ΗΥ-225: Οργάνωση Υπολογιστών
Άνοιξη 2017

Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Άσκηση 8: Μία Απλή Υλοποίηση του MIPS σε Έναν Κύκλο Ρολογιού ανά Εντολή

Προθεσμία έως Τετάρτη 22 Μαρτίου 2017, ώρα 23:59 (βδ. 7.2)
Υπενθύμιση Διαγωνισμού Προόδου: Σάββατο 18 Μαρτίου 2017, ώρα 12-2

Βιβλίο (4η έκδοση): Διαβάστε την §2.4, σελίδες 127 - 134 (και 169-170), και την §4.4, σελίδες 374 - 388· επιπροσθέτως, οι ενότητες 4.1 - 4.3 (σελ. 356-374) περιέχουν εισαγωγικό υλικό, χρήσιμο γιά την αρχική κατανόηση, αλλά υπερκαλύπτονται, όσον αφορά την τελική εκμάθηση, από την §4.4.

8.1 Γενική Ιδέα της Απλής Υλοποίησης του MIPS:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7027, και επικουρικά, εάν σας βοηθά, μπορείτε να διαβάσετε και από το βιβλίο την §4.1 (σελ. 356-361) (που όμως υπερκαλύπτεται, όσον αφορά την τελική εκμάθηση, από την §4.4).

8.2 Πολύπορτο Αρχείο Καταχωρητών:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7028. Μιά περίληψη του θέματος έχει ως εξής: Το Αρχείο Καταχωρητών (Register File - RF) του MIPS αποτελείται από 32 καταχωρητές, καθένας των 32 bits (ο καταχωρητής υπ' αριθμόν μηδέν είναι ειδικός --δεν χρειάζεται flip-flops, υλοποιείται σαν 32 σύρματα γείωσης). Γιά να πετύχουμε την απλότητα χρονισμού που χρειάζεται αυτή η υλοποίηση του ενός κύκλου ρολογιού, ας υποθέσουμε ότι οι καταχωρητές αυτοί είναι ακμοπυροδότητοι.

Γιά να διαβάσουμε (επιλέξουμε) έναν από τους 32 αυτούς καταχωρητές, όπως σε κάθε μνήμη, χρειαζόμαστε έναν τριανταδυάμπιτο πολυπλέκτη 32-σε-1, ο οποίος φυσικά χρειάζεται 5 εισόδους (5 bits) επιλογής γιά να του λένε ποιόν από τους 32 καταχωρητές θέλουμε να διαβάζουμε (επιλέγουμε) κάθε στιγμή· αυτά τα 5 bits είναι η διεύθυνση ανάγνωσης. Εάν τώρα προσθέσουμε και έναν *δεύτερο* πολυπλέκτη 32-σε-1 (τριαντδυάμπιτο και αυτόν), και τον ελέγξουμε με 5 άλλα bits "διεύθυνσης" (που έχουν εν γένει διαφορετική τιμή από τα πρώτα 5, χωρίς και να απαγορεύεται μερικές φορές να παίρνουν και την ίδια τιμή), τότε αυτός ο δεύτερος πολυπλέκτης θα μας διαβάζει (επιλέγει) έναν άλλον, δεύτερο καταχωρητή από τους 32 (που είναι εν γένει άλλος από τον πρώτον, χωρίς όμως και να αποκλείεται μερικές φορές να είναι ο ίδιος, εάν έτσι δοθούν οι δύο διευθύνσεις ανάγνωσης). Έτσι φτιάξαμε ένα αρχείο καταχωρητών *Δίπορτης Ανάγνωσης* (2-port read), δηλαδή, ανά πάσα στιγμή, μπορούμε να επιλέγουμε και βλέπουμε, στις δύο (τριανταδυάμπιτες) εξόδους δεδομένων, τα περιεχόμενα δύο οιονδήποτε από τους 32 καταχωρητές (δαιφορετικών ή ίδιων, μεταξύ τους). Από άποψη καθυστέρησης, οι δύο πολυπλέκτες δουλεύουν εν παραλλήλω --ταυτόχρονα δηλαδή-- και άρα μέσα στο χρόνο μιάς και μόνο ανάγνωση από μονόπορτο RF, εδώ επιτυγχάνουμε *δύο* αναγνώσεις αντί μίας.

Ταυτόχρονα με τα παραπάνω, εάν υπάρχει και ένας τρίτος αποκωδικοποιητής 5-σε-32 (τρίτο τον λέμε επειδή οι δύο πολυπλέκτες ανάγνωσης κρύβουν μέσα τους και από έναν αποκωδικοποιητή 5-σε-32, καθένας), και εάν αυτόν τον τρίτο αποκωδικοποιητή τον χρησιμοποιήσουμε γιά να ελέγξουμε τα 32 σήματα ενεργοποίησης φόρτωσης (load enable) των 32 καταχωρητών, τότε ταυτόχρονα με την ανάγνωση των δύο καταχωρητών, θα μπορούμε να έχουμε και εγγραφή σε έναν τρίτο (διαφορετικό ή και τον ίδιο με τους δύο πρώτους). Γιά να λειτουργήσει καθώς πρέπει η εγγραφή, και να μην εγγράφουμε σε κάποιον καταχωρητή *κάθε* κύκλο ρολογιού, καταστρέφοντας έτσι τα παλαιά περιεχόμενά του ακόμα κι αν δεν έχουμε τίποτα χρήσιμο να γράψουμε, χρειαζόμαστε κι ένα συνολικό σήμα ελέγχου φόρτωσης, που όταν είναι μηδέν αδρανοποιεί την εγγραφή γενικά, σε όλους τους καταχωρητές. Θα υποθέτουμε στις υλοποιήσεις μας ότι το Αρχείο Καταχωρητών του MIPS θα είναι έτσι

"τρίπορτο" (three-port), με δύο πόρτες ανάγνωσης και μία πόρτα εγγραφής. Επομένως θα έχει τρείς (πεντάμπιτες) εισόδους διευθύνσεων --δύο ανάγνωσης και μία εγγραφής-- καθώς και τρείς τριανταδυάμπιτες πόρτες δεδομένων: δύο με δεδομένα εξόδου, γιά την ανάγνωση, και μία με δεδομένα εισόδου, γιά την εγγραφή. Επιπλέον, έχει και ένα bit ελέγχου επίτρεψης φόρτωσης ("load enable", ή "write enable").

8.3 Προσημασμένοι Αριθμοί, Επέκταση Προσήμου:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7029, και διαβάστε τα αντίστοιχα από το βιβλίο στην §2.4 (σελ. 127 - 134). Εάν χρησιμοποιήσουμε τον ορισμό της αναπαράστασης αρνητικών αριθμών σε κωδικοποίηση συμπληρώματος ως-προς-2 όπως τον είχαμε δεί εμείς στο μάθημα της Ψηφιακής Σχεδίασης (HY-120, ενότητα 6.3), τότε η μετατροπή προσημασμένου (signed) αριθμού από λιγότερα σε περισσότερα bits με την τεχνική της επέκτασης προσήμου (sign extension) αποδεικνύεται μαθηματικά ως εξής:

΄Εστω ο προσημασμένος ακέραιος $A_{s,k}$ με *k bits*, τον οποίο θέλουμε να μετατρέψουμε στον ίδιο αριθμό $A_{s,n}$ με *n bits*: $A_{s,n} = A_{s,k}$ όπου n>k. Εάν τα bits του $A_{s,k}$ τα ερμηνεύσουμε σαν μη προσημασμένο (unsigned) ακέραιο, τότε θα μοιάζουν με (θα δηλώνουν) έναν αριθμό που ας το ονομάσουμε $A_{u,k}$ και ομοίως εάν τα bits του $A_{s,n}$ τα ερμηνεύσουμε σαν unsigned τότε θα μοιάζουν με τον $A_{u,n}$. Εάν ο $A_{s,k} = A_{s,n}$ είναι μη αρνητικός (δηλ. θετικός ή μηδέν), τότε, κατά τον ορισμό της κωδικοποίησης συμπληρώματος ως-προς-2, (α) το αριστερό bit τους θα είναι μηδέν, και (β) οι απρόσημες ερμηνίες τους θα είναι: $A_{u,k} = A_{s,k}$ και $A_{u,n} = A_{s,n}$, και αφού $A_{s,n} = A_{s,k}$ τότε θα είναι και: $A_{u,n} = A_{u,k}$. Αυτοί οι δύο τελευταίοι, αφού είνα unsigned ακέραιοι, με n και k bits αντίστοιχα, και είναι ίσοι μεταξύ τους, προκύπτει ότι ο $A_{u,n}$ που έχει περισσότερα bits θα είναι ίδιος με τον $A_{u,k}$ αλλά με n-k μηδενικά προστεθημένα αριστερά από τον $A_{u,k}$.

Αλλοιώς, εάν οι $A_{s,n} = A_{s,k}$ είναι αρνητικοί, τότε, κατά τον ορισμό μας, (α) το αριστερό bit τους θα είναι ένα, και (β) οι απρόσημες ερμηνίες τους θα είναι: $A_{u,k} = A_{s,k} + 2^k$ και $A_{u,n} = A_{s,n} + 2^n$. Δεδομένου ότι: $A_{s,n} = A_{s,k}$ προκύπτει ότι θα είναι και: $A_{u,n} - 2^n = A_{u,k} - 2^k$. Επομένως, η αναπαράσταση που ψάχνουμε είναι η: $A_{u,n} = A_{u,k} - 2^k + 2^n = A_{u,k} + (2^n - 2^k) = (2^{n-k} - 1) \cdot 2^k + A_{u,k}$. Σε αυτήν την τελευταία έκφραση, ο μεν αριστερός προσθετέος, $(2^{n-k} - 1) \cdot 2^k$, αποτελείται από (n-k) το πλήθος άσσους (που είναι ο αριθμός $2^{n-k} - 1$) ολισθημένους αριστερά κατά k θέσεις bits (που είναι ο πολλαπλασιασμός επί 2^k), ο δε δεξιός προσθετέος, $A_{u,k}$, είναι τα αρχικά k bits του αρχικού αριθμού που μας δόθηκε. Δεδομένου ότι ο πρώτος προσθετέος έχει όλο μηδενικά στις k δεξιές θέσεις, ο δε δεύτερος προσθετέος έχει όλο μηδενικά στις (n-k) αριστερές θέσεις, το άθροισμά τους θα είναι προφανώς απλώς η "συγκόλληση" (concatenation) των δύο αυτών ποσοτήτων. Επομένως, η αναπαράσταση με n bits του αρχικού αριθμού που μας δόθηκε θα αποτελείται από τα αρχικά k bits, δεξιά, μαζί με (n-k) άσσους κολλημένους ακριβώς αριστερά τους.

Συνολικά λοιπόν, γιά να μετατρέψουμε ένα προσημασμένο (signed) ακέραιο από k (λιγότερα) bits σε n (περισσότερα) bits, δεν έχουμε παρά να κάνουμε το εξής: Προσθέτουμε αριστερά από τα bits που μας δόθηκαν (n-k) πρόσθετα bits τα οποία είναι όλα τους *αντίγραφα του αριστερού* (most significant) bit του αριθμού που μας δόθηκε, δηλαδή αντίγραφα του bit που υποδεικνύει το πρόσημο του δοθέντος αριθμού (0 γιά θετικούς ή το μηδεν, 1 γιά αρνητικούς). Η πράξη αυτή ονομάζεται επομένως, προφανώς, *επέκταση προσήμου* (sign extension).

8.4 Load Upper Immediate (lui), γιά Αυθαίρετες Σταθερές 32-bits:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7031. Στο βιβλίο η εντολή αυτή, καθώς και η τυπική χρήση της, περιγράφονται στην αρχή της §2.10, σελίδες 169-170. Η εντολή `lui` (load upper immediate) γράφει τα 16 bits του πεδίου Immediate της στα αριστερά 16 bits του καταχωρητή $rt, και μηδενίζει τα δεξιά 16 bits. Παρ' ότι ονομάζεται "load" όμως **δεν** είναι εντολή προσπέλασης της μνήμης δεδομένων --δεν ανήκει στην κατηγορία των εντολών load. Στη συνήθη χρήση της ακολουθείται από μιάν εντολή `ori` (OR immediate), η οποία κάνει

το λογικό bit-wise OR του $rs με την 16-μπιτη ποσότητα Immediate της. Σε αντίθεση με την `addi`, η OR-immediate θεωρεί ότι το πεδίο Immediate της είναι απρόσημο (unsigned), και επομένως τα 16 αριστερά bits της πράξης OR είναι πάντα μηδενικά. Δεδομένου ότι και η `lui` έβαλε 16 μηδενικά στο δεξιό μισό του καταχωρητή, τελικά προκύπτει η συγκόλληση (concatenation) των δύο 16-μπιτων Immediates, άρα μιά αυθαίρετη 32-μπιτη σταθερή ποσότητα στον καταχωρητή προορισμού. Μπορούμε επίσης να χρησιμοποιήσουμε την `addiu` (add immediate unsigned) αντί της OR immediate. Παρατηρήστε ότι, εάν αντί της `ori` ή `addiu` χρησιμοποιούσαμε την `addi`, τα πράγματα θα ήταν πιό πολύπλοκα εάν το αριστερό bit του δεξιού μισού της επιθυμητής σταθεράς ήταν "1", διότι τότε η σταθερά Immediate της `addi` θα έμοιαζε αρνητική, και άρα θα είχαμε επέκταση προσήμου (των άσσων) πριν την πρόσθεση (επομένως η σταθερά στην `lui` θα έπρεπε να ήταν +1 σε σχέση με τα επιθυμητά 16 αριστερά bits της 32-μπιτης σταθεράς...).

8.5 Το Datapath του MIPS ενός κύκλου με πλήρεις λεπτομέρειες:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7048, και διαβάστε από το βιβλίο τις σελίδες 365 - 374.

8.6 Το (Συνδυαστικό) Κύκλωμα Ελέγχου:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7135, και διαβάστε από το βιβλίο τις σελίδες 374 - 387.

8.7 Πόσο αργό είναι το ρολόϊ της Απλής Υλοποίησης;

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7126, και διαβάστε από το βιβλίο τις σελίδες 387 - 389.

8.8 Περί Σπατάλης ή μη Υπολογιστικών Πόρων:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7128

8.9 Υλοποίηση Πολλαπλών Κύκλων ανά Εντολή, γιά Εξοικονόμηση Πόρων:

Δείτε το video της διάλεξης στο elearn.uoc.gr/mod/page/view.php?id=7129 . Το θέμα αυτό εμελετάτο εν εκτάσει (και ήταν αντικείμενο μικρού "project") σε παλαιότερες εκδόσεις αυτού του μαθήματος, και υπήρχε και στις προηγούμενες εκδόσεις του βιβλίου, αλλά δεν υπάρχει στην 4η έκδοση του βιβλίου, προκειμένου να επικεντρωθεί το μάθημα στην ομοχειρία (pipelining), και ομοίως και εμείς φέτος το καλύψαμε εξαιρετικά επίτροχάδην. Εάν θέλετε, πέραν της διάλεξης, να βρείτε και λεπτομερές περαιτέρω περιγραφικό υλικό, ανατρέξτε σε προηγούμενες εκδόσεις του βιβλίου, ή μελετήστε τις σημειώσεις 10, 11, και 12 του μαθήματος του έτους 2009.

Άσκηση 8: Οπτικοποίηση της Λειτουργίας του Απλού MIPS Ενός Κύκλου

Στην άσκηση αυτή θα χρησιμοποιήστε ένα εργαλείο προσομοίωσης και οπτικοποίησης της λειτουργίας της απλής υλοποίηση του MIPS σε ένα κύκλο ρολογιού ανά εντολή, όπως ακριβώς αυτή περιγράφεται στο βιβλίο, το εργαλείο "**ProcSim**" γραμμένο σε Java από τον James Garton, διαθέσιμο δωρεάν από το Διαδίκτυο (το πρωτότυπο υπάρχει στη διεύθυνση www.jamesgart.com/procsim/ και είναι μιά χαρά γιά Windows· γιά Linux και MacOS, προτιμήστε καλύτερα την τοπική έκδοση στην οποία παραπέμπει το παρακάτω εγχειρίδιο, και όπου έχει γίνει μιά μικροδιόρθωση σε σχέση με το πρωτότυπο). Διαβάστε και ακολουθήστε τις Οδηγίες Εγκατάστασης και Χρήσης του ProcSim από τη διεύθυνση:

www.csd.uoc.gr/~hy225/17a/ex08_procSim_man.pdf

Σκοπός την άσκησης είναι η εξοικείωση σας με το Datapath του επεξεργαστή και η κατανόηση της λειτουργίας των διάφορων συστατικών του μονάδων.

- Αρχικά τρέξτε το ProcSim και φορτώστε, μέσω File -> Open Sim, το αρχείο "MIPS R2000 5 all jmp and addi.xml" που είναι ολόκληρο το datapath που είδαμε στο μάθημα.

- Μελετήστε προσεκτικά το σχηματικό και αναγνωρίστε όλα τα συστατικά και τα καλώδια, τις μνήμες εντολών και δεδομένων, την ALU, τους διάφορους πολυπλέκτες, κτλ.
- Αφού μελετήσετε προσεκτικά το datapath, τρέξτε προσομοιώσεις με τα διάφορα έτοιμα προγράμματα που δίνονται μαζί με τον προσομοιωτή (αρχεία .asm). Υπενθυμίζεται οτί η υπό εκτέλεση εντολή φαίνεται γραμμοσκιασμένη στο παράθυρο της μνήμης εντολών. Δοκιμάστε ολα τα αρχεία και δείτε τα διαφορετικά μονοπάτια που χρησιμοποιούνται μέσα στο datapath, ανάλογα με τον τύπο εντολής. Παρατηρήστε προσεκτικά τις τιμές ελέγχου που ξεκινάνε απο το "Control" και πώς αυτές κανονίζουν την εκτέλεση της κάθε εντολής.
- Ανοίξτε τα αρχεία ".asm" για να δείτε τα περιεχόμενα τους. Θα βρείτε σχόλια που ξεκινούν με "#", καθώς και γραμμές της μορφής ".register $s1 4"· αυτές, δεν είναι κανονικές εντολές assembly, αλλα ψευδοεντολές του συγκεκριμένου προσομοιωτή γιά την αρχικοποίηση των καταχωρητών.

(1) Μπορούν αυτές οι ψευδοεντολές να παραλειφθούν; Θα μπορούσατε να τις αντικαταστήσετε με κανονικές εντολές του επεξεργαστή; Εάν ναι, να γράψετε τις παρακάτω αρχικοποιήσεις με κανονικές εντολές:

```
.register $s1 1
.register $s2 2
```

Προσέξτε ότι στα προγράμματα που θα γράψετε πρέπει να συμπεριλάβετε τα labels `main:` και `exit:` όπως στα υπάρχοντα παραδείγματα, για να λειτουργήσει κανονικά ο προσομοιωτής. Επίσης, προσέξτε οτι η πρώτη εντολή θα πρέπει να είναι στην ίδια γραμμή με το label "main:"

(2) Γράψτε ένα δικό σας πρόγραμμα σε ένα αντίστοιχο αρχείο .asm που να εκτελεί τις εντολές R-format που υποστηρίζει το ProcSim: `add, sub, and, or, slt`. Τις add, sub, και slt, επαναλάβετέ τις δύο φορές καθεμία, με διαφορετικά ορίσματα (καταχωρητές πηγής ή προορισμού) καθεμία. Μπορείτε να χρησιμοποιήσετε τις ψευδοεντολές αρχικοποίησης των καταχωρητών που είδαμε παραπάνω. Απο τα σήματα ελέγχου (μαζί και αυτά που ελέγχουν την ALU) ποιά είναι ίδια για όλες τις εντολές αυτού του τύπου, ποιά διαφέρουν, και **γιατί**;

(3) Μελετήστε (γράφοντας ένα δικό σας πρόγραμμα και τρέχοντας το στον προσοοιωτή) τις εντολές `lw` και `sw`, με όμοιο τρόπο όπως παραπάνω. Τι πράξη εκτελείται στην ALU για αυτές τις εντολές; Είναι η ίδια και για τις δύο και γιατί; Τα δύο σήματα "ALUop" (από τον κυρίως έλεγχο στον δευτερεύοντα, και από εκείνον στην ALU), που ελέγχουν την πράξη που θα γίνει στην ALU, τι τιμή έχουν για τις εντολές sw και lw και με ποιάν απο τις εντολές R-format του προηγούμενου ερωτήματος είναι ίδια;

(4) Δοκιμάστε την ίδια διαδικασία όπως παραπάνω και για τις εντολές I-format: `addi, andi ori`. Ποιά σήματα ελέγχου είναι ίδια για όλες τις εντολές και ποιά διαφέρουν; Ειναι τα σήματα ALUop και η αντίστοιχη πράξη που γίνεται στην ALU ίδια με αντίστοιχες εντολές R-format; Ποιά είναι η διαφορά ανάμεσα στις εισόδους της ALU; Από πού προέρχονται οι τελεστέτοι για κάθε πράξη, ποιός πολυπλέκτης το καθορίζει, και ποια είναι η διαφορά με τις αντίστοιχες εντολές R-format (π.χ add και addi);

(5) Γραψτε ένα πρόγραμμα που να χρησιμοποιεί τις εντολές διακλάδωσης, `j` (jump) και `beq`. Μελετήστε το κομμάτι του datapath που υπολογίζει την επόμενη τιμή του PC και δείτε πώς αλλάζει η επιλογή του επόμενου PC ανάλογα με την εντολή. Ποιές είναι οι δυνατές τιμές που μπορεί να πάρει ο PC στην εντολη beq, δηλαδή ποιά τιμή παίρνει αν είναι επιτυχημένη η διακλάδωση, και ποιά αν δεν είναι; Ποια τιμή παίρνει ο PC αν εκτελεστεί μια εντολή j (jump); Ποιό σήμα ελέγχου και ποιός πολυπλέκτης καθορίζει την τιμή που θα φορτωθεί τελικά;

Τρόπος Παράδοσης:
Παραδώστε την άσκηση αυτή on-line, σε μορφή **PDF** (μόνον) (μπορεί να είναι κείμενο μηχανογραφημένο ή/και "σκαναρισμένο" χειρόγραφο, αλλά *μόνον* σε μορφή PDF). Παραδώστε μέσω **`turnin ex08@hy225 [directoryName]`** ένα αρχείο ονόματι **`ex08.pdf`** που θα περιέχει τις απαντήσεις σας σε όλες τις παραπάνω ερωτήσεις (1) έως και (5), καθώς και τα προγράμματα που γράψατε (αρχεία .asm) σε μορφή απλού κειμένου μέσα στο κείμενο των απαντήσεών σας.
