Technical University of Crete

Αρχιτεκτονική Υπολογιστών

HPY 415

Αναφορά Εργαστηρίου 3 : Tomasulo + Reorder Buffer

Ομάδα Εργασίας
Μπελλώνιας Παναγιώτης - 2014030058
Σπυριδάκης Χρήστος - 2014030022

Όλες οι εικόνες που είναι ενσωματωμένες στην αναφορά υπάρχουν στον φάκελο /doc/sim και /doc/schematic

Στα σχεδιαγράμματα που ακολουθούν ισχύει ο παρακάτω χρωματικός κώδικας για τα σήματα:

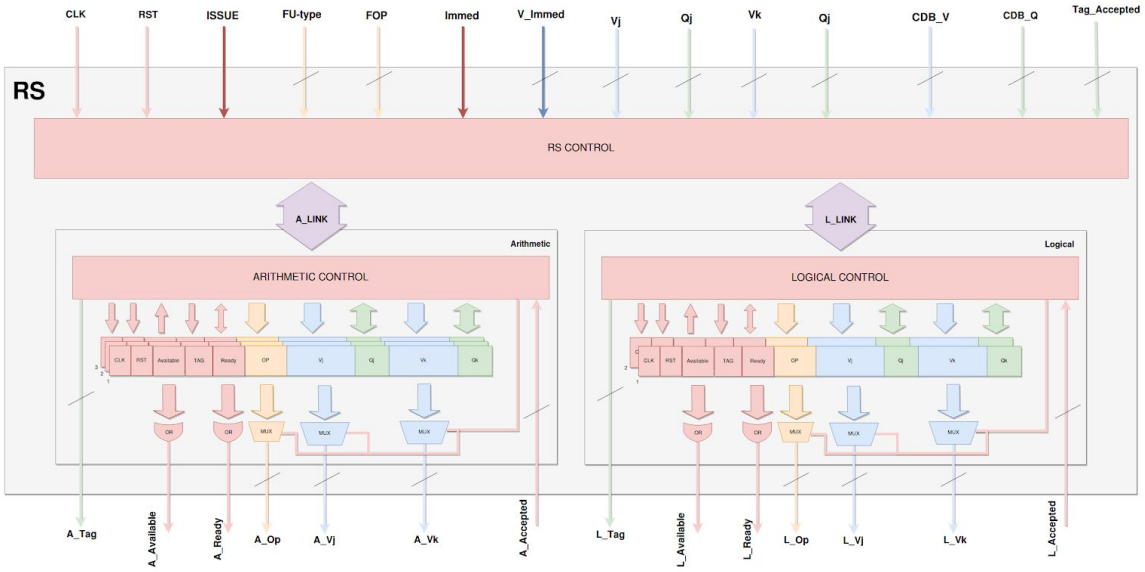
Κόκκινο - Ελέγχου
Μπλε - Δεδομένα
Πορτοκαλί - Βοηθητικά / Πληροφορίας
Πράσινο - Tag

Στα testbench κάθε κάθετο marker υποδηλώνει και ένα νέο κύκλο ρολογιού

1. Διαφορές/Αλλαγές από το εργαστήριο 2

Για την υλοποίηση του τρίτου μέρους του εργαστηρίου του μαθήματος χρειάστηκε, πέρα από την προσθήκη του Reorder Buffer, να γίνουν αλλαγές στα Reservation Stations καθώς και στο Register File.

Αλλαγές στα Reservation Stations:
Αρχικά, προστέθηκε ένα ακόμη πεδίο εισόδου, το Rob_Tag_Accepted (μέχρι τώρα παίρναμε την τιμή του Tag από το RS, πλέον αυτό το αναλαμβάνει να μας το δίνει ο ROB), μεγέθους 5 bits και μας δείχνει το Tag στο οποίο έγινε push η εντολή που μόλις κάναμε issue στον Reorder Buffer. Επιπλέον, αυτή η πληροφορία αποθηκεύεται σε έναν register του κάθε RS ώστε να γνωρίζουμε σε ποια θέση του Reorder Buffer πρέπει να αναζητήσουμε, για να πάρουμε τη σωστή πληροφορία. Μόλις ολοκληρωθεί η εντολή, γίνεται RST αυτός ο register προκειμένου στο επόμενο issue, για την εντολή που θα γίνει αποδεκτή από το εκάστοτε RS, να αποθηκευτεί το νέο Tag του ROB. Παρακάτω, παρατίθεται το νέο σχηματικό από τον RS, στο οποίο όμως μπορούμε να παρατηρήσουμε ότι δεν υπάρχουν πολλές αλλαγές.



Σχηματικό Διάγραμμα 1.1.1 *- Reservation Station*

Αλλαγές στο Register File:
Χρειάστηκε να πραγματοποιηθούν και κάποιες αλλαγές στον RF. Πιο συγκεκριμένα, για την σχεδίαση εξακολουθούμε να αποθηκεύουμε πληροφορία για το Tag, πλέον αυτό που μας έρχεται από το ROB μαζί με το Value του καθενός καταχωρητή. Η συγκεκριμένη πληροφορία εκτός του ότι μας δείχνει την κατάσταση του Register (π.χ. ότι δεν είναι valid το Value μετά από το RST), επίσης μας δείχνει από ποιο Slot του ROB τελικά θα πάρει τιμή ο συγκεκριμένος register. ΠΡΟΣΟΧΗ, παρόλο που αποθηκεύουμε αυτήν την πληροφορία

πρέπει να γίνουν ξεκάθαροι οι λόγοι που χρησιμοποιείται. Αρχικά, το ότι υπάρχει δεν σημαίνει ότι ο ROB όταν θα χρειαστεί να γράψει θα αναζητά με βάση το TAG στην θέση την οποία πρέπει να γράψει, αντίθετα υπάρχει ειδικό πεδίο στα slots του ROB όπου αποθηκεύεται η διεύθυνση του register που θα γραφτεί πληροφορία. Ο λόγος που υπάρχει εκτός από το να συνεχίζουμε να βλέπουμε αν είναι valid το Value, είναι και για να επαληθεύσουμε ότι το Slot που θέλει να γράψει σε εμάς από το ROB, είναι το τελευταίο το οποίο έχει κάνει issue με τον ίδιο καταχωρητή εγγραφής, ώστε να ικανοποιείται το WAW. Τέλος, μας χρησιμεύει επίσης στο να αναζητούμε στον ROB κατά το issue μίας εντολής,αν οι register Rj και Rk, έχουν γίνει ήδη executed είτε νωρίτερα είτε την ώρα που γίνεται το issue ώστε να παίρνουν τιμές αμέσως, ακόμα και πριν γίνει το completion.

1.1 Reorder Buffer - ROB

Στην συνέχεια χρειάστηκε να σχεδιάσουμε τον ROB. Του οποίου η εξωτερική διεπαφή περιγράφεται από την παρακάτω πίνακα.

Σήμα	Μέγεθος (bits)	Είσοδος / Έξοδος	Περιγραφή
CLK	1	In	Ρολόι
RST	1	In	Επαναφορά
ISSUE	1	In	Σήμα Issue
ISSUE_PC	32	In	Το PC της Issued εντολής
ISSUE_I_type	2	out	Τύπος εντολής
ISSUE_Dest	5	out	Καταχωρητής που θα γίνει το Issue
ROB_TAG_ACCEPTED	5	in	Slot του ROB
ISSUE_RF_Rj	5	in	Καταχωρητής Rj που γίνεται Issue
ISSUE_RF_Rj_Exists	1	out	Σήμα για Valid τιμή του Slot του ROB
ISSUE_RF_Rj_Value	32	out	Τιμή του Slot για το Rj
ISSUE_RF_Rj_Tag	5	out	Tag του Slot για τον Rj
ISSUE_RF_Rk	5	in	Καταχωρητής Rk που γίνεται Issue
ISSUE_RF_Rk_Exists	1	out	Σήμα για Valid τιμή του Slot του ROB
ISSUE_RF_Rk_Value	32	out	Τιμή του Slot για το Rk

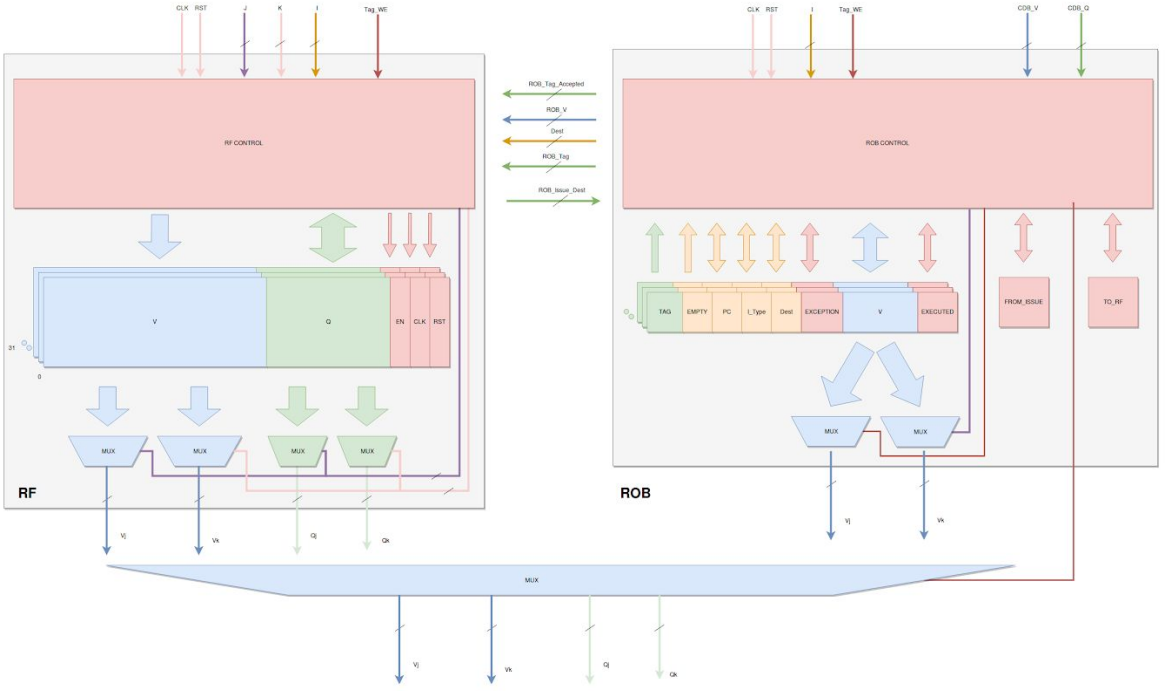
ISSUE_RF_Rk_Tag	5	out	Tag του Slot για τον Rk
CDB_Q	5	in	Είσοδος CDB_Q
CDB_V	32	in	Είσοδος CDB_V
DEST_RF	5	out	Καταχωρητής στον οποίο θα γράψω
DEST_MEM	5	out	Θέση μνήμης στην οποία θα γράψω
VALUE	32	out	Τιμή
WB_TAG	5	out	Tag του ROB που θέλει να γράψει στην RF/MEM
EXCEPTION_IN	5	in	Το exception που λαμβάνουμε
EXCEPTION	5	out	Σήμα εξαιρέσεων
PC	32	out	Είσοδος PC

Πίνακας 1.1.1 *- Διεπαφή Reorder Buffer.*

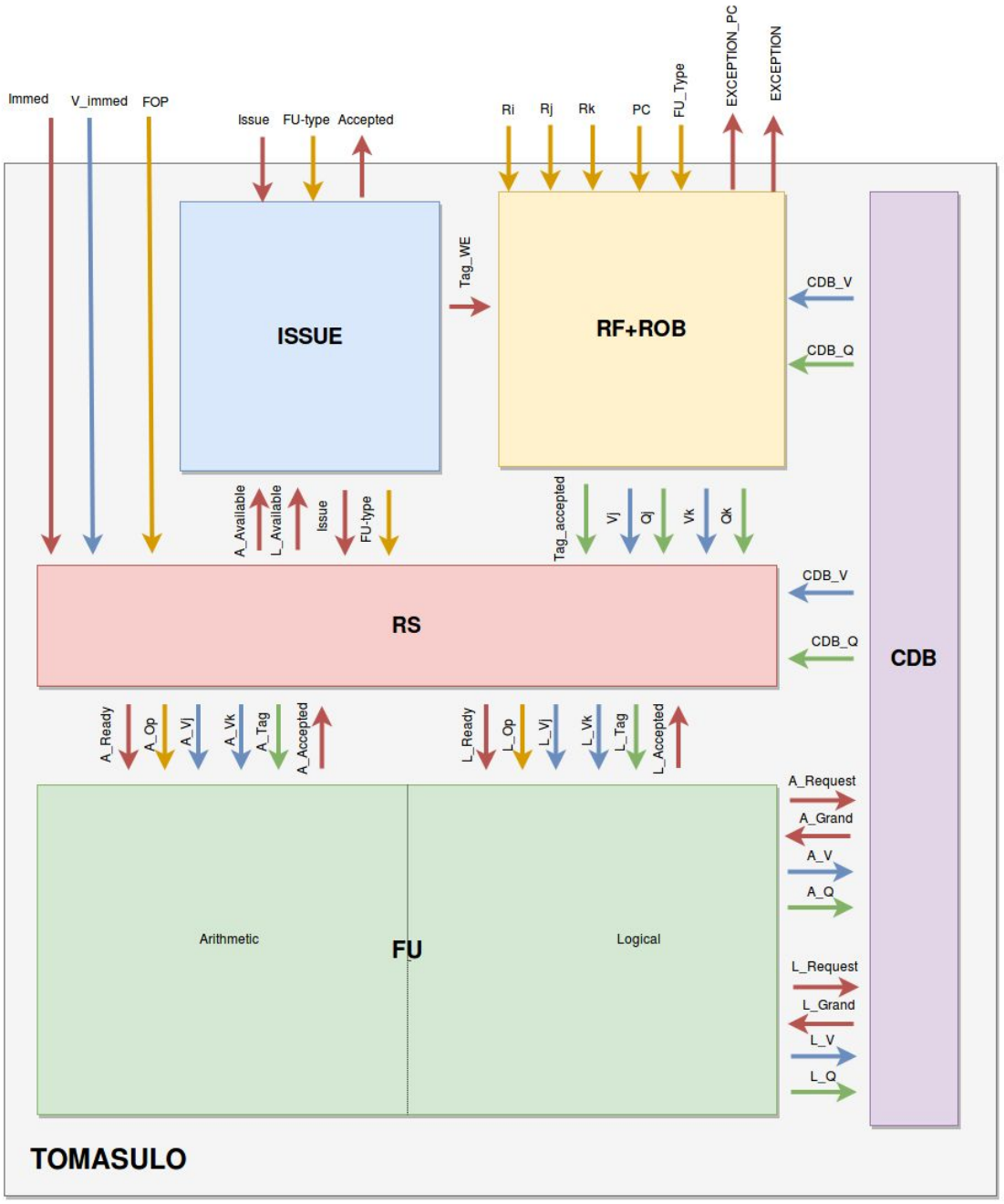
Λειτουργικότητα:

Σχετικά με την λειτουργία του ROB, θα πρέπει να την σπάσουμε ανά είδος σημάτων ώστε να αναλυθεί όσο το δυνατόν πιο κατανοητά. Αρχικά, έχουμε τα πορτοκαλί σήματα, όπου είναι τα σήματα τα οποία γίνονται push στον ROB. Συγκεκριμένα αποθηκεύουμε το PC της εντολής που θα γίνει issue, το type της καθώς και το σε ποιον register θα χρειαστεί να γράψουμε στην συνέχεια το αποτέλεσμα. Στην ίδια κατηγορία με αυτά, υπάρχει και το Tag του Slot στο οποίο θα προσθέσουμε τα παραπάνω σήματα ώστε να το χρησιμοποιήσουν ο RF και RS. Στην συνέχεια, έχουμε την κίτρινη κατηγορία από σήματα. Αυτά, τα χρησιμοποιούμε ώστε να ενημερώσουμε το ανώτερο επίπεδο αν έχουμε να δώσουμε έγκυρη πληροφορία στον RS, για τους registers ανάγνωσης που μόλις έγιναν issue (είτε μέσα στα πεδία του ROB, είτε γιατί μόλις μου ήρθε ένα αποτέλεσμα από την CDB). Συνεχίζοντας, υπάρχουν τα πράσινα σήματα τα οποία σχετίζονται με την διεπαφή του ROB με τον CDB. Ακολουθούν τα μπλέ σήματα όπου η χρήση τους είναι να μεταφέρουν την πληροφορία που χρειάζεται να γίνει commit είτε στην mem είτε στον RF. Και τέλος, είναι τα κόκκινα σήματα όπου η δουλειά τους σχετίζεται με τα exceptions που μπορεί να προκύψουν.
Επίσης, καλό είναι να σημειωθεί σε αυτό το σημείο ότι ο ROB υλοποιήθηκε από 30 μεμονωμένα Slots (προκειμένου να μην αλλάξουν υλοποίηση τα ήδη υπάρχοντα πεδία από Tag που είχαν μήκος 5 bits, ενώ 2 κωδικοποιήσεις χρησιμοποιούνται για να γνωρίζουμε την κατάσταση της πληροφορίας) τα οποία τόσο σχεδιαστικά όσο και λειτουργικά περιγράφονται παρακάτω. Ενώ κρατάμε δύο pointers όπου ο ένας δίνει πού πρέπει να κάνω το επόμενο issue και ο άλλος από πού πρόκειται να ελέγξω αν πρέπει να κάνω pop από το ROB queue.

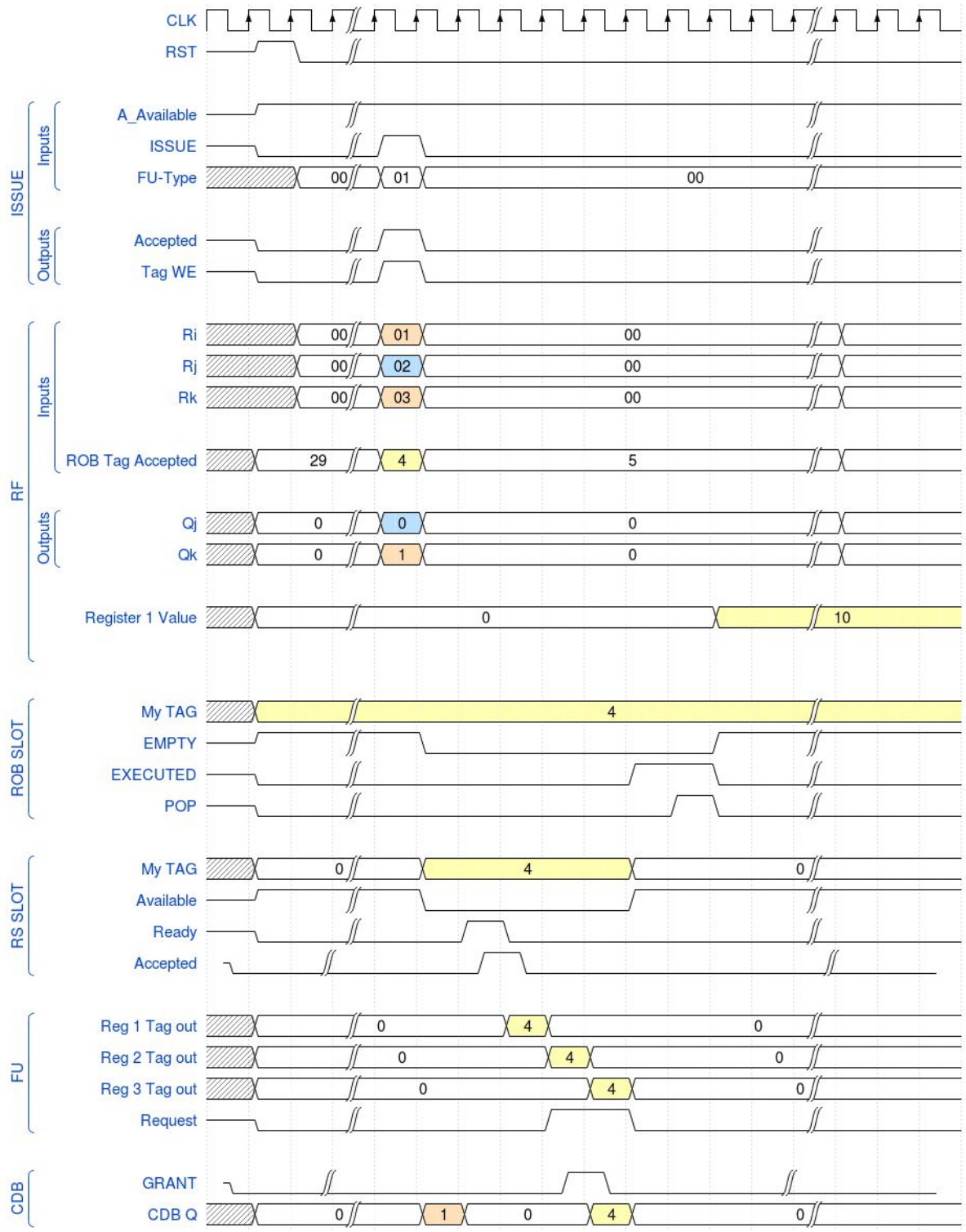
Προκειμένου να παραμείνουν σε μία ισορροπία από abstraction καθώς και σημαντικής ταυτόχρονα πληροφορίας στα παρακάτω σχεδιαγράμματα πρέπει να σημειωθούν τα εξής. Αρχικά, σχετικά με το Top Module, παρόλο που το πεδίο “RF+ROB” στην πραγματικότητα δεν είναι ένα κοινό component, από την στιγμή όπου “μοιράζονται” αρκετά κοινά αποφασίστηκε να παρουσιαστεί ως ένα, και στην συνέχεια να εμφανιστεί η εσωτερική του δομή. Επίσης, λόγο του ότι κάποια ονόματα σημάτων στην υλοποίηση του κώδικα ήταν πιο σύνθετα, στα σχεδιαγράμματα έχουν μετονομαστεί για να κρατηθεί απλά η βασική ιδέα. Τέλος, παρακάτω παρουσιάζεται και το σχηματικό του ROB που υλοποιήσαμε.



Σχηματικό Διάγραμμα 1.1.2 *- Register File and Reorder Buffer*



Σχηματικό Διάγραμμα 1.1.3 - *Tomasulo Top Module*



Σχηματικό Διάγραμμα 1.1.3 - *Timing Diagram*

Επίσης, παραπάνω παρατίθεται ένα ενιαίο timing Diagram στο οποίο εμφανίζονται σε μία πιο abstract μορφή τα πιο σημαντικά σήματα σε μία εκτέλεση τυχαίας εντολής όπου κατά το issue παρατηρούμε ότι το ένα από τα δύο πεδία που λαμβάνει είναι έτοιμο και το δεύτερο ενημερώνεται από τον CDB. Σκοπός δεν είναι τόσο να εξετάσουμε την συγκεκριμένη εντολή αλλά να παρατηρήσουμε ότι σύμφωνα με την σχεδιαση που πραγματοποιήσαμε όταν έχουμε να γράψουμε σε registers αυτό γίνεται κατά το rising edge του CLK, ενώ όλα τα

υπόλοιπα σήματα τόσο εισόδου όσο και ελέγχου σε αυτούς ενημερώνονται με συνδυαστική λογική ώστε μόλις έρθει το επόμενο rising edge να ενεργούν αμέσως χωρίς να χάνεται περισσότερος χρόνος.

1.2 Reorder Buffer Register - ROB Reg

Προκειμένου να πραγματοποιήσουμε τον ROB, αρχικά δημιουργήσαμε το module ROB_Reg, το οποίο περιλαμβάνει όλα τα σημαντικά πεδία τα οποία χρειάζεται κάθε slot του ROB, μαζί με σήματα ελέγχου.

Σήμα	Μέγεθος (bits)	Είσοδος / Έξοδος	Περιγραφή
CLK	1	In	Ρολόι
RST	1	In	Επαναφορά
MY_TAG	5	In	Tag κάθε Slot
ISSUE	1	In	Σήμα Issue
ISSUE_PC	32	In	Το PC της Issued εντολής
ISSUE_I_type	2	in	Τύπος εντολής
ISSUE_Dest	5	in	Καταχωρητής που θα γίνει το Issue
CDB_Q	5	in	Είσοδος CDB_Q
CDB_V	32	in	Είσοδος CDB_V
I_EXCEPTION	5	in	Σήμα Εξαίρεσης Slot
EXECUTED	1	out	Σήμα Executed
POP	1	in	Σήμα Pop
DEST_RF	5	out	Καταχωρητής στον οποίο θα γράψω
DEST_MEM	5	out	Θέση μνήμης στην οποία θα γράψω
VALUE	32	out	Τιμή Slot
EXCEPTION	5	out	Σήμα Exception
PC	32	out	Σήμα PC
EMPTY	1	out	Σήμα Empty
TAG	5	out	Σήμα TAG

Πίνακας 1.2.1 *- Διεπαφή ROB Register - ROB Reg.*

Λειτουργικότητα:

Στην πραγματικότητα το συγκεκριμένο component είναι ένας συνδυασμός από Registers, οι οποίοι ελέγχονται από το Control του ROB με κατάλληλο τρόπο. Συνοπτικά, θα αναφέρουμε ότι τα δύο σημαντικά σήματα ελέγχου είναι το ISSUE και το POP. Όταν έχουμε ISSUE από την εξωτερική μονάδα μίας εντολής και θέλουμε να γράψουμε στην συγκεκριμένο Register ενεργοποιούμε το συγκεκριμένο σήμα προκειμένου να γράψουμε σε αυτόν. Ενώ όταν δούμε ότι ο είναι σειρά του συγκεκριμένου Slot να γίνει pop ενεργοποιούμε το σήμα POP. Για να είναι ένα Slot έτοιμο προς completion, αναγκαστικά πρέπει να έχει γίνει issue σε αυτό και μέσω του CDB να έχει κάνει update τα πεδία του προκειμένου να έχει γίνει EXECUTED, εναλλακτικά μπορεί να δημιουργηθεί σε οποιαδήποτε στιγμή σε αυτό κάποιο exception.

Testing:

Προκειμένου να ελέγξουμε την λειτουργικότητα του μεμονωμένου slot πραγματοποιήσαμε τις παρακάτω instructions και περιμέναμε σύμφωνα με την σχεδίαση τα παρακάτω αποτελέσματα.

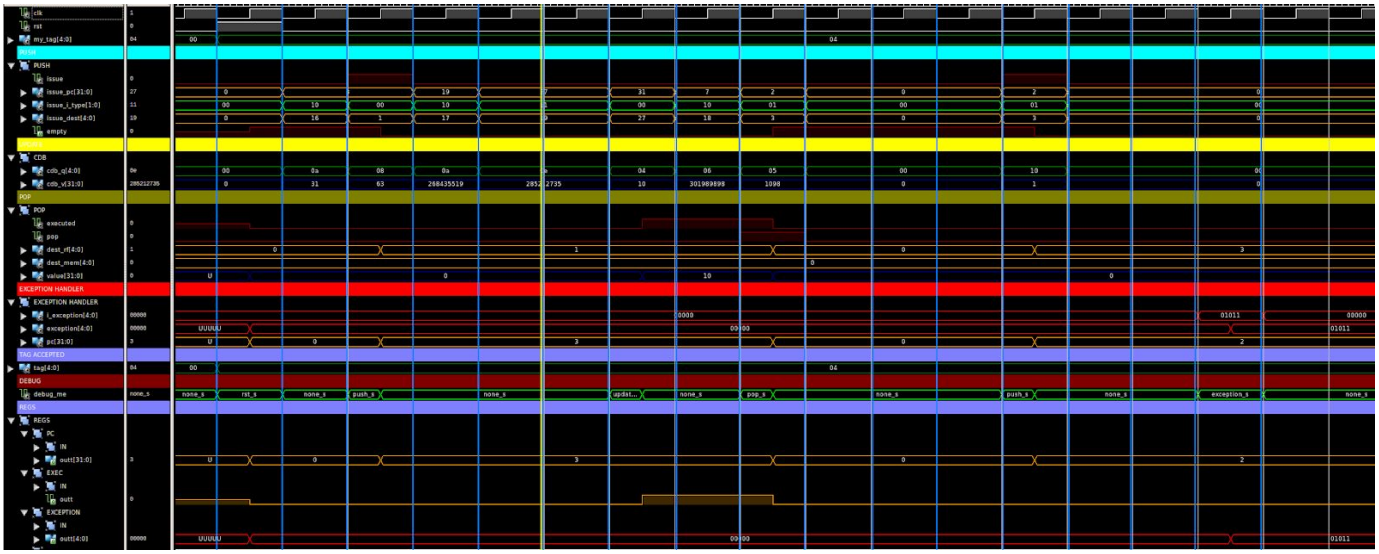
Inputs

CC	Pseudocode (MY TAG=00100)
1	RESET
2	NOP (IS EMPTY)
3	ISSUE Arithmetical with Ri=1
4	NOP (NON EMPTY) x3
7	CDB_Q=00100 and CDB_V=10
8	NOP (NON EMPTY)
9	POP
10	NOP (IS EMPTY) x3
13	ISSUE
15	NOP (IS EMPTY)
16	EXCEPTION=01011

Expected Outputs

CC	Results
1	RESET
4	EMPTY=0 + PUSHED PC, FU-type and DEST
8	EXECUTED=1 + UPDATED VALUE FROM CDB_V
10	EMPTY=1 + COMMIT DATA
14	EMPTY=0 + PUSHED PC, FU-type and DEST
17	EXCEPTION=01011

‘Οπου μπορούμε στην παρακάτω κυματομορφή να επαληθεύσουμε την λειτουργία του ROB_Reg η οποία ακολουθεί πλήρως την λογική του παραπάνω πίνακα.



Εικόνα 1.2.2 *- Simulation ROB Register - ROB Reg*

Στην συνέχεια και πριν προχωρήσουμε στην συνένωση των components αυτήν την φορά με την χρήση του ROB, δοκιμάσαμε ότι αρχικά ο ROB, ενεργεί ακριβώς έτσι όπως πρέπει. Συγκεκριμένα, θέλαμε να κάνουμε την ίδια διαδικασία με το ROB_Reg, όπου θα ελέγχουμε βάση σχεδίασης τι πρόκειται να περιμένουμε και τα τελικά αποτελέσματα τα οποία λαμβάνονται από συγκεκριμένες εντολές. Σε αυτήν όμως την πειραματική διαδικασία πρέπει να αναφερθούν με ποια λογική σχεδιάσαμε το πρόγραμμα το οποίο θα κάναμε test. Αρχικά, θέλαμε να σιγουρευτούμε ότι μπορούμε να κάνουμε issue στους registers (Push στον ROB). Στην συνέχεια, αυτό που θέλαμε να σιγουρευτούμε είναι ότι οι registers ανανεώνουν τις τιμές τους από τον CDB. Σε αυτό το σημείο πρέπει να σημειώσουμε και την δομή του queue. Το οποίο δεν είναι παρά ένα array στο οποίο “γράφουμε/διαβάζουμε/σβήνουμε” στις κατάλληλες

θέσεις. Από την στιγμή που θα χρησιμοποιούσαμε 2 registers για να κάνουμε push και pop έπρεπε επίσης σε αυτό το test να σιγουρευτούμε ότι μπορούν να κινηθούν κυκλικά (για αυτό το λόγο έχουμε να ξεκινάνε ήδη από την αρχή της λειτουργίας τους στο τέλος του πίνακα για να φανεί αυτή η ικανότητα ήδη από τα πρώτα push). Στην συνέχεια, έπρεπε να μπορούμε να επαληθεύσουμε ότι αν χρειαστεί να κάνουμε issue μία εντολή και η τιμή ενός ή και των δύο από τους register έχει ήδη υπολογιστεί ή πρόκειται να υπολογιστεί στον ίδιο κύκλο από την ROB, να μπορούμε αμέσως να τους δώσουμε τιμή, και τέλος να σιγουρευτούμε ότι σε περίπτωση exception θα γίνεται reset των κατάλληλων registers.

Testing:

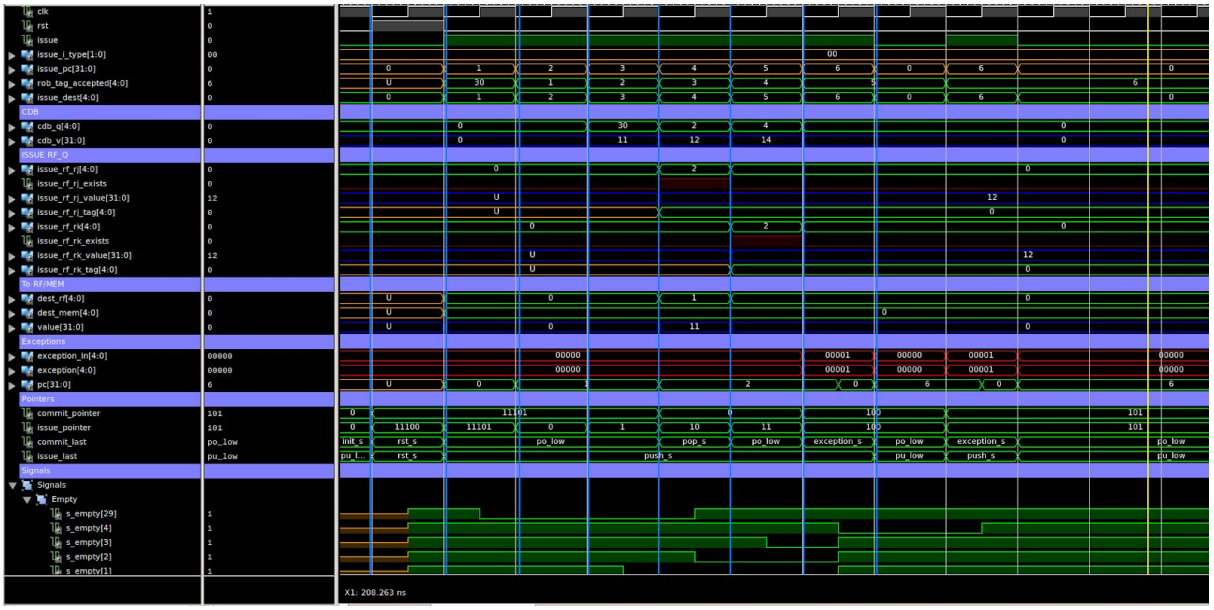
Προκειμένου να ελέγξουμε την λειτουργικότητα του μεμονωμένου slot πραγματοποιήσαμε τις παρακάτω instructions και περιμέναμε σύμφωνα με την σχεδίαση τα παρακάτω αποτελέσματα.

Inputs

CC	Pseudocode
1	RST
2	1’ ISSUE (ROG_TAG=30, ISSUE_POINTER=29)
3	2’ ISSUE (ROG_TAG=1, ISSUE_POINTER=0)
4	3’ ISSUE (ROG_TAG=2, ISSUE_POINTER=1)
5	4’ ISSUE (ROG_TAG=3, ISSUE_POINTER=2) + CDB_Q=30, CDB_V=12 + ISSUE Rj_Q=CDB_Q
6	5’ ISSUE (ROG_TAG=4, ISSUE_POINTER=3) + CDB_Q=4, CDB_V=14 + ISSUE Rk_Q=ROB_TAG(1)
7	6’ ISSUE (ROG_TAG=3, ISSUE_POINTER=4) +EXCEPTION
8	NOP
9	7’ ISSUE (ROG_TAG=4, ISSUE_POINTER=5) + EXCEPTION

Expected Outputs

CC	Results
2	ISSUE ON ROB(29) + ISSUE_POINTER=0
3	ISSUE ON ROB(0) + ISSUE_POINTER=1
4	ISSUE ON ROB(1) + ISSUE_POINTER=2
5	ISSUE ON ROB(2) + ISSUE_POINTER=3 + UPDATE ROB(29) + RF_RJ_EXISTS='1' + RF_RJ_Value=CDB_V
6	ISSUE ON ROB(3) + ISSUE_POINTER=4 +RF_Rk_EXISTS='1' + RF_Rk_Value=CDB_V
7	ISSUE ON ROB(4) + ISSUE_POINTER=5 +EXCEPTION (RST REGS) +RST regs
8	Nop
9	ISSUE ON ROB(5) + ISSUE_POINTER=6 + EXCEPTION (RST REGS) +RST regs



Εικόνα 1.2.3 *- Simulation ROB*

Από το παραπάνω test πρόγραμμα πρέπει να επισημάνουμε κάποια σημαντικά σημεία. Αυτό που θέλουμε αρχικά είναι να επαληθεύσουμε ότι κάνει in order issue, παρόλο που δεν φαίνονται τα σήματα ενεργοποίησης του issue για το κάθε slot μπορούμε να επαληθεύσουμε την ορθή λειτουργία παρατηρώντας τον issue_pointer και το issue_last (βοηθητικό debug signal, χωρίς φυσική σημασία παρόλα αυτά επειδή είναι μέσα στα if statements που παίρνονται οι αποφάσεις μπορεί να μας ενημερώσει πλήρως). Ώστε να δούμε πότε και σε

ποιο slot θα γίνει το issue. Άλλο ένα πράγμα το οποίο δοκιμάζουμε είναι στον 5 και στον 6 κύκλο να παίρνουμε πληροφορία από τον ROB που δεν έχει ακόμα γίνει commit κατά το issue. Στον 5 κύκλο, αυτή η τιμή παίρνετε από τον CDB ενώ στον 6 παίρνετε η ίδια τιμή αλλά αυτήν την φορά από τον ROB. Άλλο ένα σημαντικό κομμάτι είναι το γεγονός ότι γίνεται Pop από την Queue όταν η τιμή του executed στο στοιχείο που δίνει ο committ_pointer ενεργοποιηθεί. Και τα δύο τελευταία σημαντικά σημεία, είναι πρώτον στον 8 κύκλο ότι αδειάζει ότι έχει μετά από αυτό το στοιχείο που έφερε το exception η queue, και στην συνέχεια είναι σε θέση να συνεχίσει κανονικά την λειτουργία της.

2 Έλεγχος Συστήματος

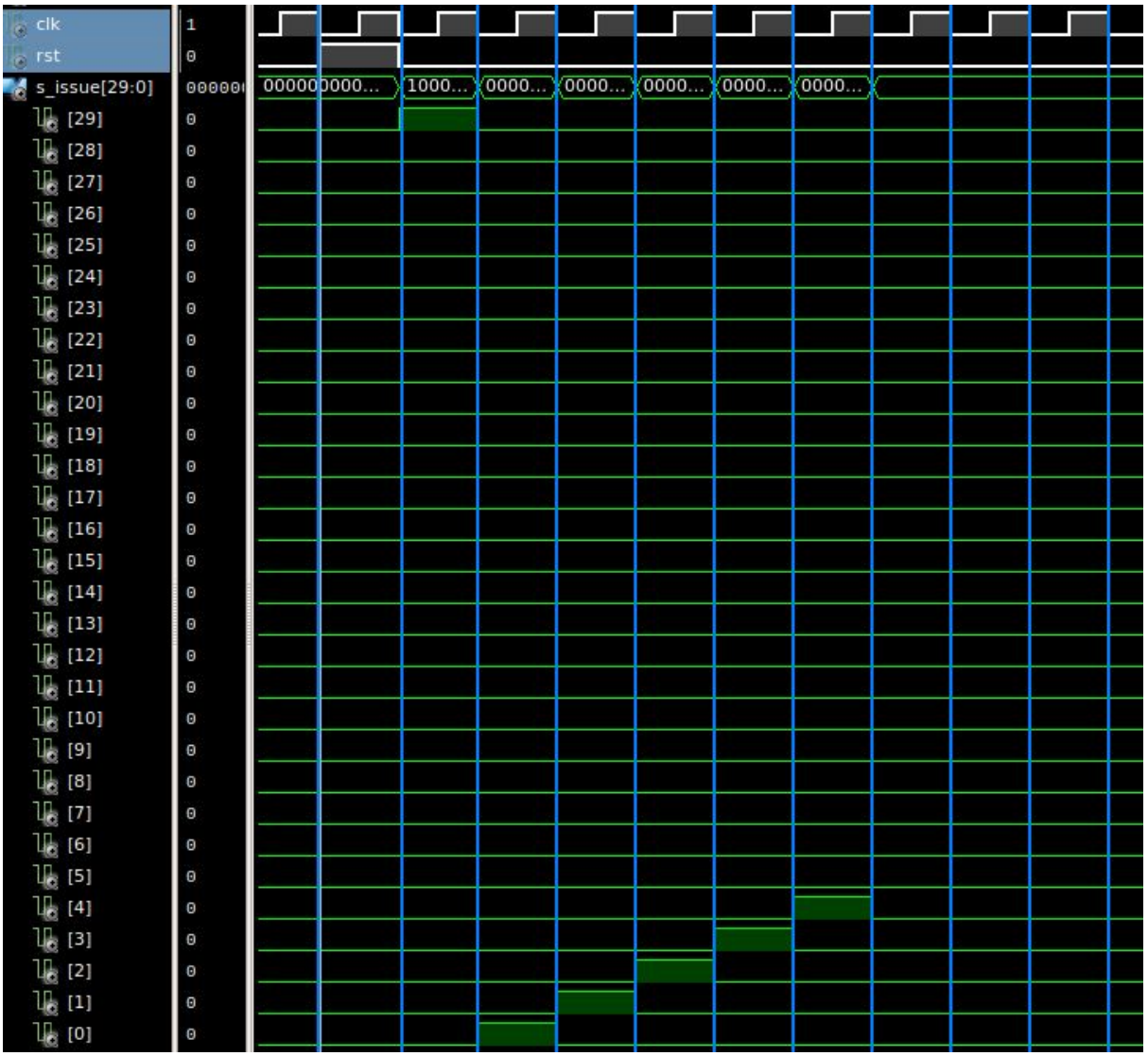
Στην συνέχεια και αφού γνωρίζαμε ότι ο ROB λειτουργεί ανεξάρτητα συνεχίσαμε με το να τον δοκιμάσουμε και στο σύνολο του με τα άλλα components. Από την στιγμή που έχει δοκιμαστεί σε κάθε ένα κομμάτι ξεχωριστά ήδη ο ROB, ο μόνος έλεγχος που θα παρουσιαστεί σε αυτό το μέρος είναι ότι μπορεί να αλληλεπιδράσει σωστά με όλα τα υπόλοιπα υποσυστήματα σύμφωνα με τα παραπάνω που έχουμε αναφέρει. Το σύνολο των εντολών με τις οποίες θα ελέγχουμε το σύστημα είναι το παρακάτω.

CC	ID	Instr	Assembly	RF
1	-	RST	-	
2	i1	addi	$1,$0,1	R1=1
3	i2	ori	$2,$1,2	R2=3
4	i3	addi	$3,$2,-1	R3=2
5	i4	ori	$4,$0,4	R4=5
6	i5	addi	$5,$0,5	R5=5
7	i6	addi	$6,$0,6	R6=6

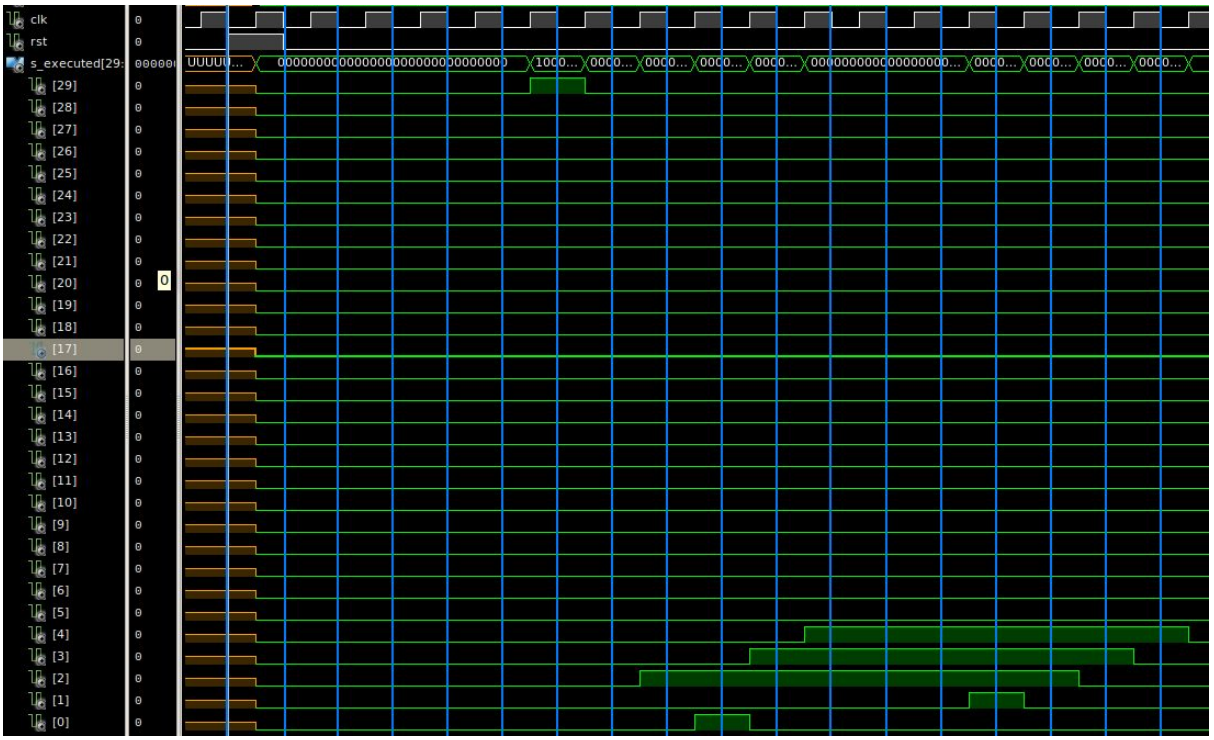
Επίσης, τόσο με βάση το διάγραμμα χρονισμού όσο και την σχεδίαση μας γνωρίζουμε ότι πρώτα θα υπολογιστεί η i1, μετά η i4, μετά η i2, μετά η i5, μετά η i6 και τελευταία η i3 λόγω των εξαρτήσεων που υπάρχουν. Συνεπώς, δοκιμάζοντας αυτό το σύνολο εντολών με το συνδυασμό Tomasulo + ROB, θα πρέπει να παρατηρήσουμε τα εξής:

- In order Issue στον ROB
- Out of order execution στον ROB
- In order completion από τον ROB (το να γίνονται οι εντολές pop)
- Επιτυχώς in order completion στην εγγραφή στην RF.

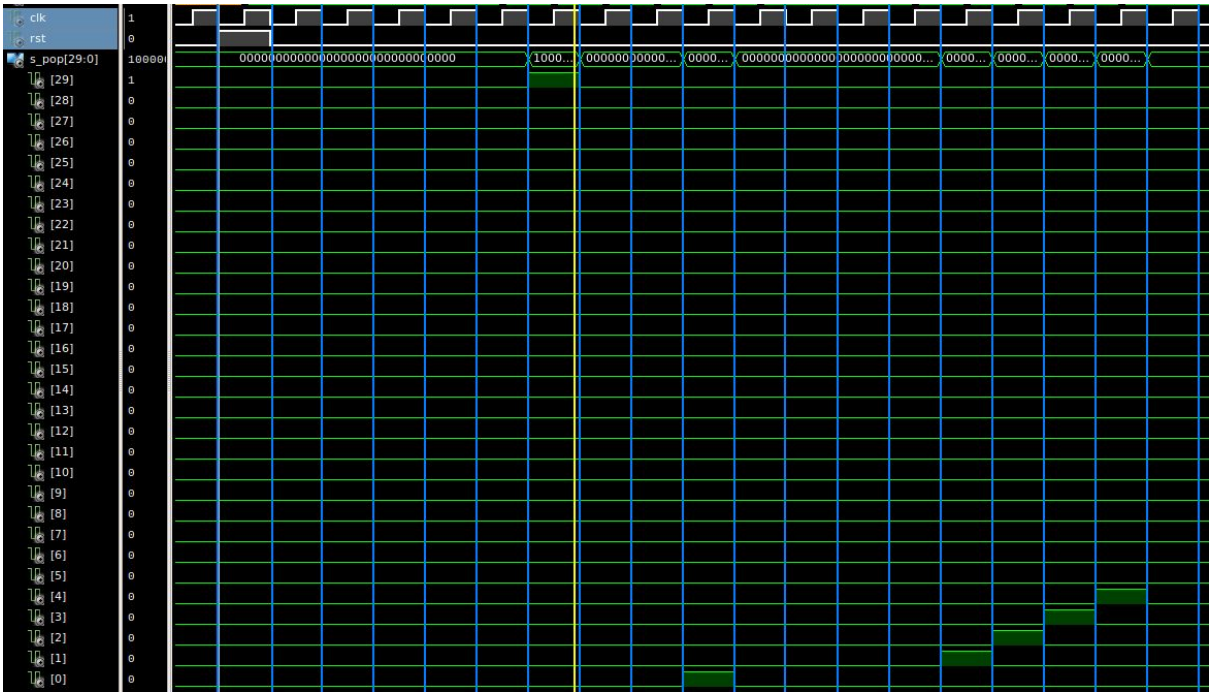
Παρακάτω παρατίθενται οι κυματομορφές από τις οποίες μπορούμε να δούμε ότι καθένα από τα παραπάνω πληρείται.



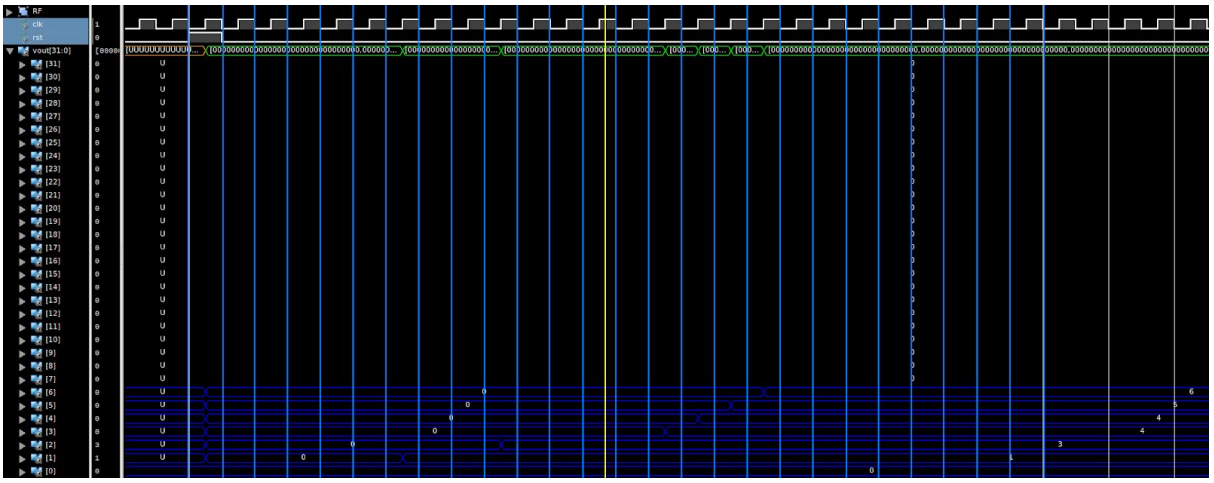
Εικόνα 2.2.1 *- Simulation ROB - ROB_Slots issue (in order)*



Εικόνα 2.2.2 *- Simulation ROB - ROB_Slots execution(out of order)*



Εικόνα 2.2.3 *- Simulation ROB - ROB_Slots pop(in order)*



Εικόνα 2.2.4 *- Simulation RF - Registers values(in order commit)*