HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2020

Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Σειρά Ασκήσεων 10: Επίδοση Επεξεργαστών, CPI

κάντε τις έως Κυριακή 26 Απριλίου 2020 (βδ. 10.0) (από βδ. 8.2)

Βιβλίο: Διαβάστε την §1.4 (σελίδες 61-74) από το Ελληνικό βιβλίο. Εναλλακτικά, μπορείτε να διαβάστε την §1.6 (pp. 28-40) από το Αγγλικό βιβλίο (οι βασικές ιδέες εδώ είναι ίδιες γιά τον MIPS (Ελληνικό βιβλίο) και γιά τον RISC-V (Αγγλικό βιβλίο)). Σχετικές και χρήσιμες είναι και οι διαφάνειες 26-37 των συγγραφέων γιά το Κεφάλαιο 1, που βρίσκονται (PPT) στο: www.elsevier.com/__data/assets/powerpoint_doc/0010/273709/Chapter_01-RISC-V.ppt

Άσκηση 10.1: Επίδοση Επεξεργαστών

Η επίδοση (performance) ενός υπολογιστή, κατά την εκτέλεση δοθέντος προγράμματος, είναι αντίστροφα ανάλογη προς τον χρόνο εκτέλεσης αυτού του προγράμματος σε αυτόν τον υπολογιστή. Η επίδοση ενός υπολογιστή, γενικά και αόριστα, ανεξαρτήτως εκτελουμένου προγράμματος, δεν μπορεί να οριστεί επακριβώς και επιστημονικά, μπορεί δε να ποικίλλει ευρέως ανάλογα με τα χαρακτηριστικά των διαφόρων προγραμμάτων.

Εάν ο υπολογιστής *A* εκτελεί ένα δοθέν πρόγραμμα σε χρόνο *tA*, ο δε υπολογιστής *B* το εκτελεί σε χρόνο *tB*, όπου *tB > tA* και *(tB / tA) = 1.xy*, τότε λέμε ότι "ο υπολογιστής Α είναι ταχύτερος του Β κατά *xy* % γιά το δοθέν πρόγραμμα". Παραδείγματος χάριν, αν tA = 4s και tB = 5s, τότε (tB/tA) = 1.25, και ο Α είναι ταχύτερος του Β κατά 25 % γιά το δοθέν πρόγραμμα. Ο χρόνος t_{exec} εκτέλεσης ενός προγράμματος σ' έναν υπολογιστή μπορεί συχνά να εκφραστεί σαν:

$$t_{exec} = N_{instructions} * CPI_{average} * T_{clock}$$

όπου $N_{instructions}$ είναι το πλήθος (ο αριθμός) των εντολών που ο υπολογιστής εκτελεί προκειμένου να ολοκληρωθεί η εκτέλεση του δοθέντος προγράμματος, $CPI_{average}$ είναι το μέσο πλήθος (μέσος αριθμός) των κύκλων ρολογιού που απαιτούνται γιά την εκτέλεση μιάς εντολής (Cycles Per Instruction --CPI), και T_{clock} είναι ο χρόνος που διαρκεί ένας κύκλος ρολογιού, δηλαδή η περίοδος του ρολογιού, δηλαδή το αντίστροφο της συχνότητας ρολογιού.

Ερώτηση: Θεωρήστε έναν υπολογιστή Α (τύπου RISC), που γιά να τελειώσει ένα δοθέν πρόγραμμα πρέπει να εκτελέσει 2,500,000 εντολές, με μέσο CPI = 3.2 κύκλους ρολογιού ανά εντολή, και με ρολόϊ 1.25 GHz. Ενας άλλος υπολογιστής Β (τύπου CISC --complex instruction set computer) έχει πιό "πλούσιο" ρεπερτόριο εντολών, κι έτσι του αρκεί να εκτελέσει μόνο 1,800,000 εντολές γιά να τελειώσει το ίδιο πρόγραμμα. Ομως, λόγω της αυξημένης πολυπλοκότητάς του, έχει μέσο CPI = 4.0 κύκλους ρολογιού ανά εντολή, και ρολόϊ 1.0 GHz. Πόσους κύκλους ρολογιού και πόσα δευτερόλεπτα χρειάζεται ο κάθε υπολογιστής γιά να εκτελέσει το δοθέν πρόγραμμα; Ποιός από τους δύο υπολογιστές είναι ταχύτερος από τον άλλον γιά το δοθέν πρόγραμμα, και πόσο ταχύτερος;

Άσκηση 10.2: Μέσο CPI Επεξεργαστή Πολλαπλών Κύκλων/Εντολή

Ας πούμε ότι κάποιος υπολογιστής έχει δύο ειδών εντολές: *(i)* εντολές τύπου A, που καθεμιά τους χρειάζεται CPI_A κύκλους ρολογιού γιά να εκτελεστεί, και *(ii)* εντολές τύπου B, που καθεμιά τους χρειάζεται CPI_B κύκλους ρολογιού γιά να εκτελεστεί. Το πρόγραμμά μας, γιά να ολοκληρωθεί η εκτέλεσή του, χρειάζεται να εκτελεστούν: N_A το πλήθος εντολές τύπου A, και N_B το πλήθος εντολές τύπου B (Προφανώς, γιά ολόκληρο το πρόγραμμα: $N_{instructions} = N_A + N_B$). Τότε, ο συνολικός χρόνος εκτέλεσης του προγράμματος θα είναι:

$$t_{exec} = (N_A * CPI_A + N_B * CPI_B) * T_{clock} = N_{instructions} * CPI_{average} * T_{clock}$$

Το αριστερό μέρος της εξίσωσης αυτής προέρχεται από το πόσο χρόνο χρειάζονται γιά να εκτελεστούν όλες οι εντολές --και οι τύπου A και οι τύπου B. Το δεξιό μέρος της εξίσωσης είναι η "απλοποιητική" σχέση της παραπάνω §10.1, η οποία χρησιμεύει και σαν ορισμός του μέσου CPI. Από τη δεξιά ισότητα, λοιπόν, απλοποιώντας το T_{clock} και διαιρώντας διά $N_{instructions}$ προκύπτει ότι *το μέσο CPI είναι ο σταθμισμένος μέσος όρος των κύκλων ρολογιού ανά εντολή, όπου οι συντελεστές στάθμισης είναι τα ποσοστά (συχνότητα) εκτέλεσης του κάθε τύπου εντολών:*

$$CPI_{average} = (N_A / N_{instructions}) * CPI_A + (N_B / N_{instructions}) * CPI_B$$

Ας εξετάσουμε λοιπόν μια συγκεκριμένη περίπτωση: Στα προγράμματα ακεραίων (όχι κινητής υποδιαστολής) μεταξύ των SPEC benchmarks, όταν αυτά εκτελούνται σε επεξεργαστές RISC, η συχνότητα εκτέλεσης των διαφόρων εντολών είναι **περίπου** ως εξής:

- 26 % load,
- 11 % store,
- 40 % ALU (add, addi, sub, and, or, slt, slti, κλπ),
- 3 % load upper immediate (lui),
- 16 % conditional branches,
- 3 % unconditional jumps σε σταθερές διευθύνσεις (jal στον RISC-V),
- 1 % unconditional jump register (jalr στον RISC-V).

(α) Στην υλοποίηση του επεξεργαστή μας σε πολλαπλούς "σύντομους" κύκλους ρολογιού (αλλά **όχι** ακόμα pipelined) που είδαμε στην §8.10, έστω ότι οι κύκλοι ρολογιού που χρειάζονται γιά την εκτέλεση του κάθε τύπου εντολής είναι: πέντε (5) κύκλοι γιά κάθε εντολή load· τέσσερεις (4) κύκλοι γιά κάθε εντολή store ή ALU· και τρείς (3) κύκλοι γιά κάθε εντολή lui, branch, ή jump.

• Βάσει της διάρκειας αυτής εκτέλεσης του κάθε τύπου εντολής, και αν υποθέσουμε τα παραπάνω ποσοστά εκτέλεσης των διαφόρων τύπων εντολών, πόσο θα είναι το μέσο CPI αυτού του επεξεργαστή γιά αυτά τα προγράμματα;

(β) Έστω τώρα ότι κάνουμε βελτιστοποιήσεις: έστω ότι κάνουμε όλες τις εντολές load upper immediate (lui) και άλματος σε σταθερή διεύθυνση (jal στον RISC-V) να εκτελούνται σε δύο (2) αντί τριών κύκλων ρολογιού καθεμία. Πόσο θα είναι τότε το νέο μέσο CPI του επεξεργαστή γιά αυτά τα προγράμματα;

(γ) Αν η βελτιστοποίηση (β) έχει σαν αρνητική παρενέργεια να αυξήσει τον κύκλο ρολογιού από 0.80 ns σε 0.85 ns, ποιός από τους δύο επεξεργαστές (α) και (β) θα είναι ταχύτερος, και κατά πόσο; (Υπόδειξη: προφανώς, το πλήθος των εκτελούμενων εντολών $N_{instructions}$ δεν αλλάζει από τον (α) στον (β)) (στην

πραγματικότητα, οι συγκεκριμένες αυτές βελτιστοποιήσεις πιθανόν να μην έχουν αρνητικές επιπτώσεις στη συχνότητα ρολογιού, αλλά εδώ το κάνουμε λίγο "φτιαχτό" γιά να γίνει η άσκηση...).

Άσκηση 10.3: Μέσο CPI Επεξεργαστή με Ομοχειρία (Pipelining) χωρίς Πρόβλεψη Διακλαδώσεων

Στο κεφάλαιο (άσκηση) 9 περί ομοχειρίας (pipelining), είδαμε ότι, όσο ο επεξεργαστής βρίσκει **ανεξάρτητες** μεταξύ τους εντολές, τις εκτελεί με ρυθμό μία εντολή ανά κύκλο ρολογιού. Έτσι, παρά ο γεγονός ότι η εκτέλεση της κάθε εντολής διαρκεί περισσότερους του ενός κύκλους ρολογιού, το συνολικό πλήθος κύκλων ρολογιού γιά την εκτέλεση ενός ολόκληρου προγράμματος --που είναι και αυτό που μας ενδιαφέρει-- είναι περίπου τόσο όσες και οι εκτελούμενες εντολές (υπό τις παραπάνω προϋποθέσεις ανεξαρτησίας). Αρα, το μέσο CPI υπό τις συνθήκες αυτές θα είναι 1, δηλαδή η κάθε εντολή μας "κοστίζει" 1 κύκλο ρολογιού, όσο δηλαδή μας "καθυστερεί" μέχρι να ξεκινήσουμε και την επόμενή της.

Ομως, όπως είπαμε, δυστυχώς, υπάρχουν και αλληλεξαρτήσεις εντολών, οι οποίες προκαλούν απώλεια κύκλου ή κύκλων ρολογιού επιπλέον του παραπάνω ενός "βασικού" κύκλου ανά εντολή. Ας θεωρήσουμε, σε σχέση και με τα ποσοστά εντολών που αναφέρθηκαν στην παραπάνω άσκηση 10.2, ότι:

- Το 33 % των εκτελουμένων load (άρα το 33% του 26% ίσον 8.6% του συνόλου των εκτελουμένων εντολών) ακολουθείται αμέσως από εξαρτημένη εντολή (εντολή που χρειάζεται τα loaded data στην 3η βαθμίδα της pipeline της), και άρα προκαλούν (αυτές οι 33% των load ίσον 8.6% του συνόλου των εντολών) την απώλεια ενός κύκλου ρολογιού επιπλέον του ενός βασικού.
- Οι διακλαδώσεις υπό συνθήκη (branch) κοστίζουν 2 κύκλους η καθεμία: ένας ο βασικός, συν την ακύρωση μιάς από τις δύο εντολές που η pipeline έφερε (fetch) τους επόμενους δύο κύκλους (μιάς "από κάτω" και μιάς από τον προορισμό), μέχρι δηλαδή να διευκρινιστεί εάν η διακλάδωση επιτυγχάνει ή αποτυγχάνει.
- Τα άλματα σε σταθερή διεύθυνση προορισμού (jal στον RISC-V) κοστίζουν 2 κύκλους ρολογιού καθένα (ένας ο "δικός τους" συν ένας έξτρα χαμένος), οι δε εντολές jump-and-link-register (jalr) κοστίζουν 3 κύκλους ρολογιού καθεμία (ένας ο "δικός της" συν δύο έξτρα χαμένοι) (επειδή η διεύθυνση προορισμού των jal υπολογίζεται στη δεύτερη βαθμίδα, ενώ των jalr στην τρίτη).

Με αυτά τα δεδομένα, πόσο θα είναι το μέσο CPI (γιά τα παραπάνω προγράμματα της άσκησης 10.2) ενός επεξεργαστή RISC-V που χρησιμοποιεί αυτή τη μορφή ομοχειρίας; Θεωρώντας ότι αυτός ο επεξεργαστής έχει το ίδιο ρολόϊ με εκείνον της άσκησης 10.2(α), και αφού, φυσικά, εκτελεί τις ίδιες εντολές ανά πρόγραμμα, πόσο ταχύτερος θα είναι αυτός ο επεξεργαστής από εκείνον της άσκησης 10.2(α);

Άσκηση 10.4: Μέσο CPI Επεξεργαστή με Ομοχειρία και Πρόβλεψη Διακλαδώσεων

Στην βασική pipeline της άσκησης 10.3 έστω ότι προσθέτουμε έναν **Branch Target Buffer (BTB)** (Πρόβλεψη Διεύθυνσης Προορισμού Διακλαδώσεων), ο οποίος λειτουργεί στην πρώτη βαθμίδα, παράλληλα με τον αθροιστή PC+4, και ο οποίος μας λέει, στο τέλος της πρώτης βαθμίδας, εάν θεωρεί πιθανόν η εντολή που τώρα διαβάζουμε από τη διεύθυνση PC να είναι επιτυχημένη διακλάδωση ή άλμα, και

εάν έτσι θεωρεί τότε ποιά είναι η πιθανή διεύθυνση προορισμού της. Άρα, εάν μας πεί κάτι τέτοιο ο BTB, τότε τον επόμενο κύκλο, αντί να φέρουμε (fetch) την εντολή από τη διεύθυνση PC+4, φέρνουμε την εντολή από τη διεύθυνση που μας υπέδειξε σαν πιθανότερη ο BTB. Εάν η πρόβλεψη του BTB αποδειχτεί τελικά σωστή, τότε η εντολή διακλάδωσης ή άλματος δεν θα χάσει κανέναν κύκλο επιπλέον τους ενός βασικού της, αλλιώς θα χαθούν τόσοι κύκλοι όσες οι λάθος προβλεφθείσες άρα ακυρούμενες εντολές που φέραμε.

Έστω ότι ο BTB ποτέ δεν περιέχει διευθύνσεις εντολών που δεν είναι διακλαδώσεις ή άλματα, επομένως ποτέ δεν προξενεί καθυστερήσεις γιά τέτοιες "υπόλοιπες" εντολές πέραν των καθυστερήσεων των load που είδαμε στην άσκηση 10.3. Γιά τις διακλαδώσεις υπό συνθήκη και γιά τα άλματα όλων των ειδών (all branch and jump), έστω ότι στο 90% όλων αυτών ο BTB προβλέπει σωστά τη διεύθυνση της επόμενής τους εντολής (άρα δεν υπάρχει απώλεια επιπλέον κύκλου ρολογιού), ενώ στο υπόλοιπο 10% των διακλαδώσεων και αλμάτων ο BTB κάνει λάθος πρόβλεψη με συνέπεια σε αυτές τις περιπτώσεις να χάνονται *δύο (2) κύκλοι* ρολογιού επιπλέον του βασικού. Υπό αυτές τις συνθήκες, ξαναϋπολογίστε το μέσο CPI αυτής της pipeline, και βρείτε πόσο γρηγορότερος είναι αυτός ο επεξεργαστής από εκείνον της άσκησης 10.3.

Άσκηση 10.5: Μέσο CPI Επεξεργαστή Superscalar με Ομοχειρία

Οι σημερινοί (μικρο-) επεξεργαστές του εμπορίου χρησιμοποιούν τόσο την τεχνική της ομοχειρίας (pipelining) που είδαμε παραπάνω, όσο και τεχνικές εκτέλεσης πολλαπλών εντολών ταυτόχρονα --συνήθως με τη μορφή που αποκαλείται "superscalar". Θεωρήστε ένα τέτοιον επεξεργαστή που σε κάθε κύκλο ρολογιού διαβάζει (fetch) από τη μνήμη τις τέσσερεις (4) επόμενες εντολές, και εκτελεί ταυτόχρονα (εν παραλλήλω) **όσες** από αυτές είναι ανεξάρτητες μεταξύ τους. Το μέσο CPI ενός τέτοιου επεξεργαστή είναι όσο θα ήταν με χρήση ομοχειρίας μόνο (χωρίς superscalarity), διηρημένο διά το μέσο πλήθος ταυτόχρονα εκτελούμενων εντολών, δεδομένου ότι τώρα ο κάθε κύκλος ρολογιού που περνάει "χρεώνεται" σε όλες τις ταυτόχρονα εκτελούμενες εντολές, άρα στην κάθε εντολή χρεώνονται αντίστοιχα λιγότεροι κύκλοι ρολογιού (λιγοστεύουν οι cycles per instruction --CPI).

Θεωρήστε ότι κάνουμε τον επεξεργαστή της άσκησης 10.4 superscalar, και ας πούμε γιά απλότητα ότι αυτό γίνεται χωρίς να αλλάξουμε την δομή της pipeline του και χωρίς να αλλάξει το ρολόϊ (στην πράξη δεν είναι έτσι). Εάν το μέσο πλήθος ταυτόχρονα εκτελούμενων ανεξάρτητων εντολών είναι **1.6** εντολές, τότε ποιό θα είναι το μέσο CPI του νέου επεξεργαστή; Πόσο γρηγορότερος θα είναι αυτός από εκείνον της άσκησης 10.2(α), πόσο από εκείνον της άσκησης 10.3, και πόσο από εκείνον της άσκησης 10.4;

Τρόπος Παράδοσης:
Κάντε την απλή αυτή άσκηση από τώρα, γιά να την έχετε έτοιμη, παρ' ότι θα την παραδώσετε λίγο αργότερα, μαζί με επόμενη σειρά ασκήσεων. 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).
