Computer Architecture

Tomasulo's Algorithm

Technical University of Crete

3rd Milestone

Ομάδα:

Ονοματεπώνυμο: Κυπαρισσάς Νικόλαος
Α.Μ.: 2012030112

Ονοματεπώνυμο: Καμπυλαυκάς Αναστάσιος
Α.Μ.: 2012030143

21/12/2017

Ο Αλγόριθμος του Tomasulo

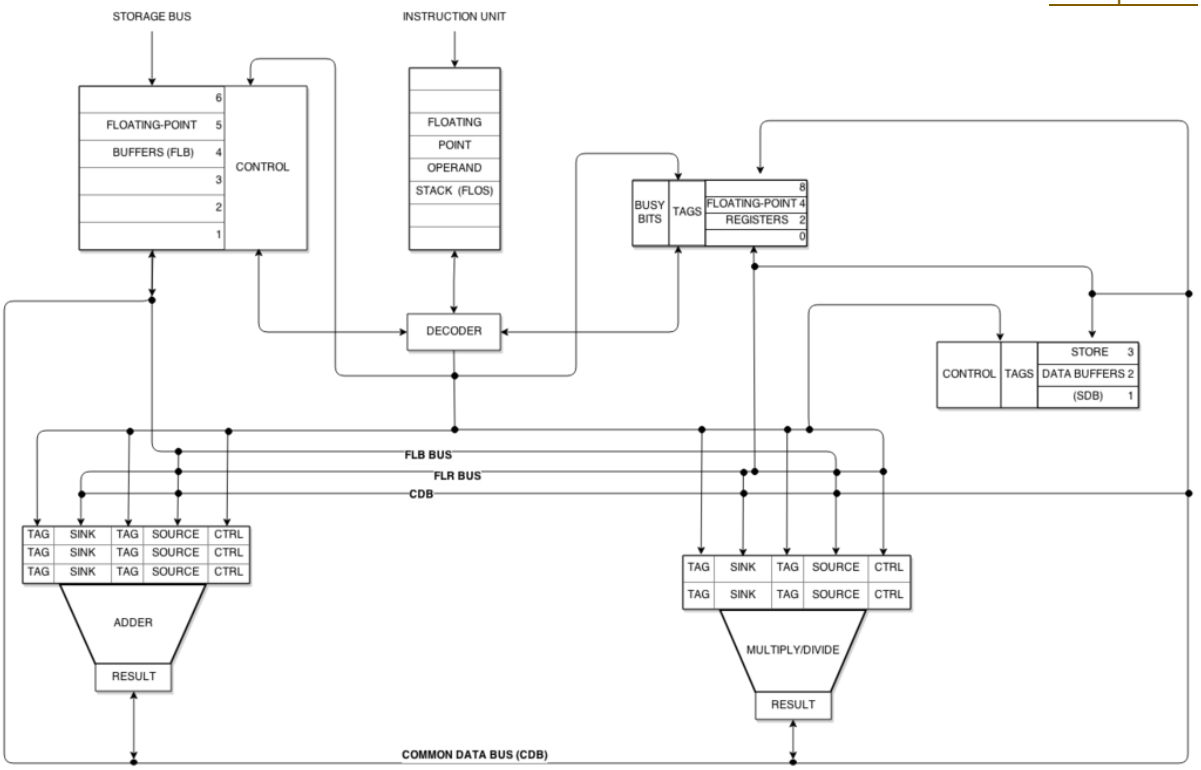
Ο αλγόριθμος του Tomasulo είναι ένας αλγόριθμος αρχιτεκτονικής υπολογιστών με σκοπό τη δυναμική χρονοδρομολόγηση εντολών που επιτρέπει την εκτέλεσή τους εκτός σειράς και συντελεί στην αποδοτική χρήση των υπολογιστικών μονάδων του επεξεργαστή.

Στις καινοτομίες του αλγορίθμου συμπεριλαμβάνονται οι τεχνικές *Register Renaming* σε hardware, τα *Reservation Stations* για όλες τις υπολογιστικές μονάδες *(Functional Units),* και ο κοινός δίαυλος δεδομένων *(Common Data Bus - CDB)* όπου ανακοινώνονται (broadcast) σε όλα τα *Reservation Stations* οι τιμές που υπολογίζονται.

Οι τεχνικές αυτές επιτρέπουν βελτιωμένη παράλληλη εκτέλεση εντολών και αποτρέπουν την αδρανοποίηση εντολών (stalling) που είναι απαραίτητη κατά τη χρήση του *Scoreboard* ή άλλων παλαιότερων αλγορίθμων.

Ο Robert Tomasulo έλαβε το βραβείο Eckert-Mauchly το 1997 για τον σχεδιασμό του εν λόγω αλγορίθμου.

Wikipedia



Σκοπός του project είναι η υλοποίηση του *αλγορίθμου του Tomasulo* σε Hardware χρησιμοποιώντας *VHDL (VHSIC Hardware Description Language)* σε τρία επιμέρους milestone.

Πρώτο Milestone

Το πρώτο milestone περιλαμβάνει την υλοποίηση και τον έλεγχο λειτουργίας όλων των επιμέρους βασικών components του *αλγορίθμου του Tomasulo*.

Αυτά είναι τα ακόλουθα:

i. *Instruction Issue (IF-Issue)*
ii. *Register File (RF)*
iii. *Reservation Stations + Functional Units (RS)*
iv. *Common Data Bus (CDB)*

Δεύτερο Milestone

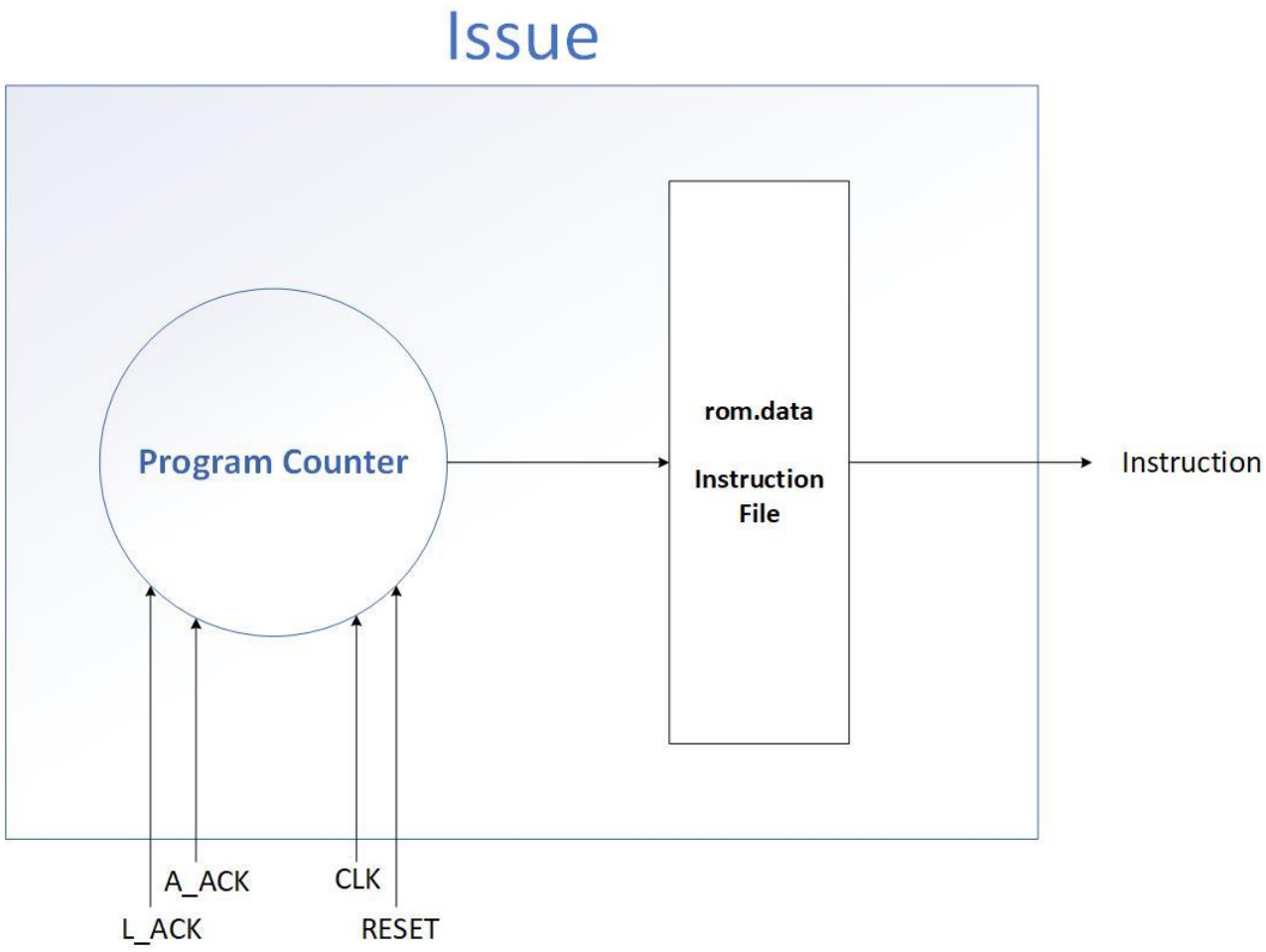
Στη δεύτερη φάση του project (δεύτερο milestone) θα γίνει η σύνδεση των τεσσάρων παραπάνω βασικών components και ο συγχρονισμός τους ώστε να λειτουργούν συντονισμένα ως ένα σύστημα.

Τρίτο Milestone

Στη τρίτη φάση του project προστίθεται στο σύστημα ένα *Reorder Buffer - ROB (FIFO)*, το οποίο φροντίζει για την σε σειρά επενέργεια (*Commit*) των εντολών στο σύστημα. Οι εντολές εξακολουθούν να εκτελούνται εκτός σειράς, δεν ισχύουν όμως για το σύστημα παρά μόνο όταν έρθει η ώρα να γίνουν Commit.

ISSUE (Instruction Issue)

Η μονάδα *Instruction Issue* είναι το component του project που παρέχει στο σύστημα τις εντολές που θα δρομολογηθούν στα επιμέρους *Functional Units.* Το component αυτό αποτελείται από δύο άλλα μικρότερα components, το αρχείο *rom.data* και το *INSTRUCTION_FETCH.vhd* αρχείο, το οποίο διαχειρίζεται το *rom.data* και περιέχει τον *Program Counter*. Το *Rom.data* είναι το αρχείο στο οποίο βρίσκονται καταχωρημένες οι εντολές του συστήματος ενώ *ο Program Counter* είναι ένας μετρητής που δείχνει σε κάθε δεδομένη χρονική στιγμή την διεύθυνση της εντολής προς έκδοση.



Σχηματικό διάγραμμα του Issue component με τις εισόδους και τις εξόδους του.

Interface

Το interface του *Issue* component είναι ιδιαίτερα απλό. Κάθε χρονική στιγμή το *Issue* εκδίδει μια εντολή στο σύστημα, η οποία γίνεται δεκτή (υπό συνθήκες) από τα επιμέρους *Functional Units.* Η εντολή 1111111111111111111 (19 bits) είναι η εντολή *NULL* (βάσει σύμβασης) και δεν γίνεται δεκτή από κανένα *Functional Unit*, συνεπώς το σύστημα σταματάει μόλις εμφανιστεί στην είσοδό του αυτή η εντολή. (Ως εντολή *NULL* είναι αντιμετωπίζεται οποιαδήποτε εντολή που αρχίζει με "11" ή "10" - δύο πρώτα bits - καθώς δεν υπάρχει στο σύστημά μας *Functional Unit με* κωδικό 11 ή 00 και συνεπώς το τελευταίο δεν υποστηρίζει και τις αντίστοιχες πράξεις)

Είσοδοι

Το *Issue* component δέχεται δύο σήματα επιβεβαίωσης (*A_ACK* και *L_ACK*), από τα οποία μόνο ένα (το πολύ) είναι ενεργό σε κάθε κύκλο. Το καθένα ενεργοποιείται από την μονάδα ελέγχου του αντίστοιχου *Reservation Station* που δέχεται την εντολή.

Κάθε φορά που ένα σήμα επιβεβαίωσης ενεργοποιείται, μία νέα εντολή ανακοινώνεται στον επόμενο κύκλο μέχρι κάποιο *Reservation Station* να την δεχτεί.

Έξοδοι

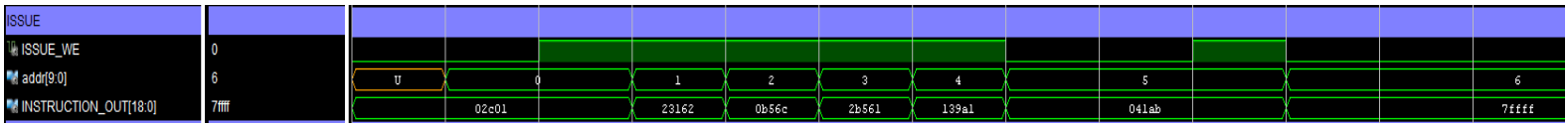
Η μόνη έξοδος του *Issue* είναι η εντολή η οποία δεν έχει ακόμη γίνει δεκτή από κάποιο *Reservation Station*.

Αξίζει να σημειωθεί ότι το σύστημα είναι ικανό να παρέχει νέα εντολή ανά κύκλο ρολογιού, χάρη στο ότι τα κυκλώματα που παράγουν τα σήματα

επιβεβαιώσεων είναι συνδυαστικά και η *Register File* είναι σχεδιασμένη Write-first.

Με αυτόν τον τρόπο το *Issue* ενημερώνεται στον ίδιο κύκλο για το αν η εντολή που δίνει στην έξοδό του γίνεται δεκτή (από κάποιο *Functional Unit*), και αν ναι, στον επόμενο κύκλο βγάζει νέα εντολή στην έξοδό του.

Behavioral Simulation - Issue



(Issue_We = A_ACK OR L_ACK)

Ουσιαστικά το σήμα Issue_We δηλώνει αν κάποιο (οποιοδήποτε, χωρίς να ξέρουμε ή να μας νοιάζει ποιο) *Reservation Station* έκανε δεκτή την εντολή που εκδίδεται αυτή τη στιγμή από το Issue και επομένως πρέπει στον επόμενο κύκλο να εκδοθεί (ή όχι) καινούρια εντολή.

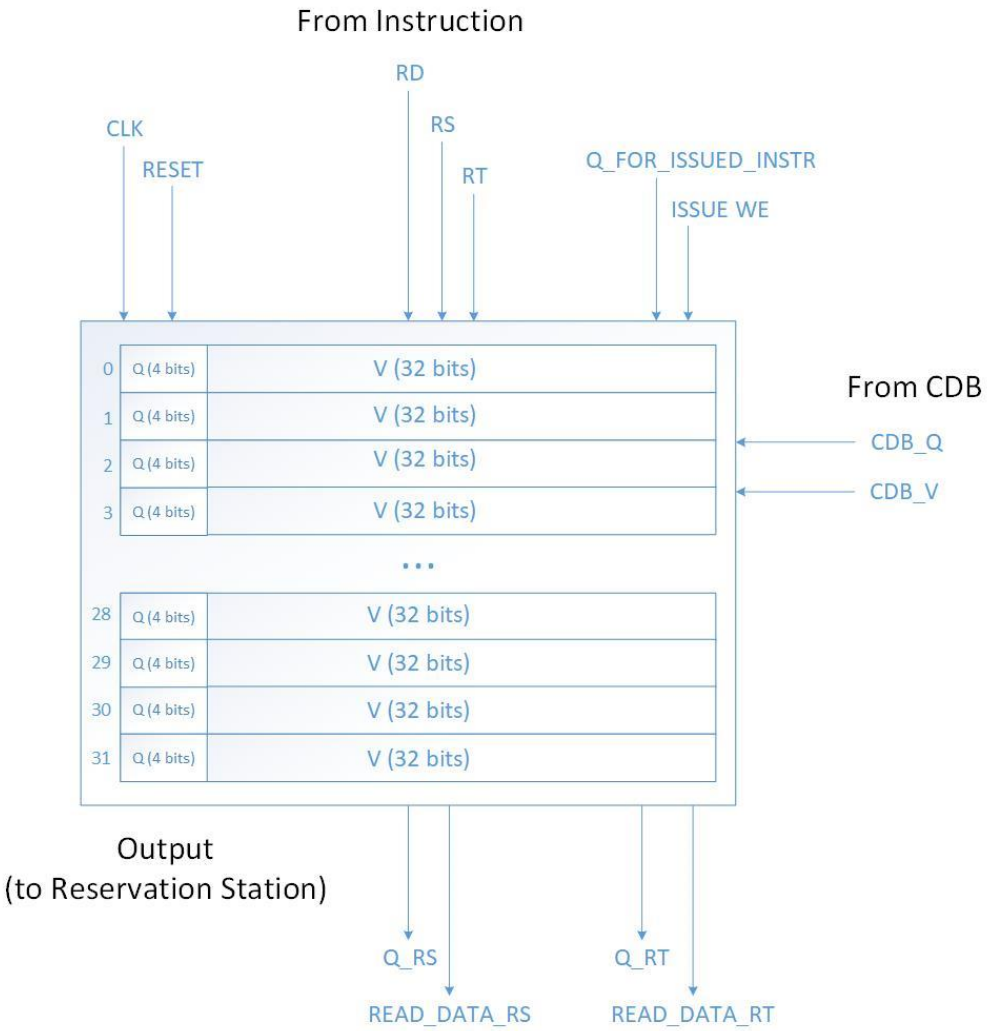
Instruction Format

Το format των εντολών του επεξεργαστή μας είναι το ακόλουθο (19 bits):

18-17	16 - 15	14 - 10	9 - 5	4 - 0
Arithmetic/Logic	FOP	RD	RS	RT

RF (Register File)

Η *Register File* είναι μια τυπική, *Write-first Register File*. Η μόνη αξιοσημείωτη τροποποίηση είναι ότι "γράφεται" μόνο από το *Common Data Bus (CDB)* και μόνο εφόσον το *Tag* της τιμής που ανακοινώνεται από το *CDB* ταιριάζει με το *Tag* κάποιου καταχωρητή της.



Σχηματικό διάγραμμα της Register File με τις εισόδους και τις εξόδους της.

Σε κάθε κύκλο ρολογιού η *Register File* δέχεται μια εντολή από το *Issue.*

Interface

Είσοδοι

Η *RF* δέχεται ως είσοδο τους τρείς καταχωρητές (*RD,RS* και *RT*) από το *Issue* καθώς και τα *A_ACK* και *L_ACK* από τα *Reservation Stations*, τα οποία περνάει από μια πύλη *OR (Issue_We signal)* και με αυτόν τον τρόπο γνωρίζει σε κάθε χρονική στιγμή αν κάποια εντολή εκδίδεται (αγνοώντας το ποιος τη λαμβάνει). Όταν το *Issue_We* είναι ενεργό η *RF* "γράφει" στο *Tag* του καταχωρητή *RD* το αντίστοιχο Tag που "παίρνει" από την λειτουργική μονάδα που δέχεται την εντολή.

Τέλος η *RF* είναι συνδεδεμένη και με την έξοδο του *CDB* (*CDB_V* και *CDB_Q*), το οποίο συνδέεται στη μοναδική πύλη εγγραφής της *RF*. Όταν η *RF* δεχτεί κάποιο έγκυρο *Q* από το *CDB* (όχι 1111 ή XXXX) τότε ελέγχει όλα τα *Tags* των καταχωρητών της και αν βρει κάποιο που να συμπίπτει με αυτό που κοινοποιεί το *CDB* τότε γράφει τα δεδομένα του (*CDB_V*) στον αντίστοιχο καταχωρητή, κάνοντας ταυτόχρονα το *Tag* του 1111, που σημαίνει πως πλέον αυτός ο καταχωρητής έχει έγκυρα δεδομένα. Αν πάλι δεν βρει κοινό *Tag*, αγνοεί το *CDB_V* καθώς ο καταχωρητής στον οποίο ήθελε να γράψει το τελευταίο έχει "πανωγραφθεί" από κάποιο άλλο *Reservation Station*.

Έξοδοι

Οι έξοδοι της *RF* είναι μόνο τα δεδομένα των καταχωρητών *RS* και *RT* καθώς και τα αντίστοιχα *Q* τους, τα οποία οδηγούνται στα *Functional Units*. Τα *Q* είναι αυτά που προσδιορίζουν αν τα *data* που μόλις βγήκαν από την *Register File* είναι έγκυρα ή πρέπει να περιμένουν να βγουν από κάποιο *Reservation Station* (*Tag* = 1111 για έγκυρα δεδομένα και οτιδήποτε άλλο για μη έγκυρα δίνοντας ταυτόχρονα και το *Reservation Station* από το οποίο τα περιμένουν).

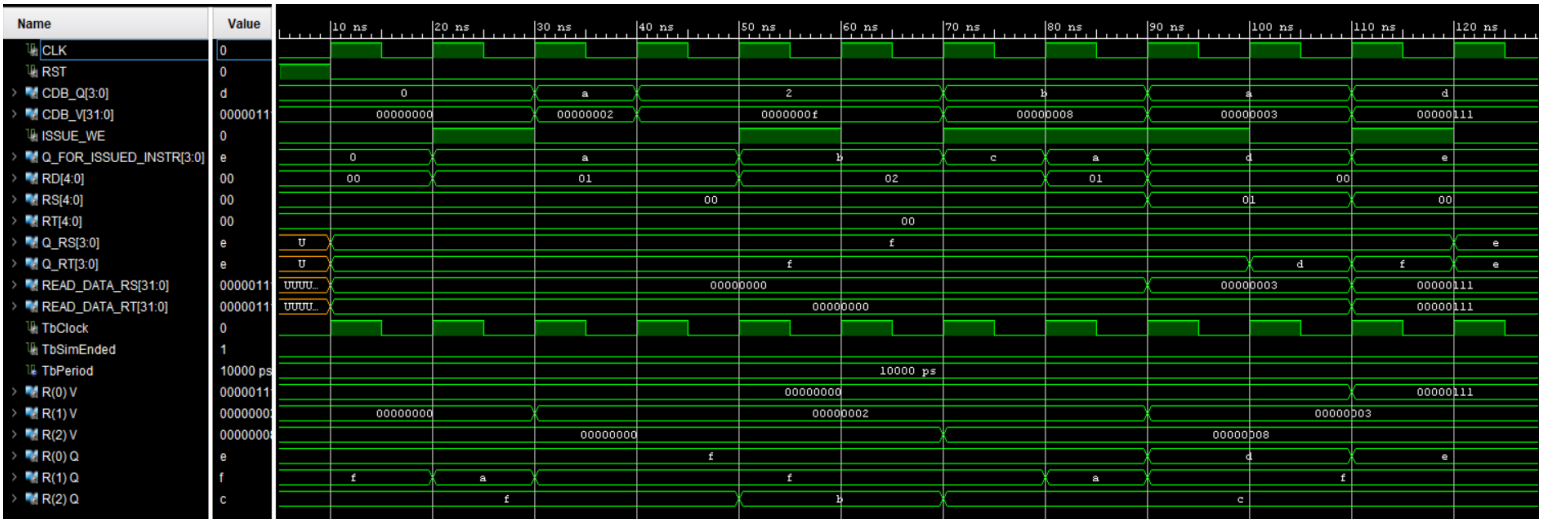
Τρείς αξιοσημείωτες περιπτώσεις είναι οι ακόλουθες:

a) Όταν ο καταχωρητής *RD* της εκδιδόμενης εντολής ταυτίζεται με τον καταχωρητή που "γράφεται" αυτή τη στιγμή στην *RF* από το *CDB*, όπου "γράφουμε" το *Q* που στέλνει το *Reservation Station* που δέχεται την εντολή στο αντίστοιχο πεδίο της *RF* γράφοντας ταυτόχρονα και την τιμή που έρχεται από το *CDB*, χωρίς όμως να κάνουμε το αντίστοιχο *Q (Tag)* 1111. Με αυτόν τον τρόπο εξασφαλίζουμε πως τα δεδομένα που βρίσκονται αυτή τη στιγμή στον συγκεκριμένο καταχωρητή δεν είναι έγκυρα και πως όποιος (από εδώ και στο εξής) ζητήσει τα δεδομένα αυτά θα πρέπει να περιμένει να τα πάρει από το *CDB* όταν ολοκληρωθεί η εντολή που μόλις εκδόθηκε και κοινοποιήσει τα δεδομένα της μέσω αυτού.

b) Ένας ή και οι δύο καταχωρητές *RS* ή *RT* της εκδιδόμενης εντολής ταυτίζονται με τον καταχωρητή που "γράφεται" αυτή τη στιγμή στην *RF* από το *CDB*, όπου στην έξοδο της *RF* δίνουμε τα δεδομένα που γράφονται εκείνη τη στιγμή από το *CDB* και ταυτόχρονα γράφονται και στην *RF* κάνοντας το αντίστοιχο *Q* έγκυρο (1111).

c) Όταν ένας καταχωρητής ο οποίος έχει ήδη ένα *Tag* από κάποιο *Reservation station* εμφανιστεί στην είσοδο της *RF* ξανά ως καταχωρητής προορισμού, όπου "πανωγράφεται" το *Q* του και πλέον οι εντολές που εκδίδονται και ζητούν τον συγκεκριμένο καταχωρητή ως *RS* ή ως *RT* θα παίρνουν ως *Tag* του καταχωρητή το νέο *Tag* και θα περιμένουν αυτό από το *CDB* για να δεχτούν τα δεδομένα του καταχωρητή.

Behavioral Simulation - RF



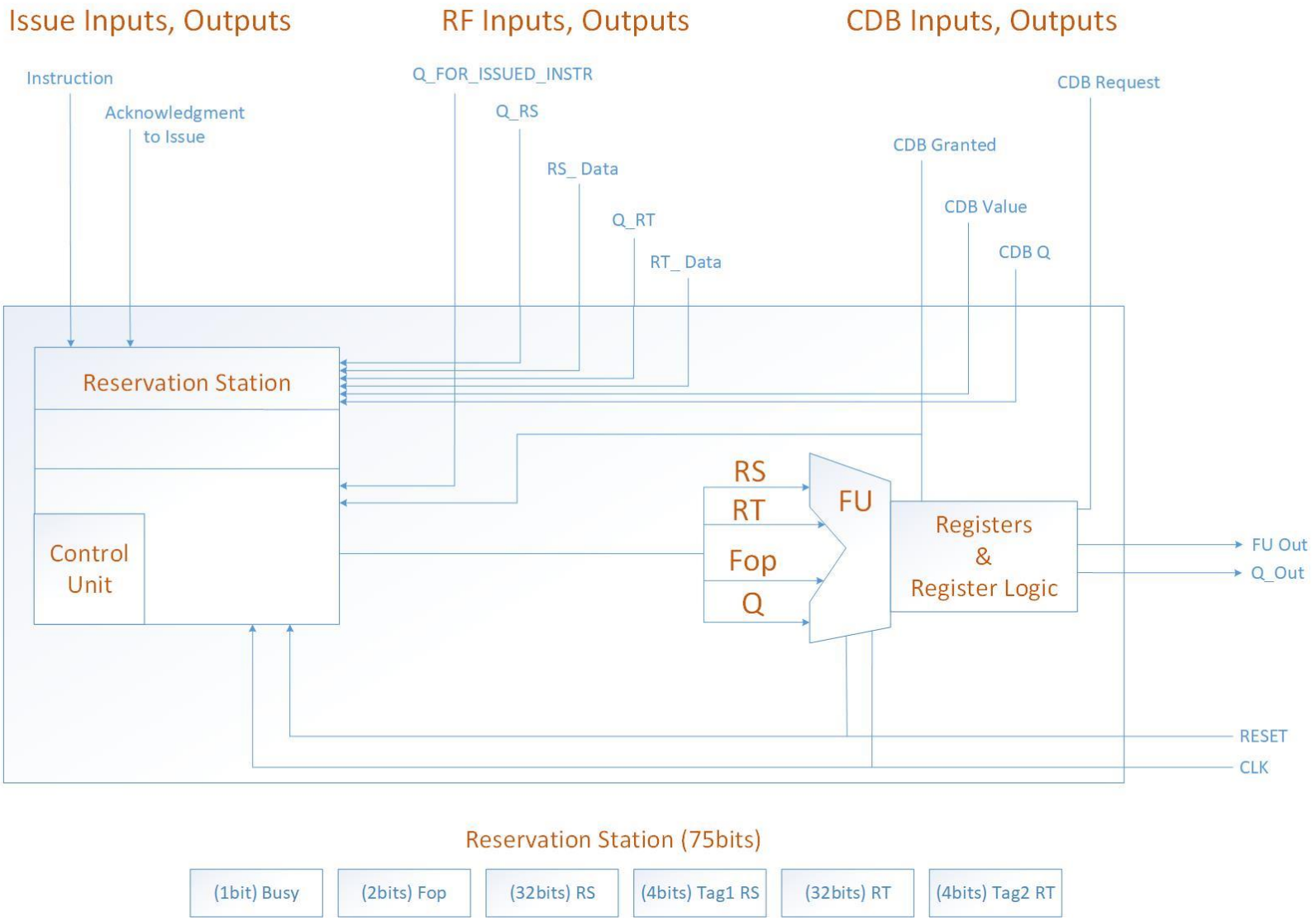
RS + FU (Reservation Station + Functional Unit)

Αυτά τα δύο components είναι υπεύθυνα για την συνεκτική εκτέλεση των εντολών με αποδοτικό τρόπο. Οι μονάδες των *Reservation Station*s αποτελούνται από τις τιμές των δύο καταχωρητών *(Register Data RS & RT)* και τα αντίστοιχα *Tags* τους *(QR & QT)*, την πράξη που πρέπει να εκτελεστεί με αυτές τις τιμές *(Fop)* και το αν το *Reservation Station* είναι διαθέσιμο ή όχι *(Busy).*

Αν υπάρχει διαθέσιμο *Reservation Station* και έρθει αντίστοιχη εντολή, το *Reservation Station* τη δέχεται, και δεσμεύεται (πλέον χαρακτηρίζεται ως *Busy*). Στον επόμενο κύκλο, το *Reservation Station* που δεσμεύτηκε αποθηκεύει και τα δεδομένα που βγαίνουν από τη *Register File.* Η έκδοση των εντολών στο σύστημα γίνεται *pipelined*, σε δύο κύκλους, ούτως ώστε να εκμεταλλευόμαστε στο έπακρον τα δομικά στοιχεία του επεξεργαστή μας και να μπορούμε να εκδίδουμε καινούρια εντολή σε κάθε κύκλο ρολογιού. Αν δεν γινόταν αυτό θα έπρεπε να περιμένουμε έναν κύκλο για την έκδοση της εντολής κι άλλον ένα για να πάρουμε τα δεδομένα από την *RF*, σύνολο δύο κύκλους (ενώ με τον παραπάνω τρόπο μας κοστίζει μόνο έναν κύκλο), πράγμα που σημαίνει πως η απόδοση του συστήματος θα μειωνόταν αρκετά.

Όποτε το *Common Data Bus* ανακοινώνει μια τιμή, το Tag της συγκρίνεται με τα αντίστοιχα Tags των RS και RT καταχωρητών των διάφορων *Reservation Stations* και οι τιμές τους ανανεώνονται αν τα Tags συμπίπτουν.

Τα *Functional Units* εκτελούν την εντολή όταν αυτή είναι έτοιμη, δηλαδή όταν τα δεδομένα και των δύο καταχωρητών της πράξης (*RS* και *RT*) είναι πλέον έγκυρα. Όταν οι τιμές ενός *Reservation Station* είναι έγκυρες (δηλαδή το Tag και των δύο είναι Q1 = Q2 = 1111), η μονάδα ελέγχου ελέγχει αν το *pipeline* του *Functional Unit* είναι γεμάτο και αν δεν είναι, στέλνει την εντολή προς εκτέλεση και αποδεσμεύει το *Reservation Station.*



Σχηματικό διάγραμμα των Reservation Station και του Functional Unit με τις εισόδους και τις εξόδους τους.

Interface

Το interface του component αυτού αποτελείται από 3 διαφορετικά μέρη: τη διεπαφή μεταξύ του *Instruction Issue* και των *Reservation Stations*, τη διεπαφή μεταξύ των *Reservation Stations* και της *Register File* και τη διεπαφή μεταξύ του *Functional Unit* και του *Common Data Bus*.

- **Διεπαφή μεταξύ του *Instruction Issue* και των *Reservation Stations***

Το *Issue* ανακοινώνει σε όλα τα *RS+FU* components μία εντολή προς έκδοση. Η μονάδα ελέγχου των *Reservation Station* ελέγχει αν ο τύπος της εντολής είναι αυτός που δέχεται το συγκεκριμένο *RS+FU* component και αν ναι, αν υπάρχει διαθέσιμο *Reservation Station* για αυτή την εντολή. Αν όλα τα παραπάνω ισχύουν, τότε δεσμεύει το συγκεκριμένο *RS*, αποθηκεύει από την εντολή ό,τι πληροφορίες χρειάζεται. Ταυτόχρονα στέλνει σήμα επιβεβαίωσης στο *Issue* στον ίδιο κύκλο.

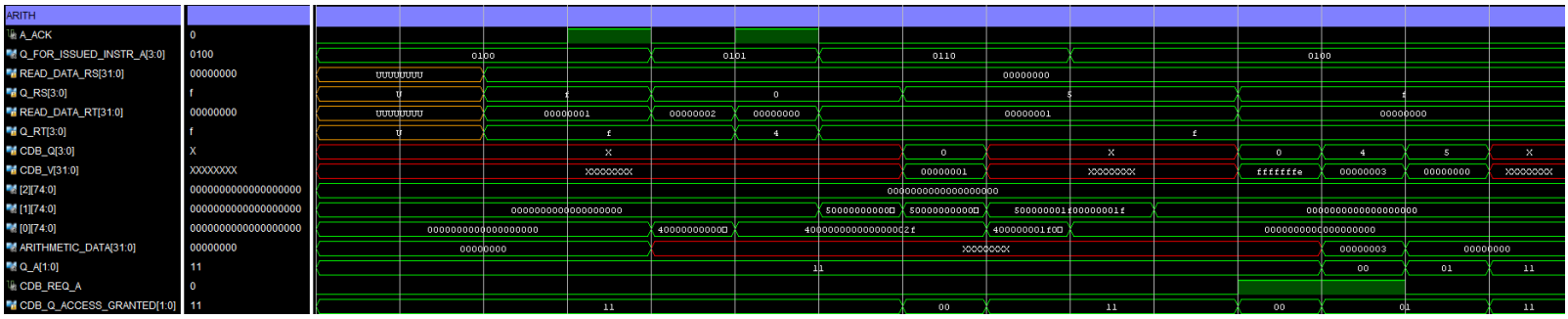
- **Διεπαφή μεταξύ των *Reservation Stations* και της *Register File***

Κάθε φορά που μια εντολή εκδίδεται (δηλ. γίνεται αποδεκτή από ένα *Reservation Station*), η μονάδα ελέγχου στέλνει στη *Register File* το *Tag* του *Reservation Station* που δεσμεύτηκε μόλις για να το γράψει η *RF* στο αντίστοιχο πεδίο *Q* του καταχωρητή *RD.* Το *Reservation Station* λαμβάνει τις τιμές των αντίστοιχων καταχωρητών (*RS* και R*T*) και τα Tags τους από την *Register File* ένα κύκλο μετά (*pipelined* – βλέπε παραπάνω, περιγραφή RS+FU).

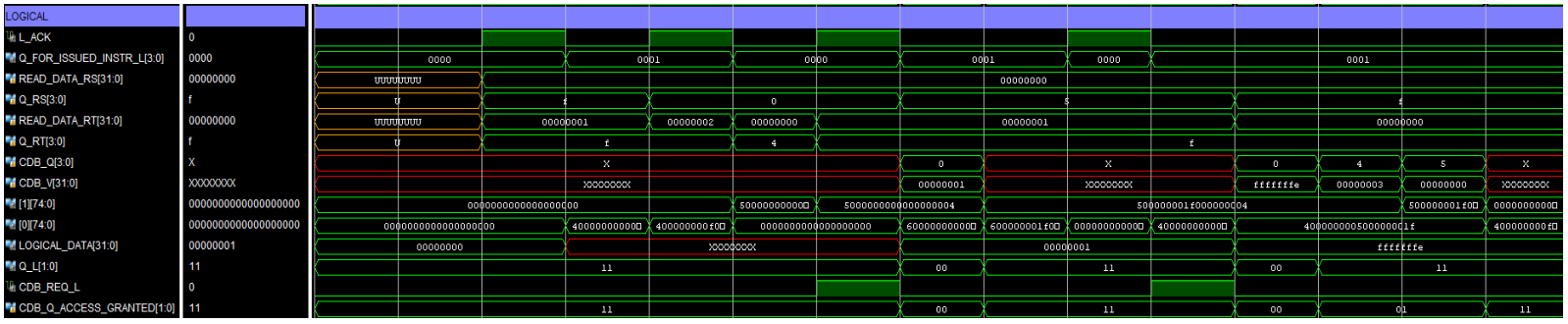
- **<u>Διεπαφή μεταξύ του *Functional Unit* και του *Common Data Bus*.</u>**

Το *Functional Unit* είναι ένα *pipelined* component υπεύθυνο για την εκτέλεση της εντολής όπως την λαμβάνει από ένα *Reservation Station.* Η μονάδα αυτή ζητάει πρόσβαση στο *Common Data Bus* ένα κύκλο πριν το τέλος κάθε υπολογισμού. Μόλις το *Common Data Bus* δώσει πρόσβαση στο *Functional Unit,* το αποτέλεσμά του και το αντίστοιχο Tag του ανακοινώνονται στον επόμενο κύκλο από το *Common Data Bus*.

<u>Behavioral Simulation - Arithmetic RS + FU</u>

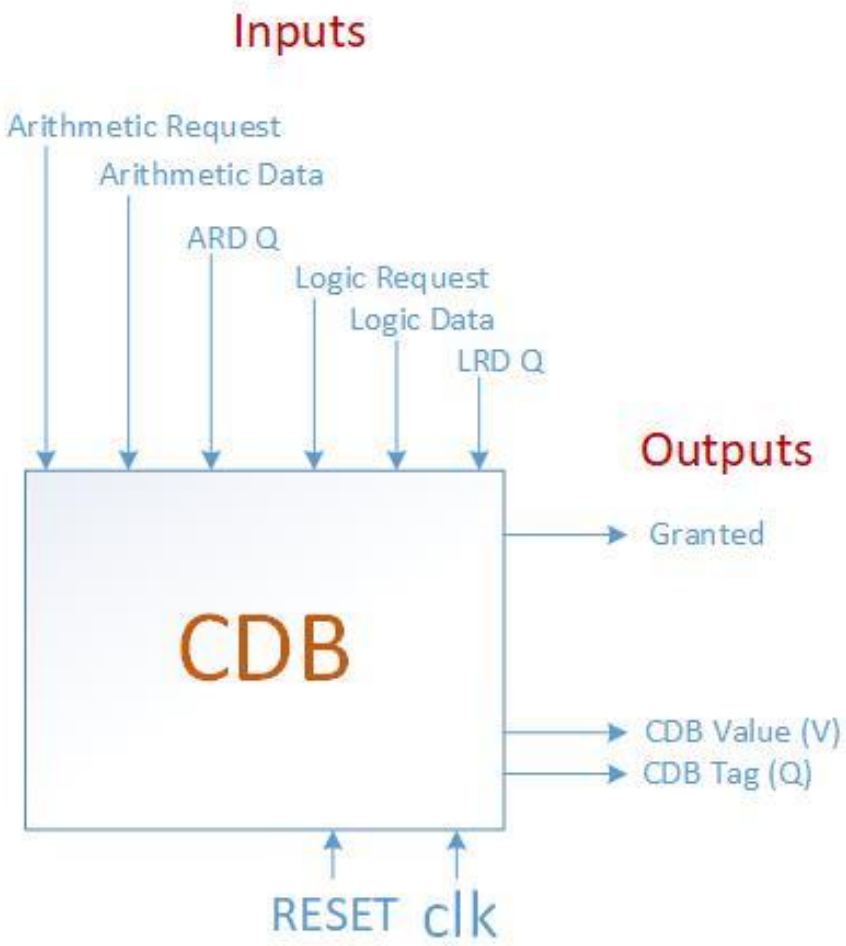


<u>Behavioral Simulation - Logical RS + FU</u>



CDB (Common Data Bus)

Το *Common Data Bus* είναι ο κοινός δίαυλος επικοινωνίας του συστήματος, μέσω του οποίου τα *Reservation Stations* ανακοινώνουν και ανανεώνουν τις τιμές τους και γράφεται η *Register File.*



Σχηματικό διάγραμμα του Common Data Bus με τις εισόδους και τις εξόδους του.

Interface

Είσοδοι

Το *CDB* έχει μια είσοδο "Αίτησης" του για κάθε *Functional Unit* που υπάρχει στο σύστημα (*Logic* και *Arithmetic Request* αντίστοιχα) καθώς και μια είσοδο για να παίρνει τα δεδομένα και τα *Tags* των τους (*Logic* και *Arithmetic Data & Q* αντίστοιχα).

Έξοδοι

Μέσω των παραπάνω δύο εισόδων (*Logic* και *Arithmetic Data & Q*) τα διάφορα *Functional Units* όταν έχουν διαθέσιμο κάποιο αποτέλεσμα μπορούν αν το κοινοποιήσουν στα υπόλοιπα μέσω του *CDB* (*CDB_Q* και *CDB_V*) αφού πρώτα κάνουν αίτηση στο *CDB* για κοινοποίηση των δεδομένων τους και το τελευταίο τους "απαντήσει" θετικά (*CDB_Granted*). Η έξοδος *CDB_Granted* πηγαίνει σε όλες τις λειτουργικές μονάδες ούτως ώστε να γνωρίζουν όλες ποιος θα κοινοποιήσει στο *CDB* στον επόμενο κύκλο ("00" για *Logical FU* και "01" για *Arithmetic FU*). Στον επόμενο κύκλο λοιπόν το *CDB* κοινοποιεί το *Value* και το *Q* της λειτουργικής μονάδας που κέρδισε το παιχνίδι διαιτησίας του προηγούμενου κύκλου, αν φυσικά υπήρξε παιχνίδι διαιτησίας. Αν μόνο μία μονάδα ζήτησε να κοινοποιήσει στον επόμενο κύκλο προφανώς και θα της δοθεί πρόσβαση στο CDB χωρίς κάποια πολύπλοκη διαδικασία.

Τέλος οι έξοδοι του *CDB, CDB_V* και *CDB_Q* οδηγούνται τόσο στην *Register File* όσο και σε στα διάφορα *Reservation Stations*. Στα *Reservation Stations* για να δώσουν την τιμή τους στους καταχωρητές που τυχόν περιμένουν το συγκεκριμένο *Reservation Stations* και στην *Register File* για να γράψουν την τιμή τους στον καταχωρητή προορισμού με τον τρόπο που αναφέρθηκε στην *Register File (*βλέπε παραπάνω – περιγραφή της *RF).*

Timing

Σε κάθε κύκλο ρολογιού το *Common Data Bus* δέχεται τις αιτήσεις των *Functional Units* τα οποία έχουν έτοιμα δεδομένα προς κοινοποίηση και αποφασίζει ποιος απ' όλους θα κοινοποιήσει στον επόμενο κύκλο. Όταν έρθει ο επόμενος κύκλος, λοιπόν, το *CDB* κοινοποιεί τον αριθμό της λειτουργικής μονάδας την οποία επέλεξε στον προηγούμενο κύκλο για να κοινοποιήσει σε αυτόν μέσω του *GRANTED* σήματος, κοινοποιώντας ταυτόχρονα τα δεδομένα και το *Tag (Tag και Value)* που του μεταβίβασε το συγκεκριμένο *Functional unit*.

Πιο συγκεκριμένα στοιχεία για τη λειτουργία του CDB

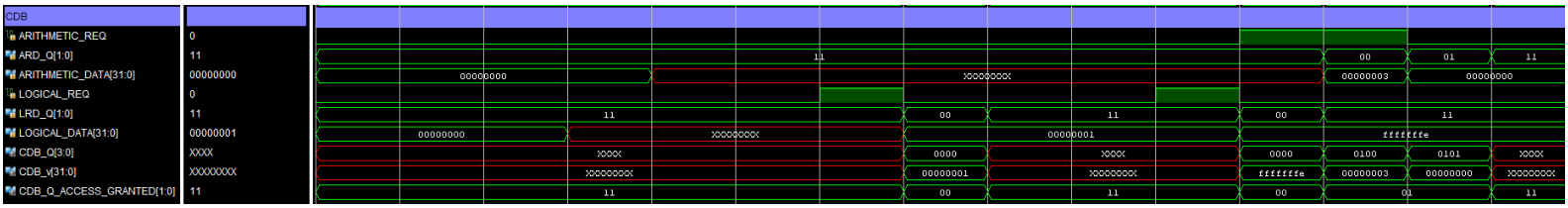
Αξίζει να σημειωθεί σε αυτό το σημείο πως το *CDB* δεν λειτουργεί με βάση τον αλγόριθμο *Round Robin* όπως μας ζητήθηκε από την εκφώνηση της άσκησης αλλά με μια παραλλαγή αυτού ούτως ώστε να λειτουργεί πιο δίκαια και συνεπώς πιο αποδοτικά. Στην υλοποίησή μας, λοιπόν, κάθε φορά που κάποιος κοινοποιεί μέσω του *CDB*, το τελευταίο κρατάει σε μια μεταβλητή την αριθμητική του τιμή, ούτως ώστε αν στον επόμενο κύκλο ξαναζητήσει το ίδιο *Functional unit* να κοινοποιήσει δεδομένα μέσω του *CDB* και έρθει σε "ρήξη" με ένα άλλο *Functional unit* που κι αυτό ζητά να κοινοποιήσει στον ίδιο κύκλο να δώσει προτεραιότητα στο άλλο *FU*, το οποίο είναι δεδομένο πως δεν κοινοποίησε στον προηγούμενο κύκλο και γι' αυτόν το λόγο δικαιούται να κοινοποιήσει σε αυτόν. Αν οι λειτουργικές μονάδες ζητούν μόνιμα το *CDB* και οι δύο, τότε αυτό δίνει πρόσβαση μία στη μία και μία στην άλλη, και συνεπώς υλοποιείται ακριβώς ο αλγόριθμος *Round Robin*. Η παραπάνω κατάσταση αποτελεί corner case αλλά και πάλι το σύστημα ανταποκρίνεται αποδοτικά. Υπό φυσιολογικές συνθήκες το σύστημα λειτουργεί πιο έξυπνα και αποδοτικά, όπως αναφέρθηκε παραπάνω.

Η έξοδος “XXXXXX” του *Common Data Bus* υποδηλώνει πως εκείνη τη στιγμή το *Bus* έχει “σκουπίδια” μέσα. Δεν πρόκειται περί λάθους, αντίθετα αποτελεί την προγραμματιστική λογική που επιλέξαμε για να είναι πιο ευδιάκριτες οι κυμματομορφές. Το *CDB* παίρνει τιμές πέραν του “XXXXXX” μόνο όταν κάποιο *Functional unit* κοινοποιεί μέσω αυτού.

Σε πλήρη αντιστοιχία με το παραπάνω η default κατάσταση του *CDB_Tag* είναι η “1111” και αυτό δεν είναι καθόλου τυχαίο. Στην υλοποίησή μας δεν υπάρχει *Reservation station* με *Tag* “1111”, έτσι όταν το *CDB* Κοινοποιεί σκουπίδια βγάζει στην έξοδό του ως Tag “1111” και ως *Value* “XXXXXX”.

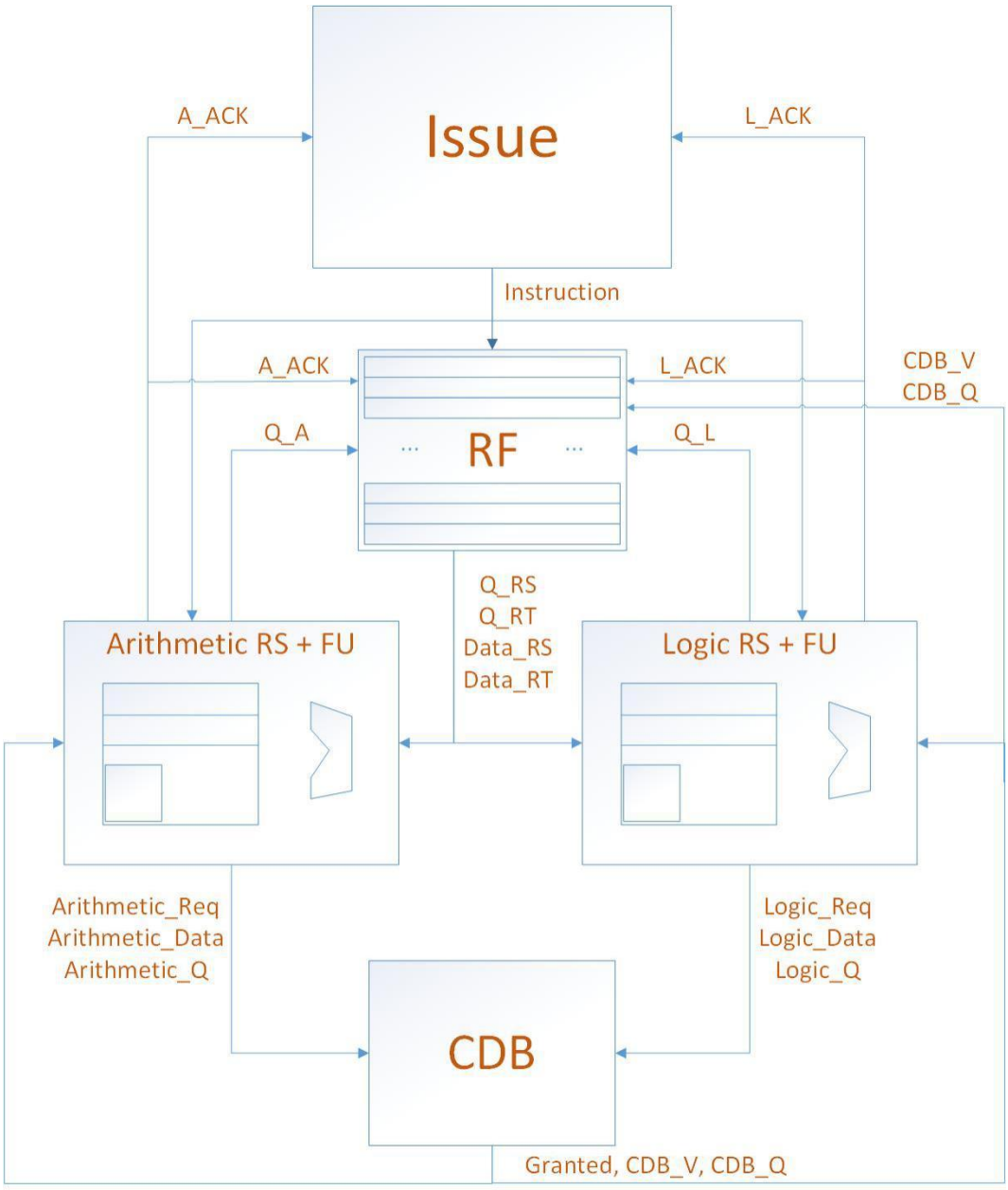
Με τον τρόπο αυτό και αφού πριν γραφτεί οτιδήποτε (, οπουδήποτε) ελέγχεται πρώτα το Tag που κοινοποιείται, δεν υπάρχει περίπτωση σύγχυσης και όλοι γνωρίζουν πότε τα δεδομένα είναι έγκυρα και πότε όχι.

Behavioral Simulation - CDB



Σύνδεση των επιμέρους Components

Στη δεύτερη φάση του project (δεύτερο milestone) πραγματοποιήθηκε η σύνδεση των τεσσάρων παραπάνω βασικών components και ο συγχρονισμός τους ώστε να λειτουργούν συντονισμένα ως ένα σύστημα υλοποιώντας τον αλγόριθμο Tomasulo.



Σχηματικό διάγραμμα του αλγορίθμου του Tomasulo που υλοποιήθηκε.

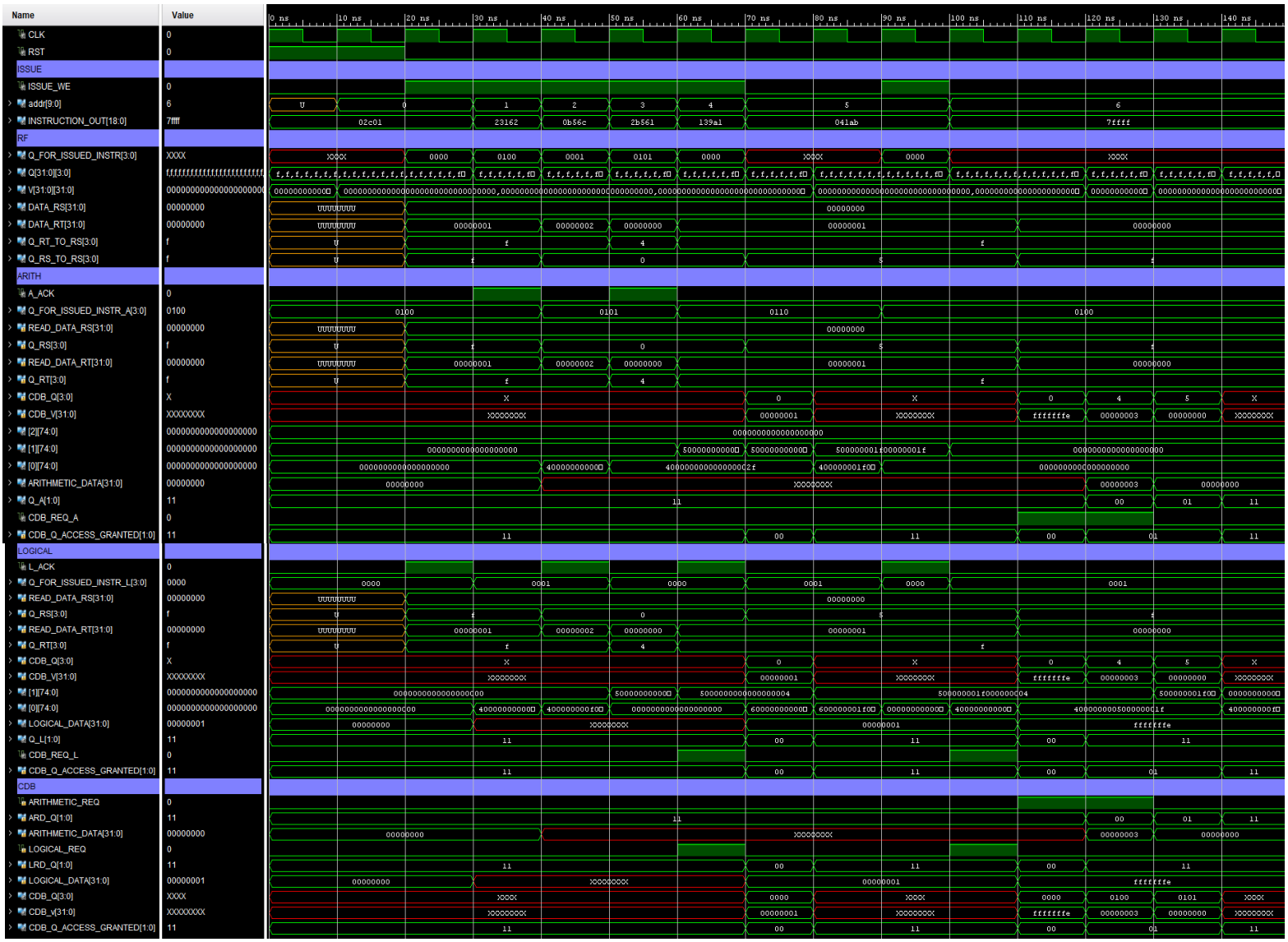
Επιβεβαίωση Ορθής Λειτουργίας - Προσομοίωση

Η ορθή λειτουργία της υλοποίησης του αλγορίθμου φαίνεται στα αποτελέσματα της παρακάτω προσομοίωσης. Ακολουθεί η αλληλουχία εντολών που δοκιμάστηκε, όπου περιγράφονται και όλες οι ιδιαίτερες περιπτώσεις και δοκιμάζεται επιτυχημένα η υλοποίησή μας.

	Type	FOP	RD	RS	RT
INSTR 1	00	00	01011	00000	00001
	LOGICAL	OR	11	0	1
INSTR 2	01	00	01100	01011	00010
	ARITHM	ADD	12	11	2
INSTR 3	00	01	01101	01011	01100
	LOGICAL	AND	13	11	12
INSTR 4	01	01	01101	01011	00001
	ARITHM	SUB	13	11	1
INSTR 5	00	10	01110	01101	01011
	LOGICAL	NOT	14	13	11
INSTR 6	00	00	10000	01101	01011
	LOGICAL	OR	16	13	11

- Εντολή 1-3: CDB broadcasts – εξαρτήσεις (*Reservation Stations* tags)
- Εντολή 4: Πανωγράφεται το Q της RF από νέα εντολή (R*+[13]) και CDB broadcast
- Εντολές 5-6: Εξάρτηση R[13] από την εντολή No 4, έλεγχος write-first (CDB broadcast και Instruction Issue για τον ίδιο καταχωρητή)
- Εντολή 6: Αργεί να γίνει αποδεκτή αφού δεν υπάρχει διαθέσιμο *Reservation Station.*

Behavioral Simulation – Tomasulo



Αξίζει να σημειωθεί σε αυτό το σημείο ότι οι εντολές εκδίδονται στο σύστημα συνεχόμενα, μία σε κάθε κύκλο και το τελευταίο λειτουργεί απροβλημάτιστα, χωρίς την ανάγκη ύπαρξης καθυστερήσεων μέχρι να εκδοθεί η πρώτη εντολή για να εκδοθεί μετά η δεύτερη, η τρίτη κ.ο.κ.

Τέλος με το Reset αρχικοποιούμε τους καταχωρητές $R_0 - R_{10}$ με τις τιμές 1-10 αντίστοιχα για να φαίνονται τα αποτελέσματα των πράξεων που γίνονται στα διάφορα *Functional Units* των *Reservation Stations* και πως το σύστημα λειτουργεί σωστά.

Reorder Buffer - *ROB (FIFO)*

Ουσιαστικά το *Reorder Buffer (ROB)* είναι μια (δομή δεδομένων) κυκλική *ουρά (FIFO – First In First Out)*. Κάθε φορά που εκδίδεται μια καινούρια εντολή στο σύστημα εγγράφεται πρώτα στο *ROB* και παίρνει ένα αναγνωριστικό – κωδικό (π.χ. *ROB12*). Στη συνέχεια το σύστημα λειτουργεί όπως ακριβώς λειτουργούσε και πριν, με τη μόνη διαφορά πως πλέον οι εξαρτήσεις μεταξύ των εντολών δεν υφίστανται με βάση τα *Reservation Stations* αλλά μεταξύ των επιμέρους εγγραφών στο *ROB*. Έτσι , λοιπόν, όταν μια εντολή ολοκληρωθεί (ας πούμε η *ROB7*), κοινοποιεί μέσω του *CDB* το αποτέλεσμά της με *Tag* το αναγνωριστικό που είχε πάρει από το *ROB (ROB7)*. Το αποτέλεσμα αυτό πάει σε όλα τα *Reservation Stations* αλλά και πίσω στο *ROB*. Όλες οι εντολές που περίμεναν το αποτέλεσμα της *ROB7* μπορούν να το χρησιμοποιήσουν πλέον και το ίδιο αποτέλεσμα γράφεται και στην αντίστοιχη εγγραφή στο *ROB*.

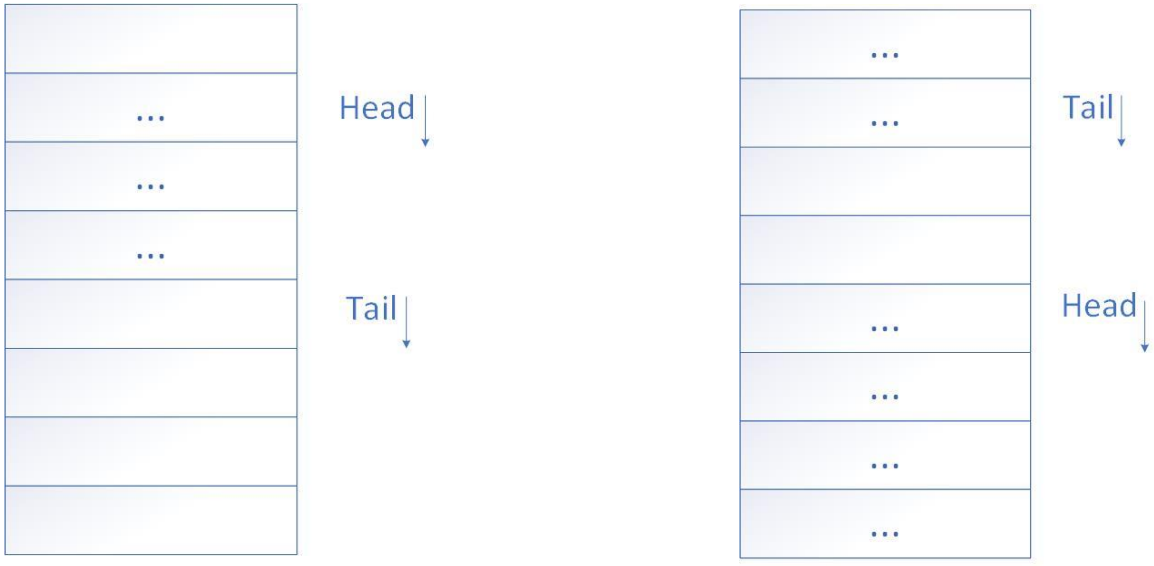
Η μόνη εντολή που επιτρέπεται να γίνει *Commit* στο σύστημα είναι η παλαιότερη που υπάρχει στο *ROB*. Όταν, λοιπόν, η παλαιότερη αυτή εντολή ολοκληρωθεί, το ROB προχωρά στην επενέργεια – γράψιμο *(Commit)* της εντολής αυτής στο σύστημα και τη σβήνει από την ουρά. Στους επόμενους κύκλους γίνονται με τη σειρά *Commit* στο σύστημα όλες οι επόμενες (της παλαιότερης) εντολές που μπορεί να είχαν ολοκληρωθεί πριν ολοκληρωθεί η εν λόγω εντολή αλλά δεν είχαν γίνει *Commit* στο σύστημα αφού δεν είχε ολοκληρωθεί η πρώτη (καθώς η σύμβαση είναι πως μόνο η πρώτη εντολή μπορεί αν γίνει *Commit*).

Όλη η παραπάνω διαδικασία γίνεται για να μπορεί το σύστημα να διατηρεί την ακεραιότητά του ακόμη και στην περίπτωση που δημιουργηθεί κάποιο *exception* στις εντολές που "τρέχουν" στο σύστημα (και εκτελούνται εκτός σειράς). Στην περίπτωση λοιπόν που δημιουργηθεί κάποιο *exception*, το οποίο αναφέρεται σε μία συγκεκριμένη εντολή, άρα και σε ένα συγκεκριμένο *PC* και σε ένα συγκεκριμένο *ROB*, "σβήνουμε" τόσο από το *ROB*, όσο και από όλα τα επιμέρους *Reservation Stations* την εντολή αυτή καθώς και όλες

εκείνες τις εντολές που εκδόθηκαν στο σύστημα μετά από αυτή. Ταυτόχρονα σταματάμε να δεχόμαστε νέες εντολές. Μένουν, με τον τρόπο αυτό, στο σύστημα μόνο οι εντολές που εκδόθηκαν πριν από την εντολή στην οποία αναφερόταν το *exception*. Οι εντολές αυτές συνεχίζουν να εκτελούνται κανονικά εκτός απροόπτου (δεύτερο *exception* που αναφέρεται σε μία από αυτές) και η λειτουργία του συστήματος σταματά όταν γίνει *Commit* από το *ROB* και η τελευταία από αυτές.

Πως λειτουργεί εσωτερικά το Reorder Buffer

Όπως αναφέρθηκε και παραπάνω το *ROB* είναι μια κυκλική ουρά δεδομένων που κρατάει τις εντολές που έχουν εκδοθεί στο σύστημα. Παρακάτω φαίνεται μια σχηματική απεικόνιση αυτού, ο τρόπος με τον οποίο γεμίζει και συζητούνται δύο οριακές περιπτώσεις:



Σχηματικό διάγραμμα του Reorder Buffer.

Το *Head* και το *Tail* αποτελούν την αρχή και το τέλος της *ουράς*. Και τα δύο αυξάνονται προς τη ίδια κατεύθυνση (προς τα κάτω όπως φαίνεται στο σχήμα). Το *Tail* δείχνει πάντα την θέση που θα εισαχθεί η επόμενη εντολή που θα εκδοθεί στο σύστημα. Έτσι, λοιπόν, όταν μια καινούρια εντολή εισάγεται στο σύστημα μπαίνει στην θέση που δείχνει το *Tail*, και αυτό αυξάνεται κατά 1. Αντίστοιχα όταν μια εντολή γίνεται *Commit* στο σύστημα (όπως αναφέρθηκε παραπάνω μόνο η πιο παλιά εντολή μπορεί να γίνει Commit) βγαίνει από τη θέση που δείχνει το *Head*, κι αυτό αυξάνεται κατά 1.

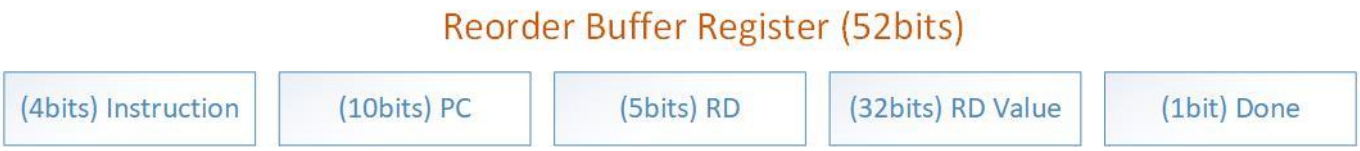
Επειδή η ουρά μας είναι κυκλική μια ιδιαίτερη περίπτωση είναι τι γίνεται όταν το *Tail* φτάσει στο φυσικό τέλος της ουράς και πρέπει να συνεχίσει να αυξάνεται καθώς εκδίδονται νέες εντολές στο σύστημα. Ο τρόπος με τον οποίο αναγνωρίζουμε αν βρισκόμαστε στην παραπάνω παθολογική κατάσταση είναι ελέγχοντας αν το *Tail* είναι μικρότερο του *Head.* Εάν λοιπόν βρισκόμαστε σε αυτή την περίπτωση, την αντιμετωπίζουμε διαχειριζόμενοι πρώτα το κομμάτι από το την φυσική αρχή της ουράς μέχρι το *Tail* (πάνω κομμάτι στο σχήμα) και έπειτα το κομμάτι από το *Head* μέχρι το φυσικό τέλος της ουράς (κάτω κομμάτι στο σχήμα).

Όπως αναφέρθηκε παραπάνω το *Tail* δείχνει πάντα την θέση στην οποία θα εισαχθεί η επόμενη εντολή που θα εκδοθεί στο σύστημα. Αν σε κάποια χρονική στιγμή φτάσει το *Tail* να δείχνει στην ίδια θέση με το *Head,* αυτό σημαίνει πως το ROB έχει γεμίσει και πως δεν μπορούμε να δεχτούμε άλλες εντολές. Η δομή ελέγχου υπάρχει (στον κώδικα) αλλά φροντίσαμε να κάνουμε το ROB αρκετά μεγάλο ούτως ώστε να γλιτώσουμε αυτή την παθολογική κατάσταση χωρίς πρακτικά να την αντιμετωπίζουμε.

Αλλαγές που έγιναν στο αρχικό σύστημα για να λειτουργεί με βάση το Reorder Buffer

Οι αλλαγές που έγιναν στην σχεδίαση του δεύτερου *Milestone* για να υποστηριχτεί η παρουσία και η απροβλημάτιστη λειτουργία του *ROB* στο σύστημα είναι οι εξής:

- Το *Issue* πλέον βγάζει στην έξοδό του και την αριθμητική τιμή του εσωτερικού του *Counter*, δηλαδή του *PC counter.* Η τιμή αυτή αποθηκεύεται μαζί με τον τύπο της εντολής και το νούμερο του καταχωρητή προορισμού *(RD)* στο *ROB*, αφήνοντας ταυτόχρονα χώρο και για την τιμή του *(RD)* όταν η εντολή ολοκληρωθεί.



- Πλέον το σήμα *Q_For_Issued_Instruction* που στο δεύτερο *Milestone* ερχόταν από τα *Reservation Stations,* δημιουργείται εσωτερικά στο ROB, το οποίο το κοινοποιεί στο *Reservation Station* που δέχεται την εντολή που εκδίδεται. (Η λειτουργία του σήματος παραμένει η ίδια)

- Η *Register File* πλέον αποτελεί την *Architectural Register File (ARF)*, δηλαδή οι τιμές που περιέχονται σε αυτήν είναι πάντα έγκυρες, χωρίς αυτό να σημαίνει όμως ότι είναι και οι τελευταίες που υπολογίστηκαν, αφού μια πιο πρόσφατη τιμή για τον κάθε καταχωρητή μπορεί να βρίσκεται στο *ROB*. Όταν εκδίδεται μια νέα εντολή και ζητά τους καταχωρητές *RS* και *RT*. Τιμές και για τους δύο καταχωρητές δίνονται από την *ARF* και από το *ROB* σε περίπτωση που υπάρχουν έγκυρα δεδομένα ή κάποια θέση (tag) καταχωρημένα για αυτόν τον καταχωρητή. Το *ROB* δίνει τις πιο πρόσφατες καταχωρήσεις για τους δύο καταχωρητές, και είναι δουλειά των *Reservation Stations* να καταλάβουν από τα αντίστοιχα σήματα και tags αν πρέπει να διαλέξει

την τιμή του *ROB*, ή αν δεν υπάρχει τέτοια ώστε να διαλέξει την τιμή της *ARF.* Το *ROB* είναι σχεδιασμένο Write-first, όπως και η *ARF* συνεπώς αν η εντολή που εκδίδεται ζητάει κάτι που γράφεται αυτή τη στιγμή το *ROB*, δεν θα υπάρξει κανένα πρόβλημα.

1. Προστέθηκε λογική στα *Reservation Stations* για να σβήνουν της εντολές που εκδόθηκαν μετά από την εντολή στην οποία δημιουργήθηκε το *exception*. Τα *exception* σήματα που σηματοδοτούν την έναρξη της παραπάνω διαδικασίας πηγαίνουν αρχικά στο *ROB*, το οποίο σβήνει από το εσωτερικό του όλες τις εντολές που εκδόθηκαν μετά από αυτή, στην οποία δημιουργήθηκε το *exception* (βλέπε παρακάτω, το Interface για τα *exception* σήματα). Αυτό όμως δεν φτάνει, οι ίδιες εντολές πρέπει να σβηστούν και από τα αντίστοιχα *Reservation Stations.* Αυτό γίνεται όταν το *ROB* κοινοποιήσει το αναγνωριστικό μίας παθολογικής εντολής. Όταν συμβεί αυτό, τα *Reservation Stations* "κοιτούν" μία-μία της εντολές που έχουν καταχωρημένες στο εσωτερικό τους και σβήνουν όποια έχει εκδοθεί μετά από την εν λόγω εντολή. Το να σβήσουμε όλες τις εντολές με αναγνωριστικό μεγαλύτερο από αυτό της παθολογικής εντολής θα ήταν λάθος καθώς στην περίπτωση που το *Tail* είναι μικρότερο του *Head* (λόγω κυκλικής ουράς) θα υπήρχε σοβαρό πρόβλημα. Λύση στο πρόβλημα αυτό έρχεται να δώσει η λογική: σβήνουμε όλες τις εντολές με αναγνωριστικό μεταξύ του αναγνωριστικού της παθολογικής εντολής και του *Tail*, με τον τρόπο που αναφέρθηκε παραπάνω, αφού η ίδια λογική ακολουθείται και στο σβήσιμο των εντολών από το *ROB*. Να θυμίσουμε ότι *Tail είναι πάντα το σήμα Q_For_Issued_Instruction – 1*.

- Το *CDB* συνδέεται πλέον με το *ROB* και όχι κατευθείαν με την *ARF.* Στην τελευταία γράφονται πλέον μόνο τα αποτελέσματα των εντολών που γίνονται *Commit* από το *ROB.*

Τρείς επισημάνσεις για τη λειτουργία του ROB:

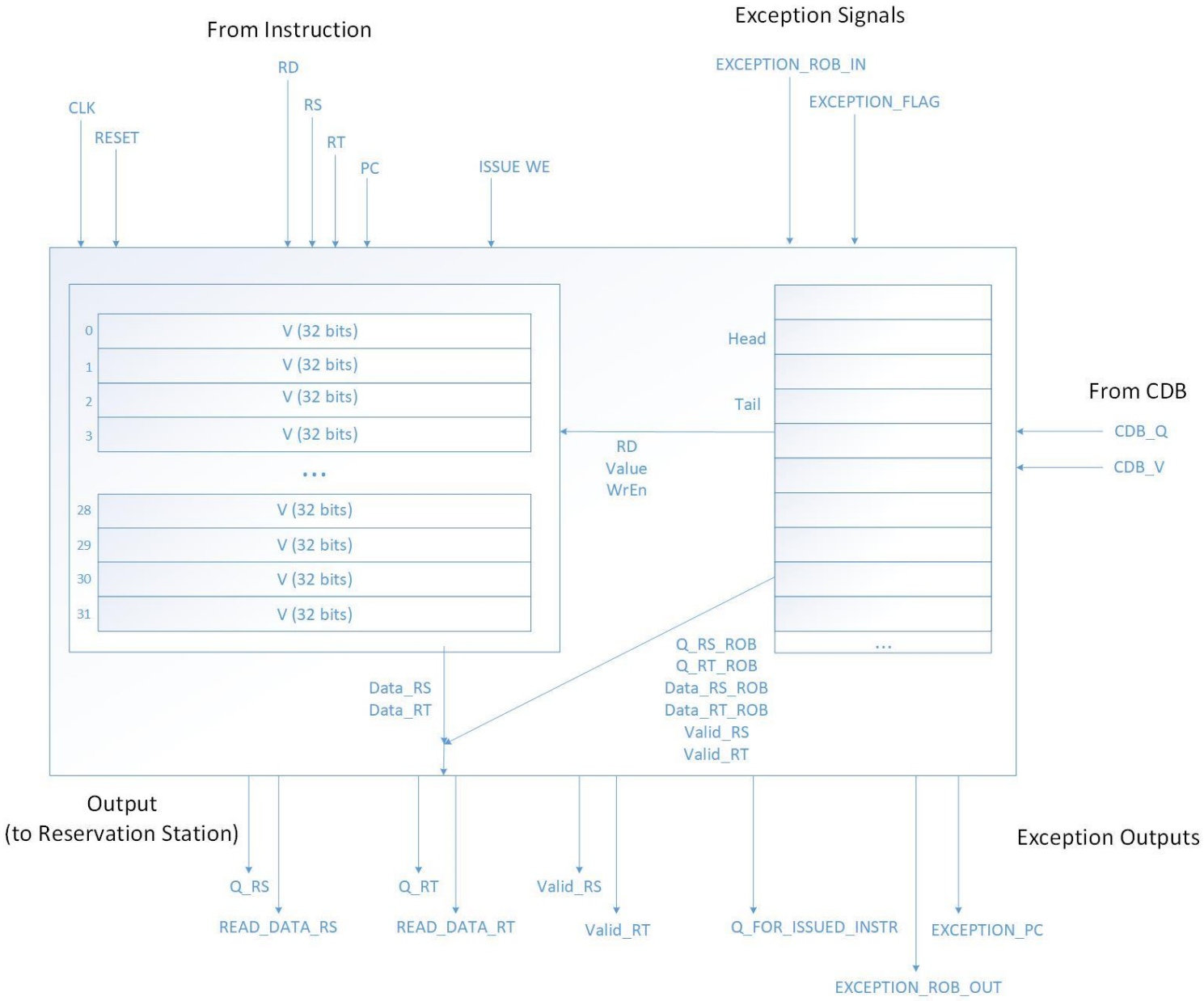
2. Ως σύμβαση (για τον έλεγχο της σχεδίασης) θεωρήσαμε πως όταν δημιουργείται ένα *exception* δίνεται στο *ROB* το αναγνωριστικό της εντολής στην οποία δημιουργήθηκε το *exception.* Με αυτό ως δεδομένο βγάζουμε στην έξοδο του συστήματος το *PC* της εν λόγω εντολής. Θα μπορούσαμε να έχουμε δουλέψει ανάποδα, δηλαδή να μας δίνεται το *PC* και να βγάζουμε το αναγνωριστικό στην έξοδο, αλλά θεωρήθηκε πιο σωστή αυτή η τακτική καθώς αν ένα *exception* δημιουργηθεί στο εσωτερικό του επεξεργαστή το μόνο που γνωρίζει η ίδια η εντολή είναι το αναγνωριστικό της και όχι το αντίστοιχο *PC* της, και τα εσωτερικά exception είναι που μας ενδιαφέρουν πιο πολύ.

3. Το αναγνωριστικό των εντολών, που δεν είναι άλλο από το σήμα *Q_For_Issued_Instruction* που αναφέρθηκε παραπάνω αποτελεί τη θέση που καταλαμβάνει η εντολή στο *ROB.* Πιο συγκεκριμένα, η πρώτη εντολή θα πάρει το *ROB1*, η δεύτερη το *ROB2* κ.ο.κ.. Όταν εκδοθούν αρκετές εντολές για να φτάσουμε στο φυσικό τέλος της ουράς, αφού η ουρά μας είναι κυκλική ξεκινάμε ξανά από την αρχή και δίνουμε στην επόμενη εντολή το αναγνωριστικό *ROB1* κτλ.. Παθολογικά φαινόμενα σύγχυσης του αναγνωριστικού μιας εντολής ή του τρόπου διαχείρισής τους αντιμετωπίζονται με τον τρόπο που περιεγράφηκε παραπάνω.

4. Το μέγεθος του *ROB*, δόθηκε αυθαίρετα και είναι 15, θεωρώντας πως με συνολικά 5 *Reservation Stations* είναι σχεδόν αδύνατον να εκδοθούν και να ολοκληρωθούν παραπάνω από 15 εντολές στο σύστημα χωρίς να έχει ολοκληρωθεί η πρώτη. Με αυτόν τον τρόπο το *ROB* δεν "γεμίζει" ποτέ και όσο και να αργήσει η πρώτη εντολή να ολοκληρωθεί πάντα θα υπάρχει χώρος για να εκδοθούν νέες εντολές όσο ολοκληρώνονται άλλες κα αδειάζουν τα αντίστοιχα *Reservation Stations.*

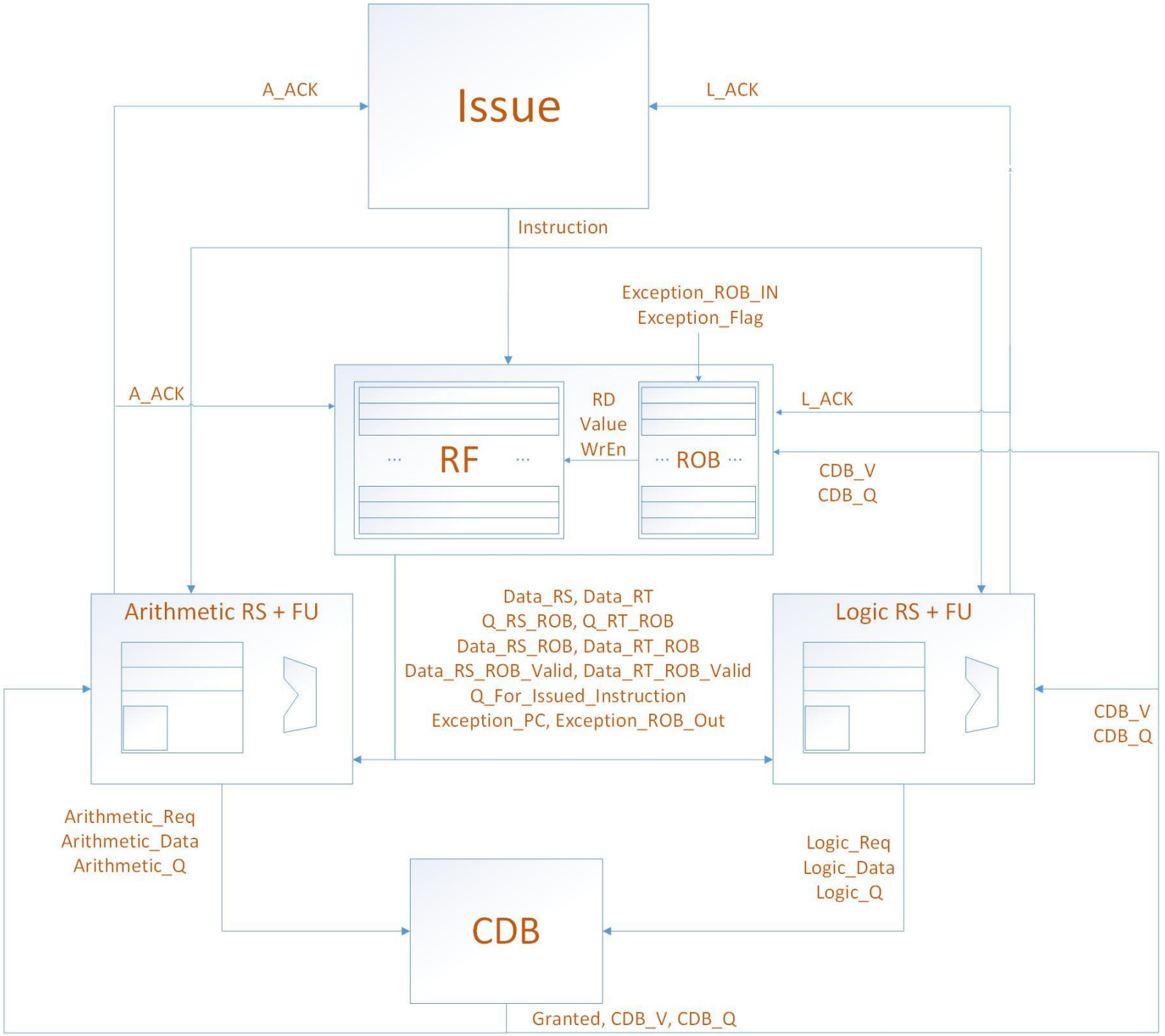
Interface

Πλέον το *ROB* γίνεται ένα component σε συνδυασμό με την *ARF* κρατώντας το Interface της *ARF* με τα υπόλοιπα components σχεδόν ίδιο. Η μόνη αλλαγή είναι πως πλέον το σήμα *Q_For_Issued_Instruction* είναι έξοδος και όχι είσοδος και δίνεται από το *ROB*. Επίσης προστέθηκαν τα σήματα σχετικά με τα *exception.*



Σχηματικό διάγραμμα του Reorder Buffer μαζί με την ARF με τις εισόδους και τις εξόδους τους.

Συνολικό σχηματικό διάγραμμα (Με το ROB)



Σχηματικό διάγραμμα του αλγορίθμου του Tomasulo με Reorder Buffer που υλοποιήθηκε.

Επιβεβαίωση Ορθής Λειτουργίας - Προσομοίωση

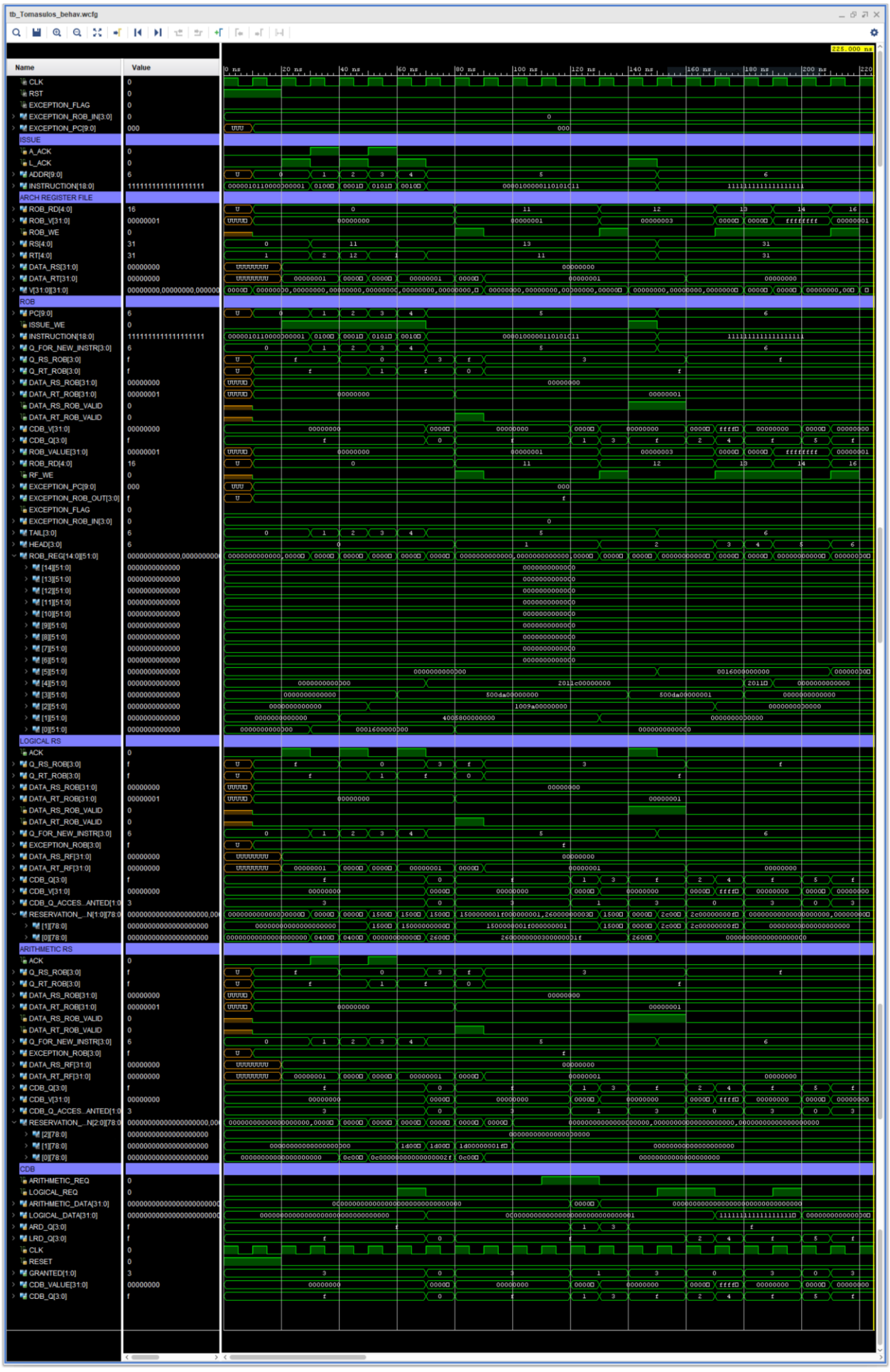
Η ορθή λειτουργία της υλοποίησης του αλγορίθμου με το *ROB* φαίνεται στα αποτελέσματα της παρακάτω προσομοίωσης. Η προσομοίωση ελέγχει τις παρακάτω ακραίες περιπτώσεις:

- CDB broadcasts και εξαρτήσεις (*Reservation Stations* tags): Εντολές 1-3:
- Εντολή 4: Νέα εντολή που γράφει στον καταχωρητή προορισμού προηγούμενης εντολής
- Εντολές 5-6: Ο ένας καταχωρητής πηγής παίρνει tag από το *ROB* και ο άλλος έγκυρα δεδομένα από την *ARF.* Έλεγχος write-first (CDB broadcast και Commit για τον ίδιο καταχωρητή)
- Εντολή 6: Αργεί να γίνει αποδεκτή αφού δεν υπάρχει διαθέσιμο *Reservation Station.*

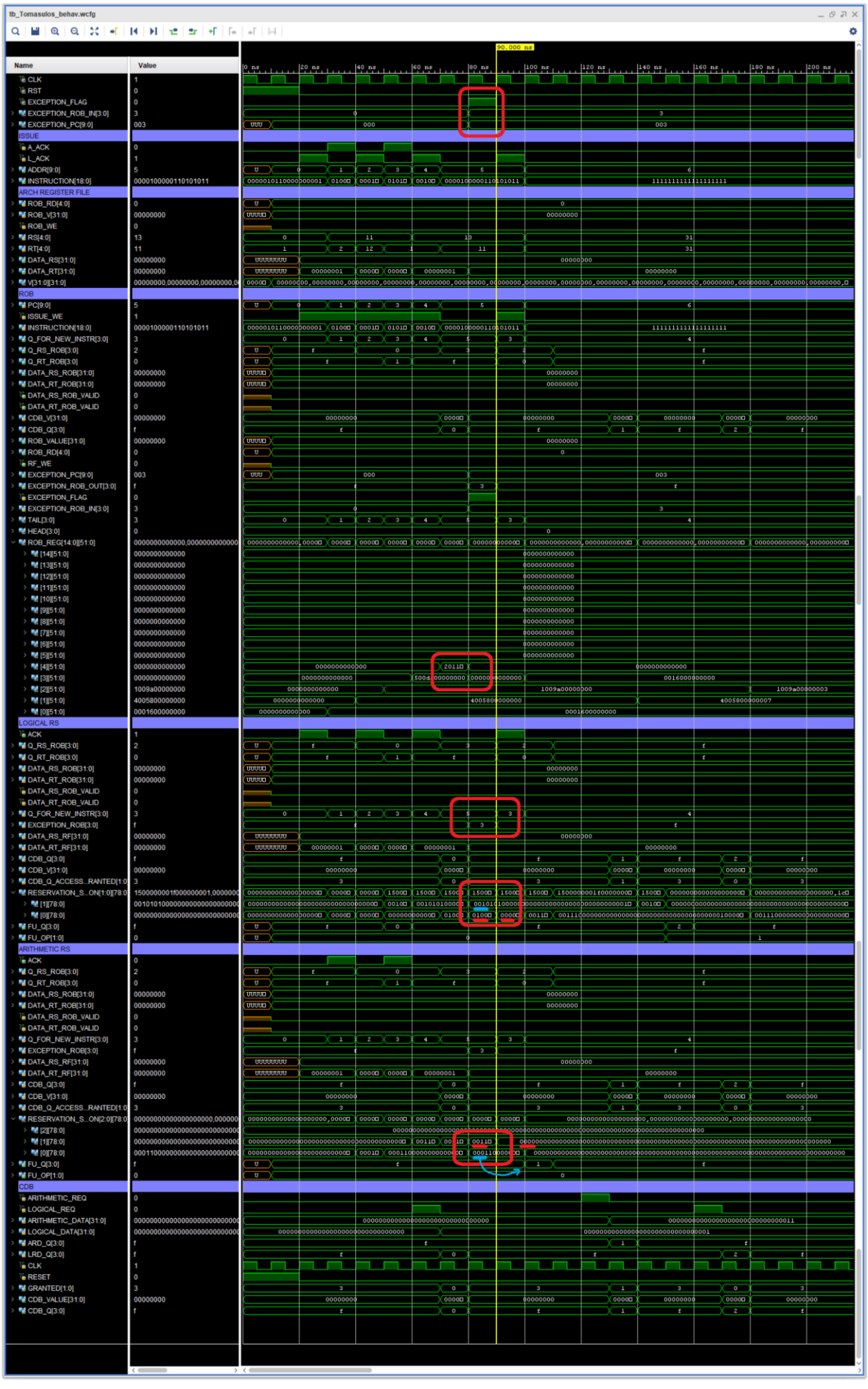
Ακολουθεί μια ροή του προγράμματος που προκαλεί τις παραπάνω περιπτώσεις.

	Type	FOP	RD	RS	RT
INSTR 1	00	00	01011	00000	00001
	LOGICAL	OR	11	0	1
INSTR 2	01	00	01100	01011	00010
	ARITHM	ADD	12	11	2
INSTR 3	00	01	01101	01011	01100
	LOGICAL	AND	13	11	12
INSTR 4	01	01	01101	01011	00001
	ARITHM	SUB	13	11	1
INSTR 5	00	10	01110	01101	01011
	LOGICAL	NOT	14	13	11
INSTR 6	00	00	10000	01101	01011
	LOGICAL	OR	16	13	11

Behavioral Simulation – Tomasulo με ROB



Behavioral Simulation – Tomasulo με ROB και exception



Στην πρώτη εξομοίωση παρατηρούμε την έκδοση των εντολών ανά κύκλο εφόσον υπάρχει διαθέσιμος χώρος για αυτές, την εκτέλεση τους εκτός σειράς, και την σε σειρά επενέργεια των αποτελεσμάτων στην *ARF* (commit).

Στη δεύτερη εξομοίωση συμβαίνει ένα exception στην εντολή του *ROB3* όταν στο σύστημα έχουν μπει 5 εντολές. Στον ίδιο κύκλο αδειάζουν οι καταχωρήσεις 3 και 4 του *ROB* και ειδοποιούνται τα *Reservation Stations,* τα οποία στον επόμενο κύκλο διαγράφουν τις αντίστοιχες καταχωρήσεις για τις εντολές *ROB3* και *ROB4*, διατηρώντας και συνεχίζοντας την εκτέλεση των εντολών *ROB0*, *ROB1* και *ROB2*.