3ή Εργαστηριακή Άσκηση

Tomasulo + Reorder Buffer

Ομάδα: LAB41538988

ΜΑΝΩΛΗΣ ΠΕΤΡΑΚΟΣ ΑΜ 2014030009
ΧΡΗΣΤΟΣ ΖΗΣΚΑΣ ΑΜ 2014030191

Διεπαφές

Reorder Buffer

Σήμα	Πλάτος	Είδος	Περιγραφή
Clk	1 bit	Είσοδος	Ρολόι
cdb_valid	1 bit	Είσοδος	Σήμα εγκυρότητας δεδομένων CDB
cdb_Q	5 bit	Είσοδος	Tag θέσης στην οποία αναφέρεται ο CDB
cdb_value	32 bit	Είσοδος	Αποτέλεσμα κοινοποίησης του CDB
available	1 bit	Έξοδος	Δείχνει αν ο ROB έχει διαθέσιμο χώρο
instr_valid	1 bit	Είσοδος	Σήμα εγκυρότητας εντολής που γίνεται issue
Fu_type	2 bit	Είσοδος	Τύπος εντολής
Ri	5 bit	Είσοδος	Αριθμός καταχωρητή προορισμού
PC_entolhs	32 bit	Είσοδος	Program counter της εντολής
tag_issue	5 bit	Έξοδος	Θέση όπου αποθηκεύτηκε η εντολή
Rj	5 bit	Είσοδος	Αριθμός καταχωρητή πηγής #1
Rk	5 bit	Είσοδος	Αριθμός καταχωρητή πηγής #2
Qj	5 bit	Έξοδος	Tag δεδομένων καταχωρητή πηγής #1
Qk	5 bit	Έξοδος	Tag δεδομένων καταχωρητή πηγής #2
forward_data_j	32 bit	Έξοδος	Δεδομένα για forward κατά το διάβασμα του καταχωρητή πηγής #1
forward_data_k	32 bit	Έξοδος	Δεδομένα για forward κατά το διάβασμα του καταχωρητή πηγής #2
forward_control_j	1 bit	Έξοδος	Σήμα ελέγχου για forward κατά το διάβασμα του καταχωρητή πηγής #1
forward_control_k	1 bit	Έξοδος	Σήμα ελέγχου για forward κατά το διάβασμα του καταχωρητή πηγής #2
commit_reg	5 bit	Έξοδος	Αριθμός καταχωρητή για commit
commit_data	32 bit	Έξοδος	Δεδομένα για commit
V_commit	1 bit	Έξοδος	Σήμα εγκυρότητας commit
PC_exception	32 bit	Έξοδος	Program counter της εντολής που έκανε exception

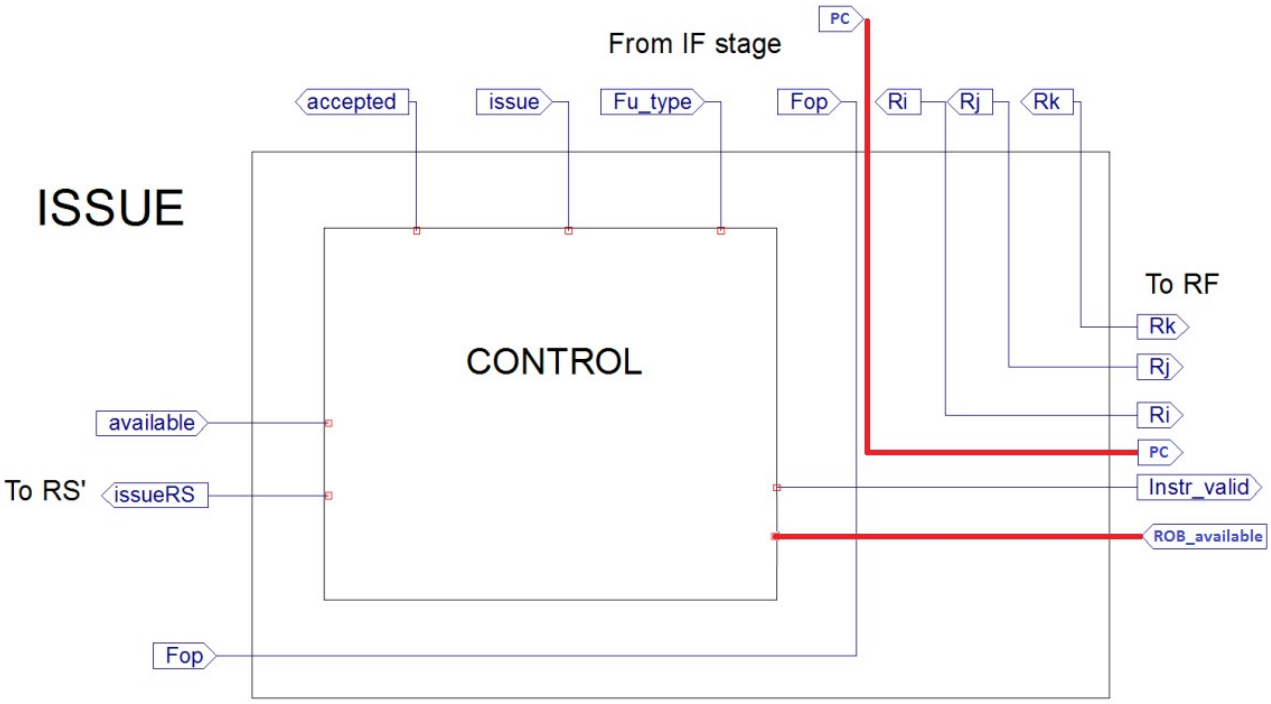
Οι υπόλοιπες διεπαφές έχουν μείνει ίδιες ή έχουν αλλάξει ελαφρά. Επειδή όλες οι αλλαγές αφορούν σήματα του παραπάνω πίνακα, δεν υπάρχει ιδιαίτερη σημασία να παρουσιαστούν, καθώς δεν

προσφέρουν κάποια παραπάνω επεξήγηση. Επίσης, η σχεδίαση του νέου συστήματος έγινε με γνώμονα να γίνουν οι λιγότερο δυνατόν αλλαγές στις υπόλοιπες μονάδες.

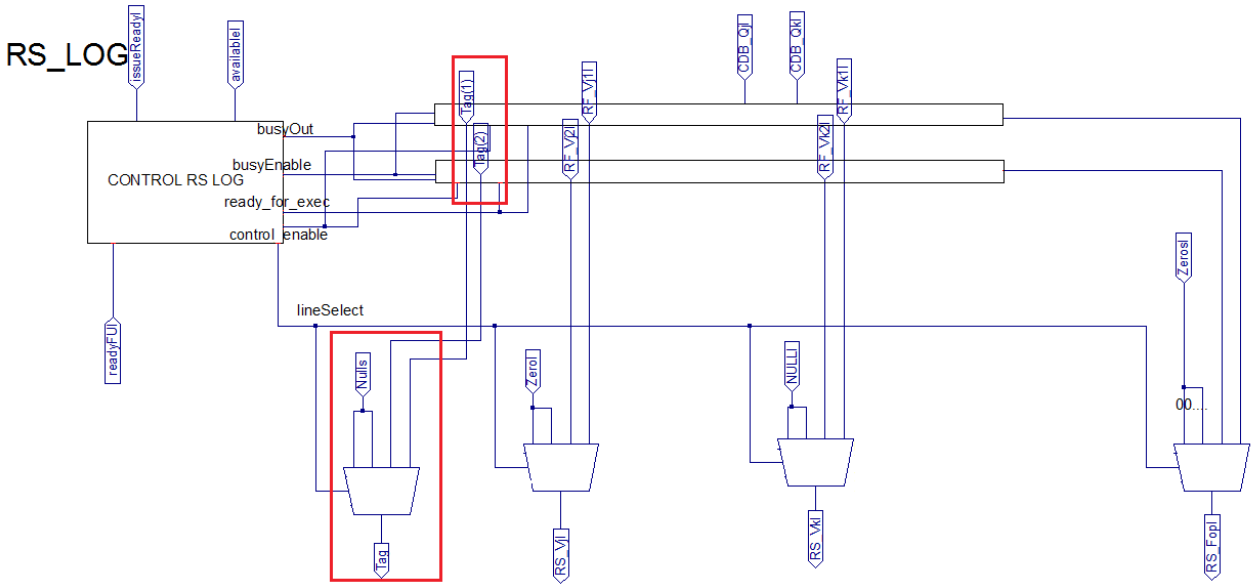
Σχηματικά διαγράμματα

Οι αλλαγές φαίνονται με κόκκινο χρώμα.

ΒΑΘΜΙΔΑ Issue

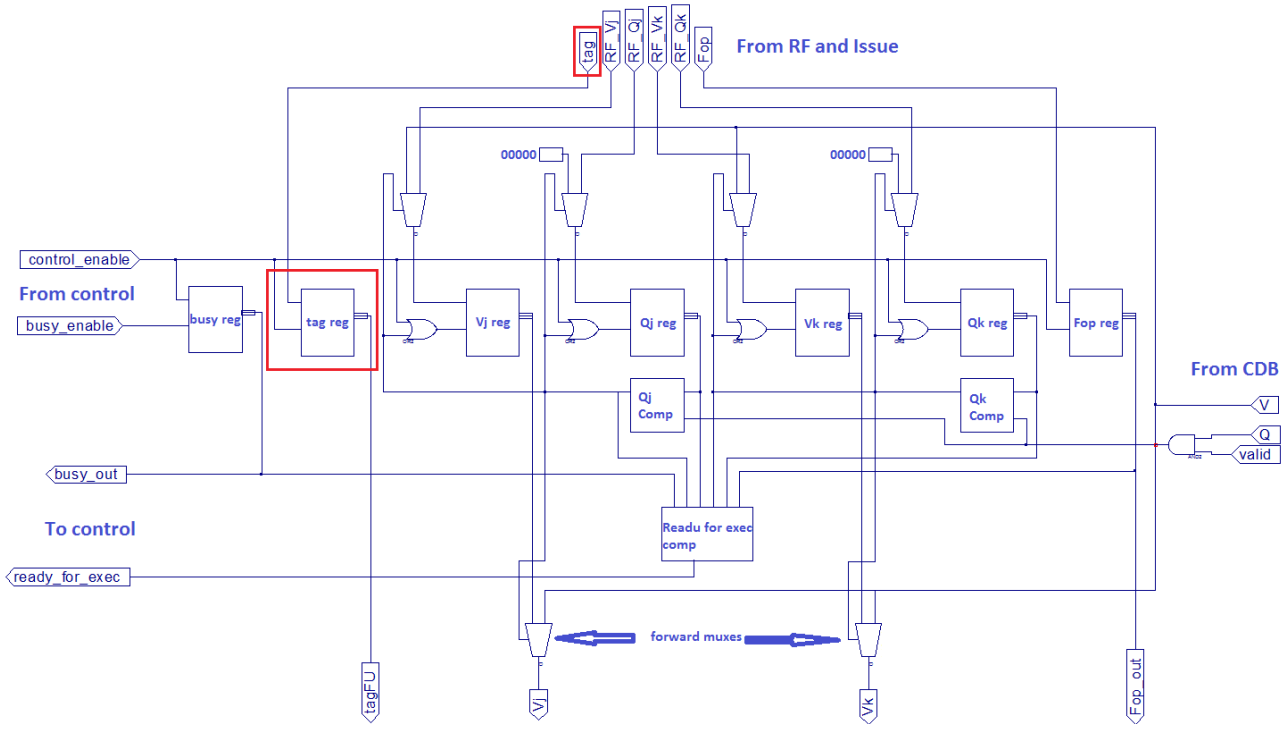


ΒΑΘΜΙΔΑ Reservation Station-logical

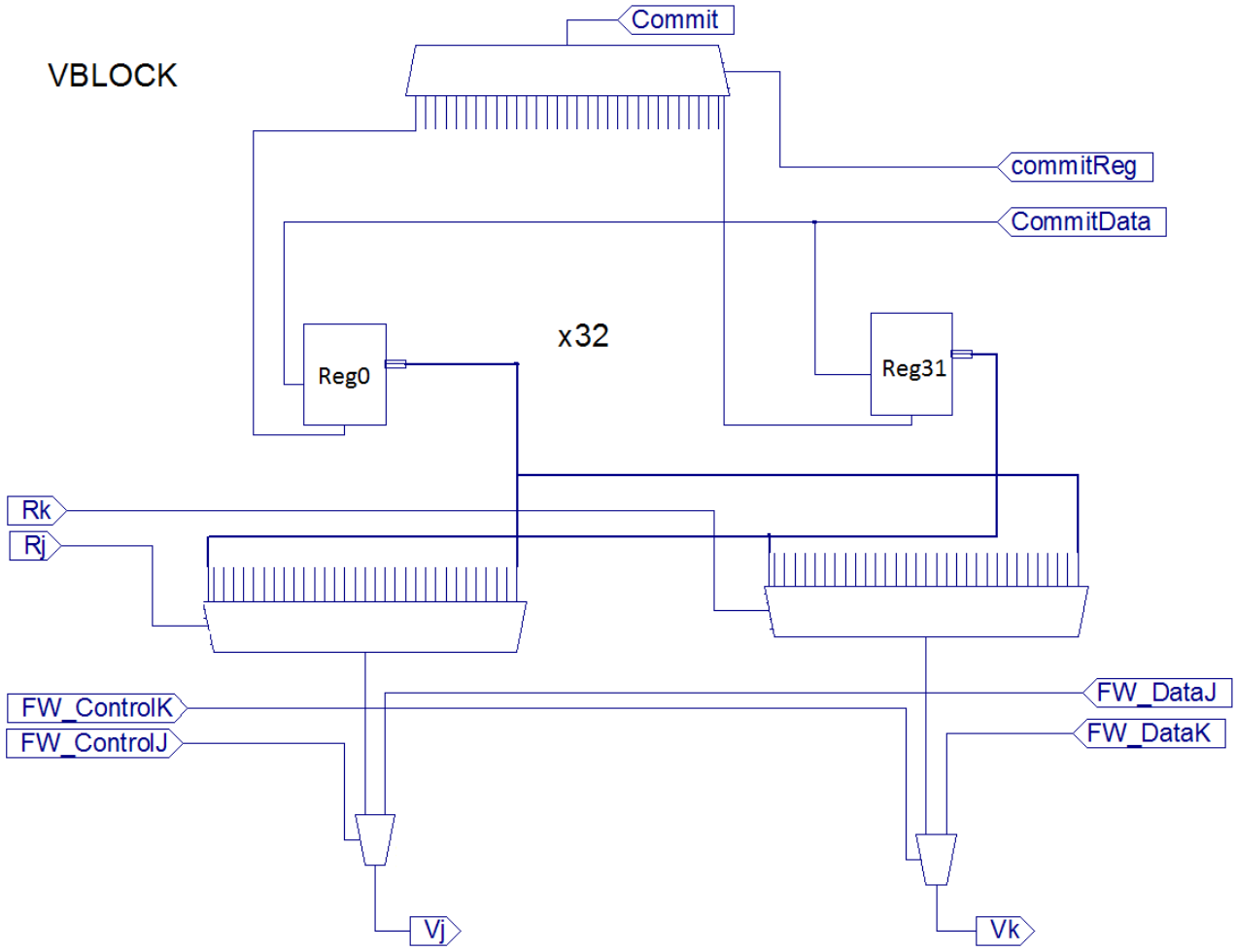


Το σχηματικό διάγραμμα της της αριθμητικής μονάδας είναι το ίδιο με μία παραπάνω θέση για εντολές. Αντίστοιχα, τα σήματα ελέγχου και οι πολυπλέκτες έχουν μεγαλύτερο πλάτος.

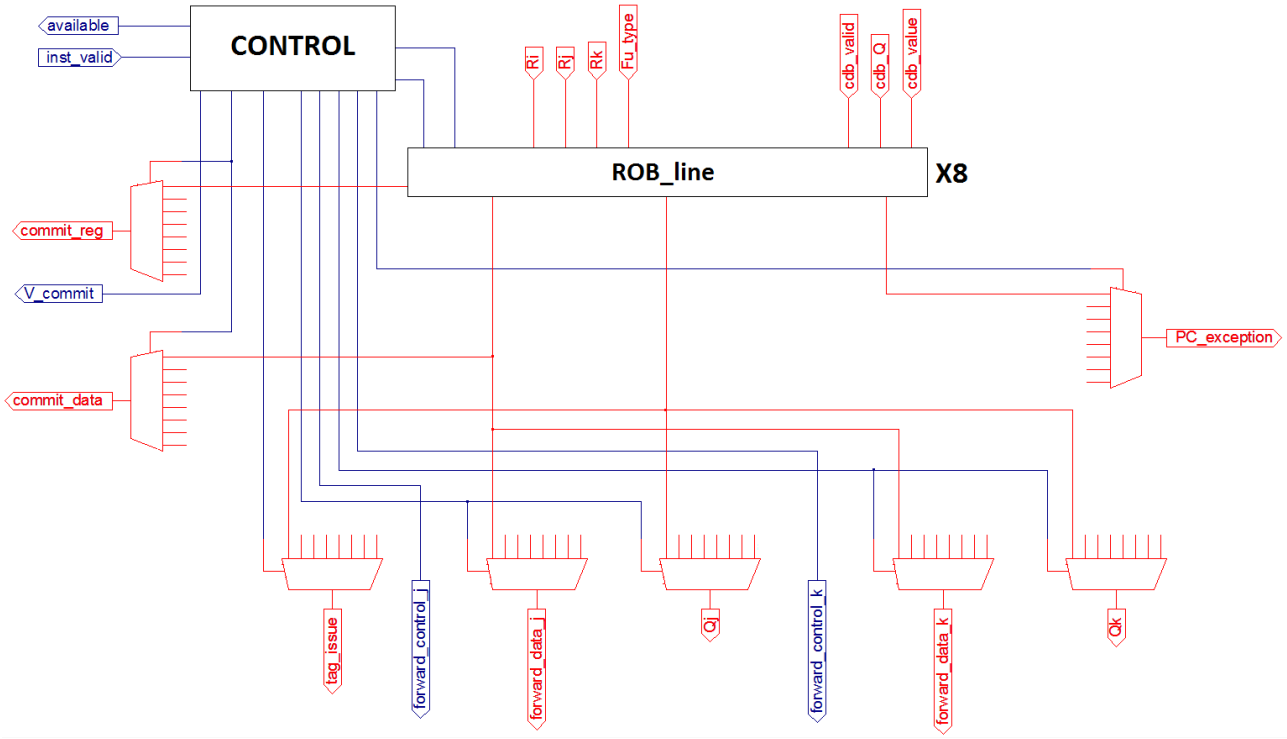
RS Block



Vblock

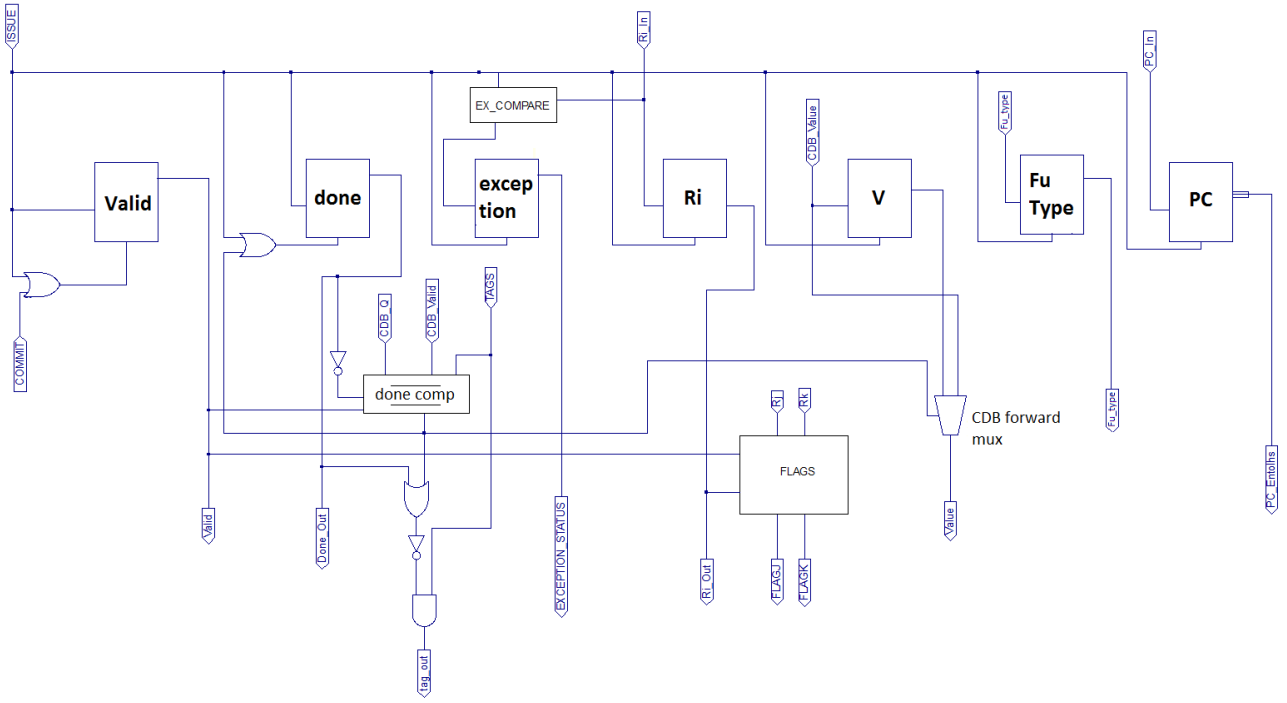


Reorder buffer

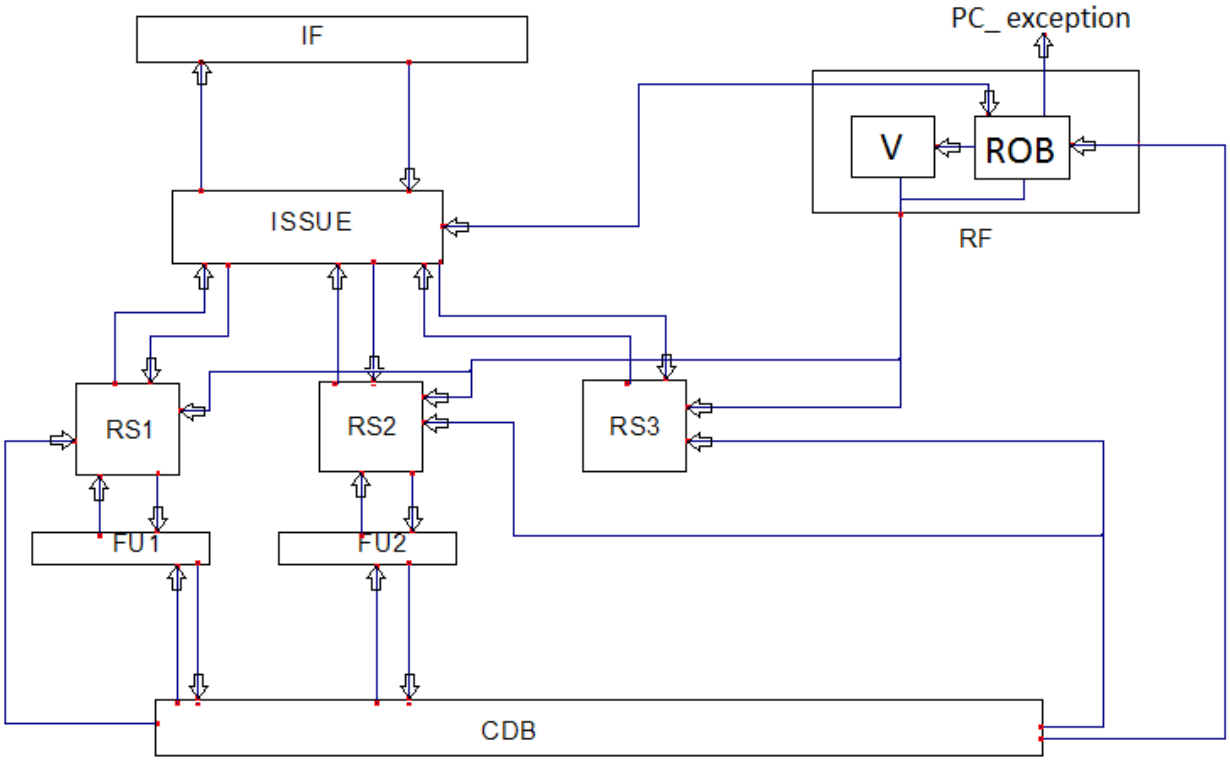


Λόγω υψηλής πολυπλοκότητας, τα σήματα που αφορούν το data-path είναι με κόκκινο χρώμα και τα σήματα ελέγχου με μπλε.

Buffer line



Διάγραμμα ολοκληρωμένου



Ο reorder buffer υλοποιήθηκε μέσα στην register file με σκοπό να γίνουν οι λιγότερο δυνατόν μετατροπές στο υπόλοιπο σύστημα και το top module.

Διάγραμμα χρονισμού

Εντολή AND. Θεωρούμε ότι δεν υπάρχουν συγκρούσεις και τα δεδομένα πηγής είναι σωστά.

➢ 1ος κύκλος
Έλεγχος δομικών κινδύνων από το Issue.
Διάβασμα καταχωρητών πηγής από τον RF. Forward από τον ROB αν χρειάζεται.
Αποθήκευση εντολής στο ROB. Αποστολή του tag στο RS.
Αποθήκευση εντολής στο RS.

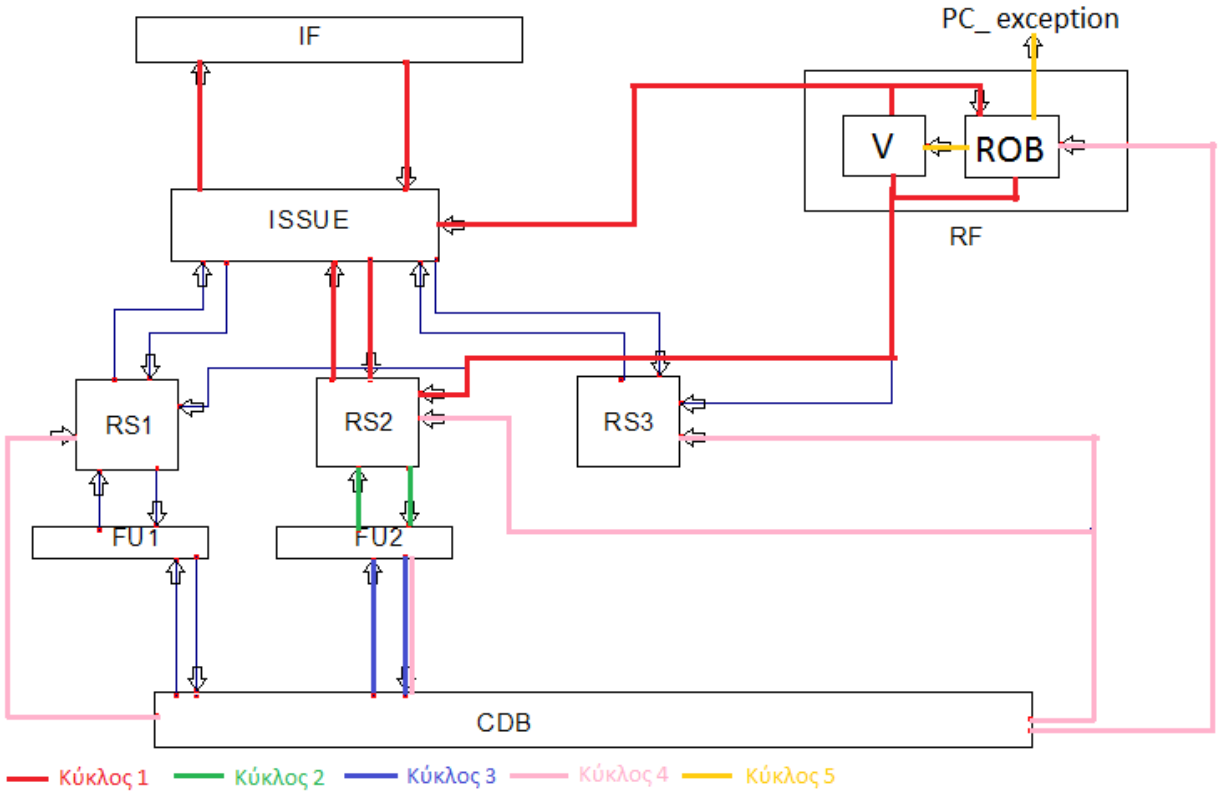
➢ 2ος κύκλος
Αποστολή της εντολής από το RS στο FU. Διαγραφή εντολής από το RS.

➢ 3ος κύκλος
Εκτέλεση πράξης στο FU.
Το FU ζητάει από τον έλεγχο του CDB άδεια πρόσβασης για τον επόμενο κύκλο.
Το CDB απαντάει άμεσα.

➢ 4ος κύκλος
Αποστολή αποτελέσματος από το FU στο CDB. Κοινοποίηση αποτελέσματος από το CDB στο ROB και τα RS.
Αποθήκευση αποτελέσματος στα RS που το περιμένουν και write-back στον ROB.

➢ 5ος κύκλος
Commit στον κατάλληλο καταχωρητή του V από το ROB. Διαγραφή εντολής στον ROB. Εφαρμογή exception αν γίνει.

Το παρακάτω διάγραμμα απεικονίζει σχηματικά την προαναφερθείσα περιγραφή.



Περιγραφή αλλαγών

Οι παρακάτω μονάδες έχουν δεχτεί διάφορες τροποποιήσεις.

Issue: Το Issue επικοινωνεί με τον Reorder buffer. Ο ROB ενημερώνει συνέχεια το Issue αν είναι διαθέσιμος να λάβει νέα εντολή. Αν ο ROB έχει διαθέσιμη γραμμή (available = '1'), το Issue εισάγει την εντολή και ανακατευθύνει τα πεδία της στις υπόλοιπες μονάδες. Επίσης, το tag της κάθε εντολής δεν περνάει πια από το Issue αλλά απευθείας μεταξύ ROB και RS. Τέλος, έχει προστεθεί το πεδίο Program Counter ενώ η υπόλοιπη λειτουργικότητα έχει διατηρηθεί.

Reservation Station: Το Tag πλέον δεν δημιουργείται από τη μονάδα RS, αλλά στέλνεται από το ROB κατά την έκδοση της εντολής. Για αυτόν το λόγο, έχουν προστεθεί καταχωρητές και πολυπλέκτες για την αποθήκευση και διάβασμα των tag. Επίσης, το RS πλέον χρειάζεται να κρατάει την εντολή μόνο μέχρι να την στείλει στο FU. Τότε θα διαγραφτεί και στον ίδιο κύκλο μπορεί να μπει μια καινούργια στη θέση της. Το τελευταίο γίνεται, αντί διαγράφοντας την παλιά εντολή, πανωγράφοντας την με την καινούργια. Σαν αποτέλεσμα, οι θέσεις είναι pipelined και δεν μένουν ποτέ ανεκμετάλλευτες. Με αυτό το optimization, αν όλες οι θέσεις είναι γεμάτες και μία αδειάζει, μπορεί να μπει απευθείας μια καινούργια.

Αφού η αποθήκευση των εντολών γίνεται και στο ROB και χρησιμοποιούν τα tag αυτού, δεν υπάρχει πρόβλημα να υπάρχουν μέσα στο σύστημα πολλές εντολές που έχουν περάσει από την ίδια θέση του RS. Πριν, επειδή τα tags προέρχονταν από τα RS αυτή ήταν μια απαγορευτική κατάσταση. Σαν αποτέλεσμα, απλοποιείται το control τους, συγχωνεύοντας τις λειτουργίες της διαγραφής και αποστολής εντολής προς το FU. Η παραπάνω λειτουργία έχει προσαρμοστεί και στους δυο τύπους RS. Η υπόλοιπη λειτουργικότητα τους διατηρείται ως έχει.

Vblock: Το VBlock αποτελείται από τους καταχωρητές της Register File και πολυπλέκτες για διάβασμα και commit αποτελεσμάτων. Κατά το διάβασμα, θα στείλει τα δεδομένα από τους καταχωρητές πηγής (Rj, Rk)

εκτός εάν ο ROB ειδοποιήσει (fw_control_j = ‘1’, fw_control_k = ‘1’) ότι έχει νεότερα αποτελέσματα για αυτούς. Τότε θα περάσουν αυτά (fw_data_j, fw_data_k). Για το commit, αν ο ROB ζητήσει (commit = ‘1’) να αποθηκευτούν δεδομένα, τότε αυτά(commit_data) στον καταχωρητή που έχει υποδειχτεί (commit_reg).

Περιγραφή Reorder Buffer

Δομή
Η μονάδα ROB αποτελείται από:

- 8 θέσεις εντολών.

Υπάρχουν οχτώ θέσεις εντολών για να υπάρχουν περισσότερες από ότι στα RS. Οι θέσεις στα RS είναι 5 και η επόμενη μεγαλύτερη δύναμη του 2 είναι το 8. Θέλουμε περισσότερες θέσεις στο ROB από ότι στα RS, για να μπορούν να χρησιμοποιηθούν όλοι οι πόροι του συστήματος ταυτόχρονα. Προτιμήθηκε ο αριθμός των θέσεων να είναι δύναμη του 2 για να γίνει πιο απλός ο έλεγχος της μονάδας.

- Σύστημα ελέγχου (control).

Ο έλεγχος είναι μερικώς αποκεντροποιημένος, καθώς οι θέσεις πέρα από καταχωρητές και πολυπλέκτες, έχουν συγκριτές και λογική. Το σύστημα ελέγχου δεν παίρνει μέρος σε όλες τις λειτουργίες και σε αυτές που συμμετέχει δεν είναι ο μοναδικός παράγοντας.

- 2 shift registers όπου λειτουργούν σαν pointers.

Έχουν 8 bit, ένα για κάθε θέση. Πάντα ένα από αυτά έχει την τιμή 1 και δείχνει το αντικείμενο των δεικτών. Ο πρώτος δείκτης (head) δείχνει την παλαιότερη έγκυρη εντολή και ο άλλος (free) την πρώτη διαθέσιμη θέση.

- Λογική datapath για την επιλογή εντολών και δεδομένων.(πολυπλέκτες κλπ.)

Λειτουργίες
Η μονάδα εξυπηρετεί πέντε λειτουργίες.

Η πρώτη είναι η αποθήκευση εντολών κατά την έκδοσή τους. Η μονάδα ενημερώνει συνέχεια το Issue module αν έχει διαθέσιμες θέσεις η όχι(available_rob). Αυτό συμβαίνει αν κάποια θέση δεν έχει έγκυρη εντολή ($valid_{θέσης}$ = ‘0’) ή κάποια εντολή διαγράφεται αυτήν την στιγμή. Αν έχει ελεύθερη θέση, το Issue module μπορεί να στείλει εντολή για αποθήκευση(Instr_valid = ‘1’). Τότε, ο ROB θα επιλέξει την πρώτη διαθέσιμη θέση και θα την καταχωρήσει εκεί. Γνωρίζει ποια είναι αυτή, χρησιμοποιώντας έναν δείκτη(free). Αυτή η θέση θα δεχτεί σήμα ελέγχου ($issue_{θέσης}$ = ‘1’) και θα αποθηκεύσει τα δεδομένα, ενώ ο δείκτης θα προχωρήσει στην επόμενη. Επίσης, στέλνεται το tag της στα RS, το οποίο λειτουργεί σαν το όνομα της θέσης και δείχνει σε ποια αφορούν τα αποτελέσματα. Η τιμή 0 λειτουργεί σαν την άκυρη τιμή. Όσο υπάρχει έγκυρη εντολή, το πεδίο valid της θέσης θα έχει την τιμή ‘1’. Τέλος, αρχικοποιείται τα πεδία done και exception με την τιμή ‘0’.

Η δεύτερη λειτουργία είναι το διάβασμα δεδομένων πηγής κατά το Issue. Είναι δυνατόν να υπάρχουν αποτελέσματα για τους καταχωρητές πηγής(Rj, Rk), τα οποία δεν έχουν ολοκληρωθεί ή αποθηκευτεί. Τότε, πρέπει να υπάρχει μια διαδικασία για να γίνονται αυτά “forward” καθώς τα δεδομένα του αρχείου καταχωρητών είναι απαρχαιομένα. Για να επιτευχθεί αυτό, χρησιμοποιείται λογική στις θέσεις αλλά και στο κεντρικό σύστημα ελέγχου. Το πρώτο βήμα είναι να βρεθούν οι θέσεις με έγκυρες εντολές που γράφουν στους καταχωρητές πηγής ($Ri_{θέσης}$ = Rj, $Ri_{θέσης}$ = Rk). Αυτό γίνεται με συγκριτές στις γραμμές, όπου τα αποτελέσματα τους($j\_flag_{θέσης}$, $k\_flag_{θέσης}$) στέλνονται στο control. Με αυτά και τον δείκτη head, βρίσκει ποια είναι η νεότερη εντολή και κατευθύνει τα αποτελέσματα της (forward_data_j, forward_data_k) προς το Vblock, μαζί με ένα σήμα για να ειδοποιήσει (fw_control_j, fw_control_k) αν είναι έγκυρα ή όχι. Όμως, μαζί με τα δεδομένα, χρειάζεται να σταλθούν και τα αντίστοιχα tags (Qj, Qk) ώστε να ξέρουν τα RS αν οι εντολές τους είναι έτοιμες ή αν πρέπει να περιμένουν νέα αποτελέσματα. Οι θέσεις, για να δείξουν ότι έχουν έγκυρα αποτελέσματα ($done_{θέσης}$ = ‘1’) βγάζουν την τιμή “00000”, αλλιώς βγάζουν το tag τους. Το

control, με τον ίδιο τρόπο όπως με τα δεδομένα, επιλέγει τα κατάλληλα Q. Αν δεν υπάρχει εντολή όπου γράφει στους καταχωρητές πηγής, τα Q παίρνουν την τιμή “00000”. Αυτά στέλνονται κατευθείαν στα RS. Τέλος, αν τα δεδομένα μιας γραμμής κοινοποιηθούν στο CDB την ίδια στιγμή με το διάβασμα τους, αυτά θα γίνουν forward και το tag θα μηδενιστεί.

Η τρίτη λειτουργία είναι το write-back. Όταν κοινοποιούνται έγκυρα αποτελέσματα στο CDB (CDB_valid = ‘1’), πρέπει να αποθηκεύονται στις κατάλληλες θέσεις. Η συγκεκριμένη λειτουργία υλοποιείται εξολοκλήρου σε αυτές, δηλαδή το control δεν συμμετέχει. Αυτές που έχουν έγκυρες εντολές, αν δουν ότι κοινοποιείται το tag τους (CDB_Q = $tag_{θέσης}$), αποθηκεύουν τα αποτελέσματα στο κατάλληλο πεδίο (value) και δείχνουν ότι έχουν ολοκληρωθεί ($done_{θέσης}$ = ‘1’).

Η τέταρτη λειτουργία είναι το commit, δηλαδή η αποθήκευση των αποτελεσμάτων στο αρχείο καταχωρητών. Υλοποιείται κατά κύριο λόγο στο control, αλλά συμμετέχουν και οι θέσεις. Από τα σήματα done που προέρχονται από τις θέσεις, το control γνωρίζει ποιες από αυτές έχουν ολοκληρωμένες εντολές. Αν η θέση που δείχνει ο head έχει έγκυρη και ολοκληρωμένη εντολή, τότε τα δεδομένα της πρέπει να αποθηκευτούν στο αρχείο καταχωρητών. Αυτό γίνεται στέλνοντας στο V_block τα δεδομένα (commit_data) μαζί με ένα σήμα εγκυρότητας (V_commit) καθώς και σε ποιον καταχωρητή πρέπει να αποθηκευτούν (commit_reg). Επίσης, πρέπει να διαγραφεί η εντολή από την θέση. Για αυτό, το control της στέλνει ένα σήμα ($delete_{θέσης}$ = ‘1’) για να την ενημερώσει και αυτή μηδενίζει το πεδίο valid. Είναι η μόνη αλλαγή που χρειάζεται να γίνει, καθώς αυτό το πεδίο συμμετέχει άμεσα στους συγκριτές και έμμεσα σε όλους τους ελέγχους.

Η τελευταία λειτουργία είναι η εξυπηρέτηση των exception, αλλά καμία μονάδα δεν παράγει. Για αυτό δημιουργήθηκε μια δικιά μας συνθήκη, απαγορεύοντας την εγγραφή δεδομένων στον καταχωρητή 31. Αν κάποια εντολή κατά την έκδοση της, προσπαθεί να γράψει σε αυτόν (Ri = “11111”), τότε το πεδίο exception θα πάρει την τιμή ‘1’(αντί του 0 της αρχικοποίησης). Η εντολή θα συνεχίσει κανονικά μέχρι να φτάσει η ώρα του commit. Τότε, το control θα δει ότι η εντολή είναι παράνομη και θα ξεκινήσει την διαδικασία του exception. Καταρχάς, το αποτέλεσμα της δεν θα αποθηκευτεί (V_commit = ‘0’). Επίσης, πρέπει να διαγραφεί, όπως και οι χρονικά επόμενες της. Αφού βρίσκεται σε θέση για commit, σημαίνει ότι είναι η παλαιότερη. Για αυτό, στέλνεται σήμα διαγραφής σε όλες τις θέσεις (delete = “11111111”), καθώς ή έχουν νεότερες εντολές ή δεν έχουν καθόλου. Επίσης, η εντολή που έρχεται στον ίδιο κύκλο ανήκει στο ίδιο πρόγραμμα, είναι επόμενη της και άρα πρέπει να διαγραφεί. Αυτό επιτυγχάνεται ρίχνοντας το σήμα διαθεσιμότητας (rob_available = ‘0’), ώστε να καταλάβει το Issue ότι δεν πρέπει να την δεχτεί. Επειδή οι pointers θα δείχνουν σε λάθος θέσεις, επαναρχικοποιούνται με την τιμή “00000001”. Τέλος, για να ενημερωθεί το front-end του επεξεργαστή ότι συνέβη exception, του στέλνεται ο Program Counter της εντολής που συνέβη. Σε αντίθετη περίπτωση, δέχεται την τιμή “0”.

Η διαδικασία του exception είναι ήδη πολύ ακριβή και μερικοί παραπάνω χαμένοι κύκλοι δεν θα κάνουν μεγάλη διαφορά στις επιδόσεις. Για αυτό επιλέξαμε να γίνεται κατά το commit, ώστε να είναι αρκετά πιο απλή από αν ενεργοποιούταν όταν εμφανιζόταν το exception. Επίσης, για να μειωθεί περισσότερο η πολυπλοκότητα, δεν καθαρίζεται το υπόλοιπο σύστημα από τις εντολές. Αυτές θα ολοκληρωθούν, αλλά δεν θα γίνουν write-back, καθώς δεν υπάρχουν έγκυρες εντολές στην μονάδα. Επίσης, δεν έχει υλοποιηθεί exception handler και θα πρέπει γίνει “trapping”, δηλαδή να καλεστεί μια συνάρτηση (ίσως του λειτουργικού) υπεύθυνη για την διόρθωση του σφάλματος. Για να γίνει αυτό, το IF stage θα πρέπει να τραβήξει εντολές από άλλη περιοχή της μνήμης, να της κάνει decode κλπ. Αυτή η διαδικασία είναι αρκετά αργή και μπορούμε να υποθέσουμε ότι δεν θα έχουν μπει καινούργιες εντολές στο back-end μέχρι αυτό να έχει αδειάσει.

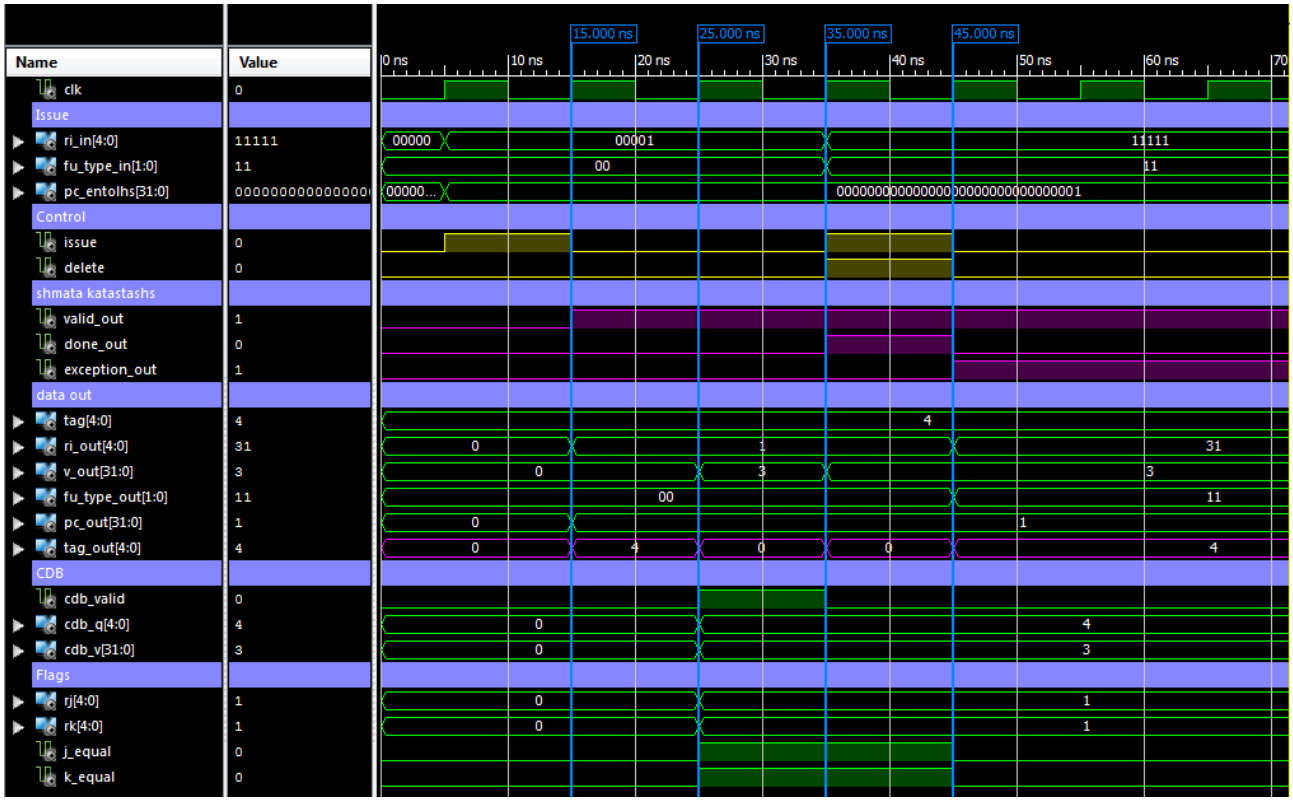
Οι θέσεις του ROB έχουν την ίδια βελτιστοποίηση με αυτές των RS. Όταν διαγράφεται η εντολή που έχουν, μπορούν ταυτόχρονα να εισάγουν καινούργια. Δηλαδή, τα δεδομένα της παλιάς θα πανογραφτούν κατευθείαν με αυτά της καινούργιας, αντί πρώτα να μηδενιστούν και μετά να μπουν τα νέα δεδομένα. Έτσι, επιτυγχάνεται καλύτερο pipe-lining και δεν μένουν πόροι ανεκμετάλλευτοι.

Κυματομορφές

Όλα τα παρακάτω τεστ υπάρχουν σε αρχεία .vhd και οι κυματομορφές σε αρχεία .wcfg με αντίστοιχα ονόματα. Κάποια πεδία αλλάζουν συνέχεια και είναι δύσκολο να φανούν στην αναφορά.

Buffer line test

(buffer_line_test.vhd, buffer_line.wcfg)



15 ns: Έρχεται το πρώτο issue. Στον επόμενο κύκλο, το valid_out γίνεται ‘1’ και έτσι γνωρίζει το σύστημα ότι η θέση έχει έγκυρη εντολή. Επίσης, τα πεδία Ri, PC και fu_type αποθηκεύονται στους καταλλήλους καταχωρητές. Τέλος, το tag_out γίνεται ίσο με το tag, ώστε να γνωρίζουν οι επόμενες εντολές που θέλουν να διαβάσουν το Ri της, από ποια θέση να περιμένουν τα δεδομένα.

25ns: Μέχρι εδώ, δεν έχει έρθει το αποτέλεσμα από τον CDB και όλα τα πεδία παραμένουν ίδια.

35ns: Το CDB κοινοποιεί αποτέλεσμα (cdb_valid = ‘1’) με tag ίσο με αυτό της γραμμής. Βλέπουμε ότι απευθείας το tag_out γίνεται ‘0’, δηλαδή ότι έχει τα σωστά δεδομένα. Επίσης, στον επόμενο κύκλο το done_out γίνεται ‘1’ καθώς το write-back έχει γίνει και η εντολή είναι έτοιμη για commit. Τέλος, επειδή έχουν αλλάξει οι καταχωρητές πηγής (Rj, Rk) και έχουν γίνει ίδιοι με τον αποθηκευμένο καταχωρητή προορισμού τα flags (j_equal, k_equal) έχουν γίνει ‘1’. Έτσι γνωρίζει το σύστημα ότι η θέση έχει κατάλληλα δεδομένα για διάβασμα.

45ns: Έρχεται delete από το control αφού βλέπει ότι done_out = ’1’ και γίνεται το commit. Όμως, την ίδια στιγμή έρχεται και issue = ’1’. Αυτό συμβαίνει γιατί στον κύκλο που διαγράφεται μια εντολή, μπορεί μια καινούργια να μπει στην θέση της. Πράγματι, στον επόμενο κύκλο βλέπουμε ότι μέσα στη θέση υπάρχουν τα δεδομένα της νέας εντολής και τα flags έχουν αλλάξει. Έτσι, φαίνεται ότι οι γραμμές του buffer είναι pipe-lined και δεν μένουν κύκλους ανεκμετάλλευτες. Τέλος, επειδή η νέα εντολή προσπαθεί να γράψει στον καταχωρητή 31 σηκώνεται το σήμα exception_out.

Reorder Buffer test

(reorder_buffer_test.vhd, rob_test.wcfg)

Στο διάστημα 5 - 85ns κάνουμε 8 issue στη σειρά(instr_valid = ‘1’). Από το πεδίο tag_issue φαίνεται σε πια θέση αποθηκεύεται η κάθε εντολή. Βλέπουμε ότι μπαίνουν σειριακά και ότι το σήμα issue ακολουθεί την ίδια πορεία. Επίσης, προχωράει και δείκτης free με κάθε εισαγωγή μέχρι να κάνει ολόκληρο τον κύκλο. Τότε, ο ROB έχει γεμίσει και το δηλώνει κάνοντας το σήμα available μηδέν. Ακόμα, η τέταρτη εντολή μπαίνει με Ri = “31” και το exception_status της θέσης της γίνεται ‘1’. Τέλος, διαβάζουμε συνέχεια τον καταχωρητή 1 (Rj = ‘1’), τον οποίο δεν γράφει καμία εντολή, και ο ROB δεν στέλνει δεδομένα(forward_control_j = ‘0’) η tag(Qj =”0”).

Στο επόμενο διάστημα, 85 – 165ns, ο CDB κοινοποιεί(cdb_valid = ‘1’) τα αποτελέσματα των εντολών με την ανάποδη σειρά από αυτήν που ήρθαν. Αυτό φαίνεται και από τις τιμές του cdb_q. Έτσι, γίνεται το write back και όλες οι θέσεις κάνουν το done τους 1. Μπορεί τα αποτελέσματα τους να έχουν έρθει ,αλλά δεν γίνεται καμία commit γιατί αυτό πρέπει να γίνει με την σειρά με την οποία έγινε το issue. Οι εντολές παραμένουν μέσα και το σήμα available παραμένει 0. Τέλος, διαβάζεται ο καταχωρητής 0 (Rj = ‘0’), τον οποίο γράφει η πρώτη εντολή. Τότε, ανεβαίνει το flag της θέσης της και τα δεδομένα της γίνονται forward (forward_control_j = ‘1’). Όμως, μέχρι τα 155ns δεν έχουν έρθει τα αποτελέσματα της από τον CDB και τα δεδομένα της είναι λάθος. Για αυτό, στέλνει το tag της (Qj = “1”) μέχρι αυτά να έρθουν.

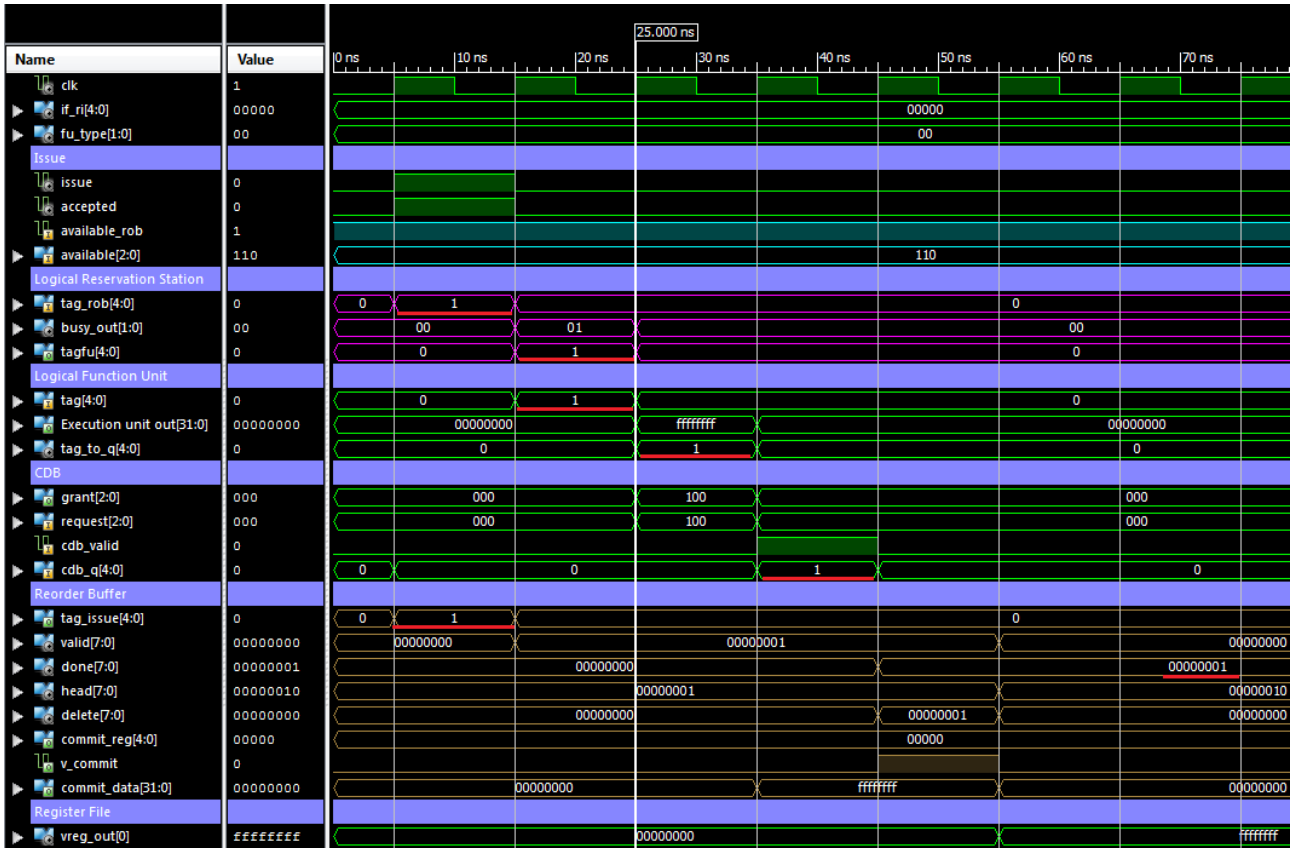
Στο τελευταίο διάστημα, 165 – 195ns, η παλαιότερη εντολή έχει ολοκληρωθεί και αρχίζουν τα commit. Βλέπουμε ότι κατευθείαν γίνεται το available 1 και μπαίνει άλλη εντολή στη θέση της(instr_valid = ‘1’). Σε κάθε κύκλο η θέση που δείχνει ο head γίνεται commit και αυτός προχωράει. Με το σήμα V_commit ενεργοποιημένο, θα γραφτεί ο καταχωρητής όπου υποδεικνύει το σήμα commit_reg. Επίσης, η θέση όπου ήταν η εντολή αποθηκευμένη αδειάζει κάνοντας το αντίστοιχο bit του delete 1. Τότε, το αντίστοιχο bit από το valid θα γίνει 0. Όμως, στο τέταρτο commit έρχεται η σειρά της εντολής με το exception. Τα αποτελέσματα της δεν θα αποθηκευτούν (V_commit = ‘0’) και το πρόγραμμα πρέπει να σταματήσει. Αυτό γίνεται κάνοντας όλα τα bit του delete 1 και σαν αποτέλεσμα όλα τα πεδία valid είναι μηδέν. Η εντολή που έρχεται σε αυτόν τον κύκλο, είναι επόμενη της και πρέπει να μπλοκαριστεί, το οποίο γίνεται ρίχνοντας το

σήμα available. Οι δείκτες free και head επαναρχικοποιούνται στην πρώτη θέση. Έτσι, όλες οι εντολές έχουν διαγραφεί και ο επεξεργαστής είναι έτοιμος να δεχτεί τον exception handler. Τέλος στέλνεται ο PC της εντολής στο IF stage για να ξέρει από που να συνεχίσει.

Back-end test

(back end_test.vhd, mia_entolh_xronismos.wcfg)

Στο παρακάτω τεστ ελέγχουμε την πορεία μια εντολής στο συνολικό σύστημα για να παρατηρήσουμε τον χρονισμό του συνολικού συστήματος.



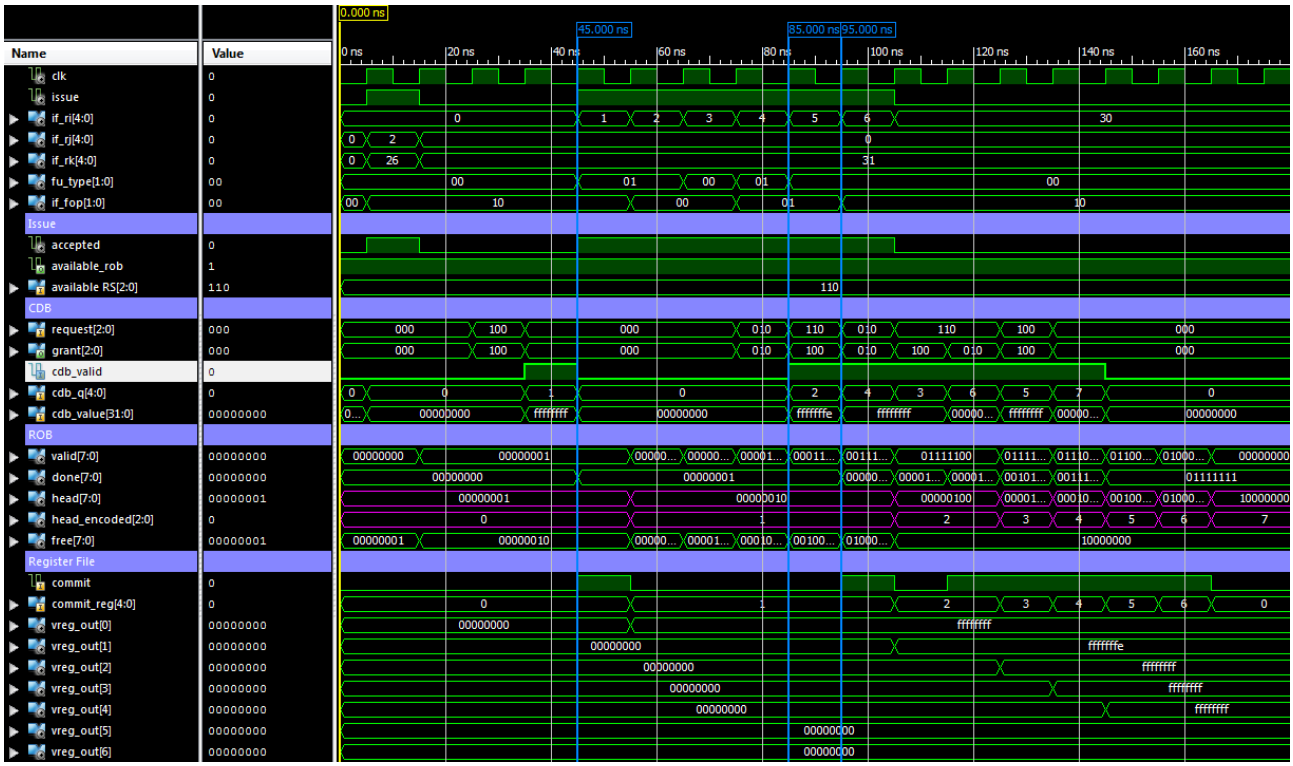
Γίνεται issue εντολής και οι μονάδες που χρειάζεται είναι διαθέσιμες(available_rob = ‘1’, available(2) = ‘1’ με fu_type = “00”). Απευθείας, το ROB στέλνει το tag της θέσης που αποθηκεύει την εντολή και αυτό πάει στο RS. Επειδή η εντολή είχε έγκυρα δεδομένα, με το που αποθηκευτεί στο RS στέλνεται στο Fu και την διαγράφει. Αυτό φαίνεται από το πεδίο busy_out όπου γίνεται “00” με το που στέλνεται η εντολή. Επειδή η εντολή είναι λογική και έχει καθυστέρηση 2 κύκλων, το FU ζητάει request όταν κάνει την πράξη. Αφού δέχεται θετικό grant, στον επόμενο κύκλο(45 ns) θα κοινοποιήσει τα δεδομένα στο CDB. Ο ROB αφού βλέπει ότι τα δεδομένα είναι έγκυρα (cdb_valid = ‘1’) θα αποθηκεύσει τα αποτελέσματα στη θέση με το tag που κοινοποιείται(cdb_q). Στον επόμενο κύκλο(55ns) από το write_back, ο ROB βλέπει ότι η εντολή είναι έτοιμη και ότι είναι αυτή που δείχνει ο δείκτης head και αποφασίζει να την κάνει commit. Η αποθήκευση των αποτελεσμάτων γίνεται ενεργοποιώντας το σήμα v_commit και το σήμα commit_reg όπου δείχνει σε ποιον καταχωρητή να γραφτούν τα δεδομένα του commit_data. Επίσης, η εντολή διαγράφεται από τον ROB με το σήμα delete στο κατάλληλο bit. Πράγματι, το valid bit της πρώτης θέσης γίνεται 0. Τέλος, βλέπουμε ότι τα αποτελέσματα που δημιούργησε το FU αποθηκεύτηκαν στον V_reg που έδειξε ο Ri.

Σημείωση: Έχει υπογραμμιστεί με κόκκινο χρώμα η πορεία του tag μέσα στο σύστημα.

Στα παρακάτω τεστ μελετάμε σύνθετες και ακραίες περιπτώσεις. Επίσης, ο καταχωρητής 0 αρχικοποιείται με μια not ώστε να υπάρχει κάποια διαφορετική τιμή και να φαίνονται καλύτερα τα αποτελέσματα. Αφού τελειώσει η εντολή ξεκινάει κανονικά το test.(5 - 45ns)

Τυχαίες εντολές χωρίς εξαρτήσεις

(tuxaies_entoles_xwris_eksarthseis.vhd, tuxaies_entoles.wcfg)

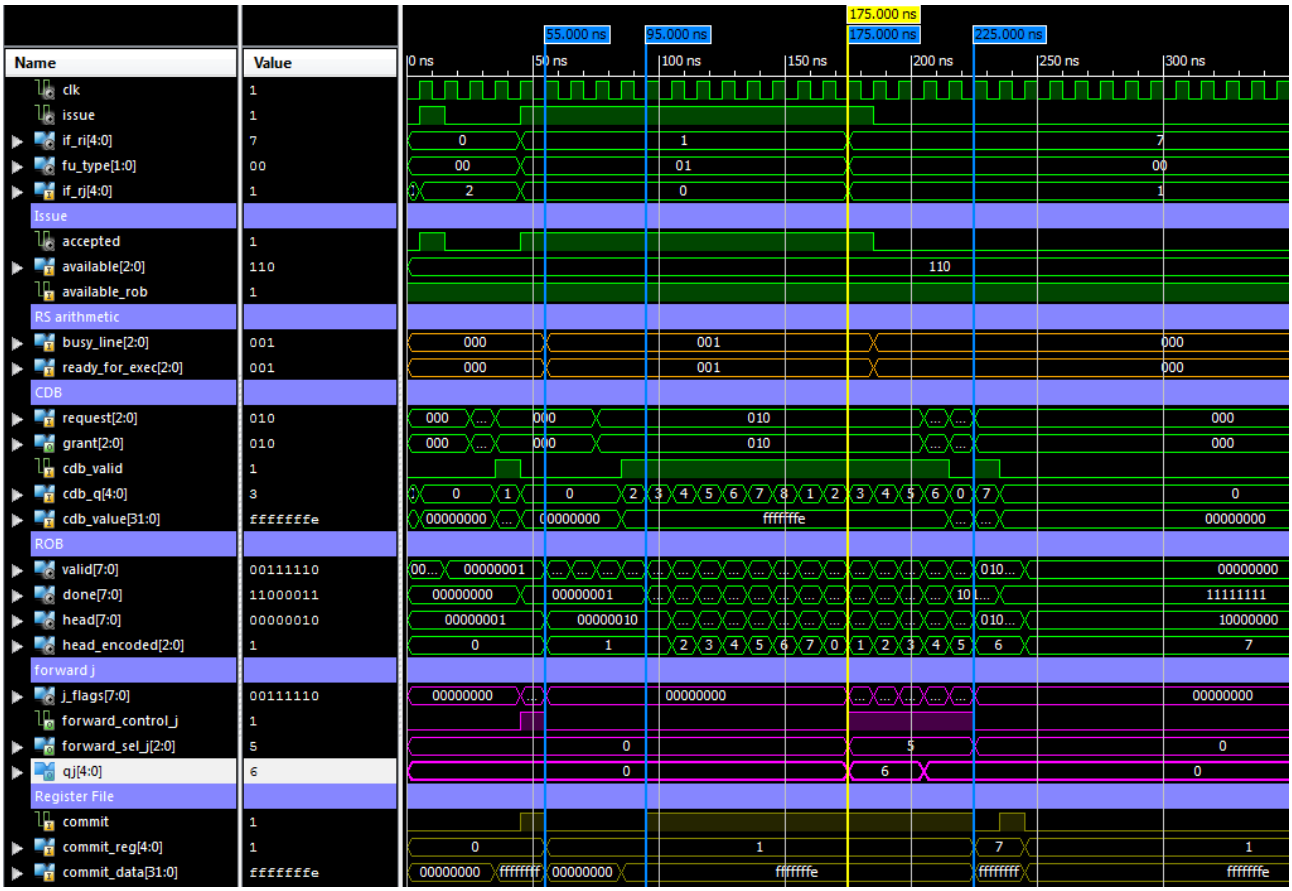


Στο παραπάνω τεστ, θέλουμε να δείξουμε ότι λόγω των συγκρούσεων του CDB τα write-back δεν γίνονται με την σειρά των Issue, αλλά τα commit θα γίνουν με αυτήν. Στα 45 ns ξεκινάνε τα Issue των 6 εντολών του προγράμματος. Η πρώτη εντολή γράφει τον καταχωρητή 1 και θα μπει στην δεύτερη θέση του ROB, η δεύτερη στον καταχωρητή 2 και την τρίτη θέση κλπ (Μπαίνουν σε +1 θέση γιατί η εντολή αρχικοποίησης έχει προχωρήσει τους pointers του ROB.). Οι εντολές απασχολούν και τα δύο RS και δεν έχουν εξαρτήσεις μεταξύ τους. Κοιτώντας τα πεδία request και grant του CDB παρατηρούμε ότι γίνονται συγκρούσεις. Έτσι οι κοινοποίησης(85 - 135 ns) δεν γίνονται με την σειρά και αυτό φαίνεται από το cdb_q. Επειδή commit γίνεται μόνο στη θέση που υποδεικνύει ο head, μπορεί να έχουν έρθει τα αποτελέσματα των επόμενων πράξεων, αλλά δεν θα γίνουν commit. Για αυτόν τον λόγο υπάρχει και μια μικρή καθυστέρηση σε αυτά. Όμως αυτά γίνονται με την σωστή σειρά, το οποίο από το commit_reg.

Σημείωση: Υπάρχει μια μικρή αναντιστοιχία μεταξύ τιμών head και tags γιατί ο head δείχνει θέσεις ενώ τα tags τα "ονόματα" των θέσεων. Η θέση 0 έχει tag 1 κλπ.

Εντολές σε μια μονάδα και εξαρτήσεις WAW

(entoles_se_mia_monada.vhd, idia_monada.wcfg)



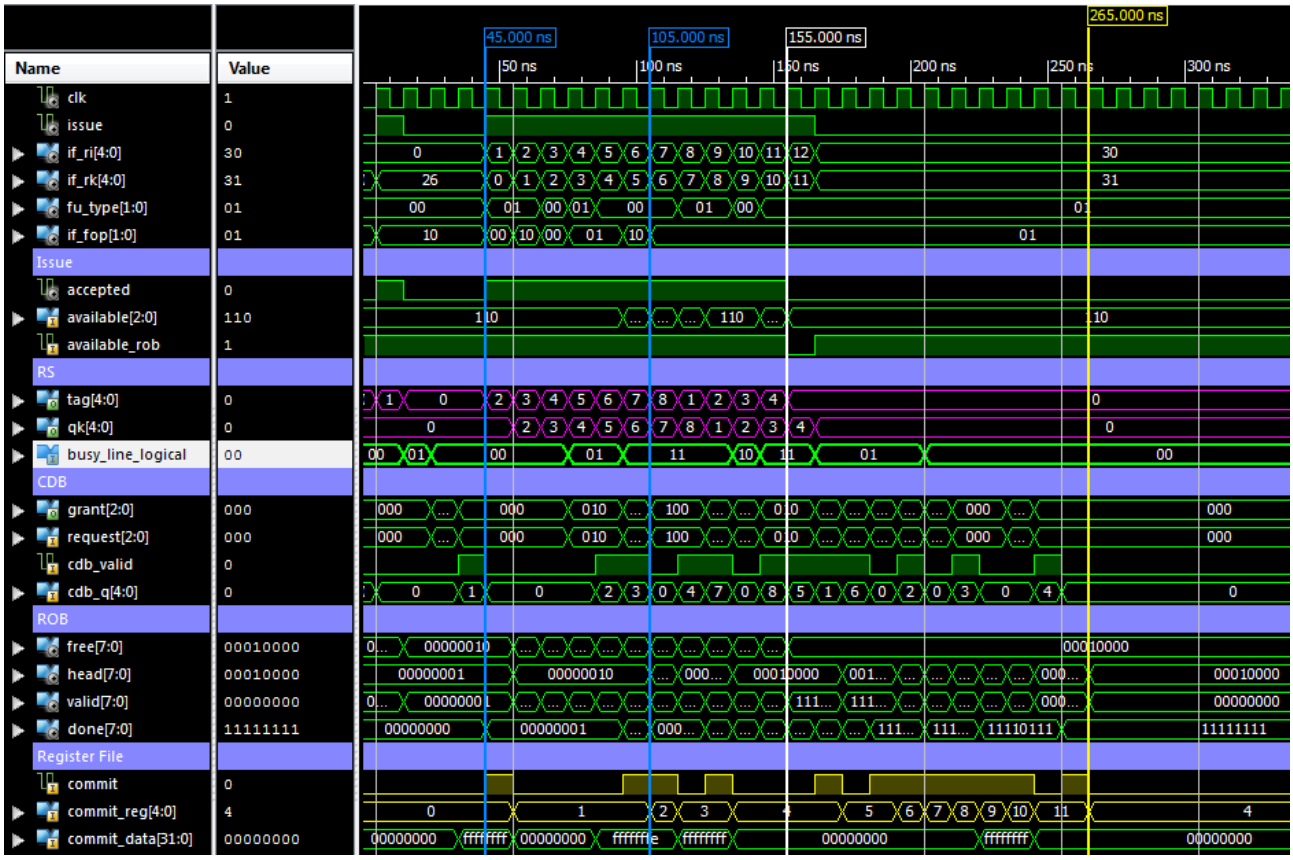
Σε αυτό το τεστ, εκτελούνται συνέχεια εντολές ίδιου τύπου. Στην υλοποίηση του προηγούμενου εργαστηρίου, η συγκεκριμένη περίπτωση προκαλούσε πρόβλημα καθώς ο RS γέμιζε και μερικές εντολές δεν γίνονταν δεκτές. Αντίθετα, τώρα όλες μπαίνουν μέσα στο σύστημα σύμφωνα με το πεδίο accepted. Επειδή οι εντολές διαγράφονται από τα RS με το στέλνονται για εκτέλεση, αυτά δεν αποτελούν πια bottleneck. Αυτό φαίνεται από το πεδίο busy_line του RS όπου δείχνει ότι απασχολείται μόνο μια γραμμή του.

Επίσης, όλες οι εντολές, εκτός από την τελευταία, γράφουν τον ίδιο καταχωρητή (if_ri = “1”). Δηλαδή υπάρχει εξάρτηση WAW. Αυτό δεν αποτελεί πρόβλημα καθώς το renaming του ROB τις αντιμετωπίζει. Όλες οι εντολές γίνονται commit με την σειρά που ήρθαν πάνω στον ίδιο καταχωρητή. Η τελευταία εντολή (175 ns) είναι διαφορετικού τύπου και έχει ως καταχωρητή πηγής αυτόν που γράφουν οι άλλες. Τότε, ο ROB πρέπει να της δώσει τα δεδομένα της πιο πρόσφατης. Από το πεδίο j_flags φαίνεται ποιες θέσεις έχουν Ri ίδιο με το Rj της νέας εντολής και από το πεδίο forward_sel_j πια από αυτές διάλεξε το control του CDB. Μέχρι αυτή η θέση να πάρει τα αποτελέσματα που περιμένει, θα βγάζει σαν Q το tag της. Όταν αυτά έρθουν, το Q θα γίνει 0. Όταν γίνει commit(225 ns), τα δεδομένα θα διαβάζονται από το αρχείο καταχωρητών (forward_control_j = ‘0’).

Τέλος, κοιτώντας το πεδίο busy_line του RS φαίνεται το optimization στις θέσεις. Μόνο μια γραμμή απασχολείται γιατί φεύγουν συνέχεια εντολές και ταυτόχρονα μπαίνουν οι επόμενες στην ίδια θέση. Στο συγκεκριμένο τεστ δεν βοηθάει ιδιαίτερα, αλλά σε άλλα, όπως στο επόμενο, που είναι πιο resource-intensive για τα RS βοηθάει πολύ σημαντικά.

Εντολές με εξαρτήσεις RAW

(tuxaies_entoles_me_eksartiseis.vhd, entoles_me_eksartiseis.wcfg)



Στο παραπάνω τεστ, βάζουμε στο σύστημα δώδεκα εντολές όπου η κάθε μια εξαρτάται από την προηγούμενη. Όλες οι εξαρτήσεις είναι RAW. Ο σκοπός του είναι να φτάσουμε το ROB στα όρια του και να γεμίσει. Αυτό συμβαίνει στα 155ns όπου η τελευταία εντολή απορρίπτεται(accepted = ‘0’) γιατί το σήμα available_rob είναι 0. Από τα πεδία tag και qk φαίνεται ποια εντολή περιμένει η κάθε μια. Επίσης, μερικές εντολές είναι not και shift όπου αδιαφορούν για τον δεύτερο τελεστή(Rk). Έτσι, αυτές οι εντολές δεν περιμένουν τις προηγούμενες, στέλνονται κατευθείαν για εκτέλεση με αποτέλεσμα να μην γίνεται το write-back με την σειρά. Όμως, το commit συνεχίζει να γίνεται με την σειρά, δηλαδή είναι ανεξάρτητο των εξαρτήσεων. Τελικά, η λειτουργικά του συστήματος είναι ορθή.

Στην προηγούμενη υλοποίηση με το ίδιο τεστ οι εντολές έμεναν μέσα στα RS μέχρι να γυρίσουν τα αποτελέσματα τους. Αυτά γέμιζαν πολύ γρήγορα και έμπαιναν λίγες εντολές στο σύστημα. Αντίθετα, τώρα μέσα στα RS υπάρχουν μόνο εντολές που περιμένουν άλλες με αποτέλεσμα να μειώνεται η συμφόρηση τους. Έτσι, δέχεται 11 εντολές στις 12 αντί 5 στις 6, μέχρι να απορριφθεί μία. Η πιθανότητα σε ένα πρόγραμμα να υπάρχουν 12 συνεχόμενα εξαρτώμενες εντολές είναι πολύ πιο μικρή από 6 και η βελτίωση είναι πολύ σημαντική.

Στα 105 ns φαίνεται το optimization που δείξαμε στο προηγούμενο τεστ. Και οι δύο θέσεις του λογικού RS είναι απασχολημένες (busy_line_logical = “11”), μπαίνει λογική εντολή (fu_type = “00”) και η εντολή γίνεται δεκτή (issue = ‘1’). Αυτό δεν θα γινόταν εφικτό χωρίς το optimization.

Τέλος, κοιτώντας το πεδίο done στο τέλος του τεστ, παρατηρούμε ότι έχει την τιμή 1 για όλες τις θέσεις. Αυτό είναι αδιάφορο, καθώς το ισχυρότερο πεδίο valid έχει την τιμή 0. Αν το valid έχει αυτήν τη τιμή, όλα τα υπόλοιπα πεδία, θεωρούνται σκουπίδια.

Exception

(exception_back_end_test.vhd, exception.wcfg)

Σε αυτό το τεστ, εισάγονται 6 εντολές όπου η τέταρτη παραβιάζει την συνθήκη του exception γράφοντας τον καταχωρητή 31. Το πρόγραμμα λειτουργεί κανονικά μέχρι τη στιγμή που γίνεται commit αυτή η εντολή. Τότε, θα γίνει η αντιμετώπιση του exception αντί commit. Πρώτα από όλα, τα αποτελέσματα της απορρίπτονται. Επίσης, διαγράφονται όλες οι εντολές μέσα στον ROB το οποίο φαίνεται από το πεδίο delete. Γίνεται ένα write-back μετά την διαγραφή, αλλά δεν θα ολοκληρωθεί λόγο της διαγραφής της εντολής του. Επειδή η εντολή που μπαίνει την ίδια στιγμή είναι του ίδιου προγράμματος και επόμενη της, πρέπει να σκοτωθεί. Αυτό γίνεται απενεργοποιώντας το σήμα available_rob. Ακόμα, στέλνεται το PC της εντολής προς το IF stage για να ξέρει από που να συνεχίσει. Τέλος, επαναρχικοποιούνται οι pointers στην αρχική τους κατάσταση.

Στρατηγική και αποτελέσματα του ελέγχου

Η στρατηγική του ελέγχου εγκυρότητας είναι παρόμοια με αυτήν του προηγούμενου εργαστηρίου, αλλά έχει προσαρμοστεί ώστε να ακολουθεί την πορεία υλοποίησης του παρόν εργαστηρίου. Καταρχάς, μερικές μονάδες (FUs, CDB) έχουν μείνει απαράλλαχτες σε σχέση με την προηγούμενη φάση. Έτσι, δεν χρειάζονται καινούργια unit tests καθώς καλύπτονται από τα είδη υπάρχων. Για άλλες μονάδες (Issue unit, RSs) όπου διαφέρουν ελαφρά, τα unit tests τους άλλαξαν ώστε να καλύπτουν τις καινούργιες εισόδους και εξόδους καθώς και να επικεντρώνονται στους καινούργιους χρονισμούς. Εφόσον μπορούμε να εγγυηθούμε ότι οι υπόλοιπες μονάδες λειτουργούν ορθά, είναι δυνατόν να προχωρήσουμε στην υλοποίηση και έλεγχο του Reorder Buffer και του ολοκληρωμένου συστήματος.

Ο έλεγχος του ROB ξεκινάει από τον έλεγχο των υπο-μονάδων του. Επειδή οι θέσεις του buffer είναι υλοποιημένες σε ξεχωριστό source file το οποίο γίνεται component σε αυτόν, είναι δυνατό να τεστάρουμε μια απομονωμένη. Σε αυτό το στάδιο, μεγαλύτερη έμφαση δίνεται στα πεδία ελέγχου (valid, done) και στα flags που παράγουν οι εσωτερικοί συγκριτές. Το πλεονέκτημα που έχει το συγκεκριμένο τεστ είναι ότι μπορεί να επικεντρωθεί στα παραπάνω πεδία, χωρίς να μας επηρεάζει το control του ROB και οι επιλογές του. Τέλος, κάποια σύνθετα μέρη του control (πχ. Forward) έχουν δοκιμαστεί μεμονωμένα λόγω τις υψηλής πολυπλοκότητας τους.

Αφού οι υπο-μονάδες φαίνεται να λειτουργούν σωστά, το επόμενο βήμα είναι να ελεγχθεί ολόκληρος ο ROB. Στόχος του συγκεκριμένου ελέγχου είναι να δείξει τις πέντε λειτουργίες του (issue, commit, write-

back, forward, exception) και την επίδραση που έχει η κάθε μία στις υπόλοιπες. Υψηλή σημασία δίδεται πως το control διαχειρίζεται τα flags και πεδία ελέγχου των θέσεων, στους δείκτες head και free˙ και τα σήματα ελέγχου που δημιουργεί. Πέρα από την ορθή λειτουργία του, σε αυτό το τεστ φαίνεται και αν η μονάδα ικανοποιεί pipe-line αρχιτεκτονική.

Εφόσον όλα τα unit τεστ δείχνουν ότι οι μονάδες λειτουργούν όπως έχουν σχεδιαστεί, μπορούμε να προχωρήσουμε στα integrity τεστ. Στο πρώτο από αυτά εισάγουμε μόνο μια εντολή, με σκοπό να δούμε ότι οι μονάδες συνεργάζονται και επικοινωνούν όπως έχουμε προβλέψει. Σε αυτό το σημείο επικεντρωνόμαστε στα μεταξύ τους σήματα, την πορεία της εντολής μέσα στο σύστημα καθώς και τον χρονισμό της. Εάν το συγκεκριμένο τεστ είναι σωστό, μπορούμε να προχωρήσουμε σε πιο σύνθετα τεστ δίνοντας λιγότερη σημασία στα παραπάνω.

Το τελευταίο κομμάτι του ελέγχου αφορά ακραίες καταστάσεις στις οποίες μπορεί να βρεθεί το συνολικό σύστημα. Χρησιμοποιώντας 4 μικρά προγράμματα, 6 – 12 εντολών, με προμελετημένες εξαρτήσεις και συγκρούσεις, απομονώνονται οι σημαντικότερες και είναι ευδιάκριτες. Σε αυτά, θεωρούνται δεδομένοι οι χρονισμοί μιας εντολής και η επικοινωνία μεταξύ των μονάδων. Μελετάται η πορεία των εντολών συναρτήσει εξαρτήσεων από άλλες, RAW ή WAW, καθώς και δομικών hazards όπου προκύπτουν επειδή οι αποθηκευτικοί πόροι του συστήματος είναι πεπερασμένοι και εξαντλούνται. Επίσης, μας ενδιαφέρει να δείξουμε ότι η λειτουργικότητα του συστήματος είναι σύμφωνη με την θεωρία. Τέλος, τα τεστ συγκρίνονται με αυτά της προηγούμενης υλοποίησης και δίνεται παραπάνω σημασία σε αντικείμενα και καταστάσεις που δεν είχε αυτή(πχ. Exception).

Γενικά, τα τεστ είναι αντιπροσωπευτικά και καλύπτουν ένα μεγάλο εύρος περιπτώσεων, αλλά δεν είναι εξαντλητικά και δεν επικεντρώνονται στα αποτελέσματα. Τα συμπεράσματα του ελέγχου είναι ότι το σύστημα είναι πλήρως λειτουργικό και αντιδράει ορθά σε ακραίες περιπτώσεις. Επίσης, είναι pipe-lined σε όλα τα στάδια, αφού δεν μένουν μονάδες ανεκμετάλλευτες εάν υπάρχει διαθέσιμη δουλειά για αυτές.

Σημειώσεις

- Μέσα στον φάκελο με τον κώδικα υπάρχουν αρχεία .vhd και .wcfg για τα test που παρουσιάζουμε και προτείνουμε να τα δείτε από εκεί. Πεδία όπως οι pointers και τα σήματα ελέγχου των θέσεων του ROB, ενώ αλλάζουν συνέχεια, δεν αναπαρίστανται με αριθμούς. Σαν αποτέλεσμα, είναι δύσκολο να παρουσιαστούν στο ίδιο επίπεδο μεγέθυνσης με τα υπόλοιπα.

- Σε δικούς τους φακέλους, επισυνάπτονται οι κυματομορφές και τα σχεδιαγράμματα.

- Η βελτιστοποίηση που αφορά τις θέσεις του RS και του ROB έχει υλοποιηθεί και στα προηγούμενα εργαστήρια (για το RS). Δεν την είχαμε περιγράψει σε βάθος στις άλλες αναφορές γιατί πιστεύαμε ότι είναι αναγκαίο για να θεωρείτε το σύστημα pipe-lined, αλλά ο καθηγητής του μαθήματος μας είπε ότι δεν είναι υποχρεωτικό να υπάρχει.

Github Link