Αρχιτεκτονική Υπολογιστών

Υλοποίηση Αλγόριθμου Tomasulo #2

Κωνσταντίνος Κασφίκης – 2013030108
Νίκος Κανάκης – 2013030175

Εισαγωγή

Στο παρόν εργαστήριο μας ζητήθηκε να συνδέσουμε τα επιμέρους κομμάτια του προηγούμενου εργαστηρίου , υλοποιώντας κατά αυτόν τον τρόπο έναν επεξεργαστή που λειτουργεί βασισμένος στον αλγόριθμο Tomasulo. Ο συγκεκριμένος επεξεργαστής στην τρέχουσα σχεδίαση είναι σε θέση να εκτελέσει τις ακόλουθες εντολές :

Logical : NOT/AND/OR

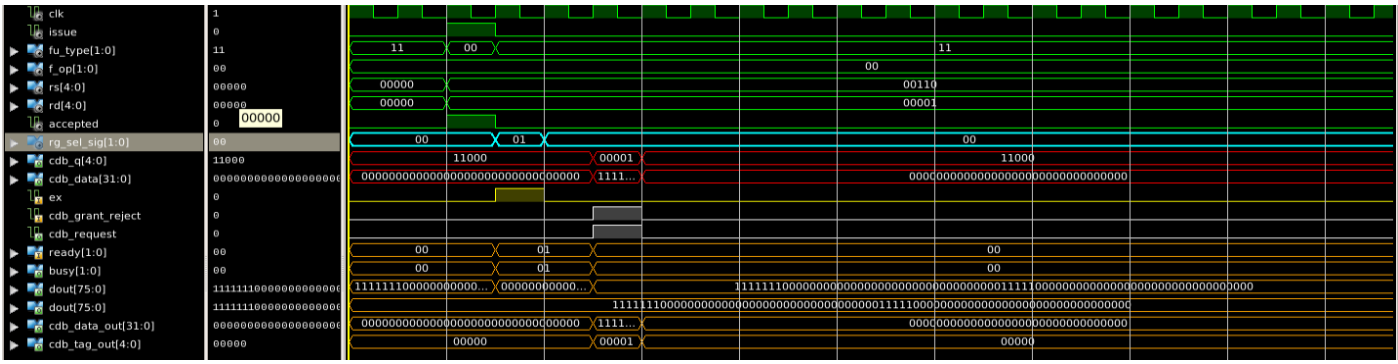
Arithmetic : ADD/SUB

Κατά την σύνδεση των επιμέρους κομματιών προέκυψαν λάθη χρονισμών , οπότε υπήρξε η ανάγκη αλλαγής της λογικής κάποιων modules, όπως του CDB και των Functional Unit. Για την πλήρη ομοχειρία του Functional Unit προσθέσαμε 2 και 3 register αντίστοιχα. Μέσω την προσθήκης των παραπάνω register , πλέον τα FU είναι σε θέση να εξυπηρετήσουν ταυτόχρονο 2 ή 3 εντολές αντίστοιχα. Ωστόσο , για την καλύτερη σύγκριση των αποτελεσμάτων μας, θα παραθέσουμε και simulation από τον αντίστοιχο επεξεργαστή χωρίς ομοχειρία στις FU.

Εκτέλεση μεμονωμένης εντολής

Παρακάτω παρουσιάζονται simulation εκτέλεσης 2 μεμονωμένων εντολών, μια για κάθε FU type.

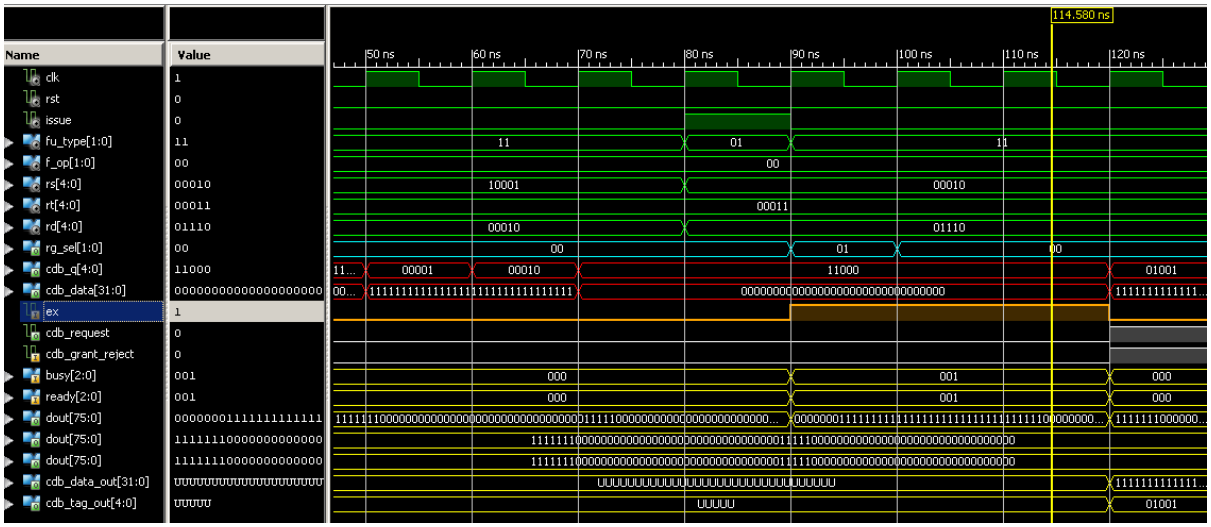
NOT (FU TYPE : "00" – FU OP : "00")



Η παραπάνω not με καταχωρητή προορισμού τον καταχωρητή 1 και Rs τον 6, στον πρώτο κύκλο από την εκδοσή της βρίσκεται στον Issuer , ο οποίος βάσει των rs,rt,rd θέτει τις εισόδους του Register File, ενώ παράλληλα σε επικοινωνία με τον ελεγκτή του Reservation

Station 1, δίνει κατάλληλο σήμα ώστε το κατάλληλο reservation station να γράψει στον επόμενο κύκλο, τα δεδομένα που θα εμφανίστουν στις εξόδους του Register File. Πράγματι παρατηρούμε στην έξοδο του πρώτου reservation station (το πρώτο σήμα dout με πορτοκαλί έξοδο), μετά από έναν κύκλο τα δεδομένα του καταχωρητή 6 (μηδενικά λόγω reset του rf). Στον ίδιο κύκλο, εφόσον το πρώτο reservation station είναι έτοιμο προς εκτέλεση παρατηρούμε ότι το rg_sel παίρνει τιμή 01 , που σημαίνει ότι το πρώτο reservation station έχει επιλεχτεί για εκτέλεση από το FU του. Μαζί με το rg_sel ενεργοποιείται και το σήμα ex που σηματοδοτεί πότε το FU πρέπει να δεχτεί τις εισόδους του και να προχωρήσει σε κοινοποίηση αποτελεσμάτων. Μετά από 2 κύκλους από την ενεργοποίηση του ex του FU 2 βλέπουμε ότι ενεργοποιείται το σήμα cdb_request (άσπρο χρώμα) του FU 2 και στον ίδιο κύκλο έρχεται έγκριση για κοινοποίηση (cdb_grant_reject – άσπρο χρώμα). Τελικά, εντός κύκλου, κοινοποιούνται τα αποτελέσματα της πράξης με το κατάλληλο tag (00001) (cdb_tag και cdb_data - κόκκινο χρώμα).

<u>ADD (FU_TYPE : “01” – FU_OP : ”00”)</u>



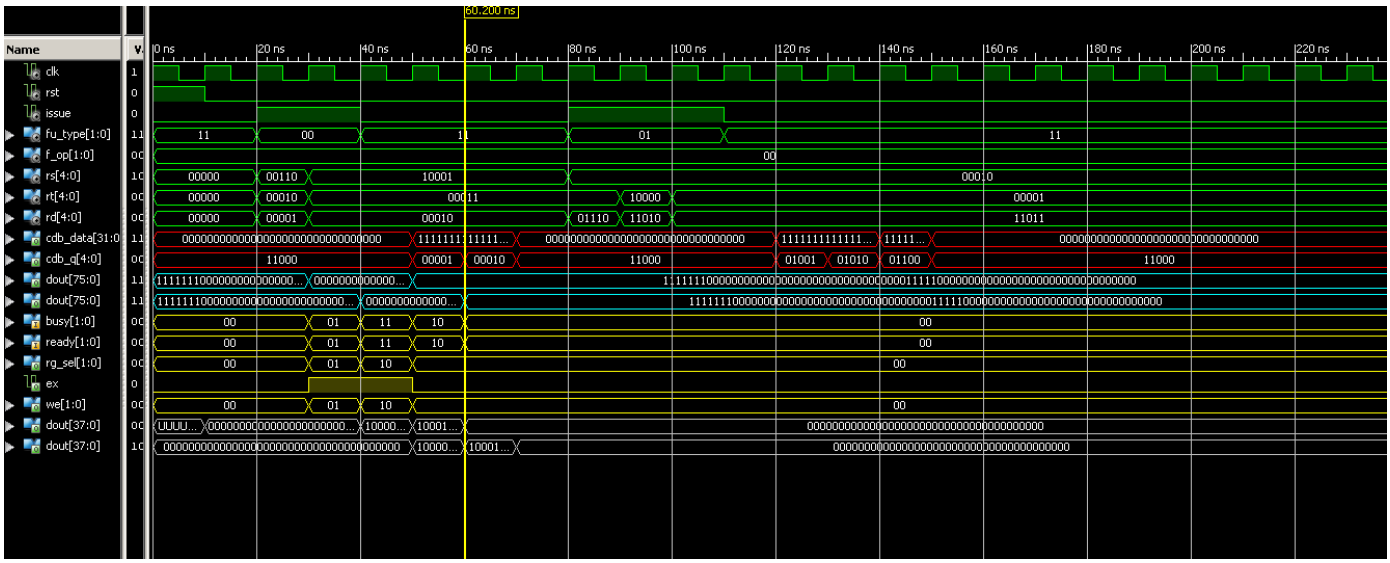
Προκειμένου να δούμε κάποιο διαφορετικό αποτέλεσμα από μηδενικά , αρχικοποιούμε 2 καταχωρητές πριν από αρκετούς κύκλους με άσσους (με χρήση 2 εντολών not), και στην συνέχεια εκτελούμε εντολή add με ορίσματα τις τιμές των 2 παραπάνω καταχωρητών. Στο παραπάνω simulation τα σήματα που παρουσιάζονται είναι αντίστοιχα με του simulation της not αλλά αφορούν το FU_1. Με αντίστοιχη λογική όπως και στην not, τα αποτελέσματα της πράξης add κοινοποιούνται μετά από 3 κύκλους από την αρχή λειτουργίας της FU και 4 κύκλους μετά την έκδοση της εντολής.

Pipelined εκτέλεση εντολών

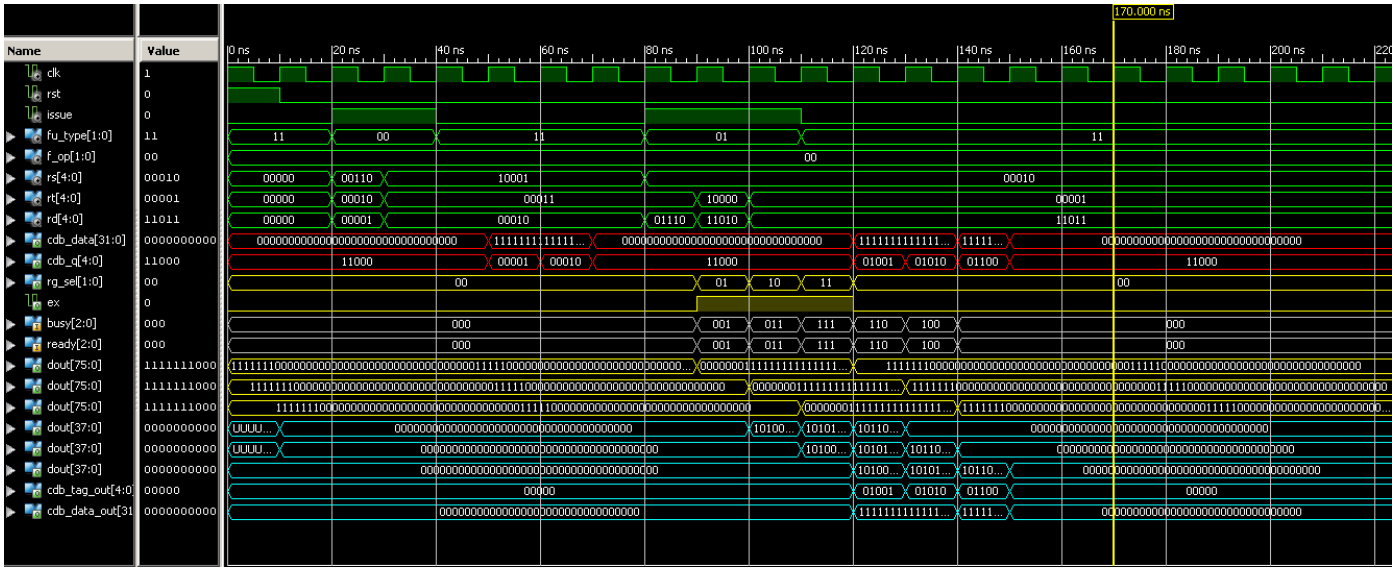
Προκειμένου να δοκιμάσουμε την ομοχειρία του επεξεργαστή, μέσω του testbench δρομολογούμε 5 εντολές, 2 εκ των οποίων είναι not – οι οποίες θα αποθηκεύσουν άσσους στους καταχωρητές προορισμού τους – και οι υπόλοιπες είναι add – οι 2 πρώτες χρησιμοποιούν σαν πρώτο όρισμα έναν από τους καταχωρητές των not και ως δεύτερο όρισμα μηδενικά από έναν αρχικοποιημένο καταχωρητή , ενώ η τρίτη πρόσθεση έχει ως ορίσματα τα αποτελέσματα και των 2 not.

Η ομοχειρία του επεξεργαστή μας φαίνεται από τις διαδοχικές κοινοποιήσεις των αποτελεσμάτων στο CDB , σε αντίθεση με το simulation του μη pipelined FU όπου οι διαδοχικές logical εντολές κοινοποιούνται με δύο κύκλους διαφορά ενώ οι διαδοχικές arithmetic με τρεις κύκλους αντίστοιχα.

Όπως προαναφέρθηκε, προκειμένου να εκτελούμε στο FU εντολές pipelined χρησιμοποιήθηκαν 3 και 2 καταχωρητές στο FU1 και FU2 αντίστοιχα. Ο πρώτος καταχωρητής, εφόσον υπάρχουν έγκυρα δεδομένα από κάποιο reservation station τα οποία έχουν διαλεχθεί μέσω του rg_sel, γεμίζει με το αποτέλεσμα της πράξης. Τα δεδομένα αυτά στους επόμενους κύκλους θα μεταφερθούν στους επόμενους registers μέχρι να φτάσουν στον 3ο και στον 2ο register αντίστοιχα όπου και θα παραμείνουν μέχρι να δοθεί ενεργό σήμα cdb_grant_reject από το CDB. Σε διαδοχικές εντολές , κάθε register του FU θα φιλοξενεί και διαφορετική εντολή. Εκτός από τους register, για την πλήρη ομοχειρία των FU απαιτείται και έλεγχος μέσα σε process για την ενεργοποίηση/απενεργοποίηση σημάτων, τόσο ώστε να πραγματοποιηθεί αίτηση κοινοποίησης και κατάλληλη διαχείριση cdb_grant_request, όσο και στο να παραχθεί σήμα FU_busy (το FU δεν δέχεται εντολές όσο είναι ενεργό το FU_busy), το οποίο ενεργοποιείται μόνο εφόσον δεν έχουμε έγκριση κοινοποίησης αμέσως μετά την αίτηση.



Στο παραπάνω σχήμα φαίνονται τα σήματα ελέγχου busy,ready,ex,rg_sel των logical reservation stations (κίτρινο χρώμα) , τα περιεχόμενα των rs (μπλε χρώμα) καθώς και οι registers του FU (γκρι χρώμα). Λόγω ομοχειρίας των functional unit παρατηρούμε ότι οι κοινοποιήσεις στο cdb γίνονται σε διαδοχικούς κύκλους . Οι δυο πρώτες not βγάζουν 32 άσσους με αντίστοιχα tag 00001(πρώτο logical rs) και 00010(δεύτερο logical rs). Παράλληλα με την κοινοποίηση παρατηρούμε ότι τόσο το εκάστοτε RS όσο και ο τελευταίος register του functional unit γίνοται reset ώστε να μπορούμε να συνεχίσουμε σε επόμενες εντολές.



Στη συνέχεια δίνουμε τρείς διαδοχικές εντολές add οι οποίες δέχονται ως ορίσματα τους καταχωρητές που έχουν πάρει άσσους από τις προηγούμενες Not. Βλέπουμε με τα ίδια χρώματα τα σήματα ελέγχου καθώς και τις διαδοχικές κοινοποιήσεις των αποτελεσμάτων και των αντίστοιχων tag από το cdb. Το cdb_data στην τρίτη κοινοποίηση αλλάζει τιμή από 32 άσσους σε 31 άσσους και παράγεται τιμή overflow (καθώς προσθέτουμε και τους 2 καταχωρητές).

Τέλος παραθέτουμε το simulation του μη pipelined FU, στο οποίο παρατηρούμε ότι για τις δύο ίδιες εντολές not χρειαζόμαστε 4 κύκλους για τις κοινοποιήσεις.