# HPY415 Αρχιτεκτονική Υπολογιστών
# Αναφορά Εργαστηρίου 3

Φλέγγας Γεώργιος 2014030161
17/12/2018

## Reorder Buffer:

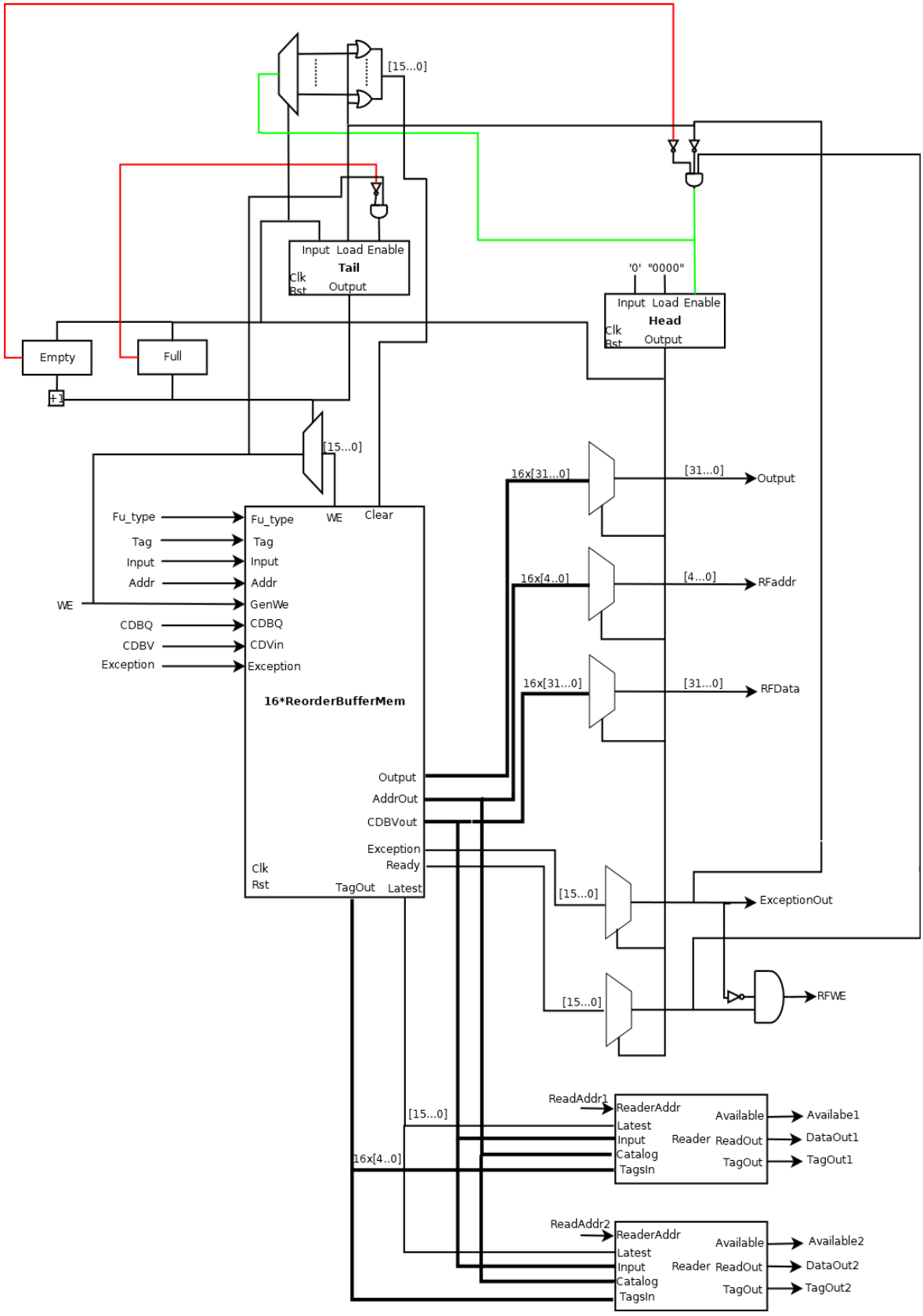
Ουσιαστικά το Reorder Buffer (ROB) είναι μια (δομή δεδομένων) κυκλική ουρά (FIFO – First In First Out). Κάθε φορά που εκδίδεται μια καινούρια εντολή στο σύστημα εγγράφεται πρώτα στο ROB. Στη συνέχεια το σύστημα λειτουργεί όπως ακριβώς λειτουργούσε και πριν, με τη μόνη διαφορά πως πλέον οι εξαρτήσεις μεταξύ των εντολών δεν υφίστανται με βάση τα Reservation Stations αλλά μεταξύ των επιμέρους εγγραφών στο ROB.
Για την δημιουργία του ReorderBuffer χρειάστηκαν ReorderBufferMem. Αυτά έχουν την παρακάτω δομή :

| **Inputs:** | **Outputs:** |
|---|---|
| Clk<br>Rst<br>Fu_type 2 bits<br>Tag 5 bits<br>Addr 5 bits<br>Input 32 bits<br>WE<br>GenWe<br>CDBQ 5 bits<br>CDBVin 32 bits<br>Clear<br>Exception | TagOut 5 bits<br>Output 32 bits<br>Latest<br>ExceptionOut<br>Ready<br>CDBVout 32 bits<br>AddrOut 5 bits<br>Fu_type_out 2 bits |

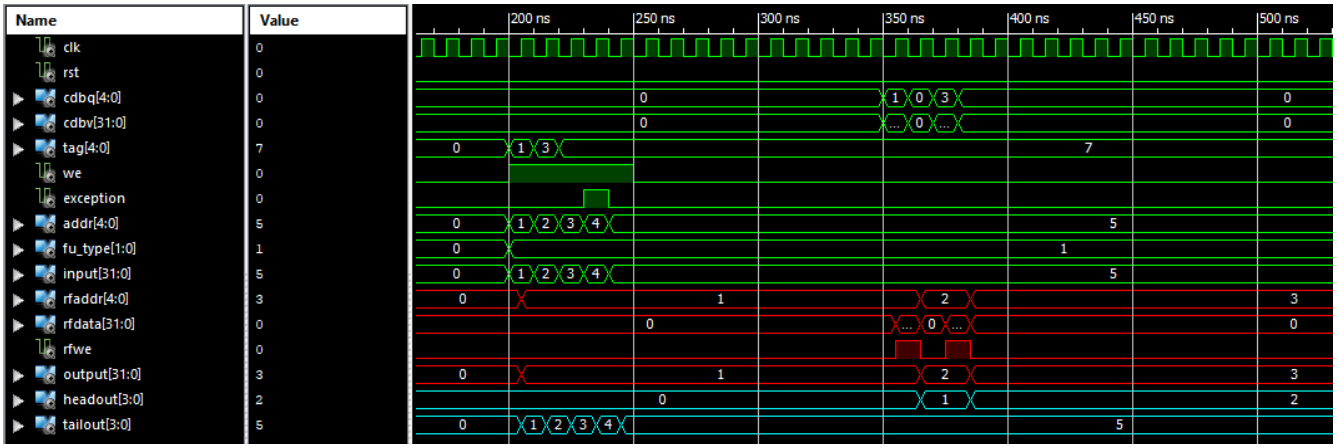
Μέσα σε αυτά γίνεται έλεγχος του CDB για τα κατάλληλα δεδομένα, και όταν αυτά βρεθούν, καταγράφονται ως *CDBVout* και το *Ready* γίνεται ‘1’. Με το σήμα *Clear* μηδενίζεται το *Ready,* όταν εκτελεστούν(Commit), ή όταν ο ROB ανιχνεύσει exception. Αξιοσημείωτο είναι επίσης το πεδίο “Newest”. Όταν μια θύρα του RoB αναζητά δεδομένα που προορίζονται για ένα συγκεκριμένο *Addr*, θα πρέπει να βρει πάντα τα πιο πρόσφατα. Για αυτό χρησιμοποιείται το *Latest*. Όταν πάρει την τιμή ‘1’, σημαίνει ότι τα δεδομένα μόλις έγιναν issue στο ROB. Αντιθέτως γίνεται '0' όταν εντοπίσει το ίδιο *Addr* στην είσοδό του με αυτό που υπάρχει στην μνήμη του, χωρίς όμως να είναι ενεργοποιημένο το *WE* του. Αυτό σημαίνει ότι κάποια νεότερη εντολή με αυτό το *Addr* εισέρχεται στο RoB.

Το Reorder Buffer αποτελείται από 16 ReorderBufferMem σε κατανομή ουράς. Για να επιτευχθεί αυτή η κατανομή χρησιμοποιήθηκαν δύο μετρητές, οι οποίοι ορίζουν τους pointers “Head” και “Tail”. Το Head και το Tail αποτελούν την αρχή και το τέλος της ουράς. Και τα δύο αυξάνονται προς τη ίδια κατεύθυνση. Το Tail δείχνει πάντα την θέση που θα εισαχθεί η επόμενη εντολή που θα εκδοθεί στο σύστημα. 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 βγαίνει από τη θέση που δείχνει το Head, κι αυτό αυξάνεται κατά 1. Ακόμα για την ορθή λειτουργία του συστήματος υπάρχουν 2 *ROBRead* τα οποία σε περίπτωση που τα ζητηθούν, βρίσκονται μέσα στο ROB βγαίνουν ως έξοδοι . Για να γίνει αυτό χρησιμοποιούνται τα σήματα *Available1* και *Available2*.

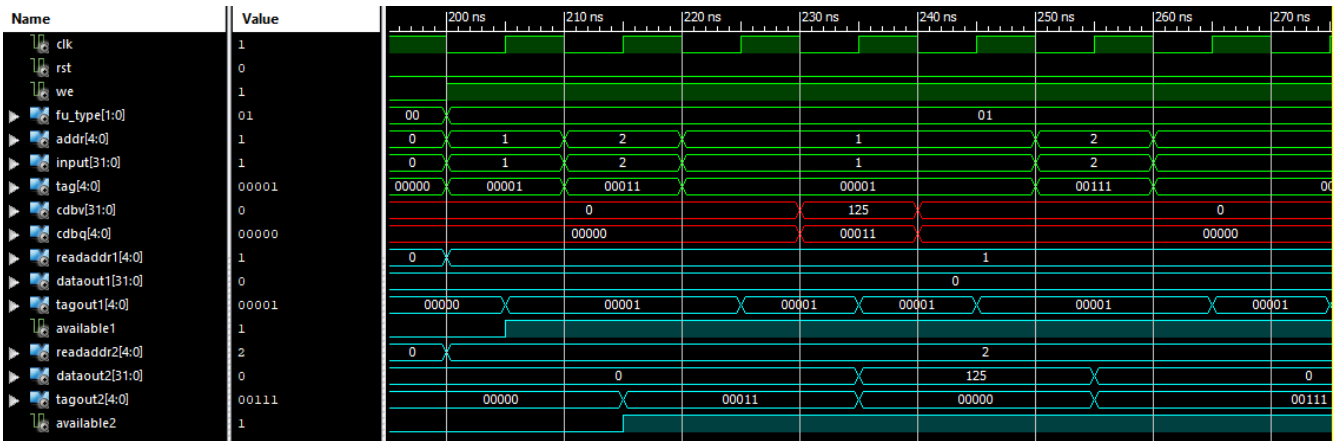
Η αναζήτηση γίνεται με βάση το *tag* και το *Latest*. Τέλος κατά την σχεδίαση του ROB λήφθηκε υπόψιν η περίπτωση που το ReorderBufferMem το οποίο πρόκειται να γίνει commit έχει υποστεί exception. Σε αυτην την περίπτωση το RoB αδειάζει την λίστα του χωρίς να κάνει commit τίποτα και εξάγει τα σήματα *ExceptionOut='1'* , και *Output* (αποτελεί το PC της εντολής για την οποία έγινε exception) τα οποία σηματοδοτούν την ύπαρξη exception.
Στο σύνολο του ο Reorder Buffer έχει την ακόλουθη μορφή :



## TestBench:

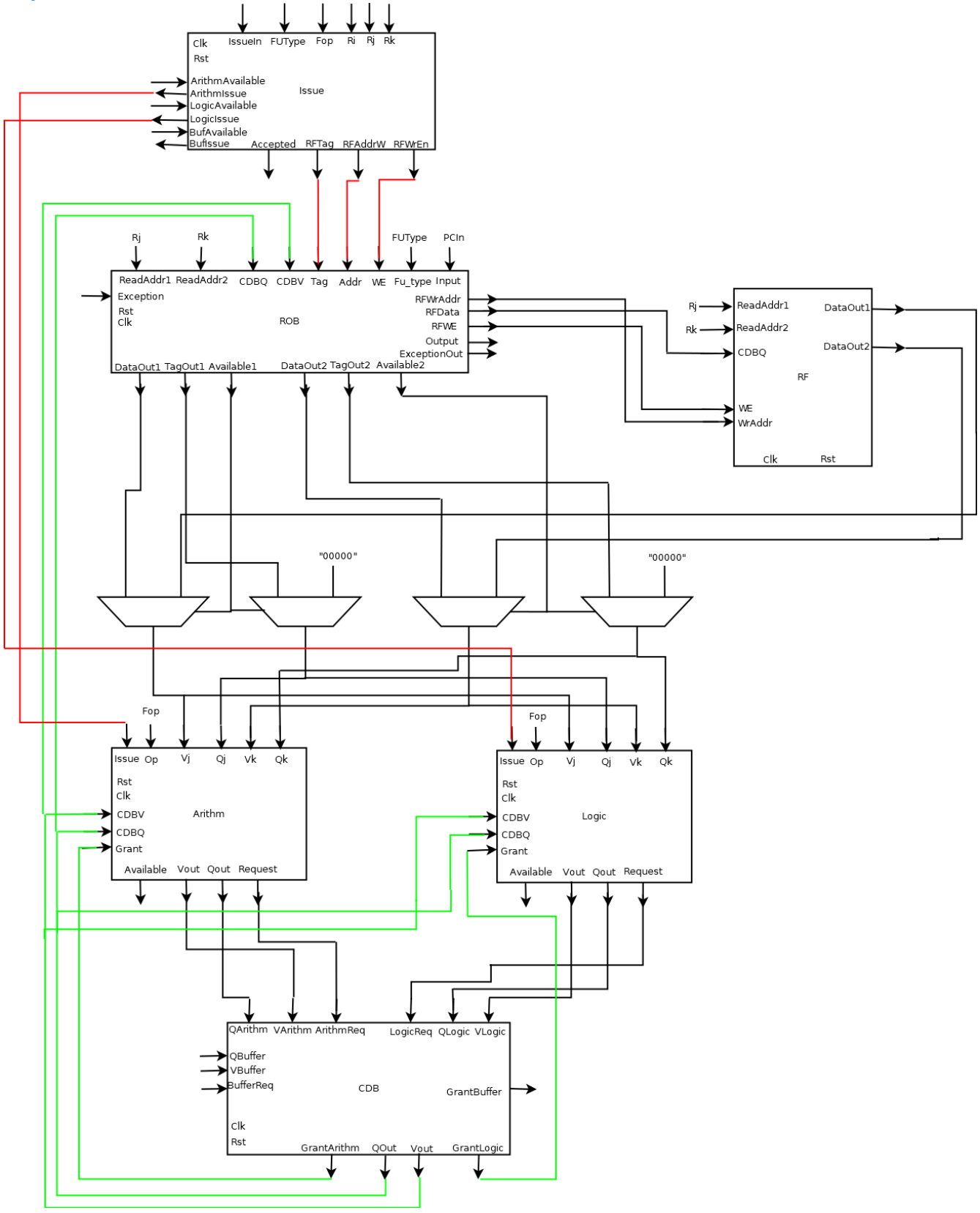



Γίνεται αρχική εισαγωγή στο ROB των στοιχείων Addr=1,2,3,4,5 με Input=1,2,3,4,5 αντίστοιχα. Όταν τα CDBQ CDBVin πάρουν τιμές για τις οποίες παρατηρούμε ότι CDBQ=1=Tag[1] και CDBQ=3=Tag[2], τότε οι αντίστοιχες τιμές των CDBVin εξάγοντε προς την RF.




Αρχικά το ROB γεμιζει 3 mem με εντολές οι οποίες έχουν Tag: ”00011”,”00001” και Addr: 1,2. Για τον έλεγχο των readers του ROB, οι ReadAddr 1 και 2 ορίζονται ως “1” και “2” αντίστοιχα. Μπορεί να παρατηρηθεί ότι μετά το γέμισμα του πρώτου mem το σήμα Available1 σηματοδοτεί ότι το ROBReader1 έχει τα κατάλληλα δεδομένα και το σήμα TagOut1 εξάγει το Tag της εντολής (“00001”). Στον αμέσως επόμενο κύκλο το σήμα “Available2” σηματοδοτεί ότι το ROBReader2 έχει τα κατάλληλα δεδομένα και το σήμα TagOut2 εξάγει το Tag της εντολής (“00011”). Ταυτόχρονα στον ίδιο κύκλο το CDB δίνει τα κατάλληλα δεδομένα για αυτήν την εντολή, άρα και στον επόμενο κύκλο η port2 εξάγει τα κατάλληλα δεδομένα στο σήμα “DataOut2” και μηδενίζει το σήμα “TagOut2”.

## Top Module


Clk
IssueIn
FUType
Fop
Ri
Rj
Rk
Rst
ArithmAvailable
ArithmIssue
LogicAvailable
LogicIssue
BufAvailable
BufIssue
Issue
Accepted
RFTag
RFAddrW
RFWrEn
Rj
Rk
FUType
PCIn
ReadAddr1
ReadAddr2
CDBQ
CDBV
Tag
Addr
WE
Fu_type
Input
Exception
Rst
Clk
ROB
RFWrAddr
RFData
RFWE
Output
ExceptionOut
DataOut1
TagOut1
Available1
DataOut2
TagOut2
Available2
Rj
Rk
ReadAddr1
ReadAddr2
CDBQ
RF
DataOut1
DataOut2
WE
WrAddr
Clk
Rst
"00000"
"00000"
Fop
Fop
Issue
Op
Vj
Qj
Vk
Qk
Rst
Clk
CDBV
CDBQ
Grant
Arithm
Available
Vout
Qout
Request
Issue
Op
Vj
Qj
Vk
Qk
Rst
Clk
CDBV
CDBQ
Grant
Logic
Available
Vout
Qout
Request
QArithm
VArithm
ArithmReq
LogicReq
QLogic
VLogic
QBuffer
VBuffer
BufferReq
CDB
GrantBuffer
Clk
Rst
GrantArithm
QOut
Vout
GrantLogic

Οι αλλαγές που πραγματοποιήθηκαν σε σχέση με το 2ο εργαστήριο, πέρα από την υλοποίηση και την προσαρμογή του ROB, είναι οι εξής :

- Απλοποίηση του Register File, πλέον κάνει απλή αποθήκευση των τιμών όταν δεχτεί WE. Δηλαδή αφαιρέθηκαν τα registers των Tags, καθώς και ο έλεγχος που γινόταν μέσω αυτών και του CDBQ.
- Προσθήκη μιας απλής λειτουργίας exception handling, που ενεργοποιείται με ένα σήμα Exception, και ουσιαστικά υλοποιείται μέσα στο ROB.
- Προσθήκη κάποιον αλλαγών στην λογική που λειτουργεί ο CDB και το IssueUnit, ώστε να μπορεί το σύστημα να υποστηρίξει χρήση Buffer, χωρίς όμως να υλοποιείται ο συνολικός buffer.
- Αλλαγή της λογικής με την οποία κάνουμε load τιμές στον RF, ώστε να αξιοποιηθούν οι νέες λειτουργίες της IssueUnit και του CDB.

Ο RoB συνδέεται με το Issue Unit για την λήψη εντολών, με τον CDB για εισαγωγή δεδομένων, και με την Register File για την εξαγωγή των τελικών αποτελεσμάτων. Ακόμη με την χρήση πολυπλεκτών, γίνετε η επιλογή των δεδομένων που θα περάσουν στα RS.

## **TestBench:**

Κατά τον έλεγχο πραγματοποιήθηκαν οι ακόλουθες πράξεις :

*ld $1,1*
*ld $2,5*
*add $3, $1, $2*
*or $4,$3,$1*
*not $5,$4*
*and $6,$2,$1*
*sub $7,$5,$6*