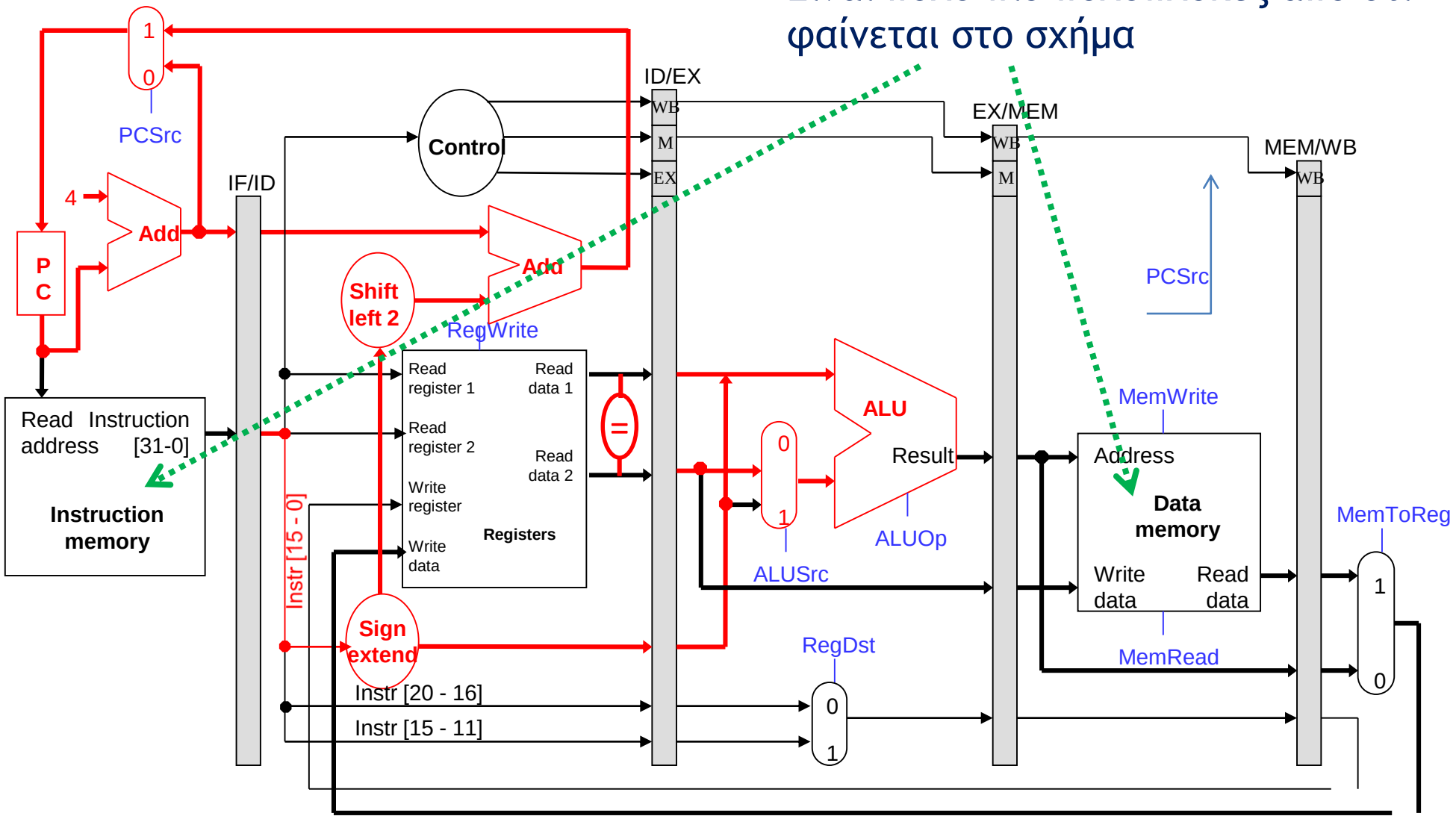
ΗΥ 232
Οργάνωση και Σχεδίαση Υπολογιστών

Διάλεξη 14
Εισαγωγή στην Ιεραρχία Μνήμης

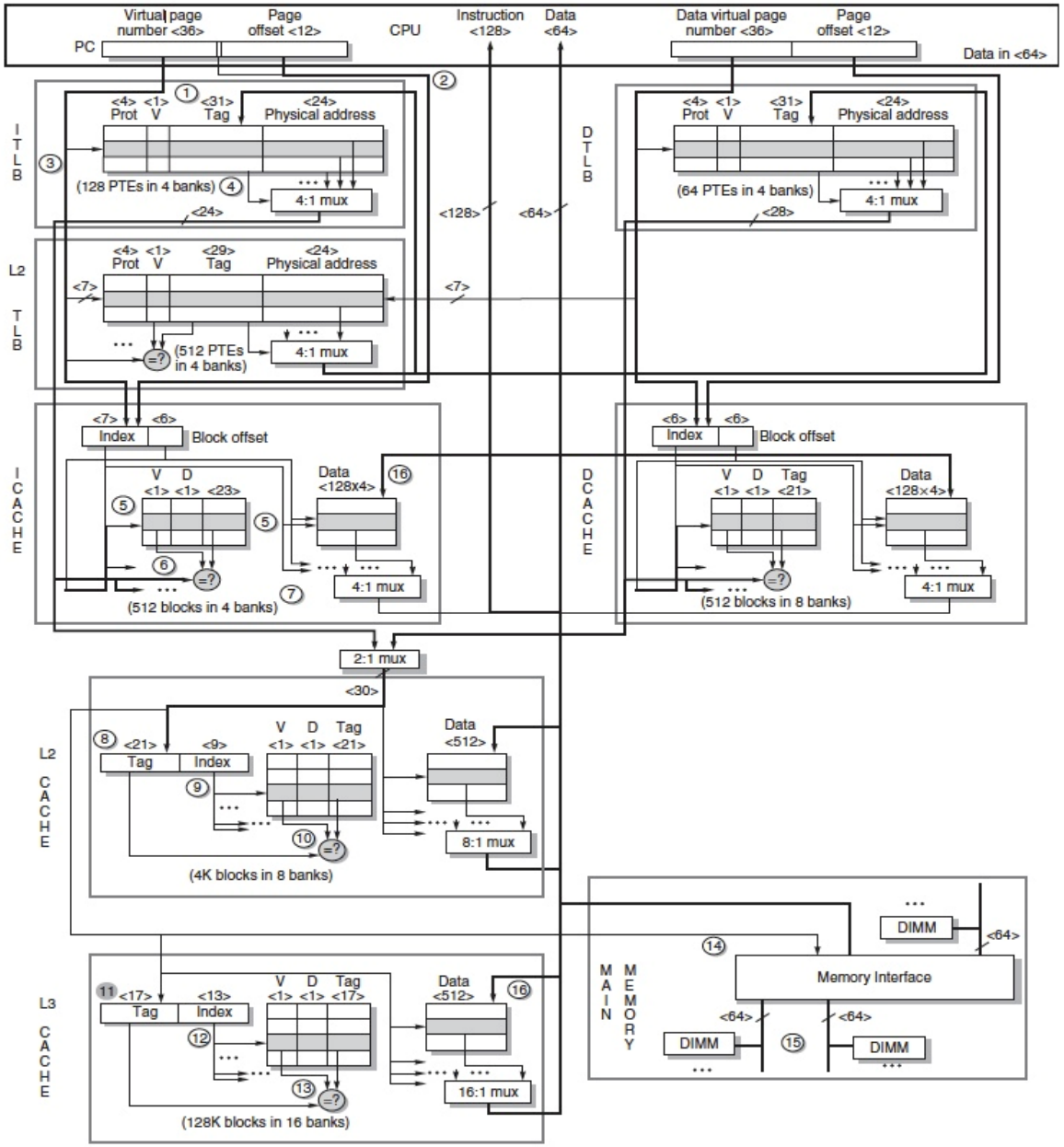
Νίκος Μπέλλας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

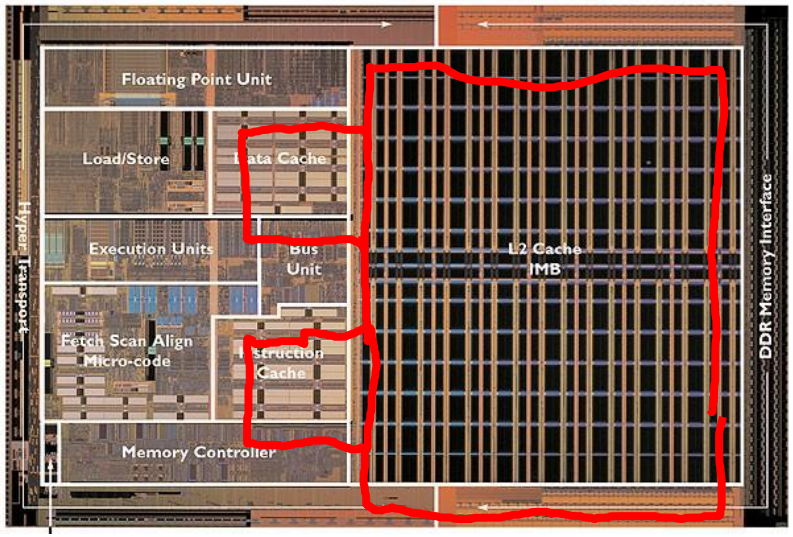
Η Μικρο-αρχιτεκτονική μας

Το data path είναι σε καλή κατάσταση.
Οι μνήμες όμως;
Είναι πολύ πιο πολύπλοκες από ότι φαίνεται στο σχήμα



Μνήμες σε πραγματικό σύστημα
Core-i7 ιεραρχία μνήμης





Ιεραρχία Μνήμης
Large and Fast

- Έχουμε δει ήδη κάποιες βασικές τεχνικές για να επιταχύνουμε την επεξεργασία δεδομένων στην CPU (πχ διοχέτευση)
- Δεν έχουμε πει όμως τίποτε για τις μνήμες.
  - Χρειαζόμαστε μεγάλες μνήμες για να αποθηκεύσουμε εντολές και δεδομένα για μεγάλες εφαρμογές όπως βάσεις δεδομένων, video και μουσική, κτλ
  - Χρειαζόμαστε γρήγορες μνήμες για να μπορούμε να παρέχουμε εντολές και δεδομένα με τον ίδιο ρυθμό που μας ζητάει η CPU. Αλλιώς η CPU θα περιμένει συνέχεια χωρίς να κάνει τίποτα.
- Το μοντέλο ότι έχουμε μια μεγάλη επίπεδη μνήμη η οποία μπορεί να παρέχει εντολές και δεδομένα σε έναν κύκλο και έχει ανεξάντλητο μέγεθος δεν ισχύει.
- Σε αυτήν την ενότητα θα δούμε πως μπορούμε να σχεδιάσουμε τις μνήμες εντολών και δεδομένων ώστε να προσομοιάζει όσο το δυνατόν το παραπάνω μοντέλο.
  - Κρυφές μνήμες (cache memories).
  - Εικονικές μνήμες (virtual memories).

Είδη μνήμης

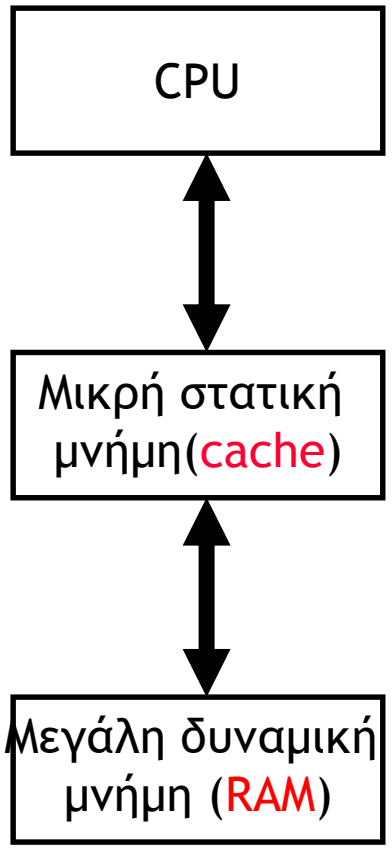
- Η τεχνολογία μνημών είναι είτε γρήγορη και ακριβή (SRAMs), είτε αργή και φθηνή (Magnetic Disks)
- Η πιο γρήγορη μνήμη (SRAM) είναι πολύ ακριβή για να χρησιμοποιηθεί εξ' ολοκλήρου σε μια CPU .
- Από την άλλη, η δυναμική μνήμη (DRAM) έχει χαμηλότερη ταχύτητα από την λογική που χρησιμοποιείται στην μικρο-αρχιτεκτονική. Εάν χρησιμοποιηθεί εξ'ολοκλήρου σε μια CPU, το σύστημα θα πρέπει να καθυστερεί πολλούς κύκλους για να διαβάζουμε δεδομένα ή εντολές.

Είδος Μνήμης	Ταχύτητα	Κόστος/MB	Χωρητικότητα
Static RAM	Η πιο γρήγορη	Ακριβή	Μικρή
Dynamic RAM	Αργή	Φθηνή	Μεγάλη
Hard disks	Η πιο αργή	Η πιο Φθηνή	Η μεγαλύτερη

Είδος Μνήμης	Ταχύτητα	Κόστος/MB	Χωρητικότητα
Static RAM	1-10 cycles	~$5	έως ~4MB
Dynamic RAM	100-200 cycles	~$0.10	128MB-4GB
Hard disks	10,000,000 cycles	~$0.0005	20GB-400GB

Μνήμη Cache

- Το ιδανικό λοιπόν είναι να βρούμε μια ισορροπία μεταξύ γρήγορης και φθηνής μνήμης. Πως μπορούμε να έχουμε τον καλύτερο συνδυασμό των δύο αυτών κόσμων;
- Η λανθάνουσα μνήμη (cache memory) είναι μια μικρή και γρήγορη μνήμη που ανταποκρίνεται σε αυτές τις προδιαγραφές. Επειδή είναι μικρή σε μέγεθος σε σχέση με την κύρια μνήμη δεν είναι πολύ ακριβή.
  - Η cache βρίσκεται ανάμεσα στον επεξεργαστή και στην (μεγαλύτερη και πιο αργή) κύρια μνήμη.
  - Κρατάει αντίγραφα των πιο συχνά εκτελούμενων εντολών και δεδομένων.
- Η προσπέλαση δεδομένων και εντολών μειώνεται με το να κάνουμε την συχνή περίπτωση πιο γρήγορη.
- Η κύρια μνήμη χρειάζεται να προσπελασθεί μόνο για δεδομένα που δεν είναι στην cache. Αυτά είναι συνήθως δεδομένα που δεν εκτελούνται τόσο συχνά



Cache (Merriam-Webster online dictionary)
- a hiding place especially for concealing and preserving provisions or implements
- a secure place of storage

Η αρχή της τοπικότητας των προσπελάσεων μνήμης

- Η ιεραρχία της μνήμης σε έναν επεξεργαστή βασίζεται σε μία πολύ βασική εμπειρική αρχή:
  - Τα προγράμματα σε κάθε χρονική στιγμή προσπελαύνουν ένα μικρό ποσοστό των δεδομένων και εντολών τους.
- Η αρχή της τοπικότητας διακρίνεται σε:
  - την αρχή της χρονικής τοπικότητας (temporal locality) που λέει ότι ένα πρόγραμμα που προσπελαύνει μια διεύθυνση μνήμης έχει αυξημένες πιθανότητες να την προσπελάσει ξανά σύντομα.
  - την αρχή της χωρικής τοπικότητας (spatial locality) που λέει ότι ένα πρόγραμμα που προσπελαύνει μια διεύθυνση μνήμης έχει αυξημένες πιθανότητες να προσπελάσει σύντομα μια γειτονική διεύθυνση μνήμης.

Χρονική τοπικότητα σε προγράμματα

- Τα loops είναι τυπικά παραδείγματα εμφάνισης χρονικής τοπικότητας σε προγράμματα.
  - Το block του loop εκτελείται πολλές φορές, και συνεπώς κάθε εντολή στο loop προσπελαύνεται συνεχώς από την μνήμη
- Εάν τοποθετήσουμε τις εντολές του loop σε μια μνήμη cache, θα μπορούμε να τις προσπελαύνουμε συνεχώς από εκεί και όχι από την κύρια μνήμη.

```
Loop:  lw    $t0, 0($s1)
       add   $t0, $t0, $s2
       sw    $t0, 0($s1)
       addi  $s1, $s1, -4
       bne   $s1, $0, Loop
```

Χρονική τοπικότητα σε δεδομένα

- Πολλά προγράμματα προσπελαύνουν τις ίδιες μεταβλητές, ειδικά μέσα σε loops.
  - Για παράδειγμα, οι μεταβλητές sum και i προσπελαύνονται συνεχώς.

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + f(i);
```

- Τέτοιες μεταβλητές πολλές φορές αποθηκεύονται σε καταχωρητές και όχι στην μνήμη, για να μπορούν να διαβαστούν χωρίς να χρειάζεται να φορτωθούν πρώτα από την μνήμη. Αυτό, όμως, δεν είναι πάντα δυνατόν να γίνει.
  - Υπάρχει περιορισμένος αριθμός καταχωρητών σε μια CPU.

Χωρική τοπικότητα σε προγράμματα

- Όταν προσπελάσουμε μια εντολή, κατά πάσα πιθανότητα, αμέσως μετά θα προσπελάσουμε και τις εντολές που έπονται στον χώρο. Εάν εκτελέσουμε μια εντολή στην διεύθυνση *i* , είναι πολύ πιθανόν να εκτελέσουμε μια εντολή στις επόμενες διευθύνσεις.
- Τμήματα κώδικα όπως τα loops χαρακτηρίζονται πολύ συχνά από χρονική και χωρική τοπικότητα ταυτόχρονα.

```
sub  $sp, $sp, 16
sw   $ra, 0($sp)
sw   $s0, 4($sp)
sw   $a0, 8($sp)
sw   $a1, 12($sp)
```

Χωρική τοπικότητα σε δεδομένα

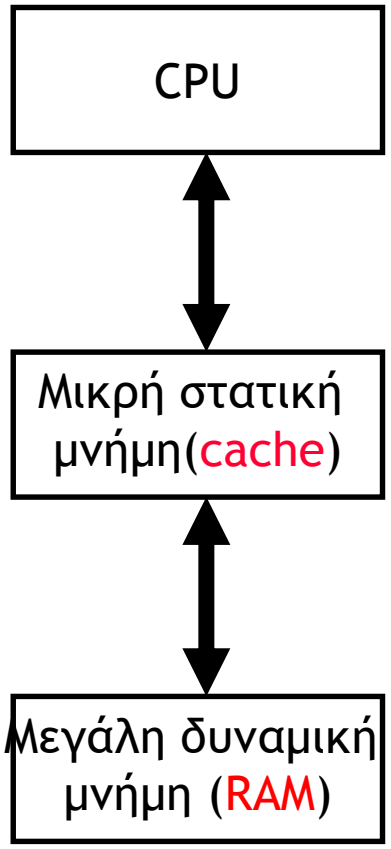
- Πολλά προγράμματα προσπελαύνουν δεδομένα που είναι αποθηκευμένα σε γειτονικές θέσεις μνήμης.
  - Οι πίνακες (όπως το array στον κώδικα) αποθηκεύονται σε συνεχείς θέσεις στην μνήμη.
  - Τα πεδία ενός struct ή object επίσης αποθηκεύονται σε συνεχείς θέσεις στην μνήμη.
- Άρα και τα δεδομένα σε loops μπορούν να έχουν χρονική και χωρική τοπικότητα ταυτόχρονα.

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + array[i];
```

```
employee.name = "Homer Simpson";
employee.boss = "Mr. Burns";
employee.age = 45;
```

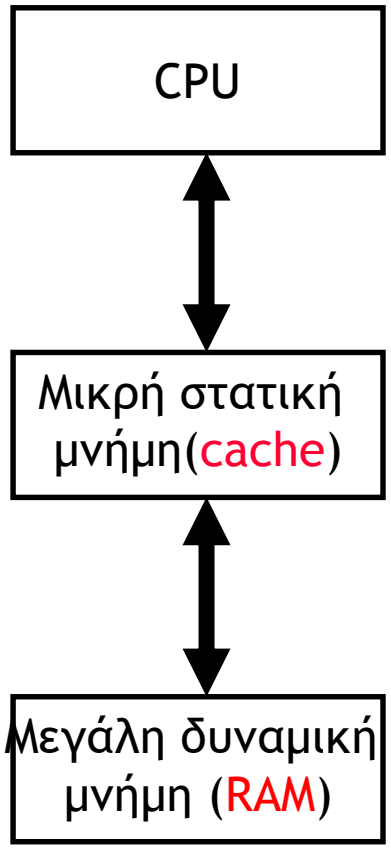
Μνήμες Cache και χρονική τοπικότητα

- Στην πρώτη προσπέλαση μιας διεύθυνσης, α) τα δεδομένα διαβάζονται από την κύρια μνήμη , και β) ένα αντίγραφο των δεδομένων μεταφέρεται και στην cache.
  - Με αυτόν τον τρόπο τα δεδομένα μπορούν να διαβάζονται στην συνέχεια από την γρήγορη cache (1 κύκλος μηχανής) και όχι από την αργή κύρια μνήμη (δεκάδες ή και εκατοντάδες κύκλοι μηχανής)
  - Η πρώτη προσπέλαση είναι αργή, αλλά αφού "ζεσταθεί" η cache με δεδομένα, οι επόμενες προσπελάσεις είναι πολύ πιο γρήγορες.
- Αυτός ο αλγόριθμος εκμεταλλεύεται την χρονική τοπικότητα. Δεδομένα και εντολές που προσπελαύνονται συχνά θα βρίσκονται στην cache.



Μνήμες Cache και χωρική τοπικότητα

- Οπως μόλις είδαμε, όταν η CPU διαβάζει από την θέση *i* της κύριας μνήμης , ένα αντίγραφο των δεδομένων στην θέση αυτή αποθηκεύεται στην cache.
- Αντί όμως να αντιγράφουμε μόνο την θέση *i* αντιγράφουμε και αρκετές άλλες θέσεις γειτονικές της i; για παράδειγμα τα δεδομένα στις θέσεις από *i* μέχρι και *i* + 3.
  - Για παράδειγμα, αντί να διαβάσουμε μόνο ένα στοιχείο A[i] του πίνακα Α, μπορούμε να διαβάσουμε τα στοιχεία A[i] .. A[i+3] ή και παραπάνω ΠΡΙΝ τα ζητήσει η CPU.
  - Prefetching
  - Τα δεδομένα σε αυτές τις θέσεις θα βρίσκονται στην cache εάν η CPU αργότερα χρειασθεί να τα προσπελάσει..

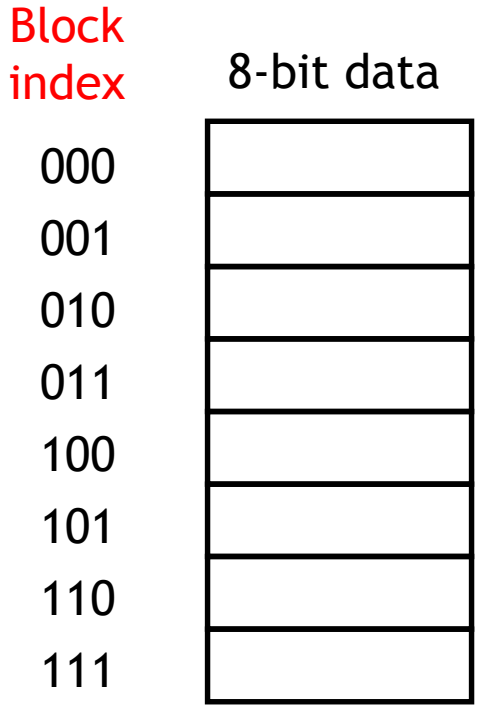


Η ιδέα της cache είναι πολύ γενικότερη

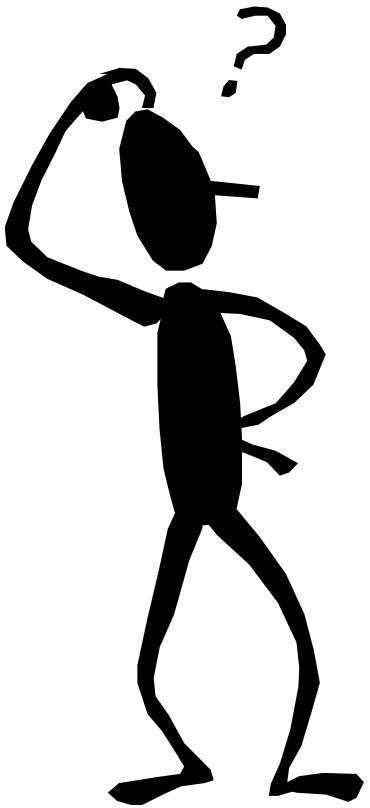
- Η ιδέα του να φέρουμε τα δεδομένα που επεξεργαζόμαστε συχνά κοντά στον τόπο επεξεργασίας είναι μία από τις πιο βασικές αρχές της επιστήμης υπολογιστών.
- Πάρτε για παράδειγμα τα δίκτυα.
  - Δίκτυα όπως το internet έχουν μεγάλο χρόνο απόκρισης (latency), και συνεπώς το να μεταφέρουμε δεδομένα συνεχώς δεν είναι επιθυμητό.
  - Web browsers όπως Firefox και Chrome αποθηκεύουν σελίδες που προσπελαύνετε συχνά στον σκληρό σας δίσκο.

Μια απλή μνήμη cache

- Οι Caches διαιρούνται σε blocks που αποτελούνται από ένα ή περισσότερα bytes.
  - Ο αριθμός των blocks σε μια cache είναι συνήθως δύναμη του 2.
  - Ας πάρουμε την απλή περίπτωση που ένα block έχει μέγεθος ένα byte. Συνήθως το block έχει πολύ μεγαλύτερο μήκος, αλλά ας το δεχτούμε αυτό για χάριν απλότητας.



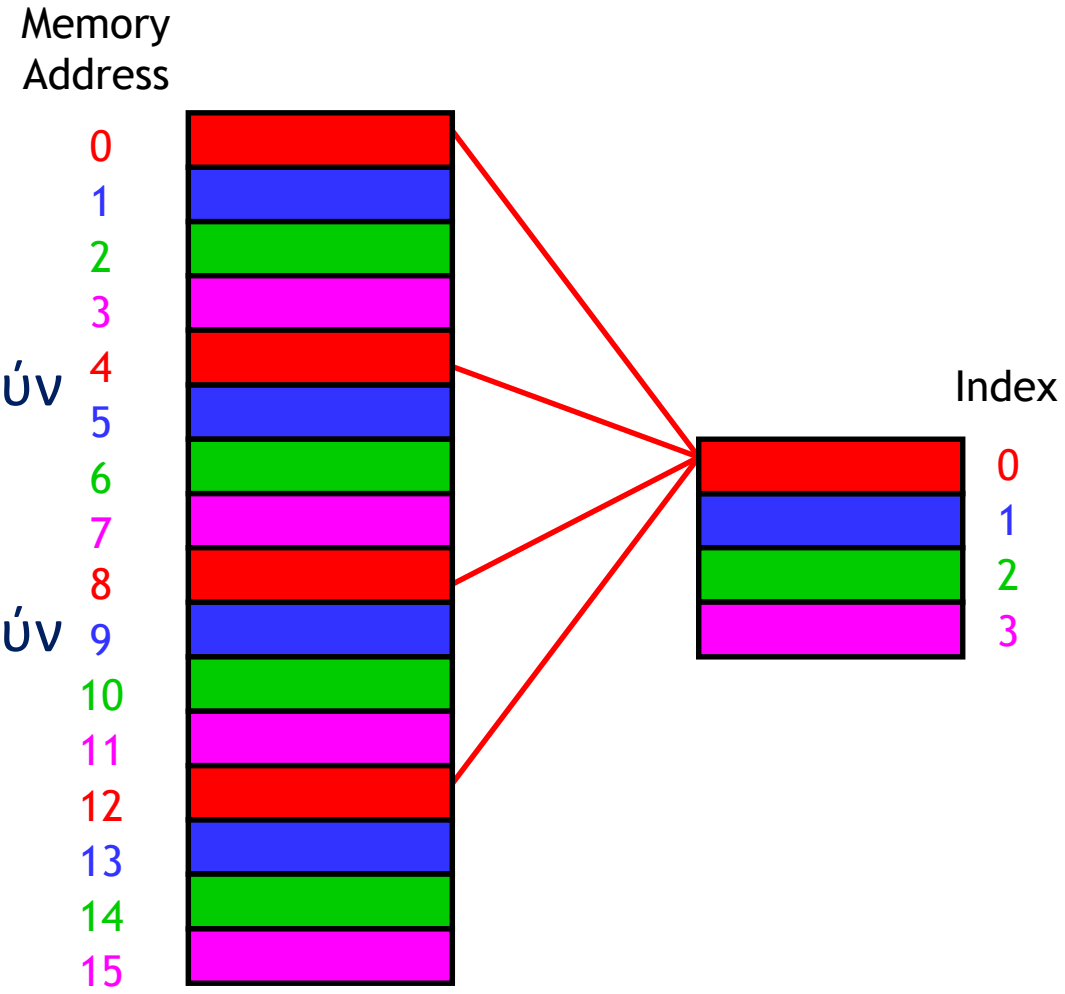
Τέσσερεις σημαντικές ερωτήσεις



1. Που ακριβώς στην cache τοποθετείται ένα block δεδομένων όταν αντιγράφεται από την κύρια μνήμη;
2. Πως ανιχνεύουμε εάν κάποια λέξη είναι ήδη στην cache, ή θα πρέπει να πρέπει να την φέρουμε από την κύρια μνήμη;
3. Τι κάνουμε όταν γεμίσει η cache με δεδομένα; Για να φέρουμε καινούργια δεδομένα από την κύρια μνήμη στην cache, πρέπει να διώξουμε ένα block από δεδομένα που είναι ήδη στην cache. Πώς επιλέγουμε αυτό το block;
4. Τι κάνουμε όταν θέλουμε να γράψουμε στην μνήμη;

1. Που τοποθετούνται νέα δεδομένα

- Μία cache άμεσης απεικόνισης (direct-mapped) ακολουθεί την πιο απλή προσέγγιση: κάθε διεύθυνση μνήμης μπορεί να απεικονιστεί σε ένα και μόνο cache block.
- Πχ. Κύρια μνήμη 16 bytes, cache 4 bytes.
- Οι διευθύνσεις 0, 4, 8 και 12 μπορούν να τοποθετηθούν ΜΟΝΟ στο block 0 της cache.
- Οι διευθύνσεις 1, 5, 9 και 11 μπορούν να τοποθετηθούν ΜΟΝΟ στο block 1 της cache.



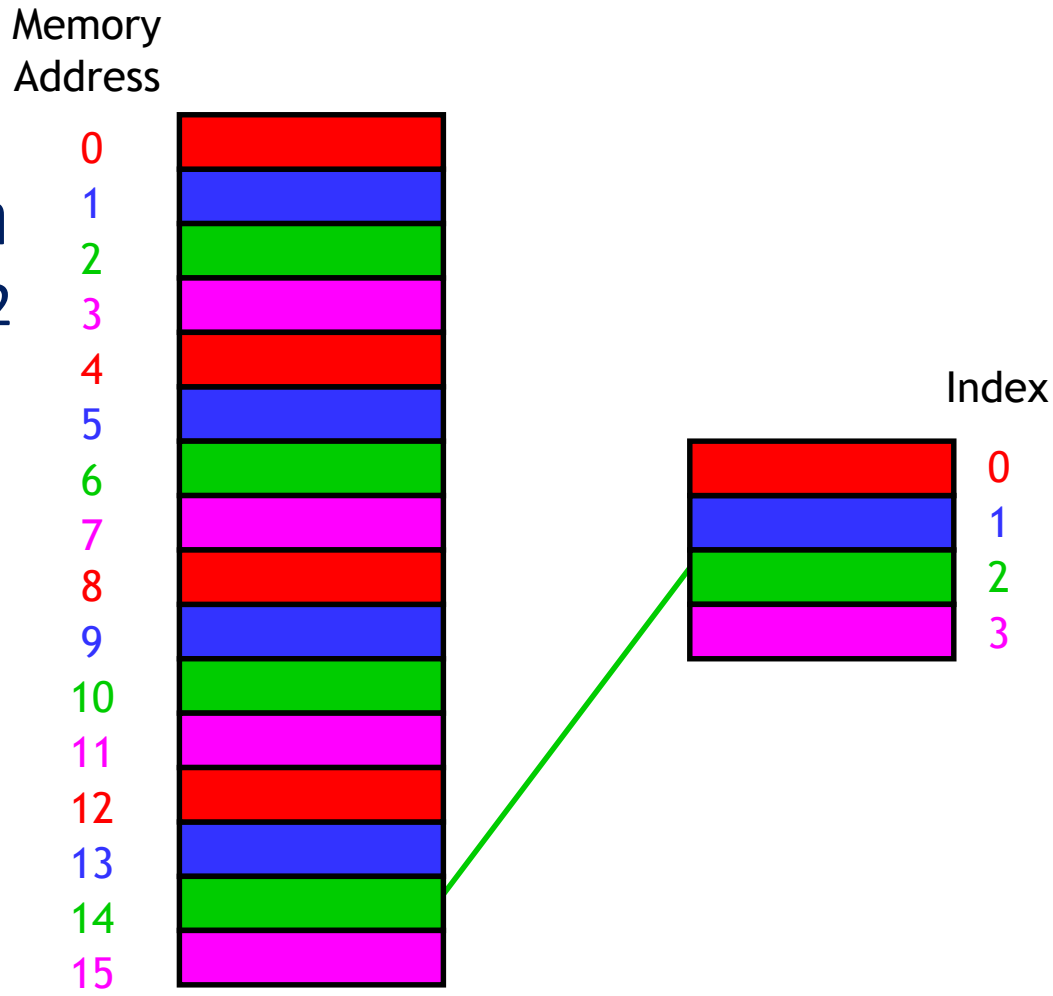
1. Που τοποθετούνται νέα δεδομένα

- Σε μια direct-mapped μνήμη cache μεγέθους 2^k blocks, η διεύθυνση *i* θα τοποθετηθεί στο cache block

  $i \bmod 2^k$

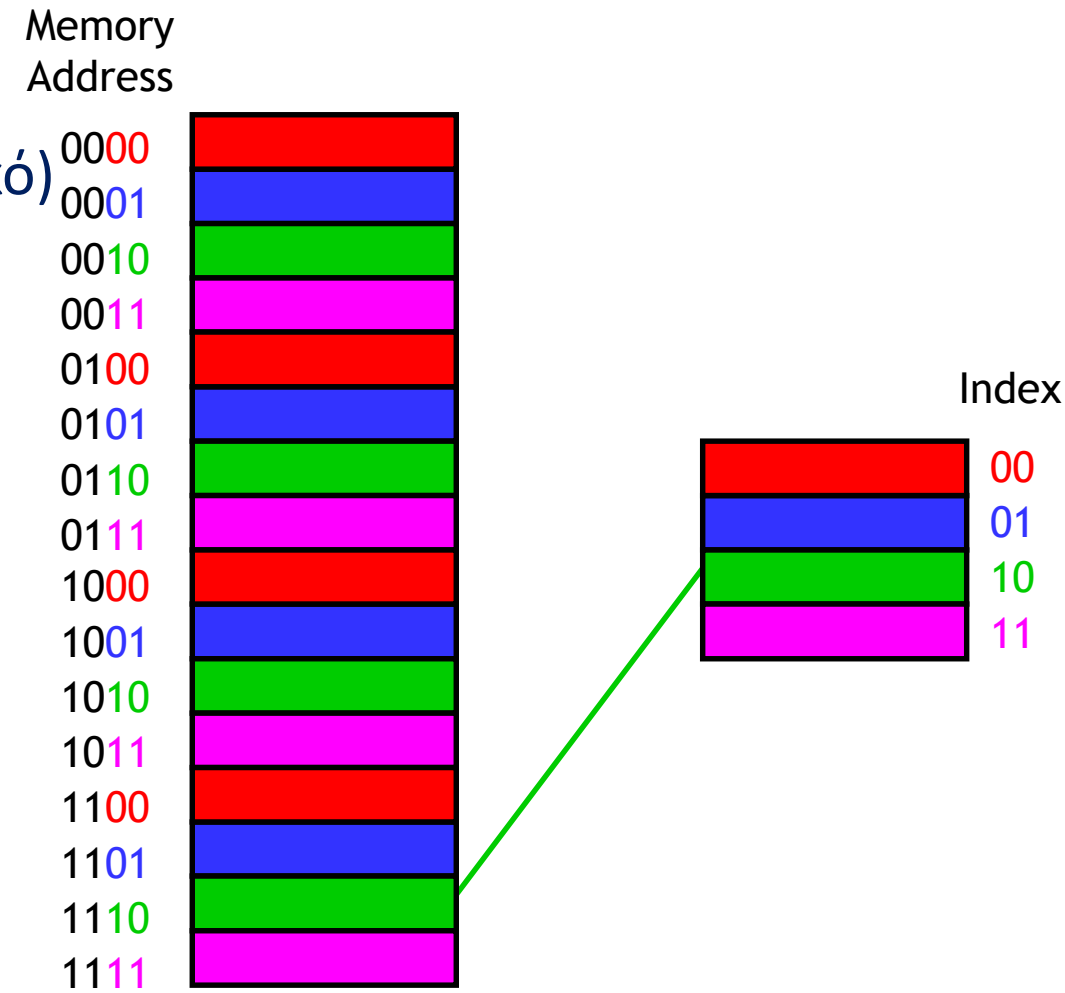
- Για παράδειγμα, σε μια cache με 4 blocks η διεύθυνση 14 θα τοποθετηθεί στο block 2

  $14 \bmod 4 = 2$



1. Που τοποθετούνται νέα δεδομένα

- Ένας απλός τρόπος να υπολογίσουμε το i mod 2^k είναι να κοιτάξουμε τα τελευταία k bits της διεύθυνσης i .
- Για την 4-bit cache, ελέγχουμε τα τελευταία 2 bits.
- Η διεύθυνσης 14 (1110 δυαδικό) θα τοποθετηθεί στο cache block 2 (10 in binary).

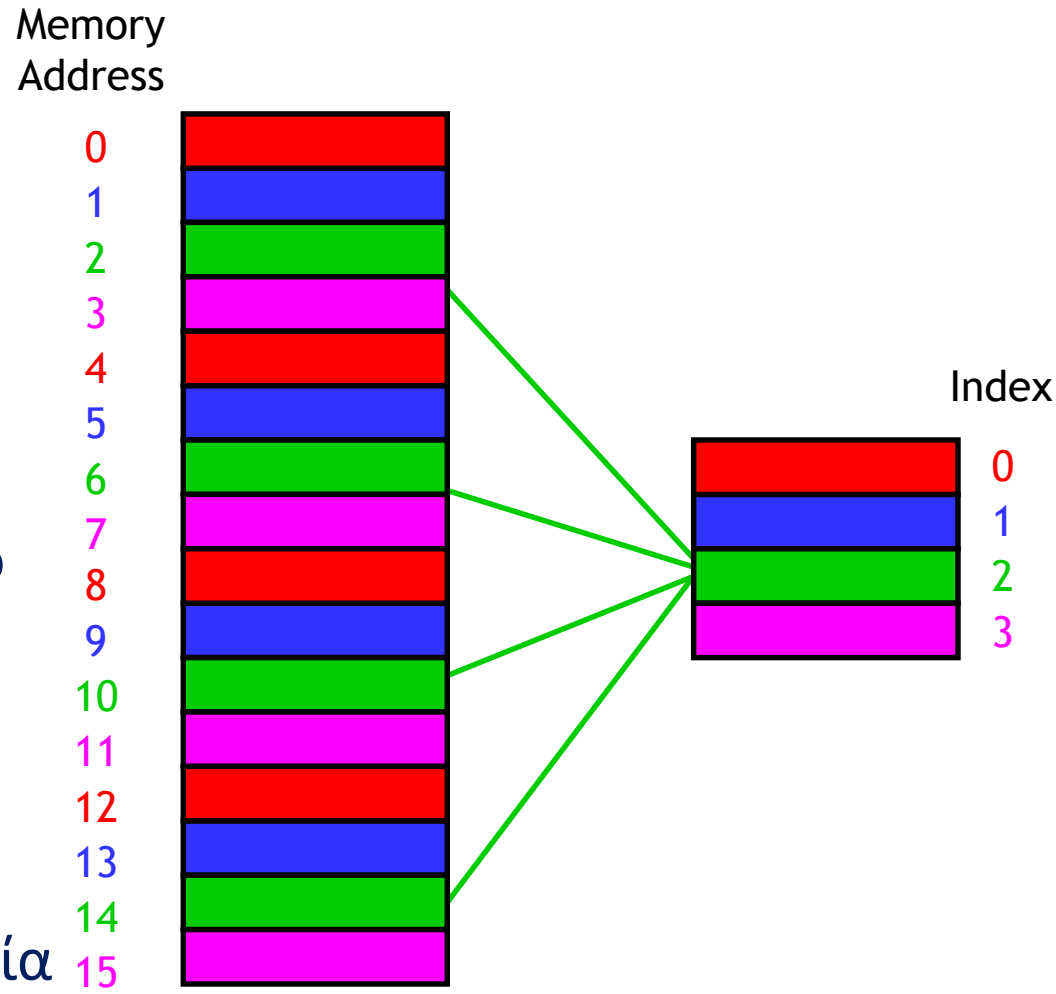


2. Πως βρίσκουμε δεδομένα στην cache

- Πως ανιχνεύουμε εάν κάποια λέξη είναι ήδη στην cache;
- Το πρώτο βήμα για να βρούμε εάν η διεύθυνση *i* είναι στην cache είναι να υπολογίσουμε το block όπου θα πρέπει να βρίσκεται η διεύθυνση *i*

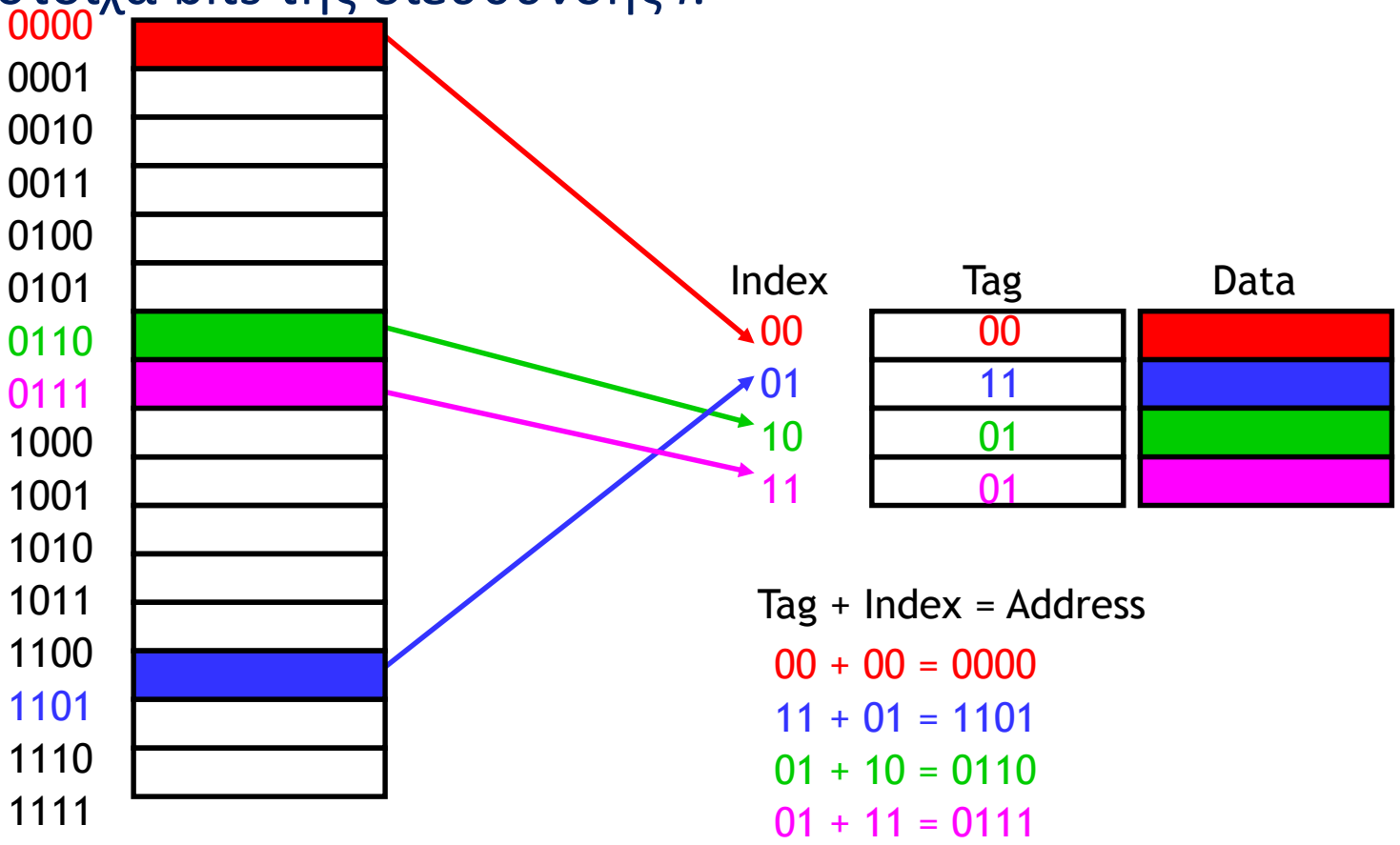
  $i \bmod 2^k$

- Όμως πολλές άλλες διευθύνσεις μπορούν να αποθηκευθούν στο ίδιο cache block.
- Για παράδειγμα, το cache block 2 μπορεί να περιέχει δεδομένα από τις διευθύνσεις 2, 6, 10 *or* 14.
- Άρα χρειάζεται επιπλέον πληροφορία



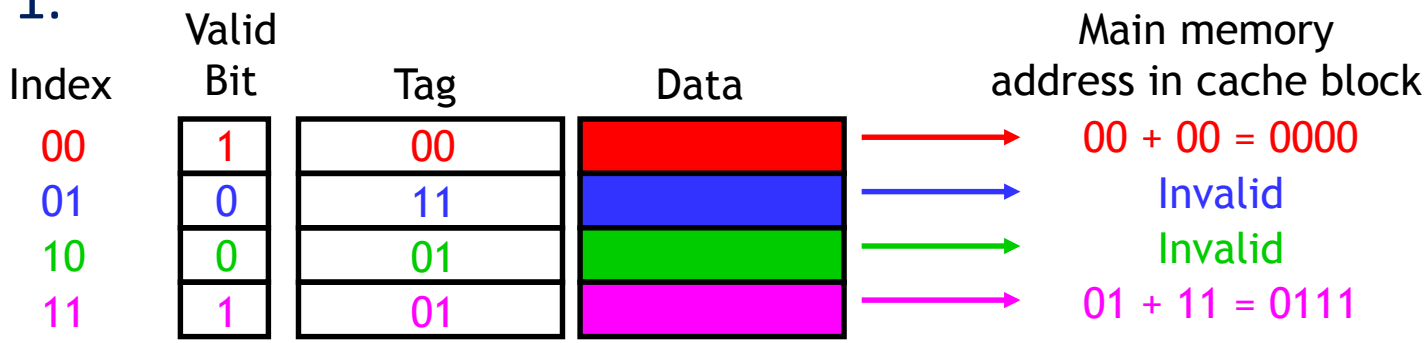
2. Πως βρίσκουμε δεδομένα στην cache; Tags!

- Χρησιμοποιούμε ως tags τα υπόλοιπα bits της διεύθυνσης για να ξεχωρίσουμε μεταξύ διαφορετικών θέσεων μνήμης που τυχαίνει να πέφτουν στο ίδιο cache block.
- Για να έχουμε match, πρέπει το tag στην cache να είναι το ίδιο με τα αντίστοιχα bits της διεύθυνσης *i*.



2. Πως βρίσκουμε δεδομένα στην cache; Valid Bit

- Όταν ξεκινάει να τρέχει ένα πρόγραμμα, η cache είναι άδεια και περιέχει άκυρα δεδομένα (invalid data).
- Για να ξεχωρίσουμε λοιπόν μεταξύ έγκυρων και άκυρων δεδομένων, προσθέτουμε ένα valid bit σε κάθε cache block.
- Όταν το σύστημα αρχικοποιείται όλα τα valid bits γίνονται 0.
- Όταν δεδομένα φορτώνονται σε ένα cache block, το αντίστοιχο valid bit γίνεται 1.
- Για να έχουμε match, πρέπει το tag στην cache να είναι το ίδιο με τα αντίστοιχα bits της διεύθυνσης *i* ΚΑΙ το valid bit του cache block να είναι 1.



Παράδειγμα λειτουργίας Cache (I)

- 8-blocks, 1 word/block, direct mapped
- Αρχική κατάσταση της cache:

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

Παράδειγμα λειτουργίας Cache (II)

Word addr	Binary addr (5 bits)	Hit/miss	Cache block
22	10 110	Miss	110

Block Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[22]**
111	N		

Παράδειγμα λειτουργίας Cache (III)

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

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[26]**
011	N		
100	N		
101	N		
110	Y	10	Mem[22]
111	N		

Παράδειγμα λειτουργίας Cache (IV)

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[26]
011	N		
100	N		
101	N		
110	Y	10	Mem[22]
111	N		

Παράδειγμα λειτουργίας Cache (V)

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

Index	V	Tag	Data
000	**Y**	**10**	**Mem[16]**
001	N		
010	Y	11	Mem[26]
011	**Y**	**00**	**Mem[3]**
100	N		
101	N		
110	Y	10	Mem[22]
111	N		

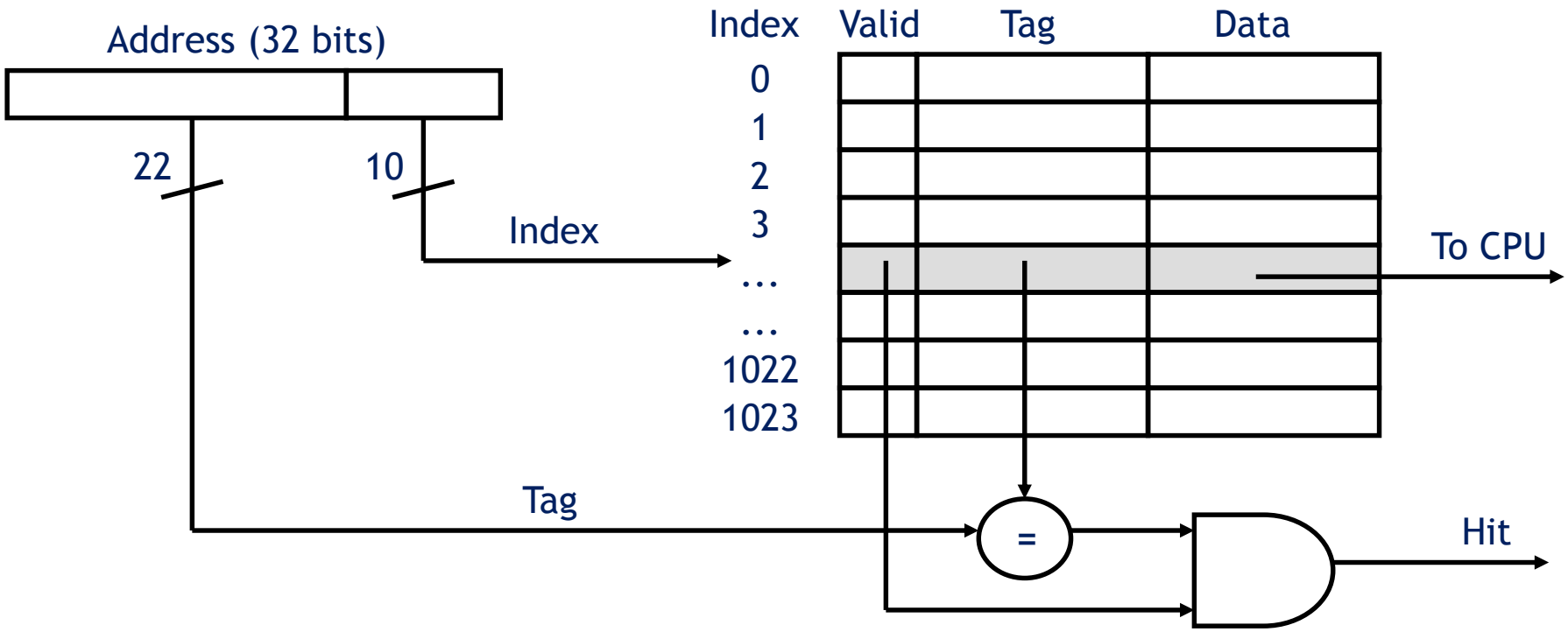
Παράδειγμα λειτουργίας Cache (VI)

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

Index	V	Tag	Data
000	Y	10	Mem[16]
001	N		
010	**Y**	**10**	**Mem[18]**
011	Y	00	Mem[3]
100	N		
101	N		
110	Y	10	Mem[22]
111	N		

Μικρο-αρχιτεκτονική της cache

- Όταν η CPU διαβάζει από την μνήμη, η διεύθυνση στέλνεται καταρχάς στον ελεγκτή της cache (cache controller).
  - Τα λιγότερο σημαντικά *k* bits της διεύθυνσης χρησιμοποιούνται σαν block index στην cache.
  - Εάν το block είναι έγκυρο (valid bit = 1) και το πεδίο tag είναι το ίδιο με τα περισσότερο σημαντικά (*m* − *k*) bits της m-bit της διεύθυνσης, τότε έχουμε cache hit και τα δεδομένα στέλνονται στην CPU

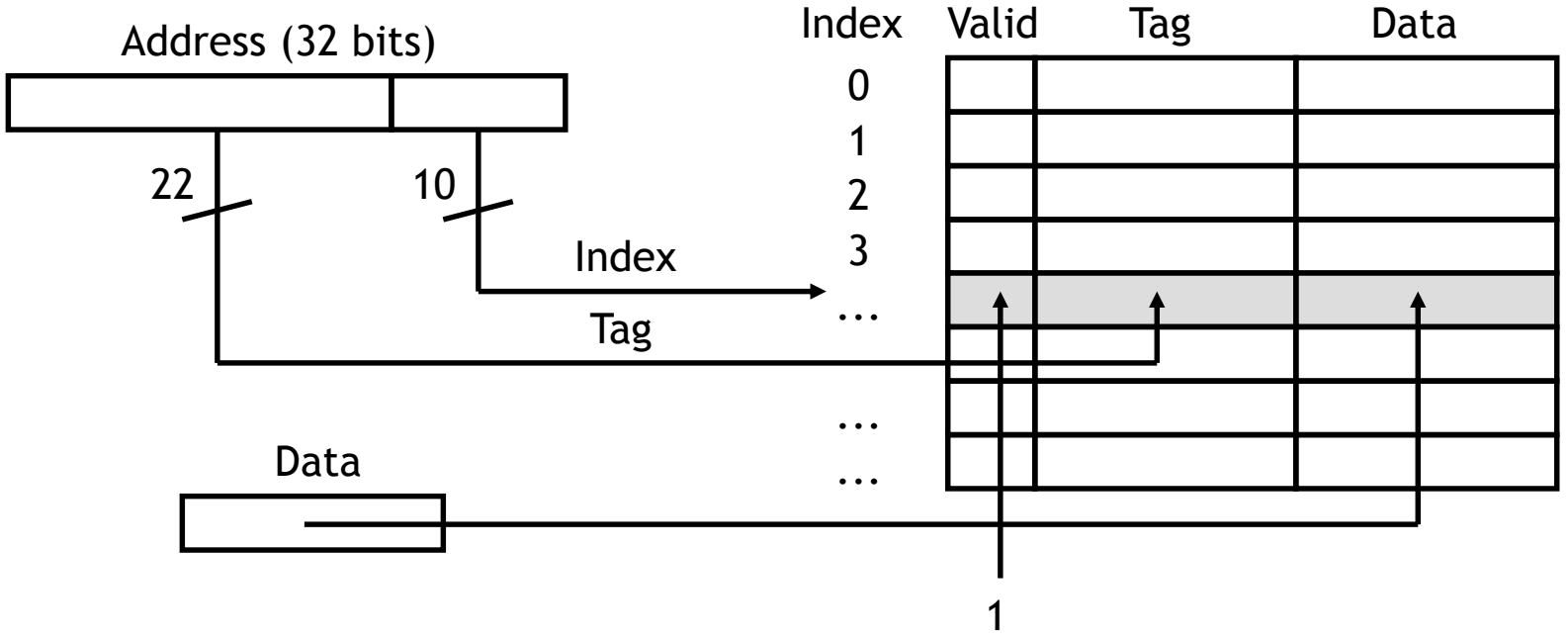


Cache Misses

- Μιας και οι μνήμες cache είναι πολύ μικρότερες από την κύρια μνήμη, είναι αναπόφευκτο να υπάρχουν φορές που δεν βρίσκουμε τα δεδομένα στην cache (cache miss).
  - Η κύρια μνήμη είναι μερικά GB, ενώ μια τυπική cache 16-32 KB
- Το ποσοστό των cache hits ανάμεσα σε όλες τις προσπελάσεις τις cache, λέγεται hit rate. Το ποσοστό των cache misses λέγεται miss rate.
- Σε περίπτωση ενός cache miss, η CPU φέρνει τα δεδομένα από την κύρια μνήμη και ταυτόχρονα τα μεταφέρει και στην cache.
- Θεωρούμε για απλούστευση ότι μέχρι να έρθουν τα δεδομένα στην cache η CPU κάνει stall. Αυτό το stall μπορεί να διαρκέσει δεκάδες ή και εκατοντάδες κύκλους μηχανής.

Cache Misses

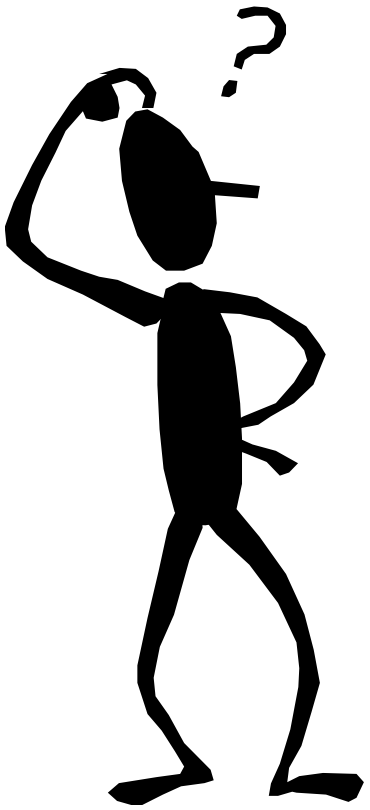
- Τα δεδομένα που έρχονται από την μνήμη μεταφέρονται και στην cache ως εξής:
  - Τα λιγότερο σημαντικά *k* bits της διεύθυνσης χρησιμοποιούνται σαν block index στην cache.
  - The περισσότερο σημαντικά (*m* − *k*) bits της διεύθυνσης αποθηκεύονται στο πεδίο tag.
  - Το δεδομένα από την κύρια μνήμη αποθηκεύονται στο πεδίο Data.
  - Θέτουμε το valid bit του cache block ίσο με 1.



3. Τί γίνεται εάν γεμίσει η cache

- Τι κάνουμε όταν γεμίσει η cache με δεδομένα; Για να φέρουμε καινούργια δεδομένα από την κύρια μνήμη στην cache, πρέπει να διώξουμε ένα block από δεδομένα που είναι ήδη στην cache. Πώς επιλέγουμε αυτό το block;
- Η απάντηση είναι πολύ απλή.
  - Ένα cache miss μεταφέρει καινούργια δεδομένα από την κύρια μνήμη στην cache. Τα δεδομένα αυτά διαγράφουν οτιδήποτε ήταν στην cache στο ίδιο block.

Τέσσερεις σημαντικές ερωτήσεις



1. Που ακριβώς στην cache τοποθετείται ένα block δεδομένων όταν αντιγράφεται από την κύρια μνήμη;
2. Πως ανιχνεύουμε εάν κάποια λέξη είναι ήδη στην cache, ή θα πρέπει να πρέπει να την φέρουμε από την κύρια μνήμη;
3. Τι κάνουμε όταν γεμίσει η cache με δεδομένα; Για να φέρουμε καινούργια δεδομένα από την κύρια μνήμη στην cache, πρέπει να διώξουμε ένα block από δεδομένα που είναι ήδη στην cache. Πώς επιλέγουμε αυτό το block;
4. **Τι κάνουμε όταν θέλουμε να γράψουμε στην μνήμη;**

4. Γράφοντας στην cache

- Η πρώτη περίπτωση είναι να γράψουμε σε μια διεύθυνση που είναι ήδη στην cache (write hit). Θεωρείστε μια direct-mapped cache.

Index	V	Tag	Data
…			
110	1	11010	42803
…			

Address	Data
…	
1101 0110	42803
…	

- Έστω ότι θέλουμε να γράψουμε νέα δεδομένα στην cache στην διεύθυνση 1101 0110. Υπάρχουν δύο στρατηγικές οργάνωσης της cache σε περίπτωση write hit.

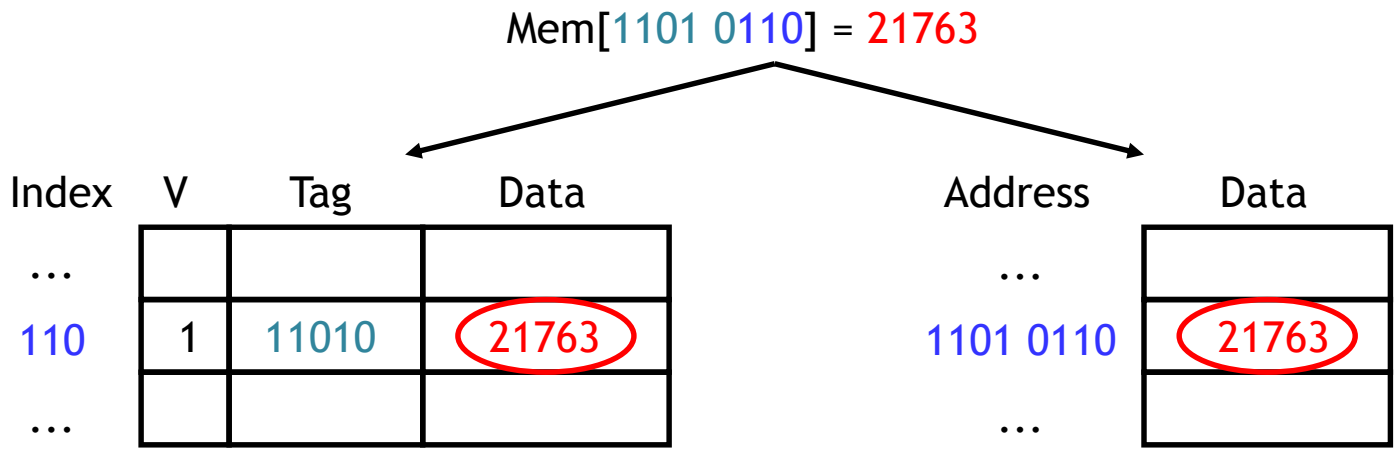
Mem[1101 0110] = 21763

Index	V	Tag	Data
…			
110	1	11010	21763
…			

Address	Data
…	
1101 0110	42803
…	

Write-through caches

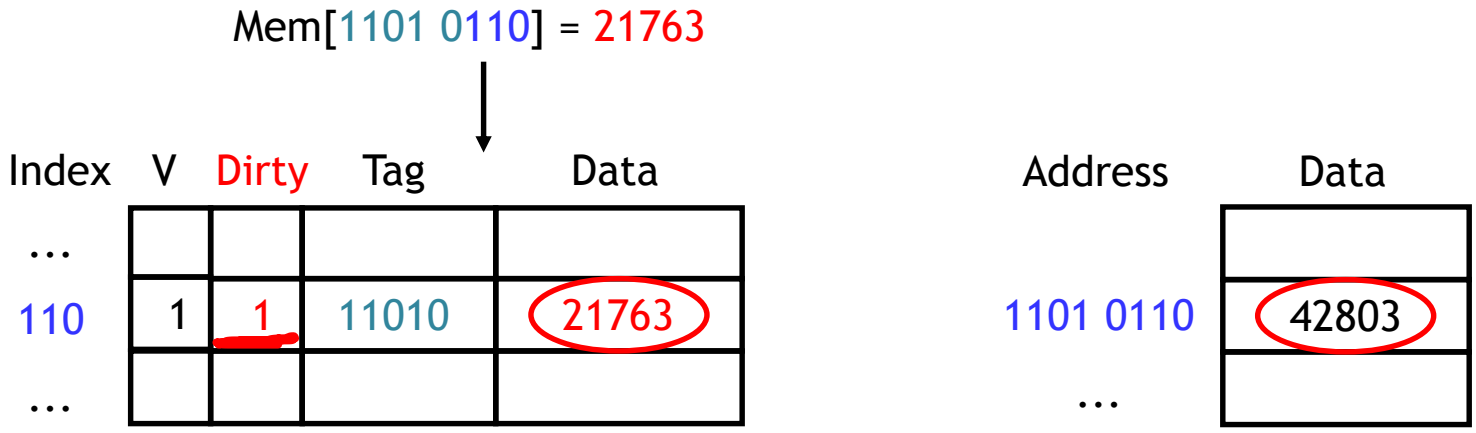
- Μια write-through cache γράφει τα δεδομένα εκτός από την cache ΚΑΙ στην μνήμη, έτσι ώστε η κύρια μνήμη να είναι πάντα ενημερωμένη με τα τελευταία δεδομένα.



- Αυτό είναι μια απλή λύση, αν και συνήθως δημιουργεί υψηλό overhead
- Δημιουργεί την ανάγκη να στέλνουμε συνέχεια δεδομένα στην κύρια μνήμη πολλές φορές χωρίς λόγο.

Write-back caches

- Σε μια write-back cache γράφουμε μόνο στην cache αμέσως αλλά όχι στην κύρια μνήμη.
- Η κύρια μνήμη ανανεώνεται με τα πιο πρόσφατα δεδομένα μόνο όταν το cache block το οποίο γράψαμε αντικαθίσταται και πρέπει να γραφεί πίσω στην μνήμη.
- Αυτό σημαίνει ότι για κάποιο χρονικό διάστημα το δεδομένα της κύριας μνήμης δεν είναι τα σωστά (πρόβλημα ασυνέπειας μνήμης = inconsistency)
- Κάθε μελλοντικό διάβασμα (read) από το ίδιο block θα εξυπηρετηθεί από την cache.
- Χρειάζεται ένα extra bit (dirty bit) για κάθε block στην cache. Γίνεται 1 όταν η CPU γράψει οποιαδήποτε byte στο block αυτό.
  - Μόνο όταν dirty bit = 1, χρειάζεται να ανανεωθεί η κύρια μνήμη σε περίπτωση αντικατάστασης ενός block.

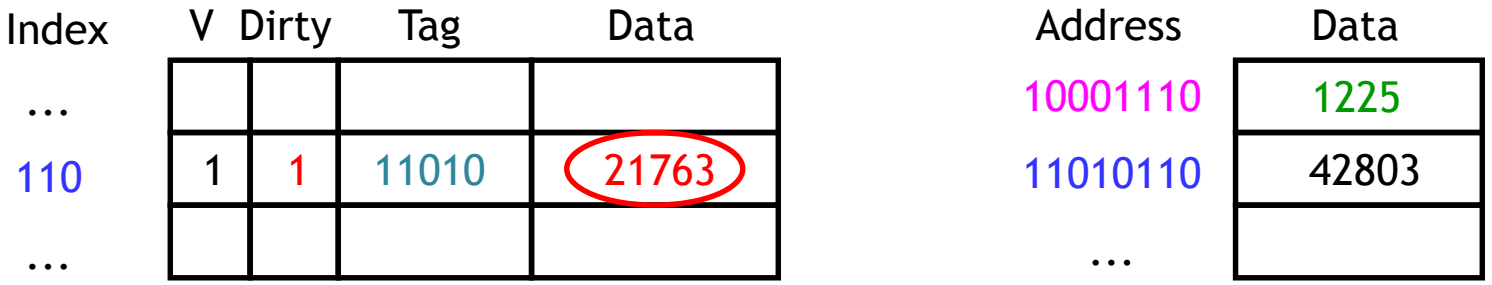


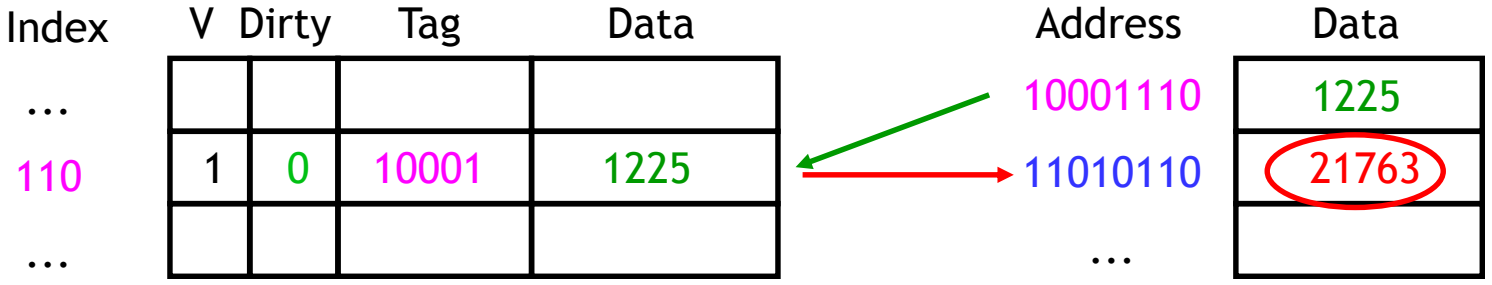
Write-back caches

- Παράδειγμα λειτουργίας μιας write-back cache στις 2 παρακάτω πράξεις

Mem[11010 110] = 21763 // Write Hit

Read Mem[10001 110] // Read Miss

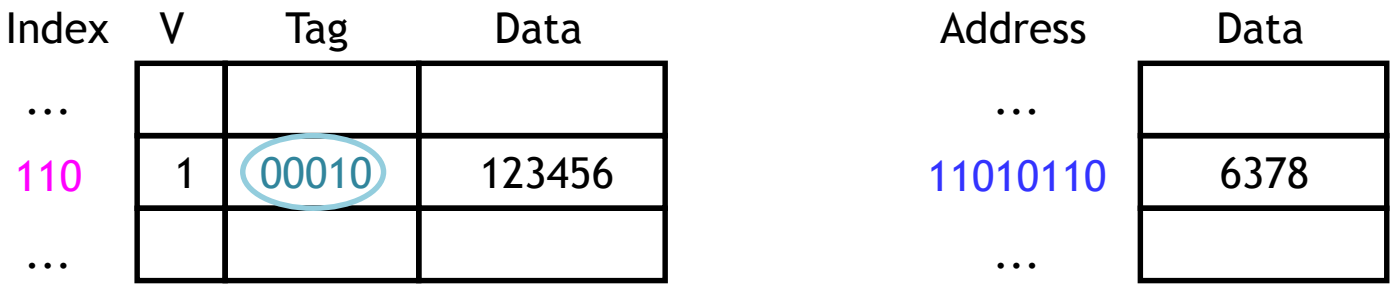




Write misses

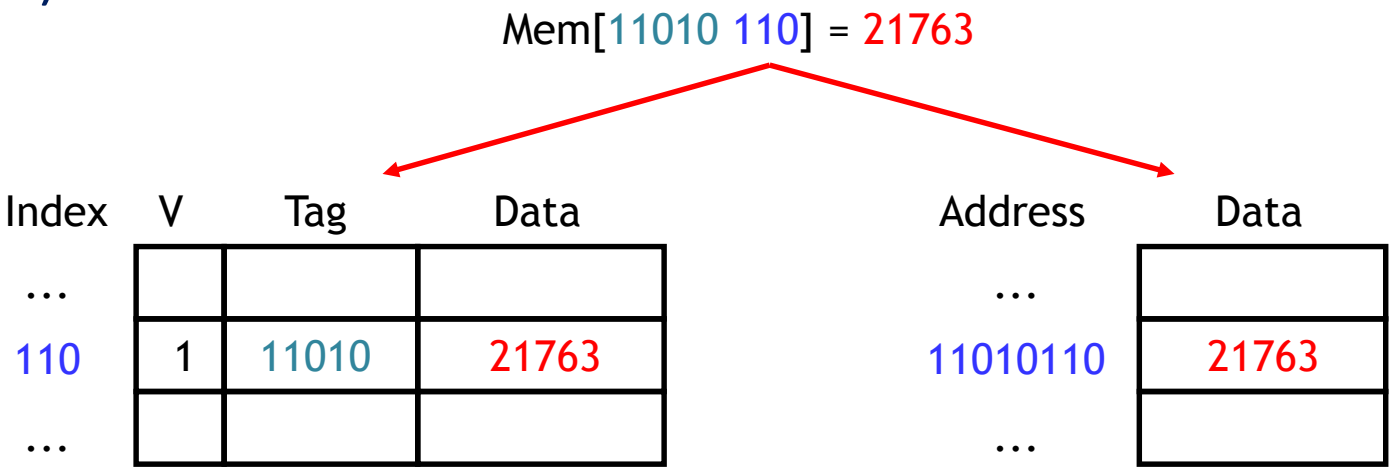
- Η δεύτερη περίπτωση είναι να γράψουμε σε μια διεύθυνση που δεν είναι στην cache (write miss). Θεωρείστε μια direct-mapped cache.
- Έστω ότι θέλουμε να γράψουμε:

  Mem[11010 110] = 21763 // Write Miss
- Πως αντιμετωπίζουμε αυτήν την περίπτωση;



Write Allocate

- Μία στρατηγική είναι να προσκομίσουμε πρώτα το block από την κύρια μνήμη και στην συνέχεια να το γράψουμε στην cache (write allocate).



- Τα δεδομένα θα παρέχονται από την cache εάν τα χρειαστούμε πάλι.
- Αυτή η στρατηγική χρησιμοποιείται κυρίως σε συνδυασμό με write-back caches.

No Write Allocate

Λειτουργεί καλά αυτή η στρατηγική για τον παρακάτω κώδικα;

```
for (int i = 0; i < LARGE; i++)
    a[i] = i;
```

Όχι, δεν έχει νόημα να βάζουμε στην cache δεδομένα όπως ο πίνακας a[.] που δεν θα ξαναχρησιμοποιηθούν για μεγάλο χρονικό διάστημα.

- Η δεύτερη στρατηγική είναι να γράψουμε απευθείας στην κύρια μνήμη, χωρίς να τοποθετήσουμε τα δεδομένα στην cache (no write allocate).