ΗΥ-225: Οργάνωση Υπολογιστών
Άνοιξη 2020

Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Σειρά Ασκήσεων 11: Κρυφές Μνήμες και η Επίδοσή τους

Παράδοση έως Κυριακή 3 Μαΐου 2020 (βδ. 11.0) ώρα 23:59 (από βδ. 9.1)

Διαφάνειες: χρησιμοποιήσαμε τις διαφάνειες που σημειώνει η κάθε παράγραφος παρακάτω από το Αγγλικό βιβλίο γιά το κεφάλαιο 5, που βρίσκονται (PPT) στο: www.elsevier.com/__data/assets/powerpoint_doc/0005/273713/Chapter_05-RISC-V.ppt
Βιβλίο (4η έκδοση, Ελληνικό): διαβάστε το πρώτο ήμισυ του κεφαλαίου 5 –§ 5.1 έως και 5.3, σελ. 524-570. Διαβάστε ιδιαίτερα προσεκτικά τις σελίδες 524-542 και 551-564.

Άσκηση 11.1: Μέσος Χρόνος Προσπέλασης Ιεραρχίας Μνήμης

Δείτε τις **Διαφάνειες** 2-5 και 38 του (Αγγλικού) βιβλίου.
Οπως είδαμε στο μάθημα, γιά τον υπολογισμό των επιδόσεων μιάς ιεραρχίας μνήμης (π.χ. κρυφή μνήμη (cache memory) με κύρια μνήμη (main memory)) ορίζουμε τις εξής παραμέτρους:

- *hit_ratio* (ποσοστό ευστοχίας): τι ποσοστό του συνολικού πλήθους προσπελάσεων στην ιεραρχία μνήμης ήταν εύστοχες προσπελάσεις.
- *miss_ratio* (ποσοστό αστοχίας): τι ποσοστό του συνολικού πλήθους προσπελάσεων στην ιεραρχία μνήμης ήταν άστοχες προσπελάσεις. Προφανώς, *hit_ratio + miss_ratio = 1*.
- t_{hit} (hit time - χρόνος ευστοχίας): χρόνος εύστοχης προσπέλασης, δηλαδή πόση ώρα παίρνει μιά προσπέλαση που βρίσκει αυτό που ζητά στην μικρή και γρήγορη μνήμη (π.χ. στην κρυφή μνήμη).
- t_{miss} (miss time - χρόνος αστοχίας): χρόνος άστοχης προσπέλασης, δηλαδή πόση ώρα παίρνει (συνολικά) μιά προσπέλαση που δεν βρίσκει αυτό που ζητά στην μικρή και γρήγορη μνήμη (π.χ. στην κρυφή μνήμη), και αναγκάζεται να το διαβάσει από την μεγάλη και αργή μνήμη (π.χ. κύρια μνήμη).
- $t_{miss\_penalty} = t_{miss} - t_{hit}$ (miss penalty - επιπλέον κόστος της αστοχίας): πόση ώρα παραπάνω μας κοστίζει η αστοχία από την ευστοχία.

Τότε, ο μέσος χρόνος προσπέλασης στην ιεραρχία μνήμης, t_{eff} (effective access time), θα είναι, προφανώς:

$$t_{eff} = hit\_ratio * t_{hit} + miss\_ratio * t_{miss} = t_{hit} + miss\_ratio * t_{miss\_penalty}$$

Άσκηση: Θεωρήστε μιά κρυφή μνήμη με ποσοστό αστοχίας δυόμισι (2.5) %, χρόνο ευστοχίας 1 κύκλο ρολογιού, και κόστος αστοχίας (miss penalty) 40 κύκλους ρολογιού. Πόσος είναι ο μέσος χρόνος προσπέλασης σε αυτή την ιεραρχία μνήμης (σε κύκλους ρολογιού);

Άσκηση 11.2: Επίδοση Επεξεργαστή με Ιεραρχία Μνήμης

Δείτε τις **Διαφάνειες** 36, 37, και 39 του (Αγγλικού) βιβλίου.
Το CPI των ασκήσεων 10.2 - 10.4 θεωρούσε ότι όλες οι προσπελάσεις μνήμης (instruction fetch, data load, data store) είναι εύστοχες, δηλαδή βρίσκουν αυτό που ζητούν στην κρυφή μνήμη (η οποία έχει χρόνο προσπέλασης 1 κύκλο ρολογιού). Δυστυχώς, η πραγματικότητα διαφέρει σημαντικά από αυτή την ιδανική εικόνα (και, δυστυχώς, τέτοιους εξιδανικευμένους και εξωπραγματικούς αριθμούς επιδόσεων δημοσιεύουν μερικές φορές εταιρείες, γιά ευνόητους λόγους...). Ας δούμε πόσο μπορεί να χειροτερέψουν τα πράγματα με μιά μη ιδανική μνήμη.

Θεωρήστε έναν επεξεργαστή RISC με ομοχειρία, μ' ένα "ιδανικό" CPI (CPI με ιδανική μνήμη) (κατ' αναλογία με όσα συζητούσαμε στην άσκηση 10.3) ίσο με 1.3 . Θεωρήστε ότι

αυτός τρέχει ένα πρόγραμμα που απαιτεί την εκτέλεση 1 εκατομμυρίου εντολών.
(α) Πόσους κύκλους ρολογιού θα χρειάζονταν αυτό το πρόγραμμα γιά να τρέξει αν η ιεραρχία μνήμης ήταν ιδανική (όλες οι προσπελάσεις εύστοχες);

Ας υπολογίσουμε τώρα πόσες προσπελάσεις μνήμης κάνει αυτό το πρόγραμμα με μη ιδανική ιεραρχία μνήμης. Κάθε έντολη κάνει μία προσπέλαση μνήμης γιά i_fetch, και, επιπλέον, οι εντολές load και store κάνουν μία ακόμα προσπέλαση μνήμης καθεμία.
(β) Θεωρήστε ότι κατά την εκτέλεση του παραπάνω προγράμματος 25% των εκτελούμενων εντολών είναι load και 15% είναι store. Υπολογίστε το πλήθος προσπελάσεων που κάνει αυτό το πρόγραμμα στην κρυφή μνήμη εντολών (ICache) (γιά προσκόμιση εντολών - IFetch) και το πλήθος των προσπελάσεων που αυτό κάνει στην κρυφή μνήμη δεδομένων (DCache) (γιά αναγνώσεις δεδομένων από εντολές load ή γιά εγγραφές δεδομένων από εντολές store).

Από τις προσπελάσεις αυτές (β), άλλες είναι εύστοχες, και άλλες άστοχες. Με τις εύστοχες προσπελάσεις, τα πράγματα έχουν όπως και στον ιδανικό υπολογιστή (α). Κάθε άστοχη προσπέλαση, όμως, μας κοστίζει miss_penalty κύκλους ρολογιού επιπλέον αυτών που ξοδεύει ο ιδανικός υπολογιστής.
(γ) Πόσες άστοχες προσπελάσεις στην κρυφή μνήμη εντολών κάνει το παραπάνω πρόγραμμα, εάν το miss_ratio αυτής της ICache είναι 2.0 % και πόσες άστοχες προσπελάσεις κάνει αυτό στην κρυφή μνήμη δεδομένων, εάν το miss_ratio της DCache είναι 5.0 % ;
(δ) Πόσους κύκλους ρολογιού (επιπλέον του ιδανικού) μας κοστίζουν αυτές οι άστοχες προσπελάσεις, εάν το miss_penalty είναι 16 κύκλοι ρολογιού; (το ίδιο, 16 κύκλοι, από οιαδήποτε από τις δύο αυτές κρυφές μνήμες πρώτου επιπέδου, L1 Caches, μέχρι την κρυφή μνήμη δευτέρου επιπέδου, L2 Cache, που είναι κοινή γιά εντολές και γιά δεδομένα, και που ας θεωρήσουμε εδώ, γιά απλότητα, ότι ποτέ δεν αστοχεί...) (μιά άλλη υπόθεση που κάνουμε εδώ είναι ότι, όσο διαρκεί ο εξυπηρέτηση μιάς αστοχίας, ο υπόλοιπος επεξεργαστής "κάθεται άπραγος", μη κάνοντας τίποτα χρήσιμο, όσο περιμένει να τελειώσει ο χειρισμός της αστοχίας (πράγμα που δεν είναι αλήθεια σε πολλούς σημερινούς προχωρημένους επεξεργαστές)).

Ο πραγματικός υπολογιστής, λοιπόν, θα αργεί περισσότερο από τον ιδανικό (α) κατά τόσους κύκλους ρολογιού όσοι χάθηκαν λόγω άστοχων προσπελάσεων (δ). Επομένως,
(ε) πόσους κύκλους ρολογιού θα χρειαστεί ο πραγματικός υπολογιστής γιά να τρέξει το παραπάνω πρόγραμμα του 1 εκατομμυρίου εντολών;
(στ) Πόσο είναι, συνεπώς, το ισοδύναμο CPI του πραγματικού υπολογιστή;
(ζ) Πόσες φορές αργότερος είναι ο πραγματικός από τον ιδανικό υπολογιστή; (δηλαδή πόσες φορές ταχύτερος είναι ο ιδανικός από τον πραγματικό –δηλαδή $t_{\text{πραγματικός}}$ / $t_{\text{ιδανικός}}$).

Άσκηση 11.3:
Κρυφή Μνήμη Μονοσήμαντης (Άμεσης) Απεικόνισης (Direct Mapped)

Δείτε τις **Διαφάνειες** 17 έως και 26 του (Αγγλικού) βιβλίου, όπως και το σχήμα εδώ παρακάτω.
Όταν τοποθετούμε ένα αντίτυπο μιάς λέξης από την κύρια μνήμη στην κρυφή μνήμη, αυτό μπορεί να τοποθετηθεί σε **ορισμένες μόνο "επιτρεπτές" θέσεις** της κρυφής μνήμης. Εάν όλες οι θέσεις της κρυφής μνήμης είναι επιτρεπτές (γιά την κάθε λέξη της κύριας μνήμη), τότε η κρυφή μνήμη λέγεται **πλήρως προσεταιριστική** (fully associative cache). Το μειονέκτημα όμως της πλήρως προσεταιριστικής κρυφής μνήμης είναι το υψηλό της κόστος: γιά να ψάξουμε να βρούμε αν μιά επιθυμητή λέξη της κύριας μνήμης έχει αυτή τη στιγμή αντίτυπο της στην κρυφή μνήμη, ώστε να το διαβάσουμε από εκεί γρήγορα, πρέπει να ψάξουμε **σε όλες** τις θέσεις της κρυφής μνήμης, αφού η επιθυμητή λέξη, όταν είχε έλθει στην κρυφή μνήμη (αν είχε έλθει, και αν είναι ακόμα εκεί), ήταν επιτρεπτό να τοποθετηθεί σε οποιαδήποτε θέση της κρυφής μνήμης.

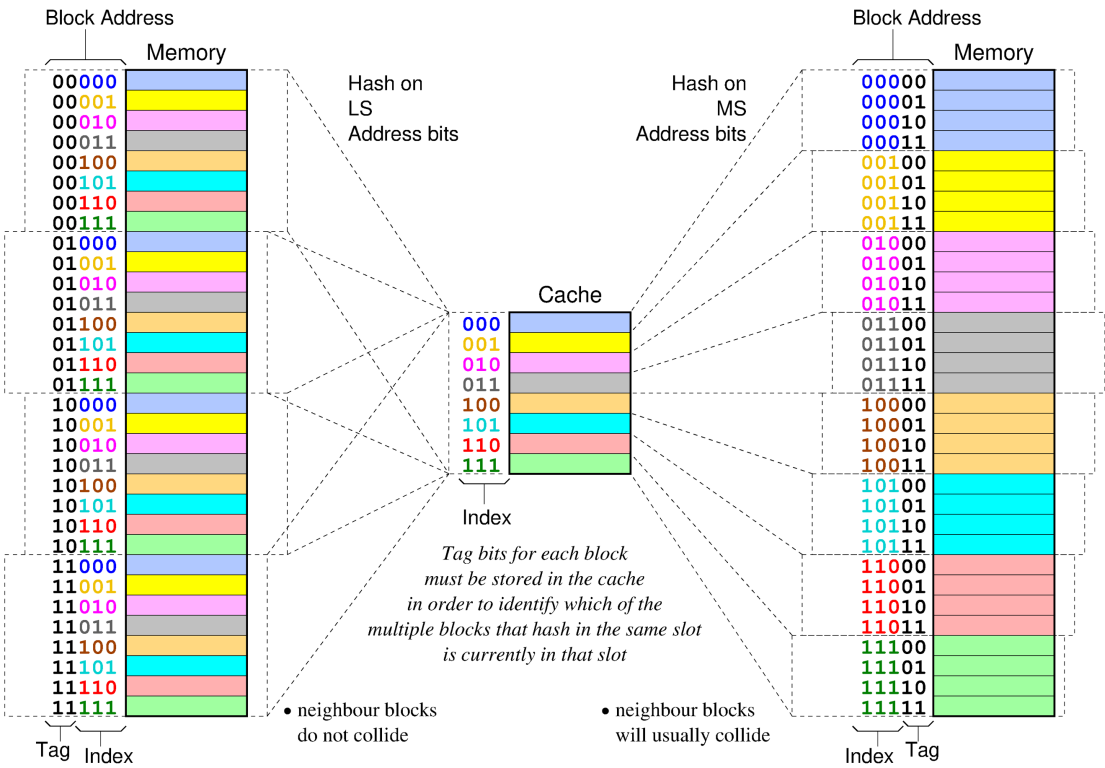
Η οργάνωση κρυφής μνήμης με το μικρότερο δυνατό κόστος είναι η **μονοσήμαντης (άμεσης) απεικόνισης** (direct mapped cache). Σε αυτή την οργάνωση, υπάρχει **μία μόνο επιτρεπτή θέση** στην κρυφή μνήμη γιά την κάθε λέξη της κύριας μνήμης. Έτσι, το ψάξιμο γιά μιά λέξη είναι απλό: πηγαίνουμε στην μοναδική επιτρεπτή θέση γιά αυτή τη λέξη, και κοιτάμε να δούμε ποιός βρίσκεται εκεί αυτή τη στιγμή: η επιθυμητή λέξη, άλλη λέξη, ή

καμία λέξη; (Η απάντηση δίδεται από το address tag και το valid bit, όπως είπαμε στο μάθημα).

Φυσικά, το πρόβλημα με την μονοσήμαντη απεικόνιση είναι ότι μόνο μία από τις πολλές λέξεις της κύριας μνήμης που έχουν όλες την ίδια επιτρεπτή θέση στην κρυφή μνήμη μπορεί να βρίσκεται στην κρυφή μνήμη σε οποιαδήποτε δεδομένη στιγμή –αν το πρόγραμμά μας τύχει να θέλει να δουλέψει (σχεδόν) "ταυτόχρονα" με δύο ή περισσότερες από αυτές τις λέξεις, τότε θα έχει συνεχείς αστοχίες: η μιά λέξη θα διώχνει την άλλη. Γιά να μειώσουμε όσο μπορούμε τις αρνητικές συνέπειες αυτού του φαινομένου, επιδιώκουμε η απεικόνιση των λέξεων της κύριας μνήμης στην κρυφή μνήμη –μιά συνάρτηση διασποράς (hashing), ουσιαστικά– να τις σπέρνει όσο πιό ομοιόμορφα μπορεί σε όλη την κρυφή μνήμη, και να τις σπέρνει χωρίς αρνητικές συσχετίσεις με τις συνηθισμένες ιδιότητες τοπικότητας των προγραμμάτων.

Οι συνηθισμένες τοπικότητες των προγραμμάτων είναι η **χρονική** (temporal - ίδια λέξη ξανά και ξανά), και η **χωρική** (spatial - γειτονικές λέξεις χρησιμοποιούνται "μαζί"). Γιά να μην έχει η απεικόνιση των λέξεων αρνητικές συσχετίσεις με την χωρική τοπικότητα, πρέπει **γειτονικές λέξεις** της κύριας μνήμης (που χρησιμοποιούνται "μαζί") να απεικονίζονται σε **διαφορετικές θέσεις** της κρυφής μνήμης (ώστε να μην διώχνει η μία την άλλη). Επιπλέον, η απεικόνιση των λέξεων πρέπει να είναι απλή, προκειμένου να υλοποιείται τάχιστα στο κύκλωμα. Η συνάρτηση διασποράς που έχει αυτές τις ιδιότητες και χρησιμοποιείται στις κρυφές μνήμες μονοσήμαντης απεικόνισης φαίνεται στο *αριστερό* μέρος του σχήματος, και είναι η εξής (όπου "modulo" είναι το υπόλοιπο της διαίρεσης –θυμηθείτε ότι διαίρεση διά δύναμη του 2 ισοδυναμεί με το να μοιράσουμε απλώς τα bits της λέξης στο κατάλληλο πλήθος MS bits (πηλίκο) και LS bits (υπόλοιπο)):

(θέση_κρυφής_μν) = (διεύθ_κύριας_μν) **modulo** (μέγεθος_κρυφής_μν)



Στο σχήμα, *Index* είναι η θέση της κρυφής μνήμης όπου πρέπει να μπεί δοθείσα λέξη από την κεντρική μνήμη, δηλαδή το αποτέλεσμα της εφαρμογής της συνάρτησης διασποράς πάνω στη διεύθυνση μνήμης της εν λόγω λέξης. Το χρώμα της κάθε λέξης μνήμης αντιστοιχεί στην θέση της κρυφής μνήμης όπου αυτή υποχρεούται να μπεί εάν θέλουμε να την φέρουμε στην κρυφή μνήμη. Το σχήμα αντιδιαστέλει την παραπάνω καλή συνάρτηση διασποράς (αριστερά στο σχήμα), που βασίζεται στα λιγότερο σημαντικά (LS) bits της διεύθυνσης, με κακές συναρτήσεις διασποράς, δεξιά στο σχήμα, εάν κοιτούσαμε άλλα, περισσότερο σημαντικά (MS) bits της διεύθυνσης. Όπως παρατηρούμε, εάν η συνάρτηση

διασποράς είναι τα LS bits της διεύθυνσης της λέξης, τότε διαδοχικές λέξεις μνήμης απεικονίζονται σε διαφορετικές (διαδοχικές) θέσεις της κρυφής μνήμης, μέχρι να χρησιμοποιηθούν όλες οι θέσεις της κρυφής μνήμης, και στη συνέχεια γυρίζουν πίσω (wrap around) και ξαναχρησιμοποιούν όλες τις θέσεις της κρυφής μνήμης μιά-μιά, κ.ο.κ. Αντίθετα, εάν η συνάρτηση διασποράς αγνοεί LS bits της διεύθυνσης της λέξης, τότε αυτό συνεπάγεται ότι *γειτονικές* λέξεις των οποίων οι διευθύνσεις διαφέρουν σε αυτά τα αγνοούμενα LS bits μόνον θα απεικονίζονται στην ίδια θέση της κρυφής μνήμης, δηλαδή θα συγκρούονται –θα διώχνουν η μία την άλλη– όπως φαίνεται από τη γειτνίαση στη μνήμη των λέξεων με το ίδιο χρώμα.

Το σχήμα, γιά απλότητα, θεωρεί ότι η κύρια μνήμη έχει μέγεθος μόνο 32 λέξεις (δηλαδή 128 Bytes, θεωρώντας 32-μπιτο datapath), και ότι η κρυφή μνήμη έχει μέγεθος μόνο 8 λέξεις (δηλ. 32 Bytes). Αφού η εδώ κρυφή μνήμη αποθηκεύει και ανακαλεί ολόκληρες *λέξεις* των 4 Bytes η κάθεμία, την κρυφή μνήμη *δεν* την ενδιαφέρουν τα 2 LS bits της πλήρους διεύθυνσης μνήμης (δηλ. της διεύθυνσης υπολογιστή Byte-Addressable), αλλά μόνον τα υπόλοιπα MS bits που είναι η διεύθυνση *λέξης*. Στο σχήμα, όλα τα ορθογώνια κουτάκια που φαίνονται είναι "blocks" –που εδώ τα θεωρούμε σαν μία λέξη το καθένα– και όλες οι διευθύνσεις που φαίνονται (και το Index της κρυφής μνήμης) είναι διευθύνσεις των blocks, δηλ. διευθύνσεις λέξεων εδώ. Εάν ο υπολογιστής μας και η κρυφή μνήμη μας ήταν 64-μπιτη, τότε η κάθε λέξη (block) θα ήταν 8 Bytes, και τότε το σχήμα θα έδειχνε μνήμη μεγέθους 32 λέξεων (blocks) = 256 Bytes (άρα με 8-μπιτες διευθύνσεις), κρυφή μνήμη μεγέθους 8 λέξεων (blocks) = 64 Bytes, και οι διευθύνσεις που φαίνονται στο σχήμα θα ήταν τα 5 MS bits από τα 8 συνολικά bits της πλήρους διεύθυνσης μνήμης, αφού τα 3 LS bits θα αφορούσαν το Byte μέσα στη λέξη (block) και άρα δεν θα ενδιέφεραν την κρυφή μνήμη. Το σχήμα μιλάει γιά blocks και block addresses γιά να μας προετοιμάσει γιά τις κρυφές μνήμες με μεγαλύτερο μέγεθος block (line) που θα δούμε στην §11.6 παρακάτω.

Άσκηση: Θεωρήστε τώρα μιά κρυφή μνήμη Μονοσήμαντης Απεικόνισης (Direct Mapped) ρεαλιστικού μεγέθους: Μέγεθος **Block (Line)** = 1 λέξη (word) = **8 Bytes** (64 bits), μέγεθος **Κρυφής Μνήμης = 64 KBytes** (πάντα, όποτε δίνουμε το μέγεθος μιάς κρυφής μνήμης, εννοούμε τα Data, *χωρίς* να περιλαμβάνουμε τα Tags), και μέγεθος **Κεντρικής Μνήμης = 4 GBytes** (δηλ. 32-μπιτες διευθύνσεις σε μνήμη Byte-Addressble).

(α) Πόσα blocks (lines, words) έχει η κεντρική μνήμη; Άρα η διεύθυνση του καθενός από αυτά τα blocks/lines/λέξεις πόσα bits πρέπει να έχει; Τα 32 bits της διεύθυνσης της μνήμης που είναι Byte-Addressable πώς χωρίζονται σε πόσα και ποιά bits της Block Address και πόσα και ποιά bits που επιλέγουν Byte μέσα στην λέξη/block/line?

(β) Πόσες θέσεις (blocks, lines, words) έχει η κρυφή μνήμη; Άρα το Index της κρυφής μνήμης (δηλ. η διεύθυνση επιλογής θέσης στην κρυφή μνήμη) πόσα bits πρέπει να έχει; Άρα, η συνάρτηση διασποράς που καθορίζει το Index αυτό πόσα και ποιά bits της Block Address (της κεντρικής μνήμης) πρέπει να είναι; Τα υπόλοιπα bits της Block Address, που αποτελούν το Tag, πόσα και ποιά είναι; Θυμίζουμε ότι γιά να βρούμε τη λέξη που θέλει ο επεξεργαστής, ψάχνουμε εκεί που μας λέει το Index. Οταν φτάσουμε εκεί, η επόμενη ερώτηση είναι εαν υπάρχει εκεί αντίτυπο κάποιας λέξης (block) μνήμης, εαν ναι ποιάς, και εαν είναι αυτής που εμείς ψάχνουμε. Το εαν υπάρχει αντίτυπο κάποιας λέξης μνήμης, το απαντάει το **Valid bit** (bit εγκυρότητας). Το ποιάς λέξης, το απαντάει το **address Tag** (ετικέτα διεύθυνσης).

(γ) Πόσες λέξεις (blocks/lines) της κύριας μνήμης απεικονίζονται στην ίδια θέση (διώχνουν η μία την άλλη), γιά κάθε μία θέση (λέξη/block/line) της κρυφής μνήμης; Ξέρουμε ότι αυτές είναι ομοιόμορφα διασπαρμένες στις θέσεις της κρυφής μνήμης, άρα η απάντηση πρέπει να ισούται με το πηλίκο του πλήθους των blocks της κεντρικής μνήμης διά το πλήθος των blocks (θέσεων) της κρυφής μνήμης. Το πλήθος που βρήκατε (blocks μνήμης ανά ίδια θέση κρυφής μνήμης) πώς σχετίζεται με το μέγεθος του Tag (πλήθος bits ανά Tag) και γιατί, βάσει της απάντησης (β); Παρατηρήστε τώρα ότι ρόλος του Tag είναι να ξεχωρίζει μεταξύ τους όλα αυτά τα blocks της κεντρικής μνήμης που απεικονίζονται στην ίδια θέση της κρυφής μνήμης, άρα που το πολύ ένα από αυτά ενδέχεται να βρίσκεται εκεί· βάσει αυτού, τώρα, πώς σχετίζεται το μέγεθος του Tag (bits per Tag) με το παραπάνω πλήθος blocks κεντρικής μνήμης ανά θέση κρυφής μνήμης;

11.4 Ο Προσομοιωτής YAC_SIM γιά Κρυφές Μνήμες

Στις επόμενες τρείς ασκήσεις, επιπλέον των ερωτημάτων που θα απαντήσετε "με το χέρι", θα χρησιμοποιήσετε και έναν απλό προσομοιωτή κρυφών μνημών, γραμμένον (2018) από τον τότε μεταπτυχιακό φοιτητή του Τμήματός μας *Ιωάννη Βάρδα* τον οποίο και ευχαριστούμε πολύ! Δίνοντας το μέγεθος και τις παραμέτρους μιάς τέτοιας κρυφής μνήμης, και μιά αλληλουχία προσπελαζομένων διευθύνσεων, ο προσομοιωτής τυπώνει το hit ratio για αυτή την αλληλουχία προσπελάσεων, και μπορεί να σας τυπώνει και τα περιεχόμενα της κρυφής μνήμης σε στιγμές που εσείς το ζητάτε. Ο προσομοιωτής αυτός λέγεται **YAC_SIM** (Yet Another Cache Simulator) και θα τον βρείτε στις μηχανές Linux του Τμήματος, στο directory:

~hy225/YAC_SIM –δηλαδή στο: /home/misc/courses/hy225/YAC_SIM

Σας προτείνουμε να αντιγράψετε ολόκληρο το παραπάνω directory στην περιοχή σας, να "μπείτε" στο σχετικό αντίγραφο, και να ξεκινήστε το εκτελέσιμο αρχείο "yac_sim": `cp -r ~hy225/YAC_SIM . ; cd YAC_SIM ; ./yac_sim config.txt;` Δεδομένου ότι ο YAC_SIM δεν χρησιμοποιεί γραφικά και ότι όλη η επικοινωνία του με τον χρήστη είναι terminal-oriented, σας προτείνουμε να τον τρέχετε πάνω στις μηχανές Linux του Τμήματος, remotely από οπουδήποτε βρίσκεστε.

Πριν ξεκινήσετε τον προσομοιωτή: Ο YAC_SIM δέχεται ένα αρχείο παραμετροποίησης –*configuration file*. Το εκτελέσιμο του YAC_SIM είναι το yac_sim, και δέχεται το όνομα του configuration file ως παράμετρο π.χ. τρέξτε: **`./yac_sim config.txt`** –το αρχείο αυτό θα πρέπει να περιέχει τιμές για τις εξής παραμέτρους και με την εξής μορφή:

memsize *αριθμός* [Μεγέθος μνήμης, σε Bytes]
wordsize *αριθμός* [Μεγέθος λέξης, σε Bytes]
cachesize *αριθμός* [Μεγέθος κρυφής μνήμης, σε Bytes]
cachelinesize *αριθμός* [Μέγεθος Γραμμής (Block/Line Size, §11.6) σε Bytes]
associativity *αριθμός* [Πλήθος Δρόμων (Ways, §11.7) Προσεταιριστικότητας]

Όλες οι παράμετροι που αφορούν μέγεθος πρέπει να είσαι σε Bytes. Επίσης όλες οι τιμές πρέπει να είναι δυνάμεις του 2 και έγκυρες, π.χ. cachelinesize < cachesize. Οι παραπάνω παράμετροι μπορεί είναι και με διαφορετική σειρά, και αν κάποια παράμετρος δεν βρεθεί θα σας ζητηθεί από το πρόγραμμα να την εισάγετε όταν θα το τρέξετε. Γιά το μήκος γραμμής, που θα το δούμε παρακάτω στην §11.6, βάλτε γιά τώρα (άσκηση 11.5) τον ίδιο αριθμό όσο το μέγεθος λέξης. Γιά την προσεταιριστικότητα, που θα την δούμε παρακάτω στην §11.7, αποδεκτές τιμές πλήθους δρόμων είναι: 1, 2, 4, 8 και 16. Για να προσομοιώθεί κρυφή μνήμη άμεσης (μονοσήμαντης) απεικόνισης ορίστε πλήθος δρόμων = 1. Σας δίδεται ένα έτοιμο configuration file στο παραπάνω directory του μαθήματος –μπορείτε να το χρησιμοποιήστε σαν οδηγό για να φτιάξετε το δικό σας.

Μόλις τρέξει ο προσομοιωτής: Αν δεν έχετε εισάγει κάποιες από τις παραμέτρους όπως παραπάνω, θα σας ζητηθεί να τις εισάγετε. Αφού εισαχθούν όλες οι παράμετροι, ξεκινάει η προσομοίωση, κατά την οποία έχετε τις εξής δυνατότητες: Δίνοντας έναν αριθμό (δεκαδικό), αυτός θεωρείται διεύθυνση, και ο προσομοιωτής κοιτάζει εάν υπάρχει ή όχι αυτή η λέξη στην κρυφή μνήμη, και σας αναφέρει αντίστοιχα "Ευστοχία" η "Αστοχία". Σε περίπτωση αστοχίας, ο προσομοιωτής εισάγει στην κρυφή μνήμη την γραμμή (§11.6) που περιέχει αυτή τη λέξη. Επειδή το να δίνετε τις προσπελάσεις μνήμης μία-μία με το χέρι είναι αργό και βαρετό, ο προσομοιωτής μπορεί να διαβάσει ένα ολόκληρο αρχείο με διευθύνσεις (μια ανα γραμμή) ("access *trace* file"), και να κάνει τις δουλειές που είπαμε γιά την καθεμιά τέτοια διεύθυνση· αυτό γίνεται με την εντολή *source*. Συνολικά οι διαθέσιμες εντολές είναι:

- *διεύθυνση* [Προσομοίωση προσπέλασης σε αυτή τη διεύθυνση]
- **source** *filename* [Διαβάζει διευθύνσεις απο αρχείο και τις προσομοιώνει]
- **display** [Τυπώνει τα περιεχόμενα της κρυφής μνήμης, το συνολικό ποσοστό ευστοχίας/αστοχίας, και τις προσπελάσεις που έγιναν]
- **flush** [Αδειάζει (invalidate) τα περιεχόμενα της κρυφής μνήμης]
- **help** [Τυπώνει τις διαθέσιμες εντολές (source, display, flush, help, exit)]
- **exit** [Τυπώνει το συνολικό ποσοστό ευστοχίας/αστοχίας και τις προσπελάσεις που έγιναν, και τερματίζει την προσομοίωση]

Άσκηση 11.5: Ποσοστό Αστοχίας σε Μνήμη Μονοσήμαντης Απεικόνισης

Θεωρήστε την κρυφή μνήμη άμεσης απεικόνισης του σχήματος της άσκησης 11.3, δηλαδή κρυφή μνήμη μεγέθους 32 Bytes = 8 lines (blocks), γιά λέξεις των 4 Bytes και γιά κεντρική μνήμη μεγέθους 128 Bytes, γιά την οποία το YAC_SIM configuration file θα το βρείτε στο αρχείο **ex11_5_config.txt** στο directory του YAC_SIM –μόνο προσέξτε ότι στο μεν σχήμα της §11.3 οι διευθύνσεις ήταν Block Addresses, ενώ ο YAC_SIM είναι Byte-Addressable (όλες οι διευθύνσεις αναφέρονται σε Bytes). Η κρυφή μνήμη είναι αρχικά κενή (όλα τα valid bits ψευδή –και σ' έναν πραγματικό υπολογιστή και στον YAC_SIM), και θεωρήστε ότι ο επεξεργαστής προσπελάζει τις εξής διευθύνσεις μνήμης, κατά χρονολογική σειρά:

```
100,  72,  56,  96,  76,  60,  52, 100,  80,  96,
 72,  52,  76, 104,  60, 100,  80,  52,  96,  84,
100,  80,  52, 108, 104,  60,  56, 108,  76,  52,
 96,  76,  56, 100,  60,  52, 104,  64,  60,  76
```

(α) Ποιές από αυτές τις προσπελάσεις είναι άστοχες και ποιές είναι εύστοχες; Γιά να το απαντήστε, γράψτε τις προσπελάσεις στη σειρά, και δίπλα σε καθεμιά σημειώστε σε παρένθεση τη θέση της κρυφής μνήμης όπου αυτή τοποθετείται, και "A" ή "E". Το "A" ή "E" προκύπτει κοιτώντας πίσω στο χρόνο και βλέποντας ποιά λέξη μνήμης είχε μπεί σε αυτή τη θέση τελευταία.

(β) Πόσες από τις προσπελάσεις είναι άστοχες; Ποιό είναι το ποσοστό αστοχίας;

(γ) Με τη βοήθεια του προσομοιωτή YAC_SIM, εκτελέστε τις προσπελάσεις αυτές και διασταυρώστε τα αποτελέσματά σας. Δείτε ποιές λέξεις (μπλοκ - cache lines) πηγαίνουν σε ποιες θέσεις της κρυφής μνήμης. Δείτε το πλήθος των misses και hits, καθώς και τα miss rate, και hit rate.

(δ) Εστω ότι ο χρόνος ευστοχίας (hit time) είναι 1 κύκλος ρολογιού, ενώ το επιπλέον κόστος αστοχίας (miss penalty) είναι 5 κύκλοι ρολογιού. Τότε, ποιός είναι ο μέσος χρόνος προσπέλασης σε αυτή την ιεραρχία μνήμης; (Βλ. άσκηση 11.1 γιά τους ορισμούς). (Το κόστος αστοχίας που δίδεται εδώ είναι υπερβολικά μικρό γιά σημερινά δεδομένα, όμως και το ποσοστό αστοχίας είναι υπερβολικά μεγάλο λόγω του τεχνητού παραδείγματος που έχουμε).

Άσκηση 11.6: Αύξηση του Μεγέθους Block (Line)

Δείτε τις **Διαφάνειες** 28 και 33-34 του (Αγγλικού) βιβλίου, όπως και τα σχήματα στο τέλος αυτής της παραγράφου.
Στις ασκήσεις 11.3 και 11.5, το **μέγεθος block** της κρυφής μνήμης ήταν μία (1) λέξη μόνο, δηλαδή σε κάθε αστοχία φέρναμε στην κρυφή μνήμη μόνο την μία συγκεκριμένη λέξη που θέλαμε και μας έλειπε. Μιά τέτοια οργάνωση, δεν εκμεταλλεύεται τη χωρική τοπικότητα, δηλαδή ότι τα προγράμματα, μόλις χρειαστούν μιά λέξη, έχουν μεγάλη πιθανότητα σε λίγο να χρειαστούν και τις διπλανές της. Γιά να εκμεταλλευτούμε αυτή την ιδιότητα (σε συνδυασμό με το γεγονός ότι προσπελάσεις σε συνεχόμενες διευθύνσεις κύριας μνήμης είναι πολύ φτηνότερες από προσπελάσεις σε τυχαίες διευθύνσεις) (και γιά να μειώσουμε και το πλήθος των address tags), αυξάνουμε το μέγεθος των blocks της κρυφής μνήμης.

Ας πούμε ότι στην κρυφή μνήμη της άσκησης 11.5 (δηλαδή του σχήματος της άσκησης 11.3) κάνουμε το μέγεθος block να είναι 2 λέξεις, δηλαδή 8 Bytes. Τότε, η κρυφή μνήμη, μεγέθους πάντα 32 Bytes, θα έχει 4 μόνο blocks, τα 0, 8, 16, και 24. Το block 0 περιέχει τις θέσεις 0 και 4, το block 8 περιέχει τις θέσεις 8 και 12, κ.ο.κ. Επίσης, οι λέξεις μνήμης θεωρούνται ζευγαρωμένες σε blocks ίσου μεγέθους, που είναι ευθυγραμμισμένα στα φυσικά τους όρια (όπως και οι γνωστοί μας περιορισμοί ευθυγράμμισης των προσπελάσεων του επεξεργαστή). Επομένως, τα ζευγάρια των λέξεων μνήμης είναι τα: 0 με 4, 8 με 12, 16 με 20, 24 με 28,... 112 με 116, και 120 με 124.

Σε κάθε αστοχία, φέρνουμε στην κρυφή μνήμη όχι μόνο τη λέξη που χρειαζόμαστε, αλλά και το "ταίρι" της (τα ταίρια της, γιά μεγαλύτερα blocks), δηλαδή **ολόκληρο το block** στο οποίο αυτή ανήκει. Αυτό το block λέξεων της κύριας μνήμης, το φέρνουμε στο block θέσεων της κρυφής μνήμης όπου αυτό απεικονίζεται, με την ίδια συνάρτηση απεικόνισης

όπως πριν.

(α) Πόσες ετικέτες διεύθυνσης (address tags) χρειάζεται τώρα η κρυφή μνήμη; Παρατηρήστε ότι, αφού πάντα φέρνουμε ολόκληρα blocks και ποτέ μεμονωμένες λέξεις, αν ξέρουμε ποιά λέξη βρίσκεται π.χ. στη θέση 24 της κρυφής μνήμης, τότε ξερουμε αυτόματα και ποιά λέξη βρίσκεται στη θέση 28, δηλαδή στην άλλη θέση του ίδιου block: το "ταίρι" της πρώτης.

(β) Θεωρήστε την ίδια σειρά προσπελάσεων όπως και στην άσκηση 11.5. Με τη νέα κρυφή μνήμη, ποιές από αυτές τις προσπελάσεις είναι άστοχες και ποιές είναι εύστοχες; Απαντήστε με τρόπο ανάλογο προς την ερώτηση 11.5(α), αλλά προσέξτε ότι τώρα η πρώτη αστοχία σε μιά λέξη ενός block φέρνει ολόκληρο το block (2 λέξεις), και επομένως η επόμενη προσπέλαση στην άλλη λέξη του ίδιου block θα ευστοχήσει.

(γ) Πόσες από τις προσπελάσεις είναι άστοχες; Ποιό είναι το ποσοστό αστοχίας;

(δ) Με τη βοήθεια του προσομοιωτή YAC_SIM, και με το νέο **ex11_6_config.txt** αυτή τη φορά, εκτελέστε ξανά τις προσπελάσεις αυτές και διασταυρώστε τα αποτελέσματά σας.

(ε) Με τη βοήθεια του YAC_SIM δοκιμάστε την παρακάτω, λίγο διαφοροποιημένη αλληλουχία προσπελάσεων, που θα την βρείτε και στο αρχείο **ex11_6e_trace.txt**:

```
100,  72,  56,  96,  76,  60,  52, 100,  80,  96,
 72,  48,  76, 104,  60, 100,  84,  52,  96,  80,
100,  84,  52, 108, 104,  60,  56,  44,  76,  52,
 40,  76,  56,  36,  60,  52,  40,  64,  60,  76
```

Τι ποσοστό ευστοχίας/αστοχίας βρίσκετε σε αυτήν την περίπτωση; Δοκιμάστε και με το προηγούμενο μέγεθος μπλοκ της άσκησης 11.5. Τώρα υπάρχει βελτίωση;

(στ) Έστω ότι ο χρόνος ευστοχίας είναι 1 κύκλος ρολογιού, ενώ το επιπλέον κόστος αστοχίας είναι 6 κύκλοι ρολογιού: 5 κύκλοι γιά την πρώτη λέξη, και 1 επιπλέον κύκλος γιά την μία επιπλέον λέξη. Η δεύτερη λέξη μας κοστίζει έναν μόνο κύκλο παραπάνω από την πρώτη επειδή οι δύο λέξεις που φέρνουμε ανήκουν στο ίδιο (ευθυγραμμισμένο) block της κύριας μνήμης, και άρα μπορούν να διαβαστούν "μαζί", πολύ γρηγορότερα από δύο τυχαίες, άσχετες αναγνώσεις που θα χρειάζονταν 5+5=10 κύκλους ρολογιού. Τώρα, ποιός είναι ο μέσος χρόνος προσπέλασης σε αυτή την ιεραρχία μνήμης;

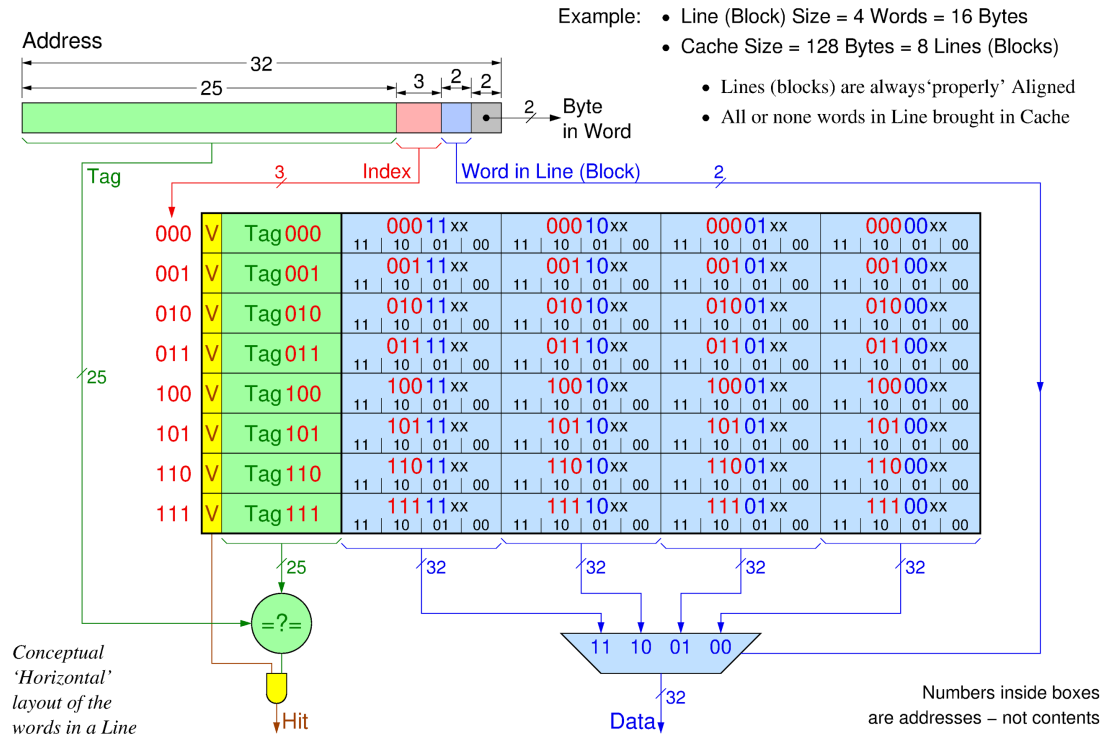
(ζ) Γράψτε ένα καινούργιο trace file με 20 προσπελάσεις που θα έχει μικρότερο ποσοστό ευστοχίας με την cache με block size = 8 Bytes από οτι με την cache με block size = 4 bytes. Τι ποσοστό επιτυχίας είχε στη μια και τι στην άλλη; Παραδώστε αυτό το trace file σας, με όνομα ex11_6z_trace.txt μαζί με την αναφορά σας.

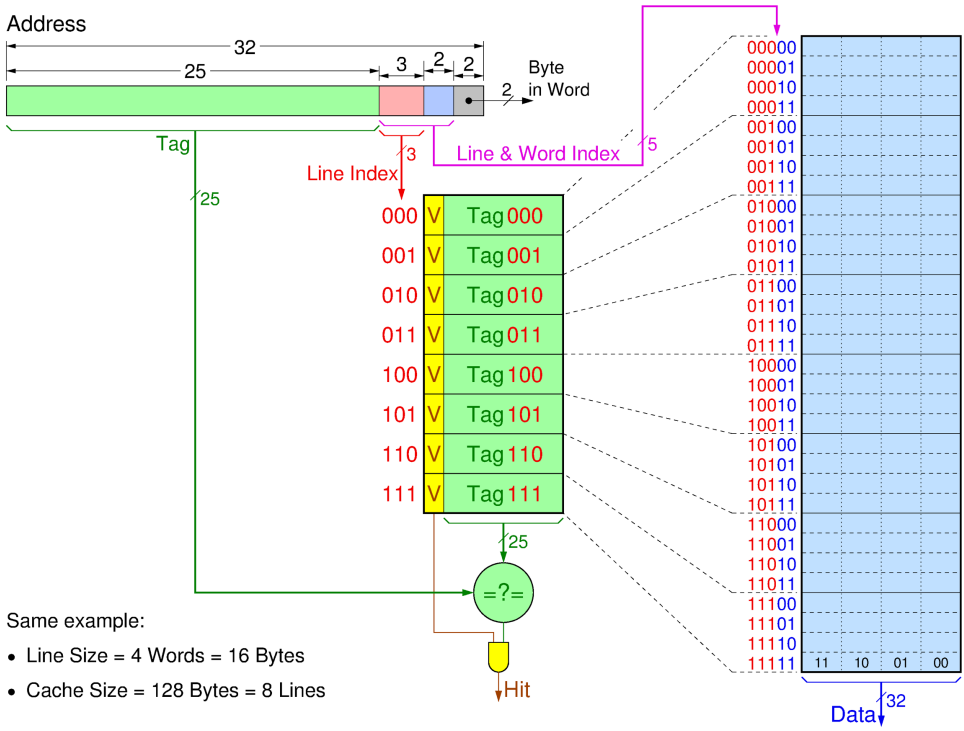
Θεωρία – πέραν της άσκησης, και επικουρικά προς τις διαφάνειες 27, 28, 33, 34 του βιβλίου: Τα δύο επόμενα σχήματα δείχνουν μιά κρυφή μνήμη μεγέθους 8 Lines (Blocks) των 4 λέξεων η κάθε γραμμή (line/block). Στο πρώτο σχήμα, το κομμάτι των data (γαλάζιο) είναι σχεδιασμένο γιά να βοηθήσει στην κατανόηση από τον αναγνώστη, ενώ στο δεύτερο σχήμα το κομμάτι αυτό είναι όπως θα κατασκευάζονταν στην πραγματικότητα. Στο πρώτο σχήμα λοιπόν, το κάθε "μπλόκ" (block), ή "γραμμή" (line), είναι σχεδιασμένο οριζόντια, σαν μία γραμμή (row) του πίνακα της κρυφής μνήμης: αποτελείται από 4 λέξεις, καθεμία των 4 Bytes, και συνοδεύεται από το Tag του και το Validity bit του.

Όταν μία γραμμή στην κρυφή μνήμη περιέχει έγκυρα δεδομένα, τότε αυτά είναι *πάντα* 4 γειτονικές, συνεχόμενες λέξεις από την κεντρική μνήμη, και πάντα *ευθυγραμμισμένες* σε φυσικά όρια γραμμών στη μνήμη, δηλαδή η διεύθυνση του πρώτου Byte της γραμμής είναι πάντα ακέραιο πολλαπλάσιο του 16 – όσο το μέγεθος της γραμμής. Αφού οι (ευθυγραμμισμένες) τετράδες γειτονικών λέξεων "ταξιδεύουν" πάντα μαζί –ή όλες θα είναι μέσα σε μία γραμμή της κρυφής μνήμης, ή καμία– προκύπτει ότι ένα μόνον Tag αρκεί γιά ολόκληρη τη γραμμή, αφού αυτό ταυτοποιεί ολόκληρο το μπλόκ/γραμμή των 4 γειτονικών (και ευθυγραμμισμένων) λέξεων.

Αφού οι γραμμές στην κεντρική μνήμη ξεκινάνε πάντα από διεύθυνση που είναι ακέραιο πολλαπλάσιο του μεγέθους γραμμής (των 16 Bytes, εδώ), προκύπτει ότι όλα τα Bytes και οι

λέξεις μέσα τους έχουν τα ίδια αριστερά (MS) bits διεύθυνσης, μέχρι και με εξαίρεση τα 4 δεξιά (LS) bits διεύθυνσης, εδώ· αυτά τα κοινά αριστερά bits διεύθυνσης αποτελούν την Block (Line) Address στη μνήμη. Αυτής της Line Address τα δεξιά bits αποτελούν τη συνάρτηση διασποράς, δηλαδή το Index που καθορίζει τη θέση μέσα στην κρυφή μνήμη όπου επιβάλεται να έλθει, όταν έλθει, η γραμμή αυτή της κεντρικής μνήμης· τα bits αυτά είναι χρωματισμένα κόκκινα στο σχήμα. Μέσα στην κρυφή μνήμη, το Index επιλέγει την "θέση", όπου οι θέσεις είναι μεγέθους μιάς γραμμής (line) καθεμία (16 Bytes εδώ), αφού η γραμμή είναι η μονάδα της εκάστοτε μεταφερόμενης ποσότητας πληροφορίας. Μέχρι στιγμής, έχουμε "ξοδέψει" 7 bits από το δεξιό μέρος της διεύθυνσης (όσα δηλαδή αντιστοιχούν στο μέγεθος της κρυφής μνήμης - 128 Bytes εδώ): 2 bits γιά Byte-in-Word, 2 bits γιά Word-in-Line, και 3 bits γιά Line-in-Cache Index, που αντιστοιχούν όντως σε: 4 Bytes/Word επί 4 Words/Line, επί 8 Lines/Cache = 128 Bytes/Cache. Τα υπόλοιπα αριστερά bits της διεύθυνσης, 25 bits εδώ, γιά 32-μπιτες διευθύνσεις, αποτελούν το Tag.





Με τον τρόπο που είναι σχεδιασμένη η κρυφή μνήμη στο πρώτο σχήμα, υπονοεί ότι σε κάθε προσπέλαση διαβάζουμε μιά ολόκληρη γραμμή, δηλαδή (μαζί με το Tag & Valid bit) και τις 4 λέξεις δεδομένων της, και στη συνέχεια επιλέγουμε μεταξύ των 4 αυτών λέξεων εκείνην που πραγματικά ζήτησε ο επεξεργαστής βάσει των 2 bits "Word in Line" της διεύθυνσης. Παρατηρήστε όμως ότι αυτά τα 2 bits επιλογής είναι γνωστά ήδη *πριν* αρχίσουμε την ανάγνωση της γραμμής των data, και είναι μάλλον σπατάλη να διαβάζουμε 4×32 = 128 bits από μιά μνήμη απλά και μόνο γιά να κρατήσουμε 32 από αυτά εκ των υστέρων, όταν ήταν ήδη γνωστό εξ αρχής ποιά 32 θέλαμε. Με άλλα λόγια, και ενθυμούμενοι ότι η ανάγνωση από μνήμη είναι κατ' ουσία ένας μεγάλος πολυπλέκτης στην κατακόρυφη διάσταση: μιά "φαρδιά" μνήμη απ' όπου διαβάζουμε πολλές λέξεις και στη συνέχεια επιλέγουμε "οριζοντίως" μία από αυτές είναι ισοδύναμη με μία μνήμη "στενότερη" και "ψηλότερη", όπως ακριβώς δείχνει το δεύτερο σχήμα.

Έτσι, το δεύτερο σχήμα δείχνει πώς πραγματικά θα κατασκεύαζε κανείς μιά κρυφή μνήμη με πολλαπλές λέξεις ανά γραμμή (block). Το κομμάτι των data έχει πλάτος μία λέξη –όσο δηλαδή είναι οι ποσσότητες που θέλουμε να διαβάζουμε– και ύψος όσες οι λέξεις εκεί, δηλαδή πλήθος γραμμών επί γραμμές ανά λέξη. Αντίθετα, το κομμάτι των Tags (με τα Validity bits) έχει *μικρότερο* ύψος: όσο το πλήθος των γραμμών μόνον, αφού *ολόκληρη* η κάθ γραμμή έχει ένα και μόνον, ενιαίο Tag (και Validity bit). Όταν ψάχνουμε μία λέξη, διαβάζουμε το Tag της υποψήφιας θέσης (γραμμής) στην οποία δείχνουν τα Index bits της διεύθυνσης που ζητάμε, ενώ ταυτόχρονα διαβάζουμε (speculatively – με την ελπίδα ευστοχίας) και τα Data από τη θέση που αυτά θα βρίσκονται, εάν έχουμε ευστοχία: τη λέξη από το Data array που αντιστοιχεί στην επιθυμητή γραμμή (Index bits) *και* επιθυμητή λέξη εντός γραμμής (word-in-line address bits).

Άσκηση 11.7:
Μερικώς Προσεταιριστικές Κρυφές Μνήμες Δύο ή περισσοτέρων Δρόμων

Δείτε τις **Διαφάνειες** 40 έως και 46 του (Αγγλικού) βιβλίου.
Στο μάθημα είδαμε ότι η κρυφή μνήμη με απ'ευθείας (μονοσήμαντη) απεικόνιση επιφέρει συγκρούσεις μεταξύ διευθύνσεων που τυχαίνει να απεικονίζονται στο ίδιο block της κρυφής μνήμης. Αυτές οι συγκρούσεις είναι ανεπιθύμητες, ειδικά εάν η κρυφή μνήμη έχει χώρο για εισερχόμενα δεδομένα (μία ή περισσότερες γραμμές είναι invalid) αλλά τα εισερχόμενα δεδομένα καταλήγουν να εκδιώξουν (αντικαταστήσουν) άλλα έγκυρα (valid) και πιθανόν χρήσιμα στο πρόγραμμα δεδομένα. Μία πλήρως προσεταιριστική (fully associative) κρυφή μνήμη επιλύει αυτό το πρόβλημα, το κόστος υλοποίησής της όμως είναι πολύ υψηλό για ρεαλιστικά μεγέθη κρυφής μνήμης (πχ. 64 KB).

Μια ενδιάμεση λύση με χαμηλότερο κόστος από την πλήρως προσεταιριστική κρυφή μνήμη είναι μία "μερικώς προσεταιριστική" κρυφή μνήμη (set associative cache). Σε αυτή την περίπτωση η κρυφή μνήμη οργανώνεται σε περισσότερες από μία στήλες (τυπικά από 2 έως 8 στήλες σε σύγχρονους επεξεργαστές). Οι στήλες αυτές ονομάζονται συχνά και "ways" ή δρόμοι. Η απεικόνιση διευθύνσεων σε γραμμές της κρυφής μνήμης είναι παρόμοια την μονοσήμαντη απεικόνιση. Ωστόσο, κάθε γραμμή της κρυφής μνήμης περιέχει τώρα ένα σύνολο (set) από blocks (τόσα όσες και οι στήλες) και ο επεξεργαστής μπορεί να "επιλέξει" το block (δηλάδη τη στήλη) στο οποίο θα απεικονίσει μία εισερχόμενη διεύθυνση. Για παράδειγμα, εάν το σύνολο έχει άκυρα blocks ένα από αυτά μπορεί να χρησιμοποιηθεί για την απεικόνιση της εισερχόμενης διεύθυνσης χωρίς να εκδιωχθούν άλλα έγκυρα (valid) blocks που υπάρχουν ήδη στο σύνολο. Μπορεί να πει κανείς ισοδύναμα, ότι το σύνολο blocks κάθε γραμμής της κρυφής μνήμης αντιμετωπίζεται σαν μία μικρή πλήρως προσεταιριστική κρυφή μνήμη για να αποφεύγονται ανεπιθύμητες συγκρούσεις. Φυσικά οι συγκρούσεις δεν μπορούν να αποφευχθούν πάντα, εφόσον σε κάθε σύνολο θα απεικονίζονται ούτως ή άλλως πολύ περισσότερες διευθύνσεις από το πλήθος των blocks του συνόλου.

Παράδειγμα: Ας υποθέσουμε ότι έχουμε μια κρυφή μνήμη μεγέθους 16 KB και το μέγεθος του block είναι 32 bytes, κατά συνέπεια τα 5 λιγότερο σημαντικά bits της διεύθυνσης δίνουν το block offset. Η μνήμη έχει 512 blocks. Αν η μνήμη χρησιμοποιούσε μονοσήμαντη απεικόνιση, τα επόμενα 9 bits (bit 6 έως bit 14) της διεύθυνσης θα χρησιμοποιούνταν σαν δείκτης του block. Ας υποθέσουμε ότι η κρυφή μνήμη χρησιμοποιεί μερικώς προσεταιριστική απεικόνιση με 2 σύνολα. Τα blocks χωρίζονται σε 2 στήλες των 256 blocks

και χρησιμοποιούνται 8 (αντί για 9 στη μονοσήμαντη απεικόνιση), δηλ. τα bits 6 έως 13, για να δεικτοδοτήσουν ένα σύνολο των 2 blocks. Η ταυτοποίηση της διεύθυνσης γίνεται με έλεγχο των tags και των 2 blocks του συνόλου. Εάν η ταυτοποίηση αποτύχει, ο επεξεργαστής επιλέγει ένα από τα 2 blocks του συνόλου για να απεικονίσει τη διεύθυνση, αντικαθιστώντας το block εάν αυτό είναι valid.

Η μερικώς προσεταιριστική κρυφή μνήμη τυπικά μειώνει σημαντικά το ποσοστό αστοχιών (miss rate) λόγω αποφυγής συγκρούσεων, χάρη στην περισσότερο ευέλικτη απεικόνιση διευθύνσεων στην κρυφή μνήμη. Πειραματικά, έχει δειχθεί ότι το ποσοστό αστοχίας μίας μερικώς προσεταιριστικής κρυφής μνήμης 2 δρόμων (2-way set-associative) είναι περίπου τόσο όσο το ποσοστό αστοχίας μίας κρυφής μνήμης μονοσήμαντης απεικόνισης διπλάσιου μεγέθους. Ωστόσο, η μείωση μόνο του ποσοστού αστοχίας μίας κρυφής μνήμης δεν σημαίνει απαραίτητα και βελτίωση της επίδοσης του επεξεργαστή! Όπως μάθαμε στο μάθημα, η επίδοση μιας ιεραρχίας με κρυφή μνήμη εξαρτάται από 3 παράγοντες: το ποσοστό αστοχίας, την καθυστέρηση της ευστοχίας, και την καθυστέρηση της αστοχίας. Στην περίπτωση των μερικώς προσεταιριστικών κρυφών μνημών, όταν υπάρχει ευστοχία (δηλ. ο επεξεργαστής απεικονίσει τη διεύθυνση στο σύνολο, ταιριάξει την ετικέτα tag της διεύθυνσης με μία από τις αποθηκευμένες ετικέττες του συνόλου, και ελέγξει ότι η αποθηκευμένη διεύθυνση είναι valid), ο επιλογέας της γραμμής επιλέγει ένα σύνολο blocks, κατά συνέπεια απαιτείται επιπλέον υλικό για να διαβαστεί το συγκεκριμένο block που αναζητεί ο επεξεργαστής. Αυτό μπορεί να γίνει με χρήση πολυπλέκτη N:1 όπου N το πλήθος των blocks κάθε συνόλου. Η χρήση του πολυπλέκτη επιφέρει επιπλέον καθυστέρηση στην ανάγνωση του σωστού block, η οποία δεν υπάρχει στην περίπτωση κρυφής μνήμης με μονοσήμαντη απεικόνιση. Μάλιστα, όσο περισσότερα blocks υπάρχουν σε ένα σύνολο, τόσο μεγαλύτερη είναι η καθυστέρηση. Κατά συνέπεια ο χρόνος ευστοχίας σε μία κρυφή μνήμη που είναι set associative είναι μεγαλύτερος από το χρόνο ευστοχίας σε μία κρυφή μνήμη που είναι direct mapped, και αυξάνεται (με ρυθμό περίπου 2% για κάθε στήλη της κρυφής μνήμης). Ο μόνος τρόπος να συμπεράνουμε εάν μία μερικώς προσεταιριστική κρυφή μνήμη επιτυγχάνει καλύτερη επίδοση από μία κρυφή μνήμη μονοσήμαντης απεικόνισης είναι να υπολογίσουμε το μέσο χρόνο πρόσβασης στη μνήμη (average memory access time = hit_time + miss_rate * miss_penalty).

Άσκηση:
Στον προσομοιωτή YAC_SIM, χρησιμοποιήστε πάλι την μνήμη της άσκησης 11.5, τρέχοντας `./yac_sim ex11_5_config.txt`, και μελετήστε την εξής αλληλουχία προσπελάσεων:
0, 32, 8, 40, 0, 32, 8, 40
Τι ποσοστό ευστοχίας έχει αυτή η ακολουθία προσπελάσεων στην κρυφή μνήμη μονοσήμαντης απεικόνισης της §11.5; Στη συνέχεια, αλλάξτε το configuration της cache ώστε να γίνει 2-way set associative: φτιάξτε ένα νεο configuration file και στην μεταβλητή associativity βάλτε τιμή 2. Τι παρατηρείτε στα bits του index σε σχέση με πρίν; Τι ποσοστό ευστοχίας έχει η ίδια ακολουθία προσπελάσεων με τη νέα προσεταιριστικότητα των 2 δρόμων; Γιατί;

Άσκηση 11.8: Πολιτικές Αντικατάστασης

Δείτε τη **Διαφάνεια** 47 του (Αγγλικού) βιβλίου.
Σε κάθε σύστημα μνήμης (κρυφή μνήμη, εικονική μνήμη, ...) στο οποίο υπάρχει επιλογή ως προς την απεικόνιση μιας διεύθυνσης (δηλαδή η απεικόνιση δεν είναι μονοσήμαντη), και όταν όλες οι δυνατές επιλογές για την απεικόνιση απεικονίζουν ήδη νόμιμες (valid) διευθύνσεις, προκύπτει το πρόβλημα της αντικατάστασης μίας νόμιμης διεύθυνσης που είναι ήδη παρούσα στη μνήμη. Στην περίπτωση των πλήρως προσεταιριστικών κρυφών μνημών για παράδειγμα, εάν μια διεύθυνση του επεξεργαστή δεν απεικονίζεται σε κανένα block και όλα τα blocks είναι νόμιμα (κατάσταση valid) ο επεξεργαστής είναι υποχρεωμένος να αντικαταστήσει (replace) ένα από τα νόμιμα blocks. Εάν ξέραμε το μέλλον, θα προτιμούσαμε να διώξουμε (evict) το block εκείνο το οποίο θα το ξαναπροσπελάσουμε πιό μακριά στο μέλλον (στο απώτατο μέλλον). Εν τη αδυναμία μας να γνωρίζουμε το μέλλον, η συνήθως καλύτερη πρόβλεψη γιά το εγγύς μέλλον είναι το εγγύς παρελθόν: τα blocks που προσπελάσαμε πρόσφατα στο παρελθόν είναι υποψήφια να τα ξαναπροσπελάσουμε και σύντομα στο μέλλον, και αντίστροφα, τα blocks που έχουμε πολύ καιρό να τα προσπελάσουμε στο παρελθόν, μάλλον θα περάσει και πολύ χρόνος μέχρι να τα ξαναπροσπελάσουμε στο μέλλον.

Έτσι, η συνήθως προτιμόμενη πολιτική αντικατάστασης (replacement policy) είναι η *Least Recently Used (LRU)*: αντικατάστησε το block εκείνο το οποίο προσπελάστηκε στο απώτατο παρελθόν (λιγότερο πρόσφατα - least recently) από τον επεξεργαστή, και γι' αυτό περιμένουμε να μην προσπελαστεί ξανά σύντομα (ή και καθόλου) στο μέλλον. Η υλοποίηση της LRU σε περίπτωση επιλογής μεταξύ 2 blocks είναι σχετικά απλή στο υλικό: μπορεί να γίνει με χρήση ενός bit το οποίο θα δείχνει εάν το πρώτο block (0) ή το δεύτερο block (1) είναι το λιγότερο πρόσφατα χρησιμοποιημένο από τον επεξεργαστή. Δυστυχώς, η υλοποίηση του αλγόριθμου LRU για σύνολα 4, 8, ... blocks δεν είναι οικονομική σε υλικό, ενώ ακόμα και σε λογισμικό η υλοποίηση για μεγάλες μνήμες μπορεί να έχει απαγορευτικό κόστος. Πρακτικά, στις κρυφές μνήμες κρατάμε συνήθως πληροφορία LRU μεταξύ ολόκληρων υποσυνόλων των blocks ενός set, αντί γιά το κάθε ένα block χωριστά, ή και πληροφορία LRU μεταξύ των blocks ενός υποσυνόλου του set, αντί του set ολόκληρου· εναλλακτικά, και η (ψευδο)τυχαία επιλογή θύματος αντικατάστασης δεν είναι πολύ κακή, ενίοτε. Στην εικονική μνήμη, όπου οι υποψήφιες προς αντικατάσταση σελίδες είναι πάρα πολλές, χρησιμοποιούμε "προσεγγίσεις" της LRU, όπου ένα ή περισσότερα bits αναφοράς (reference bits) μετρούν προσεγγιστικά το αν ένα block προσπελάστηκε στο πρόσφατο παρελθόν και κατά συνέπεια δεν είναι καλός υποψήφιος για αντικατάσταση. Το σύστημα περιοδικά μηδενίζει τα bits αναφοράς (δηλαδή την παλαιότερη ιστορία των προσπελάσεων σε διευθύνσεις μνήμης), ώστε η πληροφορία που καταγράφεται σε αυτά να είναι όσο πιο επίκαιρη γίνεται.

Άσκηση:
Ο προσομοιωτής YAC_SIM χρησιμοποιεί πολιτική αντικατάστασης LRU. Εάν οι πραγματικές προσπελάσεις τύχει να μην ακολουθούν "μοτίβο LRU", τότε οι προσεταιριστικές μνήμες μπορεί να αποβούν ακόμα και χειρότερες από τη μνήμη μονοσήμαντης απεικόνισης. Γιά την κρυφή μνήμη της τελευταίας άσκησης, §11.7, φτιάξτε μια δική σας αλληλουχία προσπελάσεων (το πολύ 20 προσπελάσεις), με όνομα **ex11_8_trace.txt**, όπου η κρυφή μνήμη 2 δρόμων να έχει *χειρότερο* ποσοστό ευστοχίας από της direct mapped επειδή θα τυχαίνει ο αλγόριθμος αντικατάστασης LRU να μην είναι εδώ ο σωστός. Παραδώστε το αρχείο αυτό με την άσκηση.

11.9 Πολιτικές Εγγραφών

Δείτε τις **Διαφάνειες** 29 έως και 32 του (Αγγλικού) βιβλίου.
Η κρυφή μνήμη περιέχει αντίγραφα δεδομένων που βρίσκονται στην κύρια μνήμη. Εάν ο επεξεργαστής μεταβάλει τα περιεχόμενα μίας διεύθυνσης δεδομένων στην κρυφή μνήμη με μια εντολή store, προκύπτει ασυνέπεια μεταξύ των περιεχομένων του αντιγράφου της διεύθυνσης στην κρυφή μνήμη και των περιεχομένων του αντιγράφου της διεύθυνσης στην κύρια μνήμη. Η ασυνέπεια αυτή επιλύεται με την αντιγραφή στην κύρια μνήμη των περιεχομένων διευθύνσεων που βρίσκονται στην κρυφή μνήμη και τα περιεχόμενά τους έχουν αλλάξει από τη στιγμή που οι διευθύνσεις αυτές απεικονίστηκαν στην κρυφή μνήμη. Οι σύγχρονοι επεξεργαστές χρησιμοποιούν δύο πολιτικές τήρησης της συνέπειας των περιεχομένων των αντιγράφων:

Ταυτόχρονη Εγγραφή (Write Through): Στην πολιτική αυτή, όταν ο επεξεργαστής εκτελεί εντολή store και μεταβάλλει τα περιεχόμενα μίας διεύθυνσης στην κρυφή μνήμη, ενημερώνει την ίδια διεύθυνση στην κύρια μνήμη με τα νέα περιεχόμενα. Η πολιτική αυτή προφανώς τηρεί τη συνέπεια σε κάθε εντολή store, αλλά καθυστερεί την εκτέλεση της εντολής store μέχρι να ενημερωθεί η κύρια μνήμη, λειτουργία που μπορεί να απαιτήσει δεκάδες ή εκατοντάδες κύκλους σε σύγχρονους επεξεργαστές. Με άλλα λόγια, η πολιτική write-through μπορεί να αυξήσει το χρόνο ευστοχίας (hit time) στην κρυφή μνήμη όταν εκτελούνται εντολές stores. Μία λύση σε αυτό το πρόβλημα είναι η ένθεση ενός ενταμιευτή εγγραφών (write buffer), ο οποίος αποθηκεύει τα νέα περιεχόμενα των διευθύνσεων μνήμης στις οποίες γίνονται stores, και η ασύγχρονη μεταφορά των δεδομένων του write buffer στην κύρια μνήμη, ενώ ο επεξεργαστής συνεχίζει να εκτελεί εντολές χωρίς να περιμένει να ολοκληρωθούν οι μεταφορές που εκκρεμούν από το write buffer. Θέματα σχεδίασης των write buffers συζητώνται στο μάθημα **HY425 - Αρχιτεκτονική Υπολογιστών**.

Ετερόχρονη Εγγραφή (Write Back): Στην πολιτική αυτή, όταν ο επεξεργαστής εκτελεί εντολή store μεταβάλλει τα περιεχόμενα της διεύθυνσης μόνο στην κρυφή μνήμη. Τα νέα περιεχόμενα της διεύθυνσης προωθούνται στην κύρια μνήμη μόνο όταν η διεύθυνση αυτή

αντικατασταθεί από την κρυφή μνήμη, λόγω σύγκρουσης με άλλη διεύθυνσης ή/και της πολιτικής αντικατάστασης που χρησιμοποιεί η κρυφή μνήμη. Σε αυτή την περίπτωση, τα δεδομένα πρέπει να προωθηθούν στην κύρια μνήμη μόνο εάν το αντίγραφο στην κρυφή μνήμη έχει αλλαγμένα περιεχόμενα λόγω ενός ή περισσοτέρων stores. Για να γνωρίζει ο επεξεργαστής ποια αντίγραφα έχουν περιεχόμενα που έχουν μεταβληθεί από τότε που τα αντίγραφα αυτά απεικονίστηκαν στην κρυφή μνήμη, χρησιμοποιεί ένα bit που ονομάζεται dirty bit. Όταν ο επεξεργαστής εκτελεί store και το store ευστοχήσει στην κρυφή μνήμη, θέτει το dirty bit του block στο οποίο έγινε το store. Αν το block αυτό αργότερα αντικατασταθεί, τα περιεχόμενά του γράφονται στην κύρια μνήμη. Αλλιώς, η αντικατάσταση του block γίνεται απλώς γράφοντας πάνω από τα παλαιά του περιεχόμενα στην κρυφή μνήμη. Η πολιτική write back δεν επιφέρει την αργοπορία εγγραφής στην κύρια μνήμη σε κάθε store, και μία διεύθυνση μπορεί να διαβαστεί ή γραφτεί πολλές φορές στην κρυφή μνήμη, πριν τα περιεχόμενά της προωθηθούν στην κύρια μνήμη. Ωστόσο, η πολιτική write back μπορεί να μεγαλώσει το χρόνο αστοχίας στην κρυφή μνήμη ως εξής: Όταν ο επεξεργαστής αστοχεί και το block της μνήμης το οποίο αντικαθίσταται έχει αλλαγμένα περιεχόμενα (δηλ. έχει το dirty bit ίσο με 1), τότε ο επεξεργαστής καθυστερεί όσο περιμένει να γραφούν τα περιεχόμενα του block που αντικαθίσταται στην κύρια μνήμη.

Write allocate/no-allocate: Όταν ο επεξεργαστής εκτελεί μια εντολή store και αστοχήσει στην κρυφή μνήμη έχει δύο επιλογές: Ο επεξεργαστής μπορεί να γράψει τα νέα περιεχόμενα της διεύθυνσης στην κύρια μνήμη χωρίς να απεικονίσει τη διεύθυνση αυτή στην κρυφή μνήμη. Η πολιτική αυτή ονομάζεται write no-allocate, δηλαδή ο επεξεργαστής δεν δεσμεύει χώρο στην κρυφή μνήμη για διευθύνσεις στις οποίες γράφει και οι οποίες δεν βρίσκονται ήδη στην κρυφή μνήμη. Η εναλλακτική επιλογή είναι ο επεξεργαστής να δημιουργήσει αντίγραφο της διεύθυνσης στην κρυφή μνήμη, και ονομάζεται πολιτική write allocate. Η write allocate αποδίδει καλύτερα εάν ο επεξεργαστής πρόκειται να διαβάσει σύντομα τα δεδομένα τα οποία έγραψε, οπότε και θα ευστοχήσει διαβάζοντας τα δεδομένα από την κρυφή μνήμη. Η write no allocate αποδίδει καλύτερα εάν ο επεξεργαστής δεν πρόκειται να χρησιμοποιήσει τα δεδομένα τα οποία έγραψε (π.χ. σε μια εφαρμογή κωδικοποίησης δεδομένων, ο επεξεργαστής παράγει σαν έξοδο μια ακολουθία χαρακτήρων κάθε ένας από τους οποίους γράφεται μία φορά). Συνήθως η πολιτική write no allocate συνδυάζεται με την πολιτική write through, ενώ η πολιτική write allocate με την πολιτική write back.

11.10 Πολυ-Επίπεδες Κρυφές Μνήμες, κ.α.

Δείτε τις **Διαφάνειες** του (Αγγλικού) βιβλίου: 48-51 γιά τις πολυ-επίπεδες κρυφές μνήμες, 52 γιά τη σχέση κρυφών μνημών με την ομοχειρία εκτός σειράς (out-of-order pipelines – §14.4), και 53 γιά την επίδραση των αλγορίθμων στην τοπικότητα των προσπελάσεων μνήμης.

Τρόπος Παράδοσης

Παραδώστε μαζί την προηγούμενη άσκηση 10 και αυτήν εδώ τη σειρά 11, on-line, σε μορφή **PDF** (μόνον) (μπορεί να είναι κείμενο μηχανογραφημένο ή/και "σκαναρισμένο" χειρόγραφο, αλλά *μόνον* σε μορφή PDF). Παραδώστε μέσω **`turnin ex1011@hy225 [directoryName]`** ένα αρχείο ονόματι **`ex10.pdf`** γιά την 10, ένα αρχείο ονόματι **`ex11.pdf`** γιά την 11, και τα αρχεία trace files: ex11_6z_trace.txt και ex11_8_trace.txt
Θα εξεταστείτε **και προφορικά** για τις ασκήσεις 10 και 11 (μαζί), από βοηθό του μαθήματος, με διαδικασία γιά την οποία θα ενημερωθείτε μέσω ηλτά (email) στη λίστα του μαθήματος.
