Οργάνωση Υπολογιστών

Αναφορά Εργασίας #1

Φοιτητής : Κωνσταντίνος Δανόπουλος

AM : 2016030131

Ο στόχος της πρώτης εργασίας ήταν η κατασκευή ενός Mips like single-cycle επεξεργαστή ο οποίος να εκτελεί την κάθε εντολή μέσα σε έναν κύκλο. Οπότε οποιαδήποτε εντολή εκτελεστεί θα χρειαστεί ένας κύκλος, παρόλο που κάποιες εντολές δεν χρειάζονται όλα τα modules του επεξεργαστή για να εκτελεστούν. Επομένως ο χρόνος του ενός κύκλου στον συγκεκριμένα επεξεργαστή θα πρέπει να είναι πιο μεγάλος από τον κύκλο των multi-cycle επεξεργαστών διότι δεν γίνεται να είναι μικρότερος από τον χρόνο που χρειάζεται να περάσει η εντολή από όλα τα modules του επεξεργαστή. Ουσιαστικά έχει μια minimum τιμή το clock το οποίο δεν μπορεί να τεθεί πιο χαμηλά από αυτήν. Αυτή η τιμή καθορίζεται από το σύνολο των καθυστερήσεων που έχουν τα modules του επεξεργαστή μέχρι τα σήματα να διασχίσουν όλο το datapath.

Ο Single Cycle Processor που υλοποιήθηκε αποτελείται από τα εξής επιμέρους modules :

1. Control
2. Ram
3. Datapath

Τα inputs και outputs του control είναι τα εξής :

```
entity CONTROL is
    Port (
        exstage_ALU_zero : in STD_LOGIC;
        exstage_ALU_ovf : in STD_LOGIC;
        exstage_ALU_cout : in STD_LOGIC;
        Instruction : in STD_LOGIC_VECTOR (31 downto 0);
        CLK : in STD_LOGIC;
        Reset : in STD_LOGIC;
        control_signals : out STD_LOGIC_VECTOR (14 downto 0));
end CONTROL;
```

Το module του control ουσιαστικά λειτουργεί σαν ένας decoder με Clock ο οποίος δέχεται ως είσοδο το ρολόι CLK , ένα σήμα Reset και ένα 32 bit σήμα το οποίο είναι το instruction dout που μας έχει επιστρέψει η μνήμη Ram για τον συγκεκριμένο κύκλο. Επίσης διαθέτει ως εισόδους τα output σήματα του ALU του επεξεργαστή που είναι το zero, το overflow και το counter out. Ως έξοδο επιστρέφει ένα σήμα 15 bits τα οποία είναι τα σήματα που ελέγχουν και ρυθμίζουν όλη την λειτουργία του επεξεργαστή ανάλογα με το ποια εντολή θέλουμε να εκτελέσουμε στον συγκεκριμένο κύκλο. Το ποια σήματα είναι αυτά και που είναι η θέση τους αποτυπώνονται στην παρακάτω φωτογραφία:

```
-- control_signals meaning...
--==========================
-- 0. Reset
-- 1. ifstage_pc_sel  ( 0->(PC+4) 1->((PC+4)+SignExt(Immed)*4) )
-- 2. ifstage_pc_LdEn  write at register at ifstage
-- 3. decstage_RF_WrEn
-- 4. decstage_RF_WrData_sel
-- 5. decstage_RF_B_sel
-- 6. decstage_ImmExt = bit 0
-- 7. decstage_ImmExt = bit 1
-- 8. exstage_ALU_Bin_sel
-- 9.  exstage_ALU_func bit 0
-- 10. exstage_ALU_func bit 1
-- 11. exstage_ALU_func bit 2
-- 12. exstage_ALU_func bit 3
-- 13. memstage_ByteOp
-- 14. memstage_Mem_WrEn
```

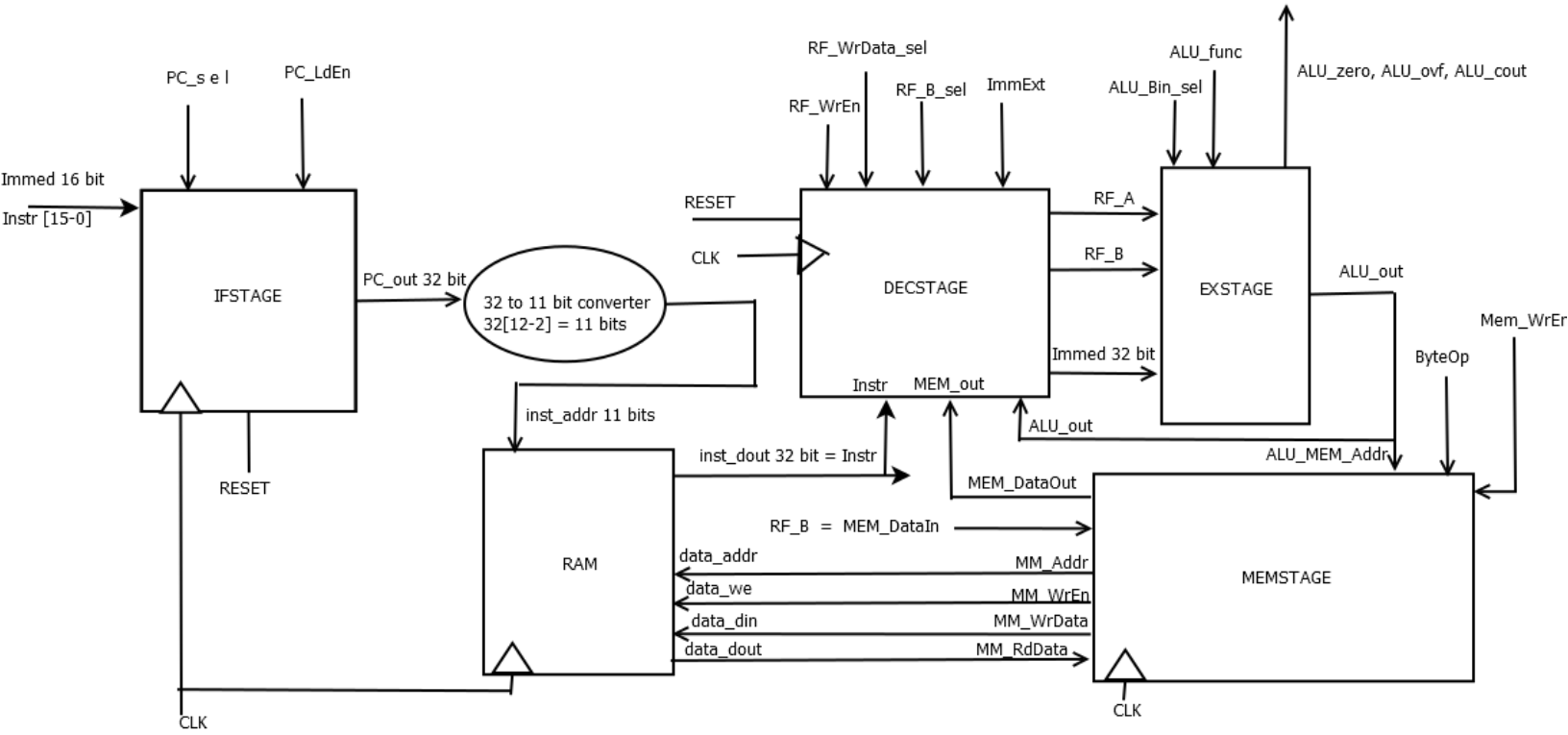
Στην συνέχεια τα παραπάνω σήματα μπαίνουν ως είσοδο στο module datapath του επεξεργαστή και κατανέμονται στις αντίστοιχες θέσεις τους ώστε να ελέγχουν όλα τα μέρη του επεξεργαστή.

Το module Ram του επεξεργαστή είναι η προσωρινή μνήμη που διαθέτει η οποία αποτελείται από 2048 θέσεις μνήμης των 32 bit. Οι πρώτες 1024 θέσεις μνήμης χρησιμοποιούνται για την αποθήκευση των εντολών που θέλουμε να εκτελέσει ο επεξεργαστής και αρχικοποιούνται μέσω ενός αρχείου rom.data που τοποθετούμε στον ίδιο φάκελο με τα αρχεία vhdl. Οι επόμενες 1024 θέσεις μνήμης χρησιμοποιούνται για την αποθήκευση των τιμών των καταχωρητών του επεξεργαστή που θέλουμε να αποθηκεύσουμε.

Το module Datapath αποτελεί το κύριο μέρος του επεξεργαστή στο οποίο πραγματοποιούνται και όλες οι λειτουργίες του που εκτελούν την κάθε εντολή. Το Datapath αποτελείται από τα εξής modules :

1. IFSTAGE
2. DECSTAGE
3. EXSTAGE
4. MEMSTAGE

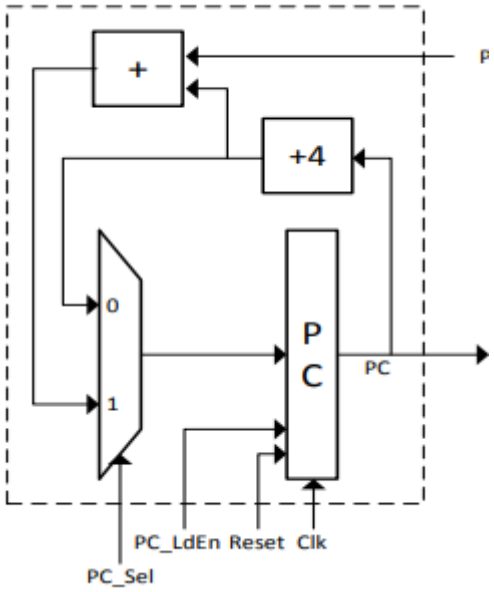
Το σχεδιάγραμμα που αποτυπώνονται όλες οι συνδέσεις του Datapath είναι το εξής :



*δεν βρίσκεται εντός του Datapath η μνήμη Ram αλλά τοποθετείται στο σχήμα για χάρη πληρότητας σχήματος

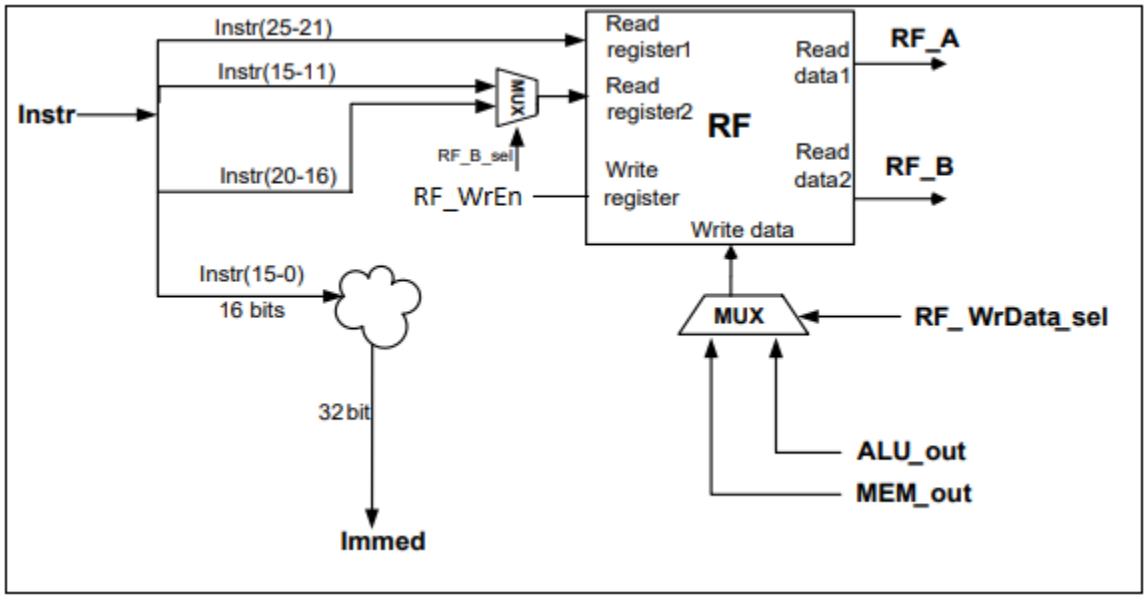
Το module IFSTAGE έχει φτιαχτεί ως structural και αποτελείται από τον Register, τον πολυπλέκτη 2 σε 1 , δύο adders και τον Immidiate 16-32 converter. Ο Immidiate converter δέχεται ως είσοδο το 16 bit Immidiate που μας δίνει το inst_dout την μνήμης Ram. Αρχικά το 16 bit Immidiate το κάνει shift κατά δύο προς τα αριστερά και στην συνέχεια το κάνει sign extention ώστε να γίνει 32 bit.

Οι υπόλοιπες λειτουργίες του IFSTAGE γίνονται ακριβώς όπως περιγράφει και η εκφώνηση υλοποιώντας το εξής σχήμα :



Immediate 32 bit

Το module DECSTAGE έχει αρχιτεκτονική structural και δέχεται ως είσοδο όλο το σήμα των 32 bit του output της Ram inst_dout. Στον RF των 32 θέσεων παρέχει μόνιμα ως είσοδο στον addr1 το [rs] του Instruction το οποίο είναι το Inst[25-21]. Στον addr2 αν ο RF_B_sel είναι 0 τότε παρέχει τον [rt] – Inst[15-21] ή αν είναι 1 τον [rd] – Inst[20-16]. Στον write_address του RF παρέχει μόνιμα τον [rd] – Inst[20-16].



Το module Immed_handler εντός του DECSTAGE δέχεται ως είσοδο το Immidiate 16 bit ή αλλίως Inst[15-0] και μέσω της μεταβλητής ImmExt του ορίζεται με ποιον τρόπο να μετατρέψει το 16 bit Immidiate σε 32 bit. Οι μετατροπές γίνονται με τον εξής τρόπο :

1. Αν ImmExt = 00 τότε Immidiate_32 = 00000000000000000000000000000000

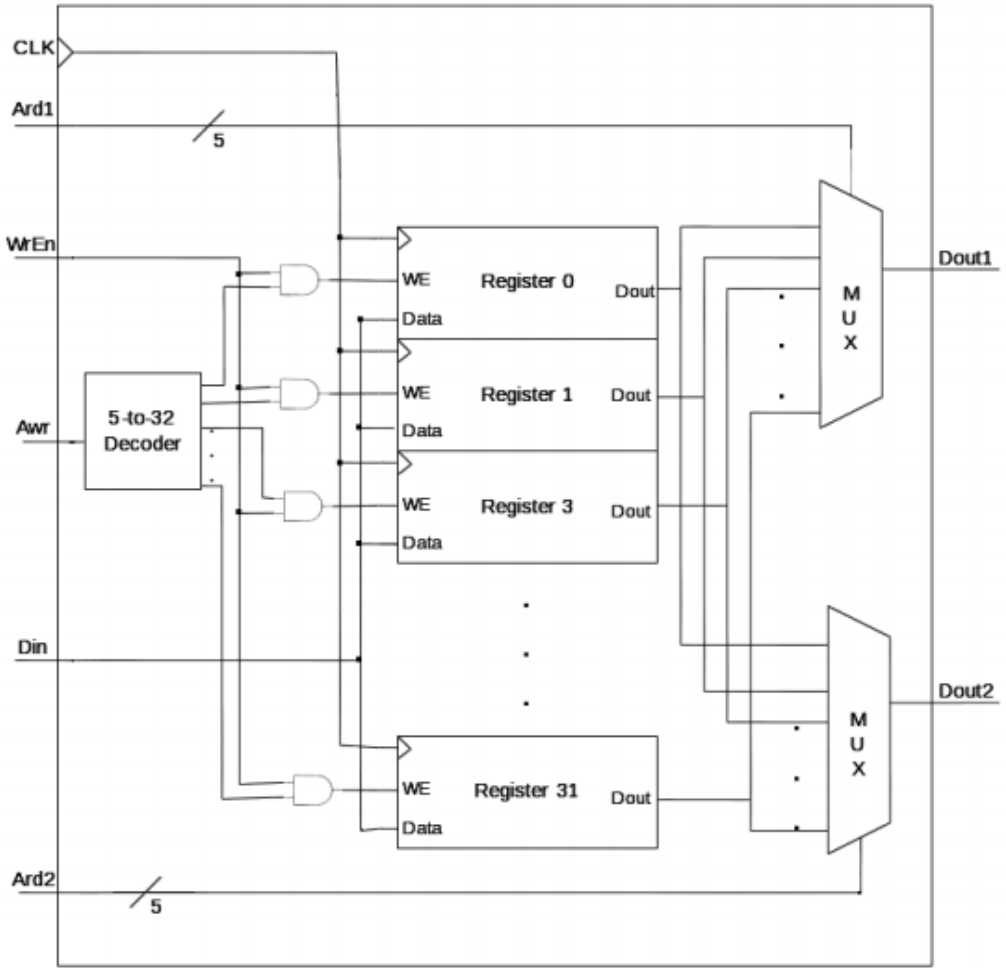
2. Αν ImmExt = 01 τότε Immidiate_32 = Sign Extend (Immidiate_16)
3. Αν ImmExt = 10 τότε Immidiate_32 = Zero Fill (Immidiate_16)
4. Αν ImmExt = 11 τότε Immidiate_32 = Shift 16 & Zero Fill (Immidiate_16)

Προκειμένου να επιτευχθεί η εκτέλεση πράξεων από καταχωρητές του RF και στην συνέχεια το αποτέλεσμα να μπορεί να αποθηκευτεί σε κάποιον άλλο καταχωρητή του RF εντός του ίδιου κύκλου χρειάστηκε να πραγματοποιηθεί πολύ debugging και να δοκιμαστούν πολλοί μέθοδοι.

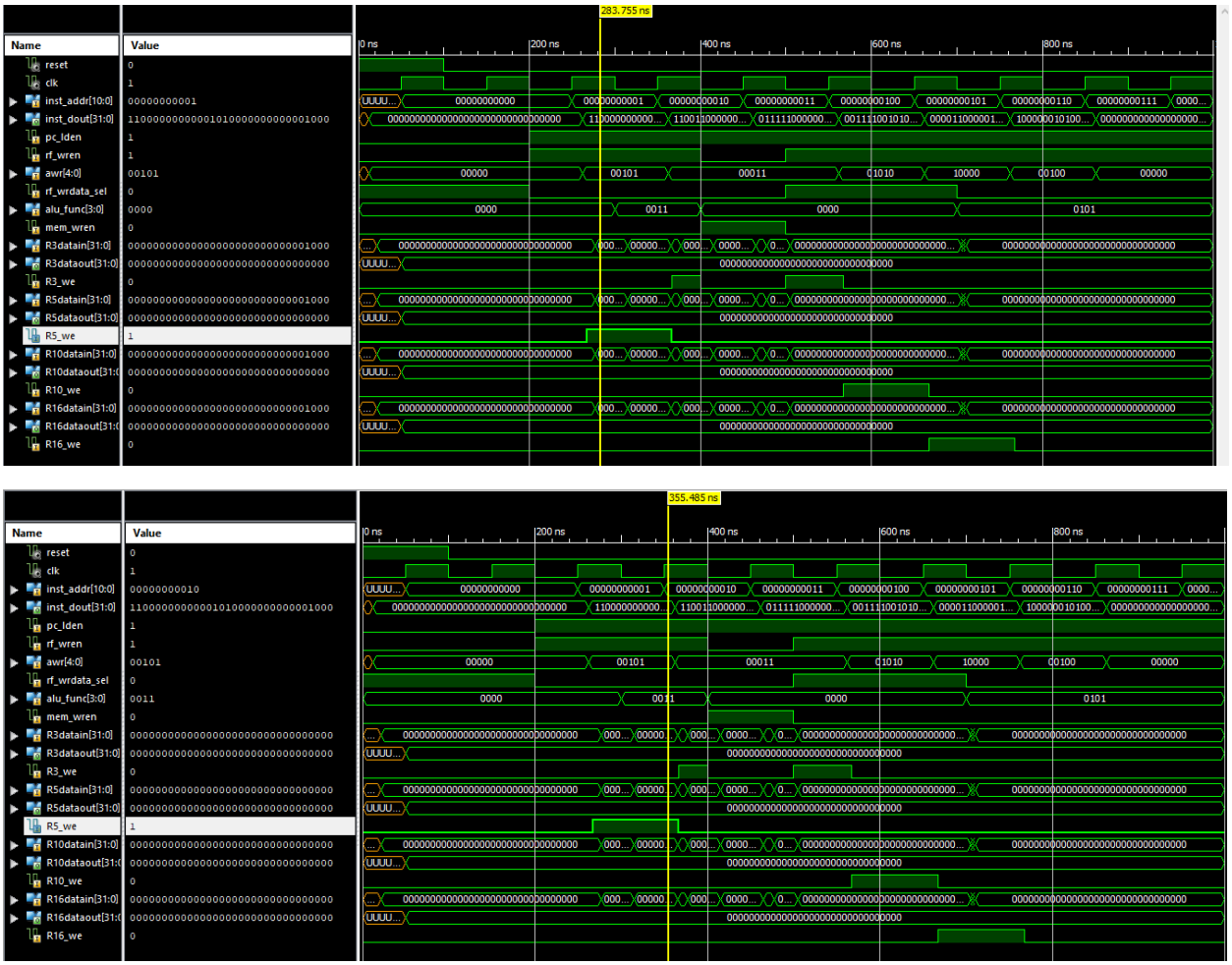
Αρχικά υπήρχε πρόβλημα με την μνήμη Ram η οποία κρατούσε την προηγούμενη εντολή εντός και του αρνητικού κύκλου του ρολογιού με αποτέλεσμα το WE πχ. του καταχωρητή r5 κατά την εντολή addi r5,r0,8 να είναι ενεργό σε όλο τον αρνητικό χρόνο του ρολογιού και ελάχιστα στην αρχή του επόμενου θετικού κύκλου μέχρι να περάσουν τα 16 nano-second που έχει ως καθυστέρηση η μνήμη Ram μέχρι να επιστρέψει την επόμενη εντολή που ζητήθηκε. Αυτό συμβαίνει διότι το WE των καταχωρητών προκύπτει από το 32 bit Instruction.

Η δομή του RF φαίνεται στο ακόλουθο σχήμα:

Inst[20-16] =



Το παραπάνω πρόβλημα φαίνεται και στην παρακάτω εικόνα :



Οπότε αρχικά αλλάχθηκε ο κώδικας της μνήμης Ram ώστε στις αρνητικές ακμές του ρολογιού να μην συνεχίζει να παρέχει την προηγούμενη εντολή στο inst_dout αλλά να το μηδενίζει.

Στην συνέχει με βάση τα testbench φαινόταν εντός του ίδιου κύκλου να πραγματοποιούνταν η πράξη μεταξύ του RF_A και RF_B ή Immediate 32 και στην συνέχεια το ALU_out πήγαινε ως όρισμα στους καταχωρητές και ήταν και ενεργό το WE του σωστού καταχωρητή όμως παρόλα αυτά το data_out αυτού του καταχωρητή παρέμενε μηδέν. Σε όλες τις δοκιμές testbench των Register, RF και Decstage λειτουργούσαν όλα τέλεια, όμως όταν ενώνονταν το ifstage,decstage,exstage και ram δημιουργούνταν αυτό το πρόβλημα. Έπειτα προκειμένου να λυθεί αυτό το πρόβλημα άρχισα να μειώνω τις καθυστερήσεις εντός των modules ώστε η καθυστέρηση που χρειάζεται ο καταχωρητής για να αποθηκεύει την νέα τιμή να γίνεται εντός του θετικού clock του ίδιου κύκλου όμως και πάλι δεν λύθηκε το πρόβλημα. Το παραπάνω φαίνεται και στο εξής waveform :

Πράξη : addi r5,r0,8

Τέλος η μνήμη Ram αλλάχθηκε ξανά ώστε να έχει την αρχική της λειτουργία και αλλάχθηκαν οι Registers ώστε το write τους να πραγματοποιείται εντός του αρνητικού χρόνου του ρολογιού. Με αυτές τις μετατροπές λύθηκε το πρόβλημα και οι εντολές εκτελούνται όλες εντός ενός κύκλου.

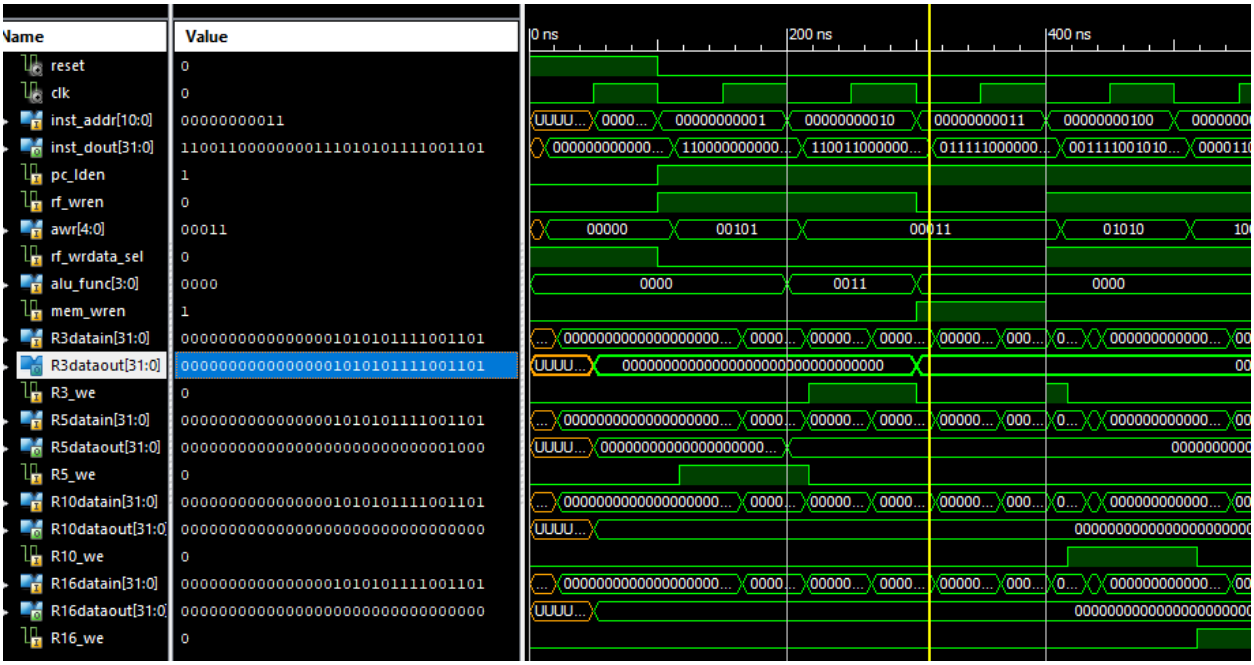
Παρακάτω φαίνονται να λειτουργούν οι εντολές -- addi r5,r0,8 και -- ori r3,r0,ABCD εντός ενός κύκλου σωστά:

-- addi r5,r0,8

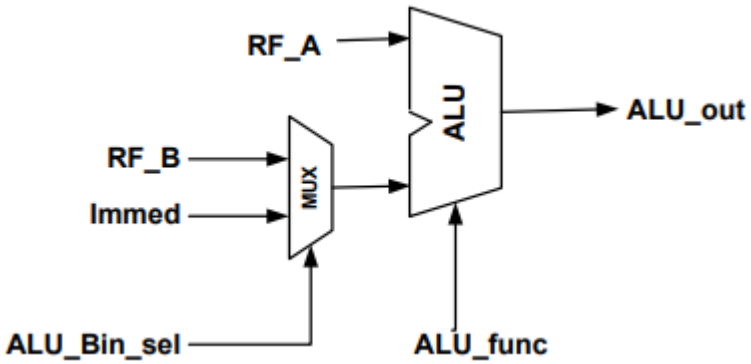

183.544 ns
0 ns
200 ns
Name
Value
reset
clk
inst_addr[10:0]
inst_dout[31:0]
pc_lden
rf_wren
awr[4:0]
rf_wrdata_sel
alu_func[3:0]
mem_wren
R3datain[31:0]
R3dataout[31:0]
R3_we
R5datain[31:0]
R5dataout[31:0]
R5_we
R10datain[31:0]
R10dataout[31:0]
R10_we
R16datain[31:0]
R16dataout[31:0]
R16_we


0 ns
200 ns
Name
Value
reset
clk
inst_addr[10:0]
inst_dout[31:0]
pc_lden
rf_wren
awr[4:0]
rf_wrdata_sel
alu_func[3:0]
mem_wren
R3datain[31:0]
R3dataout[31:0]
R3_we
R5datain[31:0]
R5dataout[31:0]
R5_we
R10datain[31:0]
R10dataout[31:0]
R10_we
R16datain[31:0]
R16dataout[31:0]
R16_we

-- ori r3,r0,ABCD



Το module EXSTAGE έχει υλοποιηθεί ως structural σύμφωνα με το ακόλουθο σχήμα:



Το module MEMSTAGE διατηρεί επίσης structural αρχιτεκτονική περιέχοντας τα εξής modules:

A) 2 Mux_4to1

B) 1 adder

Ο adder χρησιμοποιείται μόνο ώστε να προσθέτει ένα offset 1024 στην write_data_addres η οποία θα προωθηθεί στην μνήμη RAM. Οι δύο πολυπλέκτες χρησιμοποιούνται για να ρυθμίζουν της μεταβλητές MEM_DataOut και MM_WrData κατάλληλα ανάλογα με το αν έχουμε lb ή sb ή lw ή sw. Την πληροφορία για το σε ποια περίπτωση βρισκόμαστε προκείπτει από τον συνδυασμό του Mem_WrEn με το ByteOp τα οποία τοποθετούνται στα Sel των πολυπλεκτών.

Στον φάκελο WAVEFORMS που έπρεπε να βάλουμε τις κυματομορφές προσομοίωσης δεν μπόρεσα να τοποθετήσω τα αρχεία Waveform Configuration File (.wcfg) διότι το Xilinx 13.4 στα windows 10 που έχω δεν με αφήνει να τα αποθηκεύσω. Όταν προσπαθήσω να τα αποθηκεύσω μου κρασάρει και κλείνει αυτόματα και χάνω όλη την δουλειά που έχω φτιάξει με τις μεταβλητές της προσομοίωσης. Έχω αποθηκεύσει screenshots όπως είπατε και στο email σας.