HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2024

Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Άσκηση 9:
Εισαγωγή στην Ομοχειρία (Pipelining)

Από την 7η-8η για το τέλος της 9ης εβδομάδας του Εξαμήνου

Βιβλίο: Διαβάστε τις ενότητες 4.5 έως και 4.8, pp. 262-315 στο βιβλίο RISC-V (Αγγλικό, 2017) (ή σελίδες 389-448 στο βιβλίο MIPS (Ελληνικό, 2010)). Εάν δεν έχετε το Αγγλικό βιβλίο, καλή προσέγγιση σε αυτό αποτελούν οι διαφάνειες των συγγραφέων, που γιά το κεφ. 4 βρίσκονται (PPT) στο: www.elsevier.com/__data/assets/powerpoint_doc/0004/273712/Chapter_04-RISC-V.ppt

9.1 Εισαγωγικά, Παροχή, Καθυστέρηση, Παραδείγματα

Μελετήστε τις διαφάνειες 21a/09a_pipeIntro.pdf . Βοηθητικά, μπορείτε να διαβάστε τις σελίδες 389-394 του βιβλίου MIPS (Ελληνικό, 2010), ή τις σελίδες 262-267 του βιβλίου RISC-V (Αγγλικό, 2017). Όπως είπαμε, η ομοχειρία είναι μορφή αξιοποίησης *παραλληλισμού*, που ταιριάζει ιδιαίτερα όταν οι μονάδες υλικού από τις οποίες πρέπει να περάσουν οι εργασίες είναι ετερογενείς (διαφορετικές μεταξύ τους) –εξειδικευμένες σε κάτι διαφορετικό η καθεμία. Κλασσικό παράδειγμα pipeline είναι οι βιομηχανικές γραμμές συναρμολόγησης· τον Ελληνικό όρο *Ομοχειρία* τον δανειζόμαστε από περιπτώσεις ανθρώπινων αλυσίδων π.χ. στο ναυτικό/στρατό που κουβαλούν αντικείμενα περνώντας τα από χέρι σε χέρι. Η ομοχειρία δεν μειώνει το χρόνο εκτέλεσης της καθεμιάς μεμονωμένης εντολής, αλλά αρχίζει την εκτέλεση της επόμενης εντολής πριν τελειώσει η εκτέλεση της προηγούμενης, με αποτέλεσμα να αυξάνει κατά πολύ την *παροχή* (throughput), δηλαδή το πλήθος των εκτελούμενων εντολών ανά μονάδα χρόνου.

9.2 Η Βασική Ιδέα του Pipelined Datapath:

Μελετήστε τη διαφάνεια 11 από τις παραπάνω 21a/09a_pipeIntro.pdf . Βοηθητικά, μπορείτε να διαβάστε τις σελίδες 276-280 του βιβλίου RISC-V (Αγγλικό, 2017) (ή τις σελίδες 405 - 409 του βιβλίου MIPS (Ελληνικό, 2010)). Σε σχέση με την απλή υλοποίηση του ενός (μακρύ) κύκλου ρολογιού ανά εντολή, εδώ (α) έχουμε ένα ρολόϊ περίπου 5 φορές γρηγορότερο (δηλ. ο κάθε κύκλος ρολογιού διαρκεί περίπου 5 φορές συντομότερα), (β) έχουμε "κόψει" τις εργασίες της κάθε εντολής σε 5 κομάτια (*βαθμίδες - stages*), περίπου ίσης διάρκειας το καθένα, όπου το καθένα χωρά σε έναν κύκλο του νέου (γρηγορότερου) ρολογιού, και (γ) εισάγουμε (ακμοπυροδότητους) καταχωρητές γιά όλα τα δεδομένα και σήματα που "περνούν" από τη μία βαθμίδα στην επόμενη. Οι καταχωρητές κρατάνε τις εισόδους της κάθε βαθμίδας σταθερές γιά να μπορεί αυτή να λειτουργήσει, ενώ "απελευθερώνουν" από αυτή την υποχρέωση την προηγούμενη βαθμίδα,

ούτως ώστε αυτή να μπορέσει να εργαστεί γιά την επόμενη εντολή. Η κάθε εντολή, στον κάθε κύκλο ρολογιού, βρίσκεται σε μία από τις βαθμίδες, και διαδοχικές εντολές βρίσκονται σε συνεχόμενες βαθμίδες. Σε κάθε ακμή ρολογιού, όλες οι εντολές προχωρούν μία βαθμίδα δεξιά. Θέλουμε όλες μαζί να προχωρούν ταυτόχρονα προς τα δεξιά, "τακτικά και στη σειρά" (*in-order* pipeline) γιά λόγους απλότητας· συνέπεια αυτού είναι ότι όσες εντολές δεν έχουν τίποτα να κάνουν στη βαθμίδα "Data Memory" (4η βαθμίδα) απλώς περνάνε από εκεί περιμένοντας έναν κύκλο μέχρι να έλθει η σειρά τους να γράψουν στο αρχείο καταχωρητών στον 5ο κύκλο και αυτές (αντί στον 4ο), επειδή οι εντολές load είναι αναγκασμένες να γράφουν τον rd στον 5ο κύκλο.

9.3 Λεπτομερής Λειτουργία του Pipelined Datapath χωρίς Αλληλεξαρτήσεις ή Διακλαδώσεις

Μελετήστε τις διαφάνειες 1-11 στο 20a/ex09.3-5_slides.pdf . Βοηθητικά, μπορείτε να διαβάστε από το βιβλίο RISC-V (Αγγλικό, 2017) τις σελίδες 281-286 (ή από το βιβλίο MIPS (Ελληνικό, 2010) τις σελίδες 409-417).

9.4 Μονάδα Ελέγχου γιά την Ομοχειρία

Μελετήστε τις διαφάνειες 12-19 στο 20a/ex09.3-5_slides.pdf . Βοηθητικά, μπορείτε να διαβάστε από το βιβλίο RISC-V (Αγγλικό, 2017) τις σελίδες 290-294 (ή από το βιβλίο MIPS (Ελληνικό, 2010) τις σελίδες 420-425). Εδώ, η βασική ιδέα είναι ότι τα σήματα ελέγχου "ταξιδεύουν" προς τα δεξιά μαζί με τα δεδομένα (και τις διευθύνσεις) της εντολής. Όπως συνήθως έτσι και εδώ, το opcode και τα function codes της εντολής αποκωδικοποιούνται στη δεύτερη βαθμίδα της pipeline, και στη συνέχεια τα "ξεχνάμε" αυτά και κρατάμε μόνον την αποκωδικοποιημένη πληροφορία, δηλαδή τα σήματα ελέγχου γιά τις μονάδες hardware που θα μας χρειαστούν, και αυτά "ταξιδεύουν" προς τα δεξιά μέχρι να φτάσουν στη βαθμίδα της pipeline γιά την οποία προορίζονται.

9.5 Γραφική Αναπαράσταση της Ομοχειρίας

Μελετήστε τις διαφάνειες 20-24 στο 20a/ex09.3-5_slides.pdf . Βοηθητικά, μπορείτε να διαβάστε από το βιβλίο RISC-V (Αγγλικό, 2017) τις σελίδες 286-290 (ή από το βιβλίο MIPS (Ελληνικό, 2010) τις σελίδες 417-420).

9.6 Εξαρτήσεις από Εντολή ALU: Προσπέρασμα

Μελετήστε τις διαφάνειες 1-12 στο 23a/09c_depend.pdf . Βοηθητικά, μπορείτε να διαβάστε από το βιβλίο RISC-V (Αγγλικό, 2017) τα εισαγωγικά στις σελίδες 268-269, και στη συνέχεια την λεπτομερή παρουσίαση στις σελίδες 294-303 (ή από το βιβλίο MIPS (Ελληνικό, 2010) τις σελίδες 395-397 και 425-434). Στις τελευταίες 5 σελίδες, λάβετε υπ' όψη σας ότι εμείς στις διαλέξεις βάλαμε το κύκλωμα ελέγχου / ανίχνευσης γιά την ύπαρξη αλληλεξαρτήσεων και την απόφαση προσπεράσματος στη *δεύτερη* βαθμίδα της pipeline, όπως κάνουν οι κανονικοί επεξεργαστές –και όχι στην τρίτη όπως κάνει το βιβλίο γιά απλότητα. Ο λόγος να βάλει κανείς το κύκλωμα αυτό στη δεύτερη βαθμίδα είναι ότι εκεί υπάρχει "ελεύθερος" χρόνος να γίνει αυτή η δουλειά εν

παραλλήλω με την ανάγνωση καταχωρητών (δηλαδή χωρίς να χάνεται επιπλέον χρόνος γιά αυτή τη δουλειά), ενώ εάν αυτό τοποθετηθεί στην τρίτη βαθμίδα, τότε στην αρχή του κύκλου ρολογιού, στην τρίτη βαθμίδα, το datapath δεν κάνει τίποτα χρήσιμο περιμένοντας να αποφασίσει το κύκλωμα εάν πρέπει ή δεν πρέπει να γίνει προσπέρασμα, και από πού.

9.7 Εξάρτηση από προηγούμενη Εντολή Load: Αναμονή

Μελετήστε τις διαφάνειες 13-18 στο 23a/09c_depend.pdf . Βοηθητικά, μπορείτε να διαβάστε από το βιβλίο RISC-V (Αγγλικό, 2017) τα εισαγωγικά στις σελίδες 270-271, και στη συνέχεια την λεπτομερή παρουσίαση στις σελίδες 303-307 (ή από το βιβλίο MIPS (Ελληνικό, 2010) τις σελίδες 397-399 και 434-438).

Εάν θέλετε να εξασκηθείτε στη λειτουργία της Pipeline π.χ. χρωματίζοντας το διάγραμμα που χρησιμοποιήσαμε στις διαφάνειες του μαθήματος, μπορείτε να το βρείτε, κενό χρωμάτων, στο φύλλο εργασίας: Pipeline & forwarding Worksheet (PDF)

9.8 Διακλαδώσεις & Άλματα: Υπόθεση Αποτυχίας, Ακύρωση Εντολών εάν επιτύχει, Πρόβλεψη Διακλαδώσεων

Μελετήστε τις διαφάνειες στο 22a/09d_ctrlDep.pdf . Βοηθητικά, μπορείτε να διαβάστε από το βιβλίο RISC-V (Αγγλικό, 2017) τα εισαγωγικά στις σελίδες 271-274, και στη συνέχεια την λεπτομερή παρουσίαση στις σελίδες 307-315 (ή από το βιβλίο MIPS (Ελληνικό, 2010) τις σελίδες 399-405 και 438-448).

Άσκηση 9.9: Ripes: Οπτικοποίηση του Pipelining

Στην άσκηση αυτή θα χρησιμοποιήστε τον προσομοιωτή **Ripes** (γραμμένον από τον Morten Borup Petersen, τότε φοιτητή και νυν απόφοιτο του DTU (Technical University of Denmark)) γιά την οπτικοποίηση της λειτουργίας της απλής pipeline του RISC-V που είδαμε στο μάθημα. Διαβάστε τη γενική περιγραφή και τις Οδηγίες Χρήσης του *Ripes* από την ιστοσελίδα: github.com/mortbopet/Ripes/ . Μπορείτε είτε να χρησιμοποιήσετε την έκδοση online του προσομοιωτή, είτε να τον "κατεβάστε" και τρέξετε τοπικά στο μηχάνημα σας (χρειάζεται να έχετε εγκαταστήσει JAVA για να το τρέξετε τοπικά):

- **Online**: ripes.me/
- **Linux**: github.com/mortbopet/Ripes/releases/download/v2.2.6/Ripes-v2.2.6-linux-x86_64.AppImage (γιά να αλλάξετε τα permissions του αρχείου που κατεβάσατε σε εκτελέσιμο, τρέξτε στο φάκελλο όπου το κατεβάσατε: `(sudo) chmod +x Ripes-continuous-linux-x86_64.AppImage`)
- **Windows**: github.com/mortbopet/Ripes/releases/download/v2.2.6/Ripes-v2.2.6-win-x86_64.zip
- **MacOS**: github.com/mortbopet/Ripes/releases/download/v2.1.0/Ripes-v2.1.0-mac-x86_64.zip
- Αν θέλετε να κάνετε compile τον πηγαίο κώδικα (δυσκολότερο, και μη αναγκαίο): github.com/mortbopet/Ripes/archive/refs/tags/v2.2.6.zip

Χρήση του Ripes:

Αφού τρέξετε το προσομοιωτή, το πρώτο παράθυρο (tab) που σας εμφανίζεται είναι ο editor. Υπάρχουν 5 διαθέσιμα tabs, και φαίνονται πάνω αριστερά, κάτω από το *File*. Τα 5 tabs είναι ο *Editor*, ο *Processor*, η *Cache* (κρυφή μνήμη), η *Memory*, και το *I/O* (input/output, σε περίπτωση που θελήσετε να παίξετε με memory-mapped I/O). Στο Editor tab μπορείτε να γράψετε ένα πρόγραμμα σε Assembly (ή Binary) και να το τρέξετε. Στο Processor tab θα δείτε μια γραφική αναπαράσταση της Pipeline όπως εκείνη του μαθήματος (το default είναι η κλασσική pipeline 5 βαθμίδων, με forwarding και με hazard detection· υπάρχουν και άλλες επιλογές εάν θέλετε να τις ψάξετε, περιλαμβανόμενης και εκείνης του ενός μακρύ κύκλου των Ασκήσεων 8). Εάν θέλετε να δείτε την pipeline με όλα τα σήματα ελέγχου, μπορείτε να επιλεξετε το extended layout. Αρχικά πρέπει να δώσετε ένα πρόγραμμα για να τρέξει ο προσομοιωτής.

1. Ανοίξετε ένα από τα έτοιμα παραδείγματα μέσω του File→Load Example→Assembly→factorial.s (Υπολογίζει το 7! στον καταχωρητή a0). Για να δώσετε ένα δικό σας αρχείο Assembly: File→Load Assembly File ή File→Load Binary File.
2. Τρέξτε το πρόγραμμα: αφού ανοίξετε το Processor Tab (πάνω αριστερά), πατήστε το *Run (F6)*. Το πρόγραμμα θα τρέξει μέχρι τέλους και θα μπορείτε να δείτε την ροή των δεδομένων στη γραφική απεικόνιση της Pipeline. Μπορείτε να αλλάξετε την ταχύτητα που τρέχουν οι εντολές μέσω του πλαισίου επάνω, στη γραμμή εργαλείων. Μπορείτε να σταματήσετε την εκτελεση όπου θέλετε πατώντας το ίδιο κουμπί. Το κουμπί *Step (F5)* τρέχει ένα ένα τα στάδια της κάθε εντολής. Το κουμπί *F4* πηγαίνει την εκτέλεση ένα στάδιο πίσω. Το *Reset (F3)* καθαρίζει τους καταχωρητές και την pipeline και επιστρέφει στην πρώτη εντολή.
3. Αν πάτε στο *Cache* tab, θα δείτε μια απεικόνιση της κρυφής μνήμης L1 (ξεχωριστα για data και instructions), και καποια στατιστικά επίδοσης για αυτήν ανάλογα με το πρόγραμμα που θα τρέξετε. Οι κρυφές μνήμες καλύπτονται σε επόμενα μαθήματα, οπότε μπορείτε να το αγνοήσετε για τώρα.
4. Αν πάτε στο *Memory* tab, θα δείτε, εκτός απο τους καταχωρητές, όλες τις εντολές που κάνουν κάποιο memory access, και την ίδια τη μνήμη. Αν θέλετε να δείτε κάτι στην μνήμη, και ξέρετε τη διεύθυνσή του, πατήστε κάτω δεξιά Go to → Address και γράψτε την διευθυνση. Επιπλέον, μπορείτε να δείτε τα .data και .text.
5. Στο *IO* tab μπορείτε να "εγκαταστήσετε" κάποια επιπλέον περιφερειακά modules με τα οποία μπορείτε να αλληλεπιδράσετε γράφοντας ή διαβάζοντας σε συγκεκριμένες διευθύνσεις στη μνήμη (Memory-Mapped IO). Για να παίξει το παράδειγμα `leds.s` του RIPES πρέπει να έχετε ανοίξει το LED Matrix, οπότε καθώς τρέχει το πρόγραμμα θα βλέπετε να αλλάζουν τα leds ένα ένα.

Οι "ρυθμίσεις" εμφανίζονται πατώντας το κουμπί με την εικόνα ενός CPU chip. Στις ρυθμίσεις μπορεί ο χρήστης να επιλέξει το extended layout (λεπτομερής απεικόνιση), ώστε να φαίνονται περισσότερα σήματα καθώς και τα Forwarding unit και Hazard Detection. Περισσότερα για τον Ripes θα βρείτε στην ιστοσελίδα του, που αναφέραμε στην αρχή της άσκησης (έχει και τη δυνατότητα να κάνει compile και να εκτελέσει κώδικα C· δεν θα το χρειαστείτε, αλλά εάν σας

ενδιαφέρει δείτε το: github.com/mortbopet/Ripes/wiki/Building-and-Executing-C-programs-with-Ripes).

Εξοικειωθείτε με τον Ripes και με την pipeline του μαθήματος, δεδομένου και ότι θα εξεταστείτε προφορικά σε αυτά. Στη συνέχεια, απαντήστε γραπτά στα παρακάτω, και τρέξτε τις εξής προσομοιώσεις που ζητούνται:

(1) Απαριθμήστε τις βαθμίδες (στάδια) της pipeline του επεξεργαστή, και για την κάθε μία περιγράψτε περιληπτικά (1 έως 3 προτάσεις) τη λειτουργία της.

(2) Ποιές βαθμίδες είναι απαραίτητες για ολες τις εντολές; Δηλαδή, ποιά στάδια της pipeline εκτελούν κάποια χρήσιμη λειτουργία για όλες τις εντολές, ανεξαρτήτως τύπου εντολής;

(3) Ποιά βαθμίδα της pipeline δεν κάνει κάτι χρήσιμο γιά τις εντολές αριθμητικών πράξεων; Ομοίως, ποιά βαθμίδα δεν κάνει κάτι χρήσιμο γιά τις εντολές store?

(4) Γράψτε μερικές δικές σας αλληλουχίες εντολών χωρίς αλληλεξαρτήσεις και παρατηρήστε την εκτελεση τους στον Ripes. Συγκεκριμένα, γράψτε (α) μερικές εντολές αριθμητικών πράξεων μεταξύ καταχωρητών, (β) μερικές μεταξύ καταχωρητών και σταθερών, και (γ) μερικές εντολές load και store. Για κάθε μία απο αυτές τις τρεις περιπτώσεις, γράψτε στην αναφορά σας τις εντολές που εκτελέσατε (τα περιεχόμενα του αρχείου "Instructions.s") συνοδευόμενα απο ένα screenshot απο την προσομοίωση.

(5) Αλλάξτε την αλληλουχία σας εντολών 4(α) ούτως ώστε να τις κάνετε **εξαρτημένες** μεταξύ τους· κάντε τη δεύτερη να εξαρτάται από την πρώτη, και την τρίτη να εξαρτάται και από τις δύο προηγούμενες –από την πρώτη γιά τον πρώτο τελεστέο πηγής της, και από τη δεύτερη γιά τον δεύτερο. Εκτελέστε τις στον Ripes, και παρακολουθήστε προσεκτικά και κατανοήστε τα προσπεράσματα (bypasses - internal forwardings). Γράψτε στην αναφορά σας τις εντολές και δώστε ένα screenshot τη στιγμή των προσπερασμάτων γιά την τρίτη εντολή.

(6) Δεσμεύστε χώρο στη μνήμη, σε πέντε συνεχόμενες λέξεις, γιά πέντε ακέραιες μεταβλητές, τις `int a, b, c, e, f;` και αρχικοποιήστε τις με κάποιες τιμές. Επίσης αρχικοποιήστε τον καταχωρητή s0 να περιέχει τη διεύθυνση της πρώτης, οπότε μπορείτε να διαβάζετε ή γράφετε καθεμιά τους με μία εντολή load ή store με ένα μικρό offset σε σχέση με τον s0. Στη συνέχεια μεταφράστε σε Assembly, χωρίς καμία αναδιάταξη εντολών, και προσομοιώστε στον Ripes τα εξής δύο statements σε C, με τη σειρά που δίδονται:

```
f = a + b;   e = a - c;
```

Παρατηρήστε την καθυστέρηση στην pipeline, λόγω εξάρτησης, από το δεύτερο load στο πρώτο add, καθώς και από το τρίτο load στη δεύτερη πράξη. Στη συνέχεια, **αναδιατάξτε** τις εντολές (*instruction scheduling*) ούτως ώστε να γλυτώσετε αυτές τις καθυστερήσεις, όπως στην τελευταία διαφάνεια της §9.7: διαβάστε *και* το c από τη μνήμη πριν κάνετε την πρώτη πρόσθεση. Ξανατρέξτε τις αναδιαταγμένες εντολές στον Ripes, και παρατηρήστε τώρα το χρόνο

εκτέλεσης. Γράψτε στην αναφορά σας τις εντολές πριν και μετά την αναδιάταξη, και δώστε ένα screenshot από την πρώτη περίπτωση (πριν την αναδιάταξη) τη στιγμή της αναμονής της πρώτης add, καθώς και ένα screenshot από τη δεύτερη περίπτωση (μετά την αναδιάταξη) τη στιγμή του προσπεράσματος της μεταβλητής b προς την ALU γιά την πρώτη add.

(7) Τρέξτε το `factorial.s` από τα παραδείγματα που έχει έτοιμα ο Ripes. Όταν τελειώσει η εκτέλεση, θα δείτε στο execution info ότι εκτελέστηκαν 125 εντολές σε 189 κύκλους ρολογιού. Πού οφείλονται οι περισσότεροι από τους "χαμένους" κύκλους ρολογιού (εκτός από κάπου 6 κύκλους για το ecall στο τέλος); Για να μην χάνονται τόσο πολλοί κύκλοι ρολογιού σε προγράμματα σαν αυτό, είδαμε μια τεχνική στο μάθημα (πιο απλή από τον Branch Target Buffer (BTB), πριν δούμε τον BTB). Ποια είναι η κεντρική ιδέα αυτής της τεχνικής, και *χοντρικά* τι αποτελέσματα θα είχε αυτή (*περίπου* πόσους από τους χαμένους κύκλους θα γλυτώναμε) εάν την εφαρμόζαμε εδώ;

Τρόπος Παράδοσης:
Παραδώστε μαζί την προηγούμενη άσκηση 8 και αυτήν εδώ την άσκηση on-line, σε μορφή **PDF** (μόνον) (μπορεί να είναι κείμενο μηχανογραφημένο ή/και "σκαναρισμένο" χειρόγραφο, αλλά *μόνον* σε μορφή PDF). Παραδώστε μέσω **`turnin ex089@hy225 [directoryName]`** ένα αρχείο ονόματι **`ex089.pdf`** που θα περιέχει τις απαντήσεις σας σε όλες τις ασκήσεις, 8 και 9.
Θα εξεταστείτε **και προφορικά** για τις ασκήσεις 8 και 9 (μαζί), από βοηθό του μαθήματος, με διαδικασία γιά την οποία θα ενημερωθείτε μέσω ηλτά (email) στη λίστα του μαθήματος.
