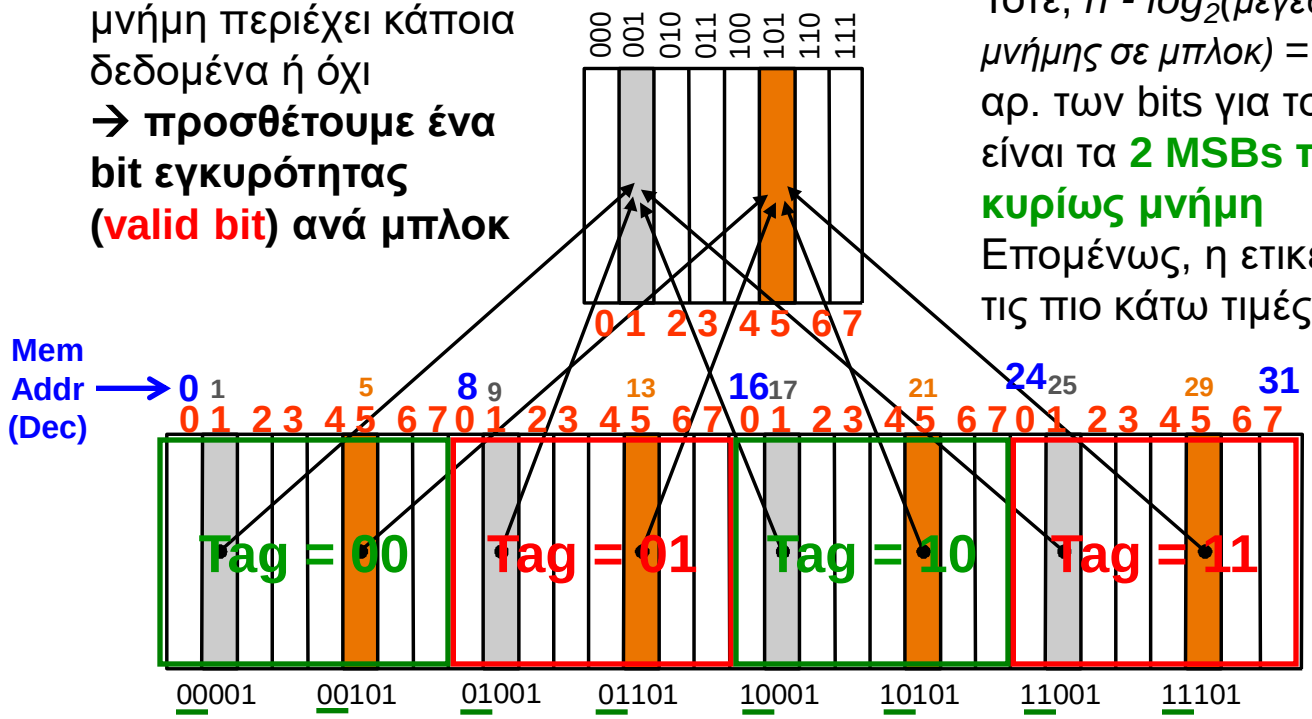
Πως γνωρίζουμε αν τα δεδομένα στην κρυφή μνήμη αντιστοιχούν στην λέξη που ζητείται;
Για παράδειγμα, αν ζητώ την λέξη **10001** **(17)**, πως ξέρω αν η κρυφή μνήμη έχει την **10001** **(17)** και όχι την **00001** **(1)** ή την **01001** **(9)**, ή την **11001** **(25)**;

Κρυφή Μνήμη Άμεσης Απεικόνισης (συν.)

- Προσθέτουμε ένα σύνολο **ετικετών** (***tags***) στην κρυφή μνήμη
- Κάθε μπλοκ στην κρυφή μνήμη έχει ένα επιπρόσθετο πεδίο για την ετικέτα, που χρησιμοποιείται για να **καθορίζει ΕΝΑ μπλοκ στην κυρίως μνήμη**

Τέλος, πρέπει να γνωρίζουμε αν το μπλοκ στην κρυφή μνήμη περιέχει κάποια δεδομένα ή όχι
→ **προσθέτουμε ένα bit εγκυρότητας (valid bit) ανά μπλοκ**

Θεωρήστε *n = # bits* διεύθυνσης κυρίως μνήμης
Τότε, *n - log_2(μέγεθος κρυφής μνήμης σε μπλοκ) = 5 - log_2(8) = 2* δίνει τον αρ. των bits για το πεδίο ετικέτας, το οποίο είναι τα **2 MSBs της διεύθυνσης στην κυρίως μνήμη**
Επομένως, η ετικέτα μπορεί να έχει μία από τις πιο κάτω τιμές {00, 01, 10, 11}



2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)

Παράδειγμα

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Incoming memory addresses from 0 to 31

- Δείξετε τα περιεχόμενα μιας κρυφής μνήμης με 8 μπλοκ της 1 λέξης και με οργάνωση άμεσης απεικόνισης (*8 single-word blocks, direct-mapped cache*) για την πιο κάτω ακολουθία αιτημάτων από τον επεξεργαστή (η κυρίως μνήμη έχει μέγεθος 32 μπλοκ, 2^5=32):

Αιτούμενη Διευθ. (Δεκαδικό)	Αιτούμενη Διευθ. (Δυαδικό)	Hit ή Miss στη Cache	Αντίστοιχο μπλοκ στη cache
22	10110	Miss	10110 mod 8 = 110
26	11010	Miss	11010 mod 8 = 010
22	10110	Hit	10110 mod 8 = 110
26	11010	Hit	11010 mod 8 = 010
16	10000	Miss	10000 mod 8 = 000
3	00011	Miss	00011 mod 8 = 011
16	10000	Hit	10000 mod 8 = 000
18	10010	miss	10010 mod 8 = 010

- Ξεκινήστε με άδεια κρυφή μνήμη …

Cache Example

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Incoming memory addresses from 0 to 31

- 8-blocks, 1 word/block, direct-mapped
- Initial state

Cache

2^3 = 8 blocks (cache size)

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Incoming memory addresses from 0 to 31

New address →

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Mem addr. 22 →

Cache Example

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Incoming memory addresses from 0 to 31

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

New address →

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Mem addr. **26** →

Mem addr. 22 →

Cache Example

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Incoming memory addresses from 0 to 31

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Mem addr. 26 →

Mem addr. 22 →

Cache Example

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Incoming memory addresses from 0 to 31

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

New address → **16**
New address → **3**

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Mem addr. **16** →
Mem addr. 26 →
Mem addr. **3** →
Mem addr. 22 →

← Notice what will happen here on next slide

Cache Example

2^3 = 8 (cache size in blocks)
2^5 = 32 (memory size in blocks)
Incoming memory addresses from 0 to 31

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

New address →

Recall: Word addr 26 = 11010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Mem addr. 16 →
Mem addr. **18** **(replaces 26)** →
Mem addr. 3 →
Mem addr. 22 →

← Example of **Replacement**/ **Eviction**

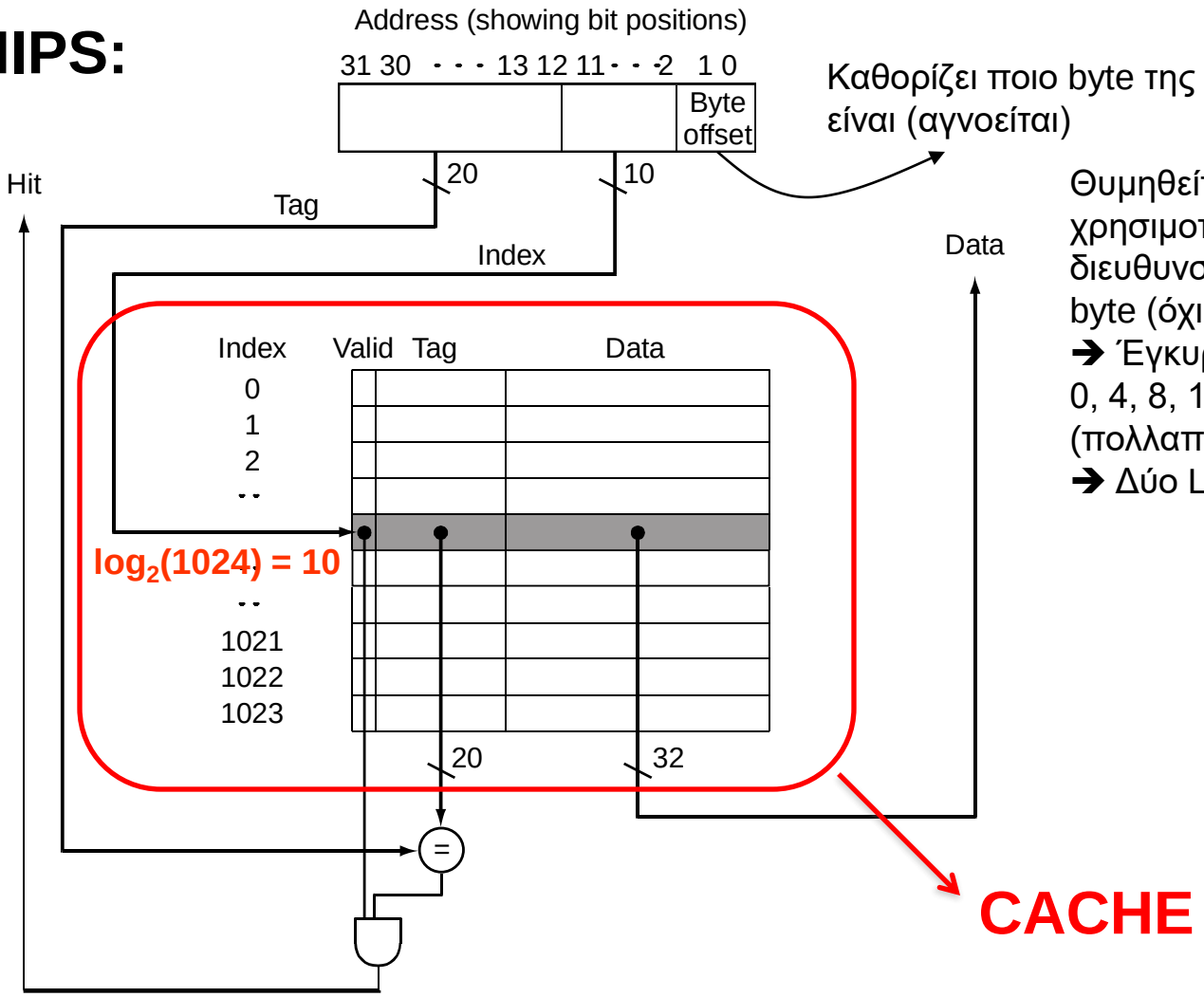
Αντικατάσταση/ Έξωση

Κρυφή Μνήμη Άμεσης Απεικόνισης (συν.)

Here, assume that the **size of the cache** (in number of blocks) is given as **1024**

Μπλοκ 1 λέξης

- **Για MIPS:**



Θυμηθείτε ότι ο MIPS χρησιμοποιεί διευθυνσιοδότηση ανά byte (όχι word ή block)
➔ Έγκυρες διευθ.:
0, 4, 8, 12, 16, …
(πολλαπλάσια του 4)
➔ Δύο LSBs πάντα 0

Απόδοση κρυφής μνήμης

- In general, cache penalties increase as a processor becomes faster and the memory system remains the same:
  - The lower the CPI the greater the amount of time spent in memory stall cycles.
  - Higher processor clock rate leads to larger miss penalties (DRAM access time does not improve as much as CPU clock rates → **Processor-Memory Gap**)
- Thus, cache performance in today's processors of low CPI and high clock rates is crucial
  - Flexible placement of blocks (reduces cache miss rate)
  - Multilevel caches (reduces miss penalty)

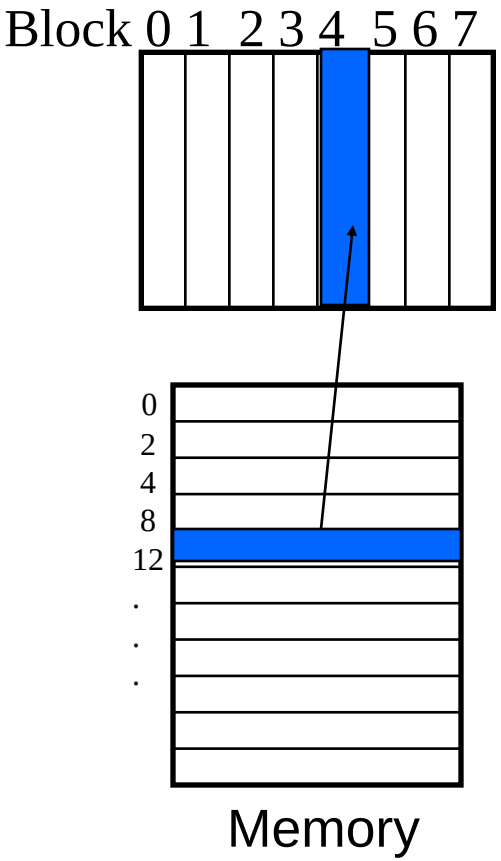
Μείωση ανεπιτυχούς ανάγνωσης (cache miss)

- Direct-mapped cache
  - a block goes in **exactly** one place in the cache
- Fully associative cache
  - a block can be placed in **any** location in the cache
  - to find a block, all cache entries need to be searched (can be implemented with parallel comparators, increases hardware)
- Set-associative cache
  - a block can be placed in a **fixed number** of locations (at least two) within each ***set*** in the cache
  - For ***n*** fixed locations (called ***ways***) per block, we have an ***n-way set-associative cache***

Fully associative cache = **Πλήρως συσχετιστική** κρυφή μνήμη
Set-associative cache = **Συσχετιστική** κρυφή μνήμη **συνόλου**
N-way set-associative cache = **Συσχετιστική** κρυφή μνήμη **συνόλου Ν δρόμων**

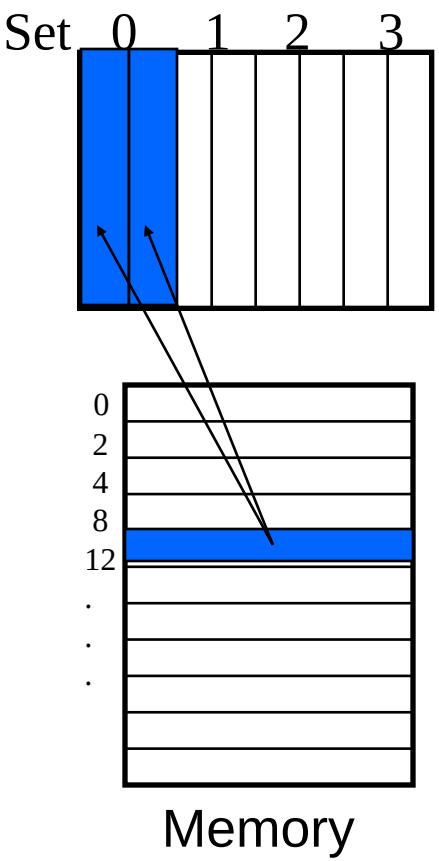
Οργάνωση Κρυφής Μνήμης

Direct-mapped



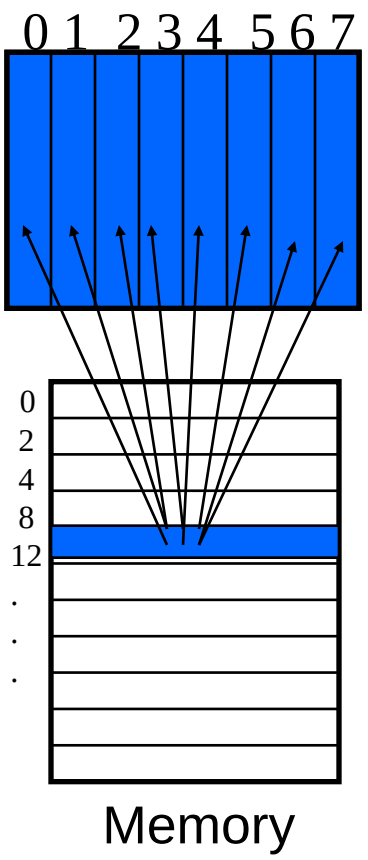
12 mod **8** = 4
searches = 1

2-way set-associative



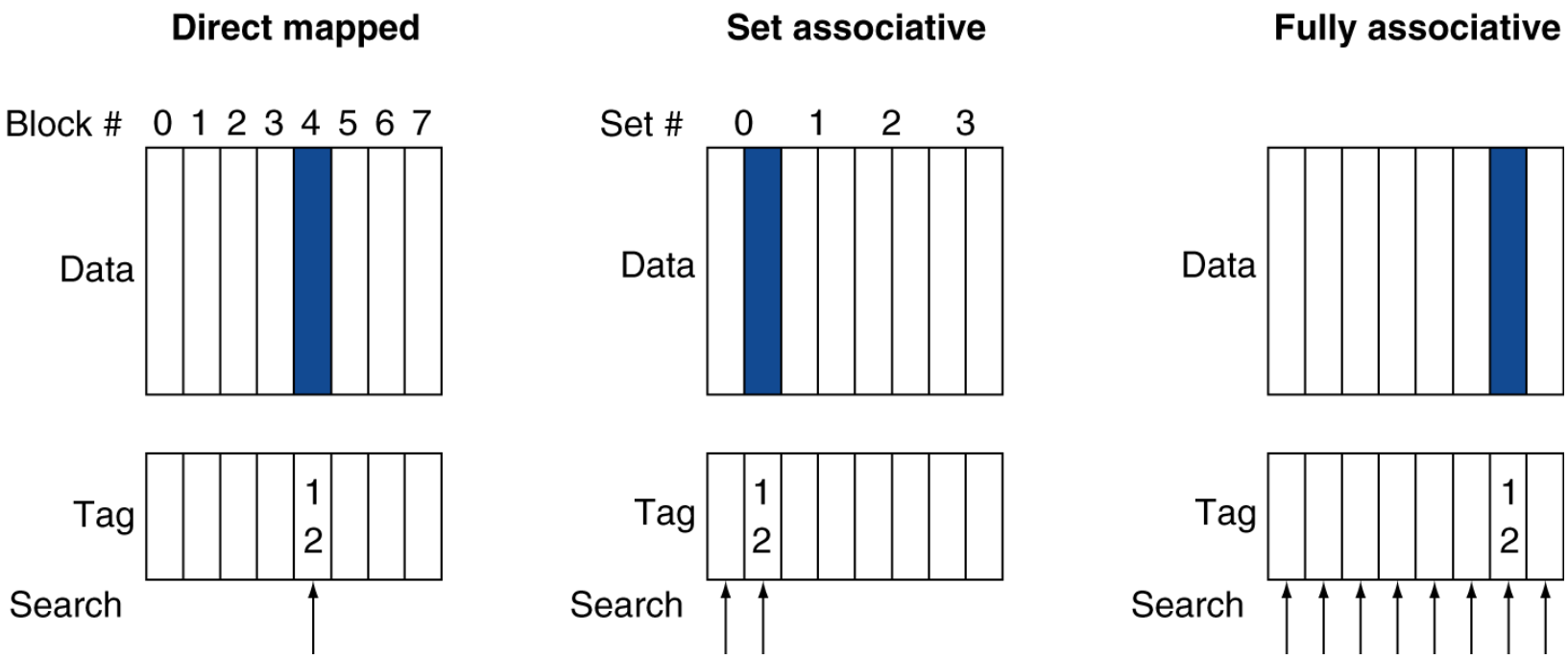
12 mod **4** = 0
searches = 2

Fully associative



searches = 8

Associative Cache Example



Number of **TAG searches** per cache **access**

N-way set-associative cache

- Cache consists of a number of ***sets;*** each ***set*** contains ***n*** blocks (***ways***)
- A memory block is associated with exactly 1 set
- A block can be placed/found in any of the *n* blocks (ways) of its associated set → all tags/elements in a set need to be searched

- **Combines direct-mapped and fully associative**:
  - Directly mapped to a set
  - Fully associative within a single set

- In general, every block placement strategy is a variation of the set associativity scheme (see the next slide)

Κρυφή Μνήμη με 8 μπλοκς

8 sets of 1 way

One-way set associative

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

4 sets of 2 ways

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

2 sets of 4 ways

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

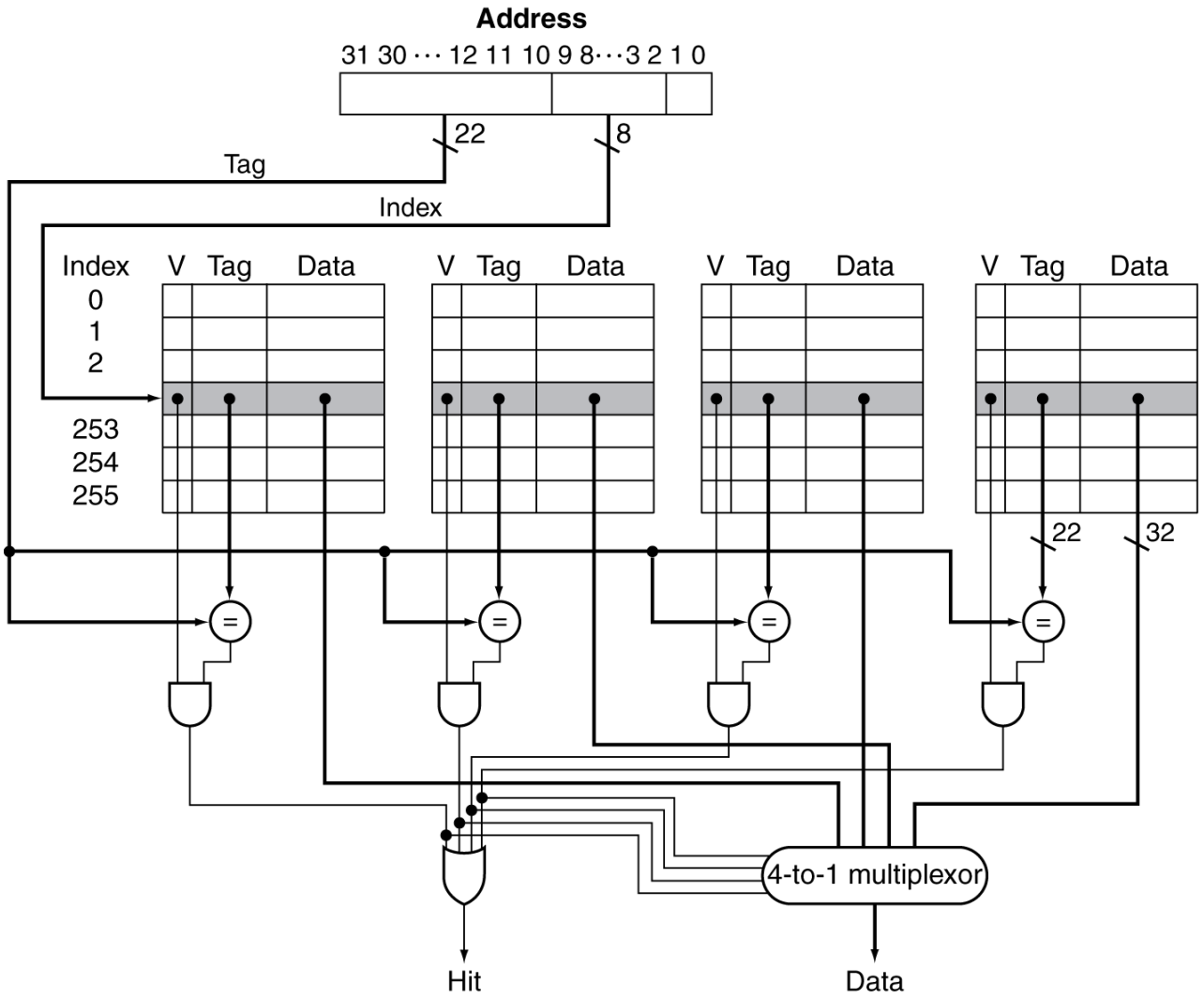
Eight-way set associative (fully associative) **1 set of 8 ways**

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

The total size of the cache *in blocks* is equal to the number of sets * the associativity. Thus, for a fixed cache size, increasing the associativity (i.e., the number of ways per set) decreases the number of sets, while increasing the number of elements (blocks) per set

Increasing degree of association: *usually* decreases miss rate, but increases hit time

Υλοποίηση: 4-way set association



Associativity Example

- Compare **4-block caches**
  - Direct-mapped, 2-way set-associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct-mapped **5 misses**

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]	

(0 modulo 4) = 0
(6 modulo 4) = 2
(8 modulo 4) = 0

0 mod n = 0

Associativity Example

(0 modulo 2) = 0
(6 modulo 2) = 0
(8 modulo 2) = 0

- 2-way set-associative **4 misses**

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]		

Set-associative caches usually replace the **Least Recently Used (LRU)** block within a set; that is, **the block that was used furthest in the past is replaced**.

- Fully associative **3 misses**

This is the best we can do, because 3 unique block addresses are accessed (i.e., 3 compulsory misses).

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]	

The fully associative cache has four cache blocks (in a single set). Any memory block can be stored in any cache block.

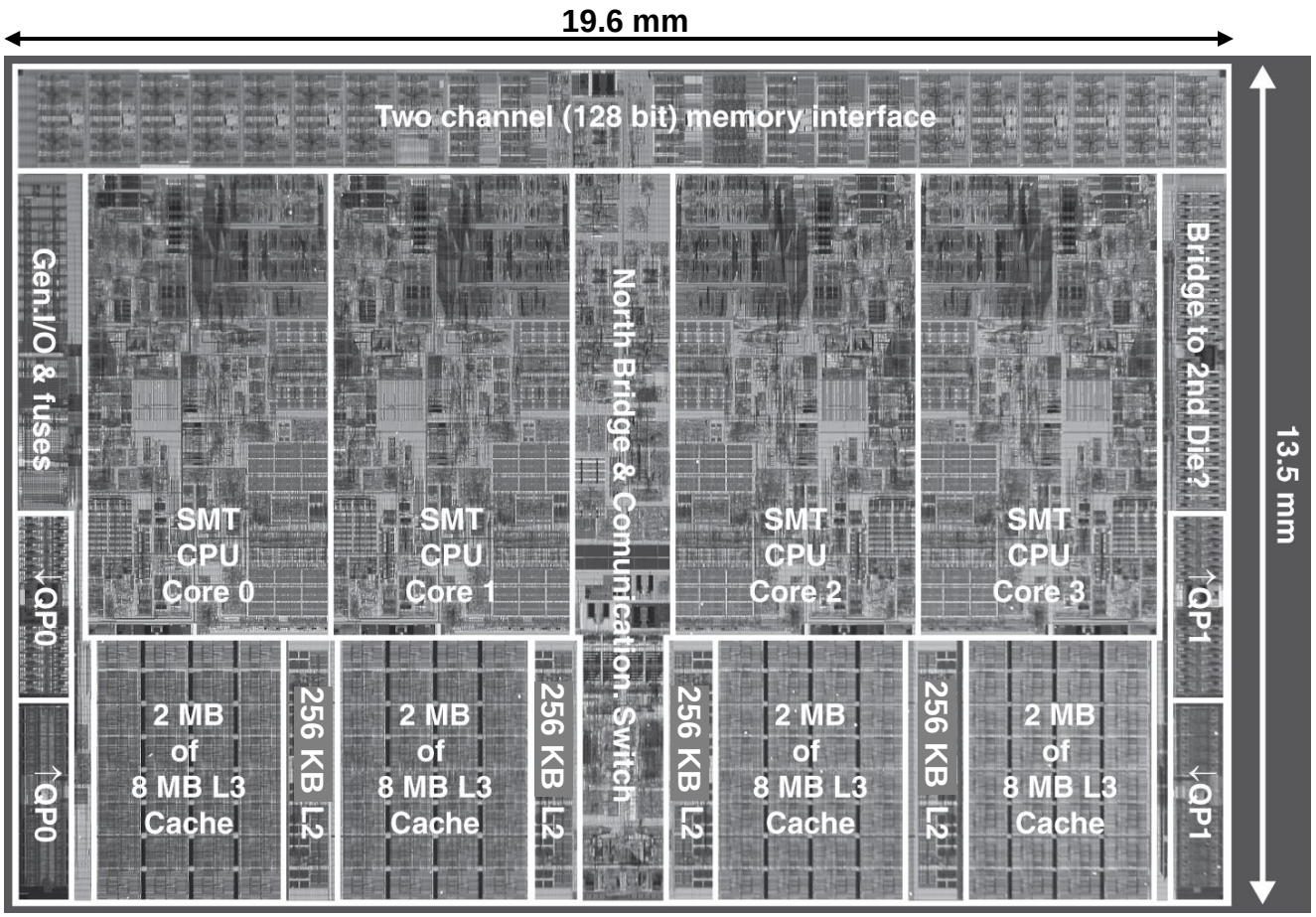
Sources of Misses in Caches

The **Three Cs** of Prof. Mark D. Hill (University of Wisconsin, Madison, USA)

- Compulsory misses (aka cold-start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully-associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully-associative cache of the same total size

Case Study: Multilevel On-Chip TLBs & Caches

Intel Nehalem 4-core processor [**Core i7**, November 2008]



Die size:
265 mm^2

Transistor count:
731 million

Per core: 32 KB L1 I-cache, 32 KB L1 D-cache, 256 KB L2 cache, 2 MB L3 cache

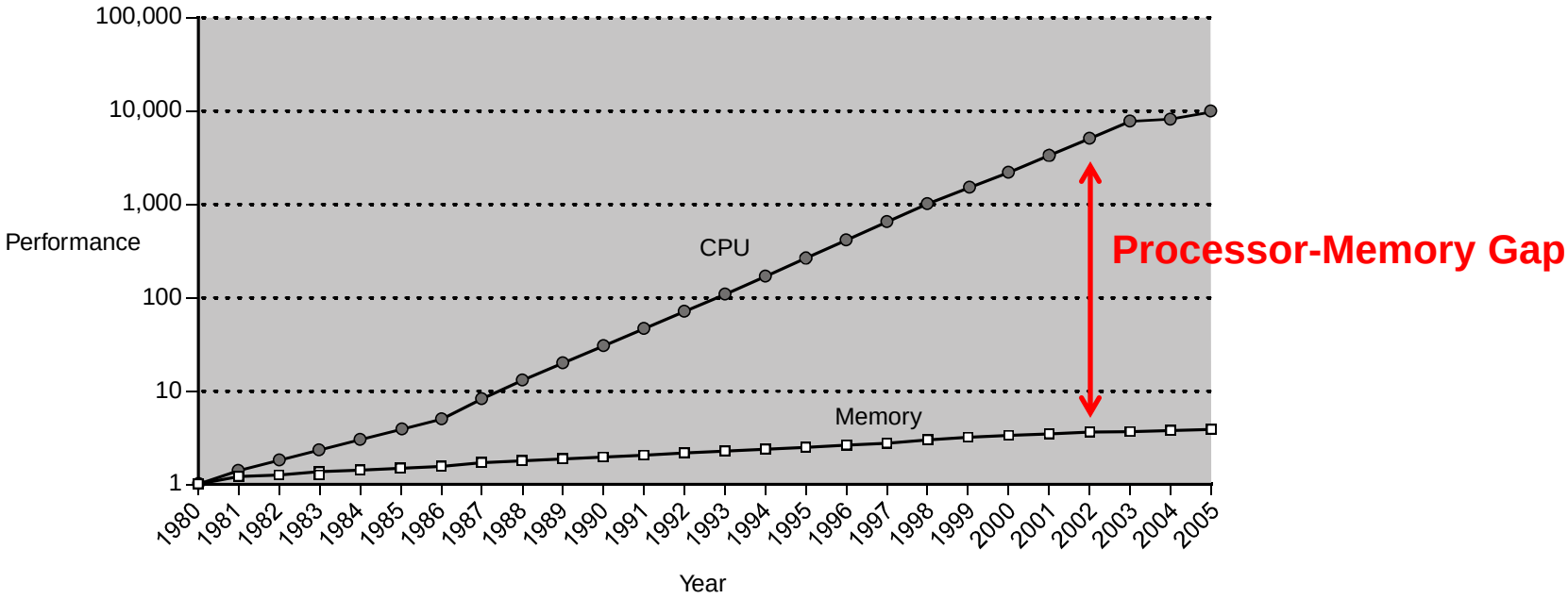
Total: 8 MB L3 cache

Intel Nehalem [2008] 3-Level Cache Organization

L1 caches (per core)	**L1 I**-cache: 32 KB, 64-byte blocks, 4-way, approx. LRU replacement, hit time 4 cycles **L1 D**-cache: 32 KB, 64-byte blocks, 8-way, approx. LRU replacement, hit time 4 cycles
L2 unified cache (per core)	256 KB, 64-byte blocks, 8-way, approx. LRU replacement, hit time <12 cycles
L3 unified cache (shared)	8 MB, 64-byte blocks, 16-way, replacement n/a, hit time 30-40 cycles

Σύγχρονα Συστήματα

- Processor speeds continue to increase very fast
  - much faster than either DRAM or disk access times



- Design challenge: dealing with this growing disparity
  - Pre-fetching? 3rd-level caches and more? Memory design?

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- **Memory system design** is *critical* for microprocessors

Σύνοψη: Εργαστηριακή Άσκηση 5

Παράδειγμα: **4-way set-associative** κρυφή μνήμη **χωρητικότητας 64ων blocks**
(δηλ., 16 σύνολα/sets των 4ων δρόμων/ways)

Cache Contents: LRU replacement policy;

Set#	64 Block, 4 -way set-associative cache			
0	0	16	-	-
1	1	-	-	-
2	2	18	-	-
3	-	-	-	-
4	-	-	-	-
5	-	-	-	-
6	-	-	-	-
7	-	-	-	-
8	-	-	-	-
9	-	-	-	-
10	10	26	42	-
11	-	-	-	-
12	12	-	-	-
13	-	-	-	-
14	14	-	-	-
15	-	-	-	-

Μερικές από τις πληροφορίες που πρέπει να εκτυπώνονται μέσα σε αρχείο εξόδου μετά την ολοκλήρωση της λειτουργίας του προσομοιωτή.