ΗΥ-225: Οργάνωση Υπολογιστών
Άνοιξη 2007

Τμ. Επ. Υπολογιστών

Σειρά Ασκήσεων 12:
Συμπλήρωση των Υπολοίπων Εντολών

Προθεσμία έως Δευτέρα 4 Ιουνίου 2007, ώρα 23:59 (βδομάδα 11)
Διαλέξεις εβδομάδων 8-10: Δε. 14/5, Τε. 16/5 - Δε. 21/5, Τε. 23/5, Πα. 25/5 - Τε. 30/5, Πα. 1/6

Άσκηση 12.1: Προσθήκη των Εντολών που λείπουν

Στην 10η και 11η σειρά ασκήσεων δεν είχαμε περιλάβει μερικές από τις εντολές που είχαμε μελετήσει στο κεφάλαιο 2. Τροποποιήστε το datapath σας και το κύκλωμα ελέγχου σας γιά να μπορεί ο επεξεργαστής σας να εκτελεί και αυτές --βάλτε τα αλλαγμένα modules σας στα αρχεία "**datapath12.v**" και "**control12.v**". Ακολουθεί ο κατάλογός των εντολών που πρέπει να προστεθούν, με μερικά σχόλια γιά καθεμία. Τα op-codes και funct-codes των εντολών του MIPS φαίνονται στη σελίδα 121 (Α' τόμος) ή 118 (Β' τόμος) του βιβλίου (Ελληνική έκδοση).

addi, slti:
Οι εντολές αυτές μοιάζουν με τις add και slt, αλλά ο δεύτερος τελεστέος τους είναι το Immx και όχι ο καταχωρητής B. Επίσης, καταχωρητής προορισμού είναι ο rt και όχι ο rd. Αρα, θα χρειαστούν αλλαγές στο κύκλωμα ελέγχου γιά να μπορεί να τις εκτελεί. Αν επιμείνετε να έχετε μόνο σήματα τύπου Moore, θα χρειαστούν νέες καταστάσεις, γιατί ο τρίτος και ο τέταρτος κύκλος της εκτέλεσης διαφέρουν από τις εντολές ALU με format τύπου R. Εναλλακτικά, μπορείτε να τις υλοποιήσετε με χρήση περισσοτέρων σημάτων τύπου Mealy, αλλά προσέξτε ότι δεν σας αρκούν πλέον το state και το funct --χρειάζεστε και το op-- γιά να ελέγξετε το ALUop, και υπάρχουν και άλλα σήματα ελέγχου που τώρα γίνονται τύπου Mealy.

lui:
Η εντολή lui (load upper immediate) χρησιμοποιείται γιά να φέρει μιά αυθαίρετη 16μπιτη σταθερά στο MS μέρος ενός καταχωρητή, και περιγράφεται στη σελίδα 146 του βιβλίου. Δεν είναι εντολή μνήμης (παρά το όνομα "load") --είναι έντολή λογικής/ολίσθησης με format τύπου I. Γιά να την υλοποιήσετε πρέπει να προσθέσετε στο datapath ένα δρόμο που να μπορεί να φέρει το Imm16 στο περισσότερο σημαντικό (MS) ήμισυ μιάς λέξης. Ο δρόμος αυτός μπορεί να περνάει από την ALU ή να την παρακάμπτει. Φυσικά, χρειάζονται και οι αντίστοιχες προσθήκες στον έλεγχο.

jr, jal:
Οι εντολές αυτές μοιάζουν με την j (jump) που ήδη έχετε, αλλά χρειάζονται νέοι δρόμοι στο datapath γιά να μπορεί να περάσει ο καταχωρητής regA μέχρι τον PC (jr), ή ο (ήδη αυξημένος κατά 4) PC να μπορεί να γραφτεί στο αρχείο καταχωρητών (jal).

beq, bne:
Η κατάσταση "branch" της FSM γιά την εκτέλεση των εντολών διακλάδωσης υπό συνθήκη περιλαμβάνονταν στην παράγραφο 11.1 (σε χρώμα μπλέ, και προσαρμοσμένη στην εντολή beq μόνο). Επίσης, στην § 11.1 εξηγήθηκε ο χρονισμός των διακλαδώσεων: Κατά τον τρίτο και τελευταίο κύκλο ρολογιού της εκτέλεσής τους, συγκρίνουν τους καταχωρητές $rs και $rt. Η σύγκριση αυτή, παρ' ότι απλή, γίνεται μέσω αφαίρεσης στην μία και μοναδική ALU που έχουμε, γιά λόγους εκπαιδευτικούς. Το αποτέλεσμα της αφαίρεσης ελέγχεται αν είναι ίσο με μηδέν, πράγμα που φαίνεται από την τιμή που παίρνει το σήμα "**zero**" αργά κατά τη διάρκεια του 3ου κύκλου ρολογιού. Λίγο πριν το τέλος του κύκλου, το σήμα επίτρεψης φόρτωσης του PC, "**pcld**", ελέγχεται γιά το αν θα ανάψει ή δεν θα ανάψει, γιά το αν δηλαδή θα αλλάξουμε ή δεν θα αλλάξουμε τελικά την τιμή του PC, με βάση την τιμή του σήματος zero, και φυσικά και με βάση το σε ποιά κατάσταση βρισκόμαστε. Ο έλεγχος αυτός γίνεται με ένα συνδυασμό πυλών AND-OR, όπως φαίνεται στο σχήμα 5.28 (σελίδα 341) του βιβλίου (Α' τόμος

Ελληνικής έκδοσης). Οι πύλες αυτές πρέπει να μας εξασφαλίζουν τη δυνατότητα να κάνουμε, ανάλογα με την κατάσταση και το σήμα zero, οιαδήποτε από τις εξής 4 δράσεις:

i. Φόρτωση του PC πάντα, δηλ. ανεξαρτήτως τιμής του "zero". Αυτό απαιτείται π.χ. στις καταστάσεις i_fetch και jump.
ii. Ο PC διατηρεί την τιμή του (όχι φόρτωση). Αυτό απαιτείται π.χ. στις καταστάσεις decode_rr, alu_exec, κλπ.
iii. Φόρτωση του PC **εάν** το σήμα zero είναι αληθές. Αυτό απαιτείται στην κατάσταση branch, εάν η εντολή είναι η μία από τις δύο --beq ή bne (ή στην μία από τις καταστάσεις τύπου branch, αν έχετε χωριστή κατάσταση γιά καθεμιά από τις beq και bne).
iv. Φόρτωση του PC **εάν** το σήμα zero είναι ψευδές. Αυτό απαιτείται στην κατάσταση branch, εάν η εντολή είναι η άλλη από τις δύο --bne ή beq (ή στην άλλη από τις δύο καταστάσεις τύπου branch, αν έχετε χωριστή κατάσταση γιά καθεμιά από τις beq και bne).

Γιά να μοντελοποιηθεί επακριβώς η χρονική συμπεριφορά και καθυστέρηση αυτής της πύλης AND-OR, προστέθηκε στη βιβλιοθήκη του project ένα σχετικό module, το "**And3or**". Αυτό περιλαμβάνεται, μαζί με όλα τα υπόλοιπα στοιχεία της βιβλιοθήκης που είχαμε ήδη από τις προηγούμενες ασκήσεις, στο αρχείο:

`~hy225/verilog/lib/lib12.v`

και χρησιμοποιείται ώς εξής:

```
And3or pcld_logic (pcld, pcld_z, pcld_nz, pcld_u, zero);
```

όπου pcld_z ("pcld-if-zero"), pcld_nz ("pcld-if-not-zero"), και pcld_u ("pcld-unconditional") είναι κατάλληλα σήματα ελέγχου που θα δημιουργήσετε. Η πύλη αυτή υλοποιεί τη λογική συνάρτηση:

*pcld = (pcld_z **AND** zero) **OR** (pcld_nz **AND** (**NOT** zero)) **OR** (pcld_u)*

και έχει καθυστέρηση 0.3 ns.

Αφού κάνετε τις αλλαγές και προσθήκες που ζητήθηκαν, προσομοιώστε τη λειτουργία του επεξεργαστή και διορθώστε τα λάθη, τόσο στο κύκλωμα ελέγχου όσο και στο datapath. Θα πρέπει να **τρέχουν σωστά** τουλάχιστο μερικές από κάθε είδος εντολής, με μη τετριμένες τιμές των τελεστέων τους.

Αφού τρέξουν όλα, υπολογίστε με τη βοήθεια του SignalScan πόση είναι η ελάχιστη δυνατή περίοδος του ρολογιού σας, και αλλάξτε το top-level ώστε να τρέχει ο επεξεργαστής σας με αυτό το **γρηγορότερο δυνατό ρολόϊ**. Πόσα MHz καταφέρατε να "πιάσετε";

Παραδώστε, όπως και στις προηγούμενες ασκήσεις, τον κώδικά σας "**control12.v**", τις αλλαγές που κάνατε στο datapath, "**datapath12.v**", το νέο test bench, "**test12.v**", τη νέα αρχικοποίηση μνήμης, "**memory12.hex**" ή "**memory12.bin**", και ένα χαρακτηριστικό στιγμιότυπο, "**signals12.jpg**", από το Signalscan της άσκησης σε μορφή jpg, πακεταρισμένα στο αρχείο "**`ask12.zip`**", μέσω:

```
zip ask12.zip   control12.v datapath12.v test12.v memory12.hex signals12.jpg
~hy225/bin/submit 12
```

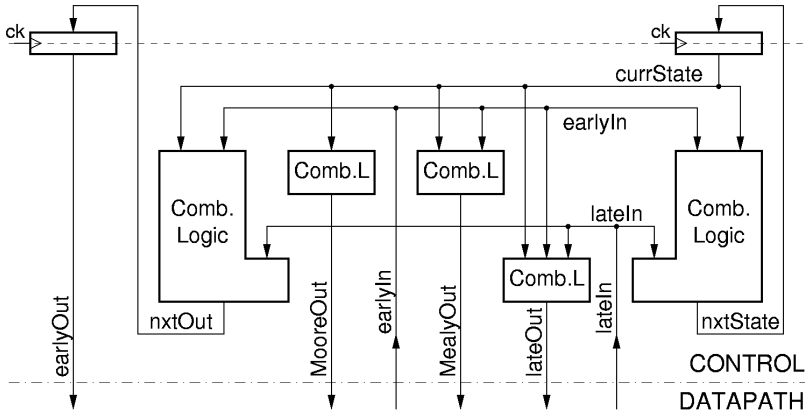
Τα παραπάνω --και ιδιαίτερα ο έλεγχος και η διόρθωση των λαθών (debugging)-- έχουν πρώτη προτεραιότητα. Εάν τα τελειώσετε εγκαίρως και έχετε όρεξη και γιά περαιτέρω βελτιώσεις, ρίξτε μιά ματιά και στις παρακάτω προαιρετικές ασκήσεις....

<u>Προαιρετική</u> Άσκηση 12.2: Βελτιστοποίηση Εντολών Αλματος

Οσοι από σας ενδιαφέρεστε ιδιαίτερα γιά το hardware και έχετε χρόνο, αλλάξτε το κύκλωμα ελέγχου, και πιθανόν και το datapath, προκειμένου οι εντολές άλματος **j** και **jr** να εκτελούνται σε **δύο (2) μόνο** κύκλους ρολογιού αντί των τριών (3), τώρα, κύκλων ρολογιού. Μπορείτε να επιτύχετε το ίδιο και γιά την εντολή **jal** ; Πόσος είναι ο ταχύτερος δυνατός κύκλος ρολογιού μετά τις αλλαγές που κάνατε; Παρατηρήστε ότι αν ο νέος κύκλος ρολογιού είναι αργότερος από τον παλαιό κατά παραπάνω από περίπου 1%, τότε οι αλλαγές δεν μας συμφέρουν, δεδομένου ότι οι εντολές που βελτιώνουμε αποτελούν μόνο περίπου το 3 με 5 % των εκτελουμένων εντολών στα συνηθισμένα προγράμματα!....

Προαιρετική Άσκηση 12.3: Βελτιστοποίηση Χρονισμού Σημάτων Ελέγχου

Οσοι από σας ενδιαφέρεστε ιδιαίτερα γιά το hardware και έχετε χρόνο, αλλάξτε τον τρόπο γέννησης ορισμένων κρίσιμων σημάτων ελέγχου, όπως θα εξηγήσουμε παρακάτω, ώστε αυτά να αποκτούν τη σωστή (έγκυρη) τιμή τους πιό νωρίς μέσα στον κύκλο ρολογιού, με αποτέλεσμα να μπορείτε να πετύχετε γρηγορότερο ρολόϊ, άρα γρηγορότερο επεξεργαστή. Το παρακάτω σχήμα είναι μιά εξειδίκευση και μετεξέλιξη του αντίστοιχου σχήματος στην §11.2, και δείχνει περισσότερες λεπτομέρειες χρονισμού, καθώς και τον νέο τρόπο γέννησης σημάτων ελέγχου που συζητάμε εδώ, "**earlyOut**".



Η "καρδιά της FSM είναι ο καταχωρητής **κατάστασης** --πάνω δεξιά στο σχήμα-- που περιέχει (θυμάται) την παρούσα κατάσταση, "**currState**", του συστήματος. Ενα μπλοκ Συνδυαστικής Λογικής (combinational logic) --κάτω από αυτόν, στο σχήμα-- υπολογίζει (αποφασίζει) ποιά θα είναι η επόμενη κατάσταση, "**nxtState**" (αμέσως μετά την επερχόμενη ακμή του ρολογιού), σαν συνάρτηση της παρούσας κατάστασης και των εισόδων του κυκλώματος ελέγχου. Οι έξοδοι του κυκλώματος ελέγχου, δηλαδή τα σήματα ελέγχου του datapath, είναι επίσης συναρτήσεις της παρούσας κατάστασης, και μπορεί να είναι συναρτήσεις και των εισόδων του κυκλώματος ελέγχου.

Τα σήματα εισόδου στο κύκλωμα ελέγχου αποκτούν έγκυρες τιμές (σταθεροποιούνται) σε διάφορες στιγμές μέσα στον κύκλο (περίοδο) του ρολογιού --άλλα πολύ νωρίς, άλλα αργότερα, κι άλλα πολύ αργά. Παράδειγμα εισόδων του ελέγχου που σταθεροποιούνται πολύ νωρίς μέσα στον κύκλο ρολογιού είναι τα **op** και **funct**, που είναι κατευθείαν έξοδοι του καταχωρητή **ir** και άρα σταθεροποιούνται σχεδόν αμέσως μόλις αρχίζει ο κύκλος --μόλις περάσει η καθυστέρηση εξόδου του καταχωρητή **ir**, μόνο, μετά την ακμή ρολογιού. Τέτοια πρώιμα σήματα, στο σχήμα παραπάνω, τα συμβολίζουμε σαν "**earlyIn**, και τα έχουμε σχεδιάσει σχετικά ψηλά ώστε να μας θυμίζουν ότι είναι διαθέσιμα από νωρίς (αν φανταστούμε το χρόνο να κυλά από πάνω προς τα κάτω).

Παράδειγμα εισόδου του ελέγχου που σταθεροποιείται πολύ αργά μέσα στον κύκλο ρολογιού είναι το σήμα **zero**, όπως είπαμε στην άσκηση 12.1 παραπάνω. Τέτοια αργοπορημένα σήματα, στο σχήμα παραπάνω, τα συμβολίζουμε σαν "**lateIn**, και τα έχουμε σχεδιάσει σχετικά χαμηλά ώστε να μας θυμίζουν ότι γίνονται διαθέσιμα πολύ αργά. Τα αργοπορημένα σήματα εισόδου του ελέγχου μπορούν να συμμετάσχουν μόνο σε πολύ απλούς, δηλαδή γρήγορους, υπολογισμούς --αλλοιώς οι έξοδοι αυτών των υπολογισμών δεν θα προλάβαιναν να σταθεροποιηθούν (να αποκτήσουν έγκυρες τιμές) πριν τελειώσει ο κύκλος ρολογιού. Στο παραπάνω σχήμα αυτό το δείχνουμε γιά μνημονικούς λόγους με "κοντύτερα" (στον κατακότυφο άξονα) μπλόκ συνδυαστικής λογικής.

Όπως έχουμε πεί και στην άσκηση 11, οι έξοδοι "**MooreOut**" της FSM είναι συναρτήσεις μόνο της παρούσας κατάστασης, ενώ οι έξοδοι "**MealyOut**" είναι συναρτήσεις και της παρούσας κατάστασης και των εισόδων της FSM.

Τα σήματα ελέγχου τύπου Moore, καθώς και τα τύπου Mealy που εξαρτώνται μόνο από πρώιμα σήματα εισόδου (earlyIn), μπορούν να στεθεροποιούνται σχετικά νωρίς στη διάρκεια του κύκλου ρολογιού,

όταν η λογική συνάρτηση που τα γεννάει είναι αρκούντως απλή (γρήγορη). Ετσι, οι λειτουργίες ή πράξεις στο datapath που εξαρτώνται από τέτοια σήματα ελέγχου δεν θα καθυστερούν πολύ να αρχίσουν (αναγκαστικά, αυτές δεν μπορούν να αρχίσουν πριν να σταθεροποιηθούν τα σήματα που τις ελέγχουν ή τις ενεργοποιούν). Παρ' όλα αυτά, η --έστω και μικρή-- καθυστέρηση της λογικής που γεννάει αυτά τα σήματα αποβαίνει σε βάρος του όλου κύκλου ρολογιού. Ουσιαστικά, στην αρχή του κάθε κύκλου ρολογιού, μόλις εμφανιστεί η νέα κατάσταση στην έξοδο του καταχωρητή currState (και τα earlyIn π.χ. σε εξόδους καταχωρητών όπως του ir), και μέχρι να τελειώσει η λειτουργία της λογικής που γεννά τα σήματα ελέγχου, κανένας άλλος μέσα στον επεξεργαστή δεν μπορεί να αρχίσει να δουλεύει --όλοι κάθονται και περιμένουν να τους πεί ο έλεγχος τι να κάνουν....

Το ελάττωμα αυτό μπορεί να διορθωθεί με τη μέθοδο που φαίνεται αριστερά στο παραπάνω σχήμα. Μπορώ να προετοιμάσω (προ-ϋπολογίσω) από τον **προηγούμενο** κύκλο ρολογιού την τιμή nxtOut που πρέπει να πάρουν ορισμένα σήματα ελέγχου στον επόμενο κύκλο ρολογιού. Αν το κάνω αυτό, και χρησιμοποιήσω και τον καταχωρητή που φαίνεται πάνω-αριστερά στο σχήμα, τότε ευθύς μόλις αρχίσει ο επόμενος κύκλος ρολογιού --μόλις περάσει ίσα-ίσα η καθυστέρηση εξόδου του καταχωρητή-- τα σήματα ελέγχου "**earlyOut**" θα πάρουν έγκυρες και σταθερές τιμές, επιτρέποντας έτσι στο datapath να αρχίσει αμέσως να δουλεύει. (Ουσιαστικά πρόκειται γιά μιά μορφή ομοχειρίας (pipelining) --τεχνική που όπως θα δούμε σύντομα μπορεί να εφαρμοστεί λίαν εποφελώς και στο datapath). Προφανώς, γιά να μπορέσω να προ-ϋπολογίσω από τον προηγούμενο κύκλο ρολογιού την τιμή που πρέπει να πάρουν ορισμένα σήματα ελέγχου στον επόμενο κύκλο ρολογιού, πρέπει να έχω όλα τα δεδομένα που μου χρειάζονται προς τούτο, άρα τα σήματα αυτά **δεν** μπορεί να εξαρτώνται από πληροφορίες που γίνονται γνωστές κατά τον "δικό" τους κύκλο ρολογιού και μόνο --πρέπει να εξαρτώνται μόνο από πληροφορίες που ήταν ήδη γνωστές από τον προηγούμενο κύκλο ρολογιού (άρα, βασικά, να ήταν τύπου Moore αν δεν τα προ-ϋπολογίζαμε, ή να μπορούσαν να γίνουν τύπου Moore με προσθήκη καταστάσεων).

Οταν θέλουμε να πετύχουμε γρήγορο ρολόϊ, την μέθοδο αυτή είναι κρίσιμο να την εφαρμόσουμε τουλάχιστο γιά εκείνα τα σήματα ελέγχου που κανονίζουν τις εργασίες που το datapath αρχίζει να τις εκτελεί στην αρχή-αρχή του κύκλου ρολογιού. Στο δικό μας datapath, τέτοια σήματα είναι τα:

- IorD, που επιλέγει τη διεύθυνση γιά να αρχίσει να δουλεύει η μνήμη,
- memRd και memWr, που λένε στη μνήμη τι να κάνει,
- regWr, mem2reg, regDst, που λένε στο RF αν, τι, και πού να γράψει,
- ALUsrcA και ALUsrcB, που επιλέγουν σε τι να δουλέψει η ALU, και
- ALUop, που λέει στην ALU τι δουλειά να κάνει.

Σε όλα αυτά είναι εφαρμόσιμη η μέθοδος, διότι όλα αυτά ξέρουμε από τον προηγούμενο κύκλο ρολογιού τι τιμή πρέπει να πάρουν. Γιά να εφαρμόσετε τη μέθοδο αυτή, ορίστε έναν καταχωρητή (του ενός ή των όσων λίγων bits χρειάζεται) γιά κάθε σήμα που θα προϋπολογίζετε, με την ίδια μέθοδο που χρησιμοποιήσαμε και γιά τον καταχωρητή currState, και περιγράψτε τη λειτουργία (φόρτωση) των καταχωρητών αυτών με ανάλογο τρόπο --και με την ίδια καθυστέρηση-- όπως κάναμε και γιά τον currState. Ο υπολογισμός των εισόδων "nxtOut" αυτών των καταχωρητών μπορεί να γίνει είτε σαν συνδυαστική συνάρτηση της currState και των εισόδων (όπως δείχνει το παραπάνω σχήμα), είτε σαν συνδυαστική συνάρτηση της nxtState --που είναι και η ίδια συνδυαστική συνάρτηση της currState και των εισόδων-- και πιθανά και των εισόδων.

Up to the Home Page of CS-225

last updated: 15 May 2007, by M. Katevenis.