ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΗΡΥ 415 - ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2020-21

Αναφορά Εργαστηρίου 2: Υλοποίηση αλγορίθμου Tomasulo σε VHDL #2

Ονοματεπώνυμο: Γεώργιος Αγορίτσης
Α.Μ.: xxxxxxxxxx

Σκοπός Εργαστηρίου 2:

Ο σκοπός του εργαστηρίου 2 είναι η ολοκλήρωση της σχεδίασης των πέντε επιμέρους μονάδων που υλοποιούν τον αλγόριθμο Tomasulo σε VHDL και ο έλεγχος ορθής εκτέλεσης εντολών από το συνολικό σύστημα. Συγκεκριμένα, δημιουργούμε ένα testbench, με το οποίο θα ελέγξουμε την εκτέλεση διαφόρων εντολών και τον τρόπο λειτουργίας του συστήματος για διάφορα corner cases.

Περιγραφή της μεθοδολογίας ανάπτυξης των ζητουμένων:

1. **Αλλαγές και βελτιώσεις**
   - Ο καταχωρητής 0 αρχικοποιείται πλέον στην τιμή x"00000002" για να χρησιμοποιηθεί σε λογικές και αριθμητικές πράξεις (κάνουμε χρήση μίας παραλλαγής του RF_register.vhd, τον RF_register_value1.vhd, μόνο για το συγκεκριμένο καταχωρητή της RF).
   - Το control module των RS και FU (CTRL_RS_FU.vhd) βελτιώθηκε ώστε να υποστηρίζεται σωστά η λογική των request. Συγκεκριμένα, η βελτίωση αφορά την περίπτωση στην οποία έχουμε ταυτόχρονα requests τόσο από τη λογική λειτουργική μονάδα, όσο και από την αριθμητική λειτουργική μονάδα.
   - Το σήμα WrEn της RF συνδέθηκε με την έξοδο της λογικής πύλης (issue_log OR issue_ar), αντί να συνδέεται απευθείας με το σήμα Issue (από IF Unit), δηλαδή έχουμε WrEn στην RF όταν το σύστημα έχει αποδεχτεί την εντολή προς έκδοση.
   - Στον RS (RS.vhd) προστέθηκε το σήμα εξόδου Avail_to_Issue. Το σήμα αυτό αντιστοιχεί στην είσοδο του Available register που υπάρχει στον RS και δείχνει αν ο συγκεκριμένος RS είναι διαθέσιμος. Στην FU1 (FU1.vhd) έγινε προσθήκη του σήματος εξόδου Avail_to_Issue(2 downto 0) (MSB: σήμα Available του RS3, LSB: σήμα Available του RS1), ενώ στην FU2 (FU2.vhd) έγινε προσθήκη του σήματος εξόδου Avail_to_Issue(1 downto 0) (MSB: σήμα Available του RS2, LSB: σήμα Available του RS1). Επιπλέον, έγιναν οι κατάλληλες ενημερώσεις των σημάτων στο top module TOMASULO_PROC.vhd.
   - Στον RS (RS.vhd) προστέθηκε το σήμα εισόδου RS_tag το οποίο συνδέεται με το αντίστοιχο σήμα εξόδου του Issue Unit. Το σήμα RS_tag γίνεται είσοδος σε ένα καταχωρητή, για τον

οποίο ενεργοποιούμε το write enable μόνο όταν εκδίδεται μία εντολή στο συγκεκριμένο RS. Χρησιμοποιούμε την έξοδο του καταχωρητή αυτού και την είσοδο CDB_Q του RS για να ελέγξουμε τα σήματα busy και available του RS. Συγκεκριμένα, ο RS παύει να είναι busy και γίνεται available όταν η έξοδος του καταχωρητή που αναφέραμε είναι ίση με το CDB_Q που έχουμε ως είσοδο. Σε αυτή την περίπτωση ο RS μπορεί να χρησιμοποιηθεί για να εκδοθεί νέα εντολή, ενημερώνοντας κατάλληλα το RS_tag του και περιμένοντας εκ νέου ταύτιση με το CDB_Q για να γίνει ξανά διαθέσιμος προς χρήση από άλλη εντολή που πρόκειται να εκδοθεί. Επιπλέον, εισήγαμε το σήμα RS_tag, τόσο στα FU1.vhd και FU2.vhd, όσο και στο top module TOMASULO_PROC.vhd.

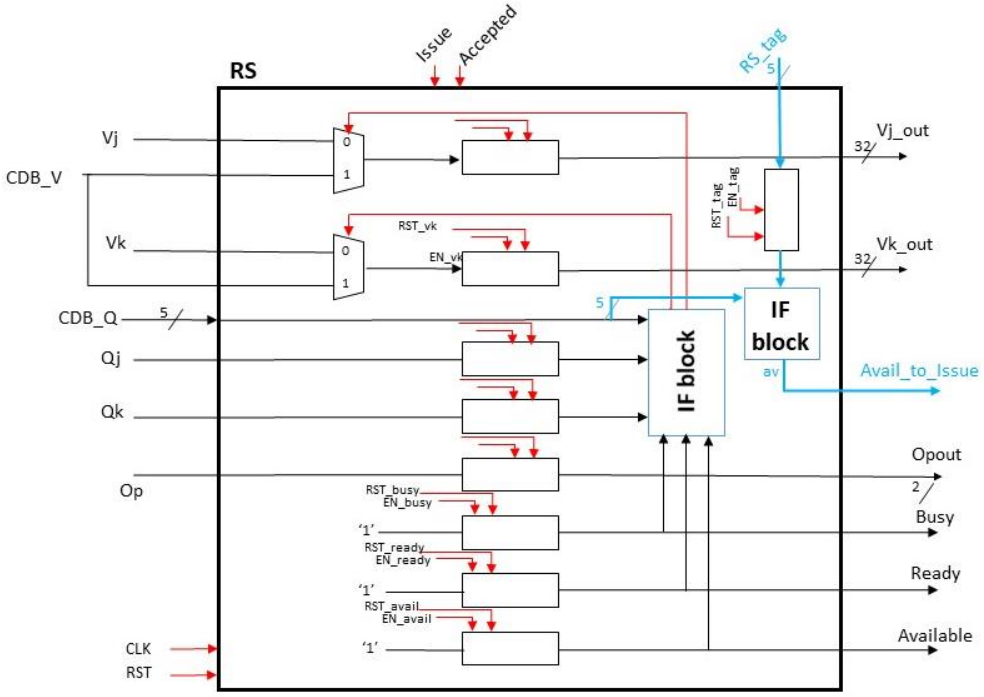
Από τις τελευταίες δύο αλλαγές επηρεάστηκαν τα σχηματικά διαγράμματα των RS, FU1, FU2 και CTRL_RS_FU. Παρακάτω παραθέτουμε εκ νέου τους πίνακες διεπαφών και τα αντίστοιχα σχηματικά τους. Οι αλλαγές φαίνονται με έντονο μπλε χρώμα στα σήματα και στα ονόματα τους. Τα διαγράμματα χρονισμού παραλείπονται, καθώς δεν υπάρχουν ουσιαστικές αλλαγές στη λειτουργία του συστήματος. Οι μόνες διαφορές θα είναι ότι **1)** το σήμα Avail_to_Issue ενώ θα είναι ίδιο με το σήμα Available, θα ενημερώνεται στην αρχή του κύκλου του ρολογιού και όχι στη θετική ακμή του ρολογιού, όπως συμβαίνει με το σήμα Available και **2)** τα σήματα Busy και Available κάθε RS (συνεπώς και το Avail_to_Issue) θα διατηρούν τις τιμές τους έως ότου η εντολή που εκδόθηκε στο συγκεκριμένο RS ολοκληρωθεί και το αποτέλεσμα της εκδοθεί στο CDB.

Οι πίνακες διεπαφών και τα σχηματικά διαγράμματα των επιμέρους μονάδων που επηρεάζονται αλλάζουν ως εξής:

Reservation Station (RS)

Σήμα	Πλάτος	Είδος	Περιγραφή
CLK	1 bit	Είσοδος	Σήμα Clock
RST	1 bit	Είσοδος	Σήμα Reset
ISSUE	1 bit	Είσοδος	Εκδίδεται εντολή
Accepted	1 bit	Είσοδος	Η εντολή έγινε δεκτή από μία FU
Op	2 bit	Είσοδος	Κωδικός πράξης προς αποθήκευση στον RS
CDB_Q	5 bit	Είσοδος	Tag των δεδομένων που βρίσκονται στο CDB
CDB_V	32 bit	Είσοδος	Δεδομένα που βρίσκονται στο CDB
Vj	32 bit	Είσοδος	Δεδομένα από RF προς αποθήκευση στον RS
Vk	32 bit	Είσοδος	Δεδομένα από RF προς αποθήκευση στον RS
Qj	5 bit	Είσοδος	Tag από RF προς αποθήκευση στον RS
QK	5 bit	Είσοδος	Tag από RF προς αποθήκευση στον RS
RS_tag	5 bit	Είσοδος	RS_tag από ISSUE_UNIT
Opout	2 bit	Έξοδος	Κωδικός πράξης προς FU όταν η εντολή γίνει δεκτή
Busy	1 bit	Έξοδος	RS απασχολημένος
Ready	1 bit	Έξοδος	RS έτοιμος για να εξυπηρετηθεί από την FU
Available	1 bit	Έξοδος	RS διαθέσιμος να δεχτεί δεδομένα
Avail_to_Issue	1 bit	Έξοδος	Σήμα «RS διαθέσιμος να δεχτεί δεδομένα» προς Issue Unit
Vjout	32 bit	Έξοδος	Δεδομένα προς FU
Vkout	32 bit	Έξοδος	Δεδομένα προς FU

Το σχηματικό διάγραμμα είναι:

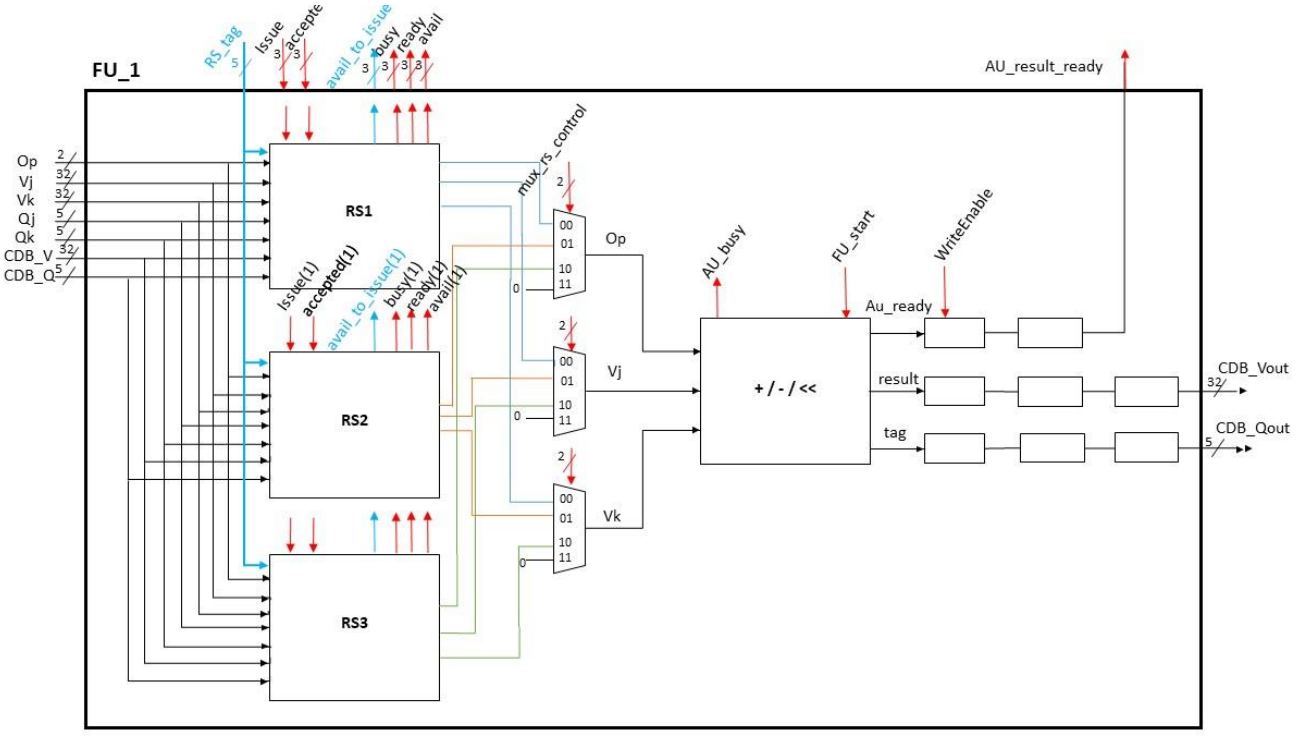


Arithmetic Functional Unit (FU_1)

Σήμα	Πλάτος	Είδος	Περιγραφή
CLK	1 bit	Είσοδος	Σήμα Clock
RST	1 bit	Είσοδος	Σήμα Reset
WriteEnable	1 bit	Είσοδος	Σήμα Write Enable για τους pipelined καταχωρητές στην έξοδο.
Op	2 bit	Είσοδος	Κωδικός πράξης προς αποθήκευση στους RS του FU
CDB_Q	5 bit	Είσοδος	Tag των δεδομένων που βρίσκονται στο CDB
CDB_V	32 bit	Είσοδος	Δεδομένα που βρίσκονται στο CDB
Vj	32 bit	Είσοδος	Δεδομένα από RF προς αποθήκευση στους RS του FU
Vk	32 bit	Είσοδος	Δεδομένα από RF προς αποθήκευση στους RS του FU
Qj	5 bit	Είσοδος	Tag από RF προς αποθήκευση στους RS του FU
QK	5 bit	Είσοδος	Tag από RF προς αποθήκευση στους RS του FU
mux_rs_control	2 bit	Είσοδος	Έλεγχος πολυπλέκτη που κατευθύνει τα δεδομένα Op, Vj, Vk στην FU. 00: RS1 σε FU, 01: RS2 σε FU, 10: RS3 σε FU, 11: zero values to FU
Accepted	3 bit	Είσοδος	000: κανένα RS δεν έγινε δεκτό για να εκτελεστεί πράξη 001/010/100: το RS1/RS2/RS3 έγινε δεκτό για τη αριθμητική πράξη 011/101/110/111: δεν ορίζεται
Issue	3 bit	Είσοδος	000: δεν εκδίδεται κάποια εντολή σε κανένα RS 001/010/100: εκδίδεται εντολή στο RS1/RS2/RS3 011/101/110/111: δεν ορίζεται
FU_start	1 bit	Είσοδος	Σήμα εκκίνησης εκτέλεσης λογικής πράξης στην FU

RS_tag	5 bit	Είσοδος	RS_tag από ISSUE_UNIT
Busy	3 bit	Έξοδος	000: κανένας RS απασχολημένος 001/010/100: RS1/RS2/RS3 απασχολημένος 011/101/110: RS2 & RS1/RS3 & RS1/RS3 & RS2 απασχολημένοι 111: RS3 & RS2 & RS1 απασχολημένοι
Ready	3 bit	Έξοδος	000: κανένας RS με έγκυρα δεδομένα 001/010/100: RS1/RS2/RS3 με έγκυρα δεδομένα 011/101/110: RS2 & RS1/RS3 & RS1/RS3 & RS2 με έγκυρα δεδομένα 111: RS3 & RS2 & RS1 με έγκυρα δεδομένα
Available	3 bit	Έξοδος	000: κανένας RS διαθέσιμος να δεχτεί δεδομένα 001/010/100: RS1/RS2/RS3 διαθέσιμος να δεχτεί δεδομένα 011/101/110: RS2 & RS1/RS3 & RS1/RS3 & RS2 διαθέσιμοι να δεχτούν δεδομένα 111: RS3 & RS2 & RS1 διαθέσιμοι να δεχτούν δεδομένα
Avail_to_Issue	3 bit	Έξοδος	Ακολουθεί την ίδια λογική με το σήμα Available. Η μόνη διαφορά είναι ότι το σήμα ενημερώνεται στην αρχή του κύκλου και δεν δημιουργεί προβλήματα χρονισμού με το Issue Unit.
CDB_Qout	5 bit	Έξοδος	Tag δεδομένων προς CDB
CDB_Vout	32 bit	Έξοδος	Δεδομένα προς CDB
AU_busy	1 bit	Έξοδος	FU απασχολημένη σε αυτό τον κύκλο ρολογιού
AU_result_ready	1 bit	Έξοδος	Έγκυρο αποτέλεσμα – ενεργό ένα κύκλο πριν τα έγκυρα δεδομένα βρεθούν στις εξόδους CDB_Qout, CDB_Vout

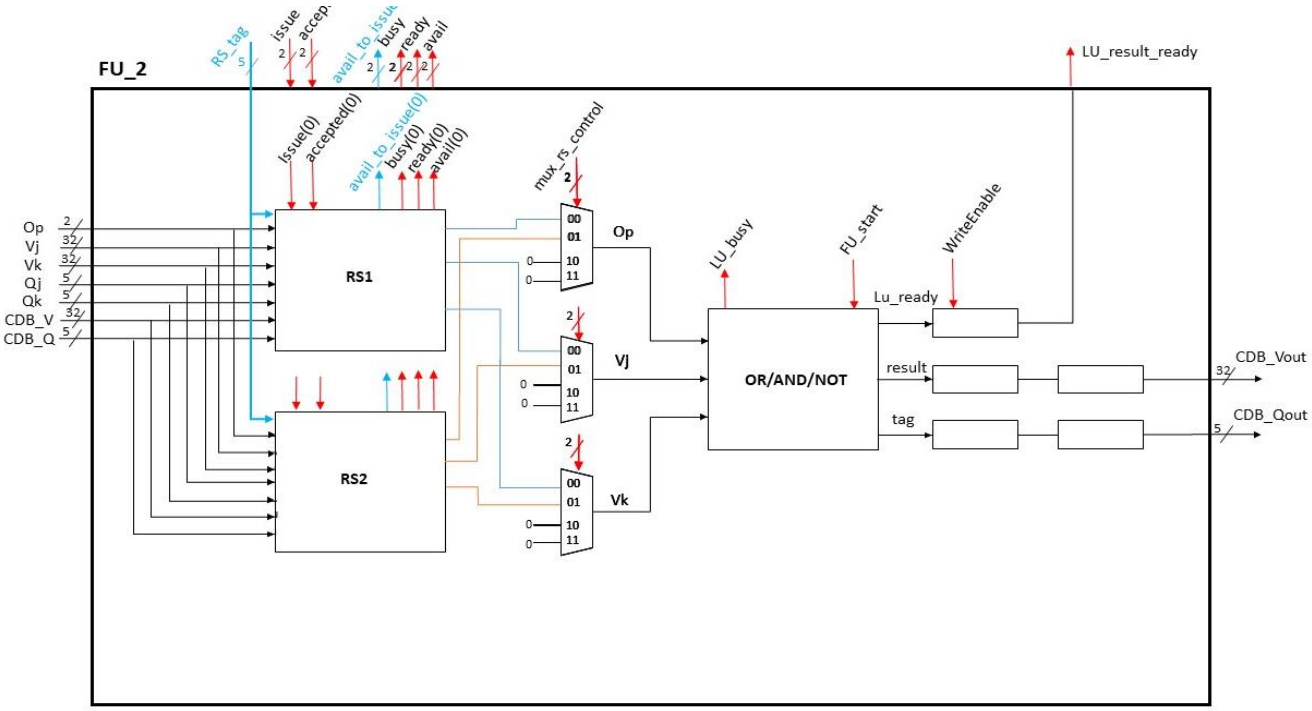
Το σχηματικό διάγραμμα είναι:



Logical Functional Unit (FU_2)

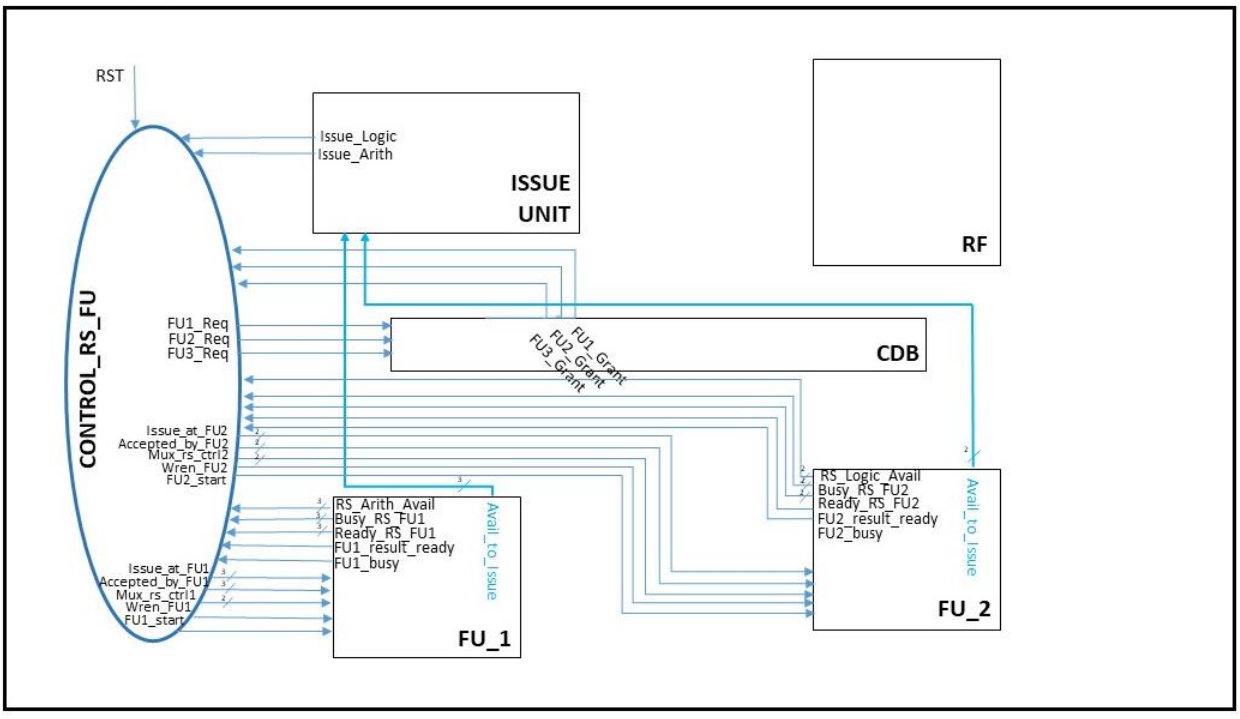
Σήμα	Πλάτος	Είδος	Περιγραφή
CLK	1 bit	Είσοδος	Σήμα Clock
RST	1 bit	Είσοδος	Σήμα Reset
WriteEnable	1 bit	Είσοδος	Σήμα Write Enable για τους pipelined καταχωρητές στην έξοδο.
Op	2 bit	Είσοδος	Κωδικός πράξης προς αποθήκευση στους RS του FU
CDB_Q	5 bit	Είσοδος	Tag των δεδομένων που βρίσκονται στο CDB
CDB_V	32 bit	Είσοδος	Δεδομένα που βρίσκονται στο CDB
Vj	32 bit	Είσοδος	Δεδομένα από RF προς αποθήκευση στους RS του FU
Vk	32 bit	Είσοδος	Δεδομένα από RF προς αποθήκευση στους RS του FU
Qj	5 bit	Είσοδος	Tag από RF προς αποθήκευση στους RS του FU
QK	5 bit	Είσοδος	Tag από RF προς αποθήκευση στους RS του FU
mux_rs_control	2 bit	Είσοδος	Έλεγχος πολυπλέκτη που κατευθύνει τα δεδομένα Op, Vj, Vk στην FU. 00: RS1 σε FU, 01: RS2 σε FU, 10/11: zero values to FU
Accepted	2 bit	Είσοδος	00: κανένα RS δεν έγινε δεκτό για να εκτελεστεί πράξη 01/10: το RS1/RS2 έγινε δεκτό για τη λογική πράξη 11: δεν ορίζεται
Issue	2 bit	Είσοδος	00: δεν εκδίδεται κάποια εντολή σε κανένα RS 01/10: εκδίδεται εντολή στο RS1/RS2 11: δεν ορίζεται
FU_start	1 bit	Είσοδος	Σήμα εκκίνησης εκτέλεσης λογικής πράξης στην FU
RS_tag	5 bit	Είσοδος	RS_tag από ISSUE_UNIT
Busy	2 bit	Έξοδος	00: κανένας RS απασχολημένος 01/10: RS1/RS2 απασχολημένος 11: RS2 & RS1 απασχολημένοι
Ready	2 bit	Έξοδος	00: κανένας RS με έγκυρα δεδομένα 01/10: RS1/RS2 με έγκυρα δεδομένα 11: RS2 & RS1 με έγκυρα δεδομένα
Available	2 bit	Έξοδος	00: κανένας RS διαθέσιμος να δεχτεί δεδομένα 01/10: RS1/RS2 διαθέσιμος να δεχτεί δεδομένα 11: RS2 & RS1 διαθέσιμοι να δεχτούν δεδομένα
Avail_to_Issue	2 bit	Έξοδος	Ακολουθεί την ίδια λογική με το σήμα Available. Η μόνη διαφορά είναι ότι το σήμα ενημερώνεται στην αρχή του κύκλου και δεν δημιουργεί προβλήματα χρονισμού με το Issue Unit.
CDB_Qout	5 bit	Έξοδος	Tag δεδομένων προς CDB
CDB_Vout	32 bit	Έξοδος	Δεδομένα προς CDB
LU_busy	1 bit	Έξοδος	FU απασχολημένη σε αυτό τον κύκλο ρολογιού
LU_result_ready	1 bit	Έξοδος	Έγκυρο αποτέλεσμα – ενεργό ένα κύκλο πριν τα έγκυρα δεδομένα βρεθούν στις εξόδους CDB_Qout, CDB_Vout

Το σχηματικό διάγραμμα είναι:



Η αλλαγή στη σύνδεση των σημάτων Available με το ISSUE UNIT φαίνεται στο σχηματικό διάγραμμα του CTRL_RS_FU. Πλέον δεν συνδέονται τα RS_Arith_Avail και RS_Logic_Avail, αλλά τα δύο νέα σήματα Avail_to_Issue με το ISSUE UNIT:

CONTROL_RS_FU



2. Στρατηγική και αποτελέσματα ελέγχου

- **Έλεγχος έκδοσης εντολών μετά την ολοκλήρωση της εντολής υπό εκτέλεση:**

Περιγραφή προσέγγισης ελέγχου:

Εκδίδουμε τις εξής εντολές:

1. **not x1 x2 x3** (Issue=1, FU_type=00, Fop=10, Ri=00001, Rj=00010, Rk=00011)
2. **add x2 x1 x3** (Issue=1, FU_type=01, Fop=00, Ri=00010, Rj=00001, Rk=00011)
3. **not x3 x0 x0** (Issue=1, FU_type=00, Fop=10, Ri=00011, Rj=00000, Rk=00000)
4. **or x4 x2 x3** (Issue=1, FU_type=00, Fop=00, Ri=00100, Rj=00011, Rk=00000)
5. **add x5 x0 x0** (Issue=1, FU_type=01, Fop=00, Ri=00101, Rj=00000, Rk=00000)
6. **sub x6 x1 x3** (Issue=1, FU_type=01, Fop=01, Ri=00110, Rj=00100, Rk=00101)
7. **sll x7 x3 x5** (Issue=1, FU_type=01, Fop=10, Ri=00111, Rj=00011, Rk=00101)

Τα αναμενόμενα αποτελέσματα στους καταχωρητές της RF είναι:

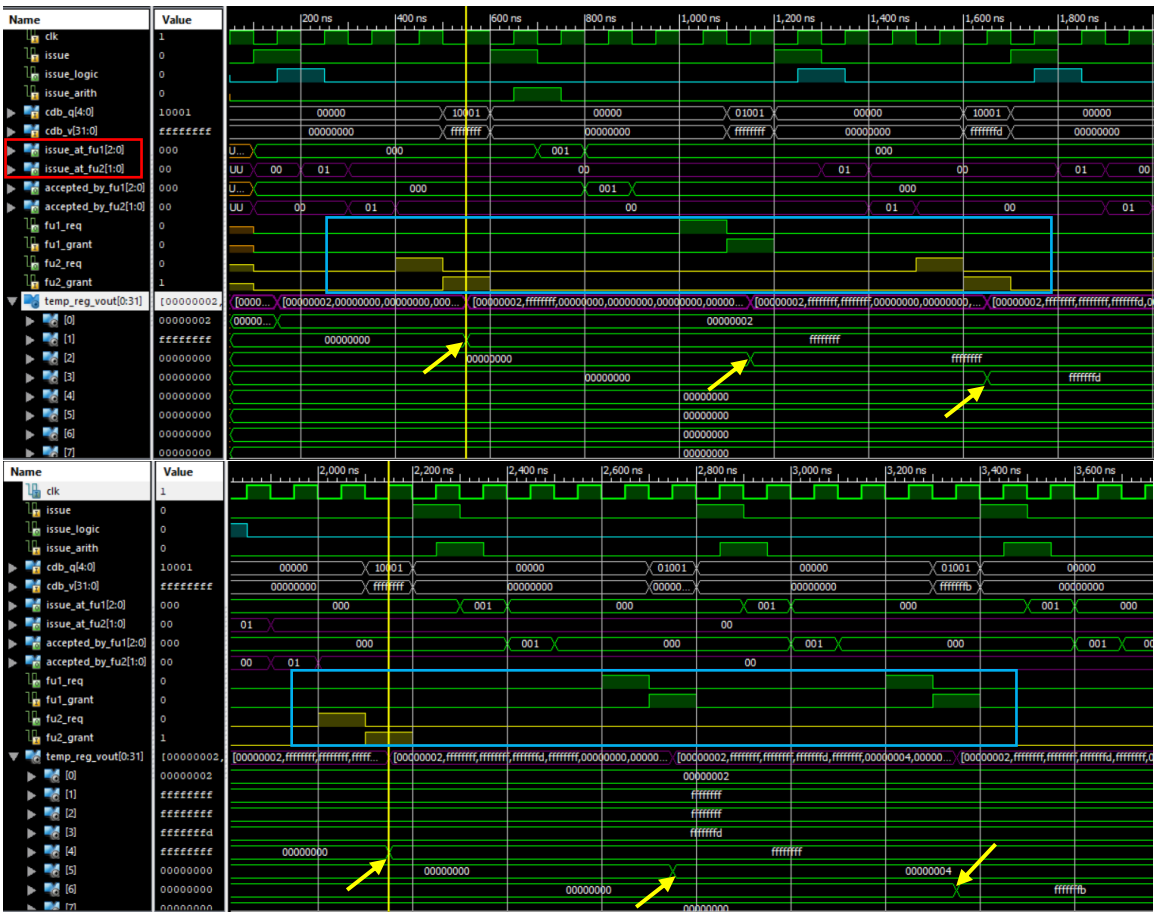
Reg1: $(ffffffff)_{16}$, Reg2: $(ffffffff)_{16}$,Reg3: $(fffffffd)_{16}$, Reg4: $(ffffffff)_{16}$, Reg5: $(00000004)_{16}$,

Reg6: $(fffffffb)_{16}$, Reg7: $(fffffd0)_{16}$

Μεταξύ της κάθε εντολής που εκδίδουμε περιμένουμε κάποιους κύκλους ρολογιού πριν εκδώσουμε την επόμενη εντολή. Συγκεκριμένα, περιμένουμε 4 κύκλους ρολογιού αν πρόκειται για λογική πράξη και 5 κύκλους ρολογιού αν πρόκειται για αριθμητική πράξη. Για τις λογικές πράξεις οι 4 κύκλοι ρολογιού δικαιολογούνται ως εξής: 1 κύκλος ρολογιού για εγγραφή των απαραίτητων δεδομένων σε ένα RS, 2 κύκλοι ρολογιού για να εμφανιστεί έγκυρο αποτέλεσμα στην έξοδο της λειτουργικής μονάδας και 1 κύκλο ρολογιού για να υπάρξει εγγραφή στην RF μέσω του CDB. Για τις αριθμητικές πράξεις οι 5 κύκλοι ρολογιού δικαιολογούνται ως εξής: 1 κύκλος ρολογιού για εγγραφή των απαραίτητων δεδομένων σε ένα RS, 3 κύκλοι ρολογιού για να εμφανιστεί έγκυρο αποτέλεσμα στην έξοδο της λειτουργικής μονάδας και 1 κύκλο ρολογιού για να υπάρξει εγγραφή στην RF μέσω του CDB. Επιβεβαιώνουμε ότι η εντολή έχει εκτελεστεί σωστά αν το αποτέλεσμα βρίσκεται στο CDB την κατάλληλη χρονική στιγμή και εγγράφεται στην RF στη θετική ακμή του ρολογιού.

Διάγραμμα χρονισμού και τεκμηρίωση λειτουργικότητας:

Στα παρακάτω διάγραμμα χρονισμού επιβεβαιώνεται η ορθή λειτουργία του συστήματος για την προσέγγιση ελέγχου του αναπτύχθηκε παραπάνω. Συγκεκριμένα, η κίτρινη γραμμή δείχνει τη στιγμή που έχουμε θετική ακμή ρολογιού και έγκυρο αποτέλεσμα στο CDB. Εκείνη τη στιγμή γίνεται και η εγγραφή στον αντίστοιχο register της RF. Το σήμα temp_reg_vout είναι ένας πίνακας 32x32 που αποθηκεύει την τιμή του κάθε V register της RF. Τα κίτρινα βέλη δείχνουν τις στιγμές που ενημερώνεται ο επιθυμητός register της RF. Η ενημέρωση της τιμής του register 7 δεν φαίνεται στα παραπάνω διαγράμματα χρονισμού, αλλά γίνεται ορθά σε επόμενους κύκλους ρολογιού. Στα μπλε πλαίσια φαίνονται τα αντίστοιχα request και grant σήματα του συστήματος. Τα issue_at_fu1 (LSB: RS1, MSB: RS3) και issue_at_fu2 (LSB: RS1, MSB: RS2) σήματα (κόκκινο πλαίσιο) δείχνουν σε ποιο RS εκδίδεται η κάθε εντολή. Η FU1 είναι η λειτουργική μονάδα αριθμητικών πράξεων και η FU2 η λειτουργική μονάδα λογικών πράξεων.



- **Έλεγχος επικάλυψης ανεξάρτητων εντολών που χρησιμοποιούν διαφορετικές λειτουργικές μονάδες:**

Περιγραφή προσέγγισης ελέγχου:

Ξεκινάμε τον συγκεκριμένο έλεγχο αφού αρχικοποιήσουμε τους καταχωρητές 1 και 2 εκδίδοντας τις εντολές και περιμένοντας να εκτελεστεί η κάθε εντολή πριν εισάγουμε την επόμενη:

1. **not x1 x2 x3** (Issue=1, FU_type=00, Fop=10, Ri=00001, Rj=00010, Rk=00011)
2. **add x2 x1 x3** (Issue=1, FU_type=01, Fop=00, Ri=00010, Rj=00001, Rk=00011)

Στη συνέχεια εκδίδουμε διαδοχικά τις εξής εντολές:

3. **sll x4 x1 x0** (Issue=1, FU_type=01, Fop=10, Ri=00100, Rj=00001, Rk=00000)
4. **not x3 x0 x1** (Issue=1, FU_type=00, Fop=10, Ri=00011, Rj=00000, Rk=00001)

Τα αναμενόμενα αποτελέσματα στους καταχωρητές της RF είναι:

Reg1: $(ffffffff)_{16}$, Reg2: $(ffffffff)_{16}$,Reg3: $(fffffffd)_{16}$, Reg4: $(fffffffc)_{16}$

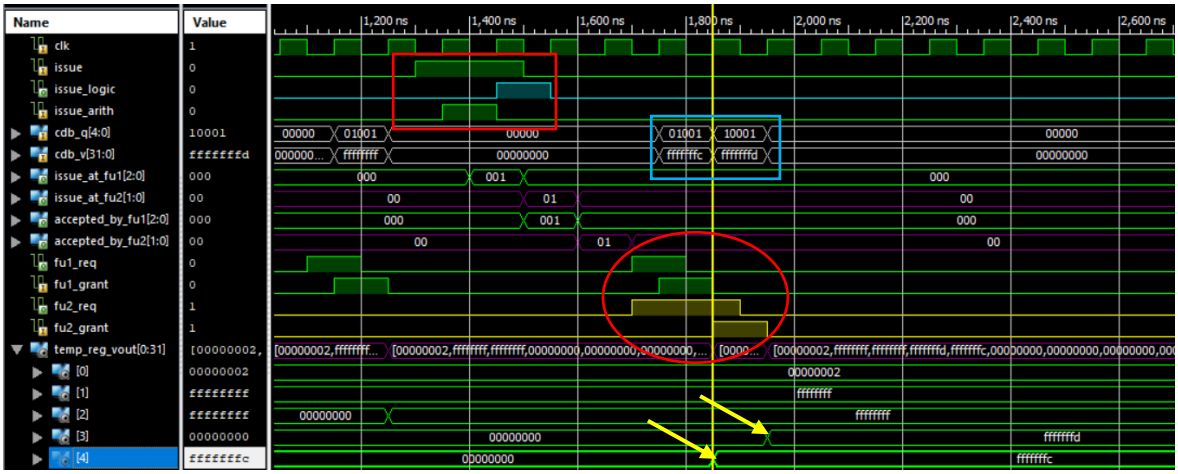
Στη συγκεκριμένη προσέγγιση ελέγχου προσέχουμε την εξής λεπτομέρεια: η sll είναι αριθμητική πράξη που απαιτεί 3 κύκλους ρολογιού για να έχει έγκυρο αποτέλεσμα στην έξοδο της αριθμητικής λειτουργικής μονάδας. Η not είναι λογική πράξη που απαιτεί 2 κύκλους ρολογιού

για να έχει έγκυρο αποτέλεσμα στην έξοδο της λογικής λειτουργικής μονάδας. Η not εντολή εκδίδεται ένα κύκλο μετά από την sll εντολή και άρα κατά την εκτέλεση τους από το σύστημα, τόσο η αριθμητική λειτουργική μονάδα, όσο και η λογική λειτουργική μονάδα θα έχουν έγκυρο αποτέλεσμα στις εξόδους τους στον ίδιο κύκλο ρολογιού. Άρα θα κάνουν ταυτόχρονα request για χρήση του CDB. Μόνο μία από τις δύο θα γίνει αποδεκτή για να χρησιμοποιήσει το CDB και περιμένουμε σε επόμενο κύκλο να υπάρξει εκ νέου request από τη λειτουργική μονάδα που έχει έγκυρο αποτέλεσμα αλλά δεν έγινε δεκτή στην προηγουμένη προσπάθεια. Αντίστοιχα περιμένουμε να υπάρξει και το αντίστοιχο grant και το αποτέλεσμα να εγγραφεί στην RF ένα κύκλο μετά, όπως θα πρέπει να έχει γίνει και με την εντολή που προηγήθηκε.

Διάγραμμα χρονισμού και τεκμηρίωση λειτουργικότητας:

Στο κόκκινο πλαίσιο φαίνεται η διαδοχική έκδοση της αριθμητικής και της λογικής εντολής που χρησιμοποιούνται για την προσέγγιση ελέγχου. Στον κόκκινο κύκλο φαίνονται τα δύο request και πως χειρίζεται το control των RS και FU και το CDB την περίπτωση που υπάρξουν ταυτόχρονα δύο request από δύο διαφορετικές λειτουργικές μονάδες. Σε κάθε grant το αποτέλεσμα που υπάρχει στο CDB (μπλε πλαίσιο) εγγράφεται στον αντίστοιχο register της RF (κίτρινα βέλη).

Το παρακάτω διάγραμμα χρονισμού επιβεβαιώνει την ορθή λειτουργία του συστήματος στην περίπτωση έκδοσης ανεξάρτητων εντολών που χρησιμοποιούν διαφορετικές λειτουργικές μονάδες.



- **Έλεγχος επικάλυψης ανεξάρτητων εντολών που χρησιμοποιούν την ίδια λειτουργική μονάδα:**

Περιγραφή προσέγγισης ελέγχου:

Ξεκινάμε τον συγκεκριμένο έλεγχο αφού αρχικοποιήσουμε τους καταχωρητές 1, 2 και 3 εκδίδοντας τις εντολές και περιμένοντας να εκτελεστεί η κάθε εντολή πριν εισάγουμε την επόμενη:

1. **not x1 x2 x3** (Issue=1, FU_type=00, Fop=10, Ri=00001, Rj=00010, Rk=00011)
2. **add x2 x1 x3** (Issue=1, FU_type=01, Fop=00, Ri=00010, Rj=00001, Rk=00011)

3. **or x3 x0 x0** (Issue=1, FU_type=00, Fop=00, Ri=00011, Rj=00000, Rk=00000)

Στη συνέχεια εκδίδουμε διαδοχικά τις εξής δύο ανεξάρτητες αριθμητικές εντολές:

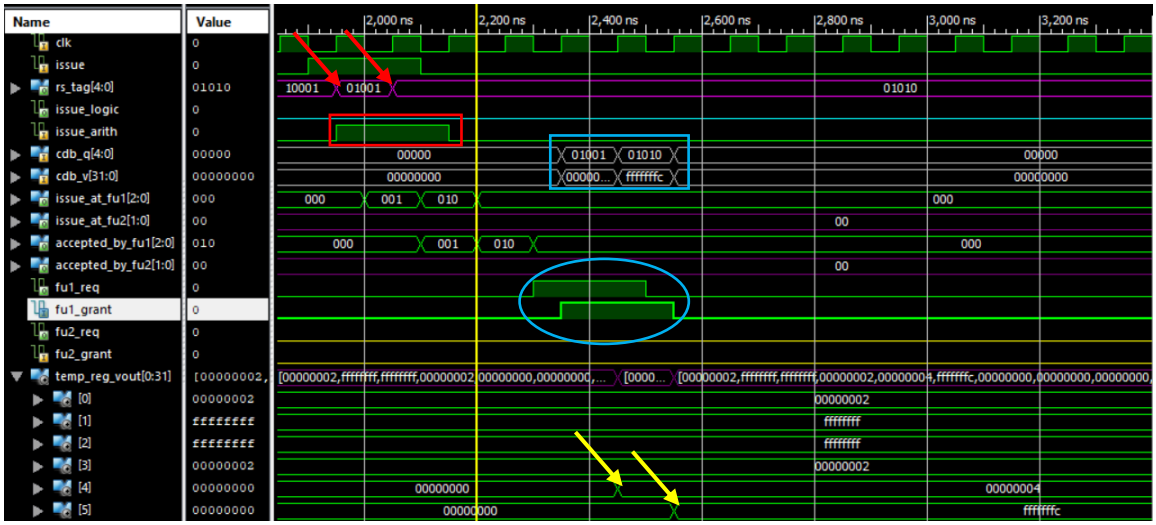
4. **add x4 x3 x3** (Issue=1, FU_type=01, Fop=00, Ri=00100, Rj=00011, Rk=00011)
5. **sll x5 x1 x0** (Issue=1, FU_type=01, Fop=10, Ri=00101, Rj=00001, Rk=00000)

Τα αναμενόμενα αποτελέσματα στους καταχωρητές της RF είναι:

Reg1: $(ffffffff)_{16}$, Reg2: $(ffffffff)_{16}$, Reg3: $(00000002)_{16}$, Reg4: $(00000004)_{16}$, Reg5: $(fffffffc)_{16}$

Διάγραμμα χρονισμού και τεκμηρίωση λειτουργικότητας:

Στο κόκκινο πλαίσιο φαίνεται ότι γίνεται έκδοση δύο διαδοχικών αριθμητικών εντολών, ενώ τα κόκκινα βέλη δείχνουν ότι το RS_tag που θα εγγραφεί στο πεδίο Q του destination register,



παίρνει τη σωστή τιμή για τα δύο διαδοχικά αριθμητικά issues (01001 για FU1 και RS1, 01010 για FU1 και RS2). Τρεις κύκλους μετά την έκδοση της πρώτης εντολής γίνεται το πρώτο request και ένα κύκλο μετά το πρώτο grant. Την ίδια λογική ακολουθεί και το δεύτερο request και grant σήμα (μπλε κύκλος). Το CDB ενημερώνει κατάλληλα τα δεδομένα του όταν γίνεται κάποιο grant ενεργό (μπλε πλαίσιο) και στη θετική ακμή του ρολογιού οι καταχωρητές (πεδίο V) της RF ενημερώνονται με την κατάλληλη τιμή (κίτρινα βέλη). Επιβεβαιώνεται έτσι η ορθή λειτουργία του συστήματος στην περίπτωση επικάλυψης ανεξάρτητων εντολών που χρησιμοποιούν την ίδια λειτουργική μονάδα.

- **Έλεγχος επικάλυψης ανεξάρτητων εντολών που χρησιμοποιούν την ίδια και διαφορετικές λειτουργικές μονάδες:**

Περιγραφή προσέγγισης ελέγχου:

Ξεκινάμε τον συγκεκριμένο έλεγχο αφού αρχικοποιήσουμε τους καταχωρητές 1, 2 και 3 εκδίδοντας τις εντολές και περιμένοντας να εκτελεστεί η κάθε εντολή πριν εισάγουμε την επόμενη:

1. **not x1 x2 x3** (Issue=1, FU_type=00, Fop=10, Ri=00001, Rj=00010, Rk=00011)
2. **add x2 x1 x3** (Issue=1, FU_type=01, Fop=00, Ri=00010, Rj=00001, Rk=00011)
3. **or x3 x0 x0** (Issue=1, FU_type=00, Fop=00, Ri=00011, Rj=00000, Rk=00000)

Στη συνέχεια εκδίδουμε διαδοχικά δύο ανεξάρτητες αριθμητικές εντολές και μία λογική εντολή:

4. **add x4 x3 x3** (Issue=1, FU_type=01, Fop=00, Ri=00100, Rj=00011, Rk=00011)
5. **sll x5 x1 x0** (Issue=1, FU_type=01, Fop=10, Ri=00101, Rj=00001, Rk=00000)
6. **not x6 x0 x1** (Issue=1, FU_type=00, Fop=10, Ri=00110, Rj=00000, Rk=00001)

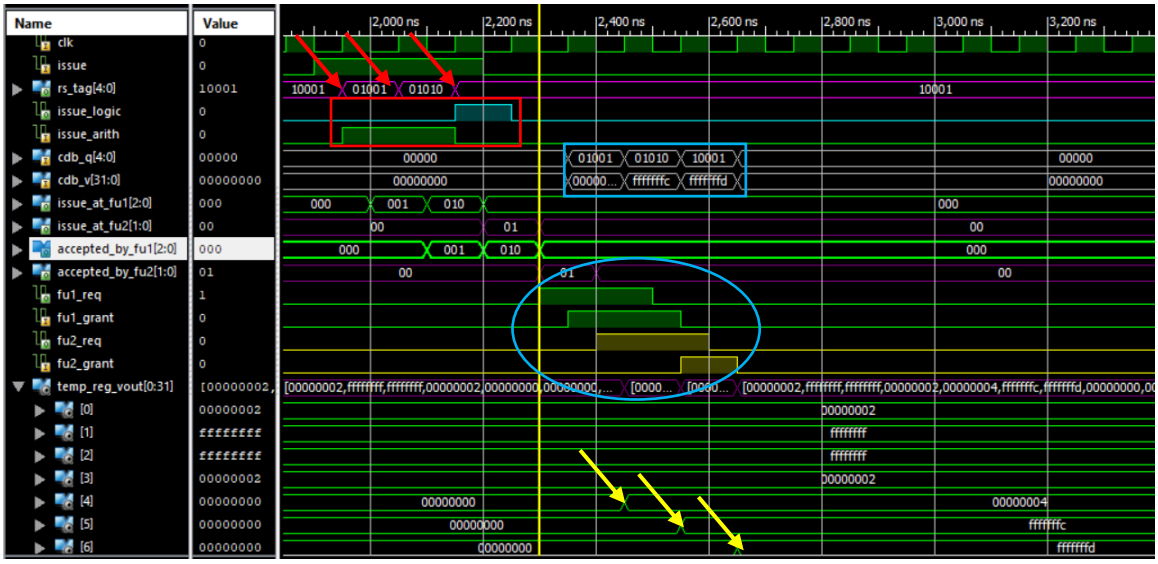
Τα αναμενόμενα αποτελέσματα στους καταχωρητές της RF είναι:

Reg1: $(ffffffff)_{16}$, Reg2: $(ffffffff)_{16}$, Reg3: $(00000002)_{16}$, Reg4: $(00000004)_{16}$, Reg5: $(fffffffc)_{16}$,

Reg6: $(fffffffd)_{16}$

<u>Διάγραμμα χρονισμού και τεκμηρίωση λειτουργικότητας:</u>

Στο κόκκινο πλαίσιο φαίνεται ότι γίνεται έκδοση δύο διαδοχικών αριθμητικών εντολών και μίας λογικής εντολής, ενώ τα κόκκινα βέλη δείχνουν ότι το RS_tag που θα εγγραφεί στο πεδίο Q του



destination register, παίρνει τη σωστή τιμή για τα δύο διαδοχικά αριθμητικά issues (01001 για FU1 και RS1, 01010 για FU1 και RS2) και για το λογικό issue (10001 για FU2 και RS1). Τρεις κύκλους μετά την έκδοση της πρώτης εντολής γίνεται το πρώτο request και ένα κύκλο μετά το πρώτο grant. Την ίδια λογική ακολουθεί και το δεύτερο request και grant σήμα (μπλε κύκλος).

Σε αυτή την προσέγγιση ελέγχου μαζί με το δεύτερο request της αριθμητικής λειτουργικής μονάδας, υπάρχει και ένα request της λογικής λειτουργικής μονάδας. Εφόσον κατά τη λειτουργία του συστήματος δεν έχουν υπάρξει δύο requests ταυτόχρονα, δίνεται προτεραιότητα στην αριθμητική λειτουργική μονάδα να χρησιμοποιήσει το CDB, ενώ το request της λογικής λειτουργικής μονάδας παραμένει ενεργό. Στον επόμενο κύκλο, υπάρχει αποδοχή και της λογικής λειτουργικής μονάδας από το CDB. Το CDB ενημερώνει κατάλληλα τα δεδομένα του όταν γίνεται κάποιο grant ενεργό (μπλε πλαίσιο) και στη θετική ακμή του ρολογιού οι καταχωρητές (πεδίο V) της RF ενημερώνονται με την κατάλληλη τιμή (κίτρινα βέλη). Επιβεβαιώνεται έτσι η ορθή

λειτουργία του συστήματος στην περίπτωση επικάλυψης ανεξάρτητων εντολών που χρησιμοποιούν την ίδια λειτουργική μονάδα.

- **Corner case 1 -** Στο σύστημα γεμίζει η λογική λειτουργική μονάδα (περιμένοντας αποτέλεσμα από την αριθμητική λειτουργική μονάδα για να γίνουν τα RS της έτοιμα προς χρήση από τη λογική λειτουργική μονάδα):

Περιγραφή προσέγγισης ελέγχου:

Ξεκινάμε τον συγκεκριμένο έλεγχο αφού αρχικοποιήσουμε τους καταχωρητές 1, 2 και 3 εκδίδοντας τις εντολές και περιμένοντας να εκτελεστεί η κάθε εντολή πριν εισάγουμε την επόμενη:

1. **not x1 x2 x3** (Issue=1, FU_type=00, Fop=10, Ri=00001, Rj=00010, Rk=00011)
2. **add x2 x1 x3** (Issue=1, FU_type=01, Fop=00, Ri=00010, Rj=00001, Rk=00011)
3. **or x3 x0 x0** (Issue=1, FU_type=00, Fop=00, Ri=00011, Rj=00000, Rk=00000)

Στη συνέχεια εκδίδουμε διαδοχικά τις εξής εντολές (μερικές από τις οποίες είναι εξαρτημένες):

4. **add x4 x3 x3** (Issue=1, FU_type=01, Fop=00, Ri=00100, Rj=00011, Rk=00011)
5. **sll x5 x1 x0** (Issue=1, FU_type=01, Fop=10, Ri=00101, Rj=00001, Rk=00000)
6. **not x6 x5 x1** (Issue=1, FU_type=00, Fop=10, Ri=00110, Rj=00101, Rk=00001)
7. **not x7 x5 x0** (Issue=1, FU_type=00, Fop=10, Ri=00111, Rj=00101, Rk=00000)
8. **and x8 x5 x4** (Issue=1, FU_type=00, Fop=01, Ri=01000, Rj=00101, Rk=00100)

Τα αναμενόμενα αποτελέσματα στους καταχωρητές της RF είναι:

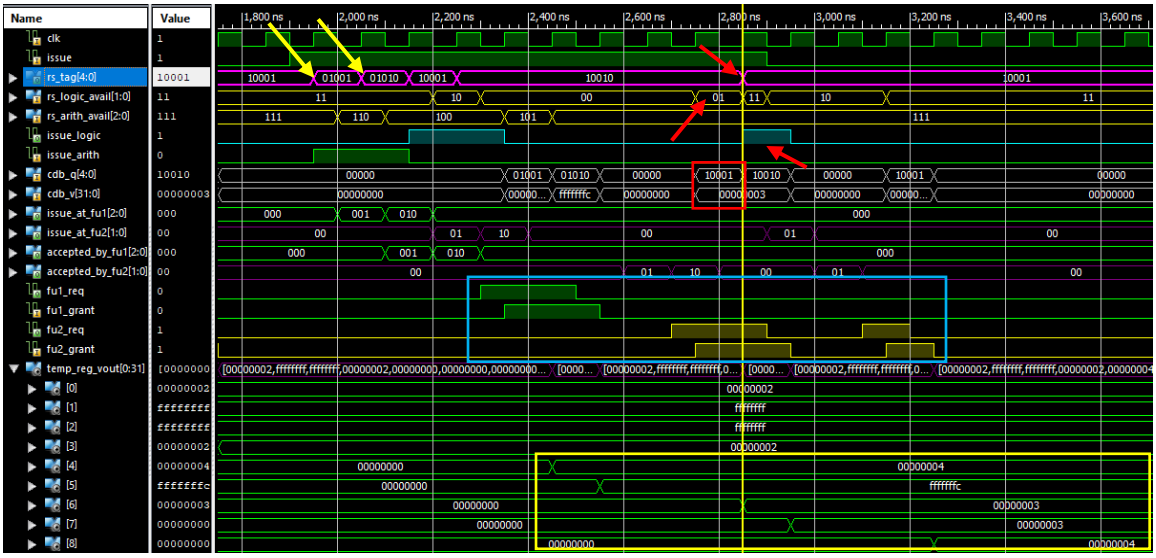
Reg1: $(ffffffff)_{16}$, Reg2: $(ffffffff)_{16}$, Reg3: $(00000002)_{16}$, Reg4: $(00000004)_{16}$, Reg5: $(fffffffc)_{16}$

Reg6: $(00000003)_{16}$, Reg7: $(00000003)_{16}$, Reg8: $(00000004)_{16}$

Διάγραμμα χρονισμού και τεκμηρίωση λειτουργικότητας:

Ξεκινάμε τον έλεγχο εκδίδοντας δύο αριθμητικές εντολές (add και sll) με RS_tag: 01001 και 01010 (κίτρινα βέλη) και στη συνέχεια τρεις λογικές εντολές. Παρατηρούμε την εξάρτηση στον καταχωρητή 5 από τις δύο πρώτες λογικές εντολές. Στην περίπτωση αυτή οι δύο λογικές εντολές που εκδίδονται στα RS θα περιμένουν το αποτέλεσμα που θα παραχθεί από την αριθμητική λειτουργική μονάδα και τον RS2 (CDB_Q=01010). Ως εκ τούτου, η τρίτη λογική εντολή δεν μπορεί να εκδοθεί έως ότου ο RS1 της λογικής λειτουργικής μονάδας γίνει διαθέσιμος προς χρήση ξανά. Αυτό θα γίνει όταν το CDB αποδεχτεί το αποτέλεσμα με tag=10001 (κόκκινο πλαίσιο). Τη συγκεκριμένη χρονική στιγμή ο RS1 της λογικής λειτουργικής μονάδας γίνεται διαθέσιμος και εκδίδεται η τελευταία λογική εντολή, με το κατάλληλο RS_tag (κόκκινα βέλη). Στο μπλε πλαίσιο φαίνονται τα requests και οι αποδοχές τους (grant) από το CDB. Τέλος, κατά τη θετική ακμή του

ρολογιού οι καταχωρητές (πεδία V) της RF ενημερώνονται με την κατάλληλη τιμή (κίτρινο πλαίσιο).



- **Corner case 2 -** Στο σύστημα, κατά τον κύκλο ολοκλήρωσης μίας εντολής, εκδίδεται μία άλλη εντολή που διαβάζει το αποτέλεσμα που γράφεται σε αυτό τον κύκλο στο CDB:

Περιγραφή προσέγγισης ελέγχου:

Ξεκινάμε τον συγκεκριμένο έλεγχο αφού αρχικοποιήσουμε τους καταχωρητές 1, 2 και 3 εκδίδοντας τις εντολές και περιμένοντας να εκτελεστεί η κάθε εντολή πριν εισάγουμε την επόμενη:

1. **not x1 x2 x3** (Issue=1, FU_type=00, Fop=10, Ri=00001, Rj=00010, Rk=00011)
2. **add x2 x1 x3** (Issue=1, FU_type=01, Fop=00, Ri=00010, Rj=00001, Rk=00011)
3. **or x3 x0 x0** (Issue=1, FU_type=00, Fop=00, Ri=00011, Rj=00000, Rk=00000)

Στη συνέχεια εκδίδουμε διαδοχικά τις εξής εξαρτημένες εντολές (η not εκδίδεται όταν η αριθμητική εντολή είναι έτοιμη να γράψει το αποτέλεσμα στο CDB, δηλαδή 4 κύκλους μετά την έκδοση της):

4. **add x4 x3 x3** (Issue=1, FU_type=01, Fop=00, Ri=00100, Rj=00011, Rk=00011)
5. **not x5 x4 x0** (Issue=1, FU_type=00, Fop=10, Ri=00101, Rj=00100, Rk=00000)

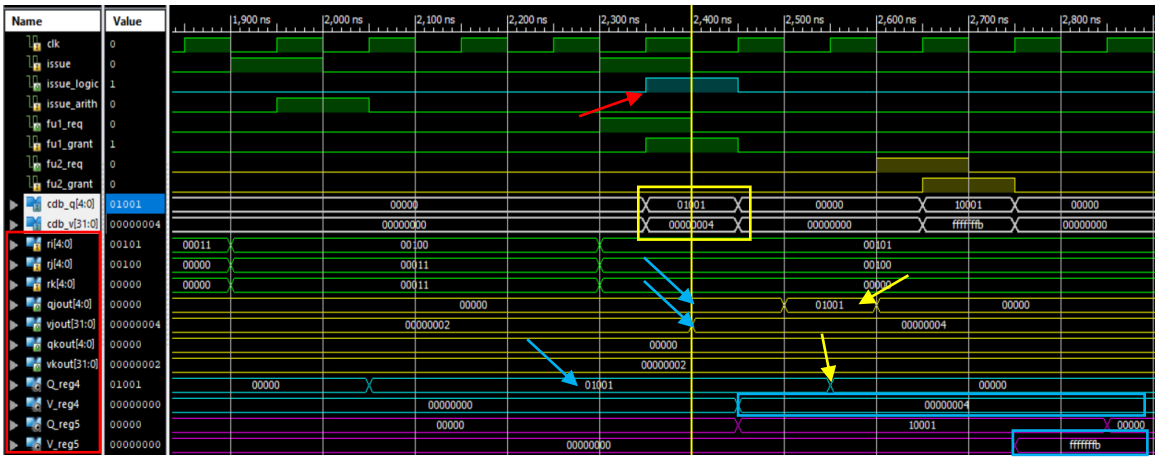
Τα αναμενόμενα αποτελέσματα στους καταχωρητές της RF είναι:

Reg1: $(ffffffff)_{16}$, Reg2: $(ffffffff)_{16}$, Reg3: $(00000002)_{16}$, Reg4: $(00000004)_{16}$, Reg5: $(fffffffb)_{16}$

Διάγραμμα χρονισμού και τεκμηρίωση λειτουργικότητας:

Στο κόκκινο πλαίσιο φαίνονται τα πεδία της RF. Με τη συγκεκριμένη προσέγγιση ελέγχου επιχειρούμε να δείξουμε ότι κατά τον κύκλο έκδοσης του αποτελέσματος της αριθμητικής πράξης στο CDB, η λογική πράξη που εκδίδεται και έχει ως source register τον destination register

της αριθμητικής εντολής, θα πάρει κατευθείαν από την RF το αποτέλεσμα που υπάρχει στο CDB, χωρίς να περιμένει τον επόμενο κύκλο για να ενημερώσει το περιεχόμενο του RS με βάση την έξοδο των καταχωρητών της RF. Με λίγα λόγια, ελέγχουμε τον τρόπο λειτουργίας της τεχνικής fall through. Κατά την έκδοση της λογικής πράξης **not x5 x4 x0** (κόκκινο βέλος) το αποτέλεσμα της αριθμητικής πράξης **add x4 x3 x3** έχει εκδοθεί στο CDB (κίτρινο πλαίσιο) και παρατηρούμε ότι ενώ το πεδίο Q του register 4 έχει αποθηκευμένο tag 01001, οι έξοδοι Qj και Vj της RF προωθούν απευθείας το αποτέλεσμα του CDB (δηλαδή το Qj έχει τιμή 00000 και το Vj έχει την τιμή του CDB_V) (μπλε βέλη). Η επανεμφάνιση του tag 01001 στο πεδίο Qj δεν επηρεάζει το σύστημα καθώς σε εκείνο τον κύκλο ρολογιού δεν υπάρχει write enable ενεργό για την RF (δεν φαίνεται στο παρακάτω διάγραμμα χρονισμού), ενώ κατά τη θετική ακμή του ίδιου κύκλου το πεδίο Q του register 4, έχοντας εξυπηρετηθεί από το CDB, ενημερώνεται με μηδενική τιμή (κίτρινα βέλη). Τέλος, κατά τη θετική ακμή του ρολογιού οι καταχωρητές (πεδία V) της RF ενημερώνονται με την κατάλληλη τιμή (μπλε πλαίσια), επιβεβαιώνοντας την ορθή λειτουργία του συστήματος.



3. Σχόλια – Παρατηρήσεις:

- Για τους παραπάνω ελέγχους χρησιμοποιήθηκε το ίδιο testbench (TOMASULO_PROC_TEST.vhd). Το testbench περιέχει 6 διαφορετικά κομμάτια κώδικα, καθένα από τα οποία ελέγχει μία από τις έξι περιπτώσεις ελέγχου της παρούσας αναφοράς. Όλα τα κομμάτια κώδικα περιέχουν κάποιες εντολές αρχικοποίησης. Για τον έλεγχο της 4ης και 6ης περίπτωσης, χρησιμοποιούνται οι αρχικοποιήσεις των κωδίκων ελέγχου των περιπτώσεων 3 και 5. Στο testbench αναφέρεται με σχόλια ποια κομμάτια κώδικα πρέπει να μπουν σε σχόλια για να λειτουργήσει κάθε έλεγχος.

- Ο φάκελος των παραδοτέων περιέχει, εκτός από την αναφορά (στο φάκελο REPORT), τους φακέλους SOURCES, TESTBENCHES και WAVEFORMS. Στον φάκελο SOURCES περιέχονται όλα τα .vhd αρχεία που χρησιμοποιούνται στο top module (TOMASULO_PROC) και στις επιμέρους μονάδες. Στον φάκελο TESTBENCHES περιέχεται μόνο το testbench (TOMASULO_PROC_TEST .vhd) που χρησιμοποιήθηκε για να επιβεβαιωθεί η ορθή λειτουργία του συνολικού συστήματος για τις 6 περιπτώσεις ελέγχου που αναλύθηκαν στην παρούσα αναφορά. Στον φάκελο WAVEFORMS περιέχεται το .wcfg αρχείο, το οποίο έχει αποθηκευμένα τα απαραίτητα σήματα που είναι χρήσιμα για το αντίστοιχο testbench.