HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2023

Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Σειρά Ασκήσεων 13: DRAM, Διαφύλλωση, Μονάδες Εισόδου/Εξόδου (I/O), DMA

κάντε τις έως Πέμπτη 18 Μαΐου 2023 (βδ. 13.2) (από βδ. 12.2)

Βιβλίο (Ελληνικό, έκδοση 4) - διαβάστε τα εξής, και με τους εξής τρόπους:

- DRAM, Διαφύλλωση (Interleaving) Μνήμης - σελ. 546-551 (από §5.2) (και τις ενότητες 13.1 - 13.3 εδώ παρακάτω): διαβάστε τις κανονικά, αποτελούν μέρος της εξεταστέας ύλης.
- Αρτηρίες (Δίαυλοι - Buses), και Σύνδεσμοι (point-to-point high-speed-serial links) - σελίδες 678-681 (πρώτο μέρος της § 6.5): διαβάστε τις κανονικά, αποτελούν μέρος της εξεταστέας ύλης.
- Επικοινωνία Επεξεργαστή, Συσκευών I/O, και Μνήμης (DMA) - σελίδες 684-696 (§ 6.6), καθώς και οι σημειώσεις και ασκήσεις εδώ παρακάτω: διαβάστε τις κανονικά, αποτελούν μέρος της εξεταστέας ύλης.

- Οργάνωση I/O επεξεργαστών x86 - σελίδες 682-683 (από § 6.5): διαβάστε τις "στα γρήγορα" - εγκυκλοπαιδικά.
- Μαγνητικοί (σκληροί) δίσκοι και μνήμες flash - σελίδες 670-678 (§ 6.3 και 6.4): διαβάστε τις "στα γρήγορα" - εγκυκλοπαιδικά.
- Συστήματα Αποθήκευσης (Storage), RAID, φερεγγυότητα, αξιοπιστία, διαθεσιμότητα - σελίδες 668-670 και 700-708 (§ 6.2 και 6.9): σημαντικό θέμα γιά περαιτέρω προσωπική σας μελέτη, αλλά εκτός ύλης αυτού του μαθήματος.
- Παράδειγμα server γιά data centers, και άλλα εγκυκλοπαιδικά θέματα - σελίδες 708-723 (§ 6.10 και 6.12): χρήσιμα γιά εγκυκλοπαιδική ενημέρωση, αλλά εκτός ύλης αυτού του μαθήματος.

13.1 SRAM, DRAM, Προσπελάσεις Συνεχόμενων Λέξεων

SRAM-DRAM: Οπως είπαμε στο μάθημα, τα chips μνήμης είναι οργανωμένα εσωτερικά σε κάμποσα blocks από στοιχεία μνήμης. Στις "στατικές" μνήμες (SRAM - Static Random Access Memory), τα στοιχεία μνήμης είναι flip-flops (με 6 transistors καθένα), και η αποθηκευμένη πληροφορία διατηρείται όσο είναι αναμένη η τάση τροφοδοσίας. Στις "δυναμικές" μνήμες (DRAM - Dynamic Random Access Memory), τα στοιχεία μνήμης είναι πυκνωτές (capacitors --ένας πυκνωτής και ένα transistor ανά bit), όπου αποθηκεύεται δυναμικά η πληροφορία. Λόγω του ρεύματος διαρροής, η πληροφορία αυτή (φορτίο στον πυκνωτή), χάνεται μέσα σε λίγα χιλιοστά του δευτερολέπτου (ms). Γιά να διατηρηθούν τα περιεχόμενα της DRAM πρέπει να τα αναζωογονούμε (refresh), δηλαδή να τα διαβάζουμε και να τα ξαναγράφουμε, κάθε περίπου 8 με 16 ms.

Μέγεθος (Χωρητικότητα - Gbits): Παρά το μειονέκτημά τους αυτό, και παρά την μεγαλύτερη καθυστέρηση προσπέλασης που έχουν, οι DRAM διαθέτουν ένα σημαντικό πλεονέκτημα: προσφέρουν περίπου μία τάξη μεγέθους μεγαλύτερη χωρητικότητα (capacity, Mbits --όχι "capacitance") ανά chip σε σχέση με τις SRAM. Ετσι, οι DRAM χρησιμοποιούνται σχεδόν πάντα γιά την κατασκευή της κύριας μνήμης (main memory) των υπολογιστών, ενώ οι SRAM χρησιμοποιούνται σχεδόν πάντα γιά τις κρυφές μνήμες (cache memories), λόγω της χαμηλότερης καθυστέρησής τους. Με την πρόοδο της τεχνολογίας κατασκευής ολοκληρωμένων κυκλωμάτων (chips), η χωρητικότητα των chips μνήμης συνεχώς αυξάνονταν. Τις προηγούμενες δεκαετίες, ο ρυθμός αυτής της αύξησης ήταν: **τετραπλασιασμός (x4)** χωρητικότητας κάθε **τρία (3) χρόνια**, όμως αυτή τη δεκαετία αυτός έχει πέσει. Το 2013, η τεχνολογία των DRAM βρίσκονταν γύρω στα 2 έως 16 Gbits ανά chip, αλλά και το 2017 μιά ματιά π.χ. στο www.micron.com δείχνει παρόμοιες χωρητικότητες. Εμπορικά, την μνήμη των υπολογιστών τη βρίσκει κανείς σε μικρές πλακέτες (modules - DIMM), που η καθεμιά έχει πάνω της συνήθως 8 (ή 9) ή 16 (ή 18) chips. Έτσι, π.χ., ένα module με 8 chips των 256M x 8 bits = 2 Gbits καθένα είχε συνολική χωρητικότητα 16 Gbits = 2 GBytes. Όταν τα chips είναι 9 αντί 8, ή 18 αντί 16, τα επιπλέον chips χρησιμοποιούνται γιά αποθήκευση κωδίκων ανίχνευσης και διόρθωσης σφαλμάτων (ECC - error correction codes).

Γραμμές και Στήλες: Μέσα στο chip της DRAM, το κάθε block είναι ένας περίπου τετράγωνος πίνακας από στοιχεία μνήμης, με γύρω στις 1024 ή παραπάνω γραμμές επί 1024 ή παραπάνω στήλες. Γιά να διαβάσουμε ένα στοιχείο μνήμης επιλέγουμε πρώτα τη γραμμή στην οποία ανήκει αυτό, δίνοντας τη διεύθυνση γραμμής (row address) στον αποκωδικοποιητή γραμμής, ο οποίος ανάβει ένα σύρμα (word line) που διατρέχει και ενεργοποιεί την επιθυμητή γραμμή. Οταν ανάψει το σύρμα αυτό, όλα τα στοιχεία μνήμης (bits) πάνω στη γραμμή αυτή διαβάζονται, δηλαδή τοποθετούν το καθένα την τιμή του (περιεχόμενό του) στο αντίστοιχο σύρμα στήλης (bit line) που διατρέχει τη στήλη του. Ετσι, στο κάτω μέρος του block της μνήμης, στις απολήξεις των συρμάτων στήλης, εμφανίζεται το περιεχόμενο όλων των bits που είναι αποθηκευμένα στην επιλεγείσα γραμμή. Ενας μεγάλος πολυπλέκτης επιλέγει τότε το bit που εμείς θέλαμε, βάσει της διεύθυνσης στήλης (column address), και το δίνει προς τα έξω. Η όλη αυτή διαδικασία, από την είσοδο της διεύθυνσης γραμμής μέχρι να βγεί το τελικό bit στην έξοδο, διαρκεί αρκετό χρόνο (γύρω στα 30 ns γιά τις σημερινές DRAM, από τα pins διεύθυνσης μέχρι τα pins δεδομένων του chip της DRAM, ενώ αυτή αυξάνει σημαντικά όταν προστεθεί και η καθυστέρηση από το chip του επεξεργαστή μέχρι το chip της DRAM και πίσω).

Γειτονικές Προσπελάσεις (sequential Accesses): Εάν μετά την παραπάνω διαδικασία, όμως, θέλουμε να διαβάσουμε και μερικά από τα "διπλανά" bits αυτού που μόλις διαβάσαμε, τότε αυτό μπορεί να γίνει πολύ γρηγορότερα: τα bits αυτά είναι "έτοιμα", στις απολήξεις των συρμάτων στήλης, και το chip της μνήμης μπορεί να τα αποστείλει στον αιτούντα την ανάγνωση (π.χ. τον επεξεργαστή) πολύ γρήγορα το ένα μετά το άλλο (περίπου 1 bit κάθε μισό με 1 ns σε καθένα από τα σύρματα δεδομένων (data) γιά τις σημερινές DRAM). Εκμεταλλευόμενοι τη δυνατότητα αυτή, πετυχαίνουμε να προσπελαύνουμε μεγάλες ομάδες γειτονικών λέξεων (π.χ. cache lines (blocks)) με πολύ μικρή επιπλέον επιβάρυνση σε σχέση με την αρχική καθυστέρηση προσπέλασης της πρώτης λέξης της ομάδας.

13.2 Κόστος Εκκίνησης και Παροχή (Startup Cost versus Throughput)

Η παραπάνω περίπτωση προσπελάσεων σε γειτονικά bits στην ίδια γραμμή μιάς DRAM είναι παράδειγμα μιάς γενικότερης περίπτωσης: πολλές φορές, υπάρχει ένα σημαντικό **Κόστος Εκκίνησης** (start-up cost / overhead) γιά μιάν εργασία (π.χ. μεταφορά δεδομένων) που είναι (σχεδόν) ανεξάρτητο του μεγέθους της εργασίας (π.χ. του όγκου των δεδομένων), και στη συνέχεια υπάρχει ένα μικρότερο (ή πολύ μικρότερο) **επιπλέον (Διαφορικό) Κόστος** (incremental cost) γιά την κάθε επιπλέον μονάδα εργασίας (π.χ. γιά κάθε επιπλέον Byte ή λέξη δεδομένων που μεταφέρονται). Το επιπλέον, διαφορικό αυτό κόστος ανά μονάδα έργου πολλές φορές το μετράμε μέσω του αντιστρόφου του: ο **Ρυθμός Παραγωγής** του έργου, ή η **Παροχή** (throughput) (π.χ. των μεταφερόμενων δεδομένων) είναι η ποσότητα του έργου που παράγεται ανά μονάδα χρόνου, ή ο όγκος των δεδομένων που μεταφέρονται ανά μονάδα χρόνου.

Γιά παράδειγμα, θεωρήστε ένα DRAM chip με οργάνωση x8 (π.χ. 256M x 8), δηλαδή ένα chip με 8 data pins, που λειτουργεί σε συγχρονισμό (SDRAM - Synchronous DRAM) με ένα ρολόϊ 500 MHz (άρα με περίοδο ρολογιού 2 ns) (το παράδειγμα είναι κάπως παλαιότερο - το 2017, τα ρολόγια των DRAM's είναι συχνά πάνω από 1000 MHz). Ο χρονισμός των data pins είναι DDR (Double Data Rate), δηλαδή το κάθε data pin μπορεί να μεταφέρει ένα data bit σε κάθε θετική ακμή του ρολογιού και ένα επιπλέον data bit σε κάθε αρνητική ακμή ρολογιού, άρα στο παράδειγμά μας ένα data bit ανά 1 ns (μισή περίοδο ρολογιού), δηλαδή με ένα ρυθμό (παροχή) μεταφοράς 1 bit / 1 ns = **1 Gbit/s ανά pin**. Έστω ότι από τη στιγμή που δίνουμε στο chip αυτό μιά διεύθυνση ανάγνωσης μέχρι να μας επιστρέψει τα 8 bits (1 Byte) (μέσω των 8 data pins) από τη διεύθυνση που του δώσαμε περνάει καθυστέρηση **40 ns**. Μετά από αυτά τα πρώτα 8 bits όμως, έστω ότι εμείς θέλουμε και τα διπλανά τους 56 bits, δηλαδή συνολικά 64 bits (8 Bytes) (π.χ. διότι έχουμε 8 τέτοια chips εν παραλλήλω, και θέλουμε να διαβάσουμε μία Cache Line των 64 Bytes από αυτά (8 Bytes/chip x 8 chips = 64 Bytes συνολικά)) (καμιά φορά, αυτά όλα τα bits ή Bytes που τα διαβάζουμε όλα μαζί τα λέμε "burst" – ομοβροντία). Αφού το κάθε data pin του chip μπορεί να μας δίνει 1 data bit ανά 1 ns, τότε τα 8 pins του chip μπορούν να μας δίνουν 8 bits ανά 1 ns, δηλαδή 1 Byte / 1 ns = 1 GByte/s.

Σε αυτό λοιπόν το παράδειγμα, το *κόστος (καθυστέρηση) εκκίνησης* της ανάγνωσης ενός τέτοιου "burst" είναι 40 ns, και αυτή η καθυστέρηση εκκίνησης αφορά τόσο το κάθε ένα chip χωριστά όσο και τα 8 εν παραλλήλω chips όλα μαζί. Από την άλλη, το *επιπλέον (διαφορικό) κόστος (καθυστέρηση)* γιά τα επιπλέον (γειτονικά) bits και Bytes είναι 1 ns γιά κάθε ένα επιπλέον bit ανά pin, ή 1 ns γιά κάθε 8 επιπλέον bit (1 Byte) ανά chip, ή 1 ns γιά κάθε 8 επιπλέον Bytes γιά όλα και τα 8 chips. Το αντίστροφο αυτού του διαφορικού κόστους (το αντίστροφο της καθυστέρησης ανά μονάδα επιπλέον όγκου δεδομένων) είναι ο *ρυθμός μεταφοράς* των επιπλέον δεδομένων, ή η *παροχή* δεδομένων, δηλαδή 1 bit/ns = 1 Gbit/s γιά κάθε ένα pin, ή 1 Byte/ns = 1 GByte/s γιά κάθε ένα chip, ή 8 Bytes/ns = 8 GBytes/s γιά όλα και τα 8 chips. Με αυτή την καθυστέρηση εκκίνησης (40 ns) και το ρυθμό μεταφοράς (8 GBytes/s = 8 Bytes/ns), η συνολική καθυστέρηση γιά να διαβάσουμε ολόκληρη την Cache Line των 64 Bytes από τα 8 εν παραλλήλω chips θα είναι: 40 ns γιά τα πρώτα 8 Bytes, συν 1 επιπλέον ns γιά κάθε μιά επιπλέον ομάδα από 8 Bytes, δηλαδή συνολικά 7 επιπλέον ns γιά τις 7 επιπλέον οκτάδες από

Bytes (64 Bytes = 8 πρώτα Bytes + (7x8=56) επιπλέον Bytes), άρα συνολικά 40 + 7 = 47 ns. Παρόμοια συμπεριφορά με την παραπάνω συναντάμε και σε πλήθος άλλων περιπτώσεων, στα υπολογιστικά συστήματα και τα δίκτυα:

- Σαν πρώτο παράδειγμα, η ομοχειρία (pipelining) έχει αυτή ακριβώς τη συμπεριφορά, έστω και αν συχνά αδιαφορούμε γιά το κόστος εκκίνησης όταν μιά εργασία απαιτεί την εκτέλεση εκατομμυρίων εντολών. Στην δική μας απλή pipeline της άσκησης 9, το κόστος εκκίνησης είναι 5 κύκλοι ρολογιού (δηλ. όση ώρα χρειάζεται γιά να τελειώσει η *πρώτη* εντολή), και στη συνέχεια ο ρυθμός ολοκλήρωσης των επομένων, επιπλέον εντολών είναι 1 εντολή ανά κύκλο ρολογιού όσο δεν υπάρχουν αλληλεξαρτήσεις εντολών ή αστοχίες κρυφών μνημών.

- Σαν δεύτερο παράδειγμα, οι σκληροί δίσκοι έχουν κι αυτοί παρόμοια συμπεριφορά: μέχρι να μπορέσουμε να αρχίσουμε να διαβάζουμε το block δεδομένων που μας ενδιαφέρει, πρέπει να περιμένουμε να κινηθεί η κεφαλή ανάγνωσης και να γυρίσει ο δίσκος ούτως ώστε να έλθει το πρώτο bit του block κάτω από την κεφαλή, που συνήθως σημαίνει από λίγα έως 10 με 20 ms (milli-second) –αυτό είναι το κόστος (καθυστέρηση) εκκίνησης της ανάγνωσης. Μετά από αυτό όμως, τα bits είναι αποθηκευμένα τόσο κοντά το ένα στο άλλο, και ο δίσκος γυρίζει τόσο γρήγορα, που η παροχή δεδομένων (ρυθμός ανάγνωσης) είναι της τάξης των μερικών δεκάδων έως λίγων εκατοντάδων MBytes/s (ας πούμε, αν είναι 100 MBytes/s = 800 Mbits/s = ένα bit κάθε 1.25 ns).

- Άλλο παράδειγμα είναι τα δίκτυα, όπου τα σύρματα (γραμμές μεταφοράς ηλεκτρομαγνητικών κυμάτων) και οι οπτικές ίνες (κυματοδηγοί φωτονίων) συμπεριφέρονται ακριβώς σαν πολύ μακρυές pipelines –σαν τραίνα από bits που κινούνται το ένα πίσω από το άλλο με την ταχύτητα του φωτός. Γιά παράδειγμα, θεωρήστε ένα δίκτυο 10-Gig Ethernet (10 Gbit/s) που περνάει μέσα από μιάν οπτική μήκους 20 km, και μέσω του οποίου ένα πακέτο μεγέθους 1 KByte. Έστω ότι η ταχύτητα του φωτός μέσα στην οπτική ίνα είναι 200 Mm/s (λόγω του δείκτη διάθλασης του υλικού της ίνας, η ταχύτητα του φωτός μέσα στην ίνα είναι χαμηλότερη απ' όσο στο κενό –ας πούμε εδώ τα 2/3 των 300 χιλιάδων km/s του κενού). Η καθυστέρηση εκκίνησης γιά τη μεταφορά του πακέτου μας μέσα από την οπτική ίνα, δηλαδή η καθυστέρηση του πρώτου bit γιά να διανύσει τα 20 km με αυτή την ταχύτητα, θα είναι 20 km / 200 Mm/s = 0.1 ms = 100 μs. Μετά το πρώτο bit, τα επόμενα ακολουθούν σε "απόσταση" (χρονική) 0.1 ns ανά bit (το αντίστροφο του 10 Gbits/s) –αυτό είναι το διαφορικό κόστος του κάθε επιπλέον bit. Άρα, τα 8 Kbits (1 KByte) του πακέτου μας θα χρειαστούν περίπου 8 χιλιάδες φορές το 0.1 ns, δηλαδή περίπου 800 ns = 0.8 μs γιά να φτάσουν, το ένα πίσω από το άλλο. Η συνολική καθυστέρηση λοιπόν είναι 100 + 0.8 = 100.8 μs.

- Παρόμοια είναι και η συμπεριφορά των αρτηριών (buses) μέσα στους υπολογιστές, και των δικτύων από συνδέσμους σημείο-προς-σημείο (point-to-point links) που σήμερα αντικαθιστούν τις αρτηρίες στις ψηλές ταχύτητες –όπως π.χ. το PCI-Express και τα "Networks-on-Chip (NoC)", όπως θα πούμε αμέσως τώρα, στην § 13.3, καθώς και παρακάτω, στην άσκηση 13.7

13.3 Διαφύλλωση (Interleaving), Αλληλοκαλυπτόμενες Δοσοληψίες

Ο *Παραλληλισμός* είναι η βασική τεχνική γιά την αύξηση της παροχής (throughput) μιάς υπηρεσίας, δηλαδή του πόσοι "πελάτες" εξυπηρετούνται ανά μονάδα χρόνου, έστω και αν δεν μειώνεται η καθυστέρηση ("εκκίνησης") της εξυπηρέτησης του καθενός πελάτη χωριστά. Η ομοχειρία (pipelining) είναι η μία μορφή παραλληλισμού, που εφαρμόζεται όταν πολλαπλές *ειδικευμένες* μονάδες εργάζονται ταυτόχρονα σε διαφορετικές φάσεις η καθεμία της εξυπηρέτησης του κάθε "πελάτη". Η άλλη μορφή παραλληλισμού είναι όταν πολλαπλές *ίδιες* μονάδες εργάζονται ταυτόχρονα εξυπηρετώντας (πλήρως) έναν διαφορετικό "πελάτη" καθεμία. Αυτή η δεύτερη μορφή παραλληλισμού, εφαρμοσμένη στις μνήμες, χρησιμοποιείται εδώ και μισό αιώνα γιά την αύξηση της παροχής των μνημών, ονομαζόμενη *Διαφύλλωση Μνήμης (Memory Interleaving)*.

Ένα Διαφυλλωμένο Σύστημα Μνήμης αποτελείται από πολλαπλές μικρότερες μνήμες, που μπορούν να δουλεύουν όλες ταυτόχρονα, προσπελαύνοντας (διαβάζοντας ή γράφοντας) κι από μιά διαφορετική λέξη (ή ομάδα γειτονικών λέξεων) καθεμία. Οι επιμέρους μικρότερες μνήμες ονομάζονται "memory banks", καθεμία. Η καθε λέξη του συνολικού συστήματος μνήμης βρίσκεται αποθηκευμένη σε μία και μόνο μία από τις banks. Όποτε συμβαίνει να χρειαζόμαστε (σχεδόν) ταυτόχρονα δύο η περισσότερες λέξεις (ή ομάδες γειτονικών λέξεων) που βρίσκονται αποθηκευμένες σε διαφορετικές banks καθεμία, μπορούμε να τις προσπελάσουμε (σχεδόν) ταυτόχρονα, βάζοντας τις banks που τις περιέχουν (πολλαπλές ανεξάρτητες μνήμες, δηλαδή) να δουλεύουν εν παραλλήλω. Ο τρόπος διαμοιρασμού των λέξεων (ή των ομάδων γειτονικών λέξεων, π.χ. cache lines) στις banks μιάς διαφυλλωμένης μνήμης ακολουθεί κάποια συνάρτηση κατακερματισμού (hashing) πάνω στις διευθύνσεις των λέξεων, επιλεγμένη ούτως ώστε να μεγιστοποιεί την πιθανότητα οι λέξεις (ή ομάδες) που ζητάμε χρονικά κοντά μεταξύ τους να βρίσκονται σε διαφορετική bank. Μιά συνήθης τέτοια συνάρτηση κατακερματισμού χρησιμοποιεί bits κοντά στο δεξιό (LS) άκρο της διεύθυνσης γιά να επιλέξει την bank, ούτως ώστε προσπελάσεις σε διαδοχικές λέξεις (ή cache lines), π.χ. λόγω σειριακών επισκέψεων σε μεγάλους πίνακες, να πέφτουν σε διαφορετικές banks.

Η παραδοσιακή οργάνωση ενός διαφυλλωμένου συστήματος μνήμης χρησιμοποιεί μιάν αρτηρία (bus) που είναι πολύ γρηγορότερη από τις memory banks που συνδέονται επάνω της. Γιά παράδειγμα, θεωρήστε μιάν αρτηρία (address bus) που μπορεί, κάθε 10 ns, να μεταφέρει κι ένα καινούργιο αίτημα προσπέλασης σε μιά memory bank (δηλαδή μιά διεύθυνση μαζί με τα bits ελέγχου που την συνοδεύουν) (γιά απλότητα, ας θεωρήσουμε μόνο προσπελάσεις ανάγνωσης εδώ). Θεωρήστε ότι στην αρτηρία αυτή συνδέουμε 8 memory banks, που η καθεμία τους έχει καθυστέρηση ανάγνωσης 80 ns. Τέλος, θεωρήστε ότι τα data που βγαίνουν (διαβάζονται) από κάθε bank τροφοδοτούν μιά δεύτερη αρτηρία (data bus) που μπορεί κάθε 10 ns να μεταφέρει πίσω στον επεξεργαστή τα data που διαβάσαμε από μία bank. Τότε, ο επεξεργαστής μπορεί, κάθε 10 ns, να ξεκινάει κι από μιά νέα προσπέλαση μνήμης, παρά το γεγονός ότι θα υπάρχουν ήδη 7 προηγούμενες "εκκρεμείς" (pending) προσπελάσεις, αρκεί η κάθε νέα προσπέλαση να απευθύνεται σε μία (στην) "ελεύθερη" bank, δηλαδή μία (την) bank εκείνη που δεν είναι απασχολημένη με μία από τις υπόλοιπες 7 εκκρεμείς προσπελάσεις. Η κάθε μία προσπέλαση (ανάγνωση) επιστρέφει τα data της μετά

από 80 ns, αλλά κάθε 10 ns υπάρχουν data που επιστρέφουν κι από μία διαφορετική από τις 8 τελευταίες προσπελάσεις που τις ξεκίνησε όλες ο επεξεργαστής πριν τελειώσουν οι 7 προηγούμενες.

Τα σημερινά DRAM chips περιέχουν μέσα τους πολλαπλές διαφυλλωμένες banks (συνήθως μεταξύ 4 και 16 banks). Το ρόλο των γρήγορων αρτηριών που είναι κοινές μεταξύ των banks τον παίζουν αφ' ενός τα address pins γιά τις αιτήσεις πρόσβασης, αφ' ετέρου τα data pins γιά τις απαντήσεις στις αναγνώσεις. Η διαφύλλωση (παραλληλισμός) μέσα σ' ένα DRAM chip είναι επωφελής διότι, π.χ., την ώρα που σ' ένα bank βρίσκεται σε εξέλιξη η ανάγνωση μιάς νέας γραμμής (που είναι μιά αργή διαδικασία), ένα άλλο bank επιστρέφει τα data μιάς προηγούμενης ανάγνωσης μέσω των data pins, κ.ο.κ.

Η αρτηρία που τροφοδοτεί ένα διαφυλλωμένο σύστημα μνήμης, ή ένα δίκτυο που αποτελεί την εξέλιξη της αρτηρίας (όπως το PCI Express) και που προορίζεται γιά την επικοινωνία ανάλογων *παράλληλων* συστημάτων –π.χ. πολλαπλοί πυρήνες, πολλαπλά περιφερειακά, συστοιχίες παράλληλων δίσκων, κλπ– πρέπει να έχει μιά χαρακτηριστική ιδιότητα που δεν την συναντούσαμε στις παραδοσιακές αρτηρίες: πρέπει να υποστηρίζει **πολλαπλές, χρονικά Αλληλοκαλυπτόμενες Δοσοληψίες** (multiple Transactions, Overlapping in time), ή αλλοιώς επονομαζόμενες πολλαπλές **Εκκρεμείς Δοσοληψίες** (multiple Pending Transactions), ή **Split Transactions**. Η ιδέα είναι η εξής: μία από τις επικοινωνούσες συσκευές (ο αιτών – requester, master) ζητά μία Δοσοληψία (Transaction) από μιάν άλλη συσκευή (ο απαντών – replyier, slave) και περιμένει μιάν απάντηση· π.χ. ο αιτών ζητά μιάν ανάγνωση, στέλνοντας μιά διεύθυνση, και ο απαντών θα απαντήσει αργότερα, στέλνοντας τα αιτηθέντα δεδομένα. Η δοσοληψία αυτή, συνήθως σήμερα, διαρκεί αρκετή ώρα, σε σχέση με το πόσο γρήγορο είναι το μέσο επικοινωνίας (αρτηρία, δίκτυο). Τον παλαιό καιρό, μέχρι να έλθει πίσω η απάντηση και να ολοκληρωθεί έτσι η δοσοληψία, δεν μπορούσε καμιά άλλη συσκευή από τις συνδεδεμένες στην αρτηρία να ξεκινήσει καινούργια δοσοληψία, προκειμένου να μην "ανακατωθούν" μεταξύ τους οι δοσοληψίες και προκύψει "μπέρδεμα"· έτσι, η αρτηρία καθόνταν ανενεργή (idle) όση ώρα έκανε ο απαντών να απαντήσει.

Η σημερινή, καλύτερη οργάνωση επιτρέπει "χωριστές" δοσοληψίες (split transactions): Η φάση του αιτήματος (request) της κάθε δοσοληψίας είναι ξεχωριστή από τη φάση της απάντησης (reply), και μεταξύ των δύο επιτρέπεται να παρεμβληθούν μία ή περισσότερες ανάλογες φάσεις (αιτήματα, απαντήσεις) μιάς ή περισσοτέρων άλλων δοσοληψιών. Ουσιαστικά δηλαδή, επιτρέπεται λειτουργία ανάλογη της διαφυλλωμένης μνήμης: ένας ή περισσότεροι αιτούντες επιτρέπεται να αποστείλουν πολλαπλά αιτήματα σε μία η περισσότερες συσκευές που θα τα απαντήσουν, προτού επιστρέψει η απάντηση στο πρώτο από τα αιτήματα. Με άλλα λόγια, επιτρέπεται οι δοσοληψίες να αλληλοκαλύπτονται χρονικά –η δεύτερη να αρχίζει πριν τελειώσει η πρώτη, κ.ο.κ.– άρα επιτρέπεται ο παραλληλισμός μεταξύ δοσοληψιών. Αυτό σημαίνει ότι επιτρέπονται πολλαπλές "εκκρεμείς" (pending) δοσοληψίες: όσο η πρώτη δεν έχει τελειώσει ακόμα, άρα είναι εκκρεμής, ξεκινάει η δεύτερη, και είναι κι εκείνη εκκρεμής μέχρι να τελειώσει, κ.ο.κ.

Από την άποψη της υλοποίησης, μία από τις σημαντικότερες απαιτήσεις γιά να δουλέψουν σωστά όλα αυτά είναι να υπάρχει ένας τρόπος όταν έρχεται μιά

απάντηση να ξέρουμε **γιά ποιό** από τα προηγούμενα αιτήματα αποτελεί απάντηση αυτή που τώρα έρχεται! Εάν όλα τα αιτήματα παίρνουν σταθερό χρόνο να απαντηθούν, τότε μπορεί να προδιαγραφεί ότι τα αιτήματα θα απαντώνται πάντα **με τη σειρά τους (in order)**, και άρα η κάθε απάντηση αφορά πάντα το παλαιότερο εκκρεμές αίτημα· τότε, μιά ουρά FIFO αρκεί γιά να θυμόμαστε τα εκκρεμή αιτήματα, και να τα "ταιριάζουμε" με τις αφικνούμενες απαντήσεις. Όμως, δυστυχώς, συνήθως, τα διάφορα αιτήματα, πιθανόν σε διαφορετικές συσκευές και σε διαφορετικές "αποστάσεις" στο δίκτυο, **δεν** παίρνουν σταθερό και πάντα τον ίδιο χρόνο να απαντηθούν, άρα οι απαντήσεις έρχονται **εκτός σειράς (out-of-order – ooo)**. Τότε, απαιτείται η κάθε δοσοληψία να συνοδεύεται από έναν "αριθμό (κωδικό) ταυτοποίησης" (**tranasction ID** – identifier), που να είναι εκάστοτε μοναδικός μεταξύ των εκκρεμών εκείνη την ώρα δοσοληψιών, και όταν έρχεται μιά απάντηση να υπάρχει ένα κύκλωμα που να την "ταιριάζει", βάσει αυτού του ID, με το αντίστοιχο αίτημα που είχε αυτό το ID· όταν ολοκληρωθεί έτσι η δοσοληψία, απελευθερώνεται και μπορεί να ανακυκλωθεί και ο αριθμός (κωδικός) που είχε πάρει αυτή η δοσοληψία.

Άσκηση 13.4: Απεικόνιση Μνήμης των Μονάδων Ε/Ε (Memory Mapped I/O)

Οπως είπαμε στο μάθημα, ένας συνηθισμένος τρόπος επικοινωνίας επεξεργαστή-μονάδων εισόδου/εξόδου (E/E - περιφερειακές συσκευές) είναι η "απεικόνιση μνήμης" των μονάδων E/E (memory-mapped I/O). Σε τέτοια συστήματα, ένα μέρος του "χώρου" φυσικών διευθύνσεων αντιστοιχεί στην κύρια μνήμη του υπολογιστή, ενώ οι υπόλοιπες φυσικές διευθύνσεις αντιστοιχούν στις περιφερειακές συσκευές. Αυτό σημαίνει ότι εντολές load και store των οποίων η εικονική διεύθυνση μεταφράζεται σε τέτοιες "άλλες" φυσικές διευθύνσεις προκαλούν μεταφορά δεδομένων από την εκάστοτε επιλεγόμενη περιφερειακή συσκευή προς τον επεξεργαστή (load) ή αντίστροφα (store), αντί να διαβάζουν ή να γράφουν μία θέση κύριας μνήμης. Η προστασία των περιφερειακών συσκευών (π.χ. αρχεία στο δίσκο) από ανεπίτρεπτες/κακόβουλες, λανθασμένες, ή ταυτόχρονες προσβάσεις εξσφαλίζεται με το να μην απεικονίζει το λειτουργικό σύστημα *καμία* εικονική σελίδα *χρήστη* σε φυσική σελίδα που αντιστοιχεί σε περιφερειακές συσκευές, και να "εμφανίζει" αυτές τις φυσικές σελίδες μόνο στο χώρο εικονικών διευθύνσεων του λειτουργικού συστήματος (πλήν περιπτώσεων ειδικών συσκευών και ειδικών χρηστών που επιτρέπεται να αποκτούν κατ'ευθείαν πρόσβαση σε αυτές).

Σαν απλοϊκό παράδειγμα, για τους σκοπούς αυτής της άσκησης, θεωρήστε ότι μιλάμε για ένα σύστημα κύριας μνήμης και συσκευών E/E που βλέπει φυσικές διευθύνσεις λέξεων (όχι bytes, δήλαδή έχουν ήδη αφαιρεθεί τα 2-3 LS bits της διεύθυνσης του επεξεργαστή) μεγέθους (οι φυσικές διευθύνσεις λέξεων) 11 bits. Τον αντίστοιχο χώρο φυσικών διευθύνσεων, μεγέθους 2048 λέξεων, αποφασίζουμε να μοιράσουμε ως εξής:

- `0xxxxxxxxxx`: 1024 λέξεις κύριας μνήμης (main memory).
- `10xxxxxxxxx`: 512 λέξεις για μία "μεγάλη" περιφερειακή συσκευή (δηλαδή μία συσκευή που περιέχει έναν μεγάλο buffer δεδομένων E/E που θέλουμε να μπορεί να βλέπει ο επεξεργαστής).
- `111110000xx`: 4 λέξεις κατανεμημένες σε 2 "μικρές" συσκευές E/E, την "IN" στις 2 μικρότερες διευθύνσεις, και την "OUT" στις 2 ψηλότερες, όπου η κάθε

μία από αυτές τις συσκευές έχει 2 εσωτερικούς καταχωρητές --τον "status" στην ζυγή διεύθυνση, και τον "data" στη μονή διεύθυνση.

(α) Σχεδιάστε, χρησιμοποιώντας πύλες AND (οσωνδήποτε εισόδων) και NOT, τον αποκωδικοποιητή διευθύνσεων που επιλέγει τη συσκευή (και τον καταχωρητή) που πρέπει να ενεργοποιηθεί κάθε φορά. Είσοδος του αποκωδικοποιητή είναι τα 11 σύρματα φυσικής διεύθυνσης λέξεων από τον επεξεργαστή (μετά τη μετάφραση της εικονικής διεύθυνσης από το TLB). Ο αποκωδικοποιητής έχει 6 σύρματα εξόδου: 1 για την κύρια μνήμη, 1 για την μεγάλη συσκευή E/E, 2 γιά την IN, και 2 γιά την OUT· γιά κάθε μιά από τις IN και OUT, το πρώτο σύρμα λέει πότε απευθυνόμαστε στον καταχωρητή της "status" και το δεύτερο λέει πότε απευθυνόμαστε στον "data". Σχεδιάστε το κύκλωμα που γεννά αυτά τα 6 σύρματα, δείχνοντας προσεκτικά ποια σύρματα εισόδου και ποιάς πολικότητας χρησιμοποιείτε σε κάθε πύλη AND.

(β) Σε ποιά λέξη μνήμης (στο δεκαδικό, αρχίζοντας από την 0), ή σε ποιά θέση του buffer της μεγάλης συσκευής (πάλι στο δεκαδικό, αρχίζοντας από την 0), ή σε ποιόν καταχωρητή ποιάς συσκευής αναφέρεται κάθε μία από τις εξής φυσικές διευθύνσεις λέξεων που δίδονται στο δεκαεξαδικό σύστημα, ή ποιές από αυτές είναι παράνομες (ανύπαρκτες): 000, 00E, 100, 200, 2FF, 300, 3FF, 400, 4FF, 500, 5FF, 600, 608, 60A, 60E, 60F, 610, 680, 6C0, 6C3, 6CF, 6FF, 700, 708, 70A, 70E, 70F, 710, 780, 7C0, 7C3, 7CF, 7FF, 800, 808, 80A, 80E, 80F, C0E, C0F, F02, F12.

Άσκηση 13.5: Καταχωρητές Κατάστασης, Busy Wait, Polling

Φυσικά, οι μονάδες E/E δεν είναι πραγματική μνήμη: συχνά, διαβάζοντας από ορισμένη διεύθυνση, *δεν* παίρνει ο επεξεργαστής την ίδια τιμή με αυτήν που είχε γράψει εκεί την τελευταία φορά που έγραψε σα αυτήν (ο επεξεργαστής) --παίρνει την τιμή που θέλει να του δώσει κάθε φορά η μονάδα E/E, η οποία τιμή συχνά αλλάζει με το χρόνο. Επίσης, τέτοιες αναγνώσεις από περιφερειακές συσκευές μπορούν να έχουν "παρενέργειες" (side-effects), όπως π.χ. να θέτουν ή να μηδενίζουν σημαίες (flag bits) που υποδεικνύουν π.χ. ότι διαβάστηκε η παρούσα τιμή εισόδου και δεν έχει έλθει ακόμα η επόμενη (νέα) τιμή εισόδου. Ομοίως, εγγραφή σε ορισμένη διεύθυνση περιφερειακής συσκευής μπορεί να προκαλεί π.χ. μετάδοση της πληροφορίας σε κάποιο σύρμα/δίκτυο, και όχι πραγματική εγγραφή σε κάποια flip-flops που να μπορούμε αργότερα να τα διαβάσουμε, και ενδέχεται επίσης η εγγραφή αυτή να προκαλεί και άλλες παρενέργειες όπως π.χ. μηδενισμό ενός flag που υποδεικνύει ότι παρελήφθη η παρούσα τιμή εξόδου και ότι η συσκευή δεν είναι ακόμα έτοιμη να παραλάβει την επόμενη τιμή.

Επειδή οι μονάδες E/E δεν συμπεριφέρονται σαν πραγματική μνήμη, οι τιμές που διαβάζουμε ή γράφουμε στις διευθύνσεις τους πρέπει να *μην* κρατιούνται στην *κρυφή* μνήμη, ειδάλως θα διαβάζουμε παλιές τιμές ή αυτά που γράφουμε δεν θα φτάνουν όλα ή αμέσως στις συσκευές E/E. Αυτό, το να παρακάμπτουν δηλαδή οι προσπελάσεις αυτές την κρυφή μνήμη, επιτυγχάνεται συνήθως με ένα επιπλέον bit στον πίνακα σελίδων, που να υποδεικνύει "non-cacheable pages", ή με το να αναγνωρίζει η κρυφή μνήμη την ειδική μορφή των φυσικών διευθύνσεων των συσκευών E/E. Επίσης, και ο Compiler πρέπει να είναι ενήμερος για τη συμπεριφορά αυτών των "θέσεων" ως μη κανονική μνήμη, ούτως ώστε π.χ. να μην αντικαθιστά πολλαπλές αναγνώσεις (load) από εκεί με χρήση τυχόν προηγουμένων αντιγράφων τους σε καταχωρητές, και σε περίπτωση πολλαπλών

συνεχόμενων εγγραφών εκεί να μην απαλείφει όλες εκτός της τελευταίας εγγραφής. Για να ενημερωθεί ο Compiler για αυτή την ειδική συμπεριφορά, πρέπει αυτές οι "θέσεις μνήμης" να δηλωθούν ως **volatile** στο πρόγραμμα που τις προσπελάζει.

Ένα άλλο σύστημα επικοινωνίας επεξεργαστή-συσκευών Ε/Ε, διαφορετικό από την απεικόνιση μνήμης των μονάδων Ε/Ε, είναι η ύπαρξη ειδικών εντολών εισόδου/εξόδου (I/O instructions) στο ρεπερτόριο εντολών του επεξεργαστή. Οι εντολές εισόδου μοιάζουν με τις load και οι εντολές εξόδου μοιάζουν με τις store, όμως οι εντολές Ε/Ε είναι "προνομιούχες" (priviledged), δηλαδή επιτρέπεται να εκτελούνται μόνο σε "kernel mode", και οι εντολές Ε/Ε ειδοποιούν την κρυφή μνήμη να μην παρέμβει. Κατά τα άλλα, στις αρτηρίες Ε/Ε, οι εντολές Ε/Ε μάλλον καταλήγει να δίνουν διευθύνσεις εντελώς ανάλογες προς αυτές που δίνουν οι εντολές load/store στα συστήματα με απεικόνιση μνήμης των μονάδων Ε/Ε.

Σε αυτή την άσκηση, θεωρήστε ότι η "μικρή" συσκευή IN της άσκησης 13.4 είναι μία συσκευή εισόδου από πληκτρολόγιο, με τους καταχωρητές "κατάστασης" (status) και "δεδομένων" (data) που είπαμε. Μόλις έλθει νέος χαρακτήρας από το πληκτρολόγιο, η συσκευή θέτει τον καταχωρητή κατάστασης στην τιμή 1, και θέτει τον καταχωρητή δεδομένων στην τιμή που αποτελεί τον κώδικα ASCII του χαρακτήρα που ήλθε. Ανάγνωση (από πλευράς επεξεργαστή) του καταχωρητή κατάστασης δεν έχει παρενέργειες, ενώ ανάγνωση του καταχωρητή δεδομένων προκαλεί μηδενισμό του καταχωρητή κατάστασης (μέχρι να έλθει ο επόμενος χαρακτήρας --έτσι ξεχωρίζουμε, αν πατηθεί το ίδιο πλήκτρο πολλές φορές, πόσες φορές πατήθηκε).

(α) Γράψτε μία διαδικασία (procedure) "**`read_kbd_busywait_char()`**" σε C (ή, στην ανάγκη, σε ψευδοκώδικα στυλ C) η οποία επιστρέφει τον επόμενο χαρακτήρα από το πληκτρολόγιο αυτό. Όπως λέει και το όνομά της, η διαδικασία αυτή θα κάνει "busy wait", δηλαδή θα περιμένει να έλθει ο επόμενος χαρακτήρας απασχολώντας εν τω μεταξύ τον επεξεργαστή με το να ελέγχει συνεχώς, ξανά και ξανά, εάν ήλθε χαρακτήρας (ανάγνωση του καταχωρητή κατάστασης) --φυσικά, πρόκειται για πολύ κακό στυλ προγραμματισμού, αλλά από κάπου πρέπει να ξεκινήσουμε... Η διαδικασία θα επιστρέφει τον χαρακτήρα (char) που ήλθε. Θεωρήστε ότι η διαδικασία θα τρέχει σε kernel mode (θα είναι μέρος του λειτουργικού συστήματος), και ότι, όταν θα τρέχει, η μετάφραση εικονικών διευθύνσεων σε φυσικές θα είναι η συνάρτηση ταυτότητας, δηλαδή η φυσική διεύθυνση θα ισούται με την εικονική που την γέννησε. Χρησιμοποιήστε type casting, από τις σταθερές ακέραιες ποσότητες των διευθύνσεων που ξέρετε, για να αρχικοποιήστε τους pointers (κατάλληλου είδους) που θα χρειαστείτε για προσπέλαση στους καταχωρητές της συσκευής.

(β) Η παραπάνω διαδικασία (α) είναι πολύ κακιά, διότι δεν αφήνει τον επεξεργαστή να κάνει τίποτα άλλο όσην ώρα αυτός περιμένει να πληκτρολογηθεί ο επόμενος χαρακτήρας. Όπως είπαμε και στο μάθημα, ένας καλύτερος τρόπος είναι να εκτελεί ο επεξεργαστής διάφορα προγράμματα, και, περιοδικά, όποτε έρχεται διακοπή από το ρολόϊ πραγματικού χρόνου (συνήθως 50 με 120 Hz --άλλο από το ρολόϊ του επεξεργαστή, των GHz ή εκατοντάδων MHz), μεταξύ άλλων περιοδικών εργασιών, να ελέγχει και εάν ήλθε κάποιος νέος χαρακτήρας από το πληκτρολόγιο (αρκεί οι χαρακτήρες να μην έρχονται πιό γρήγορα από τις διακοπές, πράγμα που ισχύει για πληκτρολόγιο και 50-120 Hz ρυθμό διακοπών).

Ο τρόπος αυτός λέγεται **δειγματοληψία (polling)** (ή περιόδευση), διότι ο επεξεργαστής παίρνει ένα "δείγμα" από την κατάσταση του πληκτρολογίου κάθε 10 με 20 ms (50-100 Hz). Γράψτε μία νέα διαδικασία "**`read_kbd_polling_char()`**", ανάλογη με την προηγούμενη, αλλά αυτή τη φορά χωρίς αναμονή. Εάν έχει έλθει νέος χαρακτήρας από το προηγούμενο κάλεσμα στην read_kbd_polling_char(), τότε θα επιστρέφει αυτόν τον χαρακτήρα, αλλοιώς (αν δεν έχει έλθει νέος χαρακτήρας) θα επιστρέφει (αμέσως) `'\0'`.

Άσκηση 13.6: Κόστος Ε/Ε βάσει Δειγματοληψίας και βάσει Διακοπών

Η περιοδική **δειγματοληψία (polling)** που είδαμε στην παραπάνω άσκηση 13.5 είναι ένας ρεαλιστικός τρόπος εισόδου/εξόδου (Ε/Ε - I/O), αρκεί η συχνότητα δειγματοληψίας να είναι αρκούντως ψηλή ώστε να μην χάνονται είσοδοι ή να μην καθυστερεί η έξοδος. Το μειονέκτημα της δειγματοληψίας είναι η σπατάλη χρόνου για την ανάγνωση του καταχωρητή κατάστασης όταν δεν έχει έλθει ακόμα νέα είσοδος ή δεν έχει τελειώσει ακόμα η προηγούμενη πράξη εξόδου.

΄Ενας εναλλακτικός τρόπος εισόδου/εξόδου είναι **Ε/Ε βάσει διακοπών (interrupt-driven I/O)**: η περιφερειακή συσκευή διακόπτει (interrupt) τον επεξεργαστή όταν υπάρχουν νέα δεδομένα εισόδου γι' αυτόν, ή όταν είναι έτοιμη να δεχτεί νέα δεδομένα εξόδου από αυτόν. ΄Ετσι, δεν σπαταλιέται χρόνος για δειγματοληψία χωρίς λόγο της συσκευής, όσο αυτή δεν είναι ακόμα έτοιμη. Το κόστος, πάντως, της Ε/Ε βάσει διακοπών είναι η ειδική φροντίδα (overhead) που απαιτεί η κάθε διακοπή, δεδομένου ότι αυτή αλλάζει τη διεργασία που τρέχει, τα περιεχόμενα της κρυφής μνήμης και του TLB, και απαιτεί δαπανηρή καταγραφή στοιχείων (book-keeping) για να λειτουργήσει σωστά. Αντ' αυτού, η δειγματοληψία μπορεί να έχει το πλεονέκτημα, ανάλογα με την περίπτωση, ότι δειγματοληπτεί "μία και καλή" πολλές συσκευές Ε/Ε για κάθε μία διακοπή από το ρολόϊ (batch processing), αντί να υφίσταται "κάθε τρείς και λίγο" το κόστος μίας επιπλέον διακοπής από μίαν άλλη συσκευή. Γιά να αποφασίσουμε τι μας συμφέρει μας ενδιαφέρουν τρεις παράμετροι:

- Πόσο κοντά χρονικά μπορεί να συμβούν δύο γεγονότα εισόδου; Η περίοδος δειγματοληψίας πρέπει να είναι βραχύτερη από αυτό, προκειμένου να μην χάσουμε το δεύτερο γεγονός. Προκειμένου περί εξόδου, πόσο δεχόμαστε να καθυστερήσουμε από την ολοκλήρωση μίας πράξης εξόδου μέχρι να το αντιληφθεί ο επεξεργαστής και να προχωρήσει στην επόμενη; Την παράμετρο αυτή μπορούμε να την μεγαλώσουμε (άρα σπανιότερη δειγματοληψία) αν η συσκευή Ε/Ε έχει έναν μεγαλύτερο ενταμιευτή (buffer) που να μπορεί να κρατά μέσα του περισσότερα δεδομένα εισόδου ή εξόδου (περισσότερη δουλειά κάθε φορά).
- Πόσος είναι ο μέσος ρυθμός των γεγονότων εισόδου; Δηλαδή, ανεξάρτητα αν δύο γεγονότα εισόδου ενδέχεται να συμβούν πολύ κοντά μεταξύ τους, *κατά μέσον όρο* πόσο κοντά χρονικά θα συμβαίνουν; Οσο σπανιότερη είναι η κατά μέσον όρο εμφάνισή τους, τόσο μεγαλύτερη είναι η σπατάλη της άσκοπης δειγματοληψίας.
- Πόσο πολλές συσκευές Ε/Ε δειγματοληπτούμε μαζί σε κάθε διακοπή του ρολογιού (batching factor); Οσο περισσότερες είναι αυτές, τόσο περισσότερο αποσβένεται μεταξύ τους το κόστος της διακοπής του ρολογιού.

Θεωρήστε, σε αυτήν την άσκηση, ότι το ρολόϊ του επεξεργαστή είναι 1 GHz (άσχετο με το ρολόϊ πραγματικού χρόνου που μας δίνει περιοδικές διακοπές), ότι η ειδική φροντίδα (overhead) για κάθε διακοπή είναι δύο χιλιάδες (2000) κύκλοι του ρολογιού του επεξεργαστή, και ότι το κόστος δειγματοληψίας μίας συσκευής Ε/Ε είναι διακόσιοι (200) κύκλοι του ρολογιού του επεξεργαστή (η κύρια αιτία αυτής της καθυστέρησης είναι το ότι οι αρτηρίες Ε/Ε (I/O buses) είναι πολύ πιό αργές από τους (γρήγορους) σημερινούς επεξεργαστές). Θέλουμε να υπολογίσουμε τι ποσοστό του συνολικού χρόνου του επεξεργαστή θα απορροφά η Ε/Ε στις παρακάτω περιπτώσεις, όταν αυτή γίνεται βάσει δειγματοληψίας ή βάσει διακοπών.

(α) Εστω ένας υπολογιστής ο οποίος λαμβάνει και καταγράφει σήματα από 40 απομακρυσμένα σημεία. Κάθε μία από τις 40 γραμμές εισόδου ενδέχεται να φέρνει νέες εισόδους κάθε 1 ms, δηλαδή με μέγιστο ρυθμό 1 kHz. Εαν χρησιμοποιήσουμε δειγματοληψία, επομένως, το ρολόϊ πρέπει να μας δίνει 1 διακοπή ανά 1 ms. Σε κάθε διακοπή, δειγματοληπτούμε 40 συσκευές. Πόσους κύκλους ρολογιού (του επεξεργαστή) ξοδεύουμε σε κάθε διακοπή, (i) για την ίδια τη διακοπή, και (ii) για τις 40 δειγματοληψίες; Δεδομένου ότι αυτό επαναλαμβάνεται 1000 φορές το δευτερόλεπτο, πόσους κύκλους ρολογιού ανά s ξοδεύουμε για Ε/Ε; Τι ποσοστό της συνολικής υπολογιστικής δυναμικότητας του επεξεργαστή αντιπροσωπεύουν αυτοί οι κύκλοι;

(β) Έστω ότι στο σύστημα (α), παρ' ότι νέες είσοδοι μπορεί να έρχονται σχετικά κοντά η μία με την άλλη (κάθε 1 ms), ο μέσος ρυθμός άφιξής τους είναι σημαντικά αραιότερος: κατά μέσον όρο έρχονται 40 νέες είσοδοι ανά δευτερόλεπτο ανά γραμμή εισόδου. Συνολικά, για όλες τις γραμμές, πόσες είναι οι νέες είσοδοι ανά s; Έστω ότι κάνουμε Ε/Ε βάσει διακοπών, και ότι κάθε νέα είσοδος (από οιαδήποτε γραμμή) προκαλεί μία διακοπή. Πόσες διακοπές ανά δευτερόλεπτο θα έχουμε, κατά μέσον όρο; Πόσους κύκλους ρολογιού θα ξοδεύει ο επεξεργαστής για να τις εξυπηρετήσει; Τι ποσοστό της συνολικής υπολογιστικής του δυναμικότητας αντιπροσωπεύουν αυτοί; Συμφέρει η δειγματοληψία (α) ή οι διακοπές (β);

(γ) Έστω τώρα ότι στο σύστημα (α) αυξάνεται ο μέσος ρυθμός άφιξης νέων εισόδων, από λίγες δεκάδες ανά γραμμή ανά δευτερόλεπτο που ήταν στο (β) σε 400 ανά γραμμή ανά δευτερόλεπτο (δηλαδή πλησιάζει περισσότερο στο μέγιστο ρυθμό, που είναι 1 kHz). Το κόστος της δειγματοληψίας δεν αλλάζει, αφού αυτή ούτως ή άλλως επισκέπτεται την κάθε γραμμή 1000 φορές το δευτερόλεπτο. Όμως, στη μέθοδο βάσει διακοπών, αυξάνει το μέσο πλήθος διακοπών ανά δευτερόλεπτο. Πώς αλλάζουν οι απαντήσεις σας της ερώτησης (β) εδώ; Συμφέρει η δειγματοληψία ή οι διακοπές, τώρα;

(δ) Στις περιπτώσεις (β) και (γ), κινδυνεύουμε να χάσουμε κάποια νέα είσοδο αν "πέσουν μαζεμένες" νέες είσοδοι από όλες τις γραμμές; Έστω ότι αμέσως μετά την έλευση μίας νέας εισόδου από τη γραμμή Α, μας έρχονται νέες είσοδοι και από τις 39 άλλες γραμμές. Γιά να εξυπηρετήσει ο επεξεργαστής τις 40 αυτές διακοπές (τη μία μετά την άλλη), πόσους κύκλους επεξεργαστή χρειάζεται; Πόσος χρόνος είναι αυτός; Το νωρίτερο που μπορεί να έλθει η επόμενη είσοδος από τη γραμμή Α είναι 1 ms μετά την προηγούμενη, όπως είπαμε στο (α). Θα έχει προλάβει ο επεξεργαστής να εξυπηρετήσει τις παραπάνω 40 διακοπές πριν έλθει η επόμενη αυτή είσοδος από τη γραμμή Α, ή θα την χάσει αυτή την επόμενη είσοδο;

(ε) Έστω τώρα ότι αντί των 40 εισόδων του (α) ο υπολογιστής μας έχει μία μόνο -- αλλά πολύ γρηγορότερη-- είσοδο: μία γραμμή δικτύου των 32 Mbit/s, δηλαδή 4 MBytes/s. Έστω ότι η συσκευή εισόδου μπορεί να κρατήσει (έχει buffer για να κρατήσει) 1 πακέτο, αλλά όχι παραπάνω. Η συσκευή μας δίνει 1 διακοπή για κάθε 1 αφικνούμενο πακέτο. Έστω ότι τα μικρότερα δυνατά πακέτα είναι μεγέθους 40 Bytes καθένα (όπως στο πρωτόκολλο του Διαδικτύου, το IP). Άρα, ο μέγιστος δυνατός ρυθμός άφιξης πακέτων είναι 4000 KBytes/s διά 40 Bytes ανά πακέτο = 100 K πακέτα/s. Εάν μας έρχονται πακέτα ελαχίστου μεγέθους και με τον μέγιστο δυνατό ρυθμό ("κολλητά" το ένα με το άλλο), ξανα-απαντήστε την ερώτηση (β).

(στ) Σήμερα, μιά γραμμή δικτύου μέτριας ταχύτητας είναι του 1 Gbit/s, δηλαδή περίπου 30 φορές γρηγορότερη από αυτήν του (ε). Μπορεί ο υπολογιστής μας να την αντέξει αν η συσκευή εισόδου συνεχίσει να έχει ενταμευτή μόνο για ένα πακέτο και συνεχίσει να μας δίνει μία διακοπή για κάθε αφικνούμενο πακέτο;;;

Άσκηση 13.7: Απευθείας Πρόσβαση Μνήμης (DMA) από Συσκευές Ε/Ε

Από τα παραπάνω νούμερα φάνηκε ότι οι γρήγορες συσκευές Ε/Ε πρέπει να έχουν μεγάλους ενταμευτές (buffers), ούτως ώστε οι διακοπές --είτε του ρολογιού είτε των συσκευών-- να μην είναι πολύ συχνές, και να μπορεί μεγάλη "ποσότητα εργασίας" να συσσωρεύεται στον ενταμευτή μεταξύ διαδοχικών "επισκέψεων" στη συσκευή από τον επεξεργαστή. Ακόμα και με αυτή τη λύση, όμως, για να μην είναι οι διακοπές πολύ συχνές, υπάρχει και ένα άλλο πρόβλημα επιδόσεων για τις γρήγορες συσκευές Ε/Ε:

Γιά να αντιγράψει ο επεξεργαστής ένα μεγάλο όγκο δεδομένων ανάμεσα στον ενταμευτή της περιφερειακής συσκευής και την κυρίως μνήμη του υπολογιστή, απαιτούνται πολλοί κύκλοι ρολογιού, επειδή οι αρτηρίες Ε/Ε (λεωφόροι Ε/Ε - I/O buses) είναι πολύ πιό αργές από τους σημερινούς (γρήγορους) επεξεργαστές. Δεδομένου ότι η αντιγραφή αυτή είναι μία πολύ απλή εργασία, θα αποτελούσε σπατάλη δαπανηρών υπολογιστικών πόρων (του επεξεργαστή) το να βάζουμε τον επεξεργαστή να την κάνει: ο επεξεργαστής, σε αυτή τη δουλειά, θα σπαταλά την περισσότερη ώρα του περιμένοντας να απαντήσει η αρτηρία Ε/Ε. Η ενδεδειγμένη λύση είναι να αποκτήσει η περιφερειακή συσκευή τη δυνατότητα να κάνει μόνη της την αντιγραφή ανάμεσα στο ενταμευτή της και στην κύρια μνήμη: Η "**Απευθείας Πρόσβαση Μνήμης (Direct Memory Access - DMA)**" από τις συσκευές Ε/Ε λειτουργεί ως εξής. Η συσκευή Ε/Ε έχει δύο ή τρείς καταχωρητές ελέγχου για τη λειτουργία DMA:

a. Διεύθυνση έναρξης --είναι η (φυσική) διεύθυνση μνήμης προς την οποία ή από την οποία θα αρχίσει η αντιγραφή δεδομένων.
b. Μέγεθος μεταφοράς --είναι το πλήθος των Bytes που θα αντιγραφούν.
c. Καταχωρητής ενεργοποίησης --είναι ο καταχωρητής εκείνος όπου μόλις ο επεξεργαστής γράψει έναν ειδικό κώδικα θα αρχίσει η αντιγραφή (αυτός μπορεί και να μην υπάρχει, π.χ. εάν η αντιγραφή αρχίζει με το που γράφουμε το μέγεθος μεταφοράς στον δεύτερο καταχωρητή).

Η συσκευή Ε/Ε έχει επίσης μία μικρή μηχανή πεπερασμένων καταστάσεων (FSM), η οποία, μόλις δοθεί το σήμα εκκίνησης, κάνει την εξής δουλειά κατ' επανάληψη:

i. Ζητά να της δοθεί η χρήση της αρτηρίας μνήμης (ή των αρτηριών E/E και μνήμης, ή του/των σχετικών συνδέσμων).
ii. Μόλις της δοθεί η χρήση, αντιγράφει την επόμενη "λέξη" του ενταμιευτή της συσκευής στην κυρίως μνήμη, εκεί που δείχνει ο καταχωρητής διεύθυνσης, ή την αντιγράφει από την κυρίως μνήμη στον ενταμιευτή της συσκευής. [Στο βήμα αυτό, η "λέξη" που αντιγράφεται συχνά δεν είναι μία μόνο λέξη του επεξεργαστή ή της αρτηρίας, αλλά μία μικρή ομάδα (burst - ριπή) λέξεων, προκειμένου να εκμεταλλευτούμε τη δυνατότητα των DRAM για οικονομικότερη προσπέλαση συνεχόμενων λέξεων (§ 13.1), καθώς και να αποσβέσουμε καλύτερα το overhead απόκτησης χρήσης της αρτηρίας μέσω της μεταφοράς περισσοτέρων Bytes κάθε φορά που την αποκτούμε].
iii. Αυξάνει τον καταχωρητή διεύθυνσης κατά το πλήθος των Bytes που μόλις μετέφερε.
iv. Μειώνει τον καταχωρητή μεγέθους μεταφοράς κατά το πλήθος των Bytes που μόλις μετέφερε.
v. Αν ο καταχωρητής μεγέθους είναι ακόμα μεγαλύτερος του μηδενός, επαναλαμβάνει από το (i).

Ας θεωρήσουμε σε αυτή την άσκηση τον ίδιο επεξεργαστή με ρολόϊ 1 GHz που είχαμε και παραπάνω, και έναν σύνδεσμο επικοινωνίας περιφερειακών - μνήμης - επεξεργαστή τύπου PCI-Express των 2.5 Gbits/s (δηλ. παλαιότερης γενιάς), και "στενό", δηλ. της μίας μόνο "λωρίδας" (x1). Ο ρυθμός μετάδοσης αυτού του συνδέσμου είναι 2.5 Gb/s = 2500 Mb/s = περίπου 312 Mbytes/s, δηλαδή περίπου 1 Byte κάθε 3.2 ns ή αλλοιώς 1 Byte κάθε 3.2 κύκλοι ρολογιού του επεξεργαστή μας. Πάνω στο σύνδεσμο αυτό οι πληροφορίες μεταφέρονται σε "πακέτα" (όπως στα δίκτυα), και κάθε πακέτο έχει μιάν επικεφαλίδα (header) ας πούμε των 12 Bytes (π.χ. διευθύνσεις πηγής, προορισμού, κωδικό πράξης, έλεγχος σφαλμάτων, κλπ) ακολουθούμενη από τα μεταφερόμενα Bytes δεδομένων.

(α) Έστω ότι δεν υπάρχει DMA, και ο επεξεργαστής κάνει την αντιγραφή μεταξύ μνήμης και ενταμιευτή περιφερειακής συσκευής, ας πούμε από τη μνήμη προς την περιφερειακή συσκευή. Η αντιγραφή γίνεται με ένα μικρό βρόχο που περιλαμβάνει μία εντολή load από τη μνήμη και μία εντολή store στον ενταμιευτή της συσκευής. Οι εντολές load και store αναφέρονται σε μία μόνο λέξη (4 Bytes) -- αφού δεν υπάρχουν εντολές load/store πολλαπλών λέξεων. Όταν ταξιδεύει η λέξη (μέσω της εντολής store) προς τη συσκευή μέσω του σύνδεσμου PCI-Express, η λέξη αυτή ταξιδεύει μέσα σε ένα πακέτο μεγέθους 12 Bytes (header) + 4 Bytes (data) = 16 Bytes ανά πακέτο. Με το ρυθμό 1 Byte κάθε 3.2 κύκλοι ρολογιού του επεξεργαστή που βρήκαμε παραπάνω, κάθε τέτοιο πακέτο των 16 Bytes (μεταφορά μίας λέξης των 4 Bytes) κοστίζει περίπου 50 κύκλους ρολογιού του επεξεργαστή. Ας υποθέσουμε ότι οι υπόλοιπες εντολές του βρόχου κοστίζουν 30 κύκλους του επεξεργαστή, κυρίως λόγω των αναπόφευκτων αστοχιών κρυφής μνήμης που θα προκαλέσουν οι επανειλημμένες εντολές load από διευθύνσεις μη πρόσφατα χρησιμοποιημένες. Συνολικά, επομένως, ο ρυθμός αντιγραφής είναι 4 Bytes ανά 80 κύκλους επεξεργαστή (80 ns). Πόσος είναι αυτός ο ρυθμός σε MBytes/s και σε Mbits/s; Εάν ο επεξεργαστής αυτός έχει να τροφοδοτεί ταυτόχρονα 2 δίσκους με παροχή 10 MBytes/s καθένας και 1 δίκτυο fast ethernet με παροχή 100 Mbits/s, τι ποσοστό του χρόνου του θα υποχρεωθεί να αφιερώνει για αντιγραφές δεδομένων από τη μνήμη του προς τους ενταμιευτές των συσκευών αυτών;

(β) Έστω τώρα ότι υπάρχει DMA. Ας υποθέσουμε, προς στιγμήν, ότι ο επεξεργαστής δεν απασχολεί καθόλου το σύνδεσμο PCI-Express μνήμης-E/E, π.χ. επειδή ευστοχεί συνεχώς στην κρυφή του μνήμη, και επομένως ο σύνδεσμος αυός είναι συνεχώς διαθέσιμος γιά τη μηχανή DMA. *(i)* Εάν η μηχανή DMA τροφοδοτούσε τις περιφερειακές συσκευές στέλνοντας μία λέξη κάθε φορά (4 data Bytes μέσα σε κάθε "πακέτο"), και εάν απασχολεί έτσι το σύνδεσμο πλήρως (100%), πόση θα ήταν η συνολική παροχή της σε MBytes/s και σε Mbits/s; *(ii)* Όμως, οι μηχανές DMA κάνουν (σχεδόν) πάντα τις μεταφορές τους σε μεγαλύτερα blocks δεδομένων, προκειμένου να αποσβέσουν σε περισσότερα δεδομένα το κόστος (overhead) της επικεφαλίδας του πακέτου, ή της απόκτησης και χρήσης της αρτηρίας, ή της έναρξης προσπέλασης σε νέα γραμμή (row) της μνήμης DRAM. Ας πούμε ότι εδώ η μηχανή DMA μας μεταφέρει τα δεδομένα σε ομάδες (blocks) των 16 λέξεων (64 Bytes) καθεμία (δηλ. ανά πακέτο)· άρα τα πακέτα θα έχουν μέγεθος 12 header Bytes + 64 data Bytes = 76 Bytes κάθε πακέτο. Με το ρυθμό 1 Byte κάθε 3.2 ns που βρήκαμε παραπάνω, κάθε τέτοιο πακέτο παίρνει 76 B x 3.2 ns/B = περίπου 250 ns ανά πακέτο, ή 4 M πακέτα (των 64 Bytes δεδομένων) ανά δευτερόλεπτο. Εάν η αρτηρία απασχολείται πάλι 100% για τέτοιες μεταφορές DMA, πόση θα είναι η συνολική παροχή της σε MBytes/s και σε Mbits/s;

(γ) Εάν οι παραπάνω μεταφορές DMA σε blocks των 64 Bytes εξυπηρετούν πάλι τους 2 δίσκους με παροχή 10 MBytes/s καθένας και το 1 δίκτυο fast ethernet με παροχή 100 Mbits/s της ερώτησης (α), τότε τι ποσοστό του χρόνου του συνδέσμου PCI-Express μνήμης-E/E απασχολούν αυτές οι μεταφορές DMA αυτών των περιφερειακών συσκευών; Συγκρίνετε αυτό το ποσοστό με το ποσοστό της απάντησης (α). Παρ'ότι πρόκειται για ανόμοια μεγέθη (το (α) ήταν ποσοστό του χρόνου του επεξεργαστή, ενώ το (γ) είναι ποσοστό του χρόνου της αρτηρίας), εξηγείστε σε ποιούς δύο παράγοντες οφείλεται η μείωση του ποσοστού απασχόλησης από το (α) στο (γ).

Επιπλέον της μείωσης αυτής, που είναι από μόνη της ένα κέρδος, παρατηρήστε ότι στο μεν (α), δηλαδή χωρίς DMA, ο επεξεργαστής αφιέρωνε ένα μη ευκαταφρόνητο μέρος του χρόνου του για να εξυπηρετεί τις μεταφορές δεδομένων αυτών των συσκευών E/E, ενώ στο (γ), δηλαδή με DMA, ο επεξεργαστής δεν αφιερώνει καθόλου χρόνο σε αυτές τις μεταφορές (φροντίζει μόνο να τις ξεκινάει, και μετά τις αφήνει να τρέχουν μόνες τους για πολλά KBytes συνήθως), και οι μεταφορές γίνονται "μόνες τους" (δηλαδή από την ή τις μηχανές DMA, που δουλεύουν παράλληλα με τον επεξεργαστή), απασχολώντας (οι μεταφορές DMA) ένα μέρος μόνο της διαθέσιμης παροχής (throughput) της μνήμης, και αφήνοντας το υπόλοιπο μέρος αυτής της παροχής διαθέσιμο για να εξυπηρετούνται οι αστοχίες της κρυφής μνήμης του επεξεργαστή.

Τρόπος Παράδοσης:
Κάντε τις από τώρα, γιά να τις έχετε έτοιμες, παρ' ότι θα τις παραδώσετε λίγο αργότερα, μαζί με την επόμενη (και τελευταία) σειρά ασκήσεων (και θα εξεταστείτε, τότε, προφορικά και στις δύο μαζί). Ο τρόπος παράδοσης θα είναι on-line, σε μορφή **PDF** (μόνον) (μπορεί να είναι κείμενο μηχανογραφημένο ή/και "σκαναρισμένο" χειρόγραφο, αλλά *μόνον* σε μορφή PDF).
